Competitive Programming

Ritik Singla

Index

1 Classical

1.1 Knapsack

```
template<typename T>
struct Knapsack {
    int N;
    T C;
    vector<T>values, weights;
    Knapsack(vector<T>values_, vector<T>weights_, T C_):
       values(values_), weights(weights_), C(C_) {
        assert((int)values.size() == (int)weights.size());
        N = (int)values.size();
    }
    T bottom_to_top_iterative() {
        vector<vector<T>>dp(N + 1, vector<T>(C + 1));
        for (int i = 1; i <= N; ++i) {
            for (int j = 0; j <= C; ++j) {
                if (j - weights[i - 1] >= 0) {
                    dp[i][j] = max(dp[i][j], dp[i - 1][j -
                       weights[i - 1]] + values[i - 1]);
                }
                dp[i][j] = max(dp[i][j], dp[i - 1][j]);
            }
        }
        return dp[N][C];
```

```
}

/*
To compute dp[i][j], we only need solution of previous
   row.
In 0-1 Knapsack Problem if we are currently on dp[i][j]
   and we include ith element
then we move j-wt[i] steps back in previous row and if
   we exclude the current
element we move on jth column in previous row. So here
   we can observe that at
a time we are working only with 2 consecutive rows.
*/
T optimized_space_iterative()
{
    vector<T>dp(C + 1, 0);
    for (int i = 0; i < N; i++) {
        for (int j = C; j >= weights[i]; j--) {
            dp[j] = max(dp[j], values[i] + dp[j -
               weights[i]]);
        }
    }
    return dp[C];
}
T bottom_to_top_recursive(int idx, int taken,
   vector<vector<T>>&dp) {
    if (taken < 0)
        return numeric_limits<T>::min();
    if (taken == 0 || idx == 0)
        return 0;
    T &ans = dp[idx][taken];
    if (ans != -1)
        return ans;
    ans = 0;
    T exclude = bottom_to_top_recursive(idx - 1, taken,
       dp);
    T include = bottom_to_top_recursive(idx - 1, taken -
       weights[idx - 1], dp) + values[idx - 1];
    ans = max(include, exclude);
    return ans;
}
T bottom_to_top_recursive() {
        vector<vector<T>>dp(N + 1, vector<T>(C + 1, -1));
        return bottom_to_top_recursive(N, C, dp);
    }
    T top_to_bottom_recursive(int idx, int taken,
       vector<vector<T>>&dp) {
        if (taken > C)
            return numeric_limits<T>::min();
        if (taken == C || idx == N)
            return 0;
        T &ans = dp[idx][taken];
        if (ans != -1)
            return ans;
        ans = 0;
        T exclude = top_to_bottom_recursive(idx + 1, taken,
           dp);
        T include = top_to_bottom_recursive(idx + 1, taken +
           weights[idx], dp) + values[idx];
        ans = max(include, exclude);
        return ans;
    }
    T top_to_bottom_recursive() {
        vector<vector<T>>dp(N + 1, vector<T>(C + 1, -1));
        return top_to_bottom_recursive(0, 0, dp);
    }
};
```

1.2 Sorting Algorithms

```
template<typename T>
struct Sort {
    int N;
    vector<T>A;
    Sort(const vector<T>&_A) {
        A = _A;
        N = (int)A.size();
    }
    void check() {
        for (int i = 0; i + 1 < N; i++) {
            assert(A[i] <= A[i + 1]);
        }
```

```
}
int partition(int lo, int hi) {
    // Choosing pivot as last element
    int pivot = A[hi];
    int l = lo - 1;
    for (int r = lo; r <= hi; r++) {
        if (A[r] < pivot) {
            swap(A[++l], A[r]);
        }
    }
    swap(A[++l], A[hi]);
    return l;
}
void quickSort(int lo, int hi) {
    if (lo < hi) {
        int p = partition(lo, hi);
        quickSort(lo, p - 1);
        quickSort(p + 1, hi);
    }
}
void quickSort() {
    int lo = 0, hi = N - 1;
    quickSort(lo, hi);
    check();
}
void merge(int lo, int mid, int hi) {
    int n = (hi - lo + 1);
    vector<T>b(n);
    int idx{};
    int l = lo, r = mid + 1;
    for (; l <= mid && r <= hi;) {
        if (A[l] <= A[r]) {
            b[idx++] = A[l++];
        }
        else {
            b[idx++] = A[r++];
        }
    }
    for (; l <= mid;) {
        b[idx++] = A[l++];
    }
    for (; r <= hi;) {
        b[idx++] = A[r++];
    }
    for (int i = lo; i <= hi; i++) {
        A[i] = b[i - lo];
    }
}
void mergeSort(int lo, int hi) {
    if (lo >= hi) return;
    int mid = (lo + (hi - lo) / 2);
    mergeSort(lo, mid);
    mergeSort(mid + 1, hi);
    merge(lo, mid, hi);
}
void mergeSort() {
    int lo = 0, hi = N - 1;
    mergeSort(lo, hi);
    check();
}
void max_heapify(int idx, int n) {
    int maxi = idx;
    int l = 2 * idx + 1;
    int r = 2 * idx + 2;
    if (l < n && A[l] > A[maxi])
        maxi = l;
    if (r < n && A[r] > A[maxi])
        maxi = r;
    if (maxi != idx) {
        swap(A[maxi], A[idx]);
        max_heapify(maxi, n);
    }
}
void maxHeapSort() {
    for (int i = N / 2 - 1; i >= 0; i--) {
        max_heapify(i, N);
    }
    for (int i = N - 1; i; i--) {
        swap(A[0], A[i]);
        max_heapify(0, i);
    }
    check();
}
void min_heapify(int idx, int n) {
```

```
    int mini = idx;
    int l = 2 * idx + 1;
    int r = 2 * idx + 2;
    if (l < n && A[l] < A[mini])
        mini = l;
    if (r < n && A[r] < A[mini])
        mini = r;
    if (mini != idx) {
        swap(A[idx], A[mini]);
        min_heapify(mini, n);
    }
}
void minHeapSort() {
    for (int i = N / 2 - 1; i >= 0; i--) {
        min_heapify(i, N);
    }
    for (int i = N - 1; i; i--) {
        swap(A[0], A[i]);
        min_heapify(0, i);
    }
    reverse(begin(A), end(A));
    check();
}
void selectionSort() {
    for (int i = 0; i + 1 < N; ++i) {
        int mini = i;
        for (int j = i + 1; j < N; j++) {
            if (A[j] < A[mini])
                mini = j;
        }
        swap(A[mini], A[i]);
    }
    check();
}
void insertionSort() {
    for (int i = 1; i < N; i++) {
        int j = i - 1;
        int x = A[i];
        while (j >= 0 && A[j] > x) {
            A[j + 1] = A[j];
            j--;
        }
        A[j + 1] = x;
    }
    check();
}
void bubbleSort() {
    for (int i = 0; i < N - 1; i++) {
        for (int j = 0; j < N - i - 1; j++) {
            if (A[j] > A[j + 1])
                swap(A[j], A[j + 1]);
        }
    }
    check();
}
void shuffle() {
    random_shuffle(begin(A), end(A));
}
void show() {
    for (auto &a : A) cout << a << ' ';
    cout << '\n';
}
};
```

2 Geometry

2.1 2D Basics

```
struct point2d
{
    ld x, y;
    point2d() {}
    point2d(ld x, ld y): x(x), y(y) {}
    point2d &operator+=(const point2d &t)
    {
        x += t.x;
        y += t.y;
        return *this;
    }
    point2d &operator-=(const point2d &t)
    {
```

```
        x -= t.x;
        y -= t.y;
        return *this;
    }
    point2d &operator*=(ld t)
    {
        x *= t;
        y *= t;
        return *this;
    }
    point2d &operator/=(ld t)
    {
        x /= t;
        y /= t;
        return *this;
    }
    point2d operator+(const point2d &t) const
    {
        return point2d(*this) += t;
    }
    point2d operator-(const point2d &t) const
    {
        return point2d(*this) -= t;
    }
    point2d operator*(ld t) const
    {
        return point2d(*this) *= t;
    }
    point2d operator/(ld t) const
    {
        return point2d(*this) /= t;
    }
};
point2d operator*(ld a, point2d b)
{
    return b * a;
}
ld cross(point2d a, point2d b)
{
    return a.x * b.y - a.y * b.x;
}
ld dot(point2d a, point2d b)
{
    return a.x * b.x + a.y * b.y;
}
// takes (x1,y1),(x2-x1,y2-y1),(x3,y3),(x4-x3,y4-y3)
// Form: r=a1+t*d1 where {a1} is a starting position vector
   and {d1} is the parallel vector to the given line.
// r=a1+t*d1 and cross_product((r-a2),d2)=0
// Therefore,
   t=a1+(cross_product((a2-a1),d2)/cross_product(d1,d2))*d1
point2d intersect(point2d a1, point2d d1, point2d a2,
   point2d d2)
{
    if(cross(d1, d2) == 0)
    {
        cout << "Parallel lines\n";
        return point2d();
    }
    return a1 + cross(a2 - a1, d2) / cross(d1, d2) * d1;
}
ld norm(point2d a)
{
    return dot(a, a);
}
ld abs(point2d a)
{
    return sqrt(norm(a));
}
ld proj(point2d a, point2d b)
{
    return dot(a, b) / abs(b);
}
ld angle(point2d a, point2d b)
{
    return acos(dot(a, b) / abs(a) / abs(b));
}
```

2.2 3D Basics

```
struct point3d
{
```

```
ld x, y, z;
point3d() {}
point3d(ld x, ld y, ld z): x(x), y(y), z(z) {}
point3d &operator+=(const point3d &t)
{
    x += t.x;
    y += t.y;
    z += t.z;
    return *this;
}
point3d &operator-=(const point3d &t)
{
    x -= t.x;
    y -= t.y;
    z -= t.z;
    return *this;
}
point3d &operator*=(ld t)
{
    x *= t;
    y *= t;
    z *= t;
    return *this;
}
point3d &operator/=(ld t
                    ld dot(point3d a, point3d b)
{
    return a.x * b.x + a.y * b.y + a.z * b.z;
})
{
    x /= t;
    y /= t;
    z /= t;
    return *this;
}
point3d operator+(const point3d &t) const
{
    return point3d(*this) += t;
}
point3d operator-(const point3d &t) const
{
    return point3d(*this) -= t;
    }
    point3d operator*(ld t) const
    {
        return point3d(*this) *= t;
    }
    point3d operator/(ld t) const
    {
        return point3d(*this) /= t;
    }
};
point3d operator*(ld a, point3d b)
{
    return b * a;
}
ld dot(point3d a, point3d b)
{
    return a.x * b.x + a.y * b.y + a.z * b.z;
}

point3d cross(point3d a, point3d b)
{
    return point3d(a.y * b.z - a.z * b.y,
                   a.z * b.x - a.x * b.z,
                   a.x * b.y - a.y * b.x);
}
ld triple(point3d a, point3d b, point3d c)
{
    return dot(a, cross(b, c));
}
// Used Cramer's Rule, triple product is just the
   determinant of the coordinates.
// Accepts form r*ni=d*ni for i=1,2,3
point3d intersect(point3d a1, point3d n1, point3d a2,
   point3d n2, point3d a3, point3d n3)
{
    point3d x(n1.x, n2.x, n3.x);
    point3d y(n1.y, n2.y, n3.y);
    point3d z(n1.z, n2.z, n3.z);
    point3d d(dot(a1, n1), dot(a2, n2), dot(a3, n3));
    return point3d(triple(d, y, z),
                   triple(x, d, z),
                   triple(x, y, d)) / triple(n1, n2, n3);
```

```
}
```

2.3 Check Intersection

```
struct pt
{
    ll x, y;
    pt(): x(), y() {}
    pt(ll x_, ll y_): x(x_), y(y_) {}
    pt operator-(const pt &a) const
    {
        return pt(x - a.x, y - a.y);
    }
    ll cross (const pt &a) const
    {
        return x * a.y - y * a.x;
    }
    ll cross (const pt &a, const pt &b) const // Cross
       Product of (a-this*),(b-this*)
    {
        return (a - *this).cross(b - *this);
    }
};
// Orientation based on sign
ll orientation(const ll &x)
{
    return (x >= 0 ? x ? 1 : 0 : -1);
}
// If two segments overlap or intersect on more than 1 point
// Check for (x1,x2,x3,x4) and then (y1,y2,y3,y4)
bool overlap(ll a, ll b, ll c, ll d)
{
    if (a > b)
        swap(a, b);
    if (c > d)
        swap(c, d);
    return max(a, c) <= min(b, d);
}
bool intersect(const pt &a, const pt &b, const pt &c, const
   pt &d)
{
    if (c.cross(a, d) == 0 && c.cross(b, d) == 0)   //If d
       lies on same side for both a,b means d is collinear
       with a,b.
        return overlap(a.x, b.x, c.x, d.x) && overlap(a.y,
           b.y, c.y, d.y);
    return orientation(a.cross(b, c)) !=
       orientation(a.cross(b, d)) && orientation(c.cross(d,
       a)) != orientation(c.cross(d, b));
}
```

2.4 Circles

```
struct Point
{
    ld x, y;
    Point(ld px, ld py)
    {
        x = px;
        y = py;
    }
    Point sub(Point p2)
    {
        return Point(x - p2.x, y - p2.y);
    }
    Point add(Point p2)
    {
        return Point(x + p2.x, y + p2.y);
    }
    ld distance(Point p2)
    {
        return sqrt((x - p2.x) * (x - p2.x) + (y - p2.y) *
           (y - p2.y));
    }
    Point normal()
    {
        ld length = sqrt(x * x + y * y);
        return Point(x / length, y / length);
    }
    Point scale(ld s)
```

```
    {
        return Point(x * s, y * s);
    }
};
struct circle
{
    ld x, y, r;
    pair<Point, Point> intersections(circle c)
    {
        Point P0(x, y);
        Point P1(c.x, c.y);
        ld d, a, h;
        d = P0.distance(P1);
        a = (r * r - c.r * c.r + d * d) / (2 * d);//
           Distance from P0 of axis of intersection
        h = sqrt(r * r - a * a);// Height of point above a
        Point P2 = P1.sub(P0).scale(a / d).add(P0);//Line
           Ratio Theorem
        ld x3, y3, x4, y4;
        x3 = P2.x + h * (P1.y - P0.y) / d;
        y3 = P2.y - h * (P1.x - P0.x) / d;
        x4 = P2.x - h * (P1.y - P0.y) / d;
        y4 = P2.y + h * (P1.x - P0.x) / d;
        return pair<Point, Point>(Point(x3, y3), Point(x4,
           y4));
    }
};
```

2.5 Closest Points

```
// O(N*log(N)*log(N))
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
const ll maxn = 1e5 + 10;
struct point
{
    ll x, y;
    point operator-(point a)
    {
        return {x - a.x, y - a.y};
    }
    ll dist()
    {
        return x * x + y * y;
    }
};
bool by_x(point &a, point &b)
{
    return a.x < b.x;
}
bool by_y(point &a, point &b)
{
    return a.y < b.y;
}
ll n, x;
vector<point>vec(maxn);
vector<ll>pre(maxn);
ll solve2(ll l, ll r)
{
    if(l == r)
        return (ll)1e18;
    ll mid = (l + r) / 2;
    sort(vec.begin() + l, vec.begin() + r + 1, by_x);
    ll d = min(solve2(l, mid), solve2(mid + 1, r));
    ll midx = vec[mid].x;
    vector<point>strip;
    forn(i, l, r)
    {
        if(point{vec[i].x - midx, 0}.dist() < d)
            strip.push_back(vec[i]);
    }
    sort(all(strip), by_y);
    rep(i, strip.size())
    {
        forn(j, i + 1, strip.size() - 1)
        {
            if(point{0, strip[i].y - strip[j].y}.dist() > d)
                break;
            d = min((strip[i] - strip[j]).dist(), d);
        }
    }
```

```
    return d;
}
void solve()
{
    cin >> n;
    rep(i, n)
    {
        cin >> x;
        pre[i + 1] = pre[i] + x;
    }
    rep(i, n)
    {
        vec[i] = point{i + 1, pre[i + 1]};
    }
    cout << solve2(0, n - 1);
}
int32_t main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);
    ll t = 1;
    rep(i, t)
    {
        solve();
    }
    return 0;
}
```

2.6 Convex Hull

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
const ll maxn = 1e5 + 10;
struct pt
{
    double x, y;
    pt() {}
    pt(double x_, double y_): x(x_), y(y_) {}
    bool operator<(const pt &a)
    {
        return (y < a.y || (y == a.y && x < a.x));
    }
    pt operator-(const pt &a) const
    {
        return pt(x - a.x, y - a.y);
    }
    double dist(const pt &a)
    {
        return x * a.x + y * a.y;
    }
};
int n, x, y;
vector<pt>p;
double circumference(vector<pt> &a)
{
    double res{};
    int n = a.size();
    if(n == 1)
        return 0.00;
    rep(i, n - 1)
    {
        res += sqrt((a[i + 1].x - a[i].x) * (a[i + 1].x -
            a[i].x) + (a[i + 1].y - a[i].y) * (a[i + 1].y -
            a[i].y));
    }
    res += sqrt((a[n - 1].x - a[0].x) * (a[n - 1].x -
        a[0].x) + (a[n - 1].y - a[0].y) * (a[n - 1].y -
        a[0].y));
    return res;
}
bool cw(pt a, pt b, pt c)
{
    return ((a.x * (b.y - c.y) + b.x * (c.y - a.y) + c.x *
        (a.y - b.y)) < 0);
}
bool ccw(pt a, pt b, pt c)
{
    return ((a.x * (b.y - c.y) + b.x * (c.y - a.y) + c.x *
        (a.y - b.y)) > 0);
}
```

```
void solve()
{
    cin >> n;
    p.clear();
    map<pair<int, int>, int>mp;
    rep(i, n)
    {
        cin >> x >> y;
        if(mp[ {x, y}] == 0)
        {
            p.emplace_back(x, y);
            mp[ {x, y}] = i + 1;
        }
    }
    n = p.size();
    sort(all(p));
    if(n <= 2)
    {
        cout << fixed << setprecision(2) << circumference(p)
           << "\n";
        rep(i, n)
        cout << mp[ {p[i].x, p[i].y}] << " ";
        cout << "\n\n";
        return;
    }
    vector<pt>up, down;
    pt a = p[0], b = p[n - 1];
    up.emplace_back(a);
    down.emplace_back(a);
    for(int i = 1; i < n; i++)
    {
        if(cw(a, p[i], b) || i == n - 1)
        {
            while(up.size() >= 2 && !cw(up[up.size() - 2],
               up[up.size() - 1], p[i]))
                up.pop_back();
            up.emplace_back(p[i]);
        }
        if(ccw(a, p[i], b) || i == n - 1)
        {
            while(down.size() >= 2 && !ccw(down[down.size()
               - 2], down[down.size() - 1], p[i]))
                down.pop_back();
            down.emplace_back(p[i]);
        }
    }
    p.clear();
    rep(i, down.size())
    p.emplace_back(down[i]);
    for(int i = up.size() - 2; i > 0; i--)
        p.emplace_back(up[i]);
    cout << fixed << setprecision(2) << circumference(p) <<
       "\n";
    rep(i, p.size())
    cout << mp[ {p[i].x, p[i].y}] << " ";
    cout << "\n\n";
    return;
}
int32_t main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);
    int t;
    cin >> t;
    rep(i, t)
    {
        solve();
    }
    return 0;
}
```

2.7 Point in Convex Polygon

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
const ll maxn = 1e5 + 10;
struct pt
{
    ll x, y;
};
```

```
//Sort slopes in anticlockwise order
//(y1-0)/(x1-0)<(y2-0)/(x2-0)
bool operator<(const pt &a, const pt &b)
{
    return (a.y * b.x < b.y * a.x);
}
// Checks if slope of line (point->a)>(point->b)
bool orientation(pt a, pt b, pt point)
{
    return (a.y - point.y) * (b.x - point.x) >= (a.x -
       point.x) * (b.y - point.y);
}
void solve()
{
    ll n, m, k;
    cin >> n >> m >> k;
    vector<pt>p(n);
    ll id{};
    rep(i, n)
    {
        cin >> p[i].x >> p[i].y;
        if(p[i].x < p[id].x || (p[i].x == p[id].x && p[i].y
           < p[id].y))
            id = i;
    }
    pt zero = p[id];
    rotate(p.begin(), p.begin() + id, p.end());
    p.erase(p.begin());
    vector<pt>seq(--n);
    rep(i, n)
    {
        seq[i].x = p[i].x - zero.x;
        seq[i].y = p[i].y - zero.y;
    }
    ll x, y, ans{};
    rep(i, m)
    {
        bool flag = false;
        pt point, q;
        cin >> point.x >> point.y;
        q.x = point.x;
        q.y = point.y;
        if(point.x > zero.x || (point.x == zero.x && point.y
           >= zero.y))
        {
            point.x = point.x - zero.x, point.y = point.y -
               zero.y;
            ll idx = upper_bound(seq.begin(), seq.end(),
               point) - seq.begin();
            if(idx == n)
            {
                // Same slope, i.e. collinear on y direction
                // point.y/point.x==seq[i].y/seq[i].x
                if(point.y * seq[n - 1].x == point.x * seq[n
                   - 1].y)
                    idx--;
            }
            if(idx && idx != n)
            {
                if(orientation(p[idx], p[idx - 1], q))
                    flag = true;
            }
        }
        ans += flag;
    }
    if(ans >= k)
        cout << "YES\n";
    else
        cout << "NO\n";
}
int32_t main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);
    ll t = 1;
    rep(i, t)
    {
        solve();
    }
    return 0;
}
```

3 Graphs

3.1 0-1 BFS

```
deque<int>q;
const int inf = 1e5 + 5;
vector<int>dist(n, inf);
dist[0] = 0;
bool vis[n];
memset(vis, 0, sizeof vis);
q.push_back(0);
while (!q.empty())
{
    int u = q.front();
    q.pop_front();
    vis[u] = 1;
    for (auto v : g.edges[u])
    {
        if (dist[v.to] > dist[u] + v.cost)
        {
            dist[v.to] = dist[u] + v.cost;
            if (v.cost)
                q.push_back(v.to);
            else
                q.push_front(v.to);
        }
    }
}
```

3.2 Bipartite Check

```
vector<int>color(n, -1);
bool flag = true;
for (int i = 0; i < n && flag; i++)
{
    if (color[i] == -1)
    {
        queue<int>q;
        q.push(i);
        color[i] = 0;
        while (!q.empty() && flag)
        {
            int u = q.front();
            q.pop();
            for (auto v : g.adj[u])
            {
                if (color[v.to] == -1)
                {
                    color[v.to] = color[u] ^ 1;
                    q.push(v.to);
                }
                else
                {
                    flag &= color[v.to] != color[u];
                }
            }
        }
    }
}
```

3.3 Bridges

```
const int maxn = 1e5 + 10;
vector<int>g[maxn];
bool vis[maxn];
int timer;
vector<int>tin, low;
int n, m;
void dfs(int u, int p = -1)
{
    vis[u] = true;
    tin[u] = low[u] = timer++;
    for (int v : g[u])
    {
        if (v == p) continue;
        if (vis[v])
        {
            low[u] = min(low[u], tin[v]);
        }
```

```
        else
        {
            dfs(v, u);
            low[u] = min(low[u], low[v]);
            if (low[v] > tin[u])
            {
                is_bridge(u, v);
            }
        }
    }
}
void find_bridges() {
    timer = 0;
    for (int i = 0; i < n; i++) {
        vis[i] = 0;
        tin[i] = low[i] = 0;
    }
    for (int i = 0; i < n; ++i) {
        if (!vis[i])
            dfs(i);
    }
}
```

3.4 Cutpoints

```
const int maxn = 1e5 + 5;
int to[maxn], tin[maxn];
int timer;
bool vis[maxn];
vector<int>g[maxn];
set<int>s;
void init(int n)
{
    for(int i = 0; i < n; i++)
    {
        g[i].clear();
        vis[i] = to[i] = tin[i] = 0;
    }
    timer = 0;
    s.clear();
}
void is_articulation_point(int u)
{
    s.insert(u); //Use set instead of incrementing as each
       vertex may be considered more than once.
}
void articulation_point(int u, int p = -1)
{
    vis[u] = 1;
    tin[u] = to[u] = timer++;
    int children = 0;
    for(auto v : g[u])
    {
        if(v == p)
            continue;
        if(vis[v])
        {
            to[u] = min(to[u], tin[v]);
        }
        else
        {
            articulation_point(v, u);
            to[u] = min(to[u], to[v]);
            if(to[v] >= tin[u] && p != -1)
            {
                is_articulation_point(u);
            }
            children++;
        }
    }
    if(p == -1 && children > 1)
    {
        is_articulation_point(u);
    }
}
```

3.5 DFS Forest

```
template <typename T>
class dfs_forest : public forest<T> {
```

```
public:
    using forest<T>::edges;
    using forest<T>::g;
    using forest<T>::n;
    vector<int> pv;
    vector<int> pe;
    vector<int> order;
    vector<int> pos;
    vector<int> end;
    vector<int> sz;
    vector<int> root;
    vector<int> depth;
    vector<T> dist;

    dfs_forest(int _n) : forest<T>(_n) {
    }

    void init() {
        pv = vector<int>(n, -1);
        pe = vector<int>(n, -1);
        order.clear();
        pos = vector<int>(n, -1);
        end = vector<int>(n, -1);
        sz = vector<int>(n, 0);
        root = vector<int>(n, -1);
        depth = vector<int>(n, -1);
        dist = vector<T>(n);
    }


    void clear() {
        pv.clear();
        pe.clear();
        order.clear();
        pos.clear();
        end.clear();
        sz.clear();
        root.clear();
        depth.clear();
        dist.clear();
    }
```

```
private:
    void do_dfs(int v) {
        pos[v] = (int) order.size();
        order.push_back(v);
        sz[v] = 1;
        for (int id : g[v]) {
            if (id == pe[v]) {
                continue;
            }
            auto &e = edges[id];
            int to = e.from ^ e.to ^ v;
            depth[to] = depth[v] + 1;
            dist[to] = dist[v] + e.cost;
            pv[to] = v;
            pe[to] = id;
            root[to] = (root[v] != -1 ? root[v] : to);
            do_dfs(to);
            sz[v] += sz[to];
        }
        end[v] = (int) order.size() - 1;
    }

    void do_dfs_from(int v) {
        depth[v] = 0;
        dist[v] = T{};
        root[v] = v;
        pv[v] = pe[v] = -1;
        do_dfs(v);
    }

public:
    void dfs(int v, bool clear_order = true) {
        if (pv.empty()) {
            init();
        } else {
            if (clear_order) {
                order.clear();
            }
        }
        do_dfs_from(v);
    }
```

```
    void dfs_all() {
        init();
        for (int v = 0; v < n; v++) {
            if (depth[v] == -1) {
                do_dfs_from(v);
            }
        }
        assert((int) order.size() == n);
    }
};
```

3.6 DFS Undigraph

```
template <typename T>
class dfs_undigraph : public undigraph<T> {
public:
    using undigraph<T>::edges;
    using undigraph<T>::g;
    using undigraph<T>::n;

    vector<int> pv;
    vector<int> pe;
    vector<int> order;
    vector<int> pos;
    vector<int> end;
    vector<int> sz;
    vector<int> root;
    vector<int> depth;
    vector<int> min_depth;
    vector<T> dist;
    vector<int> was;
    int attempt;

    dfs_undigraph(int _n) : undigraph<T>(_n) {
    }

    void init() {
        pv = vector<int>(n, -1);
        pe = vector<int>(n, -1);
        order.clear();
        pos = vector<int>(n, -1);
        end = vector<int>(n, -1);
        sz = vector<int>(n, 0);
        root = vector<int>(n, -1);
        depth = vector<int>(n, -1);
        min_depth = vector<int>(n, -1);
        dist = vector<T>(n);
        was = vector<int>(n, -1);
        attempt = 0;
    }

    void clear() {
        pv.clear();
        pe.clear();
        order.clear();
        pos.clear();
        end.clear();
        sz.clear();
        root.clear();
        depth.clear();
        min_depth.clear();
        dist.clear();
        was.clear();
    }

private:
    void do_dfs(int v) {
        was[v] = attempt;
        pos[v] = (int) order.size();
        order.push_back(v);
        sz[v] = 1;
        min_depth[v] = depth[v];
        for (int id : g[v]) {
            if (id == pe[v]) {
                continue;
            }
            auto &e = edges[id];
            int to = e.from ^ e.to ^ v;
            if (was[to] == attempt) {
                min_depth[v] = min(min_depth[v], depth[to]);
                continue;
            }
```

```
            depth[to] = depth[v] + 1;
            dist[to] = dist[v] + e.cost;
            pv[to] = v;
            pe[to] = id;
            root[to] = (root[v] != -1 ? root[v] : to);
            do_dfs(to);
            sz[v] += sz[to];
            min_depth[v] = min(min_depth[v], min_depth[to]);
        }
        end[v] = (int) order.size() - 1;
    }

    void do_dfs_from(int v) {
        ++attempt;
        depth[v] = 0;
        dist[v] = T{};
        root[v] = v;
        pv[v] = pe[v] = -1;
        do_dfs(v);
    }

public:
    void dfs(int v, bool clear_order = true) {
        if (pv.empty()) {
            init();
        } else {
            if (clear_order) {
                order.clear();
            }
        }
        do_dfs_from(v);
    }

    void dfs_all() {
        init();
        for (int v = 0; v < n; v++) {
            if (depth[v] == -1) {
                do_dfs_from(v);
            }
        }
        assert((int) order.size() == n);
    }
};
```

3.7 Dijkstra

```
template <typename T>
vector <T> dijkstra(const graph<T> &g, int start) {
  assert(0 <= start && start < g.n);
  vector<T> dist(g.n, numeric_limits<T>::max());
  priority_queue<pair<T, int>, vector<pair<T, int> >,
     greater<pair<T, int> > > s;
  dist[start] = 0;
  s.emplace(dist[start], start);
  while (!s.empty()) {
    T expected = s.top().first;
    int i = s.top().second;
    s.pop();
    if (dist[i] != expected) {
      continue;
    }
    for (int id : g.g[i]) {
      auto &e = g.edges[id];
      int to = e.from ^ e.to ^ i;
      if (dist[i] + e.cost < dist[to]) {
        dist[to] = dist[i] + e.cost;
        s.emplace(dist[to], to);
      }
    }
  }
  return dist;
  // returns numeric_limits<T>::max() if there's no path
}
```

3.8 Dominator Tree

```
template <typename T>
vector<int> find_dominators(const digraph<T> &g, int root) {
    int n = g.n;
    vector<int> pos(n, -1);
```

```
vector<int> order;
vector<int> parent(n, -1);
function<void(int)> dfs = [&g, &pos, &order, &parent,
    &dfs](int v) {
    pos[v] = (int) order.size();
    order.push_back(v);
    for (int id : g.g[v]) {
        auto &e = g.edges[id];
        int u = e.to;
        if (pos[u] == -1) {
            parent[u] = v;
            dfs(u);
        }
    }
};
dfs(root);
vector<int> p(n), best(n);
iota(p.begin(), p.end(), 0);
iota(best.begin(), best.end(), 0);
vector<int> sdom = pos;
function<int(int)> find_best = [&p, &best, &sdom,
    &find_best](int x) {
    if (p[x] != x) {
        int u = find_best(p[x]);
        if (sdom[u] < sdom[best[x]]) {
            best[x] = u;
        }
        p[x] = p[p[x]];
    }
    if (sdom[best[p[x]]] < sdom[best[x]]) {
        best[x] = best[p[x]];
    }
    return best[x];
};
digraph<int> g_rev = g.reverse();
vector<int> idom(n, -1);
vector<int> link(n, 0);
vector< vector<int> > bucket(n);
for (int it = (int) order.size() - 1; it >= 0; it--) {
    int w = order[it];
    for (int id : g_rev.g[w]) {
        auto &e = g_rev.edges[id];
            int u = e.to;
            sdom[w] = min(sdom[w], sdom[find_best(u)]);
        }
        idom[w] = order[sdom[w]];
        for (int u : bucket[w]) {
            link[u] = find_best(u);
        }
        for (int id : g.g[w]) {
            auto &e = g.edges[id];
            int u = e.to;
            if (parent[u] == w) {
                p[u] = w;
            }
        }
        bucket[order[sdom[w]]].push_back(w);
    }
    for (int it = 1; it < (int) order.size(); it++) {
        int w = order[it];
        idom[w] = idom[link[w]];
    }
    return idom;
}
```

