.

Descongelen a Victor Moreno

Contents

1   Estructuras de Datos

1.1   Unordered Map

```cpp
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;

struct custom_hash {
    static uint64_t splitmix64(uint64_t x) {
        // http://xorshift.di.unimi.it/splitmix64.c
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }

    size_t operator()(uint64_t x) const {
        static const uint64_t FIXED_RANDOM = chrono::steady_clock::now()
                .time_since_epoch().count();
        return splitmix64(x + FIXED_RANDOM);
    }
};

gp_hash_table<int, int,custom_hash> m1;

//Funcion count
m1.find(x)!=m1.end()
```

1.2   Segment tree Recursivo

```
%%
%% This is file '.tex',
%% generated with the docstrip utility.
%%
%% The original source files were:
%%
%% fileerr.dtx  (with options: 'return')
%%
%% This is a generated file.
%%
%% The source is maintained by the LaTeX Project team and bug
%% reports for it can be opened at https://latex-project.org/bugs/
%% (but please observe conditions on bug reports sent to that address!)
%%
%%
%% Copyright (C) 1993-2020
%% The LaTeX3 Project and any individual authors listed elsewhere
%% in this file.
%%
%% This file was generated from file(s) of the Standard LaTeX 'Tools
    Bundle'.
%%
    ------------------------------------------------------------------

%%
%% It may be distributed and/or modified under the
%% conditions of the LaTeX Project Public License, either version 1.3c
%% of this license or (at your option) any later version.
%% The latest version of this license is in
%%     https://www.latex-project.org/lppl.txt
%% and version 1.3c or later is part of all distributions of LaTeX
%% version 2005/12/01 or later.
%%
%% This file may only be distributed together with a copy of the LaTeX
%% 'Tools Bundle'. You may however distribute the LaTeX 'Tools Bundle'
%% without such generated files.
%%
%% The list of all files belonging to the LaTeX 'Tools Bundle' is
%% given in the file 'manifest.txt'.
%%
 \message{File ignored}
\endinput
%%
%% End of file '.tex'.
```

1.3   Segment Tree Iterativo

```cpp
//Para procesar querys de tipo k-esimo es necesario crear un arbol
    binario perfector(llenar con 0's)
template<typename T>
struct SegmentTree{
  int N;
  vector<T> ST;

  //Creacion a partir de un arreglo O(n)
  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] = creaNodo(); //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] = creaNodo();//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 = creaNodo(), resr = creaNodo();//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 querys 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 <= l) 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
```

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;    Requerir 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 requeires 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

```cpp
#define inf INT_MAX
const int MAX=5e5+2;
typedef pair<ll, ll> item;
struct node{
    item val;
    node *l, *r;
    node(): l(nullptr),r(nullptr),val({inf,inf}){};
    node(node *_l,node *_r):l(_l),r(_r){
        val=min(l->val,r->val);
    }
    node(ll value,ll pos):r(nullptr),l(nullptr){
        val=make_pair(value,pos);
    }
};
pair<ll,ll>all;
vector<node*>versions(MAX,nullptr);
node* build(int l,int r){
    if(l==r)return  new node(inf,l);
    int m=(l+r)/2;
    return new node(build(l,m),build(m+1,r));
}

node* update(node *root,int l,int r,int pos,int val){
    if(l==r){
        return new node(val,pos);}
    int m=(l+r)/2;
    if(pos<=m) return new node(update(root->l,l,m,pos,val),root->r);
    return new node(root->l,update(root->r,m+1,r,pos,val));
}
item query(node *root,int l,int r,int a,int b){
    if(a>r || b<l) return all;
    if(a<=l && r<=b) return root->val;
    int m=(l+r)/2;
    return min(query(root->l,l,m,a,b),query(root->r,m+1,r,a,b));
}
```

1.10 Sparce Table

```cpp
//Se usa para RMQ porque se puede hacer en O(1), no acepta updates
vector<int>lg;
vector<vector<int>>st;
int *nums;
void init(int n){
    int logn=(int) log2(n)+1;
    lg.assign(n+1,0);
    st.assign(logn,vector<int>(n+1));
    for(int i=0;i<n;i++) st[0][i]=nums[i];
    lg[1]=0;
    for(int i=2;i<=n;i++) lg[i]=lg[i/2]+1;
    for(int i=1;i<logn;i++)
        for(int j=0;j+(1<<i)<n;j++)st[i][j]=min(st[i-1][j],st[i-1][j
            +(1<<(i-1))]);
}
int query(int a,int b){
    int logn=lg[(b-a+1)];
    cout<<st[logn][a]<<endl;
    return min(st[logn][a],st[logn][b-(1<<logn)+1]);
}
```

