Pepcy's Templates for OI and ICPC

Pepcy Chen

updated on October 16, 2019

目录

电子版：pepcy.cf/icpc-templates/

1 Vim 配置

```
syntax on
set nocompatible
set backspace=2
set tabstop=4
set softtabstop
set shiftwidth=4
set expandtab
set smarttab
set autoindent
set cindent
set smartindent
set number
set ruler
set cursorline
set scrolloff=3
set magic
set showmatch
set showcmd
set history=1000
set mouse=a
set hlsearch
set incsearch
set ignorecase
set smartcase

map <C-a> gg0vG$

noremap <C-z> u
inoremap <C-z> <C-o>u

inoremap {<CR> {}<LEFT><CR><CR><UP><TAB>

map <C-p> i#include <bits/stdc++.h><CR><CR>int main() {<CR><CR>return 0;<UP><ESC>

map <F5> :w<CR>:!g++ % -o %< -lm -O2 -std=c++11; echo Exit code: $?; echo "==================="<CR>
imap <F5> <ESC>:w<CR>:!g++ % -o %< -lm -O2 -std=c++11; echo Exit code: $?; echo "==================="<CR>
```

This page is intetionally left blank

2 数据结构

2.1 树状数组

```cpp
#include <cstdio>
#include <algorithm>

const int MAXN = 100005;

struct BIT {
    int n;
    long long a[MAXN];

    static constexpr int lowbit(int x) { return x & -x; }

    void init(int n) {
        this->n = n;
        std::fill(a + 1, a + n + 1, 0);
    }

    void update(int pos, int val) {
        for (int i = pos; i <= n; i += lowbit(i)) a[i] += val;
    }

    long long query(int pos) {
        long long res = 0;
        for (int i = pos; i; i -= lowbit(i)) res += a[i];
        return res;
    }
    long long query(int l, int r) { return query(r) - query(l - 1); }
} bit;

int main() {
    int n;
    scanf("%d", &n);

    static int a[MAXN];
    for (int i = 1; i <= n; i++) scanf("%d", &a[i]);

    bit.init(n);
    for (int i = 1; i <= n; i++) bit.update(i, a[i]);

    int q;
    scanf("%d", &q);
    while (q--) {
        int op;
        scanf("%d", &op);

        if (op == 1) {
            int pos, val;
            scanf("%d %d", &pos, &val);
            bit.update(pos, val);
        } else {
```

```
            int l, r;
            scanf("%d %d", &l, &r);
            printf("%lld\n", bit.query(l, r));
        }
    }

    return 0;
}
```

2.2 线段树

```
#include <cstdio>
#include <algorithm>

const int MAXN = 100005;

class SegT {
private:
    struct Node {
        Node *lc, *rc;
        int l, r;
        long long sum, tag;

        Node() {}
        Node(int pos, int val) : lc(NULL), rc(NULL), l(pos), r(pos), sum(val), tag(0) {}
        Node(Node *lc, Node *rc) : lc(lc), rc(rc), l(lc->l), r(rc->r), tag(0) {
            maintain();
        }

        void add(long long d) {
            sum += (r - l + 1) * d;
            tag += d;
        }

        void pushDown() {
            if (tag) {
                lc->add(tag);
                rc->add(tag);
                tag = 0;
            }
        }

        void maintain() {
            sum = lc->sum + rc->sum;
        }

        void update(int l, int r, int d) {
            if (r < this->l || this->r < l) return;
            if (l <= this->l && this->r <= r) {
                add(d);
                return;
            }
            pushDown();
            lc->update(l, r, d);
```

```
            rc->update(l, r, d);
            maintain();
        }
        void update(int pos, int d) {
            if (l == r) {
                add(d);
                return;
            }
            pushDown();
            int mid = l + ((r - l) >> 1);
            (pos <= mid ? lc : rc)->update(pos, d);
            maintain();
        }

        long long query(int l, int r) {
            if (r < this->l || this->r < l) return 0;
            if (l <= this->l && this->r <= r) return sum;
            pushDown();
            return lc->query(l, r) + rc->query(l, r);
        }
        long long query(int pos) {
            if (l == r) return sum;
            pushDown();
            int mid = l + ((r - l) >> 1);
            return (pos <= mid ? lc : rc)->query(pos);
        }
    } *root, _pool[MAXN << 1], *_curr;

public:
    Node *build(int l, int r, int *a) {
        if (l == r) return new (_curr++) Node(l, a[l]);
        int mid = l + ((r - l) >> 1);
        return new (_curr++) Node(build(l, mid, a), build(mid + 1, r, a));
    }
    void build(int n, int *a) {
        _curr = _pool;
        root = build(1, n, a);
    }

    void update(int l, int r, int d) { root->update(l, r, d); }
    void update(int pos, int d) { root->update(pos, d); }
    long long query(int l, int r) { return root->query(l, r); }
    long long query(int pos) { return root->query(pos); }
} segT;

int main() {
    int n;
    scanf("%d", &n);

    static int a[MAXN];
    for (int i = 1; i <= n; i++) scanf("%d", &a[i]);

    segT.build(1, n, a);

    int q;
    scanf("%d", &q);
```

```
    while (q--) {
        int op, l, r;
        scanf("%d %d %d", &op, &l, &r);

        if (op == 1) {
            int d;
            scanf("%d", &d);
            segT.update(l, r, d);
        } else {
            printf("%lld\n", segT.query(l, r));
        }
    }

    return 0;
}
```

2.3 Splay

```
#include <cstdio>
#include <climits>
#include <algorithm>

const int MAXN = 100005;

template <typename T, size_t SIZE>
struct MemoryPool {
    char mem[SIZE * sizeof (T)], *memTop, *del[SIZE], **delTop;

    MemoryPool() : memTop(mem), delTop(del) {}

    void *alloc() {
        if (delTop != del) return (void *) *--delTop;
        char *res = memTop;
        memTop += sizeof (T);
        return (void *) res;
    }

    void free(void *p) { *delTop++ = (char *) p; }
};

struct Splay {
    struct Node {
        Node *c[2], *fa, *pred, *succ;
        int val, cnt, size;
        static MemoryPool<Node, MAXN> pool;

        Node() {}
        Node(Node *fa, int val) : val(val), size(1), cnt(1), c(), fa(fa), pred(NULL), succ(NULL) {}

        ~Node() {
            if (c[0]) delete c[0];
            if (c[1]) delete c[1];
        }

```

```
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
```

```
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();
```

```
        }
    } *root, _pool[MAXN], *_curr;

    Treap() : root(NULL), _curr(_pool) {}

    static int size(const Node *u) { return u ? u->size : 0; }

    Node *_build(int l, int r) {
        if (l > r) return NULL;
        int mid = l + ((r - l) >> 1);
        return new (_curr++) Node(mid, _build(l, mid - 1), _build(mid + 1, r));
    }
    void build(int l, int r) { root = _build(l, r); }

    Node *merge(Node *a, Node *b) {
        if (!a) return b;
        if (!b) return a;

        if (rand() % (a->size + b->size) < a->size) {
            a->pushDown();
            a->rc = merge(a->rc, b);
            a->maintain();
            return a;
        } else {
            b->pushDown();
            b->lc = merge(a, b->lc);
            b->maintain();
            return b;
        }
    }

    std::pair<Node *, Node *> split(Node *u, int pos) {
        std::pair<Node *, Node *> res(NULL, NULL);
        if (!u) return res;
        u->pushDown();
        if (pos <= size(u->lc)) {
            res = split(u->lc, pos);
            u->lc = res.second;
            u->maintain();
            res.second = u;
        } else {
            res = split(u->rc, pos - size(u->lc) - 1);
            u->rc = res.first;
            u->maintain();
            res.first = u;
        }
        return res;
    }

    void reverse(int l, int r) {
        std::pair<Node *, Node *> L = split(root, l - 1);
        std::pair<Node *, Node *> R = split(L.second, r - l + 1);
        R.first->reverse();
        root = merge(merge(L.first, R.first), R.second);
    }

```

```
    void print() {
        root->print();
        puts("");
    }
} treap;

int main() {
    int n, m;
    scanf("%d %d", &n, &m);

    treap.build(1, n);

    for (int i = 0, l, r; i < m; i++) {
        scanf("%d %d", &l, &r);
        treap.reverse(l, r);
    }

    treap.print();

    return 0;
}
```

2.5 替罪羊树

```
#include <cstdio>
#include <vector>
#include <algorithm>

const int MAXN = 100005;

template <typename T, size_t SIZE>
struct MemoryPool {
    char mem[SIZE * sizeof (T)], *top, *del[SIZE], **delTop;

    MemoryPool() { init(); }

    void init() {
        top = mem;
        delTop = del;
    }

    void *alloc() {
        if (delTop != del) return *(--delTop);
        char *res = top;
        top += sizeof (T);
        return (void *) res;
    }

    void free(void *p) { *(delTop++) = (char *) p; }
};

struct ScapegoatTree {
    static constexpr double ALPHA = 0.7;

```

```
31      struct Node {
32          Node *c[2];
33          int val, size, valid;
34          bool isDeled;
35
36          static MemoryPool<Node, MAXN> pool;
37
38          Node() {}
39          Node(int val) : val(val), size(1), valid(1), isDeled(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 = !isDeled + (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->isDeled) vec.push_back(u);
61          dfs(u->c[1], vec);
62          if (u->isDeled) 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) {
```

```
        return findKth(getRank(x + 1));
    }
} scapeT;
MemoryPool<ScapegoatTree::Node, MAXN> ScapegoatTree::Node::pool;

int main() {
    int n;
    scanf("%d", &n);
    while (n--) {
        int op, x;
        scanf("%d %d", &op, &x);

        if (op == 1) scapeT.insert(x);
        if (op == 2) scapeT.del(scapeT.getRank(x));
        if (op == 3) printf("%d\n", scapeT.getRank(x));
        if (op == 4) printf("%d\n", scapeT.findKth(x));
        if (op == 5) printf("%d\n", scapeT.pred(x));
        if (op == 6) printf("%d\n", scapeT.succ(x));
    }

    return 0;
}
```

2.6 Link-Cut Tree

```
#include <cstdio>
#include <algorithm>

const int MAXN = 30005;

struct LinkCutTree {
    struct Node {
        Node *c[2], *fa, *pathFa;
        int val, max;
        bool rev;

        Node() {}
        Node(int val) : val(val), max(val), rev(false), fa(NULL), pathFa(NULL), c() {}

        int relation() { return fa->c[1] == this; }

        void maintain() {
            max = val;
            if (c[0]) max = std::max(max, c[0]->max);
            if (c[1]) max = std::max(max, c[1]->max);
        }

        void pushDown() {
            if (rev) {
                std::swap(c[0], c[1]);
                if (c[0]) c[0]->rev ^= 1;
                if (c[1]) c[1]->rev ^= 1;
                rev = false;
            }
```

```
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);
```

```
            fa->maintain();
            return true;
        }

        void access() {
            expose();
            while (splice());
        }

        int query() {
            access();
            splay();
            return max;
        }
    } *N[MAXN], _pool[MAXN], *_curr;

    LinkCutTree() : _curr(_pool) {}

    void makeTree(int u, int val) {
        N[u] = new (_curr++) Node(val);
    }

    void link(int u, int v) {
        N[v]->evert();
        N[v]->pathFa = N[u];
    }

    void cut(int u, int v) {
        N[u]->evert();
        N[v]->access();
        N[v]->splay();
        N[v]->pushDown();
        N[v]->c[0]->fa = NULL;
        N[v]->c[0] = NULL;
        N[v]->maintain();
    }

    int query(int u, int v) {
        N[u]->evert();
        return N[v]->query();
    }
} lct;

int main() {

    return 0;
}
```

2.7 主席树

2.7.1 链上区间 k 小值

```
#include <cstdio>
#include <algorithm>
```

```

const int MAXN = 200005;
const int MAXN_LOG = 20;

template <typename T, size_t SIZE>
struct MemoryPool {
    char mem[sizeof(T) * SIZE], *top;

    MemoryPool() : top(mem) {}

    void *alloc() {
        char *res = top;
        top += sizeof (T);
        return (void *) res;
    }
};

class PSegT {
private:
    struct Node {
        Node *lc, *rc;
        int cnt;
        static MemoryPool<Node, MAXN * MAXN_LOG> pool;
        static Node *nil;

        Node(int cnt) : lc(nil), rc(nil), cnt(cnt) {
            static bool init = true;
            if (init) { lc = rc = this; init = false; }
        }
        Node(Node *lc = nil, Node *rc = nil) : lc(lc), rc(rc), cnt(lc->cnt + rc->cnt) {}

        void *operator new(size_t) {
            return pool.alloc();
        }

        Node *insert(int l, int r, int val) {
            if (val == l && val == r) return new Node(cnt + 1);
            int mid = l + ((r - l) >> 1);
            if (val <= mid) return new Node(lc->insert(l, mid, val), rc);
            else return new Node(lc, rc->insert(mid + 1, r, val));
        }

        int rank() const { return lc->cnt; }
    } *root[MAXN];
    int n;

public:
    void build(int *a, int n) {
        this->n = n;
        root[0] = new Node();
        for (int i = 1; i <= n; i++) root[i] = root[i - 1]->insert(1, n, a[i]);
    }

    int query(int l, int r, int k) {
        Node *L = root[l - 1], *R = root[r];
        int min = 1, max = n;
```

```
        while (min < max) {
            int mid = min + ((max - min) >> 1), temp = R->rank() - L->rank();
            if (k <= temp) L = L->lc, R = R->lc, max = mid;
            else L = L->rc, R = R->rc, k -= temp, min = mid + 1;
        }
        return min;
    }
} pSegT;
MemoryPool<PSegT::Node, MAXN * MAXN_LOG> PSegT::Node::pool;
PSegT::Node *PSegT::Node::nil = new PSegT::Node(0);

int map[MAXN];
void discrete(int *a, int n) {
    std::copy(a + 1, a + n + 1, map);
    std::sort(map, map + n);
    int *end = std::unique(map, map + n);
    for (int i = 1; i <= n; i++)
        a[i] = std::lower_bound(map, end, a[i]) - map + 1;
}

int main() {
    int n, q;
    scanf("%d %d", &n, &q);

    static int a[MAXN];
    for (int i = 1; i <= n; i++) scanf("%d", &a[i]);
    discrete(a, n);

    pSegT.build(a, n);

    while (q--) {
        int l, r, k;
        scanf("%d %d %d", &l, &r, &k);
        printf("%d\n", map[pSegT.query(l, r, k) - 1]);
    }

    return 0;
}
```

2.7.2 树上两点间 k 小值

```
#include <cstdio>
#include <climits>
#include <algorithm>

const int MAXN = 100005;
const int MAXN_LOG = 18;

template <typename T, size_t SIZE>
struct MemoryPool {
    char mem[SIZE * sizeof (T)], *top;

    MemoryPool() : top(mem) {}

    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  }
```

```

void dfs(Node *u, bool init = true) {
    if (init) {
        u->f[0] = u;
        u->dep = 1;
        u->seg = null->insert(0, INT_MAX, u->val);
    }

    for (int i = 1; i < MAXN_LOG; i++) u->f[i] = u->f[i - 1]->f[i - 1];

    for (Edge *e = u->e; e; e = e->next) if (e->v != u->f[0]) {
        e->v->f[0] = u;
        e->v->dep = u->dep + 1;
        e->v->seg = u->seg->insert(0, INT_MAX, e->v->val);

        dfs(e->v, false);
    }
}

Node *lca(Node *u, Node *v) {
    if (u->dep < v->dep) std::swap(u, v);

    for (int i = MAXN_LOG - 1; ~i; i--) {
        if (u->f[i]->dep >= v->dep) u = u->f[i];
    }

    for (int i = MAXN_LOG - 1; ~i; i--) {
        if (u->f[i] != v->f[i]) {
            u = u->f[i];
            v = v->f[i];
        }
    }

    return u == v ? u : u->f[0];
}

int query(int u, int v, int k) {
    Node *p = lca(&N[u], &N[v]);
    PSegT *su = N[u].seg, *sv = N[v].seg;
    PSegT *sp = p->seg, *sf = (p == p->f[0] ? null : p->f[0]->seg);

    int l = 0, r = INT_MAX;
    while (l < r) {
        int mid = l + (r - l) / 2;
        int temp = su->lc->cnt + sv->lc->cnt - sp->lc->cnt - sf->lc->cnt;
        if (k <= temp) {
            su = su->lc;
            sv = sv->lc;
            sp = sp->lc;
            sf = sf->lc;
            r = mid;
        } else {
            k -= temp;
            su = su->rc;
            sv = sv->rc;
            sp = sp->rc;
```

```
            sf = sf->rc;
            l = mid + 1;
        }
    }
    return l;
}

int main() {
    init();

    int n, q;
    scanf("%d %d", &n, &q);

    for (int i = 1; i <= n; i++) scanf("%d", &N[i].val);

    for (int i = 1, u, v; i < n; i++) {
        scanf("%d %d", &u, &v);
        addEdge(u, v);
    }

    dfs(&N[1]);

    int lastAns = 0;
    while (q--) {
        int u, v, k;
        scanf("%d %d %d", &u, &v, &k);
        u ^= lastAns;
        printf("%d", lastAns = query(u, v, k));
        q ? puts("") : 0;
    }

    return 0;
}
```

2.8 可持久化 Treap

```
#include <cstdio>
#include <cstdlib>
#include <algorithm>

const int MAXN = 100005;

struct PTreap {
    struct Node {
        Node *lc, *rc;
        int val, size;

        Node() {}
        Node(int val, Node *lc, Node *rc) : val(val), lc(lc), rc(rc), size((lc ? lc->size : 0) + 1 + (rc ?
            rc->size : 0)) {}
    } *root, _pool[MAXN * 20], *_curr;

    PTreap() : root(NULL), _curr(_pool) {}

```

```
    static int size(const Node *u) {
        return u ? u->size : 0;
    }

    Node *merge(Node *a, Node *b) {
        if (!a) return b;
        if (!b) return a;

        if (rand() % (size(a) + size(b)) < size(a)) {
            return new (_curr++) Node(a->val, a->lc, merge(a->rc, b));
        } else {
            return new (_curr++) Node(b->val, merge(a, b->lc), b->rc);
        }
    }

    std::pair<Node *, Node *> split(Node *u, int pos) {
        std::pair<Node *, Node *> res(NULL, NULL);
        if (!u) return res;

        if (pos <= size(u->lc)) {
            res = split(u->lc, pos);
            u = new (_curr++) Node(u->val, res.second, u->rc);
        } else {
            res = split(u->rc, pos - size(u->lc) - 1);
            u = new (_curr++) Node(u->val, u->lc, res.first);
        }

        return res;
    }
};

int main() {

    return 0;
}
```

2.9 并查集

```
#include <cstdio>

const int MAXN = 1000005;

struct DSU {
    int fa[MAXN];

    void init(int n) { for (int i = 0; i < n; i++) fa[i] = i; }
    int find(int x) { return x == fa[x] ? x : fa[x] = find(fa[x]); }
    void merge(int x, int y) { fa[find(y)] = find(x); }
    bool test(int x, int y) { return find(x) == find(y); }
} dsu;

int main() {
    int n, q;
    scanf("%d %d", &n, &q);
```

```

    dsu.init(n);

    while (q--) {
        int op, u, v;
        scanf("%d %d %d", &op, &u, &v);
        if (op == 0) dsu.merge(u, v);
        else puts(dsu.test(u, v) ? "Yes" : "No");
    }

    return 0;
}
```

2.10 Sparse Table

```
#include <cstdio>
#include <algorithm>

const int MAXN = 500005;
const int MAXN_LOG = 20;

class SparseTable {
private:
    int st[MAXN][MAXN_LOG], log[MAXN];

public:
    void init(int *a, int n) {
        int t = 0;
        for (int i = 0; i <= n; i++) {
            while (1 << (t + 1) <= i) t++;
            log[i] = t;
        }

        for (int i = 1; i <= n; i++) st[i][0] = a[i];

        for (int j = 1; j <= log[n]; j++) {
            for (int i = 1; i <= n; i++) {
                if (i + (1 << (j - 1)) <= n) st[i][j] = std::min(st[i][j - 1], st[i + (1 << (j - 1))][j -
                    1]);
                else st[i][j] = st[i][j - 1];
            }
        }
    }

    int query(int l, int r) {
        int t = log[r - l + 1];
        return std::min(st[l][t], st[r - (1 << t) + 1][t]);
    }
} st;

int main() {
    int n;
    scanf("%d", &n);

```

```
    static int a[MAXN];
    for (int i = 1; i <= n; i++) scanf("%d", &a[i]);

    st.init(a, n);

    int q;
    scanf("%d", &q);
    while (q--) {
        int l, r;
        scanf("%d %d", &l, &r);
        printf("%d\n", st.query(l, r));
    }

    return 0;
}
```

2.11 K-d Tree

2.11.1 曼哈顿距离最近点

```
#include <cstdio>
#include <climits>
#include <algorithm>
#include <functional>

const int MAXN = 500005;

struct Point {
    int x, y;
    Point(int x = 0, int y = 0) : x(x), y(y) {}
} P[MAXN];

int dist(const Point &a, const Point &b) {
    return abs(a.x - b.x) + abs(a.y - b.y);
}

std::function<bool(const Point &, const Point &)> cmp[2] = {
    [](const Point &a, const Point &b) {
        return a.y == b.y ? a.x < b.x : a.y < b.y;
    },
    [](const Point &a, const Point &b) {
        return a.x == b.x ? a.y < b.y : a.x < b.x;
    }
};

struct KDTree {
    struct Node {
        Node *c[2];
        Point p, r1, r2;

        Node() {}
        Node (Point p) : p(p), r1(p), r2(p) {
            c[0] = c[1] = NULL;
        }
```

```
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;
```

```
        while (*u) {
            int k = cmp[d](p, (*u)->p) ^ 1;
            d ^= 1;
            (*u)->r1.x = std::min(p.x, (*u)->r1.x);
            (*u)->r1.y = std::min(p.y, (*u)->r1.y);
            (*u)->r2.x = std::max(p.x, (*u)->r2.x);
            (*u)->r2.y = std::max(p.y, (*u)->r2.y);
            u = &(*u)->c[k];
        }
        *u = new (_curr++) Node(p);
    }

    int query(const Point &p) {
        int res = INT_MAX;
        root->query(p, res);
        return res;
    }
} kdT;

int main() {
    int n, m;
    scanf("%d %d", &n, &m);
    for (int i = 1; i <= n; i++) scanf("%d %d", &P[i].x, &P[i].y);

    kdT.root = kdT.build(1, n, P);

    while (m--) {
        int op;
        Point p;
        scanf("%d %d %d", &op, &p.x, &p.y);
        if (op == 1) kdT.insert(p);
        else printf("%d\n", kdT.query(p));
    }

    return 0;
}
```

