Notes and Tricks

所有数与某值的关系 <=> 极值与某数的关系

找规律

动规初始化：

- 恰好 --> -INF
- 不多于 --> 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 --> 种类并查集，即翻倍的并查集，注意调整合并操作

Data Structure

Balanced BST

```
// 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
/**
 * 插入一个整数 x。
 * 删除一个整数 x（若有多个相同的数，只删除一个）。
 * 查询整数 x 的排名（排名定义为比当前数小的数的个数 +1）。
 * 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）。
 * 求 x 的前驱（小于 x，且最大的数）。
 * 求 x 的后继（大于 x，且最小的数）。
 */
#include <bits/stdc++.h>
using namespace std;
#define ls(x) tree[x].ls
#define rs(x) tree[x].rs
#define num(x) tree[x].num
#define val(x) tree[x].val
```

```cpp
#define sz(x) tree[x].sz
#define exist(x) !(num(x) == 0 && ls(x) == 0 && rs(x) == 0)
const double ALPHA = 0.7;
const int MAXN = 2e6 + 5;
int n, m;
struct Node {
  int ls, rs, num, val, sz;
} tree[MAXN];              // 改用结构体进行存储
vector<int> FP, FN, FV;   // 存储拉平后的节点编号、数目、值
int cnt = 1;
// 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
int flatten(int pos) {
  if (exist(ls(pos)))  // 递归地拉平左子树
    flatten(ls(pos));
  int id = FP.size();  // 记下当前节点的索引
  // 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
  if (num(pos) != 0) {
    FP.push_back(pos);
    FV.push_back(val(pos));
    FN.push_back(num(pos));
  }
  // 递归地拉平右子树
  if (exist(rs(pos))) flatten(rs(pos));
  return id;
}
// 以 pos 为根节点，以 [l,r] 内的信息重建一棵平衡的树
void rebuild(int pos, int l = 0, int r = FP.size() - 1) {
  int mid = (l + r) / 2, sz1 = 0, sz2 = 0;
  if (l < mid) {
    ls(pos) = FP[(l + mid - 1) / 2];  // 重用节点编号
    rebuild(ls(pos), l, mid - 1);     // 递归地重建
    sz1 = sz(ls(pos));
  } else {
    ls(pos) = 0;
  }
  if (mid < r) {
    rs(pos) = FP[(mid + 1 + r) / 2];
    rebuild(rs(pos), mid + 1, r);
    sz2 = sz(rs(pos));
  } else {
    rs(pos) = 0;
  }
  num(pos) = FN[mid];  // 把存于 vector 中的信息复制过来
  val(pos) = FV[mid];
  sz(pos) = sz1 + sz2 + num(pos);  // 递归确定重建后树的大小
}
// 尝试重构当前子树
void try_restructure(int pos) {
  double k = max(sz(ls(pos)), sz(rs(pos))) / double(sz(pos));
  if (k > ALPHA) {
    FP.clear(), FV.clear(), FN.clear();  // 清空 vector
    int id = flatten(pos);
    // 这里是确保当前节点的编号在重构后不会改变
    swap(FP[id], FP[(FP.size() - 1) / 2]);
    rebuild(pos);
  }
}
// 接下来是普通的二叉查找树
void bst_insert(int v, int pos = 1) {
  if (!exist(pos)) {
    val(pos) = v;
    num(pos) = 1;
  } else if (v < val(pos)) {
    if (!exist(ls(pos))) ls(pos) = ++cnt;
    bst_insert(v, ls(pos));
  } else if (v > val(pos)) {
    if (!exist(rs(pos))) rs(pos) = ++cnt;
    bst_insert(v, rs(pos));
  } else
    num(pos)++;
  sz(pos)++;
  try_restructure(pos);
}
void bst_remove(int v, int pos = 1) {
  sz(pos)--;
  if (v < val(pos))
    bst_remove(v, ls(pos));
  else if (v > val(pos))
    bst_remove(v, rs(pos));
  else
    num(pos)--;
  try_restructure(pos);
}
int bst_countl(int v, int pos = 1) {
  if (v < val(pos))
    return exist(ls(pos)) ? bst_countl(v, ls(pos)) : 0;
  else if (v > val(pos))
    return sz(ls(pos)) + num(pos) + (exist(rs(pos)) ? bst_countl(v,
rs(pos)) : 0);
  else
    return sz(ls(pos));
}
int bst_countg(int v, int pos = 1) {
  if (v > val(pos))
    return exist(rs(pos)) ? bst_countg(v, rs(pos)) : 0;
  else if (v < val(pos))
    return sz(rs(pos)) + num(pos) + (exist(ls(pos)) ? bst_countg(v,
ls(pos)) : 0);
  else
    return sz(rs(pos));
}
int bst_rank(int v) { return bst_countl(v) + 1; }
int bst_kth(int k, int pos = 1) {
  if (sz(ls(pos)) + 1 > k)
    return bst_kth(k, ls(pos));
  else if (sz(ls(pos)) + num(pos) < k)
    return bst_kth(k - sz(ls(pos)) - num(pos), rs(pos));
  else
    return val(pos);
}
int bst_pre(int v) {
  int r = bst_countl(v);
  return bst_kth(r);
}
int bst_suc(int v) {
  int r = sz(1) - bst_countg(v) + 1;
  return bst_kth(r);
}
int main() {
  ios::sync_with_stdio(false);
  cin.tie(0);
  cout.tie(0);
  cin >> n >> m;
  for (int i = 0; i < n; i++) {
    int a;
    cin >> a;
    bst_insert(a);
  }
  int lasta = 0;
  vector<int> res;
  while (m--) {
    int op, x;
    cin >> op >> x;
    x ^= lasta;
    if (op == 1) // insert
      bst_insert(x);
    else if (op == 2)  // delete
      bst_remove(x);
    else if (op == 3)  // rank
      lasta = bst_rank(x);
    else if (op == 4)  // k-th
      lasta = bst_kth(x);
    else if (op == 5)  // pre
      lasta = bst_pre(x);
    else if (op == 6)  // suc
      lasta = bst_suc(x);
    if (op > 2) {
      res.push_back(lasta);
    }
  }
  int ans = 0;
  for (auto v : res) ans ^= v;
  cout << ans << endl;
  return 0;
}
```

