Backlight's Code Template

Backlight @ CSU

2021 年 4 月 30 日

目录

1 ds

1.1 AVLTree

```cpp
#include <bits/stdc++.h>
using namespace std;

#define FOR(i, l, r) for (int i = l; i <= r; ++i)
#define ROF(i, r, l) for (int i = r; i >= l; --i)
#define REP(i, l, r) for (int i = l; i < r; ++i)
#define PER(i, r, l) for (int i = r - 1; i >= l; --i)

const int N = 1e5 + 5;
using ll = long long;

template<typename T>
struct AVLTree {
    struct node {
        T v;
        int sz, h, cnt;
        node *l, *r;
        node(T _v) : v(_v)
        {
            sz = h = cnt = 1;
            l = r = nullptr;
        }
    };
    node *root = nullptr;
    int get_size(node* p)
    {
        return p ? p->sz : 0;
    }

    int get_height(node* p)
    {
        return p ? p->h : 0;
    }

    void push_up(node *p)
    {
        if (!p) return;
        p->sz = get_size(p->l) + p->cnt + get_size(p->r);
        p->h = 1 + max(get_height(p->l), get_height(p->r));
    }

    void zig(node* &p)
    {
        node* q = p;
        q = p->r;
        p->r = q->l;
        q->l = p;
        push_up(p);
        push_up(q);
        p = q;
    }

    void zag(node* &p)
    {
        node* q = p->l;
        p->l = q->r;
        q->r = p;
        push_up(p);
        push_up(q);
        p = q;
    }
```

```
62
63      void maintain(node* &p)
64      {
65          if (!p) return;
66          if (get_height(p->l) - get_height(p->r) == 2) {
67              if (get_height(p->l->l) < get_height(p->l->r)) {
68                  zig(p->l);
69              }
70              zag(p);
71          } else if (get_height(p->l) - get_height(p->r) == -2) {
72              if (get_height(p->r->l) > get_height(p->r->r)) {
73                  zag(p->r);
74              }
75              zig(p);
76          }
77      }
78
79      void ins(node* &p, T v)
80      {
81          if (!p) {
82              p = new node(v);
83              return;
84          }
85          if (p->v == v) {
86              ++(p->cnt);
87          } else {
88              if (v < p->v) {
89                  ins(p->l, v);
90              } else {
91                  ins(p->r, v);
92              }
93          }
94          push_up(p);
95          maintain(p);
96          push_up(p);
97      }
98
99      void del(node* &p, T v)
100     {
101         if (!p) return;
102         if (p->v == v) {
103             if (p->cnt == 1) {
104                 if (p->l && p->r) {
105                     node* q = p->r;
106                     while(q->l) q = q->l;
107                     p->cnt = q->cnt, p->v = q->v;
108                     q->cnt = 1;
109                     del(p->r, q->v);
110                 } else {
111                     node* q = p;
112                     if (p->l) p = p->l;
113                     else if (p->r) p = p->r;
114                     else p = nullptr;
115                     delete q;
116                     q = nullptr;
117                 }
118             } else {
119                     --p->cnt;
120             }
121         } else {
122             if (v < p->v) del(p->l, v);
123             else del(p->r, v);
124         }
125         push_up(p);
126         maintain(p);
```

```
127         push_up(p);
128     }
129 
130     void ins(T v)
131     {
132         ins(root, v);
133     }
134 
135     void del(T v)
136     {
137         del(root, v);
138     }
139 
140     int getRank(T v)
141     {
142         node* p = root;
143         int res = 0;
144         while(p) {
145             if (v == p->v) {
146                 res += get_size(p->l);
147                 break;
148             }
149             if (v < p->v) p = p->l;
150             else {
151                 res += get_size(p->l) + p->cnt;
152                 p = p->r;
153             }
154         }
155         return res + 1;
156     }
157 
158     T getKth(int k)
159     {
160         node *p = root;
161         T res = -1;
162         while(p) {
163             if (k <= get_size(p->l)) p = p->l;
164             else if (k - get_size(p->l) <= p->cnt) {
165                 res = p->v;
166                 break;
167             } else {
168                 k -= get_size(p->l) + p->cnt;
169                 p = p->r;
170             }
171         }
172         return res;
173     }
174 
175     T getPrev(T v)
176     {
177         T res = numeric_limits<T>::min();
178         node* p = root;
179         while(p) {
180             if (v == p->v) {
181                 node *q = p->l;
182                 while(q->r) q = q->r;
183                 res = q->v;
184                 break;
185             }
186 
187             if (v < p->v) {
188                 p = p->l;
189             } else {
190                     if (p->v > res) res = p->v;
191                 p = p->r;
```

```
            }
        }
        return res;
    }

    T getSucc(T v)
    {
        T res = numeric_limits<T>::max();
        node* p = root;
        while(p) {
            if (v == p->v) {
                node *q = p->r;
                while(q->l) q = q->l;
                res = q->v;
                break;
            }

            if (v < p->v) {
                    if (p->v < res) res = p->v;
                p = p->l;
            } else {
                p = p->r;
            }
        }
        return res;
    }

    void debug(node* p)
    {
        if (!p) return;
        debug(p->l);
        cerr << p->v << " ";
        debug(p->r);
    }

    void debug()
    {
        cerr << "INORDER: " << endl;
        debug(root);
        cerr << endl;
    }
};

void solve(int Case)
{
    int n;
    scanf("%d", &n);
    int op, x;
    AVLTree<int> t;
    FOR(i, 1, n) {
        scanf("%d %d", & op, &x);
//        cerr << op << " " << x << endl;
        switch(op) {
            case 1:
                t.ins(x);
                break;
            case 2:
                t.del(x);
                break;
            case 3:
                printf("%d\n", t.getRank(x));
                break;
            case 4:
                printf("%d\n", t.getKth(x));
                break;
```

```
            case 5:
                printf("%d\n", t.getPrev(x));
                break;
            case 6:
                printf("%d\n", t.getSucc(x));
                break;
        }
//        t.debug();
    }
}

int main()
{
#ifdef BACKLIGHT
    freopen("in.txt", "r", stdin);
#endif
    int T = 1;
//    scanf("%d", &T);
    for (int _ = 1; _ <= T; ++_) solve(_);
    return 0;
}
```

1.2 BTree

```
template <typename K, int BF>
class BTree
{
  public:
    typedef std::pair<K, int> value_type;

  private:
    struct Node
    {
        value_type values[2 * BF - 1];
        Node *child[2 * BF] = {nullptr};
        Node *p = nullptr;
        int keyNum = 0, size = 0;
        bool isLeaf = true;
        const K &key(int i) const { return values[i].first; }
        int &cnt(int i) { return values[i].second; }
        Node(Node *p = nullptr) : p(p) {}
    };
    Node *root = nullptr;
    static bool pairComp(const value_type &lhs, const K &rhs) { return lhs.first < rhs; }
    template <typename T>
    static void shiftBy(T *ptr, int length, int shift) { memmove(ptr + shift, ptr, length * sizeof(T)); }
    static int calcSize(Node *x)
    {
        if (!x)
            return 0;
        int nsz = 0;
        for (int i = 0; i < x->keyNum; ++i)
            nsz += getSize(x->child[i]) + x->cnt(i);
        nsz += getSize(x->child[x->keyNum]);
        return nsz;
    }
    static int getSize(Node *x)
    {
        if (!x)
            return 0;
        return x->size;
    }
    //把 where 孩子分成两个节点，都作为 x 的孩子
    void split(Node *x, int where)
```

```
41      {
42          Node *z = new Node(x);
43          Node *y = x->child[where];
44          z->isLeaf = y->isLeaf;
45          memmove(z->values, y->values + BF, (BF - 1) * sizeof(value_type));
46          if (!y->isLeaf)
47          {
48              memmove(z->child, y->child + BF, BF * sizeof(Node *));
49              for (int i = 0; i < BF; ++i)
50                  z->child[i]->p = z;
51          }
52          z->keyNum = y->keyNum = BF - 1;
53          shiftBy(x->child + where + 1, x->keyNum - where, 1); //注意 child 本身 keyNum 多一个
54          x->child[where + 1] = z;
55          shiftBy(x->values + where, x->keyNum - where, 1);
56          new (x->values + where) value_type(y->values[BF - 1]);
57
58          y->size = calcSize(y), z->size = calcSize(z);
59          ++x->keyNum;
60      }
61      void insertEmpty(Node *x, const K &key)
62      {
63          while (true)
64          {
65              int i = lower_bound(x->values, x->values + x->keyNum, key, pairComp) - x->values;
66              if (i != x->keyNum && !(key < x->values[i].first)) //重复插入
67              {
68                  ++x->cnt(i);
69                  while (x)
70                      ++x->size, x = x->p;
71                  return;
72              }
73              if (x->isLeaf)
74              {
75                  shiftBy(x->values + i, x->keyNum - i, 1);
76                  x->values[i] = {key, 1};
77                  ++x->keyNum;
78                  while (x)
79                      ++x->size, x = x->p;
80                  return;
81              }
82              if (x->child[i]->keyNum == 2 * BF - 1)
83              {
84                  split(x, i);
85                  if (x->key(i) < key)
86                      ++i;
87                  else if (!(key < x->key(i)))
88                  {
89                      ++x->cnt(i);
90                      while (x)
91                          ++x->size, x = x->p;
92                      return;
93                  }
94              }
95              x = x->child[i];
96          }
97      }
98
99      void merge(Node *x, int i) //将 x 的 i 孩子与 i+1 孩子合并，用 x 的 i 键作为分隔，这两个孩子都只有 BF-1 个孩子，合并后有
100     {
101         Node *y = x->child[i], *z = x->child[i + 1];
102         y->keyNum = 2 * BF - 1;
103         y->values[BF - 1] = std::move(x->values[i]);
104         memmove(y->values + BF, z->values, (BF - 1) * sizeof(value_type));
105         if (!y->isLeaf)
```

```
106         {
107             memmove(y->child + BF, z->child, BF * sizeof(Node *));
108             for (int j = BF; j <= 2 * BF - 1; ++j)
109                 y->child[j]->p = y;
110         }
111         shiftBy(x->values + i + 1, x->keyNum - i - 1, -1);
112         shiftBy(x->child + i + 2, x->keyNum - i - 1, -1);
113 
114         --x->keyNum;
115         y->size = calcSize(y);
116     }
117     void erase(Node *x, const K &key)
118     {
119         int i = lower_bound(x->values, x->values + x->keyNum, key, pairComp) - x->values;
120         if (i != x->keyNum && !(key < x->values[i].first)) //找到 key 了
121         {
122             if (x->cnt(i) > 1)
123             {
124                 --x->cnt(i);
125                 while (x)
126                     --x->size, x = x->p;
127                 return;
128             }
129             if (x->isLeaf) //x 是叶节点，直接删除
130             {
131                 shiftBy(x->values + i + 1, --x->keyNum - i, -1); //需要移动的内存是 x->keyNum-i-1
132                 while (x)
133                     --x->size, x = x->p;
134             }
135             else
136             {
137                 if (x->child[i]->keyNum >= BF) //前驱所在孩子有足够的孩子 (以应对它的孩子的需求)
138                 {
139                     Node *y = x->child[i];
140                     while (!y->isLeaf)
141                         y = y->child[y->keyNum]; //找前驱
142                     x->values[i] = y->values[y->keyNum - 1];
143                     if (x->cnt(i) != 1) //y 的对应节点 cnt 有多个，那么沿路减 size; 只有一个的话删除的时候会处理
144                     {
145                         y->cnt(y->keyNum - 1) = 1;
146                         while (y != x)
147                             y->size -= x->cnt(i) - 1, y = y->p;
148                     }
149 
150                     erase(x->child[i], x->key(i));
151                 }
152                 else if (x->child[i + 1]->keyNum >= BF) //后继所在孩子有足够的孩子
153                 {
154                     Node *y = x->child[i + 1];
155                     while (!y->isLeaf)
156                         y = y->child[0]; //找后继
157                     x->values[i] = y->values[0];
158                     if (x->cnt(i) != 1)
159                     {
160                         y->cnt(0) = 1;
161                         while (y != x)
162                             y->size -= x->cnt(i) - 1, y = y->p;
163                     }
164 
165                     erase(x->child[i + 1], x->key(i));
166                 }
167                 else //都没有，那么把这两个节点都合并到 y 中，并且挪动 x 的孩子和键
168                 {
169                     merge(x, i);
170                     if (root->keyNum == 0) //keyNum==0 只是没有键了，但是还可能有一个孩子，这时根变成这个孩子
```

```
                    root = x->child[i], root->p = nullptr;
                erase(x->child[i], key);
            }
        }
    }
    else if (!x->isLeaf) //没有找到 key, 只要保证 x->child[i]->keyNum 足够多即可无脑递归, 然而很难保证
    {
        if (x->child[i]->keyNum == BF - 1)
        {
            Node *y = x->child[i];
            if (i >= 1 && x->child[i - 1]->keyNum >= BF) //左兄弟, 取走它的最大孩子
            {
                //找相邻的兄弟借节点, 类似旋转操作, 把 x 的一个键移入要删的 key 所在孩子, 把它的兄弟的一个 key 和孩子移入 x
                //但是从左还是右借并不完全一样, 所以不能一概处理
                Node *z = x->child[i - 1];
                shiftBy(y->values, y->keyNum, 1);
                //是否需要考虑析构的问题? z 的 keyNum 已经减了, 不可能再去析构 z->values[z->keyNum - 1] 了
                //所以, value 的构造必须要用 new 不能用 =, 从而避开 = 的资源释放
                //但是 value 的移动似乎应该是 bitwise 的, 考虑 std::move
                new (y->values) value_type(std::move(x->values[i - 1]));
                new (x->values + i - 1) value_type(std::move(z->values[z->keyNum - 1]));
                if (!y->isLeaf)
                {
                    shiftBy(y->child, y->keyNum + 1, 1);
                    y->child[0] = z->child[z->keyNum], y->child[0]->p = y;
                }

                --z->keyNum, ++y->keyNum;
                y->size = calcSize(y), z->size = calcSize(z);
                erase(y, key);
            }
            else if (i < x->keyNum && x->child[i + 1]->keyNum >= BF) //右兄弟, 取走它的最小孩子
            {
                Node *z = x->child[i + 1];
                new (y->values + y->keyNum) value_type(std::move(x->values[i]));
                new (x->values + i) value_type(std::move(z->values[0]));
                if (!y->isLeaf) //y 和 z 深度一样, isLeaf 情况相同
                {
                    y->child[y->keyNum + 1] = z->child[0], y->child[y->keyNum + 1]->p = y;
                    shiftBy(z->child + 1, z->keyNum, -1);
                }
                shiftBy(z->values + 1, z->keyNum - 1, -1);

                --z->keyNum, ++y->keyNum;
                y->size = calcSize(y), z->size = calcSize(z);
                erase(y, key);
            }
            else //两个兄弟都没有节点借, 那么将它与随便左右哪个兄弟合并, 然而还是要特判一下
            {
                if (i != 0)
                    --i; //i==0 时, y 与 y+1 合并仍放于 y; 否则 y 与 y-1 合并放于 y-1
                y = x->child[i];
                merge(x, i);
                if (root->keyNum == 0)
                    root = y, root->p = nullptr;
                erase(y, key);
            }
        }
        else
            erase(x->child[i], key);
    }
}

public:
  BTree() : root(new Node) {}
```

```
236     void insert(const K &key)
237     {
238         //沿路向下分裂满节点，每次分裂成左右一半，孩子的中间 key 留在父亲节点中用于分隔两个新孩子
239         //insertEmpty 只保证了当前节点有空间（来容纳它的孩子的分裂），不保证 key 需要去的孩子节点也有空间
240         if (root->keyNum == 2 * BF - 1)
241         {
242             Node *x = new Node;
243             x->isLeaf = false, x->child[0] = root, x->size = root->size; //+1 操作由 insertEmpty 来做
244             root->p = x, root = x;
245             split(x, 0); //split 接受参数：node 的满子节点下标
246         }
247         insertEmpty(root, key);
248     }
249     void erase(const K &key) { erase(root, key); }
250     int next(const K &key)
251     {
252         Node *x = root;
253         int ret;
254         while (x)
255         {
256             int i = lower_bound(x->values, x->values + x->keyNum, key, pairComp) - x->values;
257             if (x->values[i].first == key)
258                 ++i;
259             if (i != x->keyNum)
260                 ret = x->values[i].first;
261             x = x->child[i];
262         }
263         return ret;
264     }
265     int prev(const K &key)
266     {
267         Node *x = root;
268         int ret;
269         while (x)
270         {
271             int i = lower_bound(x->values, x->values + x->keyNum, key, pairComp) - x->values;
272             if (i)
273                 ret = x->values[i - 1].first;
274             x = x->child[i];
275         }
276         return ret;
277     }
278     int rank(const K &key)
279     {
280         Node *x = root;
281         int ret = 0;
282         while (x)
283         {
284             if (x->key(x->keyNum - 1) < key)
285             {
286                 ret += x->size - getSize(x->child[x->keyNum]);
287                 x = x->child[x->keyNum];
288                 continue;
289             }
290             for (int i = 0; i < x->keyNum; ++i)
291             {
292                 if (x->key(i) < key)
293                     ret += getSize(x->child[i]) + x->cnt(i);
294                 else if (x->key(i) == key)
295                     return ret + getSize(x->child[i]) + 1;
296                 else
297                 {
298                     x = x->child[i];
299                     break;
300                 }
```

```
            }
        }
        return ret;
    }
    int kth(int k)
    {
        Node *x = root;
        while (true)
        {
            for (int i = 0; i <= x->keyNum; ++i)
            {
                //const int csz = getSize(x->child[i]) + (i == x->keyNum ? 1 : x->cnt(i));
                const int lb = getSize(x->child[i]) + 1, ub = getSize(x->child[i]) + (i == x->keyNum ? 1 : x->cnt(i));
                if (k >= lb && k <= ub)
                    return x->key(i);
                if (k < lb)
                {
                    x = x->child[i];
                    break;
                }
                k -= ub;
            }
        }
    }
};
```

1.3 CaptainMo

```
// Captain Mo
// 询问 [l, r] 内的元素是否互不相同
int Ans, ans[N];
int block_sz, block_id[N];
struct Query {
    int l, r, id;
    Query() {}
    Query(int _l, int _r, int _id) : l(_l), r(_r), id(_id) {}
    bool operator < (const Query& q) const {
        if (block_id[l] == block_id[q.l])
            return block_id[l] & 1 ? r < q.r : r > q.r;
        return block_id[l] < block_id[q.l];
    }
} Q[N];

int n, q, a[N];

int cnt[N], ge2;
inline void add(int p) {
    ++cnt[a[p]];
    if (cnt[a[p]] == 2) ++ge2;
}

inline void del(int p) {
    if (cnt[a[p]] == 2) --ge2;
    --cnt[a[p]];
}

void CaptainMo() {
    block_sz = sqrt(n);
    for (int i = 1; i <= n; ++i) block_id[i] = i / block_sz;
    sort(Q + 1, Q + 1 + q);

    int l = 1, r = 0;
    ge2 = 0;
    for (int i = 1; i <= q; ++i) {
```

```
        while(r < Q[i].r) ++r, add(r);
        while(l < Q[i].l) del(l), ++l;
        while(l > Q[i].l) --l, add(l);
        while(r > Q[i].r) del(r), --r;
        ans[Q[i].id] = (ge2 == 0);
    }
}
```

1.4 FenwickTree

```
template<typename T>
struct FenwickTree {
    int n;
    vector<T> c;
    FenwickTree(int _n) : n(_n), c(n + 1) {}
    inline int lb(int x) { return x & -x; }
    void add(int x, T d) { for (; x < n; x += lb(x)) c[x] += d; }
    T getsum(int x) { T r = 0; for (; x; x -= lb(x)) r += c[x]; return r; }
    T getsum(int l, int r) { return getsum(r) - getsum(l - 1); }
    T kth(int k) {
        T ans = 0, cnt = 0;
        for (int i = log2(n) + 1; i >= 0; --i) {
            ans += (1LL << i);
            if (ans >= n || cnt + c[ans] >= k) ans -= (1LL << i);
            else cnt += c[ans];
        }
        return ans + 1;
    }
};
```

1.5 LCT

```
namespace Backlight {

namespace LinkCutTree {
    using T = int;
    #define ls ch[x][0]
    #define rs ch[x][1]
    const int S = N;

    int tot, sz[S], rev[S], ch[S][2], fa[S];

    T v[S], sum[S];

    inline void init() { tot = 0; }

    inline void clear(int x) { ch[x][0] = ch[x][1] = fa[x] = sz[x] = rev[x] = sum[x] = v[x] = 0; }

    inline int get(int x) { return ch[fa[x]][1] == x; }

    inline int isroot(int x) { return ch[fa[x]][0] != x && ch[fa[x]][1] != x; }

    inline int newnode(T val) {
        ++tot;
        sz[tot] = 1;
        ch[tot][0] = ch[tot][1] = fa[tot] = rev[tot] = 0;
        sum[tot] = v[tot] = val;
        return tot;
    }

    inline void reverse(int x) {
        swap(ls, rs);
```

```
        rev[x] ^= 1;
    }

    inline void push_up(int x) {
        sz[x] = sz[ls] + 1 + sz[rs];
        sum[x] = sum[ls] ^ v[x] ^ sum[rs];
    }

    inline void push_down(int x) {
        if (rev[x]) {
            reverse(ls); reverse(rs);
            rev[x] = 0;
        }
    }

    inline void update(int x) {
        if(!isroot(x)) update(fa[x]);
        push_down(x);
    }

    inline void rotate(int x) {
        int f = fa[x], g = fa[f], i = get(x);
        if (!isroot(f)) ch[g][get(f)] = x;
        fa[x] = g;
        ch[f][i] = ch[x][i^1]; fa[ch[f][i]] = f;
        ch[x][i^1] = f; fa[f] = x;
        push_up(f); push_up(x);
    }

    inline void splay(int x) {
        update(x);
        for (; !isroot(x); rotate(x))
            if (!isroot(fa[x])) rotate(get(fa[x]) == get(x) ? fa[x] : x);
    }

    inline void access(int x) {
        for(int y = 0; x; y = x, x = fa[x]) splay(x), rs = y, push_up(x);
    }

    inline void makeroot(int x) {
        access(x);
        splay(x);
        reverse(x);
    }

    inline int findroot(int x) {
        access(x); splay(x);
        while(ls) push_down(x), x = ls;
        return x;
    }

    inline void link(int x, int y) {
        makeroot(x);
        if (findroot(y) != x) fa[x] = y;
    }

    inline void cut(int x, int y) {
        makeroot(x);
        if (findroot(y) == x && fa[x] == y && ch[y][0] == x && !ch[y][1]) {
            fa[x] = ch[y][0] = 0;
            push_up(y);
        }
    }

    inline void split(int x, int y) {
```

```
        makeroot(x);
        access(y);
        splay(y);
    }

    // x--y 路径上节点点权和
    inline int query(int x, int y) {
        split(x, y);
        return sum[y];
    }

} // namespace LinkCutTree
#define LCT LinkCutTree

} // namespace Backlight
```

1.6 LefitstTree

```
template <typename V>
struct LeftistForest {
  struct LeftistTree {
    V v;
    int dist;
    int l, r, rt;
  } t[N];
  LeftistTree& operator[](int x) { return t[x]; }
  void init(int n, V* a) {
    FOR(i, 1, n) {
      t[i].v = a[i];
      t[i].l = t[i].r = t[i].dist = 0;
      t[i].rt = i;
    }
  }
  int find(int x) { return t[x].rt == x ? x : t[x].rt = find(t[x].rt); }
  int merge(int x, int y) {
    if (!x) return y;
    if (!y) return x;
    if (t[x].v > t[y].v) swap(x, y);  // 小根堆
    t[x].r = merge(t[x].r, y);
    t[t[x].r].rt = x;
    if (t[t[x].l].dist < t[t[x].r].dist) swap(t[x].l, t[x].r);
    if (!t[x].r)
      t[x].dist = 0;
    else
      t[x].dist = t[t[x].r].dist + 1;
    return x;
  }
  V top(int x) {
    if (t[x].v == -1) return -1;
    x = find(x);
    return t[x].v;
  }
  void pop(int x) {
    if (t[x].v == -1) return;
    x = find(x);
    t[t[x].l].rt = t[x].l;
    t[t[x].r].rt = t[x].r;
    t[x].rt = merge(t[x].l, t[x].r);
    t[x].v = -1;
  }
};

int n, m, a[N];
void solve(int Case) {
```

```
  rd(n, m);
  FOR(i, 1, n) rd(a[i]);
  LeftistForest<int> T;
  T.init(n, a);

  int op, x, y;
  FOR(_, 1, m) {
    rd(op);
    debug(op);
    if (op == 1) {
      rd(x, y);
      if (T[x].v == -1 || T[y].v == -1) continue;
      x = T.find(x);
      y = T.find(y);
      if (x == y) continue;
      T[x].rt = T[y].rt = T.merge(x, y);
    } else {
      rd(x);
      pln(T.top(x));
      T.pop(x);
    }
  }
}
```

1.7 PersistentSegmentTree

```
namespace Backlight {

namespace PersistentSegmentTree {
    using T = ll;

    T s[N << 5];
    int tot, L[N << 5], R[N << 5];

    int update(int rt, int l, int r, int p) {
        int nrt = ++tot;
        L[nrt] = L[rt]; R[nrt] = R[rt]; s[nrt] = s[rt] + 1;
        if (l != r) {
            int mid = (l + r) >> 1;
            if (p <= mid) L[nrt] = update(L[rt], l, mid, p);
            else R[nrt] = update(R[rt], mid + 1, r, p);
        }
        return nrt;
    }

    // 区间第 k 小
    int query(int u, int v, int l, int r, int k) {
        if (l == r) return l;
        int lsum = s[L[v]] - s[L[u]], mid = (l + r) >> 1;;
        if (k <= lsum) return query(L[u], L[v], l, mid, k);
        return query(R[u], R[v], mid + 1, r, k - lsum);
    }
} // namespace PersistentSegmentTree
#define PST PersistentSegmentTree

} // namespace Backlight
```

1.8 rbtree-1

```
//#define __REDBLACK_DEBUG
template <typename T>
class rbtree {
```

```
#define bro(x) (((x)->ftr->lc == (x)) ? ((x)->ftr->rc) : ((x)->ftr->lc))
#define islc(x) ((x) != NULL && (x)->ftr->lc == (x))
#define isrc(x) ((x) != NULL && (x)->ftr->rc == (x))
private:
    struct Node;

    Node* _root;
    Node* _hot;

    void init(T);
    void checkconnect(Node*);
    void connect34(Node*, Node*, Node*, Node*, Node*, Node*, Node*);
    void SolveDoubleRed(Node*);
    void SolveDoubleBlack(Node*);
    Node* find(T, const int);
    Node* rfind(T, const int);
    Node* findkth(int, Node*);
    int find_rank(T, Node*);
#ifdef __REDBLACK_DEBUG
    void previs(Node*, int);
    void invis(Node*, int);
    void postvis(Node*, int);
#endif

public:
    struct iterator;

    rbtree()
        : _root(NULL)
        , _hot(NULL)
    {
    }

    int get_rank(T);
    iterator insert(T);
    bool remove(T);
    int size();
    iterator kth(int);
    iterator lower_bound(T);
    iterator upper_bound(T);
#ifdef __REDBLACK_DEBUG
    void vis();
    void correctlyconnected();
#endif
};

template <typename T>
struct rbtree<T>::Node {
    T val;
    bool RBc; ////true : Red ; false : Black .
    Node* ftr;
    Node* lc;
    Node* rc;
    int s;

    Node(T v = T(), bool RB = true,
        Node* f = NULL, Node* l = NULL, Node* r = NULL, int ss = 1)
        : val(v)
        , RBc(RB)
        , ftr(f)
        , lc(l)
        , rc(r)
        , s(ss)
    {
    }
```

```

    Node* succ()
    {
        Node* ptn = rc;
        while (ptn->lc != NULL) {
            --(ptn->s);
            ptn = ptn->lc;
        }
        return ptn;
    }

    Node* left_node()
    {
        Node* ptn = this;
        if (!lc) {
            while (ptn->ftr && ptn->ftr->lc == ptn)
                ptn = ptn->ftr;
            ptn = ptn->ftr;
        } else
            while (ptn->lc)
                ptn = ptn->lc;
        return ptn;
    }

    Node* right_node()
    {
        Node* ptn = this;
        if (!rc) {
            while (ptn->ftr && ptn->ftr->rc == ptn)
                ptn = ptn->ftr;
            ptn = ptn->ftr;
        } else
            while (ptn->rc)
                ptn = ptn->rc;
        return ptn;
    }

    void maintain()
    {
        s = 1;
        if (lc)
            s += lc->s;
        if (rc)
            s += rc->s;
    }
};

template <typename T>
void rbtree<T>::connect34(Node* nroot, Node* nlc, Node* nrc,
    Node* ntree1, Node* ntree2, Node* ntree3, Node* ntree4)
{
    nlc->lc = ntree1;
    if (ntree1 != NULL)
        ntree1->ftr = nlc;
    nlc->rc = ntree2;
    if (ntree2 != NULL)
        ntree2->ftr = nlc;
    nrc->lc = ntree3;
    if (ntree3 != NULL)
        ntree3->ftr = nrc;
    nrc->rc = ntree4;
    if (ntree4 != NULL)
        ntree4->ftr = nrc;
    nroot->lc = nlc;
    nlc->ftr = nroot;
```

```
    nroot->rc = nrc;
    nrc->ftr = nroot;
    nlc->maintain();
    nrc->maintain();
    nroot->maintain();
}

#ifdef __REDBLACK_DEBUG

int blackheight(0);

template <typename T>
void rbtree<T>::previs(Node* ptn, int cnt)
{
    if (ptn == NULL) {
        if (blackheight == -1)
            blackheight = cnt;
        assert(blackheight == cnt);
        return;
    }
    printf("%d %s %d \n", ptn->val, ptn->RBc ? "Red" : "Black", ptn->s);
    if (!(ptn->RBc))
        ++cnt;
    previs(ptn->lc, cnt);
    previs(ptn->rc, cnt);
}

template <typename T>
void rbtree<T>::invis(Node* ptn, int cnt)
{
    if (ptn == NULL) {
        if (blackheight == -1)
            blackheight = cnt;
        assert(blackheight == cnt);
        return;
    }
    if (!(ptn->RBc))
        ++cnt;
    invis(ptn->lc, cnt);
    printf("%d %s %d \n", ptn->val, ptn->RBc ? "Red" : "Black", ptn->s);
    invis(ptn->rc, cnt);
}

template <typename T>
void rbtree<T>::postvis(Node* ptn, int cnt)
{
    if (ptn == NULL) {
        if (blackheight == -1)
            blackheight = cnt;
        assert(blackheight == cnt);
        return;
    }
    if (!(ptn->RBc))
        ++cnt;
    postvis(ptn->lc, cnt);
    postvis(ptn->rc, cnt);
    printf("%d %s %d \n", ptn->val, ptn->RBc ? "Red" : "Black", ptn->s);
}

template <typename T>
void rbtree<T>::vis()
{
    printf("BlackHeight:\t%d\n", blackheight);
    printf("------pre-vis------\n");
    previs(_root, 0);
```

```
199     printf("------in-vis------\n");
200     invis(_root, 0);
201     printf("------post-vis------\n");
202     postvis(_root, 0);
203 }
204 
205 template <typename T>
206 void rbtree<T>::checkconnect(Node* ptn)
207 {
208     if (!ptn)
209         return;
210     assert(ptn->s > 0);
211     if (ptn->lc && ptn->lc->ftr != ptn) {
212         printf("Oops! %d has a lc %d, but it failed to point its ftr!\n", ptn->val, ptn->lc->val);
213     }
214     if (ptn->rc && ptn->rc->ftr != ptn) {
215         printf("Oops! %d has a rc %d, but it failed to point its ftr!\n", ptn->val, ptn->rc->val);
216     }
217     int sss = ptn->s;
218     if (ptn->lc)
219         sss -= ptn->lc->s;
220     if (ptn->rc)
221         sss -= ptn->rc->s;
222     if (sss - 1) {
223         printf("Fuck it! %d's size is %d, but the sum of its children's size is %d!\n", ptn->val, ptn->s, ptn->s - sss);
224     }
225     checkconnect(ptn->lc);
226     checkconnect(ptn->rc);
227 }
228 
229 template <typename T>
230 void rbtree<T>::correctlyconnected()
231 {
232     checkconnect(_root);
233 }
234 
235 #endif
236 
237 template <typename T>
238 void rbtree<T>::init(T v)
239 {
240     _root = new Node(v, false, NULL, NULL, NULL, 1);
241 #ifdef __REDBLACK_DEBUG
242     ++blackheight;
243 #endif
244 }
245 
246 template <typename T>
247 void rbtree<T>::SolveDoubleRed(Node* nn)
248 {
249     while ((!(nn->ftr)) || nn->ftr->RBc) {
250         if (nn == _root) {
251             _root->RBc = false;
252 #ifdef __REDBLACK_DEBUG
253             ++blackheight;
254 #endif
255             return;
256         }
257         Node* pftr = nn->ftr;
258         if (!(pftr->RBc))
259             return; ////No double-red
260         Node* uncle = bro(nn->ftr);
261         Node* grdftr = nn->ftr->ftr;
262         if (uncle != NULL && uncle->RBc) { ////RR-2
263             grdftr->RBc = true;
```

```
            uncle->RBc = false;
            pftr->RBc = false;
            nn = grdftr;
        } else { ////RR-1
            if (islc(pftr)) {
                if (islc(nn)) {
                    pftr->ftr = grdftr->ftr;
                    if (grdftr == _root)
                        _root = pftr;
                    else if (grdftr->ftr->lc == grdftr)
                        grdftr->ftr->lc = pftr;
                    else
                        grdftr->ftr->rc = pftr;
                    connect34(pftr, nn, grdftr, nn->lc, nn->rc, pftr->rc, uncle);
                    pftr->RBc = false;
                    grdftr->RBc = true;
                } else {
                    nn->ftr = grdftr->ftr;
                    if (grdftr == _root)
                        _root = nn;
                    else if (grdftr->ftr->lc == grdftr)
                        grdftr->ftr->lc = nn;
                    else
                        grdftr->ftr->rc = nn;
                    connect34(nn, pftr, grdftr, pftr->lc, nn->lc, nn->rc, uncle);
                    nn->RBc = false;
                    grdftr->RBc = true;
                }
            } else {
                if (islc(nn)) {
                    nn->ftr = grdftr->ftr;
                    if (grdftr == _root)
                        _root = nn;
                    else if (grdftr->ftr->lc == grdftr)
                        grdftr->ftr->lc = nn;
                    else
                        grdftr->ftr->rc = nn;
                    connect34(nn, grdftr, pftr, uncle, nn->lc, nn->rc, pftr->rc);
                    nn->RBc = false;
                    grdftr->RBc = true;
                } else {
                    pftr->ftr = grdftr->ftr;
                    if (grdftr == _root)
                        _root = pftr;
                    else if (grdftr->ftr->lc == grdftr)
                        grdftr->ftr->lc = pftr;
                    else
                        grdftr->ftr->rc = pftr;
                    connect34(pftr, grdftr, nn, uncle, pftr->lc, nn->lc, nn->rc);
                    pftr->RBc = false;
                    grdftr->RBc = true;
                }
            }
            return;
        }
    }
}

template <typename T>
void rbtree<T>::SolveDoubleBlack(Node* nn)
{
    while (nn != _root) {
        Node* pftr = nn->ftr;
        Node* bthr = bro(nn);
        if (bthr->RBc) { ////BB-1
```

```
            bthr->RBc = false;
            pftr->RBc = true;
            if (_root == pftr)
                _root = bthr;
            if (pftr->ftr) {
                if (pftr->ftr->lc == pftr)
                    pftr->ftr->lc = bthr;
                else
                    pftr->ftr->rc = bthr;
            }
            bthr->ftr = pftr->ftr;
            if (islc(nn)) {
                connect34(bthr, pftr, bthr->rc, nn, bthr->lc, bthr->rc->lc, bthr->rc->rc);
            } else {
                connect34(bthr, bthr->lc, pftr, bthr->lc->lc, bthr->lc->rc, bthr->rc, nn);
            }
            bthr = bro(nn);
            pftr = nn->ftr;
        }
        if (bthr->lc && bthr->lc->RBc) { ////BB-3
            bool oldRBc = pftr->RBc;
            pftr->RBc = false;
            if (pftr->lc == nn) {
                if (pftr->ftr) {
                    if (pftr->ftr->lc == pftr)
                        pftr->ftr->lc = bthr->lc;
                    else
                        pftr->ftr->rc = bthr->lc;
                }
                bthr->lc->ftr = pftr->ftr;
                if (_root == pftr)
                    _root = bthr->lc;
                connect34(bthr->lc, pftr, bthr, pftr->lc, bthr->lc->lc, bthr->lc->rc, bthr->rc);
            } else {
                bthr->lc->RBc = false;
                if (pftr->ftr) {
                    if (pftr->ftr->lc == pftr)
                        pftr->ftr->lc = bthr;
                    else
                        pftr->ftr->rc = bthr;
                }
                bthr->ftr = pftr->ftr;
                if (_root == pftr)
                    _root = bthr;
                connect34(bthr, bthr->lc, pftr, bthr->lc->lc, bthr->lc->rc, bthr->rc, pftr->rc);
            }
            pftr->ftr->RBc = oldRBc;
            return;
        } else if (bthr->rc && bthr->rc->RBc) { ////BB-3
            bool oldRBc = pftr->RBc;
            pftr->RBc = false;
            if (pftr->lc == nn) {
                bthr->rc->RBc = false;
                if (pftr->ftr) {
                    if (pftr->ftr->lc == pftr)
                        pftr->ftr->lc = bthr;
                    else
                        pftr->ftr->rc = bthr;
                }
                bthr->ftr = pftr->ftr;
                if (_root == pftr)
                    _root = bthr;
                connect34(bthr, pftr, bthr->rc, pftr->lc, bthr->lc, bthr->rc->lc, bthr->rc->rc);
            } else {
                if (pftr->ftr) {
```

```
                    if (pftr->ftr->lc == pftr)
                        pftr->ftr->lc = bthr->rc;
                    else
                        pftr->ftr->rc = bthr->rc;
                }
                bthr->rc->ftr = pftr->ftr;
                if (_root == pftr)
                    _root = bthr->rc;
                connect34(bthr->rc, bthr, pftr, bthr->lc, bthr->rc->lc, bthr->rc->rc, pftr->rc);
            }
            pftr->ftr->RBc = oldRBc;
            return;
        }
        if (pftr->RBc) { ////BB-2R
            pftr->RBc = false;
            bthr->RBc = true;
            return;
        } else { ////BB-2B
            bthr->RBc = true;
            nn = pftr;
        }
    }
#ifdef __REDBLACK_DEBUG
    --blackheight;
#endif
}

template <typename T>
typename rbtree<T>::Node* rbtree<T>::findkth(int rank, Node* ptn)
{
    if (!(ptn->lc)) {
        if (rank == 1) {
            return ptn;
        } else {
            return findkth(rank - 1, ptn->rc);
        }
    } else {
        if (ptn->lc->s == rank - 1)
            return ptn;
        else if (ptn->lc->s >= rank)
            return findkth(rank, ptn->lc);
        else
            return findkth(rank - (ptn->lc->s) - 1, ptn->rc);
    }
}

template <typename T>
int rbtree<T>::find_rank(T v, Node* ptn)
{
    if (!ptn)
        return 1;
    else if (ptn->val >= v)
        return find_rank(v, ptn->lc);
    else
        return (1 + ((ptn->lc) ? (ptn->lc->s) : 0) + find_rank(v, ptn->rc));
}

template <typename T>
int rbtree<T>::get_rank(T v)
{
    return find_rank(v, _root);
}

template <typename T>
typename rbtree<T>::Node* rbtree<T>::find(T v, const int op)
```

```
{
    Node* ptn = _root;
    _hot = NULL;
    while (ptn != NULL) {
        _hot = ptn;
        ptn->s += op;
        if (ptn->val > v)
            ptn = ptn->lc;
        else
            ptn = ptn->rc;
    }
    return ptn;
}

template <typename T>
typename rbtree<T>::Node* rbtree<T>::rfind(T v, const int op)
{
    Node* ptn = _root;
    _hot = NULL;
    while (ptn != NULL && ptn->val != v) {
        _hot = ptn;
        ptn->s += op;
        if (ptn->val > v)
            ptn = ptn->lc;
        else
            ptn = ptn->rc;
    }
    return ptn;
}

template <typename T>
struct rbtree<T>::iterator {
private:
    Node* _real__node;

public:
    iterator& operator++()
    {
        _real__node = _real__node->right_node();
        return *this;
    }

    iterator& operator--()
    {
        _real__node = _real__node->left_node();
        return *this;
    }

    T operator*()
    {
        return _real__node->val;
    }

    iterator(Node* node_nn = NULL)
        : _real__node(node_nn)
    {
    }
    iterator(T const& val_vv)
        : _real__node(rfind(val_vv, 0))
    {
    }
    iterator(iterator const& iter)
        : _real__node(iter._real__node)
    {
    }
```

```
};

template <typename T>
typename rbtree<T>::iterator rbtree<T>::insert(T v)
{
    Node* ptn = find(v, 1);
    if (_hot == NULL) {
        init(v);
        return iterator(_root);
    }
    ptn = new Node(v, true, _hot, NULL, NULL, 1);
    if (_hot->val <= v)
        _hot->rc = ptn;
    else
        _hot->lc = ptn;
    SolveDoubleRed(ptn);
    return iterator(ptn);
}

template <typename T>
bool rbtree<T>::remove(T v)
{
    Node* ptn = rfind(v, -1);
    if (!ptn)
        return false;
    Node* node_suc;
    while (ptn->lc || ptn->rc) {
        if (!(ptn->lc)) {
            node_suc = ptn->rc;
        } else if (!(ptn->rc)) {
            node_suc = ptn->lc;
        } else {
            node_suc = ptn->succ();
        }
        --(ptn->s);
        ptn->val = node_suc->val;
        ptn = node_suc;
    }
    if (!(ptn->RBc)) {
        --(ptn->s);
        SolveDoubleBlack(ptn);
    }
    if (ptn->ftr->lc == ptn)
        ptn->ftr->lc = NULL;
    else
        ptn->ftr->rc = NULL;
    delete ptn;
    return true;
}

template <typename T>
int rbtree<T>::size()
{
    return _root->s;
}

template <typename T>
typename rbtree<T>::iterator rbtree<T>::kth(int rank)
{
    return iterator(findkth(rank, _root));
}

template <typename T>
typename rbtree<T>::iterator rbtree<T>::lower_bound(T v)
{
```

```
    Node* ptn = _root;
    while (ptn) {
        _hot = ptn;
        if (ptn->val < v) {
            ptn = ptn->rc;
        } else {
            ptn = ptn->lc;
        }
    }
    if (_hot->val < v) {
        ptn = _hot;
    } else {
        ptn = _hot->left_node();
    }
    return iterator(ptn);
}

template <typename T>
typename rbtree<T>::iterator rbtree<T>::upper_bound(T v)
{
    Node* ptn = _root;
    while (ptn) {
        _hot = ptn;
        if (ptn->val > v) {
            ptn = ptn->lc;
        } else {
            ptn = ptn->rc;
        }
    }
    if (_hot->val > v) {
        ptn = _hot;
    } else {
        ptn = _hot->right_node();
    }
    return iterator(ptn);
}
```

