

Pepcy's Templates for OI and ICPC

Pepcy CHEN

updated on October 16, 2019

目录

1 Vim 配置	1
2 数据结构	3
2.1 树状数组	3
2.2 线段树	4
2.3 Splay	6
2.4 Treap(Split & Merge)	10
2.5 替罪羊树	12
2.6 Link-Cut Tree	15
2.7 主席树	17
2.7.1 链上区间 k 小值	17
2.7.2 树上两点间 k 小值	19
2.8 可持久化 Treap	22
2.9 并查集	23
2.10 Sparse Table	24
2.11 K-d Tree	25
2.11.1 曼哈顿距离最近点	25
2.11.2 全局欧几里得距离 k -远点对	27
2.12 单调队列	29
2.13 Sqrt Tree	30
3 树与图	34
3.1 单源最短路	34
3.1.1 Dijkstra	34
3.1.2 队列优化的 Bellman-Ford	35
3.2 两点间 k 短路	37
3.3 斯坦纳树	39
3.4 全局最短路	41
3.4.1 Floyd	41
3.4.2 Floyd 传递闭包	42
3.4.3 Johnson	42
3.5 点分治	45
3.6 树链剖分	47
3.7 DSU on Tree / 树上启发式合并	50
3.8 网络流	52
3.8.1 Dinic	52
3.8.2 ISAP	54

3.8.3 最小割输出方案	55
3.9 Gomory-Hu Tree	56
3.10 费用流	59
3.10.1 Dijkstra 费用流	59
3.10.2 ZKW 费用流	61
3.11 上下界网络流	64
3.11.1 无源无汇上下界可行流	64
3.11.2 有源有汇上下界最大流	66
3.11.3 有源有汇上下界最小流	68
3.12 Tarjan	70
3.12.1 有向图找强连通分量并缩点建图	70
3.12.2 无向图找边双连通分量并缩点建图	72
3.12.3 无向图找点双连通分量并缩点建图	74
3.13 2-SAT	76
3.14 欧拉回路	79
3.14.1 DFS 版本	79
3.14.2 无递归版本 有向图	80
3.14.3 无递归版本 无向图	82
3.15 最大团/最大独立集/弦图最小染色	84
3.16 最小树形图	85
3.16.1 朱刘算法	85
3.16.2 Tarjan 的优化实现	86
3.17 虚树	90
3.18 二分图最大匹配	93
3.19 二分图最大权完备匹配	94
3.20 一般图最大匹配	96
4 字符串	99
4.1 Hash	99
4.2 KMP	99
4.3 扩展 KMP	100
4.4 后缀数组	101
4.5 后缀自动机	102
4.6 AC 自动机	103
4.7 Manacher	105
4.8 回文树	106
5 数学	108
5.1 欧几里得算法	108
5.2 组合数与 Lucas 定理	108
5.3 线性预处理逆元	109

5.4 线性筛	109
5.5 杜教筛	110
5.6 Min25 筛	112
5.7 线性同余方程	114
5.8 BSGS	114
5.9 二次剩余	115
5.10 线性基	116
5.10.1 在线版本	116
5.10.2 离线版本	117
5.11 高斯消元	118
5.12 Berlekamp-Massey	119
5.13 Miller-Rabin	120
5.14 Pollard's Rho	121
5.15 快速傅立叶变换	122
5.15.1 FFT	122
5.15.2 两次 DFT 的多项式乘法	124
5.15.3 拆系数 FFT	126
5.15.4 NTT	128
5.15.5 NTT 模数及原根表	131
5.16 多项式逆元	132
5.17 挑战多项式（多项式逆元、除法、平方根、 \ln 、 \exp 、快速幂）	134
5.18 多项式 GCD	139
5.19 原根	139
5.20 常系数线性齐次递推	140
5.20.1 暴力多项式取模	140
5.20.2 FFT 多项式取模	141
5.21 拉格朗日插值	145
6 计算几何	147
6.1 点相关	147
6.2 旋转卡壳	149
6.3 半平面交	152
6.4 k 次圆覆盖	153
6.5 平面图区域 (Farmland)	156
6.6 Deluanay 三角剖分与平面欧几里得距离最小生成树	157
6.7 三维凸包	162
6.8 动态半凸壳	166
6.9 自适应辛普森积分	168
6.10 由经纬度计算球面最短距离	169

7 博弈	170
7.1 K 倍动态减法	170
7.2 Nim-3	170
8 其他	172
8.1 Java 高精度	172
8.2 三维偏序	173
8.3 $O(1)$ 取模乘	175
8.4 <code>std::bitset</code> 遍历	175
8.5 bitset 优化最长公共子序列	175
8.6 IO 优化	177

电子版：pepcy.cf/icpc-templates/

1 Vim 配置

```
1 syntax on
2 set nocompatible
3 set backspace=2
4 set tabstop=4
5 set softtabstop
6 set shiftwidth=4
7 set expandtab
8 set smarttab
9 set autoindent
10 set cindent
11 set smartindent
12 set number
13 set ruler
14 set cursorline
15 set scrolloff=3
16 set magic
17 set showmatch
18 set showcmd
19 set history=1000
20 set mouse=a
21 set hlsearch
22 set incsearch
23 set ignorecase
24 set smartcase
25
26 map <C-a> gg0vG$
27
28 noremap <C-z> u
29 inoremap <C-z> <C-o>u
30
31 inoremap {<CR> {}<LEFT><CR><CR><UP><TAB>
32
33 map <C-p> i#include <bits/stdc++.h><CR><CR>int main() {<CR><CR>return 0;<UP><ESC>
34
35 map <F5> :w<CR>!g++ % -o %< -lm -O2 -std=c++11; echo Exit code: $?; echo "======"<CR>
36 imap <F5> <ESC>:w<CR>!g++ % -o %< -lm -O2 -std=c++11; echo Exit code: $?; echo "======"<CR>
```

This page is intentionally left blank

2 数据结构

2.1 树状数组

```
1 #include <cstdio>
2 #include <algorithm>
3
4 const int MAXN = 100005;
5
6 struct BIT {
7     int n;
8     long long a[MAXN];
9
10    static constexpr int lowbit(int x) { return x & -x; }
11
12    void init(int n) {
13        this->n = n;
14        std::fill(a + 1, a + n + 1, 0);
15    }
16
17    void update(int pos, int val) {
18        for (int i = pos; i <= n; i += lowbit(i)) a[i] += val;
19    }
20
21    long long query(int pos) {
22        long long res = 0;
23        for (int i = pos; i; i -= lowbit(i)) res += a[i];
24        return res;
25    }
26    long long query(int l, int r) { return query(r) - query(l - 1); }
27 } bit;
28
29 int main() {
30     int n;
31     scanf("%d", &n);
32
33     static int a[MAXN];
34     for (int i = 1; i <= n; i++) scanf("%d", &a[i]);
35
36     bit.init(n);
37     for (int i = 1; i <= n; i++) bit.update(i, a[i]);
38
39     int q;
40     scanf("%d", &q);
41     while (q--) {
42         int op;
43         scanf("%d", &op);
44
45         if (op == 1) {
46             int pos, val;
47             scanf("%d %d", &pos, &val);
48             bit.update(pos, val);
49         } else {
```

```

50     int l, r;
51     scanf("%d %d", &l, &r);
52     printf("%lld\n", bit.query(l, r));
53 }
54 }
55
56 return 0;
57 }
```

2.2 线段树

```

1 #include <cstdio>
2 #include <algorithm>
3
4 const int MAXN = 100005;
5
6 class SegT {
7 private:
8     struct Node {
9         Node *lc, *rc;
10        int l, r;
11        long long sum, tag;
12
13        Node() {}
14        Node(int pos, int val) : lc(NULL), rc(NULL), l(pos), r(pos), sum(val), tag(0) {}
15        Node(Node *lc, Node *rc) : lc(lc), rc(rc), l(lc->l), r(rc->r), tag(0) {
16            maintain();
17        }
18
19        void add(long long d) {
20            sum += (r - l + 1) * d;
21            tag += d;
22        }
23
24        void pushDown() {
25            if (tag) {
26                lc->add(tag);
27                rc->add(tag);
28                tag = 0;
29            }
30        }
31
32        void maintain() {
33            sum = lc->sum + rc->sum;
34        }
35
36        void update(int l, int r, int d) {
37            if (r < this->l || this->r < l) return;
38            if (l <= this->l && this->r <= r) {
39                add(d);
40                return;
41            }
42            pushDown();
43            lc->update(l, r, d);
```

2 数据结构

```
44         rc->update(l, r, d);
45         maintain();
46     }
47     void update(int pos, int d) {
48         if (l == r) {
49             add(d);
50             return;
51         }
52         pushDown();
53         int mid = l + ((r - l) >> 1);
54         (pos <= mid ? lc : rc)->update(pos, d);
55         maintain();
56     }
57
58     long long query(int l, int r) {
59         if (r < this->l || this->r < l) return 0;
60         if (l <= this->l && this->r <= r) return sum;
61         pushDown();
62         return lc->query(l, r) + rc->query(l, r);
63     }
64     long long query(int pos) {
65         if (l == r) return sum;
66         pushDown();
67         int mid = l + ((r - l) >> 1);
68         return (pos <= mid ? lc : rc)->query(pos);
69     }
70 } *root, _pool[MAXN << 1], *_curr;
71
72 public:
73     Node *build(int l, int r, int *a) {
74         if (l == r) return new (_curr++) Node(l, a[l]);
75         int mid = l + ((r - l) >> 1);
76         return new (_curr++) Node(build(l, mid, a), build(mid + 1, r, a));
77     }
78     void build(int n, int *a) {
79         _curr = _pool;
80         root = build(1, n, a);
81     }
82
83     void update(int l, int r, int d) { root->update(l, r, d); }
84     void update(int pos, int d) { root->update(pos, d); }
85     long long query(int l, int r) { return root->query(l, r); }
86     long long query(int pos) { return root->query(pos); }
87 } segT;
88
89 int main() {
90     int n;
91     scanf("%d", &n);
92
93     static int a[MAXN];
94     for (int i = 1; i <= n; i++) scanf("%d", &a[i]);
95
96     segT.build(1, n, a);
97
98     int q;
99     scanf("%d", &q);
```

```

100    while (q--) {
101        int op, l, r;
102        scanf("%"d "%d %d", &op, &l, &r);
103
104        if (op == 1) {
105            int d;
106            scanf("%"d, &d);
107            segT.update(l, r, d);
108        } else {
109            printf("%"lld\n", segT.query(l, r));
110        }
111    }
112
113    return 0;
114 }
```

2.3 Splay

```

1 #include <cstdio>
2 #include <climits>
3 #include <algorithm>
4
5 const int MAXN = 100005;
6
7 template <typename T, size_t SIZE>
8 struct MemoryPool {
9     char mem[SIZE * sizeof (T)], *memTop, *del[SIZE], **delTop;
10
11     MemoryPool() : memTop(mem), delTop(del) {}
12
13     void *alloc() {
14         if (delTop != del) return (void *) --delTop;
15         char *res = memTop;
16         memTop += sizeof (T);
17         return (void *) res;
18     }
19
20     void free(void *p) { *delTop++ = (char *) p; }
21 };
22
23 struct Splay {
24     struct Node {
25         Node *c[2], *fa, *pred, *succ;
26         int val, cnt, size;
27         static MemoryPool<Node, MAXN> pool;
28
29         Node() {}
30         Node(Node *fa, int val) : val(val), size(1), cnt(1), c(), fa(fa), pred(NULL), succ(NULL) {}
31
32         ~Node() {
33             if (c[0]) delete c[0];
34             if (c[1]) delete c[1];
35     }
36 }
```

2 数据结构

```
37     void *operator new(size_t) { return pool.alloc(); }
38     void operator delete(void *p) { pool.free(p); }
39
40     void maintain() {
41         size = (c[0] ? c[0]->size : 0) + cnt + (c[1] ? c[1]->size : 0);
42     }
43
44     int relation() { return fa->c[1] == this; }
45 } *root;
46
47 Splay() : root(NULL) {}
48
49 static int size(const Node *u) { return u ? u->size : 0; }
50
51 void init() {
52     root = new Node(NULL, INT_MIN);
53     root->c[1] = new Node(root, INT_MAX);
54     root->c[1]->pred = root;
55     root->succ = root->c[1];
56     root->maintain();
57 }
58
59 void rotate(Node *u) {
60     Node *o = u->fa;
61     int x = u->relation();
62
63     u->fa = o->fa;
64     if (u->fa) u->fa->c[o->relation()] = u;
65
66     o->c[x] = u->c[x ^ 1];
67     if (u->c[x ^ 1]) u->c[x ^ 1]->fa = o;
68
69     u->c[x ^ 1] = o;
70     o->fa = u;
71
72     o->maintain();
73     u->maintain();
74
75     if (!u->fa) root = u;
76 }
77
78 Node *splay(Node *u, Node *targetFa = NULL) {
79     while (u->fa != targetFa) {
80         if (u->fa->fa == targetFa) rotate(u);
81         else if (u->relation() == u->fa->relation()) rotate(u->fa), rotate(u);
82         else rotate(u), rotate(u);
83     }
84     return u;
85 }
86
87 Node *insert(int val) {
88     Node **u = &root, *fa = NULL;
89
90     while (*u && (*u)->val != val) {
91         fa = *u;
92         fa->size++;
```

```

93         u = &(*u)->c[val > (*u)->val];
94     }
95
96     if (*u) {
97         ++(*u)->cnt;
98         ++(*u)->size;
99     } else {
100        (*u) = new Node(fa, val);
101
102        if (fa) {
103            if ((*u)->relation()) {
104                (*u)->succ = fa->succ;
105                (*u)->pred = fa;
106                if (fa->succ) fa->succ->pred = *u;
107                fa->succ = *u;
108            } else {
109                (*u)->pred = fa->pred;
110                (*u)->succ = fa;
111                if (fa->pred) fa->pred->succ = *u;
112                fa->pred = *u;
113            }
114        }
115    }
116
117    return splay(*u);
118 }
119
120 void erase(Node *u) {
121     splay(u->pred);
122     splay(u->succ, u->pred);
123
124     if (u->cnt > 1) {
125         --u->cnt;
126         --u->size;
127     } else {
128         delete u->succ->c[0];
129         u->succ->c[0] = NULL;
130         u->pred->succ = u->succ;
131         u->succ->pred = u->pred;
132     }
133
134     --u->pred->size;
135     --u->succ->size;
136 }
137 void erase(int val) {
138     Node *u = find(val);
139     if (u) erase(u);
140 }
141
142 Node *find(int val) {
143     Node *u = root;
144     while (u && u->val != val) u = u->c[val > u->val];
145     if (u) return splay(u);
146     else return NULL;
147 }
148

```

2 数据结构

```
149     Node *select(int k) {
150         Node *u = root;
151         while (k < size(u->c[0]) || k >= size(u->c[0]) + u->cnt) {
152             if (k < size(u->c[0])) u = u->c[0];
153             else {
154                 k -= size(u->c[0]) + u->cnt;
155                 u = u->c[1];
156             }
157         }
158         return splay(u);
159     }
160
161     int rank(int val) {
162         Node *u = find(val);
163         if (!u) {
164             u = insert(val);
165             int res = size(u->c[0]);
166             erase(u);
167             return res;
168         }
169         return size(u->c[0]);
170     }
171
172     int pred(int val) {
173         Node *u = find(val);
174         if (!u) {
175             u = insert(val);
176             int res = u->pred->val;
177             erase(u);
178             return res;
179         }
180         return u->pred->val;
181     }
182
183     int succ(int val) {
184         Node *u = find(val);
185         if (!u) {
186             u = insert(val);
187             int res = u->succ->val;
188             erase(u);
189             return res;
190         }
191         return u->succ->val;
192     }
193 } splay;
194
195 MemoryPool<Splay::Node, MAXN> Splay::Node::pool;
196
197 int main() {
198     splay.init();
199
200     int n;
201     scanf("%d", &n);
202
203     while (n--) {
204         int op, x;
```

```

205     scanf("%d %d", &op, &x);
206
207     if (op == 1) splay.insert(x);
208     if (op == 2) splay.erase(x);
209     if (op == 3) printf("%d\n", splay.rank(x));
210     if (op == 4) printf("%d\n", splay.select(x)->val);
211     if (op == 5) printf("%d\n", splay.pred(x));
212     if (op == 6) printf("%d\n", splay.succ(x));
213 }
214
215 return 0;
216 }
```

2.4 Treap(Split & Merge)

```

1 #include <cstdio>
2 #include <cstdlib>
3 #include <algorithm>
4
5 const int MAXN = 100005;
6
7 struct Treap {
8     struct Node {
9         Node *lc, *rc;
10        int val, size;
11        bool rev;
12
13        Node() {}
14        Node(int val, Node *lc = NULL, Node *rc = NULL) : val(val), lc(lc), rc(rc), rev(false) {
15            maintain();
16        }
17
18        void reverse() {
19            std::swap(lc, rc);
20            rev ^= 1;
21        }
22
23        void pushDown() {
24            if (rev) {
25                if (lc) lc->reverse();
26                if (rc) rc->reverse();
27                rev = false;
28            }
29        }
30
31        void maintain() {
32            size = (lc ? lc->size : 0) + 1 + (rc ? rc->size : 0);
33        }
34
35        void print() {
36            pushDown();
37            if (lc) lc->print();
38            printf("%d ", val);
39            if (rc) rc->print();
```

2 数据结构

```
40         }
41     } *_root, *_pool[MAXN], *_curr;
42
43     Treap() : root(NULL), _curr(_pool) {}
44
45     static int size(const Node *u) { return u ? u->size : 0; }
46
47     Node *_build(int l, int r) {
48         if (l > r) return NULL;
49         int mid = l + ((r - l) >> 1);
50         return new (_curr++) Node(mid, _build(l, mid - 1), _build(mid + 1, r));
51     }
52     void build(int l, int r) { root = _build(l, r); }
53
54     Node *merge(Node *a, Node *b) {
55         if (!a) return b;
56         if (!b) return a;
57
58         if (rand() % (a->size + b->size) < a->size) {
59             a->pushDown();
60             a->rc = merge(a->rc, b);
61             a->maintain();
62             return a;
63         } else {
64             b->pushDown();
65             b->lc = merge(a, b->lc);
66             b->maintain();
67             return b;
68         }
69     }
70
71     std::pair<Node *, Node *> split(Node *u, int pos) {
72         std::pair<Node *, Node *> res(NULL, NULL);
73         if (!u) return res;
74         u->pushDown();
75         if (pos <= size(u->lc)) {
76             res = split(u->lc, pos);
77             u->lc = res.second;
78             u->maintain();
79             res.second = u;
80         } else {
81             res = split(u->rc, pos - size(u->lc) - 1);
82             u->rc = res.first;
83             u->maintain();
84             res.first = u;
85         }
86         return res;
87     }
88
89     void reverse(int l, int r) {
90         std::pair<Node *, Node *> L = split(root, l - 1);
91         std::pair<Node *, Node *> R = split(L.second, r - l + 1);
92         R.first->reverse();
93         root = merge(merge(L.first, R.first), R.second);
94     }
95 }
```

```

96     void print() {
97         root->print();
98         puts("");
99     }
100 } treap;
101
102 int main() {
103     int n, m;
104     scanf("%d %d", &n, &m);
105
106     treap.build(1, n);
107
108     for (int i = 0, l, r; i < m; i++) {
109         scanf("%d %d", &l, &r);
110         treap.reverse(l, r);
111     }
112
113     treap.print();
114
115     return 0;
116 }
```

2.5 替罪羊树

```

1 #include <cstdio>
2 #include <vector>
3 #include <algorithm>
4
5 const int MAXN = 100005;
6
7 template <typename T, size_t SIZE>
8 struct MemoryPool {
9     char mem[SIZE * sizeof (T)], *top, *del[SIZE], **delTop;
10
11     MemoryPool() { init(); }
12
13     void init() {
14         top = mem;
15         delTop = del;
16     }
17
18     void *alloc() {
19         if (delTop != del) return *(--delTop);
20         char *res = top;
21         top += sizeof (T);
22         return (void *) res;
23     }
24
25     void free(void *p) { *(delTop++) = (char *) p; }
26 };
27
28 struct ScapegoatTree {
29     static constexpr double ALPHA = 0.7;
```

```
31     struct Node {
32         Node *c[2];
33         int val, size, valid;
34         bool isDeleted;
35
36         static MemoryPool<Node, MAXN> pool;
37
38         Node() {}
39         Node(int val) : val(val), size(1), valid(1), isDeleted(false) {
40             c[0] = c[1] = NULL;
41         }
42
43         void *operator new(size_t) { return pool.alloc(); }
44         void operator delete(void *p) { pool.free(p); }
45
46         bool bad() {
47             return (c[0] ? c[0]->size : 0) > ALPHA * size
48             || (c[1] ? c[1]->size : 0) > ALPHA * size;
49         }
50
51         void maintain() {
52             size = 1 + (c[0] ? c[0]->size : 0) + (c[1] ? c[1]->size : 0);
53             valid = !isDeleted + (c[0] ? c[0]->valid : 0) + (c[1] ? c[1]->valid : 0);
54         }
55     } *root;
56
57     void dfs(Node *u, std::vector<Node *> &vec) {
58         if (!u) return;
59         dfs(u->c[0], vec);
60         if (!u->isDeleted) vec.push_back(u);
61         dfs(u->c[1], vec);
62         if (u->isDeleted) delete u;
63     }
64
65     Node *build(std::vector<Node *> &vec, int l, int r) {
66         if (l >= r) return NULL;
67         int mid = l + ((r - l) >> 1);
68         Node *o = vec[mid];
69         o->c[0] = build(vec, l, mid);
70         o->c[1] = build(vec, mid + 1, r);
71         o->maintain();
72         return o;
73     }
74
75     void rebuild(Node *&u) {
76         static std::vector<Node *> vec;
77         vec.clear();
78         dfs(u, vec);
79         u = build(vec, 0, vec.size());
80     }
81
82     void insert(int x, Node *&u) {
83         if (!u) {
84             u = new Node(x);
85             return;
86         }
```

```

87
88     ++u->size;
89     ++u->valid;
90     if (x >= u->val) insert(x, u->c[1]);
91     else insert(x, u->c[0]);
92     if (u->bad()) rebuild(u);
93 }
94 void insert(int x) {
95     insert(x, root);
96 }
97
98 void del(int rank, Node *u) {
99     if (!u) return;
100    if (!u->isDeled && rank == (u->c[0] ? u->c[0]->valid : 0) + 1) {
101        u->isDeled = true;
102        --u->valid;
103        return;
104    }
105    --u->valid;
106    if (rank <= (u->c[0] ? u->c[0]->valid : 0)) del(rank, u->c[0]);
107    else del(rank - (u->c[0] ? u->c[0]->valid : 0) - !u->isDeled, u->c[1]);
108 }
109 void del(int rank) {
110     del(rank, root);
111 }
112
113 int getRank(int x) {
114     int res = 1;
115     Node *u = root;
116     while (u) {
117         if (u->val >= x) u = u->c[0];
118         else {
119             res += (u->c[0] ? u->c[0]->valid : 0) + !u->isDeled;
120             u = u->c[1];
121         }
122     }
123     return res;
124 }
125
126 int findKth(int k) {
127     Node *u = root;
128     while (u) {
129         if (!u->isDeled && (u->c[0] ? u->c[0]->valid : 0) + 1 == k) return u->val;
130         if ((u->c[0] ? u->c[0]->valid : 0) >= k) u = u->c[0];
131         else {
132             k -= (u->c[0] ? u->c[0]->valid : 0) + !u->isDeled;
133             u = u->c[1];
134         }
135     }
136 }
137
138 int pred(int x) {
139     return findKth(getRank(x) - 1);
140 }
141
142 int succ(int x) {

```

```
143         return findKth(getRank(x + 1));
144     }
145 } scapeT;
146 MemoryPool<ScapegoatTree::Node, MAXN> ScapegoatTree::Node::pool;
147
148 int main() {
149     int n;
150     scanf("%d", &n);
151     while (n--) {
152         int op, x;
153         scanf("%d %d", &op, &x);
154
155         if (op == 1) scapeT.insert(x);
156         if (op == 2) scapeT.del(scapeT.getRank(x));
157         if (op == 3) printf("%d\n", scapeT.getRank(x));
158         if (op == 4) printf("%d\n", scapeT.findKth(x));
159         if (op == 5) printf("%d\n", scapeT.pred(x));
160         if (op == 6) printf("%d\n", scapeT.succ(x));
161     }
162
163     return 0;
164 }
```

2.6 Link-Cut Tree

```
1 #include <cstdlib>
2 #include <algorithm>
3
4 const int MAXN = 30005;
5
6 struct LinkCutTree {
7     struct Node {
8         Node *c[2], *fa, *pathFa;
9         int val, max;
10        bool rev;
11
12        Node() {}
13        Node(int val) : val(val), max(val), rev(false), fa(NULL), pathFa(NULL), c() {}
14
15        int relation() { return fa->c[1] == this; }
16
17        void maintain() {
18            max = val;
19            if (c[0]) max = std::max(max, c[0]->max);
20            if (c[1]) max = std::max(max, c[1]->max);
21        }
22
23        void pushDown() {
24            if (rev) {
25                std::swap(c[0], c[1]);
26                if (c[0]) c[0]->rev ^= 1;
27                if (c[1]) c[1]->rev ^= 1;
28                rev = false;
29            }
30        }
31    };
32
33    Node *root;
34
35    LinkCutTree() : root(nullptr) {}
36
37    void insert(int val) {
38        Node *n = new Node(val);
39        n->fa = n;
40        n->pathFa = n;
41        n->c[0] = n;
42        n->c[1] = n;
43        n->max = val;
44        n->val = val;
45        n->rev = false;
46
47        if (root) {
48            Node *cur = root;
49            while (true) {
50                if (val < cur->val) {
51                    if (cur->c[0] == cur) {
52                        cur->c[0] = n;
53                        n->fa = cur;
54                        break;
55                    }
56                    cur = cur->c[0];
57                } else {
58                    if (cur->c[1] == cur) {
59                        cur->c[1] = n;
60                        n->fa = cur;
61                        break;
62                    }
63                    cur = cur->c[1];
64                }
65            }
66        } else {
67            root = n;
68        }
69    }
70
71    void update(int l, int r, int val) {
72        Node *lNode = find(l);
73        Node *rNode = find(r);
74
75        if (!lNode || !rNode) return;
76
77        lNode->val = val;
78        rNode->val = val;
79
80        if (lNode->relation()) {
81            lNode->maintain();
82        }
83        if (rNode->relation()) {
84            rNode->maintain();
85        }
86
87        if (lNode->pathFa != rNode) {
88            Node *lPathFa = lNode->pathFa;
89            Node *rPathFa = rNode->pathFa;
90
91            if (lPathFa->relation()) {
92                lPathFa->maintain();
93            }
94            if (rPathFa->relation()) {
95                rPathFa->maintain();
96            }
97
98            lPathFa->c[1] = rNode;
99            rNode->fa = lPathFa;
100           rNode->pathFa = lPathFa;
101           lPathFa->pathFa = rNode;
102
103           if (lPathFa->relation()) {
104               lPathFa->maintain();
105           }
106
107           if (rNode->relation()) {
108               rNode->maintain();
109           }
110
111           lNode->pathFa = rNode;
112           rNode->fa = lNode;
113           rNode->pathFa = lNode;
114           lNode->pathFa = rNode;
115
116           if (lNode->relation()) {
117               lNode->maintain();
118           }
119
120           if (rNode->relation()) {
121               rNode->maintain();
122           }
123
124           lNode->pathFa = rNode;
125           rNode->fa = lNode;
126           rNode->pathFa = lNode;
127           lNode->pathFa = rNode;
128
129           if (lNode->relation()) {
130               lNode->maintain();
131           }
132
133           if (rNode->relation()) {
134               rNode->maintain();
135           }
136
137           lNode->pathFa = rNode;
138           rNode->fa = lNode;
139           rNode->pathFa = lNode;
140           lNode->pathFa = rNode;
141
142           if (lNode->relation()) {
143               lNode->maintain();
144           }
145
146           if (rNode->relation()) {
147               rNode->maintain();
148           }
149
150           lNode->pathFa = rNode;
151           rNode->fa = lNode;
152           rNode->pathFa = lNode;
153           lNode->pathFa = rNode;
154
155           if (lNode->relation()) {
156               lNode->maintain();
157           }
158
159           if (rNode->relation()) {
160               rNode->maintain();
161           }
162
163           lNode->pathFa = rNode;
164           rNode->fa = lNode;
165           rNode->pathFa = lNode;
166           lNode->pathFa = rNode;
167
168           if (lNode->relation()) {
169               lNode->maintain();
170           }
171
172           if (rNode->relation()) {
173               rNode->maintain();
174           }
175
176           lNode->pathFa = rNode;
177           rNode->fa = lNode;
178           rNode->pathFa = lNode;
179           lNode->pathFa = rNode;
180
181           if (lNode->relation()) {
182               lNode->maintain();
183           }
184
185           if (rNode->relation()) {
186               rNode->maintain();
187           }
188
189           lNode->pathFa = rNode;
190           rNode->fa = lNode;
191           rNode->pathFa = lNode;
192           lNode->pathFa = rNode;
193
194           if (lNode->relation()) {
195               lNode->maintain();
196           }
197
198           if (rNode->relation()) {
199               rNode->maintain();
200           }
201
202           lNode->pathFa = rNode;
203           rNode->fa = lNode;
204           rNode->pathFa = lNode;
205           lNode->pathFa = rNode;
206
207           if (lNode->relation()) {
208               lNode->maintain();
209           }
210
211           if (rNode->relation()) {
212               rNode->maintain();
213           }
214
215           lNode->pathFa = rNode;
216           rNode->fa = lNode;
217           rNode->pathFa = lNode;
218           lNode->pathFa = rNode;
219
220           if (lNode->relation()) {
221               lNode->maintain();
222           }
223
224           if (rNode->relation()) {
225               rNode->maintain();
226           }
227
228           lNode->pathFa = rNode;
229           rNode->fa = lNode;
230           rNode->pathFa = lNode;
231           lNode->pathFa = rNode;
232
233           if (lNode->relation()) {
234               lNode->maintain();
235           }
236
237           if (rNode->relation()) {
238               rNode->maintain();
239           }
240
241           lNode->pathFa = rNode;
242           rNode->fa = lNode;
243           rNode->pathFa = lNode;
244           lNode->pathFa = rNode;
245
246           if (lNode->relation()) {
247               lNode->maintain();
248           }
249
250           if (rNode->relation()) {
251               rNode->maintain();
252           }
253
254           lNode->pathFa = rNode;
255           rNode->fa = lNode;
256           rNode->pathFa = lNode;
257           lNode->pathFa = rNode;
258
259           if (lNode->relation()) {
260               lNode->maintain();
261           }
262
263           if (rNode->relation()) {
264               rNode->maintain();
265           }
266
267           lNode->pathFa = rNode;
268           rNode->fa = lNode;
269           rNode->pathFa = lNode;
270           lNode->pathFa = rNode;
271
272           if (lNode->relation()) {
273               lNode->maintain();
274           }
275
276           if (rNode->relation()) {
277               rNode->maintain();
278           }
279
280           lNode->pathFa = rNode;
281           rNode->fa = lNode;
282           rNode->pathFa = lNode;
283           lNode->pathFa = rNode;
284
285           if (lNode->relation()) {
286               lNode->maintain();
287           }
288
289           if (rNode->relation()) {
290               rNode->maintain();
291           }
292
293           lNode->pathFa = rNode;
294           rNode->fa = lNode;
295           rNode->pathFa = lNode;
296           lNode->pathFa = rNode;
297
298           if (lNode->relation()) {
299               lNode->maintain();
300           }
301
302           if (rNode->relation()) {
303               rNode->maintain();
304           }
305
306           lNode->pathFa = rNode;
307           rNode->fa = lNode;
308           rNode->pathFa = lNode;
309           lNode->pathFa = rNode;
310
311           if (lNode->relation()) {
312               lNode->maintain();
313           }
314
315           if (rNode->relation()) {
316               rNode->maintain();
317           }
318
319           lNode->pathFa = rNode;
320           rNode->fa = lNode;
321           rNode->pathFa = lNode;
322           lNode->pathFa = rNode;
323
324           if (lNode->relation()) {
325               lNode->maintain();
326           }
327
328           if (rNode->relation()) {
329               rNode->maintain();
330           }
331
332           lNode->pathFa = rNode;
333           rNode->fa = lNode;
334           rNode->pathFa = lNode;
335           lNode->pathFa = rNode;
336
337           if (lNode->relation()) {
338               lNode->maintain();
339           }
340
341           if (rNode->relation()) {
342               rNode->maintain();
343           }
344
345           lNode->pathFa = rNode;
346           rNode->fa = lNode;
347           rNode->pathFa = lNode;
348           lNode->pathFa = rNode;
349
350           if (lNode->relation()) {
351               lNode->maintain();
352           }
353
354           if (rNode->relation()) {
355               rNode->maintain();
356           }
357
358           lNode->pathFa = rNode;
359           rNode->fa = lNode;
360           rNode->pathFa = lNode;
361           lNode->pathFa = rNode;
362
363           if (lNode->relation()) {
364               lNode->maintain();
365           }
366
367           if (rNode->relation()) {
368               rNode->maintain();
369           }
370
371           lNode->pathFa = rNode;
372           rNode->fa = lNode;
373           rNode->pathFa = lNode;
374           lNode->pathFa = rNode;
375
376           if (lNode->relation()) {
377               lNode->maintain();
378           }
379
380           if (rNode->relation()) {
381               rNode->maintain();
382           }
383
384           lNode->pathFa = rNode;
385           rNode->fa = lNode;
386           rNode->pathFa = lNode;
387           lNode->pathFa = rNode;
388
389           if (lNode->relation()) {
390               lNode->maintain();
391           }
392
393           if (rNode->relation()) {
394               rNode->maintain();
395           }
396
397           lNode->pathFa = rNode;
398           rNode->fa = lNode;
399           rNode->pathFa = lNode;
400           lNode->pathFa = rNode;
401
402           if (lNode->relation()) {
403               lNode->maintain();
404           }
405
406           if (rNode->relation()) {
407               rNode->maintain();
408           }
409
410           lNode->pathFa = rNode;
411           rNode->fa = lNode;
412           rNode->pathFa = lNode;
413           lNode->pathFa = rNode;
414
415           if (lNode->relation()) {
416               lNode->maintain();
417           }
418
419           if (rNode->relation()) {
420               rNode->maintain();
421           }
422
423           lNode->pathFa = rNode;
424           rNode->fa = lNode;
425           rNode->pathFa = lNode;
426           lNode->pathFa = rNode;
427
428           if (lNode->relation()) {
429               lNode->maintain();
430           }
431
432           if (rNode->relation()) {
433               rNode->maintain();
434           }
435
436           lNode->pathFa = rNode;
437           rNode->fa = lNode;
438           rNode->pathFa = lNode;
439           lNode->pathFa = rNode;
440
441           if (lNode->relation()) {
442               lNode->maintain();
443           }
444
445           if (rNode->relation()) {
446               rNode->maintain();
447           }
448
449           lNode->pathFa = rNode;
450           rNode->fa = lNode;
451           rNode->pathFa = lNode;
452           lNode->pathFa = rNode;
453
454           if (lNode->relation()) {
455               lNode->maintain();
456           }
457
458           if (rNode->relation()) {
459               rNode->maintain();
460           }
461
462           lNode->pathFa = rNode;
463           rNode->fa = lNode;
464           rNode->pathFa = lNode;
465           lNode->pathFa = rNode;
466
467           if (lNode->relation()) {
468               lNode->maintain();
469           }
470
471           if (rNode->relation()) {
472               rNode->maintain();
473           }
474
475           lNode->pathFa = rNode;
476           rNode->fa = lNode;
477           rNode->pathFa = lNode;
478           lNode->pathFa = rNode;
479
480           if (lNode->relation()) {
481               lNode->maintain();
482           }
483
484           if (rNode->relation()) {
485               rNode->maintain();
486           }
487
488           lNode->pathFa = rNode;
489           rNode->fa = lNode;
490           rNode->pathFa = lNode;
491           lNode->pathFa = rNode;
492
493           if (lNode->relation()) {
494               lNode->maintain();
495           }
496
497           if (rNode->relation()) {
498               rNode->maintain();
499           }
500
501           lNode->pathFa = rNode;
502           rNode->fa = lNode;
503           rNode->pathFa = lNode;
504           lNode->pathFa = rNode;
505
506           if (lNode->relation()) {
507               lNode->maintain();
508           }
509
510           if (rNode->relation()) {
511               rNode->maintain();
512           }
513
514           lNode->pathFa = rNode;
515           rNode->fa = lNode;
516           rNode->pathFa = lNode;
517           lNode->pathFa = rNode;
518
519           if (lNode->relation()) {
520               lNode->maintain();
521           }
522
523           if (rNode->relation()) {
524               rNode->maintain();
525           }
526
527           lNode->pathFa = rNode;
528           rNode->fa = lNode;
529           rNode->pathFa = lNode;
530           lNode->pathFa = rNode;
531
532           if (lNode->relation()) {
533               lNode->maintain();
534           }
535
536           if (rNode->relation()) {
537               rNode->maintain();
538           }
539
540           lNode->pathFa = rNode;
541           rNode->fa = lNode;
542           rNode->pathFa = lNode;
543           lNode->pathFa = rNode;
544
545           if (lNode->relation()) {
546               lNode->maintain();
547           }
548
549           if (rNode->relation()) {
550               rNode->maintain();
551           }
552
553           lNode->pathFa = rNode;
554           rNode->fa = lNode;
555           rNode->pathFa = lNode;
556           lNode->pathFa = rNode;
557
558           if (lNode->relation()) {
559               lNode->maintain();
560           }
561
562           if (rNode->relation()) {
563               rNode->maintain();
564           }
565
566           lNode->pathFa = rNode;
567           rNode->fa = lNode;
568           rNode->pathFa = lNode;
569           lNode->pathFa = rNode;
570
571           if (lNode->relation()) {
572               lNode->maintain();
573           }
574
575           if (rNode->relation()) {
576               rNode->maintain();
577           }
578
579           lNode->pathFa = rNode;
580           rNode->fa = lNode;
581           rNode->pathFa = lNode;
582           lNode->pathFa = rNode;
583
584           if (lNode->relation()) {
585               lNode->maintain();
586           }
587
588           if (rNode->relation()) {
589               rNode->maintain();
590           }
591
592           lNode->pathFa = rNode;
593           rNode->fa = lNode;
594           rNode->pathFa = lNode;
595           lNode->pathFa = rNode;
596
597           if (lNode->relation()) {
598               lNode->maintain();
599           }
600
601           if (rNode->relation()) {
602               rNode->maintain();
603           }
604
605           lNode->pathFa = rNode;
606           rNode->fa = lNode;
607           rNode->pathFa = lNode;
608           lNode->pathFa = rNode;
609
610           if (lNode->relation()) {
611               lNode->maintain();
612           }
613
614           if (rNode->relation()) {
615               rNode->maintain();
616           }
617
618           lNode->pathFa = rNode;
619           rNode->fa = lNode;
620           rNode->pathFa = lNode;
621           lNode->pathFa = rNode;
622
623           if (lNode->relation()) {
624               lNode->maintain();
625           }
626
627           if (rNode->relation()) {
628               rNode->maintain();
629           }
630
631           lNode->pathFa = rNode;
632           rNode->fa = lNode;
633           rNode->pathFa = lNode;
634           lNode->pathFa = rNode;
635
636           if (lNode->relation()) {
637               lNode->maintain();
638           }
639
640           if (rNode->relation()) {
641               rNode->maintain();
642           }
643
644           lNode->pathFa = rNode;
645           rNode->fa = lNode;
646           rNode->pathFa = lNode;
647           lNode->pathFa = rNode;
648
649           if (lNode->relation()) {
650               lNode->maintain();
651           }
652
653           if (rNode->relation()) {
654               rNode->maintain();
655           }
656
657           lNode->pathFa = rNode;
658           rNode->fa = lNode;
659           rNode->pathFa = lNode;
660           lNode->pathFa = rNode;
661
662           if (lNode->relation()) {
663               lNode->maintain();
664           }
665
666           if (rNode->relation()) {
667               rNode->maintain();
668           }
669
670           lNode->pathFa = rNode;
671           rNode->fa = lNode;
672           rNode->pathFa = lNode;
673           lNode->pathFa = rNode;
674
675           if (lNode->relation()) {
676               lNode->maintain();
677           }
678
679           if (rNode->relation()) {
680               rNode->maintain();
681           }
682
683           lNode->pathFa = rNode;
684           rNode->fa = lNode;
685           rNode->pathFa = lNode;
686           lNode->pathFa = rNode;
687
688           if (lNode->relation()) {
689               lNode->maintain();
690           }
691
692           if (rNode->relation()) {
693               rNode->maintain();
694           }
695
696           lNode->pathFa = rNode;
697           rNode->fa = lNode;
698           rNode->pathFa = lNode;
699           lNode->pathFa = rNode;
700
701           if (lNode->relation()) {
702               lNode->maintain();
703           }
704
705           if (rNode->relation()) {
706               rNode->maintain();
707           }
708
709           lNode->pathFa = rNode;
710           rNode->fa = lNode;
711           rNode->pathFa = lNode;
712           lNode->pathFa = rNode;
713
714           if (lNode->relation()) {
715               lNode->maintain();
716           }
717
718           if (rNode->relation()) {
719               rNode->maintain();
720           }
721
722           lNode->pathFa = rNode;
723           rNode->fa = lNode;
724           rNode->pathFa = lNode;
725           lNode->pathFa = rNode;
726
727           if (lNode->relation()) {
728               lNode->maintain();
729           }
730
731           if (rNode->relation()) {
732               rNode->maintain();
733           }
734
735           lNode->pathFa = rNode;
736           rNode->fa = lNode;
737           rNode->pathFa = lNode;
738           lNode->pathFa = rNode;
739
740           if (lNode->relation()) {
741               lNode->maintain();
742           }
743
744           if (rNode->relation()) {
745               rNode->maintain();
746           }
747
748           lNode->pathFa = rNode;
749           rNode->fa = lNode;
750           rNode->pathFa = lNode;
751           lNode->pathFa = rNode;
752
753           if (lNode->relation()) {
754               lNode->maintain();
755           }
756
757           if (rNode->relation()) {
758               rNode->maintain();
759           }
760
761           lNode->pathFa = rNode;
762           rNode->fa = lNode;
763           rNode->pathFa = lNode;
764           lNode->pathFa = rNode;
765
766           if (lNode->relation()) {
767               lNode->maintain();
768           }
769
770           if (rNode->relation()) {
771               rNode->maintain();
772           }
773
774           lNode->pathFa = rNode;
775           rNode->fa = lNode;
776           rNode->pathFa = lNode;
777           lNode->pathFa = rNode;
778
779           if (lNode->relation()) {
780               lNode->maintain();
781           }
782
783           if (rNode->relation()) {
784               rNode->maintain();
785           }
786
787           lNode->pathFa = rNode;
788           rNode->fa = lNode;
789           rNode->pathFa = lNode;
790           lNode->pathFa = rNode;
791
792           if (lNode->relation()) {
793               lNode->maintain();
794           }
795
796           if (rNode->relation()) {
797               rNode->maintain();
798           }
799
800           lNode->pathFa = rNode;
801           rNode->fa = lNode;
802           rNode->pathFa = lNode;
803           lNode->pathFa = rNode;
804
805           if (lNode->relation()) {
806               lNode->maintain();
807           }
808
809           if (rNode->relation()) {
810               rNode->maintain();
811           }
812
813           lNode->pathFa = rNode;
814           rNode->fa = lNode;
815           rNode->pathFa = lNode;
816           lNode->pathFa = rNode;
817
818           if (lNode->relation()) {
819               lNode->maintain();
820           }
821
822           if (rNode->relation()) {
823               rNode->maintain();
824           }
825
826           lNode->pathFa = rNode;
827           rNode->fa = lNode;
828           rNode->pathFa = lNode;
829           lNode->pathFa = rNode;
830
831           if (lNode->relation()) {
832               lNode->maintain();
833           }
834
835           if (rNode->relation()) {
836               rNode->maintain();
837           }
838
839           lNode->pathFa = rNode;
840           rNode->fa = lNode;
841           rNode->pathFa = lNode;
842           lNode->pathFa = rNode;
843
844           if (lNode->relation()) {
845               lNode->maintain();
846           }
847
848           if (rNode->relation()) {
849               rNode->maintain();
850           }
851
852           lNode->pathFa = rNode;
853           rNode->fa = lNode;
854           rNode->pathFa = lNode;
855           lNode->pathFa = rNode;
856
857           if (lNode->relation()) {
858               lNode->maintain();
859           }
860
861           if (rNode->relation()) {
862               rNode->maintain();
863           }
864
865           lNode->pathFa = rNode;
866           rNode->fa = lNode;
867           rNode->pathFa = lNode;
868           lNode->pathFa = rNode;
869
870           if (lNode->relation()) {
871               lNode->maintain();
872           }
873
874           if (rNode->relation()) {
875               rNode->maintain();
876           }
877
878           lNode->pathFa = rNode;
879           rNode->fa = lNode;
880           rNode->pathFa = lNode;
881           lNode->pathFa = rNode;
882
883           if (lNode->relation()) {
884               lNode->maintain();
885           }
886
887           if (rNode->relation()) {
888               rNode->maintain();
889           }
890
891           lNode->pathFa = rNode;
892           rNode->fa = lNode;
893           rNode->pathFa = lNode;
894           lNode->pathFa = rNode;
895
896           if (lNode->relation()) {
897               lNode->maintain();
898           }
899
900           if (rNode->relation()) {
901               rNode->maintain();
902           }
903
904           lNode->pathFa = rNode;
905           rNode->fa = lNode;
906           rNode->pathFa = lNode;
907           lNode->pathFa = rNode;
908
909           if (lNode->relation()) {
910               lNode->maintain();
911           }
912
913           if (rNode->relation()) {
914               rNode->maintain();
915           }
916
917           lNode->pathFa = rNode;
918           rNode->fa = lNode;
919           rNode->pathFa = lNode;
920           lNode->pathFa = rNode;
921
922           if (lNode->relation()) {
923               lNode->maintain();
924           }
925
926           if (rNode->relation()) {
927               rNode->maintain();
928           }
929
930           lNode->pathFa = rNode;
931           rNode->fa = lNode;
932           rNode->pathFa = lNode;
933           lNode->pathFa = rNode;
934
935           if (lNode->relation()) {
936               lNode->maintain();
937           }
938
939           if (rNode->relation()) {
940               rNode->maintain();
941           }
942
943           lNode->pathFa = rNode;
944           rNode->fa = lNode;
945           rNode->pathFa = lNode;
946           lNode->pathFa = rNode;
947
948           if (lNode->relation()) {
949               lNode->maintain();
950           }
951
952           if (rNode->relation()) {
953               rNode->maintain();
954           }
955
956           lNode->pathFa = rNode;
957           rNode->fa = lNode;
958           rNode->pathFa = lNode;
959           lNode->pathFa = rNode;
960
961           if (lNode->relation()) {
962               lNode->maintain();
963           }
964
965           if (rNode->relation()) {
966               rNode->maintain();
967           }
968
969           lNode->pathFa = rNode;
970           rNode->fa = lNode;
971           rNode->pathFa = lNode;
972           lNode->pathFa = rNode;
973
974           if (lNode->relation()) {
975               lNode->maintain();
976           }
977
978           if (rNode->relation()) {
979               rNode->maintain();
980           }
981
982           lNode->pathFa = rNode;
983           rNode->fa = lNode;
984           rNode->pathFa = lNode;
985           lNode->pathFa = rNode;
986
987           if (lNode->relation()) {
988               lNode->maintain();
989           }
990
991           if (rNode->relation()) {
992               rNode->maintain();
993           }
994
995           lNode->pathFa = rNode;
996           rNode->fa = lNode;
997           rNode->pathFa = lNode;
998           lNode->pathFa = rNode;
999
1000          if (lNode->relation()) {
1001              lNode->maintain();
1002          }
1003
1004          if (rNode->relation()) {
1005              rNode->maintain();
1006          }
1007
1008          lNode->pathFa = rNode;
1009          rNode->fa = lNode;
1010          rNode->pathFa = lNode;
1011          lNode->pathFa = rNode;
1012
1013          if (lNode->relation()) {
1014              lNode->maintain();
1015          }
1016
1017          if (rNode->relation()) {
1018              rNode->maintain();
1019          }
1020
1021          lNode->pathFa = rNode;
1022          rNode->fa = lNode;
1023          rNode->pathFa = lNode;
1024          lNode->pathFa = rNode;
1025
1026          if (lNode->relation()) {
1027              lNode->maintain();
1028          }
1029
1030          if (rNode->relation()) {
1031              rNode->maintain();
1032          }
1033
1034          lNode->pathFa = rNode;
1035          rNode->fa = lNode;
1036          rNode->pathFa = lNode;
1037          lNode->pathFa = rNode;
1038
1039          if (lNode->relation()) {
1040              lNode->maintain();
1041          }
1042
1043          if (rNode->relation()) {
1044              rNode->maintain();
1045          }
1046
1047          lNode->pathFa = rNode;
1048          rNode->fa = lNode;
1049          rNode->pathFa = lNode;
1050          lNode->pathFa = rNode;
1051
1052          if (lNode->relation()) {
1053              lNode->maintain();
1054          }
1055
1056          if (rNode->relation()) {
1057              rNode->maintain();
1058          }
1059
1060          lNode->pathFa = rNode;
1061          rNode->fa = lNode;
1062          rNode->pathFa = lNode;
1063          lNode->pathFa = rNode;
1064
1065          if (lNode->relation()) {
1066              lNode->maintain();
1067          }
1068
1069          if (rNode->relation()) {
1070              rNode->maintain();
1071          }
1072
1073          lNode->pathFa = rNode;
1074          rNode->fa = lNode;
1075          rNode->pathFa = lNode;
1076          lNode->pathFa = rNode;
1077
1078          if (lNode->relation()) {
1079              lNode->maintain();
1080          }
1081
1082          if (rNode->relation()) {
1083              rNode->maintain();
1084          }
1085
1086          lNode->pathFa = rNode;
1087          rNode->fa = lNode;
1088          rNode->pathFa = lNode;
1089          lNode->pathFa = rNode;
1090
1091          if (lNode->relation()) {
1092              lNode->maintain();
1093          }
1094
1095          if (rNode->relation()) {
1096              rNode->maintain();
1097          }
1098
1099          lNode->pathFa = rNode;
1100          rNode->fa = lNode;
1101          rNode->pathFa = lNode;
1102          lNode->pathFa = rNode;
1103
1104          if (lNode->relation()) {
1105              lNode->maintain();
1106          }
1107
1108          if (rNode->relation()) {
1109              rNode->maintain();
1110          }
1111
1112          lNode->pathFa = rNode;
1113          rNode->fa = lNode;
1114          rNode->pathFa = lNode;
1115          lNode->pathFa = rNode;
1116
1117          if (lNode->relation()) {
1118              lNode->maintain();
1119          }
1120
1121          if (rNode->relation()) {
1122              rNode->maintain();
1123          }
1124
1125          lNode->pathFa = rNode;
1126          rNode->fa = lNode;
1127          rNode->pathFa = lNode;
1128          lNode->pathFa = rNode;
1129
1130          if (lNode->relation()) {
1131              lNode->maintain();
1132          }
1133
1134          if (rNode->relation()) {
1135              rNode->maintain();
1136          }
1137
1138          lNode->pathFa = rNode;
1139          rNode->fa = lNode;
1140          rNode->pathFa = lNode;
1141          lNode->pathFa = rNode;
1142
1143          if (lNode->relation()) {
1144              lNode->maintain();
1145          }
1146
1147          if (rNode->relation()) {
1148              rNode->maintain();
1149          }
1150
1151          lNode->pathFa = rNode;
1152          rNode->fa = lNode;
1153          rNode->pathFa = lNode;
1154          lNode->pathFa = rNode;
1155
1156          if (lNode->relation()) {
1157              lNode->maintain();
1158          }
1159
1160          if (rNode->relation()) {
1161              rNode->maintain();
1162          }
1163
1164          lNode->pathFa = rNode;
1165          rNode->fa = lNode;
1166          rNode->pathFa = lNode;
1167          lNode->pathFa = rNode;
1168
1169          if (lNode->relation()) {
1170              lNode->maintain();
1171          }
1172
1173          if (rNode->relation()) {
1174              rNode->maintain();
1175          }
1176
1177          lNode->pathFa = rNode;
1178          rNode->fa = lNode;
1179          rNode->pathFa = lNode;
1180          lNode->pathFa = rNode;
1181
1182          if (lNode->relation()) {
1183              lNode->maintain();
1184          }
1185
1186          if (rNode->relation()) {
1187              rNode->maintain();
1188          }
1189
1190          lNode->pathFa = rNode;
1191          rNode->fa = lNode;
1192          rNode->pathFa = lNode;
1193          lNode->pathFa = rNode;
1194
1195          if (lNode->relation()) {
1196              lNode->maintain();
11
```

```

30     }
31
32     void rotate() {
33         std::swap(pathFa, fa->pathFa);
34
35         Node *o = fa;
36         int x = relation();
37
38         fa = o->fa;
39         if (fa) fa->c[o->relation()] = this;
40
41         o->c[x] = c[x ^ 1];
42         if (c[x ^ 1]) c[x ^ 1]->fa = o;
43
44         c[x ^ 1] = o;
45         o->fa = this;
46
47         o->maintain();
48         maintain();
49     }
50
51     void splay() {
52         while (fa) {
53             if (fa->fa) fa->fa->pushDown();
54             fa->pushDown();
55             pushDown();
56
57             if (!fa->fa) rotate();
58             else if (fa->relation() == relation()) fa->rotate(), rotate();
59             else rotate(), rotate();
60         }
61     }
62
63     void evert() {
64         access();
65         splay();
66         rev ^= 1;
67     }
68
69     void expose() {
70         splay();
71         pushDown();
72         if (c[1]) {
73             std::swap(c[1]->fa, c[1]->pathFa);
74             c[1] = NULL;
75             maintain();
76         }
77     }
78
79     bool splice() {
80         splay();
81         if (!pathFa) return false;
82
83         pathFa->expose();
84         pathFa->c[1] = this;
85         std::swap(fa, pathFa);

