

Descongelen a Victor Moreno

Contents

1	Estructuras de Datos	1
1.1	Unordered Map	1
1.2	Segment tree Recursivo	1
1.3	Segment Tree Iterativo	2
1.4	Segment Tree Lazy Recursivo	3
1.5	Segment Tree Lazy Iterativo	3
1.6	Rope	4
1.7	Ordered Set	5
1.8	Union Find	5
1.9	Segment Tree Persistente	5
1.10	Sparce Table	5
2	Walvet Tree	6
2.1	Trie	6
3	Strings	7
3.1	Aho Corasick	7
3.2	Hashing	7
3.3	KMP	7
3.4	Manacher	8
3.5	Suffix Automata	8
4	Geometria	8
5	Varios	21
5.1	Template	21
5.2	String a vector<int>	21
5.3	Generar permutaciones	21
5.4	2 Sat	22
5.5	Bits	22
5.6	Matrix	22
5.7	MO	23
5.8	PBS	23

1 Estructuras de Datos

1.1 Unordered Map

```
1 #include <ext/pb_ds/assoc_container.hpp>
2 using namespace __gnu_pbds;
3
4 struct custom_hash {
5     static uint64_t splitmix64(uint64_t x) {
6         // http://xorshift.di.unimi.it/splitmix64.c
7         x += 0x9e3779b97f4a7c15;
8         x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
9         x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
10        return x ^ (x >> 31);
11    }
12
13    size_t operator()(uint64_t x) const {
14        static const uint64_t FIXED_RANDOM = chrono::steady_clock::now()
15            .time_since_epoch().count();
16        return splitmix64(x + FIXED_RANDOM);
17    }
18 };
19
20 gp_hash_table<int, int, custom_hash> m1;
21
22 //Funcion count
23 m1.find(x)!=m1.end()
```

1.2 Segment tree Recursivo

```
1 %%
2 %% This is file '.tex',
3 %% generated with the docstrip utility.
4 %%
5 %% The original source files were:
6 %%
7 %% fileerr.dtx (with options: 'return')
8 %%
9 %% This is a generated file.
10 %%
11 %% The source is maintained by the LaTeX Project team and bug
12 %% reports for it can be opened at https://latex-project.org/bugs/
13 %% (but please observe conditions on bug reports sent to that address!)
```

```

14 %%
15 %%
16 %% Copyright (C) 1993–2020
17 %% The LaTeX3 Project and any individual authors listed elsewhere
18 %% in this file.
19 %%
20 %% This file was generated from file(s) of the Standard LaTeX ‘Tools
    Bundle’.
21 %%
    -----
22 %%
23 %% It may be distributed and/or modified under the
24 %% conditions of the LaTeX Project Public License, either version 1.3c
25 %% of this license or (at your option) any later version.
26 %% The latest version of this license is in
27 %% https://www.latex-project.org/lppl.txt
28 %% and version 1.3c or later is part of all distributions of LaTeX
29 %% version 2005/12/01 or later.
30 %%
31 %% This file may only be distributed together with a copy of the LaTeX
32 %% ‘Tools Bundle’. You may however distribute the LaTeX ‘Tools Bundle’
33 %% without such generated files.
34 %%
35 %% The list of all files belonging to the LaTeX ‘Tools Bundle’ is
36 %% given in the file ‘manifest.txt’.
37 %%
38 \message{File ignored}
39 \endinput
40 %%
41 %% End of file ‘.tex’.

```

1.3 Segment Tree Iterativo

```

1 //Para procesar queries de tipo k-esimo es necesario crear un arbol
    binario perfector(llevar con 0's)
2 template<typename T>
3 struct SegmentTree{
4     int N;
5     vector<T> ST;
6
7     //Creacion a partir de un arreglo 0(n)
8     SegmentTree(int N, vector<T> & arr): N(N){

```

```

9     ST.resize(N << 1);
10    for(int i = 0; i < N; ++i)
11        ST[N + i] = arr[i]; //Dato normal
12    ST[N + i] = creaNode(); //Dato compuesto
13    for(int i = N - 1; i > 0; --i)
14        ST[i] = ST[i << 1] + ST[i << 1 | 1]; //Dato normal
15        ST[i] = merge(ST[i << 1], ST[i << 1 | 1]); //Dato compuesto
16    }
17
18    //Actualizacion de un elemento en la posicion i
19    void update(int i, T value){
20        ST[i += N] = value; //Dato normal
21        ST[i += N] = creaNode(); //Dato compuesto
22        while(i >= 1)
23            ST[i] = ST[i << 1] + ST[i << 1 | 1]; //Dato normal
24            ST[i] = merge(ST[i << 1], ST[i << 1 | 1]); //Dato compuesto
25        }
26
27    //query en [l, r]
28    T query(int l, int r){
29        T res = 0; //Dato normal
30        nodo resl = creaNode(), resr = creaNode(); //Dato compuesto
31        for(l += N, r += N; l <= r; l >= 1, r >= 1){
32            if(l & 1) res += ST[l++]; //Dato normal
33            if(!(r & 1)) res += ST[r--]; //Dato normal
34
35            if(l & 1) resl = merge(resl, ST[l++]); //Dato compuesto
36            if(!(r & 1)) resr = merge(ST[r--], resr); //Dato compuesto
37        }
38        return res; //Dato normal
39        return merge(resl, resr); //Dato compuesto
40    }
41
42    //Para estas queries es necesario que el st tenga el tam de la
        siguiente potencia de 2
43    //ll nT = 1;
44    // while(nT<n) nT<=1;
45    //vector<int> a(nT,0);
46
47    //Encontrar k-esimo 1 en un st de 1's
48    int Kth_One(int k) {
49        int i = 0, s = N >> 1;
50        for(int p = 2; p < 2 * N; p <= 1, s >= 1) {

```

```

51     if(k < ST[p]) continue;
52     k -= ST[p++]; i += s;
53 }
54 return i;
55 }
56
57 //i del primer elemento >= k en todo el arr
58 int atLeastX(int k){
59     int i = 0, s = N >> 1;
60     for(int p = 2; p < 2 * N; p <= 1, s >>= 1) {
61         if(ST[p] < k) p++, i += s;
62     }
63     if(ST[N + i] < k) i = -1;
64     return i;
65 }
66
67 //i del primer elemento >= k en [l,fin]
68 //Uso atLeastX(k,l,1,nT)
69 int atLeastX(int x, int l, int p, int s) {
70     if(ST[p] < x or s <= 1) return -1;
71     if((p < 1) >= 2 * N)
72         return (ST[p] >= x) - 1;
73     int i = atLeastX(x, l, p < 1, s >> 1);
74     if(i != -1) return i;
75     i = atLeastX(x, l - (s >> 1), p < 1 | 1, s >> 1);
76     if(i == -1) return -1;
77     return (s >> 1) + i;
78 }
79 };

```

1.4 Segment Tree Lazy Recursivo

```

1 %%
2 %% This is file '.tex',
3 %% generated with the docstrip utility.
4 %%
5 %% The original source files were:
6 %%
7 %% fileerr.dtx (with options: 'return')
8 %%
9 %% This is a generated file.
10 %%
11 %% The source is maintained by the LaTeX Project team and bug

```

```

12 %% reports for it can be opened at https://latex-project.org/bugs/
13 %% (but please observe conditions on bug reports sent to that address!)
14 %%
15 %%
16 %% Copyright (C) 1993-2020
17 %% The LaTeX3 Project and any individual authors listed elsewhere
18 %% in this file.
19 %%
20 %% This file was generated from file(s) of the Standard LaTeX 'Tools
   Bundle'.
21 %%
   -----
22 %%
23 %% It may be distributed and/or modified under the
24 %% conditions of the LaTeX Project Public License, either version 1.3c
25 %% of this license or (at your option) any later version.
26 %% The latest version of this license is in
27 %% https://www.latex-project.org/lppl.txt
28 %% and version 1.3c or later is part of all distributions of LaTeX
29 %% version 2005/12/01 or later.
30 %%
31 %% This file may only be distributed together with a copy of the LaTeX
32 %% 'Tools Bundle'. You may however distribute the LaTeX 'Tools Bundle'
33 %% without such generated files.
34 %%
35 %% The list of all files belonging to the LaTeX 'Tools Bundle' is
36 %% given in the file 'manifest.txt'.
37 %%
38 \message{File ignored}
39 \endinput
40 %%
41 %% End of file '.tex'.

