

Team notebook

Pontificia Universidad Católica de Chile

April 24, 2021



Contents

1 ContestPrep	1
1.1 Template	1
2 DP	1
2.1 DivideAndConquer	1
2.2 KnuthOptimization	1
2.3 LIS	2
3 Geometry	2
3.1 2D	2
3.2 3D	5
4 Graphs	6
4.1 BellmanFord	6
4.2 Dijkstra	6
4.3 Dinic	6
4.4 FloydWarshall	7
4.5 HeavyLightDecomposition	7
4.6 HopcroftKarp	8
4.7 Hungarian	9

4.8 LCA	9
4.9 MST	10
4.10 MinCostMaxFlow	10
4.11 SCC	11
4.12 Tarjan	11
4.13 Toposort	12
5 Math	12
5.1 CRT	12
5.2 Euclid	12
5.3 FFT	12
5.4 Modular	13
5.5 ModularBinomial	13
5.6 Primality	14
5.7 PrimeFactorization	14
5.8 Sieve	14
6 Strings	14
6.1 AhoCorasick	14
6.2 Hash	15
6.3 KMP	15
6.4 Manacher	15
6.5 SuffixArray	16
6.6 SuffixAutomaton	16
6.7 Trie	17
7 Structures	17
7.1 FenwickTree	17
7.2 FenwickTree2D	18
7.3 LineContainer	18
7.4 MoQueries	18
7.5 PersistentSegmentTree	18
7.6 PersistentSegmentTreeLazy	19
7.7 PolicyBased	20
7.8 SegmentTree	20
7.9 SegmentTreeLazy	21

7.10 SparseTable	21
7.11 Treap	22
7.12 UnionFind	23
7.13 WaveletTree	23

1 ContestPrep

1.1 Template

```
#include <bits/stdc++.h>
using namespace std;

using ull = unsigned long long;
using ll = long long;
using db = double;

using pi = pair<int, int>;
using pl = pair<ll, ll>;
using pd = pair<db, db>;

using vi = vector<int>;
using vl = vector<ll>;
using vd = vector<db>;

#define rep(i, n) for (int i = 0; i < (int)n; i++)
#define repx(i, a, b) for (int i = (int)a; i < (int)b; i++)
#define all(x) begin(x), end(x)
#define eb emplace_back
#define pb push_back
#define mp make_pair
#define ff first
#define ss second
```

2 DP

2.1 DivideAndConquer

```
ll cost(int i, int j) { return ; }

vl last, now;

void compute(int l, int r, int optl, int opttr)
{
    if (l > r) return;

    int mid = (l + r) / 2;
    pair<ll, int> best = {cost(0, mid), -1};
```

```
    repx(k, max(1, optl), min(mid, opttr) + 1)
        best = min(best, {last[k - 1] + cost(k, mid), k});

    now[mid] = best.ff;

    compute(l, mid - 1, optl, best.ss);
    compute(mid + 1, r, best.ss, opttr);
}
```

2.2 KnuthOptimization

```
int N; vi A;
vector<vi> DP, OPT;

int main()
{
    DP.assign(N + 1, vi(N + 1));
    OPT.assign(N + 1, vi(N + 1));

    rep(i, N) DP[i][i + 1] = A[i + 1] - A[i], OPT[i][i + 1] = i;

    repx(d, 2, N + 1) rep(l, N + 1 - d)
    {
        int r = l + d, l_ = OPT[l][r - 1], r_ = OPT[l + 1][r];
        DP[l][r] = 1e9;
        repx(i, l_, r_ + 1)
        {
            int aux = DP[l][i] + DP[i][r] + A[r] - A[l];
            if (aux < DP[l][r]) DP[l][r] = aux, OPT[l][r] = i;
        }
    }
}
```

2.3 LIS

```
int LIS(vi &v)
{
    vi L; int S = 0;
    for(int x : v)
    {
        int i = upper_bound(all(L), x) - L.begin();
        if(i == S) L.pb(x), S++;
        else L[i] = x;
    }
    return S;
}
```

3 Geometry

3.1 2D

```

const db PI = acos(-1.0L);
const db EPS = 1e-12;

// POINT 2D

typedef db T; struct P
{
    T x, y;
    P() {} P(T x, T y) : x(x), y(y) {}

    P operator+(const P &p) const { return P(x + p.x, y + p.y); }
    P operator-(const P &p) const { return P(x - p.x, y - p.y); }
    P operator*(const db &c) const { return P(x * c, y * c); }
    P operator/(const db &c) const { return P(x / c, y / c); }
    T operator^(const P &p) const { return x * p.y - y * p.x; }
    T operator*(const P &p) const { return x * p.x + y * p.y; }
    bool operator==(const P &p) const
    {
        return abs(x - p.x) + abs(y - p.y) < EPS;
    }
    bool operator<(const P &p) const
    {
        return abs(x - p.x) > EPS ? p.x - x > EPS : p.y - y > EPS;
    }

    T norm2() const { return x * x + y * y; }
    db norm() const { return sqrt(norm2()); }
    db ang()
    {
        db a = atan2(y, x);
        if (a < 0) a += 2. * PI;
        return a;
    }
    P unit() { return (*this) / norm(); }
    P perp() { return P(-y, x); }
    P rot(P r) { return P((*this) ^ r, (*this) * r); }
    P rot(db a) { return rot(P(sin(a), cos(a))); }
};

P polar(db r, db a) { return P(r * cos(a), r * sin(a)); }
istream &operator>>(istream &s, P &p) { return s >> p.x >> p.y; }
ostream &operator<<(ostream &s, const P &p)
{
    return s << '(' << p.x << ", " << p.y << ')';
}

db ang(db a)
{
    while (a >= 2. * PI) a -= 2. * PI;
    while (a < 0) a += 2. * PI;
}

```

```

    return a;
}

T turn(P &a, P &b, P &c) { return (b - a) ^ (c - a); }

bool isConvex(vector<P> p)
{
    int n = p.size();
    bool hasPos = false, hasNeg = false, hasCol = false;
    rep(i, n)
    {
        int o = turn(p[i], p[(i + 1) % n], p[(i + 2) % n]);
        if (o > 0) hasPos = true;
        if (o < 0) hasNeg = true;
        if (o == 0) hasCol = true;
    }
    return !(hasPos && hasNeg) && !hasCol;
}

bool half(P &p) { return p.y > 0 || (p.y == 0 && p.x > 0); }

void polarSort(vector<P> &v)
{
    sort(v.begin(), v.end(), [](P &p1, P &p2)
    {
        int h1 = half(p1), h2 = half(p2);
        return h1 != h2 ? h1 > h2 : (p1 ^ p2) > 0;
    });
}

// LINE

struct L
{
    P v; T c;
    L() {} L(P v, T c) : v(v), c(c) {}
    L(T a, T b, T c) : v(P(b, -a)), c(c) {}
    L(P p, P q) : v(q - p), c(v ^ p) {}

    T side(P p) { return (v ^ p) - c; }
    db dist(P p) { return abs(side(p) / v.norm()); }
    db dist2(P p) { return side(p) * side(p) / (db)v.norm2(); }
    L perp(P p) { return L(p, p + v.perp()); }
    L translate(P t) { return L(v, c + (v ^ t)); }
    P proj(P p) { return p - v.perp() * side(p) / v.norm2(); }
    P refl(P p) { return p - v.perp() * 2 * side(p) / v.norm2(); }
};

bool parallel(L l1, L l2) { return abs(l1.v ^ l2.v) < EPS; }

// only if not parallel
P inter(L l1, L l2) { return (l2.v * l1.c - l1.v * l2.c) / (l1.v ^ l2.v); }

L bisector(L l1, L l2, bool in)
{

```

```

    db sign = in ? 1 : -1;
    return L(12.v / 12.v.norm() + 11.v / 11.v.norm() * sign,
            12.c / 12.v.norm() + 11.c / 11.v.norm() * sign);
}

struct HASH // Hashing for integer coordinates lines
{
    ll a, b, c;
    HASH(const P &p1, const P &p2)
    {
        a = p1.y - p2.y, b = p2.x - p1.x;
        c = p1.x * (p2.y - p1.y) - p1.y * (p2.x - p1.x);
        ll sgn = (a < 0 or (a == 0 and b < 0)) ? -1 : 1;
        ll g = __gcd(abs(a), __gcd(abs(b), abs(c))) * sgn;
        a /= g, b /= g, c /= g;
    }
    bool operator<(const HASH &h) const
    {
        return a < h.a or (a == h.a and (b < h.b or (b == h.b and c < h.c)));
    }
};

// SEGMENT

bool inDisk(P &a, P &b, P &p) { return (a - p) * (b - p) <= 0; }

bool onSegment(P &a, P &b, P &p)
{
    return abs(turn(a, b, p)) < EPS && inDisk(a, b, p);
}

bool properInter(P &a, P &b, P &c, P &d, P &out)
{
    T ta = turn(c, d, a), tb = turn(c, d, b),
      tc = turn(a, b, c), td = turn(a, b, d);
    out = (a * tb - b * ta) / (tb - ta);
    return (ta * tb < 0 && tc * td < 0);
}

set<P> inter(P &a, P &b, P &c, P &d)
{
    P out;
    if (properInter(a, b, c, d, out)) return {out};
    set<P> ans;
    if (onSegment(c, d, a)) ans.insert(a);
    if (onSegment(c, d, b)) ans.insert(b);
    if (onSegment(a, b, c)) ans.insert(c);
    if (onSegment(a, b, d)) ans.insert(d);
    return ans;
}

db segPoint(P &a, P &b, P &p)
{
    if ((p - a) * (b - a) >= 0 && (p - b) * (a - b) >= 0)
        return abs(((b - a) ^ (p - a)) / (b - a).norm());
}

