2024-08-29

Universidade de São Paulo - pato

antoniomsah

darling51707

Contents

Template hashing

```
sed -n $2','$3' p' $1 | cpp -dD -P -fpreprocessed | tr -d '[:space:]'
| md5sum | cut -c-6
```

The cli command `md5sum` is used to validate if the template was written correctly.

Write the one liner above without line breaks and save it as `hash.sh`, and run

```
chmod +x hash.sh
```

to make it executable. vai luan

To get the hash in `file.cpp` between lines `<l1>` and `<l2>`, run

```
./hash.sh <file.cpp> <l1> <l2>
```

Hash is made by scope, delimited by the curly brackets (`"{"`, `"}"`).

The empty hash is `d41d8` while the `m5dsum` of `hash.sh` is `9cd12`.

Matrix Inverse 2x2

$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}, A^{-1} = \frac{1}{ad-bc}\begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$

Pick Theorem

$A = i + \frac{b}{2} - 1$, where i is the number of lattice points inside the polygon and b is the number on boundary. Work for simple polygon.

Burnside Lemma

Let G be a finite group that acts on set X. Let $X^g := \{x \in X \mid g.x = x\}$. The number of orbits $|X/G|$ is

$$|X/G| = \frac{1}{|G|}\sum_{g \in G}|X^g|$$

Lucas Theorem

For prime p, $\binom{n}{m} = \prod_{i=0}^{n}\binom{n_i}{m_i} \pmod p$

Catalan

$C_n = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{n!*(n+1)!} \mid C_n^k = \frac{k+1}{n+k+1}\binom{2n+k}{n}$

Stirling numbers of the first kind

These are the number of permutation of size n with exactly k cycles

$$\begin{bmatrix} n \\ k \end{bmatrix} = (n-1)\begin{bmatrix} n-1 \\ k \end{bmatrix} + \begin{bmatrix} n-1 \\ k-1 \end{bmatrix}$$

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix} = 1, \begin{bmatrix} n \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ n \end{bmatrix} = 1$$

Stirling numbers of the second kind

These are the number of ways to partition n into exactly k non-empty sets.

$$\begin{Bmatrix} n \\ k \end{Bmatrix} = (k)\begin{Bmatrix} n-1 \\ k \end{Bmatrix} + \begin{Bmatrix} n-1 \\ k-1 \end{Bmatrix}$$

$$\begin{Bmatrix} n \\ k \end{Bmatrix} = \frac{1}{k!}\sum_{j=0}^{k}(-1)^{k-j}\binom{k}{j}j^n$$

$$\begin{Bmatrix} 0 \\ 0 \end{Bmatrix} = 1, \begin{Bmatrix} n \\ 0 \end{Bmatrix} = \begin{Bmatrix} 0 \\ n \end{Bmatrix} = 1$$

Planar Graph

If G has k connected components, then $n - m + f = k + 1$.

$m \leq 3n - 6$. If G has no triangles, $m \leq 2n - 4$.

The minimum degree is less of equal 5. And can be 6 colored in O(n+m)/

Spherical coordinates

$x = r\sin(\theta)\cos(\varphi) \mid r = \sqrt{x^2 + y^2 + z^2}$

$y = r\sin(\theta)\sin(\varphi) \mid (\theta) = \text{acos}\left(\frac{z}{r}\right)$

$z = r\cos(\theta) \mid (\varphi) = \text{atan2}\ (y, x)$

series and sums

$\frac{1}{1-x} = \sum_{k=0} x^k \mid e^x = \sum_{k=0}\frac{x^k}{k!} \mid \ln(1+x) = \sum_{k=1}(-1)^{k+1}\frac{x^k}{k}$

$\sqrt{1+x} = 1 + \sum_{k=1}(-1)^{k+1}\frac{x^k}{2^{2k-1}} \mid \frac{1}{\sqrt{1-x}} = \sum_{k=0}\frac{x^k}{4^k}\binom{2k}{k}$

$\sin x = \sum_{k=0}(-1)^k\frac{(x)^{2k+1}}{(2k+1)!} \mid \cos x = \sum_{k=0}(-1)^k\frac{x^{2k}}{(2k)!}$

$1 + 2 + ... = \frac{n(n+1)}{2} \mid 1^2 + 2^2 + .. = \frac{n(2n+1)(n+1)}{6}$

$1^3 + 2^3 + .. = \frac{n^2(n+1)^2}{4} \mid 1^4 + 2^4 + ... = \frac{n(n+1)(2n+1)(3n^2+3n+1)}{30}$

Code

divide_and_conquer

```
d41 // Divide and Conquer DP
d41 //
d41 // A dp of the form
d41 //     dp[i][j] = min_{k < j}(dp[i - 1][k]  + cost(k, j))
d41 // can be solved in O(m n logn) with divide and conquer
    optimization if we have that
d41 //     opt[i][j] <= opt[i][j + 1]
d41 // where
d41 //    dp[i][j] = dp[i - 1][opt[i][j]]  + cost(opt[i][j], j).
d41 //
d41 // Complexity: O(m n logn) time (for a partition in m
    subarrays of an array of size n)
d41 //             O(n) memory
d41
d41
547 ll dp[MAX][2];
d41
94b void solve(int k, int l, int r, int lk, int rk) {
de6     if (l > r) return;
109     int m = (l+r)/2, p = -1;
d2b     auto& ans = dp[m][k&1] = LINF;
6e2     for (int i = max(m, lk); i <= rk; i++) {
d73         ll at = dp[i+1][~k&1] + cost(m, i);
57d         if (at < ans) ans = at, p = i;
5ed     }
1ee     solve(k, l, m-1, lk, p), solve(k, m+1, r, p, rk);
b62 }
d41
cf1 ll DC(int n, int k) {
321     dp[n][0] = dp[n][1] = 0;
f27     for (int i = 0; i < n; i++) dp[i][0] = LINF;
b76     for (int i = 1; i <= k; i++) solve(i, 0, n-i, 0, n-i);
8e7     return dp[0][k&1];
5e9 }
```

dynamic_cht

```
d41 // Dynamic Convex Hull Trick
d41 //
d41 // Description:
d41 //      Maintains the convex hull of some functions.
d41 //     Copied from github.com/brunomaletta/Biblioteca/blob/
    master/Codigo/Estruturas/chtDinamico.cpp.
d41 //
d41 // Functions:
d41 //      add(a,b): adds line (ax+b) to the convex hull.
d41 //      query(x): returns the maximum value of any line on
    point x.
d41 //
d41 // Complexity:
d41 //      add:    O(logn)
d41 //      query:  O(logn)
d41 //
d41 // Details:
d41 //     If you want to maintain the bottom convex hull, it is
d41 //      easier to just change the sign. Be careful with
    overflow
d41 //      on query. Can use __int128 to avoid.
d41
72c struct Line {
073     mutable ll a, b, p;
8e3     bool operator<(const Line& o) const { return a < o.a; }
abf     bool operator<(ll x) const { return p < x; }
469 };
d41
326 struct dynamic_hull : multiset<Line, less<>> {
33a     ll div(ll a, ll b) {
a20         return a / b - ((a ^ b) < 0 and a % b);
a8a     }
d41
bbb     void update(iterator x) {
459         if (next(x) == end()) x->p = INF;
            else if (x->a == next(x)->a) x->p = x->b >= next(x)-
eec >b ? INF : -INF;
424         else x->p = div(next(x)->b - x->b, x->a - next(x)->a);
d37     }
d41
71c     bool overlap(iterator x) {
```

```
f18         update(x);
cfa         if (next(x) == end()) return 0;
a4a         if (x->a == next(x)->a) return x->b >= next(x)->b;
d40         return x->p >= next(x)->p;
901     }
d41
176     void add(ll a, ll b) {
1c7         auto x = insert({a, b, 0});
4ab         while (overlap(x)) erase(next(x)), update(x);
dbc         if (x != begin() and !overlap(prev(x))) x = prev(x), update(x);
0fc         while (x != begin() and overlap(prev(x)))
4d2             x = prev(x), erase(next(x)), update(x);
48f     }
d41
4ad     ll query(ll x) {
229         assert(!empty());
7d1         auto l = *lower_bound(x);
d41         // if(l.a and abs(x) >= abs(INF/l.a)) return INF/2;
aba         return l.a * x + l.b;
3f5     }
905 };
```

knuth

```
d41 // Knuth DP
d41
d41 // A dp of the form
d41 //     dp[l][r] = min_{l < m < r}(dp[l][m] + dp[m][r])  + cost(l, r)
d41 // can be solved in O(n^2) with Knuth opmitization if we have that
d41 //     opt[l][r - 1] <= opt[l][r] <= opt[l + 1][r]
d41 // where
d41 //     dp[l][r] = dp[l][opt[l][r]] + dp[opt[l][r]][r] + cost(l, r).
d41 // Other sufficient condition (that implies the previous one) is
d41 //    given a <= b <= c <= d, we have:
d41 //        - quadrangle inequality: cost(a, c) + cost(b, d) <= cost(a, d) + cost(b, c)
d41 //        - monotonicity: cost(b, c) <= cost(a, d)
d41 //
d41 // Complexity: O(n^2) time
d41 //             O(n^2) memory
d41
297 ll knuth(int n) {
bdf     vector<vector<ll>> dp = vector<vector<ll>>(n, vector<ll>(n));
8f9     vector<vector<int>> opt = vector<vector<int>>(n, vector<int>(n));
83e     for(int k = 0; k <= n; k++) {
831         for(int l = 0; l + k <= n; l++) {
cdd             int r = l + k;
2b2             if(k < 2) {
653                 dp[l][r] = 0; // base case
358                 opt[l][r] = l;
2ad             }
740             dp[l][r] = INF;
cba             for(int m = opt[l][r - 1]; m <= opt[l + 1][r]; m++) {
e1b                 ll cur = dp[l][m] + dp[m][r] + cost(l, r); // must define O(1) cost function
a23                 if(cur < dp[l][r]) {
3b8                     dp[l][r] = cur;
f17                     opt[l][r] = m;
9e4                 }
5ee             }
22f         }
39e     }
ec9     return dp[0][n];
124 }
```

extra

fastIO

```
d41 // fast io
d41
7a5 int read_int() {
32b     bool minus = false;
c18     int result = 0;
d28     char ch;
ca3     ch = getchar();
31e     while (1) {
b92         if (ch == '-') break;
736         if (ch >= '0' && ch <= '9') break;
ca3         ch = getchar();
771     }
822     if (ch == '-') minus = true;
62e     else result = ch-'0';
31e     while (1) {
ca3         ch = getchar();
a46         if (ch < '0' || ch > '9') break;
f79         result = result*10 + (ch - '0');
722     }
28e     if (minus) return -result;
98e     else return result;
9b5 }
```

pragmas

```
d41 // Pragmas
881 #pragma GCC optimize("O3,unroll-loops")
827 #pragma GCC target("avx2,bmi,bmi2,lzcnt,popcnt")
```

random

```
d41 // Random
798 mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());
836 shuffle(permutation.begin(), permutation.end(), rng);
1ee uniform_int_distribution<int>(a,b)(rng);
```

template

```
d41 // template
d41
2b7 #include <bits/stdc++.h>
d41
```

```
ca4 using namespace std;
d41
f51 #define fastio ios_base::sync_with_stdio(0); cin.tie(0)
d41
90e using ll = long long;
d41
0e0 #define int ll
efe #define pb push_back
d69 #define all(a) a.begin(), a.end()
d41
21e void dbg_out() { cerr << endl; }
2dc template <typename H, typename... T>
f62 void dbg_out(H h, T... t) { cerr << ' ' << h; dbg_out(t...); }
89a #define dbg(...) { cerr << #__VA_ARGS__ << ':';
    dbg_out(__VA_ARGS__); }
d41
63d void solve(){
d41
fa9 }
d41
0dd signed main(){
9ae     fastio;
0f9     solve();
78d }
```

geometry

circle

```
d41 // Circle
d41 //
d41 // Complexity: O(1) unless stated
d41
d39 const Point<int> ccw(1,0);
d41
4fc template <class T>
d2f struct Circle {
b38     using P = Point<T>;
20e     using C = Circle;
d41
543     P c; T r;
ca1     Circle() {}
439     Circle(P c, T r) : c(c), r(r) {}
d41
d4d     bool in(C rhs) { return (c-rhs.c).norm2() <= (rhs.r-r)*(rhs.r-r); }
296     bool has(P p) { return (c-p).norm2() <= r*r; }
d41
d41     // returns intersection points between line and circle
a07     vector<P> intersect(Line<T> l) {
4f0         T h2 = r * r - l.dist2(c);
95d         if (h2 < -EPS) return {};
d41
dc8         P p = l.proj(c);
b23         P h = l.v * sqrt(h2) / l.v.norm();
0c8         if (h.norm() < EPS) return {p};
49c         return {p + h, p - h};
96e     }
d41
d41     // returns intersection points between two circles
```

```
193     vector<P> intersect(const C& rhs) const {
5e1         vector<P> inter;
2b3         C d = (c - rhs.c).norm();
c27         if (d > r + rhs.r + EPS or d + min(r,rhs.r) + EPS < max(r,rhs.r)) {
c17             return inter;
800         }
d41
2a0         T x = (d * d - rhs.r * rhs.r + r * r) / (d*2),
653           y = sqrt(r*r - x*x);
485         P v = (rhs.c - c) / d;
451         inter.push_back(c + v * x - v.rot(ccw) * y);
a29         if (y > EPS) inter.push_back(c + v * x + v.rot(ccw) * y);
c17         return inter;
a0a     }
d41
d41     // returns the tangents between two circles
834     vector<pair<P,P>> tangents(C rhs, bool inner = false) {
153         if (inner) rhs.r = -rhs.r;
260         P d = (rhs.c - c);
d7e         double dr = (r - rhs.r), d2 = d.norm2(), h2 = d2 - dr * dr;
fe3         if (d2 < EPS or h2 < -EPS) return {};
d41
002         vector<pair<P, P>> ret;
092         for (double sign : {-1, 1}) {
042             P v = (d * dr - d.perp() * sqrt(h2) * sign) / d2;
1da             ret.emplace_back(c + v * r, rhs.c + v * rhs.r);
19b         }
edf         return ret;
33b     }
d41
d41     // returns the tangent between a point and a circle
6e8     vector<pair<P,P>> tangents(P p, bool inner = false) {
9c1         return tangents(C(p, 0), inner);
310     }
d85 };
d41
d41 // Given a circle defined by points (a, b, c)
d41 // checks if d is contained in that circle.
199 bool in_circle(P a, P b, P c, P d) {
421     P ad = (a-d), bd = (b-d), cd = (c-d);
11f     return (
b51         ad.norm2() * cross(bd, cd) -
392         bd.norm2() * cross(ad, cd) +
271         cd.norm2() * cross(ad, bd)
871     ) >= 0;
573 };
```