```

1.5 Segment Tree Lazy Iterativo

```

1 //Lazy propagation con incremento de u en rango y minimo
2 //Hay varias modificaciones necesarias para suma en ambos
3 template<typename T>
4 struct SegmentTreeLazy{
5     int N,h;
6     vector<T> ST, d;
7

```

```

8 //Creacion a partir de un arreglo
9 SegmentTreeLazy(int n, vector<T> &a): N(n){
10     //En caso de inicializar en cero o algo similar, revisar que la
        construccion tenga su respectivo neutro mult y 1
11     ST.resize(N << 1);
12     d.resize(N);
13     h = 64 - __builtin_clzll(n);
14
15     for(int i = 0; i < N; ++i)
16         ST[N + i] = a[i];
17     //Construir el st sobre la query que se necesita
18     for(int i = N - 1; i > 0; --i)
19         ST[i] = min(ST[i << 1] , ST[i << 1 | 1]);
20 }
21
22 //Modificar de acuerdo al tipo modificacion requerida, +,*,|,^,etc
23 void apply(int p, T value) {
24     ST[p] += value;
25     if(p<N) d[p]+= value;
26 }
27
28 // Modifica valores de los padres de p
29 //Modificar de acuerdo al tipo modificacion requerida, +,*,|,^,etc y a
        la respectiva query
30 void build(int p){
31     while(p>1){
32         p >>= 1;
33         ST[p] = min(ST[p << 1], ST[p << 1 | 1]) + d[p];
34         //ST[p] = (ST[p << 1] & ST[p << 1 | 1]) | d[p]; Ejemplos con
            bitwise
35     }
36 }
37
38 // Propagacion desde la raiz a p
39 void push(int p){
40     for (int s = h; s > 0; --s) {
41         int i = p >> s;
42         if (d[i] != 0) {
43             apply(i << 1, d[i]);
44             apply(i << 1 | 1, d[i]);
45             d[i] = 0; //Tener cuidado si estoy haciendo multiplicaciones
46         }
47     }

```

```

48 }
49
50 // Sumar v a cada elemento en el intervalo [l, r)
51 void increment(int l, int r, T value) {
52     l += N, r += N;
53     int l0 = l, r0 = r;
54     for (; l < r; l >>= 1, r >>= 1) {
55         if(l & 1) apply(l++, value);
56         if(r & 1) apply(--r, value);
57     }
58     build(l0);
59     build(r0 - 1);
60 }
61
62 // min en el intervalo [l, r)
63 T range_min(int l, int r) {
64     l += N, r += N;
65     push(l);
66     push(r - 1);
67     T res = LLONG_MAX;
68     //T res = (1 << 30) - 1; Requiere operacion and
69     for (; l < r; l >>= 1, r >>= 1) {
70         if(l & 1) res = min(res, ST[l++]);
71         //if(res >= mod) res -= mod;
72         if(r & 1) res = min(res, ST[--r]);
73         //if(res >= mod) res -= mod;
74     }
75     return res;
76 }
77
78 };

```

1.6 Rope

```

1 #include <ext/rope>
2 using namespace __gnu_cxx;
3 rope<int> s;
4 // Sequence with O(log(n)) random access, insert, erase at any position
5 // s.push_back(x);
6 // s.insert(i,r) // insert rope r at position i
7 // s.erase(i,k) // erase subsequence [i,i+k)
8 // s.substr(i,k) // return new rope corresponding to subsequence [i,i+k)
9 // s[i] // access ith element (cannot modify)

```

```

10 // s.mutable_reference_at(i) // acces ith element (allows modification)
11 // s.begin() and s.end() are const iterators (use mutable_begin(),
    mutable_end() to allow modification)

```

1.7 Ordered Set

```

1 #include<ext/pb_ds/assoc_container.hpp>
2 #include<ext/pb_ds/tree_policy.hpp>
3 using namespace __gnu_pbds;
4 typedef tree<int,null_type,less<int>,rb_tree_tag,
    tree_order_statistics_node_update> ordered_set;
5 // find_by_order(i) -> iterator to ith element
6 // order_of_key(k) -> position (int) of lower_bound of k

```

1.8 Union Find

```

1 vector<pair<int,int>>ds(MAX,{-1,0});
2 // Solo siu requieres los elementos del union find, utiliza
3 // dsext en caso contrario borrarlo
4 list<int>dsext[MAX];
5 void init(int n){
6     for(int i=0;i<n;i++)dsext[i].push_back(i);
7 }
8 int find(int x){
9     if(-1==ds[x].first) return x;
10    return ds[x].first=find(ds[x].first);
11 }
12 bool unionDs(int x, int y){
13     int px=find(x),py=find(y);
14     int &rx=ds[px].second,&ry=ds[py].second;
15     if(px==py) return false;
16     else{
17         if(rx>ry){
18             ds[py].first=px;
19         }
20         else{
21             ds[px].first=py;
22             if(rx==ry) ry+=1;
23         }
24     }
25     return true;
26 }

```

1.9 Segment Tree Persistente

```

1 #define inf INT_MAX
2 const int MAX=5e5+2;
3 typedef pair<ll, ll> item;
4 struct node{
5     item val;
6     node *l, *r;
7     node(): l(nullptr),r(nullptr),val({inf,inf}){};
8     node(node *_l,node *_r):l(_l),r(_r){
9         val=min(l->val,r->val);
10    }
11    node(ll value,ll pos):r(nullptr),l(nullptr){
12        val=make_pair(value,pos);
13    }
14 };
15 pair<ll,ll>all;
16 vector<node*>versions(MAX,nullptr);
17 node* build(int l,int r){
18     if(l==r)return new node(inf,l);
19     int m=(l+r)/2;
20     return new node(build(l,m),build(m+1,r));
21 }
22
23 node* update(node *root,int l,int r,int pos,int val){
24     if(l==r){
25         return new node(val,pos);}
26     int m=(l+r)/2;
27     if(pos<=m) return new node(update(root->l,l,m,pos,val),root->r);
28     return new node(root->l,update(root->r,m+1,r,pos,val));
29 }
30 item query(node *root,int l,int r,int a,int b){
31     if(a>r || b<l) return all;
32     if(a<=l && r<=b) return root->val;
33     int m=(l+r)/2;
34     return min(query(root->l,l,m,a,b),query(root->r,m+1,r,a,b));
35 }

```

1.10 Sparce Table

```

1 //Se usa para RMQ porque se puede hacer en O(1), no acepta updates
2 vector<int>lg;
3 vector<vector<int>>>st;
4 int *nums;
5 void init(int n){

```

```

6   int logn=(int) log2(n)+1;
7   lg.assign(n+1,0);
8   st.assign(logn,vector<int>(n+1));
9   for(int i=0;i<n;i++) st[0][i]=nums[i];
10  lg[1]=0;
11  for(int i=2;i<=n;i++) lg[i]=lg[i/2]+1;
12  for(int i=1;i<logn;i++)
13      for(int j=0;j+(1<<i)<n;j++)st[i][j]=min(st[i-1][j],st[i-1][j
        +(1<<(i-1))]);
14 }
15 int query(int a,int b){
16     int logn=lg[(b-a+1)];
17     cout<<st[logn][a]<<endl;
18     return min(st[logn][a],st[logn][b-(1<<logn)+1]);
19 }

```

2 Walvet Tree

```

1 // indexed in 1
2 // from pointer to first element and to to end
3 // x and y The minimum element and y the max element
4 // If you need only one function or more erase the others
5 // If you need tu construct other function you only required to
   undertand the limit, this
6 // are the same
7 struct wavelet_tree{
8     int lo, hi;
9     wavelet_tree *l, *r;
10    vector<int> b;
11    wavelet_tree(int *from, int *to, int x, int y){
12        lo = x, hi = y;
13        if(lo == hi or from >= to) return;
14        int mid = (lo+hi)/2;
15        auto f = [mid](int x){ return x <= mid;};
16        b.reserve(to-from+1);
17        b.pb(0);
18        for(auto it = from; it != to; it++)
19            b.push_back(b.back() + f(*it));
20        auto pivot = stable_partition(from, to, f);
21        l = new wavelet_tree(from, pivot, lo, mid);
22        r = new wavelet_tree(pivot, to, mid+1, hi);
23    }
24    //kth smallest element in [l, r]

```

```

25 int kth(int l, int r, int k){
26     if(l > r) return 0;
27     if(lo == hi) return lo;
28     int inLeft = b[r] - b[l-1];
29     int lb = b[l-1];
30     int rb = b[r];
31     if(k <= inLeft) return this->l->kth(lb+1, rb , k);
32     return this->r->kth(l-lb, r-rb, k-inLeft);
33 }
34 //count of nos in [l, r] Less than or equal to k
35 int LTE(int l, int r, int k) {
36     if(l > r or k < lo) return 0;
37     if(hi <= k) return r - l + 1;
38     int lb = b[l-1], rb = b[r];
39     return this->l->LTE(lb+1, rb, k) + this->r->LTE(l-lb, r-rb, k);
40 }
41 //count of nos in [l, r] equal to k
42 int count(int l, int r, int k) {
43     if(l > r or k < lo or k > hi) return 0;
44     if(lo == hi) return r - l + 1;
45     int lb = b[l-1], rb = b[r], mid = (lo+hi)/2;
46     if(k <= mid) return this->l->count(lb+1, rb, k);
47     return this->r->count(l-lb, r-rb, k);
48 }
49 };

```

2.1 Trie

```

1 struct trie{
2     int len,id;
3     int children[26];
4     trie(int _id){
5         len=0,id=_id;
6         for(int i=0;i<26;i++)children[i]=-1;
7     }
8 };vector<trie>Trie;Trie.push_back(trie());
9 void insertString(string str,int root){
10    int aux=root;
11    for(int i=0;i<str.size();i++){
12        int index=str[i]-'a';
13        if(Trie[aux].children[index]==-1){
14            Trie.push_back(trie(Trie.size()));
15            Trie[aux].children[index]=Trie.size()-1;

```

```

16     }
17     aux=Trie[aux].children[index];
18 }
19 Trie[aux].len=str.size();
20 }
21 bool existInTrie(string str,int root){
22     int aux=root;
23     for(int i=0;i<str.size();i++){
24         int index=str[i]-'a';
25         if(Trie[aux].children[index]==-1) return false;
26         aux=Trie[aux].children[index];
27     }
28     return Trie[aux].len;
29 }

```

3 Strings

3.1 Aho Corasick

```

1 int K, I = 1;
2 struct node {
3     int fail, ch[26] = {};
4     vector<int> lens;
5 } T[500005];
6
7 void add(string s) {
8     int x = 1;
9     for (int i = 0; i < s.size(); i++) {
10         if (T[x].ch[s[i] - 'a'] == 0)
11             T[x].ch[s[i] - 'a'] = ++I;
12         x = T[x].ch[s[i] - 'a'];
13     }
14     T[x].lens.PB(s.size());
15 }
16
17 void build() {
18     queue<int> Q;
19     int x = 1;
20     T[1].fail = 1;
21     for (int i = 0; i < 26; i++) {
22         if (T[x].ch[i])
23             T[T[x].ch[i]].fail = x, Q.push(T[x].ch[i]);
24         else