DSU on Tree

```
/**
 * https://codeforces.com/contest/600/problem/E
 * 树的节点有权，根为 1
 * 一种权占领了一个子树
 * 当且仅当没有其他权在这个子树中出现更多次
 * 求占领每个子树的所有权之和
 * 输入：
 * 节点数
 * 各节点的权
 * 边
 * 输出：
 * 各节点的占领权之和
 *************************
 * 每个节点的答案是其子树的叠加，利用这个性质处理问题
 * 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
 * 用 check[i] 表示颜色 i 有没有出现过，ans[i] 表示出现次数
 * 按以下的步骤遍历一个节点：
 * 遍历其非重儿子，获取它的 ans，但不保留遍历后它的 check
 * 遍历它的重儿子，保留它的 check
 * 再次遍历其非重儿子及其父亲，用重儿子的 check
 * 对遍历到的节点进行计算，获取整棵子树的 ans
 */
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 1e5 + 100;
int n, a[MAXN], tot = -1;
int head[MAXN], to[MAXN << 1], nxt[MAXN << 1];
int bson[MAXN], sz[MAXN];
long long ans[MAXN], sum;
int maxc, flag;
int clr[MAXN];
void add(int u, int v) {
  // 链式前向星
  nxt[++tot] = head[u];
  head[u] = tot;
  to[tot] = v;
  nxt[++tot] = head[v];
  head[v] = tot;
  to[tot] = u;
}
void dfs(int u, int f) {
  sz[u] = 1;
  for (int pp = head[u]; pp != -1; pp = nxt[pp]) {
    int nxt_id = to[pp];
    if (nxt_id == f) continue;
    dfs(nxt_id, u);
    sz[u] += sz[nxt_id];
    if (sz[nxt_id] > sz[bson[u]]) bson[u] = nxt_id;
  }
}
void add(int u, int f, int val) {
  clr[a[u]] += val;
  if (clr[a[u]] > maxc) {
    maxc = clr[a[u]];
    /**********  ans  **********/
    sum = a[u];
    /***************************/
  } else if (clr[a[u]] == maxc) {
    /**********  ans  **********/
    sum += a[u];
    /***************************/
  }
  for (int pp = head[u]; pp != -1; pp = nxt[pp]) {
    int nxt_id = to[pp];
    if (nxt_id == flag || nxt_id == f) continue;
    add(nxt_id, u, val);
  }
}
void dfs(int u, int f, bool keep) {
  for (int pp = head[u]; pp != -1; pp = nxt[pp]) {
    int nxt_id = to[pp];
    if (nxt_id == f || nxt_id == bson[u]) continue;
    dfs(nxt_id, u, 0);
  }
  if (bson[u]) {
    dfs(bson[u], u, 1);
    flag = bson[u];
  }
  add(u, f, 1);
  flag = 0;
  /**********  ans  **********/
  ans[u] = sum;
  /***************************/
  if (!keep) {
    add(u, f, -1);
    /**********  ans  **********/
    maxc = sum = 0;
    /***************************/
  }
}
void solve() {
  int u, v;
  // fill(head+1,head+n+2,-1);
  cin >> n;
  fill(head, head + n + 2, -1);
  for (int i = 1; i <= n; ++i) cin >> a[i];
  for (int i = 1; i < n; ++i) {
    cin >> u >> v;
    add(u, v);
  }
  dfs(1, -1);
  dfs(1, -1, 0);
  for (int i = 1; i < n; ++i) cout << ans[i] << " ";
  cout << ans[n] << "\n";
}
int main() {
  ios::sync_with_stdio(false);
  cin.tie(0);
  cout.tie(0);
  solve();
  return 0;
}
```

BIT

```
// start from 1
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
const ll MAXN = 100005;
ll tree[MAXN];
ll lowbit(int x) { return (x) & (-x); };
void Update(int i, ll x) {
  // increase
  for (int pos = i; pos <= MAXN; pos += lowbit(pos)) {
    tree[pos] += x;
  }
}
ll PrefixQuery(int n) {
  ll ret = 0;
  for (int pos = n; pos; pos -= lowbit(pos)) {
    ret += tree[pos];
  }
  return ret;
}
ll RangeQuery(int ql, int qr) { return PrefixQuery(qr) -
PrefixQuery(ql - 1); }
int main() {
  int a[10] = {-1, 4, 2, 1, 5, 6, 7, 2, 1, 4};
  for (int i = 1; i <= 9; i++) {
    Update(i, a[i]);
  }
  for (int i = 1; i <= 9; i++) {
    cout << PrefixQuery(i) << endl;
  }
  return 0;
}
```

Mono Queue

```
#include <bits/stdc++.h>
// monotonic descending queue, segMax at front
using namespace std;

void getSegMax(vector<int>& v, int k, vector<int>& ans) {
  deque<int> que;
  int n = v.size();
  for (int i = 0; i + 1 < k; ++i) {
    while (!que.empty() && v[que.back()] <= v[i]) que.pop_back();
    que.push_back(i);
  }
  for (int i = k - 1; i < n; ++i) {
    while (!que.empty() && v[que.back()] <= v[i]) que.pop_back();
    que.push_back(i);
    while (que.front() <= i - k) que.pop_front();
    ans.push_back(v[que.front()]);
  }
}
void getSegMin(vector<int>& v, int k, vector<int>& ans) {
  deque<int> que;
  int n = v.size();
  for (int i = 0; i + 1 < k; ++i) {
    while (!que.empty() && v[que.back()] >= v[i]) que.pop_back();
    que.push_back(i);
  }
  for (int i = k - 1; i < n; ++i) {
    while (!que.empty() && v[que.back()] >= v[i]) que.pop_back();
    que.push_back(i);
    while (que.front() <= i - k) que.pop_front();
    ans.push_back(v[que.front()]);
  }
}
int main() {
  vector<int> v = {2, 3, 1, 4, 5, 6, 7, 3};
  vector<int> ans;
  getSegMin(v, 3, ans);
  for (auto itm: ans) {
    cout << itm << " ";
  }
  return 0;
}
```

Segment Tree Range

```
#include <iostream>
using namespace std;
using ll = long long;
const int MAXN = 200005;

struct Node {
  // TODO modify to fit the need
  ll l, r;
  ll ans, mulv, addv;
  Node() {}
};
Node tree[MAXN << 2];
ll n, m, q, rawValues[MAXN];

void MergeNode(Node &f, const Node &lc, const Node &rc) {
  // TODO VARY based on different problems
  f.ans = (lc.ans + rc.ans) % m;
  f.addv = 0;
  f.mulv = 1;
}
void NodeAdd(int k, ll addv) {
}
void NodeMul(int k, ll mulv) {

}
void SpreadTag(Node &f, Node &sn) {
  // TODO VARY based on different problems
  ll addv = f.addv, mulv = f.mulv;
  sn.ans = (sn.ans * mulv % m + (sn.r - sn.l + 1) % m * addv % m) %
m;
  sn.mulv = sn.mulv * mulv % m;
  sn.addv = (sn.addv * mulv % m + addv) % m;
}
void PushUp(int k) {  // up a level
  MergeNode(tree[k], tree[k << 1], tree[k << 1 | 1]);
}
void PushDown(int k) {  // push the lazy tag down a level
  if (!(tree[k].addv == 0 && tree[k].mulv == 1)) {
    SpreadTag(tree[k], tree[k << 1]);
    SpreadTag(tree[k], tree[k << 1 | 1]);
    // TODO reset father's lazy tag
    tree[k].addv = 0;
    tree[k].mulv = 1;
  }
}
void BuildTree(int k, int l, int r) {
  // prepare the nodes
  tree[k].l = l;
  tree[k].r = r;
  if (l == r) {
    // TODO VARY based on different problems
    tree[k].ans = rawValues[l];
    tree[k].addv = 0;
    tree[k].mulv = 1;
  } else {
    int mid = l + (r - l) / 2;
    BuildTree(k << 1, l, mid);
    BuildTree(k << 1 | 1, mid + 1, r);
    PushUp(k);
  }
}
void UpdateSegMul(int k, int l, int r, ll mulv) {
  if (l <= tree[k].l && tree[k].r <= r) {
    // TODO VARY based on problems
    // record the operation for query with smaller range
    tree[k].ans = tree[k].ans * mulv % m;
    tree[k].mulv = tree[k].mulv * mulv % m;
    tree[k].addv = tree[k].addv * mulv % m;
  } else {
    PushDown(k);
    int mid = tree[k].l + (tree[k].r - tree[k].l) / 2;
    if (mid >= l)  // separated update
      UpdateSegMul(k << 1, l, r, mulv);
    if (mid < r) UpdateSegMul(k << 1 | 1, l, r, mulv);
    PushUp(k);
  }
}
void UpdateSegAdd(int k, int l, int r, ll addv) {
  if (l <= tree[k].l && tree[k].r <= r) {
    // TODO VARY based on problems
    tree[k].ans = (tree[k].ans + addv * (tree[k].r - tree[k].l + 1)
% m) % m;
    tree[k].addv = (tree[k].addv + addv) % m;
  } else {
    PushDown(k);
    int mid = tree[k].l + (tree[k].r - tree[k].l) / 2;
    if (mid >= l)  // separated update
      UpdateSegAdd(k << 1, l, r, addv);
    if (mid < r) UpdateSegAdd(k << 1 | 1, l, r, addv);
    PushUp(k);
  }
}
void UpdateDot(int k, int pos, ll val) {
  if (tree[k].l == tree[k].r) {
    // TODO VARY based on problems
    // tree[k].sum = val;
  } else {
    PushDown(k);
    int mid = tree[k].l + (tree[k].r - tree[k].l) / 2;
    if (pos <= mid)  // separated update
      UpdateDot(k << 1, pos, val);
    else
      UpdateDot(k << 1 | 1, pos, val);
    PushUp(k);
  }
}
Node Query(int k, int ql, int qr) {
  if (tree[k].l >= ql && tree[k].r <= qr) return tree[k];
  // when not single, push down firstly, then do the query
  PushDown(k);
  int mid = tree[k].l + (tree[k].r - tree[k].l) / 2;
  Node resL, resR, retVal;
  bool hasL = false, hasR = false;
  if (ql <= mid) {
    hasL = true;
    resL = Query(k << 1, ql, qr);
  }
  if (mid < qr) {
    hasR = true;
    resR = Query(k << 1 | 1, ql, qr);
  }
  if (hasL && hasR)
    MergeNode(retVal, resL, resR);
  else if (hasL)
    retVal = resL;
  else if (hasR)
    retVal = resR;
  return retVal;
}
int main() {
  ios::sync_with_stdio(false);
  cin >> n >> q >> m;
  for (int i = 1; i <= n; i++) cin >> rawValues[i];
  ///////////////////////////////
  BuildTree(1, 1, n);
  ///////////////////////////////
  int t, l, r, v;
  while (q--) {
    cin >> t >> l >> r;
    if (t == 3) {
      cout << Query(1, l, r).ans << "\n";
    } else if (t == 1) {
      cin >> v;
      UpdateSegMul(1, l, r, v);
    } else if (t == 2) {
      cin >> v;
      UpdateSegAdd(1, l, r, v);
    }
  }
  return 0;
}
```

```
/**
 * query the number of elements equal to val
 * which is previously neighbouring pos (inclusive)
 * call (1, 5, 0) for [1, 1, 1, 0, 0, 0, 0, 0] (start from 1)
 * will get 2
 */
int query_prefix_num(int k, int pos, int val) {
  // val == -1 if seg under this node all not all same
  if (tree[k].val == val) return min(pos, tree[k].r) - tree[k].l +
1;
  if (tree[k].l == tree[k].r) return tree[k].val == val;
  push_down(k);
  int mid = tree[k].l + (tree[k].r - tree[k].l) / 2;
  if (pos > mid) {
    int len = query_prefix_num(k << 1 | 1, pos, val);
    if (len == min(pos, tree[k << 1 | 1].r) - tree[k << 1 | 1].l +
1)
      len += query_prefix_num(k << 1, pos, val);
    return len;
  }
  return query_prefix_num(k << 1, pos, val);
}
```