convex_hull

```
d41 // Convex hull
d41 //
d41 // Description: Monotone chain algorithm.
d41 //
d41 // Details: Border represents the inclusion of collinear points
d41 //
d41 // Complexity: O(nlgn)
d41
67a template <typename T>
57f vector<T> convex_hull(vector<T> p, bool border=0){
905     sort(p.begin(), p.end());
d41     // removes duplicated points
582     p.erase(unique(p.begin(), p.end()), p.end());
d41
d55     if (p.size() == 1) return p;
f3d     int n = p.size(), m = 0;
1d4     vector<T> ch;
620     for (int i = 0; i < n; i++, m++) {
3f3         while (m > 1 and sign(cross(ch[m-2], ch[m-1], p[i])) <= -border) {
cc3             ch.pop_back(), m--;
2b0         }
3db         ch.push_back(p[i]);
458     }
d41
4f9     for (int i = n - 2, t = m; i >= 0; i--, m++) {
61e         while (m > t and sign(cross(ch[m-2], ch[m-1], p[i])) <= -border) {
cc3             ch.pop_back(), m--;
2e4         }
3db         ch.push_back(p[i]);
902     }
9d9     ch.pop_back(); // last element is duplicated
d41
66c     return ch;
30e }
```

halfplane

```
d41 // Halfplane
d41 //
d41 // Complexity: O(1) unless stated
d41
4fc template <class T>
c27 struct Halfplane {
b38     using P = Point<T>;
728     using H = Halfplane;
d41
812     P p, pq;
fe9     double ang;
d41
175     Halfplane() {}
58b     Halfplane(P p, P q) : p(p), pq(q-p) {
2a5         ang = atan2l(pq.y, pq.x);
973     }
d41
bc2     bool operator<(H h) const { return ang < h.ang; }
d41
062     bool operator==(H h) const { return fabsl(ang-h.ang) < EPS; }
d41
fb6     bool out(P r) { return (pq^(r-p)) < -EPS; }
d41
3a2     friend P inter(H s, H t) {
fd0         double alpha = ((t.p - s.p)^t.pq) / (s.pq^t.pq);
453         return s.p + s.pq*alpha;
5e7     }
bc0 };
```

halfplane_intersection

```
d41 // Halfplane intersection algorithm
d41 //
d41 // Description: Finds a convex polygon that is covered by all halfplanes.
d41 // If does not exist, then returns an empty vector.
d41 //
d41 // Complexity: O(nlgn)
d41
b37 vector<P> hp_intersect(vector<H>& h) {
2f9     P box[4] = {
70a         P(INF, INF),
948         P(-INF, INF),
c2b         P(-INF, -INF),
c4c         P(INF, -INF)
63b     };
d41
1cd     for (int i = 0; i < 4; i++) {
7d4         H aux(box[i], box[(i+1)%4]);
cd1         h.push_back(aux);
80f     }
d41
d77     sort(all(h));
17a     deque<H> dq;
486     int len = 0;
306     for (int i = 0; i < h.size(); i++) {
dab         while (len > 1 and h[i].out(inter(dq[len-1], dq[len-2]))) {
cfb             dq.pop_back(), --len;
6cb         }
d41
cdf         while (len > 1 and h[i].out(inter(dq[0], dq[1]))) {
21b             dq.pop_front(), --len;
edf         }
d41
26b         if (len > 0 and fabsl(h[i].pq^dq[len-1].pq) < EPS) {
82a             if (h[i].pq*dq[len-1].pq < 0.0) return vector<P>();
d41
8f5             if (h[i].out(dq[len-1].p)) {
cfb                 dq.pop_back(), --len;
a50             }
64e             else continue;
bee         }
d41
395         dq.push_back(h[i]);
250         ++len;
e45     }
d41
635     while (len > 2 and dq[0].out(inter(dq[len-1], dq[len-2]))) {
cfb         dq.pop_back(), --len;
d81     }
d41
2e3     while (len > 2 and dq[len-1].out(inter(dq[0], dq[1]))) {
21b         dq.pop_front(), --len;
365     }
```

```
d41
1a3     if (len < 3) return vector<P>();
d41
7e7     vector<P> ret(len);
cc7     for (int i = 0; i+1 < len; i++) {
01e         ret[i] = inter(dq[i], dq[i+1]);
00f     }
4fd     ret.back() = inter(dq[len-1], dq[0]);
edf     return ret;
ced }
```

intercircles

```
d41 // Circle intersection area algorithm
d41 //
d41 // Description: finds the area covered by at least k circles.
d41 //
d41 // Complexity: O(n^2 lgn) with high constant.
d41
659 using P = Point<long long>;
3dc using C = Circle<long long>;
d41
397 bool cmp(P a, P b, P o) {
a31     a = a - o, b = b - o;
318     return make_tuple(-a.side(), 0) < make_tuple(-b.side(),
    a^b);
2ce }
d41
c1b vector<double> intercircles(vector<C> &c) {
d41     // area covered by at least k circles
621     vector<double> cover(c.size()+1);
018     for (int i = 0; i < c.size(); i++) {
bfe         int k = 1;
74a         vector<pair<P,int>> p = {{c[i].c + P(1,0)*c[i].r,
    0}, {c[i].c - P(1,0)*c[i].r, 0}};
d41
226         for (int j = 0; j < c.size(); j++) {
bd4             bool b0 = c[i].in(c[j]), b1 = c[j].in(c[i]);
abe             if(b0 and (!b1 or i<j)) k++;
83e             else if (!b0 and !b1) {
793                 auto v = c[i].intersect(c[j]);
00b                 if (v.size() == 2) {
530                  p.push_back({v[0], 1}); p.push_back({v[1],
    -1});
99a                     if (cmp(v[1], v[0], c[i].c)) k++;
7d8                 }
b50             }
e05         }
d41
80e         P o = c[i].c;
381         sort(p.begin(), p.end(),[&o](auto a, auto b) {
328             return cmp(a.first,b.first,o);
c2e         });
d41
b80         for (int j = 0; j < p.size(); j++) {
bdb          P a = p[j? j-1: p.size()-1].first, // (j-1+p.size())
    %p.size()
ff1                  b = p[j].first;
3c9             double ang = (a - c[i].c).angle(b-c[i].c);
534             cover[k] += (a.x - b.x) * (a.y + b.y) / 2.0 +
    c[i].r * c[i].r * (ang - sin(ang)) / 2.0;
47c             k += p[j].second;
7ba         }
c22     }
64f     return cover;
6d9 }
```

line

```
d41 // Line
d41 //
d41 // Complexity: O(1) unless stated
d41
4fc template <class T>
72c struct Line {
b38     using P = Point<T>;
ebc     using L = Line;
d41
b79     P v; T c;
369     Line() {};
c30     Line(P p, T c) : v(v), c(c) {}
193     Line(P p, P q) : v(q - p), c(v ^ p) {}
d41
5c3     T side(P p) { return (v^p)-c; }
1a3     double dist(P p) { return sqrt(dist2(p)); }
8d1     T dist2(P p) { return side(p) * side(p) / v.norm2(); }
d41
98d     L perp(P p) { return L(p, p + v.perp()); }
d41
d41     // orthogonal projection and reflection
efc     P proj(P p) { return p - v.perp() * side(p) / v.norm2(); }
008     P refl(P p) { return p - v.perp() * 2 * side(p) /
    v.norm2(); }
d41
e24     bool parallel(L l) { return (v ^ l.v) == 0; }
d41
ce0     L translate(P t) { return L(v, c + (v ^ t)); }
d41
54c     friend bool inter(L l1, L l2, P &q) {
71b         if (l1.parallel(l2)) return false;
dd6         q = (l2.v*l1.c - l1.v*l2.c)/(l1.v^l2.v);
8a6         return true;
472     }
d41
db1     friend L bisector(L l1, L l2, bool interior) {
456         assert((l1.v^l2.v) != 0);
7e7         double sign = (interior? 1: -1);
b93         return L(l2.v / l2.v.norm() + l1.v / l1.v.norm()
    * sign,
80b                  l2.c / l2.v.norm() + l1.c / l1.v.norm()
    * sign);
304     }
494 };
d41
```

point

```
d41 // Point
d41 //
d41 // Complexity: O(1) unless stated
d41
67a template <typename T>
af1 int sign(T x) { return (x > 0) - (x < 0); }
d41
4fc template <class T>
f26 struct Point {
645     T x,y;
11b     using P = Point;
0fa     Point(T x=0, T y=0) : x(x), y(y) {}
d41
b3a     double norm() { return sqrt(norm2()); }
2c3     T norm2() { return (*this) * (*this); }
d41
286     P operator+(P p) { return P(x + p.x, y + p.y); }
f4b     P operator-(P p) { return P(x - p.x, y - p.y); }
a5b     P operator*(T t) { return P(x * t, y * t); }
da9     P operator/(T t) { return P(x / t, y / t); }
d41
301     T operator*(P p) { return x * p.x + y * p.y; }
a91     T operator^(P p) { return x * p.y - y * p.x; }
d41
163     bool operator<(P p) { return tie(x,y) < tie(p.x,p.y); }
eda     bool operator>(P p) { return tie(x,y) > tie(p.x,p.y); }
cd3    bool operator==(P p) { return tie(x,y) == tie(p.x,p.y); }
854    bool operator!=(P p) { return tie(x,y) != tie(p.x,p.y); }
d41
d41     // rotations are counter-clockwise
785     P rot(P p) { return P((*this)^p, (*this)*p); }
db2    P rot(double ang) { return rot(P(sin(ang), cos(ang))); }
d41
889     double angle() { return atan2(y,x); }
d48     double angle(P p) {
0c2         return acos(max(-1.0, min(1.0, (*this)*p/
    (norm()*p.norm()))));
69b     }
d41
56d     P perp() { return P(-y,x); }
6eb     bool is_perp(P p) { return ((*this) * p) == 0; }
d41
356     bool side() { return (y > 0) or (y == 0 and x < 0); }
d41
0e9     friend T cross(P a, P b, P c) { return (b-a)^(c-a); }
cae     friend bool left(P a, P b, P c) { return cross(a,b,c)
    > 0; }
9b8     friend bool right(P a, P b, P c) { return cross(a,b,c)
    < 0; }
d41
902     friend ostream& operator<<(ostream& os, P p) {
d80         return os<<"("<<p.x<<", "<<p.y<<")";
9a9     }
6e1 };
d41
67a template <typename T>
823 void polarsort(vector<T> &v) {
238     sort(v.begin(), v.end(), [](T a, T b) {
c05         return make_tuple(a.side(), 0) < make_tuple(b.side(),
    a ^ b);
f41     });
d35 }
```