```

```

25         T[x].ch[i] = 1;
26     }
27     while (!Q.empty()) {
28         x = Q.front(); Q.pop();
29         for (int i = 0; i < 26; i++) {
30             if (T[x].ch[i])
31                 T[T[x].ch[i]].fail = T[T[x].fail].ch[i], Q.push(T[x].ch[
32                     i]);
33             else
34                 T[x].ch[i] = T[T[x].fail].ch[i];
35         }
36     }

```

3.2 Hashing

```

1 struct Hash{
2     const int mod=1e9+123;
3     const int p=257;
4     vector<int> prefix;
5     static vector<int> pow;
6     Hash(string str){
7         int n=str.size();
8         while(pow.size()<=n){
9             pow.push_back(1LL*pow.back()*p%mod);
10        }
11        vector<int> aux(n+1);
12        prefix=aux;
13        for(int i=0;i<n;i++){
14            prefix[i+1]=(prefix[i]+1LL*str[i]*pow[i])%mod;
15        }
16    }
17    inline int getHashInInerval(int i,int len,int MxPow){
18        int hashing=prefix[i+len]-prefix[i];
19        if(hashing<0) hashing+=mod;
20        hashing=1LL*hashing*pow[MxPow-(len+i-1)]%mod;
21        return hashing;
22    }
23 };
24 vector<int> Hash::pow{1};

```

3.3 KMP

```

1 vector<int> kmp(string s){

```

```

2   int n=s.size();
3   vector<int>pi(n);
4   for(int i=1;i<n;i++){
5       int j=pi[i-1];
6       while(j>0 && s[i]!=s[j])j=pi[j-1];
7       if(s[i]==s[j]) j++;
8       pi[i]=j;
9   }
10  return pi;
11 }

```

3.4 Manacher

```

1  vector<int> manacher_odd(string s) {
2      int n = s.size();
3      s = "$" + s + "^";
4      vector<int> p(n + 2);
5      int l = 1, r = 1;
6      for(int i = 1; i <= n; i++) {
7          p[i] = max(0, min(r - i, p[l + (r - i)]));
8          while(s[i - p[i]] == s[i + p[i]]) {
9              p[i]++;
10         }
11         if(i + p[i] > r) {
12             l = i - p[i], r = i + p[i];
13         }
14     }
15     return vector<int>(begin(p) + 1, end(p) - 1);
16 }
17 vector<int> manacher_even(string s){
18     string even;
19     for(auto c:s){
20         even+='#'+c;
21     }
22     even+='#';
23     return manacher_odd(even);
24 }

```

3.5 Suffix Automata

```

1  struct node{
2      map<char,int>edges;
3      int link,length,terminal=0;
4      node(int link,int length): link(link),length(length){};

```

```

5  };vector<node>sa;
6  // init in main with sa.push_back(node(-1,0));
7  int last=0;
8  // add one by one chars in order
9  void addChar(char s, int pos){
10     sa.push_back(node(0,pos+1));
11     int r=sa.size()-1;
12     int p=last;
13     while(p >= 0 && sa[p].edges.find(s) == sa[p].edges.end()) {
14         sa[p].edges[s] = r;
15         p = sa[p].link;
16     }
17     if(p != -1) {
18         int q = sa[p].edges[s];
19         if(sa[p].length + 1 == sa[q].length) {
20             sa[r].link = q;
21         } else {
22             sa.push_back(node(sa[q].link,sa[p].length+1));
23             sa[sa.size()-1].edges=sa[q].edges;
24             int qq = sa.size()-1;
25             sa[q].link = qq;
26             sa[r].link= qq;
27             while(p >= 0 && sa[p].edges[s] == q) {
28                 sa[p].edges[s] = qq;
29                 p = sa[p].link;
30             }
31         }
32     }
33     last = r;
34 }
35 // Not necessary functions
36 void findTerminals(){
37     int p = last;
38     while(p > 0) {
39         sa[p].terminal=1;
40         p = sa[p].link;
41     }
42 }

