

Notes and Tricks

所有数与某值的关系 \Leftrightarrow 极值与某数的关系

找规律

dp 初始化:

- 恰好 $\rightarrow -\text{INF}$
- 不多于 $\rightarrow 0$

初始化为负无穷 (起点状态为 0) 可以保证答案总是由起点转移得到, 即“装满”

循环对称的关系 \rightarrow 种类并查集, 即翻倍的并查集, 注意调整合并操作

Data Structure

Balanced BST

```
1 // 改进版替罪羊树，在另外一些细节上也进行了一些更改，具体看注释
2 /**
3  * 插入一个整数 x。
4  * 删除一个整数 x（若有多个相同的数，只删除一个）。
5  * 查询整数 x 的排名（排名定义为比当前数小的数的个数 +1）。
6  * 查询排名为 x 的数（如果不存在，则认为是排名小于 x 的最大数。保证 x 不会超过当前数据结构中数的总数）。
7  * 求 x 的前驱（小于 x，且最大的数）。
8  * 求 x 的后继（大于 x，且最小的数）。
9  */
10 #include <bits/stdc++.h>
11 using namespace std;
12 #define ls(x) tree[x].ls
13 #define rs(x) tree[x].rs
14 #define num(x) tree[x].num
15 #define val(x) tree[x].val
16 #define sz(x) tree[x].sz
17 #define exist(x) !(num(x) == 0 && ls(x) == 0 && rs(x) == 0)
18 const double ALPHA = 0.7;
19 const int MAXN = 2e6 + 5;
20 int n, m;
21 struct Node {
22     int ls, rs, num, val, sz;
23 } tree[MAXN]; // 改用结构体进行存储
24 vector<int> FP, FN, FV; // 存储拉平后的节点编号、数目、值
25 int cnt = 1;
26 // 一趟中序遍历，把当前子树拉平并存到 vector 里，返回当前节点的索引
27 int flatten(int pos) {
28     if (exist(ls(pos))) // 递归地拉平左子树
29         flatten(ls(pos));
30     int id = FP.size(); // 记下当前节点的索引
31     // 如果该节点是已被删除的节点，就略过，否则把相应信息存入 vector
32     if (num(pos) != 0) {
33         FP.push_back(pos);
34         FV.push_back(val(pos));
35         FN.push_back(num(pos));
36     }
37     // 递归地拉平右子树
38     if (exist(rs(pos))) flatten(rs(pos));
39     return id;
40 }
41 // 以 pos 为根节点，以 [l,r] 内的信息重建一棵平衡的树
42 void rebuild(int pos, int l = 0, int r = FP.size() - 1) {
43     int mid = (l + r) / 2, sz1 = 0, sz2 = 0;
44     if (l < mid) {
45         ls(pos) = FP[(l + mid - 1) / 2]; // 重用节点编号
46         rebuild(ls(pos), l, mid - 1); // 递归地重建
47         sz1 = sz(ls(pos));
48     } else {
49         rs(pos) = 0;
50     }
```

```

51     if (mid < r) {
52         rs(pos) = FP[(mid + 1 + r) / 2];
53         rebuild(rs(pos), mid + 1, r);
54         sz2 = sz(rs(pos));
55     } else {
56         rs(pos) = 0;
57     }
58     num(pos) = FN[mid]; // 把存于 vector 中的信息复制过来
59     val(pos) = FV[mid];
60     sz(pos) = sz1 + sz2 + num(pos); // 递归确定重建后树的大小
61 }
62 // 尝试重构当前子树
63 void try_restructure(int pos) {
64     double k = max(sz(ls(pos)), sz(rs(pos))) / double(sz(pos));
65     if (k > ALPHA) {
66         FP.clear(), FV.clear(), FN.clear(); // 清空 vector
67         int id = flatten(pos);
68         // 这里是确保当前节点的编号在重构后不会改变
69         swap(FP[id], FP[(FP.size() - 1) / 2]);
70         rebuild(pos);
71     }
72 }
73 // 接下来是普通的二叉查找树
74 void bst_insert(int v, int pos = 1) {
75     if (!exist(pos)) {
76         val(pos) = v;
77         num(pos) = 1;
78     } else if (v < val(pos)) {
79         if (!exist(ls(pos))) ls(pos) = ++cnt;
80         bst_insert(v, ls(pos));
81     } else if (v > val(pos)) {
82         if (!exist(rs(pos))) rs(pos) = ++cnt;
83         bst_insert(v, rs(pos));
84     } else
85         num(pos)++;
86     sz(pos)++;
87     try_restructure(pos);
88 }
89 void bst_remove(int v, int pos = 1) {
90     sz(pos)--;
91     if (v < val(pos))
92         bst_remove(v, ls(pos));
93     else if (v > val(pos))
94         bst_remove(v, rs(pos));
95     else
96         num(pos)--;
97     try_restructure(pos);
98 }
99 int bst_countl(int v, int pos = 1) {
100     if (v < val(pos))
101         return exist(ls(pos)) ? bst_countl(v, ls(pos)) : 0;
102     else if (v > val(pos))
103         return sz(ls(pos)) + num(pos) + (exist(rs(pos)) ? bst_countl(v, rs(pos)) : 0);
104     else
105         return sz(ls(pos));
106 }
107 int bst_countg(int v, int pos = 1) {
108     if (v > val(pos))
109         return exist(rs(pos)) ? bst_countg(v, rs(pos)) : 0;
110     else if (v < val(pos))
111         return sz(rs(pos)) + num(pos) + (exist(ls(pos)) ? bst_countg(v, ls(pos)) : 0);

```

```

112     else
113         return sz(rs(pos));
114 }
115 int bst_rank(int v) { return bst_countl(v) + 1; }
116 int bst_kth(int k, int pos = 1) {
117     if (sz(ls(pos)) + 1 > k)
118         return bst_kth(k, ls(pos));
119     else if (sz(ls(pos)) + num(pos) < k)
120         return bst_kth(k - sz(ls(pos)) - num(pos), rs(pos));
121     else
122         return val(pos);
123 }
124 int bst_pre(int v) {
125     int r = bst_countl(v);
126     return bst_kth(r);
127 }
128 int bst_suc(int v) {
129     int r = sz(1) - bst_countg(v) + 1;
130     return bst_kth(r);
131 }
132 int main() {
133     ios::sync_with_stdio(false);
134     cin.tie(0);
135     cout.tie(0);
136     cin >> n >> m;
137     for (int i = 0; i < n; i++) {
138         int a;
139         cin >> a;
140         bst_insert(a);
141     }
142     int lasta = 0;
143     vector<int> res;
144     while (m--) {
145         int op, x;
146         cin >> op >> x;
147         x ^= lasta;
148         if (op == 1) // insert
149             bst_insert(x);
150         else if (op == 2) // delete
151             bst_remove(x);
152         else if (op == 3) // rank
153             lasta = bst_rank(x);
154         else if (op == 4) // k-th
155             lasta = bst_kth(x);
156         else if (op == 5) // pre
157             lasta = bst_pre(x);
158         else if (op == 6) // suc
159             lasta = bst_suc(x);
160         if (op > 2) {
161             res.push_back(lasta);
162         }
163     }
164     int ans = 0;
165     for (auto v : res) ans ^= v;
166     cout << ans << endl;
167     return 0;
168 }