```

```

    return min((p - a).norm(), (b - a).norm());
}

db segSeg(P &a, P &b, P &c, P &d)
{
    P aux;
    if (properInter(a, b, c, d, aux)) return 0;
    return min({segPoint(a, b, c), segPoint(a, b, d),
                segPoint(c, d, a), segPoint(c, d, b)});
}

// POLYGONS

db areaTriangle(P &a, P &b, P &c)
{
    return abs((b - a) ^ (c - a)) / 2.;
}

db areaPolygon(vector<P> &p)
{
    db ans = 0; int n = p.size();
    rep(i, n) ans += p[i] ^ p[(i + 1) % n];
    return abs(ans) / 2.;
}

bool above(P &a, P &p) { return p.y >= a.y; }

bool crossesRay(P &a, P &p, P &q)
{
    return (above(a, q) - above(a, p)) * turn(a, p, q) > 0;
}

// if strict, returns false when a is on the boundary
bool inPolygon(vector<P> &p, P &a, bool strict = true)
{
    int c = 0, n = p.size();
    rep(i, n)
    {
        if (onSegment(p[i], p[(i + 1) % n], a)) return !strict;
        c += crossesRay(a, p[i], p[(i + 1) % n]);
    }
    return c & 1;
}

db areaPolygonUnion(vector<vector<P>> &pol) // Slow  $O((NE)^2 \log(NE))$ 
{
    db area = 0;
    rep(i, pol.size()) rep(j, pol[i].size())
    {
        int m = pol[i].size();
        P p1 = pol[i][j], p2 = pol[i][(j + 1) % m];

        vector<pair<db, int>> s; s.emplace_back(1., 0);

        rep(ii, pol.size()) if (ii != i) rep(jj, pol[ii].size())

```

```

{
    int mm = pol[ii].size();
    P p3 = pol[ii][jj], p4 = pol[ii][(jj + 1) % mm];

    db t1 = turn(p1, p2, p3), t2 = turn(p1, p2, p4),
        t3 = turn(p3, p4, p1), t4 = turn(p3, p4, p2);
    if (!t1 && !t2 && (p2 - p1) * (p4 - p3) > 0 && i > ii)
    {
        s.emplace_back((p3 - p1) * (p2 - p1).unit(), 1);
        s.emplace_back((p4 - p1) * (p2 - p1).unit(), -1);
    }
    if (t1 >= 0 && t2 < 0) s.emplace_back(t3 / (t3 - t4), 1);
    if (t1 < 0 && t2 >= 0) s.emplace_back(t3 / (t3 - t4), -1);
}

sort(s.begin(), s.end());

int c = 0;
db last = 0, f = 0;
for (auto e : s)
{
    db now = min(1., max(0., e.ff));
    if (c == 0) f += now - last;
    c += e.ss, last = now;
}

area += (p1 ^ p2) * f;
}

return area;
}

vector<P> convexHull(vector<P> &p)
{
    int n = p.size(), k = 0;
    vector<P> H(2 * n); sort(p.begin(), p.end());
    rep(i, n)
    {
        while (k >= 2 && turn(H[k - 2], H[k - 1], p[i]) <= 0) k--;
        H[k++] = p[i];
    }
    for (int i = n - 2, t = k + 1; i >= 0; i--)
    {
        while (k >= t && turn(H[k - 2], H[k - 1], p[i]) <= 0) k--;
        H[k++] = p[i];
    }
    H.resize(k - 1);
    return H;
}

// MISCELLANEOUS

// Smallest Enclosing circle

P bary(P &A, P &B, P &C, db a, db b, db c)

```

```

{
    return (A * a + B * b + C * c) / (a + b + c);
}

P circum(P &A, P &B, P &C)
{
    db a = (B - C).norm2(), b = (C - A).norm2(), c = (A - B).norm2();
    return bary(A, B, C, a * (b + c - a), b * (c + a - b), c * (a + b - c));
}

pair<P, db> smallestEnclosingCircle(vector<P> &p)
{
    random_shuffle(all(p));
    P c = p[0]; db r = 0; int N = p.size();
    rep(i, N) if (i && (p[i] - c).norm() > r + EPS)
    {
        c = p[i]; r = 0;
        rep(j, i) if ((p[j] - c).norm() > r + EPS)
        {
            c = (p[i] + p[j]) * 0.5;
            r = (p[i] - c).norm();
            rep(k, j) if ((p[k] - c).norm() > r + EPS)
            {
                c = circum(p[i], p[j], p[k]);
                r = (p[k] - c).norm();
            }
        }
    }

    return {c, r};
}

// Closest pair of points from array "a" (mindist: squared mindist)

const int MAXN = 1000010;

int n; T mindist;
pair<P, P> best;
P a[MAXN], t[MAXN];

T sq(T x) { return x * x; }

bool cmpY(P &a, P &b) { return a.y < b.y; }

void update(P &p1, P &p2)
{
    T aux = (p1 - p2).norm2();
    if (aux < mindist) { mindist = aux; best = {p1, p2}; }
}

// sort "a" before usage (P must have default operator<)
void closest(int l, int r)
{
    if (r - l <= 3)
    {

```

```

    repx(i, l, r) repx(j, i + 1, r) update(a[i], a[j]);
    sort(a + l, a + r, cmpY);
    return;
}

int m = (l + r) >> 1, xm = a[m].x;
closest(l, m); closest(m, r);

merge(a + l, a + m, a + m, a + r, t, cmpY);
copy(t, t + r - l, a + l);

int tsz = 0;
repx(i, l, r) if (sq(a[i].x - xm) < mindist)
{
    for (int j = tsz - 1; j >= 0 && sq(a[i].y - t[j].y) < mindist; --j)
        update(a[i], a[j]);
    t[tsz++] = a[i];
}
}

```

3.2 3D

```

const db PI = acos(-1.0L);
const db EPS = 1e-12;

// POINT 3D

struct P
{
    db x, y, z;
    P() {}
    P(db x, db y, db z) : x(x), y(y), z(z) {}

    P operator+(const P &p) const { return P(x + p.x, y + p.y, z + p.z); }
    P operator-(const P &p) const { return P(x - p.x, y - p.y, z - p.z); }
    P operator*(const db &c) const { return P(x * c, y * c, z * c); }
    P operator/(const db &c) const { return P(x / c, y / c, z / c); }
    P operator^(const P &p) const
    {
        return P(y * p.z - z * p.y,
                z * p.x - x * p.z,
                x * p.y - y * p.x);
    }
    db operator*(const P &p) const { return x * p.x + y * p.y + z * p.z; }
    db operator%(const P &p) const
    {
        return acos((*this) * p) / (norm() * p.norm());
    }
}

bool operator==(const P &p) const
{
    return abs(x - p.x) + abs(y - p.y) + abs(z - p.z) < EPS;
}

```

```

}

db norm() const { return sqrt(norm2()); }
db norm2() const { return x * x + y * y + z * z; }
P unit() { return (*this) / norm(); }

};
P polar(db r, db a, db b)
{
    return P(r * cos(a) * cos(b), r * cos(a) * sin(b), r * sin(a));
}
istream &operator>>(istream &s, P &p) { return s >> p.x >> p.y >> p.z; }
ostream &operator<<(ostream &s, const P &p)
{
    return s << '(' << p.x << ", " << p.y << ", " << p.z << ')';
}

// ARCS

bool in_arc(P &a, P &b, P &n, P &p)
{
    db ab = a % b;
    db ap = a % p;
    P c = (a * cos(ap) + (n ^ a) * sin(ap));
    return ab > ap && p == c;
}

bool find_intersection(P &a1, P &b1, P &a2, P &b2, db &angle)
{
    P n1 = (a1 ^ b1).unit(), n2 = (a2 ^ b2).unit(), i = (n1 ^ n2);
    if (i.norm() < EPS) return false;
    i = i.unit() * a1.norm();
    if (in_arc(a1, b1, n1, i) and in_arc(a2, b2, n2, i))
    {
        angle = a1 % i;
        return true;
    }
    i = i * -1.;
    if (in_arc(a1, b1, n1, i) and in_arc(a2, b2, n2, i))
    {
        angle = a1 % i;
        return true;
    }
    return false;
}

```

4 Graphs

4.1 BellmanFord

```

const int INF = 1e9;

```

```

struct Edge
{
    int v, w;
    Edge(int v, int w) : v(v), w(w) {}
};

int n; vi D;
vector<vector<Edge>> G;

bool bellmanFord(int s)
{
    D.assign(n, INF); D[s] = 0;
    rep(i, n - 1) rep(j, n) for (Edge e : G[j])
        D[e.v] = min(D[e.v], D[j] + e.w);

    bool neg = false;
    rep(i, n) for (Edge e : G[i]) if (D[e.v] > D[i] + e.w) neg = true;

    return neg;
}

```

4.2 Dijkstra

```

const int INF = 1e9;

struct Edge
{
    int v, w; // CHECK FOR OVERFLOW
    Edge(int v, int w) : v(v), w(w) {}
    bool operator<(const Edge &e) const { return w > e.w; }
};

int n;
vector<vector<Edge>> G;

int dijkstra(int s, int t)
{
    vi C(n, INF); C[s] = 0; // CHECK FOR OVERFLOW
    priority_queue<Edge> q; q.emplace(s, 0);
    while (!q.empty())
    {
        int u = q.top().v, w = q.top().w; q.pop(); // CHECK FOR OVERFLOW
        if (C[u] < w) continue;
        for (auto e : G[u]) if (C[e.v] > e.w + w)
            C[e.v] = e.w + w, q.emplace(e.v, C[e.v]);
    }
    return C[t];
}

