

# Backlight's Code Template

Backlight @ CSU

2021 年 10 月 16 日

# 目录

<b>1</b>	<b>ds</b>	<b>1</b>
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	LeftstTree	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
<b>2</b>	<b>graph</b>	<b>49</b>
2.1	BCC-Edge	49
2.2	BCC-Point	51
2.3	BiGraphMatch	52
2.4	BiWrappMatch	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	Wrapp	98
2.24	WrappMatch	99
<b>3</b>	<b>math</b>	<b>115</b>
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
<b>4</b>	<b>other</b>	<b>181</b>
4.1	BFPRT	181
4.2	cpp-header	182
4.3	debug	187
4.4	java-header	191
4.5	SimulateAnneal	192
<b>5</b>	<b>string</b>	<b>193</b>
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

---

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 #define FOR(i, L, r) for (int i = L; i <= r; ++i)
5 #define ROF(i, r, L) for (int i = r; i >= L; --i)
6 #define REP(i, L, r) for (int i = L; i < r; ++i)
7 #define PER(i, r, L) for (int i = r - 1; i >= L; --i)
8
9 const int N = 1e5 + 5;
10 using ll = long long;
11
12 template <typename T>
13 struct AVLTree {
14     struct node {
15         T v;
16         int sz, h, cnt;
17         node *l, *r;
18         explicit node(T _v)
19             : v(_v) {
20             sz = h = cnt = 1;
21             l = r = nullptr;
22         }
23     };
24     node* root = nullptr;
25     int get_size(node* p) {
26         return p ? p->sz : 0;
27     }
28
29     int get_height(node* p) {
30         return p ? p->h : 0;
31     }
32
33     void push_up(node* p) {
34         if (!p)
35             return;
36         p->sz = get_size(p->l) + p->cnt + get_size(p->r);
37         p->h = 1 + max(get_height(p->l), get_height(p->r));
38     }
39
40     void zig(node*& p) {
41         node* q = p;
42         q = p->r;
43         p->r = q->l;
44         q->l = p;
45         push_up(p);
46         push_up(q);
47         p = q;
48     }
49
50     void zag(node*& p) {
51         node* q = p->l;
52         p->l = q->r;
53         q->r = p;
54         push_up(p);
55         push_up(q);
56         p = q;
57     }
58
59     void maintain(node*& p) {
60         if (!p)
61             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);
127     }
```

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

```



```

36 //把 where 孩子分成两个节点，都作为 x 的孩子
37 void split(Node* x, int where) {
38     Node* z = new Node(x);
39     Node* y = x->child[where];
40     z->isLeaf = y->isLeaf;
41     memmove(z->values, y->values + BF, (BF - 1) * sizeof(value_type));
42     if (!y->isLeaf) {
43         memmove(z->child, y->child + BF, BF * sizeof(Node*));
44         for (int i = 0; i < BF; ++i)
45             z->child[i]->p = z;
46     }
47     z->keyNum = y->keyNum = BF - 1;
48     shiftBy(x->child + where + 1, x->keyNum - where, 1); //注意 child 本身 keyNum 多一个
49     x->child[where + 1] = z;
50     shiftBy(x->values + where, x->keyNum - where, 1);
51     new (x->values + where) value_type(y->values[BF - 1]);
52
53     y->size = calcSize(y), z->size = calcSize(z);
54     ++x->keyNum;
55 }
56 void insertEmpty(Node* x, const K& key) {
57     while (true) {
58         int i = lower_bound(x->values, x->values + x->keyNum, key, pairComp) - x->values;
59         if (i != x->keyNum && !(key < x->values[i].first)) //重复插入
60         {
61             ++x->cnt(i);
62             while (x)
63                 ++x->size, x = x->p;
64             return;
65         }
66         if (x->isLeaf) {
67             shiftBy(x->values + i, x->keyNum - i, 1);
68             x->values[i] = {key, 1};
69             ++x->keyNum;
70             while (x)
71                 ++x->size, x = x->p;
72             return;
73         }
74         if (x->child[i]->keyNum == 2 * BF - 1) {
75             split(x, i);
76             if (x->key(i) < key)
77                 ++i;
78             else if (!(key < x->key(i))) {
79                 ++x->cnt(i);
80                 while (x)
81                     ++x->size, x = x->p;
82                 return;
83             }
84         }
85         x = x->child[i];
86     }
87 }
88
89 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
90 {
91     Node *y = x->child[i], *z = x->child[i + 1];
92     y->keyNum = 2 * BF - 1;
93     y->values[BF - 1] = std::move(x->values[i]);
94     memmove(y->values + BF, z->values, (BF - 1) * sizeof(value_type));
95     if (!y->isLeaf) {
96         memmove(y->child + BF, z->child, BF * sizeof(Node*));
97         for (int j = BF; j <= 2 * BF - 1; ++j)
98             y->child[j]->p = y;
99     }
100     shiftBy(x->values + i + 1, x->keyNum - i - 1, -1);

```

```

101     shiftBy(x->child + i + 2, x->keyNum - i - 1, -1);
102
103     --x->keyNum;
104     y->size = calcSize(y);
105 }
106 void erase(Node* x, const K& key) {
107     int i = lower_bound(x->values, x->values + x->keyNum, key, pairComp) - x->values;
108     if (i != x->keyNum && !(key < x->values[i].first)) //找到 key 了
109     {
110         if (x->cnt(i) > 1) {
111             --x->cnt(i);
112             while (x)
113                 --x->size, x = x->p;
114             return;
115         }
116         if (x->isLeaf) //x 是叶节点, 直接删除
117         {
118             shiftBy(x->values + i + 1, --x->keyNum - i, -1); //需要移动的内存是 x->keyNum-i-1
119             while (x)
120                 --x->size, x = x->p;
121         } else {
122             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)
123             {
124                 Node* y = x->child[i];
125                 while (!y->isLeaf)
126                     y = y->child[y->keyNum]; //找前驱
127                 x->values[i] = y->values[y->keyNum - 1];
128                 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
129                 {
130                     y->cnt(y->keyNum - 1) = 1;
131                     while (y != x)
132                         y->size -= x->cnt(i) - 1, y = y->p;
133                 }
134
135                 erase(x->child[i], x->key(i));
136             } else if (x->child[i + 1]->keyNum >= BF) //后继所在孩子有足够的孩子
137             {
138                 Node* y = x->child[i + 1];
139                 while (!y->isLeaf)
140                     y = y->child[0]; //找后继
141                 x->values[i] = y->values[0];
142                 if (x->cnt(i) != 1) {
143                     y->cnt(0) = 1;
144                     while (y != x)
145                         y->size -= x->cnt(i) - 1, y = y->p;
146                 }
147
148                 erase(x->child[i + 1], x->key(i));
149             } 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
150             {
151                 merge(x, i);
152                 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
153                     root = x->child[i], root->p = nullptr;
154                 erase(x->child[i], key);
155             }
156         }
157     } 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
158     {
159         if (x->child[i]->keyNum == BF - 1) {
160             Node* y = x->child[i];
161             if (i >= 1 && x->child[i - 1]->keyNum >= BF) //左兄弟, 取走它的最大孩子
162             {
163                 //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
164                 //但是从左还是右借并不完全一样, 所以不能一概处理
165                 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)

```

```

231     ++i;
232     if (i != x->keyNum)
233         ret = x->values[i].first;
234     x = x->child[i];
235 }
236 return ret;
237 }
238 int prev(const K& key) {
239     Node* x = root;
240     int ret;
241     while (x) {
242         int i = lower_bound(x->values, x->values + x->keyNum, key, pairComp) - x->values;
243         if (i)
244             ret = x->values[i - 1].first;
245         x = x->child[i];
246     }
247     return ret;
248 }
249 int rank(const K& key) {
250     Node* x = root;
251     int ret = 0;
252     while (x) {
253         if (x->key(x->keyNum - 1) < key) {
254             ret += x->size - getSize(x->child[x->keyNum]);
255             x = x->child[x->keyNum];
256             continue;
257         }
258         for (int i = 0; i < x->keyNum; ++i) {
259             if (x->key(i) < key)
260                 ret += getSize(x->child[i]) + x->cnt(i);
261             else if (x->key(i) == key)
262                 return ret + getSize(x->child[i]) + 1;
263             else {
264                 x = x->child[i];
265                 break;
266             }
267         }
268     }
269     return ret;
270 }
271 int kth(int k) {
272     Node* x = root;
273     while (true) {
274         for (int i = 0; i <= x->keyNum; ++i) {
275             //const int csz = getSize(x->child[i]) + (i == x->keyNum ? 1 : x->cnt(i));
276             const int lb = getSize(x->child[i]) + 1, ub = getSize(x->child[i]) + (i == x->keyNum ? 1 : x->cnt(i));
277             if (k >= lb && k <= ub)
278                 return x->key(i);
279             if (k < lb) {
280                 x = x->child[i];
281                 break;
282             }
283             k -= ub;
284         }
285     }
286 }
287 };

```

### 1.3 CaptainMo

```

1 // Captain Mo
2 // 询问  $[l, r]$  内的元素是否互不相同
3 int Ans, ans[N];
4 int block_sz, block_id[N];

```

```

5 struct Query {
6     int l, r, id;
7     Query() {}
8     Query(int _l, int _r, int _id)
9         : l(_l), r(_r), id(_id) {}
10    bool operator<(const Query& q) const {
11        if (block_id[l] == block_id[q.l])
12            return block_id[l] & 1 ? r < q.r : r > q.r;
13        return block_id[l] < block_id[q.l];
14    }
15 } Q[N];
16
17 int n, q, a[N];
18
19 int cnt[N], ge2;
20 inline void add(int p) {
21     ++cnt[a[p]];
22     if (cnt[a[p]] == 2)
23         ++ge2;
24 }
25
26 inline void del(int p) {
27     if (cnt[a[p]] == 2)
28         --ge2;
29     --cnt[a[p]];
30 }
31
32 void CaptainMo() {
33     block_sz = sqrt(n);
34     for (int i = 1; i <= n; ++i)
35         block_id[i] = i / block_sz;
36     sort(Q + 1, Q + 1 + q);
37
38     int l = 1, r = 0;
39     ge2 = 0;
40     for (int i = 1; i <= q; ++i) {
41         while (r < Q[i].r)
42             ++r, add(r);
43         while (l < Q[i].l)
44             del(l), ++l;
45         while (l > Q[i].l)
46             --l, add(l);
47         while (r > Q[i].r)
48             del(r), --r;
49         ans[Q[i].id] = (ge2 == 0);
50     }
51 }

```

## 1.4 CDQ

```

1 // P3810
2 // 3-D Partial Order( $a_j \leq a_i$  and  $b_j \leq b_i$  and  $c_j \leq c_i$ )
3 #include <bits/stdc++.h>
4 using namespace std;
5 using ll = int64_t;
6
7 const int N = 1e5 + 5;
8 const int MAXV = 2e5 + 5;
9 int k, c[MAXV];
10 inline int lb(int x) {
11     return x & (-x);
12 }
13 void add(int x, int d) {
14     for (; x <= k; x += lb(x))

```

```
15     c[x] += d;
16 }
17 int getsum(int x) {
18     int ret = 0;
19     for (; x; x -= lb(x))
20         ret += c[x];
21     return ret;
22 }
23
24 int n, m;
25 struct node {
26     int a, b, c, w, ans;
27     void input() {
28         scanf("%d %d %d", &a, &b, &c);
29         w = ans = 0;
30     }
31 } a[N], b[N];
32
33 int cnt[N];
34 void CDQ(int l, int r) {
35     if (l == r)
36         return;
37     int mid = (l + r) >> 1;
38     CDQ(l, mid);
39     CDQ(mid + 1, r);
40
41     sort(a + l, a + 1 + mid, [](const node& x, const node& y) -> bool {
42         if (x.b != y.b)
43             return x.b < y.b;
44         return x.c < y.c;
45     });
46     sort(a + mid + 1, a + 1 + r, [](const node& x, const node& y) -> bool {
47         if (x.b != y.b)
48             return x.b < y.b;
49         return x.c < y.c;
50     });
51
52     int p1 = l, p2 = mid + 1;
53     for (; p2 <= r; ++p2) {
54         while (a[p1].b <= a[p2].b && p1 <= mid) {
55             add(a[p1].c, a[p1].w);
56             ++p1;
57         }
58         a[p2].ans += getsum(a[p2].c);
59     }
60
61     for (int i = l; i < p1; ++i)
62         add(a[i].c, -a[i].w);
63 }
64
65 int main() {
66     scanf("%d %d", &m, &k);
67     for (int i = 1; i <= m; ++i)
68         b[i].input();
69
70     sort(b + 1, b + 1 + m, [](const node& x, const node& y) -> bool {
71         if (x.a != y.a)
72             return x.a < y.a;
73         if (x.b != y.b)
74             return x.b < y.b;
75         return x.c < y.c;
76     });
77
78     int w = 0;
79     for (int i = 1; i <= m; ++i) {
```

```

80     ++w;
81     if (b[i].a != b[i + 1].a || b[i].b != b[i + 1].b || b[i].c != b[i + 1].c) {
82         ++n;
83         a[n] = b[i];
84         a[n].w = w;
85         w = 0;
86     }
87 }
88 CDQ(1, n);
89 for (int i = 1; i <= n; ++i)
90     if (a[i].ans + a[i].w - 1 < m)
91         cnt[a[i].ans + a[i].w - 1] += a[i].w;
92 for (int i = 0; i < m; ++i)
93     printf("%d\n", cnt[i]);
94 return 0;
95 }

```

---

## 1.5 FenwickTree

```

1  template <typename T>
2  struct FenwickTree {
3      int n;
4      vector<T> c;
5      FenwickTree(int _n)
6          : n(_n), c(n + 1) {}
7      inline int lb(int x) { return x & -x; }
8      void add(int x, T d) {
9          for (; x <= n; x += lb(x))
10             c[x] += d;
11     }
12     T getsum(int x) {
13         T r = 0;
14         for (; x; x -= lb(x))
15             r += c[x];
16         return r;
17     }
18     T getsum(int l, int r) { return getsum(r) - getsum(l - 1); }
19     T kth(int k) {
20         T ans = 0, cnt = 0;
21         for (int i = __lg(n) + 1; i >= 0; --i) {
22             ans += (1LL << i);
23             if (ans >= n || cnt + c[ans] >= k)
24                 ans -= (1LL << i);
25             else
26                 cnt += c[ans];
27         }
28         return ans + 1;
29     }
30 };

```

---

## 1.6 KDTree

```

1  // KDTree
2  // 平面最近点对
3  template <typename T, int K = 2>
4  struct KDTree {
5      using node = array<double, K>;
6      int n;
7      node p[N], ma[N], mi[N];
8      double L[N], R[N], D[N], U[N];
9      int sd[N], lc[N], rc[N];
10

```

```

11  KDTree(int _n)
12      : n(_n) {
13  }
14
15  double dist(const node& nd1, const node& nd2) {
16      double res = 0;
17      for (int j = 0; j < K; ++j) {
18          res += (nd1[j] - nd2[j]) * (nd1[j] - nd2[j]);
19      }
20      return res;
21  }
22
23  double dist(int x, int y) {
24      return dist(p[x], p[y]);
25  }
26
27  double cost(int x, int y) {
28      double res = 0;
29      for (int j = 0; j < K; ++j) {
30          if (mi[y][j] > p[x][j])
31              res += (mi[y][j] - p[x][j]) * (mi[y][j] - p[x][j]);
32          if (ma[y][j] < p[x][j])
33              res += (ma[y][j] - p[x][j]) * (ma[y][j] - p[x][j]);
34      }
35      return res;
36  }
37
38  struct cmp {
39      int s;
40      cmp(int _s)
41          : s(_s) {
42      }
43      bool operator()(const node& nd1, const node& nd2) const {
44          return nd1[s] < nd2[s];
45      }
46  };
47
48  void maintain(int x) {
49      ma[x] = mi[x] = p[x];
50      if (lc[x]) {
51          for (int j = 0; j < K; ++j) {
52              ma[x][j] = max(ma[x][j], ma[lc[x]][j]);
53              mi[x][j] = min(mi[x][j], mi[lc[x]][j]);
54          }
55      }
56      if (rc[x]) {
57          for (int j = 0; j < K; ++j) {
58              ma[x][j] = max(ma[x][j], ma[rc[x]][j]);
59              mi[x][j] = min(mi[x][j], mi[rc[x]][j]);
60          }
61      }
62  }
63
64  int build(int l, int r) {
65      if (l >= r)
66          return 0;
67      int mid = (l + r) >> 1;
68
69      array<double, K> avg;
70      for (int i = l; i <= r; ++i)
71          for (int j = 0; j < K; ++j)
72              avg[j] += p[i][j];
73      for (int j = 0; j < K; ++j)
74          avg[j] /= (r - l + 1);
75

```



```
76     array<double, K> var;
77     for (int i = 1; i <= r; ++i)
78         for (int j = 0; j < K; ++j)
79             var[j] += (p[i][j] - avg[j]) * (p[i][j] - avg[j]);
80
81     sd[mid] = 0;
82     for (int j = 0; j < K; ++j)
83         if (var[j] > var[sd[mid]])
84             sd[mid] = j;
85
86     nth_element(p + 1, p + mid, p + r + 1, cmp(sd[mid]));
87
88     lc[mid] = build(1, mid - 1);
89     rc[mid] = build(mid + 1, r);
90
91     maintain(mid);
92
93     return mid;
94 }
95
96 double min_dist;
97
98 void query(int l, int r, int x) {
99     if (l > r)
100         return;
101     int mid = (l + r) >> 1;
102     if (mid != x)
103         min_dist = min(min_dist, dist(x, mid));
104     if (l == r)
105         return;
106
107     double dl = cost(x, lc[mid]);
108     double dr = cost(x, rc[mid]);
109
110     if (dl < min_dist && dr < min_dist) {
111         if (dl < dr) {
112             query(1, mid - 1, x);
113             if (dr < min_dist)
114                 query(mid + 1, r, x);
115         } else {
116             query(mid + 1, r, x);
117             if (dl < min_dist)
118                 query(1, mid - 1, x);
119         }
120     } else {
121         if (dl < min_dist)
122             query(1, mid - 1, x);
123         if (dr < min_dist)
124             query(mid + 1, r, x);
125     }
126 }
127
128 double getMindis() {
129     min_dist = 2e18;
130     for (int i = 1; i <= n; ++i)
131         query(1, n, i);
132     min_dist = sqrt(min_dist);
133     return min_dist;
134 }
135 };
```

## 1.7 LCT

---

```

1  template <typename T>
2  struct LinkCutTree {
3      #define ls ch[x][0]
4      #define rs ch[x][1]
5      #define SIZE 100005
6
7      int tot, sz[SIZE], rev[SIZE], ch[SIZE][2], fa[SIZE];
8      T v[SIZE], sum[SIZE];
9
10     LinkCutTree() { tot = 0; }
11
12     inline void init() { tot = 0; }
13
14     inline void clear(int x) {
15         ch[x][0] = ch[x][1] = fa[x] = sz[x] = rev[x] = sum[x] = v[x] = 0;
16     }
17
18     inline int get(int x) { return ch[fa[x]][1] == x; }
19
20     inline int isroot(int x) { return ch[fa[x]][0] != x && ch[fa[x]][1] != x; }
21
22     inline int newnode(T val) {
23         ++tot;
24         sz[tot] = 1;
25         ch[tot][0] = ch[tot][1] = fa[tot] = rev[tot] = 0;
26         sum[tot] = v[tot] = val;
27         return tot;
28     }
29
30     inline void reverse(int x) {
31         swap(ls, rs);
32         rev[x] ^= 1;
33     }
34
35     inline void push_up(int x) {
36         sz[x] = sz[ls] + 1 + sz[rs];
37         sum[x] = sum[ls] ^ v[x] ^ sum[rs];
38     }
39
40     inline void push_down(int x) {
41         if (rev[x]) {
42             reverse(ls);
43             reverse(rs);
44             rev[x] = 0;
45         }
46     }
47
48     inline void update(int x) {
49         if (!isroot(x))
50             update(fa[x]);
51         push_down(x);
52     }
53
54     inline void rotate(int x) {
55         int f = fa[x], g = fa[f], i = get(x);
56         if (!isroot(f))
57             ch[g][get(f)] = x;
58         fa[x] = g;
59         ch[f][i] = ch[x][i ^ 1];
60         fa[ch[f][i]] = f;
61         ch[x][i ^ 1] = f;
62         fa[f] = x;
63         push_up(f);

```

```

64     push_up(x);
65 }
66
67 inline void splay(int x) {
68     update(x);
69     for (; !isroot(x); rotate(x))
70         if (!isroot(fa[x]))
71             rotate(get(fa[x]) == get(x) ? fa[x] : x);
72 }
73
74 inline void access(int x) {
75     for (int y = 0; x; y = x, x = fa[x])
76         splay(x), rs = y, push_up(x);
77 }
78
79 inline void makeroot(int x) {
80     access(x);
81     splay(x);
82     reverse(x);
83 }
84
85 inline int findroot(int x) {
86     access(x);
87     splay(x);
88     while (ls)
89         push_down(x), x = ls;
90     return x;
91 }
92
93 inline void link(int x, int y) {
94     makeroot(x);
95     if (findroot(y) != x)
96         fa[x] = y;
97 }
98
99 inline void cut(int x, int y) {
100     makeroot(x);
101     if (findroot(y) == x && fa[x] == y && ch[y][0] == x && !ch[y][1]) {
102         fa[x] = ch[y][0] = 0;
103         push_up(y);
104     }
105 }
106
107 inline void split(int x, int y) {
108     makeroot(x);
109     access(y);
110     splay(y);
111 }
112
113 // x--y 路径上节点权值之和
114 inline int query(int x, int y) {
115     split(x, y);
116     return sum[y];
117 }
118 };
119
120 void solve(int Case) {
121     /* write code here */
122     /* gl & hf */
123     int n, m;
124     rd(n, m);
125     VI a(n + 1);
126     FOR(i, 1, n)
127         rd(a[i]);
128     LinkCutTree<int> t;

```

```

129 FOR(i, 1, n)
130 t.newnode(a[i]);
131
132 int op, x, y;
133 FOR(_, 1, m) {
134     rd(op, x, y);
135     debug(op, x, y);
136     if (op == 0) {
137         pln(t.query(x, y));
138     } else if (op == 1) {
139         t.link(x, y);
140     } else if (op == 2) {
141         t.cut(x, y);
142     } else {
143         t.v[x] = y;
144         t.makeroot(x);
145     }
146 }
147 }

```

## 1.8 LeftistTree

```

1 template <typename V>
2 struct LeftistForest {
3     struct LeftistTree {
4         V v;
5         int dist;
6         int l, r, rt;
7     } t[N];
8     LeftistTree& operator[](int x) { return t[x]; }
9     void init(int n, V* a) {
10         FOR(i, 1, n) {
11             t[i].v = a[i];
12             t[i].l = t[i].r = t[i].dist = 0;
13             t[i].rt = i;
14         }
15     }
16     int find(int x) { return t[x].rt == x ? x : t[x].rt = find(t[x].rt); }
17     int merge(int x, int y) {
18         if (!x)
19             return y;
20         if (!y)
21             return x;
22         if (t[x].v > t[y].v)
23             swap(x, y); // 小根堆
24         t[x].r = merge(t[x].r, y);
25         t[t[x].r].rt = x;
26         if (t[t[x].l].dist < t[t[x].r].dist)
27             swap(t[x].l, t[x].r);
28         if (!t[x].r)
29             t[x].dist = 0;
30         else
31             t[x].dist = t[t[x].r].dist + 1;
32         return x;
33     }
34     V top(int x) {
35         if (t[x].v == -1)
36             return -1;
37         x = find(x);
38         return t[x].v;
39     }
40     void pop(int x) {
41         if (t[x].v == -1)
42             return;

```

```

43     x = find(x);
44     t[t[x].l].rt = t[x].l;
45     t[t[x].r].rt = t[x].r;
46     t[x].rt = merge(t[x].l, t[x].r);
47     t[x].v = -1;
48 }
49 };
50
51 int n, m, a[N];
52 void solve(int Case) {
53     rd(n, m);
54     FOR(i, 1, n)
55         rd(a[i]);
56     LeftistForest<int> T;
57     T.init(n, a);
58
59     int op, x, y;
60     FOR(_, 1, m) {
61         rd(op);
62         debug(op);
63         if (op == 1) {
64             rd(x, y);
65             if (T[x].v == -1 || T[y].v == -1)
66                 continue;
67             x = T.find(x);
68             y = T.find(y);
69             if (x == y)
70                 continue;
71             T[x].rt = T[y].rt = T.merge(x, y);
72         } else {
73             rd(x);
74             pln(T.top(x));
75             T.pop(x);
76         }
77     }
78 }

```

## 1.9 PersistentSegmentTree

```

1 struct PersistentSegmentTree {
2     // SIZE = N log N
3     #define SIZE 200005 * 20
4
5     int tot;
6     int c[SIZE];
7     int L[SIZE], R[SIZE];
8
9     PersistentSegmentTree() { tot = 0; }
10
11     int update(int rt, int l, int r, int p, int d) {
12         int nrt = ++tot;
13         L[nrt] = L[rt];
14         R[nrt] = R[rt];
15         c[nrt] = c[rt] + d;
16
17         if (l != r) {
18             int mid = (l + r) >> 1;
19             if (p <= mid)
20                 L[nrt] = update(L[rt], l, mid, p, d);
21             else
22                 R[nrt] = update(R[rt], mid + 1, r, p, d);
23         }
24
25         return nrt;

```

```

26 }
27
28 // 区间第 k 小
29 int query(int u, int v, int l, int r, int k) {
30     if (l == r)
31         return l;
32     int left_size = c[L[v]] - c[L[u]];
33     int mid = (l + r) >> 1;
34     if (k <= left_size)
35         return query(L[u], L[v], l, mid, k);
36     return query(R[u], R[v], mid + 1, r, k - left_size);
37 }
38 };

```

---

## 1.10 rbtrees-1

```

1  // #define __REDBLACK_DEBUG
2  template <typename T>
3  class rbtree {
4      #define bro(x) (((x)->ftr->lc == (x)) ? ((x)->ftr->rc) : ((x)->ftr->lc))
5      #define islc(x) ((x) != NULL && (x)->ftr->lc == (x))
6      #define isrc(x) ((x) != NULL && (x)->ftr->rc == (x))
7      private:
8          struct Node;
9
10         Node* _root;
11         Node* _hot;
12
13         void init(T);
14         void checkconnect(Node*);
15         void connect34(Node*, Node*, Node*, Node*, Node*, Node*, Node*);
16         void SolveDoubleRed(Node*);
17         void SolveDoubleBlack(Node*);
18         Node* find(T, const int);
19         Node* rfind(T, const int);
20         Node* findkth(int, Node*);
21         int find_rank(T, Node*);
22     #ifdef __REDBLACK_DEBUG
23         void previs(Node*, int);
24         void invis(Node*, int);
25         void postvis(Node*, int);
26     #endif
27
28     public:
29         struct iterator;
30
31         rbtree()
32             : _root(NULL), _hot(NULL) {
33         }
34
35         int get_rank(T);
36         iterator insert(T);
37         bool remove(T);
38         int size();
39         iterator kth(int);
40         iterator lower_bound(T);
41         iterator upper_bound(T);
42     #ifdef __REDBLACK_DEBUG
43         void vis();
44         void correctlyconnected();
45     #endif
46 };
47
48 template <typename T>

```

```

49 struct rbtree<T>::Node {
50     T val;
51     bool RB;  ///true : Red ; false : Black .
52     Node* ftr;
53     Node* lc;
54     Node* rc;
55     int s;
56
57     Node(T v = T(), bool RB = true, Node* f = NULL, Node* l = NULL, Node* r = NULL, int ss = 1)
58         : val(v), RB(RB), ftr(f), lc(l), rc(r), s(ss) {
59     }
60
61     Node* succ() {
62         Node* ptn = rc;
63         while (ptn->lc != NULL) {
64             --(ptn->s);
65             ptn = ptn->lc;
66         }
67         return ptn;
68     }
69
70     Node* left_node() {
71         Node* ptn = this;
72         if (!lc) {
73             while (ptn->ftr && ptn->ftr->lc == ptn)
74                 ptn = ptn->ftr;
75             ptn = ptn->ftr;
76         } else
77             while (ptn->lc)
78                 ptn = ptn->lc;
79         return ptn;
80     }
81
82     Node* right_node() {
83         Node* ptn = this;
84         if (!rc) {
85             while (ptn->ftr && ptn->ftr->rc == ptn)
86                 ptn = ptn->ftr;
87             ptn = ptn->ftr;
88         } else
89             while (ptn->rc)
90                 ptn = ptn->rc;
91         return ptn;
92     }
93
94     void maintain() {
95         s = 1;
96         if (lc)
97             s += lc->s;
98         if (rc)
99             s += rc->s;
100     }
101 };
102
103 template <typename T>
104 void rbtree<T>::connect34(Node* nroot, Node* nlc, Node* nrc, Node* ntree1, Node* ntree2, Node* ntree3, Node* ntree4) {
105     nlc->lc = ntree1;
106     if (ntree1 != NULL)
107         ntree1->ftr = nlc;
108     nlc->rc = ntree2;
109     if (ntree2 != NULL)
110         ntree2->ftr = nlc;
111     nrc->lc = ntree3;
112     if (ntree3 != NULL)
113         ntree3->ftr = nrc;

```

```

114 nrc->rc = ntree4;
115 if (ntree4 != NULL)
116     ntree4->ftr = nrc;
117 nroot->lc = nlc;
118 nlc->ftr = nroot;
119 nroot->rc = nrc;
120 nrc->ftr = nroot;
121 nlc->maintain();
122 nrc->maintain();
123 nroot->maintain();
124 }
125
126 #ifdef __REDBLACK_DEBUG
127
128 int blackheight(0);
129
130 template <typename T>
131 void rbtree<T>::previs(Node* ptn, int cnt) {
132     if (ptn == NULL) {
133         if (blackheight == -1)
134             blackheight = cnt;
135         assert(blackheight == cnt);
136         return;
137     }
138     printf("%d %s %d \n", ptn->val, ptn->RBc ? "Red" : "Black", ptn->s);
139     if (!(ptn->RBc))
140         ++cnt;
141     previs(ptn->lc, cnt);
142     previs(ptn->rc, cnt);
143 }
144
145 template <typename T>
146 void rbtree<T>::invis(Node* ptn, int cnt) {
147     if (ptn == NULL) {
148         if (blackheight == -1)
149             blackheight = cnt;
150         assert(blackheight == cnt);
151         return;
152     }
153     if (!(ptn->RBc))
154         ++cnt;
155     invis(ptn->lc, cnt);
156     printf("%d %s %d \n", ptn->val, ptn->RBc ? "Red" : "Black", ptn->s);
157     invis(ptn->rc, cnt);
158 }
159
160 template <typename T>
161 void rbtree<T>::postvis(Node* ptn, int cnt) {
162     if (ptn == NULL) {
163         if (blackheight == -1)
164             blackheight = cnt;
165         assert(blackheight == cnt);
166         return;
167     }
168     if (!(ptn->RBc))
169         ++cnt;
170     postvis(ptn->lc, cnt);
171     postvis(ptn->rc, cnt);
172     printf("%d %s %d \n", ptn->val, ptn->RBc ? "Red" : "Black", ptn->s);
173 }
174
175 template <typename T>
176 void rbtree<T>::vis() {
177     printf("BlackHeight:\t%d\n", blackheight);
178     printf("-----pre-vis-----\n");

```



```

179     previs(_root, 0);
180     printf("-----in-vis-----\n");
181     invis(_root, 0);
182     printf("-----post-vis-----\n");
183     postvis(_root, 0);
184 }
185
186 template <typename T>
187 void rbtree<T>::checkconnect(Node* ptn) {
188     if (!ptn)
189         return;
190     assert(ptn->s > 0);
191     if (ptn->lc && ptn->lc->ftr != ptn) {
192         printf("Oops! %d has a lc %d, but it failed to point its ftr!\n", ptn->val, ptn->lc->val);
193     }
194     if (ptn->rc && ptn->rc->ftr != ptn) {
195         printf("Oops! %d has a rc %d, but it failed to point its ftr!\n", ptn->val, ptn->rc->val);
196     }
197     int sss = ptn->s;
198     if (ptn->lc)
199         sss -= ptn->lc->s;
200     if (ptn->rc)
201         sss -= ptn->rc->s;
202     if (sss - 1) {
203         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);
204     }
205     checkconnect(ptn->lc);
206     checkconnect(ptn->rc);
207 }
208
209 template <typename T>
210 void rbtree<T>::correctlyconnected() {
211     checkconnect(_root);
212 }
213
214 #endif
215
216 template <typename T>
217 void rbtree<T>::init(T v) {
218     _root = new Node(v, false, NULL, NULL, NULL, 1);
219 #ifdef __REDBLACK_DEBUG
220     ++blackheight;
221 #endif
222 }
223
224 template <typename T>
225 void rbtree<T>::SolveDoubleRed(Node* nn) {
226     while ((!nn->ftr) || nn->ftr->RBc) {
227         if (nn == _root) {
228             _root->RBc = false;
229 #ifdef __REDBLACK_DEBUG
230             ++blackheight;
231 #endif
232             return;
233         }
234         Node* pftr = nn->ftr;
235         if (!(pftr->RBc))
236             return; ////No double-red
237         Node* uncle = bro(nn->ftr);
238         Node* grdftr = nn->ftr->ftr;
239         if (uncle != NULL && uncle->RBc) { ////RR-2
240             grdftr->RBc = true;
241             uncle->RBc = false;
242             pftr->RBc = false;
243             nn = grdftr;

```

```

244 } else { ///RR-1
245     if (islc(pftr)) {
246         if (islc(nn)) {
247             pftr->ftr = grdftr->ftr;
248             if (grdftr == _root)
249                 _root = pftr;
250             else if (grdftr->ftr->lc == grdftr)
251                 grdftr->ftr->lc = pftr;
252             else
253                 grdftr->ftr->rc = pftr;
254             connect34(pftr, nn, grdftr, nn->lc, nn->rc, pftr->rc, uncle);
255             pftr->RBc = false;
256             grdftr->RBc = true;
257         } else {
258             nn->ftr = grdftr->ftr;
259             if (grdftr == _root)
260                 _root = nn;
261             else if (grdftr->ftr->lc == grdftr)
262                 grdftr->ftr->lc = nn;
263             else
264                 grdftr->ftr->rc = nn;
265             connect34(nn, pftr, grdftr, pftr->lc, nn->lc, nn->rc, uncle);
266             nn->RBc = false;
267             grdftr->RBc = true;
268         }
269     } else {
270         if (islc(nn)) {
271             nn->ftr = grdftr->ftr;
272             if (grdftr == _root)
273                 _root = nn;
274             else if (grdftr->ftr->lc == grdftr)
275                 grdftr->ftr->lc = nn;
276             else
277                 grdftr->ftr->rc = nn;
278             connect34(nn, grdftr, pftr, uncle, nn->lc, nn->rc, pftr->rc);
279             nn->RBc = false;
280             grdftr->RBc = true;
281         } else {
282             pftr->ftr = grdftr->ftr;
283             if (grdftr == _root)
284                 _root = pftr;
285             else if (grdftr->ftr->lc == grdftr)
286                 grdftr->ftr->lc = pftr;
287             else
288                 grdftr->ftr->rc = pftr;
289             connect34(pftr, grdftr, nn, uncle, pftr->lc, nn->lc, nn->rc);
290             pftr->RBc = false;
291             grdftr->RBc = true;
292         }
293     }
294     return;
295 }
296 }
297 }
298
299 template <typename T>
300 void rbtree<T>::SolveDoubleBlack(Node* nn) {
301     while (nn != _root) {
302         Node* pftr = nn->ftr;
303         Node* bthr = bro(nn);
304         if (bthr->RBc) { ///BB-1
305             bthr->RBc = false;
306             pftr->RBc = true;
307             if (_root == pftr)
308                 _root = bthr;

```

```

309     if (pftr->ftr) {
310         if (pftr->ftr->lc == pftr)
311             pftr->ftr->lc = bthr;
312         else
313             pftr->ftr->rc = bthr;
314     }
315     bthr->ftr = pftr->ftr;
316     if (islc(nn)) {
317         connect34(bthr, pftr, bthr->rc, nn, bthr->lc, bthr->rc->lc, bthr->rc->rc);
318     } else {
319         connect34(bthr, bthr->lc, pftr, bthr->lc->lc, bthr->lc->rc, bthr->rc, nn);
320     }
321     bthr = bro(nn);
322     pftr = nn->ftr;
323 }
324 if (bthr->lc && bthr->lc->RBc) { ///BB-3
325     bool oldRBc = pftr->RBc;
326     pftr->RBc = false;
327     if (pftr->lc == nn) {
328         if (pftr->ftr) {
329             if (pftr->ftr->lc == pftr)
330                 pftr->ftr->lc = bthr->lc;
331             else
332                 pftr->ftr->rc = bthr->lc;
333         }
334         bthr->lc->ftr = pftr->ftr;
335         if (_root == pftr)
336             _root = bthr->lc;
337         connect34(bthr->lc, pftr, bthr, pftr->lc, bthr->lc->lc, bthr->lc->rc, bthr->rc);
338     } else {
339         bthr->lc->RBc = false;
340         if (pftr->ftr) {
341             if (pftr->ftr->lc == pftr)
342                 pftr->ftr->lc = bthr;
343             else
344                 pftr->ftr->rc = bthr;
345         }
346         bthr->ftr = pftr->ftr;
347         if (_root == pftr)
348             _root = bthr;
349         connect34(bthr, bthr->lc, pftr, bthr->lc->lc, bthr->lc->rc, bthr->rc, pftr->rc);
350     }
351     pftr->ftr->RBc = oldRBc;
352     return;
353 } else if (bthr->rc && bthr->rc->RBc) { ///BB-3
354     bool oldRBc = pftr->RBc;
355     pftr->RBc = false;
356     if (pftr->lc == nn) {
357         bthr->rc->RBc = false;
358         if (pftr->ftr) {
359             if (pftr->ftr->lc == pftr)
360                 pftr->ftr->lc = bthr;
361             else
362                 pftr->ftr->rc = bthr;
363         }
364         bthr->ftr = pftr->ftr;
365         if (_root == pftr)
366             _root = bthr;
367         connect34(bthr, pftr, bthr->rc, pftr->lc, bthr->lc, bthr->rc->lc, bthr->rc->rc);
368     } else {
369         if (pftr->ftr) {
370             if (pftr->ftr->lc == pftr)
371                 pftr->ftr->lc = bthr->rc;
372             else
373                 pftr->ftr->rc = bthr->rc;

