.

Descongelen a Victor Moreno

Contents

1 Estructuras de Datos

1.1 Unordered Map

```
#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;
}
```

1.9 Segment Tree Persistente

```
#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

```
//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

```
// 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]
  int kth(int l, int r, int k){
    if(l > r) return 0;
    if(lo == hi) return lo;
    int inLeft = b[r] - b[l-1];
    int lb = b[l-1];
    int rb = b[r];
    if(k <= inLeft) return this->l->kth(lb+1, rb , k);
    return this->r->kth(l-lb, r-rb, k-inLeft);
  }
  //count of nos in [l, r] Less than or equal to k
  int LTE(int l, int r, int k) {
    if(l > r or k < lo) return 0;
    if(hi <= k) return r - l + 1;
    int lb = b[l-1], rb = b[r];
    return this->l->LTE(lb+1, rb, k) + this->r->LTE(l-lb, r-rb, k);
  }
  //count of nos in [l, r] equal to k
  int count(int l, int r, int k) {
    if(l > r or k < lo or k > hi) return 0;
    if(lo == hi) return r - l + 1;
    int lb = b[l-1], rb = b[r], mid = (lo+hi)/2;
    if(k <= mid) return this->l->count(lb+1, rb, k);
    return this->r->count(l-lb, r-rb, k);
  }
};
```

2 Strings

2.1 Aho Corasick

```
int K, I = 1;
struct node {
    int fail, ch[26] = {};
    vector<int> lens;
} T[500005];

void add(string s) {
    int x = 1;
    for (int i = 0; i < s.size(); i++) {
        if (T[x].ch[s[i] - 'a'] == 0)
            T[x].ch[s[i] - 'a'] = ++I;
        x = T[x].ch[s[i] - 'a'];
    }
    T[x].lens.PB(s.size());
}

void build() {
    queue<int> Q;
    int x = 1;
    T[1].fail = 1;
    for (int i = 0; i < 26; i++) {
        if (T[x].ch[i])
            T[T[x].ch[i]].fail = x, Q.push(T[x].ch[i]);
        else
            T[x].ch[i] = 1;
    }
    while (!Q.empty()) {
        x = Q.front(); Q.pop();
        for (int i = 0; i < 26; i++) {
            if (T[x].ch[i])
                T[T[x].ch[i]].fail = T[T[x].fail].ch[i], Q.push(T[x].ch[i]);
            else
                T[x].ch[i] = T[T[x].fail].ch[i];
        }
    }
}
```

2.2 Hashing

```
struct Hash{
  const int mod=1e9+123;
  const int p=257;
  vector<int> prefix;
  static vector<int>pow;
  Hash(string str){
    int n=str.size();
    while(pow.size()<=n){
      pow.push_back(1LL*pow.back()*p%mod);
    }
    vector<int> aux(n+1);
    prefix=aux;
    for(int i=0;i<n;i++){
      prefix[i+1]=(prefix[i]+1LL*str[i]*pow[i])%mod;
    }
  }
  inline int getHashInInerval(int i,int len,int MxPow){
    int hashing=prefix[i+len]-prefix[i];
    if(hashing<0) hashing+=mod;
    hashing=1LL*hashing*pow[MxPow-(len+i-1)]%mod;
    return hashing;
  }
};
vector<int> Hash::pow{1};
```

2.3 KMP

```
vector<int> kmp(string s){
    int n=s.size();
    vector<int>pi(n);
    for(int i=1;i<n;i++){
        int j=pi[i-1];
        while(j>0 && s[i]!=s[j])j=pi[j-1];
        if(s[i]==s[j]) j++;
        pi[i]=j;
    }
    return pi;
}
```

2.4 Manacher

```
vector<int> manacher_odd(string s) {
    int n = s.size();
    s = "$" + s + "^";
    vector<int> p(n + 2);
    int l = 1, r = 1;
    for(int i = 1; i <= n; i++) {
        p[i] = max(0, min(r - i, p[l + (r - i)]));
        while(s[i - p[i]] == s[i + p[i]]) {
            p[i]++;
        }
        if(i + p[i] > r) {
            l = i - p[i], r = i + p[i];
        }
    }
    return vector<int>(begin(p) + 1, end(p) - 1);
}
vector<int> manacher_even(string s){
    string even;
    for(auto c:s){
        even+='#'+c;
    }
    even+='#';
    return manacher_odd(even);
}
```

