MWNWMWNNWMWNWN
USP

Nathan Luiz, Willian Mori e Willian Wang

8 de março de 2023

Índice

1 strings

1.1 Err Tree - Palindromic Tree

```
//  Description: A tree such that each node represents a
//               palindrome of string s. It is possible to append
//               a character.
//  Complexity: Amortized O(|s|)
//

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]];
9d9         } 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));
6f7         }
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 };
```

1.2 String Hashing

```
//  Functions:
//      str_hash - Builds the hash in O(|S|)
//      operator() - Gives the number representing substring s[l,r] in O(1)

//  Details:
//          - To use more than one prime, you may use long long, __int128 or array<int>
//          - You may easily change it to handle vector<int> instead of string
//          - Other large primes: 1000041323, 100663319, 201326611
//          - If smaller primes are needed(For instance, need to store the mods in an array):
//              - 50331653, 12582917, 6291469, 3145739, 1572869
//

4ba const long long mod1 = 1000015553, mod2 = 1000028537;

878 mt19937 rng((int) chrono::steady_clock::now().time_since_epoch().count()); // random number generator

463 int uniform(int l, int r) {
a7f     uniform_int_distribution<int> uid(l, r);
f54     return uid(rng);
d9e }

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|

61c struct Hash {
        // Uses 2 primes to better avoid colisions
3b6     str_hash<mod1> H1;
b36     str_hash<mod2> H2;

e3d     Hash (string s) : H1(str_hash<mod1>(s)), H2(str_hash<mod2>(s)) {}

af7     ll operator()(int l, int r) {
f6f         ll ret1 = H1(l, r), ret2 = H2(l, r);
742         return (ret1 << 30) ^ (ret2);
d2e     }
b31 };
```

1.3 KMP

```
// mathcing(s, t) retorna os indices das ocorrencias
// de s em t
// autKMP constroi o automato do KMP

// Complexidades:
// pi - O(n)
// match - O(n + m)
// construir o automato - O(|sigma|*n)
// n = |padrao| e m = |texto|

0a1 template <typename T> vector<int> kmp(int sz, const T s[]) {
924     vector<int> pi(sz);
e8d     for(int i=1;i<sz;i++) {
730         int &j = pi[i];
6ef         for(j=pi[i-1];j>0 && s[i]!=s[j];j=pi[j-1]);
04b         if(s[i] == s[j]) j++;
4fb     }
81d     return pi;
b29 };

c10 template<typename T> vector<int> matching(T& s, T& t) {
658     vector<int> p = pi(s), match;
a1b     for (int i = 0, j = 0; i < t.size(); i++) {
6be         while (j and s[j] != t[i]) j = p[j-1];
c4d         if (s[j] == t[i]) j++;
310         if (j == s.size()) match.push_back(i-j+1), j = p[j-1];
028     }
ed8     return match;
c82 }

a2d struct KMPaut : vector<vector<int>> {
47c     KMPaut(){}
6c7     KMPaut (string& s) : vector<vector<int>>(26,
    vector<int>(s.size()+1)) {
503         vector<int> p = pi(s);
04b         auto& aut = *this;
4fa         aut[s[0]-'a'][0] = 1;
19a         for (char c = 0; c < 26; c++)
5d3             for (int i = 1; i <= s.size(); i++)
42b                 aut[c][i] = s[i]-'a' == c ? i+1 : aut[c][p[i-1]];
4bb     }
79b };
```

1.4 Suffix Array

```
// Description: Algorithm that sorts the suffixes of a string
// Complexity: O(|s| log(|s|))
//

// Suffix Array da KTH
3f4 struct SuffixArray {
ac0     string s;
716     vector<int> sa, lcp;
264     SuffixArray () {}
cb4     SuffixArray(vector<string>& v, int lim=256) { // or
    basic_string<int>
318         for(auto str : v) {
cf2             s += str;
fc8             s += '$';
ee6         }
861         int n = s.size(), k = 0, a, b;
99c         vector<int> x(all(s)+1), y(n), ws(max(n, lim)), rank(n);
8a6         sa = lcp = y; iota(all(sa), 0);
25d         for (int j = 0, p = 0; p < n; j = max(1, j * 2), lim = p) {
e59             p = j; iota(all(y), n - j);
3fc             for(int i = 0; i < n; i++) if (sa[i] >= j) y[p++] =
    sa[i] - j;
911             fill(all(ws), 0);
483             for(int i = 0; i < n; i++) ws[x[i]]++;
5d9             for(int i = 1; i < lim; i++) ws[i] += ws[i - 1];
a9e             for (int i = n; i--;) sa[--ws[x[y[i]]]] = y[i];
3ff             swap(x, y); p = 1; x[sa[0]] = 0;
b7e             for(int i = 1; i < n; i++) a = sa[i - 1], b = sa[i],
    x[b] =
da0                     (y[a] == y[b] && y[a + j] == y[b + j]) ? p - 1 :
    p++;
4b4         }
9c7         for (int i = 1; i < n; i++) rank[sa[i]] = i;
05c         for (int i = 0, j; i < n - 1; lcp[rank[i++]] = k)
9f6             for (k && k--, j = sa[rank[i] - 1]; s[i + k] == s[j +
    k]; k++);
0d0     }
38e };
```

1.5 Z

```
1a8     vector<int> ret(sz);
6ed     for(int l=0,r=0,i=1;i<sz;i++) {
52d         auto expand = [&]() {
568             while(r<sz && s[r-l]==s[r]) r++;
38b             ret[i] = r-l;
8a3         };
08f         if(i >= r) {
018             l=r=i;
eec             expand();
9d9         } else {
bb7             if(ret[i-l] < r-i) ret[i] = ret[i-l];
4e6             else {
537                 l=i;
eec                 expand();
c99             }
48c         }
5d0     }
edf     return ret;
ad3 };
```

2 data-structures

2.1 MO algorithm

```
//  Description:
//      Answers queries offline with sqrt decomposition.
//  Complexity:
//      exec - O(n*sqrt(n)*O(remove / add))
d90 const int magic = 230;

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 {
2a8         return mp(l / magic, r) < mp(o.l / magic, o.r);
717     }
d25 };

5ce struct MO {
8d9     int sum;
55c     MO(vector<ll> &v) : sum(0), v(v), cnt(N), C(N) {}

fe9     void exec(vector<Query> &queries, vector<ll> &answers) {
14d         answers.resize(queries.size());
bfa         sort(queries.begin(), queries.end());

3df         int cur_l = 0;
cf5         int cur_r = -1;

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         }
dce     }

c96     void add(int i) {
683         sum += v[i];
0c3     }

17e     void remove(int i) {
f2f         sum -= v[i];
9a0     }

3b1     ll get_answer(int l, int r) {
e66         return sum;
520     }
3f7 };
```

2.2 Search Buckets

```
//  Data structure that provides two operations on an array:
//  1) set array[i] = x
//  2) count how many i in [start, end) satisfy array[i] < value
//  Both operations take sqrt(N log N) time. Amazingly, because of
//  the cache efficiency this is faster than the(log N)^2 algorithm
//  until N = 2-5 million.

39d template<typename T> struct search_buckets {
        // values are just the values in order. buckets are sorted in
           segments of BUCKET_SIZE (last segment may be smaller)
931     int N, BUCKET_SIZE;
c8c     vector<T> values, buckets;

9f1     search_buckets(const vector<T> &initial = {}) {
b48         init(initial);
611     }

7d4     int get_bucket_end(int bucket_start) const {
5e4         return min(bucket_start + BUCKET_SIZE, N);
0e2     }

ac2     void init(const vector<T> &initial) {
51b         values = buckets = initial;
2e7         N = values.size();
ecf         BUCKET_SIZE = 3 * sqrt(N * log(N + 1)) + 1;
8bb         cerr << "Bucket size: " << BUCKET_SIZE << endl;
```

```
2fc         for (int start = 0; start < N; start += BUCKET_SIZE)
23b             sort(buckets.begin() + start, buckets.begin() +
    get_bucket_end(start));
167     }

89e     int bucket_less_than(int bucket_start, T value) const {
8b6         auto begin = buckets.begin() + bucket_start;
188         auto end = buckets.begin() + get_bucket_end(bucket_start);
6b9         return lower_bound(begin, end, value) - begin;
21f     }

92e     int less_than(int start, int end, T value) const {
b52         int count = 0;
23a         int bucket_start = start - start % BUCKET_SIZE;
23c         int bucket_end = min(get_bucket_end(bucket_start), end);

93c         if (start - bucket_start < bucket_end - start) {
af4             while (start > bucket_start)
d53                 count -= values[--start] < value;
9d9         } else {
ad3             while (start < bucket_end)
62d                 count += values[start++] < value;
358         }

590         if (start == end)
308             return count;

655         bucket_start = end - end % BUCKET_SIZE;
e51         bucket_end = get_bucket_end(bucket_start);

23c         if (end - bucket_start < bucket_end - end) {
ec0             while (end > bucket_start)
807                 count += values[--end] < value;
9d9         } else {
612             while (end < bucket_end)
8da                 count -= values[end++] < value;
250         }

7bf         while (start < end && get_bucket_end(start) <= end) {
395             count += bucket_less_than(start, value);
a28             start = get_bucket_end(start);
5b1         }

c08         assert(start == end);
308         return count;
4cf     }
```

```
ea5     int prefix_less_than(int n, T value) const {
629         return less_than(0, n, value);
e45     }

2c3     void modify(int index, T value) {
985         int bucket_start = index - index % BUCKET_SIZE;
e50         int old_pos = bucket_start +
    bucket_less_than(bucket_start, values[index]);
48b         int new_pos = bucket_start +
    bucket_less_than(bucket_start, value);

85e         if (old_pos < new_pos) {
30f             copy(buckets.begin() + old_pos + 1, buckets.begin() +
    new_pos, buckets.begin() + old_pos);
8b8             new_pos--;
                // memmove(&buckets[old_pos], &buckets[old_pos + 1],
                   (new_pos - old_pos) * sizeof(T));
9d9         } else {
670             copy_backward(buckets.begin() + new_pos,
    buckets.begin() + old_pos, buckets.begin() + old_pos + 1);
                // memmove(&buckets[new_pos + 1], &buckets[new_pos],
                   (old_pos - new_pos) * sizeof(T));
b97         }

cac         buckets[new_pos] = value;
9cf         values[index] = value;
54b     }
ec7 };
```

2.3 Segtree 2D

```
//
//  Complexity:
//      build - O(N)
//      query - O(logN^2)
//

// struct Node {
//     Node () {}
//     Node operator + (const Node &o) const{
//         return Node ();
//     }
// };

4c1 namespace Seg2D {
```

```
14e     int n,m;
2ea     Node a[MAXN][MAXN], st[2*MAXN][2*MAXN];

b45     Node op (Node a, Node b){
534         return a + b;
978     }

0a8     void build (){
6e0         for(int i = 0; i < n; i++) for(int j = 0; j < m; j++)st[i+n][j+m]=a[i][j];
034         for(int i = 0; i < n; i++) for(int j = m - 1; j; --j)
c2d             st[i + n][j] = op(st[i + n][j << 1],st[i + n][j << 1 | 1]);
61e         for(int i = n - 1; i; --i) for(int j = 0; j < 2 * m; j++)
da1             st[i][j]=op(st[2 * i][j], st[2 * i + 1][j]);
de2     }
82b     void upd (int x, int y, Node v){
365         st[x + n][y + m] = v;
2e7         for(int j = y + m; j > 1; j /= 2) st[x + n][j / 2] = op(st[x + n][j], st[x + n][j ^ 1]);
eac         for(int i = x + n; i > 1; i /= 2) for(int j = y + m; j; j /= 2)
aa4             st[i / 2][j] = op(st[i][j], st[i ^ 1][j]);
12a     }

        // essa query vai de x0, y0 ate x1 - 1, y1 - 1 !!!
243     Node query (int x0, int x1, int y0, int y1) {
2ae         Node r = Node (); // definir elemento neutro da query!!!
6a8         for (int i0 = x0 + n, i1 = x1 + n; i0 < i1; i0 /= 2, i1 /= 2){
0b4             int t[4], q = 0;
f0e             if (i0 & 1) t[q++] = i0++;
847             if (i1 & 1) t[q++] = --i1;
18d             for(int k = 0; k < q; k++) for(int j0 = y0 + m, j1 = y1 + m; j0 < j1; j0 /= 2,j1 /= 2){
acb                 if (j0 & 1) r = op(r, st[t[k]][j0++]);
401                 if (j1 & 1) r = op(r, st[t[k]][--j1]);
a9d             }
3cd         }
4c1         return r;
33a     }
388 };
```

