El BichiGol

UTN FRSF - El Rejunte

2017

Índice

1. C/C++

1.1. I/O

1.1.1. scanf Format Strings

`%[*][width][length]specifier`

spec	Tipo	Descripción
`i`	`int`	Dígitos dec. `[0-9]`, oct. `(0)[0-7]`, hexa `(0x|0X)[0-9a-fA-F]`. Con signo.
`d, u`	`int, unsigned`	Dígitos dec. `[+-0-9]`.
`o`	`unsigned`	Dígitos oct. `[+-0-7]`.
`x`	`unsigned`	Dígitos hex. `[+-0-9a-fA-F]`. Prefijo `0x,0X` opcional.
`f, e, g`	`float`	Dígitos dec. c/punto flotante `[+-.0-9]`. Prefijo `0x,0X` y sufijo `e,E` opcionales.
`c, [width]c`	`char, char*`	Siguiente carácter. Lee `width` chars y los almacena contiguamente. No agrega `\0`.
`s`	`char*`	Secuencia de chars hasta primer espacio. Agrega `\0`.
`p`	`void*`	Secuencia de chars que representa un puntero.
`[chars]`	`Scanset, char*`	Caracteres especificados entre corchetes. `]` debe ser primero en la lista, `-` primero o último. Agrega `\0`
`[^chars]`	`!Scanset, char*`	Caracteres no especificados entre corchetes.
`n`	`int`	No consume entrada. Almacena el número de chars leídos hasta el momento.
`%`		`%%` consume un `%`

sub-specifier	Descripción
`*`	Indica que se leerá el dato pero se ignorará. No necesita argumento.
`width`	Cantidad máxima de caracteres a leer.
`lenght`	Uno de `hh, h, l, ll, j, z, t, L`. Ver tabla siguiente.

length	d i	u o x
(none)	`int*`	`unsigned int*`
hh	`signed char*`	`unsigned char*`
h	`short int*`	`unsigned short int*`
l	`long int*`	`unsigned long int*`
ll	`long long int*`	`unsigned long long int*`
j	`intmax_t*`	`uintmax_t*`
z	`size_t*`	`size_t*`
t	`ptrdiff_t*`	`ptrdiff_t*`
L		

length	f e g a	c s [] [^]	p	n
(none)	`float*`	`char*`	`void**`	`int*`
hh				`signed char*`
h				`short int*`
l	`double*`	`wchar_t*`		`long int*`
ll				`long long int*`
j				`intmax_t*`
z				`size_t*`
t				`ptrdiff_t*`
L	`long double*`			

1.1.2. printf Format Strings

`%[flags][width][.precision][length]specifier`

specifier	Descripción	Ejemplo
`d or i`	Entero decimal con signo	`392`
`u`	Entero decimal sin signo	`7235`
`o`	Entero octal sin signo	`610`
`x`	Entero hexadecimal sin signo	`7fa`
`X`	Entero hexadecimal sin signo (mayúsculas)	`7FA`
`f`	Decimal punto flotante (minúsculas)	`392.65`
`F`	Decimal punto flotante (mayúsculas)	`392.65`
`e`	Notación científica (mantisa/exponente), (minúsculas)	`3.9265e+2`
`E`	Notación científica (mantisa/exponente), (mayúsculas)	`3.9265E+2`
`g`	Utilizar la representacíon más corta: `%e` ó `%f`	`392.65`
`G`	Utilizar la representacíon más corta: `%E` ó `%F`	`392.65`
`a`	Hexadecimal punto flotante (minúsculas)	`-0xc.90fep-2`
`A`	Hexadecimal punto flotante (mayúsculas)	`-0XC.90FEP-2`
`c`	Caracter	`a`
`s`	String de caracteres	`sample`

Continuación		
specifier	**Descripción**	**Ejemplo**
`p`	Dirección de puntero	`b8000000`
`n`	No imprime nada. El argumento debe ser `int*`, almacena el número de caracteres imprimidos hasta el momento.	
`%`	Un % seguido de otro % imprime un solo %	`%`

flag	**Descripción**
`-`	Justificación a la izquierda dentro del campo `width` (ver `width` sub-specifier).
`+`	Forza a preceder el resultado de texttt+ o texttt-.
`(espacio)`	Si no se va a escribir un signo, se inserta un espacio antes del valor.
`#`	Usado con `o, x, X` specifiers el valor es precedido por 0, 0x, 0X respectivamente para valores distintos de 0.
`0`	Rellena el número con texttt0 a la izquierda en lugar de espacios cuando se especifica `width`.

width	**Descripción**
`(número)`	Número mínimo de caracteres a imprimir. Si el valor es menor que `número`, el resultado es rellando con espacios. Si el valor es mayor, no es truncado.
`*`	No se especifica `width`, pero se agrega un argumento entero precediendo al argumento a ser formateado. Ej. `printf("---%*d----\n", 3, 2);` ⇒ "---- 5----".

precision	**Descripción**
`.(número)`	Para `d, i, o, u, x, X`: número mínimo de dígitos a imprimir. Si el valor es más chico que `número` se rellena con `0`. Para `a, A, e, E, f, F`: número de dígitos a imprimir después de la coma (default 6). Para `g, G`: Número máximo de cifras significativas a imprimir. Para `s`: Número máximo de caracteres a imprimir. Trunca.
`.*`	No se especifica `precision` pero se agrega un argumento entero precediendo al argumento a ser formateado.

length	**d i**	**u o x X**
(none)	`int`	`unsigned int`
hh	`signed char`	`unsigned char`
h	`short int`	`unsigned short int`
l	`long int`	`unsigned long int`
ll	`long long int`	`unsigned long long int`

Continuación		
length	**d i**	**u o x X**
j	`intmax_t`	`uintmax_t`
z	`size_t`	`size_t`
t	`ptrdiff_t`	`ptrdiff_t`
L		

length	**f F e E g G a A**	**c**	**s**	**p**	**n**
(none)	`double`	`int`	`char*`	`void*`	`int*`
hh					`signed char*`
h					`short int*`
l		`wint_t`	`wchar_t*`		`long int*`
ll					`long long int*`
j					`intmax_t*`
z					`size_t*`
t					`ptrdiff_t*`
L	`long double`				

2. Template del Rejunte

```
#include <bits/stdc++.h>
#define sqr(a) ((a)*(a))
#define rsz resize
#define forr(i,a,b) for(int i=(a);i<(b);i++)
#define forn(i,n) forr(i,0,n)
#define dforn(i,n) for(int i=n-1;i>=0;i--)
#define forall(it,v) for(auto it=v.begin();it!=v.end();it++)
#define foreach(i, v) for(auto i:v)
#define sz(c) ((int)c.size())
#define zero(v) memset(v, 0, sizeof(v))
#define pb push_back
#define mp make_pair
#define lb lower_bound
#define ub upper_bound
#define fst first
#define snd second
#define PI 3.1415926535897932384626

using namespace std;

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

int main()
{
  // agregar g++ -DREJUNTE en compilacion
  #ifdef REJUNTE
    freopen("input", "r", stdin);
    // freopen("output","w", stdout);
  #endif
  ios::sync_with_stdio(false);
  cin.tie(NULL);
  cout.tie(NULL);
  return 0;
}
```

3. Estructuras de datos

3.1. Set Mejorado

Esto solo compila desde C++11.

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
//<key,mapped type,comparator,...>
typedef tree<int,null_type,less<int>,rb_tree_tag,
  tree_order_statistics_node_update> ordered_set;
//find_by_order(i) devuelve iterador al i-esimo elemento
//order_of_key(k): devuelve la pos del lower bound de k
//Ej: 12, 100, 505, 1000, 10000.
//order_of_key(10) == 0, order_of_key(100) == 1,
//order_of_key(707) == 3, order_of_key(9999999) == 5
```

3.2. Union Find

```
struct UnionFind{
  vector<int> f, setSize; //the array f contains the parent of each node
  int cantSets;
  void init(int n)
  {
    f.clear(); setSize.clear();
    cantSets = n;
    f.rsz(n, -1);
    setSize.rsz(n, 1);
  }
  int comp(int x) {return (f[x]==-1? x : f[x]=comp(f[x]));}//O(1)
  bool join(int i,int j) //devuelve true si ya estaban juntos
  {
    int a = comp(i), b = comp(j);
    if(a != b)
    {
      cantSets--;
      if(setSize[a] > setSize[b]) swap(a,b);
      setSize[b] += setSize[a];
      f[a] = b; //el grupo mas grande (b) ahora representa al mas chico (a)
    }
    return a == b;
  }
};
```

3.3. Hash Table

```
//Compilar: g++ --std=c++11
struct Hash{
  size_t operator()(const ii &a)const
  {
    size_t s=hash<int>()(a.fst);
    return hash<int>()(a.snd)+0x9e3779b9+(s<<6)+(s>>2);
  }
  size_t operator()(const vector<int> &v)const
  {
    size_t s=0;
    for(auto &e : v) s^=hash<int>()(e)+0x9e3779b9+(s<<6)+(s>>2);
    return s;
  }
};
unordered_set<ii, Hash> s;
unordered_map<ii, int, Hash> m;//map<key, value, hasher>
```

3.4. Gain cost set

```
//stores pairs (benefit,cost) (erases non-optimal pairs)
//Note that these pairs will be increasing by g and increasing by c
//If we insert a pair that is included in other, the big one will be deleted
//For lis 2d, create a GCS por each posible length, use as (-g, c) and
//binary search looking for the longest length where (-g, c) could be added
struct GCS {
  set<ii> s;
  void add(int g, int c){
    ii x={g,c};
    auto p=s.lower_bound(x);
    if(p!=s.end()&&p->snd<=x.snd)return;
    if(p!=s.begin()) {//erase pairs with less or equal benefit and more cost
      --p;
      while(p->snd>=x.snd){
        if(p==s.begin()){s.erase(p);break;}
        s.erase(p--);
      }
    }
    s.insert(x);
  }
  int get(int gain){ // min cost for the benefit greater or equal to gain
    auto p=s.lower_bound((ii){gain,-INF});
    int r=p==s.end()?INF:p->snd;
    return r;
  }
};
```

3.5. Disjoint intervals

```
// stores disjoint intervals as [first, second)
// the final result is the union of the inserted intervals
// [1, 5), [2, 4), [10, 13), [11, 15) -> [1, 5), [10, 15)
struct disjoint_intervals {
  set<ii> s;
  void insert(ii v){
    if(v.fst>=v.snd) return;
    auto at=s.lower_bound(v);auto it=at;
    if(at!=s.begin()&&(--at)->snd>=v.fst)v.fst=at->fst,--it;
    for(;it!=s.end()&&it->fst<=v.snd;s.erase(it++))
      v.snd=max(v.snd,it->snd);
    s.insert(v);
  }
};
```

3.6. Segment Tree

3.6.1. ST static

Dado un arreglo y una operacion asociativa *idempotente*, get(i, j) opera sobre el rango [i, j). Restriccion: LVL $\geq$ ceil(logn); Usar [ ] para llenar arreglo y luego build().

```
//Solo para funciones idempotentes (como min y max, pero no sum)
//Usar la version dynamic si la funcion no es idempotente
struct RMQ{
  #define LVL 10 // LVL such that 2^LVL>n
  tipo vec[LVL][1<<(LVL+1)];
  tipo &operator[](int p) {return vec[0][p];}
  tipo get(int i, int j) {//intervalo [i,j) - O(1)
    int p = 31 - __builtin_clz(j-i);
    return min(vec[p][i], vec[p][j-(1<<p)]);
  }
  void build(int n) {//O(nlogn)
    int mp = 31 - __builtin_clz(n);
    forn(p, mp) forn(x, n-(1<<p))
      vec[p+1][x] = min(vec[p][x], vec[p][x+(1<<p)]);
  }
}; //Use: define LVL y tipo; insert data with []; call build; answer queries
```

3.6.2. ST dynamic

```
//Dado un arreglo y una operacion asociativa con neutro
#define MAXN 100010
#define operacion(x, y) max(x, y)
```

```
const tipo neutro=0;
struct RMQ {
  int sz;
  tipo t[4*MAXN];
  tipo &operator[](int p) {return t[sz+p];}
  void init(int n) {//O(nlgn)
    sz = 1 << (32 - __builtin_clz(n));
    forn(i, 2*sz) t[i] = neutro;
  }
  void updall() {dforn(i, sz) t[i]=operacion(t[2*i], t[2*i+1]);} //O(n)
  tipo get(int i, int j) {return get(i,j,1,0,sz);}
  tipo get(int i, int j, int n, int a, int b) {//O(lgn), [i, j)
    if(j<=a || i>=b) return neutro;
    if(i<=a && b<=j) return t[n];//n = node of range [a,b)
    int c = (a+b)/2;
    return operacion(get(i, j, 2*n, a, c), get(i, j, 2*n+1, c, b));
  }
  void set(int p, tipo val) {//O(lgn)
    p += sz;
    while(p>0 && t[p]!=val) {
      t[p] = val;
      p /= 2;
      val = operacion(t[p*2], t[p*2+1]);
    }
  }
}; //Use: definir operacion tipo neutro y MAXN,
//cin >> n; rmq.init(n); forn(i, n) cin >> rmq[i]; rmq.updall();
```

3.6.3. ST lazy

