**/\*basic\*/**

**//basic.cpp**

**#include<bits/stdc++.h>**

**#include<ext/pb\_ds/assoc\_container.hpp>**

**#include<ext/pb\_ds/tree\_policy.hpp>**

**#include<ext/pb\_ds/tag\_and\_trait.hpp>**

**#define overload4(\_1, \_2, \_3, \_4, name, ...) name**

**#define rep1(i, n) for (ll i = 0; i < ll(n); ++i)**

**#define rep2(i, s, n) for (ll i = ll(s); i < ll(n); ++i)**

**#define rep3(i, s, n, d) for(ll i = ll(s); i < ll(n); i+=d)**

**#define rep(...) overload4(\_\_VA\_ARGS\_\_,rep3,rep2,rep1)(\_\_VA\_ARGS\_\_)**

**#define rrep1(i, n) for (ll i = ll(n)-1; i >= 0; i--)**

**#define rrep2(i, n, t) for (ll i = ll(n)-1; i >= (ll)t; i--)**

**#define rrep3(i, n, t, d) for (ll i = ll(n)-1; i >= (ll)t; i-=d)**

**#define rrep(...) overload4(\_\_VA\_ARGS\_\_,rrep3,rrep2,rrep1)(\_\_VA\_ARGS\_\_)**

**#define all(a) a.begin(),a.end()**

**#define rall(a) a.rbegin(),a.rend()**

**#define SUM(a) accumulate(all(a),0LL)**

**#define MIN(a) \*min\_element(all(a))**

**#define MAX(a) \*max\_element(all(a))**

**#define SORT(a) sort(all(a));**

**#define REV(a) reverse(all(a));**

**#define SZ(a) int(a.size())**

**#define popcount(x) \_\_builtin\_popcountll(x)**

**#define pf push\_front**

**#define pb push\_back**

**#define ef emplace\_front**

**#define eb emplace\_back**

**#define ppf pop\_front**

**#define ppb pop\_back**

**#ifdef \_\_LOCAL**

**#define debug(...) { cout << #\_\_VA\_ARGS\_\_; cout << ": "; print(\_\_VA\_ARGS\_\_); cout << flush; }**

**#else**

**#define debug(...) void(0);**

**#endif**

**#define INT(...) int \_\_VA\_ARGS\_\_;scan(\_\_VA\_ARGS\_\_)**

**#define LL(...) ll \_\_VA\_ARGS\_\_;scan(\_\_VA\_ARGS\_\_)**

**#define STR(...) string \_\_VA\_ARGS\_\_;scan(\_\_VA\_ARGS\_\_)**

**#define CHR(...) char \_\_VA\_ARGS\_\_;scan(\_\_VA\_ARGS\_\_)**

**#define DBL(...) double \_\_VA\_ARGS\_\_;scan(\_\_VA\_ARGS\_\_)**

**#define LD(...) ld \_\_VA\_ARGS\_\_;scan(\_\_VA\_ARGS\_\_)**

**using namespace std;**

**using namespace \_\_gnu\_pbds;**

**using ll = long long;**

**using ld = long double;**

**using P = pair<int, int>;**

**using LP = pair<ll, ll>;**

**using vi = vector<int>;**

**using vvi = vector<vi>;**

**using vvvi = vector<vvi>;**

**using vl = vector<ll>;**

**using vvl = vector<vl>;**

**using vvvl = vector<vvl>;**

**using vd = vector<double>;**

**using vvd = vector<vd>;**

**using vs = vector<string>;**

**using vc = vector<char>;**

**using vvc = vector<vc>;**

**using vb = vector<bool>;**

**using vvb = vector<vb>;**

**using vp = vector<P>;**

**using vvp = vector<vp>;**

**template<class T>**

**using PQ = priority\_queue <pair<T, int>, vector<pair < T, int>>, greater <pair<T, int>>>;**

**template<class S, class T>**

**istream &operator>>(istream &is, pair <S, T> &p) { return is >> p.first >> p.second; }**

**template<class S, class T>**

**ostream &operator<<(ostream &os, const pair <S, T> &p) { return os << '{' << p.first << ", " << p.second << '}'; }**

**template<class S, class T, class U>**

**istream &operator>>(istream &is, tuple <S, T, U> &t) { return is >> get<0>(t) >> get<1>(t) >> get<2>(t); }**

**template<class S, class T, class U>**

**ostream &operator<<(ostream &os, const tuple <S, T, U> &t) {**

**return os << '{' << get<0>(t) << ", " << get<1>(t) << ", " << get<2>(t) << '}';**

**}**

**template<class T>**

**istream &operator>>(istream &is, vector <T> &v) {**

**for (T &t: v) { is >> t; }**

**return is;**

**}**

**template<class T>**

**ostream &operator<<(ostream &os, const vector <T> &v) {**

**os << '[';**

**rep(i, v.size()) os << v[i] << (i == int(v.size() - 1) ? "" : ", ");**

**return os << ']';**

**}**

**template<class T>**

**ostream &operator<<(ostream &os, const deque <T> &v) {**

**os << '[';**

**rep(i, v.size()) os << v[i] << (i == int(v.size() - 1) ? "" : ", ");**

**return os << ']';**

**}**

**template<class T>**

**ostream &operator<<(ostream &os, const set <T> &st) {**

**os << '{';**

**auto it = st.begin();**

**while (it != st.end()) {**

**os << (it == st.begin() ? "" : ", ") << \*it;**

**it++;**

**}**

**return os << '}';**

**}**

**template<class T>**

**ostream &operator<<(ostream &os, const multiset <T> &st) {**

**os << '{';**

**auto it = st.begin();**

**while (it != st.end()) {**

**os << (it == st.begin() ? "" : ", ") << \*it;**

**it++;**

**}**

**return os << '}';**

**}**

**template<class T>**

**void vecout(const vector <T> &v, char div = '\n') {**

**rep(i, v.size()) cout << v[i] << (i == int(v.size() - 1) ? '\n' : div);**

**}**

**template<class T>**

**bool chmin(T &a, T b) {**

**if (a > b) {**

**a = b;**

**return true;**

**}**

**return false;**

**}**

**template<class T>**

**bool chmax(T &a, T b) {**

**if (a < b) {**

**a = b;**

**return true;**

**}**

**return false;**

**}**

**void scan() {}**

**template<class Head, class... Tail>**

**void scan(Head &head, Tail &... tail) {**

**cin >> head;**

**scan(tail...);**

**}**

**template<class T>**

**void print(const T &t) { cout << t << '\n'; }**

**template<class Head, class... Tail>**

**void print(const Head &head, const Tail &... tail) {**

**cout << head << ' ';**

**print(tail...);**

**}**

**template<class... T>**

**void fin(const T &... a) {**

**print(a...);**

**exit(0);**

**}**

**template<class T>**

**vector <T> &operator+=(vector <T> &v, T x) {**

**for (T &t: v) t += x;**

**return v;**

**}**

**template<class T>**

**vector <T> &operator-=(vector <T> &v, T x) {**

**for (T &t: v) t -= x;**

**return v;**

**}**

**template<class T>**

**vector <T> &operator\*=(vector <T> &v, T x) {**

**for (T &t: v) t \*= x;**

**return v;**

**}**

**template<class T>**

**vector <T> &operator/=(vector <T> &v, T x) {**

**for (T &t: v) t /= x;**

**return v;**

**}**

**struct Init\_io {**

**Init\_io() {**

**ios::sync\_with\_stdio(false);**

**cin.tie(nullptr);**

**cout.tie(nullptr);**

**cout << boolalpha << fixed << setprecision(15);**

**cerr << boolalpha << fixed << setprecision(15);**

**}**

**} init\_io;**

**const string yes[] = {"no", "yes"};**

**const string Yes[] = {"No", "Yes"};**

**const string YES[] = {"NO", "YES"};**

**const int inf = 1001001001;**

**const ll linf = 1001001001001001001;**

**void rearrange(const vi &) {}**

**template<class T, class... Tail>**

**void rearrange(const vi &ord, vector <T> &head, Tail &...tail) {**

**assert(ord.size() == head.size());**

**vector <T> ori = head;**

**rep(i, ord.size()) head[i] = ori[ord[i]];**

**rearrange(ord, tail...);**

**}**

**template<class T, class... Tail>**

**void sort\_by(vector <T> &head, Tail &... tail) {**

**vi ord(head.size());**

**iota(all(ord), 0);**

**sort(all(ord), [&](int i, int j) { return head[i] < head[j]; });**

**rearrange(ord, head, tail...);**

**}**

**template<class T, class S>**

**vector <T> cumsum(const vector <S> &v, bool shift\_one = true) {**

**int n = v.size();**

**vector <T> res;**

**if (shift\_one) {**

**res.resize(n + 1);**

**rep(i, n) res[i + 1] = res[i] + v[i];**

**} else {**

**res.resize(n);**

**if (n) {**

**res[0] = v[0];**

**rep(i, 1, n) res[i] = res[i - 1] + v[i];**

**}**

**}**

**return res;**

**}**

**vvi graph(int n, int m, bool directed = false, int origin = 1) {**

**vvi G(n);**

**rep(\_, m) {**

