# 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) 1993-2021
%% The LaTeX 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

```
//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){
    ST.resize(N << 1);
    for(int i = 0; i < N; ++i)
      ST[N + i] = arr[i];     //Dato normal
      ST[N + i] = creaNodo(); //Dato compuesto
    for(int i = N - 1; i > 0; --i)
      ST[i] = ST[i << 1] + ST[i << 1 | 1];        //Dato normal
      ST[i] = merge(ST[i << 1] , ST[i << 1 | 1]); //Dato compuesto
  }

  //Actualizacion de un elemento en la posicion i
  void update(int i, T value){
    ST[i += N] = value;     //Dato normal
    ST[i += N] = creaNodo();//Dato compuesto
    while(i >>= 1)
      ST[i] = ST[i << 1] + ST[i << 1 | 1];        //Dato normal
      ST[i] = merge(ST[i << 1] , ST[i << 1 | 1]); //Dato compuesto
  }

  //query en [l, r]
  T query(int l, int r){
    T res = 0;  //Dato normal
    nodo resl = creaNodo(), resr = creaNodo();//Dato compuesto
    for(l += N, r += N; l <= r; l >>= 1, r >>= 1){
      if(l & 1)       res += ST[l++]; //Dato normal
      if(!(r & 1))    res += ST[r--]; //Dato normal

      if(l & 1)       resl = merge(resl,ST[l++]); //Dato compuesto
      if(!(r & 1))    resr = merge(ST[r--],resr); //Dato compuesto
    }
    return res;                 //Dato normal
    return merge(resl,resr);    //Dato compuesto
  }

  //Para estas querys es necesario que el st tenga el tam de la siguiente
      potencia de 2
  //ll nT = 1;
  // while(nT<n) nT<<=1;
  //vector<int> a(nT,0);

  //Encontrar k-esimo 1 en un st de 1's
  int Kth_One(int k) {
    int i = 0, s = N >> 1;
    for(int p = 2; p < 2 * N; p <<= 1, s >>= 1) {
```

```
      if(k < ST[p]) continue;
      k -= ST[p++]; i += s;
    }
    return i;
  }

  //i del primer elemento >= k en todo el arr
  int atLeastX(int k){
    int i = 0, s = N >> 1;
    for(int p = 2; p < 2 * N; p <<= 1, s >>= 1) {
      if(ST[p] < k) p++, i += s;
    }
    if(ST[N + i] < k) i = -1;
    return i;
  }

  //i del primer elemento >= k en [l,fin]
  //Uso atLeastX(k,l,1,nT)
  int atLeastX(int x, int l, int p, int s) {
    if(ST[p] < x or s <= l) return -1;
    if((p << 1) >= 2 * N)
      return (ST[p] >= x) - 1;
    int i = atLeastX(x, l, p << 1, s >> 1);
    if(i != -1) return i;
    i = atLeastX(x, l - (s >> 1), p << 1 | 1, s >> 1);
    if(i == -1) return -1;
    return (s >> 1) + i;
  }
};
```

## 1.4 Segment Tree Lazy 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) 1993-2021
%% The LaTeX 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'.
```

## 1.5 Segment Tree Lazy Iterativo

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

```

```
  //Creacion a partir de un arreglo
  SegmentTreeLazy(int n, vector<T> &a): N(n){
    //En caso de inicializar en cero o algo similar, revisar que la
        construccion tenga su respectivo neutro mult y 1
    ST.resize(N << 1);
    d.resize(N);
    h = 64 - __builtin_clzll(n);

    for(int i = 0; i < N; ++i)
      ST[N + i] = a[i];
    //Construir el st sobre la query que se necesita
    for(int i = N - 1; i > 0; --i)
      ST[i] = min(ST[i << 1] , ST[i << 1 | 1]);
  }

  //Modificar de acuerdo al tipo modificacion requerida, +,*,|,^,etc
  void apply(int p, T value) {
    ST[p] += value;
    if(p<N) d[p]+= value;
  }

  // Modifica valores de los padres de p
  //Modificar de acuerdo al tipo modificacion requerida, +,*,|,^,etc y a la
      respectiva query
  void build(int p){
    while(p>1){
      p >>= 1;
      ST[p] = min(ST[p << 1], ST[p << 1 | 1]) + d[p];
      //ST[p] = (ST[p << 1] & ST[p << 1 | 1]) | d[p]; Ejemplos con bitwise
    }
  }

  // Propagacion desde la raiz a p
  void push(int p){
    for (int s = h; s > 0; --s) {
      int i = p >> s;
      if (d[i] != 0) {
        apply(i << 1, d[i]);
        apply(i << 1 | 1, d[i]);
        d[i] = 0;   //Tener cuidado si estoy haciendo multiplicaciones
      }
    }
  }


  // Sumar v a cada elemento en el intervalo [l, r)
  void increment(int l, int r, T value) {
    l += N, r += N;
    int l0 = l, r0 = r;
    for (; l < r; l >>= 1, r >>= 1) {
      if(l & 1) apply(l++, value);
      if(r & 1) apply(--r, value);
    }
    build(l0);
    build(r0 - 1);
  }

  // min en el intervalo [l, r)
  T range_min(int l, int r) {
    l += N, r += N;
    push(l);
    push(r - 1);
    T res = LLONG_MAX;
    //T res = (1 << 30) - 1;    Requerir operacion and
    for (; l < r; l >>= 1, r >>= 1) {
      if(l & 1) res = min(res, ST[l++]);
      //if(res >= mod) res -= mod;
      if(r & 1) res = min(res, ST[--r]);
      //if(res >= mod) res -= mod;
    }
    return res;
  }

};
```