Union Set

```
#include <iostream>
using namespace std;
const int MAXN = 100005;
int father[MAXN];
int trank[MAXN];

void Init(int n) {
  for (int i = 0; i < n; ++i) {
    father[i] = i;
    trank[i] = 0;
  }
}
int Find(int x) {
  if (father[x] == x) {
    return x;
  }
  return father[x] = Find(father[x]);
}
void Unite(int x, int y) {
  x = Find(x);
  y = Find(y);
  if (x == y) {
    return;
  }
  if (trank[x] < trank[y]) {
    father[x] = y;
  } else {
    father[y] = x;
    if (trank[x] == trank[y]) {
      trank[x]++;
    }
  }
}
bool inSame(int x, int y) { return Find(x) == Find(y); }
```

Geometry

```
const double EPS = 1e-9;
bool eq(double a, double b) { return abs(a - b) < EPS; } // ==
bool gt(double a, double b) { return a - b > EPS; }      // >
bool lt(double a, double b) { return a - b < -EPS; }     // <
bool ge(double a, double b) { return a - b > -EPS; }     // >=
bool le(double a, double b) { return a - b < EPS; }      // <=
int sgn (double x) { // sign of a double
    if (fabs(x) < EPS) return 0;
    else if (x < 0) return -1;
    else return 1;
}
// 直线与直线交点
// DEPENDS eq, d*V, V*V, V+V, V^V
vector<Point> inter(Line a, Line b) {
  double c = a.v ^ b.v;
  if (eq(c, 0)) return {};
  Vec v = 1 / c * Vec{a.P ^ (a.P + a.v), b.P ^ (b.P + b.v)};
  return {{v * Vec{-b.v.x, a.v.x}, v * Vec{-b.v.y, a.v.y}}};
}

// 直线与圆交点
// DEPENDS eq, gt, V+V, V-V, V*V, d*V, len, pedal
vector<Point> inter(Line l, Circle C) {
  Point P = pedal(C.O, l);
  double h = len(P - C.O);
  if (gt(h, C.r)) return {};
  if (eq(h, C.r)) return {P};
  double d = sqrt(C.r * C.r - h * h);
  Vec vec = d / len(l.v) * l.v;
  return {P + vec, P - vec};
}

// 圆与圆的交点 注意内含和相离的情况
// DEPENDS eq, gt, V+V, V-V, d*V, len, r90c
vector<Point> inter(Circle C1, Circle C2) {
  Vec v1 = C2.O - C1.O, v2 = r90c(v1);
  double d = len(v1);
  if (gt(d, C1.r + C2.r) || gt(abs(C1.r - C2.r), d)) return {};
  if (eq(d, C1.r + C2.r) || eq(d, abs(C1.r - C2.r)))
    return {C1.O + C1.r / d * v1};
  double a = ((C1.r * C1.r - C2.r * C2.r) / d + d) / 2;
  double h = sqrt(C1.r * C1.r - a * a);
  Vec av = a / len(v1) * v1, hv = h / len(v2) * v2;
  return {C1.O + av + hv, C1.O + av - hv};
}

// 三角形的重心
Point barycenter(Point A, Point B, Point C) {
  return {(A.x + B.x + C.x) / 3, (A.y + B.y + C.y) / 3};
}

// 三角形的外心
// DEPENDS r90c, V*V, d*V, V-V, V+V
// NOTE 给定圆上三点求圆，要先判断是否三点共线
Point circumcenter(Point A, Point B, Point C) {
  double a = A * A, b = B * B, c = C * C;
  double d = 2 * (A.x * (B.y - C.y) + B.x * (C.y - A.y) + C.x * (A.y - B.y));
  return 1 / d * r90c(a * (B - C) + b * (C - A) + c * (A - B));
}

// 三角形的内心
// DEPENDS len, d*V, V-V, V+V
Point incenter(Point A, Point B, Point C) {
  double a = len(B - C), b = len(A - C), c = len(A - B);
  double d = a + b + c;
  return 1 / d * (a * A + b * B + c * C);
}

// 三角形的垂心
// DEPENDS V*V, d*V, V-V, V^V, r90c
Point orthocenter(Point A, Point B, Point C) {
  double n = B * (A - C), m = A * (B - C);
  double d = (B - C) ^ (A - C);
  return 1 / d * r90c(n * (C - B) - m * (C - A));
}
```

