

# Backlight's Code Template

Backlight @ CSU

2021 年 5 月 8 日

# 目录

<b>1</b>	<b>ds</b>	<b>1</b>
1.1	AVLTree	1
1.2	BTree	5
1.3	CaptainMo	10
1.4	FenwickTree	11
1.5	LCT	11
1.6	LeftstTree	14
1.7	PersistentSegmentTree	15
1.8	rbtree-1	16
1.9	RBTree	25
1.10	RMQ	30
1.11	RollBackCaptainMo	31
1.12	SegmentTree	32
1.13	SGTree	36
1.14	Splay	39
1.15	Treap-dynamic	41
1.16	Treap-pointer	44
1.17	Treap	46
<b>2</b>	<b>graph</b>	<b>48</b>
2.1	BCC-Edge	48
2.2	BCC-Point	50
2.3	BiGraphMatch	51
2.4	BiWrappMatch	53
2.5	BlockForest	54
2.6	BlockTree	56
2.7	Dijkstra	59
2.8	dsu-on-tree	60
2.9	FullyDCP	61
2.10	Graph	74
2.11	GraphMatch	75
2.12	HLD-Edge	78
2.13	Kosaraju	83
2.14	Kruskal	84
2.15	LCA-HLD	85
2.16	LCA	85
2.17	maxflow	87
2.18	mincostflow	88
2.19	SCC	90
2.20	SPFA	91
2.21	tree-divide	92
2.22	Wrapp	93
2.23	WrappMatch	94
<b>3</b>	<b>math</b>	<b>111</b>
3.1	2DGeometry	111
3.2	3DGeometry	123
3.3	BSGS	129
3.4	Cipolla	130
3.5	Combination	131
3.6	CRT	131
3.7	EulerSeive	132
3.8	eval	132
3.9	EXGCD	134
3.10	FFT	134
3.11	LinearBasis	136
3.12	Lucas	137
3.13	Mint	138
3.14	Mobius	141
3.15	Modular	142
3.16	NTT	142
3.17	PollardRho	143
3.18	poly-struct	146
3.19	Poly	151

3.20	Simplex	157
3.21	SimpsonIntegral	159
<b>4</b>	<b>other</b>	<b>159</b>
4.1	BFPRT	159
4.2	cpp-header	160
4.3	debug	165
4.4	java-header	170
4.5	SimulateAnneal	170
<b>5</b>	<b>string</b>	<b>172</b>
5.1	ACAM	172
5.2	GSAM	173
5.3	KMP	175
5.4	Manacher	175
5.5	PAM	176
5.6	SA	177
5.7	SAIS	178
5.8	SAM	179
5.9	SqAM	180
5.10	string-hash	181
5.11	Trie	181
5.12	ZAlgorithm	182

# 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         node(T _v) : v(_v)
19         {
20             sz = h = cnt = 1;
21             l = r = nullptr;
22         }
23     };
24     node *root = nullptr;
25     int get_size(node* p)
26     {
27         return p ? p->sz : 0;
28     }
29
30     int get_height(node* p)
31     {
32         return p ? p->h : 0;
33     }
34
35     void push_up(node *p)
36     {
37         if (!p) return;
38         p->sz = get_size(p->l) + p->cnt + get_size(p->r);
39         p->h = 1 + max(get_height(p->l), get_height(p->r));
40     }
41
42     void zig(node* &p)
43     {
44         node* q = p;
45         q = p->r;
46         p->r = q->l;
47         q->l = p;
48         push_up(p);
49         push_up(q);
50         p = q;
51     }
52
53     void zag(node* &p)
54     {
55         node* q = p->l;
56         p->l = q->r;
57         q->r = p;
58         push_up(p);
59         push_up(q);
60         p = q;
61     }

```

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

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

```
192     }
193 }
194 return res;
195 }
196
197 T getSucc(T v)
198 {
199     T res = numeric_limits<T>::max();
200     node* p = root;
201     while(p) {
202         if (v == p->v) {
203             node *q = p->r;
204             while(q->l) q = q->l;
205             res = q->v;
206             break;
207         }
208
209         if (v < p->v) {
210             if (p->v < res) res = p->v;
211             p = p->l;
212         } else {
213             p = p->r;
214         }
215     }
216     return res;
217 }
218
219 void debug(node* p)
220 {
221     if (!p) return;
222     debug(p->l);
223     cerr << p->v << " ";
224     debug(p->r);
225 }
226
227 void debug()
228 {
229     cerr << "INORDER: " << endl;
230     debug(root);
231     cerr << endl;
232 }
233 };
234
235 void solve(int Case)
236 {
237     int n;
238     scanf("%d", &n);
239     int op, x;
240     AVLTree<int> t;
241     FOR(i, 1, n) {
242         scanf("%d %d", &op, &x);
243         // cerr << op << " " << x << endl;
244         switch(op) {
245             case 1:
246                 t.ins(x);
247                 break;
248             case 2:
249                 t.del(x);
250                 break;
251             case 3:
252                 printf("%d\n", t.getRank(x));
253                 break;
254             case 4:
255                 printf("%d\n", t.getKth(x));
256                 break;
```

```

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

```

## 1.2 BTree

```

1  template <typename K, int BF>
2  class BTree
3  {
4      public:
5          typedef std::pair<K, int> value_type;
6
7      private:
8          struct Node
9          {
10             value_type values[2 * BF - 1];
11             Node *child[2 * BF] = {nullptr};
12             Node *p = nullptr;
13             int keyNum = 0, size = 0;
14             bool isLeaf = true;
15             const K &key(int i) const { return values[i].first; }
16             int &cnt(int i) { return values[i].second; }
17             Node(Node *p = nullptr) : p(p) {}
18         };
19         Node *root = nullptr;
20         static bool pairComp(const value_type &lhs, const K &rhs) { return lhs.first < rhs; }
21         template <typename T>
22         static void shiftBy(T *ptr, int length, int shift) { memmove(ptr + shift, ptr, length * sizeof(T)); }
23         static int calcSize(Node *x)
24         {
25             if (!x)
26                 return 0;
27             int nsz = 0;
28             for (int i = 0; i < x->keyNum; ++i)
29                 nsz += getSize(x->child[i]) + x->cnt(i);
30             nsz += getSize(x->child[x->keyNum]);
31             return nsz;
32         }
33         static int getSize(Node *x)
34         {
35             if (!x)
36                 return 0;
37             return x->size;
38         }
39         //把 where 孩子分成两个节点，都作为 x 的孩子
40         void split(Node *x, int where)

```



```

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

```

```

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

```

```

171         root = x->child[i], root->p = nullptr;
172         erase(x->child[i], key);
173     }
174 }
175 }
176 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
177 {
178     if (x->child[i]->keyNum == BF - 1)
179     {
180         Node *y = x->child[i];
181         if (i >= 1 && x->child[i - 1]->keyNum >= BF) //左兄弟, 取走它的最大孩子
182         {
183             //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
184             //但是从左还是右借并不完全一样, 所以不能一概处理
185             Node *z = x->child[i - 1];
186             shiftBy(y->values, y->keyNum, 1);
187             //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
188             //所以, value 的构造必须要用 new 不能用 =, 从而避开 = 的资源释放
189             //但是 value 的移动似乎应该是 bitwise 的, 考虑 std::move
190             new (y->values) value_type(std::move(x->values[i - 1]));
191             new (x->values + i - 1) value_type(std::move(z->values[z->keyNum - 1]));
192             if (!y->isLeaf)
193             {
194                 shiftBy(y->child, y->keyNum + 1, 1);
195                 y->child[0] = z->child[z->keyNum], y->child[0]->p = y;
196             }
197
198             --z->keyNum, ++y->keyNum;
199             y->size = calcSize(y), z->size = calcSize(z);
200             erase(y, key);
201         }
202         else if (i < x->keyNum && x->child[i + 1]->keyNum >= BF) //右兄弟, 取走它的最小孩子
203         {
204             Node *z = x->child[i + 1];
205             new (y->values + y->keyNum) value_type(std::move(x->values[i]));
206             new (x->values + i) value_type(std::move(z->values[0]));
207             if (!y->isLeaf) //y 和 z 深度一样, isLeaf 情况相同
208             {
209                 y->child[y->keyNum + 1] = z->child[0], y->child[y->keyNum + 1]->p = y;
210                 shiftBy(z->child + 1, z->keyNum, -1);
211             }
212             shiftBy(z->values + 1, z->keyNum - 1, -1);
213
214             --z->keyNum, ++y->keyNum;
215             y->size = calcSize(y), z->size = calcSize(z);
216             erase(y, key);
217         }
218         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
219         {
220             if (i != 0)
221                 --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
222             y = x->child[i];
223             merge(x, i);
224             if (root->keyNum == 0)
225                 root = y, root->p = nullptr;
226             erase(y, key);
227         }
228     }
229     else
230         erase(x->child[i], key);
231 }
232 }
233
234 public:
235     BTree() : root(new Node) {}

```

```

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

```

```

301     }
302 }
303 return ret;
304 }
305 int kth(int k)
306 {
307     Node *x = root;
308     while (true)
309     {
310         for (int i = 0; i <= x->keyNum; ++i)
311         {
312             //const int csz = getSize(x->child[i]) + (i == x->keyNum ? 1 : x->cnt(i));
313             const int lb = getSize(x->child[i]) + 1, ub = getSize(x->child[i]) + (i == x->keyNum ? 1 : x->cnt(i));
314             if (k >= lb && k <= ub)
315                 return x->key(i);
316             if (k < lb)
317             {
318                 x = x->child[i];
319                 break;
320             }
321             k -= ub;
322         }
323     }
324 }
325 };

```

### 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) : l(_l), r(_r), id(_id) {}
9     bool operator < (const Query& q) const {
10         if (block_id[l] == block_id[q.l])
11             return block_id[l] & 1 ? r < q.r : r > q.r;
12         return block_id[l] < block_id[q.l];
13     }
14 } Q[N];
15
16 int n, q, a[N];
17
18 int cnt[N], ge2;
19 inline void add(int p) {
20     ++cnt[a[p]];
21     if (cnt[a[p]] == 2) ++ge2;
22 }
23
24 inline void del(int p) {
25     if (cnt[a[p]] == 2) --ge2;
26     --cnt[a[p]];
27 }
28
29 void CaptainMo() {
30     block_sz = sqrt(n);
31     for (int i = 1; i <= n; ++i) block_id[i] = i / block_sz;
32     sort(Q + 1, Q + 1 + q);
33
34     int l = 1, r = 0;
35     ge2 = 0;
36     for (int i = 1; i <= q; ++i) {

```

```

37     while(r < Q[i].r) ++r, add(r);
38     while(l < Q[i].l) del(l), ++l;
39     while(l > Q[i].l) --l, add(l);
40     while(r > Q[i].r) del(r), --r;
41     ans[Q[i].id] = (ge2 == 0);
42 }
43 }

```

## 1.4 FenwickTree

```

1  template<typename T>
2  struct FenwickTree {
3      int n;
4      vector<T> c;
5      FenwickTree(int _n) : n(_n), c(n + 1) {}
6      inline int lb(int x) { return x & -x; }
7      void add(int x, T d) { for (; x < n; x += lb(x)) c[x] += d; }
8      T getsum(int x) { T r = 0; for (; x; x -= lb(x)) r += c[x]; return r; }
9      T getsum(int l, int r) { return getsum(r) - getsum(l - 1); }
10     T kth(int k) {
11         T ans = 0, cnt = 0;
12         for (int i = log2(n) + 1; i >= 0; --i) {
13             ans += (1LL << i);
14             if (ans >= n || cnt + c[ans] >= k) ans -= (1LL << i);
15             else cnt += c[ans];
16         }
17         return ans + 1;
18     }
19 };

```

## 1.5 LCT

```

1  template <typename T>
2  struct LinkCutTree
3  {
4      #define ls ch[x][0]
5      #define rs ch[x][1]
6      #define SIZE 100005
7
8      int tot, sz[SIZE], rev[SIZE], ch[SIZE][2], fa[SIZE];
9      T v[SIZE], sum[SIZE];
10
11     LinkCutTree() { tot = 0; }
12
13     inline void init() { tot = 0; }
14
15     inline void clear(int x)
16     {
17         ch[x][0] = ch[x][1] = fa[x] = sz[x] = rev[x] = sum[x] = v[x] = 0;
18     }
19
20     inline int get(int x) { return ch[fa[x]][1] == x; }
21
22     inline int isroot(int x) { return ch[fa[x]][0] != x && ch[fa[x]][1] != x; }
23
24     inline int newnode(T val)
25     {
26         ++tot;
27         sz[tot] = 1;
28         ch[tot][0] = ch[tot][1] = fa[tot] = rev[tot] = 0;
29         sum[tot] = v[tot] = val;
30         return tot;

```

```
31     }
32
33     inline void reverse(int x)
34     {
35         swap(ls, rs);
36         rev[x] ^= 1;
37     }
38
39     inline void push_up(int x)
40     {
41         sz[x] = sz[ls] + 1 + sz[rs];
42         sum[x] = sum[ls] ^ v[x] ^ sum[rs];
43     }
44
45     inline void push_down(int x)
46     {
47         if (rev[x])
48         {
49             reverse(ls);
50             reverse(rs);
51             rev[x] = 0;
52         }
53     }
54
55     inline void update(int x)
56     {
57         if (!isroot(x))
58             update(fa[x]);
59         push_down(x);
60     }
61
62     inline void rotate(int x)
63     {
64         int f = fa[x], g = fa[f], i = get(x);
65         if (!isroot(f))
66             ch[g][get(f)] = x;
67         fa[x] = g;
68         ch[f][i] = ch[x][i ^ 1];
69         fa[ch[f][i]] = f;
70         ch[x][i ^ 1] = f;
71         fa[f] = x;
72         push_up(f);
73         push_up(x);
74     }
75
76     inline void splay(int x)
77     {
78         update(x);
79         for (; !isroot(x); rotate(x))
80             if (!isroot(fa[x]))
81                 rotate(get(fa[x]) == get(x) ? fa[x] : x);
82     }
83
84     inline void access(int x)
85     {
86         for (int y = 0; x; y = x, x = fa[x]) splay(x), rs = y, push_up(x);
87     }
88
89     inline void makeroot(int x)
90     {
91         access(x);
92         splay(x);
93         reverse(x);
94     }
95
```

```

96     inline int findroot(int x)
97     {
98         access(x);
99         splay(x);
100         while (ls) push_down(x), x = ls;
101         return x;
102     }
103
104     inline void link(int x, int y)
105     {
106         makeroot(x);
107         if (findroot(y) != x)
108             fa[x] = y;
109     }
110
111     inline void cut(int x, int y)
112     {
113         makeroot(x);
114         if (findroot(y) == x && fa[x] == y && ch[y][0] == x && !ch[y][1])
115         {
116             fa[x] = ch[y][0] = 0;
117             push_up(y);
118         }
119     }
120
121     inline void split(int x, int y)
122     {
123         makeroot(x);
124         access(y);
125         splay(y);
126     }
127
128     // x--y 路径上节点点和
129     inline int query(int x, int y)
130     {
131         split(x, y);
132         return sum[y];
133     }
134 };
135
136 void solve(int Case)
137 {
138     /* write code here */
139     /* gl & hf */
140     int n, m;
141     rd(n, m);
142     VI a(n + 1);
143     FOR(i, 1, n) rd(a[i]);
144     LinkCutTree<int> t;
145     FOR(i, 1, n) t.newnode(a[i]);
146
147     int op, x, y;
148     FOR(_, 1, m)
149     {
150         rd(op, x, y);
151         debug(op, x, y);
152         if (op == 0)
153         {
154             pln(t.query(x, y));
155         }
156         else if (op == 1)
157         {
158             t.link(x, y);
159         }
160         else if (op == 2)

```



```

161     {
162         t.cut(x, y);
163     }
164     else
165     {
166         t.v[x] = y;
167         t.makeroot(x);
168     }
169 }
170 }

```

## 1.6 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) return y;
19         if (!y) return x;
20         if (t[x].v > t[y].v) swap(x, y); // 小根堆
21         t[x].r = merge(t[x].r, y);
22         t[t[x].r].rt = x;
23         if (t[t[x].l].dist < t[t[x].r].dist) swap(t[x].l, t[t[x].r]);
24         if (!t[x].r)
25             t[x].dist = 0;
26         else
27             t[x].dist = t[t[x].r].dist + 1;
28         return x;
29     }
30     V top(int x) {
31         if (t[x].v == -1) return -1;
32         x = find(x);
33         return t[x].v;
34     }
35     void pop(int x) {
36         if (t[x].v == -1) return;
37         x = find(x);
38         t[t[x].l].rt = t[x].l;
39         t[t[x].r].rt = t[x].r;
40         t[x].rt = merge(t[x].l, t[t[x].r]);
41         t[x].v = -1;
42     }
43 };
44
45 int n, m, a[N];
46 void solve(int Case) {
47     rd(n, m);
48     FOR(i, 1, n) rd(a[i]);
49     LeftistForest<int> T;
50     T.init(n, a);
51

```

```

52  int op, x, y;
53  FOR(_, 1, m) {
54      rd(op);
55      debug(op);
56      if (op == 1) {
57          rd(x, y);
58          if (T[x].v == -1 || T[y].v == -1) continue;
59          x = T.find(x);
60          y = T.find(y);
61          if (x == y) continue;
62          T[x].rt = T[y].rt = T.merge(x, y);
63      } else {
64          rd(x);
65          pln(T.top(x));
66          T.pop(x);
67      }
68  }
69 }

```

## 1.7 PersistentSegmentTree

```

1  struct PersistentSegmentTree
2  {
3      // SIZE = N log N
4      #define SIZE 200005 * 20
5
6      int tot;
7      int c[SIZE];
8      int L[SIZE], R[SIZE];
9
10     PersistentSegmentTree() { tot = 0; }
11
12     int update(int rt, int l, int r, int p, int d)
13     {
14         int nrt = ++tot;
15         L[nrt] = L[rt];
16         R[nrt] = R[rt];
17         c[nrt] = c[rt] + d;
18
19         if (l != r)
20         {
21             int mid = (l + r) >> 1;
22             if (p <= mid)
23                 L[nrt] = update(L[rt], l, mid, p, d);
24             else
25                 R[nrt] = update(R[rt], mid + 1, r, p, d);
26         }
27
28         return nrt;
29     }
30
31     // 区间第 k 小
32     int query(int u, int v, int l, int r, int k)
33     {
34         if (l == r)
35             return l;
36         int left_size = c[L[v]] - c[L[u]];
37         int mid = (l + r) >> 1;
38         if (k <= left_size)
39             return query(L[u], L[v], l, mid, k);
40         return query(R[u], R[v], mid + 1, r, k - left_size);
41     }
42 };

```

## 1.8 rbtrees-1

---

```

1  //#define __REDBLACK_DEBUG
2  template <typename T>
3  class rbtrees {
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     rbtrees()
32         : _root(NULL)
33         , _hot(NULL)
34     {
35     }
36
37     int get_rank(T);
38     iterator insert(T);
39     bool remove(T);
40     int size();
41     iterator kth(int);
42     iterator lower_bound(T);
43     iterator upper_bound(T);
44 #ifdef __REDBLACK_DEBUG
45     void vis();
46     void correctlyconnected();
47 #endif
48 };
49
50 template <typename T>
51 struct rbtrees<T>::Node {
52     T val;
53     bool RBc; ////true : Red ; false : Black .
54     Node* ftr;
55     Node* lc;
56     Node* rc;
57     int s;
58
59     Node(T v = T(), bool RB = true,
60         Node* f = NULL, Node* l = NULL, Node* r = NULL, int ss = 1)
61         : val(v)
62         , RBc(RB)
63         , ftr(f)

```

```
64         , lc(l)
65         , rc(r)
66         , s(ss)
67     {
68     }
69
70 Node* succ()
71 {
72     Node* ptn = rc;
73     while (ptn->lc != NULL) {
74         --(ptn->s);
75         ptn = ptn->lc;
76     }
77     return ptn;
78 }
79
80 Node* left_node()
81 {
82     Node* ptn = this;
83     if (!lc) {
84         while (ptn->ftr && ptn->ftr->lc == ptn)
85             ptn = ptn->ftr;
86         ptn = ptn->ftr;
87     } else
88         while (ptn->lc)
89             ptn = ptn->lc;
90     return ptn;
91 }
92
93 Node* right_node()
94 {
95     Node* ptn = this;
96     if (!rc) {
97         while (ptn->ftr && ptn->ftr->rc == ptn)
98             ptn = ptn->ftr;
99         ptn = ptn->ftr;
100    } else
101        while (ptn->rc)
102            ptn = ptn->rc;
103    return ptn;
104 }
105
106 void maintain()
107 {
108     s = 1;
109     if (lc)
110         s += lc->s;
111     if (rc)
112         s += rc->s;
113 }
114 };
115
116 template <typename T>
117 void rbtree<T>::connect34(Node* nroot, Node* nlc, Node* nrc,
118     Node* ntree1, Node* ntree2, Node* ntree3, Node* ntree4)
119 {
120     nlc->lc = ntree1;
121     if (ntree1 != NULL)
122         ntree1->ftr = nlc;
123     nlc->rc = ntree2;
124     if (ntree2 != NULL)
125         ntree2->ftr = nlc;
126     nrc->lc = ntree3;
127     if (ntree3 != NULL)
128         ntree3->ftr = nrc;
```

```

129     nrc->rc = ntree4;
130     if (ntree4 != NULL)
131         ntree4->ftr = nrc;
132     nroot->lc = nlc;
133     nlc->ftr = nroot;
134     nroot->rc = nrc;
135     nrc->ftr = nroot;
136     nlc->maintain();
137     nrc->maintain();
138     nroot->maintain();
139 }
140
141 #ifdef __REDBLACK_DEBUG
142
143 int blackheight(0);
144
145 template <typename T>
146 void rbtree<T>::previs(Node* ptn, int cnt)
147 {
148     if (ptn == NULL) {
149         if (blackheight == -1)
150             blackheight = cnt;
151         assert(blackheight == cnt);
152         return;
153     }
154     printf("%d %s %d \n", ptn->val, ptn->RBc ? "Red" : "Black", ptn->s);
155     if (!(ptn->RBc))
156         ++cnt;
157     previs(ptn->lc, cnt);
158     previs(ptn->rc, cnt);
159 }
160
161 template <typename T>
162 void rbtree<T>::invis(Node* ptn, int cnt)
163 {
164     if (ptn == NULL) {
165         if (blackheight == -1)
166             blackheight = cnt;
167         assert(blackheight == cnt);
168         return;
169     }
170     if (!(ptn->RBc))
171         ++cnt;
172     invis(ptn->lc, cnt);
173     printf("%d %s %d \n", ptn->val, ptn->RBc ? "Red" : "Black", ptn->s);
174     invis(ptn->rc, cnt);
175 }
176
177 template <typename T>
178 void rbtree<T>::postvis(Node* ptn, int cnt)
179 {
180     if (ptn == NULL) {
181         if (blackheight == -1)
182             blackheight = cnt;
183         assert(blackheight == cnt);
184         return;
185     }
186     if (!(ptn->RBc))
187         ++cnt;
188     postvis(ptn->lc, cnt);
189     postvis(ptn->rc, cnt);
190     printf("%d %s %d \n", ptn->val, ptn->RBc ? "Red" : "Black", ptn->s);
191 }
192
193 template <typename T>

```

```

194 void rbtree<T>::vis()
195 {
196     printf("BlackHeight:\t%d\n", blackheight);
197     printf("-----pre-vis-----\n");
198     previs(_root, 0);
199     printf("-----in-vis-----\n");
200     invis(_root, 0);
201     printf("-----post-vis-----\n");
202     postvis(_root, 0);
203 }
204
205 template <typename T>
206 void rbtree<T>::checkconnect(Node* ptn)
207 {
208     if (!ptn)
209         return;
210     assert(ptn->s > 0);
211     if (ptn->lc && ptn->lc->ftr != ptn) {
212         printf("Oops! %d has a lc %d, but it failed to point its ftr!\n", ptn->val, ptn->lc->val);
213     }
214     if (ptn->rc && ptn->rc->ftr != ptn) {
215         printf("Oops! %d has a rc %d, but it failed to point its ftr!\n", ptn->val, ptn->rc->val);
216     }
217     int sss = ptn->s;
218     if (ptn->lc)
219         sss -= ptn->lc->s;
220     if (ptn->rc)
221         sss -= ptn->rc->s;
222     if (sss - 1) {
223         printf("Fuck it! %d's size is %d, but the sum of its children's size is %d!\n", ptn->val, ptn->s, ptn->s - sss);
224     }
225     checkconnect(ptn->lc);
226     checkconnect(ptn->rc);
227 }
228
229 template <typename T>
230 void rbtree<T>::correctlyconnected()
231 {
232     checkconnect(_root);
233 }
234
235 #endif
236
237 template <typename T>
238 void rbtree<T>::init(T v)
239 {
240     _root = new Node(v, false, NULL, NULL, NULL, 1);
241     #ifdef __REDBLACK_DEBUG
242     ++blackheight;
243     #endif
244 }
245
246 template <typename T>
247 void rbtree<T>::SolveDoubleRed(Node* nn)
248 {
249     while ((!nn->ftr) || nn->ftr->RBc) {
250         if (nn == _root) {
251             _root->RBc = false;
252             #ifdef __REDBLACK_DEBUG
253             ++blackheight;
254             #endif
255             return;
256         }
257         Node* pftr = nn->ftr;
258         if (!pftr->RBc)

```

```

259     return; ///No double-red
260     Node* uncle = bro(nn->ftr);
261     Node* grdftr = nn->ftr->ftr;
262     if (uncle != NULL && uncle->RBc) { ///RR-2
263         grdftr->RBc = true;
264         uncle->RBc = false;
265         pftr->RBc = false;
266         nn = grdftr;
267     } else { ///RR-1
268         if (islc(pftr)) {
269             if (islc(nn)) {
270                 pftr->ftr = grdftr->ftr;
271                 if (grdftr == _root)
272                     _root = pftr;
273                 else if (grdftr->ftr->lc == grdftr)
274                     grdftr->ftr->lc = pftr;
275                 else
276                     grdftr->ftr->rc = pftr;
277                 connect34(pftr, nn, grdftr, nn->lc, nn->rc, pftr->rc, uncle);
278                 pftr->RBc = false;
279                 grdftr->RBc = true;
280             } else {
281                 nn->ftr = grdftr->ftr;
282                 if (grdftr == _root)
283                     _root = nn;
284                 else if (grdftr->ftr->lc == grdftr)
285                     grdftr->ftr->lc = nn;
286                 else
287                     grdftr->ftr->rc = nn;
288                 connect34(nn, pftr, grdftr, pftr->lc, nn->lc, nn->rc, uncle);
289                 nn->RBc = false;
290                 grdftr->RBc = true;
291             }
292         } else {
293             if (islc(nn)) {
294                 nn->ftr = grdftr->ftr;
295                 if (grdftr == _root)
296                     _root = nn;
297                 else if (grdftr->ftr->lc == grdftr)
298                     grdftr->ftr->lc = nn;
299                 else
300                     grdftr->ftr->rc = nn;
301                 connect34(nn, grdftr, pftr, uncle, nn->lc, nn->rc, pftr->rc);
302                 nn->RBc = false;
303                 grdftr->RBc = true;
304             } else {
305                 pftr->ftr = grdftr->ftr;
306                 if (grdftr == _root)
307                     _root = pftr;
308                 else if (grdftr->ftr->lc == grdftr)
309                     grdftr->ftr->lc = pftr;
310                 else
311                     grdftr->ftr->rc = pftr;
312                 connect34(pftr, grdftr, nn, uncle, pftr->lc, nn->lc, nn->rc);
313                 pftr->RBc = false;
314                 grdftr->RBc = true;
315             }
316         }
317     }
318     return;
319 }
320 }
321
322 template <typename T>
323 void rbtree<T>::SolveDoubleBlack(Node* nn)

```

```

324 {
325     while (nn != _root) {
326         Node* pftr = nn->ftr;
327         Node* bthr = bro(nn);
328         if (bthr->RBc) { ///BB-1
329             bthr->RBc = false;
330             pftr->RBc = true;
331             if (_root == pftr)
332                 _root = bthr;
333             if (pftr->ftr) {
334                 if (pftr->ftr->lc == pftr)
335                     pftr->ftr->lc = bthr;
336                 else
337                     pftr->ftr->rc = bthr;
338             }
339             bthr->ftr = pftr->ftr;
340             if (islc(nn)) {
341                 connect34(bthr, pftr, bthr->rc, nn, bthr->lc, bthr->rc->lc, bthr->rc->rc);
342             } else {
343                 connect34(bthr, bthr->lc, pftr, bthr->lc->lc, bthr->lc->rc, bthr->rc, nn);
344             }
345             bthr = bro(nn);
346             pftr = nn->ftr;
347         }
348         if (bthr->lc && bthr->lc->RBc) { ///BB-3
349             bool oldRBc = pftr->RBc;
350             pftr->RBc = false;
351             if (pftr->lc == nn) {
352                 if (pftr->ftr) {
353                     if (pftr->ftr->lc == pftr)
354                         pftr->ftr->lc = bthr->lc;
355                     else
356                         pftr->ftr->rc = bthr->lc;
357                 }
358                 bthr->lc->ftr = pftr->ftr;
359                 if (_root == pftr)
360                     _root = bthr->lc;
361                 connect34(bthr->lc, pftr, bthr, pftr->lc, bthr->lc->lc, bthr->lc->rc, bthr->rc);
362             } else {
363                 bthr->lc->RBc = false;
364                 if (pftr->ftr) {
365                     if (pftr->ftr->lc == pftr)
366                         pftr->ftr->lc = bthr;
367                     else
368                         pftr->ftr->rc = bthr;
369                 }
370                 bthr->ftr = pftr->ftr;
371                 if (_root == pftr)
372                     _root = bthr;
373                 connect34(bthr, bthr->lc, pftr, bthr->lc->lc, bthr->lc->rc, bthr->rc, pftr->rc);
374             }
375             pftr->ftr->RBc = oldRBc;
376             return;
377         } else if (bthr->rc && bthr->rc->RBc) { ///BB-3
378             bool oldRBc = pftr->RBc;
379             pftr->RBc = false;
380             if (pftr->lc == nn) {
381                 bthr->rc->RBc = false;
382                 if (pftr->ftr) {
383                     if (pftr->ftr->lc == pftr)
384                         pftr->ftr->lc = bthr;
385                     else
386                         pftr->ftr->rc = bthr;
387                 }
388                 bthr->ftr = pftr->ftr;

```



```

389         if (_root == pftr)
390             _root = bthr;
391         connect34(bthr, pftr, bthr->rc, pftr->lc, bthr->lc, bthr->rc->lc, bthr->rc->rc);
392     } else {
393         if (pftr->ftr) {
394             if (pftr->ftr->lc == pftr)
395                 pftr->ftr->lc = bthr->rc;
396             else
397                 pftr->ftr->rc = bthr->rc;
398         }
399         bthr->rc->ftr = pftr->ftr;
400         if (_root == pftr)
401             _root = bthr->rc;
402         connect34(bthr->rc, bthr, pftr, bthr->lc, bthr->rc->lc, bthr->rc->rc, pftr->rc);
403     }
404     pftr->ftr->RBc = oldRBc;
405     return;
406 }
407 if (pftr->RBc) { ///BB-2R
408     pftr->RBc = false;
409     bthr->RBc = true;
410     return;
411 } else { ///BB-2B
412     bthr->RBc = true;
413     nn = pftr;
414 }
415 }
416 #ifdef __REDBLACK_DEBUG
417     --blackheight;
418 #endif
419 }
420
421 template <typename T>
422 typename rbtree<T>::Node* rbtree<T>::findkth(int rank, Node* ptn)
423 {
424     if (!(ptn->lc)) {
425         if (rank == 1) {
426             return ptn;
427         } else {
428             return findkth(rank - 1, ptn->rc);
429         }
430     } else {
431         if (ptn->lc->s == rank - 1)
432             return ptn;
433         else if (ptn->lc->s >= rank)
434             return findkth(rank, ptn->lc);
435         else
436             return findkth(rank - (ptn->lc->s) - 1, ptn->rc);
437     }
438 }
439
440 template <typename T>
441 int rbtree<T>::find_rank(T v, Node* ptn)
442 {
443     if (!ptn)
444         return 1;
445     else if (ptn->val >= v)
446         return find_rank(v, ptn->lc);
447     else
448         return (1 + ((ptn->lc) ? (ptn->lc->s) : 0) + find_rank(v, ptn->rc));
449 }
450
451 template <typename T>
452 int rbtree<T>::get_rank(T v)
453 {

```

```
454     return find_rank(v, _root);
455 }
456
457 template <typename T>
458 typename rbtree<T>::Node* rbtree<T>::find(T v, const int op)
459 {
460     Node* ptn = _root;
461     _hot = NULL;
462     while (ptn != NULL) {
463         _hot = ptn;
464         ptn->s += op;
465         if (ptn->val > v)
466             ptn = ptn->lc;
467         else
468             ptn = ptn->rc;
469     }
470     return ptn;
471 }
472
473 template <typename T>
474 typename rbtree<T>::Node* rbtree<T>::rfind(T v, const int op)
475 {
476     Node* ptn = _root;
477     _hot = NULL;
478     while (ptn != NULL && ptn->val != v) {
479         _hot = ptn;
480         ptn->s += op;
481         if (ptn->val > v)
482             ptn = ptn->lc;
483         else
484             ptn = ptn->rc;
485     }
486     return ptn;
487 }
488
489 template <typename T>
490 struct rbtree<T>::iterator {
491 private:
492     Node* _real__node;
493
494 public:
495     iterator& operator++()
496     {
497         _real__node = _real__node->right_node();
498         return *this;
499     }
500
501     iterator& operator--()
502     {
503         _real__node = _real__node->left_node();
504         return *this;
505     }
506
507     T operator*()
508     {
509         return _real__node->val;
510     }
511
512     iterator(Node* node_nn = NULL)
513         : _real__node(node_nn)
514     {
515     }
516     iterator(T const& val_vv)
517         : _real__node(rfind(val_vv, 0))
518     {
519     }
```