**INT(u, v);**

**u -= origin, v -= origin;**

**G[u].pb(v);**

**if (!directed) G[v].pb(u);**

**}**

**return G;**

**}**

**template<class T>**

**vector <vector<pair < int, T>>>**

**weighted\_graph(int n, int m, bool directed = false, int origin = 1) {**

**vector < vector < pair < int, T>>> G(n);**

**rep(\_, m) {**

**int u, v;**

**T w;**

**scan(u, v, w);**

**u -= origin, v -= origin;**

**G[u].eb(v, w);**

**if (!directed) G[v].eb(u, w);**

**}**

**return G;**

**}**

**int main() {**

**}**

**/\*data\_structure\*/**

**//dsu.cpp**

**class dsu {**

**int n;**

**vector<int> par, rank;**

**public:**

**dsu(int n) : n(n), par(n, -1), rank(n, 0) {}**

**int leader(int x) {**

**if (par[x] < 0) return x;**

**else return par[x] = leader(par[x]);**

**}**

**bool is\_leader(int x) { return leader(x) == x; }**

**bool same(int x, int y) { return leader(x) == leader(y); };**

**bool merge(int x, int y) {**

**x = leader(x);**

**y = leader(y);**

**if (x == y) return false;**

**if (rank[x] < rank[y]) swap(x, y);**

**if (rank[x] == rank[y]) rank[x]++;**

**par[x] += par[y];**

**par[y] = x;**

**return true;**

**}**

**int size(int x) { return -par[leader(x)]; };**

**vi leaders() {**

**vi res;**

**rep(i, n) if (leader(i) == i) res.pb(i);**

**return res;**

**}**

**bool connected() {**

**return leaders().size() == 1;**

**}**

**};**

**//lazy\_segtree.cpp**

**template<class M>**

**class lazy\_segtree {**

**using S = typename M::S;**

**using F = typename M::F;**

**int \_n, sz, log;**

**vector <S> d;**

**vector <F> lz;**

**void update(int k) { d[k] = M::op(d[2 \* k], d[2 \* k + 1]); }**

**void all\_apply(int k, F f) {**

**d[k] = M::mapping(f, d[k]);**

**if (k < sz) lz[k] = M::composition(f, lz[k]);**

**}**

**void push(int k) {**

**all\_apply(2 \* k, lz[k]);**

**all\_apply(2 \* k + 1, lz[k]);**

**lz[k] = M::id;**

**}**

**public:**

**constexpr lazy\_segtree() : lazy\_segtree(0) {}**

**constexpr lazy\_segtree(int \_n) : lazy\_segtree(vector<S>(\_n, M::e)) {}**

**constexpr lazy\_segtree(const vector <S> &init) : \_n(int(init.size())) {**

**log = 0;**

**while (1 << log < \_n) log++;**

**sz = 1 << log;**

**d.assign(2 \* sz, M::e);**

**lz.assign(sz, M::id);**

**rep(i, \_n)**

**d[sz + i] = init[i];**

**rrep(i, sz, 1)**

**update(i);**

**}**

**void set(int p, S x) {**

**assert(0 <= p and p < \_n);**

**p += sz;**

**rrep(i, log + 1, 1)**

**push(p >> i);**

**d[p] = x;**

**rep(i, 1, log + 1)**

**update(p >> i);**

**}**

**template<class F>**

**void apply(int p, const F &f) {**

**assert(0 <= p and p < \_n);**

**p += sz;**

**rrep(i, log + 1, 1)**

**push(p >> i);**

**d[p] = f(d[p]);**

**rep(i, 1, log + 1)**

**update(p >> i);**

**}**

**S get(int p) {**

**assert(0 <= p and p < \_n);**

**p += sz;**

**rrep(i, log + 1, 1)**

**push(p >> i);**

**return d[p];**

**}**

**S prod(int l, int r) {**

**assert(0 <= l and l <= r and r <= \_n);**

**l += sz, r += sz;**

**rrep(i, log + 1, 1)**

**{**

**if ((l >> i) << i != l) push(l >> i);**

**if ((r >> i) << i != r) push(r >> i);**

**}**

**S sl = M::e, sr = M::e;**

**while (l < r) {**

**if (l & 1) sl = M::op(sl, d[l++]);**

**if (r & 1) sr = M::op(d[--r], sr);**

**l >>= 1, r >>= 1;**

**}**

**return M::op(sl, sr);**

**}**

**S all\_prod() {**

**return d[1];**

**}**

**void apply(int l, int r, F f) {**

**assert(0 <= l and l <= r and r <= \_n);**

**l += sz, r += sz;**

**rrep(i, log + 1, 1)**

**{**

**if ((l >> i) << i != l) push(l >> i);**

**if ((r >> i) << i != r) push(r >> i);**

**}**

**{**

**int l2 = l, r2 = r;**

**while (l < r) {**

**if (l & 1) all\_apply(l++, f);**

**if (r & 1) all\_apply(--r, f);**

**l >>= 1, r >>= 1;**

**}**

**l = l2, r = r2;**

**}**

**rep(i, 1, log + 1)**

**{**

**if ((l >> i) << i != l) update(l >> i);**

**if ((r >> i) << i != r) update(r >> i);**

**}**

**}**

**template<class F>**

**int max\_right(int l, F f) {**

**assert(0 <= l && l <= \_n);**

**assert(f(M::e));**

**if (l == \_n) return \_n;**

**l += sz;**

**rrep(i, log + 1, 1)**

**push(l >> i);**

**S now = M::e;**

**do {**

**while (~l & 1) l >>= 1;**

**if (!f(M::op(now, d[l]))) {**

**while (l < sz) {**

**push(l);**

**l \*= 2;**

**if (f(M::op(now, d[l]))) {**

**now = M::op(now, d[l]);**

**++l;**

**}**

**}**

**return l - sz;**

**}**

**now = M::op(now, d[l]);**

**++l;**

**} while ((l & -l) != l);**

**return \_n;**

**}**

**template<class F>**

**int min\_left(int r, F f) {**

**assert(0 <= r && r <= \_n);**

**assert(f(M::e));**

**if (r == 0) return 0;**

**r += sz;**

**for (int i = log; i >= 1; i--) push((r - 1) >> i);**

**S now = M::e;**

**do {**

**r--;**

**while (r > 1 && (r & 1)) r >>= 1;**

**if (!f(M::op(d[r], now))) {**

**while (r < sz) {**

**push(r);**

**r = 2 \* r + 1;**

**if (f(M::op(d[r], now))) {**

**now = M::op(d[r], now);**

**--r;**

**}**

**}**

**return r + 1 - sz;**

**}**

**now = M::op(d[r], now);**

**} while ((r & -r) != r);**

**return 0;**

**}**

**};**

**class M {**

**public:**

**using S = ;**

**static constexpr S**

**e =;**

**static constexpr S**

**op(const S &l, const S &r) {**

**return;**

**}**

**using F = ;**

**static constexpr F**

**id =;**

**static constexpr F**

**composition(const F &g, const F &f) {**

**return;**

**}**

**static constexpr S**

**mapping(const F &f, const S &x) {**

**return;**

**}**

**};**

**//segrtree.cpp**

**template<class M>**

**class segtree {**

**using S = typename M::S;**

**int \_n, sz;**

**vector <S> d;**

**public:**

**constexpr segtree() : segtree(0) {}**

**constexpr segtree(int n) : segtree(vector<S>(n, M::e)) {}**

**constexpr segtree(const vector <S> &init) : \_n(int(init.size())) {**

**sz = 1;**

**while (sz < \_n) sz \*= 2;**

**d.assign(sz \* 2, M::e);**

**rep(i, \_n)**

**d[sz + i] = init[i];**

**rrep(i, sz, 1)**

**d[i] = M::op(d[2 \* i], d[2 \* i + 1]);**

**}**

**void set(int p, S x) {**

**assert(0 <= p and p < \_n);**

**p += sz;**

**d[p] = x;**

**while (p > 1) {**

**p >>= 1;**

**d[p] = M::op(d[2 \* p], d[2 \* p + 1]);**

**}**

**}**

**template<class F>**

**void apply(int p, const F &f) {**

**assert(0 <= p and p < \_n);**

**p += sz;**

**d[p] = f(d[p]);**

**while (p > 1) {**

**p >>= 1;**

**d[p] = M::op(d[2 \* p], d[2 \* p + 1]);**

**}**

**}**

**S get(int p) {**

**assert(0 <= p and p < \_n);**

**return d[sz + p];**

**}**

**S prod(int l, int r) {**

**assert(0 <= l and l <= r and r <= \_n);**

**l += sz, r += sz;**

**S prod\_l = M::e;**

**S prod\_r = M::e;**

**while (l < r) {**

**if (l & 1) prod\_l = M::op(prod\_l, d[l++]);**

**if (r & 1) prod\_r = M::op(d[--r], prod\_r);**

**l >>= 1, r >>= 1;**

**}**

**return M::op(prod\_l, prod\_r);**

**}**

**S all\_prod() {**

**return d[1];**

**}**

**template<class F>**

**int max\_right(int l, F f) const {**

**assert(0 <= l && l <= \_n);**

**assert(f(M::e));**

**if (l == \_n) return \_n;**

**l += sz;**

**S now = M::e;**

**do {**

**while (~l & 1) l >>= 1;**

**if (!f(M::op(now, d[l]))) {**

**while (l < sz) {**

**l \*= 2;**

**if (f(M::op(now, d[l]))) {**

**now = M::op(now, d[l]);**

**++l;**

**}**

**}**

**return l - sz;**

**}**

**now = M::op(now, d[l]);**

**l++;**

**} while ((l & -l) != l);**

**return \_n;**

**}**

**template<class F>**

**int min\_left(int r, F f) const {**

**assert(0 <= r && r <= \_n);**

**assert(f(M::e));**

**if (r == 0) return 0;**

**r += sz;**

**S now = M::e;**

**do {**

**--r;**

**while (r > 1 and (r & 1)) r >>= 1;**

**if (!f(M::op(d[r], now))) {**

**while (r < sz) {**

**r = 2 \* r + 1;**

**if (f(M::op(d[r], now))) {**

**now = M::op(d[r], now);**

**--r;**

**}**

**}**

**return r + 1 - sz;**

**}**

**now = M::op(d[r], now);**

**} while ((r & -r) != r);**

**return 0;**

**}**

**};**

**class M {**

**public:**

**using S = ;**

**static constexpr S**

**e =;**

**static constexpr S**

**op(const S &l, const S &r) {**

**return;**

**}**

**};**

**//weighted\_unionfind.cpp**

**template<typename T>**

**class weighted\_unionfind {**

**int n;**

**vi par, rank;**

**vector <T> diff\_weight;**

**public:**

**weighted\_unionfind(int n) : n(n), par(n, -1), rank(n, 0), diff\_weight(n, 0) {}**

**int root(int x) {**

**if (par[x] < 0) return x;**

**else {**

**int r = root(par[x]);**

**diff\_weight[x] += diff\_weight[par[x]];**

**return par[x] = r;**

**}**

**}**

**T weight(int x) {**

**root(x);**

**return diff\_weight[x];**

**}**

**bool is\_root(int x) { return root(x) == x; }**

**bool same(int x, int y) { return root(x) == root(y); }**

**bool merge(int x, int y, T w) {**

**w += weight(x);**

**w -= weight(y);**

**x = root(x);**

**y = root(y);**

**if (x == y) return false;**

**if (rank[x] < rank[y]) swap(x, y), w = -w;**

**if (rank[x] == rank[y]) rank[x]++;**

**par[x] += par[y];**

**par[y] = x;**

**diff\_weight[y] = w;**

**return true;**

**}**

**T diff(int x, int y) {**

**assert(same(x, y));**

**return weight(y) - weight(x);**

**}**

**vi roots() {**

**vi res;**

**rep(i, n)**

**if (root(i) == i) res.pb(i);**

**return res;**

**}**

**bool connected() {**

**return roots().size() == 1;**

**}**

**};**

**/\*geometry\*/**

**//geometry.cpp**

**const double eps = 1e-9;**

**const double PI = acos(-1);**

**int sgn(double a) { return a < -eps ? -1 : (a > eps ? 1 : 0); }**

**double to\_rad(double deg) { return deg \* PI / 180; }**

**double to\_deg(double rad) { return rad \* 180 / PI; }**

**struct point {**

**double x, y;**

**point(double x = 0, double y = 0) : x(x), y(y) {}**

**point operator+(const point &p) const { return {x + p.x, y + p.y}; }**

**point operator-(const point &p) const { return {x - p.x, y - p.y}; }**

**point operator\*(double a) const { return {x \* a, y \* a}; }**

**point operator\*(const point &p) const { return point(x \* p.x - y \* p.y, x \* p.y + y \* p.x); }**

**point operator/(double a) const { return {x / a, y / a}; }**

**point operator-() const { return \*this \* (-1); }**

**bool operator==(const point &p) const { return !sgn(x - p.x) && !sgn(y - p.y); }**

**bool operator!=(const point &p) const { return !(\*this == p); }**

**bool operator<(const point &p) const { return sgn(x - p.x) ? x < p.x : y < p.y; }**

**bool operator>(const point &p) const { return sgn(x - p.x) ? x > p.x : y > p.y; }**

**double norm() const { return x \* x + y \* y; }**

**double abs() const { return sqrt(norm()); }**

**point rot(double rad) const { return point(cos(rad) \* x - sin(rad) \* y, sin(rad) \* x + cos(rad) \* y); }**

**point rot90() const { return point(-y, x); }**

**double arg() const {**

**double res = atan2(y, x);**

**if (sgn(res) < 0) res += 2 \* PI;**

**return res;**

**}**

**};**

**istream &operator>>(istream &is, point &p) { return is >> p.x >> p.y; }**

**ostream &operator<<(ostream &os, const point &p) { return os << '(' << p.x << "," << p.y << ')'; }**

**double dist(const point &a, const point &b) { return (a - b).abs(); }**

**double dot(const point &a, const point &b) { return a.x \* b.x + a.y \* b.y; }**

**double cross(const point &a, const point &b) { return a.x \* b.y - a.y \* b.x; }**

**point mid(const point &a, const point &b) { return (a + b) / 2; }**

**int ccw(const point &a, const point &b, const point &c) {**

**// 1 -> c is upper than line(a,b)**

**// -1 -> c is lower than line(a,b)**

**// 2 -> in order [a,b,c]**

**// -2 -> in order [c,a,b]**

**// 0 -> in order [a,c,b]**

**point nb = b - a, nc = c - a;**

**if (sgn(cross(nb, nc))) return sgn(cross(nb, nc));**

**if (sgn(dot(nb, nc)) < 0) return -2;**

**if (sgn(nc.abs() - nb.abs()) > 0) return 2;**

**return 0;**

**}**

**struct line {**

**point a, b;**

**line(point a = point(), point b = point()) : a(a), b(b) {}**

**bool online(const point &p) const { return abs(ccw(a, b, p)) != 1; }**

**};**

**ostream &operator<<(ostream &os, const line &l) { return os << '{' << l.a << ',' << l.b << '}'; }**

**struct segment {**

**point a, b;**

**segment(point a = point(), point b = point()) : a(a), b(b) {}**

**bool online(const point &p) const { return !ccw(a, b, p); }**

**line vertical\_bisector() const { return line(mid(a, b), mid(a, b) + (b - a).rot90()); }**

**};**

**ostream &operator<<(ostream &os, const segment &l) { return os << '{' << l.a << ',' << l.b << '}'; }**

**bool vertical(const line &l, const line &m) { return !sgn(dot(l.a - l.b, m.a - m.b)); }**

**bool vertical(const segment &l, const segment &m) { return !sgn(dot(l.a - l.b, m.a - m.b)); }**

**bool parallel(const line &l, const line &m) { return !sgn(cross(l.a - l.b, m.a - m.b)); }**

**bool parallel(const segment &l, const segment &m) { return !sgn(cross(l.a - l.b, m.a - m.b)); }**

**bool operator==(const line &l, const line &m) { return parallel(l, m) && l.online(m.a); }**

**bool operator!=(const line &l, const line &m) { return !(l == m); }**

**bool operator==(const segment &l, const segment &m) { return l.a == m.a && l.b == m.b || l.a == m.b && l.b == m.a; }**

**bool operator!=(const segment &l, const segment &m) { return !(l == m); }**

**// intersect at one point**

**bool intersect(const line &l, const line &m) { return !parallel(l, m); }**

**bool intersect(const line &l, const segment &m) {**

**return sgn(cross(l.b - l.a, m.a - l.a) \* cross(l.b - l.a, m.b - l.a)) <= 0;**

**}**

**bool intersect(const segment &l, const segment &m) {**

**return ccw(l.a, l.b, m.a) \* ccw(l.a, l.b, m.b) <= 0 &&**

**ccw(m.a, m.b, l.a) \* ccw(m.a, m.b, l.b) <= 0;**

**}**

**point intersection(const line &l, const line &m) {**

**assert(intersect(l, m));**

**return l.a + (l.b - l.a) \* cross(m.b - m.a, m.a - l.a) / cross(m.b - m.a, l.b - l.a);**

**}**

**point intersection(const line &l, const segment &m) {**

**assert(intersect(l, m));**

**return l.a + (l.b - l.a) \* cross(m.b - m.a, m.a - l.a) / cross(m.b - m.a, l.b - l.a);**

**}**

**point intersection(const segment &l, const segment &m) {**

**assert(intersect(l, m));**

**return l.a + (l.b - l.a) \* cross(m.b - m.a, m.a - l.a) / cross(m.b - m.a, l.b - l.a);**

**}**

**double dist(const line &l, const point &p) { return abs(cross(l.b - l.a, p - l.a)) / (l.b - l.a).abs(); }**

**double dist(const segment &l, const point &p) {**

**if (sgn(dot(l.b - l.a, p - l.a)) < 0) return dist(p, l.a);**

**if (sgn(dot(l.a - l.b, p - l.b)) < 0) return dist(p, l.b);**

**return dist(line(l.a, l.b), p);**

**}**

**double dist(const line &l, const line &m) {**

**if (parallel(l, m)) return