Fraction

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
struct Fraction {
  ll up, dn;
  Fraction() : up(0), dn(1) {}
  Fraction(ll _up, ll _dn) : up(_up), dn(_dn) {
    ll cd = __gcd(up, dn);
    up /= cd;
    dn /= cd;
    if (dn < 0) {
      dn = -dn;
      up = -up;
    }
  }
  void reduce() {
    ll cd = __gcd(up, dn);
    up /= cd;
    dn /= cd;
  }
  Fraction operator+(const Fraction &otr) const {
    ll n_dn = dn / __gcd(otr.dn, dn) * otr.dn;
    ////////////// possible overflow /////////////////
    ll n_up = n_dn / dn * up + n_dn / otr.dn * otr.up;
    return Fraction(n_up, n_dn);
  }
  Fraction operator-(const Fraction &otr) const {
    ll n_dn = dn / __gcd(otr.dn, dn) * otr.dn;
    ////////////// possible overflow /////////////////
    ll n_up = n_dn / dn * up - n_dn / otr.dn * otr.up;
    return Fraction(n_up, n_dn);
  }
  Fraction operator*(const Fraction &otr) const {
    ll n_dn = dn * otr.dn;
    ll n_up = up * otr.up;
    // cout << n_up << "/" << n_dn << endl;
    ll cd = __gcd(n_dn, n_up);
    return Fraction(n_up / cd, n_dn / cd);
  }
  Fraction operator/(const Fraction &otr) const {
    Fraction loprd(up, dn), roprd(otr.dn, otr.up);
    return loprd * roprd;
  }
  bool operator==(const Fraction &otr) const {
    ll uup = up, ddn = dn, cd = __gcd(up, dn);
    uup /= up, ddn /= dn;
    ll oup = otr.up, odn = otr.dn;
    cd = __gcd(oup, odn);
    oup /= cd, odn /= cd;
    return up * otr.dn == dn * otr.up;
  }
  bool operator<(const Fraction &otr) const {
    ll uup = up, ddn = dn, cd = __gcd(up, dn);
    uup /= up, ddn /= dn;
    ll oup = otr.up, odn = otr.dn;
    cd = __gcd(oup, odn);
    oup /= cd, odn /= cd;
    return uup * odn < oup * ddn;
  }
  bool operator<=(const Fraction &otr) const {
    Fraction fra{up, dn};
    return fra < otr || fra == otr;
  }
  double real_val() const { return double(up) / double(dn); }
};
int main() {
  Fraction a(1, 2), b(3, 6);
  cout << (a * b).real_val() << endl;
  cout << (a - b).real_val() << endl;
  cout << (a == b) << endl;
  return 0;
}
```

3D Sphere

```
#include <bits/stdc++.h>
using namespace std;
const double PI = acos(-1.0);
struct Sphere {
  double x, y, z, r;
  Sphere() {}
  Sphere(double x, double y, double z, double r) : x(x), y(y),
z(z), r(r) {}
};
double IntersectionVolume(Sphere o, Sphere t) {
  // basic formula: V = (3 * r - h) * h * h * PI / 3
  // calculated from spinning surface calculus
  if (o.r < t.r) swap(o, t);
  double dis = sqrt((o.x - t.x) * (o.x - t.x) + (o.y - t.y) * (o.y
- t.y) +
                    (o.z - t.z) * (o.z - t.z));
  if (dis <= o.r - t.r) {  // completely in
    return 4.0 / 3 * PI * t.r * t.r * t.r;
  } else if (dis <= o.r) {  // center of the smaller sphere in
bigger sphere
    // cosA = (b2 + c2 - a2) / 2bc
    double angleb = acos((t.r * t.r + dis * dis - o.r * o.r) / (2 *
t.r * dis));
    double anglea = PI - angleb;
    double l = t.r * cos(anglea);
    double H = o.r - l - dis;
    double h = t.r - l;
    return 4.0 / 3 * PI * t.r * t.r * t.r - PI / 3 * (3 * t.r - h)
* h * h +
           PI / 3 * (3 * o.r - H) * H * H;
  } else if (dis < o.r + t.r) {  // normal intersection
    double angler = acos((t.r * t.r + dis * dis - o.r * o.r) / (2 *
t.r * dis));
    double angleR = acos((o.r * o.r + dis * dis - t.r * t.r) / (2 *
o.r * dis));
    double H = o.r - o.r * cos(angleR);
    double h = t.r - t.r * cos(angler);
    return PI / 3 * (3 * t.r - h) * h * h + PI / 3 * (3 * o.r - H)
* H * H;
  } else {
    return 0;
  }
}
double IntersectionSurface(Sphere &o, Sphere &t) {
  // basic formula: S = 2 * PI * r * h
  if (o.r < t.r) swap(o, t);
  double dis = sqrt((o.x - t.x) * (o.x - t.x) + (o.y - t.y) * (o.y - t.y) +
          (o.z - t.z) * (o.z - t.z));
  if (dis <= o.r - t.r) {  // completely in
    return 4 * PI * t.r * t.r;
  } else if (dis <= o.r) {  // center of the smaller sphere in bigger sphere
    double angleb = acos((t.r * t.r + dis * dis - o.r * o.r) / (2 * t.r * dis));
    double anglea = PI - angleb;
    double l = t.r * cos(anglea);
    double H = o.r - l - dis;
    double h = t.r - l;
    return 4 * PI * t.r * t.r - 2 * PI * t.r * h + 2 * PI * o.r * H;
  } else if (dis < o.r + t.r) {  // normal intersection
    double angler = acos((t.r * t.r + dis * dis - o.r * o.r) / (2 * t.r * dis));
    double angleR = acos((o.r * o.r + dis * dis - t.r * t.r) / (2 * o.r * dis));
    double H = o.r - o.r * cos(angleR);
    double h = t.r - t.r * cos(angler);
    return 2 * PI * t.r * h + 2 * PI * o.r * H;
  } else {
    return 0;
  }
}
int main() {
  Sphere A, B;
  cin >> A.x >> A.y >> A.z >> A.r;
  cin >> B.x >> B.y >> B.z >> B.r;
  cout << fixed << setprecision(10) << 4*PI*(A.r*A.r+B.r*B.r) - IntersectionSurface(A, B) << endl;
  return 0;
}
```

2D Vector

```
/**
 * structs of
 * point, vector, segment
 * and some operator overloads
 */
// whether a seg AB intersects with a circle O?
// see the endpoints' tangent point (P, Q) angle
// angles: AOP + BOQ < AOB <==> intersect
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
ll MOD = 1e9 + 7;
ll QpowMod(ll bse, ll pwr) {
  ll ret = 1;
  while (pwr) {
    if (pwr & 1) ret = ret * bse % MOD;
    bse = bse * bse % MOD;
    pwr >>= 1;
  }
  return ret;
}
struct Point2 {
  ll x, y;
  Point2() : x(0), y(0) {}
  Point2(ll _x, ll _y) : x(_x), y(_y) {}
  ll Norm2() { return 1ll * x * x + 1ll * y * y; }
  double Norm() { return sqrt(Norm2()); }
  Point2 operator+(const Point2 &po) {
    return Point2(x + po.x, y + po.y);
  }
  Point2 operator-(const Point2 &po) {
    // note the direction
    return Point2(x - po.x, y - po.y);
  }
  bool operator==(const Point2 &po) {
    return x == po.x && y == po.y;
  }
};
typedef Point2 Vector2;
struct Segment2 {
  Point2 s, e;
  Segment2() {}
  Segment2(Point2 _s, Point2 _e) : s(_s), e(_e) {}
};
ll MulCross(const Point2 &p1, const Point2 &p2) {
  return p1.x * p2.y - p1.y * p2.x;
}
ll MulDot(const Point2 &p1, const Point2 &p2) {
  return p1.x * p2.x + p1.y * p2.y;
}
double DisPointToSeg(Point2 p, Point2 s1, Point2 s2) {
  Point2 v1 = p - s1, v2 = s2 - s1;
  if (MulDot(v2, v1) < 0 || MulDot(v2, v1) > v2.Norm2())
    return min(1.0 * (p - s1).Norm(), 1.0 * (p - s2).Norm());
  return abs(1.0 * MulCross(v2, v1) / v2.Norm());
}
int Dis2PointToSeg_INT(Point2 p, Point2 s1, Point2 s2) {
  // square of distance between two points
  Point2 v = p - s1, u = s2 - s1;
  if (MulDot(u, v) < 0 || MulDot(u, v) > u.Norm2())
    return min((p - s1).Norm2(), (p - s2).Norm2()) % MOD;
  return ((MulCross(v, u) % MOD) * (MulCross(v, u) % MOD)) % MOD *
         QpowMod(u.Norm2() % MOD, MOD - 2) % MOD;
}
int main() { return 0; }
```

Vector3ll

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
ll MOD = 1e9 + 7;
struct Point3fra {
  ll x, y, z;
  Point3fra() : x(0), y(0), z(0) {}
  Point3fra(ll _x, ll _y, ll _z) : x(_x), y(_y), z(_z) {}
  ll norm2() { return x * x + y * y + z * z; }
  double norm() { return sqrt(norm2()); }
  Point3fra operator+(const Point3fra &po) {
    return Point3fra(x + po.x, y + po.y, z + po.z);
  }
  Point3fra operator-(const Point3fra &po) {
    return Point3fra(x - po.x, y - po.y, z - po.z);
  }
  bool operator==(const Point3fra &po) {
    return x == po.x && y == po.y && z == po.z;
  }
};
typedef Point3fra Vector3ll;
struct Segment3ll {
  Point3fra s, e;
  Segment3ll() {}
  Segment3ll(Point3fra _s, Point3fra _e): s(_s), e(_e) {}
};
ll mul_dot(const Point3fra &p1, const Point3fra &p2) {
  return p1.x * p2.x + p1.y * p2.y + p1.z * p2.z;
}
Point3fra mul_cross(const Point3fra &p1, const Point3fra &p2) {
  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);
}
int main() {
  Point3fra a{0, 0, 1}, b{1, 1, 1};
  Point3fra c = mul_cross(a, b);
  cout << c.norm() << endl;
  return 0;
}
```