1.11 Walvet Tree

```cpp
// indexed in 1
// from pointer to first element and to to end
// x and y The minimun element and y the max element
// If you need only one function or more erase the others
// If you need tu construct other function you only required to
    undertand the limit, this
// are the same
struct wavelet_tree{
  int lo, hi;
  wavelet_tree *l, *r;
  vector<int> b;
  wavelet_tree(int *from, int *to, int x, int y){
    lo = x, hi = y;
    if(lo == hi or from >= to) return;
    int mid = (lo+hi)/2;
    auto f = [mid](int x){ return x <= mid;};
    b.reserve(to-from+1);
    b.pb(0);
    for(auto it = from; it != to; it++)
      b.push_back(b.back() + f(*it));
    auto pivot = stable_partition(from, to, f);
    l = new wavelet_tree(from, pivot, lo, mid);
    r = new wavelet_tree(pivot, to, mid+1, hi);
  }
  //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  };
```

1.12   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 inserString(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  }
```

2   Strings

2.1   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[
                        i]);
32              else
33                  T[x].ch[i] = T[T[x].fail].ch[i];
34          }
35      }
36  }
```

2.2   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};
```

2.3   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 }
```

2.4   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 }
```

2.5   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 necesary functions
36  void findTerminals(){
37     int p = last;
38     while(p > 0) {
39       sa[p].terminal=1;
40       p = sa[p].link;
41     }
42  }
```