polygon

```
d41 // Polygon structure
d41 //
d41 // Complexity: O(1) unless stated
d41
4fc template <class T>
666 struct Polygon {
b38     using P = Point<T>;
d41
1a8     int n;
573     vector<P> p;
d73     Polygon() {}
121     Polygon(vector<P> q) : n(q.size()), p(q) {}
d41
218     int prev(int i) { return (i-1+n)%n; }
88f     int next(int i) { return (i+1)%n; }
d41
d41     // Positive represents counter-clockwise order. Negative represent clockwise order.
d41     // Complexity: O(n)
320     T orientation() {
a1d         T acum=0;
603         for (int i = 0; i < n; i++) {
733             acum += p[i] ^ p[next(i)];
6f0         }
a13         return acum;
763     }
d41
d41     // Complexity: O(n)
e04     double area() { return double(area2())/2; }
19e     T area2() { return abs(orientation()); }
d41
d41     // Return true if point q is inside the polygon. The polygon must be convex and in ccw order.
d41     // Complexity: O(lgn)
df7     bool contains(P q) {
e86         if (right(p[0],p[1],q) or left(p[0],p[n-1],q)) return false;
9b1         int l=1, r=n;
219         while (r-l > 1) {
ee4             int m = (l+r)/2;
9eb             if (!right(p[0],p[m],q)) l=m;
ef3             else r=m;
638         }
aaa         if (r == n) return (q-p[0]).norm2() <= (p[n-1]-p[0]).norm2()+EPS;
e10         return !right(p[l],p[r],q);
f59     }
d41
d41     // Returns true if point q is inside the polygon (does not require to be convex).
d41     // Complexity: O(n)
35c     bool rayCasting(P q, bool strict = true) {
3dc         function<bool(P,P)> above = [](P a, P p) {
d3e             return p.y >= a.y;
940         };
d41
d66         function<bool(P,P,P)> crossesRay = [&above](P a, P p, P q) {
eba             return (above(a,q) - above(a,p)) * cross(a,p,q) > 0;
2b7         };
d41
537         int numCrossing = 0;
603         for (int i = 0; i < n; i++) {
6b5             if (Segment<T>(p[i], p[next(i)]).has(q)) {
468                 return !strict;
78a             }
8c9             numCrossing += crossesRay(q, p[i], p[next(i)]);
496         }
6be         return (numCrossing&1);
83b     }
d41
d41     // Returns the squared diameter (uses rotating calipers).
d41     // Complexity: O(n)
d5f     T diameter() {
966         T ans=0;
1ed         for (int i = 0, j = 1; i < n; i++) {
465             while (((p[next(i)]-p[i]) ^ (p[next(j)]-p[j])) > 0) j = next(j);
dcb             ans = max(ans, (p[i]-p[j]).norm2());
37e         }
ba7         return ans;
db8     }
d41
d41     // Returns the index of an extreme point
d41     // Complexity: O(lgn)
23b     int extreme(const function<bool(P, P)> &cmp) {
9b0         auto isExtreme = [&](int i, bool& curDir) -> bool {
b36             curDir = cmp(p[next(i)], p[i]);
224             return !cmp(p[prev(i)], p[i]) && !curDir;
2b9         };
d41
1a0         bool lastDir, curDir;
c7c         if (isExtreme(0, lastDir)) return 0;
993         int l = 0, r = n;
ead         while (l + 1 < r) {
27e             int m = (l + r) >> 1;
b60             if (isExtreme(m, curDir)) return m;
ac7             bool relDir = cmp(p[m], p[l]);
729             if ((!lastDir && curDir) || (lastDir == curDir && relDir == curDir)) {
8a6                 l = m;
986                 lastDir = curDir;
3dd             } else r = m;
e12         }
792         return l;
34c     }
d41
d41     // Finds the indices of the two tangents to an external point q
d41     // Complexity: O(lgn)
349     pair<int,int> tangent(P q) {
7fc         auto leftTangent = [&](P r, P s) -> bool {
60a             return cross(q, r, s) > 0;
403         };
702         auto rightTangent = [&](P r, P s) -> bool {
e5d             return cross(q, r, s) < 0;
ded         };
f49         return {extreme(leftTangent), extreme(rightTangent)};
d35     }
d41
d41     // Rotates the polygon points such that
d41     // p[0] is the lowest leftmost point
d41     // Complexity: O(n)
df5     void normalize() {
51b         rotate(p.begin(), min_element(p.begin(), p.end()), p.end());
7c6     }
d41
d41     // Minkowsky sum
d41     // Complexity: O(n)
47c     Polygon operator+(Polygon& other) {
a98         vector<P> sum;
335         normalize();
c4f         other.normalize();
f2b         ll dir;
df9         for(int i = 0, j = 0; i < n || j < other.n; i += dir >= 0, j += dir <= 0) {
a62             sum.push_back(p[i % n] + other.p[j % other.n]);
de3             dir = (p[(i + 1) % n] - p[i % n])
37e                 ^ (other.p[(j + 1) % other.n] - other.p[j % other.n]);
090         }
28c         return Polygon(sum);
e7e     }
60a };
```

segment

```
d41 // Segment
d41 //
d41 // Complexity: O(1) unless stated
d41
4fc template <class T>
fb3 struct Segment {
b38     using P = Point<T>;
cf0     using S = Segment;
d41
2ff     P p,q;
9e4     Segment() {}
07f     Segment(P p, P q) : p(p), q(q) {}
d41
2b3     bool inDisk(P r) { return (p-r)*(q-r) <= 0; }
a04     bool has(P r) { return cross(p,q,r) == 0 and inDisk(r); }
d41
259     T side(P r) { return sign((r-p)^(q-p)); }
31f     double len() { return (q-p).norm(); }
882     T len2() { return (q-p).norm2(); }
d41
d41     // for Shamos-Hoey
---     #warning caution in the case p == s.q (ex. checking if a polygon is simple)
787     bool operator<(S s) {
347         if (p == s.p) return cross(p, q, s.q) > 0;
a30         if (p.x != q.x and (s.p.x == s.q.x or p.x < s.p.x))
e68             return cross(p, q, s.p) > 0;
d57         return cross(p, s.q, s.p) > 0;
6c3     }
```

```
d41
d41     // checks if two segments intersect
7fb     friend bool inter(S a, S b) {
            if (a.has(b.p) or a.has(b.q) or b.has(a.p) or
e1e     b.has(a.q)) {
8a6             return true;
2a6         }
cba         P out;
0d1         return properInter(a, b, out);
501     }
d41
bfd     friend bool properInter(S s1, S s2, P &out) {
3c1         P a = s1.p, b = s1.q, c = s2.p, d = s2.q;
d41
8cf         double oa = cross(c, d, a),
392                ob = cross(c, d, b),
d14                oc = cross(a, b, c),
fd6                od = cross(a, b, d);
d41
b70         if (oa*ob < 0 and oc*od < 0) {
947             out = (a*ob - b*oa) / (ob-oa);
8a6             return true;
787         }
d41
d1f         return false;
8ce     }
d41
a35     Line<T> getLine() { return Line(p,q); }
cd5 };
d41
```

shamos_hoey

```
d41 // Shamos-hoey algorithm
d41 //
d41 // Description: given a set of segments finds if any two
    segments intersect
d41 //
d41 // Complexity: O(nlgn)
d41
4cb using S = Segment<ll>;
d41
a21 enum {
83e     ADD,
f5b     REM
405 };
d41
779 bool shamos_hoey(vector<S> seg) {
9ff     vector<array<ll, 3> > ev;
071     for (int i = 0; i < seg.size(); i++) {
ea5         if(seg[i].q < seg[i].p) swap(seg[i].p, seg[i].q);
ee1         ev.push_back({seg[i].p.x, ADD, i});
9f3         ev.push_back({seg[i].q.x, REM, i});
2cf     }
49e     sort(ev.begin(), ev.end());
e27     set<S> s;
d9b     for (const auto &e : ev) {
dcb         S at = seg[e[2]];
d7d         if (e[1] == ADD) {
8c2             auto nxt = s.lower_bound(at);
f67             if((nxt != s.end() && inter(*nxt, at))
                    || (nxt != s.begin() && inter(*prev(nxt),
70b     at))) {
6a5                 return 1;
e3e             }
9be             s.insert(at);
851         } else {
                auto nxt = s.upper_bound(at), cur = nxt, prv =
eb2     --cur;
b75             if(nxt != s.end() && prv != s.begin()
6d0                 && inter(*nxt, *(prev(prv)))) return 1;
005             s.erase(cur);
fe5         }
d26     }
bb3     return 0;
174 }
```

graphs

2sat

```
d41 // 2-SAT algorithm
d41
d41 // Description:
d41 //      Decideds if a 2-SAT instance has a solution and
d41 //      finds a valid solution (if possible).
d41 //
d41 // Details:
d41 //      'sz' in the constructor and reset function represents
d41 //      the number of clausures.
d41 //
d41 // Complexity:
d41 //      solve() and reset(n) are O(n).
d41
417 struct TwoSAT {
97b     int n, c, t;
61a     vector<vector<int>> adj, adjr;
b57     vector<int> topo, vis, comp, value;
d41
89b     TwoSAT(int sz) : n(2*sz), adj(n), adjr(n), topo(n),
    vis(n), comp(n), value(n) {
8bd         reset(sz);
083     }
d41
a64     void reset(int sz) {
686         n = 2*sz;
603         for(int i = 0; i < n; i++){
eab             topo[i] = vis[i] = comp[i] = value[i] = 0;
b1d             adj[i].clear();
7aa             adjr[i].clear();
245         }
a34         t = 0;
809         c = 1;
e96     }
d41
00c     int to_vertex(int i) {
9a6         if(i > 0) return (i - 1)<<1;
ccb         return to_vertex(-i) + 1;
955     }
d41
```

```
697     void add(int i, int j) { // add clause (i or j)
904         int vi = to_vertex(i);
d57         int vj = to_vertex(j);
2c9         adj[vi^1].push_back(vj);
5b0         adj[vj^1].push_back(vi);
d3b         adjr[vj].push_back(vi^1);
fb2         adjr[vi].push_back(vj^1);
4d9     }
d41
aa1     void add_true(int i) {  // add clause (i)
7b0         add(i, i);
a72     }
d41
6a7     void add_false(int i) { // add clause (neg i)
6cf         add(-i, -i);
a24     }
d41
21f     void add_xor(int i, int j) {    // add clause (i xor j)
50e         add(i, j);
3d2         add(-i, -j);
def     }
d41
af5     void add_xnor(int i, int j) { // add clause (i xnor j)
de7         add(i, -j);
dfb         add(-i, j);
ece     }
d41
839     void dfs1(int v) {
cca         vis[v] = 1;
5cf         for(auto x : adj[v]) if(vis[x] == 0) dfs1(x);
11f         topo[t++] = v;
413     }
d41
4b2     void dfs2(int v) {
d5f         comp[v] = c;
ee1         for(auto x : adjr[v]) if(comp[x] == 0) dfs2(x);
1d5     }
d41
6c0     bool solve() {
830         for(int i = 0; i < n; i++)
205             if(vis[i] == 0) dfs1(i);
45b         for(int i = n - 1; i >= 0; i--) {
b4b             if(comp[topo[i]] == 0) {
256                 dfs2(topo[i]);
6a4                 c++;
616             }
20f         }
e29         for(int i = 0; i < n; i+=2) {
5cb             if(comp[i] == comp[i^1])
d1f                 return false;
4e6             else {
df1                 value[i] = (comp[i] < comp[i^1]) ? 0 : 1;
bd8                 value[i^1] = 1 - value[i];
f18             }
e3e         }
8a6         return true;
edf     }
d41
9ca     bool get_value(int i) {
af7         return value[to_vertex(i)];
```

```
843     }
d41
30f     void print() {
e29         for(int i = 0; i < n; i += 2) {
b46             if(i != 0) cout << ' ';
2e8             cout << value[i];
222         }
1fb         cout << endl;
e99     }
926 };
```

block_cut

```
d41 // Block Cut Tree
d41 //
d41 // Descrition: builds the forest of bluck cut trees for an UNDIRECTED graph
d41 //
d41 // Complexity: O(N+M)
d41
142 struct BlockCutTree {
8d3     int ncomp; // number of components
f7a     vector<int> comp; // comp[e]: component of edge e
a1c     vector<vector<int>> gart; // gart[v]: list of components an articulation point v is adjacent to
d41                               // if v is NOT an articulation point, then gart[v] is empty
d41
d41     // assumes auto [neighbor_vertex, edge_id] = g[current_vertex][i]
deb     BlockCutTree(int n, int m, vector<pair<int,int>> g[]): ncomp(0), comp(m), gart(n) {
6bc         vector<bool> vis(n), vise(m);
594         vector<int> low(n), prof(n);
46e         stack<pair<int,int>> st;
d41
45f         function<void(int,bool)> dfs = [&](int v, bool root) {
cca             vis[v] = 1;
dc9             int arb = 0; // arborescences
e8a             for(auto [p, e]: g[v]) if(!vise[e]) {
c8a                 vise[e] = 1;
934                 int in = st.size();
20c                 st.emplace(e, vis[p] ? -1 : p);
137                 if(!vis[p]) {
f07                     arb++;
690                     low[p] = prof[p] = prof[v] + 1;
397                     dfs(p, 0);
de7                     low[v] = min(low[v], low[p]);
23d                 } else low[v] = min(low[v], prof[p]);
d41
c52                 if(low[p] >= prof[v]) {
c80                     gart[v].push_back(ncomp);
080                     while(st.size() > in) {
2b5                         auto [es, ps] = st.top();
8b3                         comp[es] = ncomp;
81d                         if(ps != -1 && !gart[ps].empty())
746                             gart[ps].push_back(ncomp);
25a                         st.pop();
229                     }
a8f                     ncomp++;
f0d                 }
863             }
7f8             if(root && arb <= 1) gart[v].clear();
5ee         };
d41
778         for(int v=0;v<n;v++) {
d0d             if(!vis[v]) dfs(v, 1);
1b3         }
6d6     }
c2c };
```

centroid

```
d41 // Centroid decomposition
d41
195 namespace centroid {
a2e     const int MAXN = 2e5+10;
d41
e81     vector<int> sz(MAXN, 1);
46f     vector<bool> removed(MAXN);
def     vector<vector<int>> adj(MAXN);
d41
765     int set_sizes(int u, int p = -1) {
267         sz[u] = 1;
087         for (int v: adj[u]) if (v != p and not removed[v]) {
8d0             sz[u] += set_sizes(v, u);
9c6         }
f93         return sz[u];
54d     }
d41
f47     int get_centroid(int u, int size) {
f8d         for (int v: adj[u]) if (sz[v] < sz[u]) {
869             if (sz[v] > size/2) return get_centroid(v, size);
7d9         }
03f         return u;
1ed     }
d41
d41     // For example
846     vector<int> dists(MAXN);
d41
8a5     int solve(int root, int k) {
f1e         int c = get_centroid(root, set_sizes(root));
6f4         removed[c] = true;
d41
d41         /**
fd0          * Answer in linear time here
f24          * Example: number of paths of length k
b44          **/
d41
11e         int res = 0;
ce6         queue<int> add, rem;
873         function<void(int, int, int)> dfs = [&](int u, int dep, int p) {
926             if (k >= dep) res += dists[k-dep];
4de             add.push(dep); rem.push(dep);
087             for (int v: adj[u]) if (v != p and not removed[v]) {
455                 dfs(v, dep+1, u);
166             }
f9f         };
d41
3e2         dists[0] = 1;
3c3         for (int v: adj[c]) if (not removed[v]) {
05d             dfs(v, 1, -1);
ce8             while (add.size()) {
021                 int d = add.front(); add.pop();
a17                 dists[d] += 1;
4bf             }
1e9         }
d41
51c         while (rem.size()) {
7ae             int d = rem.front(); rem.pop();
93b             dists[d] -= 1;
da7         }
d41
d41         /**
bfc          * Example end.
b44          **/
d41
3c3         for (int v: adj[c]) if (not removed[v]) {
2e4             res += solve(v, k);
d2b         }
b50         return res;
149     }
071 };
```