## 1.6 Rope

```
#include <ext/rope>
using namespace __gnu_cxx;
rope<int> s;
// Sequence with O(log(n)) random access, insert, erase at any position
// s.push_back(x);
// s.insert(i,r) // insert rope r at position i
// s.erase(i,k) // erase subsequence [i,i+k)
// s.substr(i,k) // return new rope corresponding to subsequence [i,i+k)
// s[i] // access ith element (cannot modify)
// s.mutable_reference_at(i) // acces ith element (allows modification)
```

```
// s.begin() and s.end() are const iterators (use mutable_begin(),
    mutable_end() to allow modification)
```

## 1.7 Ordered Set

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

## 1.8 Union Find

```
vector<pair<int,int>>ds(MAX,{-1,0});
// Solo siu requeires los elementos del union find, utiliza
// dsext en caso contrario borrarlo
list<int>dsext[MAX];
void init(int n){
    for(int i=0;i<n;i++)dsext[i].push_back(i);
}
int find(int x){
    if(-1==ds[x].first) return x;
    return ds[x].first=find(ds[x].first);
}
bool unionDs(int x, int y){
    int px=find(x),py=find(y);
    int &rx=ds[px].second,&ry=ds[py].second;
    if(px==py) return false;
    else{
        if(rx>ry){
            ds[py].first=px;
        }
        else{
            ds[px].first=py;
            if(rx==ry) ry+=1;
        }
    }
    return true;
}
```

# 2 Geometria

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

struct point{
  ld x, y;
  point(): x(0), y(0){}
  point(ld x, ld y): x(x), y(y){}

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

  point operator+=(const point & p){*this = *this + p; return *this;}
  point operator-=(const point & p){*this = *this - p; return *this;}
  point operator*=(const ld & p){*this = *this * p; return *this;}
  point operator/=(const ld & p){*this = *this / p; return *this;}

  point rotate(const ld & a) const{return point(x*cos(a) - y*sin(a), x*sin(
      a) + y*cos(a));}
  point perp() const{return point(-y, x);}
  ld ang() const{
    ld a = atan2l(y, x); a += le(a, 0) ? 2*pi : 0; return a;
  }
  ld dot(const point & p) const{return x * p.x + y * p.y;}
  ld cross(const point & p) const{return x * p.y - y * p.x;}
  ld norm() const{return x * x + y * y;}
  ld length() const{return sqrtl(x * x + y * y);}
  point unit() const{return (*this) / length();}

  bool operator==(const point & p) const{return eq(x, p.x) && eq(y, p.y);}
  bool operator!=(const point & p) const{return !(*this == p);}
  bool operator<(const point & p) const{return le(x, p.x) || (eq(x, p.x) &&
       le(y, p.y));}
```

```
  bool operator>(const point & p) const{return ge(x, p.x) || (eq(x, p.x) && ge(y, p.y));}
  bool half(const point & p) const{return le(p.cross(*this), 0) || (eq(p.cross(*this), 0) && le(p.dot(*this), 0));}
};

istream &operator>>(istream &is, point & p){return is >> p.x >> p.y;}
ostream &operator<<(ostream &os, const point & p){return os << "(" << p.x << ", " << p.y << ")";}

int sgn(ld x){
  if(ge(x, 0)) return 1;
  if(le(x, 0)) return -1;
  return 0;
}

void polarSort(vector<point> & P, const point & o, const point & v){
  //sort points in P around o, taking the direction of v as first angle
  sort(P.begin(), P.end(), [&](const point & a, const point & b){
    return point((a - o).half(v), 0) < point((b - o).half(v), (a - o).cross(b - o));
  });
}