```
//Dado un arreglo y una operacion asociativa con neutro
#define operacion(x,y) ((x)+(y))
const Elem neutro=0; const Alt neutro2=0;
#define MAXN 100010//Cambiar segun el N del problema
struct RMQ{
  int sz;
  Elem t[4*MAXN];
  Alt dirty[4*MAXN];//las alteraciones pueden ser de tipo distinto a Elem
  Elem &operator[](int p) {return t[sz+p];}
  void init(int n){//O(nlgn)
    sz = 1 << (32-__builtin_clz(n));
    forn(i, 2*sz) t[i] = neutro;
    forn(i, 2*sz) dirty[i] = neutro2;
  }
  void updall() {dforn(i, sz) t[i]=operacion(t[2*i], t[2*i+1]);} //O(n)
  void push(int n, int a, int b) {//propaga el dirty a sus hijos
    if(dirty[n] != neutro2) {// n = node of range [a,b)
      t[n] += dirty[n]*(b-a);//altera el nodo, modificar segun el problema
      if(n<sz){
        dirty[2*n] += dirty[n];
        dirty[2*n+1] += dirty[n];
      }
      dirty[n] = neutro2;
    }
  }
  Elem get(int i, int j, int n, int a, int b) {//O(lgn)
    if(j<=a || i>=b) return neutro;
    push(n, a, b);//corrige el valor antes de usarlo
    if(i<=a && b<=j) return t[n];//n = node of range [a,b)
    int c = (a+b)/2;
    return operacion(get(i, j, 2*n, a, c), get(i, j, 2*n+1, c, b));
  }
  Elem get(int i, int j) {return get(i,j,1,0,sz);}
  //altera los valores en [i, j) con una alteracion de val
  void alterar(Alt val, int i, int j, int n, int a, int b) {//O(lgn)
    push(n, a, b);
    if(j<=a || i>=b) return;
    if(i<=a && b<=j){
      dirty[n] += val; // modificar segun el problema
      push(n, a, b);
      return;
    }
    int c = (a+b)/2;
    alterar(val, i, j, 2*n, a, c), alterar(val, i, j, 2*n+1, c, b);
    t[n] = operacion(t[2*n], t[2*n+1]);//por esto es el push de arriba
  }
  void alterar(Alt val, int i, int j){alterar(val,i,j,1,0,sz);}
};//Use: definir operacion, neutros, Alt, Elem, uso de dirty, MAXN
//cin >> n; rmq.init(n); forn(i,n) cin >> rmq[i]; rmq.updall()
```

3.6.4. ST persistente

```
typedef int tipo;
tipo oper(const tipo &a, const tipo &b){
    return a+b;
}
struct node{
  tipo v; node *l, *r;
  node(tipo v) : v(v), l(NULL), r(NULL) {}
    node(node *l, node *r) : l(l), r(r) {
        if(!l) v = r->v;
        else if(!r) v = l->v;
        else v = oper(l->v, r->v);
    }
};
node *build(tipo *a, int tl, int tr) {//build a partir de un arreglo
  if(tl+1 == tr) return new node(a[tl]);
  int tm = (tl+tr)>>1;
  return new node(build(a, tl, tm), build(a, tm, tr));
}
node *update(int pos, int new_val, node *t, int tl, int tr) {
  if(tl+1 == tr) return new node(new_val);
```

```
  int tm = (tl+tr)>>1;
  if(pos < tm) return new node(update(pos, new_val, t->l, tl, tm), t->r);
  else return new node(t->l, update(pos, new_val, t->r, tm, tr));
}
tipo get(int l, int r, node *t, int tl, int tr) {
    if(l==tl && tr==r) return t->v;
  int tm = (tl+tr)>>1;
    if(r <= tm) return get(l, r, t->l, tl, tm);
    else if(l >= tm) return get(l, r, t->r, tm, tr);
  return oper(get(l, tm, t->l, tl, tm), get(tm, r, t->r, tm, tr));
}
//node t represents range [tl, tr). For roots always use tl = 0, tr = size
```

3.6.5. ST 2d

```
#define operacion(x, y) max(x, y)
int n, m;
int a[MAXN][MAXN], st[2*MAXN][2*MAXN];
void build() { //O(n*m)
  forn(i, n) forn(j, m) st[i+n][j+m] = a[i][j];
  forn(i, n) dforn(j, m) //build st of row i+n (each row independently)
    st[i+n][j] = operacion(st[i+n][j<<1], st[i+n][j<<1|1]);
  dforn(i, n) forn(j, 2*m) //build st of ranges of rows
    st[i][j] = operacion(st[i<<1][j], st[i<<1|1][j]);
}
void upd(int x, int y, int v) { //O(logn * logm)
  st[x+n][y+m] = v;
  for(int j=y+m; j>1; j>>=1)//update the ranges that contains y+m in row x+n
    st[x+n][j>>1] = operacion(st[x+n][j], st[x+n][j^1]);
  for(int i=x+n; i>1; i>>=1)//in each range that contains row x+n
    for(int j=y+m; j; j>>=1) //update the ranges that contains y+m
      st[i>>1][j] = operacion(st[i][j], st[i^1][j]);
}
int query(int x0, int x1, int y0, int y1) { //O(logn * logm)
  int r = NEUT;
  //start at the bottom and move up each time
  for(int i0=x0+n, i1=x1+n; i0<i1; i0>>=1, i1>>=1) {
    int t[4], q=0;
    //if the whole segment of row node i0 is included, then move right
    if(i0&1) t[q++] = i0++;
    //if the whole segment of row node i1-1 is included, then move left
    if(i1&1) t[q++] = --i1;
    forn(k, q) for(int j0=y0+m, j1=y1+m; j0<j1; j0>>=1, j1>>=1) {
      if(j0&1) r = operacion(r, st[t[k]][j0++]);
      if(j1&1) r = operacion(r, st[t[k]][--j1]);
    }
  }
  return r;
}
```

3.7. Treap

```
typedef struct item *pitem;
struct item {
    //pr = randomized priority, key = BST value, cnt = size of subtree
    int pr, key, cnt;
    pitem l, r;
    item(int key) : key(key), pr(rand()), cnt(1), l(NULL), r(NULL) {}
};
int cnt(pitem node) {return node ? node->cnt : 0;}
void upd_cnt(pitem node) {if(node) node->cnt = cnt(node->l)+cnt(node->r)+1;}
//splits t in l and r - l: <= key, r: > key
void split(pitem node, int key, pitem& L, pitem& R){ // O(log)
    if(!node) L = R = 0;
    // if cur > key, go left to split and cur is part of R
    else if(key < node->key) split(node->l, key, L, node->l), R = node;
    // if cur <= key, go right to split and cur is part of L
    else split(node->r, key, node->r, R), L = node;
    upd_cnt(node);
}
//1) go down the BST following the key of the new node (x), until
// you reach NULL or a node with lower pr than the new one.
//2.1) if you reach NULL, put the new node there
//2.2) if you reach a node with lower pr, split the subtree rooted at that
//node, put the new one there and put the split result as children of it
void insert(pitem& node, pitem x) { // O(log)
    if(!node) node = x;
    else if(x->pr > node->pr) split(node, x->key, x->l, x->r), node = x;
    else insert(x->key <= node->key ? node->l : node->r, x);
    upd_cnt(node);
}
//Assumes that the key of every element in L <= to the keys in R
void merge(pitem& result, pitem L, pitem R) { // O(log)
    //If one of the nodes is NULL, the merge result is the other node
    if(!L || !R) result = L ? L : R;
    //if L has higher priority than R, put L and update it's right child
    //with the merge result of L->r and R
    else if(L->pr > R->pr) merge(L->r, L->r, R), result = L;
    //if R has higher priority than L, put R and update it's left child
    //with the merge result of L and R->l
    else merge(R->l, L, R->l), result = R;
    upd_cnt(result);
}
//go down the BST following the key to erase. When the key is found,
//replace that node with the result of merging it children
void erase(pitem& node, int key) {// O(log), (erases only 1 repetition)
    if(node->key == key) merge(node, node->l, node->r);
    else erase(key < node->key ? node->l : node->r, key);
    upd_cnt(node);
}
//union of two treaps
void unite(pitem &t, pitem L, pitem R) { // O(M*log(N/M))
    if(!L || !R) {t = L ? L : R; return;}
```

```
    if(L->pr < R->pr) swap(L, R);
    pitem p1, p2; split(R, L->key, p1, p2);
    unite(L->l, L->l, p1); unite(L->r, L->r, p2);
    t = L; upd_cnt(t);
}
pitem kth(pitem t, int k) { // element at "position" k
    if(!t) return 0;
    if(k == cnt(t->l)) return t;
    return k < cnt(t->l) ? kth(t->l, k) : kth(t->r, k - cnt(t->l) - 1);
}
pair<int,int> lb(pitem t, int key) { // position and value of lower_bound
    if(!t) return {0,1<<30}; // (special value)
    if(key > t->key){
        auto w = lb(t->r,key); w.fst += cnt(t->l)+1; return w;
    }
    auto w = lb(t->l,key);
    if(w.fst == cnt(t->l)) w.snd = t->key;
    return w;
}
```

3.8. Implicit treap

```
// An array represented as a treap, where the "key" is the index.
// However, the key is not stored explicitly, but can be calculated as
// the sum of the sizes of the left child of the ancestors where the node
// is in the right subtree of it.
// (commented parts are specific to range sum queries problem)
// rng = random number generator, works better than rand in some cases
mt19937 rng;
typedef struct item *pitem;
struct item {
  int pr, cnt, val;
  bool rev;
//  int sum; // (paramters for range query)
//  int add; // (parameters for lazy prop)
  pitem l, r;
  item(int val) : pr(rng()), cnt(1), val(val), l(NULL), r(NULL),
          rev(false)/*, sum(val), add(0)*/{}
};
void push(pitem node) {
  if(node){
    if(node->rev) {
      swap(node->l, node->r);
      if(node->l) node->l->rev ^= true;
      if(node->r) node->r->rev ^= true;
      node->rev = false;
    }
    /*node->val += node->add; node->sum += node->cnt * node->add;
    if(node->l) node->l->add += node->add;
    if(node->r) node->r->add += node->add;
    node->add = 0;*/
  }
}
int cnt(pitem t) {return t ? t->cnt : 0;}
// int sum(pitem t) {return t ? push(t), t->sum : 0;}
void upd_cnt(pitem t) {
  if(t) {
    t->cnt = cnt(t->l) + cnt(t->r) + 1;
    // t->sum=t->val+sum(t->l)+sum(t->r);
  }
}
void split(pitem node, pitem& L, pitem& R, int sz) {// sz: wanted size for L
  if(!node) {L = R = 0; return;}              // O(log)
  push(node);
  //If node's left child has at least sz nodes, go left
  if(sz <= cnt(node->l)) split(node->l, L, node->l, sz), R = node;
  //Else, go right changing wanted sz
  else split(node->r, node->r, R, sz-1-cnt(node->l)), L = node;
  upd_cnt(node);
}
void merge(pitem& result, pitem L, pitem R) {// O(log)
  push(L); push(R);
  if(!L || !R) result = L ? L : R;
  else if(L->pr > R->pr) merge(L->r, L->r, R), result = L;
  else merge(R->l, L, R->l), result = R;
  upd_cnt(result);
}
void insert(pitem& node, pitem x, int pos) {// 0-index O(log)
  pitem l,r;
  split(node, l, r, pos);
  merge(l, l, x);
  merge(node, l, r);
}
void erase(pitem& node, int pos) {// 0-index O(log)
  if(!node) return;
  push(node);
    if(pos == cnt(node->l)) merge(node, node->l, node->r);
    else if(pos < cnt(node->l)) erase(node->l, pos);
    else erase(node->r, pos-1-cnt(node->l));
    upd_cnt(node);
}
void reverse(pitem &node, int L, int R) {//[L, R) O(log)
    pitem t1, t2, t3;
    split(node, t1, t2, L);
    split(t2, t2, t3, R-L);
    t2->rev ^= true;
    merge(node, t1, t2);
    merge(node, node, t3);
}
/*void add(pitem &node, int L, int R, int x) {//[L, R) O(log)
  pitem t1, t2, t3;
  split(node, t1, t2, L);
  split(t2, t2, t3, R-L);
  t2->add += x;
```

```
  merge(node, t1, t2);
  merge(node, node, t3);
}*/
/*int get(pitem &node, int L, int R) {//[L, R) O(log)
  pitem t1, t2, t3;
  split(node, t1, t2, L);
  split(t2, t2, t3, R-L);
  push(t2);
  int ret = t2->sum;
  merge(node, t1, t2);
  merge(node, node, t3);
  return ret;
}*/
void output(pitem t){ // useful for debugging
    if(!t)return;
    push(t);
    output(t->l);cout << ' ' << t->val;output(t->r);
}
```

3.9. STL 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.append(other_rope)
// s.insert(i,x)
// s.insert(i,other_rope) // 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)
```

3.10. BIGInt

```
#define BASEXP 6
#define BASE 1000000
#define LMAX 1000
struct bint{
    int l;
    ll n[LMAX];
    bint(ll x=0){
        l=1;
        forn(i, LMAX){
            if (x) l=i+1;
            n[i]=x %BASE;
            x/=BASE;

        }
    }
    bint(string x){
    l=(x.size()-1)/BASEXP+1;
        fill(n, n+LMAX, 0);
        ll r=1;
        forn(i, sz(x)){
            n[i / BASEXP] += r * (x[x.size()-1-i]-'0');
            r*=10; if(r==BASE)r=1;
        }
    }
    void out(){
    cout << n[l-1];
    dforn(i, l-1) printf("%6.6llu", n[i]);//6=BASEXP!
  }
  void invar(){
    fill(n+l, n+LMAX, 0);
    while(l>1 && !n[l-1]) l--;
  }
};
bint operator+(const bint&a, const bint&b){
  bint c;
    c.l = max(a.l, b.l);
    ll q = 0;
    forn(i, c.l) q += a.n[i]+b.n[i], c.n[i]=q %BASE, q/=BASE;
    if(q) c.n[c.l++] = q;
    c.invar();
    return c;
}
pair<bint, bool> lresta(const bint& a, const bint& b)   // c = a - b
{
  bint c;
    c.l = max(a.l, b.l);
    ll q = 0;
    forn(i, c.l) q += a.n[i]-b.n[i], c.n[i]=(q+BASE) %BASE, q=(q+BASE)/BASE-1;
    c.invar();
    return make_pair(c, !q);
}
bint& operator-= (bint& a, const bint& b){return a=lresta(a, b).first;}
bint operator- (const bint&a, const bint&b){return lresta(a, b).first;}
bool operator< (const bint&a, const bint&b){return !lresta(a, b).second;}
bool operator<= (const bint&a, const bint&b){return lresta(b, a).second;}
bool operator==(const bint&a, const bint&b){return a <= b && b <= a;}
bint operator*(const bint&a, ll b){
    bint c;
    ll q = 0;
    forn(i, a.l) q += a.n[i]*b, c.n[i] = q %BASE, q/=BASE;
    c.l = a.l;
    while(q) c.n[c.l++] = q %BASE, q/=BASE;
```

```
    c.invar();
    return c;
}
bint operator*(const bint&a, const bint&b){
    bint c;
    c.l = a.l+b.l;
    fill(c.n, c.n+b.l, 0);
    forn(i, a.l){
        ll q = 0;
        forn(j, b.l) q += a.n[i]*b.n[j]+c.n[i+j], c.n[i+j] = q %BASE, q/=
            BASE;
        c.n[i+b.l] = q;
    }
    c.invar();
    return c;
}
pair<bint, ll> ldiv(const bint& a, ll b){// c = a / b ; rm = a % b
  bint c;
  ll rm = 0;
  dforn(i, a.l){
            rm = rm * BASE + a.n[i];
            c.n[i] = rm / b;
            rm %= b;
    }
    c.l = a.l;
    c.invar();
    return make_pair(c, rm);
}
bint operator/(const bint&a, ll b){return ldiv(a, b).first;}
ll operator%(const bint&a, ll b){return ldiv(a, b).second;}
pair<bint, bint> ldiv(const bint& a, const bint& b){
  bint c;
    bint rm = 0;
    dforn(i, a.l){
        if (rm.l==1 && !rm.n[0])
            rm.n[0] = a.n[i];
        else{
            dforn(j, rm.l) rm.n[j+1] = rm.n[j];
            rm.n[0] = a.n[i];
            rm.l++;
        }
        ll q = rm.n[b.l] * BASE + rm.n[b.l-1];
        ll u = q / (b.n[b.l-1] + 1);
        ll v = q /  b.n[b.l-1] + 1;
        while (u < v-1){
            ll m = (u+v)/2;
            if (b*m <= rm) u = m;
            else v = m;
        }
        c.n[i]=u;
        rm-=b*u;
    }
  c.l=a.l;
    c.invar();
    return make_pair(c, rm);
}
bint operator/(const bint&a, const bint&b){return ldiv(a, b).first;}
bint operator%(const bint&a, const bint&b){return ldiv(a, b).second;}
```