```

519     }
520     iterator(iterator const& iter)
521         : _real__node(iter._real__node)
522     {
523     }
524 };
525
526 template <typename T>
527 typename rbtree<T>::iterator rbtree<T>::insert(T v)
528 {
529     Node* ptn = find(v, 1);
530     if (_hot == NULL) {
531         init(v);
532         return iterator(_root);
533     }
534     ptn = new Node(v, true, _hot, NULL, NULL, 1);
535     if (_hot->val <= v)
536         _hot->rc = ptn;
537     else
538         _hot->lc = ptn;
539     SolveDoubleRed(ptn);
540     return iterator(ptn);
541 }
542
543 template <typename T>
544 bool rbtree<T>::remove(T v)
545 {
546     Node* ptn = rfind(v, -1);
547     if (!ptn)
548         return false;
549     Node* node_suc;
550     while (ptn->lc || ptn->rc) {
551         if (!(ptn->lc)) {
552             node_suc = ptn->rc;
553         } else if (!(ptn->rc)) {
554             node_suc = ptn->lc;
555         } else {
556             node_suc = ptn->succ();
557         }
558         --(ptn->s);
559         ptn->val = node_suc->val;
560         ptn = node_suc;
561     }
562     if (!(ptn->RBc)) {
563         --(ptn->s);
564         SolveDoubleBlack(ptn);
565     }
566     if (ptn->ftr->lc == ptn)
567         ptn->ftr->lc = NULL;
568     else
569         ptn->ftr->rc = NULL;
570     delete ptn;
571     return true;
572 }
573
574 template <typename T>
575 int rbtree<T>::size()
576 {
577     return _root->s;
578 }
579
580 template <typename T>
581 typename rbtree<T>::iterator rbtree<T>::kth(int rank)
582 {
583     return iterator(findkth(rank, _root));

```

```

584 }
585
586 template <typename T>
587 typename rbtree<T>::iterator rbtree<T>::lower_bound(T v)
588 {
589     Node* ptn = _root;
590     while (ptn) {
591         _hot = ptn;
592         if (ptn->val < v) {
593             ptn = ptn->rc;
594         } else {
595             ptn = ptn->lc;
596         }
597     }
598     if (_hot->val < v) {
599         ptn = _hot;
600     } else {
601         ptn = _hot->left_node();
602     }
603     return iterator(ptn);
604 }
605
606 template <typename T>
607 typename rbtree<T>::iterator rbtree<T>::upper_bound(T v)
608 {
609     Node* ptn = _root;
610     while (ptn) {
611         _hot = ptn;
612         if (ptn->val > v) {
613             ptn = ptn->lc;
614         } else {
615             ptn = ptn->rc;
616         }
617     }
618     if (_hot->val > v) {
619         ptn = _hot;
620     } else {
621         ptn = _hot->right_node();
622     }
623     return iterator(ptn);
624 }

```

---

## 1.9 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     {
14         while (x->l != nil)
15             x = x->l;
16         return x;
17     }
18     node* find_max(node* x)
19     {
20         while (x->r != nil)

```

```

21         x = x->r;
22     return x;
23 }
24 node* find_node(const T& val)
25 {
26     node* x = root;
27     while (x != nil) {
28         if (x->val == val)
29             return x;
30         if (x->val < val)
31             x = x->r;
32         else
33             x = x->l;
34     }
35     return NULL;
36 }
37 void zig(node* x)
38 {
39     node* y = x->r;
40     x->r = y->l;
41     if (y->l != nil)
42         y->l->p = x;
43     y->p = x->p;
44     if (x->p == nil)
45         root = y;
46     else if (x == x->p->r)
47         x->p->r = y;
48     else
49         x->p->l = y;
50     y->l = x;
51     x->p = y;
52     y->sz = x->sz;
53     x->sz = x->l->sz + x->r->sz + x->cnt;
54     return;
55 }
56 void zag(node* x)
57 {
58     node* y = x->l;
59     x->l = y->r;
60     if (y->r != nil)
61         y->r->p = x;
62     y->p = x->p;
63     if (x->p == nil)
64         root = y;
65     else if (x == x->p->l)
66         x->p->l = y;
67     else
68         x->p->r = y;
69     y->r = x;
70     x->p = y;
71     y->sz = x->sz;
72     x->sz = x->l->sz + x->r->sz + x->cnt;
73     return;
74 }
75 void insert_fixup(node* z)
76 {
77     while (z->p->color == 1) {
78         if (z->p == z->p->p->l) {
79             node* y = z->p->p->r;
80             if (y->color == 1) {
81                 y->color = z->p->color = 0;
82                 z->p->p->color = 1;
83                 z = z->p->p;
84             } else {
85                 if (z == z->p->r) {

```

```

86         z = z->p;
87         zig(z);
88     }
89     z->p->color = 0;
90     z->p->p->color = 1;
91     zag(z->p->p);
92 }
93 } else {
94     node* y = z->p->p->l;
95     if (y->color == 1) {
96         y->color = z->p->color = 0;
97         z->p->p->color = 1;
98         z = z->p->p;
99     } else {
100         if (z == z->p->l) {
101             z = z->p;
102             zag(z);
103         }
104         z->p->color = 0;
105         z->p->p->color = 1;
106         zig(z->p->p);
107     }
108 }
109 }
110 root->color = 0;
111 return;
112 }
113 void transplant(node* x, node* y)
114 {
115     y->p = x->p;
116     if (x->p == nil)
117         root = y;
118     else if (x == x->p->l)
119         x->p->l = y;
120     else
121         x->p->r = y;
122     return;
123 }
124 void delete_fixup(node* x)
125 {
126     while (x != root && x->color == 0) {
127         if (x == x->p->l) {
128             node* w = x->p->r;
129             if (w->color == 1) {
130                 x->p->color = 1;
131                 w->color = 0;
132                 zig(x->p);
133                 w = x->p->r;
134             }
135             if (w->l->color == 0 && w->r->color == 0) {
136                 w->color = 1;
137                 x = x->p;
138             } else {
139                 if (w->r->color == 0) {
140                     w->color = 1;
141                     w->l->color = 0;
142                     zag(w);
143                     w = x->p->r;
144                 }
145                 w->color = x->p->color;
146                 x->p->color = 0;
147                 w->r->color = 0;
148                 zig(w->p);
149                 x = root;
150             }

```

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

```

216         if (z->val == val)
217             break;
218         if (z->val < val)
219             z = z->r;
220         else
221             z = z->l;
222     }
223     if (z != nil) {
224         // delete only one node
225         if (z->cnt > 1) {
226             --z->cnt;
227             return;
228         }
229
230         node* y = z;
231         node* x;
232         bool history = y->color;
233         if (z->l == nil) {
234             x = z->r;
235             transplant(z, z->r);
236         } else if (z->r == nil) {
237             x = z->l;
238             transplant(z, z->l);
239         } else {
240             y = find_min(z->r);
241             history = y->color;
242             x = y->r;
243             if (y->p == z)
244                 x->p = y;
245             else {
246                 node* w = y;
247                 while (w != z) {
248                     w->sz -= y->cnt;
249                     w = w->p;
250                 }
251                 transplant(y, y->r);
252                 y->r = z->r;
253                 y->r->p = y;
254             }
255             transplant(z, y);
256             y->l = z->l;
257             y->l->p = y;
258             y->color = z->color;
259             y->sz = y->l->sz + y->r->sz + y->cnt;
260         }
261         if (history == 0)
262             delete_fixup(x);
263     } else
264         while (w != nil) {
265             ++w->sz;
266             w = w->p;
267         }
268     return;
269 }
270 T getKth(int k)
271 {
272     T res = 0;
273     node* x = root;
274     while (x != nil) {
275         if (x->l->sz + 1 <= k && x->l->sz + x->cnt >= k) {
276             res = x->val;
277             break;
278         } else if (x->l->sz + x->cnt < k) {
279             k -= x->l->sz + x->cnt;
280             x = x->r;

```



```

281         } else {
282             x = x->l;
283         }
284     }
285     return res;
286 }
287 int getRank(const T& val)
288 {
289     int rk = 0;
290     node* x = root;
291     while (x != nil) {
292         if (x->val < val) {
293             rk += x->l->sz + x->cnt;
294             x = x->r;
295         } else {
296             if (x->val == val)
297                 ++rk;
298             x = x->l;
299         }
300     }
301     return rk;
302 }
303 T getSucc(const T& val)
304 {
305     ins(val);
306     T res = INT_MAX;
307     node* x = find_node(val);
308     if (x->r != nil) {
309         res = find_min(x->r->val);
310     } else {
311         while (x->p->r == x)
312             x = x->p;
313         if (x->p != nil)
314             res = x->p->val;
315     }
316     del(val);
317     return res;
318 }
319 T getPrev(const T& val)
320 {
321     ins(val);
322     T res = INT_MIN;
323     node* x = find_node(val);
324     if (x->l != nil)
325         res = find_max(x->l->val);
326     else {
327         while (x->p->l == x)
328             x = x->p;
329         if (x->p != nil)
330             res = x->p->val;
331     }
332     del(val);
333     return res;
334 }
335 };

```

## 1.10 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) lg[i] = lg[i >> 1] + 1;
6  }

```

```

7
8 void build_rmq(int n, int* a) {
9     for (int i = 1; i <= n; ++i) mi[i][0] = a[i];
10    for (int j = 1; j <= lg[n]; ++j) {
11        for (int i = 1; i + (1 << (j - 1)) <= n; ++i) {
12            mi[i][j] = min(mi[i][j - 1], mi[i + (1 << (j - 1))][j - 1]);
13        }
14    }
15 }
16
17 int rmqMin(int l, int r) {
18     int k = lg[r - l + 1];
19     return min(mi[l][k], mi[r - (1 << k) + 1][k]);
20 }

```

---

## 1.11 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) : l(_l), r(_r), id(_id) {}
9     bool operator < (const Query& q) const {
10         if (block_id[l] == block_id[q.l]) return r < q.r;
11         return block_id[l] < block_id[q.l];
12     }
13 } Q[N];
14
15 int n, m, q, a[N], b[N];
16
17
18 int nums[N], cn;
19 int mi[N], ma[N];
20 int __mi[N];
21
22 int brute_force(int l, int r) {
23     int res = 0;
24     for (int i = l; i <= r; ++i) __mi[a[i]] = 0;
25     for (int i = l; i <= r; ++i) {
26         if (__mi[a[i]]) res = max(res, i - __mi[a[i]]);
27         else __mi[a[i]] = i;
28     }
29     return res;
30 }
31
32 inline void addl(int p) {
33     if (ma[a[p]]) Ans = max(Ans, ma[a[p]] - p);
34     else ma[a[p]] = p;
35 }
36
37 inline void addr(int p) {
38     ma[a[p]] = p;
39     if (!mi[a[p]]) mi[a[p]] = p, nums[++cn] = a[p];
40     Ans = max(Ans, p - mi[a[p]]);
41 }
42
43 inline void dell(int p) {
44     if (ma[a[p]] == p) ma[a[p]] = 0;
45 }
46
47 inline void delr(int p) {

```

```

48 }
49 }
50
51 inline void clear() {
52     for (int i = 1; i <= cn; ++i) mi[nums[i]] = ma[nums[i]] = 0;
53 }
54
55 void RollBackCaptainMo() {
56     block_sz = sqrt(n); block_cnt = n / block_sz;
57
58     for (int i = 1; i <= block_cnt; ++i) L[i] = R[i - 1] + 1, R[i] = i * block_sz;
59     if (R[block_cnt] < n) { ++block_cnt; L[block_cnt] = R[block_cnt - 1] + 1; R[block_cnt] = n; }
60
61     for (int i = 1; i <= block_cnt; ++i)
62         for (int j = L[i]; j <= R[i]; ++j)
63             block_id[j] = i;
64
65     sort(Q + 1, Q + 1 + q);
66
67     for (int i = 1, j = 1; j <= block_cnt; ++j) {
68         int l = R[j] + 1, r = R[j];
69         Ans = 0; cn = 0;
70         for (; block_id[Q[i].l] == j; ++i) {
71             if (block_id[Q[i].l] == block_id[Q[i].r]) ans[Q[i].id] = brute_force(Q[i].l, Q[i].r);
72             else {
73                 while(r < Q[i].r) ++r, addr(r);
74                 int tmp = Ans;
75                 while(l > Q[i].l) --l, addl(l);
76                 ans[Q[i].id] = Ans;
77                 while(l <= R[j]) dell(l), ++l;
78                 Ans = tmp;
79             }
80         }
81         clear();
82     }
83 }

```

## 1.12 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 apply(int l, int r, ll addv) {
10             // 更新节点信息
11             // ie. 最大值 + 区间加: mx = mx + addv
12
13             ...
14         }
15     };
16
17     friend node operator + (const node& t1, const node& tr) {
18         node t;
19         // 合并两个区间的信息
20         // ie. 区间和: t.sum = t1.sum + t2.sum;
21
22         ...
23
24         return t;
25     }

```

```

26
27 inline void push_down(int x, int l, int r) {
28     int lc = x << 1, rc = lc | 1, mid = (l + r) >> 1;
29     // 标记下传
30     // ie. 区间加法
31     // if (tr[x].add != 0) {
32     //     tr[lc].apply(l, mid, tr[x].add);
33     //     tr[rc].apply(mid + 1, r, tr[x].add);
34     //     tr[x].add = 0;
35     // }
36
37     ...
38 }
39
40 /*****
41 inline void push_up(int x) {
42     int lc = x << 1, rc = lc | 1;
43     tr[x] = tr[lc] + tr[rc];
44 }
45
46 int n;
47 vector<node> tr;
48
49 void build(int x, int l, int r) {
50     if (l == r) {
51         return;
52     }
53     int lc = x << 1, rc = lc | 1, mid = (l + r) >> 1;
54     build(lc, l, mid);
55     build(rc, mid + 1, r);
56     push_up(x);
57 }
58
59 template<class T>
60 void build(int x, int l, int r, const vector<T>& arr){
61     if (l == r) {
62         tr[x].apply(l, r, arr[l]);
63         return;
64     }
65     int lc = x << 1, rc = lc | 1, mid = (l + r) >> 1;
66     build(lc, l, mid, arr);
67     build(rc, mid + 1, r, arr);
68     push_up(x);
69 }
70
71 template<class T>
72 void build(int x, int l, int r, T* arr){
73     if (l == r) {
74         tr[x].apply(l, r, arr[l]);
75         return;
76     }
77     int lc = x << 1, rc = lc | 1, mid = (l + r) >> 1;
78     build(lc, l, mid);
79     build(rc, mid + 1, r);
80     push_up(x);
81 }
82
83 node get(int x, int l, int r, int L, int R) {
84     if (L <= l && r <= R) {
85         return tr[x];
86     }
87     push_down(x, l, r);
88     int lc = x << 1, rc = lc | 1, mid = (l + r) >> 1;
89     node res;
90     if (R <= mid) res = get(lc, l, mid, L, R);

```

```

91     else if (L > mid) res = get(rc, mid + 1, r, L, R);
92     else res = get(lc, l, mid, L, mid) + get(rc, mid + 1, r, mid + 1, R);
93     push_up(x);
94     return res;
95 }
96
97 template<class... T>
98 void upd(int x, int l, int r, int L, int R, const T&... v) {
99     if (L <= l && r <= R) {
100         tr[x].apply(l, r, v...);
101         return;
102     }
103     push_down(x, l, r);
104     int lc = x << 1, rc = lc | 1, mid = (l + r) >> 1;
105     node res;
106     if (L <= mid) upd(lc, l, mid, L, R, v...);
107     if (R > mid) upd(rc, mid + 1, r, L, R, v...);
108     push_up(x);
109 }
110
111 int __get_first(int x, int l, int r, const function<bool(const node&)> &f) {
112     if (l == r) {
113         return l;
114     }
115     int lc = x << 1, rc = lc | 1, mid = (l + r) >> 1;
116     push_down(x, l, r);
117     int res;
118     if (f(tr[lc])) res = __get_first(lc, l, mid, f);
119     else res = __get_first(rc, mid + 1, r, f);
120     push_up(x);
121     return res;
122 }
123
124 int get_first(int x, int l, int r, int L, int R, const function<bool(const node&)> &f) {
125     if (L <= l && r <= R) {
126         if (!f(tr[x])) {
127             return -1;
128         }
129         return __get_first(x, l, r, f);
130     }
131     push_down(x, l, r);
132     int lc = x << 1, rc = lc | 1, mid = (l + r) >> 1;
133     int res;
134     if (L <= mid) res = get_first(lc, l, mid, L, R, f);
135     if (res == -1 && R > mid) res = get_first(rc, mid + 1, r, L, R, f);
136     push_up(x);
137     return res;
138 }
139
140 int __get_last(int x, int l, int r, const function<bool(const node&)> &f) {
141     if (l == r) {
142         return l;
143     }
144     int lc = x << 1, rc = lc | 1, mid = (l + r) >> 1;
145     push_down(x, l, r);
146     int res;
147     if (f(tr[lc])) res = __get_first(rc, mid + 1, r, f);
148     else res = __get_first(lc, l, mid, f);
149     push_up(x);
150     return res;
151 }
152
153 int get_last(int x, int l, int r, int L, int R, const function<bool(const node&)> &f) {
154     if (L <= l && r <= R) {
155         if (!f(tr[x])) {

```

```

156         return -1;
157     }
158     return __get_first(x, l, r, f);
159 }
160 push_down(x, l, r);
161 int lc = x << 1, rc = lc | 1, mid = (l + r) >> 1;
162 int res;
163 if (R > mid) res = get_last(rc, mid + 1, r, L, R, f);
164 if (res == -1 && L <= mid) res = get_last(lc, l, mid, L, R, f);
165 push_up(x);
166 return res;
167 }
168
169 int find_first(int l, int r, const function<bool(const node&)> &f) {
170     int L = l, R = r, mid, res = -1;
171     while(L <= R) {
172         mid = (L + R) >> 1;
173         if (f(get(l, mid))) R = mid - 1, res = mid;
174         else L = mid + 1;
175     }
176     return res;
177 }
178
179 int find_last(int l, int r, const function<bool(const node&)> &f) {
180     int L = l, R = r, mid, res = -1;
181     while(L <= R) {
182         mid = (L + R) >> 1;
183         if (f(get(l, mid))) L = mid + 1, res = mid;
184         else R = mid - 1;
185     }
186     return res;
187 }
188
189 segtree(int _n) : n(_n) {
190     assert(n > 0);
191     tr.resize((n << 2) + 5);
192     build(1, 1, n);
193 }
194
195 template<class T>
196 segtree(const vector<T>& arr) {
197     n = arr.size() - 1;
198     assert(n > 0);
199     tr.resize((n << 2) + 5);
200     build(1, 1, n, arr);
201 }
202
203 template<class T>
204 segtree(int _n, T* arr) {
205     n = _n;
206     assert(n > 0);
207     tr.resize((n << 2) + 5);
208     build(1, 1, n, arr);
209 }
210
211 node get(int l, int r) {
212     assert(l >= 1 && l <= r && r <= n);
213     return get(1, 1, n, l, r);
214 }
215
216 node get(int p) {
217     assert(1 <= p && p <= n);
218     return get(1, 1, n, p, p);
219 }
220

```

```

221 template <class... T>
222 void upd(int l, int r, const T&... v) {
223     assert(l >= 1 && l <= r && r <= n);
224     upd(1, 1, n, l, r, v...);
225 }
226
227 template <class... T>
228 void upd1(int p, const T&... v) {
229     assert(p >= 1 && p <= n);
230     upd(1, 1, n, p, p, v...);
231 }
232
233 int get_first(int l, int r, const function<bool(const node&)> &f) {
234     assert(l >= 1 && l <= r && r <= n);
235     return get_first(1, 1, n, l, r, f);
236 }
237
238
239 int get_last(int l, int r, const function<bool(const node&)> &f) {
240     assert(l >= 1 && l <= r && r <= n);
241     return get_last(1, 1, n, l, r, f);
242 }
243
244 void print(int x, int l, int r) {
245     if (l == r) {
246         cerr << tr[x].sum << " ";
247         return;
248     }
249     push_down(x, l, r);
250     int lc = x << 1, rc = lc | 1, mid = (l + r) >> 1;
251     print(lc, l, mid);
252     print(rc, mid + 1, r);
253 }
254
255 void print() {
256     #ifdef BACKLIGHT
257     cerr << "SGTREE: " << endl;
258     print(1, 1, n);
259     cerr << "\n-----" << endl;
260     #endif
261 }
262 };

```

### 1.13 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
9     SGTree()
10    {
11        root = tot = 0;
12    }
13
14    int new_node(T _v)
15    {
16        ++tot;
17        v[tot] = _v;
18        s[tot] = sz[tot] = sd[tot] = cnt[tot] = 1;
19        l[tot] = r[tot] = 0;

```

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



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

```

150     T getSucc(T val)
151     {
152         return getKth(getSuccRank(root, val));
153     }
154
155     void debug(int x)
156     {
157         if (!x) return;
158         debug(l[x]);
159         cerr << v[x] << " ";
160         debug(r[x]);
161     }
162
163     void debug()
164     {
165         cerr << "SGTree:" << endl;
166         debug(root);
167         cerr << endl;
168     }
169 };

```

## 1.14 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() { tot = rt = 0; }
14
15      inline void clear(int x) { ch[x][0] = ch[x][1] = fa[x] = sz[x] = cnt[x] = v[x] = 0; }
16
17      inline int get(int x) { return ch[fa[x]][1] == x; }
18
19      inline int newnode(T val) {
20          ++tot;
21          sz[tot] = cnt[tot] = 1;
22          ch[tot][0] = ch[tot][1] = fa[tot] = 0;
23          v[tot] = val;
24          return tot;
25      }
26
27      inline void push_up(int x) {
28          if (!x) return;
29          sz[x] = sz[ls] + cnt[x] + sz[rs];
30      }
31
32      void rotate(int x) {
33          int f = fa[x], g = fa[f], i = get(x);
34          ch[f][i] = ch[x][i^1]; fa[ch[f][i]] = f;
35          ch[x][i^1] = f; fa[f] = x;
36          fa[x] = g;
37          if (g) ch[g][ch[g][1] == f] = x;
38          push_up(f); push_up(x);
39      }
40
41      void splay(int x, int ed) {

```

```

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

```

```

107     }
108     int ort = rt;
109     int p = ch[rt][0]; while(ch[p][1]) p = ch[p][1];
110     splay(p, 0);
111     ch[rt][1] = ch[ort][1];
112     fa[ch[ort][1]] = rt;
113     clear(ort);
114     push_up(rt);
115 }
116
117 T getpre(T val) {
118     int p = rt, res = -INF;
119     while(p) {
120         if (v[p] < val && v[p] > res) res = v[p];
121         if (val > v[p]) p = ch[p][1];
122         else p = ch[p][0];
123     }
124     // splay(p, 0);
125     return res;
126 }
127
128 T getsuc(T val) {
129     int p = rt, res = INF;
130     while(p) {
131         if (v[p] > val && v[p] < res) res = v[p];
132         if (val < v[p]) p = ch[p][0];
133         else p = ch[p][1];
134     }
135     // splay(p, 0);
136     return res;
137 }
138
139 void DEBUG(int x) {
140     if (!x) return;
141     DEBUG(ls);
142     cerr << v[x] << " ";
143     DEBUG(rs);
144 }
145
146 void DEBUG() {
147     cerr << "Splay: ";
148     DEBUG(rt);
149     cerr << endl;
150 }
151 } // namespace Splay
152
153 } // namespace Backlight

```

## 1.15 Treap-dynamic

```

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

```

```
80         push_up(a);
81     } else {
82         b = p;
83         split_size(p->l, k, a, b->l);
84         push_up(b);
85     }
86 }
87 }
88
89 void ins(T val)
90 {
91     node *a, *b;
92     split_val(root, val, a, b);
93     a = merge(a, new node(val));
94     root = merge(a, b);
95 }
96
97 void del(T val)
98 {
99     node *a, *b, *c, *d;
100    split_val(root, val, a, b);
101    split_val(a, val - 1, c, d);
102    node* e = d;
103    d = merge(d->l, d->r);
104    delete e;
105    a = merge(c, d);
106    root = merge(a, b);
107 }
108
109 T getRank(T val)
110 {
111     node *a, *b;
112     split_val(root, val - 1, a, b);
113     T res = get_size(a) + 1;
114     root = merge(a, b);
115     return res;
116 }
117
118 T getKth(int k)
119 {
120     node* x = root;
121     T res = numeric_limits<T>::min();
122     while (x) {
123         if (k <= get_size(x->l))
124             x = x->l;
125         else {
126             if (get_size(x->l) + 1 == k) {
127                 res = x->v;
128                 break;
129             } else {
130                 k -= get_size(x->l) + 1;
131                 x = x->r;
132             }
133         }
134     }
135     return res;
136 }
137
138 T getPrev(T val)
139 {
140     node *a, *b;
141     split_val(root, val - 1, a, b);
142     node* p = a;
143     while (p->r)
144         p = p->r;
```

```

145     root = merge(a, b);
146     return p->v;
147 }
148
149 T getSucc(T val)
150 {
151     node *a, *b;
152     split_val(root, val, a, b);
153     node* p = b;
154     while (p->l)
155         p = p->l;
156     root = merge(a, b);
157     return p->v;
158 }
159 };

```

---

## 1.16 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)
16             , r(NULL)
17             , rnd(rng())
18             , sz(1)
19             , v(_v)
20         {
21         }
22     };
23
24     inline int get_size(node*& p)
25     {
26         return p ? p->sz : 0;
27     }
28
29     inline void push_up(node*& p)
30     {
31         if (!p)
32             return;
33         p->sz = get_size(p->l) + get_size(p->r) + 1;
34     }
35
36     node* root = NULL;
37
38     node* merge(node* a, node* b)
39     {
40         if (!a)
41             return b;
42         if (!b)
43             return a;
44         if (a->rnd < b->rnd) {
45             a->r = merge(a->r, b);
46             push_up(a);

```

```

47         return a;
48     } else {
49         b->l = merge(a, b->l);
50         push_up(b);
51         return b;
52     }
53 }
54
55 void split_val(node* p, const T& k, node*& a, node*& b)
56 {
57     if (!p)
58         a = b = NULL;
59     else {
60         if (p->v <= k) {
61             a = p;
62             split_val(p->r, k, a->r, b);
63             push_up(a);
64         } else {
65             b = p;
66             split_val(p->l, k, a, b->l);
67             push_up(b);
68         }
69     }
70 }
71
72 void split_size(node* p, int k, node*& a, node*& b)
73 {
74     if (!p)
75         a = b = NULL;
76     else {
77         if (get_size(p->l) < k) {
78             a = p;
79             split_size(p->r, k - get_size(p->l) - 1, a->r, b);
80             push_up(a);
81         } else {
82             b = p;
83             split_size(p->l, k, a, b->l);
84             push_up(b);
85         }
86     }
87 }
88
89 void ins(T val)
90 {
91     node *a, *b;
92     split_val(root, val, a, b);
93     a = merge(a, new node(val));
94     root = merge(a, b);
95 }
96
97 void del(T val)
98 {
99     node *a, *b, *c, *d;
100     split_val(root, val, a, b);
101     split_val(a, val - 1, c, d);
102     node* e = d;
103     d = merge(d->l, d->r);
104     delete e;
105     a = merge(c, d);
106     root = merge(a, b);
107 }
108
109 T getRank(T val)
110 {
111     node *a, *b;

```



```

112     split_val(root, val - 1, a, b);
113     T res = get_size(a) + 1;
114     root = merge(a, b);
115     return res;
116 }
117
118 T getKth(int k)
119 {
120     node* x = root;
121     T res = numeric_limits<T>::min();
122     while (x) {
123         if (k <= get_size(x->l))
124             x = x->l;
125         else {
126             if (get_size(x->l) + 1 == k) {
127                 res = x->v;
128                 break;
129             } else {
130                 k -= get_size(x->l) + 1;
131                 x = x->r;
132             }
133         }
134     }
135     return res;
136 }
137
138 T getPrev(T val)
139 {
140     node *a, *b;
141     split_val(root, val - 1, a, b);
142     node* p = a;
143     while (p->r)
144         p = p->r;
145     root = merge(a, b);
146     return p->v;
147 }
148
149 T getSucc(T val)
150 {
151     node *a, *b;
152     split_val(root, val, a, b);
153     node* p = b;
154     while (p->l)
155         p = p->l;
156     root = merge(a, b);
157     return p->v;
158 }
159 };

```

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

```

79     int u = rt;
80     while(true) {
81         if (k <= sz[L[u]]) u = L[u];
82         else {
83             if (sz[L[u]] + 1 == k) break;
84             else k -= sz[L[u]] + 1, u = R[u];
85         }
86     }
87     return v[u];
88 }
89
90 T getpre(T val) {
91     int x, y;
92     split(rt, val - 1, x, y);
93     int p = x;
94     while(R[p]) p = R[p];
95     rt = merge(x, y);
96     return v[p];
97 }
98
99 T getsuc(T val) {
100     int x, y;
101     split(rt, val, x, y);
102     int p = y;
103     while(L[p]) p = L[p];
104     rt = merge(x, y);
105     return v[p];
106 }
107
108 void DEBUG(int u) {
109     if (!u) return;
110     DEBUG(L[u]);
111     cerr << v[u] << " ";
112     DEBUG(R[u]);
113 }
114
115 void DEBUG() {
116     cerr << "Treap: ";
117     DEBUG(rt);
118     cerr << endl;
119 }
120 }

```

## 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): v(_v), nxt(_nxt) {}
9     };
10
11     int V, E, tot;
12     vector<int> h;
13     vector<Edge> e;
14
15     Graph() : V(0) {}
16     Graph(int _V, int _E) : V(_V), E(2 * _E), tot(0), h(_V + 1), e(2 * _E + 1) {}

```

```

17
18 inline void addarc(int u, int v) {
19     assert(1 <= u && u <= V);
20     assert(1 <= v && v <= V);
21
22     e[++tot] = Edge(v, h[u]); h[u] = tot;
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, bcc_cnt;
32 vector<int> dfn, low, belong, bcc_size;
33 vector<vector<int>> bcc;
34 vector<bool> bridge;
35
36 void tarjan(int u, int fa) {
37     dfn[u] = low[u] = ++bcc_clock;
38     fore(i, u) {
39         int v = e[i].v;
40         if (v == fa) continue;
41
42         if (!dfn[v]) {
43             tarjan(v, u);
44             low[u] = min(low[u], low[v]);
45             if (dfn[u] < low[v]) {
46                 bridge[i] = true;
47                 if (i & 1) bridge[i + 1] = true;
48                 else bridge[i - 1] = true;
49             }
50         } else if (dfn[v] < dfn[u]) {
51             low[u] = min(low[u], dfn[v]);
52         }
53     }
54 }
55
56 void blood_fill(int u) {
57     belong[u] = bcc_cnt; bcc[bcc_cnt].push_back(u);
58     fore(i, u) {
59         if (bridge[i]) continue;
60         int v = e[i].v;
61         if (!belong[v]) blood_fill(v);
62     }
63 }
64
65 void build_bcc_point() {
66     bcc_clock = bcc_cnt = 0;
67     dfn = vector<int>(V + 1);
68     low = vector<int>(V + 1);
69     belong = vector<int>(V + 1);
70     bridge = vector<bool>(E + 1);
71     bcc = vector<vector<int>>(1);
72
73     for (int i = 1; i <= V; ++i) {
74         if (!dfn[i]) {
75             tarjan(i, i);
76         }
77     }
78
79     for (int i = 1; i <= V; ++i) {
80         if (!belong[i]) {
81             ++bcc_cnt;

```

```

82         bcc.push_back(vector<int>());
83         blood_fill(i);
84     }
85 }
86
87 bcc_size = vector<int> (bcc_cnt + 1);
88 for (int i = 1; i <= bcc_cnt; ++i) bcc_size[i] = bcc[i].size();
89 }
90 };
91
92 }

```

## 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): u(_u), v(_v) {}
8     };
9
10    int V;
11    vector<vector<Edge>> G;
12
13    Graph() : V(0) {}
14    Graph(int _V) : V(_V), G(_V + 1) {}
15
16    inline void addarc(int u, int v) {
17        assert(1 <= u && u <= V);
18        assert(1 <= v && v <= V);
19        G[u].push_back(Edge(u, v));
20    }
21
22    inline void addedge(int u, int v) {
23        addarc(u, v);
24        addarc(v, u);
25    }
26
27    /*****
28    int bcc_clock;
29    vector<int> dfn, low;
30    vector<vector<int>> bcc;
31    vector<bool> cut;
32    stack<int> stk;
33
34    void tarjan(int u, int fa) {
35        dfn[u] = low[u] = ++bcc_clock; stk.push(u);
36
37        if (u == fa && G[u].empty()) {
38            vector<int> nb;
39            nb.push_back(u);
40            bcc.push_back(nb);
41            return;
42        }
43
44        int son = 0;
45        for (Edge& e: G[u]) {
46            int v = e.v;
47            if (v == fa) continue;
48
49            if (!dfn[v]) {
50                tarjan(v, u);

```

```

51         low[u] = min(low[u], low[v]);
52         if (dfn[u] <= low[v]) {
53             ++son;
54             if (u != fa || son > 1) cut[u] = true;
55             vector<int> nb;
56             int top;
57             do {
58                 top = stk.top(); stk.pop();
59                 nb.push_back(top);
60             } while(top != v);
61             nb.push_back(u);
62             bcc.push_back(nb);
63         }
64     } else low[u] = min(low[u], dfn[v]);
65 }
66 }
67
68 void build_bcc_point() {
69     bcc_clock = 0;
70     dfn = vector<int>(V + 1);
71     low = vector<int>(V + 1);
72     cut = vector<bool>(V + 1);
73     bcc = vector<vector<int>>(1);
74
75     for (int i = 1; i <= V; ++i) {
76         if (!dfn[i]) {
77             while(!stk.empty()) stk.pop();
78             tarjan(i, i);
79         }
80     }
81 }
82 };
83
84 }

```

## 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; nr = _nr;
14        G = vector<vector<int>>(nl + 1);
15    }
16
17    void addarc(int u, int v) {
18        G[u].push_back(v);
19    }
20
21    void addedge(int u, int v) {
22        G[u].push_back(v);
23        G[v].push_back(u);
24    }
25
26    bool bfs() {
27        queue<int> q;

```

```

28     bool res = false;
29
30     for (int i = 1; i <= n1; ++i) {
31         if (ml[i]) ll[i] = 0;
32         else ll[i] = 1, q.push(i);
33     }
34
35     for (int i = 1; i <= nr; ++i) lr[i] = 0;
36
37     while(!q.empty()) {
38         int u = q.front(); q.pop();
39         for (int v: G[u]) {
40             if (lr[v] == 0) {
41                 lr[v] = ll[u] + 1;
42                 if (mr[v]) {
43                     ll[mr[v]] = lr[v] + 1;
44                     q.push(mr[v]);
45                 } else res = true;
46             }
47         }
48     }
49
50     return res;
51 };
52
53 bool dfs(int u) {
54     for (int v: G[u]) {
55         if (lr[v] == ll[u] + 1 && vis[v] != dfn) {
56             vis[v] = dfn;
57             if (mr[v] == 0 || dfs(mr[v])) {
58                 mr[v] = u; ml[u] = v;
59                 return true;
60             }
61         }
62     }
63     return false;
64 };
65
66 int HK() {
67     ml = vector<int> (n1 + 1);
68     mr = vector<int> (nr + 1);
69     ll = vector<int> (n1 + 1);
70     lr = vector<int> (nr + 1);
71     vis = vector<int> (nr + 1);
72
73     int res = 0;
74     while(bfs()) {
75         ++dfn;
76         for (int i = 1; i <= n1; ++i)
77             if (!ml[i]) res += dfs(i);
78     }
79     return res;
80 }
81 };
82
83 /**
84  * 最小覆盖数 = 最大匹配数
85  * 最大独立集 = 顶点数 - 二分图匹配数
86  * DAG 最小路径覆盖数 = 结点数 - 拆点后二分图最大匹配数
87 */

```