2.4 Sparse Segtree 2D

```
// Grid of dimensions N x M
//
// Operations:
//          update(x, y, val) <- update on point (x, y)
//          query(lx, rx, ly, ry) <- query on rectangle [lx..rx] x [ly..ry]
//
// O(logNlogM) complexity per operation
// O(N + UlogNlogM) memory, where U is the number of updates
//
// Possible changes:
//     - Speed: Use iterative segment tree or BIT on N axis
//     - O(UlogNlogM) memory: Make N axis sparse too
//

b5b namespace seg2d {
        // YOU ONLY NEED TO CHANGE THIS BLOCK
9a8     const int N = 200'000, M = 200'000;
0cb     using T = int32_t;
0ce     const T zero = 0; // INF if maintaining minimum, for example
cad     T merge(T a, T b) {
534         return a + b;
7f7     }

bf2     struct Node {
9fa         T s = zero;
8d9         int32_t l = 0, r = 0;
09f     };
28a     int root[4*N];
afe     vector<Node> v;

288     void upd(int& no, int l, int r, int pos, T val) {
270         if(not no) {
2ec             no = v.size();
                //assert(no < v.capacity());
903             v.emplace_back();
74e         }
ad4         if(l == r) v[no].s = val; // !!! OR v[no].s = merge(v[no].s, val) !!!
4e6         else {
ee4             int m = (l+r)/2;
611             auto &[s, nl, nr] = v[no];
303             if(pos <= m) upd(nl, l, m, pos, val);
926             else upd(nr, m+1, r, pos, val);
064             s = merge(v[nl].s, v[nr].s);
c01         }
741     }
a21     T qry(int no, int l, int r, int ql, int qr) {
3c6         if(not no) return zero;
966         if(qr < l || r < ql) return zero;
611         auto &[s, nl, nr] = v[no];
856         if(ql <= l && r <= qr) return s;
ee4         int m = (l+r)/2;
84f         return merge(qry(nl, l, m, ql, qr),
a48                 qry(nr, m+1, r, ql, qr));
eb6     }

389     void upd(int no, int l, int r, int x, int y, T val) {
30a         upd(root[no], 0, M-1, y, val);
8ce         if(l == r) return;
ee4         int m = (l+r)/2;
410         if(x <= m) upd(2*no, l, m, x, y, val);
1c3         else upd(2*no+1, m+1, r, x, y, val);
a50     }

89a     T qry(int no, int l, int r, int lx, int rx, int ly, int ry) {
8db         if(rx < l || r < lx) return zero;
060         if(lx <= l && r <= rx) return qry(root[no], 0, M-1, ly,
    ry);
ee4         int m = (l+r)/2;
019         return merge( qry(2*no, l, m, lx, rx, ly, ry),
d11                 qry(2*no+1, m+1, r, lx, rx, ly, ry) );
4df     }

153     void build(int no, int l, int r) {
fee         root[no] = v.size();
903         v.emplace_back();
8ce         if(l == r) return;
ee4         int m = (l + r) / 2;
b4b         build(2*no, l, m);
4d7         build(2*no+1, m+1, r);
88e     }

8d3     void update(int x, int y, T val) {
561         upd(1, 0, N-1, x, y, val);
96e     }

fad     int query(int lx, int rx, int ly, int ry) {
4c7         return qry(1, 0, N-1, lx, rx, ly, ry);
82c     }

        // receives max number of updates
        // each update creates at most logN logM nodes
        // RTE if we reserve less than number of nodes created
977     void init(int maxu) {
618         v.reserve(400*maxu);
903         v.emplace_back();
826         build(1, 0, N-1);
466     }
00e }
```

2.5 Binary Indexed Tree

```
// !! zero indexed !!
// all operations are O(logN)

273 template<typename T> struct Bit {
678     vector<T> bit;
052     Bit(int n): bit(n) {}

f3c     void update(int id, T val) {
bd2         for(id+=1; id<=int(bit.size()); id+=id&-id)
28c             bit[id-1] += val;
5bb     }

32d     T query(int id) {
e86         T sum = T();
2d6         for(id+=1; id>0; id-=id&-id)
fee             sum += bit[id-1];
e66         return sum;
dd6     }

        // returns the first prefix for which sum of 0..=pos >= val
        // returns bit.size() if such prefix doesnt exists
        // it is necessary that v[i] >= 0 for all i for monotonicity
ccc     int lower_bound(T val) {
e86         T sum = T();
bec         int pos = 0;
7f2         int logn = 31 - __builtin_clz(bit.size());
a99         for(int i=logn;i>=0;i--) {
148             if(pos + (1<<i) <= int(bit.size())
8f3                     && sum + bit[pos + (1<<i) - 1] < val) {
b1b                 sum += bit[pos + (1<<i) - 1];
b2c                 pos += (1<<i);
7ba             }
8f9         }
d75         return pos;
e0c     }
e4f };
```

2.6 Binary Indexed Tree 2D

```
// 0-indexed
// update(x, y, val): m[row][col] += val
// query(x, y): returns sum m[0..=x][0..=y]
ecd template <typename T> struct Bit2D {
14e     int n, m;
678     vector<T> bit;
26f     Bit2D(int _n, int _m): n(_n), m(_m), bit(n*m) {}

848     T query(int x, int y) {
19a         T res = 0;
ab3         for(x+=1;x>0;x-=x&-x)
aad             for(int z=y+1;z>0;z-=z&-z)
50c                 res += bit[(x-1)*m+z-1];
b50         return res;
a3e     }

8d3     void update(int x, int y, T val) {
157         for(x+=1;x<=n;x+=x&-x)
a36             for(int z=y+1;z<=m;z+=z&-z)
522                 bit[(x-1)*m+z-1] += val;
08d     }
5c8 };
```

2.7 Implicit Lazy Treap

```
// All operations are O(log N)
// If changes need to be made in lazy propagation,
// see Treap::push() and Treap::pull()
//
// Important functions:
// Treap::insert(int ind, T info)
// Treap::erase(int ind)
// Treap::reverse(int l, int r)
// Treap::operator[](int ind)

798 mt19937_64
   rng(chrono::steady_clock::now().time_since_epoch().count());

451 template <typename T> struct Treap {
3c9     struct node {
247         T info;
5ba         int l, r, sz;
5fa         uint64_t h;
aa6         bool rev;
f93         node() {}
f43         node(T _info): info(_info), l(0), r(0), sz(1), h(rng()),
    rev(0) {}
c9a     };

2a8     int root, ptr;
899     unique_ptr<node[]> v;
        // max: maximum number of insertions
e17     Treap(int max): root(0), ptr(0), v(new node[max+1]) {
            // v[0] is a placeholder node such that v[0].sz = 0
336         v[0].sz = 0;
541     }

6b4     void push(int nd) {
75a         node &x = v[nd];
974         if(x.rev) {
7f7             swap(x.l, x.r);
cd4             v[x.l].rev ^= 1;
acf             v[x.r].rev ^= 1;
49c             x.rev = 0;
c31         }
090     }
4b1     void pull(int nd) {
75a         node& x = v[nd];
f49         x.sz = v[x.l].sz + v[x.r].sz + 1;
a0a     }
27b     int new_node(T info) {
183         v[++ptr] = node(info);
500         return ptr;
71b     }
632     int getl(int nd) {
6ca         return v[v[nd].l].sz;
c0d     }

0ca     void merge(int l, int r, int& res) {
9b5         if(!l || !r) {
07e             res = l + r;
505             return;
75b         }
b8e         push(l); push(r);
a21         if(v[l].h > v[r].h) {
8ee             res = l;
8b4             merge(v[l].r, r, v[l].r);
9d9         } else {
516             res = r;
ca7             merge(l, v[r].l, v[r].l);
9e1         }
f39         pull(res);
66e     }
        // left treap has size pos
309     void split(int nd, int &l, int &r, int pos, int ra = 0) {
b36         if(!nd) {
c77             l = r = 0;
505             return;
62d         }
1d5         push(nd);
1c6         if(pos <= ra + getl(nd)) {
6fb             split(v[nd].l, l, r, pos, ra);
852             v[nd].l = r;
ca7             r = nd;
9d9         } else {
3e3             split(v[nd].r, l, r, pos, ra + getl(nd) + 1);
efd             v[nd].r = l;
2ac             l = nd;
065         }
afe         pull(nd);
6fa     }

        // Merges all s and makes them root
cff     template <int SZ> void merge(array<int, SZ> s) {
947         root = s[0];
724         for(int i=1;i<SZ;i++)
672             merge(root, s[i], root);
416     }

        // Splits root into SZ EXCLUSIVE intervals
        // [0..s[0]), [s[0]..s[1]), [s[1]..s[2])... [s[SZ-1]..end)
        // Example: split<2>({l, r}) gets the exclusive interval [l, r)
b2c     template <int SZ> array<int, SZ> split(array<int, SZ-1> s) {
7c5         array<int, SZ> res;
dc9         split(root, res[0], res[1], s[0]);
588         for(int i=1;i<SZ-1;i++) {
291             split(res[i], res[i], res[i+1], s[i]-s[i-1]);
775         }
815         root = 0;
b50         return res;
3a2     }

4b4     void insert(int ind, T info) {
488         auto s = split<2>({ind});
7a1         merge<3>({s[0], new_node(info), s[1]});
74e     }
97a     void erase(int ind) {
4e4         auto s = split<3>({ind, ind+1});
e6f         merge<2>({s[0], s[2]});
00b     }
        // Inclusive
8c1     void reverse(int l, int r) {
866         auto s = split<3>({l, r+1});
390         v[s[1]].rev ^= 1;
598         merge<3>(s);
518     }
420     T operator[](int ind) {
fbb         int nd = root;
            //assert(0 <= ind && ind < x->sz);
1d5         push(nd);
b3d         for(int ra=0, nra=getl(nd); nra != ind; nra = ra +
    getl(nd)) {
59f             if(nra < ind) ra = nra + 1, nd = v[nd].r;
9ef             else nd = v[nd].l;
1d5             push(nd);
341         }
464         return v[nd].info;
567     }
634 };
```