```

```
86         fa->maintain();
87         return true;
88     }
89
90     void access() {
91         expose();
92         while (splice());
93     }
94
95     int query() {
96         access();
97         splay();
98         return max;
99     }
100 } *N[MAXN], _pool[MAXN], *_curr;
101
102 LinkCutTree() : _curr(_pool) {}
103
104 void makeTree(int u, int val) {
105     N[u] = new (_curr++) Node(val);
106 }
107
108 void link(int u, int v) {
109     N[v]->evert();
110     N[v]->pathFa = N[u];
111 }
112
113 void cut(int u, int v) {
114     N[u]->evert();
115     N[v]->access();
116     N[v]->splay();
117     N[v]->pushDown();
118     N[v]->c[0]->fa = NULL;
119     N[v]->c[0] = NULL;
120     N[v]->maintain();
121 }
122
123 int query(int u, int v) {
124     N[u]->evert();
125     return N[v]->query();
126 }
127 } lct;
128
129 int main() {
130
131     return 0;
132 }
```

2.7 主席树

2.7.1 链上区间 k 小值

```
1 #include <cstdio>
2 #include <algorithm>
```

```

3
4 const int MAXN = 200005;
5 const int MAXN_LOG = 20;
6
7 template <typename T, size_t SIZE>
8 struct MemoryPool {
9     char mem[sizeof(T) * SIZE], *top;
10
11     MemoryPool() : top(mem) {}
12
13     void *alloc() {
14         char *res = top;
15         top += sizeof (T);
16         return (void *) res;
17     }
18 };
19
20 class PSegT {
21 private:
22     struct Node {
23         Node *lc, *rc;
24         int cnt;
25         static MemoryPool<Node, MAXN * MAXN_LOG> pool;
26         static Node *nil;
27
28         Node(int cnt) : lc(nil), rc(nil), cnt(cnt) {
29             static bool init = true;
30             if (init) { lc = rc = this; init = false; }
31         }
32         Node(Node *lc = nil, Node *rc = nil) : lc(lc), rc(rc), cnt(lc->cnt + rc->cnt) {}
33
34         void *operator new(size_t) {
35             return pool.alloc();
36         }
37
38         Node *insert(int l, int r, int val) {
39             if (val == l && val == r) return new Node(cnt + 1);
40             int mid = l + ((r - l) >> 1);
41             if (val <= mid) return new Node(lc->insert(l, mid, val), rc);
42             else return new Node(lc, rc->insert(mid + 1, r, val));
43         }
44
45         int rank() const { return lc->cnt; }
46     } *root[MAXN];
47     int n;
48
49 public:
50     void build(int *a, int n) {
51         this->n = n;
52         root[0] = new Node();
53         for (int i = 1; i <= n; i++) root[i] = root[i - 1]->insert(1, n, a[i]);
54     }
55
56     int query(int l, int r, int k) {
57         Node *L = root[l - 1], *R = root[r];
58         int min = 1, max = n;

```

```
59     while (min < max) {
60         int mid = min + ((max - min) >> 1), temp = R->rank() - L->rank();
61         if (k <= temp) L = L->lc, R = R->lc, max = mid;
62         else L = L->rc, R = R->rc, k -= temp, min = mid + 1;
63     }
64     return min;
65 }
66 } pSegT;
67 MemoryPool<PSegT::Node, MAXN * MAXN_LOG> PSegT::Node::pool;
68 PSegT::Node *PSegT::Node::nil = new PSegT::Node(0);
69
70 int map[MAXN];
71 void discrete(int *a, int n) {
72     std::copy(a + 1, a + n + 1, map);
73     std::sort(map, map + n);
74     int *end = std::unique(map, map + n);
75     for (int i = 1; i <= n; i++)
76         a[i] = std::lower_bound(map, end, a[i]) - map + 1;
77 }
78
79 int main() {
80     int n, q;
81     scanf("%d %d", &n, &q);
82
83     static int a[MAXN];
84     for (int i = 1; i <= n; i++) scanf("%d", &a[i]);
85     discrete(a, n);
86
87     pSegT.build(a, n);
88
89     while (q--) {
90         int l, r, k;
91         scanf("%d %d %d", &l, &r, &k);
92         printf("%d\n", map[pSegT.query(l, r, k) - 1]);
93     }
94
95     return 0;
96 }
```

2.7.2 树上两点间 k 小值

```
1 #include <cstdio>
2 #include <climits>
3 #include <algorithm>
4
5 const int MAXN = 100005;
6 const int MAXN_LOG = 18;
7
8 template <typename T, size_t SIZE>
9 struct MemoryPool {
10     char mem[SIZE * sizeof (T)], *top;
11
12     MemoryPool() : top(mem) {}
13
14     void *alloc() {
```

```

15     char *res = top;
16     top += sizeof (T);
17     return (void *) res;
18 }
19 };
20
21 struct PSegT *null;
22 struct PSegT {
23     PSegT *lc, *rc;
24     int cnt;
25     static MemoryPool<PSegT, MAXN * 40> pool;
26
27     PSegT(int cnt) : cnt(cnt), lc(null), rc(null) {}
28     PSegT(PSegT *lc, PSegT *rc) : lc(lc), rc(rc), cnt(lc->cnt + rc->cnt) {}
29
30     void *operator new(size_t) {
31         return pool.alloc();
32     }
33
34     PSegT *insert(int l, int r, int val) {
35         if (val < l || val > r) return null;
36         if (val == l && val == r) return new PSegT(cnt + 1);
37         int mid = l + (r - l) / 2;
38         if (val <= mid) return new PSegT(lc->insert(l, mid, val), rc);
39         else return new PSegT(lc, rc->insert(mid + 1, r, val));
40     }
41 };
42 MemoryPool<PSegT, MAXN * 40> PSegT::pool;
43
44 struct Edge;
45 struct Node;
46
47 struct Node {
48     Edge *e;
49     PSegT *seg;
50     Node *f[MAXN_LOG];
51     int dep, val;
52 } N[MAXN];
53
54 struct Edge {
55     Node *u, *v;
56     Edge *next;
57
58     Edge() {}
59     Edge(Node *u, Node *v) : u(u), v(v), next(u->e) {}
60 } _pool[MAXN << 1], *_curr = _pool;
61
62 void addEdge(int u, int v) {
63     N[u].e = new (_curr++) Edge(&N[u], &N[v]);
64     N[v].e = new (_curr++) Edge(&N[v], &N[u]);
65 }
66
67 void init() {
68     null = new PSegT(0);
69     null->lc = null->rc = null;
70 }

```

```
71
72 void dfs(Node *u, bool init = true) {
73     if (init) {
74         u->f[0] = u;
75         u->dep = 1;
76         u->seg = null->insert(0, INT_MAX, u->val);
77     }
78
79     for (int i = 1; i < MAXN_LOG; i++) u->f[i] = u->f[i - 1]->f[i - 1];
80
81     for (Edge *e = u->e; e; e = e->next) if (e->v != u->f[0]) {
82         e->v->f[0] = u;
83         e->v->dep = u->dep + 1;
84         e->v->seg = u->seg->insert(0, INT_MAX, e->v->val);
85
86         dfs(e->v, false);
87     }
88 }
89
90 Node *lca(Node *u, Node *v) {
91     if (u->dep < v->dep) std::swap(u, v);
92
93     for (int i = MAXN_LOG - 1; ~i; i--) {
94         if (u->f[i]->dep >= v->dep) u = u->f[i];
95     }
96
97     for (int i = MAXN_LOG - 1; ~i; i--) {
98         if (u->f[i] != v->f[i]) {
99             u = u->f[i];
100            v = v->f[i];
101        }
102    }
103
104    return u == v ? u : u->f[0];
105 }
106
107 int query(int u, int v, int k) {
108     Node *p = lca(&N[u], &N[v]);
109     PSegT *su = N[u].seg, *sv = N[v].seg;
110     PSegT *sp = p->seg, *sf = (p == p->f[0] ? null : p->f[0]->seg);
111
112     int l = 0, r = INT_MAX;
113     while (l < r) {
114         int mid = l + (r - l) / 2;
115         int temp = su->lc->cnt + sv->lc->cnt - sp->lc->cnt - sf->lc->cnt;
116         if (k <= temp) {
117             su = su->lc;
118             sv = sv->lc;
119             sp = sp->lc;
120             sf = sf->lc;
121             r = mid;
122         } else {
123             k -= temp;
124             su = su->rc;
125             sv = sv->rc;
126             sp = sp->rc;
```

```

127         sf = sf->rc;
128         l = mid + 1;
129     }
130 }
131 return l;
132 }
133
134 int main() {
135     init();
136
137     int n, q;
138     scanf("%d %d", &n, &q);
139
140     for (int i = 1; i <= n; i++) scanf("%d", &N[i].val);
141
142     for (int i = 1, u, v; i < n; i++) {
143         scanf(" %d %d", &u, &v);
144         addEdge(u, v);
145     }
146
147     dfs(&N[1]);
148
149     int lastAns = 0;
150     while (q--) {
151         int u, v, k;
152         scanf(" %d %d %d", &u, &v, &k);
153         u ^= lastAns;
154         printf(" %d", lastAns = query(u, v, k));
155         q ? puts("") : 0;
156     }
157
158     return 0;
159 }
```

2.8 可持久化 Treap

```

1 #include <cstdio>
2 #include <cstdlib>
3 #include <algorithm>
4
5 const int MAXN = 100005;
6
7 struct PTreap {
8     struct Node {
9         Node *lc, *rc;
10        int val, size;
11
12        Node() {}
13        Node(int val, Node *lc, Node *rc) : val(val), lc(lc), rc(rc), size((lc ? lc->size : 0) + 1 + (rc ?
14            rc->size : 0)) {}
15    } *_root, _pool[MAXN * 20], *_curr;
16
17    PTreap() : root(NULL), _curr(_pool) {}
```

```
18     static int size(const Node *u) {
19         return u ? u->size : 0;
20     }
21
22     Node *merge(Node *a, Node *b) {
23         if (!a) return b;
24         if (!b) return a;
25
26         if (rand() % (size(a) + size(b)) < size(a)) {
27             return new (_curr++) Node(a->val, a->lc, merge(a->rc, b));
28         } else {
29             return new (_curr++) Node(b->val, merge(a, b->lc), b->rc);
30         }
31     }
32
33     std::pair<Node *, Node *> split(Node *u, int pos) {
34         std::pair<Node *, Node *> res(NULL, NULL);
35         if (!u) return res;
36
37         if (pos <= size(u->lc)) {
38             res = split(u->lc, pos);
39             u = new (_curr++) Node(u->val, res.second, u->rc);
40         } else {
41             res = split(u->rc, pos - size(u->lc) - 1);
42             u = new (_curr++) Node(u->val, u->lc, res.first);
43         }
44
45         return res;
46     }
47 };
48
49 int main() {
50
51     return 0;
52 }
```

2.9 并查集

```
1 #include <cstdio>
2
3 const int MAXN = 1000005;
4
5 struct DSU {
6     int fa[MAXN];
7
8     void init(int n) { for (int i = 0; i < n; i++) fa[i] = i; }
9     int find(int x) { return x == fa[x] ? x : fa[x] = find(fa[x]); }
10    void merge(int x, int y) { fa[find(y)] = find(x); }
11    bool test(int x, int y) { return find(x) == find(y); }
12 } dsu;
13
14 int main() {
15     int n, q;
16     scanf("%d %d", &n, &q);
```

```

17
18     dsu.init(n);
19
20     while (q--) {
21         int op, u, v;
22         scanf("%d %d %d", &op, &u, &v);
23         if (op == 0) dsu.merge(u, v);
24         else puts(dsu.test(u, v) ? "Yes" : "No");
25     }
26
27     return 0;
28 }
```

2.10 Sparse Table

```

1 #include <cstdio>
2 #include <algorithm>
3
4 const int MAXN = 500005;
5 const int MAXN_LOG = 20;
6
7 class SparseTable {
8 private:
9     int st[MAXN][MAXN_LOG], log[MAXN];
10
11 public:
12     void init(int *a, int n) {
13         int t = 0;
14         for (int i = 0; i <= n; i++) {
15             while (1 << (t + 1) <= i) t++;
16             log[i] = t;
17         }
18
19         for (int i = 1; i <= n; i++) st[i][0] = a[i];
20
21         for (int j = 1; j <= log[n]; j++) {
22             for (int i = 1; i <= n; i++) {
23                 if (i + (1 << (j - 1)) <= n) st[i][j] = std::min(st[i][j - 1], st[i + (1 << (j - 1))][j - 1]);
24                 else st[i][j] = st[i][j - 1];
25             }
26         }
27     }
28
29     int query(int l, int r) {
30         int t = log[r - l + 1];
31         return std::min(st[l][t], st[r - (1 << t) + 1][t]);
32     }
33 } st;
34
35 int main() {
36     int n;
37     scanf("%d", &n);
38 }
```

```
39     static int a[MAXN];
40     for (int i = 1; i <= n; i++) scanf("%d", &a[i]);
41
42     st.init(a, n);
43
44     int q;
45     scanf("%d", &q);
46     while (q--) {
47         int l, r;
48         scanf("%d %d", &l, &r);
49         printf("%d\n", st.query(l, r));
50     }
51
52     return 0;
53 }
```

2.11 K-d Tree

2.11.1 曼哈顿距离最近点

```
1 #include <iostream>
2 #include <climits>
3 #include <algorithm>
4 #include <functional>
5
6 const int MAXN = 500005;
7
8 struct Point {
9     int x, y;
10    Point(int x = 0, int y = 0) : x(x), y(y) {}
11 } P[MAXN];
12
13 int dist(const Point &a, const Point &b) {
14     return abs(a.x - b.x) + abs(a.y - b.y);
15 }
16
17 std::function<bool(const Point &, const Point &)> cmp[2] = {
18     [] (const Point &a, const Point &b) {
19         return a.y == b.y ? a.x < b.x : a.y < b.y;
20     },
21     [] (const Point &a, const Point &b) {
22         return a.x == b.x ? a.y < b.y : a.x < b.x;
23     }
24 };
25
26 struct KDTree {
27     struct Node {
28         Node *c[2];
29         Point p, r1, r2;
30
31         Node() {}
32         Node (Point p) : p(p), r1(p), r2(p) {
33             c[0] = c[1] = NULL;
34         }
35     };
36 }
```

```

35
36     void maintain() {
37         if (c[0]) {
38             r1.x = std::min(r1.x, c[0]->r1.x);
39             r1.y = std::min(r1.y, c[0]->r1.y);
40             r2.x = std::max(r2.x, c[0]->r2.x);
41             r2.y = std::max(r2.y, c[0]->r2.y);
42         }
43         if (c[1]) {
44             r1.x = std::min(r1.x, c[1]->r1.x);
45             r1.y = std::min(r1.y, c[1]->r1.y);
46             r2.x = std::max(r2.x, c[1]->r2.x);
47             r2.y = std::max(r2.y, c[1]->r2.y);
48         }
49     }
50
51     int dist(const Point &p) {
52         int res = 0;
53         if (p.x < r1.x || r2.x < p.x) res += p.x < r1.x ? r1.x - p.x : p.x - r2.x;
54         if (p.y < r1.y || r2.y < p.y) res += p.y < r1.y ? r1.y - p.y : p.y - r2.y;
55         return res;
56     }
57
58     void query(const Point &p, int &res) {
59         res = std::min(res, ::dist(this->p, p));
60
61         if (!(c[0] || c[1])) return;
62         int k = (c[0] && c[1] ? c[0]->dist(p) > c[1]->dist(p) : (c[0] ? 0 : 1));
63
64         c[k]->query(p, res);
65         if (c[k ^ 1] && c[k ^ 1]->dist(p) < res) c[k ^ 1]->query(p, res);
66     }
67 } *root, _pool[MAXN << 1], *_curr;
68
69 KDTree() : root(NULL) {
70     _curr = _pool;
71 }
72
73 Node *build(int l, int r, Point P[], int d = 0) {
74     if (l > r) return NULL;
75     if (l == r) return new (_curr++) Node(P[l]);
76
77     int mid = l + ((r - l) >> 1);
78     std::nth_element(P + l, P + mid, P + r + 1, cmp[d]);
79
80     Node *u = new (_curr++) Node(P[mid]);
81     u->c[0] = build(l, mid - 1, P, d ^ 1);
82     u->c[1] = build(mid + 1, r, P, d ^ 1);
83     u->maintain();
84
85     return u;
86 }
87
88 void insert(const Point &p) {
89     Node **u = &root;
90     int d = 0;

```

```
91     while (*u) {
92         int k = cmp[d](p, (*u)->p) ^ 1;
93         d ^= 1;
94         (*u)->r1.x = std::min(p.x, (*u)->r1.x);
95         (*u)->r1.y = std::min(p.y, (*u)->r1.y);
96         (*u)->r2.x = std::max(p.x, (*u)->r2.x);
97         (*u)->r2.y = std::max(p.y, (*u)->r2.y);
98         u = &(*u)->c[k];
99     }
100    *u = new (_curr++) Node(p);
101 }
102
103    int query(const Point &p) {
104        int res = INT_MAX;
105        root->query(p, res);
106        return res;
107    }
108 } kdT;
109
110 int main() {
111     int n, m;
112     scanf("%d %d", &n, &m);
113     for (int i = 1; i <= n; i++) scanf("%d %d", &P[i].x, &P[i].y);
114
115     kdT.root = kdT.build(1, n, P);
116
117     while (m--) {
118         int op;
119         Point p;
120         scanf("%d %d %d", &op, &p.x, &p.y);
121         if (op == 1) kdT.insert(p);
122         else printf("%d\n", kdT.query(p));
123     }
124
125     return 0;
126 }
```

2.11.2 全局欧几里得距离 k -远点对

```
1 #include <cstdio>
2 #include <climits>
3 #include <vector>
4 #include <queue>
5 #include <algorithm>
6 #include <functional>
7
8 const int MAXN = 100005;
9
10 struct Point {
11     int x, y;
12     Point(int x = 0, int y = 0) : x(x), y(y) {}
13 } P[MAXN];
14
15 long long dist(const Point &a, const Point &b) {
16     return (long long) (a.x - b.x) * (a.x - b.x) + (long long) (a.y - b.y) * (a.y - b.y);
```

```

17 }
18
19 std::function<bool(const Point &, const Point &)> cmp[2] = {
20     [](const Point &a, const Point &b) {
21         return a.x == b.x ? a.y < b.y : a.x < b.x;
22     },
23     [](const Point &a, const Point &b) {
24         return a.y == b.y ? a.x < b.x : a.y < b.y;
25     }
26 };
27
28 struct KDTree {
29     struct Node {
30         Node *c[2];
31         Point p, r1, r2;
32
33         Node() {}
34         Node(Point p) : p(p), r1(p), r2(p) {
35             c[0] = c[1] = NULL;
36         }
37
38         void maintain() {
39             if (c[0]) {
40                 r1.x = std::min(r1.x, c[0]->r1.x);
41                 r1.y = std::min(r1.y, c[0]->r1.y);
42                 r2.x = std::max(r2.x, c[0]->r2.x);
43                 r2.y = std::max(r2.y, c[0]->r2.y);
44             }
45             if (c[1]) {
46                 r1.x = std::min(r1.x, c[1]->r1.x);
47                 r1.y = std::min(r1.y, c[1]->r1.y);
48                 r2.x = std::max(r2.x, c[1]->r2.x);
49                 r2.y = std::max(r2.y, c[1]->r2.y);
50             }
51         }
52
53         long long dist(const Point &p) {
54             return std::max({::dist(p, r1), ::dist(p, r2),
55                             ::dist(p, Point(r1.x, r2.y)),
56                             ::dist(p, Point(r2.x, r1.y))});
57         }
58
59         void query(const Point &p, std::priority_queue<long long, std::vector<long long>, std::greater<long
60             long> > &q) {
61             long long d = ::dist(p, this->p);
62
63             if (d > q.top()) q.pop(), q.push(d);
64             if (!(c[0] || c[1])) return;
65
66             long long dis[2] = {c[0] ? c[0]->dist(p) : INT_MIN,
67                                 c[1] ? c[1]->dist(p) : INT_MIN};
68             int k = dis[0] < dis[1];
69
70             c[k]->query(p, q);
71             if (c[k ^ 1] && dis[k ^ 1] > q.top()) c[k ^ 1]->query(p, q);
72         }
73     }
74 }
```

```

72     } *_root, *_pool[MAXN], *_curr;
73
74     KDTree() : root(NULL) {
75         _curr = _pool;
76     }
77
78     Node *build(Point *l, Point *r, int d = 0) {
79         if (l > r) return NULL;
80         if (l == r) return new (_curr++) Node(*l);
81
82         Point *mid = l + ((r - l) >> 1);
83         std::nth_element(l, mid, r + 1, cmp[d]);
84
85         Node *u = new (_curr++) Node(*mid);
86         u->c[0] = build(l, mid - 1, d ^ 1);
87         u->c[1] = build(mid + 1, r, d ^ 1);
88         u->maintain();
89
90         return u;
91     }
92
93     long long query(Point P[], int n, int k) {
94         std::priority_queue<long long, std::vector<long long>, std::greater<long long> > q;
95         while (!q.empty()) q.pop();
96         for (int i = 0; i < k << 1; i++) q.push(-1);
97         for (int i = 1; i <= n; i++) root->query(P[i], q);
98         return q.top();
99     }
100 } kdT;
101
102 int main() {
103     int n, k;
104     scanf("%d %d", &n, &k);
105     for (int i = 1; i <= n; i++) scanf("%d %d", &P[i].x, &P[i].y);
106
107     kdT.root = kdT.build(P + 1, P + n);
108     printf("%lld\n", kdT.query(P, n, k));
109
110     return 0;
111 }
```

2.12 单调队列

```

1 #include <cstdio>
2 #include <queue>
3
4 template <typename T, typename Cmp = std::less<T> > // maximum by default
5 struct MonoQueue {
6     std::deque<T> q, m;
7     Cmp cmp;
8
9     void push(int x) {
10         q.push_back(x);
11         while (!m.empty() && cmp(m.back(), x)) m.pop_back();
```

```

12         m.push_back(x);
13     }
14
15     void pop() {
16         int x = q.front();
17         q.pop_front();
18         if (x == m.front()) m.pop_front();
19     }
20
21     size_t size() { return q.size(); }
22
23     int top() { return m.front(); }
24 };
25
26 int main() {
27
28     return 0;
29 }
```

2.13 Sqrt Tree

```

1 #include <cstdio>
2 #include <vector>
3 #include <algorithm>
4
5 inline int log2Up(int n) {
6     int res = 0;
7     while ((1 << res) < n) {
8         res++;
9     }
10    return res;
11 }
12
13 template <typename T, T (*op)(T, T)>
14 class SqrtTree {
15 private:
16     int n, lg, indexSz;
17     std::vector<T> v;
18     std::vector<int> clz, layers, onLayer;
19     std::vector<std::vector<T>> pref, suf, between;
20
21     void buildBlock(int layer, int l, int r) {
22         pref[layer][l] = v[l];
23         for (int i = l + 1; i < r; i++) {
24             pref[layer][i] = op(pref[layer][i - 1], v[i]);
25         }
26         suf[layer][r - 1] = v[r - 1];
27         for (int i = r - 2; i >= l; i--) {
28             suf[layer][i] = op(v[i], suf[layer][i + 1]);
29         }
30     }
31
32     void buildBetween(int layer, int lBound, int rBound, int betweenOffs) {
33         int bSzLog = (layers[layer] + 1) >> 1;
```

```
34     int bCntLog = layers[layer] >> 1;
35     int bSz = 1 << bSzLog;
36     int bCnt = (rBound - lBound + bSz - 1) >> bSzLog;
37     for (int i = 0; i < bCnt; i++) {
38         T ans;
39         for (int j = i; j < bCnt; j++) {
40             T add = suf[layer][lBound + (j << bSzLog)];
41             ans = (i == j) ? add : op(ans, add);
42             between[layer - 1][betweenOffs + lBound + (i << bCntLog) + j] = ans;
43         }
44     }
45 }
46
47 void buildBetweenZero() {
48     int bSzLog = (lg + 1) >> 1;
49     for (int i = 0; i < indexSz; i++) {
50         v[n + i] = suf[0][i << bSzLog];
51     }
52     build(1, n, n + indexSz, (1 << lg) - n);
53 }
54
55 void updateBetweenZero(int bid) {
56     int bSzLog = (lg + 1) >> 1;
57     v[n + bid] = suf[0][bid << bSzLog];
58     update(1, n, n + indexSz, (1 << lg) - n, n + bid);
59 }
60
61 void build(int layer, int lBound, int rBound, int betweenOffs) {
62     if (layer >= (int)layers.size()) return;
63     int bSz = 1 << ((layers[layer] + 1) >> 1);
64     for (int l = lBound; l < rBound; l += bSz) {
65         int r = std::min(l + bSz, rBound);
66         buildBlock(layer, l, r);
67         build(layer + 1, l, r, betweenOffs);
68     }
69     if (layer == 0) {
70         buildBetweenZero();
71     } else {
72         buildBetween(layer, lBound, rBound, betweenOffs);
73     }
74 }
75
76 void update(int layer, int lBound, int rBound, int betweenOffs, int x) {
77     if (layer >= (int)layers.size()) return;
78     int bSzLog = (layers[layer] + 1) >> 1;
79     int bSz = 1 << bSzLog;
80     int blockIdx = (x - lBound) >> bSzLog;
81     int l = lBound + (blockIdx << bSzLog);
82     int r = std::min(l + bSz, rBound);
83     buildBlock(layer, l, r);
84     if (layer == 0) {
85         updateBetweenZero(blockIdx);
86     } else {
87         buildBetween(layer, lBound, rBound, betweenOffs);
88     }
89     update(layer + 1, l, r, betweenOffs, x);
```

```

90     }
91
92     T query(int l, int r, int betweenOffs, int base) {
93         if (l == r) return v[l];
94         if (l + 1 == r) return op(v[l], v[r]);
95         int layer = onLayer[clz[(l - base) ^ (r - base)]];;
96         int bSzLog = (layers[layer] + 1) >> 1;
97         int bCntLog = layers[layer] >> 1;
98         int lBound = (((l - base) >> layers[layer]) << layers[layer]) + base;
99         int lBlock = ((l - lBound) >> bSzLog) + 1;
100        int rBlock = ((r - lBound) >> bSzLog) - 1;
101        T ans = suf[layer][l];
102        if (lBlock <= rBlock) {
103            T add = (layer == 0)
104                ? (query(n + lBlock, n + rBlock, (1 << lg) - n, n))
105                : (between[layer - 1][betweenOffs + lBound + (lBlock << bCntLog) + rBlock]);
106            ans = op(ans, add);
107        }
108        ans = op(ans, pref[layer][r]);
109        return ans;
110    }
111
112 public:
113     inline T query(int l, int r) { return query(l, r, 0, 0); }
114
115     inline void update(int x, const T &item) {
116         v[x] = item;
117         update(0, 0, n, 0, x);
118     }
119
120     SqrtTree(const std::vector<T> &a) : n((int) a.size()), lg(log2Up(n)), v(a), clz(1 << lg), onLayer(lg +
121         1) {
122         clz[0] = 0;
123         for (int i = 1; i < (int)clz.size(); i++) {
124             clz[i] = clz[i >> 1] + 1;
125         }
126         int tlg = lg;
127         while (tlg > 1) {
128             onLayer[tlg] = (int)layers.size();
129             layers.push_back(tlg);
130             tlg = (tlg + 1) >> 1;
131         }
132         for (int i = lg - 1; i >= 0; i--) {
133             onLayer[i] = std::max(onLayer[i], onLayer[i + 1]);
134         }
135         int betweenLayers = std::max(0, (int)layers.size() - 1);
136         int bSzLog = (lg + 1) >> 1;
137         int bSz = 1 << bSzLog;
138         indexSz = (n + bSz - 1) >> bSzLog;
139         v.resize(n + indexSz);
140         pref.assign(layers.size(), std::vector<T>(n + indexSz));
141         suf.assign(layers.size(), std::vector<T>(n + indexSz));
142         between.assign(betweenLayers, std::vector<T>((1 << lg) + bSz));
143         build(0, 0, n, 0);
144     }
145 };