3.9 Eulerian Tour

```
template <typename T>
vector<int> find_eulerian_path(const graph<T> &g, int &root)
    {
    // in_deg and out_deg are fake for undigraph!
    vector<int> in_deg(g.n, 0);
    vector<int> out_deg(g.n, 0);
    int cnt_edges = 0;
    for (int id = 0; id < (int) g.edges.size(); id++) {
        cnt_edges++;
        auto &e = g.edges[id];
        out_deg[e.from]++;
        in_deg[e.to]++;
    }
    root = -1;
    int odd = 0;
```

```
for (int i = 0; i < g.n; i++) {
    if ((in_deg[i] + out_deg[i]) % 2 == 1) {
        odd++;
        if (root == -1 || out_deg[i] - in_deg[i] >
            out_deg[root] - in_deg[root]) {
            root = i;
        }
    }
}
if (odd > 2) {
    root = -1;
    return vector<int>();
}
if (root == -1) {
    root = 0;
    while (root < g.n && in_deg[root] + out_deg[root] ==
        0) {
        root++;
    }
    if (root == g.n) {
        // an empty path
        root = 0;
        return vector<int>();
    }
}
vector<bool> used(g.edges.size(), false);
vector<int> ptr(g.n, 0);
vector<int> balance(g.n, 0);
vector<int> res(cnt_edges);
int stack_ptr = 0;
int write_ptr = cnt_edges;
int v = root;
while (true) {
    bool found = false;
    while (ptr[v] < (int) g.g[v].size()) {
        int id = g.g[v][ptr[v]++];
        if (used[id]) {
            continue;
        }
        used[id] = true;
        res[stack_ptr++] = id;
        auto &e = g.edges[id];
        balance[v]++;
        v ^= e.from ^ e.to;
        balance[v]--;
        found = true;
        break;
    }
    if (!found) {
        if (stack_ptr == 0) {
            break;
        }
        int id = res[--stack_ptr];
        res[--write_ptr] = id;
        auto &e = g.edges[id];
        v ^= e.from ^ e.to;
    }
}
int disbalance = 0;
for (int i = 0; i < g.n; i++) {
    disbalance += abs(balance[i]);
}
if (write_ptr != 0 || disbalance > 2) {
    root = -1;
    return vector<int>();
}
return res;
// returns edge ids in the path (or the cycle if it
    exists)
// root == -1 if there is no path
// (or res.empty(), but this is also true when there are
    no edges)
}
```

3.10 Dinic

```
template <typename T>
class dinic {
public:
    flow_graph<T> &g;

    vector<int> ptr;
```

```
    vector<int> d;
    vector<int> q;

    dinic(flow_graph<T> &_g) : g(_g) {
        ptr.resize(g.n);
        d.resize(g.n);
        q.resize(g.n);
    }

    bool expath() {
        fill(d.begin(), d.end(), -1);
        q[0] = g.fin;
        d[g.fin] = 0;
        int beg = 0, end = 1;
        while (beg < end) {
            int i = q[beg++];
            for (int id : g.g[i]) {
                const auto &e = g.edges[id];
                const auto &back = g.edges[id ^ 1];
                if (back.c - back.f > g.eps && d[e.to] ==
                    -1) {
                    d[e.to] = d[i] + 1;
                    if (e.to == g.st) {
                        return true;
                    }
                    q[end++] = e.to;
                }
            }
        }
        return false;
    }

    T dfs(int v, T w) {
        if (v == g.fin) {
            return w;
        }
        int &j = ptr[v];
        while (j >= 0) {
            int id = g.g[v][j];
            const auto &e = g.edges[id];
            if (e.c - e.f > g.eps && d[e.to] == d[v] - 1) {
                T t = dfs(e.to, min(e.c - e.f, w));
                if (t > g.eps) {
                    g.edges[id].f += t;
                    g.edges[id ^ 1].f -= t;
                    return t;
                }
            }
            j--;
        }
        return 0;
    }

    T max_flow() {
        while (expath()) {
            for (int i = 0; i < g.n; i++) {
                ptr[i] = (int) g.g[i].size() - 1;
            }
            T big_add = 0;
            while (true) {
                T add = dfs(g.st, numeric_limits<T>::max());
                if (add <= g.eps) {
                    break;
                }
                big_add += add;
            }
            if (big_add <= g.eps) {
                break;
            }
            g.flow += big_add;
        }
        return g.flow;
    }

    vector<bool> min_cut() {
        max_flow();
        vector<bool> ret(g.n);
        for (int i = 0; i < g.n; i++) {
            ret[i] = (d[i] != -1);
        }
        return ret;
    }
};
```

3.11 Flow Graph

```
template <typename T>
class flow_graph {
public:
    static constexpr T eps = (T) 1e-9;

    struct edge {
        int from;
        int to;
        T c;
        T f;
    };

    vector<vector<int>> g;
    vector<edge> edges;
    int n;
    int st;
    int fin;
    T flow;

    flow_graph(int _n, int _st, int _fin) : n(_n), st(_st),
       fin(_fin) {
        assert(0 <= st && st < n && 0 <= fin && fin < n &&
           st != fin);
        g.resize(n);
        flow = 0;
    }

    void clear_flow() {
        for (const edge &e : edges) {
            e.f = 0;
        }
        flow = 0;
    }

    int add(int from, int to, T forward_cap, T backward_cap)
       {
        assert(0 <= from && from < n && 0 <= to && to < n);
        int id = (int) edges.size();
        g[from].push_back(id);
        edges.push_back({from, to, forward_cap, 0});
        g[to].push_back(id + 1);
        edges.push_back({to, from, backward_cap, 0});
        return id;
    }
};
```

3.12 Ford Fulkerson

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
using ld = long double;
const int maxn = 2e2 + 5;
int n;
ll c[maxn][maxn], f[maxn][maxn];
int used[maxn], p[maxn];
int timer;
bool bfs(int s, int t)
{
    queue<int>q;
    q.push(s);
    while(!q.empty())
    {
        int u = q.front();
        q.pop();
        if(u == t)
            return true;
        for(int i = 0; i < n; i++)
        {
            if(used[i] != timer && f[u][i] < c[u][i])
            {
                used[i] = timer;
                q.push(i);
                p[i] = u;
            }
        }
    }
    return false;
}
int32_t main()
```

```
{
    ios::sync_with_stdio(false);
    cin.tie(0);
    int tt;
    cin >> tt;
    while(tt--)
    {
        cin >> n;
        ll max_flow = 0;
        timer = 1;
        memset(c, 0, sizeof c);
        memset(f, 0, sizeof f);
        memset(used, 0, sizeof used);
        for(int i = 0; i < n - 1; i++)
        {
            int m;
            cin >> m;
            for(int j = 0; j < m; j++)
            {
                int u;
                cin >> u;
                u--;
                if(i == 0 || u == n - 1)
                    c[i][u] += 1;
                else
                    c[i][u] = 1e9;
            }
        }
        while(bfs(0, n - 1))
        {
            timer++;
            ll mn = numeric_limits<ll>::max();
            for(int i = n - 1; i; i = p[i])
            {
                mn = min(mn, c[p[i]][i] - f[p[i]][i]);
            }
            for(int i = n - 1; i; i = p[i])
            {
                f[p[i]][i] += mn;
                f[i][p[i]] -= mn;
            }
            max_flow += mn;
        }
        cout << max_flow << "\n";
    }
}
```

3.13 Hungarian

```
template<typename T>
class hungarian {
public:
    int n;
    int m;
    vector< vector<T> > a;
    vector<T> u;
    vector<T> v;
    vector<int> pa;
    vector<int> pb;
    vector<int> way;
    vector<T> minv;
    vector<bool> used;
    T inf;

    hungarian(int _n, int _m) : n(_n), m(_m) {
        assert(n <= m);
        a = vector< vector<T> >(n, vector<T>(m));
        u = vector<T>(n + 1);
        v = vector<T>(m + 1);
        pa = vector<int>(n + 1, -1);
        pb = vector<int>(m + 1, -1);
        way = vector<int>(m, -1);
        minv = vector<T>(m);
        used = vector<bool>(m + 1);
        inf = numeric_limits<T>::max();
    }
    inline void add_row(int i) {
        fill(minv.begin(), minv.end(), inf);
        fill(used.begin(), used.end(), false);
        pb[m] = i;
        pa[i] = m;
        int j0 = m;
```

```
    do {
        used[j0] = true;
        int i0 = pb[j0];
        T delta = inf;
        int j1 = -1;
        for (int j = 0; j < m; j++) {
            if (!used[j]) {
                T cur = a[i0][j] - u[i0] - v[j];
                if (cur < minv[j]) {
                    minv[j] = cur;
                    way[j] = j0;
                }
                if (minv[j] < delta) {
                    delta = minv[j];
                    j1 = j;
                }
            }
        }
        for (int j = 0; j <= m; j++) {
            if (used[j]) {
                u[pb[j]] += delta;
                v[j] -= delta;
            } else {
                minv[j] -= delta;
            }
        }
        j0 = j1;
    } while (pb[j0] != -1);
    do {
        int j1 = way[j0];
        pb[j0] = pb[j1];
        pa[pb[j0]] = j0;
        j0 = j1;
    } while (j0 != m);
}

inline T current_score() {
    return -v[m];
}

inline T solve() {
    for (int i = 0; i < n; i++) {
            add_row(i);
        }
        return current_score();
    }
};
```

3.14 Min Cost Max Flow

```
template <typename T, typename C>
class mcmf {
  public:
  static constexpr T eps = (T) 1e-9;

  struct edge {
    int from;
    int to;
    T c;
    T f;
    C cost;
  };

  vector< vector<int> > g;
  vector<edge> edges;
  vector<C> d;
  vector<int> q;
  vector<bool> in_queue;
  vector<int> pe;
  int n;
  int st, fin;
  // T bound;
  T flow;
  C cost;

  // Pass the upper_bound value if needed
  mcmf(int _n, int _st, int _fin) : n(_n), st(_st),
     fin(_fin) {
    assert(0 <= st && st < n && 0 <= fin && fin < n && st !=
       fin);
    g.resize(n);
    d.resize(n);
```

```
  in_queue.resize(n);
  pe.resize(n);
  flow = 0;
  cost = 0;
}

void clear_flow() {
  for (const edge &e : edges) {
    e.f = 0;
  }
  flow = 0;
}

void add(int from, int to, T forward_cap, T backward_cap,
   C cost) {
  assert(0 <= from && from < n && 0 <= to && to < n);
  g[from].push_back((int) edges.size());
  edges.push_back({from, to, forward_cap, 0, cost});
  g[to].push_back((int) edges.size());
  edges.push_back({to, from, backward_cap, 0, -cost});
}

bool expath() {
  fill(d.begin(), d.end(), numeric_limits<C>::max());
  q.clear();
  q.push_back(st);
  d[st] = 0;
  in_queue[st] = true;
  int beg = 0;
  bool found = false;
  while (beg < (int) q.size()) {
    int i = q[beg++];
    if (i == fin) {
      found = true;
    }
    in_queue[i] = false;
    for (int id : g[i]) {
      const edge &e = edges[id];
      if (e.c - e.f > eps && d[i] + e.cost < d[e.to]) {
        d[e.to] = d[i] + e.cost;
        pe[e.to] = id;
        if (!in_queue[e.to]) {
          q.push_back(e.to);
          in_queue[e.to] = true;
        }
      }
    }
  }
  if (found) {
    //  T push = bound - flow;
    T push = numeric_limits<T>::max();
    int v = fin;
    while (v != st) {
      const edge &e = edges[pe[v]];
      push = min(push, e.c - e.f);
      v = e.from;
    }
    v = fin;
    while (v != st) {
      edge &e = edges[pe[v]];
      e.f += push;
      edge &back = edges[pe[v] ^ 1];
      back.f -= push;
      v = e.from;
    }
    flow += push;
    cost += push * d[fin];
  }
  return found;
}

pair<T, C> max_flow_min_cost() {
  // while (flow < bound && expath())
  while (expath()) {
  }
  return make_pair(flow, cost);
}
};
```

3.15 Floyd Warshall

```
int g[n][n];
```

```
for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++) {
        cin >> g[i][j];
    }
for (int k = 0; k < n; k--) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (g[i][k] < inf && g[k][j] < inf) {
                if (g[i][j] > g[i][k] + g[k][j])
                    g[i][j] = g[i][k] + g[k][j];
            }
        }
    }
}
```

3.16 Heavy Light Decomposition Forest

```
template <typename T>
class hld_forest : public lca_forest<T> {
public:
    using lca_forest<T>::edges;
    using lca_forest<T>::g;
    using lca_forest<T>::n;
    using lca_forest<T>::pv;
    using lca_forest<T>::sz;
    using lca_forest<T>::pos;
    using lca_forest<T>::order;
    using lca_forest<T>::depth;
    using lca_forest<T>::dfs;
    using lca_forest<T>::dfs_all;
    using lca_forest<T>::lca;
    using lca_forest<T>::build_lca;

    vector<int> head;
    vector<int> visited;

    hld_forest(int _n) : lca_forest<T>(_n) {
        visited.resize(n);
    }
    void build_hld(const vector<int> &vs) {
    for (int tries = 0; tries < 2; tries++) {
        if (vs.empty()) {
            dfs_all();
        } else {
            order.clear();
            for (int v : vs) {
                dfs(v, false);
            }
            assert((int) order.size() == n);
        }
        if (tries == 1) {
            break;
        }
        for (int i = 0; i < n; i++) {
            if (g[i].empty()) {
                continue;
            }
            int best = -1, bid = 0;
            for (int j = 0; j < (int) g[i].size(); j++) {
                int id = g[i][j];
                int v = edges[id].from ^ edges[id].to ^
                    i;
                if (pv[v] != i) {
                    continue;
                }
                if (sz[v] > best) {
                    best = sz[v];
                    bid = j;
                }
            }
            swap(g[i][0], g[i][bid]);
        }
    }
    build_lca();
    head.resize(n);
    for (int i = 0; i < n; i++) {
        head[i] = i;
    }
    for (int i = 0; i < n - 1; i++) {
        int x = order[i];
        int y = order[i + 1];
        if (pv[y] == x) {
```

```
            head[y] = head[x];
        }
    }
}

void build_hld(int v) {
    build_hld(vector<int>(1, v));
}

void build_hld_all() {
    build_hld(vector<int>());
}

bool apply_on_path(int x, int y, bool with_lca,
   function<void(int, int, bool)> f) {
    // f(x, y, up): up -- whether this part of the path
       goes up
    assert(!head.empty());
    int z = lca(x, y);
    if (z == -1) {
        return false;
    }
    {
        int v = x;
        while (v != z) {
            if (depth[head[v]] <= depth[z]) {
                f(pos[z] + 1, pos[v], true);
                break;
            }
            f(pos[head[v]], pos[v], true);
            v = pv[head[v]];
        }
    }
    if (with_lca) {
        f(pos[z], pos[z], false);
    }
    {
        int v = y;
        int cnt_visited = 0;
        while (v != z) {
            if (depth[head[v]] <= depth[z]) {
                f(pos[z] + 1, pos[v], false);
                break;
            }
            visited[cnt_visited++] = v;
            v = pv[head[v]];
        }
        for (int at = cnt_visited - 1; at >= 0; at--) {
            v = visited[at];
            f(pos[head[v]], pos[v], false);
        }
    }
    return true;
}
};
int main() {
    int n;
    cin >> n;
    hld_forest<int>g(n);
    for (int i = 2; i <= n; i++) {
        int x, y;
        cin >> x >> y;
        --x, --y;
        g.add(x, y);
    }
    g.dfs(0);
    g.build_hld(0);
    int q;
    cin >> q;
    segtree st(n);
    for (int i = 0; i < q; ++i) {
        char t;
        cin >> t;
        if (t == 'I') {
            int u, v;
            cin >> u >> v;
            u--;
            g.apply_on_path(u, u, true, [&](int from, int
               to, bool) {
                st.modify(from, to, v);
            });
        }
        else {
            int u, v;
```

```
            cin >> u >> v;
            --u, --v;
            long long ans = 0;
            g.apply_on_path(u, v, true, [&](int from, int
                to, bool) {
                ans = max(ans, st.get(from, to).mx);
            });
            cout << ans << '\n';
        }
    }
}
```

3.17 Kosaraju

```
// Strongly Connected Components
void dfs(int u)
{
    vis[u] = true;
    for (auto v : g[u])
        if (!vis[v])
            dfs(v);
    order.push_back(u);
}
void scc(int u, int c)
{
    vis[u] = 1;
    comp[u] = c;
    for (auto v : rg[u])
        if (!vis[v])
            scc(v, c);
}
void kosaraju()
{
    for (int i = 0; i < n; i++)
        if (!vis[i])
            dfs(i);
    reverse(order.begin(), order.end());
    memset(vis, 0, sizeof vis);
    for (int i = 0; i < n; i++)
    {
        int u = order[i];
        if (!vis[u])
            scc(u, u);
    }
}
```

3.18 Least Common Ancestor Forest

```
template <typename T>
class lca_forest : public dfs_forest<T> {
public:
    using dfs_forest<T>::edges;
    using dfs_forest<T>::g;
    using dfs_forest<T>::n;
    using dfs_forest<T>::pv;
    using dfs_forest<T>::pos;
    using dfs_forest<T>::end;
    using dfs_forest<T>::depth;

    int h;
    vector<vector<int>> pr;

    lca_forest(int _n) : dfs_forest<T>(_n) {
    }

    inline void build_lca() {
        assert(!pv.empty());
        int max_depth = 0;
        for (int i = 0; i < n; i++) {
            max_depth = max(max_depth, depth[i]);
        }
        h = 1;
        while ((1 << h) <= max_depth) {
            h++;
        }
        pr.resize(n);
        for (int i = 0; i < n; i++) {
            pr[i].resize(h);
            pr[i][0] = pv[i];
        }
```

```
    for (int j = 1; j < h; j++) {
        for (int i = 0; i < n; i++) {
            pr[i][j] = (pr[i][j - 1] == -1 ? -1 :
                pr[pr[i][j - 1]][j - 1]);
        }
    }
}

inline bool anc(int x, int y) {
    return (pos[x] <= pos[y] && end[y] <= end[x]);
}

inline int go_up(int x, int up) {
    assert(!pr.empty());
    up = min(up, (1 << h) - 1);
    for (int j = h - 1; j >= 0; j--) {
        if (up & (1 << j)) {
            x = pr[x][j];
            if (x == -1) {
                break;
            }
        }
    }
    return x;
}

inline int lca(int x, int y) {
    assert(!pr.empty());
    if (anc(x, y)) {
        return x;
    }
    if (anc(y, x)) {
        return y;
    }
    if (depth[x] > depth[y])
        swap(x, y);
    for (int j = h - 1; j >= 0; j--) {
        if (pr[x][j] != -1 && !anc(pr[x][j], y)) {
            x = pr[x][j];
        }
    }
    return pr[x][0];
```

```
    }
};
```

3.19 Edmond's Blossom

```
const int MAX = 5000;
vector<int> g[MAX];
int match[MAX];
int n, parent[MAX], base[MAX], vis[MAX];
queue<int> q;
void contract(int u, int v, bool first = 1) {
    static vector<bool> blossom;
    static int l;
    if (first) {
        blossom = vector<bool>(n, 0);
        vector<bool> teve(n, 0);
        int k = u; l = v;
        while (1) {
            teve[k = base[k]] = 1;
            if (match[k] == -1) break;
            k = parent[match[k]];
        }
        while (!teve[l = base[l]]) l = parent[match[l]];
    }
    while (base[u] != l) {
        blossom[base[u]] = blossom[base[match[u]]] = 1;
        parent[u] = v;
        v = match[u];
        u = parent[match[u]];
    }
    if (!first) return;
    contract(v, u, 0);
    for (int i = 0; i < n; i++) if (blossom[base[i]]) {
            base[i] = l;
            if (!vis[i]) q.push(i);
            vis[i] = 1;
        }
}

int getpath(int s) {
```

```
    for (int i = 0; i < n; i++) base[i] = i, parent[i] = -1,
       vis[i] = 0;
    vis[s] = 1; q = queue<int>(); q.push(s);
    while (q.size()) {
        int u = q.front(); q.pop();
        for (int v : g[u]) {
            if (base[v] == base[u] or match[u] == v)
               continue;
            if (v == s or (match[v] != -1 and
               parent[match[v]] != -1))
                contract(u, v);
            else if (parent[v] == -1) {
                parent[v] = u;
                if (match[v] == -1) return v;
                v = match[v];
                vis[v] = 1; q.push(v);
            }
        }
    }
    return -1;
}

int blossom() {
    int ans = 0;
    memset(match, -1, sizeof(match));
    for (int i = 0; i < n; i++) if (match[i] == -1)
            for (int j : g[i]) if (match[j] == -1) {
                    match[i] = j;
                    match[j] = i;
                    ans++;
                    break;
                }
    for (int i = 0; i < n; i++) if (match[i] == -1) {
            int j = getpath(i);
            if (j == -1) continue;
            ans++;
            while (j != -1) {
                int p = parent[j], pp = match[p];
                match[p] = j;
                match[j] = p;
                j = pp;
            }
        }
    return ans;
}
```

3.20 Hopcroft Karp

```
constexpr int maxn = 1e5 + 5;
constexpr int inf = 1e9;
vector<int>g[maxn];
int l[maxn], r[maxn], ldist[maxn], rdist[maxn];
struct hopcroft
{
    int _n;
    hopcroft(int n): _n(n)
    {
        for(int i = 0; i < _n; i++)
        {
            g[i].clear();
            l[i] = r[i] = -1;
        }
    }
    void add(int u, int v)
    {
        assert(u >= 0 && u < _n && v >= 0 && v < _n);
        g[u].push_back(v);
    }
    bool bfs()
    {
        queue<int>q;
        bool found = false;
        for(int i = 0; i < _n; i++)
        {
            ldist[i] = rdist[i] = 0;
            if(l[i] == -1)
                q.push(i);
        }
        while(!q.empty())
        {
            int u = q.front();
            q.pop();
```

```
        for(int v : g[u])
        {
            if(rdist[v] == 0)
            {
                rdist[v] = ldist[u] + 1;
                if(r[v] == -1)
                    found = true;
                else
                    ldist[r[v]] = rdist[v] + 1,
                        q.push(r[v]);
            }
        }
    }
    return found;
}
bool dfs(int u)
{
    for(int v : g[u])
    {
        if(rdist[v] == ldist[u] + 1)
        {
            rdist[v] = 0;
            if(r[v] == -1 || dfs(r[v]))
            {
                l[u] = v;
                r[v] = u;
                return true;
            }
        }
    }
    return false;
}
int solve()
{
    int res = 0;
    while(bfs())
    {
        for(int i = 0; i < _n; i++)
            if(l[i] == -1)
                if(dfs(i))
                    res++;
    }
        return res;
    }
};
int32_t main()
{
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int n, m, p;
    cin >> n >> m >> p;
    hopcroft bpm(n + m);
    for(int i = 0; i < p; i++)
    {
        int u, v;
        cin >> u >> v;
        u--;
        v--;
        v += n;
        bpm.add(u, v);
    }
    cout << bpm.solve();
}
```

3.21 Kruskal's Algorithm

```
// O(M*log(N))
template <typename T>
vector<int> find_mst(const undigraph<T> &g, T &ans) {
    vector<int> order(g.edges.size());
    iota(order.begin(), order.end(), 0);
    sort(order.begin(), order.end(), [&g](int a, int b) {
        return g.edges[a].cost < g.edges[b].cost;
    });
    DSU d(g.n);
    vector<int> ans_list;
    ans = 0;
    for (int id : order) {
        auto &e = g.edges[id];
        if (d.find(e.from) != d.find(e.to)) {
            d.merge(e.from, e.to);
            ans_list.push_back(id);
```

```
            ans += e.cost;
        }
    }
    return ans_list;
}
```

3.22 Topological Sort

```
template <typename T>vector<int> find_topsort(const
    digraph<T> &g) {
    vector<int> deg(g.n, 0);
    for (int id = 0; id < (int) g.edges.size(); id++) {
        deg[g.edges[id].to]++;
    }
    vector<int> x;
    for (int i = 0; i < g.n; i++) {
        if (deg[i] == 0) {
            x.push_back(i);
        }
    }
    for (int ptr = 0; ptr < (int) x.size(); ptr++) {
        int i = x[ptr]; for (int id : g.g[i]) {
            auto &e = g.edges[id];
            int to = e.to;
            if (--deg[to] == 0) {
                x.push_back(to);
            }
        }
    }
    if ((int) x.size() != g.n) {
        return vector<int>();
    }
    return x;
}
```

4 Mathematics

4.1 2-Satisfiability

```
const int maxn = 1e5 + 10;
vector<int>g[maxn];
vector<int>order;
int vis[maxn], comp[maxn];
vector<bool>assignment;

// Both functions accepts 1-based index
inline int positive(int x)
{
    //As it's 2*x for zero based(unchanged)
    return (2 * (x - 1));
}
inline int negative(int x)
{
    //As it's 2*x+1 for zero based(changed)
    return (2 * (x - 1) + 1);
}

void add_edge(int u, int v)
{
    g[u].emplace_back(v);
    g[v].emplace_back(u);
}

void dfs(int u)
{
    vis[u] = 1;
    for (auto it : g[u])
    {
        if (!vis[it])
            dfs(it);
    }
    order.emplace_back(u);
}

void dfs2(int u, int c)
{
```

```
    comp[u] = c;
    for (auto it : g[u])
    {
        if (comp[it] == -1)
            dfs2(it, c);
    }
}
int main() {
    memset(vis, 0, sizeof vis);
    for (int i = 0; i < n; i++) {
        if (!vis[i]) dfs(i);
    }
    memset(comp, -1, sizeof comp);
    for (int i = n - 1, j = 0; i >= 0; i--) {
        int v = order[i];
        if (comp[v] == -1) dfs2(v, j++);
    }
    bool flag = true;
    vector<int>ans;
    for (int i = 0; i < n; i += 2) {
        if (comp[i] == comp[i | 1]) {
            flag = false;
            break;
        }
        if (comp[i] > comp[i | 1]) {
            ans.emplace_back(i);
        }
    }
    if (flag) {
        cout << ans.size() << "\n";
        for (auto it : ans) {
            cout << (it) / 2 + 1 << " ";
        }
    }
    else
    {
        cout << "Impossible\n";
    }
    return 0;
}
```

4.2 Arbitrary Precision Arithmetic

```
// Process big numbers which cannot fit in C++ data types
// Base is decimal (10 or power of 10). Usually it is
   int(1e9)
// There are no leading zeros in the result as long as there
   are no leading zeros in operands.
// Zero is represented as empty vector or single digit 0.
// Number is stored from LSB to MSB in a vector.

const int base = int(1e9);
const int base_digits = int(9);
struct bignum {
    vector<int> A;
    int sign;

    bignum(): sign(1) {}
    bignum(int64_t X) {
        *this = X;
    }
    bignum(const bignum& other): sign(other.sign),
       A(other.A) {}
    bignum(const string& S) {
        read(S);
    }
    void operator=(const bignum& other) {
        sign = other.sign;
        A = other.A;
    }
    void operator=(int64_t X) {
        sign = 1;
        if (X < 0)
            sign = -1, X = -X;
        for (; X > 0; X /= base) {
            A.emplace_back(X % base);
        }
    }
    void trim() {
        while (!A.empty() && A.back() == 0)
            A.pop_back();
        if (A.empty())
            sign = 1;
```

```
}
void read(const string& S) {
    sign = 1;
    A.clear();
    int pos = 0;
    while (pos < int(S.size()) && (S[pos] == '-' ||
       S[pos] == '+')) {
        if (S[pos] == '-') sign = -sign;
        ++pos;
    }
    for (int i = int(S.size()) - 1; i >= pos; i -=
       base_digits) {
        int X = 0;
        for (int j = max(pos, i - base_digits + 1); j <=
           i; j++) {
            assert(isdigit(S[j]));
            X = 10 * X + (S[j] - '0');
        }
        A.emplace_back(X);
    }
    trim();
}
bignum operator-() const {
    bignum res = *this;
    res.sign = -sign;
    return res;
}

bignum abs() const {
    bignum res = *this;
    res.sign *= res.sign;
    return res;
}

bool is_zero() const {
    return A.empty() || (int(A.size()) == 1 && !A[0]);
}
bool operator<(const bignum &other) const {
    if (sign != other.sign)
        return (sign < other.sign);
    if (int(A.size()) != int(other.A.size()))
        return (int(A.size()) * sign <
           int(other.A.size()) * other.sign);
    for (int i = int(A.size()) - 1; i >= 0; i--)
        if (A[i] != other.A[i])
            return (A[i] * sign < other.A[i] * sign);
    return false;
}

bool operator>(const bignum &other) const {
    return (other < *this);
}
bool operator<=(const bignum &other) const {
    return !(other < *this);
}
bool operator>=(const bignum &other) const {
    return !(*this < other);
}
bool operator==(const bignum &other) const {
    return !(*this < other) && !(other < *this);
}
bool operator!=(const bignum &other) const {
    return (*this < other || other < *this);
}
bignum operator+(const bignum &other) const {
    if (sign == other.sign) {
        bignum res = other;
        int N = max(int(A.size()), int(other.A.size()));
        for (int i = 0, carry = 0; (i < N) || (carry !=
           0); ++i) {
            if (i == int(res.A.size())) {
                res.A.emplace_back(0);
            }
            res.A[i] += carry + (i < int(A.size()) ?
               A[i] : 0);
            carry = (res.A[i] >= base);
            if (carry != 0) {
                res.A[i] -= base;
            }
        }
        return res;
    }
    return *this - (-other);
}
```

```
bignum operator-(const bignum &other) const {
    if (sign == other.sign) {
        if (abs() >= other.abs()) {
            bignum res = *this;
            for (int i = 0, carry = 0; (i <
               int(other.A.size())) || (carry != 0);
               ++i) {
                res.A[i] -= carry + (i <
                   int(other.A.size()) ? other.A[i] : 0);
                carry = (res.A[i] < 0);
                if (carry != 0)
                    res.A[i] += base;
            }
            res.trim();
            return res;
        }
        return -(other - *this);
    }
    return *this + (-other);
}

void operator*=(int other) {
    if (other < 0)
        sign = -sign, other = -other;
    for (int i = 0, carry = 0; i < int(A.size()) ||
       (carry != 0); ++i) {
        if (i == int(A.size()))
            A.push_back(0);
        int64_t cur = A[i] * 1ll * other + carry;
        carry = (int) (cur / base);
        A[i] = (int) (cur % base);
    }
    trim();
}
friend pair<bignum, bignum> divmod(const bignum &a1,
   const bignum &b1) {
    int norm = base / (b1.A.back() + 1);
    bignum a = a1.abs() * norm;
    bignum b = b1.abs() * norm;
    bignum q, r;
    q.A.resize(a.A.size());
    for (int i = a.A.size() - 1; i >= 0; i--) {
        r *= base;
        r += a.A[i];
        int s1 = r.A.size() <= b.A.size() ? 0 :
           r.A[b.A.size()];
        int s2 = r.A.size() <= b.A.size() - 1 ? 0 :
           r.A[b.A.size() - 1];
        int d = ((int64_t) base * s1 + s2) / b.A.back();
        r -= b * d;
        while (r < 0)
            r += b, --d;
        q.A[i] = d;
    }

    q.sign = a1.sign * b1.sign;
    r.sign = a1.sign;
    q.trim();
    r.trim();
    return make_pair(q, r / norm);
}
static vector<int> convert_base(const vector<int> &V,
   int old_digits, int new_digits) {
    vector<int64_t> p(max(old_digits, new_digits) + 1);
    p[0] = 1;
    for (int i = 1; i < (int) p.size(); i++)
        p[i] = p[i - 1] * 10;
    vector<int> res;
    int64_t cur = 0;
    int cur_digits = 0;
    for (int i = 0; i < (int) V.size(); i++) {
        cur += V[i] * p[cur_digits];
        cur_digits += old_digits;
        while (cur_digits >= new_digits) {
            res.push_back(int(cur % p[new_digits]));
            cur /= p[new_digits];
            cur_digits -= new_digits;
        }
    }
    res.push_back((int) cur);
    while (!res.empty() && !res.back())
        res.pop_back();
```