2.11.2 全局欧几里得距离 k-远点对

```
#include <cstdio>
#include <climits>
#include <vector>
#include <queue>
#include <algorithm>
#include <functional>

const int MAXN = 100005;

struct Point {
    int x, y;
    Point(int x = 0, int y = 0) : x(x), y(y) {}
} P[MAXN];

long long dist(const Point &a, const Point &b) {
    return (long long) (a.x - b.x) * (a.x - b.x) + (long long) (a.y - b.y) * (a.y - b.y);
```

```
}

std::function<bool(const Point &, const Point &)> cmp[2] = {
    [](const Point &a, const Point &b) {
        return a.x == b.x ? a.y < b.y : a.x < b.x;
    },
    [](const Point &a, const Point &b) {
        return a.y == b.y ? a.x < b.x : a.y < b.y;
    }
};

struct KDTree {
    struct Node {
        Node *c[2];
        Point p, r1, r2;

        Node() {}
        Node(Point p) : p(p), r1(p), r2(p) {
            c[0] = c[1] = NULL;
        }

        void maintain() {
            if (c[0]) {
                r1.x = std::min(r1.x, c[0]->r1.x);
                r1.y = std::min(r1.y, c[0]->r1.y);
                r2.x = std::max(r2.x, c[0]->r2.x);
                r2.y = std::max(r2.y, c[0]->r2.y);
            }
            if (c[1]) {
                r1.x = std::min(r1.x, c[1]->r1.x);
                r1.y = std::min(r1.y, c[1]->r1.y);
                r2.x = std::max(r2.x, c[1]->r2.x);
                r2.y = std::max(r2.y, c[1]->r2.y);
            }
        }

        long long dist(const Point &p) {
            return std::max({::dist(p, r1), ::dist(p, r2),
                             ::dist(p, Point(r1.x, r2.y)),
                             ::dist(p, Point(r2.x, r1.y))});
        }

        void query(const Point &p, std::priority_queue<long long, std::vector<long long>, std::greater<long
             long> > &q) {
            long long d = ::dist(p, this->p);

            if (d > q.top()) q.pop(), q.push(d);
            if (!(c[0] || c[1])) return;

            long long dis[2] = {c[0] ? c[0]->dist(p) : INT_MIN,
                                c[1] ? c[1]->dist(p) : INT_MIN};
            int k = dis[0] < dis[1];

            c[k]->query(p, q);
            if (c[k ^ 1] && dis[k ^ 1] > q.top()) c[k ^ 1]->query(p, q);
        }
```

```
    } *root, _pool[MAXN], *_curr;

    KDTree() : root(NULL) {
        _curr = _pool;
    }

    Node *build(Point *l, Point *r, int d = 0) {
        if (l > r) return NULL;
        if (l == r) return new (_curr++) Node(*l);

        Point *mid = l + ((r - l) >> 1);
        std::nth_element(l, mid, r + 1, cmp[d]);

        Node *u = new (_curr++) Node(*mid);
        u->c[0] = build(l, mid - 1, d ^ 1);
        u->c[1] = build(mid + 1, r, d ^ 1);
        u->maintain();

        return u;
    }

    long long query(Point P[], int n, int k) {
        std::priority_queue<long long, std::vector<long long>, std::greater<long long> > q;
        while (!q.empty()) q.pop();
        for (int i = 0; i < k << 1; i++) q.push(-1);
        for (int i = 1; i <= n; i++) root->query(P[i], q);
        return q.top();
    }
} kdT;

int main() {
    int n, k;
    scanf("%d %d", &n, &k);
    for (int i = 1; i <= n; i++) scanf("%d %d", &P[i].x, &P[i].y);

    kdT.root = kdT.build(P + 1, P + n);
    printf("%lld\n", kdT.query(P, n, k));

    return 0;
}
```

2.12 单调队列

```
#include <cstdio>
#include <queue>

template <typename T, typename Cmp = std::less<T> > // maximum by default
struct MonoQueue {
    std::deque<T> q, m;
    Cmp cmp;

    void push(int x) {
        q.push_back(x);
        while (!m.empty() && cmp(m.back(), x)) m.pop_back();
```

```
        m.push_back(x);
    }

    void pop() {
        int x = q.front();
        q.pop_front();
        if (x == m.front()) m.pop_front();
    }

    size_t size() { return q.size(); }

    int top() { return m.front(); }
};

int main() {

    return 0;
}
```

2.13 Sqrt Tree

```
#include <cstdio>
#include <vector>
#include <algorithm>

inline int log2Up(int n) {
    int res = 0;
    while ((1 << res) < n) {
        res++;
    }
    return res;
}

template <typename T, T (*op)(T, T)>
class SqrtTree {
private:
    int n, lg, indexSz;
    std::vector<T> v;
    std::vector<int> clz, layers, onLayer;
    std::vector<std::vector<T> > pref, suf, between;

    void buildBlock(int layer, int l, int r) {
        pref[layer][l] = v[l];
        for (int i = l + 1; i < r; i++) {
            pref[layer][i] = op(pref[layer][i - 1], v[i]);
        }
        suf[layer][r - 1] = v[r - 1];
        for (int i = r - 2; i >= l; i--) {
            suf[layer][i] = op(v[i], suf[layer][i + 1]);
        }
    }

    void buildBetween(int layer, int lBound, int rBound, int betweenOffs) {
        int bSzLog = (layers[layer] + 1) >> 1;
```

```
        int bCntLog = layers[layer] >> 1;
        int bSz = 1 << bSzLog;
        int bCnt = (rBound - lBound + bSz - 1) >> bSzLog;
        for (int i = 0; i < bCnt; i++) {
            T ans;
            for (int j = i; j < bCnt; j++) {
                T add = suf[layer][lBound + (j << bSzLog)];
                ans = (i == j) ? add : op(ans, add);
                between[layer - 1][betweenOffs + lBound + (i << bCntLog) + j] = ans;
            }
        }
    }

    void buildBetweenZero() {
        int bSzLog = (lg + 1) >> 1;
        for (int i = 0; i < indexSz; i++) {
            v[n + i] = suf[0][i << bSzLog];
        }
        build(1, n, n + indexSz, (1 << lg) - n);
    }

    void updateBetweenZero(int bid) {
        int bSzLog = (lg + 1) >> 1;
        v[n + bid] = suf[0][bid << bSzLog];
        update(1, n, n + indexSz, (1 << lg) - n, n + bid);
    }

    void build(int layer, int lBound, int rBound, int betweenOffs) {
        if (layer >= (int)layers.size()) return;
        int bSz = 1 << ((layers[layer] + 1) >> 1);
        for (int l = lBound; l < rBound; l += bSz) {
            int r = std::min(l + bSz, rBound);
            buildBlock(layer, l, r);
            build(layer + 1, l, r, betweenOffs);
        }
        if (layer == 0) {
            buildBetweenZero();
        } else {
            buildBetween(layer, lBound, rBound, betweenOffs);
        }
    }

    void update(int layer, int lBound, int rBound, int betweenOffs, int x) {
        if (layer >= (int)layers.size()) return;
        int bSzLog = (layers[layer] + 1) >> 1;
        int bSz = 1 << bSzLog;
        int blockIdx = (x - lBound) >> bSzLog;
        int l = lBound + (blockIdx << bSzLog);
        int r = std::min(l + bSz, rBound);
        buildBlock(layer, l, r);
        if (layer == 0) {
            updateBetweenZero(blockIdx);
        } else {
            buildBetween(layer, lBound, rBound, betweenOffs);
        }
        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 +
            1) {
121         clz[0] = 0;
122         for (int i = 1; i < (int)clz.size(); i++) {
123             clz[i] = clz[i >> 1] + 1;
124         }
125         int tlg = lg;
126         while (tlg > 1) {
127             onLayer[tlg] = (int)layers.size();
128             layers.push_back(tlg);
129             tlg = (tlg + 1) >> 1;
130         }
131         for (int i = lg - 1; i >= 0; i--) {
132             onLayer[i] = std::max(onLayer[i], onLayer[i + 1]);
133         }
134         int betweenLayers = std::max(0, (int)layers.size() - 1);
135         int bSzLog = (lg + 1) >> 1;
136         int bSz = 1 << bSzLog;
137         indexSz = (n + bSz - 1) >> bSzLog;
138         v.resize(n + indexSz);
139         pref.assign(layers.size(), std::vector<T>(n + indexSz));
140         suf.assign(layers.size(), std::vector<T>(n + indexSz));
141         between.assign(betweenLayers, std::vector<T>((1 << lg) + bSz));
142         build(0, 0, n, 0);
143     }
144 };
```

```cpp

int Op(int a, int b) {
    return a + b; // gcd, min, max, &, |, ^
}

int main() {
    int n;
    scanf("%d", &n);
    std::vector<int> a(n);
    for (int i = 0; i < n; i++) scanf("%d", &a[i]);

    SqrtTree<int, Op> sqrtT(a);

    return 0;
}
```

3 树与图

3.1 单源最短路

3.1.1 Dijkstra

```cpp
#include <cstdio>
#include <climits>
#include <queue>
#include <algorithm>

const int MAXN = 100005;
const int MAXM = 500005;

struct Edge;
struct Node;

struct Node {
    Edge *e; // use std::vector<Edge> when dense graph.
    int dist;
    bool vis;
} N[MAXN];

struct Edge {
    Node *u, *v;
    Edge *next;
    int w;

    Edge() {}
    Edge(Node *u, Node *v, int w) : u(u), v(v), w(w), next(u->e) {}
} _pool[MAXM << 1], *_curr = _pool;

void addEdge(int u, int v, int w) {
    N[u].e = new (_curr++) Edge(&N[u], &N[v], w);
    N[v].e = new (_curr++) Edge(&N[v], &N[u], w);
}

namespace Dijkstra {
    struct HeapNode {
        Node *u;
        int dist;

        HeapNode(int dist, Node *u) : u(u), dist(dist) {}

        bool operator<(const HeapNode &rhs) const {
            return dist > rhs.dist;
        }
    };

    void dijkstra(Node *s) {
        std::priority_queue<HeapNode> q;
        s->dist = 0;
        q.emplace(0, s);
```

```

        while (!q.empty()) {
            Node *u = q.top().u;
            q.pop();

            if (u->vis) continue;
            u->vis = true;

            for (Edge *e = u->e; e; e = e->next) {
                if (e->v->dist > u->dist + e->w) {
                    e->v->dist = u->dist + e->w;

                    q.emplace(e->v->dist, e->v);
                }
            }
        }
    }

    int solve(int s, int t, int n) {
        for (int i = 1; i <= n; i++) {
            N[i].dist = INT_MAX;
            N[i].vis = false;
        }

        dijkstra(&N[s]);

        return N[t].dist;
    }
}

int main() {
    int n, m, s, t;
    scanf("%d %d %d %d", &n, &m, &s, &t);

    for (int i = 0, u, v, w; i < m; i++) {
        scanf("%d %d %d", &u, &v, &w);
        addEdge(u, v, w);
    }

    printf("%d\n", Dijkstra::solve(s, t, n));

    return 0;
}
```

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。

```
#include <cstdio>
#include <climits>
#include <queue>
#include <algorithm>

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::vetor<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;
```

```
        }

        return bellmanFord(&N[s], n) ? N[t].dist : -1;
    }
}

int main() {
    int n, m, s, t;
    scanf("%d %d %d %d", &n, &m, &s, &t);

    for (int i = 0, u, v, w; i < m; i++) {
        scanf("%d %d %d", &u ,&v, &w);
        addEdge(u, v, w);
    }

    printf("%d\n", BellmanFord::solve(s, t, n));

    return 0;
}
```

3.2 两点间 k 短路

```
#include <cstdio>
#include <climits>
#include <queue>
#include <algorithm>

const int MAXN = 1005;
const int MAXM = 100005;

struct Edge;
struct Node {
    Edge *e, *eo;
    int dist, times;
    bool vis;
} N[MAXN];

struct Edge {
    Node *u, *v;
    Edge *next;
    int w;

    Edge() {}
    Edge(Node *u, Node *v, int w, Edge *next) : u(u), v(v), w(w), next(next) {}
} _pool[MAXM << 1], *_curr = _pool;

void addEdge(int u, int v, int w) {
    N[u].e = new (_curr++) Edge(&N[u], &N[v], w, N[u].e);
    N[v].eo = new (_curr++) Edge(&N[v], &N[u], w, N[v].eo);
}

namespace Dijkstra {
    struct HeapNode {
        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;
```

```
            int curr = q.top().curr;
            int last = q.top().last;
            q.pop();

            ++u->times;
            if (u->times == k && u == t) return curr + last;
            if (u->times > k) continue;

            for (Edge *e = u->e; e; e = e->next) q.emplace(e->v, curr + e->w, e->v->dist);
        }

        return -1;
    }

    int solve(int s, int t, int k, int n) {
        Dijkstra::solve(t, n);
        return astar(&N[s], &N[t], k);
    }
}

int main() {
    int n, m;
    scanf("%d %d", &n, &m);
    for (int i = 0, u, v, w; i < m; i++) {
        scanf("%d %d %d", &u, &v, &w);
        addEdge(u, v, w);
    }

    int s, t, k;
    scanf("%d %d %d", &s, &t, &k);
    if (s == t) ++k;

    int ans = KthShortest::solve(s, t, k, n);
    printf("%d\n", ans);

    return 0;
}
```

3.3 斯坦纳树

```
#include <cstdio>
#include <climits>
#include <queue>
#include <algorithm>

const int MAXN = 50005;
const int MAXM = 300005;
const int MAXD = 5;

struct Edge;
struct Node {
    Edge *e;
    bool vis[1 << MAXD];
    long long f[1 << MAXD];
```

```
} N[MAXN];

struct Edge {
    Node *u, *v;
    Edge *next;
    int w;

    Edge() {}
    Edge(Node *u, Node *v, int w) : u(u), v(v), w(w), next(u->e) {}
} _pool[MAXM << 1], *_curr = _pool;

void addEdge(int u, int v, int w) {
    N[u].e = new (_curr++) Edge(&N[u], &N[v], w);
    N[v].e = new (_curr++) Edge(&N[v], &N[u], w);
}

namespace Dijkstra {
    struct HeapNode {
        Node *u;
        int s;
        long long dist;

        HeapNode(Node *u, int s, long long dist) : u(u), s(s), dist(dist) {}

        bool operator<(const HeapNode &rhs) const {
            return dist > rhs.dist;
        }
    };

    std::priority_queue<HeapNode> q;
    void dijkstra() {
        while (!q.empty()) {
            Node *u = q.top().u;
            int s = q.top().s;
            q.pop();

            if (u->vis[s]) continue;
            u->vis[s] = true;

            for (Edge *e = u->e; e; e = e->next) {
                if (e->v->f[s] > u->f[s] + e->w) {
                    e->v->f[s] = u->f[s] + e->w;
                    q.emplace(e->v, s, e->v->f[s]);
                }
            }
        }
    }

    void init(int n, int d) {
        for (int i = 1; i <= n; i++) for (int s = 0; s < 1 << d; s++) {
            N[i].f[s] = LLONG_MAX >> 1ll;
            N[i].vis[s] = false;
        }
        while (!q.empty()) q.pop();
    }
}
```

```

// O(3^d + 2^d * m \log n)
long long steiner(int n, int *p, int d) {
    Dijkstra::init(n, d);
    for (int i = 0; i < d; i++) N[p[i]].f[1 << i] = 0;

    for (int S = 0; S < 1 << d; S++) {
        for (int s = (S - 1) & S; s; s = (s - 1) & S)
            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]);
        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]);
        Dijkstra::dijkstra();
    }

    long long res = LLONG_MAX;
    for (int i = 1; i <= n; i++) res = std::min(res, N[i].f[(1 << d) - 1]);
    return res;
}

int main() {
    int n, m, d;
    scanf("%d %d %d", &n, &m, &d);
    for (int i = 0, u, v, w; i < m; i++) {
        scanf("%d %d %d", &u, &v, &w);
        addEdge(u, v, w);
    }

    static int p[MAXN];
    for (int i = 0; i < d; i++) scanf("%d", &p[i]);

    long long ans = steiner(n, p, d);
    printf("%lld\n", ans);

    return 0;
}
```

3.4 全局最短路

3.4.1 Floyd

```
#include <cstdio>
#include <climits>
#include <algorithm>

const int MAXN = 1005;

int dist[MAXN][MAXN];

void floyd(int n) {
    for (int k = 1; k <= n; k++) for (int i = 1; i <= n; i++) if (dist[i][k] < INT_MAX)
        for (int j = 1; j <= n; j++) if (dist[k][j] < INT_MAX)
            dist[i][j] = std::min(dist[i][j], dist[i][k] + dist[k][j]);
}

```

```
int main() {
    int n, m;
    scanf("%d %d", &n, &m);

    for (int i = 1; i <= n; i++) std::fill(dist[i] + 1, dist[i] + n + 1, INT_MAX);

    for (int i = 0, u, v, w; i < m; i++) {
        scanf("%d %d %d", &u, &v, &w);
        dist[u][v] = dist[v][u] = std::min(dist[u][v], w);
    }

    floyd(n);
    // dist[i][i] is the length of minimum circle through i if it's a digraph

    return 0;
}
```

3.4.2 Floyd 传递闭包

```
#include <cstdio>
#include <bitset>
#include <algorithm>

const int MAXN = 2005;

std::bitset<MAXN> G[MAXN];

void floyd(int n) {
    for (int k = 0; k < n; k++) for (int i = 0; i < n; i++)
        if (G[i][k]) G[i] |= G[k];
}

int main() {
    int n, m;
    scanf("%d %d", &n, &m);

    for (int i = 0, u, v; i < m; i++) {
        scanf("%d %d", &u, &v);
        G[u].set(v);
    }
    for (int i = 0; i < n; i++) G[i].set(i);

    floyd(n);

    return 0;
}
```

3.4.3 Johnson

```
#include <cstdio>
#include <climits>
#include <queue>
#include <algorithm>

```

```cpp
const int MAXN = 2005;
const int MAXM = 10005;

struct Edge;
struct Node {
    Edge *e;
    int dist[MAXN], cnt, h; // N[j].dist[i] denotes the shortest path from i to j
    bool inq, vis;
} N[MAXN];

struct Edge {
    Node *u, *v;
    Edge *next;
    int w;

    Edge() {}
    Edge(Node *u, Node *v, int w) : u(u), v(v), w(w), next(u->e) {}
} _pool[MAXM + MAXN], *_curr = _pool;

void addEdge(int u, int v, int w) {
    N[u].e = new (_curr++) Edge(&N[u], &N[v], w);
}

namespace BellmanFord {
    bool bellmanFord(Node *s, int n) {
        std::queue<Node *> q;
        q.push(s);
        s->h = 0;

        while (!q.empty()) {
            Node *u = q.front();
            q.pop();
            u->inq = false;

            for (Edge *e = u->e; e; e = e->next) {
                if (e->v->h > u->h + e->w) {
                    e->v->h = u->h + e->w;

                    if (++e->v->cnt >= n) return false;
                    if (!e->v->inq) {
                        e->v->inq = true;
                        q.push(e->v);
                    }
                }
            }
        }
        return true;
    }

    void solve(int s, int n) {
        for (int i = 0; i < n; i++) {
            N[i].h = INT_MAX;
            N[i].cnt = 0;
            N[i].inq = false;
        }

```

```
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
```

```
    for (int i = 1; i <= n; i++) addEdge(0, i, 0);
    BellmanFord::solve(0, n + 1);
    for (Edge *e = _pool; e < _curr - n; e++) e->w += e->u->h - e->v->h;
    for (int i = 1; i <= n; i++) {
        Dijkstra::solve(i, n);
        for (int j = 1; j <= n; j++) N[j].dist[i] -= N[i].h - N[j].h;
    }

    return 0;
}
```

3.5 点分治

```
#include <cstdio>
#include <vector>
#include <stack>
#include <algorithm>

const int MAXN = 100005;

struct Node;
struct Edge;

struct Node {
    Edge *e;
    bool solved;
    int size, max;
} N[MAXN];

struct Edge {
    Node *u, *v;
    Edge *next;

    Edge() {}
    Edge(Node *u, Node *v) : u(u), v(v), next(u->e) {}
} _pool[MAXN << 1], *_curr = _pool;

void addEdge(int u, int v) {
    N[u].e = new (_curr++) Edge(&N[u], &N[v]);
    N[v].e = new (_curr++) Edge(&N[v], &N[u]);
}

void dfs(Node *u, Node *fa, std::vector<Node *> &vec) {
    u->size = 1;
    u->max = 0;
    vec.push_back(u);
    for (Edge *e = u->e; e; e = e->next) if (!e->v->solved && e->v != fa) {
        dfs(e->v, u, vec);
        u->size += e->v->size;
        u->max = std::max(u->max, e->v->size);
    }
}

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

```cpp
#include <cstdio>
#include <climits>
#include <algorithm>

const int MAXN = 100005;

struct Node;
struct Edge;
struct Chain;

struct Node {
    Edge *e;
    Chain *c;
    Node *max, *fa;
    int dfn, dfnR, dep, size, val; // use dfnR when there're operations on subtrees
} N[MAXN];

struct Edge {
    Node *u, *v;
    Edge *next;