3   Graph

```
1  struct disjointSet{
2    int N;
3    vector<short int> rank;
4    vi parent, count;
5
6    disjointSet(int N): N(N), parent(N), count(N), rank(N){}
7
8    void makeSet(int v){
9      count[v] = 1;
10     parent[v] = v;
11   }
12
13   int findSet(int v){
14     if(v == parent[v]) return v;
15     return parent[v] = findSet(parent[v]);
16   }
17
18   void unionSet(int a, int b){
19     a = findSet(a), b = findSet(b);
20     if(a == b) return;
21     if(rank[a] < rank[b]){
22       parent[a] = b;
23       count[b] += count[a];
24     }else{
25       parent[b] = a;
26       count[a] += count[b];
27       if(rank[a] == rank[b]) ++rank[a];
28     }
29   }
30  };
31
32  struct edge{
33    int source, dest, cost;
34
35    edge(): source(0), dest(0), cost(0){}
36
37    edge(int dest, int cost): dest(dest), cost(cost){}
38
39    edge(int source, int dest, int cost): source(source), dest(dest), cost
         (cost){}
40
41    bool operator==(const edge & b) const{
42      return source == b.source && dest == b.dest && cost == b.cost;
43    }
44    bool operator<(const edge & b) const{
```

```
45        return cost < b.cost;
46      }
47      bool operator>(const edge & b) const{
48        return cost > b.cost;
49      }
50    };
51
52    struct path{
53      int cost = inf;
54      deque<int> vertices;
55      int size = 1;
56      int prev = -1;
57    };
58
59    struct graph{
60      vector<vector<edge>> adjList;
61      vector<vb> adjMatrix;
62      vector<vi> costMatrix;
63      vector<edge> edges;
64      int V = 0;
65      bool dir = false;
66
67      graph(int n, bool dir): V(n), dir(dir), adjList(n), edges(n),
              adjMatrix(n, vb(n)), costMatrix(n, vi(n)){
68        for(int i = 0; i < n; ++i)
69          for(int j = 0; j < n; ++j)
70            costMatrix[i][j] = (i == j ? 0 : inf);
71      }
72
73      void add(int source, int dest, int cost){
74        adjList[source].emplace_back(source, dest, cost);
75        edges.emplace_back(source, dest, cost);
76        adjMatrix[source][dest] = true;
77        costMatrix[source][dest] = cost;
78        if(!dir){
79          adjList[dest].emplace_back(dest, source, cost);
80          adjMatrix[dest][source] = true;
81          costMatrix[dest][source] = cost;
82        }
83      }
84
85      void buildPaths(vector<path> & paths){
86        for(int i = 0; i < V; i++){
87          int u = i;
88          for(int j = 0; j < paths[i].size; j++){
89            paths[i].vertices.push_front(u);
90            u = paths[u].prev;
91          }
92        }
93      }
94
95      vector<path> dijkstra(int start){
96        priority_queue<edge, vector<edge>, greater<edge>> cola;
97        vector<path> paths(V);
98        cola.emplace(start, 0);
99        paths[start].cost = 0;
100       while(!cola.empty()){
101         int u = cola.top().dest; cola.pop();
102         for(edge & current : adjList[u]){
103           int v = current.dest;
104           int nuevo = paths[u].cost + current.cost;
105           if(nuevo == paths[v].cost && paths[u].size + 1 < paths[v].size){
106             paths[v].prev = u;
107             paths[v].size = paths[u].size + 1;
108           }else if(nuevo < paths[v].cost){
109             paths[v].prev = u;
110             paths[v].size = paths[u].size + 1;
111             cola.emplace(v, nuevo);
112             paths[v].cost = nuevo;
113           }
114         }
115       }
116       buildPaths(paths);
117       return paths;
118     }
119
120     vector<path> bellmanFord(int start){
121       vector<path> paths(V, path());
122       vi processed(V);
123       vb inQueue(V);
124       queue<int> Q;
125       paths[start].cost = 0;
126       Q.push(start);
127       while(!Q.empty()){
128         int u = Q.front(); Q.pop(); inQueue[u] = false;
129         if(paths[u].cost == inf) continue;
```

```
130        ++processed[u];
131        if(processed[u] == V){
132          cout << "Negative cycle\n";
133          return {};
134        }
135        for(edge & current : adjList[u]){
136          int v = current.dest;
137          int nuevo = paths[u].cost + current.cost;
138          if(nuevo == paths[v].cost && paths[u].size + 1 < paths[v].size){
139            paths[v].prev = u;
140            paths[v].size = paths[u].size + 1;
141          }else if(nuevo < paths[v].cost){
142            if(!inQueue[v]){
143              Q.push(v);
144              inQueue[v] = true;
145            }
146            paths[v].prev = u;
147            paths[v].size = paths[u].size + 1;
148            paths[v].cost = nuevo;
149          }
150        }
151      }
152      buildPaths(paths);
153      return paths;
154    }
155
156
157
158
159
160    vector<vi> floyd(){
161      vector<vi> tmp = costMatrix;
162      for(int k = 0; k < V; ++k)
163        for(int i = 0; i < V; ++i)
164          for(int j = 0; j < V; ++j)
165            if(tmp[i][k] != inf && tmp[k][j] != inf)
166              tmp[i][j] = min(tmp[i][j], tmp[i][k] + tmp[k][j]);
167      return tmp;
168    }
169
170    vector<vb> transitiveClosure(){
171      vector<vb> tmp = adjMatrix;
172      for(int k = 0; k < V; ++k)
173        for(int i = 0; i < V; ++i)
174          for(int j = 0; j < V; ++j)
175            tmp[i][j] = tmp[i][j] || (tmp[i][k] && tmp[k][j]);
176      return tmp;
177    }
178
179    vector<vb> transitiveClosureDFS(){
180      vector<vb> tmp(V, vb(V));
181      function<void(int, int)> dfs = [&](int start, int u){
182        for(edge & current : adjList[u]){
183          int v = current.dest;
184          if(!tmp[start][v]){
185            tmp[start][v] = true;
186            dfs(start, v);
187          }
188        }
189      };
190      for(int u = 0; u < V; u++)
191        dfs(u, u);
192      return tmp;
193    }
194
195    bool isBipartite(){
196      vi side(V, -1);
197      queue<int> q;
198      for (int st = 0; st < V; ++st){
199        if(side[st] != -1) continue;
200        q.push(st);
201        side[st] = 0;
202        while(!q.empty()){
203          int u = q.front();
204          q.pop();
205          for (edge & current : adjList[u]){
206            int v = current.dest;
207            if(side[v] == -1) {
208              side[v] = side[u] ^ 1;
209              q.push(v);
210            }else{
211              if(side[v] == side[u]) return false;
212            }
213          }
214        }
215      }
```

```
216        return true;
217      }
218
219      vi topologicalSort(){
220        int visited = 0;
221        vi order, indegree(V);
222        for(auto & node : adjList){
223          for(edge & current : node){
224            int v = current.dest;
225            ++indegree[v];
226          }
227        }
228        queue<int> Q;
229        for(int i = 0; i < V; ++i){
230          if(indegree[i] == 0) Q.push(i);
231        }
232        while(!Q.empty()){
233          int source = Q.front();
234          Q.pop();
235          order.push_back(source);
236          ++visited;
237          for(edge & current : adjList[source]){
238            int v = current.dest;
239            --indegree[v];
240            if(indegree[v] == 0) Q.push(v);
241          }
242        }
243        if(visited == V) return order;
244        else return {};
245      }
246
247      bool hasCycle(){
248        vi color(V);
249        function<bool(int, int)> dfs = [&](int u, int parent){
250          color[u] = 1;
251          bool ans = false;
252          int ret = 0;
253          for(edge & current : adjList[u]){
254            int v = current.dest;
255            if(color[v] == 0)
256              ans |= dfs(v, u);
257            else if(color[v] == 1 && (dir || v != parent || ret++))
258              ans = true;
```

```
259          }
260          color[u] = 2;
261          return ans;
262        };
263        for(int u = 0; u < V; ++u)