```
    return res;
}

static vector<int64_t> karatsuba_multiply(const
    vector<int64_t> &V, const vector<int64_t> &b) {
    int n = V.size();
    vector<int64_t> res(n + n);
    if (n <= 32) {
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                res[i + j] += V[i] * b[j];
        return res;
    }

    int k = n >> 1;
    vector<int64_t> a1(V.begin(), V.begin() + k);
    vector<int64_t> a2(V.begin() + k, V.end());
    vector<int64_t> b1(b.begin(), b.begin() + k);
    vector<int64_t> b2(b.begin() + k, b.end());

    vector<int64_t> a1b1 = karatsuba_multiply(a1, b1);
    vector<int64_t> a2b2 = karatsuba_multiply(a2, b2);

    for (int i = 0; i < k; i++)
        a2[i] += a1[i];
    for (int i = 0; i < k; i++)
        b2[i] += b1[i];

    vector<int64_t> r = karatsuba_multiply(a2, b2);
    for (int i = 0; i < (int) a1b1.size(); i++)
        r[i] -= a1b1[i];
    for (int i = 0; i < (int) a2b2.size(); i++)
        r[i] -= a2b2[i];

    for (int i = 0; i < (int) r.size(); i++)
        res[i + k] += r[i];
    for (int i = 0; i < (int) a1b1.size(); i++)
        res[i] += a1b1[i];
    for (int i = 0; i < (int) a2b2.size(); i++)
        res[i + n] += a2b2[i];
    return res;
}
```

```
bignum operator*(const bignum &v) const {
    vector<int> a6 = convert_base(this->A, base_digits,
       6);
    vector<int> b6 = convert_base(v.A, base_digits, 6);
    vector<int64_t> V(a6.begin(), a6.end());
    vector<int64_t> b(b6.begin(), b6.end());
    while (V.size() < b.size())
        V.push_back(0);
    while (b.size() < V.size())
        b.push_back(0);
    while (V.size() & (V.size() - 1))
        V.push_back(0), b.push_back(0);
    vector<int64_t> c = karatsuba_multiply(V, b);
    bignum res;
    res.sign = sign * v.sign;
    for (int i = 0, carry = 0; i < (int) c.size(); i++) {
        int64_t cur = c[i] + carry;
        res.A.push_back((int) (cur % 1000000));
        carry = (int) (cur / 1000000);
    }
    res.A = convert_base(res.A, 6, base_digits);
    res.trim();
    return res;
}
int64_t long_value() const {
    int64_t res = 0;
    for (int i = A.size() - 1; i >= 0; i--)
        res = res * base + A[i];
    return res * sign;
}

friend bignum gcd(const bignum &a, const bignum &b) {
    return b.is_zero() ? a : gcd(b, a % b);
}
friend bignum lcm(const bignum &a, const bignum &b) {
    return a / gcd(a, b) * b;
}
bignum operator/(const bignum &other) const {
    return divmod(*this, other).first;
}
```

```
bignum operator%(const bignum &other) const {
    return divmod(*this, other).second;
}

void operator/=(int other) {
    if (other < 0)
        sign = -sign, other = -other;
    for (int i = (int) A.size() - 1, rem = 0; i >= 0;
       --i) {
        int64_t cur = A[i] + rem * (int64_t) base;
        A[i] = (int) (cur / other);
        rem = (int) (cur % other);
    }
    trim();
}

bignum operator/(int other) const {
    bignum res = *this;
    res /= other;
    return res;
}

int operator%(int other) const {
    if (other < 0)
        other = -other;
    int m = 0;
    for (int i = A.size() - 1; i >= 0; --i)
        m = (A[i] + m * (int64_t) base) % other;
    return m * sign;
}
void operator+=(const bignum &other) {
    *this = *this + other;
}
void operator-=(const bignum &other) {
    *this = *this - other;
}
bignum operator*(int other) const {
    bignum res = *this;
    res *= other;
    return res;
}
void operator*=(const bignum &other) {
    *this = *this * other;
}
void operator/=(const bignum &other) {
    *this = *this / other;
}
friend istream& operator>>(istream &stream, bignum
   &other) {
    string s;
    stream >> s;
    other.read(s);
    return stream;
}

friend ostream& operator<<(ostream &stream, const bignum
   &other) {
    if (other.sign == -1)
        stream << '-';
    stream << (other.A.empty() ? 0 : other.A.back());
    for (int i = int(other.A.size()) - 2; i >= 0; --i)
        stream << setw(base_digits) << setfill('0') <<
            other.A[i];
    return stream;
}
};
```

4.3 Combination

```
vector<mint> fact(1, 1);
vector<mint> inv_fact(1, 1);

mint C(int n, int k) {
    if (k < 0 || k > n) {
        return 0;
    }
    while ((int) fact.size() < n + 1) {
        fact.push_back(fact.back() * (int) fact.size());
        inv_fact.push_back(1 / fact.back());
    }
    return fact[n] * inv_fact[k] * inv_fact[n - k];
}
```

4.4 Digit DP

```
long long dp[19][180][2];
int num[19];
int sz;
long long solve(int idx, int taken, bool not_tight) {
    if (idx == sz) {
        return 1;
    }
    long long &ans = dp[idx][taken][not_tight];
    if (ans != -1) {
        return ans;
    }
    ans = 0;
    int lo = 0, hi = 9;
    if (!not_tight)
        hi = num[idx];
    for (int i = lo; i <= hi; ++i)
        ans += solve(idx + 1, taken + i, (not_tight | i <
            num[idx]));
    return ans;
}
long long f(long long x) {
    sz = 0;
    memset(dp, -1, sizeof dp);
    while (x) {
        num[sz++] = x % 10;
        x /= 10;
    }
    // Initially number is tight
    reverse(num, num + sz);
    return solve(0, 0, 0);
}
```

4.5 Discrete Logarithm

```
// Solves A^{X} mod M = B mod M in X
// Solution does not always exists
template<typename T>
class discrete_logarithm {
private:
    T A, B, M;
public:
    discrete_logarithm(T A_, T B_, T M_): A(A_), B(B_),
        M(M_) {}

    // Time: O(sqrt(M))
    T solve() {
        A %= M;
        B %= M;
        T K = 1, add{};
        T g = __gcd(A, M);
        while (g > 1) {
            if (B == K) return add;
            if (B % g != 0) return -1;
            B /= g, M /= g;
            add++;
            K = (K * 1ll * A / g) % M;
            g = __gcd(A, M);
        }
        T N = sqrt(M) + 1;
        T an = 1;
        for (T i = 0; i < N; ++i)
            an = (an * 1ll * A) % M;

        unordered_map<T, T> vals;
        vals.reserve(N + 1);
        for (T q = 0, cur = B; q <= N; ++q) {
            vals[cur] = q;
            cur = (cur * 1ll * A) % M;
        }

        for (T p = 1, cur = K; p <= N; ++p) {
            cur = (cur * 1ll * an) % M;
            if (vals.count(cur)) {
                T ans = N * p - vals[cur] + add;
                return ans;
            }
        }
        return -1;
    }
};
```

4.6 Discrete Root

```
// Solves X^{K} mod N = A mod N for prime N, A, K in X
template<typename T>
class discrete_root {
private:
    T N, A, K;
public:
    discrete_root(T K_, T A_, T N_) : K(K_), A(A_), N(N_)  {}
    vector<T>solve() {
        if (A == 0) {
            return {0};
        }
        primitive_root<T>PR(N);
        auto primitive_root_solution = PR.solve(1);
        assert(int(primitive_root_solution.size()) == 1);
        T g = primitive_root_solution[0];
        modular<T>M(N);
        discrete_logarithm<T>DL(M.power(g, K), A, N);
        T ans = DL.solve();
        if (ans == -1) {
            return vector<T>();
        }
        vector<T>solutions;
        T delta = (N - 1) / __gcd(K, N - 1);
        for (T cur = ans % delta; cur < N - 1; cur += delta)
            {
            solutions.emplace_back(M.power(g, cur));
        }
        return solutions;
    }
};
```

4.7 Factorizer

```
template<typename T>
struct factorizer {
private:
    int precalculated = 1;
public:
    vector<T> least = {0, 1};
    vector<T> primes; // Primes can be greater than int in
        segmented_sieve
    vector<bool>is_prime;
    bool trial_division(const T &N) {
        for (T x = 2; x * x <= N; x++) {
            if (N % x == 0) return false;
        }
        return true;
    }
    template<typename U>
    bool miller_rabin(const U &N, const vector<U>& bases) {
        if (N < 2) {
            return false;
        }
        vector<U> small_primes = {2, 3, 5, 7, 11, 13, 17,
            19, 23, 29};
        for (const U& x : small_primes) {
            if (N % x == 0) {
                return N == x;
            }
        }
        if (N < 31 * 31) {
            return true;
        }
        int s = 0;
        U d = N - 1;
        while ((d & 1) == 0) {
            d >>= 1;
            s++;
        }
        // N - 1 = 2^{s} * d
        for (const U& a : bases) {
            if (a % N == 0) {
                continue;
            }
            modular<U>M(N);
            U cur = M.power(a, d);
            if (cur == 1) {
                continue;
            }
            bool witness = true;
```

```
        for (int r = 0; r < s; r++) {
            if (cur == N - 1) {
                witness = false;
                break;
            }
            cur = M.mul(cur, cur);
        }
        if (witness) {
            return false;
        }
    }
    return true;
}

bool miller_rabin(const int32_t &N) {
    return miller_rabin(N, {2, 7, 61});
}

bool miller_rabin(const int64_t &N) {
    return miller_rabin(N, {2, 325, 9375, 28178, 450775,
       9780504, 1795265022});
}
// Only if you really need uint64_t version
/* bool miller_rabin(const uint64_t &N) {
    if (N < 2) {
        return false;
    }
    vector<uint32_t> small_primes = {2, 3, 5, 7, 11, 13,
       17, 19, 23, 29};
    for (uint32_t x : small_primes) {
        if (N == x) {
            return true;
        }
        if (N % x == 0) {
            return false;
        }
    }
    if (N < 31 * 31) {
        return true;
    }
    uint32_t s = __builtin_ctzll(N - 1);
    uint64_t d = (N - 1) >> s;
    function<bool(uint64_t)> witness = [&N, &s,
       &d](uint64_t a) {
        uint64_t cur = 1, p = d;
        while (p > 0) {
            if (p & 1) {
                cur = (__uint128_t) cur * a % N;
            }
            a = (__uint128_t) a * a % N;
            p >>= 1;
        }
        if (cur == 1) {
            return false;
        }
        for (uint32_t r = 0; r < s; r++) {
            if (cur == N - 1) {
                return false;
            }
            cur = (__uint128_t) cur * cur % N;
        }
        return true;
    };
    vector<uint64_t> bases_64bit = {2, 325, 9375, 28178,
       450775, 9780504, 1795265022};
    for (uint64_t a : bases_64bit) {
        if (a % N == 0) {
            return true;
        }
        if (witness(a)) {
            return false;
        }
    }
    return true;
}*/

// Time: O(N*ln(ln( N ))) + O(N)
// Memory: O(N)
void slow_sieve(const T &N) {
    least.assign(N + 1, 0);
    for (T i = 2; i * i <= N; i++) {
        if (least[i] == 0) {
            for (T j = i * i; j <= N; j += i) {
                if (least[j] == 0) {
```

```
                    least[j] = i;
                }
            }
        }
    }
    primes.clear();
    for (T i = 2; i <= N; i++) {
        if (least[i] == 0) {
            least[i] = i;
            primes.push_back(i);
        }
    }
    precalculated = N;
}
// Time: O(N)
// Memory: O(N)
void linear_sieve(const T &N) {
    least.assign(N + 1, 0);
    is_prime.assign(N + 1, 0);
    primes.clear();
    for (T i = 2; i <= N; i++) {
        if (least[i] == 0) {
            least[i] = i;
            is_prime[i] = 1;
            primes.push_back(i);
        }
        for (T x : primes) {
            if (x > least[i] || i * x > N) {
                break;
            }
            least[i * x] = x;
        }
    }
    precalculated = N;
}
// Time: O((R     L + 1).log(log(R)) +
   ( R )log(log( R )))
// Space: O(max(R - L + 1,   R ))
// NOTE: Does not update least vector
void segmented_sieve(T l, T r) {
    // assert(l >= 1 && (r - l + 1) <= (T)1e7);
    if (r < l) {
        return;
    }
    T sqrt_r = sqrt(r);
    vector<int>mark(sqrt_r + 1, 0);
    primes.clear();
    // Primes till sqrt_r
    for (T i = 2; i <= sqrt_r; ++i) {
        if (mark[i] == 0) {
            primes.push_back(i);
            for (T j = i * i; j <= sqrt_r; j += i) {
                mark[j] = i;
            }
        }
    }
    vector<int>vis(r - l + 1, 0);
    for (T p : primes) {
        for (T i = max(p * p, (l + p - 1) / p * p); i <=
           r; i += p) {
            assert(i - l >= 0 && i - l <= r - l);
            if (vis[i - l] == 0)
                vis[i - l] = p;
        }
    }
    if (l == 1) {
        vis[0] = 1;
    }
    primes.clear();
    for (T i = max(l, (T)2); i <= r; ++i) {
        if (vis[i - l] == 0) {
            primes.push_back(i);
        }
    }
}
vector<pair<T, int>> merge_factors(const vector<pair<T,
   int>>&A, const vector<pair<T, int>>&B) {
    vector<pair<T, int>>C;
    int i = 0;
    int j = 0;
    while (i < int(A.size()) || j < int(B.size())) {
        if (i < int(A.size()) && j < int(B.size()) &&
           A[i].first == B[j].first) {
            C.emplace_back(A[i].first, A[i].second +
```

```
                B[j].second);
            ++i;
            ++j;
            continue;
        }
        if (j == int(B.size()) || (i < int(A.size()) &&
            A[i].first < B[j].first)) {
            C.push_back(A[i++]);
        } else {
            C.push_back(B[j++]);
        }
    }
    return C;
}
vector<pair<T, int>> pollard_rho(const T &N, const T&c) {
    if (N <= 1) {
        return {};
    }
    if ((N & 1) == 0) {
        return merge_factors({{2, 1}}, pollard_rho(N /
            2, c));
    }
    if (miller_rabin(N)) {
        return {{N, 1}};
    }

    // Brent's cycle detection
    T power = 1;
    T lambda = 1;
    modular<T>M(N);
    T x = 2;
    T saved = 2;
    while (true) {
        x = M.add(M.mul(x, x), c);
        T g = __gcd(M.sub(x, saved), N);
        if (g != 1) {
            return merge_factors(pollard_rho(g, c + 1),
                pollard_rho(N / g, c + 1));
        }
        if (power == lambda) {
            saved = x;
            power <<= 1;
            lambda = 0;
        }
        lambda++;
    }
    return {};
}
vector<pair<T, int>> pollard_rho(const T& N) {
    return pollard_rho(N, static_cast<T>(1));
}

vector<pair<T, int>> factorize(T X) {
    if (X <= 1) {
        return {};
    }
    if (X <= precalculated) {
        vector<pair<T, int>> ret;
        while (X > 1) {
            if (!ret.empty() && ret.back().first ==
                least[X]) {
                ret.back().second++;
            } else {
                ret.emplace_back(least[X], 1);
            }
            X /= least[X];
        }
        return ret;
    }
    if (X <= static_cast<int64_t>(precalculated) *
        precalculated) {
        vector<pair<T, int>> ret;
        if (!miller_rabin(X)) {
            for (T i : primes) {
                T t = X / i;
                if (i > t) {
                    break;
                }
                if (X == t * i) {
                    int cnt = 0;
                    while (X % i == 0) {
                        X /= i;
                        cnt++;
                    }
```

```
                    ret.emplace_back(i, cnt);
                    if (miller_rabin(X)) {
                        break;
                    }
                }
            }
        }
        if (X > 1) {
            ret.emplace_back(X, 1);
        }
        return ret;
    }
    return pollard_rho(X);
}

vector<T> build_divisors_from_factors(const
    vector<pair<T, int>>& factors) {
    vector<T> divisors = {1};
    for (auto& p : factors) {
        int sz = int(divisors.size());
        for (int i = 0; i < sz; i++) {
            T cur = divisors[i];
            for (int j = 0; j < p.second; j++) {
                cur *= p.first;
                divisors.push_back(cur);
            }
        }
    }
    sort(begin(divisors), end(divisors));
    return divisors;
}
// Time: O(  N  )
// Count of numbers in [1,N] coprime to N
int euler_totient(int N) {
    T res = N;
    for (int i = 2; i * i <= N; ++i) {
        if (N % i == 0) {
            while (N % i == 0) N /= i;
            res -= res / i;
        }
    }
    if (N > 1) {
        res -= res / N;
    }
    return res;
}

// Time: O(N.log(log(N)))
// Count each prime's contribution only once
vector<int> euler_totient_one_to_n(const int &N) {
    vector<int>phi(N + 1);
    iota(begin(phi), end(phi), 0);
    for (int i = 2; i <= N; ++i) {
        if (phi[i] == i) {
            for (int j = i; j <= N; j += i) {
                phi[j] -= phi[j] / i;
            }
        }
    }
    return phi;
}

// Time: O(N.log(N))
// As     (d|n) phi(d) = n by Gauss
vector<int> slow_euler_totient_one_to_n(const int &N) {
    vector<int>phi(N + 1);
    phi[0] = 0;
    phi[1] = 1;
    iota(begin(phi) + 2, end(phi), 1);
    for (int i = 2; i <= N; i++) {
        for (int j = 2 * i; j <= N; j += i) {
            phi[j] -= phi[i];
        }
    }
    return phi;
}

T number_of_divisors(const T &N) {
    auto v = factorize(N);
    T res = 1;
    for (auto x : v) {
        res *= (x.second + 1);
    }
    return res;
```

```
    }

    int64_t sum_of_divisors(const T &N) {
        int64_t res = 1;
        const int64_t MOD_ = 9223372036854775783;
        modular<int64_t>M(MOD_);
        auto v = factorize(N);
        for (auto x : v) {
            int64_t p = x.first;
            int64_t e = x.second;
            res = M.mul(res, M.mul(M.sub(M.power(p, e + 1),
                1), M.inv(p - 1)));
        }
        return (int64_t)res;
    }
};
```

4.8 FFT Namespace

```
namespace fft {

const double PI = acos(-1);

struct num {
    double x, y;

    num(double x = 0, double y = 0) : x(x), y(y) {
    }

    num operator+(const num& o) const {
        return num(x + o.x, y + o.y);
    }

    num operator-(const num& o) const {
        return num(x - o.x, y - o.y);
    }

    num operator*(const num& o) const {
        return num(x * o.x - y * o.y, x * o.y + y * o.x);
    }
};

num conj(num a) {
    return num(a.x, -a.y);
}

vector<num> fa, fb, roots = {num(0, 0), num(1, 0)};
vector<int> rev = {0, 1};
int base = 1;

void ensure_base(int nbase) {
    if (nbase <= base) {
        return;
    }
    rev.resize(1 << nbase);
    for (int i = 0; i < (1 << nbase); ++i) {
        rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (nbase -
            1));
    }
    roots.resize(1 << nbase);
    while (base < nbase) {
        double angle = 2 * PI / (1 << (base + 1));
        for (int i = 1 << (base - 1); i < (1 << base); ++i) {
            roots[i << 1] = roots[i];
            double ang = angle * ((i << 1) + 1 - (1 <<
                base));
            roots[i << 1 | 1] = num(cos(ang), sin(ang));
        }
        ++base;
    }
}

void dft(vector<num>& a, int n) {
    int zeros = __builtin_ctz(n);
    ensure_base(zeros);
    int shift = base - zeros;
    for (int i = 0; i < n; ++i) {
        if (i < (rev[i] >> shift)) {
            swap(a[i], a[rev[i] >> shift]);
        }
    }
    for (int i = 1; i < n; i <<= 1) {
```

```
        for (int j = 0; j < n; j += i << 1) {
            for (int k = 0; k < i; ++k) {
                num x = a[j + k], y = a[j + k + i] * roots[i
                    + k];
                a[j + k] = x + y;
                a[j + k + i] = x - y;
            }
        }
    }
}

vector<long long> multiply(const vector<int>& a, const
    vector<int>& b) {
    int need = a.size() + b.size() - 1, nbase = 0;
    while (1 << nbase < need) {
        ++nbase;
    }
    ensure_base(nbase);
    bool equal = a == b;
    int sz = 1 << nbase;
    if (sz > (int) fa.size()) {
        fa.resize(sz);
    }
    if (sz > (int) fb.size()) {
        fb.resize(sz);
    }
    for (int i = 0; i < (int) a.size(); i++) {
        int x = a[i];
        fa[i] = num(x & ((1 << 15) - 1), x >> 15);
    }
    for (int i = (int) a.size(); i < sz; ++i) {
        fa[i] = num(0, 0);
    }
    dft(fa, sz);
    if (equal) {
        for (int i = 0; i < sz; ++i) {
            fb[i] = fa[i];
        }
    } else {
        for (int i = 0; i < (int) b.size(); ++i) {
            int x = b[i];
            fb[i] = num(x & ((1 << 15) - 1), x >> 15);
        }
        for (int i = (int) b.size(); i < sz; ++i) {
            fb[i] = num(0, 0);
        }
        dft(fb, sz);
    }
    double ratio = 0.25 / sz;
    num r1(1, 0), r2(0, -1), r3(ratio, 0), r4(0, -ratio),
        r5(0, 1);
    for (int i = 0; i <= sz >> 1; ++i) {
        int j = (sz - i) & (sz - 1);
        num a1 = (fa[i] + conj(fa[j])) * r1;
        num a2 = (fa[i] - conj(fa[j])) * r2;
        num b1 = (fb[i] + conj(fb[j])) * r3;
        num b2 = (fb[i] - conj(fb[j])) * r4;
        if (i != j) {
            num c1 = (fa[j] + conj(fa[i])) * r1;
            num c2 = (fa[j] - conj(fa[i])) * r2;
            num d1 = (fb[j] + conj(fb[i])) * r3;
            num d2 = (fb[j] - conj(fb[i])) * r4;
            fa[i] = c1 * d1 + c2 * d2 * r5;
            fb[i] = c1 * d2 + c2 * d1;
        }
        fa[j] = a1 * b1 + a2 * b2 * r5;
        fb[j] = a1 * b2 + a2 * b1;
    }
    dft(fa, sz);
    dft(fb, sz);
    vector<long long> c(need);
    for (int i = 0; i < need; i++) {
        long long aa = fa[i].x + 0.5;
        long long bb = fb[i].x + 0.5;
        long long cc = fa[i].y + 0.5;
        c[i] = aa + (bb << 15) + (cc << 30);
    }
    return c;
}
}

using fft::multiply;

vector<long long> operator*(vector<int>& a, const
```

```
    vector<int>& b) {
    return multiply(a, b);
}

template <typename T>
vector<T>& operator+=(vector<T>& a, const vector<T>& b) {
    if (a.size() < b.size()) {
        a.resize(b.size());
    }
    for (int i = 0; i < (int) b.size(); i++) {
        a[i] += b[i];
    }
    return a;
}

template <typename T>
vector<T> operator+(const vector<T>& a, const vector<T>& b) {
    vector<T> c = a;
    return c += b;
}
```

4.9 NTT Namespace

```
const int mod = (119 << 23) + 1;//998244353
inline void add(int &x, int y)
{
    x += y;
    if (x >= mod)
    {
        x -= mod;
    }
}

inline void sub(int &x, int y)
{
    x -= y;
    if (x < 0)
    {
        x += mod;
    }
}

inline int mul(int x, int y)
{
    return (long long) x * y % mod;
}

inline int power(int x, int y)
{
    int res = 1;
    for (; y; y >>= 1, x = mul(x, x))
    {
        if (y & 1)
        {
            res = mul(res, x);
        }
    }
    return res;
}

inline int inv(int a)
{
    a %= mod;
    if (a < 0)
    {
        a += mod;
    }
    int b = mod, u = 0, v = 1;
    while (a)
    {
        int t = b / a;
        b -= t * a;
        swap(a, b);
        u -= t * v;
        swap(u, v);
    }
    if (u < 0)
    {
        u += mod;
    }
    return u;
}
```

```
namespace ntt
{
    // nbase is needed base
    // {current limit is 1<<base,primitive
       root,mod=c*pow(2,max_base)+1}
    // Therefore nbase<=base<=max_base must hold at all times
    int base = 1, root = -1, max_base = -1;
    // {rev is for inplace computation,roots are roots of
       unity under modulo m}
    // roots are started utilizing from index 1 hence first
       element is zero
    // roots are {1,w,w*2,...,w^(n-1)}
    vector<int> rev = {0, 1}, roots = {0, 1};

    void init()
    {
        /*
        We use the fact that for modules of the form
           p=c(2^k)+1 (and p is prime),
        there always exists the 2^k-th root of unity. It can
           be shown that g^c
        is such a 2^k-th root of unity, where g is a
           primitive root of p.
        */
        int temp = mod - 1;
        max_base = 0;
        while (temp % 2 == 0)
        {
            temp >>= 1;
            ++max_base;
        }
        // mod=(temp<<max_base+1);
        root = 2; // Primitive Root
        while (true)
        {
            if (power(root, 1 << max_base) == 1 &&
               power(root, 1 << max_base - 1) != 1)
            {
                break;
            }
            ++root;
        }
    }

    void ensure_base(int nbase)
    {
        if (max_base == -1)
        {
            init();
        }
        if (nbase <= base)
        {
            return;
        }
        assert(nbase <= max_base);
        rev.resize(1 << nbase);
        for (int i = 0; i < 1 << nbase; ++i)
        {
            rev[i] = rev[i >> 1] >> 1 | (i & 1) << nbase - 1;
        }
        roots.resize(1 << nbase);
        while (base < nbase)
        {
            int z = power(root, 1 << max_base - 1 - base);
            for (int i = 1 << base - 1; i < 1 << base; ++i)
            {
                roots[i << 1] = roots[i];
                roots[i << 1 | 1] = mul(roots[i], z);
            }
            ++base;
        }
    }
    void dft(vector<int> &a)
    {
        int n = a.size();
        int zeros = __builtin_ctz(n); //log2(n)
        ensure_base(zeros);// ensure that size of array
           should be less than equal to 1<<max_base
        int shift = base - zeros;
        for (int i = 0; i < n; ++i)
        {
            if (i < rev[i] >> shift)
            {
```

```
            swap(a[i], a[rev[i] >> shift]);
        }
    }
    for (int i = 1; i < n; i <<= 1)
    {
        for (int j = 0; j < n; j += i << 1)
        {
            for (int k = 0; k < i; ++k)
            {
                int x = a[j + k], y = mul(a[j + k + i],
                    roots[i + k]);
                a[j + k] = (x + y) % mod;
                a[j + k + i] = (x + mod - y) % mod;
            }
        }
    }
}

vector<int> multiply(vector<int> a, vector<int> b)
{
    int need = a.size() + b.size() - 1, nbase = 0;
    while (1 << nbase < need)
    {
        ++nbase;
    }
    ensure_base(nbase);
    int sz = 1 << nbase;
    a.resize(sz);
    b.resize(sz);
    bool equal = a == b;
    dft(a);
    if (equal)
    {
        b = a;
    }
    else
    {
        dft(b);
    }
    int inv_sz = inv(sz);
    for (int i = 0; i < sz; ++i)
    {
        a[i] = mul(mul(a[i], b[i]), inv_sz);
    }
    reverse(a.begin() + 1, a.end());
    dft(a);
    a.resize(need);
    return a;
}

vector<int> inverse(vector<int> a)
{
    int n = a.size(), m = n + 1 >> 1;
    if (n == 1)
    {
        return vector<int>(1, inv(a[0]));
    }
    else
    {
        vector<int> b = inverse(vector<int>(a.begin(),
            a.begin() + m));
        int need = n << 1, nbase = 0;
        while (1 << nbase < need)
        {
            ++nbase;
        }
        ensure_base(nbase);
        int sz = 1 << nbase;
        a.resize(sz);
        b.resize(sz);
        dft(a);
        dft(b);
        int inv_sz = inv(sz);
        for (int i = 0; i < sz; ++i)
        {
            a[i] = mul(mul(mod + 2 - mul(a[i], b[i]),
                b[i]), inv_sz);
        }
        reverse(a.begin() + 1, a.end());
        dft(a);
        a.resize(n);
        return a;
    }
}
```

```
}

using ntt::multiply;
using ntt::inverse;

vector<int> &operator += (vector<int> &a, const vector<int>
    &b)
{
    if (a.size() < b.size())
    {
        a.resize(b.size());
    }
    for (int i = 0; i < b.size(); ++i)
    {
        add(a[i], b[i]);
    }
    return a;
}

vector<int> operator + (const vector<int> &a, const
    vector<int> &b)
{
    vector<int> c = a;
    return c += b;
}

vector<int> &operator -= (vector<int> &a, const vector<int>
    &b)
{
    if (a.size() < b.size())
    {
        a.resize(b.size());
    }
    for (int i = 0; i < b.size(); ++i)
    {
        sub(a[i], b[i]);
    }
    return a;
}

vector<int> operator - (const vector<int> &a, const
    vector<int> &b)
{
    vector<int> c = a;
    return c -= b;
}

vector<int> &operator *= (vector<int> &a, const vector<int>
    &b)
{
    if (min(a.size(), b.size()) < 128)
    {
        vector<int> c = a;
        a.assign(a.size() + b.size() - 1, 0);
        for (int i = 0; i < c.size(); ++i)
        {
            for (int j = 0; j < b.size(); ++j)
            {
                add(a[i + j], mul(c[i], b[j]));
            }
        }
    }
    else
    {
        a = multiply(a, b);
    }
    return a;
}

vector<int> operator * (const vector<int> &a, const
    vector<int> &b)
{
    vector<int> c = a;
    return c *= b;
}

vector<int> &operator /= (vector<int> &a, const vector<int>
    &b)
{
    int n = a.size(), m = b.size();
    if (n < m)
    {
        a.clear();
    }
```

```
    else
    {
        vector<int> c = b;
        reverse(a.begin(), a.end());
        reverse(c.begin(), c.end());
        c.resize(n - m + 1);
        a *= inverse(c);
        a.erase(a.begin() + n - m + 1, a.end());
        reverse(a.begin(), a.end());
    }
    return a;
}

vector<int> operator / (const vector<int> &a, const
   vector<int> &b)
{
    vector<int> c = a;
    return c /= b;
}

vector<int> &operator %= (vector<int> &a, const vector<int>
   &b)
{
    int n = a.size(), m = b.size();
    if (n >= m)
    {
        vector<int> c = (a / b) * b;
        a.resize(m - 1);
        for (int i = 0; i < m - 1; ++i)
        {
            sub(a[i], c[i]);
        }
    }
    return a;
}

vector<int> operator % (const vector<int> &a, const
   vector<int> &b)
{
    vector<int> c = a;
    return c %= b;
}
```

```
// Derivative of f(x)->(n-1) terms
vector<int> derivative(const vector<int> &a)
{
    int n = a.size();
    vector<int> b(n - 1);
    for (int i = 1; i < n; ++i)
    {
        b[i - 1] = mul(a[i], i);
    }
    return b;
}
// Integration of f(x)=g(x) such that g(0)=0->(n+1) terms
vector<int> integration(const vector<int> &a)
{
    int n = a.size();
    vector<int> b(n + 1), invs(n + 1);
    for (int i = 1; i <= n; ++i)
    {
        invs[i] = i == 1 ? 1 : mul(mod - mod / i, invs[mod %
           i]);
        b[i] = mul(a[i - 1], invs[i]);
    }
    return b;
}

vector<int> logarithm(const vector<int> &a)
{
    vector<int> b = integration(derivative(a) * inverse(a));
    b.resize(a.size());
    return b;
}

vector<int> exponent(const vector<int> &a)
{
    vector<int> b(1, 1);
    while (b.size() < a.size())
    {
        vector<int> c(a.begin(), a.begin() + min(a.size(),
           b.size() << 1));
        add(c[0], 1);
        vector<int> old_b = b;
        b.resize(b.size() << 1);
```