bool pointInLine(const point & a, const point & v, const point & p){
  //line a+tv, point p
  return eq((p - a).cross(v), 0);
}

bool pointInSegment(const point & a, const point & b, const point & p){
  //segment ab, point p
  return pointInLine(a, b - a, p) && leq((a - p).dot(b - p), 0);
}

int intersectLinesInfo(const point & a1, const point & v1, const point & a2, const point & v2){
  //lines a1+tv1 and a2+tv2
  ld det = v1.cross(v2);
  if(eq(det, 0)){
    if(eq((a2 - a1).cross(v1), 0)){
      return -1; //infinity points
    }else{
      return 0; //no points
    }
  }else{
    return 1; //single point
  }
}

point intersectLines(const point & a1, const point & v1, const point & a2, const point & v2){
  //lines a1+tv1, a2+tv2
  //assuming that they intersect
  ld det = v1.cross(v2);
  return a1 + v1 * ((a2 - a1).cross(v2) / det);
}

int intersectLineSegmentInfo(const point & a, const point & v, const point & c, const point & d){
  //line a+tv, segment cd
  point v2 = d - c;
  ld det = v.cross(v2);
  if(eq(det, 0)){
    if(eq((c - a).cross(v), 0)){
      return -1; //infinity points
    }else{
      return 0; //no point
    }
  }else{
    return sgn(v.cross(c - a)) != sgn(v.cross(d - a)); //1: single point, 0: no point
  }
}

int intersectSegmentsInfo(const point & a, const point & b, const point & c, const point & d){
  //segment ab, segment cd
  point v1 = b - a, v2 = d - c;
  int t = sgn(v1.cross(c - a)), u = sgn(v1.cross(d - a));
  if(t == u){
    if(t == 0){
      if(pointInSegment(a, b, c) || pointInSegment(a, b, d) ||
          pointInSegment(c, d, a) || pointInSegment(c, d, b)){
        return -1; //infinity points
      }else{
        return 0; //no point
```

```
      }
    }else{
      return 0; //no point
    }
  }else{
    return sgn(v2.cross(a - c)) != sgn(v2.cross(b - c)); //1: single point,
        0: no point
  }
}

ld distancePointLine(const point & a, const point & v, const point & p){
  //line: a + tv, point p
  return abs(v.cross(p - a)) / v.length();
}

ld perimeter(vector<point> & P){
  int n = P.size();
  ld ans = 0;
  for(int i = 0; i < n; i++){
    ans += (P[i] - P[(i + 1) % n]).length();
  }
  return ans;
}

ld area(vector<point> & P){
  int n = P.size();
  ld ans = 0;
  for(int i = 0; i < n; i++){
    ans += P[i].cross(P[(i + 1) % n]);
  }
  return abs(ans / 2);
}

vector<point> convexHull(vector<point> P){
  sort(P.begin(), P.end());
  vector<point> L, U;
  for(int i = 0; i < P.size(); i++){
    while(L.size() >= 2 && leq((L[L.size() - 2] - P[i]).cross(L[L.size() -
        1] - P[i]), 0)){
      L.pop_back();
    }
    L.push_back(P[i]);
  }
  for(int i = P.size() - 1; i >= 0; i--){
    while(U.size() >= 2 && leq((U[U.size() - 2] - P[i]).cross(U[U.size() -
        1] - P[i]), 0)){
      U.pop_back();
    }
    U.push_back(P[i]);
  }
  L.pop_back();
  U.pop_back();
  L.insert(L.end(), U.begin(), U.end());
  return L;
}

bool pointInPerimeter(const vector<point> & P, const point & p){
  int n = P.size();
  for(int i = 0; i < n; i++){
    if(pointInSegment(P[i], P[(i + 1) % n], p)){
      return true;
    }
  }
  return false;
}

bool crossesRay(const point & a, const point & b, const point & p){
  return (geq(b.y, p.y) - geq(a.y, p.y)) * sgn((a - p).cross(b - p)) > 0;
}

int pointInPolygon(const vector<point> & P, const point & p){
  if(pointInPerimeter(P, p)){
    return -1; //point in the perimeter
  }
  int n = P.size();
  int rays = 0;
  for(int i = 0; i < n; i++){
    rays += crossesRay(P[i], P[(i + 1) % n], p);
  }
  return rays & 1; //0: point outside, 1: point inside
}