```

4.3 Dinic

```

class Dinic
{
    struct Edge { int to, rev; ll f, c; };
    int n, t_; vector<vector<Edge>> G;
    vi D; vi q, W;

    bool bfs(int s, int t)
    {
        W.assign(n, 0); D.assign(n, -1); D[s] = 0;
        int f = 0, l = 0; q[l++] = s;
        while (f < l)
        {
            int u = q[f++];
            for (const Edge &e : G[u]) if (D[e.to] == -1 && e.f < e.c)
                D[e.to] = D[u] + 1, q[l++] = e.to;
        }
        return D[t] != -1;
    }

    ll dfs(int u, ll f)
    {
        if (u == t_) return f;
        for (int &i = W[u]; i < (int)G[u].size(); ++i)
        {
            Edge &e = G[u][i]; int v = e.to;
            if (e.c <= e.f || D[v] != D[u] + 1) continue;
            ll df = dfs(v, min(f, e.c - e.f));
            if (df > 0) { e.f += df, G[v][e.rev].f -= df; return df; }
        }
        return 0;
    }

public:
    Dinic(int N) : n(N), G(N), D(N), q(N) {}
    void addEdge(int u, int v, ll cap)
    {
        G[u].push_back({v, (int)G[v].size(), 0, cap});
        G[v].push_back({u, (int)G[u].size() - 1, 0, 0}); // cap if bidirectional
    }
    ll maxFlow(int s, int t)
    {
        t_ = t; ll ans = 0;
        while (bfs(s, t)) while (ll dl = dfs(s, LLONG_MAX)) ans += dl;
        return ans;
    }
};

```

4.4 FloydWarshall

```

const int INF = 1e9;

int n;
vector<vi> D;

//D is an adjacency matrix at the beginning
void floydWarshall ()
{
    rep(k, n) rep(i, n) rep(j, n) if (D[i][k] < INF && D[k][j] < INF)
        D[i][j] = min(D[i][j], D[i][k] + D[k][j]);

    // opcional, si hay ciclos negativos
    rep(k, n) rep(i, n) rep(j, n)
        if (D[i][k] < INF && D[k][j] < INF && D[k][k] < 0) D[i][j] = -INF;
}

```

4.5 HeavyLightDecomposition

```

template <class ST, class node>
class HLD
{
    ST st;
    vi A, H, D, R, P;

    int dfs(vector<vi> &G, int u)
    {
        int ans = 1, M = 0, s;
        for (int v : G[u]) if (v != A[u])
        {
            A[v] = u, D[v] = D[u] + 1;
            s = dfs(G, v), ans += s;
            if (s > M) H[u] = v, M = s;
        }
        return ans;
    }
    template <class OP>
    void path(int u, int v, OP op)
    {
        for (; R[u] != R[v]; v = A[R[v]])
        {
            if (D[R[u]] > D[R[v]]) swap(u, v);
            op(P[R[v]], P[v] + 1);
        }
        if (D[u] > D[v]) swap(u, v);
        op(P[u], P[v] + 1); // VALUES ON VERTEX
        // op(P[u] + 1, P[v] + 1); // VALUES ON EDGE
    }

public:
    HLD(vector<vi> &G, int n) : A(n), st(n), D(n), R(n), P(n)
    {

```

```

        H.assign(n, -1); A[0] = -1, D[0] = 0; dfs(G, 0); int p = 0;
        rep(i, n) if (A[i] == -1 || H[A[i]] != i)
            for (int j = i; j != -1; j = H[j]) R[j] = i, P[j] = p++;
    }
    void set(int v, const node &x) { st.set(P[v], x); } // VALUES ON VERTEX
    // void set(int u, int v, const node &x) // VALUES ON EDGE
    // {
    //     if (D[u] > D[v]) swap(u, v);
    //     st.set(P[v], x);
    // }
    void update(int u, int v, const node& x) // OPTIONAL FOR RANGE
        UPDATES
    { path(u, v, [this, &x](int l, int r) { st.update(l, r, x); }); }
    node query(int u, int v)
    {
        node ans = node();
        path(u, v, [this, &ans](int l, int r) { ans = node(ans, st.query(l, r)); });
        return ans;
    }
};

// USAGE: HLD<ST<Node>, Node> hld(G, N);

//// NON COMMUTATIVE QUERIES :

template <class ST, class node>
class HLD
{
    ST st;
    vi A, H, D, R, P;

    int dfs(vector<vi> &G, int u)
    {
        int ans = 1, M = 0, s;
        for (int v : G[u]) if (v != A[u])
        {
            A[v] = u, D[v] = D[u] + 1;
            s = dfs(G, v), ans += s;
            if (s > M) H[u] = v, M = s;
        }
        return ans;
    }

public:
    node path(int u, int v)
    {
        node ans1, ans2; bool d = 0;
        for (; R[u] != R[v]; v = A[R[v]])
        {
            if (D[R[u]] > D[R[v]]) swap(u, v), d = !d;
            if (d) ans1 = node(st.query(P[R[v]], P[v] + 1), ans1);
            else ans2 = node(st.query(P[R[v]], P[v] + 1), ans2);
        }
        if (D[u] > D[v]) swap(u, v), d = !d;

```



```

    if (d) ans1 = node(st.query(P[u], P[v] + 1), ans1);
    else ans2 = node(st.query(P[u], P[v] + 1), ans2);
    ans1.sw(); return node(ans1, ans2);
}
HLD(vector<vi> &G, int n) : A(n), st(n), D(n), R(n), P(n)
{
    H.assign(n, -1); A[0] = -1, D[0] = 0; dfs(G, 0); int p = 0;
    rep(i, n) if (A[i] == -1 || H[A[i]] != i)
        for (int j = i; j != -1; j = H[j]) R[j] = i, P[j] = p++;
}
void set(int v, const node &x) { st.set(P[v], x); }
};

```

4.6 HopcroftKarp

```

class Hopcroft
{
    vi U, dist;
    vector<vi> g;
    int inf = 1e9;

    bool bfs() {
        queue<int> q;
        for (int u : U) if (match[u] == nil) dist[u] = 0, q.push(u);
        else dist[u] = inf;
        dist[nil] = inf;
        while (!q.empty())
        {
            int u = q.front(); q.pop();
            if (u != nil) for (int v : g[u]) if (dist[match[v]] == inf)
            {
                dist[match[v]] = dist[u] + 1;
                q.push(match[v]);
            }
        }
        return (dist[nil] != inf);
    }

    bool dfs(int u) {
        if (u == nil)
            return true;
        for (int v : g[u]) if (dist[match[v]] == dist[u]+1 and dfs(match[v]))
        {
            match[v] = u, match[u] = v;
            return true;
        }
        dist[u] = inf;
        return false;
    }
}

public:

```

```

vi match;
int nil, isPerfect, matchSize = 0;

// gg is a bidirectional graph, UU has the nodes in the left partition
Hopcroft(vector<vi> &gg, vi &UU)
{
    g = gg; U = UU; nil = g.size();
    match.assign(g.size() + 1, nil);
    dist.assign(g.size() + 1, inf);
    while (bfs()) for (int u : U) if (match[u] == nil and dfs(u))
        matchSize++;
    isPerfect = (matchSize == U.size() and g.size() == U.size() * 2);
}
};

```

4.7 Hungarian

```

template<class T>
class Hungarian
{
    T inf = numeric_limits<T>::max() / 2;
    bool maxi, swapped = false;
    vector<vector<T>> cost;
    vector<T> u, v;
    vi p, way;
    int l, r;

public:
    // left/right == partition sizes
    Hungarian(int left, int right, bool maximizing)
    {
        l = left, r = right, maxi = maximizing;
        if (swapped = l > r) swap(l, r);
        cost.assign(l + 1, vector<T>(r + 1, 0));
        u.assign(l + 1, 0); v.assign(r + 1, 0);
        p.assign(r + 1, 0); way.assign(r + 1, 0);
    }

    void add_edge(int l, int r, T w)
    {
        assert(l and r); // indices start from 1 !!
        if (swapped) swap(l, r);
        cost[l][r] = maxi ? -w : w;
    }

    // execute after all edges were added
    void calculate()
    {
        rep(x, 1, l + 1)
        {
            vector<bool> used(r+1, false);

```

```

vector<T> minv(r+1, inf);
int j0 = 0, p[0] = i;

while (p[j0])
{
    int j1, i0 = p[j0], used[j0] = true;
    T delta = inf;
    repx(j, 1, r + 1) if (not used[j])
    {
        T cur = cost[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;
    }
    rep(j, r + 1)
    {
        if (used[j]) u[p[j]] += delta, v[j] -= delta;
        else minv[j] -= delta;
    }
    j0 = j1;

    while (j0) p[j0] = p[way[j0]], j0 = way[j0];
}

// execute after executing calculate()
T answer() { return maxi ? v[0] : -v[0]; }

bool are_matched(int l, int r)
{
    if (swapped) swap(l, r);
    return p[r] == l;
}
};

```

4.8 LCA

```

struct LCA
{
    vector<vi> G;
    int N, LOG; vi A, D;
    int &anc(int u, int l) { return A[l * N + u]; }
    LCA(vector<vi> &G, int N, int root) : G(G), N(N)
    {
        D.assign(N, -1); A.resize(N * (LOG + 1));
        dfs(root, -1, 0), LOG = 31 - __builtin_clz(N);
        rep(l, LOG + 1) if (l) rep(u, N)
        {
            int a = anc(u, l - 1);
            anc(u, l) = (a == -1 ? -1 : anc(a, l - 1));
        }
    }
};

```

```

}
void dfs(int u, int p, int depth)
{
    anc(u, 0) = p, D[u] = depth;
    for (int v : G[u]) if (D[v] == -1) dfs(v, u, depth + 1);
}
int raise(int u, int k)
{
    for (int l = 0; k; l++, k >>= 1) if (k & 1) u = anc(u, l);
    return u;
}
int lca(int u, int v)
{
    if (D[u] < D[v]) swap(u, v);
    u = raise(u, D[u] - D[v]);
    if (u == v) return u;
    for (int l = LOG; l >= 0; l--) if (anc(u, l) != anc(v, l))
        u = anc(u, l), v = anc(v, l);
    return anc(u, 0);
}
int dist(int u, int v) { return D[u] + D[v] - 2 * D[lca(u, v)]; }
int raise_in_path(int u, int v, int k)
{
    if (D[u] - D[lca(u, v)] >= k) return raise(u, k);
    return raise(v, dist(u, v) - k);
}
int add_child(int p, int u)
{
    G[p].pb(u);
    D[u] = D[p] + 1, anc(u, 0) = p;
    rep(l, LOG) if (l)
    {
        p = anc(p, l - 1);
        if (p == -1) break;
        anc(u, l) = p;
    }
}
};