dist(l, m.a);**

**return 0;**

**}**

**double dist(const line &l, const segment &m) {**

**if (intersect(l, m)) return 0;**

**return min(dist(l, m.a), dist(l, m.b));**

**}**

**double dist(const segment &l, const segment &m) {**

**if (intersect(l, m)) return 0;**

**return min({dist(l, m.a), dist(l, m.b), dist(m, l.a), dist(m, l.b)});**

**}**

**point projection(const line &l, const point &p) {**

**double d = dot(p - l.a, l.b - l.a) / (l.b - l.a).norm();**

**return l.a + (l.b - l.a) \* d;**

**}**

**point circumcenter(const point &a, const point &b, const point &c) {**

**return intersection(segment(a, b).vertical\_bisector(), segment(b, c).vertical\_bisector());**

**}**

**struct circle {**

**point o;**

**double r;**

**circle(point o = point(), double r = 0) : o(o), r(r) {}**

**bool inside(const point &p) const { return sgn(r - dist(o, p)) >= 0; }**

**double area() const { return r \* r \* PI; }**

**};**

**ostream &operator<<(ostream &os, const circle &c) { return os << '{' << c.o << ',' << c.r << '}'; }**

**bool intersect(const circle &c, const line &l) { return sgn(dist(l, c.o) - c.r) <= 0; }**

**bool intersect(const circle &c, const segment &l) {**

**if (sgn(dist(l, c.o) - c.r) > 0) return false;**

**return sgn(max((c.o - l.a).abs(), (c.o - l.b).abs()) - c.r) >= 0;**

**}**

**vector <point> intersection(const circle &c, const line &l) {**

**point p = projection(l, c.o);**

**if (!intersect(c, l)) return {};**

**if (sgn(dist(l, c.o) - c.r) == 0) return {p};**

**point e = (l.b - l.a) / (l.b - l.a).abs();**

**double d = sqrt(c.r \* c.r - (p - c.o).norm());**

**return {p - e \* d, p + e \* d};**

**}**

**vector <point> intersection(const circle &c, const segment &l) {**

**auto v = intersection(c, line(l.a, l.b));**

**vector <point> ret;**

**for (point p : v) if (l.online(p)) ret.pb(p);**

**return ret;**

**}**

**vector <point> intersection(const circle &a, const circle &b) {**

**double d = dist(a.o, b.o);**

**if (!sgn(a.r + b.r - d)) return {a.o + (b.o - a.o) \* a.r / d};**

**if (!sgn(a.r - b.r - d)) return {a.o + (b.o - a.o) \* a.r / d};**

**if (!sgn(b.r - a.r - d)) return {b.o + (a.o - b.o) \* b.r / d};**

**if (sgn(abs(a.r - b.r) - d) > 0 || sgn(a.r + b.r - d) < 0) return {};**

**double x = (a.r \* a.r + d \* d - b.r \* b.r) / (2 \* d);**

**double y = sqrt(a.r \* a.r - x \* x);**

**point p = (b.o - a.o).rot90() \* y / d;**

**point to\_mid = a.o + (b.o - a.o) \* x / d;**

**return {to\_mid - p, to\_mid + p};**

**}**

**vector <circle> circle\_with\_two\_points\_and\_radius(const point &a, const point &b, const double &r) {**

**if (sgn(dist(a, b) - 2 \* r) > 0) return {};**

**circle A(a, r), B(b, r);**

**auto v = intersection(A, B);**

**vector <circle> ret;**

**for (point p : v) ret.eb(p, r);**

**return ret;**

**};**

**vector <point> tangent\_point(const circle &c, const point &p) {**

**int s = sgn(dist(c.o, p) - c.r);**

**if (s < 0) return {};**

**if (s == 0) return {p};**

**double d = (p - c.o).norm() - c.r \* c.r;**

**return intersection(c, circle(p, sqrt(d)));**

**}**

**vector <line> tangent\_line(const circle &c, const point &p) {**

**vector <point> v = tangent\_point(c, p);**

**if (v.empty()) return {};**

**if (v.size() == 1) return {line(p, p + (c.o - p).rot90())};**

**vector <line> res;**

**for (auto tp : v) res.eb(p, tp);**

**return res;**

**}**

**vector <line> tangent\_line(const circle &a, const circle &b) {**

**if (sgn(a.r - b.r) < 0) return tangent\_line(b, a);**

**double ar = a.r, br = b.r, d = dist(a.o, b.o);**

**if (sgn(d - (ar - br)) < 0) return {};**

**else if (sgn(d - (ar - br)) == 0) {**

**point p = (a.o \* (-br) + b.o \* ar) / (ar - br);**

**return {line(p, p + (a.o - p).rot90())};**

**} else {**

**vector <line> res;**

**{**

**double theta = acos((ar - br) / d);**

**{**

**point p = a.o + ((b.o - a.o) / d \* ar).rot(-theta);**

**res.eb(p, p + (a.o - p).rot90());**

**}**

**{**

**point p = a.o + ((b.o - a.o) / d \* ar).rot(theta);**

**res.eb(p, p + (a.o - p).rot90());**

**}**

**}**

**if (sgn(d - (ar + br)) >= 0) {**

**point p = (a.o \* br + b.o \* ar) / (ar + br);**

**vector <line> lines = tangent\_line(a, p);**

**for (line l : lines) res.pb(l);**

**}**

**return res;**

**}**

**}**

**vector <point> convex\_hull(vector <point> v) {**

**sort(all(v));**

**int n = v.size(), k = 0;**

**vector <point> res(2 \* n);**

**for (int i = 0; i < n; res[k++] = v[i++])**

**while (k > 1 && ccw(res[k - 2], res[k - 1], v[i]) <= 0) k--;**

**for (int i = n - 2, t = k; i >= 0; res[k++] = v[i--])**

**while (k > t && ccw(res[k - 2], res[k - 1], v[i]) <= 0) k--;**

**res.resize(k - 1);**

**return res;**

**}**

**/\*以下，not\_verified\*/**

**vector <point> convex\_hull\_upper(){**

**sort(all(v));**

**int n = v.size(), k = 0;**

**vector <point> res(2 \* n);**

**for (int i = 0; i < n; res[k++] = v[i++])**

**while (k > 1 && ccw(res[k - 2], res[k - 1], v[i]) <= 0) k--;**

**res.resize(k - 1);**

**return res;**

**}**

**vector <point> convex\_hull\_lower(){**

**sort(all(v));**

**int n = v.size(), k = 0;**

**vector <point> res(2 \* n);**

**for (int i = n - 2, t = k; i >= 0; res[k++] = v[i--])**

**while (k > t && ccw(res[k - 2], res[k - 1], v[i]) <= 0) k--;**

**res.resize(k - 1);**

**return res;**

**}**

**//polar\_sort.cpp**

**// -pi to pi**

**// no (0, 0)**

**bool arg\_cmp(const LP &a, const LP &b) {**

**int ua = a.second > 0 or (a.second == 0 and a.first >= 0);**

**int ub = b.second > 0 or (b.second == 0 and b.first >= 0);**

**if (ua == ub) {**

**ll tmp = a.first \* b.second - a.second \* b.first;**

**if (tmp != 0) return tmp > 0;**

**else return a.first ? abs(a.first) < abs(b.first) : abs(a.second) < abs(b.second);**

**} else return ua < ub;**

**}**

**/\*graph\*/**

**struct edge {**

**int a,b; // from a to b**

**ll len;**

**edge(int a,int b,ll l):a(a),b(b),len(l) {}**

**};**

**// return value is {} if there is a negative cycle.**

**/\* detect only if a certain node(goal) is reachable**

**from a negative cycle (if goal != -1) \*/**

**// detect any negative cycles (if goal == -1)**

**vector<ll> bellman\_ford(int n,const vector<edge>& v,int start,int goal=-1) {**

**vector<ll> dist(n,linf);**

**dist[start] = 0;**

**if(goal == -1) {**

**rep(i,n) {**

**for(auto e : v) {**

**if(dist[e.a] < linf && chmin(dist[e.b],dist[e.a]+e.len)) {**

**if(i == n-1) return {};**

**}**

**}**

**}**

**} else {**

**rep(i,n\*2) {**

**for(auto e : v) {**

**if(dist[e.a] < linf && chmin(dist[e.b],dist[e.a]+e.len)) {**

**if(i >= n-1 && e.b == goal) return {};**

**else if(i >= n-1) dist[e.b] = -linf;**

**}**

**}**

**}**

**}**

**return dist;**

**}**

**//bfs.cpp**

**vi BFS(const vvi &G, int start) {**

**int n = G.size();**

**vi dist(n, inf);**

**dist[start] = 0;**

**queue<int> q;**

**q.push(start);**

**while (!q.empty()) {**

**int u = q.front();**

**q.pop();**

**for (int v : G[u]) {**

**if (chmin(dist[v], dist[u] + 1)) q.push(v);**

**}**

**}**

**return dist;**

**}**

**//cycle.cpp**

**// find cycle of functional graph**

**vb find\_cycle(const vi &to) {**

**int n = to.size();**

**vb res(n);**

**vi seen(n);**

**rep(i, n)**

**{**

**if (seen[i]) continue;**

