Backlight's Code Template

Backlight @ CSU

2021 年 10 月 16 日

目录

1 ds **1**

1.1 AVLTree . . . 1
1.2 BTree . . . 5
1.3 CaptainMo . . . 9
1.4 CDQ . . . 10
1.5 FenwickTree . . . 12
1.6 KDTree . . . 12
1.7 LCT . . . 15
1.8 LefitstTree . . . 17
1.9 PersistentSegmentTree . . . 18
1.10 rbtree-1 . . . 19
1.11 RBTree . . . 28
1.12 RMQ . . . 33
1.13 RollBackCaptainMo . . . 33
1.14 SegmentTree . . . 35
1.15 SGTree . . . 39
1.16 Splay . . . 42
1.17 Treap-pointer . . . 45
1.18 Treap . . . 47

2 graph **49**

2.1 BCC-Edge . . . 49
2.2 BCC-Point . . . 51
2.3 BiGraphMatch . . . 52
2.4 BiWraphMatch . . . 54
2.5 BlockForest . . . 56
2.6 BlockTree . . . 57
2.7 Dijkstra . . . 61
2.8 dsu-on-tree . . . 62
2.9 ExKruskal . . . 63
2.10 FullyDCP . . . 66
2.11 Graph . . . 79
2.12 GraphMatch . . . 80
2.13 HLD-Edge . . . 83
2.14 Kosaraju . . . 87
2.15 Kruskal . . . 88
2.16 LCA-HLD . . . 89
2.17 LCA . . . 90
2.18 maxflow . . . 91
2.19 mincostflow . . . 93
2.20 SCC . . . 94
2.21 SPFA . . . 96
2.22 tree-divide . . . 97
2.23 Wraph . . . 98
2.24 WraphMatch . . . 99

3 math **115**

3.1 2DGeometry . . . 115
3.2 3DGeometry . . . 128
3.3 BigInt . . . 134
3.4 BSGS . . . 136
3.5 Cipolla . . . 138
3.6 Combination . . . 139
3.7 CRT . . . 139
3.8 du . . . 140
3.9 EulerSeive . . . 142
3.10 eval . . . 142
3.11 EXGCD . . . 143
3.12 FFT . . . 144
3.13 FWT . . . 146
3.14 LinearBasis . . . 148
3.15 Lucas . . . 149
3.16 min25 . . . 151
3.17 Mint . . . 154

3.18 Mobius . . . 158
3.19 Modular . . . 158
3.20 NTT . . . 159
3.21 PollardRho . . . 160
3.22 poly-struct . . . 163
3.23 Poly . . . 169
3.24 PowerfulNumber . . . 176
3.25 Simplex . . . 178
3.26 SimpsonIntegral . . . 181

4 other **181**
4.1 BFPRT . . . 181
4.2 cpp-header . . . 182
4.3 debug . . . 187
4.4 java-header . . . 191
4.5 SimulateAnneal . . . 192

5 string **193**
5.1 ACAM . . . 193
5.2 GSAM . . . 195
5.3 KMP . . . 196
5.4 Manacher . . . 197
5.5 PAM . . . 198
5.6 SA . . . 199
5.7 SAIS . . . 200
5.8 SAM . . . 201
5.9 SqAM . . . 202
5.10 string-hash . . . 203
5.11 SuffixBST . . . 203
5.12 Trie . . . 207
5.13 ZAlgorithm . . . 207

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;
    explicit 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;
  }

  void maintain(node*& p) {
    if (!p)
      return;
```

```
62    if (get_height(p->l) - get_height(p->r) == -2) {
63      if (p->r && get_height(p->r->l) > get_height(p->r->r)) {
64        zag(p->r);
65      }
66      zig(p);
67      return;
68    }
69    if (get_height(p->l) - get_height(p->r) == 2) {
70      if (p->l && get_height(p->l->l) < get_height(p->l->r)) {
71        zig(p->l);
72      }
73      zag(p);
74      return;
75    }
76  }
77
78  void ins(node*& p, T v) {
79    if (!p) {
80      p = new node(v);
81      return;
82    }
83    if (p->v == v) {
84      ++(p->cnt);
85    } else {
86      if (v < p->v) {
87        ins(p->l, v);
88      } else {
89        ins(p->r, v);
90      }
91    }
92    push_up(p);
93    maintain(p);
94    push_up(p);
95  }
96
97  void del(node*& p, T v) {
98    if (!p)
99      return;
100   if (p->v == v) {
101     --p->cnt;
102     if (p->cnt == 0) {
103       if (p->l && p->r) {
104         node* q = p->r;
105         while (q->l)
106           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)
113           p = p->l;
114         else if (p->r)
115           p = p->r;
116         else
117           p = nullptr;
118         delete q;
119         q = nullptr;
120       }
121     }
122   } else {
123     if (v < p->v)
124       del(p->l, v);
125     else
126       del(p->r, v);
```

```
    }
    push_up(p);
    maintain(p);
    push_up(p);
  }

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

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

  int getRank(T v) {
    node* p = root;
    int res = 0;
    while (p) {
      if (v == p->v) {
        res += get_size(p->l);
        break;
      }
      if (v < p->v)
        p = p->l;
      else {
        res += get_size(p->l) + p->cnt;
        p = p->r;
      }
    }
    return res + 1;
  }

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

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

      if (v < p->v) {
        p = p->l;
```

```
192       } else {
193         if (p->v > res)
194           res = p->v;
195         p = p->r;
196       }
197     }
198     return res;
199   }
200
201   T getSucc(T v) {
202     T res = numeric_limits<T>::max();
203     node* p = root;
204     while (p) {
205       if (v == p->v) {
206         node* q = p->r;
207         if (q) {
208           while (q->l)
209             q = q->l;
210           res = q->v;
211         }
212         break;
213       }
214
215       if (v < p->v) {
216         if (p->v < res)
217           res = p->v;
218         p = p->l;
219       } else {
220         p = p->r;
221       }
222     }
223     return res;
224   }
225
226   void debug(node* p) {
227     if (!p)
228       return;
229     debug(p->l);
230     cerr << p->v << " ";
231     debug(p->r);
232   }
233
234   void debug() {
235     cerr << "INORDER: " << endl;
236     debug(root);
237     cerr << endl;
238   }
239 };
240
241 void solve(int Case) {
242   int n;
243   scanf("%d", &n);
244   int op, x;
245   AVLTree<int> t;
246   FOR(i, 1, n) {
247     scanf("%d %d", &op, &x);
248     //        cerr << op << " " << x << endl;
249     switch (op) {
250       case 1:
251         t.ins(x);
252         break;
253       case 2:
254         t.del(x);
255         break;
256       case 3:
```

```
257         printf("%d\n", t.getRank(x));
258         break;
259       case 4:
260         printf("%d\n", t.getKth(x));
261         break;
262       case 5:
263         printf("%d\n", t.getPrev(x));
264         break;
265       case 6:
266         printf("%d\n", t.getSucc(x));
267         break;
268     }
269     //        t.debug();
270   }
271 }
272
273 int main() {
274 #ifdef BACKLIGHT
275   freopen("a.in", "r", stdin);
276 #endif
277   int T = 1;
278   //    scanf("%d", &T);
279   for (int _ = 1; _ <= T; ++_)
280     solve(_);
281   return 0;
282 }
```

1.2 BTree

```
1  template <typename K, int BF>
2  class BTree {
3   public:
4    typedef std::pair<K, int> value_type;
5
6   private:
7    struct Node {
8      value_type values[2 * BF - 1];
9      Node* child[2 * BF] = {nullptr};
10     Node* p = nullptr;
11     int keyNum = 0, size = 0;
12     bool isLeaf = true;
13     const K& key(int i) const { return values[i].first; }
14     int& cnt(int i) { return values[i].second; }
15     Node(Node* p = nullptr)
16         : p(p) {}
17   };
18   Node* root = nullptr;
19   static bool pairComp(const value_type& lhs, const K& rhs) { return lhs.first < rhs; }
20   template <typename T>
21   static void shiftBy(T* ptr, int length, int shift) { memmove(ptr + shift, ptr, length * sizeof(T)); }
22   static int calcSize(Node* x) {
23     if (!x)
24       return 0;
25     int nsz = 0;
26     for (int i = 0; i < x->keyNum; ++i)
27       nsz += getSize(x->child[i]) + x->cnt(i);
28     nsz += getSize(x->child[x->keyNum]);
29     return nsz;
30   }
31   static int getSize(Node* x) {
32     if (!x)
33       return 0;
34     return x->size;
35   }
```

```cpp
//把 where 孩子分成两个节点，都作为 x 的孩子
void split(Node* x, int where) {
  Node* z = new Node(x);
  Node* y = x->child[where];
  z->isLeaf = y->isLeaf;
  memmove(z->values, y->values + BF, (BF - 1) * sizeof(value_type));
  if (!y->isLeaf) {
    memmove(z->child, y->child + BF, BF * sizeof(Node*));
    for (int i = 0; i < BF; ++i)
      z->child[i]->p = z;
  }
  z->keyNum = y->keyNum = BF - 1;
  shiftBy(x->child + where + 1, x->keyNum - where, 1);  //注意 child 本身 keyNum 多一个
  x->child[where + 1] = z;
  shiftBy(x->values + where, x->keyNum - where, 1);
  new (x->values + where) value_type(y->values[BF - 1]);

  y->size = calcSize(y), z->size = calcSize(z);
  ++x->keyNum;
}
void insertEmpty(Node* x, const K& key) {
  while (true) {
    int i = lower_bound(x->values, x->values + x->keyNum, key, pairComp) - x->values;
    if (i != x->keyNum && !(key < x->values[i].first))  //重复插入
    {
      ++x->cnt(i);
      while (x)
        ++x->size, x = x->p;
      return;
    }
    if (x->isLeaf) {
      shiftBy(x->values + i, x->keyNum - i, 1);
      x->values[i] = {key, 1};
      ++x->keyNum;
      while (x)
        ++x->size, x = x->p;
      return;
    }
    if (x->child[i]->keyNum == 2 * BF - 1) {
      split(x, i);
      if (x->key(i) < key)
        ++i;
      else if (!(key < x->key(i))) {
        ++x->cnt(i);
        while (x)
          ++x->size, x = x->p;
        return;
      }
    }
    x = x->child[i];
  }
}

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 2
{
  Node *y = x->child[i], *z = x->child[i + 1];
  y->keyNum = 2 * BF - 1;
  y->values[BF - 1] = std::move(x->values[i]);
  memmove(y->values + BF, z->values, (BF - 1) * sizeof(value_type));
  if (!y->isLeaf) {
    memmove(y->child + BF, z->child, BF * sizeof(Node*));
    for (int j = BF; j <= 2 * BF - 1; ++j)
      y->child[j]->p = y;
  }
  shiftBy(x->values + i + 1, x->keyNum - i - 1, -1);
```

```
    shiftBy(x->child + i + 2, x->keyNum - i - 1, -1);

    --x->keyNum;
    y->size = calcSize(y);
  }
  void erase(Node* x, const K& key) {
    int i = lower_bound(x->values, x->values + x->keyNum, key, pairComp) - x->values;
    if (i != x->keyNum && !(key < x->values[i].first))  //找到 key 了
    {
      if (x->cnt(i) > 1) {
        --x->cnt(i);
        while (x)
          --x->size, x = x->p;
        return;
      }
      if (x->isLeaf)  //x 是叶节点，直接删除
      {
        shiftBy(x->values + i + 1, --x->keyNum - i, -1);  //需要移动的内存是 x->keyNum-i-1
        while (x)
          --x->size, x = x->p;
      } else {
        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)
        {
          Node* y = x->child[i];
          while (!y->isLeaf)
            y = y->child[y->keyNum];  //找前驱
          x->values[i] = y->values[y->keyNum - 1];
          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
          {
            y->cnt(y->keyNum - 1) = 1;
            while (y != x)
              y->size -= x->cnt(i) - 1, y = y->p;
          }

          erase(x->child[i], x->key(i));
        } else if (x->child[i + 1]->keyNum >= BF)  //后继所在孩子有足够的孩子
        {
          Node* y = x->child[i + 1];
          while (!y->isLeaf)
            y = y->child[0];  //找后继
          x->values[i] = y->values[0];
          if (x->cnt(i) != 1) {
            y->cnt(0) = 1;
            while (y != x)
              y->size -= x->cnt(i) - 1, y = y->p;
          }

          erase(x->child[i + 1], x->key(i));
        } 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
        {
          merge(x, i);
          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];
```

```
166           shiftBy(y->values, y->keyNum, 1);
167           //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
168           //所以，value 的构造必须要用 new 不能用 =，从而避开 = 的资源释放
169           //但是 value 的移动似乎应该是 bitwise 的，考虑 std::move
170           new (y->values) value_type(std::move(x->values[i - 1]));
171           new (x->values + i - 1) value_type(std::move(z->values[z->keyNum - 1]));
172           if (!y->isLeaf) {
173             shiftBy(y->child, y->keyNum + 1, 1);
174             y->child[0] = z->child[z->keyNum], y->child[0]->p = y;
175           }
176
177           --z->keyNum, ++y->keyNum;
178           y->size = calcSize(y), z->size = calcSize(z);
179           erase(y, key);
180         } else if (i < x->keyNum && x->child[i + 1]->keyNum >= BF)  //右兄弟，取走它的最小孩子
181         {
182           Node* z = x->child[i + 1];
183           new (y->values + y->keyNum) value_type(std::move(x->values[i]));
184           new (x->values + i) value_type(std::move(z->values[0]));
185           if (!y->isLeaf)  //y 和 z 深度一样，isLeaf 情况相同
186           {
187             y->child[y->keyNum + 1] = z->child[0], y->child[y->keyNum + 1]->p = y;
188             shiftBy(z->child + 1, z->keyNum, -1);
189           }
190           shiftBy(z->values + 1, z->keyNum - 1, -1);
191
192           --z->keyNum, ++y->keyNum;
193           y->size = calcSize(y), z->size = calcSize(z);
194           erase(y, key);
195         } 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
196         {
197           if (i != 0)
198             --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
199           y = x->child[i];
200           merge(x, i);
201           if (root->keyNum == 0)
202             root = y, root->p = nullptr;
203           erase(y, key);
204         }
205       } else
206         erase(x->child[i], key);
207     }
208   }
209
210   public:
211   BTree()
212       : root(new Node) {}
213   void insert(const K& key) {
214     //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
215     //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
216     if (root->keyNum == 2 * BF - 1) {
217       Node* x = new Node;
218       x->isLeaf = false, x->child[0] = root, x->size = root->size;  //+1 操作由 insertEmpty 来做
219       root->p = x, root = x;
220       split(x, 0);  //split 接受参数: node 的满子节点下标
221     }
222     insertEmpty(root, key);
223   }
224   void erase(const K& key) { erase(root, key); }
225   int next(const K& key) {
226     Node* x = root;
227     int ret;
228     while (x) {
229       int i = lower_bound(x->values, x->values + x->keyNum, key, pairComp) - x->values;
230       if (x->values[i].first == key)
```

```
      ++i;
    if (i != x->keyNum)
      ret = x->values[i].first;
    x = x->child[i];
  }
  return ret;
}
int prev(const K& key) {
  Node* x = root;
  int ret;
  while (x) {
    int i = lower_bound(x->values, x->values + x->keyNum, key, pairComp) - x->values;
    if (i)
      ret = x->values[i - 1].first;
    x = x->child[i];
  }
  return ret;
}
int rank(const K& key) {
  Node* x = root;
  int ret = 0;
  while (x) {
    if (x->key(x->keyNum - 1) < key) {
      ret += x->size - getSize(x->child[x->keyNum]);
      x = x->child[x->keyNum];
      continue;
    }
    for (int i = 0; i < x->keyNum; ++i) {
      if (x->key(i) < key)
        ret += getSize(x->child[i]) + x->cnt(i);
      else if (x->key(i) == key)
        return ret + getSize(x->child[i]) + 1;
      else {
        x = x->child[i];
        break;
      }
    }
  }
  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 CDQ

```
// P3810
// 3-D Partial Order(a_j <= a_iandb_j <= b_i&&c_j <= c_i)
#include <bits/stdc++.h>
using namespace std;
using ll = int64_t;

const int N = 1e5 + 5;
const int MAXV = 2e5 + 5;
int k, c[MAXV];
inline int lb(int x) {
  return x & (-x);
}
void add(int x, int d) {
  for (; x <= k; x += lb(x))
```

```
    c[x] += d;
}
int getsum(int x) {
  int ret = 0;
  for (; x; x -= lb(x))
    ret += c[x];
  return ret;
}

int n, m;
struct node {
  int a, b, c, w, ans;
  void input() {
    scanf("%d %d %d", &a, &b, &c);
    w = ans = 0;
  }
} a[N], b[N];

int cnt[N];
void CDQ(int l, int r) {
  if (l == r)
    return;
  int mid = (l + r) >> 1;
  CDQ(l, mid);
  CDQ(mid + 1, r);

  sort(a + l, a + 1 + mid, [](const node& x, const node& y) -> bool {
    if (x.b != y.b)
      return x.b < y.b;
    return x.c < y.c;
  });
  sort(a + mid + 1, a + 1 + r, [](const node& x, const node& y) -> bool {
    if (x.b != y.b)
      return x.b < y.b;
    return x.c < y.c;
  });

  int p1 = l, p2 = mid + 1;
  for (; p2 <= r; ++p2) {
    while (a[p1].b <= a[p2].b && p1 <= mid) {
      add(a[p1].c, a[p1].w);
      ++p1;
    }
    a[p2].ans += getsum(a[p2].c);
  }

  for (int i = l; i < p1; ++i)
    add(a[i].c, -a[i].w);
}

int main() {
  scanf("%d %d", &m, &k);
  for (int i = 1; i <= m; ++i)
    b[i].input();

  sort(b + 1, b + 1 + m, [](const node& x, const node& y) -> bool {
    if (x.a != y.a)
      return x.a < y.a;
    if (x.b != y.b)
      return x.b < y.b;
    return x.c < y.c;
  });

  int w = 0;
  for (int i = 1; i <= m; ++i) {
```

```
    ++w;
    if (b[i].a != b[i + 1].a || b[i].b != b[i + 1].b || b[i].c != b[i + 1].c) {
      ++n;
      a[n] = b[i];
      a[n].w = w;
      w = 0;
    }
  }
  CDQ(1, n);
  for (int i = 1; i <= n; ++i)
    if (a[i].ans + a[i].w - 1 < m)
      cnt[a[i].ans + a[i].w - 1] += a[i].w;
  for (int i = 0; i < m; ++i)
    printf("%d\n", cnt[i]);
  return 0;
}
```

1.5 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 = __lg(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;
  }
};
```

1.6 KDTree

```
// KDTree
// 平面最近点对
template <typename T, int K = 2>
struct KDTree {
  using node = array<double, K>;
  int n;
  node p[N], ma[N], mi[N];
  double L[N], R[N], D[N], U[N];
  int sd[N], lc[N], rc[N];

```

```
  KDTree(int _n)
      : n(_n) {
  }

  double dist(const node& nd1, const node& nd2) {
    double res = 0;
    for (int j = 0; j < K; ++j) {
      res += (nd1[j] - nd2[j]) * (nd1[j] - nd2[j]);
    }
    return res;
  }

  double dist(int x, int y) {
    return dist(p[x], p[y]);
  }

  double cost(int x, int y) {
    double res = 0;
    for (int j = 0; j < K; ++j) {
      if (mi[y][j] > p[x][j])
        res += (mi[y][j] - p[x][j]) * (mi[y][j] - p[x][j]);
      if (ma[y][j] < p[x][j])
        res += (ma[y][j] - p[x][j]) * (ma[y][j] - p[x][j]);
    }
    return res;
  }

  struct cmp {
    int s;
    cmp(int _s)
        : s(_s) {
    }
    bool operator()(const node& nd1, const node& nd2) const {
      return nd1[s] < nd2[s];
    }
  };

  void maintain(int x) {
    ma[x] = mi[x] = p[x];
    if (lc[x]) {
      for (int j = 0; j < K; ++j) {
        ma[x][j] = max(ma[x][j], ma[lc[x]][j]);
        mi[x][j] = min(mi[x][j], mi[lc[x]][j]);
      }
    }
    if (rc[x]) {
      for (int j = 0; j < K; ++j) {
        ma[x][j] = max(ma[x][j], ma[rc[x]][j]);
        mi[x][j] = min(mi[x][j], mi[rc[x]][j]);
      }
    }
  }

  int build(int l, int r) {
    if (l >= r)
      return 0;
    int mid = (l + r) >> 1;

    array<double, K> avg;
    for (int i = l; i <= r; ++i)
      for (int j = 0; j < K; ++j)
        avg[j] += p[i][j];
    for (int j = 0; j < K; ++j)
      avg[j] /= (r - l + 1);

```

```
    array<double, K> var;
    for (int i = l; i <= r; ++i)
      for (int j = 0; j < K; ++j)
        var[j] += (p[i][j] - avg[j]) * (p[i][j] - avg[j]);

    sd[mid] = 0;
    for (int j = 0; j < K; ++j)
      if (var[j] > var[sd[mid]])
        sd[mid] = j;

    nth_element(p + l, p + mid, p + r + 1, cmp(sd[mid]));

    lc[mid] = build(l, mid - 1);
    rc[mid] = build(mid + 1, r);

    maintain(mid);

    return mid;
  }

  double min_dist;

  void query(int l, int r, int x) {
    if (l > r)
      return;
    int mid = (l + r) >> 1;
    if (mid != x)
      min_dist = min(min_dist, dist(x, mid));
    if (l == r)
      return;

    double dl = cost(x, lc[mid]);
    double dr = cost(x, rc[mid]);

    if (dl < min_dist && dr < min_dist) {
      if (dl < dr) {
        query(l, mid - 1, x);
        if (dr < min_dist)
          query(mid + 1, r, x);
      } else {
        query(mid + 1, r, x);
        if (dl < min_dist)
          query(l, mid - 1, x);
      }
    } else {
      if (dl < min_dist)
        query(l, mid - 1, x);
      if (dr < min_dist)
        query(mid + 1, r, x);
    }
  }

  double getMindis() {
    min_dist = 2e18;
    for (int i = 1; i <= n; ++i)
      query(1, n, i);
    min_dist = sqrt(min_dist);
    return min_dist;
  }
};
```

1.7 LCT