dinic

```
d41 // Dinitz algorithm for finding maximum flows
d41 //
d41 // Complexity: O(n^2m) (O(m sqrt(n)) on unit networks)
d41
67a template <typename T>
14d struct Dinic {
e9b     struct Edge {
791         int to;
d90         T cap, flow;
63d         Edge(int _to, T _cap) :
e02             to(_to), cap(_cap), flow(0) {}
ffe     };
d41
327     int src,snk;
6ff     vector<int> nxt, ptr, frst, dist, q;
321     vector<Edge> edges;
d41
7ab     Dinic(int n) :
02b         dist(n), frst(n,-1), q(n) {}
d41
4c7     void add(int u, int v, T cap){
2b3         edges.emplace_back(v,cap);
05a         nxt.push_back(frst[u]);
197         frst[u] = edges.size()-1;
d41
265         edges.emplace_back(u,0);
741         nxt.push_back(frst[v]);
67b         frst[v] = edges.size()-1;
6e7     }
d41
838     bool bfs() {
4f7         fill(dist.begin(), dist.end(), -1);
f3c         int st=0,ed=0;
```

```
af1         q[ed++] = src;
e13         dist[src] = 0;
d88         while(st<ed){
eff             int u = q[st++];
7f0             for(int e=frst[u]; e != -1; e = nxt[e]) {
9ac                 int v = edges[e].to;
1b4                 if(dist[v] == -1 and edges[e].cap > edges[e].flow){
8a0                     dist[v] = dist[u]+1;
a38                     q[ed++] = v;
1df                 }
6ff             }
c9d         }
27b         return dist[snk] != -1;
a67     }
d41
013     T dfs(int u, T f=INF){
9b2         if(u == snk or f == 0) return f;
dca         for(int &e=ptr[u]; e != -1; e=nxt[e]) {
9ac             int v = edges[e].to;
b5f             if(dist[u]+1 == dist[v] and edges[e].cap > edges[e].flow){
444                 int df = dfs(v, min(f,edges[e].cap-edges[e].flow));
f2d                 if(df>0){
4bc                     edges[e].flow += df;
1ef                     edges[e^1].flow -= df;
4ad                     return df;
e90                 }
eb8             }
1c4         }
bb3         return 0;
6ac     }
d41
dfd     T flow(int _src, int _snk) {
401         src=_src, snk=_snk;
8f6         T flow=0;
6d4         while(bfs()){
89d             ptr=frst;
ee5             while(T add_flow=dfs(src)) flow += add_flow;
b15         }
99d         return flow;
59a     }
971 };
```

ebcc

```
d41 // Tarjan algorithm for edge-biconnected components
d41 //
d41 // Description:
d41 //      Builds an array ebcc such that ebcc[u] = ebcc[v] iff
d41 //      u and v are in the same component.
d41 //
d41 //      Builds an array sz such that sz[ebcc[u]] is the size
d41 //      of u's component.
d41 //
d41 // Complexity: O(n+m)
d41
cc5 struct tarjan_ebcc {
0d8     int nebcc;
24e     vector<int> ebcc, sz;
d41
5f7     tarjan_ebcc(int n, vector<int> adj[]) : nebcc(0), ebcc(n) {
aa0         int t=0, sn=0;
9a9         vector<int> disc(n,-1), low(n,-1), stk(n);
d41
87f         function<void(int,int)> dfs = [&](int u, int p) {
782             disc[u] = low[u] = t++;
de4             stk[sn++] = u;
d1c             for(auto v: adj[u]) {
a30                 if(disc[v] == -1) {
95e                     dfs(v,u);
ab6                     low[u] = min(low[u], low[v]);
b83                 }
f9b                 else if(v != p) low[u] = min(low[u], disc[v]);
263             }
c23             if(low[u] == disc[u]) {
d93                 int v;
f41                 sz.emplace_back(0);
016                 do {
1ac                     v = stk[--sn];
5a5                     ebcc[v] = nebcc;
1d5                     sz[nebcc]++;
759                 } while(u != v);
e59                 nebcc++;
030             }
e07         };
d41
603         for(int i=0; i<n; i++) {
f2f             if(disc[i] == -1) dfs(i,i);
527         }
daf     }
d41
a68     vector<vector<int>> condensed_graph(int n, vector<int> adj[]) {
246         vector<vector<int>> adj_ebcc(nebcc);
687         for(int u=0; u<n; u++) {
372             for(int v: adj[u]) {
3c9                 if(ebcc[v] != ebcc[u]) adj_ebcc[ebcc[u]].push_back(ebcc[v]);
b05             }
665         }
ddb         return adj_ebcc;
874     }
5eb };
```

hld_vertex

```
d41 // Heavy-light decomposition for vertices
d41 //
d41 // Description:
d41 //      Allows range queries and updates on trees. Template uses
d41 //      lazy segment tree, keep in mind that if structure is changed
d41 //      then the complexities change.
d41 //
d41 // Functions:
d41 //      query_path(a,b):     sum of elements in the path a -> b
d41 //      update_path(a,b,x):  sums x to elements in path a -> b
d41 //      query_subtree(a):    sum of elements in the subtree of a
d41 //     update_subtree(a,x): sums x to elements in the subtree of a
d41 //
d41 // Complexity:
d41 //      query_path:     O(lg^2 n)
d41 //      update_path:    O(lg^2 n)
d41 //      query_subtree:  O(lgn)
d41 //      update_subtree: O(lgn)
d41
833 const int MAXN = 6e5;
d41
826 namespace hld {
885     segtree<long long> seg;
094     vector<int> h(MAXN), v(MAXN), p(MAXN), pos(MAXN), sz(MAXN), peso(MAXN);
d41
baf     void build(int root, int n, vector<int> adj[]) {
6bb         int t=0;
d41
5ae         function<void(int,bool)> dfs = [&](int u, bool f) {
f1a             v[pos[u]=t++] = peso[u];
267             sz[u] = 1;
cab             for(auto &v: adj[u]) {
567                 if(v == p[u]) continue;
e42                 p[v] = u;
e60                 h[v] = (v == adj[u][0]? h[u]: v);
1ae                 dfs(v,f);
cc3                 sz[u] += sz[v];
880                 if(sz[v] > sz[adj[u][0]] or adj[u][0] == p[v]) {
d67                     swap(v, adj[u][0]);
bbf                 }
155             }
94c             if(u == root and f) dfs(h[u]=u,t=false);
f60         };
d41
09a         dfs(root,true);
0da         seg = segtree<long long>(t,v);
e94     }
d41
4f6     long long query_path(int a, int b) {
aa1         if(pos[a] < pos[b]) swap(a,b);
a42         if(h[a] == h[b]) return seg.query(pos[b],pos[a]);
c74         return seg.query(pos[h[a]], pos[a]) + query_path(p[h[a]], b);
567     }
d41
920     void update_path(int a, int b, int x) {
aa1         if(pos[a] < pos[b]) swap(a,b);
ebf         if(h[a] == h[b]) return seg.update(pos[b],pos[a],x);
370         seg.update(pos[h[a]], pos[a], x);
7c6         update_path(p[h[a]], b, x);
1db     }
d41
```

```
665     long long query_subtree(int a) {
77d         return seg.query(pos[a], pos[a]+sz[a]-1);
7f8     }
d41
acc     void update_subtree(int a, int x) {
6b7         seg.update(pos[a], pos[a]+sz[a]-1, x);
002     }
827 };
```

kuhn

```
d41 // Kuhn algorithm
d41 //
d41 // Description: computes a maximum matching in a bipartite graph
d41 //
d41 // Complexity: O(VE)
d41 //
d41 // Details: if not using add, the adjacency of only one side is needed
d41
d41
878 mt19937 rng((int) chrono::steady_clock::now().time_since_epoch().count());
d41
e12 struct Kuhn {
14e     int n,m;
903     vector<vector<int>> adj;
d23     vector<int> mu, mv;
d41
0f2     Kuhn(int n, int m) :
fb4         n(n), m(m), adj(n), mu(n,-1), mv(m,-1) {}
d41
0a9     void add(int u, int v) {
cc9         adj[u].push_back(v);
451     }
d41
bf7     int matching() {
6a9         vector<bool> vis(n+m, false);
d41
6a4         function<bool(int)> dfs = [&](int u) {
b9c             vis[u] = true;
f07             for(int v: adj[u]) if(!vis[n+v]) {
a46                 vis[n+v] = true;
ea2                 if(mv[v] == -1 or dfs(mv[v])) {
12d                     mu[u] = v, mv[v] = u;
8a6                     return true;
32a                 }
b31             }
d1f             return false;
0a2         };
d41
1ae         int ret=0, aum=1;
92b         for(auto& u: adj) shuffle(u.begin(), u.end(), rng);
392         while(aum) {
0fa             fill(vis.begin()+n, vis.end(), false);
c5d             aum=0;
687             for(int u=0; u<n; u++) {
5da                 if(mu[u] == -1 and dfs(u)) ret++, aum=1;
5e9             }
891         }
edf         return ret;
140     }
2f5 };
```

lca-rmq

```
d41 // Lowest common ancestor using RMQ
d41 //
d41 // Description: finds the LCA in O(1) using RMQ
d41
74b namespace LCA {
adc     vector<int> dep, pos, inv;
e16     RMQ<pair<int,int>> rmq;
d41
7a2     void build(int root, const vector<vector<int>>& adj) {
19f         dep.resize(2 * adj.size());
940         inv.resize(2 * adj.size());
be5         pos.resize(adj.size());
d41
6bb         int t = 0;
0ff         function<void(int, int, int)> dfs = [&](int u, int d, int p) {
a86             inv[t] = u, pos[u] = t, dep[t++] = d;
914             for (int v: adj[u]) if (v != p) {
1fe                 dfs(v, d+1, u);
678                 inv[t] = u, dep[t++] = d;
043             }
18e         };
d08         dfs(root, 0, -1);
904         rmq = RMQ(dep);
d41
8a4         vector<pair<int,int>> vals(t);
960         for (int i = 0; i < t; i++) {
f29             vals[i] = {dep[i], i};
7ba         }
d9d         rmq = RMQ(vals);
a67     }
d41
310     int lca(int u, int v) {
bdc         if (pos[u] > pos[v]) swap(u, v);
e91         return inv[rmq.query(pos[u], pos[v])];
f02     }
d41
c11     int dist(int u, int v) {
bb1        return dep[pos[u]] + dep[pos[v]] - 2 * dep[pos[lca(u, v)]];
10d     }
340 }
```

scc

```
d41 // Tarjan algorithm for strongly connected components
d41 //
d41 // Description:
d41 //      Builds an array scc such that scc[u] = scc[v]
d41 //      iff u and v are in the same component. And build an
d41 //      array sz such that sz[scc[u]] is the size of u's component.
d41 //
d41 // Complexity: O(n+m)
d41
24b struct tarjan_scc {
f43     int nscc;
35f     vector<int> scc, sz;
d41
11b     tarjan_scc(int n, vector<int> adj[]) : nscc(0), scc(n) {
aa0         int t=0, sn=0;
9a9         vector<int> disc(n,-1), low(n,-1), stk(n);
d41
214         function<void(int)> dfs = [&](int u) {
782             disc[u] = low[u] = t++;
de4             stk[sn++] = u;
d1c             for(auto v: adj[u]) {
0b4                 if(disc[v] == -1) dfs(v);
ab6                 low[u] = min(low[u], low[v]);
8a2             }
c23             if(low[u] == disc[u]) {
d93                 int v;
f41                 sz.emplace_back(0);
016                 do {
1ac                     v = stk[--sn];
2fb                     scc[v] = nscc;
2a5                     sz[nscc]++;
b62                     low[v] = INF;
db0                 } while(u != v);
983                 nscc++;
3cd             }
47b         };
d41
603         for(int i=0; i<n; i++) {
162             if(disc[i] == -1) dfs(i);
5cd         }
9fc     }
d41
d41     // Complexity: O(n+m)
a68     vector<vector<int>> condensed_graph(int n, vector<int> adj[]) {
2a3         vector<vector<int>> adj_scc(nscc);
687         for(int u=0; u<n; u++) {
372             for(int v: adj[u]) {
caa                 if(scc[v] != scc[u]) adj_scc[scc[u]].push_back(scc[v]);
6e8             }
cc7         }
6f4         return adj_scc;
f3d     }
ecd };
```