    Edge() {}
    Edge(Node *u, Node *v) : u(u), v(v), next(u->e) {}
} _poolE[MAXN << 1], *_currE = _poolE;
void addEdge(int u, int v) {
    N[u].e = new (_currE++) Edge(&N[u], &N[v]);
    N[v].e = new (_currE++) Edge(&N[v], &N[u]);
}

struct Chain {
    Node *top;

    Chain(Node *top = NULL) : top(top) {}
} _poolC[MAXN], *_currC = _poolC;

void dfs1(Node *u) {
    u->size = 1;

    for (Edge *e = u->e; e; e = e->next) if (e->v != u->fa) {
        e->v->dep = u->dep + 1;
        e->v->fa = u;

        dfs1(e->v);

        u->size += e->v->size;
        if (!u->max || u->max->size < e->v->size) u->max = e->v;
    }
}
void dfs2(Node *u, Node *fa = NULL) {
    static int dfsClock = 0;
    u->dfn = ++dfsClock;

    if (!u->fa || u->fa->max != u) u->c = new (_currC++) Chain(u);
    else u->c = u->fa->c;
```

```

    if (u->max) dfs2(u->max);
    for (Edge *e = u->e; e; e = e->next) if (e->v != u->fa && e->v != u->max) dfs2(e->v);

    u->dfnR = dfsClock;
}
void split() {
    N[1].dep = 1;
    dfs1(&N[1]);
    dfs2(&N[1]);
}

struct SegT {
    struct Node {
        int l, r;
        Node *lc, *rc;
        int max, tag;

        Node() {}
        Node(int pos, int val) : l(pos), r(pos), max(val), tag(0), lc(NULL), rc(NULL) {}
        Node(Node *lc, Node *rc) : l(lc->l), r(rc->r), lc(lc), rc(rc), tag(0) {
            maintain();
        }

        void add(int d) {
            max += d;
            tag += d;
        }

        void pushDown() {
            if (tag) {
                lc->add(tag);
                rc->add(tag);
                tag = 0;
            }
        }

        void maintain() {
            max = std::max(lc->max, rc->max);
        }

        void update(int l, int r, int d) {
            if (l > this->r || this->l > r) return;
            if (l <= this->l && this->r <= r) {
                add(d);
                return;
            }
            pushDown();
            lc->update(l, r, d);
            rc->update(l, r, d);
            maintain();
        }

        int query(int l, int r) {
            if (l > this->r || this->l > r) return INT_MIN;
            if (l <= this->l && this->r <= r) return max;
```

```
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;
```

```
    scanf("%d", &n);

    for (int i = 1; i <= n; i++) scanf("%d", &N[i].val);

    for (int i = 1, u, v; i < n; i++) {
        scanf("%d %d", &u, &v);
        addEdge(u, v);
    }

    split();

    static int temp[MAXN];
    for (int i = 1; i <= n; i++) temp[N[i].dfn] = N[i].val;
    segT.build(1, n, temp);

    int q;
    scanf("%d", &q);

    while (q--) {
        int op, u, v;
        scanf("%d %d %d", &op, &u, &v);
        if (op == 1) {
            int d;
            scanf("%d", &d);
            update(u, v, d);
        } else printf("%d\n", query(u, v));
    }

    return 0;
}
```

3.7 DSU on Tree / 树上启发式合并

```
#include <cstdio>
#include <vector>
#include <algorithm>

const int MAXN = 100005;

struct Graph {
    void addEdge(int u, int v) {
        G[u].push_back(v);
        G[v].push_back(u);
    }

    std::vector<int> G[MAXN];
    long long ans[MAXN];
    int col[MAXN];
    int n;
} G;

class DSUT {
public:
    void solve(Graph &G) {
```

```
        this->G = &G;
        findHeavy(1);
        dfs(1);
    }

private:
    Graph *G;
    int cnt[MAXN];
    int size[MAXN], son[MAXN];
    int skip, max;
    long long sum;

    void findHeavy(int u, int fa = -1) {
        size[u] = 1;
        for (int v : G->G[u]) if (v != fa) {
            findHeavy(v, u);
            size[u] += size[v];
            if (size[v] > size[son[u]]) son[u] = v;
        }
    }

    void edit(int u, int fa, int d) {
        cnt[G->col[u]] += d;
        if (d > 0) {
            if (cnt[G->col[u]] > max) {
                sum = G->col[u];
                max = cnt[G->col[u]];
            } else if (cnt[G->col[u]] == max) {
                sum += G->col[u];
            }
        }
        for (int v : G->G[u]) if (v != fa && v != skip) edit(v, u, d);
    }

    void dfs(int u, int fa = -1, bool keep = false) {
        for (int v : G->G[u]) if (v != fa && v != son[u]) dfs(v, u);
        if (son[u]) {
            dfs(son[u], u, true);
            skip = son[u];
        }
        edit(u, fa, 1);
        G->ans[u] = sum;
        skip = -1;
        if (!keep) {
            edit(u, fa, -1);
            sum = 0;
            max = 0;
        }
    }
} dsut;

int main() {
    int n;
    scanf("%d", &n);
    G.n = n;

```

```
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()) {
```

```
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

```cpp
#include <cstdio>
#include <climits>
#include <algorithm>

const int MAXN = 105;
const int MAXM = 5005;

struct Edge;
struct Node {
    Edge *e, *curr; // use std::vector<Edge> when dense graph
    int dist;
} N[MAXN];

struct Edge {
    Node *u, *v;
    Edge *next, *rev;
    int cap, flow;

    Edge() {}
    Edge(Node *u, Node *v, int cap) : u(u), v(v), cap(cap), flow(0), next(u->e) {}
} _pool[MAXM << 1], *_curr = _pool;

void addEdge(int u, int v, int cap) {
    N[u].e = new (_curr++) Edge(&N[u], &N[v], cap);
    N[v].e = new (_curr++) Edge(&N[v], &N[u], 0);
    N[u].e->rev = N[v].e;
    N[v].e->rev = N[u].e;
}

namespace ISAP {
    int cnt[MAXN], n;
    Node *s, *t;

    int flow(Node *u, int limit = INT_MAX) {
        if (u == t) return limit;

        int temp = limit;
        for (Edge *&e = u->curr; e; e = e->next) if (u->dist == e->v->dist + 1 && e->cap > e->flow) {
            int f = flow(e->v, std::min(temp, e->cap - e->flow));
            e->flow += f;
            e->rev->flow -= f;
            temp -= f;
            if (!temp) return limit;
        }

        if (!(--cnt[u->dist++])) s->dist = n + 1;
        ++cnt[u->dist];
        u->curr = u->e;

        return limit - temp;
    }

    long long solve(int s, int t, int n) {
        ISAP::n = n;
```

```
        ISAP::s = &N[s];
        ISAP::t = &N[t];
        for (int i = 1; i <= n; i++) {
            N[i].curr = N[i].e;
            N[i].dist = 0;
            cnt[i] = 0;
        }
        cnt[0] = n;

        long long res = 0;
        while (N[s].dist <= n) res += flow(&N[s]);
        return res;
    }
}

int main() {
    int n, m, s, t;
    scanf("%d %d %d %d", &n, &m, &s, &t);

    for (int i = 0, u, v, c; i < m; i++) {
        scanf("%d %d %d", &u, &v, &c);
        addEdge(u, v, c);
    }

    long long ans = ISAP::solve(s, t, n);
    printf("%lld\n", ans);

    return 0;
}
```

3.8.3 最小割输出方案

```
std::vector<Edge> cuts;
void bfs(int s, int t, int n) {
    static std::queue<Node *> q;
    while (!q.empty()) q.pop();
    q.push(&N[s]);
    N[s].vis = true;

    while (!q.empty()) {
        Node *u = q.front();
        q.pop();

        for (Edge e : u->e) if (e.cap > e.flow && !e.v->vis) {
            e.v->vis = true;
            q.push(e.v);
        }
    }

    for (int i = 0; i < n; i++) if (N[i].vis) for (Edge e : N[i].e) {
        if (e.cap == e.flow && e.cap > 0 && !e.v->vis) {
            cust.push_back(e);
        }
    }
}
```

3.9 Gomory-Hu Tree

```cpp
#include <cstdio>
#include <climits>
#include <vector>
#include <queue>
#include <algorithm>

const int MAXN = 3005;
const int MAXN_LOG = 13;

struct Edge;
struct Node {
    std::vector<Edge> e;
    Edge *curr;
    int level;
    bool vis;
} N[MAXN];

struct Edge {
    Node *u, *v;
    int cap, flow, rev;

    Edge(Node *u, Node *v, int cap, int rev) : u(u), v(v), cap(cap), flow(0), rev(rev) {}
};

bool G[MAXN][MAXN];

void addEdge(int u, int v, int cap) {
    N[u].e.emplace_back(&N[u], &N[v], cap, N[v].e.size());
    N[v].e.emplace_back(&N[v], &N[u], cap, N[u].e.size() - 1);
    G[u][v] = G[v][u] = true;
}

namespace Dinic {
    bool level(Node *s, Node *t, int n) {
        for (int i = 1; i <= n; i++) N[i].level = 0;
        static std::queue<Node *> q;
        q.push(s);
        s->level = 1;
        while (!q.empty()) {
            Node *u = q.front();
            q.pop();

            for (Edge &e : u->e) {
                if (e.cap > e.flow && e.v->level == 0) {
                    e.v->level = u->level + 1;
                    q.push(e.v);
                }
            }
        }
        return t->level;
    }

    int findPath(Node *u, Node *t, int limit = INT_MAX) {
        if (u == t) return limit;
```

```
        int res = 0;
        for (Edge *&e = u->curr; e && e <= &u->e.back(); e++) {
            if (e->cap > e->flow && e->v->level == u->level + 1) {
                int flow = findPath(e->v, t, std::min(limit, e->cap - e->flow));
                if (flow > 0) {
                    e->flow += flow;
                    e->v->e[e->rev].flow -= flow;
                    limit -= flow;
                    res += flow;
                    if (!limit) return res;
                } else e->v->level = -1;
            }
        }
        return res;
    }

    int solve(int s, int t, int n) {
        for (int i = 1; i <= n; i++) for (Edge &e : N[i].e) e.flow = 0;

        int res = 0;
        while (level(&N[s], &N[t], n)) {
            for (int i = 1; i <= n; i++) N[i].curr = &N[i].e.front();
            int flow;
            while ((flow = findPath(&N[s], &N[t])) > 0) res += flow;
        }
        return res;
    }

    void bfs(Node *s, int n) {
        for (int i = 1; i <= n; i++) N[i].vis = false;

        static std::queue<Node *> q;
        q.push(s);
        s->vis = true;
        while (!q.empty()) {
            Node *u = q.front();
            q.pop();
            for (const Edge &e : u->e) {
                if (e.cap > e.flow && !e.v->vis) {
                    e.v->vis = true;
                    q.push(e.v);
                }
            }
        }
    }
}

namespace GomoryHuTree {
    struct Edge;
    struct Node {
        Edge *e;
        Node *f[MAXN_LOG];
        int dep, min[MAXN_LOG];
    } N[MAXN];

    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
```

```
        std::vector<int> ln, rn;
        for (int u : nodes) {
            if (::N[u].vis) ln.push_back(u);
            else rn.push_back(u);
        }

        build(ln, n);
        build(rn, n);
    }

    void build(int n) {
        _curr = _pool;
        std::vector<int> vec(n);
        for (int i = 1; i <= n; i++) vec[i - 1] = i;
        build(vec, n);

        N[1].min[0] = INT_MAX;
        dfs(&N[1]);
    }
}

int main() {
    int n, m;
    scanf("%d %d", &n, &m);
    for (int i = 0, u, v, w; i < m; i++) {
        scanf("%d %d %d", &u, &v, &w);
        addEdge(u, v, w);
    }

    GomoryHuTree::build(n);

    int q;
    scanf("%d", &q);
    while (q--) {
        int u, v;
        scanf("%d %d", &u, &v);

        int ans = GomoryHuTree::query(u, v);
        printf("%d\n", ans);
    }

    return 0;
}
```

3.10 费用流

3.10.1 Dijkstra 费用流

```
#include <cstdio>
#include <climits>
#include <vector>
#include <queue>
#include <algorithm>

```

```
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[] first
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;
```

```
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;
```

```

struct Graph {
    struct Edge {
        int v, cap, flow, cost, rev;

        Edge(int v, int cap, int cost, int rev) : v(v), cap(cap), flow(0), cost(cost), rev(rev) {}
    };

    void addEdge(int u, int v, int cap, int cost) {
        G[u].emplace_back(v, cap, cost, G[v].size());
        G[v].emplace_back(u, 0, -cost, G[u].size() - 1);
    }

    std::vector<Edge> G[MAXN];
    int n;
} G;

class MCMF {
public:
    std::pair<int, int> solve(int s, int t, Graph &G) {
        this->s = s;
        this->t = t;
        this->G = &G;

        int flow = 0, cost = 0;
        int rem = INT_MAX;
        while (argue()) {
            std::fill_n(ptr, G.n, 0);
            int d = dfs(s, rem - flow);
            flow += d;
            cost += d * dist[t];
        }

        return std::make_pair(flow, cost);
    }

private:
    int s, t;
    Graph *G;
    int dist[MAXN], ptr[MAXN];
    bool vis[MAXN];

    bool argue() {
        std::fill_n(dist, G->n, INT_MAX);
        std::fill_n(vis, G->n, false);
        dist[s] = 0;

        std::queue<int> q;
        q.push(s);
        while (!q.empty()) {
            int u = q.front();
            q.pop();

            vis[u] = false;
            for (auto &e : G->G[u]) {
                if (e.cap > e.flow && dist[e.v] > dist[u] + e.cost) {
```

```
                    dist[e.v] = dist[u] + e.cost;
                    if (!vis[e.v]) {
                        vis[e.v] = true;
                        q.push(e.v);
                    }
                }
            }
        }

        return dist[t] != INT_MAX;
    }

    int dfs(int u, int r) {
        if (u == t) return r;

        vis[u] = true;
        int res = 0;

        for (int &i = ptr[u]; i < G->G[u].size(); i++) {
            auto &e = G->G[u][i];
            if (e.cap > e.flow && dist[e.v] == dist[u] + e.cost && !vis[e.v]) {
                int d = dfs(e.v, std::min(r - res, e.cap - e.flow));
                res += d;
                e.flow += d;
                G->G[e.v][e.rev].flow -= d;
                if (res == r) {
                    vis[u] = false;
                    break;
                }
            }
        }

        return res;
    }
} mcmf;

int main() {
    int n, m, s, t;
    scanf("%d %d %d %d", &n, &m, &s, &t);

    G.n = n;
    for (int i = 0, u, v, c, w; i < m; i++) {
        scanf("%d %d %d %d", &u, &v, &c, &w);
        G.addEdge(u, v, c, w);
    }

    auto [flow, cost] = mcmf.solve(s, t, G);
    printf("%d %d\n", flow, cost);

    return 0;
}
```

3.11 上下界网络流

3.11.1 无源无汇上下界可行流

```cpp
#include <cstdio>
#include <climits>
#include <vector>
#include <queue>
#include <algorithm>

const int MAXN = 205;
const int MAXM = 10205;

struct Edge;
struct Node;

struct Node {
    std::vector<Edge> e;
    Edge *curr;
    int level, extra;
} N[MAXN];

struct Edge {
    Node *u, *v;
    int cap, flow, rev;

    Edge(Node *u, Node *v, int cap, int rev)
        : u(u), v(v), cap(cap), flow(0), rev(rev) {}
};

struct Pair {
    int u, num, lower;

    Pair() {}
    Pair(int u, int num, int lower) : u(u), num(num), lower(lower) {}

    int flow() const {
        return N[u].e[num].flow + lower;
    }
} E[MAXM];

Pair addEdge(int u, int v, int lower, int upper) {
    int cap = upper - lower;
    N[u].e.push_back(Edge(&N[u], &N[v], cap, N[v].e.size()));
    N[v].e.push_back(Edge(&N[v], &N[u], 0, N[u].e.size() - 1));

    N[u].extra -= lower;
    N[v].extra += lower;

    return Pair(u, N[u].e.size() - 1, lower);
}

namespace Dinic {
    bool level(Node *s, Node *t, int n) {
        for (int i = 0; i < n; i++) N[i].level = 0;
        static std::queue<Node *> q;
```

```
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++) {
```

```
        if (N[i].extra > 0) {
            sum += N[i].extra;
            addEdge(s, i, 0, N[i].extra);
        } else if (N[i].extra < 0) {
            addEdge(i, t, 0, -N[i].extra);
        }
    }

    int maxFlow = Dinic::solve(s, t, n + 2);
    if (maxFlow < sum) {
        puts("NO");
    } else {
        puts("YES");
        for (int i = 0; i < m; i++) printf("%d\n", E[i].flow());
    }

    return 0;
}
```

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。

```
#include <cstdio>
#include <climits>
#include <vector>
#include <queue>
#include <algorithm>

const int MAXN = 205;
const int MAXM = 10000;

struct Edge;
struct Node;

struct Node {
    std::vector<Edge> e;
    Edge *curr;
    int level, extra;
} N[MAXN];

struct Edge {
    Node *u, *v;
    int cap, flow, rev;

    Edge(Node *u, Node *v, int cap, int rev) : u(u), v(v), cap(cap), flow(0), rev(rev) {}
};

void addEdge(int u, int v, int lower, int upper) {
    int cap = upper - lower;
    N[u].e.push_back(Edge(&N[u], &N[v], cap, N[v].e.size()));
    N[v].e.push_back(Edge(&N[v], &N[u], 0, N[u].e.size() - 1));

    N[u].extra -= lower;
    N[v].extra += lower;
}
```

```

namespace Dinic {
    bool level(Node *s, Node *t, int n) {
        for (int i = 0; i < n; i++) N[i].level = 0;
        static std::queue<Node *> q;
        q.push(s);
        s->level = 1;
        while (!q.empty()) {
            Node *u = q.front();
            q.pop();
            for (Edge *e = &u->e.front(); e <= &u->e.back(); e++) {
                if (e->cap > e->flow && e->v->level == 0) {
                    e->v->level = u->level + 1;
                    q.push(e->v);
                }
            }
        }
        return t->level;
    }

    int findPath(Node *u, Node *t, int limit = INT_MAX) {
        if (u == t) return limit;
        int res = 0;
        for (Edge *&e = u->curr; e <= &u->e.back(); e++) {
            if (e->cap > e->flow && e->v->level == u->level + 1) {
                int flow = findPath(e->v, t, std::min(limit, e->cap - e->flow));
                if (flow > 0) {
                    e->flow += flow;
                    e->v->e[e->rev].flow -= flow;
                    limit -= flow;
                    res += flow;
                    if (limit <= 0) return res;
                } else e->v->level = -1;
            }
        }
        return res;
    }

    int solve(int s, int t, int n) {
        int res = 0;
        while (level(&N[s], &N[t], n)) {
            for (int i = 0; i < n; i++) N[i].curr = &N[i].e.front();
            int flow;
            while ((flow = findPath(&N[s], &N[t])) > 0) res += flow;
        }
        return res;
    }
}

int main() {
    int n, m, s, t;
    scanf("%d %d %d %d", &n, &m, &s, &t);
    const int S = 0, T = n + 1;

    for (int i = 0, u, v, lower, upper; i < m; i++) {
        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);
```

```
    N[v].e = new (_curr++) Edge(&N[v], &N[u], 0);
    (N[u].e->rev = N[v].e)->rev = N[u].e;

    N[u].extra -= lower;
    N[v].extra += lower;

    return N[u].e;
}

namespace Dinic {
    bool level(Node *s, Node *t, int n) {
        for (int i = 0; i < n; i++) N[i].level = 0;
        static std::queue<Node *> q;
        q.push(s);
        s->level = 1;
        while (!q.empty()) {
            Node *u = q.front();
            q.pop();
            for (Edge *e = u->e; e; e = e->next) {
                if (e->cap > e->flow && e->v->level == 0) {
                    e->v->level = u->level + 1;
                    q.push(e->v);
                }
            }
        }
        return t->level;
    }

    int findPath(Node *u, Node *t, int limit = INT_MAX) {
        if (u == t) return limit;
        int res = 0;
        for (Edge *&e = u->curr; e; e = e->next) {
            if (e->cap > e->flow && e->v->level == u->level + 1) {
                int flow = findPath(e->v, t, std::min(limit, e->cap - e->flow));
                if (flow > 0) {
                    e->flow += flow;
                    e->rev->flow -= flow;
                    limit -= flow;
                    res += flow;
                    if (limit <= 0) return res;
                } else e->v->level = -1;
            }
        }
        return res;
    }

    int solve(int s, int t, int n) {
        int res = 0;
        while (level(&N[s], &N[t], n)) {
            for (int i = 0; i < n; i++) N[i].curr = N[i].e;
            int flow;
            while ((flow = findPath(&N[s], &N[t])) > 0) res += flow;
        }
        return res;
    }
}
```

```

int main() {
    int n, m, s, t;
    scanf("%d %d %d %d", &n, &m, &s, &t);
    const int S = 0, T = n + 1;

    for (int i = 0, u, v, lower, upper; i < m; i++) {
        scanf("%d %d %d %d", &u, &v, &lower, &upper);
        addEdge(u, v, lower, upper);
    }
    Edge *e = addEdge(t, s, 0, INT_MAX);
    int sum = 0;
    for (int i = 1; i <= n; i++) {
        if (N[i].extra > 0) {
            sum += N[i].extra;
            addEdge(S, i, 0, N[i].extra);
        } else if (N[i].extra < 0) {
            addEdge(i, T, 0, -N[i].extra);
        }
    }

    int maxFlow = Dinic::solve(S, T, n + 2);
    if (maxFlow < sum) {
        puts("No");
    } else {
        puts("Yes");
        int flow = e->flow;
        e->cap = e->rev->cap = 0;
        printf("%d\n", flow - Dinic::solve(t, s, n + 2));
    }

    return 0;
}
```