1.9 RBTree

```
template <typename T>
struct rbtree {
    struct node {
        T val;
        int sz, cnt;
        node *l, *r, *p;
        bool color;
    };
    node buf[N << 3], *s = buf;
    node* nil = ++s;
    node* root = nil;
    node* find_min(node* x)
    {
        while (x->l != nil)
            x = x->l;
        return x;
    }
    node* find_max(node* x)
    {
        while (x->r != nil)
            x = x->r;
        return x;
    }
    node* find_node(const T& val)
    {
```

```
        node* x = root;
        while (x != nil) {
            if (x->val == val)
                return x;
            if (x->val < val)
                x = x->r;
            else
                x = x->l;
        }
        return NULL;
    }
    void zig(node* x)
    {
        node* y = x->r;
        x->r = y->l;
        if (y->l != nil)
            y->l->p = x;
        y->p = x->p;
        if (x->p == nil)
            root = y;
        else if (x == x->p->r)
            x->p->r = y;
        else
            x->p->l = y;
        y->l = x;
        x->p = y;
        y->sz = x->sz;
        x->sz = x->l->sz + x->r->sz + x->cnt;
        return;
    }
    void zag(node* x)
    {
        node* y = x->l;
        x->l = y->r;
        if (y->r != nil)
            y->r->p = x;
        y->p = x->p;
        if (x->p == nil)
            root = y;
        else if (x == x->p->l)
            x->p->l = y;
        else
            x->p->r = y;
        y->r = x;
        x->p = y;
        y->sz = x->sz;
        x->sz = x->l->sz + x->r->sz + x->cnt;
        return;
    }
    void insert_fixup(node* z)
    {
        while (z->p->color == 1) {
            if (z->p == z->p->p->l) {
                node* y = z->p->p->r;
                if (y->color == 1) {
                    y->color = z->p->color = 0;
                    z->p->p->color = 1;
                    z = z->p->p;
                } else {
                    if (z == z->p->r) {
                        z = z->p;
                        zig(z);
                    }
                    z->p->color = 0;
                    z->p->p->color = 1;
```

```
                zag(z->p->p);
            }
        } else {
            node* y = z->p->p->l;
            if (y->color == 1) {
                y->color = z->p->color = 0;
                z->p->p->color = 1;
                z = z->p->p;
            } else {
                if (z == z->p->l) {
                    z = z->p;
                    zag(z);
                }
                z->p->color = 0;
                z->p->p->color = 1;
                zig(z->p->p);
            }
        }
    }
    root->color = 0;
    return;
}
void transplant(node* x, node* y)
{
    y->p = x->p;
    if (x->p == nil)
        root = y;
    else if (x == x->p->l)
        x->p->l = y;
    else
        x->p->r = y;
    return;
}
void delete_fixup(node* x)
{
    while (x != root && x->color == 0) {
        if (x == x->p->l) {
            node* w = x->p->r;
            if (w->color == 1) {
                x->p->color = 1;
                w->color = 0;
                zig(x->p);
                w = x->p->r;
            }
            if (w->l->color == 0 && w->r->color == 0) {
                w->color = 1;
                x = x->p;
            } else {
                if (w->r->color == 0) {
                    w->color = 1;
                    w->l->color = 0;
                    zag(w);
                    w = x->p->r;
                }
                w->color = x->p->color;
                x->p->color = 0;
                w->r->color = 0;
                zig(w->p);
                x = root;
            }
        } else {
            node* w = x->p->l;
            if (w->color == 1) {
                x->p->color = 1;
                w->color = 0;
```

```
156                     zag(x->p);
157                     w = x->p->l;
158                 }
159                 if (w->r->color == 0 && w->l->color == 0) {
160                     w->color = 1;
161                     x = x->p;
162                 } else {
163                     if (w->l->color == 0) {
164                         w->color = 1;
165                         w->r->color = 0;
166                         zig(w);
167                         w = x->p->l;
168                     }
169                     w->color = x->p->color;
170                     x->p->color = 0;
171                     w->l->color = 0;
172                     zag(w->p);
173                     x = root;
174                 }
175             }
176         }
177         x->color = 0;
178         return;
179     }
180     void ins(const T& val)
181     {
182         node* x = root;
183         node* y = nil;
184         while (x != nil) {
185             y = x;
186             ++y->sz;
187             if (x->val == val) {
188                 ++x->cnt;
189                 return;
190             }
191             if (x->val < val)
192                 x = x->r;
193             else
194                 x = x->l;
195         }
196         node* z = ++s;
197         *z = (node) { val, 1, 1, nil, nil, y, 1 };
198         if (y == nil)
199             root = z;
200         else {
201             if (y->val < val)
202                 y->r = z;
203             else
204                 y->l = z;
205         }
206         insert_fixup(z);
207         return;
208     }
209     void del(const T& val)
210     {
211         node* z = root;
212         node* w = nil;
213         while (z != nil) {
214             w = z;
215             --w->sz;
216             if (z->val == val)
217                 break;
218             if (z->val < val)
219                 z = z->r;
220             else
```

```
                z = z->l;
        }
        if (z != nil) {
            // delete only one node
            if (z->cnt > 1) {
                --z->cnt;
                return;
            }

            node* y = z;
            node* x;
            bool history = y->color;
            if (z->l == nil) {
                x = z->r;
                transplant(z, z->r);
            } else if (z->r == nil) {
                x = z->l;
                transplant(z, z->l);
            } else {
                y = find_min(z->r);
                history = y->color;
                x = y->r;
                if (y->p == z)
                    x->p = y;
                else {
                    node* w = y;
                    while (w != z) {
                        w->sz -= y->cnt;
                        w = w->p;
                    }
                    transplant(y, y->r);
                    y->r = z->r;
                    y->r->p = y;
                }
                transplant(z, y);
                y->l = z->l;
                y->l->p = y;
                y->color = z->color;
                y->sz = y->l->sz + y->r->sz + y->cnt;
            }
            if (history == 0)
                delete_fixup(x);
        } else
            while (w != nil) {
                ++w->sz;
                w = w->p;
            }
        return;
    }
    T getKth(int k)
    {
        T res = 0;
        node* x = root;
        while (x != nil) {
            if (x->l->sz + 1 <= k && x->l->sz + x->cnt >= k) {
                res = x->val;
                break;
            } else if (x->l->sz + x->cnt < k) {
                k -= x->l->sz + x->cnt;
                x = x->r;
            } else {
                x = x->l;
            }
        }
        return res;
```

```
    }
    int getRank(const T& val)
    {
        int rk = 0;
        node* x = root;
        while (x != nil) {
            if (x->val < val) {
                rk += x->l->sz + x->cnt;
                x = x->r;
            } else {
                if (x->val == val)
                    ++rk;
                x = x->l;
            }
        }
        return rk;
    }
    T getSucc(const T& val)
    {
        ins(val);
        T res = INT_MAX;
        node* x = find_node(val);
        if (x->r != nil) {
            res = find_min(x->r)->val;
        } else {
            while (x->p->r == x)
                x = x->p;
            if (x->p != nil)
                res = x->p->val;
        }
        del(val);
        return res;
    }
    T getPrev(const T& val)
    {
        ins(val);
        T res = INT_MIN;
        node* x = find_node(val);
        if (x->l != nil)
            res = find_max(x->l)->val;
        else {
            while (x->p->l == x)
                x = x->p;
            if (x->p != nil)
                res = x->p->val;
        }
        del(val);
        return res;
    }
};
```

1.10 RMQ

```
const int LG = log2(N) + 1;
int mi[N][LG], lg[N];
void init_rmq(int n) {
    lg[1] = 0;
    for (int i = 2; i <= n; ++i) lg[i] = lg[i >> 1] + 1;
}

void build_rmq(int n, int* a) {
    for (int i = 1; i <= n; ++i) mi[i][0] = a[i];
    for (int j = 1; j <= lg[n]; ++j) {
        for (int i = 1; i + (1 << j) <= n; ++i) {
```

```
            mi[i][j] = min(mi[i][j - 1], mi[i + (1 << (j - 1))][j - 1]);
        }
    }
}

int rmqMin(int l, int r) {
    int k = lg[r - l + 1];
    return min(mi[l][k], mi[r - (1 << k) + 1][k]);
}
```

1.11 RollBackCaptainMo

```
// Roll Back Captain Mo
// 询问 [l, r] 内值相同的元素的最远距离
int Ans, ans[N];
int block_sz, block_cnt, block_id[N], L[N], R[N];
struct Query {
    int l, r, id;
    Query() {}
    Query(int _l, int _r, int _id) : l(_l), r(_r), id(_id) {}
    bool operator < (const Query& q) const {
        if (block_id[l] == block_id[q.l]) return r < q.r;
        return block_id[l] < block_id[q.l];
    }
} Q[N];

int n, m, q, a[N], b[N];


int nums[N], cn;
int mi[N], ma[N];
int __mi[N];

int brute_force(int l, int r) {
    int res = 0;
    for (int i = l; i <= r; ++i) __mi[a[i]]= 0;
    for (int i = l; i <= r; ++i) {
        if (__mi[a[i]]) res = max(res, i - __mi[a[i]]);
        else __mi[a[i]] = i;
    }
    return res;
}

inline void addl(int p) {
    if (ma[a[p]]) Ans = max(Ans, ma[a[p]] - p);
    else ma[a[p]] = p;
}

inline void addr(int p) {
    ma[a[p]] = p;
    if (!mi[a[p]]) mi[a[p]] = p, nums[++cn] = a[p];
    Ans = max(Ans, p - mi[a[p]]);
}

inline void dell(int p) {
    if (ma[a[p]] == p) ma[a[p]] = 0;
}

inline void delr(int p) {

}

inline void clear() {
    for (int i = 1; i <= cn; ++i) mi[nums[i]] = ma[nums[i]] = 0;
```

```
}

void RollBackCaptainMo() {
    block_sz = sqrt(n); block_cnt = n / block_sz;

    for (int i = 1; i <= block_cnt; ++i) L[i] = R[i - 1] + 1, R[i] = i * block_sz;
    if (R[block_cnt] < n) { ++block_cnt; L[block_cnt] = R[block_cnt - 1] + 1; R[block_cnt] = n; }

    for (int i = 1; i <= block_cnt; ++i)
        for (int j = L[i]; j <= R[i]; ++j)
            block_id[j] = i;

    sort(Q + 1, Q + 1 + q);

    for (int i = 1, j = 1; j <= block_cnt; ++j) {
        int l = R[j] + 1, r = R[j];
        Ans = 0; cn = 0;
        for (; block_id[Q[i].l] == j; ++i) {
            if (block_id[Q[i].l] == block_id[Q[i].r]) ans[Q[i].id] = brute_force(Q[i].l, Q[i].r);
            else {
                while(r < Q[i].r) ++r, addr(r);
                int tmp = Ans;
                while(l > Q[i].l) --l, addl(l);
                ans[Q[i].id] = Ans;
                while(l <= R[j]) dell(l), ++l;
                Ans = tmp;
            }
        }
        clear();
    }
}
```

1.12 SegmentTree

```
class segtree {
public:
    struct node {
        // 声明变量，记得设置初始值
        // ie. 最大值: int mx = INT_MIN;

        ...

        void apply(int l, int r, ll addv) {
            // 更新节点信息
            // ie. 最大值 + 区间加: mx = mx + addv

            ...
        }
    };

    friend node operator + (const node& tl, const node& tr) {
        node t;
        // 合并两个区间的信息
        // ie. 区间和: t.sum = t1.sum + t2.sum;

        ...

        return t;
    }

    inline void push_down(int x, int l, int r) {
        int lc = x << 1, rc = lc | 1, mid = (l + r) >> 1;
        // 标记下传
        // ie. 区间加法
```

```
        // if (tr[x].add != 0) {
        //     tr[lc].apply(l, mid, tr[x].add);
        //     tr[rc].apply(mid + 1, r, tr[x].add);
        //     tr[x].add = 0;
        // }

        ...
    }

    /*************************************************************************/
    inline void push_up(int x) {
        int lc = x << 1, rc = lc | 1;
        tr[x] = tr[lc] + tr[rc];
    }

    int n;
    vector<node> tr;

    void build(int x, int l, int r) {
        if (l == r) {
            return;
        }
        int lc = x << 1, rc = lc | 1, mid = (l + r) >> 1;
        build(lc, l, mid);
        build(rc, mid + 1, r);
        push_up(x);
    }

    template<class T>
    void build(int x, int l, int r, const vector<T>& arr){
        if (l == r) {
            tr[x].apply(l, r, arr[l]);
            return;
        }
        int lc = x << 1, rc = lc | 1, mid = (l + r) >> 1;
        build(lc, l, mid, arr);
        build(rc, mid + 1, r, arr);
        push_up(x);
    }

    template<class T>
    void build(int x, int l, int r, T* arr){
        if (l == r) {
            tr[x].apply(l, r, arr[l]);
            return;
        }
        int lc = x << 1, rc = lc | 1, mid = (l + r) >> 1;
        build(lc, l, mid);
        build(rc, mid + 1, r);
        push_up(x);
    }

    node get(int x, int l, int r, int L, int R) {
        if (L <= l && r <= R) {
            return tr[x];
        }
        push_down(x, l, r);
        int lc = x << 1, rc = lc | 1, mid = (l + r) >> 1;
        node res;
        if (R <= mid) res = get(lc, l, mid, L, R);
        else if (L > mid) res = get(rc, mid + 1, r, L, R);
        else res = get(lc, l, mid, L, mid) + get(rc, mid + 1, r, mid + 1, R);
        push_up(x);
        return res;
    }
```

```
96
97      template<class... T>
98      void upd(int x, int l, int r, int L, int R, const T&... v) {
99          if (L <= l && r <= R) {
100             tr[x].apply(l, r, v...);
101             return;
102         }
103         push_down(x, l, r);
104         int lc = x << 1, rc = lc | 1, mid = (l + r) >> 1;
105         node res;
106         if (L <= mid) upd(lc, l, mid, L, R, v...);
107         if (R > mid) upd(rc, mid + 1, r, L, R, v...);
108         push_up(x);
109     }
110
111     int __get_first(int x, int l, int r, const function<bool(const node&)> &f) {
112         if (l == r) {
113             return l;
114         }
115         int lc = x << 1, rc = lc | 1, mid = (l + r) >> 1;
116         push_down(x, l, r);
117         int res;
118         if (f(tr[lc])) res = __get_first(lc, l, mid, f);
119         else res = __get_first(rc, mid + 1, r, f);
120         push_up(x);
121         return res;
122     }
123
124     int get_first(int x, int l, int r, int L, int R, const function<bool(const node&)> &f) {
125         if (L <= l && r <= R) {
126             if (!f(tr[x])) {
127                 return -1;
128             }
129             return __get_first(x, l, r, f);
130         }
131         push_down(x, l, r);
132         int lc = x << 1, rc = lc | 1, mid = (l + r) >> 1;
133         int res;
134         if (L <= mid) res = get_first(lc, l, mid, L, R, f);
135         if (res == -1 && R > mid) res = get_first(rc, mid + 1, r, L, R, f);
136         push_up(x);
137         return res;
138     }
139
140     int __get_last(int x, int l, int r, const function<bool(const node&)> &f) {
141         if (l == r) {
142             return l;
143         }
144         int lc = x << 1, rc = lc | 1, mid = (l + r) >> 1;
145         push_down(x, l, r);
146         int res;
147         if (f(tr[lc])) res = __get_first(rc, mid + 1, r, f);
148         else res = __get_first(lc, l, mid, f);
149         push_up(x);
150         return res;
151     }
152
153     int get_last(int x, int l, int r, int L, int R, const function<bool(const node&)> &f) {
154         if (L <= l && r <= R) {
155             if (!f(tr[x])) {
156                 return -1;
157             }
158             return __get_first(x, l, r, f);
159         }
160         push_down(x, l, r);
```

```
        int lc = x << 1, rc = lc | 1, mid = (l + r) >> 1;
        int res;
        if (R > mid) res = get_last(rc, mid + 1, r, L, R, f);
        if (res == -1 && L <= mid) res = get_last(lc, l, mid, L, R, f);
        push_up(x);
        return res;
    }

    int find_first(int l, int r, const function<bool(const node&)> &f) {
        int L = l, R = r, mid, res = -1;
        while(L <= R) {
            mid = (L + R) >> 1;
            if (f(get(l, mid))) R = mid - 1, res = mid;
            else L = mid + 1;
        }
        return res;
    }

    int find_last(int l, int r, const function<bool(const node&)> &f) {
        int L = l, R = r, mid, res = -1;
        while(L <= R) {
            mid = (L + R) >> 1;
            if (f(get(l, mid))) L = mid + 1, res = mid;
            else R = mid - 1;
        }
        return res;
    }

    segtree(int _n) : n(_n) {
        assert(n > 0);
        tr.resize((n << 2) + 5);
        build(1, 1, n);
    }

    template<class T>
    segtree(const vector<T>& arr) {
        n = arr.size() - 1;
        assert(n > 0);
        tr.resize((n << 2) + 5);
        build(1, 1, n, arr);
    }

    template<class T>
    segtree(int _n, T* arr) {
        n = _n;
        assert(n > 0);
        tr.resize((n << 2) + 5);
        build(1, 1, n, arr);
    }

    node get(int l, int r) {
        assert(l >= 1 && l <= r && r <= n);
        return get(1, 1, n, l, r);
    }

    node get(int p) {
        assert(1 <= p && p <= n);
        return get(1, 1, n, p, p);
    }

    template <class... T>
    void upd(int l, int r, const T&... v) {
        assert(l >= 1 && l <= r && r <= n);
        upd(1, 1, n, l, r, v...);
    }
```

```

    template <class... T>
    void upd1(int p, const T&... v) {
        assert(p >= 1 && p <= n);
        upd(1, 1, n, p, p, v...);
    }

    int get_first(int l, int r, const function<bool(const node&)> &f) {
        assert(l >= 1 && l <= r && r <= n);
        return get_first(1, 1, n, l, r, f);
    }


    int get_last(int l, int r, const function<bool(const node&)> &f) {
        assert(l >= 1 && l <= r && r <= n);
        return get_last(1, 1, n, l, r, f);
    }

    void print(int x, int l, int r) {
        if (l == r) {
            cerr << tr[x].sum << " ";
            return;
        }
        push_down(x, l, r);
        int lc = x << 1, rc = lc | 1, mid = (l + r) >> 1;
        print(lc, l, mid);
        print(rc, mid + 1, r);
    }

    void print() {
        #ifdef BACKLIGHT
        cerr << "SEGTREE: " << endl;
        print(1, 1, n);
        cerr << "\n----------------------------" << endl;
        #endif
    }
};
```

1.13 SGTree

```
template<typename T>
struct SGTree {
    static constexpr double alpha = 0.75; // alpha \in (0.5, 1)
    int root, tot, buf_size;
    T v[N];
    int s[N], sz[N], sd[N], cnt[N], l[N], r[N], buf[N];


    SGTree()
    {
        root = tot = 0;
    }

    int new_node(T _v)
    {
        ++tot;
        v[tot] = _v;
        s[tot] = sz[tot] = sd[tot] = cnt[tot] = 1;
        l[tot] = r[tot] = 0;
        return tot;
    }

    void push_up(int x)
    {
```

```
        if (!x) return;
        int lc = l[x], rc = r[x];
        s[x] = s[lc] + 1 + s[rc];
        sz[x] = sz[lc] + cnt[x] + sz[rc];
        sd[x] = sd[lc] + (cnt[x] != 0) + sd[rc];
    }

    bool balance(int x)
    {
        int lc = l[x], rc = r[x];
        if (alpha * s[x] <= max(s[lc], s[rc])) return false;
        if (alpha * s[x] >= sd[x]) return false;
        return true;
    }

    void flatten(int x)
    {
        if (!x) return;
        flatten(l[x]);
        if (cnt[x]) buf[++buf_size] = x;
        flatten(r[x]);
    }

    void build(int& x, int L, int R)
    {
        if (L > R) {
            x = 0;
            return;
        }
        int mid = (L + R) >> 1;
        x = buf[mid];
        build(l[x], L, mid - 1);
        build(r[x], mid + 1, R);
        push_up(x);
    }

    void rebuild(int& x)
    {
        buf_size = 0;
        flatten(x);
        build(x, 1, buf_size);
    }

    void ins(int& rt, T val)
    {
        if (!rt) {
            rt = new_node(val);
            return;
        }
        if (val == v[rt]) {
            ++cnt[rt];
        } else if (val < v[rt]) {
            ins(l[rt], val);
        } else {
            ins(r[rt], val);
        }
        push_up(rt);
        if (!balance(rt)) rebuild(rt);
    }

    void del(int &rt, T val)
    {
        if (!rt) return;

        if (val == v[rt]) {
```

```
            if (cnt[rt]) --cnt[rt];
        } else if (val < v[rt]) {
            del(l[rt], val);
        } else {
            del(r[rt], val);
        }
        push_up(rt);
        if (!balance(rt)) rebuild(rt);
    }

    int getPrevRank(int rt, T val)
    {
        if (!rt) return 0;
        if (v[rt] == val && cnt[rt]) return sz[l[rt]];
        if (v[rt] < val) return sz[l[rt]] + cnt[rt] + getPrevRank(r[rt], val);
        return getPrevRank(l[rt], val);
    }

    int getSuccRank(int rt, T val)
    {
        if (!rt) return 1;
        if (v[rt] == val && cnt[rt]) return sz[l[rt]] + cnt[rt] + 1;
        if (v[rt] < val) return sz[l[rt]] + cnt[rt] + getSuccRank(r[rt], val);
        return getSuccRank(l[rt], val);
    }


    T getKth(int rt, int k)
    {
        if (!rt) return 0;
        if (k <= sz[l[rt]]) return getKth(l[rt], k);
        if (k - sz[l[rt]] <= cnt[rt]) return v[rt];
        return getKth(r[rt], k - sz[l[rt]] - cnt[rt]);
    }

    void ins(T val)
    {
        ins(root, val);
    }

    void del(T val)
    {
        del(root, val);
    }

    int getRank(T val)
    {
        return getPrevRank(root, val) + 1;
    }

    T getKth(int k)
    {
        return getKth(root, k);
    }

    T getPrev(T val)
    {
        return getKth(getPrevRank(root, val));
    }

    T getSucc(T val)
    {
        return getKth(getSuccRank(root, val));
    }

```

```
    void debug(int x)
    {
        if (!x) return;
        debug(l[x]);
        cerr << v[x] << " ";
        debug(r[x]);
    }

    void debug()
    {
        cerr << "SGTree:" << endl;
        debug(root);
        cerr << endl;
    }
};
```

1.14 Splay

```
namespace Backlight {

namespace Splay {
    using T = int;
    #define ls ch[x][0]
    #define rs ch[x][1]
    const int S = N;

    int tot, rt, sz[S], cnt[S], ch[S][2], fa[S];

    T v[S];

    inline void init() { tot = rt = 0; }

    inline void clear(int x) { ch[x][0] = ch[x][1] = fa[x] = sz[x] = cnt[x] = v[x] = 0; }

    inline int get(int x) { return ch[fa[x]][1] == x; }

    inline int newnode(T val) {
        ++tot;
        sz[tot] = cnt[tot] = 1;
        ch[tot][0] = ch[tot][1] = fa[tot] = 0;
        v[tot] = val;
        return tot;
    }

    inline void push_up(int x) {
        if (!x) return;
        sz[x] = sz[ls] + cnt[x] + sz[rs];
    }

    void rotate(int x) {
        int f = fa[x], g = fa[f], i = get(x);
        ch[f][i] = ch[x][i^1]; fa[ch[f][i]] = f;
        ch[x][i^1] = f; fa[f] = x;
        fa[x] = g;
        if (g) ch[g][ch[g][1] == f] = x;
        push_up(f); push_up(x);
    }

    void splay(int x, int ed) {
        for (int f; (f = fa[x]) != ed; rotate(x))
            if (fa[f] != ed) rotate((get(x) == get(f) ? f : x));
        if (ed == 0) rt = x;
    }

```

```

    void insert(T val) {
        if (rt == 0) { rt = newnode(val); return; }
        int p = rt, f = 0;
        while(true) {
            if (val == v[p]) {
                ++cnt[p];
                push_up(p); push_up(f);
                break;
            }
            f = p;
            p = ch[p][v[p] < val];
            if (p == 0) {
                p = newnode(val);
                fa[p] = f; ch[f][v[f] < val] = p;
                push_up(f);
                break;
            }
        }
        splay(p, 0);
    }

    int getrank(T val) {
        int p = rt, res = 0;
        while(p) {
            if (v[p] > val) p = ch[p][0];
            else {
                res += sz[ch[p][0]];
                if (v[p] == val) break;
                res += cnt[p];
                p = ch[p][1];
            }
        }
        assert(p != 0);
        splay(p, 0);
        return res + 1;
    }

    T getkth(int k) {
        int p = rt, res = 0;
        while(p) {
            if (k <= sz[ch[p][0]]) p = ch[p][0];
            else {
                if (k <= sz[ch[p][0]] + cnt[p]) { res = v[p]; break; }
                else k -= sz[ch[p][0]] + cnt[p], p = ch[p][1];
            }
        }
        assert(p != 0);
        splay(p, 0);
        return res;
    }

    void remove(T val) {
        getrank(val); // splay val to root
        if (cnt[rt] > 1) { --cnt[rt]; push_up(rt); return; }
        if (!ch[rt][0] && !ch[rt][1]) { clear(rt); rt = 0; return; }
        if (!ch[rt][0] || !ch[rt][1]) {
            int nrt = ch[rt][0] ? ch[rt][0] : ch[rt][1];
            clear(rt); rt = nrt; fa[rt] = 0;
            return;
        }
        int ort = rt;
        int p = ch[rt][0]; while(ch[p][1]) p = ch[p][1];
        splay(p, 0);
        ch[rt][1] = ch[ort][1];
```

```
        fa[ch[ort][1]] = rt;
        clear(ort);
        push_up(rt);
    }

    T getpre(T val) {
        int p = rt, res = -INF;
        while(p) {
            if (v[p] < val && v[p] > res) res = v[p];
            if (val > v[p]) p = ch[p][1];
            else p = ch[p][0];
        }
        // splay(p, 0);
        return res;
    }

    T getsuc(T val) {
        int p = rt, res = INF;
        while(p) {
            if (v[p] > val && v[p] < res) res = v[p];
            if (val < v[p]) p = ch[p][0];
            else p = ch[p][1];
        }
        // splay(p, 0);
        return res;
    }

    void DEBUG(int x) {
        if (!x) return;
        DEBUG(ls);
        cerr << v[x] << " ";
        DEBUG(rs);
    }

    void DEBUG() {
        cerr << "Splay: ";
        DEBUG(rt);
        cerr << endl;
    }
} // namespace Splay

} // namespace Backlight
```

1.15 Treap-dynamic

```
// mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
// inline unsigned rng() {
//     static unsigned x = 7;
//     return x = x * 0xdefaced + 1;
// }

template <typename T>
struct Treap {
    struct node {
        node *l, *r;
        unsigned rnd;
        T v;
        int sz;
        node(T _v)
            : l(NULL)
            , r(NULL)
            , rnd(rng())
            , sz(1)
            , v(_v)
```

```
20          {
21          }
22      };
23
24      inline int get_size(node*& p)
25      {
26          return p ? p->sz : 0;
27      }
28
29      inline void push_up(node*& p)
30      {
31          if (!p)
32              return;
33          p->sz = get_size(p->l) + get_size(p->r) + 1;
34      }
35
36      node* root = NULL;
37
38      node* merge(node* a, node* b)
39      {
40          if (!a)
41              return b;
42          if (!b)
43              return a;
44          if (a->rnd < b->rnd) {
45              a->r = merge(a->r, b);
46              push_up(a);
47              return a;
48          } else {
49              b->l = merge(a, b->l);
50              push_up(b);
51              return b;
52          }
53      }
54
55      void split_val(node* p, const T& k, node*& a, node*& b)
56      {
57          if (!p)
58              a = b = NULL;
59          else {
60              if (p->v <= k) {
61                  a = p;
62                  split_val(p->r, k, a->r, b);
63                  push_up(a);
64              } else {
65                  b = p;
66                  split_val(p->l, k, a, b->l);
67                  push_up(b);
68              }
69          }
70      }
71
72      void split_size(node* p, int k, node*& a, node*& b)
73      {
74          if (!p)
75              a = b = NULL;
76          else {
77              if (get_size(p->l) <= k) {
78                  a = p;
79                  split_size(p->r, k - get_size(p->l) - 1, a->r, b);
80                  push_up(a);
81              } else {
82                  b = p;
83                  split_size(p->l, k, a, b->l);
84                  push_up(b);
```

```
            }
        }
    }

    void ins(T val)
    {
        node *a, *b;
        split_val(root, val, a, b);
        a = merge(a, new node(val));
        root = merge(a, b);
    }

    void del(T val)
    {
        node *a, *b, *c, *d;
        split_val(root, val, a, b);
        split_val(a, val - 1, c, d);
        node* e = d;
        d = merge(d->l, d->r);
        delete e;
        a = merge(c, d);
        root = merge(a, b);
    }

    T getRank(T val)
    {
        node *a, *b;
        split_val(root, val - 1, a, b);
        T res = get_size(a) + 1;
        root = merge(a, b);
        return res;
    }

    T getKth(int k)
    {
        node* x = root;
        T res = numeric_limits<T>::min();
        while (x) {
            if (k <= get_size(x->l))
                x = x->l;
            else {
                if (get_size(x->l) + 1 == k) {
                    res = x->v;
                    break;
                } else {
                    k -= get_size(x->l) + 1;
                    x = x->r;
                }
            }
        }
        return res;
    }

    T getPrev(T val)
    {
        node *a, *b;
        split_val(root, val - 1, a, b);
        node* p = a;
        while (p->r)
            p = p->r;
        root = merge(a, b);
        return p->v;
    }

    T getSucc(T val)
```

```
    {
        node *a, *b;
        split_val(root, val, a, b);
        node* p = b;
        while (p->l)
            p = p->l;
        root = merge(a, b);
        return p->v;
    }
};
```

1.16 Treap-pointer

```
// mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
// inline unsigned rng() {
//     static unsigned x = 7;
//     return x = x * 0xdefaced + 1;
// }

template <typename T>
struct Treap {
    struct node {
        node *l, *r;
        unsigned rnd;
        T v;
        int sz;
        node(T _v)
            : l(NULL)
            , r(NULL)
            , rnd(rng())
            , sz(1)
            , v(_v)
        {
        }
    };

    inline int get_size(node*& p)
    {
        return p ? p->sz : 0;
    }

    inline void push_up(node*& p)
    {
        if (!p)
            return;
        p->sz = get_size(p->l) + get_size(p->r) + 1;
    }

    node* root = NULL;

    node* merge(node* a, node* b)
    {
        if (!a)
            return b;
        if (!b)
            return a;
        if (a->rnd < b->rnd) {
            a->r = merge(a->r, b);
            push_up(a);
            return a;
        } else {
            b->l = merge(a, b->l);
            push_up(b);
            return b;
```

```
        }
    }

    void split_val(node* p, const T& k, node*& a, node*& b)
    {
        if (!p)
            a = b = NULL;
        else {
            if (p->v <= k) {
                a = p;
                split_val(p->r, k, a->r, b);
                push_up(a);
            } else {
                b = p;
                split_val(p->l, k, a, b->l);
                push_up(b);
            }
        }
    }

    void split_size(node* p, int k, node*& a, node*& b)
    {
        if (!p)
            a = b = NULL;
        else {
            if (get_size(p->l) <= k) {
                a = p;
                split_size(p->r, k - get_size(p->l) - 1, a->r, b);
                push_up(a);
            } else {
                b = p;
                split_size(p->l, k, a, b->l);
                push_up(b);
            }
        }
    }

    void ins(T val)
    {
        node *a, *b;
        split_val(root, val, a, b);
        a = merge(a, new node(val));
        root = merge(a, b);
    }

    void del(T val)
    {
        node *a, *b, *c, *d;
        split_val(root, val, a, b);
        split_val(a, val - 1, c, d);
        node* e = d;
        d = merge(d->l, d->r);
        delete e;
        a = merge(c, d);
        root = merge(a, b);
    }

    T getRank(T val)
    {
        node *a, *b;
        split_val(root, val - 1, a, b);
        T res = get_size(a) + 1;
        root = merge(a, b);
        return res;
    }
```

```

    T getKth(int k)
    {
        node* x = root;
        T res = numeric_limits<T>::min();
        while (x) {
            if (k <= get_size(x->l))
                x = x->l;
            else {
                if (get_size(x->l) + 1 == k) {
                    res = x->v;
                    break;
                } else {
                    k -= get_size(x->l) + 1;
                    x = x->r;
                }
            }
        }
        return res;
    }

    T getPrev(T val)
    {
        node *a, *b;
        split_val(root, val - 1, a, b);
        node* p = a;
        while (p->r)
            p = p->r;
        root = merge(a, b);
        return p->v;
    }

    T getSucc(T val)
    {
        node *a, *b;
        split_val(root, val, a, b);
        node* p = b;
        while (p->l)
            p = p->l;
        root = merge(a, b);
        return p->v;
    }
};
```

1.17 Treap

```
namespace Backlight {

namespace Treap {
    using T = long long;
    const int S = N;
    mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());

    int tot, rt, sz[S], L[S], R[S], rnd[S];

    T v[S];

    inline void init() {
        tot = rt = 0;
    }

    inline int newnode(T val) {
        ++tot;
        sz[tot] = 1;
```

```
        L[tot] = R[tot] = 0;
        rnd[tot] = rng();
        v[tot] = val;
        return tot;
    }

    inline void push_up(int x) {
        sz[x] = sz[L[x]] + 1 + sz[R[x]];
    }

    void split(int u, T k, int &x, int &y) {
        if (!u) x = y = 0;
        else {
            if (v[u] <= k) {
                x = u;
                split(R[u], k, R[u], y);
            } else {
                y = u;
                split(L[u], k, x, L[u]);
            }
            push_up(u);
        }
    }

    int merge(int x, int y) {
        if (!x || !y) return x | y;
        if (rnd[x] < rnd[y]) {
            R[x] = merge(R[x], y);
            push_up(x);
            return x;
        } else {
            L[y] = merge(x, L[y]);
            push_up(y);
            return y;
        }
    }

    void insert(T val) {
        int x, y;
        split(rt, val, x, y);
        x = merge(x, newnode(val));
        rt = merge(x, y);
    }

    void remove(T val) {
        int x1, y1, x2, y2;
        split(rt, val, x1, y1);
        split(x1, val - 1, x2, y2);
        y2 = merge(L[y2], R[y2]);
        x1 = merge(x2, y2);
        rt = merge(x1, y1);
    }

    int getrank(T val) {
        int x, y;
        split(rt, val - 1, x, y);
        int res = sz[x] + 1;
        rt = merge(x, y);
        return res;
    }

    T getkth(int k) {
        int u = rt;
        while(true) {
            if (k <= sz[L[u]]) u = L[u];
```

```
            else {
                if (sz[L[u]] + 1 == k) break;
                else k -= sz[L[u]] + 1, u = R[u];
            }
        }
        return v[u];
    }

    T getpre(T val) {
        int x, y;
        split(rt, val - 1, x, y);
        int p = x;
        while(R[p]) p = R[p];
        rt = merge(x, y);
        return v[p];
    }

    T getsuc(T val) {
        int x, y;
        split(rt, val, x, y);
        int p = y;
        while(L[p]) p = L[p];
        rt = merge(x, y);
        return v[p];
    }

    void DEBUG(int u) {
        if (!u) return;
        DEBUG(L[u]);
        cerr << v[u] << " ";
        DEBUG(R[u]);
    }

    void DEBUG() {
        cerr << "Treap: ";
        DEBUG(rt);
        cerr << endl;
    }
} // namespace Treap

} // namespace Backlight
```