```

```
145
146 int Op(int a, int b) {
147     return a + b; // gcd, min, max, &, |, ^
148 }
149
150 int main() {
151     int n;
152     scanf("%d", &n);
153     std::vector<int> a(n);
154     for (int i = 0; i < n; i++) scanf("%d", &a[i]);
155
156     SqrtTree<int, Op> sqrtT(a);
157
158     return 0;
159 }
```

3 树与图

3.1 单源最短路

3.1.1 Dijkstra

```
1 #include <cstdio>
2 #include <climits>
3 #include <queue>
4 #include <algorithm>
5
6 const int MAXN = 100005;
7 const int MAXM = 500005;
8
9 struct Edge;
10 struct Node;
11
12 struct Node {
13     Edge *e; // use std::vector<Edge> when dense graph.
14     int dist;
15     bool vis;
16 } N[MAXN];
17
18 struct Edge {
19     Node *u, *v;
20     Edge *next;
21     int w;
22
23     Edge() {}
24     Edge(Node *u, Node *v, int w) : u(u), v(v), w(w), next(u->e) {}
25 } _pool[MAXM << 1], *_curr = _pool;
26
27 void addEdge(int u, int v, int w) {
28     N[u].e = new (_curr++) Edge(&N[u], &N[v], w);
29     N[v].e = new (_curr++) Edge(&N[v], &N[u], w);
30 }
31
32 namespace Dijkstra {
33     struct HeapNode {
34         Node *u;
35         int dist;
36
37         HeapNode(int dist, Node *u) : u(u), dist(dist) {}
38
39         bool operator<(const HeapNode &rhs) const {
40             return dist > rhs.dist;
41         }
42     };
43
44     void dijkstra(Node *s) {
45         std::priority_queue<HeapNode> q;
46         s->dist = 0;
47         q.emplace(0, s);
```

```
48
49     while (!q.empty()) {
50         Node *u = q.top().u;
51         q.pop();
52
53         if (u->vis) continue;
54         u->vis = true;
55
56         for (Edge *e = u->e; e; e = e->next) {
57             if (e->v->dist > u->dist + e->w) {
58                 e->v->dist = u->dist + e->w;
59
60                 q.emplace(e->v->dist, e->v);
61             }
62         }
63     }
64 }
65
66 int solve(int s, int t, int n) {
67     for (int i = 1; i <= n; i++) {
68         N[i].dist = INT_MAX;
69         N[i].vis = false;
70     }
71
72     dijkstra(&N[s]);
73
74     return N[t].dist;
75 }
76 }
77
78 int main() {
79     int n, m, s, t;
80     scanf("%d %d %d %d", &n, &m, &s, &t);
81
82     for (int i = 0, u, v, w; i < m; i++) {
83         scanf("%d %d %d", &u, &v, &w);
84         addEdge(u, v, w);
85     }
86
87     printf("%d\n", Dijkstra::solve(s, t, n));
88
89     return 0;
90 }
```

3.1.2 队列优化的 Bellman-Ford

不应该使用，除非是含负权边的单次最短路（如差分约束）。求负权边的多次最短路时，请使用类似 Johnson 算法的方式。

```
1 #include <cstdio>
2 #include <climits>
3 #include <queue>
4 #include <algorithm>
5
6 const int MAXN = 100005;
```

```

7  const int MAXM = 500005;
8
9  struct Edge;
10 struct Node;
11
12 struct Node {
13     Edge *e; // use std::vector<Edge> when dense graph
14     int dist, cnt;
15     bool inq;
16 } N[MAXN];
17
18 struct Edge {
19     Node *u, *v;
20     Edge *next;
21     int w;
22
23     Edge() {}
24     Edge(Node *u, Node *v, int w) : u(u), v(v), w(w), next(u->e) {}
25 } _pool[MAXM << 1], *_curr = _pool;
26
27 void addEdge(int u, int v, int w) {
28     N[u].e = new (_curr++) Edge(&N[u], &N[v], w);
29     N[v].e = new (_curr++) Edge(&N[v], &N[u], w);
30 }
31
32 namespace BellmanFord {
33     bool bellmanFord(Node *s, int n) {
34         std::queue<Node *> q;
35         q.push(s);
36         s->dist = 0;
37
38         while (!q.empty()) {
39             Node *u = q.front();
40             q.pop();
41             u->inq = false;
42
43             for (Edge *e = u->e; e; e = e->next) {
44                 if (e->v->dist > u->dist + e->w) {
45                     e->v->dist = u->dist + e->w;
46
47                     if (++e->v->cnt >= n) return false;
48                     if (!e->v->inq) {
49                         e->v->inq = true;
50                         q.push(e->v);
51                     }
52                 }
53             }
54         }
55         return true;
56     }
57
58     int solve(int s, int t, int n) {
59         for (int i = 1; i <= n; i++) {
60             N[i].dist = INT_MAX;
61             N[i].cnt = 0;
62             N[i].inq = false;

```

```
63         }
64
65     return bellmanFord(&N[s], n) ? N[t].dist : -1;
66 }
67 }
68
69 int main() {
70     int n, m, s, t;
71     scanf("%d %d %d %d", &n, &m, &s, &t);
72
73     for (int i = 0, u, v, w; i < m; i++) {
74         scanf(" %d %d %d", &u, &v, &w);
75         addEdge(u, v, w);
76     }
77
78     printf("%d\n", BellmanFord::solve(s, t, n));
79
80     return 0;
81 }
```

3.2 两点间 k 短路

```
1 #include <cstdio>
2 #include <climits>
3 #include <queue>
4 #include <algorithm>
5
6 const int MAXN = 1005;
7 const int MAXM = 100005;
8
9 struct Edge;
10 struct Node {
11     Edge *e, *eo;
12     int dist, times;
13     bool vis;
14 } N[MAXN];
15
16 struct Edge {
17     Node *u, *v;
18     Edge *next;
19     int w;
20
21     Edge() {}
22     Edge(Node *u, Node *v, int w, Edge *next) : u(u), v(v), w(w), next(next) {}
23 } _pool[MAXM << 1], *_curr = _pool;
24
25 void addEdge(int u, int v, int w) {
26     N[u].e = new (_curr++) Edge(&N[u], &N[v], w, N[u].e);
27     N[v].eo = new (_curr++) Edge(&N[v], &N[u], w, N[v].eo);
28 }
29
30 namespace Dijkstra {
31     struct HeapNode {
32         Node *u;
```

```

33     int dist;
34
35     HeapNode(Node *u, int dist) : u(u), dist(dist) {}
36
37     bool operator<(const HeapNode &rhs) const {
38         return dist > rhs.dist;
39     }
40 };
41
42 void dijkstra(Node *s) {
43     static std::priority_queue<HeapNode> q;
44     s->dist = 0;
45     q.emplace(s, 0);
46     while (!q.empty()) {
47         Node *u = q.top().u;
48         q.pop();
49
50         if (u->vis) continue;
51         u->vis = true;
52
53         for (Edge *e = u->eo; e; e = e->next) {
54             if (e->v->dist > u->dist + e->w) {
55                 e->v->dist = u->dist + e->w;
56                 q.emplace(e->v, e->v->dist);
57             }
58         }
59     }
60 }
61
62 void solve(int s, int n) {
63     for (int i = 1; i <= n; i++) {
64         N[i].dist = INT_MAX;
65         N[i].vis = false;
66     }
67
68     dijkstra(&N[s]);
69 }
70 }
71
72 namespace KthShortest {
73     struct HeapNode {
74         Node *u;
75         int curr, last;
76
77         HeapNode(Node *u, int curr, int last) : u(u), curr(curr), last(last) {}
78
79         bool operator<(const HeapNode &rhs) const {
80             return curr + last > rhs.curr + rhs.last;
81         }
82     };
83
84     int astar(Node *s, Node *t, int k) {
85         static std::priority_queue<HeapNode> q;
86         q.emplace(s, 0, s->dist);
87         while (!q.empty()) {
88             Node *u = q.top().u;

```

```

89         int curr = q.top().curr;
90         int last = q.top().last;
91         q.pop();
92
93         ++u->times;
94         if (u->times == k && u == t) return curr + last;
95         if (u->times > k) continue;
96
97         for (Edge *e = u->e; e; e = e->next) q.emplace(e->v, curr + e->w, e->v->dist);
98     }
99
100    return -1;
101 }
102
103 int solve(int s, int t, int k, int n) {
104     Dijkstra::solve(t, n);
105     return astar(&N[s], &N[t], k);
106 }
107 }
108
109 int main() {
110     int n, m;
111     scanf("%d %d", &n, &m);
112     for (int i = 0, u, v, w; i < m; i++) {
113         scanf("%d %d %d", &u, &v, &w);
114         addEdge(u, v, w);
115     }
116
117     int s, t, k;
118     scanf("%d %d %d", &s, &t, &k);
119     if (s == t) ++k;
120
121     int ans = KthShortest::solve(s, t, k, n);
122     printf("%d\n", ans);
123
124     return 0;
125 }
```

3.3 斯坦纳树

```

1 #include <cstdio>
2 #include <climits>
3 #include <queue>
4 #include <algorithm>
5
6 const int MAXN = 50005;
7 const int MAXM = 300005;
8 const int MAXD = 5;
9
10 struct Edge;
11 struct Node {
12     Edge *e;
13     bool vis[1 << MAXD];
14     long long f[1 << MAXD];
```

```

15 } N[MAXN];
16
17 struct Edge {
18     Node *u, *v;
19     Edge *next;
20     int w;
21
22     Edge() {}
23     Edge(Node *u, Node *v, int w) : u(u), v(v), w(w), next(u->e) {}
24 } _pool[MAXM << 1], *_curr = _pool;
25
26 void addEdge(int u, int v, int w) {
27     N[u].e = new (_curr++) Edge(&N[u], &N[v], w);
28     N[v].e = new (_curr++) Edge(&N[v], &N[u], w);
29 }
30
31 namespace Dijkstra {
32     struct HeapNode {
33         Node *u;
34         int s;
35         long long dist;
36
37         HeapNode(Node *u, int s, long long dist) : u(u), s(s), dist(dist) {}
38
39         bool operator<(const HeapNode &rhs) const {
40             return dist > rhs.dist;
41         }
42     };
43
44     std::priority_queue<HeapNode> q;
45     void dijkstra() {
46         while (!q.empty()) {
47             Node *u = q.top().u;
48             int s = q.top().s;
49             q.pop();
50
51             if (u->vis[s]) continue;
52             u->vis[s] = true;
53
54             for (Edge *e = u->e; e; e = e->next) {
55                 if (e->v->f[s] > u->f[s] + e->w) {
56                     e->v->f[s] = u->f[s] + e->w;
57                     q.emplace(e->v, s, e->v->f[s]);
58                 }
59             }
60         }
61     }
62
63     void init(int n, int d) {
64         for (int i = 1; i <= n; i++) for (int s = 0; s < 1 << d; s++) {
65             N[i].f[s] = LLONG_MAX >> 1ll;
66             N[i].vis[s] = false;
67         }
68         while (!q.empty()) q.pop();
69     }
70 }

```

3 树与图

```
71 // O(3^d + 2^d * m \log n)
72 long long steiner(int n, int *p, int d) {
73     Dijkstra::init(n, d);
74     for (int i = 0; i < d; i++) N[p[i]].f[1 << i] = 0;
75
76     for (int S = 0; S < 1 << d; S++) {
77         for (int s = (S - 1) & S; s; s = (s - 1) & S)
78             for (int i = 1; i <= n; i++) N[i].f[S] = std::min(N[i].f[S], N[i].f[s] + N[i].f[S ^ s]);
79         for (int i = 1; i <= n; i++) if (N[i].f[S] < LLONG_MAX >> 1ll) Dijkstra::q.emplace(&N[i], S, N[i].f[S]);
80     }
81     Dijkstra::dijkstra();
82 }
83
84 long long res = LLONG_MAX;
85 for (int i = 1; i <= n; i++) res = std::min(res, N[i].f[(1 << d) - 1]);
86 return res;
87 }
88
89 int main() {
90     int n, m, d;
91     scanf("%d %d %d", &n, &m, &d);
92     for (int i = 0, u, v, w; i < m; i++) {
93         scanf("%d %d %d", &u, &v, &w);
94         addEdge(u, v, w);
95     }
96
97     static int p[MAXN];
98     for (int i = 0; i < d; i++) scanf("%d", &p[i]);
99
100    long long ans = steiner(n, p, d);
101    printf("%lld\n", ans);
102
103    return 0;
104 }
```

3.4 全局最短路

3.4.1 Floyd

```
1 #include <cstdio>
2 #include <climits>
3 #include <algorithm>
4
5 const int MAXN = 1005;
6
7 int dist[MAXN][MAXN];
8
9 void floyd(int n) {
10     for (int k = 1; k <= n; k++) for (int i = 1; i <= n; i++) if (dist[i][k] < INT_MAX)
11         for (int j = 1; j <= n; j++) if (dist[k][j] < INT_MAX)
12             dist[i][j] = std::min(dist[i][j], dist[i][k] + dist[k][j]);
13 }
```

```

15 int main() {
16     int n, m;
17     scanf("%d %d", &n, &m);
18
19     for (int i = 1; i <= n; i++) std::fill(dist[i] + 1, dist[i] + n + 1, INT_MAX);
20
21     for (int i = 0, u, v, w; i < m; i++) {
22         scanf("%d %d %d", &u, &v, &w);
23         dist[u][v] = dist[v][u] = std::min(dist[u][v], w);
24     }
25
26     floyd(n);
27     // dist[i][i] is the length of minimum circle through i if it's a digraph
28
29     return 0;
30 }
```

3.4.2 Floyd 传递闭包

```

1 #include <cstdio>
2 #include <bitset>
3 #include <algorithm>
4
5 const int MAXN = 2005;
6
7 std::bitset<MAXN> G[MAXN];
8
9 void floyd(int n) {
10     for (int k = 0; k < n; k++) for (int i = 0; i < n; i++)
11         if (G[i][k]) G[i] |= G[k];
12 }
13
14 int main() {
15     int n, m;
16     scanf("%d %d", &n, &m);
17
18     for (int i = 0, u, v; i < m; i++) {
19         scanf("%d %d", &u, &v);
20         G[u].set(v);
21     }
22     for (int i = 0; i < n; i++) G[i].set(i);
23
24     floyd(n);
25
26     return 0;
27 }
```

3.4.3 Johnson

```

1 #include <cstdio>
2 #include <climits>
3 #include <queue>
4 #include <algorithm>
5
```

3 树与图

```
6 const int MAXN = 2005;
7 const int MAXM = 10005;
8
9 struct Edge;
10 struct Node {
11     Edge *e;
12     int dist[MAXN], cnt, h; // N[j].dist[i] denotes the shortest path from i to j
13     bool inq, vis;
14 } N[MAXN];
15
16 struct Edge {
17     Node *u, *v;
18     Edge *next;
19     int w;
20
21     Edge() {}
22     Edge(Node *u, Node *v, int w) : u(u), v(v), w(w), next(u->e) {}
23 } _pool[MAXM + MAXN], *_curr = _pool;
24
25 void addEdge(int u, int v, int w) {
26     N[u].e = new (_curr++) Edge(&N[u], &N[v], w);
27 }
28
29 namespace BellmanFord {
30     bool bellmanFord(Node *s, int n) {
31         std::queue<Node *> q;
32         q.push(s);
33         s->h = 0;
34
35         while (!q.empty()) {
36             Node *u = q.front();
37             q.pop();
38             u->inq = false;
39
40             for (Edge *e = u->e; e; e = e->next) {
41                 if (e->v->h > u->h + e->w) {
42                     e->v->h = u->h + e->w;
43
44                     if (++e->v->cnt >= n) return false;
45                     if (!e->v->inq) {
46                         e->v->inq = true;
47                         q.push(e->v);
48                     }
49                 }
50             }
51         }
52         return true;
53     }
54
55     void solve(int s, int n) {
56         for (int i = 0; i < n; i++) {
57             N[i].h = INT_MAX;
58             N[i].cnt = 0;
59             N[i].inq = false;
60         }
61     }
}
```

```

62         bellmanFord(&N[s], n);
63     }
64 } // namespace BellmanFord
65
66 namespace Dijkstra {
67     struct HeapNode {
68         Node *u;
69         int dist;
70
71         HeapNode(int dist, Node *u) : u(u), dist(dist) {}
72
73         bool operator<(const HeapNode &rhs) const {
74             return dist > rhs.dist;
75         }
76     };
77
78     void dijkstra(Node *s, int id) {
79         std::priority_queue<HeapNode> q;
80         s->dist[id] = 0;
81         q.emplace(0, s);
82
83         while (!q.empty()) {
84             Node *u = q.top().u;
85             q.pop();
86
87             if (u->vis) continue;
88             u->vis = true;
89
90             for (Edge *e = u->e; e; e = e->next) {
91                 if (e->v->dist[id] > u->dist[id] + e->w) {
92                     e->v->dist[id] = u->dist[id] + e->w;
93
94                     q.emplace(e->v->dist[id], e->v);
95                 }
96             }
97         }
98     }
99
100    void solve(int s, int n) {
101        for (int i = 1; i <= n; i++) {
102            N[i].dist[s] = INT_MAX;
103            N[i].vis = false;
104        }
105
106        dijkstra(&N[s], s);
107    }
108 } // namespace Dijkstra
109
110 int main() {
111     int n, m;
112     scanf("%d %d", &n, &m);
113     for (int i = 0, u, v, w; i < m; i++) {
114         scanf("%d %d %d", &u, &v, &w);
115         addEdge(u, v, w);
116     }
117 }
```

3 树与图

```
118     for (int i = 1; i <= n; i++) addEdge(0, i, 0);
119     BellmanFord::solve(0, n + 1);
120     for (Edge *e = _pool; e < _curr - n; e++) e->w += e->u->h - e->v->h;
121     for (int i = 1; i <= n; i++) {
122         Dijkstra::solve(i, n);
123         for (int j = 1; j <= n; j++) N[j].dist[i] -= N[i].h - N[j].h;
124     }
125
126     return 0;
127 }
```

3.5 点分治

```
1 #include <cstdio>
2 #include <vector>
3 #include <stack>
4 #include <algorithm>
5
6 const int MAXN = 100005;
7
8 struct Node;
9 struct Edge;
10
11 struct Node {
12     Edge *e;
13     bool solved;
14     int size, max;
15 } N[MAXN];
16
17 struct Edge {
18     Node *u, *v;
19     Edge *next;
20
21     Edge() {}
22     Edge(Node *u, Node *v) : u(u), v(v), next(u->e) {}
23 } _pool[MAXN << 1], *_curr = _pool;
24
25 void addEdge(int u, int v) {
26     N[u].e = new (_curr++) Edge(&N[u], &N[v]);
27     N[v].e = new (_curr++) Edge(&N[v], &N[u]);
28 }
29
30 void dfs(Node *u, Node *fa, std::vector<Node *> &vec) {
31     u->size = 1;
32     u->max = 0;
33     vec.push_back(u);
34     for (Edge *e = u->e; e; e = e->next) if (!e->v->solved && e->v != fa) {
35         dfs(e->v, u, vec);
36         u->size += e->v->size;
37         u->max = std::max(u->max, e->v->size);
38     }
39 }
40
41 Node *center(Node *s) {
```

```

42     static std::vector<Node *> vec;
43     dfs(s, NULL, vec);
44
45     Node *res = NULL;
46     for (int i = 0; i < vec.size(); i++) {
47         vec[i]->max = std::max(vec[i]->max, s->size - vec[i]->size);
48         if (!res || res->max > vec[i]->max) res = vec[i];
49     }
50
51     return res;
52 }
53
54 int calc(Node *u, Node *fa = NULL) {
55     int res = 0;
56     for (Edge *e = u->e; e; e = e->next) if (e->v != fa && !e->v->solved) {
57         res += calc(e->v, u);
58         // do something...
59     }
60     // do something...
61     return res;
62 }
63
64 int solve() {
65     static std::stack<Node *> s;
66     s.push(&N[1]);
67
68     int ans = 0;
69     while (!s.empty()) {
70         Node *u = s.top();
71         s.pop();
72
73         Node *root = center(u);
74         root->solved = true;
75
76         ans += calc(root);
77
78         for (Edge *e = root->e; e; e = e->next) if (!e->v->solved) s.push(e->v);
79     }
80
81     return ans;
82 }
83
84 int main() {
85     int n;
86     scanf("%d", &n);
87
88     for (int i = 1, u, v; i < n; i++) {
89         scanf("%d %d", &u, &v);
90         addEdge(u, v);
91     }
92
93     int ans = solve();
94     printf("%d\n", ans);
95 }
```

3.6 树链剖分

```

1 #include <cstdio>
2 #include <climits>
3 #include <algorithm>
4
5 const int MAXN = 100005;
6
7 struct Node;
8 struct Edge;
9 struct Chain;
10
11 struct Node {
12     Edge *e;
13     Chain *c;
14     Node *max, *fa;
15     int dfn, dfnR, dep, size, val; // use dfnR when there're operations on subtrees
16 } N[MAXN];
17
18 struct Edge {
19     Node *u, *v;
20     Edge *next;
21
22     Edge() {}
23     Edge(Node *u, Node *v) : u(u), v(v), next(u->e) {}
24 } _poolE[MAXN << 1], *_currE = _poolE;
25 void addEdge(int u, int v) {
26     N[u].e = new (_currE++) Edge(&N[u], &N[v]);
27     N[v].e = new (_currE++) Edge(&N[v], &N[u]);
28 }
29
30 struct Chain {
31     Node *top;
32
33     Chain(Node *top = NULL) : top(top) {}
34 } _poolC[MAXN], *_currC = _poolC;
35
36 void dfs1(Node *u) {
37     u->size = 1;
38
39     for (Edge *e = u->e; e; e = e->next) if (e->v != u->fa) {
40         e->v->dep = u->dep + 1;
41         e->v->fa = u;
42
43         dfs1(e->v);
44
45         u->size += e->v->size;
46         if (!u->max || u->max->size < e->v->size) u->max = e->v;
47     }
48 }
49 void dfs2(Node *u, Node *fa = NULL) {
50     static int dfsClock = 0;
51     u->dfn = ++dfsClock;
52
53     if (!u->fa || u->fa->max != u) u->c = new (_currC++) Chain(u);
54     else u->c = u->fa->c;

```

```

55
56     if (u->max) dfs2(u->max);
57     for (Edge *e = u->e; e; e = e->next) if (e->v != u->fa && e->v != u->max) dfs2(e->v);
58
59     u->dfnR = dfsClock;
60 }
61 void split() {
62     N[1].dep = 1;
63     dfs1(&N[1]);
64     dfs2(&N[1]);
65 }
66
67 struct SegT {
68     struct Node {
69         int l, r;
70         Node *lc, *rc;
71         int max, tag;
72
73         Node() {}
74         Node(int pos, int val) : l(pos), r(pos), max(val), tag(0), lc(NULL), rc(NULL) {}
75         Node(Node *lc, Node *rc) : l(lc->l), r(rc->r), lc(lc), rc(rc), tag(0) {
76             maintain();
77         }
78
79         void add(int d) {
80             max += d;
81             tag += d;
82         }
83
84         void pushDown() {
85             if (tag) {
86                 lc->add(tag);
87                 rc->add(tag);
88                 tag = 0;
89             }
90         }
91
92         void maintain() {
93             max = std::max(lc->max, rc->max);
94         }
95
96         void update(int l, int r, int d) {
97             if (l > this->r || this->l > r) return;
98             if (l <= this->l && this->r <= r) {
99                 add(d);
100                return;
101            }
102            pushDown();
103            lc->update(l, r, d);
104            rc->update(l, r, d);
105            maintain();
106        }
107
108        int query(int l, int r) {
109            if (l > this->r || this->l > r) return INT_MIN;
110            if (l <= this->l && this->r <= r) return max;

```

3 树与图

```
111         pushDown();
112         return std::max(lc->query(l, r), rc->query(l, r));
113     }
114 } *_root, *_pool[MAXN << 1], *_curr;
115
116 SegT() : root(NULL), _curr(_pool) {}
117
118 Node *_build(int l, int r, int *a) {
119     if (l == r) return new (_curr++) Node(l, a[l]);
120     int mid = l + ((r - l) >> 1);
121     return new (_curr++) Node(_build(l, mid, a), _build(mid + 1, r, a));
122 }
123 void build(int l, int r, int *a) {
124     root = _build(l, r, a);
125 }
126
127 void update(int l, int r, int d) {
128     root->update(l, r, d);
129 }
130
131 int query(int l, int r) {
132     return root->query(l, r);
133 }
134 } segT;
135
136 void update(int a, int b, int d) {
137     Node *u = &N[a], *v = &N[b];
138
139     while (u->c != v->c) {
140         if (u->c->top->dep < v->c->top->dep) std::swap(u, v);
141         segT.update(u->c->top->dfn, u->dfn, d);
142         u = u->c->top->fa;
143     }
144
145     if (u->dep > v->dep) std::swap(u, v);
146     segT.update(u->dfn, v->dfn, d);
147 }
148
149 int query(int a, int b) {
150     Node *u = &N[a], *v = &N[b];
151     int res = INT_MIN;
152
153     while (u->c != v->c) {
154         if (u->c->top->dep < v->c->top->dep) std::swap(u, v);
155         res = std::max(res, segT.query(u->c->top->dfn, u->dfn));
156         u = u->c->top->fa;
157     }
158
159     if (u->dep > v->dep) std::swap(u, v);
160     res = std::max(res, segT.query(u->dfn, v->dfn));
161
162     return res;
163 }
164
165 int main() {
166     int n;
```

```

167     scanf("%d", &n);
168
169     for (int i = 1; i <= n; i++) scanf("%d", &N[i].val);
170
171     for (int i = 1, u, v; i < n; i++) {
172         scanf("%d %d", &u, &v);
173         addEdge(u, v);
174     }
175
176     split();
177
178     static int temp[MAXN];
179     for (int i = 1; i <= n; i++) temp[N[i].dfn] = N[i].val;
180     segT.build(1, n, temp);
181
182     int q;
183     scanf("%d", &q);
184
185     while (q--) {
186         int op, u, v;
187         scanf("%d %d %d", &op, &u, &v);
188         if (op == 1) {
189             int d;
190             scanf("%d", &d);
191             update(u, v, d);
192         } else printf("%d\n", query(u, v));
193     }
194
195     return 0;
196 }
```

3.7 DSU on Tree / 树上启发式合并

```

1 #include <cstdio>
2 #include <vector>
3 #include <algorithm>
4
5 const int MAXN = 100005;
6
7 struct Graph {
8     void addEdge(int u, int v) {
9         G[u].push_back(v);
10        G[v].push_back(u);
11    }
12
13    std::vector<int> G[MAXN];
14    long long ans[MAXN];
15    int col[MAXN];
16    int n;
17 } G;
18
19 class DSUT {
20 public:
21     void solve(Graph &G) {
```

3 树与图

```
22     this->G = &G;
23     findHeavy(1);
24     dfs(1);
25 }
26
27 private:
28     Graph *G;
29     int cnt[MAXN];
30     int size[MAXN], son[MAXN];
31     int skip, max;
32     long long sum;
33
34     void findHeavy(int u, int fa = -1) {
35         size[u] = 1;
36         for (int v : G->G[u]) if (v != fa) {
37             findHeavy(v, u);
38             size[u] += size[v];
39             if (size[v] > size[son[u]]) son[u] = v;
40         }
41     }
42
43     void edit(int u, int fa, int d) {
44         cnt[G->col[u]] += d;
45         if (d > 0) {
46             if (cnt[G->col[u]] > max) {
47                 sum = G->col[u];
48                 max = cnt[G->col[u]];
49             } else if (cnt[G->col[u]] == max) {
50                 sum += G->col[u];
51             }
52         }
53         for (int v : G->G[u]) if (v != fa && v != skip) edit(v, u, d);
54     }
55
56     void dfs(int u, int fa = -1, bool keep = false) {
57         for (int v : G->G[u]) if (v != fa && v != son[u]) dfs(v, u);
58         if (son[u]) {
59             dfs(son[u], u, true);
60             skip = son[u];
61         }
62         edit(u, fa, 1);
63         G->ans[u] = sum;
64         skip = -1;
65         if (!keep) {
66             edit(u, fa, -1);
67             sum = 0;
68             max = 0;
69         }
70     }
71 } dsut;
72
73 int main() {
74     int n;
75     scanf("%"d", &n);
76     G.n = n;
77 }
```

```

78     for (int i = 1; i <= n; i++) scanf("%d", &G.col[i]);
79     for (int i = 1, u, v; i < n; i++) {
80         scanf("%d %d", &u, &v);
81         G.addEdge(u, v);
82     }
83
84     dsut.solve(G);
85
86     for (int i = 1; i <= n; i++) printf("%lld%c", G.ans[i], " \n"[i == n]);
87
88     return 0;
89 }
```

3.8 网络流

3.8.1 Dinic

```

1 #include <cstdio>
2 #include <climits>
3 #include <queue>
4 #include <vector>
5 #include <algorithm>
6
7 const int MAXN = 105;
8 const int MAXM = 5005;
9
10 struct Edge;
11 struct Node;
12
13 struct Node {
14     std::vector<Edge> e; // use Edge* when sparse graph
15     Edge *curr;
16     int level;
17 } N[MAXN];
18
19 struct Edge {
20     Node *u, *v;
21     int cap, flow, rev;
22
23     Edge(Node *u, Node *v, int cap, int rev) : u(u), v(v), cap(cap), flow(0), rev(rev) {}
24 };
25
26 void addEdge(int u, int v, int cap) {
27     N[u].e.emplace_back(&N[u], &N[v], cap, N[v].e.size());
28     N[v].e.emplace_back(&N[v], &N[u], 0, N[u].e.size() - 1);
29 }
30
31 namespace Dinic {
32     bool level(Node *s, Node *t, int n) {
33         for (int i = 0; i < n; i++) N[i].level = 0;
34         static std::queue<Node *> q;
35         q.push(s);
36         s->level = 1;
37         while (!q.empty()) {
```

3 树与图

```
38         Node *u = q.front();
39         q.pop();
40
41         for (Edge *e = &u->e.front(); e && e <= &u->e.back(); e++) {
42             if (e->cap > e->flow && e->v->level == 0) {
43                 e->v->level = u->level + 1;
44                 q.push(e->v);
45             }
46         }
47     }
48     return t->level;
49 }
50
51 int findPath(Node *u, Node *t, int limit = INT_MAX) {
52     if (u == t) return limit;
53     int res = 0;
54     for (Edge *&e = u->curr; e && e <= &u->e.back(); e++) {
55         if (e->cap > e->flow && e->v->level == u->level + 1) {
56             int flow = findPath(e->v, t, std::min(limit, e->cap - e->flow));
57             if (flow > 0) {
58                 e->flow += flow;
59                 e->v->e[e->rev].flow -= flow;
60                 limit -= flow;
61                 res += flow;
62                 if (limit <= 0) return res;
63             } else e->v->level = -1;
64         }
65     }
66     return res;
67 }
68
69 long long solve(int s, int t, int n) {
70     long long res = 0;
71     while (level(&N[s], &N[t], n)) {
72         for (int i = 0; i < n; i++) N[i].curr = &N[i].e.front();
73         int flow;
74         while ((flow = findPath(&N[s], &N[t])) > 0) res += flow;
75     }
76     return res;
77 }
78 }
79
80 int main() {
81     int n, m, s, t;
82     scanf("%d %d %d %d", &n, &m, &s, &t);
83
84     for (int i = 0, u, v, w; i < m; i++) {
85         scanf("%d %d %d", &u, &v, &w);
86         addEdge(u, v, w);
87     }
88
89     printf("%lld\n", Dinic::solve(s, t, n));
90
91     return 0;
92 }
```

3.8.2 ISAP

```

1 #include <cstdio>
2 #include <climits>
3 #include <algorithm>
4
5 const int MAXN = 105;
6 const int MAXM = 5005;
7
8 struct Edge;
9 struct Node {
10     Edge *e, *curr; // use std::vector<Edge> when dense graph
11     int dist;
12 } N[MAXN];
13
14 struct Edge {
15     Node *u, *v;
16     Edge *next, *rev;
17     int cap, flow;
18
19     Edge() {}
20     Edge(Node *u, Node *v, int cap) : u(u), v(v), cap(cap), flow(0), next(u->e) {}
21 } _pool[MAXM << 1], *_curr = _pool;
22
23 void addEdge(int u, int v, int cap) {
24     N[u].e = new (_curr++) Edge(&N[u], &N[v], cap);
25     N[v].e = new (_curr++) Edge(&N[v], &N[u], 0);
26     N[u].e->rev = N[v].e;
27     N[v].e->rev = N[u].e;
28 }
29
30 namespace ISAP {
31     int cnt[MAXN], n;
32     Node *s, *t;
33
34     int flow(Node *u, int limit = INT_MAX) {
35         if (u == t) return limit;
36
37         int temp = limit;
38         for (Edge *&e = u->curr; e; e = e->next) if (u->dist == e->v->dist + 1 && e->cap > e->flow) {
39             int f = flow(e->v, std::min(temp, e->cap - e->flow));
40             e->flow += f;
41             e->rev->flow -= f;
42             temp -= f;
43             if (!temp) return limit;
44         }
45
46         if (!(--cnt[u->dist++])) s->dist = n + 1;
47         ++cnt[u->dist];
48         u->curr = u->e;
49
50         return limit - temp;
51     }
52
53     long long solve(int s, int t, int n) {
54         ISAP::n = n;

```

```

55     ISAP::s = &N[s];
56     ISAP::t = &N[t];
57     for (int i = 1; i <= n; i++) {
58         N[i].curr = N[i].e;
59         N[i].dist = 0;
60         cnt[i] = 0;
61     }
62     cnt[0] = n;
63
64     long long res = 0;
65     while (N[s].dist <= n) res += flow(&N[s]);
66     return res;
67 }
68 }
69
70 int main() {
71     int n, m, s, t;
72     scanf("%d %d %d %d", &n, &m, &s, &t);
73
74     for (int i = 0, u, v, c; i < m; i++) {
75         scanf("%d %d %d", &u, &v, &c);
76         addEdge(u, v, c);
77     }
78
79     long long ans = ISAP::solve(s, t, n);
80     printf("%lld\n", ans);
81
82     return 0;
83 }
```

3.8.3 最小割输出方案

```

1 std::vector<Edge> cuts;
2 void bfs(int s, int t, int n) {
3     static std::queue<Node *> q;
4     while (!q.empty()) q.pop();
5     q.push(&N[s]);
6     N[s].vis = true;
7
8     while (!q.empty()) {
9         Node *u = q.front();
10        q.pop();
11
12        for (Edge e : u->e) if (e.cap > e.flow && !e.v->vis) {
13            e.v->vis = true;
14            q.push(e.v);
15        }
16    }
17
18    for (int i = 0; i < n; i++) if (N[i].vis) for (Edge e : N[i].e) {
19        if (e.cap == e.flow && e.cap > 0 && !e.v->vis) {
20            cust.push_back(e);
21        }
22    }
23 }
```

3.9 Gomory-Hu Tree

```

1 #include <cstdio>
2 #include <climits>
3 #include <vector>
4 #include <queue>
5 #include <algorithm>
6
7 const int MAXN = 3005;
8 const int MAXN_LOG = 13;
9
10 struct Edge;
11 struct Node {
12     std::vector<Edge> e;
13     Edge *curr;
14     int level;
15     bool vis;
16 } N[MAXN];
17
18 struct Edge {
19     Node *u, *v;
20     int cap, flow, rev;
21
22     Edge(Node *u, Node *v, int cap, int rev) : u(u), v(v), cap(cap), flow(0), rev(rev) {}
23 };
24
25 bool G[MAXN][MAXN];
26
27 void addEdge(int u, int v, int cap) {
28     N[u].e.emplace_back(&N[u], &N[v], cap, N[v].e.size());
29     N[v].e.emplace_back(&N[v], &N[u], cap, N[u].e.size() - 1);
30     G[u][v] = G[v][u] = true;
31 }
32
33 namespace Dinic {
34     bool level(Node *s, Node *t, int n) {
35         for (int i = 1; i <= n; i++) N[i].level = 0;
36         static std::queue<Node *> q;
37         q.push(s);
38         s->level = 1;
39         while (!q.empty()) {
40             Node *u = q.front();
41             q.pop();
42
43             for (Edge &e : u->e) {
44                 if (e.cap > e.flow && e.v->level == 0) {
45                     e.v->level = u->level + 1;
46                     q.push(e.v);
47                 }
48             }
49         }
50         return t->level;
51     }
52
53     int findPath(Node *u, Node *t, int limit = INT_MAX) {
54         if (u == t) return limit;

```

3 树与图

```
55     int res = 0;
56     for (Edge *&e = u->curr; e && e <= &u->e.back(); e++) {
57         if (e->cap > e->flow && e->v->level == u->level + 1) {
58             int flow = findPath(e->v, t, std::min(limit, e->cap - e->flow));
59             if (flow > 0) {
60                 e->flow += flow;
61                 e->v->e[e->rev].flow -= flow;
62                 limit -= flow;
63                 res += flow;
64                 if (!limit) return res;
65             } else e->v->level = -1;
66         }
67     }
68     return res;
69 }
70
71 int solve(int s, int t, int n) {
72     for (int i = 1; i <= n; i++) for (Edge &e : N[i].e) e.flow = 0;
73
74     int res = 0;
75     while (level(&N[s], &N[t], n)) {
76         for (int i = 1; i <= n; i++) N[i].curr = &N[i].e.front();
77         int flow;
78         while ((flow = findPath(&N[s], &N[t])) > 0) res += flow;
79     }
80     return res;
81 }
82
83 void bfs(Node *s, int n) {
84     for (int i = 1; i <= n; i++) N[i].vis = false;
85
86     static std::queue<Node *> q;
87     q.push(s);
88     s->vis = true;
89     while (!q.empty()) {
90         Node *u = q.front();
91         q.pop();
92         for (const Edge &e : u->e) {
93             if (e.cap > e.flow && !e.v->vis) {
94                 e.v->vis = true;
95                 q.push(e.v);
96             }
97         }
98     }
99 }
100 }
101
102 namespace GomoryHuTree {
103     struct Edge;
104     struct Node {
105         Edge *e;
106         Node *f[MAXN_LOG];
107         int dep, min[MAXN_LOG];
108     } N[MAXN];
109
110     struct Edge {
```

```

111     Node *u, *v;
112     Edge *next;
113     int w;
114
115     Edge() {}
116     Edge(Node *u, Node *v, int w) : u(u), v(v), next(u->e), w(w) {}
117 } _pool[MAXN << 1], *_curr;
118
119 void addEdge(int u, int v, int w) {
120     N[u].e = new (_curr++) Edge(&N[u], &N[v], w);
121     N[v].e = new (_curr++) Edge(&N[v], &N[u], w);
122 }
123
124 void dfs(Node *u, Node *fa = NULL) {
125     u->f[0] = (fa ? fa : u);
126     u->dep = (fa ? fa->dep : 0) + 1;
127     for (int i = 1; i < MAXN_LOG; i++) {
128         u->f[i] = u->f[i - 1]->f[i - 1];
129         u->min[i] = std::min(u->min[i - 1], u->f[i - 1]->min[i - 1]);
130     }
131
132     for (Edge *e = u->e; e; e = e->next) if (e->v != fa) {
133         e->v->min[0] = e->w;
134         dfs(e->v, u);
135     }
136 }
137
138 int query(Node *u, Node *v) {
139     if (u->dep < v->dep) std::swap(u, v);
140     int res = INT_MAX;
141
142     for (int i = MAXN_LOG - 1; ~i; i--) if (u->f[i]->dep >= v->dep) {
143         res = std::min(res, u->min[i]);
144         u = u->f[i];
145     }
146
147     for (int i = MAXN_LOG - 1; ~i; i--) if (u->f[i] != v->f[i]) {
148         res = std::min({res, u->min[i], v->min[i]});
149         u = u->f[i];
150         v = v->f[i];
151     }
152
153     if (u != v) res = std::min({res, u->min[0], v->min[0]});
154
155     return res;
156 }
157 int query(int u, int v) { return query(&N[u], &N[v]); }
158
159 void build(const std::vector<int> &nodes, int n) {
160     if (nodes.size() <= 1) return;
161     int s = nodes[0], t = nodes[1];
162     int flow = Dinic::solve(s, t, n);
163     addEdge(s, t, flow);
164
165     Dinic::bfs(&N[s], n);
166