4. Algoritmos

4.1. Longest Increasing Subsecuence

```
//Para non-increasing, cambiar comparaciones y revisar busq binaria
//Given an array, paint it in the least number of colors so that each color
    turns to a non-increasing subsequence.
//Solution:Min number of colors=Length of the longest increasing subsequence
int N, a[MAXN];//secuencia y su longitud
ii d[MAXN+1];//d[i]=ultimo valor de la subsecuencia de tamanio i
int p[MAXN];//padres
vector<int> R;//respuesta
void rec(int i){
  if(i==-1) return;
  R.push_back(a[i]);
  rec(p[i]);
}
int lis(){//O(nlogn)
  d[0] = ii(-INF, -1); forn(i, N) d[i+1]=ii(INF, -1);
  forn(i, N){
    int j = upper_bound(d, d+N+1, ii(a[i], INF))-d;
    if (d[j-1].first < a[i]&&a[i] < d[j].first){
      p[i]=d[j-1].second;
      d[j] = ii(a[i], i);
    }
  }
  R.clear();
  dforn(i, N+1) if(d[i].first!=INF){
    rec(d[i].second);//reconstruir
    reverse(R.begin(), R.end());
    return i;//longitud
  }
  return 0;
}
```

4.2. Mo's

$$O(q * \sqrt{n})$$

```
int n,sq;
struct Qu{//queries [l, r]
    //intervalos cerrado abiertos !!! importante!!
```

```
    int l, r, id;
}qs[MAXN];
int ans[MAXN], curans;//ans[i]=ans to ith query
bool bymos(const Qu &a, const Qu &b){
    if(a.l/sq!=b.l/sq) return a.l<b.l;
    return (a.l/sq)&1? a.r<b.r : a.r>b.r;
}
void mos(){
    forn(i, t) qs[i].id=i;
    sort(qs, qs+t, bymos);
    int cl=0, cr=0;
    sq=sqrt(n);
    curans=0;
    forn(i, t){ //intervalos cerrado abiertos !!! importante!!
        Qu &q=qs[i];
        while(cl>q.l) add(--cl);
        while(cr<q.r) add(cr++);
        while(cl<q.l) remove(cl++);
        while(cr>q.r) remove(--cr);
        ans[q.id]=curans;
    }
}
```

5. Strings

5.1. Hashing

```
struct Hash {
  //P must be a prime number, could be randomly generated,
  //sometimes is good to make it close to alphabet size
  int P=1777771, MOD[2], PI[2];
  vector<int> h[2], pi[2];
  vector<ll> vp[2]; //Only used if getChanged is used (delete it if not)
  Hash(string& s) {
    //MOD[i] must be a prime of this order, could be randomly generated
    MOD[0]=999727999; MOD[1]=1070777777;
    //PI[i] = P^-1 % MOD[i]
    PI[0]=325255434; PI[1]=10018302;
    forn(k, 2)
      h[k].rsz(s.size()+1), pi[k].rsz(s.size()+1), vp[k].rsz(s.size()+1);
    forn(k, 2) {
      h[k][0] = 0; pi[k][0] = vp[k][0] = 1;
      ll p=1;
      forr(i, 1, s.size()+1) {
        h[k][i] = (h[k][i-1] + p*s[i-1]) % MOD[k];
        pi[k][i] = (1LL * pi[k][i-1] * PI[k]) % MOD[k];
        vp[k][i] = p = (p*P) % MOD[k];
      }
    }
  }
  ll get(int s, int e) { // get hash value of the substring [s, e)
    ll H[2];
    forn(i, 2) {
      H[i] = (h[i][e] - h[i][s] + MOD[i]) % MOD[i];
      H[i] = (1LL * H[i] * pi[i][s]) % MOD[i];
    }
    return (H[0]<<32)|H[1];
  }
  //get hash value of [s, e) if origVal in pos is changed to val
  //Assumes s <= pos < e. If multiple changes are needed,
    //do what is done in the for loop for every change
    ll getChanged(int s, int e, int pos, int val, int origVal) {
        ll hv = get(s,e), h[2];
        h[1] = hv & ((1LL<<32)-1);
        h[0] = hv >> 32;
        forn(i, 2)
          h[i] = (h[i] + vp[i][pos] * (val - origVal + MOD[i])) % MOD[i];
        return (h[0]<<32)|h[1];
    }
};
```

5.2. Hashing 128 bits

```
#define bint __int128 // needs gcc compiler?
struct Hash {
  bint MOD=212345678987654321LL, P=1777771, PI=106955741089659571LL;
  vector<bint> h, pi;
  Hash(string& s) {
    assert((P*PI)%MOD == 1);
    h.resize(s.size()+1); pi.resize(s.size()+1);
    h[0]=0; pi[0]=1;
    bint p=1;
    forr(i, 1, s.size()+1) {
      h[i] = (h[i-1] + p*s[i-1]) % MOD;
      pi[i] = (pi[i-1] * PI) % MOD;
      p = (p*P) % MOD;
    }
  }
  ll get(int s, int e){ // get hash value of the substring [s, e)
    return (((h[e]-h[s]+MOD)%MOD)*pi[s])%MOD;
  }
};
```

5.3. Manacher

```
int d1[MAXN];//d1[i] = max odd palindrome centered on i
int d2[MAXN];//d2[i] = max even palindrome centered on i
//s  aabbaacaabbaa
```

```
//d1 1111117111111
//d2 0103010010301
void manacher(string &s) { // O(|S|) - find longest palindromic substring
  int l=0, r=-1, n=s.size();
  forn(i, n) { // build d1
    int k = i>r? 1 : min(d1[l+r-i], r-i);
    while(i+k<n && i-k>=0 && s[i+k]==s[i-k]) k++;
    d1[i] = k--;
    if(i+k > r) l=i-k, r=i+k;
  }
  l=0, r=-1;
  forn(i, n) { // build d2
    int k = (i>r? 0 : min(d2[l+r-i+1], r-i+1))+1;
    while(i+k<=n && i-k>=0 && s[i+k-1]==s[i-k]) k++;
    d2[i] = --k;
    if(i+k-1 > r) l=i-k, r=i+k-1;
  }
}
```

5.4. Z function

```
//z[i] = length of longest substring starting from s[i] that is prefix of s
//z[i] = max k: s[0,k) == s[i,i+k)
vector<int> zFunction(string &s) {
  int l=0, r=0, n=s.size();
  vector<int> z(n,0);
  forr(i, 1, n) {
    if(i<=r) z[i] = min(r-i+1, z[i-l]);
    while(i+z[i]<n && s[z[i]]==s[i+z[i]]) z[i]++;
    if(i+z[i]-1>r) l=i, r=i+z[i]-1;
  }
  return z;
}
void match(string &T,string &P) { //Text, Pattern -- O(|T|+|P|)
  string s = P+'$'+T;//'$' should be a character that is not present in T
  vector<int> z = zFunction(s);
  forr(i, P.size()+1, s.size())
    if(z[i] == P.size()); //match found, idx = i-P.size()-1
}
```

5.5. KMP

```
// b[i] = longest border of t[0,i) = length of the longest prefix of
// the substring P[0..i-1) that is also suffix of the substring P[0..i)
// For "AABAACAABAA", b[i] = {-1, 0, 1, 0, 1, 2, 0, 1, 2, 3, 4, 5}
vector<int> kmppre(string& P) { //
  vector<int> b(P.size()+1); b[0]=-1;
  int j=-1;
  forn(i, P.size()) {
    while(j>=0 && P[i]!=P[j]) j = b[j];
    b[i+1] = ++j;
  }
  return b;
}
void kmp(string& T, string& P) { //Text, Pattern -- O(|T|+|P|)
  int j = 0;
  vector<int> b = kmppre(P);
  forn(i, T.size()) {
    while(j>=0 && T[i]!=P[j]) j = b[j];
    if(++j == P.size())
    {
      //Match at i-j+1
      j=b[j];
    }
  }
}
```

5.6. Trie

```
struct trie{
  map<char, trie> m;
  void add(const string &s, int p=0)
  {
    if(s[p]) m[s[p]].add(s, p+1);
  }
  void dfs()
  {
    //Do stuff
    forall(it, m)
    it->second.dfs();
  }
};
```

5.7. Suffix array

5.7.1. Slow version O(n*logn*logn)

```
pair<int, int> sf[MAXN];
bool sacomp(int lhs, int rhs) {return sf[lhs] < sf[rhs];}
vector<int> constructSA(string& s) { // O(n log^2(n))
  int n = s.size();                  // (sometimes fast enough)
  vector<int> sa(n), r(n);
  forn(i,n) r[i] = s[i]; //r[i]: equivalence class of s[i..i+m)
  for(int m=1; m<n; m*=2) {
    //sf[i] = {r[i], r[i+m]}, used to sort for next equivalence classes
    forn(i,n) sa[i] = i, sf[i] = {r[i], i+m<n ? r[i+m] : -1};
```

```
    stable_sort(sa.begin(), sa.end(), sacomp); //O(n log(n))
    r[sa[0]] = 0;
    //if sf[sa[i]] == sf[sa[i-1]] then same equivalence class
    forr(i,1,n) r[sa[i]] = sf[sa[i]]!=sf[sa[i-1]] ? i : r[sa[i-1]];
  }
  return sa;
}
```

5.7.2. Fast version O(n*logn)

```
#define RB(x) (x<n ? r[x] : 0)
void csort(vector<int>& sa, vector<int>& r, int k){ //counting sort O(n)
    int n = sa.size();
    vector<int> f(max(255,n),0), t(n);
    forn(i, n) f[RB(i+k)]++;
    int sum = 0;
    forn(i, max(255,n)) f[i] = (sum+=f[i]) - f[i];
    forn(i, n) t[f[RB(sa[i]+k)]++] = sa[i];
    sa = t;
}
vector<int> constructSA(string& s){ // O(n logn)
    int n = s.size(), rank;
    vector<int> sa(n), r(n), t(n);
    forn(i,n) sa[i] = i, r[i] = s[i];//r[i]: equivalence class of s[i..i+k)
    for(int k=1; k<n; k*=2) {
        csort(sa, r, k); csort(sa, r, 0); //counting sort, O(n)
        t[sa[0]] = rank = 0; //t : equivalence classes array for next size
        forr(i, 1, n) {
            //check if sa[i] and sa[i-1] are in te same equivalence class
            if(r[sa[i]]!=r[sa[i-1]] || RB(sa[i]+k)!=RB(sa[i-1]+k)) rank++;
            t[sa[i]] = rank;
        }
        r = t;
        if(r[sa[n-1]] == n-1) break;
    }
    return sa;
}
```

5.8. Longest common prefix (LCP)

```
//LCP(sa[i], sa[j]) = min(lcp[i+1], lcp[i+2], ..., lcp[j])
//example: "banana", sa = {5,3,1,0,4,2}, lcp = {0,1,3,0,0,2}
//Num of dif substrings: (n*n+n)/2 - (sum over lcp array)
//Build suffix array (sa) before calling
vector<int> computeLCP(string& s, vector<int>& sa) {
  int n = s.size(), L = 0;
  vector<int> lcp(n), plcp(n), phi(n);
  phi[sa[0]] = -1;
  forr(i, 1, n) phi[sa[i]] = sa[i-1];
  forn(i, n) {
    if(phi[i]<0) {plcp[i] = 0; continue;}
    while(s[i+L] == s[phi[i]+L]) L++;
    plcp[i] = L;
    L = max(L-1, 0);
  }
  forn(i,n) lcp[i] = plcp[sa[i]];
  return lcp; // lcp[i]=LCP(sa[i-1],sa[i])
}
```