```

4 Geometria

```

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

```



```

42     && le(y, p.y));}
43     bool operator>(const point & p) const{return ge(x, p.x) || (eq(x, p.x)
44         && ge(y, p.y));}
45     bool half(const point & p) const{return le(p.cross(*this), 0) || (eq(p
46         .cross(*this), 0) && le(p.dot(*this), 0));}
47 };
48
49 istream &operator>>(istream &is, point & p){return is >> p.x >> p.y;}
50 ostream &operator<<(ostream &os, const point & p){return os << "(" << p.
51     x << ", " << p.y << ")";}
52
53 int sgn(ld x){
54     if(ge(x, 0)) return 1;
55     if(le(x, 0)) return -1;
56     return 0;
57 }
58
59 void polarSort(vector<point> & P, const point & o, const point & v){
60     //sort points in P around o, taking the direction of v as first angle
61     sort(P.begin(), P.end(), [&](const point & a, const point & b){
62         return point((a - o).half(v), 0) < point((b - o).half(v), (a - o).
63             cross(b - o));
64     });
65 }
66
67 bool pointInLine(const point & a, const point & v, const point & p){
68     //line a+tv, point p
69     return eq((p - a).cross(v), 0);
70 }
71
72 bool pointInSegment(const point & a, const point & b, const point & p){
73     //segment ab, point p
74     return pointInLine(a, b - a, p) && leq((a - p).dot(b - p), 0);
75 }
76
77 int intersectLinesInfo(const point & a1, const point & v1, const point &
78     a2, const point & v2){
79     //lines a1+tv1 and a2+tv2
80     ld det = v1.cross(v2);
81     if(eq(det, 0)){
82         if(eq((a2 - a1).cross(v1), 0)){
83             return -1; //infinity points
84         }else{
85             return 0; //no intersection
86         }
87     }
88     ld t1 = (a2 - a1).cross(v2);
89     ld t2 = v1.cross(a2 - a1);
90     if(t1 > 0 || t2 > 0) return 0;
91     ld t = t1 / det;
92     ld t2 = t2 / det;
93     if(t >= 0 && t2 >= 0) return 1;
94     return 0;
95 }
96
97 int intersectLines(const point & a1, const point & v1, const point & a2,
98     const point & v2){
99     return intersectLinesInfo(a1, v1, a2, v2) > 0;
100 }
101
102 int intersectLineSegment(const point & a1, const point & v1, const point &
103     a2, const point & b2){
104     return intersectLines(a1, v1, a2, b2 - a2) > 0;
105 }
106
107 int intersectSegment(const point & a1, const point & b1, const point & a2,
108     const point & b2){
109     return intersectLineSegment(a1, b1 - a1, a2, b2 - a2) > 0;
110 }
111
112 int intersect(const point & a1, const point & b1, const point & a2, const
113     point & b2){
114     return intersectSegment(a1, b1, a2, b2);
115 }
116
117 int intersect(const point & a1, const point & b1, const point & a2, const
118     point & b2, int & type){
119     type = intersectSegment(a1, b1, a2, b2);
120     return type > 0;
121 }
122
123 int intersect(const point & a1, const point & b1, const point & a2, const
124     point & b2, int & type, int & t1, int & t2){
125     type = intersectLineSegment(a1, b1, a2, b2);
126     if(type > 0){
127         ld t1 = (a2 - a1).cross(b1 - a1);
128         ld t2 = (b2 - a1).cross(b1 - a1);
129         ld det = (b2 - a1).cross(b1 - a1);
130         t1 = t1 / det;
131         t2 = t2 / det;
132     }
133     return type > 0;
134 }
135
136 int intersect(const point & a1, const point & b1, const point & a2, const
137     point & b2, int & type, int & t1, int & t2, int & t3, int & t4){
138     type = intersectSegment(a1, b1, a2, b2);
139     if(type > 0){
140         ld t1 = (a2 - a1).cross(b1 - a1);
141         ld t2 = (b2 - a1).cross(b1 - a1);
142         ld t3 = (a2 - a1).cross(b2 - a1);
143         ld t4 = (b2 - a1).cross(b1 - a1);
144         ld det = (b2 - a1).cross(b1 - a1);
145         t1 = t1 / det;
146         t2 = t2 / det;
147         t3 = t3 / det;
148         t4 = t4 / det;
149     }
150     return type > 0;
151 }
152
153 int intersect(const point & a1, const point & b1, const point & a2, const
154     point & b2, int & type, int & t1, int & t2, int & t3, int & t4, int &
155     t5, int & t6){
156     type = intersectSegment(a1, b1, a2, b2);
157     if(type > 0){
158         ld t1 = (a2 - a1).cross(b1 - a1);
159         ld t2 = (b2 - a1).cross(b1 - a1);
160         ld t3 = (a2 - a1).cross(b2 - a1);
161         ld t4 = (b2 - a1).cross(b1 - a1);
162         ld t5 = (a2 - a1).cross(b2 - a1);
163         ld t6 = (b2 - a1).cross(b1 - a1);
164         ld det = (b2 - a1).cross(b1 - a1);
165         t1 = t1 / det;
166         t2 = t2 / det;
167         t3 = t3 / det;
168         t4 = t4 / det;
169         t5 = t5 / det;
170         t6 = t6 / det;
171     }
172     return type > 0;
173 }
174
175 int intersect(const point & a1, const point & b1, const point & a2, const
176     point & b2, int & type, int & t1, int & t2, int & t3, int & t4, int &
177     t5, int & t6, int & t7, int & t8){
178     type = intersectSegment(a1, b1, a2, b2);
179     if(type > 0){
180         ld t1 = (a2 - a1).cross(b1 - a1);
181         ld t2 = (b2 - a1).cross(b1 - a1);
182         ld t3 = (a2 - a1).cross(b2 - a1);
183         ld t4 = (b2 - a1).cross(b1 - a1);
184         ld t5 = (a2 - a1).cross(b2 - a1);
185         ld t6 = (b2 - a1).cross(b1 - a1);
186         ld t7 = (a2 - a1).cross(b2 - a1);
187         ld t8 = (b2 - a1).cross(b1 - a1);
188         ld det = (b2 - a1).cross(b1 - a1);
189         t1 = t1 / det;
190         t2 = t2 / det;
191         t3 = t3 / det;
192         t4 = t4 / det;
193         t5 = t5 / det;
194         t6 = t6 / det;
195         t7 = t7 / det;
196         t8 = t8 / det;
197     }
198     return type > 0;
199 }
200
201 int intersect(const point & a1, const point & b1, const point & a2, const
202     point & b2, int & type, int & t1, int & t2, int & t3, int & t4, int &
203     t5, int & t6, int & t7, int & t8, int & t9, int & t10){
204     type = intersectSegment(a1, b1, a2, b2);
205     if(type > 0){
206         ld t1 = (a2 - a1).cross(b1 - a1);
207         ld t2 = (b2 - a1).cross(b1 - a1);
208         ld t3 = (a2 - a1).cross(b2 - a1);
209         ld t4 = (b2 - a1).cross(b1 - a1);
210         ld t5 = (a2 - a1).cross(b2 - a1);
211         ld t6 = (b2 - a1).cross(b1 - a1);
212         ld t7 = (a2 - a1).cross(b2 - a1);
213         ld t8 = (b2 - a1).cross(b1 - a1);
214         ld t9 = (a2 - a1).cross(b2 - a1);
215         ld t10 = (b2 - a1).cross(b1 - a1);
216         ld det = (b2 - a1).cross(b1 - a1);
217         t1 = t1 / det;
218         t2 = t2 / det;
219         t3 = t3 / det;
220         t4 = t4 / det;
221         t5 = t5 / det;
222         t6 = t6 / det;
223         t7 = t7 / det;
224         t8 = t8 / det;
225         t9 = t9 / det;
226         t10 = t10 / det;
227     }
228     return type > 0;
229 }
230
231 int intersect(const point & a1, const point & b1, const point & a2, const
232     point & b2, int & type, int & t1, int & t2, int & t3, int & t4, int &
233     t5, int & t6, int & t7, int & t8, int & t9, int & t10, int & t11, int &
234     t12){
235     type = intersectSegment(a1, b1, a2, b2);
236     if(type > 0){
237         ld t1 = (a2 - a1).cross(b1 - a1);
238         ld t2 = (b2 - a1).cross(b1 - a1);
239         ld t3 = (a2 - a1).cross(b2 - a1);
240         ld t4 = (b2 - a1).cross(b1 - a1);
241         ld t5 = (a2 - a1).cross(b2 - a1);
242         ld t6 = (b2 - a1).cross(b1 - a1);
243         ld t7 = (a2 - a1).cross(b2 - a1);
244         ld t8 = (b2 - a1).cross(b1 - a1);
245         ld t9 = (a2 - a1).cross(b2 - a1);
246         ld t10 = (b2 - a1).cross(b1 - a1);
247         ld t11 = (a2 - a1).cross(b2 - a1);
248         ld t12 = (b2 - a1).cross(b1 - a1);
249         ld det = (b2 - a1).cross(b1 - a1);
250         t1 = t1 / det;
251         t2 = t2 / det;
252         t3 = t3 / det;
253         t4 = t4 / det;
254         t5 = t5 / det;
255         t6 = t6 / det;
256         t7 = t7 / det;
257         t8 = t8 / det;
258         t9 = t9 / det;
259         t10 = t10 / det;
260         t11 = t11 / det;
261         t12 = t12 / det;
262     }
263     return type > 0;
264 }
265
266 int intersect(const point & a1, const point & b1, const point & a2, const
267     point & b2, int & type, int & t1, int & t2, int & t3, int & t4, int &
268     t5, int & t6, int & t7, int & t8, int & t9, int & t10, int & t11, int &
269     t12, int & t13, int & t14){
270     type = intersectSegment(a1, b1, a2, b2);
271     if(type &gt
```

```

79     return 0; //no points
80 }
81 }else{
82     return 1; //single point
83 }
84 }
85
86 point intersectLines(const point & a1, const point & v1, const point &
    a2, const point & v2){
87     //lines a1+tv1, a2+tv2
88     //assuming that they intersect
89     ld det = v1.cross(v2);
90     return a1 + v1 * ((a2 - a1).cross(v2) / det);
91 }
92
93 int intersectLineSegmentInfo(const point & a, const point & v, const
    point & c, const point & d){
94     //line a+tv, segment cd
95     point v2 = d - c;
96     ld det = v.cross(v2);
97     if(eq(det, 0)){
98         if(eq((c - a).cross(v), 0)){
99             return -1; //infinity points
100         }else{
101             return 0; //no point
102         }
103     }else{
104         return sgn(v.cross(c - a)) != sgn(v.cross(d - a)); //1: single point
105         , 0: no point
106     }
107 }
108
109 int intersectSegmentsInfo(const point & a, const point & b, const point
    & c, const point & d){
110     //segment ab, segment cd
111     point v1 = b - a, v2 = d - c;
112     int t = sgn(v1.cross(c - a)), u = sgn(v1.cross(d - a));
113     if(t == u){
114         if(t == 0){
115             if(pointInSegment(a, b, c) || pointInSegment(a, b, d) ||
                pointInSegment(c, d, a) || pointInSegment(c, d, b)){
116                 return -1; //infinity points
117             }else{

```

```

117         return 0; //no point
118     }
119     }else{
120         return 0; //no point
121     }
122     }else{
123         return sgn(v2.cross(a - c)) != sgn(v2.cross(b - c)); //1: single
            point, 0: no point
124     }
125 }
126
127 ld distancePointLine(const point & a, const point & v, const point & p){
128     //line: a + tv, point p
129     return abs(v.cross(p - a)) / v.length();
130 }
131
132 ld perimeter(vector<point> & P){
133     int n = P.size();
134     ld ans = 0;
135     for(int i = 0; i < n; i++){
136         ans += (P[i] - P[(i + 1) % n]).length();
137     }
138     return ans;
139 }
140
141 ld area(vector<point> & P){
142     int n = P.size();
143     ld ans = 0;
144     for(int i = 0; i < n; i++){
145         ans += P[i].cross(P[(i + 1) % n]);
146     }
147     return abs(ans / 2);
148 }
149
150 vector<point> convexHull(vector<point> P){
151     sort(P.begin(), P.end());
152     vector<point> L, U;
153     for(int i = 0; i < P.size(); i++){
154         while(L.size() >= 2 && leq((L[L.size() - 2] - P[i]).cross(L[L.size()
            - 1] - P[i]), 0)){
155             L.pop_back();
156         }
157         L.push_back(P[i]);

```

```

158 }
159 for(int i = P.size() - 1; i >= 0; i--){
160     while(U.size() >= 2 && leq((U[U.size() - 2] - P[i]).cross(U[U.size()
161         - 1] - P[i]), 0)){
162         U.pop_back();
163     }
164     U.push_back(P[i]);
165 }
166 L.pop_back();
167 U.pop_back();
168 L.insert(L.end(), U.begin(), U.end());
169 return L;
170 }
171 bool pointInPerimeter(const vector<point> & P, const point & p){
172     int n = P.size();
173     for(int i = 0; i < n; i++){
174         if(pointInSegment(P[i], P[(i + 1) % n], p)){
175             return true;
176         }
177     }
178     return false;
179 }
180
181 bool crossesRay(const point & a, const point & b, const point & p){
182     return (geq(b.y, p.y) - geq(a.y, p.y)) * sgn((a - p).cross(b - p)) >
183         0;
184 }
185
186 int pointInPolygon(const vector<point> & P, const point & p){
187     if(pointInPerimeter(P, p)){
188         return -1; //point in the perimeter
189     }
190     int n = P.size();
191     int rays = 0;
192     for(int i = 0; i < n; i++){
193         rays += crossesRay(P[i], P[(i + 1) % n], p);
194     }
195     return rays & 1; //0: point outside, 1: point inside
196 }
197 //point in convex polygon in O(log n)
198 //make sure that P is convex and in ccw

```

```

199 //before the queries, do the preprocess on P:
200 // rotate(P.begin(), min_element(P.begin(), P.end()), P.end());
201 // int right = max_element(P.begin(), P.end()) - P.begin();
202 //returns 0 if p is outside, 1 if p is inside, -1 if p is in the
    perimeter
203 int pointInConvexPolygon(const vector<point> & P, const point & p, int
    right){
204     if(p < P[0] || P[right] < p) return 0;
205     int orientation = sgn((P[right] - P[0]).cross(p - P[0]));
206     if(orientation == 0){
207         if(p == P[0] || p == P[right]) return -1;
208         return (right == 1 || right + 1 == P.size()) ? -1 : 1;
209     }else if(orientation < 0){
210         auto r = lower_bound(P.begin() + 1, P.begin() + right, p);
211         int det = sgn((p - r[-1]).cross(r[0] - r[-1])) - 1;
212         if(det == -2) det = 1;
213         return det;
214     }else{
215         auto l = upper_bound(P.rbegin(), P.rend() - right - 1, p);
216         int det = sgn((p - l[0]).cross((l == P.rbegin() ? P[0] : l[-1]) - l
217             [0])) - 1;
218         if(det == -2) det = 1;
219         return det;
220     }
221 }
222
223
224
225
226 vector<point> cutPolygon(const vector<point> & P, const point & a, const
    point & v){
227     //returns the part of the convex polygon P on the left side of line a +
    tv
228     int n = P.size();
229     vector<point> lhs;
230     for(int i = 0; i < n; ++i){
231         if(geq(v.cross(P[i] - a), 0)){
232             lhs.push_back(P[i]);
233         }
234         if(intersectLineSegmentInfo(a, v, P[i], P[(i+1)%n]) == 1){
235             point p = intersectLines(a, v, P[i], P[(i+1)%n] - P[i]);
236             if(p != P[i] && p != P[(i+1)%n]){