```

```
167     std::vector<int> ln, rn;
168     for (int u : nodes) {
169         if (::N[u].vis) ln.push_back(u);
170         else rn.push_back(u);
171     }
172
173     build(ln, n);
174     build(rn, n);
175 }
176
177 void build(int n) {
178     _curr = _pool;
179     std::vector<int> vec(n);
180     for (int i = 1; i <= n; i++) vec[i - 1] = i;
181     build(vec, n);
182
183     N[1].min[0] = INT_MAX;
184     dfs(&N[1]);
185 }
186 }
187
188 int main() {
189     int n, m;
190     scanf("%d %d", &n, &m);
191     for (int i = 0, u, v, w; i < m; i++) {
192         scanf("%d %d %d", &u, &v, &w);
193         addEdge(u, v, w);
194     }
195
196     GomoryHuTree::build(n);
197
198     int q;
199     scanf("%d", &q);
200     while (q--) {
201         int u, v;
202         scanf("%d %d", &u, &v);
203
204         int ans = GomoryHuTree::query(u, v);
205         printf("%d\n", ans);
206     }
207
208     return 0;
209 }
```

3.10 费用流

3.10.1 Dijkstra 费用流

```
1 #include <cstdio>
2 #include <climits>
3 #include <vector>
4 #include <queue>
5 #include <algorithm>
6
```

```

7  const int MAXN = 405;
8
9  struct Edge;
10 struct Node;
11
12 struct Node {
13     std::vector<Edge> e;
14     Edge *pre;
15     int flow, h, dist;
16 } N[MAXN];
17
18 struct Edge {
19     Node *u, *v;
20     int cap, flow, cost, rev;
21
22     Edge(Node *u, Node *v, int cap, int cost, int rev) : u(u), v(v), rev(rev), cap(cap), flow(0), cost(cost)
23         {}
24 };
25 void addEdge(int u, int v, int cap, int cost) {
26     N[u].e.push_back(Edge(&N[u], &N[v], cap, cost, N[v].e.size()));
27     N[v].e.push_back(Edge(&N[v], &N[u], 0, -cost, N[u].e.size() - 1));
28 }
29
30 namespace EdmondsKarp {
31     struct HeapNode {
32         Node *u;
33         int dist;
34
35         HeapNode(Node *u, int dist) : u(u), dist(dist) {}
36
37         bool operator<(const HeapNode &rhs) const {
38             return dist > rhs.dist;
39         }
40     };
41
42     void solve(int s, int t, int n, int &flow, int &cost) {
43         flow = cost = 0;
44         // if there exists negative cost, run bellman-ford on h[]
45         while (true) {
46             for (int i = 1; i <= n; i++) {
47                 N[i].dist = INT_MAX;
48                 N[i].flow = 0;
49                 N[i].pre = NULL;
50             }
51
52             std::priority_queue<HeapNode> q;
53             q.push(HeapNode(&N[s], 0));
54
55             N[s].dist = 0;
56             N[s].flow = INT_MAX;
57
58             while (!q.empty()) {
59                 HeapNode un = q.top();
60                 q.pop();
61                 Node *u = un.u;

```

3 树与图

```
62
63     if (u->dist != un.dist) continue;
64
65     for (Edge &e : u->e) {
66         int newCost = e.cost + u->h - e.v->h;
67         if (e.cap > e.flow && e.v->dist > u->dist + newCost) {
68             e.v->dist = u->dist + newCost;
69             e.v->flow = std::min(u->flow, e.cap - e.flow);
70             e.v->pre = &e;
71
72             q.push(HeapNode(e.v, e.v->dist));
73         }
74     }
75 }
76
77 if (N[t].dist == INT_MAX) break; // minimum cost maximum flow
78 // if (N[t].dist + N[t].h > 0) break; // minimum cost available flow
79
80 for (int i = 1; i <= n; i++) N[i].h = std::min(N[i].h + N[i].dist, INT_MAX >> 1);
81
82 for (Edge *e = N[t].pre; e; e = e->u->pre) {
83     e->flow += N[t].flow;
84     e->v->e[e->rev].flow -= N[t].flow;
85 }
86
87 flow += N[t].flow;
88 cost += N[t].h * N[t].flow;
89 }
90 }
91 }
92
93 int main() {
94     int n, m;
95     scanf("%d %d", &n, &m);
96
97     for (int i = 0, u, v, c, w; i < m; i++) {
98         scanf("%d %d %d %d", &u, &v, &c, &w);
99         addEdge(u, v, c, w);
100    }
101
102    int flow, cost;
103    EdmondsKarp::solve(1, n, n, flow, cost);
104    printf("%d %d\n", flow, cost);
105
106    return 0;
107 }
```

3.10.2 ZKW 费用流

```
1 #include <cstdio>
2 #include <climits>
3 #include <queue>
4 #include <algorithm>
5
6 const int MAXN = 405;
```

```

7
8 struct Graph {
9     struct Edge {
10         int v, cap, flow, cost, rev;
11
12         Edge(int v, int cap, int cost, int rev) : v(v), cap(cap), flow(0), cost(cost), rev(rev) {}
13     };
14
15     void addEdge(int u, int v, int cap, int cost) {
16         G[u].emplace_back(v, cap, cost, G[v].size());
17         G[v].emplace_back(u, 0, -cost, G[u].size() - 1);
18     }
19
20     std::vector<Edge> G[MAXN];
21     int n;
22 } G;
23
24 class MCMF {
25 public:
26     std::pair<int, int> solve(int s, int t, Graph &G) {
27         this->s = s;
28         this->t = t;
29         this->G = &G;
30
31         int flow = 0, cost = 0;
32         int rem = INT_MAX;
33         while (argue()) {
34             std::fill_n(ptr, G.n, 0);
35             int d = dfs(s, rem - flow);
36             flow += d;
37             cost += d * dist[t];
38         }
39
40         return std::make_pair(flow, cost);
41     }
42
43 private:
44     int s, t;
45     Graph *G;
46     int dist[MAXN], ptr[MAXN];
47     bool vis[MAXN];
48
49     bool argue() {
50         std::fill_n(dist, G->n, INT_MAX);
51         std::fill_n(vis, G->n, false);
52         dist[s] = 0;
53
54         std::queue<int> q;
55         q.push(s);
56         while (!q.empty()) {
57             int u = q.front();
58             q.pop();
59
60             vis[u] = false;
61             for (auto &e : G->G[u]) {
62                 if (e.cap > e.flow && dist[e.v] > dist[u] + e.cost) {

```

```

63             dist[e.v] = dist[u] + e.cost;
64             if (!vis[e.v]) {
65                 vis[e.v] = true;
66                 q.push(e.v);
67             }
68         }
69     }
70 }
71
72     return dist[t] != INT_MAX;
73 }
74
75     int dfs(int u, int r) {
76         if (u == t) return r;
77
78         vis[u] = true;
79         int res = 0;
80
81         for (int &i = ptr[u]; i < G->G[u].size(); i++) {
82             auto &e = G->G[u][i];
83             if (e.cap > e.flow && dist[e.v] == dist[u] + e.cost && !vis[e.v]) {
84                 int d = dfs(e.v, std::min(r - res, e.cap - e.flow));
85                 res += d;
86                 e.flow += d;
87                 G->G[e.v][e.rev].flow -= d;
88                 if (res == r) {
89                     vis[u] = false;
90                     break;
91                 }
92             }
93         }
94
95         return res;
96     }
97 } mcmf;
98
99     int main() {
100         int n, m, s, t;
101         scanf("%d %d %d %d", &n, &m, &s, &t);
102
103         G.n = n;
104         for (int i = 0, u, v, c, w; i < m; i++) {
105             scanf("%d %d %d %d", &u, &v, &c, &w);
106             G.addEdge(u, v, c, w);
107         }
108
109         auto [flow, cost] = mcmf.solve(s, t, G);
110         printf("%d %d\n", flow, cost);
111
112         return 0;
113     }

```

3.11 上下界网络流

3.11.1 无源无汇上下界可行流

```

1 #include <iostream>
2 #include <climits>
3 #include <vector>
4 #include <queue>
5 #include <algorithm>
6
7 const int MAXN = 205;
8 const int MAXM = 10205;
9
10 struct Edge;
11 struct Node;
12
13 struct Node {
14     std::vector<Edge> e;
15     Edge *curr;
16     int level, extra;
17 } N[MAXN];
18
19 struct Edge {
20     Node *u, *v;
21     int cap, flow, rev;
22
23     Edge(Node *u, Node *v, int cap, int rev)
24         : u(u), v(v), cap(cap), flow(0), rev(rev) {}
25 };
26
27 struct Pair {
28     int u, num, lower;
29
30     Pair() {}
31     Pair(int u, int num, int lower) : u(u), num(num), lower(lower) {}
32
33     int flow() const {
34         return N[u].e[num].flow + lower;
35     }
36 } E[MAXM];
37
38 Pair addEdge(int u, int v, int lower, int upper) {
39     int cap = upper - lower;
40     N[u].e.push_back(Edge(&N[u], &N[v], cap, N[v].e.size()));
41     N[v].e.push_back(Edge(&N[v], &N[u], 0, N[u].e.size() - 1));
42
43     N[u].extra -= lower;
44     N[v].extra += lower;
45
46     return Pair(u, N[u].e.size() - 1, lower);
47 }
48
49 namespace Dinic {
50     bool level(Node *s, Node *t, int n) {
51         for (int i = 0; i < n; i++) N[i].level = 0;
52         static std::queue<Node *> q;

```

3 树与图

```
53     q.push(s);
54     s->level = 1;
55     while (!q.empty()) {
56         Node *u = q.front();
57         q.pop();
58         for (Edge *e = &u->e.front(); e <= &u->e.back(); e++) {
59             if (e->cap > e->flow && e->v->level == 0) {
60                 e->v->level = u->level + 1;
61                 q.push(e->v);
62             }
63         }
64     }
65     return t->level;
66 }

67
68 int findPath(Node *u, Node *t, int limit = INT_MAX) {
69     if (u == t) return limit;
70     int res = 0;
71     for (Edge *&e = u->curr; e <= &u->e.back(); e++) {
72         if (e->cap > e->flow && e->v->level == u->level + 1) {
73             int flow = findPath(e->v, t, std::min(limit, e->cap - e->flow));
74             if (flow > 0) {
75                 e->flow += flow;
76                 e->v->e[e->rev].flow -= flow;
77                 limit -= flow;
78                 res += flow;
79                 if (limit <= 0) return res;
80             } else e->v->level = -1;
81         }
82     }
83     return res;
84 }

85
86 int solve(int s, int t, int n) {
87     int res = 0;
88     while (level(&N[s], &N[t], n)) {
89         for (int i = 0; i < n; i++) N[i].curr = &N[i].e.front();
90         int flow;
91         while ((flow = findPath(&N[s], &N[t])) > 0) res += flow;
92     }
93     return res;
94 }
95 }

96
97 int main() {
98     int n, m;
99     scanf("%d %d", &n, &m);
100    const int s = 0, t = n + 1;
101
102    for (int i = 0, u, v, lower, upper; i < m; i++) {
103        scanf("%d %d %d %d", &u, &v, &lower, &upper);
104        E[i] = addEdge(u, v, lower, upper);
105    }
106
107    int sum = 0;
108    for (int i = 1; i <= n; i++) {
```

```

109         if (N[i].extra > 0) {
110             sum += N[i].extra;
111             addEdge(s, i, 0, N[i].extra);
112         } else if (N[i].extra < 0) {
113             addEdge(i, t, 0, -N[i].extra);
114         }
115     }
116
117     int maxFlow = Dinic::solve(s, t, n + 2);
118     if (maxFlow < sum) {
119         puts("NO");
120     } else {
121         puts("YES");
122         for (int i = 0; i < m; i++) printf("%d\n", E[i].flow());
123     }
124
125     return 0;
126 }
```

3.11.2 有源有汇上下界最大流

若要求「有源有汇上下界费用流」，将网络流改为 Edmonds-Karp 即可，最后费用是两次的费用和。

```

1 #include <iostream>
2 #include <climits>
3 #include <vector>
4 #include <queue>
5 #include <algorithm>
6
7 const int MAXN = 205;
8 const int MAXM = 10000;
9
10 struct Edge;
11 struct Node;
12
13 struct Node {
14     std::vector<Edge> e;
15     Edge *curr;
16     int level, extra;
17 } N[MAXN];
18
19 struct Edge {
20     Node *u, *v;
21     int cap, flow, rev;
22
23     Edge(Node *u, Node *v, int cap, int rev) : u(u), v(v), cap(cap), flow(0), rev(rev) {}
24 };
25
26 void addEdge(int u, int v, int lower, int upper) {
27     int cap = upper - lower;
28     N[u].e.push_back(Edge(&N[u], &N[v], cap, N[v].e.size()));
29     N[v].e.push_back(Edge(&N[v], &N[u], 0, N[u].e.size() - 1));
30
31     N[u].extra -= lower;
32     N[v].extra += lower;
33 }
```

3 树与图

```
34
35 namespace Dinic {
36     bool level(Node *s, Node *t, int n) {
37         for (int i = 0; i < n; i++) N[i].level = 0;
38         static std::queue<Node *> q;
39         q.push(s);
40         s->level = 1;
41         while (!q.empty()) {
42             Node *u = q.front();
43             q.pop();
44             for (Edge *e = &u->e.front(); e <= &u->e.back(); e++) {
45                 if (e->cap > e->flow && e->v->level == 0) {
46                     e->v->level = u->level + 1;
47                     q.push(e->v);
48                 }
49             }
50         }
51         return t->level;
52     }
53
54     int findPath(Node *u, Node *t, int limit = INT_MAX) {
55         if (u == t) return limit;
56         int res = 0;
57         for (Edge *&e = u->curr; e <= &u->e.back(); e++) {
58             if (e->cap > e->flow && e->v->level == u->level + 1) {
59                 int flow = findPath(e->v, t, std::min(limit, e->cap - e->flow));
60                 if (flow > 0) {
61                     e->flow += flow;
62                     e->v->e[e->rev].flow -= flow;
63                     limit -= flow;
64                     res += flow;
65                     if (limit <= 0) return res;
66                 } else e->v->level = -1;
67             }
68         }
69         return res;
70     }
71
72     int solve(int s, int t, int n) {
73         int res = 0;
74         while (level(&N[s], &N[t], n)) {
75             for (int i = 0; i < n; i++) N[i].curr = &N[i].e.front();
76             int flow;
77             while ((flow = findPath(&N[s], &N[t])) > 0) res += flow;
78         }
79         return res;
80     }
81 }
82
83 int main() {
84     int n, m, s, t;
85     scanf("%d %d %d %d", &n, &m, &s, &t);
86     const int S = 0, T = n + 1;
87
88     for (int i = 0, u, v, lower, upper; i < m; i++) {
89         scanf("%d %d %d %d", &u, &v, &lower, &upper);
```

```

90         addEdge(u, v, lower, upper);
91     }
92     addEdge(t, s, 0, INT_MAX);
93     int sum = 0;
94     for (int i = 1; i <= n; i++) {
95         if (N[i].extra > 0) {
96             sum += N[i].extra;
97             addEdge(S, i, 0, N[i].extra);
98         } else if (N[i].extra < 0) {
99             addEdge(i, T, 0, -N[i].extra);
100        }
101    }
102
103    int maxFlow = Dinic::solve(S, T, n + 2);
104    if (maxFlow < sum) {
105        puts("No");
106    } else {
107        puts("Yes");
108        printf("%d\n", Dinic::solve(s, t, n + 2));
109    }
110
111    return 0;
112 }
```

3.11.3 有源有汇上下界最小流

```

1 #include <cstdio>
2 #include <climits>
3 #include <vector>
4 #include <queue>
5 #include <algorithm>
6
7 const int MAXN = 50005;
8 const int MAXM = 125005;
9
10 struct Edge;
11 struct Node;
12
13 struct Node {
14     Edge *e, *curr;
15     int level, extra;
16 } N[MAXN];
17
18 struct Edge {
19     Node *u, *v;
20     Edge *next, *rev;
21     int cap, flow;
22
23     Edge() {}
24     Edge(Node *u, Node *v, int cap) : u(u), v(v), cap(cap), flow(0), next(u->e) {}
25 } _pool[MAXM + MAXN << 1], *_curr = _pool;
26
27 Edge *addEdge(int u, int v, int lower, int upper) {
28     int cap = upper - lower;
29     N[u].e = new (_curr++) Edge(&N[u], &N[v], cap);
```

3 树与图

```
30     N[v].e = new (_curr++) Edge(&N[v], &N[u], 0);
31     (N[u].e->rev = N[v].e)->rev = N[u].e;
32
33     N[u].extra -= lower;
34     N[v].extra += lower;
35
36     return N[u].e;
37 }
38
39 namespace Dinic {
40     bool level(Node *s, Node *t, int n) {
41         for (int i = 0; i < n; i++) N[i].level = 0;
42         static std::queue<Node *> q;
43         q.push(s);
44         s->level = 1;
45         while (!q.empty()) {
46             Node *u = q.front();
47             q.pop();
48             for (Edge *e = u->e; e; e = e->next) {
49                 if (e->cap > e->flow && e->v->level == 0) {
50                     e->v->level = u->level + 1;
51                     q.push(e->v);
52                 }
53             }
54         }
55         return t->level;
56     }
57
58     int findPath(Node *u, Node *t, int limit = INT_MAX) {
59         if (u == t) return limit;
60         int res = 0;
61         for (Edge *&e = u->curr; e; e = e->next) {
62             if (e->cap > e->flow && e->v->level == u->level + 1) {
63                 int flow = findPath(e->v, t, std::min(limit, e->cap - e->flow));
64                 if (flow > 0) {
65                     e->flow += flow;
66                     e->rev->flow -= flow;
67                     limit -= flow;
68                     res += flow;
69                     if (limit <= 0) return res;
70                 } else e->v->level = -1;
71             }
72         }
73         return res;
74     }
75
76     int solve(int s, int t, int n) {
77         int res = 0;
78         while (level(&N[s], &N[t], n)) {
79             for (int i = 0; i < n; i++) N[i].curr = N[i].e;
80             int flow;
81             while ((flow = findPath(&N[s], &N[t])) > 0) res += flow;
82         }
83         return res;
84     }
85 }
```

```

86
87 int main() {
88     int n, m, s, t;
89     scanf("%d %d %d %d", &n, &m, &s, &t);
90     const int S = 0, T = n + 1;
91
92     for (int i = 0, u, v, lower, upper; i < m; i++) {
93         scanf("%d %d %d %d", &u, &v, &lower, &upper);
94         addEdge(u, v, lower, upper);
95     }
96     Edge *e = addEdge(t, s, 0, INT_MAX);
97     int sum = 0;
98     for (int i = 1; i <= n; i++) {
99         if (N[i].extra > 0) {
100             sum += N[i].extra;
101             addEdge(S, i, 0, N[i].extra);
102         } else if (N[i].extra < 0) {
103             addEdge(i, T, 0, -N[i].extra);
104         }
105     }
106
107     int maxFlow = Dinic::solve(S, T, n + 2);
108     if (maxFlow < sum) {
109         puts("No");
110     } else {
111         puts("Yes");
112         int flow = e->flow;
113         e->cap = e->rev->cap = 0;
114         printf("%d\n", flow - Dinic::solve(t, s, n + 2));
115     }
116
117     return 0;
118 }
```

3.12 Tarjan

3.12.1 有向图找强连通分量并缩点建图

```

1 #include <cstdio>
2 #include <stack>
3 #include <algorithm>
4
5 const int MAXN = 100005;
6 const int MAXM = 500005;
7
8 template <typename T>
9 struct Edge {
10     T *u, *v;
11     Edge *next;
12
13     Edge() {}
14     Edge(T *u, T *v) : u(u), v(v), next(u->e) {}
15 };
16
```

3 树与图

```
17 struct Comp {
18     Edge<Comp> *e;
19     int size;
20 } C[MAXN];
21 Edge<Comp> _poolC[MAXM], *_currC = _poolC;
22
23 void addEdge(Comp *u, Comp *v) {
24     u->e = new (_currC++) Edge<Comp>(u, v);
25 }
26
27 struct Node {
28     Edge<Node> *e;
29     Comp *c;
30     int dfn, low;
31     bool ins;
32 } N[MAXN];
33 Edge<Node> _poolN[MAXM], *_currN = _poolN;
34
35 void addEdge(int u, int v) {
36     N[u].e = new (_currN++) Edge<Node>(&N[u], &N[v]);
37 }
38
39 namespace Tarjan {
40     int dfsClock, compCnt;
41     std::stack<Node *> s;
42
43     void dfs(Node *u) {
44         u->dfn = u->low = ++dfsClock;
45         s.push(u);
46         u->ins = true;
47
48         for (Edge<Node> *e = u->e; e; e = e->next) {
49             if (!e->v->dfn) {
50                 dfs(e->v);
51                 u->low = std::min(u->low, e->v->low);
52             } else if (e->v->ins) {
53                 u->low = std::min(u->low, e->v->dfn);
54             }
55         }
56
57         if (u->dfn == u->low) {
58             Comp *c = &C[++compCnt];
59             Node *v;
60
61             do {
62                 v = s.top();
63                 s.pop();
64                 v->ins = false;
65                 v->c = c;
66                 c->size++;
67             } while (u != v);
68         }
69     }
70
71     void findSCC(int n) {
72         dfsClock = compCnt = 0;
```

```

73     while (!s.empty()) s.pop();
74
75     for (int i = 1; i <= n; i++) if (!N[i].dfn) dfs(&N[i]);
76 }
77
78 void rebuild() {
79     for (Edge<Node> *e = _poolN; e != _currN; e++) if (e->u->c != e->v->c) addEdge(e->u->c, e->v->c);
80 }
81 }
82
83 int main() {
84     int n, m;
85     scanf("%d %d", &n, &m);
86
87     for (int i = 0, u, v; i < m; i++) {
88         scanf("%d %d", &u, &v);
89         addEdge(u, v);
90     }
91
92     Tarjan::findSCC(n);
93     Tarjan::rebuild();
94
95     return 0;
96 }
```

3.12.2 无向图找边双连通分量并缩点建图

```

1 #include <cstdio>
2 #include <vector>
3 #include <queue>
4 #include <algorithm>
5
6 const int MAXN = 300005;
7
8 template <typename T>
9 struct Edge {
10     T *u, *v;
11     Edge *next;
12
13     Edge() {}
14     Edge(T *u, T *v) : u(u), v(v), next(u->e) {}
15 };
16
17 struct Comp {
18     Edge<Comp> *e;
19     int dist;
20 } C[MAXN];
21 Edge<Comp> _poolC[MAXN << 1], *_currC = _poolC;
22 void addEdgeC(int u, int v) {
23     C[u].e = new (_currC++) Edge<Comp>(&C[u], &C[v]);
24     C[v].e = new (_currC++) Edge<Comp>(&C[v], &C[u]);
25 }
26
27 struct Node {
28     Edge<Node> *e;
```

3 树与图

```
29     int dfn, low;
30 } N[MAXN];
31 Edge<Node> _poolN[MAXN << 1], *_currN = _poolN;
32 void addEdgeN(int u, int v) {
33     N[u].e = new (_currN++) Edge<Node>(&N[u], &N[v]);
34     N[v].e = new (_currN++) Edge<Node>(&N[v], &N[u]);
35 }
36
37 namespace Tarjan {
38     struct DJS {
39         int f[MAXN];
40
41         void init(int n) {
42             for (int i = 1; i <= n; i++) f[i] = i;
43         }
44
45         int find(int x) {
46             return x == f[x] ? x : f[x] = find(f[x]);
47         }
48
49         void merge(int x, int y) {
50             f[find(x)] = find(y);
51         }
52     } djs;
53
54     int dfsClock;
55     std::vector<Edge<Node> *> bridges;
56
57     void dfs(Node *u, Node *fa = NULL) {
58         u->dfn = ++dfsClock;
59         u->low = u->dfn;
60
61         for (Edge<Node> *e = u->e; e; e = e->next) if (e->v != fa) {
62             if (e->v->dfn) {
63                 u->low = std::min(u->low, e->v->dfn);
64             } else {
65                 dfs(e->v, u);
66                 u->low = std::min(u->low, e->v->low);
67                 if (e->v->low > u->dfn) {
68                     bridges.push_back(e);
69                 } else {
70                     djs.merge(u - N, e->v - N);
71                 }
72             }
73         }
74     }
75
76     void findECC(int n) {
77         djs.init(n);
78         dfsClock = 0;
79         dfs(&N[1]);
80     }
81
82     void rebuild() {
83         for (auto e : bridges) {
84             int x = djs.find(e->u - N);
```

```

85         int y = djs.find(e->v - N);
86         if (x != y) addEdgeC(x, y);
87     }
88 }
89 } // namespace Tarjan
90
91 int main() {
92     int n, m;
93     scanf("%d %d", &n, &m);
94
95     for (int i = 0, u, v; i < m; i++) {
96         scanf("%d %d", &u, &v);
97         addEdgeN(u, v);
98     }
99
100    Tarjan::findECC(n);
101    Tarjan::rebuild();
102
103    return 0;
104 }
```

3.12.3 无向图找点双连通分量并缩点建图

```

1 #include <cstdio>
2 #include <vector>
3 #include <stack>
4 #include <algorithm>
5
6 const int MAXN = 300005;
7
8 template <typename T>
9 struct Edge {
10     T *u, *v;
11     Edge *next;
12
13     Edge() {}
14     Edge(T *u, T *v) : u(u), v(v), next(u->e) {}
15 };
16
17 struct Comp {
18     Edge<Comp> *e;
19 } C[MAXN];
20 Edge<Comp> _poolC[MAXN << 1], *_currC = _poolC;
21
22 void addEdge(Comp *u, Comp *v) {
23     u->e = new (_currC++) Edge<Comp>(u, v);
24     v->e = new (_currC++) Edge<Comp>(v, u);
25 }
26
27 struct Node {
28     Edge<Node> *e;
29     Comp *c; // a cut's comp is meaningless
30     int dfn, low;
31     bool isCut;
32 } N[MAXN];
```

3 树与图

```
33 Edge<Node> _poolN[MAXN << 1], *_currN = _poolN;
34
35 void addEdge(int u, int v) {
36     N[u].e = new (_currN++) Edge<Node>(&N[u], &N[v]);
37     N[v].e = new (_currN++) Edge<Node>(&N[v], &N[u]);
38 }
39
40 namespace Tarjan {
41     int dfsClock, compCnt;
42     std::stack<Edge<Node> *> s;
43     std::vector<Node *> cuts;
44
45     void dfs(Node *u, Node *fa = NULL) {
46         u->dfn = ++dfsClock;
47         u->low = u->dfn;
48         int size = 0;
49
50         for (Edge<Node> *e = u->e; e; e = e->next) if (e->v != fa) {
51             s.push(e);
52
53             if (e->v->dfn) {
54                 u->low = std::min(u->low, e->v->dfn);
55             } else {
56                 ++size;
57                 dfs(e->v, u);
58                 u->low = std::min(u->low, e->v->low);
59                 if (e->v->low >= u->dfn) {
60                     u->isCut = true;
61                     cuts.push_back(u);
62
63                     Comp *c = &C[++compCnt];
64                     while (true) {
65                         Edge<Node> *t = s.top();
66                         s.pop();
67
68                         if (t->u->c != c) t->u->c = c;
69                         if (t->v->c != c) t->v->c = c;
70
71                         if (t->u == u && t->v == e->v) break;
72                     }
73                 }
74             }
75         }
76
77         if (!fa && size > 1) {
78             cuts.push_back(u);
79             u->isCut = true;
80         }
81     }
82
83     void findVCC(int n) {
84         dfsClock = 0;
85         while (!s.empty()) s.pop();
86         dfs(&N[1]);
87     }
88 }
```

```

89     void rebuild() {
90         for (Edge<Node> *e = _poolN; e < _currN; e++) if (e->u->c != e->v->c)
91             addEdge(e->u->c, e->v->c);
92     }
93 }
94
95 int main() {
96     int n, m;
97     scanf("%d %d", &n, &m);
98
99     for (int i = 0, u, v; i < m; i++) {
100         scanf("%d %d", &u, &v);
101         addEdge(u, v);
102     }
103
104     Tarjan::findVCC(n);
105     Tarjan::rebuild();
106
107     return 0;
108 }
```

3.13 2-SAT

```

1 #include <cstdio>
2 #include <stack>
3 #include <algorithm>
4
5 const int MAXN = 1005;
6 const int MAXM = 2000005;
7
8 template<typename T>
9 struct Edge {
10     T *u, *v;
11     Edge *next;
12
13     Edge() {}
14     Edge(T *u, T *v) : u(u), v(v), next(u->e) {}
15 };
16
17 struct Comp {
18     Edge<Comp> *e;
19     Comp *opp;
20     int deg, mark;
21
22     Comp() : mark(-1) {}
23 } C[MAXN << 1];
24 Edge<Comp> _poolC[MAXM], *_currC = _poolC;
25 void addEdge(Comp *u, Comp *v) {
26     u->e = new (_currC++) Edge<Comp>(u, v);
27     ++v->deg;
28 }
29
30 struct Node {
31     Edge<Node> *e;
```

3 树与图

```
32     int dfn, low;
33     Comp *belong;
34     bool ins;
35 } N[MAXN << 1];
36 Edge<Node> _poolN[MAXM], *_currN = _poolN;
37 void addEdge(int u, int v) {
38     N[u].e = new (_currN++) Edge<Node>(&N[u], &N[v]);
39 }
40
41 void rebuild() {
42     for (Edge<Node> *e = _poolN; e != _currN; e++) if (e->u->belong != e->v->belong)
43         addEdge(e->v->belong, e->u->belong);
44 }
45
46 int getV(int x, int k) {
47     return (x << 1) - k;
48 }
49
50 namespace TwoSat {
51     std::stack<Node *> s;
52     int dfsClock, sccCnt;
53
54     void tarjan(Node *u) {
55         s.push(u);
56         u->dfn = u->low = ++dfsClock;
57         u->ins = true;
58
59         for (Edge<Node> *e = u->e; e; e = e->next) {
60             if (e->v->dfn == 0) {
61                 tarjan(e->v);
62                 u->low = std::min(u->low, e->v->low);
63             } else if (e->v->ins) {
64                 u->low = std::min(u->low, e->v->dfn);
65             }
66         }
67
68         if (u->dfn == u->low) {
69             Node *v;
70             Comp *c = &C[sccCnt++];
71             do {
72                 v = s.top();
73                 s.pop();
74                 v->ins = false;
75                 v->belong = c;
76             } while (v != u);
77         }
78     }
79
80     bool check(int n) {
81         for (int i = 1; i <= n; i++) if (N[getV(i, 0)].belong == N[getV(i, 1)].belong)
82             return false;
83         return true;
84     }
85
86     bool solve(int n) {
87         sccCnt = dfsClock = 0;
```

```

88     for (int i = 1; i <= n << 1; i++) if (N[i].dfn == 0) tarjan(&N[i]);
89     return check(n);
90   }
91 } // namespace TwoSat
92
93 namespace TopoSort {
94   std::stack<Comp *> s;
95
96   void dfs(Comp *u) {
97     if (u->mark != -1) return;
98
99     u->mark = 0;
100    for (Edge<Comp> *e; e; e = e->next) dfs(e->v);
101  }
102
103  void solve(int n) {
104    for (int i = 1; i <= n; i++) if (C[i].deg == 0) s.push(&C[i]);
105
106    while (!s.empty()) {
107      Comp *u = s.top();
108      s.pop();
109
110      if (u->mark != -1) continue;
111
112      u->mark = 1;
113      dfs(u->opp);
114
115      for (Edge<Comp> *e = u->e; e; e = e->next) if (!(--e->v->deg))
116        s.push(e->v);
117    }
118  }
119 } // namespace TopoSort
120
121 int main() {
122   int n, m;
123   scanf("%d %d", &n, &m);
124
125   for (int i = 0, u, v, op; i < m; i++) {
126     scanf("%d %d %d", &u, &v, &op);
127
128     if (op == 1) { // u and v can't be true at the same time
129       addEdge(getV(u, 1), getV(v, 0));
130       addEdge(getV(v, 1), getV(u, 0));
131     } else if (op == 2) { // u and v can't be false at the same time
132       addEdge(getV(u, 0), getV(v, 1));
133       addEdge(getV(v, 0), getV(u, 1));
134     } else if (op == 3) { // u and v must be different
135       addEdge(getV(u, 1), getV(v, 0));
136       addEdge(getV(v, 1), getV(u, 0));
137       addEdge(getV(u, 0), getV(v, 1));
138       addEdge(getV(v, 0), getV(u, 1));
139     } else { // u and v must be the same
140       addEdge(getV(u, 1), getV(v, 1));
141       addEdge(getV(v, 1), getV(u, 1));
142       addEdge(getV(u, 0), getV(v, 0));
143       addEdge(getV(v, 0), getV(u, 0));

```

```
144         }
145     }
146
147     TwoSat::solve(n);
148
149     rebuild();
150
151     TopoSort::solve(TwoSat::sccCnt);
152
153     return 0;
154 }
```

3.14 欧拉回路

3.14.1 DFS 版本

适用于有向图和无向图，代码中为无向图。

```
1 #include <cstdio>
2 #include <vector>
3 #include <algorithm>
4
5 const int MAXN = 100005;
6 const int MAXM = 200005;
7
8 struct Edge;
9 struct Node {
10     Edge *e, *curr;
11     int deg;
12 } N[MAXN];
13
14 struct Edge {
15     Node *u, *v;
16     Edge *next, *rev;
17     bool used;
18
19     Edge() {}
20     Edge(Node *u, Node *v) : u(u), v(v), next(u->e), used(false) {}
21 } _pool[MAXM << 1], *_curr = _pool;
22
23 void addEdge(int u, int v) {
24     N[u].e = new (_curr++) Edge(&N[u], &N[v]);
25     N[v].e = new (_curr++) Edge(&N[v], &N[u]);
26     N[u].e->rev = N[v].e;
27     N[v].e->rev = N[u].e;
28     ++N[u].deg;
29     ++N[v].deg;
30 }
31
32 namespace Euler {
33     std::vector<Edge *> path;
34
35     void dfs(Node *u) {
36         for (Edge *&e = u->curr; e; e = e->next) if (!e->used) {
37             e->used = e->rev->used = true;
```

```

38         dfs(e->v);
39         path.push_back(e);
40     }
41 }
42
43 bool solve(int n) {
44     int cnt = 0;
45     for (int i = 1; i <= n; i++) cnt += N[i].deg % 2;
46     if (cnt > 2) return false;
47     for (int i = 1; i <= n; i++) N[i].curr = N[i].e;
48     int s = -1;
49     if (cnt) for (int i = 1; i <= n; i++) if (N[i].deg % 2) {
50         s = i;
51         break;
52     }
53     s = (s == -1 ? 1 : s);
54     dfs(&N[s]);
55     std::reverse(path.begin(), path.end());
56     return true;
57 }
58 }
59
60 int main() {
61     int n, m;
62     scanf("%d %d", &n, &m);
63     for (int i = 0, u, v; i < m; i++) {
64         scanf("%d %d", &u, &v);
65         addEdge(u, v);
66     }
67
68     if (Euler::solve(n)) {
69         printf("%d", Euler::path[0]->u - N);
70         for (int i = 0; i < Euler::path.size(); i++) {
71             printf(" -> %d", Euler::path[i]->v - N);
72         }
73         puts("");
74     } else {
75         puts("-1");
76     }
77
78     return 0;
79 }
```

3.14.2 无递归版本 有向图

```

1 #include <cstdio>
2 #include <vector>
3 #include <stack>
4 #include <algorithm>
5
6 const int MAXN = 100005;
7 const int MAXM = 400005;
8
9 struct Edge;
10 struct Node {
```

3 树与图

```
11     Edge *e, *curr;
12     int ideg, odeg;
13 } N[MAXN];
14
15 struct Edge {
16     Node *u, *v;
17     Edge *next;
18
19     Edge() {}
20     Edge(Node *u, Node *v) : u(u), v(v), next(u->e) {}
21 } _pool[MAXM], *_curr = _pool;
22
23 void addEdge(int u, int v) {
24     N[u].e = new (_curr++) Edge(&N[u], &N[v]);
25     ++N[u].odeg;
26     ++N[v].ideg;
27 }
28
29 namespace Hierholzer {
30     std::stack<Node *> currPath;
31     std::vector<Node *> euler;
32
33     void hierholzer(int s) {
34         currPath.push(&N[s]); // delete this line if a Eulerian Path is required
35         Node *u = &N[s];
36         do {
37             if (u->odeg == 0) {
38                 euler.push_back(currPath.top());
39                 currPath.pop();
40                 if (!currPath.empty()) u = currPath.top();
41             } else {
42                 Node *v = u->curr->v;
43                 u->curr = u->curr->next;
44                 --u->odeg;
45                 u = v;
46                 currPath.push(u);
47             }
48         } while (!currPath.empty());
49     }
50
51     bool solve(int n) {
52         int cntS = 0, cntT = 0;
53         for (int i = 1; i <= n; i++) {
54             if (N[i].ideg + 1 == N[i].odeg) ++cntS;
55             else if (N[i].odeg + 1 == N[i].ideg) ++cntT;
56             else if (N[i].ideg != N[i].odeg) return false;
57         }
58         if (cntS > 1 || cntT > 1) return false;
59         int s = -1;
60         for (int i = 1; i <= n; i++) {
61             if (N[i].ideg + 1 == N[i].odeg) s = i;
62             N[i].curr = N[i].e;
63         }
64         s = (s == -1 ? 1 : s);
65         hierholzer(s);
66         return true;
67 }
```

```

67     }
68 }
69
70 int main() {
71     int n, m;
72     scanf("%d %d", &n, &m);
73     for (int i = 0, u, v; i < m; i++) {
74         scanf("%d %d", &u, &v);
75         addEdge(u, v);
76     }
77
78     if (Hierholzer::solve(n)) {
79         for (int i = Hierholzer::euler.size() - 1; i >= 0; i--) {
80             printf("%d", Hierholzer::euler[i] - N);
81             if (i) printf(" -> ");
82         }
83         puts("");
84     } else {
85         puts("-1");
86     }
87
88     return 0;
89 }
```

3.14.3 无递归版本 无向图

```

1 #include <cstdio>
2 #include <list>
3 #include <vector>
4 #include <stack>
5 #include <algorithm>
6
7 const int MAXN = 100005;
8 const int MAXM = 400005;
9
10 struct Edge;
11 struct Node {
12     std::list<Edge> e;
13     int deg;
14 } N[MAXN];
15
16 struct Edge {
17     Node *u, *v;
18     std::list<Edge>::iterator rev;
19
20     Edge() {}
21     Edge(Node *u, Node *v) : u(u), v(v) {}
22 };
23
24 void addEdge(int u, int v) {
25     N[u].e.emplace_front(&N[u], &N[v]);
26     N[v].e.emplace_front(&N[v], &N[u]);
27     N[u].e.front().rev = N[v].e.begin();
28     N[v].e.front().rev = N[u].e.begin();
29     ++N[u].deg;
```

3 树与图

```
30     ++N[v].deg;
31 }
32
33 namespace Hierholzer {
34     std::stack<Node *> currPath;
35     std::vector<Node *> euler;
36
37     void hierholzer(Node *s) {
38         currPath.push(s); // delete this line if a Eulerian Path is required
39         Node *u = s;
40         do {
41             if (u->deg == 0) {
42                 euler.push_back(currPath.top());
43                 currPath.pop();
44                 if (!currPath.empty()) u = currPath.top();
45             } else {
46                 Node *v = u->e.front().v;
47                 v->e.erase(u->e.front().rev);
48                 u->e.erase(u->e.begin());
49                 --u->deg;
50                 --v->deg;
51                 u = v;
52                 currPath.push(u);
53             }
54         } while (!currPath.empty());
55     }
56
57     bool solve(int n) {
58         int cnt = 0;
59         for (int i = 1; i <= n; i++) cnt += N[i].deg % 2;
60         if (cnt > 2) return false;
61         int s = -1;
62         if (cnt) for (int i = 1; i <= n; i++) if (N[i].deg % 2) {
63             s = i;
64             break;
65         }
66         s = (s == -1 ? 1 : s);
67         hierholzer(&N[s]);
68         return true;
69     }
70 }
71
72 int main() {
73     int n, m;
74     scanf("%d %d", &n, &m);
75     for (int i = 0, u, v; i < m; i++) {
76         scanf("%d %d", &u, &v);
77         addEdge(u, v);
78     }
79
80     if (Hierholzer::solve(n)) {
81         for (int i = Hierholzer::euler.size() - 1; i >= 0; i--) {
82             printf("%d", Hierholzer::euler[i] - N);
83             if (i) printf(" -> ");
84         }
85         puts("");
86     }
87 }
```

```

86     } else {
87         puts("-1");
88     }
89
90     return 0;
91 }
```