2 graph

2.1 BCC-Edge

```
namespace Backlight {

struct Graph {
    #define fore(i, u) for (int i = h[u]; i; i = e[i].nxt)
    struct Edge {
        int v, nxt;
        Edge(){}
        Edge(int _v, int _nxt): v(_v), nxt(_nxt) {}
    };

    int V, E, tot;
    vector<int> h;
    vector<Edge> e;

    Graph() : V(0) {}
    Graph(int _V, int _E) : V(_V), E(2 * _E), tot(0), h(_V + 1), e(2 * _E + 1) { }

```

```
    inline void addarc(int u, int v) {
        assert(1 <= u && u <= V);
        assert(1 <= v && v <= V);

        e[++tot] = Edge(v, h[u]); h[u] = tot;
    }

    inline void addedge(int u, int v) {
        addarc(u, v);
        addarc(v, u);
    }

    /************************************************/
    int bcc_clock, bcc_cnt;
    vector<int> dfn, low, belong, bcc_size;
    vector<vector<int>> bcc;
    vector<bool> bridge;

    void tarjan(int u, int fa) {
        dfn[u] = low[u] = ++bcc_clock;
        fore(i, u) {
            int v = e[i].v;
            if (v == fa) continue;

            if (!dfn[v]) {
                tarjan(v, u);
                low[u] = min(low[u], low[v]);
                if (dfn[u] < low[v]) {
                    bridge[i] = true;
                    if (i & 1) bridge[i + 1] = true;
                    else bridge[i - 1] = true;
                }
            } else if (dfn[v] < dfn[u]) {
                low[u] = min(low[u], dfn[v]);
            }
        }
    }

    void blood_fill(int u) {
        belong[u] = bcc_cnt; bcc[bcc_cnt].push_back(u);
        fore(i, u) {
            if (bridge[i]) continue;
            int v = e[i].v;
            if (!belong[v]) blood_fill(v);
        }
    }

    void build_bcc_point() {
        bcc_clock = bcc_cnt = 0;
        dfn = vector<int>(V + 1);
        low = vector<int>(V + 1);
        belong = vector<int>(V + 1);
        bridge = vector<bool>(E + 1);
        bcc = vector<vector<int>>(1);

        for (int i = 1; i <= V; ++i) {
            if (!dfn[i]) {
                tarjan(i, i);
            }
        }

        for (int i = 1; i <= V; ++i) {
            if (!belong[i]) {
                ++bcc_cnt;
                bcc.push_back(vector<int>());
```

```
                blood_fill(i);
            }
        }

        bcc_size = vector<int> (bcc_cnt + 1);
        for (int i = 1; i <= bcc_cnt; ++i) bcc_size[i] = bcc[i].size();
    }
};

}
```

2.2 BCC-Point

```
namespace Backlight {

struct Graph {
    struct Edge {
        int u, v;
        Edge(){}
        Edge(int _u, int _v): u(_u), v(_v) {}
    };

    int V;
    vector<vector<Edge>> G;

    Graph() : V(0) {}
    Graph(int _V) : V(_V), G(_V + 1) {}

    inline void addarc(int u, int v) {
        assert(1 <= u && u <= V);
        assert(1 <= v && v <= V);
        G[u].push_back(Edge(u, v));
    }

    inline void addedge(int u, int v) {
        addarc(u, v);
        addarc(v, u);
    }

    /*************************************************/
    int bcc_clock;
    vector<int> dfn, low;
    vector<vector<int>> bcc;
    vector<bool> cut;
    stack<int> stk;

    void tarjan(int u, int fa) {
        dfn[u] = low[u] = ++bcc_clock; stk.push(u);

        if (u == fa && G[u].empty()) {
            vector<int> nb;
            nb.push_back(u);
            bcc.push_back(nb);
            return;
        }

        int son = 0;
        for (Edge& e: G[u]) {
            int v = e.v;
            if (v == fa) continue;

            if (!dfn[v]) {
                tarjan(v, u);
                low[u] = min(low[u], low[v]);
```

```
                if (dfn[u] <= low[v]) {
                    ++son;
                    if (u != fa || son > 1) cut[u] = true;
                    vector<int> nb;
                    int top;
                    do {
                        top = stk.top(); stk.pop();
                        nb.push_back(top);
                    } while(top != v);
                    nb.push_back(u);
                    bcc.push_back(nb);
                }
            } else low[u] = min(low[u], dfn[v]);
        }
    }

    void build_bcc_point() {
        bcc_clock = 0;
        dfn = vector<int>(V + 1);
        low = vector<int>(V + 1);
        cut = vector<bool>(V + 1);
        bcc = vector<vector<int>>(1);

        for (int i = 1; i <= V; ++i) {
            if (!dfn[i]) {
                while(!stk.empty()) stk.pop();
                tarjan(i, i);
            }
        }
    }
};

}
```

2.3 BiGraphMatch

```
// Hopcroft Karp, O(\sqrt{V}E)
struct bigraph {
    int dfn;

    vector<vector<int>> G;

    int nl, nr;
    vector<int> ml, mr;
    vector<int> ll, lr;
    vector<int> vis;

    bigraph(int _nl, int _nr) {
        nl = _nl; nr = _nr;
        G = vector<vector<int>>(nl + 1);
    }

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

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

    bool bfs() {
        queue<int> q;
        bool res = false;
```

```

        for (int i = 1; i <= nl; ++i) {
            if (ml[i]) ll[i] = 0;
            else ll[i] = 1, q.push(i);
        }

        for (int i = 1; i <= nr; ++i) lr[i] = 0;

        while(!q.empty()) {
            int u = q.front(); q.pop();
            for (int v: G[u]) {
                if (lr[v] == 0) {
                    lr[v] = ll[u] + 1;
                    if (mr[v]) {
                        ll[mr[v]] = lr[v] + 1;
                        q.push(mr[v]);
                    } else res = true;
                }
            }
        }

        return res;
    };

    bool dfs(int u) {
        for (int v: G[u]) {
            if (lr[v] == ll[u] + 1 && vis[v] != dfn) {
                vis[v] = dfn;
                if (mr[v] == 0 || dfs(mr[v])) {
                    mr[v] = u; ml[u] = v;
                    return true;
                }
            }
        }
        return false;
    };

    int HK() {
        ml = vector<int> (nl + 1);
        mr = vector<int> (nr + 1);
        ll = vector<int> (nl + 1);
        lr = vector<int> (nr + 1);
        vis = vector<int> (nr + 1);

        int res = 0;
        while(bfs()) {
            ++dfn;
            for (int i = 1; i <= nl; ++i)
                if (!ml[i]) res += dfs(i);
        }
        return res;
    }
};

/**
 * 最小覆盖数 = 最大匹配数
 * 最大独立集 = 顶点数 - 二分图匹配数
 * DAG 最小路径覆盖数 = 结点数 - 拆点后二分图最大匹配数
 */
```

2.4 BiWraphMatch

```
// Kuhn Munkres, O(V^3)
template<typename T>
```

```cpp
struct biwraph {
    T TMAX, TMIN;

    int n, nl, nr;
    vector<vector<T>> G;
    vector<T> highl, highr;
    vector<T> slack;
    vector<int> matchl, matchr; // match
    vector<int> pre; // pre node
    vector<bool> visl, visr; // vis
    vector<int> q;
    int ql, qr;

    biwraph(int _nl, int _nr) {
        TMAX = numeric_limits<T>::max();

        nl = _nl; nr = _nr; n = max(nl, nr);
        G = vector<vector<T>> (n + 1, vector<T> (n + 1));
        highl = vector<T> (n + 1);
        highr = vector<T> (n + 1);
        slack = vector<T> (n + 1);
        matchl = vector<int> (n + 1);
        matchr = vector<int> (n + 1);
        pre = vector<int> (n + 1);
        visl = vector<bool> (n + 1);
        visr = vector<bool> (n + 1);
        q = vector<int> (n + 1);
    }

    void addarc(int u, int v, T w) {
        G[u][v] = max(G[u][v], w);
    }

    bool check(int v) {
        visr[v] = true;
        if (matchr[v]) {
            q[qr++] = matchr[v];
            visl[matchr[v]] = true;
            return false;
        }

        while(v) {
            matchr[v] = pre[v];
            swap(v, matchl[pre[v]]);
        }

        return true;
    }

    void bfs(int now) {
        ql = qr = 0; q[qr++] = now; visl[now] = 1;
        while(true) {
            while(ql < qr) {
                int u = q[ql++];
                for (int v = 1; v <= n; ++v) {
                    if (!visr[v]) {
                        T delta = highl[u] + highr[v] - G[u][v];
                        if (slack[v] >= delta) {
                            pre[v] = u;
                            if (delta) slack[v] = delta;
                            else if (check(v)) return;
                        }
                    }
                }
            }
```

```

            T a = TMAX;
            for (int i = 1; i <= n; ++i) if (!visr[i]) a = min(a, slack[i]);
            for (int i = 1; i <= n; ++i) {
                if (visl[i]) highl[i] -= a;
                if (visr[i]) highr[i] += a;
                else slack[i] -= a;
            }
            for (int i = 1; i <= n; ++i)
                if (!visr[i] && !slack[i] && check(i)) return;
        }
    }

    void match() {
        fill(highr.begin(), highr.end(), 0);
        fill(matchl.begin(), matchl.end(), 0);
        fill(matchr.begin(), matchr.end(), 0);
        for (int i = 1; i <= n; ++i) highl[i] = *max_element(G[i].begin() + 1, G[i].end());

        for (int i = 1; i <= n; ++i) {
            fill(slack.begin(), slack.end(), TMAX);
            fill(visl.begin(), visl.end(), false);
            fill(visr.begin(), visr.end(), false);
            bfs(i);
        }
    }

    T getMaxMatch() {
        T res = 0;
        match();
        for (int i = 1; i <= n; ++i) {
            if (G[i][matchl[i]] > 0) res += G[i][matchl[i]];
            else matchl[i] = 0;
        }
        return res;
    }
};
```

2.5 BlockForest

```
// 「APIO2018」铁人两项 (https://loj.ac/p/2587)
// 给定一张简单无向图，问有多少对三元组 <s, c, f>（s, c, f 互不相同）使得存在一条简单路径从 s 出发，经过 c 到达 f 。
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
const int N = 2e5 + 5;

int n, m;
int w[N];
vector<int> G[N], F[N];

int cc, scc;
int dfc, dfn[N], low[N];
int top, stk[N];
void tarjan(int u) {
    ++cc;
    dfn[u] = low[u] = ++dfc;
    stk[++top] = u;
    for (int v: G[u]) {
        if (!dfn[v]) {
            tarjan(v);
            low[u] = min(low[u], low[v]);
            if (low[v] == dfn[u]) {
                ++scc;
```

```
                int np = n + scc;
                w[np] = 0;
                for (int x = 0; x != v; --top) {
                    x = stk[top];
                    F[np].push_back(x);
                    F[x].push_back(np);
                    ++w[np];
                }
                F[np].push_back(u);
                F[u].push_back(np);
                ++w[np];
            }
        } else low[u] = min(low[u], dfn[v]);
    }
}

ll ans;
int sz[N];
void dfs(int u, int fa) {
    sz[u] = (u <= n);
    for (int v: F[u]) if (v != fa) {
        dfs(v, u);
        ans += 2ll * w[u] * sz[u] * sz[v];
        sz[u] += sz[v];
    }
    ans += 2ll * w[u] * sz[u] * (cc - sz[u]);
}

void buildBlockForest() {
    for (int i = 1; i <= n; ++i) if (!dfn[i]) {
        cc = 0;
        tarjan(i);
        --top;
        dfs(i, i);
    }
}

void solve(int Case) {
    scanf("%d %d", &n, &m);
    fill(w + 1, w + 1 + n, -1);
    int u, v;
    for (int i = 1; i <= m; ++i) {
        scanf("%d %d", &u, &v);
        G[u].push_back(v);
        G[v].push_back(u);
    }
    buildBlockForest();
    printf("%lld\n", ans);
}

int main () {
    int T = 1;
    // scanf("%d", &T);
    for (int i = 1; i <= T; ++i) solve(i);
    return 0;
}
```

2.6 Dijkstra

```
namespace Backlight {

template<typename T>
struct Wraph {
    struct Edge {
```

```
        int u, v;
        T w;
        Edge(){}
        Edge(int _u, int _v, T _w): u(_u), v(_v), w(_w) {}
    };

    int V;
    vector<vector<Edge>> G;

    Wraph() : V(0) {}
    Wraph(int _V) : V(_V), G(_V + 1) {}

    inline void addarc(int u, int v, T w) {
        assert(1 <= u && u <= V);
        assert(1 <= v && v <= V);
        G[u].push_back(Edge(u, v, w));
    }

    inline void addedge(int u, int v, T w) {
        addarc(u, v, w);
        addarc(v, u, w);
    }

    /*************************************************/
    vector<T> dijkstra(int S, T T_MAX) {
        typedef pair<T, int> Node;
        priority_queue<Node, vector<Node>, greater<Node>> q;
        vector<T> dis(V + 1);
        for (int i = 1; i <= V; i++) dis[i] = T_MAX;
        dis[S] = 0; q.push(Node(0, S));
        while (!q.empty()){
            Node p = q.top(); q.pop();
            T cost = p.first; int u = p.second;
            if (dis[u] != cost) continue;

            for (Edge e: G[u]){
                int v = e.v;
                T w = e.w;
                if (dis[v] > dis[u] + w) {
                    dis[v] = dis[u] + w;
                    q.push(Node(dis[v], v));
                }
            }
        }
        return dis;
    }
};

}
```

2.7 dsu-on-tree

```
// CF600E
// 对于每个节点，输出其子树中出现次数最多的颜色之和。
vector<int> G[N];
inline void addedge(int u, int v) {
    G[u].push_back(v);
    G[v].push_back(u);
}

int n, color[N];

int sz[N], son[N], cnt[N], ma;
ll cur, ans[N];
```

```cpp
void dfs1(int u, int fa) {
    sz[u] = 1; son[u] = -1;
    for (int v: G[u]) {
        if (v == fa) continue;
        dfs1(v, u);
        sz[u] += sz[v];
        if (sz[v] > sz[son[u]]) son[u] = v;
    }
}

void add(int u, int fa, int Son, int d) {
    // update data here
    cnt[color[u]] += d;
    if (cnt[color[u]] > ma) ma = cnt[color[u]], cur = 0;
    if (cnt[color[u]] == ma) cur += color[u];

    for (int v: G[u]) {
        if (v == fa || v == Son) continue;
        add(v, u, Son, d);
    }
}

void dfs2(int u, int fa, bool keep) {
    for (int v: G[u]) {
        if (v == fa || v == son[u]) continue;
        dfs2(v, u, false);
    }
    if (son[u] != -1) dfs2(son[u], u, true);

    add(u, fa, son[u], 1);

    // answer queries here
    ans[u] = cur;

    if (!keep) {
        add(u, fa, -1, -1);
        ma = 0; cur = 0;
    }
}

void solve() {
    read(n);
    FOR(i, 1, n) read(color[i]);

    int u, v;
    FOR(i, 2, n) {
        read(u, v);
        addedge(u, v);
    }

    dfs1(1, 0);
    dfs2(1, 0, 0);

    FOR(i, 1, n - 1) printf("%lld ", ans[i]);
    println(ans[n]);
}
```

2.8 FullyDCP

```cpp
// Got this code from LOJ
#include <bits/stdc++.h>
using namespace std;

struct Xor128 {
```

```
    unsigned x, y, z, w;
    Xor128() : x(123456789), y(362436069), z(521288629), w(88675123) {}
    unsigned next() {
        unsigned t = x ^ (x << 11);
        x = y;
        y = z;
        z = w;
        return w = w ^ (w >> 19) ^ (t ^ (t >> 8));
    }
    //手Fき
    inline unsigned next(unsigned n) { return next() % n; }
};

// bottom up な Treap
//脱再F!
// randomized binary search にするには choiceRandomly を
//        bool choiceRandomly(Ref l, Ref r) { return rng.next(l->size + r->size) < l->size; }
//に書きFえるだけでよい。
template <typename Node>
struct BottomupTreap {
    Xor128 rng;
    typedef Node *Ref;
    static int size(Ref t) { return !t ? 0 : t->size; }

    unsigned nextRand() { return rng.next(); }

private:
    bool choiceRandomly(Ref l, Ref r) { return l->priority < r->priority; }

public:
    Ref join(Ref l, Ref r) {
        if (!l)
            return r;
        if (!r)
            return l;

        Ref t = NULL;
        unsigned long long dirs = 0;
        int h;
        for (h = 0;; ++h) {
            if (h >= sizeof(dirs) * 8 - 2) {
                // dirs のオFバFフロFを防ぐために再Fする。
                //あくまでセFフティガFドなのでバランスは多少崩れるかもしれない
                t = join(l->right, r->left);
                dirs = dirs << 2 | 1;
                h++;
                break;
            }
            dirs <<= 1;
            if (choiceRandomly(l, r)) {
                Ref c = l->right;
                if (!c) {
                    t = r;
                    r = r->parent;
                    break;
                }
                l = c;
            } else {
                dirs |= 1;
                Ref c = r->left;
                if (!c) {
                    t = l;
                    l = l->parent;
                    break;
                }
```

```
                r = c;
            }
        }
        for (; h >= 0; --h) {
            if (!(dirs & 1)) {
                Ref p = l->parent;
                t = l->linkr(t);
                l = p;
            } else {
                Ref p = r->parent;
                t = r->linkl(t);
                r = p;
            }
            dirs >>= 1;
        }
        return t;
    }

    typedef std::pair<Ref, Ref> RefPair;

    // l < t□r の (l,r) に分割する
    RefPair split2(Ref t) {
        Ref p, l = t->left, r = t;
        Node::cut(l);
        t->linkl(NULL);
        while (p = t->parent) {
            t->parent = NULL;
            if (p->left == t)
                r = p->linkl(r);
            else
                l = p->linkr(l);
            t = p;
        }
        return RefPair(l, r);
    }
    // l < t < r の (l,t,r) に分割する。(l,r) を返す
    RefPair split3(Ref t) {
        Ref p, l = t->left, r = t->right;
        Node::cut(l), Node::cut(r);
        t->linklr(NULL, NULL);
        while (p = t->parent) {
            t->parent = NULL;
            if (p->left == t)
                r = p->linkl(r);
            else
                l = p->linkr(l);
            t = p;
        }
        return RefPair(l, r);
    }
    Ref cons(Ref h, Ref t) {
        assert(size(h) == 1);
        if (!t)
            return h;
        Ref u = NULL;
        while (true) {
            if (choiceRandomly(h, t)) {
                Ref p = t->parent;
                u = h->linkr(t);
                t = p;
                break;
            }
            Ref l = t->left;
            if (!l) {
                u = h;
```

```
                break;
            }
            t = l;
        }
        while (t) {
            u = t->linkl(u);
            t = t->parent;
        }
        return u;
    }
};

// free tree のために、Fを基本としてFう
class EulerTourTreeWithMarks {
    struct Node {
        typedef BottomupTreap<Node> BST;

        Node *left, *right, *parent;
        int size;
        unsigned priority;
        char marks, markUnions;  // 0 ビット目が edgeMark, 1 ビット目が vertexMark

        Node() : left(NULL), right(NULL), parent(NULL), size(1), priority(0), marks(0), markUnions(0) {}

        inline Node *update() {
            int size_t = 1, markUnions_t = marks;
            if (left) {
                size_t += left->size;
                markUnions_t |= left->markUnions;
            }
            if (right) {
                size_t += right->size;
                markUnions_t |= right->markUnions;
            }
            size = size_t, markUnions = markUnions_t;
            return this;
        }

        inline Node *linkl(Node *c) {
            if (left = c)
                c->parent = this;
            return update();
        }
        inline Node *linkr(Node *c) {
            if (right = c)
                c->parent = this;
            return update();
        }
        inline Node *linklr(Node *l, Node *r) {
            if (left = l)
                l->parent = this;
            if (right = r)
                r->parent = this;
            return update();
        }
        static Node *cut(Node *t) {
            if (t)
                t->parent = NULL;
            return t;
        }

        static const Node *findRoot(const Node *t) {
            while (t->parent) t = t->parent;
            return t;
        }
```

```
        static std::pair<Node *, int> getPosition(Node *t) {
            int k = BST::size(t->left);
            Node *p;
            while (p = t->parent) {
                if (p->right == t)
                    k += BST::size(p->left) + 1;
                t = p;
            }
            return std::make_pair(t, k);
        }
        static const Node *findHead(const Node *t) {
            while (t->left) t = t->left;
            return t;
        }
        static void updatePath(Node *t) {
            while (t) {
                t->update();
                t = t->parent;
            }
        }
    };

    typedef Node::BST BST;
    BST bst;

    std::vector<Node> nodes;
    //各頂点にFしてその頂点から出ている arc を 1 つだけ代表として持つ (無い場合は-1)
    //逆に arc にFしてFFする頂点はたかだか 1 つである
    std::vector<int> firstArc;
    //F・頂点にFする属性
    std::vector<bool> edgeMark, vertexMark;

    inline int getArcIndex(const Node *a) const { return a - &nodes[0]; }

    inline int arc1(int ei) const { return ei; }
    inline int arc2(int ei) const { return ei + (numVertices() - 1); }

public:
    inline int numVertices() const { return firstArc.size(); }
    inline int numEdges() const { return numVertices() - 1; }

    inline bool getEdgeMark(int a) const { return a < numEdges() ? edgeMark[a] : false; }
    inline bool getVertexMark(int v) const { return vertexMark[v]; }

private:
    void updateMarks(int a, int v) {
        Node *t = &nodes[a];
        t->marks = getEdgeMark(a) << 0 | getVertexMark(v) << 1;
        Node::updatePath(t);
    }

    // firstArc のF更にFじて更新する
    void firstArcChanged(int v, int a, int b) {
        if (a != -1)
            updateMarks(a, v);
        if (b != -1)
            updateMarks(b, v);
    }

public:
    class TreeRef {
        friend class EulerTourTreeWithMarks;
        const Node *ref;

    public:
```

```
        TreeRef() {}
        TreeRef(const Node *ref_) : ref(ref_) {}
        bool operator==(const TreeRef &that) const { return ref == that.ref; }
        bool operator!=(const TreeRef &that) const { return ref != that.ref; }
        bool isIsolatedVertex() const { return ref == NULL; }
    };

    void init(int N) {
        int M = N - 1;
        firstArc.assign(N, -1);
        nodes.assign(M * 2, Node());
        for (int i = 0; i < M * 2; i++) nodes[i].priority = bst.nextRand();
        edgeMark.assign(M, false);
        vertexMark.assign(N, false);
    }

    TreeRef getTreeRef(int v) const {
        int a = firstArc[v];
        return TreeRef(a == -1 ? NULL : Node::findRoot(&nodes[a]));
    }

    bool isConnected(int v, int w) const {
        if (v == w)
            return true;
        int a = firstArc[v], b = firstArc[w];
        if (a == -1 || b == -1)
            return false;
        return Node::findRoot(&nodes[a]) == Node::findRoot(&nodes[b]);
    }

    static int getSize(TreeRef t) {
        if (t.isIsolatedVertex())
            return 1;
        else
            return t.ref->size / 2 + 1;
    }

    void link(int ti, int v, int w) {
        int a1 = arc1(ti), a2 = arc2(ti);
        // v→w が a1 にFFするようにする
        if (v > w)
            std::swap(a1, a2);

        int va = firstArc[v], wa = firstArc[w];

        Node *l, *m, *r;
        if (va != -1) {
            // evert。順番を入れ替えるだけ
            std::pair<Node *, Node *> p = bst.split2(&nodes[va]);
            m = bst.join(p.second, p.first);
        } else {
            // v が孤立点の場合
            m = NULL;
            firstArc[v] = a1;
            firstArcChanged(v, -1, a1);
        }
        if (wa != -1) {
            std::pair<Node *, Node *> p = bst.split2(&nodes[wa]);
            l = p.first, r = p.second;
        } else {
            // w が孤立点の場合
            l = r = NULL;
            firstArc[w] = a2;
            firstArcChanged(w, -1, a2);
        }
```

```
        // w→v のFを m の先頭= l の末尾に insert
        m = bst.cons(&nodes[a2], m);
        // v→w のFを m の末尾= r の先頭に insert
        r = bst.cons(&nodes[a1], r);

        bst.join(bst.join(l, m), r);
    }

    void cut(int ti, int v, int w) {
        // v→w が a1 にFFするようにする
        if (v > w)
            std::swap(v, w);

        int a1 = arc1(ti), a2 = arc2(ti);
        std::pair<Node *, Node *> p = bst.split3(&nodes[a1]);
        int prsize = BST::size(p.second);
        std::pair<Node *, Node *> q = bst.split3(&nodes[a2]);
        Node *l, *m, *r;
        // a1,a2 の順番を判定する。a1 < a2 なら p.second がFわっているはず
        if (p.second == &nodes[a2] || BST::size(p.second) != prsize) {
            l = p.first, m = q.first, r = q.second;
        } else {
            // a2 < a1 の順番である。v→w のFが a1 であって親 → 子であることにする
            std::swap(v, w);
            std::swap(a1, a2);
            l = q.first, m = q.second, r = p.second;
        }

        // firstArc を必要にFじて書きFえる
        if (firstArc[v] == a1) {
            int b;
            if (r != NULL) {
                // v が根じゃないなら右側の最初のFでよい
                b = getArcIndex(Node::findHead(r));
            } else {
                // v が根なら最初のFでよい。孤立点になるなら-1
                b = !l ? -1 : getArcIndex(Node::findHead(l));
            }
            firstArc[v] = b;
            firstArcChanged(v, a1, b);
        }
        if (firstArc[w] == a2) {
            // w が根になるので最初のFでよい。孤立点になるなら-1
            int b = !m ? -1 : getArcIndex(Node::findHead(m));
            firstArc[w] = b;
            firstArcChanged(w, a2, b);
        }

        bst.join(l, r);
    }

    void changeEdgeMark(int ti, bool b) {
        assert(ti < numEdges());
        edgeMark[ti] = b;
        Node *t = &nodes[ti];
        t->marks = (b << 0) | (t->marks & (1 << 1));
        Node::updatePath(t);
    }
    void changeVertexMark(int v, bool b) {
        vertexMark[v] = b;
        int a = firstArc[v];
        if (a != -1) {
            Node *t = &nodes[a];
            t->marks = (t->marks & (1 << 0)) | (b << 1);
            Node::updatePath(t);
```

```
        }
    }

    template <typename Callback>
    bool enumMarkedEdges(TreeRef tree, Callback callback) const {
        return enumMarks<0, Callback>(tree, callback);
    }
    //孤立点の場合は呼び側でその頂点だけF理する必要がある
    template <typename Callback>
    bool enumMarkedVertices(TreeRef tree, Callback callback) const {
        return enumMarks<1, Callback>(tree, callback);
    }

private:
    // callback : TreeEdgeIndex×2 -> Bool
    //引数は頂点をそこからの incident arc で示し、"(正方向 ? 0 : N-1) +
    // treeEdgeIndex" を表す。方向は v,w の大小でF理すればよい
    // callback はFFするかどうかを bool で返す。最後まで列Fし終えたかどうかを返す。
    template <int Mark, typename Callback>
    bool enumMarks(TreeRef tree, Callback callback) const {
        if (tree.isIsolatedVertex())
            return true;
        const Node *t = tree.ref;
        if (t->markUnions >> Mark & 1)
            return enumMarksRec<Mark, Callback>(t, callback);
        else
            return true;
    }

    //平衡木なので深さは深くないので再Fして問題ない
    template <int Mark, typename Callback>
    bool enumMarksRec(const Node *t, Callback callback) const {
        const Node *l = t->left, *r = t->right;
        if (l && (l->markUnions >> Mark & 1))
            if (!enumMarksRec<Mark, Callback>(l, callback))
                return false;
        if (t->marks >> Mark & 1)
            if (!callback(getArcIndex(t)))
                return false;
        if (r && (r->markUnions >> Mark & 1))
            if (!enumMarksRec<Mark, Callback>(r, callback))
                return false;
        return true;
    }

public:
    //デバッグ用
    void debugEnumEdges(std::vector<int> &out_v) const {
        int M = numEdges();
        for (int ti = 0; ti < M; ti++) {
            const Node *t = &nodes[ti];
            if (t->left || t->right || t->parent)
                out_v.push_back(ti);
        }
    }
};

// treeEdge にはそれぞれ 0~N-1 のインデックスが与えられる。これは全てのレベルで共通。
//ところで"level up" って和F英語なんだ。promote でいいかな。
// Sampling heuristic ランダムケFスで超速く (4 倍とか) なったんだけど! いいね!
//
// References
// · Holm, Jacob, Kristian De Lichtenberg, and Mikkel Thorup. "Poly-logarithmic deterministic fully-dynamic
// algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity." Journal of the ACM
//(JACM) 48.4 (2001): 723-760. · Iyer, Raj, et al. "An experimental study of polylogarithmic, fully dynamic,
```

```
// connectivity algorithms." Journal of Experimental Algorithmics (JEA) 6 (2001): 4.

class HolmDeLichtenbergThorup {
    typedef HolmDeLichtenbergThorup This;
    typedef EulerTourTreeWithMarks Forest;
    typedef Forest::TreeRef TreeRef;

    int numVertices_m;
    int numSamplings;

    // DynamicTree はコピ🄵できないけどまあその状態で使わなきゃいいじゃんということで…
    std::vector<Forest> forests;

    std::vector<char> edgeLevel;
    std::vector<int> treeEdgeIndex;          // : EdgeIndex -> TreeEdgeIndex
    std::vector<int> treeEdgeMap;            // : TreeEdgeIndex -> EdgeIndex
    std::vector<int> treeEdgeIndexFreeList;  // : [TreeEdgeIndex]

    // arc も方向は EulerTourTree と同じように v,w の大小に合わせる
    std::vector<int> arcHead;

    std::vector<std::vector<int>> firstIncidentArc;
    std::vector<int> nextIncidentArc, prevIncidentArc;

    //一時的に使う。使い回して使う
    std::vector<bool> edgeVisited;
    std::vector<int> visitedEdges;  // : [EdgeIndex | TreeEdgeIndex]

    int arc1(int ei) const { return ei; }
    int arc2(int ei) const { return numMaxEdges() + ei; }
    int arcEdge(int i) const { return i >= numMaxEdges() ? i - numMaxEdges() : i; }

    bool replace(int lv, int v, int w) {
        Forest &forest = forests[lv];

        TreeRef vRoot = forest.getTreeRef(v), wRoot = forest.getTreeRef(w);
        assert(vRoot.isIsolatedVertex() || wRoot.isIsolatedVertex() || vRoot != wRoot);

        int vSize = forest.getSize(vRoot), wSize = forest.getSize(wRoot);

        int u;
        TreeRef uRoot;
        int uSize;
        if (vSize <= wSize)
            u = v, uRoot = vRoot, uSize = vSize;
        else
            u = w, uRoot = wRoot, uSize = wSize;

        // replacement edge を探す
        int replacementEdge = -1;
        enumIncidentArcs(forest, uRoot, u, lv, FindReplacementEdge(uRoot, &replacementEdge));

        //"Sampling heuristic"
        //早い時点で見つかったなら T_u, 他の incident arcs をレベルアップさせなくても計算量的に問題ない
        if (replacementEdge != -1 && (int)visitedEdges.size() + 1 <= numSamplings) {
            // replacementEdge を🄵理する
            deleteNontreeEdge(replacementEdge);
            addTreeEdge(replacementEdge);
            for (int i = 0; i < (int)visitedEdges.size(); i++) edgeVisited[visitedEdges[i]] = false;
            visitedEdges.clear();
            return true;
        }

        //見つけた incident arcs を一🄵にレベルアップさせる。edgeVisited の後🄵理もする
        for (int i = 0; i < (int)visitedEdges.size(); i++) {
```

```
            int ei = visitedEdges[i];
            edgeVisited[ei] = false;

            deleteNontreeEdge(ei);

            ++edgeLevel[ei];

            insertNontreeEdge(ei);
        }
        visitedEdges.clear();

        //このレベルの T_u のFを列Fする
        forest.enumMarkedEdges(uRoot, EnumLevelTreeEdges(this));
        //列Fした T_u のFを一Fにレベルアップさせる
        for (int i = 0; i < (int)visitedEdges.size(); i++) {
            int ti = visitedEdges[i];

            int ei = treeEdgeMap[ti];
            int v = arcHead[arc2(ei)], w = arcHead[arc1(ei)];
            int lv = edgeLevel[ei];

            edgeLevel[ei] = lv + 1;

            forests[lv].changeEdgeMark(ti, false);
            forests[lv + 1].changeEdgeMark(ti, true);

            forests[lv + 1].link(ti, v, w);
        }
        visitedEdges.clear();

        if (replacementEdge != -1) {
            // T_u のF列Fの前に構造がFわると困るので replacementEdge はこのタイミングでF理する
            deleteNontreeEdge(replacementEdge);
            addTreeEdge(replacementEdge);
            return true;
        } else if (lv > 0) {
            return replace(lv - 1, v, w);
        } else {
            return false;
        }
    }

    struct EnumLevelTreeEdges {
        This *thisp;
        EnumLevelTreeEdges(This *thisp_) : thisp(thisp_) {}

        inline bool operator()(int a) {
            thisp->enumLevelTreeEdges(a);
            return true;
        }
    };
    void enumLevelTreeEdges(int ti) { visitedEdges.push_back(ti); }

    //孤立点の時特Fな Fを理をするなどしなければいけないのでヘルパF
    template <typename Callback>
    bool enumIncidentArcs(Forest &forest, TreeRef t, int u, int lv, Callback callback) {
        if (t.isIsolatedVertex())
            return enumIncidentArcsWithVertex<Callback>(lv, u, callback);
        else
            return forest.enumMarkedVertices(t, EnumIncidentArcs<Callback>(this, lv, callback));
    }

    template <typename Callback>
    struct EnumIncidentArcs {
        This *thisp;
```

```
591         int lv;
592         Callback callback;
593 
594         EnumIncidentArcs(This *thisp_, int lv_, Callback callback_)
595             : thisp(thisp_), lv(lv_), callback(callback_) {}
596 
597         inline bool operator()(int tii) const {
598             return thisp->enumIncidentArcsWithTreeArc(tii, lv, callback);
599         }
600     };
601 
602     template <typename Callback>
603     bool enumIncidentArcsWithTreeArc(int tii, int lv, Callback callback) {
604         bool dir = tii >= numVertices() - 1;
605         int ti = dir ? tii - (numVertices() - 1) : tii;
606         int ei = treeEdgeMap[ti];
607         int v = arcHead[arc2(ei)], w = arcHead[arc1(ei)];
608         //方向を求め、その arc の tail の頂点を取得する
609         int u = !(dir != (v > w)) ? v : w;
610 
611         return enumIncidentArcsWithVertex(lv, u, callback);
612     }
613 
614     // 1 つの頂点をF理する
615     template <typename Callback>
616     bool enumIncidentArcsWithVertex(int lv, int u, Callback callback) {
617         int it = firstIncidentArc[lv][u];
618         while (it != -1) {
619             if (!callback(this, it))
620                 return false;
621             it = nextIncidentArc[it];
622         }
623         return true;
624     }
625 
626     struct FindReplacementEdge {
627         TreeRef uRoot;
628         int *replacementEdge;
629         FindReplacementEdge(TreeRef uRoot_, int *replacementEdge_)
630             : uRoot(uRoot_), replacementEdge(replacementEdge_) {}
631 
632         inline bool operator()(This *thisp, int a) const {
633             return thisp->findReplacementEdge(a, uRoot, replacementEdge);
634         }
635     };
636 
637     // 1 つの arc をF理する
638     bool findReplacementEdge(int a, TreeRef uRoot, int *replacementEdge) {
639         int ei = arcEdge(a);
640         if (edgeVisited[ei])
641             return true;
642 
643         int lv = edgeLevel[ei];
644         TreeRef hRoot = forests[lv].getTreeRef(arcHead[a]);
645 
646         if (hRoot.isIsolatedVertex() || hRoot != uRoot) {
647             //Fの木に渡されているなら replacement edge である。
648             *replacementEdge = ei;
649             return false;
650         }
651         // replacement edge は visitedEdges に入れたくないのでこの位置でマFクする
652         edgeVisited[ei] = true;
653         visitedEdges.push_back(ei);
654         return true;
655     }
```

```
656
657     void addTreeEdge(int ei) {
658         int v = arcHead[arc2(ei)], w = arcHead[arc1(ei)];
659         int lv = edgeLevel[ei];
660
661         int ti = treeEdgeIndexFreeList.back();
662         treeEdgeIndexFreeList.pop_back();
663         treeEdgeIndex[ei] = ti;
664         treeEdgeMap[ti] = ei;
665
666         forests[lv].changeEdgeMark(ti, true);
667
668         for (int i = 0; i <= lv; i++) forests[i].link(ti, v, w);
669     }
670
671     void insertIncidentArc(int a, int v) {
672         int ei = arcEdge(a);
673         int lv = edgeLevel[ei];
674         assert(treeEdgeIndex[ei] == -1);
675
676         int next = firstIncidentArc[lv][v];
677         firstIncidentArc[lv][v] = a;
678         nextIncidentArc[a] = next;
679         prevIncidentArc[a] = -1;
680         if (next != -1)
681             prevIncidentArc[next] = a;
682
683         if (next == -1)
684             forests[lv].changeVertexMark(v, true);
685     }
686
687     void deleteIncidentArc(int a, int v) {
688         int ei = arcEdge(a);
689         int lv = edgeLevel[ei];
690         assert(treeEdgeIndex[ei] == -1);
691
692         int next = nextIncidentArc[a], prev = prevIncidentArc[a];
693         nextIncidentArc[a] = prevIncidentArc[a] = -2;
694
695         if (next != -1)
696             prevIncidentArc[next] = prev;
697         if (prev != -1)
698             nextIncidentArc[prev] = next;
699         else
700             firstIncidentArc[lv][v] = next;
701
702         if (next == -1 && prev == -1)
703             forests[lv].changeVertexMark(v, false);
704     }
705
706     void insertNontreeEdge(int ei) {
707         int a1 = arc1(ei), a2 = arc2(ei);
708         insertIncidentArc(a1, arcHead[a2]);
709         insertIncidentArc(a2, arcHead[a1]);
710     }
711
712     void deleteNontreeEdge(int ei) {
713         int a1 = arc1(ei), a2 = arc2(ei);
714         deleteIncidentArc(a1, arcHead[a2]);
715         deleteIncidentArc(a2, arcHead[a1]);
716     }
717
718 public:
719     HolmDeLichtenbergThorup() : numVertices_m(0), numSamplings(0) {}
720
```

```
721     int numVertices() const { return numVertices_m; }
722     int numMaxEdges() const { return edgeLevel.size(); }
723
724     void init(int N, int M) {
725         numVertices_m = N;
726
727         int levels = 1;
728         while (1 << levels <= N / 2) levels++;
729
730         //サンプリング数を設定する。適切な🄵はよくわからない
731         numSamplings = (int)(levels * 1);
732
733         forests.resize(levels);
734         for (int lv = 0; lv < levels; lv++) forests[lv].init(N);
735
736         edgeLevel.assign(M, -1);
737
738         treeEdgeIndex.assign(M, -1);
739         treeEdgeMap.assign(N - 1, -1);
740
741         treeEdgeIndexFreeList.resize(N - 1);
742         for (int ti = 0; ti < N - 1; ti++) treeEdgeIndexFreeList[ti] = ti;
743
744         arcHead.assign(M * 2, -1);
745
746         firstIncidentArc.resize(levels);
747         for (int lv = 0; lv < levels; lv++) firstIncidentArc[lv].assign(N, -1);
748         nextIncidentArc.assign(M * 2, -2);
749         prevIncidentArc.assign(M * 2, -2);
750
751         edgeVisited.assign(M, false);
752     }
753
754     bool insertEdge(int ei, int v, int w) {
755         if (!(0 <= ei && ei < numMaxEdges() && 0 <= v && v < numVertices() && 0 <= w && w < numVertices())) {
756             system("pause");
757         }
758         assert(0 <= ei && ei < numMaxEdges() && 0 <= v && v < numVertices() && 0 <= w && w < numVertices());
759         assert(edgeLevel[ei] == -1);
760
761         int a1 = arc1(ei), a2 = arc2(ei);
762         arcHead[a1] = w, arcHead[a2] = v;
763
764         bool treeEdge = !forests[0].isConnected(v, w);
765
766         edgeLevel[ei] = 0;
767         if (treeEdge) {
768             addTreeEdge(ei);
769         } else {
770             treeEdgeIndex[ei] = -1;
771             //ル🄵プは見たくないのでリストにも入れない
772             if (v != w)
773                 insertNontreeEdge(ei);
774         }
775
776         return treeEdge;
777     }
778
779     bool deleteEdge(int ei) {
780         assert(0 <= ei && ei < numMaxEdges() && edgeLevel[ei] != -1);
781
782         int a1 = arc1(ei), a2 = arc2(ei);
783         int v = arcHead[a2], w = arcHead[a1];
784
785         int lv = edgeLevel[ei];
```

```
        int ti = treeEdgeIndex[ei];

        bool splitted = false;
        if (ti != -1) {
            treeEdgeMap[ti] = -1;
            treeEdgeIndex[ei] = -1;
            treeEdgeIndexFreeList.push_back(ti);

            for (int i = 0; i <= lv; i++) forests[i].cut(ti, v, w);

            forests[lv].changeEdgeMark(ti, false);

            splitted = !replace(lv, v, w);
        } else {
            //ルFプはリストに入ってない
            if (v != w)
                deleteNontreeEdge(ei);
        }

        arcHead[a1] = arcHead[a2] = -1;
        edgeLevel[ei] = -1;

        return splitted;
    }

    bool isConnected(int v, int w) const { return forests[0].isConnected(v, w); }
};
typedef HolmDeLichtenbergThorup FullyDynamicConnectivity;
map<int, map<int, int>> mp;

int main() {
    int n, m;
    scanf("%d%d", &n, &m);
    mp.clear();
    FullyDynamicConnectivity fdc;
    fdc.init(n + 1, m + 1);
    int posE = 0;
    int lstans = 0;
    for (int i = 1, op, u, v, _u, _v; i <= m; ++i) {
        scanf("%d%d%d", &op, &u, &v);
        u ^= lstans;
        v ^= lstans;
        _u = u, _v = v;
        if (u < v)
            swap(u, v);
        if (op == 0) {
            mp[u][v] = ++posE;
            fdc.insertEdge(posE, u, v);
        } else if (op == 1) {
            fdc.deleteEdge(mp[u][v]);
            mp[u].erase(v);
        } else {
            int ok = fdc.isConnected(u, v);
            if (ok)
                lstans = _u;
            else
                lstans = _v;
            printf("%c\n", "NY"[ok]);
        }
    }
    return 0;
}
```