2.5 Suffix Automata

```
struct node{
  map<char,int>edges;
  int link,length,terminal=0;
  node(int link,int length): link(link),length(length){};
};vector<node>sa;
// init in main with sa.push_back(node(-1,0));
int last=0;
// add one by one chars in order
void addChar(char s, int pos){
    sa.push_back(node(0,pos+1));
    int r=sa.size()-1;
    int p=last;
    while(p >= 0 && sa[p].edges.find(s) == sa[p].edges.end()) {
      sa[p].edges[s] = r;
      p = sa[p].link;
    }
    if(p != -1) {
      int q = sa[p].edges[s];
      if(sa[p].length + 1 == sa[q].length) {
        sa[r].link = q;
      } else {
        sa.push_back(node(sa[q].link,sa[p].length+1));
        sa[sa.size()-1].edges=sa[q].edges;
        int qq = sa.size()-1;
        sa[q].link = qq;
        sa[r].link= qq;
        while(p >= 0 && sa[p].edges[s] == q) {
          sa[p].edges[s] = qq;
          p = sa[p].link;
        }
      }
    }
    last = r;
}
// Not necesary functions
```

```
void findTerminals(){
    int p = last;
    while(p > 0) {
       sa[p].terminal=1;
       p = sa[p].link;
    }
}
```

2.6 Trie

```
struct trie{
    int len,id;
    int children[26];
    trie(int _id){
        len=0,id=_id;
        for(int i=0;i<26;i++)children[i]=-1;
    }
};vector<trie>Trie;Trie.push_back(trie());
void inserString(string str,int root){
    int aux=root;
    for(int i=0;i<str.size();i++){
        int index=str[i]-'a';
        if(Trie[aux].children[index]==-1){
            Trie.push_back(trie(Trie.size()));
            Trie[aux].children[index]=Trie.size()-1;
        }
        aux=Trie[aux].children[index];
    }
    Trie[aux].len=str.size();
}
bool existInTrie(string str,int root){
    int aux=root;
    for(int i=0;i<str.size();i++){
        int index=str[i]-'a';
        if(Trie[aux].children[index]==-1) return false;
        aux=Trie[aux].children[index];
    }
    return Trie[aux].len;
}
```

3 Geometria

3.1 Puntos y lineas

```
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();
}
```

3.2 Circulos

```
ld distancePointCircle(const point & c, ld r, const point & p){
  //point p, circle with center c and radius r
  return max((ld)0, (p - c).length() - r);
}

point projectionPointCircle(const point & c, ld r, const point & p){
  //point p (outside the circle), circle with center c and radius r
  return c + (p - c).unit() * r;
}

pair<point, point> pointsOfTangency(const point & c, ld r, const point & p){
  //point p (outside the circle), circle with center c and radius r
  point v = (p - c).unit() * r;
  ld d2 = (p - c).norm(), d = sqrt(d2);
  point v1 = v * (r / d), v2 = v.perp() * (sqrt(d2 - r*r) / d);
  return {c + v1 - v2, c + v1 + v2};
}

vector<point> intersectLineCircle(const point & a, const point & v, const point & c, ld r){
  //line a+tv, circle with center c and radius r
  ld h2 = r*r - v.cross(c - a) * v.cross(c - a) / v.norm();
  point p = a + v * v.dot(c - a) / v.norm();
  if(eq(h2, 0)) return {p}; //line tangent to circle
  else if(le(h2, 0)) return {}; //no intersection
  else{
    point u = v.unit() * sqrt(h2);
    return {p - u, p + u}; //two points of intersection (chord)
  }
}

vector<point> intersectSegmentCircle(const point & a, const point & b, const point & c, ld r){
  //segment ab, circle with center c and radius r
  vector<point> P = intersectLineCircle(a, b - a, c, r), ans;
  for(const point & p : P){
    if(pointInSegment(a, b, p)) ans.push_back(p);
  }
  return ans;
}

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;
}
```

3.3 Poligonos

```
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;
}
```

```cpp

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;
}
```

4 Matematicas

4.1 Exponenciacion Binaria

```
ll binpow(ll a, ll b, ll mod) {
    a %= mod;
    ll res = 1;
    while (b > 0) {
        if (b & 1)
            res = res * a % mod;
        a = a * a % mod;
        b >>= 1;
    }
    return res;
}

ll binpow(ll a, ll b) {
    if (b == 0)
        return 1;
    ll res = binpow(a, b / 2);
    if (b % 2)
        return res * res * a;
    else
        return res * res;
}
```