3.15 最大团/最大独立集/弦图最小染色

最大团 = 补图最大独立集

对于弦图：最大团 = 最小染色

```

1 #include <cstdio>
2 #include <algorithm>
3
4 const int MAXN = 42;
5
6 bool G[MAXN][MAXN];
7
8 namespace MaxClique {
9
10 int n, ans, cnt[MAXN], group[MAXN], vis[MAXN];
11
12 bool dfs(int u, int pos) {
13     for (int i = u + 1; i <= n; i++) {
14         if (cnt[i] + pos <= ans) return false;
15         if (G[u][i]) {
16             int j;
17             for (j = 0; j < pos; j++) if (!G[i][vis[j]]) break;
18             if (j == pos) {
19                 vis[pos] = i;
20                 if (dfs(i, pos + 1)) return true;
21             }
22         }
23     }
24
25     if (pos > ans) {
26         for (int i = 0; i < pos; i++) group[i] = vis[i];
27         ans = pos;
28         return true;
29     }
30
31     return false;
32 }
33
34 int solve(int n) {
35     ans = -1;
36     MaxClique::n = n;
37
38     for (int i = n; i; i--) {
39         vis[0] = i;
40         dfs(i, 1);
41         cnt[i] = ans;
42     }
}
```

```

43
44     return ans < 0 ? 0 : ans;
45 }
46
47 } // namespace MaxClique
48
49 int main() {
50     int n, m;
51     scanf("%d %d", &n, &m);
52
53     for (int i = 1; i <= n; i++) std::fill(G[i] + 1, G[i] + n + 1, true);
54     for (int i = 0, u, v; i < m; i++) {
55         scanf("%d %d", &u, &v);
56         G[u][v] = G[v][u] = false;
57     }
58
59     int ans = MaxClique::solve(n);
60     printf("%d\n", ans);
61
62     return 0;
63 }
```

3.16 最小树形图

3.16.1 朱刘算法

```

1 #include <cstdio>
2 #include <climits>
3 #include <algorithm>
4
5 const int MAXN = 3005;
6
7 namespace Chuliu {
8     int G[MAXN][MAXN], n, eg[MAXN], queue[MAXN];
9     bool used[MAXN], pass[MAXN], more;
10
11    void combine(int id, long long &res) {
12        int tot = 0, from;
13        for (; id != 0 && !pass[id]; id = eg[id]) {
14            queue[tot++] = id;
15            pass[id] = true;
16        }
17        for (from = 0; from < tot && queue[from] != id; ++from) {}
18        if (from == tot) return;
19        more = true;
20        for (int i = from; i < tot; i++) {
21            res += G[eg[queue[i]]][queue[i]];
22            if (i != from) {
23                used[queue[i]] = true;
24                for (int j = 1; j <= n; j++) if (!used[j])
25                    G[id][j] = std::min(G[id][j], G[queue[i]][j]);
26            }
27        }
28        for (int i = 1; i <= n; i++) if (!used[i] && i != id) {
```

```

29         for (int j = from; j < tot; j++) {
30             int k = queue[j];
31             G[i][id] = std::min(G[i][id], G[i][k] - G[eg[k]][k]);
32         }
33     }
34 }
35
36 long long solve(int n, int root) {
37     ChuLiu::n = n;
38     std::fill(used + 1, used + n + 1, false);
39     long long res = 0;
40     for (more = true; more; ) {
41         more = false;
42         std::fill(eg + 1, eg + n + 1, 0);
43         for (int i = 1, k; i <= n; i++) if (!used[i] && i != root) {
44             k = 0;
45             for (int j = 1; j <= n; j++) if (!used[j] && i != j) {
46                 if (k == 0 || G[j][i] < G[k][i])
47                     k = j;
48             }
49             eg[i] = k;
50         }
51         std::fill(pass + 1, pass + n + 1, false);
52         for (int i = 1; i <= n; i++) if (!used[i] && !pass[i] && i != root)
53             combine(i, res);
54     }
55     for (int i = 1; i <= n; i++) if (!used[i] && i != root) {
56         if (G[eg[i]][i] == INT_MAX) return -1;
57         res += G[eg[i]][i];
58     }
59     return res;
60 }
61 }
62
63 int main() {
64     int n, m, rt;
65     scanf("%d %d %d", &n, &m, &rt);
66     for (int i = 1; i <= n; i++)
67         std::fill(ChuLiu::G[i] + 1, ChuLiu::G[i] + n + 1, INT_MAX);
68     for (int i = 0, u, v, w; i < m; i++) {
69         scanf("%d %d %d", &u, &v, &w);
70         ChuLiu::G[u][v] = w;
71     }
72
73     long long ans = ChuLiu::solve(n, rt);
74     printf("%lld\n", ans);
75
76     return 0;
77 }
```

3.16.2 Tarjan 的优化实现

内存占用较大。

```

1 #include <cstdio>
2 #include <vector>
```

3 树与图

```
3 #include <stack>
4 #include <queue>
5 #include <algorithm>
6
7 /*
8  * It takes O(n^2) time on dense graph
9  * To get O(m \log n) time on sparse graph, change 'std::vector<Edge *> in' to __gnu_pbds::priority_queue<
10 * Edge *>,
11 * and change line 71-72, 113-138
12 */
13 const int MAXN = 2505;
14
15 struct Graph {
16     struct Edge {
17         int u, v, w, ow;
18         bool removed;
19         std::vector<Edge *> ch;
20         Edge *pa;
21
22         Edge() {}
23         Edge(int u, int v, int w) : u(u), v(v), w(w), ow(w), pa(NULL), removed(false) {}
24     };
25     std::vector<Edge> E;
26
27     void addEdge(int u, int v, int w) {
28         if (u == v) return;
29         E[eid] = Edge(u, v, w);
30         G[u].push_back(&E[eid]);
31         in[v].push_back(&E[eid]);
32         ++eid;
33     }
34
35     std::vector<Edge *> G[MAXN], in[MAXN];
36     int n, eid;
37
38     void init(int n, int m) {
39         this->n = n;
40         E.resize(m);
41         this->eid = 0;
42     }
43 } G;
44
45 struct DSU {
46     int f[MAXN];
47
48     void init(int n) {
49         for (int i = 0; i <= n; i++) f[i] = i;
50     }
51     int find(int x) { return x == f[x] ? x : f[x] = find(f[x]); }
52     bool test(int x, int y) { return find(x) == find(y); }
53     void merge(int x, int y) { f[find(y)] = find(x); }
54 };
55
56 class OptimumBranching {
57 public:
```

```

58     long long solve(Graph &G, int rt) {
59         std::fill_n(change + 1, G.n, 0);
60         S.init(G.n);
61         W.init(G.n);
62
63         std::stack<int> roots;
64         std::vector<Edge *> F;
65
66         for (int i = 1; i <= G.n; i++) if (i != rt) roots.push(i);
67         while (!roots.empty()) {
68             int u = roots.top();
69             roots.pop();
70
71             Edge *min = NULL;
72             for (Edge *e : G.in[u]) if (!min || e->w < min->w) min = e;
73
74             F.push_back(min);
75             for (Edge *e : cycle[u]) {
76                 e->pa = min;
77                 min->ch.push_back(e);
78             }
79
80             if (cycle[u].empty()) lambda[u] = min;
81
82             if (!W.test(min->u, min->v)) {
83                 enter[u] = min;
84                 W.merge(min->u, min->v);
85             } else {
86                 std::vector<Edge *> cycleEdge;
87                 std::vector<int> cycleRepr;
88
89                 Edge *max = min;
90                 enter[u] = NULL;
91
92                 cycleEdge.push_back(min);
93                 cycleRepr.push_back(S.find(min->v));
94                 for (int v = S.find(min->u); enter[v]; v = S.find(enter[v]->u)) {
95                     cycleEdge.push_back(enter[v]);
96                     cycleRepr.push_back(v);
97                     if (max->w < enter[v]->w) max = enter[v];
98                 }
99
100                for (Edge *e : cycleEdge) change[S.find(e->v)] = max->w - e->w;
101
102                int nr = cycleRepr.front();
103                for (int v : cycleRepr) {
104                    S.merge(v, nr);
105                    nr = S.find(nr);
106                }
107                roots.push(nr);
108                cycle[nr].swap(cycleEdge);
109
110                for (int v : cycleRepr) for (Edge *e : G.in[v]) e->w += change[v];
111
112                std::vector<Edge *> nin;
113                for (int i = 1; i < cycleRepr.size(); i++) {

```

3 树与图

```
114         auto i1 = G.in[cycleRepr[i]].begin();
115         auto e1 = G.in[cycleRepr[i]].end();
116         auto i2 = G.in[cycleRepr[i - 1]].begin();
117         auto e2 = G.in[cycleRepr[i - 1]].end();
118
119         while (i1 != e1 || i2 != e2) {
120             while (i1 != e1 && S.test((*i1)->u, nr)) ++i1;
121             while (i2 != e2 && S.test((*i2)->u, nr)) ++i2;
122
123             if (i1 == e1 && i2 == e2) break;
124             else if (i1 == e1) nin.push_back(*i2++);
125             else if (i2 == e2) nin.push_back(*i1++);
126             else if ((*i1)->u < (*i2)->u) nin.push_back(*i1++);
127             else if ((*i1)->u > (*i2)->u) nin.push_back(*i2++);
128             else {
129                 if ((*i1)->w < (*i2)->w) nin.push_back(*i1);
130                 else nin.push_back(*i2);
131                 ++i1;
132                 ++i2;
133             }
134         }
135
136         G.in[cycleRepr[i]].swap(nin);
137         nin.clear();
138     }
139     G.in[nr].swap(G.in[cycleRepr.back()]);
140     change[nr] = 0;
141 }
142 }
143
144 long long res = 0;
145 std::stack<Edge *> froots;
146 for (Edge *e : F) if (e->pa == NULL) froots.push(e);
147 while (!froots.empty()) {
148     Edge *e = froots.top();
149     froots.pop();
150     if (e->removed) continue;
151     res += e->ow;
152     remove(lambda[e->v], froots);
153 }
154
155 return res;
156 }
157
158 private:
159     using Edge = Graph::Edge;
160     DSU S, W;
161     std::vector<Edge *> cycle[MAXN];
162     Edge *lambda[MAXN], *enter[MAXN];
163     int change[MAXN];
164
165     void remove(Edge *e, std::stack<Edge *> &roots) {
166         for (; e; e = e->pa) {
167             e->removed = true;
168             for (Edge *c : e->ch) {
169                 roots.push(c);
```

```

170             c->pa = NULL;
171         }
172         e->ch.clear();
173     }
174 }
175 } ob;
176
177 bool check(int rt, int n) {
178     static int vis[MAXN];
179
180     std::queue<int> q;
181     q.push(rt);
182     vis[rt] = true;
183     while (!q.empty()) {
184         int u = q.front();
185         q.pop();
186         for (Graph::Edge *e : G.G[u]) if (!vis[e->v]) {
187             vis[e->v] = true;
188             q.push(e->v);
189         }
190     }
191     for (int i = 1; i <= n; i++) if (!vis[i]) return false;
192     return true;
193 }
194
195 int main() {
196     int n, m, rt;
197     scanf("%d %d %d", &n, &m, &rt);
198
199     G.init(n, m);
200     for (int i = 0, u, v, w; i < m; i++) {
201         scanf("%d %d %d", &u, &v, &w);
202         G.addEdge(u, v, w);
203     }
204
205     if (!check(rt, n)) return puts("-1"), 0;
206
207     long long ans = ob.solve(G, rt);
208     printf("%lld\n", ans);
209
210     return 0;
211 }
```

3.17 虚树

```

1 #include <cstdio>
2 #include <algorithm>
3
4 const int MAXN = 300005;
5 const int MAXN_LOG = 20;
6
7 template <typename T>
8 struct Edge {
9     T *u, *v;
```

3 树与图

```
10     Edge *next;
11
12     Edge() {}
13     Edge(T *u, T *v) : u(u), v(v), next(u->e) {}
14 };
15
16 struct Node {
17     Edge<Node> *e;
18     Node *f[MAXN_LOG];
19     int dfn, dep;
20 } N[MAXN];
21 Edge<Node> _poolN[MAXN << 1], *_currN = _poolN;
22 void addEdge(int u, int v) {
23     N[u].e = new (_currN++) Edge<Node>(&N[u], &N[v]);
24     N[v].e = new (_currN++) Edge<Node>(&N[v], &N[u]);
25 }
26
27 struct VirN {
28     Edge<VirN> *e;
29     Node *r;
30 } V[MAXN];
31 Edge<VirN> _poolV[MAXN << 1], *_currV = _poolV;
32 void addEdge(VirN *u, VirN *v) {
33     u->e = new (_currV++) Edge<VirN>(u, v);
34     v->e = new (_currV++) Edge<VirN>(v, u);
35 }
36
37 void dfs(Node *u, bool init = true) {
38     if (init) {
39         u->dep = 1;
40         u->f[0] = u;
41     }
42
43     static int dfsClock = 0;
44     u->dfn = ++dfsClock;
45
46     for (int i = 1; i < MAXN_LOG; i++) u->f[i] = u->f[i - 1]->f[i - 1];
47
48     for (Edge<Node> *e = u->e; e; e = e->next) if (e->v != u->f[0]) {
49         e->v->f[0] = u;
50         e->v->dep = u->dep + 1;
51         dfs(e->v, false);
52     }
53 }
54
55 Node *lca(Node *u, Node *v) {
56     if (u->dep < v->dep) std::swap(u, v);
57
58     for (int i = MAXN_LOG - 1; ~i; i--) {
59         if (u->f[i]->dep >= v->dep) u = u->f[i];
60     }
61
62     for (int i = MAXN_LOG - 1; ~i; i--) {
63         if (u->f[i] != v->f[i]) {
64             u = u->f[i];
65             v = v->f[i];
66         }
67     }
68 }
```

```

66         }
67     }
68
69     return u == v ? u : u->f[0];
70 }
71
72 void build(bool flag, int n, int &tot) {
73     static VirN *stack[MAXN];
74     int top = 0;
75
76     if (!flag) stack[top++] = &V[0];
77
78     for (int i = 1; i <= n; i++) {
79         if (!top) {
80             stack[top++] = &V[i];
81             continue;
82         }
83         Node *p = lca(stack[top - 1]->r, V[i].r);
84         if (p == stack[top - 1]->r) {
85             stack[top++] = &V[i];
86             continue;
87         }
88         while (top - 2 >= 0 && stack[top - 2]->r->dep >= p->dep) {
89             addEdge(stack[top - 2], stack[top - 1]);
90             top--;
91         }
92         if (stack[top - 1]->r != p) {
93             V[++tot].r = p;
94             addEdge(&V[tot], stack[--top]);
95             stack[top++] = &V[tot];
96         }
97         stack[top++] = &V[i];
98     }
99
100    for (int i = 0; i < top - 1; i++) addEdge(stack[i], stack[i + 1]);
101 }
102
103 bool cmp(int i, int j) { return N[i].dfn < N[j].dfn; }
104
105 void clear(int n) {
106     _currV = _poolV;
107     for (int i = 0; i <= n; i++) V[i].e = NULL;
108 }
109
110 void solve() {
111     int k;
112     scanf("%"D, &k);
113     clear(k);
114
115     static int order[MAXN], h[MAXN];
116     for (int i = 0; i < k; i++) scanf("%"D, &h[i]), order[i] = h[i];
117
118     std::sort(h, h + k, cmp);
119     int tot = 0;
120     bool flag = h[0] == 1;
121     for (int i = 0; i < k; i++) V[++tot].r = &N[h[i]];

```

```

122     build(flag, tot, tot);
123
124     // do something...
125 }
126
127 int main() {
128     int n;
129     scanf("%d", &n);
130     for (int i = 1, u, v; i < n; i++) {
131         scanf("%d %d", &u, &v);
132         addEdge(u, v);
133     }
134
135     dfs(&N[1]);
136
137     V[0].r = &N[1];
138     int q;
139     scanf("%d", &q);
140     while (q--) solve();
141
142     return 0;
143 }
```

3.18 二分图最大匹配

```

1 #include <cstdio>
2 #include <vector>
3 #include <queue>
4 #include <algorithm>
5
6 const int MAXN = 1005; // one side
7 const int MAXM = 1005; // the other side
8
9 namespace HopcroftKarp {
10     std::vector<int> G[MAXN];
11     int matx[MAXN], maty[MAXM], dx[MAXN], dy[MAXM];
12     bool vis[MAXM];
13
14     bool find(int u) {
15         for (int v : G[u]) if (!vis[v] && dy[v] == dx[u] + 1) {
16             vis[v] = true;
17             if (!maty[v] || find(maty[v])) {
18                 matx[u] = v;
19                 maty[v] = u;
20                 return true;
21             }
22         }
23     }
24 }
25
26 int solve(int n, int m) {
27     std::fill(matx + 1, matx + n + 1, 0);
28     std::fill(maty + 1, maty + m + 1, 0);
29     int res = 0;
```

```

30     while (true) {
31         static std::queue<int> q;
32         while (!q.empty()) q.pop();
33         bool flag = false;
34         std::fill(dx + 1, dx + n + 1, 0);
35         std::fill(dy + 1, dy + m + 1, 0);
36         for (int i = 1; i <= n; i++) if (!matx[i]) q.push(i);
37         while (!q.empty()) {
38             int u = q.front();
39             q.pop();
40             for (int v : G[u]) if (!dy[v]) {
41                 dy[v] = dx[u] + 1;
42                 if (maty[v]) {
43                     dx[maty[v]] = dy[v] + 1;
44                     q.push(maty[v]);
45                 } else {
46                     flag = true;
47                 }
48             }
49         }
50         if (!flag) break;
51         std::fill(vis + 1, vis + m + 1, false);
52         for (int i = 1; i <= n; i++) if (!matx[i] && find(i)) ++res;
53     }
54     return res;
55 }
56 } // namespace HopcroftKarp
57
58 int main() {
59     int n1, n2, m;
60     scanf("%d %d %d", &n1, &n2, &m);
61     for (int i = 0, u, v; i < m; i++) {
62         scanf("%d %d", &u, &v);
63         HopcroftKarp::G[u].push_back(v);
64     }
65     printf("%d\n", HopcroftKarp::solve(n1, n2));
66
67     return 0;
68 }
```

3.19 二分图最大权完备匹配

```

1 #include <cstdio>
2 #include <climits>
3 #include <algorithm>
4
5 const int MAXN = 505;
6
7 namespace KuhnMunkres {
8     int n, G[MAXN][MAXN], x[MAXN], y[MAXN], prevx[MAXN], prevy[MAXN], mat[MAXN], slack[MAXN], par[MAXN];
9
10    void adjust(int u) {
11        mat[u] = prevy[u];
12        if (prevy[mat[u]] != -2) adjust(prevx[mat[u]]);
13    }
14
15    int solve() {
16        for (int i = 0; i < n; i++) {
17            for (int j = 0; j < n; j++) {
18                if (mat[j] == -1) {
19                    if (slack[i] > 0) {
20                        mat[j] = i;
21                        par[i] = j;
22                        for (int k = 0; k < n; k++) {
23                            if (slack[k] > 0) slack[k] -= slack[i];
24                            else slack[k] += slack[i];
25                        }
26                    }
27                }
28            }
29        }
30        int res = 0;
31        for (int i = 0; i < n; i++) {
32            if (mat[i] != -1) res++;
33        }
34        return res;
35    }
36}
```

3 树与图

```
13     }
14
15     bool find(int u) {
16         for (int i = 1; i <= n; i++) {
17             if (prevy[i] == -1) {
18                 if (slack[i] > x[u] + y[i] - G[u][i]) {
19                     slack[i] = x[u] + y[i] - G[u][i];
20                     par[i] = u;
21                 }
22                 if (x[u] + y[i] == G[u][i]) {
23                     prevy[i] = u;
24                     if (mat[i] == -1) {
25                         adjust(i);
26                         return true;
27                     }
28                     if (prevx[mat[i]] != -1) continue;
29                     prevx[mat[i]] = i;
30                     if (find(mat[i])) return true;
31                 }
32             }
33         }
34         return false;
35     }
36
37     int solve(int n) {
38         KuhnMunkres::n = n;
39         std::fill(mat + 1, mat + n + 1, -1);
40         std::fill(y + 1, y + n + 1, -1);
41         for (int i = 1; i <= n; i++) {
42             x[i] = 0;
43             for (int j = 1; j <= n; j++) x[i] = std::max(x[i], G[i][j]);
44         }
45         bool flag = false;
46         for (int i = 1; i <= n; i++) {
47             std::fill(prevx + 1, prevx + n + 1, -1);
48             std::fill(prevy + 1, prevy + n + 1, -1);
49             std::fill(slack + 1, slack + n + 1, INT_MAX);
50             prevx[i] = -2;
51             if (find(i)) continue;
52             flag = false;
53             while (!flag) {
54                 int m = INT_MAX;
55                 for (int j = 1; j <= n; j++) if (prevy[j] == -1) m = std::min(m, slack[j]);
56                 for (int j = 1; j <= n; j++) {
57                     if (prevx[j] != -1) x[j] -= m;
58                     if (prevy[j] != -1) y[j] += m;
59                     else slack[j] -= m;
60                 }
61                 for (int j = 1; j <= n; j++) {
62                     if (prevy[j] == -1 && slack[j]) {
63                         prevy[j] = par[j];
64                         if (mat[j] == -1) {
65                             adjust(j);
66                             flag = true;
67                             break;
68                         }
69                 }
70             }
71         }
72     }
73 }
```

```

69             prevx[mat[j]] = j;
70             if (find(mat[j])) {
71                 flag = true;
72                 break;
73             }
74         }
75     }
76 }
77 int res = 0;
78 for (int i = 1; i <= n; i++) res += G[mat[i]][i];
79 return res;
80 }
81 }
82 } // namespace KuhnMunkres
83
84 int main() {
85     int n1, n2, m;
86     scanf("%d %d %d", &n1, &n2, &m);
87     for (int i = 0, u, v, w; i < m; i++) {
88         scanf("%d %d %d", &u, &v, &w);
89         KuhnMunkres::G[u][v] = w;
90     }
91     printf("%d\n", KuhnMunkres::solve(std::max(n1, n2)));
92
93     return 0;
94 }
```

3.20 一般图最大匹配

```

1 #include <bits/stdc++.h>
2
3 const int MAXN = 505;
4
5 struct Graph {
6     std::vector<int> G[MAXN];
7     int n, mate[MAXN];
8
9     void addEdge(int u, int v) {
10         G[u].push_back(v);
11         G[v].push_back(u);
12     }
13 } G;
14
15 class Blossom {
16 public:
17     int solve(Graph &G) {
18         this->G = &G;
19         for (int i = 1; i <= G.n; i++) G.mate[i] = -1;
20         for (int i = 1; i <= G.n; i++) if (G.mate[i] == -1) aug(i);
21
22         int res = 0;
23         for (int i = 1; i <= G.n; i++) res += (G.mate[i] > i);
24         return res;
25     }

```

3 树与图

```
26
27     private:
28         struct DJS {
29             int f[MAXN];
30
31             void init(int n) {
32                 for (int i = 1; i <= n; i++) f[i] = i;
33             }
34
35             int find(int x) { return x == f[x] ? x : f[x] = find(f[x]); }
36             bool test(int x, int y) { return find(x) == find(y); }
37             void merge(int x, int y) { f[find(x)] = find(y); }
38         } djs;
39
40         int next[MAXN], dsu[MAXN], mark[MAXN], vis[MAXN];
41         Graph *G;
42
43         int lca(int x, int y) {
44             static int t = 0;
45             ++t;
46             for (; ; std::swap(x, y)) if (x != -1) {
47                 if (vis[x = djs.find(x)] == t) return x;
48                 vis[x] = t;
49                 x = (G->mate[x] != -1) ? next[G->mate[x]] : -1;
50             }
51         }
52
53         std::queue<int> q;
54         void group(int a, int p) {
55             for (int b, c; a != p; djs.merge(a, b), djs.merge(b, c), a = c) {
56                 b = G->mate[a], c = next[b];
57                 if (djs.find(c) != p) next[c] = b;
58                 if (mark[b] == 2) mark[b] = 1, q.push(b);
59                 if (mark[c] == 2) mark[c] = 1, q.push(c);
60             }
61         }
62
63         void aug(int s) {
64             for (int i = 1; i <= G->n; i++) {
65                 next[i] = vis[i] = -1;
66                 mark[i] = 0;
67             }
68             djs.init(G->n);
69             while (!q.empty()) q.pop();
70
71             q.push(s);
72             mark[s] = 1;
73             while (G->mate[s] == -1 && !q.empty()) {
74                 int x = q.front();
75                 q.pop();
76
77                 for (int y : G->G[x]) {
78                     if (y != G->mate[x] && !djs.test(x, y) && mark[y] != 2) {
79                         if (mark[y] == 1) {
80                             int p = lca(x, y);
81                             if (djs.find(x) != p) next[x] = y;
```

```

82             if (djs.find(y) != p) next[y] = x;
83             group(x, p);
84             group(y, p);
85         } else if (G->mate[y] == -1) {
86             next[y] = x;
87             for (int j = y, k, l; j != -1; j = l) {
88                 k = next[j];
89                 l = G->mate[k];
90                 G->mate[j] = k;
91                 G->mate[k] = j;
92             }
93             break;
94         } else {
95             next[y] = x;
96             q.push(G->mate[y]);
97             mark[G->mate[y]] = 1;
98             mark[y] = 2;
99         }
100    }
101}
102}
103}
104} blossom;
105
106int main() {
107    int n, m;
108    scanf("%d %d", &n, &m);
109
110    G.n = n;
111    for (int i = 0, u, v; i < m; i++) {
112        scanf("%d %d", &u, &v);
113        G.addEdge(u, v);
114    }
115
116    blossom.solve(G);
117
118    for (int i = 1; i <= n; i++) printf("%d%c", G.mate[i], " \n"[i == n]);
119
120    return 0;
121}

```

4 字符串

4.1 Hash

4.2 KMP

```
1 #include <cstdio>
2 #include <cstring>
3
4 const int MAXN = 1000005;
5
6 int fail[MAXN];
7 void calcFail(char *s) {
8     int n = strlen(s + 1);
9
10    fail[1] = 0;
11    for (int i = 2; i <= n; i++) {
12        int j = fail[i - 1];
13        while (j && s[j + 1] != s[i]) j = fail[j];
14        fail[i] = s[i] == s[j + 1] ? j + 1 : 0;
15    }
16 }
17
18 int kmp(char *s, char *t) {
19     int res = 0;
20     int n = strlen(s + 1), m = strlen(t + 1);
21
22     for (int i = 1, j = 0; i <= n; i++) {
23         while (j && t[j + 1] != s[i]) j = fail[j];
24         if (t[j + 1] == s[i]) j++;
25         if (j == m) {
26             res++;
27             j = fail[j]; // j = 0 when not allowed overlapping
28         }
29     }
30
31     return res;
32 }
33
34 int main() {
35     static char a[MAXN], b[MAXN];
36     scanf("%s %s", a + 1, b + 1);
37
38     calcFail(b);
39
40     printf("%d\n", kmp(a, b));
41
42     return 0;
43 }
```

4.3 扩展 KMP

```

1 #include <cstdio>
2 #include <cstring>
3
4 const int MAXN = 1000005;
5
6 namespace ExKmp {
7     int next[MAXN], extend[MAXN];
8
9     void getNext(char *str, int n) {
10         int i = 0;
11         while (str[i] == str[i + 1] && i + 1 < n) i++;
12         next[0] = n;
13         next[1] = i;
14         int pos = 1;
15         for (i = 2; i < n; i++) {
16             if (next[i - pos] + i < next[pos] + pos) next[i] = next[i - pos];
17             else {
18                 int j = next[pos] + pos - i;
19                 if (j < 0) j = 0;
20                 while (i + j < n && str[j] == str[j + i]) ++j;
21                 next[i] = j;
22                 pos = i;
23             }
24         }
25     }
26
27     void getExtend(char *s, char *t) {
28         int n = strlen(s), m = strlen(t);
29         getNext(t, m);
30         int i = 0;
31         while (s[i] == t[i] && i < m && i < n) i++;
32         extend[0] = i;
33         int pos = 0;
34         for (i = 1; i < n; i++) {
35             if (next[i - pos] + i < extend[pos] + pos) extend[i] = next[i - pos];
36             else {
37                 int j = extend[pos] + pos - i;
38                 if (j < 0) j = 0;
39                 while (i + j < n && j < m && s[j + i] == t[j]) ++j;
40                 extend[i] = j;
41                 pos = i;
42             }
43         }
44     }
45 }
46
47 char s[MAXN], t[MAXN];
48
49 int main() {
50     scanf("%s %s", s, t);
51     ExKmp::getExtend(s, t);
52
53     return 0;
54 }
```

4.4 后缀数组

```

1 #include <cstdio>
2 #include <cstring>
3 #include <algorithm>
4
5 const int MAXN = 1000005;
6
7 namespace SuffixArray {
8     int n, sa[MAXN], rank[MAXN], height[MAXN];
9     char str[MAXN];
10
11     void buildSA(int m) {
12         static int fir[MAXN], sec[MAXN], buc[MAXN], temp[MAXN];
13         n = strlen(str);
14         str[n++] = 0;
15
16         std::fill(buc, buc + m, 0);
17         for (int i = 0; i < n; i++) buc[(int)str[i]]++;
18         for (int i = 1; i < m; i++) buc[i] += buc[i - 1];
19         for (int i = 0; i < n; i++) rank[i] = buc[(int)str[i]] - 1;
20
21         for (int l = 1; l < n; l <= 1) {
22             for (int i = 0; i < n; i++)
23                 fir[i] = rank[i], sec[i] = i + l < n ? rank[i + l] : 0;
24
25             std::fill(buc, buc + n, 0);
26             for (int i = 0; i < n; i++) buc[sec[i]]++;
27             for (int i = 1; i < n; i++) buc[i] += buc[i - 1];
28             for (int i = n - 1; ~i; i--) temp[--buc[sec[i]]] = i;
29
30             std::fill(buc, buc + n, 0);
31             for (int i = 0; i < n; i++) buc[fir[i]]++;
32             for (int i = 1; i < n; i++) buc[i] += buc[i - 1];
33             for (int i = n - 1; ~i; i--) sa[--buc[fir[temp[i]]]] = temp[i];
34
35             rank[sa[0]] = 0;
36             bool unique = true;
37             for (int i = 1; i < n; i++) {
38                 rank[sa[i]] = rank[sa[i - 1]];
39                 if (fir[sa[i]] == fir[sa[i - 1]] && sec[sa[i]] == sec[sa[i - 1]])
40                     unique = false;
41                 else rank[sa[i]]++;
42             }
43
44             if (unique) break;
45         }
46     }
47
48     void getHeight() {
49         int k = 0;
50         for (int i = 0; i < n - 1; i++) {
51             k ? k-- : 0;
52             int j = sa[rank[i] - 1];
53             while (str[i + k] == str[j + k]) k++;
54             height[rank[i]] = k;

```

```

55         }
56     }
57 }
58
59 int main() {
60     char *str = SuffixArray::str;
61     scanf("%"s, str);
62
63     SuffixArray::buildSA(128);
64     int *sa = SuffixArray::sa, n = SuffixArray::n;
65     for (int i = 1; i < n; i++) printf("d%c", sa[i] + 1, " \n"[i == n - 1]);
66
67     SuffixArray::getHeight();
68     int *height = SuffixArray::height + 1;
69     for (int i = 1; i < n - 1; i++) printf("d%c", height[i], " \n"[i == n - 2]);
70
71     return 0;
72 }
```

4.5 后缀自动机

```

1 #include <cstdio>
2 #include <vector>
3 #include <algorithm>
4
5 const int MAXN = 100005;
6 const int CHAR_SET = 26;
7
8 struct SAM {
9     struct Node {
10        Node *c[CHAR_SET], *next;
11        int max, posCnt;
12
13        Node(int max = 0, bool newSuffix = false) : max(max), posCnt(newSuffix), next(NULL), c() {}
14
15        int min() const {
16            return next->max + 1;
17        }
18    } *start, *last, _pool[MAXN << 1], *_curr;
19
20    SAM() {
21        init();
22    }
23
24    void init() {
25        _curr = _pool;
26        start = last = new (_curr++) Node;
27    }
28
29    Node *extend(int c) {
30        Node *u = new (_curr++) Node(last->max + 1, true), *v = last;
31
32        for (; v && !v->c[c]; v = v->next) v->c[c] = u;
33    }

```

```

34     if (!v) {
35         u->next = start;
36     } else if (v->c[c]->max == v->max + 1) {
37         u->next = v->c[c];
38     } else {
39         Node *n = new (_curr++) Node(v->max + 1), *o = v->c[c];
40         std::copy(o->c, o->c + CHAR_SET, n->c);
41         n->next = o->next;
42         u->next = o->next = n;
43         for (; v && v->c[c] == o; v = v->next) v->c[c] = n;
44     }
45
46     return last = u;
47 }
48
49 std::vector<Node *> topo;
50 std::vector<Node *> toposort() {
51     static int buc[MAXN << 1];
52     int max = 0;
53     for (Node *p = _pool; p != _curr; p++) {
54         max = std::max(max, p->max);
55         buc[p->max]++;
56     }
57     for (int i = 1; i <= max; i++) buc[i] += buc[i - 1];
58
59     topo.resize(_curr - _pool);
60     for (Node *p = _pool; p != _curr; p++) topo[--buc[p->max]] = p;
61
62     return topo;
63 }
64
65 void calc() {
66     toposort();
67
68     for (int i = topo.size() - 1; i; i--) topo[i]->next->posCnt += topo[i]->posCnt;
69 }
70 } sam;
71
72 int main() {
73
74     return 0;
75 }
```

4.6 AC 自动机

```

1 #include <cstdio>
2 #include <queue>
3 #include <algorithm>
4
5 const int MAXN = 100005;
6 const int CHAR_SET = 26;
7
8 struct ACAM {
9     struct Node {
```

```

10     Node *c[CHAR_SET], *next, *fail;
11     bool isWord;
12     int refCnt;
13
14     Node(bool isWord = false) : c(), next(NULL), fail(NULL), isWord(isWord), refCnt(0) {}
15
16     void apply() {
17         refCnt++;
18         if (next) next->apply();
19     }
20 } *root;
21
22 ACAM() : root(new Node) {}
23
24 Node *insert(char *begin, char *end) {
25     Node **u = &root;
26     for (char *p = begin; p != end; p++) {
27         if (!(*u)) *u = new Node;
28         u = &(*u)->c[*p - 'a'];
29     }
30
31     if (!(*u)) *u = new Node(true);
32     else (*u)->isWord = true;
33
34     return *u;
35 }
36
37 void build() {
38     std::queue<Node *> q;
39     q.push(root);
40     root->fail = root;
41     root->next = NULL;
42
43     while (!q.empty()) {
44         Node *u = q.front();
45         q.pop();
46
47         for (int i = 0; i < CHAR_SET; i++) {
48             Node *&c = u->c[i];
49
50             if (!c) {
51                 c = u == root ? root : u->fail->c[i];
52                 continue;
53             }
54
55             Node *v = u->fail;
56             c->fail = u != root && v->c[i] ? v->c[i] : root;
57             c->next = c->fail->isWord ? c->fail : c->fail->next;
58             q.push(c);
59         }
60     }
61 }
62
63 void exec(char *l, char *r) {
64     Node *u = root;
65     for (char *p = l; p != r; p++) {

```

```

66         u = u->c[*p - 'a'];
67         if (u->isWord) u->apply();
68         else if (u->next) u->next->apply();
69     }
70 }
71 } acam;
72
73 int main() {
74
75     return 0;
76 }
```

4.7 Manacher

```

1 #include <cstdio>
2 #include <cstring>
3 #include <algorithm>
4
5 const int MAXN = 1000005;
6
7 namespace Manacher {
8     char s[MAXN << 1];
9     int r[MAXN << 1], len;
10
11    void prepare(char *str) {
12        len = 0;
13        s[++len] = '@';
14        s[++len] = '#';
15        int n = strlen(str);
16        for (int i = 0; i < n; i++) s[++len] = str[i], s[++len] = '#';
17        s[++len] = '\0';
18    }
19
20    void manacher() {
21        int right = 0, pos = 0;
22        for (int i = 1; i <= len; i++) {
23            int x = right < i ? 1 : std::min(r[2 * pos - i], right - i);
24            while (s[i + x] == s[i - x]) ++x;
25            r[i] = x;
26            if (x + i > right) {
27                right = x + i;
28                pos = i;
29            }
30        }
31    }
32
33    void calc(char *str) {
34        prepare(str);
35        manacher();
36    }
37 }
38
39 int main() {
40     static char s[MAXN];
```

```

41     scanf("%s", s);
42
43     Manacher::calc(s);
44
45     int ans = 0, len = Manacher::len, *r = Manacher::r;
46     for (int i = 1; i <= len; i++) ans = std::max(ans, r[i] - 1);
47
48     printf("%d\n", ans);
49
50     return 0;
51 }
```

4.8 回文树

注：该实现的常数很大。

```

1 #include <cstdio>
2 #include <cstring>
3 #include <algorithm>
4
5 const int MAXN = 300005;
6 const int CHAR_SET = 26;
7
8 struct PalinT {
9     struct Node {
10         Node *c[CHAR_SET], *fail;
11         int len, cnt;
12
13         Node(int len = 0) : len(len), cnt(0), c(), fail(NULL) {}
14     } *even, *odd, *last, _pool[MAXN], *_curr;
15     char str[MAXN];
16     int size;
17
18     PalinT() : str() {
19         _curr = _pool;
20         last = even = new (_curr++) Node();
21         even->fail = odd = new (_curr++) Node(-1);
22         odd->fail = odd;
23         str[size = 0] = -1;
24     }
25
26     Node *extend(int c) {
27         str[++size] = c;
28         Node *v = last;
29         for (; str[size - v->len - 1] != str[size]; v = v->fail) {}
30
31         Node *u = v->c[c];
32         if (!u) {
33             u = v->c[c] = new (_curr++) Node(v->len + 2);
34             Node *p = v->fail;
35             for (; str[size - p->len - 1] != str[size]; p = p->fail) {}
36             u->fail = v == odd ? even : p->c[c];
37         }
38         u->cnt++;
39 }
```

```
40         return last = u;
41     }
42
43     void build(char *begin, char *end) {
44         for (char *p = begin; p != end; p++) extend(*p - 'a');
45     }
46
47     void count() {
48         for (Node *p = _curr - 1; p >= _pool; p--) p->fail->cnt += p->cnt;
49     }
50 } palinT;
51
52 int main() {
53     static char s[MAXN];
54     scanf("%s", s);
55
56     int n = strlen(s);
57     palinT.build(s, s + n);
58     palinT.count();
59
60     long long ans = 0;
61     for (PalinT::Node *p = palinT._pool; p != palinT._curr; p++)
62         ans = std::max(ans, (long long) p->len * p->cnt);
63
64     printf("%lld\n", ans);
65
66     return 0;
67 }
```

5 数学

5.1 欧几里得算法

```
1 #include <cstdio>
2
3 int gcd(int a, int b) {
4     return b ? gcd(b, a % b) : a;
5 }
6
7 void exgcd(int a, int b, int &g, int &x, int &y) {
8     if (b == 0) x = 1, y = 0, g = a;
9     else exgcd(b, a % b, g, y, x), y -= a / b * x;
10 }
11
12 int main() {
13
14     return 0;
15 }
```

5.2 组合数与 Lucas 定理

```
1 #include <cstdio>
2
3 const int MOD = 1000003;
4
5 int fact[MOD], inv[MOD];
6
7 long long pow(long long a, long long n) {
8     long long res = 1;
9     for (; n; n >= 1, a = a * a % MOD) if (n & 1) res = res * a % MOD;
10    return res;
11 }
12
13 void prepare() {
14     fact[0] = 1;
15     for (int i = 1; i < MOD; i++) fact[i] = (long long) fact[i - 1] * i % MOD;
16
17     inv[MOD - 1] = pow(fact[MOD - 1], MOD - 2);
18     for (int i = MOD - 2; i; i--) inv[i] = (long long) inv[i + 1] * (i + 1) % MOD;
19 }
20
21 int combi(int n, int m) {
22     if (m > n) return 0;
23     if (n < MOD) return (long long) fact[n] * inv[m] % MOD * inv[n - m] % MOD;
24     return (long long) combi(n / MOD, m / MOD) * combi(n % MOD, m % MOD) % MOD;
25 }
26
27 int main() {
28     prepare();
29 }
```

```
30     return 0;
31 }
```

5.3 线性预处理逆元

```
1 #include <cstdio>
2
3 const int MAXN = 3000005;
4
5 int fact[MAXN], invFact[MAXN], inv[MAXN];
6
7 void exgcd(int a, int b, int &x, int &y) {
8     if (b == 0) x = 1, y = 0;
9     else exgcd(b, a % b, y, x), y -= x * (a / b);
10 }
11
12 int getInv(int x, int mod) {
13     int res, temp;
14     exgcd(x, mod, res, temp);
15     return res;
16 }
17
18 int main() {
19     int n, p;
20     scanf("%"d "%d", &n, &p);
21
22     inv[1] = 1;
23     for (int i = 2; i <= n; i++) inv[i] = (long long) (p - p / i) * inv[p % i] % p;
24
25     fact[0] = 1;
26     for (int i = 1; i <= n; i++) fact[i] = (long long) fact[i - 1] * i % p;
27     invFact[n] = getInv(fact[n], p);
28     for (int i = n - 1; i; i--) invFact[i] = (long long) invFact[i + 1] * (i + 1) % p;
29
30     return 0;
31 }
```