2.8 Iterative Segment Tree

```
// Supports non-commutative operations
//
// functions:
//  update(pos, val): set leaf node in pos to val
//  query(l, r): get sum of nodes in l and r
//
// Example: Range minimum queries segtree:
//  struct Node {
//      using T = int;
//      T mn;
//      Node(): mn(numeric_limits<T>::max()) {}
//      Node(T x): mn(x) {}
//      friend Node operator+(Node lhs, Node rhs) {
//          return Node(min(lhs.mn, rhs.mn));
//      }
//  };
//  using SegMin = SegIt<Node>;
//
//  int main() {
//      vector<int> v{3,1,3};
//      SegMin seg(v);
//      assert(seg.query(0, 2).mn == 1);
//      seg.update(1, 5);
//      assert(seg.query(0, 2).mn == 3);
//      assert(seg.query(1, 1).mn == 5);
//  }
//
// Submission: https://codeforces.com/contest/380/submission/193484078

a2c template <typename ND, typename T = typename ND::T>
2a0 struct SegIt {
1a8     int n;
c50     vector<ND> t;

0d6     SegIt(int _n): n(_n), t(2*n) {}
681     SegIt(vector<T> &v): n(v.size()), t(2*n) {
830         for(int i=0;i<n;i++)
766             t[i+n] = ND(v[i]);
6f2         build();
20d     }

0a8     void build() {
917         for(int i=n-1;i>0;i--)
f23             t[i] = t[2*i] + t[2*i+1];
6b1     }

6a3     void update(int pos, T val) {
f11         int p = pos + n;
5e3         t[p] = ND(val);
d08         while(p) {
d31             p /= 2;
6c7             t[p] = t[2*p] + t[2*p+1];
05a         }
283     }

a64     ND query(int l, int r) {
844         ND tl, tr;
e5f         r++; // to make query inclusive
4f7         for(l += n, r += n; l < r; l /= 2, r /= 2) {
e91             if(l&1) tl = tl + t[l++];
ae4             if(r&1) tr = t[--r] + tr;
c73         }
cf9         return tl + tr;
efd     }
7d4 };
```

2.9 NCE

```
// op(l, i), op(r, i) = true if they exist
// l = -1, r = v.size() otherwise
//
// Example: nce(v, greater<T>()): for each i returns
// nce[i] = {
//   biggest l < i such that v[l] > v[i]
//   smallest r > i such that v[r] > v[i]
// }
//
// Complexity: O(N)

751 template <typename T, typename OP>
101 vector<pair<int, int>> nce(vector<T> v, OP op) {
3d2     int n = v.size();
a3d     vector<pair<int, int>> res(n);
fd9     vector<pair<T, int>> st;
603     for(int i=0;i<n;i++) {
195         while(!st.empty() && !op(st.back().first, v[i]))
d73             st.pop_back();
a33         if(st.empty()) res[i].first = -1;
53d         else res[i].first = st.back().second;
e89         st.emplace_back(v[i], i);
cdc     }
23e     st.clear();
45b     for(int i=n-1;i>=0;i--) {
195         while(!st.empty() && !op(st.back().first, v[i]))
d73             st.pop_back();
0b7         if(st.empty()) res[i].second = n;
ce3         else res[i].second = st.back().second;
e89         st.emplace_back(v[i], i);
ba8     }
b50     return res;
793 }
```

2.10 Ordered Set

```
30f #include <ext/pb_ds/tree_policy.hpp>
0d7 using namespace __gnu_pbds;
// iterator find_by_order(size_t index), size_t order_of_key(T key)
67a template <typename T>
994 using ordered_set=tree<T, null_type, less<T>, rb_tree_tag,
        tree_order_statistics_node_update>;
```

2.11 Persistent segment tree.

```
// Complexity: O(logn) memory and time per query/update

c35 template<class T, int SZ> struct pseg {
ec3     static const int LIMIT = 1e7; // adjust
749     int l[LIMIT], r[LIMIT], nex = 0;
984     T val[LIMIT], lazy[LIMIT];

a69     int copy(int cur) {
269         int x = nex++;
5d0         val[x] = val[cur], l[x] = l[cur], r[x] = r[cur]; //
    lazy[x] = lazy[cur];
ea5         return x;
c0e     }
f57     T comb(T a, T b) { return a+b; }
c85     void pull(int x) { val[x] = comb(val[l[x]],val[r[x]]); }
    //  void push(int cur, int L, int R) {
    //      if (!lazy[cur]) return;
    //      if (L != R) {
    //          l[cur] = copy(l[cur]);
    //          val[l[cur]] += lazy[cur];
    //          lazy[l[cur]] += lazy[cur];
    //
    //          r[cur] = copy(r[cur]);
    //          val[r[cur]] += lazy[cur];
    //          lazy[r[cur]] += lazy[cur];
    //      }
    //      lazy[cur] = 0;
    //  }

        //// MAIN FUNCTIONS
e73     T query(int cur, int lo, int hi, int L, int R) {
e3f         if (lo <= L && R <= hi) return val[cur];
65a         if (R < lo || hi < L) return 0;
331         int M = (L+R)/2;
fb1         return comb(query(l[cur],lo,hi,L,M),
    query(r[cur],lo,hi,M+1,R));
e5b     }
14c     int upd(int cur, int pos, T v, int L, int R) {
b63         if (R < pos || pos < L) return cur;

dc1         int x = copy(cur);
7e8         if (pos <= L && R <= pos) { val[x] = v; return x; }

331         int M = (L+R)/2;
6e0         l[x] = upd(l[x],pos,v,L,M), r[x] = upd(r[x],pos,v,M+1,R);
d65         pull(x); return x;
89f     }
4eb     int build(vector<T>& arr, int L, int R) {
6a9         int cur = nex++;
651         if (L == R) {
d8c             if (L < (int) arr.size ()) val[cur] = arr[L];
75e             return cur;
62d         }

331         int M = (L+R)/2;
3a7         l[cur] = build(arr,L,M), r[cur] = build(arr,M+1,R);
c36         pull(cur); return cur;
a98     }

        //// PUBLIC
b3e     vector<int> loc;
        //void upd(int lo, int hi, T v) {
            loc.pb(upd(loc.back(),lo,hi,v,0,SZ-1)); }
        //T query(int ti, int lo, int hi) { return
            query(loc[ti],lo,hi,0,SZ-1); }
fa1     void build(vector<T>& arr) { loc.pb(build(arr,0,SZ-1)); }
e10 };
```

2.12 RMQ

```
//      Answers queries on a range.
//  Complexity:
//      build - O(N logN)
//      query - O(1)

f81 template <typename T> struct RMQ {
572     vector<vector<T>> dp;
6bc     T ops(T a, T b) { return min(a,b); }
fae     RMQ() {}
f16     RMQ(vector<T> v) {
3d2         int n = v.size();
1e7         int log = 32-__builtin_clz(n);
ca2         dp.assign(log, vector<T>(n));
79c         copy(all(v), dp[0].begin());
738         for(int l=1;l<log;l++) for(int i=0;i<n;i++) {
447             auto &cur = dp[l], &ant = dp[l-1];
c4e             cur[i] = ops(ant[i], ant[min(i+(1<<(l-1)), n-1)]);
c57         }
ec3     }
0ad     T query(int a, int b)  {
90f         if(a == b) return dp[0][a];
6a7         int p = 31-__builtin_clz(b-a);
dd7         auto &cur = dp[p];
ec5         return ops(cur[a], cur[b-(1<<p)+1]);
089     }
386 };
```

3 flow-and-matching

3.1 Dinitz

```
// get_flow(s, t): Returns max flow with source s and sink t
//
// Complexity: O(E*V^2). If unit edges only: O(E*sqrt(V))

14d struct Dinic {
670     struct edge {
b7a         int to, cap, flow;
0e3     };

789     vector<vector<int>> g;
1e7     vector<int> lvl;
37c     vector<edge> e;

db3     Dinic(int sz): g(sz), lvl(sz) {}

233     void add_edge(int s, int t, int cap) {
1f3         int id = e.size();
ffd         g[s].push_back(id);
614         e.push_back({t, cap, 0});
634         g[t].push_back(++id);
ff7         e.push_back({s, cap, cap});
8e0     }

123     bool bfs(int s, int t) {
5c1         fill(all(lvl), INF);
0d6         lvl[s] = 0;
26a         queue<int> q;
08b         q.push(s);
f76         while(!q.empty() && lvl[t] == INF) {
b1e             int v = q.front();
833             q.pop();
ca6             for(int id: g[v]) {
5c7                 auto [p, cap, flow] = e[id];
bd9                 if(lvl[p] != INF || cap == flow)
5e2                     continue;
ed5                 lvl[p] = lvl[v] + 1;
00a                 q.push(p);
e2f             }
e19         }
8de         return lvl[t] != INF;
c9d     }

2b1     int dfs(int v, int pool, int t, vector<int>& st) {
23a         if(!pool) return 0;
413         if(v == t) return pool;
138         for(;st[v]<(int)g[v].size();st[v]++) {
59b             int id = g[v][st[v]];
56f             auto &[p, cap, flow] = e[id];
783             if(lvl[v]+1 != lvl[p] || cap == flow) continue;
1de             int f = dfs(p, min(cap-flow, pool) , t, st);
235             if(f) {
c87                 flow += f;
ef4                 e[id^1].flow -= f;
abe                 return f;
964             }
e0b         }
bb3         return 0;
7a0     }

704     int get_flow(int s, int t) {
            //reset to initial state
            //for(int i=0;i<e.size();i++) e[i].flow = (i&1) ? e[i].cap
                : 0;
11e         int res = 0;
678         vector<int> start(g.size());
8ce         while(bfs(s,t)) {
cb6             fill(all(start), 0);
449             while(int f = dfs(s,INF,t,start))
5f5                 res += f;
7a9         }
b50         return res;
c83     }
a7c };
```

3.2 Hungarian

```
// Resolve o problema de assignment (matriz n x n)
// Colocar os valores da matriz em 'a' (pode < 0)
// assignment() retorna um par com o valor do
// assignment minimo, e a coluna escolhida por cada linha
```

```
//
// O(n^3)

513 template<typename T> struct Hungarian {
c04     static constexpr T INF = numeric_limits<T>::max();
1a8     int n;
a08     vector<vector<T>> a;
f36     vector<T> u, v;
5ff     vector<int> p, way;

0e9     Hungarian(int n_) : n(n_), a(n, vector<T>(n)), u(n+1), v(n+1),
    p(n+1), way(n+1) {}

40e     void set(int i, int j, T w) { a[i][j] = w; }

d67     pair<T, vector<int>> assignment() {
78a         for (int i = 1; i <= n; i++) {
8c9             p[0] = i;
625             int j0 = 0;
f49             vector<T> minv(n+1, INF);
0c1             vector<bool> used(n+1);
016             do {
472                 used[j0] = true;
d24                 int i0 = p[j0], j1 = -1;
8bc                 T delta = INF;
9ac                 for (int j = 1; j <= n; j++) if (!used[j]) {
7bf                     T cur = a[i0-1][j-1] - u[i0] - v[j];
9f2                     if (cur < minv[j]) minv[j] = cur, way[j] = j0;
821                     if (minv[j] < delta) delta = minv[j], j1 = j;
4d1                 }
f63                 for (int j = 0; j <= n; j++)
2c5                     if (used[j]) u[p[j]] += delta, v[j] -= delta;
6ec                     else minv[j] -= delta;
6d4                 j0 = j1;
52a             } while (p[j0] != 0);
016             do {
4c5                 int j1 = way[j0];
0d7                 p[j0] = p[j1];
6d4                 j0 = j1;
886             } while (j0);
431         }
306         vector<int> ans(n);
6db         for (int j = 1; j <= n; j++) ans[p[j]-1] = j-1;
def         return {-v[0], ans};
06e     }
7b6 };
```