```

DSU on Tree

```
1  /**
2   * https://codeforces.com/contest/600/problem/E
3   * 树的节点有权，根为 1
4   * 一种权占领了一个子树
5   * 当且仅当没有其他权在这个子树中出现更多次
6   * 求占领每个子树的所有权之和
7   * 输入：
8   * 节点数
9   * 各节点的权
10  * 边
11  * 输出：
12  * 各节点的占领权之和
13  ****
14  * 每个节点的答案是其子树的叠加，利用这个性质处理问题
15  * 预处理出每个节点子树的 size 和它的重儿子(节点最多子树的儿子)，可以O(n)完成
16  * 用 check[i] 表示颜色 i 有没有出现过，ans[i] 表示出现次数
17  * 按以下的步骤遍历一个节点：
18  * 遍历其非重儿子，获取它的 ans，但不保留遍历后它的 check
19  * 遍历它的重儿子，保留它的 check
20  * 再次遍历其非重儿子及其父亲，用重儿子的 check
21  * 对遍历到的节点进行计算，获取整棵子树的 ans
22  */
23 #include <bits/stdc++.h>
24 using namespace std;
25 const int MAXN = 1e5 + 100;
26 int n, a[MAXN], tot = -1;
27 int head[MAXN], to[MAXN << 1], nxt[MAXN << 1];
28 int bson[MAXN], sz[MAXN];
29 long long ans[MAXN], sum;
30 int maxc, flag;
31 int clr[MAXN];
32 void add(int u, int v) {
33     // 链式前向星
34     nxt[++tot] = head[u];
35     head[u] = tot;
36     to[tot] = v;
37     nxt[++tot] = head[v];
38     head[v] = tot;
39     to[tot] = u;
40 }
41 void dfs(int u, int f) {
42     sz[u] = 1;
43     for (int pp = head[u]; pp != -1; pp = nxt[pp]) {
44         int nxt_id = to[pp];
45         if (nxt_id == f) continue;
46         dfs(nxt_id, u);
47         sz[u] += sz[nxt_id];
48         if (sz[nxt_id] > sz[bson[u]]) bson[u] = nxt_id;
49     }
50 }
51 void add(int u, int f, int val) {
52     clr[a[u]] += val;
53     if (clr[a[u]] > maxc) {
54         maxc = clr[a[u]];
55         /***** ans *****/
56         sum = a[u];
57         /*****/
58     } else if (clr[a[u]] == maxc) {
```

```

59     /***** ans *****/
60     sum += a[u];
61     /*****/
62 }
63 for (int pp = head[u]; pp != -1; pp = nxt[pp]) {
64     int nxt_id = to[pp];
65     if (nxt_id == flag || nxt_id == f) continue;
66     add(nxt_id, u, val);
67 }
68 }
69 void dfs(int u, int f, bool keep) {
70     for (int pp = head[u]; pp != -1; pp = nxt[pp]) {
71         int nxt_id = to[pp];
72         if (nxt_id == f || nxt_id == bson[u]) continue;
73         dfs(nxt_id, u, 0);
74     }
75     if (bson[u]) {
76         dfs(bson[u], u, 1);
77         flag = bson[u];
78     }
79     add(u, f, 1);
80     flag = 0;
81     /***** ans *****/
82     ans[u] = sum;
83     /*****/
84     if (!keep) {
85         add(u, f, -1);
86         /***** ans *****/
87         maxc = sum = 0;
88         /*****/
89     }
90 }
91 void solve() {
92     int u, v;
93     // fill(head+1, head+n+2, -1);
94     cin >> n;
95     fill(head, head + n + 2, -1);
96     for (int i = 1; i <= n; ++i) cin >> a[i];
97     for (int i = 1; i < n; ++i) {
98         cin >> u >> v;
99         add(u, v);
100     }
101     dfs(1, -1);
102     dfs(1, -1, 0);
103     for (int i = 1; i < n; ++i) cout << ans[i] << " ";
104     cout << ans[n] << "\n";
105 }
106 int main() {
107     ios::sync_with_stdio(false);
108     cin.tie(0);
109     cout.tie(0);
110     solve();
111     return 0;
112 }

```

```

2  #include <bits/stdc++.h>
3  using namespace std;
4  using ll = long long;
5  const ll MAXN = 100005;
6  ll tree[MAXN];
7  ll lowbit(int x) { return (x) & (-x); }
8  void Update(int i, ll x) {
9      // increase
10     for (int pos = i; pos <= MAXN; pos += lowbit(pos)) {
11         tree[pos] += x;
12     }
13 }
14 ll PrefixQuery(int n) {
15     ll ret = 0;
16     for (int pos = n; pos; pos -= lowbit(pos)) {
17         ret += tree[pos];
18     }
19     return ret;
20 }
21 ll RangeQuery(int ql, int qr) { return PrefixQuery(qr) - PrefixQuery(ql - 1); }
22 int main() {
23     int a[10] = {-1, 4, 2, 1, 5, 6, 7, 2, 1, 4};
24     for (int i = 1; i <= 9; i++) {
25         Update(i, a[i]);
26     }
27     for (int i = 1; i <= 9; i++) {
28         cout << PrefixQuery(i) << endl;
29     }
30     return 0;
31 }
32

```

Mono Queue

```

1  #include <bits/stdc++.h>
2  // monotonic descending queue, segMax at front
3  using namespace std;
4
5  void getSegMax(vector<int>& v, int k, vector<int>& ans) {
6      deque<int> que;
7      int n = v.size();
8      for (int i = 0; i + 1 < k; ++i) {
9          while (!que.empty() && v[que.back()] <= v[i]) que.pop_back();
10         que.push_back(i);
11     }
12     for (int i = k - 1; i < n; ++i) {
13         while (!que.empty() && v[que.back()] <= v[i]) que.pop_back();
14         que.push_back(i);
15         while (que.front() <= i - k) que.pop_front();
16         ans.push_back(v[que.front()]);
17     }
18 }
19 void getSegMin(vector<int>& v, int k, vector<int>& ans) {
20     deque<int> que;
21     int n = v.size();
22     for (int i = 0; i + 1 < k; ++i) {
23         while (!que.empty() && v[que.back()] >= v[i]) que.pop_back();
24         que.push_back(i);
25     }
26     for (int i = k - 1; i < n; ++i) {
27         while (!que.empty() && v[que.back()] >= v[i]) que.pop_back();
28         que.push_back(i);
29         while (que.front() <= i - k) que.pop_front();
30         ans.push_back(v[que.front()]);
31     }
32 }
33

```

```

25     }
26     for (int i = k - 1; i < n; ++i) {
27         while (!que.empty() && v[que.back()] >= v[i]) que.pop_back();
28         que.push_back(i);
29         while (que.front() <= i - k) que.pop_front();
30         ans.push_back(v[que.front()]);
31     }
32 }
33 int main() {
34     vector<int> v = {2, 3, 1, 4, 5, 6, 7, 3};
35     vector<int> ans;
36     getSegMin(v, 3, ans);
37     for (auto itm: ans) {
38         cout << itm << " ";
39     }
40     return 0;
41 }