math

bit_iterator

```
d41 // Bit iterator
d41 // iterator through all masks with n bits and m set bits
d41 // use: for(auto it: BitIterator(n,m) { int mask = *it; ... }
d41
368 struct BitIterator {
41c     struct Mask {
f79         uint32_t msk;
```

```
5f7         Mask(uint32_t _msk): msk(_msk) {}
22e         bool operator!=(const Mask& rhs) const { return msk < rhs.msk; };
29f             void operator++(){const uint32_t t=msk|(msk-1);msk=(t+1)|(((~t&-~t)-1)>>__builtin_ffs(msk));}
600         uint32_t operator*() const { return msk; }
cc7     };
1dc     uint32_t n, m;
75a     BitIterator(uint32_t _n, uint32_t _m): n(_n), m(_m) {}
17a     Mask begin() const { return Mask((1<<m)-1); }
d0d     Mask end() const { return Mask((1<<n)); }
8ca };
```

convolutions

```
d41 // Convolutions
d41 // Description: Multiply two polinomial
d41 // Complexity: O(N logN)
d41
d41 // Functions:
d41 //      multiply(a, b)
d41 //      multiply_mod(a, b, m) - return answer modulo m
d41
d41 // Details:
d41 //      For function multiply_mod, any modulo can be used.
d41 //      It is implemented using the technique of dividing
d41 //      in sqrt to use less fft. Function multiply may have
d41 //      precision problems.
d41 //      This code is faster than normal. So you may use it
d41 //      if TL e tight.
d41
d32 const double PI=acos(-1.0);
35b namespace fft {
3b2     struct num {
662         double x,y;
c0a         num() {x = y = 0;}
6da         num(double x,double y): x(x), y(y){}
cd4     };
4d4     inline num operator+(num a, num b) {return num(a.x + b.x, a.y + b.y);}
f7b     inline num operator-(num a, num b) {return num(a.x - b.x, a.y - b.y);}
b7b     inline num operator*(num a, num b) {
9f0         return num(a.x * b.x - a.y * b.y, a.x * b.y + a.y * b.x);
d63     }
db0     inline num conj(num a) {return num(a.x, -a.y);}
d41
b58     int base = 1;
e47     vector<num> roots={{0,0}, {1,0}};
8a4     vector<ll> rev={0, 1};
148     const double PI=acosl(-1.0);
d41
d41     // always try to increase the base
d50     void ensure_base(int nbase) {
11e         if(nbase <= base) return;
49f         rev.resize(1 << nbase);
55a         for (int i = 0; i < (1 << nbase); i++)
19e             rev[i] = (rev[i>>1] >> 1) + ((i&1) << (nbase-1));
2b8         roots.resize(1<<nbase);
775         while(base<nbase) {
21f             double angle = 2*PI / (1<<(base+1));
8cf             for(int i = 1<<(base-1); i < (1<<base); i++) {
52a                 roots[i<<1] = roots[i];
aef                 double angle_i = angle * (2*i+1-(1<<base));
922                 roots[(i<<1)+1] = num(cos(angle_i),sin(angle_i));
958             }
96d             base++;
af4         }
ae9     }
d41
b94     void fft(vector<num> &a,int n=-1) {
05e         if(n==-1) n=a.size();
421         assert((n&(n-1)) == 0);
2fd         int zeros = __builtin_ctz(n);
a02         ensure_base(zeros);
4fa         int shift = base - zeros;
603         for (int i = 0; i < n; i++) {
3fc             if(i < (rev[i] >> shift)) {
b8b                 swap(a[i],a[rev[i] >> shift]);
9ac             }
b97         }
7cd         for(int k = 1; k < n; k <<= 1) {
cda             for(int i = 0; i < n; i += 2*k) {
0c2                 for(int j = 0; j < k; j++) {
d85                     num z = a[i+j+k] * roots[j+k];
20a                     a[i+j+k] = a[i+j] - z;
c9a                     a[i+j] = a[i+j] + z;
ee1                 }
804             }
b62         }
382     }
d41
ba5     vector<num> fa, fb;
d41     // multiply with less fft by using complex numbers.
318     vector<ll> multiply(vector<ll> &a, vector<ll> &b);
d41
d41     // using the technique of dividing in sqrt to use less fft.
966     vector<ll> multiply_mod(vector<ll> &a, vector<ll> &b, ll m, ll eq=0);
754     vector<ll> square_mod(vector<ll>&a, ll m);
7a3 };
d41
d41 // 16be45
7b3 vector<ll> fft::multiply(vector<ll> &a, vector<ll> &b) {
fe9     int need = a.size() + b.size() - 1;
d41
217     int nbase = 0;
8da     while((1 << nbase) < need) nbase++;
4e5     ensure_base(nbase);
d41
729     int sz = 1 << nbase;
9db     if(sz > (int)fa.size()) fa.resize(sz);
887     for(int i = 0; i < sz; i++) {
422         ll x = (i < (int)a.size() ? a[i] : 0);
435         ll y = (i < (int)b.size() ? b[i] : 0);
685         fa[i] = num(x, y);
3e3     }
d41
650     fft(fa, sz);
4db     num r(0,-0.25/sz);
3ec     for(int i = 0; i <= (sz>>1); i++) {
b13         int j = (sz-i) & (sz-1);
afc         num z = (fa[j] * fa[j] - conj(fa[i] * fa[i])) * r;
f07         if(i != j) fa[j] = (fa[i] * fa[i] - conj(fa[j] * fa[j])) * r;
386         fa[i] = z;
488     }
d41
650     fft(fa, sz);
5e0     vector<ll> res(need);
07f     for(int i = 0; i < need; i++) res[i] = fa[i].x + 0.5;
b50     return res;
16b }
d41
d41 // 4eb347
d99 vector<ll> fft::multiply_mod(vector<ll> &a, vector<ll> &b, ll m, ll eq) {
fe9     int need = a.size() + b.size() - 1;
217     int nbase = 0;
8da     while((1 << nbase) < need) nbase++;
4e5     ensure_base(nbase);
729     int sz = 1 << nbase;
9db     if(sz > (int)fa.size()) fa.resize(sz);
c0e     for(int i = 0; i < (int)a.size(); i++) {
538         ll x = (a[i] % m + m) % m;
7e5         fa[i] = num(x & ((1 << 15) - 1), x >> 15);
b60     }
26e     fill(fa.begin() + a.size(), fa.begin() + sz, num{0,0});
650     fft(fa, sz);
32a     if(sz > (int)fb.size()) fb.resize(sz);
b19     if(eq) copy(fa.begin(), fa.begin() + sz, fb.begin());
4e6     else {
1da         for(int i = 0; i < (int)b.size(); i++) {
044             ll x = (b[i] % m + m) % m;
418             fb[i] = num(x & ((1 << 15) - 1), x >> 15);
9f0         }
535         fill(fb.begin() + b.size(), fb.begin() + sz, num{0,0});
07e         fft(fb,sz);
59c     }
df3     double ratio = 0.25 / sz;
dc2     num r2(0, -1), r3(ratio, 0), r4(0, -ratio), r5(0,1);
3ec     for(int i = 0; i <= (sz>>1); i++) {
b13         int j = (sz - i) & (sz - 1);
d96         num a1 = (fa[i] + conj(fa[j]));
6c3         num a2 = (fa[i] - conj(fa[j])) * r2;
712         num b1 = (fb[i] + conj(fb[j])) * r3;
e45         num b2 = (fb[i] - conj(fb[j])) * r4;
41e         if(i != j) {
123             num c1 = (fa[j] + conj(fa[i]));
7ce             num c2 = (fa[j] - conj(fa[i])) * r2;
92b             num d1 = (fb[j] + conj(fb[i])) * r3;
a76             num d2 = (fb[j] - conj(fb[i])) * r4;
35f             fa[i] = c1 * d1 + c2 * d2 * r5;
525             fb[i] = c1 * d2 + c2 * d1;
55a         }
dc5         fa[j] = a1 * b1 + a2 * b2 * r5;
```

```
d92         fb[j] = a1 * b2 + a2 * b1;
dc3     }
f68     fft(fa, sz); fft(fb, sz);
5e0     vector<ll> res(need);
ae6     for(int i = 0; i < need; i++) {
6e0         ll aa = fa[i].x + 0.5;
bb7         ll bb = fb[i].x + 0.5;
0ee         ll cc = fa[i].y + 0.5;
407         res[i] = (aa + ((bb%m) << 15) + ((cc%m) << 30))%m;
0d6     }
b50     return res;
ca4 }
d41
b86 vector<ll> fft::square_mod(vector<ll> &a, ll m) {
dfb     return multiply_mod(a, a, m, 1);
dde }
```

crt

```
d41 // Chinese Remainder Theorem
d41 // Joins modular linear equations: x = a (mod m)
d41 // if there are no solutions, a = -1
d41
153 template<typename T> tuple<T, T, T> ext_gcd(T a, T b) {
3bd     if (!a) return {b, 0, 1};
550     auto [g, x, y] = ext_gcd(b%a, a);
c59     return {g, y - b/a*x, x};
537 }
d41
bfe template<typename T = ll> struct crt {
627     T a, m;
d41
5f3     crt() : a(0), m(1) {}
7eb     crt(T a_, T m_) : a(a_), m(m_) {}
911     crt operator * (crt C) {
238         auto [g, x, y] = ext_gcd(m, C.m);
dc0         if ((a - C.a) % g) a = -1;
4f9         if (a == -1 or C.a == -1) return crt(-1, 0);
d09         T lcm = m/g*C.m;
eb2         T ans = a + (x*(C.a-a)/g % (C.m/g))*m;
d8d         return crt((ans % lcm + lcm) % lcm, lcm);
1f2     }
0d9 };
```

division_trick

```
d41 // Division trick
d41 //
d41 // Complexity: O(sqrt(n))
d41
79c for(int l = 1, r; l <= n; l = r + 1) {
746     r = n / (n / l);
d41     // n / i has the same value for l <= i <= r
5bf }
```

ext_gcd

```
d41 // Extended GCD
d41 //
d41 // Descrption:
d41 //      Let g = gcd(a,b)
d41 //      Returns {g,x,y} such that ax+by = g
d41 //
d41 // Complexity: O(log(min(a,b)))
d41
6f9 tuple<int,int,int> ext_gcd(int a, int b) {
3bd     if (!a) return {b, 0, 1};
550     auto [g, x, y] = ext_gcd(b%a, a);
c59     return {g, y - b/a*x, x};
596 }
```

fwht

```
d41 // Fast Walsh-Hadamard trasform
d41 // Description: Multiply two polynomials such that x^a * x^b
    = x^(op(a, b))
d41 //  - op(a, b) = a "xor" b, a "or" b, a "and" b
d41 // Complexity: O(n log n)
d41
29a const ll N = 1<<20;
d41
67a template <typename T>
372 struct FWHT {
a69     void fwht(T io[], ll n) {
495         for (ll d = 1; d < n; d <<= 1) {
b98             for (ll i = 0, m = d<<1; i < n; i += m) {
499                 for (ll j = 0; j < d; j++) { /// Don't forget
    modulo if required
bdc                     T x = io[i+j], y = io[i+j+d];
703                     io[i+j] = (x+y), io[i+j+d] = (x-y); // xor
d41                     // io[i+j] = x+y; // and
d41                     // io[i+j+d] = x+y; // or
afa                 }
7f3             }
cf6         }
fe6     }
1f8     void ufwht(T io[], ll n) {
495         for (ll d = 1; d < n; d <<= 1) {
b98             for (ll i = 0, m = d<<1; i < n; i += m) {
499                 for (ll j = 0; j < d; j++) { /// Don't forget
    modulo if required
bdc                     T x = io[i+j], y = io[i+j+d];
d41                     /// Modular inverse if required here
174                     io[i+j] = (x+y)>>1, io[i+j+d] = (x-
    y)>>1; // xor
d41                     // io[i+j] = x-y; // and
d41                     // io[i+j+d] = y-x; // or
027                 }
711             }
fd4         }
e69     }
d41     // a, b are two polynomials and n is size which is power
    of two
683     void convolution(T a[], T b[], ll n) {
26b         fwht(a, n), fwht(b, n);
e95         for (ll i = 0; i < n; i++)
33e             a[i] = a[i]*b[i];
23c         ufwht(a, n);
5b4     }
d41     // for a*a
f9a     void self_convolution(T a[], ll n) {
d85         fwht(a, n);
e95         for (ll i = 0; i < n; i++)
35c             a[i] = a[i]*a[i];
23c         ufwht(a, n);
4c8     }
ba3 };
d0d FWHT<ll> fwht;
```

integration

```
d41 // Numerical Integration
d41 //
d41 // Error grows O((b - a)^5)
d41
676 const int N = 3*100; // multiple of 3
287 ld integrate(ld a, ld b, function<ld(ld)> f) {
b4d     ld s = 0, h = (b - a)/N;
067     for (int i = 1 ; i < N; i++) s += f(a + i*h)*(i%3 ?
    3 : 2);
0da     return (f(a) + s + f(b))*3*h/8;
c7e }
```

matrix

```
d41 // Matrix structure
d41 //
d41 // Complexity:
d41 //      operator*: O(n^3)
d41 //      operator^: O(n^3 lgn)
d41
67a template<typename T>
953 struct Matrix : vector<vector<T>> {
14e     int n,m;
36b             Matrix(const   vector<vector<T>>&   c)   :
    vector<vector<T>>(c), n(c.size()), m(c[0].size()) {}
cf8     Matrix(int n, int m) : vector<vector<T>>(n,vector<T>(m,
    0)),n(n),m(m){}
bd0     Matrix(const initializer_list<initializer_list<T>>& c) {
f7e         vector<vector<T>> val;
212         for(auto& i:c) val.push_back(i);
5a8         *this = Matrix(val);
ad3     }
254     Matrix operator*(Matrix& r) {
1e2         assert(m == r.n);
0f7         Matrix M(n, r.m);
830         for(int i=0;i<n;i++)
800             for(int k=0;k<m;k++)
418                 for(int j=0;j<r.m;j++)
47e                     M[i][j] += (*this)[i][k]*r[k][j];
474         return M;
658     }
7cf     Matrix operator^(ll e){
0d6         Matrix M(n,n);
40a         for(int i=0;i<n;i++) M[i][i] = 1;
13c         for(Matrix at=*this;e;e/=2,at=at*at)
2e2             if(e&1) M = M*at;
474         return M;
9ee     }
b3e };
```