```
// Kuhn Munkres,  $O(V^3)$ 
template<typename T>
struct biwraph {
    T TMAX, TMIN;

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

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

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

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

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

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

        return true;
    }

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



```

64         }
65     }
66 }
67
68
69 T a = TMAX;
70 for (int i = 1; i <= n; ++i) if (!visr[i]) a = min(a, slack[i]);
71 for (int i = 1; i <= n; ++i) {
72     if (visl[i]) highl[i] -= a;
73     if (visr[i]) highr[i] += a;
74     else slack[i] -= a;
75 }
76 for (int i = 1; i <= n; ++i)
77     if (!visr[i] && !slack[i] && check(i)) return;
78 }
79 }
80
81 void match() {
82     fill(highr.begin(), highr.end(), 0);
83     fill(matchl.begin(), matchl.end(), 0);
84     fill(matchr.begin(), matchr.end(), 0);
85     for (int i = 1; i <= n; ++i) highl[i] = *max_element(G[i].begin() + 1, G[i].end());
86
87     for (int i = 1; i <= n; ++i) {
88         fill(slack.begin(), slack.end(), TMAX);
89         fill(visl.begin(), visl.end(), false);
90         fill(visr.begin(), visr.end(), false);
91         bfs(i);
92     }
93 }
94
95 T getMaxMatch() {
96     T res = 0;
97     match();
98     for (int i = 1; i <= n; ++i) {
99         if (G[i][matchl[i]] > 0) res += G[i][matchl[i]];
100        else matchl[i] = 0;
101    }
102    return res;
103 }
104 };

```

## 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 low[u] = min(low[u], dfn[v]);
38 }
39 }
40
41 ll ans;
42 int sz[N];
43 void dfs(int u, int fa) {
44     sz[u] = (u <= n);
45     for (int v: F[u]) if (v != fa) {
46         dfs(v, u);
47         ans += 2ll * w[u] * sz[u] * sz[v];
48         sz[u] += sz[v];
49     }
50     ans += 2ll * w[u] * sz[u] * (cc - sz[u]);
51 }
52
53 void buildBlockForest() {
54     for (int i = 1; i <= n; ++i) if (!dfn[i]) {
55         cc = 0;
56         tarjan(i);
57         --top;
58         dfs(i, i);
59     }
60 }
61
62 void solve(int Case) {
63     scanf("%d %d", &n, &m);
64     fill(w + 1, w + 1 + n, -1);
65     int u, v;
66     for (int i = 1; i <= m; ++i) {
67         scanf("%d %d", &u, &v);
68         G[u].push_back(v);
69         G[v].push_back(u);
70     }
71     buildBlockForest();
72     printf("%lld\n", ans);
73 }
74
75 int main () {
76     int T = 1;
77     // scanf("%d", &T);
78     for (int i = 1; i <= T; ++i) solve(i);
79     return 0;
80 }

```

## 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 {
13     int v, nxt;
14 } e[ N << 1 ];
15 void init( int n )
16 {
17     tot = 0;
18     for ( int i = 1; i <= n; ++i )
19         head[ i ] = 0;
20 }
21 void add( int u, int v )
22 {
23     ++tot;
24     e[ tot ] = ( edge ){ v, head[ u ] };
25     head[ u ] = tot;
26 }
27 #define fore( i, u ) for ( int i = head[ u ]; i; i = e[ i ].nxt )
28
29 int sz[ N ], son[ N ], f[ N ], h[ N ], top[ N ];
30
31 void dfs1( int u, int fa )
32 {
33     f[ u ] = fa;
34     h[ u ] = h[ fa ] + 1;
35     sz[ u ] = 1;
36     son[ u ] = 0;
37     fore( i, u )
38     {
39         int v = e[ i ].v;
40         if ( v == fa )
41             continue;
42         dfs1( v, u );
43         sz[ u ] += sz[ v ];
44         if ( sz[ v ] > sz[ son[ u ] ] )
45             son[ u ] = v;
46     }
47 }
48
49 void dfs2( int u, int fa, int k )
50 {
51     top[ u ] = k;
52     if ( son[ u ] )
53         dfs2( son[ u ], u, k );
54     fore( i, u )
55     {
56         int v = e[ i ].v;
57         if ( v == fa || v == son[ u ] )
58             continue;
59         dfs2( v, u, v );
60     }
61 }
62
63 int lca( int u, int v )

```

```

64 {
65     while ( top[ u ] != top[ v ] )
66     {
67         if ( h[ top[ u ] ] < h[ top[ v ] ] )
68             swap( u, v );
69         u = f[ top[ u ] ];
70     }
71     if ( h[ u ] > h[ v ] )
72         swap( u, v );
73     return u;
74 }
75
76 int dep[ N ], max_dep[ N ], pa[ N ];
77 int key_cnt, keyid[ N ];
78
79 const int COLORCNT = 4e4 + 2;
80 const int KEYCNT   = 101;
81 const int gap      = 400;
82
83 bitset< COLORCNT > c[ KEYCNT ][ KEYCNT ];
84
85 int stk[ N ], tp;
86
87 void dfs_key( int u, int fa )
88 {
89     dep[ u ] = dep[ fa ] + 1;
90     max_dep[ u ] = dep[ u ];
91     fore( i, u )
92     {
93         int v = e[ i ].v;
94         if ( v == fa )
95             continue;
96         dfs_key( v, u );
97         if ( max_dep[ v ] > max_dep[ u ] )
98             max_dep[ u ] = max_dep[ v ];
99     }
100     if ( max_dep[ u ] - dep[ u ] >= gap )
101     {
102         keyid[ u ] = ++key_cnt;
103         max_dep[ u ] = dep[ u ];
104     }
105 }
106
107 void dfs_bitset( int u )
108 {
109     if ( keyid[ u ] && u != stk[ tp ] )
110     {
111         for ( int x = u; x != stk[ tp ]; x = f[ x ] )
112             c[ keyid[ stk[ tp ] ] ][ keyid[ u ] ].set( a[ x ] );
113
114         for ( int i = 1; i < tp; ++i )
115         {
116             c[ keyid[ stk[ i ] ] ][ keyid[ u ] ] = c[ keyid[ stk[ i ] ] ][ keyid[ stk[ tp ] ] ];
117             c[ keyid[ stk[ i ] ] ][ keyid[ u ] ] |= c[ keyid[ stk[ tp ] ] ][ keyid[ u ] ];
118         }
119         pa[ u ] = stk[ tp ];
120         stk[ ++tp ] = u;
121     }
122     for ( int i = head[ u ]; i; i = e[ i ].nxt )
123     {
124         if ( e[ i ].v != f[ u ] )
125             dfs_bitset( e[ i ].v );
126     }
127     if ( keyid[ u ] )
128         --tp;

```

```

129 }
130
131 void build_block_tree()
132 {
133     key_cnt = 0;
134     dfs_key( 1, 1 );
135     if ( !keyid[ 1 ] )
136         keyid[ 1 ] = ++key_cnt;
137
138     tp      = 1;
139     stk[ 1 ] = 1;
140     dfs_bitset( 1 );
141 }
142
143 bitset< COLORCNT > res;
144
145 int query( int u, int v )
146 {
147     res.reset();
148     int uv = lca( u, v );
149
150     // step 1: jump to nearest key node
151     while ( u != uv && !keyid[ u ] )
152     {
153         res.set( a[ u ] );
154         u = f[ u ];
155     }
156     while ( v != uv && !keyid[ v ] )
157     {
158         res.set( a[ v ] );
159         v = f[ v ];
160     }
161
162     // step 2: jump to Lowest key node
163     int pu = u;
164     while ( dep[ pa[ pu ] ] >= dep[ uv ] )
165         pu = pa[ pu ];
166     if ( pu != u )
167     {
168         res |= c[ keyid[ pu ] ][ keyid[ u ] ];
169         u = pu;
170     }
171
172     int pv = v;
173     while ( dep[ pa[ pv ] ] >= dep[ uv ] )
174         pv = pa[ pv ];
175     if ( pv != v )
176     {
177         res |= c[ keyid[ pv ] ][ keyid[ v ] ];
178         v = pv;
179     }
180
181     // step 3: jump to lca
182     while ( u != uv )
183     {
184         res.set( a[ u ] );
185         u = f[ u ];
186     }
187     while ( v != uv )
188     {
189         res.set( a[ v ] );
190         v = f[ v ];
191     }
192
193     // step 4: set lca

```

```

194     res.set( a[ uv ] );
195
196     return res.count();
197 }
198
199 void solve( int Case )
200 {
201     scanf( "%d %d", &n, &m );
202     for ( int i = 1; i <= n; ++i )
203     {
204         scanf( "%d", &a[ i ] );
205         t[ i ] = a[ i ];
206     }
207
208     sort( t + 1, t + 1 + n );
209     nt = unique( t + 1, t + 1 + n ) - ( t + 1 );
210
211     for ( int i = 1; i <= n; ++i )
212         a[ i ] = lower_bound( t + 1, t + 1 + nt, a[ i ] ) - t;
213
214     init( n );
215     int u, v;
216     for ( int i = 1; i <= n - 1; ++i )
217     {
218         scanf( "%d %d", &u, &v );
219         add( u, v );
220         add( v, u );
221     }
222
223     dfs1( 1, 1 );
224     dfs2( 1, 1, 1 );
225
226     build_block_tree();
227
228     int lastans = 0;
229     for ( int i = 1; i <= m; ++i )
230     {
231         scanf( "%d %d", &u, &v );
232         u ^= lastans;
233         lastans = query( u, v );
234         printf( "%d\n", lastans );
235     }
236 }
237
238 int main()
239 {
240     int T = 1;
241     // scanf( "%d", &T );
242     for ( int _ = 1; _ <= T; _++ )
243         solve( _ );
244     return 0;
245 }

```

## 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): u(_u), v(_v), w(_w) {}

```

```

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

```

## 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; son[u] = -1;
15     for (int v: G[u]) {
16         if (v == fa) continue;

```

```

17     dfs1(v, u);
18     sz[u] += sz[v];
19     if (sz[v] > sz[son[u]]) son[u] = v;
20 }
21 }
22
23 void add(int u, int fa, int Son, int d) {
24     // update data here
25     cnt[color[u]] += d;
26     if (cnt[color[u]] > ma) ma = cnt[color[u]], cur = 0;
27     if (cnt[color[u]] == ma) cur += color[u];
28
29     for (int v: G[u]) {
30         if (v == fa || v == Son) continue;
31         add(v, u, Son, d);
32     }
33 }
34
35 void dfs2(int u, int fa, bool keep) {
36     for (int v: G[u]) {
37         if (v == fa || v == son[u]) continue;
38         dfs2(v, u, false);
39     }
40     if (son[u] != -1) dfs2(son[u], u, true);
41
42     add(u, fa, son[u], 1);
43
44     // answer queries here
45     ans[u] = cur;
46
47     if (!keep) {
48         add(u, fa, -1, -1);
49         ma = 0; cur = 0;
50     }
51 }
52
53 void solve() {
54     read(n);
55     FOR(i, 1, n) read(color[i]);
56
57     int u, v;
58     FOR(i, 2, n) {
59         read(u, v);
60         addedge(u, v);
61     }
62
63     dfs1(1, 0);
64     dfs2(1, 0, 0);
65
66     FOR(i, 1, n - 1) printf("%lld ", ans[i]);
67     println(ans[n]);
68 }

```

## 2.9 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() : x(123456789), y(362436069), z(521288629), w(88675123) {}
8     unsigned next() {
9         unsigned t = x ^ (x << 11);

```



```

10     x = y;
11     y = z;
12     z = w;
13     return w = w ^ (w >> 19) ^ (t ^ (t >> 8));
14 }
15 //手回き
16 inline unsigned next(unsigned n) { return next() % n; }
17 };
18
19 // bottom up な Treap
20 //脱再!
21 // randomized binary search するには choiceRandomly を
22 // bool choiceRandomly(Ref l, Ref r) { return rng.next(l->size + r->size) < l->size; }
23 //に書き込めるだけでよい。
24 template <typename Node>
25 struct BottomupTreap {
26     Xor128 rng;
27     typedef Node *Ref;
28     static int size(Ref t) { return !t ? 0 : t->size; }
29
30     unsigned nextRand() { return rng.next(); }
31
32 private:
33     bool choiceRandomly(Ref l, Ref r) { return l->priority < r->priority; }
34
35 public:
36     Ref join(Ref l, Ref r) {
37         if (!l)
38             return r;
39         if (!r)
40             return l;
41
42         Ref t = NULL;
43         unsigned long long dirs = 0;
44         int h;
45         for (h = 0;; ++h) {
46             if (h >= sizeof(dirs) * 8 - 2) {
47                 // dirs のオーバーフローを防ぐために再帰する。
48                 //あくまでセーフティガードなのでバランスは多少崩れるかもしれない
49                 t = join(l->right, r->left);
50                 dirs = dirs << 2 | 1;
51                 h++;
52                 break;
53             }
54             dirs <<= 1;
55             if (choiceRandomly(l, r)) {
56                 Ref c = l->right;
57                 if (!c) {
58                     t = r;
59                     r = r->parent;
60                     break;
61                 }
62                 l = c;
63             } else {
64                 dirs |= 1;
65                 Ref c = r->left;
66                 if (!c) {
67                     t = l;
68                     l = l->parent;
69                     break;
70                 }
71                 r = c;
72             }
73         }
74         for (; h >= 0; --h) {

```

```

75         if (!(dirs & 1)) {
76             Ref p = l->parent;
77             t = l->linkr(t);
78             l = p;
79         } else {
80             Ref p = r->parent;
81             t = r->linkl(t);
82             r = p;
83         }
84         dirs >>= 1;
85     }
86     return t;
87 }
88
89 typedef std::pair<Ref, Ref> RefPair;
90
91 // L < t < r の (L,r) に分割する
92 RefPair split2(Ref t) {
93     Ref p, l = t->left, r = t;
94     Node::cut(l);
95     t->linkl(NULL);
96     while (p = t->parent) {
97         t->parent = NULL;
98         if (p->left == t)
99             r = p->linkl(r);
100         else
101             l = p->linkr(l);
102         t = p;
103     }
104     return RefPair(l, r);
105 }
106 // L < t < r の (L,t,r) に分割する。(L,r) を返す
107 RefPair split3(Ref t) {
108     Ref p, l = t->left, r = t->right;
109     Node::cut(l), Node::cut(r);
110     t->linklr(NULL, NULL);
111     while (p = t->parent) {
112         t->parent = NULL;
113         if (p->left == t)
114             r = p->linkl(r);
115         else
116             l = p->linkr(l);
117         t = p;
118     }
119     return RefPair(l, r);
120 }
121 Ref cons(Ref h, Ref t) {
122     assert(size(h) == 1);
123     if (!t)
124         return h;
125     Ref u = NULL;
126     while (true) {
127         if (choiceRandomly(h, t)) {
128             Ref p = t->parent;
129             u = h->linkr(t);
130             t = p;
131             break;
132         }
133         Ref l = t->left;
134         if (!l) {
135             u = h;
136             break;
137         }
138         t = l;
139     }

```

```

140     while (t) {
141         u = t->linkl(u);
142         t = t->parent;
143     }
144     return u;
145 }
146 };
147
148 // free tree のために、 $\mathbb{F}$ を基本として $\mathbb{F}$ う
149 class EulerTourTreeWithMarks {
150     struct Node {
151         typedef BottomupTreap<Node> BST;
152
153         Node *left, *right, *parent;
154         int size;
155         unsigned priority;
156         char marks, markUnions; // 0 ビット目が edgeMark, 1 ビット目が vertexMark
157
158         Node() : left(NULL), right(NULL), parent(NULL), size(1), priority(0), marks(0), markUnions(0) {}
159
160         inline Node *update() {
161             int size_t = 1, markUnions_t = marks;
162             if (left) {
163                 size_t += left->size;
164                 markUnions_t |= left->markUnions;
165             }
166             if (right) {
167                 size_t += right->size;
168                 markUnions_t |= right->markUnions;
169             }
170             size = size_t, markUnions = markUnions_t;
171             return this;
172         }
173
174         inline Node *linkl(Node *c) {
175             if (left = c)
176                 c->parent = this;
177             return update();
178         }
179         inline Node *linkr(Node *c) {
180             if (right = c)
181                 c->parent = this;
182             return update();
183         }
184         inline Node *linklr(Node *l, Node *r) {
185             if (left = l)
186                 l->parent = this;
187             if (right = r)
188                 r->parent = this;
189             return update();
190         }
191         static Node *cut(Node *t) {
192             if (t)
193                 t->parent = NULL;
194             return t;
195         }
196
197         static const Node *findRoot(const Node *t) {
198             while (t->parent) t = t->parent;
199             return t;
200         }
201         static std::pair<Node *, int> getPosition(Node *t) {
202             int k = BST::size(t->left);
203             Node *p;
204             while (p = t->parent) {

```

```

205         if (p->right == t)
206             k += BST::size(p->left) + 1;
207         t = p;
208     }
209     return std::make_pair(t, k);
210 }
211 static const Node *findHead(const Node *t) {
212     while (t->left) t = t->left;
213     return t;
214 }
215 static void updatePath(Node *t) {
216     while (t) {
217         t->update();
218         t = t->parent;
219     }
220 }
221 };
222
223 typedef Node::BST BST;
224 BST bst;
225
226 std::vector<Node> nodes;
227 //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)
228 //逆に arc に④して④④する頂点はたかだか 1 つである
229 std::vector<int> firstArc;
230 //④・頂点に④する属性
231 std::vector<bool> edgeMark, vertexMark;
232
233 inline int getArcIndex(const Node *a) const { return a - &nodes[0]; }
234
235 inline int arc1(int ei) const { return ei; }
236 inline int arc2(int ei) const { return ei + (numVertices() - 1); }
237
238 public:
239     inline int numVertices() const { return firstArc.size(); }
240     inline int numEdges() const { return numVertices() - 1; }
241
242     inline bool getEdgeMark(int a) const { return a < numEdges() ? edgeMark[a] : false; }
243     inline bool getVertexMark(int v) const { return vertexMark[v]; }
244
245 private:
246     void updateMarks(int a, int v) {
247         Node *t = &nodes[a];
248         t->marks = getEdgeMark(a) << 0 | getVertexMark(v) << 1;
249         Node::updatePath(t);
250     }
251
252     // firstArc の④更に④じて更新する
253     void firstArcChanged(int v, int a, int b) {
254         if (a != -1)
255             updateMarks(a, v);
256         if (b != -1)
257             updateMarks(b, v);
258     }
259
260 public:
261     class TreeRef {
262     friend class EulerTourTreeWithMarks;
263     const Node *ref;
264
265     public:
266         TreeRef() {}
267         TreeRef(const Node *ref_) : ref(ref_) {}
268         bool operator==(const TreeRef &that) const { return ref == that.ref; }
269         bool operator!=(const TreeRef &that) const { return ref != that.ref; }

```

```

270     bool isIsolatedVertex() const { return ref == NULL; }
271 };
272
273 void init(int N) {
274     int M = N - 1;
275     firstArc.assign(N, -1);
276     nodes.assign(M * 2, Node());
277     for (int i = 0; i < M * 2; i++) nodes[i].priority = bst.nextRand();
278     edgeMark.assign(M, false);
279     vertexMark.assign(N, false);
280 }
281
282 TreeRef getTreeRef(int v) const {
283     int a = firstArc[v];
284     return TreeRef(a == -1 ? NULL : Node::findRoot(&nodes[a]));
285 }
286
287 bool isConnected(int v, int w) const {
288     if (v == w)
289         return true;
290     int a = firstArc[v], b = firstArc[w];
291     if (a == -1 || b == -1)
292         return false;
293     return Node::findRoot(&nodes[a]) == Node::findRoot(&nodes[b]);
294 }
295
296 static int getSize(TreeRef t) {
297     if (t.isIsolatedVertex())
298         return 1;
299     else
300         return t.ref->size / 2 + 1;
301 }
302
303 void link(int ti, int v, int w) {
304     int a1 = arc1(ti), a2 = arc2(ti);
305     // v→w が a1 にⒺⒺするようにする
306     if (v > w)
307         std::swap(a1, a2);
308
309     int va = firstArc[v], wa = firstArc[w];
310
311     Node *l, *m, *r;
312     if (va != -1) {
313         // event. 順番を入れ替えるだけ
314         std::pair<Node *, Node *> p = bst.split2(&nodes[va]);
315         m = bst.join(p.second, p.first);
316     } else {
317         // v が孤立点の場合
318         m = NULL;
319         firstArc[v] = a1;
320         firstArcChanged(v, -1, a1);
321     }
322     if (wa != -1) {
323         std::pair<Node *, Node *> p = bst.split2(&nodes[wa]);
324         l = p.first, r = p.second;
325     } else {
326         // w が孤立点の場合
327         l = r = NULL;
328         firstArc[w] = a2;
329         firstArcChanged(w, -1, a2);
330     }
331     // w→v のⒺを m の先頭 = l の末尾に insert
332     m = bst.cons(&nodes[a2], m);
333     // v→w のⒺを m の末尾 = r の先頭に insert
334     r = bst.cons(&nodes[a1], r);

```

```

335     bst.join(bst.join(l, m), r);
336 }
337
338
339 void cut(int ti, int v, int w) {
340     // v→w が a1 に繋がるようにする
341     if (v > w)
342         std::swap(v, w);
343
344     int a1 = arc1(ti), a2 = arc2(ti);
345     std::pair<Node *, Node *> p = bst.split3(&nodes[a1]);
346     int prsize = BST::size(p.second);
347     std::pair<Node *, Node *> q = bst.split3(&nodes[a2]);
348     Node *l, *m, *r;
349     // a1, a2 の順番を判定する。a1 < a2 なら p.second が繋がっているはず
350     if (p.second == &nodes[a2] || BST::size(p.second) != prsize) {
351         l = p.first, m = q.first, r = q.second;
352     } else {
353         // a2 < a1 の順番である。v→w の繋がり方が a1 であって親 → 子であることにする
354         std::swap(v, w);
355         std::swap(a1, a2);
356         l = q.first, m = q.second, r = p.second;
357     }
358
359     // firstArc を必要に応じて書き換える
360     if (firstArc[v] == a1) {
361         int b;
362         if (r != NULL) {
363             // v が根じゃないなら右側の最初の繋がり方
364             b = getArcIndex(Node::findHead(r));
365         } else {
366             // v が根なら最初の繋がり方。孤立点になるなら -1
367             b = !l ? -1 : getArcIndex(Node::findHead(l));
368         }
369         firstArc[v] = b;
370         firstArcChanged(v, a1, b);
371     }
372     if (firstArc[w] == a2) {
373         // w が根になるので最初の繋がり方。孤立点になるなら -1
374         int b = !m ? -1 : getArcIndex(Node::findHead(m));
375         firstArc[w] = b;
376         firstArcChanged(w, a2, b);
377     }
378
379     bst.join(l, r);
380 }
381
382 void changeEdgeMark(int ti, bool b) {
383     assert(ti < numEdges());
384     edgeMark[ti] = b;
385     Node *t = &nodes[ti];
386     t->marks = (b << 0) | (t->marks & (1 << 1));
387     Node::updatePath(t);
388 }
389
390 void changeVertexMark(int v, bool b) {
391     vertexMark[v] = b;
392     int a = firstArc[v];
393     if (a != -1) {
394         Node *t = &nodes[a];
395         t->marks = (t->marks & (1 << 0)) | (b << 1);
396         Node::updatePath(t);
397     }
398 }
399
400 template <typename Callback>

```

```

400     bool enumMarkedEdges(TreeRef tree, Callback callback) const {
401         return enumMarks<0, Callback>(tree, callback);
402     }
403     //孤立点の場合は呼び側でその頂点だけ管理する必要がある
404     template <typename Callback>
405     bool enumMarkedVertices(TreeRef tree, Callback callback) const {
406         return enumMarks<1, Callback>(tree, callback);
407     }
408
409 private:
410     // callback : TreeEdgeIndex*2 -> Bool
411     // 引数は頂点をそこからの incident arc で示し、"(正方向 ? 0 : N-1) +
412     // treeEdgeIndex" を表す。方向は v,w の大小で管理すればよい
413     // callback は管理するかどうかを bool で返す。最後まで列挙し終えたかどうかを返す。
414     template <int Mark, typename Callback>
415     bool enumMarks(TreeRef tree, Callback callback) const {
416         if (tree.isIsolatedVertex())
417             return true;
418         const Node *t = tree.ref;
419         if (t->markUnions >> Mark & 1)
420             return enumMarksRec<Mark, Callback>(t, callback);
421         else
422             return true;
423     }
424
425     //平衡木なので深さは深くないので再帰して問題ない
426     template <int Mark, typename Callback>
427     bool enumMarksRec(const Node *t, Callback callback) const {
428         const Node *l = t->left, *r = t->right;
429         if (l && (l->markUnions >> Mark & 1))
430             if (!enumMarksRec<Mark, Callback>(l, callback))
431                 return false;
432         if (t->marks >> Mark & 1)
433             if (!callback(getArcIndex(t)))
434                 return false;
435         if (r && (r->markUnions >> Mark & 1))
436             if (!enumMarksRec<Mark, Callback>(r, callback))
437                 return false;
438         return true;
439     }
440
441 public:
442     //デバッグ用
443     void debugEnumEdges(std::vector<int> &out_v) const {
444         int M = numEdges();
445         for (int ti = 0; ti < M; ti++) {
446             const Node *t = &nodes[ti];
447             if (t->left || t->right || t->parent)
448                 out_v.push_back(ti);
449         }
450     }
451 };
452
453 // 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。
454 //ところで"Level up" って和英語なんだ。promote でいいかな。
455 // Sampling heuristic ランダム探索で超速く (4 倍とか) なったんだけど! いいね!
456 //
457 // References
458 // · Holm, Jacob, Kristian De Lichtenberg, and Mikkel Thorup. "Poly-Logarithmic deterministic fully-dynamic
459 // algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity." Journal of the ACM
460 // (JACM) 48.4 (2001): 723-760. · Iyer, Raj, et al. "An experimental study of polylogarithmic, fully dynamic,
461 // connectivity algorithms." Journal of Experimental Algorithmics (JEA) 6 (2001): 4.
462
463 class HolmDeLichtenbergThorup {
464     typedef HolmDeLichtenbergThorup This;

```

```

465 typedef EulerTourTreeWithMarks Forest;
466 typedef Forest::TreeRef TreeRef;
467
468 int numVertices_m;
469 int numSamplings;
470
471 // 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...
472 std::vector<Forest> forests;
473
474 std::vector<char> edgeLevel;
475 std::vector<int> treeEdgeIndex;           // : EdgeIndex -> TreeEdgeIndex
476 std::vector<int> treeEdgeMap;           // : TreeEdgeIndex -> EdgeIndex
477 std::vector<int> treeEdgeIndexFreeList; // : [TreeEdgeIndex]
478
479 // arc も方向は EulerTourTree と同じように v,w の大小に合わせる
480 std::vector<int> arcHead;
481
482 std::vector<std::vector<int>> firstIncidentArc;
483 std::vector<int> nextIncidentArc, prevIncidentArc;
484
485 //一時的に使う。使い回して使う
486 std::vector<bool> edgeVisited;
487 std::vector<int> visitedEdges; // : [EdgeIndex | TreeEdgeIndex]
488
489 int arc1(int ei) const { return ei; }
490 int arc2(int ei) const { return numMaxEdges() + ei; }
491 int arcEdge(int i) const { return i >= numMaxEdges() ? i - numMaxEdges() : i; }
492
493 bool replace(int lv, int v, int w) {
494     Forest &forest = forests[lv];
495
496     TreeRef vRoot = forest.getTreeRef(v), wRoot = forest.getTreeRef(w);
497     assert(vRoot.isIsolatedVertex() || wRoot.isIsolatedVertex() || vRoot != wRoot);
498
499     int vSize = forest.getSize(vRoot), wSize = forest.getSize(wRoot);
500
501     int u;
502     TreeRef uRoot;
503     int uSize;
504     if (vSize <= wSize)
505         u = v, uRoot = vRoot, uSize = vSize;
506     else
507         u = w, uRoot = wRoot, uSize = wSize;
508
509     // replacement edge を探す
510     int replacementEdge = -1;
511     enumIncidentArcs(forest, uRoot, u, lv, FindReplacementEdge(uRoot, &replacementEdge));
512
513     // "Sampling heuristic"
514     //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
515     if (replacementEdge != -1 && (int)visitedEdges.size() + 1 <= numSamplings) {
516         // replacementEdge を削除する
517         deleteNontreeEdge(replacementEdge);
518         addTreeEdge(replacementEdge);
519         for (int i = 0; i < (int)visitedEdges.size(); i++) edgeVisited[visitedEdges[i]] = false;
520         visitedEdges.clear();
521         return true;
522     }
523
524     //見つけた incident arcs を一箇所にレベルアップさせる。edgeVisited の後処理もする
525     for (int i = 0; i < (int)visitedEdges.size(); i++) {
526         int ei = visitedEdges[i];
527         edgeVisited[ei] = false;
528
529         deleteNontreeEdge(ei);

```



```

530         ++edgeLevel[ei];
531
532         insertNontreeEdge(ei);
533     }
534     visitedEdges.clear();
535
536     //このレベルの  $T_u$  の  $E$  を列  $E$  する
537     forest.enumMarkedEdges(uRoot, EnumLevelTreeEdges(this));
538     //列  $E$  した  $T_u$  の  $E$  を一  $E$  にレベルアップさせる
539     for (int i = 0; i < (int)visitedEdges.size(); i++) {
540         int ti = visitedEdges[i];
541
542         int ei = treeEdgeMap[ti];
543         int v = arcHead[arc2(ei)], w = arcHead[arc1(ei)];
544         int lv = edgeLevel[ei];
545
546         edgeLevel[ei] = lv + 1;
547
548         forests[lv].changeEdgeMark(ti, false);
549         forests[lv + 1].changeEdgeMark(ti, true);
550
551         forests[lv + 1].link(ti, v, w);
552     }
553     visitedEdges.clear();
554
555     if (replacementEdge != -1) {
556         //  $T_u$  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
557         deleteNontreeEdge(replacementEdge);
558         addTreeEdge(replacementEdge);
559         return true;
560     } else if (lv > 0) {
561         return replace(lv - 1, v, w);
562     } else {
563         return false;
564     }
565 }
566
567 struct EnumLevelTreeEdges {
568     This *thisp;
569     EnumLevelTreeEdges(This *thisp_) : thisp(thisp_) {}
570
571     inline bool operator()(int a) {
572         thisp->enumLevelTreeEdges(a);
573         return true;
574     }
575 };
576
577 void enumLevelTreeEdges(int ti) { visitedEdges.push_back(ti); }
578
579 //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  $E$ 
580 template <typename Callback>
581 bool enumIncidentArcs(Forest &forest, TreeRef t, int u, int lv, Callback callback) {
582     if (t.isIsolatedVertex())
583         return enumIncidentArcsWithVertex<Callback>(lv, u, callback);
584     else
585         return forest.enumMarkedVertices(t, EnumIncidentArcs<Callback>(this, lv, callback));
586 }
587
588 template <typename Callback>
589 struct EnumIncidentArcs {
590     This *thisp;
591     int lv;
592     Callback callback;
593
594     EnumIncidentArcs(This *thisp_, int lv_, Callback callback_)

```

```

595         : thisp(thisp_), lv(lv_), callback(callback_) {}
596
597     inline bool operator()(int tii) const {
598         return thisp->enumIncidentArcsWithTreeArc(tii, lv, callback);
599     }
600 };
601
602 template <typename Callback>
603 bool enumIncidentArcsWithTreeArc(int tii, int lv, Callback callback) {
604     bool dir = tii >= numVertices() - 1;
605     int ti = dir ? tii - (numVertices() - 1) : tii;
606     int ei = treeEdgeMap[ti];
607     int v = arcHead[arc2(ei)], w = arcHead[arc1(ei)];
608     //方向を求め、その arc の tail の頂点を取得する
609     int u = !(dir != (v > w)) ? v : w;
610
611     return enumIncidentArcsWithVertex(lv, u, callback);
612 }
613
614 // 1 つの頂点を図理する
615 template <typename Callback>
616 bool enumIncidentArcsWithVertex(int lv, int u, Callback callback) {
617     int it = firstIncidentArc[lv][u];
618     while (it != -1) {
619         if (!callback(this, it))
620             return false;
621         it = nextIncidentArc[it];
622     }
623     return true;
624 }
625
626 struct FindReplacementEdge {
627     TreeRef uRoot;
628     int *replacementEdge;
629     FindReplacementEdge(TreeRef uRoot_, int *replacementEdge_)
630         : uRoot(uRoot_), replacementEdge(replacementEdge_) {}
631
632     inline bool operator()(This *thisp, int a) const {
633         return thisp->findReplacementEdge(a, uRoot, replacementEdge);
634     }
635 };
636
637 // 1 つの arc を図理する
638 bool findReplacementEdge(int a, TreeRef uRoot, int *replacementEdge) {
639     int ei = arcEdge(a);
640     if (edgeVisited[ei])
641         return true;
642
643     int lv = edgeLevel[ei];
644     TreeRef hRoot = forests[lv].getTreeRef(arcHead[a]);
645
646     if (hRoot.isIsolatedVertex() || hRoot != uRoot) {
647         //図の木に渡されているなら replacement edge である。
648         *replacementEdge = ei;
649         return false;
650     }
651     // replacement edge は visitedEdges に入れたくないのでこの位置でマ図クする
652     edgeVisited[ei] = true;
653     visitedEdges.push_back(ei);
654     return true;
655 }
656
657 void addTreeEdge(int ei) {
658     int v = arcHead[arc2(ei)], w = arcHead[arc1(ei)];
659     int lv = edgeLevel[ei];

```