//point in convex polygon in O(log n)
//make sure that P is convex and in ccw
//before the queries, do the preprocess on P:
// rotate(P.begin(), min_element(P.begin(), P.end()), P.end());
```

```
// int right = max_element(P.begin(), P.end()) - P.begin();
//returns 0 if p is outside, 1 if p is inside, -1 if p is in the perimeter
int pointInConvexPolygon(const vector<point> & P, const point & p, int
    right){
  if(p < P[0] || P[right] < p) return 0;
  int orientation = sgn((P[right] - P[0]).cross(p - P[0]));
  if(orientation == 0){
    if(p == P[0] || p == P[right]) return -1;
    return (right == 1 || right + 1 == P.size()) ? -1 : 1;
  }else if(orientation < 0){
    auto r = lower_bound(P.begin() + 1, P.begin() + right, p);
    int det = sgn((p - r[-1]).cross(r[0] - r[-1])) - 1;
    if(det == -2) det = 1;
    return det;
  }else{
    auto l = upper_bound(P.rbegin(), P.rend() - right - 1, p);
    int det = sgn((p - l[0]).cross((l == P.rbegin() ? P[0] : l[-1]) - l[0])
        ) - 1;
    if(det == -2) det = 1;
    return det;
  }
}









vector<point> cutPolygon(const vector<point> & P, const point & a, const
    point & v){
  //returns the part of the convex polygon P on the left side of line a+tv
  int n = P.size();
  vector<point> lhs;
  for(int i = 0; i < n; ++i){
    if(geq(v.cross(P[i] - a), 0)){
      lhs.push_back(P[i]);
    }
    if(intersectLineSegmentInfo(a, v, P[i], P[(i+1)%n]) == 1){
      point p = intersectLines(a, v, P[i], P[(i+1)%n] - P[i]);
      if(p != P[i] && p != P[(i+1)%n]){
        lhs.push_back(p);
      }
    }
  }
  return lhs;
}









point centroid(vector<point> & P){
  point num;
  ld den = 0;
  int n = P.size();
  for(int i = 0; i < n; ++i){
    ld cross = P[i].cross(P[(i + 1) % n]);
    num += (P[i] + P[(i + 1) % n]) * cross;
    den += cross;
  }
  return num / (3 * den);
}

vector<pair<int, int>> antipodalPairs(vector<point> & P){
  vector<pair<int, int>> ans;
  int n = P.size(), k = 1;
  auto f = [&](int u, int v, int w){return abs((P[v%n]-P[u%n]).cross(P[w%n
      ]-P[u%n]));};
  while(ge(f(n-1, 0, k+1), f(n-1, 0, k))) ++k;
  for(int i = 0, j = k; i <= k && j < n; ++i){
    ans.emplace_back(i, j);
    while(j < n-1 && ge(f(i, i+1, j+1), f(i, i+1, j)))
      ans.emplace_back(i, ++j);
  }
  return ans;
}
```

```
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
      ){
```

```
  //find circle that passes through points p, q, r
  point c = intersectLines((n + m) / 2, (n - m).perp(), (p + n) / 2, (p - n
      ).perp());
  ld r = (c - m).length();
  return {c, r};
}

vector<point> intersectionCircles(const point & c1, ld r1, const point & c2
    , ld r2){
  //circle 1 with center c1 and radius r1
  //circle 2 with center c2 and radius r2
  point d = c2 - c1;
  ld d2 = d.norm();
  if(eq(d2, 0)) return {}; //concentric circles
  ld pd = (d2 + r1*r1 - r2*r2) / 2;
  ld h2 = r1*r1 - pd*pd/d2;
  point p = c1 + d*pd/d2;
  if(eq(h2, 0)) return {p}; //circles touch at one point
  else if(le(h2, 0)) return {}; //circles don't intersect
  else{
    point u = d.perp() * sqrt(h2/d2);
    return {p - u, p + u};
  }
}

int circleInsideCircle(const point & c1, ld r1, const point & c2, ld r2){
  //test if circle 2 is inside circle 1
  //returns "-1" if 2 touches internally 1, "1" if 2 is inside 1, "0" if
      they overlap
  ld l = r1 - r2 - (c1 - c2).length();
  return (ge(l, 0) ? 1 : (eq(l, 0) ? -1 : 0));
}

int circleOutsideCircle(const point & c1, ld r1, const point & c2, ld r2){
  //test if circle 2 is outside circle 1
  //returns "-1" if they touch externally, "1" if 2 is outside 1, "0" if
      they overlap
  ld l = (c1 - c2).length() - (r1 + r2);
  return (ge(l, 0) ? 1 : (eq(l, 0) ? -1 : 0));
}