```cpp
template <typename T>
struct LinkCutTree {
#define ls ch[x][0]
#define rs ch[x][1]
#define SIZE 100005

  int tot, sz[SIZE], rev[SIZE], ch[SIZE][2], fa[SIZE];
  T v[SIZE], sum[SIZE];

  LinkCutTree() { tot = 0; }

  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];
  }
};

void solve(int Case) {
  /* write code here */
  /* gl & hf */
  int n, m;
  rd(n, m);
  VI a(n + 1);
  FOR(i, 1, n)
  rd(a[i]);
  LinkCutTree<int> t;
```

```
  FOR(i, 1, n)
  t.newnode(a[i]);

  int op, x, y;
  FOR(_, 1, m) {
    rd(op, x, y);
    debug(op, x, y);
    if (op == 0) {
      pln(t.query(x, y));
    } else if (op == 1) {
      t.link(x, y);
    } else if (op == 2) {
      t.cut(x, y);
    } else {
      t.v[x] = y;
      t.makeroot(x);
    }
  }
}
```

1.8 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);
    }
  }
}
```

1.9 PersistentSegmentTree

```
struct PersistentSegmentTree {
// SIZE = N log N
#define SIZE 200005 * 20

  int tot;
  int c[SIZE];
  int L[SIZE], R[SIZE];

  PersistentSegmentTree() { tot = 0; }

  int update(int rt, int l, int r, int p, int d) {
    int nrt = ++tot;
    L[nrt] = L[rt];
    R[nrt] = R[rt];
    c[nrt] = c[rt] + d;

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

    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 left_size = c[L[v]] - c[L[u]];
    int mid = (l + r) >> 1;
    if (k <= left_size)
      return query(L[u], L[v], l, mid, k);
    return query(R[u], R[v], mid + 1, r, k - left_size);
  }
};
```

1.10 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);
  printf("------in-vis------\n");
  invis(_root, 0);
  printf("------post-vis------\n");
  postvis(_root, 0);
}

template <typename T>
void rbtree<T>::checkconnect(Node* ptn) {
  if (!ptn)
    return;
  assert(ptn->s > 0);
  if (ptn->lc && ptn->lc->ftr != ptn) {
    printf("Oops! %d has a lc %d, but it failed to point its ftr!\n", ptn->val, ptn->lc->val);
  }
  if (ptn->rc && ptn->rc->ftr != ptn) {
    printf("Oops! %d has a rc %d, but it failed to point its ftr!\n", ptn->val, ptn->rc->val);
  }
  int sss = ptn->s;
  if (ptn->lc)
    sss -= ptn->lc->s;
  if (ptn->rc)
    sss -= ptn->rc->s;
  if (sss - 1) {
    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);
  }
  checkconnect(ptn->lc);
  checkconnect(ptn->rc);
}

template <typename T>
void rbtree<T>::correctlyconnected() {
  checkconnect(_root);
}

#endif

template <typename T>
void rbtree<T>::init(T v) {
  _root = new Node(v, false, NULL, NULL, NULL, 1);
#ifdef __REDBLACK_DEBUG
  ++blackheight;
#endif
}

template <typename T>
void rbtree<T>::SolveDoubleRed(Node* nn) {
  while ((!(nn->ftr)) || nn->ftr->RBc) {
    if (nn == _root) {
      _root->RBc = false;
#ifdef __REDBLACK_DEBUG
      ++blackheight;
#endif
      return;
    }
    Node* pftr = nn->ftr;
    if (!(pftr->RBc))
      return;  ////No double-red
    Node* uncle = bro(nn->ftr);
    Node* grdftr = nn->ftr->ftr;
    if (uncle != NULL && uncle->RBc) {  ////RR-2
      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;
```

```
439     else
440       ptn = ptn->rc;
441   }
442   return ptn;
443 }
444
445 template <typename T>
446 typename rbtree<T>::Node* rbtree<T>::rfind(T v, const int op) {
447   Node* ptn = _root;
448   _hot = NULL;
449   while (ptn != NULL && ptn->val != v) {
450     _hot = ptn;
451     ptn->s += op;
452     if (ptn->val > v)
453       ptn = ptn->lc;
454     else
455       ptn = ptn->rc;
456   }
457   return ptn;
458 }
459
460 template <typename T>
461 struct rbtree<T>::iterator {
462  private:
463   Node* _real__node;
464
465  public:
466   iterator& operator++() {
467     _real__node = _real__node->right_node();
468     return *this;
469   }
470
471   iterator& operator--() {
472     _real__node = _real__node->left_node();
473     return *this;
474   }
475
476   T operator*() {
477     return _real__node->val;
478   }
479
480   iterator(Node* node_nn = NULL)
481       : _real__node(node_nn) {
482   }
483   iterator(T const& val_vv)
484       : _real__node(rfind(val_vv, 0)) {
485   }
486   iterator(iterator const& iter)
487       : _real__node(iter._real__node) {
488   }
489 };
490
491 template <typename T>
492 typename rbtree<T>::iterator rbtree<T>::insert(T v) {
493   Node* ptn = find(v, 1);
494   if (_hot == NULL) {
495     init(v);
496     return iterator(_root);
497   }
498   ptn = new Node(v, true, _hot, NULL, NULL, 1);
499   if (_hot->val <= v)
500     _hot->rc = ptn;
501   else
502     _hot->lc = ptn;
503   SolveDoubleRed(ptn);
```

```
504   return iterator(ptn);
505 }
506
507 template <typename T>
508 bool rbtree<T>::remove(T v) {
509   Node* ptn = rfind(v, -1);
510   if (!ptn)
511     return false;
512   Node* node_suc;
513   while (ptn->lc || ptn->rc) {
514     if (!(ptn->lc)) {
515       node_suc = ptn->rc;
516     } else if (!(ptn->rc)) {
517       node_suc = ptn->lc;
518     } else {
519       node_suc = ptn->succ();
520     }
521     --(ptn->s);
522     ptn->val = node_suc->val;
523     ptn = node_suc;
524   }
525   if (!(ptn->RBc)) {
526     --(ptn->s);
527     SolveDoubleBlack(ptn);
528   }
529   if (ptn->ftr->lc == ptn)
530     ptn->ftr->lc = NULL;
531   else
532     ptn->ftr->rc = NULL;
533   delete ptn;
534   return true;
535 }
536
537 template <typename T>
538 int rbtree<T>::size() {
539   return _root->s;
540 }
541
542 template <typename T>
543 typename rbtree<T>::iterator rbtree<T>::kth(int rank) {
544   return iterator(findkth(rank, _root));
545 }
546
547 template <typename T>
548 typename rbtree<T>::iterator rbtree<T>::lower_bound(T v) {
549   Node* ptn = _root;
550   while (ptn) {
551     _hot = ptn;
552     if (ptn->val < v) {
553       ptn = ptn->rc;
554     } else {
555       ptn = ptn->lc;
556     }
557   }
558   if (_hot->val < v) {
559     ptn = _hot;
560   } else {
561     ptn = _hot->left_node();
562   }
563   return iterator(ptn);
564 }
565
566 template <typename T>
567 typename rbtree<T>::iterator rbtree<T>::upper_bound(T v) {
568   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);
}
```

1.11 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;
```

```
47    x->p = y;
48    y->sz = x->sz;
49    x->sz = x->l->sz + x->r->sz + x->cnt;
50    return;
51  }
52  void zag(node* x) {
53    node* y = x->l;
54    x->l = y->r;
55    if (y->r != nil)
56      y->r->p = x;
57    y->p = x->p;
58    if (x->p == nil)
59      root = y;
60    else if (x == x->p->l)
61      x->p->l = y;
62    else
63      x->p->r = y;
64    y->r = x;
65    x->p = y;
66    y->sz = x->sz;
67    x->sz = x->l->sz + x->r->sz + x->cnt;
68    return;
69  }
70  void insert_fixup(node* z) {
71    while (z->p->color == 1) {
72      if (z->p == z->p->p->l) {
73        node* y = z->p->p->r;
74        if (y->color == 1) {
75          y->color = z->p->color = 0;
76          z->p->p->color = 1;
77          z = z->p->p;
78        } else {
79          if (z == z->p->r) {
80            z = z->p;
81            zig(z);
82          }
83          z->p->color = 0;
84          z->p->p->color = 1;
85          zag(z->p->p);
86        }
87      } else {
88        node* y = z->p->p->l;
89        if (y->color == 1) {
90          y->color = z->p->color = 0;
91          z->p->p->color = 1;
92          z = z->p->p;
93        } else {
94          if (z == z->p->l) {
95            z = z->p;
96            zag(z);
97          }
98          z->p->color = 0;
99          z->p->p->color = 1;
100         zig(z->p->p);
101       }
102     }
103   }
104   root->color = 0;
105   return;
106 }
107 void transplant(node* x, node* y) {
108   y->p = x->p;
109   if (x->p == nil)
110     root = y;
111   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;
          zag(x->p);
          w = x->p->l;
        }
        if (w->r->color == 0 && w->l->color == 0) {
          w->color = 1;
          x = x->p;
        } else {
          if (w->l->color == 0) {
            w->color = 1;
            w->r->color = 0;
            zig(w);
            w = x->p->l;
          }
          w->color = x->p->color;
          x->p->color = 0;
          w->l->color = 0;
          zag(w->p);
          x = root;
        }
      }
    }
    x->color = 0;
    return;
  }
  void ins(const T& val) {
    node* x = root;
    node* y = nil;
    while (x != nil) {
      y = x;
```

```
177       ++y->sz;
178       if (x->val == val) {
179         ++x->cnt;
180         return;
181       }
182       if (x->val < val)
183         x = x->r;
184       else
185         x = x->l;
186     }
187     node* z = ++s;
188     *z = (node){val, 1, 1, nil, nil, y, 1};
189     if (y == nil)
190       root = z;
191     else {
192       if (y->val < val)
193         y->r = z;
194       else
195         y->l = z;
196     }
197     insert_fixup(z);
198     return;
199   }
200   void del(const T& val) {
201     node* z = root;
202     node* w = nil;
203     while (z != nil) {
204       w = z;
205       --w->sz;
206       if (z->val == val)
207         break;
208       if (z->val < val)
209         z = z->r;
210       else
211         z = z->l;
212     }
213     if (z != nil) {
214       // delete only one node
215       if (z->cnt > 1) {
216         --z->cnt;
217         return;
218       }
219
220       node* y = z;
221       node* x;
222       bool history = y->color;
223       if (z->l == nil) {
224         x = z->r;
225         transplant(z, z->r);
226       } else if (z->r == nil) {
227         x = z->l;
228         transplant(z, z->l);
229       } else {
230         y = find_min(z->r);
231         history = y->color;
232         x = y->r;
233         if (y->p == z)
234           x->p = y;
235         else {
236           node* w = y;
237           while (w != z) {
238             w->sz -= y->cnt;
239             w = w->p;
240           }
241           transplant(y, y->r);
```

```
242           y->r = z->r;
243           y->r->p = y;
244         }
245         transplant(z, y);
246         y->l = z->l;
247         y->l->p = y;
248         y->color = z->color;
249         y->sz = y->l->sz + y->r->sz + y->cnt;
250       }
251       if (history == 0)
252         delete_fixup(x);
253     } else
254       while (w != nil) {
255         ++w->sz;
256         w = w->p;
257       }
258     return;
259   }
260   T getKth(int k) {
261     T res = 0;
262     node* x = root;
263     while (x != nil) {
264       if (x->l->sz + 1 <= k && x->l->sz + x->cnt >= k) {
265         res = x->val;
266         break;
267       } else if (x->l->sz + x->cnt < k) {
268         k -= x->l->sz + x->cnt;
269         x = x->r;
270       } else {
271         x = x->l;
272       }
273     }
274     return res;
275   }
276   int getRank(const T& val) {
277     int rk = 0;
278     node* x = root;
279     while (x != nil) {
280       if (x->val < val) {
281         rk += x->l->sz + x->cnt;
282         x = x->r;
283       } else {
284         if (x->val == val)
285           ++rk;
286         x = x->l;
287       }
288     }
289     return rk;
290   }
291   T getSucc(const T& val) {
292     ins(val);
293     T res = INT_MAX;
294     node* x = find_node(val);
295     if (x->r != nil) {
296       res = find_min(x->r)->val;
297     } else {
298       while (x->p->r == x)
299         x = x->p;
300       if (x->p != nil)
301         res = x->p->val;
302     }
303     del(val);
304     return res;
305   }
306   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;
  }
};
```

1.12 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 - 1)) <= 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]);
}
```

1.13 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();
  }
}
```

1.14 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;
  }

  template <class... T>
  void upd(int x, int l, int r, int L, int R, const T&... v) {
    if (L <= l && r <= R) {
      tr[x].apply(l, r, v...);
      return;
    }
    push_down(x, l, r);
    int lc = x << 1, rc = lc | 1, mid = (l + r) >> 1;
```

```
    node res;
    if (L <= mid)
      upd(lc, l, mid, L, R, v...);
    if (R > mid)
      upd(rc, mid + 1, r, L, R, v...);
    push_up(x);
  }

  int __get_first(int x, int l, int r, const function<bool(const node&)>& f) {
    if (l == r) {
      return l;
    }
    int lc = x << 1, rc = lc | 1, mid = (l + r) >> 1;
    push_down(x, l, r);
    int res;
    if (f(tr[lc]))
      res = __get_first(lc, l, mid, f);
    else
      res = __get_first(rc, mid + 1, r, f);
    push_up(x);
    return res;
  }

  int get_first(int x, int l, int r, int L, int R, const function<bool(const node&)>& f) {
    if (L <= l && r <= R) {
      if (!f(tr[x])) {
        return -1;
      }
      return __get_first(x, l, r, f);
    }
    push_down(x, l, r);
    int lc = x << 1, rc = lc | 1, mid = (l + r) >> 1;
    int res;
    if (L <= mid)
      res = get_first(lc, l, mid, L, R, f);
    if (res == -1 && R > mid)
      res = get_first(rc, mid + 1, r, L, R, f);
    push_up(x);
    return res;
  }

  int __get_last(int x, int l, int r, const function<bool(const node&)>& f) {
    if (l == r) {
      return l;
    }
    int lc = x << 1, rc = lc | 1, mid = (l + r) >> 1;
    push_down(x, l, r);
    int res;
    if (f(tr[lc]))
      res = __get_first(rc, mid + 1, r, f);
    else
      res = __get_first(lc, l, mid, f);
    push_up(x);
    return res;
  }

  int get_last(int x, int l, int r, int L, int R, const function<bool(const node&)>& f) {
    if (L <= l && r <= R) {
      if (!f(tr[x])) {
        return -1;
      }
      return __get_first(x, l, r, f);
    }
    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
  }
};
```

1.15 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;
  }

```

```
20  void push_up(int x) {
21    if (!x)
22      return;
23    int lc = l[x], rc = r[x];
24    s[x] = s[lc] + 1 + s[rc];
25    sz[x] = sz[lc] + cnt[x] + sz[rc];
26    sd[x] = sd[lc] + (cnt[x] != 0) + sd[rc];
27  }
28
29  bool balance(int x) {
30    int lc = l[x], rc = r[x];
31    if (alpha * s[x] <= max(s[lc], s[rc]))
32      return false;
33    if (alpha * s[x] >= sd[x])
34      return false;
35    return true;
36  }
37
38  void flatten(int x) {
39    if (!x)
40      return;
41    flatten(l[x]);
42    if (cnt[x])
43      buf[++buf_size] = x;
44    flatten(r[x]);
45  }
46
47  void build(int& x, int L, int R) {
48    if (L > R) {
49      x = 0;
50      return;
51    }
52    int mid = (L + R) >> 1;
53    x = buf[mid];
54    build(l[x], L, mid - 1);
55    build(r[x], mid + 1, R);
56    push_up(x);
57  }
58
59  void rebuild(int& x) {
60    buf_size = 0;
61    flatten(x);
62    build(x, 1, buf_size);
63  }
64
65  void ins(int& rt, T val) {
66    if (!rt) {
67      rt = new_node(val);
68      return;
69    }
70    if (val == v[rt]) {
71      ++cnt[rt];
72    } else if (val < v[rt]) {
73      ins(l[rt], val);
74    } else {
75      ins(r[rt], val);
76    }
77    push_up(rt);
78    if (!balance(rt))
79      rebuild(rt);
80  }
81
82  void del(int& rt, T val) {
83    if (!rt)
84      return;
```

```
85
86    if (val == v[rt]) {
87      if (cnt[rt])
88        --cnt[rt];
89    } else if (val < v[rt]) {
90      del(l[rt], val);
91    } else {
92      del(r[rt], val);
93    }
94    push_up(rt);
95    if (!balance(rt))
96      rebuild(rt);
97  }
98
99  int getPrevRank(int rt, T val) {
100   if (!rt)
101     return 0;
102   if (v[rt] == val && cnt[rt])
103     return sz[l[rt]];
104   if (v[rt] < val)
105     return sz[l[rt]] + cnt[rt] + getPrevRank(r[rt], val);
106   return getPrevRank(l[rt], val);
107 }
108
109 int getSuccRank(int rt, T val) {
110   if (!rt)
111     return 1;
112   if (v[rt] == val && cnt[rt])
113     return sz[l[rt]] + cnt[rt] + 1;
114   if (v[rt] < val)
115     return sz[l[rt]] + cnt[rt] + getSuccRank(r[rt], val);
116   return getSuccRank(l[rt], val);
117 }
118
119 T getKth(int rt, int k) {
120   if (!rt)
121     return 0;
122   if (k <= sz[l[rt]])
123     return getKth(l[rt], k);
124   if (k - sz[l[rt]] <= cnt[rt])
125     return v[rt];
126   return getKth(r[rt], k - sz[l[rt]] - cnt[rt]);
127 }
128
129 void ins(T val) {
130   ins(root, val);
131 }
132
133 void del(T val) {
134   del(root, val);
135 }
136
137 int getRank(T val) {
138   return getPrevRank(root, val) + 1;
139 }
140
141 T getKth(int k) {
142   return getKth(root, k);
143 }
144
145 T getPrev(T val) {
146   return getKth(getPrevRank(root, val));
147 }
148
149 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;
  }
};
```

1.16 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
```

1.17 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;
  }
};
```

1.18 Treap

```
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
```

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) {
```

```
84      if (!dfn[i]) {
85        tarjan(i, i);
86      }
87    }
88
89    for (int i = 1; i <= V; ++i) {
90      if (!belong[i]) {
91        ++bcc_cnt;
92        bcc.push_back(vector<int>());
93        blood_fill(i);
94      }
95    }
96
97    bcc_size = vector<int>(bcc_cnt + 1);
98    for (int i = 1; i <= bcc_cnt; ++i)
99      bcc_size[i] = bcc[i].size();
100 }
101 };
102
103 }  // namespace Backlight
```

2.2 BCC-Point

```
1  namespace Backlight {
2
3  struct Graph {
4    struct Edge {
5      int u, v;
6      Edge() {}
7      Edge(int _u, int _v)
8          : u(_u), v(_v) {}
9    };
10
11   int V;
12   vector<vector<Edge>> G;
13
14   Graph()
15       : V(0) {}
16   Graph(int _V)
17       : V(_V), G(_V + 1) {}
18
19   inline void addarc(int u, int v) {
20     assert(1 <= u && u <= V);
21     assert(1 <= v && v <= V);
22     G[u].push_back(Edge(u, v));
23   }
24
25   inline void addedge(int u, int v) {
26     addarc(u, v);
27     addarc(v, u);
28   }
29
30   /***********************************************/
31   int bcc_clock;
32   vector<int> dfn, low;
33   vector<vector<int>> bcc;
34   vector<bool> cut;
35   stack<int> stk;
36
37   void tarjan(int u, int fa) {
38     dfn[u] = low[u] = ++bcc_clock;
39     stk.push(u);
40
41     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);
      }
    }
  }
};

}  // namespace Backlight
```

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>
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 BlockTree

```
// 树分块: uv 之间路径上不同的颜色数 (强制在线)
#include <bits/stdc++.h>
using namespace std;

const int N = 4e4 + 5;

int n, m, a[N];
int nt, t[N];

int tot, head[N];
struct edge {
  int v, nxt;
} e[N << 1];
void init(int n) {
  tot = 0;
  for (int i = 1; i <= n; ++i)
    head[i] = 0;
}
void add(int u, int v) {
  ++tot;
  e[tot] = (edge){v, head[u]};
  head[u] = tot;
}
#define fore(i, u) for (int i = head[u]; i; i = e[i].nxt)
```

```

int sz[N], son[N], f[N], h[N], top[N];

void dfs1(int u, int fa) {
  f[u] = fa;
  h[u] = h[fa] + 1;
  sz[u] = 1;
  son[u] = 0;
  fore(i, u) {
    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 k) {
  top[u] = k;
  if (son[u])
    dfs2(son[u], u, k);
  fore(i, u) {
    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;
}

int dep[N], max_dep[N], pa[N];
int key_cnt, keyid[N];

const int COLORCNT = 4e4 + 2;
const int KEYCNT = 101;
const int gap = 400;

bitset<COLORCNT> c[KEYCNT][KEYCNT];

int stk[N], tp;

void dfs_key(int u, int fa) {
  dep[u] = dep[fa] + 1;
  max_dep[u] = dep[u];
  fore(i, u) {
    int v = e[i].v;
    if (v == fa)
      continue;
    dfs_key(v, u);
    if (max_dep[v] > max_dep[u])
      max_dep[u] = max_dep[v];
  }
  if (max_dep[u] - dep[u] >= gap) {
```