```

```

374     }
375     bthr->rc->ftr = pftr->ftr;
376     if (_root == pftr)
377         _root = bthr->rc;
378     connect34(bthr->rc, bthr, pftr, bthr->lc, bthr->rc->lc, bthr->rc->rc, pftr->rc);
379 }
380 pftr->ftr->RBc = oldRBc;
381 return;
382 }
383 if (pftr->RBc) { ///BB-2R
384     pftr->RBc = false;
385     bthr->RBc = true;
386     return;
387 } else { ///BB-2B
388     bthr->RBc = true;
389     nn = pftr;
390 }
391 }
392 #ifdef __REDBLACK_DEBUG
393     --blackheight;
394 #endif
395 }
396
397 template <typename T>
398 typename rbtree<T>::Node* rbtree<T>::findkth(int rank, Node* ptn) {
399     if (!ptn->lc) {
400         if (rank == 1) {
401             return ptn;
402         } else {
403             return findkth(rank - 1, ptn->rc);
404         }
405     } else {
406         if (ptn->lc->s == rank - 1)
407             return ptn;
408         else if (ptn->lc->s >= rank)
409             return findkth(rank, ptn->lc);
410         else
411             return findkth(rank - (ptn->lc->s) - 1, ptn->rc);
412     }
413 }
414
415 template <typename T>
416 int rbtree<T>::find_rank(T v, Node* ptn) {
417     if (!ptn)
418         return 1;
419     else if (ptn->val >= v)
420         return find_rank(v, ptn->lc);
421     else
422         return (1 + ((ptn->lc) ? (ptn->lc->s) : 0) + find_rank(v, ptn->rc));
423 }
424
425 template <typename T>
426 int rbtree<T>::get_rank(T v) {
427     return find_rank(v, _root);
428 }
429
430 template <typename T>
431 typename rbtree<T>::Node* rbtree<T>::find(T v, const int op) {
432     Node* ptn = _root;
433     _hot = NULL;
434     while (ptn != NULL) {
435         _hot = ptn;
436         ptn->s += op;
437         if (ptn->val > v)
438             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;
```

```

569 while (ptn) {
570     _hot = ptn;
571     if (ptn->val > v) {
572         ptn = ptn->lc;
573     } else {
574         ptn = ptn->rc;
575     }
576 }
577 if (_hot->val > v) {
578     ptn = _hot;
579 } else {
580     ptn = _hot->right_node();
581 }
582 return iterator(ptn);
583 }

```

---

## 1.11 RBTREE

---

```

1  template <typename T>
2  struct rbtree {
3      struct node {
4          T val;
5          int sz, cnt;
6          node *l, *r, *p;
7          bool color;
8      };
9      node buf[N << 3], *s = buf;
10     node* nil = ++s;
11     node* root = nil;
12     node* find_min(node* x) {
13         while (x->l != nil)
14             x = x->l;
15         return x;
16     }
17     node* find_max(node* x) {
18         while (x->r != nil)
19             x = x->r;
20         return x;
21     }
22     node* find_node(const T& val) {
23         node* x = root;
24         while (x != nil) {
25             if (x->val == val)
26                 return x;
27             if (x->val < val)
28                 x = x->r;
29             else
30                 x = x->l;
31         }
32         return NULL;
33     }
34     void zig(node* x) {
35         node* y = x->r;
36         x->r = y->l;
37         if (y->l != nil)
38             y->l->p = x;
39         y->p = x->p;
40         if (x->p == nil)
41             root = y;
42         else if (x == x->p->r)
43             x->p->r = y;
44         else
45             x->p->l = y;
46         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)

```



```
112     x->p->l = y;
113 else
114     x->p->r = y;
115 return;
116 }
117 void delete_fixup(node* x) {
118     while (x != root && x->color == 0) {
119         if (x == x->p->l) {
120             node* w = x->p->r;
121             if (w->color == 1) {
122                 x->p->color = 1;
123                 w->color = 0;
124                 zig(x->p);
125                 w = x->p->r;
126             }
127             if (w->l->color == 0 && w->r->color == 0) {
128                 w->color = 1;
129                 x = x->p;
130             } else {
131                 if (w->r->color == 0) {
132                     w->color = 1;
133                     w->l->color = 0;
134                     zag(w);
135                     w = x->p->r;
136                 }
137                 w->color = x->p->color;
138                 x->p->color = 0;
139                 w->r->color = 0;
140                 zig(w->p);
141                 x = root;
142             }
143         } else {
144             node* w = x->p->l;
145             if (w->color == 1) {
146                 x->p->color = 1;
147                 w->color = 0;
148                 zag(x->p);
149                 w = x->p->l;
150             }
151             if (w->r->color == 0 && w->l->color == 0) {
152                 w->color = 1;
153                 x = x->p;
154             } else {
155                 if (w->l->color == 0) {
156                     w->color = 1;
157                     w->r->color = 0;
158                     zig(w);
159                     w = x->p->l;
160                 }
161                 w->color = x->p->color;
162                 x->p->color = 0;
163                 w->l->color = 0;
164                 zag(w->p);
165                 x = root;
166             }
167         }
168     }
169     x->color = 0;
170     return;
171 }
172 void ins(const T& val) {
173     node* x = root;
174     node* y = nil;
175     while (x != nil) {
176         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) {

```

```

307     ins(val);
308     T res = INT_MIN;
309     node* x = find_node(val);
310     if (x->l != nil)
311         res = find_max(x->l)->val;
312     else {
313         while (x->p->l == x)
314             x = x->p;
315         if (x->p != nil)
316             res = x->p->val;
317     }
318     del(val);
319     return res;
320 }
321 };

```

---

## 1.12 RMQ

```

1  const int LG = log2(N) + 1;
2  int mi[N][LG], lg[N];
3  void init_rmq(int n) {
4      lg[1] = 0;
5      for (int i = 2; i <= n; ++i)
6          lg[i] = lg[i >> 1] + 1;
7  }
8
9  void build_rmq(int n, int* a) {
10     for (int i = 1; i <= n; ++i)
11         mi[i][0] = a[i];
12     for (int j = 1; j <= lg[n]; ++j) {
13         for (int i = 1; i + (1 << (j - 1)) <= n; ++i) {
14             mi[i][j] = min(mi[i][j - 1], mi[i + (1 << (j - 1))][j - 1]);
15         }
16     }
17 }
18
19 int rmqMin(int l, int r) {
20     int k = lg[r - l + 1];
21     return min(mi[l][k], mi[r - (1 << k) + 1][k]);
22 }

```

---

## 1.13 RollBackCaptainMo

```

1  // Roll Back Captain Mo
2  // 询问 [L, r] 内值相同的元素的最远距离
3  int Ans, ans[N];
4  int block_sz, block_cnt, block_id[N], L[N], R[N];
5  struct Query {
6      int l, r, id;
7      Query() {}
8      Query(int _l, int _r, int _id)
9          : l(_l), r(_r), id(_id) {}
10     bool operator<(const Query& q) const {
11         if (block_id[l] == block_id[q.l])
12             return r < q.r;
13         return block_id[l] < block_id[q.l];
14     }
15 } Q[N];
16
17 int n, m, q, a[N], b[N];
18
19 int nums[N], cn;

```

```
20 int mi[N], ma[N];
21 int __mi[N];
22
23 int brute_force(int l, int r) {
24     int res = 0;
25     for (int i = l; i <= r; ++i)
26         __mi[a[i]] = 0;
27     for (int i = l; i <= r; ++i) {
28         if (__mi[a[i]])
29             res = max(res, i - __mi[a[i]]);
30         else
31             __mi[a[i]] = i;
32     }
33     return res;
34 }
35
36 inline void addl(int p) {
37     if (ma[a[p]])
38         Ans = max(Ans, ma[a[p]] - p);
39     else
40         ma[a[p]] = p;
41 }
42
43 inline void addr(int p) {
44     ma[a[p]] = p;
45     if (!mi[a[p]])
46         mi[a[p]] = p, nums[++cn] = a[p];
47     Ans = max(Ans, p - mi[a[p]]);
48 }
49
50 inline void dell(int p) {
51     if (ma[a[p]] == p)
52         ma[a[p]] = 0;
53 }
54
55 inline void delr(int p) {
56 }
57
58 inline void clear() {
59     for (int i = 1; i <= cn; ++i)
60         mi[nums[i]] = ma[nums[i]] = 0;
61 }
62
63 void RollBackCaptainMo() {
64     block_sz = sqrt(n);
65     block_cnt = n / block_sz;
66
67     for (int i = 1; i <= block_cnt; ++i)
68         L[i] = R[i - 1] + 1, R[i] = i * block_sz;
69     if (R[block_cnt] < n) {
70         ++block_cnt;
71         L[block_cnt] = R[block_cnt - 1] + 1;
72         R[block_cnt] = n;
73     }
74
75     for (int i = 1; i <= block_cnt; ++i)
76         for (int j = L[i]; j <= R[i]; ++j)
77             block_id[j] = i;
78
79     sort(Q + 1, Q + 1 + q);
80
81     for (int i = 1, j = 1; j <= block_cnt; ++j) {
82         int l = R[j] + 1, r = R[j];
83         Ans = 0;
84         cn = 0;
```

```

85     for (; block_id[Q[i].l] == j; ++i) {
86         if (block_id[Q[i].l] == block_id[Q[i].r])
87             ans[Q[i].id] = brute_force(Q[i].l, Q[i].r);
88         else {
89             while (r < Q[i].r)
90                 ++r, addr(r);
91             int tmp = Ans;
92             while (l > Q[i].l)
93                 --l, addl(l);
94             ans[Q[i].id] = Ans;
95             while (l <= R[j])
96                 dell(l), ++l;
97             Ans = tmp;
98         }
99     }
100     clear();
101 }
102 }

```

## 1.14 SegmentTree

```

1  class segtree {
2  public:
3      struct node {
4          // 声明变量, 记得设置初始值
5          // ie. 最大值: int mx = INT_MIN;
6
7          ...
8
9          void
10         apply(int l, int r, ll addv) {
11             // 更新节点信息
12             // ie. 最大值 + 区间加: mx = mx + addv
13
14             ...
15         }
16     };
17
18     friend node operator+(const node& t1, const node& tr) {
19         node t;
20         // 合并两个区间的信息
21         // ie. 区间和: t.sum = t1.sum + t2.sum;
22
23         ...
24
25         return t;
26     }
27
28     inline void push_down(int x, int l, int r) {
29         int lc = x << 1, rc = lc | 1, mid = (l + r) >> 1;
30         // 标记下传
31         // ie. 区间加法
32         // if (tr[x].add != 0) {
33         //     tr[lc].apply(l, mid, tr[x].add);
34         //     tr[rc].apply(mid + 1, r, tr[x].add);
35         //     tr[x].add = 0;
36         // }
37
38         ...
39     }
40
41     /*****
42     inline void push_up(int x) {
43         int lc = x << 1, rc = lc | 1;

```

```

44     tr[x] = tr[lc] + tr[rc];
45 }
46
47 int n;
48 vector<node> tr;
49
50 void build(int x, int l, int r) {
51     if (l == r) {
52         return;
53     }
54     int lc = x << 1, rc = lc | 1, mid = (l + r) >> 1;
55     build(lc, l, mid);
56     build(rc, mid + 1, r);
57     push_up(x);
58 }
59
60 template <class T>
61 void build(int x, int l, int r, const vector<T>& arr) {
62     if (l == r) {
63         tr[x].apply(l, r, arr[l]);
64         return;
65     }
66     int lc = x << 1, rc = lc | 1, mid = (l + r) >> 1;
67     build(lc, l, mid, arr);
68     build(rc, mid + 1, r, arr);
69     push_up(x);
70 }
71
72 template <class T>
73 void build(int x, int l, int r, T* arr) {
74     if (l == r) {
75         tr[x].apply(l, r, arr[l]);
76         return;
77     }
78     int lc = x << 1, rc = lc | 1, mid = (l + r) >> 1;
79     build(lc, l, mid);
80     build(rc, mid + 1, r);
81     push_up(x);
82 }
83
84 node get(int x, int l, int r, int L, int R) {
85     if (L <= l && r <= R) {
86         return tr[x];
87     }
88     push_down(x, l, r);
89     int lc = x << 1, rc = lc | 1, mid = (l + r) >> 1;
90     node res;
91     if (R <= mid)
92         res = get(lc, l, mid, L, R);
93     else if (L > mid)
94         res = get(rc, mid + 1, r, L, R);
95     else
96         res = get(lc, l, mid, L, mid) + get(rc, mid + 1, r, mid + 1, R);
97     push_up(x);
98     return res;
99 }
100
101 template <class... T>
102 void upd(int x, int l, int r, int L, int R, const T&... v) {
103     if (L <= l && r <= R) {
104         tr[x].apply(l, r, v...);
105         return;
106     }
107     push_down(x, l, r);
108     int lc = x << 1, rc = lc | 1, mid = (l + r) >> 1;

```

```

109     node res;
110     if (L <= mid)
111         upd(lc, l, mid, L, R, v...);
112     if (R > mid)
113         upd(rc, mid + 1, r, L, R, v...);
114     push_up(x);
115 }
116
117 int __get_first(int x, int l, int r, const function<bool(const node&)>& f) {
118     if (l == r) {
119         return l;
120     }
121     int lc = x << 1, rc = lc | 1, mid = (l + r) >> 1;
122     push_down(x, l, r);
123     int res;
124     if (f(tr[lc]))
125         res = __get_first(lc, l, mid, f);
126     else
127         res = __get_first(rc, mid + 1, r, f);
128     push_up(x);
129     return res;
130 }
131
132 int get_first(int x, int l, int r, int L, int R, const function<bool(const node&)>& f) {
133     if (L <= l && r <= R) {
134         if (!f(tr[x])) {
135             return -1;
136         }
137         return __get_first(x, l, r, f);
138     }
139     push_down(x, l, r);
140     int lc = x << 1, rc = lc | 1, mid = (l + r) >> 1;
141     int res;
142     if (L <= mid)
143         res = get_first(lc, l, mid, L, R, f);
144     if (res == -1 && R > mid)
145         res = get_first(rc, mid + 1, r, L, R, f);
146     push_up(x);
147     return res;
148 }
149
150 int __get_last(int x, int l, int r, const function<bool(const node&)>& f) {
151     if (l == r) {
152         return l;
153     }
154     int lc = x << 1, rc = lc | 1, mid = (l + r) >> 1;
155     push_down(x, l, r);
156     int res;
157     if (f(tr[lc]))
158         res = __get_first(rc, mid + 1, r, f);
159     else
160         res = __get_first(lc, l, mid, f);
161     push_up(x);
162     return res;
163 }
164
165 int get_last(int x, int l, int r, int L, int R, const function<bool(const node&)>& f) {
166     if (L <= l && r <= R) {
167         if (!f(tr[x])) {
168             return -1;
169         }
170         return __get_first(x, l, r, f);
171     }
172     push_down(x, l, r);
173     int lc = x << 1, rc = lc | 1, mid = (l + r) >> 1;

```



```
174     int res;
175     if (R > mid)
176         res = get_last(rc, mid + 1, r, L, R, f);
177     if (res == -1 && L <= mid)
178         res = get_last(lc, l, mid, L, R, f);
179     push_up(x);
180     return res;
181 }
182
183 int find_first(int l, int r, const function<bool(const node*)>& f) {
184     int L = l, R = r, mid, res = -1;
185     while (L <= R) {
186         mid = (L + R) >> 1;
187         if (f(get(l, mid)))
188             R = mid - 1, res = mid;
189         else
190             L = mid + 1;
191     }
192     return res;
193 }
194
195 int find_last(int l, int r, const function<bool(const node*)>& f) {
196     int L = l, R = r, mid, res = -1;
197     while (L <= R) {
198         mid = (L + R) >> 1;
199         if (f(get(l, mid)))
200             L = mid + 1, res = mid;
201         else
202             R = mid - 1;
203     }
204     return res;
205 }
206
207 segtree(int _n)
208     : n(_n) {
209     assert(n > 0);
210     tr.resize((n << 2) + 5);
211     build(1, 1, n);
212 }
213
214 template <class T>
215 segtree(const vector<T>& arr) {
216     n = arr.size() - 1;
217     assert(n > 0);
218     tr.resize((n << 2) + 5);
219     build(1, 1, n, arr);
220 }
221
222 template <class T>
223 segtree(int _n, T* arr) {
224     n = _n;
225     assert(n > 0);
226     tr.resize((n << 2) + 5);
227     build(1, 1, n, arr);
228 }
229
230 node get(int l, int r) {
231     assert(l >= 1 && l <= r && r <= n);
232     return get(1, 1, n, l, r);
233 }
234
235 node get(int p) {
236     assert(1 <= p && p <= n);
237     return get(1, 1, n, p, p);
238 }
```

```

239
240 template <class... T>
241 void upd(int l, int r, const T&... v) {
242     assert(l >= 1 && l <= r && r <= n);
243     upd(1, 1, n, l, r, v...);
244 }
245
246 template <class... T>
247 void upd1(int p, const T&... v) {
248     assert(p >= 1 && p <= n);
249     upd(1, 1, n, p, p, v...);
250 }
251
252 int get_first(int l, int r, const function<bool(const node&)>& f) {
253     assert(l >= 1 && l <= r && r <= n);
254     return get_first(1, 1, n, l, r, f);
255 }
256
257 int get_last(int l, int r, const function<bool(const node&)>& f) {
258     assert(l >= 1 && l <= r && r <= n);
259     return get_last(1, 1, n, l, r, f);
260 }
261
262 void print(int x, int l, int r) {
263     if (l == r) {
264         cerr << tr[x].sum << " ";
265         return;
266     }
267     push_down(x, l, r);
268     int lc = x << 1, rc = lc | 1, mid = (l + r) >> 1;
269     print(lc, l, mid);
270     print(rc, mid + 1, r);
271 }
272
273 void print() {
274 #ifdef BACKLIGHT
275     cerr << "SGTREE: " << endl;
276     print(1, 1, n);
277     cerr << "\n-----" << endl;
278 #endif
279 }
280 };

```

## 1.15 SGTree

```

1 template <typename T>
2 struct SGTree {
3     static constexpr double alpha = 0.75; //  $\alpha \in (0.5, 1)$ 
4     int root, tot, buf_size;
5     T v[N];
6     int s[N], sz[N], sd[N], cnt[N], l[N], r[N], buf[N];
7
8     SGTree() {
9         root = tot = 0;
10    }
11
12    int new_node(T _v) {
13        ++tot;
14        v[tot] = _v;
15        s[tot] = sz[tot] = sd[tot] = cnt[tot] = 1;
16        l[tot] = r[tot] = 0;
17        return tot;
18    }
19

```

```
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) {
```

```

150     return getKth(getSuccRank(root, val));
151 }
152
153 void debug(int x) {
154     if (!x)
155         return;
156     debug(l[x]);
157     cerr << v[x] << " ";
158     debug(r[x]);
159 }
160
161 void debug() {
162     cerr << "SGTree:" << endl;
163     debug(root);
164     cerr << endl;
165 }
166 };

```

---

## 1.16 Splay

```

1 namespace Backlight {
2
3 namespace Splay {
4     using T = int;
5     #define ls ch[x][0]
6     #define rs ch[x][1]
7     const int S = N;
8
9     int tot, rt, sz[S], cnt[S], ch[S][2], fa[S];
10
11     T v[S];
12
13     inline void init() {
14         tot = rt = 0;
15     }
16
17     inline void clear(int x) {
18         ch[x][0] = ch[x][1] = fa[x] = sz[x] = cnt[x] = v[x] = 0;
19     }
20
21     inline int get(int x) {
22         return ch[fa[x]][1] == x;
23     }
24
25     inline int newnode(T val) {
26         ++tot;
27         sz[tot] = cnt[tot] = 1;
28         ch[tot][0] = ch[tot][1] = fa[tot] = 0;
29         v[tot] = val;
30         return tot;
31     }
32
33     inline void push_up(int x) {
34         if (!x)
35             return;
36         sz[x] = sz[ls] + cnt[x] + sz[rs];
37     }
38
39     void rotate(int x) {
40         int f = fa[x], g = fa[f], i = get(x);
41         ch[f][i] = ch[x][i ^ 1];
42         fa[ch[f][i]] = f;
43         ch[x][i ^ 1] = f;
44         fa[f] = x;

```

```

45     fa[x] = g;
46     if (g)
47         ch[g][ch[g][1] == f] = x;
48     push_up(f);
49     push_up(x);
50 }
51
52 void splay(int x, int ed) {
53     for (int f; (f = fa[x]) != ed; rotate(x))
54         if (fa[f] != ed)
55             rotate((get(x) == get(f) ? f : x));
56     if (ed == 0)
57         rt = x;
58 }
59
60 void insert(T val) {
61     if (rt == 0) {
62         rt = newnode(val);
63         return;
64     }
65     int p = rt, f = 0;
66     while (true) {
67         if (val == v[p]) {
68             ++cnt[p];
69             push_up(p);
70             push_up(f);
71             break;
72         }
73         f = p;
74         p = ch[p][v[p] < val];
75         if (p == 0) {
76             p = newnode(val);
77             fa[p] = f;
78             ch[f][v[f] < val] = p;
79             push_up(f);
80             break;
81         }
82     }
83     splay(p, 0);
84 }
85
86 int getrank(T val) {
87     int p = rt, res = 0;
88     while (p) {
89         if (v[p] > val)
90             p = ch[p][0];
91         else {
92             res += sz[ch[p][0]];
93             if (v[p] == val)
94                 break;
95             res += cnt[p];
96             p = ch[p][1];
97         }
98     }
99     assert(p != 0);
100     splay(p, 0);
101     return res + 1;
102 }
103
104 T getkth(int k) {
105     int p = rt, res = 0;
106     while (p) {
107         if (k <= sz[ch[p][0]])
108             p = ch[p][0];
109         else {

```

```

110     if (k <= sz[ch[p][0]] + cnt[p]) {
111         res = v[p];
112         break;
113     } else
114         k -= sz[ch[p][0]] + cnt[p], p = ch[p][1];
115     }
116 }
117 assert(p != 0);
118 splay(p, 0);
119 return res;
120 }
121
122 void remove(T val) {
123     getrank(val); // splay val to root
124     if (cnt[rt] > 1) {
125         --cnt[rt];
126         push_up(rt);
127         return;
128     }
129     if (!ch[rt][0] && !ch[rt][1]) {
130         clear(rt);
131         rt = 0;
132         return;
133     }
134     if (!ch[rt][0] || !ch[rt][1]) {
135         int nrt = ch[rt][0] ? ch[rt][0] : ch[rt][1];
136         clear(rt);
137         rt = nrt;
138         fa[rt] = 0;
139         return;
140     }
141     int ort = rt;
142     int p = ch[rt][0];
143     while (ch[p][1])
144         p = ch[p][1];
145     splay(p, 0);
146     ch[rt][1] = ch[ort][1];
147     fa[ch[ort][1]] = rt;
148     clear(ort);
149     push_up(rt);
150 }
151
152 T getpre(T val) {
153     int p = rt, res = -INF;
154     while (p) {
155         if (v[p] < val && v[p] > res)
156             res = v[p];
157         if (val > v[p])
158             p = ch[p][1];
159         else
160             p = ch[p][0];
161     }
162     // splay(p, 0);
163     return res;
164 }
165
166 T getsuc(T val) {
167     int p = rt, res = INF;
168     while (p) {
169         if (v[p] > val && v[p] < res)
170             res = v[p];
171         if (val < v[p])
172             p = ch[p][0];
173         else
174             p = ch[p][1];

```

```

175     }
176     // splay(p, 0);
177     return res;
178 }
179
180 void DEBUG(int x) {
181     if (!x)
182         return;
183     DEBUG(ls);
184     cerr << v[x] << " ";
185     DEBUG(rs);
186 }
187
188 void DEBUG() {
189     cerr << "Splay: ";
190     DEBUG(rt);
191     cerr << endl;
192 }
193 } // namespace Splay
194
195 } // namespace Backlight

```

---

## 1.17 Treap-pointer

---

```

1 // mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
2 // inline unsigned rng() {
3 //     static unsigned x = 7;
4 //     return x = x * 0xdefaced + 1;
5 // }
6
7 template <typename T>
8 struct Treap {
9     struct node {
10         node *l, *r;
11         unsigned rnd;
12         T v;
13         int sz;
14         node(T _v)
15             : l(NULL), r(NULL), rnd(rng()), sz(1), v(_v) {
16         }
17     };
18
19     inline int get_size(node*& p) {
20         return p ? p->sz : 0;
21     }
22
23     inline void push_up(node*& p) {
24         if (!p)
25             return;
26         p->sz = get_size(p->l) + get_size(p->r) + 1;
27     }
28
29     node* root = NULL;
30
31     node* merge(node* a, node* b) {
32         if (!a)
33             return b;
34         if (!b)
35             return a;
36         if (a->rnd < b->rnd) {
37             a->r = merge(a->r, b);
38             push_up(a);
39             return a;
40         } else {

```



```

41     b->l = merge(a, b->l);
42     push_up(b);
43     return b;
44 }
45 }
46
47 void split_val(node* p, const T& k, node*& a, node*& b) {
48     if (!p)
49         a = b = NULL;
50     else {
51         if (p->v <= k) {
52             a = p;
53             split_val(p->r, k, a->r, b);
54             push_up(a);
55         } else {
56             b = p;
57             split_val(p->l, k, a, b->l);
58             push_up(b);
59         }
60     }
61 }
62
63 void split_size(node* p, int k, node*& a, node*& b) {
64     if (!p)
65         a = b = NULL;
66     else {
67         if (get_size(p->l) <= k) {
68             a = p;
69             split_size(p->r, k - get_size(p->l) - 1, a->r, b);
70             push_up(a);
71         } else {
72             b = p;
73             split_size(p->l, k, a, b->l);
74             push_up(b);
75         }
76     }
77 }
78
79 void ins(T val) {
80     node *a, *b;
81     split_val(root, val, a, b);
82     a = merge(a, new node(val));
83     root = merge(a, b);
84 }
85
86 void del(T val) {
87     node *a, *b, *c, *d;
88     split_val(root, val, a, b);
89     split_val(a, val - 1, c, d);
90     node* e = d;
91     d = merge(d->l, d->r);
92     delete e;
93     a = merge(c, d);
94     root = merge(a, b);
95 }
96
97 T getRank(T val) {
98     node *a, *b;
99     split_val(root, val - 1, a, b);
100     T res = get_size(a) + 1;
101     root = merge(a, b);
102     return res;
103 }
104
105 T getKth(int k) {

```

```

106     node* x = root;
107     T res = numeric_limits<T>::min();
108     while (x) {
109         if (k <= get_size(x->l))
110             x = x->l;
111         else {
112             if (get_size(x->l) + 1 == k) {
113                 res = x->v;
114                 break;
115             } else {
116                 k -= get_size(x->l) + 1;
117                 x = x->r;
118             }
119         }
120     }
121     return res;
122 }
123
124 T getPrev(T val) {
125     node *a, *b;
126     split_val(root, val - 1, a, b);
127     node* p = a;
128     while (p->r)
129         p = p->r;
130     root = merge(a, b);
131     return p->v;
132 }
133
134 T getSucc(T val) {
135     node *a, *b;
136     split_val(root, val, a, b);
137     node* p = b;
138     while (p->l)
139         p = p->l;
140     root = merge(a, b);
141     return p->v;
142 }
143 };

```

---

## 1.18 Treap

```

1  namespace Treap {
2  using T = long long;
3  const int S = N;
4  mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
5
6  int tot, rt, sz[S], L[S], R[S], rnd[S];
7
8  T v[S];
9
10 inline void init() {
11     tot = rt = 0;
12 }
13
14 inline int newnode(T val) {
15     ++tot;
16     sz[tot] = 1;
17     L[tot] = R[tot] = 0;
18     rnd[tot] = rng();
19     v[tot] = val;
20     return tot;
21 }
22
23 inline void push_up(int x) {

```

```

24     sz[x] = sz[L[x]] + 1 + sz[R[x]];
25 }
26
27 void split(int u, T k, int& x, int& y) {
28     if (!u)
29         x = y = 0;
30     else {
31         if (v[u] <= k) {
32             x = u;
33             split(R[u], k, R[u], y);
34         } else {
35             y = u;
36             split(L[u], k, x, L[u]);
37         }
38         push_up(u);
39     }
40 }
41
42 int merge(int x, int y) {
43     if (!x || !y)
44         return x | y;
45     if (rnd[x] < rnd[y]) {
46         R[x] = merge(R[x], y);
47         push_up(x);
48         return x;
49     } else {
50         L[y] = merge(x, L[y]);
51         push_up(y);
52         return y;
53     }
54 }
55
56 void insert(T val) {
57     int x, y;
58     split(rt, val, x, y);
59     x = merge(x, newnode(val));
60     rt = merge(x, y);
61 }
62
63 void remove(T val) {
64     int x1, y1, x2, y2;
65     split(rt, val, x1, y1);
66     split(x1, val - 1, x2, y2);
67     y2 = merge(L[y2], R[y2]);
68     x1 = merge(x2, y2);
69     rt = merge(x1, y1);
70 }
71
72 int getrank(T val) {
73     int x, y;
74     split(rt, val - 1, x, y);
75     int res = sz[x] + 1;
76     rt = merge(x, y);
77     return res;
78 }
79
80 T getkth(int k) {
81     int u = rt;
82     while (true) {
83         if (k <= sz[L[u]])
84             u = L[u];
85         else {
86             if (sz[L[u]] + 1 == k)
87                 break;
88             else

```

```

89     k -= sz[L[u]] + 1, u = R[u];
90 }
91 }
92 return v[u];
93 }
94
95 T getpre(T val) {
96     int x, y;
97     split(rt, val - 1, x, y);
98     int p = x;
99     while (R[p])
100         p = R[p];
101     rt = merge(x, y);
102     return v[p];
103 }
104
105 T getsuc(T val) {
106     int x, y;
107     split(rt, val, x, y);
108     int p = y;
109     while (L[p])
110         p = L[p];
111     rt = merge(x, y);
112     return v[p];
113 }
114
115 void DEBUG(int u) {
116     if (!u)
117         return;
118     DEBUG(L[u]);
119     cerr << v[u] << " ";
120     DEBUG(R[u]);
121 }
122
123 void DEBUG() {
124     cerr << "Treap: ";
125     DEBUG(rt);
126     cerr << endl;
127 }
128 } // namespace Treap

```

## 2 graph

### 2.1 BCC-Edge

```

1 namespace Backlight {
2
3 struct Graph {
4     #define fore(i, u) for (int i = h[u]; i; i = e[i].nxt)
5     struct Edge {
6         int v, nxt;
7         Edge() {}
8         Edge(int _v, int _nxt)
9             : v(_v), nxt(_nxt) {}
10    };
11
12    int V, E, tot;
13    vector<int> h;
14    vector<Edge> e;
15
16    Graph()
17        : V(0) {}
18    Graph(int _V, int _E)

```

```

19         : V(_V), E(2 * _E), tot(0), h(_V + 1), e(2 * _E + 1) {}
20
21 inline void addarc(int u, int v) {
22     assert(1 <= u && u <= V);
23     assert(1 <= v && v <= V);
24
25     e[++tot] = Edge(v, h[u]);
26     h[u] = tot;
27 }
28
29 inline void addedge(int u, int v) {
30     addarc(u, v);
31     addarc(v, u);
32 }
33
34 /*****
35 int bcc_clock, bcc_cnt;
36 vector<int> dfn, low, belong, bcc_size;
37 vector<vector<int>> bcc;
38 vector<bool> bridge;
39
40 void tarjan(int u, int fa) {
41     dfn[u] = low[u] = ++bcc_clock;
42     fore(i, u) {
43         int v = e[i].v;
44         if (v == fa)
45             continue;
46
47         if (!dfn[v]) {
48             tarjan(v, u);
49             low[u] = min(low[u], low[v]);
50             if (dfn[u] < low[v]) {
51                 bridge[i] = true;
52                 if (i & 1)
53                     bridge[i + 1] = true;
54                 else
55                     bridge[i - 1] = true;
56             }
57         } else if (dfn[v] < dfn[u]) {
58             low[u] = min(low[u], dfn[v]);
59         }
60     }
61 }
62
63 void blood_fill(int u) {
64     belong[u] = bcc_cnt;
65     bcc[bcc_cnt].push_back(u);
66     fore(i, u) {
67         if (bridge[i])
68             continue;
69         int v = e[i].v;
70         if (!belong[v])
71             blood_fill(v);
72     }
73 }
74
75 void build_bcc_point() {
76     bcc_clock = bcc_cnt = 0;
77     dfn = vector<int>(V + 1);
78     low = vector<int>(V + 1);
79     belong = vector<int>(V + 1);
80     bridge = vector<bool>(E + 1);
81     bcc = vector<vector<int>>(1);
82
83     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()) {

```

```

42     vector<int> nb;
43     nb.push_back(u);
44     bcc.push_back(nb);
45     return;
46 }
47
48 int son = 0;
49 for (Edge& e : G[u]) {
50     int v = e.v;
51     if (v == fa)
52         continue;
53
54     if (!dfn[v]) {
55         tarjan(v, u);
56         low[u] = min(low[u], low[v]);
57         if (dfn[u] <= low[v]) {
58             ++son;
59             if (u != fa || son > 1)
60                 cut[u] = true;
61             vector<int> nb;
62             int top;
63             do {
64                 top = stk.top();
65                 stk.pop();
66                 nb.push_back(top);
67             } while (top != v);
68             nb.push_back(u);
69             bcc.push_back(nb);
70         }
71     } else
72         low[u] = min(low[u], dfn[v]);
73 }
74 }
75
76 void build_bcc_point() {
77     bcc_clock = 0;
78     dfn = vector<int>(V + 1);
79     low = vector<int>(V + 1);
80     cut = vector<bool>(V + 1);
81     bcc = vector<vector<int>>(1);
82
83     for (int i = 1; i <= V; ++i) {
84         if (!dfn[i]) {
85             while (!stk.empty())
86                 stk.pop();
87             tarjan(i, i);
88         }
89     }
90 }
91 };
92
93 } // namespace Backlight

```

## 2.3 BiGraphMatch

```

1 // Hopcroft Karp,  $O(\sqrt{V}E)$ 
2 struct bigraph {
3     int dfn;
4
5     vector<vector<int>> G;
6
7     int nl, nr;
8     vector<int> ml, mr;
9     vector<int> ll, lr;

```

```
10  vector<int> vis;
11
12  bigraph(int _nl, int _nr) {
13      nl = _nl;
14      nr = _nr;
15      G = vector<vector<int>>(nl + 1);
16  }
17
18  void addarc(int u, int v) {
19      G[u].push_back(v);
20  }
21
22  void addedge(int u, int v) {
23      G[u].push_back(v);
24      G[v].push_back(u);
25  }
26
27  bool bfs() {
28      queue<int> q;
29      bool res = false;
30
31      for (int i = 1; i <= nl; ++i) {
32          if (ml[i])
33              ll[i] = 0;
34          else
35              ll[i] = 1, q.push(i);
36      }
37
38      for (int i = 1; i <= nr; ++i)
39          lr[i] = 0;
40
41      while (!q.empty()) {
42          int u = q.front();
43          q.pop();
44          for (int v : G[u]) {
45              if (lr[v] == 0) {
46                  lr[v] = ll[u] + 1;
47                  if (mr[v]) {
48                      ll[mr[v]] = lr[v] + 1;
49                      q.push(mr[v]);
50                  } else
51                      res = true;
52              }
53          }
54      }
55
56      return res;
57  };
58
59  bool dfs(int u) {
60      for (int v : G[u]) {
61          if (lr[v] == ll[u] + 1 && vis[v] != dfn) {
62              vis[v] = dfn;
63              if (mr[v] == 0 || dfs(mr[v])) {
64                  mr[v] = u;
65                  ml[u] = v;
66                  return true;
67              }
68          }
69      }
70      return false;
71  };
72
73  int HK() {
74      ml = vector<int>(nl + 1);
```



```

75     mr = vector<int>(nr + 1);
76     ll = vector<int>(nl + 1);
77     lr = vector<int>(nr + 1);
78     vis = vector<int>(nr + 1);
79
80     int res = 0;
81     while (bfs()) {
82         ++dfn;
83         for (int i = 1; i <= nl; ++i)
84             if (!ml[i])
85                 res += dfs(i);
86     }
87     return res;
88 }
89 };
90
91 /**
92  * 最小覆盖数 = 最大匹配数
93  * 最大独立集 = 顶点数 - 二分图匹配数
94  * DAG 最小路径覆盖数 = 结点数 - 拆点后二分图最大匹配数
95  */

```

## 2.4 BiWrapMatch

```

1 // Kuhn Munkres,  $O(V^3)$ 
2 template <typename T>
3 struct biwrap {
4     T TMAX, TMIN;
5
6     int n, nl, nr;
7     vector<vector<T>> G;
8     vector<T> highl, highr;
9     vector<T> slack;
10    vector<int> matchl, matchr; // match
11    vector<int> pre;           // pre node
12    vector<bool> visl, visr;   // vis
13    vector<int> q;
14    int ql, qr;
15
16    biwrap(int _nl, int _nr) {
17        TMAX = numeric_limits<T>::max();
18
19        nl = _nl;
20        nr = _nr;
21        n = max(nl, nr);
22        G = vector<vector<T>>(n + 1, vector<T>(n + 1));
23        highl = vector<T>(n + 1);
24        highr = vector<T>(n + 1);
25        slack = vector<T>(n + 1);
26        matchl = vector<int>(n + 1);
27        matchr = vector<int>(n + 1);
28        pre = vector<int>(n + 1);
29        visl = vector<bool>(n + 1);
30        visr = vector<bool>(n + 1);
31        q = vector<int>(n + 1);
32    }
33
34    void addarc(int u, int v, T w) {
35        G[u][v] = max(G[u][v], w);
36    }
37
38    bool check(int v) {
39        visr[v] = true;
40        if (matchr[v]) {