**vi ls;**

**int now = i;**

**while (true) {**

**ls.pb(now);**

**seen[now] = 1;**

**now = to[now];**

**if (seen[now] == 2) break;**

**if (seen[now] == 1) {**

**rrep(j, SZ(ls))**

**{**

**if (ls[j] == now) {**

**rep(k, j, SZ(ls))**

**res[ls[k]] = true;**

**break;**

**}**

**}**

**break;**

**}**

**}**

**for (int j: ls) seen[j] = 2;**

**}**

**return res;**

**}**

**//diameter.cpp**

**// G is the tree**

**// can calc diameter by "dfs(dfs(0).second).first"**

**P dfs(const vvi &G, int u, int p = -1) {**

**int res = 0, ind = u;**

**for (int v : G[u]) {**

**if (v == p) continue;**

**auto d = dfs(G, v, u);**

**if (chmax(res, d.first + 1)) ind = d.second;**

**}**

**return {res, ind};**

**}**

**//dijkstra.cpp**

**// (to, len)**

**using vve = vector <vector<pair < int, ll>>>;**

**vl dijkstra(const vve &G, int start) {**

**int sz = G.size();**

**vl dist(sz, linf);**

**PQ <ll> q;**

**q.emplace(0, start);**

**dist[start] = 0;**

**while (!q.empty()) {**

**auto[d, u] = q.top();**

**q.pop();**

**if (dist[u] < d) continue;**

**for (auto[v, len]: G[u]) {**

**if (chmin(dist[v], d + len)) {**

**q.emplace(d + len, v);**

**}**

**}**

**}**

**return dist;**

**}**

**//dinic.cpp**

**template<typename T>**

**class dinic {**

**struct edge {**

**int to;**

**T cap;**

**int rev;**

**edge(int to, T cap, int rev) : to(to), cap(cap), rev(rev) {}**

**};**

**int n;**

**vector <vector<edge>> G;**

**vi level, iter;**

**void bfs(int s) {**

**level.assign(n, -1);**

**queue<int> q;**

**level[s] = 0;**

**q.push(s);**

**while (!q.empty()) {**

**int v = q.front();**

**q.pop();**

**for (auto &e : G[v]) {**

**if (e.cap > 0 && level[e.to] < 0) {**

**level[e.to] = level[v] + 1;**

**q.push(e.to);**

**}**

**}**

**}**

**}**

**T dfs(int v, int t, T f) {**

**if (v == t) return f;**

**for (int &i = iter[v]; i < (int) G[v].size(); i++) {**

**auto &e = G[v][i];**

**if (e.cap > 0 && level[v] < level[e.to]) {**

**T d = dfs(e.to, t, min(f, e.cap));**

**if (d > 0) {**

**e.cap -= d;**

**G[e.to][e.rev].cap += d;**

**return d;**

**}**

**}**

**}**

**return 0;**

**}**

**public:**

**explicit dinic(int n) : n(n), G(n), level(n), iter(n) {}**

**void add\_edge(int from, int to, T cap) {**

**G[from].eb(to, cap, G[to].size());**

**G[to].eb(from, 0, G[from].size() - 1);**

**}**

**// O(|E||V|^2)**

**T max\_flow(int s, int t) {**

**T fl = 0;**

**while (true) {**

**bfs(s);**

**if (level[t] < 0) return fl;**

**iter.assign(n, 0);**

**T f;**

**while ((f = dfs(s, t, numeric\_limits<T>::max())) > 0) fl += f;**

**}**

**}**

**};**

**//dxdy.cpp**

**const int dx[8] = {0, 1, 0, -1, 1, 1, -1, -1};**

**const int dy[8] = {1, 0, -1, 0, 1, -1, 1, -1};**

**const int dx[4] = {0, 1, 0, -1};**

**const int dy[4] = {1, 0, -1, 0};**

**//ford\_fulkerson.cpp**

**template<typename T>**

**class ford\_fulkerson {**

**struct edge {**

**int to;**

**T cap;**

**int rev;**

**edge(int to, T cap, int rev) : to(to), cap(cap), rev(rev) {}**

**};**

**int n;**

**vector<vector<edge>> G;**

**vb used;**

**T dfs(int v, int t, T f) {**

**if (v == t) return f;**

**used[v] = true;**

**rep(i, G[v].size()) {**

**edge &e = G[v][i];**

**if (e.cap > 0 && !used[e.to]) {**

**T d = dfs(e.to, t, min(f, e.cap));**

**if (d > 0) {**

**e.cap -= d;**

**G[e.to][e.rev].cap += d;**

**return d;**

**}**

**}**

**}**

**return 0;**

**}**

**public:**

**explicit ford\_fulkerson(int n) : n(n), G(n), used(n) {}**

**void add\_edge(int from, int to, T cap) {**

**G[from].eb(to, cap, G[to].size());**

**G[to].eb(from, 0, G[from].size() - 1);**

**}**

**T max\_flow(int s, int t) {**

**T fl = 0;**

**while (true) {**

**used.assign(n, false);**

**T f = dfs(s, t, inf);**

**if (f == 0) return fl;**

**fl += f;**

**}**

**}**

**};**

**//HLD.cpp**

**template<typename T>**

**class ford\_fulkerson {**

**struct edge {**

**int to;**

**T cap;**

**int rev;**

**edge(int to, T cap, int rev) : to(to), cap(cap), rev(rev) {}**

**};**

**int n;**

**vector<vector<edge>> G;**

**vb used;**

**T dfs(int v, int t, T f) {**

**if (v == t) return f;**

**used[v] = true;**

**rep(i, G[v].size()) {**

**edge &e = G[v][i];**

**if (e.cap > 0 && !used[e.to]) {**

**T d = dfs(e.to, t, min(f, e.cap));**

**if (d > 0) {**

**e.cap -= d;**

**G[e.to][e.rev].cap += d;**

**return d;**

**}**

**}**

**}**

**return 0;**

**}**

**public:**

**explicit ford\_fulkerson(int n) : n(n), G(n), used(n) {}**

**void add\_edge(int from, int to, T cap) {**

**G[from].eb(to, cap, G[to].size());**

**G[to].eb(from, 0, G[from].size() - 1);**

**}**

**T max\_flow(int s, int t) {**

**T fl = 0;**

**while (true) {**

**used.assign(n, false);**

**T f = dfs(s, t, inf);**

**if (f == 0) return fl;**

**fl += f;**

**}**

**}**

**};**

**//kruskal.cpp**

**struct edge {**

**int x, y;**

**ll cost;**

**edge(int x, int y, ll cost) : x(x), y(y), cost(cost) {}**

**};**

**ll kruskal(int n, vector <edge> &v) {**

**unionfind uf(n);**

**sort(all(v), [](const edge &a, const edge &b) { return a.cost < b.cost; });**

**ll ret = 0;**

**for (auto e : v) {**

**if (uf.same(e.x, e.y)) continue;**

**uf.merge(e.x, e.y);**

**ret += e.cost;**

**}**

**return ret;**

**}**

**//LCA.cpp**

**class LCA {**

**int n;**

**vvi G;**

**vi dep;**

**vvi par;**

**void dfs(int u, int p, int d) {**

**par[0][u] = p;**

**dep[u] = d;**

**for (int v: G[u]) if (v != p) dfs(v, u, d + 1);**

**}**

**void init() {**

**n = G.size();**

**dep.assign(n, -1);**

**par.resize(30);**

**rep(i, 30)**

**par[i].resize(n);**

**rep(i, n)**

**if (dep[i] == -1) dfs(i, -1, 0);**

**rep(k, 29)**

**rep(i, n)**

**{**

**if (par[k][i] < 0) par[k + 1][i] = -1;**

**else par[k + 1][i] = par[k][par[k][i]];**

**}**

**}**

**public:**

**LCA(const vvi &G) : G(G) { init(); }**

**int operator()(int u, int v) {**

**if (dep[u] > dep[v]) swap(u, v);**

**rep(k, 30)**

**if ((dep[v] - dep[u]) >> k & 1) v = par[k][v];**

**if (u == v) return u;**

**rrep(k, 30)**

**{**

**if (par[k][u] != par[k][v]) {**

**u = par[k][u];**

**v = par[k][v];**

**}**

**}**

**assert(par[0][u] == par[0][v]);**

**return par[0][u];**

**}**

**int dist(int u, int v) {**

**int w = this->operator()(u, v);**

**return dep[u] + dep[v] - dep[w] \* 2;**

**}**

**// path from u to v (including u, v)**

**vi path(int u, int v) {**

**int l = this->operator()(u, v);**

**vi ul, vl;**

**while (l != u) {**

**ul.pb(u);**

**u = par[0][u];**

**}**

**while (l != v) {**

**vl.pb(v);**

**v = par[0][v];**

**}**

**ul.pb(l);**

**ul.insert(ul.end(), rall(vl));**

**return ul;**

**}**

**};**

**//lowlink.cpp**

**class LCA {**

**int n;**

**vvi G;**

**vi dep;**

**vvi par;**

**void dfs(int u, int p, int d) {**

**par[0][u] = p;**

**dep[u] = d;**

**for (int v: G[u]) if (v != p) dfs(v, u, d + 1);**

**}**

**void init() {**

**n = G.size();**

**dep.assign(n, -1);**

**par.resize(30);**

**rep(i, 30)**

**par[i].resize(n);**

**rep(i, n)**

**if (dep[i] == -1) dfs(i, -1, 0);**

**rep(k, 29)**

**rep(i, n)**

**{**

**if (par[k][i] < 0) par[k + 1][i] = -1;**

**else par[k + 1][i] = par[k][par[k][i]];**

**}**

**}**

**public:**

**LCA(const vvi &G) : G(G) { init(); }**

**int operator()(int