```

660
661     int ti = treeEdgeIndexFreelist.back();
662     treeEdgeIndexFreelist.pop_back();
663     treeEdgeIndex[ei] = ti;
664     treeEdgeMap[ti] = ei;
665
666     forests[lv].changeEdgeMark(ti, true);
667
668     for (int i = 0; i <= lv; i++) forests[i].link(ti, v, w);
669 }
670
671 void insertIncidentArc(int a, int v) {
672     int ei = arcEdge(a);
673     int lv = edgeLevel[ei];
674     assert(treeEdgeIndex[ei] == -1);
675
676     int next = firstIncidentArc[lv][v];
677     firstIncidentArc[lv][v] = a;
678     nextIncidentArc[a] = next;
679     prevIncidentArc[a] = -1;
680     if (next != -1)
681         prevIncidentArc[next] = a;
682
683     if (next == -1)
684         forests[lv].changeVertexMark(v, true);
685 }
686
687 void deleteIncidentArc(int a, int v) {
688     int ei = arcEdge(a);
689     int lv = edgeLevel[ei];
690     assert(treeEdgeIndex[ei] == -1);
691
692     int next = nextIncidentArc[a], prev = prevIncidentArc[a];
693     nextIncidentArc[a] = prevIncidentArc[a] = -2;
694
695     if (next != -1)
696         prevIncidentArc[next] = prev;
697     if (prev != -1)
698         nextIncidentArc[prev] = next;
699     else
700         firstIncidentArc[lv][v] = next;
701
702     if (next == -1 && prev == -1)
703         forests[lv].changeVertexMark(v, false);
704 }
705
706 void insertNontreeEdge(int ei) {
707     int a1 = arc1(ei), a2 = arc2(ei);
708     insertIncidentArc(a1, arcHead[a2]);
709     insertIncidentArc(a2, arcHead[a1]);
710 }
711
712 void deleteNontreeEdge(int ei) {
713     int a1 = arc1(ei), a2 = arc2(ei);
714     deleteIncidentArc(a1, arcHead[a2]);
715     deleteIncidentArc(a2, arcHead[a1]);
716 }
717
718 public:
719     HolmDeLichtenbergThorup() : numVertices_m(0), numSamplings(0) {}
720
721     int numVertices() const { return numVertices_m; }
722     int numMaxEdges() const { return edgeLevel.size(); }
723
724     void init(int N, int M) {

```

```

725     numVertices_m = N;
726
727     int levels = 1;
728     while (1 << levels <= N / 2) levels++;
729
730     //サンプリング数を設定する。適切な $\square$ はよくわからない
731     numSamplings = (int)(levels * 1);
732
733     forests.resize(levels);
734     for (int lv = 0; lv < levels; lv++) forests[lv].init(N);
735
736     edgeLevel.assign(M, -1);
737
738     treeEdgeIndex.assign(M, -1);
739     treeEdgeMap.assign(N - 1, -1);
740
741     treeEdgeIndexFreeList.resize(N - 1);
742     for (int ti = 0; ti < N - 1; ti++) treeEdgeIndexFreeList[ti] = ti;
743
744     arcHead.assign(M * 2, -1);
745
746     firstIncidentArc.resize(levels);
747     for (int lv = 0; lv < levels; lv++) firstIncidentArc[lv].assign(N, -1);
748     nextIncidentArc.assign(M * 2, -2);
749     prevIncidentArc.assign(M * 2, -2);
750
751     edgeVisited.assign(M, false);
752 }
753
754 bool insertEdge(int ei, int v, int w) {
755     if (!(0 <= ei && ei < numMaxEdges() && 0 <= v && v < numVertices() && 0 <= w && w < numVertices())) {
756         system("pause");
757     }
758     assert(0 <= ei && ei < numMaxEdges() && 0 <= v && v < numVertices() && 0 <= w && w < numVertices());
759     assert(edgeLevel[ei] == -1);
760
761     int a1 = arc1(ei), a2 = arc2(ei);
762     arcHead[a1] = w, arcHead[a2] = v;
763
764     bool treeEdge = !forests[0].isConnected(v, w);
765
766     edgeLevel[ei] = 0;
767     if (treeEdge) {
768         addTreeEdge(ei);
769     } else {
770         treeEdgeIndex[ei] = -1;
771         //ル $\square$ は見たくないのでリストにも入れない
772         if (v != w)
773             insertNontreeEdge(ei);
774     }
775
776     return treeEdge;
777 }
778
779 bool deleteEdge(int ei) {
780     assert(0 <= ei && ei < numMaxEdges() && edgeLevel[ei] != -1);
781
782     int a1 = arc1(ei), a2 = arc2(ei);
783     int v = arcHead[a2], w = arcHead[a1];
784
785     int lv = edgeLevel[ei];
786     int ti = treeEdgeIndex[ei];
787
788     bool splitted = false;
789     if (ti != -1) {

```

```

790         treeEdgeMap[ti] = -1;
791         treeEdgeIndex[ei] = -1;
792         treeEdgeIndexFreeList.push_back(ti);
793
794         for (int i = 0; i <= lv; i++) forests[i].cut(ti, v, w);
795
796         forests[lv].changeEdgeMark(ti, false);
797
798         splitted = !replace(lv, v, w);
799     } else {
800         //ルEはリストに入っていない
801         if (v != w)
802             deleteNontreeEdge(ei);
803     }
804
805     arcHead[a1] = arcHead[a2] = -1;
806     edgeLevel[ei] = -1;
807
808     return splitted;
809 }
810
811 bool isConnected(int v, int w) const { return forests[0].isConnected(v, w); }
812 };
813 typedef HolmDeLichtenbergThorup FullyDynamicConnectivity;
814 map<int, map<int, int>> mp;
815
816 int main() {
817     int n, m;
818     scanf("%d%d", &n, &m);
819     mp.clear();
820     FullyDynamicConnectivity fdc;
821     fdc.init(n + 1, m + 1);
822     int posE = 0;
823     int lstans = 0;
824     for (int i = 1, op, u, v, _u, _v; i <= m; ++i) {
825         scanf("%d%d%d", &op, &u, &v);
826         u ^= lstans;
827         v ^= lstans;
828         _u = u, _v = v;
829         if (u < v)
830             swap(u, v);
831         if (op == 0) {
832             mp[u][v] = ++posE;
833             fdc.insertEdge(posE, u, v);
834         } else if (op == 1) {
835             fdc.deleteEdge(mp[u][v]);
836             mp[u].erase(v);
837         } else {
838             int ok = fdc.isConnected(u, v);
839             if (ok)
840                 lstans = _u;
841             else
842                 lstans = _v;
843             printf("%c\n", "NY"[ok]);
844         }
845     }
846     return 0;
847 }

```

## 2.10 Graph

```

1 namespace Backlight {
2
3 struct Graph {

```

```

4   struct Edge {
5       int u, v;
6       Edge(){}
7       Edge(int _u, int _v): u(_u), v(_v) {}
8   };
9
10  int V;
11  vector<vector<Edge>> G;
12
13  Graph() : V(0) {}
14  Graph(int _V) : V(_V), G(_V + 1) {}
15
16  inline void addarc(int u, int v) {
17      assert(1 <= u && u <= V);
18      assert(1 <= v && v <= V);
19      G[u].push_back(Edge(u, v));
20  }
21
22  inline void addedge(int u, int v) {
23      addarc(u, v);
24      addarc(v, u);
25  }
26 };
27
28 }

```

---

## 2.11 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     {
19         g.resize(n);
20     }
21     virtual int add(int from, int to, T cost) = 0;
22 };
23
24 // undirectedgraph
25 template <typename T>
26 class undirectedgraph : public graph<T> {
27 public:
28     using graph<T>::edges;
29     using graph<T>::g;
30     using graph<T>::n;
31
32     undirectedgraph(int _n)
33         : graph<T>(_n)
34     {
35     }
36     int add(int from, int to, T cost = 1)

```

```

37     {
38         assert(0 <= from && from < n && 0 <= to && to < n);
39         int id = (int)edges.size();
40         g[from].push_back(id);
41         g[to].push_back(id);
42         edges.push_back({ from, to, cost });
43         return id;
44     }
45 };
46
47 // blossom / find_max_unweighted_matching
48 template <typename T>
49 vector<int> find_max_unweighted_matching(const undirectedgraph<T>& g)
50 {
51     std::mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
52     vector<int> match(g.n, -1); // 匹配
53     vector<int> aux(g.n, -1); // 时间戳记
54     vector<int> label(g.n); // "o" or "i"
55     vector<int> orig(g.n); // 花根
56     vector<int> parent(g.n, -1); // 父节点
57     queue<int> q;
58     int aux_time = -1;
59
60     auto lca = [&](int v, int u) {
61         aux_time++;
62         while (true) {
63             if (v != -1) {
64                 if (aux[v] == aux_time) { // 找到拜访过的点 也就是 LCA
65                     return v;
66                 }
67                 aux[v] = aux_time;
68                 if (match[v] == -1) {
69                     v = -1;
70                 } else {
71                     v = orig[parent[match[v]]]; // 以匹配点的父节点继续寻找
72                 }
73             }
74             swap(v, u);
75         }
76     }; // lca
77
78     auto blossom = [&](int v, int u, int a) {
79         while (orig[v] != a) {
80             parent[v] = u;
81             u = match[v];
82             if (label[u] == 1) { // 初始点设为 "o" 找增广路
83                 label[u] = 0;
84                 q.push(u);
85             }
86             orig[v] = orig[u] = a; // 缩花
87             v = parent[u];
88         }
89     }; // blossom
90
91     auto augment = [&](int v) {
92         while (v != -1) {
93             int pv = parent[v];
94             int next_v = match[pv];
95             match[v] = pv;
96             match[pv] = v;
97             v = next_v;
98         }
99     }; // augment
100
101     auto bfs = [&](int root) {

```

```

102     fill(label.begin(), label.end(), -1);
103     iota(orig.begin(), orig.end(), 0);
104     while (!q.empty()) {
105         q.pop();
106     }
107     q.push(root);
108     // 初始点设为 "o", 这里以 "0" 代替 "o", "1" 代替 "i"
109     label[root] = 0;
110     while (!q.empty()) {
111         int v = q.front();
112         q.pop();
113         for (int id : g.g[v]) {
114             auto& e = g.edges[id];
115             int u = e.from ^ e.to ^ v;
116             if (label[u] == -1) { // 找到未拜访点
117                 label[u] = 1; // 标记 "i"
118                 parent[u] = v;
119                 if (match[u] == -1) { // 找到未匹配点
120                     augment(u); // 寻找增广路径
121                     return true;
122                 }
123                 // 找到已匹配点 将与她匹配的点丢入 queue 延伸交错树
124                 label[match[u]] = 0;
125                 q.push(match[u]);
126                 continue;
127             } 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"
128                 int a = lca(orig[v], orig[u]);
129                 // 找 LCA 然后缩花
130                 blossom(u, v, a);
131                 blossom(v, u, a);
132             }
133         }
134     }
135     return false;
136 }; // bfs
137
138 auto greedy = [&]() {
139     vector<int> order(g.n);
140     // 随机打乱 order
141     iota(order.begin(), order.end(), 0);
142     shuffle(order.begin(), order.end(), rng);
143
144     // 将可以匹配的点匹配
145     for (int i : order) {
146         if (match[i] == -1) {
147             for (auto id : g.g[i]) {
148                 auto& e = g.edges[id];
149                 int to = e.from ^ e.to ^ i;
150                 if (match[to] == -1) {
151                     match[i] = to;
152                     match[to] = i;
153                     break;
154                 }
155             }
156         }
157     }
158 }; // greedy
159
160 // 一开始先随机匹配
161 greedy();
162 // 对未匹配点找增广路
163 for (int i = 0; i < g.n; i++) {
164     if (match[i] == -1) {
165         bfs(i);
166     }

```



```

167     }
168     return match;
169 }
170 int main()
171 {
172     ios::sync_with_stdio(0), cin.tie(0);
173     int n, m;
174     cin >> n >> m;
175     undirectedgraph<int> g(n);
176     int u, v;
177     for (int i = 0; i < m; i++) {
178         cin >> u >> v;
179         u--;
180         v--;
181         g.add(u, v, 1);
182     }
183     auto blossom_match = find_max_unweighted_matching(g);
184     vector<int> ans;
185     int tot = 0;
186     for (int i = 0; i < blossom_match.size(); i++) {
187         ans.push_back(blossom_match[i]);
188         if (blossom_match[i] != -1) {
189             tot++;
190         }
191     }
192     cout << (tot >> 1) << "\n";
193     for (auto x : ans) {
194         cout << x + 1 << " ";
195     }
196 }

```

## 2.12 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  {
10     int v, w, nxt;
11 } e[ N << 1 ];
12 int tot, head[ N ];
13 void init_graph( int n )
14 {
15     tot = 0;
16     fill( head + 1, head + 1 + n, 0 );
17 }
18 void add( int u, int v, int w )
19 {
20     ++tot;
21     e[ tot ] = ( edge ){ v, w, head[ u ] };
22     head[ u ] = tot;
23 }
24
25 int sz[ N ], son[ N ], h[ N ], f[ N ], w[ N ];
26 void dfs1( int u, int fa )
27 {
28     h[ u ] = h[ fa ] + 1;
29     f[ u ] = fa;
30     sz[ u ] = 1;
31     son[ u ] = 0;

```

```

32     for ( int i = head[ u ]; i; i = e[ i ].nxt )
33     {
34         int v = e[ i ].v;
35         if ( v == fa )
36             continue;
37         w[ v ] = e[ i ].w;
38         dfs1( v, u );
39         sz[ u ] += sz[ v ];
40         if ( sz[ v ] > sz[ son[ u ] ] )
41             son[ u ] = v;
42     }
43 }
44 int dfs_clock, dfn[ N ], rk[ N ], top[ N ];
45 void dfs2( int u, int fa, int tp )
46 {
47     ++dfs_clock;
48     dfn[ dfs_clock ] = w[ u ];
49     rk[ u ] = dfs_clock;
50     top[ u ] = tp;
51     if ( son[ u ] )
52         dfs2( son[ u ], u, tp );
53     for ( int i = head[ u ]; i; i = e[ i ].nxt )
54     {
55         int v = e[ i ].v;
56         if ( v == fa || v == son[ u ] )
57             continue;
58         dfs2( v, u, v );
59     }
60 }
61
62 #define mid ( ( l + r ) >> 1 )
63 #define lc ( x << 1 )
64 #define rc ( x << 1 | 1 )
65 #define lson lc, l, mid
66 #define rson rc, mid + 1, r
67 int sum[ N << 2 ], ma[ N << 2 ], mi[ N << 2 ], tag_inv[ N << 2 ];
68 void push_up( int x )
69 {
70     sum[ x ] = sum[ lc ] + sum[ rc ];
71     ma[ x ] = max( ma[ lc ], ma[ rc ] );
72     mi[ x ] = min( mi[ lc ], mi[ rc ] );
73 }
74 void push_down( int x )
75 {
76     if ( tag_inv[ x ] != 1 )
77     {
78         sum[ lc ] = -sum[ lc ];
79         swap( ma[ lc ], mi[ lc ] );
80         ma[ lc ] = -ma[ lc ];
81         mi[ lc ] = -mi[ lc ];
82         tag_inv[ lc ] = -tag_inv[ lc ];
83
84         sum[ rc ] = -sum[ rc ];
85         swap( ma[ rc ], mi[ rc ] );
86         ma[ rc ] = -ma[ rc ];
87         mi[ rc ] = -mi[ rc ];
88         tag_inv[ rc ] = -tag_inv[ rc ];
89
90         tag_inv[ x ] = 1;
91     }
92 }
93 void build( int x, int l, int r )
94 {
95     tag_inv[ x ] = 1;
96     if ( l == r )

```

```
97     {
98         sum[ x ] = ma[ x ] = mi[ x ] = dfn[ l ];
99         return;
100     }
101     build( lson );
102     build( rson );
103     push_up( x );
104 }
105
106 void update( int x, int l, int r, int p, int w )
107 {
108     if ( l == r )
109     {
110         sum[ x ] = ma[ x ] = mi[ x ] = w;
111         return;
112     }
113     push_down( x );
114     if ( p <= mid )
115         update( lson, p, w );
116     else
117         update( rson, p, w );
118     push_up( x );
119 }
120
121 void inverse( int x, int l, int r, int L, int R )
122 {
123     if ( l == L && r == R )
124     {
125         sum[ x ] = -sum[ x ];
126         swap( ma[ x ], mi[ x ] );
127         ma[ x ] = -ma[ x ];
128         mi[ x ] = -mi[ x ];
129         tag_inv[ x ] = -tag_inv[ x ];
130         return;
131     }
132     push_down( x );
133     if ( R <= mid )
134         inverse( lson, L, R );
135     else if ( L > mid )
136         inverse( rson, L, R );
137     else
138     {
139         inverse( lson, L, mid );
140         inverse( rson, mid + 1, R );
141     }
142     push_up( x );
143 }
144
145 int getsum( int x, int l, int r, int L, int R )
146 {
147     if ( l == L && r == R )
148         return sum[ x ];
149     push_down( x );
150     if ( R <= mid )
151         return getsum( lson, L, R );
152     else if ( L > mid )
153         return getsum( rson, L, R );
154     return getsum( lson, L, mid ) + getsum( rson, mid + 1, R );
155 }
156
157 int getmax( int x, int l, int r, int L, int R )
158 {
159     if ( l == L && r == R )
160         return ma[ x ];
161     push_down( x );
```

```

162     if ( R <= mid )
163         return getmax( lson, L, R );
164     else if ( L > mid )
165         return getmax( rson, L, R );
166     return max( getmax( lson, L, mid ), getmax( rson, mid + 1, R ) );
167 }
168
169 int getmin( int x, int l, int r, int L, int R )
170 {
171     if ( l == L && r == R )
172         return mi[ x ];
173     push_down( x );
174     if ( R <= mid )
175         return getmin( lson, L, R );
176     else if ( L > mid )
177         return getmin( rson, L, R );
178     return min( getmin( lson, L, mid ), getmin( rson, mid + 1, R ) );
179 }
180
181 void INVERSE( int u, int v )
182 {
183     while ( top[ u ] != top[ v ] )
184     {
185         if ( h[ top[ u ] ] < h[ top[ v ] ] )
186             swap( u, v );
187         inverse( 1, 1, n, rk[ top[ u ] ], rk[ u ] );
188         u = f[ top[ u ] ];
189     }
190     if ( h[ u ] != h[ v ] )
191     {
192         if ( h[ u ] > h[ v ] )
193             swap( u, v );
194         inverse( 1, 1, n, rk[ son[ u ] ], rk[ v ] );
195     }
196 }
197
198 int QSUM( int u, int v )
199 {
200     int res = 0;
201     while ( top[ u ] != top[ v ] )
202     {
203         if ( h[ top[ u ] ] < h[ top[ v ] ] )
204             swap( u, v );
205         res += getsum( 1, 1, n, rk[ top[ u ] ], rk[ u ] );
206         u = f[ top[ u ] ];
207     }
208     if ( h[ u ] != h[ v ] )
209     {
210         if ( h[ u ] > h[ v ] )
211             swap( u, v );
212         res += getsum( 1, 1, n, rk[ son[ u ] ], rk[ v ] );
213     }
214     return res;
215 }
216
217 int QMAX( int u, int v )
218 {
219     int res = INT_MIN;
220     while ( top[ u ] != top[ v ] )
221     {
222         if ( h[ top[ u ] ] < h[ top[ v ] ] )
223             swap( u, v );
224         res = max( res, getmax( 1, 1, n, rk[ top[ u ] ], rk[ u ] ) );
225         u = f[ top[ u ] ];
226     }

```

```

227     if ( h[ u ] != h[ v ] )
228     {
229         if ( h[ u ] > h[ v ] )
230             swap( u, v );
231         res = max( res, getMax( 1, 1, n, rk[ son[ u ] ], rk[ v ] ) );
232     }
233     return res;
234 }
235
236 int QMIN( int u, int v )
237 {
238     int res = INT_MAX;
239     while ( top[ u ] != top[ v ] )
240     {
241         if ( h[ top[ u ] ] < h[ top[ v ] ] )
242             swap( u, v );
243         res = min( res, getmin( 1, 1, n, rk[ top[ u ] ], rk[ u ] ) );
244         u = f[ top[ u ] ];
245     }
246     if ( h[ u ] != h[ v ] )
247     {
248         if ( h[ u ] > h[ v ] )
249             swap( u, v );
250         res = min( res, getmin( 1, 1, n, rk[ son[ u ] ], rk[ v ] ) );
251     }
252     return res;
253 }
254
255 int tu[ N ], tv[ N ];
256 void solve( int Case )
257 {
258     /* write code here */
259     /* gl & hf */
260     scanf( "%d", &n );
261     int u, v, w;
262     for ( int i = 1; i <= n - 1; ++i )
263     {
264         scanf( "%d %d %d", &u, &v, &w );
265         ++u, ++v;
266         add( u, v, w );
267         add( v, u, w );
268
269         tu[ i ] = u;
270         tv[ i ] = v;
271     }
272
273     dfs1( 1, 1 );
274     dfs2( 1, 1, 1 );
275
276     build( 1, 1, n );
277
278     scanf( "%d", &q );
279     char op[ 5 ];
280     int x, y;
281     for ( int i = 1; i <= q; ++i )
282     {
283         scanf( "%s %d %d", op, &x, &y );
284         ++x, ++y;
285         if ( op[ 0 ] == 'C' )
286         {
287             --x, --y;
288             int id = h[ tu[ x ] ] > h[ tv[ x ] ] ? tu[ x ] : tv[ x ];
289             update( 1, 1, n, rk[ id ], y );
290         }
291         else if ( op[ 0 ] == 'N' )

```

```

292     {
293         INVERSE( x, y );
294     }
295     else if ( op[ 0 ] == 'S' )
296     {
297         printf( "%d\n", QSUM( x, y ) );
298     }
299     else if ( op[ 1 ] == 'A' )
300     {
301         printf( "%d\n", QMAX( x, y ) );
302     }
303     else if ( op[ 1 ] == 'I' )
304     {
305         printf( "%d\n", QMIN( x, y ) );
306     }
307 }
308 }
309
310 int main()
311 {
312     int T = 1;
313     for ( int _ = 1; _ <= T; _++ )
314         solve( _ );
315     return 0;
316 }

```

## 2.13 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) G[i].clear(), R[i].clear();
5  }
6  inline void addarc(int u, int v) {
7      G[u].push_back(v);
8      R[v].push_back(u);
9  }
10
11 int n, m;
12 int dfs_clock, scc_cnt;
13 int dfn[N], belong[N];
14 bool vis[N];
15 void dfs1(int u) {
16     vis[u] = true;
17     for (const int& v: G[u]) {
18         if (!vis[v]) dfs1(v);
19     }
20     dfn[++dfs_clock] = u;
21 }
22 void dfs2(int u) {
23     belong[u] = scc_cnt;
24     for (const int& v: R[u]) {
25         if (!belong[v]) dfs2(v);
26     }
27 }
28 void kosaraju() {
29     dfs_clock = scc_cnt = 0;
30     fill(dfn + 1, dfn + 1 + n, 0);
31     fill(belong + 1, belong + 1 + n, 0);
32     fill(vis + 1, vis + 1 + n, false);
33     for (int i = 1; i <= n; ++i) {
34         if (!vis[i]) dfs1(i);
35     }
36 }

```

```

37     for (int i = n; i >= 1; --i) {
38         if (!belong[dfn[i]]) {
39             ++scc_cnt;
40             dfs2(dfn[i]);
41         }
42     }
43 }

```

---

## 2.14 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): u(_u), v(_v), w(_w) {}
10        bool operator < (const Edge& e) {
11            return w < e.w;
12        }
13    };
14
15    int V;
16    vector<vector<Edge>> G;
17    vector<Edge> E;
18
19    Wraph() : V(0) {}
20    Wraph(int _V) : V(_V), G(_V + 1) {}
21
22    inline void addarc(int u, int v, T w) {
23        assert(1 <= u && u <= V);
24        assert(1 <= v && v <= V);
25        G[u].push_back(Edge(u, v, w));
26        E.push_back(Edge(u, v, w));
27    }
28
29    inline void addedge(int u, int v, T w) {
30        addarc(u, v, w);
31        addarc(v, u, w);
32    }
33
34    /*****
35    T kruskal() {
36        vector<int> fa(V + 1);
37        for (int i = 1; i <= V; ++i) fa[i] = i;
38
39        auto find = [&fa] (auto self, int x) {
40            if (x == fa[x]) return x;
41            fa[x] = self(self, fa[x]);
42            return fa[x];
43        };
44
45        auto merge = [&fa, find] (int x, int y) {
46            x = find(find, x); y = find(find, y);
47            if (x == y) return false;
48            fa[x] = y;
49            return true;
50        };
51
52        T cost = 0;
53        int cnt = 0;
54        sort(E.begin(), E.end());

```

```

55     for (int i = 0; i < (int)E.size(); ++i) {
56         Edge e = E[i];
57         if (merge(e.u, e.v)) {
58             cost = e.w;
59             ++cnt;
60             if (cnt == V - 1) break;
61         }
62     }
63     return cost;
64 }
65 };
66
67 }

```

## 2.15 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; e[tot] = (Edge){v, head[u]}; head[u] = tot;
8      ++tot; e[tot] = (Edge){u, head[v]}; head[v] = tot;
9  }
10
11 int h[N], f[N], sz[N], son[N], top[N];
12 void dfs1(int u, int fa) {
13     h[u] = h[fa] + 1; f[u] = fa;
14     sz[u] = 1; son[u] = 0;
15     for (int i = head[u]; i; i = e[i].nxt) {
16         int v = e[i].v;
17         if (v == fa) continue;
18         dfs1(v, u);
19         sz[u] += sz[v];
20         if (sz[v] > sz[son[u]]) son[u] = v;
21     }
22 }
23
24 void dfs2(int u, int fa, int tp) {
25     top[u] = tp;
26     if (son[u]) dfs2(son[u], u, tp);
27     for (int i = head[u]; i; i = e[i].nxt) {
28         int v = e[i].v;
29         if (v == fa || v == son[u]) continue;
30         dfs2(v, u, v);
31     }
32 }
33
34 int LCA(int u, int v) {
35     while (top[u] != top[v]) {
36         if (h[top[u]] < h[top[v]]) swap(u, v);
37         u = f[top[u]];
38     }
39     if (h[u] > h[v]) swap(u, v);
40     return u;
41 }

```

## 2.16 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): u(_u), v(_v), w(_w) {}
10 };
11
12 int V;
13 vector<vector<Edge>> G;
14
15 Wraph() : V(0) {}
16 Wraph(int _V) : V(_V), G(_V + 1) {}
17
18 inline void addarc(int u, int v, T w = 1) {
19     assert(1 <= u && u <= V);
20     assert(1 <= v && v <= V);
21     G[u].push_back(Edge(u, v, w));
22 }
23
24 inline void addedge(int u, int v, T w = 1) {
25     addarc(u, v, w);
26     addarc(v, u, w);
27 }
28
29 /*****
30 vector<int> dep;
31 vector<T> dis;
32 vector<vector<int>> par;
33 int rt, LG;
34 void dfs(int u, int fa, int d1, int d2) {
35     dep[u] = d1; dis[u] = d2;
36     if (u == rt) {
37         for (int i = 0; i < LG; ++i) par[u][i] = rt;
38     } else {
39         par[u][0] = fa;
40         for (int i = 1; i < LG; ++i) {
41             par[u][i] = par[par[u][i - 1]][i - 1];
42         }
43     }
44
45     for (Edge& e: G[u]) {
46         int v = e.v; T w = e.w;
47         if (v == fa) continue;
48         dfs(v, u, d1 + 1, d2 + w);
49     }
50 }
51
52 inline void build_lca(int _rt) {
53     rt = _rt; LG = __lg(V + 1) + 1;
54     dep = vector<int>(V + 1);
55     dis = vector<T>(V + 1);
56     par = vector<vector<int>>(V + 1, vector<int>(LG));
57     dfs(rt, rt, 0, 0);
58 }
59
60 inline int jump(int u, int d) {
61     for (int j = LG - 1; j >= 0; --j) {
62         if ((1<<j) & d) u = par[u][j];
63     }
64     return u;
65 }
66
67 int lca(int u, int v) {

```

```

68     if (dep[u] < dep[v]) swap(u, v);
69     u = jump(u, dep[u] - dep[v]);
70     if (u == v) return u;
71     for(int i = LG - 1; i >= 0; --i){
72         if(par[u][i] != par[v][i]){
73             u = par[u][i];
74             v = par[v][i];
75         }
76     }
77     return par[u][0];
78 }
79 };
80
81 };

```

## 2.17 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): v(_v), nxt(_nxt), c(_c), f(0) {}
12        };
13
14        int V, E;
15        vector<int> h;
16        vector<Edge> e;
17
18        mf_graph() : V(0) {}
19        mf_graph(int _V) : V(_V), h(_V + 1, -1) {}
20
21        inline void addarc(int u, int v, Cap c) {
22            assert(1 <= u && u <= V);
23            assert(1 <= v && v <= V);
24            assert(0 <= c);
25
26            e.push_back(Edge(v, h[u], c)); h[u] = e.size() - 1;
27        }
28
29        inline void addedge(int u, int v, Cap c) {
30            addarc(u, v, c);
31            addarc(v, u, 0);
32        }
33
34        Cap maxflow(int s, int t) {
35            assert(1 <= s && s <= V);
36            assert(1 <= t && t <= V);
37            assert(s != t);
38
39            vector<int> f(V + 1), d(V + 1), st(V + 1);
40
41            auto bfs = [&]() {
42                fill(d.begin(), d.end(), -1);
43                queue<int> q;
44                q.push(s); d[s] = 0;
45                while(!q.empty()){
46                    int u = q.front(); q.pop();
47                    for(int i = h[u]; i != -1; i = e[i].nxt) {

```

```

48         int v = e[i].v;
49         if(e[i].c > e[i].f && d[v] == -1) {
50             d[v] = d[u] + 1;
51             if (v == t) break;
52             q.push(v);
53         }
54     }
55 }
56 return (d[t] != -1);
57 };
58
59 auto dfs = [&] (auto self, int u, Cap up) {
60     if(u == t || up == 0) return up;
61     Cap res = 0;
62     for(int& i = f[u]; i != -1; i = e[i].nxt) {
63         int v = e[i].v;
64         if(d[u] + 1 == d[v]) {
65             Cap nf = self(self, v, min(up, e[i].c - e[i].f));
66             if (nf <= 0) continue;
67             up -= nf;
68             res += nf;
69             e[i].f += nf;
70             e[i ^ 1].f -= nf;
71             if(up == 0) break;
72         }
73     }
74     if(res == 0) d[u] = -1;
75     return res;
76 };
77
78 Cap res = 0;
79 while(bfs()) {
80     f = h;
81     res += dfs(dfs, s, INF);
82 }
83 return res;
84 }
85 };
86
87 } // namespace Backlight

```

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

```

```

22
23 inline void addarc(int u, int v, Cap cap, Cost cost) {
24     assert(1 <= u && u <= V);
25     assert(1 <= v && v <= V);
26     e.push_back(Edge(v, h[u], cap, cost)); h[u] = e.size() - 1;
27 }
28
29 inline void addedge(int u, int v, Cap cap, Cost cost) {
30     addarc(u, v, cap, cost);
31     addarc(v, u, 0, -cost);
32 }
33
34 pair<Cap, Cost> mcmf(int s, int t) {
35     assert(1 <= s && s <= V);
36     assert(1 <= t && t <= V);
37     assert(s != t);
38
39     Cap flow = 0;
40     Cost cost = 0;
41
42     vector<int> pe(V + 1);
43     vector<bool> inq(V + 1);
44     vector<Cost> dis(V + 1);
45     vector<Cap> incf(V + 1);
46
47     auto spfa = [&]() {
48         fill(dis.begin(), dis.end(), INF);
49         queue<int> q;
50         q.push(s); dis[s] = 0; incf[s] = INF; incf[t] = 0;
51         while(!q.empty()) {
52             int u = q.front(); q.pop();
53             inq[u] = false;
54             for (int i = h[u]; i != -1; i = e[i].nxt) {
55                 int v = e[i].v, _cap = e[i].cap, _cost = e[i].cost;
56                 if (_cap == 0 || dis[v] <= dis[u] + _cost) continue;
57                 dis[v] = dis[u] + _cost;
58                 incf[v] = min(_cap, incf[u]);
59                 pe[v] = i;
60                 if (!inq[v]) q.push(v), inq[v] = true;
61             }
62         }
63         return incf[t];
64     };
65
66     auto update = [&]() {
67         flow += incf[t];
68         for (int i = t; i != s; i = e[pe[i] ^ 1].v) {
69             e[pe[i]].cap -= incf[t];
70             e[pe[i] ^ 1].cap += incf[t];
71             cost += incf[t] * e[pe[i]].cost;
72         }
73     };
74
75     while(spfa()) update();
76
77     return make_pair(flow, cost);
78 }
79
80 };
81
82 } // namespace Backlight

```

## 2.19 SCC

```

1 namespace Backlight {
2
3 struct Graph {
4     struct Edge {
5         int u, v;
6         Edge(){}
7         Edge(int _u, int _v): u(_u), v(_v) {}
8     };
9
10    int V;
11    vector<vector<Edge>> G;
12
13    Graph() : V(0) {}
14    Graph(int _V) : V(_V), G(_V + 1) {}
15
16    inline void addarc(int u, int v) {
17        assert(1 <= u && u <= V);
18        assert(1 <= v && v <= V);
19        G[u].push_back(Edge(u, v));
20    }
21
22    inline void addedge(int u, int v) {
23        addarc(u, v);
24        addarc(v, u);
25    }
26
27    /*****
28    int scc_clock, scc_cnt;
29    vector<int> dfn, low, belong, scc_size;
30    vector<bool> ins;
31    stack<int> stk;
32
33    void tarjan(int u, int fa) {
34        dfn[u] = low[u] = ++scc_clock;
35        ins[u] = true;
36        stk.push(u);
37
38        // bool flag = false;
39        for (Edge& e: G[u]) {
40            int v = e.v;
41            // if (v == fa && !flag) {
42            //     flag = true;
43            //     continue;
44            // }
45
46            if (!dfn[v]) {
47                tarjan(v, u);
48                low[u] = min(low[u], low[v]);
49            } else if (ins[v]) low[u] = min(low[u], dfn[v]);
50        }
51
52        if (dfn[u] == low[u]) {
53            ++scc_cnt; scc_size.push_back(0);
54            int top;
55            do {
56                top = stk.top(); stk.pop();
57                ins[top] = false;
58                belong[top] = scc_cnt;
59                ++scc_size[scc_cnt];
60            } while(u != top);
61        }
62    }
63

```