5.4 线性筛

```
1 #include <cstdio>
2
3 const int MAXN = 10000005;
4
5 int prime[MAXN], primeCnt, mu[MAXN], phi[MAXN], d[MAXN];
6 long long s[MAXN];
7 bool notPrime[MAXN];
8
9 void sieve() {
10     static int minFact[MAXN], minPow[MAXN];
11
12     notPrime[0] = notPrime[1] = true;
13     mu[1] = phi[1] = d[1] = s[1] = 1;
```

```

14     minFact[1] = minPow[1] = 1;
15
16     for (int i = 2; i < MAXN; i++) {
17         if (!notPrime[i]) {
18             prime[primeCnt++] = i;
19             mu[i] = -1;
20             phi[i] = i - 1;
21             d[i] = 2;
22             s[i] = i + 1;
23             minFact[i] = i;
24             minPow[i] = 1;
25         }
26
27         for (int j = 0; j < primeCnt && i * prime[j] < MAXN; j++) {
28             notPrime[i * prime[j]] = true;
29             if (i % prime[j] == 0) {
30                 mu[i * prime[j]] = 0;
31                 phi[i * prime[j]] = phi[i] * prime[j];
32                 d[i * prime[j]] = d[i] / (minPow[i] + 1) * (minPow[i] + 2);
33                 if (i == minFact[i]) {
34                     s[i * prime[j]] = s[i] + i * prime[j];
35                 } else {
36                     s[i * prime[j]] = s[i / minFact[i]] * s[prime[j] * minFact[i]];
37                 }
38                 minPow[i * prime[j]] = minPow[i] + 1;
39                 minFact[i * prime[j]] = minFact[i] * prime[j];
40                 break;
41             }
42             mu[i * prime[j]] = -mu[i];
43             phi[i * prime[j]] = phi[i] * (prime[j] - 1);
44             d[i * prime[j]] = d[i] * 2;
45             s[i * prime[j]] = s[i] * s[prime[j]];
46             minPow[i * prime[j]] = 1;
47             minFact[i * prime[j]] = prime[j];
48         }
49     }
50 }
51
52 int main() {
53     sieve();
54
55     return 0;
56 }
```

5.5 杜教筛

```

1 #include <cstdio>
2 #include <map>
3
4 const int MAXNN = 1600000;
5
6 long long phi[MAXNN];
7 int prime[MAXNN], primeCnt;
8 bool notPrime[MAXNN];
```

```

9
10 void sieve() {
11     phi[1] = 1;
12     notPrime[0] = notPrime[1] = true;
13     for (int i = 2; i < MAXNN; i++) {
14         if (!notPrime[i]) {
15             prime[++primeCnt] = i;
16             phi[i] = i - 1;
17         }
18
19         for (int j = 1; j <= primeCnt && i * prime[j] < MAXNN; j++) {
20             notPrime[i * prime[j]] = true;
21             if (i % prime[j] == 0) {
22                 phi[i * prime[j]] = phi[i] * prime[j];
23                 break;
24             }
25             phi[i * prime[j]] = phi[i] * (prime[j] - 1);
26         }
27     }
28
29     for (int i = 2; i < MAXNN; i++) phi[i] += phi[i - 1];
30 }
31
32 /*
33 * h = f * g
34 * g(1) F(n) = H(n) - \sum_{i = 2}^{n} g(i) F(\lfloor \frac{n}{i} \rfloor)
35 */
36
37 int n;
38 long long phiH[MAXNN];
39 long long pSumPhi(int n) {
40     if (n < MAXNN) return phi[n];
41
42     int id = ::n / n;
43
44     if (phiH[id] != -1) return phiH[id];
45
46     long long res = (long long) n * (n + 1) / 2;
47     for (int i = 2, last; i <= n; i = last + 1) {
48         last = n / (n / i);
49         res -= (last - i + 1) * pSumPhi(n / i);
50     }
51     return phiH[id] = res;
52 }
53
54 int main() {
55     sieve();
56
57     scanf("%d", &n);
58     std::fill(phiH, phiH + n / MAXNN + 1, -1);
59     printf("%lld\n", pSumPhi(n));
60
61     return 0;
62 }
```

5.6 Min25 篩

```

1 #include <cstdio>
2 #include <cmath>
3
4 const int MAXN = 100005; // sqrt n
5 const int MOD = 1000000007;
6 int INV2, INV6;
7
8 long long qpow(long long a, long long n) {
9     long long res = 1;
10    for (; n; n >= 1, a = a * a % MOD) if (n & 1) res = res * a % MOD;
11    return res;
12 }
13
14 int prime[MAXN], primeCnt, lpf[MAXN];
15 long long sprime[MAXN], ssprime[MAXN];
16 bool notPrime[MAXN];
17
18 void sieve() {
19     for (int i = 2; i < MAXN; i++) {
20         if (!notPrime[i]) {
21             prime[++primeCnt] = i; // index from 1
22             sprime[primeCnt] = (sprime[primeCnt - 1] + i) % MOD;
23             ssprime[primeCnt] = (ssprime[primeCnt - 1] + 1ll * i * i) % MOD;
24             lpf[i] = primeCnt;
25         }
26         for (int j = 1; j <= primeCnt && i * prime[j] < MAXN; j++) {
27             notPrime[i * prime[j]] = true;
28             if (i % prime[j] == 0) break;
29             lpf[i * prime[j]] = j;
30         }
31     }
32 }
33
34 class Min25 {
35 private:
36     long long G[MAXN][3], Fprime[MAXN];
37     int list[MAXN], cnt;
38
39     int le[MAXN], ge[MAXN], lim, n;
40     inline int &id(int v) { return v <= lim ? le[v] : ge[n / v]; }
41
42     void init(int n) {
43         cnt = 0;
44         for (int i = 1, j, v; i <= n; i = n / j + 1) {
45             j = n / i;
46             v = j % MOD;
47             list[++cnt] = j;
48             id(j) = cnt;
49             // sum from 2 to j (assuming all the values are calculated as if they are prime)
50             // i^0
51             G[cnt][0] = v ? v - 1 : MOD - 1;
52             // i^1
53             G[cnt][1] = (2ll + v) * (v - 1ll) % MOD * INV2 % MOD;
54             // i^2

```

```

55         G[cnt][2] = (1ll * v * (v + 1) % MOD * (2 * v + 1) % MOD * INV6 % MOD - 1 + MOD) % MOD;
56     }
57 }
58
59 void calcFprime() {
60     for (int k = 1; k <= primeCnt && prime[k] <= lim; k++) {
61         int p = prime[k];
62         long long sqrp = 1ll * p * p;
63         for (int i = 1; list[i] >= sqrp; i++) {
64             int v = list[i] / p;
65             int id = this->id(v);
66             G[i][0] = (G[i][0] - G[id][0] + k - 1 + MOD) % MOD;
67             G[i][1] = (G[i][1] - 1ll * p * (G[id][1] - sprime[k - 1] + MOD) % MOD + MOD) % MOD;
68             G[i][2] = (G[i][2] - 1ll * p * p % MOD * (G[id][2] - ssprime[k - 1] + MOD) % MOD + MOD) %
69                         MOD;
70         }
71     }
72     for (int i = 1; i <= cnt; i++) {
73         // prefix sum of values at primes
74         Fprime[i] = (G[i][1] - G[i][0] + MOD) % MOD;
75     }
76 }
77
78 int fp(int p, int c) {
79     // value at power of prime
80     return qpow(p, c - 1) * (p - 1) % MOD;
81 }
82
83 long long F(int K, int n) {
84     if (n < prime[K] || n <= 1) return 0;
85     int id = this->id(n);
86     long long ans = Fprime[id] - (sprime[K - 1] - (K - 1));
87     ans = (ans % MOD + MOD) % MOD;
88     for (int i = K; i <= primeCnt && 1ll * prime[i] * prime[i] <= n; i++) {
89         long long pw = prime[i], pw2 = pw * pw;
90         for (int c = 1; pw2 <= n; pw = pw2, pw2 *= prime[i], c++) {
91             ans = (ans + fp(prime[i], c) * F(i + 1, n / pw) + fp(prime[i], c + 1)) % MOD;
92         }
93     }
94     return ans;
95 }
96
97 public:
98     long long solve(int n) {
99         this->n = n;
100        lim = sqrt(n);
101        init(n);
102        calcFprime();
103        long long res = F(1, n) + 1; // remember to add f(1)
104        return res >= MOD ? res -= MOD : res;
105    }
106
107 int main() {
108     sieve();
109     INV2 = qpow(2, MOD - 2);

```

```

110     INV6 = qpow(6, MOD - 2);
111
112     int n;
113     scanf("%d", &n);
114     long long ans = min25.solve(n);
115     printf("%lld\n", ans);
116
117     return 0;
118 }
```

5.7 线性同余方程

```

1 #include <cstdio>
2
3 const int MAXN = 100005;
4
5 void exgcd(long long a, long long b, long long &g, long long &x, long long &y) {
6     if (b == 0) x = 1, y = 0, g = a;
7     else exgcd(b, a % b, g, y, x), y -= a / b * x;
8 }
9
10 int mod[MAXN], rem[MAXN];
11 long long solveCongruence(int n) {
12     long long res = 0, K = 1;
13     for (int i = 0; i < n; i++) {
14         long long x, y, g;
15         exgcd(K, mod[i], g, x, y);
16         if ((rem[i] - res) % g) return -1;
17         x = (x * (rem[i] - res) / g + mod[i] / g) % (mod[i] / g);
18         y = (K / g * mod[i]);
19         res = ((x * K + res) % y + y) % y;
20         K = y;
21     }
22     return res;
23 }
24
25 int main() {
26     int n;
27     scanf("%d", &n);
28     for (int i = 0; i < n; i++) scanf("%d %d", &mod[i], &rem[i]);
29     long long ans = solveCongruence(n);
30     printf("%lld\n", ans);
31
32     return 0;
33 }
```

5.8 BSGS

```

1 #include <cstdio>
2 #include <cmath>
3 #include <map>
4
```

```

5  long long pow(long long a, long long n, long long p) {
6      long long res = 1;
7      for (; n; n >= 1, a = a * a % p) if (n & 1) res = res * a % p;
8      return res;
9  }
10
11 long long inv(long long a, long long p) {
12     return pow(a, p - 2, p);
13 }
14
15 long long bsgs(long long a, long long b, long long p) {
16     a %= p, b %= p;
17     if (a == 0) return b == 0 ? 1 : -1;
18
19     std::map<long long, long long> map;
20
21     long long m = std::ceil(std::sqrt(p)), t = 1;
22     for (int i = 0; i < m; i++) {
23         if (!map.count(t)) map[t] = i;
24         t = t * a % p;
25     }
26
27     long long k = inv(t, p), w = b;
28     for (int i = 0; i < m; i++) {
29         if (map.count(w)) return i * m + map[w];
30         w = w * k % p;
31     }
32
33     return -1;
34 }
35
36 int main () {
37     long long a, b, p;
38     scanf("%lld %lld %lld", &a, &b, &p);
39
40     long long ans = bsgs(a, b, p);
41     printf("%lld\n", ans);
42
43     return 0;
44 }
```

5.9 二次剩余

```

1 #include <cstdio>
2 #include <tuple>
3
4 long long qpow(long long a, long long n, long long p) {
5     long long res = 1;
6     for (; n; n >= 1, a = a * a % p) if (n & 1) res = res * a % p;
7     return res;
8 }
9
10 int modSqrt(int a, int p) {
11     if (p == 2) return a % p;
```

```

12     int x;
13     if (qpow(a, (p - 1) >> 1, p) == 1) {
14         if (p % 4 == 3) {
15             x = qpow(a, (p + 1) >> 2, p);
16         } else {
17             long long w;
18             for (w = 1; qpow((w * w - a + p) % p, (p - 1) >> 1, p) == 1; w++) {}
19             long long b0 = w, b1 = 1;
20             w = (w * w - a + p) % p;
21             long long r0 = 1, r1 = 0;
22             int exp = (p + 1) >> 1;
23             for (; exp; exp >>= 1) {
24                 if (exp & 1)
25                     std::tie(r0, r1) = std::make_tuple((r0 * b0 + r1 * b1 % p * w) % p, (r0 * b1 + r1 * b0) % p);
26                     std::tie(b0, b1) = std::make_tuple((b0 * b0 + b1 * b1 % p * w) % p, 2 * b0 * b1 % p);
27             }
28             x = r0;
29         }
30         if (x * 2 > p) x = p - x;
31         return x;
32     }
33     return -1;
34 }
35
36 int main() {
37     int n, p;
38     scanf("%d %d", &n, &p);
39     int ans = modSqrt(n, p);
40     printf("%d\n", ans);
41
42     return 0;
43 }
```

5.10 线性基

5.10.1 在线版本

```

1 #include <cstdio>
2
3 const int MAXL = 50;
4
5 struct LinearBasis {
6     long long a[MAXL + 1];
7
8     bool extend(long long t) {
9         for (int i = MAXL; ~i; i--) {
10             if (!(t & (1ll << i))) continue;
11             if (!t) return false;
12
13             if (a[i]) t ^= a[i];
14             else {
15                 for (int j = 0; j < i; j++) if (t & (1ll << j)) t ^= a[j];
16                 for (int j = i + 1; j <= MAXL; j++) if (a[j] & (1ll << i)) a[j] ^= t;
17             }
18         }
19     }
20
21     void print() {
22         for (int i = 0; i < MAXL + 1; i++) cout << a[i] << " ";
23     }
24 }
```

```

17         a[i] = t;
18         return true;
19     }
20 }
21
22     return false;
23 }
24
25 long long queryMax() {
26     long long res = 0;
27     for (int i = 0; i <= MAXL; i++) res ^= a[i];
28     return res;
29 }
30 } lb;
31
32 int main() {
33     int n;
34     scanf("%d", &n);
35
36     for (int i = 0; i < n; i++) {
37         long long x;
38         scanf("%lld", &x);
39         lb.extend(x);
40     }
41
42     printf("%lld\n", lb.queryMax());
43
44     return 0;
45 }
```

5.10.2 离线版本

```

1 #include <cstdio>
2 #include <vector>
3 #include <algorithm>
4
5 const int MAXN = 100005;
6 const int MAXL = 50;
7
8 struct LinearBasis {
9     std::vector<long long> v;
10    int n;
11
12    void init(long long *x, int n) {
13        this->n = n;
14        static long long a[MAXL + 1];
15
16        for (int i = 0; i < n; i++) {
17            long long t = x[i];
18
19            for (int j = MAXL; ~j; j--) {
20                if (!(t & (1ll << j))) continue;
21                if (!t) break;
22
23                if (a[j]) t ^= a[j];
24            }
25        }
26    }
27
28    long long extend(long long x) {
29        if (x == 0) return 0;
30
31        long long t = x;
32        for (int j = MAXL; ~j; j--) {
33            if (t & (1ll << j)) return j;
34        }
35        return -1;
36    }
37
38    long long queryMax() {
39        long long res = 0;
40        for (int i = 0; i < n; i++) res ^= v[i];
41        return res;
42    }
43
44    void update(int i, long long x) {
45        v[i] = x;
46    }
47
48    void clear() {
49        n = 0;
50        v.clear();
51    }
52
53    long long operator[](int i) {
54        return v[i];
55    }
56}
```

```

24     else {
25         for (int k = 0; k < j; k++) if (t & (1ll << k)) t ^= a[k];
26         for (int k = j + 1; k <= MAXL; k++) if (a[k] & (1ll << j)) a[k] ^= t;
27         a[j] = t;
28         break;
29     }
30 }
31 }
32
33 v.clear();
34 for (int i = 0; i <= MAXL; i++) if (a[i]) v.push_back(a[i]);
35 }
36
37 long long query(int k) {
38     if (v.size() != n) k--;
39     if (k >= (1ll << v.size())) return -1;
40
41     long long res = 0;
42     for (int i = 0; i < v.size(); i++) if (k & (1ll << i)) res ^= v[i];
43     return res;
44 }
45 } lb;
46
47 int main() {
48     int n;
49     scanf("%d", &n);
50
51     static long long a[MAXN];
52     for (int i = 0; i < n; i++) scanf("%lld", &a[i]);
53     lb.init(a, n);
54
55     int q;
56     scanf("%d", &q);
57     while (q--) {
58         long long k;
59         scanf("%lld", &k);
60         printf("%lld\n", lb.query(k));
61     }
62     return 0;
63 }
```

5.11 高斯消元

```

1 #include <cstdio>
2 #include <cmath>
3 #include <algorithm>
4
5 const int MAXN = 105;
6 const double EPS = 1e-8;
7
8 int dcmp(double a, double b = 0.0) {
9     double d = a - b;
10    return std::abs(d) <= EPS ? 0 : (d > 0 ? 1 : -1);
11 }
```

```

12
13 namespace GaussJordan {
14     double a[MAXN][MAXN];
15
16     bool solve(int n) {
17         for (int i = 0; i < n; i++) {
18             int max = i;
19             for (int j = i + 1; j < n; j++)
20                 if (dcmp(std::abs(a[j][i]), std::abs(a[max][i])) > 0) max = j;
21
22             if (!dcmp(a[max][i])) return false;
23             if (max != i) for (int j = i; j <= n; j++) std::swap(a[i][j], a[max][j]);
24
25             for (int j = 0; j < n; j++) if (i != j) for (int k = n; k >= i; k--)
26                 a[j][k] -= a[i][k] / a[i][i] * a[j][i];
27         }
28
29         return true;
30     }
31 }
32
33 int main() {
34     int n;
35     scanf("%d", &n);
36
37     using GaussJordan::a;
38
39     for (int i = 0; i < n; i++) for (int j = 0; j <= n; j++)
40         scanf("%lf", &a[i][j]);
41
42     if (!GaussJordan::solve(n)) puts("-1");
43     else for (int i = 0; i < n; i++) printf("%.4lf\n", a[i][n] / a[i][i]);
44
45     return 0;
46 }
```

5.12 Berlekamp-Massey

```

1 #include <cstdio>
2 #include <vector>
3 #include <algorithm>
4
5 const int MAXN = 2505;
6 const int MOD = 998244353;
7
8 long long qpow(long long a, long long n) {
9     long long res = 1;
10    for (; n >= 1, a = a * a % MOD) if (n & 1) res = res * a % MOD;
11    return res;
12 }
13
14 class BerlekampMassey {
15 public:
16     std::vector<int> solve(int *x, int n) {
```

```

17     int L = 0, m = 1, b = 1;
18     C = B = { 1 };
19     for (int i = 0; i < n; i++) {
20         int d = x[i];
21         for (int j = 1; j <= L; j++) d = (d + 1ll * C[j] * x[i - j]) % MOD;
22         if (d == 0) {
23             ++m;
24         } else if (2 * L <= i) {
25             T = C;
26             C.resize(i + 2 - L);
27             int ib = qpow(b, MOD - 2);
28             d = MOD - d;
29             for (int i = 0; i < B.size(); i++) C[i + m] = (C[i + m] + 1ll * d * ib % MOD * B[i]) % MOD;
30             L = i + 1 - L;
31             B.swap(T);
32             b = MOD - d;
33             m = 1;
34         } else {
35             int ib = qpow(b, MOD - 2);
36             d = MOD - d;
37             for (int i = 0; i < B.size(); i++) C[i + m] = (C[i + m] + 1ll * d * ib % MOD * B[i]) % MOD;
38             ++m;
39         }
40     }
41     for (int i = 1; i <= L; i++) C[i] = (MOD - C[i]) % MOD;
42     return C;
43 }
44
45 private:
46     std::vector<int> C, B, T;
47 } bm;
48
49 int x[MAXN];
50
51 int main() {
52     int n;
53     scanf("%d", &n);
54     for (int i = 0; i < n; i++) scanf("%d", &x[i]);
55
56     auto f = bm.solve(x, n);
57     for (int i = 1; i < f.size(); i++) { // from 1
58         printf("%d%c", f[i], " \n"[i == f.size() - 1]);
59     }
60
61     return 0;
62 }
```

5.13 Miller-Rabin

```

1 long long pow(long long a, long long n, long long mod) {
2     long long res = 1;
3     for (; n; n >= 1, a = mul(a, a, mod)) if (n & 1) res = mul(res, a, mod);
4     return res;
5 }
```

```

6
7 bool isPrime(long long n) {
8     const static int primes[12] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37};
9
10    long long s = 0, d = n - 1;
11    while (d % 2 == 0) d /= 2, s++;
12    if (s == 0) return n == 2;
13
14    for (int i = 0; i < 12 && primes[i] < n; i++) {
15        long long a = primes[i];
16        if (pow(a, d, n) != 1) {
17            bool flag = true;
18            for (int r = 0; r < s; r++) {
19                if (flag && pow(a, d * (1 << r), n) == n - 1) flag = false;
20            if (flag) return false;
21        }
22    }
23
24    return true;
25 }
```

5.14 Pollard's Rho

```

1 long long gcd(long long a, long long b) {
2     return b ? gcd(b, a % b) : a;
3 }
4
5 namespace PollardRho {
6     long long g(long long x, long long n, long long c) {
7         return (mul(x, x, n) + c) % n;
8     }
9
10    long long rho(long long n, long long c) {
11        long long x = rand() % n, y = x, d = 1;
12
13        for (long long i = 1, k = 2; d == 1; i++) {
14            x = g(x, n, c);
15            d = gcd(x > y ? x - y : y - x, n);
16            if (x == y) return n;
17            if (i == k) k <= 1, y = x;
18        }
19
20        return d;
21    }
22
23    void find(long long n, long long c, std::map<long long, int> &res) {
24        if (n == 1) return;
25        if (isPrime(n)) {
26            res[n]++;
27            return;
28        }
29
30        long long p = n;
31        while (p == n) p = rho(p, c++);
```

```

32     find(p, c, res);
33     find(n / p, c, res);
34 }
35
36 std::map<long long, int> divide(long long n) {
37     static std::map<long long, int> res;
38     res.clear();
39     find(n, 1, res);
40     return res;
41 }
42 }
```

5.15 快速傅立叶变换

5.15.1 FFT

```

1 #include <cstdio>
2 #include <cmath>
3 #include <algorithm>
4
5 const int MAXN = 262144 + 1;
6 const double PI = std::acos(-1.0);
7
8 struct Complex {
9     double r, i;
10
11     Complex(double r = 0, double i = 0) : r(r), i(i) {}
12
13     Complex conj() const { return Complex(r, -i); }
14
15     Complex operator+(const Complex &rhs) const { return Complex(r + rhs.r, i + rhs.i); }
16     Complex operator-(const Complex &rhs) const { return Complex(r - rhs.r, i - rhs.i); }
17     Complex operator*(const Complex &rhs) const { return Complex(r * rhs.r - i * rhs.i, r * rhs.i + i * rhs
18 .r); }
19     Complex operator/(double rhs) const { return Complex(r / rhs, i / rhs); }
20 };
21
22 class FFT {
23 private:
24     static const int N = 262144;
25
26     Complex omega[N + 1], omegaInv[N + 1];
27
28     void init() {
29         for (int i = 0; i < N; i++) {
30             omega[i] = Complex(std::cos(2 * PI / N * i), std::sin(2 * PI / N * i));
31             omegaInv[i] = omega[i].conj();
32         }
33     }
34
35     void reverse(Complex *a, int n) {
36         for (int i = 0, j = 0; i < n; i++) {
37             if (i < j) std::swap(a[i], a[j]);
38             for (int l = n >> 1; (j ^= l) < l; l >>= 1) {}
```

```
38         }
39     }
40
41     void transform(Complex *a, int n, Complex *omega) {
42         reverse(a, n);
43
44         for (int l = 2; l <= n; l <= 1) {
45             int hl = l >> 1;
46             for (Complex *x = a; x != a + n; x += l) {
47                 for (int i = 0; i < hl; i++) {
48                     Complex t = omega[N / l * i] * x[i + hl];
49                     x[i + hl] = x[i] - t;
50                     x[i] = x[i] + t;
51                 }
52             }
53         }
54     }
55
56 public:
57     FFT() { init(); }
58
59     int extend(int n) {
60         int res = 1;
61         while (res < n) res <= 1;
62         return res;
63     }
64
65     void dft(Complex *a, int n) {
66         transform(a, n, omega);
67     }
68
69     void idft(Complex *a, int n) {
70         transform(a, n, omegaInv);
71         for (int i = 0; i < n; i++) a[i] = a[i] / n;
72     }
73 } fft;
74
75 int main() {
76     int n, m;
77     scanf("%d %d", &n, &m);
78
79     static Complex a[MAXN], b[MAXN];
80     for (int i = 0; i <= n; i++) scanf("%lf", &a[i].r);
81     for (int i = 0; i <= m; i++) scanf("%lf", &b[i].r);
82
83     int N = fft.extend(n + m + 1);
84
85     fft.dft(a, N);
86     fft.dft(b, N);
87     for (int i = 0; i < N; i++) a[i] = a[i] * b[i];
88     fft.idft(a, N);
89
90     for (int i = 0; i < n + m + 1; i++)
91         printf("%.0lf%c", a[i].r + 0.001, " \n"[i == n + m]);
92
93     return 0;
94 }
```

94 }

5.15.2 两次 DFT 的多项式乘法

注：精度低于普通 FFT

```

1 #include <cstdio>
2 #include <cmath>
3 #include <algorithm>
4
5 const int MAXN = 262144 + 1;
6 const double PI = std::acos(-1.0);
7
8 struct Complex {
9     double r, i;
10
11    Complex(double r = 0, double i = 0) : r(r), i(i) {}
12
13    Complex conj() const { return Complex(r, -i); }
14
15    Complex operator+(const Complex &rhs) const { return Complex(r + rhs.r, i + rhs.i); }
16    Complex operator-(const Complex &rhs) const { return Complex(r - rhs.r, i - rhs.i); }
17    Complex operator*(const Complex &rhs) const { return Complex(r * rhs.r - i * rhs.i, r * rhs.i + i * rhs
18 .r); }
19    Complex operator/(double rhs) const { return Complex(r / rhs, i / rhs); }
20 };
21
22 class FFT {
23 private:
24     static const int N = 262144;
25
26     Complex omega[N + 1], omegaInv[N + 1];
27
28     void init() {
29         for (int i = 0; i < N; i++) {
30             omega[i] = Complex(std::cos(2 * PI / N * i), std::sin(2 * PI / N * i));
31             omegaInv[i] = omega[i].conj();
32         }
33     }
34
35     void reverse(Complex *a, int n) {
36         for (int i = 0, j = 0; i < n; i++) {
37             if (i < j) std::swap(a[i], a[j]);
38             for (int l = n >> 1; (j ^= l) < l; l >>= 1) {}
39         }
40     }
41
42     void transform(Complex *a, int n, Complex *omega) {
43         reverse(a, n);
44
45         for (int l = 2; l <= n; l <= 1) {
46             int hl = l >> 1;
47             for (Complex *x = a; x != a + n; x += l) {
48                 for (int i = 0; i < hl; i++) {
49                     Complex t = omega[N / l * i] * x[i + hl];
50                     x[i + hl] = x[i] - t;
51                 }
52             }
53         }
54     }
55 }
```

```

50             x[i] = x[i] + t;
51         }
52     }
53 }
54 }
55
56 public:
57     FFT() { init(); }
58
59     int extend(int n) {
60         int res = 1;
61         while (res < n) res <= 1;
62         return res;
63     }
64
65     void dft(Complex *a, int n) {
66         transform(a, n, omega);
67     }
68
69     void idft(Complex *a, int n) {
70         transform(a, n, omegaInv);
71         for (int i = 0; i < n; i++) a[i] = a[i] / n;
72     }
73 } fft;
74
75 int main() {
76     int n, m;
77     scanf("%"d "%d", &n, &m);
78
79     static Complex a[MAXN], b[MAXN];
80
81     for (int i = 0, x; i <= n; i++) {
82         scanf("%"d, &x);
83         a[i] = Complex(x, 0);
84     }
85
86     for (int i = 0, x; i <= m; i++) {
87         scanf("%"d, &x);
88         a[i] = a[i] + Complex(0, x);
89     }
90
91     int len = n + m + 1;
92     int N = fft.extend(len);
93
94     fft.dft(a, N);
95     for (int i = 1; i < N; i++) {
96         double x1 = a[i].r, y1 = a[i].i;
97         double x2 = a[N - i].r, y2 = a[N - i].i;
98         Complex t1((x1 + x2) * 0.5, (y1 - y2) * 0.5);
99         Complex t2((y1 + y2) * 0.5, (x2 - x1) * 0.5);
100        b[i] = t1 * t2;
101    }
102    b[0] = a[0].r * a[0].i;
103    fft.idft(b, N);
104
105    for (int i = 0; i < len; i++)

```

```

106     printf("%d%c", (int) (round(b[i].r)), " \n"[i == len - 1]);
107
108     return 0;
109 }

```

5.15.3 拆系数 FFT

```

1 #include <cstdio>
2 #include <cmath>
3 #include <algorithm>
4
5 const int MAXN = 262144 + 1;
6 const double PI = std::acos(-1.0);
7 const int MOD = 1000000007;
8
9 struct Complex {
10     double r, i;
11
12     Complex(double r = 0, double i = 0) : r(r), i(i) {}
13
14     Complex conj() const { return Complex(r, -i); }
15
16     Complex operator-(const Complex &rhs) const { return Complex(r - rhs.r, i - rhs.i); }
17     Complex operator+(const Complex &rhs) const { return Complex(r + rhs.r, i + rhs.i); }
18     Complex operator*(const Complex &rhs) const { return Complex(r * rhs.r - i * rhs.i, r * rhs.i + i * rhs
19                     .r); }
20     Complex operator/(double rhs) const { return Complex(r / rhs, i / rhs); }
21 };
22
23 class FFT {
24 private:
25     static const int N = 262144;
26
27     Complex omega[N + 1], omegaInv[N + 1];
28
29     void init() {
30         for (int i = 0; i < N; i++) {
31             omega[i] = Complex(std::cos(2 * PI / N * i), std::sin(2 * PI / N * i));
32             omegaInv[i] = omega[i].conj();
33         }
34     }
35
36     void reverse(Complex *a, int n) {
37         for (int i = 0, j = 0; i < n; i++) {
38             if (i < j) std::swap(a[i], a[j]);
39             for (int l = n >> 1; (j ^= l) < l; l >>= 1) {}
40         }
41     }
42
43     void transform(Complex *a, int n, Complex *omega) {
44         reverse(a, n);
45
46         for (int l = 2; l <= n; l <= 1) {
47             int hl = l >> 1;
48             for (Complex *x = a; x != a + n; x += l) {

```

```

48         for (int i = 0; i < hl; i++) {
49             Complex t = omega[N / l * i] * x[i + hl];
50             x[i + hl] = x[i] - t;
51             x[i] = x[i] + t;
52         }
53     }
54 }
55 }
56
57 public:
58 FFT() { init(); }
59
60 int extend(int n) {
61     int res = 1;
62     while (res < n) res <<= 1;
63     return res;
64 }
65
66 void dft(Complex *a, int n) {
67     transform(a, n, omega);
68 }
69
70 void idft(Complex *a, int n) {
71     transform(a, n, omegaInv);
72     for (int i = 0; i < n; i++) a[i] = a[i] / n;
73 }
74 } fft;
75
76 void polyMul(long long *a, long long *b, int n, long long *res) {
77     static Complex a0[MAXN], a1[MAXN], b0[MAXN], b1[MAXN];
78     static const int M = (1 << 15) - 1;
79
80     for (int i = 0; i < n; i++) {
81         a0[i] = a[i] >> 15;
82         a1[i] = a[i] & M;
83         b0[i] = b[i] >> 15;
84         b1[i] = b[i] & M;
85     }
86     fft.dft(a0, n), fft.dft(a1, n);
87     fft.dft(b0, n), fft.dft(b1, n);
88     for (int i = 0; i < n; i++) {
89         Complex _a = a0[i], _b = a1[i], _c = b0[i], _d = b1[i];
90         a0[i] = _a * _c;
91         a1[i] = _a * _d + _b * _c;
92         b0[i] = _b * _d;
93     }
94     fft.idft(a0, n), fft.idft(a1, n), fft.idft(b0, n);
95     for (int i = 0; i < n; i++) {
96         res[i] = (((long long) (a0[i].r + 0.5) % MOD) << 30) % MOD
97             + (((long long) (a1[i].r + 0.5) % MOD) << 15) % MOD
98             + (long long) (b0[i].r + 0.5) % MOD) % MOD;
99     }
100 }
101
102 int main() {
103     int n, m;

```

```

104     scanf("%d %d", &n, &m);
105
106     static long long a[MAXN], b[MAXN], ans[MAXN];
107     for (int i = 0; i <= n; i++) scanf("%lld", &a[i]);
108     for (int i = 0; i <= m; i++) scanf("%lld", &b[i]);
109     for (int i = 0; i <= n; i++) a[i] < 0 ? a[i] += MOD : 0;
110     for (int i = 0; i <= m; i++) b[i] < 0 ? b[i] += MOD : 0;
111
112     int len = n + m + 1;
113     int p = fft.extend(len);
114     polyMul(a, b, p, ans);
115     for (int i = 0; i < len; i++)
116         printf("%lld%c", ans[i], " \n"[i == len - 1]);
117
118     return 0;
119 }
```

5.15.4 NTT

```

1 #include <cstdio>
2 #include <algorithm>
3
4 const int MAXN = 262144 + 1;
5 const int MOD = 998244353;
6 const int G = 3;
7
8 long long qpow(long long a, long long n) {
9     long long res = 1;
10    for (; n; n >>= 1, a = a * a % MOD) if (n & 1) res = res * a % MOD;
11    return res;
12 }
13
14 long long inv(long long x) {
15     return qpow(x, MOD - 2);
16 }
17
18 class NTT {
19     static const int N = 262144;
20
21     long long omega[N + 1], omegaInv[N + 1];
22
23     void init() {
24         long long g = qpow(G, (MOD - 1) / N), ig = inv(g);
25         omega[0] = omegaInv[0] = 1;
26         for (int i = 1; i < N; i++) {
27             omega[i] = omega[i - 1] * g % MOD;
28             omegaInv[i] = omegaInv[i - 1] * ig % MOD;
29         }
30     }
31
32     void reverse(long long *a, int n) {
33         for (int i = 0, j = 0; i < n; i++) {
34             if (i < j) std::swap(a[i], a[j]);
35             for (int l = n >> 1; (j ^= l) < l; l >>= 1) {}
36         }
37     }
38 }
```

```

37     }
38
39     void transform(long long *a, int n, long long *omega) {
40         reverse(a, n);
41
42         for (int l = 2; l <= n; l <= 1) {
43             int hl = l >> 1;
44             for (long long *x = a; x != a + n; x += l) {
45                 for (int i = 0; i < hl; i++) {
46                     long long t = omega[N / l * i] * x[i + hl] % MOD;
47                     x[i + hl] = (x[i] - t + MOD) % MOD;
48                     x[i] += t;
49                     x[i] >= MOD ? x[i] -= MOD : 0;
50                 }
51             }
52         }
53     }
54
55     public:
56     NTT() { init(); }
57
58     int extend(int n) {
59         int res = 1;
60         while (res < n) res <= 1;
61         return res;
62     }
63
64     void dft(long long *a, int n) {
65         transform(a, n, omega);
66     }
67
68     void idft(long long *a, int n) {
69         transform(a, n, omegaInv);
70         long long t = inv(n);
71         for (int i = 0; i < n; i++) a[i] = a[i] * t % MOD;
72     }
73 } ntt;
74
75 int main() {
76     int n, m;
77     scanf("%d %d", &n, &m);
78
79     static long long a[MAXN], b[MAXN];
80     for (int i = 0; i <= n; i++) scanf("%lld", &a[i]);
81     for (int i = 0; i <= m; i++) scanf("%lld", &b[i]);
82
83     int N = ntt.extend(n + m + 1);
84
85     ntt.dft(a, N);
86     ntt.dft(b, N);
87     for (int i = 0; i < N; i++) a[i] = a[i] * b[i] % MOD;
88     ntt.idft(a, N);
89
90     for (int i = 0; i < n + m + 1; i++)
91         printf("%lld%c", a[i], " \n"[i == n + m]);
92

```

```
93     return 0;  
94 }
```

5.15.5 NTT 模数及原根表

$r \cdot 2^k + 1$	r	k	g
3	1	1	2
5	1	2	2
17	1	4	3
97	3	5	5
193	3	6	5
257	1	8	3
7681	15	9	17
12289	3	12	11
40961	5	13	3
65537	1	16	3
786433	3	18	10
5767169	11	19	3
7340033	7	20	3
23068673	11	21	3
104857601	25	22	3
167772161	5	25	3
469762049	7	26	3
998244353	110	23	3
1004535809	479	21	3
2013265921	15	27	31
2281701377	17	27	3
3221225473	3	30	5
75161927681	35	31	3
77309411329	9	33	7
206158430209	3	36	22
2061584302081	15	37	7

表 5.1: NTT 模数表

5.16 多项式逆元

若不需要写拆系数 FFT，或可以使用 NTT，则两次多项式乘法可用一次代替。

```

1 #include <cstdio>
2 #include <cmath>
3 #include <algorithm>
4
5 const int MAXN = 262144 + 1;
6 const double PI = std::acos(-1.0);
7 const int MOD = 1000000007;
8
9 struct Complex {
10     double r, i;
11
12     Complex(double r = 0, double i = 0) : r(r), i(i) {}
13
14     Complex conj() const { return Complex(r, -i); }
15
16     Complex operator-(const Complex &rhs) const { return Complex(r - rhs.r, i - rhs.i); }
17     Complex operator+(const Complex &rhs) const { return Complex(r + rhs.r, i + rhs.i); }
18     Complex operator*(const Complex &rhs) const { return Complex(r * rhs.r - i * rhs.i, r * rhs.i + i * rhs
19         .r); }
20     Complex operator/(double rhs) const { return Complex(r / rhs, i / rhs); }
21 };
22
23 class FFT {
24 private:
25     static const int N = 262144;
26
27     Complex omega[N + 1], omegaInv[N + 1];
28
29     void init() {
30         double per = 2 * PI / N;
31         for (int i = 0; i < N; i++) {
32             omega[i] = Complex(std::cos(i * per), std::sin(i * per));
33             omegaInv[i] = omega[i].conj();
34         }
35     }
36
37     void reverse(Complex *a, int n) {
38         for (int i = 0, j = 0; i < n; i++) {
39             if (i < j) std::swap(a[i], a[j]);
40             for (int l = n >> 1; (j ^= l) < l; l >>= 1) {}
41         }
42     }
43
44     void transform(Complex *a, int n, Complex *omega) {
45         reverse(a, n);
46
47         for (int l = 2; l <= n; l <= 1) {
48             int hl = l >> 1;
49             for (Complex *x = a; x != a + n; x += l) {
50                 for (int i = 0; i < hl; i++) {
51                     Complex t = omega[N / l * i] * x[i + hl];
52                     x[i + hl] = x[i] - t;
53                 }
54             }
55         }
56     }
57 }
```

```

52             x[i] = x[i] + t;
53         }
54     }
55 }
56 }
57
58 public:
59     FFT() { init(); }
60
61     int extend(int n) {
62         int res = 1;
63         while (res < n) res <<= 1;
64         return res;
65     }
66
67     void dft(Complex *a, int n) {
68         transform(a, n, omega);
69     }
70
71     void idft(Complex *a, int n) {
72         transform(a, n, omegaInv);
73         for (int i = 0; i < n; i++) a[i] = a[i] / n;
74     }
75 } fft;
76
77 void polyMul(long long *a, long long *b, int n, long long *res) {
78     static Complex a0[MAXN], a1[MAXN], b0[MAXN], b1[MAXN];
79     static const int M = (1 << 15) - 1;
80
81     for (int i = 0; i < n; i++) {
82         a0[i] = a[i] >> 15;
83         a1[i] = a[i] & M;
84         b0[i] = b[i] >> 15;
85         b1[i] = b[i] & M;
86     }
87     fft.dft(a0, n), fft.dft(a1, n);
88     fft.dft(b0, n), fft.dft(b1, n);
89     for (int i = 0; i < n; i++) {
90         Complex _a = a0[i], _b = a1[i], _c = b0[i], _d = b1[i];
91         a0[i] = _a * _c;
92         a1[i] = _a * _d + _b * _c;
93         b0[i] = _b * _d;
94     }
95     fft.idft(a0, n), fft.idft(a1, n), fft.idft(b0, n);
96     for (int i = 0; i < n; i++) {
97         res[i] = (((long long) (a0[i].r + 0.5) % MOD) << 30) % MOD
98             + (((long long) (a1[i].r + 0.5) % MOD) << 15) % MOD
99             + (long long) (b0[i].r + 0.5) % MOD;
100    }
101 }
102
103 void polyInverse(long long *a, long long *res, int k) {
104     if (k == 1) {
105         res[0] = 1;
106         return;
107     }

```

```

108     polyInverse(a, res, (k + 1) >> 1);
109
110     static long long t1[MAXN], t2[MAXN];
111     int N = fft.extend(k << 1);
112     std::copy(a, a + k, t1);
113     std::fill(t1 + k, t1 + N, 0);
114     polyMul(res, res, N, t2);
115     polyMul(t1, t2, N, t1);
116     for (int i = 0; i < k; i++) res[i] = (2 * res[i] % MOD - t1[i] + MOD) % MOD;
117     std::fill(res + k, res + N, 0);
118 }
119
120 long long a[MAXN], res[MAXN];
121
122 int main() {
123     int n, k;
124     scanf("%d %d", &n, &k);
125     for (int i = 0; i <= n; i++) scanf("%lld", &a[i]);
126
127     polyInverse(a, res, k);
128     for (int i = 0; i < k; i++) printf("%lld%c", res[i], " \n"[i == k - 1]);
129
130     return 0;
131 }
```