u, int v) {**

**if (dep[u] > dep[v]) swap(u, v);**

**rep(k, 30)**

**if ((dep[v] - dep[u]) >> k & 1) v = par[k][v];**

**if (u == v) return u;**

**rrep(k, 30)**

**{**

**if (par[k][u] != par[k][v]) {**

**u = par[k][u];**

**v = par[k][v];**

**}**

**}**

**assert(par[0][u] == par[0][v]);**

**return par[0][u];**

**}**

**int dist(int u, int v) {**

**int w = this->operator()(u, v);**

**return dep[u] + dep[v] - dep[w] \* 2;**

**}**

**// path from u to v (including u, v)**

**vi path(int u, int v) {**

**int l = this->operator()(u, v);**

**vi ul, vl;**

**while (l != u) {**

**ul.pb(u);**

**u = par[0][u];**

**}**

**while (l != v) {**

**vl.pb(v);**

**v = par[0][v];**

**}**

**ul.pb(l);**

**ul.insert(ul.end(), rall(vl));**

**return ul;**

**}**

**};**

**//MCF.cpp**

**template<class Cap, class Cost>**

**class MCF {**

**struct \_edge {**

**int to, rev;**

**Cap cap;**

**Cost cost;**

**\_edge(int to, int rev, Cap cap, Cost cost) : to(to), rev(rev), cap(cap), cost(cost) {}**

**};**

**int n;**

**vp pos;**

**vector <vector<\_edge>> G;**

**public:**

**explicit MCF(int n) : n(n), G(n) {}**

**int add\_edge(int from, int to, Cap cap, Cost cost) {**

**assert(0 <= from and from < n);**

**assert(0 <= to and to < n);**

**assert(from != to);**

**pos.eb(from, G[from].size());**

**G[from].eb(to, G[to].size(), cap, cost);**

**G[to].eb(from, G[from].size() - 1, 0, -cost);**

**return pos.size() - 1;**

**}**

**struct edge {**

**int from, to;**

**Cap cap, flow;**

**Cost cost;**

**edge(int from, int to, Cap cap, Cap flow, Cost cost) : from(from), to(to), cap(cap), flow(flow), cost(cost) {}**

**};**

**edge get\_edge(int i) {**

**assert(0 <= i && i < int(pos.size()));**

**auto e = G[pos[i].first][pos[i].second];**

**auto re = G[e.to][e.rev];**

**return edge(pos[i].first, e.to, e.cap + re.cap, re.cap, e.cost);**

**}**

**vector <edge> edges() {**

**int m = pos.size();**

**vector <edge> res(m);**

**rep(i, m)**

**res[i] = get\_edge(i);**

**return res;**

**}**

**pair <Cap, Cost> flow(int s, int t) {**

**return flow(s, t, numeric\_limits<Cap>::max());**

**}**

**pair <Cap, Cost> flow(int s, int t, Cap flow\_limit) {**

**return slope(s, t, flow\_limit).back();**

**}**

**vector <pair<Cap, Cost>> slope(int s, int t) {**

**return slope(s, t, numeric\_limits<Cap>::max());**

**}**

**// this must not be called more than once**

**// O(F (E + V) log V)**

**vector <pair<Cap, Cost>> slope(int s, int t, Cap flow\_limit) {**

**assert(0 <= s and s < n);**

**assert(0 <= t and t < n);**

**assert(s != t);**

**// variants (C = maxcost):**

**// -(n-1)C <= dual[s] <= dual[i] <= dual[t] = 0**

**// reduced cost (= e.cost + dual[e.from] - dual[e.to]) >= 0 for all \_edge**

**vector <Cost> dual(n, 0), dist(n);**

**vi pv(n), pe(n);**

**vb seen(n);**

**auto dual\_ref = [&]() -> bool {**

**fill(all(dist), numeric\_limits<Cost>::max());**

**fill(all(pv), -1);**

**fill(all(pe), -1);**

**fill(all(seen), false);**

**priority\_queue < pair < Cost, int >, vector < pair < Cost, int >>, greater < pair < Cost, int>>> q;**

**dist[s] = 0;**

**q.emplace(0, s);**

**while (not q.empty()) {**

**int u = q.top().second;**

**q.pop();**

**if (seen[u]) continue;**

**seen[u] = true;**

**if (u == t) break;**

**// dist[u] = shortest(s, u) + dual[s] - dual[u]**

**// dist[u] >= 0 (all reduced cost are positive)**

**// dist[u] <= (n-1)C**

**rep(i, G[u].size())**

**{**

**auto e = G[u][i];**

**if (seen[e.to] or !e.cap) continue;**

**// |-dual[e.to] + dual[u]| <= (n-1)C**

**// cost <= C - -(n-1)C + 0 = nC**

**Cost cost = e.cost - dual[e.to] + dual[u];**

**if (chmin(dist[e.to], dist[u] + cost)) {**

**pv[e.to] = u;**

**pe[e.to] = i;**

**q.emplace(dist[e.to], e.to);**

**}**

**}**

**}**

**if (!seen[t]) {**

**return false;**

**}**

**rep(u, n)**

**{**

**if (!seen[u]) continue;**

**// dual[u] = dual[u] - dist[t] + dist[u]**

**// = dual[u] - (shortest(s, t) + dual[s] - dual[t]) + (shortest(s, u) + dual[s] - dual[u])**

**// = - shortest(s, t) + dual[t] + shortest(s, v)**

**// = shortest(s, v) - shortest(s, t) >= 0 - (n-1)C**

**dual[u] -= dist[t] - dist[u];**

**}**

**return true;**

**};**

**Cap flow = 0;**

**Cost cost = 0, prev\_cost\_per\_flow = -1;**

**vector <pair<Cap, Cost>> res;**

**res.eb(flow, cost);**

**while (flow < flow\_limit) {**

**if (!dual\_ref()) break;**

**Cap c = flow\_limit - flow;**

**for (int u = t; u != s; u = pv[u]) {**

**chmin(c, G[pv[u]][pe[u]].cap);**

**}**

**for (int u = t; u != s; u = pv[u]) {**

**auto &e = G[pv[u]][pe[u]];**

**e.cap -= c;**

**G[u][e.rev].cap += c;**

**}**

**Cost d = -dual[s];**

**flow += c;**

**cost += c \* d;**

**if (prev\_cost\_per\_flow == d) {**

**res.pop\_back();**

**}**

**res.eb(flow, cost);**

**prev\_cost\_per\_flow = d;**

**}**

**return res;**

**}**

**};**

**//rerooting.cpp**

**template<class D>**

**class rerooting {**

**using T = typename D::T;**

**int n;**

**vvi tree;**

**vector<vector<T>> dp;**

**vector<T> ans;**

**T dfs(int u = 0, int p = -1) {**

**T sum = D::id;**

**dp[u].resize(tree[u].size());**

**rep(i, tree[u].size()) {**

**int v = tree[u][i];**

**if (v == p) continue;**

**dp[u][i] = dfs(v, u);**

**sum = D::merge(sum, D::add\_root(dp[u][i], v, u));**

**}**

**return sum;**

**}**

**void dfs2(T dpP, int u = 0, int p = -1) {**

**int sz = tree[u].size();**

**rep(i, sz) if (tree[u][i] == p) dp[u][i] = dpP;**

**vector<T> sumL(sz + 1, D::id), sumR(sz + 1, D::id);**

**rep(i, sz) sumL[i + 1] = D::merge(sumL[i], D::add\_root(dp[u][i], tree[u][i], u));**

**rrep(i, sz) sumR[i] = D::merge(sumR[i + 1], D::add\_root(dp[u][i], tree[u][i], u));**

**ans[u] = D::add\_root(sumL[sz], u, -1);**

**rep(i, sz) {**

**int v = tree[u][i];**

**if (v == p) continue;**

**T t = D::merge(sumL[i], sumR[i + 1]);**

**dfs2(t, v, u);**

**}**

**}**

**public:**

**explicit rerooting(const vvi &tree) : n(tree.size()), tree(tree), dp(n), ans(n) {**

**dfs();**

**dfs2(D::id);**

**};**

**T get\_ans(int i) {**

**return ans[i];**

**}**

**};**

**struct D {**

**using T = ;**

**static const T id = ;**

**static T merge(const T &a, const T &b) {**

**return ;**

**}**

**// u : root, p : parent of u**

**static T add\_root(const T &a, [[maybe\_unused]] int u, [[maybe\_unused]] int p) {**

**return ;**

**}**

**};**

**//SCC.cpp**

**class SCC {**

**int n;**

**vvi G;**

**vi ord, low;**

**stack<int> st;**

**void dfs(int u, int &k) {**

**ord[u] = low[u] = k++;**

**st.push(u);**

**for (int v: G[u]) {**

**if (ord[v] == -1) {**

**dfs(v, k);**

**chmin(low[u], low[v]);**

**} else {**

**chmin(low[u], ord[v]);**

**}**

**}**

**if (low[u] == ord[u]) {**

**while (true) {**

**int now = st.top();**

**st.pop();**

**ord[now] = inf;**

**id[now] = num;**

**if (now == u) break;**

**}**

**++num;**

**}**

**}**

**public:**

**// number of components**

**int num;**

**vi id;**

**vvi scc\_list;**

**SCC(const vvi &G) : G(G) {**

**n = G.size();**

**ord.assign(n, -1);**

**low.resize(n);**

**id.resize(n);**

**num = 0;**

**int k = 0;**

**rep(i, n)**

**if (ord[i] == -1) dfs(i, k);**

**vi cnt(num);**

**rep(i, n)**

**{**

**id[i] = num - 1 - id[i];**

**++cnt[id[i]];**

**}**