```
    keyid[u] = ++key_cnt;
    max_dep[u] = dep[u];
  }
}

void dfs_bitset(int u) {
  if (keyid[u] && u != stk[tp]) {
    for (int x = u; x != stk[tp]; x = f[x])
      c[keyid[stk[tp]]][keyid[u]].set(a[x]);

    for (int i = 1; i < tp; ++i) {
      c[keyid[stk[i]]][keyid[u]] = c[keyid[stk[i]]][keyid[stk[tp]]];
      c[keyid[stk[i]]][keyid[u]] |= c[keyid[stk[tp]]][keyid[u]];
    }
    pa[u] = stk[tp];
    stk[++tp] = u;
  }
  for (int i = head[u]; i; i = e[i].nxt) {
    if (e[i].v != f[u])
      dfs_bitset(e[i].v);
  }
  if (keyid[u])
    --tp;
}

void build_block_tree() {
  key_cnt = 0;
  dfs_key(1, 1);
  if (!keyid[1])
    keyid[1] = ++key_cnt;

  tp = 1;
  stk[1] = 1;
  dfs_bitset(1);
}

bitset<COLORCNT> res;

int query(int u, int v) {
  res.reset();
  int uv = lca(u, v);

  // step 1: jump to nearest key node
  while (u != uv && !keyid[u]) {
    res.set(a[u]);
    u = f[u];
  }
  while (v != uv && !keyid[v]) {
    res.set(a[v]);
    v = f[v];
  }

  // step 2: jump to lowest key node
  int pu = u;
  while (dep[pa[pu]] >= dep[uv])
    pu = pa[pu];
  if (pu != u) {
    res |= c[keyid[pu]][keyid[u]];
    u = pu;
  }

  int pv = v;
  while (dep[pa[pv]] >= dep[uv])
    pv = pa[pv];
  if (pv != v) {
```

```
    res |= c[keyid[pv]][keyid[v]];
    v = pv;
  }

  // step 3: jump to lca
  while (u != uv) {
    res.set(a[u]);
    u = f[u];
  }
  while (v != uv) {
    res.set(a[v]);
    v = f[v];
  }

  // step 4: set lca
  res.set(a[uv]);

  return res.count();
}

void solve(int Case) {
  scanf("%d %d", &n, &m);
  for (int i = 1; i <= n; ++i) {
    scanf("%d", &a[i]);
    t[i] = a[i];
  }

  sort(t + 1, t + 1 + n);
  nt = unique(t + 1, t + 1 + n) - (t + 1);

  for (int i = 1; i <= n; ++i)
    a[i] = lower_bound(t + 1, t + 1 + nt, a[i]) - t;

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

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

  build_block_tree();

  int lastans = 0;
  for (int i = 1; i <= m; ++i) {
    scanf("%d %d", &u, &v);
    u ^= lastans;
    lastans = query(u, v);
    printf("%d\n", lastans);
  }
}

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

2.7 Dijkstra

```cpp
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;
  }
};

}  // namespace Backlight
```

2.8 dsu-on-tree

```cpp
// 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];
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]);
}
```

2.9 ExKruskal

```
// https://loj.ac/p/6021
// https://oi-wiki.org/graph/mst/#kruskal_1
#include <bits/stdc++.h>
using namespace std;
#ifdef BACKLIGHT
#include "debug.h"
#else
#define debug(...)
#endif

const int __BUFFER_SIZE__ = 1 << 20;
bool NEOF = 1;
int __top;
char __buf[__BUFFER_SIZE__], *__p1 = __buf, *__p2 = __buf, __stk[996];
inline char nc() {
  if (!NEOF)
    return EOF;
  if (__p1 == __p2) {
    __p1 = __buf;
    __p2 = __buf + fread(__buf, 1, __BUFFER_SIZE__, stdin);
    if (__p1 == __p2) {
      NEOF = 0;
      return EOF;
    }
  }
  return *__p1++;
}

template <typename T>
inline bool rd(T& x) {
  char c = nc();
  bool f = 0;
  x = 0;
  while (!isdigit(c))
    c == '-' && (f = 1), c = nc();
  while (isdigit(c))
    x = x * 10 + (c ^ 48), c = nc();
  if (f)
    x = -x;
  return NEOF;
}

typedef unsigned long long ull;
ull myRand(ull& k1, ull& k2) {
  ull k3 = k1, k4 = k2;
```

```
  k1 = k4;
  k3 ^= (k3 << 23);
  k2 = k3 ^ k4 ^ (k3 >> 17) ^ (k4 >> 26);
  return k2 + k4;
}
pair<int, int> myRanq(ull& k1, ull& k2, int MAXN) {
  int x = myRand(k1, k2) % MAXN + 1, y = myRand(k1, k2) % MAXN + 1;
  if (x > y)
    return make_pair(y, x);
  else
    return make_pair(x, y);
}

struct LCAGraph {
  int n, m, dfs_clock;
  vector<vector<int>> G;
  vector<int> dfn, lg, dep;
  vector<vector<int>> st;
  LCAGraph(int _n = 0)
      : n(_n), m(n + n - 1), G(n), dfn(n), lg(m + 1), dep(n), st(m) {
    lg[1] = 0;
    for (int i = 2; i <= m; ++i)
      lg[i] = lg[i >> 1] + 1;
  }
  void addedge(int u, int v) { G[u].push_back(v); }
  void dfs(int u, int fa) {
    dfn[u] = dfs_clock;
    dep[u] = dep[fa] + 1;
    st[dfs_clock][0] = u;
    ++dfs_clock;
    for (int v : G[u]) {
      if (v == fa)
        continue;
      dfs(v, u);
      st[dfs_clock][0] = u;
      ++dfs_clock;
    }
  }
  void build(int rt) {
    dfs_clock = 0;
    int g = lg[m];
    for (int i = 0; i < m; ++i)
      st[i].resize(g + 1);
    dfs(rt, rt);
    for (int j = 1; j <= g; ++j) {
      for (int i = 0; i + (1 << (j - 1)) < m; ++i) {
        if (dep[st[i][j - 1]] < dep[st[i + (1 << (j - 1))][j - 1]])
          st[i][j] = st[i][j - 1];
        else
          st[i][j] = st[i + (1 << (j - 1))][j - 1];
      }
    }
  }
  int query(int u, int v) {
    int l = dfn[u], r = dfn[v];
    if (l > r)
      swap(l, r);
    int g = lg[r - l + 1];
    if (dep[st[l][g]] < dep[st[r - (1 << g) + 1][g]])
      return st[l][g];
    else
      return st[r - (1 << g) + 1][g];
  }
};

```

```cpp
template <typename T>
struct ExKruskal {
  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) const { return w < e.w; }
  };
  int n, m;
  vector<Edge> E;
  vector<T> W;
  vector<int> f;
  LCAGraph G;

  int find(int x) { return x == f[x] ? x : f[x] = find(f[x]); }

  ExKruskal(int _n = 0)
      : n(_n), W(n + n - 1), f(n + n - 1), G(n + n - 1) {
    iota(f.begin(), f.end(), 0);
  }

  void addedge(int u, int v, T w) { E.emplace_back(u, v, w); }

  void build() {
    sort(E.begin(), E.end());
    int id = n - 1, cnt = 0;
    for (auto& [u, v, w] : E) {
      u = find(u);
      v = find(v);
      if (u != v) {
        ++id;
        G.addedge(id, u);
        G.addedge(id, v);
        f[u] = f[v] = id;
        W[id] = w;
        ++cnt;
        if (cnt == n - 1)
          break;
      }
    }
    G.build(id);
  }

  int query(int u, int v) {
    int lca = G.query(u, v);
    return W[lca];
  }
};

int n, m, q;
ull a1, a2;
void solve(int Case) {
  rd(n), rd(m);
  ExKruskal<int> ek(n);

  int u, v, w;
  for (int i = 1; i <= m; ++i) {
    rd(u), rd(v), rd(w);
    --u;
    --v;
    ek.addedge(u, v, w);
  }

```

```
  ek.build();

  rd(q), rd(a1), rd(a2);

  int ans = 0;
  for (int i = 1; i <= q; ++i) {
    pair<int, int> p = myRanq(a1, a2, n);
    --p.first;
    --p.second;
    ans ^= ek.query(p.first, p.second);
  }
  printf("%d\n", ans);
}

int main() {
#ifdef BACKLIGHT
  freopen("a.in", "r", stdin);
#endif
  int T = 1;
  // cin >> T;
  for (int t = 1; t <= T; ++t)
    solve(t);
  return 0;
}
```

2.10 FullyDCP

```
// 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🄵き
  inline unsigned next(unsigned n) { return next() % n; }
};

// bottom up な Treap
//脱再🄵!
// 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🄵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) {
```

```
38    if (!l)
39      return r;
40    if (!r)
41      return l;
42
43    Ref t = NULL;
44    unsigned long long dirs = 0;
45    int h;
46    for (h = 0;; ++h) {
47      if (h >= sizeof(dirs) * 8 - 2) {
48        // dirs のオ🄵バ🄵フロ🄵を防ぐために再🄵する。
49        //あくまでセ🄵フティガ🄵ドなのでバランスは多少崩れるかもしれない
50        t = join(l->right, r->left);
51        dirs = dirs << 2 | 1;
52        h++;
53        break;
54      }
55      dirs <<= 1;
56      if (choiceRandomly(l, r)) {
57        Ref c = l->right;
58        if (!c) {
59          t = r;
60          r = r->parent;
61          break;
62        }
63        l = c;
64      } else {
65        dirs |= 1;
66        Ref c = r->left;
67        if (!c) {
68          t = l;
69          l = l->parent;
70          break;
71        }
72        r = c;
73      }
74    }
75    for (; h >= 0; --h) {
76      if (!(dirs & 1)) {
77        Ref p = l->parent;
78        t = l->linkr(t);
79        l = p;
80      } else {
81        Ref p = r->parent;
82        t = r->linkl(t);
83        r = p;
84      }
85      dirs >>= 1;
86    }
87    return t;
88  }
89
90  typedef std::pair<Ref, Ref> RefPair;
91
92  // l < t⊡r の (l,r) に分割する
93  RefPair split2(Ref t) {
94    Ref p, l = t->left, r = t;
95    Node::cut(l);
96    t->linkl(NULL);
97    while (p = t->parent) {
98      t->parent = NULL;
99      if (p->left == t)
100       r = p->linkl(r);
101     else
102       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;
      }
```

```
168       if (right) {
169         size_t += right->size;
170         markUnions_t |= right->markUnions;
171       }
172       size = size_t, markUnions = markUnions_t;
173       return this;
174     }
175
176     inline Node* linkl(Node* c) {
177       if (left = c)
178         c->parent = this;
179       return update();
180     }
181     inline Node* linkr(Node* c) {
182       if (right = c)
183         c->parent = this;
184       return update();
185     }
186     inline Node* linklr(Node* l, Node* r) {
187       if (left = l)
188         l->parent = this;
189       if (right = r)
190         r->parent = this;
191       return update();
192     }
193     static Node* cut(Node* t) {
194       if (t)
195         t->parent = NULL;
196       return t;
197     }
198
199     static const Node* findRoot(const Node* t) {
200       while (t->parent)
201         t = t->parent;
202       return t;
203     }
204     static std::pair<Node*, int> getPosition(Node* t) {
205       int k = BST::size(t->left);
206       Node* p;
207       while (p = t->parent) {
208         if (p->right == t)
209           k += BST::size(p->left) + 1;
210         t = p;
211       }
212       return std::make_pair(t, k);
213     }
214     static const Node* findHead(const Node* t) {
215       while (t->left)
216         t = t->left;
217       return t;
218     }
219     static void updatePath(Node* t) {
220       while (t) {
221         t->update();
222         t = t->parent;
223       }
224     }
225   };
226
227   typedef Node::BST BST;
228   BST bst;
229
230   std::vector<Node> nodes;
231   //各頂点に🄵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)
232   //逆に arc に🄵して🄵🄵する頂点はたかだか 1 つである
```

```
233   std::vector<int> firstArc;
234   //🅵 · 頂点に🅵する属性
235   std::vector<bool> edgeMark, vertexMark;
236
237   inline int getArcIndex(const Node* a) const { return a - &nodes[0]; }
238
239   inline int arc1(int ei) const { return ei; }
240   inline int arc2(int ei) const { return ei + (numVertices() - 1); }
241
242  public:
243   inline int numVertices() const { return firstArc.size(); }
244   inline int numEdges() const { return numVertices() - 1; }
245
246   inline bool getEdgeMark(int a) const { return a < numEdges() ? edgeMark[a] : false; }
247   inline bool getVertexMark(int v) const { return vertexMark[v]; }
248
249  private:
250   void updateMarks(int a, int v) {
251     Node* t = &nodes[a];
252     t->marks = getEdgeMark(a) << 0 | getVertexMark(v) << 1;
253     Node::updatePath(t);
254   }
255
256   // firstArc の🅵更に🅵じて更新する
257   void firstArcChanged(int v, int a, int b) {
258     if (a != -1)
259       updateMarks(a, v);
260     if (b != -1)
261       updateMarks(b, v);
262   }
263
264  public:
265   class TreeRef {
266     friend class EulerTourTreeWithMarks;
267     const Node* ref;
268
269    public:
270     TreeRef() {}
271     TreeRef(const Node* ref_)
272         : ref(ref_) {}
273     bool operator==(const TreeRef& that) const { return ref == that.ref; }
274     bool operator!=(const TreeRef& that) const { return ref != that.ref; }
275     bool isIsolatedVertex() const { return ref == NULL; }
276   };
277
278   void init(int N) {
279     int M = N - 1;
280     firstArc.assign(N, -1);
281     nodes.assign(M * 2, Node());
282     for (int i = 0; i < M * 2; i++)
283       nodes[i].priority = bst.nextRand();
284     edgeMark.assign(M, false);
285     vertexMark.assign(N, false);
286   }
287
288   TreeRef getTreeRef(int v) const {
289     int a = firstArc[v];
290     return TreeRef(a == -1 ? NULL : Node::findRoot(&nodes[a]));
291   }
292
293   bool isConnected(int v, int w) const {
294     if (v == w)
295       return true;
296     int a = firstArc[v], b = firstArc[w];
297     if (a == -1 || b == -1)
```

```
298       return false;
299     return Node::findRoot(&nodes[a]) == Node::findRoot(&nodes[b]);
300   }
301
302   static int getSize(TreeRef t) {
303     if (t.isIsolatedVertex())
304       return 1;
305     else
306       return t.ref->size / 2 + 1;
307   }
308
309   void link(int ti, int v, int w) {
310     int a1 = arc1(ti), a2 = arc2(ti);
311     // v→w が a1 にFFするようにする
312     if (v > w)
313       std::swap(a1, a2);
314
315     int va = firstArc[v], wa = firstArc[w];
316
317     Node *l, *m, *r;
318     if (va != -1) {
319       // evert。順番を入れ替えるだけ
320       std::pair<Node*, Node*> p = bst.split2(&nodes[va]);
321       m = bst.join(p.second, p.first);
322     } else {
323       // v が孤立点の場合
324       m = NULL;
325       firstArc[v] = a1;
326       firstArcChanged(v, -1, a1);
327     }
328     if (wa != -1) {
329       std::pair<Node*, Node*> p = bst.split2(&nodes[wa]);
330       l = p.first, r = p.second;
331     } else {
332       // w が孤立点の場合
333       l = r = NULL;
334       firstArc[w] = a2;
335       firstArcChanged(w, -1, a2);
336     }
337     // w→v のFを m の先頭＝ l の末尾に insert
338     m = bst.cons(&nodes[a2], m);
339     // v→w のFを m の末尾＝ r の先頭に insert
340     r = bst.cons(&nodes[a1], r);
341
342     bst.join(bst.join(l, m), r);
343   }
344
345   void cut(int ti, int v, int w) {
346     // v→w が a1 にFFするようにする
347     if (v > w)
348       std::swap(v, w);
349
350     int a1 = arc1(ti), a2 = arc2(ti);
351     std::pair<Node*, Node*> p = bst.split3(&nodes[a1]);
352     int prsize = BST::size(p.second);
353     std::pair<Node*, Node*> q = bst.split3(&nodes[a2]);
354     Node *l, *m, *r;
355     // 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
356     if (p.second == &nodes[a2] || BST::size(p.second) != prsize) {
357       l = p.first, m = q.first, r = q.second;
358     } else {
359       // 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
360       std::swap(v, w);
361       std::swap(a1, a2);
362       l = q.first, m = q.second, r = p.second;
```

```
    }

    // firstArc を必要に🅵じて書き🅵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🅵でよい
        b = getArcIndex(Node::findHead(r));
      } else {
        // v が根なら最初の🅵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🅵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🅵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🅵理すればよい
  // callback は🅵🅵するかどうかを bool で返す。最後まで列🅵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F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;
```

```
493   std::vector<int> visitedEdges;  // : [EdgeIndex | TreeEdgeIndex]
494
495   int arc1(int ei) const { return ei; }
496   int arc2(int ei) const { return numMaxEdges() + ei; }
497   int arcEdge(int i) const { return i >= numMaxEdges() ? i - numMaxEdges() : i; }
498
499   bool replace(int lv, int v, int w) {
500     Forest& forest = forests[lv];
501
502     TreeRef vRoot = forest.getTreeRef(v), wRoot = forest.getTreeRef(w);
503     assert(vRoot.isIsolatedVertex() || wRoot.isIsolatedVertex() || vRoot != wRoot);
504
505     int vSize = forest.getSize(vRoot), wSize = forest.getSize(wRoot);
506
507     int u;
508     TreeRef uRoot;
509     int uSize;
510     if (vSize <= wSize)
511       u = v, uRoot = vRoot, uSize = vSize;
512     else
513       u = w, uRoot = wRoot, uSize = wSize;
514
515     // replacement edge を探す
516     int replacementEdge = -1;
517     enumIncidentArcs(forest, uRoot, u, lv, FindReplacementEdge(uRoot, &replacementEdge));
518
519     //"Sampling heuristic"
520     //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
521     if (replacementEdge != -1 && (int)visitedEdges.size() + 1 <= numSamplings) {
522       // replacementEdge を🄵理する
523       deleteNontreeEdge(replacementEdge);
524       addTreeEdge(replacementEdge);
525       for (int i = 0; i < (int)visitedEdges.size(); i++)
526         edgeVisited[visitedEdges[i]] = false;
527       visitedEdges.clear();
528       return true;
529     }
530
531     //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
532     for (int i = 0; i < (int)visitedEdges.size(); i++) {
533       int ei = visitedEdges[i];
534       edgeVisited[ei] = false;
535
536       deleteNontreeEdge(ei);
537
538       ++edgeLevel[ei];
539
540       insertNontreeEdge(ei);
541     }
542     visitedEdges.clear();
543
544     //このレベルの T_u の🄵を列🄵する
545     forest.enumMarkedEdges(uRoot, EnumLevelTreeEdges(this));
546     //列🄵した T_u の🄵を一🄵にレベルアップさせる
547     for (int i = 0; i < (int)visitedEdges.size(); i++) {
548       int ti = visitedEdges[i];
549
550       int ei = treeEdgeMap[ti];
551       int v = arcHead[arc2(ei)], w = arcHead[arc1(ei)];
552       int lv = edgeLevel[ei];
553
554       edgeLevel[ei] = lv + 1;
555
556       forests[lv].changeEdgeMark(ti, false);
557       forests[lv + 1].changeEdgeMark(ti, true);
```

```
558
559       forests[lv + 1].link(ti, v, w);
560     }
561     visitedEdges.clear();
562
563     if (replacementEdge != -1) {
564       // T_u の🄵列🄵の前に構造が🄵わると困るので replacementEdge はこのタイミングで🄵理する
565       deleteNontreeEdge(replacementEdge);
566       addTreeEdge(replacementEdge);
567       return true;
568     } else if (lv > 0) {
569       return replace(lv - 1, v, w);
570     } else {
571       return false;
572     }
573   }
574
575   struct EnumLevelTreeEdges {
576     This* thisp;
577     EnumLevelTreeEdges(This* thisp_)
578         : thisp(thisp_) {}
579
580     inline bool operator()(int a) {
581       thisp->enumLevelTreeEdges(a);
582       return true;
583     }
584   };
585   void enumLevelTreeEdges(int ti) { visitedEdges.push_back(ti); }
586
587   //孤立点の時特🄵な🄵理をするなどしなければいけないのでヘルパ🄵
588   template <typename Callback>
589   bool enumIncidentArcs(Forest& forest, TreeRef t, int u, int lv, Callback callback) {
590     if (t.isIsolatedVertex())
591       return enumIncidentArcsWithVertex<Callback>(lv, u, callback);
592     else
593       return forest.enumMarkedVertices(t, EnumIncidentArcs<Callback>(this, lv, callback));
594   }
595
596   template <typename Callback>
597   struct EnumIncidentArcs {
598     This* thisp;
599     int lv;
600     Callback callback;
601
602     EnumIncidentArcs(This* thisp_, int lv_, Callback callback_)
603         : thisp(thisp_), lv(lv_), callback(callback_) {}
604
605     inline bool operator()(int tii) const {
606       return thisp->enumIncidentArcsWithTreeArc(tii, lv, callback);
607     }
608   };
609
610   template <typename Callback>
611   bool enumIncidentArcsWithTreeArc(int tii, int lv, Callback callback) {
612     bool dir = tii >= numVertices() - 1;
613     int ti = dir ? tii - (numVertices() - 1) : tii;
614     int ei = treeEdgeMap[ti];
615     int v = arcHead[arc2(ei)], w = arcHead[arc1(ei)];
616     //方向を求め、その arc の tail の頂点を取得する
617     int u = !(dir != (v > w)) ? v : w;
618
619     return enumIncidentArcsWithVertex(lv, u, callback);
620   }
621
622   // 1 つの頂点を🄵理する
```

```
623  template <typename Callback>
624  bool enumIncidentArcsWithVertex(int lv, int u, Callback callback) {
625    int it = firstIncidentArc[lv][u];
626    while (it != -1) {
627      if (!callback(this, it))
628        return false;
629      it = nextIncidentArc[it];
630    }
631    return true;
632  }
633
634  struct FindReplacementEdge {
635    TreeRef uRoot;
636    int* replacementEdge;
637    FindReplacementEdge(TreeRef uRoot_, int* replacementEdge_)
638        : uRoot(uRoot_), replacementEdge(replacementEdge_) {}
639
640    inline bool operator()(This* thisp, int a) const {
641      return thisp->findReplacementEdge(a, uRoot, replacementEdge);
642    }
643  };
644
645  // 1 つの arc を🄵理する
646  bool findReplacementEdge(int a, TreeRef uRoot, int* replacementEdge) {
647    int ei = arcEdge(a);
648    if (edgeVisited[ei])
649      return true;
650
651    int lv = edgeLevel[ei];
652    TreeRef hRoot = forests[lv].getTreeRef(arcHead[a]);
653
654    if (hRoot.isIsolatedVertex() || hRoot != uRoot) {
655      //🄵の木に渡されているなら replacement edge である。
656      *replacementEdge = ei;
657      return false;
658    }
659    // replacement edge は visitedEdges に入れたくないのでこの位置でマ🄵クする
660    edgeVisited[ei] = true;
661    visitedEdges.push_back(ei);
662    return true;
663  }
664
665  void addTreeEdge(int ei) {
666    int v = arcHead[arc2(ei)], w = arcHead[arc1(ei)];
667    int lv = edgeLevel[ei];
668
669    int ti = treeEdgeIndexFreeList.back();
670    treeEdgeIndexFreeList.pop_back();
671    treeEdgeIndex[ei] = ti;
672    treeEdgeMap[ti] = ei;
673
674    forests[lv].changeEdgeMark(ti, true);
675
676    for (int i = 0; i <= lv; i++)
677      forests[i].link(ti, v, w);
678  }
679
680  void insertIncidentArc(int a, int v) {
681    int ei = arcEdge(a);
682    int lv = edgeLevel[ei];
683    assert(treeEdgeIndex[ei] == -1);
684
685    int next = firstIncidentArc[lv][v];
686    firstIncidentArc[lv][v] = a;
687    nextIncidentArc[a] = next;
```

```
    prevIncidentArc[a] = -1;
    if (next != -1)
      prevIncidentArc[next] = a;

    if (next == -1)
      forests[lv].changeVertexMark(v, true);
  }

  void deleteIncidentArc(int a, int v) {
    int ei = arcEdge(a);
    int lv = edgeLevel[ei];
    assert(treeEdgeIndex[ei] == -1);

    int next = nextIncidentArc[a], prev = prevIncidentArc[a];
    nextIncidentArc[a] = prevIncidentArc[a] = -2;

    if (next != -1)
      prevIncidentArc[next] = prev;
    if (prev != -1)
      nextIncidentArc[prev] = next;
    else
      firstIncidentArc[lv][v] = next;

    if (next == -1 && prev == -1)
      forests[lv].changeVertexMark(v, false);
  }

  void insertNontreeEdge(int ei) {
    int a1 = arc1(ei), a2 = arc2(ei);
    insertIncidentArc(a1, arcHead[a2]);
    insertIncidentArc(a2, arcHead[a1]);
  }

  void deleteNontreeEdge(int ei) {
    int a1 = arc1(ei), a2 = arc2(ei);
    deleteIncidentArc(a1, arcHead[a2]);
    deleteIncidentArc(a2, arcHead[a1]);
  }

public:
  HolmDeLichtenbergThorup()
      : numVertices_m(0), numSamplings(0) {}

  int numVertices() const { return numVertices_m; }
  int numMaxEdges() const { return edgeLevel.size(); }

  void init(int N, int M) {
    numVertices_m = N;

    int levels = 1;
    while (1 << levels <= N / 2)
      levels++;

    //サンプリング数を設定する。適切なFはよくわからない
    numSamplings = (int)(levels * 1);

    forests.resize(levels);
    for (int lv = 0; lv < levels; lv++)
      forests[lv].init(N);

    edgeLevel.assign(M, -1);

    treeEdgeIndex.assign(M, -1);
    treeEdgeMap.assign(N - 1, -1);