```
        c -= logarithm(b);
        c *= old_b;
        for (int i = b.size() >> 1; i < b.size(); ++i)
        {
            b[i] = c[i];
        }
    }
    b.resize(a.size());
    return b;
}

vector<int> power(const vector<int> &a, int m)
{
    int n = a.size(), p = -1;
    vector<int> b(n);
    for (int i = 0; i < n; ++i)
    {
        if (a[i])
        {
            p = i;
            break;
        }
    }
    if (p == -1)
    {
        b[0] = !m;
        return b;
    }
    if ((long long) m * p >= n)
    {
        return b;
    }
    int mu = power(a[p], m), di = inv(a[p]);
    vector<int> c(n - m * p);
    for (int i = 0; i < n - m * p; ++i)
    {
        c[i] = mul(a[i + p], di);
    }
    c = logarithm(c);
    for (int i = 0; i < n - m * p; ++i)
    {
        c[i] = mul(c[i], m);
    }
    c = exponent(c);
    for (int i = 0; i < n - m * p; ++i)
    {
        b[i + m * p] = mul(c[i], mu);
    }
    return b;
}

vector<int> sqrt(const vector<int> &a)
{
    vector<int> b(1, 1);
    while (b.size() < a.size())
    {
        vector<int> c(a.begin(), a.begin() + min(a.size(),
            b.size() << 1));
        vector<int> old_b = b;
        b.resize(b.size() << 1);
        c *= inverse(b);
        for (int i = b.size() >> 1; i < b.size(); ++i)
        {
            b[i] = mul(c[i], mod + 1 >> 1);
        }
    }
    b.resize(a.size());
    return b;
}

vector<int> multiply_all(int l, int r, vector<vector<int>>
    &all)
{
    if (l > r)
    {
        return vector<int>();
    }
    else if (l == r)
    {
        return all[l];
    }
    else
    {
        int y = l + r >> 1;
```

```
        return multiply_all(l, y, all) * multiply_all(y + 1,
            r, all);
    }
}

vector<int> evaluate(const vector<int> &f, const vector<int>
    &x)
{
    int n = x.size();
    if (!n)
    {
        return vector<int>();
    }
    vector<vector<int>> up(n * 2);
    for (int i = 0; i < n; ++i)
    {
        up[i + n] = vector<int> {(mod - x[i]) % mod, 1};
    }
    for (int i = n - 1; i; --i)
    {
        up[i] = up[i << 1] * up[i << 1 | 1];
    }
    vector<vector<int>> down(n * 2);
    down[1] = f % up[1];
    for (int i = 2; i < n * 2; ++i)
    {
        down[i] = down[i >> 1] % up[i];
    }
    vector<int> y(n);
    for (int i = 0; i < n; ++i)
    {
        y[i] = down[i + n][0];
    }
    return y;
}

vector<int> interpolate(const vector<int> &x, const
    vector<int> &y)
{
    int n = x.size();
    vector<vector<int>> up(n * 2);
    for (int i = 0; i < n; ++i)
    {
        up[i + n] = vector<int> {(mod - x[i]) % mod, 1};
    }
    for (int i = n - 1; i; --i)
    {
        up[i] = up[i << 1] * up[i << 1 | 1];
    }
    vector<int> a = evaluate(derivative(up[1]), x);
    for (int i = 0; i < n; ++i)
    {
        a[i] = mul(y[i], inv(a[i]));
    }
    vector<vector<int>> down(n * 2);
    for (int i = 0; i < n; ++i)
    {
        down[i + n] = vector<int>(1, a[i]);
    }
    for (int i = n - 1; i; --i)
    {
        down[i] = down[i << 1] * up[i << 1 | 1] + down[i <<
            1 | 1] * up[i << 1];
    }
    return down[1];
}
/*  Usage cases
    vector<int>a{1, 2, 3, 4};
    vector<int>b{2, 3};
    vector<int>c = multiply(a, b);
    vector<vector<int>>d{{1, 2, 3}, {2, 3}, {5, 6, 7}};
    vector<int>e = multiply_all(0, 2, d);
*/
```

4.10 Primitive Root

```
const int maxn = 1e5 + 10;
constexpr int mod = 998244353;
inline int add(int x, int y)
{
    return (x % mod + y % mod + mod) % mod;
}
```

```
inline int mul(int x, int y)
{
    return (x % mod * (y % mod)) % mod;
}
int power(int x, int y)
{
    int res = 1;
    x %= mod;
    for (; y; y >>= 1)
    {
        if (y & 1)
            res = res * x % mod;
        x = x * x % mod;
    }
    return res;
}
void generator(int n)
{
    int x = n - 1;
    vector<int>factors;
    for (int i = 2; i * 1ll * i <= x; i++)
    {
        if (x % i == 0)
        {
            while (x % i == 0)
                x /= i;
            factors.emplace_back(i);
        }
    }
    if (x != 1)
        factors.emplace_back(x);
    for (int i = 2; i <= 100; ++i) {
        bool flag = true;
        for (int j = 0; j < (int)factors.size(); ++j)
            flag &= (power(i, (n - 1) / factors[j]) != 1);
        if (flag)
            cout << i << '\n';
    }
}
// generator(mod);
```

4.11 NTT

```
const int maxn = 1 << 18;
constexpr int mod = 163577857;
const int g = 23;
int invfact[maxn], fact[maxn];
int gpow[30], invgpow[30];
inline int mul(int x, int y)
{
    return (x % mod * (y % mod)) % mod;
}
inline int add(int x, int y)
{
    return (x % mod + y % mod + mod) % mod;
}
int power(int x, int y)
{
    int res = 1;
    x %= mod;
    for (; y; y >>= 1)
    {
        if (y & 1)
            res = res * x % mod;
        x = x * x % mod;
    }
    return res;
}
int ncr(int n, int r)
{
    if (r<0 or r>n)
        return 0;
    return mul(fact[n], mul(invfact[r], invfact[n - r]));
}
void pre()
{
    fact[0] = 1;
    forn(i, 1, maxn - 1)
    fact[i] = fact[i - 1] * i % mod;
    invfact[maxn - 1] = power(fact[maxn - 1], mod - 2);
    for (int i = maxn - 2; i >= 0; i--)
        invfact[i] = invfact[i + 1] * (i + 1) % mod;
    int where = (mod - 1) / 2, invg = power(g, mod - 2);
```

```
    int idx = 0;
    while (where % 2 == 0)
    {
        idx++;
        gpow[idx] = power(g, where);
        invgpow[idx] = power(invg, where);
        where /= 2;
    }
}
struct NTT
{
    vector<int>a, b;
    int n_;
    NTT(int n)
    {
        this->n_ = 1 << (int)(ceil((ld)log2(n + 1int)));
        a.resize(n_, 0), b.resize(n_, 0);
    }
    void fft(vector<int> &a, bool invert)
    {
        for (int i = 1, j = 0; i < n_; i++)
        {
            int bit = n_ >> 1;
            for (; j & bit; bit >>= 1)
                j ^= bit;
            j ^= bit;
            if (i < j) swap(a[i], a[j]);
        }
        for (int len = 2, idx = 1; len <= n_; len <<= 1,
           idx++)
        {
            int wlen = (invert ? invgpow[idx] : gpow[idx]);
            for (int i = 0; i < n_; i += len)
            {
                int w = 1;
                for (int j = i; j < i + (len >> 1); j++)
                {
                    int u = a[j], v = mul(a[j + (len >> 1)],
                       w);
                    a[j] = add(u, v);
                    a[j + (len >> 1)] = add(u, -v);
                    w = mul(w, wlen);
                }
            }
        }
        if (invert)
        {
            int n_1 = power(n_, mod - 2);
            rep(i, n_)
            a[i] = mul(a[i], n_1);
        }
    }
    void multiply(int na, int nb)
    {
        na++;
        nb++;
        fft(a, 0);
        fft(b, 0);
        rep(i, n_)
        a[i] = mul(a[i], b[i]);
        fft(a, 1);
        for (int i = na + nb - 1; i < n_; i++)
        {
            a[i] = 0;
        }
    }
};
/*
pre();
int n;
cin>>n;
NTT ntt(n);
for(int i = 0; i < n; ++i) {
    cin >> ntt.a[i];
}
for(int i = 0; i < n; ++i) {
    cin >> ntt.b[i];
}
ntt.multiply(n, n);
*/
```

4.12 FFT

```
using cd = complex<double>;
const int maxn = 1e6 + 10;
const long double pi = acos(-1);
long long a[maxn], b[maxn];
void fft(vector<cd> &a, bool invert)
{
    long long n = a.size();
    for (long long i = 1, j = 0; i < n; i++)
    {
        long long bit = n >> 1;
        for (; j & bit; bit >>= 1)
            j ^= bit;
        j ^= bit;
        if (i < j)
            swap(a[i], a[j]);
    }
    for (long long len = 2; len <= n; len <<= 1)
    {
        double ang = (2 * pi / len) * (invert ? -1 : 1);
        cd wlen(cos(ang), sin(ang));
        for (long long i = 0; i < n; i += len)
        {
            cd w(1);
            for (long long j = i; j < i + (len >> 1); j++)
            {
                cd u = a[j];
                cd v = w * a[j + (len >> 1)];
                a[j] = u + v;
                a[j + (len >> 1)] = u - v;
                w = w * wlen;
            }
        }
    }
    if (invert)
    {
        for (cd &x : a)
            x /= n;
    }
}
void multiply(vector<long long>a, vector<long long>b,int n)
{
    vector<cd>na(begin(a), end(a)), nb(begin(b), end(b));
    int n_ = 1;
    while (n_ < 2 * n + 1)
        n_ <<= 1;
    na.resize(n_);
    nb.resize(n_);
    fft(na, false);
    fft(nb, false);
    rep(i, n_)
    na[i] = na[i] * nb[i];
    fft(na, true);
    rep(i, 2 * n - 1)
    cout << (long long)round(na[i].real()) << " ";
    cout << "\n";
}
```

4.13 FFT2

```
using cd = complex<double>;
const long double PI = acos(-1);
vector<int>rev;
int bit = 2;
void get_inv()
{
    rev.resize(bit);
    for (int i = 0; i < bit; i++)
    {
        rev[i] = ((rev[i >> 1] >> 1) | ((bit >> 1) * (i &
            1)));
    }
}
void fft(vector<cd> &a, int invert)
{
    for (int i = 0; i < bit; i++)
        if (i < rev[i])
            swap(a[i], a[rev[i]]);
    for (int m = 1; m < bit; m <<= 1)
    {
        cd wlen(cos(PI / m), invert * sin(PI / m));
        for (int i = 0; i < bit; i += (m << 1))
        {
```

```
            cd w(1, 0);
            for (int j = 0; j < m; j++, w = w * wlen)
            {
                cd x = a[i + j], y = a[i + j + m] * w;
                a[i + j] = x + y;
                a[i + j + m] = x - y;
            }
        }
    }
    if (invert == -1)
    {
        for (auto &x : a)
        {
            x /= bit;
        }
    }
}
int main() {
    get_inv();
    int n;
    cin >> n;
    while (bit <= n + n) (bit <<= 1);
    vector<cd>a(bit, 0), b(bit, 0);
    for (int i = 0 ; i < n; i++) {
        cin >> a[i];
    }
    for (int i = 0 ; i < n; i++) {
        cin >> b[i];
    }
    fft(a, 1);
    fft(b, 1);
    rep(i, bit)
    a[i] *= b[i];
    fft(a, -1);
    int ans[n];
    for (int i = 0 ; i < n; i++) {
        ans[i] = (int)(a[i].real() + 0.5);
    }
}
```

4.14 Linear Congruence

```
// Find solutions for AX mod B = C mod B in X
template<typename T>
class linear_congruence {
private:
    T A;
    T B;
    T C;
    T gcd;
public:
    linear_congruence(T A_, T B_, T C_): A(A_), B(B_), C(C_)
       {
        gcd = __gcd(A, B);
    }
    vector<T> solve() {
        if (gcd == 1) { // Unique solution
            modular<T>M(B);
            return {M.mul(M.inv(A), C)};
        }
        else if (C % gcd == 0) { // Number of unique
           solutions = gcd;
            vector<T>solutions;
            T a = A / gcd;
            T b = B / gcd;
            T c = C / gcd;
            // aX mod b = c mod b
            modular<T>M1(b);
            T X = M1.mul(M1.inv(a), c);
            modular<T>M2(B);
            for (T i = 0; i < gcd; ++i) {
                solutions.emplace_back(M2.add(X, M2.mul(i,
                   b)));
            }
            return solutions;
        }
        else {
            return {}; // No solution
        }
    }
};
```

4.15 Linear Diophantine

```
// Solves ax + by = c (when a, b, and c are given).
// If a = b = 0 then there is no solution if c != 0 and
   infinitely many solutions if c = 0. We will ignore this
   case
// From extended_gcd algorithm find xg, yg such that a.xg +
   b.yg = gcd(a, b) = g(say);
// Then on solution is a.xg.c / g + b.yg.c / g = c; with x =
   xg.c/g and y = yg.c/g
// As g | a and g | b therefore g must also divide any
   linear combination of a and b.
// This gives the proof to the following algorithm.

template<typename T>
class linear_diophantine :  public linear_equations<T> {
protected:
    using linear_equations<T>::A;
    using linear_equations<T>::B;
    using linear_equations<T>::gcd;
    using linear_equations<T>::extended_gcd;
public:
    T C;
    linear_diophantine(T A_, T B_, T C_):
       linear_equations<T>(A_, B_) {
        C = C_;
    }

    vector<T> find_one_solution() {
        T x, y;
        T g = extended_gcd(abs(A), abs(B), x, y);
        if (C % g != 0) {
            return vector<T>();
        }
        vector<T> solution;
        x *= C / g;
        y *= C / g;
        if (A < 0) x = -x;
        if (B < 0) y = -y;
        assert(A * x + B * y == C);
        solution.push_back(x);
        solution.push_back(y);
        return solution;
    }

    vector<vector<T>> n_solutions(int N) {
        auto solution = find_one_solution();
        if (solution.empty()) {
            return vector<vector<T>>();
        }
        T x = solution[0], y = solution[1];
        vector<vector<T>> solutions;
        for (int i = 0; i < N; ++i) {
            T x_ = x + B / gcd * i;
            T y_ = y - A / gcd * i;
            assert(A * x_ + B * y_ == C);
            solutions.push_back({x_, y_});
        }
        return solutions;
    }
    T solve_with_one_var(T coef, T C, T lower, T upper) {
        if (C % coef != 0) {
            return 0;
        }
        T solution = -C / coef;
        return (T) (solution >= lower && solution <= upper)
           ? 1 : 0;
    }
    void shift_solution(T & x, T & y, T a, T b, T cnt) {
        x += cnt * b;
        y -= cnt * a;
    }
    T find_inrange(T lx, T rx, T ly, T ry) {
        if (rx < lx || ry < ly) {
            return 0;
        }

        if (A == 0 && B == 0) {
            if (C != 0) {
                return 0;
            }
            return (rx - lx + 1) * (ry - ly + 1);
        }
```

```
if (A == 0) {
    return solve_with_one_var(B, C, ly, ry) * (rx -
        lx + 1);
}
if (B == 0) {
    return solve_with_one_var(A, C, lx, rx) * (ry -
        ly + 1);
}

if (C % gcd != 0) {
    return 0;
}

T u = A / gcd;
T v = B / gcd;
auto s = find_one_solution();

int sign_a = u > 0 ? +1 : -1;
int sign_b = v > 0 ? +1 : -1;
T x = s[0], y = s[1];

// Find lx1
shift_solution(x, y, u, v, (lx - x) / v);
if (x < lx)
    shift_solution(x, y, u, v, sign_b);
T lx1 = x;

// Find rx1
shift_solution(x, y, u, v, (rx - x) / v);
if (x > rx)
    shift_solution(x, y, u, v, -sign_b);
T rx1 = x;

if (lx1 > rx1)
    swap(lx1, rx1);

// Find lx2
shift_solution(x, y, u, v, -(ly - y) / u);
if (y < ly)
    shift_solution(x, y, u, v, -sign_a);
T lx2 = x;

// Find rx2
shift_solution(x, y, u, v, -(ry - y) / u);
if (y > ry)
    shift_solution(x, y, u, v, sign_a);
T rx2 = x;

if (lx2 > rx2)
    swap(lx2, rx2);
T l = max(lx1, lx2);
T r = min(rx1, rx2);

if (l > r)
    return 0;
return (r - l) / abs(v) + 1;
    }
};
```

4.16 Linear Equations

```
// Solves the GCD, Extended GCD, and Linear Diophantine
   Equations
template<typename T>
class linear_equations {
protected:
    T A, B;
    T gcd;
public:
    linear_equations(T A_, T B_): A(A_), B(B_) {
        gcd = euclid_gcd(A, B);
    }

    // a >= 0, b > 0
    // O(log(min(a, b)))
    // Two consecutive fibonacci numbers are the worst case
       input for euclid's gcd
    // As g | a and g | b therefore g must also divide any
       linear combination of a and b.
    // This gives the proof to the following algorithm.
    T euclid_gcd(T a, T b) {
        if (b == 0) {
```

```
            return a;
        }
        return euclid_gcd(b, a % b);
    }

    T lcm() {
        // Avoid possible T-overflow when g first divides a
           or b
        return (T) A / gcd * B;
    }

// ax + by = gcd(a, b) = g;
// bx1 + (a mod b)y1 = g;
// bx1 + [a - (a/b)*b]y1 = g;
// ay1 + b(x1 - (a/b)y1) = g;
// Therfore, x = y1 and y = x1 - (a/b)y1;
    T extended_gcd(T a, T b, T &x, T &y) {
        if (b == 0) {
            x = 1, y = 0;
            return a;
        }
        T x1, y1;
        T g = extended_gcd(b, a % b, x1, y1);
        x = y1;
        y = x1 - (a / b) * y1;
        return g;
    }
};
```

4.17 Markov Chains

```
const int maxn = 1e5 + 10;
void multiply(vector<vector<ld>> &a, vector<vector<ld>>b,
   int n)
{
    vector<vector<ld>>c(n, vector<ld>(n, 0));
    for(int i = 0; i < n; i++)
    {
        for(int j = 0; j < n; j++)
        {
            c[i][j] = 0;
            for(int k = 0; k < n; k++)
            {
                c[i][j] += (a[i][k] * b[k][j]);
            }
        }
    }
    a = c;
    return;
}
struct Query
{
    int start, finish, t, idx;
    bool operator<(const Query &other) const
    {
        return t < other.t;
    }
};
int32_t main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);
    cout << fixed << setprecision(10);
    int n;
    cin >> n;
    vector<vector<ld>>markov_matrix(n, vector<ld>(n));
    rep(i, n)
    {
        ld sum{};
        rep(j, n)
        {
            cin >> markov_matrix[i][j];
            sum += markov_matrix[i][j];
        }
        rep(j, n)
        {
            markov_matrix[i][j] /= sum;
        }
    }
    vector<vector<ld>>markov_matrixb = markov_matrix;
    int q;
```

```
    cin >> q;
    vector<Query>query(q);
    rep(i, q)
    {
        cin >> query[i].start >> query[i].finish >>
           query[i].t;
        query[i].idx = i;
    }
    sort(begin(query), end(query));
    vector<ld>out(q);
    int curr_time = 1;
    rep(i, q)
    {
        while(curr_time < query[i].t)
        {
            multiply(markov_matrix, markov_matrixb, n);
            curr_time++;
        }
        out[query[i].idx] = markov_matrix[query[i].start -
           1][query[i].finish - 1];
    }
    rep(i, q)
    {
        cout << out[i] << "\n";
    }
    return 0;
}
```

4.18 Matrix Exponentiation (Iterative)

```
// Fibonacci Numbers
using ll = long long;
const int mod = 1e9 + 7;
#define mat vector<vector<ll>>
inline ll add(ll a, ll b)
{
    a += b;
    if (a >= mod) a -= mod;
    return a;
}
inline ll mul(ll a, ll b)
{
    return  (a % mod * (b % mod)) % mod;
}
mat multiply( mat a, mat b)
{
    assert(a[0].size() == b.size());
    mat c(a.size(), vector<ll>(b[0].size(), 0));
    for(int i = 0; i < a.size(); ++i)
        for(int j = 0; j < b[0].size(); ++j)
            for(int k = 0; k < b.size(); ++k)
                c[i][j] = add(c[i][j], mul(a[i][k],
                   b[k][j]));
    return c;
}
int32_t main()
{
    mat a = {{0, 1}};
    mat b = {{0, 1}, {1, 1}};
    ll n;
    cin >> n;
    while(n)
    {
        if(n & 1)
            a = multiply(a, b);
        b = multiply(b, b);
        n >>= 1;
    }
    cout << a[0][0];
    return 0;
}
```

4.19 Matrix Exponentiation (Recursive)

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
const ll maxn = 1e5 + 5;
constexpr ll md = 1e9 + 7;
inline ll add(ll x, ll y)
```

```
{
    if(x + y >= md)
        return x + y - md;
    return x + y;
}
inline ll mul(ll x, ll y)
{
    return (x * 1ll * y) % md;
}
inline ll sub(ll x, ll y)
{
    if(x < y)
        return x + md - y;
    return x - y;
}
void multiply( vector<vector<ll>> &a,  vector<vector<ll>>b)
{
    vector<vector<ll>>c(2, vector<ll>(2, 0));
    for(int i = 0; i < 2; i++)
    {
        for(int j = 0; j < 2; j++)
        {
            c[i][j] = 0;
            for(int k = 0; k < 2; k++)
            {
                c[i][j] = add(c[i][j], mul(a[i][k],
                   b[k][j]));
            }
        }
    }
    a = c;
    return;
}
void raise_power(vector<vector<ll>> &b, ll n)
{
    vector<vector<ll>> f = b;
    if(n == 1)
        return;
    raise_power(b, n / 2);
    multiply(b, b);
    if(n & 1)
        multiply(b, f);
}
int32_t main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);
    int tt;
    cin >> tt;
    while(tt--)
    {
        ll n;
        cin >> n;
        /*
        [fn fn+1]=[fn-1 fn][0 1]
                          [1 1]
        [fn fn+1]=[fn-2 fn-1][0 1]^2
                             [1 1]
        [fn fn+1]=[f0 f1][0 1]^n
                         [1 1]
        */
        vector<vector<ll>>b(2, vector<ll>(2, 0));
        b[0][0] = 0;
        b[0][1] = 1;
        b[1][0] = 1;
        b[1][1] = 1;
        raise_power(b, 2 * n);
        cout << (b[1][0] - n + md) % md << "\n";
    }
    return 0;
}
```

4.20 Mint

```
template<const int _mod>
struct modular_int
{
    int x;
    static vector<int> inverse_list ;
    const static int inverse_limit;
    const static bool is_prime;
```

```
modular_int()
{
    x = 0;
}
template<typename T>
modular_int(const T z)
{
    x = (z % _mod);
    if (x < 0) x += _mod;
}
modular_int(const modular_int<_mod> *z)
{
    x = z->x;
}
modular_int(const modular_int<_mod> &z)
{
    x = z.x;
}
modular_int operator - (const modular_int<_mod> &m) const
{
    modular_int<_mod> U;
    U.x = x - m.x;
    if (U.x < 0) U.x += _mod;
    return U;
}
modular_int operator + (const modular_int<_mod> &m) const
{
    modular_int<_mod> U;
    U.x = x + m.x;
    if (U.x >= _mod) U.x -= _mod;
    return U;
}
modular_int &operator -= (const modular_int<_mod> &m)
{
    x -= m.x;
    if (x < 0) x += _mod;
    return *this;
}
modular_int &operator += (const modular_int<_mod> &m)
{
    x += m.x;
    if (x >= _mod) x -= _mod;
    return *this;
}
modular_int operator * (const modular_int<_mod> &m) const
{
    modular_int<_mod> U;
    U.x = (x * 1ull * m.x) % _mod;
    return U;
}
modular_int &operator *= (const modular_int<_mod> &m)
{
    x = (x * 1ull * m.x) % _mod;
    return *this;
}
template<typename T>
friend modular_int operator + (const T &l, const
   modular_int<_mod> &p)
{
    return (modular_int<_mod>(l) + p);
}
template<typename T>
friend modular_int operator - (const T &l, const
   modular_int<_mod> &p)
{
    return (modular_int<_mod>(l) - p);
}
template<typename T>
friend modular_int operator * (const T &l, const
   modular_int<_mod> &p)
{
    return (modular_int<_mod>(l) * p);
}
template<typename T>
friend modular_int operator / (const T &l, const
   modular_int<_mod> &p)
{
    return (modular_int<_mod>(l) / p);
}

int value() const
{
    return x;
}
```

```
modular_int operator ^ (const modular_int<_mod> &cpower)
    const
{
    modular_int<_mod> ans;
    ans.x = 1;
    modular_int<_mod> curr(this);
    int power = cpower.x;

    if (curr.x <= 1)
    {
        if (power == 0) curr.x = 1;
        return curr;
    }
    while ( power > 0)
    {
        if (power & 1)
        {
            ans *= curr;
        }
        power >>= 1;
        if (power) curr *= curr;
    }
    return ans;
}
modular_int operator ^ (long long power) const
{
    modular_int<_mod> ans;
    ans.x = 1;
    modular_int<_mod> curr(this);
    if (curr.x <= 1)
    {
        if (power == 0) curr.x = 1;
        return curr;
    }
    // Prime Mods
    if (power >= _mod && is_prime)
    {
        power %= (_mod - 1);
    }

    while ( power > 0)
    {
        if (power & 1)
        {
            ans *= curr;
        }
        power >>= 1;
        if (power) curr *= curr;

    }
    return ans;
}
modular_int operator ^ (int power) const
{
    modular_int<_mod> ans;
    ans.x = 1;
    modular_int<_mod> curr(this);
    if (curr.x <= 1)
    {
        if (power == 0) curr.x = 1;
        return curr;
    }
    while ( power > 0)
    {
        if (power & 1)
        {
            ans *= curr;
        }
        power >>= 1;
        if (power) curr *= curr;
    }
    return ans;
}
template<typename T>
modular_int &operator ^= (T power)
{
    modular_int<_mod> res = (*this)^power;
    x = res.x;
    return *this;
}
template<typename T>
modular_int pow(T x)
{
    return (*this)^x;
```

```
}
pair<long long, long long> gcd(const int a, const int b)
    const
{
    if (b == 0) return {1, 0};
    pair<long long, long long> c = gcd(b, a % b);
    return { c.second, c.first - (a / b) *c.second};
}
inline void init_inverse_list() const
{
    vector<int> &dp = modular_int<_mod>::inverse_list;
    dp.resize(modular_int<_mod>::inverse_limit + 1);
    int n = modular_int<_mod>::inverse_limit;
    dp[0] = 1;
    if (n > 1) dp[1] = 1;
    for (int i = 2; i <= n; ++i)
    {
        dp[i] = (dp[_mod % i] * 1ull * (_mod - _mod /
            i)) % _mod;
    }
}
modular_int<_mod> get_inv() const
{
    if (modular_int<_mod>::inverse_list.size() <
        modular_int<_mod>::inverse_limit + 1)
        init_inverse_list();
    if (this->x <= modular_int<_mod>::inverse_limit)
    {
        return modular_int<_mod>::inverse_list[this->x];
    }
    pair<long long, long long> G = gcd(this->x, _mod);
    return modular_int<_mod>(G.first);
}
modular_int<_mod> operator - () const
{
    modular_int<_mod> v(0);
    v -= (*this);
    return v;
}
modular_int operator / (const modular_int<_mod> &m) const
{
    modular_int<_mod> U(this);
    U *= m.get_inv();
    return U;
}
modular_int &operator /= (const modular_int<_mod> &m)
{
    (*this) *= m.get_inv();
    return *this;
}
bool operator==(const modular_int<_mod> &m) const
{
    return x == m.x;
}
bool operator < (const modular_int<_mod> &m) const
{
    return x < m.x;
}
template<typename T>
bool operator == (const T &m) const
{
    return (*this) == (modular_int<_mod>(m));
}
template<typename T>
bool operator < (const T &m) const
{
    return x < (modular_int<_mod>(m)).x;
}
template<typename T>
bool operator > (const T &m) const
{
    return x > (modular_int<_mod>(m)).x;
}
template<typename T>
friend bool operator == (const T &x, const
    modular_int<_mod> &m)
{
    return (modular_int<_mod>(x)).x == m.x;
}
template<typename T>
friend bool operator < (const T &x, const
    modular_int<_mod> &m)
{
    return (modular_int<_mod>(x)).x < m.x;
```

```
    }
    template<typename T>
    friend bool operator > (const T &x, const
       modular_int<_mod> &m)
    {
        return (modular_int<_mod>(x)).x > m.x;
    }
    friend istream &operator >> (istream &is,
       modular_int<_mod> &p)
    {
        int64_t val;
        is >> val;
        p.x = (val % _mod);
        if (p.x < 0) p.x += _mod;
        return is;
    }
    friend ostream &operator << (ostream &os, const
       modular_int<_mod> &p)
    {
        return os << p.x;
    }
};
const int mod = (int)1e9 + 7;
using mint = modular_int<mod>;
template<const int mod>
vector<int> modular_int<mod>::inverse_list;
template<const int mod>
const int modular_int<mod>::inverse_limit = -1;
template<const int mod>
const bool modular_int<mod>::is_prime = true;
```

4.21 Modular Arithmetic

```
template<typename T>
struct modular {
private:
    const T mod;
    int precalculated = 1;
public:
    vector<T>inverse{1, 1};
    modular(T mod_): mod(mod_) {}
    inline T add(T a, T b) {
        if (a >= mod) a %= mod;
        if (b >= mod) b %= mod;

        a += b;
        if (a >= mod) a -= mod;
        return a;
    }
    inline T sub(T a, T b) {
        if (a >= mod) a %= mod;
        if (b >= mod) b %= mod;

        a -= b;
        if (a < 0) a += mod;
        return a;
    }
    T mul(T a, T b)
    {
        long double x;
        T c;
        T r;
        if (a >= mod) a %= mod;
        if (b >= mod) b %= mod;
        x = a;
        c = x * b / mod;
        r = (T)(a * b - c * mod) % (T)mod;
        return r < 0 ? r + mod : r;
    }
    inline T power(T a, T b) {
        T res = 1;
        if (a >= mod) a %= mod;
        // If a and m are coprime, then Euler's theorem
           states that a^{phi(m)} mod m = 1
        // This reduces to a^{m-1}mod m = 1 for prime m
           which is Fermat's Little Theorem
        // Therefore, a^{n} mod m = a^{n mod phi(m)} mod m

        // Check if phi(mod) = m then only do b %= (m);

        while (b > 0) {
            if (b & 1)
```

```
                res = mul(res, a);
            a = mul(a, a);
            b = b >> 1;
        }
        return res;
    }
    inline T inv(T a) {
        a %= mod;
        if (a < 0) a += mod;
        T b = mod, u = 0, v = 1;
        while (a) {
            T t = b / a;
            b -= t * a;
            swap(a, b);
            u -= t * v;
            swap(u, v);
        }
        assert(b == 1);
        if (u < 0) u += mod;
        return u;
    }
    void inverse_inrange(const int &N) {
        if (N <= precalculated) return;
        inverse.resize(N + 1);
        for (int i = precalculated + 1; i <= N; ++i) {
            inverse[i] = sub(0, mul((mod / i), inverse[mod %
                i]));
        }
        precalculated = N;
    }
};
```

4.22 Primitive Root

```
// g is primitive root mod N, If and only if for all A
   coprime to N,
// there exists an integer K such that (g^ {K} mod N) == (A
   mod N), i.e.,
// powers of g can generate all such A's and thus also
   called generator
// of the multiplicative group of integers mod N;
// K is called index or discrete logarithm of A to the base
   g mod N;

// In most cases N is prime. In that case K runs from 1 to N
   - 1 as in NTT.
// N can be {1, 2, 4, power of odd prime, 2 * (power of odd
   prime)}

// Number of primitive root mod N =
   euler_totient(euler_totient(N)) or none.
template<typename T>
class primitive_root {
private:
    T N;
public:
    primitive_root(T N_): N(N_) {}
    vector<T> solve(int limit = 10) {
        vector<T> fact;
        vector<T>solutions;
        solutions.reserve(limit);

        const T phi = N - 1;
        T n = phi;
        for (T i = 2; i * i <= n; ++i)
            if (n % i == 0) {
                fact.push_back (i);
                while (n % i == 0)
                    n /= i;
            }
        if (n > 1)
            fact.push_back (n);

        modular<T>M(N);
        for (T res = 2; res <= N; ++res) {
            bool ok = true;
            for (size_t i = 0; i < fact.size() && ok; ++i) {
                ok &= (M.power(res, phi / fact[i]) != 1);
            }
            if (ok)  {
                solutions.emplace_back(res);
                if (int(solutions.size()) == limit) {
```

```
                    return solutions;
                }
            }
        }
        return solutions;
    }
};
```