2.9 Graph

```cpp
namespace Backlight {

struct Graph {
    struct Edge {
        int u, v;
        Edge(){}
        Edge(int _u, int _v): u(_u), v(_v) {}
    };

    int V;
    vector<vector<Edge>> G;

    Graph() : V(0) {}
    Graph(int _V) : V(_V), G(_V + 1) {}

    inline void addarc(int u, int v) {
        assert(1 <= u && u <= V);
        assert(1 <= v && v <= V);
        G[u].push_back(Edge(u, v));
    }

    inline void addedge(int u, int v) {
        addarc(u, v);
        addarc(v, u);
    }
};

}
```

2.10 GraphMatch

```cpp
#include <bits/stdc++.h>
using namespace std;

// graph
template <typename T>
class graph {
public:
    struct edge {
        int from;
        int to;
        T cost;
    };
    vector<edge> edges;
    vector<vector<int>> g;
    int n;
    graph(int _n)
        : n(_n)
    {
        g.resize(n);
    }
    virtual int add(int from, int to, T cost) = 0;
};

// undirectedgraph
template <typename T>
class undirectedgraph : public graph<T> {
public:
    using graph<T>::edges;
    using graph<T>::g;
    using graph<T>::n;
```

```

    undirectedgraph(int _n)
        : graph<T>(_n)
    {
    }
    int add(int from, int to, T cost = 1)
    {
        assert(0 <= from && from < n && 0 <= to && to < n);
        int id = (int)edges.size();
        g[from].push_back(id);
        g[to].push_back(id);
        edges.push_back({ from, to, cost });
        return id;
    }
};

// blossom / find_max_unweighted_matching
template <typename T>
vector<int> find_max_unweighted_matching(const undirectedgraph<T>& g)
{
    std::mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
    vector<int> match(g.n, -1); // 匹配
    vector<int> aux(g.n, -1); // 时间戳记
    vector<int> label(g.n); // "o" or "i"
    vector<int> orig(g.n); // 花根
    vector<int> parent(g.n, -1); // 父节点
    queue<int> q;
    int aux_time = -1;

    auto lca = [&](int v, int u) {
        aux_time++;
        while (true) {
            if (v != -1) {
                if (aux[v] == aux_time) { // 找到拜访过的点 也就是 LCA
                    return v;
                }
                aux[v] = aux_time;
                if (match[v] == -1) {
                    v = -1;
                } else {
                    v = orig[parent[match[v]]]; // 以匹配点的父节点继续寻找
                }
            }
            swap(v, u);
        }
    }; // lca

    auto blossom = [&](int v, int u, int a) {
        while (orig[v] != a) {
            parent[v] = u;
            u = match[v];
            if (label[u] == 1) { // 初始点设为"o" 找增广路
                label[u] = 0;
                q.push(u);
            }
            orig[v] = orig[u] = a; // 缩花
            v = parent[u];
        }
    }; // blossom

    auto augment = [&](int v) {
        while (v != -1) {
            int pv = parent[v];
            int next_v = match[pv];
            match[v] = pv;
```

```
            match[pv] = v;
            v = next_v;
        }
    }; // augment

    auto bfs = [&](int root) {
        fill(label.begin(), label.end(), -1);
        iota(orig.begin(), orig.end(), 0);
        while (!q.empty()) {
            q.pop();
        }
        q.push(root);
        // 初始点设为 "o", 这里以"0" 代替"o", "1" 代替"i"
        label[root] = 0;
        while (!q.empty()) {
            int v = q.front();
            q.pop();
            for (int id : g.g[v]) {
                auto& e = g.edges[id];
                int u = e.from ^ e.to ^ v;
                if (label[u] == -1) { // 找到未拜访点
                    label[u] = 1; // 标记 "i"
                    parent[u] = v;
                    if (match[u] == -1) { // 找到未匹配点
                        augment(u); // 寻找增广路径
                        return true;
                    }
                    // 找到已匹配点 将与她匹配的点丢入 queue 延伸交错树
                    label[match[u]] = 0;
                    q.push(match[u]);
                    continue;
                } else if (label[u] == 0 && orig[v] != orig[u]) { // 找到已拜访点 且标记同为"o" 代表找到" 花"
                    int a = lca(orig[v], orig[u]);
                    // 找 LCA 然后缩花
                    blossom(u, v, a);
                    blossom(v, u, a);
                }
            }
        }
        return false;
    }; // bfs

    auto greedy = [&]() {
        vector<int> order(g.n);
        // 随机打乱 order
        iota(order.begin(), order.end(), 0);
        shuffle(order.begin(), order.end(), rng);

        // 将可以匹配的点匹配
        for (int i : order) {
            if (match[i] == -1) {
                for (auto id : g.g[i]) {
                    auto& e = g.edges[id];
                    int to = e.from ^ e.to ^ i;
                    if (match[to] == -1) {
                        match[i] = to;
                        match[to] = i;
                        break;
                    }
                }
            }
        }
    }; // greedy

    // 一开始先随机匹配
```

```
    greedy();
    // 对未匹配点找增广路
    for (int i = 0; i < g.n; i++) {
        if (match[i] == -1) {
            bfs(i);
        }
    }
    return match;
}
int main()
{
    ios::sync_with_stdio(0), cin.tie(0);
    int n, m;
    cin >> n >> m;
    undirectedgraph<int> g(n);
    int u, v;
    for (int i = 0; i < m; i++) {
        cin >> u >> v;
        u--;
        v--;
        g.add(u, v, 1);
    }
    auto blossom_match = find_max_unweighted_matching(g);
    vector<int> ans;
    int tot = 0;
    for (int i = 0; i < blossom_match.size(); i++) {
        ans.push_back(blossom_match[i]);
        if (blossom_match[i] != -1) {
            tot++;
        }
    }
    cout << (tot >> 1) << "\n";
    for (auto x : ans) {
        cout << x + 1 << " ";
    }
}
```

2.11 Kosaraju

```
const int N = 1e5 + 5;
vector<int> G[N], R[N];
void init(int n) {
    for (int i = 1; i <= n; ++i) G[i].clear(), R[i].clear();
}
inline void addarc(int u, int v) {
    G[u].push_back(v);
    R[v].push_back(u);
}

int n, m;
int dfs_clock, scc_cnt;
int dfn[N], belong[N];
bool vis[N];
void dfs1(int u) {
    vis[u] = true;
    for (const int& v: G[u]) {
        if (!vis[v]) dfs1(v);
    }
    dfn[++dfs_clock] = u;
}
void dfs2(int u) {
    belong[u] = scc_cnt;
    for (const int& v: R[u]) {
        if (!belong[v]) dfs2(v);
```

```
    }
}
void kosaraju() {
    dfs_clock = scc_cnt = 0;
    fill(dfn + 1, dfn + 1 + n, 0);
    fill(belong + 1, belong + 1 + n, 0);
    fill(vis + 1, vis + 1 + n, false);
    for (int i = 1; i <= n; ++i) {
        if (!vis[i]) dfs1(i);
    }

    for (int i = n; i >= 1; --i) {
        if (!belong[dfn[i]]) {
            ++scc_cnt;
            dfs2(dfn[i]);
        }
    }
}
```

2.12 Kruskal

```
namespace Backlight {

template<typename T>
struct Wraph {
    struct Edge {
        int u, v;
        T w;
        Edge(){}
        Edge(int _u, int _v, T _w): u(_u), v(_v), w(_w) {}
        bool operator < (const Edge& e) {
            return w < e.w;
        }
    };

    int V;
    vector<vector<Edge>> G;
    vector<Edge> E;

    Wraph() : V(0) {}
    Wraph(int _V) : V(_V), G(_V + 1) {}

    inline void addarc(int u, int v, T w) {
        assert(1 <= u && u <= V);
        assert(1 <= v && v <= V);
        G[u].push_back(Edge(u, v, w));
        E.push_back(Edge(u, v, w));
    }

    inline void addedge(int u, int v, T w) {
        addarc(u, v, w);
        addarc(v, u, w);
    }

    /*************************************************/
    T kruskal() {
        vector<int> fa(V + 1);
        for (int i = 1; i <= V; ++i) fa[i] = i;

        auto find = [&fa] (auto self, int x) {
            if (x == fa[x]) return x;
            fa[x] = self(self, fa[x]);
            return fa[x];
        };
```

```

        auto merge = [&fa, find] (int x, int y) {
            x = find(find, x); y = find(find, y);
            if (x == y) return false;
            fa[x] = y;
            return true;
        };

        T cost = 0;
        int cnt = 0;
        sort(E.begin(), E.end());
        for (int i = 0; i < (int)E.size(); ++i) {
            Edge e = E[i];
            if (merge(e.u, e.v)) {
                cost = e.w;
                ++cnt;
                if (cnt == V - 1) break;
            }
        }
        return cost;
    }
};

}
```

2.13 LCA-HLD

```
int tot, head[N];
struct Edge {
    int v, nxt;
}e[M];

void addedge(int u, int v) {
    ++tot; e[tot] = (Edge){v, head[u]}; head[u] = tot;
    ++tot; e[tot] = (Edge){u, head[v]}; head[v] = tot;
}

int h[N], f[N], sz[N], son[N], top[N];
void dfs1(int u, int fa) {
    h[u] = h[fa] + 1; f[u] = fa;
    sz[u] = 1; son[u] = 0;
    for (int i = head[u]; i; i = e[i].nxt) {
        int v = e[i].v;
        if (v == fa) continue;
        dfs1(v, u);
        sz[u] += sz[v];
        if(sz[v] > sz[son[u]]) son[u] = v;
    }
}

void dfs2(int u, int fa, int tp) {
    top[u] = tp;
    if(son[u]) dfs2(son[u], u, tp);
    for (int i = head[u]; i; i = e[i].nxt) {
        int v = e[i].v;
        if (v == fa || v == son[u]) continue;
        dfs2(v, u, v);
    }
}

int LCA(int u, int v) {
    while(top[u] != top[v]) {
        if (h[top[u]] < h[top[v]]) swap(u, v);
        u = f[top[u]];
```

```
    }
    if (h[u] > h[v]) swap(u, v);
    return u;
}
```

2.14 LCA

```
namespace Backlight {

template<typename T>
struct Wraph {
    struct Edge {
        int u, v;
        T w;
        Edge() {}
        Edge(int _u, int _v, T _w): u(_u), v(_v), w(_w) {}
    };

    int V;
    vector<vector<Edge>> G;

    Wraph() : V(0) {}
    Wraph(int _V) : V(_V), G(_V + 1) {}

    inline void addarc(int u, int v, T w = 1) {
        assert(1 <= u && u <= V);
        assert(1 <= v && v <= V);
        G[u].push_back(Edge(u, v, w));
    }

    inline void addedge(int u, int v, T w = 1) {
        addarc(u, v, w);
        addarc(v, u, w);
    }

    /*************************************************/
    vector<int> dep;
    vector<T> dis;
    vector<vector<int>> par;
    int rt, LG;
    void dfs(int u, int fa, int d1, int d2) {
        dep[u] = d1; dis[u] = d2;
        if (u == rt) {
            for (int i = 0; i < LG; ++i) par[u][i] = rt;
        } else {
            par[u][0] = fa;
            for(int i = 1; i < LG; ++i) {
                par[u][i] = par[par[u][i - 1]][i - 1];
            }
        }

        for(Edge& e: G[u]) {
            int v = e.v; T w = e.w;
            if(v == fa) continue;
            dfs(v, u, d1 + 1, d2 + w);
        }
    }

    inline void build_lca(int _rt) {
        rt = _rt; LG = __lg(V + 1) + 1;
        dep = vector<int>(V + 1);
        dis = vector<T>(V + 1);
        par = vector<vector<int>>(V + 1, vector<int>(LG));
        dfs(rt, rt, 0, 0);
```

```
    }

    inline int jump(int u, int d) {
        for(int j = LG - 1; j >= 0; --j){
            if((1<<j) & d) u = par[u][j];
        }
        return u;
    }

    int lca(int u, int v) {
        if (dep[u] < dep[v]) swap(u, v);
        u = jump(u, dep[u] - dep[v]);
        if (u == v) return u;
        for(int i = LG - 1; i >= 0; --i){
            if(par[u][i] != par[v][i]){
                u = par[u][i];
                v = par[v][i];
            }
        }
        return par[u][0];
    }
};

};
```

2.15 maxflow

```
namespace Backlight {

    template<typename Cap>
    struct mf_graph {
        static const Cap INF = numeric_limits<Cap>::max();

        struct Edge {
            int v, nxt;
            Cap c, f;
            Edge(){}
            Edge(int _v, int _nxt, Cap _c): v(_v), nxt(_nxt), c(_c), f(0) {}
        };

        int V, E;
        vector<int> h;
        vector<Edge> e;

        mf_graph() : V(0) {}
        mf_graph(int _V) : V(_V), h(_V + 1, -1) { }

        inline void addarc(int u, int v, Cap c) {
            assert(1 <= u && u <= V);
            assert(1 <= v && v <= V);
            assert(0 <= c);

            e.push_back(Edge(v, h[u], c)); h[u] = e.size() - 1;
        }

        inline void addedge(int u, int v, Cap c) {
            addarc(u, v, c);
            addarc(v, u, 0);
        }

        Cap maxflow(int s, int t) {
            assert(1 <= s && s <= V);
            assert(1 <= t && t <= V);
            assert(s != t);
```

```
            vector<int> f(V + 1), d(V + 1), st(V + 1);

            auto bfs = [&] () {
                fill(d.begin(), d.end(), -1);
                queue<int> q;
                q.push(s); d[s] = 0;
                while(!q.empty()){
                    int u = q.front(); q.pop();
                    for(int i = h[u]; i != -1; i = e[i].nxt) {
                        int v = e[i].v;
                        if(e[i].c > e[i].f && d[v] == -1) {
                            d[v] = d[u] + 1;
                            if (v == t) break;
                            q.push(v);
                        }
                    }
                }
                return (d[t] != -1);
            };

            auto dfs = [&] (auto self, int u, Cap up) {
                if(u == t || up == 0) return up;
                Cap res = 0;
                for(int& i = f[u]; i != -1; i = e[i].nxt) {
                    int v = e[i].v;
                    if(d[u] + 1 == d[v]) {
                        Cap nf = self(self, v, min(up, e[i].c - e[i].f));
                        if (nf <= 0) continue;
                        up -= nf;
                        res += nf;
                        e[i].f += nf;
                        e[i ^ 1].f -= nf;
                        if(up == 0) break;
                    }
                }
                if(res == 0) d[u] = -1;
                return res;
            };

            Cap res = 0;
            while(bfs()) {
                f = h;
                res += dfs(dfs, s, INF);
            }
            return res;
        }
    };

} // namespace Backlight
```

2.16 mincostflow

```
namespace Backlight {

    template<typename Cap, typename Cost>
    struct mcmf_graph {
        static const Cap INF = numeric_limits<Cap>::max();

        struct Edge {
            int v, nxt;
            Cap cap, flow;
            Cost cost;
            Edge() {}
```

```
        Edge(int _v, int _nxt, Cap _cap, Cost _cost)
            : v(_v), nxt(_nxt), cap(_cap), flow(0), cost(_cost) {}
    };

    int V, E;
    vector<int> h;
    vector<Edge> e;

    mcmf_graph() : V(0) {}
    mcmf_graph(int _V) : V(_V), h(_V + 1, -1) { }

    inline void addarc(int u, int v, Cap cap, Cost cost) {
        assert(1 <= u && u <= V);
        assert(1 <= v && v <= V);
        e.push_back(Edge(v, h[u], cap, cost)); h[u] = e.size() - 1;
    }

    inline void addedge(int u, int v, Cap cap, Cost cost) {
        addarc(u, v, cap, cost);
        addarc(v, u, 0, -cost);
    }

    pair<Cap, Cost> mcmf(int s, int t) {
        assert(1 <= s && s <= V);
        assert(1 <= t && t <= V);
        assert(s != t);

        Cap flow = 0;
        Cost cost = 0;

        vector<int> pe(V + 1);
        vector<bool> inq(V + 1);
        vector<Cost> dis(V + 1);
        vector<Cap> incf(V + 1);

        auto spfa = [&] () {
            fill(dis.begin(), dis.end(), INF);
            queue<int> q;
            q.push(s); dis[s] = 0; incf[s] = INF; incf[t] = 0;
            while(!q.empty()) {
                int u = q.front(); q.pop();
                inq[u] = false;
                for (int i = h[u]; i != -1; i = e[i].nxt) {
                    int v = e[i].v, _cap = e[i].cap, _cost = e[i].cost;
                    if (_cap == 0 || dis[v] <= dis[u] + _cost) continue;
                    dis[v] = dis[u] + _cost;
                    incf[v] = min(_cap, incf[u]);
                    pe[v] = i;
                    if (!inq[v]) q.push(v), inq[v] = true;
                }
            }
            return incf[t];
        };

        auto update = [&] () {
            flow += incf[t];
            for (int i = t; i != s; i = e[pe[i] ^ 1].v) {
                e[pe[i]].cap -= incf[t];
                e[pe[i] ^ 1].cap += incf[t];
                cost += incf[t] * e[pe[i]].cost;
            }
        };

        while(spfa()) update();

```

```
            return make_pair(flow, cost);
        }

    };

}  // namespace Backlight
```

2.17 SCC

```
namespace Backlight {

struct Graph {
    struct Edge {
        int u, v;
        Edge(){}
        Edge(int _u, int _v): u(_u), v(_v) {}
    };

    int V;
    vector<vector<Edge>> G;

    Graph() : V(0) {}
    Graph(int _V) : V(_V), G(_V + 1) {}

    inline void addarc(int u, int v) {
        assert(1 <= u && u <= V);
        assert(1 <= v && v <= V);
        G[u].push_back(Edge(u, v));
    }

    inline void addedge(int u, int v) {
        addarc(u, v);
        addarc(v, u);
    }

    /*************************************************/
    int scc_clock, scc_cnt;
    vector<int> dfn, low, belong, scc_size;
    vector<bool> ins;
    stack<int> stk;

    void tarjan(int u, int fa) {
        dfn[u] = low[u] = ++scc_clock;
        ins[u] = true;
        stk.push(u);

        // bool flag = false;
        for (Edge& e: G[u]) {
            int v = e.v;
            // if (v == fa && !flag) {
            //     flag = true;
            //     continue;
            // }

            if (!dfn[v]) {
                tarjan(v, u);
                low[u] = min(low[u], low[v]);
            } else if (ins[v]) low[u] = min(low[u], dfn[v]);
        }

        if (dfn[u] == low[u]) {
            ++scc_cnt; scc_size.push_back(0);
            int top;
            do {
```

```
                top = stk.top(); stk.pop();
                ins[top] = false;
                belong[top] = scc_cnt;
                ++scc_size[scc_cnt];
            } while(u != top);
        }
    }

    void build_scc() {
        scc_clock = scc_cnt = 0;
        dfn = vector<int>(V + 1);
        low = vector<int>(V + 1);
        belong = vector<int>(V + 1);
        ins = vector<bool>(V + 1);
        scc_size = vector<int>(1);

        for (int i = 1; i <= V; ++i) {
            if (!dfn[i]) tarjan(i, i);
        }
    }
};

}
```

2.18 SPFA

```
namespace Backlight {

template<typename T>
struct Wraph {
    struct Edge {
        int u, v;
        T w;
        Edge(){}
        Edge(int _u, int _v, T _w): u(_u), v(_v), w(_w) {}
    };

    int V;
    vector<vector<Edge>> G;

    Wraph() : V(0) {}
    Wraph(int _V) : V(_V), G(_V + 1) {}

    inline void addarc(int u, int v, T w) {
        assert(1 <= u && u <= V);
        assert(1 <= v && v <= V);
        G[u].push_back(Edge(u, v, w));
    }

    inline void addedge(int u, int v, T w) {
        addarc(u, v, w);
        addarc(v, u, w);
    }

    /**************************************************/
    vector<T> spfa(int S, T T_MAX) {
        queue<int> q;
        vector<T> dis(V + 1, T_MAX);
        vector<bool> inq(V + 1, 0);
        q.push(S); dis[S] = 0;
        while(!q.empty()) {
            int u = q.front(); q.pop();
            inq[u] = 0;
            for(Edge e: G[u]) {
```

```
                if(dis[e.v] > dis[u] + e.w) {
                    dis[e.v] = dis[u] + e.w;
                    if(!inq[e.v]) {
                        inq[e.v] = 1;
                        q.push(e.v);
                    }
                }
            }
        }
        return dis;
    }
};

}
```

2.19 tree-divide

```
struct Edge {
    int v, w;
    Edge(){}
    Edge(int _v, int _w): v(_v), w(_w) {}
};

vector<Edge> G[N];
inline void addedge(int u, int v, int w) {
    G[u].push_back(Edge(v, w));
    G[v].push_back(Edge(u, w));
}

bool vis[N];
int sz[N], max_sz[N];
void dfs_size(int u, int fa) {
    sz[u] = 1; max_sz[u] = 0;
    for (const Edge& e: G[u]) {
        int v = e.v;
        if (v == fa || vis[v]) continue;
        dfs_size(v, u);
        sz[u] += sz[v];
        max_sz[u] = max(max_sz[u], sz[v]);
    }
}

int Max, rt;
void dfs_root(int r, int u, int fa) {
    max_sz[u] = max(max_sz[u], sz[r] - sz[u]);
    if (Max > max_sz[u]) Max = max_sz[u], rt = u;
    for (const Edge& e: G[u]) {
        int v = e.v;
        if (v == fa || vis[v]) continue;
        dfs_root(r, v, u);
    }
}

int dcnt, dis[N];
void dfs_dis(int u, int fa, int d) {
    dis[++dcnt] = d;
    for (const Edge& e: G[u]) {
        int v = e.v, w = e.w;
        if (v == fa || vis[v]) continue;
        dfs_dis(v, u, d + w);
    }
}

int ans[K];
```

```
void calc(int u, int w, int delta) {
    dcnt = 0; dfs_dis(u, -1, w);
    for (int i = 1; i <= dcnt; ++i) {
        for (int j = i + 1; j <= dcnt; ++j) {
            ans[dis[i] + dis[j]] += delta;
        }
    }
}

int n, m;
void DFS(int u) {
    Max = n; dfs_size(u, -1); dfs_root(u, u, -1);
    vis[rt] = 1;
    calc(rt, 0, 1);
    for (const Edge& e: G[rt]) {
        int v = e.v, w = e.w;
        if (vis[v]) continue;
        calc(v, w, -1);
        DFS(v);
    }
}

void solve() {
    read(n, m);

    int u, v, w;
    FOR(i, 2, n) {
        read(u, v, w);
        addedge(u, v, w);
    }

    DFS(1);

    int k;
    FOR(i, 1, m) {
        read(k);
        puts(ans[k] ? "AYE" : "NAY");
    }
}
```

2.20 Wraph

```
namespace Backlight {

template<typename T>
struct Wraph {
    struct Edge {
        int u, v;
        T w;
        Edge(){}
        Edge(int _u, int _v, T _w): u(_u), v(_v), w(_w) {}
    };

    int V;
    vector<vector<Edge>> G;

    Wraph() : V(0) {}
    Wraph(int _V) : V(_V), G(_V + 1) {}

    inline void addarc(int u, int v, T w = 1) {
        assert(1 <= u && u <= V);
        assert(1 <= v && v <= V);
        G[u].push_back(Edge(u, v, w));
    }
```

```
23
24      inline void addedge(int u, int v, T w = 1) {
25          addarc(u, v, w);
26          addarc(v, u, w);
27      }
28  };
29
30  }
```