5.9. Suffix automaton

```
//The substrings of S can be decomposed into equivalence classes
//2 substr are of the same class if they have the same set of endpos
//Example: endpos("bc") = {2, 4, 6} in "abcbcbc"
//Each class is a node of the automaton.
//Len is the longest substring of each class
//Link in state X is the state where the longest suffix of the longest
//substring in X, with a different endpos set, belongs
//The links form a tree rooted at 0
//last is the state of the whole string after each extention
struct state {int len, link; map<char,int> next;}; //clear next!!
state st[MAXN];
int sz, last;
void sa_init() {
  last = st[0].len = 0; sz = 1;
  st[0].link = -1;
}
void sa_extend(char c) {
  int k = sz++, p; //k = new state
  st[k].len = st[last].len + 1;
  //while c is not present in p assign it as edge to the new state and
  //move through link (note that p always corresponds to a suffix state)
  for(p=last; p!=-1 && !st[p].next.count(c); p=st[p].link) st[p].next[c]=k;
  if(p == -1) st[k].link = 0;
  else {
    //state p already goes to state q through char c. Then, link of k
    //should go to a state with len = st[p].len + 1 (because of c)
    int q = st[p].next[c];
    if(st[p].len+1 == st[q].len) st[k].link = q;
    else {
      //q is not the state we are looking for. Then, we
      //create a clone of q (w) with the desired length
      int w = sz++;
      st[w].len = st[p].len + 1;
      st[w].next = st[q].next; st[w].link = st[q].link;
      //go through links from p and while next[c] is q, change it to w
      for(; p!=-1 && st[p].next[c]==q; p=st[p].link) st[p].next[c] = w;
      //change link of q from p to w, and finally set link of k to w
      st[q].link = st[k].link = w;
    }
```

```
  }
  last = k;
}
//  input: abcbcbc
//  i,link,len,next
//  0 -1 0 (a,1) (b,5) (c,7)
//  1 0 1 (b,2)
//  2 5 2 (c,3)
//  3 7 3 (b,4)
//  4 9 4 (c,6)
//  5 0 1 (c,7)
//  6 11 5 (b,8)
//  7 0 2 (b,9)
//  8 9 6 (c,10)
//  9 5 3 (c,11)
//  10 11 7
//  11 7 4 (b,8)
```

5.10. Suffix tree

```
//The SuffixTree of S is the compressed trie that would result
//after inserting every suffix of S.
//As it is a COMPRESSED trie, some edges may correspond to strings, instead
//of chars, and the compression is done in a way that every vertex that
//doesn't correspond to a suffix and has only one descendent, is omitted.
struct SuffixTree {
  char s[MAXN];
  map<int,int> to[2*MAXN];//key is fst char of substring on edge to value
  //s[fpos[i], fpos[i]+len[i]) is the substring on the edge between
  //i's father and i.
  //link[i] goes to the node that corresponds to the substring that result
  //after "removing" the first character of the substring that i represents
  //note that link is only defined for every internal (non-leaf) node.
  int len[2*MAXN] = {INF}, fpos[2*MAXN], link[2*MAXN];
  int node = 0, pos = 0, sz = 1, n = 0;
  int make_node(int p, int l) {
    fpos[sz] = p; len[sz] = l; return sz++;}
  void go_edge() {
    while(pos > len[to[node][s[n-pos]]]) {
      node = to[node][s[n-pos]];
      pos -= len[node];
    }
  }
  void add(char c) {
    s[n++] = c; pos++;
    int last = 0;
    while(pos > 0) {
      go_edge();
      int edge = s[n-pos];
      int& v = to[node][edge];
      int t = s[fpos[v]+pos-1];
      if(v == 0) {
        v = make_node(n-pos, INF);
        link[last] = node; last = 0;
      }
      else if(t == c) {link[last] = node; return;}
      else {
        int u = make_node(fpos[v], pos-1);
        to[u][c] = make_node(n-1, INF);
        to[u][t] = v;
        fpos[v] += pos-1; len[v] -= pos-1;
        v = u; link[last] = u; last = u;
      }
      if(node == 0) pos--;
      else node = link[node];
    }
  }
};
```

5.11. Aho Corasick

```
struct Trie{
  map<char, Trie> next;
  Trie* tran[256];//transiciones del automata
  int idhoja, szhoja;//id de la hoja o 0 si no lo es
  //link lleva al sufijo mas largo, nxthoja lleva al mas largo pero que es
      hoja
  Trie *padre, *link, *nxthoja;
  char pch;//caracter que conecta con padre
  //Trie(): tran(),  idhoja(), padre(), link() {}
  //comento linea de arriba porque me daba errores usarla.
  void insert(const string &s, int id=1, int p=0) //id>0!!!
  {
    if(p<sz(s))
    {
      Trie &ch=next[s[p]];
      tran[(int)s[p]]=&ch;
      ch.padre=this, ch.pch=s[p];
      ch.insert(s, id, p+1);
    }
    else idhoja=id, szhoja=sz(s);
  }
  Trie* get_link()
  {
    if(!link)
    {
      if(!padre) link=this;//es la raiz
      else if(!padre->padre) link=padre;//hijo de la raiz
      else link=padre->get_link()->get_tran(pch);
    }
    return link;
  }
```

```
  Trie* get_tran(int c)
  {
    if(!tran[c]) tran[c] = !padre? this : this->get_link()->get_tran(c);
    return tran[c];
  }
  Trie *get_nxthoja()
  {
    if(!nxthoja) nxthoja = get_link()->idhoja? link : link->nxthoja;
    return nxthoja;
  }
  void print(int p)
  {
    if(idhoja) cout << "found " << idhoja << "  at position " << p-szhoja <<
        endl;
    if(get_nxthoja()) get_nxthoja()->print(p);
  }
  void matching(const string &s, int p=0) //O(|s| + tamanio palabras)
  {
    print(p); if(p<sz(s)) get_tran(s[p])->matching(s, p+1);
  }
};
```

6. Geometría

6.1. Punto

```
struct pto{
  double x, y;
  pto(double x=0, double y=0):x(x),y(y){}
  pto operator+(pto a){return pto(x+a.x, y+a.y);}
  pto operator-(pto a){return pto(x-a.x, y-a.y);}
  pto operator+(double a){return pto(x+a, y+a);}
  pto operator*(double a){return pto(x*a, y*a);}
  pto operator/(double a){return pto(x/a, y/a);}
  //dot product, producto interno:
  double operator*(pto a){return x*a.x+y*a.y;}
  //module of the cross product or vectorial product:
  //if a is less than 180 clockwise from b, a^b>0
  double operator^(pto a){return x*a.y-y*a.x;}
  //returns true if this is at the left side of line qr
  bool left(pto q, pto r){return ((q-*this)^(r-*this))>0;}
  bool operator<(const pto &a) const{return x<a.x-EPS || (abs(x-a.x)<EPS &&
      y<a.y-EPS);}
bool operator==(pto a){return abs(x-a.x)<EPS && abs(y-a.y)<EPS;}
  double norm(){return sqrt(x*x+y*y);}
  double norm_sq(){return x*x+y*y;}
};
double dist(pto a, pto b){return (b-a).norm();}
typedef pto vec;

double angle(pto a, pto o, pto b){
  pto oa=a-o, ob=b-o;
  return atan2(oa^ob, oa*ob);}

//rotate p by theta rads CCW w.r.t. origin (0,0)
pto rotate(pto p, double theta){
  return pto(p.x*cos(theta)-p.y*sin(theta),
     p.x*sin(theta)+p.y*cos(theta));
}
```

6.2. Orden Radial de Puntos

```
struct Cmp{//orden total de puntos alrededor de un punto r
  pto r;
  Cmp(pto r):r(r) {}
  int cuad(const pto &a) const{
    if(a.x > 0 && a.y >= 0)return 0;
    if(a.x <= 0 && a.y > 0)return 1;
    if(a.x < 0 && a.y <= 0)return 2;
    if(a.x >= 0 && a.y < 0)return 3;
    assert(a.x ==0 && a.y==0);
    return -1;
  }
  bool cmp(const pto&p1, const pto&p2)const{
    int c1 = cuad(p1), c2 = cuad(p2);
    if(c1==c2) return p1.y*p2.x<p1.x*p2.y;
        else return c1 < c2;
  }
    bool operator()(const pto&p1, const pto&p2) const{
    return cmp(pto(p1.x-r.x,p1.y-r.y),pto(p2.x-r.x,p2.y-r.y));
    }
};
```

6.3. Linea

```
int sgn(ll x){return x<0? -1 : !!x;}
struct line{
  line() {}
  double a,b,c;//Ax+By=C
//pto MUST store float coordinates!
  line(double a, double b, double c):a(a),b(b),c(c){}
  // TO DO chequear porque paso problema metiendo negativo el C (-(todo el
      calculo como esta))
  line(pto p, pto q): a(q.y-p.y), b(p.x-q.x), c(a*p.x+b*p.y) {}
  int side(pto p){return sgn(ll(a) * p.x + ll(b) * p.y - c);}
};
bool parallels(line l1, line l2){return abs(l1.a*l2.b-l2.a*l1.b)<EPS;}
pto inter(line l1, line l2){//intersection
```

```
  double det=l1.a*l2.b-l2.a*l1.b;
  if(abs(det)<EPS) return pto(INF, INF);//parallels
  return pto(l2.b*l1.c-l1.b*l2.c, l1.a*l2.c-l2.a*l1.c)/det;
}
```

6.4. Segmento

```
struct segm{
  pto s,f;
  segm(pto s, pto f):s(s), f(f) {}
  pto closest(pto p) {//use for dist to point
     double l2 = dist_sq(s, f);
     if(l2==0.) return s;
     double t =((p-s)*(f-s))/l2;
     if (t<0.) return s;//not write if is a line
     else if(t>1.)return f;//not write if is a line
     return s+((f-s)*t);
  }
    bool inside(pto p){return abs(dist(s, p)+dist(p, f)-dist(s, f))<EPS;}
};

//NOTA: Si los segmentos son coolineales solo devuelve un punto de
    interseccion
pto inter(segm s1, segm s2){
    if(s1.inside(s2.s)) return s2.s; //Fix cuando son colineales
    if(s1.inside(s2.f)) return s2.f; //Fix cuando son colineales
  pto r=inter(line(s1.s, s1.f), line(s2.s, s2.f));
    if(s1.inside(r) && s2.inside(r)) return r;
  return pto(INF, INF);
}
```

6.5. Rectangulo

```
struct rect{
  //lower-left and upper-right corners
  pto lw, up;
};
//returns if there's an intersection and stores it in r
bool inter(rect a, rect b, rect &r){
  r.lw=pto(max(a.lw.x, b.lw.x), max(a.lw.y, b.lw.y));
  r.up=pto(min(a.up.x, b.up.x), min(a.up.y, b.up.y));
//check case when only a edge is common
  return r.lw.x<r.up.x && r.lw.y<r.up.y;
}
```

6.6. Circulo

```
vec perp(vec v){return vec(-v.y, v.x);}
line bisector(pto x, pto y){
  line l=line(x, y); pto m=(x+y)/2;
  return line(-l.b, l.a, -l.b*m.x+l.a*m.y);
}
struct Circle{
  pto o;
  double r;
  Circle(pto x, pto y, pto z){
    o=inter(bisector(x, y), bisector(y, z));
    r=dist(o, x);
  }
  pair<pto, pto> ptosTang(pto p){
    pto m=(p+o)/2;
    tipo d=dist(o, m);
    tipo a=r*r/(2*d);
    tipo h=sqrt(r*r-a*a);
    pto m2=o+(m-o)*a/d;
    vec per=perp(m-o)/d;
    return make_pair(m2-per*h, m2+per*h);
  }
};
//finds the center of the circle containing p1 and p2 with radius r
//as there may be two solutions swap p1, p2 to get the other
bool circle2PtsRad(pto p1, pto p2, double r, pto &c){
        double d2=(p1-p2).norm_sq(), det=r*r/d2-0.25;
        if(det<0) return false;
        c=(p1+p2)/2+perp(p2-p1)*sqrt(det);
        return true;
}
#define sqr(a) ((a)*(a))
#define feq(a,b) (fabs((a)-(b))<EPS)
pair<tipo, tipo> ecCuad(tipo a, tipo b, tipo c){//a*x*x+b*x+c=0
  tipo dx = sqrt(b*b-4.0*a*c);
  return make_pair((-b + dx)/(2.0*a),(-b - dx)/(2.0*a));
}
pair<pto, pto> interCL(Circle c, line l){
  bool sw=false;
  if((sw=feq(0,l.b))){
  swap(l.a, l.b);
  swap(c.o.x, c.o.y);
  }
  pair<tipo, tipo> rc = ecCuad(
  sqr(l.a)+sqr(l.b),
  2.0*l.a*l.b*c.o.y-2.0*(sqr(l.b)*c.o.x+l.c*l.a),
  sqr(l.b)*(sqr(c.o.x)+sqr(c.o.y)-sqr(c.r))+sqr(l.c)-2.0*l.c*l.b*c.o.y
  );
  pair<pto, pto> p( pto(rc.first, (l.c - l.a * rc.first) / l.b),
              pto(rc.second, (l.c - l.a * rc.second) / l.b) );
  if(sw){
  swap(p.first.x, p.first.y);
  swap(p.second.x, p.second.y);
  }
```

```
  return p;
}
pair<pto, pto> interCC(Circle c1, Circle c2){
  line l;
  l.a = c1.o.x-c2.o.x;
  l.b = c1.o.y-c2.o.y;
  l.c = (sqr(c2.r)-sqr(c1.r)+sqr(c1.o.x)-sqr(c2.o.x)+sqr(c1.o.y)
  -sqr(c2.o.y))/2.0;
  return interCL(c1, l);
}
```

6.7. Area de poligono

```
double area(vector<pto> &p){//O(sz(p))
  double area=0;
  forn(i, sz(p)) area+=p[i]^p[(i+1)%sz(p)];
  //if points are in clockwise order then area is negative
  return abs(area)/2;
}
//Area ellipse = M_PI*a*b where a and b are the semi axis lengths
//Area triangle = sqrt(s*(s-a)(s-b)(s-c)) where s=(a+b+c)/2
```

6.8. Punto en poligono

```
//checks if v is inside of P, using ray casting
//works with convex and concave.
//excludes boundaries, handle it separately using segment.inside()
bool inPolygon(pto v, vector<pto>& P) {
  bool c = false;
  forn(i, sz(P)){
    int j=(i+1)%sz(P);
    if((P[j].y>v.y) != (P[i].y > v.y) &&
  (v.x < (P[i].x - P[j].x) * (v.y-P[j].y) / (P[i].y - P[j].y) + P[j].x))
      c = !c;
  }
  return c;
}
```