5 Strings

5.1 Aho Corasick

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
const int K = 26;
struct Vertex
{
    int next[K];
    Vertex()
    {
        fill(begin(next), end(next), -1);
    }
};
template<int ALPHABET_SIZE, char MINIMAL_CHAR>
struct Aho_Corasick
{
    static constexpr int ROOT_ID = 0;
    vector<array<int, ALPHABET_SIZE>>edges;
    vector<int>suffixLink;
    vector<ll>sum;
    int current_node;
    explicit Aho_Corasick(const vector<pair<string, int>>
       &s): current_node(ROOT_ID), edges(1), suffixLink(1,
       -1), sum(1, 0)
    {
        edges[ROOT_ID].fill(-1);
        for(const auto &p : s)
        {
            int node = ROOT_ID;
            for(char c : p.first)
            {
                int req = c - 'a';
                if(edges[node][req] == -1)
                {
                    edges[node][req] = edges.size();
                    edges.emplace_back();
                    edges.back().fill(-1);
                    suffixLink.emplace_back(-1);
                    sum.emplace_back(0);
                }
                node = edges[node][req];
            }
            sum[node] += p.second;
        }
        queue<int>q;
        for(int i = 0; i < ALPHABET_SIZE; i++)
        {
            if(edges[ROOT_ID][i] == -1)
                edges[ROOT_ID][i] = 0;
        }
        for(int i = 0; i < ALPHABET_SIZE; i++)
        {
            if(edges[ROOT_ID][i] != ROOT_ID)
            {
                suffixLink[edges[ROOT_ID][i]] =
                   ROOT_ID;//All non empty links to root
                q.push(edges[ROOT_ID][i]);//Flood the queue
            }
        }
        //Go down one level each time using queue
        while(!q.empty())
        {
            int node = q.front();
            q.pop();
            for(int i = 0; i < ALPHABET_SIZE; i++)
            {
                if(edges[node][i] != -1)
                {
                    int suffix = suffixLink[node];
                    while(edges[suffix][i] == -1)
```

```
                {
                    suffix = suffixLink[suffix];
                }
                suffix = edges[suffix][i];
                suffixLink[edges[node][i]] = suffix;
                sum[edges[node][i]] += sum[suffix];
                q.push(edges[node][i]);
            }
        }
    }
    void setNode(int node)
    {
        current_node = node;
    }
    void resetNode()
    {
        setNode(ROOT_ID);
    }
    long long getCurrentNodeSum()
    {
        return sum[current_node];
    }
    void move(char c)
    {
        int req = c - 'a';
        while (edges[current_node][req] == -1)
            current_node = suffixLink[current_node];
        current_node = edges[current_node][req];
    }
};
int32_t main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);
    int tt;
    cin >> tt;
    while(tt--)
    {
        string a, b;
        cin >> a >> b;
        int n;
        cin >> n;
        vector<pair<string, int>>vec(n);
        for(int i = 0; i < n; i++)
        {
            cin >> vec[i].first >> vec[i].second;
        }
        Aho_Corasick<26, 'a'>aho_Corasick(vec);
        aho_Corasick.resetNode();
        vector<ll>prefASum(a.size());
        vector<int> prefANode(a.size());
        for (int i = 0; i < a.size(); i++)
        {
            aho_Corasick.move(a[i]);
            prefASum[i] = aho_Corasick.getCurrentNodeSum();
            if (i != 0)
            {
                prefASum[i] += prefASum[i - 1];
            }
            prefANode[i] = aho_Corasick.current_node;
        }
        aho_Corasick.resetNode();
        vector<long long> suffBSum(b.size());
        for (int i = 0; i < b.size(); i++)
        {
            aho_Corasick.move(b[i]);
            suffBSum[i] = aho_Corasick.getCurrentNodeSum();
        }
        for (int i = (int) b.size() - 2; i >= 0; i--)
        {
            suffBSum[i] += suffBSum[i + 1];
        }
        long long ans = 0;
        for (int i = 0; i < a.size(); i++)
        {
            for (int j = 0; j < b.size(); j++)
            {
                long long cur = prefASum[i];

                aho_Corasick.setNode(prefANode[i]);
                for (int k = j; k <= j + 24 && k < b.size();
                    k++)
                {
```

```
                aho_Corasick.move(b[k]);
                cur += aho_Corasick.getCurrentNodeSum();
            }
            if (j + 25 < b.size())
            {
                cur += suffBSum[j + 25];
            }
            ans = max(ans, cur);
        }
    }
    cout << ans << '\n';
  }
  return 0;
}
```

5.2 Polynomial Rolling Hashing Function

```
// A rolling hash has two parameters (p, a ) where p is the
   modulo and 0     a     < p  the base.
// 'p' should be a big prime and 'a' must be larger than the
   size  of the alphabet size.
// If two strings are same function never fails but it may
   fail if two different strings are compared.
// In the following code, fixed prime number is used and a
   is picked randomly over {alphabet_size + 1, p - 1}
// Let the rolling function be faild for two different
   strings S,T of equal length n. Therefore, h(S) = h(T)
// Sum_over_n (A^{n-i-1} * S[i]) mod p = Sum_over_n
   (A^{n-i-1} * T[i]) mod p
// Sum_over_n (A^{n-i-1} * S[i] - T[i]) = 0 (mod p)
// Therefore h(S) = h(T) if the equation uses A as a root of
   degree <= (n-1)
// Probability of choosing A as a root = (n-1)/p

template<int MOD_>
struct str_hash {

    static int P;
    int N;
    string S;
    vector<int64_t>Hash, Powers;

    str_hash(string S_) {
        N = static_cast<int>(S_.size());
        Hash.resize(N);
        Powers.resize(N);
        S = S_;
        Powers[0] = 1;
        for (int i = 1; i < N; ++i) {
            Powers[i] = Powers[i - 1] * P % MOD_;
        }
        Hash[0] = S[0];
        for (int i = 1; i < N; ++i) {
            Hash[i] = (Hash[i - 1] * P + S[i]) % MOD_;
        }
    }
    int operator()(int l, int r) {
        assert(l <= r && l >= 0 && r < N);
        int res = Hash[r];
        if (l) {
            res = res - Hash[l - 1] * Powers[r - (l - 1)] %
               MOD_;
        }
        if (res < 0) res += MOD_;
        return res;
    }

};

mt19937 rng((int)
   chrono::steady_clock::now().time_since_epoch().count());

int uniform(int l, int r) {
    uniform_int_distribution<int> uid(l, r);
    return uid(rng);
}

const int ALPHABET_SIZE = 26;
template<int MOD> int str_hash<MOD>::P =
   uniform(ALPHABET_SIZE + 1, MOD - 1);
const int MOD_ = int(1e9) + 7;
```

5.3 String Rolling Hashing - modulo 2^61 - 1

```
// (Works for Mersenne Primes only!)
const int64_t MOD_ = int64_t(1LL << 61) - 1;
template<int64_t MOD_>
struct str_hash {

    static int64_t P;
    int N;
    string S;
    vector<int64_t>Hash, Powers;

    int64_t mul_mod(int64_t a, int64_t b) {
        const int64_t LOWER = (1LL << 30) - 1;
        const int64_t HIGHER = (1LL << 31) - 1;
        int64_t a1 = (a & LOWER), a2 = (a >> 30);
        int64_t b1 = (b & LOWER), b2 = (b >> 30);
        int64_t m = a1 * b2 + a2 * b1, h = a2 * b2;
        int64_t ans = a1 * b1 + (h >> 1) + ((h & 1) << 60) +
            (m >> 31) + ((m & HIGHER) << 30) + 1;
        ans = (ans & MOD_) + (ans >> 61);
        ans = (ans & MOD_) + (ans >> 61);
        return ans - 1;
    }

    str_hash(string S_) {
        N = static_cast<int>(S_.size());
        Hash.resize(N);
        Powers.resize(N);
        S = S_;
        Powers[0] = 1;
        for (int i = 1; i < N; ++i) {
            Powers[i] = mul_mod(Powers[i - 1], P);
        }
        Hash[0] = S[0];
        for (int i = 1; i < N; ++i) {
            Hash[i] = (mul_mod(Hash[i - 1], P) + S[i]) %
                MOD_;
        }
    }
    int64_t operator()(int l, int r) {
        assert(l <= r && l >= 0 && r < N);
        int64_t res = Hash[r];
        if (l) {
            res = res - mul_mod(Hash[l - 1], Powers[r - (l -
                1)]);
        }
        if (res < 0) res += MOD_;
        return res;
    }

};

mt19937_64
    rng((int64_t)chrono::steady_clock::now().time_since_epoch().cou

int64_t uniform(int64_t l, int64_t r) {
    uniform_int_distribution<int64_t> uid(l, r);
    return uid(rng);
}

const int ALPHABET_SIZE = 26;
template<int64_t MOD> int64_t str_hash<MOD>::P =
    uniform(ALPHABET_SIZE + 1, MOD - 1);
```

5.4 Knuth Morris Pratt Algorithm

```
template<typename T>
struct knuth_morris_pratt {
    int N;
    vector<int> P;
    T S;
    knuth_morris_pratt(const T& S_) : S(S_) {
        N = int(S.size());
        P.resize(N);
        int k = 0;
        for (int i = 1; i < N; ++i) {
            while (k > 0 && !(S[i] == S[k])) {
                k = P[k - 1];
            }
            if (S[i] == S[k]) {
                k++;
```

```
            }
            P[i] = k;
        }
    }

    // returns 0-indexed positions of occurences of S in H
    vector<int> kmp_search(const T& H) {
        vector<int>res;
        int M = int(H.size());
        if (M < N) {
            return res;
        }
        int k = 0;
        for (int i = 0; i < M; ++i) {
            while (k > 0 && (k == N || !(H[i] == S[k]))) {
                k = P[k - 1];
            }
            if (H[i] == S[k]) {
                k++;
            }
            if (k == N) {
                res.emplace_back(i - N + 1);
            }
        }
        return res;
    }
};
```

5.5 LCS using Suffix Array

```
string longest_common_substring(const string& S, const
    string& T) {
    int N = int(S.size());
    const char EXTRA = '$';
    assert(S.find(EXTRA) == string::npos);
    assert(T.find(EXTRA) == string::npos);
    suffix_array<string>SA(S + EXTRA + T);
    int split = SA.sa[0];
    int max_len = 0, idx = 0;
    for (int i = 1; i + 1 < SA.N; ++i) {
        if ((SA.sa[i] < split && SA.sa[i + 1] > split) ||
            (SA.sa[i] > split && SA.sa[i + 1] < split)) {
            if (max_len < SA.lcp[i]) {
                max_len = SA.lcp[i];
                idx = SA.sa[i];
            }
        }
    }
    if (idx >= N) {
        return T.substr(idx - N - 1, max_len);
    }
    else
        return S.substr(idx, max_len);
}
```

5.6 Suffix Array

```
template<typename String>
struct suffix_array {
public:
    String S;
    int N;
    sparse_table<int>ST;
    vector<int> sa, rank, lcp;

    suffix_array(const String& S_): S(S_), N(int(S.size())) {
        build_sa();
        build_rank();
        build_lcp();
        build_rmq();
    }

    int get_lcp(int a, int b) const {
        if (a == b) return N - a;
        a = rank[a], b = rank[b];
        if (a > b) {
            swap(a, b);
        }
        return ST.get_idempotent(a, b - 1);
    }
```

```
    int compare(int a, int b, int length) const {
        if (a == b) {
            return 0;
        }
        int common = get_lcp(a, b);
        if (common >= length) {
            return 0;
        }
        if (a + common >= N || b + common >= N) {
            return a + common >= N ? -1 : 1;
        }
        return S[a + common] < S[b + common] ? -1 : (S[a +
           common] == S[b + common] ? 0 : 1);
    }

private:
    const int alphabet_size = 256;
    void build_sa() {
        sa.resize(N);
        auto init = [&]() {
            vector<int> aux(alphabet_size, 0);
            for (int i = 0; i < N; i++) {
                aux[S[i]]++;
            }
            int sum = 0;
            for (int i = 0; i < alphabet_size; i++) {
                int add = aux[i];
                aux[i] = sum;
                sum += add;
            }
            for (int i = 0; i < N; i++) {
                sa[aux[S[i]]++] = i;
            }
        };
        init();

        vector<int> sorted_by_second(N);
        vector<int> ptr_group(N);
        vector<int> new_group(N);
        vector<int> group(N);
        group[sa[0]] = 0;
        for (int i = 1; i < N; i++) {
            group[sa[i]] = group[sa[i - 1]] + (!(S[sa[i]] ==
               S[sa[i - 1]]));
        }
        int cnt = group[sa[N - 1]] + 1;
        int step = 1;
        while (cnt < N) {
            int at = 0;
            for (int i = N - step; i < N; i++) {
                sorted_by_second[at++] = i;
            }
            for (int i = 0; i < N; i++) {
                if (sa[i] - step >= 0) {
                    sorted_by_second[at++] = sa[i] - step;
                }
            }
            for (int i = N - 1; i >= 0; i--) {
                ptr_group[group[sa[i]]] = i;
            }
            for (int i = 0; i < N; i++) {
                int x = sorted_by_second[i];
                sa[ptr_group[group[x]]++] = x;
            }
            new_group[sa[0]] = 0;
            for (int i = 1; i < N; i++) {
                if (group[sa[i]] != group[sa[i - 1]]) {
                    new_group[sa[i]] = new_group[sa[i - 1]]
                       + 1;
                } else {
                    int pre = (sa[i - 1] + step >= N ? -1 :
                       group[sa[i - 1] + step]);
                    int cur = (sa[i] + step >= N ? -1 :
                       group[sa[i] + step]);
                    new_group[sa[i]] = new_group[sa[i - 1]]
                       + (pre != cur);
                }
            }
            swap(group, new_group);
            cnt = group[sa[N - 1]] + 1;
            step <<= 1;
        }
    }
```

```
    void build_rank() {
        rank.resize(N);
        for (int i = 0; i < N; ++i) {
            rank[sa[i]] = i;
        }
    }

    void build_lcp() {
        assert(int(sa.size()) == N);
        lcp.resize(N);
        for (int i = 0, k = 0; i < N; ++i) {
            k = max(0, k - 1);
            if (rank[i] == N - 1) {
                k = 0;
            } else {
                int j = sa[rank[i] + 1];
                while (i + k < N && j + k < N && S[i + k] ==
                   S[j + k]) {
                    k++;
                }
                lcp[rank[i]] = k;
            }
        }
    }
    void build_rmq() {
        ST = sparse_table<int>(lcp, [&](int i, int j) {
            return min(i, j);
        });
    }
public:
    int lower_bound(const String& P) const {
        int M = int(P.size());
        auto check = [&](const int m) {
            return S.compare(sa[m], M, P);
        };
        int l = 0, r = N;
        while (l < r)  {
            int m = l + (r - l) / 2;
            if (check(m) >= 0) {
                r = m ;
            } else {
                l = m + 1;
            }
        }
        return l;
    }
    int upper_bound(const String& P) const {
        int M = int(P.size());
        auto check = [&](const int m) {
            return S.compare(sa[m], M, P);
        };
        int l = 0, r = N;
        while (l < r)  {
            int m = l + (r - l) / 2;
            if (check(m) <= 0) {
                l = m + 1;
            } else {
                r = m;
            }
        }
        return r;
    }
};
```

5.7 Z Algorithm

```
template <typename T>
vector<int> z_function(int n, const T &s) {
    vector<int> z(n, n);
    int l = 0, r = 0;
    for (int i = 1; i < n; i++) {
        z[i] = (i > r ? 0 : min(r - i + 1, z[i - l]));
        while (i + z[i] < n && s[z[i]] == s[i + z[i]]) {
            z[i]++;
        }
        if (i + z[i] - 1 > r) {
            l = i;
            r = i + z[i] - 1;
        }
    }
    return z;
}
```

```
template <typename T>
vector<int> z_function(const T &s) {
    return z_function((int) s.size(), s);
}
```

6 Structures

6.1 DSU

```
struct DSU
{
    vector<int>par, sz;
    int comp;
    DSU(int n)
    {
        par.resize(n);
        comp = n;
        sz = vector<int>(n, 1);
        iota(par.begin(), par.end(), 0);
    }
    int find(int x)
    {
        if (x != par[x])
            return par[x] = find(par[x]);
        return x;
    }
    bool merge(int a, int b)
    {
        a = find(a);
        b = find(b);
        if (a == b)
            return false;
        if (sz[a] < sz[b])
            swap(a, b);
        comp--;
        par[b] = a;
        sz[a] += sz[b];
        sz[b] = 0;
        return true;
    }
    int get_sz(int a) {
        return sz[find(a)];
    }
};
```

6.2 Fenwick 2D

```
template <typename T>
class fenwick2d {
private:
    vector<vector<T>> mat;
    int N, M;
public:
    fenwick2d(int N_, int M_) : N(N_), M(M_) {
        mat.resize(N);
        for (int i = 0; i < N; i++) {
            mat[i].resize(M);
        }
    }
    inline void modify(int i, int j, T v) {
        assert(i >= 0 && j >= 0 && i < N && j < M);
        int x = i;
        while (x < N) {
            int y = j;
            while (y < M) {
                mat[x][y] += v;
                y |= (y + 1);
            }
            x |= (x + 1);
        }
    }
    inline T get(int i, int j) {
        assert(i >= 0 && j >= 0 && i < N && j < M);
        T v{};
        int x = i;
        while (x >= 0) {
            int y = j;
            while (y >= 0) {
```

```
                v += mat[x][y];
                y = (y & (y + 1)) - 1;
            }
            x = (x & (x + 1)) - 1;
        }
        return v;
    }
    inline T get(int x1, int y1, int x2, int y2) {
        assert(x1 <= x2 && y1 <= y2 && x1 >= 0 && y1 >= 0 &&
           x2 < N && y2 < M);
        T res{};
        res += get(x2, y2);
        if (x1 > 0)
            res -= get(x1 - 1, y2);
        if (y1 > 0)
            res -= get(x2, y1 - 1);
        if (x1 > 0 && y1 > 0)
            res += get(x1 - 1, y1 - 1);
        return res;
    }
};
```

6.3 Fenwick 3D

```
template <typename T>
class fenwick3d {
private:
    tensor<T, 3>*mat;
    int A, B, C;
public:
    fenwick3d(int A_, int B_, int C_) : A(A_), B(B_), C(C_) {
        mat = new tensor<T, 3>({A, B, C});
    }
    inline void modify(int i, int j, int k, T v) {
        assert(i >= 0 && j >= 0 && i < A && j < B && k >= 0
           && k < C);
        int x = i;
        while (x < A) {
            int y = j;
            while (y < B) {
                int z = k;
                while (z < C) {
                    mat->at({x, y, z}) += v;
                    z |= (z + 1);
                }
                y |= (y + 1);
            }
            x |= (x + 1);
        }
    }
    inline T get(int i, int j, int k) {
        assert(i >= 0 && j >= 0 && i < A && j < B && k >= 0
           && k < C);
        T v{};
        int x = i;
        while (x >= 0) {
            int y = j;
            while (y >= 0) {
                int z = k;
                while (z >= 0) {
                    v += mat->at({x, y, z});
                    z = (z & (z + 1)) - 1;
                }
                y = (y & (y + 1)) - 1;
            }
            x = (x & (x + 1)) - 1;
        }
        return v;
    }
    inline T get(int x1, int y1, int z1, int x2, int y2, int
       z2) {
        assert(x1 <= x2 && y1 <= y2 && x1 >= 0 && y1 >= 0 &&
           x2 < A && y2 < B && z1 <= z2 && z1 >= 0 && z2 <
           C);
        T res{};
        res += get (x2, y2, z2);
        if (x1 > 0)
            res -= get (x1 - 1, y2, z2);
        if (y1 > 0)
            res -= get (x2, y1 - 1, z2);
        if (z1 > 0)
            res -= get (x2, y2, z1 - 1);
```

```
        if (y1 > 0 && z1 > 0)
            res += get (x2, y1 - 1, z1 - 1);
        if (x1 > 0 && z1 > 0)
            res += get (x1 - 1, y2, z1 - 1);
        if (x1 > 0 && y1 > 0)
            res += get (x1 - 1, y1 - 1, z2);
        if (x1 > 0 && y1 > 0 && z1 > 0)
            res -= get (x1 - 1, y1 - 1, z1 - 1);
        return res;
    }
};
```

6.4 Fenwick Tree

```
template <typename T>
class fenwick {
private:
    vector<T> fenw;
    int N;
public:
    fenwick (): N() {}
    fenwick(const int& N_) : N(N_) {
        fenw.resize(N);
    }

    fenwick(const vector<T>& A) {
        N = static_cast<T>(A.size());
        fenw.resize(N);
        for (int i = 0; i < N; ++i) {
            modify(i, A[i]);
        }
    }

    // 1. Point update and range query
    void modify(int x, const T& v) {
        assert(x >= 0);
        while (x < N) {
            fenw[x] += v;
            x |= (x + 1);
        }
    }

    T get(int x) {
        assert(x < N);
        T v{};
        while (x >= 0) {
            v += fenw[x];
            x = (x & (x + 1)) - 1;
        }
        return v;
    }

    // 2. Range update and point query
    // Have to call modify(i, i, v) for initial vector
    void modify(int x, int y, const T& v) {
        assert(x <= y && x >= 0 && y < N);
        modify(x, v);
        modify(y + 1, -v);
    }
    // Same get method for A[x] not A[0:x]
};
```

6.5 Efficient Range Query Fenwick Tree

```
// https://ioinformatics.org/journal/v9_2015_39_44.pdf
// N.log(N) preprocessing
// update is O(log(N)) and query is O(log(N))
// Faster than lazy segment tree!
template<typename T, class F = function<T(const T&, const
   T&)>>
class range_query_fenwick {
private:
    vector<T>BIT1;
    vector<T>BIT2;
    F func;
    bool indexing; // Indexing values type: only (0 or 1)
    vector<T>A;
    T initializer; // Initial value (eg. 0 for sum, inf for
       min, -inf for max)
    int N;
```

```
    void modify1(int idx, const T& v) {
        while (idx < N) {
            BIT1[idx] = func(BIT1[idx], v);
            idx += idx & -idx;
        }
    }

    void modify2(int idx, const T& v) {
        while (idx > 0) {
            BIT2[idx] = func(BIT2[idx], v);
            idx -= idx & -idx;
        }
    }
public:
    range_query_fenwick(int N_, const T& t, const F& f, bool
       indexing_ = 0): N(N_ + 1), func(f),
       indexing(indexing_) {
        A.resize(N + 1, t);
        initializer = t;
        BIT1.resize(N + 1, t);
        BIT2.resize(N + 1, t);
    }
    int size() const {
        assert(BIT1.size() == BIT2.size() &&
           int(BIT1.size()) == N);
        return N;
    }
    const vector<T>& data1() const {
        return BIT1;
    }
    vector<T>& data1() {
        return BIT1;
    }
    const vector<T>& data2() const {
        return BIT2;
    }
    vector<T>& data2() {
        return BIT2;
    }
    range_query_fenwick(const vector<T>& V, const T& t,
       const F& f, bool indexing_ = 0): func(f),
       indexing(indexing_) {
        N = static_cast<int>(V.size()) + 1;
        initializer = t;
        A.resize(N + 1, t);
        BIT1.resize(N + 1, t);
        BIT2.resize(N + 1, t);
        for (int i = 1; i <= N; ++i) {
            A[i] = V[i - 1];
            modify1(i, V[i - 1]);
            modify2(i, V[i - 1]);
        }
    }

    T get(int l, int r) {
        if (indexing == 0) {
            l++, r++;
        }
        assert(l <= r && l >= 1 && r <= N - 1);
        int idx = l;
        T res = initializer;
        while (idx + (idx & -idx) <= r) {
            res = func(res, BIT2[idx]);
            idx += idx & -idx;
        }
        res = func(res, A[idx]);
        idx = r;
        while (idx - (idx & -idx) >= l) {
            res = func(res, BIT1[idx]);
            idx -= idx & -idx;
        }
        return res;
    }

    void modify(int p, const T& v) {
        if (indexing == 0) {
            p++;
        }
        assert(p >= 1 && p <= N - 1);
        A[p] = v;

        // Update BIT1
        int idx = p;
```

```
int ptr1 =  p - 1; // pointer at BIT2[p-1]
int ptr2 = p + 1; // pointer at BIT1[p+1]
T res = v;
while (idx < N) {
    int x = idx - (idx & -idx) + 1;
    int y = idx;
    T extra = initializer;
    if (p - 1 >= x) {
        while (ptr1 >= x) {
            res = func(res, BIT1[ptr1]);
            ptr1 -= (ptr1 & -ptr1);
        }
    }
    if (p + 1 <= y) {
        while (ptr2 < y) {
            res = func(res, BIT2[ptr2]);
            ptr2 += (ptr2 & -ptr2);
        }
        extra = func(extra, A[ptr2]);
    }
    BIT1[idx] = func(extra, res);
    idx += idx & -idx;
}

// Update BIT2
ptr1 =  p - 1; // pointer at BIT2[p-1]
ptr2 = p + 1; // pointer at BIT1[p+1]
res = v;
idx = p;
while (idx > 0) {
    int x = idx;
    int y = idx + (idx & -idx) - 1;
    y = min(y, N - 1);
    T extra = initializer;
    if (p - 1 >= x) {
        while (ptr1 > x) {
            res = func(res, BIT1[ptr1]);
            ptr1 -= (ptr1 & -ptr1);
        }
        extra = func(extra, A[ptr1]);
    }
    if (p + 1 <= y) {
        while (ptr2 <= y) {
            res = func(res, BIT2[ptr2]);
            ptr2 += (ptr2 & -ptr2);
        }
    }
    BIT2[idx] = func(extra, res);
    idx -= idx & -idx;
  }
 }
};
```

6.6 Range query, range update Fenwick Tree

```
template<typename T>
class range_fenwick {
private:
    int N;
    fenwick<T>*B1, *B2;
public:
    range_fenwick(const int &N_): N(N_) {
        B1 = new fenwick<T>(N);
        B2 = new fenwick<T>(N);
    }
    void modify(int l, int r, const T& v) {
        B1->modify(l, v);
        B1->modify(r + 1, -v);
        B2->modify(l, v * (l - 1));
        B2->modify(r + 1, -v * r);
    }
    T get(int idx) {
        return B1->get(idx) * (idx) - B2->get(idx);
    }
    T get(int l, int r) {
        return get(r) - get(l - 1);
    }
};
```

6.7 Link Cut Trees

```
const int maxn = 2e5 + 5;
int p[maxn], a[maxn], b[maxn];
vector<int>g[maxn], g2[maxn];
bool vis[maxn];
ll d[maxn];
deque<ll>pre, vec;
struct node
{
    int p, ch[2];
    ll val, sub;
    bool rev;
    int sz;
    ll lazy;
    node() {}
    node(int v) : p(-1), val(v), sub(v), rev(0), sz(1),
       lazy(0)
    {
        ch[0] = ch[1] = -1;
    }
};
vector<node>t(maxn);
void prop(int x)
{
    if (t[x].lazy)
    {
        t[x].val += t[x].lazy, t[x].sub += t[x].lazy *
           t[x].sz;
        if (t[x].ch[0] + 1) t[t[x].ch[0]].lazy += t[x].lazy;
        if (t[x].ch[1] + 1) t[t[x].ch[1]].lazy += t[x].lazy;
    }
    if (t[x].rev)
    {
        swap(t[x].ch[0], t[x].ch[1]);
        if (t[x].ch[0] + 1) t[t[x].ch[0]].rev ^= 1;
        if (t[x].ch[1] + 1) t[t[x].ch[1]].rev ^= 1;
    }
    t[x].lazy = 0, t[x].rev = 0;
}
void update(int x)
{
    t[x].sz = 1, t[x].sub = t[x].val;
    for (int i = 0; i < 2; i++) if (t[x].ch[i] + 1)
    {
        prop(t[x].ch[i]);
        t[x].sz += t[t[x].ch[i]].sz;
        t[x].sub += t[t[x].ch[i]].sub;
    }
}
bool  is_root ( int x)
{
    return t[x].p == -1 or (t[t[x].p].ch[0] != x and
       t[t[x].p].ch[1] != x);
}
void rotate(int x)
{
    int p = t [x]. p, pp = t [p]. p ;
    if (!is_root(p)) t[pp].ch[t[pp].ch[1] == p] = x;
    bool d = t[p].ch[0] == x;
    t[p].ch[!d] = t[x].ch[d], t[x].ch[d] = p;
    if (t[p].ch[!d] + 1) t[t[p].ch[!d]].p = p;
    t [x]. p = pp, t [p]. p = x;
    update(p), update(x);
}
int splay(int x)
{
    while (!is_root(x))
    {
        int p = t [x]. p, pp = t [p]. p ;
        if (!is_root(p)) prop(pp);
        prop(p), prop(x);
        if (!is_root(p)) rotate((t[pp].ch[0] == p) ^
           (t[p].ch[0] == x) ? x : p);
        rotate(x);
    }
    return prop(x), x;
}
int access(int v)
{
    int last = -1;
    for (int w = v; w + 1; update(last = w), splay(v), w =
       t[v].p)
        splay(w), t[w].ch[1] = (last == -1 ? -1 : v);
    return last;
}
```

```
void make_tree(int v, int w)
{
    t[v] = node(w);
}
int find_root(int v)
{
    access(v), prop(v);
    while (t[v].ch[0] + 1) v = t[v].ch[0], prop(v);
    return splay(v);
}
bool connected(int v, int w)
{
    access(v), access(w);
    return v == w ? true : t[v].p != -1;
}
void rootify(int v)
{
    access(v);
    t[v].rev ^= 1;
}
ll query(int v, int w)
{
    rootify(w), access(v);
    return t[v].sub;
}
void update(int v, int w, int x)
{
    rootify(w), access(v);
    t[v].lazy += x;
}
void link(int v, int w)
{
    rootify(w);
    t[w].p = v;
}
void cut(int v, int w)
{
    rootify(w), access(v);
    t[v].ch[0] = t[t[v].ch[0]].p = -1;
}
int lca(int v, int w)
{
    access(v);
    return access(w);
}
int mp[maxn];
void dfs(int u, int p)
{
    vec.push_back(u);
    int prev = 0;
    if(pre.size())
        prev = pre.back();
    pre.push_back(prev + a[u]);
    int idx = lower_bound(all(pre), b[u]) - pre.begin();
    if(idx != pre.size())
    {
        d[u] = mp[vec[idx]];
    }
    for(auto v : g2[u])
    {
        if(v != p)
        {
            dfs(v, u);
        }
    }
    pre.pop_back();
    vec.pop_back();
}
```

6.8 Intesecting Segments

```
/*
    Given 2n points representing ranges find for each
       number, number of segment intersecting with it.
    Example:    n=2 and array=[1,2,1,2]. 1 and 2 intersect
       hence answer is [1,1] for 1 and 2.
*/
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
struct segtree
{
```

```
struct node
{
    ll sum = 0;
    ll add = 0;
    void apply(int l, int r, const ll &v)
    {
        sum += (r - l + 1) * v;
        add += v;
    }
};
int n;
vector<node>tree;
static node unite(const node &a, const node &b)
{
    node c;
    c.sum = a.sum + b.sum;
    return c;
}
void pull(int x, int y)
{
    tree[x] = unite(tree[x + 1], tree[y]);
}
void push(int x, int l, int r)
{

    int m = (l + r) / 2;
    int y = x + ((m - l + 1) << 1);
    if(tree[x].add != 0)
    {
        tree[x + 1].apply(l, m, tree[x].add);
        tree[y].apply(m + 1, r, tree[x].add);
    }
    tree[x].add = 0;
}
segtree(int _n): n(_n)
{
    tree.resize(2 * n - 1);
}
void modify(int x, int l, int r, int ql, int qr, const
   ll &v)
{
    if(ql <= l && r <= qr)
    {
        tree[x].apply(l, r, v);
        return;
    }
    int m = (l + r) / 2;
    int y = x + ((m - l + 1) << 1);
    push(x, l, r);
    if(ql <= m)
        modify(x + 1, l, m, ql, qr, v);
    if(qr > m)
        modify(y, m + 1, r, ql, qr, v);
    pull(x, y);
}
void modify(int l, int r, const ll &v)
{
    modify(0, 0, n - 1, l, r, v);
}
node get(int x, int l, int r, int ql, int qr)
{
    if(ql <= l && r <= qr)
        return tree[x];
    int m = (l + r) / 2;
    int y = x + ((m - l + 1) << 1);
    push(x, l, r);
    node res;
    if(qr <= m)
        res = get(x + 1, l, m, ql, qr);
    else if(ql > m)
        res = get(y, m + 1, r, ql, qr);
    else
        res = unite(get(x + 1, l, m, ql, qr), get(y, m +
           1, r, ql, qr));
    pull(x, y);
    return res;
}
node get(int l, int r)
{
    return get(0, 0, n - 1, l, r);
}
};
int32_t main()
{
```