2.21 WraphMatch

```
1   // Got this code from UOJ
2   #include <bits/stdc++.h>
3   using namespace std;
4
5   template <typename CostType, typename TotalCostType = int64_t>
6   class MaximumWeightedMatching {
7       /*
8       Maximum Weighted Matching in General Graphs.
9       - O(nm log(n)) time
10      - O(n + m) space
11
12      Note: each vertex is 1-indexed.
13    */
14  public:
15      using cost_t = CostType;
16      using tcost_t = TotalCostType;
17
18  private:
19      enum Label { kSeparated = -2,
20          kInner = -1,
21          kFree = 0,
22          kOuter = 1 };
23      static constexpr cost_t Inf = cost_t(1) << (sizeof(cost_t) * 8 - 2);
24
25  private:
26      template <typename T>
27      class BinaryHeap {
28      public:
29          struct Node {
30              bool operator<(const Node& rhs) const { return value < rhs.value; }
31              T value;
32              int id;
33          };
34          BinaryHeap() { }
35          BinaryHeap(int N)
36              : size_(0)
37              , node(N + 1)
38              , index(N, 0)
39          {
40          }
41          int size() const { return size_; }
42          bool empty() const { return size_ == 0; }
43          void clear()
44          {
45              while (size_ > 0)
46                  index[node[size_--].id] = 0;
47          }
48          T min() const { return node[1].value; }
49          int argmin() const { return node[1].id; } // argmin ?
50          T get_val(int id) const { return node[index[id]].value; }
51          void pop()
52          {
53              if (size_ > 0)
```

```
            pop(1);
        }
        void erase(int id)
        {
            if (index[id])
                pop(index[id]);
        }
        bool has(int id) const { return index[id] != 0; }
        void update(int id, T v)
        {
            if (!has(id))
                return push(id, v);
            bool up = (v < node[index[id]].value);
            node[index[id]].value = v;
            if (up)
                up_heap(index[id]);
            else
                down_heap(index[id]);
        }
        void decrease_key(int id, T v)
        {
            if (!has(id))
                return push(id, v);
            if (v < node[index[id]].value)
                node[index[id]].value = v, up_heap(index[id]);
        }
        void push(int id, T v)
        {
            // assert(!has(id));
            index[id] = ++size_;
            node[size_] = { v, id };
            up_heap(size_);
        }

    private:
        void pop(int pos)
        {
            index[node[pos].id] = 0;
            if (pos == size_) {
                --size_;
                return;
            }
            bool up = (node[size_].value < node[pos].value);
            node[pos] = node[size_--];
            index[node[pos].id] = pos;
            if (up)
                up_heap(pos);
            else
                down_heap(pos);
        }
        void swap_node(int a, int b)
        {
            swap(node[a], node[b]);
            index[node[a].id] = a;
            index[node[b].id] = b;
        }
        void down_heap(int pos)
        {
            for (int k = pos, nk = k; 2 * k <= size_; k = nk) {
                if (node[2 * k] < node[nk])
                    nk = 2 * k;
                if (2 * k + 1 <= size_ && node[2 * k + 1] < node[nk])
                    nk = 2 * k + 1;
                if (nk == k)
                    break;
```

```
119                 swap_node(k, nk);
120             }
121         }
122         void up_heap(int pos)
123         {
124             for (int k = pos; k > 1 && node[k] < node[k >> 1]; k >>= 1)
125                 swap_node(k, k >> 1);
126         }
127         int size_;
128         vector<Node> node;
129         vector<int> index;
130     };
131
132     template <typename Key>
133     class PairingHeaps {
134     private:
135         struct Node {
136             Node()
137                 : prev(-1)
138             {
139             } // "prev < 0" means the node is unused.
140             Node(Key v)
141                 : key(v)
142                 , child(0)
143                 , next(0)
144                 , prev(0)
145             {
146             }
147             Key key;
148             int child, next, prev;
149         };
150
151     public:
152         PairingHeaps(int H, int N)
153             : heap(H)
154             , node(N)
155         {
156             // It consists of `H` Pairing heaps.
157             // Each heap-node ID can appear at most 1 time(s) among heaps
158             // and should be in [1, N).
159         }
160
161         void clear(int h)
162         {
163             if (heap[h])
164                 clear_rec(heap[h]), heap[h] = 0;
165         }
166         void clear_all()
167         {
168             for (size_t i = 0; i < heap.size(); ++i)
169                 heap[i] = 0;
170             for (size_t i = 0; i < node.size(); ++i)
171                 node[i] = Node();
172         }
173         bool empty(int h) const { return !heap[h]; }
174         bool used(int v) const { return node[v].prev >= 0; }
175         Key min(int h) const { return node[heap[h]].key; }
176         int argmin(int h) const { return heap[h]; }
177
178         void pop(int h)
179         {
180             // assert(!empty(h));
181             erase(h, heap[h]);
182         }
183         void push(int h, int v, Key key)
```

```
184         {
185             // assert(!used(v));
186             node[v] = Node(key);
187             heap[h] = merge(heap[h], v);
188         }
189         void erase(int h, int v)
190         {
191             if (!used(v))
192                 return;
193             int w = two_pass_pairing(node[v].child);
194             if (!node[v].prev)
195                 heap[h] = w;
196             else {
197                 cut(v);
198                 heap[h] = merge(heap[h], w);
199             }
200             node[v].prev = -1;
201         }
202         void decrease_key(int h, int v, Key key)
203         {
204             if (!used(v))
205                 return push(h, v, key);
206             if (!node[v].prev)
207                 node[v].key = key;
208             else {
209                 cut(v);
210                 node[v].key = key;
211                 heap[h] = merge(heap[h], v);
212             }
213         }
214 
215     private:
216         void clear_rec(int v)
217         {
218             for (; v; v = node[v].next) {
219                 if (node[v].child)
220                     clear_rec(node[v].child);
221                 node[v].prev = -1;
222             }
223         }
224 
225         inline void cut(int v)
226         {
227             auto& n = node[v];
228             int pv = n.prev, nv = n.next;
229             auto& pn = node[pv];
230             if (pn.child == v)
231                 pn.child = nv;
232             else
233                 pn.next = nv;
234             node[nv].prev = pv;
235             n.next = n.prev = 0;
236         }
237 
238         int merge(int l, int r)
239         {
240             if (!l)
241                 return r;
242             if (!r)
243                 return l;
244             if (node[l].key > node[r].key)
245                 swap(l, r);
246             int lc = node[r].next = node[l].child;
247             node[l].child = node[lc].prev = r;
248             return node[r].prev = l;
```

```
249         }
250
251         int two_pass_pairing(int root)
252         {
253             if (!root)
254                 return 0;
255             int a = root;
256             root = 0;
257             while (a) {
258                 int b = node[a].next, na = 0;
259                 node[a].prev = node[a].next = 0;
260                 if (b)
261                     na = node[b].next, node[b].prev = node[b].next = 0;
262                 a = merge(a, b);
263                 node[a].next = root;
264                 root = a;
265                 a = na;
266             }
267             int s = node[root].next;
268             node[root].next = 0;
269             while (s) {
270                 int t = node[s].next;
271                 node[s].next = 0;
272                 root = merge(root, s);
273                 s = t;
274             }
275             return root;
276         }
277
278     private:
279         vector<int> heap;
280         vector<Node> node;
281     };
282
283     template <typename T>
284     struct PriorityQueue : public priority_queue<T, vector<T>, greater<T>> {
285         PriorityQueue() { }
286         PriorityQueue(int N) { this->c.reserve(N); }
287         T min() { return this->top(); }
288         void clear() { this->c.clear(); }
289     };
290
291     template <typename T>
292     struct Queue {
293         Queue() { }
294         Queue(int N)
295             : qh(0)
296             , qt(0)
297             , data(N)
298         {
299         }
300         T operator[](int i) const { return data[i]; }
301         void enqueue(int u) { data[qt++] = u; }
302         int dequeue() { return data[qh++]; }
303         bool empty() const { return qh == qt; }
304         void clear() { qh = qt = 0; }
305         int size() const { return qt; }
306         int qh, qt;
307         vector<T> data;
308     };
309
310 public:
311     struct InputEdge {
312         int from, to;
313         cost_t cost;
```

```
    };

private:
    template <typename T>
    using ModifiableHeap = BinaryHeap<T>;
    template <typename T>
    using ModifiableHeaps = PairingHeaps<T>;
    template <typename T>
    using FastHeap = PriorityQueue<T>;

    struct Edge {
        int to;
        cost_t cost;
    };
    struct Link {
        int from, to;
    };
    struct Node {
        struct NodeLink {
            int b, v;
        };
        Node() { }
        Node(int u)
            : parent(0)
            , size(1)
        {
            link[0] = link[1] = { u, u };
        }
        int next_v() const { return link[0].v; }
        int next_b() const { return link[0].b; }
        int prev_v() const { return link[1].v; }
        int prev_b() const { return link[1].b; }
        int parent, size;
        NodeLink link[2];
    };
    struct Event {
        Event() { }
        Event(cost_t time, int id)
            : time(time)
            , id(id)
        {
        }
        bool operator<(const Event& rhs) const { return time < rhs.time; }
        bool operator>(const Event& rhs) const { return time > rhs.time; }
        cost_t time;
        int id;
    };
    struct EdgeEvent {
        EdgeEvent() { }
        EdgeEvent(cost_t time, int from, int to)
            : time(time)
            , from(from)
            , to(to)
        {
        }
        bool operator>(const EdgeEvent& rhs) const { return time > rhs.time; }
        bool operator<(const EdgeEvent& rhs) const { return time < rhs.time; }
        cost_t time;
        int from, to;
    };

public:
    MaximumWeightedMatching(int N, const vector<InputEdge>& in)
        : N(N)
        , B((N - 1) / 2)
```

```
        , S(N + B + 1)
        , ofs(N + 2)
        , edges(in.size() * 2)
        , heap2(S)
        , heap2s(S, S)
        , heap3(edges.size())
        , heap4(S)
    {

        for (auto& e : in)
            ofs[e.from + 1]++, ofs[e.to + 1]++;
        for (int i = 1; i <= N + 1; ++i)
            ofs[i] += ofs[i - 1];
        for (auto& e : in) {
            edges[ofs[e.from]++] = { e.to, e.cost * 2 };
            edges[ofs[e.to]++] = { e.from, e.cost * 2 };
        }
        for (int i = N + 1; i > 0; --i)
            ofs[i] = ofs[i - 1];
        ofs[0] = 0;
    }

    pair<tcost_t, vector<int>> maximum_weighted_matching(bool init_matching = false)
    {
        initialize();
        set_potential();
        if (init_matching)
            find_maximal_matching();
        for (int u = 1; u <= N; ++u)
            if (!mate[u])
                do_edmonds_search(u);
        tcost_t ret = compute_optimal_value();
        return make_pair(ret, mate);
    }

private:
    tcost_t compute_optimal_value() const
    {
        tcost_t ret = 0;
        for (int u = 1; u <= N; ++u)
            if (mate[u] > u) {
                cost_t max_c = 0;
                for (int eid = ofs[u]; eid < ofs[u + 1]; ++eid) {
                    if (edges[eid].to == mate[u])
                        max_c = max(max_c, edges[eid].cost);
                }
                ret += max_c;
            }
        return ret >> 1;
    }

    inline tcost_t reduced_cost(int u, int v, const Edge& e) const
    {
        return tcost_t(potential[u]) + potential[v] - e.cost;
    }

    void rematch(int v, int w)
    {
        int t = mate[v];
        mate[v] = w;
        if (mate[t] != v)
            return;
        if (link[v].to == surface[link[v].to]) {
            mate[t] = link[v].from;
            rematch(mate[t], t);
```

```
444         } else {
445             int x = link[v].from, y = link[v].to;
446             rematch(x, y);
447             rematch(y, x);
448         }
449     }
450 
451     void fix_mate_and_base(int b)
452     {
453         if (b <= N)
454             return;
455         int bv = base[b], mv = node[bv].link[0].v, bmv = node[bv].link[0].b;
456         int d = (node[bmv].link[1].v == mate[mv]) ? 0 : 1;
457         while (1) {
458             int mv = node[bv].link[d].v, bmv = node[bv].link[d].b;
459             if (node[bmv].link[1 ^ d].v != mate[mv])
460                 break;
461             fix_mate_and_base(bv);
462             fix_mate_and_base(bmv);
463             bv = node[bmv].link[d].b;
464         }
465         fix_mate_and_base(base[b] = bv);
466         mate[b] = mate[bv];
467     }
468 
469     void reset_time()
470     {
471         time_current_ = 0;
472         event1 = { Inf, 0 };
473     }
474 
475     void reset_blossom(int b)
476     {
477         label[b] = kFree;
478         link[b].from = 0;
479         slack[b] = Inf;
480         lazy[b] = 0;
481     }
482 
483     void reset_all()
484     {
485         label[0] = kFree;
486         link[0].from = 0;
487         for (int v = 1; v <= N; ++v) { // should be optimized for sparse graphs.
488             if (label[v] == kOuter)
489                 potential[v] -= time_current_;
490             else {
491                 int bv = surface[v];
492                 potential[v] += lazy[bv];
493                 if (label[bv] == kInner)
494                     potential[v] += time_current_ - time_created[bv];
495             }
496             reset_blossom(v);
497         }
498         for (int b = N + 1, r = B - unused_bid_idx_; r > 0 && b < S; ++b)
499             if (base[b] != b) {
500                 if (surface[b] == b) {
501                     fix_mate_and_base(b);
502                     if (label[b] == kOuter)
503                         potential[b] += (time_current_ - time_created[b]) << 1;
504                     else if (label[b] == kInner)
505                         fix_blossom_potential<kInner>(b);
506                     else
507                         fix_blossom_potential<kFree>(b);
508                 }
```

```
                heap2s.clear(b);
                reset_blossom(b);
                --r;
            }

        que.clear();
        reset_time();
        heap2.clear();
        heap3.clear();
        heap4.clear();
    }

    void do_edmonds_search(int root)
    {
        if (potential[root] == 0)
            return;
        link_blossom(surface[root], { 0, 0 });
        push_outer_and_fix_potentials(surface[root], 0);
        for (bool augmented = false; !augmented;) {
            augmented = augment(root);
            if (augmented)
                break;
            augmented = adjust_dual_variables(root);
        }
        reset_all();
    }

    template <Label Lab>
    inline cost_t fix_blossom_potential(int b)
    {
        // Return the amount.
        // (If v is an atom, the potential[v] will not be changed.)
        cost_t d = lazy[b];
        lazy[b] = 0;
        if (Lab == kInner) {
            cost_t dt = time_current_ - time_created[b];
            if (b > N)
                potential[b] -= dt << 1;
            d += dt;
        }
        return d;
    }

    template <Label Lab>
    inline void update_heap2(int x, int y, int by, cost_t t)
    {
        if (t >= slack[y])
            return;
        slack[y] = t;
        best_from[y] = x;
        if (y == by) {
            if (Lab != kInner)
                heap2.decrease_key(y, EdgeEvent(t + lazy[y], x, y));
        } else {
            int gy = group[y];
            if (gy != y) {
                if (t >= slack[gy])
                    return;
                slack[gy] = t;
            }
            heap2s.decrease_key(by, gy, EdgeEvent(t, x, y));
            if (Lab == kInner)
                return;
            EdgeEvent m = heap2s.min(by);
            heap2.decrease_key(by, EdgeEvent(m.time + lazy[by], m.from, m.to));
```

```
        }
    }

    void activate_heap2_node(int b)
    {
        if (b <= N) {
            if (slack[b] < Inf)
                heap2.push(b, EdgeEvent(slack[b] + lazy[b], best_from[b], b));
        } else {
            if (heap2s.empty(b))
                return;
            EdgeEvent m = heap2s.min(b);
            heap2.push(b, EdgeEvent(m.time + lazy[b], m.from, m.to));
        }
    }

    void swap_blossom(int a, int b)
    {
        // Assume that `b` is a maximal blossom.
        swap(base[a], base[b]);
        if (base[a] == a)
            base[a] = b;
        swap(heavy[a], heavy[b]);
        if (heavy[a] == a)
            heavy[a] = b;
        swap(link[a], link[b]);
        swap(mate[a], mate[b]);
        swap(potential[a], potential[b]);
        swap(lazy[a], lazy[b]);
        swap(time_created[a], time_created[b]);
        for (int d = 0; d < 2; ++d)
            node[node[a].link[d].b].link[1 ^ d].b = b;
        swap(node[a], node[b]);
    }

    void set_surface_and_group(int b, int sf, int g)
    {
        surface[b] = sf, group[b] = g;
        if (b <= N)
            return;
        for (int bb = base[b]; surface[bb] != sf; bb = node[bb].next_b()) {
            set_surface_and_group(bb, sf, g);
        }
    }

    void merge_smaller_blossoms(int bid)
    {
        int lb = bid, largest_size = 1;
        for (int beta = base[bid], b = beta;;) {
            if (node[b].size > largest_size)
                largest_size = node[b].size, lb = b;
            if ((b = node[b].next_b()) == beta)
                break;
        }
        for (int beta = base[bid], b = beta;;) {
            if (b != lb)
                set_surface_and_group(b, lb, b);
            if ((b = node[b].next_b()) == beta)
                break;
        }
        group[lb] = lb;
        if (largest_size > 1) {
            surface[bid] = heavy[bid] = lb;
            swap_blossom(lb, bid);
        } else
```

```
            heavy[bid] = 0;
    }

    void contract(int x, int y, int eid)
    {
        int bx = surface[x], by = surface[y];
        assert(bx != by);
        const int h = -(eid + 1);
        link[surface[mate[bx]]].from = link[surface[mate[by]]].from = h;

        int lca = -1;
        while (1) {
            if (mate[by] != 0)
                swap(bx, by);
            bx = lca = surface[link[bx].from];
            if (link[surface[mate[bx]]].from == h)
                break;
            link[surface[mate[bx]]].from = h;
        }

        const int bid = unused_bid[--unused_bid_idx_];
        assert(unused_bid_idx_ >= 0);
        int tree_size = 0;
        for (int d = 0; d < 2; ++d) {
            for (int bv = surface[x]; bv != lca;) {
                int mv = mate[bv], bmv = surface[mv], v = mate[mv];
                int f = link[v].from, t = link[v].to;
                tree_size += node[bv].size + node[bmv].size;
                link[mv] = { x, y };

                if (bv > N)
                    potential[bv] += (time_current_ - time_created[bv]) << 1;
                if (bmv > N)
                    heap4.erase(bmv);
                push_outer_and_fix_potentials(bmv, fix_blossom_potential<kInner>(bmv));

                node[bv].link[d] = { bmv, mv };
                node[bmv].link[1 ^ d] = { bv, v };
                node[bmv].link[d] = { bv = surface[f], f };
                node[bv].link[1 ^ d] = { bmv, t };
            }
            node[surface[x]].link[1 ^ d] = { surface[y], y };
            swap(x, y);
        }
        if (lca > N)
            potential[lca] += (time_current_ - time_created[lca]) << 1;
        node[bid].size = tree_size + node[lca].size;
        base[bid] = lca;
        link[bid] = link[lca];
        mate[bid] = mate[lca];
        label[bid] = kOuter;
        surface[bid] = bid;
        time_created[bid] = time_current_;
        potential[bid] = 0;
        lazy[bid] = 0;

        merge_smaller_blossoms(bid); // O(n log n) time / Edmonds search
    }

    void link_blossom(int v, Link l)
    {
        link[v] = { l.from, l.to };
        if (v <= N)
            return;
        int b = base[v];
```

```
        link_blossom(b, l);
        int pb = node[b].prev_b();
        l = { node[pb].next_v(), node[b].prev_v() };
        for (int bv = b;;) {
            int bw = node[bv].next_b();
            if (bw == b)
                break;
            link_blossom(bw, l);
            Link nl = { node[bw].prev_v(), node[bv].next_v() };
            bv = node[bw].next_b();
            link_blossom(bv, nl);
        }
    }

    void push_outer_and_fix_potentials(int v, cost_t d)
    {
        label[v] = kOuter;
        if (v > N) {
            for (int b = base[v]; label[b] != kOuter; b = node[b].next_b()) {
                push_outer_and_fix_potentials(b, d);
            }
        } else {
            potential[v] += time_current_ + d;
            if (potential[v] < event1.time)
                event1 = { potential[v], v };
            que.enqueue(v);
        }
    }

    bool grow(int root, int x, int y)
    {
        int by = surface[y];
        bool visited = (label[by] != kFree);
        if (!visited)
            link_blossom(by, { 0, 0 });
        label[by] = kInner;
        time_created[by] = time_current_;
        heap2.erase(by);
        if (y != by)
            heap4.update(by, time_current_ + (potential[by] >> 1));
        int z = mate[by];
        if (z == 0 && by != surface[root]) {
            rematch(x, y);
            rematch(y, x);
            return true;
        }
        int bz = surface[z];
        if (!visited)
            link_blossom(bz, { x, y });
        else
            link[bz] = link[z] = { x, y };
        push_outer_and_fix_potentials(bz, fix_blossom_potential<kFree>(bz));
        time_created[bz] = time_current_;
        heap2.erase(bz);
        return false;
    }

    void free_blossom(int bid)
    {
        unused_bid[unused_bid_idx_++] = bid;
        base[bid] = bid;
    }

    int recalculate_minimum_slack(int b, int g)
    {
```

```
        // Return the destination of the best edge of blossom `g`.
        if (b <= N) {
            if (slack[b] >= slack[g])
                return 0;
            slack[g] = slack[b];
            best_from[g] = best_from[b];
            return b;
        }
        int v = 0;
        for (int beta = base[b], bb = beta;;) {
            int w = recalculate_minimum_slack(bb, g);
            if (w != 0)
                v = w;
            if ((bb = node[bb].next_b()) == beta)
                break;
        }
        return v;
    }

    void construct_smaller_components(int b, int sf, int g)
    {
        surface[b] = sf, group[b] = g; // `group[b] = g` is unneeded.
        if (b <= N)
            return;
        for (int bb = base[b]; surface[bb] != sf; bb = node[bb].next_b()) {
            if (bb == heavy[b]) {
                construct_smaller_components(bb, sf, g);
            } else {
                set_surface_and_group(bb, sf, bb);
                int to = 0;
                if (bb > N)
                    slack[bb] = Inf, to = recalculate_minimum_slack(bb, bb);
                else if (slack[bb] < Inf)
                    to = bb;
                if (to > 0)
                    heap2s.push(sf, bb, EdgeEvent(slack[bb], best_from[bb], to));
            }
        }
    }

    void move_to_largest_blossom(int bid)
    {
        const int h = heavy[bid];
        cost_t d = (time_current_ - time_created[bid]) + lazy[bid];
        lazy[bid] = 0;
        for (int beta = base[bid], b = beta;;) {
            time_created[b] = time_current_;
            lazy[b] = d;
            if (b != h)
                construct_smaller_components(b, b, b), heap2s.erase(bid, b);
            if ((b = node[b].next_b()) == beta)
                break;
        }
        if (h > 0)
            swap_blossom(h, bid), bid = h;
        free_blossom(bid);
    }

    void expand(int bid)
    {
        int mv = mate[base[bid]];
        move_to_largest_blossom(bid); // O(n log n) time / Edmonds search
        Link old_link = link[mv];
        int old_base = surface[mate[mv]], root = surface[old_link.to];
        int d = (mate[root] == node[root].link[0].v) ? 1 : 0;
```

```
        for (int b = node[old_base].link[d ^ 1].b; b != root;) {
            label[b] = kSeparated;
            activate_heap2_node(b);
            b = node[b].link[d ^ 1].b;
            label[b] = kSeparated;
            activate_heap2_node(b);
            b = node[b].link[d ^ 1].b;
        }
        for (int b = old_base;; b = node[b].link[d].b) {
            label[b] = kInner;
            int nb = node[b].link[d].b;
            if (b == root)
                link[mate[b]] = old_link;
            else
                link[mate[b]] = { node[b].link[d].v, node[nb].link[d ^ 1].v };
            link[surface[mate[b]]] = link[mate[b]]; // fix tree links
            if (b > N) {
                if (potential[b] == 0)
                    expand(b);
                else
                    heap4.push(b, time_current_ + (potential[b] >> 1));
            }
            if (b == root)
                break;
            push_outer_and_fix_potentials(nb, fix_blossom_potential<kInner>(b = nb));
        }
    }

    bool augment(int root)
    {
        // Return true if an augmenting path is found.
        while (!que.empty()) {
            int x = que.dequeue(), bx = surface[x];
            if (potential[x] == time_current_) {
                if (x != root)
                    rematch(x, 0);
                return true;
            }
            for (int eid = ofs[x]; eid < ofs[x + 1]; ++eid) {
                auto& e = edges[eid];
                int y = e.to, by = surface[y];
                if (bx == by)
                    continue;
                Label l = label[by];
                if (l == kOuter) {
                    cost_t t = reduced_cost(x, y, e) >> 1; // < 2 * Inf
                    if (t == time_current_) {
                        contract(x, y, eid);
                        bx = surface[x];
                    } else if (t < event1.time) {
                        heap3.emplace(t, x, eid);
                    }
                } else {
                    tcost_t t = reduced_cost(x, y, e); // < 3 * Inf
                    if (t >= Inf)
                        continue;
                    if (l != kInner) {
                        if (cost_t(t) + lazy[by] == time_current_) {
                            if (grow(root, x, y))
                                return true;
                        } else
                            update_heap2<kFree>(x, y, by, t);
                    } else {
                        if (mate[x] != y)
                            update_heap2<kInner>(x, y, by, t);
```

```
                }
            }
        }
    }
    return false;
}

bool adjust_dual_variables(int root)
{
    // delta1 : rematch
    cost_t time1 = event1.time;

    // delta2 : grow
    cost_t time2 = Inf;
    if (!heap2.empty())
        time2 = heap2.min().time;

    // delta3 : contract : O(m log n) time / Edmonds search [ bottleneck (?) ]
    cost_t time3 = Inf;
    while (!heap3.empty()) {
        EdgeEvent e = heap3.min();
        int x = e.from, y = edges[e.to].to; // e.to is some edge id.
        if (surface[x] != surface[y]) {
            time3 = e.time;
            break;
        } else
            heap3.pop();
    }

    // delta4 : expand
    cost_t time4 = Inf;
    if (!heap4.empty())
        time4 = heap4.min();

    // -- events --
    cost_t time_next = min(min(time1, time2), min(time3, time4));
    assert(time_current_ <= time_next && time_next < Inf);
    time_current_ = time_next;

    if (time_current_ == event1.time) {
        int x = event1.id;
        if (x != root)
            rematch(x, 0);
        return true;
    }
    while (!heap2.empty() && heap2.min().time == time_current_) {
        int x = heap2.min().from, y = heap2.min().to;
        if (grow(root, x, y))
            return true; // `grow` function will call `heap2.erase(by)`.
    }
    while (!heap3.empty() && heap3.min().time == time_current_) {
        int x = heap3.min().from, eid = heap3.min().to;
        int y = edges[eid].to;
        heap3.pop();
        if (surface[x] == surface[y])
            continue;
        contract(x, y, eid);
    }
    while (!heap4.empty() && heap4.min() == time_current_) {
        int b = heap4.argmin();
        heap4.pop();
        expand(b);
    }
    return false;
}
```

```
964
965 private:
966     void initialize()
967     {
968         que = Queue<int>(N);
969         mate.assign(S, 0);
970         link.assign(S, { 0, 0 });
971         label.assign(S, kFree);
972         base.resize(S);
973         for (int u = 1; u < S; ++u)
974             base[u] = u;
975         surface.resize(S);
976         for (int u = 1; u < S; ++u)
977             surface[u] = u;
978
979         potential.resize(S);
980         node.resize(S);
981         for (int b = 1; b < S; ++b)
982             node[b] = Node(b);
983
984         unused_bid.resize(B);
985         for (int i = 0; i < B; ++i)
986             unused_bid[i] = N + B - i;
987         unused_bid_idx_ = B;
988
989         // for O(nm log n) implementation
990         reset_time();
991         time_created.resize(S);
992         slack.resize(S);
993         for (int i = 0; i < S; ++i)
994             slack[i] = Inf;
995         best_from.assign(S, 0);
996         heavy.assign(S, 0);
997         lazy.assign(S, 0);
998         group.resize(S);
999         for (int i = 0; i < S; ++i)
1000            group[i] = i;
1001    }
1002
1003    void set_potential()
1004    {
1005        for (int u = 1; u <= N; ++u) {
1006            cost_t max_c = 0;
1007            for (int eid = ofs[u]; eid < ofs[u + 1]; ++eid) {
1008                max_c = max(max_c, edges[eid].cost);
1009            }
1010            potential[u] = max_c >> 1;
1011        }
1012    }
1013
1014    void find_maximal_matching()
1015    {
1016        // Find a maximal matching naively.
1017        for (int u = 1; u <= N; ++u)
1018            if (!mate[u]) {
1019                for (int eid = ofs[u]; eid < ofs[u + 1]; ++eid) {
1020                    auto& e = edges[eid];
1021                    int v = e.to;
1022                    if (mate[v] > 0 || reduced_cost(u, v, e) > 0)
1023                        continue;
1024                    mate[u] = v;
1025                    mate[v] = u;
1026                    break;
1027                }
1028            }
```

```
    }

private:
    int N, B, S; // N = |V|, B = (|V| - 1) / 2, S = N + B + 1
    vector<int> ofs;
    vector<Edge> edges;

    Queue<int> que;
    vector<int> mate, surface, base;
    vector<Link> link;
    vector<Label> label;
    vector<cost_t> potential;

    vector<int> unused_bid;
    int unused_bid_idx_;
    vector<Node> node;

    // for O(nm log n) implementation
    vector<int> heavy, group;
    vector<cost_t> time_created, lazy, slack;
    vector<int> best_from;

    cost_t time_current_;
    Event event1;
    ModifiableHeap<EdgeEvent> heap2;
    ModifiableHeaps<EdgeEvent> heap2s;
    FastHeap<EdgeEvent> heap3;
    ModifiableHeap<cost_t> heap4;
};

using MWM = MaximumWeightedMatching<int>;
using Edge = MWM::InputEdge;

int main()
{
        ios::sync_with_stdio(false); cin.tie(0); cout.tie(0);
        int N, M;
        cin >> N >> M;
        vector<Edge> edges(2 * M);
        vector<int> ou(N + 2), ov(N + 2);

        int u, v, c;
        for (int i = 0; i < M; ++i) {
                cin >> u >> v >> c;
                edges[i] = { u, v, c };
                ou[u + 1] += 1;
                ov[v + 1] += 1;
        }
        for (int i = 1; i <= N + 1; ++i)
                ov[i] += ov[i - 1];
        for (int i = 0; i < M; ++i)
                edges[M + (ov[edges[i].to]++)] = edges[i];
        for (int i = 1; i <= N + 1; ++i)
                ou[i] += ou[i - 1];
        for (int i = 0; i < M; ++i)
                edges[ou[edges[i + M].from]++] = edges[i + M];
        edges.resize(M);

        auto ans = MWM(N, edges).maximum_weighted_matching();
        cout << ans.first << endl;
        for (int i = 1; i <= N; ++i) {
                cout << ans.second[i] << (i == N ? '\n' : ' ');
        }
    return 0;
}
```

3 math

3.1 2DGeometry

```cpp
namespace Geometry
{
    // 定义以及防止精度出错
    const double eps = 1e-8;
    const double inf = 1e9;
    const double pi = acos(-1.0);

    inline int sgn(double x) {
        if(fabs(x) < eps) return 0;
        if(x < 0) return -1;
        return 1;
    }

    // 单位换算
    inline double degree2radian(const double& alpha) {
        return alpha / 180 * pi;
    }

    inline double radian2degree(const double& alpha) {
        return alpha / pi * 180;
    }

    // 点 (向量)
    // 也是远点到该点的向量
    struct point
    {
        double x, y;
        point(double _x = 0, double _y = 0) : x(_x), y(_y) {}

        point operator - (const point& b) const {
            return point(x - b.x, y - b.y);
        }

        point operator + (const point& b) const {
            return point(x + b.x, y + b.y);
        }

        bool operator < (const point& b) const {
            return sgn(x - b.x) == 0 ? sgn(y - b.y) < 0 : sgn(x - b.x) < 0;
        }

        bool operator == (const point& b) const {
            return sgn(x - b.x) == 0 && sgn(y - b.y) == 0;
        }

        point operator * (const double& b) {
            return point(x * b, y * b);
        }

        point operator / (const double& b) {
            return point(x / b, y / b);
        }

        // 绕原点逆时针旋转，给出正弦和余弦值
        // 若绕另一点 p，则先转换成以 p 为原点，完成旋转，再转换回来
        void transxy(const double& sinb, const double& cosb) {
            double tx = x, ty = y;
            x = tx * cosb - ty * sinb;
            y = tx * sinb + ty * cosb;
        }

```

```
62          // 绕原点逆时针旋转，给出旋转弧度
63          void transxy(const double& b) {
64              double tx = x, ty = y;
65              x=tx * cos(b) - ty * sin(b);
66              y=tx * sin(b) + ty * cos(b);
67          }
68
69          // 逆时针旋转 90 度
70          point trans90() {
71              return point(-y, x);
72          }
73
74          // 顺时针旋转 90 度
75          point trans270() {
76              return point(y, -x);
77          }
78
79          // 与原点的距离
80          // a,b 之间的距离: (b- a).length()
81          double length() {
82              return sqrt(x * x + y * y);
83          }
84
85          // 与原点的距离的平方
86          double length2() {
87              return x * x + y * y;
88          }
89
90          // 与点 a 之间的距离
91          double disTo(const point& a) {
92              return (a - *this).length();
93          }
94
95          // 与 x 轴正方向的夹角，单位为弧度
96          double alpha() {
97              return atan2(y, x);
98          }
99
100         // 单位向量
101         point unit() {
102             return point(x, y) / length();
103         }
104     };
105
106     // 向量 Oa 和向量 Ob 的叉积
107     inline double det(const point& a,const point& b) {
108         return a.x * b.y - a.y * b.x;
109     }
110
111     // 向量 ab 和向量 ac 的叉积
112     inline double det(const point& a,const point& b,const point& c) {
113         return det(b - a, c - a);
114     }
115
116     // 向量 Oa 和向量 Ob 的点积
117     inline double dot(const point&a,const point& b) {
118         return a.x * b.x + a.y * b.y;
119     }
120
121     // 向量 ab 和向量 ac 的点积
122     inline double dot(const point&a, const point& b,const point& c) {
123         return dot(b - a, c - a);
124     }
125
126     // 两点间距离
```

```
127     inline double distance(const point& a,const point& b) {
128         return (a - b).length();
129     }
130
131     // 两点间距离的平方
132     inline double distance2(const point& a,const point& b) {
133         return (b.x - a.x) * (b.x - a.x) + (b.y - a.y) * (b.y - a.y);
134     }
135
136     // LightOJ1203
137     // 最终答案会在凸包上，然后算 ab 与 ac 的夹角，单位为弧度
138     // ab 与 ac 的夹角
139     double radian(point a, point b, point c) {
140         return fabs(atan2(fabs(det(a, b, c)), dot(a, b, c)));
141     }
142
143     double angle(point a, point b, point c) {
144         double r = radian(a, b, c);
145         return radian2degree(r);
146     }
147
148     // 从点 a，由 b 遮挡，能否看见 c
149     bool canSee(point a, point b, point c) {
150         return sgn(det(a, b, c)) <= 0;
151     }
152
153     // 直线或者线段
154     struct line
155     {
156         point s, e;         // 直线端点
157         double a, b, c;     // ax+by+c=0
158         double k;           // 斜率,[-pi, pi]
159
160         line(point _s = point(), point _e = point()) : s(_s), e(_e) {
161             k = atan2(e.y - s.y, e.x - s.x);
162             a = e.y - s.y;
163             b = s.x - e.x;
164             c = e.x * s.y - e.y * s.x;
165         }
166
167         // ax + by + c = 0;
168         line(const double& _a, const double& _b, const double& _c) : a(_a), b(_b), c(_c) {
169             if (sgn(a) == 0) {
170                 s = point(0, -c / b);
171                 e = point(1, -c / b);
172             } else if (sgn(b) == 0) {
173                 s = point(-c / a, 0);
174                 e = point(-c / a, 1);
175             } else {
176                 s = point(0, -c / b);
177                 e = point(1, (-c - a) / b);
178             }
179         }
180
181         // 点和倾斜角确定直线
182         line(const point& a, const double b) : s(a) {
183             if (sgn(b - pi / 2) == 0) e = s + point(0, 1);
184             else e = s + point(1, tan(b));
185         }
186
187         bool operator == (const line& l) {
188             return (s == l.s) && (e == l.e);
189         }
190
191         void adjust() {
```

```
192             if(e < s) swap(s, e);
193         }
194 
195         double length() {
196             return s.disTo(e);
197         }
198 
199         // 判断点和直线的关系
200         // 1 在直线左侧
201         // 2 在直线右侧
202         // 3 在直线上
203         int relationToPoint(point p) {
204             int c = sgn(det(s, p, e));
205             if(c < 0) return 1;
206             else if(c > 0) return 2;
207             else return 3;
208         }
209 
210         // 判断点 p 是否在线段上
211         bool isPointOnLine(const point& p) {
212             return sgn(det(p - s, e - s)) == 0 && sgn(det(p - s, p - e)) <= 0;
213         }
214 
215         // 判断两直线是否平行
216         bool parallelTo(line l) {
217             return sgn(det(e - s, l.e - l.s)) == 0;
218         }
219 
220         // 线段相交判断
221         // 0 不相交
222         // 1 交点是端点
223         // 2 交点不是端点
224         int isSegCrossSeg(line l) {
225             int d1 = sgn(det(s, e, l.s));
226             int d2 = sgn(det(s, e, l.e));
227             int d3 = sgn(det(l.s, l.e, s));
228             int d4 = sgn(det(l.s, l.e, e));
229             if((d1^d2) == -2 && (d3^d4) == -2) return 2;
230             return (d1 == 0 && sgn(dot(l.s - s, l.s - e)) <= 0)
231                 || (d2 == 0 && sgn(dot(l.e - s, l.e - e)) <= 0)
232                 || (d3 == 0 && sgn(dot(s - l.s, s - l.e)) <= 0)
233                 || (d4 == 0 && sgn(dot(e - l.s, e - l.e)) <= 0);
234         }
235 
236         // 直线相交判断
237         // 0 平行
238         // 1 重合
239         // 2 相交
240         bool isLineCrossLine(line l) {
241             if(parallelTo(l))
242                 return l.relationToPoint(s) == 3;
243             return 2;
244         }
245 
246         // 本直线与线段 v 相交判断
247         // 0 不相交
248         // 1 交点是端点
249         // 2 交点不是端点
250         int isLineCrossSeg(line seg) {
251             int d1 = sgn(det(s, e, seg.s));
252             int d2 = sgn(det(s, e, seg.e));
253             if((d1^d2) == -2) return 2;
254             return (d1 == 0 || d2 == 0);
255         }
256 
```

```
257         // 求两直线交点
258         // 要求两直线不平行或重合
259         point getCrossPoint(line l) {
260             double a1 = det(l.s,l.e,s);
261             double a2 = -det(l.s,l.e,e);
262             return (s * a2 + e * a1) / (a1 + a2);
263         }
264 
265         // 点到直线的距离
266         double disPointToLine(const point& p) {
267             double d = det(s, p, e) / length();
268             return fabs(d);
269         }
270 
271         // 点到线段的距离
272         double disPointToSeg(const point& p) {
273             if (sgn(dot(s, p, e)) < 0 || sgn(dot(e, p, s)) < 0)
274                 return min(distance(p, s), distance(p, e));
275             return fabs(disPointToLine(p));
276         }
277 
278         // 线段到线段的距离
279         double disSegToSeg(line& l) {
280             if(isSegCrossSeg(l) == 0) {
281                 double d1 = min(disPointToSeg(l.s), disPointToSeg(l.e));
282                 double d2 = min(l.disPointToSeg(s), l.disPointToSeg(e));
283                 return min(d1,d2);
284             }
285             return 0;
286         }
287 
288         // 点在直线上的投影
289         point projectionPointOnLine(const point& p) {
290             return s + (dot(e - s, dot(s, e, p))) / ((e - s).length2());
291         }
292 
293         // 点关于直线的对称点
294         point  symmetryPoint(const point& p) {
295             point q = projectionPointOnLine(p);
296             return point(2 * q.x - p.x, 2 * q.y - p.y);
297         }
298 
299         // 垂直平分线
300         line getVerticalBisector() {
301             point m = (s + e) / 2;
302             double radian = (e - s).alpha() + pi / 2;
303             return line(m, radian);
304         }
305     };
306 
307     point getLineCrossLine(line l1, line l2) {
308         return l1.getCrossPoint(l2);
309     }
310 
311     // 向量表示法, 方向为由 s -> e
312     // struct line
313     // {
314     //     point s,v;
315     //     line(point a=point(),point b=point()) {
316     //         s=a;
317     //         v.x=b.x-a.x;
318     //         v.y=b.y-a.y;
319     //     }
320     // };
321 
```

```cpp
// 圆
struct circle
{
    point p;            // 圆心
    double r;           // 半径

    circle() {}

    circle(point _p, double _r) : p(_p), r(_r) {}
    circle(double _x, double _y, double _r) : p(point(_x, _y)), r(_r) {}

    // 圆上三点确定圈
    circle(point x1, point x2, point x3) {
        double a = x2.x - x1.x;
        double b = x2.y - x1.y;
        double c = x3.x - x2.x;
        double d = x3.y - x2.y;
        double e = x2.x * x2.x + x2.y * x2.y - x1.x * x1.x - x1.y * x1.y;
        double f = x3.x * x3.x + x3.y * x3.y - x2.x * x2.x - x2.y * x2.y;

        p = point((f * b - e * d) / (c * b - a * d) / 2, (a * f - e * c) / (a * d - b * c) / 2);
        r = distance(p, x1);
    }

    double area() {
        return pi * r * r;
    }

    double perimeter() {
        return 2 * pi * r;
    }

    // 点和圆的关系
    // 0 圆外
    // 1 圆上
    // 2 圆内
    int relationToPoint(point a) {
        double d2 = distance2(p, a);
        if(sgn(d2 - r * r) < 0) return 2;
        else if(sgn(d2 - r * r) == 0) return 1;
        return 0;
    }

    // 圆和直线的关系
    // 0 圆外
    // 1 圆上
    // 2 圆内
    int relationToLine(line l) {
        double d = l.disPointToLine(p);
        if (sgn(d - r) < 0) return 2;
        else if(sgn(d - r) == 0) return 1;
        return 0;
    }

    // 圆和线段的关系
    // 0 圆外
    // 1 圆上
    // 2 圆内
    int relationToSeg(line l) {
        double d = l.disPointToSeg(p);
        if (sgn(d - r) < 0) return 2;
        else if (sgn(d - r) == 0) return 1;
        return 0;
    }

```

```
        // 圆和圆的关系
        // 5 相离
        // 4 外切
        // 3 相交
        // 2 内切
        // 1 内含
        int relationToCircle(circle c) {
            double d = distance(p, c.p);
            if(sgn(d - r - c.r) > 0) return 5;
            if(sgn(d - r - c.r) == 0) return 4;
            double l = fabs(r - c.r);
            if(sgn(d - r - c.r) < 0 && sgn(d - l) > 0) return 3;
            if(sgn(d - l) == 0) return 2;
            if(sgn(d - l) < 0) return 1;
            return -1;
        }
    };

    // 多边形
    struct polygon
    {
        int n;                  // 顶点个数
        vector<point> p;        // 顶点
        vector<line> l;         // 边

        polygon() : n(0) {}
        polygon(int _n) : n(_n), p(n) {}

        point& operator [] (int idx) { return p[idx]; }

        void resize(int _n) {
            n = _n;
            p.resize(n);
        }

        // 多边形周长
        double perimeter() {
            double sum = 0;
            for(int i = 0; i < n; i++) sum += (p[(i + 1) % n] - p[i]).length();
            return sum;
        }

        // 多边形面积
        double area() {
            double sum = 0;
            for(int i = 0; i < n; i++) sum += det(p[i], p[(i + 1) % n]);
            return fabs(sum) / 2;
        }

        void getline() {
            l.resize(n);
            for(int i = 0; i < n; i++) l[i] = line(p[i], p[(i + 1) % n]);
        }

        // 极角排序
        struct cmp {
            point p;
            cmp(const point& _p) : p(_p) {}
            bool operator () (const point& a,const point& b) const {
                int d = sgn(det(p, a, b));
                if(d == 0) return sgn(distance(a, p) - distance(b, p)) < 0;
                return d > 0;
            }
        };
```

```
452         // 标准化，即极角排序（逆时针）
453         void norm() {
454             point mi = p[0];
455             for(int i = 1; i < n; i++) mi = min(mi, p[i]);
456             sort(p.begin(), p.end(), cmp(mi));
457         }
458 
459         // 凸包（非严格）
460         // 若要求严格，则需要再将共线的点除了端点全删去
461         polygon getComvex() {
462             norm();
463             if (n == 0) return polygon(0);
464             else if(n == 1) {
465                 polygon convex(1);
466                 convex[0] = p[0];
467                 return convex;
468             } else if (n == 2) {
469                 if (p[0] == p[1]) {
470                     polygon convex(1);
471                     convex[0] = p[0];
472                     return convex;
473                 }
474                 polygon convex(2);
475                 convex[0] = p[0];
476                 convex[1] = p[1];
477                 return convex;
478             }
479 
480             polygon convex(n);
481             convex.p[0] = p[0];
482             convex.p[1] = p[1];
483             int top = 2;
484             for(int i = 2; i < n; i++) {
485                 while(top > 1 && sgn(det(convex.p[top - 2], convex.p[top - 1], p[i])) <= 0) --top;
486                 convex.p[top++] = p[i];
487             }
488             convex.resize(top);
489             if(convex.n == 2 && convex.p[0] == convex.p[1]) convex.resize(1);
490 
491             return convex;
492         }
493 
494         bool isConvex() {
495             bool s[3] = {0, 0, 0};
496             for(int i = 0, j, k; i < n; i++) {
497                 j = (i + 1) % n;
498                 k = (j + 1) % n;
499                 s[sgn(det(p[i], p[j], p[k])) + 1] = true;
500                 if(s[0] && s[2]) return false;
501             }
502             return true;
503         }
504 
505         // 多边形方向
506         // 1 逆时针
507         // 2 顺时针
508         int direction() {
509             double sum = 0;
510             for(int i = 0; i < n; i++) sum += det(p[i], p[(i + 1) % n]);
511             if(sgn(sum) > 0) return 1;
512             return 0;
513         }
514 
515         // 凸包上最远点对
516         // 平面最远点对就是点集的凸包上的最远点对
```

```
517         pair<point, point> getMaxPair() {
518             assert(n >= 2);
519             if (n == 2) return make_pair(p[0], p[1]);
520             point p1 = p[0], p2 = p[1];
521             double dis = distance(p1, p2);
522 
523             // 旋转卡 (qia) 壳 (qiao)
524             int k = 1;
525             for (int i = 0; i < n; ++i) {
526                 int j = (i + 1) % n;
527                 while(sgn(det(p[i], p[j], p[k]) - det(p[i], p[j], p[(k + 1) % n])) <= 0) k = (k + 1) % n;
528 
529                 if (sgn(distance(p[i], p[k]) - dis) > 0) p1 = p[i], p2 = p[k], dis = distance(p1, p2);
530                 if (sgn(distance(p[j], p[k]) - dis) > 0) p1 = p[j], p2 = p[k], dis = distance(p1, p2);
531             }
532             return make_pair(p1, p2);
533         }
534 
535         double getMaxDis() {
536             pair<point, point> pr = getMaxPair();
537             return distance(pr.first, pr.second);
538         }
539 
540         // 平面最近点对 (P1257, P1429)
541         // 分治法求解平面最近点对，复杂度 O(n log n)
542         void __getMinPair(int l, int r, point& p1, point& p2, double& dis) {
543             if (r - l <= 9) {
544                 for (int i = l; i <= r; ++i) {
545                     for (int j = i + 1; j <= r; ++j) {
546                         double d = distance(p[i], p[j]);
547                         if (d < dis) {
548                             dis = d;
549                             p1 = p[i];
550                             p2 = p[j];
551                         }
552                     }
553                 }
554                 return;
555             }
556 
557             int m = (l + r) >> 1;
558             __getMinPair(l, m, p1, p2, dis); __getMinPair(m, r, p1, p2, dis);
559             vector<point> tmp;
560             for (int i = l; i <= r; ++i) if (abs(p[i].x - p[m].x) <= dis) tmp.push_back(p[i]);
561             sort(tmp.begin(), tmp.end(), [] (const point& a, const point& b) {
562                 return a.y < b.y;
563             });
564             for (int i = 1; i < (int)tmp.size(); ++i) {
565                 for (int j = i - 1; j >= 0; --j) {
566                     if (tmp[j].y < tmp[i].y - dis) break;
567                     double d = distance(tmp[i], tmp[j]);
568                     if (d < dis) {
569                         dis = d;
570                         p1 = tmp[i];
571                         p2 = tmp[j];
572                     }
573                 }
574             }
575         }
576 
577         pair<point, point> getMinPair() {
578             assert(n >= 1);
579             if (n == 2) return make_pair(p[0], p[1]);
580 
581             sort(p.begin(), p.end(), [] (const point& a, const point& b) {
```

```
            return a.x < b.x;
        });
        point p1 = p[0], p2 = p[1];
        double dis = distance(p1, p2);
        __getMinPair(0, n - 1, p1, p2, dis);
        return make_pair(p1, p2);
    }

    double getMinDis() {
        assert(n >= 1);
        if (n == 2) return distance(p[0], p[1]);

        sort(p.begin(), p.end(), [] (const point& a, const point& b) {
            return a.x < b.x;
        });
        point p1 = p[0], p2 = p[1];
        double dis = distance(p1, p2);
        __getMinPair(0, n - 1, p1, p2, dis);
        return dis;
    }

    // 最小圆覆盖 (P2253, P1472)
    // 随机增量法求解最小圆覆盖问题，在随机顺序的点集上，期望复杂度为 O(n)
    circle getMinCircle() {
        // 随机打乱顺序
        srand(time(0));
        for (int i = n - 1; i >= 1; --i) swap(p[i], p[rand() % i]);

        circle c(p[0], 0);
        for (int i = 0; i < n; ++i) {
            if (c.relationToPoint(p[i]) == 2) continue;
            c.p = (p[0] + p[i]) / 2;
            c.r = distance(p[0], p[i]) / 2;

            for (int j = 1; j < i; ++j) {
                if (c.relationToPoint(p[j]) == 2) continue;
                c.p = (p[i] + p[j]) / 2;
                c.r = distance(p[i], p[j]) / 2;

                for (int k = 1; k < j; ++k) {
                    if (c.relationToPoint(p[k]) == 2) continue;
                    c = circle(p[i], p[j], p[k]);
                }
            }
        }
        return c;
    }


    // 点与多边形的位置关系
    // 0 外部
    // 1 内部
    // 2 边上
    // 3 点上
    int relationToPoint(point a) {
        for (int i = 0; i < n; ++i) if (p[i] == a) return 3;

        getline();
        for (int i = 0; i < n; ++i) if (l[i].relationToPoint(a) == 3) return 2;

        int cnt = 0;
        for (int i = 0, j; i < n; ++i) {
            j = (i + 1) % n;
            int k = sgn(det(p[j], a, p[i]));
            int u = sgn(p[i].y - a.y);
```

```
647                 int v = sgn(p[j].y - a.y);
648                 if (k > 0 && u < 0 && v >= 0) ++cnt;
649                 if (k < 0 && v < 0 && u >= 0) --cnt;
650             }
651             return cnt != 0;
652         }
653 
654         void DEBUG() {
655             cout << n << endl;
656             for (int i = 0; i < n; ++i) {
657                 cout << p[i].x << " " << p[i].y << endl;
658             }
659         }
660     };
661 
662     // 半平面 (ax + by + c >= 0), 其实也就是直线
663     // 对于直线 (s, e), h.s 为起点, h.e 为方向向量 (e - s)
664     struct halfplane {
665         point s, v;
666         double k;
667         halfplane() {}
668         halfplane(point _s, point _v) : s(_s), v(_v) {
669             k = v.alpha();
670         }
671         bool operator < (const halfplane& h) const {
672             return k < h.k;
673         }
674     };
675 
676     // 点和半平面的位置关系
677     // 0 不在右侧
678     // 1 在右侧
679     int relationPointToHalfplane(point p, halfplane h) {
680         return sgn(det(h.v, p - h.s)) < 0;
681     }
682 
683     // 半平面交点
684     point HalfplaneCrossHalfplane(halfplane h1, halfplane h2) {
685         double a = det(h2.v, h1.s - h2.s) / det(h1.v, h2.v);
686         return h1.s + h1.v * a;
687     }
688 
689     // 从点集构造出半平面集
690     // 多边形的半平面集即为多边形边集
691     void getHalfPlanes(polygon& p, vector<halfplane>& h) {
692         if (p.direction() != 1) reverse(p.p.begin(), p.p.end());
693         int n = p.n;
694         for (int i = 0, j; i < n; ++i) {
695             j = (i + 1) % n;
696             h.push_back(halfplane(p[i], p[j] - p[i]));
697         }
698     }
699 
700     // 有时候题目给的不一定是闭合图形, 需要自行添加边界
701     // (x1, y1) 为矩形边界左下角, (x2, y2) 为矩形边界右上角
702     // Usage: addBorderHalfPlanes(0, 0, 1e4, 1e4, h);
703     // POJ2451
704     void addBorderHalfPlanes(double x1, double y1, double x2, double y2, vector<halfplane>& h) {
705         polygon p(4);
706         p[0] = point(x1, y1);
707         p[1] = point(x2, y1);
708         p[2] = point(x2, y2);
709         p[3] = point(x1, y2);
710         getHalfPlanes(p, h);
711     }
```

```cpp

    // 半平面交
    // 排序随机增量法 (SI) 求解半平面交，复杂度为 O(n log n)
    // 瓶颈为排序算法，用基数排序则为 O(n)
    // 最终的结果为一个凸包，若少于 3 个点则说明无解

    // 多边形的核：位于多边形内且可以看到多边形内所有点的点集 (P5969, POJ1279)
    // 多边形的半平面交即为多边形的核 (P4196)

    bool getHalfPlaneIntersection(vector<halfplane>& h, polygon& hpi) {
        int n = int(h.size()), l, r;
        sort(h.begin(), h.end());

        vector<point> p(n);
        vector<halfplane> q(n);

        l = r = 0;
        q[l] = h[0];
        for (int i = 1; i < n; ++i) {
            while(l < r && relationPointToHalfplane(p[r - 1], h[i])) --r;
            while(l < r && relationPointToHalfplane(p[l], h[i])) ++l;
            q[++r] = h[i];
            if (l < r && sgn(det(q[r].v, q[r - 1].v)) == 0) {
                --r;
                if (!relationPointToHalfplane(h[i].s, q[r])) q[r] = h[i];
            }
            if (l < r) p[r - 1] = HalfplaneCrossHalfplane(q[r - 1], q[r]);
        }
        while(l < r && relationPointToHalfplane(p[r - 1], q[l])) --r;
        if (r - l + 1 <= 2) return false; // 交不存在
        p[r] = HalfplaneCrossHalfplane(q[l], q[r]);

        hpi.resize(r - l + 1);
        for (int i = l, j = 0; i <= r; ++i) hpi[j++] = p[i];

        return true;
    }

    // 多边形内部半径最大的圆半径 (POJ3525)
    // 二分半径，对多边形边集向内部进行平移，若平移后的多边形存在核，则可行
    double getMaxInsideCircleRadius(polygon& p) {
        if (p.direction() != 1) reverse(p.p.begin(), p.p.end());
        int n = p.n;

        // 方向向量，垂直单位向量
        vector<point> d(n), v(n);
        for (int i = 0; i < n; ++i) {
            d[i] = p[(i + 1) % n] - p[i];
            v[i] = d[i].trans90().unit();
        }

        double l = 0, r = 1e4, m;
        while(r - l >= eps) {
            m = (l + r) / 2;

            vector<halfplane> h(n);
            polygon hpi;
            for (int i = 0; i < n; ++i) h[i] = halfplane(p[i] + v[i] * m, d[i]);
            bool can = getHalfPlaneIntersection(h, hpi);

            if (can) l = m;
            else r = m;
        }
        return l;
    }
```

```
}
using namespace Geometry;
```