264          if(color[u] == 0 && dfs(u, -1))
265            return true;
266        return false;
267      }
268
269      pair<vb, vector<edge>> articulationBridges(){
270        vi low(V), label(V);
271        vb points(V);
272        vector<edge> bridges;
273        int time = 0;
274        function<int(int, int)> dfs = [&](int u, int p){
275          label[u] = low[u] = ++time;
276          int hijos = 0, ret = 0;
277          for(edge & current : adjList[u]){
278            int v = current.dest;
279            if(v == p && !ret++) continue;
280            if(!label[v]){
281              ++hijos;
282              dfs(v, u);
283              if(label[u] <= low[v])
284                points[u] = true;
285              if(label[u] < low[v])
286                bridges.push_back(current);
287              low[u] = min(low[u], low[v]);
288            }
289            low[u] = min(low[u], label[v]);
290          }
291          return hijos;
292        };
293        for(int u = 0; u < V; ++u)
294          if(!label[u])
295            points[u] = dfs(u, -1) > 1;
296        return make_pair(points, bridges);
297      }
298
299      vector<vi> scc(){
300        vi low(V), label(V);
301        int time = 0;
```

```
302        vector<vi> ans;
303        stack<int> S;
304        function<void(int)> dfs = [&](int u){
305          label[u] = low[u] = ++time;
306          S.push(u);
307          for(edge & current : adjList[u]){
308            int v = current.dest;
309            if(!label[v]) dfs(v);
310            low[u] = min(low[u], low[v]);
311          }
312          if(label[u] == low[u]){
313            vi comp;
314            while(S.top() != u){
315              comp.push_back(S.top());
316              low[S.top()] = V + 1;
317              S.pop();
318            }
319            comp.push_back(S.top());
320            S.pop();
321            ans.push_back(comp);
322            low[u] = V + 1;
323          }
324        };
325        for(int u = 0; u < V; ++u)
326          if(!label[u]) dfs(u);
327        return ans;
328      }
329
330    vector<edge> kruskal(){
331      sort(edges.begin(), edges.end());
332      vector<edge> MST;
333      disjointSet DS(V);
334      for(int u = 0; u < V; ++u)
335        DS.makeSet(u);
336      int i = 0;
337      while(i < edges.size() && MST.size() < V - 1){
338        edge current = edges[i++];
339        int u = current.source, v = current.dest;
340        if(DS.findSet(u) != DS.findSet(v)){
341          MST.push_back(current);
342          DS.unionSet(u, v);
343        }
344      }
345      return MST;
346    }
347
348    bool tryKuhn(int u, vb & used, vi & left, vi & right){
349      if(used[u]) return false;
350      used[u] = true;
351      for(edge & current : adjList[u]){
352        int v = current.dest;
353        if(right[v] == -1 || tryKuhn(right[v], used, left, right)){
354          right[v] = u;
355          left[u] = v;
356          return true;
357        }
358      }
359      return false;
360    }
361
362    bool augmentingPath(int u, vb & used, vi & left, vi & right){
363      used[u] = true;
364      for(edge & current : adjList[u]){
365        int v = current.dest;
366        if(right[v] == -1){
367          right[v] = u;
368          left[u] = v;
369          return true;
370        }
371      }
372      for(edge & current : adjList[u]){
373        int v = current.dest;
374        if(!used[right[v]] && augmentingPath(right[v], used, left, right))
375        {
376          right[v] = u;
377          left[u] = v;
378          return true;
379        }
380      }
381      return false;
382    }

//vertices from the left side numbered from 0 to l-1
//vertices from the right side numbered from 0 to r-1
//graph[u] represents the left side
//graph[u][v] represents the right side
```

```
387      //we can use tryKuhn() or augmentingPath()
388      vector<pair<int, int>> maxMatching(int l, int r){
389        vi left(l, -1), right(r, -1);
390        vb used(l);
391        for(int u = 0; u < l; ++u){
392          tryKuhn(u, used, left, right);
393          fill(used.begin(), used.end(), false);
394        }
395        vector<pair<int, int>> ans;
396        for(int u = 0; u < r; ++u){
397          if(right[u] != -1){
398            ans.emplace_back(right[u], u);
399          }
400        }
401        return ans;
402      }
403
404      void dfs(int u, vi & status, vi & parent){
405        status[u] = 1;
406        for(edge & current : adjList[u]){
407          int v = current.dest;
408          if(status[v] == 0){ //not visited
409            parent[v] = u;
410            dfs(v, status, parent);
411          }else if(status[v] == 1){ //explored
412            if(v == parent[u]){
413              //bidirectional node u<-->v
414            }else{
415              //back edge u-v
416            }
417          }else if(status[v] == 2){ //visited
418            //forward edge u-v
419          }
420        }
421        status[u] = 2;
422      }
423    };
424
425    struct tree{
426      vi parent, level, weight;
427      vector<vi> dists, DP;
428      int n, root;
429
430      void dfs(int u, graph & G){
431        for(edge & curr : G.adjList[u]){
432          int v = curr.dest;
433          int w = curr.cost;
434          if(v != parent[u]){
435            parent[v] = u;
436            weight[v] = w;
437            level[v] = level[u] + 1;
438            dfs(v, G);
439          }
440        }
441      }
442
443      tree(int n, int root): n(n), root(root), parent(n), level(n), weight(n
            ), dists(n, vi(20)), DP(n, vi(20)){
444        parent[root] = root;
445      }
446
447      tree(graph & G, int root): n(G.V), root(root), parent(G.V), level(G.V)
            , weight(G.V), dists(G.V, vi(20)), DP(G.V, vi(20)){
448        parent[root] = root;
449        dfs(root, G);
450      }
451
452      void pre(){
453        for(int u = 0; u < n; u++){
454          DP[u][0] = parent[u];
455          dists[u][0] = weight[u];
456        }
457        for(int i = 1; (1 << i) <= n; ++i){
458          for(int u = 0; u < n; ++u){
459            DP[u][i] = DP[DP[u][i - 1]][i - 1];
460            dists[u][i] = dists[u][i - 1] + dists[DP[u][i - 1]][i - 1];
461          }
462        }
463      }
464
465      int ancestor(int p, int k){
466        int h = level[p] - k;
467        if(h < 0) return -1;
468        int lg;
469        for(lg = 1; (1 << lg) <= level[p]; ++lg);
470        lg--;
```

```
471      for(int i = lg; i >= 0; --i){
472        if(level[p] - (1 << i) >= h){
473          p = DP[p][i];
474        }
475      }
476      return p;
477    }
478
479    int lca(int p, int q){
480      if(level[p] < level[q]) swap(p, q);
481      int lg;
482      for(lg = 1; (1 << lg) <= level[p]; ++lg);
483      lg--;
484      for(int i = lg; i >= 0; --i){
485        if(level[p] - (1 << i) >= level[q]){
486          p = DP[p][i];
487        }
488      }
489      if(p == q) return p;
490
491      for(int i = lg; i >= 0; --i){
492        if(DP[p][i] != -1 && DP[p][i] != DP[q][i]){
493          p = DP[p][i];
494          q = DP[q][i];
495        }
496      }
497      return parent[p];
498    }
499
500    int dist(int p, int q){
501      if(level[p] < level[q]) swap(p, q);
502      int lg;
503      for(lg = 1; (1 << lg) <= level[p]; ++lg);
504      lg--;
505      int sum = 0;
506      for(int i = lg; i >= 0; --i){
507        if(level[p] - (1 << i) >= level[q]){
508          sum += dists[p][i];
509          p = DP[p][i];
510        }
511      }
512      if(p == q) return sum;
513
```

```
514      for(int i = lg; i >= 0; --i){
515        if(DP[p][i] != -1 && DP[p][i] != DP[q][i]){
516          sum += dists[p][i] + dists[q][i];
517          p = DP[p][i];
518          q = DP[q][i];
519        }
520      }
521      sum += dists[p][0] + dists[q][0];
522      return sum;
523    }
524  };
```