```
int n;
cin >> n;
segtree st(2 * n);
vector<int>a(2 * n);
int vis[n];
int ans[n];
memset(vis, -1, sizeof vis);
for (int i = 0; i < 2 * n; ++i)
{
    cin >> a[i];
    a[i]--;
}
for(int i = 0; i < 2 * n; i++)
{
    if(vis[a[i]] == -1)
    {
        vis[a[i]] = i;
        st.modify(i, i, 1);
    }
    else
    {
        st.modify(vis[a[i]], vis[a[i]], -1);
        ans[a[i]] = st.get(vis[a[i]], i).sum;
        vis[a[i]] = -1;
    }
}
reverse(begin(a), end(a));
for(int i = 0; i < 2 * n; i++)
{
    if(vis[a[i]] == -1)
    {
        vis[a[i]] = i;
        st.modify(i, i, 1);
    }
    else
    {
        st.modify(vis[a[i]], vis[a[i]], -1);
        ans[a[i]] += st.get(vis[a[i]], i).sum;
    }
}
for (int i = 0; i < n; ++i)
{
    cout << ans[i] << ' ';
}
return 0;
}
```

6.9 Inversions

```
/*
    For each index of array find number of previous elements
       greater than it, i.e.
    for each index i find number if j such that j<i and
       a[j]>a[i].
*/
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
struct segtree
{
    struct node
    {
        ll sum = 0;
        ll add = 0;
        void apply(int l, int r, const ll &v)
        {
            sum += (r - l + 1) * v;
            add += v;
        }
    };
    int n;
    vector<node>tree;
    static node unite(const node &a, const node &b)
    {
        node c;
        c.sum = a.sum + b.sum;
        return c;
    }
    void pull(int x, int y)
    {
        tree[x] = unite(tree[x + 1], tree[y]);
    }
```

```
void push(int x, int l, int r)
{

    int m = (l + r) / 2;
    int y = x + ((m - l + 1) << 1);
    if(tree[x].add != 0)
    {
        tree[x + 1].apply(l, m, tree[x].add);
        tree[y].apply(m + 1, r, tree[x].add);
    }
    tree[x].add = 0;
}
segtree(int _n): n(_n)
{
    tree.resize(2 * n - 1);
}
void modify(int x, int l, int r, int ql, int qr, const
   ll &v)
{
    if(ql <= l && r <= qr)
    {
        tree[x].apply(l, r, v);
        return;
    }
    int m = (l + r) / 2;
    int y = x + ((m - l + 1) << 1);
    push(x, l, r);
    if(ql <= m)
        modify(x + 1, l, m, ql, qr, v);
    if(qr > m)
        modify(y, m + 1, r, ql, qr, v);
    pull(x, y);
}
void modify(int l, int r, const ll &v)
{
    modify(0, 0, n - 1, l, r, v);
}
node get(int x, int l, int r, int ql, int qr)
{
    if(ql <= l && r <= qr)
        return tree[x];
    int m = (l + r) / 2;
        int y = x + ((m - l + 1) << 1);
        push(x, l, r);
        node res;
        if(qr <= m)
            res = get(x + 1, l, m, ql, qr);
        else if(ql > m)
            res = get(y, m + 1, r, ql, qr);
        else
            res = unite(get(x + 1, l, m, ql, qr), get(y, m +
               1, r, ql, qr));
        pull(x, y);
        return res;
    }
    node get(int l, int r)
    {
        return get(0, 0, n - 1, l, r);
    }
};
int32_t main()
{
    int n;
    cin >> n;
    vector<ll>a(n);
    for(int i = 0; i < n; ++i)
    {
        cin >> a[i];
        --a[i];
    }
    segtree st(n);
    int ans[n];
    for (int i = 0; i < n; ++i)
    {
        ans[i] = st.get(a[i], n - 1).sum;
        st.modify(a[i], a[i], 1);
    }
    for (int i = 0; i < n; ++i)
        cout << ans[i] << ' ';
    return 0;
}
```

6.10 Kth One

```
/*
    Find the kth(one based) one's index in a binary array.
*/
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
struct segtree
{
    struct node
    {
        ll sum = 0;
        ll add = 0;
        void apply(int l, int r, const ll &v)
        {
            sum += (r - l + 1) * v;
            add += v;
        }
        void make_data(const ll &v)
        {
            sum = v;
        }
    };
    int n;
    vector<node>tree;
    static node unite(const node &a, const node &b)
    {
        node c;
        c.sum = a.sum + b.sum;
        return c;
    }
    void pull(int x, int y)
    {
        tree[x] = unite(tree[x + 1], tree[y]);
    }
    void push(int x, int l, int r)
    {

        int m = (l + r) / 2;
        int y = x + ((m - l + 1) << 1);
        if(tree[x].add != 0)
        {
            tree[x + 1].apply(l, m, tree[x].add);
            tree[y].apply(m + 1, r, tree[x].add);
        }
        tree[x].add = 0;
    }
    void build(int x, int l, int r, const vector<ll> &v)
    {
        if(l == r)
        {
            tree[x].make_data(v[l]);
            return;
        }
        int m = (l + r) / 2;
        int y = x + ((m - l + 1) << 1);
        build(x + 1, l, m, v);
        build(y, m + 1, r, v);
        pull(x, y);
    }
    segtree(const vector<ll> &v)
    {
        n = v.size();
        tree.resize(2 * n);
        build(0, 0, n - 1, v);
    }
    void modify(int x, int l, int r, int ql, int qr, const
       ll &v)
    {
        if(ql <= l && r <= qr)
        {
            tree[x].make_data(v);
            return;
        }
        int m = (l + r) / 2;
        int y = x + ((m - l + 1) << 1);
        push(x, l, r);
        if(ql <= m)
            modify(x + 1, l, m, ql, qr, v);
        if(qr > m)
            modify(y, m + 1, r, ql, qr, v);
        pull(x, y);
    }
```

```
void modify(int l, int r, const ll &v)
{
    modify(0, 0, n - 1, l, r, v);
}
node get(int x, int l, int r, int ql, int qr)
{
    if(ql <= l && r <= qr)
        return tree[x];
    int m = (l + r) / 2;
    int y = x + ((m - l + 1) << 1);
    push(x, l, r);
    node res;
    if(qr <= m)
        res = get(x + 1, l, m, ql, qr);
    else if(ql > m)
        res = get(y, m + 1, r, ql, qr);
    else
        res = unite(get(x + 1, l, m, ql, qr), get(y, m +
            1, r, ql, qr));
    pull(x, y);
    return res;
}
node get(int l, int r)
{
    return get(0, 0, n - 1, l, r);
}
int kth_one(int x, int l, int r, int k)
{
    if(k > tree[x].sum)
        return -1;
    if(l == r)
        return l;
    int m = (l + r) / 2;
    int y = x + ((m - l + 1) << 1);
    push(x, l, r);
    if(tree[x + 1].sum >= k)
        return kth_one(x + 1, l, m, k);
    else
        return kth_one(y, m + 1, r, k - tree[x + 1].sum);
}
int kth_one(int k)
{
        return kth_one(0, 0, n - 1, k);
    }
};
int32_t main()
{
    int n, q;
    cin >> n >> q;
    vector<ll>a(n);
    for(int i = 0; i < n; ++i)
        cin >> a[i];
    segtree st(a);
    while(q--)
    {
        int u, v;
        cin >> u >> v;
        if(u == 2)
        {
            cout << st.kth_one(v + 1) << '\n';
        }
        else
        {
            st.modify(v, v, 1ll - a[v]);
            (a[v] ^= 1);
        }
    }
    return 0;
}
```

6.11 KQuery Offline

```
// Find each query (l,r,k) find the number of elements in
   range [a,b] strictly greater than k.
// Offline Segment Tree Implementation
/**
 *    author:  ritik singla
 *    created: 09.10.2020 19:59:08
**/
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
```

```
using ld = long double;
#define ff first
#define ss second
#define all(x) x.begin(), x.end()
#define rep(i,n) for(int i = 0; i < (n); i++)
#define forn(i,a,b) for(int i = (a); i <= (b); i++)
#define trav(a,x) for(auto &a : x)
typedef pair<int, int> pi;
typedef pair<ll, ll> pl;
typedef vector<int> vi;
typedef vector<ll> vl;
typedef vector<pi> vpi;
typedef vector<pl> vpl;
template <typename A, typename B>
string to_string(pair<A, B> p);

template <typename A, typename B, typename C>
string to_string(tuple<A, B, C> p);

template <typename A, typename B, typename C, typename D>
string to_string(tuple<A, B, C, D> p);

string to_string(const string &s) {
    return '"' + s + '"';
}

string to_string(const char *s) {
    return to_string((string) s);
}

string to_string(bool b) {
    return (b ? "true" : "false");
}

string to_string(vector<bool> v) {
    bool first = true;
    string res = "{";
    for (int i = 0; i < static_cast<int>(v.size()); i++)
    {
        if (!first)
        {
            res += ", ";
        }
        first = false;
        res += to_string(v[i]);
    }
    res += "}";
    return res;
}

template <size_t N>
string to_string(bitset<N> v) {
    string res = "";
    for (size_t i = 0; i < N; i++)
    {
        res += static_cast<char>('0' + v[i]);
    }
    return res;
}

template <typename A>
string to_string(A v) {
    bool first = true;
    string res = "{";
    for (const auto &x : v)
    {
        if (!first)
        {
            res += ", ";
        }
        first = false;
        res += to_string(x);
    }
    res += "}";
    return res;
}

template <typename A, typename B>
string to_string(pair<A, B> p) {
    return "(" + to_string(p.first) + ", " +
       to_string(p.second) + ")";
}

template <typename A, typename B, typename C>
```

```
string to_string(tuple<A, B, C> p) {
    return "(" + to_string(get<0>(p)) + ", " +
       to_string(get<1>(p)) + ", " + to_string(get<2>(p)) +
       ")";
}

template <typename A, typename B, typename C, typename D>
string to_string(tuple<A, B, C, D> p) {
    return "(" + to_string(get<0>(p)) + ", " +
       to_string(get<1>(p)) + ", " + to_string(get<2>(p)) +
       ", " + to_string(get<3>(p)) + ")";
}

void debug_out() {
    cerr << endl;
}

template <typename Head, typename... Tail>
void debug_out(Head H, Tail... T) {
    cerr << " " << to_string(H);
    debug_out(T...);
}

#ifdef LOCAL
#define debug(...) cerr << "[" << #__VA_ARGS__ << "]:",
   debug_out(__VA_ARGS__)
#else
#define debug(...) 42
#endif
struct segtree
{
    struct node
    {
        ll sum = 0;
        ll add = 0;
        ll mn = 0;
        ll xr = 0;
        ll mx = 0;
        void apply(int l, int r, ll v)
        {
            sum += (r - l + 1) * v;
            add += v;
            mn += v;
            mx += v;
            xr += v;
        }
};
int n;
vector<node>tree;
static node unite(const node &a, const node &b)
{
    node res;
    res.sum = a.sum + b.sum;
    res.mn = min(a.mn, b.mn);
    res.mx = max(a.mx, b.mx);
    res.xr = (a.xr ^ b.xr);
    return res;
}
void push(int x, int l, int r)
{
    int m = (l + r) >> 1;
    int y = x + ((m - l + 1) << 1);
    if (tree[x].add != 0)
    {
        tree[x + 1].apply(l, m, tree[x].add);
        tree[y].apply(m + 1, r, tree[x].add);
        tree[x].add = 0;
    }
}
void pull(int x, int y)
{
    tree[x] = unite(tree[x + 1], tree[y]);
}
template<typename T>
void build(int x, int l, int r, const vector<T> &v)
{
    if (l == r)
    {
        tree[x].apply(l, r, v[l]);
        return;
    }
    int m = (l + r) >> 1;
    int y = x + ((m - l + 1) << 1);
    build(x + 1, l, m, v);
```

```
    build(y, m + 1, r, v);
    pull(x, y);
}
void build(int x, int l, int r)
{
    if (l == r)
    {
        tree[x].apply(l, r, 0ll);
        return;
    }
    int m = (l + r) >> 1;
    int y = x + ((m - l + 1) << 1);
    build(x + 1, l, m);
    build(y, m + 1, r);
    pull(x, y);
}
node get(int x, int l, int r, int ql, int qr)
{
    if (ql <= l && r <= qr)
        return tree[x];
    int m = (l + r) >> 1;
    int y = x + ((m - l + 1) << 1);
    push(x, l, r);
    node res;
    if (qr <= m)
        res = get(x + 1, l, m, ql, qr);
    else if (ql > m)
        res = get(y, m + 1, r, ql, qr);
    else
        res = unite(get(x + 1, l, m, ql, qr), get(y, m +
            1, r, ql, qr));
    pull(x, y);
    return res;
}
template<typename T>
void modify(int x, int l, int r, int ql, int qr, const T
    &v)
{
    if (ql <= l && r <= qr)
    {
        tree[x].apply(l, r, v);
        return;
    }
    int m = (l + r) >> 1;
    int y = x + ((m - l + 1) << 1);
    push(x, l, r);
    if (ql <= m)
        modify(x + 1, l, m, ql, qr, v);
    if (qr > m)
        modify(y, m + 1, r, ql, qr, v);
    pull(x, y);
}
template<typename T>
segtree(const vector<T> &v)
{
    n = v.size();
    tree.resize(2 * n - 1);
    build(0, 0, n - 1, v);
}
segtree(int _n): n(_n)
{
    tree.resize(2 * n - 1);
    build(0, 0, n - 1);
}
template<typename T>
void modify(int l, int r, const T &v)
{
    modify(0, 0, n - 1, l, r, v);
}
node get(int l, int r)
{
    return get(0, 0, n - 1, l, r);
}
int find_first_knowingly(int x, int l, int r, const
    function<bool(const node &)> &f)
{
    if (l == r)
    {
        return l;
    }
    push(x, l, r);
    int m = (l + r) >> 1;
    int y = x + ((m - l + 1) << 1);
    int res;
```

```
    if (f(tree[x + 1]))
    {
        res = find_first_knowingly(x + 1, l, m, f);
    }
    else
    {
        res = find_first_knowingly(y, m + 1, r, f);
    }
    pull(x, y);
    return res;
}
int find_first(int x, int l, int r, int ql, int qr,
    const function<bool(const node &)> &f)
{
    if (ql <= l && r <= qr)
    {
        if (!f(tree[x]))
        {
            return -1;
        }
        return find_first_knowingly(x, l, r, f);
    }
    push(x, l, r);
    int m = (l + r) >> 1;
    int y = x + ((m - l + 1) << 1);
    int res = -1;
    if (ql <= m)
    {
        res = find_first(x + 1, l, m, ql, qr, f);
    }
    if (qr > m && res == -1)
    {
        res = find_first(y, m + 1, r, ql, qr, f);
    }
    pull(x, y);
    return res;
}
int find_last_knowingly(int x, int l, int r, const
    function<bool(const node &)> &f)
{
    if (l == r)
    {
        return l;
    }
    push(x, l, r);
    int m = (l + r) >> 1;
    int y = x + ((m - l + 1) << 1);
    int res;
    if (f(tree[y]))
    {
        res = find_last_knowingly(y, m + 1, r, f);
    }
    else
    {
        res = find_last_knowingly(x + 1, l, m, f);
    }
    pull(x, y);
    return res;
}
int find_last(int x, int l, int r, int ql, int qr, const
    function<bool(const node &)> &f)
{
    if (ql <= l && r <= qr)
    {
        if (!f(tree[x]))
        {
            return -1;
        }
        return find_last_knowingly(x, l, r, f);
    }
    push(x, l, r);
    int m = (l + r) >> 1;
    int y = x + ((m - l + 1) << 1);
    int res = -1;
    if (qr > m)
    {
        res = find_last(y, m + 1, r, ql, qr, f);
    }
    if (ql <= m && res == -1)
    {
        res = find_last(x + 1, l, m, ql, qr, f);
    }
    pull(x, y);
    return res;
```

```
    }
    int find_first(int ql, int qr, const function<bool(const
       node &)> &f)
    {
        assert(0 <= ql && ql <= qr && qr <= n - 1);
        return find_first(0, 0, n - 1, ql, qr, f);
    }
    int find_last(int ql, int qr, const function<bool(const
       node &)> &f)
    {
        assert(0 <= ql && ql <= qr && qr <= n - 1);
        return find_last(0, 0, n - 1, ql, qr, f);
    }
};
int32_t main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);
    int n;
    cin >> n;
    ll a[n];
    rep(i, n)
    cin >> a[i];
    int q;
    cin >> q;
    vi b(n, 1);
    segtree st(b);
    vector<pair<ll, pi>>query(q);
    vi p(n);
    iota(all(p), 0);
    sort(all(p), [&](int i, int j)
    {
        return a[i] < a[j];
    });
    vi p2(q);
    rep(i, q)
    {
        cin >> query[i].ss.ff >> query[i].ss.ss >>
           query[i].ff;
    }
    iota(all(p2), 0);
    sort(all(p2), [&](int i, int j)
    {
        return query[i].ff < query[j].ff;
    });
    int idx = 0;
    vi ans(q);
    rep(i, q)
    {
        int idx2 = p2[i];
        ll k = query[idx2].ff;
        while (idx < n && a[p[idx]] <= k)
        {
            st.modify(p[idx], p[idx], -1);
            idx++;
        }
        ans[idx2] = st.get(query[idx2].ss.ff - 1,
           query[idx2].ss.ss - 1).sum;
    }
    rep(i, q)
    cout << ans[i] << '\n';
    return 0;
}
```

6.12 KQuery Online

```
// Find each query (l,r,k) find the number of elements in
   range [a,b] strictly greater than k.
// Online Segment Tree Implementation
#include <bits/stdc++.h>
using namespace std;
struct segtree
{
    struct node
    {
        vector<int> vec;
    };
    int n;
    vector<node>tree;
    static node unite(const node &a, const node &b)
    {
        node res;
```

```
    res.vec.resize(a.vec.size() + b.vec.size());
    merge(all(a.vec), all(b.vec), res.vec.begin());
    return res;
}
void pull(int x, int y)
{
    tree[x] = unite(tree[x + 1], tree[y]);
}
template<typename T>
void build(int x, int l, int r, const vector<T> &v)
{
    if (l == r)
    {
        tree[x].vec.emplace_back(v[l]);
        return;
    }
    int m = (l + r) >> 1;
    int y = x + ((m - l + 1) << 1);
    build(x + 1, l, m, v);
    build(y, m + 1, r, v);
    pull(x, y);
}
template<typename T>
segtree(const vector<T> &v)
{
    n = v.size();
    tree.resize(2 * n - 1);
    build(0, 0, n - 1, v);
}
int count(int x, int l, int r, int ql, int qr, int k)
{
    if (l > qr || r < ql)
        return 0;
    if (ql <= l && r <= qr)
    {
        int ans = tree[x].vec.end() -
           upper_bound(all(tree[x].vec), k);
        return ans;
    }
    int m = (l + r) >> 1;
    int y = x + ((m - l + 1) << 1);
    return count(x + 1, l, m, ql, qr, k) + count(y, m +
           1, r, ql, qr, k);
    }
    int count(int l, int r, int k)
    {
        return count(0, 0, n - 1, l, r, k);
    }
};
int32_t main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);
    int n;
    cin >> n;
    vi a(n);
    rep(i, n)
    cin >> a[i];
    int q;
    cin >> q;
    segtree st(a);
    vi ans(q);
    int prev_ans = 0;
    rep(i, q)
    {
        int x, y, z;
        cin >> x >> y >> z;
        x ^= prev_ans;
        y ^= prev_ans;
        z ^= prev_ans;
        if (x < 1)
            x = 1;
        if (y > n)
            y = n;
        if (x > y)
        {
            ans[i] = 0;
            prev_ans = 0;
            continue;
        }
        ans[i] = st.count(x-1, y-1, z);
        prev_ans = ans[i];
    }
    rep(i, q)
```

```
    cout << ans[i] << '\n';
    return 0;
}
```

6.13 Lower Bound in Range

```
/*
    Find the lower bound of the number in a given range.
*/
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
struct segtree
{
    struct node
    {
        ll add = 0;
        ll mx = 0;
        void apply(int l, int r, const ll &v)
        {
            add += v;
            mx += v;
        }
    };
    int n;
    vector<node>tree;
    static node unite(const node &a, const node &b)
    {
        node c;
        c.mx = max(a.mx, b.mx);
        return c;
    }
    void pull(int x, int y)
    {
        tree[x] = unite(tree[x + 1], tree[y]);
    }
    void push(int x, int l, int r)
    {

        int m = (l + r) / 2;
        int y = x + ((m - l + 1) << 1);
        if(tree[x].add != 0)
        {
            tree[x + 1].apply(l, m, tree[x].add);
            tree[y].apply(m + 1, r, tree[x].add);
        }
        tree[x].add = 0;
    }
    void build(int x, int l, int r, const vector<ll> &v)
    {
        if(l == r)
        {
            tree[x].apply(l, r, v[l]);
            return;
        }
        int m = (l + r) / 2;
        int y = x + ((m - l + 1) << 1);
        build(x + 1, l, m, v);
        build(y, m + 1, r, v);
        pull(x, y);
    }
    segtree(const vector<ll> &v)
    {
        n = v.size();
        tree.resize(2 * n);
        build(0, 0, n - 1, v);
    }
    void modify(int x, int l, int r, int ql, int qr, const
       ll &v)
    {
        if(ql <= l && r <= qr)
        {
            tree[x].apply(l, r, v);
            return;
        }
        int m = (l + r) / 2;
        int y = x + ((m - l + 1) << 1);
        push(x, l, r);
        if(ql <= m)
            modify(x + 1, l, m, ql, qr, v);
        if(qr > m)
            modify(y, m + 1, r, ql, qr, v);
```

```
    pull(x, y);
}
void modify(int l, int r, const ll &v)
{
    modify(0, 0, n - 1, l, r, v);
}
node get(int x, int l, int r, int ql, int qr)
{
    if(ql <= l && r <= qr)
        return tree[x];
    int m = (l + r) / 2;
    int y = x + ((m - l + 1) << 1);
    push(x, l, r);
    node res;
    if(qr <= m)
        res = get(x + 1, l, m, ql, qr);
    else if(ql > m)
        res = get(y, m + 1, r, ql, qr);
    else
        res = unite(get(x + 1, l, m, ql, qr), get(y, m +
            1, r, ql, qr));
    pull(x, y);
    return res;
}
node get(int l, int r)
{
    return get(0, 0, n - 1, l, r);
}
int find_first_knowingly(int x, int l, int r, const
    function<bool(const node &)> &f)
{
    if(l == r)
        return l;
    push(x, l, r);
    int m = (l + r) / 2;
    int y = x + ((m - l + 1) << 1);
    int res;
    if (f(tree[x + 1]))
    {
        res = find_first_knowingly(x + 1, l, m, f);
    }
    else
    {
        res = find_first_knowingly(y, m + 1, r, f);
    }
    pull(x, y);
    return res;
}
// Return first index where condition is true for the
    first time.
int find_first(int x, int l, int r, int ql, int qr,
    const function<bool(const node &)> &f)
{
    if(ql <= l && r <= qr)
    {
        if(!f(tree[x]))
            return -1;
        return find_first_knowingly(x, l, r, f);
    }
    push(x, l, r);
    int m = (l + r) / 2;
    int y = x + ((m - l + 1) << 1);
    int res = -1;
    if(ql <= m)
        res = find_first(x + 1, l, m, ql, qr, f);
    if(qr > m && res == -1)
        res = find_first(y, m + 1, r, ql, qr, f);
    pull(x, y);
    return res;
}
int find_first(int l, int r, const function<bool(const
    node &)> &f)
{
    return find_first(0, 0, n - 1, 0, n - 1, f);
}
};
int32_t main()
{
    int n, q;
    cin >> n >> q;
    vector<ll>a(n);
    for(int i = 0; i < n; ++i)
        cin >> a[i];
    segtree st(a);
```

```
    while(q--)
    {
        int t;
        cin >> t;
        if(t == 1)
        {
            int u;
            ll v;
            cin >> u >> v;
            st.modify(u, u, v - a[u]);
            a[u] = v;
        }
        else
        {
            ll v;
            cin >> v;
            function<bool(const segtree::node &)>f =
               [&](const segtree::node & a)
            {
                return a.mx >= v;
            };
            cout << st.find_first(0, n - 1, f) << '\n';
        }
    }
    return 0;
}
```

6.14 Matrix Multiplication

```
/*
    Given n 2X2 matrices, print the matrix multiplication of
       Ai,....,Aj modulo m
*/
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
int mod;
inline int add(int x, int y)
{
    x += y;
    if(x >= mod)
        x -= mod;
    return x;
}
inline int mul(int x, int y)
{
    return (x * y % mod);
}
struct segtree
{
    struct node
    {
        int matrix[2][2];
        make_data(int x, int a[][2][2])
        {
            rep(i, 2)
            rep(j, 2)
            matrix[i][j] = a[x][i][j];
        }
    };
    int n;
    vector<node>tree;
    static node unite(const node &a, const node &b)
    {
        node c;
        c.matrix[0][0] = add(mul(a.matrix[0][0],
           b.matrix[0][0]), mul(a.matrix[0][1],
           b.matrix[1][0]));
        c.matrix[0][1] = add(mul(a.matrix[0][0],
           b.matrix[0][1] ), mul(a.matrix[0][1],
           b.matrix[1][1]));
        c.matrix[1][0] = add(mul(a.matrix[1][0],
           b.matrix[0][0]), mul(a.matrix[1][1],
           b.matrix[1][0]));
        c.matrix[1][1] = add(mul(a.matrix[1][0],
           b.matrix[0][1] ), mul(a.matrix[1][1],
           b.matrix[1][1]));
        return c;
    }
    void pull(int x, int y)
    {
        tree[x] = unite(tree[x + 1], tree[y]);
```

```
    }
    void build(int x, int l, int r, int v[][2][2])
    {
        if(l == r)
        {
            tree[x].make_data(l, v);
            return;
        }
        int m = (l + r) / 2;
        int y = x + ((m - l + 1) << 1);
        build(x + 1, l, m, v);
        build(y, m + 1, r, v);
        pull(x, y);
    }
    segtree(int _n, int a[][2][2])
    {
        n = _n;
        tree.resize(2 * n - 1);
        build(0, 0, n - 1, a);
    }
    node get(int x, int l, int r, int ql, int qr)
    {
        if(ql <= l && r <= qr)
            return tree[x];
        int m = (l + r) / 2;
        int y = x + ((m - l + 1) << 1);
        node res;
        if(qr <= m)
            res = get(x + 1, l, m, ql, qr);
        else if(ql > m)
            res = get(y, m + 1, r, ql, qr);
        else
            res = unite(get(x + 1, l, m, ql, qr), get(y, m +
                1, r, ql, qr));
        pull(x, y);
        return res;
    }
    node get(int l, int r)
    {
        return get(0, 0, n - 1, l, r);
    }
};
```

```
int32_t main()
{
    int  n, q;
    cin >> mod >> n >> q;
    int a[n][2][2];
    rep(i, n)
    {
        rep(j, 2)
        rep(k, 2)
        {
            cin >> a[i][j][k];
            a[i][j][k] %= mod;
        }
    }
    segtree st(n, a);
    rep(i, q)
    {
        int l, r;
        cin >> l >> r, --l, --r;
        segtree::node res = st.get(l, r);
        rep(j, 2)
        rep(k, 2)
        cout << res.matrix[j][k] << " \n"[k == 1];
        cout << '\n';
    }
    return 0;
}
```

6.15 Merge Sort Tree

```
/*  Find number of elements in the range greater than x
    Construction: O(NlogN)
    Space: O(NlogN)
    Each Query: O(NlogNlogN)
*/
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
#define all(x) x->begin(), x->end()
#define rep(i,n) for(int i=0;i<(n);i++)
```

```
struct segtree
{
    struct node
    {
        vector<ll> *vec;
    };
    int n;
    vector<node>tree;
    static node unite(const node &a, const node &b)
    {
        node res;
        int l = a.vec->size();
        int r = b.vec->size();
        res.vec = new vector<ll>(l + r);
        int i = 0, j = 0, idx = 0;
        while(i < l && j < r)
        {
            if(a.vec->at(i) <= b.vec->at(j))
                res.vec->at(idx++) = a.vec->at(i++);
            else if(a.vec->at(i) > b.vec->at(j))
                res.vec->at(idx++) = b.vec->at(j++);
        }
        while(i < l)
            res.vec->at(idx++) = a.vec->at(i++);
        while(j < r)
            res.vec->at(idx++) = b.vec->at(j++);
        return res;
    }
    void pull(int x)
    {
        tree[x] = unite(tree[2 * x + 1], tree[2 * x + 2]);
    }
    void build(int x, int l, int r, const vector<ll> &v)
    {
        if(l == r)
        {
            tree[x].vec = new vector<ll> {v[l]};
            return;
        }
        int m = (l + r) / 2;
        build(2 * x + 1, l, m, v);
        build(2 * x + 2, m + 1, r, v);
        pull(x);
    }
    int get(int x, int l, int r, int ql, int qr, const ll &v)
    {
        if(l > r || ql > qr || qr < l || ql > r)
            return 0;
        if(l == ql && r == qr)
        {
            int idx = upper_bound(all(tree[x].vec), v) -
               tree[x].vec->begin();
            return (tree[x].vec->size() - idx);
        }
        int m = (l + r) / 2;
        return get(2 * x + 1, l, m, ql, min(qr, m), v) +
           get(2 * x + 2, m + 1, r, max(ql, m + 1), qr, v);
    }
    segtree(const vector<ll> &v)
    {
        n = v.size();
        tree.resize(4 * n);
        build(0, 0, n - 1, v);
    }
    int get(int l, int r, const ll &v)
    {
        return get(0, 0, n - 1, l, r, v);
    }
};
int main()
{
    int n;
    cin >> n;
    vector<ll>arr(n);
    rep(i, n)
    cin >> arr[i];
    segtree st(arr);
    int q;
    cin >> q;
    ll ans = 0;
    rep(i, q)
    {
        int a, b, c;
        cin >> a >> b >> c;
```

```
        a = (a ^ ans);
        b = (b ^ ans);
        c = (c ^ ans);
        if(a < 1)
            a = 1;
        if(b > n)
            b = n;
        if(a > b)
            ans = 0;
        else
            ans = st.get(a - 1, b - 1, c);
        cout << ans << "\n";
    }
    return 0;
}
```

6.16 Nested Segments

```
/*
    Given 2n points representing ranges find for each
       number, number of segment nested within it.
    Example:    n=2 and array=[1,2,2,1]. 2 nested in 1 and
       hence answer is [1,0] for 1 and 2.
*/
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
struct segtree
{
    struct node
    {
        ll sum = 0;
        ll add = 0;
        void apply(int l, int r, const ll &v)
        {
            sum += (r - l + 1) * v;
            add += v;
        }
    };
    int n;
    vector<node>tree;
    static node unite(const node &a, const node &b)
    {
        node c;
        c.sum = a.sum + b.sum;
        return c;
    }
    void pull(int x, int y)
    {
        tree[x] = unite(tree[x + 1], tree[y]);
    }
    void push(int x, int l, int r)
    {

        int m = (l + r) / 2;
        int y = x + ((m - l + 1) << 1);
        if(tree[x].add != 0)
        {
            tree[x + 1].apply(l, m, tree[x].add);
            tree[y].apply(m + 1, r, tree[x].add);
        }
        tree[x].add = 0;
    }
    segtree(int _n): n(_n)
    {
        tree.resize(2 * n - 1);
    }
    void modify(int x, int l, int r, int ql, int qr, const
       ll &v)
    {
        if(ql <= l && r <= qr)
        {
            tree[x].apply(l, r, v);
            return;
        }
        int m = (l + r) / 2;
        int y = x + ((m - l + 1) << 1);
        push(x, l, r);
        if(ql <= m)
            modify(x + 1, l, m, ql, qr, v);
        if(qr > m)
            modify(y, m + 1, r, ql, qr, v);
```