6.9. Punto en Poligono Convexo

$O(\log n)$

```
void normalize(vector<pto> &pt) //delete collinear points first!
{
  //this makes it clockwise:
  if(pt[2].left(pt[0], pt[1])) reverse(pt.begin(), pt.end());
  int n=sz(pt), pi=0;
  forn(i, n)
    if(pt[i].x<pt[pi].x || (pt[i].x==pt[pi].x && pt[i].y<pt[pi].y))
      pi=i;
  vector<pto> shift(n);//puts pi as first point
  forn(i, n) shift[i]=pt[(pi+i)%n];
  pt.swap(shift);
}
bool inPolygon(pto p, const vector<pto> &pt)
{
  //call normalize first!
  if(p.left(pt[0], pt[1]) || p.left(pt[sz(pt)-1], pt[0])) return false;
  int a=1, b=sz(pt)-1;
  while(b-a>1)
  {
    int c=(a+b)/2;
    if(!p.left(pt[0], pt[c])) a=c;
    else b=c;
  }
  return !p.left(pt[a], pt[a+1]);
}
```

6.10. Chequeo de Convex

```

bool isConvex(vector<int> &p){//O(N), delete collinear points!
  int N=sz(p);
  if(N<3) return false;
  bool isLeft=p[0].left(p[1], p[2]);
  forr(i, 1, N)
    if(p[i].left(p[(i+1)%N], p[(i+2)%N])!=isLeft)
      return false;
  return true; }
```

6.11. Convex Hull

```
//stores convex hull of P in S, CCW order
//left must return >=0 to delete collinear points!
void CH(vector<pto>& P, vector<pto> &S){
  S.clear();
  sort(P.begin(), P.end());//first x, then y
  forn(i, sz(P)){//lower hull
    while(sz(S)>= 2 && S[sz(S)-1].left(S[sz(S)-2], P[i])) S.pop_back();
    S.pb(P[i]);
  }
  S.pop_back();
  int k=sz(S);
  dforn(i, sz(P)){//upper hull
    while(sz(S) >= k+2 && S[sz(S)-1].left(S[sz(S)-2], P[i])) S.pop_back();
```

```
    S.pb(P[i]);
  }
  S.pop_back();
}
```

6.12. Convex Hull Trick

```
struct Line{tipo m,h;};
tipo inter(Line a, Line b){
    tipo x=b.h-a.h, y=a.m-b.m;
    return x/y+(x%y?!((x>0)^(y>0)):0);//==ceil(x/y)
}
struct CHT {
  vector<Line> c;
  bool mx;
  int pos;
  CHT(bool mx=0):mx(mx),pos(0){}//mx=1 si las query devuelven el max
  inline Line acc(int i){return c[c[0].m>c.back().m? i : sz(c)-1-i];}
  inline bool irre(Line x, Line y, Line z){
    return c[0].m>z.m? inter(y, z) <= inter(x, y)
                     : inter(y, z) >= inter(x, y);
  }
  void add(tipo m, tipo h) {//O(1), los m tienen que entrar ordenados
        if(mx) m*=-1, h*=-1;
    Line l=(Line){m, h};
        if(sz(c) && m==c.back().m) { l.h=min(h, c.back().h), c.pop_back();
            if(pos) pos--; }
        while(sz(c)>=2 && irre(c[sz(c)-2], c[sz(c)-1], l)) { c.pop_back();
            if(pos) pos--; }
        c.pb(l);
  }
  inline bool fbin(tipo x, int m) {return inter(acc(m), acc(m+1))>x;}
  tipo eval(tipo x){
    int n = sz(c);
    //query con x no ordenados O(lgn)
    int a=-1, b=n-1;
    while(b-a>1) { int m = (a+b)/2;
      if(fbin(x, m)) b=m;
      else a=m;
    }
    return (acc(b).m*x+acc(b).h)*(mx?-1:1);
        //query O(1)
    while(pos>0 && fbin(x, pos-1)) pos--;
    while(pos<n-1 && !fbin(x, pos)) pos++;
    return (acc(pos).m*x+acc(pos).h)*(mx?-1:1);
  }
} ch;
```

6.13. Convex Hull Trick Dinamico

```
const ll is_query = -(1LL<<62);
struct Line {
    ll m, b;
    mutable multiset<Line>::iterator it;
    const Line *succ(multiset<Line>::iterator it) const;
    bool operator<(const Line& rhs) const {
```

```
        if (rhs.b != is_query) return m < rhs.m;
        const Line *s=succ(it);
        if(!s) return 0;
        ll x = rhs.m;
        return b - s->b < (s->m - m) * x;
    }
};
struct HullDynamic : public multiset<Line>{ // will maintain upper hull for
    maximum
    bool bad(iterator y) {
        iterator z = next(y);
        if (y == begin()) {
            if (z == end()) return 0;
            return y->m == z->m && y->b <= z->b;
        }
        iterator x = prev(y);
        if (z == end()) return y->m == x->m && y->b <= x->b;
        return (x->b - y->b)*(z->m - y->m) >= (y->b - z->b)*(y->m - x->m);
    }
    iterator next(iterator y){return ++y;}
    iterator prev(iterator y){return --y;}
    void insert_line(ll m, ll b) {
        iterator y = insert((Line) { m, b });
        y->it=y;
        if (bad(y)) { erase(y); return; }
        while (next(y) != end() && bad(next(y))) erase(next(y));
        while (y != begin() && bad(prev(y))) erase(prev(y));
    }
    ll eval(ll x) {
        Line l = *lower_bound((Line) { x, is_query });
        return l.m * x + l.b;
    }
}h;
const Line *Line::succ(multiset<Line>::iterator it) const{
    return (++it==h.end()? NULL : &*it);}
```

6.14. Cortar poligono

```
//cuts polygon Q along the line ab
//stores the left side (swap a, b for the right one) in P
void cutPolygon(pto a, pto b, vector<pto> Q, vector<pto> &P){
  P.clear();
  forn(i, sz(Q)){
    double left1=(b-a)^(Q[i]-a), left2=(b-a)^(Q[(i+1)%sz(Q)]-a);
    if(left1>=0) P.pb(Q[i]);
    if(left1*left2<0)
      P.pb(inter(line(Q[i], Q[(i+1)%sz(Q)]), line(a, b)));
  }
}
```

6.15. Intersección de Circulos

```
struct event {
  double x; int t;
  event(double xx, int tt) : x(xx), t(tt) {}
  bool operator <(const event &o) const { return x < o.x; }
};
typedef vector<Circle> VC;
typedef vector<event> VE;
int n;
double cuenta(VE &v, double A,double B)
{
  sort(v.begin(), v.end());
  double res = 0.0, lx = ((v.empty())?0.0:v[0].x);
  int contador = 0;
  forn(i,sz(v))
  { //interseccion de todos (contador == n), union de todos (contador > 0)
    //conjunto de puntos cubierto por exacta k Circulos (contador == k)
    if (contador == n) res += v[i].x - lx;
    contador += v[i].t, lx = v[i].x;
  }
  return res;
}
// Primitiva de sqrt(r*r - x*x) como funcion double de una variable x.
inline double primitiva(double x,double r)
{
  if (x >= r) return r*r*M_PI/4.0;
  if (x <= -r) return -r*r*M_PI/4.0;
  double raiz = sqrt(r*r-x*x);
  return 0.5 * (x * raiz + r*r*atan(x/raiz));
}
double interCircle(VC &v)
{
  vector<double> p; p.reserve(v.size() * (v.size() + 2));
  forn(i,sz(v))  p.push_back(v[i].c.x + v[i].r), p.push_back(v[i].c.x - v[i
      ].r);
  forn(i,sz(v)) forn(j,i)
  {
    Circle &a = v[i], b = v[j];
    double d = (a.c - b.c).norm();
    if (fabs(a.r - b.r) < d && d < a.r + b.r)
    {
      double alfa = acos((sqr(a.r) + sqr(d) - sqr(b.r)) / (2.0 * d * a.r));
      pto vec = (b.c - a.c) * (a.r / d);
      p.pb((a.c + rotate(vec, alfa)).x), p.pb((a.c + rotate(vec, -alfa)).x);
    }
  }
  sort(p.begin(), p.end());
  double res = 0.0;
  forn(i,sz(p)-1)
  {
    const double A = p[i], B = p[i+1];
    VE ve; ve.reserve(2 * v.size());
```

```
    forn(j,sz(v))
    {
      const Circle &c = v[j];
      double arco = primitiva(B-c.c.x,c.r) - primitiva(A-c.c.x,c.r);
      double base = c.c.y * (B-A);
      ve.push_back(event(base + arco,-1));
      ve.push_back(event(base - arco, 1));
    }
    res += cuenta(ve,A,B);
  }
  return res;
}
```

6.16. Rotar Matriz

```
//rotates matrix t 90 degrees clockwise
//using auxiliary matrix t2(faster)
void rotate()
{
  forn(x, n) forn(y, n)
    t2[n-y-1][x]=t[x][y];
  memcpy(t, t2, sizeof(t));
}
```

7. Matemática

7.1. Identidades

$$\sum_{i=0}^{n} \binom{n}{i} = 2^n$$
$$\sum_{i=0}^{n} i\binom{n}{i} = n * 2^{n-1}$$
$$\sum_{i=m}^{n} i = \frac{n(n+1)}{2} - \frac{m(m-1)}{2} = \frac{(n+1-m)(n+m)}{2}$$
$$\sum_{i=0}^{n} i = \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$
$$\sum_{i=0}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} = \frac{n^3}{3} + \frac{n^2}{2} + \frac{n}{6}$$
$$\sum_{i=0}^{n} i(i-1) = \frac{8}{6}(\frac{n}{2})(\frac{n}{2}+1)(n+1) \text{ (doubles)} \rightarrow \text{Sino ver caso impar y par}$$
$$\sum_{i=0}^{n} i^3 = \left(\frac{n(n+1)}{2}\right)^2 = \frac{n^4}{4} + \frac{n^3}{2} + \frac{n^2}{4} = \left[\sum_{i=1}^{n} i\right]^2$$
$$\sum_{i=0}^{n} i^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30} = \frac{n^5}{5} + \frac{n^4}{2} + \frac{n^3}{3} - \frac{n}{30}$$
$$\sum_{i=0}^{n} i^p = \frac{(n+1)^{p+1}}{p+1} + \sum_{k=1}^{p} \frac{B_k}{p-k+1}\binom{p}{k}(n+1)^{p-k+1}$$
$$r = e - v + k + 1$$

Teorema de Pick: (Area, puntos interiores y puntos en el borde)

$$A = I + \frac{B}{2} - 1$$

7.2. Ec. Caracteristica

$$a_0T(n) + a_1T(n-1) + ... + a_kT(n-k) = 0$$
$$p(x) = a_0x^k + a_1x^{k-1} + ... + a_k$$

Sean $r_1, r_2, ..., r_q$ las raíces distintas, de mult. $m_1, m_2, ..., m_q$

$$T(n) = \sum_{i=1}^{q} \sum_{j=0}^{m_i-1} c_{ij} n^j r_i^n$$

Las constantes c_{ij} se determinan por los casos base.

7.3. Teorema Chino del Resto

$$y = \sum_{j=1}^{n} (x_j * (\prod_{i=1, i\neq j}^{n} m_i)^{-1}_{m_j} * \prod_{i=1, i\neq j}^{n} m_i)$$

```
//Chinese remainder theorem (special case): find z such that
//z % m1 = r1, z % m2 = r2.  Here, z is unique modulo M = lcm(m1, m2).
//Return (z, M).  On failure, M = -1.
ii chinese_remainder_theorem(int m1, int r1, int m2, int r2)
{ //{xx,yy,d} son variables globales usadas en extendedEuclid
  extendedEuclid(m1, m2);
  if (r1%d != r2%d) return make_pair(0,-1);
```

```
  return mp(sumMod(xx*r2*m1, yy*r1*m2, m1*m2) / d, m1*m2 / d);
}
//Chinese remainder theorem: find z such that z % m[i] = r[i] for all i.
//Note that the solution is unique modulo M = lcm_i (m[i]).
//Return (z, M). On failure, M = -1.
//Note that we do not require the a[i]'s to be relatively prime.
ii chinese_remainder_theorem(const vector<int> &m, const vector<int> &r)
{
  ii ret=mp(r[0], m[0]);
  forr(i,1,m.size())
  {
    ret=chinese_remainder_theorem(ret.snd, ret.fst, m[i], r[i]);
    if (ret.snd==-1) break;
  }
  return ret;
}
```

7.4. GCD & LCM

```
int gcd(int a, int b) {return b? gcd(b,a%b) : a;}
int lcm(int a, int b) {return a*(b/gcd(a,b));}
```

7.5. Euclides Extendido

```
//ecuacion diofantica lineal
//sea d=gcd(a,b); la ecuacion a * x + b * y = c tiene soluciones enteras si
//d|c. La siguiente funcion nos sirve para esto. De forma general sera:
//x = x0 + (b/d)n     x0 = xx*c/d
//y = y0 - (a/d)n     y0 = yy*c/d
ll xx,yy,d;
void extendedEuclid(ll a, ll b)  //a * xx + b * yy = d
{
  if (!b) {xx=1; yy=0; d=a; return;}
  extendedEuclid (b,a%b);
  ll x1=yy;
  ll y1=xx-(a/b)*yy;
  xx=x1; yy=y1;
}
```

7.6. Combinatoria

```
void cargarComb()//O(MAXN^2)
{
  forn(i, MAXN+1) //comb[i][k]=i tomados de a k = i!/(k!*(i-k)!)
  {
    comb[0][i]=0;
    comb[i][0]=comb[i][i]=1;
    forr(k, 1, i) comb[i][k]=(comb[i-1][k-1]+comb[i-1][k]) %MOD;
  }
}
ll lucas (ll n, ll k, int p)
{ //Calcula (n,k)%p teniendo comb[p][p] precalculado.
  ll aux = 1;
  while (n + k)
  {
    aux = (aux * comb[n %p][k %p]) %p;
    n/=p, k/=p;
  }
  return aux;
}
```