```

Segment Tree Range

```

1  #include <iostream>
2  using namespace std;
3  using ll = long long;
4  const int MAXN = 200005;
5
6  struct Node {
7      // TODO modify to fit the need
8      ll l, r;
9      ll ans, mulv, addv;
10     Node() {}
11 };
12 Node tree[MAXN << 2];
13 ll n, m, q, rawValues[MAXN];
14
15 void MergeNode(Node &f, const Node &lc, const Node &rc) {
16     // TODO VARY based on different problems
17     f.ans = (lc.ans + rc.ans) % m;
18     f.addv = 0;
19     f.mulv = 1;
20 }
21 void NodeAdd(int k, ll addv) {
22
23 }
24 void NodeMul(int k, ll mulv) {
25
26 }
27 void SpreadTag(Node &f, Node &sn) {
28     // TODO VARY based on different problems
29     ll addv = f.addv, mulv = f.mulv;
30     sn.ans = (sn.ans * mulv % m + (sn.r - sn.l + 1) % m * addv % m) % m;
31     sn.mulv = sn.mulv * mulv % m;
32     sn.addv = (sn.addv * mulv % m + addv) % m;
33 }
34 void PushUp(int k) { // up a level
35     MergeNode(tree[k], tree[k << 1], tree[k << 1 | 1]);
36 }
37 void PushDown(int k) { // push the lazy tag down a level
38     if (!(tree[k].addv == 0 && tree[k].mulv == 1)) {

```



```

39     SpreadTag(tree[k], tree[k << 1]);
40     SpreadTag(tree[k], tree[k << 1 | 1]);
41     // TODO reset father's lazy tag
42     tree[k].addv = 0;
43     tree[k].mulv = 1;
44 }
45 }
46 void BuildTree(int k, int l, int r) {
47     // prepare the nodes
48     tree[k].l = l;
49     tree[k].r = r;
50     if (l == r) {
51         // TODO VARY based on different problems
52         tree[k].ans = rawValues[l];
53         tree[k].addv = 0;
54         tree[k].mulv = 1;
55     } else {
56         int mid = l + (r - l) / 2;
57         BuildTree(k << 1, l, mid);
58         BuildTree(k << 1 | 1, mid + 1, r);
59         PushUp(k);
60     }
61 }
62 void UpdateSegMul(int k, int l, int r, ll mulv) {
63     if (l <= tree[k].l && tree[k].r <= r) {
64         // TODO VARY based on problems
65         // record the operation for query with smaller range
66         tree[k].ans = tree[k].ans * mulv % m;
67         tree[k].mulv = tree[k].mulv * mulv % m;
68         tree[k].addv = tree[k].addv * mulv % m;
69     } else {
70         PushDown(k);
71         int mid = tree[k].l + (tree[k].r - tree[k].l) / 2;
72         if (mid >= l) // separated update
73             UpdateSegMul(k << 1, l, r, mulv);
74         if (mid < r) UpdateSegMul(k << 1 | 1, l, r, mulv);
75         PushUp(k);
76     }
77 }
78 void UpdateSegAdd(int k, int l, int r, ll addv) {
79     if (l <= tree[k].l && tree[k].r <= r) {
80         // TODO VARY based on problems
81         tree[k].ans = (tree[k].ans + addv * (tree[k].r - tree[k].l + 1) % m) % m;
82         tree[k].addv = (tree[k].addv + addv) % m;
83     } else {
84         PushDown(k);
85         int mid = tree[k].l + (tree[k].r - tree[k].l) / 2;
86         if (mid >= l) // separated update
87             UpdateSegAdd(k << 1, l, r, addv);
88         if (mid < r) UpdateSegAdd(k << 1 | 1, l, r, addv);
89         PushUp(k);
90     }
91 }
92 void UpdateDot(int k, int pos, ll val) {
93     if (tree[k].l == tree[k].r) {
94         // TODO VARY based on problems
95         // tree[k].sum = val;
96     } else {
97         PushDown(k);
98         int mid = tree[k].l + (tree[k].r - tree[k].l) / 2;
99         if (pos <= mid) // separated update

```

```

100     UpdateDot(k << 1, pos, val);
101     else
102         UpdateDot(k << 1 | 1, pos, val);
103     PushUp(k);
104 }
105 }
106 Node Query(int k, int ql, int qr) {
107     if (tree[k].l >= ql && tree[k].r <= qr) return tree[k];
108     // when not single, push down firstly, then do the query
109     PushDown(k);
110     int mid = tree[k].l + (tree[k].r - tree[k].l) / 2;
111     Node resL, resR, retVal;
112     bool hasL = false, hasR = false;
113     if (ql <= mid) {
114         hasL = true;
115         resL = Query(k << 1, ql, qr);
116     }
117     if (mid < qr) {
118         hasR = true;
119         resR = Query(k << 1 | 1, ql, qr);
120     }
121     if (hasL && hasR)
122         MergeNode(retVal, resL, resR);
123     else if (hasL)
124         retVal = resL;
125     else if (hasR)
126         retVal = resR;
127     return retVal;
128 }
129 int main() {
130     ios::sync_with_stdio(false);
131     cin >> n >> q >> m;
132     for (int i = 1; i <= n; i++) cin >> rawValues[i];
133     //////////////////////////////////////
134     BuildTree(1, 1, n);
135     //////////////////////////////////////
136     int t, l, r, v;
137     while (q--) {
138         cin >> t >> l >> r;
139         if (t == 3) {
140             cout << Query(1, l, r).ans << "\n";
141         } else if (t == 1) {
142             cin >> v;
143             UpdateSegMul(1, l, r, v);
144         } else if (t == 2) {
145             cin >> v;
146             UpdateSegAdd(1, l, r, v);
147         }
148     }
149     return 0;
150 }

```

Union Set

```

1  #include <iostream>
2  using namespace std;
3  const int MAXN = 100005;
4  int father[MAXN];

```

```

5  int trank[MAXN];
6
7  void Init(int n) {
8      for (int i = 0; i < n; ++i) {
9          father[i] = i;
10         trank[i] = 0;
11     }
12 }
13 int Find(int x) {
14     if (father[x] == x) {
15         return x;
16     }
17     return father[x] = Find(father[x]);
18 }
19 void Unite(int x, int y) {
20     x = Find(x);
21     y = Find(y);
22     if (x == y) {
23         return;
24     }
25     if (trank[x] < trank[y]) {
26         father[x] = y;
27     } else {
28         father[y] = x;
29         if (trank[x] == trank[y]) {
30             trank[x]++;
31         }
32     }
33 }
34 bool inSame(int x, int y) { return Find(x) == Find(y); }