3.12 Tarjan

3.12.1 有向图找强连通分量并缩点建图

```
#include <cstdio>
#include <stack>
#include <algorithm>

const int MAXN = 100005;
const int MAXM = 500005;

template <typename T>
struct Edge {
    T *u, *v;
    Edge *next;

    Edge() {}
    Edge(T *u, T *v) : u(u), v(v), next(u->e) {}
};

```

```
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;
```

```
        while (!s.empty()) s.pop();

        for (int i = 1; i <= n; i++) if (!N[i].dfn) dfs(&N[i]);
    }

    void rebuild() {
        for (Edge<Node> *e = _poolN; e != _currN; e++) if (e->u->c != e->v->c) addEdge(e->u->c, e->v->c);
    }
}

int main() {
    int n, m;
    scanf("%d %d", &n, &m);

    for (int i = 0, u, v; i < m; i++) {
        scanf("%d %d", &u, &v);
        addEdge(u, v);
    }

    Tarjan::findSCC(n);
    Tarjan::rebuild();

    return 0;
}
```

3.12.2 无向图找边双连通分量并缩点建图

```
#include <cstdio>
#include <vector>
#include <queue>
#include <algorithm>

const int MAXN = 300005;

template <typename T>
struct Edge {
    T *u, *v;
    Edge *next;

    Edge() {}
    Edge(T *u, T *v) : u(u), v(v), next(u->e) {}
};

struct Comp {
    Edge<Comp> *e;
    int dist;
} C[MAXN];
Edge<Comp> _poolC[MAXN << 1], *_currC = _poolC;
void addEdgeC(int u, int v) {
    C[u].e = new (_currC++) Edge<Comp>(&C[u], &C[v]);
    C[v].e = new (_currC++) Edge<Comp>(&C[v], &C[u]);
}

struct Node {
    Edge<Node> *e;
```

```
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];
```

```
Edge<Node> _poolN[MAXN << 1], *_currN = _poolN;

void addEdge(int u, int v) {
    N[u].e = new (_currN++) Edge<Node>(&N[u], &N[v]);
    N[v].e = new (_currN++) Edge<Node>(&N[v], &N[u]);
}

namespace Tarjan {
    int dfsClock, compCnt;
    std::stack<Edge<Node> *> s;
    std::vector<Node *> cuts;

    void dfs(Node *u, Node *fa = NULL) {
        u->dfn = ++dfsClock;
        u->low = u->dfn;
        int size = 0;

        for (Edge<Node> *e = u->e; e; e = e->next) if (e->v != fa) {
            s.push(e);

            if (e->v->dfn) {
                u->low = std::min(u->low, e->v->dfn);
            } else {
                ++size;
                dfs(e->v, u);
                u->low = std::min(u->low, e->v->low);
                if (e->v->low >= u->dfn) {
                    u->isCut = true;
                    cuts.push_back(u);

                    Comp *c = &C[++compCnt];
                    while (true) {
                        Edge<Node> *t = s.top();
                        s.pop();

                        if (t->u->c != c) t->u->c = c;
                        if (t->v->c != c) t->v->c = c;

                        if (t->u == u && t->v == e->v) break;
                    }
                }
            }
        }

        if (!fa && size > 1) {
            cuts.push_back(u);
            u->isCut = true;
        }
    }

    void findVCC(int n) {
        dfsClock = 0;
        while (!s.empty()) s.pop();
        dfs(&N[1]);
    }

```

```
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;
```

```
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));
```

```
        }
    }

    TwoSat::solve(n);

    rebuild();

    TopoSort::solve(TwoSat::sccCnt);

    return 0;
}
```

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。

```
#include <cstdio>
#include <vector>
#include <algorithm>

const int MAXN = 100005;
const int MAXM = 200005;

struct Edge;
struct Node {
    Edge *e, *curr;
    int deg;
} N[MAXN];

struct Edge {
    Node *u, *v;
    Edge *next, *rev;
    bool used;

    Edge() {}
    Edge(Node *u, Node *v) : u(u), v(v), next(u->e), used(false) {}
} _pool[MAXM << 1], *_curr = _pool;

void addEdge(int u, int v) {
    N[u].e = new (_curr++) Edge(&N[u], &N[v]);
    N[v].e = new (_curr++) Edge(&N[v], &N[u]);
    N[u].e->rev = N[v].e;
    N[v].e->rev = N[u].e;
    ++N[u].deg;
    ++N[v].deg;
}

namespace Euler {
    std::vector<Edge *> path;

    void dfs(Node *u) {
        for (Edge *&e = u->curr; e; e = e->next) if (!e->used) {
            e->used = e->rev->used = true;
```

```
            dfs(e->v);
            path.push_back(e);
        }
    }

    bool solve(int n) {
        int cnt = 0;
        for (int i = 1; i <= n; i++) cnt += N[i].deg % 2;
        if (cnt > 2) return false;
        for (int i = 1; i <= n; i++) N[i].curr = N[i].e;
        int s = -1;
        if (cnt) for (int i = 1; i <= n; i++) if (N[i].deg % 2) {
            s = i;
            break;
        }
        s = (s == -1 ? 1 : s);
        dfs(&N[s]);
        std::reverse(path.begin(), path.end());
        return true;
    }
}

int main() {
    int n, m;
    scanf("%d %d", &n, &m);
    for (int i = 0, u, v; i < m; i++) {
        scanf("%d %d", &u, &v);
        addEdge(u, v);
    }

    if (Euler::solve(n)) {
        printf("%d", Euler::path[0]->u - N);
        for (int i = 0; i < Euler::path.size(); i++) {
            printf(" -> %d", Euler::path[i]->v - N);
        }
        puts("");
    } else {
        puts("-1");
    }

    return 0;
}
```

3.14.2 无递归版本 有向图

```
#include <cstdio>
#include <vector>
#include <stack>
#include <algorithm>

const int MAXN = 100005;
const int MAXM = 400005;

struct Edge;
struct Node {
```

```
    Edge *e, *curr;
    int ideg, odeg;
} N[MAXN];

struct Edge {
    Node *u, *v;
    Edge *next;

    Edge() {}
    Edge(Node *u, Node *v) : u(u), v(v), next(u->e) {}
} _pool[MAXM], *_curr = _pool;

void addEdge(int u, int v) {
    N[u].e = new (_curr++) Edge(&N[u], &N[v]);
    ++N[u].odeg;
    ++N[v].ideg;
}

namespace Hierholzer {
    std::stack<Node *> currPath;
    std::vector<Node *> euler;

    void hierholzer(int s) {
        currPath.push(&N[s]); // delete this line if a Eulerian Path is required
        Node *u = &N[s];
        do {
            if (u->odeg == 0) {
                euler.push_back(currPath.top());
                currPath.pop();
                if (!currPath.empty()) u = currPath.top();
            } else {
                Node *v = u->curr->v;
                u->curr = u->curr->next;
                --u->odeg;
                u = v;
                currPath.push(u);
            }
        } while (!currPath.empty());
    }

    bool solve(int n) {
        int cntS = 0, cntT = 0;
        for (int i = 1; i <= n; i++) {
            if (N[i].ideg + 1 == N[i].odeg) ++cntS;
            else if (N[i].odeg + 1 == N[i].ideg) ++cntT;
            else if (N[i].ideg != N[i].odeg) return false;
        }
        if (cntS > 1 || cntT > 1) return false;
        int s = -1;
        for (int i = 1; i <= n; i++) {
            if (N[i].ideg + 1 == N[i].odeg) s = i;
            N[i].curr = N[i].e;
        }
        s = (s == -1 ? 1 : s);
        hierholzer(s);
        return true;
```

```
    }
}

int main() {
    int n, m;
    scanf("%d %d", &n, &m);
    for (int i = 0, u, v; i < m; i++) {
        scanf("%d %d", &u, &v);
        addEdge(u, v);
    }

    if (Hierholzer::solve(n)) {
        for (int i = Hierholzer::euler.size() - 1; i >= 0; i--) {
            printf("%d", Hierholzer::euler[i] - N);
            if (i) printf(" -> ");
        }
        puts("");
    } else {
        puts("-1");
    }

    return 0;
}
```

3.14.3 无递归版本 无向图

```
#include <cstdio>
#include <list>
#include <vector>
#include <stack>
#include <algorithm>

const int MAXN = 100005;
const int MAXM = 400005;

struct Edge;
struct Node {
    std::list<Edge> e;
    int deg;
} N[MAXN];

struct Edge {
    Node *u, *v;
    std::list<Edge>::iterator rev;

    Edge() {}
    Edge(Node *u, Node *v) : u(u), v(v) {}
};

void addEdge(int u, int v) {
    N[u].e.emplace_front(&N[u], &N[v]);
    N[v].e.emplace_front(&N[v], &N[u]);
    N[u].e.front().rev = N[v].e.begin();
    N[v].e.front().rev = N[u].e.begin();
    ++N[u].deg;
```

```
    ++N[v].deg;
}

namespace Hierholzer {
    std::stack<Node *> currPath;
    std::vector<Node *> euler;

    void hierholzer(Node *s) {
        currPath.push(s); // delete this line if a Eulerian Path is required
        Node *u = s;
        do {
            if (u->deg == 0) {
                euler.push_back(currPath.top());
                currPath.pop();
                if (!currPath.empty()) u = currPath.top();
            } else {
                Node *v = u->e.front().v;
                v->e.erase(u->e.front().rev);
                u->e.erase(u->e.begin());
                --u->deg;
                --v->deg;
                u = v;
                currPath.push(u);
            }
        } while (!currPath.empty());
    }

    bool solve(int n) {
        int cnt = 0;
        for (int i = 1; i <= n; i++) cnt += N[i].deg % 2;
        if (cnt > 2) return false;
        int s = -1;
        if (cnt) for (int i = 1; i <= n; i++) if (N[i].deg % 2) {
            s = i;
            break;
        }
        s = (s == -1 ? 1 : s);
        hierholzer(&N[s]);
        return true;
    }
}

int main() {
    int n, m;
    scanf("%d %d", &n, &m);
    for (int i = 0, u, v; i < m; i++) {
        scanf("%d %d", &u, &v);
        addEdge(u, v);
    }

    if (Hierholzer::solve(n)) {
        for (int i = Hierholzer::euler.size() - 1; i >= 0; i--) {
            printf("%d", Hierholzer::euler[i] - N);
            if (i) printf(" -> ");
        }
        puts("");
```

```
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      }
```

```

    return ans < 0 ? 0 : ans;
}

} // namespace MaxClique

int main() {
    int n, m;
    scanf("%d %d", &n, &m);

    for (int i = 1; i <= n; i++) std::fill(G[i] + 1, G[i] + n + 1, true);
    for (int i = 0, u, v; i < m; i++) {
        scanf("%d %d", &u, &v);
        G[u][v] = G[v][u] = false;
    }

    int ans = MaxClique::solve(n);
    printf("%d\n", ans);

    return 0;
}
```

3.16 最小树形图

3.16.1 朱刘算法

```
#include <cstdio>
#include <climits>
#include <algorithm>

const int MAXN = 3005;

namespace ChuLiu {
    int G[MAXN][MAXN], n, eg[MAXN], queue[MAXN];
    bool used[MAXN], pass[MAXN], more;

    void combine(int id, long long &res) {
        int tot = 0, from;
        for (; id != 0 && !pass[id]; id = eg[id]) {
            queue[tot++] = id;
            pass[id] = true;
        }
        for (from = 0; from < tot && queue[from] != id; ++from) {}
        if (from == tot) return;
        more = true;
        for (int i = from; i < tot; i++) {
            res += G[eg[queue[i]]][queue[i]];
            if (i != from) {
                used[queue[i]] = true;
                for (int j = 1; j <= n; j++) if (!used[j])
                    G[id][j] = std::min(G[id][j], G[queue[i]][j]);
            }
        }
        for (int i = 1; i <= n; i++) if (!used[i] && i != id) {
```

```
            for (int j = from; j < tot; j++) {
                int k = queue[j];
                G[i][id] = std::min(G[i][id], G[i][k] - G[eg[k]][k]);
            }
        }
    }

    long long solve(int n, int root) {
        ChuLiu::n = n;
        std::fill(used + 1, used + n + 1, false);
        long long res = 0;
        for (more = true; more; ) {
            more = false;
            std::fill(eg + 1, eg + n + 1, 0);
            for (int i = 1, k; i <= n; i++) if (!used[i] && i != root) {
                k = 0;
                for (int j = 1; j <= n; j++) if (!used[j] && i != j) {
                    if (k == 0 || G[j][i] < G[k][i])
                        k = j;
                }
                eg[i] = k;
            }
            std::fill(pass + 1, pass + n + 1, false);
            for (int i = 1; i <= n; i++) if (!used[i] && !pass[i] && i != root)
                combine(i, res);
        }
        for (int i = 1; i <= n; i++) if (!used[i] && i != root) {
            if (G[eg[i]][i] == INT_MAX) return -1;
            res += G[eg[i]][i];
        }
        return res;
    }
}

int main() {
    int n, m, rt;
    scanf("%d %d %d", &n, &m, &rt);
    for (int i = 1; i <= n; i++)
        std::fill(ChuLiu::G[i] + 1, ChuLiu::G[i] + n + 1, INT_MAX);
    for (int i = 0, u, v, w; i < m; i++) {
        scanf("%d %d %d", &u, &v, &w);
        ChuLiu::G[u][v] = w;
    }

    long long ans = ChuLiu::solve(n, rt);
    printf("%lld\n", ans);

    return 0;
}
```

3.16.2 Tarjan 的优化实现

内存占用较大。

```
#include <cstdio>
#include <vector>
```

```cpp
#include <stack>
#include <queue>
#include <algorithm>

/*
 * It takes O(n^2) time on dense graph
 * To get O(m \log n) time on sparse graph, change 'std::vector<Edge *> in' to __gnu_pbds::priority_queue<Edge *>,
 * and change line 71-72, 113-138
 */

const int MAXN = 2505;

struct Graph {
    struct Edge {
        int u, v, w, ow;
        bool removed;
        std::vector<Edge *> ch;
        Edge *pa;

        Edge() {}
        Edge(int u, int v, int w) : u(u), v(v), w(w), ow(w), pa(NULL), removed(false) {}
    };
    std::vector<Edge> E;

    void addEdge(int u, int v, int w) {
        if (u == v) return;
        E[eid] = Edge(u, v, w);
        G[u].push_back(&E[eid]);
        in[v].push_back(&E[eid]);
        ++eid;
    }

    std::vector<Edge *> G[MAXN], in[MAXN];
    int n, eid;

    void init(int n, int m) {
        this->n = n;
        E.resize(m);
        this->eid = 0;
    }
} G;

struct DSU {
    int f[MAXN];

    void init(int n) {
        for (int i = 0; i <= n; i++) f[i] = i;
    }
    int find(int x) { return x == f[x] ? x : f[x] = find(f[x]); }
    bool test(int x, int y) { return find(x) == find(y); }
    void merge(int x, int y) { f[find(y)] = find(x); }
};

class OptimumBranching {
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++) {
```

```
                    auto i1 = G.in[cycleRepr[i]].begin();
                    auto e1 = G.in[cycleRepr[i]].end();
                    auto i2 = G.in[cycleRepr[i - 1]].begin();
                    auto e2 = G.in[cycleRepr[i - 1]].end();

                    while (i1 != e1 || i2 != e2) {
                        while (i1 != e1 && S.test((*i1)->u, nr)) ++i1;
                        while (i2 != e2 && S.test((*i2)->u, nr)) ++i2;

                        if (i1 == e1 && i2 == e2) break;
                        else if (i1 == e1) nin.push_back(*i2++);
                        else if (i2 == e2) nin.push_back(*i1++);
                        else if ((*i1)->u < (*i2)->u) nin.push_back(*i1++);
                        else if ((*i1)->u > (*i2)->u) nin.push_back(*i2++);
                        else {
                            if ((*i1)->w < (*i2)->w) nin.push_back(*i1);
                            else nin.push_back(*i2);
                            ++i1;
                            ++i2;
                        }
                    }

                    G.in[cycleRepr[i]].swap(nin);
                    nin.clear();
                }
                G.in[nr].swap(G.in[cycleRepr.back()]);
                change[nr] = 0;
            }
        }

        long long res = 0;
        std::stack<Edge *> froots;
        for (Edge *e : F) if (e->pa == NULL) froots.push(e);
        while (!froots.empty()) {
            Edge *e = froots.top();
            froots.pop();
            if (e->removed) continue;
            res += e->ow;
            remove(lambda[e->v], froots);
        }

        return res;
    }

private:
    using Edge = Graph::Edge;
    DSU S, W;
    std::vector<Edge *> cycle[MAXN];
    Edge *lambda[MAXN], *enter[MAXN];
    int change[MAXN];

    void remove(Edge *e, std::stack<Edge *> &roots) {
        for (; e; e = e->pa) {
            e->removed = true;
            for (Edge *c : e->ch) {
                roots.push(c);
```

```
                c->pa = NULL;
            }
            e->ch.clear();
        }
    }
} ob;

bool check(int rt, int n) {
    static int vis[MAXN];

    std::queue<int> q;
    q.push(rt);
    vis[rt] = true;
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        for (Graph::Edge *e : G.G[u]) if (!vis[e->v]) {
            vis[e->v] = true;
            q.push(e->v);
        }
    }
    for (int i = 1; i <= n; i++) if (!vis[i]) return false;
    return true;
}

int main() {
    int n, m, rt;
    scanf("%d %d %d", &n, &m, &rt);

    G.init(n, m);
    for (int i = 0, u, v, w; i < m; i++) {
        scanf("%d %d %d", &u, &v, &w);
        G.addEdge(u, v, w);
    }

    if (!check(rt, n)) return puts("-1"), 0;

    long long ans = ob.solve(G, rt);
    printf("%lld\n", ans);

    return 0;
}
```

3.17 虚树

```
#include <cstdio>
#include <algorithm>

const int MAXN = 300005;
const int MAXN_LOG = 20;

template <typename T>
struct Edge {
    T *u, *v;
```

```
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
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]];
```

```
    build(flag, tot, tot);

    // do something...
}

int main() {
    int n;
    scanf("%d", &n);
    for (int i = 1, u, v; i < n; i++) {
        scanf("%d %d", &u, &v);
        addEdge(u, v);
    }

    dfs(&N[1]);

    V[0].r = &N[1];
    int q;
    scanf("%d", &q);
    while (q--) solve();

    return 0;
}
```

3.18 二分图最大匹配

```
#include <cstdio>
#include <vector>
#include <queue>
#include <algorithm>

const int MAXN = 1005; // one side
const int MAXM = 1005; // the other side

namespace HopcroftKarp {
    std::vector<int> G[MAXN];
    int matx[MAXN], maty[MAXM], dx[MAXN], dy[MAXM];
    bool vis[MAXM];

    bool find(int u) {
        for (int v : G[u]) if (!vis[v] && dy[v] == dx[u] + 1) {
            vis[v] = true;
            if (!maty[v] || find(maty[v])) {
                matx[u] = v;
                maty[v] = u;
                return true;
            }
        }
        return false;
    }

    int solve(int n, int m) {
        std::fill(matx + 1, matx + n + 1, 0);
        std::fill(maty + 1, maty + m + 1, 0);
        int res = 0;
```

```
        while (true) {
            static std::queue<int> q;
            while (!q.empty()) q.pop();
            bool flag = false;
            std::fill(dx + 1, dx + n + 1, 0);
            std::fill(dy + 1, dy + m + 1, 0);
            for (int i = 1; i <= n; i++) if (!matx[i]) q.push(i);
            while (!q.empty()) {
                int u = q.front();
                q.pop();
                for (int v : G[u]) if (!dy[v]) {
                    dy[v] = dx[u] + 1;
                    if (maty[v]) {
                        dx[maty[v]] = dy[v] + 1;
                        q.push(maty[v]);
                    } else {
                        flag = true;
                    }
                }
            }
            if (!flag) break;
            std::fill(vis + 1, vis + m + 1, false);
            for (int i = 1; i <= n; i++) if (!matx[i] && find(i)) ++res;
        }
        return res;
    }
} // namespace HopcroftKarp

int main() {
    int n1, n2, m;
    scanf("%d %d %d", &n1, &n2, &m);
    for (int i = 0, u, v; i < m; i++) {
        scanf("%d %d", &u, &v);
        HopcroftKarp::G[u].push_back(v);
    }
    printf("%d\n", HopcroftKarp::solve(n1, n2));

    return 0;
}
```

3.19 二分图最大权完备匹配

```
#include <cstdio>
#include <climits>
#include <algorithm>

const int MAXN = 505;

namespace KuhnMunkres {
    int n, G[MAXN][MAXN], x[MAXN], y[MAXN], prevx[MAXN], prevy[MAXN], mat[MAXN], slack[MAXN], par[MAXN];

    void adjust(int u) {
        mat[u] = prevy[u];
        if (prevy[mat[u]] != -2) adjust(prevx[mat[u]]);
```

```
    }

    bool find(int u) {
        for (int i = 1; i <= n; i++) {
            if (prevy[i] == -1) {
                if (slack[i] > x[u] + y[i] - G[u][i]) {
                    slack[i] = x[u] + y[i] - G[u][i];
                    par[i] = u;
                }
                if (x[u] + y[i] == G[u][i]) {
                    prevy[i] = u;
                    if (mat[i] == -1) {
                        adjust(i);
                        return true;
                    }
                    if (prevx[mat[i]] != -1) continue;
                    prevx[mat[i]] = i;
                    if (find(mat[i])) return true;
                }
            }
        }
        return false;
    }

    int solve(int n) {
        KuhnMunkres::n = n;
        std::fill(mat + 1, mat + n + 1, -1);
        std::fill(y + 1, y + n + 1, -1);
        for (int i = 1; i <= n; i++) {
            x[i] = 0;
            for (int j = 1; j <= n; j++) x[i] = std::max(x[i], G[i][j]);
        }
        bool flag = false;
        for (int i = 1; i <= n; i++) {
            std::fill(prevx + 1, prevx + n + 1, -1);
            std::fill(prevy + 1, prevy + n + 1, -1);
            std::fill(slack + 1, slack + n + 1, INT_MAX);
            prevx[i] = -2;
            if (find(i)) continue;
            flag = false;
            while (!flag) {
                int m = INT_MAX;
                for (int j = 1; j <= n; j++) if (prevy[j] == -1) m = std::min(m, slack[j]);
                for (int j = 1; j <= n; j++) {
                    if (prevx[j] != -1) x[j] -= m;
                    if (prevy[j] != -1) y[j] += m;
                    else slack[j] -= m;
                }
                for (int j = 1; j <= n; j++) {
                    if (prevy[j] == -1 && slack[j]) {
                        prevy[j] = par[j];
                        if (mat[j] == -1) {
                            adjust(j);
                            flag = true;
                            break;
                        }
```

```
                        prevx[mat[j]] = j;
                        if (find(mat[j])) {
                            flag = true;
                            break;
                        }
                    }
                }
            }
        }
        int res = 0;
        for (int i = 1; i <= n; i++) res += G[mat[i]][i];
        return res;
    }
} // namespace KuhnMunkres

int main() {
    int n1, n2, m;
    scanf("%d %d %d", &n1, &n2, &m);
    for (int i = 0, u, v, w; i < m; i++) {
        scanf("%d %d %d", &u, &v, &w);
        KuhnMunkres::G[u][v] = w;
    }
    printf("%d\n", KuhnMunkres::solve(std::max(n1, n2)));

    return 0;
}
```