```

```

41     q[qr++] = matchr[v];
42     visl[matchr[v]] = true;
43     return false;
44 }
45
46 while (v) {
47     matchr[v] = pre[v];
48     swap(v, matchl[pre[v]]);
49 }
50
51 return true;
52 }
53
54 void bfs(int now) {
55     ql = qr = 0;
56     q[qr++] = now;
57     visl[now] = 1;
58     while (true) {
59         while (ql < qr) {
60             int u = q[ql++];
61             for (int v = 1; v <= n; ++v) {
62                 if (!visr[v]) {
63                     T delta = highl[u] + highr[v] - G[u][v];
64                     if (slack[v] >= delta) {
65                         pre[v] = u;
66                         if (delta)
67                             slack[v] = delta;
68                         else if (check(v))
69                             return;
70                     }
71                 }
72             }
73         }
74
75         T a = TMAX;
76         for (int i = 1; i <= n; ++i)
77             if (!visr[i])
78                 a = min(a, slack[i]);
79         for (int i = 1; i <= n; ++i) {
80             if (visl[i])
81                 highl[i] -= a;
82             if (visr[i])
83                 highr[i] += a;
84             else
85                 slack[i] -= a;
86         }
87         for (int i = 1; i <= n; ++i)
88             if (!visr[i] && !slack[i] && check(i))
89                 return;
90     }
91 }
92
93 void match() {
94     fill(highr.begin(), highr.end(), 0);
95     fill(matchl.begin(), matchl.end(), 0);
96     fill(matchr.begin(), matchr.end(), 0);
97     for (int i = 1; i <= n; ++i)
98         highl[i] = *max_element(G[i].begin() + 1, G[i].end());
99
100     for (int i = 1; i <= n; ++i) {
101         fill(slack.begin(), slack.end(), TMAX);
102         fill(visl.begin(), visl.end(), false);
103         fill(visr.begin(), visr.end(), false);
104         bfs(i);
105     }

```

```

106 }
107
108 T getMaxMatch() {
109     T res = 0;
110     match();
111     for (int i = 1; i <= n; ++i) {
112         if (G[i][matchl[i]] > 0)
113             res += G[i][matchl[i]];
114         else
115             matchl[i] = 0;
116     }
117     return res;
118 }
119 };

```

## 2.5 BlockForest

```

1 // 「APIO2018」铁人两项 (https://loj.ac/p/2587)
2 // 给定一张简单无向图，问有多少对三元组  $\langle s, c, f \rangle$  ( $s, c, f$  互不相同) 使得存在一条简单路径从  $s$  出发，经过  $c$  到达  $f$ 。
3 #include <bits/stdc++.h>
4 using namespace std;
5 using ll = long long;
6 const int N = 2e5 + 5;
7
8 int n, m;
9 int w[N];
10 vector<int> G[N], F[N];
11
12 int cc, scc;
13 int dfc, dfn[N], low[N];
14 int top, stk[N];
15 void tarjan(int u) {
16     ++cc;
17     dfn[u] = low[u] = ++dfc;
18     stk[++top] = u;
19     for (int v : G[u]) {
20         if (!dfn[v]) {
21             tarjan(v);
22             low[u] = min(low[u], low[v]);
23             if (low[v] == dfn[u]) {
24                 ++scc;
25                 int np = n + scc;
26                 w[np] = 0;
27                 for (int x = 0; x != v; --top) {
28                     x = stk[top];
29                     F[np].push_back(x);
30                     F[x].push_back(np);
31                     ++w[np];
32                 }
33                 F[np].push_back(u);
34                 F[u].push_back(np);
35                 ++w[np];
36             }
37         } else
38             low[u] = min(low[u], dfn[v]);
39     }
40 }
41
42 ll ans;
43 int sz[N];
44 void dfs(int u, int fa) {
45     sz[u] = (u <= n);
46     for (int v : F[u])
47         if (v != fa) {

```

```

48     dfs(v, u);
49     ans += 211 * w[u] * sz[u] * sz[v];
50     sz[u] += sz[v];
51 }
52 ans += 211 * w[u] * sz[u] * (cc - sz[u]);
53 }
54
55 void buildBlockForest() {
56     for (int i = 1; i <= n; ++i)
57         if (!dfn[i]) {
58             cc = 0;
59             tarjan(i);
60             --top;
61             dfs(i, i);
62         }
63 }
64
65 void solve(int Case) {
66     scanf("%d %d", &n, &m);
67     fill(w + 1, w + 1 + n, -1);
68     int u, v;
69     for (int i = 1; i <= m; ++i) {
70         scanf("%d %d", &u, &v);
71         G[u].push_back(v);
72         G[v].push_back(u);
73     }
74     buildBlockForest();
75     printf("%lld\n", ans);
76 }
77
78 int main() {
79     int T = 1;
80     // scanf("%d", &T);
81     for (int i = 1; i <= T; ++i)
82         solve(i);
83     return 0;
84 }

```

## 2.6 BlockTree

```

1 // 树分块: uv 之间路径上不同的颜色数 (强制在线)
2 #include <bits/stdc++.h>
3 using namespace std;
4
5 const int N = 4e4 + 5;
6
7 int n, m, a[N];
8 int nt, t[N];
9
10 int tot, head[N];
11 struct edge {
12     int v, nxt;
13 } e[N << 1];
14 void init(int n) {
15     tot = 0;
16     for (int i = 1; i <= n; ++i)
17         head[i] = 0;
18 }
19 void add(int u, int v) {
20     ++tot;
21     e[tot] = (edge){v, head[u]};
22     head[u] = tot;
23 }
24 #define fore(i, u) for (int i = head[u]; i; i = e[i].nxt)

```

```
25
26 int sz[N], son[N], f[N], h[N], top[N];
27
28 void dfs1(int u, int fa) {
29     f[u] = fa;
30     h[u] = h[fa] + 1;
31     sz[u] = 1;
32     son[u] = 0;
33     fore(i, u) {
34         int v = e[i].v;
35         if (v == fa)
36             continue;
37         dfs1(v, u);
38         sz[u] += sz[v];
39         if (sz[v] > sz[son[u]])
40             son[u] = v;
41     }
42 }
43
44 void dfs2(int u, int fa, int k) {
45     top[u] = k;
46     if (son[u])
47         dfs2(son[u], u, k);
48     fore(i, u) {
49         int v = e[i].v;
50         if (v == fa || v == son[u])
51             continue;
52         dfs2(v, u, v);
53     }
54 }
55
56 int lca(int u, int v) {
57     while (top[u] != top[v]) {
58         if (h[top[u]] < h[top[v]])
59             swap(u, v);
60         u = f[top[u]];
61     }
62     if (h[u] > h[v])
63         swap(u, v);
64     return u;
65 }
66
67 int dep[N], max_dep[N], pa[N];
68 int key_cnt, keyid[N];
69
70 const int COLORCNT = 4e4 + 2;
71 const int KEYCNT = 101;
72 const int gap = 400;
73
74 bitset<COLORCNT> c[KEYCNT][KEYCNT];
75
76 int stk[N], tp;
77
78 void dfs_key(int u, int fa) {
79     dep[u] = dep[fa] + 1;
80     max_dep[u] = dep[u];
81     fore(i, u) {
82         int v = e[i].v;
83         if (v == fa)
84             continue;
85         dfs_key(v, u);
86         if (max_dep[v] > max_dep[u])
87             max_dep[u] = max_dep[v];
88     }
89     if (max_dep[u] - dep[u] >= gap) {
```

```

90     keyid[u] = ++key_cnt;
91     max_dep[u] = dep[u];
92 }
93 }
94
95 void dfs_bitset(int u) {
96     if (keyid[u] && u != stk[tp]) {
97         for (int x = u; x != stk[tp]; x = f[x])
98             c[keyid[stk[tp]]][keyid[u]].set(a[x]);
99
100        for (int i = 1; i < tp; ++i) {
101            c[keyid[stk[i]]][keyid[u]] = c[keyid[stk[i]]][keyid[stk[tp]]];
102            c[keyid[stk[i]]][keyid[u]] |= c[keyid[stk[tp]]][keyid[u]];
103        }
104        pa[u] = stk[tp];
105        stk[++tp] = u;
106    }
107    for (int i = head[u]; i; i = e[i].nxt) {
108        if (e[i].v != f[u])
109            dfs_bitset(e[i].v);
110    }
111    if (keyid[u])
112        --tp;
113 }
114
115 void build_block_tree() {
116     key_cnt = 0;
117     dfs_key(1, 1);
118     if (!keyid[1])
119         keyid[1] = ++key_cnt;
120
121     tp = 1;
122     stk[1] = 1;
123     dfs_bitset(1);
124 }
125
126 bitset<COLORCNT> res;
127
128 int query(int u, int v) {
129     res.reset();
130     int uv = lca(u, v);
131
132     // step 1: jump to nearest key node
133     while (u != uv && !keyid[u]) {
134         res.set(a[u]);
135         u = f[u];
136     }
137     while (v != uv && !keyid[v]) {
138         res.set(a[v]);
139         v = f[v];
140     }
141
142     // step 2: jump to lowest key node
143     int pu = u;
144     while (dep[pa[pu]] >= dep[uv])
145         pu = pa[pu];
146     if (pu != u) {
147         res |= c[keyid[pu]][keyid[u]];
148         u = pu;
149     }
150
151     int pv = v;
152     while (dep[pa[pv]] >= dep[uv])
153         pv = pa[pv];
154     if (pv != v) {

```

```
155     res |= c[keyid[pv]][keyid[v]];
156     v = pv;
157 }
158
159 // step 3: jump to lca
160 while (u != uv) {
161     res.set(a[u]);
162     u = f[u];
163 }
164 while (v != uv) {
165     res.set(a[v]);
166     v = f[v];
167 }
168
169 // step 4: set lca
170 res.set(a[uv]);
171
172 return res.count();
173 }
174
175 void solve(int Case) {
176     scanf("%d %d", &n, &m);
177     for (int i = 1; i <= n; ++i) {
178         scanf("%d", &a[i]);
179         t[i] = a[i];
180     }
181
182     sort(t + 1, t + 1 + n);
183     nt = unique(t + 1, t + 1 + n) - (t + 1);
184
185     for (int i = 1; i <= n; ++i)
186         a[i] = lower_bound(t + 1, t + 1 + nt, a[i]) - t;
187
188     init(n);
189     int u, v;
190     for (int i = 1; i <= n - 1; ++i) {
191         scanf("%d %d", &u, &v);
192         add(u, v);
193         add(v, u);
194     }
195
196     dfs1(1, 1);
197     dfs2(1, 1, 1);
198
199     build_block_tree();
200
201     int lastans = 0;
202     for (int i = 1; i <= m; ++i) {
203         scanf("%d %d", &u, &v);
204         u ^= lastans;
205         lastans = query(u, v);
206         printf("%d\n", lastans);
207     }
208 }
209
210 int main() {
211     int T = 1;
212     // scanf( "%d", &T );
213     for (int _ = 1; _ <= T; _++)
214         solve(_);
215     return 0;
216 }
```

## 2.7 Dijkstra

---

```

1 namespace Backlight {
2
3 template <typename T>
4 struct Wraph {
5     struct Edge {
6         int u, v;
7         T w;
8         Edge() {}
9         Edge(int _u, int _v, T _w)
10             : u(_u), v(_v), w(_w) {}
11     };
12
13     int V;
14     vector<vector<Edge>> G;
15
16     Wraph()
17         : V(0) {}
18     Wraph(int _V)
19         : V(_V), G(_V + 1) {}
20
21     inline void addarc(int u, int v, T w) {
22         assert(1 <= u && u <= V);
23         assert(1 <= v && v <= V);
24         G[u].push_back(Edge(u, v, w));
25     }
26
27     inline void addedge(int u, int v, T w) {
28         addarc(u, v, w);
29         addarc(v, u, w);
30     }
31
32     /*****
33     vector<T> dijkstra(int S, T T_MAX) {
34         typedef pair<T, int> Node;
35         priority_queue<Node, vector<Node>, greater<Node>> q;
36         vector<T> dis(V + 1);
37         for (int i = 1; i <= V; i++)
38             dis[i] = T_MAX;
39         dis[S] = 0;
40         q.push(Node(0, S));
41         while (!q.empty()) {
42             Node p = q.top();
43             q.pop();
44             T cost = p.first;
45             int u = p.second;
46             if (dis[u] != cost)
47                 continue;
48
49             for (Edge e : G[u]) {
50                 int v = e.v;
51                 T w = e.w;
52                 if (dis[v] > dis[u] + w) {
53                     dis[v] = dis[u] + w;
54                     q.push(Node(dis[v], v));
55                 }
56             }
57         }
58         return dis;
59     }
60 };
61
62 } // namespace Backlight

```

---



## 2.8 dsu-on-tree

```
1 // CF600E
2 // 对于每个节点，输出其子树中出现次数最多的颜色之和。
3 vector<int> G[N];
4 inline void addedge(int u, int v) {
5     G[u].push_back(v);
6     G[v].push_back(u);
7 }
8
9 int n, color[N];
10
11 int sz[N], son[N], cnt[N], ma;
12 ll cur, ans[N];
13 void dfs1(int u, int fa) {
14     sz[u] = 1;
15     son[u] = -1;
16     for (int v : G[u]) {
17         if (v == fa)
18             continue;
19         dfs1(v, u);
20         sz[u] += sz[v];
21         if (sz[v] > sz[son[u]])
22             son[u] = v;
23     }
24 }
25
26 void add(int u, int fa, int Son, int d) {
27     // update data here
28     cnt[color[u]] += d;
29     if (cnt[color[u]] > ma)
30         ma = cnt[color[u]], cur = 0;
31     if (cnt[color[u]] == ma)
32         cur += color[u];
33
34     for (int v : G[u]) {
35         if (v == fa || v == Son)
36             continue;
37         add(v, u, Son, d);
38     }
39 }
40
41 void dfs2(int u, int fa, bool keep) {
42     for (int v : G[u]) {
43         if (v == fa || v == son[u])
44             continue;
45         dfs2(v, u, false);
46     }
47     if (son[u] != -1)
48         dfs2(son[u], u, true);
49
50     add(u, fa, son[u], 1);
51
52     // answer queries here
53     ans[u] = cur;
54
55     if (!keep) {
56         add(u, fa, -1, -1);
57         ma = 0;
58         cur = 0;
59     }
60 }
61
62 void solve() {
63     read(n);
```

```

64  FOR(i, 1, n)
65  read(color[i]);
66
67  int u, v;
68  FOR(i, 2, n) {
69      read(u, v);
70      addedge(u, v);
71  }
72
73  dfs1(1, 0);
74  dfs2(1, 0, 0);
75
76  FOR(i, 1, n - 1)
77  printf("%lld ", ans[i]);
78  println(ans[n]);
79 }

```

---

## 2.9 ExKruskal

---

```

1  // https://loj.ac/p/6021
2  // https://oi-wiki.org/graph/mst/#kruskal_1
3  #include <bits/stdc++.h>
4  using namespace std;
5  #ifdef BACKLIGHT
6  #include "debug.h"
7  #else
8  #define debug(...)
9  #endif
10
11 const int __BUFFER_SIZE__ = 1 << 20;
12 bool NEOF = 1;
13 int __top;
14 char __buf[__BUFFER_SIZE__], *__p1 = __buf, *__p2 = __buf, __stk[996];
15 inline char nc() {
16     if (!NEOF)
17         return EOF;
18     if (__p1 == __p2) {
19         __p1 = __buf;
20         __p2 = __buf + fread(__buf, 1, __BUFFER_SIZE__, stdin);
21         if (__p1 == __p2) {
22             NEOF = 0;
23             return EOF;
24         }
25     }
26     return *__p1++;
27 }
28
29 template <typename T>
30 inline bool rd(T& x) {
31     char c = nc();
32     bool f = 0;
33     x = 0;
34     while (!isdigit(c))
35         c == '-' && (f = 1), c = nc();
36     while (isdigit(c))
37         x = x * 10 + (c ^ 48), c = nc();
38     if (f)
39         x = -x;
40     return NEOF;
41 }
42
43 typedef unsigned long long ull;
44 ull myRand(ull& k1, ull& k2) {
45     ull k3 = k1, k4 = k2;

```

```

46  k1 = k4;
47  k3 ^= (k3 << 23);
48  k2 = k3 ^ k4 ^ (k3 >> 17) ^ (k4 >> 26);
49  return k2 + k4;
50 }
51 pair<int, int> myRanq(u1l& k1, u1l& k2, int MAXN) {
52     int x = myRand(k1, k2) % MAXN + 1, y = myRand(k1, k2) % MAXN + 1;
53     if (x > y)
54         return make_pair(y, x);
55     else
56         return make_pair(x, y);
57 }
58
59 struct LCAGraph {
60     int n, m, dfs_clock;
61     vector<vector<int>> G;
62     vector<int> dfn, lg, dep;
63     vector<vector<int>> st;
64     LCAGraph(int _n = 0)
65         : n(_n), m(n + n - 1), G(n), dfn(n), lg(m + 1), dep(n), st(m) {
66         lg[1] = 0;
67         for (int i = 2; i <= m; ++i)
68             lg[i] = lg[i >> 1] + 1;
69     }
70     void addedge(int u, int v) { G[u].push_back(v); }
71     void dfs(int u, int fa) {
72         dfn[u] = dfs_clock;
73         dep[u] = dep[fa] + 1;
74         st[dfs_clock][0] = u;
75         ++dfs_clock;
76         for (int v : G[u]) {
77             if (v == fa)
78                 continue;
79             dfs(v, u);
80             st[dfs_clock][0] = u;
81             ++dfs_clock;
82         }
83     }
84     void build(int rt) {
85         dfs_clock = 0;
86         int g = lg[m];
87         for (int i = 0; i < m; ++i)
88             st[i].resize(g + 1);
89         dfs(rt, rt);
90         for (int j = 1; j <= g; ++j) {
91             for (int i = 0; i + (1 << (j - 1)) < m; ++i) {
92                 if (dep[st[i][j - 1]] < dep[st[i + (1 << (j - 1))][j - 1]])
93                     st[i][j] = st[i][j - 1];
94                 else
95                     st[i][j] = st[i + (1 << (j - 1))][j - 1];
96             }
97         }
98     }
99     int query(int u, int v) {
100         int l = dfn[u], r = dfn[v];
101         if (l > r)
102             swap(l, r);
103         int g = lg[r - l + 1];
104         if (dep[st[l][g]] < dep[st[r - (1 << g) + 1][g]])
105             return st[l][g];
106         else
107             return st[r - (1 << g) + 1][g];
108     }
109 };
110

```

```

111 template <typename T>
112 struct ExKruskal {
113     struct Edge {
114         int u, v;
115         T w;
116         Edge() {}
117         Edge(int _u, int _v, T _w)
118             : u(_u), v(_v), w(_w) {}
119         bool operator<(const Edge& e) const { return w < e.w; }
120     };
121     int n, m;
122     vector<Edge> E;
123     vector<T> W;
124     vector<int> f;
125     LCAGraph G;
126
127     int find(int x) { return x == f[x] ? x : f[x] = find(f[x]); }
128
129     ExKruskal(int _n = 0)
130         : n(_n), W(n + n - 1), f(n + n - 1), G(n + n - 1) {
131         iota(f.begin(), f.end(), 0);
132     }
133
134     void addedge(int u, int v, T w) { E.emplace_back(u, v, w); }
135
136     void build() {
137         sort(E.begin(), E.end());
138         int id = n - 1, cnt = 0;
139         for (auto& [u, v, w] : E) {
140             u = find(u);
141             v = find(v);
142             if (u != v) {
143                 ++id;
144                 G.addedge(id, u);
145                 G.addedge(id, v);
146                 f[u] = f[v] = id;
147                 W[id] = w;
148                 ++cnt;
149                 if (cnt == n - 1)
150                     break;
151             }
152         }
153         G.build(id);
154     }
155
156     int query(int u, int v) {
157         int lca = G.query(u, v);
158         return W[lca];
159     }
160 };
161
162 int n, m, q;
163 ull a1, a2;
164 void solve(int Case) {
165     rd(n), rd(m);
166     ExKruskal<int> ek(n);
167
168     int u, v, w;
169     for (int i = 1; i <= m; ++i) {
170         rd(u), rd(v), rd(w);
171         --u;
172         --v;
173         ek.addedge(u, v, w);
174     }
175

```

```

176   ek.build();
177
178   rd(q), rd(a1), rd(a2);
179
180   int ans = 0;
181   for (int i = 1; i <= q; ++i) {
182       pair<int, int> p = myRanq(a1, a2, n);
183       --p.first;
184       --p.second;
185       ans ^= ek.query(p.first, p.second);
186   }
187   printf("%d\n", ans);
188 }
189
190 int main() {
191     #ifdef BACKLIGHT
192         freopen("a.in", "r", stdin);
193     #endif
194     int T = 1;
195     // cin >> T;
196     for (int t = 1; t <= T; ++t)
197         solve(t);
198     return 0;
199 }

```

## 2.10 FullyDCP

```

1  // Got this code from LOJ
2  #include <bits/stdc++.h>
3  using namespace std;
4
5  struct Xor128 {
6      unsigned x, y, z, w;
7      Xor128()
8          : x(123456789), y(362436069), z(521288629), w(88675123) {}
9      unsigned next() {
10         unsigned t = x ^ (x << 11);
11         x = y;
12         y = z;
13         z = w;
14         return w = w ^ (w >> 19) ^ (t ^ (t >> 8));
15     }
16     //手回き
17     inline unsigned next(unsigned n) { return next() % n; }
18 };
19
20 // bottom up な Treap
21 //脱再回!
22 // randomized binary search するには choiceRandomly を
23 // bool choiceRandomly(Ref l, Ref r) { return rng.next(l->size + r->size) < l->size; }
24 //に書き回えるだけでよい。
25 template <typename Node>
26 struct BottomupTreap {
27     Xor128 rng;
28     typedef Node* Ref;
29     static int size(Ref t) { return !t ? 0 : t->size; }
30
31     unsigned nextRand() { return rng.next(); }
32
33 private:
34     bool choiceRandomly(Ref l, Ref r) { return l->priority < r->priority; }
35
36 public:
37     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 のオーバーフローを防ぐために再帰する。
49             // あくまでセグ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);

```

```

103     t = p;
104 }
105 return RefPair(l, r);
106 }
107 //  $l < t < r$  の  $(l, t, r)$  に分割する。 $(l, r)$  を返す
108 RefPair split3(Ref t) {
109     Ref p, l = t->left, r = t->right;
110     Node::cut(l), Node::cut(r);
111     t->linklr(NULL, NULL);
112     while (p = t->parent) {
113         t->parent = NULL;
114         if (p->left == t)
115             r = p->linkl(r);
116         else
117             l = p->linkr(l);
118         t = p;
119     }
120     return RefPair(l, r);
121 }
122 Ref cons(Ref h, Ref t) {
123     assert(size(h) == 1);
124     if (!t)
125         return h;
126     Ref u = NULL;
127     while (true) {
128         if (choiceRandomly(h, t)) {
129             Ref p = t->parent;
130             u = h->linkr(t);
131             t = p;
132             break;
133         }
134         Ref l = t->left;
135         if (!l) {
136             u = h;
137             break;
138         }
139         t = l;
140     }
141     while (t) {
142         u = t->linkl(u);
143         t = t->parent;
144     }
145     return u;
146 }
147 };
148
149 // free tree のために、 $\mathbb{F}$  を基本として  $\mathbb{F}$  う
150 class EulerTourTreeWithMarks {
151     struct Node {
152         typedef BottomupTreap<Node> BST;
153
154         Node *left, *right, *parent;
155         int size;
156         unsigned priority;
157         char marks, markUnions; // 0 ビット目が edgeMark, 1 ビット目が vertexMark
158
159         Node()
160             : left(NULL), right(NULL), parent(NULL), size(1), priority(0), marks(0), markUnions(0) {}
161
162         inline Node* update() {
163             int size_t = 1, markUnions_t = marks;
164             if (left) {
165                 size_t += left->size;
166                 markUnions_t |= left->markUnions;
167             }

```

```

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④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④して④④する頂点はただか 1 つである

```



```

233 std::vector<int> firstArc;
234 // 頂点に属する属性
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⌈⌋するようになる
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⌈⌋を m の先頭 = l の末尾に insert
338     m = bst.cons(&nodes[a2], m);
339     // v→w の⌈⌋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⌈⌋するようになる
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⌈⌋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⌈⌋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;

```

```

363     }
364
365     // firstArc を必要に☐じて書き☐える
366     if (firstArc[v] == a1) {
367         int b;
368         if (r != NULL) {
369             // v が根じゃないなら右側の最初の☐でよい
370             b = getArcIndex(Node::findHead(r));
371         } else {
372             // v が根なら最初の☐でよい。孤立点になるなら-1
373             b = !l ? -1 : getArcIndex(Node::findHead(l));
374         }
375         firstArc[v] = b;
376         firstArcChanged(v, a1, b);
377     }
378     if (firstArc[w] == a2) {
379         // w が根になるので最初の☐でよい。孤立点になるなら-1
380         int b = !m ? -1 : getArcIndex(Node::findHead(m));
381         firstArc[w] = b;
382         firstArcChanged(w, a2, b);
383     }
384
385     bst.join(l, r);
386 }
387
388 void changeEdgeMark(int ti, bool b) {
389     assert(ti < numEdges());
390     edgeMark[ti] = b;
391     Node* t = &nodes[ti];
392     t->marks = (b << 0) | (t->marks & (1 << 1));
393     Node::updatePath(t);
394 }
395 void changeVertexMark(int v, bool b) {
396     vertexMark[v] = b;
397     int a = firstArc[v];
398     if (a != -1) {
399         Node* t = &nodes[a];
400         t->marks = (t->marks & (1 << 0)) | (b << 1);
401         Node::updatePath(t);
402     }
403 }
404
405 template <typename Callback>
406 bool enumMarkedEdges(TreeRef tree, Callback callback) const {
407     return enumMarks<0, Callback>(tree, callback);
408 }
409 //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
410 template <typename Callback>
411 bool enumMarkedVertices(TreeRef tree, Callback callback) const {
412     return enumMarks<1, Callback>(tree, callback);
413 }
414
415 private:
416 // callback : TreeEdgeIndex×2 -> Bool
417 //引数は頂点をそこからの incident arc で示し、"(正方向 ? 0 : N-1) +
418 // treeEdgeIndex" を表す。方向は v,w の大小で☐理すればよい
419 // 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。
420 template <int Mark, typename Callback>
421 bool enumMarks(TreeRef tree, Callback callback) const {
422     if (tree.isIsolatedVertex())
423         return true;
424     const Node* t = tree.ref;
425     if (t->markUnions >> Mark & 1)
426         return enumMarksRec<Mark, Callback>(t, callback);
427     else

```

```

428     return true;
429 }
430
431 //平衡木なので深さは深くないので再帰して問題ない
432 template <int Mark, typename Callback>
433 bool enumMarksRec(const Node* t, Callback callback) const {
434     const Node *l = t->left, *r = t->right;
435     if (l && (l->markUnions >> Mark & 1))
436         if (!enumMarksRec<Mark, Callback>(l, callback))
437             return false;
438     if (t->marks >> Mark & 1)
439         if (!callback(getArcIndex(t)))
440             return false;
441     if (r && (r->markUnions >> Mark & 1))
442         if (!enumMarksRec<Mark, Callback>(r, callback))
443             return false;
444     return true;
445 }
446
447 public:
448     //デバッグ用
449     void debugEnumEdges(std::vector<int>& out_v) const {
450         int M = numEdges();
451         for (int ti = 0; ti < M; ti++) {
452             const Node* t = &nodes[ti];
453             if (t->left || t->right || t->parent)
454                 out_v.push_back(ti);
455         }
456     }
457 };
458
459 // 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。
460 //ところで"level up" って和英語なんだ。promote でいいかな。
461 // Sampling heuristic ランダムケツで超速く (4 倍とか) なったんだけど! いいね!
462 //
463 // References
464 // · Holm, Jacob, Kristian De Lichtenberg, and Mikkel Thorup. "Poly-Logarithmic deterministic fully-dynamic
465 // algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity." Journal of the ACM
466 // (JACM) 48.4 (2001): 723-760. · Iyer, Raj, et al. "An experimental study of polylogarithmic, fully dynamic,
467 // connectivity algorithms." Journal of Experimental Algorithmics (JEA) 6 (2001): 4.
468
469 class HolmDeLichtenbergThorup {
470     typedef HolmDeLichtenbergThorup This;
471     typedef EulerTourTreeWithMarks Forest;
472     typedef Forest::TreeRef TreeRef;
473
474     int numVertices_m;
475     int numSamplings;
476
477     // DynamicTree はコピー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...
478     std::vector<Forest> forests;
479
480     std::vector<char> edgeLevel;
481     std::vector<int> treeEdgeIndex;           // : EdgeIndex -> TreeEdgeIndex
482     std::vector<int> treeEdgeMap;           // : TreeEdgeIndex -> EdgeIndex
483     std::vector<int> treeEdgeIndexFreeList; // : [TreeEdgeIndex]
484
485     // arc も方向は EulerTourTree と同じように v,w の大小に合わせる
486     std::vector<int> arcHead;
487
488     std::vector<std::vector<int>> firstIncidentArc;
489     std::vector<int> nextIncidentArc, prevIncidentArc;
490
491     //一時的に使う。使い回して使う
492     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  $\mathbb{F}$  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  $\mathbb{F}$  にレベルアップさせる。edgeVisited の後  $\mathbb{F}$  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  $\mathbb{F}$  を列  $\mathbb{F}$  する
545     forest.enumMarkedEdges(uRoot, EnumLevelTreeEdges(this));
546     // 列  $\mathbb{F}$  した  $T_u$  の  $\mathbb{F}$  を  $\mathbb{F}$  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     forests[lv + 1].link(ti, v, w);
559 }
560 visitedEdges.clear();
561
562 if (replacementEdge != -1) {
563     // T_u の E 列 E の前に構造が E 変わると困るので replacementEdge はこのタイミングで E 理する
564     deleteNontreeEdge(replacementEdge);
565     addTreeEdge(replacementEdge);
566     return true;
567 } else if (lv > 0) {
568     return replace(lv - 1, v, w);
569 } else {
570     return false;
571 }
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 E な E 理をするなどしなければいけないのでヘルパ E
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 E 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 を図理する
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         // 図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図クする
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;

```

```

688     prevIncidentArc[a] = -1;
689     if (next != -1)
690         prevIncidentArc[next] = a;
691
692     if (next == -1)
693         forests[lv].changeVertexMark(v, true);
694 }
695
696 void deleteIncidentArc(int a, int v) {
697     int ei = arcEdge(a);
698     int lv = edgeLevel[ei];
699     assert(treeEdgeIndex[ei] == -1);
700
701     int next = nextIncidentArc[a], prev = prevIncidentArc[a];
702     nextIncidentArc[a] = prevIncidentArc[a] = -2;
703
704     if (next != -1)
705         prevIncidentArc[next] = prev;
706     if (prev != -1)
707         nextIncidentArc[prev] = next;
708     else
709         firstIncidentArc[lv][v] = next;
710
711     if (next == -1 && prev == -1)
712         forests[lv].changeVertexMark(v, false);
713 }
714
715 void insertNontreeEdge(int ei) {
716     int a1 = arc1(ei), a2 = arc2(ei);
717     insertIncidentArc(a1, arcHead[a2]);
718     insertIncidentArc(a2, arcHead[a1]);
719 }
720
721 void deleteNontreeEdge(int ei) {
722     int a1 = arc1(ei), a2 = arc2(ei);
723     deleteIncidentArc(a1, arcHead[a2]);
724     deleteIncidentArc(a2, arcHead[a1]);
725 }
726
727 public:
728     HolmDeLichtenbergThorup()
729         : numVertices_m(0), numSamplings(0) {}
730
731     int numVertices() const { return numVertices_m; }
732     int numMaxEdges() const { return edgeLevel.size(); }
733
734     void init(int N, int M) {
735         numVertices_m = N;
736
737         int levels = 1;
738         while (1 << levels <= N / 2)
739             levels++;
740
741         //サンプリング数を設定する。適切なFはよくわからない
742         numSamplings = (int)(levels * 1);
743
744         forests.resize(levels);
745         for (int lv = 0; lv < levels; lv++)
746             forests[lv].init(N);
747
748         edgeLevel.assign(M, -1);
749
750         treeEdgeIndex.assign(M, -1);
751         treeEdgeMap.assign(N - 1, -1);
752

```



```

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いない
816         if (v != w)
817             deleteNontreeEdge(ei);

```

```

818     }
819
820     arcHead[a1] = arcHead[a2] = -1;
821     edgeLevel[ei] = -1;
822
823     return splitted;
824 }
825
826 bool isConnected(int v, int w) const { return forests[0].isConnected(v, w); }
827 };
828 typedef HolmDeLichtenbergThorup FullyDynamicConnectivity;
829 map<int, map<int, int>> mp;
830
831 int main() {
832     int n, m;
833     scanf("%d%d", &n, &m);
834     mp.clear();
835     FullyDynamicConnectivity fdc;
836     fdc.init(n + 1, m + 1);
837     int posE = 0;
838     int lstans = 0;
839     for (int i = 1, op, u, v, _u, _v; i <= m; ++i) {
840         scanf("%d%d%d", &op, &u, &v);
841         u ^= lstans;
842         v ^= lstans;
843         _u = u, _v = v;
844         if (u < v)
845             swap(u, v);
846         if (op == 0) {
847             mp[u][v] = ++posE;
848             fdc.insertEdge(posE, u, v);
849         } else if (op == 1) {
850             fdc.deleteEdge(mp[u][v]);
851             mp[u].erase(v);
852         } else {
853             int ok = fdc.isConnected(u, v);
854             if (ok)
855                 lstans = _u;
856             else
857                 lstans = _v;
858             printf("%c\n", "NY"[ok]);
859         }
860     }
861     return 0;
862 }

```

## 2.11 Graph

```

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 } // namespace Backlight

```

---

## 2.12 GraphMatch

---

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  // graph
5  template <typename T>
6  class graph {
7  public:
8      struct edge {
9          int from;
10         int to;
11         T cost;
12     };
13     vector<edge> edges;
14     vector<vector<int>> g;
15     int n;
16     graph(int _n)
17         : n(_n) {
18         g.resize(n);
19     }
20     virtual int add(int from, int to, T cost) = 0;
21 };
22
23 // undirectedgraph
24 template <typename T>
25 class undirectedgraph : public graph<T> {
26 public:
27     using graph<T>::edges;
28     using graph<T>::g;
29     using graph<T>::n;
30
31     undirectedgraph(int _n)
32         : graph<T>(_n) {
33     }
34     int add(int from, int to, T cost = 1) {
35         assert(0 <= from && from < n && 0 <= to && to < n);
36         int id = (int)edges.size();
37         g[from].push_back(id);
38         g[to].push_back(id);
39         edges.push_back({from, to, cost});
40         return id;
41     }
42 };
43
44 // blossom / find_max_unweighted_matching
45 template <typename T>
46 vector<int> find_max_unweighted_matching(const undirectedgraph<T>& g) {

```

```

47  std::mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
48  vector<int> match(g.n, -1);    // 匹配
49  vector<int> aux(g.n, -1);      // 时间戳记
50  vector<int> label(g.n);        // "o" or "i"
51  vector<int> orig(g.n);         // 花根
52  vector<int> parent(g.n, -1);   // 父节点
53  queue<int> q;
54  int aux_time = -1;
55
56  auto lca = [&](int v, int u) {
57      aux_time++;
58      while (true) {
59          if (v != -1) {
60              if (aux[v] == aux_time) { // 找到拜访过的点 也就是 LCA
61                  return v;
62              }
63              aux[v] = aux_time;
64              if (match[v] == -1) {
65                  v = -1;
66              } else {
67                  v = orig[parent[match[v]]]; // 以匹配点的父节点继续寻找
68              }
69          }
70          swap(v, u);
71      }
72  }; // lca
73
74  auto blossom = [&](int v, int u, int a) {
75      while (orig[v] != a) {
76          parent[v] = u;
77          u = match[v];
78          if (label[u] == 1) { // 初始点设为"o" 找增广路
79              label[u] = 0;
80              q.push(u);
81          }
82          orig[v] = orig[u] = a; // 缩花
83          v = parent[u];
84      }
85  }; // blossom
86
87  auto augment = [&](int v) {
88      while (v != -1) {
89          int pv = parent[v];
90          int next_v = match[pv];
91          match[v] = pv;
92          match[pv] = v;
93          v = next_v;
94      }
95  }; // augment
96
97  auto bfs = [&](int root) {
98      fill(label.begin(), label.end(), -1);
99      iota(orig.begin(), orig.end(), 0);
100     while (!q.empty()) {
101         q.pop();
102     }
103     q.push(root);
104     // 初始点设为 "o", 这里以 "0" 代替"o", "1" 代替"i"
105     label[root] = 0;
106     while (!q.empty()) {
107         int v = q.front();
108         q.pop();
109         for (int id : g.g[v]) {
110             auto& e = g.edges[id];
111             int u = e.from ^ e.to ^ v;

```