```

```
753     treeEdgeIndexFreeList.resize(N - 1);
754     for (int ti = 0; ti < N - 1; ti++)
755       treeEdgeIndexFreeList[ti] = ti;
756
757     arcHead.assign(M * 2, -1);
758
759     firstIncidentArc.resize(levels);
760     for (int lv = 0; lv < levels; lv++)
761       firstIncidentArc[lv].assign(N, -1);
762     nextIncidentArc.assign(M * 2, -2);
763     prevIncidentArc.assign(M * 2, -2);
764
765     edgeVisited.assign(M, false);
766   }
767
768   bool insertEdge(int ei, int v, int w) {
769     if (!(0 <= ei && ei < numMaxEdges() && 0 <= v && v < numVertices() && 0 <= w && w < numVertices())) {
770       system("pause");
771     }
772     assert(0 <= ei && ei < numMaxEdges() && 0 <= v && v < numVertices() && 0 <= w && w < numVertices());
773     assert(edgeLevel[ei] == -1);
774
775     int a1 = arc1(ei), a2 = arc2(ei);
776     arcHead[a1] = w, arcHead[a2] = v;
777
778     bool treeEdge = !forests[0].isConnected(v, w);
779
780     edgeLevel[ei] = 0;
781     if (treeEdge) {
782       addTreeEdge(ei);
783     } else {
784       treeEdgeIndex[ei] = -1;
785       //ループは見たくないのでリストにも入れない
786       if (v != w)
787         insertNontreeEdge(ei);
788     }
789
790     return treeEdge;
791   }
792
793   bool deleteEdge(int ei) {
794     assert(0 <= ei && ei < numMaxEdges() && edgeLevel[ei] != -1);
795
796     int a1 = arc1(ei), a2 = arc2(ei);
797     int v = arcHead[a2], w = arcHead[a1];
798
799     int lv = edgeLevel[ei];
800     int ti = treeEdgeIndex[ei];
801
802     bool splitted = false;
803     if (ti != -1) {
804       treeEdgeMap[ti] = -1;
805       treeEdgeIndex[ei] = -1;
806       treeEdgeIndexFreeList.push_back(ti);
807
808       for (int i = 0; i <= lv; i++)
809         forests[i].cut(ti, v, w);
810
811       forests[lv].changeEdgeMark(ti, false);
812
813       splitted = !replace(lv, v, w);
814     } else {
815       //ループはリストに入ってない
816       if (v != w)
817         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;
}
```

2.11 Graph

```
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);
  }
};

}  // namespace Backlight
```

2.12 GraphMatch

```
#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) {
```

```cpp
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 << " ";
  }
}
```

2.13 HLD-Edge

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

const int N = 2e5 + 5;

int n, q;

struct edge {
  int v, w, nxt;
} e[N << 1];
int tot, head[N];
void init_graph(int n) {
  tot = 0;
  fill(head + 1, head + 1 + n, 0);
}
void add(int u, int v, int w) {
  ++tot;
  e[tot] = (edge){v, w, head[u]};
  head[u] = tot;
}

int sz[N], son[N], h[N], f[N], w[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;
    w[v] = e[i].w;
    dfs1(v, u);
    sz[u] += sz[v];
    if (sz[v] > sz[son[u]])
      son[u] = v;
  }
}
int dfs_clock, dfn[N], rk[N], top[N];
void dfs2(int u, int fa, int tp) {
  ++dfs_clock;
  dfn[dfs_clock] = w[u];
  rk[u] = dfs_clock;
  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);
  }
}

#define mid ((l + r) >> 1)
#define lc (x << 1)
#define rc (x << 1 | 1)
#define lson lc, l, mid
#define rson rc, mid + 1, r
int sum[N << 2], ma[N << 2], mi[N << 2], tag_inv[N << 2];
void push_up(int x) {
  sum[x] = sum[lc] + sum[rc];
  ma[x] = max(ma[lc], ma[rc]);
  mi[x] = min(mi[lc], mi[rc]);
}
void push_down(int x) {
  if (tag_inv[x] != 1) {
    sum[lc] = -sum[lc];
    swap(ma[lc], mi[lc]);
    ma[lc] = -ma[lc];
    mi[lc] = -mi[lc];
    tag_inv[lc] = -tag_inv[lc];

    sum[rc] = -sum[rc];
    swap(ma[rc], mi[rc]);
    ma[rc] = -ma[rc];
    mi[rc] = -mi[rc];
    tag_inv[rc] = -tag_inv[rc];

    tag_inv[x] = 1;
  }
}
void build(int x, int l, int r) {
  tag_inv[x] = 1;
  if (l == r) {
    sum[x] = ma[x] = mi[x] = dfn[l];
    return;
  }
  build(lson);
  build(rson);
  push_up(x);
}

void update(int x, int l, int r, int p, int w) {
  if (l == r) {
    sum[x] = ma[x] = mi[x] = w;
    return;
  }
  push_down(x);
  if (p <= mid)
    update(lson, p, w);
  else
    update(rson, p, w);
  push_up(x);
}

void inverse(int x, int l, int r, int L, int R) {
  if (l == L && r == R) {
    sum[x] = -sum[x];
    swap(ma[x], mi[x]);
    ma[x] = -ma[x];
```

```
    mi[x] = -mi[x];
    tag_inv[x] = -tag_inv[x];
    return;
  }
  push_down(x);
  if (R <= mid)
    inverse(lson, L, R);
  else if (L > mid)
    inverse(rson, L, R);
  else {
    inverse(lson, L, mid);
    inverse(rson, mid + 1, R);
  }
  push_up(x);
}

int getsum(int x, int l, int r, int L, int R) {
  if (l == L && r == R)
    return sum[x];
  push_down(x);
  if (R <= mid)
    return getsum(lson, L, R);
  else if (L > mid)
    return getsum(rson, L, R);
  return getsum(lson, L, mid) + getsum(rson, mid + 1, R);
}

int getmax(int x, int l, int r, int L, int R) {
  if (l == L && r == R)
    return ma[x];
  push_down(x);
  if (R <= mid)
    return getmax(lson, L, R);
  else if (L > mid)
    return getmax(rson, L, R);
  return max(getmax(lson, L, mid), getmax(rson, mid + 1, R));
}

int getmin(int x, int l, int r, int L, int R) {
  if (l == L && r == R)
    return mi[x];
  push_down(x);
  if (R <= mid)
    return getmin(lson, L, R);
  else if (L > mid)
    return getmin(rson, L, R);
  return min(getmin(lson, L, mid), getmin(rson, mid + 1, R));
}

void INVERSE(int u, int v) {
  while (top[u] != top[v]) {
    if (h[top[u]] < h[top[v]])
      swap(u, v);
    inverse(1, 1, n, rk[top[u]], rk[u]);
    u = f[top[u]];
  }
  if (h[u] != h[v]) {
    if (h[u] > h[v])
      swap(u, v);
    inverse(1, 1, n, rk[son[u]], rk[v]);
  }
}

int QSUM(int u, int v) {
  int res = 0;
```

```
  while (top[u] != top[v]) {
    if (h[top[u]] < h[top[v]])
      swap(u, v);
    res += getsum(1, 1, n, rk[top[u]], rk[u]);
    u = f[top[u]];
  }
  if (h[u] != h[v]) {
    if (h[u] > h[v])
      swap(u, v);
    res += getsum(1, 1, n, rk[son[u]], rk[v]);
  }
  return res;
}

int QMAX(int u, int v) {
  int res = INT_MIN;
  while (top[u] != top[v]) {
    if (h[top[u]] < h[top[v]])
      swap(u, v);
    res = max(res, getmax(1, 1, n, rk[top[u]], rk[u]));
    u = f[top[u]];
  }
  if (h[u] != h[v]) {
    if (h[u] > h[v])
      swap(u, v);
    res = max(res, getmax(1, 1, n, rk[son[u]], rk[v]));
  }
  return res;
}

int QMIN(int u, int v) {
  int res = INT_MAX;
  while (top[u] != top[v]) {
    if (h[top[u]] < h[top[v]])
      swap(u, v);
    res = min(res, getmin(1, 1, n, rk[top[u]], rk[u]));
    u = f[top[u]];
  }
  if (h[u] != h[v]) {
    if (h[u] > h[v])
      swap(u, v);
    res = min(res, getmin(1, 1, n, rk[son[u]], rk[v]));
  }
  return res;
}

int tu[N], tv[N];
void solve(int Case) {
  /* write code here */
  /* gl & hf */
  scanf("%d", &n);
  int u, v, w;
  for (int i = 1; i <= n - 1; ++i) {
    scanf("%d %d %d", &u, &v, &w);
    ++u, ++v;
    add(u, v, w);
    add(v, u, w);

    tu[i] = u;
    tv[i] = v;
  }

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

```

```
  build(1, 1, n);

  scanf("%d", &q);
  char op[5];
  int x, y;
  for (int i = 1; i <= q; ++i) {
    scanf("%s %d %d", op, &x, &y);
    ++x, ++y;
    if (op[0] == 'C') {
      --x, --y;
      int id = h[tu[x]] > h[tv[x]] ? tu[x] : tv[x];
      update(1, 1, n, rk[id], y);
    } else if (op[0] == 'N') {
      INVERSE(x, y);
    } else if (op[0] == 'S') {
      printf("%d\n", QSUM(x, y));
    } else if (op[1] == 'A') {
      printf("%d\n", QMAX(x, y));
    } else if (op[1] == 'I') {
      printf("%d\n", QMIN(x, y));
    }
  }
}

int main() {
  int T = 1;
  for (int _ = 1; _ <= T; _++)
    solve(_);
  return 0;
}
```

2.14 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]);
    }
  }
}
```

2.15 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;
  }
};

}  // namespace Backlight
```

2.16 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;
}
```

2.17 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];
  }
};

};  // namespace Backlight
```

2.18 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);

    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
```

2.19 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
```

2.20 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);
    }
  }
};

}  // namespace Backlight
```

2.21 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;
  }
};

}  // namespace Backlight
```

2.22 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");
  }
}
```