7.7. Exponenciación de Matrices y Fibonacci

```
#define SIZE 350
int NN;
void mul(double a[SIZE][SIZE], double b[SIZE][SIZE])
{
  double res[SIZE][SIZE] = {{0}};
  forn(i, NN) forn(j, NN) forn(k, NN) res[i][j]+=a[i][k]*b[k][j];
  forn(i, NN) forn(j, NN) a[i][j]=res[i][j];
}
void powmat(double a[SIZE][SIZE], int n, double res[SIZE][SIZE])
{
  forn(i, NN) forn(j, NN) res[i][j]=(i==j);
  while(n)
  {
    if(n&1) mul(res, a), n--;
    else mul(a, a), n/=2;
  }
}

struct M22{    // |a b|
  tipo a,b,c,d;// |c d| -- TIPO
  M22 operator*(const M22 &p) const {
  return (M22){a*p.a+b*p.c, a*p.b+b*p.d, c*p.a+d*p.c,c*p.b+d*p.d};}
};
M22 operator^(const M22 &p, int n)
{//VER COMO SE PUEDE PONER DENTRO DEL STRUCT
  if(!n) return (M22){1, 0, 0, 1};//identidad
  M22 q=p^(n/2); q=q*q;
  return n %2? p * q : q;
}

ll fibo(ll n)//calcula el fibonacci enesimo en O(logN)
{
  M22 mat=(M22){0, 1, 1, 1}^n;
  return mat.a*f0+mat.b*f1;//f0 y f1 son los valores iniciales
```

```
}
```

7.8. Operaciones Modulares

```
ll mulMod(ll a,ll b,ll m=MOD) //O(log b)
{ //returns (a*b) %c, and minimize overfloor
  ll x=0, y=a%m;
  while(b>0)
  {
    if(b%2==1) x=(x+y)%m;
    y=(y*2)%m;
    b/=2;
  }
  return x%m;
}
ll expMod(ll b,ll e,ll m=MOD) //O(log b)
{
  if(!e) return 1;
  ll q=expMod(b,e/2,m);
  q=mulMod(q,q,m);
  return e%2? mulMod(b,q,m) : q;
}
ll sumMod(ll a,ll b,ll m=MOD)
{
  a%=m;
  b%=m;
  if(a<0) a+=m;
  if(b<0) b+=m;
  return (a+b)%m;
}
ll difMod(ll a,ll b,ll m=MOD)
{
  a%=m;
  b%=m;
  if(a<0) a+=m;
  if(b<0) b+=m;
  ll ret=a-b;
  if(ret<0) ret+=m;
  return ret;
}
ll divMod(ll a,ll b,ll m=MOD)
{
  return mulMod(a,inverso(b),m);
}
```

7.9. Funciones de Primos

Sea $n = \prod p_i^{k_i}$, fact(n) genera un map donde a cada p_i le asocia su k_i

```
#define MAXP 100000 //no necesariamente primo
int criba[MAXP+1];
void crearCriba()
{
  int w[] = {4,2,4,2,4,6,2,6};
  for(int p=25;p<=MAXP;p+=10) criba[p]=5;
  for(int p=9;p<=MAXP;p+=6) criba[p]=3;
  for(int p=4;p<=MAXP;p+=2) criba[p]=2;
  for(int p=7,cur=0;p*p<=MAXP;p+=w[cur++&7]) if (!criba[p])
  for(int j=p*p;j<=MAXP;j+=(p<<1)) if(!criba[j]) criba[j]=p;
}
vector<int> primos;
void buscarPrimos()
{
  crearCriba();
  forr (i,2,MAXP+1) if (!criba[i]) primos.push_back(i);
}

//factoriza bien numeros hasta MAXP^2
void fact(ll n,map<ll,ll> &f) //O (cant primos)
{ //llamar a buscarPrimos antes
  forall(p, primos){
    while(!(n %*p))
    {
      f[*p]++;//divisor found
      n/=*p;
    }
  }
  if(n>1) f[n]++;
}

//factoriza bien numeros hasta MAXP
void fact2(ll n,map<ll,ll> &f) //O (lg n)
{ //llamar a crearCriba antes
  while (criba[n])
  {
    f[criba[n]]++;
    n/=criba[n];
  }
  if(n>1) f[n]++;
}

//Usar asi: divisores(fac, divs, fac.begin()); NO ESTA ORDENADO
void divisores(map<ll,ll> &f,vector<ll> &divs,map<ll,ll>::iterator it,ll n
    =1)
{
  if(it==f.begin()) divs.clear();
  if(it==f.end())
  {
```

```
    divs.pb(n);
    return;
  }
  ll p=it->fst, k=it->snd; ++it;
  forn(_, k+1) divisores(f, divs, it, n), n*=p;
}
ll cantDivs(map<ll,ll> &f)
{
  ll ret=1;
  forall(it, f) ret*=(it->second+1);
  return ret;
}
ll sumDivs(map<ll,ll> &f)
{
  ll ret=1;
  forall(it, f)
  {
    ll pot=1, aux=0;
    forn(i, it->snd+1) aux+=pot, pot*=it->fst;
    ret*=aux;
  }
  return ret;
}

ll eulerPhi(ll n) // con criba: O(lg n)
{
  map<ll,ll> f;
  fact(n,f);
  ll ret=n;
  forall(it, f) ret-=ret/it->first;
  return ret;
}
ll eulerPhi2(ll n) // O (sqrt n)
{
  ll r = n;
  forr(i,2,n+1)
  {
    if((ll)i*i>n) break;
    if(n%i==0)
    {
      while(n%i==0) n/=i;
      r -= r/i;
    }
  }
  if (n != 1) r-= r/n;
  return r;
}
```

7.10. Phollard's Rho

```
bool es_primo_prob(ll n, int a)
{
  if(n==a) return true;
  ll s=0,d=n-1;
  while(d%2==0) s++,d/=2;
  ll x=expMod(a,d,n);
  if((x==1) || (x+1==n)) return true;
  forn(i,s-1)
  {
    x=mulMod(x, x, n);
    if(x==1) return false;
    if(x+1==n) return true;
  }
  return false;
}
bool rabin (ll n)  //devuelve true si n es primo
{
  if(n==1) return false;
  const int ar[]={2,3,5,7,11,13,17,19,23};
  forn(j,9) if(!es_primo_prob(n,ar[j])) return false;
  return true;
}
ll rho(ll n)
{
  if((n&1)==0) return 2;
  ll x=2,y=2,d=1;
  ll c=rand()%n+1;
  while(d==1)
  {
    x=(mulMod(x,x,n)+c)%n;
    y=(mulMod(y,y,n)+c)%n;
    y=(mulMod(y,y,n)+c)%n;
    if(x-y>=0) d=gcd(n,x-y);
    else d=gcd(n,y-x);
  }
  return d==n? rho(n):d;
}
void factRho (ll n,map<ll,ll> &f) //O (lg n)^3 un solo numero
{
  if (n == 1) return;
  if (rabin(n))
  {
    f[n]++;
    return;
  }
  ll factor = rho(n);
  factRho(factor,f);
  factRho(n/factor,f);
}
```

7.11. Inversos

```
#define MAXMOD 15485867
ll inv[MAXMOD];//inv[i]*i=1 mod MOD
void calc(int p) //O(p)
{
  inv[1]=1;
  forr(i,2,p) inv[i]=p-((p/i)*inv[p%i])%p;
}
int inverso(int x) //O(log x)
{
  return expMod(x, eulerPhi(MOD)-2);//si mod no es primo(sacar a mano)
  return expMod(x, MOD-2);//si mod es primo
}
```

7.12. Fracciones

```
struct frac{
  int p,q;
  frac(int p=0,int q=1):p(p),q(q) {norm();}
  void norm()
  {
    int a=gcd(q,p);
    if(a) p/=a, q/=a;
    else q=1;
    if (q<0) q=-q, p=-p;
  }
  frac operator+(const frac& o)
  {
    int a=gcd(o.q,q);
    return frac(p*(o.q/a)+o.p*(q/a),q*(o.q/a));
  }
  frac operator-(const frac& o)
  {
    int a=gcd(o.q,q);
    return frac(p*(o.q/a)-o.p*(q/a),q*(o.q/a));
  }
  frac operator*(frac o)
  {
    int a=gcd(o.p,q), b=gcd(p,o.q);
    return frac((p/b)*(o.p/a),(q/a)*(o.q/b));
  }
  frac operator/(frac o)
  {
    int a=gcd(o.q,q), b=gcd(p,o.p);
    return frac((p/b)*(o.q/a),(q/a)*(o.p/b));
  }
  bool operator<(const frac &o) const{return p*o.q < o.p*q;}
  bool operator==(frac o){return p==o.p&&q==o.q;}
};
```

7.13. Simpson

```
double integral(double a, double b, int n=10000) //O(n), n=cantdiv
{
  double area=0, h=(b-a)/n, fa=f(a), fb;
  forn(i, n)
  {
    fb=f(a+h*(i+1));
    area+=fa+ 4*f(a+h*(i+0.5)) +fb, fa=fb;
  }
  return area*h/6.;
}
```

7.14. Tablas y cotas (Primos, Divisores, Factoriales, etc)

Factoriales

0! = 1	11! = 39.916.800
1! = 1	12! = 479.001.600 (∈ `int`)
2! = 2	13! = 6.227.020.800
3! = 6	14! = 87.178.291.200
4! = 24	15! = 1.307.674.368.000
5! = 120	16! = 20.922.789.888.000
6! = 720	17! = 355.687.428.096.000
7! = 5.040	18! = 6.402.373.705.728.000
8! = 40.320	19! = 121.645.100.408.832.000
9! = 362.880	20! = 2.432.902.008.176.640.000 (∈ `tint`)
10! = 3.628.800	21! = 51.090.942.171.709.400.000

max signed tint = 9.223.372.036.854.775.807

max unsigned tint = 18.446.744.073.709.551.615

Primos

2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97 101 103 107 109 113 127 131 137 139 149 151 157 163 167 173 179 181 191 193 197 199 211 223 227 229 233 239 241 251 257 263 269 271 277 281 283 293 307 311 313 317 331 337 347 349 353 359 367 373 379 383 389 397 401 409 419 421 431 433 439 443 449 457 461 463 467 479 487 491 499 503 509 521 523 541 547 557 563 569 571 577 587 593 599 601 607 613 617 619 631 641 643 647 653 659 661 673 677 683 691 701 709 719 727 733 739 743 751 757 761 769 773 787 797 809 811 821 823 827 829 839 853 857 859 863 877 881 883 887 907 911 919 929 937 941 947 953 967 971 977 983 991 997 1009 1013

1019 1021 1031 1033 1039 1049 1051 1061 1063 1069 1087 1091 1093 1097 1103 1109 1117 1123 1129 1151 1153 1163 1171 1181 1187 1193 1201 1213 1217 1223 1229 1231 1237 1249 1259 1277 1279 1283 1289 1291 1297 1301 1303 1307 1319 1321 1327 1361 1367 1373 1381 1399 1409 1423 1427 1429 1433 1439 1447 1451 1453 1459 1471 1481 1483 1487 1489 1493 1499 1511 1523 1531 1543 1549 1553 1559 1567 1571 1579 1583 1597 1601 1607 1609 1613 1619 1621 1627 1637 1657 1663 1667 1669 1693 1697 1699 1709 1721 1723 1733 1741 1747 1753 1759 1777 1783 1787 1789 1801 1811 1823 1831 1847 1861 1867 1871 1873 1877 1879 1889 1901 1907 1913 1931 1933 1949 1951 1973 1979 1987 1993 1997 1999 2003 2011 2017 2027 2029 2039 2053 2063 2069 2081

Primos cercanos a 10^n

9941 9949 9967 9973 10007 10009 10037 10039 10061 10067 10069 10079
99961 99971 99989 99991 100003 100019 100043 100049 100057 100069
999959 999961 999979 999983 1000003 1000033 1000037 1000039
9999943 9999971 9999973 9999991 10000019 10000079 10000103 10000121
99999941 99999959 99999971 99999989 100000007 100000037 100000039 100000049
999999893 999999929 999999937 1000000007 1000000009 1000000021 1000000033

Cantidad de primos menores que 10^n

$\pi(10^1) = 4$; $\pi(10^2) = 25$; $\pi(10^3) = 168$; $\pi(10^4) = 1229$; $\pi(10^5) = 9592$
$\pi(10^6) = 78.498$; $\pi(10^7) = 664.579$; $\pi(10^8) = 5.761.455$; $\pi(10^9) = 50.847.534$
$\pi(10^{10}) = 455.052{,}511$; $\pi(10^{11}) = 4.118.054.813$; $\pi(10^{12}) = 37.607.912.018$

7.15. Números Catalanes

Utiles para problemas de Combinatoria

$$Cat(n) = \frac{\binom{2n}{n}}{n+1} = \frac{(2n)!}{n!\,(n+1)!}$$

Con $Cat(0) = 1$.