```

64 void build_scc() {
65     scc_clock = scc_cnt = 0;
66     dfn = vector<int>(V + 1);
67     low = vector<int>(V + 1);
68     belong = vector<int>(V + 1);
69     ins = vector<bool>(V + 1);
70     scc_size = vector<int>(1);
71
72     for (int i = 1; i <= V; ++i) {
73         if (!dfn[i]) tarjan(i, i);
74     }
75 }
76 };
77
78 }

```

## 2.20 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): u(_u), v(_v), w(_w) {}
10    };
11
12    int V;
13    vector<vector<Edge>> G;
14
15    Wraph() : V(0) {}
16    Wraph(int _V) : V(_V), G(_V + 1) {}
17
18    inline void addarc(int u, int v, T w) {
19        assert(1 <= u && u <= V);
20        assert(1 <= v && v <= V);
21        G[u].push_back(Edge(u, v, w));
22    }
23
24    inline void addedge(int u, int v, T w) {
25        addarc(u, v, w);
26        addarc(v, u, w);
27    }
28
29    /*****
30    vector<T> spfa(int S, T T_MAX) {
31        queue<int> q;
32        vector<T> dis(V + 1, T_MAX);
33        vector<bool> inq(V + 1, 0);
34        q.push(S); dis[S] = 0;
35        while(!q.empty()) {
36            int u = q.front(); q.pop();
37            inq[u] = 0;
38            for(Edge e: G[u]) {
39                if(dis[e.v] > dis[u] + e.w) {
40                    dis[e.v] = dis[u] + e.w;
41                    if(!inq[e.v]) {
42                        inq[e.v] = 1;
43                        q.push(e.v);
44                    }
45                }
46            }
47        }
48    }
49    */

```

```

47     }
48     return dis;
49 }
50 };
51
52 }

```

---

## 2.21 tree-divide

---

```

1  struct Edge {
2      int v, w;
3      Edge(){}
4      Edge(int _v, int _w): v(_v), w(_w) {}
5  };
6
7  vector<Edge> G[N];
8  inline void addedge(int u, int v, int w) {
9      G[u].push_back(Edge(v, w));
10     G[v].push_back(Edge(u, w));
11 }
12
13 bool vis[N];
14 int sz[N], max_sz[N];
15 void dfs_size(int u, int fa) {
16     sz[u] = 1; max_sz[u] = 0;
17     for (const Edge& e: G[u]) {
18         int v = e.v;
19         if (v == fa || vis[v]) continue;
20         dfs_size(v, u);
21         sz[u] += sz[v];
22         max_sz[u] = max(max_sz[u], sz[v]);
23     }
24 }
25
26 int Max, rt;
27 void dfs_root(int r, int u, int fa) {
28     max_sz[u] = max(max_sz[u], sz[r] - sz[u]);
29     if (Max > max_sz[u]) Max = max_sz[u], rt = u;
30     for (const Edge& e: G[u]) {
31         int v = e.v;
32         if (v == fa || vis[v]) continue;
33         dfs_root(r, v, u);
34     }
35 }
36
37 int dcnt, dis[N];
38 void dfs_dis(int u, int fa, int d) {
39     dis[++dcnt] = d;
40     for (const Edge& e: G[u]) {
41         int v = e.v, w = e.w;
42         if (v == fa || vis[v]) continue;
43         dfs_dis(v, u, d + w);
44     }
45 }
46
47 int ans[K];
48 void calc(int u, int w, int delta) {
49     dcnt = 0; dfs_dis(u, -1, w);
50     for (int i = 1; i <= dcnt; ++i) {
51         for (int j = i + 1; j <= dcnt; ++j) {
52             ans[dis[i] + dis[j]] += delta;
53         }
54     }
55 }

```

```

56
57 int n, m;
58 void DFS(int u) {
59     Max = n; dfs_size(u, -1); dfs_root(u, u, -1);
60     vis[rt] = 1;
61     calc(rt, 0, 1);
62     for (const Edge& e: G[rt]) {
63         int v = e.v, w = e.w;
64         if (vis[v]) continue;
65         calc(v, w, -1);
66         DFS(v);
67     }
68 }
69
70 void solve() {
71     read(n, m);
72
73     int u, v, w;
74     FOR(i, 2, n) {
75         read(u, v, w);
76         addedge(u, v, w);
77     }
78
79     DFS(1);
80
81     int k;
82     FOR(i, 1, m) {
83         read(k);
84         puts(ans[k] ? "AYE" : "NAY");
85     }
86 }

```

## 2.22 Wrap

```

1 namespace Backlight {
2
3 template<typename T>
4 struct Wrap {
5     struct Edge {
6         int u, v;
7         T w;
8         Edge(){}
9         Edge(int _u, int _v, T _w): u(_u), v(_v), w(_w) {}
10    };
11
12    int V;
13    vector<vector<Edge>> G;
14
15    Wrap() : V(0) {}
16    Wrap(int _V) : V(_V), G(_V + 1) {}
17
18    inline void addarc(int u, int v, T w = 1) {
19        assert(1 <= u && u <= V);
20        assert(1 <= v && v <= V);
21        G[u].push_back(Edge(u, v, w));
22    }
23
24    inline void addedge(int u, int v, T w = 1) {
25        addarc(u, v, w);
26        addarc(v, u, w);
27    }
28 };
29
30 }

```



## 2.23 WrapMatch

```

1 // Got this code from UOJ
2 #include <bits/stdc++.h>
3 using namespace std;
4
5 template <typename CostType, typename TotalCostType = int64_t>
6 class MaximumWeightedMatching {
7     /*
8      * Maximum Weighted Matching in General Graphs.
9      * -  $O(nm \log(n))$  time
10     * -  $O(n + m)$  space
11
12     * Note: each vertex is 1-indexed.
13     */
14 public:
15     using cost_t = CostType;
16     using tcost_t = TotalCostType;
17
18 private:
19     enum Label { kSeparated = -2,
20                 kInner = -1,
21                 kFree = 0,
22                 kOuter = 1 };
23     static constexpr cost_t Inf = cost_t(1) << (sizeof(cost_t) * 8 - 2);
24
25 private:
26     template <typename T>
27     class BinaryHeap {
28     public:
29         struct Node {
30             bool operator<(const Node& rhs) const { return value < rhs.value; }
31             T value;
32             int id;
33         };
34         BinaryHeap() { }
35         BinaryHeap(int N)
36             : size_(0)
37             , node(N + 1)
38             , index(N, 0)
39         {
40         }
41         int size() const { return size_; }
42         bool empty() const { return size_ == 0; }
43         void clear()
44         {
45             while (size_ > 0)
46                 index[node[size_--].id] = 0;
47         }
48         T min() const { return node[1].value; }
49         int argmin() const { return node[1].id; } // argmin ?
50         T get_val(int id) const { return node[index[id]].value; }
51         void pop()
52         {
53             if (size_ > 0)
54                 pop(1);
55         }
56         void erase(int id)
57         {
58             if (index[id])
59                 pop(index[id]);
60         }

```

```

61     bool has(int id) const { return index[id] != 0; }
62     void update(int id, T v)
63     {
64         if (!has(id))
65             return push(id, v);
66         bool up = (v < node[index[id]].value);
67         node[index[id]].value = v;
68         if (up)
69             up_heap(index[id]);
70         else
71             down_heap(index[id]);
72     }
73     void decrease_key(int id, T v)
74     {
75         if (!has(id))
76             return push(id, v);
77         if (v < node[index[id]].value)
78             node[index[id]].value = v, up_heap(index[id]);
79     }
80     void push(int id, T v)
81     {
82         // assert(!has(id));
83         index[id] = ++size_;
84         node[size_] = { v, id };
85         up_heap(size_);
86     }
87
88 private:
89     void pop(int pos)
90     {
91         index[node[pos].id] = 0;
92         if (pos == size_) {
93             --size_;
94             return;
95         }
96         bool up = (node[size_].value < node[pos].value);
97         node[pos] = node[size_--];
98         index[node[pos].id] = pos;
99         if (up)
100             up_heap(pos);
101         else
102             down_heap(pos);
103     }
104     void swap_node(int a, int b)
105     {
106         swap(node[a], node[b]);
107         index[node[a].id] = a;
108         index[node[b].id] = b;
109     }
110     void down_heap(int pos)
111     {
112         for (int k = pos, nk = k; 2 * k <= size_; k = nk) {
113             if (node[2 * k] < node[nk])
114                 nk = 2 * k;
115             if (2 * k + 1 <= size_ && node[2 * k + 1] < node[nk])
116                 nk = 2 * k + 1;
117             if (nk == k)
118                 break;
119             swap_node(k, nk);
120         }
121     }
122     void up_heap(int pos)
123     {
124         for (int k = pos; k > 1 && node[k] < node[k >> 1]; k >>= 1)
125             swap_node(k, k >> 1);

```

```

126     }
127     int size_;
128     vector<Node> node;
129     vector<int> index;
130 };
131
132 template <typename Key>
133 class PairingHeaps {
134 private:
135     struct Node {
136         Node()
137             : prev(-1)
138         {
139             // "prev < 0" means the node is unused.
140         }
141         Node(Key v)
142             : key(v)
143             , child(0)
144             , next(0)
145             , prev(0)
146         {
147         }
148         Key key;
149         int child, next, prev;
150     };
151 public:
152     PairingHeaps(int H, int N)
153         : heap(H)
154         , node(N)
155     {
156         // It consists of `H` Pairing heaps.
157         // Each heap-node ID can appear at most 1 time(s) among heaps
158         // and should be in [1, N).
159     }
160
161     void clear(int h)
162     {
163         if (heap[h])
164             clear_rec(heap[h]), heap[h] = 0;
165     }
166     void clear_all()
167     {
168         for (size_t i = 0; i < heap.size(); ++i)
169             heap[i] = 0;
170         for (size_t i = 0; i < node.size(); ++i)
171             node[i] = Node();
172     }
173     bool empty(int h) const { return !heap[h]; }
174     bool used(int v) const { return node[v].prev >= 0; }
175     Key min(int h) const { return node[heap[h]].key; }
176     int argmin(int h) const { return heap[h]; }
177
178     void pop(int h)
179     {
180         // assert(!empty(h));
181         erase(h, heap[h]);
182     }
183     void push(int h, int v, Key key)
184     {
185         // assert(!used(v));
186         node[v] = Node(key);
187         heap[h] = merge(heap[h], v);
188     }
189     void erase(int h, int v)
190     {

```

```
191     if (!used(v))
192         return;
193     int w = two_pass_pairing(node[v].child);
194     if (!node[v].prev)
195         heap[h] = w;
196     else {
197         cut(v);
198         heap[h] = merge(heap[h], w);
199     }
200     node[v].prev = -1;
201 }
202 void decrease_key(int h, int v, Key key)
203 {
204     if (!used(v))
205         return push(h, v, key);
206     if (!node[v].prev)
207         node[v].key = key;
208     else {
209         cut(v);
210         node[v].key = key;
211         heap[h] = merge(heap[h], v);
212     }
213 }
214
215 private:
216 void clear_rec(int v)
217 {
218     for (; v; v = node[v].next) {
219         if (node[v].child)
220             clear_rec(node[v].child);
221         node[v].prev = -1;
222     }
223 }
224
225 inline void cut(int v)
226 {
227     auto& n = node[v];
228     int pv = n.prev, nv = n.next;
229     auto& pn = node[pv];
230     if (pn.child == v)
231         pn.child = nv;
232     else
233         pn.next = nv;
234     node[nv].prev = pv;
235     n.next = n.prev = 0;
236 }
237
238 int merge(int l, int r)
239 {
240     if (!l)
241         return r;
242     if (!r)
243         return l;
244     if (node[l].key > node[r].key)
245         swap(l, r);
246     int lc = node[r].next = node[l].child;
247     node[l].child = node[lc].prev = r;
248     return node[r].prev = l;
249 }
250
251 int two_pass_pairing(int root)
252 {
253     if (!root)
254         return 0;
255     int a = root;
```

```

256     root = 0;
257     while (a) {
258         int b = node[a].next, na = 0;
259         node[a].prev = node[a].next = 0;
260         if (b)
261             na = node[b].next, node[b].prev = node[b].next = 0;
262         a = merge(a, b);
263         node[a].next = root;
264         root = a;
265         a = na;
266     }
267     int s = node[root].next;
268     node[root].next = 0;
269     while (s) {
270         int t = node[s].next;
271         node[s].next = 0;
272         root = merge(root, s);
273         s = t;
274     }
275     return root;
276 }
277
278 private:
279     vector<int> heap;
280     vector<Node> node;
281 };
282
283 template <typename T>
284 struct PriorityQueue : public priority_queue<T, vector<T>, greater<T>> {
285     PriorityQueue() { }
286     PriorityQueue(int N) { this->c.reserve(N); }
287     T min() { return this->top(); }
288     void clear() { this->c.clear(); }
289 };
290
291 template <typename T>
292 struct Queue {
293     Queue() { }
294     Queue(int N)
295         : qh(0)
296         , qt(0)
297         , data(N)
298     {
299     }
300     T operator[](int i) const { return data[i]; }
301     void enqueue(int u) { data[qt++] = u; }
302     int dequeue() { return data[qh++]; }
303     bool empty() const { return qh == qt; }
304     void clear() { qh = qt = 0; }
305     int size() const { return qt; }
306     int qh, qt;
307     vector<T> data;
308 };
309
310 public:
311     struct InputEdge {
312         int from, to;
313         cost_t cost;
314     };
315
316 private:
317     template <typename T>
318     using ModifiableHeap = BinaryHeap<T>;
319     template <typename T>
320     using ModifiableHeaps = PairingHeaps<T>;

```

```

321 template <typename T>
322 using FastHeap = PriorityQueue<T>;
323
324 struct Edge {
325     int to;
326     cost_t cost;
327 };
328 struct Link {
329     int from, to;
330 };
331 struct Node {
332     struct NodeLink {
333         int b, v;
334     };
335     Node() { }
336     Node(int u)
337         : parent(0)
338         , size(1)
339     {
340         link[0] = link[1] = { u, u };
341     }
342     int next_v() const { return link[0].v; }
343     int next_b() const { return link[0].b; }
344     int prev_v() const { return link[1].v; }
345     int prev_b() const { return link[1].b; }
346     int parent, size;
347     NodeLink link[2];
348 };
349 struct Event {
350     Event() { }
351     Event(cost_t time, int id)
352         : time(time)
353         , id(id)
354     {
355     }
356     bool operator<(const Event& rhs) const { return time < rhs.time; }
357     bool operator>(const Event& rhs) const { return time > rhs.time; }
358     cost_t time;
359     int id;
360 };
361 struct EdgeEvent {
362     EdgeEvent() { }
363     EdgeEvent(cost_t time, int from, int to)
364         : time(time)
365         , from(from)
366         , to(to)
367     {
368     }
369     bool operator>(const EdgeEvent& rhs) const { return time > rhs.time; }
370     bool operator<(const EdgeEvent& rhs) const { return time < rhs.time; }
371     cost_t time;
372     int from, to;
373 };
374
375 public:
376     MaximumWeightedMatching(int N, const vector<InputEdge>& in)
377         : N(N)
378         , B((N - 1) / 2)
379         , S(N + B + 1)
380         , ofs(N + 2)
381         , edges(in.size() * 2)
382         , heap2(S)
383         , heap2s(S, S)
384         , heap3(edges.size())
385         , heap4(S)

```

```

386     {
387
388         for (auto& e : in)
389             ofs[e.from + 1]++, ofs[e.to + 1]++;
390         for (int i = 1; i <= N + 1; ++i)
391             ofs[i] += ofs[i - 1];
392         for (auto& e : in) {
393             edges[ofs[e.from]++] = { e.to, e.cost * 2 };
394             edges[ofs[e.to]++] = { e.from, e.cost * 2 };
395         }
396         for (int i = N + 1; i > 0; --i)
397             ofs[i] = ofs[i - 1];
398         ofs[0] = 0;
399     }
400
401     pair<tcost_t, vector<int>> maximum_weighted_matching(bool init_matching = false)
402     {
403         initialize();
404         set_potential();
405         if (init_matching)
406             find_maximal_matching();
407         for (int u = 1; u <= N; ++u)
408             if (!mate[u])
409                 do_edmonds_search(u);
410         tcost_t ret = compute_optimal_value();
411         return make_pair(ret, mate);
412     }
413
414 private:
415     tcost_t compute_optimal_value() const
416     {
417         tcost_t ret = 0;
418         for (int u = 1; u <= N; ++u)
419             if (mate[u] > u) {
420                 cost_t max_c = 0;
421                 for (int eid = ofs[u]; eid < ofs[u + 1]; ++eid) {
422                     if (edges[eid].to == mate[u])
423                         max_c = max(max_c, edges[eid].cost);
424                 }
425                 ret += max_c;
426             }
427         return ret >> 1;
428     }
429
430     inline tcost_t reduced_cost(int u, int v, const Edge& e) const
431     {
432         return tcost_t(potential[u]) + potential[v] - e.cost;
433     }
434
435     void rematch(int v, int w)
436     {
437         int t = mate[v];
438         mate[v] = w;
439         if (mate[t] != v)
440             return;
441         if (link[v].to == surface[link[v].to]) {
442             mate[t] = link[v].from;
443             rematch(mate[t], t);
444         } else {
445             int x = link[v].from, y = link[v].to;
446             rematch(x, y);
447             rematch(y, x);
448         }
449     }
450

```

```

451 void fix_mate_and_base(int b)
452 {
453     if (b <= N)
454         return;
455     int bv = base[b], mv = node[bv].link[0].v, bmv = node[bv].link[0].b;
456     int d = (node[bmv].link[1].v == mate[mv]) ? 0 : 1;
457     while (1) {
458         int mv = node[bv].link[d].v, bmv = node[bv].link[d].b;
459         if (node[bmv].link[1 ^ d].v != mate[mv])
460             break;
461         fix_mate_and_base(bv);
462         fix_mate_and_base(bmv);
463         bv = node[bmv].link[d].b;
464     }
465     fix_mate_and_base(base[b] = bv);
466     mate[b] = mate[bv];
467 }
468
469 void reset_time()
470 {
471     time_current_ = 0;
472     event1 = { Inf, 0 };
473 }
474
475 void reset_blossom(int b)
476 {
477     label[b] = kFree;
478     link[b].from = 0;
479     slack[b] = Inf;
480     lazy[b] = 0;
481 }
482
483 void reset_all()
484 {
485     label[0] = kFree;
486     link[0].from = 0;
487     for (int v = 1; v <= N; ++v) { // should be optimized for sparse graphs.
488         if (label[v] == kOuter)
489             potential[v] -= time_current_;
490         else {
491             int bv = surface[v];
492             potential[v] += lazy[bv];
493             if (label[bv] == kInner)
494                 potential[v] += time_current_ - time_created[bv];
495         }
496         reset_blossom(v);
497     }
498     for (int b = N + 1, r = B - unused_bid_idx; r > 0 && b < S; ++b)
499         if (base[b] != b) {
500             if (surface[b] == b) {
501                 fix_mate_and_base(b);
502                 if (label[b] == kOuter)
503                     potential[b] += (time_current_ - time_created[b]) << 1;
504                 else if (label[b] == kInner)
505                     fix_blossom_potential<kInner>(b);
506                 else
507                     fix_blossom_potential<kFree>(b);
508             }
509             heap2s.clear(b);
510             reset_blossom(b);
511             --r;
512         }
513
514     que.clear();
515     reset_time();

```



```

516     heap2.clear();
517     heap3.clear();
518     heap4.clear();
519 }
520
521 void do_edmonds_search(int root)
522 {
523     if (potential[root] == 0)
524         return;
525     link_blossom(surface[root], { 0, 0 });
526     push_outer_and_fix_potentials(surface[root], 0);
527     for (bool augmented = false; !augmented;) {
528         augmented = augment(root);
529         if (augmented)
530             break;
531         augmented = adjust_dual_variables(root);
532     }
533     reset_all();
534 }
535
536 template <Label Lab>
537 inline cost_t fix_blossom_potential(int b)
538 {
539     // Return the amount.
540     // (If v is an atom, the potential[v] will not be changed.)
541     cost_t d = lazy[b];
542     lazy[b] = 0;
543     if (Lab == kInner) {
544         cost_t dt = time_current_ - time_created[b];
545         if (b > N)
546             potential[b] -= dt << 1;
547         d += dt;
548     }
549     return d;
550 }
551
552 template <Label Lab>
553 inline void update_heap2(int x, int y, int by, cost_t t)
554 {
555     if (t >= slack[y])
556         return;
557     slack[y] = t;
558     best_from[y] = x;
559     if (y == by) {
560         if (Lab != kInner)
561             heap2.decrease_key(y, EdgeEvent(t + lazy[y], x, y));
562     } else {
563         int gy = group[y];
564         if (gy != y) {
565             if (t >= slack[gy])
566                 return;
567             slack[gy] = t;
568         }
569         heap2s.decrease_key(by, gy, EdgeEvent(t, x, y));
570         if (Lab == kInner)
571             return;
572         EdgeEvent m = heap2s.min(by);
573         heap2.decrease_key(by, EdgeEvent(m.time + lazy[by], m.from, m.to));
574     }
575 }
576
577 void activate_heap2_node(int b)
578 {
579     if (b <= N) {
580         if (slack[b] < Inf)

```

```

581         heap2.push(b, EdgeEvent(slack[b] + lazy[b], best_from[b], b));
582     } else {
583         if (heap2s.empty(b))
584             return;
585         EdgeEvent m = heap2s.min(b);
586         heap2.push(b, EdgeEvent(m.time + lazy[b], m.from, m.to));
587     }
588 }
589
590 void swap_blossom(int a, int b)
591 {
592     // Assume that `b` is a maximal blossom.
593     swap(base[a], base[b]);
594     if (base[a] == a)
595         base[a] = b;
596     swap(heavy[a], heavy[b]);
597     if (heavy[a] == a)
598         heavy[a] = b;
599     swap(link[a], link[b]);
600     swap(mate[a], mate[b]);
601     swap(potential[a], potential[b]);
602     swap(lazy[a], lazy[b]);
603     swap(time_created[a], time_created[b]);
604     for (int d = 0; d < 2; ++d)
605         node[node[a].link[d].b].link[1 ^ d].b = b;
606     swap(node[a], node[b]);
607 }
608
609 void set_surface_and_group(int b, int sf, int g)
610 {
611     surface[b] = sf, group[b] = g;
612     if (b <= N)
613         return;
614     for (int bb = base[b]; surface[bb] != sf; bb = node[bb].next_b()) {
615         set_surface_and_group(bb, sf, g);
616     }
617 }
618
619 void merge_smaller_blossoms(int bid)
620 {
621     int lb = bid, largest_size = 1;
622     for (int beta = base[bid], b = beta;;) {
623         if (node[b].size > largest_size)
624             largest_size = node[b].size, lb = b;
625         if ((b = node[b].next_b()) == beta)
626             break;
627     }
628     for (int beta = base[bid], b = beta;;) {
629         if (b != lb)
630             set_surface_and_group(b, lb, b);
631         if ((b = node[b].next_b()) == beta)
632             break;
633     }
634     group[lb] = lb;
635     if (largest_size > 1) {
636         surface[bid] = heavy[bid] = lb;
637         swap_blossom(lb, bid);
638     } else
639         heavy[bid] = 0;
640 }
641
642 void contract(int x, int y, int eid)
643 {
644     int bx = surface[x], by = surface[y];
645     assert(bx != by);

```

```

646     const int h = -(eid + 1);
647     link[surface[mate[bx]]].from = link[surface[mate[by]]].from = h;
648
649     int lca = -1;
650     while (1) {
651         if (mate[by] != 0)
652             swap(bx, by);
653         bx = lca = surface[link[bx].from];
654         if (link[surface[mate[bx]]].from == h)
655             break;
656         link[surface[mate[bx]]].from = h;
657     }
658
659     const int bid = unused_bid[--unused_bid_idx_];
660     assert(unused_bid_idx_ >= 0);
661     int tree_size = 0;
662     for (int d = 0; d < 2; ++d) {
663         for (int bv = surface[x]; bv != lca;) {
664             int mv = mate[bv], bmv = surface[mv], v = mate[mv];
665             int f = link[v].from, t = link[v].to;
666             tree_size += node[bv].size + node[bmv].size;
667             link[mv] = { x, y };
668
669             if (bv > N)
670                 potential[bv] += (time_current_ - time_created[bv]) << 1;
671             if (bmv > N)
672                 heap4.erase(bmv);
673             push_outer_and_fix_potentials(bmv, fix_blossom_potential<kInner>(bmv));
674
675             node[bv].link[d] = { bmv, mv };
676             node[bmv].link[1 ^ d] = { bv, v };
677             node[bmv].link[d] = { bv = surface[f], f };
678             node[bv].link[1 ^ d] = { bmv, t };
679         }
680         node[surface[x]].link[1 ^ d] = { surface[y], y };
681         swap(x, y);
682     }
683     if (lca > N)
684         potential[lca] += (time_current_ - time_created[lca]) << 1;
685     node[bid].size = tree_size + node[lca].size;
686     base[bid] = lca;
687     link[bid] = link[lca];
688     mate[bid] = mate[lca];
689     label[bid] = kOuter;
690     surface[bid] = bid;
691     time_created[bid] = time_current_;
692     potential[bid] = 0;
693     lazy[bid] = 0;
694
695     merge_smaller_blossoms(bid); // O(n log n) time / Edmonds search
696 }
697
698 void link_blossom(int v, Link l)
699 {
700     link[v] = { l.from, l.to };
701     if (v <= N)
702         return;
703     int b = base[v];
704     link_blossom(b, l);
705     int pb = node[b].prev_b();
706     l = { node[pb].next_v(), node[b].prev_v() };
707     for (int bv = b;;) {
708         int bw = node[bv].next_b();
709         if (bw == b)
710             break;

```

```

711         link_blossom(bw, l);
712         Link n1 = { node[bw].prev_v(), node[bv].next_v() };
713         bv = node[bw].next_b();
714         link_blossom(bv, n1);
715     }
716 }
717
718 void push_outer_and_fix_potentials(int v, cost_t d)
719 {
720     label[v] = kOuter;
721     if (v > N) {
722         for (int b = base[v]; label[b] != kOuter; b = node[b].next_b()) {
723             push_outer_and_fix_potentials(b, d);
724         }
725     } else {
726         potential[v] += time_current_ + d;
727         if (potential[v] < event1.time)
728             event1 = { potential[v], v };
729         que.enqueue(v);
730     }
731 }
732
733 bool grow(int root, int x, int y)
734 {
735     int by = surface[y];
736     bool visited = (label[by] != kFree);
737     if (!visited)
738         link_blossom(by, { 0, 0 });
739     label[by] = kInner;
740     time_created[by] = time_current_;
741     heap2.erase(by);
742     if (y != by)
743         heap4.update(by, time_current_ + (potential[by] >> 1));
744     int z = mate[by];
745     if (z == 0 && by != surface[root]) {
746         rematch(x, y);
747         rematch(y, x);
748         return true;
749     }
750     int bz = surface[z];
751     if (!visited)
752         link_blossom(bz, { x, y });
753     else
754         link[bz] = link[z] = { x, y };
755     push_outer_and_fix_potentials(bz, fix_blossom_potential<kFree>(bz));
756     time_created[bz] = time_current_;
757     heap2.erase(bz);
758     return false;
759 }
760
761 void free_blossom(int bid)
762 {
763     unused_bid[unused_bid_idx++] = bid;
764     base[bid] = bid;
765 }
766
767 int recalculate_minimum_slack(int b, int g)
768 {
769     // Return the destination of the best edge of blossom `g`.
770     if (b <= N) {
771         if (slack[b] >= slack[g])
772             return 0;
773         slack[g] = slack[b];
774         best_from[g] = best_from[b];
775         return b;

```

```

776     }
777     int v = 0;
778     for (int beta = base[b], bb = beta;;) {
779         int w = recalculate_minimum_slack(bb, g);
780         if (w != 0)
781             v = w;
782         if ((bb = node[bb].next_b()) == beta)
783             break;
784     }
785     return v;
786 }
787
788 void construct_smaller_components(int b, int sf, int g)
789 {
790     surface[b] = sf, group[b] = g; // `group[b] = g` is unneeded.
791     if (b <= N)
792         return;
793     for (int bb = base[b]; surface[bb] != sf; bb = node[bb].next_b()) {
794         if (bb == heavy[b]) {
795             construct_smaller_components(bb, sf, g);
796         } else {
797             set_surface_and_group(bb, sf, bb);
798             int to = 0;
799             if (bb > N)
800                 slack[bb] = Inf, to = recalculate_minimum_slack(bb, bb);
801             else if (slack[bb] < Inf)
802                 to = bb;
803             if (to > 0)
804                 heap2s.push(sf, bb, EdgeEvent(slack[bb], best_from[bb], to));
805         }
806     }
807 }
808
809 void move_to_largest_blossom(int bid)
810 {
811     const int h = heavy[bid];
812     cost_t d = (time_current_ - time_created[bid]) + lazy[bid];
813     lazy[bid] = 0;
814     for (int beta = base[bid], b = beta;;) {
815         time_created[b] = time_current_;
816         lazy[b] = d;
817         if (b != h)
818             construct_smaller_components(b, b, b), heap2s.erase(bid, b);
819         if ((b = node[b].next_b()) == beta)
820             break;
821     }
822     if (h > 0)
823         swap_blossom(h, bid), bid = h;
824     free_blossom(bid);
825 }
826
827 void expand(int bid)
828 {
829     int mv = mate[base[bid]];
830     move_to_largest_blossom(bid); // O(n log n) time / Edmonds search
831     Link old_link = link[mv];
832     int old_base = surface[mate[mv]], root = surface[old_link.to];
833     int d = (mate[root] == node[root].link[0].v) ? 1 : 0;
834     for (int b = node[old_base].link[d ^ 1].b; b != root;) {
835         label[b] = kSeparated;
836         activate_heap2_node(b);
837         b = node[b].link[d ^ 1].b;
838         label[b] = kSeparated;
839         activate_heap2_node(b);
840         b = node[b].link[d ^ 1].b;

```

```

841     }
842     for (int b = old_base;; b = node[b].link[d].b) {
843         label[b] = kInner;
844         int nb = node[b].link[d].b;
845         if (b == root)
846             link[mate[b]] = old_link;
847         else
848             link[mate[b]] = { node[b].link[d].v, node[nb].link[d ^ 1].v };
849         link[surface[mate[b]]] = link[mate[b]]; // fix tree links
850         if (b > N) {
851             if (potential[b] == 0)
852                 expand(b);
853             else
854                 heap4.push(b, time_current_ + (potential[b] >> 1));
855         }
856         if (b == root)
857             break;
858         push_outer_and_fix_potentials(nb, fix_blossom_potential<kInner>(b = nb));
859     }
860 }
861
862 bool augment(int root)
863 {
864     // Return true if an augmenting path is found.
865     while (!que.empty()) {
866         int x = que.dequeue(), bx = surface[x];
867         if (potential[x] == time_current_) {
868             if (x != root)
869                 rematch(x, 0);
870             return true;
871         }
872         for (int eid = ofs[x]; eid < ofs[x + 1]; ++eid) {
873             auto& e = edges[eid];
874             int y = e.to, by = surface[y];
875             if (bx == by)
876                 continue;
877             Label l = label[by];
878             if (l == kOuter) {
879                 cost_t t = reduced_cost(x, y, e) >> 1; // < 2 * Inf
880                 if (t == time_current_) {
881                     contract(x, y, eid);
882                     bx = surface[x];
883                 } else if (t < event1.time) {
884                     heap3.emplace(t, x, eid);
885                 }
886             } else {
887                 tcost_t t = reduced_cost(x, y, e); // < 3 * Inf
888                 if (t >= Inf)
889                     continue;
890                 if (l != kInner) {
891                     if (cost_t(t) + lazy[by] == time_current_) {
892                         if (grow(root, x, y))
893                             return true;
894                     } else
895                         update_heap2<kFree>(x, y, by, t);
896                 } else {
897                     if (mate[x] != y)
898                         update_heap2<kInner>(x, y, by, t);
899                 }
900             }
901         }
902     }
903     return false;
904 }
905

```

```

906 bool adjust_dual_variables(int root)
907 {
908     // delta1 : rematch
909     cost_t time1 = event1.time;
910
911     // delta2 : grow
912     cost_t time2 = Inf;
913     if (!heap2.empty())
914         time2 = heap2.min().time;
915
916     // delta3 : contract :  $O(m \log n)$  time / Edmonds search [ bottleneck (?) ]
917     cost_t time3 = Inf;
918     while (!heap3.empty()) {
919         EdgeEvent e = heap3.min();
920         int x = e.from, y = edges[e.to].to; // e.to is some edge id.
921         if (surface[x] != surface[y]) {
922             time3 = e.time;
923             break;
924         } else
925             heap3.pop();
926     }
927
928     // delta4 : expand
929     cost_t time4 = Inf;
930     if (!heap4.empty())
931         time4 = heap4.min().time;
932
933     // -- events --
934     cost_t time_next = min(min(time1, time2), min(time3, time4));
935     assert(time_current_ <= time_next && time_next < Inf);
936     time_current_ = time_next;
937
938     if (time_current_ == event1.time) {
939         int x = event1.id;
940         if (x != root)
941             rematch(x, 0);
942         return true;
943     }
944     while (!heap2.empty() && heap2.min().time == time_current_) {
945         int x = heap2.min().from, y = heap2.min().to;
946         if (grow(root, x, y))
947             return true; // `grow` function will call `heap2.erase(by)`.
948     }
949     while (!heap3.empty() && heap3.min().time == time_current_) {
950         int x = heap3.min().from, eid = heap3.min().to;
951         int y = edges[eid].to;
952         heap3.pop();
953         if (surface[x] == surface[y])
954             continue;
955         contract(x, y, eid);
956     }
957     while (!heap4.empty() && heap4.min().time == time_current_) {
958         int b = heap4.argmin();
959         heap4.pop();
960         expand(b);
961     }
962     return false;
963 }
964
965 private:
966 void initialize()
967 {
968     que = Queue<int>(N);
969     mate.assign(S, 0);
970     link.assign(S, { 0, 0 });

```

```

971     label.assign(S, kFree);
972     base.resize(S);
973     for (int u = 1; u < S; ++u)
974         base[u] = u;
975     surface.resize(S);
976     for (int u = 1; u < S; ++u)
977         surface[u] = u;
978
979     potential.resize(S);
980     node.resize(S);
981     for (int b = 1; b < S; ++b)
982         node[b] = Node(b);
983
984     unused_bid.resize(B);
985     for (int i = 0; i < B; ++i)
986         unused_bid[i] = N + B - i;
987     unused_bid_idx_ = B;
988
989     // for O(nm Log n) implementation
990     reset_time();
991     time_created.resize(S);
992     slack.resize(S);
993     for (int i = 0; i < S; ++i)
994         slack[i] = Inf;
995     best_from.assign(S, 0);
996     heavy.assign(S, 0);
997     lazy.assign(S, 0);
998     group.resize(S);
999     for (int i = 0; i < S; ++i)
1000         group[i] = i;
1001 }
1002
1003 void set_potential()
1004 {
1005     for (int u = 1; u <= N; ++u) {
1006         cost_t max_c = 0;
1007         for (int eid = ofs[u]; eid < ofs[u + 1]; ++eid) {
1008             max_c = max(max_c, edges[eid].cost);
1009         }
1010         potential[u] = max_c >> 1;
1011     }
1012 }
1013
1014 void find_maximal_matching()
1015 {
1016     // Find a maximal matching naively.
1017     for (int u = 1; u <= N; ++u)
1018         if (!mate[u]) {
1019             for (int eid = ofs[u]; eid < ofs[u + 1]; ++eid) {
1020                 auto& e = edges[eid];
1021                 int v = e.to;
1022                 if (mate[v] > 0 || reduced_cost(u, v, e) > 0)
1023                     continue;
1024                 mate[u] = v;
1025                 mate[v] = u;
1026                 break;
1027             }
1028         }
1029 }
1030
1031 private:
1032     int N, B, S; // N = |V|, B = (|V| - 1) / 2, S = N + B + 1
1033     vector<int> ofs;
1034     vector<Edge> edges;
1035