```

112     if (label[u] == -1) { // 找到未拜访点
113         label[u] = 1; // 标记 "i"
114         parent[u] = v;
115         if (match[u] == -1) { // 找到未匹配点
116             augment(u); // 寻找增广路径
117             return true;
118         }
119         // 找到已匹配点 将与她匹配的点丢入 queue 延伸交错树
120         label[match[u]] = 0;
121         q.push(match[u]);
122         continue;
123     } 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"
124         int a = lca(orig[v], orig[u]);
125         // 找 LCA 然后缩花
126         blossom(u, v, a);
127         blossom(v, u, a);
128     }
129 }
130 }
131 return false;
132 }; // bfs
133
134 auto greedy = [&]() {
135     vector<int> order(g.n);
136     // 随机打乱 order
137     iota(order.begin(), order.end(), 0);
138     shuffle(order.begin(), order.end(), rng);
139
140     // 将可以匹配的点匹配
141     for (int i : order) {
142         if (match[i] == -1) {
143             for (auto id : g.g[i]) {
144                 auto& e = g.edges[id];
145                 int to = e.from ^ e.to ^ i;
146                 if (match[to] == -1) {
147                     match[i] = to;
148                     match[to] = i;
149                     break;
150                 }
151             }
152         }
153     }
154 }; // greedy
155
156 // 一开始先随机匹配
157 greedy();
158 // 对未匹配点找增广路
159 for (int i = 0; i < g.n; i++) {
160     if (match[i] == -1) {
161         bfs(i);
162     }
163 }
164 return match;
165 }
166 int main() {
167     ios::sync_with_stdio(0), cin.tie(0);
168     int n, m;
169     cin >> n >> m;
170     undirectedgraph<int> g(n);
171     int u, v;
172     for (int i = 0; i < m; i++) {
173         cin >> u >> v;
174         u--;
175         v--;
176         g.add(u, v, 1);

```

```

177 }
178 auto blossom_match = find_max_unweighted_matching(g);
179 vector<int> ans;
180 int tot = 0;
181 for (int i = 0; i < blossom_match.size(); i++) {
182     ans.push_back(blossom_match[i]);
183     if (blossom_match[i] != -1) {
184         tot++;
185     }
186 }
187 cout << (tot >> 1) << "\n";
188 for (auto x : ans) {
189     cout << x + 1 << " ";
190 }
191 }

```

---

## 2.13 HLD-Edge

---

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 const int N = 2e5 + 5;
5
6 int n, q;
7
8 struct edge {
9     int v, w, nxt;
10 } e[N << 1];
11 int tot, head[N];
12 void init_graph(int n) {
13     tot = 0;
14     fill(head + 1, head + 1 + n, 0);
15 }
16 void add(int u, int v, int w) {
17     ++tot;
18     e[tot] = (edge){v, w, head[u]};
19     head[u] = tot;
20 }
21
22 int sz[N], son[N], h[N], f[N], w[N];
23 void dfs1(int u, int fa) {
24     h[u] = h[fa] + 1;
25     f[u] = fa;
26     sz[u] = 1;
27     son[u] = 0;
28     for (int i = head[u]; i; i = e[i].nxt) {
29         int v = e[i].v;
30         if (v == fa)
31             continue;
32         w[v] = e[i].w;
33         dfs1(v, u);
34         sz[u] += sz[v];
35         if (sz[v] > sz[son[u]])
36             son[u] = v;
37     }
38 }
39 int dfs_clock, dfn[N], rk[N], top[N];
40 void dfs2(int u, int fa, int tp) {
41     ++dfs_clock;
42     dfn[dfs_clock] = w[u];
43     rk[u] = dfs_clock;
44     top[u] = tp;
45     if (son[u])
46         dfs2(son[u], u, tp);

```

```

47     for (int i = head[u]; i; i = e[i].nxt) {
48         int v = e[i].v;
49         if (v == fa || v == son[u])
50             continue;
51         dfs2(v, u, v);
52     }
53 }
54
55 #define mid ((l + r) >> 1)
56 #define lc (x << 1)
57 #define rc (x << 1 | 1)
58 #define lson lc, l, mid
59 #define rson rc, mid + 1, r
60 int sum[N << 2], ma[N << 2], mi[N << 2], tag_inv[N << 2];
61 void push_up(int x) {
62     sum[x] = sum[lc] + sum[rc];
63     ma[x] = max(ma[lc], ma[rc]);
64     mi[x] = min(mi[lc], mi[rc]);
65 }
66 void push_down(int x) {
67     if (tag_inv[x] != 1) {
68         sum[lc] = -sum[lc];
69         swap(ma[lc], mi[lc]);
70         ma[lc] = -ma[lc];
71         mi[lc] = -mi[lc];
72         tag_inv[lc] = -tag_inv[lc];
73
74         sum[rc] = -sum[rc];
75         swap(ma[rc], mi[rc]);
76         ma[rc] = -ma[rc];
77         mi[rc] = -mi[rc];
78         tag_inv[rc] = -tag_inv[rc];
79
80         tag_inv[x] = 1;
81     }
82 }
83 void build(int x, int l, int r) {
84     tag_inv[x] = 1;
85     if (l == r) {
86         sum[x] = ma[x] = mi[x] = dfn[l];
87         return;
88     }
89     build(lson);
90     build(rson);
91     push_up(x);
92 }
93
94 void update(int x, int l, int r, int p, int w) {
95     if (l == r) {
96         sum[x] = ma[x] = mi[x] = w;
97         return;
98     }
99     push_down(x);
100     if (p <= mid)
101         update(lson, p, w);
102     else
103         update(rson, p, w);
104     push_up(x);
105 }
106
107 void inverse(int x, int l, int r, int L, int R) {
108     if (l == L && r == R) {
109         sum[x] = -sum[x];
110         swap(ma[x], mi[x]);
111         ma[x] = -ma[x];

```

```

112     mi[x] = -mi[x];
113     tag_inv[x] = -tag_inv[x];
114     return;
115 }
116 push_down(x);
117 if (R <= mid)
118     inverse(lson, L, R);
119 else if (L > mid)
120     inverse(rson, L, R);
121 else {
122     inverse(lson, L, mid);
123     inverse(rson, mid + 1, R);
124 }
125 push_up(x);
126 }
127
128 int getsum(int x, int l, int r, int L, int R) {
129     if (l == L && r == R)
130         return sum[x];
131     push_down(x);
132     if (R <= mid)
133         return getsum(lson, L, R);
134     else if (L > mid)
135         return getsum(rson, L, R);
136     return getsum(lson, L, mid) + getsum(rson, mid + 1, R);
137 }
138
139 int getmax(int x, int l, int r, int L, int R) {
140     if (l == L && r == R)
141         return ma[x];
142     push_down(x);
143     if (R <= mid)
144         return getmax(lson, L, R);
145     else if (L > mid)
146         return getmax(rson, L, R);
147     return max(getmax(lson, L, mid), getmax(rson, mid + 1, R));
148 }
149
150 int getmin(int x, int l, int r, int L, int R) {
151     if (l == L && r == R)
152         return mi[x];
153     push_down(x);
154     if (R <= mid)
155         return getmin(lson, L, R);
156     else if (L > mid)
157         return getmin(rson, L, R);
158     return min(getmin(lson, L, mid), getmin(rson, mid + 1, R));
159 }
160
161 void INVERSE(int u, int v) {
162     while (top[u] != top[v]) {
163         if (h[top[u]] < h[top[v]])
164             swap(u, v);
165         inverse(1, 1, n, rk[top[u]], rk[u]);
166         u = f[top[u]];
167     }
168     if (h[u] != h[v]) {
169         if (h[u] > h[v])
170             swap(u, v);
171         inverse(1, 1, n, rk[son[u]], rk[v]);
172     }
173 }
174
175 int QSUM(int u, int v) {
176     int res = 0;

```



```
177 while (top[u] != top[v]) {
178     if (h[top[u]] < h[top[v]])
179         swap(u, v);
180     res += getsum(1, 1, n, rk[top[u]], rk[u]);
181     u = f[top[u]];
182 }
183 if (h[u] != h[v]) {
184     if (h[u] > h[v])
185         swap(u, v);
186     res += getsum(1, 1, n, rk[son[u]], rk[v]);
187 }
188 return res;
189 }
190
191 int QMAX(int u, int v) {
192     int res = INT_MIN;
193     while (top[u] != top[v]) {
194         if (h[top[u]] < h[top[v]])
195             swap(u, v);
196         res = max(res, getmax(1, 1, n, rk[top[u]], rk[u]));
197         u = f[top[u]];
198     }
199     if (h[u] != h[v]) {
200         if (h[u] > h[v])
201             swap(u, v);
202         res = max(res, getmax(1, 1, n, rk[son[u]], rk[v]));
203     }
204     return res;
205 }
206
207 int QMIN(int u, int v) {
208     int res = INT_MAX;
209     while (top[u] != top[v]) {
210         if (h[top[u]] < h[top[v]])
211             swap(u, v);
212         res = min(res, getmin(1, 1, n, rk[top[u]], rk[u]));
213         u = f[top[u]];
214     }
215     if (h[u] != h[v]) {
216         if (h[u] > h[v])
217             swap(u, v);
218         res = min(res, getmin(1, 1, n, rk[son[u]], rk[v]));
219     }
220     return res;
221 }
222
223 int tu[N], tv[N];
224 void solve(int Case) {
225     /* write code here */
226     /* gl & hf */
227     scanf("%d", &n);
228     int u, v, w;
229     for (int i = 1; i <= n - 1; ++i) {
230         scanf("%d %d %d", &u, &v, &w);
231         ++u, ++v;
232         add(u, v, w);
233         add(v, u, w);
234
235         tu[i] = u;
236         tv[i] = v;
237     }
238
239     dfs1(1, 1);
240     dfs2(1, 1, 1);
241 }
```

```

242 build(1, 1, n);
243
244 scanf("%d", &q);
245 char op[5];
246 int x, y;
247 for (int i = 1; i <= q; ++i) {
248     scanf("%s %d %d", op, &x, &y);
249     ++x, ++y;
250     if (op[0] == 'C') {
251         --x, --y;
252         int id = h[tu[x]] > h[tv[x]] ? tu[x] : tv[x];
253         update(1, 1, n, rk[id], y);
254     } else if (op[0] == 'N') {
255         INVERSE(x, y);
256     } else if (op[0] == 'S') {
257         printf("%d\n", QSUM(x, y));
258     } else if (op[1] == 'A') {
259         printf("%d\n", QMAX(x, y));
260     } else if (op[1] == 'I') {
261         printf("%d\n", QMIN(x, y));
262     }
263 }
264 }
265
266 int main() {
267     int T = 1;
268     for (int _ = 1; _ <= T; _++)
269         solve(_);
270     return 0;
271 }

```

## 2.14 Kosaraju

```

1  const int N = 1e5 + 5;
2  vector<int> G[N], R[N];
3  void init(int n) {
4      for (int i = 1; i <= n; ++i)
5          G[i].clear(), R[i].clear();
6  }
7  inline void addarc(int u, int v) {
8      G[u].push_back(v);
9      R[v].push_back(u);
10 }
11
12 int n, m;
13 int dfs_clock, scc_cnt;
14 int dfn[N], belong[N];
15 bool vis[N];
16 void dfs1(int u) {
17     vis[u] = true;
18     for (const int& v : G[u]) {
19         if (!vis[v])
20             dfs1(v);
21     }
22     dfn[++dfs_clock] = u;
23 }
24 void dfs2(int u) {
25     belong[u] = scc_cnt;
26     for (const int& v : R[u]) {
27         if (!belong[v])
28             dfs2(v);
29     }
30 }
31 void kosaraju(){

```

```

32 dfs_clock = scc_cnt = 0;
33 fill(dfn + 1, dfn + 1 + n, 0);
34 fill(belong + 1, belong + 1 + n, 0);
35 fill(vis + 1, vis + 1 + n, false);
36 for (int i = 1; i <= n; ++i) {
37     if (!vis[i])
38         dfs1(i);
39 }
40
41 for (int i = n; i >= 1; --i) {
42     if (!belong[dfn[i]]) {
43         ++scc_cnt;
44         dfs2(dfn[i]);
45     }
46 }
47 }

```

---

## 2.15 Kruskal

---

```

1 namespace Backlight {
2
3 template <typename T>
4 struct Wraph {
5     struct Edge {
6         int u, v;
7         T w;
8         Edge() {}
9         Edge(int _u, int _v, T _w)
10             : u(_u), v(_v), w(_w) {}
11         bool operator<(const Edge& e) {
12             return w < e.w;
13         }
14     };
15
16     int V;
17     vector<vector<Edge>> G;
18     vector<Edge> E;
19
20     Wraph()
21         : V(0) {}
22     Wraph(int _V)
23         : V(_V), G(_V + 1) {}
24
25     inline void addarc(int u, int v, T w) {
26         assert(1 <= u && u <= V);
27         assert(1 <= v && v <= V);
28         G[u].push_back(Edge(u, v, w));
29         E.push_back(Edge(u, v, w));
30     }
31
32     inline void addedge(int u, int v, T w) {
33         addarc(u, v, w);
34         addarc(v, u, w);
35     }
36
37     /*****/
38     T kruskal() {
39         vector<int> fa(V + 1);
40         for (int i = 1; i <= V; ++i)
41             fa[i] = i;
42
43         auto find = [&fa](auto self, int x) {
44             if (x == fa[x])
45                 return x;

```

```

46     fa[x] = self(self, fa[x]);
47     return fa[x];
48 };
49
50 auto merge = [&fa, find](int x, int y) {
51     x = find(find, x);
52     y = find(find, y);
53     if (x == y)
54         return false;
55     fa[x] = y;
56     return true;
57 };
58
59 T cost = 0;
60 int cnt = 0;
61 sort(E.begin(), E.end());
62 for (int i = 0; i < (int)E.size(); ++i) {
63     Edge e = E[i];
64     if (merge(e.u, e.v)) {
65         cost = e.w;
66         ++cnt;
67         if (cnt == V - 1)
68             break;
69     }
70 }
71 return cost;
72 }
73 };
74
75 } // namespace BackLight

```

## 2.16 LCA-HLD

```

1  int tot, head[N];
2  struct Edge {
3      int v, nxt;
4  } e[M];
5
6  void addedge(int u, int v) {
7      ++tot;
8      e[tot] = (Edge){v, head[u]};
9      head[u] = tot;
10     ++tot;
11     e[tot] = (Edge){u, head[v]};
12     head[v] = tot;
13 }
14
15 int h[N], f[N], sz[N], son[N], top[N];
16 void dfs1(int u, int fa) {
17     h[u] = h[fa] + 1;
18     f[u] = fa;
19     sz[u] = 1;
20     son[u] = 0;
21     for (int i = head[u]; i; i = e[i].nxt) {
22         int v = e[i].v;
23         if (v == fa)
24             continue;
25         dfs1(v, u);
26         sz[u] += sz[v];
27         if (sz[v] > sz[son[u]])
28             son[u] = v;
29     }
30 }
31

```

```

32 void dfs2(int u, int fa, int tp) {
33     top[u] = tp;
34     if (son[u])
35         dfs2(son[u], u, tp);
36     for (int i = head[u]; i; i = e[i].nxt) {
37         int v = e[i].v;
38         if (v == fa || v == son[u])
39             continue;
40         dfs2(v, u, v);
41     }
42 }
43
44 int LCA(int u, int v) {
45     while (top[u] != top[v]) {
46         if (h[top[u]] < h[top[v]])
47             swap(u, v);
48         u = f[top[u]];
49     }
50     if (h[u] > h[v])
51         swap(u, v);
52     return u;
53 }

```

## 2.17 LCA

```

1 namespace Backlight {
2
3 template <typename T>
4 struct Wraph {
5     struct Edge {
6         int u, v;
7         T w;
8         Edge() {}
9         Edge(int _u, int _v, T _w)
10             : u(_u), v(_v), w(_w) {}
11     };
12
13     int V;
14     vector<vector<Edge>> G;
15
16     Wraph()
17         : V(0) {}
18     Wraph(int _V)
19         : V(_V), G(_V + 1) {}
20
21     inline void addarc(int u, int v, T w = 1) {
22         assert(1 <= u && u <= V);
23         assert(1 <= v && v <= V);
24         G[u].push_back(Edge(u, v, w));
25     }
26
27     inline void addedge(int u, int v, T w = 1) {
28         addarc(u, v, w);
29         addarc(v, u, w);
30     }
31
32     /*****
33     vector<int> dep;
34     vector<T> dis;
35     vector<vector<int>> par;
36     int rt, LG;
37     void dfs(int u, int fa, int d1, int d2) {
38         dep[u] = d1;
39         dis[u] = d2;

```

```

40     if (u == rt) {
41         for (int i = 0; i < LG; ++i)
42             par[u][i] = rt;
43     } else {
44         par[u][0] = fa;
45         for (int i = 1; i < LG; ++i) {
46             par[u][i] = par[par[u][i - 1]][i - 1];
47         }
48     }
49
50     for (Edge& e : G[u]) {
51         int v = e.v;
52         T w = e.w;
53         if (v == fa)
54             continue;
55         dfs(v, u, d1 + 1, d2 + w);
56     }
57 }
58
59 inline void build_lca(int _rt) {
60     rt = _rt;
61     LG = __lg(V + 1) + 1;
62     dep = vector<int>(V + 1);
63     dis = vector<T>(V + 1);
64     par = vector<vector<int>>(V + 1, vector<int>(LG));
65     dfs(rt, rt, 0, 0);
66 }
67
68 inline int jump(int u, int d) {
69     for (int j = LG - 1; j >= 0; --j) {
70         if ((1 << j) & d)
71             u = par[u][j];
72     }
73     return u;
74 }
75
76 int lca(int u, int v) {
77     if (dep[u] < dep[v])
78         swap(u, v);
79     u = jump(u, dep[u] - dep[v]);
80     if (u == v)
81         return u;
82     for (int i = LG - 1; i >= 0; --i) {
83         if (par[u][i] != par[v][i]) {
84             u = par[u][i];
85             v = par[v][i];
86         }
87     }
88     return par[u][0];
89 }
90 };
91
92 }; // namespace Backlight

```

## 2.18 maxflow

```

1 namespace Backlight {
2
3 template <typename Cap>
4 struct mf_graph {
5     static const Cap INF = numeric_limits<Cap>::max();
6
7     struct Edge {
8         int v, nxt;

```

```

9     Cap c, f;
10     Edge() {}
11     Edge(int _v, int _nxt, Cap _c)
12         : v(_v), nxt(_nxt), c(_c), f(0) {}
13 };
14
15 int V, E;
16 vector<int> h;
17 vector<Edge> e;
18
19 mf_graph()
20     : V(0) {}
21 mf_graph(int _V)
22     : V(_V), h(_V + 1, -1) {}
23
24 inline void addarc(int u, int v, Cap c) {
25     assert(1 <= u && u <= V);
26     assert(1 <= v && v <= V);
27     assert(0 <= c);
28
29     e.push_back(Edge(v, h[u], c));
30     h[u] = e.size() - 1;
31 }
32
33 inline void addedge(int u, int v, Cap c) {
34     addarc(u, v, c);
35     addarc(v, u, 0);
36 }
37
38 Cap maxflow(int s, int t) {
39     assert(1 <= s && s <= V);
40     assert(1 <= t && t <= V);
41     assert(s != t);
42
43     vector<int> f(V + 1), d(V + 1);
44
45     auto bfs = [&]() {
46         fill(d.begin(), d.end(), -1);
47         queue<int> q;
48         q.push(s);
49         d[s] = 0;
50         while (!q.empty()) {
51             int u = q.front();
52             q.pop();
53             for (int i = h[u]; i != -1; i = e[i].nxt) {
54                 int v = e[i].v;
55                 if (e[i].c > e[i].f && d[v] == -1) {
56                     d[v] = d[u] + 1;
57                     if (v == t)
58                         break;
59                     q.push(v);
60                 }
61             }
62         }
63         return (d[t] != -1);
64     };
65
66     auto dfs = [&](auto self, int u, Cap up) {
67         if (u == t || up == 0)
68             return up;
69         Cap res = 0;
70         for (int& i = f[u]; i != -1; i = e[i].nxt) {
71             int v = e[i].v;
72             if (d[u] + 1 == d[v]) {
73                 Cap nf = self(self, v, min(up, e[i].c - e[i].f));

```

```

74         if (nf <= 0)
75             continue;
76         up -= nf;
77         res += nf;
78         e[i].f += nf;
79         e[i ^ 1].f -= nf;
80         if (up == 0)
81             break;
82     }
83 }
84 if (res == 0)
85     d[u] = -1;
86 return res;
87 };
88
89 Cap res = 0;
90 while (bfs()) {
91     f = h;
92     res += dfs(dfs, s, INF);
93 }
94 return res;
95 }
96 };
97
98 } // namespace Backlight

```

## 2.19 mincostflow

```

1 namespace Backlight {
2
3 template <typename Cap, typename Cost>
4 struct mcmf_graph {
5     static const Cap INF = numeric_limits<Cap>::max();
6
7     struct Edge {
8         int v, nxt;
9         Cap cap, flow;
10        Cost cost;
11        Edge() {}
12        Edge(int _v, int _nxt, Cap _cap, Cost _cost)
13            : v(_v), nxt(_nxt), cap(_cap), flow(0), cost(_cost) {}
14    };
15
16    int V, E;
17    vector<int> h;
18    vector<Edge> e;
19
20    mcmf_graph()
21        : V(0) {}
22    mcmf_graph(int _V)
23        : V(_V), h(_V + 1, -1) {}
24
25    inline void addarc(int u, int v, Cap cap, Cost cost) {
26        assert(1 <= u && u <= V);
27        assert(1 <= v && v <= V);
28        e.push_back(Edge(v, h[u], cap, cost));
29        h[u] = e.size() - 1;
30    }
31
32    inline void addedge(int u, int v, Cap cap, Cost cost) {
33        addarc(u, v, cap, cost);
34        addarc(v, u, 0, -cost);
35    }
36

```



```

37 pair<Cap, Cost> mcmf(int s, int t) {
38     assert(1 <= s && s <= V);
39     assert(1 <= t && t <= V);
40     assert(s != t);
41
42     Cap flow = 0;
43     Cost cost = 0;
44
45     vector<int> pe(V + 1);
46     vector<bool> inq(V + 1);
47     vector<Cost> dis(V + 1);
48     vector<Cap> incf(V + 1);
49
50     auto spfa = [&]() {
51         fill(dis.begin(), dis.end(), INF);
52         queue<int> q;
53         q.push(s);
54         dis[s] = 0;
55         incf[s] = INF;
56         incf[t] = 0;
57         while (!q.empty()) {
58             int u = q.front();
59             q.pop();
60             inq[u] = false;
61             for (int i = h[u]; i != -1; i = e[i].nxt) {
62                 int v = e[i].v, _cap = e[i].cap, _cost = e[i].cost;
63                 if (_cap == 0 || dis[v] <= dis[u] + _cost)
64                     continue;
65                 dis[v] = dis[u] + _cost;
66                 incf[v] = min(_cap, incf[u]);
67                 pe[v] = i;
68                 if (!inq[v])
69                     q.push(v), inq[v] = true;
70             }
71         }
72         return incf[t];
73     };
74
75     auto update = [&]() {
76         flow += incf[t];
77         for (int i = t; i != s; i = e[pe[i] ^ 1].v) {
78             e[pe[i]].cap -= incf[t];
79             e[pe[i] ^ 1].cap += incf[t];
80             cost += incf[t] * e[pe[i]].cost;
81         }
82     };
83
84     while (spfa())
85         update();
86
87     return make_pair(flow, cost);
88 }
89 };
90
91 } // namespace Backlight

```

## 2.20 SCC

```

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 scc_clock, scc_cnt;
32 vector<int> dfn, low, belong, scc_size;
33 vector<bool> ins;
34 stack<int> stk;
35
36 void tarjan(int u, int fa) {
37     dfn[u] = low[u] = ++scc_clock;
38     ins[u] = true;
39     stk.push(u);
40
41     // bool flag = false;
42     for (Edge& e : G[u]) {
43         int v = e.v;
44         // if (v == fa && !flag) {
45         //     flag = true;
46         //     continue;
47         // }
48
49         if (!dfn[v]) {
50             tarjan(v, u);
51             low[u] = min(low[u], low[v]);
52         } else if (ins[v])
53             low[u] = min(low[u], dfn[v]);
54     }
55
56     if (dfn[u] == low[u]) {
57         ++scc_cnt;
58         scc_size.push_back(0);
59         int top;
60         do {
61             top = stk.top();
62             stk.pop();
63             ins[top] = false;
64             belong[top] = scc_cnt;
65             ++scc_size[scc_cnt];
66         } while (u != top);
67     }
68 }
69
70 void build_scc() {
71     scc_clock = scc_cnt = 0;

```

```

72     dfn = vector<int>(V + 1);
73     low = vector<int>(V + 1);
74     belong = vector<int>(V + 1);
75     ins = vector<bool>(V + 1);
76     scc_size = vector<int>(1);
77
78     for (int i = 1; i <= V; ++i) {
79         if (!dfn[i])
80             tarjan(i, i);
81     }
82 }
83 };
84
85 } // namespace Backlight

```

---

## 2.21 SPFA

```

1 namespace Backlight {
2
3 template <typename T>
4 struct Wraph {
5     struct Edge {
6         int u, v;
7         T w;
8         Edge() {}
9         Edge(int _u, int _v, T _w)
10             : u(_u), v(_v), w(_w) {}
11     };
12
13     int V;
14     vector<vector<Edge>> G;
15
16     Wraph()
17         : V(0) {}
18     Wraph(int _V)
19         : V(_V), G(_V + 1) {}
20
21     inline void addarc(int u, int v, T w) {
22         assert(1 <= u && u <= V);
23         assert(1 <= v && v <= V);
24         G[u].push_back(Edge(u, v, w));
25     }
26
27     inline void addedge(int u, int v, T w) {
28         addarc(u, v, w);
29         addarc(v, u, w);
30     }
31
32     /*****/
33     vector<T> spfa(int S, T T_MAX) {
34         queue<int> q;
35         vector<T> dis(V + 1, T_MAX);
36         vector<bool> inq(V + 1, 0);
37         q.push(S);
38         dis[S] = 0;
39         while (!q.empty()) {
40             int u = q.front();
41             q.pop();
42             inq[u] = 0;
43             for (Edge e : G[u]) {
44                 if (dis[e.v] > dis[u] + e.w) {
45                     dis[e.v] = dis[u] + e.w;
46                     if (!inq[e.v]) {
47                         inq[e.v] = 1;

```

```

48         q.push(e.v);
49     }
50 }
51 }
52 }
53 return dis;
54 }
55 };
56
57 } // namespace Backlight

```

---

## 2.22 tree-divide

---

```

1 struct Edge {
2     int v, w;
3     Edge() {}
4     Edge(int _v, int _w)
5         : v(_v), w(_w) {}
6 };
7
8 vector<Edge> G[N];
9 inline void addedge(int u, int v, int w) {
10     G[u].push_back(Edge(v, w));
11     G[v].push_back(Edge(u, w));
12 }
13
14 bool vis[N];
15 int sz[N], max_sz[N];
16 void dfs_size(int u, int fa) {
17     sz[u] = 1;
18     max_sz[u] = 0;
19     for (const Edge& e : G[u]) {
20         int v = e.v;
21         if (v == fa || vis[v])
22             continue;
23         dfs_size(v, u);
24         sz[u] += sz[v];
25         max_sz[u] = max(max_sz[u], sz[v]);
26     }
27 }
28
29 int Max, rt;
30 void dfs_root(int r, int u, int fa) {
31     max_sz[u] = max(max_sz[u], sz[r] - sz[u]);
32     if (Max > max_sz[u])
33         Max = max_sz[u], rt = u;
34     for (const Edge& e : G[u]) {
35         int v = e.v;
36         if (v == fa || vis[v])
37             continue;
38         dfs_root(r, v, u);
39     }
40 }
41
42 int dcnt, dis[N];
43 void dfs_dis(int u, int fa, int d) {
44     dis[++dcnt] = d;
45     for (const Edge& e : G[u]) {
46         int v = e.v, w = e.w;
47         if (v == fa || vis[v])
48             continue;
49         dfs_dis(v, u, d + w);
50     }
51 }

```

```

52
53 int ans[K];
54 void calc(int u, int w, int delta) {
55     dcnt = 0;
56     dfs_dis(u, -1, w);
57     for (int i = 1; i <= dcnt; ++i) {
58         for (int j = i + 1; j <= dcnt; ++j) {
59             ans[dis[i] + dis[j]] += delta;
60         }
61     }
62 }
63
64 int n, m;
65 void DFS(int u) {
66     Max = n;
67     dfs_size(u, -1);
68     dfs_root(u, u, -1);
69     vis[rt] = 1;
70     calc(rt, 0, 1);
71     for (const Edge& e : G[rt]) {
72         int v = e.v, w = e.w;
73         if (vis[v])
74             continue;
75         calc(v, w, -1);
76         DFS(v);
77     }
78 }
79
80 void solve() {
81     read(n, m);
82
83     int u, v, w;
84     FOR(i, 2, n) {
85         read(u, v, w);
86         addedge(u, v, w);
87     }
88
89     DFS(1);
90
91     int k;
92     FOR(i, 1, m) {
93         read(k);
94         puts(ans[k] ? "AYE" : "NAY");
95     }
96 }

```

## 2.23 Wraph

```

1 namespace Backlight {
2
3 template <typename T>
4 struct Wraph {
5     struct Edge {
6         int u, v;
7         T w;
8         Edge() {}
9         Edge(int _u, int _v, T _w)
10             : u(_u), v(_v), w(_w) {}
11     };
12
13     int V;
14     vector<vector<Edge>> G;
15
16     Wraph()

```

```

17     : V(0) {}
18     Wraph(int _V)
19     : V(_V), G(_V + 1) {}
20
21     inline void addarc(int u, int v, T w = 1) {
22         assert(1 <= u && u <= V);
23         assert(1 <= v && v <= V);
24         G[u].push_back(Edge(u, v, w));
25     }
26
27     inline void addedge(int u, int v, T w = 1) {
28         addarc(u, v, w);
29         addarc(v, u, w);
30     }
31 };
32
33 } // namespace Backlight

```

## 2.24 WraphMatch

```

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

```



```
175     node[v].prev = -1;
176 }
177 void decrease_key(int h, int v, Key key) {
178     if (!used(v))
179         return push(h, v, key);
180     if (!node[v].prev)
181         node[v].key = key;
182     else {
183         cut(v);
184         node[v].key = key;
185         heap[h] = merge(heap[h], v);
186     }
187 }
188
189 private:
190 void clear_rec(int v) {
191     for (; v; v = node[v].next) {
192         if (node[v].child)
193             clear_rec(node[v].child);
194         node[v].prev = -1;
195     }
196 }
197
198 inline void cut(int v) {
199     auto& n = node[v];
200     int pv = n.prev, nv = n.next;
201     auto& pn = node[pv];
202     if (pn.child == v)
203         pn.child = nv;
204     else
205         pn.next = nv;
206     node[nv].prev = pv;
207     n.next = n.prev = 0;
208 }
209
210 int merge(int l, int r) {
211     if (!l)
212         return r;
213     if (!r)
214         return l;
215     if (node[l].key > node[r].key)
216         swap(l, r);
217     int lc = node[r].next = node[l].child;
218     node[l].child = node[lc].prev = r;
219     return node[r].prev = l;
220 }
221
222 int two_pass_pairing(int root) {
223     if (!root)
224         return 0;
225     int a = root;
226     root = 0;
227     while (a) {
228         int b = node[a].next, na = 0;
229         node[a].prev = node[a].next = 0;
230         if (b)
231             na = node[b].next, node[b].prev = node[b].next = 0;
232         a = merge(a, b);
233         node[a].next = root;
234         root = a;
235         a = na;
236     }
237     int s = node[root].next;
238     node[root].next = 0;
239     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<cost_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) {

```

```

370         if (edges[eid].to == mate[u])
371             max_c = max(max_c, edges[eid].cost);
372     }
373     ret += max_c;
374 }
375 return ret >> 1;
376 }
377
378 inline tcost_t reduced_cost(int u, int v, const Edge& e) const {
379     return tcost_t(potential[u]) + potential[v] - e.cost;
380 }
381
382 void rematch(int v, int w) {
383     int t = mate[v];
384     mate[v] = w;
385     if (mate[t] != v)
386         return;
387     if (link[v].to == surface[link[v].to]) {
388         mate[t] = link[v].from;
389         rematch(mate[t], t);
390     } else {
391         int x = link[v].from, y = link[v].to;
392         rematch(x, y);
393         rematch(y, x);
394     }
395 }
396
397 void fix_mate_and_base(int b) {
398     if (b <= N)
399         return;
400     int bv = base[b], mv = node[bv].link[0].v, bmv = node[bv].link[0].b;
401     int d = (node[bmv].link[1].v == mate[mv]) ? 0 : 1;
402     while (1) {
403         int mv = node[bv].link[d].v, bmv = node[bv].link[d].b;
404         if (node[bmv].link[1 ^ d].v != mate[mv])
405             break;
406         fix_mate_and_base(bv);
407         fix_mate_and_base(bmv);
408         bv = node[bmv].link[d].b;
409     }
410     fix_mate_and_base(base[b] = bv);
411     mate[b] = mate[bv];
412 }
413
414 void reset_time() {
415     time_current_ = 0;
416     event1 = {Inf, 0};
417 }
418
419 void reset_blossom(int b) {
420     label[b] = kFree;
421     link[b].from = 0;
422     slack[b] = Inf;
423     lazy[b] = 0;
424 }
425
426 void reset_all() {
427     label[0] = kFree;
428     link[0].from = 0;
429     for (int v = 1; v <= N; ++v) { // should be optimized for sparse graphs.
430         if (label[v] == kOuter)
431             potential[v] -= time_current_;
432         else {
433             int bv = surface[v];
434             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)

```

```

500     heap2.decrease_key(y, EdgeEvent(t + lazy[y], x, y));
501 } else {
502     int gy = group[y];
503     if (gy != y) {
504         if (t >= slack[gy])
505             return;
506         slack[gy] = t;
507     }
508     heap2s.decrease_key(by, gy, EdgeEvent(t, x, y));
509     if (Lab == kInner)
510         return;
511     EdgeEvent m = heap2s.min(by);
512     heap2.decrease_key(by, EdgeEvent(m.time + lazy[by], m.from, m.to));
513 }
514 }
515
516 void activate_heap2_node(int b) {
517     if (b <= N) {
518         if (slack[b] < Inf)
519             heap2.push(b, EdgeEvent(slack[b] + lazy[b], best_from[b], b));
520     } else {
521         if (heap2s.empty(b))
522             return;
523         EdgeEvent m = heap2s.min(b);
524         heap2.push(b, EdgeEvent(m.time + lazy[b], m.from, m.to));
525     }
526 }
527
528 void swap_blossom(int a, int b) {
529     // Assume that `b` is a maximal blossom.
530     swap(base[a], base[b]);
531     if (base[a] == a)
532         base[a] = b;
533     swap(heavy[a], heavy[b]);
534     if (heavy[a] == a)
535         heavy[a] = b;
536     swap(link[a], link[b]);
537     swap(mate[a], mate[b]);
538     swap(potential[a], potential[b]);
539     swap(lazy[a], lazy[b]);
540     swap(time_created[a], time_created[b]);
541     for (int d = 0; d < 2; ++d)
542         node[node[a].link[d].b].link[1 ^ d].b = b;
543     swap(node[a], node[b]);
544 }
545
546 void set_surface_and_group(int b, int sf, int g) {
547     surface[b] = sf, group[b] = g;
548     if (b <= N)
549         return;
550     for (int bb = base[b]; surface[bb] != sf; bb = node[bb].next_b()) {
551         set_surface_and_group(bb, sf, g);
552     }
553 }
554
555 void merge_smaller_blossoms(int bid) {
556     int lb = bid, largest_size = 1;
557     for (int beta = base[bid], b = beta;;) {
558         if (node[b].size > largest_size)
559             largest_size = node[b].size, lb = b;
560         if ((b = node[b].next_b()) == beta)
561             break;
562     }
563     for (int beta = base[bid], b = beta;;) {
564         if (b != lb)

```

```

565     set_surface_and_group(b, lb, b);
566     if ((b = node[b].next_b()) == beta)
567         break;
568 }
569 group[lb] = lb;
570 if (largest_size > 1) {
571     surface[bid] = heavy[bid] = lb;
572     swap_blossom(lb, bid);
573 } else
574     heavy[bid] = 0;
575 }
576
577 void contract(int x, int y, int eid) {
578     int bx = surface[x], by = surface[y];
579     assert(bx != by);
580     const int h = -(eid + 1);
581     link[surface[mate[bx]]].from = link[surface[mate[by]]].from = h;
582
583     int lca = -1;
584     while (1) {
585         if (mate[by] != 0)
586             swap(bx, by);
587         bx = lca = surface[link[bx].from];
588         if (link[surface[mate[bx]]].from == h)
589             break;
590         link[surface[mate[bx]]].from = h;
591     }
592
593     const int bid = unused_bid[--unused_bid_idx_];
594     assert(unused_bid_idx_ >= 0);
595     int tree_size = 0;
596     for (int d = 0; d < 2; ++d) {
597         for (int bv = surface[x]; bv != lca;) {
598             int mv = mate[bv], bmv = surface[mv], v = mate[mv];
599             int f = link[v].from, t = link[v].to;
600             tree_size += node[bv].size + node[bmv].size;
601             link[mv] = {x, y};
602
603             if (bv > N)
604                 potential[bv] += (time_current_ - time_created[bv]) << 1;
605             if (bmv > N)
606                 heap4.erase(bmv);
607             push_outer_and_fix_potentials(bmv, fix_blossom_potential<kInner>(bmv));
608
609             node[bv].link[d] = {bmv, mv};
610             node[bmv].link[1 ^ d] = {bv, v};
611             node[bmv].link[d] = {bv = surface[f], f};
612             node[bv].link[1 ^ d] = {bmv, t};
613         }
614         node[surface[x]].link[1 ^ d] = {surface[y], y};
615         swap(x, y);
616     }
617     if (lca > N)
618         potential[lca] += (time_current_ - time_created[lca]) << 1;
619     node[bid].size = tree_size + node[lca].size;
620     base[bid] = lca;
621     link[bid] = link[lca];
622     mate[bid] = mate[lca];
623     label[bid] = kOuter;
624     surface[bid] = bid;
625     time_created[bid] = time_current_;
626     potential[bid] = 0;
627     lazy[bid] = 0;
628
629     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;