3.3 Mincost Max-Flow

```
// shortest paths. Useful when the edges costs are negative.
// Infinite loop if there's a negative cycle.
//
// Constructor:
// MinCost(n, s, t)
// n - number of nodes in the flow graph.
// s - source of the flow graph.
// t - sink of the flow graph.
//
// Methods:
// - add_edge(u, v, cap, cost)
//   adds a directed edge from u to v with capacity `cap` and cost
   `cost`.
// - get_flow()
//   returns a pair of integers in which the first value is the
   maximum flow and the
//   second is the minimum cost to achieve this flow.
//
// Complexity: There are two upper bounds to the time complexity of
   getFlow
//              - O(max_flow * (E log V))
//              - O(V * E * (E log V))

cfd struct MinCost {
cd7     static constexpr int INF = 1e18;
670     struct edge {
22a         int to, next, cap, cost;
30a     };
748     int n, s, t;
439     vector<int> first, prev, dist;
70d     vector<bool> queued;
93b     vector<edge> g;

10d     MinCost(int _n, int _s, int _t) : n(_n), s(_s), t(_t),
52d         first(n, -1), prev(n), dist(n), queued(n) {};

5cb     void add_edge(int u, int v, int cap, int cost) {
270         int id = g.size();
4a6         g.pb({v, first[u], cap, cost});
c19         first[u] = id;
3a5         g.pb({u, first[v], 0, -cost});
727         first[v] = ++id;
b65     }

cbc     bool augment() {
```

```
04e         fill(all(dist), INF);
a93         dist[s] = 0;
0d9         queued[s] = 1;
26a         queue<int> q;
08b         q.push(s);
14d         while(!q.empty()) {
e4a             int u = q.front();
833             q.pop();
a04             queued[u] = 0;
ba2             for(int e = first[u]; e != -1; e = g[e].next) {
17a                 int v = g[e].to;
762                 int ndist = dist[u] + g[e].cost;
de7                 if(g[e].cap > 0 && ndist < dist[v]) {
d72                     dist[v] = ndist;
20e                     prev[v] = e;
076                     if(!queued[v]) {
2a1                         q.push(v);
b84                         queued[v] = 1;
67c                     }
90d                 }
cd6             }
a9a         }
85d         return dist[t] < INF;
            //UNCOMMENT FOR MIN COST WITH ANY FLOW (NOT NECESSARILY
                MAXIMUM)
            //return dist[t] <= 0;
cc6     }

a53     pair<int, int> get_flow() {
05f         int flow = 0, cost = 0;
456         while(augment()) {
612             int cur = t, curf = INF;
9c2             while(cur != s) {
a51                 int e = prev[cur];
887                 curf = min(curf, g[e].cap);
58c                 cur = g[e^1].to;
574             }
8bc             flow += curf;
1fc             cost += dist[t] * curf;
cd6             cur = t;
9c2             while(cur != s) {
a51                 int e = prev[cur];
09b                 g[e].cap -= curf;
787                 g[e^1].cap += curf;
58c                 cur = g[e^1].to;
765             }
24b         }
884             return {flow, cost};
42b     }
9e2 };
```

4 problems

4.1 LIS-2D

```
//      Given N pairs of numbers, find the lenght of the biggest
//      sequence such that a_i < a_i+1, b_i<b_i+1
//  Complexity:
//      O(N (logN)^2)
//  Details:
//      It uses divide & conquer with a segtree to make all
//      comparisons fast. memo[i] contains the answer for
//      the biggest sequence ending in i.
//  0eb0fc
//

093 const int N = 2e5 + 10;

2ad int n, memo[N];
89b pair<int, int> a[N];

a2f struct segTree {
1a8     int n;
2e6     vector<ll> st;
aac     ll combine(ll a, ll b) {
a16         return max (a, b); // TODO define merge operator
d19     }
401     segTree() {}
8d5     segTree(int n) : n (n), st (2 * n, -1) {}
625     void update(int i, ll x) {
cbf         st[i += n] = max (x, st[i + n]); // TODO change update
    operation
b8f         while (i > 1) {
29a             i >>= 1;
4f9             st[i] = combine(st[i << 1], st[i << 1 | 1]);
479         }
16b     }
        // query from l to r, inclusive
02a     ll query(int l, int r) {
721         ll resl = -1, resr = -1;
326         for (l += n, r += n+1; l < r; l >>= 1, r >>= 1) {
```

```
ced             if (l & 1) resl = combine(resl, st[l++]);
386             if (r & 1) resr = combine(st[--r], resr);
97c         }
f9d         return combine(resl, resr);
4a1     }
220 };


6cb void divide_conquer (int l, int r) {
8ce     if (l == r) return;
ee4     int m = (l + r) / 2;
917     divide_conquer (l, m); // calculamos o valor para esquerda

        // propagamos para a direita
        // temos que comprimir coordenadas
f2a     vector<int> M;
b08     for (int j = l; j <= m; j++) {
ee6         M.push_back (a[j].first + 1);
153         M.push_back (a[j].second);
d22     }
38c     for (int j = m + 1; j <= r; j++) {
cd2         M.push_back (a[j].first);
153         M.push_back (a[j].second);
1d0     }
862     sort (all (M));
8fd     unique (all (M));
078     auto find_pos = [&] (int x) {
23d         return (int) (lower_bound (all (M), x) - M.begin ());
bae     };

ea3     vector<array<int, 4>> events;
        // coord_x, L/R, coord_y, memo/ind
917     for (int j = l; j <= m; j++)
64b         events.pb ({find_pos(a[j].first + 1), 0,
    find_pos(a[j].second), memo[j]});
ed6     for (int j = m + 1; j <= r; j++)
992         events.pb ({find_pos(a[j].first), 1,
    find_pos(a[j].second), j});
bb0     sort (all (events));

e5d     segTree st (M.size () + 1);
653     for (auto [x, op, y, M] : events) {
5f7         if (op == 0) st.update (y, M);
4e2         else memo[M] = max (memo[M], st.query (0, y - 1) + 1);
e82     }
7cf     divide_conquer (m + 1, r); // calculamos o valor para direita
76b }
```

5 math

5.1 Coprimes

```
//      Given a set o integers, calculates the quantity of integers
//      in the set coprimes with x. You can actually make queries on
//      anything related to the coprimes. For example, sum of
   comprimes.
//  Complexity:
//      precalc - O(n logn)
//      add - O(sigma(N))
//      coprime - O(sigma(N))
//  Details:
//      It uses Mobius Function. To add or remove an integer of the set
//      just change sign to +1 or -1.

49c struct Coprimes {
1a8     int n;
bae     vector<ll> cnt;
afe     vector<int> U;
74f     vector<vector<int>> fat;
bbe     Coprimes () {}
7bb     Coprimes (int n) : n(n), U(n), fat(n), cnt(n) {
e91         precalc ();
67b     }
fe8     void precalc () {
9cf         U[1] = 1;
f65         for (int i = 1; i < n; i++) fat[i].pb (1);
6f5         for (int i = 1; i < n; i++) {
2ef             for (int j = 2 * i; j < n; j += i) U[j] -= U[i];
850             if (fat[i].size () == 1 && i > 1) {
1ec                 for (int j = i; j < n; j += i)
c25                     for (int k = fat[j].size () - 1; k >= 0; k--)
d6d                         fat[j].pb (i * fat[j][k]);
62c             }
100         }
ab4     }
2f1     void add(int x, int sign){
2ed         for(auto d : fat[x]) cnt[d] += sign;
37f     }
a33     ll coprimo(int x){
92b         ll quant = 0;
41d         for(auto d : fat[x]){
903             quant += U[d] * cnt[d];
3b0         }
2bd         return quant;
0ce     }
1dc };
```

5.2 Gauss elimination - modulo 2

```
//  Description:
//      Solves a linear system with n equations and m - 1 variables.
//      Is faster duo to the use of bitset.
//  Complexity: O(n^2 * m / 32)


//  Details:
//      Function solve return a boolean indicating if system is possible
//      or not. Also, if it is possible, the parameter maintains the answer.

146 struct Gauss_mod2 {
14e     int n, m;
66e     array<int, M> pos;
3e5     int rank = 0;
75f     vector<bitset<M>> a;

        // n equations, m-1 variables, last column is for coefficients
616     Gauss_mod2(int n, int m, vector<bitset<M>> &a): n(n), m(m), a(a) {
e55         pos.fill(-1);
eac     }

728     int solve(bitset<M> &ans) {
a73         for (int col = 0, row = 0; col < m && row < n; col++) {
896             int one = -1;
016             for (int i = row; i < n; i++) {
e6e                 if (a[i][col]) {
7ba                     one = i;
c2b                     break;
dff                 }
edb             }

b1a             if (one == -1) { continue; }

5fb             swap(a[one], a[row]);

8a0             pos[col] = row;

79f             for (int i = row + 1; i < n; i++) {
505                 if (a[i][col])
95d                     a[i] ^= a[row];
ecc             }
616             ++row, ++rank;
400         }

d16         ans.reset();

ca1         for (int i = m - 1; i >= 0; i--) {
413             if (pos[i] == -1) ans[i] = true;
4e6             else {
ec8                 int k = pos[i];
322                 for (int j = i + 1; j < m; j++) if (a[k][j]) ans[i] = ans[i] ^ ans[j];
506                 ans[i] = ans[i] ^ a[k][m];
4cc             }
e3a         }

332         for (int i = rank; i < n; i++) if (a[i][m]) return 0;

6a5         return 1;
dfa     }
f27 };
```

5.3 Gauss Xor - Gauss elimination mod 2

```
//                 maintains a basis of the set.
//  Complexity: query - O(D)
//              add - O(D)

//  Functions:
//      query(mask) - returns the biggest number that can
//                    be made if you initially have cur and
//                    it cannot be bigger than lim.
//      add(mask) - adds mask to the basis.

//  Details:
//      We are assuming the vectors have size D <= 64. For general
//      case, you may change ll basis[] for bitset<D> basis[].

189 const int logN = 30;

3d7 struct Gauss_xor {
387     int basis[logN];
c9f     Gauss_xor () { memset (basis, 0, sizeof (basis)); }
5b8     void add (int x) {
a28         for (int j = logN - 1; j >= 0; j--) {
be0             if (x & (1ll << j)) {
335                 if (basis[j]) x ^= basis[j];
4e6                 else {
467                     basis[j] = x;
505                     return;
681                 }
58a             }
3f8         }
78e     }

cbd     int query (int j, int cur, int lim, bool mn) {
bfc         if (j < 0) return cur;
c5f         if (mn) {
5ad             return query (j - 1, max (cur, cur ^ basis[j]), lim,
   1);
ec1         }
4e6         else {
9bc             if (lim & (1ll << j)) {
2c4                 if (cur & (1ll << j)) {
8ee                     int res = query (j - 1, cur, lim, 0);
d1e                     if (res) return res;
a86                 }
4e6                 else {
05a                     if (basis[j]) {
4ea                         int res = query (j - 1, cur ^ basis[j],
   lim, 0);
d1e                         if (res) return res;
591                     }
7d9                 }
ce3                 int val = min (cur, cur ^ basis[j]);
e2d                 if ((val & (1ll << j)) == 0) return query (j - 1,
   val, lim, 1);
98b                 else return 0;
39a             }
4e6             else {
2c4                 if (cur & (1ll << j)) {
e5f                     if (!basis[j]) return 0;
7c0                 }
12a                 return query (j - 1, min (cur, cur ^ basis[j]),
   lim, 0);
a33             }
651         }
bda     }
5db };
```