```

Geometry

```
1  const double EPS = 1e-9;
2  bool eq(double a, double b) { return abs(a - b) < EPS; } // ==
3  bool gt(double a, double b) { return a - b > EPS; } // >
4  bool lt(double a, double b) { return a - b < -EPS; } // <
5  bool ge(double a, double b) { return a - b > -EPS; } // >=
6  bool le(double a, double b) { return a - b < EPS; } // <=
7  int sgn (double x) { // sign of a double
8      if (fabs(x) < EPS) return 0;
9      else if (x < 0) return -1;
10     else return 1;
11 }
12 // 直线与直线交点
13 // DEPENDS eq, d*V, V*V, V+V, V^V
14 vector<Point> inter(Line a, Line b) {
15     double c = a.v ^ b.v;
16     if (eq(c, 0)) return {};
17     Vec v = 1 / c * Vec{a.P ^ (a.P + a.v), b.P ^ (b.P + b.v)};
18     return {{v * Vec{-b.v.x, a.v.x}, v * Vec{-b.v.y, a.v.y}}};
19 }
20
21 // 直线与圆交点
22 // DEPENDS eq, gt, V+V, V-V, V*V, d*V, len, pedal
23 vector<Point> inter(Line l, Circle C) {
24     Point P = pedal(C.O, l);
25     double h = len(P - C.O);
26     if (gt(h, C.r)) return {};
27     if (eq(h, C.r)) return {P};
28     double d = sqrt(C.r * C.r - h * h);
29     Vec vec = d / len(l.v) * l.v;
30     return {P + vec, P - vec};
31 }
32
33 // 圆与圆的交点 注意内含和相离的情况
34 // DEPENDS eq, gt, V+V, V-V, d*V, len, r90c
35 vector<Point> inter(Circle C1, Circle C2) {
36     Vec v1 = C2.O - C1.O, v2 = r90c(v1);
37     double d = len(v1);
38     if (gt(d, C1.r + C2.r) || gt(abs(C1.r - C2.r), d)) return {};
39     if (eq(d, C1.r + C2.r) || eq(d, abs(C1.r - C2.r)))
40         return {C1.O + C1.r / d * v1};
41     double a = ((C1.r * C1.r - C2.r * C2.r) / d + d) / 2;
42     double h = sqrt(C1.r * C1.r - a * a);
43     Vec av = a / len(v1) * v1, hv = h / len(v2) * v2;
44     return {C1.O + av + hv, C1.O + av - hv};
45 }
46
47 // 三角形的重心
48 Point barycenter(Point A, Point B, Point C) {
49     return {(A.x + B.x + C.x) / 3, (A.y + B.y + C.y) / 3};
50 }
51
52 // 三角形的外心
53 // DEPENDS r90c, V*V, d*V, V-V, V+V
```

```

54 // NOTE 给定圆上三点求圆，要先判断是否三点共线
55 Point circumcenter(Point A, Point B, Point C) {
56     double a = A * A, b = B * B, c = C * C;
57     double d = 2 * (A.x * (B.y - C.y) + B.x * (C.y - A.y) + C.x * (A.y - B.y));
58     return 1 / d * r90c(a * (B - C) + b * (C - A) + c * (A - B));
59 }
60
61 // 三角形的内心
62 // DEPENDS len, d*V, V-V, V+V
63 Point incenter(Point A, Point B, Point C) {
64     double a = len(B - C), b = len(A - C), c = len(A - B);
65     double d = a + b + c;
66     return 1 / d * (a * A + b * B + c * C);
67 }
68
69 // 三角形的垂心
70 // DEPENDS V*V, d*V, V-V, V^V, r90c
71 Point orthocenter(Point A, Point B, Point C) {
72     double n = B * (A - C), m = A * (B - C);
73     double d = (B - C) ^ (A - C);
74     return 1 / d * r90c(n * (C - B) - m * (C - A));
75 }
76

```

Fraction

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  using ll = long long;
4  struct Fraction {
5      ll up, dn;
6      Fraction() : up(0), dn(1) {}
7      Fraction(ll _up, ll _dn) : up(_up), dn(_dn) {
8          ll cd = __gcd(up, dn);
9          up /= cd;
10         dn /= cd;
11         if (dn < 0) {
12             dn = -dn;
13             up = -up;
14         }
15     }
16     void reduce() {
17         ll cd = __gcd(up, dn);
18         up /= cd;
19         dn /= cd;
20     }
21     Fraction operator+(const Fraction &otr) const {
22         ll n_dn = dn / __gcd(otr.dn, dn) * otr.dn;
23         ////////////// possible overflow //////////////////
24         ll n_up = n_dn / dn * up + n_dn / otr.dn * otr.up;
25         return Fraction(n_up, n_dn);
26     }
27     Fraction operator-(const Fraction &otr) const {
28         ll n_dn = dn / __gcd(otr.dn, dn) * otr.dn;
29         ////////////// possible overflow //////////////////
30         ll n_up = n_dn / dn * up - n_dn / otr.dn * otr.up;
31         return Fraction(n_up, n_dn);
32     }

```

```

33 Fraction operator*(const Fraction &otr) const {
34     ll n_dn = dn * otr.dn;
35     ll n_up = up * otr.up;
36     // cout << n_up << "/" << n_dn << endl;
37     ll cd = __gcd(n_dn, n_up);
38     return Fraction(n_up / cd, n_dn / cd);
39 }
40 Fraction operator/(const Fraction &otr) const {
41     Fraction loprd(up, dn), roprd(otr.dn, otr.up);
42     return loprd * roprd;
43 }
44 bool operator==(const Fraction &otr) const {
45     ll uup = up, ddn = dn, cd = __gcd(up, dn);
46     uup /= up, ddn /= dn;
47     ll oup = otr.up, odn = otr.dn;
48     cd = __gcd(oup, odn);
49     oup /= cd, odn /= cd;
50     return up * otr.dn == dn * otr.up;
51 }
52 bool operator<(const Fraction &otr) const {
53     ll uup = up, ddn = dn, cd = __gcd(up, dn);
54     uup /= up, ddn /= dn;
55     ll oup = otr.up, odn = otr.dn;
56     cd = __gcd(oup, odn);
57     oup /= cd, odn /= cd;
58     return uup * odn < oup * ddn;
59 }
60 bool operator<=(const Fraction &otr) const {
61     Fraction fra{up, dn};
62     return fra < otr || fra == otr;
63 }
64 double real_val() const { return double(up) / double(dn); }
65 };
66 int main() {
67     Fraction a(1, 2), b(3, 6);
68     cout << (a * b).real_val() << endl;
69     cout << (a - b).real_val() << endl;
70     cout << (a == b) << endl;
71     return 0;
72 }

```

3D Sphere

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  const double PI = acos(-1.0);
4  struct Sphere {
5      double x, y, z, r;
6      Sphere() {}
7      Sphere(double x, double y, double z, double r) : x(x), y(y), z(z), r(r) {}
8  };
9  double IntersectionVolume(Sphere o, Sphere t) {
10     // basic formula: V = (3 * r - h) * h * h * PI / 3
11     // calculated from spinning surface calculus
12     if (o.r < t.r) swap(o, t);

```

```

13 double dis = sqrt((o.x - t.x) * (o.x - t.x) + (o.y - t.y) * (o.y - t.y) +
14                 (o.z - t.z) * (o.z - t.z));
15 if (dis <= o.r - t.r) { // completely in
16     return 4.0 / 3 * PI * t.r * t.r * t.r;
17 } else if (dis <= o.r) { // center of the smaller sphere in bigger sphere
18     // cosA = (b2 + c2 - a2) / 2bc
19     double angleb = acos((t.r * t.r + dis * dis - o.r * o.r) / (2 * t.r * dis));
20     double anglea = PI - angleb;
21     double l = t.r * cos(anglea);
22     double H = o.r - l - dis;
23     double h = t.r - l;
24     return 4.0 / 3 * PI * t.r * t.r * t.r - PI / 3 * (3 * t.r - h) * h * h +
25           PI / 3 * (3 * o.r - H) * H * H;
26 } else if (dis < o.r + t.r) { // normal intersection
27     double angler = acos((t.r * t.r + dis * dis - o.r * o.r) / (2 * t.r * dis));
28     double angleR = acos((o.r * o.r + dis * dis - t.r * t.r) / (2 * o.r * dis));
29     double H = o.r - o.r * cos(angleR);
30     double h = t.r - t.r * cos(angler);
31     return PI / 3 * (3 * t.r - h) * h * h + PI / 3 * (3 * o.r - H) * H * H;
32 } else {
33     return 0;
34 }
35 }
36 double IntersectionSurface(Sphere &o, Sphere &t) {
37     // basic formula: S = 2 * PI * r * h
38     if (o.r < t.r) swap(o, t);
39     double dis = sqrt((o.x - t.x) * (o.x - t.x) + (o.y - t.y) * (o.y - t.y) +
40                     (o.z - t.z) * (o.z - t.z));
41     if (dis <= o.r - t.r) { // completely in
42         return 4 * PI * t.r * t.r;
43     } else if (dis <= o.r) { // center of the smaller sphere in bigger sphere
44         double angleb = acos((t.r * t.r + dis * dis - o.r * o.r) / (2 * t.r * dis));
45         double anglea = PI - angleb;
46         double l = t.r * cos(anglea);
47         double H = o.r - l - dis;
48         double h = t.r - l;
49         return 4 * PI * t.r * t.r - 2 * PI * t.r * h + 2 * PI * o.r * H;
50     } else if (dis < o.r + t.r) { // normal intersection
51         double angler = acos((t.r * t.r + dis * dis - o.r * o.r) / (2 * t.r * dis));
52         double angleR = acos((o.r * o.r + dis * dis - t.r * t.r) / (2 * o.r * dis));
53         double H = o.r - o.r * cos(angleR);
54         double h = t.r - t.r * cos(angler);
55         return 2 * PI * t.r * h + 2 * PI * o.r * H;
56     } else {
57         return 0;
58     }
59 }
60 int main() {
61     Sphere A, B;
62     cin >> A.x >> A.y >> A.z >> A.r;
63     cin >> B.x >> B.y >> B.z >> B.r;
64     cout << fixed << setprecision(10) << 4*PI*(A.r*A.r+B.r*B.r) - IntersectionSurface(A, B) << endl;
65     return 0;
66 }