int pointInCircle(const point & c, ld r, const point & p){
  //test if point p is inside the circle with center c and radius r
  //returns "0" if it's outside, "-1" if it's in the perimeter, "1" if it's
      inside
  ld l = (p - c).length() - r;
  return (le(l, 0) ? 1 : (eq(l, 0) ? -1 : 0));
}

vector<vector<point>> tangents(const point & c1, ld r1, const point & c2,
    ld r2, bool inner){
  //returns a vector of segments or a single point
  if(inner) r2 = -r2;
  point d = c2 - c1;
  ld dr = r1 - r2, d2 = d.norm(), h2 = d2 - dr*dr;
  if(eq(d2, 0) || le(h2, 0)) return {};
  point v = d*dr/d2;
  if(eq(h2, 0)) return {{c1 + v*r1}};
  else{
    point u = d.perp()*sqrt(h2)/d2;
    return {{c1 + (v - u)*r1, c2 + (v - u)*r2}, {c1 + (v + u)*r1, c2 + (v +
        u)*r2}};
  }
}

ld signed_angle(const point & a, const point & b){
  return sgn(a.cross(b)) * acosl(a.dot(b) / (a.length() * b.length()));
}

ld intersectPolygonCircle(const vector<point> & P, const point & c, ld r){
  //Gets the area of the intersection of the polygon with the circle
  int n = P.size();
  ld ans = 0;
  for(int i = 0; i < n; ++i){
    point p = P[i], q = P[(i+1)%n];
    bool p_inside = (pointInCircle(c, r, p) != 0);
    bool q_inside = (pointInCircle(c, r, q) != 0);
    if(p_inside && q_inside){
      ans += (p - c).cross(q - c);
    }else if(p_inside && !q_inside){
      point s1 = intersectSegmentCircle(p, q, c, r)[0];
      point s2 = intersectSegmentCircle(c, q, c, r)[0];
      ans += (p - c).cross(s1 - c) + r*r * signed_angle(s1 - c, s2 - c);
    }else if(!p_inside && q_inside){
      point s1 = intersectSegmentCircle(c, p, c, r)[0];
      point s2 = intersectSegmentCircle(p, q, c, r)[0];
```

```
      ans += (s2 - c).cross(q - c) + r*r * signed_angle(s1 - c, s2 - c);
    }else{
      auto info = intersectSegmentCircle(p, q, c, r);
      if(info.size() <= 1){
        ans += r*r * signed_angle(p - c, q - c);
      }else{
        point s2 = info[0], s3 = info[1];
        point s1 = intersectSegmentCircle(c, p, c, r)[0];
        point s4 = intersectSegmentCircle(c, q, c, r)[0];
        ans += (s2 - c).cross(s3 - c) + r*r * (signed_angle(s1 - c, s2 - c)
             + signed_angle(s3 - c, s4 - c));
      }
    }
  }
  return abs(ans)/2;
}

pair<point, ld> mec2(vector<point> & S, const point & a, const point & b,
    int n){
  ld hi = inf, lo = -hi;
  for(int i = 0; i < n; ++i){
    ld si = (b - a).cross(S[i] - a);
    if(eq(si, 0)) continue;
    point m = getCircle(a, b, S[i]).first;
    ld cr = (b - a).cross(m - a);
    if(le(si, 0)) hi = min(hi, cr);
    else lo = max(lo, cr);
  }
  ld v = (ge(lo, 0) ? lo : le(hi, 0) ? hi : 0);
  point c = (a + b) / 2 + (b - a).perp() * v / (b - a).norm();
  return {c, (a - c).norm()};
}

pair<point, ld> mec(vector<point> & S, const point & a, int n){
  random_shuffle(S.begin(), S.begin() + n);
  point b = S[0], c = (a + b) / 2;
  ld r = (a - c).norm();
  for(int i = 1; i < n; ++i){
    if(ge((S[i] - c).norm(), r)){
      tie(c, r) = (n == S.size() ? mec(S, S[i], i) : mec2(S, a, S[i], i));
    }
  }
  return {c, r};
}

pair<point, ld> smallestEnclosingCircle(vector<point> S){
  assert(!S.empty());
  auto r = mec(S, S[0], S.size());
  return {r.first, sqrt(r.second)};
}