3.20 一般图最大匹配

```
#include <bits/stdc++.h>

const int MAXN = 505;

struct Graph {
    std::vector<int> G[MAXN];
    int n, mate[MAXN];

    void addEdge(int u, int v) {
        G[u].push_back(v);
        G[v].push_back(u);
    }
} G;

class Blossom {
  public:
    int solve(Graph &G) {
        this->G = &G;
        for (int i = 1; i <= G.n; i++) G.mate[i] = -1;
        for (int i = 1; i <= G.n; i++) if (G.mate[i] == -1) aug(i);

        int res = 0;
        for (int i = 1; i <= G.n; i++) res += (G.mate[i] > i);
        return res;
    }
```

```
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
106 int 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

```cpp
#include <cstdio>
#include <cstring>

const int MAXN = 1000005;

int fail[MAXN];
void calcFail(char *s) {
    int n = strlen(s + 1);

    fail[1] = 0;
    for (int i = 2; i <= n; i++) {
        int j = fail[i - 1];
        while (j && s[j + 1] != s[i]) j = fail[j];
        fail[i] = s[i] == s[j + 1] ? j + 1 : 0;
    }
}

int kmp(char *s, char *t) {
    int res = 0;
    int n = strlen(s + 1), m = strlen(t + 1);

    for (int i = 1, j = 0; i <= n; i++) {
        while (j && t[j + 1] != s[i]) j = fail[j];
        if (t[j + 1] == s[i]) j++;
        if (j == m) {
            res++;
            j = fail[j]; // j = 0 when not allowed overlapping
        }
    }

    return res;
}

int main() {
    static char a[MAXN], b[MAXN];
    scanf("%s %s", a + 1, b + 1);

    calcFail(b);

    printf("%d\n", kmp(a, b));

    return 0;
}
```

4.3 扩展 KMP

```cpp
#include <cstdio>
#include <cstring>

const int MAXN = 1000005;

namespace ExKmp {
    int next[MAXN], extend[MAXN];

    void getNext(char *str, int n) {
        int i = 0;
        while (str[i] == str[i + 1] && i + 1 < n) i++;
        next[0] = n;
        next[1] = i;
        int pos = 1;
        for (i = 2; i < n; i++) {
            if (next[i - pos] + i < next[pos] + pos) next[i] = next[i - pos];
            else {
                int j = next[pos] + pos - i;
                if (j < 0) j = 0;
                while (i + j < n && str[j] == str[j + i]) ++j;
                next[i] = j;
                pos = i;
            }
        }
    }

    void getExtand(char *s, char *t) {
        int n = strlen(s), m = strlen(t);
        getNext(t, m);
        int i = 0;
        while (s[i] == t[i] && i < m && i < n) i++;
        extend[0] = i;
        int pos = 0;
        for (i = 1; i < n; i++) {
            if (next[i - pos] + i < extend[pos] + pos) extend[i] = next[i - pos];
            else {
                int j = extend[pos] + pos - i;
                if (j < 0) j = 0;
                while (i + j < n && j < m && s[j + i] == t[j]) ++j;
                extend[i] = j;
                pos = i;
            }
        }
    }
}

char s[MAXN], t[MAXN];

int main() {
    scanf("%s %s", s, t);
    ExKmp::getExtand(s, t);

    return 0;
}
```

4.4 后缀数组

```cpp
#include <cstdio>
#include <cstring>
#include <algorithm>

const int MAXN = 1000005;

namespace SuffixArray {
    int n, sa[MAXN], rank[MAXN], height[MAXN];
    char str[MAXN];

    void buildSA(int m) {
        static int fir[MAXN], sec[MAXN], buc[MAXN], temp[MAXN];
        n = strlen(str);
        str[n++] = 0;

        std::fill(buc, buc + m, 0);
        for (int i = 0; i < n; i++) buc[(int) str[i]]++;
        for (int i = 1; i < m; i++) buc[i] += buc[i - 1];
        for (int i = 0; i < n; i++) rank[i] = buc[(int) str[i]] - 1;

        for (int l = 1; l < n; l <<= 1) {
            for (int i = 0; i < n; i++)
                fir[i] = rank[i], sec[i] = i + l < n ? rank[i + l] : 0;

            std::fill(buc, buc + n, 0);
            for (int i = 0; i < n; i++) buc[sec[i]]++;
            for (int i = 1; i < n; i++) buc[i] += buc[i - 1];
            for (int i = n - 1; ~i; i--) temp[--buc[sec[i]]] = i;

            std::fill(buc, buc + n, 0);
            for (int i = 0; i < n; i++) buc[fir[i]]++;
            for (int i = 1; i < n; i++) buc[i] += buc[i - 1];
            for (int i = n - 1; ~i; i--) sa[--buc[fir[temp[i]]]] = temp[i];

            rank[sa[0]] = 0;
            bool unique = true;
            for (int i = 1; i < n; i++) {
                rank[sa[i]] = rank[sa[i - 1]];
                if (fir[sa[i]] == fir[sa[i - 1]] && sec[sa[i]] == sec[sa[i - 1]])
                    unique = false;
                else rank[sa[i]]++;
            }

            if (unique) break;
        }
    }

    void getHeight() {
        int k = 0;
        for (int i = 0; i < n - 1; i++) {
            k ? k-- : 0;
            int j = sa[rank[i] - 1];
            while (str[i + k] == str[j + k]) k++;
            height[rank[i]] = k;
```

```
        }
    }
}

int main() {
    char *str = SuffixArray::str;
    scanf("%s", str);

    SuffixArray::buildSA(128);
    int *sa = SuffixArray::sa, n = SuffixArray::n;
    for (int i = 1; i < n; i++) printf("%d%c", sa[i] + 1, " \n"[i == n - 1]);

    SuffixArray::getHeight();
    int *height = SuffixArray::height + 1;
    for (int i = 1; i < n - 1; i++) printf("%d%c", height[i], " \n"[i == n - 2]);

    return 0;
}
```

4.5 后缀自动机

```
#include <cstdio>
#include <vector>
#include <algorithm>

const int MAXN = 100005;
const int CHAR_SET = 26;

struct SAM {
    struct Node {
        Node *c[CHAR_SET], *next;
        int max, posCnt;

        Node(int max = 0, bool newSuffix = false) : max(max), posCnt(newSuffix), next(NULL), c() {}

        int min() const {
            return next->max + 1;
        }
    } *start, *last, _pool[MAXN << 1], *_curr;

    SAM() {
        init();
    }

    void init() {
        _curr = _pool;
        start = last = new (_curr++) Node;
    }

    Node *extend(int c) {
        Node *u = new (_curr++) Node(last->max + 1, true), *v = last;

        for (; v && !v->c[c]; v = v->next) v->c[c] = u;

```

```
        if (!v) {
            u->next = start;
        } else if (v->c[c]->max == v->max + 1) {
            u->next = v->c[c];
        } else {
            Node *n = new (_curr++) Node(v->max + 1), *o = v->c[c];
            std::copy(o->c, o->c + CHAR_SET, n->c);
            n->next = o->next;
            u->next = o->next = n;
            for (; v && v->c[c] == o; v = v->next) v->c[c] = n;
        }

        return last = u;
    }

    std::vector<Node *> topo;
    std::vector<Node *> toposort() {
        static int buc[MAXN << 1];
        int max = 0;
        for (Node *p = _pool; p != _curr; p++) {
            max = std::max(max, p->max);
            buc[p->max]++;
        }
        for (int i = 1; i <= max; i++) buc[i] += buc[i - 1];

        topo.resize(_curr - _pool);
        for (Node *p = _pool; p != _curr; p++) topo[--buc[p->max]] = p;

        return topo;
    }

    void calc() {
        toposort();

        for (int i = topo.size() - 1; i; i--) topo[i]->next->posCnt += topo[i]->posCnt;
    }
} sam;

int main() {

    return 0;
}
```

4.6 AC 自动机

```
#include <cstdio>
#include <queue>
#include <algorithm>

const int MAXN = 100005;
const int CHAR_SET = 26;

struct ACAM {
    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++) {
```

```
            u = u->c[*p - 'a'];
            if (u->isWord) u->apply();
            else if (u->next) u->next->apply();
        }
    }
} acam;

int main() {

    return 0;
}
```

4.7 Manacher

```
#include <cstdio>
#include <cstring>
#include <algorithm>

const int MAXN = 1000005;

namespace Manacher {
    char s[MAXN << 1];
    int r[MAXN << 1], len;

    void prepare(char *str) {
        len = 0;
        s[++len] = '@';
        s[++len] = '#';
        int n = strlen(str);
        for (int i = 0; i < n; i++) s[++len] = str[i], s[++len] = '#';
        s[++len] = '\0';
    }

    void manacher() {
        int right = 0, pos = 0;
        for (int i = 1; i <= len; i++) {
            int x = right < i ? 1 : std::min(r[2 * pos - i], right - i);
            while (s[i + x] == s[i - x]) ++x;
            r[i] = x;
            if (x + i > right) {
                right = x + i;
                pos = i;
            }
        }
    }

    void calc(char *str) {
        prepare(str);
        manacher();
    }
}

int main() {
    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

```cpp
#include <cstdio>

int gcd(int a, int b) {
    return b ? gcd(b, a % b) : a;
}

void exgcd(int a, int b, int &g, int &x, int &y) {
    if (b == 0) x = 1, y = 0, g = a;
    else exgcd(b, a % b, g, y, x), y -= a / b * x;
}

int main() {

    return 0;
}
```

5.2 组合数与 Lucas 定理

```cpp
#include <cstdio>

const int MOD = 1000003;

int fact[MOD], inv[MOD];

long long pow(long long a, long long n) {
    long long res = 1;
    for (; n; n >>= 1, a = a * a % MOD) if (n & 1) res = res * a % MOD;
    return res;
}

void prepare() {
    fact[0] = 1;
    for (int i = 1; i < MOD; i++) fact[i] = (long long) fact[i - 1] * i % MOD;

    inv[MOD - 1] = pow(fact[MOD - 1], MOD - 2);
    for (int i = MOD - 2; i; i--) inv[i] = (long long) inv[i + 1] * (i + 1) % MOD;
}

int combi(int n, int m) {
    if (m > n) return 0;
    if (n < MOD) return (long long) fact[n] * inv[m] % MOD * inv[n - m] % MOD;
    return (long long) combi(n / MOD, m / MOD) * combi(n % MOD, m % MOD) % MOD;
}

int main() {
    prepare();

```

```
    return 0;
}
```

5.3 线性预处理逆元

```
#include <cstdio>

const int MAXN = 3000005;

int fact[MAXN], invFact[MAXN], inv[MAXN];

void exgcd(int a, int b, int &x, int &y) {
    if (b == 0) x = 1, y = 0;
    else exgcd(b, a % b, y, x), y -= x * (a / b);
}

int getInv(int x, int mod) {
    int res, temp;
    exgcd(x, mod, res, temp);
    return res;
}

int main() {
    int n, p;
    scanf("%d %d", &n, &p);

    inv[1] = 1;
    for (int i = 2; i <= n; i++) inv[i] = (long long) (p - p / i) * inv[p % i] % p;

    fact[0] = 1;
    for (int i = 1; i <= n; i++) fact[i] = (long long) fact[i - 1] * i % p;
    invFact[n] = getInv(fact[n], p);
    for (int i = n - 1; i; i--) invFact[i] = (long long) invFact[i + 1] * (i + 1)% p;

    return 0;
}
```

5.4 线性筛

```
#include <cstdio>

const int MAXN = 10000005;

int prime[MAXN], primeCnt, mu[MAXN], phi[MAXN], d[MAXN];
long long s[MAXN];
bool notPrime[MAXN];

void sieve() {
    static int minFact[MAXN], minPow[MAXN];

    notPrime[0] = notPrime[1] = true;
    mu[1] = phi[1] = d[1] = s[1] = 1;
```

```
    minFact[1] = minPow[1] = 1;

    for (int i = 2; i < MAXN; i++) {
        if (!notPrime[i]) {
            prime[primeCnt++] = i;
            mu[i] = -1;
            phi[i] = i - 1;
            d[i] = 2;
            s[i] = i + 1;
            minFact[i] = i;
            minPow[i] = 1;
        }

        for (int j = 0; j < primeCnt && i * prime[j] < MAXN; j++) {
            notPrime[i * prime[j]] = true;
            if (i % prime[j] == 0) {
                mu[i * prime[j]] = 0;
                phi[i * prime[j]] = phi[i] * prime[j];
                d[i * prime[j]] = d[i] / (minPow[i] + 1) * (minPow[i] + 2);
                if (i == minFact[i]) {
                    s[i * prime[j]] = s[i] + i * prime[j];
                } else {
                    s[i * prime[j]] = s[i / minFact[i]] * s[prime[j] * minFact[i]];
                }
                minPow[i * prime[j]] = minPow[i] + 1;
                minFact[i * prime[j]] = minFact[i] * prime[j];
                break;
            }
            mu[i * prime[j]] = -mu[i];
            phi[i * prime[j]] = phi[i] * (prime[j] - 1);
            d[i * prime[j]] = d[i] * 2;
            s[i * prime[j]] = s[i] * s[prime[j]];
            minPow[i * prime[j]] = 1;
            minFact[i * prime[j]] = prime[j];
        }
    }
}

int main() {
    sieve();

    return 0;
}
```

5.5 杜教筛

```
#include <cstdio>
#include <map>

const int MAXNN = 1600000;

long long phi[MAXNN];
int prime[MAXNN], primeCnt;
bool notPrime[MAXNN];
```

```

void sieve() {
    phi[1] = 1;
    notPrime[0] = notPrime[1] = true;
    for (int i = 2; i < MAXNN; i++) {
        if (!notPrime[i]) {
            prime[++primeCnt] = i;
            phi[i] = i - 1;
        }

        for (int j = 1; j <= primeCnt && i * prime[j] < MAXNN; j++) {
            notPrime[i * prime[j]] = true;
            if (i % prime[j] == 0) {
                phi[i * prime[j]] = phi[i] * prime[j];
                break;
            }
            phi[i * prime[j]] = phi[i] * (prime[j] - 1);
        }
    }

    for (int i = 2; i < MAXNN; i++) phi[i] += phi[i - 1];
}

/*
 *  h = f * g
 *  g(1) F(n) = H(n) - \sum_{i = 2}^{n} g(i) F(\lfloor \frac{n}{i} \rfloor)
 */

int n;
long long phiH[MAXNN];
long long pSumPhi(int n) {
    if (n < MAXNN) return phi[n];

    int id = ::n / n;

    if (phiH[id] != -1) return phiH[id];

    long long res = (long long) n * (n + 1) / 2;
    for (int i = 2, last; i <= n; i = last + 1) {
        last = n / (n / i);
        res -= (last - i + 1) * pSumPhi(n / i);
    }
    return phiH[id] = res;
}

int main() {
    sieve();

    scanf("%d", &n);
    std::fill(phiH, phiH + n / MAXNN + 1, -1);
    printf("%lld\n", pSumPhi(n));

    return 0;
}
```

5.6 Min25 筛

```cpp
#include <cstdio>
#include <cmath>

const int MAXN = 100005; // sqrt n
const int MOD = 1000000007;
int INV2, INV6;

long long qpow(long long a, long long n) {
    long long res = 1;
    for (; n; n >>= 1, a = a * a % MOD) if (n & 1) res = res * a % MOD;
    return res;
}

int prime[MAXN], primeCnt, lpf[MAXN];
long long sprime[MAXN], ssprime[MAXN];
bool notPrime[MAXN];

void sieve() {
    for (int i = 2; i < MAXN; i++) {
        if (!notPrime[i]) {
            prime[++primeCnt] = i; // index from 1
            sprime[primeCnt] = (sprime[primeCnt - 1] + i) % MOD;
            ssprime[primeCnt] = (ssprime[primeCnt - 1] + 1ll * i * i) % MOD;
            lpf[i] = primeCnt;
        }
        for (int j = 1; j <= primeCnt && i * prime[j] < MAXN; j++) {
            notPrime[i * prime[j]] = true;
            if (i % prime[j] == 0) break;
            lpf[i * prime[j]] = j;
        }
    }
}

class Min25 {
private:
    long long G[MAXN][3], Fprime[MAXN];
    int list[MAXN], cnt;

    int le[MAXN], ge[MAXN], lim, n;
    inline int &id(int v) { return v <= lim ? le[v] : ge[n / v]; }

    void init(int n) {
        cnt = 0;
        for (int i = 1, j, v; i <= n; i = n / j + 1) {
            j = n / i;
            v = j % MOD;
            list[++cnt] = j;
            id(j) = cnt;
            // sum from 2 to j (assuming all the values are calculated as if they are prime)
            // i^0
            G[cnt][0] = v ? v - 1 : MOD - 1;
            // i^1
            G[cnt][1] = (2ll + v) * (v - 1ll) % MOD * INV2 % MOD;
            // 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) %
                        MOD;
69              }
70          }
71          for (int i = 1; i <= cnt; i++) {
72              // prefix sum of values at primes
73              Fprime[i] = (G[i][1] - G[i][0] + MOD) % MOD;
74          }
75      }
76
77      int fp(int p, int c) {
78          // value at power of prime
79          return qpow(p, c - 1) * (p - 1) % MOD;
80      }
81
82      long long F(int K, int n) {
83          if (n < prime[K] || n <= 1) return 0;
84          int id = this->id(n);
85          long long ans = Fprime[id] - (sprime[K - 1] - (K - 1));
86          ans = (ans % MOD + MOD) % MOD;
87          for (int i = K; i <= primeCnt && 1ll * prime[i] * prime[i] <= n; i++) {
88              long long pw = prime[i], pw2 = pw * pw;
89              for (int c = 1; pw2 <= n; pw = pw2, pw2 *= prime[i], c++) {
90                  ans = (ans + fp(prime[i], c) * F(i + 1, n / pw) + fp(prime[i], c + 1)) % MOD;
91              }
92          }
93          return ans;
94      }
95
96  public:
97      long long solve(int n) {
98          this->n = n;
99          lim = sqrt(n);
100         init(n);
101         calcFprime();
102         long long res = F(1, n) + 1; // remember to add f(1)
103         return res >= MOD ? res -= MOD : res;
104     }
105 } min25;
106
107 int main() {
108     sieve();
109     INV2 = qpow(2, MOD - 2);
```

```
    INV6 = qpow(6, MOD - 2);

    int n;
    scanf("%d", &n);
    long long ans = min25.solve(n);
    printf("%lld\n", ans);

    return 0;
}
```

5.7 线性同余方程

```
#include <cstdio>

const int MAXN = 100005;

void exgcd(long long a, long long b, long long &g, long long &x, long long &y) {
    if (b == 0) x = 1, y = 0, g = a;
    else exgcd(b, a % b, g, y, x), y -= a / b * x;
}

int mod[MAXN], rem[MAXN];
long long solveCongruence(int n) {
    long long res = 0, K = 1;
    for (int i = 0; i < n; i++) {
        long long x, y, g;
        exgcd(K, mod[i], g, x, y);
        if ((rem[i] - res) % g) return -1;
        x = (x * (rem[i] - res) / g + mod[i] / g) % (mod[i] / g);
        y = (K / g *mod[i]);
        res = ((x * K + res) % y + y) % y;
        K = y;
    }
    return res;
}

int main() {
    int n;
    scanf("%d", &n);
    for (int i = 0; i < n; i++) scanf("%d %d", &mod[i], &rem[i]);
    long long ans = solveCongruence(n);
    printf("%lld\n", ans);

    return 0;
}
```

5.8 BSGS

```
#include <cstdio>
#include <cmath>
#include <map>

```

```cpp
long long pow(long long a, long long n, long long p) {
    long long res = 1;
    for (; n; n >>= 1, a = a * a % p) if (n & 1) res = res * a % p;
    return res;
}

long long inv(long long a, long long p) {
    return pow(a, p - 2, p);
}

long long bsgs(long long a, long long b, long long p) {
    a %= p, b %= p;
    if (a == 0) return b == 0 ? 1 : -1;

    std::map<long long, long long> map;

    long long m = std::ceil(std::sqrt(p)), t = 1;
    for (int i = 0; i < m; i++) {
        if (!map.count(t)) map[t] = i;
        t = t * a % p;
    }

    long long k = inv(t, p), w = b;
    for (int i = 0; i < m; i++) {
        if (map.count(w)) return i * m + map[w];
        w = w * k % p;
    }

    return -1;
}

int main () {
    long long a, b, p;
    scanf("%lld %lld %lld", &a, &b, &p);

    long long ans = bsgs(a, b, p);
    printf("%lld\n", ans);

    return 0;
}
```