mint

```
d41 // Modular integer structure
d41
9a3 template <const int MOD>
e54 struct mint {
3ec     int x;
9f5     mint(): x(0) {}
609     mint(int _x): x(_x%MOD<0?_x%MOD+MOD:_x%MOD) {}
5b8     void operator+=(mint rhs) { x+=rhs.x; if(x>=MOD) x-
=MOD; }
a9a     void operator-=(mint rhs) { x-=rhs.x; if(x<0)x+=MOD; }
d08     void operator*=(mint rhs) { x*=rhs.x; x%=MOD; }
152     void operator/=(mint rhs) { *this *= rhs.inv(); }
9a2     mint operator+(mint rhs) { mint res=*this; res+=rhs;
return res; }
ee4     mint operator-(mint rhs) { mint res=*this; res-=rhs;
return res; }
384     mint operator*(mint rhs) { mint res=*this; res*=rhs;
return res; }
dd6     mint operator/(mint rhs) { mint res=*this; res/=rhs;
return res; }
7ea     mint inv() { return this->pow(MOD-2); }
714     mint pow(int e) {
30b         mint res(1);
65a         for(mint p=*this;e>0;e/=2,p*=p) if(e%2)
bbc             res*=p;
b50         return res;
f35     }
b64 };
d41
7e5 using Z = mint<998244353>;
```

xor_basis

```
d41 // XOR basis
d41 //
d41 // Description:
d41 //      builds a basis in \Integers_2 that spans the
d41 //      vectors inserted.
d41 //
d41 //      think of integers as vectors in \Integers_2,
d41 //      i.e., 4 -> (1,0,0).
d41 //
d41 // Complexity: O(SZ), unless stated otherwise
d41
722 template <const int SZ>
b5d struct XorBasis {
727     int base[SZ];
d41
79f     XorBasis() { memset(base, 0, sizeof base); }
d41
1f5     bool bit_on(int x, int i) {
2ab         return (x&(1LL<<i));
627     }
d41
1b3     void insert(int v) {
da0         for (int i = SZ-1; i >= 0; i--) {
0b3             if (bit_on(v, i)) {
7e8                 if (!base[i]) {
e02                     base[i] ^= v;
505                     return;
7ed                 }
0d7                 v ^= base[i];
1e7             }
9b5         }
736     }
d41
d41     // merges two basis in O(SZ^2)
92f     void merge(XorBasis &b) {
da0         for (int i = SZ-1; i >= 0; i--) {
f59             if (b.base[i]) insert(b.base[i]);
019         }
b87     }
d41
d41     // checks if v can be achieved
3e7     bool check(int v) {
c67         int x = 0;
592         for (int i = SZ-1; i >= 0; i--) if (base[i]) {
5aa             if (bit_on(x, i) != bit_on(v, i)) {
25a                 x ^= base[i];
2af             }
fe8         }
57d         return (x == v);
8d8     }
d41
d41     // returns the maximum value
e92     int max() {
404         int value = 0;
781         for (int i = SZ-1; i >= 0; i--) if (!bit_on(value,
i)) {
cca             value ^= base[i];
214         }
af2         return value;
e92     }
2b4 };
```

strings

aho

```
d41 // Aho-Corasick
d41 //
d41 // Description: data structure that allows search
d41 // for multiple patterns in text.
d41 //
d41 // Details: K is the alphabet size, in template
d41 // it is using alphabet from 'a' to 'z' with offset 'a'
d41
f12 template <const int K = 26>
d3c struct aho_corasick {
3c9     struct node {
714         int next[K], len,
78f             link, // suffix link
a92          term_link; // suffix link to "first" terminal state
403         bool is_term; // terminal state (i.e., a word from
dictionary ends here)
cf9         node() : link(-1), term_link(-1), is_term(false) {
237             fill(next, next+K, -1);
e2c         }
a46     };
d41
bbb     vector<node> t;
d41
968     aho_corasick() : t(1) {}
cbd     aho_corasick(int size) : t(1) { t.reserve(size); }
d41
498     int new_node() {
83d         t.emplace_back();
b18         return t.size()-1;
1ae     }
d41
d41     // O(|s|)
d45     void add_string(const string& s) {
400         int u = 0;
aec         for (const char& ch: s) {
15f             int c = ch - 'a';
a1c           if (t[u].next[c] == -1) t[u].next[c] = new_node();
2c2             u = t[u].next[c];
01d         }
d34         t[u].is_term = true, t[u].len = s.size();
842     }
d41
d41     // O(|dictionary size|)
0a8     void build() {
0bb         queue<int> q; q.push(0);
402         while (q.size()) {
be1             int u = q.front(); q.pop();
176             for (int c = 0; c < K; c++) if (t[u].next[c] !=
-1) {
0d6                 int v = t[u].next[c], l = t[u].link;
ceb                 while (l != -1 and t[l].next[c] == -1) {
a96                     l = t[l].link;
35b                 }
786                 t[v].link = (l == -1? 0: t[l].next[c]);
f42                 t[v].term_link = (t[t[v].link].is_term?
t[v].link: t[t[v].link].term_link);
2a1                 q.push(v);
a60             }
ff7         }
227     }
d41
d41     // O(|dictionary size|)
c3b     void add_dictionary(const vector<string>& dict) {
4bb         for (const string& s: dict) add_string(s);
6f2         build();
dd5     }
d41
d41     /**
61c      * Example: CSES Word Combinations.
2fa      * dp[i] is the # of ways to generate string [1...i]
6a0      * (where s[1...0] = empty string)
b44      **/
b2f     mint query(const string& s) {
b49         int at = 0, i = 0;
0d7         vector<mint> dp(s.size()+1); dp[0] = 1;
aec         for (const char& ch: s) {
15f             int c = ch - 'a';
d41
3e4             if (t[at].term) dp[i] += dp[i-t[at].len];
```

```
759             for (int e = t[at].term_link; e != -1; e =
t[e].term_link) {
d70                 dp[i] += dp[i-t[e].len];
eee             }
d41
385             while (at != -1 and t[at].next[c] == -1) {
054                 at = t[at].link;
409             }
2bd             at = (at == -1? 0: t[at].next[c]);
0dd             i++;
db2         }
d41
3e4         if (t[at].term) dp[i] += dp[i-t[at].len];
759         for (int e = t[at].term_link; e != -1; e =
t[e].term_link) {
d70             dp[i] += dp[i-t[e].len];
eee         }
d41
648         return dp[s.size()];
9a3     }
c42 };
```

hash

```
d41 //  String Hashing
d41 //  Description: A data structure that transforms a string
into a number
d41
d41 //  Functions:
d41 //      str_hash - Builds the hash in O(|S|)
d41 //     operator() - Gives the number representing substring
s[l,r] in O(1)
d41
d41 //  Details:
d41 //          - To use more than one prime, you may use long
long, __int128 or array<int>
d41 //          - You may easily change it to handle vector<int>
instead of string
d41 //          - Other large primes: 1000041323, 100663319,
201326611
d41 //          - If smaller primes are needed(For instance,
need to store the mods in an array):
d41 //           - 50331653, 12582917, 6291469, 3145739, 1572869
d41 //
d41
4ba const long long mod1 = 1000015553, mod2 = 1000028537;
d41
878 mt19937 rng((int) chrono::steady_clock::now().time_since_epoch()).co
random number generator
d41
463 int uniform(int l, int r) {
a7f     uniform_int_distribution<int> uid(l, r);
f54     return uid(rng);
d9e }
d41
3fb template<int MOD>
d7d struct str_hash {
c63     static int P;
dcf     vector<ll> h, p;
0e1     str_hash () {}
ea8     str_hash(string s) : h(s.size()), p(s.size()) {
7a2         p[0] = 1, h[0] = s[0];
ad7         for (int i = 1; i < s.size(); i++)
84c             p[i] = p[i - 1]*P%MOD, h[i] = (h[i - 1]*P +
s[i])%MOD;
1ef     }
af7     ll operator()(int l, int r) { // retorna hash s[l...r]
749         ll hash = h[r] - (l ? h[l - 1]*p[r - l + 1]%MOD : 0);
dfd         return hash < 0 ? hash + MOD : hash;
3ba     }
977 };
217 template<int MOD> int str_hash<MOD>::P = uniform(256, MOD -
1); // l > |sigma|
d41
61c struct Hash {
d41     // Uses 2 primes to better avoid colisions
3b6     str_hash<mod1> H1;
b36     str_hash<mod2> H2;
d41
e3d     Hash (string s) : H1(str_hash<mod1>(s)),
H2(str_hash<mod2>(s)) {}
d41
af7     ll operator()(int l, int r) {
f6f         ll ret1 = H1(l, r), ret2 = H2(l, r);
742         return (ret1 << 30) ^ (ret2);
d2e     }
b31 };
```

kmp

```
d41 // Knuth-Morris-Pratt algorithm (KMP)
d41 //
d41 // Description:
d41 //      The prefix function, p[i] is the longest suffix that
ends at i
d41 //      that is also a suffix.
d41 //
d41 // Complexity: O(n)
d41
67a template<typename T>
29c vector<int> kmp(int sz, const T s[]) {
8e9     vector<int> p(sz);
e8d     for(int i=1;i<sz;i++) {
42d         int &j = p[i];
084         for(j=p[i-1];j>0 && s[i]!=s[j];j=p[j-1]);
04b         if(s[i] == s[j]) j++;
b13     }
74e     return p;
8af }
d41
f42 vector<vector<int>> automaton(string s) {
46e     s += '#';
c73     int n = (int)s.size();
cf1     auto p = kmp(s.size(), s.data());
c3d     vector<vector<int>> aut(n,vector<int>(26,0));
603     for(int i=0;i<n;i++) {
1e1         for(int c=0;c<26;c++) {
42c             if(i>0 && 'a'+c != s[i]) aut[i][c] = aut[p[i-1]]
[c];
325             else aut[i][c] = i + ('a' + c == s[i]);
237         }
40c     }
8a7     return aut;
593 }
```

manacher

```
d41 // Manacher algorithm
d41 //
d41 // Description:
d41 //      Finds a vector with palindromes size.
d41 //      ret[2*i]   = size of largest palindrome with center
in i
d41 //      ret[2*i+1] = size of largest palindrome with center
in i and i+1
d41 //
d41 // Complexity: O(n)
d41
67a template <typename T>
44d vector<int> manacher(const T& s) {
18f     int l = 0, r = -1, n = s.size();
fc9     vector<int> d1(n), d2(n);
603     for (int i = 0; i < n; i++) {
821         int k = i > r ? 1 : min(d1[l+r-i], r-i);
61a         while (i+k < n && i-k >= 0 && s[i+k] == s[i-k]) k++;
61e         d1[i] = k--;
9f6         if (i+k > r) l = i-k, r = i+k;
950     }
e03     l = 0, r = -1;
603     for (int i = 0; i < n; i++) {
a64         int k = i > r ? 0 : min(d2[l+r-i+1], r-i+1); k++;
2c6         while (i+k <= n && i-k >= 0 && s[i+k-1] == s[i-k])
k++;
eaa         d2[i] = --k;
26d         if (i+k-1 > r) l = i-k, r = i+k-1;
4fe     }
c41     vector<int> ret(2*n-1);
e6b     for (int i = 0; i < n; i++) ret[2*i] = 2*d1[i]-1;
e1d     for (int i = 0; i < n-1; i++) ret[2*i+1] = 2*d2[i+1];
edf     return ret;
9ca }
```

minimum_rotation

```
d41 // Lexicographically smallest rotation of a string
d41 // O(n)
d41
cc8 int minimum_rotation(string s) {
37f     int a = 0, n = s.size(); s += s;
a0e     for (int b = 0; b < n; b++) for (int k = 0; k < n; k+
+) {
20d         if (a+k == b or s[a+k] < s[b+k]) { b += max<int>(0,
k-1); break; }
068         if (s[a+k] > s[b+k]) { a = b; break; }
5c3     }
3f5     return a;
aa9 }
```

palindromic_tree

```
d41 // Palindromic tree (eertree)
d41 //
d41 // Complexity: O(|s|) amortized
d41
c71 struct palindromic_tree {
3c9     struct node {
ff6         int length, link;
3a4         map<char, int> to;
697       node(int length, int link): length(length), link(link) {}
45d     };
b9e     vector<node> nodes;
9fc     int current;
d36     palindromic_tree(): current(1) {
31a         nodes.push_back(node(-1, 0));
5ec         nodes.push_back(node(0, 0));
2fa     }
ea5     void add(int i, string& s) {
8f5         int parent = nodes[current].length == i ? nodes[current].link : current;
f2b         while (s[i - nodes[parent].length - 1] != s[i])
c9c             parent = nodes[parent].link;
4d9         if(nodes[parent].to.find(s[i]) != nodes[parent].to.end()) {
b6c             current = nodes[parent].to[s[i]];
9a4         }
4e6         else {
490             int link = nodes[parent].link;
0bb             while (s[i - nodes[link].length - 1] != s[i])
304                 link = nodes[link].link;
569             link = max(1, nodes[link].to[s[i]]);
b39             current = nodes[parent].to[s[i]] = nodes.size();
bcf             nodes.push_back(node(nodes[parent].length + 2, link));
739         }
ec3     }
bfc     void insert(string& s) {
dd9         current = 1;
9a8         for (int i = 0; i < int(s.size()); i++)
df9             add(i, s);
c50     }
0d1 };
d41
```