```

2D Vector

```

2  * structs of
3  * point, vector, segment
4  * and some operator overloads
5  */
6  // whether a seg AB intersects with a circle O?
7  // see the endpoints' tangent point (P, Q) angle
8  // angles: AOP + BOQ < AOB <==> intersect
9  #include <bits/stdc++.h>
10 using namespace std;
11 using ll = long long;
12 ll MOD = 1e9 + 7;
13 ll QpowMod(ll bse, ll pwr) {
14     ll ret = 1;
15     while (pwr) {
16         if (pwr & 1) ret = ret * bse % MOD;
17         bse = bse * bse % MOD;
18         pwr >>= 1;
19     }
20     return ret;
21 }
22 struct Point2 {
23     ll x, y;
24     Point2() : x(0), y(0) {}
25     Point2(ll _x, ll _y) : x(_x), y(_y) {}
26     ll Norm2() { return 1ll * x * x + 1ll * y * y; }
27     double Norm() { return sqrt(Norm2()); }
28     Point2 operator+(const Point2 &po) {
29         return Point2(x + po.x, y + po.y);
30     }
31     Point2 operator-(const Point2 &po) {
32         // note the direction
33         return Point2(x - po.x, y - po.y);
34     }
35     bool operator==(const Point2 &po) {
36         return x == po.x && y == po.y;
37     }
38 };
39 typedef Point2 Vector2;
40 struct Segment2 {
41     Point2 s, e;
42     Segment2() {}
43     Segment2(Point2 _s, Point2 _e) : s(_s), e(_e) {}
44 };
45 ll MulCross(const Point2 &p1, const Point2 &p2) {
46     return p1.x * p2.y - p1.y * p2.x;
47 }
48 ll MulDot(const Point2 &p1, const Point2 &p2) {
49     return p1.x * p2.x + p1.y * p2.y;
50 }
51 double DisPointToSeg(Point2 p, Point2 s1, Point2 s2) {
52     Point2 v1 = p - s1, v2 = s2 - s1;
53     if (MulDot(v2, v1) < 0 || MulDot(v2, v1) > v2.Norm2())
54         return min(1.0 * (p - s1).Norm(), 1.0 * (p - s2).Norm());
55     return abs(1.0 * MulCross(v2, v1) / v2.Norm());
56 }
57 int Dis2PointToSeg_INT(Point2 p, Point2 s1, Point2 s2) {
58     // square of distance between two points
59     Point2 v = p - s1, u = s2 - s1;
60     if (MulDot(u, v) < 0 || MulDot(u, v) > u.Norm2())
61         return min((p - s1).Norm2(), (p - s2).Norm2()) % MOD;
62     return ((MulCross(v, u) % MOD) * (MulCross(v, u) % MOD)) % MOD *

```



```

63     QpowMod(u.Norm2() % MOD, MOD - 2) % MOD;
64 }
65 int main() { return 0; }

```

Vector3ll

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  using ll = long long;
4  ll MOD = 1e9 + 7;
5  struct Point3fra {
6      ll x, y, z;
7      Point3fra() : x(0), y(0), z(0) {}
8      Point3fra(ll _x, ll _y, ll _z) : x(_x), y(_y), z(_z) {}
9      ll norm2() { return x * x + y * y + z * z; }
10     double norm() { return sqrt(norm2()); }
11     Point3fra operator+(const Point3fra &po) {
12         return Point3fra(x + po.x, y + po.y, z + po.z);
13     }
14     Point3fra operator-(const Point3fra &po) {
15         return Point3fra(x - po.x, y - po.y, z - po.z);
16     }
17     bool operator==(const Point3fra &po) {
18         return x == po.x && y == po.y && z == po.z;
19     }
20 };
21 typedef Point3fra Vector3ll;
22 struct Segment3ll {
23     Point3fra s, e;
24     Segment3ll() {}
25     Segment3ll(Point3fra _s, Point3fra _e) : s(_s), e(_e) {}
26 };
27 ll mul_dot(const Point3fra &p1, const Point3fra &p2) {
28     return p1.x * p2.x + p1.y * p2.y + p1.z * p2.z;
29 }
30 Point3fra mul_cross(const Point3fra &p1, const Point3fra &p2) {
31     return Point3fra(p1.y * p2.z - p1.z * p2.y, p1.z * p2.x - p1.x * p2.z, p1.x * p2.y - p1.y * p2.x);
32 }
33 int main() {
34     Point3fra a{0, 0, 1}, b{1, 1, 1};
35     Point3fra c = mul_cross(a, b);
36     cout << c.norm() << endl;
37     return 0;
38 }

```

Vector3Fra

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  using ll = long long;
4  ll MOD = 1e9 + 7;
5  struct Fraction {

```

```

6   ll up, dn;
7   Fraction() : up(0), dn(1) {}
8   Fraction(ll _up, ll _dn) : up(_up), dn(_dn) {
9       ll cd = __gcd(up, dn);
10      up /= cd;
11      dn /= cd;
12  }
13  Fraction(Fraction fup, Fraction fdn) {
14      Fraction tmp = fup / fdn;
15      tmp.reduce();
16      up = tmp.up;
17      dn = tmp.dn;
18  }
19  void reduce() {
20      ll cd = abs(__gcd(up, dn));
21      up /= cd;
22      dn /= cd;
23      neg_sign();
24  }
25  void neg_sign() {
26      if (dn < 0) {
27          dn = -dn;
28          up = -up;
29      }
30  }
31  Fraction operator+(const Fraction &otr) const {
32      ll n_dn = dn / __gcd(otr.dn, dn) * otr.dn;
33      ll n_up = n_dn / dn * up + n_dn / otr.dn * otr.up;
34      Fraction ret{n_up, n_dn};
35      ret.reduce();
36      ret.neg_sign();
37      return ret;
38  }
39  Fraction operator-(const Fraction &otr) const {
40      ll n_dn = dn / __gcd(otr.dn, dn) * otr.dn;
41      ll n_up = n_dn / dn * up - n_dn / otr.dn * otr.up;
42      Fraction ret{n_up, n_dn};
43      ret.reduce();
44      ret.neg_sign();
45      return ret;
46  }
47  Fraction operator*(const Fraction &otr) const {
48      ll n_dn = dn * otr.dn;
49      ll n_up = up * otr.up;
50      // cout << n_up << "/" << n_dn << endl;
51      ll cd = abs(__gcd(n_dn, n_up));
52      n_up /= cd, n_dn /= cd;
53      Fraction ret{n_up, n_dn};
54      ret.reduce();
55      ret.neg_sign();
56      return ret;
57  }
58  Fraction operator/(const Fraction &otr) const {
59      Fraction loprd(up, dn), roprd(otr.dn, otr.up);
60      return loprd * roprd;
61  }
62  bool operator==(const Fraction &otr) const {
63      ll uup = up, ddn = dn;
64      if (ddn < 0) uup = -uup, ddn = -ddn;
65      ll cd = abs(__gcd(up, dn));
66      uup /= up, ddn /= dn;