4   Flow

4.1   Dinics

```
1  struct Dinic{
2    int nodes,src,dst;
3    vector<int> dist,q,work;
4    struct edge {int to,rev;ll f,cap;};
5    vector<vector<edge>> g;
6    Dinic(int x):nodes(x),g(x),dist(x),q(x),work(x){}
7    void add_edge(int s, int t, ll cap){
8      g[s].pb((edge){t,SZ(g[t]),0,cap});
9      g[t].pb((edge){s,SZ(g[s])-1,0,0});
10   }
11   bool dinic_bfs(){
12     fill(ALL(dist),-1);dist[src]=0;
13     int qt=0;q[qt++]=src;
14     for(int qh=0;qh<qt;qh++){
15       int u=q[qh];
16       fore(i,0,SZ(g[u])){
17         edge &e=g[u][i];int v=g[u][i].to;
18         if(dist[v]<0&&e.f<e.cap)dist[v]=dist[u]+1,q[qt++]=v;
19       }
20     }
21     return dist[dst]>=0;
22   }
23   ll dinic_dfs(int u, ll f){
24     if(u==dst)return f;
25     for(int &i=work[u];i<SZ(g[u]);i++){
26       edge &e=g[u][i];
27       if(e.cap<=e.f)continue;
```

```
28        int v=e.to;
29        if(dist[v]==dist[u]+1){
30          ll df=dinic_dfs(v,min(f,e.cap-e.f));
31          if(df>0){e.f+=df;g[v][e.rev].f-=df;return df;}
32        }
33      }
34      return 0;
35    }
36    ll max_flow(int _src, int _dst){
37      src=_src;dst=_dst;
38      ll result=0;
39      while(dinic_bfs()){
40        fill(ALL(work),0);
41        while(ll delta=dinic_dfs(src,INF))result+=delta;
42      }
43      return result;
44    }
45  };
```