5.17 挑战多项式（多项式逆元、除法、平方根、ln、exp、快速幂）

```

1 #include <cstdio>
2 #include <vector>
3 #include <tuple>
4 #include <algorithm>
5
6 const int MAXN = 262144 + 1;
7 const int MOD = 998244353;
8 const int G = 3;
9
10 long long qpow(long long a, long long n) {
11     long long res = 1;
12     for (; n; n >= 1, a = a * a % MOD) if (n & 1) res = res * a % MOD;
13     return res;
14 }
15
16 int numinv[MAXN];
17 void init() {
18     numinv[1] = 1;
19     for (int i = 2; i < MAXN; i++) numinv[i] = (long long) (MOD - MOD / i) * numinv[MOD % i] % MOD;
20 }
21
22 long long inv(long long x) {
23     return x < MAXN ? numinv[x] : qpow(x, MOD - 2);
24 }
25
26 class NTT {
27 private:
```

```

28     static const int N = 262144;
29
30     long long omega[N + 1], omegaInv[N + 1];
31
32     void init() {
33         long long g = qpow(G, (MOD - 1) / N), ig = inv(g);
34         omega[0] = omegaInv[0] = 1;
35         for (int i = 1; i < N; i++) {
36             omega[i] = omega[i - 1] * g % MOD;
37             omegaInv[i] = omegaInv[i - 1] * ig % MOD;
38         }
39     }
40
41     void reverse(long long *a, int n) const {
42         for (int i = 0, j = 0; i < n; i++) {
43             if (i < j) std::swap(a[i], a[j]);
44             for (int l = n >> 1; (j ^= l) < l; l >>= 1) {}
45         }
46     }
47
48     void transform(long long *a, int n, const long long *omega) const {
49         reverse(a, n);
50
51         for (int l = 2; l <= n; l <= 1) {
52             int hl = l >> 1;
53             for (long long *x = a; x != a + n; x += l) {
54                 for (int i = 0; i < hl; i++) {
55                     long long t = omega[N / l * i] * x[i + hl] % MOD;
56                     x[i + hl] = (x[i] - t + MOD) % MOD;
57                     x[i] += t;
58                     x[i] >= MOD ? x[i] -= MOD : 0;
59                 }
60             }
61         }
62     }
63
64     public:
65     NTT() {
66         init();
67     }
68
69     int extend(int n) const {
70         int res = 1;
71         while (res < n) res <= 1;
72         return res;
73     }
74
75     void dft(long long *a, int n) const {
76         transform(a, n, omega);
77     }
78
79     void idft(long long *a, int n) const {
80         transform(a, n, omegaInv);
81         long long t = inv(n);
82         for (int i = 0; i < n; i++) a[i] = a[i] * t % MOD;
83     }

```

```

84 } ntt;
85
86 int modSqrt(int a, int p = MOD) {
87     if (p == 2) return a % p;
88     int x;
89     if (qpow(a, (p - 1) >> 1) == 1) {
90         if (p % 4 == 3) {
91             x = qpow(a, (p + 1) >> 2);
92         } else {
93             long long w;
94             for (w = 1; qpow((w * w - a + p) % p, (p - 1) >> 1) == 1; w++) {}
95             long long b0 = w, b1 = 1;
96             w = (w * w - a + p) % p;
97             long long r0 = 1, r1 = 0;
98             int exp = (p + 1) >> 1;
99             for (; exp; std::tie(b0, b1) = std::make_tuple((b0 * b0 + b1 * b1 % p * w) % p, 2 * b0 * b1 % p
100                  ), exp >>= 1) {
101                 if (exp & 1)
102                     std::tie(r0, r1) = std::make_tuple((r0 * b0 + r1 * b1 % p * w) % p, (r0 * b1 + r1 * b0)
103                                              % p);
104             }
105             x = r0;
106         }
107     }
108     return -1;
109 }
110
111 class Poly : public std::vector<long long> {
112 public:
113     using std::vector<long long>::vector;
114
115     static void dft(Poly &a, int n) {
116         a.resize(n);
117         ntt.dft(a.data(), n);
118     }
119     static void idft(Poly &a, int n) {
120         a.resize(n);
121         ntt.idft(a.data(), n);
122     }
123
124     Poly operator+(const Poly &rhs) const { // not commutative
125         Poly res = *this;
126         for (int i = 0; i < size(); i++) res[i] = (res[i] + rhs[i]) % MOD;
127         return res;
128     }
129
130     Poly operator*(const Poly &rhs) const {
131         if (size() < BRUTE_LIM || rhs.size() < BRUTE_LIM) {
132             Poly res(size() + rhs.size() - 1);
133             for (int i = 0; i < size(); i++)
134                 for (int j = 0; j < rhs.size(); j++)
135                     res[i + j] = (res[i + j] + (*this)[i] * rhs[j]) % MOD;
136             return res;
137         }

```

```

138     Poly t1 = *this, t2 = rhs;
139     int n = t1.size() + t2.size() - 1;
140     int N = ntt.extend(n);
141     dft(t1, N);
142     dft(t2, N);
143     Poly res(N);
144     for (int i = 0; i < N; i++) res[i] = t1[i] * t2[i] % MOD;
145     idft(res, N);
146     res.resize(n);
147     return res;
148 }
149
150 static Poly inv(const Poly &a, int k = -1) {
151     if (k == -1) k = a.size();
152     if (k == 1) return { ::inv(a[0]) };
153     Poly b = inv(a, (k + 1) >> 1), temp(a.begin(), a.begin() + k);
154     int N = ntt.extend(2 * k - 1);
155     dft(b, N);
156     dft(temp, N);
157     Poly res(N);
158     for (int i = 0; i < N; i++) res[i] = (MOD + 2 - b[i] * temp[i] % MOD) * b[i] % MOD;
159     idft(res, N);
160     res.resize(k);
161     return res;
162 }
163
164 static void div(const Poly &a, const Poly &b, Poly &d, Poly &r) {
165     if (b.size() > a.size()) {
166         d.clear();
167         r = a;
168         return;
169     }
170
171     int n = a.size(), m = b.size();
172
173     Poly A = a, B = b;
174     std::reverse(A.begin(), A.end());
175     std::reverse(B.begin(), B.end());
176     B.resize(n - m + 1);
177     Poly iB = inv(B, n - m + 1);
178     d = A * iB;
179     d.resize(n - m + 1);
180     std::reverse(d.begin(), d.end());
181
182     r = b * d;
183     r.resize(m - 1);
184     for (int i = 0; i < m - 1; i++) r[i] = (a[i] - r[i] + MOD) % MOD;
185 }
186
187 static Poly derivative(const Poly &a) {
188     Poly res(a.size() - 1);
189     for (int i = 1; i < a.size(); i++) res[i - 1] = a[i] * i % MOD;
190     return res;
191 }
192
193 static Poly integral(const Poly &a) {

```

```

194     Poly res(a.size() + 1);
195     for (int i = 0; i < a.size(); i++) res[i + 1] = a[i] * ::inv(i + 1) % MOD;
196     return res;
197 }
198
199 static Poly log(const Poly &a) {
200     Poly res = derivative(a) * inv(a);
201     res.resize(a.size() - 1);
202     return integral(res);
203 }
204
205 static Poly exp(const Poly &a, int k = -1) {
206     if (k == -1) k = a.size();
207     if (k == 1) return { 1 };
208     Poly res = exp(a, (k + 1) >> 1);
209     res.resize(k);
210     Poly temp = log(res);
211     for (auto &i : temp) i = i ? MOD - i : 0;
212     ++temp[0];
213     res = res * (temp + a);
214     res.resize(k);
215     return res;
216 }
217
218 static Poly sqrt(const Poly &a, int k = -1) {
219     if (k == -1) k = a.size();
220     if (k == 1) return { modSqrt(a[0]) };
221     Poly res = sqrt(a, (k + 1) >> 1), temp(a.begin(), a.begin() + k);
222     res.resize(k);
223     res = res + temp * inv(res);
224     res.resize(k);
225     for (auto &i : res) i = (i % 2 ? ((i + MOD) >> 1) : (i >> 1));
226     return res;
227 }
228
229 static Poly pow(const Poly &a, int n) {
230     Poly res = log(a);
231     for (auto &i : res) i = i * n % MOD;
232     return exp(res);
233 }
234
235
236 private:
237     static const int BRUTE_LIM = 20;
238 };
239
240 int main() {
241     init();
242
243     int n, k;
244     scanf("%d %d", &n, &k);
245
246     Poly a(n + 1);
247     for (int i = 0; i <= n; i++) scanf("%lld", &a[i]);
248
249     return 0;

```

250 }

5.18 多项式 GCD

```

1 #include <vector>
2
3 const int MOD = 1000000007;
4
5 long long qpow(long long a, long long n) {
6     long long res = 1;
7     for (; n; n >>= 1, a = a * a % MOD) if (n & 1) res = res * a % MOD;
8     return res;
9 }
10
11 // a[i] is the coefficient of x^i
12 std::vector<int> polyGcd(std::vector<int> a, const std::vector<int> &b) {
13     if (b.empty()) return a;
14     int t = a.size() - b.size();
15     for (int i = 0; i <= t; i++) {
16         long long temp = a[a.size() - 1 - i] * qpow(b[b.size() - 1], MOD - 2) % MOD;
17         for (int j = 0; j < b.size(); j++)
18             a[a.size() - 1 - i - j] = (a[a.size() - 1 - i - j] - temp * b[b.size() - 1 - j] % MOD + MOD) %
19             MOD;
20     }
21     int p = -1;
22     for (int i = a.size() - 1; ~i; i--) if (a[i]) {
23         p = i;
24         break;
25     }
26     a.resize(p + 1);
27     return polyGcd(b, a);
28 }
29
30 int main() {
31     return 0;
32 }
```

5.19 原根

```

1 #include <cstdio>
2
3 const int MAXN = 1000005;
4
5 int prime[MAXN], primeCnt;
6
7 void sieve() {
8     static bool notPrime[MAXN];
9     notPrime[0] = notPrime[1] = true;
10    primeCnt = 0;
11
12    for (int i = 2; i < MAXN; i++) {
```

```

13     if (!notPrime[i]) prime[primeCnt++] = i;
14
15     for (int j = 0; j < primeCnt && i * prime[j] < MAXN; j++)
16         notPrime[i * prime[j]] = true;
17     }
18 }
19
20 int pow(int a, int n, int p) {
21     int res = 1;
22     for (; n; n >= 1, a = a * a % p) if (n & 1) res = res * a % p;
23     return res;
24 }
25
26 int getRoot(int p) {
27     for (int g = 2, pp = p - 1; ; g++) {
28         bool flag = true;
29         for (int i = 0; i < primeCnt && prime[i] < p; i++) {
30             if (pp % prime[i] == 0 && pow(g, pp / prime[i], p) == 1) {
31                 flag = false;
32                 break;
33             }
34         }
35         if (flag) return g;
36     }
37 }
38
39
40 int main() {
41     return 0;
42 }
43 }
```

5.20 常系数线性齐次递推

5.20.1 暴力多项式取模

```

1 #include <cstdio>
2 #include <algorithm>
3
4 const int MOD = 1000000007;
5 const int MAXK = 2005;
6
7 long long a[MAXK];
8 void modMul(long long *A, long long *B, int n, long long *r) {
9     static long long res[MAXK << 1];
10    std::fill(res, res + (n << 1), 0);
11
12    for (int i = 0; i < n; i++) for (int j = 0; j < n; j++) {
13        res[i + j] += A[i] * B[j] % MOD;
14        res[i + j] >= MOD ? res[i + j] -= MOD : 0;
15    }
16
17    for (int i = (n << 1) - 2; i >= n; i--) if (res[i]) for (int j = n - 1; ~j; j--) {
18        res[i - n + j] += res[i] * a[j] % MOD;
```

```

19         res[i - n + j] >= MOD ? res[i - n + j] -= MOD : 0;
20     }
21
22     for (int i = 0; i < n; i++) r[i] = res[i];
23 }
24
25 int main() {
26     int n, k;
27     scanf("%d %d", &n, &k);
28
29     static long long h[MAXK];
30     for (int i = 0; i < k; i++) {
31         scanf("%lld", &a[i]);
32         a[i] < 0 ? a[i] += MOD : 0;
33     }
34     std::reverse(a, a + k);
35     a[k] = 1;
36
37     for (int i = 0; i < k; i++) {
38         scanf("%lld", &h[i]);
39         h[i] < 0 ? h[i] += MOD : 0;
40     }
41
42     if (n < k) return printf("%lld\n", h[n]), 0;
43
44     static long long m[MAXK], t[MAXK];
45     m[0] = t[1] = 1;
46
47     for (int i = n; i; i >>= 1, modMul(t, t, k, t)) if (i & 1) modMul(m, t, k, m);
48
49     long long hn = 0;
50     for (int i = 0; i < k; i++) hn = (hn + m[i] * h[i] % MOD) % MOD;
51
52     printf("%lld\n", hn);
53
54     return 0;
55 }
```

5.20.2 FFT 多项式取模

```

1 #include <cstdio>
2 #include <cmath>
3 #include <algorithm>
4
5 const int MAXN = 131072 + 1;
6 const int MOD = 1000000007;
7 const double PI = std::acos(-1);
8
9 long long qpow(long long a, long long n) {
10     long long res = 1;
11     for (; n; n >>= 1, a = a * a % MOD) if (n & 1) res = res * a % MOD;
12     return res;
13 }
14
15 long long inv(long long x) {
```

```

16     return qpow(x, MOD - 2);
17 }
18
19 struct Complex {
20     double r, i;
21
22     Complex(double r = 0, double i = 0) : r(r), i(i) {}
23
24     Complex conj() const { return Complex(r, -i); }
25
26     Complex operator-(const Complex &rhs) const { return Complex(r - rhs.r, i - rhs.i); }
27     Complex operator+(const Complex &rhs) const { return Complex(r + rhs.r, i + rhs.i); }
28     Complex operator*(const Complex &rhs) const { return Complex(r * rhs.r - i * rhs.i, r * rhs.i + i * rhs
29         .r); }
30     Complex operator/(double rhs) const { return Complex(r / rhs, i / rhs); }
31 };
32
33 class FFT {
34     private:
35         static const int N = 131072;
36
37         Complex omega[N + 1], omegaInv[N + 1];
38
39         void init() {
40             double per = 2 * PI / N;
41             for (int i = 0; i < N; i++) {
42                 omega[i] = Complex(std::cos(i * per), std::sin(i * per));
43                 omegaInv[i] = omega[i].conj();
44             }
45         }
46
47         void reverse(Complex *a, int n) {
48             for (int i = 0, j = 0; i < n; i++) {
49                 if (i < j) std::swap(a[i], a[j]);
50                 for (int l = n >> 1; (j ^= l) < l; l >>= 1) {}
51             }
52         }
53
54         void transform(Complex *a, int n, Complex *omega) {
55             reverse(a, n);
56
57             for (int l = 2; l <= n; l <= 1) {
58                 int hl = l >> 1;
59                 for (Complex *x = a; x != a + n; x += l) {
60                     for (int i = 0; i < hl; i++) {
61                         Complex t = omega[N / l * i] * x[i + hl];
62                         x[i + hl] = x[i] - t;
63                         x[i] = x[i] + t;
64                     }
65                 }
66             }
67         }
68
69     public:
70         FFT() { init(); }

```

```

71     int extend(int n) {
72         int res = 1;
73         while (res < n) res <<= 1;
74         return res;
75     }
76
77     void dft(Complex *a, int n) {
78         transform(a, n, omega);
79     }
80
81     void idft(Complex *a, int n) {
82         transform(a, n, omegaInv);
83         for (int i = 0; i < n; i++) a[i] = a[i] / n;
84     }
85 } fft;
86
87 void polyMul(int n, int m, long long *a, long long *b, long long *res) {
88     static Complex a0[MAXN], a1[MAXN], b0[MAXN], b1[MAXN];
89     static const int M = (1 << 15) - 1;
90     int N = fft.extend(n + m - 1);
91
92     for (int i = 0; i < n; i++) {
93         a0[i] = a[i] >> 15;
94         a1[i] = a[i] & M;
95     }
96     std::fill(a0 + n, a0 + N, 0);
97     std::fill(a1 + n, a1 + N, 0);
98     for (int i = 0; i < m; i++) {
99         b0[i] = b[i] >> 15;
100        b1[i] = b[i] & M;
101    }
102    std::fill(b0 + n, b0 + N, 0);
103    std::fill(b1 + n, b1 + N, 0);
104
105    fft.dft(a0, N), fft.dft(a1, N);
106    fft.dft(b0, N), fft.dft(b1, N);
107    for (int i = 0; i < N; i++) {
108        Complex _a = a0[i], _b = a1[i], _c = b0[i], _d = b1[i];
109        a0[i] = _a * _c;
110        a1[i] = _a * _d + _b * _c;
111        b0[i] = _b * _d;
112    }
113    fft.idft(a0, N), fft.idft(a1, N), fft.idft(b0, N);
114    for (int i = 0; i < N; i++) {
115        res[i] = (((long long) (a0[i].r + 0.5) % MOD) << 30) % MOD
116            + (((long long) (a1[i].r + 0.5) % MOD) << 15) % MOD
117            + (long long) (b0[i].r + 0.5) % MOD) % MOD;
118    }
119 }
120
121 void polyInv(int k, long long *a, long long *res) {
122     if (k == 1) {
123         res[0] = inv(a[0]);
124         return;
125     }
126     polyInv((k + 1) >> 1, a, res);

```

```

127
128     static long long t1[MAXN], t2[MAXN];
129     int N = fft.extend(k << 1);
130     std::copy(a, a + k, t1);
131     std::fill(t1 + k, t1 + N, 0);
132     polyMul(N, N, res, res, t2);
133     polyMul(N, N, t1, t2, t1);
134     for (int i = 0; i < k; i++) res[i] = (2 * res[i] % MOD - t1[i] + MOD) % MOD;
135     std::fill(res + k, res + N, 0);
136 }
137
138 void polyDiv(int n, int m, long long *A, long long *B, long long *D, long long *R) {
139     static long long A0[MAXN], B0[MAXN];
140
141     int t = n - m + 1;
142     int N = fft.extend(t << 1);
143
144     std::fill(A0, A0 + N, 0);
145     std::reverse_copy(B, B + m, A0);
146     polyInv(t, A0, B0);
147
148     std::reverse_copy(A, A + n, A0);
149
150     polyMul(t, t, A0, B0, A0);
151
152     std::reverse(A0, A0 + t);
153     std::copy(A0, A0 + t, D);
154
155     N = fft.extend(n);
156     std::copy(B, B + m, B0);
157     polyMul(t, m, A0, B0, A0);
158     for (int i = 0; i < m; i++) R[i] = (A[i] - A0[i] + MOD) % MOD;
159     std::fill(R + m, R + N, 0);
160 }
161
162 void polyPow(int n, long long *a, long long *mod, int k, long long *res) {
163     res[0] = 1;
164     static long long D[MAXN], R[MAXN];
165     for (; k; k >= 1) {
166         if (k & 1) {
167             polyMul(n, n, res, a, res);
168             polyDiv(2 * n - 1, n, res, mod, D, R);
169             std::copy(R, R + n, res);
170         }
171         polyMul(n, n, a, a, a);
172         polyDiv(2 * n - 1, n, a, mod, D, R);
173         std::copy(R, R + n, a);
174     }
175 }
176
177 long long a[MAXN], x[MAXN], f[MAXN], t[MAXN], m[MAXN];
178
179 int main() {
180     int n, k;
181     scanf("%d %d", &n, &k);
182     for (int i = 1; i <= k; i++) {

```

```

183     scanf("%lld", &a[i]); // coeff
184     a[i] < 0 && (a[i] += MOD);
185 }
186 for (int i = 0; i < k; i++) {
187     scanf("%lld", &x[i]); // value
188     x[i] < 0 && (x[i] += MOD);
189 }
190
191 f[k] = 1;
192 for (int i = 0; i < k; i++) f[i] = a[k - i] ? MOD - a[k - i] : 0;
193
194 t[1] = 1;
195 polyPow(k + 1, t, f, n, m);
196
197 long long ans = 0;
198 for (int i = 0; i < k; i++) ans = (ans + m[i] * x[i]) % MOD;
199 printf("%lld\n", ans);
200
201 return 0;
202 }
```

5.21 拉格朗日插值

给出 n 次多项式在 $0 \sim n$ 的值和任意定义域范围内的 x , 在 $O(n)$ 的时间内求出 $f(x)$ 。

```

1 #include <cstdio>
2
3 const int MAXN = 1000005;
4 const int MOD = 1000000007;
5
6 long long qpow(long long a, long long n) {
7     long long res = 1;
8     for (; n; n >>= 1, a = a * a % MOD) if (n & 1) res = res * a % MOD;
9     return res;
10 }
11
12 long long fact[MAXN], invFact[MAXN];
13 void init() {
14     fact[0] = 1;
15     for (int i = 1; i < MAXN; i++) fact[i] = fact[i - 1] * i % MOD;
16     invFact[MAXN - 1] = qpow(fact[MAXN - 1], MOD - 2);
17     for (int i = MAXN - 2; ~i; i--) invFact[i] = invFact[i + 1] * (i + 1) % MOD;
18 }
19
20 long long Lagrange(long long *a, int n, long long x) {
21     if (x <= n) return a[x];
22
23     static long long f1[MAXN], f2[MAXN];
24     f1[0] = f2[n + 1] = 1;
25     for (int i = 1; i <= n + 1; i++) f1[i] = f1[i - 1] * (x - i + 1) % MOD;
26     for (int i = n; ~i; i--) f2[i] = f2[i + 1] * (x - i) % MOD;
27
28     long long res = 0;
29     for (int i = 0; i <= n; i++) {
30         long long temp = a[i] * f1[i] % MOD * f2[i + 1] % MOD * invFact[i] % MOD * invFact[n - i] % MOD;
```

```
31     (n - i) % 2 ? res -= temp : res += temp;
32     res >= MOD ? res -= MOD : 0;
33     res < 0 ? res += MOD : 0;
34 }
35
36     return res;
37 }
38
39 int main() {
40     init();
41
42     int n;
43     scanf("%d", &n);
44
45     static long long a[MAXN + 1];
46     for (int i = 0; i <= n; i++) scanf("%lld", &a[i]);
47
48     long long x;
49     scanf("%lld", &x);
50     long long ans = Lagrange(a, n, x);
51     printf("%lld\n", ans);
52
53     return 0;
54 }
```

6 计算几何

6.1 点相关

```
1 #include <cstdio>
2 #include <cmath>
3 #include <algorithm>
4
5 const int MAXN = 100005;
6 const double EPS = 1e-9;
7
8 int dcmp(double a, double b = 0) {
9     double d = a - b;
10    return std::abs(d) <= EPS ? 0 : (d > 0 ? 1 : -1);
11 }
12
13 struct Point {
14     double x, y;
15
16     Point(double x = 0, double y = 0) : x(x), y(y) {}
17
18     bool operator<(const Point &rhs) const { return x == rhs.x ? y < rhs.y : x < rhs.x; }
19
20     Point operator+(const Point &rhs) const { return Point(x + rhs.x, y + rhs.y); }
21     Point operator-(const Point &rhs) const { return Point(x - rhs.x, y - rhs.y); }
22     Point operator*(const double a) const { return Point(x * a, y * a); }
23     Point operator/(const double a) const { return Point(x / a, y / a); }
24     friend double dot(const Point &a, const Point &b) { return a.x * b.x + a.y * b.y; }
25     friend double cross(const Point &a, const Point &b) { return a.x * b.y - a.y * b.x; }
26
27     friend double angle(const Point &a, const Point &b) {
28         return std::acos(dot(a, b) / a.length() / b.length());
29     }
30
31     Point rotate(double rad) {
32         return Point(x * std::cos(rad) - y * std::sin(rad),
33                     x * std::sin(rad) + y * std::cos(rad));
34     }
35
36     double length() const { return std::sqrt(dot(*this, *this)); }
37
38     Point getNormal() const {
39         Point res(-y, x);
40         return res / length();
41     }
42 } P[MAXN], ch[MAXN];
43
44 bool parallel(const Point &as, const Point &at, const Point &bs, const Point &bt) {
45     return dcmp(cross(at - as, bt - bs)) == 0;
46 }
47
48 bool getLineInter(const Point &as, const Point &at, const Point &bs, const Point &bt, Point &res) {
49     if (parallel(as, at, bs, bt)) return false;
```

```

50     double c1 = cross(as - bs, bt - bs);
51     double c2 = cross(at - bs, bt - bs);
52     res = (at * c1 - as * c2) / (c1 - c2);
53     return true;
54 }
55
56 bool getSegInter(const Point &as, const Point &at, const Point &bs, const Point &bt, Point &p) {
57     if (!dcmp(cross(at - as, bt - bs))) return false;
58     double c1 = cross(bs - as, at - as), c2 = cross(bt - as, at - as);
59     double c3 = cross(as - bs, bt - bs), c4 = cross(at - bs, bt - bs);
60     if (dcmp(c1) * dcmp(c2) <= 0 && dcmp(c3) * dcmp(c4) <= 0) {
61         p = (at * c3 - as * c4) / (c3 - c4);
62         return true;
63     } else return false;
64 }
65
66 Point getLineProj(const Point &p, const Point &s, const Point &t) {
67     Point u = t - s;
68     return s + u * (dot(u, p - s) / dot(u, u));
69 }
70
71 double getDistToLine(const Point &p, const Point &s, const Point &t) {
72     return std::abs(cross(t - s, p - s)) / (t - s).length();
73 }
74
75 double getDistToSeg(const Point &p, const Point &s, const Point &t) {
76     if (dcmp(dot(t - s, p - s)) < 0) return (p - s).length();
77     else if (dcmp(dot(t - s, p - t)) > 0) return (p - t).length();
78     else return std::abs(cross(t - s, p - s)) / (t - s).length();
79 }
80
81 bool doesPointOnSeg(const Point &p, const Point &s, const Point &t) {
82     return dcmp(cross(s - p, t - p)) == 0 && dcmp(dot(s - p, t - p)) < 0;
83 }
84
85 int getConvexHull(int n) {
86     std::sort(P, P + n);
87
88     int m = 0;
89     for (int i = 0; i < n; i++) {
90         while (m > 1 && cross(ch[m - 1] - ch[m - 2], P[i] - ch[m - 2]) <= 0) m--;
91         ch[m++] = P[i];
92     }
93
94     int k = m;
95     for (int i = n - 1; ~i; i--) {
96         while (m > k && cross(ch[m - 1] - ch[m - 2], P[i] - ch[m - 2]) <= 0) m--;
97         ch[m++] = P[i];
98     }
99
100    m > 1 ? m-- : 0;
101    return m;
102 }
103
104 int getv(const Point &p){
105     if (p.x >= 0 && p.y > 0) return 1;

```

```

106     if (p.x < 0 && p.y >= 0) return 2;
107     if (p.x <= 0 && p.y < 0) return 3;
108     if (p.x > 0 && p.y <= 0) return 4;
109 }
110
111 bool cmpByAngle(const Point &a, const Point &b) {
112     if (getv(a) == getv(b)) return cross(a, b) > 0;
113     else return getv(a) < getv(b);
114 }
115
116 // *r == *l should be satisfied for the following 3 functions
117 double polyArea(Point *l, Point *r) {
118     double sum = 0;
119     for (Point *p = l; p < r; p++)
120         sum += cross(*(p + 1), *p);
121     return std::abs(sum / 2.0);
122 }
123
124 bool insidePoly(Point *l, Point *r, const Point &p) {
125     int num = 0;
126     for (Point *q = l; q < r; q++) {
127         if (doesPointOnSeg(p, *q, *(q + 1))) return true;
128         int k = dcmp(cross(*(q + 1) - *q, p - *q));
129         int d1 = dcmp(q->y - p.y);
130         int d2 = dcmp((q + 1)->y - p.y);
131         if (k > 0 && d1 <= 0 && d2 > 0) ++num;
132         if (k < 0 && d2 <= 0 && d1 > 0) --num;
133     }
134     return num;
135 }
136
137 Point polyWeight(Point *l, Point *r, double area) {
138     Point res(0, 0);
139     for (Point *p = l; p < r; p++)
140         res = res + (*p + *(p + 1)) * cross(*(p + 1), *p);
141     return res / area / 6.0;
142 }
143
144 int main() {
145
146     return 0;
147 }
```

6.2 旋转卡壳

旋转卡壳求最小矩形覆盖：

```

1 #include <cstdio>
2 #include <cfloat>
3 #include <cmath>
4 #include <algorithm>
5
6 const int MAXN = 50005;
7 const double EPS = 1e-9;
8
```

```

9  struct Point {
10    double x, y;
11
12    Point(double x = 0, double y = 0) : x(x), y(y) {}
13
14    bool operator<(const Point &rhs) const { return x == rhs.x ? y < rhs.y : x < rhs.x; }
15
16    Point operator+(const Point &rhs) const { return Point(x + rhs.x, y + rhs.y); }
17    Point operator-(const Point &rhs) const { return Point(x - rhs.x, y - rhs.y); }
18    Point operator*(const double a) const { return Point(x * a, y * a); }
19    Point operator/(const double a) const { return Point(x / a, y / a); }
20    friend double dot(const Point &a, const Point &b) { return a.x * b.x + a.y * b.y; }
21    friend double cross(const Point &a, const Point &b) { return a.x * b.y - a.y * b.x; }
22
23    friend double dist(const Point &a, const Point &b) {
24        return sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y));
25    }
26
27    void print() {
28        if (std::abs(x) <= EPS) x = 0;
29        if (std::abs(y) <= EPS) y = 0;
30        printf("%.5f %.5f\n", x, y);
31    }
32
33    Point getPerpendicular() {
34        double X = sqrt(1 / (1 + (x / y) * (x / y)));
35
36        int sx, sy;
37        if (x > 0 && y > 0) sx = 1, sy = -1;
38        else if (x > 0 && y <= 0) sx = 1, sy = 1;
39        else if (x <= 0 && y > 0) sx = -1, sy = -1;
40        else sx = 1, sy = -1;
41
42        return Point(sx * X, sy * std::sqrt(1 - X * X));
43    }
44 } P[MAXN], ch[MAXN];
45
46 struct Rectangle {
47     Point p[4];
48
49     void print() {
50         int temp = 0;
51         for (int i = 1; i < 4; i++)
52             if (p[i].y < p[temp].y || (p[i].y == p[temp].y && p[i].x < p[temp].x)) temp = i;
53         for (int i = 0; i < 4; i++) p[(i + temp) % 4].print();
54     }
55
56     Point &operator[](int i) { return p[i]; }
57 };
58
59 int getConvexHull(int n) {
60     std::sort(P + 1, P + n + 1);
61
62     int m = 0;
63     for (int i = 1; i <= n; i++) {
64         while (m > 1 && cross(ch[m - 1] - ch[m - 2], P[i] - ch[m - 2]) <= 0) m--;

```

6 计算几何

```
65         ch[m++] = P[i];
66     }
67
68     int k = m;
69     for (int i = n; i; i--) {
70         while (m > k && cross(ch[m - 1] - ch[m - 2], P[i] - ch[m - 2]) <= 0) m--;
71         ch[m++] = P[i];
72     }
73
74     m > 1 ? m-- : 0;
75     return m;
76 }
77
78 Rectangle ans;
79 double ansArea = DBL_MAX;
80 void rotatingCalipers(int n) {
81     for (int curr = 0, right = 1, up = 1, left = 1; curr < n; curr++) {
82         while (dot(ch[curr + 1] - ch[curr], ch[right + 1] - ch[right]) >= 0)
83             right = (right + 1) % n;
84
85         curr ? 0 : up = right;
86         while (cross(ch[curr + 1] - ch[curr], ch[up + 1] - ch[up]) >= 0)
87             up = (up + 1) % n;
88
89         curr ? 0 : left = up;
90         while (cross(ch[curr + 1] - ch[curr], ch[left + 1] - ch[left]) <= 0)
91             left = (left + 1) % n;
92
93         Point currV = ch[curr + 1] - ch[curr];
94         double currLen = dist(ch[curr], ch[curr + 1]);
95         double height = std::abs(cross(currV, ch[up] - ch[curr]) / currLen);
96         double bottom = std::abs(dot(currV, ch[left] - ch[curr]) / currLen)
97             + std::abs(dot(currV, ch[right] - ch[curr]) / currLen);
98
99         double currArea = bottom * height;
100        Point currPerpendicular = currV.getPerpendicular();
101
102        if (currArea < ansArea) {
103            ansArea = currArea;
104            ans[0] = ch[curr] + currV * fabs((dot(currV, ch[right] - ch[curr])) / currLen) / currLen;
105            ans[1] = ans[0] + currPerpendicular * height;
106            ans[2] = ans[1] - currV * bottom / currLen;
107            ans[3] = ans[2] - currPerpendicular * height;
108        }
109    }
110 }
111
112 int main() {
113     int n;
114     scanf("%d", &n);
115
116     for (int i = 1; i <= n; i++) scanf("%lf %lf", &P[i].x, &P[i].y);
117
118     int m = getConvexHull(n);
119     rotatingCalipers(m);
120 }
```

```

121     printf("%.5f\n", ansArea);
122     ans.print();
123
124     return 0;
125 }
```

6.3 半平面交

```

1 #include <cstdio>
2 #include <cfloat>
3 #include <cmath>
4 #include <algorithm>
5
6 const int MAXN = 305;
7 const double EPS = 1e-14;
8
9 int dcmp(double a, double b = 0) {
10     double d = a - b;
11     return std::abs(d) <= EPS ? 0 : (d > 0 ? 1 : -1);
12 }
13
14 struct Point {
15     double x, y;
16
17     Point(double x = 0, double y = 0) : x(x), y(y) {}
18
19     Point operator+(const Point &rhs) const { return Point(x + rhs.x, y + rhs.y); }
20     Point operator-(const Point &rhs) const { return Point(x - rhs.x, y - rhs.y); }
21     Point operator*(double rhs) const { return Point(x * rhs, y * rhs); }
22     friend double cross(const Point &a, const Point &b) { return a.x * b.y - a.y * b.x; }
23 } P[MAXN], hpi[MAXN];
24
25 struct Line {
26     Point p, v;
27     double slop;
28
29     Line() {}
30     Line(const Point &p, const Point &v) : p(p), v(v) {
31         slop = std::atan2(v.y, v.x);
32     }
33
34     Point getVal(double t) const { return p + v * t; }
35
36     bool operator<(const Line &another) const {
37         return slop < another.slop || (slop == another.slop && v.x
38             && getVal(-p.x / v.x).y > getVal(-another.p.x / another.v.x).y);
39     }
40
41     friend Point getIntersection(const Line &a, const Line &b) {
42         double t = cross(b.v, a.p - b.p) / cross(a.v, b.v);
43         return a.getVal(t);
44     }
45 } L[MAXN];
46
```

```

47 int n;
48 int halfplaneIntersection() {
49     int cnt = 0;
50     L[cnt++] = L[0];
51     for (int i = 1; i <= n; i++) if (dcmp(L[i].slop, L[i - 1].slop)) L[cnt++] = L[i];
52     std::sort(L, L + cnt);
53
54     static Line q[MAXN];
55     static Point p[MAXN];
56     int l = 0, r = 0;
57     q[l] = L[0];
58     for (int i = 1; i < cnt; i++) {
59         while (l < r && dcmp(cross(L[i].v, p[r - 1] - L[i].p)) < 0) r--;
60         while (l < r && dcmp(cross(L[i].v, p[l] - L[i].p)) < 0) l++;
61         q[++r] = L[i];
62         if (l < r) p[r - 1] = getIntersection(q[r - 1], q[r]);
63     }
64     while (l < r && dcmp(cross(q[l].v, p[r - 1] - q[l].p)) < 0) r--;
65     while (l < r && dcmp(cross(q[r].v, p[l] - q[r].p)) < 0) l++;
66
67     if (r - l <= 1) return 0;
68
69     cnt = 0;
70     for (int i = l; i < r; i++) hpi[++cnt] = p[i];
71     return cnt;
72 }
73
74 int main() {
75     scanf("%d", &n);
76     for (int i = 1; i <= n; i++) scanf("%lf", &P[i].x);
77     for (int i = 1; i <= n; i++) scanf("%lf", &P[i].y);
78
79     P[0] = Point(P[1].x, P[1].y + 1);
80     P[n + 1] = Point(P[n].x, P[n].y + 1);
81     for (int i = 0; i <= n; i++) L[i] = Line(P[i], P[i + 1] - P[i]);
82
83     int m = halfplaneIntersection();
84
85     return 0;
86 }
```

6.4 k 次圆覆盖

求被 n 个圆中（至少） $1 \sim n$ 个圆覆盖的面积

```

1 #include <cstdio>
2 #include <cmath>
3 #include <algorithm>
4
5 const int MAXN = 1005;
6 const double EPS = 1e-9;
7 const double PI = std::acos(-1.0);
8
9 int dcmp(double a, double b = 0.0) {
10     double d = a - b;
```

```

11     return std::abs(d) <= EPS ? 0 : (d > 0 ? 1 : -1);
12 }
13
14 inline double sqr(double x) { return x * x; }
15 inline double safeSqrt(double x) { return !dcmp(x) ? 0 : std::sqrt(x); }
16
17 struct Point {
18     double x, y;
19
20     Point(double x = 0, double y = 0) : x(x), y(y) {}
21
22     Point operator+(const Point &rhs) const { return Point(x + rhs.x, y + rhs.y); }
23     Point operator-(const Point &rhs) const { return Point(x - rhs.x, y - rhs.y); }
24     friend double cross(const Point &a, const Point &b) { return a.x * b.y - a.y * b.x; }
25     double length() const { return std::sqrt(x * x + y * y); }
26 };
27
28 double dist(const Point &a, const Point &b) {
29     return std::sqrt(sqr(a.x - b.x) + sqr(a.y - b.y));
30 }
31
32 struct Circle {
33     Point p;
34     double r, angle;
35     int d;
36
37     Circle() {}
38     Circle(const Point &p, double r) : p(p), r(r), d(1), angle(0) {}
39     Circle(double x, double y, double angle = 0, int d = 0) : p(Point(x, y)), r(r), d(d), angle(angle) {}
40 } C[MAXN];
41
42 namespace CircleUnion {
43
44     Circle tp[MAXN << 1];
45     double area[MAXN];
46
47     double calc(const Circle &c, const Circle &cp1, const Circle &cp2) {
48         double ans = (cp2.angle - cp1.angle) * sqr(c.r) - cross(cp1.p - c.p, cp2.p - c.p) + cross(cp1.p, cp2.p)
49             ;
50         return ans / 2.0;
51     }
52
53     int numberOfCircleCross(const Circle a, const Circle b, Circle &cp1, Circle &cp2) {
54         double mx = b.p.x - a.p.x, sx = b.p.x + a.p.x, mx2 = mx * mx;
55         double my = b.p.y - a.p.y, sy = b.p.y + a.p.y, my2 = my * my;
56         double sq = mx2 + my2, d = -(sq - sqr(a.r - b.r)) * (sq - sqr(a.r + b.r));
57
58         if (dcmp(d) < 0) return 0;
59         d = safeSqrt(d);
60
61         double x = mx * ((a.r + b.r) * (a.r - b.r) + mx * sx) + sx * my2;
62         double y = my * ((a.r + b.r) * (a.r - b.r) + my * sy) + sy * mx2;
63         double dx = mx * d, dy = my * d;
64         sq *= 2;
65
66         cp1.p.x = (x - dy) / sq;

```

```

66     cp1.p.y = (y + dx) / sq;
67     cp2.p.x = (x + dy) / sq;
68     cp2.p.y = (y - dx) / sq;
69     return dcmp(d) > 0 ? 2 : 1;
70 }
71
72 void circleUnion(Circle *C, int n) {
73     std::sort(C, C + n, [](const Circle &a, const Circle &b) {return a.r < b.r;});
74
75     for (int i = 0; i < n; i++) for (int j = i + 1; j < n; j++)
76         if (dcmp(dist(C[i].p, C[j].p) + C[i].r - C[j].r) <= 0) ++C[i].d;
77
78     for (int i = 0; i < n; i++) {
79         int tn = 0, cnt = 0;
80         for (int j = 0; j < n; j++) if (i != j) {
81             static Circle cp1, cp2;
82             // pay attention to the order of parameter!!!
83             if (numberOfCircleCross(C[i], C[j], cp2, cp1) < 2) continue;
84             cp1.angle = std::atan2(cp1.p.y - C[i].p.y, cp1.p.x - C[i].p.x);
85             cp2.angle = std::atan2(cp2.p.y - C[i].p.y, cp2.p.x - C[i].p.x);
86             cp1.d = 1;
87             cp2.d = -1;
88             tp[tn++] = cp1;;
89             tp[tn++] = cp2;
90             if (dcmp(cp1.angle, cp2.angle) > 0) ++cnt;
91         }
92         tp[tn++] = Circle(C[i].p.x - C[i].r, C[i].p.y, PI, -cnt);
93         tp[tn++] = Circle(C[i].p.x - C[i].r, C[i].p.y, -PI, cnt);
94         std::sort(tp, tp + tn, [](const Circle &a, const Circle &b) {
95             return !dcmp(a.angle, b.angle) ? a.d > b.d : a.angle < b.angle;
96         });
97         int s = C[i].d + tp[0].d;
98         for (int j = 1; j < tn; j++) {
99             int p = s;
100            s += tp[j].d;
101            area[p] += calc(C[i], tp[j - 1], tp[j]);
102        }
103    }
104 }
105
106 void solve(Circle *C, int n) {
107     std::fill(area + 1, area + n + 1, 0);
108     circleUnion(C, n);
109
110     // delete this line if you want to know area covered by at least i-times instead of exactly i-times
111     for (int i = 1; i < n; i++) area[i] -= area[i + 1];
112 }
113
114 } // namespace CircleUnion
115
116 int main() {
117     int n;
118     scanf("%"d", &n);
119     for (int i = 0; i < n; i++) {
120         scanf("%"lf "%lf %lf", &C[i].p.x, &C[i].p.y, &C[i].r);
121         C[i].d = 1;