```



```

1036 Queue<int> que;
1037 vector<int> mate, surface, base;
1038 vector<Link> link;
1039 vector<Label> label;
1040 vector<cost_t> potential;
1041
1042 vector<int> unused_bid;
1043 int unused_bid_idx_;
1044 vector<Node> node;
1045
1046 // for O(nm log n) implementation
1047 vector<int> heavy, group;
1048 vector<cost_t> time_created, lazy, slack;
1049 vector<int> best_from;
1050
1051 cost_t time_current_;
1052 Event event1;
1053 ModifiableHeap<EdgeEvent> heap2;
1054 ModifiableHeaps<EdgeEvent> heap2s;
1055 FastHeap<EdgeEvent> heap3;
1056 ModifiableHeap<cost_t> heap4;
1057 };
1058
1059 using MWM = MaximumWeightedMatching<int>;
1060 using Edge = MWM::InputEdge;
1061
1062 int main()
1063 {
1064     ios::sync_with_stdio(false); cin.tie(0); cout.tie(0);
1065     int N, M;
1066     cin >> N >> M;
1067     vector<Edge> edges(2 * M);
1068     vector<int> ou(N + 2), ov(N + 2);
1069
1070     int u, v, c;
1071     for (int i = 0; i < M; ++i) {
1072         cin >> u >> v >> c;
1073         edges[i] = { u, v, c };
1074         ou[u + 1] += 1;
1075         ov[v + 1] += 1;
1076     }
1077     for (int i = 1; i <= N + 1; ++i)
1078         ov[i] += ov[i - 1];
1079     for (int i = 0; i < M; ++i)
1080         edges[M + (ov[edges[i].to]++)] = edges[i];
1081     for (int i = 1; i <= N + 1; ++i)
1082         ou[i] += ou[i - 1];
1083     for (int i = 0; i < M; ++i)
1084         edges[ou[edges[i + M].from]++] = edges[i + M];
1085     edges.resize(M);
1086
1087     auto ans = MWM(N, edges).maximum_weighted_matching();
1088     cout << ans.first << endl;
1089     for (int i = 1; i <= N; ++i) {
1090         cout << ans.second[i] << (i == N ? '\n' : ' ');
1091     }
1092     return 0;
1093 }

```

## 3 math

### 3.1 2DGeometry

---

```

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

```

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

```

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

```

```

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

```

```

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

```

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

```

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

```



```

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

```

```

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

```

```

582         return a.x < b.x;
583     });
584     point p1 = p[0], p2 = p[1];
585     double dis = distance(p1, p2);
586     __getMinPair(0, n - 1, p1, p2, dis);
587     return make_pair(p1, p2);
588 }
589
590 double getMinDis() {
591     assert(n >= 1);
592     if (n == 2) return distance(p[0], p[1]);
593
594     sort(p.begin(), p.end(), [] (const point& a, const point& b) {
595         return a.x < b.x;
596     });
597     point p1 = p[0], p2 = p[1];
598     double dis = distance(p1, p2);
599     __getMinPair(0, n - 1, p1, p2, dis);
600     return dis;
601 }

```

```

603 // 最小圆覆盖 (P2253, P1472)
604 // 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  $O(n)$ 
605 circle getMinCircle() {
606     // 随机打乱顺序
607     srand(time(0));
608     for (int i = n - 1; i >= 1; --i) swap(p[i], p[rand() % i]);
609
610     circle c(p[0], 0);
611     for (int i = 0; i < n; ++i) {
612         if (c.relationToPoint(p[i]) == 2) continue;
613         c.p = (p[0] + p[i]) / 2;
614         c.r = distance(p[0], p[i]) / 2;
615
616         for (int j = 1; j < i; ++j) {
617             if (c.relationToPoint(p[j]) == 2) continue;
618             c.p = (p[i] + p[j]) / 2;
619             c.r = distance(p[i], p[j]) / 2;
620
621             for (int k = 1; k < j; ++k) {
622                 if (c.relationToPoint(p[k]) == 2) continue;
623                 c = circle(p[i], p[j], p[k]);
624             }
625         }
626     }
627     return c;
628 }

```

```

630
631 // 点与多边形的位置关系
632 // 0 外部
633 // 1 内部
634 // 2 边上
635 // 3 点上
636 int relationToPoint(point a) {
637     for (int i = 0; i < n; ++i) if (p[i] == a) return 3;
638
639     getline();
640     for (int i = 0; i < n; ++i) if (l[i].relationToPoint(a) == 3) return 2;
641
642     int cnt = 0;
643     for (int i = 0, j; i < n; ++i) {
644         j = (i + 1) % n;
645         int k = sgn(det(p[j], a, p[i]));
646         int u = sgn(p[i].y - a.y);

```

```

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

```

```

712 // 半平面交
713 // 排序随机增量法 (SI) 求解半平面交, 复杂度为  $O(n \log n)$ 
714 // 瓶颈为排序算法, 用基数排序则为  $O(n)$ 
715 // 最终的结果为一个凸包, 若少于 3 个点则说明无解
716
717
718 // 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)
719 // 多边形的半平面交即为多边形的核 (P4196)
720
721 bool getHalfPlaneIntersection(vector<halfplane>& h, polygon& hpi) {
722     int n = int(h.size()), l, r;
723     sort(h.begin(), h.end());
724
725     vector<point> p(n);
726     vector<halfplane> q(n);
727
728     l = r = 0;
729     q[l] = h[0];
730     for (int i = 1; i < n; ++i) {
731         while(l < r && relationPointToHalfplane(p[r - 1], h[i])) --r;
732         while(l < r && relationPointToHalfplane(p[l], h[i])) ++l;
733         q[++r] = h[i];
734         if (l < r && sgn(det(q[r].v, q[r - 1].v)) == 0) {
735             --r;
736             if (!relationPointToHalfplane(h[i].s, q[r])) q[r] = h[i];
737         }
738         if (l < r) p[r - 1] = HalfplaneCrossHalfplane(q[r - 1], q[r]);
739     }
740     while(l < r && relationPointToHalfplane(p[r - 1], q[l])) --r;
741     if (r - l + 1 <= 2) return false; // 交不存在
742     p[r] = HalfplaneCrossHalfplane(q[l], q[r]);
743
744     hpi.resize(r - l + 1);
745     for (int i = l, j = 0; i <= r; ++i) hpi[j++] = p[i];
746
747     return true;
748 }
749
750 // 多边形内部半径最大的圆半径 (POJ3525)
751 // 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
752 double getMaxInsideCircleRadius(polygon& p) {
753     if (p.direction() != 1) reverse(p.p.begin(), p.p.end());
754     int n = p.n;
755
756     // 方向向量, 垂直单位向量
757     vector<point> d(n), v(n);
758     for (int i = 0; i < n; ++i) {
759         d[i] = p[(i + 1) % n] - p[i];
760         v[i] = d[i].trans90().unit();
761     }
762
763     double l = 0, r = 1e4, m;
764     while(r - l >= eps) {
765         m = (l + r) / 2;
766
767         vector<halfplane> h(n);
768         polygon hpi;
769         for (int i = 0; i < n; ++i) h[i] = halfplane(p[i] + v[i] * m, d[i]);
770         bool can = getHalfPlaneIntersection(h, hpi);
771
772         if (can) l = m;
773         else r = m;
774     }
775     return l;
776 }

```

```

777 }
778 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) return 0;
6         if (x < 0) return -1;
7         return 1;
8     }
9
10    struct point3 {
11        double x, y, z;
12        point3(double _x = 0, double _y = 0, double _z = 0) : x(_x), y(_y), z(_z) {}
13
14        bool operator == (const point3& p) const {
15            return sgn(x - p.x) == 0 && sgn(y - p.y) == 0 && sgn(z - p.z) == 0;
16        }
17
18        bool operator < (const point3& p) const {
19            if (sgn(x - p.x) != 0) return sgn(x - p.x) < 0;
20            if (sgn(y - p.y) != 0) return sgn(y - p.y) < 0;
21            return sgn(z - p.z) < 0;
22        }
23
24        point3 operator - (const point3& p) const {
25            return point3(x - p.x, y - p.y, z - p.z);
26        }
27
28        point3 operator + (const point3& p) const {
29            return point3(x + p.x, y + p.y, z + p.z);
30        }
31
32        point3 operator * (const double& a) const {
33            return point3(x * a, y * a, z * a);
34        }
35
36        point3 operator / (const double& a) const {
37            return point3(x / a, y / a, z / a);
38        }
39
40        double operator * (const point3& p) const {
41            return x * p.x + y * p.y + z * p.z;
42        }
43
44        point3 operator ^ (const point3& p) const {
45            return point3(y * p.z - z * p.y, z * p.x - x * p.z, x * p.y - y * p.x);
46        }
47
48        double length() {
49            return sqrt(x * x + y * y + z * z);
50        }
51
52        double length2() {
53            return x * x + y * y + z * z;
54        }
55
56        double disTo(const point3& p) {
57            return (p - *this).length();
58        }
59    }

```

```

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

```

```

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

```



```

190     }
191
192     // 平面和直线是否相交
193     int PlaneCrossLine(line3 l, point3& p) {
194         double x = o * (l.e - a);
195         double y = o * (l.s - a);
196         double d = x - y;
197         if (sgn(d) == 0) return 0;
198         p = ((l.s * x) - (l.e * y)) / d;
199         return 1;
200     }
201
202     // 点到平面的最近点
203     point3 PointToPlane(point3 p) {
204         line3 l = line3(p, p + o);
205         PlaneCrossLine(l, p);
206         return p;
207     }
208
209     // 平面和平面是否相交
210     int PlaneCrossPlane(plane f, line3& l) {
211         point3 o1 = o ^ f.o;
212         point3 o2 = o ^ o1;
213
214         double d = fabs(f.o * o2);
215         if (sgn(d) == 0) return 0;
216         point3 p = a + (o2 * (f.o * (f.a - a)) / d);
217         l = line3(p, p + o1);
218         return 1;
219     }
220 };
221
222 struct polygon3 {
223     struct face {
224         int a, b, c;
225         bool ok;
226     };
227
228     int n;
229     vector<point3> P;
230
231     int num;
232     vector<face> F;
233     vector<vector<int>> > G;
234
235     polygon3() : n(0) {}
236     polygon3(int _n) : n(_n), P(n), F(8 * n), G(n, vector<int>(n)) {}
237
238     double cmp(point3 p, face f) {
239         point3 p1 = P[f.b] - P[f.a];
240         point3 p2 = P[f.c] - P[f.a];
241         point3 p3 = p - P[f.a];
242         return (p1 ^ p2) * p3;
243     }
244
245     void deal(int p, int a, int b) {
246         int f = G[a][b];
247         if (F[f].ok) {
248             if (cmp(P[p], F[f]) > eps)
249                 dfs(p, f);
250             else {
251                 face add = {b, a, p, true};
252                 G[p][b] = G[a][p] = G[b][a] = num;
253                 F[num++] = add;
254             }

```

```

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

```

```

320         if (F[j].ok && cmp(P[i], F[j]) > eps) {
321             dfs(i, j);
322             break;
323         }
324     }
325 }
326
327 int tmp = num;
328 num = 0;
329 for (int i = 0; i < tmp; ++i) if (F[i].ok) {
330     F[num++] = F[i];
331 }
332 }
333
334 // 三维凸包表面积 (POJ3528)
335 double area() {
336     if (n == 3) return det(P[0], P[1], P[2]).length() / 2;
337
338     double res = 0;
339     for (int i = 0; i < num; ++i)
340         res += triArea(P[F[i].a], P[F[i].b], P[F[i].c]);
341     return res;
342 }
343
344 // 三维凸包体积
345 double volume() {
346     double res = 0;
347     point3 tmp(0, 0, 0);
348     for (int i = 0; i < num; ++i)
349         res += QuadVolume(tmp, P[F[i].a], P[F[i].b], P[F[i].c]);
350     return fabs(res);
351 }
352
353 // 表面三角形个数
354 double getTriangleCount() {
355     return num;
356 }
357
358 // 表面多边形个数 (HDU3662)
359 int getPolygonCount() {
360     int res = 0;
361     for (int i = 0; i < num; ++i) {
362         bool flag = true;
363         for (int j = 0; j < i; ++j) {
364             if (same(i, j)) {
365                 flag = 0;
366                 break;
367             }
368         }
369         res += flag;
370     }
371     return res;
372 }
373
374 // 重心 (HDU4273)
375 point3 getBaryCenter() {
376     point3 ans(0, 0, 0);
377     point3 o(0, 0, 0);
378
379     double all = 0;
380     for (int i = 0; i < num; ++i) {
381         double v = QuadVolume6(o, P[F[i].a], P[F[i].b], P[F[i].c]);
382         ans = ans + (((o + P[F[i].a] + P[F[i].b] + P[F[i].c]) / 4) * v);
383         all += v;
384     }

```

```

385         ans = ans / all;
386         return ans;
387     }
388
389     // 点到凸包第 i 个面上的距离
390     double PointToFace(point3 p, int i) {
391         double v1 = fabs(QuadVolume6(P[F[i].a], P[F[i].b], P[F[i].c], p));
392         double v2 = det(P[F[i].a], P[F[i].b], P[F[i].c]).length();
393         return v1 / v2;
394     }
395 };
396 }
397 using namespace Geometry3;

```

### 3.3 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; y = 0;
9             return a;
10        }
11        ll d = exgcd(b, a % b, x, y);
12        ll z = x; x = y; y = z - y * (a / b);
13        return d;
14    }
15
16    ll qpow(ll a, ll n, ll p) {
17        ll ans = 1;
18        for (; n; n >>= 1) {
19            if (n & 1) ans = ans * a % p;
20            a = a * a % p;
21        }
22        return ans;
23    }
24
25    // solve  $a^x = b \pmod p$ ,  $p$  is a prime must hold
26    ll BSGS(ll a, ll b, ll p) {
27        unordered_map<ll, int> mp;
28        if (__gcd(a, p) != 1) return -1;
29        if (b % p == 1) return 0;
30        a %= p; b %= p;
31        ll k = sqrt(p), t = qpow(a, k, p), s = b;
32        for (int i = 0; i <= k; i++, s = s * a % p) mp[s] = i;
33        s = 1;
34        for (int i = 0; i <= k; i++, s = s * t % p) {
35            int ans = mp.count(s) ? mp[s] : -1;
36            if (ans != -1 && i * k - ans >= 0) return i * k - ans;
37        }
38        return -1;
39    }
40
41    // solve  $a^x = b \pmod p$ ,  $p$  don't need to be a prime
42    ll EXBSGS(ll a, ll b, ll p) {
43        ll k = 0, d, c = 1, x, y;
44        a %= p; b %= p;
45        if (a == b) return 1;
46        if (b == 1) return 0;
47        while ((d = __gcd(a, p)) != 1) {
48            if (b % d) return -1;

```

```

49         k++; b /= d; p /= d; c = c * (a / d) % p;
50         if(c == b) return k;
51     }
52     if(p == 1) return k;
53     exgcd(c, p, x, y); b = (b * x % p + p) % p; a %= p;
54     ll ans = BSGS(a, b, p);
55     return ans == -1 ? ans : ans + k;
56 }
57 }
58
59 }

```

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

```

```

51
52 } // namespace Backlight

```

---

### 3.5 Combination

```

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

```

---

### 3.6 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; y = 0;
7             return a;
8         }
9         ll d = exgcd(b, a % b, x, y);
10        ll z = x; x = y; y = z - y * (a / b);
11        return d;
12    }
13
14
15
16    // CRT: solve x = a_i (mod m_i) for i in [0, n)
17
18    // GCD(m_i, m_j) = 1 hold
19    ll CRT(vector<ll>& a, vector<ll>& m) {
20        assert(a.size() == m.size());
21        assert(a.size() > 0);
22        int n = a.size();
23        ll M = 1, res = 0;
24        for (int i = 0; i < n; ++i) M *= m[i];
25        ll _M, x, y;
26        for (int i = 0; i < n; ++i) {
27            _M = M / m[i];
28            exgcd(_M, m[i], x, y);
29            res = (res + a[i] * _M % M * x % M) % M;
30        }
31        if (res < 0) res += M;
32        return res;
33    }
34
35    ll mul(ll a, ll b, ll mod) {
36        ll res = 0;
37        while(b) {

```

```

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

```

---

### 3.7 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]) primes.push_back(i);
9         for (int j = 0; j < (int)primes.size(); ++j) {
10             ll nxt = 1ll * primes[j] * i;
11             if (nxt > n) break;
12             is[nxt] = false;
13             if (i % primes[j] == 0) break;
14         }
15     }
16     return primes;
17 }
18
19 }

```

---

### 3.8 eval

```

1 int pri(char c)
2 {
3     if (c == '^') return 3;
4     if (c == '*' || c == '/') return 2;
5     if (c == '+' || c == '-') return 1;
6     return 0;
7 }
8
9 void in2post(char *s, char *t)
10 {
11     int n = strlen(s), j = 0;

```

```
12     stack<char> ops;
13     for (int i = 0; i < n; ++i) {
14         t[j] = 0;
15         if (islower(s[i])) {
16             while(i < n && isdigit(s[i])) {
17                 t[j++] = s[i++];
18             }
19             t[j++] = ' ';
20             --i;
21         } else if (s[i] == '(') {
22             ops.push('(');
23         } else if (s[i] == ')') {
24             char op = 0;
25             while(!ops.empty()) {
26                 op = ops.top();
27                 ops.pop();
28                 if (op == '(') break;
29                 t[j++] = op;
30                 t[j++] = ' ';
31             }
32             assert(op == '(');
33         } else {
34             while(!ops.empty() && pri(s[i]) <= pri(ops.top())) {
35                 t[j++] = ops.top();
36                 t[j++] = ' ';
37                 ops.pop();
38             }
39             ops.push(s[i]);
40         }
41     }
42     while(!ops.empty()) {
43         assert(ops.top() != '(');
44         t[j++] = ops.top();
45         t[j++] = ' ';
46         ops.pop();
47     }
48     t[j] = 0;
49 }
50
51 int eval(char* s)
52 {
53     int n = strlen(s);
54     stack<int> nums;
55     for (int i = 0; i < n; ++i) {
56         if (isdigit(s[i])) {
57             int num = 0;
58             while(i < n && isdigit(s[i])) {
59                 num = num * 10 + s[i++] - '0';
60             }
61             nums.push(num);
62             --i;
63             continue;
64         }
65
66         if (s[i] == ' ') continue;
67
68         assert(nums.size() >= 2);
69         int num2 = nums.top();
70         nums.pop();
71         int num1 = nums.top();
72         nums.pop();
73         switch(s[i]) {
74             case '+':
75                 nums.push(num1 + num2);
76                 break;
```



```

77         case '-':
78             nums.push(num1 - num2);
79             break;
80         case '*':
81             nums.push(num1 * num2);
82             break;
83         case '/':
84             nums.push(num1 / num2);
85             break;
86         default:
87             assert(false);
88             break;
89     }
90 }
91 assert(nums.size() == 1);
92 return nums.top();
93 }

```

### 3.9 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; y = 0;
11         return a;
12     }
13     ll d = exgcd(b, a % b, x, y);
14     ll z = x; x = y; y = z - y * (a / b);
15     return d;
16 }
17
18 }

```

### 3.10 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() : r(0), i(0) {}
7         Complex(long double _r, long double _i) : r(_r), i(_i) {}
8         Complex conj() { return Complex(r, -i); }
9         inline Complex operator-(const Complex &c) const { return Complex(r - c.r, i - c.i); }
10        inline Complex operator+(const Complex &c) const { return Complex(r + c.r, i + c.i); }
11        inline Complex operator*(const Complex &c) const { return Complex(r * c.r - i * c.i, r * c.i + i * c.r); }
12    };
13    ostream& operator << (ostream& os, Complex& c) { return os << "(" << c.r << ", " << c.i << ")"; }
14
15    int N;
16    vector<int> r;
17    void init(int n) {
18        N = 1; while(N <= n) N <<= 1;
19        r.resize(N);
20        for(int i = 1; i < N; ++i) r[i] = (r[i >> 1] >> 1) + ((i & 1) ? (N >> 1) : 0);
21    }

```

```

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

```

```

87         cc = LL(b[i].r + 0.5) % p;
88         h[i] = ((1ll * aa * (1 << 30) % p + 1ll * bb * (1 << 15) % p + cc) % p + p) % p;
89     }
90     return h;
91 }
92 } // namespace FFT

```

---

### 3.11 LinearBasis

```

1 struct LinearBasis {
2     static const int B = 62;
3     ll b[B];
4     int tot, n;
5
6     LinearBasis() {
7         tot = 0; n = 0;
8         memset(b, 0, sizeof(b));
9     }
10
11     bool insert(ll x) {
12         ++n;
13         for (int i = B - 1; i >= 0; --i) {
14             if (!(x >> i)) continue;
15             if (!b[i]) {
16                 ++tot;
17                 b[i] = x;
18                 break;
19             }
20             x ^= b[i];
21         }
22         return x > 0;
23     }
24
25     bool query(ll x) {
26         for (int i = B - 1; i >= 0; --i) {
27             if (!(x >> i)) continue;
28             if (!b[i]) return false;
29             x ^= b[i];
30         }
31         return x == 0;
32     }
33
34     ll queryMax() {
35         ll res = 0;
36         for (int i = B - 1; i >= 0; --i) {
37             if ((res ^ b[i]) > res) res ^= b[i];
38         }
39         return res;
40     }
41
42     ll queryMin() {
43         for (int i = 0; i < B; ++i) if (b[i]) return b[i];
44         return -1;
45     }
46
47     ll count() {
48         return 1LL << tot;
49     }
50
51     void rebuild() {
52         for (int i = B - 1; i >= 0; --i) {
53             for (int j = i - 1; j >= 0; --j) {
54                 if (b[i] & (1LL << j))
55                     b[i] ^= b[j];

```

```

56         }
57     }
58 }
59
60 // need rebuild first
61 ll queryKth(int k) {
62     if (k == 1 && tot < n) return 0;
63     if (tot < n) --k;
64     if (k > (1LL << tot) - 1) return -1;
65     ll res = 0;
66     for (int i = 0; i < B; ++i) {
67         if (b[i]) {
68             if (k & 1) res ^= b[i];
69             k >>= 1;
70         }
71     }
72     return res;
73 }
74 };

```

---

### 3.12 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) res = res * a % p;
10            a = a * a % p;
11            b >>= 1;
12        }
13        return res;
14    }
15
16    inline ll inv1(ll n, ll p) { return pow(n, p - 2, p); }
17
18    inline ll C1(ll n, ll m, ll p) {
19        if (m > n) return 0;
20        if (m > n - m) m = n - m;
21        ll u = 1, d = 1;
22        for (ll i = 1; i <= m; ++i) {
23            u = u * (n - i + 1) % p;
24            d = d * i % p;
25        }
26        return u * inv1(d, p) % p;
27    }
28
29 // solve n choose m (mod p) while p is a prime
30 ll lucas(ll n, ll m, ll p) {
31     if (m == 0) return 1;
32     return C1(n % p, m % p, p) * lucas(n / p, m / p, p) % p;
33 }
34
35
36 ll exgcd(ll a, ll b, ll& x, ll& y) {
37     if (b == 0) {
38         x = 1; y = 0;
39         return a;
40     }
41     ll d = exgcd(b, a % b, x, y);
42     ll z = x; x = y; y = z - y * (a / b);

```

```

43     return d;
44 }
45
46 inline ll inv2(ll n, ll p) {
47     ll x, y;
48     ll d = exgcd(n, p, x, y);
49     return d == 1 ? (p + x % p) % p : -1;
50 }
51
52 // n! mod pk without pi^x
53 ll f(ll n, ll pi, ll pk) {
54     if (!n) return 1;
55     ll res = 1;
56     if (n / pk) {
57         for (ll i = 2; i <= pk; ++i)
58             if (i % pi) res = res * i % pk;
59         res = pow(res, n / pk, pk);
60     }
61     for (ll i = 2; i <= n % pk; ++i)
62         if (i % pi) res = res * i % pk;
63     return res * f(n / pi, pi, pk) % pk;
64 }
65
66 ll C2(ll n, ll m, ll p, ll pi, ll pk) {
67     if (m > n) return 0;
68     ll a = f(n, pi, pk), b = f(m, pi, pk), c = f(n - m, pi, pk);
69     ll k = 0;
70     for (ll i = n; i; i /= pi) k += i / pi;
71     for (ll i = m; i; i /= pi) k -= i / pi;
72     for (ll i = n - m; i; i /= pi) k -= i / pi;
73     ll ans = a * inv2(b, pk) % pk * inv2(c, pk) % pk * pow(pi, k, pk) % pk;
74     ans = ans * (p / pk) % p * inv2(p / pk, pk) % p;
75     return ans;
76 }
77
78 // solve n choose m (mod p) while p might not be a prime
79 ll exlucas(ll n, ll m, ll p) {
80     ll x = p;
81     ll ans = 0;
82     for (ll i = 2; i <= p; ++i) {
83         if (x % i == 0) {
84             ll pk = 1;
85             while(x % i == 0) pk *= i, x /= i;
86             ans = (ans + C2(n, m, p, i, pk)) % p;
87         }
88     }
89     return ans;
90 }
91
92 } // namespace Lucas
93
94 } // namespace Backlight

```

### 3.13 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; swap(a, m);
8         u -= t * v; swap(u, v);
9     }

```

```

10  assert(m == 1);
11  return u;
12 }
13
14 template <typename T>
15 class Modular {
16 public:
17     using Type = typename decay<decltype(T::value)>::type;
18
19     constexpr Modular() : value() {}
20     template <typename U>
21     Modular(const U& x) {
22         value = normalize(x);
23     }
24
25     template <typename U>
26     static Type normalize(const U& x) {
27         Type v;
28         if (-mod() <= x && x < mod()) v = static_cast<Type>(x);
29         else v = static_cast<Type>(x % mod());
30         if (v < 0) v += mod();
31         return v;
32     }
33
34     const Type& operator()() const { return value; }
35     template <typename U>
36     explicit operator U() const { return static_cast<U>(value); }
37     constexpr static Type mod() { return T::value; }
38
39     Modular& operator+=(const Modular& other) { if ((value += other.value) >= mod()) value -= mod(); return *this; }
40     Modular& operator-=(const Modular& other) { if ((value -= other.value) < 0) value += mod(); return *this; }
41     template <typename U> Modular& operator+=(const U& other) { return *this += Modular(other); }
42     template <typename U> Modular& operator-=(const U& other) { return *this -= Modular(other); }
43     Modular& operator++() { return *this += 1; }
44     Modular& operator--() { return *this -= 1; }
45     Modular operator++(int) { Modular result(*this); *this += 1; return result; }
46     Modular operator--(int) { Modular result(*this); *this -= 1; return result; }
47     Modular operator-() const { return Modular(-value); }
48
49     template <typename U = T>
50     typename enable_if<is_same<typename Modular<U>::Type, int>::value, Modular>::type& operator*=(const Modular& rhs) {
51 #ifdef _WIN32
52         uint64_t x = static_cast<int64_t>(value) * static_cast<int64_t>(rhs.value);
53         uint32_t xh = static_cast<uint32_t>(x >> 32), xl = static_cast<uint32_t>(x), d, m;
54         asm(
55             "divl %4; \n\t"
56             : "=a" (d), "=d" (m)
57             : "d" (xh), "a" (xl), "r" (mod())
58         );
59         value = m;
60 #else
61         value = normalize(static_cast<int64_t>(value) * static_cast<int64_t>(rhs.value));
62 #endif
63         return *this;
64     }
65     template <typename U = T>
66     typename enable_if<is_same<typename Modular<U>::Type, long long>::value, Modular>::type& operator*=(const Modular& rhs) {
67         long long q = static_cast<long long>(static_cast<long double>(value) * rhs.value / mod());
68         value = normalize(value * rhs.value - q * mod());
69         return *this;
70     }
71     template <typename U = T>
72     typename enable_if<!is_integral<typename Modular<U>::Type>::value, Modular>::type& operator*=(const Modular& rhs) {
73         value = normalize(value * rhs.value);
74         return *this;

```

```

75     }
76
77     Modular& operator/=(const Modular& other) { return *this *= Modular(inverse(other.value, mod())); }
78
79     friend const Type& abs(const Modular& x) { return x.value; }
80
81     template <typename U>
82     friend bool operator==(const Modular<U>& lhs, const Modular<U>& rhs);
83
84     template <typename U>
85     friend bool operator<(const Modular<U>& lhs, const Modular<U>& rhs);
86
87     template <typename V, typename U>
88     friend V& operator>>(V& stream, Modular<U>& number);
89
90 private:
91     Type value;
92 };
93
94 template <typename T> bool operator==(const Modular<T>& lhs, const Modular<T>& rhs) { return lhs.value == rhs.value; }
95 template <typename T, typename U> bool operator==(const Modular<T>& lhs, U rhs) { return lhs == Modular<T>(rhs); }
96 template <typename T, typename U> bool operator==(U lhs, const Modular<T>& rhs) { return Modular<T>(lhs) == rhs; }
97
98 template <typename T> bool operator!=(const Modular<T>& lhs, const Modular<T>& rhs) { return !(lhs == rhs); }
99 template <typename T, typename U> bool operator!=(const Modular<T>& lhs, U rhs) { return !(lhs == rhs); }
100 template <typename T, typename U> bool operator!=(U lhs, const Modular<T>& rhs) { return !(lhs == rhs); }
101
102 template <typename T> bool operator<(const Modular<T>& lhs, const Modular<T>& rhs) { return lhs.value < rhs.value; }
103
104 template <typename T> Modular<T> operator+(const Modular<T>& lhs, const Modular<T>& rhs) { return Modular<T>(lhs) += rhs; }
105 template <typename T, typename U> Modular<T> operator+(const Modular<T>& lhs, U rhs) { return Modular<T>(lhs) += rhs; }
106 template <typename T, typename U> Modular<T> operator+(U lhs, const Modular<T>& rhs) { return Modular<T>(lhs) += rhs; }
107
108 template <typename T> Modular<T> operator-(const Modular<T>& lhs, const Modular<T>& rhs) { return Modular<T>(lhs) -= rhs; }
109 template <typename T, typename U> Modular<T> operator-(const Modular<T>& lhs, U rhs) { return Modular<T>(lhs) -= rhs; }
110 template <typename T, typename U> Modular<T> operator-(U lhs, const Modular<T>& rhs) { return Modular<T>(lhs) -= rhs; }
111
112 template <typename T> Modular<T> operator*(const Modular<T>& lhs, const Modular<T>& rhs) { return Modular<T>(lhs) *= rhs; }
113 template <typename T, typename U> Modular<T> operator*(const Modular<T>& lhs, U rhs) { return Modular<T>(lhs) *= rhs; }
114 template <typename T, typename U> Modular<T> operator*(U lhs, const Modular<T>& rhs) { return Modular<T>(lhs) *= rhs; }
115
116 template <typename T> Modular<T> operator/(const Modular<T>& lhs, const Modular<T>& rhs) { return Modular<T>(lhs) /= rhs; }
117 template <typename T, typename U> Modular<T> operator/(const Modular<T>& lhs, U rhs) { return Modular<T>(lhs) /= rhs; }
118 template <typename T, typename U> Modular<T> operator/(U lhs, const Modular<T>& rhs) { return Modular<T>(lhs) /= rhs; }
119
120 template<typename T, typename U>
121 Modular<T> power(const Modular<T>& a, const U& b) {
122     assert(b >= 0);
123     Modular<T> x = a, res = 1;
124     U p = b;
125     while (p > 0) {
126         if (p & 1) res *= x;
127         x *= x;
128         p >>= 1;
129     }
130     return res;
131 }
132
133 template <typename T>
134 bool IsZero(const Modular<T>& number) {
135     return number() == 0;
136 }
137
138 template <typename T>
139 string to_string(const Modular<T>& number) {

```

```

140     return to_string(number());
141 }
142
143 // U == std::ostream? but done this way because of fastoutput
144 template <typename U, typename T>
145 U& operator<<(U& stream, const Modular<T>& number) {
146     return stream << number();
147 }
148
149 // U == std::istream? but done this way because of fastinput
150 template <typename U, typename T>
151 U& operator>>(U& stream, Modular<T>& number) {
152     typename common_type<typename Modular<T>::Type, long long>::type x;
153     stream >> x;
154     number.value = Modular<T>::normalize(x);
155     return stream;
156 }
157
158 /*
159 using ModType = int;
160
161 struct VarMod { static ModType value; };
162 ModType VarMod::value;
163 ModType& md = VarMod::value;
164 using Mint = Modular<VarMod>;
165 */
166
167 const int md = 998244353;
168 using Mint = Modular<std::integral_constant<decay<decltype(MOD)>::type, MOD>>;
169
170 /*
171 vector<Mint> fact(1, 1);
172 vector<Mint> inv_fact(1, 1);
173
174 Mint C(int n, int k) {
175     if (k < 0 || k > n) {
176         return 0;
177     }
178     while ((int) fact.size() < n + 1) {
179         fact.push_back(fact.back() * (int) fact.size());
180         inv_fact.push_back(1 / fact.back());
181     }
182     return fact[n] * inv_fact[k] * inv_fact[n - k];
183 }
184 */

```

### 3.14 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; mu[1] = 1;
6      for (int i = 2; i < N; ++i) is[i] = true;
7      for (int i = 2; i < N; ++i) {
8          if (is[i]) primes[++pcnt] = i, mu[i] = -1;
9          for (int j = 1; j <= pcnt; ++j) {
10             ll nxt = 1ll * i * primes[j];
11             if (nxt >= N) break;
12             is[nxt] = false;
13             if (i % primes[j] == 0) {
14                 mu[nxt] = 0;
15                 break;
16             }

```



```

17         mu[nxt] = -mu[i];
18     }
19 }
20 for (int i = 1; i < N; ++i) mu[i] += mu[i - 1];
21 }

```

---

### 3.15 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 }