Diferentes aplicaciones:
1. Contar la cantidad de diferentes arboles binarios con n nodos que se pueden armar.
2. Contar las formas en que un polígono convexo de $n + 2$ lados puede ser triangulado.
3. Contar la cantidad de caminos monotonos a lo largo de los lados de una grilla $n * n$, que no cruzan la diagonal.
4. Contar el número de expresiones que contienen n pares de paréntesis correctamente colocados

7.15.1. Primeros 25 Catalanes

1 1 2 5 14 42 132 429 1430 4862 16796 58786 208012 742900 2674440 9694845 35357670 129644790 477638700 1767263190 6564120420 24466267020 91482563640 343059613650 1289904147324 4861946401452

8. Grafos

8.1. Dijkstra

```
#define INF 1e9
int N;
#define MAX_V 250001
vector<ii> G[MAX_V];
//To add an edge use
#define add(a, b, w) G[a].pb(make_pair(w, b))
ll dijkstra(int s, int t){//O(|E| log |V|)
  priority_queue<ii, vector<ii>, greater<ii> > Q;
  vector<ll> dist(N, INF); vector<int> dad(N, -1);
  Q.push(make_pair(0, s)); dist[s] = 0;
  while(sz(Q)){
    ii p = Q.top(); Q.pop();
    if(p.snd == t) break;
    forall(it, G[p.snd])
      if(dist[p.snd]+it->first < dist[it->snd]){
        dist[it->snd] = dist[p.snd] + it->fst;
        dad[it->snd] = p.snd;
        Q.push(make_pair(dist[it->snd], it->snd));  }
  }
  return dist[t];
  if(dist[t]<INF)//path generator
    for(int i=t; i!=-1; i=dad[i])
      printf("%d%c", i, (i==s?'\n':' '));}
```

8.2. Bellman-Ford

```
//Mas lento que Dijsktra, pero maneja arcos con peso negativo
vector<ii> G[MAX_N];//ady. list with pairs (weight, dst)
int dist[MAX_N];
void bford(int src){//O(VE)
  dist[src]=0;
  forn(i, N-1) forn(j, N) if(dist[j]!=INF) forall(it, G[j])
    dist[it->snd]=min(dist[it->snd], dist[j]+it->fst);
}

bool hasNegCycle(){
  forn(j, N) if(dist[j]!=INF) forall(it, G[j])
    if(dist[it->snd]>dist[j]+it->fst) return true;
  //inside if: all points reachable from it->snd will have -INF distance(do
      bfs)
  return false;
}
```

8.3. Floyd-Warshall

```
// Camino minimo en grafos dirigidos ponderados, en todas las parejas de
    nodos.
//G[i][j] contains weight of edge (i, j) or INF
//G[i][i]=0
int G[MAX_N][MAX_N];
void floyd(){//O(N^3)
forn(k, N) forn(i, N) if(G[i][k]!=INF) forn(j, N) if(G[k][j]!=INF)
  G[i][j]=min(G[i][j], G[i][k]+G[k][j]);
}
bool inNegCycle(int v){
  return G[v][v]<0;}
//checks if there's a neg. cycle in path from a to b
bool hasNegCycle(int a, int b){
  forn(i, N) if(G[a][i]!=INF && G[i][i]<0 && G[i][b]!=INF)
    return true;
  return false;
}
```

8.4. Kruskal

```
struct UF{
    void init(int n){}
    void unir(int a, int v){}
    int comp(int n){return 0;}
}uf;
vector<ii> G[MAXN];
int n;

struct Ar{int a,b,w;};
bool operator<(const Ar& a, const Ar &b){return a.w<b.w;}
vector<Ar> E;

// Minimun Spanning Tree in O(e log e)
ll kruskal(){
    ll cost=0;
    sort(E.begin(), E.end());//ordenar aristas de menor a mayor
    uf.init(n);
    forall(it, E){
        if(uf.comp(it->a)!=uf.comp(it->b)){//si no estan conectados
            uf.unir(it->a, it->b);//conectar
            cost+=it->w;
        }
    }
    return cost;
}
```

8.5. Prim

```
vector<ii> G[MAXN];
bool taken[MAXN];
priority_queue<ii, vector<ii>, greater<ii> > pq;//min heap
void process(int v){
    taken[v]=true;
    forall(e, G[v])
        if(!taken[e->second]) pq.push(*e);
}
// Minimun Spanning Tree in O(n^2)
ll prim(){
    zero(taken);
    process(0);
    ll cost=0;
    while(sz(pq)){
        ii e=pq.top(); pq.pop();
        if(!taken[e.second]) cost+=e.first, process(e.second);
    }
    return cost;
}
```

8.6. Kosaraju SCC

Componente Fuertemente Conexa

```
#define MAXN 1000000
vector<int> G[MAXN],gt[MAXN]; //Limpiar si se corre mas de una vez
//nodos 0...N-1 ; componentes 0...cantcomp-1
int comp[MAXN],N,cantcomp,used[MAXN];
stack<int> pila;
void add(int a, int b){ G[a].pb(b);gt[b].pb(a);}
void dfs1(int nodo)
{
  used[nodo]=1;
  forall(it,G[nodo]) if(!used[*it]) dfs1(*it);
  pila.push(nodo);
}
void dfs2(int nodo)
{
  used[nodo]=2;
  comp[nodo]=cantcomp-1;
  forall(it,gt[nodo]) if(used[*it]!=2) dfs2(*it);
}
void kosaraju()
{
  cantcomp=0;
  memset(used,0,sizeof(used));
  forn(i,N) if(!used[i]) dfs1(i);
  while(!pila.empty())
  {
    if(used[pila.top()]!=2)
    {
      cantcomp++;
      dfs2(pila.top());
    }
    pila.pop();
  }
}
```

8.7. 2-SAT + Tarjan SCC

```
//We have a vertex representing a var and other for his negation.
//Every edge stored in G represents an implication. To add an equation of
    the form a||b, use addor(a, b)
//MAX=max cant var, n=cant var
#define addor(a, b) (G[neg(a)].pb(b), G[neg(b)].pb(a))
vector<int> G[MAX*2];
//idx[i]=index assigned in the dfs
//lw[i]=lowest index(closer from the root) reachable from i
int lw[MAX*2], idx[MAX*2], qidx;
stack<int> q;
int qcmp, cmp[MAX*2];
//verdad[cmp[i]]=valor de la variable i
bool verdad[MAX*2+1];

int neg(int x) { return x>=n? x-n : x+n;}
void tjn(int v){
  lw[v]=idx[v]=++qidx;
  q.push(v), cmp[v]=-2;
  forall(it, G[v]){
    if(!idx[*it] || cmp[*it]==-2){
      if(!idx[*it]) tjn(*it);
      lw[v]=min(lw[v], lw[*it]);
    }
  }
  if(lw[v]==idx[v]){
    int x;
    do{x=q.top(); q.pop(); cmp[x]=qcmp;}while(x!=v);
    verdad[qcmp]=(cmp[neg(v)]<0);
    qcmp++;
  }
}
//remember to CLEAR G!!!
bool satisf(){//O(n)
  memset(idx, 0, sizeof(idx)), qidx=0;
  memset(cmp, -1, sizeof(cmp)), qcmp=0;
  forn(i, n){
    if(!idx[i]) tjn(i);
    if(!idx[neg(i)]) tjn(neg(i));
  }
  forn(i, n) if(cmp[i]==cmp[neg(i)]) return false;
```

```
  return true;
}
```

8.8. Puntos de Articulación

```
int N;
vector<int> G[1000000];
//V[i]=node number(if visited), L[i]= lowest V[i] reachable from i
int qV, V[1000000], L[1000000], P[1000000];
void dfs(int v, int f){
  L[v]=V[v]=++qV;
  forall(it, G[v])
    if(!V[*it]){
      dfs(*it, v);
      L[v] = min(L[v], L[*it]);
      P[v]+= L[*it]>=V[v];
    }
    else if(*it!=f)
      L[v]=min(L[v], V[*it]);
}
int cantart(){ //O(n)
  qV=0;
  zero(V), zero(P);
  dfs(1, 0); P[1]--;
  int q=0;
  forn(i, N) if(P[i]) q++;
return q;
}
```

8.9. Least Common Ancestor + Climb

```
const int MAXN=100001, LOGN=20;
//f[v][k] holds the 2^k father of v
//L[v] holds the level of v
int N, f[MAXN][LOGN], L[MAXN];
//call before build:
void dfs(int v, int fa=-1, int lvl=0){//generate required data
  f[v][0]=fa, L[v]=lvl;
  forall(it, G[v])if(*it!=fa) dfs(*it, v, lvl+1); }
void build(){//f[i][0] must be filled previously, O(nlgn)
  forn(k, LOGN-1) forn(i, N) f[i][k+1]=f[f[i][k]][k];}
#define lg(x) (31-__builtin_clz(x))//=floor(log2(x))
int climb(int a, int d){//O(lgn)
  if(!d) return a;
  dforn(i, lg(L[a])+1) if(1<<i<=d) a=f[a][i], d-=1<<i;
    return a;}
int lca(int a, int b){//O(lgn)
  if(L[a]<L[b]) swap(a, b);
  a=climb(a, L[a]-L[b]);
  if(a==b) return a;
  dforn(i, lg(L[a])+1) if(f[a][i]!=f[b][i]) a=f[a][i], b=f[b][i];
  return f[a][0]; }
int dist(int a, int b) {//returns distance between nodes
  return L[a]+L[b]-2*L[lca(a, b)];}
```

8.10. Heavy Light Decomposition

```
vector<int> G[MAXN];
int treesz[MAXN];//cantidad de nodos en el subarbol del nodo v
int dad[MAXN];//dad[v]=padre del nodo v
void dfs1(int v, int p=-1){//pre-dfs
  dad[v]=p;
  treesz[v]=1;
  forall(it, G[v]) if(*it!=p){
    dfs1(*it, v);
    treesz[v]+=treesz[*it];
  }
}
//PONER Q EN 0  !!!!!
int pos[MAXN], q;//pos[v]=posicion del nodo v en el recorrido de la dfs
//Las cadenas aparecen continuas en el recorrido!
int cantcad;
int homecad[MAXN];//dada una cadena devuelve su nodo inicial
int cad[MAXN];//cad[v]=cadena a la que pertenece el nodo
void heavylight(int v, int cur=-1){
  if(cur==-1) homecad[cur=cantcad++]=v;
  pos[v]=q++;
  cad[v]=cur;
  int mx=-1;
  forn(i, sz(G[v])) if(G[v][i]!=dad[v])
    if(mx==-1 || treesz[G[v][mx]]<treesz[G[v][i]]) mx=i;
  if(mx!=-1) heavylight(G[v][mx], cur);
  forn(i, sz(G[v])) if(i!=mx && G[v][i]!=dad[v])
    heavylight(G[v][i], -1);
}
//ejemplo de obtener el maximo numero en el camino entre dos nodos
//RTA: max(query(low, u), query(low, v)), con low=lca(u, v)
//esta funcion va trepando por las cadenas
int query(int an, int v){//O(logn)
  //si estan en la misma cadena:
  if(cad[an]==cad[v]) return rmq.get(pos[an], pos[v]+1);
  return max(query(an, dad[homecad[cad[v]]]),
         rmq.get(pos[homecad[cad[v]]], pos[v]+1));
}
```

8.11. Centroid Decomposition

```
vector<int> G[MAXN];
bool taken[MAXN];//poner todos en FALSE al principio!!
int padre[MAXN];//padre de cada nodo en el centroid tree

int szt[MAXN];
void calcsz(int v, int p) {
  szt[v] = 1;
  forall(it,G[v]) if (*it!=p && !taken[*it])
    calcsz(*it,v), szt[v]+=szt[*it];
}
void centroid(int v=0, int f=-1, int lvl=0, int tam=-1) {//O(nlogn)
  if(tam==-1) calcsz(v, -1), tam=szt[v];
  forall(it, G[v]) if(!taken[*it] && szt[*it]>=tam/2)
    {szt[v]=0; centroid(*it, f, lvl, tam); return;}
  taken[v]=true;
  padre[v]=f;
  forall(it, G[v]) if(!taken[*it])
    centroid(*it, v, lvl+1, -1);
}
```

8.12. Ciclo Euleriano

```
int n,m,ars[MAXE], eq;
vector<int> G[MAXN];//fill G,n,m,ars,eq
list<int> path;
int used[MAXN];
bool usede[MAXE];
queue<list<int>::iterator> q;
int get(int v){
  while(used[v]<sz(G[v]) && usede[ G[v][used[v]] ]) used[v]++;
  return used[v];
}
void explore(int v, int r, list<int>::iterator it){
  int ar=G[v][get(v)]; int u=v^ars[ar];
  usede[ar]=true;
  list<int>::iterator it2=path.insert(it, u);
  if(u!=r) explore(u, r, it2);
  if(get(v)<sz(G[v])) q.push(it);
}
void euler(){
  zero(used), zero(usede);
  path.clear();
  q=queue<list<int>::iterator>();
  path.push_back(0); q.push(path.begin());
  while(sz(q)){
    list<int>::iterator it=q.front(); q.pop();
    if(used[*it]<sz(G[*it])) explore(*it, *it, it);
  }
  reverse(path.begin(), path.end());
}
void addEdge(int u, int v){
  G[u].pb(eq), G[v].pb(eq);
  ars[eq++]=u^v;
}
```

8.13. Diametro Árbol

```
vector<int> G[MAXN]; int n,m,p[MAXN],d[MAXN],d2[MAXN];
int bfs(int r, int *d) {
  queue<int> q;
  d[r]=0; q.push(r);
  int v;
  while(sz(q)) { v=q.front(); q.pop();
    forall(it,G[v]) if (d[*it]==-1)
      d[*it]=d[v]+1, p[*it]=v, q.push(*it);
  }
  return v;//ultimo nodo visitado
}
vector<int> diams; vector<ii> centros;
void diametros(){
  memset(d,-1,sizeof(d));
  memset(d2,-1,sizeof(d2));
  diams.clear(), centros.clear();
  forn(i, n) if(d[i]==-1){
    int v,c;
    c=v=bfs(bfs(i, d2), d);
    forn(_,d[v]/2) c=p[c];
    diams.pb(d[v]);
    if(d[v]&1) centros.pb(ii(c, p[c]));
    else centros.pb(ii(c, c));
  }
}
```

8.14. Componentes Biconexas y Puentes

```
vector<int> G[MAXN];

struct edge{
  int u,v, comp;
  bool bridge;
};
vector<edge> e;
void addEdge(int u, int v)
{
  G[u].pb(sz(e)), G[v].pb(sz(e));
  e.pb((edge){u,v,-1,false});
}
//d[i]=id de la dfs
//b[i]=lowest id reachable from i
int d[MAXN], b[MAXN], t;
int nbc;//cant componentes
int comp[MAXN];//comp[i]=cant comp biconexas a la cual pertenece i
void initDfs(int n)
{
  zero(G), zero(comp);
  e.clear();
  forn(i,n) d[i]=-1;
  nbc = t = 0;
}
stack<int> st;
void dfs(int u,int pe) //O(n + m)
{
  b[u]=d[u]=t++;
  comp[u]=(pe!=-1);
  forall(ne,G[u]) if(*ne!=pe)
  {
    int v=e[*ne].u ^ e[*ne].v ^ u;
    if(d[v]==-1)
    {
      st.push(*ne);
      dfs(v,*ne);
      if(b[v]>d[u]) e[*ne].bridge=true; // bridge
      if(b[v]>=d[u])  // art
      {
        int last;
        do
        {
          las=st.top(); st.pop();
          e[last].comp=nbc;
        }while(last!=*ne);
        nbc++;
        comp[u]++;
      }
      b[u]=min(b[u],b[v]);
    }
    else if(d[v]<d[u])  // back edge
    {
      st.push(*ne);
      b[u]=min(b[u], d[v]);
    }
  }
}
```