suffix_array

```
d41 // Suffix array
d41 //
d41 // Description:
d41 //      Builds a sorted array of all suffixes of a string.
d41 //
d41 // Complexity:
d41 //      build: O(nlgn)
d41
3f4 struct SuffixArray {
19b     vector<int> v, inv, lcp;
d41
67a     template <typename T>
292     SuffixArray(int sz, const T s[]): v(sz+1), inv(sz+1), lcp(sz) {
5b8         iota(v.begin()+1, v.end(), 0);
d5f         sort(v.begin()+1, v.end(), [&](int i, int j) { return s[i] < s[j]; });
b8a         v[0] = sz++;
311         vector<int> ra(sz), newra(sz), newv(sz), cnt(sz+1);
941         ra[v[1]] = 1;
248         for(int i=2;i<sz;i++) ra[v[i]] = ra[v[i-1]] + (s[v[i]] != s[v[i-1]]);
22f         for(int k=1;k<sz;k*=2) {
0ca             for(int i=0;i<sz;i++) v[i] = (v[i]-k+sz)%sz;
35b             fill(cnt.begin(), cnt.end(), 0);
1b1             for(int x: ra) cnt[x+1]++;
6db             partial_sum(cnt.begin(), cnt.end(), cnt.begin());
eb3             for(int i=0;i<sz;i++) newv[cnt[ra[v[i]]]++] = v[i];
2de             v.swap(newv);
e8d             for(int i=1;i<sz;i++) {
3f9                 int diff = ra[v[i]] != ra[v[i-1]] || ra[(v[i]+k)%sz] != ra[(v[i-1]+k)%sz];
0ec                 newra[v[i]] = newra[v[i-1]] + diff;
729             }
694             ra.swap(newra);
db2         }
f28         for(int i=0;i<sz;i++) inv[v[i]] = i;
f7a         for(int l=0,i=0;i<sz-1;i++) {
e3e             int j = v[inv[i]-1];
040             while(max(i,j)+l<sz-1 && s[i+l] == s[j+l]) l++;
f58             lcp[inv[i]-1] = l;
89d             if(l) l--;
a89         }
afc     }
901 };
```

suffix_automaton

```
d41 // Suffix Automaton
d41 //
d41 // Description:
d41 //      Builds a minimal DFA that accepts all suffixes
d41 //      of a string
d41 //
d41 // Complexity:
d41 //      build: O(n)
d41
92e struct SuffixAutomaton {
bf2     struct Node {
a46         int len, link;
3a4         map<char, int> to;
ad3         Node() : len(0), link(-1) {}
a4d     };
d41
0f1     int n, sz, last;
f5b     vector<Node> t;
d41
244     void add(char c) {
37a         int cur = sz++;
24a         t[cur].len = t[last].len + 1;
81e         int p = last;
8d7         while (p != -1 && !t[p].to.count(c)) {
ad1             t[p].to[c] = cur;
e06             p = t[p].link;
2a9         }
924         if (p == -1) {
b64             t[cur].link = 0;
c44         } else {
a7d             int q = t[p].to[c];
9a9             if (t[p].len + 1 == t[q].len) {
cdd                 t[cur].link = q;
913             } else {
90a                 int clone = sz++;
477                 t[clone].len = t[p].len + 1;
46d                 t[clone].to = t[q].to;
b6f                 t[clone].link = t[q].link;
520                 while (p != -1 && t[p].to[c] == q) {
b1e                     t[p].to[c] = clone;
e06                     p = t[p].link;
b85                 }
762                 t[q].link = t[cur].link = clone;
914             }
a9c         }
691         last = cur;
607     }
d41
8ff     SuffixAutomaton(string &s) : n(s.size()), sz(1), last(0), t(2*n) {
217         for (auto c: s) add(c);
2f3     };
99b };
```

trie

```
d41 // Trie data structure
d41 //
d41 // Description:
d41 //      Data structure for strings.
d41 //
d41 // Complexity:
d41 //      add: O(|s|)
d41 //      erase: O(|s|)
d41 //      find: O(|s|)
d41
ab5 struct trie {
1b9     int sigma, offset;
5e5     vector<int> wrd, cnt;
e1a     vector<vector<int>> to;
d41
498     int new_node() {
a2e         wrd.push_back(0);
a02         cnt.push_back(0);
526         to.push_back(vector<int>(sigma));
501         return to.size()-1;
a02     }
d41
30f     trie(int _sigma=26, int _offset='a') :
bc6         sigma(_sigma), offset(_offset) {
839         new_node();
819     }
d41
d41     // adds s to the trie
22d     void add(string &s) {
```

```
181         int i = 0;
b4f         for(char c: s) {
2a6             int &nxt = to[i][c-offset];
29d             if(!nxt) nxt = new_node();
b70             cnt[i=nxt]++;
f14         }
c8f         wrd[i]++;
333     }
d41
d41     // if s is in the trie removes it and returns true,
d41     // else returns false
5c9     bool erase(string &s) {
3d0         if(!find(s)) return false;
181         int i=0;
b4f         for(char c: s) {
2a6             int &nxt = to[i][c-offset];
d25             cnt[i=nxt]--;
4d9             if(!cnt[nxt]) nxt=0;
678         }
8cc         wrd[i]--;
8a6         return true;
65e     }
d41
d41     // if s is in the trie returns true,
d41     // else returns false
9b8     bool find(string &s) {
181         int i=0;
b4f         for(char c: s) {
2a6             int &nxt = to[i][c-offset];
59c             if(!nxt) return false;
799             i = nxt;
ff6         }
243         return (wrd[i] > 0);
bbd     }
789 };
```

z

```
d41 // Z-function
d41 //
d41 // Description:
d41 //   z[i]: is the longest suffix that starts in i that is also
d41 //    a suffix that starts at 0.
d41 //
d41 // Complexity: O(n)
d41
67a template<typename T>
7f4 vector<int> z_function(int n, const T s[]) {
2e8     vector<int> z(n);
2c3     int x = 0, y = 0;
6f5     for (int i = 1; i < n; i++) {
063         z[i] = max(0,min(z[i-x],y-i+1));
0d8         while (i+z[i] < n && s[z[i]] == s[i+z[i]]) {
654             x = i; y = i+z[i]; z[i]++;
54c         }
a64     }
070     return z;
944 }
```

structures

bit

```
d41 // BIT (Fenwick tree)
d41 //
d41 // Functions:
d41 //      query(l,r):  sum of elements in range [l,r]
d41 //      update(i,x): sums x to value in position i
d41 //
d41 // Details: query and update are 1-indexed.
d41 //
d41 // Complexity:
d41 //      build:  O(n)
d41 //      query:  O(lgn)
d41 //      update: O(lgn)
d41
67a template <typename T>
714 struct BIT{
1a8     int n;
678     vector<T> bit;
d41
03f     BIT(int n) : n(n+1), bit(n+1){}
6c8     BIT(vector<int> &v) : n(ssize(v)+1), bit(ssize(v)+1) {
465         for(int i=1;i<n;i++) bit[i] = v[i-1];
6f5         for(int i=1;i<n;i++) {
edf             int j = i+(i&-i);
2b8             if(j<n) bit[j] += bit[i];
604         }
64b     }
d41
b13     T pref_query(int i) {
501         T sum = 0;
185         for(;i;i-=i&-i) sum+=bit[i];
e66         return sum;
bea     }
d41
b7a     T query(int l, int r) {
6b8         return pref_query(r)-pref_query(l-1);
1d4     }
d41
2ac     void update(int i, T x) {
4a3         for(;i<n;i+=i&-i) bit[i]+=x;
0e8     }
099 };
```

lazy_seg

```
d41 // Segment tree with lazy propagation
d41 //
d41 // Functions:
d41 //      update(l,r,x): sums x to all elements in range [l,r]
d41 //      query(l,r): sum of all elements in range [l,r]
d41 //
d41 // Complexity:
d41 //      build: O(n)
d41 //      update: O(lgn)
d41 //      query: O(lgn)
d41
d41 // neutral element
cad const int NEUT = 0;
d41
67a template <typename T>
```

```
07c struct segtree {
1a8     int n;
243     vector<T> seg, lazy;
d41
ff0     T build(int p, int l, int r, vector<T> &v) {
3c7         lazy[p] = 0;
6cd         if(l == r) return seg[p] = v[l];
ee4         int m = (l+r)/2;
e2a       return seg[p] = build(2*p,l,m,v)+build(2*p+1,m+1,r,v);
06d     }
d41
731     segtree(vector<T> &v) : n(v.size()), seg(4*n), lazy(4*n)
{ build(1,0,n-1,v); }
9a8     segtree(int n) : n(n), seg(4*n), lazy(4*n) {}
d41
ceb     void prop(int p, int l, int r) {
cdf         seg[p] += lazy[p]*(r-l+1);
2c9         if(l != r) lazy[2*p] += lazy[p], lazy[2*p+1] +=
lazy[p];
cf4         lazy[p] = NEUT;
7fd     }
d41
69e     T update(int a, int b, int x, int p, int l, int r) {
6b9         prop(p,l,r);
e9f         if(b < l or r < a) return seg[p];
9a3         if(a <= l and r <= b) {
b94             lazy[p] += x;
6b9             prop(p,l,r);
534             return seg[p];
821         }
ee4         int m=(l+r)/2;
6f7         return seg[p] = update(a,b,x,2*p,l,m)+update(a,b,x,
2*p+1,m+1,r);
9d4     }
d41
88b     T query(int a, int b, int p, int l, int r){
6b9         prop(p,l,r);
527         if(a <= l and r <= b) return seg[p];
d60         if(b < l or r < a) return NEUT;
ee4         int m = (l+r)/2;
b1f         return query(a,b,2*p,l,m)+query(a,b,2*p+1,m+1,r);
98b     }
d41
b6d     void update(int l, int r, int x) { update(l,r,x,
1,0,n-1); }
190     T query(int l, int r) { return query(l,r,1,0,n-1); }
a37 };
d41
```

median

```
d41 // Median
d41
2b0 struct Median {
d7b     multiset<ll, greater<ll>> sm;
33d     multiset<ll> gt;
689     int size = 0;
d41
6a2     void balance() {
731         if (sm.size() < (size + 1) / 2) {
```

```
6c5             ll x = *(gt.begin());
9ad             gt.erase(gt.begin());
2da             sm.insert(x);
e12         } else if (sm.size() > (size + 1) / 2) {
e79             ll x = *(sm.begin());
440             sm.erase(sm.begin());
27d             gt.insert(x);
b89         }
b39     }
d41
e1b     void insert(ll v) {
8ab         if (size == 0 || v <= median_first()) sm.insert(v);
005         else gt.insert(v);
b56         size++;
da6         balance();
ef3     }
d41
32d     bool remove(ll v) {
b47         if (size == 0) return false;
4b9         if (v <= median_first()) {
8f2             auto it = sm.find(v);
5e2             if (it == sm.end()) return false;
570             sm.erase(it);
5cc         } else {
ba5             auto it = gt.find(v);
5dc             if (it == gt.end()) return false;
59d             gt.erase(it);
e93         }
1ec         size--;
da6         balance();
8a6         return true;
579     }
d41
fcd     ll median_first() {
4fb         if (size % 2 == 1) return *(sm.begin());
e77         return *(sm.begin());
187     }
d41
23d     double median() {
4fb         if (size % 2 == 1) return *(sm.begin());
a29         return (*(sm.begin()) + *(gt.begin())) / 2.0;
9ab     }
d41
393     string median_string() {
bdc         if (size % 2 == 1) return to_string(*(sm.begin()));
bf7         ll a = *(sm.begin()) + *(gt.begin());
ac0         string s;
9c6         if (a < 0) s += '-';
5e9         s += to_string(abs(a) / 2);
013         if (abs(a) % 2 == 1) s += ".5";
047         return s;
2f1     }
55f };
```

mo

```
d41 // MOs algorithm
d41 //
d41 // Description:
d41 //      Answers queries offline with sqrt decomposition
d41 //
d41 // Complexity:
d41 //      exec: O(nsqrt(n)(O(remove)+O(add)))
d41
678 const int SZ = 230;
d41
670 struct Query {
738     int l, r, idx;
9a6     Query () {}
e7d     Query (int _l, int _r, int _idx) : l(_l), r(_r), idx(_idx) {}
9ae     bool operator < (const Query &o) const {
04d         return {l / SZ, r} < {o.l / SZ, o.r};
4a3     }
d0f };
d41
67a template <typename T>
5ce struct MO {
8d9     int sum;
b61     MO(vector<T> &v) : sum(0), v(v), cnt(MAXN), C(MAXN) {}
d41
f5d     void exec(vector<Query> &queries, vector<T> &answers) {
14d         answers.resize(queries.size());
bfa         sort(queries.begin(), queries.end());
d41
3df         int cur_l = 0;
cf5         int cur_r = -1;
d41
275         for (Query q : queries) {
71e             while (cur_l > q.l) {
ec6                 cur_l--;
939                 add(cur_l);
60c             }
294             while (cur_r < q.r) {
bda                 cur_r++;
d95                 add(cur_r);
c3b             }
b32             while (cur_l < q.l) {
631                 remove(cur_l);
cf9                 cur_l++;
ddf             }
6eb             while (cur_r > q.r) {
198                 remove(cur_r);
99e                 cur_r--;
d76             }
553             answers[q.idx] = get_answer(cur_l, cur_r);
8bc         }
c0e     }
d41
c96     void add(int i) {
683         sum += v[i];
0c3     }
d41
17e     void remove(int i) {
f2f         sum -= v[i];
9a0     }
d41
83a     T get_answer(int l, int r) {
e66         return sum;
42d     }
d84 };
```