Vector3Fra

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
ll MOD = 1e9 + 7;
struct Fraction {
  ll up, dn;
  Fraction() : up(0), dn(1) {}
  Fraction(ll _up, ll _dn) : up(_up), dn(_dn) {
    ll cd = __gcd(up, dn);
    up /= cd;
    dn /= cd;
  }
  Fraction(Fraction fup, Fraction fdn) {
    Fraction tmp = fup / fdn;
    tmp.reduce();
    up = tmp.up;
    dn = tmp.dn;
  }
  void reduce() {
    ll cd = abs(__gcd(up, dn));
    up /= cd;
    dn /= cd;
    neg_sign();
  }
  void neg_sign() {
    if (dn < 0) {
      dn = -dn;
      up = -up;
    }
  }
  Fraction operator+(const Fraction &otr) const {
    ll n_dn = dn / __gcd(otr.dn, dn) * otr.dn;
    ll n_up = n_dn / dn * up + n_dn / otr.dn * otr.up;
    Fraction ret{n_up, n_dn};
    ret.reduce();
    ret.neg_sign();
    return ret;
  }
  Fraction operator-(const Fraction &otr) const {
    ll n_dn = dn / __gcd(otr.dn, dn) * otr.dn;
    ll n_up = n_dn / dn * up - n_dn / otr.dn * otr.up;
    Fraction ret{n_up, n_dn};
    ret.reduce();
    ret.neg_sign();
    return ret;
  }
  Fraction operator*(const Fraction &otr) const {
    ll n_dn = dn * otr.dn;
    ll n_up = up * otr.up;
    // cout << n_up << "/" << n_dn << endl;
    ll cd = abs(__gcd(n_dn, n_up));
    n_up /= cd, n_dn /= cd;
    Fraction ret{n_up, n_dn};
    ret.reduce();
    ret.neg_sign();
    return ret;
  }
```

```
    Fraction operator/(const Fraction &otr) const {
      Fraction loprd(up, dn), roprd(otr.dn, otr.up);
      return loprd * roprd;
    }
    bool operator==(const Fraction &otr) const {
      ll uup = up, ddn = dn;
      if (ddn < 0) uup = -uup, ddn = -ddn;
      ll cd = abs(__gcd(up, dn));
      uup /= up, ddn /= dn;
      ll oup = otr.up, odn = otr.dn;
      if (odn < 0) oup = -oup, odn = -odn;
      cd = abs(__gcd(oup, odn));
      oup /= cd, odn /= cd;
      return up * otr.dn == dn * otr.up;
    }
    bool operator<(const Fraction &otr) const {
      ll uup = up, ddn = dn;
      if (ddn < 0) uup = -uup, ddn = -ddn;
      ll cd = abs(__gcd(up, dn));
      ll oup = otr.up, odn = otr.dn;
      if (odn < 0) oup = -oup, odn = -odn;
      cd = abs(__gcd(oup, odn));
      oup /= cd, odn /= cd;
      return uup * odn < oup * ddn;
    }
    bool operator<=(const Fraction &otr) const {
      Fraction fra{up, dn};
      return fra < otr || fra == otr;
    }
    double real_val() const { return double(up) / double(dn); }
  };
  struct Point3fra {
    Fraction x, y, z;
    Point3fra() : x(0, 1), y(0, 1), z(0, 1) {}
    Point3fra(Fraction _x, Fraction _y, Fraction _z) : x(_x), y(_y),
  z(_z) {}
    Fraction norm2() { return (x * x) + (y * y) + (z * z); }
    double norm() { return sqrt(norm2().real_val()); }
    Point3fra operator+(const Point3fra &po) {
      return Point3fra(x + po.x, y + po.y, z + po.z);
    }
    Point3fra operator-(const Point3fra &po) {
```

```
    return Point3fra(x - po.x, y - po.y, z - po.z);
  }
  bool operator==(Point3fra &po) {
    return (x == po.x) && (y == po.y) && (z == po.z);
  }
};
typedef Point3fra Vector3fra;
/******** types done ********/
/******* functions go *******/
Fraction frac_zero{0, 1}, frac_one{1, 1};
Fraction mul_dot(const Point3fra &p1, const Point3fra &p2) {
  return p1.x * p2.x + p1.y * p2.y + p1.z * p2.z;
}
Point3fra mul_cross(const Point3fra &p1, const Point3fra &p2) {
  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);
}
Point3fra mul_scale(const Point3fra &p1, const Fraction &s) {
  Fraction sc{s.up, s.dn};
  sc.reduce();
  return Point3fra(p1.x * sc, p1.y * sc, p1.z * sc);
}
bool is_segs_intersect(Point3fra A, Point3fra B, Point3fra C,
Point3fra D) {
  Vector3fra ac = C - A, ad = D - A, ca = A - C, cb = B - C;
  Vector3fra nm_abc = mul_cross(B - A, ac);
  Vector3fra nm_abd = mul_cross(B - A, ad);
  Vector3fra nm_acd = mul_cross(D - C, ca);
  Vector3fra nm_bcd = mul_cross(D - C, cb);
  bool flg1 = mul_dot(nm_abc, nm_abd) < frac_zero &&
mul_cross(nm_abc, nm_abd).norm2() == frac_zero;
  bool flg2 = mul_dot(nm_acd, nm_bcd) < frac_zero &&
mul_cross(nm_acd, nm_bcd).norm2() == frac_zero;
  return flg1 && flg2;
}
Fraction point_to_point2(Point3fra A, Point3fra B) { return (A -
B).norm2(); }
Fraction point_to_seg2(Point3fra P, Point3fra A, Point3fra B) {
  if (A == B) return point_to_point2(P, A);
  Vector3fra ap = P - A, ab = B - A, bp = P - B, ba = A - B;
  if (mul_dot(ap, ab) <= frac_zero || mul_dot(bp, ba) <= frac_zero)
{
    Fraction ret = point_to_point2(P, A);
    ret = min(ret, point_to_point2(P, B));
    return ret;
  } else {
    Vector3fra pa = A - P, pb = B - P, ab = B - A;
    Fraction up = mul_cross(pa, pb).norm2(), dn = ab.norm2();
    return Fraction{up, dn};
  }
}
Fraction seg_to_seg2(Point3fra A, Point3fra B, Point3fra C,
Point3fra D) {
  Vector3fra ca = A - C, cb = B - C, cd = D - C, ab = B - A, ac = C
- A;
  Fraction tmp = mul_dot(mul_cross(ca, cb), cd);
  bool is_intersec = is_segs_intersect(A, B, C, D);
  if (tmp == frac_zero || is_intersec) {
    // same plane or intersect
    if (is_intersec) return frac_zero;
    Fraction ret = point_to_seg2(A, C, D);
    ret = min(ret, point_to_seg2(B, C, D));
    ret = min(ret, point_to_seg2(C, A, B));
    ret = min(ret, point_to_seg2(D, A, B));
    return ret;
  } else {
    // not in same plane, using maxima of two-variable function
    Fraction dn = mul_dot(ab, cd) * mul_dot(ab, cd) - ab.norm2() *
cd.norm2();
    Fraction t(ab.norm2() * mul_dot(cd, ac) - mul_dot(ab, cd) *
mul_dot(ab, ac),
               dn);
    Fraction s(mul_dot(ab, cd) * mul_dot(cd, ac) - cd.norm2() *
mul_dot(ab, ac),
               dn);
    t.reduce();
    s.reduce();
    if (frac_zero < t && t < frac_one && frac_zero < s && s <
frac_one) {
      return point_to_point2(A + mul_scale(ab, s), C +
mul_scale(cd, t));
    } else {
      Fraction ret = point_to_seg2(A, C, D);
      ret = min(ret, point_to_seg2(B, C, D));
      ret = min(ret, point_to_seg2(C, A, B));
      ret = min(ret, point_to_seg2(D, A, B));
      return ret;
    }
  }
}