5.9 二次剩余

```cpp
#include <cstdio>
#include <tuple>

long long qpow(long long a, long long n, long long p) {
    long long res = 1;
    for (; n; n >>= 1, a = a * a % p) if (n & 1) res = res * a % p;
    return res;
}

int modSqrt(int a, int p) {
    if (p == 2) return a % p;
```

```
    int x;
    if (qpow(a, (p - 1) >> 1, p) == 1) {
        if (p % 4 == 3) {
            x = qpow(a, (p + 1) >> 2, p);
        } else {
            long long w;
            for (w = 1; qpow((w * w - a + p) % p, (p - 1) >> 1, p) == 1; w++) {}
            long long b0 = w, b1 = 1;
            w = (w * w - a + p) % p;
            long long r0 = 1, r1 = 0;
            int exp = (p + 1) >> 1;
            for (; exp; exp >>= 1) {
                if (exp & 1)
                    std::tie(r0, r1) = std::make_tuple((r0 * b0 + r1 * b1 % p * w) % p, (r0 * b1 + r1 * b0)
                        % p);
                std::tie(b0, b1) = std::make_tuple((b0 * b0 + b1 * b1 % p * w) % p, 2 * b0 * b1 % p);
            }
            x = r0;
        }
        if (x * 2 > p) x = p - x;
        return x;
    }
    return -1;
}

int main() {
    int n, p;
    scanf("%d %d", &n, &p);
    int ans = modSqrt(n, p);
    printf("%d\n", ans);

    return 0;
}
```

5.10 线性基

5.10.1 在线版本

```
#include <cstdio>

const int MAXL = 50;

struct LinearBasis {
    long long a[MAXL + 1];

    bool extend(long long t) {
        for (int i = MAXL; ~i; i--) {
            if (!(t & (1ll << i))) continue;
            if (!t) return false;

            if (a[i]) t ^= a[i];
            else {
                for (int j = 0; j < i; j++) if (t & (1ll << j)) t ^= a[j];
                for (int j = i + 1; j <= MAXL; j++) if (a[j] & (1ll << i)) a[j] ^= t;
```

```
                a[i] = t;
                return true;
            }
        }

        return false;
    }

    long long queryMax() {
        long long res = 0;
        for (int i = 0; i <= MAXL; i++) res ^= a[i];
        return res;
    }
} lb;

int main() {
    int n;
    scanf("%d", &n);

    for (int i = 0; i < n; i++) {
        long long x;
        scanf("%lld", &x);
        lb.extend(x);
    }

    printf("%lld\n", lb.queryMax());

    return 0;
}
```

5.10.2 离线版本

```
#include <cstdio>
#include <vector>
#include <algorithm>

const int MAXN = 100005;
const int MAXL = 50;

struct LinearBasis {
    std::vector<long long> v;
    int n;

    void init(long long *x, int n) {
        this->n = n;
        static long long a[MAXL + 1];

        for (int i = 0; i < n; i++) {
            long long t = x[i];

            for (int j = MAXL; ~j; j--) {
                if (!(t & (1ll << j))) continue;
                if (!t) break;

                if (a[j]) t ^= a[j];
```

```
                else {
                    for (int k = 0; k < j; k++) if (t & (1ll << k)) t ^= a[k];
                    for (int k = j + 1; k <= MAXL; k++) if (a[k] & (1ll << j)) a[k] ^= t;
                    a[j] = t;
                    break;
                }
            }
        }

        v.clear();
        for (int i = 0; i <= MAXL; i++) if (a[i]) v.push_back(a[i]);
    }

    long long query(int k) {
        if (v.size() != n) k--;
        if (k >= (1ll << v.size())) return -1;

        long long res = 0;
        for (int i = 0; i < v.size(); i++) if (k & (1ll << i)) res ^= v[i];
        return res;
    }
} lb;

int main() {
    int n;
    scanf("%d", &n);

    static long long a[MAXN];
    for (int i = 0; i < n; i++) scanf("%lld", &a[i]);
    lb.init(a, n);

    int q;
    scanf("%d", &q);
    while (q--) {
        long long k;
        scanf("%lld", &k);
        printf("%lld\n", lb.query(k));
    }
    return 0;
}
```

5.11 高斯消元

```
#include <cstdio>
#include <cmath>
#include <algorithm>

const int MAXN = 105;
const double EPS = 1e-8;

int dcmp(double a, double b = 0.0) {
    double d = a - b;
    return std::abs(d) <= EPS ? 0 : (d > 0 ? 1 : -1);
}
```

```

namespace GaussJordan {
    double a[MAXN][MAXN];

    bool solve(int n) {
        for (int i = 0; i < n; i++) {
            int max = i;
            for (int j = i + 1; j < n; j++)
                if (dcmp(std::abs(a[j][i]), std::abs(a[max][i])) > 0) max = j;

            if (!dcmp(a[max][i])) return false;
            if (max != i) for (int j = i; j <= n; j++) std::swap(a[i][j], a[max][j]);

            for (int j = 0; j < n; j++) if (i != j) for (int k = n; k >= i; k--)
                a[j][k] -= a[i][k] / a[i][i] * a[j][i];
        }

        return true;
    }
}

int main() {
    int n;
    scanf("%d", &n);

    using GaussJordan::a;

    for (int i = 0; i < n; i++) for (int j = 0; j <= n; j++)
        scanf("%lf", &a[i][j]);

    if (!GaussJordan::solve(n)) puts("-1");
    else for (int i = 0; i < n; i++) printf("%.4lf\n", a[i][n] / a[i][i]);

    return 0;
}
```

5.12 Berlekamp-Massey

```
#include <cstdio>
#include <vector>
#include <algorithm>

const int MAXN = 2505;
const int MOD = 998244353;

long long qpow(long long a, long long n) {
    long long res = 1;
    for (; n; n >>= 1, a = a * a % MOD) if (n & 1) res = res * a % MOD;
    return res;
}

class BerlekampMassey {
public:
    std::vector<int> solve(int *x, int n) {
```

```
        int L = 0, m = 1, b = 1;
        C = B = { 1 };
        for (int i = 0; i < n; i++) {
            int d = x[i];
            for (int j = 1; j <= L; j++) d = (d + 1ll * C[j] * x[i - j]) % MOD;
            if (d == 0) {
                ++m;
            } else if (2 * L <= i) {
                T = C;
                C.resize(i + 2 - L);
                int ib = qpow(b, MOD - 2);
                d = MOD - d;
                for (int i = 0; i < B.size(); i++) C[i + m] = (C[i + m] + 1ll * d * ib % MOD * B[i]) % MOD;
                L = i + 1 - L;
                B.swap(T);
                b = MOD - d;
                m = 1;
            } else {
                int ib = qpow(b, MOD - 2);
                d = MOD - d;
                for (int i = 0; i < B.size(); i++) C[i + m] = (C[i + m] + 1ll * d * ib % MOD * B[i]) % MOD;
                ++m;
            }
        }
        for (int i = 1; i <= L; i++) C[i] = (MOD - C[i]) % MOD;
        return C;
    }

private:
    std::vector<int> C, B, T;
} bm;

int x[MAXN];

int main() {
    int n;
    scanf("%d", &n);
    for (int i = 0; i < n; i++) scanf("%d", &x[i]);

    auto f = bm.solve(x, n);
    for (int i = 1; i < f.size(); i++) { // from 1
        printf("%d%c", f[i], " \n"[i == f.size() - 1]);
    }

    return 0;
}
```

5.13 Miller-Rabin

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

```

bool isPrime(long long n) {
    const static int primes[12] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37};

    long long s = 0, d = n - 1;
    while (d % 2 == 0) d /= 2, s++;
    if (s == 0) return n == 2;

    for (int i = 0; i < 12 && primes[i] < n; i++) {
        long long a = primes[i];
        if (pow(a, d, n) != 1) {
            bool flag = true;
            for (int r = 0; r < s; r++)
                if (flag && pow(a, d * (1 << r), n) == n - 1) flag = false;
            if (flag) return false;
        }
    }

    return true;
}
```

5.14 Pollard's Rho

```
long long gcd(long long a, long long b) {
    return b ? gcd(b, a % b) : a;
}

namespace PollardRho {
    long long g(long long x, long long n, long long c) {
        return (mul(x, x, n) + c) % n;
    }

    long long rho(long long n, long long c) {
        long long x = rand() % n, y = x, d = 1;

        for (long long i = 1, k = 2; d == 1; i++) {
            x = g(x, n, c);
            d = gcd(x > y ? x - y : y - x, n);
            if (x == y) return n;
            if (i == k) k <<= 1, y = x;
        }

        return d;
    }

    void find(long long n, long long c, std::map<long long, int> &res) {
        if (n == 1) return;
        if (isPrime(n)) {
            res[n]++;
            return;
        }

        long long p = n;
        while (p == n) p = rho(p, c++);
```

```
        find(p, c, res);
        find(n / p, c, res);
    }

    std::map<long long, int> divide(long long n) {
        static std::map<long long, int> res;
        res.clear();
        find(n, 1, res);
        return res;
    }
}
```

5.15 快速傅立叶变换

5.15.1 FFT

```
#include <cstdio>
#include <cmath>
#include <algorithm>

const int MAXN = 262144 + 1;
const double PI = std::acos(-1.0);

struct Complex {
    double r, i;

    Complex(double r = 0, double i = 0) : r(r), i(i) {}

    Complex conj() const { return Complex(r, -i); }

    Complex operator+(const Complex &rhs) const { return Complex(r + rhs.r, i + rhs.i); }
    Complex operator-(const Complex &rhs) const { return Complex(r - rhs.r, i - rhs.i); }
    Complex operator*(const Complex &rhs) const { return Complex(r * rhs.r - i * rhs.i, r * rhs.i + i * rhs
        .r); }
    Complex operator/(double rhs) const { return Complex(r / rhs, i / rhs); }
};

class FFT {
private:
    static const int N = 262144;

    Complex omega[N + 1], omegaInv[N + 1];

    void init() {
        for (int i = 0; i < N; i++) {
            omega[i] = Complex(std::cos(2 * PI / N * i), std::sin(2 * PI / N * i));
            omegaInv[i] = omega[i].conj();
        }
    }

    void reverse(Complex *a, int n) {
        for (int i = 0, j = 0; i < n; i++) {
            if (i < j) std::swap(a[i], a[j]);
            for (int l = n >> 1; (j ^= l) < l; l >>= 1) {}
```

```
        }
    }

    void transform(Complex *a, int n, Complex *omega) {
        reverse(a, n);

        for (int l = 2; l <= n; l <<= 1) {
            int hl = l >> 1;
            for (Complex *x = a; x != a + n; x += l) {
                for (int i = 0; i < hl; i++) {
                    Complex t = omega[N / l * i] * x[i + hl];
                    x[i + hl] = x[i] - t;
                    x[i] = x[i] + t;
                }
            }
        }
    }

public:
    FFT() { init(); }

    int extend(int n) {
        int res = 1;
        while (res < n) res <<= 1;
        return res;
    }

    void dft(Complex *a, int n) {
        transform(a, n, omega);
    }

    void idft(Complex *a, int n) {
        transform(a, n, omegaInv);
        for (int i = 0; i < n; i++) a[i] = a[i] / n;
    }
} fft;

int main() {
    int n, m;
    scanf("%d %d", &n, &m);

    static Complex a[MAXN], b[MAXN];
    for (int i = 0; i <= n; i++) scanf("%lf", &a[i].r);
    for (int i = 0; i <= m; i++) scanf("%lf", &b[i].r);

    int N = fft.extend(n + m + 1);

    fft.dft(a, N);
    fft.dft(b, N);
    for (int i = 0; i < N; i++) a[i] = a[i] * b[i];
    fft.idft(a, N);

    for (int i = 0; i < n + m + 1; i++)
        printf("%.0lf%c", a[i].r + 0.001, " \n"[i == n + m]);

    return 0;
```

```
}
```

5.15.2 两次 DFT 的多项式乘法

注：精度低于普通 FFT

```
#include <cstdio>
#include <cmath>
#include <algorithm>

const int MAXN = 262144 + 1;
const double PI = std::acos(-1.0);

struct Complex {
    double r, i;

    Complex(double r = 0, double i = 0) : r(r), i(i) {}

    Complex conj() const { return Complex(r, -i); }

    Complex operator+(const Complex &rhs) const { return Complex(r + rhs.r, i + rhs.i); }
    Complex operator-(const Complex &rhs) const { return Complex(r - rhs.r, i - rhs.i); }
    Complex operator*(const Complex &rhs) const { return Complex(r * rhs.r - i * rhs.i, r * rhs.i + i * rhs
        .r); }
    Complex operator/(double rhs) const { return Complex(r / rhs, i / rhs); }
};

class FFT {
private:
    static const int N = 262144;

    Complex omega[N + 1], omegaInv[N + 1];

    void init() {
        for (int i = 0; i < N; i++) {
            omega[i] = Complex(std::cos(2 * PI / N * i), std::sin(2 * PI / N * i));
            omegaInv[i] = omega[i].conj();
        }
    }

    void reverse(Complex *a, int n) {
        for (int i = 0, j = 0; i < n; i++) {
            if (i < j) std::swap(a[i], a[j]);
            for (int l = n >> 1; (j ^= l) < l; l >>= 1) {}
        }
    }

    void transform(Complex *a, int n, Complex *omega) {
        reverse(a, n);

        for (int l = 2; l <= n; l <<= 1) {
            int hl = l >> 1;
            for (Complex *x = a; x != a + n; x += l) {
                for (int i = 0; i < hl; i++) {
                    Complex t = omega[N / l * i] * x[i + hl];
                    x[i + hl] = x[i] - t;
```

```
                    x[i] = x[i] + t;
                }
            }
        }
    }

public:
    FFT() { init(); }

    int extend(int n) {
        int res = 1;
        while (res < n) res <<= 1;
        return res;
    }

    void dft(Complex *a, int n) {
        transform(a, n, omega);
    }

    void idft(Complex *a, int n) {
        transform(a, n, omegaInv);
        for (int i = 0; i < n; i++) a[i] = a[i] / n;
    }
} fft;

int main() {
    int n, m;
    scanf("%d %d", &n, &m);

    static Complex a[MAXN], b[MAXN];

    for (int i = 0, x; i <= n; i++) {
        scanf("%d", &x);
        a[i] = Complex(x, 0);
    }

    for (int i = 0, x; i <= m; i++) {
        scanf("%d", &x);
        a[i] = a[i] + Complex(0, x);
    }

    int len = n + m + 1;
    int N = fft.extend(len);

    fft.dft(a, N);
    for (int i = 1; i < N; i++) {
        double x1 = a[i].r, y1 = a[i].i;
        double x2 = a[N - i].r, y2 = a[N - i].i;
        Complex t1((x1 + x2) * 0.5, (y1 - y2) * 0.5);
        Complex t2((y1 + y2) * 0.5, (x2 - x1) * 0.5);
        b[i] = t1 * t2;
    }
    b[0] = a[0].r * a[0].i;
    fft.idft(b, N);

    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
           .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                      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
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。

```cpp
#include <cstdio>
#include <cmath>
#include <algorithm>

const int MAXN = 262144 + 1;
const double PI = std::acos(-1.0);
const int MOD = 1000000007;

struct Complex {
    double r, i;

    Complex(double r = 0, double i = 0) : r(r), i(i) {}

    Complex conj() const { return Complex(r, -i); }

    Complex operator-(const Complex &rhs) const { return Complex(r - rhs.r, i - rhs.i); }
    Complex operator+(const Complex &rhs) const { return Complex(r + rhs.r, i + rhs.i); }
    Complex operator*(const Complex &rhs) const { return Complex(r * rhs.r - i * rhs.i, r * rhs.i + i * rhs
        .r); }
    Complex operator/(double rhs) const { return Complex(r / rhs, i / rhs); }
};

class FFT {
private:
    static const int N = 262144;

    Complex omega[N + 1], omegaInv[N + 1];

    void init() {
        double per = 2 * PI / N;
        for (int i = 0; i < N; i++) {
            omega[i] = Complex(std::cos(i * per), std::sin(i * per));
            omegaInv[i] = omega[i].conj();
        }
    }

    void reverse(Complex *a, int n) {
        for (int i = 0, j = 0; i < n; i++) {
            if (i < j) std::swap(a[i], a[j]);
            for (int l = n >> 1; (j ^= l) < l; l >>= 1) {}
        }
    }

    void transform(Complex *a, int n, Complex *omega) {
        reverse(a, n);

        for (int l = 2; l <= n; l <<= 1) {
            int hl = l >> 1;
            for (Complex *x = a; x != a + n; x += l) {
                for (int i = 0; i < hl; i++) {
                    Complex t = omega[N / l * i] * x[i + hl];
                    x[i + hl] = x[i] - t;
```

```
                    x[i] = x[i] + t;
                }
            }
        }
    }

public:
    FFT() { init();}

    int extend(int n) {
        int res = 1;
        while (res < n) res <<= 1;
        return res;
    }

    void dft(Complex *a, int n) {
        transform(a, n, omega);
    }

    void idft(Complex *a, int n) {
        transform(a, n, omegaInv);
        for (int i = 0; i < n; i++) a[i] = a[i] / n;
    }
} fft;

void polyMul(long long *a, long long *b, int n, long long *res) {
    static Complex a0[MAXN], a1[MAXN], b0[MAXN], b1[MAXN];
    static const int M = (1 << 15) - 1;

    for (int i = 0; i < n; i++) {
        a0[i] = a[i] >> 15;
        a1[i] = a[i] & M;
        b0[i] = b[i] >> 15;
        b1[i] = b[i] & M;
    }
    fft.dft(a0, n), fft.dft(a1, n);
    fft.dft(b0, n), fft.dft(b1, n);
    for (int i = 0; i < n; i++) {
        Complex _a = a0[i], _b = a1[i], _c = b0[i], _d = b1[i];
        a0[i] = _a * _c;
        a1[i] = _a * _d + _b * _c;
        b0[i] = _b * _d;
    }
    fft.idft(a0, n), fft.idft(a1, n), fft.idft(b0, n);
    for (int i = 0; i < n; i++) {
        res[i] = ((((long long) (a0[i].r + 0.5) % MOD) << 30) % MOD
                + (((long long) (a1[i].r + 0.5) % MOD) << 15) % MOD
                  + (long long) (b0[i].r + 0.5) % MOD) % MOD;
    }
}

void polyInverse(long long *a, long long *res, int k) {
    if (k == 1) {
        res[0] = 1;
        return;
    }
```

```
    polyInverse(a, res, (k + 1) >> 1);

    static long long t1[MAXN], t2[MAXN];
    int N = fft.extend(k << 1);
    std::copy(a, a + k, t1);
    std::fill(t1 + k, t1 + N, 0);
    polyMul(res, res, N, t2);
    polyMul(t1, t2, N, t1);
    for (int i = 0; i < k; i++) res[i] = (2 * res[i] % MOD - t1[i] + MOD) % MOD;
    std::fill(res + k, res + N, 0);
}

long long a[MAXN], res[MAXN];

int main() {
    int n, k;
    scanf("%d %d", &n, &k);
    for (int i = 0; i <= n; i++) scanf("%lld", &a[i]);

    polyInverse(a, res, k);
    for (int i = 0; i < k; i++) printf("%lld%c", res[i], " \n"[i == k - 1]);

    return 0;
}
```

5.17 挑战多项式（多项式逆元、除法、平方根、ln、exp、快速幂）

```
#include <cstdio>
#include <vector>
#include <tuple>
#include <algorithm>

const int MAXN = 262144 + 1;
const int MOD = 998244353;
const int G = 3;

long long qpow(long long a, long long n) {
    long long res = 1;
    for (; n; n >>= 1, a = a * a % MOD) if (n & 1) res = res * a % MOD;
    return res;
}

int numinv[MAXN];
void init() {
    numinv[1] = 1;
    for (int i = 2; i < MAXN; i++) numinv[i] = (long long) (MOD - MOD / i) * numinv[MOD % i] % MOD;
}

long long inv(long long x) {
    return x < MAXN ? numinv[x] : qpow(x, MOD - 2);
}

class NTT {
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
                    ), exp >>= 1) {
100                 if (exp & 1)
101                     std::tie(r0, r1) = std::make_tuple((r0 * b0 + r1 * b1 % p * w) % p, (r0 * b1 + r1 * b0)
                            % p);
102             }
103             x = r0;
104         }
105         if (x * 2 > p) x = p - x;
106         return x;
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;
```

```
}
```

5.18 多项式 GCD

```cpp
#include <vector>

const int MOD = 1000000007;

long long qpow(long long a, long long n) {
    long long res = 1;
    for (; n; n >>= 1, a = a * a % MOD) if (n & 1) res = res * a % MOD;
    return res;
}