**scc\_list.resize(num);**

**rep(i, num)**

**scc\_list[i].reserve(cnt[i]);**

**rep(i, n)**

**scc\_list[id[i]].pb(i);**

**}**

**};**

**//topological\_sort.cpp**

**vi topological\_sort(const vvi &G) {**

**int n = G.size();**

**vi in(n);**

**rep(i, n)**

**for (int j : G[i]) in[j]++;**

**queue<int> q;**

**rep(i, n)**

**if (!in[i]) q.push(i);**

**vi res;**

**while (!q.empty()) {**

**int u = q.front();**

**q.pop();**

**res.pb(u);**

**for (int v : G[u]) {**

**if (--in[v] == 0) q.push(v);**

**}**

**}**

**return res;**

**}**

**/\*math\*/**

**//gcdlcm.cpp**

**ll gcd(ll a, ll b) { return b?gcd(b,a%b):a;}**

**ll lcm(ll a, ll b) { return a/gcd(a,b)\*b;}**

**//combination.cpp**

**class combination {**

**public:**

**vector <mint> fact, ifact;**

**combination(int n) : fact(n + 1), ifact(n + 1) {**

**fact[0] = 1;**

**for (int i = 1; i <= n; ++i) fact[i] = fact[i - 1] \* i;**

**ifact[n] = fact[n].inv();**

**for (int i = n; i >= 1; --i) ifact[i - 1] = ifact[i] \* i;**

**}**

**mint operator()(int n, int k) {**

**if (k < 0 || k > n) return 0;**

**return fact[n] \* ifact[k] \* ifact[n - k];**

**}**

**} comb();**

**//convolution.cpp**

**using comp = complex<double>;**

**const double PI = acos(-1);**

**// Fast Fourier Transform**

**void fft(vector<comp> &v, bool inverse = false) {**

**int n = v.size();**

**int h = \_\_builtin\_ctz(n);**

**assert(1 << h == n);**

**rep(i, n) {**

**int j = 0;**

**rep(k, h) j |= (i >> k & 1) << (h - 1 - k);**

**if (i < j) swap(v[i], v[j]);**

**}**

**// b \* 2 -> size of block**

**for (int b = 1; b < n; b \*= 2) {**

**rep(j, b) {**

**comp w = polar(1.0, (2 \* PI) / (2 \* b) \* j \* (inverse ? 1 : -1));**

**for (int k = 0; k < n; k += b \* 2) {**

**comp s = v[j + k];**

**comp t = v[j + k + b] \* w;**

**v[j + k] = s + t;**

**v[j + k + b] = s - t;**

**}**

**}**

**}**

**if (inverse) rep(i, n) v[i] /= n;**

**}**

**// sum\_{i + j = k} {a\_i \* b\_j} = c\_k**

**vd convolution(const vd &a, const vd &b) {**

**int s = a.size() + b.size() - 1;**

**int t = 1;**

**while (t < s) t \*= 2;**

**vector<comp> A(t), B(t);**

**rep(i, a.size()) A[i] = comp(a[i], 0);**

**rep(i, b.size()) B[i] = comp(b[i], 0);**

**fft(A);**

**fft(B);**

**rep(i, t) A[i] \*= B[i];**

**fft(A, true);**

**vd res(s);**

**rep(i, s) res[i] = A[i].real();**

**return res;**

**}**

**vl convolution(const vi &f, const vi &g) {**

**vd nf(f.size()), ng(g.size());**

**rep(i, f.size()) nf[i] = f[i];**

**rep(i, g.size()) ng[i] = g[i];**

**vd v = convolution(nf, ng);**

**vl res(v.size());**

**rep(i, v.size()) res[i] = llround(move(v[i]));**

**return res;**

**}**

**//crt.cpp**

**// solve x = r[i] (mod m[i])**

**// return (R, M), when the answer is x = R (mod M)**

**// return (0, 0), when there is no answer**

**// condition: lcm(m[i]) <= LLONG\_MAX**

**LP crt(const vl &r, const vl &m) {**

**assert(r.size() == m.size());**

**int n = r.size();**

**ll R = 0, M = 1;**

**rep(i, n)**

**{**

**assert(m[i] >= 1);**

**ll p, q;**

**ll g = ext\_gcd(M, m[i], p, q); // p is inv of M/d (mod. m[i]/d)**

**if ((r[i] - R) % g != 0) return P(0, 0);**

**ll tmp = (r[i] - R) / g \* p % (m[i] / g);**

**R += M \* tmp;**

**M \*= m[i] / g;**

**R = (R % M + M) % M;**

**}**

**return {R, M};**

**}**

**//diophantine.cpp**

**// solve a + bx = c (mod m)**

**ll diophantine(ll a, ll b, ll c, ll m) {**

**int g = gcd(b, m);**

**assert(a % g == c % g);**

**b = (b % m + m) % m**

**ll res = ((c - a) / g % m + m) % m;**

**res \*= mod\_inverse(b / g, m / g);**

**res %= m / g;**

**return res;**

**}**

**//euler\_totient.cpp**

**int euler\_totient(int n) {**

**prime<int> pr(n);**

**auto fac = pr.unique\_factor();**

**int phi = n;**

**for (int p : fac) {**

**phi /= p;**

**phi \*= p - 1;**

**}**

**return phi;**

**}**

**//ext\_gcd.cpp**

**// return gcd(a, b)**

**// solve ax + by = gcd(a, b)**

**// |x| <= b, |y| <= a (if a\*b != 0)**

**ll ext\_gcd(ll a, ll b, ll &x, ll &y) {**

**if (b == 0) {**

**x = 1;**

**y = 0;**

**return a;**

**}**

**ll d = ext\_gcd(b, a % b, y, x);**

**y -= a / b \* x;**

**return d;**

**}**

**//floor\_sum.cpp**

**// sum floor((a \* i + b) / m), i = 0 to n-1**

**ll floor\_sum(ll n, ll m, ll a, ll b) {**

**ll ans = 0;**

**if (a >= m) {**

**ans += (n - 1) \* n \* (a / m) / 2;**

**a %= m;**

**}**

**if (b >= m) {**

**ans += n \* (b / m);**

**b %= m;**

**}**

**ll y\_max = (a \* n + b) / m, x\_max = (y\_max \* m - b);**

**if (y\_max == 0) return ans;**

**ans += (n - (x\_max + a - 1) / a) \* y\_max;**

**ans += floor\_sum(y\_max, a, m, (a - x\_max % a) % a);**

**return ans;**

**}**

**//fmt.cpp**

**// Fast Mobius Transform**

**// inverse fzt**

**template<class T>**

**void fmt(vector<T> &v) {**

**int n = v.size();**

**assert(n == 1 << \_\_builtin\_ctz(n));**

**for (int i = 1; i < n; i <<= 1) {**

**rep(j, n) {**

**if ((j & i) == 0) {**

**// convolve subset**

**v[j | i] -= v[j];**

**// convolve superset**

**// v[j] -= v[j | i];**

**}**

**}**

**}**

**}**

**//fzt.cpp**

**// Fast Zeta Transform**

**template<class T>**

**void fzt(vector<T> &v) {**

**int n = v.size();**

**assert(n == 1 << \_\_builtin\_ctz(n));**

**for (int i = 1; i < n; i <<= 1) {**

**rep(j, n) {**

**if ((j & i) == 0) {**

**// convolve subset**

**v[j | i] += v[j];**

**// convolve superset**

**// v[j] += v[j | i];**

**}**

**}**

**}**

**}**

**//mod\_inv.cpp**

**// composite\_mod**

**// gcd(a, mod) must be 1**

**ll mod\_inverse(ll a, ll mod) {**

**ll x, y;**

**extGCD(a, mod, x, y);**

**return (x % mod + mod) % mod;**

**}**

**//prime.cpp**

**template<typename T>**

**class prime {**

**T n;**

**public:**

**prime(T n) : n(n) {}**

**map<int, int> factor\_list() {**

**if (n == 1) return {};**

**T nn = n;**

**map<int, int> ret;**

**for (int i = 2; (ll) i \* i <= n; i++) {**

**if (nn % i != 0) continue;**

**int cnt = 0;**

**while (nn % i == 0) {**

**cnt++;**

**nn /= i;**

**}**

**ret[i] = cnt;**

**}**

**if (nn != 1) ret[nn]++;**

**return ret;**

**}**

**vi unique\_factor() {**

**map<int, int> m = factor\_list();**

**vi ret;**

**for (P p : m) ret.pb(p.first);**

**return ret;**

**};**

**bool is\_prime() {**

**auto v = factor\_list();**

**return v.size() == 1 && v[0].second == 1;**

**}**

**int count\_divisor() {**

**int ret = 1;**

**map<int, int> fl = factor\_list(n);**

**for (auto p : fl) ret \*= p.second + 1;**

**return ret;**

**};**

**vector <T> enum\_divisors() {**

**vector <T> res;**

**for (T i = 1; i \* i <= n; i++) {**

**if (n % i == 0) {**

**res.pb(i);**

**if (n / i != i) res.pb(n / i);**

**}**

**}**

**sort(all(res));**

**return res;**

**}**

**};**

**/\*other\*/**

**//cint.cpp**

**#include <boost/multiprecision/cpp\_int.hpp>**

**using cint = boost::multiprecision::cpp\_int;**

**//compress.cpp**

**struct Compress{**

**vl v;**

**void insert(ll x){**

**v.push\_back(x);**

**}**

**void init(){**

**sort(all(v));**

**v.erase(std::unique(v.begin(), v.end()), v.end());**

**}**

**ll get(ll x){**

**return lower\_bound(all(v),x)-v.begin();**

**}**

**ll rev(ll x){**

**return v[x];**

**}**

**ll getmax(){**

**return v.size();**

**}**

**};**

**//inversion\_number.cpp**

**ll