int main() {
  ios::sync_with_stdio(false);
  cin.tie(0);
  cout.tie(0);
  int t = 1;
  cin >> t;
  while (t--) {
    ll ax, ay, az, bx, by, bz;
    ll cx, cy, cz, dx, dy, dz;
    cin >> ax >> ay >> az >> bx >> by >> bz;
    cin >> cx >> cy >> cz >> dx >> dy >> dz;
    Point3fra A{{ax, 1}, {ay, 1}, {az, 1}};
    Point3fra B{{bx, 1}, {by, 1}, {bz, 1}};
    Point3fra C{{cx, 1}, {cy, 1}, {cz, 1}};
    Point3fra D{{dx, 1}, {dy, 1}, {dz, 1}};
    Fraction ans = seg_to_seg2(A, B, C, D);
    ans.reduce();
    cout << abs(ans.up) << " " << abs(ans.dn) << endl;
  }
  return 0;
}
```

3D dis segment to segment

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
struct Point3 {
  int x, y, z;
  Point3(int x = 0, int y = 0, int z = 0) : x(x), y(y), z(z) {}
  bool operator<(const Point3& u) const {
    return x - u.x < 0 || (x - u.x == 0 && y - u.y < 0) ||
           (x - u.x == 0 && y - u.y == 0 && z - u.z < 0);
  }
  bool operator>(const Point3& u) const { return u < (*this); }
  bool operator==(const Point3& u) const {
    return !(u < (*this) || (*this) < u);
  }
  bool operator!=(const Point3& u) const { return !((*this) == u);
}
  bool operator<=(const Point3& u) const { return *this < u ||
*this == u; }
  bool operator>=(const Point3& u) const { return *this > u ||
*this == u; }
  Point3 operator+(const Point3& u) const {
    return Point3(x + u.x, y + u.y, z + u.z);
  }
  Point3 operator-(const Point3& u) const {
    return Point3(x - u.x, y - u.y, z - u.z);
  }
  Point3 operator*(const int u) const { return Point3(x * u, y * u,
z * u); }
  Point3 operator/(const int u) const { return Point3(x / u, y / u,
z / u); }
  void read() { scanf("%d%d%d", &x, &y, &z); }
};

typedef Point3 Vector3;
ll getDot(Vector3 a, Vector3 b) { return a.x * b.x + a.y * b.y +
a.z * b.z; }
Vector3 getCross(Vector3 a, Vector3 b) {
  return Vector3(a.y * b.z - a.z * b.y, a.z * b.x - a.x * b.z,
                 a.x * b.y - a.y * b.x);
}
ll getPowerLength(Vector3 u) { return getDot(u, u); }
ll gcd(ll a, ll b) { return b == 0 ? a : gcd(b, a % b); }
struct Rat {
  ll s, m;
  Rat(ll s = 0, ll m = 1) {
    ll d = gcd(s, m);
    s /= d, m /= d;
    if (m < 0) m = -m, s = -s;
    this->s = s;
    this->m = m;
  }
  Rat operator+(const Rat& u) const {
    ll d = gcd(m, u.m);
    return Rat(s * (u.m / d) + u.s * (m / d), m * (u.m / d));
  }
  Rat operator-(const Rat& u) const {
    ll d = gcd(m, u.m);
    return Rat(s * (u.m / d) - u.s * (m / d), m * (u.m / d));
  }
  Rat operator*(const Rat& u) const { return Rat(s * u.s, m * u.m);
}
  // Rat operator * (const int& u) const { return Rat(s*u, m); }
  // Rat operator / (const Rat& u) const { return Rat(s*u.m,
m*u.s); }
  // Rat operator / (const int& u) const { return Rat(s, m*u); }
  bool operator<(const Rat& u) const { return s * u.m < u.s * m; }
  bool operator>(const Rat& u) const { return u < (*this); }
  bool operator==(const Rat& u) const { return !(u < (*this) ||
(*this) < u); }
  bool operator!=(const Rat& u) const { return !((*this) == u); }
  bool operator<=(const Rat& u) const { return *this < u || *this
== u; }
  bool operator>=(const Rat& u) const { return *this > u || *this
== u; }
};
inline int dcmp(Rat u) {
  if (u.s == 0)
    return 0;
  else
    return u.s < 0 ? -1 : 1;
}
Rat getDistancePointToSegment(Point3 p, Point3 a, Point3 b) {
  if (a == b) return getPowerLength(p - a);
  Vector3 v1 = b - a, v2 = p - a, v3 = p - b;
  if (getDot(v1, v2) < 0)
    return getPowerLength(v2);
  else if (getDot(v1, v3) > 0)
    return getPowerLength(v3);
  else
    return Rat(getPowerLength(getCross(v1, v2)),
getPowerLength(v1));
}
```

```
bool getDistanceLineToLine(Point3 p1, Vector3 u, Point3 p2, Vector3
v, Rat& s) {
  ll b = getDot(u, u) * getDot(v, v) - getDot(u, v) * getDot(u, v);
  if (b == 0) return false;
  ll a = getDot(u, v) * getDot(v, p1 - p2) - getDot(v, v) *
getDot(u, p1 - p2);
  s = Rat(a, b);
  return true;
}
const ll inf = 0x3f3f3f3f;
int main() {
  int cas;
  scanf("%d", &cas);
  while (cas--) {
    Point3 a, b, c, d;
    a.read(), b.read(), c.read(), d.read();
    Rat s, t, ans(inf);
    bool flag1 = getDistanceLineToLine(a, b - a, c, d - c, s);
    bool flag2 = getDistanceLineToLine(c, d - c, a, b - a, t);
    if (flag1 && flag2 && s.s > 0 && s.s < s.m && t.s > 0 && t.s <
t.m) {
      Vector3 u = b - a, v = d - c;
      Rat x1 = Rat(a.x) + s * u.x, y1 = Rat(a.y) + s * u.y,
          z1 = Rat(a.z) + s * u.z;
      Rat x2 = Rat(c.x) + t * v.x, y2 = Rat(c.y) + t * v.y,
          z2 = Rat(c.z) + t * v.z;
      ans =
          (x2 - x1) * (x2 - x1) + (y2 - y1) * (y2 - y1) + (z2 - z1)
* (z2 - z1);
    } else {
      ans = min(ans, getDistancePointToSegment(a, c, d));
      ans = min(ans, getDistancePointToSegment(b, c, d));
      ans = min(ans, getDistancePointToSegment(c, a, b));
      ans = min(ans, getDistancePointToSegment(d, a, b));
    }
    printf("%lld %lld\n", ans.s, ans.m);
  }
  return 0;
}
```