4.2 Edmon

```
1  struct Edmons{
2      #define ll long long
3      int n;
4      vector<int>d;
5      vector<tuple<int,ll,ll>>edges;
6      vector<vector<int>> adj;
7      vector<pair<int,int>>cam;
8      Edmons(int _n):adj(_n+1),n(_n){}
9      ll sentFlow(int s,int t,ll f){
10         if(s==t)return f;
11         auto &[u,idx]=cam[t];
12         auto cap=get<1>(edges[idx]),&flow=get<2>(edges[idx]);
13         ll push=sentFlow(s,u,min(cap-flow,f));
14         flow+=push;
15         auto &flowr=get<2>(edges[idx^1]);
16         flowr-=push;
17         return push;
18      }
19      bool bfs(int s,int t){
20         d.assign(n+1,-1); d[s]=0;
21         cam.assign(n+1,{-1,-1});
22         queue<int> q({s});
```

```
23         while(!q.empty()){
24             int u=q.front();
25             q.pop();
26             for(auto idx:adj[u]){
27                 auto &v=get<0>(edges[idx]);auto &cap=get<1>(edges[idx])
                         ,&flow=get<2>(edges[idx]);
28                 if(cap-flow>0 && d[v]==-1) d[v]=d[u]+1,cam[v]={u,idx},q.
                         push(v);
29             }
30         }
31         return d[t]!=-1;
32     }
33     ll maxFlow(int s,int t){
34         ll flow=0;
35         while(bfs(s,t)){
36             ll push=sentFlow(s,t,1e18);
37             if(!push) return flow;
38             flow+=push;
39         }
40         return flow;
41     }
42     void addEdge(int u,int v, ll c, bool dire=true){
43         if(u==v) return;
44         edges.emplace_back(v,c,0);
45         adj[u].push_back(edges.size()-1);
46         edges.emplace_back(u,(dire?0:c),0);
47         adj[v].push_back(edges.size()-1);
48     }
49  };
```