```

4.9 MST

```

// PRIM //

struct Edge
{
    int u, v, w;
    Edge(int u, int v, int w) : u(u), v(v), w(w) {}
    bool operator>(const Edge &e) const { return w > e.w; }
};

int find_mst(vector<vector<Edge>> &g, vector<vector<Edge>> &T)

```

```

{
    int n = g.size();
    vector<bool> V(n, 0);
    T.assign(n, {});

    int ans = 0, c = 1; V[0] = 1;
    priority_queue<Edge, vector<Edge>, greater<Edge>> q;
    for (Edge &p : g[0]) q.emplace(0, p.v, p.w);

    while (!q.empty())
    {
        Edge e = q.top(); q.pop();

        if (V[e.v]) continue;

        int u = e.u, v = e.v, w = e.w;
        V[v] = true, ans += w;
        T[u].eb(u, v, w);
        T[v].eb(v, u, w);

        if (++c == n) break;

        for (Edge &p : g[v]) if (!V[p.v]) q.emplace(v, p.v, p.w);
    }

    return ans;
}

```

4.10 MinCostMaxFlow

```

template <class T>
class MCMF
{
    typedef pair<T, T> pTT;
    T INF = numeric_limits<T>::max();
    struct Edge
    {
        int v; T c, w;
        Edge(int v, T c, T w) : v(v), c(c), w(w) {}
    };

    int n; vector<vi> E;
    vector<Edge> L; vi F; vector<T> D, P; vector<bool> V;

    bool dij(int s, int t)
    {
        D.assign(n, INF); F.assign(n, -1); V.assign(n, false);
        D[s] = 0;
        rep(_, n)
        {
            int best = -1;

```

```

            rep(i, n) if (!V[i] && (best == -1 || D[best] > D[i])) best = i;
            if (D[best] >= INF) break;
            V[best] = true;
            for (int e : E[best])
            {
                Edge ed = L[e];
                if (ed.c == 0) continue;
                T toD = D[best] + ed.w + P[best] - P[ed.v];
                if (toD < D[ed.v]) D[ed.v] = toD, F[ed.v] = e;
            }
        }
        return D[t] < INF;
    }

    pTT augment(int s, int t)
    {
        pTT flow(L[F[t]].c, 0);
        for (int v = t; v != s; v = L[F[v] ^ 1].v)
            flow.ff = min(flow.ff, L[F[v]].c), flow.ss += L[F[v]].w;
        for (int v = t; v != s; v = L[F[v] ^ 1].v)
            L[F[v]].c -= flow.ff, L[F[v] ^ 1].c += flow.ff;
        return flow;
    }

public:
    MCMF(int n) : n(n), E(n), D(n), P(n, 0), V(n, 0) {}
    pTT mcmf(int s, int t)
    {
        pTT ans(0, 0);
        if (!dij(s, t)) return ans;
        rep(i, n) if (D[i] < INF) P[i] += D[i];
        while (dij(s, t))
        {
            auto flow = augment(s, t);
            ans.ff += flow.ff, ans.ss += flow.ff * flow.ss;
            rep(i, n) if (D[i] < INF) P[i] += D[i];
        }
        return ans;
    }

    void addEdge(int u, int v, T c, T w)
    {
        E[u].pb(L.size()); L.eb(v, c, w);
        E[v].pb(L.size()); L.eb(u, 0, -w);
    }
};

```

4.11 SCC

```

int N, id;
vector<vi> G; // Directed Graph
vi D, L, I; stack<int> S;

```

```

void dfs(int u)
{
    D[u] = L[u] = id++, I[u] = 1; S.push(u);
    for (int v : G[u])
    {
        if (D[v] == -1) { dfs(v); L[u] = min(L[v], L[u]); }
        else if (I[v]) L[u] = min(L[v], L[u]);
    }
    if (L[u] == D[u]) while (1) // SCC FOUND
    {
        int x = S.top(); S.pop(); I[x] = 0;
        if (x == u) break;
    }
}

void find_sccs()
{
    D.assign(N, -1); L.resize(N); I.assign(N, 0);
    id = 0; rep(u, N) if (D[u] == -1) dfs(u);
}

```

4.12 Tarjan

```

vector<vi> G;
vi D, L;

void dfs(int u, int p, int d)
{
    D[u] = L[u] = d;
    for (int v : G[u]) if (v != p)
    {
        if (D[v] == -1)
        {
            dfs(v, u, d + 1);
            if (L[v] > D[u]) {} // (u - v) cut edge
            L[u] = min(L[u], L[v]);
        }
        else L[u] = min(L[u], D[v]);
    }
}

int rc = 0;
void dfs(int u, int p, int d)
{
    D[u] = L[u] = d;
    for (int v : G[u]) if (v != p)
    {
        if (D[v] == -1)
        {
            dfs(v, u, d + 1);
            if ((p == -1 && ++rc == 2) || (p != -1 && L[v] >= d)) {} // u is AP

```

```

        L[u] = min(L[u], L[v]);
    }
    else if (D[v] < d) L[u] = min(L[u], D[v]);
}

stack<pi> S;
void dfs(int u, int p, int d)
{
    D[u] = L[u] = d;
    for (int v : G[u]) if (v != p)
    {
        if (D[v] == -1)
        {
            S.emplace(u, v); dfs(v, u, d + 1);
            if (p == -1 or L[v] >= d) while (1) // BCC found
            {
                pi e = S.top(); S.pop();
                if (e == make_pair(u, v)) break;
            }
            L[u] = min(L[u], L[v]);
        }
        else if (D[v] < d) { S.emplace(u, v); L[u] = min(L[u], D[v]); }
    }
}

```

4.13 Toposort

```

int N; vi V, S;
vector<vi> G;

void dfs(int u)
{
    V[u] = 1;
    for (int v : G[u]) if (!V[v]) dfs(v);
    S.pb(u);
}

void topo_sort()
{
    V.assign(N, 0); S.clear();
    rep(i, N) if (!V[i]) dfs(i);
}

```

5 Math

5.1 CRT

```
#include <Euclid.cpp>

pl CRT(pl a, pl b)
{
    if (a.ss < b.ss) swap(a, b);
    ll x, y; tie(x, y) = euclid(a.ss, b.ss);
    ll g = a.ss * x + b.ss * y, l = a.ss / g * b.ss;
    if ((b.ff - a.ff) % g) return {-1, -1}; // no solution
    x = (b.ff - a.ff) % b.ss * x % b.ss / g * a.ss + a.ff;
    return {x + (x < 0) * l, l};
}

pl CRT(vector<pl> &v)
{
    int N = v.size(); pl ans = v[0];
    rep(i, N) if (i) ans = CRT(ans, v[i]);
    return ans;
}
```

5.2 Euclid

```
// find (x, y) such that Ax + By = gcd(A, B), and |Ax|, |By| <= AB/gcd(A, B)
pl euclid(ll A, ll B)
{
    if (!B) return {1, 0};
    pl p = euclid(B, A % B);
    return {p.ss, p.ff - (A / B) * p.ss};
}

// find x in [0, M) such that Ax = 1 mod M
ll minv(ll A, ll M)
{
    pl p = euclid(A, M);
    assert(p.ff * A + p.ss * M == 1);
    return p.ff + (p.ff < 0) * M;
}

// find (x, y)'s such that Ax + By = R where R is multiple of gcd(A, B);
pair<pl, pl> diophantine(ll A, ll B, ll R)
{
    ll g = __gcd(A, B), x, y; A /= g, B /= g, R /= g;
    tie(x, y) = euclid(A, B); x *= R, y *= R;
    assert(A * x + B * y == R);
    return {{x, y}, {-B, A}}; // solutions: p+t*ans.snd
}
```

5.3 FFT

```
#define PI acos(-1.0L)

typedef complex<double> C;
void fft(vector<C> &a)
{
    int n = a.size(), L = 31 - __builtin_clz(n);
    static vector<complex<long double>> R(2, 1);
    static vector<C> rt(2, 1);
    for (static int k = 2; k < n; k *= 2)
    {
        R.resize(n); rt.resize(n);
        auto x = polar(1.0L, PI / k);
        rep(i, k, 2 * k) rt[i] = R[i] = i & 1 ? R[i / 2] * x : R[i / 2];
    }
    vector<int> rev(n);
    rep(i, n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
    rep(i, n) if (i < rev[i]) swap(a[i], a[rev[i]]);
    for (int k = 1; k < n; k *= 2) for (int i = 0; i < n; i += 2 * k) rep(j, k)
    {
        auto x = (double *)&rt[j + k], y = (double *)&a[i + j + k];
        C z(x[0] * y[0] - x[1] * y[1], x[0] * y[1] + x[1] * y[0]);
        a[i + j + k] = a[i + j] - z, a[i + j] += z;
    }
}

vd conv(const vd &a, const vd &b)
{
    if (a.empty() || b.empty()) return {};
    vd res(a.size() + b.size() - 1);
    int L = 32 - __builtin_clz(res.size()), n = 1 << L;
    vector<C> in(n), out(n);
    copy(a.begin(), a.end(), in.begin());
    rep(i, b.size()) in[i].imag(b[i]);
    fft(in); for (auto &x : in) x *= x;
    rep(i, n) out[i] = in[-i & (n - 1)] - conj(in[i]);
    fft(out); rep(i, res.size()) res[i] = imag(out[i]) / (4 * n);
    return res;
}

vl convMod(const vl &a, const vl &b, int M)
{
    if (a.empty() || b.empty()) return {};
    vl res(a.size() + b.size() - 1);
    int B = 32 - __builtin_clz(res.size()), n = 1 << B, cut = int(sqrt(M));
    vector<C> L(n), R(n), outs(n), outl(n);
    rep(i, a.size()) L[i] = C((int)a[i] / cut, (int)a[i] % cut);
    rep(i, b.size()) R[i] = C((int)b[i] / cut, (int)b[i] % cut);
    fft(L), fft(R);
    rep(i, n)
    {
        int j = -i & (n - 1);
        outl[j] = (L[i] + conj(L[j])) * R[i] / (2.0 * n);
        outs[j] = (L[i] - conj(L[j])) * R[i] / (2.0 * n) / 1i;
    }
    fft(outl), fft(outs);
}
```

```

rep(i, res.size())
{
    ll av = ll(real(outl[i]) + .5), cv = ll(imag(outs[i]) + .5);
    ll bv = ll(imag(outl[i]) + .5) + ll(real(outs[i]) + .5);
    res[i] = ((av % M * cut + bv) % M * cut + cv) % M;
}
return res;
}