3.2 3DGeometry

```
namespace Geometry3 {
    const double eps = 1e-8;

    int sgn(double x) {
        if (fabs(x) < eps) return 0;
        if (x < 0) return -1;
        return 1;
    }

    struct point3 {
        double x, y, z;
        point3(double _x = 0, double _y = 0, double _z = 0) : x(_x), y(_y), z(_z) {}

        bool operator == (const point3& p) const {
            return sgn(x - p.x) == 0 && sgn(y - p.y) == 0 && sgn(z - p.z) == 0;
        }

        bool operator < (const point3& p) const {
            if (sgn(x - p.x) != 0) return sgn(x - p.x) < 0;
            if (sgn(y - p.y) != 0) return sgn(y - p.y) < 0;
            return sgn(z - p.z) < 0;
        }

        point3 operator - (const point3& p) const {
            return point3(x - p.x, y - p.y, z - p.z);
        }

        point3 operator + (const point3& p) const {
            return point3(x + p.x, y + p.y, z + p.z);
        }

        point3 operator * (const double& a) const {
            return point3(x * a, y * a, z * a);
        }

        point3 operator / (const double& a) const {
            return point3(x / a, y / a, z / a);
        }

        double operator * (const point3& p) const {
            return x * p.x + y * p.y + z * p.z;
        }

        point3 operator ^ (const point3& p) const {
            return point3(y * p.z - z * p.y, z * p.x - x * p.z, x * p.y - y * p.x);
        }

        double length() {
            return sqrt(x * x + y * y + z * z);
        }

        double length2() {
            return x * x + y * y + z * z;
        }

        double disTo(const point3& p) {
            return (p - *this).length();
        }

```

```
60          point3 trunc (double r) {
61              double l = length();
62              if (sgn(l) == 0) return *this;
63              r /= l;
64              return *this * r;
65          }
66      };
67
68      double distance(point3 a, point3 b) {
69          return (b - a).length();
70      }
71
72      double distance2(point3 a, point3 b) {
73          return (b - a).length2();
74      }
75
76      point3 det(point3 a, point3 b) {
77          return a ^ b;
78      }
79
80      point3 det(point3 a, point3 b, point3 c) {
81          return (b - a) ^ (c - a);
82      }
83
84      double dot(point3 a, point3 b) {
85          return a * b;
86      }
87
88      double dot(point3 a, point3 b, point3 c) {
89          return (b - a) * (c - a);
90      }
91
92      // ab 与 ac 之间的夹角
93      double radian(point3 a, point3 b, point3 c) {
94          return acos((b - a) * (c - a)) / (distance(a, b), distance(a, c));
95      }
96
97      // 三角形面积
98      double triArea(point3 a, point3 b, point3 c) {
99          return (det(a, b, c)).length() / 2;
100     }
101
102     double triArea2(point3 a, point3 b, point3 c) {
103         return (det(a, b, c)).length();
104     }
105
106     // 四面体有向面积
107     double QuadVolume(point3 a, point3 b, point3 c, point3 d) {
108         return (det(a, b, c) * (d - a)) / 6;
109     };
110
111     double QuadVolume6(point3 a, point3 b, point3 c, point3 d) {
112         return det(a, b, c) * (d - a);
113     };
114
115     struct line3 {
116         point3 s, e;
117
118         line3(point3 _s = point3(), point3 _e = point3()) : s(_s), e(_e) {}
119
120         bool operator == (const line3& l) const {
121             return (s == l.s) && (e == l.e);
122         }
123
124         // 点到直线的距离
```

```
125         double disPointToLine(point3 p) {
126             return det(s, e, p).length() / distance(s, e);
127         }
128 
129         // 点到线段的距离
130         double disPointToSeg(point3 p) {
131             if (sgn(dot(s, p, e)) < 0 || sgn(dot(e, p, s)) < 0)
132                 return min(distance(s, p), distance(e, p));
133             return disPointToLine(p);
134         }
135 
136         // 点在直线上的投影
137         point3 projectionPointOnLine(point3 p) {
138             return s + (((e - s) * dot(s, e, p)) / (e - s).length2());
139         }
140 
141         // 绕 p 旋转 alpha 度
142         point3 rotate(point3 p, double alpha) {
143             if (sgn(det(p, s, e).length()) == 0) return p;
144             point3 p1 = det(s, e, p);
145             point3 p2 = det(e - s, p1);
146             double len = det(p, s, e).length() / distance(s, e);
147             p1 = p1.trunc(len); p2 = p2.trunc(len);
148             point3 p3 = p + p2;
149             point3 p4 = p3 + p1;
150             return p3 + ((p - p3) * cos(alpha) + (p4 - p3) * sin(alpha));
151         }
152 
153         // 点在线段上
154         bool isPointOnSeg(point3 p) {
155             return sgn(det(p, s, e).length()) == 0 && sgn(dot(p, s, e)) == 0;
156         }
157     };
158 
159     struct plane {
160         point3 a, b, c; // 3 点确定平面
161         point3 o; // 平面的法向量
162 
163         point3 pvec() {
164             return det(a, b, c);
165         }
166 
167         plane(point3 _a, point3 _b, point3 _c) : a(_a), b(_b), c(_c) {}
168 
169         plane(point3 _a, point3 _o) : a(_a), o(_o) {}
170 
171         // ax + by + cz + d = 0;
172         plane(double _a, double _b, double _c, double _d) {
173             o = point3(_a, _b, _c);
174             if (sgn(_a) != 0)
175                 a = point3((-_d - _c - _b) / _a, 1, 1);
176             else if (sgn(_b) != 0)
177                 a = point3(1, (-_d - _c - _a) / _b, 1);
178             else if(sgn(_c != 0))
179                 a = point3(1, 1, (-_d - _b - _a) / _c);
180         }
181 
182         // 点在平面上
183         bool isPointOnPlane(point3 p) {
184             return sgn((p - a) * o) == 0;
185         }
186 
187         // 两平面夹角
188         double angle(plane f) {
189             return acos(o * f.o) / (o.length() * f.o.length());
```

```
        }

        // 平面和直线是否相交
        int PlaneCrossLine(line3 l, point3& p) {
            double x = o * (l.e - a);
            double y = o * (l.s - a);
            double d = x - y;
            if (sgn(d) == 0) return 0;
            p = ((l.s * x) - (l.e * y)) / d;
            return 1;
        }

        // 点到平面的最近点
        point3 PointToPlane(point3 p) {
            line3 l = line3(p, p + o);
            PlaneCrossLine(l, p);
            return p;
        }

        // 平面和平面是否相交
        int PlaneCrossPlane(plane f, line3& l) {
            point3 o1 = o ^ f.o;
            point3 o2 = o ^ o1;

            double d = fabs(f.o * o2);
            if (sgn(d) == 0) return 0;
            point3 p = a + (o2 * (f.o * (f.a - a)) / d);
            l = line3(p, p + o1);
            return 1;
        }
    };

    struct polygon3 {
        struct face {
            int a, b, c;
            bool ok;
        };

        int n;
        vector<point3> P;

        int num;
        vector<face> F;
        vector<vector<int> > G;

        polygon3() : n(0) {}
        polygon3(int _n) : n(_n), P(n), F(8 * n), G(n, vector<int>(n)) {}

        double cmp(point3 p, face f) {
            point3 p1 = P[f.b] - P[f.a];
            point3 p2 = P[f.c] - P[f.a];
            point3 p3 = p - P[f.a];
            return (p1 ^ p2) * p3;
        }

        void deal(int p, int a, int b) {
            int f = G[a][b];
            if (F[f].ok) {
                if (cmp(P[p], F[f]) > eps)
                    dfs(p, f);
                else {
                    face add = {b, a, p, true};
                    G[p][b] = G[a][p] = G[b][a] = num;
                    F[num++] = add;
                }
```

```
        }
    }

    void dfs(int p, int now) {
        F[now].ok = false;
        deal(p, F[now].b, F[now].a);
        deal(p, F[now].c, F[now].b);
        deal(p, F[now].a, F[now].c);
    }

    bool same(int s, int t) {
        point3 a = P[F[s].a];
        point3 b = P[F[s].b];
        point3 c = P[F[s].c];

        bool flag = sgn(QuadVolume6(a, b, c, P[F[t].a])) == 0 &&
            sgn(QuadVolume6(a, b, c, P[F[t].b])) == 0 &&
            sgn(QuadVolume6(a, b, c, P[F[t].c])) == 0;

        return flag;
    }

    void buildConvex3() {
        // step 1: 确保前 4 点不共面
        bool flag = true;
        for (int i = 1; i < n; ++i) {
            if (!(P[0] == P[i])) {
                swap(P[1], P[i]);
                flag = false;
                break;
            }
        }
        if (flag) return;

        flag = true;
        for (int i = 2; i < n; ++i) {
            if (det(P[0], P[1], P[i]).length() > eps) {
                swap(P[2], P[i]);
                flag = false;
                break;
            }
        }
        if (flag) return;

        flag = true;
        for (int i = 3; i < n; ++i) {
            if (fabs(det(P[0], P[1], P[2]) * (P[i] - P[0])) > eps) {
                swap(P[3], P[i]);
                flag = false;
                break;
            }
        }
        if (flag) return;

        // step 2
        num = 0;
        for (int i = 0; i < 4; ++i) {
            face add = {(i + 1) % 4, (i + 2) % 4, (i + 3) % 4, true};
            if (cmp(P[i], add) > 0) swap(add.b, add.c);
            G[add.a][add.b] = G[add.b][add.c] = G[add.c][add.a] = num;
            F[num++] = add;
        }

        for (int i = 4; i < n; ++i) {
            for (int j = 0; j < num; ++j) {
```

```
                if (F[j].ok && cmp(P[i], F[j]) > eps) {
                    dfs(i, j);
                    break;
                }
            }
        }

        int tmp = num;
        num = 0;
        for (int i = 0; i < tmp; ++i) if (F[i].ok) {
            F[num++] = F[i];
        }
    }

    // 三维凸包表面积 (POJ3528)
    double area() {
        if (n == 3) return det(P[0], P[1], P[2]).length() / 2;

        double res = 0;
        for (int i = 0; i < num; ++i)
            res += triArea(P[F[i].a], P[F[i].b], P[F[i].c]);
        return res;
    }

    // 三维凸包体积
    double volume() {
        double res = 0;
        point3 tmp(0, 0, 0);
        for (int i = 0; i < num; ++i)
            res += QuadVolume(tmp, P[F[i].a], P[F[i].b], P[F[i].c]);
        return fabs(res);
    }

    // 表面三角形个数
    double getTriangleCount() {
        return num;
    }

    // 表面多边形个数 (HDU3662)
    int getPolygonCount() {
        int res = 0;
        for (int i = 0; i < num; ++i) {
            bool flag = true;
            for (int j = 0; j < i; ++j) {
                if (same(i, j)) {
                    flag = 0;
                    break;
                }
            }
            res += flag;
        }
        return res;
    }

    // 重心 (HDU4273)
    point3 getBaryCenter() {
        point3 ans(0, 0, 0);
        point3 o(0, 0, 0);

        double all = 0;
        for (int i = 0; i < num; ++i) {
            double v = QuadVolume6(o, P[F[i].a], P[F[i].b], P[F[i].c]);
            ans = ans + (((o + P[F[i].a] + P[F[i].b] + P[F[i].c]) / 4) * v);
            all += v;
        }
```

```
            ans = ans / all;
            return ans;
        }

        // 点到凸包第 i 个面上的距离
        double PointToFace(point3 p, int i) {
            double v1 = fabs(QuadVolume6(P[F[i].a], P[F[i].b], P[F[i].c], p));
            double v2 = det(P[F[i].a], P[F[i].b], P[F[i].c]).length();
            return v1 / v2;
        }
    };
}
using namespace Geometry3;
```

3.3 BSGS

```
namespace Backlight {

namespace BSGS {
    typedef long long ll;

    ll exgcd(ll a, ll b, ll& x, ll& y) {
        if (b == 0) {
            x = 1; y = 0;
            return a;
        }
        ll d = exgcd(b, a % b, x, y);
        ll z = x; x = y; y = z - y * (a / b);
        return d;
    }

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

    // solve a^x = b (mod p), p is a prime must hold
    ll BSGS(ll a, ll b, ll p) {
        unordered_map<ll, int> mp;
        if(__gcd(a, p) != 1) return -1;
        if(b % p == 1) return 0;
        a %= p; b %= p;
        ll k = sqrt(p), t = qpow(a, k, p), s = b;
        for(int i = 0; i <= k; i++, s = s * a % p) mp[s] = i;
        s=1;
        for(int i = 0; i <= k; i++, s = s * t % p) {
            int ans = mp.count(s) ? mp[s] : -1;
            if(ans != -1 && i * k - ans >= 0) return i* k - ans;
        }
        return -1;
    }

    // solve a^x = b (mod p), p don't need to be a prime
    ll EXBSGS(ll a, ll b, ll p) {
        ll k = 0, d, c = 1, x, y;
        a %= p; b %= p;
        if(a == b) return 1;
        if(b == 1) return 0;
        while((d=__gcd(a,p)) != 1) {
            if(b % d) return -1;
```

```
            k++; b /= d; p /= d; c = c * (a / d) % p;
            if(c == b) return k;
        }
        if(p == 1) return k;
        exgcd(c, p, x, y); b = (b * x % p + p) % p; a %= p;
        ll ans = BSGS(a, b, p);
        return ans == -1 ? ans : ans + k;
    }
}

}
```

3.4 Cipolla

```
namespace Backlight {

namespace Cipolla {
    mt19937 rnd(chrono::steady_clock::now().time_since_epoch().count());
    ll W, P;
    struct complex {
        ll r, i;
        complex(ll _r, ll _i) : r(_r), i(_i) {}
        inline complex operator * (const complex& c) const { return complex((r * c.r % P + i * c.i % P * W) % P, (r * c
    };

    inline complex pow(complex a, int b) {
        complex res(1, 0);
        while(b) {
            if (b & 1) res = res * a;
            a = a * a;
            b >>= 1;
        }
        return res;
    }

    inline ll pow(ll a, ll b, ll p) {
        ll res = 1;
        while(b) {
            if (b & 1) res = res * a % p;
            a = a * a % p;
            b >>= 1;
        }
        return res;
    }

    // solve x for x^2 = a (mod p)
    ll solve(ll a, ll p) {
        P = p; a %= p;
        if (a == 0) return 0;

        ll t = pow(a, (p - 1) / 2, p);
        if (t != 1) return -1;
        while(true) {
            t = rnd() % p;
            ll c = (t * t % p + p - a) % p;
            if (pow(c, (p - 1) / 2, p) == p - 1) break;
        }

        W = (t * t % p + p - a) % p;
        ll x = pow(complex(t, 1), (p + 1) / 2).r;
        return x;
    }

} // namespace Cipolla
```

```

} // namespace Backlight
```

3.5 Combination

```
struct Combination {
    int N;
    vector<Mint> f, g;

    Combination() : N(0) {}
    Combination(int _n) : N(_n), f(N + 1), g(N + 1) {
        f[0] = 1;
        for (int i = 1; i <= N; ++i) f[i] = f[i - 1] * i;
        g[N] = f[N].inv();
        for (int i = N - 1; i >= 0; --i) g[i] = g[i + 1] * (i + 1);
    }

    Mint get(int n, int m) {
        if (n < 0 || m < 0 || n < m) return 0;
        return f[n] * g[m] * g[n - m];
    }
} C(N);
```

3.6 CRT

```
namespace Backlight {

// get x, y for ax + by = GCD(a, b)
ll exgcd(ll a, ll b, ll& x, ll& y) {
    if (b == 0) {
        x = 1; y = 0;
        return a;
    }
    ll d = exgcd(b, a % b, x, y);
    ll z = x; x = y; y = z - y * (a / b);
    return d;
}



// CRT: solve x = a_i (mod m_i) for i in [0, n)

// GCD(m_i, m_j) = 1 hold
ll CRT(vector<ll>& a, vector<ll>& m) {
    assert(a.size() == m.size());
    assert(a.size() > 0);
    int n = a.size();
    ll M = 1, res = 0;
    for (int i = 0; i < n; ++i) M *= m[i];
    ll _M, x, y;
    for (int i = 0; i < n; ++i) {
        _M = M / m[i];
        exgcd(_M, m[i], x, y);
        res = (res + a[i] * _M % M * x % M) % M;
    }
    if (res < 0) res += M;
    return res;
}

ll mul(ll a, ll b, ll mod) {
    ll res = 0;
    while(b) {
```

```
        if (b & 1) res = (res + a) % mod;
        b >>= 1;
        a = (a + a) % mod;
    }
    return res;
}

// GCD(m_i, m_j) = 1 not hold
ll EXCRT(vector<ll>& a, vector<ll>& m) {
    assert(a.size() == m.size());
    assert(a.size() > 0);
    int n = a.size();
    ll res = a[0], M = m[0], B, g, x, y;
    for (int i = 1; i < n; ++i) {
        B = ((a[i] - res) % m[i] + m[i]) % m[i];
        g = exgcd(M, m[i], x, y);
        x = mul(x, B / g, m[i]);
        res += M * x;
        M *= m[i] / g;
        res = (res + M) % M;
    }
    return res;
}


}
```

3.7 EulerSeive

```
namespace Backlight {

vector<int> euler_seive(int n) {
    vector<int> primes;
    vector<bool> is(n + 1, 1);

    for (int i = 2; i <= n; ++i) {
        if (is[i]) primes.push_back(i);
        for (int j = 0; j < (int)primes.size(); ++j) {
            ll nxt = 1ll * primes[j] * i;
            if (nxt > n) break;
            is[nxt] = false;
            if (i % primes[j] == 0) break;
        }
    }
    return primes;
}

}
```

3.8 eval

```
int pri(char c)
{
    if (c == '^') return 3;
    if (c == '*' || c == '/') return 2;
    if (c == '+' || c == '-') return 1;
    return 0;
}

void in2post(char *s, char *t)
{
    int n = strlen(s), j = 0;
```

```
    stack<char> ops;
    for (int i = 0; i < n; ++i) {
        t[j] = 0;
        if (islower(s[i])) {
            while(i < n && isdigit(s[i])) {
                t[j++] = s[i++];
            }
            t[j++] = ' ';
            --i;
        } else if (s[i] == '(') {
            ops.push('(');
        } else if (s[i] == ')') {
            char op = 0;
            while(!ops.empty()) {
                op = ops.top();
                ops.pop();
                if (op == '(') break;
                t[j++] = op;
                t[j++] = ' ';
            }
            assert(op == '(');
        } else {
            while(!ops.empty() && pri(s[i]) <= pri(ops.top())) {
                t[j++] = ops.top();
                t[j++] = ' ';
                ops.pop();
            }
            ops.push(s[i]);
        }
    }
    while(!ops.empty()) {
        assert(ops.top() != '(');
        t[j++] = ops.top();
        t[j++] = ' ';
        ops.pop();
    }
    t[j] = 0;
}

int eval(char* s)
{
    int n = strlen(s);
    stack<int> nums;
    for (int i = 0; i < n; ++i) {
        if (isdigit(s[i])) {
            int num = 0;
            while(i < n && isdigit(s[i])) {
                num = num * 10 + s[i++] - '0';
            }
            nums.push(num);
            --i;
            continue;
        }

        if (s[i] == ' ') continue;

        assert(nums.size() >= 2);
        int num2 = nums.top();
        nums.pop();
        int num1 = nums.top();
        nums.pop();
        switch(s[i]) {
            case '+':
                nums.push(num1 + num2);
                break;
```

```
            case '-':
                nums.push(num1 - num2);
                break;
            case '*':
                nums.push(num1 * num2);
                break;
            case '/':
                nums.push(num1 / num2);
                break;
            default:
                assert(false);
                break;
        }
    }
    assert(nums.size() == 1);
    return nums.top();
}
```

3.9 EXGCD

```
namespace Backlight {

// get x_0, y_0 for ax + by = GCD(a, b)
// x = x_0 + bt
// y = y_0 - at
//    for all interger t
#define EXGCD
ll exgcd(ll a, ll b, ll& x, ll& y) {
    if (b == 0) {
        x = 1; y = 0;
        return a;
    }
    ll d = exgcd(b, a % b, x, y);
    ll z = x; x = y; y = z - y * (a / b);
    return d;
}

}
```

3.10 FFT

```
namespace FFT {
    const long double PI = acos(-1.0);
    using LL = int64_t;
    struct Complex {
        long double r, i;
        Complex() : r(0), i(0) {}
        Complex(long double _r, long double _i) : r(_r), i(_i) {}
        Complex conj() { return Complex(r, -i); }
        inline Complex operator-(const Complex &c) const { return Complex(r - c.r, i - c.i); }
        inline Complex operator+(const Complex &c) const { return Complex(r + c.r, i + c.i); }
        inline Complex operator*(const Complex &c) const { return Complex(r * c.r - i * c.i, r * c.i + i * c.r); }
    };
    ostream& operator << (ostream& os, Complex& c)  { return os << "(" << c.r << ", " << c.i << ")"; }

    int N;
    vector<int> r;
    void init(int n) {
        N = 1; while(N <= n) N <<= 1;
        r.resize(N);
        for(int i = 1; i < N; ++i) r[i] = (r[i >> 1] >> 1) + ((i & 1) ? (N >> 1) : 0);
    }
```

```
22
23      void FFT(vector<Complex>& a, int op) {
24          for (int i = 1; i < N; ++i) if (i < r[i]) swap(a[i], a[r[i]]);
25          for(int i = 2; i <= N; i <<= 1){
26              int l = i >> 1;
27              Complex w, x, wk(cos(PI / l), op * sin(PI / l));
28              for(int j = 0; j < N; j += i) {
29                  w = Complex(1, 0);
30                  for(int k = j; k < j + l; ++k) {
31                      x = a[k + l] * w;
32                      a[k + l] = a[k] - x;
33                      a[k] = a[k] + x;
34                      w = w * wk;
35                  }
36              }
37          }
38          if(op == -1)
39              for(int i = 0 ; i < N; i++) a[i].r /= N, a[i].i /= N;
40      }
41
42      inline void FFT(vector<Complex>& a) { FFT(a, 1); }
43      inline void IFT(vector<Complex>& a) { FFT(a, -1); }
44
45      vector<int> convolution(const vector<int>& f, const vector<int>& g) {
46          int n = f.size(), m = g.size(), k = n + m - 1;
47          init(k);
48          vector<Complex> a(N), b(N);
49          for (int i = 0; i < n; ++i) a[i] = Complex(f[i], 0);
50          for (int i = 0; i < m; ++i) b[i] = Complex(g[i], 0);
51
52          FFT(a); FFT(b);
53          for (int i = 0; i < N; ++i) a[i] = a[i] * b[i];
54          IFT(a);
55
56          vector<int> h(k);
57          for (int i = 0; i < k; ++i) h[i] = int(a[i].r + 0.5);
58          return h;
59      }
60
61      // 任意模数 FFT
62      vector<int> convolutionM(const vector<int>& f, const vector<int>& g, int p) {
63          int n = f.size(), m = g.size(), k = n + m - 1;
64          init(k);
65          vector<Complex> a(N), b(N), c(N), d(N);
66          for (int i = 0; i < n; ++i) a[i] = Complex(f[i] >> 15, f[i] & 32767);
67          for (int i = 0; i < m; ++i) c[i] = Complex(g[i] >> 15, g[i] & 32767);
68          FFT(a); FFT(c);
69          for (int i = 1; i < N; ++i) b[i] = a[N - i].conj();
70          for (int i = 1; i < N; ++i) d[i] = c[N - i].conj();
71          b[0] = a[0].conj(); d[0] = c[0].conj();
72          for (int i = 0; i < N; ++i) {
73              Complex aa, bb, cc, dd;
74              aa = (a[i] + b[i]) * Complex(0.5, 0);
75              bb = (a[i] - b[i]) * Complex(0, -0.5);
76              cc = (c[i] + d[i]) * Complex(0.5, 0);
77              dd = (c[i] - d[i]) * Complex(0, -0.5);
78              a[i] = aa * cc + Complex(0, 1) * (aa * dd + bb * cc);
79              b[i] = bb * dd;
80          }
81          IFT(a); IFT(b);
82          vector<int> h(k);
83          for (int i = 0; i < k; ++i) {
84              int aa, bb, cc;
85              aa = LL(a[i].r + 0.5) % p;
86              bb = LL(a[i].i + 0.5) % p;
```

```
            cc = LL(b[i].r + 0.5) % p;
            h[i] = ((1ll * aa * (1 << 30) % p + 1ll * bb * (1 << 15) % p + cc) % p + p) % p;
        }
        return h;
    }
} // namespace FFT
```

3.11 LinearBasis

```
struct LinearBasis {
    static const int B = 62;
    ll b[B];
    int tot, n;

    LinearBasis() {
        tot = 0; n = 0;
        memset(b, 0, sizeof(b));
    }

    bool insert(ll x) {
        ++n;
        for (int i = B - 1; i >= 0; --i) {
            if (!(x >> i)) continue;
            if (!b[i]) {
                ++tot;
                b[i] = x;
                break;
            }
            x ^= b[i];
        }
        return x > 0;
    }

    bool query(ll x) {
        for (int i = B - 1; i >= 0; --i) {
            if (!(x >> i)) continue;
            if (!b[i]) return false;
            x ^= b[i];
        }
        return x == 0;
    }

    ll queryMax() {
        ll res = 0;
        for (int i = B - 1; i >= 0; --i) {
            if ((res ^ b[i]) > res) res ^= b[i];
        }
        return res;
    }

    ll queryMin() {
        for (int i = 0; i < B; ++i) if (b[i]) return b[i];
        return -1;
    }

    ll count() {
        return 1LL << tot;
    }

    void rebuild() {
        for (int i = B - 1; i >= 0; --i) {
            for (int j = i - 1; j >= 0; --j) {
                if (b[i] & (1LL << j))
                    b[i] ^= b[j];
```

```
            }
        }
    }

    // need rebuid first
    ll queryKth(int k) {
        if (k == 1 && tot < n) return 0;
        if (tot < n) --k;
        if (k > (1LL << tot) - 1) return -1;
        ll res = 0;
        for (int i = 0; i < B; ++i) {
            if (b[i]) {
                if (k & 1) res ^= b[i];
                k >>= 1;
            }
        }
        return res;
    }
};
```

3.12 Lucas

```
namespace Backlight {

// use this when n, m is really large and p is small
namespace Lucas {
    inline ll pow(ll a, ll b, ll p) {
        ll res = 1;
        a %= p;
        while(b) {
            if (b & 1) res = res * a % p;
            a = a * a % p;
            b >>= 1;
        }
        return res;
    }

    inline ll inv1(ll n, ll p) { return pow(n, p - 2, p); }

    inline ll C1(ll n, ll m, ll p) {
        if (m > n) return 0;
        if (m > n - m) m = n - m;
        ll u = 1, d = 1;
        for (ll i = 1; i <= m; ++i) {
            u = u * (n - i + 1) % p;
            d = d * i % p;
        }
        return u * inv1(d, p) % p;
    }

    // solve n choose m (mod p) while p is a prime
    ll lucas(ll n, ll m, ll p) {
        if (m == 0) return 1;
        return C1(n % p, m % p, p) * lucas(n / p, m / p, p) % p;
    }


    ll exgcd(ll a, ll b, ll& x, ll& y) {
        if (b == 0) {
            x = 1; y = 0;
            return a;
        }
        ll d = exgcd(b, a % b, x, y);
        ll z = x; x = y; y = z - y * (a / b);
```

```
        return d;
    }

    inline ll inv2(ll n, ll p) {
        ll x, y;
        ll d = exgcd(n, p, x, y);
        return d == 1 ? (p + x % p) % p : -1;
    }

    // n! mod pk without pi^x
    ll f(ll n, ll pi, ll pk) {
        if (!n) return 1;
        ll res = 1;
        if (n / pk) {
            for (ll i = 2; i <= pk; ++i)
                if (i % pi) res = res * i % pk;
            res = pow(res, n / pk, pk);
        }
        for (ll i = 2; i <= n % pk; ++i)
            if (i % pi) res = res * i % pk;
        return res * f(n / pi, pi, pk) % pk;
    }

    ll C2(ll n, ll m, ll p, ll pi, ll pk) {
        if (m > n) return 0;
        ll a = f(n, pi, pk), b = f(m, pi, pk), c = f(n - m, pi, pk);
        ll k = 0;
        for (ll i = n; i; i /= pi) k += i / pi;
        for (ll i = m; i; i /= pi) k -= i / pi;
        for (ll i = n - m; i; i /= pi) k -= i / pi;
        ll ans = a * inv2(b, pk) % pk * inv2(c, pk) % pk * pow(pi, k, pk) % pk;
        ans = ans * (p / pk) % p * inv2(p / pk, pk) % p;
        return ans;
    }

    // solve n choose m (mod p) while p might not be a prime
    ll exlucas(ll n, ll m, ll p) {
        ll x = p;
        ll ans = 0;
        for (ll i = 2; i <= p; ++i) {
            if (x % i == 0) {
                ll pk = 1;
                while(x % i == 0) pk *= i, x /= i;
                ans = (ans + C2(n, m, p, i, pk)) % p;
            }
        }
        return ans;
    }

} // namespace Lucas

} // namespace Backlight
```

3.13 Mint

```
// Author: tourist
template <typename T>
T inverse(T a, T m) {
  T u = 0, v = 1;
  while (a != 0) {
    T t = m / a;
    m -= t * a; swap(a, m);
    u -= t * v; swap(u, v);
  }
```

```
  assert(m == 1);
  return u;
}

template <typename T>
class Modular {
 public:
  using Type = typename decay<decltype(T::value)>::type;

  constexpr Modular() : value() {}
  template <typename U>
  Modular(const U& x) {
    value = normalize(x);
  }

  template <typename U>
  static Type normalize(const U& x) {
    Type v;
    if (-mod() <= x && x < mod()) v = static_cast<Type>(x);
    else v = static_cast<Type>(x % mod());
    if (v < 0) v += mod();
    return v;
  }

  const Type& operator()() const { return value; }
  template <typename U>
  explicit operator U() const { return static_cast<U>(value); }
  constexpr static Type mod() { return T::value; }

  Modular& operator+=(const Modular& other) { if ((value += other.value) >= mod()) value -= mod(); return *this; }
  Modular& operator-=(const Modular& other) { if ((value -= other.value) < 0) value += mod(); return *this; }
  template <typename U> Modular& operator+=(const U& other) { return *this += Modular(other); }
  template <typename U> Modular& operator-=(const U& other) { return *this -= Modular(other); }
  Modular& operator++() { return *this += 1; }
  Modular& operator--() { return *this -= 1; }
  Modular operator++(int) { Modular result(*this); *this += 1; return result; }
  Modular operator--(int) { Modular result(*this); *this -= 1; return result; }
  Modular operator-() const { return Modular(-value); }

  template <typename U = T>
  typename enable_if<is_same<typename Modular<U>::Type, int>::value, Modular>::type& operator*=(const Modular& rhs) {
#ifdef _WIN32
    uint64_t x = static_cast<int64_t>(value) * static_cast<int64_t>(rhs.value);
    uint32_t xh = static_cast<uint32_t>(x >> 32), xl = static_cast<uint32_t>(x), d, m;
    asm(
      "divl %4; \n\t"
      : "=a" (d), "=d" (m)
      : "d" (xh), "a" (xl), "r" (mod())
    );
    value = m;
#else
    value = normalize(static_cast<int64_t>(value) * static_cast<int64_t>(rhs.value));
#endif
    return *this;
  }
  template <typename U = T>
  typename enable_if<is_same<typename Modular<U>::Type, long long>::value, Modular>::type& operator*=(const Modular& rh
    long long q = static_cast<long long>(static_cast<long double>(value) * rhs.value / mod());
    value = normalize(value * rhs.value - q * mod());
    return *this;
  }
  template <typename U = T>
  typename enable_if<!is_integral<typename Modular<U>::Type>::value, Modular>::type& operator*=(const Modular& rhs) {
    value = normalize(value * rhs.value);
    return *this;
```

```
  }

  Modular& operator/=(const Modular& other) { return *this *= Modular(inverse(other.value, mod())); }

  friend const Type& abs(const Modular& x) { return x.value; }

  template <typename U>
  friend bool operator==(const Modular<U>& lhs, const Modular<U>& rhs);

  template <typename U>
  friend bool operator<(const Modular<U>& lhs, const Modular<U>& rhs);

  template <typename V, typename U>
  friend V& operator>>(V& stream, Modular<U>& number);

 private:
  Type value;
};

template <typename T> bool operator==(const Modular<T>& lhs, const Modular<T>& rhs) { return lhs.value == rhs.value; }
template <typename T, typename U> bool operator==(const Modular<T>& lhs, U rhs) { return lhs == Modular<T>(rhs); }
template <typename T, typename U> bool operator==(U lhs, const Modular<T>& rhs) { return Modular<T>(lhs) == rhs; }

template <typename T> bool operator!=(const Modular<T>& lhs, const Modular<T>& rhs) { return !(lhs == rhs); }
template <typename T, typename U> bool operator!=(const Modular<T>& lhs, U rhs) { return !(lhs == rhs); }
template <typename T, typename U> bool operator!=(U lhs, const Modular<T>& rhs) { return !(lhs == rhs); }

template <typename T> bool operator<(const Modular<T>& lhs, const Modular<T>& rhs) { return lhs.value < rhs.value; }

template <typename T> Modular<T> operator+(const Modular<T>& lhs, const Modular<T>& rhs) { return Modular<T>(lhs) += rh
template <typename T, typename U> Modular<T> operator+(const Modular<T>& lhs, U rhs) { return Modular<T>(lhs) += rhs; }
template <typename T, typename U> Modular<T> operator+(U lhs, const Modular<T>& rhs) { return Modular<T>(lhs) += rhs; }

template <typename T> Modular<T> operator-(const Modular<T>& lhs, const Modular<T>& rhs) { return Modular<T>(lhs) -= rh
template <typename T, typename U> Modular<T> operator-(const Modular<T>& lhs, U rhs) { return Modular<T>(lhs) -= rhs; }
template <typename T, typename U> Modular<T> operator-(U lhs, const Modular<T>& rhs) { return Modular<T>(lhs) -= rhs; }

template <typename T> Modular<T> operator*(const Modular<T>& lhs, const Modular<T>& rhs) { return Modular<T>(lhs) *= rh
template <typename T, typename U> Modular<T> operator*(const Modular<T>& lhs, U rhs) { return Modular<T>(lhs) *= rhs; }
template <typename T, typename U> Modular<T> operator*(U lhs, const Modular<T>& rhs) { return Modular<T>(lhs) *= rhs; }

template <typename T> Modular<T> operator/(const Modular<T>& lhs, const Modular<T>& rhs) { return Modular<T>(lhs) /= rh
template <typename T, typename U> Modular<T> operator/(const Modular<T>& lhs, U rhs) { return Modular<T>(lhs) /= rhs; }
template <typename T, typename U> Modular<T> operator/(U lhs, const Modular<T>& rhs) { return Modular<T>(lhs) /= rhs; }

template<typename T, typename U>
Modular<T> power(const Modular<T>& a, const U& b) {
  assert(b >= 0);
  Modular<T> x = a, res = 1;
  U p = b;
  while (p > 0) {
    if (p & 1) res *= x;
    x *= x;
    p >>= 1;
  }
  return res;
}

template <typename T>
bool IsZero(const Modular<T>& number) {
  return number() == 0;
}

template <typename T>
string to_string(const Modular<T>& number) {
```

```
  return to_string(number());
}

// U == std::ostream? but done this way because of fastoutput
template <typename U, typename T>
U& operator<<(U& stream, const Modular<T>& number) {
  return stream << number();
}

// U == std::istream? but done this way because of fastinput
template <typename U, typename T>
U& operator>>(U& stream, Modular<T>& number) {
  typename common_type<typename Modular<T>::Type, long long>::type x;
  stream >> x;
  number.value = Modular<T>::normalize(x);
  return stream;
}

/*
using ModType = int;

struct VarMod { static ModType value; };
ModType VarMod::value;
ModType& md = VarMod::value;
using Mint = Modular<VarMod>;
*/

const int md = 998244353;
using Mint = Modular<std::integral_constant<decay<decltype(MOD)>::type, MOD>>;

/*
vector<Mint> fact(1, 1);
vector<Mint> inv_fact(1, 1);

Mint C(int n, int k) {
  if (k < 0 || k > n) {
    return 0;
  }
  while ((int) fact.size() < n + 1) {
    fact.push_back(fact.back() * (int) fact.size());
    inv_fact.push_back(1 / fact.back());
  }
  return fact[n] * inv_fact[k] * inv_fact[n - k];
}
*/
```