2D Convex Hull

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
ll MOD = 1e9 + 7;
ll QpowMod(ll bse, ll pwr) {
  ll ret = 1;
  while (pwr) {
    if (pwr & 1) ret = ret * bse % MOD;
    bse = bse * bse % MOD;
    pwr >>= 1;
  }
  return ret;
}
struct Point2 {
  ll x, y;
  Point2() : x(0), y(0) {}
  Point2(ll _x, ll _y) : x(_x), y(_y) {}
  ll Norm2() { return 1ll * x * x + 1ll * y * y; }
  double Norm() { return sqrt(Norm2()); }
  Point2 operator+(const Point2 &po) { return Point2(x + po.x, y + po.y); }
  Point2 operator-(const Point2 &po) {
    // note the direction
    return Point2(x - po.x, y - po.y);
  }
  bool operator==(const Point2 &po) { return x == po.x && y == po.y; }
  bool operator<(const Point2 &po) {
    if (y == po.y) return x < po.x;
    return y < po.y;
  }
};
ll MulCross(const Point2 &p1, const Point2 &p2) {
  return p1.x * p2.y - p1.y * p2.x;
}
ll MulDot(const Point2 &p1, const Point2 &p2) {
  return p1.x * p2.x + p1.y * p2.y;
}
vector<Point2> pts;
bool cmp(int a, int b) {
  if (pts[a].y == pts[b].y) return pts[a].x >= pts[b].x;
  return pts[a].y < pts[b].y;
}
int main() {
  ios::sync_with_stdio(false);
  cin.tie(0);
  cout.tie(0);
  int n;
  cin >> n;
  for (int i = 0; i < n; i++) {
    Point2 t;
    cin >> t.x >> t.y;
    pts.push_back(t);
  }
  sort(pts.begin(), pts.end());
  vector<int> stk;
  stk.push_back(0);
  stk.push_back(1);
  Point2 vec0, vec1;
  for (int i = 2; i < n; i++) {
    if (stk.size() < 2) continue;
    vec0 = pts[stk[stk.size() - 1]] - pts[stk[stk.size() - 2]];
    vec1 = pts[i] - pts[stk.back()];
    while (MulCross(vec0, vec1) < 0) {
      stk.pop_back();
      if (stk.size() < 2) break;
      vec0 = pts[stk[stk.size() - 1]] - pts[stk[stk.size() - 2]];
      vec1 = pts[i] - pts[stk.back()];
    }
    stk.push_back(i);
  }
  stk.push_back(n - 2);
  for (int i = n - 3; i >= 0; i--) {
    if (stk.size() < 2) continue;
    vec0 = pts[stk[stk.size() - 1]] - pts[stk[stk.size() - 2]];
    vec1 = pts[i] - pts[stk.back()];
    while (MulCross(vec0, vec1) < 0) {
      stk.pop_back();
      if (stk.size() < 2) break;
      vec0 = pts[stk[stk.size() - 1]] - pts[stk[stk.size() - 2]];
      vec1 = pts[i] - pts[stk.back()];
    }
    stk.push_back(i);
  }
  stk.pop_back();
  cout << stk.size() << endl;
  for (auto x : stk) {
    cout << pts[x].x << " " << pts[x].y << endl;
  }
  return 0;
}
```

Math

C_n^m

```cpp
#include <stdio.h>
using ll = long long;
const ll MN = 2000000;
const ll MOD = 1000000007;
int fac[MN + 5], inv[MN + 5];

ll qpowMod(ll bse, ll pwr) {
  ll ret = 1;
  while (pwr) {
    if (pwr & 1) ret = ret * bse % MOD;
    bse = bse * bse % MOD;
    pwr >>= 1;
  }
  return ret;
}
void init() {
  fac[0] = 1;
  for (int i = 1; i <= MN; i++) fac[i] = 1ll * fac[i - 1] * i % MOD;
  inv[MN] = qpowMod(fac[MN], MOD - 2);
  for (int i = MN - 1; i >= 0; i--) inv[i] = 1ll * inv[i + 1] * (i + 1) % MOD;
}
int C(int n, int m) {
  if (m > n) return 0;
  return 1ll * fac[n] * inv[m] % MOD * inv[n - m] % MOD;
}
int main() {
  init();
  printf("%d\n", C(5, 3));
  return 0;
}
```

Euler Primers

```cpp
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
const int MAXN = 1e6 + 5;
const int MOD = 1e9 + 7;

int prime[MAXN];
bool vis[MAXN];
int cnt = 0;
ll maxv = -1;
void EulerPrime(int n) {
  for (int i = 2; i <= n; ++i) {
    if (vis[i] == 0) {
      prime[cnt++] = i;
      vis[i] = 1;
    }
    for (int j = 0; i * prime[j] <= n; ++j) {
      vis[i * prime[j]] = 1;
      if (i % prime[j] == 0) break;  // key of O(n)
    }
  }
}
int main() {
  EulerPrime(100);
  for (int i = 0; i < cnt; ++i) printf("%d ", prime[i]);
  printf("\n");
  return 0;
}
```

Josephus Ring

```cpp
// 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）
#include <cstdio>
long long josephus(int n, int k) {
  if (n == 1)
    return 0;
  else
    return (josephus(n - 1, k) + k) % n;
}
```

```
int main(void) {
  long long n, k;
  scanf("%lld %lld", &n, &k);

  printf("%lld\n", 1 + josephus(n, k));
  return 0;
}
// total n, k-th out, find the m-th out, start from 1
void solve(int casei) {
  cout << "Case #" << casei << ": ";
  long long ans = (K - 1) % (N - M + 1);
  if (K == 1) {
    cout << M << endl;
    return;
  }
  for (ll i = N - M + 2; i <= N; i++) {
    ans = (ans + K) % i; // normal iteration
    // jump forward
    ll rem = (i - ans - 1) / K;
    rem = min(rem, N - i); // limit the times of jump
    i += rem; // jump
    ans += rem * K;
  }
  cout << ans + 1 << endl;
}
```

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

```
#include <bits/stdc++.h>
#define inf 0x3f3f3f3f
using namespace std;
typedef long long ll;
const int N = 205, mod = 998244353, MS = 205;
struct Mat {
  ll a[MS][MS];
  ll n, m;
  Mat(int n = 0, int m = 0) : n(n), m(m) { memset(a, 0, sizeof(a));
}
  Mat operator*(const Mat& B) const {
    Mat C(n, B.m);
    for (int i = 1; i <= n; i++)
      for (int j = 1; j <= B.m; j++)
        for (int k = 1; k <= m; k++)
          C.a[i][j] = (C.a[i][j] + a[i][k] * B.a[k][j]) % mod;
    return C;
  }
};
Mat qpow(Mat a, int n) {
  Mat ans(a.n, a.n);
  for (int i = 1; i <= a.n; i++) ans.a[i][i] = 1;
  for (; n; n >>= 1, a = a * a)
    if (n & 1) ans = ans * a;
  return ans;
}
int main() {
  ll n;
  cin >> n;
  string s;
  cin >> s;
  ll now = stol(s);
  Mat A(100, 100);
  A = qpow(A, n);

  Mat B(100, 100);
  B.a[1][1] = 1;
  B = B * A;
  cout << B.a[1][now];
}
```

Quick Power

```
#include <cstdio>
// a^(-1) mod p => a^(p - 2) mod p
// n * n * (n + 1) * (n + 1) / 4 = \sum_{1}^{n} i^3
// n * (n + 1) * (2n + 1) / 6 = \sum_{1}^{n} i^2
using ll = long long;
ll MOD = 1e9+7;
ll QpowMod(ll bse, ll pwr) {
  ll ret = 1;
  while (pwr) {
    if (pwr & 1) ret = ret * bse % MOD;
    bse = bse * bse % MOD;
    pwr >>= 1;
  }
  return ret;
}
int main() {
  printf("%lld", QpowMod(2, 199) * 6 % MOD);
  return 0;
}
```

Graph

SCC kosaraju

```
#include <cstdio>
#include <stack>
using namespace std;
stack<int> stk;
// adjacent matrix
int mp[10][10];
// reversed graph
int mpt[10][10];
int vst[10];
int clr[10];
int vn, en;
void dfs1(int s) {
  if (vst[s] == 1) return;
  vst[s] = 1;
  // dfs routine
  for (int i = 1; i <= vn; ++i) {
    if (mp[s][i] < 0x3f3f3f3f) {
      dfs1(i);
    }
  }
  // push
  stk.push(s);
}
void dfs2(int s, int cnt) {
  if (vst[s] == 0) return;
  clr[s] = cnt;
  vst[s] = 0;
  for (int i = 1; i <= vn; ++i) {
    if (mpt[s][i] < 0x3f3f3f3f) {
      dfs2(i, cnt);
    }
  }
}
```