```
        pull(x, y);
    }
    void modify(int l, int r, const ll &v)
    {
        modify(0, 0, n - 1, l, r, v);
    }
    node get(int x, int l, int r, int ql, int qr)
    {
        if(ql <= l && r <= qr)
            return tree[x];
        int m = (l + r) / 2;
        int y = x + ((m - l + 1) << 1);
        push(x, l, r);
        node res;
        if(qr <= m)
            res = get(x + 1, l, m, ql, qr);
        else if(ql > m)
            res = get(y, m + 1, r, ql, qr);
        else
            res = unite(get(x + 1, l, m, ql, qr), get(y, m +
                1, r, ql, qr));
        pull(x, y);
        return res;
    }
    node get(int l, int r)
    {
        return get(0, 0, n - 1, l, r);
    }
};
int32_t main()
{
    int n;
    cin >> n;
    segtree st(2 * n);
    vector<int>a(2 * n);
    int vis[n];
    int ans[n];
    memset(vis, -1, sizeof vis);
    for (int i = 0; i < 2 * n; ++i)
    {
        cin >> a[i];
        a[i]--;
    }
    for(int i = 0; i < 2 * n; i++)
    {
        if(vis[a[i]] == -1)
            vis[a[i]] = i;
        else
        {
            ans[a[i]] = st.get(vis[a[i]], i).sum;
            st.modify(vis[a[i]], vis[a[i]], 1);
        }
    }
    for (int i = 0; i < n; ++i)
    {
        cout << ans[i] << ' ';
    }
    return 0;
}
```

6.17 Number of Distinct in range

```
/*
    Given array of integers of small value(a[i]<=40),
    find the number of distinct elements in the range[a,b].
*/
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
const int maxn = 41;
struct segtree
{
    struct node
    {
        bool sum[maxn] = {0};
    };
    int n;
    vector<node>tree;
    static node unite(const node &a, const node &b)
    {
        node c;
        forn(i, 1, 40)
```

```
    c.sum[i] = (a.sum[i] | b.sum[i]);
    return c;
}
void pull(int x, int y)
{
    tree[x] = unite(tree[x + 1], tree[y]);
}
void build(int x, int l, int r, const vector<int> &a)
{
    if(l == r)
    {
        tree[x].sum[a[l]] = 1;
        return;
    }
    int m = (l + r) / 2;
    int y = x + ((m - l + 1) << 1);
    build(x + 1, l, m, a);
    build(y, m + 1, r, a);
    pull(x, y);
}
segtree(const vector<int> &a)
{
    n = a.size();
    tree.resize(2 * n - 1);
    build(0, 0, n - 1, a);
}
void modify(int x, int l, int r, int ql, int qr, int v,
   bool rem)
{
    if(l == r)
    {
        if(rem)
            tree[x].sum[v] = 0;
        else
            tree[x].sum[v] = 1;
        return;
    }
    int m = (l + r) / 2;
    int y = x + ((m - l + 1) << 1);
    if(ql <= m)
        modify(x + 1, l, m, ql, qr, v, rem);
    if(qr > m)
        modify(y, m + 1, r, ql, qr, v, rem);
    pull(x, y);
}
void modify(int l, int r, int v, bool rem)
{
    modify(0, 0, n - 1, l, r, v, rem);
}
void get(int x, int l, int r, int ql, int qr, bool ans[])
{
    if(ql <= l && r <= qr)
    {
        forn(i, 1, 40)
        ans[i] |= (tree[x].sum[i]);
        return;
    }
    int m = (l + r) / 2;
    int y = x + ((m - l + 1) << 1);
    if(qr <= m)
        get(x + 1, l, m, ql, qr, ans);
    else if(ql > m)
        get(y, m + 1, r, ql, qr, ans);
    else
    {
        get(x + 1, l, m, ql, qr, ans);
        get(y, m + 1, r, ql, qr, ans);
    }
}
void get(int l, int r, bool ans[])
{
    return get(0, 0, n - 1, l, r, ans);
}
};
int32_t main()
{
    int  n, q;
    cin >> n >> q;
    vector<int>a(n);
    rep(i, n)
    {
        cin >> a[i];
    }
    segtree st(a);
```

```
    rep(i, q)
    {
        int t;
        cin >> t;
        if(t == 1)
        {
            int l, r;
            cin >> l >> r, --l, --r;
            bool ans[41];
            memset(ans, 0, sizeof ans);
            st.get(l, r, ans);
            int cnt = 0;
            forn(i, 1, 40)
            cnt += (ans[i] == 1);
            cout << cnt << "\n";
        }
        else
        {
            int l, r;
            cin >> l >> r, --l;
            st.modify(l, l, a[l], true);
            st.modify(l, l, r, false);
            a[l] = r;
        }
    }
    return 0;
}
```

6.18 Number of min and their count in range

```
// Find the minimum element in the range and number of times
   it occurs.
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
struct segtree
{
    struct node
    {
        ll add = 0;
        ll mn = 0;
        ll cnt = 1;
        void apply(int l, int r, ll v)
        {
            add += v;
            mn += v;
        }
};
int n;
vector<node>tree;
static node unite(const node &a, const node &b)
{
    node c;
    c.mn = min(a.mn, b.mn);
    if(a.mn < b.mn)
        c.cnt = a.cnt;
    else if(b.mn < a.mn)
        c.cnt = b.cnt;
    else
        c.cnt = a.cnt + b.cnt;
    return c;
}
void pull(int x, int y)
{
    tree[x] = unite(tree[x + 1], tree[y]);
}
void push(int x, int l, int r)
{
    if(tree[x].add != 0)
    {
        int m = (l + r) / 2;
        int y = x + ((m - l + 1) << 1);
        tree[x + 1].apply(l, m, tree[x].add);
        tree[y].apply(m + 1, r, tree[x].add);
    }
    tree[x].add = 0;
}
void build(int x, int l, int r, const vector<ll> &v)
{
    if(l == r)
    {
        tree[x].apply(l, r, v[l]);
```

```
        return;
    }
    int m = (l + r) / 2;
    int y = x + ((m - l + 1) << 1);
    build(x + 1, l, m, v);
    build(y, m + 1, r, v);
    pull(x, y);
}
segtree(const vector<ll> &v)
{
    n = v.size();
    tree.resize(2 * n);
    build(0, 0, n - 1, v);
}
void modify(int x, int l, int r, int ql, int qr, const
   ll &v)
{
    if(ql <= l && r <= qr)
    {
        tree[x].apply(l, r, v);
        return;
    }
    int m = (l + r) / 2;
    int y = x + ((m - l + 1) << 1);
    if(ql <= m)
        modify(x + 1, l, m, ql, qr, v);
    if(qr > m)
        modify(y, m + 1, r, ql, qr, v);
    pull(x, y);
}
void modify(int l, int r, const ll &v)
{
    modify(0, 0, n - 1, l, r, v);
}
node get(int x, int l, int r, int ql, int qr)
{
    if(ql <= l && r <= qr)
        return tree[x];
    int m = (l + r) / 2;
    int y = x + ((m - l + 1) << 1);
    push(x, l, r);
    node res;
    if(qr <= m)
        res = get(x + 1, l, m, ql, qr);
    else if(ql > m)
        res = get(y, m + 1, r, ql, qr);
    else
        res = unite(get(x + 1, l, m, ql, qr), get(y, m +
            1, r, ql, qr));
    pull(x, y);
    return res;
}
node get(int l, int r)
{
    return get(0, 0, n - 1, l, r);
}
};
int32_t main()
{
    int n, q;
    cin >> n >> q;
    vector<ll>a(n);
    for(int i = 0; i < n; ++i)
        cin >> a[i];
    segtree st(a);
    while(q--)
    {
        int t, u, v;
        cin >> t >> u >> v;
        if(t == 1)
        {
            st.modify(u, u, v - a[u]);
            a[u] = v;
        }
        else
        {
            segtree::node x = st.get(u, v - 1);
            cout << x.mn << ' ' << x.cnt << '\n';
        }
    }
    return 0;
}
```

6.19 Maximum Subarray sum with modifications

```
/*Find the maximum subarray sum with modifications.*/
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
struct segtree
{
    struct node
    {
        ll sum = 0;
        ll add = 0;
        ll mn = 0;
        ll pref, suf, ans;
        void apply(int l, int r, const ll &v)
        {
            sum += (r - l + 1) * v;
            add += v;
            mn += v;
        }
        void make_data(const ll &v)
        {
            sum = v;
            pref = suf = ans = max(0ll, v);
        }
    };
    int n;
    vector<node>tree;
    static node unite(const node &a, const node &b)
    {
        node c;
        c.sum = a.sum + b.sum;
        c.mn = min(a.mn, b.mn);
        c.pref = max(a.pref, a.sum + b.pref);
        c.suf = max(b.suf, a.suf + b.sum);
        c.ans = max({a.ans, b.ans, a.suf + b.pref});
        return c;
    }
    void pull(int x, int y)
    {
        tree[x] = unite(tree[x + 1], tree[y]);
    }
    void push(int x, int l, int r)
    {

        int m = (l + r) / 2;
        int y = x + ((m - l + 1) << 1);
        if(tree[x].add != 0)
        {
            tree[x + 1].apply(l, m, tree[x].add);
            tree[y].apply(m + 1, r, tree[x].add);
        }
        tree[x].add = 0;
    }
    void build(int x, int l, int r, const vector<ll> &v)
    {
        if(l == r)
        {
            tree[x].make_data(v[l]);
            return;
        }
        int m = (l + r) / 2;
        int y = x + ((m - l + 1) << 1);
        build(x + 1, l, m, v);
        build(y, m + 1, r, v);
        pull(x, y);
    }
    segtree(const vector<ll> &v)
    {
        n = v.size();
        tree.resize(2 * n);
        build(0, 0, n - 1, v);
    }
    void modify(int x, int l, int r, int ql, int qr, const
       ll &v)
    {
        if(ql <= l && r <= qr)
        {
            tree[x].make_data(v);
            return;
        }
        int m = (l + r) / 2;
        int y = x + ((m - l + 1) << 1);
        if(ql <= m)
```

```
            modify(x + 1, l, m, ql, qr, v);
        if(qr > m)
            modify(y, m + 1, r, ql, qr, v);
        pull(x, y);
    }
    void modify(int l, int r, const ll &v)
    {
        modify(0, 0, n - 1, l, r, v);
    }
    node get(int x, int l, int r, int ql, int qr)
    {
        if(ql <= l && r <= qr)
            return tree[x];
        int m = (l + r) / 2;
        int y = x + ((m - l + 1) << 1);
        node res;
        if(qr <= m)
            res = get(x + 1, l, m, ql, qr);
        else if(ql > m)
            res = get(y, m + 1, r, ql, qr);
        else
            res = unite(get(x + 1, l, m, ql, qr), get(y, m +
                1, r, ql, qr));
        pull(x, y);
        return res;
    }
    node get(int l, int r)
    {
        return get(0, 0, n - 1, l, r);
    }
};
int32_t main()
{
    int n, q;
    cin >> n >> q;
    vector<ll>a(n);
    for(int i = 0; i < n; ++i)
        cin >> a[i];
    segtree st(a);
    while(q--)
    {
        int u, v;
        cin >> u >> v;
        cout << st.get(0, n - 1).ans << '\n';
        st.modify(u, u, v);
    }
    cout << st.get(0, n - 1).ans << '\n';
    return 0;
}
```

6.20 Segtree BFS

```
#include<bits/stdc++.h>
using namespace std;
using ll = long long;
constexpr int maxn = 5e5 + 5;
ll lz[maxn], st[maxn];
int n;
void apply(int x, ll v, int sz)
{
    st[x] += v * sz;
    lz[x] += v;
}
void push(int l, int r, int x)
{
    int m = (l + r) >> 1;
    if(lz[x] != 0)
    {
        apply(2 * x + 1, lz[x], m - l + 1);
        apply(2 * x + 2, lz[x], r - m);
        lz[x] = 0;
    }
}
void pull(int x)
{
    st[x] = st[2 * x + 1] + st[2 * x + 2];
}
void modify(int ql, int qr, ll v, int x = 0, int l = 0, int
   r = n - 1)
{
    if(ql <= l && r <= qr)
    {
```

```
        apply(x, v, r - l + 1);
        return;
    }
    push(l, r, x);
    int m = (l + r) >> 1;
    if(ql <= m)
        modify(ql, qr, v, 2 * x + 1, l, m);
    if(qr > m)
        modify(ql, qr, v, 2 * x + 2, m + 1, r);
    pull(x);
}
ll get(int ql, int qr, int x = 0, int l = 0, int r = n - 1)
{
    if(ql <= l && r <= qr)
    {
        return st[x];
    }
    push(l, r, x);
    int m = (l + r) >> 1;
    ll res;
    if(qr <= m)
        res = get(ql, qr, 2 * x + 1, l, m);
    else if(ql > m)
        res = get(ql, qr, 2 * x + 2, m + 1, r);
    else
        res = get(ql, qr, 2 * x + 1, l, m) + get(ql, qr, 2 *
            x + 2, m + 1, r);
    pull(x);
    return res;
}
int32_t main()
{
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int tt;
    cin >> tt;
    while(tt--)
    {
        int q;
        cin >> n >> q;
        memset(st, 0, sizeof st);
        memset(lz, 0, sizeof lz);
        for(int i = 0; i < q; i++)
        {
            bool t;
            cin >> t;
            if(t)
            {
                int x, y;
                cin >> x >> y;
                cout << get(x - 1, y - 1) << '\n';
            }
            else
            {
                int x, y;
                ll v;
                cin >> x >> y;
                cin >> v;
                modify(x - 1, y - 1, v);
            }
        }
    }
}
```

6.21 Segtree DFS

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
struct segtree
{
    struct node
    {
        ll sum = 0;
        ll add = 0;
        void apply(int l, int r, ll v)
        {
            sum += (r - l + 1) * v;
            add += v;
        }
    };
    int n;
```

```
vector<node>tree;
node unite(const node &a, const node &b) const
{
    node res;
    res.sum = a.sum + b.sum;
    return res;
}
void push(int x, int l, int r)
{
    int m = (l + r) >> 1;
    int y = x + ((m - l + 1) << 1);
    if(tree[x].add != 0)
    {
        tree[x + 1].apply(l, m, tree[x].add);
        tree[y].apply(m + 1, r, tree[x].add);
        tree[x].add = 0;
    }
    return;
}
void pull(int x, int y)
{
    tree[x] = unite(tree[x + 1], tree[y]);
}
template<typename T>
void build(int x, int l, int r, const vector<T> &v)
{
    if(l == r)
    {
        tree[x].apply(l, r, v[l]);
        return;
    }
    int m = (l + r) >> 1;
    int y = x + ((m - l + 1) << 1);
    build(x + 1, l, m, v);
    build(y, m + 1, r, v);
    pull(x, y);
}
void build(int x, int l, int r)
{
    if (l == r)
    {
        tree[x].apply(l, r, 0ll);
        return;
    }
    int m = (l + r) >> 1;
    int y = x + ((m - l + 1) << 1);
    build(x + 1, l, m);
    build(y, m + 1, r);
    pull(x, y);
}
node get(int x, int l, int r, int ql, int qr)
{
    if(ql <= l && r <= qr)
        return tree[x];
    int m = (l + r) >> 1;
    int y = x + ((m - l + 1) << 1);
    push(x, l, r);
    node res;
    if(qr <= m)
        res = get(x + 1, l, m, ql, qr);
    else if(ql > m)
        res = get(y, m + 1, r, ql, qr);
    else
        res = unite(get(x + 1, l, m, ql, qr), get(y, m +
            1, r, ql, qr));
    pull(x, y);
    return res;
}
template<typename T>
void modify(int x, int l, int r, int ql, int qr, const T
    &v)
{
    if(ql <= l && r <= qr)
    {
        tree[x].apply(l, r, v);
        return;
    }
    int m = (l + r) >> 1;
    int y = x + ((m - l + 1) << 1);
    push(x, l, r);
    if(ql <= m)
        modify(x + 1, l, m, ql, qr, v);
    if(qr > m)
        modify(y, m + 1, r, ql, qr, v);
```

```
    pull(x, y);
}
template<typename T>
segtree(const vector<T> &v)
{
    n = v.size();
    tree.resize(2 * n - 1);
    build(0, 0, n - 1, v);
}
segtree(int _n): n(_n)
{
    tree.resize(2 * n - 1);
    build(0, 0, n - 1);
}
node get(int l, int r)
{
    return get(0, 0, n - 1, l, r);
}
template<typename T>
void modify(int l, int r, const T &v)
{
    modify(0, 0, n - 1, l, r, v);
}
// Find first element where condition is true.
int find_first_knowingly(int x, int l, int r, const
   function<bool(const node &)> &f)
{
    if (l == r)
    {
        return l;
    }
    push(x, l, r);
    int m = (l + r) >> 1;
    int y = x + ((m - l + 1) << 1);
    int res;
    if (f(tree[x + 1]))
    {
        res = find_first_knowingly(x + 1, l, m, f);
    }
    else
    {
        res = find_first_knowingly(y, m + 1, r, f);
    }
    pull(x, y);
    return res;
}
int find_first(int x, int l, int r, int ql, int qr,
   const function<bool(const node &)> &f)
{
    if (ql <= l && r <= qr)
    {
        if (!f(tree[x]))
        {
            return -1;
        }
        return find_first_knowingly(x, l, r, f);
    }
    push(x, l, r);
    int m = (l + r) >> 1;
    int y = x + ((m - l + 1) << 1);
    int res = -1;
    if (ql <= m)
    {
        res = find_first(x + 1, l, m, ql, qr, f);
    }
    if (qr > m && res == -1)
    {
        res = find_first(y, m + 1, r, ql, qr, f);
    }
    pull(x, y);
    return res;
}
//  Find last element where condition is true.
int find_last_knowingly(int x, int l, int r, const
   function<bool(const node &)> &f)
{
    if (l == r)
    {
        return l;
    }
    push(x, l, r);
    int m = (l + r) >> 1;
    int y = x + ((m - l + 1) << 1);
    int res;
```

```
    if (f(tree[y]))
    {
        res = find_last_knowingly(y, m + 1, r, f);
    }
    else
    {
        res = find_last_knowingly(x + 1, l, m, f);
    }
    pull(x, y);
    return res;
}
int find_last(int x, int l, int r, int ql, int qr, const
    function<bool(const node &)> &f)
{
    if (ql <= l && r <= qr)
    {
        if (!f(tree[x]))
        {
            return -1;
        }
        return find_last_knowingly(x, l, r, f);
    }
    push(x, l, r);
    int m = (l + r) >> 1;
    int y = x + ((m - l + 1) << 1);
    int res = -1;
    if (qr > m)
    {
        res = find_last(y, m + 1, r, ql, qr, f);
    }
    if (ql <= m && res == -1)
    {
        res = find_last(x + 1, l, m, ql, qr, f);
    }
    pull(x, y);
    return res;
}
int find_first(int ql, int qr, const function<bool(const
    node &)> &f)
{
    assert(0 <= ql && ql <= qr && qr <= n - 1);
    return find_first(0, 0, n - 1, ql, qr, f);
}
int find_last(int ql, int qr, const function<bool(const
    node &)> &f)
{
    assert(0 <= ql && ql <= qr && qr <= n - 1);
    return find_last(0, 0, n - 1, ql, qr, f);
}
};
int32_t main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);
    int tt;
    cin >> tt;
    while(tt--)
    {
        int n, q;
        cin >> n >> q;
        segtree st(n);
        while(q--)
        {
            bool t;
            cin >> t;
            if(t)
            {
                int u, v;
                cin >> u >> v;
                u--;
                v--;
                cout << st.get(u, v).sum << "\n";
            }
            else
            {
                int u, v;
                ll val;
                cin >> u >> v >> val;
                u--;
                v--;
                st.modify(u, v, val);
            }
        }
    }
```

```
}
```

6.22 Fast Segtree

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
/* t is our segment tree
We take the values directly in the tree from n to 2n-1
Tree is one based index therefore segment tree's size is 2n
   instead of 4n.
*/
ll n;
void build()
{
    for(ll i = n - 1; i > 0; --i)
        t[i] = t[i << 1] + t[i << 1 | 1];
}
// Add val to all values in the range [l,r) where l,r are 0
   based indices
void modify(ll l, ll r, ll t[], ll val)
{
    for(l += n, r += n; l < r; l >>= 1, r >>= 1)
    {
        if(l & 1)
            t[l++] += val;
        if(r & 1)
            t[--r] += val;
    }
}
// Set val at position p zero based
void modify(ll p, ll t[], ll val)
{
    for(t[p += n] = val; p > 1; p >>= 1)
        t[p >> 1] = t[p] + t[p ^ 1];
}
// Push all values down to leaves to access each value if
   needed
void push(ll t[])
{
    for(ll i = 1; i < n; i++)
    {
        t[i << 1] += t[i];
        t[i << 1 | 1] += t[i];
        t[i] = 0;
    }
}
// Return Sum of the range [l,r) where l,r are zero based
   indices
ll query(ll l, ll r, ll t[])
{
    ll res{};
    for(l += n, r += n; l < r; l >>= 1, r >>= 1)
    {
        if(l & 1)
            res += t[l++];
        if(r & 1)
            res += t[--r];
    }
    return res;
}
// Returns the value at position p zero based after number
   of operations
ll query(ll p, ll t[])
{
    ll res{};
    for(p += n; p > 0; p >>= 1)
    {
        res += t[p];
    }
    return res;
}
// Lazy Propagation
const int N = 1e5;
int n;
int seg[2 * N];
int h;
int d[N];
void apply(int p, int value)
{
    seg[p] += value;
    if (p < n) d[p] += value;
```

```
}
void build(int p)
{
    while (p > 1) p >>= 1, seg[p] = seg[p << 1] + seg[p << 1
       | 1] + d[p];
}
void push(int p)
{
    for (int s = h; s > 0; --s)
    {
        int i = p >> s;
        if (d[i] != 0)
        {
            apply(i << 1, d[i]);
            apply(i << 1 | 1, d[i]);
            d[i] = 0;
        }
    }
}
void inc(int l, int r, int value)
{
    l += n, r += n;
    int l0 = l, r0 = r;
    for (; l < r; l >>= 1, r >>= 1)
    {
        if (l & 1) apply(l++, value);
        if (r & 1) apply(--r, value);
    }
    build(l0);
    build(r0 - 1);
}
int query(int l, int r)
{
    l += n, r += n;
    push(l);
    push(r - 1);
    int res = 0;
    for (; l < r; l >>= 1, r >>= 1)
    {
        if (l & 1) res += seg[l++];
        if (r & 1) res += seg[--r];
    }
    return res;
}
```

6.23 Persistent Segment Tree

```
struct perseg {
    struct node {
        node *l, *r;
        int sum;
        node (int val): l(nullptr), r(nullptr), sum(val) {}
        node (node *l, node *r): l(l), r(r), sum(0) {
            if (l) sum += l->sum;
            if (r) sum += r->sum;
        }
    };
    int n;
    vector<node*>tree; // vector of roots
    template<typename T>
    node* build(int tl, int tr, const vector<T>&v)
    {
        if (tl == tr)
        {
            return new node(v[tl]);
        }
        int tm = (tl + tr) >> 1;
        return new node(build(tl, tm, v), build(tm + 1, tr,
           v));
    }
    node* build(int tl, int tr)
    {
        if (tl == tr)
        {
            return new node(0);
        }
        int tm = (tl + tr) >> 1;
        return new node(build(tl, tm), build(tm + 1, tr));
    }
    template<typename T>
    perseg(const vector<T> &v)
    {
```

```
    n = (int)v.size();
    tree.clear();
    node *a = build(0, n - 1, v);
    tree.push_back(a);
}
perseg(int n_): n(n_)
{
    tree.clear();
    node *a = build(0, n - 1);
    tree.push_back(a);
}
int get(node *root, int tl, int tr, int ql, int qr)
{
    if (ql > tr || qr < tl)
        return 0;
    if (ql <= tl && tr <= qr)
        return root->sum;
    int tm = (tl + tr) >> 1;
    return get(root->l, tl, tm, ql, qr) + get(root->r,
       tm + 1, tr, ql, qr);
}
int get(int root, int l, int r)
{
    return get(tree[root], 0, n - 1, l, r);
}
template<typename T>
node* modify(node *root, int tl, int tr, int pos, const
   T &v) {
    if (tl == tr) {
        return new node(root->sum + v);
    }
    int tm = (tl + tr) >> 1;
    if (pos <= tm)
        return new node(modify(root->l, tl, tm, pos, v),
           root->r);
    else
        return new node(root->l, modify(root->r, tm + 1,
           tr, pos, v));
}
template<typename T>
void modify(int root, int pos, const T &v) {
    node *a = modify(tree[root], 0, n - 1, pos, v);
    tree.push_back(a);
}
int query(node* u, node* v, int l, int r, int k) {
    if (l == r)
        return l;
    int m = (l + r) >> 1;
    int leftval = (v->l->sum) - (u->l->sum);
    int rightval = (v->r->sum) - (u->r->sum);
    if (leftval >= k)
        return query(u->l, v->l, l, m, k);
    else
        return query(u->r, v->r, m + 1, r, k - leftval);
}
int query(int u, int v, int k) {
    return query(tree[u], tree[v], 0, n - 1, k);
}
};
```

6.24 Sparse Table

```
// usage:
// auto fun = [&](int i, int j) { return min(i, j); };
// sparse_table<int, decltype(fun)> st(a, fun);
// or :
// sparse_table<int> st(a, [&](int i, int j) { return min(i,
   j); });
// vector must be immutable
template <typename T, class F = function<T(const T&, const
   T&)>>
class sparse_table {
public:
    int n;
    vector<vector<T>> mat;
    F func;
    int max_log;
    // O(N.log(N)) preprocessing and memory
    sparse_table(const vector<T>& a, const F& f) : func(f) {
        n = static_cast<int>(a.size());
        max_log = 32 - __builtin_clz(n);
        mat.resize(max_log);
```

```
        mat[0] = a;
        for (int j = 1; j < max_log; j++) {
            mat[j].resize(n - (1 << j) + 1);
            for (int i = 0; i <= n - (1 << j); i++) {
                mat[j][i] = func(mat[j - 1][i], mat[j - 1][i
                    + (1 << (j - 1))]);
            }
        }
    }

    // O(1) : min, max, gcd, lcm, and
    // X = func(X, X)
    T get_idempotent(int from, int to) const {
        assert(0 <= from && from <= to && to <= n - 1);
        int lg = 32 - __builtin_clz(to - from + 1) - 1;
        return func(mat[lg][from], mat[lg][to - (1 << lg) +
            1]);
    }

    // O(log(N)) Change to sqrt_tree, disjoint_sparse_table
        if need faster
    // or, xor, sum, product
    T get_non_idempotent(int from, int to) const {
        assert(0 <= from && from <= to && to <= n - 1);
        T res = 0; // Change according to question
        for (int j = max_log; j >= 0; j--) {
            if ((1 << j) <= to - from + 1) {
                res = func(res, mat[j][from]);
                from += (1 << j);
            }
        }
        return res;
    }
};
```

6.25 Splay Trees

```
template<typename T>
struct splaytree
{
    struct node
    {
        node *ch[2], *p;
        int sz;
        T val;
        node(T v)
        {
            ch[0] = ch[1] = p = nullptr;
            sz = 1;
            val = v;
        }
        void update()
        {
            sz = 1;
            for (int i = 0; i < 2; i++)
                if (ch[i])
                {
                    sz += ch[i]->sz;
                }
        }
    };

    node *root;

    splaytree()
    {
        root = nullptr;
    }
    ~splaytree()
    {
        vector<node *> q = {root};
        while (q.size())
        {
            node *x = q.back();
            q.pop_back();
            if (!x)
                continue;
            q.push_back(x->ch[0]);
            q.push_back(x->ch[1]);
            delete x;
        }
    }
```

```
void rotate(node *x)   // x will stay on top
{
    node *p = x->p, *pp = p->p;
    if (pp)
    {
        pp->ch[pp->ch[1] == p] = x;
    }
    // (d=0)=>{left child} and (d=1)=>{right child}
    bool d = p->ch[1] == x;
    // Right child of current becomes left child of
       parent or
    // Left child of current becomes right child of
       parent
    p->ch[d] = x->ch[!d];
    x->ch[!d] = p;
    if (p->ch[d])
    {
        p->ch[d]->p = p;
    }
    x->p = pp;
    p->p = x;
    p->update();
    x->update();
}
node *splay(node *x)
{
    if (!x) return x;
    root = x;
    while (x->p)
    {
        node *p = x->p, *pp = p->p;
        if (!pp) // zig
        {
            rotate(x);
            return x;
        }
        if ((pp->ch[0] == p) ^ (p->ch[0] == x))//
           zigzag(lr and rl)
        {
            rotate(x);
            rotate(x);
        }
        else // zigzig(ll and rr)
        {
            rotate(p);
            rotate(x);
        }
    }
    return x;
}
// lower_bound lb=1 if we need to find lower_bound
// Thus if lb=1 then no insertion will take place
node *insert(T v, bool lb = 0)
{
    if (!root)
        if (lb == 1)
        {
            return nullptr;
        }
        else
        {
            return root = new node(v);
        }
    node *x = root, *last = nullptr;
    while (1)
    {
        // (d=0)=>{left child} and (d=1)=>{right child}
        bool d = x->val < v;
        // if left child then while finding lb, lb can
           exist in right subtree
        // hence splay again from the nearest upper val
           possible
        if (d == 0) last = x;
        if (x->val == v) break;
        if (x->ch[d])
        {
            x = x->ch[d];
        }
        else
        {
            if (lb) break;
            x->ch[d] = new node(v);
            x->ch[d]->p = x;
            x = x->ch[d];
```

```
            break;
        }
    }
    splay(x);
    return lb ? splay(last) : x;
}
int size()
{
    return root ? root->sz : 0;
}
// 1 if present and 0 if absent
// First splay at value then check if it's equal to v
int count(T v)
{
    return (insert(v, 1) && root->val == v);
}
node *lower_bound(T v)
{
    return insert(v, 1);
}
void erase(T v)
{
    if (!count(v)) return;
    // count(v) splay the tree and now x is current node
       with val v
    node *x = root, *l = x->ch[0];
    if (!l)
    {
        root = x->ch[1];
        if (root) root->p = nullptr;
        return delete x;
    }
    root = l, l->p = nullptr;
    // Finding the predecessor of the node to be deleted
    while (l->ch[1]) l = l->ch[1];
    splay(l);
    l->ch[1] = x->ch[1];
    if (l->ch[1]) l->ch[1]->p = l;
    delete x;
    l->update();
}
// If key does not exist, it returns where it will be
   inserted
// return zero-based index
int order_of_key(T v)
{
    if (!count(v)) return -1;
    if (!lower_bound(v))
        return root ? root->sz : 0;
    return root->ch[0] ? root->ch[0]->sz : 0;
}
node *find_by_order(int k)
{
    if (k < 0 || k >= size())
        return nullptr;
    node *x = root;
    while (1)
    {
        if (x->ch[0] and x->ch[0]->sz >= k + 1)
        {
            x = x->ch[0];
        }
        else
        {
            if (x->ch[0])
            {
                k -= x->ch[0]->sz;
            }
            if (k == 0)
            {
                return splay(x);
            }
            if (!x->ch[1])
            {
                splay(x);
                return nullptr;
            }
            k--; // for current root
            x = x->ch[1];
        }
    }
}
T min()
{
```

```
        node *x = root;
        while (x->ch[0]) x = x->ch[0]; // max -> ch[1]
        return splay(x)->val;
    }
};
```

6.26 Tensor