```

---

### 3.16 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; a %= P;
9             while(b) {
10                 if (b & 1) res = res * a % P;
11                 a = a * a % P;
12                 b >>= 1;
13             }
14             return res;
15         }
16
17         int N, L;
18         vector<ll> r;
19         void init(vector<ll>& a, vector<ll>& b) {
20             int l = a.size() + b.size();
21             N = 1; L = 0; while(N < l) N <<= 1, ++L;
22             a.resize(N); b.resize(N); r.resize(N);
23             for (int i = 0; i < N; ++i)
24                 r[i] = (r[i >> 1] >> 1) | ((i & 1) << (L - 1));
25         }
26
27         void work(vector<ll>& a, int flag) {
28             for(int i = 0; i < N; i++)
29                 if(i < r[i]) swap(a[i], a[r[i]]);
30             for(int mid = 1; mid < N; mid <<= 1) {
31                 ll wn = pow(flag == 1 ? G : Gi, (P - 1) / (mid << 1));
32                 for(int j = 0; j < N; j += (mid << 1)) {
33                     ll w = 1;
34                     for(int k = 0; k < mid; k++, w = (w * wn) % P) {
35                         int x = a[j + k], y = w * a[j + k + mid] % P;
36                         a[j + k] = (x + y) % P,
37                         a[j + k + mid] = (x - y + P) % P;
38                     }
39                 }
40             }
41         }
42     }
43 }

```

```

39     }
40 }
41 }
42
43 inline void NTT(vector<ll>& a) { work(a, 1); }
44 inline void INTT(vector<ll>& a) { work(a, -1); }
45
46 vector<ll> convolution(vector<ll> a, vector<ll> b) {
47     init(a, b);
48     NTT(a); NTT(b);
49     for (int i = 0; i < N; ++i) a[i] = a[i] * b[i] % P;
50     INTT(a);
51     ll inv = pow(N, P - 2);
52     for (int i = 0; i < N; ++i) a[i] = a[i] * inv % P;
53     return a;
54 }
55 } // namespace NTT
56
57 } // namespace Backlight

```

### 3.17 PollardRho

```

1 namespace Backlight {
2
3 namespace Pollard_Rho {
4     typedef long long ll;
5     typedef pair<ll, ll> PLL;
6     mt19937 rnd(chrono::steady_clock::now().time_since_epoch().count());
7
8     const int N = 1010000;
9     ll C, fac[10010], n, mut, a[1001000];
10    int T, cnt, i, l, prime[N], p[N], psize, _cnt;
11    ll _e[100], _pr[100];
12    vector<ll> d;
13
14    inline ll mul(ll a, ll b, ll p) {
15        if (p <= 1000000000) return a * b % p;
16        else if (p <= 1000000000000ll) return (((a*(b>>20)%p)<<20)+(a*(b&((1<<20)-1))))%p;
17        else {
18            ll d = (ll)floor(a*(long double)b / p + 0.5);
19            ll ret = (a * b - d * p) % p;
20            if (ret < 0) ret += p;
21            return ret;
22        }
23    }
24
25    void prime_table(){
26        int i, j, tot, t1;
27        for (i = 1; i <= psize; i++) p[i] = i;
28        for (i = 2, tot = 0; i <= psize; i++) {
29            if (p[i] == i) prime[++tot] = i;
30            for (j = 1; j <= tot && (t1 = prime[j] * i) <= psize; j++){
31                p[t1] = prime[j];
32                if (i % prime[j] == 0) break;
33            }
34        }
35    }
36
37    void init(int ps) {
38        psize = ps;
39        prime_table();
40    }
41
42    ll powl(ll a, ll n, ll p) {

```

```

43     ll ans = 1;
44     for (; n; n >>= 1) {
45         if (n & 1) ans = mul(ans, a, p);
46         a = mul(a, a, p);
47     }
48     return ans;
49 }
50
51 bool witness(ll a, ll n) {
52     int t = 0;
53     ll u = n - 1;
54     for (; ~u&1; u >>= 1) t++;
55     ll x = powl(a, u, n), _x = 0;
56     for (; t; t--) {
57         _x = mul(x, x, n);
58         if (_x == 1 && x != 1 && x != n - 1) return 1;
59         x = _x;
60     }
61     return _x != 1;
62 }
63
64 bool miller(ll n) {
65     if (n < 2) return 0;
66     if (n <= psize) return p[n] == n;
67     if (~n & 1) return 0;
68     for (int j = 0; j <= 7; j++) if (witness(rnd() % (n - 1) + 1, n)) return 0;
69     return 1;
70 }
71
72 ll gcd(ll a, ll b) {
73     ll ret = 1;
74     while (a != 0) {
75         if ((~a&1) && (~b&1)) ret <<= 1, a >>= 1, b >>= 1;
76         else if (~a&1) a >>= 1;
77         else if (~b&1) b >>= 1;
78         else {
79             if (a < b) swap(a, b);
80             a -= b;
81         }
82     }
83     return ret * b;
84 }
85
86 ll rho(ll n) {
87     for (;;) {
88         ll X = rnd() % n, Y, Z, T = 1, *lY = a, *lX = lY;
89         int tmp = 20;
90         C = rnd() % 10 + 3;
91         X = mul(X, X, n) + C; *(lY++) = X; lX++;
92         Y = mul(X, X, n) + C; *(lY++) = Y;
93         for (; X != Y;) {
94             ll t = X - Y + n;
95             Z = mul(T, t, n);
96             if (Z == 0) return gcd(T, n);
97             tmp--;
98             if (tmp == 0) {
99                 tmp = 20;
100                 Z = gcd(Z, n);
101                 if (Z != 1 && Z != n) return Z;
102             }
103             T = Z;
104             Y = *(lY++) = mul(Y, Y, n) + C;
105             Y = *(lY++) = mul(Y, Y, n) + C;
106             X = *(lX++);
107         }

```

```

108     }
109 }
110
111 void _factor(ll n) {
112     for (int i = 0; i < cnt; i++) {
113         if (n % fac[i] == 0) n /= fac[i], fac[cnt++] = fac[i];
114     }
115     if (n <= psize) {
116         for (; n != 1; n = p[n]) fac[cnt++] = p[n];
117         return;
118     }
119     if (miller(n)) fac[cnt++] = n;
120     else {
121         ll x = rho(n);
122         _factor(x); _factor(n / x);
123     }
124 }
125
126 void dfs(ll x, int dep) {
127     if (dep == _cnt) d.push_back(x);
128     else {
129         dfs(x, dep+1);
130         for (int i = 1; i <= _e[dep]; i++) dfs(x * _pr[dep], dep + 1);
131     }
132 }
133
134 void norm() {
135     sort(fac, fac + cnt);
136     _cnt = 0;
137     for (int i = 0; i < cnt; ++i)
138         if (i == 0 || fac[i] != fac[i-1]) _pr[_cnt] = fac[i], _e[_cnt++] = 1;
139     else _e[_cnt-1]++;
140 }
141
142 vector<ll> getd() {
143     d.clear();
144     dfs(1, 0);
145     return d;
146 }
147
148 /*****
149
150 // Attention: call init() before use
151
152 // get all factors
153 vector<ll> factorA(ll n) {
154     cnt = 0;
155     _factor(n);
156     norm();
157     vector<ll> d = getd();
158     sort(d.begin(), d.end());
159     return d;
160 }
161
162 // get prime factors
163 vector<ll> factorP(ll n) {
164     cnt = 0;
165     _factor(n);
166     norm();
167     vector<ll> d(_cnt);
168     for (int i = 0; i < _cnt; ++i) d[i] = _pr[i];
169     return d;
170 }
171
172 // get prime factors,  $n = pr_i^{e_i}$ 

```

```

173 vector<PLL> factorG(ll n) {
174     cnt = 0;
175     _factor(n);
176     norm();
177     vector<PLL> d(_cnt);
178     for (int i = 0; i < _cnt; ++i) d[i] = make_pair(_pr[i], _e[i]);
179     return d;
180 }
181
182 bool is_primitive(ll a, ll p) {
183     assert(miller(p));
184     vector<PLL> D = factorG(p - 1);
185     for (int i = 0; i < (int)D.size(); ++i) if (powl(a, (p-1) / D[i].first, p) == 1) return 0;
186     return 1;
187 }
188 }
189
190 }

```

### 3.18 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) : a(f) {
67         while (!a.empty() && !a.back())
68             a.pop_back();
69     }
70     poly(const vector<int> &f, int n) : a(f) {
71         a.resize(n);
72     }
73     int size() const {
74         return a.size();
75     }
76     int deg() const {
77         return a.size() - 1;
78     }
79     int operator[](int idx) const {
80         if (idx < 0 || idx >= size())
81             return 0;
82         return a[idx];
83     }
84     void input(int n) {
85         a.resize(n);
86         FE(v, a) rd(v);
87     }
88     void output(int n) {
89         for (int i = 0; i < n - 1; ++i) printf("%d ", (*this)[i]);
90         printf("%d\n", (*this)[n - 1]);
91     }
92     poly mulxk(int k) const {
93         auto b = a;
94         b.insert(b.begin(), k, 0);
95         return poly(b);
96     }
97     poly modxk(int k) const {
98         k = min(k, size());
99         return poly(std::vector<int>(a.begin(), a.begin() + k));
100    }
101    poly alignxk(int k) const {
102        return poly(a, k);
103    }
104    poly divxk(int k) const {
105        if (size() <= k)
106            return poly();
107        return poly(vector<int>(a.begin() + k, a.end()));
108    }

```

```

109 friend poly operator+(const poly& f, const poly& g) {
110     int k = max(f.size(), g.size());
111     vector<int> res(k);
112     for (int i = 0; i < k; ++i) {
113         res[i] = f[i] + g[i];
114         if (res[i] >= P)
115             res[i] -= P;
116     }
117     return poly(res);
118 }
119 friend poly operator - (const poly& f, const poly &g) {
120     int k = max(f.size(), g.size());
121     vector<int> res(k);
122     for (int i = 0; i < k; ++i) {
123         res[i] = f[i] - g[i];
124         if (res[i] < 0)
125             res[i] += P;
126     }
127     return poly(res);
128 }
129 friend poly operator * (const poly& f, const poly& g) {
130     int sz = 1, k = f.size() + g.size() - 1;
131     while (sz < k) sz *= 2;
132     vector<int> p = f.a, q = g.a;
133     p.resize(sz); q.resize(sz);
134     dft(p); dft(q);
135     for (int i = 0; i < sz; ++i)
136         p[i] = 1ll * p[i] * q[i] % P;
137     idft(p);
138     return poly(p);
139 }
140 friend poly operator / (const poly& f, const poly& g) {
141     return f.divide(g).first;
142 }
143 friend poly operator % (const poly& f, const poly& g) {
144     return f.divide(g).second;
145 }
146 poly &operator += (const poly& f) {
147     return (*this) = (*this) + f;
148 }
149 poly &operator -= (const poly& f) {
150     return (*this) = (*this) - f;
151 }
152 poly &operator *= (const poly& f) {
153     return (*this) = (*this) * f;
154 }
155 poly &operator /= (const poly& f) {
156     return (*this) = divide(f).first;
157 }
158 poly &operator %= (const poly& f) {
159     return (*this) = divide(f).second;
160 }
161 poly derivative() const {
162     if (a.empty()) return poly();
163     int n = a.size();
164     vector<int> res(n - 1);
165     for (int i = 0; i < n - 1; ++i)
166         res[i] = 1ll * (i + 1) * a[i + 1] % P;
167     return poly(res);
168 }
169 poly integral() const {
170     if (a.empty()) return poly();
171     int n = a.size();
172     vector<int> res(n + 1);
173     for (int i = 0; i < n; ++i)

```

```

174         res[i + 1] = 111 * a[i] * power(i + 1, P - 2) % P;
175     return poly(res);
176 }
177 poly rev() const {
178     return poly(vector<int>(a.rbegin(), a.rend()));
179 }
180 poly inv(int m) const {
181     poly x(power(a[0], P - 2));
182     int k = 1;
183     while (k < m) {
184         k *= 2;
185         x = (x * (2 - modxk(k) * x)).modxk(k);
186     }
187     return x.modxk(m);
188 }
189 poly log(int m) const {
190     return (derivative() * inv(m)).integral().modxk(m);
191 }
192 poly exp(int m) const {
193     poly x(1);
194     int k = 1;
195     while (k < m) {
196         k *= 2;
197         x = (x * (1 - x.log(k) + modxk(k))).modxk(k);
198     }
199     return x.modxk(m);
200 }
201 poly sqrt(int m) const {
202     poly x(1);
203     int k = 1;
204     while (k < m) {
205         k *= 2;
206         x = (x + (modxk(k) * x.inv(k)).modxk(k)) * ((P + 1) / 2);
207     }
208     return x.modxk(m);
209 }
210 poly sin() const {
211     int g = 3; // g: the ord of P
212     int i = power(g, (P - 1) / 4);
213     poly p = i * (*this);
214     p = p.exp(p.size());
215
216     poly q = (P - i) * (*this);
217     q = q.exp(q.size());
218
219     poly r = (p - q) * power(2 * i % P, P - 2);
220     return r;
221 }
222 poly cos() const {
223     int g = 3; // g: the ord of P
224     int i = power(g, (P - 1) / 4);
225     poly p = i * (*this);
226     p = p.exp(p.size());
227
228     poly q = (P - i) * (*this);
229     q = q.exp(q.size());
230
231     poly r = (p + q) * power(2, P - 2);
232     return r;
233 }
234 poly tan() const {
235     return sin() / cos();
236 }
237 poly cot() const {
238     return cos() / sin();

```



```

239     }
240     poly arcsin() {
241         poly sq = (*this) * (*this).modxk(size());
242         for (int i = 0; i < size(); ++i) sq.a[i] = sq.a[i] ? P - sq.a[i] : 0;
243         sq.a[0] = 1 + sq.a[0];
244         if (sq.a[0] >= P) sq.a[0] -= P;
245         poly r = (derivative() * sq.sqrt(size()).inv(size())).integral();
246         return r;
247     }
248     poly arccos() {
249         poly r = arcsin();
250         for (int i = 0; i < size(); ++i) r.a[i] = r.a[i] ? P - r.a[i] : 0;
251         return r;
252     }
253     poly arctan() {
254         poly sq = (*this) * (*this).modxk(size());
255         sq.a[0] = 1 + sq.a[0];
256         if (sq.a[0] >= P) sq.a[0] -= P;
257         poly r = (derivative() * sq.inv(size())).integral();
258         return r;
259     }
260     poly arccot() {
261         poly r = arctan();
262         for (int i = 0; i < size(); ++i) r.a[i] = r.a[i] ? P - r.a[i] : 0;
263         return r;
264     }
265     poly mult(const poly& b) const {
266         if (b.size() == 0)
267             return poly();
268         int n = b.size();
269         return ((*this) * b.rev()).divxk(n - 1);
270     }
271     pair<poly, poly> divide(const poly& g) const {
272         int n = a.size(), m = g.size();
273         if (n < m) return make_pair(poly(), a);
274
275         poly fR = rev();
276         poly gR = g.rev().alignxk(n - m + 1);
277         poly gRI = gR.inv(gR.size());
278
279         poly qR = (fR * gRI).modxk(n - m + 1);
280
281         poly q = qR.rev();
282
283         poly r = ((*this) - g * q).modxk(m - 1);
284
285         return make_pair(q, r);
286     }
287     vector<int> eval(vector<int> x) const {
288         if (size() == 0)
289             return vector<int>(x.size(), 0);
290         const int n = max(int(x.size()), size());
291         vector<poly> q(4 * n);
292         vector<int> ans(x.size());
293         x.resize(n);
294         function<void(int, int, int)> build = [&](int p, int l, int r) {
295             if (r - l == 1) {
296                 q[p] = vector<int>{1, (P - x[l]) % P};
297             } else {
298                 int m = (l + r) / 2;
299                 build(2 * p, l, m);
300                 build(2 * p + 1, m, r);
301                 q[p] = q[2 * p] * q[2 * p + 1];
302             }
303         };

```

```

304     build(1, 0, n);
305     function<void(int, int, int, const poly &)> work = [&](int p, int l, int r, const poly &num) {
306         if (r - l == 1) {
307             if (l < int(ans.size()))
308                 ans[l] = num[0];
309         } else {
310             int m = (l + r) / 2;
311             work(2 * p, l, m, num.mulT(q[2 * p + 1]).modxk(m - 1));
312             work(2 * p + 1, m, r, num.mulT(q[2 * p]).modxk(r - m));
313         }
314     };
315     work(1, 0, n, mulT(q[1].inv(n)));
316     return ans;
317 }
318 };

```

### 3.19 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; a %= P;
9         while(b) {
10             if (b & 1) res = res * a % P;
11             a = a * a % P;
12             b >>= 1;
13         }
14         return res;
15     }
16
17     const int Gi = qp(G, P - 2);
18     const int I2 = qp(2, P - 2);
19     int r[MAXN];
20     ll t1[MAXN], t2[MAXN], t3[MAXN], t4[MAXN], t5[MAXN], t6[MAXN], t7[MAXN];
21
22     // int N, L;
23     // void init(int n) {
24     //     int N = 1, l = -1; while(N <= n << 1) N <<= 1, l++;
25     //     for(int i = 1; i < N; ++i) r[i] = (r[i >> 1] >> 1) | ((i & 1) << L);
26     // }
27
28     void inplaceNTT(ll *a, int n, int op) {
29         for(int i = 0; i < n; ++i) if(i < r[i]) swap(a[i], a[r[i]]);
30         for(int m2 = 2, m = 1; m2 <= n; m = m2, m2 <<= 1) {
31             ll wn = qp(op == 1 ? G : Gi, (P - 1) / m2), x, y;
32             for(int l = 0; l < n; l += m2) {
33                 ll w = 1;
34                 for(int i = l; i < l + m; ++i) {
35                     x = a[i], y = w * a[i + m] % P;
36                     a[i] = (x + y) % P;
37                     a[i + m] = (x + P - y) % P;
38                     w = w * wn % P;
39                 }
40             }
41         }
42         if (op == -1) {
43             ll inv = qp(n, P - 2);
44             for(int i = 0; i < n; ++i) a[i] = a[i] * inv % P;
45         }
46     }

```

```

47 inline void NTT(ll *a, int n) { inplaceNTT(a, n, 1); }
48 inline void INTT(ll *a, int n) { inplaceNTT(a, n, -1); }
49
50 // 多项式微分 (求导)
51 inline void Derivative(ll *a, ll *b, int n) {
52     for(int i = 0; i < n; ++i) b[i] = a[i + 1] * (i + 1) % P;
53     b[n - 1] = 0;
54 }
55
56 // 多项式积分
57 inline void Integral(ll *a, ll *b, int n) {
58     for(int i = 0; i < n; ++i) b[i + 1] = a[i] * qp(i + 1, P - 2) % P;
59     b[0] = 0;
60 }
61
62 // 多项式翻转
63 //  $b(x) = x^n a(\frac{1}{x})$ 
64 inline void Reverse(ll *a, ll *b, int n) {
65     for (int i = 0; i < n; ++i) b[i] = a[n - i - 1];
66 }
67
68 // 多项式乘法逆
69 //  $b(x) = a^{-1}(x) \bmod x^n$ 
70 void __Inverse(ll *a, ll *b, int n) {
71     if(n == 1) {
72         b[0] = qp(a[0], P - 2);
73         return;
74     }
75
76     __Inverse(a, b, (n + 1) >> 1);
77
78     int N = 1, l = -1; while(N <= n << 1) N <<= 1, l++;
79     for(int i = 1; i < N; ++i) r[i] = (r[i >> 1] >> 1) | ((i & 1) << 1);
80
81     memcpy(t1, a, sizeof(a[0]) * n); fill(t1 + n, t1 + N, 0);
82
83     NTT(t1, N); NTT(b, N);
84     for(int i = 0; i < N; ++i) b[i] = ((b[i] << 1) % P + P - t1[i] * b[i] % P * b[i] % P) % P;
85     INTT(b, N);
86
87     fill(b + n, b + N, 0);
88 }
89
90 inline void Inverse(ll *a, ll *b, int n) {
91     fill(b, b + (n << 2), 0);
92     __Inverse(a, b, n);
93 }
94
95 // 多项式对数函数
96 //  $b(x) = \ln a(x) \bmod x^n$ 
97 void Ln(ll *a, ll *b, int n) {
98     #define aD t3
99     #define aI t4
100
101     Derivative(a, aD, n); Inverse(a, aI, n);
102     int N = 1, l = -1; while(N <= n << 1) N <<= 1, l++;
103     for(int i = 1; i < N; ++i) r[i] = (r[i >> 1] >> 1) | ((i & 1) << 1);
104     NTT(aD, N); NTT(aI, N);
105     for(int i = 0; i < N; ++i) aD[i] = aD[i] * aI[i] % P;
106     INTT(aD, N); Integral(aD, b, n);
107
108     #undef aD
109     #undef aI
110 }
111

```

```

112 // 多项式指数函数
113 //  $b(x) = \exp a(x) \bmod x^n$ 
114 void Exp(ll *a, ll *b, int n) {
115     #define Lnb t2
116
117     if(n == 1) {
118         b[0] = 1;
119         return;
120     }
121     Exp(a, b, (n + 1) >> 1);
122     Ln(b, Lnb, n);
123     int N = 1, l = -1; while(N <= n << 1) N <<= 1, l++;
124     for(int i = 1; i < N; ++i) r[i] = (r[i >> 1] >> 1) | ((i & 1) << 1);
125
126     memcpy(t1, a, sizeof(a[0]) * n); fill(t1 + n, t1 + N, 0);
127     fill(Lnb + n, Lnb + N, 0);
128
129     for(int i = 0; i < N; ++i) t1[i] = ((t1[i] - Lnb[i]) % P + P) % P;
130     ++t1[0];
131     NTT(b, N); NTT(t1, N);
132     for(int i = 0; i < N; ++i) b[i] = b[i] * t1[i] % P;
133     INTT(b, N);
134
135     fill(b + n, b + N, 0);
136     #undef Lnb
137 }
138
139 // 多项式乘法 (卷积)
140 //  $c(x) = a(x) * b(x) \bmod x^{(n+m)}$ 
141 //  $\deg c = n + m - 1$ 
142 void Convolution(ll *a, int n, ll *b, int m, ll *c) {
143     int N = 1, l = -1; while(N <= (n + m) << 1) N <<= 1, l++;
144     for(int i = 1; i < N; ++i) r[i] = (r[i >> 1] >> 1) | ((i & 1) << 1);
145
146     memcpy(t1, a, sizeof(a[0]) * n); fill(t1 + n, t1 + N, 0);
147     memcpy(t2, b, sizeof(b[0]) * m); fill(t2 + m, t2 + N, 0);
148
149     NTT(t1, N); NTT(t2, N);
150     for (int i = 0; i < N; ++i) c[i] = t1[i] * t2[i] % P;
151     INTT(c, N);
152     fill(c + n + m, c + N, 0);
153 }
154 #define Multiply Convolution
155
156 // 多项式除法
157 //  $a(x) = b(x)Q(x) + R(x)$ 
158 //  $\deg Q = n - m + 1$ 
159 //  $\deg R = m - 1$ 
160 void Divide(ll *a, int n, ll *b, int m, ll *Q, ll *R) {
161     #define aR t3
162     #define bR t4
163     #define bRi t5
164     #define QR t6
165     #define bQ t7
166
167     int degQ = n - m + 1;
168     int degR = m - 1;
169
170     Reverse(a, aR, n); Reverse(b, bR, m);
171     for (int i = degQ; i < m; ++i) bR[i] = 0;
172
173     // get Q(x)
174     Inverse(bR, bRi, degQ);
175     Multiply(aR, n, bRi, degQ, QR);
176     Reverse(QR, Q, degQ);

```

```

177
178 // get R(x)
179 Multiply(b, m, Q, degQ, bQ);
180 for (int i = 0; i < degR; ++i) R[i] = (a[i] - bQ[i] + P) % P;
181
182 #undef aR
183 #undef bR
184 #undef bRi
185 #undef QR
186 }
187
188 // 多项式求平方根
189 //  $b^2(x) = a(x)$ 
190 #define bI t3
191 void __Sqrt(ll *a, ll* b, int n) {
192     if (n == 1) {
193         b[0] = 1;
194         return;
195     }
196
197     __Sqrt(a, b, (n + 1) >> 1);
198
199     Inverse(b, bI, n);
200     Multiply(a, n, bI, n, bI);
201     for (int i = 0; i < n; ++i) b[i] = (b[i] + bI[i]) * I2 % P;
202 }
203 inline void Sqrt(ll *a, ll *b, int n) {
204     fill(bI, bI + (n << 2), 0);
205     __Sqrt(a, b, n);
206 }
207 #undef bI
208
209 struct poly {
210     vector<ll> a;
211     int size() const { return a.size(); }
212     int deg() const { return size() - 1; }
213     ll& operator [] (int i) { assert(i < size()); return a[i]; }
214     ll operator [] (int i) const { return i < size() ? a[i] : 0LL; }
215     void reverse() { std::reverse(a.begin(), a.end()); }
216     void resize(int n) { a.resize(n); }
217     poly(int n = 0) : a(n, 0) {}
218
219     void DEBUG() {
220         cerr << "Poly DEBUG: " << endl;
221         for (const ll& v: a) cerr << v << " ";
222         cerr << endl;
223     }
224
225     void DEBUG() const {
226         cerr << "Poly DEBUG: " << endl;
227         for (const ll& v: a) cerr << v << " ";
228         cerr << endl;
229     }
230
231     void input() {
232         for (ll& x: a) read(x);
233     }
234
235     void output() {
236         if (a.empty()) { puts(""); return; }
237         int n = a.size();
238         for (int i = 0; i < n - 1; ++i) printf("%lld ", a[i]);
239         printf("%lld\n", a[n - 1]);
240     }
241 }

```

```

242
243 void output() const {
244     if (a.empty()) { puts(""); return; }
245     int n = a.size();
246     for (int i = 0; i < n - 1; ++i) printf("%lld ", a[i]);
247     printf("%lld\n", a[n - 1]);
248 }
249
250 poly inv(int n = -1) const {
251     if (n == -1) n = size();
252     static ll f[MAXN], g[MAXN];
253     for (int i = 0; i < n; ++i) f[i] = a[i];
254     Inverse(f, g, n);
255     poly res(n);
256     for (int i = 0; i < n; ++i) res[i] = g[i];
257     return res;
258 }
259
260 poly rev() const {
261     int n = size();
262     poly r(n);
263     for (int i = 0; i < n; ++i) r[i] = a[n - i - 1];
264     return r;
265 }
266
267 poly sqrt() {
268     int n = a.size();
269     static ll f[MAXN], g[MAXN];
270     for (int i = 0; i < n; ++i) f[i] = a[i];
271     Sqrt(f, g, n);
272     poly res(n);
273     for (int i = 0; i < n; ++i) res[i] = g[i];
274     return res;
275 }
276 };
277
278 poly operator + (const poly& a, const poly& b) {
279     int k = max(a.size(), b.size());
280     poly c(k);
281     for (int i = 0; i < k; ++i) c[i] = (a[i] + b[i]) % P;
282     return c;
283 }
284
285 poly operator - (const poly& a, const poly& b) {
286     int k = max(a.size(), b.size());
287     poly c(k);
288     for (int i = 0; i < k; ++i) c[i] = (a[i] - b[i] + P) % P;
289     return c;
290 }
291
292 poly operator * (const poly& a, const poly& b) {
293     static ll ta[MAXN], tb[MAXN];
294     int n = a.size(), m = b.size(), k = n + m - 1;
295     for (int i = 0; i < n; ++i) ta[i] = a[i];
296     for (int i = 0; i < m; ++i) tb[i] = b[i];
297
298     Multiply(ta, n, tb, m, ta);
299
300     poly c(k);
301     for (int i = 0; i < k; ++i) c[i] = ta[i];
302     return c;
303 }
304
305 pair<poly, poly> Divide(const poly& a, const poly& b) {
306     static ll ta[MAXN], tb[MAXN], tq[MAXN], tr[MAXN];

```

```

307     int n = a.size(), m = b.size();
308     if (n < m) return make_pair(poly(0), a);
309
310     int degQ = n - m + 1, degR = m - 1;
311     for (int i = 0; i < n; ++i) ta[i] = a[i];
312     for (int i = 0; i < m; ++i) tb[i] = b[i];
313
314     Divide(ta, n, tb, m, tq, tr);
315
316     poly q(degQ); for (int i = 0; i < degQ; ++i) q[i] = tq[i];
317     poly r(degR); for (int i = 0; i < degR; ++i) r[i] = tr[i];
318
319     return make_pair(q, r);
320 }
321
322 poly operator / (const poly &a, const poly &b) { return Divide(a, b).first; }
323 poly operator % (const poly &a, const poly &b) { return Divide(a, b).second; }
324
325
326
327 // given a(x), deg a = n
328 // calc y_i = a(x_i) for i in [0, m), O(n \log^2 n)
329 poly t[N << 2], p[N];
330 void build(int o, int l, int r) {
331     if (l == r) {
332         t[o] = p[l];
333         return;
334     }
335     int mid = (l + r) >> 1;
336     build(o << 1, l, mid);
337     build(o << 1 | 1, mid + 1, r);
338     t[o] = t[o << 1] * t[o << 1 | 1];
339 }
340 void __calcValue(int o, int l, int r, const poly& f, ll *x, ll *y) {
341     // if (l == r) {
342     //     y[l] = f[0];
343     //     return;
344     // }
345     if (r - l <= 75) { // 降低常数 (魔法)
346         for (int i = l; i <= r; ++i) {
347             ll v = 0;
348             for (int j = f.size() - 1; j >= 0; --j)
349                 v = (v * x[i] % P + f[j]) % P;
350             y[i] = v;
351         }
352         return;
353     }
354
355     int mid = (l + r) >> 1, lc = o << 1, rc = o << 1 | 1;
356     __calcValue(lc, l, mid, f % t[lc], x, y);
357     __calcValue(rc, mid + 1, r, f % t[rc], x, y);
358 }
359 void calcValue(const poly& f, ll *x, ll *y, int m) {
360     for (int i = l; i <= m; ++i) {
361         p[i].resize(2);
362         p[i][0] = P - x[i];
363         p[i][1] = 1;
364     }
365     build(1, 1, m);
366     __calcValue(1, 1, m, f % t[1], x, y);
367 }
368 }

```

## 3.20 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) : n(_n), m(_m) {
32          A = vector<vector<double>>(m + 1, vector<double>(n + 1));
33          b = vector<double>(m + 1);
34          x = vector<double>(n + 1);
35          idx = vector<int>(m + 1);
36          idy = vector<int>(n + 1);
37      }
38
39      void input() {
40          for (int i = 1; i <= n; ++i) read(A[0][i]); //  $A_{\{0,i\}} = c_i$ 
41          for (int i = 1; i <= m; ++i) {
42              for (int j = 1; j <= n; ++j) read(A[i][j]);
43              read(b[i]);
44          }
45      }
46
47      void pivot(int x, int y) {
48          swap(idx[x], idy[y]);
49
50          double k = A[x][y];
51          for (int i = 1; i <= n; ++i) A[x][i] /= k;
52          b[x] /= k;
53          A[x][y] = 1 / k;
54
55          for (int i = 0; i <= m; ++i) if (i != x) {
56              k = A[i][y];
57              b[i] -= k * b[x];
58              A[i][y] = 0;
59              for (int j = 1; j <= n; ++j) A[i][j] -= k * A[x][j];
60          }
61      }
62
63      void init(){

```



```

64     flag = OK;
65     idx[0] = INT_MAX; for (int i = 1; i <= m; ++i) idx[i] = n + i;
66     idy[0] = INT_MAX; for (int i = 1; i <= n; ++i) idy[i] = i;
67
68     for(;;) {
69         int x = 0, y = 0;
70         for (int i = 1; i <= m; ++i) if (b[i] < -eps && idx[i] < idx[x]) x = i;
71         if (!x) break;
72
73         for (int i = 1; i <= n; ++i) if (A[x][i] < -eps && idy[i] < idy[y]) y = i;
74         if (!y) { flag = INFEASIBLE; break; }
75
76         pivot(x, y);
77     }
78 }
79
80 void simplex() {
81     for(;;) {
82         int x = 0, y = 0;
83         for (int i = 1; i <= n; ++i) if (A[0][i] > eps && idy[i] < idy[y]) y = i;
84         if (!y) break;
85
86         for (int i = 1; i <= m; ++i) if (A[i][y] > eps) {
87             if (!x) x = i;
88             else {
89                 double delta = b[i] / A[i][y] - b[x] / A[x][y];
90                 if (delta < -eps) x = i;
91                 else if (delta < eps && idx[i] < idx[x]) x = i;
92             }
93         }
94         if (!x) { flag = UNBOUNDED; break; }
95
96         pivot(x, y);
97     }
98     z = -b[0];
99 }
100
101 void work() {
102     init();
103     if (flag == OK) simplex();
104     if (flag == OK) {
105         for (int i = 1; i <= n; ++i) {
106             x[i] = 0;
107             for (int j = 1; j <= m; ++j) if (idx[j] == i) { x[i] = b[j]; break; }
108         }
109     }
110 }
111
112 void DEBUG() {
113     cerr << fixed << setprecision(3);
114     cerr << "Simplex Debug: \n";
115     for (int i = 1; i <= m; ++i) {
116         for (int j = 1; j <= n; ++j) {
117             cerr << A[i][j] << " ";
118         }
119         cerr << "\n";
120     }
121     for (int i = 1; i <= n; ++i) cerr << x[i] << " ";
122     cerr << endl;
123     cerr << "Z = " << z << endl;
124 }
125 };

```

## 3.21 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 }

```

---

## 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) return insert_sort(a, l, r, cmp);
26        int lst = l - 1;
27        for (int i = l; i + 4 <= r; i += 5) {
28            int p = insert_sort(a, i, i + 4, cmp);
29            swap(a[++lst], a[p]);
30        }
31        return kth_index<T>(a, l, lst, (lst - l + 1) / 2 + 1, cmp);
32    }

```

```

33
34 template<typename T, typename Cmp>
35 int partition(T* a, int l, int r, Cmp cmp) {
36     int p = pivot(a, l, r, cmp);
37     swap(a[p], a[r]);
38     int lst = l - 1;
39     for (int i = l; i < r; ++i) {
40         if (cmp(a[i], a[r])) swap(a[++lst], a[i]);
41     }
42     swap(a[++lst], a[r]);
43     return lst;
44 }
45
46 template<typename T, typename Cmp>
47 T kth_index(T* a, int l, int r, int k, Cmp cmp) {
48     int p = partition(a, l, r, cmp);
49     int d = p - l + 1;
50     if (d == k) return p;
51     else if (d < k) return kth_index(a, p + 1, r, k - d, cmp);
52     else return kth_index(a, l, p - 1, k, cmp);
53 }
54
55 template<typename T>
56 T kth_index(T* a, int l, int r, int k) {
57     return kth_index(a, l, r, k, less<T>());
58 }
59 };