```

5.4 Modular

```

const ll MOD = 1e9 + 7;

inline ll add(ll x, ll y, ll M = MOD) { return (x + y) % M; }
inline ll mul(ll x, ll y, ll M = MOD) { return (x * y) % M; }

ll binPow(ll a, ll b, const ll M)
{
    if (b == 0) return 1;
    ll r = binPow(a, b / 2, M); r = mul(r, r, M);
    return (b & 1) ? mul(r, a, M) : r;
}

// If multiplication goes over ll

ul modMul(ul a, ul b, const ul M)
{
    ll r = a * b - M * (ul)((db)a * b / M);
    return r + ((r < 0) - (r >= (ll)M)) * M;
}

ul modPow(ul a, ul b, const ul M)
{
    if (b == 0) return 1;
    ul r = modPow(a, b / 2, M); r = modMul(r, r, M);
    return b & 1 ? modMul(r, a, M) : r;
}

```

5.5 ModularBinomial

```

#include <Euclid.cpp>
#include <Modular.cpp>

const ll MOD = 1e9 + 7;
const int MAXN = 1000;

// DP bottom-up
ll C[MAXN + 1][MAXN + 1];
void init_C()

```

```

{
    rep(m, MAXN + 1) if (m)
    {
        C[m][0] = C[m][m] = 1;
        rep(k, m) if (k) C[m][k] = add(C[m - 1][k], C[m - 1][k - 1]);
    }
}

// By definition and euclid
ll F[MAXN];
ll choose(int n, int k)
{
    return mul(F[n], minv(mul(F[k], F[n - k]), MOD));
}

// Multinomial Coefficient
ll multinomial(vector<int> K)
{
    ll n = 0, ans = 1;
    for (int k : K) n += k, ans = mul(ans, choose(n, k));
    return ans;
}

```

5.6 Primality

```

#include <Modular.cpp>

bool prime(ul n)
{
    if (n < 2 || n % 6 % 4 != 1) return n - 2 < 2;
    ul A[] = {2, 325, 9375, 28178, 450775, 9780504, 1795265022};
    ul s = __builtin_ctzll(n - 1), d = n >> s;
    for (int a : A)
    {
        ul p = modPow(a, d, n), i = s;
        while (p != 1 && p != n - 1 && a % n && i--) p = modMul(p, p, n);
        if (p != n - 1 && i != s) return 0;
    }
    return 1;
}

```

5.7 PrimeFactorization

```

// Naive with precomputed primes

vi P;
vi factor(int x)
{
    vi ans;

```

```

for (int d : P)
{
    if (d * d > x) break;
    while (x % d == 0)
    {
        ans.pb(d);
        if ((x /= d) == 1) return ans;
    }
}
if (x > 1) ans.pb(x);
return ans;
}

// Pollard Rho (  $O(n^{1/4})$  )

#include <Primalty.cpp>

ul pollard(ul n) // return some nontrivial factor of n
{
    auto f = [n](ul x) { return modMul(x, x, n) + 1; };
    ul x = 0, y = 0, t = 30, prd = 2, i = 1, q;
    while (t++ % 40 || __gcd(prd, n) == 1)
    {
        if (x == y) x = ++i, y = f(x);
        if ((q = modMul(prd, max(x, y) - min(x, y), n)) prd = q;
        x = f(x), y = f(f(y));
    }
    return __gcd(prd, n);
}

void rec(ul n, map<ul, int> &C)
{
    if (n == 1) return;
    if (prime(n)) { C[n]++; return; }
    ul u = pollard(n); rec(u, C), rec(n / u, C);
}

vector<pair<ul, int>> factor(ul n)
{
    map<ul, int> C; rec(n, C);
    return vector<pair<ul, int>>(all(C));
}

```

5.8 Sieve

```

template<int SZ> struct Sieve
{
    bitset<SZ> I; vi P;
    Sieve()
    {
        I.set();

```

```

        for (int i = 2; i * i < SZ; i++) if (I[i])
            for (int j = i * i; j < SZ; j += i) I[j] = 0;
        rep(i, SZ) if (i > 1 && I[i]) P.pb(i);
    }
};

```

```
Sieve<320000> S;
```

6 Strings

6.1 AhoCorasick

```

struct AC
{
    static const int MAX = 1e4, ASZ = 26;
    int N[MAX][ASZ] = {0}, L[MAX] = {0}, E[MAX] = {0}, c = 0;
    void add(string s)
    {
        int p = 0;
        for (char l : s)
        {
            int t = l - 'a';
            if (!N[p][t]) N[p][t] = ++c;
            p = N[p][t];
        } E[p] = 1;
    }
    void init()
    {
        queue<int> q; q.push(0); L[0] = -1;
        while (!q.empty())
        {
            int p = q.front(); q.pop();
            rep(c, ASZ)
            {
                int u = N[p][c]; if (!u) continue;
                L[u] = L[p] == -1 ? 0 : N[L[p]][c], q.push(u);
            }
            if (p) rep(c, ASZ) if (!N[p][c]) N[p][c] = N[L[p]][c];
        }
    }
};

```

6.2 Hash

```

struct RH
{
    int B = 1777771, M[2] = {999727999, 1070777777}, P[2] = {325255434, 10018302};

```

```

vi H[2], I[2];
RH(string &s)
{
    int N = s.size(); rep(k, 2)
    {
        H[k].resize(N + 1), I[k].resize(N + 1);
        H[k][0] = 0, I[k][0] = 1; ll b = 1;
        rep(i, N + 1) if (i)
        {
            H[k][i] = (H[k][i - 1] + b * s[i - 1]) % M[k];
            I[k][i] = (1LL * I[k][i - 1] * P[k]) % M[k];
            b = (b * B) % M[k];
        }
    }
    ll get(int l, int r) // inclusive - exclusive
    {
        ll h0 = (H[0][r] - H[0][l] + M[0]) % M[0];
        h0 = (1LL * h0 * I[0][l]) % M[0];
        ll h1 = (H[1][r] - H[1][l] + M[1]) % M[1];
        h1 = (1LL * h1 * I[1][l]) % M[1];
        return (h0 << 32) | h1;
    }
};

```

6.3 KMP

```

int match(string &p, string &t)
{
    int n = p.size(), m = t.size(), L[n]; L[0] = 0;
    rep(j, n - 1)
    {
        int i = L[j]; while (p[i] != p[j + 1] && i) i = L[i - 1];
        L[j + 1] = (p[i] == p[j + 1] ? i + 1 : 0);
    }
    int ans = 0, i = 0; rep(j, m)
    {
        while (p[i] != t[j] && i) i = L[i - 1];
        if (p[i] == t[j] && ++i == n) i = L[n - 1], ans++;
    }
    return ans;
}

```

6.4 Manacher

```

int n;
string s;

```

```

int main()
{
    vi d1(n); // odd sized palindromes
    for (int i = 0, l = 0, r = -1; i < n; i++) {
        int k = (i > r) ? 1 : min(d1[l + r - i], r - i + 1);
        while (0 <= i - k && i + k < n && s[i - k] == s[i + k]) k++;
        d1[i] = k--;
        if (i + k > r) l = i - k, r = i + k;
    }
    vi d2(n); // even sized palindromes (center to the right)
    for (int i = 0, l = 0, r = -1; i < n; i++) {
        int k = (i > r) ? 0 : min(d2[l + r - i + 1], r - i + 1);
        while (0 <= i - k - 1 && i + k < n && s[i - k - 1] == s[i + k]) k++;
        d2[i] = k--;
        if (i + k > r) l = i - k - 1, r = i + k;
    }
}

```

6.5 SuffixArray

```

struct SA
{
    int n; vi C, R, R_, sa, sa_, lcp;
    inline int gr(int i) { return i < n ? R[i] : 0; }
    void csort(int maxv, int k)
    {
        C.assign(maxv + 1, 0); rep(i, n) C[gr(i + k)]++;
        repx(i, 1, maxv + 1) C[i] += C[i - 1];
        for (int i = (int)n - 1; i >= 0; i--) sa[--C[gr(sa[i] + k)]] = sa[i];
        sa.swap(sa_);
    }
    void getSA(vi& s)
    {
        R = R_ = sa = sa_ = vi(n); rep(i, n) sa[i] = i;
        sort(all(sa), [&s](int i, int j) { return s[i] < s[j]; });
        int r = R[sa[0]] = 1;
        repx(i, 1, n) R[sa[i]] = (s[sa[i]] != s[sa[i - 1]]) ? ++r : r;
        for (int h = 1; h < n && r < n; h <= 1)
        {
            csort(r, h); csort(r, 0); r = R[sa[0]] = 1;
            repx(i, 1, n)
            {
                if (R[sa[i]] != R[sa[i - 1]] || gr(sa[i] + h) != gr(sa[i - 1] + h))
                    r++;
                R[sa[i]] = r;
            }
            R.swap(R_);
        }
    }
    void getLCP(vi &s)
    {
        lcp.assign(n, 0); int k = 0;

```



```

rep(i, n)
{
    int r = R[i] - 1;
    if (r == n - 1) { k = 0; continue; }
    int j = sa[r + 1];
    while (i + k < n && j + k < n and s[i + k] == s[j + k]) k++;
    lcp[r] = k; if (k) k--;
}
}
SA(vi &s) { n = s.size(); getSA(s); getLCP(s); }
};