// a[i] is the coefficient of x^i
std::vector<int> polyGcd(std::vector<int> a, const std::vector<int> &b) {
    if (b.empty()) return a;
    int t = a.size() - b.size();
    for (int i = 0; i <= t; i++) {
        long long temp = a[a.size() - 1 - i] * qpow(b[b.size() - 1], MOD - 2) % MOD;
        for (int j = 0; j < b.size(); j++)
            a[a.size() - 1 - i - j] = (a[a.size() - 1 - i - j] - temp * b[b.size() - 1 - j] % MOD + MOD) %
                MOD;
    }
    int p = -1;
    for (int i = a.size() - 1; ~i; i--) if (a[i]) {
        p = i;
        break;
    }
    a.resize(p + 1);
    return polyGcd(b, a);
}

int main() {

    return 0;
}
```

5.19 原根

```cpp
#include <cstdio>

const int MAXN = 1000005;

int prime[MAXN], primeCnt;

void sieve() {
    static bool notPrime[MAXN];
    notPrime[0] = notPrime[1] = true;
    primeCnt = 0;

    for (int i = 2; i < MAXN; i++) {
```

```
        if (!notPrime[i]) prime[primeCnt++] = i;

        for (int j = 0; j < primeCnt && i * prime[j] < MAXN; j++)
            notPrime[i * prime[j]] = true;
    }
}

int pow(int a, int n, int p) {
    int res = 1;
    for (; n; n >>= 1, a = a * a % p) if (n & 1) res = res * a % p;
    return res;
}

int getRoot(int p) {
    for (int g = 2, pp = p - 1; ; g++) {
        bool flag = true;
        for (int i = 0; i < primeCnt && prime[i] < p; i++) {
            if (pp % prime[i] == 0 && pow(g, pp / prime[i], p) == 1) {
                flag = false;
                break;
            }
        }

        if (flag) return g;
    }
}

int main() {

    return 0;
}
```

5.20 常系数线性齐次递推

5.20.1 暴力多项式取模

```
#include <cstdio>
#include <algorithm>

const int MOD = 1000000007;
const int MAXK = 2005;

long long a[MAXK];
void modMul(long long *A, long long *B, int n, long long *r) {
    static long long res[MAXK << 1];
    std::fill(res, res + (n << 1), 0);

    for (int i = 0; i < n; i++) for (int j = 0; j < n; j++) {
        res[i + j] += A[i] * B[j] % MOD;
        res[i + j] >= MOD ? res[i + j] -= MOD : 0;
    }

    for (int i = (n << 1) - 2; i >= n; i--) if (res[i]) for (int j = n - 1; ~j; j--) {
        res[i - n + j] += res[i] * a[j] % MOD;
```

```
        res[i - n + j] >= MOD ? res[i - n + j] -= MOD : 0;
    }

    for (int i = 0; i < n; i++) r[i] = res[i];
}

int main() {
    int n, k;
    scanf("%d %d", &n, &k);

    static long long h[MAXK];
    for (int i = 0; i < k; i++) {
        scanf("%lld", &a[i]);
        a[i] < 0 ? a[i] += MOD : 0;
    }
    std::reverse(a, a + k);
    a[k] = 1;

    for (int i = 0; i < k; i++) {
        scanf("%lld", &h[i]);
        h[i] < 0 ? h[i] += MOD : 0;
    }

    if (n < k) return printf("%lld\n", h[n]), 0;

    static long long m[MAXK], t[MAXK];
    m[0] = t[1] = 1;

    for (int i = n; i; i >>= 1, modMul(t, t, k, t)) if (i & 1) modMul(m, t, k, m);

    long long hn = 0;
    for (int i = 0; i < k; i++) hn = (hn + m[i] * h[i] % MOD) % MOD;

    printf("%lld\n", hn);

    return 0;
}
```

5.20.2 FFT 多项式取模

```
#include <cstdio>
#include <cmath>
#include <algorithm>

const int MAXN = 131072 + 1;
const int MOD = 1000000007;
const double PI = std::acos(-1);

long long qpow(long long a, long long n) {
    long long res = 1;
    for (; n; n >>= 1, a = a * a % MOD) if (n & 1) res = res * a % MOD;
    return res;
}

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

```
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);
```

```

    static long long t1[MAXN], t2[MAXN];
    int N = fft.extend(k << 1);
    std::copy(a, a + k, t1);
    std::fill(t1 + k, t1 + N, 0);
    polyMul(N, N, res, res, t2);
    polyMul(N, N, t1, t2, t1);
    for (int i = 0; i < k; i++) res[i] = (2 * res[i] % MOD - t1[i] + MOD) % MOD;
    std::fill(res + k, res + N, 0);
}

void polyDiv(int n, int m, long long *A, long long *B, long long *D, long long *R) {
        static long long A0[MAXN], B0[MAXN];

        int t = n - m + 1;
    int N = fft.extend(t << 1);

    std::fill(A0, A0 + N, 0);
    std::reverse_copy(B, B + m, A0);
        polyInv(t, A0, B0);

    std::reverse_copy(A, A + n, A0);

    polyMul(t, t, A0, B0, A0);

    std::reverse(A0, A0 + t);
    std::copy(A0, A0 + t, D);

    N = fft.extend(n);
    std::copy(B, B + m, B0);
    polyMul(t, m, A0, B0, A0);
        for (int i = 0; i < m; i++) R[i] = (A[i] - A0[i] + MOD) % MOD;
    std::fill(R + m, R + N, 0);
}

void polyPow(int n, long long *a, long long *mod, int k, long long *res) {
    res[0] = 1;
    static long long D[MAXN], R[MAXN];
    for (; k; k >>= 1) {
        if (k & 1) {
            polyMul(n, n, res, a, res);
            polyDiv(2 * n - 1, n, res, mod, D, R);
            std::copy(R, R + n, res);
        }
        polyMul(n, n, a, a, a);
        polyDiv(2 * n - 1, n, a, mod, D, R);
        std::copy(R, R + n, a);
    }
}

long long a[MAXN], x[MAXN], f[MAXN], t[MAXN], m[MAXN];

int main() {
    int n, k;
    scanf("%d %d", &n, &k);
    for (int i = 1; i <= k; i++) {
```

```
        scanf("%lld", &a[i]); // coeff
        a[i] < 0 && (a[i] += MOD);
    }
    for (int i = 0; i < k; i++) {
        scanf("%lld", &x[i]); // value
        x[i] < 0 && (x[i] += MOD);
    }

    f[k] = 1;
    for (int i = 0; i < k; i++) f[i] = a[k - i] ? MOD - a[k - i] : 0;

    t[1] = 1;
    polyPow(k + 1, t, f, n, m);

    long long ans = 0;
    for (int i = 0; i < k; i++) ans = (ans + m[i] * x[i]) % MOD;
    printf("%lld\n", ans);

    return 0;
}
```

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 $f(x)$。

```
#include <cstdio>

const int MAXN = 1000005;
const int MOD = 1000000007;

long long qpow(long long a, long long n) {
    long long res = 1;
    for (; n; n >>= 1, a = a * a % MOD) if (n & 1) res = res * a % MOD;
    return res;
}

long long fact[MAXN], invFact[MAXN];
void init() {
    fact[0] = 1;
    for (int i = 1; i < MAXN; i++) fact[i] = fact[i - 1] * i % MOD;
    invFact[MAXN - 1] = qpow(fact[MAXN - 1], MOD - 2);
    for (int i = MAXN - 2; ~i; i--) invFact[i] = invFact[i + 1] * (i + 1) % MOD;
}

long long Lagrange(long long *a, int n, long long x) {
    if (x <= n) return a[x];

    static long long f1[MAXN], f2[MAXN];
    f1[0] = f2[n + 1] = 1;
    for (int i = 1; i <= n + 1; i++) f1[i] = f1[i - 1] * (x - i + 1) % MOD;
    for (int i = n; ~i; i--) f2[i] = f2[i + 1] * (x - i) % MOD;

    long long res = 0;
    for (int i = 0; i <= n; i++) {
        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

```cpp
#include <cstdio>
#include <cmath>
#include <algorithm>

const int MAXN = 100005;
const double EPS = 1e-9;

int dcmp(double a, double b = 0) {
    double d = a - b;
    return std::abs(d) <= EPS ? 0 : (d > 0 ? 1 : -1);
}

struct Point {
    double x, y;

    Point(double x = 0, double y = 0) : x(x), y(y) {}

    bool operator<(const Point &rhs) const { return x == rhs.x ? y < rhs.y : x < rhs.x; }

    Point operator+(const Point &rhs) const { return Point(x + rhs.x, y + rhs.y); }
    Point operator-(const Point &rhs) const { return Point(x - rhs.x, y - rhs.y); }
    Point operator*(const double a) const { return Point(x * a, y * a); }
    Point operator/(const double a) const { return Point(x / a, y / a); }
    friend double dot(const Point &a, const Point &b) { return a.x * b.x + a.y * b.y; }
    friend double cross(const Point &a, const Point &b) { return a.x * b.y - a.y * b.x; }

    friend double angle(const Point &a, const Point &b) {
        return std::acos(dot(a, b) / a.length() / b.length());
    }

    Point rotate(double rad) {
        return Point(x * std::cos(rad) - y * std::sin(rad),
                     x * std::sin(rad) + y * std::cos(rad));
    }

    double length() const { return std::sqrt(dot(*this, *this)); }

    Point getNormal() const {
        Point res(-y, x);
        return res / length();
    }
} P[MAXN], ch[MAXN];

bool parallel(const Point &as, const Point &at, const Point &bs, const Point &bt) {
    return dcmp(cross(at - as, bt - bs)) == 0;
}

bool getLineInter(const Point &as, const Point &at, const Point &bs, const Point &bt, Point &res) {
    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;
```

```
    if (p.x < 0 && p.y >= 0) return 2;
    if (p.x <= 0 && p.y < 0) return 3;
    if (p.x > 0 && p.y <= 0) return 4;
}

bool cmpByAngle(const Point &a, const Point &b) {
    if (getv(a) == getv(b)) return cross(a, b) > 0;
    else return getv(a) < getv(b);
}

// *r == *l should be satisfied for the following 3 functions
double polyArea(Point *l, Point *r) {
    double sum = 0;
    for (Point *p = l; p < r; p++)
        sum += cross(*(p + 1), *p);
    return std::abs(sum / 2.0);
}

bool insidePoly(Point *l, Point *r, const Point &p) {
    int num = 0;
    for (Point *q = l; q < r; q++) {
        if (doesPointOnSeg(p, *q, *(q + 1))) return true;
        int k = dcmp(cross(*(q + 1) - *q, p - *q));
        int d1 = dcmp(q->y - p.y);
        int d2 = dcmp((q + 1)->y - p.y);
        if (k > 0 && d1 <= 0 && d2 > 0) ++num;
        if (k < 0 && d2 <= 0 && d1 > 0) --num;
    }
    return num;
}

Point polyWeight(Point *l, Point *r, double area) {
    Point res(0, 0);
    for (Point *p = l; p < r; p++)
        res = res + (*p + *(p + 1)) * cross(*(p + 1), *p);
    return res / area / 6.0;
}

int main() {

    return 0;
}
```

6.2 旋转卡壳

旋转卡壳求最小矩形覆盖：

```
#include <cstdio>
#include <cfloat>
#include <cmath>
#include <algorithm>

const int MAXN = 50005;
const double EPS = 1e-9;

```

```cpp
struct Point {
    double x, y;

    Point(double x = 0, double y = 0) : x(x), y(y) {}

    bool operator<(const Point &rhs) const { return x == rhs.x ? y < rhs.y : x < rhs.x; }

    Point operator+(const Point &rhs) const { return Point(x + rhs.x, y + rhs.y); }
    Point operator-(const Point &rhs) const { return Point(x - rhs.x, y - rhs.y); }
    Point operator*(const double a) const { return Point(x * a, y * a); }
    Point operator/(const double a) const { return Point(x / a, y / a); }
    friend double dot(const Point &a, const Point &b) { return a.x * b.x + a.y * b.y; }
    friend double cross(const Point &a, const Point &b) { return a.x * b.y - a.y * b.x; }

    friend double dist(const Point &a, const Point &b) {
        return sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y));
    }

    void print() {
        if (std::abs(x) <= EPS) x = 0;
        if (std::abs(y) <= EPS) y = 0;
        printf("%.5f %.5f\n", x, y);
    }

    Point getPerpendicular() {
        double X = sqrt(1 / (1 + (x / y) * (x / y)));

        int sx, sy;
        if (x > 0 && y > 0) sx = 1, sy = -1;
        else if (x > 0 && y <= 0) sx = 1, sy = 1;
        else if (x <= 0 && y > 0) sx = -1, sy = -1;
        else sx = 1, sy = -1;

        return Point(sx * X, sy * std::sqrt(1 - X * X));
    }
} P[MAXN], ch[MAXN];

struct Rectangle {
    Point p[4];

    void print() {
        int temp = 0;
        for (int i = 1; i < 4; i++)
            if (p[i].y < p[temp].y || (p[i].y == p[temp].y && p[i].x < p[temp].x)) temp = i;
        for (int i = 0; i < 4; i++) p[(i + temp) % 4].print();
    }

    Point &operator[](int i) { return p[i]; }
};

int getConvexHull(int n) {
    std::sort(P + 1, P + n + 1);

    int m = 0;
    for (int i = 1; i <= n; i++) {
        while (m > 1 && cross(ch[m - 1] - ch[m - 2], P[i] - ch[m - 2]) <= 0) m--;
```

```
        ch[m++] = P[i];
    }

    int k = m;
    for (int i = n; i; i--) {
        while (m > k && cross(ch[m - 1] - ch[m - 2], P[i] - ch[m - 2]) <= 0) m--;
        ch[m++] = P[i];
    }

    m > 1 ? m-- : 0;
    return m;
}

Rectangle ans;
double ansArea = DBL_MAX;
void rotatingCalipers(int n) {
    for (int curr = 0, right = 1, up = 1, left = 1; curr < n; curr++) {
        while (dot(ch[curr + 1] - ch[curr], ch[right + 1] - ch[right]) >= 0)
            right = (right + 1) % n;

        curr ? 0 : up = right;
        while (cross(ch[curr + 1] - ch[curr], ch[up + 1] - ch[up]) >= 0)
            up = (up + 1) % n;

        curr ? 0 : left = up;
        while (dot(ch[curr + 1] - ch[curr], ch[left + 1] - ch[left]) <= 0)
            left = (left + 1) % n;

        Point currV = ch[curr + 1] - ch[curr];
        double currLen = dist(ch[curr], ch[curr + 1]);
        double height = std::abs(cross(currV, ch[up] - ch[curr]) / currLen);
        double bottom = std::abs(dot(currV, ch[left] - ch[curr]) / currLen)
                      + std::abs(dot(currV, ch[right] - ch[curr]) / currLen);

        double currArea = bottom * height;
        Point currPerpendicular = currV.getPerpendicular();

        if (currArea < ansArea) {
            ansArea = currArea;
            ans[0] = ch[curr] + currV * fabs((dot(currV, ch[right] - ch[curr])) / currLen) / currLen;
            ans[1] = ans[0] + currPerpendicular * height;
            ans[2] = ans[1] - currV * bottom / currLen;
            ans[3] = ans[2] - currPerpendicular * height;
        }
    }
}

int main() {
    int n;
    scanf("%d", &n);

    for (int i = 1; i <= n; i++) scanf("%lf %lf", &P[i].x, &P[i].y);

    int m = getConvexHull(n);
    rotatingCalipers(m);

```

```
    printf("%.5f\n", ansArea);
    ans.print();

    return 0;
}
```

6.3 半平面交

```
#include <cstdio>
#include <cfloat>
#include <cmath>
#include <algorithm>

const int MAXN = 305;
const double EPS = 1e-14;

int dcmp(double a, double b = 0) {
    double d = a - b;
    return std::abs(d) <= EPS ? 0 : (d > 0 ? 1 : -1);
}

struct Point {
    double x, y;

    Point(double x = 0, double y = 0) : x(x), y(y) {}

    Point operator+(const Point &rhs) const { return Point(x + rhs.x, y + rhs.y); }
    Point operator-(const Point &rhs) const { return Point(x - rhs.x, y - rhs.y); }
    Point operator*(double rhs) const { return Point(x * rhs, y * rhs); }
    friend double cross(const Point &a, const Point &b) { return a.x * b.y - a.y * b.x; }
} P[MAXN], hpi[MAXN];

struct Line {
    Point p, v;
    double slop;

    Line() {}
    Line(const Point &p, const Point &v) : p(p), v(v) {
        slop = std::atan2(v.y, v.x);
    }

    Point getVal(double t) const { return p + v * t; }

    bool operator<(const Line &another) const {
        return slop < another.slop || (slop == another.slop && v.x
            && getVal(-p.x / v.x).y > getVal(-another.p.x / another.v.x).y);
    }

    friend Point getIntersection(const Line &a, const Line &b) {
        double t = cross(b.v, a.p - b.p) / cross(a.v, b.v);
        return a.getVal(t);
    }
} L[MAXN];

```

```cpp
int n;
int halfplaneIntersection() {
    int cnt = 0;
    L[cnt++] = L[0];
    for (int i = 1; i <= n; i++) if (dcmp(L[i].slop, L[i - 1].slop)) L[cnt++] = L[i];
    std::sort(L, L + cnt);

    static Line q[MAXN];
    static Point p[MAXN];
    int l = 0, r = 0;
    q[l] = L[0];
    for (int i = 1; i < cnt; i++) {
        while (l < r && dcmp(cross(L[i].v, p[r - 1] - L[i].p)) < 0) r--;
        while (l < r && dcmp(cross(L[i].v, p[l] - L[i].p)) < 0) l++;
        q[++r] = L[i];
        if (l < r) p[r - 1] = getIntersection(q[r - 1], q[r]);
    }
    while (l < r && dcmp(cross(q[l].v, p[r - 1] - q[l].p)) < 0) r--;
    while (l < r && dcmp(cross(q[r].v, p[l] - q[r].p)) < 0) l++;

    if (r - l <= 1) return 0;

    cnt = 0;
    for (int i = l; i < r; i++) hpi[++cnt] = p[i];
    return cnt;
}

int main() {
    scanf("%d", &n);
    for (int i = 1; i <= n; i++) scanf("%lf", &P[i].x);
    for (int i = 1; i <= n; i++) scanf("%lf", &P[i].y);

    P[0] = Point(P[1].x, P[1].y + 1);
    P[n + 1] = Point(P[n].x, P[n].y + 1);
    for (int i = 0; i <= n; i++) L[i] = Line(P[i], P[i + 1] - P[i]);

    int m = halfplaneIntersection();

    return 0;
}
```

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

```cpp
#include <cstdio>
#include <cmath>
#include <algorithm>

const int MAXN = 1005;
const double EPS = 1e-9;
const double PI = std::acos(-1.0);

int dcmp(double a, double b = 0.0) {
    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(0), 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      return ans / 2.0;
50  }
51
52  int numberOfCircleCross(const Circle a, const Circle b, Circle &cp1, Circle &cp2) {
53      double mx = b.p.x - a.p.x, sx = b.p.x + a.p.x, mx2 = mx * mx;
54      double my = b.p.y - a.p.y, sy = b.p.y + a.p.y, my2 = my * my;
55      double sq = mx2 + my2, d = -(sq - sqr(a.r - b.r)) * (sq - sqr(a.r + b.r));
56
57      if (dcmp(d) < 0) return 0;
58      d = safeSqrt(d);
59
60      double x = mx * ((a.r + b.r) * (a.r - b.r) + mx * sx) + sx * my2;
61      double y = my * ((a.r + b.r) * (a.r - b.r) + my * sy) + sy * mx2;
62      double dx = mx * d, dy = my * d;
63      sq *= 2;
64
65      cp1.p.x = (x - dy) / sq;
```

```
    cp1.p.y = (y + dx) / sq;
    cp2.p.x = (x + dy) / sq;
    cp2.p.y = (y - dx) / sq;
    return dcmp(d) > 0 ? 2 : 1;
}

void circleUnion(Circle *C, int n) {
    std::sort(C, C + n, [](const Circle &a, const Circle &b) {return a.r < b.r;});

    for (int i = 0; i < n; i++) for (int j = i + 1; j < n; j++)
        if (dcmp(dist(C[i].p, C[j].p) + C[i].r - C[j].r) <= 0) ++C[i].d;

    for (int i = 0; i < n; i++) {
        int tn = 0, cnt = 0;
        for (int j = 0; j < n; j++) if (i != j) {
            static Circle cp1, cp2;
            // pay attention to the order of parameter!!!
            if (numberOfCircleCross(C[i], C[j], cp2, cp1) < 2) continue;
            cp1.angle = std::atan2(cp1.p.y - C[i].p.y, cp1.p.x - C[i].p.x);
            cp2.angle = std::atan2(cp2.p.y - C[i].p.y, cp2.p.x - C[i].p.x);
            cp1.d = 1;
            cp2.d = -1;
            tp[tn++] = cp1;;
            tp[tn++] = cp2;
            if (dcmp(cp1.angle, cp2.angle) > 0) ++cnt;
        }
        tp[tn++] = Circle(C[i].p.x - C[i].r, C[i].p.y, PI, -cnt);
        tp[tn++] = Circle(C[i].p.x - C[i].r, C[i].p.y, -PI, cnt);
        std::sort(tp, tp + tn, [](const Circle &a, const Circle &b) {
            return !dcmp(a.angle, b.angle) ? a.d > b.d : a.angle < b.angle;
        });
        int s = C[i].d + tp[0].d;
        for (int j = 1; j < tn; j++) {
            int p = s;
            s += tp[j].d;
            area[p] += calc(C[i], tp[j - 1], tp[j]);
        }
    }
}

void solve(Circle *C, int n) {
    std::fill(area + 1, area + n + 1, 0);
    circleUnion(C, n);

    // delete this line if you want to know area coverd by at least i-times instead of exactly i-times
    for (int i = 1; i < n; i++) area[i] -= area[i + 1];
}

} // namespace CircleUnion

int main() {
    int n;
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        scanf("%lf %lf %lf", &C[i].p.x, &C[i].p.y, &C[i].r);
        C[i].d = 1;
```

```
    }

    CircleUnion::solve(C, n);

    return 0;
}
```

6.5 平面图区域（Farmland）

```
#include <cstdio>
#include <cmath>
#include <vector>
#include <algorithm>

const int MAXN = 200005;
const double EPS = 1e-9;
const double PI = std::acos(-1.0);

int dcmp(double a, double b = 0) {
    double d = a - b;
    return std::abs(d) <= EPS ? 0 : (d > 0 ? 1 : -1);
}

struct Point {
    int x, y;

    friend long long cross(const Point &a, const Point &b) {
        return 1ll * a.x * b.y - 1ll * a.y * b.x;
    }
} P[MAXN];

struct Edge {
    int u, v;
    double a;
    bool vis;

    bool operator<(const Edge &rhs) const { return a < rhs.a; }

    Edge(int u, int v, double a) : u(u), v(v), a(a), vis(false) {}
};
std::vector<Edge> N[MAXN];

void addEdge(int u, int v, double a1, double a2) {
    N[u].emplace_back(u, v, a1);
    N[v].emplace_back(v, u, a2);
}

int find(const std::vector<Edge> &e, double a) {
    double d = a + PI - EPS;
    if (dcmp(d, PI) > 0) d -= 2 * PI;
    int res = std::upper_bound(e.begin(), e.end(), d, [](double a, const Edge &b) {
                return a < b.a;
            }) - e.begin() - 1;
    if (res < 0) res += e.size();
```