```
template<typename T, int NDIMS> struct tensor_view {
    static_assert(NDIMS >= 0, "NDIMS must be non_negative");

protected:
    array<int, NDIMS>shape;
    array<int, NDIMS>strides;
    T*data;

    tensor_view(array<int, NDIMS>shape_, array<int,
       NDIMS>strides_, T*data_): shape(shape_),
       strides(strides_), data(data_) {}

public:
    tensor_view(): shape( {0}), strides({0}), data(nullptr)
       {}

protected:
    int flatten_index(array<int, NDIMS> idx) const {
        int res{};
        for (int i = 0; i < NDIMS; i++) {
            assert(0 <= idx[i] && idx[i] < shape[i]);
            res += idx[i] * strides[i];
        }
        return res;
    }
    // int flatten_index_checked(array<int, NDIMS> idx)
       const {
    //     int res = 0;
    //     for (int i = 0; i < NDIMS; i++) {
    //         assert(0 <= idx[i] && idx[i] < shape[i]);
    //         res += idx[i] * strides[i];
    //     }
    //     return res;
    // }

public:

    T& operator[](array<int, NDIMS> idx) const {
        return data[flatten_index(idx)];
    }

    T& at(array<int, NDIMS> idx) const {
        return data[flatten_index(idx)];
    }

    // template <int D = NDIMS>
    // enable_if_t < (0 < D), tensor_view < T, NDIMS - 1 >>
       operator[] (int idx) const {
    //     array < int, NDIMS - 1 > nshape;
       copy(shape.begin() + 1, shape.end(), nshape.begin());
    //     array < int, NDIMS - 1 > nstrides;
       copy(strides.begin() + 1, strides.end(),
       nstrides.begin());
    //     T* ndata = data + (strides[0] * idx);
    //     return tensor_view < T, NDIMS - 1 > (nshape,
       nstrides, ndata);
    // }
    // template <int D = NDIMS>
    // enable_if_t < (0 < D), tensor_view < T, NDIMS - 1 >>
       at(int idx) const {
    //     assert(0 <= idx && idx < shape[0]);
    //     return operator[](idx);
    // }

    // template <int D = NDIMS>
    // enable_if_t<(0 == D), T&> operator * () const {
    //     return *data;
    // }

    template <typename U, int D> friend struct tensor_view;
    template <typename U, int D> friend struct tensor;
};

template<typename T, int NDIMS> struct tensor {
```

```
    static_assert(NDIMS >= 0, "NDIMS must be non_negative");

protected:
    array<int, NDIMS> shape;
    array<int, NDIMS> strides;
    int len;
    T* data;

public:
    tensor(): shape( {0}), strides({0}), len(0),
       data(nullptr) {}

    // t = 0 by default if not passed
    explicit tensor(array<int, NDIMS> shape_, const T&t =
       T()) {
        // Depth first, height second , width third rule
        shape = shape_;
        strides[NDIMS - 1] = 1;
        for (int i = NDIMS - 2; i >= 0; i--) {
            strides[i] = strides[i + 1] * shape[i + 1];
        }
        // Flatten tensor size
        len = strides[0] * shape[0];
        data = new T[len];
        fill(data, data + len, t);
    }
    // Simple copy constructor (l value reference)
    tensor(const tensor &o): shape(o.shape),
       strides(o.strides), len(o.len), data(new T[len]) {
        for (int i = 0; i < len; ++i) {
            data[i] = o.data[i];
        }
    }

    // = overloaded with rvalue reference
    tensor&    operator = (tensor&& o ) noexcept {
        swap(shape, o.shape);
        swap(strides, o.strides);
        swap(len, o.len);
        swap(data, o.data);
        return *this;
    }
```

```
    // Simple copy constructor (r value reference)
    // First call default constructor then use prefect
       forwarding
    tensor(tensor&& o) : tensor() {
        *this = move(o);
    }

    tensor& operator = (const tensor& o) {
        return *this = tensor(0);
    }

    ~tensor() { delete[] data; }

    using view_t = tensor_view<T, NDIMS>;

    view_t view() {
        return view_t(shape, strides, data);
    }
    operator view_t() {
        return view();
    }

    using const_view_t = tensor_view<const T, NDIMS>;
    const_view_t view() const {
        return tensor_view<const T, NDIMS>(shape, strides,
           data);
    }
    operator const_view_t() const {
        return view();
    }

    T& operator[] (std::array<int, NDIMS> idx) { return
       view()[idx]; }
    T& at(std::array<int, NDIMS> idx) { return
       view().at(idx); }
    const T& operator[] (std::array<int, NDIMS> idx) const {
       return view()[idx]; }
    const T& at(std::array<int, NDIMS> idx) const { return
       view().at(idx); }

    // template <int D = NDIMS>
```

```
    // std::enable_if_t < (0 < D), tensor_view < T, NDIMS -
       1 >> operator[] (int idx) {
    //     return view()[idx];
    // }
    // template <int D = NDIMS>
    // std::enable_if_t < (0 < D), tensor_view < T, NDIMS -
       1 >> at(int idx) {
    //     return view().at(idx);
    // }

    // template <int D = NDIMS>
    // std::enable_if_t < (0 < D), tensor_view < const T,
       NDIMS - 1 >> operator[] (int idx) const {
    //     return view()[idx];
    // }
    // template <int D = NDIMS>
    // std::enable_if_t < (0 < D), tensor_view < const T,
       NDIMS - 1 >> at(int idx) const {
    //     return view().at(idx);
    // }

    // template <int D = NDIMS>
    // std::enable_if_t<(0 == D), T&> operator * () {
    //     return *view();
    // }
    // template <int D = NDIMS>
    // std::enable_if_t<(0 == D), const T&> operator * ()
       const {
    //     return *view();
    // }
};
```

6.27 Treap

```
mt19937_64
    rng(chrono::steady_clock::now().time_since_epoch().count());
struct node
{
    int mx;
    int sum;
    int add;
    int val;

    int x;//Key
    int y;//priority
    int sz;
    node *l, *r;
    node(int _x, int _val): val(_val), mx(_val), add(0),
       x(_x), sz(1), sum(_val), y(rng()), l(nullptr),
       r(nullptr) {}
    void apply(int v)
    {
        val += v;
        mx += v;
        sum += v * sz;
        add += v;
    }
    void push()
    {
        if(add != 0)
        {
            if(l != nullptr)
                l->apply(add);
            if(r != nullptr)
                r->apply(add);
            add = 0;
        }
    }
    void pull()
    {
        mx = max(val, max(get_max(l), get_max(r)));
        sum = val + get_sum(l) + get_sum(r);
        sz = 1 + get_size(l) + get_size(r);
    }
    static int get_max(node *t)
    {
        return (t == nullptr ? numeric_limits<int>::min() :
           t->mx);
    }
    static int get_sum(node *t)
    {
        return (t == nullptr ? 0 : t->sum);
```

```
    }
    static int get_size(node *t)
    {
        return (t == nullptr ? 0 : t->sz);
    }
};
// splits tree into l,r such that all keys in l are <x and
   in r >x
void split(node *t, int x, node *&l, node *&r)
{
    if(!t)
    {
        l = r = nullptr;
    }
    else
    {
        t->push();
        if(t->x >= x)
        {
            split(t->l, x, l, t->l);
            r = t;
        }
        else
        {
            split(t->r, x, t->r, r);
            l = t;
        }
        t->pull();
    }
}
//(all keys X in T1 are smaller than keys in T2)
void merge(node *&t, node *&l, node *&r)
{
    if (!l || !r)
    {
        t = l ? l : r;
    }
    else
    {
        l->push();
        r->push();
        if(l->y > r->y)
        {
            merge(l->r, l->r, r);
            t = l;
        }
        else
        {
            merge(r->l, l, r->l);
            t = r;
        }
        t->pull();
    }
}
void insert(node *&t, int x, int val)
{
    node *l, *r;
    split(t, x, l, r);
    auto mid = new node(x, val);
    merge(t, l, mid);
    merge(t, t, r);
}
void remove(node *&t, int x)
{
    node *l1, *r1, *l2, *r2;
    split(t, x, l1, r1);
    split(r1, x + 1, l2, r2);
    delete l2;
    merge(t, l1, r2);
}

void modify(node *&t, int ql, int qr, int delta)
{
    node *l1, *r1, *l2, *r2;
    split(t, qr + 1, l1, r1);
    split(l1, ql, l2, r2);
    if (r2 != nullptr)
        r2->apply(delta);
    merge(t, l2, r2);
    merge(t, t, r1);
}

node query(node *&t, int ql, int qr)
{
```

```
    node *l1, *r1, *l2, *r2;
    split(t, qr + 1, l1, r1);
    split(l1, ql, l2, r2);
    node res(0, 0);
    if (r2)
        res = *r2;
    merge(t, l2, r2);
    merge(t, t, r1);
    return res;
}

void clear(node *&t)
{
    if (!t)
        return;
    clear(t->l);
    clear(t->r);
    delete t;
    t = nullptr;
}

void print(node *t)
{
    if (!t)
        return;
    print(t->l);
    cout << t->val << '\n';
    print(t->r);
}
```

6.28 Tries

```
#include<bits/stdc++.h>
using namespace std;
template<int max_ascii>
struct trie
{
    struct node
    {
        node *c[max_ascii];
        int cnt = 0;
    };
    node *root = new node();
    void insert(const int &x)
    {
        node *head = root;
        for(int i = 30; i >= 0; i--)
        {
            int d = (x >> i) & 1;
            if(!head->c[d])
            {
                head->c[d] = new node();
            }
            head = head->c[d];
            head->cnt++;
        }
    }
    void remove(const int &x)
    {
        node *head = root;
        for(int i = 30; i >= 0; i--)
        {
            int d = (x >> i) & 1;
            head = head->c[d];
            head->cnt--;
        }
    }
    int query(const int &x)
    {
        node *head = root;
        int res{};
        for(int i = 30; i >= 0; i--)
        {
            int d = (x >> i) & 1;
            if(head->c[(d ^ 1)] != nullptr && head->c[(d ^
                1)]->cnt > 0)
            {
                res += (1 << i);
                head = head->c[(d ^ 1)];
            }
            else
            {
```

```
                head = head->c[d];
            }
        }
        return res;
    }
};
```

7 Techniques

7.1 Binary Lift Fenwick

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 1 << 20 + 1;
int ft[maxn], a[maxn];
int n, q;
const int lg = 20;
void add(int idx, int val)
{
    for(; idx <= n; idx += idx & -idx)
    {
        ft[idx] += val;
    }
}
int binary_lifting(int x)
{
    int sum{};
    int pos = 0;
    for(int h = lg; h >= 0; h--)
    {
        if( (pos + (1 << h)) < n && sum + ft[(pos + (1 <<
            h))] < x)
        {
            sum += ft[(pos + (1 << h))];
            pos += (1 << h);
        }
    }
    return pos + 1;
}
```

7.2 Binary Search Fenwick

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 1e6 + 1;
int ft[maxn], a[maxn];
int n, x, q, l, r;
void add(int idx, int val)
{
    for(idx; idx <= n; idx += idx & -idx)
    {
        ft[idx] += val;
    }
}
int query(int idx)
{
    int sum{};
    for(; idx; idx -= idx & -idx)
    {
        sum += ft[idx];
    }
    return sum;
}
int binary_search(int x)
{
    l = 1, r = n;
    while(l < r)
    {
        int mid = (l + r) >> 1;
        int val = query(mid);
        if(val >= x)
            r = mid;
        else
            l = mid + 1;
    }
    return l;
}
```

7.3 Convex Hull Trick

```cpp
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
using ld = long double;
#define sz(x) (int)(x).size()
// dp[i] = max,min(dp[i], dp[j] + fun(i) * fun(j));
/*  Container where you can add lines of the form ax+b, and
    get_max maximum values at points x. For each line, also
       keeps a value p,
    which is the last (maximum) point for which the current
       line is dominant.
    (obviously, for the last line, p is infinity) Useful for
       dynamic programming.
*/
/*
    Suppose l_1, l_2, and l_3 are the second line from the
       top,
    the line at the top, and the line to be added,
       respectively.
    Then, l_2 becomes irrelevant if and only if the
       intersection point
    of l_1 and l_3 is to the left of the intersection of l_1
       and l_2.
    (This makes sense because it means that the interval in
       which l_3
    is minimal subsumes that in which l_2 was previously.)
    We have assumed for the sake of simplicity that no three
       lines
    are concurrent.
*/
vector<ll>M, C;
int p;
bool bad(ll l1, ll l2, ll l3)
{
    return (C[l3] - C[l1]) * (ld)(M[l2] - M[l3]) <= (C[l3] -
       C[l2]) * (ld)(M[l1] - M[l3]);
}
void add(ll m, ll c)
{
    M.push_back(m), C.push_back(c);
    while(sz(M) >= 3 && bad(sz(M) - 3, sz(M) - 2, sz(M) - 1))
    {
        M[sz(M) - 2] = M[sz(M) - 1];
        C[sz(M) - 2] = C[sz(M) - 1];
        M.pop_back();
        C.pop_back();
    }
}
ll query(ll x)
{
    if(p >= sz(M))
        p = sz(M) - 1;
    // change sign accordingly to min,max
    while(p + 1 < sz(M) && M[p]*x + C[p] <= M[p + 1]*x + C[p
       + 1])
        p++;
    return M[p] * x + C[p];
}
```

7.4 Mo's Algorithm

```cpp
#include <bits/stdc++.h>
using namespace std;
const int maxn = 2e5 + 1;
const int blk_sz = 500;
struct event
{
    int l, r, idx;
    bool operator<(const event &a) const
    {
        if(l / blk_sz != a.l / blk_sz)
            return l / blk_sz < a.l / blk_sz;
        return r < a.r;
    }
};
vector<event>vec;
ll res;
ll a[maxn];
ll mp[1000001];
void add(ll idx)
{
    ll prevFreq = mp[a[idx]];
```

```
    mp[a[idx]]++;
    res -= (prevFreq * prevFreq) * a[idx];
    prevFreq++;
    res += (prevFreq * prevFreq) * a[idx];
}
void remove(ll idx)
{
    ll prevFreq = mp[a[idx]];
    mp[a[idx]]--;
    res -=  (prevFreq * prevFreq) * a[idx];
    prevFreq --;
    res +=  (prevFreq * prevFreq) * a[idx];
}
int32_t main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);
    int n, q;
    cin >> n >> q;
    vec.resize(q);
    rep(i, n)
    {
        cin >> a[i];
    }
    rep(i, q)
    {
        ll x, y;
        cin >> vec[i].l >> vec[i].r;
        vec[i].l--;
        vec[i].r--;
        vec[i].idx = i;
    }
    sort(vec.begin(), vec.end());
    ll ML = 0, MR = -1;
    ll ans[q];
    rep(i, q)
    {
        ll l = vec[i].l;
        ll r = vec[i].r;
        while(ML > l)
            ML--, add(ML);
        while(MR < r)
            MR++, add(MR);
        while(ML < l)
            remove(ML), ML++;
        while(MR > r)
            remove(MR), MR--;
        ans[vec[i].idx] = res;
    }
    rep(i, q)
    cout << ans[i] << "\n";
    return 0;
}
```

7.5 Nearest greater/smaller elements

```
// Nearest greater/smaller than equal to elements
// replace with equal to sign for strictly smaller/greater.
vector<int>rge(n, -1), lge(n, -1), rse(n, -1), lse(n, -1);
// Nearest greater from the left
for (int i = 0; i < n; i++)
{
    while (!st.empty() && a[st.back()] < a[i])
        st.pop_back();
    if (!st.empty())
        lge[i] = st.back();
    st.push_back(i);
}
st.clear();
// Nearest smaller from the left
for (int i = 0; i < n; i++)
{
    while (!st.empty() && a[st.back()] > a[i])
        st.pop_back();
    if (!st.empty())
        lse[i] = st.back();
    st.push_back(i);
}
st.clear();
// Nearest greater from the right
for (int i = n - 1; i >= 0; i--)
```

```
{
    while (!st.empty() && a[st.back()] < a[i])
    {
        st.pop_back();
    }
    if (!st.empty())
        rge[i] = st.back();
    st.push_back(i);
}
st.clear();
// Nearest smaller from the right
for (int i = n - 1; i >= 0; i--)
{
    while (!st.empty() && a[st.back()] > a[i])
        st.pop_back();
    if (!st.empty())
        rse[i] = st.back();
    st.push_back(i);
}
st.clear();
```

7.6 Parallel Binary Search

```
#include <bits/stdc++.h>
using namespace std;
template <typename T>
class fenwick
{
public:
    vector<T> fenw;
    int n;
    fenwick(int _n) : n(_n)
    {
        fenw.resize(n);
    }
    fenwick(const vector<T>&v)
    {
        n = v.size();
        fenw.resize(n);
        for (int i = 0; i < n; ++i) {
            modify(i, v[i]);
        }
    }
    void modify(int x, T v)
    {
        while (x < n)
        {
            fenw[x] += v;
            x |= (x + 1);
        }
    }
    long long get(int x)
    {
        long long v{};
        while (x >= 0)
        {
            v += fenw[x];
            x = (x & (x + 1)) - 1;
        }
        return v;
    }
    void apply(int l, int r, T v) {
        if (l <= r) {
            modify(l, v);
            modify(r + 1, -v);
        }
        else {
            modify(1, v);
            modify(r + 1, -v);
            modify(l, v);
        }
    }
};
int32_t main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);
    int n, m;
    cin >> n >> m;

    vector<int>a(m + 1);
    vector<int>owns[n + 1];
```

```
for (int i = 1; i <= m; i++) {
    cin >> a[i];
    owns[a[i]].push_back(i);
}

vector<long long>reqd(n + 1);
for (int i = 1; i <= n; i++)
    cin >> reqd[i];

int q;
cin >> q;

vector<int>ql(q + 1), qr(q + 1);
vector<long long> qa(q + 1);

for (int i = 1; i <= q; i++)
    cin >> ql[i] >> qr[i] >> qa[i];

vector<int>lo(n + 1), hi(n + 1), mid(n + 1);

function<void(void)>paralong longel_binary = [&]() {
    vector<int>vec[q + 1];
    for (int i = 1; i <= n; i++) {
        if (mid[i] > 0)
            vec[mid[i]].emplace_back(i);
    }

    fenwick<long long>ft(m + 1);

    function<bool(int)>check = [&](int idx) {
        long long req = reqd[idx];
        for (auto &x : owns[idx]) {
            long long y = ft.get(x);
            req -= y;
            if (req < 0)
                break;
        }
        if (req <= 0)
            return true;
        return false;
    };

    for (int i = 1; i <= q; i++) {
        ft.apply(ql[i], qr[i], qa[i]);
        for (auto &x : vec[i]) {
            if (check(x))
                hi[x] = i;
            else
                lo[x] = i + 1;
        }
    }
};

function<void(void)>solve = [&]() {
    for (int i = 1; i <= n; i++)
        lo[i] = 1, hi[i] = q + 1;
    bool is_valid = true;
    while (is_valid) {
        is_valid = false;
        for (int i = 1; i <= n; i++) {
            if (lo[i] < hi[i]) {
                is_valid = true;
                mid[i] = (lo[i] + hi[i]) >> 1;
            }
            else
                mid[i] = -1;
        }
        paralong longel_binary();
    }
};

solve();

for (int i = 1; i <= n; i++) {
    if (lo[i] <= q)
        cout << lo[i] << '\n';
    else
        cout << "NIE" << '\n';
}
return 0;
}
```

8 Trees

8.1 Centroid Decomposition

```
const int maxn = 1e5 + 10;
const int lg = 18;
int n, q, u, v;
vector<int>adj[maxn];
int parent[maxn][lg], decomposedparent[maxn], subtree[maxn];
int closestred[maxn], level[maxn];
namespace init
{
    void dfs(int u, int p = 0, int lev = 1)
    {
        level[u] = lev;
        parent[u][0] = p;
        for(auto it : adj[u])
        {
            if(it != p)
                dfs(it, u, lev + 1);
        }
    }
    void precompute()
    {
        for(int l = 1; l < lg; l++)
        {
            for(int i = 1; i <= n; i++)
            {
                if(parent[i][l - 1])
                {
                    parent[i][l] = parent[parent[i][l -
                       1]][l - 1];
                }
            }
        }
    }
}
int lca(int u, int v)
{
    if(level[u] > level[v])
        swap(u, v);
    int diff = level[v] - level[u];
    for(int i = lg - 1; i >= 0; i--)
    {
        if(diff & (1 << i))
            v = parent[v][i];
    }
    if(u == v)
        return u;
    for(int i = lg - 1; i >= 0; i--)
    {
        if(parent[u][i] && parent[u][i] != parent[v][i])
        {
            u = parent[u][i];
            v = parent[v][i];
        }
    }
    return parent[u][0];
}
namespace centroid
{
    int nodes;
    void dfs(int u, int p)
    {
        subtree[u] = 1;
        nodes++;
        for(auto it : adj[u])
        {
            if(it != p)
            {
                dfs(it, u);
                subtree[u] += subtree[it];
            }
        }
    }
    int centroid(int u, int p)
    {
        for(auto it : adj[u])
        {
            if(it != p)
            {
                if(subtree[it] > (nodes >> 1))
                    return centroid(it, u);
```

```
            }
        }
        return u;
    }
    void decompose(int u, int p = 0)
    {
        nodes = 0;
        dfs(u, u);
        int c = centroid(u, u);
        decomposedparent[c] = p;
        for(auto it : adj[c])
        {
            adj[it].erase(remove(all(adj[it]), c),
               adj[it].end());
            decompose(it, c);
        }
    }
}
int distance(int u, int v) {
    return (level[u] + level[v] - 2 * level[lca(u, v)]);
}
```

8.2 Tree Diameter using single DFS

```
#include<bits/stdc++.h>
using namespace std;
const int maxn = 2e5;
vector<int>g[maxn];
int d[maxn];
int ans;
void dfs(int u, int p) {
    for(auto v : g[u])
        if(v != p) {
            dfs(v, u);
            ans = max(ans, d[u] + d[v] + 1);
            d[u] = max(d[u], d[v] + 1);
        }
}
// dfs(0, -1);
// cout << ans;
```

8.3 DSU on Trees

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 1e5 + 5;
vector<int>g[maxn];
int sz[maxn], cnt[maxn];
int color[maxn];
void getsz(int u, int p)
{
    sz[u] = 1;
    for (auto v : g[u])
    {
        if (v != p)
        {
            getsz(v, u);
            sz[u] += sz[v];
        }
    }
}
vector<int> *vec[maxn];
long long ans[maxn];
int maxi[maxn];
/*
Keep is 0 for small children and 1 for big child
*/
void dfs(int u, int p, bool keep)
{
    int mx = -1, bigChild = -1;
    for (auto v : g[u])
        if (v != p)
            if (sz[v] > mx)
                mx = sz[v], bigChild = v;
    for (auto v : g[u])
        if (v != p && v != bigChild)
            dfs(v, u, 0);
    if (bigChild != -1)
        dfs(bigChild, u, 1), vec[u] = vec[bigChild], ans[u]
           = ans[bigChild], maxi[u] = maxi[bigChild];
    else
        vec[u] = new vector<int> (), ans[u] = 0, maxi[u] = 0;
    vec[u]->push_back(u);
```

```
    // Now vec has information about it's big child and
       itself
    // vec[v]=v+bigChild
    cnt[color[u]]++;
    if (cnt[color[u]] > maxi[u])
        maxi[u] = cnt[color[u]], ans[u] = 0;
    if (cnt[color[u]] == maxi[u])
        ans[u] += color[u];
    for (auto v : g[u])
        if (v != p && v != bigChild)
            for (auto x : *vec[v])
            {
                cnt[color[x]]++;
                if (cnt[color[x]] > maxi[u])
                    maxi[u] = cnt[color[x]], ans[u] = 0;
                if (maxi[u] == cnt[color[x]])
                    ans[u] += color[x];
                vec[u]->push_back(x);
            }
    // Now vec has information about it's whole subtree
    // vec[v] = v + bigChild+vec[all(smallChild)]

    /*
    Go down to small child, we have to delete its
       information about
    color count so that the query answer of v's first child
       subtree
    won't affect second and so on. (because we are using
       global array)
    That is also the reason why we have to call small child
       before big
    child recur (to make bigchild and small child answer not
       overlap).
    */
    if (keep == 0)
        for (auto it : *vec[u])
            cnt[color[it]]--;
}
```

8.4 Heavy Light Decomposition - 1

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 1e4 + 10;
const int lg = 14;
int parent[maxn][lg], level[maxn];
int head[maxn], pos[maxn], heavy[maxn], rpos[maxn];
int st[2 * maxn];
int previous[maxn];
int timer;
int n;
const int inf = 1e6 + 1;
vector<pair<int, int> >adj[maxn];
void init()
{
    forn(i, 1, n)
    {
        adj[i].clear();
        head[i] = 0;
        heavy[i] = -1;
        pos[i] = 0;
        rpos[i] = 0;
        previous[i] = 0;
        st[i + n - 1] = -inf;
        forn(j, 0, lg - 1)
        parent[i][j] = 0;
    }
    timer = 1;
    return;
}
int dfs(int u, int p = 0, int lev = 1)
{
    parent[u][0] = p;
    level[u] = lev;
    int h = 1;
    int maxc = 0;
    for(auto it : adj[u])
    {
        if(it.ff != p)
        {
            previous[it.ff] = it.ss;
            int c = dfs(it.ff, u, lev + 1);
            h += c;
```

```
            if(c > maxc)
            {
                maxc = c;
                heavy[u] = it.ff;
            }
        }
    }
    return h;
}
void precompute()
{
    for(int l = 1; l < lg; l++)
    {
        for(int i = 1; i <= n; i++)
        {
            if(parent[i][l - 1])
                parent[i][l] = parent[parent[i][l - 1]][l -
                    1];
        }
    }
    return;
}
void hld(int u, int p)
{
    pos[timer] = u;
    rpos[u] = timer++;
    head[u] = p;
    if(heavy[u] != -1)
    {
        hld(heavy[u], p);
    }
    for(auto it : adj[u])
    {
        if(heavy[u] != it.ff && it.ff != parent[u][0])
        {
            hld(it.ff, it.ff);
        }
    }
    return;
}
void build()
{
    for(int i = n - 1; i > 0; i--)
        st[i] = max(st[i << 1], st[i << 1 | 1]);
    return;
}
int segquery(int u, int v)
{
    u--;
    int res{};
    for(u += n, v += n; u < v; u >>= 1, v >>= 1)
    {
        if(u & 1)
            res = max(res, st[u++]);
        if(v & 1)
            res = max(res, st[--v]);
    }
    return res;
}
int query(int u, int v)
{
    int res{};
    for(; head[u] != head[v]; v = parent[head[v]][0])
    {
        if(level[head[u]] > level[head[v]])
            swap(u, v);
        int curmax = segquery(rpos[head[v]], rpos[v]);
        res = max(res, curmax);
    }
    if(u == v)
        return res;
    if (level[u] > level[v])
    {
        swap(u, v);
    }
    int a = rpos[v];
    int h = head[v];
    int diff = level[v] - level[u] - 1;
    for(int l = lg - 1; l >= 0; l--)
    {
        if(diff & (1 << l))
        {
            v = parent[v][l];
        }
```

```
    }
    int curmax = segquery(rpos[v], a);
    res = max(res, curmax);
    return res;
}
```

8.5 Heavy Light Decomposition - 2

```
struct HLD
{
    vector<vector<int>> tree;
    bool vov; // True if values are on vertices, false if
       values are on edges.
    segtree segment_tree;
    vector<int> parent;
    vector<int> lv;
    vector<int> head;
    vector<int> tin;
    HLD(const vector<vector<int>> &t, bool _vov): tree(t),
       vov(_vov), segment_tree(t.size()), parent(t.size()),
       lv(t.size()), head(t.size()), tin(t.size())
    {
        int time = 0;
        parent[0] = -1;
        function<int(int)> dfs1 = [&](int u)
        {
            int h = 1;
            int maxc = 0;
            for (int &v : tree[u])
            {
                if (v == parent[u])
                    continue;
                parent[v] = u;
                lv[v] = lv[u] + 1;
                int c = dfs1(v);
                if (c > maxc)
                {
                    maxc = c;
                    swap(v, tree[u][0]);
                }
                h += c;
            }
            return h;
        };
        function<void(int)> decompose = [&](int u)
        {
            tin[u] = time++;
            for (int v : t[u])
            {
                if (v == parent[u])
                    continue;
                head[v] = v == t[u][0] ? head[u] : v;
                decompose(v);
            }
        };
        dfs1(0);
        decompose(0);
}
segtree::node get(int u, int v) {
    segtree::node res;
    process_path(u, v, [this, &res](int a, int b) {
        res = segtree::unite(res, segment_tree.get(a,
           b));
    });
    return res;
}
void modify(int u, int v, long long delta) {
    process_path(u, v, [this, delta](int a, int b) {
        segment_tree.modify(a, b, delta);
    });
}
void process_path(int u, int v, const function<void(int
   x, int y)> &operation) {
    for (; head[u] != head[v]; v = parent[head[v]]) {
        if (lv[head[u]] > lv[head[v]])
            swap(u, v);
        operation(tin[head[v]], tin[v]);
    }
    if (u != v || vov)
        operation(min(tin[u], tin[v]) + (vov ? 0 : 1),
           max(tin[u], tin[v]));
}
```

```
};
```

8.6 Mo's Algorithm on Trees

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 2e5 + 5;
const int blk = 500;
struct Query
{
    int l, r, idx, lc;
    bool flag;
    bool operator<(const Query &q) const
    {
        int x = l / blk;
        int y = q.l / blk;
        if (x != y)
            return x < y;
        return (x % 2 == 1 ? r<q.r: r>q.r);
    }
};
vector<int>g[maxn];
int a[maxn];
int lvl[maxn];
int st[maxn], en[maxn];
int euler[maxn];
int timer = 1;
const int lg = 19;
int dp[maxn][lg];
void dfs(int u, int p)
{
    lvl[u] = lvl[p] + 1;
    euler[timer] = u;
    st[u] = timer++;
    dp[u][0] = p;
    for (auto v : g[u])
    {
        if (v != p)
        {
            dfs(v, u);
        }
    }
    euler[timer] = u;
    en[u] = timer++;
}
void preprocess(int n)
{
    for (int l = 1; l < lg; l++)
    {
        for (int i = 1; i <= n; i++)
        {
            dp[i][l] = dp[dp[i][l - 1]][l - 1];
        }
    }
}
int lca(int a, int b)
{
    if (lvl[a] < lvl[b])
        swap(a, b);
    int dist = lvl[a] - lvl[b];
    for (int l = lg - 1; l >= 0; l--)
    {
        if (((dist >> l) & 1))
            a = dp[a][l];
    }
    if (a == b)
        return a;
    for (int l = lg - 1; l >= 0; l--)
    {
        if (dp[a][l] && dp[a][l] != dp[b][l])
        {
            a = dp[a][l];
            b = dp[b][l];
        }
    }
    return dp[a][0];
}
bool vis[maxn];
int freq[maxn];
int ans;
void upd(int node, int val)
{
```

```
    if (!vis[node])
    {
        vis[node] = 1;
        freq[val]++;
        if (freq[val] == 1)
            ans++;
    }
    else
    {
        vis[node] = 0;
        freq[val]--;
        if (freq[val] == 0)
            ans--;
    }
}
ans = 0;
timer = 1;
memset(freq, 0, sizeof freq);
memset(vis, 0, sizeof vis);
dfs(1, 0);
preprocess(n);
vector<Query>qry(q);
for (int i = 0; i < q; i++)
{
    int u, v;
    cin >> u >> v;
    qry[i].lc = lca(u, v);
    if (st[u] > st[v])
        swap(u, v);
    if (qry[i].lc == u)
        qry[i].l = st[u], qry[i].r = st[v], qry[i].flag = 0;
    else
        qry[i].l = en[u], qry[i].r = st[v], qry[i].flag = 1;
    qry[i].idx = i;
}
sort(begin(qry), end(qry));
int res[q];
int ml = 1, mr = 0;
for (int i = 0; i < q; i++)
{
    int l = qry[i].l, r = qry[i].r;
    bool flag = qry[i].flag;
    int lc = qry[i].lc;
    while (ml < l)
    {
        upd(euler[ml], a[euler[ml]]);
        ml++;
    }
    while (ml > l)
    {
        ml--;
        upd(euler[ml], a[euler[ml]]);
    }
    while (mr < r)
    {
        mr++;
        upd(euler[mr], a[euler[mr]]);
    }
    while (mr > r)
    {
        upd(euler[mr], a[euler[mr]]);
        mr--;
    }
    if (flag)
    {
        upd(lc, a[lc]);
    }
    res[qry[i].idx] = ans;
    if (flag)
    {
        upd(lc, a[lc]);
    }
}
```