3.14 Mobius

```
int primes[N], pcnt;
bool is[N];
int mu[N]; // 莫比乌斯函数，在这里是其前缀和
void seive() {
    pcnt = 0; mu[1] = 1;
    for (int i = 2; i < N; ++i) is[i] = true;
    for (int i = 2; i < N; ++i) {
        if (is[i]) primes[++pcnt] = i, mu[i] = -1;
        for (int j = 1; j <= pcnt; ++j) {
            ll nxt = 1ll * i * primes[j];
            if (nxt >= N) break;
            is[nxt] = false;
            if (i % primes[j] == 0) {
                mu[nxt] = 0;
                break;
            }
```

```
            mu[nxt] = -mu[i];
        }
    }
    for (int i = 1; i < N; ++i) mu[i] += mu[i - 1];
}
```

3.15 Modular

```
const int MOD = 1e9 + 7;
int add(int x, int y) {
    return x + y >= MOD ? x + y - MOD : x + y;
}
int mul(int x, int y) {
    return 1ll * x * y % MOD;
}
int sub(int x, int y) {
    return x - y < 0 ? x - y + MOD : x - y;
}
int dvd(int x, int y) {
    return 1ll * x * qp(y, MOD - 2) % MOD;
}
```

3.16 NTT

```
namespace Backlight {

namespace NTT {
    // 998244353, 1004535809
    const int P = 998244353, G = 3, Gi = 332748118;

    inline ll pow(ll a, ll b) {
        ll res = 1; a %= P;
        while(b) {
            if (b & 1) res = res * a % P;
            a = a * a % P;
            b >>= 1;
        }
        return res;
    }

    int N, L;
    vector<ll> r;
    void init(vector<ll>& a, vector<ll>& b) {
        int l = a.size() + b.size();
        N = 1; L = 0; while(N < l) N <<= 1, ++L;
        a.resize(N); b.resize(N); r.resize(N);
        for (int i = 0; i < N; ++i)
            r[i] = (r[i >> 1] >> 1) | ((i & 1) << (L - 1));
    }

    void work(vector<ll>& a, int flag) {
        for(int i = 0; i < N; i++)
            if(i < r[i]) swap(a[i], a[r[i]]);
        for(int mid = 1; mid < N; mid <<= 1) {
            ll wn = pow(flag == 1 ? G : Gi, (P - 1) / (mid << 1));
            for(int j = 0; j < N; j += (mid << 1)) {
                ll w = 1;
                for(int k = 0; k < mid; k++, w = (w * wn) % P) {
                    int x = a[j + k], y = w * a[j + k + mid] % P;
                    a[j + k] = (x + y) % P,
                    a[j + k + mid] = (x - y + P) % P;
                }
```

```
            }
        }
    }

    inline void NTT(vector<ll>& a) { work(a, 1); }
    inline void INTT(vector<ll>& a) { work(a, -1); }

    vector<ll> convolution(vector<ll> a, vector<ll> b) {
        init(a, b);
        NTT(a); NTT(b);
        for (int i = 0; i < N; ++i) a[i] = a[i] * b[i] % P;
        INTT(a);
        ll inv = pow(N, P - 2);
        for (int i = 0; i < N; ++i) a[i] = a[i] * inv % P;
        return a;
    }
} // namespace NTT

} // namespace Backlight
```

3.17 PollardRho

```
namespace Backlight {

namespace Pollard_Rho {
    typedef long long ll;
    typedef pair<ll, ll> PLL;
    mt19937 rnd(chrono::steady_clock::now().time_since_epoch().count());

    const int N = 1010000;
    ll C, fac[10010], n, mut, a[1001000];
    int T, cnt, i, l, prime[N], p[N], psize, _cnt;
    ll _e[100], _pr[100];
    vector<ll> d;

    inline ll mul(ll a, ll b, ll p) {
        if (p <= 1000000000) return a * b % p;
        else if (p <= 100000000000001l) return (((a*(b>>20)%p)<<20)+(a*(b&((1<<20)-1))))%p;
        else {
            ll d = (ll)floor(a*(long double)b / p + 0.5);
            ll ret = (a * b - d * p) % p;
            if (ret < 0) ret += p;
            return ret;
        }
    }

    void prime_table(){
        int i, j, tot, t1;
        for (i = 1; i <= psize; i++) p[i] = i;
        for (i = 2, tot = 0; i <= psize; i++) {
            if (p[i] == i) prime[++tot] = i;
            for (j = 1; j <= tot && (t1 = prime[j] * i) <= psize; j++){
                p[t1] = prime[j];
                if (i % prime[j] == 0) break;
            }
        }
    }

    void init(int ps) {
        psize = ps;
        prime_table();
    }

    ll powl(ll a, ll n, ll p) {
```

```
        ll ans = 1;
        for (; n; n >>= 1) {
            if (n & 1) ans = mul(ans, a, p);
            a = mul(a, a, p);
        }
        return ans;
    }

    bool witness(ll a, ll n) {
        int t = 0;
        ll u = n - 1;
        for (; ~u&1; u >>= 1) t++;
        ll x = powl(a, u, n), _x = 0;
        for (; t; t--) {
            _x = mul(x, x, n);
            if (_x == 1 && x != 1 && x != n - 1) return 1;
            x = _x;
        }
        return _x != 1;
    }

    bool miller(ll n) {
        if (n < 2) return 0;
        if (n <= psize) return p[n] == n;
        if (~n & 1) return 0;
        for (int j = 0; j <= 7; j++) if (witness(rnd() % (n - 1) + 1, n)) return 0;
        return 1;
    }

    ll gcd(ll a,ll b) {
        ll ret = 1;
        while (a != 0) {
            if ((~a&1) && (~b&1)) ret <<= 1, a >>= 1,b >>= 1;
            else if (~a&1) a >>= 1;
            else if (~b&1) b >>= 1;
            else {
                if (a < b) swap(a, b);
                a -= b;
            }
        }
        return ret * b;
    }

    ll rho(ll n) {
        for (;;) {
            ll X = rnd() % n, Y, Z, T = 1, *lY = a, *lX = lY;
            int tmp = 20;
            C = rnd() % 10 + 3;
            X = mul(X, X, n) + C; *(lY++) = X; lX++;
            Y = mul(X, X, n) + C; *(lY++) = Y;
            for(; X != Y;) {
                ll t = X - Y + n;
                Z = mul(T, t, n);
                if(Z == 0) return gcd(T, n);
                tmp--;
                if (tmp == 0) {
                    tmp = 20;
                    Z = gcd(Z, n);
                    if (Z != 1 && Z != n) return Z;
                }
                T = Z;
                Y = *(lY++) = mul(Y, Y, n) + C;
                Y = *(lY++) = mul(Y, Y, n) + C;
                X = *(lX++);
            }
```

```
        }
    }

    void _factor(ll n) {
        for (int i = 0; i < cnt; i++) {
            if (n % fac[i] == 0) n /= fac[i], fac[cnt++] = fac[i];
        }
        if (n <= psize) {
            for (; n != 1; n /= p[n]) fac[cnt++] = p[n];
            return;
        }
        if (miller(n)) fac[cnt++] = n;
        else {
            ll x = rho(n);
            _factor(x); _factor(n / x);
        }
    }

    void dfs(ll x,int dep) {
        if (dep == _cnt) d.push_back(x);
        else {
            dfs(x, dep+1);
            for (int i = 1; i <= _e[dep]; i++) dfs(x *=_pr[dep], dep + 1);
        }
    }

    void norm() {
        sort(fac, fac + cnt);
        _cnt = 0;
        for(int i = 0; i < cnt; ++i)
            if (i == 0 || fac[i] != fac[i-1]) _pr[_cnt] = fac[i], _e[_cnt++] = 1;
            else _e[_cnt-1]++;
    }

    vector<ll> getd() {
        d.clear();
        dfs(1, 0);
        return d;
    }

    /************************************************/

    // Attention: call init() before use

    // get all factors
    vector<ll> factorA(ll n) {
        cnt = 0;
        _factor(n);
        norm();
        vector<ll> d = getd();
        sort(d.begin(), d.end());
        return d;
    }

    // get prime factors
    vector<ll> factorP(ll n) {
        cnt = 0;
        _factor(n);
        norm();
        vector<ll> d(_cnt);
        for (int i = 0; i < _cnt; ++i) d[i] = _pr[i];
        return d;
    }

    // get prime factors, n = pr_i^e_i
```

```
    vector<PLL> factorG(ll n) {
        cnt = 0;
        _factor(n);
        norm();
        vector<PLL> d(_cnt);
        for (int i = 0; i < _cnt; ++i) d[i] = make_pair(_pr[i], _e[i]);
        return d;
    }

    bool is_primitive(ll a,ll p) {
        assert(miller(p));
        vector<PLL> D = factorG(p - 1);
        for (int i = 0; i < (int)D.size(); ++i) if (powl(a, (p-1) / D[i].first, p) == 1) return 0;
        return 1;
    }
}

}
```

3.18 poly-struct

```
constexpr int P = 998244353;
vector<int> rev, roots{0, 1};
int power(int a, int b) {
    int r = 1;
    while(b) {
        if (b & 1)
            r = 1ll * r * a % P;
        a = 1ll * a * a % P;
        b >>= 1;
    }
    return r;
}
void dft(vector<int> &a) {
    int n = a.size();
    if (int(rev.size()) != n) {
        int k = __builtin_ctz(n) - 1;
        rev.resize(n);
        for (int i = 0; i < n; ++i)
            rev[i] = rev[i >> 1] >> 1 | (i & 1) << k;
    }
    for (int i = 0; i < n; ++i)
        if (rev[i] < i)
            swap(a[i], a[rev[i]]);
    if (int(roots.size()) < n) {
        int k = __builtin_ctz(roots.size());
        roots.resize(n);
        while ((1 << k) < n) {
            int e = power(3, (P - 1) >> (k + 1));
            for (int i = 1 << (k - 1); i < (1 << k); ++i) {
                roots[2 * i] = roots[i];
                roots[2 * i + 1] = 1ll * roots[i] * e % P;
            }
            ++k;
        }
    }
    for (int k = 1; k < n; k *= 2) {
        for (int i = 0; i < n; i += 2 * k) {
            for (int j = 0; j < k; ++j) {
                int u = a[i + j];
                int v = 1ll * a[i + j + k] * roots[k + j] % P;
                int x = u + v;
                if (x >= P)
                    x -= P;
```

```
                a[i + j] = x;
                x = u - v;
                if (x < 0)
                    x += P;
                a[i + j + k] = x;
            }
        }
    }
}
void idft(vector<int> &a) {
    int n = a.size();
    reverse(a.begin() + 1, a.end());
    dft(a);
    int inv = power(n, P - 2);
    for (int i = 0; i < n; ++i)
        a[i] = 1ll * a[i] * inv % P;
}
struct poly {
    vector<int> a;

    poly() {}
    poly(int f0) { a = {f0}; }
    poly(const vector<int> &f) : a(f) {
        while (!a.empty() && !a.back())
            a.pop_back();
    }
    poly(const vector<int> &f, int n) : a(f) {
        a.resize(n);
    }
    int size() const {
        return a.size();
    }
    int deg() const {
        return a.size() - 1;
    }
    int operator[](int idx) const {
        if (idx < 0 || idx >= size())
            return 0;
        return a[idx];
    }
    void input(int n) {
        a.resize(n);
        FE(v, a) rd(v);
    }
    void output(int n) {
        for (int i = 0; i < n - 1; ++i) printf("%d ", (*this)[i]);
        printf("%d\n", (*this)[n - 1]);
    }
    poly mulxk(int k) const {
        auto b = a;
        b.insert(b.begin(), k, 0);
        return poly(b);
    }
    poly modxk(int k) const {
        k = min(k, size());
        return poly(std::vector<int>(a.begin(), a.begin() + k));
    }
    poly alignxk(int k) const {
        return poly(a, k);
    }
    poly divxk(int k) const {
        if (size() <= k)
            return poly();
        return poly(vector<int>(a.begin() + k, a.end()));
    }
```

```
    friend poly operator+(const poly& f, const poly& g) {
        int k = max(f.size(), g.size());
        vector<int> res(k);
        for (int i = 0; i < k; ++i) {
            res[i] = f[i] + g[i];
            if (res[i] >= P)
                res[i] -= P;
        }
        return poly(res);
    }
    friend poly operator - (const poly& f, const poly &g) {
        int k = max(f.size(), g.size());
        vector<int> res(k);
        for (int i = 0; i < k; ++i) {
            res[i] = f[i] - g[i];
            if (res[i] < 0)
                res[i] += P;
        }
        return poly(res);
    }
    friend poly operator * (const poly& f, const poly& g) {
        int sz = 1, k = f.size() + g.size() - 1;
        while (sz < k) sz *= 2;
        vector<int> p = f.a, q = g.a;
        p.resize(sz); q.resize(sz);
        dft(p); dft(q);
        for (int i = 0; i < sz; ++i)
            p[i] = 1ll * p[i] * q[i] % P;
        idft(p);
        return poly(p);
    }
    friend poly operator / (const poly& f, const poly& g) {
        return f.divide(g).first;
    }
    friend poly operator % (const poly& f, const poly& g) {
        return f.divide(g).second;
    }
    poly &operator += (const poly& f) {
        return (*this) = (*this) + f;
    }
    poly &operator -= (const poly& f) {
        return (*this) = (*this) - f;
    }
    poly &operator *= (const poly& f) {
        return (*this) = (*this) * f;
    }
    poly &operator /= (const poly& f) {
        return (*this) = divide(f).first;
    }
    poly &operator %= (const poly& f) {
        return (*this) = divide(f).second;
    }
    poly derivative() const {
        if (a.empty()) return poly();
        int n = a.size();
        vector<int> res(n - 1);
        for (int i = 0; i < n - 1; ++i)
            res[i] = 1ll * (i + 1) * a[i + 1] % P;
        return poly(res);
    }
    poly integral() const {
        if (a.empty()) return poly();
        int n = a.size();
        vector<int> res(n + 1);
        for (int i = 0; i < n; ++i)
```

```
            res[i + 1] = 1ll * a[i] * power(i + 1, P - 2) % P;
        return poly(res);
    }
    poly rev() const {
        return poly(vector<int>(a.rbegin(), a.rend()));
    }
    poly inv(int m) const {
        poly x(power(a[0], P - 2));
        int k = 1;
        while (k < m) {
            k *= 2;
            x = (x * (2 - modxk(k) * x)).modxk(k);
        }
        return x.modxk(m);
    }
    poly log(int m) const {
        return (derivative() * inv(m)).integral().modxk(m);
    }
    poly exp(int m) const {
        poly x(1);
        int k = 1;
        while (k < m) {
            k *= 2;
            x = (x * (1 - x.log(k) + modxk(k))).modxk(k);
        }
        return x.modxk(m);
    }
    poly sqrt(int m) const {
        poly x(1);
        int k = 1;
        while (k < m) {
            k *= 2;
            x = (x + (modxk(k) * x.inv(k)).modxk(k)) * ((P + 1) / 2);
        }
        return x.modxk(m);
    }
    poly sin() const {
        int g = 3; // g: the ord of P
        int i = power(g, (P - 1) / 4);
        poly p = i * (*this);
        p = p.exp(p.size());

        poly q = (P - i) * (*this);
        q = q.exp(q.size());

        poly r = (p - q) * power(2 * i % P, P - 2);
        return r;
    }
    poly cos() const {
        int g = 3; // g: the ord of P
        int i = power(g, (P - 1) / 4);
        poly p = i * (*this);
        p = p.exp(p.size());

        poly q = (P - i) * (*this);
        q = q.exp(q.size());

        poly r = (p + q) * power(2, P - 2);
        return r;
    }
    poly tan() const {
        return sin() / cos();
    }
    poly cot() const {
        return cos() / sin();
```

```
    }
    poly arcsin() {
        poly sq = (*this) * (*this).modxk(size());
        for (int i = 0; i < size(); ++i) sq.a[i] = sq.a[i] ? P - sq.a[i] : 0;
        sq.a[0] = 1 + sq.a[0];
        if (sq.a[0] >= P) sq.a[0] -= P;
        poly r = (derivative() * sq.sqrt(size()).inv(size())).integral();
        return r;
    }
    poly arccos() {
        poly r = arcsin();
        for (int i = 0; i < size(); ++i) r.a[i] = r.a[i] ? P - r.a[i] : 0;
        return r;
    }
    poly arctan() {
        poly sq = (*this) * (*this).modxk(size());
        sq.a[0] = 1 + sq.a[0];
        if (sq.a[0] >= P) sq.a[0] -= P;
        poly r = (derivative() * sq.inv(size())).integral();
        return r;
    }
    poly arccot() {
        poly r = arctan();
        for (int i = 0; i < size(); ++i) r.a[i] = r.a[i] ? P - r.a[i] : 0;
        return r;
    }
    poly mulT(const poly& b) const {
        if (b.size() == 0)
            return poly();
        int n = b.size();
        return ((*this) * b.rev()).divxk(n - 1);
    }
    pair<poly, poly> divide(const poly& g) const {
        int n = a.size(), m = g.size();
        if (n < m) return make_pair(poly(), a);

        poly fR = rev();
        poly gR = g.rev().alignxk(n - m + 1);
        poly gRI = gR.inv(gR.size());

        poly qR = (fR * gRI).modxk(n - m + 1);

        poly q = qR.rev();

        poly r = ((*this) - g * q).modxk(m - 1);

        return make_pair(q, r);
    }
    vector<int> eval(vector<int> x) const {
        if (size() == 0)
            return vector<int>(x.size(), 0);
        const int n = max(int(x.size()), size());
        vector<poly> q(4 * n);
        vector<int> ans(x.size());
        x.resize(n);
        function<void(int, int, int)> build = [&](int p, int l, int r) {
            if (r - l == 1) {
                q[p] = vector<int>{1, (P - x[l]) % P};
            } else {
                int m = (l + r) / 2;
                build(2 * p, l, m);
                build(2 * p + 1, m, r);
                q[p] = q[2 * p] * q[2 * p + 1];
            }
        };
```

```
        build(1, 0, n);
        function<void(int, int, int, const poly &)> work = [&](int p, int l, int r, const poly &num) {
            if (r - l == 1) {
                if (l < int(ans.size()))
                    ans[l] = num[0];
            } else {
                int m = (l + r) / 2;
                work(2 * p, l, m, num.mulT(q[2 * p + 1]).modxk(m - l));
                work(2 * p + 1, m, r, num.mulT(q[2 * p]).modxk(r - m));
            }
        };
        work(1, 0, n, mulT(q[1].inv(n)));
        return ans;
    }
};
```

3.19 Poly

```
namespace Poly {
    const int N = ...;
    const int MAXN = N << 3;
    const int P = 998244353;
    const int G = 3;

    ll qp(ll a, ll b) {
        ll res = 1; a %= P;
        while(b) {
            if (b & 1) res = res * a % P;
            a = a * a % P;
            b >>= 1;
        }
        return res;
    }

    const int Gi = qp(G, P - 2);
    const int I2 = qp(2, P - 2);
    int r[MAXN];
    ll t1[MAXN], t2[MAXN], t3[MAXN], t4[MAXN], t5[MAXN], t6[MAXN], t7[MAXN];

    // int N, l;
    // void init(int n) {
    //     int N = 1, l = -1; while(N <= n << 1) N <<= 1, l++;
    //     for(int i = 1; i < N; ++i) r[i] = (r[i >> 1] >> 1) | ((i & 1) << l);
    // }

    void inplaceNTT(ll *a, int n, int op) {
        for(int i = 0; i < n; ++i) if(i < r[i]) swap(a[i],a[r[i]]);
        for(int m2 = 2, m = 1; m2 <= n; m = m2, m2 <<= 1) {
            ll wn = qp(op == 1 ? G : Gi, (P - 1) / m2), x, y;
            for(int l = 0; l < n; l += m2) {
                ll w = 1;
                for(int i = l; i < l + m; ++i) {
                    x = a[i], y = w * a[i + m] % P;
                    a[i] = (x + y) % P;
                    a[i + m] = (x + P - y) % P;
                    w = w * wn % P;
                }
            }
        }
        if (op == -1) {
            ll inv = qp(n, P - 2);
            for(int i = 0; i < n; ++i) a[i] = a[i] * inv % P;
        }
    }
```

```cpp
inline void NTT(ll *a, int n) { inplaceNTT(a, n, 1); }
inline void INTT(ll *a, int n) { inplaceNTT(a, n, -1); }

// 多项式微分 (求导)
inline void Derivative(ll *a, ll *b, int n) {
    for(int i = 0; i < n; ++i) b[i] = a[i + 1] * (i + 1) % P;
    b[n - 1] = 0;
}

// 多项式积分
inline void Integral(ll *a, ll *b, int n) {
    for(int i = 0; i < n; ++i) b[i + 1] = a[i] * qp(i + 1, P - 2) % P;
    b[0] = 0;
}

// 多项式翻转
// b(x) = x^{n} a(\frac{1}{x})
inline void Reverse(ll *a, ll *b, int n) {
    for (int i = 0; i < n; ++i) b[i] = a[n - i - 1];
}

// 多项式乘法逆
// b(x) = a^{-1}(x) mod x^n
void __Inverse(ll *a, ll *b, int n) {
    if(n == 1) {
        b[0] = qp(a[0], P - 2);
        return;
    }

    __Inverse(a, b, (n + 1) >> 1);

    int N = 1, l = -1; while(N <= n << 1) N <<= 1, l++;
    for(int i = 1; i < N; ++i) r[i] = (r[i >> 1] >> 1) | ((i & 1) << l);

    memcpy(t1, a, sizeof(a[0]) * n); fill(t1 + n, t1 + N, 0);

    NTT(t1, N); NTT(b, N);
    for(int i = 0; i < N; ++i) b[i] = ((b[i] << 1) % P + P - t1[i] * b[i] % P * b[i] % P) % P;
    INTT(b, N);

    fill(b + n, b + N, 0);
}

inline void Inverse(ll *a, ll *b, int n) {
    fill(b, b + (n << 2), 0);
    __Inverse(a, b, n);
}

// 多项式对数函数
// b(x) = Ln a(x) mod x^n
void Ln(ll *a, ll *b, int n) {
    #define aD t3
    #define aI t4

    Derivative(a, aD, n); Inverse(a, aI, n);
    int N = 1, l = -1; while(N <= n << 1) N <<= 1, l++;
    for(int i = 1; i < N; ++i) r[i] = (r[i >> 1] >> 1) | ((i & 1) << l);
    NTT(aD, N); NTT(aI, N);
    for(int i = 0; i < N; ++i) aD[i] = aD[i] * aI[i] % P;
    INTT(aD, N); Integral(aD, b, n);

    #undef aD
    #undef aI
}
```

```
112     // 多项式指数函数
113     // b(x) = exp a(x) mod x^n
114     void Exp(ll *a, ll *b, int n) {
115         #define Lnb t2
116 
117         if(n == 1) {
118             b[0] = 1;
119             return;
120         }
121         Exp(a, b, (n + 1) >> 1);
122         Ln(b, Lnb, n);
123         int N = 1, l = -1; while(N <= n << 1) N <<= 1, l++;
124         for(int i = 1; i < N; ++i) r[i] = (r[i >> 1] >> 1) | ((i & 1) << l);
125 
126         memcpy(t1, a, sizeof(a[0]) * n); fill(t1 + n, t1 + N, 0);
127         fill(Lnb + n, Lnb + N, 0);
128 
129         for(int i = 0; i < N; ++i) t1[i] = ((t1[i] - Lnb[i]) % P + P) % P;
130         ++t1[0];
131         NTT(b, N); NTT(t1, N);
132         for(int i = 0; i < N; ++i) b[i] = b[i] * t1[i] % P;
133         INTT(b, N);
134 
135         fill(b + n, b + N, 0);
136         #undef Lnb
137     }
138 
139     // 多项式乘法 (卷积)
140     // c(x) = a(x) * b(x) mod x^(n + m)
141     // deg c = n + m - 1
142     void Convolution(ll *a, int n, ll *b, int m, ll *c) {
143         int N = 1, l = -1; while(N <= (n + m) << 1) N <<= 1, l++;
144         for(int i = 1; i < N; ++i) r[i] = (r[i >> 1] >> 1) | ((i & 1) << l);
145 
146         memcpy(t1, a, sizeof(a[0]) * n); fill(t1 + n, t1 + N, 0);
147         memcpy(t2, b, sizeof(b[0]) * m); fill(t2 + m, t2 + N, 0);
148 
149         NTT(t1, N); NTT(t2, N);
150         for (int i = 0; i < N; ++i) c[i] = t1[i] * t2[i] % P;
151         INTT(c, N);
152         fill(c + n + m, c + N, 0);
153     }
154     #define Multiply Convolution
155 
156     // 多项式除法
157     // a(x) = b(x)Q(x) + R(x)
158     // deg Q = n - m + 1
159     // deg R = m - 1
160     void Divide(ll *a, int n, ll *b, int m, ll *Q, ll *R) {
161         #define aR t3
162         #define bR t4
163         #define bRi t5
164         #define QR t6
165         #define bQ t7
166 
167         int degQ = n - m + 1;
168         int degR = m - 1;
169 
170         Reverse(a, aR, n); Reverse(b, bR, m);
171         for (int i = degQ; i < m; ++i) bR[i] = 0;
172 
173         // get Q(x)
174         Inverse(bR, bRi, degQ);
175         Multiply(aR, n, bRi, degQ, QR);
176         Reverse(QR, Q, degQ);
```

```
177
178         // get R(x)
179         Multiply(b, m, Q, degQ, bQ);
180         for (int i = 0; i < degR; ++i) R[i] = (a[i] - bQ[i] + P) % P;
181
182         #undef aR
183         #undef bR
184         #undef bRi
185         #undef QR
186     }
187
188     // 多项式求平方根
189     // b^{2}(x) = a(x)
190     #define bI t3
191     void __Sqrt(ll *a, ll* b, int n) {
192         if (n == 1) {
193             b[0] = 1;
194             return;
195         }
196
197         __Sqrt(a, b, (n + 1) >> 1);
198
199         Inverse(b, bI, n);
200         Multiply(a, n, bI, n, bI);
201         for (int i = 0; i < n; ++i) b[i] = (b[i] + bI[i]) * I2 % P;
202     }
203     inline void Sqrt(ll *a, ll *b, int n) {
204         fill(bI, bI + (n << 2), 0);
205         __Sqrt(a, b, n);
206     }
207     #undef bI
208
209     struct poly {
210         vector<ll> a;
211         int size() const { return a.size(); }
212         int deg() const { return size() - 1; }
213         ll& operator [] (int i) { assert(i < size()); return a[i]; }
214         ll operator [] (int i) const { return i < size() ? a[i] : 0LL; }
215         void reverse() { std::reverse(a.begin(), a.end()); }
216         void resize(int n) { a.resize(n); }
217         poly(int n = 0) : a(n, 0) {}
218
219         void DEBUG() {
220             cerr << "Poly DEBUG: " << endl;
221             for (const ll& v: a) cerr << v << " ";
222             cerr << endl;
223         }
224
225         void DEBUG() const {
226             cerr << "Poly DEBUG: " << endl;
227             for (const ll& v: a) cerr << v << " ";
228             cerr << endl;
229         }
230
231
232         void input() {
233             for (ll& x: a) read(x);
234         }
235
236         void output() {
237             if (a.empty()) { puts(""); return; }
238             int n = a.size();
239             for (int i = 0; i < n - 1; ++i) printf("%lld ", a[i]);
240             printf("%lld\n", a[n - 1]);
241         }
```

```
242
243         void output() const  {
244             if (a.empty()) { puts(""); return; }
245             int n = a.size();
246             for (int i = 0; i < n - 1; ++i) printf("%lld ", a[i]);
247             printf("%lld\n", a[n - 1]);
248         }
249
250         poly inv(int n = -1) const {
251             if (n == -1) n = size();
252             static ll f[MAXN], g[MAXN];
253             for (int i = 0; i < n; ++i) f[i] = a[i];
254             Inverse(f, g, n);
255             poly res(n);
256             for (int i = 0; i < n; ++i) res[i] = g[i];
257             return res;
258         }
259
260         poly rev() const {
261             int n = size();
262             poly r(n);
263             for (int i = 0; i < n; ++i) r[i] = a[n - i - 1];
264             return r;
265         }
266
267         poly sqrt() {
268             int n = a.size();
269             static ll f[MAXN], g[MAXN];
270             for (int i = 0; i < n; ++i) f[i] = a[i];
271             Sqrt(f, g, n);
272             poly res(n);
273             for (int i = 0; i < n; ++i) res[i] = g[i];
274             return res;
275         }
276     };
277
278     poly operator + (const poly& a, const poly& b) {
279         int k = max(a.size(), b.size());
280         poly c(k);
281         for (int i = 0; i < k; ++i) c[i] = (a[i] + b[i]) % P;
282         return c;
283     }
284
285     poly operator - (const poly& a, const poly& b) {
286         int k = max(a.size(), b.size());
287         poly c(k);
288         for (int i = 0; i < k; ++i) c[i] = (a[i] - b[i] + P) % P;
289         return c;
290     }
291
292     poly operator * (const poly& a, const poly& b) {
293         static ll ta[MAXN], tb[MAXN];
294         int n = a.size(), m = b.size(), k = n + m - 1;
295         for (int i = 0; i < n; ++i) ta[i] = a[i];
296         for (int i = 0; i < m; ++i) tb[i] = b[i];
297
298         Multiply(ta, n, tb, m, ta);
299
300         poly c(k);
301         for (int i = 0; i < k; ++i) c[i] = ta[i];
302         return c;
303     }
304
305     pair<poly, poly> Divide(const poly& a, const poly& b) {
306         static ll ta[MAXN], tb[MAXN], tq[MAXN], tr[MAXN];
```

```
        int n = a.size(), m = b.size();
        if (n < m) return make_pair(poly(0), a);

        int degQ = n - m + 1, degR = m - 1;
        for (int i = 0; i < n; ++i) ta[i] = a[i];
        for (int i = 0; i < m; ++i) tb[i] = b[i];

        Divide(ta, n, tb, m, tq, tr);

        poly q(degQ); for (int i = 0; i < degQ; ++i) q[i] = tq[i];
        poly r(degR); for (int i = 0; i < degR; ++i) r[i] = tr[i];

        return make_pair(q, r);
    }

    poly operator / (const poly &a, const poly &b) { return Divide(a, b).first; }
    poly operator % (const poly &a, const poly &b) { return Divide(a, b).second; }



    // given a(x), deg a = n
    // calc y_i = a(x_i) for i in [0, m), O(n \log^2 n)
    poly t[N << 2], p[N];
    void build(int o, int l, int r) {
        if (l == r) {
            t[o] = p[l];
            return;
        }
        int mid = (l + r) >> 1;
        build(o << 1, l, mid);
        build(o << 1 | 1, mid + 1, r);
        t[o] = t[o << 1] * t[o << 1 | 1];
    }
    void __calcValue(int o, int l, int r, const poly& f, ll *x, ll *y) {
        // if (l == r) {
        //     y[l] = f[0];
        //     return;
        // }
        if (r - l <= 75) { // 降低常数 (魔法)
            for (int i = l; i <= r; ++i) {
                ll v = 0;
                for (int j = f.size() - 1; j >= 0; --j)
                    v = (v * x[i] % P + f[j]) % P;
                y[i] = v;
            }
            return;
        }

        int mid = (l + r) >> 1, lc = o << 1, rc = o << 1 | 1;
        __calcValue(lc, l, mid, f % t[lc], x, y);
        __calcValue(rc, mid + 1, r, f % t[rc], x, y);
    }
    void calcValue(const poly& f, ll *x, ll*y, int m) {
        for (int i = 1; i <= m; ++i) {
            p[i].resize(2);
            p[i][0] = P - x[i];
            p[i][1] = 1;
        }
        build(1, 1, m);
        __calcValue(1, 1, m, f % t[1], x, y);
    }
}
```

3.20 Simplex

```cpp
/**
 * Simplex Alogorithm:
 * solve $\max z = \sum_{j=1}^{n} c_j x_j$
 * with restrictions like: $\sum_{j=1}^{n} a_{ij} x_j = b_j, i = 1, 2, ..., m$
 *                         $x_j \geq 0$
 * in $O(knm)$, where k is a const number。
 *
 * Tips: 1. min => -min => max
 *       2. $x_1 + 2x_2 \leq 9$ => $x_1 + x_2 + x_3 = 9, x_3 \geq 0$
 *       3. x_k without restrictions => $x_k = x_m - x_m and x_m, x_n \geq 0$
 *
 * Notes: 1. c = A_{0}
 *        2. z = max cx
 *        3. Ax = b
 */
enum {
    OK = 1,
    UNBOUNDED = 2,
    INFEASIBLE = 3
};
struct Simplex {
    constexpr static double eps = 1e-10;

    int n, m;
    int flag;
    double z;
    vector<vector<double>> A;
    vector<double> b, x;
    vector<int> idx, idy;

    Simplex(int _n, int _m) : n(_n), m(_m) {
        A = vector<vector<double>>(m + 1, vector<double>(n + 1));
        b = vector<double>(m + 1);
        x = vector<double>(n + 1);
        idx = vector<int>(m + 1);
        idy = vector<int>(n + 1);
    }

    void input() {
        for (int i = 1; i <= n; ++i) read(A[0][i]); // A_{0,i} = c_i
        for (int i = 1; i <= m; ++i) {
            for (int j = 1; j <= n; ++j) read(A[i][j]);
            read(b[i]);
        }
    }

    void pivot(int x, int y) {
        swap(idx[x], idy[y]);

        double k = A[x][y];
        for (int i = 1; i <= n; ++i) A[x][i] /= k;
        b[x] /= k;
        A[x][y] = 1 / k;

        for (int i = 0; i <= m; ++i) if (i != x) {
            k = A[i][y];
            b[i] -= k * b[x];
            A[i][y] = 0;
            for (int j = 1; j <= n; ++j) A[i][j] -= k * A[x][j];
        }
    }

    void init() {
```

```
        flag = OK;
        idx[0] = INT_MAX; for (int i = 1; i <= m; ++i) idx[i] = n + i;
        idy[0] = INT_MAX; for (int i = 1; i <= n; ++i) idy[i] = i;

        for(;;) {
            int x = 0, y = 0;
            for (int i = 1; i <= m; ++i) if (b[i] < -eps && idx[i] < idx[x]) x = i;
            if (!x) break;

            for (int i = 1; i <= n; ++i) if (A[x][i] < -eps && idy[i] < idy[y]) y = i;
            if (!y) { flag = INFEASIBLE; break; }

            pivot(x, y);
        }
    }

    void simplex() {
        for(;;) {
            int x = 0, y = 0;
            for (int i = 1; i <= n; ++i) if (A[0][i] > eps && idy[i] < idy[y]) y = i;
            if (!y) break;

            for (int i = 1; i <= m; ++i) if (A[i][y] > eps) {
                if (!x) x = i;
                else {
                    double delta = b[i] / A[i][y] - b[x] / A[x][y];
                    if (delta < -eps) x = i;
                    else if (delta < eps && idx[i] < idx[x]) x = i;
                }
            }
            if (!x) { flag = UNBOUNDED; break; }

            pivot(x, y);
        }
        z = -b[0];
    }

    void work() {
        init();
        if (flag == OK) simplex();
        if (flag == OK) {
            for (int i = 1; i <= n; ++i) {
                x[i] = 0;
                for (int j = 1; j <= m; ++j) if (idx[j] == i) { x[i] = b[j]; break; }
            }
        }
    }

    void DEBUG() {
        cerr << fixed << setprecision(3);
        cerr << "Simplex Debug: \n";
        for (int i = 1; i <= m; ++i) {
            for (int j = 1; j <= n; ++j) {
                cerr << A[i][j] << " ";
            }
            cerr << "\n";
        }
        for (int i = 1; i <= n; ++i) cerr << x[i] << " ";
        cerr << endl;
        cerr << "Z = " << z << endl;
    }
};
```

3.21 SimpsonIntegral

```cpp
namespace SimpsonIntegral {
    // calculate \int_l^r f(x) dx

    double f(double x) {
        return (c * x + d) / (a * x + b);
    }

    double simpson(double l, double r) {
        double mid = (l + r) / 2;
        return (r - l) * (f(l) + 4 * f(mid) + f(r)) / 6;
    }

    double integral(double l, double r, double eps, double ans) {
        double mid = (l + r) / 2;
        double fl = simpson(l, mid), fr = simpson(mid, r);
        if (abs(fl + fr - ans) <= 15 * eps)
            return fl + fr + (fl + fr - ans) / 15;
        return integral(l, mid, eps / 2, fl) + integral(mid, r, eps / 2, fr);
    }

    double integral(double l, double r, double eps = 1e-8) {
        return integral(l, r, eps, simpson(l, r));
    }
}
```