```

```

67     ll oup = otr.up, odn = otr.dn;
68     if (odn < 0) oup = -oup, odn = -odn;
69     cd = abs(__gcd(oup, odn));
70     oup /= cd, odn /= cd;
71     return up * otr.dn == dn * otr.up;
72 }
73 bool operator<(const Fraction &otr) const {
74     ll uup = up, ddn = dn;
75     if (ddn < 0) uup = -uup, ddn = -ddn;
76     ll cd = abs(__gcd(up, dn));
77     ll oup = otr.up, odn = otr.dn;
78     if (odn < 0) oup = -oup, odn = -odn;
79     cd = abs(__gcd(oup, odn));
80     oup /= cd, odn /= cd;
81     return uup * odn < oup * ddn;
82 }
83 bool operator<=(const Fraction &otr) const {
84     Fraction fra{up, dn};
85     return fra < otr || fra == otr;
86 }
87 double real_val() const { return double(up) / double(dn); }
88 };
89 struct Point3fra {
90     Fraction x, y, z;
91     Point3fra() : x(0, 1), y(0, 1), z(0, 1) {}
92     Point3fra(Fraction _x, Fraction _y, Fraction _z) : x(_x), y(_y), z(_z) {}
93     Fraction norm2() { return (x * x) + (y * y) + (z * z); }
94     double norm() { return sqrt(norm2().real_val()); }
95     Point3fra operator+(const Point3fra &po) {
96         return Point3fra(x + po.x, y + po.y, z + po.z);
97     }
98     Point3fra operator-(const Point3fra &po) {
99         return Point3fra(x - po.x, y - po.y, z - po.z);
100    }
101    bool operator==(Point3fra &po) {
102        return (x == po.x) && (y == po.y) && (z == po.z);
103    }
104 };
105 typedef Point3fra Vector3fra;
106 /***** types done *****/
107 /***** functions go *****/
108 Fraction frac_zero{0, 1}, frac_one{1, 1};
109 Fraction mul_dot(const Point3fra &p1, const Point3fra &p2) {
110     return p1.x * p2.x + p1.y * p2.y + p1.z * p2.z;
111 }
112 Point3fra mul_cross(const Point3fra &p1, const Point3fra &p2) {
113     return Point3fra(p1.y * p2.z - p1.z * p2.y, p1.z * p2.x - p1.x * p2.z,
114                     p1.x * p2.y - p1.y * p2.x);
115 }
116 Point3fra mul_scale(const Point3fra &p1, const Fraction &s) {
117     Fraction sc{s.up, s.dn};
118     sc.reduce();
119     return Point3fra(p1.x * sc, p1.y * sc, p1.z * sc);
120 }
121 bool is_segs_intersect(Point3fra A, Point3fra B, Point3fra C, Point3fra D) {
122     Vector3fra ac = C - A, ad = D - A, ca = A - C, cb = B - C;
123     Vector3fra nm_abc = mul_cross(B - A, ac);
124     Vector3fra nm_abd = mul_cross(B - A, ad);
125     Vector3fra nm_acd = mul_cross(D - C, ca);
126     Vector3fra nm_bcd = mul_cross(D - C, cb);
127     bool flg1 = mul_dot(nm_abc, nm_abd) < frac_zero && mul_cross(nm_abc, nm_abd).norm2() == frac_zero;

```

```

128     bool flg2 = mul_dot(nm_acd, nm_bcd) < frac_zero && mul_cross(nm_acd, nm_bcd).norm2() == frac_zero;
129     return flg1 && flg2;
130 }
131 Fraction point_to_point2(Point3fra A, Point3fra B) { return (A - B).norm2(); }
132 Fraction point_to_seg2(Point3fra P, Point3fra A, Point3fra B) {
133     if (A == B) return point_to_point2(P, A);
134     Vector3fra ap = P - A, ab = B - A, bp = P - B, ba = A - B;
135     if (mul_dot(ap, ab) <= frac_zero || mul_dot(bp, ba) <= frac_zero) {
136         Fraction ret = point_to_point2(P, A);
137         ret = min(ret, point_to_point2(P, B));
138         return ret;
139     } else {
140         Vector3fra pa = A - P, pb = B - P, ab = B - A;
141         Fraction up = mul_cross(pa, pb).norm2(), dn = ab.norm2();
142         return Fraction{up, dn};
143     }
144 }
145 Fraction seg_to_seg2(Point3fra A, Point3fra B, Point3fra C, Point3fra D) {
146     Vector3fra ca = A - C, cb = B - C, cd = D - C, ab = B - A, ac = C - A;
147     Fraction tmp = mul_dot(mul_cross(ca, cb), cd);
148     bool is_intersec = is_segs_intersect(A, B, C, D);
149     if (tmp == frac_zero || is_intersec) {
150         // same plane or intersect
151         if (is_intersec) return frac_zero;
152         Fraction ret = point_to_seg2(A, C, D);
153         ret = min(ret, point_to_seg2(B, C, D));
154         ret = min(ret, point_to_seg2(C, A, B));
155         ret = min(ret, point_to_seg2(D, A, B));
156         return ret;
157     } else {
158         // not in same plane, using maxima of two-variable function
159         Fraction dn = mul_dot(ab, cd) * mul_dot(ab, cd) - ab.norm2() * cd.norm2();
160         Fraction t(ab.norm2() * mul_dot(cd, ac) - mul_dot(ab, cd) * mul_dot(ab, ac),
161                 dn);
162         Fraction s(mul_dot(ab, cd) * mul_dot(cd, ac) - cd.norm2() * mul_dot(ab, ac),
163                 dn);
164         t.reduce();
165         s.reduce();
166         if (frac_zero < t && t < frac_one && frac_zero < s && s < frac_one) {
167             return point_to_point2(A + mul_scale(ab, s), C + mul_scale(cd, t));
168         } else {
169             Fraction ret = point_to_seg2(A, C, D);
170             ret = min(ret, point_to_seg2(B, C, D));
171             ret = min(ret, point_to_seg2(C, A, B));
172             ret = min(ret, point_to_seg2(D, A, B));
173             return ret;
174         }
175     }
176 }
177
178 int main() {
179     ios::sync_with_stdio(false);
180     cin.tie(0);
181     cout.tie(0);
182     int t = 1;
183     cin >> t;
184     while (t--) {
185         ll ax, ay, az, bx, by, bz;
186         ll cx, cy, cz, dx, dy, dz;
187         cin >> ax >> ay >> az >> bx >> by >> bz;
188         cin >> cx >> cy >> cz >> dx >> dy >> dz;

```

```

189 Point3fra A{{ax, 1}, {ay, 1}, {az, 1}};
190 Point3fra B{{bx, 1}, {by, 1}, {bz, 1}};
191 Point3fra C{{cx, 1}, {cy, 1}, {cz, 1}};
192 Point3fra D{{dx, 1}, {dy, 1}, {dz, 1}};
193 Fraction ans = seg_to_seg2(A, B, C, D);
194 ans.reduce();
195 cout << abs(ans.up) << " " << abs(ans.dn) << endl;
196 }
197 return 0;
198 }