5 Geometria

```
1  #include <bits/stdc++.h>
2  using namespace std;
3  using ld = long double;
4  const ld eps = 1e-9, inf = numeric_limits<ld>::max(), pi = acos(-1);
5  // For use with integers, just set eps=0 and everything remains the same
6  bool geq(ld a, ld b){return a-b >= -eps;}     //a >= b
7  bool leq(ld a, ld b){return b-a >= -eps;}     //a <= b
8  bool ge(ld a, ld b){return a-b > eps;}        //a > b
9  bool le(ld a, ld b){return b-a > eps;}        //a < b
10 bool eq(ld a, ld b){return abs(a-b) <= eps;}  //a == b
11 bool neq(ld a, ld b){return abs(a-b) > eps;}  //a != b
```

```
12
13   struct point{
14     ld x, y;
15     point(): x(0), y(0){}
16     point(ld x, ld y): x(x), y(y){}
17
18     point operator+(const point & p) const{return point(x + p.x, y + p.y)
           ;}
19     point operator-(const point & p) const{return point(x - p.x, y - p.y)
           ;}
20     point operator*(const ld & k) const{return point(x * k, y * k);}
21     point operator/(const ld & k) const{return point(x / k, y / k);}
22
23     point operator+=(const point & p){*this = *this + p; return *this;}
24     point operator-=(const point & p){*this = *this - p; return *this;}
25     point operator*=(const ld & p){*this = *this * p; return *this;}
26     point operator/=(const ld & p){*this = *this / p; return *this;}
27
28     point rotate(const ld & a) const{return point(x*cos(a) - y*sin(a), x*
           sin(a) + y*cos(a));}
29     point perp() const{return point(-y, x);}
30     ld ang() const{
31       ld a = atan2l(y, x); a += le(a, 0) ? 2*pi : 0; return a;
32     }
33     ld dot(const point & p) const{return x * p.x + y * p.y;}
34     ld cross(const point & p) const{return x * p.y - y * p.x;}
35     ld norm() const{return x * x + y * y;}
36     ld length() const{return sqrtl(x * x + y * y);}
37     point unit() const{return (*this) / length();}
38
39     bool operator==(const point & p) const{return eq(x, p.x) && eq(y, p.y)
           ;}
40     bool operator!=(const point & p) const{return !(*this == p);}
41     bool operator<(const point & p) const{return le(x, p.x) || (eq(x, p.x)
           && le(y, p.y));}
42     bool operator>(const point & p) const{return ge(x, p.x) || (eq(x, p.x)
           && ge(y, p.y));}
43     bool half(const point & p) const{return le(p.cross(*this), 0) || (eq(p
           .cross(*this), 0) && le(p.dot(*this), 0));}
44   };
45
46   istream &operator>>(istream &is, point & p){return is >> p.x >> p.y;}
47   ostream &operator<<(ostream &os, const point & p){return os << "(" << p.
```

```
       x << ",␣" << p.y << ")";}
48
49   int sgn(ld x){
50     if(ge(x, 0)) return 1;
51     if(le(x, 0)) return -1;
52     return 0;
53   }
54
55   void polarSort(vector<point> & P, const point & o, const point & v){
56     //sort points in P around o, taking the direction of v as first angle
57     sort(P.begin(), P.end(), [&](const point & a, const point & b){
58       return point((a - o).half(v), 0) < point((b - o).half(v), (a - o).
             cross(b - o));
59     });
60   }
61
62   bool pointInLine(const point & a, const point & v, const point & p){
63     //line a+tv, point p
64     return eq((p - a).cross(v), 0);
65   }
66
67   bool pointInSegment(const point & a, const point & b, const point & p){
68     //segment ab, point p
69     return pointInLine(a, b - a, p) && leq((a - p).dot(b - p), 0);
70   }
71
72   int intersectLinesInfo(const point & a1, const point & v1, const point &
         a2, const point & v2){
73     //lines a1+tv1 and a2+tv2
74     ld det = v1.cross(v2);
75     if(eq(det, 0)){
76       if(eq((a2 - a1).cross(v1), 0)){
77         return -1; //infinity points
78       }else{
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
              , 0: no point
105     }
106   }
107
108   int intersectSegmentsInfo(const point & a, const point & b, const point
          & c, const point & d){
109     //segment ab, segment cd
110     point v1 = b - a, v2 = d - c;
111     int t = sgn(v1.cross(c - a)), u = sgn(v1.cross(d - a));
112     if(t == u){
113       if(t == 0){
114         if(pointInSegment(a, b, c) || pointInSegment(a, b, d) ||
                pointInSegment(c, d, a) || pointInSegment(c, d, b)){
115           return -1; //infinity points
116         }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()
                - 1] - P[i]), 0)){
161         U.pop_back();
162       }
163       U.push_back(P[i]);
164     }
165     L.pop_back();
```

```
166      U.pop_back();
167      L.insert(L.end(), U.begin(), U.end());
168      return L;
169    }
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)) >
             0;
183    }
184
185    int pointInPolygon(const vector<point> & P, const point & p){
186      if(pointInPerimeter(P, p)){
187        return -1; //point in the perimeter
188      }
189      int n = P.size();
190      int rays = 0;
191      for(int i = 0; i < n; i++){
192        rays += crossesRay(P[i], P[(i + 1) % n], p);
193      }
194      return rays & 1; //0: point outside, 1: point inside
195    }
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
             [0])) - 1;
217        if(det == -2) det = 1;
218        return det;
219      }
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
```

```
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      return {c, r};
```

```
481  }
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
                ->onext->dest())){
721           QuadEdge* t = lcand->onext;
722           delete_edge(lcand);
723           lcand = t;
724         }
725       }
726       QuadEdge* rcand = basel->oprev();
727       if(valid(rcand)){
728         while(in_circle(basel->dest(), basel->origin, rcand->dest(), rcand
```

```
728                    ->oprev()->dest())){
729              QuadEdge* t = rcand->oprev();
730              delete_edge(rcand);
731              rcand = t;
732            }
733          }
734          if(!valid(lcand) && !valid(rcand))
735            break;
736          if(!valid(lcand) || (valid(rcand) && in_circle(lcand->dest(), lcand
                 ->origin, rcand->origin, rcand->dest())))
737            basel = connect(rcand, basel->rev());
738          else
739            basel = connect(basel->rev(), lcand->rev());
740        }
741        return {ldo, rdo};
742      }
743
744      vector<tuple<point, point, point>> delaunay(vector<point> & P){
745        sort(P.begin(), P.end());
746        auto res = build_tr(0, (int)P.size() - 1, P);
747        QuadEdge* e = res.first;
748        vector<QuadEdge*> edges = {e};
749        while(le((e->dest() - e->onext->dest()).cross(e->origin - e->onext->
                 dest()), 0))
750          e = e->onext;
751        auto add = [&P, &e, &edges](){
752          QuadEdge* curr = e;
753          do{
754            curr->used = true;
755            P.push_back(curr->origin);
756            edges.push_back(curr->rev());
757            curr = curr->lnext();
758          }while(curr != e);
759        };
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.
                 r);
814        if(cruce.size() == 2){
815          ld a1 = (cruce[0] - prev.c).ang();
816          ld a2 = (cruce[1] - prev.c).ang();
817          ld b1 = (cruce[1] - nuevo.c).ang();
818          ld b2 = (cruce[0] - nuevo.c).ang();
819          if(a1 < a2){
820            prev.disable(a1, a2);
821          }else{
822            prev.disable(a1, 2*pi);
823            prev.disable(0, a2);
824          }
825          if(b1 < b2){
826            nuevo.disable(b1, b2);
827          }else{
828            nuevo.disable(b1, 2*pi);
829            nuevo.disable(0, b2);
830          }
831        }
832      }
833    }
834    valid.push_back(nuevo);
835  }
836  ld ans = 0;
837  for(const circ & curr : valid){
838    for(const auto & range : curr.getActive()){
839      ld l, r;
840      tie(l, r) = range;
841      ans += curr.r*(curr.c.x * (sin(r) - sin(l)) - curr.c.y * (cos(r) -
               cos(l))) + curr.r*curr.r*(r-l);
842    }
843  }
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
           outside
858     return le(v.cross(p - a), 0);
859   }
860
861   bool inside(const point& p) const{ // test if point p is inside or in
           the boundary
862     return geq(v.cross(p - a), 0);
863   }
864
865   bool operator<(const plane& p) const{ // sort by angle
866     auto lhs = make_tuple(v.half({1, 0}), ld(0), v.cross(p.a - a));
867     auto rhs = make_tuple(p.v.half({1, 0}), v.cross(p.v), ld(0));
868     return lhs < rhs;
869   }
870
871   bool operator==(const plane& p) const{ // paralell and same directions
           , not really equal
872     return eq(v.cross(p.v), 0) && ge(v.dot(p.v), 0);
873   }
874 };
875
876 vector<point> halfPlaneIntersection(vector<plane> planes){
877   planes.push_back({{0, -inf}, {1, 0}});
878   planes.push_back({{inf, 0}, {0, 1}});
879   planes.push_back({{0, inf}, {-1, 0}});
880   planes.push_back({{-inf, 0}, {0, -1}});
881   sort(planes.begin(), planes.end());
882   planes.erase(unique(planes.begin(), planes.end()), planes.end());
883   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.
```

```
891          size()-1]));
             }
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() %
            1000000000, rand() % 1000000000);
970     polarSort(P, o, v);
971     auto ang = [&](point p){
972       ld th = atan2(p.y, p.x);
973       if(th < 0) th += acosl(-1)*2;
974       ld t = atan2(v.y, v.x);
975       if(t < 0) t += acosl(-1)*2;
976       if(th < t) th += acosl(-1)*2;
977       return th;
978     };
979     for(int i = 0; i < P.size()-1; ++i){
980       assert(leq(ang(P[i] - o), ang(P[i+1] - o)));
981     }*/
982     return 0;
983 }
```