```
void init() {
  for (int i = 1; i <= vn; ++i) {
    for (int j = 1; j <= vn; ++j) {
      mp[i][j] = mp[j][i] = 0x3f3f3f3f;
      mpt[i][j] = mpt[j][i] = 0x3f3f3f3f;
    }
    mpt[i][i] = mp[i][i] = 0;
  }
}
void SCC_kor() {
  for (int i = 1; i <= vn; ++i) {
    if (vst[i] == 0) dfs1(i);
  }
  int cnt = 1;
  while (!stk.empty()) {
    int s = stk.top();
    stk.pop();
    if (vst[s] == 0) continue;
    dfs2(s, cnt++);
  }
  // vertexes with same value in clr[] is in one SCC
  for (int i = 1; i <= vn; ++i) {
    printf("%d ", clr[i]);
  }
  printf("\n");
}
int main() {
  scanf("%d %d", &vn, &en);
  init();
  for (int i = 1; i <= en; ++i) {
    int fr, to;
    scanf("%d %d", &fr, &to);
    mp[fr][to] = 1;
    mpt[to][fr] = 1;
  }
  SCC_kor();
  return 0;
}
```

SCC tarjan

```cpp
#include <bits/stdc++.h>
using namespace std;
int n, m;
struct node {
  vector<int> nxt;
} g[100000];
int dfn[100000], low[100000], d[100000], col[100000], cnt[100000],
stk[100000];
int vis[100000];
int top, deep, colour;
void tarjan(int u) {
  dfn[u] = low[u] = ++deep;
  stk[top++] = u;
  vis[u] = 1;
  for (int i = 0; i < g[u].nxt.size(); i++) {
    int v = g[u].nxt[i];
    if (!vis[v]) {
      tarjan(v);
      low[u] = min(low[v], low[u]);
    } else {
      low[u] = min(low[v], low[u]);
    }
  }
  if (dfn[u] == low[u]) {
    int node;
    colour++;
    while (node != u) {
      node = stk[top - 1];
      top--;
      col[node] = colour;
    }
  }
}
```

String

KMP

```
int nxt[100005], ns, nt;
char t[100005], s[100005];
void get_next() {
  nxt[0] = -1;
  int k = -1, j = 0;
  while (t[j] != '\0') {
    if (k == -1 || t[k] == t[j]) {
      nxt[++j] = ++k;
    } else {
      k = nxt[k];
    }
  }
}
int search() {
  int id = 0;
  for (int i = 0; i < ns; i++) {
    if (s[i] == t[id]) {
      id++;
    } else {
      while (id != -1 && s[i] != t[id]) id = nxt[id];
      id++;
    }
    if (id == nt) return i - nt + 1;
  }
}
```

Manarcher

```
// find the palindrome in O(n)
#include <bits/stdc++.h>
using namespace std;
char s[100005];
int ps = 0;
int p[100005], ctr, maxr, mirr;
void solve() {
  ctr = maxr = 0;
  for (int i = 0; i < ps; ++i) {
    mirr = 2 * ctr - i;
    if (i < maxr) {
      p[i] = min(maxr - i, p[mirr]);
    } else {
      p[i] = 0;
    }
    while (s[i - 1 - p[i]] == s[i + 1 + p[i]]) {
      p[i]++;
    }
    if (p[i] + i > maxr) {
      ctr = i;
      maxr = p[i] + i;
    }
  }
  int maxi = 0;
  for (int i = 0; i < ps; ++i) {
    maxi = p[maxi] < p[i] ? i : maxi;
  }
  printf("%d\n", p[maxi]);
  for (int i = maxi - p[maxi]; i <= maxi + p[maxi]; ++i) {
    if (s[i] != '#') {
      printf("%c", s[i]);
    }
  }
  printf("\n");
}
int main() {
  int Case = 1;
  while (Case--) {
    char c = getchar();
    s[ps++] = '#';
    while (c != '\n') {
      s[ps++] = c;
      s[ps++] = '#';
      c = getchar();
    }
    solve();
  }
```

```
    return 0;
}
```

Misc

fastIO

```
namespace GTI
{
    char gc(void)
    {
        const int S=1<<17;
        static char buf[S],*s=buf,*t=buf;
        if (s==t) t=buf+fread(s=buf,1,S,stdin);
        if (s==t) return EOF;
        return *s++;
    }
    int gti(void)
    {
        int a=0,b=1,c=gc();
        for (;!isdigit(c);c=gc()) b^=(c=='-');
        for (;isdigit(c);c=gc()) a=a*10+c-'0';
        return b?a:-a;
    }
};
```

Discretization

```
namespace GTI
{
    char gc(void)
    {
        const int S=1<<17;
        static char buf[S],*s=buf,*t=buf;
        if (s==t) t=buf+fread(s=buf,1,S,stdin);
        if (s==t) return EOF;
        return *s++;
    }
    int gti(void)
    {
        int a=0,b=1,c=gc();
        for (;!isdigit(c);c=gc()) b^=(c=='-');
        for (;isdigit(c);c=gc()) a=a*10+c-'0';
        return b?a:-a;
    }
};
```

Inverse Pair Merge Sort

```
using ll = long long;
ll MAXN = 2e5 + 5;
ll n, q[MAXN], tmp[MAXN];
// [l, r]
ll merge_sort(int l, int r) {
  if (l >= r) return 0;
  ll mid = (l + r) >> 1;
  ll res = merge_sort(l, mid) + merge_sort(mid + 1, r);

  ll k = 0, i = l, j = mid + 1;
  while (i <= mid && j <= r) {
    if (q[i] <= q[j])
      tmp[k++] = q[i++];
    else {
      tmp[k++] = q[j++];
      res += mid - i + 1;
    }
  }
  while (i <= mid) tmp[k++] = q[i++];
  while (j <= r) tmp[k++] = q[j++];
  for (ll i = l, j = 0; i <= r; i++, j++) q[i] = tmp[j];
  return res;
}
```

Modui

```cpp
/**
 * Modui range number of distinct values
 */
#include <bits/stdc++.h>
using namespace std;
#define endl "\n";
#define IOS_ONLY               \
  ios::sync_with_stdio(false); \
  cin.tie(0);                  \
  cout.tie(0);
const int MAXN = 30005, MAXQ = 200005, MAXM = 1000005;
int sq;
struct Query {
  int ql, qr, id;
  bool operator<(const Query &o) const {
    // sqrt(n) partitions, assign sq with sqrt(n) first
    if (ql / sq != o.ql / sq) return ql < o.ql;
    if (ql / sq & 1) return qr < o.qr;  // order by parity
    return qr > o.qr;
  }
} Q[MAXQ];
int A[MAXN], ans[MAXQ], Cnt[MAXM], cur, pl = 1, pr = 0, n;
inline void add(int pos) {
  if (Cnt[A[pos]] == 0) cur++;
  Cnt[A[pos]]++;
}
inline void del(int pos) {
  Cnt[A[pos]]--;
  if (Cnt[A[pos]] == 0) cur--;
}
int main() {
  IOS_ONLY
  cin >> n;
  sq = sqrt(n);
  for (int i = 1; i <= n; ++i) cin >> A[i];
  int q;
  cin >> q;
  for (int i = 0; i < q; ++i) {  // offline query
    cin >> Q[i].ql >> Q[i].qr;
    Q[i].id = i;
  }
  sort(Q, Q + q);  // sort, KEY of modui
  for (int i = 0; i < q; ++i) {
    while (pl > Q[i].ql) add(--pl);
    while (pr < Q[i].qr) add(++pr);
    while (pl < Q[i].ql) del(pl++);
    while (pr > Q[i].qr) del(pr--);
    ans[Q[i].id] = cur;  // store the rasult
  }
  for (int i = 0; i < q; ++i) cout << ans[i] << endl;
  return 0;
}
```