```

```

237     lhs.push_back(p);
238 }
239 }
240 }
241 return lhs;
242 }
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259 point centroid(vector<point> & P){
260     point num;
261     ld den = 0;
262     int n = P.size();
263     for(int i = 0; i < n; ++i){
264         ld cross = P[i].cross(P[(i + 1) % n]);
265         num += (P[i] + P[(i + 1) % n]) * cross;
266         den += cross;
267     }
268     return num / (3 * den);
269 }
270
271 vector<pair<int, int>> antipodalPairs(vector<point> & P){
272     vector<pair<int, int>> ans;
273     int n = P.size(), k = 1;
274     auto f = [&](int u, int v, int w){return abs((P[v%n]-P[u%n]).cross(P[w
        %n]-P[u%n]));};
275     while(ge(f(n-1, 0, k+1), f(n-1, 0, k))) ++k;
276     for(int i = 0, j = k; i <= k && j < n; ++i){
277         ans.emplace_back(i, j);
278         while(j < n-1 && ge(f(i, i+1, j+1), f(i, i+1, j)))

```

```

279         ans.emplace_back(i, ++j);
280     }
281     return ans;
282 }
283
284 pair<ld, ld> diameterAndWidth(vector<point> & P){
285     int n = P.size(), k = 0;
286     auto dot = [&](int a, int b){return (P[(a+1)%n]-P[a]).dot(P[(b+1)%n]-P
        [b]);};
287     auto cross = [&](int a, int b){return (P[(a+1)%n]-P[a]).cross(P[(b+1)%
        n]-P[b]);};
288     ld diameter = 0;
289     ld width = inf;
290     while(ge(dot(0, k), 0)) k = (k+1) % n;
291     for(int i = 0; i < n; ++i){
292         while(ge(cross(i, k), 0)) k = (k+1) % n;
293         //pair: (i, k)
294         diameter = max(diameter, (P[k] - P[i]).length());
295         width = min(width, distancePointLine(P[i], P[(i+1)%n] - P[i], P[k]));
296     }
297     return {diameter, width};
298 }
299
300 pair<ld, ld> smallestEnclosingRectangle(vector<point> & P){
301     int n = P.size();
302     auto dot = [&](int a, int b){return (P[(a+1)%n]-P[a]).dot(P[(b+1)%n]-P
        [b]);};
303     auto cross = [&](int a, int b){return (P[(a+1)%n]-P[a]).cross(P[(b+1)%
        n]-P[b]);};
304     ld perimeter = inf, area = inf;
305     for(int i = 0, j = 0, k = 0, m = 0; i < n; ++i){
306         while(ge(dot(i, j), 0)) j = (j+1) % n;
307         if(!i) k = j;
308         while(ge(cross(i, k), 0)) k = (k+1) % n;
309         if(!i) m = k;
310         while(le(dot(i, m), 0)) m = (m+1) % n;
311         //pairs: (i, k) , (j, m)
312         point v = P[(i+1)%n] - P[i];
313         ld h = distancePointLine(P[i], v, P[k]);
314         ld w = distancePointLine(P[j], v.perp(), P[m]);
315         perimeter = min(perimeter, 2 * (h + w));
316         area = min(area, h * w);

```

```

317 }
318 return {area, perimeter};
319 }
320
321 ld distancePointCircle(const point & c, ld r, const point & p){
322     //point p, circle with center c and radius r
323     return max((ld)0, (p - c).length() - r);
324 }
325
326 point projectionPointCircle(const point & c, ld r, const point & p){
327     //point p (outside the circle), circle with center c and radius r
328     return c + (p - c).unit() * r;
329 }
330
331 pair<point, point> pointsOfTangency(const point & c, ld r, const point &
    p){
332     //point p (outside the circle), circle with center c and radius r
333     point v = (p - c).unit() * r;
334     ld d2 = (p - c).norm(), d = sqrt(d2);
335     point v1 = v * (r / d), v2 = v.perp() * (sqrt(d2 - r*r) / d);
336     return {c + v1 - v2, c + v1 + v2};
337 }
338
339 vector<point> intersectLineCircle(const point & a, const point & v,
    const point & c, ld r){
340     //line a+tv, circle with center c and radius r
341     ld h2 = r*r - v.cross(c - a) * v.cross(c - a) / v.norm();
342     point p = a + v * v.dot(c - a) / v.norm();
343     if(eq(h2, 0)) return {p}; //line tangent to circle
344     else if(le(h2, 0)) return {}; //no intersection
345     else{
346         point u = v.unit() * sqrt(h2);
347         return {p - u, p + u}; //two points of intersection (chord)
348     }
349 }
350
351 vector<point> intersectSegmentCircle(const point & a, const point & b,
    const point & c, ld r){
352     //segment ab, circle with center c and radius r
353     vector<point> P = intersectLineCircle(a, b - a, c, r), ans;
354     for(const point & p : P){
355         if(pointInSegment(a, b, p)) ans.push_back(p);
356     }

```

```

357     return ans;
358 }
359
360 pair<point, ld> getCircle(const point & m, const point & n, const point
    & p){
361     //find circle that passes through points p, q, r
362     point c = intersectLines((n + m) / 2, (n - m).perp(), (p + n) / 2, (p
        - n).perp());
363     ld r = (c - m).length();
364     return {c, r};
365 }
366
367 vector<point> intersectionCircles(const point & c1, ld r1, const point &
    c2, ld r2){
368     //circle 1 with center c1 and radius r1
369     //circle 2 with center c2 and radius r2
370     point d = c2 - c1;
371     ld d2 = d.norm();
372     if(eq(d2, 0)) return {}; //concentric circles
373     ld pd = (d2 + r1*r1 - r2*r2) / 2;
374     ld h2 = r1*r1 - pd*pd/d2;
375     point p = c1 + d*pd/d2;
376     if(eq(h2, 0)) return {p}; //circles touch at one point
377     else if(le(h2, 0)) return {}; //circles don't intersect
378     else{
379         point u = d.perp() * sqrt(h2/d2);
380         return {p - u, p + u};
381     }
382 }
383
384 int circleInsideCircle(const point & c1, ld r1, const point & c2, ld r2)
    {
385     //test if circle 2 is inside circle 1
386     //returns "-1" if 2 touches internally 1, "1" if 2 is inside 1, "0" if
        they overlap
387     ld l = r1 - r2 - (c1 - c2).length();
388     return (ge(l, 0) ? 1 : (eq(l, 0) ? -1 : 0));
389 }
390
391 int circleOutsideCircle(const point & c1, ld r1, const point & c2, ld r2)
    {
392     //test if circle 2 is outside circle 1
393     //returns "-1" if they touch externally, "1" if 2 is outside 1, "0" if

```

```

        they overlap
394     ld l = (c1 - c2).length() - (r1 + r2);
395     return (ge(l, 0) ? 1 : (eq(l, 0) ? -1 : 0));
396 }
397
398 int pointInCircle(const point & c, ld r, const point & p){
399     //test if point p is inside the circle with center c and radius r
400     //returns "0" if it's outside, "-1" if it's in the perimeter, "1" if
        it's inside
401     ld l = (p - c).length() - r;
402     return (le(l, 0) ? 1 : (eq(l, 0) ? -1 : 0));
403 }
404
405 vector<vector<point>> tangents(const point & c1, ld r1, const point & c2
        , ld r2, bool inner){
406     //returns a vector of segments or a single point
407     if(inner) r2 = -r2;
408     point d = c2 - c1;
409     ld dr = r1 - r2, d2 = d.norm(), h2 = d2 - dr*dr;
410     if(eq(d2, 0) || le(h2, 0)) return {};
411     point v = d*dr/d2;
412     if(eq(h2, 0)) return {{c1 + v*r1}};
413     else{
414         point u = d.perp()*sqrt(h2)/d2;
415         return {{c1 + (v - u)*r1, c2 + (v - u)*r2}, {c1 + (v + u)*r1, c2 + (
            v + u)*r2}};
416     }
417 }
418
419 ld signed_angle(const point & a, const point & b){
420     return sgn(a.cross(b)) * acosl(a.dot(b) / (a.length() * b.length()));
421 }
422
423 ld intersectPolygonCircle(const vector<point> & P, const point & c, ld r
        ){
424     //Gets the area of the intersection of the polygon with the circle
425     int n = P.size();
426     ld ans = 0;
427     for(int i = 0; i < n; ++i){
428         point p = P[i], q = P[(i+1)%n];
429         bool p_inside = (pointInCircle(c, r, p) != 0);
430         bool q_inside = (pointInCircle(c, r, q) != 0);
431         if(p_inside && q_inside){