4.2 GCD y LCD

```
ll gcd(ll a, ll b){
  ll r;
  while(b != 0) r = a % b, a = b, b = r;
  return a;
}

ll lcm(ll a, ll b){
  return b * (a / gcd(a, b));
}

ll gcd(const vector<ll>& nums){
  ll ans = 0;
  for(ll num : nums) ans = gcd(ans, num);
  return ans;
}

ll lcm(const vector<ll>& nums){
  ll ans = 1;
  for(ll num : nums) ans = lcm(ans, num);
  return ans;
}
```

4.3 Euclides extendido e inverso modular

```
tuple<lli, lli, lli> extendedGcd(lli a, lli b){
  if(b == 0){
    if(a > 0) return {a, 1, 0};
    else return {-a, -1, 0};
  }else{
    auto[d, x, y] = extendedGcd(b, a%b);
    return {d, y, x - y*(a/b)};
  }
}

lli modularInverse(lli a, lli m){
  auto[d, x, y] = extendedGcd(a, m);
  if(d != 1) return -1; // inverse doesn't exist
  if(x < 0) x += m;
  return x;
}
```

4.4 Fibonacci

```
//very fast fibonacci
inline void modula(lli & n, lli mod){
  while(n >= mod) n -= mod;
}

lli fibo(lli n, lli mod){
  array<lli, 2> F = {1, 0};
  lli p = 1;
  for(lli v = n; v >>= 1; p <<= 1);
  array<lli, 4> C;
  do{
    int d = (n & p) != 0;
    C[0] = C[3] = 0;
    C[d] = F[0] * F[0] % mod;
    C[d+1] = (F[0] * F[1] << 1) % mod;
    C[d+2] = F[1] * F[1] % mod;
    F[0] = C[0] + C[2] + C[3];
    F[1] = C[1] + C[2] + (C[3] << 1);
    modula(F[0], mod), modula(F[1], mod);
  }while(p >>= 1);
  return F[1];
}

const long M = 1000000007; // modulo
map<long, long> F;

long f(long n) {
  if (F.count(n)) return F[n];
  long k=n/2;
  if (n%2==0) { // n=2*k
    return F[n] = (f(k)*f(k) + f(k-1)*f(k-1)) % M;
  } else { // n=2*k+1
    return F[n] = (f(k)*f(k+1) + f(k-1)*f(k)) % M;
  }
}

main(){
  long n;
  F[0]=F[1]=1;
  while (cin >> n)
  cout << (n==0 ? 0 : f(n-1)) << endl;
}
```

4.5 Criba de Primos

```
vector<int> linearPrimeSieve(int n){
  vector<int> primes;
  vector<bool> isPrime(n+1, true);
  for(int i = 2; i <= n; ++i){
    if(isPrime[i])
      primes.push_back(i);
    for(int p : primes){
      int d = i * p;
      if(d > n) break;
      isPrime[d] = false;
      if(i % p == 0) break;
    }
  }
  return primes;
}
```

4.6 Triangulo de Pascal

```
vector<vector<lli>> ncrSieve(int n){
  vector<vector<lli>> Ncr(n+1);
  Ncr[0] = {1};
  for(int i = 1; i <= n; ++i){
    Ncr[i].resize(i + 1);
    Ncr[i][0] = Ncr[i][i] = 1;
    for(int j = 1; j <= i / 2; j++)
      Ncr[i][i - j] = Ncr[i][j] = Ncr[i - 1][j - 1] + Ncr[i - 1][j];
  }
  return Ncr;
}
```

4.7 Cambio de bases

```
string decimalToBaseB(lli n, lli b){
  string ans = "";
  lli d;
  do{
    d = n % b;
    if(0 <= d && d <= 9) ans = (char)(48 + d) + ans;
    else if(10 <= d && d <= 35) ans = (char)(55 + d) + ans;
    n /= b;
  }while(n != 0);
  return ans;
}

lli baseBtoDecimal(const string & n, lli b){
  lli ans = 0;
  for(const char & d : n){
    if(48 <= d && d <= 57) ans = ans * b + (d - 48);
    else if(65 <= d && d <= 90) ans = ans * b + (d - 55);
    else if(97 <= d && d <= 122) ans = ans * b + (d - 87);
  }
  return ans;
}
```

4.8 Factorizacion

```
vector<pair<lli, int>> factorize(lli n){
  vector<pair<lli, int>> f;
  for(lli p : primes){
    if(p * p > n) break;
    int pot = 0;
    while(n % p == 0){
      pot++;
      n /= p;
    }
    if(pot) f.emplace_back(p, pot);
  }
  if(n > 1) f.emplace_back(n, 1);
  return f;
}
```

5 Varios

5.1 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;
}
```

5.2 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);
}
```

5.3 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)));
```