2.23 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));
  }

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

}  // namespace Backlight
```

2.24 WraphMatch

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

template <typename CostType, typename TotalCostType = int64_t>
class MaximumWeightedMatching {
  /*
    Maximum Weighted Matching in General Graphs.
    - O(nm log(n)) time
    - O(n + m) space

    Note: each vertex is 1-indexed.
  */
 public:
  using cost_t = CostType;
  using tcost_t = TotalCostType;

 private:
  enum Label { kSeparated = -2,
               kInner = -1,
               kFree = 0,
               kOuter = 1 };
  static constexpr cost_t Inf = cost_t(1) << (sizeof(cost_t) * 8 - 2);

 private:
  template <typename T>
  class BinaryHeap {
   public:
    struct Node {
      bool operator<(const Node& rhs) const { return value < rhs.value; }
      T value;
      int id;
    };
    BinaryHeap() {}
    BinaryHeap(int N)
        : size_(0), node(N + 1), index(N, 0) {
    }
    int size() const { return size_; }
    bool empty() const { return size_ == 0; }
    void clear() {
      while (size_ > 0)
        index[node[size_--].id] = 0;
    }
    T min() const { return node[1].value; }
```

```
45    int argmin() const { return node[1].id; }  // argmin ?
46    T get_val(int id) const { return node[index[id]].value; }
47    void pop() {
48      if (size_ > 0)
49        pop(1);
50    }
51    void erase(int id) {
52      if (index[id])
53        pop(index[id]);
54    }
55    bool has(int id) const { return index[id] != 0; }
56    void update(int id, T v) {
57      if (!has(id))
58        return push(id, v);
59      bool up = (v < node[index[id]].value);
60      node[index[id]].value = v;
61      if (up)
62        up_heap(index[id]);
63      else
64        down_heap(index[id]);
65    }
66    void decrease_key(int id, T v) {
67      if (!has(id))
68        return push(id, v);
69      if (v < node[index[id]].value)
70        node[index[id]].value = v, up_heap(index[id]);
71    }
72    void push(int id, T v) {
73      // assert(!has(id));
74      index[id] = ++size_;
75      node[size_] = {v, id};
76      up_heap(size_);
77    }
78
79   private:
80    void pop(int pos) {
81      index[node[pos].id] = 0;
82      if (pos == size_) {
83        --size_;
84        return;
85      }
86      bool up = (node[size_].value < node[pos].value);
87      node[pos] = node[size_--];
88      index[node[pos].id] = pos;
89      if (up)
90        up_heap(pos);
91      else
92        down_heap(pos);
93    }
94    void swap_node(int a, int b) {
95      swap(node[a], node[b]);
96      index[node[a].id] = a;
97      index[node[b].id] = b;
98    }
99    void down_heap(int pos) {
100     for (int k = pos, nk = k; 2 * k <= size_; k = nk) {
101       if (node[2 * k] < node[nk])
102         nk = 2 * k;
103       if (2 * k + 1 <= size_ && node[2 * k + 1] < node[nk])
104         nk = 2 * k + 1;
105       if (nk == k)
106         break;
107       swap_node(k, nk);
108     }
109   }
```

```
110     void up_heap(int pos) {
111       for (int k = pos; k > 1 && node[k] < node[k >> 1]; k >>= 1)
112         swap_node(k, k >> 1);
113     }
114     int size_;
115     vector<Node> node;
116     vector<int> index;
117   };
118
119   template <typename Key>
120   class PairingHeaps {
121    private:
122     struct Node {
123       Node()
124           : prev(-1) {
125       }  // "prev < 0" means the node is unused.
126       Node(Key v)
127           : key(v), child(0), next(0), prev(0) {
128       }
129       Key key;
130       int child, next, prev;
131     };
132
133    public:
134     PairingHeaps(int H, int N)
135         : heap(H), node(N) {
136       // It consists of `H` Pairing heaps.
137       // Each heap-node ID can appear at most 1 time(s) among heaps
138       // and should be in [1, N).
139     }
140
141     void clear(int h) {
142       if (heap[h])
143         clear_rec(heap[h]), heap[h] = 0;
144     }
145     void clear_all() {
146       for (size_t i = 0; i < heap.size(); ++i)
147         heap[i] = 0;
148       for (size_t i = 0; i < node.size(); ++i)
149         node[i] = Node();
150     }
151     bool empty(int h) const { return !heap[h]; }
152     bool used(int v) const { return node[v].prev >= 0; }
153     Key min(int h) const { return node[heap[h]].key; }
154     int argmin(int h) const { return heap[h]; }
155
156     void pop(int h) {
157       // assert(!empty(h));
158       erase(h, heap[h]);
159     }
160     void push(int h, int v, Key key) {
161       // assert(!used(v));
162       node[v] = Node(key);
163       heap[h] = merge(heap[h], v);
164     }
165     void erase(int h, int v) {
166       if (!used(v))
167         return;
168       int w = two_pass_pairing(node[v].child);
169       if (!node[v].prev)
170         heap[h] = w;
171       else {
172         cut(v);
173         heap[h] = merge(heap[h], w);
174       }
```

```
    node[v].prev = -1;
  }
  void decrease_key(int h, int v, Key key) {
    if (!used(v))
      return push(h, v, key);
    if (!node[v].prev)
      node[v].key = key;
    else {
      cut(v);
      node[v].key = key;
      heap[h] = merge(heap[h], v);
    }
  }

 private:
  void clear_rec(int v) {
    for (; v; v = node[v].next) {
      if (node[v].child)
        clear_rec(node[v].child);
      node[v].prev = -1;
    }
  }

  inline void cut(int v) {
    auto& n = node[v];
    int pv = n.prev, nv = n.next;
    auto& pn = node[pv];
    if (pn.child == v)
      pn.child = nv;
    else
      pn.next = nv;
    node[nv].prev = pv;
    n.next = n.prev = 0;
  }

  int merge(int l, int r) {
    if (!l)
      return r;
    if (!r)
      return l;
    if (node[l].key > node[r].key)
      swap(l, r);
    int lc = node[r].next = node[l].child;
    node[l].child = node[lc].prev = r;
    return node[r].prev = l;
  }

  int two_pass_pairing(int root) {
    if (!root)
      return 0;
    int a = root;
    root = 0;
    while (a) {
      int b = node[a].next, na = 0;
      node[a].prev = node[a].next = 0;
      if (b)
        na = node[b].next, node[b].prev = node[b].next = 0;
      a = merge(a, b);
      node[a].next = root;
      root = a;
      a = na;
    }
    int s = node[root].next;
    node[root].next = 0;
    while (s) {
```

```
240         int t = node[s].next;
241         node[s].next = 0;
242         root = merge(root, s);
243         s = t;
244       }
245       return root;
246     }
247
248    private:
249     vector<int> heap;
250     vector<Node> node;
251   };
252
253   template <typename T>
254   struct PriorityQueue : public priority_queue<T, vector<T>, greater<T>> {
255     PriorityQueue() {}
256     PriorityQueue(int N) { this->c.reserve(N); }
257     T min() { return this->top(); }
258     void clear() { this->c.clear(); }
259   };
260
261   template <typename T>
262   struct Queue {
263     Queue() {}
264     Queue(int N)
265         : qh(0), qt(0), data(N) {
266     }
267     T operator[](int i) const { return data[i]; }
268     void enqueue(int u) { data[qt++] = u; }
269     int dequeue() { return data[qh++]; }
270     bool empty() const { return qh == qt; }
271     void clear() { qh = qt = 0; }
272     int size() const { return qt; }
273     int qh, qt;
274     vector<T> data;
275   };
276
277  public:
278   struct InputEdge {
279     int from, to;
280     cost_t cost;
281   };
282
283  private:
284   template <typename T>
285   using ModifiableHeap = BinaryHeap<T>;
286   template <typename T>
287   using ModifiableHeaps = PairingHeaps<T>;
288   template <typename T>
289   using FastHeap = PriorityQueue<T>;
290
291   struct Edge {
292     int to;
293     cost_t cost;
294   };
295   struct Link {
296     int from, to;
297   };
298   struct Node {
299     struct NodeLink {
300       int b, v;
301     };
302     Node() {}
303     Node(int u)
304         : parent(0), size(1) {
```

```
305       link[0] = link[1] = {u, u};
306     }
307     int next_v() const { return link[0].v; }
308     int next_b() const { return link[0].b; }
309     int prev_v() const { return link[1].v; }
310     int prev_b() const { return link[1].b; }
311     int parent, size;
312     NodeLink link[2];
313   };
314   struct Event {
315     Event() {}
316     Event(cost_t time, int id)
317         : time(time), id(id) {
318     }
319     bool operator<(const Event& rhs) const { return time < rhs.time; }
320     bool operator>(const Event& rhs) const { return time > rhs.time; }
321     cost_t time;
322     int id;
323   };
324   struct EdgeEvent {
325     EdgeEvent() {}
326     EdgeEvent(cost_t time, int from, int to)
327         : time(time), from(from), to(to) {
328     }
329     bool operator>(const EdgeEvent& rhs) const { return time > rhs.time; }
330     bool operator<(const EdgeEvent& rhs) const { return time < rhs.time; }
331     cost_t time;
332     int from, to;
333   };
334
335  public:
336   MaximumWeightedMatching(int N, const vector<InputEdge>& in)
337       : 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(
338     for (auto& e : in)
339       ofs[e.from + 1]++, ofs[e.to + 1]++;
340     for (int i = 1; i <= N + 1; ++i)
341       ofs[i] += ofs[i - 1];
342     for (auto& e : in) {
343       edges[ofs[e.from]++] = {e.to, e.cost * 2};
344       edges[ofs[e.to]++] = {e.from, e.cost * 2};
345     }
346     for (int i = N + 1; i > 0; --i)
347       ofs[i] = ofs[i - 1];
348     ofs[0] = 0;
349   }
350
351   pair<tcost_t, vector<int>> maximum_weighted_matching(bool init_matching = false) {
352     initialize();
353     set_potential();
354     if (init_matching)
355       find_maximal_matching();
356     for (int u = 1; u <= N; ++u)
357       if (!mate[u])
358         do_edmonds_search(u);
359     tcost_t ret = compute_optimal_value();
360     return make_pair(ret, mate);
361   }
362
363  private:
364   tcost_t compute_optimal_value() const {
365     tcost_t ret = 0;
366     for (int u = 1; u <= N; ++u)
367       if (mate[u] > u) {
368         cost_t max_c = 0;
369         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);
  } else {
    int x = link[v].from, y = link[v].to;
    rematch(x, y);
    rematch(y, x);
  }
}

void fix_mate_and_base(int b) {
  if (b <= N)
    return;
  int bv = base[b], mv = node[bv].link[0].v, bmv = node[bv].link[0].b;
  int d = (node[bmv].link[1].v == mate[mv]) ? 0 : 1;
  while (1) {
    int mv = node[bv].link[d].v, bmv = node[bv].link[d].b;
    if (node[bmv].link[1 ^ d].v != mate[mv])
      break;
    fix_mate_and_base(bv);
    fix_mate_and_base(bmv);
    bv = node[bmv].link[d].b;
  }
  fix_mate_and_base(base[b] = bv);
  mate[b] = mate[bv];
}

void reset_time() {
  time_current_ = 0;
  event1 = {Inf, 0};
}

void reset_blossom(int b) {
  label[b] = kFree;
  link[b].from = 0;
  slack[b] = Inf;
  lazy[b] = 0;
}

void reset_all() {
  label[0] = kFree;
  link[0].from = 0;
  for (int v = 1; v <= N; ++v) {  // should be optimized for sparse graphs.
    if (label[v] == kOuter)
      potential[v] -= time_current_;
    else {
      int bv = surface[v];
      potential[v] += lazy[bv];
```

```
435         if (label[bv] == kInner)
436           potential[v] += time_current_ - time_created[bv];
437       }
438       reset_blossom(v);
439     }
440     for (int b = N + 1, r = B - unused_bid_idx_; r > 0 && b < S; ++b)
441       if (base[b] != b) {
442         if (surface[b] == b) {
443           fix_mate_and_base(b);
444           if (label[b] == kOuter)
445             potential[b] += (time_current_ - time_created[b]) << 1;
446           else if (label[b] == kInner)
447             fix_blossom_potential<kInner>(b);
448           else
449             fix_blossom_potential<kFree>(b);
450         }
451         heap2s.clear(b);
452         reset_blossom(b);
453         --r;
454       }
455
456     que.clear();
457     reset_time();
458     heap2.clear();
459     heap3.clear();
460     heap4.clear();
461   }
462
463   void do_edmonds_search(int root) {
464     if (potential[root] == 0)
465       return;
466     link_blossom(surface[root], {0, 0});
467     push_outer_and_fix_potentials(surface[root], 0);
468     for (bool augmented = false; !augmented;) {
469       augmented = augment(root);
470       if (augmented)
471         break;
472       augmented = adjust_dual_variables(root);
473     }
474     reset_all();
475   }
476
477   template <Label Lab>
478   inline cost_t fix_blossom_potential(int b) {
479     // Return the amount.
480     // (If v is an atom, the potential[v] will not be changed.)
481     cost_t d = lazy[b];
482     lazy[b] = 0;
483     if (Lab == kInner) {
484       cost_t dt = time_current_ - time_created[b];
485       if (b > N)
486         potential[b] -= dt << 1;
487       d += dt;
488     }
489     return d;
490   }
491
492   template <Label Lab>
493   inline void update_heap2(int x, int y, int by, cost_t t) {
494     if (t >= slack[y])
495       return;
496     slack[y] = t;
497     best_from[y] = x;
498     if (y == by) {
499       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
```

```
630   }
631
632   void link_blossom(int v, Link l) {
633     link[v] = {l.from, l.to};
634     if (v <= N)
635       return;
636     int b = base[v];
637     link_blossom(b, l);
638     int pb = node[b].prev_b();
639     l = {node[pb].next_v(), node[b].prev_v()};
640     for (int bv = b;;) {
641       int bw = node[bv].next_b();
642       if (bw == b)
643         break;
644       link_blossom(bw, l);
645       Link nl = {node[bw].prev_v(), node[bv].next_v()};
646       bv = node[bw].next_b();
647       link_blossom(bv, nl);
648     }
649   }
650
651   void push_outer_and_fix_potentials(int v, cost_t d) {
652     label[v] = kOuter;
653     if (v > N) {
654       for (int b = base[v]; label[b] != kOuter; b = node[b].next_b()) {
655         push_outer_and_fix_potentials(b, d);
656       }
657     } else {
658       potential[v] += time_current_ + d;
659       if (potential[v] < event1.time)
660         event1 = {potential[v], v};
661       que.enqueue(v);
662     }
663   }
664
665   bool grow(int root, int x, int y) {
666     int by = surface[y];
667     bool visited = (label[by] != kFree);
668     if (!visited)
669       link_blossom(by, {0, 0});
670     label[by] = kInner;
671     time_created[by] = time_current_;
672     heap2.erase(by);
673     if (y != by)
674       heap4.update(by, time_current_ + (potential[by] >> 1));
675     int z = mate[by];
676     if (z == 0 && by != surface[root]) {
677       rematch(x, y);
678       rematch(y, x);
679       return true;
680     }
681     int bz = surface[z];
682     if (!visited)
683       link_blossom(bz, {x, y});
684     else
685       link[bz] = link[z] = {x, y};
686     push_outer_and_fix_potentials(bz, fix_blossom_potential<kFree>(bz));
687     time_created[bz] = time_current_;
688     heap2.erase(bz);
689     return false;
690   }
691
692   void free_blossom(int bid) {
693     unused_bid[unused_bid_idx_++] = bid;
694     base[bid] = bid;
```

```
695   }
696
697   int recalculate_minimum_slack(int b, int g) {
698     // Return the destination of the best edge of blossom `g`.
699     if (b <= N) {
700       if (slack[b] >= slack[g])
701         return 0;
702       slack[g] = slack[b];
703       best_from[g] = best_from[b];
704       return b;
705     }
706     int v = 0;
707     for (int beta = base[b], bb = beta;;) {
708       int w = recalculate_minimum_slack(bb, g);
709       if (w != 0)
710         v = w;
711       if ((bb = node[bb].next_b()) == beta)
712         break;
713     }
714     return v;
715   }
716
717   void construct_smaller_components(int b, int sf, int g) {
718     surface[b] = sf, group[b] = g;  // `group[b] = g` is unneeded.
719     if (b <= N)
720       return;
721     for (int bb = base[b]; surface[bb] != sf; bb = node[bb].next_b()) {
722       if (bb == heavy[b]) {
723         construct_smaller_components(bb, sf, g);
724       } else {
725         set_surface_and_group(bb, sf, bb);
726         int to = 0;
727         if (bb > N)
728           slack[bb] = Inf, to = recalculate_minimum_slack(bb, bb);
729         else if (slack[bb] < Inf)
730           to = bb;
731         if (to > 0)
732           heap2s.push(sf, bb, EdgeEvent(slack[bb], best_from[bb], to));
733       }
734     }
735   }
736
737   void move_to_largest_blossom(int bid) {
738     const int h = heavy[bid];
739     cost_t d = (time_current_ - time_created[bid]) + lazy[bid];
740     lazy[bid] = 0;
741     for (int beta = base[bid], b = beta;;) {
742       time_created[b] = time_current_;
743       lazy[b] = d;
744       if (b != h)
745         construct_smaller_components(b, b, b), heap2s.erase(bid, b);
746       if ((b = node[b].next_b()) == beta)
747         break;
748     }
749     if (h > 0)
750       swap_blossom(h, bid), bid = h;
751     free_blossom(bid);
752   }
753
754   void expand(int bid) {
755     int mv = mate[base[bid]];
756     move_to_largest_blossom(bid);  // O(n log n) time / Edmonds search
757     Link old_link = link[mv];
758     int old_base = surface[mate[mv]], root = surface[old_link.to];
759     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;
 }

private:
```

```
  void initialize() {
    que = Queue<int>(N);
    mate.assign(S, 0);
    link.assign(S, {0, 0});
    label.assign(S, kFree);
    base.resize(S);
    for (int u = 1; u < S; ++u)
      base[u] = u;
    surface.resize(S);
    for (int u = 1; u < S; ++u)
      surface[u] = u;

    potential.resize(S);
    node.resize(S);
    for (int b = 1; b < S; ++b)
      node[b] = Node(b);

    unused_bid.resize(B);
    for (int i = 0; i < B; ++i)
      unused_bid[i] = N + B - i;
    unused_bid_idx_ = B;

    // for O(nm log n) implementation
    reset_time();
    time_created.resize(S);
    slack.resize(S);
    for (int i = 0; i < S; ++i)
      slack[i] = Inf;
    best_from.assign(S, 0);
    heavy.assign(S, 0);
    lazy.assign(S, 0);
    group.resize(S);
    for (int i = 0; i < S; ++i)
      group[i] = i;
  }

  void set_potential() {
    for (int u = 1; u <= N; ++u) {
      cost_t max_c = 0;
      for (int eid = ofs[u]; eid < ofs[u + 1]; ++eid) {
        max_c = max(max_c, edges[eid].cost);
      }
      potential[u] = max_c >> 1;
    }
  }

  void find_maximal_matching() {
    // Find a maximal matching naively.
    for (int u = 1; u <= N; ++u)
      if (!mate[u]) {
        for (int eid = ofs[u]; eid < ofs[u + 1]; ++eid) {
          auto& e = edges[eid];
          int v = e.to;
          if (mate[v] > 0 || reduced_cost(u, v, e) > 0)
            continue;
          mate[u] = v;
          mate[v] = u;
          break;
        }
      }
  }

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

```cpp
  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;
  }
```

```cpp

  // 绕原点逆时针旋转，给出旋转弧度
  void transxy(const double& b) {
    double tx = x, ty = y;
    x = tx * cos(b) - ty * sin(b);
    y = tx * sin(b) + ty * cos(b);
  }

  // 逆时针旋转 90 度
  point trans90() {
    return point(-y, x);
  }

  // 顺时针旋转 90 度
  point trans270() {
    return point(y, -x);
  }

  // 与原点的距离
  // a,b 之间的距离: (b- a).length()
  double length() {
    return sqrt(x * x + y * y);
  }

  // 与原点的距离的平方
  double length2() {
    return x * x + y * y;
  }

  // 与点 a 之间的距离
  double disTo(const point& a) {
    return (a - *this).length();
  }

  // 与 x 轴正方向的夹角，单位为弧度
  double alpha() {
    return atan2(y, x);
  }

  // 单位向量
  point unit() {
    return point(x, y) / length();
  }
};

// 向量 Oa 和向量 Ob 的叉积
inline double det(const point& a, const point& b) {
  return a.x * b.y - a.y * b.x;
}

// 向量 ab 和向量 ac 的叉积
inline double det(const point& a, const point& b, const point& c) {
  return det(b - a, c - a);
}

// 向量 Oa 和向量 Ob 的点积
inline double dot(const point& a, const point& b) {
  return a.x * b.x + a.y * b.y;
}

// 向量 ab 和向量 ac 的点积
inline double dot(const point& a, const point& b, const point& c) {
  return dot(b - a, c - a);
}

```

```cpp
// 两点间距离
inline double distance(const point& a, const point& b) {
  return (a - b).length();
}

// 两点间距离的平方
inline double distance2(const point& a, const point& b) {
  return (b.x - a.x) * (b.x - a.x) + (b.y - a.y) * (b.y - a.y);
}

// LightOJ1203
// 最终答案会在凸包上，然后算 ab 与 ac 的夹角，单位为弧度
// ab 与 ac 的夹角
double radian(point a, point b, point c) {
  return fabs(atan2(fabs(det(a, b, c)), dot(a, b, c)));
}

double angle(point a, point b, point c) {
  double r = radian(a, b, c);
  return radian2degree(r);
}

// 从点 a，由 b 遮挡，能否看见 c
bool canSee(point a, point b, point c) {
  return sgn(det(a, b, c)) <= 0;
}

// 直线或者线段
struct line {
  point s, e;      // 直线端点
  double a, b, c;  // ax+by+c=0
  double k;        // 斜率,[-pi, pi]

  line(point _s = point(), point _e = point())
      : s(_s), e(_e) {
    k = atan2(e.y - s.y, e.x - s.x);
    a = e.y - s.y;
    b = s.x - e.x;
    c = e.x * s.y - e.y * s.x;
  }

  // ax + by + c = 0;
  line(const double& _a, const double& _b, const double& _c)
      : a(_a), b(_b), c(_c) {
    if (sgn(a) == 0) {
      s = point(0, -c / b);
      e = point(1, -c / b);
    } else if (sgn(b) == 0) {
      s = point(-c / a, 0);
      e = point(-c / a, 1);
    } else {
      s = point(0, -c / b);
      e = point(1, (-c - a) / b);
    }
  }

  // 点和倾斜角确定直线
  line(const point& a, const double b)
      : s(a) {
    if (sgn(b - pi / 2) == 0)
      e = s + point(0, 1);
    else
      e = s + point(1, tan(b));
  }

```

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

```
  int isLineCrossSeg(line seg) {
    int d1 = sgn(det(s, e, seg.s));
    int d2 = sgn(det(s, e, seg.e));
    if ((d1 ^ d2) == -2)
      return 2;
    return (d1 == 0 || d2 == 0);
  }

  // 求两直线交点
  // 要求两直线不平行或重合
  point getCrossPoint(line l) {
    double a1 = det(l.s, l.e, s);
    double a2 = -det(l.s, l.e, e);
    return (s * a2 + e * a1) / (a1 + a2);
  }

  // 点到直线的距离
  double disPointToLine(const point& p) {
    double d = det(s, p, e) / length();
    return fabs(d);
  }

  // 点到线段的距离
  double disPointToSeg(const point& p) {
    if (sgn(dot(s, p, e)) < 0 || sgn(dot(e, p, s)) < 0)
      return min(distance(p, s), distance(p, e));
    return fabs(disPointToLine(p));
  }

  // 线段到线段的距离
  double disSegToSeg(line& l) {
    if (isSegCrossSeg(l) == 0) {
      double d1 = min(disPointToSeg(l.s), disPointToSeg(l.e));
      double d2 = min(l.disPointToSeg(s), l.disPointToSeg(e));
      return min(d1, d2);
    }
    return 0;
  }

  // 点在直线上的投影
  point projectionPointOnLine(const point& p) {
    return s + (dot(e - s, dot(s, e, p))) / ((e - s).length2());
  }

  // 点关于直线的对称点
  point symmetryPoint(const point& p) {
    point q = projectionPointOnLine(p);
    return point(2 * q.x - p.x, 2 * q.y - p.y);
  }

  // 垂直平分线
  line getVerticalBisector() {
    point m = (s + e) / 2;
    double radian = (e - s).alpha() + pi / 2;
    return line(m, radian);
  }
};

point getLineCrossLine(line l1, line l2) {
  return l1.getCrossPoint(l2);
}

// 向量表示法, 方向为由 s -> e
// struct line
// {
```

```
//     point s,v;
//     line(point a=point(),point b=point()) {
//         s=a;
//         v.x=b.x-a.x;
//         v.y=b.y-a.y;
//     }
// };

// 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
```

```
452  double area() {
453    double sum = 0;
454    for (int i = 0; i < n; i++)
455      sum += det(p[i], p[(i + 1) % n]);
456    return fabs(sum) / 2;
457  }
458
459  void getline() {
460    l.resize(n);
461    for (int i = 0; i < n; i++)
462      l[i] = line(p[i], p[(i + 1) % n]);
463  }
464
465  // 极角排序
466  struct cmp {
467    point p;
468    cmp(const point& _p)
469        : p(_p) {}
470    bool operator()(const point& a, const point& b) const {
471      int d = sgn(det(p, a, b));
472      if (d == 0)
473        return sgn(distance(a, p) - distance(b, p)) < 0;
474      return d > 0;
475    }
476  };
477
478  // 标准化，即极角排序（逆时针）
479  void norm() {
480    point mi = p[0];
481    for (int i = 1; i < n; i++)
482      mi = min(mi, p[i]);
483    sort(p.begin(), p.end(), cmp(mi));
484  }
485
486  // 凸包（非严格）
487  // 若要求严格，则需要再将共线的点除了端点全删去
488  polygon getComvex() {
489    norm();
490    if (n == 0)
491      return polygon(0);
492    else if (n == 1) {
493      polygon convex(1);
494      convex[0] = p[0];
495      return convex;
496    } else if (n == 2) {
497      if (p[0] == p[1]) {
498        polygon convex(1);
499        convex[0] = p[0];
500        return convex;
501      }
502      polygon convex(2);
503      convex[0] = p[0];
504      convex[1] = p[1];
505      return convex;
506    }
507
508    polygon convex(n);
509    convex.p[0] = p[0];
510    convex.p[1] = p[1];
511    int top = 2;
512    for (int i = 2; i < n; i++) {
513      while (top > 1 && sgn(det(convex.p[top - 2], convex.p[top - 1], p[i])) <= 0)
514        --top;
515      convex.p[top++] = p[i];
516    }
```

```
517     convex.resize(top);
518     if (convex.n == 2 && convex.p[0] == convex.p[1])
519       convex.resize(1);
520
521     return convex;
522   }
523
524   bool isConvex() {
525     bool s[3] = {0, 0, 0};
526     for (int i = 0, j, k; i < n; i++) {
527       j = (i + 1) % n;
528       k = (j + 1) % n;
529       s[sgn(det(p[i], p[j], p[k])) + 1] = true;
530       if (s[0] && s[2])
531         return false;
532     }
533     return true;
534   }
535
536   // 多边形方向
537   // 1 逆时针
538   // 2 顺时针
539   int direction() {
540     double sum = 0;
541     for (int i = 0; i < n; i++)
542       sum += det(p[i], p[(i + 1) % n]);
543     if (sgn(sum) > 0)
544       return 1;
545     return 0;
546   }
547
548   // 凸包上最远点对
549   // 平面最远点对就是点集的凸包上的最远点对
550   pair<point, point> getMaxPair() {
551     assert(n >= 2);
552     if (n == 2)
553       return make_pair(p[0], p[1]);
554     point p1 = p[0], p2 = p[1];
555     double dis = distance(p1, p2);
556
557     // 旋转卡 (qia) 壳 (qiao)
558     int k = 1;
559     for (int i = 0; i < n; ++i) {
560       int j = (i + 1) % n;
561       while (sgn(det(p[i], p[j], p[k]) - det(p[i], p[j], p[(k + 1) % n])) <= 0)
562         k = (k + 1) % n;
563
564       if (sgn(distance(p[i], p[k]) - dis) > 0)
565         p1 = p[i], p2 = p[k], dis = distance(p1, p2);
566       if (sgn(distance(p[j], p[k]) - dis) > 0)
567         p1 = p[j], p2 = p[k], dis = distance(p1, p2);
568     }
569     return make_pair(p1, p2);
570   }
571
572   double getMaxDis() {
573     pair<point, point> pr = getMaxPair();
574     return distance(pr.first, pr.second);
575   }
576
577   // 平面最近点对 (P1257, P1429)
578   // 分治法求解平面最近点对，复杂度 O(n log n)
579   void __getMinPair(int l, int r, point& p1, point& p2, double& dis) {
580     if (r - l <= 9) {
581       for (int i = l; i <= r; ++i) {
```