ordered_set

```
d41 // Ordered set (0-indexed)
d41 //
d41 // Description:
d41 //      It has the same properties and functions of the regular set and
d41 //      two extra functions.
d41 //
d41 // Functions:
d41 //      find_by_order(k) - iterator to the element with index k
d41 //      order_of_key(k) - index of the element k in the set
d41 //
d41 // Complexity:
d41 //      find_by_order: O(logn)
d41 //      order_of_key: O(logn)
d41
774 #include <ext/pb_ds/assoc_container.hpp>
30f #include <ext/pb_ds/tree_policy.hpp>
0d7 using namespace __gnu_pbds;
4fc template <class T>
def using ord_set = tree<T, null_type, less<T>, rb_tree_tag,
3a1     tree_order_statistics_node_update>;
```

persistent_seg

```
d41 // Persistent Segment Tree
d41 //
d41 // Functions:
d41 //      update(p,x,t):  updates the value at position p to x of version t
d41 //      update(p,x):    updates the value at position p to x of the last version
d41 //      query(l,r,t):   sum of all elements in range [l,r] at version t
d41 //
d41 // Complexity:
d41 //      build:  O(n)
d41 //      update: O(lgn)
d41 //      query:  O(lgn)
d41
67a template <typename T>
e22 struct pseg{
86f     const int LIM = 1e7;
87c     int n, rt, nxt;
130     vector<T> seg;
8b1     vector<int> L, R;
d41
af3     pseg(int n): seg(LIM), L(LIM), R(LIM), n(n), nxt(0) {}
d41
188     int new_node(T x, int l=0, int r=0) {
2f6         int nn = nxt++;
21d         seg[nn] = x; L[nn] = l; R[nn] = r;
7c3         return nn;
605     }
d41
```

```
7e7     int build(vector<T> &v, int l, int r) {
ed7         if(l==r) return new_node(v[l]);
565         int m=(l+r)/2, nl=build(v,l,m), nr=build(v,m+1,r);
e0f         return new_node(seg[nl]+seg[nr],nl,nr);
ccb     }
d41
291     int build(vector<T> &v) { return rt = build(v,0,n-1); }
d41
a15     int update(int p, T x, int k, int l, int r) {
4ca         int nn = new_node(seg[k],L[k],R[k]);
e88         if(l == r){seg[nn]=x;return nn;}
ee4         int m = (l+r)/2;
bd6         if(p <= m) L[nn] = update(p,x,L[nn],l,m);
c4e         else R[nn] = update(p,x,R[nn],m+1,r);
fff         seg[nn] = seg[L[nn]] + seg[R[nn]];
7c3         return nn;
0a1     }
d41
d99     T query(int a, int b, int k, int l, int r) {
a2c         if(r < a or b < l) return 0;
f49         if(a <= l and r <= b) return seg[k];
ee4         int m = (l+r)/2;
84e         return query(a,b,L[k],l,m) + query(a,b,R[k],m+1,r);
a22     }
d41
ab2     int update(int p, T x, int t) { return rt=update(p,x,t,
0,n-1); }
2b0     int update(int p, T x) { return update(p,x,rt); }
c72     T query(int l, int r, int t) { return query(l,r,t,
0,n-1); }
1ad };
```

seg_iterative

```
d41 // Segment tree
d41
4fc template <class T>
9cc struct segment_tree {
374     T NEUT = T();
d41
1a8     int n;
130     vector<T> seg;
d41
d41     // O(n)
409     segment_tree(vector<T> &v) : n(v.size()), seg(2*n) {
1a1         for (int i = 0; i < n; i++) seg[i+n] = v[i];
d15         for (int i = n-1; i; i--) seg[i] = seg[2*i] +
 seg[2*i+1];
d13     }
d41
d41     // O(lgn)
6a3     void update(int pos, T val) {
fa3         for (seg[pos += n] = val; pos >>= 1;)
385             seg[pos] = seg[2*pos] + seg[2*pos+1];
1d1     }
d41
d41     // O(lgn)
b7a     T query(int l, int r) {
66d         T tl = NEUT, tr = NEUT;
ae9         for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
15d             if (l&1) tl = tl + seg[l++];
a6c             if (r&1) tr = tr + seg[--r];
da0         }
cf9         return tl + tr;
452     }
cc9 };
```

segtree

```
d41 // Segment tree
d41 //
d41 // Functions:
d41 //      update(p,x): updates the value at position p to x
d41 //      query(l,r):  the sum of all elements in range [l,r]
d41 //
d41 // Complexity:
d41 //      build:   O(n)
d41 //      update:  O(lgn)
d41 //      query:   O(lgn)
d41
67a template <typename T>
07c struct segtree {
1a8     int n;
130     vector<T> seg;
d41
ff0     T build(int p, int l, int r, vector<T> &v) {
6cd         if(l == r) return seg[p] = v[l];
ee4         int m = (l+r)/2;
e2a       return seg[p] = build(2*p,l,m,v)+build(2*p+1,m+1,r,v);
94d     }
d41
f87     segtree(vector<T> &v) : n(v.size()), seg(4*n) {
7ed         build(1,0,n-1,v);
811     }
4f2     segtree(int n) : n(n), seg(4*n) {}
d41
6c6     T update(int a, T x, int p, int l, int r) {
fa6         if(a < l or r < a) return seg[p];
b0e         if(l == r) return seg[p] = x;
ee4         int m=(l+r)/2;
9bd         return seg[p] = update(a,x,2*p,l,m)+update(a,x,
2*p+1,m+1,r);
394     }
d41
88b     T query(int a, int b, int p, int l, int r){
527         if(a <= l and r <= b) return seg[p];
786         if(b < l or r < a) return 0;
ee4         int m = (l+r)/2;
b1f         return query(a,b,2*p,l,m)+query(a,b,2*p+1,m+1,r);
85d     }
d41
7d4     void update(int p, T x) { update(p,x,1,0,n-1); }
190     T query(int l, int r) { return query(l,r,1,0,n-1); }
2e5 };
```

slope_trick

```
d41 // SlopeTrick
d41 //
d41 // Armazena uma estrutura convexa piecewise linear
d41 // Permite adicionar slopes sem peso e realizar query de
    minimo
d41 // Comentarios acima das funcoes para explicar o que cada
    uma faz
d41
67a template<typename T>
406 struct SlopeTrick {
64e     T inf = numeric_limits<T>::max() / 3;
acc     T min_f;
f32     priority_queue<T, vector<T>, less<>> L;
6ef     priority_queue<T, vector<T>, greater<>> R;
a20     T add_l, add_r;
d41
055     T top_R() {
a34         if (R.empty()) return inf;
ffe         else return R.top() + add_r;
074     }
d41
70c     T pop_R() {
66f         T val = top_R();
8e0         if (R.size()) R.pop();
d94         return val;
21d     }
d41
d9d     T top_L() {
b7b         if (L.empty()) return -inf;
470         else return L.top() + add_l;
31d     }
d41
821     T pop_L() {
66a         T val = top_L();
1e0         if (L.size()) L.pop();
d94         return val;
dfd     }
d41
86d     size_t size() {
7ff         return L.size() + R.size();
c4b     }
d41
0e8     SlopeTrick() : min_f(0), add_l(0), add_r(0) {};
d41
d41     // return {min f(x), lx, rx}
d41     // Em que [lx, rx] eh o intervalo que atinge o minimo
5ee     array<T, 3> query() {
e8a         return {min_f, top_L(), top_R()};
14f     }
d41
d41     // f(x) += a
ad4     void add_all(T a) {
f8c         min_f += a;
78a     }
d41
d41     // add \_
d41     // f(x) += max(a - x, 0)
60a     void add_a_minus_x(T a) {
8c6         min_f += max(T(0), a - top_R());
cdb         R.push(a - add_r);
416         L.push(pop_R() - add_l);
44c     }
d41
```

```
d41     // add _/
d41     // f(x) += max(x - a, 0)
7a9     void add_x_minus_a(T a) {
b36         min_f += max(T(0), top_L() - a);
988         L.push(a - add_l);
e5a         R.push(pop_L() - add_r);
f3a     }
d41
d41     // add \/
d41     // f(x) += abs(x - a)
825     void add_abs(T a) {
9cc         add_a_minus_x(a);
e55         add_x_minus_a(a);
639     }
d41
d41     // \/ -> \_
d41     // f_{new} (x) = min f(y) (y <= x)
73b     void clear_right() {
b8e         while (R.size()) R.pop();
2b3     }
d41
d41     // \/ -> _/
d41     // f_{new} (x) = min f(y) (y >= x)
fd5     void clear_left() {
e21         while (L.size()) L.pop();
bc4     }
d41
d41     // \/ -> \_/
d41     // f_{new} (x) = min f(y) (x-b <= y <= x-a)
564     void shift(T a, T b) {
25b         assert(a <= b);
b95         add_l += a;
165         add_r += b;
29a     }
d41
d41     // \/. -> .\/
d41     // f_{new} (x) = f(x - a)
5d6     void shift(T a) {
a77         shift(a, a);
af1     }
d41
d41     // Retorna f(x)
d41     // O(size)
c2a     T get(T x) {
7ce         auto L2 = L;
202         auto R2 = R;
bf4         T ret = min_f;
6a9         while (L.size()) {
efd             ret += max(T(0), pop_L() - x);
e50         }
886         while (R.size()) {
97b             ret += max(T(0), x - pop_R());
8ef         }
98a         L = L2, R = R2;
edf         return ret;
093     }
d41
d41     // O(min(size, st.size))
9e9     void merge(SlopeTrick &st) {
f68         if (st.size() > size()) {
079             swap(*this, st);
788         }
1a3         while (st.R.size()) {
85b             add_x_minus_a(st.pop_R());
2c5         }
8c6         while (st.L.size()) {
897             add_a_minus_x(st.pop_L());
b31         }
eaf         min_f += st.min_f;
3df     }
f24 };
```

sparse_seg

```
d41 // Sparse segment tree
d41 //
d41 // Functions:
d41 //      query(l,r):     sum in range [l,r]
d41 //      update(l,r,x):  sums x to all elements in [l,r]
d41 //
d41 // Complexity:
d41 //      build:  O(1)
d41 //      query:  O(lgn)
d41 //      update: O(lgn)
d41
67a template <typename T>
75a struct sseg{
86f     const int LIM = 1e7;
e42     const int N = 1e9;
d41
2a7     int nxt;
243     vector<T> seg, lazy;
8b1     vector<int> L,R;
d41
9a3     sseg() : seg(LIM), lazy(LIM), L(LIM), R(LIM), nxt(2) {}
d41
e9a     int get_l(int i) {
ba7         if(!L[i]) L[i] = nxt++;
a96         return L[i];
a5a     }
d41
943     int get_r(int i) {
2e9         if(!R[i]) R[i] = nxt++;
283         return R[i];
cb5     }
d41
ceb     void prop(int p, int l, int r) {
cdf         seg[p] += lazy[p]*(r-l+1);
821         if(l!=r) lazy[get_l(p)]+=lazy[p], lazy[get_r(p)]
+=lazy[p];
3c7         lazy[p] = 0;
cae     }
d41
183     T update(int a, int b, T x, int p, int l, int r) {
6b9         prop(p,l,r);
9a3         if(a <= l and r <= b){
b94             lazy[p] += x;
6b9             prop(p,l,r);
534             return seg[p];
821         }
e9f         if(b < l or r < a) return seg[p];
ee4         int m = (l+r)/2;
b66         return seg[p] = update(a,b,x,get_l(p),l,m) +
update(a,b,x,get_r(p),m+1,r);
610     }
d41
88b     T query(int a, int b, int p, int l, int r) {
6b9         prop(p,l,r);
527         if(a <= l and r <= b) return seg[p];
786         if(b < l or r < a) return 0;
ee4         int m = (l+r)/2;
818         return query(a,b,get_l(p),l,m)
+query(a,b,get_r(p),m+1,r);
ca4     }
d41
6b7     void update(int l, int r, T x){update(l,r,x,1,0,N-1);};
fab     T query(int l, int r){return query(l,r,1,0,N-1);};
f87 };
```

sparse_table

```
d41 // Sparse table
d41 //
d41 // Functions:
d41 //      query(l,r): minimum element in range [l,r]
d41 //
d41 // Complexity:
d41 //      build: O(nlgn)
d41 //      query: O(1)
d41
67a template <typename T>
991 struct sparse {
1a8     int n;
5a6     vector<vector<T>> m;
d41
5c0     sparse(vector<T> &v) : n(v.size()) {
020         int lg = 32 - __builtin_clz(n);
60a         m.assign(lg,vector<T>(n));
78e         for(int i=0;i<n;i++) m[0][i] = v[i];
05c         for(int j=1;(1<<j)<=n;j++) {
ed9             for(int i=0;i+(1<<j)<=n;i++) {
5d5             m[j][i] = min(m[j-1][i], m[j-1][i+(1<<(j-1))]);
642             }
b3a         }
7d1     }
d41
b7a     T query(int l, int r) {
133         int j = 31 - __builtin_clz(r-l+1);
56f         return min(m[j][l], m[j][r-(1<<j)+1]);
7b9     }
87a };
```