bool comp1(const point & a, const point & b){
  return le(a.y, b.y);
}
pair<point, point> closestPairOfPoints(vector<point> P){
  sort(P.begin(), P.end(), comp1);
  set<point> S;
  ld ans = inf;
  point p, q;
  int pos = 0;
  for(int i = 0; i < P.size(); ++i){
    while(pos < i && geq(P[i].y - P[pos].y, ans)){
      S.erase(P[pos++]);
    }
    auto lower = S.lower_bound({P[i].x - ans - eps, -inf});
    auto upper = S.upper_bound({P[i].x + ans + eps, -inf});
    for(auto it = lower; it != upper; ++it){
      ld d = (P[i] - *it).length();
      if(le(d, ans)){
        ans = d;
        p = P[i];
        q = *it;
      }
    }
    S.insert(P[i]);
  }
  return {p, q};
}

struct vantage_point_tree{
  struct node
  {
    point p;
    ld th;
    node *l, *r;
  }*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;
```

```
  QuadEdge* rot = nullptr;
  QuadEdge* onext = nullptr;
  bool used = false;
  QuadEdge* rev() const{return rot->rot;}
  QuadEdge* lnext() const{return rot->rev()->onext->rot;}
  QuadEdge* oprev() const{return rot->onext->rot;}
  point dest() const{return rev()->origin;}
};

QuadEdge* make_edge(const point & from, const point & to){
  QuadEdge* e1 = new QuadEdge;
  QuadEdge* e2 = new QuadEdge;
  QuadEdge* e3 = new QuadEdge;
  QuadEdge* e4 = new QuadEdge;
  e1->origin = from;
  e2->origin = to;
  e3->origin = e4->origin = inf_pt;
  e1->rot = e3;
  e2->rot = e4;
  e3->rot = e2;
  e4->rot = e1;
  e1->onext = e1;
  e2->onext = e2;
  e3->onext = e4;
  e4->onext = e3;
  return e1;
}

void splice(QuadEdge* a, QuadEdge* b){
  swap(a->onext->rot->onext, b->onext->rot->onext);
  swap(a->onext, b->onext);
}

void delete_edge(QuadEdge* e){
  splice(e, e->oprev());
  splice(e->rev(), e->rev()->oprev());
  delete e->rot;
  delete e->rev()->rot;
  delete e;
  delete e->rev();
}

QuadEdge* connect(QuadEdge* a, QuadEdge* b){
  QuadEdge* e = make_edge(a->dest(), b->origin);
  splice(e, a->lnext());
  splice(e->rev(), b);
  return e;
}

bool left_of(const point & p, QuadEdge* e){
  return ge((e->origin - p).cross(e->dest() - p), 0);
}

bool right_of(const point & p, QuadEdge* e){
  return le((e->origin - p).cross(e->dest() - p), 0);
}

ld det3(ld a1, ld a2, ld a3, ld b1, ld b2, ld b3, ld c1, ld c2, ld c3) {
  return a1 * (b2 * c3 - c2 * b3) - a2 * (b1 * c3 - c1 * b3) + a3 * (b1 *
      c2 - c1 * b2);
}

bool in_circle(const point & a, const point & b, const point & c, const
    point & d) {
  ld det = -det3(b.x, b.y, b.norm(), c.x, c.y, c.norm(), d.x, d.y, d.norm()
      );
  det += det3(a.x, a.y, a.norm(), c.x, c.y, c.norm(), d.x, d.y, d.norm());
  det -= det3(a.x, a.y, a.norm(), b.x, b.y, b.norm(), d.x, d.y, d.norm());
  det += det3(a.x, a.y, a.norm(), b.x, b.y, b.norm(), c.x, c.y, c.norm());
  return ge(det, 0);
}

pair<QuadEdge*, QuadEdge*> build_tr(int l, int r, vector<point> & P){
  if(r - l + 1 == 2){
    QuadEdge* res = make_edge(P[l], P[r]);
    return {res, res->rev()};
  }
  if(r - l + 1 == 3){
    QuadEdge *a = make_edge(P[l], P[l + 1]), *b = make_edge(P[l + 1], P[r])
        ;
    splice(a->rev(), b);
    int sg = sgn((P[l + 1] - P[l]).cross(P[r] - P[l]));
    if(sg == 0)
      return {a, b->rev()};
    QuadEdge* c = connect(b, a);
    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->
              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{
```