6   Varios

6.1   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 }
```

6.2   String a vector¡int¿

```
1  //Convertir una cadena de numeros separados por " " en vector de enteros
2  //Leer varias de esas querys
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 }
```

6.3   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)));
```

6.4   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         for(int i=0;i<s;i++){
49             if(ids[i]==ids[i+s]) return false;
50             val[i]=(ids[i]>ids[i+s]?0:1);
51         }
52         return true;
53     }
54 };
```

6.5   Bits

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

6.6   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);
```

6.7   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
```

```cpp
int get_answer();  // TODO: extract the current answer of the data
    structure

int block_size;//Recomended sqrt(n)

struct Query {
    int l, r, idx;
    bool operator<(Query other) const
    {
        return make_pair(l / block_size, r) <
                make_pair(other.l / block_size, other.r);
    }
};

vector<int> mo_s_algorithm(vector<Query> queries) {
    vector<int> answers(queries.size());
    sort(queries.begin(), queries.end());

    // TODO: initialize data structure

    int cur_l = 0;
    int cur_r = -1;
    // invariant: data structure will always reflect the range [cur_l,
        cur_r]
    for (Query q : queries) {
        while (cur_l > q.l) {
            cur_l--;
            add(cur_l);
        }
        while (cur_r < q.r) {
            cur_r++;
            add(cur_r);
        }
        while (cur_l < q.l) {
            remove(cur_l);
            cur_l++;
        }
        while (cur_r > q.r) {
            remove(cur_r);
            cur_r--;
        }
        answers[q.idx] = get_answer();
    }
```

```cpp
    return answers;
}
```

6.8   PBS

```cpp
    1.Crear un arreglo con para procesar
    2.Para cada elemento inicialicar 1 l y en q+1 r;
    for(int i=1;i<=n;i++){
        m[i].x=1,m[i].y=q+1;
    }
    bool flag=true;
    while(flag){
        flag=false;
        // limpiar la estructura de datos
        for(int i=0;i<=4*n+5;i++)st[i]=0,lazy[i]=0;
        for(int i=1;i<=n;i++)
            //Si es diefente l!=r se procesa;
            if(m[i].x!=m[i].y){ flag=true;tocheck[(m[i].x+m[i].y)/2].
                push_back(i);}
        for(int i=1;i<=q;i++){
            if(!flag)break;
            // Se aplican las queries
            update(0,n-1,qs[i].x,qs[i].y,qs[i].z,0);
            update(0,n-1,qs[i].x,qs[i].x,qs[i].k,0);
            while(tocheck[i].size()){
                int id=tocheck[i].back();
                tocheck[i].pop_back();
                // Se obserba si se cumblio la caondicion para el
                        elemeto
                if(ai[id]<=query(0,n-1,S[id],S[id],0)) m[id].y=i;
                else m[id].x=i+1;
            }
        }
    }
    // Solo se imprime
    for(int i=1;i<=n;i++){
        if(m[i].x<=q) cout<<m[i].x<<endl;
        else cout<<-1<<endl;
    }
```