```

```

760     for (int b = node[old_base].link[d ^ 1].b; b != root;) {
761         label[b] = kSeparated;
762         activate_heap2_node(b);
763         b = node[b].link[d ^ 1].b;
764         label[b] = kSeparated;
765         activate_heap2_node(b);
766         b = node[b].link[d ^ 1].b;
767     }
768     for (int b = old_base;; b = node[b].link[d].b) {
769         label[b] = kInner;
770         int nb = node[b].link[d].b;
771         if (b == root)
772             link[mate[b]] = old_link;
773         else
774             link[mate[b]] = {node[b].link[d].v, node[nb].link[d ^ 1].v};
775         link[surface[mate[b]]] = link[mate[b]]; // fix tree links
776         if (b > N) {
777             if (potential[b] == 0)
778                 expand(b);
779             else
780                 heap4.push(b, time_current_ + (potential[b] >> 1));
781         }
782         if (b == root)
783             break;
784         push_outer_and_fix_potentials(nb, fix_blossom_potential<kInner>(b = nb));
785     }
786 }
787
788 bool augment(int root) {
789     // Return true if an augmenting path is found.
790     while (!que.empty()) {
791         int x = que.dequeue(), bx = surface[x];
792         if (potential[x] == time_current_) {
793             if (x != root)
794                 rematch(x, 0);
795             return true;
796         }
797         for (int eid = ofs[x]; eid < ofs[x + 1]; ++eid) {
798             auto& e = edges[eid];
799             int y = e.to, by = surface[y];
800             if (bx == by)
801                 continue;
802             Label l = label[by];
803             if (l == kOuter) {
804                 cost_t t = reduced_cost(x, y, e) >> 1; // < 2 * Inf
805                 if (t == time_current_) {
806                     contract(x, y, eid);
807                     bx = surface[x];
808                 } else if (t < event1.time) {
809                     heap3.emplace(t, x, eid);
810                 }
811             } else {
812                 tcost_t t = reduced_cost(x, y, e); // < 3 * Inf
813                 if (t >= Inf)
814                     continue;
815                 if (l != kInner) {
816                     if (cost_t(t) + lazy[by] == time_current_) {
817                         if (grow(root, x, y))
818                             return true;
819                     } else
820                         update_heap2<kFree>(x, y, by, t);
821                 } else {
822                     if (mate[x] != y)
823                         update_heap2<kInner>(x, y, by, t);
824                 }
825             }
826         }
827     }
828 }

```

```

825     }
826 }
827 }
828 return false;
829 }
830
831 bool adjust_dual_variables(int root) {
832     // delta1 : rematch
833     cost_t time1 = event1.time;
834
835     // delta2 : grow
836     cost_t time2 = Inf;
837     if (!heap2.empty())
838         time2 = heap2.min().time;
839
840     // delta3 : contract :  $O(m \log n)$  time / Edmonds search [ bottleneck (?) ]
841     cost_t time3 = Inf;
842     while (!heap3.empty()) {
843         EdgeEvent e = heap3.min();
844         int x = e.from, y = edges[e.to].to; // e.to is some edge id.
845         if (surface[x] != surface[y]) {
846             time3 = e.time;
847             break;
848         } else
849             heap3.pop();
850     }
851
852     // delta4 : expand
853     cost_t time4 = Inf;
854     if (!heap4.empty())
855         time4 = heap4.min().time;
856
857     // -- events --
858     cost_t time_next = min(min(time1, time2), min(time3, time4));
859     assert(time_current_ <= time_next && time_next < Inf);
860     time_current_ = time_next;
861
862     if (time_current_ == event1.time) {
863         int x = event1.id;
864         if (x != root)
865             rematch(x, 0);
866         return true;
867     }
868     while (!heap2.empty() && heap2.min().time == time_current_) {
869         int x = heap2.min().from, y = heap2.min().to;
870         if (grow(root, x, y))
871             return true; // `grow` function will call `heap2.erase(by)`.
872     }
873     while (!heap3.empty() && heap3.min().time == time_current_) {
874         int x = heap3.min().from, eid = heap3.min().to;
875         int y = edges[eid].to;
876         heap3.pop();
877         if (surface[x] == surface[y])
878             continue;
879         contract(x, y, eid);
880     }
881     while (!heap4.empty() && heap4.min().time == time_current_) {
882         int b = heap4.argmin();
883         heap4.pop();
884         expand(b);
885     }
886     return false;
887 }
888
889 private:

```

```

890 void initialize() {
891     que = Queue<int>(N);
892     mate.assign(S, 0);
893     link.assign(S, {0, 0});
894     label.assign(S, kFree);
895     base.resize(S);
896     for (int u = 1; u < S; ++u)
897         base[u] = u;
898     surface.resize(S);
899     for (int u = 1; u < S; ++u)
900         surface[u] = u;
901
902     potential.resize(S);
903     node.resize(S);
904     for (int b = 1; b < S; ++b)
905         node[b] = Node(b);
906
907     unused_bid.resize(B);
908     for (int i = 0; i < B; ++i)
909         unused_bid[i] = N + B - i;
910     unused_bid_idx_ = B;
911
912     // for  $O(nm \log n)$  implementation
913     reset_time();
914     time_created.resize(S);
915     slack.resize(S);
916     for (int i = 0; i < S; ++i)
917         slack[i] = Inf;
918     best_from.assign(S, 0);
919     heavy.assign(S, 0);
920     lazy.assign(S, 0);
921     group.resize(S);
922     for (int i = 0; i < S; ++i)
923         group[i] = i;
924 }
925
926 void set_potential() {
927     for (int u = 1; u <= N; ++u) {
928         cost_t max_c = 0;
929         for (int eid = ofs[u]; eid < ofs[u + 1]; ++eid) {
930             max_c = max(max_c, edges[eid].cost);
931         }
932         potential[u] = max_c >> 1;
933     }
934 }
935
936 void find_maximal_matching() {
937     // Find a maximal matching naively.
938     for (int u = 1; u <= N; ++u)
939         if (!mate[u]) {
940             for (int eid = ofs[u]; eid < ofs[u + 1]; ++eid) {
941                 auto& e = edges[eid];
942                 int v = e.to;
943                 if (mate[v] > 0 || reduced_cost(u, v, e) > 0)
944                     continue;
945                 mate[u] = v;
946                 mate[v] = u;
947                 break;
948             }
949         }
950 }
951
952 private:
953 int N, B, S; //  $N = |V|$ ,  $B = (|V| - 1) / 2$ ,  $S = N + B + 1$ 
954 vector<int> ofs;

```

```

955     vector<Edge> edges;
956
957     Queue<int> que;
958     vector<int> mate, surface, base;
959     vector<Link> link;
960     vector<Label> label;
961     vector<cost_t> potential;
962
963     vector<int> unused_bid;
964     int unused_bid_idx_;
965     vector<Node> node;
966
967     // for O(nm log n) implementation
968     vector<int> heavy, group;
969     vector<cost_t> time_created, lazy, slack;
970     vector<int> best_from;
971
972     cost_t time_current_;
973     Event event1;
974     ModifiableHeap<EdgeEvent> heap2;
975     ModifiableHeaps<EdgeEvent> heap2s;
976     FastHeap<EdgeEvent> heap3;
977     ModifiableHeap<cost_t> heap4;
978 };
979
980 using MWM = MaximumWeightedMatching<int>;
981 using Edge = MWM::InputEdge;
982
983 int main() {
984     ios::sync_with_stdio(false);
985     cin.tie(0);
986     cout.tie(0);
987     int N, M;
988     cin >> N >> M;
989     vector<Edge> edges(2 * M);
990     vector<int> ou(N + 2), ov(N + 2);
991
992     int u, v, c;
993     for (int i = 0; i < M; ++i) {
994         cin >> u >> v >> c;
995         edges[i] = {u, v, c};
996         ou[u + 1] += 1;
997         ov[v + 1] += 1;
998     }
999     for (int i = 1; i <= N + 1; ++i)
1000         ov[i] += ov[i - 1];
1001     for (int i = 0; i < M; ++i)
1002         edges[M + (ov[edges[i].to]++)] = edges[i];
1003     for (int i = 1; i <= N + 1; ++i)
1004         ou[i] += ou[i - 1];
1005     for (int i = 0; i < M; ++i)
1006         edges[ou[edges[i + M].from]++] = edges[i + M];
1007     edges.resize(M);
1008
1009     auto ans = MWM(N, edges).maximum_weighted_matching();
1010     cout << ans.first << endl;
1011     for (int i = 1; i <= N; ++i) {
1012         cout << ans.second[i] << (i == N ? '\n' : ' ');
1013     }
1014     return 0;
1015 }

```

## 3 math

### 3.1 2DGeometry

---

```

1 namespace Geometry {
2 // 定义以及防止精度出错
3 const double eps = 1e-8;
4 const double inf = 1e9;
5 const double pi = acos(-1.0);
6
7 inline int sgn(double x) {
8     if (fabs(x) < eps)
9         return 0;
10    if (x < 0)
11        return -1;
12    return 1;
13 }
14
15 // 单位换算
16 inline double degree2radian(const double& alpha) {
17     return alpha / 180 * pi;
18 }
19
20 inline double radian2degree(const double& alpha) {
21     return alpha / pi * 180;
22 }
23
24 // 点 (向量)
25 // 也是远点到该点的向量
26 struct point {
27     double x, y;
28     point(double _x = 0, double _y = 0)
29         : x(_x), y(_y) {}
30
31     point operator-(const point& b) const {
32         return point(x - b.x, y - b.y);
33     }
34
35     point operator+(const point& b) const {
36         return point(x + b.x, y + b.y);
37     }
38
39     bool operator<(const point& b) const {
40         return sgn(x - b.x) == 0 ? sgn(y - b.y) < 0 : sgn(x - b.x) < 0;
41     }
42
43     bool operator==(const point& b) const {
44         return sgn(x - b.x) == 0 && sgn(y - b.y) == 0;
45     }
46
47     point operator*(const double& b) {
48         return point(x * b, y * b);
49     }
50
51     point operator/(const double& b) {
52         return point(x / b, y / b);
53     }
54
55     // 绕原点逆时针旋转, 给出正弦和余弦值
56     // 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
57     void transxy(const double& sinb, const double& cosb) {
58         double tx = x, ty = y;
59         x = tx * cosb - ty * sinb;
60         y = tx * sinb + ty * cosb;
61     }

```

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

```

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

```



```

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

```

```

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

```

```
322 //      point s,v;
323 //      line(point a=point(),point b=point()) {
324 //          s=a;
325 //          v.x=b.x-a.x;
326 //          v.y=b.y-a.y;
327 //      }
328 // };
329
330 // 圆
331 struct circle {
332     point p;    // 圆心
333     double r;   // 半径
334
335     circle() {}
336
337     circle(point _p, double _r)
338         : p(_p), r(_r) {}
339     circle(double _x, double _y, double _r)
340         : p(point(_x, _y)), r(_r) {}
341
342     // 圆上三点确定圆
343     circle(point x1, point x2, point x3) {
344         double a = x2.x - x1.x;
345         double b = x2.y - x1.y;
346         double c = x3.x - x2.x;
347         double d = x3.y - x2.y;
348         double e = x2.x * x2.x + x2.y * x2.y - x1.x * x1.x - x1.y * x1.y;
349         double f = x3.x * x3.x + x3.y * x3.y - x2.x * x2.x - x2.y * x2.y;
350
351         p = point((f * b - e * d) / (c * b - a * d) / 2, (a * f - e * c) / (a * d - b * c) / 2);
352         r = distance(p, x1);
353     }
354
355     double area() {
356         return pi * r * r;
357     }
358
359     double perimeter() {
360         return 2 * pi * r;
361     }
362
363     // 点和圆的关系
364     // 0 圆外
365     // 1 圆上
366     // 2 圆内
367     int relationToPoint(point a) {
368         double d2 = distance2(p, a);
369         if (sgn(d2 - r * r) < 0)
370             return 2;
371         else if (sgn(d2 - r * r) == 0)
372             return 1;
373         return 0;
374     }
375
376     // 圆和直线的关系
377     // 0 圆外
378     // 1 圆上
379     // 2 圆内
380     int relationToLine(line l) {
381         double d = l.disPointToLine(p);
382         if (sgn(d - r) < 0)
383             return 2;
384         else if (sgn(d - r) == 0)
385             return 1;
386         return 0;
```

```
387 }
388
389 // 圆和线段的关系
390 // 0 圆外
391 // 1 圆上
392 // 2 圆内
393 int relationToSeg(line l) {
394     double d = l.disPointToSeg(p);
395     if (sgn(d - r) < 0)
396         return 2;
397     else if (sgn(d - r) == 0)
398         return 1;
399     return 0;
400 }
401
402 // 圆和圆的关系
403 // 5 相离
404 // 4 外切
405 // 3 相交
406 // 2 内切
407 // 1 内含
408 int relationToCircle(circle c) {
409     double d = distance(p, c.p);
410     if (sgn(d - r - c.r) > 0)
411         return 5;
412     if (sgn(d - r - c.r) == 0)
413         return 4;
414     double l = fabs(r - c.r);
415     if (sgn(d - r - c.r) < 0 && sgn(d - l) > 0)
416         return 3;
417     if (sgn(d - l) == 0)
418         return 2;
419     if (sgn(d - l) < 0)
420         return 1;
421     return -1;
422 }
423 };
424
425 // 多边形
426 struct polygon {
427     int n; // 顶点个数
428     vector<point> p; // 顶点
429     vector<line> l; // 边
430
431     polygon()
432         : n(0) {}
433     polygon(int _n)
434         : n(_n), p(n) {}
435
436     point& operator[](int idx) { return p[idx]; }
437
438     void resize(int _n) {
439         n = _n;
440         p.resize(n);
441     }
442
443     // 多边形周长
444     double perimeter() {
445         double sum = 0;
446         for (int i = 0; i < n; i++)
447             sum += (p[(i + 1) % n] - p[i]).length();
448         return sum;
449     }
450
451     // 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 getConvex() {
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)

```

```

647 // 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  $O(n)$ 
648 circle getMinCircle() {
649     // 随机打乱顺序
650     srand(time(0));
651     for (int i = n - 1; i >= 1; --i)
652         swap(p[i], p[rand() % i]);
653
654     circle c(p[0], 0);
655     for (int i = 0; i < n; ++i) {
656         if (c.relationToPoint(p[i]) == 2)
657             continue;
658         c.p = (p[0] + p[i]) / 2;
659         c.r = distance(p[0], p[i]) / 2;
660
661         for (int j = 1; j < i; ++j) {
662             if (c.relationToPoint(p[j]) == 2)
663                 continue;
664             c.p = (p[i] + p[j]) / 2;
665             c.r = distance(p[i], p[j]) / 2;
666
667             for (int k = 1; k < j; ++k) {
668                 if (c.relationToPoint(p[k]) == 2)
669                     continue;
670                 c = circle(p[i], p[j], p[k]);
671             }
672         }
673     }
674     return c;
675 }
676
677 // 点与多边形的位置关系
678 // 0 外部
679 // 1 内部
680 // 2 边上
681 // 3 点上
682 int relationToPoint(point a) {
683     for (int i = 0; i < n; ++i)
684         if (p[i] == a)
685             return 3;
686
687     getline();
688     for (int i = 0; i < n; ++i)
689         if (l[i].relationToPoint(a) == 3)
690             return 2;
691
692     int cnt = 0;
693     for (int i = 0, j; i < n; ++i) {
694         j = (i + 1) % n;
695         int k = sgn(det(p[j], a, p[i]));
696         int u = sgn(p[i].y - a.y);
697         int v = sgn(p[j].y - a.y);
698         if (k > 0 && u < 0 && v >= 0)
699             ++cnt;
700         if (k < 0 && v < 0 && u >= 0)
701             --cnt;
702     }
703     return cnt != 0;
704 }
705
706 void DEBUG() {
707     cout << n << endl;
708     for (int i = 0; i < n; ++i) {
709         cout << p[i].x << " " << p[i].y << endl;
710     }
711 }

```



```

712 };
713
714 // 半平面 ( $ax + by + c \geq 0$ ), 其实也就是直线
715 // 对于直线 ( $s, e$ ),  $h.s$  为起点,  $h.e$  为方向向量 ( $e - s$ )
716 struct halfplane {
717     point s, v;
718     double k;
719     halfplane() {}
720     halfplane(point _s, point _v)
721         : s(_s), v(_v) {
722         k = v.alpha();
723     }
724     bool operator<(const halfplane& h) const {
725         return k < h.k;
726     }
727 };
728
729 // 点和半平面的位置关系
730 // 0 不在右侧
731 // 1 在右侧
732 int relationPointToHalfplane(point p, halfplane h) {
733     return sgn(det(h.v, p - h.s)) < 0;
734 }
735
736 // 半平面交点
737 point HalfplaneCrossHalfplane(halfplane h1, halfplane h2) {
738     double a = det(h2.v, h1.s - h2.s) / det(h1.v, h2.v);
739     return h1.s + h1.v * a;
740 }
741
742 // 从点集构造出半平面集
743 // 多边形的半平面集即为多边形边集
744 void getHalfPlanes(polygon& p, vector<halfplane>& h) {
745     if (p.direction() != 1)
746         reverse(p.p.begin(), p.p.end());
747     int n = p.n;
748     for (int i = 0, j; i < n; ++i) {
749         j = (i + 1) % n;
750         h.push_back(halfplane(p[i], p[j] - p[i]));
751     }
752 }
753
754 // 有时候题目给的不一定是闭合图形, 需要自行添加边界
755 // ( $x_1, y_1$ ) 为矩形边界左下角, ( $x_2, y_2$ ) 为矩形边界右上角
756 // Usage: addBorderHalfPlanes(0, 0, 1e4, 1e4, h);
757 // POJ2451
758 void addBorderHalfPlanes(double x1, double y1, double x2, double y2, vector<halfplane>& h) {
759     polygon p(4);
760     p[0] = point(x1, y1);
761     p[1] = point(x2, y1);
762     p[2] = point(x2, y2);
763     p[3] = point(x1, y2);
764     getHalfPlanes(p, h);
765 }
766
767 // 半平面交
768 // 排序随机增量法 (SI) 求解半平面交, 复杂度为  $O(n \log n)$ 
769 // 瓶颈为排序算法, 用基数排序则为  $O(n)$ 
770 // 最终的结果为一个凸包, 若少于 3 个点则说明无解
771
772 // 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)
773 // 多边形的半平面交即为多边形的核 (P4196)
774
775 bool getHalfPlaneIntersection(vector<halfplane>& h, polygon& hpi) {
776     int n = int(h.size()), l, r;

```

```

777 sort(h.begin(), h.end());
778
779 vector<point> p(n);
780 vector<halfplane> q(n);
781
782 l = r = 0;
783 q[l] = h[0];
784 for (int i = 1; i < n; ++i) {
785     while (l < r && relationPointToHalfplane(p[r - 1], h[i]))
786         --r;
787     while (l < r && relationPointToHalfplane(p[l], h[i]))
788         ++l;
789     q[++r] = h[i];
790     if (l < r && sgn(det(q[r].v, q[r - 1].v)) == 0) {
791         --r;
792         if (!relationPointToHalfplane(h[i].s, q[r]))
793             q[r] = h[i];
794     }
795     if (l < r)
796         p[r - 1] = HalfplaneCrossHalfplane(q[r - 1], q[r]);
797 }
798 while (l < r && relationPointToHalfplane(p[r - 1], q[l]))
799     --r;
800 if (r - l + 1 <= 2)
801     return false; // 交不存在
802 p[r] = HalfplaneCrossHalfplane(q[l], q[r]);
803
804 hpi.resize(r - l + 1);
805 for (int i = l, j = 0; i <= r; ++i)
806     hpi[j++] = p[i];
807
808 return true;
809 }
810
811 // 多边形内部半径最大的圆半径 (POJ3525)
812 // 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
813 double getMaxInsideCircleRadius(polygon& p) {
814     if (p.direction() != 1)
815         reverse(p.p.begin(), p.p.end());
816     int n = p.n;
817
818     // 方向向量, 垂直单位向量
819     vector<point> d(n), v(n);
820     for (int i = 0; i < n; ++i) {
821         d[i] = p[(i + 1) % n] - p[i];
822         v[i] = d[i].trans90().unit();
823     }
824
825     double l = 0, r = 1e4, m;
826     while (r - l >= eps) {
827         m = (l + r) / 2;
828
829         vector<halfplane> h(n);
830         polygon hpi;
831         for (int i = 0; i < n; ++i)
832             h[i] = halfplane(p[i] + v[i] * m, d[i]);
833         bool can = getHalfPlaneIntersection(h, hpi);
834
835         if (can)
836             l = m;
837         else
838             r = m;
839     }
840     return l;
841 }

```

```

842 } // namespace Geometry
843 using namespace Geometry;

```

---

## 3.2 3DGeometry

---

```

1 namespace Geometry3 {
2     const double eps = 1e-8;
3
4     int sgn(double x) {
5         if (fabs(x) < eps)
6             return 0;
7         if (x < 0)
8             return -1;
9         return 1;
10    }
11
12    struct point3 {
13        double x, y, z;
14        point3(double _x = 0, double _y = 0, double _z = 0)
15            : x(_x), y(_y), z(_z) {}
16
17        bool operator==(const point3& p) const {
18            return sgn(x - p.x) == 0 && sgn(y - p.y) == 0 && sgn(z - p.z) == 0;
19        }
20
21        bool operator<(const point3& p) const {
22            if (sgn(x - p.x) != 0)
23                return sgn(x - p.x) < 0;
24            if (sgn(y - p.y) != 0)
25                return sgn(y - p.y) < 0;
26            return sgn(z - p.z) < 0;
27        }
28
29        point3 operator-(const point3& p) const {
30            return point3(x - p.x, y - p.y, z - p.z);
31        }
32
33        point3 operator+(const point3& p) const {
34            return point3(x + p.x, y + p.y, z + p.z);
35        }
36
37        point3 operator*(const double& a) const {
38            return point3(x * a, y * a, z * a);
39        }
40
41        point3 operator/(const double& a) const {
42            return point3(x / a, y / a, z / a);
43        }
44
45        double operator*(const point3& p) const {
46            return x * p.x + y * p.y + z * p.z;
47        }
48
49        point3 operator^(const point3& p) const {
50            return point3(y * p.z - z * p.y, z * p.x - x * p.z, x * p.y - y * p.x);
51        }
52
53        double length() {
54            return sqrt(x * x + y * y + z * z);
55        }
56
57        double length2() {
58            return x * x + y * y + z * z;
59        }

```

```
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())
```

```

125         : s(_s), e(_e) {}
126
127     bool operator==(const line3& l) const {
128         return (s == l.s) && (e == l.e);
129     }
130
131     // 点到直线的距离
132     double disPointToLine(point3 p) {
133         return det(s, e, p).length() / distance(s, e);
134     }
135
136     // 点到线段的距离
137     double disPointToSeg(point3 p) {
138         if (sgn(dot(s, p, e)) < 0 || sgn(dot(e, p, s)) < 0)
139             return min(distance(s, p), distance(e, p));
140         return disPointToLine(p);
141     }
142
143     // 点在直线上的投影
144     point3 projectionPointOnLine(point3 p) {
145         return s + ((e - s) * dot(s, e, p)) / (e - s).length2();
146     }
147
148     // 绕 p 旋转 alpha 度
149     point3 rotate(point3 p, double alpha) {
150         if (sgn(det(p, s, e).length()) == 0)
151             return p;
152         point3 p1 = det(s, e, p);
153         point3 p2 = det(e - s, p1);
154         double len = det(p, s, e).length() / distance(s, e);
155         p1 = p1.trunc(len);
156         p2 = p2.trunc(len);
157         point3 p3 = p + p2;
158         point3 p4 = p3 + p1;
159         return p3 + ((p - p3) * cos(alpha) + (p4 - p3) * sin(alpha));
160     }
161
162     // 点在线段上
163     bool isPointOnSeg(point3 p) {
164         return sgn(det(p, s, e).length()) == 0 && sgn(dot(p, s, e)) == 0;
165     }
166 };
167
168 struct plane {
169     point3 a, b, c; // 3 点确定平面
170     point3 o;       // 平面的法向量
171
172     point3 pvec() {
173         return det(a, b, c);
174     }
175
176     plane(point3 _a, point3 _b, point3 _c)
177         : a(_a), b(_b), c(_c) {}
178
179     plane(point3 _a, point3 _o)
180         : a(_a), o(_o) {}
181
182     // ax + by + cz + d = 0;
183     plane(double _a, double _b, double _c, double _d) {
184         o = point3(_a, _b, _c);
185         if (sgn(_a) != 0)
186             a = point3((-_d - _c - _b) / _a, 1, 1);
187         else if (sgn(_b) != 0)
188             a = point3(1, (-_d - _c - _a) / _b, 1);
189         else if (sgn(_c) != 0)

```

```

190     a = point3(1, 1, (-_d - _b - _a) / _c);
191 }
192
193 // 点在平面上
194 bool isPointOnPlane(point3 p) {
195     return sgn((p - a) * o) == 0;
196 }
197
198 // 两平面夹角
199 double angle(plane f) {
200     return acos(o * f.o) / (o.length() * f.o.length());
201 }
202
203 // 平面和直线是否相交
204 int PlaneCrossLine(line3 l, point3& p) {
205     double x = o * (l.e - a);
206     double y = o * (l.s - a);
207     double d = x - y;
208     if (sgn(d) == 0)
209         return 0;
210     p = ((l.s * x) - (l.e * y)) / d;
211     return 1;
212 }
213
214 // 点到平面的最近点
215 point3 PointToPlane(point3 p) {
216     line3 l = line3(p, p + o);
217     PlaneCrossLine(l, p);
218     return p;
219 }
220
221 // 平面和平面是否相交
222 int PlaneCrossPlane(plane f, line3& l) {
223     point3 o1 = o ^ f.o;
224     point3 o2 = o ^ o1;
225
226     double d = fabs(f.o * o2);
227     if (sgn(d) == 0)
228         return 0;
229     point3 p = a + (o2 * (f.o * (f.a - a)) / d);
230     l = line3(p, p + o1);
231     return 1;
232 }
233 };
234
235 struct polygon3 {
236     struct face {
237         int a, b, c;
238         bool ok;
239     };
240
241     int n;
242     vector<point3> P;
243
244     int num;
245     vector<face> F;
246     vector<vector<int>> > G;
247
248     polygon3()
249         : n(0) {}
250     polygon3(int _n)
251         : n(_n), P(n), F(8 * n), G(n, vector<int>(n)) {}
252
253     double cmp(point3 p, face f) {
254         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);

```

```

28 }
29 int get_pos(unsigned pos) const {
30     return pos >= v.size() ? 0 : v[pos];
31 }
32 BigInt& setsign(int newsign, int rev) {
33     for (int i = (int)v.size() - 1; i > 0 && v[i] == 0; i--)
34         v.erase(v.begin() + i);
35     sign = (v.size() == 0 || (v.size() == 1 && v[0] == 0))
36           ? 1
37           : (rev ? newsign * sign : newsign);
38     return *this;
39 }
40 std::string to_str() const {
41     BigInt b = *this;
42     std::string s;
43     for (int i = (b.zip(1), 0); i < (int)b.v.size(); ++i)
44         s += char(*(b.v.rbegin() + i) + '0');
45     return (sign < 0 ? "-" : "") + (s.empty() ? std::string("0") : s);
46 }
47 bool absless(const BigInt& b) const {
48     if (v.size() != b.v.size())
49         return v.size() < b.v.size();
50     for (int i = (int)v.size() - 1; i >= 0; i--)
51         if (v[i] != b.v[i])
52             return v[i] < b.v[i];
53     return false;
54 }
55 BigInt operator-() const {
56     BigInt c = *this;
57     c.sign = (v.size() > 1 || v[0]) ? -c.sign : 1;
58     return c;
59 }
60 BigInt& operator=(const std::string& s) {
61     if (s[0] == '-')
62         *this = s.substr(1);
63     else {
64         for (int i = (v.clear(), 0); i < (int)s.size(); ++i)
65             v.push_back(*(s.rbegin() + i) - '0');
66         zip(0);
67     }
68     return setsign(s[0] == '-' ? -1 : 1, sign = 1);
69 }
70 bool operator<(const BigInt& b) const {
71     return sign != b.sign ? sign < b.sign : (sign == 1 ? absless(b) : !absless(b));
72 }
73 bool operator==(const BigInt& b) const {
74     return v == b.v && sign == b.sign;
75 }
76 BigInt& operator+=(const BigInt& b) {
77     if (sign != b.sign)
78         return *this = (*this) - b;
79     v.resize(std::max(v.size(), b.v.size()) + 1);
80     for (int i = 0, carry = 0; i < (int)b.v.size() || carry; i++) {
81         carry += v[i] + b.get_pos(i);
82         v[i] = carry % 10000, carry /= 10000;
83     }
84     return setsign(sign, 0);
85 }
86 BigInt operator+(const BigInt& b) const {
87     BigInt c = *this;
88     return c += b;
89 }
90 void add_mul(const BigInt& b, int mul) {
91     v.resize(std::max(v.size(), b.v.size()) + 2);
92     for (int i = 0, carry = 0; i < (int)b.v.size() || carry; i++) {

```

```

93     carry += v[i] + b.get_pos(i) * mul;
94     v[i] = carry % 10000, carry /= 10000;
95 }
96 }
97 BigInt operator-(const BigInt& b) const {
98     if (sign != b.sign)
99         return (*this) + -b;
100    if (absless(b))
101        return -(b - *this);
102    BigInt c;
103    for (int i = 0, borrow = 0; i < (int)v.size(); i++) {
104        borrow += v[i] - b.get_pos(i);
105        c.v.push_back(borrow);
106        c.v.back() -= 10000 * (borrow >= 31);
107    }
108    return c.setsign(sign, 0);
109 }
110 BigInt operator*(const BigInt& b) const {
111     if (b < *this)
112         return b * *this;
113     BigInt c, d = b;
114     for (int i = 0; i < (int)v.size(); i++, d.v.insert(d.v.begin(), 0))
115         c.add_mul(d, v[i]);
116     return c.setsign(sign * b.sign, 0);
117 }
118 BigInt operator/(const BigInt& b) const {
119     BigInt c, d;
120     d.v.resize(v.size());
121     double db = 1.0 / (b.v.back() + (b.get_pos((unsigned)b.v.size() - 2) / 1e4) +
122                      (b.get_pos((unsigned)b.v.size() - 3) + 1) / 1e8);
123     for (int i = (int)v.size() - 1; i >= 0; i--) {
124         c.v.insert(c.v.begin(), v[i]);
125         int m = (int)((c.get_pos((int)b.v.size()) * 10000 +
126                      c.get_pos((int)b.v.size() - 1)) *
127                  db);
128         c = c - b * m, d.v[i] += m;
129         while (!(c < b))
130             c = c - b, d.v[i] += 1;
131     }
132     return d.setsign(sign * b.sign, 0);
133 }
134 BigInt operator%(const BigInt& b) const {
135     return *this - *this / b * b;
136 }
137 bool operator>(const BigInt& b) const {
138     return b < *this;
139 }
140 bool operator<=(const BigInt& b) const {
141     return !(b < *this);
142 }
143 bool operator>=(const BigInt& b) const {
144     return !(*this < b);
145 }
146 bool operator!=(const BigInt& b) const {
147     return !(*this == b);
148 }
149 };

```

### 3.4 BSGS

```

1 namespace Backlight {
2
3 namespace BSGS {
4 typedef long long ll;

```

```
5
6 ll exgcd(ll a, ll b, ll& x, ll& y) {
7     if (b == 0) {
8         x = 1;
9         y = 0;
10        return a;
11    }
12    ll d = exgcd(b, a % b, x, y);
13    ll z = x;
14    x = y;
15    y = z - y * (a / b);
16    return d;
17 }
18
19 ll qpow(ll a, ll n, ll p) {
20     ll ans = 1;
21     for (; n; n >>= 1) {
22         if (n & 1)
23             ans = ans * a % p;
24         a = a * a % p;
25     }
26     return ans;
27 }
28
29 // solve  $a^x = b \pmod p$ ,  $p$  is a prime must hold
30 ll BSGS(ll a, ll b, ll p) {
31     unordered_map<ll, int> mp;
32     if (__gcd(a, p) != 1)
33         return -1;
34     if (b % p == 1)
35         return 0;
36     a %= p;
37     b %= p;
38     ll k = sqrt(p), t = qpow(a, k, p), s = b;
39     for (int i = 0; i <= k; i++, s = s * a % p)
40         mp[s] = i;
41     s = 1;
42     for (int i = 0; i <= k; i++, s = s * t % p) {
43         int ans = mp.count(s) ? mp[s] : -1;
44         if (ans != -1 && i * k - ans >= 0)
45             return i * k - ans;
46     }
47     return -1;
48 }
49
50 // solve  $a^x = b \pmod p$ ,  $p$  don't need to be a prime
51 ll EXBSGS(ll a, ll b, ll p) {
52     ll k = 0, d, c = 1, x, y;
53     a %= p;
54     b %= p;
55     if (a == b)
56         return 1;
57     if (b == 1)
58         return 0;
59     while ((d = __gcd(a, p)) != 1) {
60         if (b % d)
61             return -1;
62         k++;
63         b /= d;
64         p /= d;
65         c = c * (a / d) % p;
66         if (c == b)
67             return k;
68     }
69     if (p == 1)
```

```

70     return k;
71     exgcd(c, p, x, y);
72     b = (b * x % p + p) % p;
73     a %= p;
74     ll ans = BSGS(a, b, p);
75     return ans == -1 ? ans : ans + k;
76 }
77 } // namespace BSGS
78
79 } // namespace Backlight

```

---

### 3.5 Cipolla

---

```

1 namespace Backlight {
2
3 namespace Cipolla {
4 mt19937 rnd(chrono::steady_clock::now().time_since_epoch().count());
5 ll W, P;
6 struct complex {
7     ll r, i;
8     complex(ll _r, ll _i)
9         : r(_r), i(_i) {}
10     inline complex operator*(const complex& c) const { return complex((r * c.r % P + i * c.i % P * W) % P, (r * c.i % P -
11 );};
12
13 inline complex pow(complex a, int b) {
14     complex res(1, 0);
15     while (b) {
16         if (b & 1)
17             res = res * a;
18         a = a * a;
19         b >>= 1;
20     }
21     return res;
22 }
23
24 inline ll pow(ll a, ll b, ll p) {
25     ll res = 1;
26     while (b) {
27         if (b & 1)
28             res = res * a % p;
29         a = a * a % p;
30         b >>= 1;
31     }
32     return res;
33 }
34
35 // solve x for x^2 = a (mod p)
36 ll solve(ll a, ll p) {
37     P = p;
38     a %= p;
39     if (a == 0)
40         return 0;
41
42     ll t = pow(a, (p - 1) / 2, p);
43     if (t != 1)
44         return -1;
45     while (true) {
46         t = rnd() % p;
47         ll c = (t * t % p + p - a) % p;
48         if (pow(c, (p - 1) / 2, p) == p - 1)
49             break;
50     }
51

```

---

```

52 W = (t * t % p + p - a) % p;
53 ll x = pow(complex(t, 1), (p + 1) / 2).r;
54 return x;
55 }
56
57 } // namespace Cipolla
58
59 } // namespace Backlight

```

---

### 3.6 Combination

---

```

1 struct Combination {
2     int N;
3     vector<Mint> f, g;
4
5     Combination()
6         : N(0) {}
7     Combination(int _n)
8         : N(_n), f(N + 1), g(N + 1) {
9         f[0] = 1;
10        for (int i = 1; i <= N; ++i)
11            f[i] = f[i - 1] * i;
12        g[N] = f[N].inv();
13        for (int i = N - 1; i >= 0; --i)
14            g[i] = g[i + 1] * (i + 1);
15    }
16
17    Mint get(int n, int m) {
18        if (n < 0 || m < 0 || n < m)
19            return 0;
20        return f[n] * g[m] * g[n - m];
21    }
22 } C(N);

```

---

### 3.7 CRT

---

```

1 namespace Backlight {
2
3     // get x, y for ax + by = GCD(a, b)
4     ll exgcd(ll a, ll b, ll& x, ll& y) {
5         if (b == 0) {
6             x = 1;
7             y = 0;
8             return a;
9         }
10        ll d = exgcd(b, a % b, x, y);
11        ll z = x;
12        x = y;
13        y = z - y * (a / b);
14        return d;
15    }
16
17    // CRT: solve x = a_i (mod m_i) for i in [0, n)
18
19    // GCD(m_i, m_j) = 1 hold
20    ll CRT(vector<ll>& a, vector<ll>& m) {
21        assert(a.size() == m.size());
22        assert(a.size() > 0);
23        int n = a.size();
24        ll M = 1, res = 0;
25        for (int i = 0; i < n; ++i)
26            M *= m[i];

```

```

27 ll _M, x, y;
28 for (int i = 0; i < n; ++i) {
29     _M = M / m[i];
30     exgcd(_M, m[i], x, y);
31     res = (res + a[i] * _M % M * x % M) % M;
32 }
33 if (res < 0)
34     res += M;
35 return res;
36 }
37
38 ll mul(ll a, ll b, ll mod) {
39     ll res = 0;
40     while (b) {
41         if (b & 1)
42             res = (res + a) % mod;
43         b >>= 1;
44         a = (a + a) % mod;
45     }
46     return res;
47 }
48
49 // GCD(m_i, m_j) = 1 not hold
50 ll EXCRT(vector<ll>& a, vector<ll>& m) {
51     assert(a.size() == m.size());
52     assert(a.size() > 0);
53     int n = a.size();
54     ll res = a[0], M = m[0], B, g, x, y;
55     for (int i = 1; i < n; ++i) {
56         B = ((a[i] - res) % m[i] + m[i]) % m[i];
57         g = exgcd(M, m[i], x, y);
58         x = mul(x, B / g, m[i]);
59         res += M * x;
60         M *= m[i] / g;
61         res = (res + M) % M;
62     }
63     return res;
64 }
65
66 } // namespace Backlight

```

### 3.8 du

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 using ll = int64_t;
5
6 const int LIM = 1e7;
7
8 int pcnt, prime[LIM], mu[LIM];
9 bool vis[LIM];
10 void seive(int n) {
11     pcnt = 0;
12     mu[0] = 0;
13     mu[1] = 1;
14     for (int i = 2; i <= n; ++i) {
15         if (!vis[i]) {
16             prime[++pcnt] = i;
17             mu[i] = -1;
18         }
19         for (int j = 1; j <= pcnt; ++j) {
20             ll nxt = 1ll * i * prime[j];
21             if (nxt > n)

```