```
  point c;
  ld r;
  circ() {}
  circ(const point & c, ld r): c(c), r(r) {}
  set<pair<ld, ld>> ranges;

  void disable(ld l, ld r){
    ranges.emplace(l, r);
  }

  auto getActive() const{
    vector<pair<ld, ld>> ans;
    ld maxi = 0;
    for(const auto & dis : ranges){
      ld l, r;
      tie(l, r) = dis;
      if(l > maxi){
        ans.emplace_back(maxi, l);
      }
      maxi = max(maxi, r);
    }
    if(!eq(maxi, 2*pi)){
      ans.emplace_back(maxi, 2*pi);
    }
    return ans;
  }
};

ld areaUnionCircles(const vector<circ> & circs){
  vector<circ> valid;
  for(const circ & curr : circs){
    if(eq(curr.r, 0)) continue;
    circ nuevo = curr;
    for(circ & prev : valid){
      if(circleInsideCircle(prev.c, prev.r, nuevo.c, nuevo.r)){
        nuevo.disable(0, 2*pi);
      }else if(circleInsideCircle(nuevo.c, nuevo.r, prev.c, prev.r)){
        prev.disable(0, 2*pi);
      }else{
        auto cruce = interseccionCircles(prev.c, prev.r, nuevo.c, nuevo.r);
        if(cruce.size() == 2){
          ld a1 = (cruce[0] - prev.c).ang();
          ld a2 = (cruce[1] - prev.c).ang();
          ld b1 = (cruce[1] - nuevo.c).ang();
          ld b2 = (cruce[0] - nuevo.c).ang();
          if(a1 < a2){
            prev.disable(a1, a2);
          }else{
            prev.disable(a1, 2*pi);
            prev.disable(0, a2);
          }
          if(b1 < b2){
            nuevo.disable(b1, b2);
          }else{
            nuevo.disable(b1, 2*pi);
            nuevo.disable(0, b2);
          }
        }
      }
    }
    valid.push_back(nuevo);
  }
  ld ans = 0;
  for(const circ & curr : valid){
    for(const auto & range : curr.getActive()){
      ld l, r;
      tie(l, r) = range;
      ans += curr.r*(curr.c.x * (sin(r) - sin(l)) - curr.c.y * (cos(r) -
          cos(l))) + curr.r*curr.r*(r-l);
    }
  }
  return ans/2;
};

struct plane{
  point a, v;
  plane(): a(), v(){}
  plane(const point& a, const point& v): a(a), v(v){}

  point intersect(const plane& p) const{
    ld t = (p.a - a).cross(p.v) / v.cross(p.v);
    return a + v*t;
  }

  bool outside(const point& p) const{ // test if point p is strictly
      outside
```

```
    return le(v.cross(p - a), 0);
  }

  bool inside(const point& p) const{ // test if point p is inside or in the boundary
    return geq(v.cross(p - a), 0);
  }

  bool operator<(const plane& p) const{ // sort by angle
    auto lhs = make_tuple(v.half({1, 0}), ld(0), v.cross(p.a - a));
    auto rhs = make_tuple(p.v.half({1, 0}), v.cross(p.v), ld(0));
    return lhs < rhs;
  }

  bool operator==(const plane& p) const{ // paralell and same directions, not really equal
    return eq(v.cross(p.v), 0) && ge(v.dot(p.v), 0);
  }
};

vector<point> halfPlaneIntersection(vector<plane> planes){
  planes.push_back({{0, -inf}, {1, 0}});
  planes.push_back({{inf, 0}, {0, 1}});
  planes.push_back({{0, inf}, {-1, 0}});
  planes.push_back({{-inf, 0}, {0, -1}});
  sort(planes.begin(), planes.end());
  planes.erase(unique(planes.begin(), planes.end()), planes.end());
  deque<plane> ch;
  deque<point> poly;
  for(const plane& p : planes){
    while(ch.size() >= 2 && p.outside(poly.back())) ch.pop_back(), poly.pop_back();
    while(ch.size() >= 2 && p.outside(poly.front())) ch.pop_front(), poly.pop_front();
    if(p.v.half({1, 0}) && poly.empty()) return {};
    ch.push_back(p);
    if(ch.size() >= 2) poly.push_back(ch[ch.size()-2].intersect(ch[ch.size()-1]));
  }
  while(ch.size() >= 3 && ch.front().outside(poly.back())) ch.pop_back(), poly.pop_back();
  while(ch.size() >= 3 && ch.back().outside(poly.front())) ch.pop_front(), poly.pop_front();
  poly.push_back(ch.back().intersect(ch.front()));
  return vector<point>(poly.begin(), poly.end());
}