```

```

432         ans += (p - c).cross(q - c);
433     }else if(p_inside && !q_inside){
434         point s1 = intersectSegmentCircle(p, q, c, r)[0];
435         point s2 = intersectSegmentCircle(c, q, c, r)[0];
436         ans += (p - c).cross(s1 - c) + r*r * signed_angle(s1 - c, s2 - c);
437     }else if(!p_inside && q_inside){
438         point s1 = intersectSegmentCircle(c, p, c, r)[0];
439         point s2 = intersectSegmentCircle(p, q, c, r)[0];
440         ans += (s2 - c).cross(q - c) + r*r * signed_angle(s1 - c, s2 - c);
441     }else{
442         auto info = intersectSegmentCircle(p, q, c, r);
443         if(info.size() <= 1){
444             ans += r*r * signed_angle(p - c, q - c);
445         }else{
446             point s2 = info[0], s3 = info[1];
447             point s1 = intersectSegmentCircle(c, p, c, r)[0];
448             point s4 = intersectSegmentCircle(c, q, c, r)[0];
449             ans += (s2 - c).cross(s3 - c) + r*r * (signed_angle(s1 - c, s2 -
                c) + signed_angle(s3 - c, s4 - c));
450         }
451     }
452 }
453 return abs(ans)/2;
454 }
455
456 pair<point, ld> mec2(vector<point> & S, const point & a, const point & b
        , int n){
457     ld hi = inf, lo = -hi;
458     for(int i = 0; i < n; ++i){
459         ld si = (b - a).cross(S[i] - a);
460         if(eq(si, 0)) continue;
461         point m = getCircle(a, b, S[i]).first;
462         ld cr = (b - a).cross(m - a);
463         if(le(si, 0)) hi = min(hi, cr);
464         else lo = max(lo, cr);
465     }
466     ld v = (ge(lo, 0) ? lo : le(hi, 0) ? hi : 0);
467     point c = (a + b) / 2 + (b - a).perp() * v / (b - a).norm();
468     return {c, (a - c).norm()};
469 }
470
471 pair<point, ld> mec(vector<point> & S, const point & a, int n){
472     random_shuffle(S.begin(), S.begin() + n);

```

```

473 point b = S[0], c = (a + b) / 2;
474 ld r = (a - c).norm();
475 for(int i = 1; i < n; ++i){
476     if(ge((S[i] - c).norm(), r)){
477         tie(c, r) = (n == S.size() ? mec(S, S[i], i) : mec2(S, a, S[i], i)
478             );
479     }
480 }
481 return {c, r};
482 }
483 pair<point, ld> smallestEnclosingCircle(vector<point> S){
484     assert(!S.empty());
485     auto r = mec(S, S[0], S.size());
486     return {r.first, sqrt(r.second)};
487 }
488
489 bool comp1(const point & a, const point & b){
490     return le(a.y, b.y);
491 }
492 pair<point, point> closestPairOfPoints(vector<point> P){
493     sort(P.begin(), P.end(), comp1);
494     set<point> S;
495     ld ans = inf;
496     point p, q;
497     int pos = 0;
498     for(int i = 0; i < P.size(); ++i){
499         while(pos < i && geq(P[i].y - P[pos].y, ans)){
500             S.erase(P[pos++]);
501         }
502         auto lower = S.lower_bound({P[i].x - ans - eps, -inf});
503         auto upper = S.upper_bound({P[i].x + ans + eps, -inf});
504         for(auto it = lower; it != upper; ++it){
505             ld d = (P[i] - *it).length();
506             if(le(d, ans)){
507                 ans = d;
508                 p = P[i];
509                 q = *it;
510             }
511         }
512         S.insert(P[i]);
513     }
514     return {p, q};

```

```

515 }
516
517 struct vantage_point_tree{
518     struct node
519     {
520         point p;
521         ld th;
522         node *l, *r;
523     }*root;
524
525     vector<pair<ld, point>> aux;
526
527     vantage_point_tree(vector<point> &ps){
528         for(int i = 0; i < ps.size(); ++i)
529             aux.push_back({ 0, ps[i] });
530         root = build(0, ps.size());
531     }
532
533     node *build(int l, int r){
534         if(l == r)
535             return 0;
536         swap(aux[l], aux[l + rand() % (r - l)]);
537         point p = aux[l++].second;
538         if(l == r)
539             return new node({ p });
540         for(int i = l; i < r; ++i)
541             aux[i].first = (p - aux[i].second).dot(p - aux[i].second);
542         int m = (l + r) / 2;
543         nth_element(aux.begin() + l, aux.begin() + m, aux.begin() + r);
544         return new node({ p, sqrt(aux[m].first), build(l, m), build(m, r) });
545     }
546
547     priority_queue<pair<ld, node*>> que;
548
549     void k_nn(node *t, point p, int k){
550         if(!t)
551             return;
552         ld d = (p - t->p).length();
553         if(que.size() < k)
554             que.push({ d, t });
555         else if(ge(que.top().first, d)){
556             que.pop();

```



```

557     que.push({ d, t });
558 }
559 if(!t->l && !t->r)
560     return;
561 if(le(d, t->th)){
562     k_nn(t->l, p, k);
563     if(leq(t->th - d, que.top().first))
564         k_nn(t->r, p, k);
565 }else{
566     k_nn(t->r, p, k);
567     if(leq(d - t->th, que.top().first))
568         k_nn(t->l, p, k);
569 }
570 }
571
572 vector<point> k_nn(point p, int k){
573     k_nn(root, p, k);
574     vector<point> ans;
575     for(; !que.empty(); que.pop())
576         ans.push_back(que.top().second->p);
577     reverse(ans.begin(), ans.end());
578     return ans;
579 }
580 };
581
582 vector<point> minkowskiSum(vector<point> A, vector<point> B){
583     int na = (int)A.size(), nb = (int)B.size();
584     if(A.empty() || B.empty()) return {};
585
586     rotate(A.begin(), min_element(A.begin(), A.end()), A.end());
587     rotate(B.begin(), min_element(B.begin(), B.end()), B.end());
588
589     int pa = 0, pb = 0;
590     vector<point> M;
591
592     while(pa < na && pb < nb){
593         M.push_back(A[pa] + B[pb]);
594         ld x = (A[(pa + 1) % na] - A[pa]).cross(B[(pb + 1) % nb] - B[pb]);
595         if(leq(x, 0)) pb++;
596         if(geq(x, 0)) pa++;
597     }
598
599     while(pa < na) M.push_back(A[pa++] + B[0]);

```

```

600     while(pb < nb) M.push_back(B[pb++] + A[0]);
601
602     return M;
603 }
604
605 //Delaunay triangulation in O(n log n)
606 const point inf_pt(inf, inf);
607
608 struct QuadEdge{
609     point origin;
610     QuadEdge* rot = nullptr;
611     QuadEdge* onext = nullptr;
612     bool used = false;
613     QuadEdge* rev() const{return rot->rot;}
614     QuadEdge* lnext() const{return rot->rev()->onext->rot;}
615     QuadEdge* oprev() const{return rot->onext->rot;}
616     point dest() const{return rev()->origin;}
617 };
618
619 QuadEdge* make_edge(const point & from, const point & to){
620     QuadEdge* e1 = new QuadEdge;
621     QuadEdge* e2 = new QuadEdge;
622     QuadEdge* e3 = new QuadEdge;
623     QuadEdge* e4 = new QuadEdge;
624     e1->origin = from;
625     e2->origin = to;
626     e3->origin = e4->origin = inf_pt;
627     e1->rot = e3;
628     e2->rot = e4;
629     e3->rot = e2;
630     e4->rot = e1;
631     e1->onext = e1;
632     e2->onext = e2;
633     e3->onext = e4;
634     e4->onext = e3;
635     return e1;
636 }
637
638 void splice(QuadEdge* a, QuadEdge* b){
639     swap(a->onext->rot->onext, b->onext->rot->onext);
640     swap(a->onext, b->onext);
641 }
642

```



```

643 void delete_edge(QuadEdge* e){
644     splice(e, e->oprev());
645     splice(e->rev(), e->rev()->oprev());
646     delete e->rot;
647     delete e->rev()->rot;
648     delete e;
649     delete e->rev();
650 }
651
652 QuadEdge* connect(QuadEdge* a, QuadEdge* b){
653     QuadEdge* e = make_edge(a->dest(), b->origin);
654     splice(e, a->lnext());
655     splice(e->rev(), b);
656     return e;
657 }
658
659 bool left_of(const point & p, QuadEdge* e){
660     return ge((e->origin - p).cross(e->dest() - p), 0);
661 }
662
663 bool right_of(const point & p, QuadEdge* e){
664     return le((e->origin - p).cross(e->dest() - p), 0);
665 }
666
667 ld det3(ld a1, ld a2, ld a3, ld b1, ld b2, ld b3, ld c1, ld c2, ld c3) {
668     return a1 * (b2 * c3 - c2 * b3) - a2 * (b1 * c3 - c1 * b3) + a3 * (b1
        * c2 - c1 * b2);
669 }
670
671 bool in_circle(const point & a, const point & b, const point & c, const
    point & d) {
672     ld det = -det3(b.x, b.y, b.norm(), c.x, c.y, c.norm(), d.x, d.y, d.
        norm());
673     det += det3(a.x, a.y, a.norm(), c.x, c.y, c.norm(), d.x, d.y, d.norm()
        );
674     det -= det3(a.x, a.y, a.norm(), b.x, b.y, b.norm(), d.x, d.y, d.norm()
        );
675     det += det3(a.x, a.y, a.norm(), b.x, b.y, b.norm(), c.x, c.y, c.norm()
        );
676     return ge(det, 0);
677 }
678
679 pair<QuadEdge*, QuadEdge*> build_tr(int l, int r, vector<point> & P){