4 other

4.1 BFPRT

```cpp
/**
 * BFPRT: find the kth element of an array in O(n) using Divide and Conquer method.
 * you can use std::nth_element(a, a + k, a + n) instead
**/
namespace BFPRT {
    template<typename T, typename Cmp>
    T kth_index(T* a, int l, int r, int k, Cmp cmp);

    template<typename T, typename Cmp>
    int insert_sort(T* a, int l, int r, Cmp cmp) {
        for (int i = l + 1; i <= r; ++i) {
            int tmp = a[i];
            int j = i - 1;
            while(j >= l && a[j] > tmp) {
                a[j + 1] = a[j];
                --j;
            }
            a[j + 1] = tmp;
        }
        return l + (r - l) / 2;
    }

    template<typename T, typename Cmp>
    int pivot(T* a, int l, int r, Cmp cmp) {
        if (r - l < 5) return insert_sort(a, l, r, cmp);
        int lst = l - 1;
        for (int i = l; i + 4 <= r; i += 5) {
            int p = insert_sort(a, i, i + 4, cmp);
            swap(a[++lst], a[p]);
        }
        return kth_index<T>(a, l, lst, (lst - l + 1) / 2 + 1, cmp);
    }
```

```

    template<typename T, typename Cmp>
    int partition(T* a, int l, int r, Cmp cmp) {
        int p = pivot(a, l, r, cmp);
        swap(a[p], a[r]);
        int lst = l - 1;
        for (int i = l; i < r; ++i) {
            if (cmp(a[i], a[r])) swap(a[++lst], a[i]);
        }
        swap(a[++lst], a[r]);
        return lst;
    }

    template<typename T, typename Cmp>
    T kth_index(T* a, int l, int r, int k, Cmp cmp) {
        int p = partition(a, l, r, cmp);
        int d = p - l + 1;
        if (d == k) return p;
        else if (d < k) return kth_index(a, p + 1, r, k - d, cmp);
        else return kth_index(a, l, p - 1, k, cmp);
    }

    template<typename T>
    T kth_index(T* a, int l, int r, int k) {
        return kth_index(a, l, r, k, less<T>());
    }
};
```

4.2 cpp-header

```
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;
typedef unsigned long long ull;
typedef pair<int,int> PII;
typedef vector<int> VI;
typedef vector<ll> VL;
typedef vector<vector<int>> VVI;
typedef vector<vector<ll>> VVL;

#define REP(i, _, __) for (int i = (_); i < (__); ++i)
#define PER(i, _, __) for (int i = (_-1); i >= (__); --i)
#define FOR(i, _, __) for (int i = (_); i <= (__); ++i)
#define ROF(i, _, __) for (int i = (_); i >= (__); --i)
#define FE(v, V) for (auto& v: V)

#define EB emplace_back
#define PB push_back
#define MP make_pair
#define FI first
#define SE second
#define SZ(x) ((int)(x).size())
#define ALL(x) (x).begin(),(x).end()
#define LLA(x) (x).rbegin(),(x).rend()

const double PI = acos(-1.0);

namespace Backlight {
    const int __BUFFER_SIZE__ = 1 << 20;
    bool NEOF = 1; //为 0 表示文件结尾
    int __top;
    char __buf[__BUFFER_SIZE__], *__p1 = __buf, *__p2 = __buf, __stk[996];

```

```
35  inline bool isdowncase(char c) { return (c >= 'a') && (c <= 'z'); }
36  inline bool isupcase(char c) { return (c >= 'A') && (c <= 'Z'); }
37  inline bool isdigit(char c) { return (c >= '0') && (c <= '9'); }
38
39  template<typename T>
40  T MIN(T a, T b) { return min(a, b); }
41
42  template<typename First, typename... Rest>
43  First MIN(First f, Rest... r) { return min(f, MIN(r...)); }
44
45  template<typename T>
46  T MAX(T a, T b) { return max(a, b); }
47
48  template<typename First, typename... Rest>
49  First MAX(First f, Rest... r) { return max(f, MAX(r...)); }
50
51  template<typename T>
52  void updMin(T& a, T b) { if (a > b) a = b; }
53
54  template<typename T>
55  void updMax(T& a, T b) { if (a < b) a = b; }
56
57  inline char nc() {
58      return __p1 == __p2 && NEOF && (__p2 = (__p1 = __buf) + fread(__buf, 1, __BUFFER_SIZE__, stdin), __p1 == __p2)
59  }
60
61  template<typename T>
62  inline bool read(T &x) {
63      char c = nc();
64      bool f = 0; x = 0;
65      while (!isdigit(c)) c == '-' && (f = 1), c = nc();
66      while (isdigit(c)) x = (x << 3) + (x << 1) + (c ^ 48), c = nc();
67      if (f) x = -x;
68      return NEOF;
69  }
70
71  inline bool read_db(double &x){
72      bool f = 0; char c = nc(); x = 0;
73      while (!isdigit(c))  { f |= (c == '-'); c = nc(); }
74      while (isdigit(c)) { x = x * 10.0 + (c ^ 48); c = nc(); }
75      if (c == '.') {
76          double temp = 1; c = nc();
77          while (isdigit(c)) { temp = temp / 10.0; x = x + temp * (c ^ 48); c = nc(); }
78      }
79      if (f) x = -x;
80      return NEOF;
81  }
82
83  template<typename T, typename... T2>
84  inline bool read(T &x, T2 &... rest) {
85      read(x);
86      return read(rest...);
87  }
88
89  // inline bool need(char c) { return (c == '-') || (c == '>') || (c == '<');}
90  // inline bool need(char c) { return isdowncase(c) || isupcase(c) || isdigit(c) || c == '.' || c == '#'; }
91  inline bool need(char c) { return isdowncase(c) || isupcase(c) || isdigit(c); }
92
93  inline bool read_str(char *a) {
94      while ((*a = nc()) && need(*a) && NEOF) ++a; *a = '\0';
95      return NEOF;
96  }
97
98  template<typename T>
99  inline void print(T x) {
```

```
100         if (x < 0) putchar('-'), x = -x;
101         if (x == 0) { putchar('0'); return; }
102         __top = 0;
103         while(x) {
104             __stk[++__top] = x % 10 + '0';
105             x /= 10;
106         }
107         while(__top) {
108             putchar(__stk[__top]);
109             --__top;
110         }
111     }
112
113     template<typename First, typename... Rest>
114     inline void print(First f, Rest... r) {
115         print(f); putchar(' ');
116         print(r...);
117     }
118
119     template<typename T>
120     inline void println(T x) {
121         print(x);
122         putchar('\n');
123     }
124
125     template<typename First, typename... Rest>
126     inline void println(First f, Rest... r) {
127         print(f); putchar(' ');
128         println(r...);
129     }
130
131
132
133     template<typename T>
134     inline void _dbg(const char *format, T value) { cerr << format << '=' << value << endl; }
135
136     template<typename First, typename... Rest>
137     inline void _dbg(const char *format, First f, Rest... r) {
138         while(*format != ',') cerr << *format++;
139         cerr << '=' << f << ", ";
140         _dbg(format + 1, r...);
141     }
142
143     template<typename T>
144     ostream &operator<<(ostream& os, vector<T> V) {
145         os << "[ "; for (auto v : V) os << v << ","; return os << " ]";
146     }
147
148     template<typename T>
149     ostream &operator<<(ostream& os, set<T> V) {
150         os << "[ "; for (auto v : V) os << v << ","; return os << " ]";
151     }
152
153     template<typename T>
154     ostream &operator<<(ostream& os, multiset<T> V) {
155         os << "[ "; for (auto v : V) os << v << ","; return os << " ]";
156     }
157
158     template<typename T1, typename T2>
159     ostream &operator<<(ostream& os, map<T1,T2> V) {
160         os << "[ "; for (auto v : V) os << v << ","; return os << " ]";
161     }
162
163     template<typename L, typename R>
164     ostream &operator<<(ostream &os, pair<L,R> P) {
```

```
        return os << "(" << P.first << "," << P.second << ")";
    }

    #ifdef BACKLIGHT
    #define debug(...) cerr << "\033[31m"; _dbg(#__VA_ARGS__, __VA_ARGS__); cerr << "\033[0m";
    // #define debug(...) _dbg(#__VA_ARGS__, __VA_ARGS__);
    #else
    #define debug(...)
    #endif
}

/************     Backlight     **********
 * 一发入魂
 * 仔细读题，注意边界条件
 * 没有思路就试试逆向思维
 * wdnmd! 我柜子动了不打了
 * 能不能把我掉的分还给我
 ************     Backlight     **********/

// mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
// int rnd(int l, int r) { return l + rng() % (r - l + 1); }

using namespace Backlight;
const int N = 2e7 + 5;
const int M = 5e5 + 5;
const int V = 5e7 + 5;
const ll MOD = 1e9+ 7 ;              // 998244353 1e9+7
const int INF = 0x3f3f3f3f;             // 1e9+7 0x3f3f3f3f
const ll LLINF = 0x3f3f3f3f3f3f3f3f;    // 1e18+9 0x3f3f3f3f3f3f3f3f
const double eps = 1e-8;

void solve(int Case) { // printf("Case #%d: ", Case);

}

int main() {
#ifdef BACKLIGHT
    freopen("in.txt", "r", stdin);
    auto begin = std::chrono::steady_clock::now();
#endif

    // ios::sync_with_stdio(false); cin.tie(0); cout.tie(0);
    int T = 1;
    // read(T);
    for (int _ = 1; _ <= T; _++) solve(_);

#ifdef BACKLIGHT
    freopen("in.txt", "r", stdin);
    auto end = std::chrono::steady_clock::now();
    auto duration = std::chrono::duration_cast<std::chrono::milliseconds>(end - begin);
    cerr << "\033[32mTime Elasped: " << (double) (duration.count()) << " ms\033[0m" << endl;
#endif
    return 0;
}
```

4.3 debug

```
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;
typedef unsigned long long ull;
typedef pair<int,int> PII;
typedef vector<int> VI;
```

```
typedef vector<ll> VL;
typedef vector<vector<int>> VVI;
typedef vector<vector<ll>> VVL;

#define REP(i, _, __) for (int i = (_); i < (__); ++i)
#define PER(i, _, __) for (int i = (_-1); i >= (__); --i)
#define FOR(i, _, __) for (int i = (_); i <= (__); ++i)
#define ROF(i, _, __) for (int i = (_); i >= (__); --i)
#define FE(v, V) for (auto& v: V)

#define EB emplace_back
#define PB push_back
#define MP make_pair
#define FI first
#define SE second
#define SZ(x) ((int)(x).size())
#define ALL(x) (x).begin(),(x).end()
#define LLA(x) (x).rbegin(),(x).rend()

const double PI = acos(-1.0);

namespace Backlight {
    const int __BUFFER_SIZE__ = 1 << 20;
    bool NEOF = 1; //为 0 表示文件结尾
    int __top;
    char __buf[__BUFFER_SIZE__], *__p1 = __buf, *__p2 = __buf, __stk[996];

    inline bool isdowncase(char c) { return (c >= 'a') && (c <= 'z'); }
    inline bool isupcase(char c) { return (c >= 'A') && (c <= 'Z'); }
    inline bool isdigit(char c) { return (c >= '0') && (c <= '9'); }

    template<typename T>
    T MIN(T a, T b) { return min(a, b); }

    template<typename First, typename... Rest>
    First MIN(First f, Rest... r) { return min(f, MIN(r...)); }

    template<typename T>
    T MAX(T a, T b) { return max(a, b); }

    template<typename First, typename... Rest>
    First MAX(First f, Rest... r) { return max(f, MAX(r...)); }

    template<typename T>
    void updMin(T& a, T b) { if (a > b) a = b; }

    template<typename T>
    void updMax(T& a, T b) { if (a < b) a = b; }

    inline char nc() {
        return __p1 == __p2 && NEOF && (__p2 = (__p1 = __buf) + fread(__buf, 1, __BUFFER_SIZE__, stdin), __p1 == __p2)
    }

    template<typename T>
    inline bool read(T &x) {
        char c = nc();
        bool f = 0; x = 0;
        while (!isdigit(c)) c == '-' && (f = 1), c = nc();
        while (isdigit(c)) x = (x << 3) + (x << 1) + (c ^ 48), c = nc();
        if (f) x = -x;
        return NEOF;
    }

    inline bool read_db(double &x){
        bool f = 0; char c = nc(); x = 0;
```

```
73          while (!isdigit(c))  { f |= (c == '-'); c = nc(); }
74          while (isdigit(c)) { x = x * 10.0 + (c ^ 48); c = nc(); }
75          if (c == '.') {
76              double temp = 1; c = nc();
77              while (isdigit(c)) { temp = temp / 10.0; x = x + temp * (c ^ 48); c = nc(); }
78          }
79          if (f) x = -x;
80          return NEOF;
81      }
82
83      template<typename T, typename... T2>
84      inline bool read(T &x, T2 &... rest) {
85          read(x);
86          return read(rest...);
87      }
88
89      // inline bool need(char c) { return (c == '-') || (c == '>') || (c == '<');}
90      // inline bool need(char c) { return isdowncase(c) || isupcase(c) || isdigit(c) || c == '.' || c == '#'; }
91      inline bool need(char c) { return isdowncase(c) || isupcase(c) || isdigit(c); }
92
93      inline bool read_str(char *a) {
94          while ((*a = nc()) && need(*a) && NEOF) ++a; *a = '\0';
95          return NEOF;
96      }
97
98      template<typename T>
99      inline void print(T x) {
100         if (x < 0) putchar('-'), x = -x;
101         if (x == 0) { putchar('0'); return; }
102         __top = 0;
103         while(x) {
104             __stk[++__top] = x % 10 + '0';
105             x /= 10;
106         }
107         while(__top) {
108             putchar(__stk[__top]);
109             --__top;
110         }
111     }
112
113     template<typename First, typename... Rest>
114     inline void print(First f, Rest... r) {
115         print(f); putchar(' ');
116         print(r...);
117     }
118
119     template<typename T>
120     inline void println(T x) {
121         print(x);
122         putchar('\n');
123     }
124
125     template<typename First, typename... Rest>
126     inline void println(First f, Rest... r) {
127         print(f); putchar(' ');
128         println(r...);
129     }
130
131
132
133     template<typename T>
134     inline void _dbg(const char *format, T value) { cerr << format << '=' << value << endl; }
135
136     template<typename First, typename... Rest>
137     inline void _dbg(const char *format, First f, Rest... r) {
```

```
        while(*format != ',') cerr << *format++;
        cerr << '=' << f << ", ";
        _dbg(format + 1, r...);
    }

    template<typename T>
    ostream &operator<<(ostream& os, vector<T> V) {
        os << "[ "; for (auto v : V) os << v << ","; return os << " ]";
    }

    template<typename T>
    ostream &operator<<(ostream& os, set<T> V) {
        os << "[ "; for (auto v : V) os << v << ","; return os << " ]";
    }

    template<typename T>
    ostream &operator<<(ostream& os, multiset<T> V) {
        os << "[ "; for (auto v : V) os << v << ","; return os << " ]";
    }

    template<typename T1, typename T2>
    ostream &operator<<(ostream& os, map<T1,T2> V) {
        os << "[ "; for (auto v : V) os << v << ","; return os << " ]";
    }

    template<typename L, typename R>
    ostream &operator<<(ostream &os, pair<L,R> P) {
        return os << "(" << P.first << "," << P.second << ")";
    }

    #ifdef BACKLIGHT
    #define debug(...) cerr << "\033[31m"; _dbg(#__VA_ARGS__, __VA_ARGS__); cerr << "\033[0m";
    // #define debug(...) _dbg(#__VA_ARGS__, __VA_ARGS__);
    #else
    #define debug(...)
    #endif
}

/************     Backlight     **********
 * 一发入魂
 * 仔细读题，注意边界条件
 * 没有思路就试试逆向思维
 * wdnmd! 我柜子动了不打了
 * 能不能把我掉的分还给我
 ************     Backlight     **********/

// mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
// int rnd(int l, int r) { return l + rng() % (r - l + 1); }

using namespace Backlight;
const int N = 2e7 + 5;
const int M = 5e5 + 5;
const int V = 5e7 + 5;
const ll MOD = 1e9+ 7 ;              // 998244353 1e9+7
const int INF = 0x3f3f3f3f;             // 1e9+7 0x3f3f3f3f
const ll LLINF = 0x3f3f3f3f3f3f3f3f;    // 1e18+9 0x3f3f3f3f3f3f3f3f
const double eps = 1e-8;

void solve(int Case) { // printf("Case #%d: ", Case);

}

int main() {
#ifdef BACKLIGHT
    freopen("in.txt", "r", stdin);
```

```
    auto begin = std::chrono::steady_clock::now();
#endif

    // ios::sync_with_stdio(false); cin.tie(0); cout.tie(0);
    int T = 1;
    // read(T);
    for (int _ = 1; _ <= T; _++) solve(_);

#ifdef BACKLIGHT
    freopen("in.txt", "r", stdin);
    auto end = std::chrono::steady_clock::now();
    auto duration = std::chrono::duration_cast<std::chrono::milliseconds>(end - begin);
    cerr << "\033[32mTime Elasped: " << (double) (duration.count()) << " ms\033[0m" << endl;
#endif
    return 0;
}
```

4.4 java-header

```
import java.io.*;
import java.util.*;
import java.math.*;

public class Main {
    public static void main(String[] args) {
        InputStream inputStream = System.in;
        OutputStream outputStream = System.out;
        InputReader in = new InputReader(inputStream);
        PrintWriter out = new PrintWriter(outputStream);
        Task solver = new Task();

        int T = 1;
        // T = in.nextInt();
        for (int i = 1; i <= T; ++i)
            solver.solve(i, in, out);

        out.close();
    }

    static class Task {
        public void solve(int testNumber, InputReader in, PrintWriter out) {
            // write your solution here
            out.println("Hello World");
        }
    }

    static class InputReader {
        public BufferedReader reader;
        public StringTokenizer tokenizer;

        public InputReader(InputStream stream) {
            reader = new BufferedReader(new InputStreamReader(stream), 32768);
            tokenizer = null;
        }

        public String next() {
            while (tokenizer == null || !tokenizer.hasMoreTokens()) {
                try {
                    tokenizer = new StringTokenizer(reader.readLine());
                } catch (IOException e) {
                    throw new RuntimeException(e);
                }
            }
            return tokenizer.nextToken();
```

```
        }

        public int nextInt() {
            return Integer.parseInt(next());
        }

    }
}
```

4.5 SimulateAnneal

```
struct SimulateAnneal {
    constexpr static double p = 0.996;
    inline double Rand() { return 1.0 * rand() / RAND_MAX; }

    int n;
    vector<int> X, Y, W;
    double ax, ay;

    SimulateAnneal(int _n) : n(_n), X(n), Y(n), W(n) {}

    void input() {
        for (int i = 0; i < n; ++i) {
            read(X[i], Y[i], W[i]);
        }
    }

    double cost(double x, double y) {
        double res = 0;
        for (int i = 0; i < n; ++i) {
            double dx = X[i] - x;
            double dy = Y[i] - y;
            double d = sqrt(dx * dx + dy * dy);
            res += d * W[i];
        }
        return res;
    }

    void init() {
        ax = 0; ay = 0;
        for (int i = 0; i < n; ++i) ax += X[i], ay += Y[i];
        ax /= n; ay /= n;
    }

    void simulate_anneal() {
        srand(time(0));
        double T = 1e6, TE = 1e-8;
        double cx = ax, cy = ay, cc = cost(cx, cy);
        while(T > TE) {
            double nx = ax + (2 * Rand() - 1) * T;
            double ny = ay + (2 * Rand() - 1) * T;

            double nc = cost(nx, ny);
            double d = nc - cc;

            if (d < 0) cc = nc, ax = cx = nx, ay = cy = ny;
            else if (exp(-d / T) > Rand()) {
                cx = nx;
                cy = ny;
            }

            T *= p;
        }
    }
```

```

    void work() {
        init();
        // try a try, AC is ok.
        simulate_anneal();
        simulate_anneal();
        simulate_anneal();
        simulate_anneal();
    }
};
```

5 string

5.1 ACAM

```
namespace ACAM {
    const int __N = 3e5 + 5;
    const int __M = 26;
    int tot, tr[__N][__M], fail[__N], last[__N];
    int f[__N], e[__N];

    int eid[__N];
    multiset<int> st[__N];

    inline int idx(const char& c) { return c - 'a';  }

    inline void init() {
        tot = 0;
        memset(tr[0], 0, sizeof(tr[0]));
        f[0] = e[0] = 0;
    }

    inline int newnode() {
        ++tot;
        memset(tr[tot], 0, sizeof(tr[tot]));
        f[tot] = e[tot] = 0;
        return tot;
    }

    void insert(char* s, int n, int id) {
        int p = 0, c;
        for (int i = 0 ; i < n; ++i) {
            c = idx(s[i]);
            if (!tr[p][c]) tr[p][c] = newnode();
            p = tr[p][c];
            ++f[p];
        }
        ++e[p];


        eid[id] = p;
        st[p].insert(0);
    }

    // 字典图优化
    // void getfail() {
    //     queue<int> q;
    //     for (int i = 0; i < __M; ++i) if (tr[0][i]) fail[tr[0][i]] = 0, q.push(tr[0][i]);
    //     while(!q.empty()) {
    //         int p = q.front(); q.pop();
    //         for (int c = 0; c < __M; ++c) {
    //             int nxt = tr[p][c];
    //             if (nxt) fail[nxt] = tr[fail[p]][c], q.push(nxt);
```

```
    //             else nxt = tr[fail[p]][c];
    //         }
    //     }
    // }

    // int query(char* t) {
    //     int n = strlen(t), p = 0, res = 0;
    //     for (int i = 0; i < n; ++i) {
    //         p = tr[p][t[i] - 'a'];
    //         for (int j = p; j && e[j] != -1; j = fail[j]) res += e[j], e[j] = -1;
    //     }
    //     return res;
    // }

    // 跳 fail 链
    void getfail() {
        queue<int> q;
        fail[0] = 0;
        for (int c = 0; c < __M; ++c) if (tr[0][c]) fail[tr[0][c]] = last[tr[0][c]] = 0, q.push(tr[0][c]);
        while(!q.empty()) {
            int p = q.front(); q.pop();
            for (int c = 0; c < __M; ++c) {
                int u = tr[p][c];
                if (u) {
                    q.push(u);
                    int v = fail[p];
                    while(v && !tr[v][c]) v = fail[v];
                    fail[u] = tr[v][c];
                    last[u] = e[fail[u]] ? fail[u] : last[fail[u]];
                }
            }
        }
    }

    int queryMax(char* t, int n) {
        int p = 0, res = -1, c;
        for (int i = 0; i < n; ++i) {
            c = idx(t[i]);
            while(p && !tr[p][c]) p = fail[p];
            p = tr[p][c];
            for (int j = p; j; j = last[j]) if (e[j]) updMax(res, (*st[j].rbegin()));
        }
        return res;
    }
} // namespace ACAM
```

5.2 GSAM

```
namespace GSAM {
    using T = char;

    inline int idx(T c) { return c - 'a'; }

    const int __N = N << 1;
    const int __M = 26;

    int tot, next[__N][__M];
    int len[__N], fail[__N];

    inline void init() {
        tot = 0;
        fail[0] = -1; len[0] = 0;
        memset(next[0], 0, sizeof(next[0]));
    }
```

```
17
18      inline int newnode() {
19          ++tot;
20          fail[tot] = 0; len[tot] = 0;
21          memset(next[tot], 0, sizeof(next[tot]));
22          return tot;
23      }
24
25      void insertTrie(const T* s, int n) {
26          int p = 0, c;
27          for (int i = 0; i < n; ++i) {
28              c = idx(s[i]);
29              if (!next[p][c]) next[p][c] = newnode();
30              p = next[p][c];
31          }
32      }
33
34      int extendSAM(int last, int c) {
35          int cur = next[last][c];
36          if (len[cur]) return cur;
37          len[cur] = len[last] + 1;
38
39          int p = fail[last];
40          while(p != -1) {
41              if (!next[p][c]) next[p][c] = cur;
42              else break;
43              p = fail[p];
44          }
45
46          if (p == -1) {
47              fail[cur] = 0;
48              return cur;
49          }
50
51          int q = next[p][c];
52          if (len[p] + 1 == len[q]) {
53              fail[cur] = q;
54              return cur;
55          }
56
57          int clone = newnode();
58          for (int i = 0; i < __M; ++i)
59              next[clone][i] = len[next[q][i]] ? next[q][i] : 0;
60
61          len[clone] = len[p] + 1;
62          while(p != -1 && next[p][c] == q) {
63              next[p][c] = clone;
64              p = fail[p];
65          }
66          fail[clone] = fail[q];
67          fail[cur] = clone;
68          fail[q] = clone;
69          return cur;
70      }
71
72      void build() {
73          queue<pair<int, int>> q;
74          for (int i = 0; i < __M; ++i)
75              if (next[0][i]) q.push(make_pair(0, i));
76
77          while(!q.empty()) {
78              pair<int, int> u = q.front(); q.pop();
79              int last = extendSAM(u.first, u.second);
80              for (int i = 0; i < __M; ++i)
81                  if (next[last][i]) q.push(make_pair(last, i));
```

```
        }
    }

    // 多模式串--本质不同子串数
    ll count() {
        ll res = 0;
        for (int i = 1; i <= tot; ++i)
            res += len[i] - len[fail[i]];
        return res;
    }
}
```

5.3 KMP

```
namespace KMP {
    // pi_i = s[0...i] 最长 border
    void getPi(char* s, int n, int* pi) {
        pi[0] = 0;
        for (int i = 1; i < n; ++i) {
            int j = pi[i - 1];
            while(j > 0 && s[j] != s[i]) j = pi[j - 1];
            if (s[i] == s[j]) ++j;
            pi[i] = j;
        }
    }

    vector<int> getAllMatchPosition(char* s, int n, int* pi, char* t, int m) {
        s[n] = '#'; s[n + 1] = 0; ++n;
        KMP::getPi(s, n, pi);

        vector<int> ans;

        int p = 0;
        for (int i = 0; i < m; ++i) {
            while(p > 0 && t[i] != s[p]) p = pi[p - 1];
            if (t[i] == s[p]) {
                ++p;
                if (p == n - 1) {
                    ans.push_back(i + 2 - n);
                }
            }
        }

        return ans;
    }

    int getPeriod(int n, int* pi) {
        return n - pi[n - 1];
    }
}
```

5.4 Manacher

```
namespace Manacher {
    // 1-based

    const int __N = N << 1;

    char s[__N];
    int n, len[__N];

    // @ t1 t2 t3 \0
```

```
    // ==> @ # t1 # t2 # t3 # \0
    inline void init(char* t, int m) {
        n = 2 * m + 1;
        s[0] = '@'; s[n] = '#'; s[n + 1] = 0;
        for (int i = 1; i <= m; ++i) {
            s[2 * i - 1] = '#';
            s[2 * i] = t[i];
        }
    }

    // s[i-len[i]...i+len[i]] is palindromic
    // len[i]-1 is palindromic length in t
    void manacher(char* t, int m) {
        init(t, m);
        for (int i = 1, l = 0, r = 0, k; i <= n; ++i) {
            k = i > r ? 1 : min(r - i, len[l + r - i]);
            while(s[i - k] == s[i + k]) ++k;
            len[i] = k--;
            if (i + k > r) {
                l = i -  k;
                r = i + k;
            }
        }
    }

    int getMaxPalindromicLength(char* t, int m) {
        manacher(t, m);
        int ma = 0;
        for (int i = 1; i <= n; ++i) updMax(ma, len[i]);
        return ma - 1;
    }
}
```

5.5 PAM

```
//最长双倍回文串长度
#include<bits/stdc++.h>
using namespace std;

typedef long long ll;
const int N = 5e5 + 5;

struct Palindromic_Automaton{
    //0 偶根 1 奇根  range[2-tot]
    int s[N << 1],now;
    int next[N << 1][26], fail[N << 1], len[N << 1], last, tot;
    int cnt[N << 1]; //状态 i 表示的回文串数目

    // extend
    int trans[N << 1];

    void init(){
        s[0]=len[1]=-1;
        fail[0]=tot=now=1;
        last=len[0]=0;
        memset(next[0],0,sizeof(next[0]));
        memset(next[1],0,sizeof(next[1]));
    }
    int newnode(){
        tot++;
        memset(next[tot],0,sizeof(next[tot]));
        fail[tot]=cnt[tot]=len[tot]=0;
        return tot;
    }
```

```
    int getfail(int x){
        while(s[now-len[x]-2]!=s[now-1])x=fail[x];
        return x;
    }
    void extend(int c){
        s[now++]=c;
        int cur=getfail(last);
        if(!next[cur][c]){
            int p=newnode();len[p]=len[cur]+2;
            fail[p]=next[getfail(fail[cur])][c];
            next[cur][c]=p;

            // extend
            if(len[p]<=2)trans[p]=fail[p];
            else{
                int tmp=trans[cur];
                while(s[now-len[tmp]-2] != s[now-1] || (len[tmp]+2)*2>len[p])tmp=fail[tmp];
                trans[p]=next[tmp][c];
            }
        }
        last=next[cur][c];
        cnt[last]++;
    }
    int count(){return tot-1;}
    void calc(){
        for(int i=tot;i>=2;--i) cnt[fail[i]]+=cnt[i];
        cnt[0]=cnt[1]=0;
    }
    int getans(){
        int ans=0;
        for(int i=2;i<=tot;i++){
            if(len[i]>ans && len[trans[i]]*2==len[i] && len[trans[i]]%2==0)ans=len[i];
        }
        return ans;
    }
}pam;

char t[N];

int main()
{
    int n;
    scanf("%d",&n);
    scanf("%s",t);
    pam.init();
    for(int i=0;i<n;++i){
        pam.extend(t[i]-'a');
    }
    printf("%d\n",pam.getans());
    return 0;
}
```

5.6 SA

```
namespace SA {
    // 0 based, 倍增法构建, O(nlogn)
    int height[N], c[N], x[N], y[N], sa[N], rk[N];
    void build_sa(int* s, int n) {
        n++;
        int i, j, k, m = 256; //m 为字符集大小, max(s[i])<m
        for (i = 0; i < m; i++) c[i] = 0;
        for (i = 0; i < n; i++) c[x[i] = s[i]]++;
        for (i = 1; i < m; i++) c[i] += c[i - 1];
        for (i = n - 1; i >= 0; i--) sa[--c[x[i]]] = i;
```

```
        for (j = 1; j <= n; j <<= 1) {
            k = 0;
            for (i = n - j; i < n; i++) y[k++] = i;
            for (i = 0; i < n; i++) if (sa[i] >= j) y[k++] = sa[i] - j;
            for (i = 0; i < m; i++) c[i] = 0;
            for (i = 0; i < n; i++) c[x[y[i]]]++;
            for (i = 1; i < m; i++) c[i] += c[i - 1];
            for (i = n - 1; i >= 0; i--) sa[--c[x[y[i]]]] = y[i];
            swap(x, y);
            m = 0;
            x[sa[0]] = m++;
            for (i = 1; i < n; i++) {
                if (y[sa[i]] == y[sa[i - 1]] && y[sa[i] + j] == y[sa[i - 1] + j])
                    x[sa[i]] = m - 1;
                else
                    x[sa[i]] = m++;
            }
            if (m >= n) break;
        }
        k = 0;
        for (i = 0; i < n; i++) rk[sa[i]] = i;
        for (i = 0; i < n - 1; i++) {
            if (k) k--;
            j = sa[rk[i] - 1];
            while (s[i + k] == s[j + k]) k++;
            height[rk[i]] = k;
        }
    }
}
```

5.7 SAIS

```
namespace SAIS {
    // 1 based, O(n)
    int s[N << 1], t[N << 1], height[N], sa[N], rk[N], p[N], c[N], w[N];
    inline int trans(int n, int* S)
    {
        int m = *max_element(S + 1, S + 1 + n);
        for (int i = 1; i <= n; ++i)
            rk[S[i]] = 1;
        for (int i = 1; i <= m; ++i)
            rk[i] += rk[i - 1];
        for (int i = 1; i <= n; ++i)
            s[i] = rk[S[i]];
        return rk[m];
    }
    #define ps(x) sa[w[s[x]]--] = x
    #define pl(x) sa[w[s[x]]++] = x
    inline void radix(int* v, int* s, int* t, int n, int m, int n1)
    {
        memset(sa, 0, n + 1 << 2);
        memset(c, 0, m + 1 << 2);
        for (int i = 1; i <= n; ++i)
            ++c[s[i]];
        for (int i = 1; i <= m; ++i)
            w[i] = c[i] += c[i - 1];
        for (int i = n1; i; --i)
            ps(v[i]);
        for (int i = 1; i <= m; ++i)
            w[i] = c[i - 1] + 1;
        for (int i = 1; i <= n; ++i)
            if (sa[i] > 1 && t[sa[i] - 1])
                pl(sa[i] - 1);
        for (int i = 1; i <= m; ++i)
```

```
            w[i] = c[i];
        for (int i = n; i; --i)
            if (sa[i] > 1 && !t[sa[i] - 1])
                ps(sa[i] - 1);
    }
    inline void SAIS(int n, int m, int* s, int* t, int* p)
    {
        int n1 = 0, ch = rk[1] = 0, *s1 = s + n;
        t[n] = 0;
        for (int i = n - 1; i; --i)
            t[i] = s[i] == s[i + 1] ? t[i + 1] : s[i] > s[i + 1];
        for (int i = 2; i <= n; ++i)
            rk[i] = t[i - 1] && !t[i] ? (p[++n1] = i, n1) : 0;
        radix(p, s, t, n, m, n1);
        for (int i = 1, x, y; i <= n; ++i)
            if (x = rk[sa[i]]) {
                if (ch <= 1 || p[x + 1] - p[x] != p[y + 1] - p[y])
                    ++ch;
                else
                    for (int j = p[x], k = p[y]; j <= p[x + 1]; ++j, ++k)
                        if ((s[j] << 1 | t[j]) ^ (s[k] << 1 | t[k])) {
                            ++ch;
                            break;
                        }
                s1[y = x] = ch;
            }
        if (ch < n1)
            SAIS(n1, ch, s1, t + n, p + n1);
        else
            for (int i = 1; i <= n1; ++i)
                sa[s1[i]] = i;
        for (int i = 1; i <= n1; ++i)
            s1[i] = p[sa[i]];
        radix(s1, s, t, n, m, n1);
    }
    inline void build_sa(int* S, int n)
    {
        int m = trans(++n, S);
        SAIS(n, m, s, t, p);
        for (int i = 1; i < n; ++i)
            rk[sa[i] = sa[i + 1]] = i;
        for (int i = 1, j, k = 0; i < n; ++i)
            if (rk[i] > 1) {
                for (j = sa[rk[i] - 1]; S[i + k] == S[j + k]; ++k)
                    ;
                if (height[rk[i]] = k)
                    --k;
            }
    }
}
```

5.8 SAM

```
//广义后缀自动机: insert 后重新将 last 赋 1  (复杂度好像有可能退化)
#include<bits/stdc++.h>
using namespace std;

typedef long long ll;
const int maxn=1e6+5;

char s[maxn];
struct Suffix_Automaton
{
    //初始状态为 0,range[0...tot-1]
```

```
    struct state{
        int len,link;
        map<char,int>next;
    }st[maxn<<1];
    int last,tot;

    void init(){
        st[0].len=0;st[0].link=-1;
        tot++;
        last=0;
    }

    void extend(char c){
        int cur=tot++;
        st[cur].len=st[last].len+1;
        int p=last;
        while(p!=-1 && !st[p].next.count(c)){
            st[p].next[c]=cur;
            p=st[p].link;
        }
        if(p==-1)st[cur].link=0;
        else{
            int q=st[p].next[c];
            if(st[p].len+1==st[q].len)st[cur].link=q;
            else{
                int clone=tot++;
                st[clone].len=st[p].len+1;
                st[clone].next=st[q].next;
                st[clone].link=st[q].link;
                while(p!=-1 && st[p].next[c]==q){
                    st[p].next[c]=clone;
                    p=st[p].link;
                }
                st[q].link=st[cur].link=clone;
            }
        }
        last=cur;
    }

    ll count(){
        ll res=0;
        for(int i=0;i<tot;i++)res+=st[i].len-st[st[i].link].len;
        return res;
    }
} sam;

int main()
{
    scanf("%s",s);
    sam.init();
    for(int i=0;s[i]!=0;i++)sam.extend(s[i]);
    printf("%lld\n",sam.count());
    return 0;
}
```

5.9 SqAM

```
/**
 * 识别一个串的子序列, O(n^2)
 * 用法类似后缀自动机
*/
struct SqAM{
    int next[N << 1][26], pre[N << 1], lst[26];
    int root, tot;
```

```
    void init(){
        root = tot = 1;
        for(int i = 0; i < 26; i++) lst[i] = 1;
    }

    void extend(int c){
        int p = lst[c], np = ++tot;
        pre[np] = p;
        for (int i = 0; i < 26; i++)
        for (int j = lst[i]; j && !next[j][c]; j = pre[j])
            next[j][c] = np;
        lst[c]=np;
    }
};
```

5.10 string-hash

```
namespace Hash {
    // 1 based, double hash
    typedef long long ll;
    const ll P1 = 29;
    const ll P2 = 131;
    const ll MOD1 = 1e9 + 7;
    const ll MOD2 = 1e9 + 9;
    ll p1[N], p2[N], h1[N], h2[N];
    void init_hash(char* s, int n) {
        p1[0] = p2[0] = 1;
        for(int i = 1; i <= n; i++) p1[i] = (p1[i - 1] * P1) % MOD1;
        for(int i = 1; i <= n; i++) p2[i] = (p2[i - 1] * P2) % MOD2;
        for(int i = 1; i <= n; i++) h1[i] = (h1[i - 1] * P1 + s[i]) % MOD1;
        for(int i = 1; i <= n; i++) h2[i] = (h2[i - 1] * P2 + s[i]) % MOD2;
    }

    ll get_hash(int l, int r) {
        ll H1 = ((h1[r] - h1[l - 1] * p1[r - l + 1]) % MOD1 + MOD1) % MOD1;
        ll H2 = ((h2[r] - h2[l - 1] * p2[r - l + 1]) % MOD2 + MOD2) % MOD2;
        return H1 * MOD2 + H2;
    }
}
```

5.11 Trie

```
namespace Trie {
    // 1-based
    const int __N = 4e6 + 5;
    const int __M = 26;
    int tot;
    int ch[__N][__M];
    int f[__N], e[__N];

    inline void init() {
        tot = 0;
        memset(ch[0], 0, sizeof(ch[0]));
        f[0] = e[0] = 0;
    }

    inline int newnode() {
        ++tot;
        memset(ch[tot], 0, sizeof(ch[tot]));
        f[tot] = e[tot] = 0;
        return tot;
    }
```

```

    inline int idx(char c) { return c - 'a'; }

    void insert(char* s) {
        int n = strlen(s + 1), p = 0, c;
        for (int i = 1; i <= n; ++i) {
            c = idx(s[i]);
            if (!ch[p][c]) ch[p][c] = newnode();
            p = ch[p][c];
            ++f[p];
        }
        ++e[p];
    }

    int query(char* s) {
        int p = 0, n = strlen(s + 1), c;
        for(int i = 1; i <= n; i++){
            c = idx(s[i]);
            if(!ch[p][c]) return 0;
            p = ch[p][c];
        }
        return e[p];
    }
}
```

5.12 ZAlgorithm

```
namespace ZAlgorithm {
    // 1-based

    // z_i = LCP(s, s[i..n])
    void getZ(char* s, int n, int* z) {
        z[1] = n;
        for (int i = 2, l = 0, r = 0; i <= n; ++i) {
            if (i <= r) z[i] = min(r - i + 1, z[i - l + 1]);
            else z[i] = 0;
            while(i + z[i] <= n && s[z[i] + 1] == s[i + z[i]]) ++z[i];
            if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
        }
    }

    // p_i = LCP(s, t[i...m])
    void EXKMP(char* s, int n, int* z, char* t, int m, int* p) {
        getZ(s, n, z);
        for (int i = 1, l = 0, r = 0; i <= m; ++i) {
            if (i <= r) p[i] = min(r - i + 1, z[i - l + 1]);
            else p[i] = 0;
            while(i + p[i] <= m && s[p[i] + 1] == t[i + p[i]]) ++p[i];
            if (i + p[i] - 1 > r) l = i, r = i + p[i] - 1;
        }
    }
}
```