```

6.6 SuffixAutomaton

```

// vector implementation
struct SA
{
    int sz, l; vi L, Lk, S, C, T;
    vector<vi> N, Ilk;

    SA(string s, int n) : L(2 * n), Lk(2 * n), C(2 * n), N(2 * n, vi(26, -1))
    {
        l = L[0] = 0, Lk[0] = -1, sz = 1; int p;
        for (char c : s) p = extend(c - 'A');
        Ilk.resize(sz); S.assign(sz, -1);
        rep(i, sz) if (i) Ilk[Lk[i]].pb(i);
        T.assign(sz, 0); while (p != -1) T[p] = 1, p = Lk[p];
    }
    int extend(char c)
    {
        int cur = sz++, p = 1; C[cur] = 0, L[cur] = L[l] + 1;
        while (p != -1 && N[p][c] == -1) N[p][c] = cur, p = Lk[p];
        if (p == -1) { Lk[cur] = 0, l = cur; return cur; }
        int q = N[p][c];
        if (L[p] + 1 == L[q]) { Lk[cur] = q, l = cur; return cur; }
        int w = sz++; C[w] = 1, L[w] = L[p] + 1, Lk[w] = Lk[q], N[w] = N[q];
        while (p != -1 && N[p][c] == q) N[p][c] = w, p = Lk[p];
        Lk[q] = Lk[cur] = w, l = cur; return cur;
    }
    int size(int p)
    {
        if (S[p] != -1) return S[p];
        for (int i : Ilk[p]) S[p] += size(i);
        return S[p] += (1 - C[p]) + 1;
    }
};

// 101 vector implementation
struct SA
{
    int sz, l; vi L, Lk;

```

```

vector<vi> N, Ilk;

SA(string s, int n) : L(2 * n), Lk(2 * n), N(2 * n, vi(26, -1))
{
    l = L[0] = 0, Lk[0] = -1, sz = 1; int p;
    for (char c : s) p = extend(c - 'A');
}
int extend(char c)
{
    int cur = sz++, p = 1; L[cur] = L[l] + 1;
    while (p != -1 && N[p][c] == -1) N[p][c] = cur, p = Lk[p];
    if (p == -1) { Lk[cur] = 0, l = cur; return cur; }
    int q = N[p][c];
    if (L[p] + 1 == L[q]) { Lk[cur] = q, l = cur; return cur; }
    int w = sz++; L[w] = L[p] + 1, Lk[w] = Lk[q], N[w] = N[q];
    while (p != -1 && N[p][c] == q) N[p][c] = w, p = Lk[p];
    Lk[q] = Lk[cur] = w, l = cur; return cur;
}
};

// 101 map implementation
struct SA
{
    int sz, l; vi L, Lk;
    vector<map<char, int>> N;

    SA(string s, int n) : L(2 * n), Lk(2 * n), N(2 * n)
    {
        l = L[0] = 0, Lk[0] = -1, sz = 1;
        for (char c : s) extend(c);
    }
    void extend(char c)
    {
        int cur = sz++, p = 1; L[cur] = L[l] + 1;
        while (p != -1 && !N[p].count(c)) N[p][c] = cur, p = Lk[p];
        if (p == -1) { Lk[cur] = 0, l = cur; return; }
        int q = N[p][c];
        if (L[p] + 1 == L[q]) { Lk[cur] = q, l = cur; return; }
        int w = sz++; L[w] = L[p] + 1, Lk[w] = Lk[q], N[w] = N[q];
        while (p != -1 && N[p][c] == q) N[p][c] = w, p = Lk[p];
        Lk[q] = Lk[cur] = w, l = cur;
    }
};

```

6.7 Trie

```

struct Trie
{
    static const int MAX = 1e6;
    int N[MAX][26] = {0}, S[MAX] = {0}, c = 0;
    void add(string s, int a = 1)

```

```

{
    int p = 0; S[p] += a;
    for (char l : s)
    {
        int t = l - 'a';
        if (!N[p][t]) N[p][t] = ++c;
        S[p] = N[p][t] += a;
    }
}
};

struct TrieXOR
{
    static const int MAX = 1e6;
    int N[MAX][2] = {0}, S[MAX] = {0}, c = 0;
    void add(int x, int a = 1)
    {
        int p = 0; S[p] += a;
        rep(i, 31)
        {
            int t = (x >> (30 - i)) & 1;
            if (!N[p][t]) N[p][t] = ++c;
            S[p] = N[p][t] += a;
        }
    }
    int get(int x)
    {
        if (!S[0]) return -1;
        int p = 0; rep(i, 31)
        {
            int t = ((x >> (30 - i)) & 1) ^ 1;
            if (!N[p][t] || !S[N[p][t]]) t ^= 1;
            p = N[p][t]; if (t) x ^= (1 << (30 - i));
        }
        return x;
    }
};

```

7 Structures

7.1 FenwickTree

```

// 1 - indexed / inclusive - inclusive
struct FT
{
    vi t;
    FT(int N) { t.resize(N + 1, 0); }
    int query(int i)
    {
        int ans = 0;
        for (; i; i -= i & (-i)) ans += t[i];
    }
};

```

```

        return ans;
    }
    int query(int i, int j) { return query(j) - query(i - 1); }
    void update(int i, int v)
    {
        int s = query(i, i); // Sets
        for (; i < t.size(); i += i & (-i)) t[i] += v - s;
    }
    void update(int i, int j, int v)
    {
        update(i, v); update(j + 1, -v);
    }
};

```

7.2 FenwickTree2D

```

// 0 - indexed / inclusive - inclusive
template <class T>
class FT2D
{
    vector<vector<T>>> t;
    int n, m;

public:
    FT2D() {}
    FT2D(int n, int m) : n(n), m(m) { t.assign(n, vector<T>(m, 0)); }

    void add(int r, int c, T value)
    {
        for (int i = r; i < n; i |= i + 1)
            for (int j = c; j < m; j |= j + 1)
                t[i][j] += value;
    }

    T sum(int r, int c)
    {
        T res = 0;
        for (int i = r; i >= 0; i = (i & (i + 1)) - 1)
            for (int j = c; j >= 0; j = (j & (j + 1)) - 1)
                res += t[i][j];
        return res;
    }

    T sum(int r1, int c1, int r2, int c2)
    {
        return sum(r2, c2) - sum(r1 - 1, c2) - sum(r2, c1 - 1) +
            sum(r1 - 1, c1 - 1);
    }

    T get(int r, int c) { return sum(r, c, r, c); }
};

```

```
void set(int r, int c, T value) { add(r, c, -get(r, c) + value); }
};
```

7.3 LineContainer

```
struct Line
{
    mutable ll k, m, p;
    bool operator<(const Line &o) const { return k < o.k; }
    bool operator<(ll x) const { return p < x; }
};

// (for doubles, use inf = 1/.0, div(a,b) = a/b)
struct LineContainer : multiset<Line, less<>>
{
    const ll inf = LLONG_MAX;
    ll div(ll a, ll b) { return a / b - ((a ^ b) < 0 && a % b); }
    bool isect(iterator x, iterator y)
    {
        if (y == end()) { x->p = inf; return false; }
        if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
        else x->p = div(y->m - x->m, x->k - y->k);
        return x->p >= y->p;
    }
    void add(ll k, ll m)
    {
        auto z = insert({k, m, 0}), y = z++, x = y;
        while (isect(y, z)) z = erase(z);
        if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
        while ((y = x) != begin() && (--x)->p >= y->p) isect(x, erase(y));
    }
    ll query(ll x)
    {
        assert(!empty());
        auto l = *lower_bound(x);
        return l.k * x + l.m;
    }
};
```

7.4 MoQueries

```
int S; // Size of buckets (generally sqrt(N))

struct Query
{
    int l, r, id;
    Query(int l, int r, int id) : l(l), r(r), id(id) {}
    bool operator<(Query &q)
```

```
{
    return l / S < q.l / S or (l / S == q.l / S and r < q.r);
}
};
```

7.5 PersistentSegmentTree

```
struct Node
{
    int v, l, r;
    Node() : v(0), l(0), r(0) {}
    Node(int v) : v(v) {}
    Node(const Node &a, const Node &b, int l, int r) : l(l), r(r)
    { v = a.v + b.v; }
};

// 0 - indexed / inclusive - inclusive
template <class node>
struct PST
{
    int cnt = 0, n, rc = 0;
    vector<node> st; vi rt;

    node query(int u, int a, int b, int i, int j)
    {
        if (j < a || b < i) return node();
        if (a <= i && j <= b) return st[u];
        int m = (i + j) / 2, l = st[u].l, r = st[u].r;
        return node(query(l, a, b, i, m), query(r, a, b, m + 1, j), l, r);
    }
    int update(int u, int p, node v, int i, int j)
    {
        if (j < p || p < i) return u;
        st[cnt] = st[u]; int x = cnt++, m = (i + j) / 2;
        if (i == j) { st[x] = v; return x; }
        int l = st[x].l = update(st[x].l, p, v, i, m);
        int r = st[x].r = update(st[x].r, p, v, m + 1, j);
        st[x] = node(st[l], st[r], l, r); return x;
    }
    int build(vector<node> &arr, int i, int j)
    {
        int u = cnt++, m = (i + j) / 2;
        if (i == j) { st[u] = arr[i]; return u; }
        int l = st[u].l = build(arr, i, m);
        int r = st[u].r = build(arr, m + 1, j);
        st[u] = node(st[l], st[r], l, r); return u;
    }

    PST(vector<node> &arr) : st(1e7), rt(1e5)
    { n = arr.size(); rt[rc++] = build(arr, 0, n - 1); }
```

```

void update(int t, int p, node v) { rt[rc++] = update(rt[t], p, v, 0, n - 1);
}
node query(int t, int a, int b) { return query(rt[t], a, b, 0, n - 1); }
};

// Init with null's (no build) (requires l = r = 0 as default)
template <class node>
struct PST
{
    int cnt = 1, n, rc = 1;
    vector<node> st; vi rt;

    node query(int u, int a, int b, int i, int j)
    {
        if (j < a || b < i) return node();
        if (a <= i && j <= b) return st[u];
        int m = (i + j) / 2, l = st[u].l, r = st[u].r;
        return node(query(l, a, b, i, m), query(r, a, b, m + 1, j), l, r);
    }
    int update(int u, int p, node v, int i, int j)
    {
        if (j < p || p < i) return u;
        st[cnt] = st[u]; int x = cnt++; m = (i + j) / 2;
        if (i == j) { st[x] = v; return x; }
        int l = st[x].l = update(st[x].l, p, v, i, m);
        int r = st[x].r = update(st[x].r, p, v, m + 1, j);
        st[x] = node(st[l], st[r], l, r); return x;
    }

    PST(int N) : st(1e7), rt(1e5), n(N) {}
    void update(int t, int p, node v) { rt[rc++] = update(rt[t], p, v, 0, n - 1);
    }
    node query(int t, int a, int b) { return query(rt[t], a, b, 0, n - 1); }
};