```

```

680     if(r - l + 1 == 2){
681         QuadEdge* res = make_edge(P[l], P[r]);
682         return {res, res->rev()};
683     }
684     if(r - l + 1 == 3){
685         QuadEdge *a = make_edge(P[l], P[l + 1]), *b = make_edge(P[l + 1], P[
            r]);
686         splice(a->rev(), b);
687         int sg = sgn((P[l + 1] - P[l]).cross(P[r] - P[l]));
688         if(sg == 0)
689             return {a, b->rev()};
690         QuadEdge* c = connect(b, a);
691         if(sg == 1)
692             return {a, b->rev()};
693         else
694             return {c->rev(), c};
695     }
696     int mid = (l + r) / 2;
697     QuadEdge *ldo, *ldi, *rdo, *rdi;
698     tie(ldo, ldi) = build_tr(l, mid, P);
699     tie(rdi, rdo) = build_tr(mid + 1, r, P);
700     while(true){
701         if(left_of(rdi->origin, ldi)){
702             ldi = ldi->lnext();
703             continue;
704         }
705         if(right_of(ldi->origin, rdi)){
706             rdi = rdi->rev()->onext;
707             continue;
708         }
709         break;
710     }
711     QuadEdge* basel = connect(rdi->rev(), ldi);
712     auto valid = [&basel](QuadEdge* e){return right_of(e->dest(), basel)
        ;};
713     if(ldi->origin == ldo->origin)
714         ldo = basel->rev();
715     if(rdi->origin == rdo->origin)
716         rdo = basel;
717     while(true){
718         QuadEdge* lcand = basel->rev()->onext;
719         if(valid(lcand)){
720             while(in_circle(basel->dest(), basel->origin, lcand->dest(), lcand

```

```

721     ->onext->dest())){
722     QuadEdge* t = lcand->onext;
723     delete_edge(lcand);
724     lcand = t;
725 }
726 QuadEdge* rcand = basel->oprev();
727 if(valid(rcand)){
728     while(in_circle(basel->dest(), basel->origin, rcand->dest(), rcand
729         ->oprev()->dest())){
730         QuadEdge* t = rcand->oprev();
731         delete_edge(rcand);
732         rcand = t;
733     }
734 }
735 if(!valid(lcand) && !valid(rcand))
736     break;
737 if(!valid(lcand) || (valid(rcand) && in_circle(lcand->dest(), lcand
738     ->origin, rcand->origin, rcand->dest()))
739     basel = connect(rcand, basel->rev());
740 else
741     basel = connect(basel->rev(), lcand->rev());
742 }
743 return {ldo, rdo};
744 }
745
746 vector<tuple<point, point, point>> delaunay(vector<point> & P){
747     sort(P.begin(), P.end());
748     auto res = build_tr(0, (int)P.size() - 1, P);
749     QuadEdge* e = res.first;
750     vector<QuadEdge*> edges = {e};
751     while(!e->dest() - e->onext->dest()).cross(e->origin - e->onext->
752         dest()), 0))
753         e = e->onext;
754     auto add = [&P, &e, &edges]() {
755         QuadEdge* curr = e;
756         do{
757             curr->used = true;
758             P.push_back(curr->origin);
759             edges.push_back(curr->rev());
760             curr = curr->lnext();
761         }while(curr != e);
762     };

```

```

760     add();
761     P.clear();
762     int kek = 0;
763     while(kek < (int)edges.size())
764         if(!(e = edges[kek++])->used)
765             add();
766     vector<tuple<point, point, point>> ans;
767     for(int i = 0; i < (int)P.size(); i += 3){
768         ans.emplace_back(P[i], P[i + 1], P[i + 2]);
769     }
770     return ans;
771 }
772
773 struct circ{
774     point c;
775     ld r;
776     circ() {}
777     circ(const point & c, ld r): c(c), r(r) {}
778     set<pair<ld, ld>> ranges;
779
780     void disable(ld l, ld r){
781         ranges.emplace(l, r);
782     }
783
784     auto getActive() const{
785         vector<pair<ld, ld>> ans;
786         ld maxi = 0;
787         for(const auto & dis : ranges){
788             ld l, r;
789             tie(l, r) = dis;
790             if(l > maxi){
791                 ans.emplace_back(maxi, l);
792             }
793             maxi = max(maxi, r);
794         }
795         if(!eq(maxi, 2*pi)){
796             ans.emplace_back(maxi, 2*pi);
797         }
798         return ans;
799     }
800 };
801
802 ld areaUnionCircles(const vector<circ> & circs){

```

```

803 vector<circ> valid;
804 for(const circ & curr : circs){
805     if(eq(curr.r, 0)) continue;
806     circ nuevo = curr;
807     for(circ & prev : valid){
808         if(circleInsideCircle(prev.c, prev.r, nuevo.c, nuevo.r)){
809             nuevo.disable(0, 2*pi);
810         }else if(circleInsideCircle(nuevo.c, nuevo.r, prev.c, prev.r)){
811             prev.disable(0, 2*pi);
812         }else{
813             auto cruce = intersectionCircles(prev.c, prev.r, nuevo.c, nuevo.
814                 r);
815             if(cruce.size() == 2){
816                 ld a1 = (cruce[0] - prev.c).ang();
817                 ld a2 = (cruce[1] - prev.c).ang();
818                 ld b1 = (cruce[1] - nuevo.c).ang();
819                 ld b2 = (cruce[0] - nuevo.c).ang();
820                 if(a1 < a2){
821                     prev.disable(a1, a2);
822                 }else{
823                     prev.disable(a1, 2*pi);
824                     prev.disable(0, a2);
825                 }
826                 if(b1 < b2){
827                     nuevo.disable(b1, b2);
828                 }else{
829                     nuevo.disable(b1, 2*pi);
830                     nuevo.disable(0, b2);
831                 }
832             }
833         }
834     }
835     valid.push_back(nuevo);
836 }
837 ld ans = 0;
838 for(const circ & curr : valid){
839     for(const auto & range : curr.getActive()){
840         ld l, r;
841         tie(l, r) = range;
842         ans += curr.r*(curr.c.x * (sin(r) - sin(l)) - curr.c.y * (cos(r) -
843             cos(l))) + curr.r*curr.r*(r-l);
844     }
845 }

```

```

844     return ans/2;
845 };
846
847 struct plane{
848     point a, v;
849     plane(): a(), v(){}
850     plane(const point& a, const point& v): a(a), v(v){}
851
852     point intersect(const plane& p) const{
853         ld t = (p.a - a).cross(p.v) / v.cross(p.v);
854         return a + v*t;
855     }
856
857     bool outside(const point& p) const{ // test if point p is strictly
858         outside
859         return le(v.cross(p - a), 0);
860     }
861
862     bool inside(const point& p) const{ // test if point p is inside or in
863         the boundary
864         return geq(v.cross(p - a), 0);
865     }
866
867     bool operator<(const plane& p) const{ // sort by angle
868         auto lhs = make_tuple(v.half({1, 0}), ld(0), v.cross(p.a - a));
869         auto rhs = make_tuple(p.v.half({1, 0}), v.cross(p.v), ld(0));
870         return lhs < rhs;
871     }
872
873     bool operator==(const plane& p) const{ // paralell and same directions
874         , not really equal
875         return eq(v.cross(p.v), 0) && ge(v.dot(p.v), 0);
876     }
877 };
878
879 vector<point> halfPlaneIntersection(vector<plane> planes){
880     planes.push_back({{0, -inf}, {1, 0}});
881     planes.push_back({{inf, 0}, {0, 1}});
882     planes.push_back({{0, inf}, {-1, 0}});
883     planes.push_back({{-inf, 0}, {0, -1}});
884     sort(planes.begin(), planes.end());
885     planes.erase(unique(planes.begin(), planes.end()), planes.end());
886     deque<plane> ch;

```

```

884 deque<point> poly;
885 for(const plane& p : planes){
886     while(ch.size() >= 2 && p.outside(poly.back())) ch.pop_back(), poly.
        pop_back();
887     while(ch.size() >= 2 && p.outside(poly.front())) ch.pop_front(),
        poly.pop_front();
888     if(p.v.half({1, 0}) && poly.empty()) return {};
889     ch.push_back(p);
890     if(ch.size() >= 2) poly.push_back(ch[ch.size()-2].intersect(ch[ch.
        size()-1]));
891 }
892 while(ch.size() >= 3 && ch.front().outside(poly.back())) ch.pop_back()
    , poly.pop_back();
893 while(ch.size() >= 3 && ch.back().outside(poly.front())) ch.pop_front
    (), poly.pop_front();
894 poly.push_back(ch.back().intersect(ch.front()));
895 return vector<point>(poly.begin(), poly.end());
896 }
897
898 vector<point> halfPlaneIntersectionRandomized(vector<plane> planes){
899     point p = planes[0].a;
900     int n = planes.size();
901     random_shuffle(planes.begin(), planes.end());
902     for(int i = 0; i < n; ++i){
903         if(planes[i].inside(p)) continue;
904         ld lo = -inf, hi = inf;
905         for(int j = 0; j < i; ++j){
906             ld A = planes[j].v.cross(planes[i].v);
907             ld B = planes[j].v.cross(planes[j].a - planes[i].a);
908             if(ge(A, 0)){
909                 lo = max(lo, B/A);
910             }else if(le(A, 0)){
911                 hi = min(hi, B/A);
912             }else{
913                 if(ge(B, 0)) return {};
914             }
915             if(ge(lo, hi)) return {};
916         }
917         p = planes[i].a + planes[i].v*lo;
918     }
919     return {p};
920 }
921