5.4 NTT - Number Theoretic Transform

```
//  Complexity: O(N logN)

//  Functions:
//      multiply(a, b)

//  Details:
//      Not all primes can be used and p = 998244353 is the most used
   prime.
//      To multiply it for a general modulus, use 3 different possible
   primes
//      and use Chinese Remainder Theorem to get the answear.

//  Possibilities
//  { 7340033, 5, 4404020, 1 << 20 },
//  { 415236097, 73362476, 247718523, 1 << 22 },
//  { 463470593, 428228038, 182429, 1 << 21},
//  { 998244353, 15311432, 469870224, 1 << 23 },
//  { 918552577, 86995699, 324602258, 1 << 22 }

ea8 namespace NTT {
7e5     using Z = mint<998244353>;
a92     const Z root(15311432), root_1(469870224);
32f     int root_pw = 1<<23;

506     void fft(vector<Z> & a, bool invert) {
94d     int n = a.size();

5f9         for (int i = 1, j = 0; i < n; i++) {
4af             int bit = n >> 1;
474             for (; j & bit; bit >>= 1)
53c                 j ^= bit;
53c             j ^= bit;
aa5             if (i < j) swap(a[i], a[j]);
f3a         }

eb7         for (int len = 2; len <= n; len <<= 1) {
cf9             Z wlen = invert ? root_1 : root;
5ae             for (int i = len; i < root_pw; i <<= 1)
fe1                 wlen *= wlen;

6c8             for (int i = 0; i < n; i += len) {
973                 Z w(1);
2ae                 for (int j = 0; j < len / 2; j++) {
80c                     Z u = a[i+j], v = a[i+j+len/2] * w;
6c3                     a[i+j] = u + v;
273                     a[i+j+len/2] = u - v;
3e4                     w *= wlen;
6f5                 }
092             }
0da         }

eb5         if (invert) {
c61             Z n_1 = Z(n).inv();
bdf             for (Z & x : a) x *= n_1;
ff8         }
9e7     }

2e8     vector<Z> multiply(vector<Z> &a, vector<Z> &b) {
2a8         vector<Z> fa = a, fb = b;
015         int sz = a.size() + b.size() - 1, n = 1;
4ba         while (n < sz) n <<= 1;

75e         fa.resize(n), fb.resize(n);
404         fft(fa, 0), fft(fb, 0);
991         for (int i = 0; i < fa.size(); i++) fa[i] *= fb[i];

e55         fft(fa, 1);
c5c         fa.resize(sz);
83d         return fa;
61f     }
bf1 };
```

5.5 Bit iterator

```
// use: for(auto it: BitIterator(n,m) { int mask = *it; ... }

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 };
```

5.6 Convolutions

```
// Complexity: O(N logN)

// Functions:
//     multiply(a, b)
//     multiply_mod(a, b, m) - return answer modulo m

// Details:
//     For function multiply_mod, any modulo can be used.
//     It is implemented using the technique of dividing
//     in sqrt to use less fft. Function multiply may have
//     precision problems.
//     This code is faster than normal. So you may use it
//     if TL e tight.

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);}

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);

        // 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     }

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     }

ba5     vector<num> fa, fb;
        // multiply with less fft by using complex numbers.
318     vector<ll> multiply(vector<ll> &a, vector<ll> &b);

        // 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 };

// 16be45
7b3 vector<ll> fft::multiply(vector<ll> &a, vector<ll> &b) {
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);
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     }

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     }

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 }

// 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 }

b86 vector<ll> fft::square_mod(vector<ll> &a, ll m) {
dfb     return multiply_mod(a, a, m, 1);
dde }
```

5.7 Dirichlet Trick

```
//      Find the partial sum of a multiplicative function.
//      This code works for Phi or Mobius functions.
// Details:
//      It is necessary to precalculate the values of at least
//      sqrt (N). But, the optimal value might be around N^(2/3)

04a namespace Dirichlet {
d9a     vector<int> f;
ce9     map<int,int> mp;

4ce     void init (vector<int> &mul_func) {
8dc         f.resize (mul_func.size ());
6d7         for (int i = 1; i < mul_func.size (); i++) f[i] = f[i - 1]
    + mul_func[i];
5ef     }

cfc     int calc (int x) {
486         if(x<=N) return f[x];
c4e         if(mp.find(x)!=mp.end()) return mp[x];
651         int ans = x * (x + 1) / 2;
166         for(int i = 2, r; i <= x; i = r + 1) {
37f             r=x/(x/i);
2a9             ans -= calc(x/i)*(r-i+1);
624         }
b6f         return mp[x]=ans;
3b5     }
187 }
```

5.8 Extended gcd

```
5d9 pair<int,int> egcd(int a, int b) {
02b     if(b == 0) return {1, 0};
60e     auto [x, y] = egcd(b, a%b);
f07     return {y, x - y * (a/b)};
5e5 }
```

5.9 Fast Walsh-Hadamard trasform

```
//  - op(a, b) = a "xor" b, a "or" b, a "and" b
// Complexity: O(n log n)

29a const ll N = 1<<20;

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
                        // io[i+j] = x+y; // and
                        // 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];
                         /// Modular inverse if required here
174                     io[i+j] = (x+y)>>1, io[i+j+d] = (x-y)>>1; //
    xor
                        // io[i+j] = x-y; // and
                        // io[i+j+d] = y-x; // or
027                 }
711             }
fd4         }
e69     }
        // 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     }
        // 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;
```

5.10 Gauss elimination

```
76a using arr = valarray<double>;
320 struct Gauss {
14e     int n, m;
146     vector<arr> v;
b2f     Gauss(int _n, int _m): n(_n), m(_m), v(n, arr(m)) {}

d5f     arr& operator[](int i) { return v[i]; }

958     void eliminate() {
            // eliminate column j
8ec         for(int j=0;j<min(n, m);j++) {
2e9             v[j].swap(*max_element(v.begin()+j, v.end(),
73a                 [&](arr& a, arr& b) { return abs(a[j]) <
   abs(b[j]); }));
8c0             for(int i=j+1;i<n;i++)
f68                 v[i] -= v[i][j] / v[j][j] * v[j];
06c         }
b9c     }
acf };
```

5.11 Mint

```
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 };
```

5.12 Polynomial operations

```
// Multi-point polynomial evaluation: O(n*log^2(n))
// Polynomial interpolation: O(n*log^2(n))

// Works with NTT. For FFT, just replace the type.

f66 #define SZ(s) int(s.size())
7e5 using Z = mint<998244353>;
689 typedef vector<Z> poly;

de4 poly add(poly &a, poly &b) {
bea     int n = SZ(a), m = SZ(b);
dba     poly ans(max(n, m));
0cc     for (int i = 0; i < max(n, m); i++) {
d20         if (i < n)
7a8             ans[i] += a[i];
1d6         if (i < m)
229             ans[i] += b[i];
51a     }
277     while (SZ(ans) > 1 && !ans.back().x) ans.pop_back();
ba7     return ans;
3d8 }

dfb poly invert(poly &b, int d) {
5d9     poly c = {b[0].inv ()};
cfd     while (SZ(c) <= d) {
0c0         int j = 2 * SZ(c);
6aa         auto bb = b;
3df         bb.resize(j);
55c         poly cb = NTT::multiply(c, bb);
fbf         for (int i = 0; i < SZ(cb); i++) cb[i] = Z(0) - cb[i];
6d7         cb[0] += 2;
beb         c = NTT::multiply(c, cb);
4cd         c.resize(j);
6bd     }
ea9     c.resize(d + 1);
807     return c;
089 }

b8c pair<poly, poly> divslow(poly &a, poly &b) {
bea     poly q, r = a;
f69     while (SZ(r) >= SZ(b))
f95     {
759         q.pb(r.back() * b.back().inv ());
41f         if (q.back().x)
5c5             for (int i = 0; i < SZ(b); i++)
f95             {
ab0                 r[SZ(r) - i - 1] = r[SZ(r) - i - 1] - q.back() *
    b[SZ(b) - i - 1];
864             }
515         r.pop_back();
07b     }
bb2     reverse(all(q));
442     return {q, r};
7dc }

958 pair<poly, poly> divide(poly &a, poly &b) { // returns
    {quotient,remainder}
d01     int m = SZ(a), n = SZ(b), MAGIC = 750;
fbc     if (m < n)
33d         return {{0}, a};
61b     if (min(m - n, n) < MAGIC)
7c0         return divslow(a, b);
64a     poly ap = a; reverse(all(ap));
424     poly bp = b; reverse(all(bp));
160     bp = invert(bp, m - n);
c5b     poly q = NTT::multiply(ap, bp);
63e     q.resize(SZ(q) + m - n - SZ(q) + 1, 0);
bb2     reverse(all(q));
72d     poly bq = NTT::multiply(b, q);
df0     for (int i = 0; i < SZ(bq); i++) bq[i] = Z(0) - bq[i];
b56     poly r = add(a, bq);
442     return {q, r};
224 }

204 vector<poly> tree;

1ee void filltree(vector<Z> &x) {
b3a     int k = SZ(x);
ffb     tree.resize(2 * k);
13c     for (int i = k; i < 2 * k; i++) tree[i] = {Z(0) - x[i - k], 1};
bcd     for (int i = k - 1; i; i--)
9ec         tree[i] = NTT::multiply(tree[2 * i], tree[2 * i + 1]);
6f6 }

591 vector<Z> evaluate(poly &a, vector<Z> &x) {
2b8     filltree(x);
b3a     int k = SZ(x);
a34     vector<poly> ans(2 * k);
8f8     ans[1] = divide(a, tree[1]).second;
7db     for (int i = 2; i < 2 * k; i++) ans[i] = divide(ans[i >> 1],
    tree[i]).second;
607     vector<Z> r;
c02     for (int i = 0; i < k; i++) r.pb(ans[i + k][0]);
4c1     return r;
f5a }

e05 poly derivate(poly &p) {
c5e     poly ans(SZ(p) - 1);
ff7     for (int i = 1; i < SZ(p); i++) ans[i - 1] = p[i] * i;
ba7     return ans;
c7f }

5a1 poly interpolate(vector<Z> &x, vector<Z> &y) {
2b8     filltree(x);
8c4     poly p = derivate(tree[1]);
3ed     int k = SZ(y);
4fe     vector<Z> d = evaluate(p, x);
888     vector<poly> intree(2 * k);
a0d     for (int i = k; i < 2 * k; i++) intree[i] = {y[i - k] / d[i -
    k]};
a79     for (int i = k - 1; i; i--) {
a49         poly p1 = NTT::multiply(tree[2 * i], intree[2 * i + 1]);
026         poly p2 = NTT::multiply(tree[2 * i + 1], intree[2 * i]);
452         intree[i] = add(p1, p2);
d48     }
2a3     return intree[1];
ae8 }
```

6 dynamic-programming

6.1 CHT - Dynamic Convex Hull Trick

```
// Complexity:
//     add - O(logN)
//     query - O(logN)

// Functions:
//     add(a, b) - add line (a * x + b) to the convex hull.
//     query (x) - return the maximum value of any line on point x.

// Details:
//     If you want to maintain the bottom convex hull, it is
//     easier to just change the sign. Be careful with overflow
//     on query. Can use __int128 to avoid.

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 };

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     }

bbb     void update(iterator x) {
b2a         if (next(x) == end()) x->p = LINF;
772         else if (x->a == next(x)->a) x->p = x->b >= next(x)->b ?
    LINF : -LINF;
424         else x->p = div(next(x)->b - x->b, x->a - next(x)->a);
0c4     }

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     }

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     }

4ad     ll query(ll x) {
229         assert(!empty());
7d1         auto l = *lower_bound(x);
aba         return l.a * x + l.b;
3f5     }
8f2 };
```

7 graphs

7.1 Euler Walk

```
//               starting at src. Not necesseraly a cycle. Works for both
//               directed and undirected. Returns vector
//               of \{vertex,label of edge to vertex\}.
//               Second element of first pair is always $-1$.
//  Complexity: O(N + M)
//

843 template<bool directed> struct Euler {
a06     using pii = pair<int, int>;
060     int N;
109     vector<vector<pii>> adj;
1f1     vector<vector<pii>::iterator> its;
cbd     vector<bool> used;
ee1     Euler (int _N) : N (_N), adj (_N) {}
010     void add_edge(int a, int b) {
e63         int M = used.size (); used.push_back(0);
215         adj[a].emplace_back(b, M);
f91         if (!directed) adj[b].emplace_back(a, M);
e01     }
94e     vector<pii> solve(int src = 0) {
29e         its.resize(N);
3c7         for (int i = 0; i < N; i++) its[i] = begin (adj[i]);

805         vector<pii> ans, s{{src,-1}}; // {{vert,prev vert},edge label}
2f5         int lst = -1; // ans generated in reverse order
bdd         while (s.size ()) {
723             int x = s.back ().first; auto& it=its[x], en=end(adj[x]);
0d5             while (it != en && used[it->second]) ++it;
8af             if (it == en) { // no more edges out of vertex
9c7                 if (lst != -1 && lst != x) return {};
                    // not a path, no tour exists
f10                 ans.push_back(s.back ()); s.pop_back();
816                 if (s.size ()) lst=s.back ().first;
38e             } else s.push_back(*it), used[it->second] = 1;
acb         } // must use all edges
0f8         if (ans.size () != used.size () + 1) return {};
9ee         reverse(all(ans)); return ans;
340     }
d90 };
```