```

```

122     }
123
124     CircleUnion::solve(C, n);
125
126     return 0;
127 }
```

6.5 平面图区域 (Farmland)

```

1 #include <cstdio>
2 #include <cmath>
3 #include <vector>
4 #include <algorithm>
5
6 const int MAXN = 200005;
7 const double EPS = 1e-9;
8 const double PI = std::acos(-1.0);
9
10 int dcmp(double a, double b = 0) {
11     double d = a - b;
12     return std::abs(d) <= EPS ? 0 : (d > 0 ? 1 : -1);
13 }
14
15 struct Point {
16     int x, y;
17
18     friend long long cross(const Point &a, const Point &b) {
19         return 1ll * a.x * b.y - 1ll * a.y * b.x;
20     }
21 } P[MAXN];
22
23 struct Edge {
24     int u, v;
25     double a;
26     bool vis;
27
28     bool operator<(const Edge &rhs) const { return a < rhs.a; }
29
30     Edge(int u, int v, double a) : u(u), v(v), a(a), vis(false) {}
31 };
32 std::vector<Edge> N[MAXN];
33
34 void addEdge(int u, int v, double a1, double a2) {
35     N[u].emplace_back(u, v, a1);
36     N[v].emplace_back(v, u, a2);
37 }
38
39 int find(const std::vector<Edge> &e, double a) {
40     double d = a + PI - EPS;
41     if (dcmp(d, PI) > 0) d -= 2 * PI;
42     int res = std::upper_bound(e.begin(), e.end(), d, [](double a, const Edge &b) {
43         return a < b.a;
44     }) - e.begin() - 1;
45     if (res < 0) res += e.size();
```

```

46     return res;
47 }
48
49 int main() {
50     int n, m;
51     scanf("%d %d", &n, &m);
52     for (int i = 1; i <= n; i++) scanf("%d %d", &P[i].x, &P[i].y);
53     for (int i = 0, u, v; i < m; i++) {
54         scanf("%d %d", &u, &v);
55         double a1 = std::atan2(P[v].y - P[u].y, P[v].x - P[u].x);
56         double a2 = std::atan2(P[u].y - P[v].y, P[u].x - P[v].x);
57         addEdge(u, v, a1, a2);
58     }
59     for (int i = 1; i <= n; i++) std::sort(N[i].begin(), N[i].end());
60
61     static std::vector<long long> ans;
62     for (int i = 1; i <= n; i++) {
63         for (Edge &e : N[i]) if (!e.vis) {
64             int u = i, v = e.v;
65             long long S = cross(P[u], P[v]);
66             double lasta = e.a;
67             e.vis = true;
68             do {
69                 int t = find(N[v], lasta);
70                 if (N[v][t].vis) {
71                     S = 0;
72                     break;
73                 }
74                 u = v;
75                 v = N[u][t].v;
76                 lasta = N[u][t].a;
77                 N[u][t].vis = true;
78                 S += cross(P[u], P[v]);
79             } while (v != i);
80             if (S > 0) ans.push_back(S);
81         }
82     }
83
84     std::sort(ans.begin(), ans.end());
85     printf("%zu\n", ans.size());
86     for (auto i : ans) printf("%lld\n", i);
87
88     return 0;
89 }
```

6.6 Deluanay 三角剖分与平面欧几里得距离最小生成树

```

1 #include <cstdio>
2 #include <cmath>
3 #include <set>
4 #include <list>
5 #include <vector>
6 #include <algorithm>
7
```

```

8  const int MAXN = 100005;
9  const double EPS = 1e-9;
10
11 int dcmp(double a, double b = 0) {
12     double d = a - b;
13     return std::abs(d) <= EPS ? 0 : (d > 0 ? 1 : -1);
14 }
15
16 double sqr(double x) { return x * x; }
17
18 struct Point {
19     double x, y;
20     int id;
21
22     Point(double x = 0, double y = 0, int id = -1) : x(x), y(y), id(id) {}
23
24     bool operator<(const Point &rhs) const {
25         return dcmp(x, rhs.x) == 0 ? y < rhs.y : x < rhs.x;
26     }
27
28     Point operator+(const Point &rhs) const { return Point(x + rhs.x, y + rhs.y); }
29     Point operator-(const Point &rhs) const { return Point(x - rhs.x, y - rhs.y); }
30     friend double dot(const Point &a, const Point &b) { return a.x * b.x + a.y * b.y; }
31     friend double cross(const Point &a, const Point &b) { return a.x * b.y - a.y * b.x; }
32
33     friend double dist(const Point &a, const Point &b) {
34         return std::sqrt(sqr(a.x - b.x) + sqr(a.y - b.y));
35     }
36     friend double distSqr(const Point &a, const Point &b) {
37         return sqr(a.x - b.x) + sqr(a.y - b.y);
38     }
39 } P[MAXN];
40
41 namespace Delaunay {
42
43 struct Point3D {
44     double x, y, z;
45
46     Point3D(double x = 0, double y = 0, double z = 0) : x(x), y(y), z(z) {}
47     Point3D(const Point &p) : x(p.x), y(p.y), z(sqr(p.x) + sqr(p.y)) {}
48
49     friend Point3D operator-(const Point3D &a, const Point3D &b) {
50         return Point3D(a.x - b.x, a.y - b.y, a.z - b.z);
51     }
52
53     friend double dot(const Point3D &a, const Point3D &b) {
54         return a.x * b.x + a.y * b.y + a.z * b.z;
55     }
56     friend Point3D cross(const Point3D &a, const Point3D &b) {
57         return Point3D(a.y * b.z - a.z * b.y, a.z * b.x - a.x * b.z, a.x * b.y - a.y * b.x);
58     }
59     friend double mix(const Point3D &a, const Point3D &b, const Point3D &c) {
60         return dot(a, cross(b, c));
61     }
62 }
63 };

```

```

64 struct Edge {
65     int id;
66     std::list<Edge>::iterator c;
67
68     Edge(int id = 0) : id(id) {}
69 };
70
71 bool doesPointInCircle(const Point &a, Point b, Point c, const Point &p) {
72     if (cross(b - a, c - a) < 0) std::swap(b, c);
73     Point3D a3(a), b3(b), c3(c), p3(p);
74     b3 = b3 - a3, c3 = c3 - a3, p3 = p3 - a3;
75     return dcmp(mix(p3, b3, c3)) < 0;
76 }
77
78 bool hasIntersection(const Point &a, const Point &b, const Point &c, const Point &d) {
79     return dcmp(cross(c - a, b - a)) * dcmp(cross(b - a, d - a)) > 0 &&
80         dcmp(cross(a - c, d - c)) * dcmp(cross(d - c, b - c)) > 0;
81 }
82
83 std::list<Edge> head[MAXN];
84 Point p[MAXN];
85 int n, rename[MAXN];
86
87 void addEdge(int u, int v) {
88     head[u].push_front(Edge(v));
89     head[v].push_front(Edge(u));
90     head[u].begin()->c = head[v].begin();
91     head[v].begin()->c = head[u].begin();
92 }
93
94 void divide(int l, int r) {
95     if (r - l <= 2) {
96         for (int i = l; i <= r; i++) for (int j = i + 1; j <= r; j++) addEdge(i, j);
97         return;
98     }
99
100    int mid = l + ((r - l) >> 1);
101    divide(l, mid);
102    divide(mid + 1, r);
103
104    int nowl = l, nowr = r;
105    for (bool updated = true; updated; ) {
106        updated = false;
107        Point ptL = p[nowl], ptR = p[nowr];
108        for (auto i : head[nowl]) {
109            Point t = p[i.id];
110            double v = cross(ptL - ptR, t - ptR);
111            if (dcmp(v) > 0 || (!dcmp(v) && distSqr(ptR, t) < distSqr(ptR, ptL))) {
112                nowl = i.id;
113                updated = true;
114                break;
115            }
116        }
117        if (updated) continue;
118        for (auto i : head[nowr]) {
119            Point t = p[i.id];

```

```

120     double v = cross(ptR - ptL, t - ptL);
121     if (dcmp(v) < 0 || (!dcmp(v) && distSqr(ptL, t) < distSqr(ptL, ptR))) {
122         nowr = i.id;
123         updated = true;
124         break;
125     }
126 }
127 }
128
129 addEdge(nowl, nowr);
130 while (true) {
131     Point ptL = p[nowl], ptR = p[nowr];
132     int ch = -1, side = 0;
133     for (auto i : head[nowl]) {
134         Point t = p[i.id];
135         if (dcmp(cross(ptR - ptL, t - ptL)) > 0
136             && (ch == -1 || doesPointInCircle(ptL, ptR, p[ch], t))) {
137             ch = i.id;
138             side = -1;
139         }
140     }
141     for (auto i : head[nowr]) {
142         Point t = p[i.id];
143         if (dcmp(cross(t - ptR, ptL - ptR)) > 0
144             && (ch == -1 || doesPointInCircle(ptL, ptR, p[ch], t))) {
145             ch = i.id;
146             side = 1;
147         }
148     }
149
150     if (ch == -1) break;
151     if (side == -1) {
152         for (auto it = head[nowl].begin(); it != head[nowl].end(); ) {
153             if (hasIntersection(ptL, p[it->id], ptR, p[ch])) {
154                 head[it->id].erase(it->c);
155                 head[nowl].erase(it++);
156             } else {
157                 it++;
158             }
159         }
160         nowl = ch;
161         addEdge(nowl, nowr);
162     } else {
163         for (auto it = head[nowr].begin(); it != head[nowr].end(); ) {
164             if (hasIntersection(ptR, p[it->id], ptL, p[ch])) {
165                 head[it->id].erase(it->c);
166                 head[nowr].erase(it++);
167             } else {
168                 it++;
169             }
170         }
171         nowr = ch;
172         addEdge(nowl, nowr);
173     }
174 }
175 }
```

```

176
177 bool isParallel(const Point &a, const Point &b, const Point &c) {
178     return !dcmp(cross(b - a, c - a));
179 }
180
181 void getEdge(std::vector<std::pair<int, int> &ret) {
182     ret.reserve(n);
183     static std::set<std::pair<int, int>> vis;
184     vis.clear();
185
186     for (int i = 0; i < n; i++) for (auto j : head[i]) {
187         if (j.id < i) continue;
188         Point now = p[j.id];
189         for (auto k : head[i]) {
190             if (k.id < i) continue;
191             if (isParallel(p[i], p[j.id], p[k.id])
192                 && distSqr(p[k.id], p[i]) < distSqr(now, p[i]))
193                 now = p[k.id];
194         }
195         if (vis.find(std::make_pair(p[i].id, now.id)) == vis.end()) {
196             vis.insert(std::make_pair(p[i].id, now.id));
197             ret.push_back(std::make_pair(p[i].id, now.id));
198         }
199     }
200 }
201
202 void init(int n, Point *P) {
203     std::copy(P, P + n, p);
204     std::sort(p, p + n);
205     for (int i = 0; i < n; i++) rename[p[i].id] = i;
206     Delaunay::n = n;
207     divide(0, n - 1);
208 }
209
210 void split(int n, Point *P, std::vector<std::pair<int, int> &ret) {
211     for (int i = 0; i < n; i++) P[i].id = i;
212     init(n, P);
213     getEdge(ret);
214 }
215
216 } // namespace Delaunay
217
218 namespace Kruskal {
219
220     struct DSU {
221         int f[MAXN];
222
223         void init(int n) { for (int i = 1; i <= n; i++) f[i] = i; }
224         int find(int x) { return x == f[x] ? x : f[x] = find(f[x]); }
225         void merge(int x, int y) { f[find(y)] = find(x); }
226         bool test(int x, int y) { return find(x) == find(y); }
227     } dsu;
228
229     struct Edge {
230         int u, v;
231         double w;

```

```

232
233     Edge(int u, int v, double w) : u(u), v(v), w(w) {}
234 };
235 std::vector<Edge> edges;
236
237 double solve(int n) {
238     std::sort(edges.begin(), edges.end(), [](const Edge &a, const Edge &b){
239         return a.w < b.w;
240     });
241     dsu.init(n);
242
243     double ans = 0;
244     int added = 0;
245     for (auto e : edges) if (!dsu.test(e.u, e.v)) {
246         dsu.merge(e.u, e.v);
247         ans += e.w;
248         if (++added == n - 1) break;
249     }
250
251     return ans;
252 }
253
254 } // namespace Kruskal
255
256 int main() {
257     int n;
258     scanf("%d", &n);
259     for (int i = 0; i < n; i++) scanf("%lf %lf", &P[i].x, &P[i].y);
260
261     static std::vector<std::pair<int, int> > edges;
262     Delaunay::split(n, P, edges);
263
264     for (auto e : edges)
265         Kruskal::edges.emplace_back(e.first + 1, e.second + 1, dist(P[e.first], P[e.second]));
266     double ans = Kruskal::solve(n);
267     printf("%lld\n", ans);
268
269     return 0;
270 }
```

6.7 三维凸包

```

1 #include <cstdio>
2 #include <cmath>
3 #include <cstring>
4 #include <vector>
5 #include <algorithm>
6
7 const int MAXN = 505;
8 const double EPS = 1e-9;
9
10 int dcmp(double a, double b = 0) {
11     double d = a - b;
12     return std::abs(d) <= EPS ? 0 : (d > 0 ? 1 : -1);
```

```

13 }
14
15 double asqrt(double x) { return x <= EPS ? 0 : std::sqrt(x); }
16 double sqr(double x) { return x * x; }
17
18 struct Point {
19     double x, y, z;
20
21     Point(double x = 0, double y = 0, double z = 0) : x(x), y(y), z(z) {}
22
23     Point operator+(const Point &rhs) const {
24         return Point(x + rhs.x, y + rhs.y, z + rhs.z);
25     }
26     Point operator-(const Point &rhs) const {
27         return Point(x - rhs.x, y - rhs.y, z - rhs.z);
28     }
29     bool operator==(const Point &rhs) const {
30         return !dcmp(x, rhs.x) && !dcmp(y, rhs.y) && !dcmp(z, rhs.z);
31     }
32     bool operator<(const Point &rhs) const {
33         if (dcmp(x, rhs.x)) return x < rhs.x;
34         if (dcmp(y, rhs.y)) return y < rhs.y;
35         return z < rhs.z;
36     }
37
38     friend double dot(const Point &a, const Point &b) {
39         return a.x * b.x + a.y * b.y + a.z * b.z;
40     }
41     friend Point cross(const Point &a, const Point &b) {
42         return Point(a.y * b.z - a.z * b.y, a.z * b.x - a.x * b.z, a.x * b.y - a.y * b.x);
43     }
44     friend double mix(const Point &a, const Point &b, const Point &c) {
45         return dot(a, cross(b, c));
46     }
47
48     double length() const {
49         return std::sqrt(sqr(x) + sqr(y) + sqr(z));
50     }
51 } P[MAXN];
52
53 double volumn(const Point &a, const Point &b, const Point &c, const Point &d) {
54     return mix(b - a, c - a, d - a) / 6.0;
55 }
56
57 bool doesPointOnLine(const Point &p, const Point &s, const Point &t) {
58     return dcmp(cross(p - s, t - s).length()) == 0;
59 }
60
61 namespace ConvexHull3D {
62
63     struct Face {
64         int a, b, c;
65         bool judged;
66
67         Face(int a = 0, int b = 0, int c = 0) : a(a), b(b), c(c), judged(false) {}
68     };

```

```

69     int &operator[](int i) {
70         return i == 0 ? a : (i == 1 ? b : c);
71     }
72     const int &operator[](int i) const {
73         return i == 0 ? a : (i == 1 ? b : c);
74     }
75 };
76 std::vector<Face> ch;
77
78 bool same(int a, int b) {
79     return !dcmp(volumn(P[ch[a][0]], P[ch[a][1]], P[ch[a][2]], P[ch[b][0]]))
80         && !dcmp(volumn(P[ch[a][0]], P[ch[a][1]], P[ch[a][2]], P[ch[b][1]]))
81         && !dcmp(volumn(P[ch[a][0]], P[ch[a][1]], P[ch[a][2]], P[ch[b][2]]));
82 }
83
84 bool doesFaceHaveEdge(const Face &f, int s, int t) {
85     for (int i = 0; i < 3; i++) {
86         if (f[i] == s && f[(i + 1) % 3] == t) return true;
87         if (f[i] == t && f[(i + 1) % 3] == s) return true;
88     }
89     return false;
90 }
91
92 bool find(int n) {
93     for (int i = 2; i < n; i++) {
94         if (doesPointOnLine(P[i], P[0], P[1])) continue;
95         std::swap(P[2], P[i]);
96         for (int j = i + 1; j < n; j++) {
97             if (dcmp(volumn(P[0], P[1], P[2], P[j]))) {
98                 std::swap(P[3], P[j]);
99                 ch.emplace_back(0, 1, 2);
100                ch.emplace_back(0, 2, 1);
101                return true;
102            }
103        }
104    }
105    return false;
106 }
107
108 int mark[MAXN][MAXN];
109
110 void add(int pi, int &cnt) {
111     static std::vector<Face> temp;
112     temp.clear();
113
114     ++cnt;
115     for (auto f : ch) {
116         int a = f[0], b = f[1], c = f[2];
117         if (dcmp(volumn(P[pi], P[a], P[b], P[c])) < 0) {
118             mark[a][b] = mark[b][a] = cnt;
119             mark[a][c] = mark[c][a] = cnt;
120             mark[b][c] = mark[c][b] = cnt;
121         } else {
122             temp.push_back(f);
123         }
124     }

```

```

125     ch = temp;
126     for (auto f : temp) {
127         int a = f[0], b = f[1], c = f[2];
128         if (mark[a][b] == cnt) ch.emplace_back(b, a, pi);
129         if (mark[b][c] == cnt) ch.emplace_back(c, b, pi);
130         if (mark[c][a] == cnt) ch.emplace_back(a, c, pi);
131     }
132 }
133
134 bool getConvexHull(int n) {
135     std::sort(P, P + n);
136     n = std::unique(P, P + n) - P;
137     std::random_shuffle(P, P + n);
138
139     if (find(n)) {
140         memset(mark, 0, sizeof(mark));
141         int cnt = 0;
142         for (int i = 3; i < n; i++) add(i, cnt);
143     } else {
144         return false;
145     }
146
147     // E, F, V: The number of edges, faces and vertices when each face is a triangle
148     // EE, FF, VV: The number of edges, faces and vertices after dealing with co-planar triangles
149     int F = ch.size(), V = (4 + F) / 2, E = V + F - 2;
150     int FF = 0, EE = E;
151     for (int i = 0; i < ch.size(); i++) if (!ch[i].judged) {
152         static std::vector<Face> f;
153         f.clear();
154         for (int j = 0; j < ch.size(); j++) if (!ch[j].judged && same(i, j)) {
155             f.push_back(ch[j]);
156             ch[j].judged = true;
157         }
158         for (int j = 0; j < f.size(); j++) for (int k = 0; k < 3; k++) for (int l = 0; l < j; l++)
159             if (doesFaceHaveEdge(f[l], f[j][k], f[j][(k + 1) % 3])) {
160                 --EE;
161                 break;
162             }
163             ++FF;
164         }
165         int VV = EE + 2 - FF;
166
167         double area = 0; // surface area
168         for (auto f : ch) {
169             Point p = cross(P[f[0]] - P[f[1]], P[f[2]] - P[f[1]]);
170             area += p.length() / 2.0;
171         }
172
173         double vol = 0; // volume
174         const Point &vp = P[ch[0][0]];
175         for (auto f : ch) vol += std::abs(volumn(vp, P[f[0]], P[f[1]], P[f[2]]));
176
177         return true;
178     }
179 }
180 } // namespace ConvexHull3D

```

```

181
182 int main() {
183     int n;
184     scanf("%d", &n);
185     for (int i = 0; i < n; i++) scanf("%lf %lf %lf", &P[i].x, &P[i].y, &P[i].z);
186
187     // Deal with co-linear points
188     static bool banned[MAXN];
189     for (int i = 0; i < n; i++) if (!banned[i]) for (int j = 0; j < i; j++) if (!banned[j]) {
190         static std::vector<std::pair<Point, int> > l;
191         l.clear();
192         for (int k = 0; k < n; k++) if (!banned[k] && doesPointOnLine(P[k], P[i], P[j]))
193             l.emplace_back(P[k], k);
194         std::sort(l.begin(), l.end());
195         for (int k = 1; k < l.size() - 1; k++) banned[l[k].second] = true;
196     }
197
198     int p = 0;
199     for (int i = 0; i < n; i++) if (!banned[i]) P[p++] = P[i];
200
201     ConvexHull3D::getConvexHull(p);
202
203     return 0;
204 }
```

6.8 动态半凸壳

```

1 #include <cstdio>
2 #include <algorithm>
3
4 const int MAXN = 1000005;
5 const double EPS = 1e-9;
6
7 int dcmp(double a, double b = 0) {
8     double d = a - b;
9     return std::abs(d) <= EPS ? 0 : (d > 0 ? 1 : -1);
10 }
11
12 struct Splay {
13     struct Node {
14         Node *c[2], *fa, *pred, *succ;
15         double x, y;
16
17         Node() {}
18         Node(Node *fa, double x, double y) : x(x), y(y), c(), fa(fa), pred(NULL), succ(NULL) {}
19
20         int relation() { return fa->c[1] == this; }
21     } *_root, *_pool[MAXN], *_curr;
22
23     Splay() : root(NULL), _curr(_pool) {}
24
25     void rotate(Node *u) {
26         Node *o = u->fa;
27         int x = u->relation();
```

```

28
29     u->fa = o->fa;
30     if (u->fa) u->fa->c[o->relation()] = u;
31
32     o->c[x] = u->c[x ^ 1];
33     if (u->c[x ^ 1]) u->c[x ^ 1]->fa = o;
34
35     u->c[x ^ 1] = o;
36     o->fa = u;
37 }
38
39 Node *splay(Node *u, Node *targetFa = NULL) {
40     while (u->fa != targetFa) {
41         if (u->fa->fa == targetFa) rotate(u);
42         else if (u->relation() == u->fa->relation()) rotate(u->fa), rotate(u);
43         else rotate(u), rotate(u);
44     }
45     if (!targetFa) root = u;
46     return u;
47 }
48
49 double predSlope(Node *u) {
50     return u->pred ? (u->y - u->pred->y) / (u->x - u->pred->x) : 1.0 / 0.0;
51 }
52
53 double succSlope(Node *u) {
54     return u->succ ? (u->y - u->succ->y) / (u->x - u->succ->x) : -1.0 / 0.0;
55 }
56
57 Node *insert(double x, double y) {
58     if (!root) {
59         root = new (_curr++) Node(NULL, x, y);
60         return root;
61     }
62
63     Node **u = &root, *fa = NULL;
64
65     while (*u && dcmp(x, (*u)->x)) {
66         fa = *u;
67         u = &(*u)->c[dcmp(x, (*u)->x) > 0];
68     }
69
70     if (*u) {
71         if (dcmp((*u)->y, y) >= 0) return splay(*u);
72         (*u)->y = y;
73     } else {
74         (*u) = new (_curr++) Node(fa, x, y);
75
76         if ((*u)->relation()) {
77             (*u)->succ = fa->succ;
78             (*u)->pred = fa;
79             if (fa->succ) fa->succ->pred = *u;
80             fa->succ = *u;
81         } else {
82             (*u)->pred = fa->pred;
83             (*u)->succ = fa;

```

```

84         if (fa->pred) fa->pred->succ = *u;
85         fa->pred = *u;
86     }
87 }
88
89 Node *v = *u;
90 if (dcmp(predSlope(v), succSlope(v)) <= 0) {
91     splay(v->pred);
92     splay(v->succ, v->pred);
93     v->pred->succ = v->succ;
94     v->succ->pred = v->pred;
95     v->succ->c[0] = NULL;
96     return NULL;
97 }
98
99 while (v->pred && dcmp(predSlope(v->pred), predSlope(v)) <= 0)
100    v->pred = v->pred->pred;
101 if (v->pred) {
102     splay(v->pred);
103     splay(v, v->pred);
104     v->pred->succ = v;
105 }
106 v->c[0] = NULL;
107
108 while (v->succ && dcmp(succSlope(v->succ), succSlope(v)) >= 0)
109    v->succ = v->succ->succ;
110 if (v->succ) {
111     splay(v->succ);
112     splay(v, v->succ);
113     v->succ->pred = v;
114 }
115 v->c[1] = NULL;
116
117 return splay(v);
118 }
119
120 Node *find(double slope) {
121     Node *u = root;
122     while (true) {
123         if (dcmp(predSlope(u), slope) < 0) u = u->c[0];
124         else if (dcmp(succSlope(u), slope) > 0) u = u->c[1];
125         else return u;
126     }
127 }
128 } splay;
129
130 int main() {
131
132     return 0;
133 }
```

6.9 自适应辛普森积分

```
2 #include <cmath>
3
4 double f(double x) {
5     return std::sqrt(x);
6 }
7
8 double simpson(double l, double r) {
9     double mid = (l + r) / 2.0;
10    return (f(l) + 4.0 * f(mid) + f(r)) * (r - l) / 6.0;
11 }
12
13 double integral(double l, double r, double eps) {
14     double mid = (l + r) / 2.0;
15     double ST = simpson(l, r), SL = simpson(l, mid), SR = simpson(mid, r);
16     if (std::abs(SL + SR - ST) <= 15.0 * eps) return SL + SR + (SL + SR - ST) / 15.0;
17     return integral(l, mid, eps / 2.0) + integral(mid, r, eps / 2.0);
18 }
19
20 int main() {
21     double l, r, eps;
22     scanf("%lf %lf %lf", &l, &r, &eps);
23     printf("%.6f\n", integral(l, r, eps));
24
25     return 0;
26 }
```

6.10 由经纬度计算球面最短距离

$$\text{dist} = R \arccos(\cos(lat_0) \cos(lat_1) \cos(lon_0 - lon_1) + \sin(lat_0) \sin(lat_1))$$

7 博弈

7.1 K 倍动态减法

一堆石子有 n 个，第一次可以取 $1 \sim n - 1$ 个石子，之后至多可取 k 倍上次所取的石子数。 k 时的先手必败集合是一个 k 次的递推。

$$\begin{aligned}k = 1 : a_n &= 2a_{n-1} \\k = 2 : a_n &= a_{n-1} + a_{n-2} \\k = 3 : a_n &= a_{n-1} + a_{n-2} - a_{n-3} + 1\end{aligned}$$

```
1 #include <cstdio>
2
3 const int MAXN = 2000005;
4
5 int a[MAXN], b[MAXN];
6
7 int main() {
8     int n, k; // n <= 100,000,000, k <= 100,000
9     scanf("%d %d", &n, &k);
10
11    a[0] = b[0] = 1;
12    int i = 0, j = 0;
13    while (a[i] < n) {
14        ++i;
15        a[i] = b[i - 1] + 1;
16        while (a[j + 1] * k < a[i]) ++j;
17        if (a[j] * k < a[i]) b[i] = a[i] + b[j];
18        else b[i] = a[i];
19    }
20
21    if (a[i] == n) puts("Lose");
22    else {
23        int ans = 0;
24        while (n) {
25            if (n >= a[i]) {
26                n -= a[i];
27                ans = a[i];
28            }
29            --i;
30        }
31        printf("%d\n", ans);
32    }
33
34    return 0;
35 }
```

7.2 Nim-3

三人版的 Nim 取石子游戏，取完的人为一位，他的下一位为三位。

先手三位的条件是每个二进制位的 1 的个数为 3 的倍数。

```
1 #include <cstdio>
2
3 const int MAXN = 1000005;
4
5 long long a[MAXN];
6 int cnt[63];
7
8 int main() {
9     int n;
10    scanf("%d", &n);
11    for (int i = 0; i < n; i++) {
12        scanf("%lld", &a[i]);
13        for (int j = 0; j < 63; j++) cnt[j] += !(a[i] & (1ll << j));
14    }
15
16    bool isC = true;
17    for (int i = 0; i < 63; i++) if (cnt[i] % 3) {
18        isC = false;
19        break;
20    }
21
22    if (isC) {
23        puts("C");
24    } else {
25        bool isA = false;
26        for (int i = 0; i < n; i++) {
27            bool flag = true;
28            long long temp = 0;
29            for (int j = 0; j < 63; j++) {
30                int c = (cnt[j] - !(a[i] & (1ll << j))) % 3;
31                if (c == 1) {
32                    flag = false;
33                    break;
34                }
35                if (c == 2) temp |= (1ll << j);
36            }
37            if (flag && temp < a[i]) {
38                isA = true;
39                break;
40            }
41        }
42        puts(isA ? "A" : "B");
43    }
44
45    return 0;
46 }
```

8 其他

8.1 Java 高精度

```
1 import java.math.BigInteger;
2 import java.util.Scanner;
3
4 public class Main {
5     public static void main(String args[]) {
6         // read
7         Scanner in = new Scanner(System.in);
8         BigInteger a = in.nextBigInteger();
9
10        // 3 special numbers
11        a = BigInteger.ZERO;
12        a = BigInteger.ONE;
13        a = BigInteger.TEN;
14
15        // convert an int to BigInteger
16        BigInteger b = BigInteger.valueOf(2333);
17        BigInteger p = BigInteger.valueOf(998244353);
18
19        // convert a BigInteger to int
20        int i = b.intValue();
21        // convert a BigInteger to long
22        long l = b.longValue();
23
24        // operations:
25        // a + b;                                BigInteger add(BigInteger b);
26        a.add(b);
27        // a - b;                                BigInteger subtract(BigInteger b);
28        a.subtract(b);
29        // a * b;                                BigInteger multiply(BigInteger b);
30        a.multiply(b);
31        // a / b;                                BigInteger divide(BigInteger b);
32        a.divide(b);
33        // a % b;                                BigInteger mod(BigInteger b);
34        a.mod(b);
35        // -a;                                    BigInteger negate();
36        a.negative();
37        // a < 0 ? -1 : (a > 0 ? 1 : 0);      int signum();
38        a.signum();
39        // signum(a - b);                      int compareTo(BigInteger b);
40        a.compareTo(b);
41        // a == b;                                boolean equals(BigInteger b);
42        a.equals(b);
43
44        // abs(a);                                BigInteger abs();
45        a.abs();
46        // max(a, b);                            BigInteger max(BigInteger b);
47        a.max(b);
48        // min(a, b);                            BigInteger min(BigInteger b);
49        a.min(b);
```

```

50      // gcd(a, b);          BigInteger gcd(BigInteger b);
51      a.gcd(b);
52      // pow(a, i);          BigInteger pow(int i);
53      a.pow(i);
54
55      // modPow(a, b, p);    BigInteger modPow(BigInteger b, BigInteger p);
56      a.modPow(b, p);
57      // modPow(a, p - 2, p); BigInteger modInverse(BigInteger p);
58      a.modInverse(p);
59      // isPrime(a); (probability:1 - 0.5^i) boolean isProbablePrime(int certainty);
60      a.isProbablePrime(i);
61
62      // a << i;            BigInteger shiftLeft(int i);
63      a.shiftLeft(i);
64      // a >> i;            BigInteger shiftRight(int i);
65      a.shiftRight(i);
66      // a ^ b;              BigInteger xor(BigInteger b);
67      a.xor(b);
68      // a | b;              BigInteger or(BigInteger b);
69      a.or(b);
70      // a & b;              BigInteger and(BigInteger b);
71      a.and(b);
72      // ~a;                 BigInteger not();
73      a.not();
74      // a & ~b;             BigInteger andNot(BigInteger b);
75      a.andNot(b);
76
77      // ((a >> i) & 1) == 1; BigInteger testBit(int i);
78      a.testBit(i);
79      // a | (1 << i);    BigInteger setBit(int i);
80      a.setBit(i);
81      // a & ~(1 << i);  BigInteger clearBit(int i);
82      a.clearBit(i);
83      // a ^ (1 << i);   BigInteger flipBit(int i);
84      a.flipBit(i);
85      // a & -a;           BigInteger getLowerSetBit();
86      a.getLowestSetBit();
87      // __builtin_popcount(a); int bitCount();
88      a.bitCount();
89      // ceil(log2(this < 0 ? -this : this+1)) int bitLength();
90      a.bitLength();
91  }
92 }
```

8.2 三维偏序

```

1 #include <cstdio>
2 #include <algorithm>
3
4 const int MAXN = 100005;
5 const int MAXK = 200005;
6
7 struct Data {
8     int a, b, c, cnt, ans;
```

```

9
10    bool operator<(const Data &rhs) const {
11        return a < rhs.a || (a == rhs.a && b < rhs.b) || (a == rhs.a && b == rhs.b && c < rhs.c);
12    }
13
14    bool operator==(const Data &rhs) const {
15        return a == rhs.a && b == rhs.b && c == rhs.c;
16    }
17 } a[MAXN];
18
19 struct BIT {
20     int c[MAXK], n;
21
22     static constexpr int lowbit(int x) { return x & -x; }
23
24     void init(int n) {
25         this->n = n;
26         std::fill(c, c + n + 1, 0);
27     }
28
29     void update(int pos, int d) {
30         for (int i = pos; i <= n; i += lowbit(i)) c[i] += d;
31     }
32
33     int query(int pos) {
34         int res = 0;
35         for (int i = pos; i; i -= lowbit(i)) res += c[i];
36         return res;
37     }
38
39     void clear(int pos) {
40         for (int i = pos; i <= n; i += lowbit(i)) {
41             if (c[i]) c[i] = 0;
42             else break;
43         }
44     }
45 } bit;
46
47 void divide(Data *l, Data *r) {
48     if (l == r) {
49         l->ans += l->cnt - 1;
50         return;
51     }
52
53     Data *mid = l + (r - l) / 2;
54     divide(l, mid), divide(mid + 1, r);
55
56     static Data temp[MAXN];
57     for (Data *p = temp, *pl = l, *pr = mid + 1; p <= temp + (r - l); p++) {
58         if (pr > r || (pl <= mid && pl->b <= pr->b)) {
59             *p = *pl++;
60             bit.update(p->c, p->cnt);
61         } else {
62             *p = *pr++;
63             p->ans += bit.query(p->c);
64         }
65     }
66 }
```

```

65     }
66
67     for (Data *p = temp, *q = l; q <= r; q++, p++) {
68         bit.clear(q->c);
69         *q = *p;
70     }
71 }
72
73 int main() {
74     int n, k;
75     scanf("%d %d", &n, &k);
76
77     for (int i = 0; i < n; i++) {
78         scanf("%d %d %d", &a[i].a, &a[i].b, &a[i].c);
79         a[i].cnt = 1;
80     }
81
82     std::sort(a, a + n);
83     int cnt = 0;
84     for (int i = 0; i < n; i++) {
85         if (i && a[i] == a[i - 1]) a[cnt - 1].cnt++;
86         else a[cnt++] = a[i];
87     }
88
89     bit.init(k);
90     divide(a, a + cnt - 1);
91
92     static int ans[MAXN];
93     for (int i = 0; i < cnt; i++) ans[a[i].ans] += a[i].cnt;
94     for (int i = 0; i < n; i++) printf("%.0f\n", ans[i]);
95
96     return 0;
97 }
```

8.3 $O(1)$ 取模乘

```

1 long long mul(long long a, long long b, long long p) {
2     return (a * b - (long long) (a / (long double) p * b + 1e-3) * p + p) % p;
3 }
```

8.4 **std::bitset** 遍历

```

1 for (int i = bs._Find_first(); i < bs.size(); i = bs._Find_next(i)) {
2     // TODO
3 }
```

8.5 bitset 优化最长公共子序列

由于需要实现减法，故不能使用 **std::bitset**。

```

1 #include <cstdio>
2 #include <cstring>
3 #include <algorithm>
4
5 const int MAXN = 5005, SIGMA = 26;
6
7 struct Bitset {
8     static constexpr int W = 62;
9
10    static constexpr size_t size = MAXN / W + !(MAXN % W);
11    unsigned long long u[size];
12
13    Bitset &operator=(const Bitset &rhs) {
14        std::copy(rhs.u, rhs.u + size, u);
15        return *this;
16    }
17
18    void reset() {
19        std::fill(u, u + size, 0);
20    }
21    void set(int x) {
22        u[x / W] |= 1ull << (x % W);
23    }
24    bool get(int x) const {
25        return u[x / W] & (1ull << (x % W));
26    }
27
28    Bitset operator|(const Bitset &rhs) const {
29        Bitset res;
30        for (int i = 0; i < size; i++) res.u[i] = u[i] | rhs.u[i];
31        return res;
32    }
33    Bitset operator&(const Bitset &rhs) const {
34        Bitset res;
35        for (int i = 0; i < size; i++) res.u[i] = u[i] & rhs.u[i];
36        return res;
37    }
38    Bitset operator^(const Bitset &rhs) const {
39        Bitset res;
40        for (int i = 0; i < size; i++) res.u[i] = u[i] ^ rhs.u[i];
41        return res;
42    }
43
44    Bitset operator-(const Bitset &rhs) const {
45        Bitset res;
46        for (int i = 0; i < size; i++) res.u[i] = u[i] - rhs.u[i];
47        for (int i = 0; i < size; ++ i) if (res.u[i] < 0) {
48            res.u[i] += 1ll << W;
49            --res.u[i + 1];
50        }
51        res.u[size] = 0;
52        return res;
53    }
54
55    void shlOr1() {
56        unsigned long long c = 1;

```

```

57     for (int i = 0; i < size; ++ i) {
58         u[i] <= 1; u[i] |= c;
59         c = u[i] >> W & 1;
60         u[i] ^= c << W;
61     }
62 }
63
64 size_t count() const {
65     size_t res = 0;
66     for (int i = 0; i < size; ++ i) res += __builtin_popcountll(u[i]);
67     return res;
68 }
69 };
70 Bitset row, as[SIGMA], x;
71
72 char s[MAXN], t[MAXN];
73
74 int main() {
75     scanf("%s %s", s, t);
76     int n = strlen(s), m = strlen(t);
77
78     for (int i = 0; i < SIGMA; i++) as[i].reset();
79     for (int i = 0; i < n; i++) as[s[i] - 'a'].set(i);
80
81     row.reset();
82     for (int i = 0; i < m; i++) {
83         x = row | as[t[i] - 'a'];
84         row.shlOr1();
85         row = x - row;
86         row = (row ^ x) & x;
87         printf("%d\n", row.count());
88     }
89     return 0;
90 }
```

8.6 IO 优化

```

1 struct IO {
2     static constexpr int BUF_SIZE = 1048576;
3     char ibuf[BUF_SIZE], *ip, *ie, obuf[BUF_SIZE], *op;
4     std::streambuf *rbf, *ofb;
5
6     IO() : op(obuf) {
7         ip = ie = NULL;
8         rfb = std::cin.rdbuf();
9         ofb = std::cout.rdbuf();
10    }
11 ~IO() {
12     ofb->sputn(obuf, op - obuf);
13 }
14
15     char read() {
16         if (ip == ie) ie = (ip = ibuf) + rfb->sgetn(ibuf, BUF_SIZE);
17         return ip == ie ? -1 : *ip++;
18 }
```

```

18     }
19
20     void read(int &x) {
21         static char c;
22         static bool isneg;
23
24         for (isneg = false; !std::isdigit(c) && c != -1; c = read()) isneg |= c == '-';
25         for (x = 0; std::isdigit(c) && c != -1; x = x * 10 + (c ^ 48), c = read()) {}
26
27         x = (isneg ? -x : x);
28     }
29
30     template <typename T>
31     IO &operator>>(T &x) {
32         read(x);
33         return *this;
34     }
35
36     void print(char c) {
37         if (op == obuf + BUF_SIZE) {
38             ofb->sputn(obuf, BUF_SIZE);
39             op = obuf;
40         }
41         *op++ = c;
42     }
43
44     template <typename T>
45     void print(T x) {
46         static char buf[21];
47         static int cnt;
48         if (x < 0) {
49             print('-');
50             x = -x;
51         }
52         cnt = 0;
53         do {
54             buf[++cnt] = x % 10 + 48;
55         } while (x /= 10);
56         while (cnt) print((char) buf[cnt--]);
57     }
58
59     template <typename T>
60     IO &operator<<((const T &x) {
61         print(x);
62         return *this;
63     }
64 } in, out;

```