```
582         for (int j = i + 1; j <= r; ++j) {
583           double d = distance(p[i], p[j]);
584           if (d < dis) {
585             dis = d;
586             p1 = p[i];
587             p2 = p[j];
588           }
589         }
590       }
591       return;
592     }
593
594     int m = (l + r) >> 1;
595     __getMinPair(l, m, p1, p2, dis);
596     __getMinPair(m, r, p1, p2, dis);
597     vector<point> tmp;
598     for (int i = l; i <= r; ++i)
599       if (abs(p[i].x - p[m].x) <= dis)
600         tmp.push_back(p[i]);
601     sort(tmp.begin(), tmp.end(), [](const point& a, const point& b) {
602       return a.y < b.y;
603     });
604     for (int i = 1; i < (int)tmp.size(); ++i) {
605       for (int j = i - 1; j >= 0; --j) {
606         if (tmp[j].y < tmp[i].y - dis)
607           break;
608         double d = distance(tmp[i], tmp[j]);
609         if (d < dis) {
610           dis = d;
611           p1 = tmp[i];
612           p2 = tmp[j];
613         }
614       }
615     }
616   }
617
618   pair<point, point> getMinPair() {
619     assert(n >= 1);
620     if (n == 2)
621       return make_pair(p[0], p[1]);
622
623     sort(p.begin(), p.end(), [](const point& a, const point& b) {
624       return a.x < b.x;
625     });
626     point p1 = p[0], p2 = p[1];
627     double dis = distance(p1, p2);
628     __getMinPair(0, n - 1, p1, p2, dis);
629     return make_pair(p1, p2);
630   }
631
632   double getMinDis() {
633     assert(n >= 1);
634     if (n == 2)
635       return distance(p[0], p[1]);
636
637     sort(p.begin(), p.end(), [](const point& a, const point& b) {
638       return a.x < b.x;
639     });
640     point p1 = p[0], p2 = p[1];
641     double dis = distance(p1, p2);
642     __getMinPair(0, n - 1, p1, p2, dis);
643     return dis;
644   }
645
646   // 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);
    int v = sgn(p[j].y - a.y);
    if (k > 0 && u < 0 && v >= 0)
      ++cnt;
    if (k < 0 && v < 0 && u >= 0)
      --cnt;
  }
  return cnt != 0;
}

void DEBUG() {
  cout << n << endl;
  for (int i = 0; i < n; ++i) {
    cout << p[i].x << " " << p[i].y << endl;
  }
}
```

```
};

// 半平面 (ax + by + c >= 0), 其实也就是直线
// 对于直线 (s, e), h.s 为起点, h.e 为方向向量 (e - s)
struct halfplane {
  point s, v;
  double k;
  halfplane() {}
  halfplane(point _s, point _v)
      : s(_s), v(_v) {
    k = v.alpha();
  }
  bool operator<(const halfplane& h) const {
    return k < h.k;
  }
};

// 点和半平面的位置关系
// 0 不在右侧
// 1 在右侧
int relationPointToHalfplane(point p, halfplane h) {
  return sgn(det(h.v, p - h.s)) < 0;
}

// 半平面交点
point HalfplaneCrossHalfplane(halfplane h1, halfplane h2) {
  double a = det(h2.v, h1.s - h2.s) / det(h1.v, h2.v);
  return h1.s + h1.v * a;
}

// 从点集构造出半平面集
// 多边形的半平面集即为多边形边集
void getHalfPlanes(polygon& p, vector<halfplane>& h) {
  if (p.direction() != 1)
    reverse(p.p.begin(), p.p.end());
  int n = p.n;
  for (int i = 0, j; i < n; ++i) {
    j = (i + 1) % n;
    h.push_back(halfplane(p[i], p[j] - p[i]));
  }
}

// 有时候题目给的不一定是闭合图形, 需要自行添加边界
// (x1, y1) 为矩形边界左下角, (x2, y2) 为矩形边界右上角
// Usage: addBorderHalfPlanes(0, 0, 1e4, 1e4, h);
// POJ2451
void addBorderHalfPlanes(double x1, double y1, double x2, double y2, vector<halfplane>& h) {
  polygon p(4);
  p[0] = point(x1, y1);
  p[1] = point(x2, y1);
  p[2] = point(x2, y2);
  p[3] = point(x1, y2);
  getHalfPlanes(p, h);
}

// 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;
```

```cpp
  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;
}
```

```
} // namespace Geometry
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;
  }
```

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

```
      : s(_s), e(_e) {}

  bool operator==(const line3& l) const {
    return (s == l.s) && (e == l.e);
  }

  // 点到直线的距离
  double disPointToLine(point3 p) {
    return det(s, e, p).length() / distance(s, e);
  }

  // 点到线段的距离
  double disPointToSeg(point3 p) {
    if (sgn(dot(s, p, e)) < 0 || sgn(dot(e, p, s)) < 0)
      return min(distance(s, p), distance(e, p));
    return disPointToLine(p);
  }

  // 点在直线上的投影
  point3 projectionPointOnLine(point3 p) {
    return s + (((e - s) * dot(s, e, p)) / (e - s).length2());
  }

  // 绕 p 旋转 alpha 度
  point3 rotate(point3 p, double alpha) {
    if (sgn(det(p, s, e).length()) == 0)
      return p;
    point3 p1 = det(s, e, p);
    point3 p2 = det(e - s, p1);
    double len = det(p, s, e).length() / distance(s, e);
    p1 = p1.trunc(len);
    p2 = p2.trunc(len);
    point3 p3 = p + p2;
    point3 p4 = p3 + p1;
    return p3 + ((p - p3) * cos(alpha) + (p4 - p3) * sin(alpha));
  }

  // 点在线段上
  bool isPointOnSeg(point3 p) {
    return sgn(det(p, s, e).length()) == 0 && sgn(dot(p, s, e)) == 0;
  }
};

struct plane {
  point3 a, b, c;  // 3 点确定平面
  point3 o;        // 平面的法向量

  point3 pvec() {
    return det(a, b, c);
  }

  plane(point3 _a, point3 _b, point3 _c)
      : a(_a), b(_b), c(_c) {}

  plane(point3 _a, point3 _o)
      : a(_a), o(_o) {}

  // ax + by + cz + d = 0;
  plane(double _a, double _b, double _c, double _d) {
    o = point3(_a, _b, _c);
    if (sgn(_a) != 0)
      a = point3((-_d - _c - _b) / _a, 1, 1);
    else if (sgn(_b) != 0)
      a = point3(1, (-_d - _c - _a) / _b, 1);
    else if (sgn(_c != 0))
```

```
      a = point3(1, 1, (-_d - _b - _a) / _c);
  }

  // 点在平面上
  bool isPointOnPlane(point3 p) {
    return sgn((p - a) * o) == 0;
  }

  // 两平面夹角
  double angle(plane f) {
    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];
```

```
255   point3 p2 = P[f.c] - P[f.a];
256   point3 p3 = p - P[f.a];
257   return (p1 ^ p2) * p3;
258 }
259
260 void deal(int p, int a, int b) {
261   int f = G[a][b];
262   if (F[f].ok) {
263     if (cmp(P[p], F[f]) > eps)
264       dfs(p, f);
265     else {
266       face add = {b, a, p, true};
267       G[p][b] = G[a][p] = G[b][a] = num;
268       F[num++] = add;
269     }
270   }
271 }
272
273 void dfs(int p, int now) {
274   F[now].ok = false;
275   deal(p, F[now].b, F[now].a);
276   deal(p, F[now].c, F[now].b);
277   deal(p, F[now].a, F[now].c);
278 }
279
280 bool same(int s, int t) {
281   point3 a = P[F[s].a];
282   point3 b = P[F[s].b];
283   point3 c = P[F[s].c];
284
285   bool flag = sgn(QuadVolume6(a, b, c, P[F[t].a])) == 0 &&
286               sgn(QuadVolume6(a, b, c, P[F[t].b])) == 0 &&
287               sgn(QuadVolume6(a, b, c, P[F[t].c])) == 0;
288
289   return flag;
290 }
291
292 void buildConvex3() {
293   // step 1: 确保前 4 点不共面
294   bool flag = true;
295   for (int i = 1; i < n; ++i) {
296     if (!(P[0] == P[i])) {
297       swap(P[1], P[i]);
298       flag = false;
299       break;
300     }
301   }
302   if (flag)
303     return;
304
305   flag = true;
306   for (int i = 2; i < n; ++i) {
307     if (det(P[0], P[1], P[i]).length() > eps) {
308       swap(P[2], P[i]);
309       flag = false;
310       break;
311     }
312   }
313   if (flag)
314     return;
315
316   flag = true;
317   for (int i = 3; i < n; ++i) {
318     if (fabs(det(P[0], P[1], P[2]) * (P[i] - P[0])) > eps) {
319       swap(P[3], P[i]);
```

```
320         flag = false;
321         break;
322       }
323     }
324     if (flag)
325       return;
326
327     // step 2
328     num = 0;
329     for (int i = 0; i < 4; ++i) {
330       face add = {(i + 1) % 4, (i + 2) % 4, (i + 3) % 4, true};
331       if (cmp(P[i], add) > 0)
332         swap(add.b, add.c);
333       G[add.a][add.b] = G[add.b][add.c] = G[add.c][add.a] = num;
334       F[num++] = add;
335     }
336
337     for (int i = 4; i < n; ++i) {
338       for (int j = 0; j < num; ++j) {
339         if (F[j].ok && cmp(P[i], F[j]) > eps) {
340           dfs(i, j);
341           break;
342         }
343       }
344     }
345
346     int tmp = num;
347     num = 0;
348     for (int i = 0; i < tmp; ++i)
349       if (F[i].ok) {
350         F[num++] = F[i];
351       }
352   }
353
354   // 三维凸包表面积 (POJ3528)
355   double area() {
356     if (n == 3)
357       return det(P[0], P[1], P[2]).length() / 2;
358
359     double res = 0;
360     for (int i = 0; i < num; ++i)
361       res += triArea(P[F[i].a], P[F[i].b], P[F[i].c]);
362     return res;
363   }
364
365   // 三维凸包体积
366   double volume() {
367     double res = 0;
368     point3 tmp(0, 0, 0);
369     for (int i = 0; i < num; ++i)
370       res += QuadVolume(tmp, P[F[i].a], P[F[i].b], P[F[i].c]);
371     return fabs(res);
372   }
373
374   // 表面三角形个数
375   double getTriangleCount() {
376     return num;
377   }
378
379   // 表面多边形个数 (HDU3662)
380   int getPolygonCount() {
381     int res = 0;
382     for (int i = 0; i < num; ++i) {
383       bool flag = true;
384       for (int j = 0; j < i; ++j) {
```

```
385         if (same(i, j)) {
386           flag = 0;
387           break;
388         }
389       }
390       res += flag;
391     }
392     return res;
393   }
394
395   // 重心 (HDU4273)
396   point3 getBaryCenter() {
397     point3 ans(0, 0, 0);
398     point3 o(0, 0, 0);
399
400     double all = 0;
401     for (int i = 0; i < num; ++i) {
402       double v = QuadVolume6(o, P[F[i].a], P[F[i].b], P[F[i].c]);
403       ans = ans + (((o + P[F[i].a] + P[F[i].b] + P[F[i].c]) / 4) * v);
404       all += v;
405     }
406     ans = ans / all;
407     return ans;
408   }
409
410   // 点到凸包第 i 个面上的距离
411   double PointToFace(point3 p, int i) {
412     double v1 = fabs(QuadVolume6(P[F[i].a], P[F[i].b], P[F[i].c], p));
413     double v2 = det(P[F[i].a], P[F[i].b], P[F[i].c]).length();
414     return v1 / v2;
415   }
416 };
417 }  // namespace Geometry3
418 using namespace Geometry3;
```

3.3 BigInt

```
1  // Source: https://github.com/Baobaobear/MiniBigInteger/blob/main/bigint_tiny.h
2  // Author: https://github.com/Baobaobear
3  struct BigInt {
4    int sign;
5    std::vector<int> v;
6
7    BigInt()
8        : sign(1) {
9    }
10   BigInt(const std::string& s) {
11     *this = s;
12   }
13   BigInt(int v) {
14     char buf[21];
15     sprintf(buf, "%d", v);
16     *this = buf;
17   }
18   void zip(int unzip) {
19     if (unzip == 0) {
20       for (int i = 0; i < (int)v.size(); i++)
21         v[i] = get_pos(i * 4) + get_pos(i * 4 + 1) * 10 + get_pos(i * 4 + 2) * 100 +
22                get_pos(i * 4 + 3) * 1000;
23     } else
24       for (int i = (v.resize(v.size() * 4), (int)v.size() - 1), a; i >= 0; i--)
25         a = (i % 4 >= 2) ? v[i / 4] / 100 : v[i / 4] % 100,
26         v[i] = (i & 1) ? a / 10 : a % 10;
27     setsign(1, 1);
```

```
  }
  int get_pos(unsigned pos) const {
    return pos >= v.size() ? 0 : v[pos];
  }
  BigInt& setsign(int newsign, int rev) {
    for (int i = (int)v.size() - 1; i > 0 && v[i] == 0; i--)
      v.erase(v.begin() + i);
    sign = (v.size() == 0 || (v.size() == 1 && v[0] == 0))
               ? 1
               : (rev ? newsign * sign : newsign);
    return *this;
  }
  std::string to_str() const {
    BigInt b = *this;
    std::string s;
    for (int i = (b.zip(1), 0); i < (int)b.v.size(); ++i)
      s += char(*(b.v.rbegin() + i) + '0');
    return (sign < 0 ? "-" : "") + (s.empty() ? std::string("0") : s);
  }
  bool absless(const BigInt& b) const {
    if (v.size() != b.v.size())
      return v.size() < b.v.size();
    for (int i = (int)v.size() - 1; i >= 0; i--)
      if (v[i] != b.v[i])
        return v[i] < b.v[i];
    return false;
  }
  BigInt operator-() const {
    BigInt c = *this;
    c.sign = (v.size() > 1 || v[0]) ? -c.sign : 1;
    return c;
  }
  BigInt& operator=(const std::string& s) {
    if (s[0] == '-')
      *this = s.substr(1);
    else {
      for (int i = (v.clear(), 0); i < (int)s.size(); ++i)
        v.push_back(*(s.rbegin() + i) - '0');
      zip(0);
    }
    return setsign(s[0] == '-' ? -1 : 1, sign = 1);
  }
  bool operator<(const BigInt& b) const {
    return sign != b.sign ? sign < b.sign : (sign == 1 ? absless(b) : !absless(b));
  }
  bool operator==(const BigInt& b) const {
    return v == b.v && sign == b.sign;
  }
  BigInt& operator+=(const BigInt& b) {
    if (sign != b.sign)
      return *this = (*this) - -b;
    v.resize(std::max(v.size(), b.v.size()) + 1);
    for (int i = 0, carry = 0; i < (int)b.v.size() || carry; i++) {
      carry += v[i] + b.get_pos(i);
      v[i] = carry % 10000, carry /= 10000;
    }
    return setsign(sign, 0);
  }
  BigInt operator+(const BigInt& b) const {
    BigInt c = *this;
    return c += b;
  }
  void add_mul(const BigInt& b, int mul) {
    v.resize(std::max(v.size(), b.v.size()) + 2);
    for (int i = 0, carry = 0; i < (int)b.v.size() || carry; i++) {
```

```
      carry += v[i] + b.get_pos(i) * mul;
      v[i] = carry % 10000, carry /= 10000;
    }
  }
  BigInt operator-(const BigInt& b) const {
    if (sign != b.sign)
      return (*this) + -b;
    if (absless(b))
      return -(b - *this);
    BigInt c;
    for (int i = 0, borrow = 0; i < (int)v.size(); i++) {
      borrow += v[i] - b.get_pos(i);
      c.v.push_back(borrow);
      c.v.back() -= 10000 * (borrow >>= 31);
    }
    return c.setsign(sign, 0);
  }
  BigInt operator*(const BigInt& b) const {
    if (b < *this)
      return b * *this;
    BigInt c, d = b;
    for (int i = 0; i < (int)v.size(); i++, d.v.insert(d.v.begin(), 0))
      c.add_mul(d, v[i]);
    return c.setsign(sign * b.sign, 0);
  }
  BigInt operator/(const BigInt& b) const {
    BigInt c, d;
    d.v.resize(v.size());
    double db = 1.0 / (b.v.back() + (b.get_pos((unsigned)b.v.size() - 2) / 1e4) +
                       (b.get_pos((unsigned)b.v.size() - 3) + 1) / 1e8);
    for (int i = (int)v.size() - 1; i >= 0; i--) {
      c.v.insert(c.v.begin(), v[i]);
      int m = (int)((c.get_pos((int)b.v.size()) * 10000 +
                     c.get_pos((int)b.v.size() - 1)) *
                    db);
      c = c - b * m, d.v[i] += m;
      while (!(c < b))
        c = c - b, d.v[i] += 1;
    }
    return d.setsign(sign * b.sign, 0);
  }
  BigInt operator%(const BigInt& b) const {
    return *this - *this / b * b;
  }
  bool operator>(const BigInt& b) const {
    return b < *this;
  }
  bool operator<=(const BigInt& b) const {
    return !(b < *this);
  }
  bool operator>=(const BigInt& b) const {
    return !(*this < b);
  }
  bool operator!=(const BigInt& b) const {
    return !(*this == b);
  }
};
```

3.4 BSGS

```
namespace Backlight {

namespace BSGS {
typedef long long ll;
```

```cpp

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;
}
}  // namespace BSGS

}  // namespace Backlight
```

3.5 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.i % P +
};

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
```

3.6 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);
```

3.7 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;
}

}  // namespace Backlight
```

3.8 du

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

using ll = int64_t;

const int LIM = 1e7;

int pcnt, prime[LIM], mu[LIM];
bool vis[LIM];
void seive(int n) {
  pcnt = 0;
  mu[0] = 0;
  mu[1] = 1;
  for (int i = 2; i <= n; ++i) {
    if (!vis[i]) {
      prime[++pcnt] = i;
      mu[i] = -1;
    }
    for (int j = 1; j <= pcnt; ++j) {
      ll nxt = 1ll * i * prime[j];
      if (nxt > n)
```

```
      break;
    vis[nxt] = true;
    if (i % prime[j] == 0) {
      mu[nxt] = 0;
      break;
    }
    mu[nxt] = -mu[i];
  }
 }
 for (int i = 1; i <= n; ++i)
  mu[i] += mu[i - 1];
}

map<ll, ll> mp;

// S(n) = 1 - \sum_{i=2} S(\lfloor \frac{n}{i} \rfloor)
// Time Complexity: O(n^{\frac{2}{3}})
ll S_mu(ll n) {
 if (n < LIM)
  return mu[n];
 if (mp.count(n))
  return mp[n];

 ll ret = 0;
 for (ll i = 2, j; i <= n; i = j + 1) {
  j = n / (n / i);
  ret += (j - i + 1) * S_mu(n / i);
 }
 ret = 1 - ret;

 mp[n] = ret;
 return ret;
}

// S(n) = \frac{(n+1)n}{2} - \sum_{i=2}^{n} S(\lfloor \frac{n}{i} \rfloor)
// S(n) = \sum_{d=1}^{n} \mu(d) \lfloor \frac{n}{d} \rfloor \lfloor \frac{n}{d} \rfloor
ll S_phi(ll n) {
 ll ret = 0;
 for (ll i = 1, j; i <= n; i = j + 1) {
  j = n / (n / i);
  ret += (S_mu(j) - S_mu(i - 1)) * (n / i) * (n / i);
 }
 ret = (ret - 1) / 2 + 1;
 return ret;
}

void solve(int Case) {
 ll n;
 scanf("%lld", &n);
 printf("%lld %lld\n", S_phi(n), S_mu(n));
}

int main() {
 seive(LIM - 1);
 int T = 1;
 scanf("%d", &T);
 for (int _ = 1; _ <= T; _++)
  solve(_);
 return 0;
}
```

3.9 EulerSeive

```cpp
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;
}

}  // namespace Backlight
```

3.10 eval

```cpp
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();
}
```

3.11 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;
}

}  // namespace Backlight
```

3.12 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);
}

void FFT(vector<Complex>& a, int op) {
  for (int i = 1; i < N; ++i)
    if (i < r[i])
      swap(a[i], a[r[i]]);
  for (int i = 2; i <= N; i <<= 1) {
    int l = i >> 1;
    Complex w, x, wk(cos(PI / l), op * sin(PI / l));
    for (int j = 0; j < N; j += i) {
      w = Complex(1, 0);
      for (int k = j; k < j + l; ++k) {
        x = a[k + l] * w;
        a[k + l] = a[k] - x;
        a[k] = a[k] + x;
```

```
        w = w * wk;
      }
    }
  }
  if (op == -1)
    for (int i = 0; i < N; i++)
      a[i].r /= N, a[i].i /= N;
}

inline void FFT(vector<Complex>& a) {
  FFT(a, 1);
}
inline void IFT(vector<Complex>& a) {
  FFT(a, -1);
}

vector<int> convolution(const vector<int>& f, const vector<int>& g) {
  int n = f.size(), m = g.size(), k = n + m - 1;
  init(k);
  vector<Complex> a(N), b(N);
  for (int i = 0; i < n; ++i)
    a[i] = Complex(f[i], 0);
  for (int i = 0; i < m; ++i)
    b[i] = Complex(g[i], 0);

  FFT(a);
  FFT(b);
  for (int i = 0; i < N; ++i)
    a[i] = a[i] * b[i];
  IFT(a);

  vector<int> h(k);
  for (int i = 0; i < k; ++i)
    h[i] = int(a[i].r + 0.5);
  return h;
}

// 任意模数 FFT
vector<int> convolutionM(const vector<int>& f, const vector<int>& g, int p) {
  int n = f.size(), m = g.size(), k = n + m - 1;
  init(k);
  vector<Complex> a(N), b(N), c(N), d(N);
  for (int i = 0; i < n; ++i)
    a[i] = Complex(f[i] >> 15, f[i] & 32767);
  for (int i = 0; i < m; ++i)
    c[i] = Complex(g[i] >> 15, g[i] & 32767);
  FFT(a);
  FFT(c);
  for (int i = 1; i < N; ++i)
    b[i] = a[N - i].conj();
  for (int i = 1; i < N; ++i)
    d[i] = c[N - i].conj();
  b[0] = a[0].conj();
  d[0] = c[0].conj();
  for (int i = 0; i < N; ++i) {
    Complex aa, bb, cc, dd;
    aa = (a[i] + b[i]) * Complex(0.5, 0);
    bb = (a[i] - b[i]) * Complex(0, -0.5);
    cc = (c[i] + d[i]) * Complex(0.5, 0);
    dd = (c[i] - d[i]) * Complex(0, -0.5);
    a[i] = aa * cc + Complex(0, 1) * (aa * dd + bb * cc);
    b[i] = bb * dd;
  }
  IFT(a);
  IFT(b);
```