```

22         break;
23         vis[nxt] = true;
24         if (i % prime[j] == 0) {
25             mu[nxt] = 0;
26             break;
27         }
28         mu[nxt] = -mu[i];
29     }
30 }
31 for (int i = 1; i <= n; ++i)
32     mu[i] += mu[i - 1];
33 }
34
35 map<ll, ll> mp;
36
37 //  $S(n) = 1 - \sum_{i=2}^n S(\lfloor \frac{n}{i} \rfloor)$ 
38 // Time Complexity:  $O(n^{\frac{2}{3}})$ 
39 ll S_mu(ll n) {
40     if (n < LIM)
41         return mu[n];
42     if (mp.count(n))
43         return mp[n];
44
45     ll ret = 0;
46     for (ll i = 2, j; i <= n; i = j + 1) {
47         j = n / (n / i);
48         ret += (j - i + 1) * S_mu(n / i);
49     }
50     ret = 1 - ret;
51
52     mp[n] = ret;
53     return ret;
54 }
55
56 //  $S(n) = \frac{(n+1)n}{2} - \sum_{i=2}^n S(\lfloor \frac{n}{i} \rfloor)$ 
57 //  $S(n) = \sum_{d=1}^n \mu(d) \lfloor \frac{n}{d} \rfloor \lfloor \frac{n}{d} \rfloor$ 
58 ll S_phi(ll n) {
59     ll ret = 0;
60     for (ll i = 1, j; i <= n; i = j + 1) {
61         j = n / (n / i);
62         ret += (S_mu(j) - S_mu(i - 1)) * (n / i) * (n / i);
63     }
64     ret = (ret - 1) / 2 + 1;
65     return ret;
66 }
67
68 void solve(int Case) {
69     ll n;
70     scanf("%lld", &n);
71     printf("%lld %lld\n", S_phi(n), S_mu(n));
72 }
73
74 int main() {
75     seive(LIM - 1);
76     int T = 1;
77     scanf("%d", &T);
78     for (int _ = 1; _ <= T; _++)
79         solve(_);
80     return 0;
81 }

```



### 3.9 EulerSeive

---

```
1 namespace Backlight {
2
3 vector<int> euler_seive(int n) {
4     vector<int> primes;
5     vector<bool> is(n + 1, 1);
6
7     for (int i = 2; i <= n; ++i) {
8         if (is[i])
9             primes.push_back(i);
10        for (int j = 0; j < (int)primes.size(); ++j) {
11            ll nxt = 1ll * primes[j] * i;
12            if (nxt > n)
13                break;
14            is[nxt] = false;
15            if (i % primes[j] == 0)
16                break;
17        }
18    }
19    return primes;
20 }
21
22 } // namespace Backlight
```

---

### 3.10 eval

---

```
1 int pri(char c) {
2     if (c == '^')
3         return 3;
4     if (c == '*' || c == '/')
5         return 2;
6     if (c == '+' || c == '-')
7         return 1;
8     return 0;
9 }
10
11 void in2post(char* s, char* t) {
12     int n = strlen(s), j = 0;
13     stack<char> ops;
14     for (int i = 0; i < n; ++i) {
15         t[j] = 0;
16         if (islower(s[i])) {
17             while (i < n && isdigit(s[i])) {
18                 t[j++] = s[i++];
19             }
20             t[j++] = ' ';
21             --i;
22         } else if (s[i] == '(') {
23             ops.push('(');
24         } else if (s[i] == ')') {
25             char op = 0;
26             while (!ops.empty()) {
27                 op = ops.top();
28                 ops.pop();
29                 if (op == '(')
30                     break;
31                 t[j++] = op;
32                 t[j++] = ' ';
33             }
34             assert(op == '(');
35         } else {
36             while (!ops.empty() && pri(s[i]) <= pri(ops.top())) {
```

```

37         t[j++] = ops.top();
38         t[j++] = ' ';
39         ops.pop();
40     }
41     ops.push(s[i]);
42 }
43 }
44 while (!ops.empty()) {
45     assert(ops.top() != '(');
46     t[j++] = ops.top();
47     t[j++] = ' ';
48     ops.pop();
49 }
50 t[j] = 0;
51 }
52
53 int eval(char* s) {
54     int n = strlen(s);
55     stack<int> nums;
56     for (int i = 0; i < n; ++i) {
57         if (isdigit(s[i])) {
58             int num = 0;
59             while (i < n && isdigit(s[i])) {
60                 num = num * 10 + s[i++] - '0';
61             }
62             nums.push(num);
63             --i;
64             continue;
65         }
66
67         if (s[i] == ' ')
68             continue;
69
70         assert(nums.size() >= 2);
71         int num2 = nums.top();
72         nums.pop();
73         int num1 = nums.top();
74         nums.pop();
75         switch (s[i]) {
76             case '+':
77                 nums.push(num1 + num2);
78                 break;
79             case '-':
80                 nums.push(num1 - num2);
81                 break;
82             case '*':
83                 nums.push(num1 * num2);
84                 break;
85             case '/':
86                 nums.push(num1 / num2);
87                 break;
88             default:
89                 assert(false);
90                 break;
91         }
92     }
93     assert(nums.size() == 1);
94     return nums.top();
95 }

```

### 3.11 EXGCD

```

1 namespace Backlight {
2

```

```

3 // get x_0, y_0 for ax + by = GCD(a, b)
4 // x = x_0 + bt
5 // y = y_0 - at
6 // for all interger t
7 #define EXGCD
8 ll exgcd(ll a, ll b, ll& x, ll& y) {
9     if (b == 0) {
10         x = 1;
11         y = 0;
12         return a;
13     }
14     ll d = exgcd(b, a % b, x, y);
15     ll z = x;
16     x = y;
17     y = z - y * (a / b);
18     return d;
19 }
20
21 } // namespace Backlight

```

### 3.12 FFT

```

1 namespace FFT {
2     const long double PI = acos(-1.0);
3     using LL = int64_t;
4     struct Complex {
5         long double r, i;
6         Complex()
7             : r(0), i(0) {}
8         Complex(long double _r, long double _i)
9             : r(_r), i(_i) {}
10        Complex conj() { return Complex(r, -i); }
11        inline Complex operator-(const Complex& c) const { return Complex(r - c.r, i - c.i); }
12        inline Complex operator+(const Complex& c) const { return Complex(r + c.r, i + c.i); }
13        inline Complex operator*(const Complex& c) const { return Complex(r * c.r - i * c.i, r * c.i + i * c.r); }
14    };
15    ostream& operator<<(ostream& os, Complex& c) {
16        return os << "(" << c.r << ", " << c.i << ")";
17    }
18
19    int N;
20    vector<int> r;
21    void init(int n) {
22        N = 1;
23        while (N <= n)
24            N <<= 1;
25        r.resize(N);
26        for (int i = 1; i < N; ++i)
27            r[i] = (r[i >> 1] >> 1) + ((i & 1) ? (N >> 1) : 0);
28    }
29
30    void FFT(vector<Complex>& a, int op) {
31        for (int i = 1; i < N; ++i)
32            if (i < r[i])
33                swap(a[i], a[r[i]]);
34        for (int i = 2; i <= N; i <<= 1) {
35            int l = i >> 1;
36            Complex w, x, wk(cos(PI / l), op * sin(PI / l));
37            for (int j = 0; j < N; j += i) {
38                w = Complex(1, 0);
39                for (int k = j; k < j + l; ++k) {
40                    x = a[k + 1] * w;
41                    a[k + 1] = a[k] - x;
42                    a[k] = a[k] + x;

```

```

43     w = w * wk;
44 }
45 }
46 }
47 if (op == -1)
48     for (int i = 0; i < N; i++)
49         a[i].r /= N, a[i].i /= N;
50 }
51
52 inline void FFT(vector<Complex>& a) {
53     FFT(a, 1);
54 }
55 inline void IFT(vector<Complex>& a) {
56     FFT(a, -1);
57 }
58
59 vector<int> convolution(const vector<int>& f, const vector<int>& g) {
60     int n = f.size(), m = g.size(), k = n + m - 1;
61     init(k);
62     vector<Complex> a(N), b(N);
63     for (int i = 0; i < n; ++i)
64         a[i] = Complex(f[i], 0);
65     for (int i = 0; i < m; ++i)
66         b[i] = Complex(g[i], 0);
67
68     FFT(a);
69     FFT(b);
70     for (int i = 0; i < N; ++i)
71         a[i] = a[i] * b[i];
72     IFT(a);
73
74     vector<int> h(k);
75     for (int i = 0; i < k; ++i)
76         h[i] = int(a[i].r + 0.5);
77     return h;
78 }
79
80 // 任意模数 FFT
81 vector<int> convolutionM(const vector<int>& f, const vector<int>& g, int p) {
82     int n = f.size(), m = g.size(), k = n + m - 1;
83     init(k);
84     vector<Complex> a(N), b(N), c(N), d(N);
85     for (int i = 0; i < n; ++i)
86         a[i] = Complex(f[i] >> 15, f[i] & 32767);
87     for (int i = 0; i < m; ++i)
88         c[i] = Complex(g[i] >> 15, g[i] & 32767);
89     FFT(a);
90     FFT(c);
91     for (int i = 1; i < N; ++i)
92         b[i] = a[N - i].conj();
93     for (int i = 1; i < N; ++i)
94         d[i] = c[N - i].conj();
95     b[0] = a[0].conj();
96     d[0] = c[0].conj();
97     for (int i = 0; i < N; ++i) {
98         Complex aa, bb, cc, dd;
99         aa = (a[i] + b[i]) * Complex(0.5, 0);
100        bb = (a[i] - b[i]) * Complex(0, -0.5);
101        cc = (c[i] + d[i]) * Complex(0.5, 0);
102        dd = (c[i] - d[i]) * Complex(0, -0.5);
103        a[i] = aa * cc + Complex(0, 1) * (aa * dd + bb * cc);
104        b[i] = bb * dd;
105    }
106    IFT(a);
107    IFT(b);

```

```

108 vector<int> h(k);
109 for (int i = 0; i < k; ++i) {
110     int aa, bb, cc;
111     aa = LL(a[i].r + 0.5) % p;
112     bb = LL(a[i].i + 0.5) % p;
113     cc = LL(b[i].r + 0.5) % p;
114     h[i] = ((111 * aa * (1 << 30) % p + 111 * bb * (1 << 15) % p + cc) % p + p) % p;
115 }
116 return h;
117 }
118 } // namespace FFT

```

---

### 3.13 FWT

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 const int MOD = 998244353;
5
6 inline int add(int x, int y) {
7     return x + y >= MOD ? x + y - MOD : x + y;
8 }
9 inline int mul(int x, int y) {
10    return 111 * x * y % MOD;
11 }
12 inline int sub(int x, int y) {
13    return x - y < 0 ? x - y + MOD : x - y;
14 }
15 inline int qp(int x, int y) {
16    int r = 1;
17    for (; y; y >>= 1) {
18        if (y & 1)
19            r = mul(r, x);
20        x = mul(x, x);
21    }
22    return r;
23 }
24 inline int inv(int x) {
25    return qp(x, MOD - 2);
26 }
27 inline int dvd(int x, int y) {
28    return 111 * x * qp(y, MOD - 2) % MOD;
29 }
30
31 namespace FWT {
32 void OR(int* a, int n) {
33     for (int o = 2, k = 1; o <= n; o <<= 1, k <<= 1)
34         for (int i = 0; i < n; i += o)
35             for (int j = 0; j < k; ++j)
36                 a[i + j + k] = add(a[i + j + k], a[i + j]);
37 }
38
39 void IOR(int* a, int n) {
40     for (int o = 2, k = 1; o <= n; o <<= 1, k <<= 1)
41         for (int i = 0; i < n; i += o)
42             for (int j = 0; j < k; ++j)
43                 a[i + j + k] = sub(a[i + j + k], a[i + j]);
44 }
45
46 void AND(int* a, int n) {
47     for (int o = 2, k = 1; o <= n; o <<= 1, k <<= 1)
48         for (int i = 0; i < n; i += o)
49             for (int j = 0; j < k; ++j)
50                 a[i + j] = add(a[i + j], a[i + j + k]);

```

```

51 }
52
53 void IAND(int* a, int n) {
54     for (int o = 2, k = 1; o <= n; o <<= 1, k <<= 1)
55         for (int i = 0; i < n; i += o)
56             for (int j = 0; j < k; ++j)
57                 a[i + j] = sub(a[i + j], a[i + j + k]);
58 }
59
60 void XOR(int* a, int n) {
61     int x, y;
62     for (int o = 2, k = 1; o <= n; o <<= 1, k <<= 1)
63         for (int i = 0; i < n; i += o)
64             for (int j = 0; j < k; ++j) {
65                 x = a[i + j], y = a[i + j + k];
66                 a[i + j] = add(x, y);
67                 a[i + j + k] = sub(x, y);
68             }
69 }
70
71 int inv2 = inv(2);
72 void IXOR(int* a, int n) {
73     int x, y;
74     for (int o = 2, k = 1; o <= n; o <<= 1, k <<= 1)
75         for (int i = 0; i < n; i += o)
76             for (int j = 0; j < k; ++j) {
77                 x = a[i + j], y = a[i + j + k];
78                 a[i + j] = mul(add(x, y), inv2);
79                 a[i + j + k] = mul(sub(x, y), inv2);
80             }
81 }
82 } // namespace FWT
83
84 const int N = (1 << 17) + 5;
85
86 int n;
87 int A[N], B[N], a[N], b[N], c[N];
88
89 int main() {
90     scanf("%d", &n);
91     n = 1 << n;
92
93     int x;
94     for (int i = 0; i < n; ++i)
95         scanf("%d", &A[i]);
96     for (int i = 0; i < n; ++i)
97         scanf("%d", &B[i]);
98
99     // OR
100     for (int i = 0; i < n; ++i)
101         a[i] = A[i], b[i] = B[i];
102     FWT::OR(a, n);
103     FWT::OR(b, n);
104     for (int i = 0; i < n; ++i)
105         c[i] = mul(a[i], b[i]);
106     FWT::IOR(c, n);
107
108     for (int i = 0; i < n - 1; ++i)
109         printf("%d ", c[i]);
110     printf("%d\n", c[n - 1]);
111
112     // AND
113     for (int i = 0; i < n; ++i)
114         a[i] = A[i], b[i] = B[i];
115     FWT::AND(a, n);

```

```

116 FWT::AND(b, n);
117 for (int i = 0; i < n; ++i)
118     c[i] = mul(a[i], b[i]);
119 FWT::IAND(c, n);
120 for (int i = 0; i < n - 1; ++i)
121     printf("%d ", c[i]);
122 printf("%d\n", c[n - 1]);
123
124 // XOR
125 for (int i = 0; i < n; ++i)
126     a[i] = A[i], b[i] = B[i];
127 FWT::XOR(a, n);
128 FWT::XOR(b, n);
129 for (int i = 0; i < n; ++i)
130     c[i] = mul(a[i], b[i]);
131 FWT::IXOR(c, n);
132 for (int i = 0; i < n - 1; ++i)
133     printf("%d ", c[i]);
134 printf("%d\n", c[n - 1]);
135 return 0;
136 }

```

---

### 3.14 LinearBasis

---

```

1 struct LinearBasis {
2     static const int B = 62;
3     ll b[B];
4     int tot, n;
5
6     LinearBasis() {
7         tot = 0;
8         n = 0;
9         memset(b, 0, sizeof(b));
10    }
11
12    bool insert(ll x) {
13        ++n;
14        for (int i = B - 1; i >= 0; --i) {
15            if (!(x >> i))
16                continue;
17            if (!b[i]) {
18                ++tot;
19                b[i] = x;
20                break;
21            }
22            x ^= b[i];
23        }
24        return x > 0;
25    }
26
27    bool query(ll x) {
28        for (int i = B - 1; i >= 0; --i) {
29            if (!(x >> i))
30                continue;
31            if (!b[i])
32                return false;
33            x ^= b[i];
34        }
35        return x == 0;
36    }
37
38    ll queryMax() {
39        ll res = 0;
40        for (int i = B - 1; i >= 0; --i) {

```

```

41     if ((res ^ b[i]) > res)
42         res ^= b[i];
43     }
44     return res;
45 }
46
47 ll queryMin() {
48     for (int i = 0; i < B; ++i)
49         if (b[i])
50             return b[i];
51     return -1;
52 }
53
54 ll count() {
55     return 1LL << tot;
56 }
57
58 void rebuild() {
59     for (int i = B - 1; i >= 0; --i) {
60         for (int j = i - 1; j >= 0; --j) {
61             if (b[i] & (1LL << j))
62                 b[i] ^= b[j];
63         }
64     }
65 }
66
67 // need rebuild first
68 ll queryKth(int k) {
69     if (k == 1 && tot < n)
70         return 0;
71     if (tot < n)
72         --k;
73     if (k > (1LL << tot) - 1)
74         return -1;
75     ll res = 0;
76     for (int i = 0; i < B; ++i) {
77         if (b[i]) {
78             if (k & 1)
79                 res ^= b[i];
80             k >>= 1;
81         }
82     }
83     return res;
84 }
85 };

```

---

### 3.15 Lucas

```

1 namespace Backlight {
2
3     // use this when n, m is really large and p is small
4     namespace Lucas {
5         inline ll pow(ll a, ll b, ll p) {
6             ll res = 1;
7             a %= p;
8             while (b) {
9                 if (b & 1)
10                     res = res * a % p;
11                 a = a * a % p;
12                 b >>= 1;
13             }
14             return res;
15         }
16

```



```

17 inline ll inv1(ll n, ll p) {
18     return pow(n, p - 2, p);
19 }
20
21 inline ll C1(ll n, ll m, ll p) {
22     if (m > n)
23         return 0;
24     if (m > n - m)
25         m = n - m;
26     ll u = 1, d = 1;
27     for (ll i = 1; i <= m; ++i) {
28         u = u * (n - i + 1) % p;
29         d = d * i % p;
30     }
31     return u * inv1(d, p) % p;
32 }
33
34 // solve n choose m (mod p) while p is a prime
35 ll lucas(ll n, ll m, ll p) {
36     if (m == 0)
37         return 1;
38     return C1(n % p, m % p, p) * lucas(n / p, m / p, p) % p;
39 }
40
41 ll exgcd(ll a, ll b, ll& x, ll& y) {
42     if (b == 0) {
43         x = 1;
44         y = 0;
45         return a;
46     }
47     ll d = exgcd(b, a % b, x, y);
48     ll z = x;
49     x = y;
50     y = z - y * (a / b);
51     return d;
52 }
53
54 inline ll inv2(ll n, ll p) {
55     ll x, y;
56     ll d = exgcd(n, p, x, y);
57     return d == 1 ? (p + x % p) % p : -1;
58 }
59
60 // n! mod pk without pi^x
61 ll f(ll n, ll pi, ll pk) {
62     if (!n)
63         return 1;
64     ll res = 1;
65     if (n / pk) {
66         for (ll i = 2; i <= pk; ++i)
67             if (i % pi)
68                 res = res * i % pk;
69         res = pow(res, n / pk, pk);
70     }
71     for (ll i = 2; i <= n % pk; ++i)
72         if (i % pi)
73             res = res * i % pk;
74     return res * f(n / pi, pi, pk) % pk;
75 }
76
77 ll C2(ll n, ll m, ll p, ll pi, ll pk) {
78     if (m > n)
79         return 0;
80     ll a = f(n, pi, pk), b = f(m, pi, pk), c = f(n - m, pi, pk);
81     ll k = 0;

```

```

82     for (ll i = n; i; i /= pi)
83         k += i / pi;
84     for (ll i = m; i; i /= pi)
85         k -= i / pi;
86     for (ll i = n - m; i; i /= pi)
87         k -= i / pi;
88     ll ans = a * inv2(b, pk) % pk * inv2(c, pk) % pk * pow(pi, k, pk) % pk;
89     ans = ans * (p / pk) % p * inv2(p / pk, pk) % p;
90     return ans;
91 }
92
93 // solve n choose m (mod p) while p might not be a prime
94 ll exlucas(ll n, ll m, ll p) {
95     ll x = p;
96     ll ans = 0;
97     for (ll i = 2; i <= p; ++i) {
98         if (x % i == 0) {
99             ll pk = 1;
100             while (x % i == 0)
101                 pk *= i, x /= i;
102             ans = (ans + C2(n, m, p, i, pk)) % p;
103         }
104     }
105     return ans;
106 }
107
108 } // namespace Lucas
109
110 } // namespace Backlight

```

### 3.16 min25

```

1  #include <bits/stdc++.h>
2
3  using namespace std;
4  using ll = int64_t;
5
6  const int MOD = 1e9 + 7;
7
8  template <typename T>
9  inline int mint(T x) {
10     x %= MOD;
11     if (x < 0)
12         x += MOD;
13     return x;
14 }
15 inline int add(int x, int y) {
16     return x + y >= MOD ? x + y - MOD : x + y;
17 }
18 inline int mul(int x, int y) {
19     return 1ll * x * y % MOD;
20 }
21 inline int sub(int x, int y) {
22     return x < y ? x - y + MOD : x - y;
23 }
24 inline int qp(int x, int y) {
25     int r = 1;
26     for (; y; y >>= 1) {
27         if (y & 1)
28             r = mul(r, x);
29         x = mul(x, x);
30     }
31     return r;
32 }

```

```

33 inline int inv(int x) {
34     return qp(x, MOD - 2);
35 }
36 inline int dvd(int x, int y) {
37     return 1ll * x * qp(y, MOD - 2) % MOD;
38 }
39 inline void inc(int& x, int y) {
40     x += y;
41     if (x >= MOD)
42         x -= MOD;
43 }
44 inline void dec(int& x, int y) {
45     x -= y;
46     if (x < 0)
47         x += MOD;
48 }
49
50 namespace min25 {
51     /*
52      calc the prefix sum of multiplicative function.
53      Requirements:
54          Assume p is a prime number.
55          1. f(p) is a polynomial with small deg or can be calc quickly.
56          2. f(p^e) can be calc quickly.
57      Time complexity:  $O(\frac{n^{0.75}}{\log n})$ 
58      Steps: assume  $\deg(f(p)) = n$ .
59          1. split f(p) into n parts by it exponent.
60          2. calc them separately.
61          3. then sum them up.
62      e.g.:  $f(p) = \phi(p) = p - 1$ 
63          1. calc  $ans_0$  for  $f(p) = p$ .
64          2. calc  $ans_1$  for  $f(p) = 1$ .
65          3.  $ans = ans_0 - ans_1$ .
66      */
67     const int LIM = 2e5 + 9;
68
69     ll gn, w[LIM];
70     int rt, lim, id1[LIM], id2[LIM];
71     #define idx(v) (v <= rt ? id1[v] : id2[gn / v])
72
73     int pcnt, prime[LIM];
74     bool isp[LIM];
75
76     //  $sp_{i,j} = \sum_{k=1}^j [p \text{ is a prime}] p^i$ 
77     int sp1[LIM];
78
79     void seive(const int& n) {
80         pcnt = 0;
81         fill(isp, isp + n + 1, true);
82         for (int i = 2; i <= n; ++i) {
83             if (isp[i]) {
84                 ++pcnt;
85                 prime[pcnt] = i;
86             }
87             for (int j = 1; j <= pcnt; ++j) {
88                 ll nxt = 1ll * i * prime[j];
89                 if (nxt > n)
90                     break;
91                 isp[nxt] = false;
92                 if (i % prime[j] == 0)
93                     break;
94             }
95         }
96         for (int i = 1; i <= pcnt; ++i)
97             sp1[i] = add(sp1[i - 1], prime[i]);

```

```

98 }
99
100 int G[LIM][2], H[LIM];
101
102 void initG0(const ll& n) {
103     lim = 0;
104     int inv2 = inv(2);
105     for (ll i = 1, j, v; i <= n; i = n / j + 1) {
106         j = n / i;
107
108         w[++lim] = j;
109         idx(j) = lim;
110
111         v = j % MOD;
112
113         // init  $G_0 = \sum_{i=2}^n g(i)$ 
114         G[lim][0] = sub(v, 1);
115         G[lim][1] = mul(mul(mint(v + 2), mint(v - 1)), inv2);
116     }
117 }
118
119 void calcH() {
120     for (int k = 1; k <= pcnt; ++k) {
121         const int p = prime[k];
122         const ll p2 = 1ll * p * p;
123         for (int i = 1; w[i] >= p2; ++i) {
124             const ll v = w[i] / p;
125             int id = idx(v);
126             dec(G[i][0], sub(G[id][0], k - 1));
127             dec(G[i][1], mul(p, sub(G[id][1], sp1[k - 1])));
128         }
129     }
130     for (int i = 1; i <= lim; ++i)
131         H[i] = sub(G[i][1], G[i][0]);
132 }
133
134 //  $f(p^e)$ 
135 inline int fpe(const int& p, const int& e) {
136     return p xor e;
137 }
138
139 int F(const int& k, const ll& n) {
140     if (n < prime[k] || n <= 1)
141         return 0;
142
143     int r1 = 0;
144     for (int i = k; i <= pcnt; ++i) {
145         ll pi = prime[i];
146         if (1ll * pi * pi > n)
147             break;
148         ll pc = pi, pc2 = pi * pi;
149         for (int c = 1; pc2 <= n; ++c) {
150             inc(r1, add(mul(fpe(pi, c), F(i + 1, n / pc)), fpe(pi, c + 1)));
151             pc = pc2;
152             pc2 = pc2 * pi;
153         }
154     }
155
156     //  $H(n) - H(p_{k-1})$ 
157     const int id = idx(n);
158     int r2 = sub(H[id], sub(sp1[k - 1], k - 1));
159     if (k == 1)
160         inc(r2, 2);
161     int ans = add(r1, r2);
162     return ans;

```

```

163 }
164
165 int solve(ll n) {
166     gn = n;
167     rt = sqrt(gn);
168     seive(rt + 5);
169     initG0(gn);
170     calCH();
171     return add(F(1, n), 1);
172 }
173 } // namespace min25
174
175 int main() {
176     ll n;
177     scanf("%lld", &n);
178     printf("%d\n", min25::solve(n));
179     return 0;
180 }

```

---

### 3.17 Mint

---

```

1 // Author: tourist
2 template <typename T>
3 T inverse(T a, T m) {
4     T u = 0, v = 1;
5     while (a != 0) {
6         T t = m / a;
7         m -= t * a;
8         swap(a, m);
9         u -= t * v;
10        swap(u, v);
11    }
12    assert(m == 1);
13    return u;
14 }
15
16 template <typename T>
17 class Modular {
18 public:
19     using Type = typename decay<decltype(T::value)>::type;
20
21     constexpr Modular()
22         : value() {}
23     template <typename U>
24     Modular(const U& x) {
25         value = normalize(x);
26     }
27
28     template <typename U>
29     static Type normalize(const U& x) {
30         Type v;
31         if (-mod() <= x && x < mod())
32             v = static_cast<Type>(x);
33         else
34             v = static_cast<Type>(x % mod());
35         if (v < 0)
36             v += mod();
37         return v;
38     }
39
40     const Type& operator()() const { return value; }
41     template <typename U>
42     explicit operator U() const { return static_cast<U>(value); }
43     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& rhs) {
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

```

```

174 template <typename T>
175 Modular<T> operator*(const Modular<T>& lhs, const Modular<T>& rhs) {
176     return Modular<T>(lhs) *= rhs;
177 }
178 template <typename T, typename U>
179 Modular<T> operator*(const Modular<T>& lhs, U rhs) {
180     return Modular<T>(lhs) *= rhs;
181 }
182 template <typename T, typename U>
183 Modular<T> operator*(U lhs, const Modular<T>& rhs) {
184     return Modular<T>(lhs) *= rhs;
185 }
186
187 template <typename T>
188 Modular<T> operator/(const Modular<T>& lhs, const Modular<T>& rhs) {
189     return Modular<T>(lhs) /= rhs;
190 }
191 template <typename T, typename U>
192 Modular<T> operator/(const Modular<T>& lhs, U rhs) {
193     return Modular<T>(lhs) /= rhs;
194 }
195 template <typename T, typename U>
196 Modular<T> operator/(U lhs, const Modular<T>& rhs) {
197     return Modular<T>(lhs) /= rhs;
198 }
199
200 template <typename T, typename U>
201 Modular<T> power(const Modular<T>& a, const U& b) {
202     assert(b >= 0);
203     Modular<T> x = a, res = 1;
204     U p = b;
205     while (p > 0) {
206         if (p & 1)
207             res *= x;
208         x *= x;
209         p >>= 1;
210     }
211     return res;
212 }
213
214 template <typename T>
215 bool IsZero(const Modular<T>& number) {
216     return number() == 0;
217 }
218
219 template <typename T>
220 string to_string(const Modular<T>& number) {
221     return to_string(number());
222 }
223
224 // U == std::ostream? but done this way because of fastoutput
225 template <typename U, typename T>
226 U& operator<<(U& stream, const Modular<T>& number) {
227     return stream << number();
228 }
229
230 // U == std::istream? but done this way because of fastinput
231 template <typename U, typename T>
232 U& operator>>(U& stream, Modular<T>& number) {
233     typename common_type<typename Modular<T>::Type, long long>::type x;
234     stream >> x;
235     number.value = Modular<T>::normalize(x);
236     return stream;
237 }
238

```



```

239 /*
240 using ModType = int;
241
242 struct VarMod { static ModType value; };
243 ModType VarMod::value;
244 ModType& md = VarMod::value;
245 using Mint = Modular<VarMod>;
246 */
247
248 const int md = 998244353;
249 using Mint = Modular<std::integral_constant<decay<decltype(MOD)>::type, MOD>>;
250
251 /*
252 vector<Mint> fact(1, 1);
253 vector<Mint> inv_fact(1, 1);
254
255 Mint C(int n, int k) {
256     if (k < 0 || k > n) {
257         return 0;
258     }
259     while ((int) fact.size() < n + 1) {
260         fact.push_back(fact.back() * (int) fact.size());
261         inv_fact.push_back(1 / fact.back());
262     }
263     return fact[n] * inv_fact[k] * inv_fact[n - k];
264 }
265 */

```

---

### 3.18 Mobius

```

1 int primes[N], pcnt;
2 bool is[N];
3 int mu[N]; // 莫比乌斯函数, 在这里是其前缀和
4 void seive() {
5     pcnt = 0;
6     mu[1] = 1;
7     for (int i = 2; i < N; ++i)
8         is[i] = true;
9     for (int i = 2; i < N; ++i) {
10         if (is[i])
11             primes[++pcnt] = i, mu[i] = -1;
12         for (int j = 1; j <= pcnt; ++j) {
13             ll nxt = 1ll * i * primes[j];
14             if (nxt >= N)
15                 break;
16             is[nxt] = false;
17             if (i % primes[j] == 0) {
18                 mu[nxt] = 0;
19                 break;
20             }
21             mu[nxt] = -mu[i];
22         }
23     }
24     for (int i = 1; i < N; ++i)
25         mu[i] += mu[i - 1];
26 }

```

---

### 3.19 Modular

```

1 const int MOD = 1e9 + 7;
2 int add(int x, int y) {
3     return x + y >= MOD ? x + y - MOD : x + y;

```

```

4 }
5 int mul(int x, int y) {
6     return 1ll * x * y % MOD;
7 }
8 int sub(int x, int y) {
9     return x - y < 0 ? x - y + MOD : x - y;
10 }
11 int dvd(int x, int y) {
12     return 1ll * x * qp(y, MOD - 2) % MOD;
13 }

```

---

## 3.20 NTT

---

```

1 namespace Backlight {
2
3 namespace NTT {
4     // 998244353, 1004535809
5     const int P = 998244353, G = 3, Gi = 332748118;
6
7     inline ll pow(ll a, ll b) {
8         ll res = 1;
9         a %= P;
10        while (b) {
11            if (b & 1)
12                res = res * a % P;
13            a = a * a % P;
14            b >>= 1;
15        }
16        return res;
17    }
18
19    int N, L;
20    vector<ll> r;
21    void init(vector<ll>& a, vector<ll>& b) {
22        int l = a.size() + b.size();
23        N = 1;
24        L = 0;
25        while (N < l)
26            N <<= 1, ++L;
27        a.resize(N);
28        b.resize(N);
29        r.resize(N);
30        for (int i = 0; i < N; ++i)
31            r[i] = (r[i >> 1] >> 1) | ((i & 1) << (L - 1));
32    }
33
34    void work(vector<ll>& a, int flag) {
35        for (int i = 0; i < N; i++)
36            if (i < r[i])
37                swap(a[i], a[r[i]]);
38        for (int mid = 1; mid < N; mid <<= 1) {
39            ll wn = pow(flag == 1 ? G : Gi, (P - 1) / (mid << 1));
40            for (int j = 0; j < N; j += (mid << 1)) {
41                ll w = 1;
42                for (int k = 0; k < mid; k++, w = (w * wn) % P) {
43                    int x = a[j + k], y = w * a[j + k + mid] % P;
44                    a[j + k] = (x + y) % P,
45                    a[j + k + mid] = (x - y + P) % P;
46                }
47            }
48        }
49    }
50
51    inline void NTT(vector<ll>& a) {

```



```
41 }
42
43 void init(int ps) {
44     psize = ps;
45     prime_table();
46 }
47
48 ll powl(ll a, ll n, ll p) {
49     ll ans = 1;
50     for (; n; n >>= 1) {
51         if (n & 1)
52             ans = mul(ans, a, p);
53         a = mul(a, a, p);
54     }
55     return ans;
56 }
57
58 bool witness(ll a, ll n) {
59     int t = 0;
60     ll u = n - 1;
61     for (; ~u & 1; u >>= 1)
62         t++;
63     ll x = powl(a, u, n), _x = 0;
64     for (; t; t--) {
65         _x = mul(x, x, n);
66         if (_x == 1 && x != 1 && x != n - 1)
67             return 1;
68         x = _x;
69     }
70     return _x != 1;
71 }
72
73 bool miller(ll n) {
74     if (n < 2)
75         return 0;
76     if (n <= psize)
77         return p[n] == n;
78     if (~n & 1)
79         return 0;
80     for (int j = 0; j <= 7; j++)
81         if (witness(rnd() % (n - 1) + 1, n))
82             return 0;
83     return 1;
84 }
85
86 ll gcd(ll a, ll b) {
87     ll ret = 1;
88     while (a != 0) {
89         if ((~a & 1) && (~b & 1))
90             ret <<= 1, a >>= 1, b >>= 1;
91         else if (~a & 1)
92             a >>= 1;
93         else if (~b & 1)
94             b >>= 1;
95         else {
96             if (a < b)
97                 swap(a, b);
98             a -= b;
99         }
100     }
101     return ret * b;
102 }
103
104 ll rho(ll n) {
105     for (;;) {
```

```

106     ll X = rnd() % n, Y, Z, T = 1, *lY = a, *lX = lY;
107     int tmp = 20;
108     C = rnd() % 10 + 3;
109     X = mul(X, X, n) + C;
110     *(lY++) = X;
111     lX++;
112     Y = mul(X, X, n) + C;
113     *(lY++) = Y;
114     for (; X != Y;) {
115         ll t = X - Y + n;
116         Z = mul(T, t, n);
117         if (Z == 0)
118             return gcd(T, n);
119         tmp--;
120         if (tmp == 0) {
121             tmp = 20;
122             Z = gcd(Z, n);
123             if (Z != 1 && Z != n)
124                 return Z;
125         }
126         T = Z;
127         Y = *(lY++) = mul(Y, Y, n) + C;
128         Y = *(lY++) = mul(Y, Y, n) + C;
129         X = *(lX++);
130     }
131 }
132 }
133
134 void _factor(ll n) {
135     for (int i = 0; i < cnt; i++) {
136         if (n % fac[i] == 0)
137             n /= fac[i], fac[cnt++] = fac[i];
138     }
139     if (n <= psize) {
140         for (; n != 1; n /= p[n])
141             fac[cnt++] = p[n];
142         return;
143     }
144     if (miller(n))
145         fac[cnt++] = n;
146     else {
147         ll x = rho(n);
148         _factor(x);
149         _factor(n / x);
150     }
151 }
152
153 void dfs(ll x, int dep) {
154     if (dep == _cnt)
155         d.push_back(x);
156     else {
157         dfs(x, dep + 1);
158         for (int i = 1; i <= _e[dep]; i++)
159             dfs(x * _pr[dep], dep + 1);
160     }
161 }
162
163 void norm() {
164     sort(fac, fac + cnt);
165     _cnt = 0;
166     for (int i = 0; i < cnt; ++i)
167         if (i == 0 || fac[i] != fac[i - 1])
168             _pr[_cnt] = fac[i], _e[_cnt++] = 1;
169     else
170         _e[_cnt - 1]++;

```

```

171 }
172
173 vector<ll> getd() {
174     d.clear();
175     dfs(1, 0);
176     return d;
177 }
178
179 /*****
180
181 // Attention: call init() before use
182
183 // get all factors
184 vector<ll> factorA(ll n) {
185     cnt = 0;
186     _factor(n);
187     norm();
188     vector<ll> d = getd();
189     sort(d.begin(), d.end());
190     return d;
191 }
192
193 // get prime factors
194 vector<ll> factorP(ll n) {
195     cnt = 0;
196     _factor(n);
197     norm();
198     vector<ll> d(cnt);
199     for (int i = 0; i < cnt; ++i)
200         d[i] = _pr[i];
201     return d;
202 }
203
204 // get prime factors,  $n = p_i^{e_i}$ 
205 vector<PLL> factorG(ll n) {
206     cnt = 0;
207     _factor(n);
208     norm();
209     vector<PLL> d(cnt);
210     for (int i = 0; i < cnt; ++i)
211         d[i] = make_pair(_pr[i], _e[i]);
212     return d;
213 }
214
215 bool is_primitive(ll a, ll p) {
216     assert(miller(p));
217     vector<PLL> D = factorG(p - 1);
218     for (int i = 0; i < (int)D.size(); ++i)
219         if (powl(a, (p - 1) / D[i].first, p) == 1)
220             return 0;
221     return 1;
222 }
223 } // namespace Pollard_Rho
224
225 } // namespace Backlight