```

7.6 PersistentSegmentTreeLazy

```

struct Node
{
    int v, l = 0, r = 0, lzv = 0;
    bool lz = false;
    Node() : v(0) {}
    Node(int x) : v(x) {}
    Node(const Node &a, const Node &b, int l, int r) : v(a.v + b.v), l(l), r(r) {}
    Node(int x, int i, int j, const Node &b) : l(b.l), r(b.r)
    { v = b.v + (j - i + 1) * x; }
};

// 0 - indexed / inclusive - inclusive
template <class node>
struct PSTL

```

```

{
    int cnt = 0, n, rc = 0;
    vector<node> st; vi rt;

    void push(int u, int v, int i, int j)
    {
        st[u] = node(v, i, j, st[u]);
        if (i == j) return;
        st[cnt] = st[st[u].l]; int l = cnt++;
        st[cnt] = st[st[u].r]; int r = cnt++;
        st[u].l = l, st[l].lz = 1, st[l].lzv += v;
        st[u].r = r, st[r].lz = 1, st[r].lzv += v;
    }
    node query(int u, int a, int b, int i, int j)
    {
        if (j < a || b < i) return node();
        if (st[u].lz) push(u, st[u].lzv, i, j);
        if (a <= i && j <= b) return st[u];
        int m = (i + j) / 2, l = st[u].l, r = st[u].r;
        return node(query(l, a, b, i, m), query(r, a, b, m + 1, j), l, r);
    }
    int update(int u, int a, int b, int v, int i, int j)
    {
        if (st[u].lz) push(u, st[u].lzv, i, j);
        if (j < a || b < i) return u;
        st[cnt] = st[u]; int x = cnt++, m = (i + j) / 2;
        if (a <= i && j <= b) { push(x, v, i, j); return x; }
        int l = st[x].l = update(st[x].l, a, b, v, i, m);
        int r = st[x].r = update(st[x].r, a, b, v, m + 1, j);
        st[x] = node(st[l], st[r], l, r); return x;
    }
    int build(vector<node> &arr, int i, int j)
    {
        int u = cnt++, m = (i + j) / 2;
        if (i == j) { st[u] = arr[i]; return u; }
        int l = st[u].l = build(arr, i, m);
        int r = st[u].r = build(arr, m + 1, j);
        st[u] = node(st[l], st[r], l, r); return u;
    }

    PSTL(vector<node> &arr) : st(1e7), rt(1e5)
    { n = arr.size(); rt[rc++] = build(arr, 0, n - 1); }
    void update(int t, int a, int b, int v)
    { rt[rc++] = update(rt[t], a, b, v, 0, n - 1); }
    node query(int t, int a, int b) { return query(rt[t], a, b, 0, n - 1); }
};

// Direct accumulate (No Push) Faster and shorter
struct Node
{
    int v = 0, l = 0, r = 0, lzv = 0;
    bool lz = false;
    Node() {}
    Node(int x) : v(x) {}
    Node(const Node &a, const Node &b, int l, int r) : v(a.v + b.v), l(l), r(r) {}

```

```

Node(int x, int i, int j, const Node &b)
{ *this = b; v += (j - i + 1) * x; } // *this = b needed in this variant
    (keeps lazy)
};

template <class node>
struct PSTL
{
    int cnt = 0, n, rc = 0;
    vector<node> st; vi rt;

    node query(int u, int a, int b, int i, int j, ll acc)
    {
        if (j < a || b < i) return node();
        if (st[u].lz) acc += st[u].lzv;
        int m = (i + j) / 2, l = st[u].l, r = st[u].r;
        if (a <= i && j <= b) return node(acc, i, j, st[u]);
        return node(query(l, a, b, i, m, acc), query(r, a, b, m + 1, j, acc), l,
            r);
    }
    int update(int u, int a, int b, int v, int i, int j)
    {
        if (j < a || b < i) return u;
        st[cnt] = st[u]; int x = cnt++, m = (i + j) / 2;
        if (a <= i && j <= b) { st[x].lz = 1, st[x].lzv += v; return x; }
        int l = st[x].l = update(st[x].l, a, b, v, i, m);
        int r = st[x].r = update(st[x].r, a, b, v, m + 1, j);
        st[x] = node(v, max(i, a), min(j, b), st[x]); return x;
    }
    int build(vector<node> &arr, int i, int j)
    {
        int u = cnt++, m = (i + j) / 2;
        if (i == j) { st[u] = arr[i]; return u; }
        int l = st[u].l = build(arr, i, m);
        int r = st[u].r = build(arr, m + 1, j);
        st[u] = node(st[l], st[r], l, r); return u;
    }

    PSTL(vector<node> &arr) : st(5e6), rt(2e5)
    { n = arr.size(); rt[rc++] = build(arr, 0, n - 1); }
    int update(int t, int a, int b, int v)
    { rt[rc] = update(rt[t], a, b, v, 0, n - 1); return rc++; }
    node query(int t, int a, int b) { return query(rt[t], a, b, 0, n - 1, 0); }
};

```

7.7 PolicyBased

```

#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

```

```
using namespace __gnu_pbds;
```

```

template <typename T, typename Comparator = less<T>>
using super_set = tree<T, null_type, Comparator, rb_tree_tag,
    tree_order_statistics_node_update>;

// order_of_key(T x)
// -> returns the number of elements strictly smaller than x

// find_by_order(size_t i)
// -> returns iterator to i-th largest element (counting from 0)

```

7.8 SegmentTree

```

struct Node
{
    int v;
    Node() { v = 0; } // neutro
    Node(int v) : v(v) {}
    Node(const Node &a, const Node &b) { v = a.v + b.v; }
};

// 0 - indexed / inclusive - exclusive
template <class node>
struct ST
{
    vector<node> t; int n;

    ST(vector<node> &arr, int N) : n(N), t(N * 2)
    {
        copy(arr.begin(), arr.end(), t.begin() + n);
        for (int i = n - 1; i > 0; --i) t[i] = node(t[i << 1], t[i << 1 | 1]);
    }
    void set(int p, const node &value)
    {
        for (t[p += n] = value; p >>= 1;)
            t[p] = node(t[p << 1], t[p << 1 | 1]);
    }
    node query(int l, int r)
    {
        node ans1, ansr;
        for (l += n, r += n; l < r; l >>= 1, r >>= 1)
        {
            if (l & 1) ans1 = node(ans1, t[l++]);
            if (r & 1) ansr = node(t[--r], ansr);
        }
        return node(ans1, ansr);
    }
};

```

7.9 SegmentTreeLazy

```

struct Node
{
    int v, lzv = 0;
    bool lz = false;
    Node() : v(0) {}
    Node(int x) : v(x) {}
    Node(const Node &a, const Node &b) : v(a.v + b.v) {}
    Node(int x, int i, int j, const Node &b)
    {
        v = b.v + (j - i + 1) * x;
    }
};

// 0 - indexed / inclusive - inclusive
template <class node>
struct STL
{
    vector<node> st; int n;

    void build(int u, int i, int j, vector<node> &arr)
    {
        if (i == j) { st[u] = arr[i]; return; }
        int m = (i + j) / 2, l = u * 2 + 1, r = u * 2 + 2;
        build(l, i, m, arr), build(r, m + 1, j, arr);
        st[u] = node(st[l], st[r]);
    }
    void push(int u, int i, int j, int x)
    {
        st[u] = node(x, i, j, st[u]);
        if (i == j) return;
        st[u * 2 + 1].lz = 1, st[u * 2 + 1].lzv += x;
        st[u * 2 + 2].lz = 1, st[u * 2 + 2].lzv += x;
    }
    node query(int a, int b, int u, int i, int j)
    {
        if (j < a || b < i) return node();
        int m = (i + j) / 2, l = u * 2 + 1, r = u * 2 + 2;
        if (st[u].lz) push(u, i, j, st[u].lzv);
        if (a <= i && j <= b) return st[u];
        return node(query(a, b, l, i, m), query(a, b, r, m + 1, j));
    }
    void update(int a, int b, int v, int u, int i, int j)
    {
        if (st[u].lz) push(u, i, j, st[u].lzv);
        if (j < a || b < i) return;
        int m = (i + j) / 2, l = u * 2 + 1, r = u * 2 + 2;
        if (a <= i && j <= b) { push(u, i, j, v); return; }
        update(a, b, v, l, i, m); update(a, b, v, r, m + 1, j);
        st[u] = node(st[l], st[r]);
    }
};

STL(vector<node> &v, int N) : n(N), st(N * 4 + 5) { build(0, 0, n - 1, v); }
node query(int a, int b) { return query(a, b, 0, 0, n - 1); }

```

```

void update(int a, int b, int v) { update(a, b, v, 0, 0, n - 1); }
};