inversion\_number(int n,const vi& v) {**

**ll ret = 0;**

**BIT<int> bt(n);**

**rep(i,n) {**

**ret += i-bt.sum(v[i]);**

**bt.add(v[i]);**

**}**

**return ret;**

**}**

**//lis.cpp**

**int lis(const vi &v, bool strict = true) {**

**int n = v.size();**

**vi dp(n, inf);**

**dp[0] = v[0];**

**int now = 0;**

**rep(i, 1, n)**

**{**

**if (v[i] > dp[now]) {**

**dp[now + 1] = v[i];**

**now++;**

**} else {**

**if (strict) {**

**auto it = lower\_bound(all(dp), v[i]);**

**\*it = v[i];**

**} else {**

**auto it = upper\_bound(all(dp), v[i]);**

**\*it = v[i];**

**}**

**}**

**}**

**return now + 1;**

**}**

**//Mo.cpp**

**int lis(const vi &v, bool strict = true) {**

**int n = v.size();**

**vi dp(n, inf);**

**dp[0] = v[0];**

**int now = 0;**

**rep(i, 1, n)**

**{**

**if (v[i] > dp[now]) {**

**dp[now + 1] = v[i];**

**now++;**

**} else {**

**if (strict) {**

**auto it = lower\_bound(all(dp), v[i]);**

**\*it = v[i];**

**} else {**

**auto it = upper\_bound(all(dp), v[i]);**

**\*it = v[i];**

**}**

**}**

**}**

**return now + 1;**

**}**

**//QCFium.cpp**

**#pragma GCC target("avx2")**

**#pragma GCC optimize("O3")**

**#pragma GCC optimize("unroll-loops")**

**//rnd.cpp**

**random\_device seed\_gen;**

**mt19937\_64 engine(seed\_gen());**

**template<class T>**

**T rnd(T min\_val, T max\_val) {**

**assert(min\_val <= max\_val);**

**uniform\_int\_distribution <T> dist(min\_val, max\_val);**

**return dist(engine);**

**}**

**template<>**

**double rnd(double min\_val, double max\_val) {**

**assert(min\_val - 1e8 <= max\_val);**

**uniform\_real\_distribution<double> dist(min\_val, max\_val);**

**return dist(engine);**

**}**

**//zip.cpp**

**template<class T>**

**void resemble(vector <T> &v) {}**

**template<class T, class... Tail>**

**void resemble(vector <T> &v, vector <T> &head, Tail &...tail) {**

**for (T &e : head) v.pb(e);**

**resemble(v, tail...);**

**}**

**template<class T>**

**void renumber(vector <T> &v) {}**

**template<class T, class... Tail>**

**void renumber(vector <T> &v, vector <T> &head, Tail &...tail) {**

**for (T &e : head) e = lower\_bound(all(v), e) - v.begin();**

**renumber(v, tail...);**

**}**

**template<class T, class... Tail>**

**vector <T> zip(vector <T> &head, Tail &... tail) {**

**vector <T> v;**

**resemble(v, head, tail...);**

**sort(all(v));**

**v.erase(unique(all(v)), v.end());**

**renumber(v, head, tail...);**

**return v;**

**}**

**/\*string\*/**

**//manacher.cpp**

**vi manacher(const string &s) {**

**int n = SZ(s);**

**vi res(n);**

**int i = 0, j = 0;**

**while (i < n) {**

**while (i - j >= 0 && i + j < n && s[i - j] == s[i + j]) ++j;**

**res[i] = j;**

**int k = 1;**

**while (i - k >= 0 && k + res[i - k] < j) res[i + k] = res[i - k], ++k;**

**i += k;**

**j -= k;**

**}**

**return res;**

**}**

**//rolling\_hash.cpp**

**using ull = unsigned long long;**

**const ull mod = (1ull << 61) - 1;**

**const ull mask30 = (1ull << 30) - 1;**

**const ull mask31 = (1ull << 31) - 1;**

**const ull mask61 = mod;**

**ull calc\_mod(ull x) {**

**ull xu = x >> 61;**

**ull xd = x & mask61;**

**ull res = xu + xd;**

**if (res >= mod) res -= mod;**

**return res;**

**}**

**// a\*b mod 2^61-1**

**ull mul(ull a,ull b) {**

**ull au = a>>31;**

**ull ad = a&mask31;**

**ull bu = b>>31;**

**ull bd = b&mask31;**

**ull mid = ad\*bu+au\*bd;**

**ull midu = mid>>30;**

**ull midd = mid&mask30;**

**return au\*bu\*2+midu+(midd<<31)+ad\*bd;**

**}**

**class rolling\_hash {**

**ull base1;**

**ull base2;**

**int n;**

**string s;**

**vector<ull> hash1,hash2,pow1,pow2;**

**void init() {**

**random\_device rnd;**

**mt19937\_64 mt(rnd());**

**uniform\_int\_distribution<ull> dist(2,mod-2);**

**base1 = dist(mt);**

**base2 = dist(mt);**

**// base1 = calc\_mod(mt());**

**// base2 = calc\_mod(mt());**

**// while(base1 < 2 || base1 > mod-2) base1 = calc\_mod(mt());**

**// while(base2 < 2 || base2 > mod-2) base2 = calc\_mod(mt());**

**hash1.assign(n+1,0);**

**hash2.assign(n+1,0);**

**pow1.assign(n+1, 1);**

**pow2.assign(n+1,1);**

**rep(i,n) {**

**hash1[i+1] = calc\_mod(mul(hash1[i],base1)+s[i]);**

**hash2[i+1] = calc\_mod(mul(hash2[i],base2)+s[i]);**

**pow1[i+1] = calc\_mod(mul(pow1[i], base1));**

**pow2[i+1] = calc\_mod(mul(pow2[i], base2));**

**}**

**}**

**public:**

**rolling\_hash(string s):s(s),n(s.size()) {**

**init();**

**}**

**// return hash of [l,r) of S**

**pair<ull,ull> get(int l,int r) {**

**ll res1 = calc\_mod(hash1[r]+mod\*4-mul(hash1[l], pow1[r-l]));**

**ll res2 = calc\_mod(hash2[r]+mod\*4-mul(hash2[l], pow2[r-l]));**

**return make\_pair(res1,res2);**

**}**

**// return hash of T**

**pair<ull,ull> get(string t) {**

**ull ht1 = 0,ht2 = 0;**

**rep(i,t.size()) {**

**ht1 = calc\_mod(mul(ht1,base1)+t[i]);**

**ht2 = calc\_mod(mul(ht2,base2)+t[i]);**

**}**

**return make\_pair(ht1,ht2);**

**}**

**int count(string t) {**

**if(t.size() > n) return 0;**

**pair<ull,ull> ht = get(t);**

**int res = 0;**

**rep(i,n-t.size()+1) {**

**if(get(i,i+t.size()) == ht) res++;**

**}**

**return res;**

**}**

**};**

**//run\_length\_compress.cpp**

**template<class T>**

**vector <pair<T, int>> run\_length\_compress(const vector <T> &v) {**

**if (v.empty()) return {};**

**vector <pair<T, int>> res;**

**T now = v[0];**

**int cnt = 1;**

**rep(i, 1, v.size())**

**{**

**if (now != v[i]) {**

**res.eb(now, cnt);**

**now = v[i];**

**cnt = 0;**

**}**

**cnt++;**

**}**

**res.eb(now, cnt);**

**return res;**

**}**

**vector <pair<char, int>> run\_length\_compress(const string &s) {**

**if (s.empty()) return {};**

**vector <pair<char, int>> res;**

**char now = s[0];**

**int cnt = 1;**

**rep(i, 1, s.size())**

**{**

**if (now != s[i]) {**

**res.eb(now, cnt);**

**now = s[i];**

**cnt = 0;**

**}**

**cnt++;**

**}**

**res.eb(now, cnt);**

**return res;**

**}**

**//trie.cpp**

**template<int char\_size>**

**struct Node {**

**vi next;**

**int accept;**

**int c;**

**Node(int c) : next(char\_size, -1), accept(0), c(c) {}**

**};**

**template<int char\_size, int base>**

**class trie {**

**using node = Node<char\_size>;**

**vector<node> nodes;**

**public:**

**trie() { nodes.pb(node(0)); }**

**void insert(const string &s) {**

**int pos = 0;**

**rep(i, s.size()) {**

**int c = (int) s[i] - base;**

**int &next = nodes[pos].next[c];**

**if (next == -1) {**

**next = nodes.size();**

**nodes.pb(node(c));**

**}**

**pos = next;**

**}**

**nodes[pos].accept++;**

**}**

**int count(const string &s) {**

**int pos = 0;**

**rep(i, s.size()) {**

**int c = (int) s[i] - base;**

**int &next = nodes[pos].next[c];**

**if (next == -1) {**

**return 0;**

**}**

**pos = next;**

**}**

**return nodes[pos].accept;**

**}**

**};**

**//z\_algorithm.cpp**

**vi z\_algorithm(const string &s) {**

**if (s.empty()) return {};**

**vi res(s.size());**

**res[0] = s.size();**

**int i = 1, j = 0;**

**while (i < s.size()) {**

**while (i + j < s.size() and s[j] == s[i + j]) j++;**

**res[i] = j;**

**if (j == 0) {**

**i++;**

**continue;**

**}**

**int k = 1;**

**while (i + k < s.size() and k + res[k] < j) res[i + k] = res[k], k++;**

**i += k, j -= k;**

**}**

**return res;**

**}**