```

### 3.22 poly-struct

```

1 constexpr int P = 998244353;
2 vector<int> rev, roots{0, 1};
3 int power(int a, int b) {
4     int r = 1;
5     while (b) {
6         if (b & 1)

```

```

7         r = 1ll * r * a % P;
8         a = 1ll * a * a % P;
9         b >>= 1;
10    }
11    return r;
12 }
13 void dft(vector<int>& a) {
14     int n = a.size();
15     if (int(rev.size()) != n) {
16         int k = __builtin_ctz(n) - 1;
17         rev.resize(n);
18         for (int i = 0; i < n; ++i)
19             rev[i] = rev[i >> 1] >> 1 | (i & 1) << k;
20     }
21     for (int i = 0; i < n; ++i)
22         if (rev[i] < i)
23             swap(a[i], a[rev[i]]);
24     if (int(roots.size()) < n) {
25         int k = __builtin_ctz(roots.size());
26         roots.resize(n);
27         while ((1 << k) < n) {
28             int e = power(3, (P - 1) >> (k + 1));
29             for (int i = 1 << (k - 1); i < (1 << k); ++i) {
30                 roots[2 * i] = roots[i];
31                 roots[2 * i + 1] = 1ll * roots[i] * e % P;
32             }
33             ++k;
34         }
35     }
36     for (int k = 1; k < n; k *= 2) {
37         for (int i = 0; i < n; i += 2 * k) {
38             for (int j = 0; j < k; ++j) {
39                 int u = a[i + j];
40                 int v = 1ll * a[i + j + k] * roots[k + j] % P;
41                 int x = u + v;
42                 if (x >= P)
43                     x -= P;
44                 a[i + j] = x;
45                 x = u - v;
46                 if (x < 0)
47                     x += P;
48                 a[i + j + k] = x;
49             }
50         }
51     }
52 }
53 void idft(vector<int>& a) {
54     int n = a.size();
55     reverse(a.begin() + 1, a.end());
56     dft(a);
57     int inv = power(n, P - 2);
58     for (int i = 0; i < n; ++i)
59         a[i] = 1ll * a[i] * inv % P;
60 }
61 struct poly {
62     vector<int> a;
63
64     poly() {}
65     poly(int f0) { a = {f0}; }
66     poly(const vector<int>& f)
67         : a(f) {
68         while (!a.empty() && !a.back())
69             a.pop_back();
70     }
71     poly(const vector<int>& f, int n)

```

```

72     : a(f) {
73     a.resize(n);
74     }
75     int size() const {
76     return a.size();
77     }
78     int deg() const {
79     return a.size() - 1;
80     }
81     int operator[](int idx) const {
82     if (idx < 0 || idx >= size())
83     return 0;
84     return a[idx];
85     }
86     void input(int n) {
87     a.resize(n);
88     FE(v, a)
89     rd(v);
90     }
91     void output(int n) {
92     for (int i = 0; i < n - 1; ++i)
93     printf("%d ", (*this)[i]);
94     printf("%d\n", (*this)[n - 1]);
95     }
96     poly mulxk(int k) const {
97     auto b = a;
98     b.insert(b.begin(), k, 0);
99     return poly(b);
100    }
101    poly modxk(int k) const {
102    k = min(k, size());
103    return poly(std::vector<int>(a.begin(), a.begin() + k));
104    }
105    poly alignxk(int k) const {
106    return poly(a, k);
107    }
108    poly divxk(int k) const {
109    if (size() <= k)
110    return poly();
111    return poly(vector<int>(a.begin() + k, a.end()));
112    }
113    friend poly operator+(const poly& f, const poly& g) {
114    int k = max(f.size(), g.size());
115    vector<int> res(k);
116    for (int i = 0; i < k; ++i) {
117    res[i] = f[i] + g[i];
118    if (res[i] >= P)
119    res[i] -= P;
120    }
121    return poly(res);
122    }
123    friend poly operator-(const poly& f, const poly& g) {
124    int k = max(f.size(), g.size());
125    vector<int> res(k);
126    for (int i = 0; i < k; ++i) {
127    res[i] = f[i] - g[i];
128    if (res[i] < 0)
129    res[i] += P;
130    }
131    return poly(res);
132    }
133    friend poly operator*(const poly& f, const poly& g) {
134    int sz = 1, k = f.size() + g.size() - 1;
135    while (sz < k)
136    sz *= 2;

```



```

137     vector<int> p = f.a, q = g.a;
138     p.resize(sz);
139     q.resize(sz);
140     dft(p);
141     dft(q);
142     for (int i = 0; i < sz; ++i)
143         p[i] = 1ll * p[i] * q[i] % P;
144     idft(p);
145     return poly(p);
146 }
147 friend poly operator/(const poly& f, const poly& g) {
148     return f.divide(g).first;
149 }
150 friend poly operator%(const poly& f, const poly& g) {
151     return f.divide(g).second;
152 }
153 poly& operator+=(const poly& f) {
154     return (*this) = (*this) + f;
155 }
156 poly& operator-=(const poly& f) {
157     return (*this) = (*this) - f;
158 }
159 poly& operator*=(const poly& f) {
160     return (*this) = (*this) * f;
161 }
162 poly& operator/=(const poly& f) {
163     return (*this) = divide(f).first;
164 }
165 poly& operator%=(const poly& f) {
166     return (*this) = divide(f).second;
167 }
168 poly derivative() const {
169     if (a.empty())
170         return poly();
171     int n = a.size();
172     vector<int> res(n - 1);
173     for (int i = 0; i < n - 1; ++i)
174         res[i] = 1ll * (i + 1) * a[i + 1] % P;
175     return poly(res);
176 }
177 poly integral() const {
178     if (a.empty())
179         return poly();
180     int n = a.size();
181     vector<int> res(n + 1);
182     for (int i = 0; i < n; ++i)
183         res[i + 1] = 1ll * a[i] * power(i + 1, P - 2) % P;
184     return poly(res);
185 }
186 poly rev() const {
187     return poly(vector<int>(a.rbegin(), a.rend()));
188 }
189 poly inv(int m) const {
190     poly x(power(a[0], P - 2));
191     int k = 1;
192     while (k < m) {
193         k *= 2;
194         x = (x * (2 - modxk(k) * x)).modxk(k);
195     }
196     return x.modxk(m);
197 }
198 poly log(int m) const {
199     return (derivative() * inv(m)).integral().modxk(m);
200 }
201 poly exp(int m) const {

```

```

202     poly x(1);
203     int k = 1;
204     while (k < m) {
205         k *= 2;
206         x = (x * (1 - x.log(k) + modxk(k))).modxk(k);
207     }
208     return x.modxk(m);
209 }
210 poly sqrt(int m) const {
211     poly x(1);
212     int k = 1;
213     while (k < m) {
214         k *= 2;
215         x = (x + (modxk(k) * x.inv(k)).modxk(k)) * ((P + 1) / 2);
216     }
217     return x.modxk(m);
218 }
219 poly sin() const {
220     int g = 3; // g: the ord of P
221     int i = power(g, (P - 1) / 4);
222     poly p = i * (*this);
223     p = p.exp(p.size());
224
225     poly q = (P - i) * (*this);
226     q = q.exp(q.size());
227
228     poly r = (p - q) * power(2 * i % P, P - 2);
229     return r;
230 }
231 poly cos() const {
232     int g = 3; // g: the ord of P
233     int i = power(g, (P - 1) / 4);
234     poly p = i * (*this);
235     p = p.exp(p.size());
236
237     poly q = (P - i) * (*this);
238     q = q.exp(q.size());
239
240     poly r = (p + q) * power(2, P - 2);
241     return r;
242 }
243 poly tan() const {
244     return sin() / cos();
245 }
246 poly cot() const {
247     return cos() / sin();
248 }
249 poly arcsin() {
250     poly sq = (*this) * (*this).modxk(size());
251     for (int i = 0; i < size(); ++i)
252         sq.a[i] = sq.a[i] ? P - sq.a[i] : 0;
253     sq.a[0] = 1 + sq.a[0];
254     if (sq.a[0] >= P)
255         sq.a[0] -= P;
256     poly r = (derivative() * sq.sqrt(size()).inv(size())).integral();
257     return r;
258 }
259 poly arccos() {
260     poly r = arcsin();
261     for (int i = 0; i < size(); ++i)
262         r.a[i] = r.a[i] ? P - r.a[i] : 0;
263     return r;
264 }
265 poly arctan() {
266     poly sq = (*this) * (*this).modxk(size());

```

```

267     sq.a[0] = 1 + sq.a[0];
268     if (sq.a[0] >= P)
269         sq.a[0] -= P;
270     poly r = (derivative() * sq.inv(size())).integral();
271     return r;
272 }
273 poly arccot() {
274     poly r = arctan();
275     for (int i = 0; i < size(); ++i)
276         r.a[i] = r.a[i] ? P - r.a[i] : 0;
277     return r;
278 }
279 poly mult(const poly& b) const {
280     if (b.size() == 0)
281         return poly();
282     int n = b.size();
283     return ((*this) * b.rev()).divxk(n - 1);
284 }
285 pair<poly, poly> divide(const poly& g) const {
286     int n = a.size(), m = g.size();
287     if (n < m)
288         return make_pair(poly(), a);
289
290     poly fR = rev();
291     poly gR = g.rev().alignxk(n - m + 1);
292     poly gRI = gR.inv(gR.size());
293
294     poly qR = (fR * gRI).modxk(n - m + 1);
295
296     poly q = qR.rev();
297
298     poly r = ((*this) - g * q).modxk(m - 1);
299
300     return make_pair(q, r);
301 }
302 vector<int> eval(vector<int> x) const {
303     if (size() == 0)
304         return vector<int>(x.size(), 0);
305     const int n = max(int(x.size()), size());
306     vector<poly> q(4 * n);
307     vector<int> ans(x.size());
308     x.resize(n);
309     function<void(int, int, int)> build = [&](int p, int l, int r) {
310         if (r - l == 1) {
311             q[p] = vector<int>{1, (P - x[l]) % P};
312         } else {
313             int m = (l + r) / 2;
314             build(2 * p, l, m);
315             build(2 * p + 1, m, r);
316             q[p] = q[2 * p] * q[2 * p + 1];
317         }
318     };
319     build(1, 0, n);
320     function<void(int, int, int, const poly&)> work = [&](int p, int l, int r, const poly& num) {
321         if (r - l == 1) {
322             if (l < int(ans.size()))
323                 ans[l] = num[0];
324         } else {
325             int m = (l + r) / 2;
326             work(2 * p, l, m, num.mult(q[2 * p + 1]).modxk(m - 1));
327             work(2 * p + 1, m, r, num.mult(q[2 * p]).modxk(r - m));
328         }
329     };
330     work(1, 0, n, mult(q[1].inv(n)));
331     return ans;

```

```

332 }
333 };

```

### 3.23 Poly

```

1 namespace Poly {
2   const int N = ...;
3   const int MAXN = N << 3;
4   const int P = 998244353;
5   const int G = 3;
6
7   ll qp(ll a, ll b) {
8     ll res = 1;
9     a %= P;
10    while (b) {
11      if (b & 1)
12        res = res * a % P;
13      a = a * a % P;
14      b >>= 1;
15    }
16    return res;
17  }
18
19  const int Gi = qp(G, P - 2);
20  const int I2 = qp(2, P - 2);
21  int r[MAXN];
22  ll t1[MAXN], t2[MAXN], t3[MAXN], t4[MAXN], t5[MAXN], t6[MAXN], t7[MAXN];
23
24  // int N, L;
25  // void init(int n) {
26  //   int N = 1, L = -1; while(N <= n << 1) N <<= 1, L++;
27  //   for(int i = 1; i < N; ++i) r[i] = (r[i >> 1] >> 1) | ((i & 1) << L);
28  // }
29
30  void inplaceNTT(ll* a, int n, int op) {
31    for (int i = 0; i < n; ++i)
32      if (i < r[i])
33        swap(a[i], a[r[i]]);
34    for (int m2 = 2, m = 1; m2 <= n; m = m2, m2 <<= 1) {
35      ll wn = qp(op == 1 ? G : Gi, (P - 1) / m2), x, y;
36      for (int l = 0; l < n; l += m2) {
37        ll w = 1;
38        for (int i = l; i < l + m; ++i) {
39          x = a[i], y = w * a[i + m] % P;
40          a[i] = (x + y) % P;
41          a[i + m] = (x + P - y) % P;
42          w = w * wn % P;
43        }
44      }
45    }
46    if (op == -1) {
47      ll inv = qp(n, P - 2);
48      for (int i = 0; i < n; ++i)
49        a[i] = a[i] * inv % P;
50    }
51  }
52  inline void NTT(ll* a, int n) {
53    inplaceNTT(a, n, 1);
54  }
55  inline void INTT(ll* a, int n) {
56    inplaceNTT(a, n, -1);
57  }
58
59  // 多项式微分 (求导)

```

```

60 inline void Derivative(ll* a, ll* b, int n) {
61     for (int i = 0; i < n; ++i)
62         b[i] = a[i + 1] * (i + 1) % P;
63     b[n - 1] = 0;
64 }
65
66 // 多项式积分
67 inline void Integral(ll* a, ll* b, int n) {
68     for (int i = 0; i < n; ++i)
69         b[i + 1] = a[i] * qp(i + 1, P - 2) % P;
70     b[0] = 0;
71 }
72
73 // 多项式翻转
74 //  $b(x) = x^n a(\frac{1}{x})$ 
75 inline void Reverse(ll* a, ll* b, int n) {
76     for (int i = 0; i < n; ++i)
77         b[i] = a[n - i - 1];
78 }
79
80 // 多项式乘法逆
81 //  $b(x) = a^{-1}(x) \bmod x^n$ 
82 void __Inverse(ll* a, ll* b, int n) {
83     if (n == 1) {
84         b[0] = qp(a[0], P - 2);
85         return;
86     }
87
88     __Inverse(a, b, (n + 1) >> 1);
89
90     int N = 1, l = -1;
91     while (N <= n << 1)
92         N <<= 1, l++;
93     for (int i = 1; i < N; ++i)
94         r[i] = (r[i >> 1] >> 1) | ((i & 1) << 1);
95
96     memcpy(t1, a, sizeof(a[0]) * n);
97     fill(t1 + n, t1 + N, 0);
98
99     NTT(t1, N);
100    NTT(b, N);
101    for (int i = 0; i < N; ++i)
102        b[i] = ((b[i] << 1) % P + P - t1[i] * b[i] % P * b[i] % P) % P;
103    INTT(b, N);
104
105    fill(b + n, b + N, 0);
106 }
107
108 inline void Inverse(ll* a, ll* b, int n) {
109     fill(b, b + (n << 2), 0);
110     __Inverse(a, b, n);
111 }
112
113 // 多项式对数函数
114 //  $b(x) = \ln a(x) \bmod x^n$ 
115 void Ln(ll* a, ll* b, int n) {
116     #define aD t3
117     #define aI t4
118
119     Derivative(a, aD, n);
120     Inverse(a, aI, n);
121     int N = 1, l = -1;
122     while (N <= n << 1)
123         N <<= 1, l++;
124     for (int i = 1; i < N; ++i)

```

```

125     r[i] = (r[i >> 1] >> 1) | ((i & 1) << 1);
126     NTT(aD, N);
127     NTT(aI, N);
128     for (int i = 0; i < N; ++i)
129         aD[i] = aD[i] * aI[i] % P;
130     INTT(aD, N);
131     Integral(aD, b, n);
132
133     #undef aD
134     #undef aI
135 }
136
137 // 多项式指数函数
138 //  $b(x) = \exp a(x) \bmod x^n$ 
139 void Exp(ll* a, ll* b, int n) {
140     #define Lnb t2
141
142     if (n == 1) {
143         b[0] = 1;
144         return;
145     }
146     Exp(a, b, (n + 1) >> 1);
147     Ln(b, Lnb, n);
148     int N = 1, l = -1;
149     while (N <= n << 1)
150         N <<= 1, l++;
151     for (int i = 1; i < N; ++i)
152         r[i] = (r[i >> 1] >> 1) | ((i & 1) << 1);
153
154     memcpy(t1, a, sizeof(a[0]) * n);
155     fill(t1 + n, t1 + N, 0);
156     fill(Lnb + n, Lnb + N, 0);
157
158     for (int i = 0; i < N; ++i)
159         t1[i] = ((t1[i] - Lnb[i]) % P + P) % P;
160     ++t1[0];
161     NTT(b, N);
162     NTT(t1, N);
163     for (int i = 0; i < N; ++i)
164         b[i] = b[i] * t1[i] % P;
165     INTT(b, N);
166
167     fill(b + n, b + N, 0);
168     #undef Lnb
169 }
170
171 // 多项式乘法 (卷积)
172 //  $c(x) = a(x) * b(x) \bmod x^{(n+m)}$ 
173 //  $\deg c = n + m - 1$ 
174 void Convolution(ll* a, int n, ll* b, int m, ll* c) {
175     int N = 1, l = -1;
176     while (N <= (n + m) << 1)
177         N <<= 1, l++;
178     for (int i = 1; i < N; ++i)
179         r[i] = (r[i >> 1] >> 1) | ((i & 1) << 1);
180
181     memcpy(t1, a, sizeof(a[0]) * n);
182     fill(t1 + n, t1 + N, 0);
183     memcpy(t2, b, sizeof(b[0]) * m);
184     fill(t2 + m, t2 + N, 0);
185
186     NTT(t1, N);
187     NTT(t2, N);
188     for (int i = 0; i < N; ++i)
189         c[i] = t1[i] * t2[i] % P;

```

```

190     INTT(c, N);
191     fill(c + n + m, c + N, 0);
192 }
193 #define Multiply Convolution
194
195 // 多项式除法
196 //  $a(x) = b(x)Q(x) + R(x)$ 
197 // deg  $Q = n - m + 1$ 
198 // deg  $R = m - 1$ 
199 void Divide(ll* a, int n, ll* b, int m, ll* Q, ll* R) {
200     #define aR t3
201     #define bR t4
202     #define bRi t5
203     #define QR t6
204     #define bQ t7
205
206     int degQ = n - m + 1;
207     int degR = m - 1;
208
209     Reverse(a, aR, n);
210     Reverse(b, bR, m);
211     for (int i = degQ; i < m; ++i)
212         bR[i] = 0;
213
214     // get  $Q(x)$ 
215     Inverse(bR, bRi, degQ);
216     Multiply(aR, n, bRi, degQ, QR);
217     Reverse(QR, Q, degQ);
218
219     // get  $R(x)$ 
220     Multiply(b, m, Q, degQ, bQ);
221     for (int i = 0; i < degR; ++i)
222         R[i] = (a[i] - bQ[i] + P) % P;
223
224     #undef aR
225     #undef bR
226     #undef bRi
227     #undef QR
228 }
229
230 // 多项式求平方根
231 //  $b^{\{2\}}(x) = a(x)$ 
232 #define bI t3
233 void __Sqrt(ll* a, ll* b, int n) {
234     if (n == 1) {
235         b[0] = 1;
236         return;
237     }
238
239     __Sqrt(a, b, (n + 1) >> 1);
240
241     Inverse(b, bI, n);
242     Multiply(a, n, bI, n, bI);
243     for (int i = 0; i < n; ++i)
244         b[i] = (b[i] + bI[i]) * I2 % P;
245 }
246 inline void Sqrt(ll* a, ll* b, int n) {
247     fill(bI, bI + (n << 2), 0);
248     __Sqrt(a, b, n);
249 }
250 #undef bI
251
252 struct poly {
253     vector<ll> a;
254     int size() const { return a.size(); }

```

```
255 int deg() const { return size() - 1; }
256 ll& operator[](int i) {
257     assert(i < size());
258     return a[i];
259 }
260 ll operator[](int i) const { return i < size() ? a[i] : 0LL; }
261 void reverse() { std::reverse(a.begin(), a.end()); }
262 void resize(int n) { a.resize(n); }
263 poly(int n = 0)
264     : a(n, 0) {}
265
266 void DEBUG() {
267     cerr << "Poly DEBUG: " << endl;
268     for (const ll& v : a)
269         cerr << v << " ";
270     cerr << endl;
271 }
272
273 void DEBUG() const {
274     cerr << "Poly DEBUG: " << endl;
275     for (const ll& v : a)
276         cerr << v << " ";
277     cerr << endl;
278 }
279
280 void input() {
281     for (ll& x : a)
282         read(x);
283 }
284
285 void output() {
286     if (a.empty()) {
287         puts("");
288         return;
289     }
290     int n = a.size();
291     for (int i = 0; i < n - 1; ++i)
292         printf("%lld ", a[i]);
293     printf("%lld\n", a[n - 1]);
294 }
295
296 void output() const {
297     if (a.empty()) {
298         puts("");
299         return;
300     }
301     int n = a.size();
302     for (int i = 0; i < n - 1; ++i)
303         printf("%lld ", a[i]);
304     printf("%lld\n", a[n - 1]);
305 }
306
307 poly inv(int n = -1) const {
308     if (n == -1)
309         n = size();
310     static ll f[MAXN], g[MAXN];
311     for (int i = 0; i < n; ++i)
312         f[i] = a[i];
313     Inverse(f, g, n);
314     poly res(n);
315     for (int i = 0; i < n; ++i)
316         res[i] = g[i];
317     return res;
318 }
319
```



```

320 poly rev() const {
321     int n = size();
322     poly r(n);
323     for (int i = 0; i < n; ++i)
324         r[i] = a[n - i - 1];
325     return r;
326 }
327
328 poly sqrt() {
329     int n = a.size();
330     static ll f[MAXN], g[MAXN];
331     for (int i = 0; i < n; ++i)
332         f[i] = a[i];
333     Sqrt(f, g, n);
334     poly res(n);
335     for (int i = 0; i < n; ++i)
336         res[i] = g[i];
337     return res;
338 }
339 };
340
341 poly operator+(const poly& a, const poly& b) {
342     int k = max(a.size(), b.size());
343     poly c(k);
344     for (int i = 0; i < k; ++i)
345         c[i] = (a[i] + b[i]) % P;
346     return c;
347 }
348
349 poly operator-(const poly& a, const poly& b) {
350     int k = max(a.size(), b.size());
351     poly c(k);
352     for (int i = 0; i < k; ++i)
353         c[i] = (a[i] - b[i] + P) % P;
354     return c;
355 }
356
357 poly operator*(const poly& a, const poly& b) {
358     static ll ta[MAXN], tb[MAXN];
359     int n = a.size(), m = b.size(), k = n + m - 1;
360     for (int i = 0; i < n; ++i)
361         ta[i] = a[i];
362     for (int i = 0; i < m; ++i)
363         tb[i] = b[i];
364
365     Multiply(ta, n, tb, m, ta);
366
367     poly c(k);
368     for (int i = 0; i < k; ++i)
369         c[i] = ta[i];
370     return c;
371 }
372
373 pair<poly, poly> Divide(const poly& a, const poly& b) {
374     static ll ta[MAXN], tb[MAXN], tq[MAXN], tr[MAXN];
375     int n = a.size(), m = b.size();
376     if (n < m)
377         return make_pair(poly(0), a);
378
379     int degQ = n - m + 1, degR = m - 1;
380     for (int i = 0; i < n; ++i)
381         ta[i] = a[i];
382     for (int i = 0; i < m; ++i)
383         tb[i] = b[i];
384

```

```

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

---

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  using ll = int64_t;
4
5  constexpr int MOD = 1e9 + 7; // 998244353 1e9 + 7
6  template <typename T>
7  inline int mint(T x) {
8      x %= MOD;
9      if (x < 0)
10         x += MOD;
11     return x;
12 }
13 inline int add(int x, int y) {
14     return x + y >= MOD ? x + y - MOD : x + y;
15 }
16 inline int mul(int x, int y) {
17     return 1ll * x * y % MOD;
18 }
19 inline int sub(int x, int y) {
20     return x < y ? x - y + MOD : x - y;
21 }
22 inline int qp(int x, int y) {
23     int r = 1;
24     for (; y; y >>= 1) {
25         if (y & 1)
26             r = mul(r, x);
27         x = mul(x, x);
28     }
29     return r;
30 }
31 inline int inv(int x) {
32     return qp(x, MOD - 2);
33 }
34
35 namespace PNS {
36     const int N = 2e6 + 5;
37     const int M = 35;
38
39     ll global_n;
40
41     int g[N], sg[N];
42
43     int h[N][M];
44     bool vis_h[N][M];
45
46     int ans;
47
48     int pcnt, prime[N], phi[N];
49     bool isp[N];
50
51     void sieve(int n) {
52         pcnt = 0;
53         for (int i = 2; i <= n; ++i)
54             isp[i] = true;
55         phi[1] = 1;
56         for (int i = 2; i <= n; ++i) {
57             if (isp[i]) {
58                 ++pcnt;
59                 prime[pcnt] = i;
60                 phi[i] = i - 1;
61             }
62             for (int j = 1; j <= pcnt; ++j) {
63                 ll nxt = 1ll * i * prime[j];

```

```

64     if (nxt > n)
65         break;
66     isp[nxt] = false;
67     if (i % prime[j] == 0) {
68         phi[nxt] = phi[i] * prime[j];
69         break;
70     }
71     phi[nxt] = phi[i] * phi[prime[j]];
72 }
73 }
74
75 for (int i = 1; i <= n; ++i)
76     g[i] = mul(i, phi[i]);
77
78 sg[0] = 0;
79 for (int i = 1; i <= n; ++i)
80     sg[i] = add(sg[i - 1], g[i]);
81 }
82
83 int inv2, inv6;
84 void init() {
85     sieve(N - 1);
86     for (int i = 1; i <= pcnt; ++i)
87         h[i][0] = 1, h[i][1] = 0;
88     for (int i = 1; i <= pcnt; ++i)
89         vis_h[i][0] = vis_h[i][1] = true;
90     inv2 = inv(2);
91     inv6 = inv(6);
92 }
93
94 int S1(ll n) {
95     return mul(mul(mint(n), mint(n + 1)), inv2);
96 }
97
98 int S2(ll n) {
99     return mul(mul(mint(n), mul(mint(n + 1), mint(n * 2 + 1))), inv6);
100 }
101
102 map<ll, int> mp_g;
103
104 int G(ll n) {
105     if (n < N)
106         return sg[n];
107     if (mp_g.count(n))
108         return mp_g[n];
109
110     int ret = S2(n);
111     for (ll i = 2, j; i <= n; i = j + 1) {
112         j = n / (n / i);
113         ret = sub(ret, mul(sub(S1(j), S1(i - 1)), G(n / i)));
114     }
115     mp_g[n] = ret;
116     return ret;
117 }
118
119 void dfs(ll d, int hd, int pid) {
120     ans = add(ans, mul(hd, G(global_n / d)));
121
122     for (int i = pid, p; i <= pcnt; ++i) {
123         if (i > 1 && d > global_n / prime[i] / prime[i])
124             break;
125
126         int c = 2;
127         for (ll x = d * prime[i] * prime[i]; x <= global_n; x *= prime[i], ++c) {
128             if (!vis_h[i][c]) {

```

```

129     int f = qp(prime[i], c);
130     f = mul(f, sub(f, 1));
131     int g = mul(prime[i], prime[i] - 1);
132     int t = mul(prime[i], prime[i]);
133
134     for (int j = 1; j <= c; ++j) {
135         f = sub(f, mul(g, h[i][c - j]));
136         g = mul(g, t);
137     }
138     h[i][c] = f;
139     vis_h[i][c] = true;
140 }
141
142 if (h[i][c])
143     dfs(x, mul(hd, h[i][c]), i + 1);
144 }
145 }
146 }
147
148 int solve(ll n) {
149     global_n = n;
150     ans = 0;
151     dfs(1, 1, 1);
152     return ans;
153 }
154 } // namespace PNS
155
156 int main() {
157     PNS::init();
158     ll n;
159     scanf("%lld", &n);
160     printf("%d\n", PNS::solve(n));
161     return 0;
162 }

```

### 3.25 Simplex

```

1  /**
2   * Simplex Alogorithm:
3   * solve  $\max z = \sum_{j=1}^n c_j x_j$ 
4   * with restrictions like:  $\sum_{j=1}^n a_{ij} x_j = b_j, i = 1, 2, \dots, m$ 
5   *  $x_j \geq 0$ 
6   * in  $O(knm)$ , where  $k$  is a const number.
7   *
8   * Tips: 1.  $\min \Rightarrow -\min \Rightarrow \max$ 
9   *        2.  $x_1 + 2x_2 \leq 9 \Rightarrow x_1 + x_2 + x_3 = 9, x_3 \geq 0$ 
10  *        3.  $x_k$  without restrictions  $\Rightarrow x_k = x_m - x_m$  and  $x_m, x_n \geq 0$ 
11  *
12  * Notes: 1.  $c = A_{\{0\}}$ 
13  *          2.  $z = \max cx$ 
14  *          3.  $Ax = b$ 
15  */
16  enum {
17      OK = 1,
18      UNBOUNDED = 2,
19      INFEASIBLE = 3
20  };
21  struct Simplex {
22      constexpr static double eps = 1e-10;
23
24      int n, m;
25      int flag;
26      double z;
27      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     pivot(x, y);
94 }
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;

```

```

158     }
159 };

```

---

## 3.26 SimpsonIntegral

---

```

1 namespace SimpsonIntegral {
2 // calculate  $\int_l^r f(x) dx$ 
3
4 double f(double x) {
5     return (c * x + d) / (a * x + b);
6 }
7
8 double simpson(double l, double r) {
9     double mid = (l + r) / 2;
10    return (r - l) * (f(l) + 4 * f(mid) + f(r)) / 6;
11 }
12
13 double integral(double l, double r, double eps, double ans) {
14     double mid = (l + r) / 2;
15     double fl = simpson(l, mid), fr = simpson(mid, r);
16     if (abs(fl + fr - ans) <= 15 * eps)
17         return fl + fr + (fl + fr - ans) / 15;
18     return integral(l, mid, eps / 2, fl) + integral(mid, r, eps / 2, fr);
19 }
20
21 double integral(double l, double r, double eps = 1e-8) {
22     return integral(l, r, eps, simpson(l, r));
23 }
24 } // namespace SimpsonIntegral

```

---

## 4 other

### 4.1 BFPRT

---

```

1 /**
2  * BFPRT: find the kth element of an array in  $O(n)$  using Divide and Conquer method.
3  * you can use std::nth_element(a, a + k, a + n) instead
4  */
5 namespace BFPRT {
6 template <typename T, typename Cmp>
7 T kth_index(T* a, int l, int r, int k, Cmp cmp);
8
9 template <typename T, typename Cmp>
10 int insert_sort(T* a, int l, int r, Cmp cmp) {
11     for (int i = l + 1; i <= r; ++i) {
12         int tmp = a[i];
13         int j = i - 1;
14         while (j >= l && a[j] > tmp) {
15             a[j + 1] = a[j];
16             --j;
17         }
18         a[j + 1] = tmp;
19     }
20     return l + (r - l) / 2;
21 }
22
23 template <typename T, typename Cmp>
24 int pivot(T* a, int l, int r, Cmp cmp) {
25     if (r - l < 5)
26         return insert_sort(a, l, r, cmp);
27     int lst = l - 1;

```



```

28     for (int i = l; i + 4 <= r; i += 5) {
29         int p = insert_sort(a, i, i + 4, cmp);
30         swap(a[++lst], a[p]);
31     }
32     return kth_index<T>(a, l, lst, (lst - l + 1) / 2 + 1, cmp);
33 }
34
35 template <typename T, typename Cmp>
36 int partition(T* a, int l, int r, Cmp cmp) {
37     int p = pivot(a, l, r, cmp);
38     swap(a[p], a[r]);
39     int lst = l - 1;
40     for (int i = l; i < r; ++i) {
41         if (cmp(a[i], a[r]))
42             swap(a[++lst], a[i]);
43     }
44     swap(a[++lst], a[r]);
45     return lst;
46 }
47
48 template <typename T, typename Cmp>
49 T kth_index(T* a, int l, int r, int k, Cmp cmp) {
50     int p = partition(a, l, r, cmp);
51     int d = p - l + 1;
52     if (d == k)
53         return a[p];
54     else if (d < k)
55         return kth_index(a, p + 1, r, k - d, cmp);
56     else
57         return kth_index(a, l, p - 1, k, cmp);
58 }
59
60 template <typename T>
61 T kth_index(T* a, int l, int r, int k) {
62     return kth_index(a, l, r, k, less<T>());
63 }
64 }; // namespace BFPRT

```

## 4.2 cpp-header

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  typedef long long ll;
5  typedef unsigned long long ull;
6  typedef pair<int, int> PII;
7  typedef vector<int> VI;
8  typedef vector<ll> VL;
9  typedef vector<vector<int>> VVI;
10 typedef vector<vector<ll>> VVL;
11
12 #define REP(i, __, __) for (int i = (__); i < (__); ++i)
13 #define PER(i, __, __) for (int i = (__ - 1); i >= (__); --i)
14 #define FOR(i, __, __) for (int i = (__); i <= (__); ++i)
15 #define ROF(i, __, __) for (int i = (__); i >= (__); --i)
16 #define FE(v, V) for (auto& v : V)
17
18 #define EB emplace_back
19 #define PB push_back
20 #define MP make_pair
21 #define FI first
22 #define SE second
23 #define SZ(x) ((int)(x).size())
24 #define ALL(x) (x).begin(), (x).end()

```

```

25 #define LLA(x) (x).rbegin(), (x).rend()
26
27 const double PI = acos(-1.0);
28
29 namespace Backlight {
30 const int __BUFFER_SIZE__ = 1 << 20;
31 bool NEOF = 1; //为 0 表示文件结尾
32 int __top;
33 char __buf[__BUFFER_SIZE__], *__p1 = __buf, *__p2 = __buf, __stk[996];
34
35 inline bool isdowncase(char c) {
36     return (c >= 'a') && (c <= 'z');
37 }
38 inline bool isupcase(char c) {
39     return (c >= 'A') && (c <= 'Z');
40 }
41 inline bool isdigit(char c) {
42     return (c >= '0') && (c <= '9');
43 }
44
45 template <typename T>
46 T MIN(T a, T b) {
47     return min(a, b);
48 }
49
50 template <typename First, typename... Rest>
51 First MIN(First f, Rest... r) {
52     return min(f, MIN(r...));
53 }
54
55 template <typename T>
56 T MAX(T a, T b) {
57     return max(a, b);
58 }
59
60 template <typename First, typename... Rest>
61 First MAX(First f, Rest... r) {
62     return max(f, MAX(r...));
63 }
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) ? (NEO
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 == '#'; }
129 inline bool need(char c) {
130     return isdowncase(c) || isupcase(c) || isdigit(c);
131 }
132
133 inline bool read_str(char* a) {
134     while ((*a = nc()) && need(*a) && NEOF)
135         ++a;
136     *a = '\0';
137     return NEOF;
138 }
139
140 template <typename T>
141 inline void print(T x) {
142     if (x < 0)
143         putchar('-'), x = -x;
144     if (x == 0) {
145         putchar('0');
146         return;
147     }
148     __top = 0;
149     while (x) {
150         __stk[++__top] = x % 10 + '0';
151         x /= 10;
152     }
153     while (__top) {
154         putchar(__stk[__top]);

```

```
155     --__top;
156 }
157 }
158
159 template <typename First, typename... Rest>
160 inline void print(First f, Rest... r) {
161     print(f);
162     putchar(' ');
163     print(r...);
164 }
165
166 template <typename T>
167 inline void println(T x) {
168     print(x);
169     putchar('\n');
170 }
171
172 template <typename First, typename... Rest>
173 inline void println(First f, Rest... r) {
174     print(f);
175     putchar(' ');
176     println(r...);
177 }
178
179 template <typename T>
180 inline void _dbg(const char* format, T value) {
181     cerr << format << '=' << value << endl;
182 }
183
184 template <typename First, typename... Rest>
185 inline void _dbg(const char* format, First f, Rest... r) {
186     while (*format != ',')
187         cerr << *format++;
188     cerr << '=' << f << ", ";
189     _dbg(format + 1, r...);
190 }
191
192 template <typename T>
193 ostream& operator<<(ostream& os, vector<T> V) {
194     os << "[";
195     for (auto v : V)
196         os << v << ", ";
197     return os << "]";
198 }
199
200 template <typename T>
201 ostream& operator<<(ostream& os, set<T> V) {
202     os << "[";
203     for (auto v : V)
204         os << v << ", ";
205     return os << "]";
206 }
207
208 template <typename T>
209 ostream& operator<<(ostream& os, multiset<T> V) {
210     os << "[";
211     for (auto v : V)
212         os << v << ", ";
213     return os << "]";
214 }
215
216 template <typename T1, typename T2>
217 ostream& operator<<(ostream& os, map<T1, T2> V) {
218     os << "[";
219     for (auto v : V)
```