```

Math

$$C_n^m$$

```
1 #include <stdio.h>
2 using ll = long long;
3 const ll MN = 2000000;
4 const ll MOD = 1000000007;
5 int fac[MN + 5], inv[MN + 5];
6
7 ll qpowMod(ll bse, ll pwr) {
8     ll ret = 1;
9     while (pwr) {
10         if (pwr & 1) ret = ret * bse % MOD;
11         bse = bse * bse % MOD;
12         pwr >>= 1;
13     }
14     return ret;
15 }
16 void init() {
17     fac[0] = 1;
18     for (int i = 1; i <= MN; i++) fac[i] = 1ll * fac[i - 1] * i % MOD;
19     inv[MN] = qpowMod(fac[MN], MOD - 2);
20     for (int i = MN - 1; i >= 0; i--) inv[i] = 1ll * inv[i + 1] * (i + 1) % MOD;
21 }
22 int C(int n, int m) {
23     if (m > n) return 0;
24     return 1ll * fac[n] * inv[m] % MOD * inv[n - m] % MOD;
25 }
26 int main() {
27     init();
28     printf("%d\n", C(5, 3));
29     return 0;
30 }
```

Euler Primers

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using ll = long long;
4 const int MAXN = 1e6 + 5;
5 const int MOD = 1e9 + 7;
6 // priority_queue<ll, vector<ll>, greater<ll>> minor_que;
7
8 int prime[MAXN];
9 bool vis[MAXN];
10 int cnt = 0;
11 ll maxv = -1;
12 void EulerPrime(int n) {
13     for (int i = 2; i <= n; ++i) {
```

```

14     if (vis[i] == 0) {
15         prime[cnt++] = i;
16         vis[i] = 1;
17     }
18     for (int j = 0; i * prime[j] <= n; ++j) {
19         vis[i * prime[j]] = 1;
20         if (i % prime[j] == 0) break; // key of O(n)
21     }
22 }
23 }
24 int main() {
25     EulerPrime(100);
26     for (int i = 0; i < cnt; ++i) printf("%d ", prime[i]);
27     printf("\n");
28     return 0;
29 }

```

Josephus Ring

```

1 // n - 1 规模时留下的最后一人, 与 n 规模的相差了一个偏移量 k。J_{n, k} = (J_{n - 1, k} + k) mod n。 (从 0
  编号, 下同, 答案加一个偏移即可)
2 #include <cstdio>
3 long long josephus(int n, int k) {
4     if (n == 1)
5         return 0;
6     else
7         return (josephus(n - 1, k) + k) % n;
8 }
9 int main(void) {
10     long long n, k;
11     scanf("%lld %lld", &n, &k);
12
13     printf("%lld\n", 1 + josephus(n, k));
14     return 0;
15 }
16 // total n, k-th out, find the m-th out, start from 1
17 void solve(int casei) {
18     cout << "Case #" << casei << ": ";
19     long long ans = (K - 1) % (N - M + 1);
20     if (K == 1) {
21         cout << M << endl;
22         return;
23     }
24     for (ll i = N - M + 2; i <= N; i++) {
25         ans = (ans + K) % i; // normal iteration
26         // jump forward
27         ll rem = (i - ans - 1) / K;
28         rem = min(rem, N - i); // limit the times of jump
29         i += rem; // jump
30         ans += rem * K;
31     }
32     cout << ans + 1 << endl;
33 }

```

Matrix Inverse Element

Inverse element of 2x2 matrix $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ is $\begin{pmatrix} d & -b \\ -c & a \end{pmatrix} / (ad - bc)$.

Matrix Power

```
1  #include <bits/stdc++.h>
2  #define inf 0x3f3f3f3f
3  using namespace std;
4  typedef long long ll;
5  const int N = 205, mod = 998244353, MS = 205;
6  struct Mat {
7      ll a[MS][MS];
8      ll n, m;
9      Mat(int n = 0, int m = 0) : n(n), m(m) { memset(a, 0, sizeof(a)); }
10     Mat operator*(const Mat& B) const {
11         Mat C(n, B.m);
12         for (int i = 1; i <= n; i++)
13             for (int j = 1; j <= B.m; j++)
14                 for (int k = 1; k <= m; k++)
15                     C.a[i][j] = (C.a[i][j] + a[i][k] * B.a[k][j]) % mod;
16         return C;
17     }
18 };
19 Mat qpow(Mat a, int n) {
20     Mat ans(a.n, a.n);
21     for (int i = 1; i <= a.n; i++) ans.a[i][i] = 1;
22     for (; n >= 1; a = a * a)
23         if (n & 1) ans = ans * a;
24     return ans;
25 }
26 int main() {
27     ll n;
28     cin >> n;
29     string s;
30     cin >> s;
31     ll now = stol(s);
32     Mat A(100, 100);
33     A = qpow(A, n);
34
35     Mat B(100, 100);
36     B.a[1][1] = 1;
37     B = B * A;
38     cout << B.a[1][now];
39 }
```

Quick Power

```
1  #include <cstdio>
2  // a^(-1) mod p => a^(p - 2) mod p
3  // n * n * (n + 1) * (n + 1) / 4 = \sum_{1}^{n} i^3
4  // n * (n + 1) * (2n + 1) / 6 = \sum_{1}^{n} i^2
5  using ll = long long;
```



```
6  ll MOD = 1e9+7;
7  ll QpowMod(ll bse, ll pwr) {
8      ll ret = 1;
9      while (pwr) {
10         if (pwr & 1) ret = ret * bse % MOD;
11         bse = bse * bse % MOD;
12         pwr >>= 1;
13     }
14     return ret;
15 }
16 int main() {
17     printf("%lld", QpowMod(2, 199) * 6 % MOD);
18     return 0;
19 }
```

Graph

SCC kosaraju

```
1  #include <cstdio>
2  #include <stack>
3  using namespace std;
4  stack<int> stk;
5  // adjacent matrix
6  int mp[10][10];
7  // reversed graph
8  int mpt[10][10];
9  int vst[10];
10 int clr[10];
11 int vn, en;
12 void dfs1(int s) {
13     if (vst[s] == 1) return;
14     vst[s] = 1;
15     // dfs routine
16     for (int i = 1; i <= vn; ++i) {
17         if (mp[s][i] < 0x3f3f3f3f) {
18             dfs1(i);
19         }
20     }
21     // push
22     stk.push(s);
23 }
24 void dfs2(int s, int cnt) {
25     if (vst[s] == 0) return;
26     clr[s] = cnt;
27     vst[s] = 0;
28     for (int i = 1; i <= vn; ++i) {
29         if (mpt[s][i] < 0x3f3f3f3f) {
30             dfs2(i, cnt);
31         }
32     }
33 }
34 void init() {
35     for (int i = 1; i <= vn; ++i) {
36         for (int j = 1; j <= vn; ++j) {
37             mp[i][j] = mp[j][i] = 0x3f3f3f3f;
38             mpt[i][j] = mpt[j][i] = 0x3f3f3f3f;
39         }
40         mpt[i][i] = mp[i][i] = 0;
41     }
42 }
43 void SCC_kor() {
44     for (int i = 1; i <= vn; ++i) {
45         if (vst[i] == 0) dfs1(i);
46     }
47     int cnt = 1;
48     while (!stk.empty()) {
49         int s = stk.top();
50         stk.pop();
```