7.2 Stable Marriage problem

```
//  Given n men and n women, where each person has ranked all
//  members of the opposite sex in order of preference, marry
//  the men and women together such that there are no two people
//  of opposite sex who would both rather have each other than
//  their current partners. When there are no such pairs of
//  people, the set of marriages is deemed stable.
//
//  If the lists are complete, there is always a solution that
//  can be founc in O(n * m).
//
//  a - Rank list of first group
//  b - Rank list of first group
//  solve () - Gives an stable matching covering the first group.
//             It is necessary that n <= m.
//

7da struct StableMarriage {
14e     int n, m;
fbe     using vvi = vector<vector<int>>;
10e     vvi a, b;
142     StableMarriage (int n, int m, vvi a, vvi b) : n (n), m (m), a (a), b (b) {};

201     vector<pair<int, int>> solve () {
5af         assert (n <= m);
d81         vector<int> p (n), mb (m, -1);
8e0         vector rank (m, vector<int> (n));
fd3         for (int i = 0; i < m; i++) for (int j = 0; j < n; j++) rank[i][b[i][j]] = j;
26a         queue<int> q;

5af         for (int i = 0; i < n; i++) q.push (i);
402         while (q.size ()) {
be1             int u = q.front (); q.pop ();

838             int v = a[u][p[u]++];
af0             if (mb[v] == -1) {
4d0                 mb[v] = u;
36a             }
4e6             else {
b60                 int other_u = mb[v];
a70                 if (rank[v][u] < rank[v][other_u]) {
4d0                     mb[v] = u;
76b                     q.push (other_u);
66b                 }
4e6                 else {
f73                     q.push (u);
366                 }
5ed             }
f7b         }
f77         vector<pair<int, int>> ans;
6e1         for (int i = 0; i < m; i++) if (mb[i] != -1) ans.pb({mb[i], i});
ba7         return ans;
fe6     }
ab7 };
```

7.3 2-SAT

```
// Complexity: O(|V| + |E|)
//
// Functions:
//     either (a, b) - (a | b) is true
//     implies (a, b) - (a -> b) is true
//     must (x) - x is true
//     atMostOne (v) - ensure that at most one of these
//                     variables is true
//     solve () - returns the answer if system is possible.
//
// Details:
//      Not x is equivalente to ~x on this template.

bf0 struct SCC {
dc0     int N, ti = 0; vector<vector<int>> adj;
70b     vector<int> disc, comp, st, comps;
d5d     void init(int _N) {
b77         N = _N;
0f3         adj.resize(N);
9a3         disc.resize(N);
a4e         comp = vector<int>(N,-1);
d84     }
768     void add_edge(int x, int y) { adj[x].push_back(y); }
e34     int dfs(int x) {
4b4         int low = disc[x] = ++ti; st.push_back(x); // disc[y] != 0 -> in stack
989         for (auto y : adj[x]) if (comp[y] == -1) {
494             auto b = disc[y] ? : dfs(y); auto &a = low;
28a             b < a ? a = b, 1 : 0;
46d         }
e79         if (low == disc[x]) { // make new SCC, pop off stack until you find x
b3d             comps.push_back(x); for (int y = -1; y != x;)
e45                 comp[y = st.back()] = x, st.pop_back();
90f         }
b2b         return low;
22e     }
761     void gen() {
50d         for (int i = 0; i < N; i++) if (!disc[i]) dfs(i);
3a5         reverse(all(comps));
592     }
b15 };

417 struct TwoSAT {
5ec     int N = 0; vector<pair<int, int>> edges;
8b2     void init (int _N) { N = _N; }
4b3     int addVar () { return N++; }
8c0     void either (int x, int y) {
8f5         x = max(2 * x, -1 - 2 * x), y = max(2 * y, -1 - 2 * y);
599         edges.push_back ({x, y});
c50     }
77e     void implies (int x, int y) {
7ab         either (~x,y);
288     }
fa9     void must (int x) {
f97         either (x,x);
b95     }
0b6     void atMostOne (const vector<int>& li) {
414         if (li.size () <= 1) return;
da9         int cur = ~li[0];
113         for (int i = 2; i < li.size (); i++) {
b70             int next = addVar();
698             either(cur, ~li[i]); either(cur, next);
0af             either(~li[i], next); cur = ~next;
c0d         }
ed7         either(cur, ~li[1]);
a57     }
28e     vector<bool> solve() {
4ad         SCC S; S.init(2 * N);
d62         for (auto [x, y] : edges)
7ce             S.add_edge(x ^ 1, y), S.add_edge(y ^ 1, x);
f58         S.gen(); reverse(all(S.comps)); // reverse topo order
76d         for (int i = 0; i < 2 * N; i += 2)
7bf             if (S.comp[i] == S.comp[i^1]) return {};
586         vector<int> tmp(2 * N);
6de         for (auto i : S.comps) if (!tmp[i])
94d             tmp[i] = 1, tmp[S.comp[i^1]] = -1;
f18         vector<bool> ans(N);
45f         for (int i = 0; i < N; i++) ans[i] = tmp[S.comp[2*i]] == 1;
ba7         return ans;
b35     }
46a };
```

7.4 Block Cut Tree

```
// Constructor: SCC(|V|, |E|, [[v, e]; |V|])
// Complexity: O(N+M)

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
                              // if v is NOT an articulation point, then
                                  gart[v] is empty

        // 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;

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]);
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         };
0f0         for(int v=0;v<n;v++) if(!vis[v]) dfs(v, 1);
ff8     }
f70 };
```

7.5 LCA

```
33e struct LCA {
0ce     vector<int> pre, dep; // preorder traversal and depth
e16     RMQ<pair<int,int>> rmq;

c67     LCA() {}
1a3     LCA(int sz, vector<int> g[], int root): pre(sz), dep(sz) {
837         vector<pair<int,int>> tour; tour.reserve(2*sz-1);
6be         auto dfs = [&](int v, int dad, auto& self) -> void {
e17             pre[v] = tour.size();
95e             tour.push_back({dep[v],v});
27e             for(int p: g[v]) if(p != dad) {
5b8                 dep[p] = dep[v]+1;
f5e                 self(p,v,self);
95e                 tour.push_back({dep[v],v});
af6             }
61f         };
862         dfs(root, root, dfs);
b69         rmq = RMQ<pair<int,int>>(tour);
234     }

4ea     int query(int a, int b) {
ca7         if(pre[a] > pre[b]) swap(a,b);
d1b         return rmq.query(pre[a],pre[b]).second;
f05     }

b5d     int dist(int a, int b) {
969         int c = query(a,b);
5a3         return dep[a] + dep[b] - 2*dep[c];
3de     }
788 };
```

7.6 Tarjan for undirected graphs

```
// Constructor: SCC(|V|, |E|, [[v, e]; |V|])
//
// Complexity: O(N+M)
```

```
bf0 struct SCC {
27d     vector<bool> bridge; // bridge[e]: true if edge e is a bridge
f7a     vector<int> comp; // comp[v]: component of vertex v

8d3     int ncomp; // number of components
1df     vector<int> sz; // sz[c]: size of component i (number of
    vertexes)
413     vector<vector<pair<int, int>>> gc; // gc[i]: list of adjacent
    components

        // assumes auto [neighbor_vertex, edge_id] =
            g[current_vertex][i]
d90     SCC(int n, int m, vector<pair<int, int>> g[]): bridge(m),
    comp(n, -1), ncomp(0) {
5c8         vector<bool> vis(n);
594         vector<int> low(n), prof(n);

208         function<void(int,int)> dfs = [&](int v, int dad) {
cca             vis[v] = 1;
290             for(auto [p, e]: g[v]) if(p != dad) {
137                 if(!vis[p]) {
690                     low[p] = prof[p] = prof[v] + 1;
345                     dfs(p, v);
de7                     low[v] = min(low[v], low[p]);
c9b                 } else low[v] = min(low[v], prof[p]);
edd             }
3f2             if(low[v] == prof[v]) ncomp++;
729         };
548         for(int i=0;i<n;i++) if(!vis[i]) dfs(i, -1);

7cc         sz.resize(ncomp); gc.resize(ncomp);

ac9         int cnt = 0;
c64         function<void(int,int)> build = [&](int v, int c) {
440             if(low[v] == prof[v]) c = cnt++;
d5f             comp[v] = c;
24a             sz[c]++;
936             for(auto [p, e]: g[v]) if(comp[p] == -1) {
5e7                 build(p, c);
a54                 int pc = comp[p];
d59                 if(c != pc) {
442                     bridge[e] = true;
718                     gc[c].emplace_back(pc, e);
2a3                     gc[pc].emplace_back(c, e);
b6e                 }
cf9             }
731         };
c7d         for(int i=0;i<n;i++) if(comp[i] == -1) build(i, -1);
561     }
a1e };
```

7.7 Virtual Tree

```
f03 namespace vtree {
dbb     vector<int> vg[MAX];

        // receives list of vertexes and returns root of virtual tree
        // v must NOT be empty
cf3     int build(vector<int> vs, LCA& lca) {
aa3         auto cmp = [&](int i, int j) {
d31             return lca.pre[i] < lca.pre[j];
645         };
de1         sort(all(vs), cmp);
7b1         for(int i=vs.size()-1; i>0; i--)
    vs.push_back(lca.query(vs[i-1], vs[i]));
47a         sort(all(vs));
f7c         vs.resize(unique(all(vs))-vs.begin());
de1         sort(all(vs), cmp);
a9f         for(auto v: vs) vg[v].clear();
ab1         for(int i=1;i<vs.size();i++) {
258             int dad = lca.query(vs[i-1], vs[i]);
993             vg[dad].push_back(vs[i]);
d85             vg[vs[i]].push_back(dad);
d34         }
367         return vs[0];
373     }
ea9 }
```

8 geometry

8.1 Circle

```
// only with double numbers since most of the operations of a circle
    can't be
// done with only integers. Therefore, this template depends on
    point_double.cpp.
//
// All operations' time complexity are O(1)
```

```
1d5 const double PI = acos(-1);

aa8 struct circle {
664     point o; double r;
d0b     circle() {}
187     circle(point _o, double _r) : o(_o), r(_r) {}
223     bool has(point p) {
804         return (o - p).norm2() < r*r + EPS;
003     }
8b0     vector<point> operator/(circle c) { // Intersection of circles.
4b4         vector<point> inter;                  // The points in
    the output are in ccw order.
6ac         double d = (o - c.o).norm();
376         if(r + c.r < d - EPS || d + min(r, c.r) < max(r, c.r) -
    EPS)
21d             return {};
ea5         double x = (r*r - c.r*c.r + d*d) / (2*d);
260         double y = sqrt(r*r - x*x);
5e0         point v = (c.o - o) / d;
645         inter.pb(o + v*x + v.rotate(cw90)*y);
c66         if(y > EPS) inter.pb(o + v*x + v.rotate(ccw90)*y);
c17         return inter;
945     }
196     vector<point> tang(point p){
903         double d = sqrt((p - o).norm2() - r*r);
164         return *this / circle(p, d);
15e     }
fb6     bool in(circle c){ // non strictly inside
6ac         double d = (o - c.o).norm();
ee4         return d + r < c.r + EPS;
5fd     }
0f4 };
```