```

```

922 int main(){
923     /*vector<pair<point, point>> centers = {{point(-2, 5), point(-8, -7)},
        {point(14, 4), point(18, 6)}, {point(9, 20), point(9, 28)},
924         {point(21, 20), point(21, 29)}, {point(8, -10),
            point(14, -10)}, {point(24, -6), point(34, -6)}
        },
925         {point(34, 8), point(36, 9)}, {point(50, 20),
            point(56, 24.5)}};
926     vector<pair<ld, ld>> radii = {{7, 4}, {3, 5}, {4, 4}, {4, 5}, {3, 3},
        {4, 6}, {5, 1}, {10, 2.5}};
927     int n = centers.size();
928     for(int i = 0; i < n; ++i){
929         cout << "\n" << centers[i].first << " " << radii[i].first << " " <<
            centers[i].second << " " << radii[i].second << "\n";
930         auto extLines = tangents(centers[i].first, radii[i].first, centers[i]
            ].second, radii[i].second, false);
931         cout << "Exterior tangents:\n";
932         for(auto par : extLines){
933             for(auto p : par){
934                 cout << p << " ";
935             }
936             cout << "\n";
937         }
938         auto intLines = tangents(centers[i].first, radii[i].first, centers[i]
            ].second, radii[i].second, true);
939         cout << "Interior tangents:\n";
940         for(auto par : intLines){
941             for(auto p : par){
942                 cout << p << " ";
943             }
944             cout << "\n";
945         }
946     }*/
947
948     /*int n;
949     cin >> n;
950     vector<point> P(n);
951     for(auto & p : P) cin >> p;
952     auto triangulation = delaunay(P);
953     for(auto triangle : triangulation){
954         cout << get<0>(triangle) << " " << get<1>(triangle) << " " << get
            <2>(triangle) << "\n";
955     }*/

```

```

956
957 /*int n;
958 cin >> n;
959 vector<point> P(n);
960 for(auto & p : P) cin >> p;
961 auto ans = smallestEnclosingCircle(P);
962 cout << ans.first << " " << ans.second << "\n";*/
963
964 /*vector<point> P;
965 srand(time(0));
966 for(int i = 0; i < 1000; ++i){
967     P.emplace_back(rand() % 1000000000, rand() % 1000000000);
968 }
969 point o(rand() % 1000000000, rand() % 1000000000), v(rand() %
970     1000000000, rand() % 1000000000);
971 polarSort(P, o, v);
972 auto ang = [&](point p){
973     ld th = atan2(p.y, p.x);
974     if(th < 0) th += acosl(-1)*2;
975     ld t = atan2(v.y, v.x);
976     if(t < 0) t += acosl(-1)*2;
977     if(th < t) th += acosl(-1)*2;
978     return th;
979 };
980 for(int i = 0; i < P.size()-1; ++i){
981     assert(leq(ang(P[i] - o), ang(P[i+1] - o)));
982 }*/
983 return 0;
984 }

```

5 Varios

5.1 Template

```

1 #include<bits/stdc++.h>
2 using namespace std;
3
4 #define forn(i,n)      for(int i=0; i<n; i++)
5 #define forr(i,a,n)    for(int i=a; i<n; i++)
6 #define fore(i,a,n)    for(int i=a; i<=n; i++)
7 #define each(a,b)      for(auto a: b)
8 #define all(v)          v.begin(),v.end()
9 #define sz(a)           (int)a.size()

```

```

10 #define debln(a)        cout << a << "\n"
11 #define deb(a)          cout << a << " "
12 #define pb              push_back
13
14 typedef long long ll;
15 typedef vector<int> vi;
16 typedef pair<int,int> ii;
17
18 void sol(){
19
20 }
21
22 int main(){
23     ios::sync_with_stdio(false);cin.tie(0);
24
25     int t=1;
26     cin>>t;
27     while(t--){
28         sol();
29     }
30
31     return 0;
32 }

```

5.2 String a vector<int>

```

1 //Convertir una cadena de numeros separados por " " en vector de enteros
2 //Leer varias de esas queries
3 cin.ignore();
4 while(q--){
5     string s;
6     getline(cin, s);
7     vector<int> qr;
8     stringstream ss(s);
9     int num;
10    while (ss >> num)    qr.push_back(num);
11 }

```

5.3 Generar permutaciones

```

1 //Generar todas las permutaciones de un arreglo
2 sort(all(a));
3 do{
4     //hacer lo que quieras con la perm generada

```

```
5 }while(next_permutation(all(a)));
```

5.4 2 Sat

```
1 struct twoSat{
2     int s;
3     vector<vector<int>> g,gr;
4     vector<int> visited,ids,topologic_sort,val;
5     twoSat(int n){
6         s=n;
7         g.assign(n*2+1,vector<int>());
8         gr.assign(n*2+1,vector<int>());
9         visited.assign(n*2+1,0);
10        ids.assign(n*2+1,0);
11        val.assign(n+1,0);
12    }
13    void addEdge(int a,int b){
14        g[a].push_back(b);
15        gr[b].push_back(a);
16    }
17    void addOr(int a,bool ba,int b,bool bb){
18        addEdge(a+(ba?s:0),b+(bb?0:s));
19        addEdge(b+(bb?s:0),a+(ba?0:s));
20    }
21    void addXor(int a,bool ba,int b,bool bb){
22        addOr(a,ba,b,bb);
23        addOr(a,!ba,b,!bb);
24    }
25    void addAnd(int a,bool ba,int b,bool bb){
26        addXor(a,!ba,b,bb);
27    }
28    void dfs(int u){
29        if(visited[u]!=0) return;
30        visited[u]=1;
31        for(int node:g[u])dfs(node);
32        topologic_sort.push_back(u);
33    }
34    void dfsr(int u,int id){
35        if(visited[u]!=0) return;
36        visited[u]=1;
37        ids[u]=id;
38        for(int node:gr[u])dfsr(node,id);
39    }
```

```
40    bool algo(){
41        for(int i=0;i<s*2;i++) if(visited[i]==0) dfs(i);
42        fill(visited.begin(),visited.end(),0);
43        reverse(topologic_sort.begin(),topologic_sort.end());
44        int id=0;
45        for(int i=0;i<topologic_sort.size();i++){
46            if(visited[topologic_sort[i]]==0)dfsr(topologic_sort[i],id
47                ++);
48        }
49        for(int i=0;i<s;i++){
50            if(ids[i]==ids[i+s]) return false;
51            val[i]=(ids[i]>ids[i+s]?0:1);
52        }
53        return true;
54    }
55 }
```

5.5 Bits

```
1 __builtin_popcount(maks) // Count the numbers of on bits
```

5.6 Matrix

```
1 const int N=100, MOD=1e9+7;
2 struct Matrix {
3     ll a[N][N];
4     Matrix() {memset(a,0,sizeof(a));}
5     Matrix operator *(Matrix other) { // Product of a matrix
6         Matrix product=Matrix();
7         rep(i,0,N) rep(j,0,N) rep(k,0,N) {
8             product.a[i][k]+=a[i][j]*other.a[j][k];
9             product.a[i][k]%=MOD;
10        }
11        return product;
12    }
13 };
14 Matrix expo_power(Matrix a, ll n) { // Matrix exponentiation
15     Matrix res=Matrix();
16     rep(i,0,N) res.a[i][i]=1; // Matriz identidad
17     while(n){
18         if(n&1) res=res*a;
19         n>>=1;
20         a=a*a;
21     }
```

```

22     return res;
23 } // Ej. Matrix M=Matrix(); M.a[0][0]=1; M=M*M; Matrix res=
    expo_power(M,k);

```

5.7 MO

```

1 void remove(idx); // TODO: remove value at idx from data structure
2 void add(idx);    // TODO: add value at idx from data structure
3 int get_answer(); // TODO: extract the current answer of the data
    structure
4
5 int block_size;//Recomended sqrt(n)
6
7 struct Query {
8     int l, r, idx;
9     bool operator<(Query other) const
10    {
11        return make_pair(l / block_size, r) <
12            make_pair(other.l / block_size, other.r);
13    }
14 };
15
16 vector<int> mo_s_algorithm(vector<Query> queries) {
17     vector<int> answers(queries.size());
18     sort(queries.begin(), queries.end());
19
20     // TODO: initialize data structure
21
22     int cur_l = 0;
23     int cur_r = -1;
24     // invariant: data structure will always reflect the range [cur_l,
25         cur_r]
26     for (Query q : queries) {
27         while (cur_l > q.l) {
28             cur_l--;
29             add(cur_l);
30         }
31         while (cur_r < q.r) {
32             cur_r++;
33             add(cur_r);
34         }
35         while (cur_l < q.l) {
36             remove(cur_l);

```

```

36         cur_l++;
37     }
38     while (cur_r > q.r) {
39         remove(cur_r);
40         cur_r--;
41     }
42     answers[q.idx] = get_answer();
43 }
44 return answers;
45 }

```

5.8 PBS

```

1
2 1.Crear un arreglo con para procesar
3 2.Para cada elemento inicialicar 1 l y en q+1 r;
4 for(int i=1;i<=n;i++){
5     m[i].x=1,m[i].y=q+1;
6 }
7 bool flag=true;
8 while(flag){
9     flag=false;
10    // limpiar la estructura de datos
11    for(int i=0;i<=4*n+5;i++)st[i]=0,lazy[i]=0;
12    for(int i=1;i<=n;i++)
13        //Si es diefente l!=r se procesa;
14        if(m[i].x!=m[i].y){ flag=true;tocheck[(m[i].x+m[i].y)/2].
15            push_back(i);}
16    for(int i=1;i<=q;i++){
17        if(!flag)break;
18        // Se aplican las queries
19        update(0,n-1,qs[i].x,qs[i].y,qs[i].z,0);
20        update(0,n-1,qs[i].x,qs[i].x,qs[i].k,0);
21        while(tocheck[i].size()){
22            int id=tocheck[i].back();
23            tocheck[i].pop_back();
24            // Se obserba si se cumblío la caondicion para el
25            elemeto
26            if(ai[id]<=query(0,n-1,S[id],S[id],0)) m[id].y=i;
27            else m[id].x=i+1;
28        }
29    }
30 }

```

```
29 // Solo se imprime
30 for(int i=1;i<=n;i++){
31     if(m[i].x<=q) cout<<m[i].x<<endl;
32     else cout<<-1<<endl;
33 }
```