```

51     if (vst[s] == 0) continue;
52     dfs2(s, cnt++);
53 }
54 // vertexes with same value in clr[] is in one SCC
55 for (int i = 1; i <= vn; ++i) {
56     printf("%d ", clr[i]);
57 }
58 printf("\n");
59 }
60 int main() {
61     scanf("%d %d", &vn, &en);
62     init();
63     for (int i = 1; i <= en; ++i) {
64         int fr, to;
65         scanf("%d %d", &fr, &to);
66         mp[fr][to] = 1;
67         mpt[to][fr] = 1;
68     }
69     SCC_kor();
70     return 0;
71 }

```

SCC tarjan

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  int n, m;
4  struct node {
5      vector<int> nxt;
6  } g[100000];
7  int dfn[100000], low[100000], d[100000], col[100000], cnt[100000], stk[100000];
8  int vis[100000];
9  int top, deep, colour;
10 void tarjan(int u) {
11     dfn[u] = low[u] = ++deep;
12     stk[top++] = u;
13     vis[u] = 1;
14     for (int i = 0; i < g[u].nxt.size(); i++) {
15         int v = g[u].nxt[i];
16         if (!vis[v]) {
17             tarjan(v);
18             low[u] = min(low[v], low[u]);
19         } else {
20             low[u] = min(low[v], low[u]);
21         }
22     }
23     if (dfn[u] == low[u]) {
24         int node;
25         colour++;
26         while (node != u) {
27             node = stk[top - 1];
28             top--;
29             col[node] = colour;
30         }
31     }
32 }

```

String

KMP

```
1  int nxt[100005];
2  char t[100005];
3  void getNext() {
4      nxt[0] = -1;
5      int k = -1, j = 0;
6      while (t[j] != '\0') {
7          if (k == -1 || t[k] == t[j]) {
8              nxt[++j] = ++k;
9          } else {
10             k = nxt[k];
11         }
12     }
13 }
```

Manacher

```
1  // find the palindrome in O(n)
2  #include <bits/stdc++.h>
3  using namespace std;
4  char s[100005];
5  int ps = 0;
6  int p[100005], ctr, maxr, mirr;
7  void solve() {
8      ctr = maxr = 0;
9      for (int i = 0; i < ps; ++i) {
10         mirr = 2 * ctr - i;
11         if (i < maxr) {
12             p[i] = min(maxr - i, p[mirr]);
13         } else {
14             p[i] = 0;
15         }
16         while (s[i - 1 - p[i]] == s[i + 1 + p[i]]) {
17             p[i]++;
18         }
19         if (p[i] + i > maxr) {
20             ctr = i;
21             maxr = p[i] + i;
22         }
23     }
24     int maxi = 0;
25     for (int i = 0; i < ps; ++i) {
26         maxi = p[maxi] < p[i] ? i : maxi;
27     }
28     printf("%d\n", p[maxi]);
29     for (int i = maxi - p[maxi]; i <= maxi + p[maxi]; ++i) {
30         if (s[i] != '#') {
```

```
31     printf("%c", s[i]);
32 }
33 }
34 printf("\n");
35 }
36 int main() {
37     int Case = 1;
38     while (Case--) {
39         char c = getchar();
40         s[ps++] = '#';
41         while (c != '\n') {
42             s[ps++] = c;
43             s[ps++] = '#';
44             c = getchar();
45         }
46         solve();
47     }
48     return 0;
49 }
```

Misc

fastIO

```
1 namespace GTI
2 {
3     char gc(void)
4     {
5         const int S=1<<17;
6         static char buf[S],*s=buf,*t=buf;
7         if (s==t) t=buf+fread(s=buf,1,S,stdin);
8         if (s==t) return EOF;
9         return *s++;
10    }
11    int gti(void)
12    {
13        int a=0,b=1,c=gc();
14        for (;!isdigit(c);c=gc()) b^=(c=='-');
15        for (;isdigit(c);c=gc()) a=a*10+c-'0';
16        return b?a:-a;
17    }
18 };
19
```

Discretization

```
1 namespace GTI
2 {
3     char gc(void)
4     {
5         const int S=1<<17;
6         static char buf[S],*s=buf,*t=buf;
7         if (s==t) t=buf+fread(s=buf,1,S,stdin);
8         if (s==t) return EOF;
9         return *s++;
10    }
11    int gti(void)
12    {
13        int a=0,b=1,c=gc();
14        for (;!isdigit(c);c=gc()) b^=(c=='-');
15        for (;isdigit(c);c=gc()) a=a*10+c-'0';
16        return b?a:-a;
17    }
18 };
```

Inverse Pair Merge Sort

```
1 using ll = long long;
2 ll MAXN = 2e5 + 5;
3 ll n, q[MAXN], tmp[MAXN];
4 // [l, r]
5 ll merge_sort(int l, int r) {
6     if (l >= r) return 0;
7     ll mid = (l + r) >> 1;
8     ll res = merge_sort(l, mid) + merge_sort(mid + 1, r);
9
10    ll k = 0, i = l, j = mid + 1;
11    while (i <= mid && j <= r) {
12        if (q[i] <= q[j])
13            tmp[k++] = q[i++];
14        else {
15            tmp[k++] = q[j++];
16            res += mid - i + 1;
17        }
18    }
19    while (i <= mid) tmp[k++] = q[i++];
20    while (j <= r) tmp[k++] = q[j++];
21    for (ll i = l, j = 0; i <= r; i++, j++) q[i] = tmp[j];
22    return res;
23 }
```

Modui

```
1 /**
2  * Modui range number of distinct values
3  */
4 #include <bits/stdc++.h>
5 using namespace std;
6 #define endl "\n";
7 #define IOS_ONLY \
8     ios::sync_with_stdio(false); \
9     cin.tie(0); \
10    cout.tie(0);
11 const int MAXN = 30005, MAXQ = 200005, MAXM = 1000005;
12 int sq;
13 struct Query {
14     int ql, qr, id;
15     bool operator<(const Query &o) const {
16         // sqrt(n) partitions, assign sq with sqrt(n) first
17         if (ql / sq != o.ql / sq) return ql < o.ql;
18         if (ql / sq & 1) return qr < o.qr; // order by parity
19         return qr > o.qr;
20     }
21 } Q[MAXQ];
22 int A[MAXN], ans[MAXQ], Cnt[MAXM], cur, pl = 1, pr = 0, n;
23 inline void add(int pos) {
24     if (Cnt[A[pos]] == 0) cur++;
25     Cnt[A[pos]]++;
26 }
```

```

27 inline void del(int pos) {
28     Cnt[A[pos]]--;
29     if (Cnt[A[pos]] == 0) cur--;
30 }
31 int main() {
32     IOS_ONLY
33     cin >> n;
34     sq = sqrt(n);
35     for (int i = 1; i <= n; ++i) cin >> A[i];
36     int q;
37     cin >> q;
38     for (int i = 0; i < q; ++i) { // offline query
39         cin >> Q[i].ql >> Q[i].qr;
40         Q[i].id = i;
41     }
42     sort(Q, Q + q); // sort, KEY of modui
43     for (int i = 0; i < q; ++i) {
44         while (pl > Q[i].ql) add(--pl);
45         while (pr < Q[i].qr) add(++pr);
46         while (pl < Q[i].ql) del(pl++);
47         while (pr > Q[i].qr) del(pr--);
48         ans[Q[i].id] = cur; // store the rasult
49     }
50     for (int i = 0; i < q; ++i) cout << ans[i] << endl;
51     return 0;
52 }

```