8.2 Convex Hull

```
// Returns in CCW order (reversed in x in UPPER)
// Complexity: O(NlogN)

9c0 template <bool UPPER>
6d8 vector<point> hull(vector<point> v) {
805     vector<point> res;
6cd     if(UPPER) for(auto& p: v) p.x = -p.x, p.y = -p.y;
304     sort(all(v));
3f5     for(auto& p: v) {
1e7         if(res.empty()) { res.push_back(p); continue; }
89e         if(res.back().x == p.x) continue;
ca3         while(res.size() >= 2) {
dd1             point a = res[res.size()-2], b = res.back();
039             if(!left(a, b, p)) res.pop_back();
                //to include collinear points
                //if(right(a, b, p)) res.pop_back();
f97             else break;
d33         }
6f7         res.push_back(p);
806     }
96b     if(UPPER) for(auto& p: res) p.x = -p.x, p.y = -p.y;
b50     return res;
72a }
```

8.3 Double geometry

```
ad8 constexpr double EPS = 1e-10;

664 bool zero(double x) {
efc     return abs(x) <= EPS;
e8f }

// CORNER: point = (0, 0)
be5 struct point {
662     double x, y;

5cb     point(): x(), y() {}
581     point(double _x, double _y): x(_x), y(_y) {}

587     point operator+(point rhs) { return point(x+rhs.x, y+rhs.y); }
2f1     point operator-(point rhs) { return point(x-rhs.x, y-rhs.y); }
df3     point operator*(double k) { return point(x*k, y*k); }
d22     point operator/(double k) { return point(x/k, y/k); }
027     double operator*(point rhs) { return x*rhs.x + y*rhs.y; }
c47     double operator^(point rhs) { return x*rhs.y - y*rhs.x; }

aa4     point rotated(point polar) { return
    point(*this^polar,*this*polar); }
b9a     point rotated(double ang) { return
    (*this).rotated(point(sin(ang),cos(ang))); }
b7c     double norm2() { return *this * *this; }
b3a     double norm() { return sqrt(norm2()); }

5fa     bool operator<(const point& rhs) const {
70b         return x < rhs.x - EPS || (zero(x-rhs.x) && y < rhs.y -
    EPS);
f87     }

bfa     bool operator==(const point& rhs) const {
d38         return zero(x-rhs.x) && zero(y-rhs.y);
4f7     }
71f };

e17 const point ccw90(1, 0), cw90(-1, 0);

// angular comparison in [0, 2pi)
// smallest is (1, 0)
// CORNER: a || b == (0, 0)
a43 bool ang_cmp(point a, point b) {
b41     auto quad = [](point p) -> bool {
            // 0 if ang in [0, pi), 1 if in [pi, 2pi)
cfb         return p.y < 0 || (p.y == 0 && p.x < 0);
428     };
028     using tup = tuple<bool, double>;
dab     return tup{quad(a), 0} < tup{quad(b), a^b};
7d8 }

b5e double dist2(point p, point q) { // squared distance
f70     return (p - q)*(p - q);
60f }

cf4 double dist(point p, point q) {
d92     return sqrt(dist2(p, q));
a75 }

70f double area2(point a, point b, point c) { // two times signed area
    of triangle abc
b44     return (b - a) ^ (c - a);
556 }

97b bool left(point a, point b, point c) {
f3e     return area2(a, b, c) > EPS; // counterclockwise
483 }

18a bool right(point a, point b, point c) {
682     return area2(a, b, c) < -EPS; // clockwise
cc2 }

62c bool collinear(point a, point b, point c) {
56f     return zero(area2(a,b,c));
16b }

// CORNER: a || b == (0, 0)
e00 int parallel(point a, point b) {
046     if(!zero(a ^ b)) return 0;
8bb     return (a.x>0) == (b.x>0) && (a.y > 0) == (b.y > 0) ? 1 : -1;
e6c }

// CORNER: a == b
565 struct segment {
393     point a, b;

889     segment() {}
e93     segment(point _a, point _b): a(_a), b(_b) {}

988     point v() { return b - a; }

1d6 };

5db bool contains(segment r, point p) {
9c1     return r.a==p || r.b==p || parallel(r.a-p, r.b-p) == -1;
12b }

e58 bool intersects(segment r, segment s) {
2fb     if(contains(r, s.a) || contains(r, s.b) || contains(s, r.a) ||
    contains(s, r.b)) return 1;
9ff     return left(r.a, r.b, s.a) != left(r.a, r.b, s.b) &&
0a2         left(s.a, s.b, r.a) != left(s.a, s.b, r.b);
3dc }

6cc bool parallel(segment r, segment s) {
260     return parallel(r.v(), s.v());
bef }

737 point line_intersection(segment r, segment s) {
2de     if(parallel(r, s)) return point(HUGE_VAL, HUGE_VAL);
a80     point vr = r.v(), vs = s.v();
68c     double cr = vr ^ r.a, cs = vs ^ s.a;
47e     return (vs*cr - vr*cs) / (vr ^ vs);
243 }

694 point proj(segment r, point p) {
3cd     p = p - r.a;
1a5     point v = r.v();
607     return r.a + v*((p*v)/(v*v));
4f2 }

d2f struct polygon {
768     vector<point> vp;
1a8     int n;
```

```
66a     polygon(vector<point>& _vp): vp(_vp), n(vp.size()) {}

a2f     int nxt(int i) { return i+1<n ? i+1 : 0; }
6af     int prv(int i) { return i ? i-1 : 0; }

        // If positive, the polygon is in ccw order. It is in cw order otherwise.
720     double orientation() { // O(n
745         int acum = 0;
830         for(int i = 0; i < n; i++)
159             acum += vp[i] ^ vp[nxt(i)];
a13         return acum;
587     }

0d8     double area2() { // O(n)
64e         return abs(orientation());
355     }

9b0     void turnCcw() { // O(n)
057         if(orientation() < -EPS) reverse(all(vp));
7ba     }

223     bool has(point p) { // O(log n). The polygon must be convex and in ccw order
947         if(right(vp[0], vp[1], p) || left(vp[0], vp[n-1], p)) return 0;
9da         int lo = 1, hi = n;
3d1         while(lo + 1 < hi) {
c86             int mid = (lo + hi) / 2;
395             if(!right(vp[0], vp[mid], p)) lo = mid;
8c0             else hi = mid;
a27         }
b27         return hi != n ? !right(vp[lo], vp[hi], p) : dist2(vp[0], p) < dist2(vp[0], vp[n-1]) + EPS;
8fe     }

8d5     double calipers() { // O(n). The polygon must be convex and in ccw order.
e9c         double ans = 0;
1ed         for(int i = 0, j = 1; i < n; i++) {
d97             point v = vp[nxt(i)] - vp[i];
d5f             while((v ^ (vp[nxt(j)] - vp[j])) > EPS) j = nxt(j);
e88             ans = max(ans, dist2(vp[i], vp[j])); // Example with polygon diameter squared
121         }
ba7         return ans;
63b     }

8ff     int extreme(const function<bool(point, point)> &cmp) {
9b0         auto isExtreme = [&](int i, bool& curDir) -> bool {
a46             curDir = cmp(vp[nxt(i)], vp[i]);
f40             return !cmp(vp[prv(i)], vp[i]) && !curDir;
cb5         };
1a0         bool lastDir, curDir;
c7c         if(isExtreme(0, lastDir)) return 0;
a04         int lo = 0, hi = n;
3d1         while(lo + 1 < hi) {
591             int m = (lo + hi) / 2;
b60             if(isExtreme(m, curDir)) return m;
254             bool relDir = cmp(vp[m], vp[lo]);
729             if((!lastDir && curDir) || (lastDir == curDir && relDir == curDir)) {
04a                 lo = m;
986                 lastDir = curDir;
58b             } else hi = m;
5cb         }
253         return lo;
298     }

6fb     pair<int, int> tangent(point p) { // O(log n) for convex polygon in ccw orientation
            // Finds the indices of the two tangents to an external point q
f2d         auto leftTangent = [&](point r, point s) -> bool {
f70             return right(p, r, s);
5f6         };
29e         auto rightTangent = [&](point r, point s) -> bool {
f88             return left(p, r, s);
2b2         };
f49         return {extreme(leftTangent), extreme(rightTangent)};
f00     }

a9e     int maximize(point v) { // O(log n) for convex polygon in ccw orientation
            // Finds the extreme point in the direction of the vector
db6         return extreme([&](point p, point q) {return p * v > q * v + EPS;});
f05     }

df5     void normalize() { // p[0] becomes the lowest leftmost point
b2f         rotate(vp.begin(), min_element(all(vp)), vp.end());
7e8     }
0da     polygon operator+(polygon& rhs) { // Minkowsky sum
244         vector<point> sum;
335         normalize();
61f         rhs.normalize();
ccc         double dir;
337         for(int i = 0, j = 0; i < n || j < rhs.n; i += dir > -EPS,
    j += dir < EPS) {
c6f             sum.push_back(vp[i % n] + rhs.vp[j % rhs.n]);
727             dir = (vp[(i + 1) % n] - vp[i % n])
59c                 ^ (rhs.vp[(j + 1) % rhs.n] - rhs.vp[j % rhs.n]);
d98         }
6b4         return polygon(sum);
e1f     }
494 };
```

8.4 Half-plane intersection

```
// empty or a convex polygon (maybe degenerated). This template
    depends on double.cpp
//
// h - (input) set of half-planes to be intersected. Each half-plane
    is described as a pair
// of points such that the half-plane is at the left of them.
// pol - the intersection of the half-planes as a vector of points. If
    not empty, these
// points describe the vertices of the resulting polygon in clock-wise
    order.
// WARNING: Some points of the polygon might be repeated. This may be
    undesirable in some
// cases but it's useful to distinguish between empty intersections
    and degenerated
// polygons (such as a point, line, segment or half-line).
//
// Time complexity: O(n logn)

7a9 struct halfplane: public segment {
fe9     double ang;
077     halfplane() {}
7c9     halfplane(point _a, point _b) {
cab         a = _a; b = _b;
a36         ang = atan2(v().y, v().x);
461     }
535     bool operator <(const halfplane& rhs) const {
287         if (fabsl(ang - rhs.ang) < EPS) return right(a, b, rhs.a);
004             return ang < rhs.ang;
576     }
3b2     bool operator ==(const halfplane& rhs) const {
a0f         return fabs(ang - rhs.ang) < EPS;
745     }
83c     bool out(point r) {
ad7         return right(a, b, r);
6ae     }
485 };

7d1 constexpr double INF = 1e19;
0cd vector<point> hp_intersect(vector<halfplane> h) {
a85     array<point, 4> box = {
765         point(-INF, -INF),
822         point(INF, -INF),
ac0         point(INF, INF),
006         point(-INF, INF),
9bb     };
c63     for(int i = 0; i < 4; i++)
e4b         h.emplace_back(box[i], box[(i+1) % 4]);
d77     sort(all(h));
b1b     h.resize(unique(all(h)) - h.begin());
ff6     deque<halfplane> dq;

c76     auto sz = [&]() -> int { return dq.size(); };

6e3     for(auto hp: h) {
673         while(sz() > 1 && hp.out(line_intersection(dq.back(),
    dq[sz() - 2])))
c70             dq.pop_back();
70c         while(sz() > 1 && hp.out(line_intersection(dq[0], dq[1])))
c68             dq.pop_front();
1d5         dq.push_back(hp);
34d     }
a26     while(sz() > 2 && dq[0].out(line_intersection(dq.back(),
    dq[sz() - 2])))
c70         dq.pop_back();
430     while(sz() > 2 && dq.back().out(line_intersection(dq[0],
    dq[1])))
c68         dq.pop_front();
040     if(sz() < 3) return {};
e5f     vector<point> pol(sz());
21d     for(int i = 0; i < sz(); i++) {
3bb         pol[i] = line_intersection(dq[i], dq[(i+1) % sz()]);
39e     }
b22     return pol;
7c5 }
```