```
  vector<int> h(k);
  for (int i = 0; i < k; ++i) {
    int aa, bb, cc;
    aa = LL(a[i].r + 0.5) % p;
    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
```

3.13 FWT

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

const int MOD = 998244353;

inline int add(int x, int y) {
  return x + y >= MOD ? x + y - MOD : x + y;
}
inline int mul(int x, int y) {
  return 1ll * x * y % MOD;
}
inline int sub(int x, int y) {
  return x - y < 0 ? x - y + MOD : x - y;
}
inline int qp(int x, int y) {
  int r = 1;
  for (; y; y >>= 1) {
    if (y & 1)
      r = mul(r, x);
    x = mul(x, x);
  }
  return r;
}
inline int inv(int x) {
  return qp(x, MOD - 2);
}
inline int dvd(int x, int y) {
  return 1ll * x * qp(y, MOD - 2) % MOD;
}

namespace FWT {
void OR(int* a, int n) {
  for (int o = 2, k = 1; o <= n; o <<= 1, k <<= 1)
    for (int i = 0; i < n; i += o)
      for (int j = 0; j < k; ++j)
        a[i + j + k] = add(a[i + j + k], a[i + j]);
}

void IOR(int* a, int n) {
  for (int o = 2, k = 1; o <= n; o <<= 1, k <<= 1)
    for (int i = 0; i < n; i += o)
      for (int j = 0; j < k; ++j)
        a[i + j + k] = sub(a[i + j + k], a[i + j]);
}

void AND(int* a, int n) {
  for (int o = 2, k = 1; o <= n; o <<= 1, k <<= 1)
    for (int i = 0; i < n; i += o)
      for (int j = 0; j < k; ++j)
        a[i + j] = add(a[i + j], a[i + j + k]);
```

```
}

void IAND(int* a, int n) {
  for (int o = 2, k = 1; o <= n; o <<= 1, k <<= 1)
    for (int i = 0; i < n; i += o)
      for (int j = 0; j < k; ++j)
        a[i + j] = sub(a[i + j], a[i + j + k]);
}

void XOR(int* a, int n) {
  int x, y;
  for (int o = 2, k = 1; o <= n; o <<= 1, k <<= 1)
    for (int i = 0; i < n; i += o)
      for (int j = 0; j < k; ++j) {
        x = a[i + j], y = a[i + j + k];
        a[i + j] = add(x, y);
        a[i + j + k] = sub(x, y);
      }
}

int inv2 = inv(2);
void IXOR(int* a, int n) {
  int x, y;
  for (int o = 2, k = 1; o <= n; o <<= 1, k <<= 1)
    for (int i = 0; i < n; i += o)
      for (int j = 0; j < k; ++j) {
        x = a[i + j], y = a[i + j + k];
        a[i + j] = mul(add(x, y), inv2);
        a[i + j + k] = mul(sub(x, y), inv2);
      }
}
}  // namespace FWT

const int N = (1 << 17) + 5;

int n;
int A[N], B[N], a[N], b[N], c[N];

int main() {
  scanf("%d", &n);
  n = 1 << n;

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

  // OR
  for (int i = 0; i < n; ++i)
    a[i] = A[i], b[i] = B[i];
  FWT::OR(a, n);
  FWT::OR(b, n);
  for (int i = 0; i < n; ++i)
    c[i] = mul(a[i], b[i]);
  FWT::IOR(c, n);

  for (int i = 0; i < n - 1; ++i)
    printf("%d ", c[i]);
  printf("%d\n", c[n - 1]);

  // AND
  for (int i = 0; i < n; ++i)
    a[i] = A[i], b[i] = B[i];
  FWT::AND(a, n);
```

```
  FWT::AND(b, n);
  for (int i = 0; i < n; ++i)
    c[i] = mul(a[i], b[i]);
  FWT::IAND(c, n);
  for (int i = 0; i < n - 1; ++i)
    printf("%d ", c[i]);
  printf("%d\n", c[n - 1]);

  // XOR
  for (int i = 0; i < n; ++i)
    a[i] = A[i], b[i] = B[i];
  FWT::XOR(a, n);
  FWT::XOR(b, n);
  for (int i = 0; i < n; ++i)
    c[i] = mul(a[i], b[i]);
  FWT::IXOR(c, n);
  for (int i = 0; i < n - 1; ++i)
    printf("%d ", c[i]);
  printf("%d\n", c[n - 1]);
  return 0;
}
```

3.14 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;
  }
};
```

3.15 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;
}

```

```cpp
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
```

3.16 min25

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

using namespace std;
using ll = int64_t;

const int MOD = 1e9 + 7;

template <typename T>
inline int mint(T x) {
  x %= MOD;
  if (x < 0)
    x += MOD;
  return x;
}
inline int add(int x, int y) {
  return x + y >= MOD ? x + y - MOD : x + y;
}
inline int mul(int x, int y) {
  return 1ll * x * y % MOD;
}
inline int sub(int x, int y) {
  return x < y ? x - y + MOD : x - y;
}
inline int qp(int x, int y) {
  int r = 1;
  for (; y; y >>= 1) {
    if (y & 1)
      r = mul(r, x);
    x = mul(x, x);
  }
  return r;
}
```

```cpp
inline int inv(int x) {
  return qp(x, MOD - 2);
}
inline int dvd(int x, int y) {
  return 1ll * x * qp(y, MOD - 2) % MOD;
}
inline void inc(int& x, int y) {
  x += y;
  if (x >= MOD)
    x -= MOD;
}
inline void dec(int& x, int y) {
  x -= y;
  if (x < 0)
    x += MOD;
}

namespace min25 {
/*
    calc the prefix sum of  multiplicative function.
    Requirements:
        Assume $p$ is a prime number.
        1. $f(p)$ is a polynomial with small deg or can be calc quickly.
        2. $f(p^e)$ can be calc quickly.
    Time complexity: $O(\frac{n^{0.75}}{\log n})$
    Steps: assume $deg(f(p)) = n$.
        1. split $f(p)$ into $n$ parts by it exponent.
        2. calc them separately.
        3. then sum them up.
    e.g.: $f(p) = \phi(p) = p - 1$
    1. calc $ans_0$ for $f(p) = p$.
    2. calc $ans_1$ for $f(p) = 1$.
    3. $ans = ans_0 - ans_1$.
    */
const int LIM = 2e5 + 9;

ll gn, w[LIM];
int rt, lim, id1[LIM], id2[LIM];
#define idx(v) (v <= rt ? id1[v] : id2[gn / v])

int pcnt, prime[LIM];
bool isp[LIM];

// $sp_{i,j} = \sum_{k=1}^{j} [p \text{ is a prime}] p^i$
int sp1[LIM];

void seive(const int& n) {
  pcnt = 0;
  fill(isp, isp + n + 1, true);
  for (int i = 2; i <= n; ++i) {
    if (isp[i]) {
      ++pcnt;
      prime[pcnt] = i;
    }
    for (int j = 1; j <= pcnt; ++j) {
      ll nxt = 1ll * i * prime[j];
      if (nxt > n)
        break;
      isp[nxt] = false;
      if (i % prime[j] == 0)
        break;
    }
  }
  for (int i = 1; i <= pcnt; ++i)
    sp1[i] = add(sp1[i - 1], prime[i]);
```

```
}

int G[LIM][2], H[LIM];

void initG0(const ll& n) {
  lim = 0;
  int inv2 = inv(2);
  for (ll i = 1, j, v; i <= n; i = n / j + 1) {
    j = n / i;

    w[++lim] = j;
    idx(j) = lim;

    v = j % MOD;

    // init G_0 = \sum_{i=2}^{n} g(i)
    G[lim][0] = sub(v, 1);
    G[lim][1] = mul(mul(mint(v + 2), mint(v - 1)), inv2);
  }
}

void calcH() {
  for (int k = 1; k <= pcnt; ++k) {
    const int p = prime[k];
    const ll p2 = 1ll * p * p;
    for (int i = 1; w[i] >= p2; ++i) {
      const ll v = w[i] / p;
      int id = idx(v);
      dec(G[i][0], sub(G[id][0], k - 1));
      dec(G[i][1], mul(p, sub(G[id][1], sp1[k - 1])));
    }
  }
  for (int i = 1; i <= lim; ++i)
    H[i] = sub(G[i][1], G[i][0]);
}

// f(p^e)
inline int fpe(const int& p, const int& e) {
  return p xor e;
}

int F(const int& k, const ll& n) {
  if (n < prime[k] || n <= 1)
    return 0;

  int r1 = 0;
  for (int i = k; i <= pcnt; ++i) {
    ll pi = prime[i];
    if (1ll * pi * pi > n)
      break;
    ll pc = pi, pc2 = pi * pi;
    for (int c = 1; pc2 <= n; ++c) {
      inc(r1, add(mul(fpe(pi, c), F(i + 1, n / pc)), fpe(pi, c + 1)));
      pc = pc2;
      pc2 = pc2 * pi;
    }
  }

  // H(n) - H(p_{k-1})
  const int id = idx(n);
  int r2 = sub(H[id], sub(sp1[k - 1], k - 1));
  if (k == 1)
    inc(r2, 2);
  int ans = add(r1, r2);
  return ans;
```

```
}

int solve(ll n) {
  gn = n;
  rt = sqrt(gn);
  seive(rt + 5);
  initG0(gn);
  calcH();
  return add(F(1, n), 1);
}
}  // namespace min25

int main() {
  ll n;
  scanf("%lld", &n);
  printf("%d\n", min25::solve(n));
  return 0;
}
```

3.17 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; }
```

```
44
45    Modular& operator+=(const Modular& other) {
46      if ((value += other.value) >= mod())
47        value -= mod();
48      return *this;
49    }
50    Modular& operator-=(const Modular& other) {
51      if ((value -= other.value) < 0)
52        value += mod();
53      return *this;
54    }
55    template <typename U>
56    Modular& operator+=(const U& other) { return *this += Modular(other); }
57    template <typename U>
58    Modular& operator-=(const U& other) { return *this -= Modular(other); }
59    Modular& operator++() { return *this += 1; }
60    Modular& operator--() { return *this -= 1; }
61    Modular operator++(int) {
62      Modular result(*this);
63      *this += 1;
64      return result;
65    }
66    Modular operator--(int) {
67      Modular result(*this);
68      *this -= 1;
69      return result;
70    }
71    Modular operator-() const { return Modular(-value); }
72
73    template <typename U = T>
74    typename enable_if<is_same<typename Modular<U>::Type, int>::value, Modular>::type& operator*=(const Modular& rhs) {
75  #ifdef _WIN32
76      uint64_t x = static_cast<int64_t>(value) * static_cast<int64_t>(rhs.value);
77      uint32_t xh = static_cast<uint32_t>(x >> 32), xl = static_cast<uint32_t>(x), d, m;
78      asm(
79          "divl %4; \n\t"
80          : "=a"(d), "=d"(m)
81          : "d"(xh), "a"(xl), "r"(mod()));
82      value = m;
83  #else
84      value = normalize(static_cast<int64_t>(value) * static_cast<int64_t>(rhs.value));
85  #endif
86      return *this;
87    }
88    template <typename U = T>
89    typename enable_if<is_same<typename Modular<U>::Type, long long>::value, Modular>::type& operator*=(const Modular& rh
90      long long q = static_cast<long long>(static_cast<long double>(value) * rhs.value / mod());
91      value = normalize(value * rhs.value - q * mod());
92      return *this;
93    }
94    template <typename U = T>
95    typename enable_if<!is_integral<typename Modular<U>::Type>::value, Modular>::type& operator*=(const Modular& rhs) {
96      value = normalize(value * rhs.value);
97      return *this;
98    }
99
100   Modular& operator/=(const Modular& other) { return *this *= Modular(inverse(other.value, mod())); }
101
102   friend const Type& abs(const Modular& x) { return x.value; }
103
104   template <typename U>
105   friend bool operator==(const Modular<U>& lhs, const Modular<U>& rhs);
106
107   template <typename U>
108   friend bool operator<(const Modular<U>& lhs, const Modular<U>& rhs);
```

```
109
110   template <typename V, typename U>
111   friend V& operator>>(V& stream, Modular<U>& number);
112
113  private:
114   Type value;
115 };
116
117 template <typename T>
118 bool operator==(const Modular<T>& lhs, const Modular<T>& rhs) {
119   return lhs.value == rhs.value;
120 }
121 template <typename T, typename U>
122 bool operator==(const Modular<T>& lhs, U rhs) {
123   return lhs == Modular<T>(rhs);
124 }
125 template <typename T, typename U>
126 bool operator==(U lhs, const Modular<T>& rhs) {
127   return Modular<T>(lhs) == rhs;
128 }
129
130 template <typename T>
131 bool operator!=(const Modular<T>& lhs, const Modular<T>& rhs) {
132   return !(lhs == rhs);
133 }
134 template <typename T, typename U>
135 bool operator!=(const Modular<T>& lhs, U rhs) {
136   return !(lhs == rhs);
137 }
138 template <typename T, typename U>
139 bool operator!=(U lhs, const Modular<T>& rhs) {
140   return !(lhs == rhs);
141 }
142
143 template <typename T>
144 bool operator<(const Modular<T>& lhs, const Modular<T>& rhs) {
145   return lhs.value < rhs.value;
146 }
147
148 template <typename T>
149 Modular<T> operator+(const Modular<T>& lhs, const Modular<T>& rhs) {
150   return Modular<T>(lhs) += rhs;
151 }
152 template <typename T, typename U>
153 Modular<T> operator+(const Modular<T>& lhs, U rhs) {
154   return Modular<T>(lhs) += rhs;
155 }
156 template <typename T, typename U>
157 Modular<T> operator+(U lhs, const Modular<T>& rhs) {
158   return Modular<T>(lhs) += rhs;
159 }
160
161 template <typename T>
162 Modular<T> operator-(const Modular<T>& lhs, const Modular<T>& rhs) {
163   return Modular<T>(lhs) -= rhs;
164 }
165 template <typename T, typename U>
166 Modular<T> operator-(const Modular<T>& lhs, U rhs) {
167   return Modular<T>(lhs) -= rhs;
168 }
169 template <typename T, typename U>
170 Modular<T> operator-(U lhs, const Modular<T>& rhs) {
171   return Modular<T>(lhs) -= rhs;
172 }
173
```

```cpp
template <typename T>
Modular<T> operator*(const Modular<T>& lhs, const Modular<T>& rhs) {
  return Modular<T>(lhs) *= rhs;
}
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) /= rhs;
}
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];
}
*/
```

3.18 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];
}
```

3.19 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;
}
```

3.20 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
```

3.21 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;
}
}  // namespace Pollard_Rho

}  // namespace Backlight
```

3.22 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;
```

```
  }
};
```

3.23 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;
  }
}
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)
```

```cpp
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
}

// 多项式指数函数
// b(x) = exp a(x) mod x^n
void Exp(ll* a, ll* b, int n) {
#define Lnb t2

  if (n == 1) {
    b[0] = 1;
    return;
  }
  Exp(a, b, (n + 1) >> 1);
  Ln(b, Lnb, 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);

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

  for (int i = 0; i < N; ++i)
    t1[i] = ((t1[i] - Lnb[i]) % P + P) % P;
  ++t1[0];
  NTT(b, N);
  NTT(t1, N);
  for (int i = 0; i < N; ++i)
    b[i] = b[i] * t1[i] % P;
  INTT(b, N);

  fill(b + n, b + N, 0);
#undef Lnb
}

// 多项式乘法 (卷积)
// c(x) = a(x) * b(x) mod x^(n + m)
// deg c = n + m - 1
void Convolution(ll* a, int n, ll* b, int m, ll* c) {
  int N = 1, l = -1;
  while (N <= (n + m) << 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);
  memcpy(t2, b, sizeof(b[0]) * m);
  fill(t2 + m, t2 + N, 0);

  NTT(t1, N);
  NTT(t2, N);
  for (int i = 0; i < N; ++i)
    c[i] = t1[i] * t2[i] % P;
```

```
  INTT(c, N);
  fill(c + n + m, c + N, 0);
}
#define Multiply Convolution

// 多项式除法
// a(x) = b(x)Q(x) + R(x)
// deg Q = n - m + 1
// deg R = m - 1
void Divide(ll* a, int n, ll* b, int m, ll* Q, ll* R) {
#define aR t3
#define bR t4
#define bRi t5
#define QR t6
#define bQ t7

  int degQ = n - m + 1;
  int degR = m - 1;

  Reverse(a, aR, n);
  Reverse(b, bR, m);
  for (int i = degQ; i < m; ++i)
    bR[i] = 0;

  // get Q(x)
  Inverse(bR, bRi, degQ);
  Multiply(aR, n, bRi, degQ, QR);
  Reverse(QR, Q, degQ);

  // get R(x)
  Multiply(b, m, Q, degQ, bQ);
  for (int i = 0; i < degR; ++i)
    R[i] = (a[i] - bQ[i] + P) % P;

#undef aR
#undef bR
#undef bRi
#undef QR
}

// 多项式求平方根
// b^{2}(x) = a(x)
#define bI t3
void __Sqrt(ll* a, ll* b, int n) {
  if (n == 1) {
    b[0] = 1;
    return;
  }

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

  Inverse(b, bI, n);
  Multiply(a, n, bI, n, bI);
  for (int i = 0; i < n; ++i)
    b[i] = (b[i] + bI[i]) * I2 % P;
}
inline void Sqrt(ll* a, ll* b, int n) {
  fill(bI, bI + (n << 2), 0);
  __Sqrt(a, b, n);
}
#undef bI

struct poly {
  vector<ll> a;
  int size() const { return a.size(); }
```

```
  int deg() const { return size() - 1; }
  ll& operator[](int i) {
    assert(i < size());
    return a[i];
  }
  ll operator[](int i) const { return i < size() ? a[i] : 0LL; }
  void reverse() { std::reverse(a.begin(), a.end()); }
  void resize(int n) { a.resize(n); }
  poly(int n = 0)
      : a(n, 0) {}

  void DEBUG() {
    cerr << "Poly DEBUG: " << endl;
    for (const ll& v : a)
      cerr << v << " ";
    cerr << endl;
  }

  void DEBUG() const {
    cerr << "Poly DEBUG: " << endl;
    for (const ll& v : a)
      cerr << v << " ";
    cerr << endl;
  }

  void input() {
    for (ll& x : a)
      read(x);
  }

  void output() {
    if (a.empty()) {
      puts("");
      return;
    }
    int n = a.size();
    for (int i = 0; i < n - 1; ++i)
      printf("%lld ", a[i]);
    printf("%lld\n", a[n - 1]);
  }

  void output() const {
    if (a.empty()) {
      puts("");
      return;
    }
    int n = a.size();
    for (int i = 0; i < n - 1; ++i)
      printf("%lld ", a[i]);
    printf("%lld\n", a[n - 1]);
  }

  poly inv(int n = -1) const {
    if (n == -1)
      n = size();
    static ll f[MAXN], g[MAXN];
    for (int i = 0; i < n; ++i)
      f[i] = a[i];
    Inverse(f, g, n);
    poly res(n);
    for (int i = 0; i < n; ++i)
      res[i] = g[i];
    return res;
  }

```

```
  poly rev() const {
    int n = size();
    poly r(n);
    for (int i = 0; i < n; ++i)
      r[i] = a[n - i - 1];
    return r;
  }

  poly sqrt() {
    int n = a.size();
    static ll f[MAXN], g[MAXN];
    for (int i = 0; i < n; ++i)
      f[i] = a[i];
    Sqrt(f, g, n);
    poly res(n);
    for (int i = 0; i < n; ++i)
      res[i] = g[i];
    return res;
  }
};

poly operator+(const poly& a, const poly& b) {
  int k = max(a.size(), b.size());
  poly c(k);
  for (int i = 0; i < k; ++i)
    c[i] = (a[i] + b[i]) % P;
  return c;
}

poly operator-(const poly& a, const poly& b) {
  int k = max(a.size(), b.size());
  poly c(k);
  for (int i = 0; i < k; ++i)
    c[i] = (a[i] - b[i] + P) % P;
  return c;
}

poly operator*(const poly& a, const poly& b) {
  static ll ta[MAXN], tb[MAXN];
  int n = a.size(), m = b.size(), k = n + m - 1;
  for (int i = 0; i < n; ++i)
    ta[i] = a[i];
  for (int i = 0; i < m; ++i)
    tb[i] = b[i];

  Multiply(ta, n, tb, m, ta);

  poly c(k);
  for (int i = 0; i < k; ++i)
    c[i] = ta[i];
  return c;
}

pair<poly, poly> Divide(const poly& a, const poly& b) {
  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];