```

220     OS << V << ",";
221     return OS << " ]";
222 }
223
224 template <typename L, typename R>
225 ostream& operator<<(ostream& os, pair<L, R> P) {
226     return os << "(" << P.first << "," << P.second << ")";
227 }
228
229 #ifdef BACKLIGHT
230 #define debug(...) \
231     cerr << "\033[31m"; \
232     _dbg(#__VA_ARGS__, __VA_ARGS__); \
233     cerr << "\033[0m";
234 // #define debug(...) _dbg(#__VA_ARGS__, __VA_ARGS__);
235 #else
236 #define debug(...)
237 #endif
238 } // namespace Backlight
239
240 /*****          Backlight          *****/
241 * 一发入魂
242 * 仔细读题，注意边界条件
243 * 没有思路就试试逆向思维
244 * wdnmd! 我柜子动了不打了
245 * 能不能把我掉的分还给我
246 *****/
247
248 // mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
249 // int rnd(int L, int r) { return L + rng() % (r - L + 1); }
250
251 using namespace Backlight;
252 const int N = 2e7 + 5;
253 const int M = 5e5 + 5;
254 const int V = 5e7 + 5;
255 const ll MOD = 1e9 + 7; // 998244353 1e9+7
256 const int INF = 0x3f3f3f3f; // 1e9+7 0x3f3f3f3f
257 const ll LLINF = 0x3f3f3f3f3f3f3f3f; // 1e18+9 0x3f3f3f3f3f3f3f3f
258 const double eps = 1e-8;
259
260 void solve(int Case) { // printf("Case #%d: ", Case);
261 }
262
263 int main() {
264 #ifdef BACKLIGHT
265     freopen("in.txt", "r", stdin);
266     auto begin = std::chrono::steady_clock::now();
267 #endif
268
269     // ios::sync_with_stdio(false); cin.tie(0); cout.tie(0);
270     int T = 1;
271     // read(T);
272     for (int _ = 1; _ <= T; _++)
273         solve(_);
274
275 #ifdef BACKLIGHT
276     freopen("in.txt", "r", stdin);
277     auto end = std::chrono::steady_clock::now();
278     auto duration = std::chrono::duration_cast<std::chrono::milliseconds>(end - begin);
279     cerr << "\033[32mTime Elapsed: " << (double)(duration.count()) << " ms\033[0m" << endl;
280 #endif
281     return 0;
282 }

```

### 4.3 debug

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  typedef long long ll;
5  typedef unsigned long long ull;
6  typedef pair<int, int> PII;
7  typedef vector<int> VI;
8  typedef vector<ll> VL;
9  typedef vector<vector<int>> VVI;
10 typedef vector<vector<ll>> VVL;
11
12 #define REP(i, __, __) for (int i = (__); i < (__); ++i)
13 #define PER(i, __, __) for (int i = (__ - 1); i >= (__); --i)
14 #define FOR(i, __, __) for (int i = (__); i <= (__); ++i)
15 #define ROF(i, __, __) for (int i = (__); i >= (__); --i)
16 #define FE(v, V) for (auto& v : V)
17
18 #define EB emplace_back
19 #define PB push_back
20 #define MP make_pair
21 #define FI first
22 #define SE second
23 #define SZ(x) ((int)(x).size())
24 #define ALL(x) (x).begin(), (x).end()
25 #define LLA(x) (x).rbegin(), (x).rend()
26
27 const double PI = acos(-1.0);
28
29 namespace Backlight {
30 const int __BUFFER_SIZE__ = 1 << 20;
31 bool NEOF = 1;  //为 0 表示文件结尾
32 int __top;
33 char __buf[__BUFFER_SIZE__], *__p1 = __buf, *__p2 = __buf, __stk[996];
34
35 inline bool isdowncase(char c) {
36     return (c >= 'a') && (c <= 'z');
37 }
38 inline bool isupcase(char c) {
39     return (c >= 'A') && (c <= 'Z');
40 }
41 inline bool isdigit(char c) {
42     return (c >= '0') && (c <= '9');
43 }
44
45 template <typename T>
46 T MIN(T a, T b) {
47     return min(a, b);
48 }
49
50 template <typename First, typename... Rest>
51 First MIN(First f, Rest... r) {
52     return min(f, MIN(r...));
53 }
54
55 template <typename T>
56 T MAX(T a, T b) {
57     return max(a, b);
58 }
59
60 template <typename First, typename... Rest>
61 First MAX(First f, Rest... r) {
62     return max(f, MAX(r...));
63 }

```

```

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) ? (NEOF) : (*__p1++);
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 == '#'; }

```

```
129 inline bool need(char c) {
130     return isdowncase(c) || isupcase(c) || isdigit(c);
131 }
132
133 inline bool read_str(char* a) {
134     while ((*a = nc()) && need(*a) && !NEOF)
135         ++a;
136     *a = '\0';
137     return NEOF;
138 }
139
140 template <typename T>
141 inline void print(T x) {
142     if (x < 0)
143         putchar('-'), x = -x;
144     if (x == 0) {
145         putchar('0');
146         return;
147     }
148     __top = 0;
149     while (x) {
150         __stk[++__top] = x % 10 + '0';
151         x /= 10;
152     }
153     while (__top) {
154         putchar(__stk[__top]);
155         --__top;
156     }
157 }
158
159 template <typename First, typename... Rest>
160 inline void print(First f, Rest... r) {
161     print(f);
162     putchar(' ');
163     print(r...);
164 }
165
166 template <typename T>
167 inline void println(T x) {
168     print(x);
169     putchar('\n');
170 }
171
172 template <typename First, typename... Rest>
173 inline void println(First f, Rest... r) {
174     print(f);
175     putchar(' ');
176     println(r...);
177 }
178
179 template <typename T>
180 inline void _dbg(const char* format, T value) {
181     cerr << format << '=' << value << endl;
182 }
183
184 template <typename First, typename... Rest>
185 inline void _dbg(const char* format, First f, Rest... r) {
186     while (*format != ',')
187         cerr << *format++;
188     cerr << '=' << f << ", ";
189     _dbg(format + 1, r...);
190 }
191
192 template <typename T>
193 ostream& operator<<(ostream& os, vector<T> V) {
```



```

194     os << "[ ";
195     for (auto v : V)
196         os << v << ",";
197     return os << " ]";
198 }
199
200 template <typename T>
201 ostream& operator<<(ostream& os, set<T> V) {
202     os << "[ ";
203     for (auto v : V)
204         os << v << ",";
205     return os << " ]";
206 }
207
208 template <typename T>
209 ostream& operator<<(ostream& os, multiset<T> V) {
210     os << "[ ";
211     for (auto v : V)
212         os << v << ",";
213     return os << " ]";
214 }
215
216 template <typename T1, typename T2>
217 ostream& operator<<(ostream& os, map<T1, T2> V) {
218     os << "[ ";
219     for (auto v : V)
220         os << v << ",";
221     return os << " ]";
222 }
223
224 template <typename L, typename R>
225 ostream& operator<<(ostream& os, pair<L, R> P) {
226     return os << "(" << P.first << "," << P.second << ")";
227 }
228
229 #ifdef BACKLIGHT
230 #define debug(...) \
231     cerr << "\033[31m"; \
232     _dbg(#__VA_ARGS__, __VA_ARGS__); \
233     cerr << "\033[0m";
234 // #define debug(...) _dbg(#__VA_ARGS__, __VA_ARGS__);
235 #else
236 #define debug(...)
237 #endif
238 } // namespace Backlight
239
240 /*****          Backlight          *****/
241 * 一发入魂
242 * 仔细读题，注意边界条件
243 * 没有思路就试试逆向思维
244 * wdnmd! 我柜子动了不打了
245 * 能不能把我掉的分还给我
246 *****/
247
248 // mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
249 // int rnd(int l, int r) { return l + rng() % (r - l + 1); }
250
251 using namespace Backlight;
252 const int N = 2e7 + 5;
253 const int M = 5e5 + 5;
254 const int V = 5e7 + 5;
255 const ll MOD = 1e9 + 7; // 998244353 1e9+7
256 const int INF = 0x3f3f3f3f; // 1e9+7 0x3f3f3f3f
257 const ll LLINF = 0x3f3f3f3f3f3f3f3f; // 1e18+9 0x3f3f3f3f3f3f3f3f
258 const double eps = 1e-8;

```

```

259
260 void solve(int Case) { // printf("Case #%d: ", Case);
261 }
262
263 int main() {
264 #ifdef BACKLIGHT
265     freopen("in.txt", "r", stdin);
266     auto begin = std::chrono::steady_clock::now();
267 #endif
268
269     // ios::sync_with_stdio(false); cin.tie(0); cout.tie(0);
270     int T = 1;
271     // read(T);
272     for (int _ = 1; _ <= T; _++)
273         solve(_);
274
275 #ifdef BACKLIGHT
276     freopen("in.txt", "r", stdin);
277     auto end = std::chrono::steady_clock::now();
278     auto duration = std::chrono::duration_cast<std::chrono::milliseconds>(end - begin);
279     cerr << "\033[32mTime Elapsed: " << (double)(duration.count()) << " ms\033[0m" << endl;
280 #endif
281     return 0;
282 }

```

---

## 4.4 java-header

---

```

1 import java.io.*;
2 import java.util.*;
3 import java.math.*;
4
5 public class Main {
6     public static void main(String[] args) {
7         InputStream inputStream = System.in;
8         OutputStream outputStream = System.out;
9         InputReader in = new InputReader(inputStream);
10        PrintWriter out = new PrintWriter(outputStream);
11        Task solver = new Task();
12
13        int T = 1;
14        // T = in.nextInt();
15        for (int i = 1; i <= T; ++i)
16            solver.solve(i, in, out);
17
18        out.close();
19    }
20
21    static class Task {
22        public void solve(int testNumber, InputReader in, PrintWriter out) {
23            // write your solution here
24            out.println("Hello World");
25        }
26    }
27
28    static class InputReader {
29        public BufferedReader reader;
30        public StringTokenizer tokenizer;
31
32        public InputReader(InputStream stream) {
33            reader = new BufferedReader(new InputStreamReader(stream), 32768);
34            tokenizer = null;
35        }
36
37        public String next() {

```

```

38         while (tokenizer == null || !tokenizer.hasMoreTokens()) {
39             try {
40                 tokenizer = new StringTokenizer(reader.readLine());
41             } catch (IOException e) {
42                 throw new RuntimeException(e);
43             }
44         }
45         return tokenizer.nextToken();
46     }
47
48     public int nextInt() {
49         return Integer.parseInt(next());
50     }
51
52 }
53 }

```

---

## 4.5 SimulateAnneal

---

```

1 struct SimulateAnneal {
2     constexpr static double p = 0.996;
3     inline double Rand() { return 1.0 * rand() / RAND_MAX; }
4
5     int n;
6     vector<int> X, Y, W;
7     double ax, ay;
8
9     SimulateAnneal(int _n)
10         : n(_n), X(n), Y(n), W(n) {}
11
12     void input() {
13         for (int i = 0; i < n; ++i) {
14             read(X[i], Y[i], W[i]);
15         }
16     }
17
18     double cost(double x, double y) {
19         double res = 0;
20         for (int i = 0; i < n; ++i) {
21             double dx = X[i] - x;
22             double dy = Y[i] - y;
23             double d = sqrt(dx * dx + dy * dy);
24             res += d * W[i];
25         }
26         return res;
27     }
28
29     void init() {
30         ax = 0;
31         ay = 0;
32         for (int i = 0; i < n; ++i)
33             ax += X[i], ay += Y[i];
34         ax /= n;
35         ay /= n;
36     }
37
38     void simulate_anneal() {
39         srand(time(0));
40         double T = 1e6, TE = 1e-8;
41         double cx = ax, cy = ay, cc = cost(cx, cy);
42         while (T > TE) {
43             double nx = ax + (2 * Rand() - 1) * T;
44             double ny = ay + (2 * Rand() - 1) * T;
45

```

```

46     double nc = cost(nx, ny);
47     double d = nc - cc;
48
49     if (d < 0)
50         cc = nc, ax = cx = nx, ay = cy = ny;
51     else if (exp(-d / T) > Rand()) {
52         cx = nx;
53         cy = ny;
54     }
55
56     T *= p;
57 }
58 }
59
60 void work() {
61     init();
62     // try a try, AC is ok.
63     simulate_anneal();
64     simulate_anneal();
65     simulate_anneal();
66     simulate_anneal();
67 }
68 };

```

---

## 5 string

### 5.1 ACAM

```

1  namespace ACAM {
2  const int __N = 3e5 + 5;
3  const int __M = 26;
4  int tot, tr[__N][__M], fail[__N], last[__N];
5  int f[__N], e[__N];
6
7  int eid[__N];
8  multiset<int> st[__N];
9
10 inline int idx(const char& c) {
11     return c - 'a';
12 }
13
14 inline void init() {
15     tot = 0;
16     memset(tr[0], 0, sizeof(tr[0]));
17     f[0] = e[0] = 0;
18 }
19
20 inline int newnode() {
21     ++tot;
22     memset(tr[tot], 0, sizeof(tr[tot]));
23     f[tot] = e[tot] = 0;
24     return tot;
25 }
26
27 void insert(char* s, int n, int id) {
28     int p = 0, c;
29     for (int i = 0; i < n; ++i) {
30         c = idx(s[i]);
31         if (!tr[p][c])
32             tr[p][c] = newnode();
33         p = tr[p][c];
34         ++f[p];
35     }

```

```

36     ++e[p];
37
38     eid[id] = p;
39     st[p].insert(0);
40 }
41
42 // 字典图优化
43 // void getfail() {
44 //     queue<int> q;
45 //     for (int i = 0; i < __M; ++i) if (tr[0][i]) fail[tr[0][i]] = 0, q.push(tr[0][i]);
46 //     while(!q.empty()) {
47 //         int p = q.front(); q.pop();
48 //         for (int c = 0; c < __M; ++c) {
49 //             int nxt = tr[p][c];
50 //             if (nxt) fail[nxt] = tr[fail[p]][c], q.push(nxt);
51 //             else nxt = tr[fail[p]][c];
52 //         }
53 //     }
54 // }
55
56 // int query(char* t) {
57 //     int n = strlen(t), p = 0, res = 0;
58 //     for (int i = 0; i < n; ++i) {
59 //         p = tr[p][t[i] - 'a'];
60 //         for (int j = p; j && e[j] != -1; j = fail[j]) res += e[j], e[j] = -1;
61 //     }
62 //     return res;
63 // }
64
65 // 跳 fail 链
66 void getfail() {
67     queue<int> q;
68     fail[0] = 0;
69     for (int c = 0; c < __M; ++c)
70         if (tr[0][c])
71             fail[tr[0][c]] = last[tr[0][c]] = 0, q.push(tr[0][c]);
72     while (!q.empty()) {
73         int p = q.front();
74         q.pop();
75         for (int c = 0; c < __M; ++c) {
76             int u = tr[p][c];
77             if (u) {
78                 q.push(u);
79                 int v = fail[p];
80                 while (v && !tr[v][c])
81                     v = fail[v];
82                 fail[u] = tr[v][c];
83                 last[u] = e[fail[u]] ? fail[u] : last[fail[u]];
84             }
85         }
86     }
87 }
88
89 int queryMax(char* t, int n) {
90     int p = 0, res = -1, c;
91     for (int i = 0; i < n; ++i) {
92         c = idx(t[i]);
93         while (p && !tr[p][c])
94             p = fail[p];
95         p = tr[p][c];
96         for (int j = p; j; j = last[j])
97             if (e[j])
98                 updMax(res, (*st[j].rbegin()));
99     }
100     return res;

```

```
101 }  
102 } // namespace ACAM
```

---

## 5.2 GSAM

---

```
1 namespace GSAM {  
2 using T = char;  
3  
4 inline int idx(T c) {  
5     return c - 'a';  
6 }  
7  
8 const int __N = N << 1;  
9 const int __M = 26;  
10  
11 int tot, next[__N][__M];  
12 int len[__N], fail[__N];  
13  
14 inline void init() {  
15     tot = 0;  
16     fail[0] = -1;  
17     len[0] = 0;  
18     memset(next[0], 0, sizeof(next[0]));  
19 }  
20  
21 inline int newnode() {  
22     ++tot;  
23     fail[tot] = 0;  
24     len[tot] = 0;  
25     memset(next[tot], 0, sizeof(next[tot]));  
26     return tot;  
27 }  
28  
29 void insertTrie(const T* s, int n) {  
30     int p = 0, c;  
31     for (int i = 0; i < n; ++i) {  
32         c = idx(s[i]);  
33         if (!next[p][c])  
34             next[p][c] = newnode();  
35         p = next[p][c];  
36     }  
37 }  
38  
39 int extendSAM(int last, int c) {  
40     int cur = next[last][c];  
41     if (len[cur])  
42         return cur;  
43     len[cur] = len[last] + 1;  
44  
45     int p = fail[last];  
46     while (p != -1) {  
47         if (!next[p][c])  
48             next[p][c] = cur;  
49         else  
50             break;  
51         p = fail[p];  
52     }  
53  
54     if (p == -1) {  
55         fail[cur] = 0;  
56         return cur;  
57     }  
58  
59     int q = next[p][c];
```

```

60  if (len[p] + 1 == len[q]) {
61      fail[cur] = q;
62      return cur;
63  }
64
65  int clone = newnode();
66  for (int i = 0; i < __M; ++i)
67      next[clone][i] = len[next[q][i]] ? next[q][i] : 0;
68
69  len[clone] = len[p] + 1;
70  while (p != -1 && next[p][c] == q) {
71      next[p][c] = clone;
72      p = fail[p];
73  }
74  fail[clone] = fail[q];
75  fail[cur] = clone;
76  fail[q] = clone;
77  return cur;
78 }
79
80 void build() {
81     queue<pair<int, int>> q;
82     for (int i = 0; i < __M; ++i)
83         if (next[0][i])
84             q.push(make_pair(0, i));
85
86     while (!q.empty()) {
87         pair<int, int> u = q.front();
88         q.pop();
89         int last = extendSAM(u.first, u.second);
90         for (int i = 0; i < __M; ++i)
91             if (next[last][i])
92                 q.push(make_pair(last, i));
93     }
94 }
95
96 // 多模式串--本质不同子串数
97 ll count() {
98     ll res = 0;
99     for (int i = 1; i <= tot; ++i)
100         res += len[i] - len[fail[i]];
101     return res;
102 }
103 } // namespace GSAM

```

## 5.3 KMP

```

1  namespace KMP {
2  // pi_i = s[0...i] 最长 border
3  void getPi(char* s, int n, int* pi) {
4      pi[0] = 0;
5      for (int i = 1; i < n; ++i) {
6          int j = pi[i - 1];
7          while (j > 0 && s[j] != s[i])
8              j = pi[j - 1];
9          if (s[i] == s[j])
10             ++j;
11         pi[i] = j;
12     }
13 }
14
15 vector<int> getAllMatchPosition(char* s, int n, int* pi, char* t, int m) {
16     s[n] = '#';
17     s[n + 1] = 0;

```

```

18 ++n;
19 KMP::getPi(s, n, pi);
20
21 vector<int> ans;
22
23 int p = 0;
24 for (int i = 0; i < m; ++i) {
25     while (p > 0 && t[i] != s[p])
26         p = pi[p - 1];
27     if (t[i] == s[p]) {
28         ++p;
29         if (p == n - 1) {
30             ans.push_back(i + 2 - n);
31         }
32     }
33 }
34
35 return ans;
36 }
37
38 int getPeriod(int n, int* pi) {
39     return n - pi[n - 1];
40 }
41 } // namespace KMP

```

## 5.4 Manacher

```

1 namespace Manacher {
2 // 1-based
3
4 const int __N = N << 1;
5
6 char s[__N];
7 int n, len[__N];
8
9 // @ t1 t2 t3 \0
10 // ==> @ # t1 # t2 # t3 # \0
11 inline void init(char* t, int m) {
12     n = 2 * m + 1;
13     s[0] = '@';
14     s[n] = '#';
15     s[n + 1] = 0;
16     for (int i = 1; i <= m; ++i) {
17         s[2 * i - 1] = '#';
18         s[2 * i] = t[i];
19     }
20 }
21
22 // s[i-len[i]...i+len[i]] is palindromic
23 // len[i]-1 is palindromic length in t
24 void manacher(char* t, int m) {
25     init(t, m);
26     for (int i = 1, l = 0, r = 0, k; i <= n; ++i) {
27         k = i > r ? 1 : min(r - i, len[l + r - i]);
28         while (s[i - k] == s[i + k])
29             ++k;
30         len[i] = k--;
31         if (i + k > r) {
32             l = i - k;
33             r = i + k;
34         }
35     }
36 }
37

```



```

38 int getMaxPalindromicLength(char* t, int m) {
39     manacher(t, m);
40     int ma = 0;
41     for (int i = 1; i <= n; ++i)
42         updMax(ma, len[i]);
43     return ma - 1;
44 }
45 } // namespace Manacher

```

## 5.5 PAM

```

1 //最长双倍回文串长度
2 #include <bits/stdc++.h>
3 using namespace std;
4
5 typedef long long ll;
6 const int N = 5e5 + 5;
7
8 struct Palindromic_Automaton {
9     //0 偶根 1 奇根 range[2-tot]
10    int s[N << 1], now;
11    int next[N << 1][26], fail[N << 1], len[N << 1], last, tot;
12    int cnt[N << 1]; //状态 i 表示的回文串数目
13
14    // extend
15    int trans[N << 1];
16
17    void init() {
18        s[0] = len[1] = -1;
19        fail[0] = tot = now = 1;
20        last = len[0] = 0;
21        memset(next[0], 0, sizeof(next[0]));
22        memset(next[1], 0, sizeof(next[1]));
23    }
24    int newnode() {
25        tot++;
26        memset(next[tot], 0, sizeof(next[tot]));
27        fail[tot] = cnt[tot] = len[tot] = 0;
28        return tot;
29    }
30    int getfail(int x) {
31        while (s[now - len[x] - 2] != s[now - 1])
32            x = fail[x];
33        return x;
34    }
35    void extend(int c) {
36        s[now++] = c;
37        int cur = getfail(last);
38        if (!next[cur][c]) {
39            int p = newnode();
40            len[p] = len[cur] + 2;
41            fail[p] = next[getfail(fail[cur])][c];
42            next[cur][c] = p;
43
44            // extend
45            if (len[p] <= 2)
46                trans[p] = fail[p];
47            else {
48                int tmp = trans[cur];
49                while (s[now - len[tmp] - 2] != s[now - 1] || (len[tmp] + 2) * 2 > len[p])
50                    tmp = fail[tmp];
51                trans[p] = next[tmp][c];
52            }
53        }

```

```

54     last = next[cur][c];
55     cnt[last]++;
56 }
57 int count() { return tot - 1; }
58 void calc() {
59     for (int i = tot; i >= 2; --i)
60         cnt[fail[i]] += cnt[i];
61     cnt[0] = cnt[1] = 0;
62 }
63 int getans() {
64     int ans = 0;
65     for (int i = 2; i <= tot; i++) {
66         if (len[i] > ans && len[trans[i]] * 2 == len[i] && len[trans[i]] % 2 == 0)
67             ans = len[i];
68     }
69     return ans;
70 }
71 } pam;
72
73 char t[N];
74
75 int main() {
76     int n;
77     scanf("%d", &n);
78     scanf("%s", t);
79     pam.init();
80     for (int i = 0; i < n; ++i) {
81         pam.extend(t[i] - 'a');
82     }
83     printf("%d\n", pam.getans());
84     return 0;
85 }

```

## 5.6 SA

```

1 namespace SA {
2     // 0 based, 倍增法构建, O(nlogn)
3     int height[N], c[N], x[N], y[N], sa[N], rk[N];
4     void build_sa(int* s, int n) {
5         n++;
6         int i, j, k, m = 256; // m 为字符集大小, max(s[i]) < m
7         for (i = 0; i < m; i++)
8             c[i] = 0;
9         for (i = 0; i < n; i++)
10             c[x[i] = s[i]]++;
11         for (i = 1; i < m; i++)
12             c[i] += c[i - 1];
13         for (i = n - 1; i >= 0; i--)
14             sa[--c[x[i]]] = i;
15         for (j = 1; j <= n; j <= 1) {
16             k = 0;
17             for (i = n - j; i < n; i++)
18                 y[k++] = i;
19             for (i = 0; i < n; i++)
20                 if (sa[i] >= j)
21                     y[k++] = sa[i] - j;
22             for (i = 0; i < m; i++)
23                 c[i] = 0;
24             for (i = 0; i < n; i++)
25                 c[x[y[i]]]++;
26             for (i = 1; i < m; i++)
27                 c[i] += c[i - 1];
28             for (i = n - 1; i >= 0; i--)
29                 sa[--c[x[y[i]]]] = y[i];

```

```

30     swap(x, y);
31     m = 0;
32     x[sa[0]] = m++;
33     for (i = 1; i < n; i++) {
34         if (y[sa[i]] == y[sa[i - 1]] && y[sa[i] + j] == y[sa[i - 1] + j])
35             x[sa[i]] = m - 1;
36         else
37             x[sa[i]] = m++;
38     }
39     if (m >= n)
40         break;
41 }
42 k = 0;
43 for (i = 0; i < n; i++)
44     rk[sa[i]] = i;
45 for (i = 0; i < n - 1; i++) {
46     if (k)
47         k--;
48     j = sa[rk[i] - 1];
49     while (s[i + k] == s[j + k])
50         k++;
51     height[rk[i]] = k;
52 }
53 }
54 } // namespace SA

```

## 5.7 SAIS

```

1 namespace SAIS {
2 // 1 based, O(n)
3 int s[N << 1], t[N << 1], height[N], sa[N], rk[N], p[N], c[N], w[N];
4 inline int trans(int n, int* S) {
5     int m = *max_element(S + 1, S + 1 + n);
6     for (int i = 1; i <= n; ++i)
7         rk[S[i]] = 1;
8     for (int i = 1; i <= m; ++i)
9         rk[i] += rk[i - 1];
10    for (int i = 1; i <= n; ++i)
11        s[i] = rk[S[i]];
12    return rk[m];
13 }
14 #define ps(x) sa[w[s[x]]--] = x
15 #define pl(x) sa[w[s[x]]++] = x
16 inline void radix(int* v, int* s, int* t, int n, int m, int n1) {
17     memset(sa, 0, n + 1 << 2);
18     memset(c, 0, m + 1 << 2);
19     for (int i = 1; i <= n; ++i)
20         ++c[s[i]];
21     for (int i = 1; i <= m; ++i)
22         w[i] = c[i] += c[i - 1];
23     for (int i = n1; i; --i)
24         ps(v[i]);
25     for (int i = 1; i <= m; ++i)
26         w[i] = c[i - 1] + 1;
27     for (int i = 1; i <= n; ++i)
28         if (sa[i] > 1 && t[sa[i] - 1])
29             pl(sa[i] - 1);
30     for (int i = 1; i <= m; ++i)
31         w[i] = c[i];
32     for (int i = n; i; --i)
33         if (sa[i] > 1 && !t[sa[i] - 1])
34             ps(sa[i] - 1);
35 }
36 inline void SAIS(int n, int m, int* s, int* t, int* p) {

```

```

37  int n1 = 0, ch = rk[1] = 0, *s1 = s + n;
38  t[n] = 0;
39  for (int i = n - 1; i; --i)
40      t[i] = s[i] == s[i + 1] ? t[i + 1] : s[i] > s[i + 1];
41  for (int i = 2; i <= n; ++i)
42      rk[i] = t[i - 1] && !t[i] ? (p[++n1] = i, n1) : 0;
43  radix(p, s, t, n, m, n1);
44  for (int i = 1, x, y; i <= n; ++i)
45      if (x = rk[sa[i]]) {
46          if (ch <= 1 || p[x + 1] - p[x] != p[y + 1] - p[y])
47              ++ch;
48          else
49              for (int j = p[x], k = p[y]; j <= p[x + 1]; ++j, ++k)
50                  if ((s[j] << 1 | t[j]) ^ (s[k] << 1 | t[k])) {
51                      ++ch;
52                      break;
53                  }
54          s1[y = x] = ch;
55      }
56  if (ch < n1)
57      SAIS(n1, ch, s1, t + n, p + n1);
58  else
59      for (int i = 1; i <= n1; ++i)
60          sa[s1[i]] = i;
61  for (int i = 1; i <= n1; ++i)
62      s1[i] = p[sa[i]];
63  radix(s1, s, t, n, m, n1);
64 }
65 inline void build_sa(int* S, int n) {
66     int m = trans(++n, S);
67     SAIS(n, m, s, t, p);
68     for (int i = 1; i < n; ++i)
69         rk[sa[i] = sa[i + 1]] = i;
70     for (int i = 1, j, k = 0; i < n; ++i)
71         if (rk[i] > 1) {
72             for (j = sa[rk[i] - 1]; S[i + k] == S[j + k]; ++k)
73                 ;
74             if (height[rk[i]] = k)
75                 --k;
76         }
77 }
78 } // namespace SAIS

```

## 5.8 SAM

```

1  //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)
2  #include <bits/stdc++.h>
3  using namespace std;
4
5  typedef long long ll;
6  const int maxn = 1e6 + 5;
7
8  char s[maxn];
9  struct Suffix_Automaton {
10     //初始状态为 0, range[0...tot-1]
11     struct state {
12         int len, link;
13         map<char, int> next;
14     } st[maxn << 1];
15     int last, tot;
16
17     void init() {
18         st[0].len = 0;
19         st[0].link = -1;

```

```

20     tot++;
21     last = 0;
22 }
23
24 void extend(char c) {
25     int cur = tot++;
26     st[cur].len = st[last].len + 1;
27     int p = last;
28     while (p != -1 && !st[p].next.count(c)) {
29         st[p].next[c] = cur;
30         p = st[p].link;
31     }
32     if (p == -1)
33         st[cur].link = 0;
34     else {
35         int q = st[p].next[c];
36         if (st[p].len + 1 == st[q].len)
37             st[cur].link = q;
38         else {
39             int clone = tot++;
40             st[clone].len = st[p].len + 1;
41             st[clone].next = st[q].next;
42             st[clone].link = st[q].link;
43             while (p != -1 && st[p].next[c] == q) {
44                 st[p].next[c] = clone;
45                 p = st[p].link;
46             }
47             st[q].link = st[cur].link = clone;
48         }
49     }
50     last = cur;
51 }
52
53 ll count() {
54     ll res = 0;
55     for (int i = 0; i < tot; i++)
56         res += st[i].len - st[st[i].link].len;
57     return res;
58 }
59 } sam;
60
61 int main() {
62     scanf("%s", s);
63     sam.init();
64     for (int i = 0; s[i] != 0; i++)
65         sam.extend(s[i]);
66     printf("%lld\n", sam.count());
67     return 0;
68 }

```

## 5.9 SqAM

```

1  /**
2   * 识别一个串的子序列,  $O(n^2)$ 
3   * 用法类似后缀自动机
4   */
5  struct SqAM {
6      int next[N << 1][26], pre[N << 1], lst[26];
7      int root, tot;
8      void init() {
9          root = tot = 1;
10         for (int i = 0; i < 26; i++)
11             lst[i] = 1;
12     }

```

```

13
14 void extend(int c) {
15     int p = lst[c], np = ++tot;
16     pre[np] = p;
17     for (int i = 0; i < 26; i++)
18         for (int j = lst[i]; j && !next[j][c]; j = pre[j])
19             next[j][c] = np;
20     lst[c] = np;
21 }
22 };

```

---

## 5.10 string-hash

```

1 namespace Hash {
2 // 1 based, double hash
3 typedef long long ll;
4 const ll P1 = 29;
5 const ll P2 = 131;
6 const ll MOD1 = 1e9 + 7;
7 const ll MOD2 = 1e9 + 9;
8 ll p1[N], p2[N], h1[N], h2[N];
9 void init_hash(char* s, int n) {
10     p1[0] = p2[0] = 1;
11     for (int i = 1; i <= n; i++)
12         p1[i] = (p1[i - 1] * P1) % MOD1;
13     for (int i = 1; i <= n; i++)
14         p2[i] = (p2[i - 1] * P2) % MOD2;
15     for (int i = 1; i <= n; i++)
16         h1[i] = (h1[i - 1] * P1 + s[i]) % MOD1;
17     for (int i = 1; i <= n; i++)
18         h2[i] = (h2[i - 1] * P2 + s[i]) % MOD2;
19 }
20
21 ll get_hash(int l, int r) {
22     ll H1 = ((h1[r] - h1[l - 1] * p1[r - l + 1]) % MOD1 + MOD1) % MOD1;
23     ll H2 = ((h2[r] - h2[l - 1] * p2[r - l + 1]) % MOD2 + MOD2) % MOD2;
24     return H1 * MOD2 + H2;
25 }
26 } // namespace Hash

```

---

## 5.11 SuffixBST

```

1 /**
2  * 1. 在当前字符串的后面插入字符
3  * 2. 在当前字符串的后面删除字符
4  * 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
5  */
6 #include <bits/stdc++.h>
7 using namespace std;
8
9 const int N = 8e5 + 5;
10 const double INF = 1e18;
11
12 void decode(char* s, int len, int mask) {
13     for (int i = 0; i < len; ++i) {
14         mask = (mask * 131 + i) % len;
15         swap(s[i], s[mask]);
16     }
17 }
18
19 int q, n, na;
20 char a[N], t[N];

```

```
21
22 // SuffixBST(SGT Ver)
23
24 // 顺序加入，查询时将询问串翻转
25 // 以 i 结束的前缀，对应节点的编号为 i
26 // 注意：不能写懒惰删除，否则可能会破坏树的结构
27 const double alpha = 0.75;
28 int root;
29 int sz[N], L[N], R[N];
30 double tag[N];
31 int buffer_size, buffer[N];
32
33 bool cmp(int x, int y) {
34     if (t[x] != t[y])
35         return t[x] < t[y];
36     return tag[x - 1] < tag[y - 1];
37 }
38
39 void init() {
40     root = 0;
41 }
42
43 void new_node(int& rt, int p, double lv, double rv) {
44     rt = p;
45     sz[rt] = 1;
46     tag[rt] = (lv + rv) / 2;
47     L[rt] = R[rt] = 0;
48 }
49
50 void push_up(int x) {
51     if (!x)
52         return;
53     sz[x] = sz[L[x]] + 1 + sz[R[x]];
54 }
55
56 bool balance(int rt) {
57     return alpha * sz[rt] > max(sz[L[rt]], sz[R[rt]]);
58 }
59
60 void flatten(int rt) {
61     if (!rt)
62         return;
63     flatten(L[rt]);
64     buffer[++buffer_size] = rt;
65     flatten(R[rt]);
66 }
67
68 void build(int& rt, int l, int r, double lv, double rv) {
69     if (l > r) {
70         rt = 0;
71         return;
72     }
73     int mid = (l + r) >> 1;
74     double mv = (lv + rv) / 2;
75
76     rt = buffer[mid];
77     tag[rt] = mv;
78     build(L[rt], l, mid - 1, lv, mv);
79     build(R[rt], mid + 1, r, mv, rv);
80     push_up(rt);
81 }
82
83 void rebuild(int& rt, double lv, double rv) {
84     buffer_size = 0;
85     flatten(rt);
```

```
86     build(rt, 1, buffer_size, lv, rv);
87 }
88
89 void insert(int& rt, int p, double lv, double rv) {
90     if (!rt) {
91         new_node(rt, p, lv, rv);
92         return;
93     }
94
95     if (cmp(p, rt))
96         insert(L[rt], p, lv, tag[rt]);
97     else
98         insert(R[rt], p, tag[rt], rv);
99
100     push_up(rt);
101     if (!balance(rt))
102         rebuild(rt, lv, rv);
103 }
104
105 void remove(int& rt, int p, double lv, double rv) {
106     if (!rt)
107         return;
108
109     if (rt == p) {
110         if (!L[rt] || !R[rt]) {
111             rt = (L[rt] | R[rt]);
112         } else {
113             // 找到 rt 的前驱来替换 rt
114             int nrt = L[rt], fa = rt;
115             while (R[nrt]) {
116                 fa = nrt;
117                 sz[fa]--;
118                 nrt = R[nrt];
119             }
120             if (fa == rt) {
121                 R[nrt] = R[rt];
122             } else {
123                 L[nrt] = L[rt];
124                 R[nrt] = R[rt];
125                 R[fa] = 0;
126             }
127             rt = nrt;
128             tag[rt] = (lv + rv) / 2;
129         }
130     } else {
131         double mv = (lv + rv) / 2;
132         if (cmp(p, rt))
133             remove(L[rt], p, lv, mv);
134         else
135             remove(R[rt], p, mv, rv);
136     }
137
138     push_up(rt);
139     if (!balance(rt))
140         rebuild(rt, lv, rv);
141 }
142
143 bool cmp1(char* s, int len, int p) {
144     for (int i = 1; i <= len; ++i, --p) {
145         if (s[i] < t[p])
146             return true;
147         if (s[i] > t[p])
148             return false;
149     }
150 }
```



```

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;

```

```

216     for (int i = 1; i <= T; ++i)
217         solve(i);
218     return 0;
219 }

```

---

## 5.12 Trie

```

1  namespace Trie {
2  // 1-based
3  const int __N = 4e6 + 5;
4  const int __M = 26;
5  int tot;
6  int ch[__N][__M];
7  int f[__N], e[__N];
8
9  inline void init() {
10     tot = 0;
11     memset(ch[0], 0, sizeof(ch[0]));
12     f[0] = e[0] = 0;
13 }
14
15 inline int newnode() {
16     ++tot;
17     memset(ch[tot], 0, sizeof(ch[tot]));
18     f[tot] = e[tot] = 0;
19     return tot;
20 }
21
22 inline int idx(char c) {
23     return c - 'a';
24 }
25
26 void insert(char* s) {
27     int n = strlen(s + 1), p = 0, c;
28     for (int i = 1; i <= n; ++i) {
29         c = idx(s[i]);
30         if (!ch[p][c])
31             ch[p][c] = newnode();
32         p = ch[p][c];
33         ++f[p];
34     }
35     ++e[p];
36 }
37
38 int query(char* s) {
39     int p = 0, n = strlen(s + 1), c;
40     for (int i = 1; i <= n; ++i) {
41         c = idx(s[i]);
42         if (!ch[p][c])
43             return 0;
44         p = ch[p][c];
45     }
46     return e[p];
47 }
48 } // namespace Trie

```

---

## 5.13 ZAlgorithm

```

1  namespace ZAlgorithm {
2  // 1-based
3
4  // z_i = LCP(s, s[i..n])

```

```
5 void getZ(char* s, int n, int* z) {
6     z[1] = n;
7     for (int i = 2, l = 0, r = 0; i <= n; ++i) {
8         if (i <= r)
9             z[i] = min(r - i + 1, z[i - l + 1]);
10        else
11            z[i] = 0;
12        while (i + z[i] <= n && s[z[i] + 1] == s[i + z[i]])
13            ++z[i];
14        if (i + z[i] - 1 > r)
15            l = i, r = i + z[i] - 1;
16    }
17 }
18
19 // p_i = LCP(s, t[i..m])
20 void EXKMP(char* s, int n, int* z, char* t, int m, int* p) {
21     getZ(s, n, z);
22     for (int i = 1, l = 0, r = 0; i <= m; ++i) {
23         if (i <= r)
24             p[i] = min(r - i + 1, z[i - l + 1]);
25         else
26             p[i] = 0;
27         while (i + p[i] <= m && s[p[i] + 1] == t[i + p[i]])
28             ++p[i];
29         if (i + p[i] - 1 > r)
30             l = i, r = i + p[i] - 1;
31     }
32 }
33 } // namespace ZAlgorithm
```

---