```

7.10 SparseTable

```

template <class t>
class ST
{
    int n;
    vi memo, *arr;

public:
    ST(vi &_arr)
    {
        arr = &_arr; n = arr->size();
        int maxlog = 31 - __builtin_clz(n);
        memo.assign(n * (maxlog + 1), -1);
    }
    int dp(int i, int e)
    {
        int &ans = memo[e * n + i];
        if (ans != -1) return ans;
        if (e == 0) return ans = (*arr)[i];
        return ans = t::merge(dp(i, e - 1), dp(i + (1 << (e - 1)), e - 1));
    }

    int query_01(int l, int r)
    {
        int e = 31 - __builtin_clz(r - l + 1);
        return t::merge(dp(l, e), dp(r - (1 << e) + 1, e));
    }

    int query_0logn(int l, int r)
    {
        int ans = t::neutro;
        int d = r - l + 1;
        for (int e = 0; d; e++, d >>= 1) if (d & 1)
        {
            ans = t::merge(ans, dp(l, e));
            l += 1 << e;
        }
        return ans;
    }
};

struct OP
{
    static const int neutro = 0;
    static int merge(int x, int y) { return x + y; }
};

```

7.11 Treap

```

mt19937 gen(chrono::high_resolution_clock::now().time_since_epoch().count());
typedef pair<int, int> ii;

// 101 Treap //

struct Node
{
    int p, sz = 0, v, acc, l = -1, r = -1;
    Node() : v(0), acc(0) {}
    Node(int x): p(gen()), sz(1), v(x), acc(x) {}
    void recalc(const Node &a, const Node &b)
    {
        sz = a.sz + b.sz + 1;
        acc = v + a.acc + b.acc;
    }
};

template <class node>
struct Treap
{
    vector<node> t; int n, r = -1;

    node get(int u) { return u != -1 ? t[u] : node(); }
    void recalc(int u) { t[u].recalc(get(t[u].l), get(t[u].r)); }
    int merge(int l, int r)
    {
        if (min(l, r) == -1) return l != -1 ? l : r;
        int ans = (t[l].p < t[r].p) ? l : r;
        if (ans == l) t[l].r = merge(t[l].r, r), recalc(l);
        if (ans == r) t[r].l = merge(l, t[r].l), recalc(r);
        return ans;
    }
    ii split(int u, int id)
    {
        if (u == -1) return {-1, -1};
        int szl = get(t[u].l).sz;
        if (szl >= id)
        {
            ii ans = split(t[u].l, id);
            t[u].l = ans.ss; recalc(u);
            return {ans.ff, u};
        }
        ii ans = split(t[u].r, id - szl - 1);
        t[u].r = ans.ff; recalc(u);
        return {u, ans.ss};
    }

    Treap(vector<int> &v) : n(v.size())
    { for (int i = 0; i < n; i++) t.eb(v[i]), r = merge(r, i); }
};

```

```
// Complete Treap with Lazy propagation //
```

```

struct Node
{
    int p, sz = 0, v, acc, l = -1, r = -1, par = -1, lzv = 0;
    bool lz = false, f = false;
    Node() : v(0), acc(0) {}
    Node(int x): p(gen()), sz(1), v(x), acc(x) {}
    void recalc(const Node &a, const Node &b)
    {
        sz = a.sz + b.sz + 1;
        acc = v + a.acc + b.acc;
    }
    void upd_lazy(int x) { lz = 1, lzv += x; }
    void lazy() { v += lzv, acc += sz * lzv, lz = 0, lzv = 0; }
    void flip() { swap(l, r), f = 0; }
};

template <class node>
struct Treap
{
    vector<node> t; int n, r = -1;

    node get(int u) { return u != -1 ? t[u] : node(); }
    void recalc(int u)
    {
        int l = t[u].l, r = t[u].r;
        push(l); push(r); flip(l); flip(r);
        t[u].recalc(get(l), get(r));
    }
    void push(int u)
    {
        if (u == -1 || !t[u].lz) return;
        int l = t[u].l, r = t[u].r;
        if (l != -1) t[l].upd_lazy(t[u].lzv);
        if (r != -1) t[r].upd_lazy(t[u].lzv);
        t[u].lazy();
    }
    void flip(int u)
    {
        if (u == -1 || !t[u].f) return;
        int l = t[u].l, r = t[u].r;
        if (l != -1) t[l].f ^= 1;
        if (r != -1) t[r].f ^= 1;
        t[u].flip();
    }
    int merge(int l, int r)
    {
        if (min(l, r) == -1) return l != -1 ? l : r;
        push(l); push(r); flip(l); flip(r);
        int ans = (t[l].p < t[r].p) ? l : r;
        if (ans == l) t[l].r = merge(t[l].r, r), recalc(l);
        if (ans == r) t[r].l = merge(l, t[r].l), recalc(r);
        if (t[ans].l != -1) t[t[ans].l].par = ans; // only if parent needed
        if (t[ans].r != -1) t[t[ans].r].par = ans; // only if parent needed
    }
};

```

```

    return ans;
}
ii split(int u, int id)
{
    if (u == -1) return {-1, -1};
    push(u); flip(u);
    int szl = get(t[u].l).sz;
    if (szl >= id)
    {
        ii ans = split(t[u].l, id);
        if (ans.ss != -1) t[ans.ss].par = u; // only if parent needed
        if (ans.ff != -1) t[ans.ff].par = -1; // only if parent needed
        t[u].l = ans.ss; recalc(u);
        return {ans.ff, u};
    }
    ii ans = split(t[u].r, id - szl - 1);
    if (ans.ff != -1) t[ans.ff].par = u; // only if parent needed
    if (ans.ss != -1) t[ans.ss].par = -1; // only if parent needed
    t[u].r = ans.ff; recalc(u);
    return {u, ans.ss};
}
int update(int u, int l, int r, int v)
{
    ii a = split(u, l), b = split(a.ss, r - l + 1);
    t[b.ff].upd_lazy(v);
    return merge(a.ff, merge(b.ff, b.ss));
}
void print(int u)
{
    if (u == -1) return;
    push(u); flip(u);
    print(t[u].l);
    cout << t[u].v << ' ';
    print(t[u].r);
}

Treap(vector<int> &v) : n(v.size())
{ for (int i = 0; i < n; i++) t.eb(v[i]), r = merge(r, i); }
};

```

7.12 UnionFind

```

struct DSU
{
    vi p;
    DSU(int N) : p(N, -1) {}
    int get(int x) { return p[x] < 0 ? x : p[x] = get(p[x]); }
    bool sameSet(int a, int b) { return get(a) == get(b); }
    int size(int x) { return -p[get(x)]; }
    void unite(int x, int y)
    {

```

```

        if ((x = get(x)) == (y = get(y))) return;
        if (p[x] > p[y]) swap(x,y);
        p[x] += p[y], p[y] = x;
    }
};

```

7.13 WaveletTree

```

class WT
{
    typedef vi::iterator iter;
    vector<vi> r0;
    vi arrCopy;
    int n, s;

    void build(iter b, iter e, int l, int r, int u)
    {
        if (l == r) return;
        int m = (l + r) / 2;
        r0[u].reserve(e - b + 1); r0[u].pb(0);
        for (iter it = b; it != e; ++it)
            r0[u].pb(r0[u].back() + (*it <= m));
        iter p = stable_partition(b, e, [=](int i) { return i <= m; });
        build(b, p, l, m, u * 2); build(p, e, m + 1, r, u * 2 + 1);
    }

    int q, w;
    int range(int a, int b, int l, int r, int u)
    {
        if (r < q or w < l) return 0;
        if (q <= l && r <= w) return b - a;
        int m = (l + r) / 2, za = r0[u][a], zb = r0[u][b];
        return range(za, zb, l, m, u * 2) +
            range(a - za, b - zb, m + 1, r, u * 2 + 1);
    }

public:
    // arr[i] in [0,sigma)
    WT(vi arr, int sigma)
    {
        n = arr.size(); s = sigma; r0.resize(s * 2);
        arrCopy = arr;
        build(all(arr), 0, s - 1, 1);
    }

    // k in [1,n], [a,b) is 0-indexed, -1 if error
    int quantile(int k, int a, int b)
    {
        // extra conditions disabled
        if (/*a < 0 or b > n or*/ k < 1 or k > b - a) return -1;
        int l = 0, r = s - 1, u = 1, m, za, zb;

```



```

while (l != r)
{
    m = (l + r) / 2;
    za = r0[u][a], zb = r0[u][b], u *= 2;
    if (k <= zb - za) a = za, b = zb, r = m;
    else k -= zb - za, a -= za, b -= zb, l = m + 1, ++u;
}
return r;
}

// counts numbers in [x,y] in positions [a,b]
int range(int x, int y, int a, int b)
{
    if (y < x or b <= a) return 0;
    q = x, w = y;
    return range(a, b, 0, s - 1, 1);
}

// count occurrences of x in positions [0,k)
int rank(int x, int k)
{
    int l = 0, r = s - 1, u = 1, m, z;
    while (l != r)
    {
        m = (l + r) / 2;
        z = r0[u][k], u *= 2;
        if (x <= m) k = z, r = m;
        else k -= z, l = m + 1, ++u;
    }
    return k;
}

// x in [0,sigma)
void pb(int x)
{
    int l = 0, r = s - 1, u = 1, m, p;
    ++n;
    while (l != r)
    {
        m = (l + r) / 2;
        p = (x <= m);
        r0[u].pb(r0[u].back() + p);
        u *= 2;
    }
}

```

```

        if (p) r = m;
        else l = m + 1, ++u;
    }
}

// doesn't check if empty
void pop_back()
{
    int l = 0, r = s - 1, u = 1, m, p, k;
    --n;
    while (l != r)
    {
        m = (l + r) / 2;
        k = r0[u].size(), p = r0[u][k - 1] - r0[u][k - 2];
        r0[u].pop_back();
        u *= 2;
        if (p) r = m;
        else l = m + 1, ++u;
    }
}

// swap arr[i] with arr[i+1], i in [0,n-1]
void swap_adj(int i)
{
    int &x = arrCopy[i], &y = arrCopy[i + 1];
    int l = 0, r = s - 1, u = 1;
    while (l != r)
    {
        int m = (l + r) / 2, p = (x <= m), q = (y <= m);
        if (p != q)
        {
            r0[u][i + 1] ^= r0[u][i] ^ r0[u][i + 2];
            break;
        }
        u *= 2;
        if (p) r = m;
        else l = m + 1, ++u;
    }
    swap(x, y);
}
};

```