```

---

## 4.2 cpp-header

---

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 using ll = int64_t;
5 using ull = uint64_t;
6 using uint = uint32_t;
7 using VI = vector<int>;
8 using VL = vector<ll>;
9 using VVI = vector<vector<int>>;
10 using VVL = vector<vector<ll>>;
11 using PII = pair<int, int>;
12 using PLL = pair<ll, ll>;
13
14 #define REP(i, _, _) for (int i = (_); i < (_); ++i)
15 #define PER(i, _, _) for (int i = (_ - 1); i >= (_); --i)
16 #define FOR(i, _, _) for (int i = (_); i <= (_); ++i)
17 #define ROF(i, _, _) for (int i = (_); i >= (_); --i)
18 #define FC(v, V) for (const auto& v : V)
19 #define FE(v, V) for (auto& v : V)
20
21 #define EB emplace_back
22 #define PB push_back
23 #define MP make_pair
24 #define FI first
25 #define SE second
26 #define SZ(x) (int)((x).size())
27 #define ALL(x) (x).begin(), (x).end()
28 #define LLA(x) (x).rbegin(), (x).rend()
29
30 #define rd read
31 #define pr print
32 #define pf printf
33 #define ps prints
34 #define pln println

```

```
35
36 #ifdef BACKLIGHT
37 #include "debug.h"
38 #else
39 #define debug(...)
40 #endif
41
42 template <typename T>
43 T MIN(T a, T b)
44 {
45     return min(a, b);
46 }
47
48 template <typename First, typename... Rest>
49 First MIN(First f, Rest... r)
50 {
51     return min(f, MIN(r...));
52 }
53
54 template <typename T>
55 T MAX(T a, T b)
56 {
57     return max(a, b);
58 }
59
60 template <typename First, typename... Rest>
61 First MAX(First f, Rest... r)
62 {
63     return max(f, MAX(r...));
64 }
65
66 template <typename T>
67 inline void umin(T& a, const T& b)
68 {
69     if (a > b)
70         a = b;
71 }
72
73 template <typename T>
74 inline void umax(T& a, const T& b)
75 {
76     if (a < b)
77         a = b;
78 }
79
80 ll FIRSTTRUE(ll l, ll r, function<bool(ll)> f)
81 {
82     ll res = l - 1, mid;
83     while (l <= r)
84     {
85         mid = (l + r) >> 1;
86         if (f(mid))
87             r = mid - 1, res = mid;
88         else
89             l = mid + 1;
90     }
91     return res;
92 }
93
94 ll LASTTRUE(ll l, ll r, function<bool(ll)> f)
95 {
96     ll res = l - 1, mid;
97     while (l <= r)
98     {
99         mid = (l + r) >> 1;
```

```

100         if (f(mid))
101             l = mid + 1, res = mid;
102         else
103             r = mid - 1;
104     }
105     return res;
106 }
107
108 const int __BUFFER_SIZE__ = 1 << 20;
109 bool NEOF = 1;
110 int __top;
111 char __buf[__BUFFER_SIZE__], *__p1 = __buf, *__p2 = __buf, __stk[996];
112 inline char nc()
113 {
114     if (!NEOF)
115         return EOF;
116     if (__p1 == __p2)
117     {
118         __p1 = __buf;
119         __p2 = __buf + fread(__buf, 1, __BUFFER_SIZE__, stdin);
120         if (__p1 == __p2)
121         {
122             NEOF = 0;
123             return EOF;
124         }
125     }
126     return *__p1++;
127 }
128
129 #define rd read
130 template <typename T>
131 inline bool read(T& x)
132 {
133     char c = nc();
134     bool f = 0;
135     x = 0;
136     while (!isdigit(c)) c == '-' && (f = 1), c = nc();
137     while (isdigit(c)) x = (x << 3) + (x << 1) + (c ^ 48), c = nc();
138     if (f)
139         x = -x;
140     return NEOF;
141 }
142
143 inline bool need(char c) { return (c != '\n') && (c != ' '); }
144
145 inline bool read(char& a)
146 {
147     while ((a = nc()) && need(a) && NEOF)
148         ;
149     return NEOF;
150 }
151
152 inline bool read(char* a)
153 {
154     while ((*a = nc()) && need(*a) && NEOF) ++a;
155     *a = '\0';
156     return NEOF;
157 }
158
159 inline bool read(double& x)
160 {
161     bool f = 0;
162     char c = nc();
163     x = 0;
164     while (!isdigit(c))

```

```
165     {
166         f |= (c == '-');
167         c = nc();
168     }
169     while (isdigit(c))
170     {
171         x = x * 10.0 + (c ^ 48);
172         c = nc();
173     }
174     if (c == '.')
175     {
176         double temp = 1;
177         c = nc();
178         while (isdigit(c))
179         {
180             temp = temp / 10.0;
181             x = x + temp * (c ^ 48);
182             c = nc();
183         }
184     }
185     if (f)
186         x = -x;
187     return NEOF;
188 }
189
190 template <typename First, typename... Rest>
191 inline bool read(First& f, Rest&... r)
192 {
193     read(f);
194     return read(r...);
195 }
196
197 template <typename T>
198 inline void print(T x)
199 {
200     if (x < 0)
201         putchar('-'), x = -x;
202     if (x == 0)
203     {
204         putchar('0');
205         return;
206     }
207     __top = 0;
208     while (x)
209     {
210         __stk[++__top] = x % 10 + '0';
211         x /= 10;
212     }
213     while (__top)
214     {
215         putchar(__stk[__top]);
216         --__top;
217     }
218 }
219
220 template <typename First, typename... Rest>
221 inline void print(First f, Rest... r)
222 {
223     print(f);
224     putchar(' ');
225     print(r...);
226 }
227
228 template <typename T>
229 inline void prints(T x)
```

```

230 {
231     print(x);
232     putchar(' ');
233 }
234
235 template <typename T>
236 inline void println(T x)
237 {
238     print(x);
239     putchar('\n');
240 }
241
242 template <typename First, typename... Rest>
243 inline void println(First f, Rest... r)
244 {
245     print(f);
246     putchar(' ');
247     println(r...);
248 }
249
250 template <typename T>
251 void println(const vector<T>& V)
252 {
253     for (const auto& v : V) print(v), putchar(' ');
254     putchar('\n');
255 }
256
257 mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
258 int rnd(int l, int r) { return l + rng() % (r - l + 1); }
259
260 const int N = 5e5 + 5;
261 const int M = 3e6 + 5;
262 const int K = 1e7 + 5;
263 const int MOD = 1e9 + 7; // 998244353 1e9 + 7
264 const int INF = 0x3f3f3f3f; // 1e9 + 7 0x3f3f3f3f
265 const ll LLINF = 0x3f3f3f3f3f3f3f3f; // 1e18 + 9 0x3f3f3f3f3f3f3f3f
266 const double EPS = 1e-8;
267 const double PI = acos(-1.0);
268
269 int qp(int a, int b, int p = MOD)
270 {
271     int r = 1;
272     for (; b; b >>= 1)
273     {
274         if (b & 1)
275             r = 1ll * r * a % p;
276         a = 1ll * a * a % p;
277     }
278     return r;
279 }
280
281 void solve(int Case)
282 {
283     /* write code here */
284     /* gl & hf */
285 }
286
287 int main()
288 {
289     #ifdef BACKLIGHT
290         freopen("a.in", "r", stdin);
291         // freopen("a.out", "w", stdout);
292         auto begin = std::chrono::steady_clock::now();
293     #endif
294     int T = 1;

```

```

295     rd(T);
296     for (int _ = 1; _ <= T; _++) solve(_);
297
298     #ifdef BACKLIGHT
299         auto end = std::chrono::steady_clock::now();
300         auto duration =
301             std::chrono::duration_cast<std::chrono::milliseconds>(end - begin);
302         cerr << "\033[32mTime Elapsed: " << duration.count() << " ms\033[0m"
303             << endl;
304     #endif
305     return 0;
306 }

```

---

### 4.3 debug

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  using ll = int64_t;
5  using ull = uint64_t;
6  using uint = uint32_t;
7  using VI = vector<int>;
8  using VL = vector<ll>;
9  using VVI = vector<vector<int>>>;
10 using VVL = vector<vector<ll>>>;
11 using PII = pair<int, int>;
12 using PLL = pair<ll, ll>;
13
14 #define REP(i, _, __) for (int i = (_); i < (__); ++i)
15 #define PER(i, _, __) for (int i = (_ - 1); i >= (__); --i)
16 #define FOR(i, _, __) for (int i = (_); i <= (__); ++i)
17 #define ROF(i, _, __) for (int i = (_); i >= (__); --i)
18 #define FC(v, V) for (const auto& v : V)
19 #define FE(v, V) for (auto& v : V)
20
21 #define EB emplace_back
22 #define PB push_back
23 #define MP make_pair
24 #define FI first
25 #define SE second
26 #define SZ(x) (int)((x).size())
27 #define ALL(x) (x).begin(), (x).end()
28 #define LLA(x) (x).rbegin(), (x).rend()
29
30 #define rd read
31 #define pr print
32 #define pf printf
33 #define ps prints
34 #define pln println
35
36 #ifdef BACKLIGHT
37     #include "debug.h"
38 #else
39     #define debug(...)
40 #endif
41
42 template <typename T>
43 T MIN(T a, T b)
44 {
45     return min(a, b);
46 }
47
48 template <typename First, typename... Rest>
49 First MIN(First f, Rest... r)

```



```

50 {
51     return min(f, MIN(r...));
52 }
53
54 template <typename T>
55 T MAX(T a, T b)
56 {
57     return max(a, b);
58 }
59
60 template <typename First, typename... Rest>
61 First MAX(First f, Rest... r)
62 {
63     return max(f, MAX(r...));
64 }
65
66 template <typename T>
67 inline void umin(T& a, const T& b)
68 {
69     if (a > b)
70         a = b;
71 }
72
73 template <typename T>
74 inline void umax(T& a, const T& b)
75 {
76     if (a < b)
77         a = b;
78 }
79
80 ll FIRSTTRUE(ll l, ll r, function<bool(ll)> f)
81 {
82     ll res = l - 1, mid;
83     while (l <= r)
84     {
85         mid = (l + r) >> 1;
86         if (f(mid))
87             r = mid - 1, res = mid;
88         else
89             l = mid + 1;
90     }
91     return res;
92 }
93
94 ll LASTTRUE(ll l, ll r, function<bool(ll)> f)
95 {
96     ll res = l - 1, mid;
97     while (l <= r)
98     {
99         mid = (l + r) >> 1;
100         if (f(mid))
101             l = mid + 1, res = mid;
102         else
103             r = mid - 1;
104     }
105     return res;
106 }
107
108 const int __BUFFER_SIZE__ = 1 << 20;
109 bool NEOF = 1;
110 int __top;
111 char __buf[__BUFFER_SIZE__], *__p1 = __buf, *__p2 = __buf, __stk[996];
112 inline char nc()
113 {
114     if (!NEOF)

```

```

115     return EOF;
116     if (__p1 == __p2)
117     {
118         __p1 = __buf;
119         __p2 = __buf + fread(__buf, 1, __BUFFER_SIZE__, stdin);
120         if (__p1 == __p2)
121         {
122             NEOF = 0;
123             return EOF;
124         }
125     }
126     return *__p1++;
127 }
128
129 #define rd read
130 template <typename T>
131 inline bool read(T& x)
132 {
133     char c = nc();
134     bool f = 0;
135     x = 0;
136     while (!isdigit(c)) c == '-' && (f = 1), c = nc();
137     while (isdigit(c)) x = (x << 3) + (x << 1) + (c ^ 48), c = nc();
138     if (f)
139         x = -x;
140     return NEOF;
141 }
142
143 inline bool need(char c) { return (c != '\n') && (c != ' '); }
144
145 inline bool read(char& a)
146 {
147     while ((a = nc()) && need(a) && NEOF)
148         ;
149     return NEOF;
150 }
151
152 inline bool read(char* a)
153 {
154     while ((*a = nc()) && need(*a) && NEOF) ++a;
155     *a = '\0';
156     return NEOF;
157 }
158
159 inline bool read(double& x)
160 {
161     bool f = 0;
162     char c = nc();
163     x = 0;
164     while (!isdigit(c))
165     {
166         f |= (c == '-');
167         c = nc();
168     }
169     while (isdigit(c))
170     {
171         x = x * 10.0 + (c ^ 48);
172         c = nc();
173     }
174     if (c == '.')
175     {
176         double temp = 1;
177         c = nc();
178         while (isdigit(c))
179         {

```

```
180         temp = temp / 10.0;
181         x = x + temp * (c ^ 48);
182         c = nc();
183     }
184 }
185 if (f)
186     x = -x;
187 return NEOF;
188 }
189
190 template <typename First, typename... Rest>
191 inline bool read(First& f, Rest&... r)
192 {
193     read(f);
194     return read(r...);
195 }
196
197 template <typename T>
198 inline void print(T x)
199 {
200     if (x < 0)
201         putchar('-'), x = -x;
202     if (x == 0)
203     {
204         putchar('0');
205         return;
206     }
207     __top = 0;
208     while (x)
209     {
210         __stk[++__top] = x % 10 + '0';
211         x /= 10;
212     }
213     while (__top)
214     {
215         putchar(__stk[__top]);
216         --__top;
217     }
218 }
219
220 template <typename First, typename... Rest>
221 inline void print(First f, Rest... r)
222 {
223     print(f);
224     putchar(' ');
225     print(r...);
226 }
227
228 template <typename T>
229 inline void prints(T x)
230 {
231     print(x);
232     putchar(' ');
233 }
234
235 template <typename T>
236 inline void println(T x)
237 {
238     print(x);
239     putchar('\n');
240 }
241
242 template <typename First, typename... Rest>
243 inline void println(First f, Rest... r)
244 {
```

```

245     print(f);
246     putchar(' ');
247     println(r...);
248 }
249
250 template <typename T>
251 void println(const vector<T>& V)
252 {
253     for (const auto& v : V) print(v), putchar(' ');
254     putchar('\n');
255 }
256
257 mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
258 int rnd(int l, int r) { return l + rng() % (r - l + 1); }
259
260 const int N = 5e5 + 5;
261 const int M = 3e6 + 5;
262 const int K = 1e7 + 5;
263 const int MOD = 1e9 + 7;           // 998244353 1e9 + 7
264 const int INF = 0x3f3f3f3f;       // 1e9 + 7 0x3f3f3f3f
265 const ll LLINF = 0x3f3f3f3f3f3f3f3f; // 1e18 + 9 0x3f3f3f3f3f3f3f3f
266 const double EPS = 1e-8;
267 const double PI = acos(-1.0);
268
269 int qp(int a, int b, int p = MOD)
270 {
271     int r = 1;
272     for (; b; b >>= 1)
273     {
274         if (b & 1)
275             r = 1ll * r * a % p;
276         a = 1ll * a * a % p;
277     }
278     return r;
279 }
280
281 void solve(int Case)
282 {
283     /* write code here */
284     /* gl & hf */
285 }
286
287 int main()
288 {
289     #ifdef BACKLIGHT
290         freopen("a.in", "r", stdin);
291         // freopen("a.out", "w", stdout);
292         auto begin = std::chrono::steady_clock::now();
293     #endif
294     int T = 1;
295     rd(T);
296     for (int _ = 1; _ <= T; _++) solve(_);
297
298     #ifdef BACKLIGHT
299         auto end = std::chrono::steady_clock::now();
300         auto duration =
301             std::chrono::duration_cast<std::chrono::milliseconds>(end - begin);
302         cerr << "\033[32mTime Elapsed: " << duration.count() << " ms\033[0m"
303              << endl;
304     #endif
305     return 0;
306 }

```

## 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) : n(_n), X(n), Y(n), W(n) {}
10
11  void input() {
12      for (int i = 0; i < n; ++i) {
13          read(X[i], Y[i], W[i]);
14      }
15  }
16
17  double cost(double x, double y) {
18      double res = 0;
19      for (int i = 0; i < n; ++i) {
20          double dx = X[i] - x;
21          double dy = Y[i] - y;
22          double d = sqrt(dx * dx + dy * dy);
23          res += d * W[i];
24      }
25      return res;
26  }
27
28  void init() {
29      ax = 0; ay = 0;
30      for (int i = 0; i < n; ++i) ax += X[i], ay += Y[i];
31      ax /= n; ay /= n;
32  }
33
34  void simulate_anneal() {
35      srand(time(0));
36      double T = 1e6, TE = 1e-8;
37      double cx = ax, cy = ay, cc = cost(cx, cy);
38      while(T > TE) {
39          double nx = ax + (2 * Rand() - 1) * T;
40          double ny = ay + (2 * Rand() - 1) * T;
41
42          double nc = cost(nx, ny);
43          double d = nc - cc;
44
45          if (d < 0) cc = nc, ax = cx = nx, ay = cy = ny;
46          else if (exp(-d / T) > Rand()) {
47              cx = nx;
48              cy = ny;
49          }
50
51          T *= p;
52      }
53  }
54
55  void work() {
56      init();
57      // try a try, AC is ok.
58      simulate_anneal();
59      simulate_anneal();
60      simulate_anneal();
61      simulate_anneal();
62  }
63 };
```

## 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) { return c - 'a'; }
11
12    inline void init() {
13        tot = 0;
14        memset(tr[0], 0, sizeof(tr[0]));
15        f[0] = e[0] = 0;
16    }
17
18    inline int newnode() {
19        ++tot;
20        memset(tr[tot], 0, sizeof(tr[tot]));
21        f[tot] = e[tot] = 0;
22        return tot;
23    }
24
25    void insert(char* s, int n, int id) {
26        int p = 0, c;
27        for (int i = 0; i < n; ++i) {
28            c = idx(s[i]);
29            if (!tr[p][c]) tr[p][c] = newnode();
30            p = tr[p][c];
31            ++f[p];
32        }
33        ++e[p];
34
35        eid[id] = p;
36        st[p].insert(0);
37    }
38 }
39
40 // 字典图优化
41 // void getfail() {
42 //     queue<int> q;
43 //     for (int i = 0; i < __M; ++i) if (tr[0][i]) fail[tr[0][i]] = 0, q.push(tr[0][i]);
44 //     while(!q.empty()) {
45 //         int p = q.front(); q.pop();
46 //         for (int c = 0; c < __M; ++c) {
47 //             int nxt = tr[p][c];
48 //             if (nxt) fail[nxt] = tr[fail[p]][c], q.push(nxt);
49 //             else nxt = tr[fail[p]][c];
50 //         }
51 //     }
52 // }
53
54 // int query(char* t) {
55 //     int n = strlen(t), p = 0, res = 0;
56 //     for (int i = 0; i < n; ++i) {
57 //         p = tr[p][t[i] - 'a'];
58 //         for (int j = p; j && e[j] != -1; j = fail[j]) res += e[j], e[j] = -1;
59 //     }
60 //     return res;
61 // }

```

```

62
63 // 跳 fail 链
64 void getfail() {
65     queue<int> q;
66     fail[0] = 0;
67     for (int c = 0; c < __M; ++c) if (tr[0][c]) fail[tr[0][c]] = last[tr[0][c]] = 0, q.push(tr[0][c]);
68     while(!q.empty()) {
69         int p = q.front(); q.pop();
70         for (int c = 0; c < __M; ++c) {
71             int u = tr[p][c];
72             if (u) {
73                 q.push(u);
74                 int v = fail[p];
75                 while(v && !tr[v][c]) v = fail[v];
76                 fail[u] = tr[v][c];
77                 last[u] = e[fail[u]] ? fail[u] : last[fail[u]];
78             }
79         }
80     }
81 }
82
83 int queryMax(char* t, int n) {
84     int p = 0, res = -1, c;
85     for (int i = 0; i < n; ++i) {
86         c = idx(t[i]);
87         while(p && !tr[p][c]) p = fail[p];
88         p = tr[p][c];
89         for (int j = p; j; j = last[j]) if (e[j]) updMax(res, (*st[j].rbegin()));
90     }
91     return res;
92 }
93 } // namespace ACAM

```

## 5.2 GSAM

```

1 namespace GSAM {
2     using T = char;
3
4     inline int idx(T c) { return c - 'a'; }
5
6     const int __N = N << 1;
7     const int __M = 26;
8
9     int tot, next[__N][__M];
10    int len[__N], fail[__N];
11
12    inline void init() {
13        tot = 0;
14        fail[0] = -1; len[0] = 0;
15        memset(next[0], 0, sizeof(next[0]));
16    }
17
18    inline int newnode() {
19        ++tot;
20        fail[tot] = 0; len[tot] = 0;
21        memset(next[tot], 0, sizeof(next[tot]));
22        return tot;
23    }
24
25    void insertTrie(const T* s, int n) {
26        int p = 0, c;
27        for (int i = 0; i < n; ++i) {
28            c = idx(s[i]);
29            if (!next[p][c]) next[p][c] = newnode();

```



```

30         p = next[p][c];
31     }
32 }
33
34 int extendSAM(int last, int c) {
35     int cur = next[last][c];
36     if (len[cur]) return cur;
37     len[cur] = len[last] + 1;
38
39     int p = fail[last];
40     while(p != -1) {
41         if (!next[p][c]) next[p][c] = cur;
42         else break;
43         p = fail[p];
44     }
45
46     if (p == -1) {
47         fail[cur] = 0;
48         return cur;
49     }
50
51     int q = next[p][c];
52     if (len[p] + 1 == len[q]) {
53         fail[cur] = q;
54         return cur;
55     }
56
57     int clone = newnode();
58     for (int i = 0; i < __M; ++i)
59         next[clone][i] = len[next[q][i]] ? next[q][i] : 0;
60
61     len[clone] = len[p] + 1;
62     while(p != -1 && next[p][c] == q) {
63         next[p][c] = clone;
64         p = fail[p];
65     }
66     fail[clone] = fail[q];
67     fail[cur] = clone;
68     fail[q] = clone;
69     return cur;
70 }
71
72 void build() {
73     queue<pair<int, int>> q;
74     for (int i = 0; i < __M; ++i)
75         if (next[0][i]) q.push(make_pair(0, i));
76
77     while(!q.empty()) {
78         pair<int, int> u = q.front(); q.pop();
79         int last = extendSAM(u.first, u.second);
80         for (int i = 0; i < __M; ++i)
81             if (next[last][i]) q.push(make_pair(last, i));
82     }
83 }
84
85 // 多模式串--本质不同子串数
86 ll count() {
87     ll res = 0;
88     for (int i = 1; i <= tot; ++i)
89         res += len[i] - len[fail[i]];
90     return res;
91 }
92 }

```

## 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]) j = pi[j - 1];
8             if (s[i] == s[j]) ++j;
9             pi[i] = j;
10        }
11    }
12
13    vector<int> getAllMatchPosition(char* s, int n, int* pi, char* t, int m) {
14        s[n] = '#'; s[n + 1] = 0; ++n;
15        KMP::getPi(s, n, pi);
16
17        vector<int> ans;
18
19        int p = 0;
20        for (int i = 0; i < m; ++i) {
21            while(p > 0 && t[i] != s[p]) p = pi[p - 1];
22            if (t[i] == s[p]) {
23                ++p;
24                if (p == n - 1) {
25                    ans.push_back(i + 2 - n);
26                }
27            }
28        }
29
30        return ans;
31    }
32
33    int getPeriod(int n, int* pi) {
34        return n - pi[n - 1];
35    }
36 }

```

---

## 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] = '@'; s[n] = '#'; s[n + 1] = 0;
14        for (int i = 1; i <= m; ++i) {
15            s[2 * i - 1] = '#';
16            s[2 * i] = t[i];
17        }
18    }
19
20    // s[i-len[i]...i+len[i]] is palindromic
21    // len[i]-1 is palindromic length in t
22    void manacher(char* t, int m) {

```

---

```

23     init(t, m);
24     for (int i = 1, l = 0, r = 0, k; i <= n; ++i) {
25         k = i > r ? 1 : min(r - i, len[l + r - i]);
26         while(s[i - k] == s[i + k]) ++k;
27         len[i] = k--;
28         if (i + k > r) {
29             l = i - k;
30             r = i + k;
31         }
32     }
33 }
34
35 int getMaxPalindromicLength(char* t, int m) {
36     manacher(t, m);
37     int ma = 0;
38     for (int i = 1; i <= n; ++i) updMax(ma, len[i]);
39     return ma - 1;
40 }
41 }

```

## 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])x=fail[x];
32        return x;
33    }
34    void extend(int c){
35        s[now++]=c;
36        int cur=getfail(last);
37        if(!next[cur][c]){
38            int p=newnode();len[p]=len[cur]+2;
39            fail[p]=next[getfail(fail[cur])][c];
40            next[cur][c]=p;
41
42            // extend

```

```

43         if(len[p]<=2)trans[p]=fail[p];
44     else{
45         int tmp=trans[cur];
46         while(s[now-len[tmp]-2] != s[now-1] || (len[tmp]+2)*2>len[p])tmp=fail[tmp];
47         trans[p]=next[tmp][c];
48     }
49 }
50 last=next[cur][c];
51 cnt[last]++;
52 }
53 int count(){return tot-1;}
54 void calc(){
55     for(int i=tot;i>=2;--i) cnt[fail[i]]+=cnt[i];
56     cnt[0]=cnt[1]=0;
57 }
58 int getans(){
59     int ans=0;
60     for(int i=2;i<=tot;i++){
61         if(len[i]>ans && len[trans[i]]*2==len[i] && len[trans[i]]%2==0)ans=len[i];
62     }
63     return ans;
64 }
65 }pam;
66
67 char t[N];
68
69 int main()
70 {
71     int n;
72     scanf("%d",&n);
73     scanf("%s",t);
74     pam.init();
75     for(int i=0;i<n;++i){
76         pam.extend(t[i]-'a');
77     }
78     printf("%d\n",pam.getans());
79     return 0;
80 }

```

## 5.6 SA

```

1 namespace SA {
2     // 0 based, 倍增法构建,  $O(n\log n)$ 
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++) c[i] = 0;
8         for (i = 0; i < n; i++) c[x[i] = s[i]]++;
9         for (i = 1; i < m; i++) c[i] += c[i - 1];
10        for (i = n - 1; i >= 0; i--) sa[--c[x[i]]] = i;
11        for (j = 1; j <= n; j <= 1) {
12            k = 0;
13            for (i = n - j; i < n; i++) y[k++] = i;
14            for (i = 0; i < n; i++) if (sa[i] >= j) y[k++] = sa[i] - j;
15            for (i = 0; i < m; i++) c[i] = 0;
16            for (i = 0; i < n; i++) c[x[y[i]]]++;
17            for (i = 1; i < m; i++) c[i] += c[i - 1];
18            for (i = n - 1; i >= 0; i--) sa[--c[x[y[i]]]] = y[i];
19            swap(x, y);
20            m = 0;
21            x[sa[0]] = m++;
22            for (i = 1; i < n; i++) {
23                if (y[sa[i]] == y[sa[i - 1]] && y[sa[i] + j] == y[sa[i - 1] + j])

```

```

24         x[sa[i]] = m - 1;
25     else
26         x[sa[i]] = m++;
27     }
28     if (m >= n) break;
29 }
30 k = 0;
31 for (i = 0; i < n; i++) rk[sa[i]] = i;
32 for (i = 0; i < n - 1; i++) {
33     if (k) k--;
34     j = sa[rk[i] - 1];
35     while (s[i + k] == s[j + k]) k++;
36     height[rk[i]] = k;
37 }
38 }
39 }

```

## 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     {
6         int m = *max_element(S + 1, S + 1 + n);
7         for (int i = 1; i <= n; ++i)
8             rk[S[i]] = 1;
9         for (int i = 1; i <= m; ++i)
10             rk[i] += rk[i - 1];
11         for (int i = 1; i <= n; ++i)
12             s[i] = rk[S[i]];
13         return rk[m];
14     }
15     #define ps(x) sa[w[s[x]]--] = x
16     #define pl(x) sa[w[s[x]]++] = x
17     inline void radix(int* v, int* s, int* t, int n, int m, int n1)
18     {
19         memset(sa, 0, n + 1 << 2);
20         memset(c, 0, m + 1 << 2);
21         for (int i = 1; i <= n; ++i)
22             ++c[s[i]];
23         for (int i = 1; i <= m; ++i)
24             w[i] = c[i] + c[i - 1];
25         for (int i = n1; i; --i)
26             ps(v[i]);
27         for (int i = 1; i <= m; ++i)
28             w[i] = c[i - 1] + 1;
29         for (int i = 1; i <= n; ++i)
30             if (sa[i] > 1 && t[sa[i] - 1])
31                 pl(sa[i] - 1);
32         for (int i = 1; i <= m; ++i)
33             w[i] = c[i];
34         for (int i = n; i; --i)
35             if (sa[i] > 1 && !t[sa[i] - 1])
36                 ps(sa[i] - 1);
37     }
38     inline void SAIS(int n, int m, int* s, int* t, int* p)
39     {
40         int n1 = 0, ch = rk[1] = 0, *s1 = s + n;
41         t[n] = 0;
42         for (int i = n - 1; i; --i)
43             t[i] = s[i] == s[i + 1] ? t[i + 1] : s[i] > s[i + 1];
44         for (int i = 2; i <= n; ++i)
45             rk[i] = t[i - 1] && !t[i] ? (p[++n1] = i, n1) : 0;

```

```

46     radix(p, s, t, n, m, n1);
47     for (int i = 1, x, y; i <= n; ++i)
48         if (x = rk[sa[i]]) {
49             if (ch <= 1 || p[x + 1] - p[x] != p[y + 1] - p[y])
50                 ++ch;
51             else
52                 for (int j = p[x], k = p[y]; j <= p[x + 1]; ++j, ++k)
53                     if ((s[j] << 1 | t[j]) ^ (s[k] << 1 | t[k])) {
54                         ++ch;
55                         break;
56                     }
57             s1[y = x] = ch;
58         }
59     if (ch < n1)
60         SAIS(n1, ch, s1, t + n, p + n1);
61     else
62         for (int i = 1; i <= n1; ++i)
63             sa[s1[i]] = i;
64     for (int i = 1; i <= n1; ++i)
65         s1[i] = p[sa[i]];
66     radix(s1, s, t, n, m, n1);
67 }
68 inline void build_sa(int* S, int n)
69 {
70     int m = trans(++n, S);
71     SAIS(n, m, s, t, p);
72     for (int i = 1; i < n; ++i)
73         rk[sa[i] = sa[i + 1]] = i;
74     for (int i = 1, j, k = 0; i < n; ++i)
75         if (rk[i] > 1) {
76             for (j = sa[rk[i] - 1]; S[i + k] == S[j + k]; ++k)
77                 ;
78             if (height[rk[i]] = k)
79                 --k;
80         }
81 }
82 }

```

## 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 {
11     //初始状态为 0,range[0...tot-1]
12     struct state{
13         int len,link;
14         map<char,int>next;
15     }st[maxn<<1];
16     int last,tot;
17
18     void init(){
19         st[0].len=0;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)st[cur].link=0;
33     else{
34         int q=st[p].next[c];
35         if(st[p].len+1==st[q].len)st[cur].link=q;
36         else{
37             int clone=tot++;
38             st[clone].len=st[p].len+1;
39             st[clone].next=st[q].next;
40             st[clone].link=st[q].link;
41             while(p!=-1 && st[p].next[c]==q){
42                 st[p].next[c]=clone;
43                 p=st[p].link;
44             }
45             st[q].link=st[cur].link=clone;
46         }
47     }
48     last=cur;
49 }
50
51 ll count(){
52     ll res=0;
53     for(int i=0;i<tot;i++)res+=st[i].len-st[st[i].link].len;
54     return res;
55 }
56 } sam;
57
58 int main()
59 {
60     scanf("%s",s);
61     sam.init();
62     for(int i=0;s[i]!=0;i++)sam.extend(s[i]);
63     printf("%lld\n",sam.count());
64     return 0;
65 }

```

## 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++) lst[i] = 1;
11     }
12
13     void extend(int c){
14         int p = lst[c], np = ++tot;
15         pre[np] = p;
16         for (int i = 0; i < 26; i++)
17             for (int j = lst[i]; j && !next[j][c]; j = pre[j])
18                 next[j][c] = np;
19         lst[c]=np;
20     }

```

---

21 };

## 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++) p1[i] = (p1[i - 1] * P1) % MOD1;
12         for(int i = 1; i <= n; i++) p2[i] = (p2[i - 1] * P2) % MOD2;
13         for(int i = 1; i <= n; i++) h1[i] = (h1[i - 1] * P1 + s[i]) % MOD1;
14         for(int i = 1; i <= n; i++) h2[i] = (h2[i - 1] * P2 + s[i]) % MOD2;
15     }
16
17     ll get_hash(int l, int r) {
18         ll H1 = ((h1[r] - h1[l - 1] * p1[r - l + 1]) % MOD1 + MOD1) % MOD1;
19         ll H2 = ((h2[r] - h2[l - 1] * p2[r - l + 1]) % MOD2 + MOD2) % MOD2;
20         return H1 * MOD2 + H2;
21     }
22 }
```

---

## 5.11 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) { return c - 'a'; }
23
24     void insert(char* s) {
25         int n = strlen(s + 1), p = 0, c;
26         for (int i = 1; i <= n; ++i) {
27             c = idx(s[i]);
28             if (!ch[p][c]) ch[p][c] = newnode();
29             p = ch[p][c];
30             ++f[p];
31         }
32         ++e[p];
33     }
```



```

34
35     int query(char* s) {
36         int p = 0, n = strlen(s + 1), c;
37         for(int i = 1; i <= n; i++){
38             c = idx(s[i]);
39             if(!ch[p][c]) return 0;
40             p = ch[p][c];
41         }
42         return e[p];
43     }
44 }

```

---

## 5.12 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) z[i] = min(r - i + 1, z[i - l + 1]);
9              else z[i] = 0;
10             while(i + z[i] <= n && s[z[i] + 1] == s[i + z[i]]) ++z[i];
11             if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
12         }
13     }
14
15     // p_i = LCP(s, t[i..m])
16     void EXKMP(char* s, int n, int* z, char* t, int m, int* p) {
17         getZ(s, n, z);
18         for (int i = 1, l = 0, r = 0; i <= m; ++i) {
19             if (i <= r) p[i] = min(r - i + 1, z[i - l + 1]);
20             else p[i] = 0;
21             while(i + p[i] <= m && s[p[i] + 1] == t[i + p[i]]) ++p[i];
22             if (i + p[i] - 1 > r) l = i, r = i + p[i] - 1;
23         }
24     }
25 }

```

---