```
    return res;
}

int main() {
    int n, m;
    scanf("%d %d", &n, &m);
    for (int i = 1; i <= n; i++) scanf("%d %d", &P[i].x, &P[i].y);
    for (int i = 0, u, v; i < m; i++) {
        scanf("%d %d", &u, &v);
        double a1 = std::atan2(P[v].y - P[u].y, P[v].x - P[u].x);
        double a2 = std::atan2(P[u].y - P[v].y, P[u].x - P[v].x);
        addEdge(u, v, a1, a2);
    }
    for (int i = 1; i <= n; i++) std::sort(N[i].begin(), N[i].end());

    static std::vector<long long> ans;
    for (int i = 1; i <= n; i++) {
        for (Edge &e : N[i]) if (!e.vis) {
            int u = i, v = e.v;
            long long S = cross(P[u], P[v]);
            double lasta = e.a;
            e.vis = true;
            do {
                int t = find(N[v], lasta);
                if (N[v][t].vis) {
                    S = 0;
                    break;
                }
                u = v;
                v = N[u][t].v;
                lasta = N[u][t].a;
                N[u][t].vis = true;
                S += cross(P[u], P[v]);
            } while (v != i);
            if (S > 0) ans.push_back(S);
        }
    }

    std::sort(ans.begin(), ans.end());
    printf("%zu\n", ans.size());
    for (auto i : ans) printf("%lld\n", i);

    return 0;
}
```

6.6 Deluanay 三角剖分与平面欧几里得距离最小生成树

```
#include <cstdio>
#include <cmath>
#include <set>
#include <list>
#include <vector>
#include <algorithm>

```

```cpp
const int MAXN = 100005;
const double EPS = 1e-9;

int dcmp(double a, double b = 0) {
    double d = a - b;
    return std::abs(d) <= EPS ? 0 : (d > 0 ? 1 : -1);
}

double sqr(double x) { return x * x; }

struct Point {
    double x, y;
    int id;

    Point(double x = 0, double y = 0, int id = -1) : x(x), y(y), id(id) {}

    bool operator<(const Point &rhs) const {
        return dcmp(x, rhs.x) == 0 ? y < rhs.y : x < rhs.x;
    }

    Point operator+(const Point &rhs) const { return Point(x + rhs.x, y + rhs.y); }
    Point operator-(const Point &rhs) const { return Point(x - rhs.x, y - rhs.y); }
    friend double dot(const Point &a, const Point &b) { return a.x * b.x + a.y * b.y; }
    friend double cross(const Point &a, const Point &b) { return a.x * b.y - a.y * b.x; }

    friend double dist(const Point &a, const Point &b) {
        return std::sqrt(sqr(a.x - b.x) + sqr(a.y - b.y));
    }
    friend double distSqr(const Point &a, const Point &b) {
        return sqr(a.x - b.x) + sqr(a.y - b.y);
    }
} P[MAXN];

namespace Delaunay {

struct Point3D {
    double x, y, z;

    Point3D(double x = 0, double y = 0, double z = 0) : x(x), y(y), z(z) {}
    Point3D(const Point &p) : x(p.x), y(p.y), z(sqr(p.x) + sqr(p.y)) {}

    friend Point3D operator-(const Point3D &a, const Point3D &b) {
        return Point3D(a.x - b.x, a.y - b.y, a.z - b.z);
    }

    friend double dot(const Point3D &a, const Point3D &b) {
        return a.x * b.x + a.y * b.y + a.z * b.z;
    }
    friend Point3D cross(const Point3D &a, const Point3D &b) {
        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);
    }
    friend double mix(const Point3D &a, const Point3D &b, const Point3D &c) {
        return dot(a, cross(b, c));
    }
};

```

```
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 }
```

```

bool isParallel(const Point &a, const Point &b, const Point &c) {
    return !dcmp(cross(b - a, c - a));
}

void getEdge(std::vector<std::pair<int, int> > &ret) {
    ret.reserve(n);
    static std::set<std::pair<int, int> > vis;
    vis.clear();

    for (int i = 0; i < n; i++) for (auto j : head[i]) {
        if (j.id < i) continue;
        Point now = p[j.id];
        for (auto k : head[i]) {
            if (k.id < i) continue;
            if (isParallel(p[i], p[j.id], p[k.id])
                    && distSqr(p[k.id], p[i]) < distSqr(now, p[i]))
                now = p[k.id];
        }
        if (vis.find(std::make_pair(p[i].id, now.id)) == vis.end()) {
            vis.insert(std::make_pair(p[i].id, now.id));
            ret.push_back(std::make_pair(p[i].id, now.id));
        }
    }
}

void init(int n, Point *P) {
    std::copy(P, P + n, p);
    std::sort(p, p + n);
    for (int i = 0; i < n; i++) rename[p[i].id] = i;
    Delaunay::n = n;
    divide(0, n - 1);
}

void split(int n, Point *P, std::vector<std::pair<int, int> > &ret) {
    for (int i = 0; i < n; i++) P[i].id = i;
    init(n, P);
    getEdge(ret);
}

} // namespace Delaunay

namespace Kruskal {

struct DSU {
    int f[MAXN];

    void init(int n) { for (int i = 1; i <= n; i++) f[i] = i; }
    int find(int x) { return x == f[x] ? x : f[x] = find(f[x]); }
    void merge(int x, int y) { f[find(y)] = find(x); }
    bool test(int x, int y) { return find(x) == find(y); }
} dsu;

struct Edge {
    int u, v;
    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);
```

```
}

double asqrt(double x) { return x <= EPS ? 0 : std::sqrt(x); }
double sqr(double x) { return x * x; }

struct Point {
    double x, y, z;

    Point(double x = 0, double y = 0, double z = 0) : x(x), y(y), z(z) {}

    Point operator+(const Point &rhs) const {
        return Point(x + rhs.x, y + rhs.y, z + rhs.z);
    }
    Point operator-(const Point &rhs) const {
        return Point(x - rhs.x, y - rhs.y, z - rhs.z);
    }
    bool operator==(const Point &rhs) const {
        return !dcmp(x, rhs.x) && !dcmp(y, rhs.y) && !dcmp(z, rhs.z);
    }
    bool operator<(const Point &rhs) const {
        if (dcmp(x, rhs.x)) return x < rhs.x;
        if (dcmp(y, rhs.y)) return y < rhs.y;
        return z < rhs.z;
    }

    friend double dot(const Point &a, const Point &b) {
        return a.x * b.x + a.y * b.y + a.z * b.z;
    }
    friend Point cross(const Point &a, const Point &b) {
        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);
    }
    friend double mix(const Point &a, const Point &b, const Point &c) {
        return dot(a, cross(b, c));
    }

    double length() const {
        return std::sqrt(sqr(x) + sqr(y) + sqr(z));
    }
} P[MAXN];

double volumn(const Point &a, const Point &b, const Point &c, const Point &d) {
    return mix(b - a, c - a, d - a) / 6.0;
}

bool doesPointOnLine(const Point &p, const Point &s, const Point &t) {
    return dcmp(cross(p - s, t - s).length()) == 0;
}

namespace ConvexHull3D {

struct Face {
    int a, b, c;
    bool judged;

    Face(int a = 0, int b = 0, int c = 0) : a(a), b(b), c(c), judged(false) {}

```

```
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     }
```

```
    ch = temp;
    for (auto f : temp) {
        int a = f[0], b = f[1], c = f[2];
        if (mark[a][b] == cnt) ch.emplace_back(b, a, pi);
        if (mark[b][c] == cnt) ch.emplace_back(c, b, pi);
        if (mark[c][a] == cnt) ch.emplace_back(a, c, pi);
    }
}

bool getConvexHull(int n) {
    std::sort(P, P + n);
    n = std::unique(P, P + n) - P;
    std::random_shuffle(P, P + n);

    if (find(n)) {
        memset(mark, 0, sizeof (mark));
        int cnt = 0;
        for (int i = 3; i < n; i++) add(i, cnt);
    } else {
        return false;
    }

    // E, F, V: The number of edges, faces and vertices when each face is a triangle
    // EE, FF, VV: The number of edges, faces and vertices after dealing with co-planar triangles
    int F = ch.size(), V = (4 + F) / 2, E = V + F - 2;
    int FF = 0, EE = E;
    for (int i = 0; i < ch.size(); i++) if (!ch[i].judged) {
        static std::vector<Face> f;
        f.clear();
        for (int j = 0; j < ch.size(); j++) if (!ch[j].judged && same(i, j)) {
            f.push_back(ch[j]);
            ch[j].judged = true;
        }
        for (int j = 0; j < f.size(); j++) for (int k = 0; k < 3; k++) for (int l = 0; l < j; l++)
            if (doesFaceHaveEdge(f[l], f[j][k], f[j][(k + 1) % 3])) {
                --EE;
                break;
            }
        ++FF;
    }
    int VV = EE + 2 - FF;

    double area = 0; // surface area
    for (auto f : ch) {
        Point p = cross(P[f[0]] - P[f[1]], P[f[2]] - P[f[1]]);
        area += p.length() / 2.0;
    }

    double vol = 0; // volumn
    const Point &vp = P[ch[0][0]];
    for (auto f : ch) vol += std::abs(volumn(vp, P[f[0]], P[f[1]], P[f[2]]));

    return true;
}

} // namespace ConvexHull3D
```

```

int main() {
    int n;
    scanf("%d", &n);
    for (int i = 0; i < n; i++) scanf("%lf %lf %lf", &P[i].x, &P[i].y, &P[i].z);

    // Deal with co-linear points
    static bool banned[MAXN];
    for (int i = 0; i < n; i++) if (!banned[i]) for (int j = 0; j < i; j++) if (!banned[j]) {
        static std::vector<std::pair<Point, int> > l;
        l.clear();
        for (int k = 0; k < n; k++) if (!banned[k] && doesPointOnLine(P[k], P[i], P[j]))
            l.emplace_back(P[k], k);
        std::sort(l.begin(), l.end());
        for (int k = 1; k < l.size() - 1; k++) banned[l[k].second] = true;
    }

    int p = 0;
    for (int i = 0; i < n; i++) if (!banned[i]) P[p++] = P[i];

    ConvexHull3D::getConvexHull(p);

    return 0;
}
```

6.8 动态半凸壳

```
#include <cstdio>
#include <algorithm>

const int MAXN = 1000005;
const double EPS = 1e-9;

int dcmp(double a, double b = 0) {
    double d = a - b;
    return std::abs(d) <= EPS ? 0 : (d > 0 ? 1 : -1);
}

struct Splay {
    struct Node {
        Node *c[2], *fa, *pred, *succ;
        double x, y;

        Node() {}
        Node(Node *fa, double x, double y) : x(x), y(y), c(), fa(fa), pred(NULL), succ(NULL) {}

        int relation() { return fa->c[1] == this; }
    } *root, _pool[MAXN], *_curr;

    Splay() : root(NULL), _curr(_pool) {}

    void rotate(Node *u) {
        Node *o = u->fa;
        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

```
1  #include <cstdio>
```

```cpp
#include <cmath>

double f(double x) {
    return std::sqrt(x);
}

double simpson(double l, double r) {
    double mid = (l + r) / 2.0;
    return (f(l) + 4.0 * f(mid) + f(r)) * (r - l) / 6.0;
}

double integral(double l, double r, double eps) {
    double mid = (l + r) / 2.0;
    double ST = simpson(l, r), SL = simpson(l, mid), SR = simpson(mid, r);
    if (std::abs(SL + SR - ST) <= 15.0 * eps) return SL + SR + (SL + SR - ST) / 15.0;
    return integral(l, mid, eps / 2.0) + integral(mid, r, eps / 2.0);
}

int main() {
    double l, r, eps;
    scanf("%lf %lf %lf", &l, &r, &eps);
    printf("%.6f\n", integral(l, r, eps));

    return 0;
}
```

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。

$k=1:\ a_n = 2a_{n-1}$

$k=2:\ a_n = a_{n-1} + a_{n-2}$

$k=3:\ a_n = a_{n-1} + a_{n-2} - a_{n-3} + 1$

```cpp
#include <cstdio>

const int MAXN = 2000005;

int a[MAXN], b[MAXN];

int main() {
    int n, k; // n <= 100,000,000, k <= 100,000
    scanf("%d %d", &n, &k);

    a[0] = b[0] = 1;
    int i = 0, j = 0;
    while (a[i] < n) {
        ++i;
        a[i] = b[i - 1] + 1;
        while (a[j + 1] * k < a[i]) ++j;
        if (a[j] * k < a[i]) b[i] = a[i] + b[j];
        else b[i] = a[i];
    }

    if (a[i] == n) puts("Lose");
    else {
        int ans = 0;
        while (n) {
            if(n >= a[i]) {
                n -= a[i];
                ans = a[i];
            }
            --i;
        }
        printf("%d\n",ans);
    }

    return 0;
}
```

7.2 Nim-3

三人版的 Nim 取石子游戏，取完的人为一位，他的下一人为三位。

先手三位的条件是每个二进制位的 1 的个数为 3 的倍数。

```cpp
#include <cstdio>

const int MAXN = 1000005;

long long a[MAXN];
int cnt[63];

int main() {
    int n;
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        scanf("%lld", &a[i]);
        for (int j = 0; j < 63; j++) cnt[j] += !!(a[i] & (1ll << j));
    }

    bool isC = true;
    for (int i = 0; i < 63; i++) if (cnt[i] % 3) {
        isC = false;
        break;
    }

    if (isC) {
        puts("C");
    } else {
        bool isA = false;
        for (int i = 0; i < n; i++) {
            bool flag = true;
            long long temp = 0;
            for (int j = 0; j < 63; j++) {
                int c = (cnt[j] - !!(a[i] & (1ll << j))) % 3;
                if (c == 1) {
                    flag = false;
                    break;
                }
                if (c == 2) temp |= (1ll << j);
            }
            if (flag && temp < a[i]) {
                isA = true;
                break;
            }
        }
        puts(isA ? "A" : "B");
    }

    return 0;
}
```

8 其他

8.1 Java 高精度

```java
import java.math.BigInteger;
import java.util.Scanner;

public class Main {
    public static void main(String args[]) {
        // read
        Scanner in = new Scanner(System.in);
        BigInteger a = in.nextBigInteger();

        // 3 special numbers
        a = BigInteger.ZERO;
        a = BigInteger.ONE;
        a = BigInteger.TEN;

        // convert an int to BigInteger
        BigInteger b = BigInteger.valueOf(2333);
        BigInteger p = BigInteger.valueOf(998244353);

        // convert a BigInteger to int
        int i = b.intValue();
        // convert a BigInteger to long
        long l = b.longValue();

        // operations:
        // a + b;                               BigInteger add(BigInteger b);
        a.add(b);
        // a - b;                               BigInteger subtract(BigInteger b);
        a.subtract(b);
        // a * b;                               BigInteger multiply(BigInteger b);
        a.multiply(b);
        // a / b;                               BigInteger divide(BigInteger b);
        a.divide(b);
        // a % b;                               BigInteger mod(BigInteger b);
        a.mod(b);
        // -a;                                  BigInteger negate();
        a.negate();
        // a < 0 ? -1 : (a > 0 ? 1 : 0);        int signum();
        a.signum();
        // signum(a - b);                       int compareTo(BigInteger b);
        a.compareTo(b);
        // a == b;                              boolean equals(BigInteger b);
        a.equals(b);

        // abs(a);                              BigInteger abs();
        a.abs();
        // max(a, b);                           BigInteger max(BigInteger b);
        a.max(b);
        // min(a, b);                           BigInteger min(BigInteger b);
        a.min(b);
```

```
        // gcd(a, b);                           BigInteger gcd(BigInteger b);
        a.gcd(b);
        // pow(a, i);                           BigInteger pow(int i);
        a.pow(i);

        // modPow(a, b, p);              BigInteger modPow(BigInteger b, BigInteger p);
        a.modPow(b, p);
        // modPow(a, p - 2, p);                 BigInteger modInverse(BigInteger p);
        a.modInverse(p);
        // isPrime(a); (probability:1 - 0.5^i)  boolean isProbablePrime(int certainty);
        a.isProbablePrime(i);

        // a << i;                              BigInteger shiftLeft(int i);
        a.shiftLeft(i);
        // a >> i;                              BigInteger shiftRight(int i);
        a.shiftRight(i);
        // a ^ b;                               BigInteger xor(BigInteger b);
        a.xor(b);
        // a | b;                               BigInteger or(BigInteger b);
        a.or(b);
        // a & b;                               BigInteger and(BigInteger b);
        a.and(b);
        // ~a;                                  BigInteger not();
        a.not();
        // a & ~b;                              BigInteger andNot(BigInteger b);
        a.andNot(b);

        // ((a >> i) & 1) == 1;                 BigInteger testBit(int i);
        a.testBit(i);
        // a | (1 << i);                        BigInteger setBit(int i);
        a.setBit(i);
        // a & ~(1 << i);                       BigInteger clearBit(int i);
        a.clearBit(i);
        // a ^ (1 << i);                        BigInteger flipBit(int i);
        a.flipBit(i);
        // a & -a;                              BigInteger getLowerSetBit();
        a.getLowestSetBit();
        // __builtin_popcount(a);               int bitCount();
        a.bitCount();
        // ceil(log2(this < 0 ? -this : this+1)) int bitLength();
        a.bitLength();
    }
}
```

8.2 三维偏序

```
#include <cstdio>
#include <algorithm>

const int MAXN = 100005;
const int MAXK = 200005;

struct Data {
    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          }
```

```
    }

    for (Data *p = temp, *q = l; q <= r; q++, p++) {
        bit.clear(q->c);
        *q = *p;
    }
}

int main() {
    int n, k;
    scanf("%d %d", &n, &k);

    for (int i = 0; i < n; i++) {
        scanf("%d %d %d", &a[i].a, &a[i].b, &a[i].c);
        a[i].cnt = 1;
    }

    std::sort(a, a + n);
    int cnt = 0;
    for (int i = 0; i < n; i++) {
        if (i && a[i] == a[i - 1]) a[cnt - 1].cnt++;
        else a[cnt++] = a[i];
    }

    bit.init(k);
    divide(a, a + cnt - 1);

    static int ans[MAXN];
    for (int i = 0; i < cnt; i++) ans[a[i].ans] += a[i].cnt;
    for (int i = 0; i < n; i++) printf("%d\n", ans[i]);

    return 0;
}
```

8.3 $O(1)$ 取模乘

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

8.4 std::bitset 遍历

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

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。

```cpp
#include <cstdio>
#include <cstring>
#include <algorithm>

const int MAXN = 5005, SIGMA = 26;

struct Bitset {
    static constexpr int W = 62;

    static constexpr size_t size = MAXN / W + !!(MAXN % W);
    unsigned long long u[size];

    Bitset &operator=(const Bitset &rhs) {
        std::copy(rhs.u, rhs.u + size, u);
        return *this;
    }

    void reset() {
        std::fill(u, u + size, 0);
    }
    void set(int x) {
        u[x / W] |= 1ull << (x % W);
    }
    bool get(int x) const {
        return u[x / W] & (1ull << (x % W));
    }

    Bitset operator|(const Bitset &rhs) const {
        Bitset res;
        for (int i = 0; i < size; i++) res.u[i] = u[i] | rhs.u[i];
        return res;
    }
    Bitset operator&(const Bitset &rhs) const {
        Bitset res;
        for (int i = 0; i < size; i++) res.u[i] = u[i] & rhs.u[i];
        return res;
    }
    Bitset operator^(const Bitset &rhs) const {
        Bitset res;
        for (int i = 0; i < size; i++) res.u[i] = u[i] ^ rhs.u[i];
        return res;
    }

    Bitset operator-(const Bitset &rhs) const {
        Bitset res;
        for (int i = 0; i < size; i++) res.u[i] = u[i] - rhs.u[i];
        for (int i = 0; i < size; ++ i) if (res.u[i] < 0) {
            res.u[i] += 1ll << W;
            --res.u[i + 1];
        }
        res.u[size] = 0;
        return res;
    }

    void shlOr1() {
        unsigned long long c = 1;
```

```
        for (int i = 0; i < size; ++ i) {
            u[i] <<= 1; u[i] |= c;
            c = u[i] >> W & 1;
            u[i] ^= c << W;
        }
    }

    size_t count() const {
        size_t res = 0;
        for (int i = 0; i < size; ++ i) res += __builtin_popcountll(u[i]);
        return res;
    }
};
Bitset row, as[SIGMA], x;

char s[MAXN], t[MAXN];

int main() {
    scanf("%s %s", s, t);
    int n = strlen(s), m = strlen(t);

    for (int i = 0; i < SIGMA; i++) as[i].reset();
    for (int i = 0; i < n; i++) as[s[i] - 'a'].set(i);

    row.reset();
    for (int i = 0; i < m; i++) {
        x = row | as[t[i] - 'a'];
        row.shlOr1();
        row = x - row;
        row = (row ^ x) & x;
        printf("%d\n", row.count());
    }
    return 0;
}
```

8.6 IO 优化

```
struct IO {
    static constexpr int BUF_SIZE = 1048576;
    char ibuf[BUF_SIZE], *ip, *ie, obuf[BUF_SIZE], *op;
    std::streambuf *rfb, *ofb;

    IO() : op(obuf) {
        ip = ie = NULL;
        rfb = std::cin.rdbuf();
        ofb = std::cout.rdbuf();
    }
    ~IO() {
        ofb->sputn(obuf, op - obuf);
    }

    char read() {
        if (ip == ie) ie = (ip = ibuf) + rfb->sgetn(ibuf, BUF_SIZE);
        return ip == ie ? -1 : *ip++;
```

```
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;
```