vector<point> halfPlaneIntersectionRandomized(vector<plane> planes){
  point p = planes[0].a;
  int n = planes.size();
  random_shuffle(planes.begin(), planes.end());
  for(int i = 0; i < n; ++i){
    if(planes[i].inside(p)) continue;
    ld lo = -inf, hi = inf;
    for(int j = 0; j < i; ++j){
      ld A = planes[j].v.cross(planes[i].v);
      ld B = planes[j].v.cross(planes[j].a - planes[i].a);
      if(ge(A, 0)){
        lo = max(lo, B/A);
      }else if(le(A, 0)){
        hi = min(hi, B/A);
      }else{
        if(ge(B, 0)) return {};
      }
      if(ge(lo, hi)) return {};
    }
    p = planes[i].a + planes[i].v*lo;
  }
  return {p};
}

int main(){
  /*vector<pair<point, point>> centers = {{point(-2, 5), point(-8, -7)}, {point(14, 4), point(18, 6)}, {point(9, 20), point(9, 28)},
                        {point(21, 20), point(21, 29)}, {point(8, -10), point(14, -10)}, {point(24, -6), point(34, -6)},
                        {point(34, 8), point(36, 9)}, {point(50, 20), point(56, 24.5)}};
  vector<pair<ld, ld>> radii = {{7, 4}, {3, 5}, {4, 4}, {4, 5}, {3, 3}, {4, 6}, {5, 1}, {10, 2.5}};
  int n = centers.size();
  for(int i = 0; i < n; ++i){
    cout << "\n" << centers[i].first << " " << radii[i].first << " " << centers[i].second << " " << radii[i].second << "\n";
    auto extLines = tangents(centers[i].first, radii[i].first, centers[i].second, radii[i].second, false);
```

```
    cout << "Exterior tangents:\n";
    for(auto par : extLines){
      for(auto p : par){
        cout << p << " ";
      }
      cout << "\n";
    }
    auto intLines = tangents(centers[i].first, radii[i].first, centers[i].
        second, radii[i].second, true);
    cout << "Interior tangents:\n";
    for(auto par : intLines){
      for(auto p : par){
        cout << p << " ";
      }
      cout << "\n";
    }
  }*/

  /*int n;
  cin >> n;
  vector<point> P(n);
  for(auto & p : P) cin >> p;
  auto triangulation = delaunay(P);
  for(auto triangle : triangulation){
    cout << get<0>(triangle) << " " << get<1>(triangle) << " " << get<2>(
        triangle) << "\n";
  }*/

  /*int n;
  cin >> n;
  vector<point> P(n);
  for(auto & p : P) cin >> p;
  auto ans = smallestEnclosingCircle(P);
  cout << ans.first << " " << ans.second << "\n";*/

  /*vector<point> P;
  srand(time(0));
  for(int i = 0; i < 1000; ++i){
    P.emplace_back(rand() % 1000000000, rand() % 1000000000);
  }
  point o(rand() % 1000000000, rand() % 1000000000), v(rand() % 1000000000,
       rand() % 1000000000);
  polarSort(P, o, v);
  auto ang = [&](point p){
    ld th = atan2(p.y, p.x);
    if(th < 0) th += acosl(-1)*2;
    ld t = atan2(v.y, v.x);
    if(t < 0) t += acosl(-1)*2;
    if(th < t) th += acosl(-1)*2;
    return th;
  };
  for(int i = 0; i < P.size()-1; ++i){
    assert(leq(ang(P[i] - o), ang(P[i+1] - o)));
  }*/
  return 0;
}
```

# 3 Varios

## 3.1 Template

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

#define forn(i,n)       for(int i=0; i<n; i++)
#define forr(i,a,n)     for(int i=a; i<n; i++)
#define fore(i,a,n)     for(int i=a; i<=n; i++)
#define each(a,b)       for(auto a: b)
#define all(v)          v.begin(),v.end()
#define sz(a)           (int)a.size()
#define debln(a)        cout << a << "\n"
#define deb(a)          cout << a << " "
#define pb              push_back

typedef long long ll;
typedef vector<int> vi;
typedef pair<int,int> ii;

void sol(){

}

int main(){
    ios::sync_with_stdio(false);cin.tie(0);

    int t=1;
```

```
    cin>>t;
    while(t--){
        sol();
    }

    return 0;
}
```

## 3.2 String a vector<int>

```
//Convertir una cadena de numeros separados por " " en vector de enteros
//Leer varias de esas querys
cin.ignore();
while(q--){
  string s;
  getline(cin, s);
  vector<int> qr;
  stringstream ss(s);
  int num;
  while (ss >> num)   qr.push_back(num);
}
```

## 3.3 Generar permutaciones

```
//Generar todas las permutaciones de un arreglo
sort(all(a));
do{
  //hacer lo que quieras con la perm generada
}while(next_permutation(all(a)));
```