8.5 Integer Geometry

```
8d0 bool zero(int x) {
5db     return x == 0;
9b6 }

// CORNER: point = (0, 0)
be5 struct point {
e91     int x, y;

5cb     point(): x(), y() {}
4b6     point(int _x, int _y): x(_x), y(_y) {}

587     point operator+(point rhs) { return point(x+rhs.x, y+rhs.y); }
2f1     point operator-(point rhs) { return point(x-rhs.x, y-rhs.y); }
f24     int operator*(point rhs) { return x*rhs.x + y*rhs.y; }
55a     int operator^(point rhs) { return x*rhs.y - y*rhs.x; }

950     int norm2() { return *this * *this; }

e1c     using tup = tuple<int, int>;

5fa     bool operator<(const point& rhs) const {
046         return tup{x, y} < tup{rhs.x, rhs.y};
4a4     }

bfa     bool operator==(const point& rhs) const {
024         return tup{x, y} == tup{rhs.x, rhs.y};
77f     }
5ad };

// angular comparison in [0, 2pi)
// smallest is (1, 0)
// CORNER: a || b == (0, 0)
a43 bool ang_cmp(point a, point b) {
b41     auto quad = [](point p) -> bool {
            // 0 if ang in [0, pi), 1 if in [pi, 2pi)
cfb         return p.y < 0 || (p.y == 0 && p.x < 0);
428     };
c41     using tup = tuple<bool, int>;
dab     return tup{quad(a), 0} < tup{quad(b), a^b};
401 }

4c6 int dist2(point p, point q) { // squared distance
f70     return (p - q)*(p - q);
288 }

5bf int area2(point a, point b, point c) { // two times signed area of
    triangle abc
b44     return (b - a) ^ (c - a);
214 }

97b bool left(point a, point b, point c) {
8a5     return area2(a, b, c) > 0; // counterclockwise
8fd }

18a bool right(point a, point b, point c) {
c85     return area2(a, b, c) < 0; // clockwise
ece }

62c bool collinear(point a, point b, point c) {
56f     return zero(area2(a,b,c));
16b }

// CORNER: a || b == (0, 0)
e00 int parallel(point a, point b) {
046     if(!zero(a ^ b)) return 0;
8bb     return (a.x>0) == (b.x>0) && (a.y > 0) == (b.y > 0) ? 1 : -1;
e6c }

// CORNER: a == b
565 struct segment {
393     point a, b;

877     segment(): a(), b() {}
e93     segment(point _a, point _b): a(_a), b(_b) {}

988     point v() { return b - a; }
a42 };

5db bool contains(segment r, point p) {
9c1     return r.a==p || r.b==p || parallel(r.a-p,r.b-p) == -1;
12b }

e58 bool intersects(segment r, segment s) {
2fb     if(contains(r, s.a) || contains(r, s.b) || contains(s, r.a) ||
    contains(s, r.b)) return 1;
9ff     return left(r.a,r.b,s.a) != left(r.a,r.b,s.b) &&
0a2         left(s.a, s.b, r.a) != left(s.a, s.b, r.b);
3dc }

6cc bool parallel(segment r, segment s) {
260     return parallel(r.v(), s.v());
```

```
bef }

d2f struct polygon {
768     vector<point> vp;
1a8     int n;

66a     polygon(vector<point>& _vp): vp(_vp), n(vp.size()) {}

a2f     int nxt(int i) { return i+1<n ? i+1 : 0; }
6af     int prv(int i) { return i ? i-1 : 0; }

        // If positive, the polygon is in ccw order. It is in cw order
            otherwise.
882     int orientation() { // O(n
745         int acum = 0;
830         for(int i = 0; i < n; i++)
159             acum += vp[i] ^ vp[nxt(i)];
a13         return acum;
ea7     }

82b     int area2() { // O(n)
64e         return abs(orientation());
eb3     }

9b0     void turnCcw() { // O(n)
3d8         if(orientation() < 0) reverse(all(vp));
6b2     }

223     bool has(point p) { // O(log n). The polygon must be convex
    and in ccw order
947         if(right(vp[0], vp[1], p) || left(vp[0], vp[n-1], p))
    return 0;
9da         int lo = 1, hi = n;
3d1         while(lo + 1 < hi) {
c86             int mid = (lo + hi) / 2;
395             if(!right(vp[0], vp[mid], p)) lo = mid;
8c0             else hi = mid;
a27         }
78d         return hi != n ? !right(vp[lo], vp[hi], p) : dist2(vp[0],
    p) <= dist2(vp[0], vp[n-1]);
aa8     }

be9     int calipers() { // O(n). The polygon must be convex and in
    ccw order.
1a4         int ans = 0;
1ed         for(int i = 0, j = 1; i < n; i++) {
d97             point v = vp[nxt(i)] - vp[i];
775             while((v ^ (vp[nxt(j)] - vp[j])) > 0) j = nxt(j);
e88             ans = max(ans, dist2(vp[i], vp[j])); // Example with
    polygon diameter squared
c95         }
ba7         return ans;
e14     }

8ff     int extreme(const function<bool(point, point)> &cmp) {
9b0         auto isExtreme = [&](int i, bool& curDir) -> bool {
a46             curDir = cmp(vp[nxt(i)], vp[i]);
f40             return !cmp(vp[prv(i)], vp[i]) && !curDir;
cb5         };
1a0         bool lastDir, curDir;
c7c         if(isExtreme(0, lastDir)) return 0;
a04         int lo = 0, hi = n;
3d1         while(lo + 1 < hi) {
591             int m = (lo + hi) / 2;
b60             if(isExtreme(m, curDir)) return m;
254             bool relDir = cmp(vp[m], vp[lo]);
729             if((!lastDir && curDir) || (lastDir == curDir &&
    relDir == curDir)) {
04a                 lo = m;
986                 lastDir = curDir;
58b             } else hi = m;
5cb         }
253         return lo;
298     }

6fb     pair<int, int> tangent(point p) { // O(log n) for convex
    polygon in ccw orientation
            // Finds the indices of the two tangents to an external
                point q
f2d         auto leftTangent = [&](point r, point s) -> bool {
f70             return right(p, r, s);
5f6         };
29e         auto rightTangent = [&](point r, point s) -> bool {
f88             return left(p, r, s);
2b2         };
f49         return {extreme(leftTangent), extreme(rightTangent)};
f00     }

a9e     int maximize(point v) { // O(log n) for convex polygon in ccw
    orientation
            // Finds the extreme point in the direction of the vector
003         return extreme([&](point p, point q) {return p * v > q *
    v;});
f56     }
df5     void normalize() { // p[0] becomes the lowest leftmost point
b2f         rotate(vp.begin(), min_element(all(vp)), vp.end());
7e8     }

0da     polygon operator+(polygon& rhs) { // Minkowsky sum
244         vector<point> sum;
335         normalize();
61f         rhs.normalize();
755         for(int i = 0, j = 0, dir; i < n || j < rhs.n; i += dir >=
    0, j += dir <= 0) {
c6f             sum.push_back(vp[i % n] + rhs.vp[j % rhs.n]);
727             dir = (vp[(i + 1) % n] - vp[i % n])
59c                 ^ (rhs.vp[(j + 1) % rhs.n] - rhs.vp[j % rhs.n]);
520         }
6b4         return polygon(sum);
f2a     }
b14 };
```

8.6 Nearest Points

```
// Complexity: O(NlogN)

505 template <typename C_T>
e26 C_T nearest_points(vector<point> v) {
695     using lim = numeric_limits<C_T>;
50a     C_T res = lim::max(), sq = sqrt((double)res);
304     sort(all(v));
6e3     for(int i=1;i<v.size();i++) if(v[i] == v[i-1]) return 0;
e54     auto by_y = [](const point& a, const point& b) {
c0c         using tup = tuple<C_T, C_T>;
1b4         return tup{a.y, a.x} < tup{b.y, b.x};
58e     };
aa9     queue<point> active;
252     set<point, decltype(by_y)> pts(by_y);
3f5     for(auto& p: v) {
c24         while(!active.empty() && p.x-active.front().x > sq) {
56c             pts.erase(active.front());
1a0             active.pop();
ab0         }
abd         auto it = pts.lower_bound({lim::min(), p.y-sq});
97f         while(it != pts.end() && it->y <= p.y + sq) {
6fc             C_T d = dist2(p, *it);
424             if(d < res) {
b9f                 res = d;
a2c                 sq = sqrt((double)res);
bc7             }
40d             it++;
16e         }
381         active.push(p);
aa4         pts.insert(p);
367     }
b50     return res;
558 }
```

8.7 Shamos Hoey

```
// SEGMENTOS N O DEVEM SER DEGENERADOS
//
// Checa se existem segmentos que se intersectam
// Complexidade: O(N logN)

4d0 bool shamos_hoey(vector<segment> seg) {
        // create sweep segment events {x, type, seg_id}
900     vector<tuple<point, bool, int>> ev;
071     for(int i=0; i<seg.size(); i++) {
035         if(seg[i].b < seg[i].a) swap(seg[i].a, seg[i].b);
4ed         ev.emplace_back(seg[i].a, 0, i);
d2a         ev.emplace_back(seg[i].b, 1, i);
3d7     }
075     sort(all(ev));
2e7     auto cmp = [](segment r, segment s) -> bool {
6c3         if(r.a == s.a) return left(r.a, r.b, s.b);
4c1         else if(r.a < s.a) return left(r.a, r.b, s.a);
8ec         else return !left(s.a, s.b, r.a);
6ab     };
91a     set<segment, decltype(cmp)> s(cmp);
2af     for(auto [_, b, id]: ev) {
4ea         segment at = seg[id];
22d         if(!b) {
8c2             auto nxt = s.lower_bound(at);
556             if((nxt != s.end() && intersects(*nxt, at))
0b1                 || (nxt != s.begin() && intersects(*prev(nxt),
at)))
6a5                     return 1;
9be             s.insert(at);
9d9         } else {
381             auto cur = s.find(at);
a98             if(cur != s.begin() && cur != s.end() &&
38b                     intersects(*prev(cur), *next(cur)))
6a5                 return 1;
50d             s.erase(at);
a83         }
c4b     }
bb3     return 0;
107 }
```

9 Extra

9.1 template.cpp

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

#define all(x) x.begin(), x.end()
#define int int64_t
#define pb push_back

void dbg_out() { cerr << endl; }
template <typename H, typename... T>
void dbg_out(H h, T... t) { cerr << ' ' << h; dbg_out(t...); }
#define dbg(...) { cerr << #__VA_ARGS__ << ':'; dbg_out(__VA_ARGS__); }

void solve() {
}

signed main(){
    ios::sync_with_stdio(false); cin.tie(0);
    solve();
}
```

9.2 hash.sh

```
# hash.sh
# Para usar (hash das linhas [l1, l2]):
# ./hash.sh arquivo.cpp l1 l2
# md5sum do hash.sh: 9cd1295ed4344001c20548b1d6eb55b2
#
# Hash acumulativo, linha por linha:
# for i in $(seq $2 $3); do
#   echo -n "$i "
#   sed -n $2','$i' p' $1 | cpp -dD -P -fpreprocessed | tr -d
    '[:space:]' | md5sum | cut -c-6
# done
sed -n $2','$3' p' $1 | cpp -dD -P -fpreprocessed | tr -d '[:space:]'
    | md5sum | cut -c-6
```

9.3 random.cpp

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

9.4 clock.cpp

```
// Clock
clock_t startTime = clock();
double getCurrentTime() {
    return (double)(clock() - startTime) / CLOCKS_PER_SEC;
}
```

9.5 pragma.cpp

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