```

```
385   Divide(ta, n, tb, m, tq, tr);
386
387   poly q(degQ);
388   for (int i = 0; i < degQ; ++i)
389     q[i] = tq[i];
390   poly r(degR);
391   for (int i = 0; i < degR; ++i)
392     r[i] = tr[i];
393
394   return make_pair(q, r);
395 }
396
397 poly operator/(const poly& a, const poly& b) {
398   return Divide(a, b).first;
399 }
400 poly operator%(const poly& a, const poly& b) {
401   return Divide(a, b).second;
402 }
403
404 // given a(x), deg a = n
405 // calc y_i = a(x_i) for i in [0, m), O(n \log^2 n)
406 poly t[N << 2], p[N];
407 void build(int o, int l, int r) {
408   if (l == r) {
409     t[o] = p[l];
410     return;
411   }
412   int mid = (l + r) >> 1;
413   build(o << 1, l, mid);
414   build(o << 1 | 1, mid + 1, r);
415   t[o] = t[o << 1] * t[o << 1 | 1];
416 }
417 void __calcValue(int o, int l, int r, const poly& f, ll* x, ll* y) {
418   // if (l == r) {
419   //     y[l] = f[0];
420   //     return;
421   // }
422   if (r - l <= 75) {  // 降低常数 (魔法)
423     for (int i = l; i <= r; ++i) {
424       ll v = 0;
425       for (int j = f.size() - 1; j >= 0; --j)
426         v = (v * x[i] % P + f[j]) % P;
427       y[i] = v;
428     }
429     return;
430   }
431
432   int mid = (l + r) >> 1, lc = o << 1, rc = o << 1 | 1;
433   __calcValue(lc, l, mid, f % t[lc], x, y);
434   __calcValue(rc, mid + 1, r, f % t[rc], x, y);
435 }
436 void calcValue(const poly& f, ll* x, ll* y, int m) {
437   for (int i = 1; i <= m; ++i) {
438     p[i].resize(2);
439     p[i][0] = P - x[i];
440     p[i][1] = 1;
441   }
442   build(1, 1, m);
443   __calcValue(1, 1, m, f % t[1], x, y);
444 }
445 }  // namespace Poly
```

3.24 PowerfulNumber

```cpp
#include <bits/stdc++.h>
using namespace std;
using ll = int64_t;

constexpr int MOD = 1e9 + 7;  // 998244353 1e9 + 7
template <typename T>
inline int mint(T x) {
  x %= MOD;
  if (x < 0)
    x += MOD;
  return x;
}
inline int add(int x, int y) {
  return x + y >= MOD ? x + y - MOD : x + y;
}
inline int mul(int x, int y) {
  return 1ll * x * y % MOD;
}
inline int sub(int x, int y) {
  return x < y ? x - y + MOD : x - y;
}
inline int qp(int x, int y) {
  int r = 1;
  for (; y; y >>= 1) {
    if (y & 1)
      r = mul(r, x);
    x = mul(x, x);
  }
  return r;
}
inline int inv(int x) {
  return qp(x, MOD - 2);
}

namespace PNS {
const int N = 2e6 + 5;
const int M = 35;

ll global_n;

int g[N], sg[N];

int h[N][M];
bool vis_h[N][M];

int ans;

int pcnt, prime[N], phi[N];
bool isp[N];

void sieve(int n) {
  pcnt = 0;
  for (int i = 2; i <= n; ++i)
    isp[i] = true;
  phi[1] = 1;
  for (int i = 2; i <= n; ++i) {
    if (isp[i]) {
      ++pcnt;
      prime[pcnt] = i;
      phi[i] = i - 1;
    }
    for (int j = 1; j <= pcnt; ++j) {
      ll nxt = 1ll * i * prime[j];
```

```
      if (nxt > n)
        break;
      isp[nxt] = false;
      if (i % prime[j] == 0) {
        phi[nxt] = phi[i] * prime[j];
        break;
      }
      phi[nxt] = phi[i] * phi[prime[j]];
    }
  }

  for (int i = 1; i <= n; ++i)
    g[i] = mul(i, phi[i]);

  sg[0] = 0;
  for (int i = 1; i <= n; ++i)
    sg[i] = add(sg[i - 1], g[i]);
}

int inv2, inv6;
void init() {
  sieve(N - 1);
  for (int i = 1; i <= pcnt; ++i)
    h[i][0] = 1, h[i][1] = 0;
  for (int i = 1; i <= pcnt; ++i)
    vis_h[i][0] = vis_h[i][1] = true;
  inv2 = inv(2);
  inv6 = inv(6);
}

int S1(ll n) {
  return mul(mul(mint(n), mint(n + 1)), inv2);
}

int S2(ll n) {
  return mul(mul(mint(n), mul(mint(n + 1), mint(n * 2 + 1))), inv6);
}

map<ll, int> mp_g;

int G(ll n) {
  if (n < N)
    return sg[n];
  if (mp_g.count(n))
    return mp_g[n];

  int ret = S2(n);
  for (ll i = 2, j; i <= n; i = j + 1) {
    j = n / (n / i);
    ret = sub(ret, mul(sub(S1(j), S1(i - 1)), G(n / i)));
  }
  mp_g[n] = ret;
  return ret;
}

void dfs(ll d, int hd, int pid) {
  ans = add(ans, mul(hd, G(global_n / d)));

  for (int i = pid, p; i <= pcnt; ++i) {
    if (i > 1 && d > global_n / prime[i] / prime[i])
      break;

    int c = 2;
    for (ll x = d * prime[i] * prime[i]; x <= global_n; x *= prime[i], ++c) {
      if (!vis_h[i][c]) {
```

```
        int f = qp(prime[i], c);
        f = mul(f, sub(f, 1));
        int g = mul(prime[i], prime[i] - 1);
        int t = mul(prime[i], prime[i]);

        for (int j = 1; j <= c; ++j) {
          f = sub(f, mul(g, h[i][c - j]));
          g = mul(g, t);
        }
        h[i][c] = f;
        vis_h[i][c] = true;
      }

      if (h[i][c])
        dfs(x, mul(hd, h[i][c]), i + 1);
    }
  }
}

int solve(ll n) {
  global_n = n;
  ans = 0;
  dfs(1, 1, 1);
  return ans;
}
}  // namespace PNS

int main() {
  PNS::init();
  ll n;
  scanf("%lld", &n);
  printf("%d\n", PNS::solve(n));
  return 0;
}
```

3.25 Simplex

```
/**
 * 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;
```

```
28  vector<double> b, x;
29  vector<int> idx, idy;
30
31  Simplex(int _n, int _m)
32      : n(_n), m(_m) {
33    A = vector<vector<double>>(m + 1, vector<double>(n + 1));
34    b = vector<double>(m + 1);
35    x = vector<double>(n + 1);
36    idx = vector<int>(m + 1);
37    idy = vector<int>(n + 1);
38  }
39
40  void input() {
41    for (int i = 1; i <= n; ++i)
42      read(A[0][i]);  // A_{0,i} = c_i
43    for (int i = 1; i <= m; ++i) {
44      for (int j = 1; j <= n; ++j)
45        read(A[i][j]);
46      read(b[i]);
47    }
48  }
49
50  void pivot(int x, int y) {
51    swap(idx[x], idy[y]);
52
53    double k = A[x][y];
54    for (int i = 1; i <= n; ++i)
55      A[x][i] /= k;
56    b[x] /= k;
57    A[x][y] = 1 / k;
58
59    for (int i = 0; i <= m; ++i)
60      if (i != x) {
61        k = A[i][y];
62        b[i] -= k * b[x];
63        A[i][y] = 0;
64        for (int j = 1; j <= n; ++j)
65          A[i][j] -= k * A[x][j];
66      }
67  }
68
69  void init() {
70    flag = OK;
71    idx[0] = INT_MAX;
72    for (int i = 1; i <= m; ++i)
73      idx[i] = n + i;
74    idy[0] = INT_MAX;
75    for (int i = 1; i <= n; ++i)
76      idy[i] = i;
77
78    for (;;) {
79      int x = 0, y = 0;
80      for (int i = 1; i <= m; ++i)
81        if (b[i] < -eps && idx[i] < idx[x])
82          x = i;
83      if (!x)
84        break;
85
86      for (int i = 1; i <= n; ++i)
87        if (A[x][i] < -eps && idy[i] < idy[y])
88          y = i;
89      if (!y) {
90        flag = INFEASIBLE;
91        break;
92      }
```

```
93
94      pivot(x, y);
95    }
96  }
97
98  void simplex() {
99    for (;;) {
100     int x = 0, y = 0;
101     for (int i = 1; i <= n; ++i)
102       if (A[0][i] > eps && idy[i] < idy[y])
103         y = i;
104     if (!y)
105       break;
106
107     for (int i = 1; i <= m; ++i)
108       if (A[i][y] > eps) {
109         if (!x)
110           x = i;
111         else {
112           double delta = b[i] / A[i][y] - b[x] / A[x][y];
113           if (delta < -eps)
114             x = i;
115           else if (delta < eps && idx[i] < idx[x])
116             x = i;
117         }
118       }
119     if (!x) {
120       flag = UNBOUNDED;
121       break;
122     }
123
124     pivot(x, y);
125   }
126   z = -b[0];
127 }
128
129 void work() {
130   init();
131   if (flag == OK)
132     simplex();
133   if (flag == OK) {
134     for (int i = 1; i <= n; ++i) {
135       x[i] = 0;
136       for (int j = 1; j <= m; ++j)
137         if (idx[j] == i) {
138           x[i] = b[j];
139           break;
140         }
141     }
142   }
143 }
144
145 void DEBUG() {
146   cerr << fixed << setprecision(3);
147   cerr << "Simplex Debug: \n";
148   for (int i = 1; i <= m; ++i) {
149     for (int j = 1; j <= n; ++j) {
150       cerr << A[i][j] << " ";
151     }
152     cerr << "\n";
153   }
154   for (int i = 1; i <= n; ++i)
155     cerr << x[i] << " ";
156   cerr << endl;
157   cerr << "Z = " << z << endl;
```

```
  }
};
```

3.26 SimpsonIntegral

```
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));
}
}  // namespace SimpsonIntegral
```

4 other

4.1 BFPRT

```
/**
 * 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>());
}
};  // namespace BFPRT
```

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()
```

```cpp
#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) ? (NE
}

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;
  while (!isdigit(c)) {
    f |= (c == '-');
    c = nc();
  }
  while (isdigit(c)) {
    x = x * 10.0 + (c ^ 48);
    c = nc();
  }
  if (c == '.') {
    double temp = 1;
    c = nc();
    while (isdigit(c)) {
      temp = temp / 10.0;
      x = x + temp * (c ^ 48);
      c = nc();
    }
  }
  if (f)
    x = -x;
  return NEOF;
}

template <typename T, typename... T2>
inline bool read(T& x, T2&... rest) {
  read(x);
  return read(rest...);
}

// inline bool need(char c) { return (c == '-') || (c == '>') || (c == '<');}
// inline bool need(char c) { return isdowncase(c) || isupcase(c) || isdigit(c) || c == '.' || c == '#'; }
inline bool need(char c) {
  return isdowncase(c) || isupcase(c) || isdigit(c);
}

inline bool read_str(char* a) {
  while ((*a = nc()) && need(*a) && NEOF)
    ++a;
  *a = '\0';
  return NEOF;
}

template <typename T>
inline void print(T x) {
  if (x < 0)
    putchar('-'), x = -x;
  if (x == 0) {
    putchar('0');
    return;
  }
  __top = 0;
  while (x) {
    __stk[++__top] = x % 10 + '0';
    x /= 10;
  }
  while (__top) {
    putchar(__stk[__top]);
```

```
    --__top;
  }
}

template <typename First, typename... Rest>
inline void print(First f, Rest... r) {
  print(f);
  putchar(' ');
  print(r...);
}

template <typename T>
inline void println(T x) {
  print(x);
  putchar('\n');
}

template <typename First, typename... Rest>
inline void println(First f, Rest... r) {
  print(f);
  putchar(' ');
  println(r...);
}

template <typename T>
inline void _dbg(const char* format, T value) {
  cerr << format << '=' << value << endl;
}

template <typename First, typename... Rest>
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
}  // namespace Backlight

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

```cpp
#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...));
}
```

```
64
65 template <typename T>
66 void updMin(T& a, T b) {
67   if (a > b)
68     a = b;
69 }
70
71 template <typename T>
72 void updMax(T& a, T b) {
73   if (a < b)
74     a = b;
75 }
76
77 inline char nc() {
78   return __p1 == __p2 && NEOF && (__p2 = (__p1 = __buf) + fread(__buf, 1, __BUFFER_SIZE__, stdin), __p1 == __p2) ? (NE
79 }
80
81 template <typename T>
82 inline bool read(T& x) {
83   char c = nc();
84   bool f = 0;
85   x = 0;
86   while (!isdigit(c))
87     c == '-' && (f = 1), c = nc();
88   while (isdigit(c))
89     x = (x << 3) + (x << 1) + (c ^ 48), c = nc();
90   if (f)
91     x = -x;
92   return NEOF;
93 }
94
95 inline bool read_db(double& x) {
96   bool f = 0;
97   char c = nc();
98   x = 0;
99   while (!isdigit(c)) {
100    f |= (c == '-');
101    c = nc();
102  }
103  while (isdigit(c)) {
104    x = x * 10.0 + (c ^ 48);
105    c = nc();
106  }
107  if (c == '.') {
108    double temp = 1;
109    c = nc();
110    while (isdigit(c)) {
111      temp = temp / 10.0;
112      x = x + temp * (c ^ 48);
113      c = nc();
114    }
115  }
116  if (f)
117    x = -x;
118  return NEOF;
119 }
120
121 template <typename T, typename... T2>
122 inline bool read(T& x, T2&... rest) {
123   read(x);
124   return read(rest...);
125 }
126
127 // inline bool need(char c) { return (c == '-') || (c == '>') || (c == '<');}
128 // inline bool need(char c) { return isdowncase(c) || isupcase(c) || isdigit(c) || c == '.' || c == '#'; }
```

```
inline bool need(char c) {
  return isdowncase(c) || isupcase(c) || isdigit(c);
}

inline bool read_str(char* a) {
  while ((*a = nc()) && need(*a) && NEOF)
    ++a;
  *a = '\0';
  return NEOF;
}

template <typename T>
inline void print(T x) {
  if (x < 0)
    putchar('-'), x = -x;
  if (x == 0) {
    putchar('0');
    return;
  }
  __top = 0;
  while (x) {
    __stk[++__top] = x % 10 + '0';
    x /= 10;
  }
  while (__top) {
    putchar(__stk[__top]);
    --__top;
  }
}

template <typename First, typename... Rest>
inline void print(First f, Rest... r) {
  print(f);
  putchar(' ');
  print(r...);
}

template <typename T>
inline void println(T x) {
  print(x);
  putchar('\n');
}

template <typename First, typename... Rest>
inline void println(First f, Rest... r) {
  print(f);
  putchar(' ');
  println(r...);
}

template <typename T>
inline void _dbg(const char* format, T value) {
  cerr << format << '=' << value << endl;
}

template <typename First, typename... Rest>
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) {
```

```cpp
  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
}  // namespace Backlight

/************     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;
```

```cpp

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

```java
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]));
}

inline int newnode() {
  ++tot;
  fail[tot] = 0;
  len[tot] = 0;
  memset(next[tot], 0, sizeof(next[tot]));
  return tot;
}

void insertTrie(const T* s, int n) {
  int p = 0, c;
  for (int i = 0; i < n; ++i) {
    c = idx(s[i]);
    if (!next[p][c])
      next[p][c] = newnode();
    p = next[p][c];
  }
}

int extendSAM(int last, int c) {
  int cur = next[last][c];
  if (len[cur])
    return cur;
  len[cur] = len[last] + 1;

  int p = fail[last];
  while (p != -1) {
    if (!next[p][c])
      next[p][c] = cur;
    else
      break;
    p = fail[p];
  }

  if (p == -1) {
    fail[cur] = 0;
    return cur;
  }

  int q = next[p][c];
```

```
  if (len[p] + 1 == len[q]) {
    fail[cur] = q;
    return cur;
  }

  int clone = newnode();
  for (int i = 0; i < __M; ++i)
    next[clone][i] = len[next[q][i]] ? next[q][i] : 0;

  len[clone] = len[p] + 1;
  while (p != -1 && next[p][c] == q) {
    next[p][c] = clone;
    p = fail[p];
  }
  fail[clone] = fail[q];
  fail[cur] = clone;
  fail[q] = clone;
  return cur;
}

void build() {
  queue<pair<int, int>> q;
  for (int i = 0; i < __M; ++i)
    if (next[0][i])
      q.push(make_pair(0, i));

  while (!q.empty()) {
    pair<int, int> u = q.front();
    q.pop();
    int last = extendSAM(u.first, u.second);
    for (int i = 0; i < __M; ++i)
      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;
}
}  // namespace GSAM
```

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];
}
}  // namespace KMP
```

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;
}
}  // namespace Manacher
```

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;
  }
}
}  // namespace SA
```

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;
    }
}
}  // namespace SAIS
```

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;
}
}  // namespace Hash
```

5.11 SuffixBST

```
/**
 * 1. 在当前字符串的后面插入字符
 * 2. 在当前字符串的后面删除字符
 * 3. 询问字符串 t 作为连续子串在当前字符串中出现了几次
 * */
#include <bits/stdc++.h>
using namespace std;

const int N = 8e5 + 5;
const double INF = 1e18;

void decode(char* s, int len, int mask) {
  for (int i = 0; i < len; ++i) {
    mask = (mask * 131 + i) % len;
    swap(s[i], s[mask]);
  }
}

int q, n, na;
char a[N], t[N];
```

```cpp

// SuffixBST(SGT Ver)

// 顺序加入，查询时将询问串翻转
// 以 i 结束的前缀，对应节点的编号为 i
// 注意：不能写懒惰删除，否则可能会破坏树的结构
const double alpha = 0.75;
int root;
int sz[N], L[N], R[N];
double tag[N];
int buffer_size, buffer[N];

bool cmp(int x, int y) {
  if (t[x] != t[y])
    return t[x] < t[y];
  return tag[x - 1] < tag[y - 1];
}

void init() {
  root = 0;
}

void new_node(int& rt, int p, double lv, double rv) {
  rt = p;
  sz[rt] = 1;
  tag[rt] = (lv + rv) / 2;
  L[rt] = R[rt] = 0;
}

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

bool balance(int rt) {
  return alpha * sz[rt] > max(sz[L[rt]], sz[R[rt]]);
}

void flatten(int rt) {
  if (!rt)
    return;
  flatten(L[rt]);
  buffer[++buffer_size] = rt;
  flatten(R[rt]);
}

void build(int& rt, int l, int r, double lv, double rv) {
  if (l > r) {
    rt = 0;
    return;
  }
  int mid = (l + r) >> 1;
  double mv = (lv + rv) / 2;

  rt = buffer[mid];
  tag[rt] = mv;
  build(L[rt], l, mid - 1, lv, mv);
  build(R[rt], mid + 1, r, mv, rv);
  push_up(rt);
}

void rebuild(int& rt, double lv, double rv) {
  buffer_size = 0;
  flatten(rt);
```

```
  build(rt, 1, buffer_size, lv, rv);
}

void insert(int& rt, int p, double lv, double rv) {
  if (!rt) {
    new_node(rt, p, lv, rv);
    return;
  }

  if (cmp(p, rt))
    insert(L[rt], p, lv, tag[rt]);
  else
    insert(R[rt], p, tag[rt], rv);

  push_up(rt);
  if (!balance(rt))
    rebuild(rt, lv, rv);
}

void remove(int& rt, int p, double lv, double rv) {
  if (!rt)
    return;

  if (rt == p) {
    if (!L[rt] || !R[rt]) {
      rt = (L[rt] | R[rt]);
    } else {
      // 找到 rt 的前驱来替换 rt
      int nrt = L[rt], fa = rt;
      while (R[nrt]) {
        fa = nrt;
        sz[fa]--;
        nrt = R[nrt];
      }
      if (fa == rt) {
        R[nrt] = R[rt];
      } else {
        L[nrt] = L[rt];
        R[nrt] = R[rt];
        R[fa] = 0;
      }
      rt = nrt;
      tag[rt] = (lv + rv) / 2;
    }
  } else {
    double mv = (lv + rv) / 2;
    if (cmp(p, rt))
      remove(L[rt], p, lv, mv);
    else
      remove(R[rt], p, mv, rv);
  }

  push_up(rt);
  if (!balance(rt))
    rebuild(rt, lv, rv);
}

bool cmp1(char* s, int len, int p) {
  for (int i = 1; i <= len; ++i, --p) {
    if (s[i] < t[p])
      return true;
    if (s[i] > t[p])
      return false;
  }
}
```

```
151
152 int query(int rt, char* s, int len) {
153   if (!rt)
154     return 0;
155   if (cmp1(s, len, rt))
156     return query(L[rt], s, len);
157   else
158     return sz[L[rt]] + 1 + query(R[rt], s, len);
159 }
160
161 void solve(int Case) {
162   n = 0;
163   scanf("%d", &q);
164   init();
165
166   scanf("%s", a + 1);
167   na = strlen(a + 1);
168   for (int i = 1; i <= na; ++i) {
169     t[++n] = a[i];
170     insert(root, n, 0, INF);
171   }
172
173   int mask = 0;
174   char op[10];
175   for (int i = 1; i <= q; ++i) {
176     scanf("%s", op);
177     if (op[0] == 'A') {
178       scanf("%s", a + 1);
179       na = strlen(a + 1);
180       decode(a + 1, na, mask);
181
182       for (int i = 1; i <= na; ++i) {
183         t[++n] = a[i];
184         insert(root, n, 0, INF);
185       }
186     } else if (op[0] == 'D') {
187       int x;
188       scanf("%d", &x);
189       while (x) {
190         remove(root, n, 0, INF);
191         --n;
192         --x;
193       }
194     } else if (op[0] == 'Q') {
195       scanf("%s", a + 1);
196       na = strlen(a + 1);
197       decode(a + 1, na, mask);
198
199       reverse(a + 1, a + 1 + na);
200
201       a[na + 1] = 'Z' + 1;
202       a[na + 2] = 0;
203       int ans = query(root, a, na + 1);
204
205       --a[na];
206       ans -= query(root, a, na + 1);
207
208       printf("%d\n", ans);
209       mask ^= ans;
210     }
211   }
212 }
213
214 int main() {
215   int T = 1;
```

```
  for (int i = 1; i <= T; ++i)
    solve(i);
  return 0;
}
```

5.12 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];
}
}  // namespace Trie
```

5.13 ZAlgorithm

```
namespace ZAlgorithm {
// 1-based

// z_i = LCP(s, s[i..n])
```

```cpp
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;
  }
}
}  // namespace ZAlgorithm
```