8.15. Hungarian

```
//Dado un grafo bipartito completo con costos no negativos, encuentra el
    matching perfecto de minimo costo.
#define tipo double
tipo cost[N][N], lx[N], ly[N], slack[N]; //llenar: cost=matriz de adyacencia
int n, max_match, xy[N], yx[N], slackx[N],prev2[N];//n=cantidad de nodos
bool S[N], T[N]; //sets S and T in algorithm
void add_to_tree(int x, int prevx) {
  S[x] = true, prev2[x] = prevx;
  forn(y, n) if (lx[x] + ly[y] - cost[x][y] < slack[y] - EPS)
    slack[y] = lx[x] + ly[y] - cost[x][y], slackx[y] = x;
}
```

```
void update_labels(){
  tipo delta = INF;
  forn (y, n) if (!T[y]) delta = min(delta, slack[y]);
  forn (x, n) if (S[x]) lx[x] -= delta;
  forn (y, n) if (T[y]) ly[y] += delta; else slack[y] -= delta;
}
void init_labels(){
  zero(lx), zero(ly);
  forn (x,n) forn(y,n) lx[x] = max(lx[x], cost[x][y]);
}
void augment() {
  if (max_match == n) return;
  int x, y, root, q[N], wr = 0, rd = 0;
  memset(S, false, sizeof(S)), memset(T, false, sizeof(T));
  memset(prev2, -1, sizeof(prev2));
  forn (x, n) if (xy[x] == -1){
    q[wr++] = root = x, prev2[x] = -2;
    S[x] = true; break; }
  forn (y, n) slack[y] = lx[root] + ly[y] - cost[root][y], slackx[y] = root;
  while (true){
    while (rd < wr){
      x = q[rd++];
      for (y = 0; y < n; y++) if (cost[x][y] == lx[x] + ly[y] && !T[y]){
        if (yx[y] == -1) break; T[y] = true;
        q[wr++] = yx[y], add_to_tree(yx[y], x); }
      if (y < n) break; }
    if (y < n) break;
    update_labels(), wr = rd = 0;
    for (y = 0; y < n; y++) if (!T[y] && slack[y] == 0){
      if (yx[y] == -1){x = slackx[y]; break;}
      else{
        T[y] = true;
        if (!S[yx[y]]) q[wr++] = yx[y], add_to_tree(yx[y], slackx[y]);
      }}
    if (y < n) break; }
  if (y < n){
    max_match++;
    for (int cx = x, cy = y, ty; cx != -2; cx = prev2[cx], cy = ty)
      ty = xy[cx], yx[cy] = cx, xy[cx] = cy;
    augment(); }
}
tipo hungarian(){
  tipo ret = 0; max_match = 0, memset(xy, -1, sizeof(xy));
  memset(yx, -1, sizeof(yx)), init_labels(), augment(); //steps 1-3
  forn (x,n) ret += cost[x][xy[x]]; return ret;
}
```

8.16. Dynamic Connectivity

```
struct UnionFind {
  int n, comp;
  vector<int> pre,si,c;
  UnionFind(int n=0):n(n), comp(n), pre(n), si(n, 1) {
    forn(i,n) pre[i] = i; }
  int find(int u){return u==pre[u]?u:find(pre[u]);}
  bool merge(int u, int v)
  {
    if((u=find(u))==(v=find(v))) return false;
    if(si[u]<si[v]) swap(u, v);
    si[u]+=si[v], pre[v]=u, comp--, c.pb(v);
    return true;
  }
  int snap(){return sz(c);}
  void rollback(int snap)
  {
    while(sz(c)>snap)
    {
      int v = c.back(); c.pop_back();
      si[pre[v]] -= si[v], pre[v] = v, comp++;
    }
  }
};
enum {ADD,DEL,QUERY};
struct Query {int type,u,v;};
struct DynCon{//bidirectional graphs; create vble as DynCon name(cant_nodos)
  vector<Query> q;
  UnionFind dsu;
  vector<int> match,res;
  map<ii,int> last;//se puede no usar cuando hay identificador para cada
      arista (mejora poco)
  DynCon(int n=0):dsu(n){}
  void add(int u, int v) //to add an edge
  {
    if(u>v) swap(u,v);
    q.pb((Query){ADD, u, v}), match.pb(-1);
    last[ii(u,v)] = sz(q)-1;
  }
  void remove(int u, int v) //to remove an edge
  {
    if(u>v) swap(u,v);
    q.pb((Query){DEL, u, v});
    int prev = last[ii(u,v)];
    match[prev] = sz(q)-1;
    match.pb(prev);
  }
  void query() //to add a question (query) type of query
  {
    q.pb((Query){QUERY, -1, -1}), match.pb(-1);
  }
  void process() //call this to process queries in the order of q
```

```
  {
    forn(i,sz(q)) if (q[i].type == ADD && match[i] == -1) match[i] = sz(q);
    go(0,sz(q));
  }
  void go(int l, int r)
  {
    if(l+1==r)
    {
      if (q[l].type == QUERY)//Aqui responder la query usando el dsu!
        res.pb(dsu.comp);//aqui query=cantidad de componentes conexas
      return;
    }
    int s=dsu.snap(), m = (l+r) / 2;
    forr(i,m,r) if(match[i]!=-1 && match[i]<l) dsu.merge(q[i].u, q[i].v);
    go(l,m);
    dsu.rollback(s);
    s = dsu.snap();
    forr(i,l,m) if(match[i]!=-1 && match[i]>=r) dsu.merge(q[i].u, q[i].v);
    go(m,r);
    dsu.rollback(s);
  }
};
```

9. Flow

9.1. Edmond Karp

```
#define MAX_V 1000
#define INF 1e9
//special nodes
#define SRC 0
#define SNK 1
map<int, int> G[MAX_V];//limpiar esto -- unordered_map mejora
//To add an edge use
#define add(a, b, w) G[a][b]=w
int f, p[MAX_V];
void augment(int v, int minE)
{
  if(v==SRC) f=minE;
  else if(p[v]!=-1)
  {
    augment(p[v], min(minE, G[p[v]][v]));
    G[p[v]][v]-=f, G[v][p[v]]+=f;
  }
}
ll maxflow()//O(min(VE^2,Mf*E))
{
  ll Mf=0;
  do
  {
    f=0;
    char used[MAX_V]; queue<int> q; q.push(SRC);
    zero(used), memset(p, -1, sizeof(p));
    while(sz(q))
    {
      int u=q.front(); q.pop();
      if(u==SNK) break;
      forall(it, G[u])
        if(it->snd>0 && !used[it->fst])
        used[it->fst]=true, q.push(it->fst), p[it->fst]=u;
    }
    augment(SNK, INF);
    Mf+=f;
  }while(f);
  return Mf;
}
```

9.2. Min Cut

```
//Suponemos un grafo con el formato definido en Edmond Karp o Push relabel
bitset<MAX_V> type,used; //reset this
void dfs1(int nodo)
{
  type.set(nodo);
  forall(it,G[nodo]) if(!type[it->fst] && it->snd>0) dfs1(it->fst);
}
void dfs2(int nodo)
{
  used.set(nodo);
  forall(it,G[nodo])
  {
    if(!type[it->fst])
    {
      //edge nodo -> (it->fst) pertenece al min_cut
      //y su peso original era: it->snd + G[it->fst][nodo]
      //si no existia arista original al reves
    }
    else if(!used[it->fst]) dfs2(it->fst);
  }
}
void minCut() //antes correr algun maxflow()
{
  dfs1(SRC);
  dfs2(SRC);
  return;
}
```

9.3. Push Relabel

```
#define MAX_V 1000
int N;//valid nodes are [0...N-1]
#define INF 1e9
//special nodes
#define SRC 0
#define SNK 1
map<int, int> G[MAX_V];//limpiar esto -- unordered_map mejora
//To add an edge use
#define add(a, b, w) G[a][b]=w
ll excess[MAX_V];
int height[MAX_V], active[MAX_V], cuenta[2*MAX_V+1];
queue<int> Q;

void enqueue(int v)
{
  if (!active[v] && excess[v] > 0) active[v]=true, Q.push(v);
}
void push(int a, int b)
{
  int amt = min(excess[a], ll(G[a][b]));
  if(height[a] <= height[b] || amt == 0) return;
  G[a][b]-=amt, G[b][a]+=amt;
  excess[b] += amt, excess[a] -= amt;
  enqueue(b);
}
void gap(int k)
{
  forn(v, N)
  {
    if (height[v] < k) continue;
    cuenta[height[v]]--;
    height[v] = max(height[v], N+1);
    cuenta[height[v]]++;
    enqueue(v);
  }
}
void relabel(int v)
{
  cuenta[height[v]]--;
  height[v] = 2*N;
  forall(it, G[v])
  if(it->snd) height[v] = min(height[v], height[it->fst] + 1);
  cuenta[height[v]]++;
  enqueue(v);
}
ll maxflow() //O(V^3)
{
  zero(height), zero(active), zero(cuenta), zero(excess);
  cuenta[0]=N-1; cuenta[N]=1;
  height[SRC] = N;
  active[SRC] = active[SNK] = true;
  forall(it, G[SRC])
  {
    excess[SRC] += it->snd;
    push(SRC, it->fst);
  }
  while(sz(Q))
  {
    int v = Q.front(); Q.pop();
    active[v]=false;
    forall(it, G[v]) push(v, it->fst);
    if(excess[v] > 0)
    cuenta[height[v]] == 1? gap(height[v]):relabel(v);
  }
  ll mf=0;
  forall(it, G[SRC]) mf+=G[it->fst][SRC];
  return mf;
}
```

9.4. Dinic

```
struct Edge {
  int u, v;
  ll cap, flow;
  Edge() {}
  Edge(int u, int v, ll cap): u(u), v(v), cap(cap), flow(0) {}
};
struct Dinic {
  int N;
  vector<Edge> E;
  vector<vector<int>> g;
  vector<int> d, pt;
  Dinic(int N): N(N), E(0), g(N), d(N), pt(N) {} //clear and init
  void addEdge(int u, int v, ll cap)
  {
    if (u != v)
    {
      E.emplace_back(Edge(u, v, cap));
      g[u].emplace_back(E.size() - 1);
      E.emplace_back(Edge(v, u, 0));
      g[v].emplace_back(E.size() - 1);
    }
  }
  bool BFS(int S, int T)
  {
    queue<int> q({S});
    fill(d.begin(), d.end(), N + 1);
    d[S] = 0;
    while(!q.empty())
    {
      int u = q.front(); q.pop();
      if (u == T) break;
      for (int k: g[u])
      {
        Edge &e = E[k];
        if (e.flow < e.cap && d[e.v] > d[e.u] + 1)
        {
          d[e.v] = d[e.u] + 1;
          q.emplace(e.v);
        }
      }
    }
    return d[T] != N + 1;
  }
  ll DFS(int u, int T, ll flow = -1)
  {
    if (u == T || flow == 0) return flow;
    for (int &i = pt[u]; i < g[u].size(); ++i)
    {
      Edge &e = E[g[u][i]];
      Edge &oe = E[g[u][i]^1];
      if (d[e.v] == d[e.u] + 1)
      {
        ll amt = e.cap - e.flow;
        if (flow != -1 && amt > flow) amt = flow;
        if (ll pushed = DFS(e.v, T, amt))
        {
          e.flow += pushed;
          oe.flow -= pushed;
          return pushed;
        }
      }
    }
    return 0;
  }
  ll maxFlow(int S,int T)
  {
    ll total = 0;
    while(BFS(S, T))
    {
      fill(pt.begin(), pt.end(), 0);
      while (ll flow = DFS(S, T)) total += flow;
    }
    return total;
  }
};
```

9.5. Min cost - Max flow

```
const int MAXN=10000;
typedef ll tf;
typedef ll tc;
const tf INFFLUJO = 1e14;
const tc INFCOSTO = 1e14;
struct edge {
  int u, v;
  tf cap, flow;
  tc cost;
  tf rem() { return cap - flow; }
};
int nodes; //numero de nodos
vector<int> G[MAXN]; // limpiar!
vector<edge> e;  // limpiar!
void addEdge(int u, int v, tf cap, tc cost)
{
  G[u].pb(sz(e)); e.pb((edge){u,v,cap,0,cost});
  G[v].pb(sz(e)); e.pb((edge){v,u,0,0,-cost});
}
tc dist[MAXN], mnCost;
int pre[MAXN];
tf cap[MAXN], mxFlow;
bool in_queue[MAXN];
void flow(int s, int t)
```

```
{
  zero(in_queue);
  mxFlow=mnCost=0;
  while(1)
  {
    fill(dist, dist+nodes, INFCOSTO); dist[s] = 0;
    memset(pre, -1, sizeof(pre)); pre[s]=0;
    zero(cap); cap[s] = INFFLUJO;
    queue<int> q; q.push(s); in_queue[s]=1;
    while(sz(q))
    {
      int u=q.front(); q.pop(); in_queue[u]=0;
      for(auto it:G[u])
      {
        edge &E = e[it];
        if(E.rem() && dist[E.v] > dist[u] + E.cost + 1e-9) // ojo EPS
        {
          dist[E.v]=dist[u]+E.cost;
          pre[E.v] = it;
          cap[E.v] = min(cap[u], E.rem());
          if(!in_queue[E.v]) q.push(E.v), in_queue[E.v]=1;
        }
      }
    }
    if (pre[t] == -1) break;
    mxFlow +=cap[t];
    mnCost +=cap[t]*dist[t];
    for (int v = t; v != s; v = e[pre[v]].u)
    {
      e[pre[v]].flow += cap[t];
      e[pre[v]^1].flow -= cap[t];
    }
  }
}
```

10. Juegos

10.1. Nim Game

Juego en el que hay N pilas, con objetos. Cada jugador debe sacar al menos un objeto de una pila. GANA el jugador que saca el último objeto.

$$P_0 \oplus P_1 \oplus ... \oplus P_n = R$$

Si $R \neq 0$ gana el jugador 1.

10.1.1. Misere Game

Es un juego con las mismas reglas que Nim, pero PIERDE el que saca el último objeto. Entonces teniendo el resultado de la suma R, y si todas las pilas tienen 1 solo objeto $todos1 = true$, podemos decir que el jugador2 GANA si:

$$(R=0) \& \neg todos1 \| (R \neq 0) \& todos1$$

10.2. Ajedrez

10.2.1. Non-Attacking N Queen

Utiliza: `<algorithm>`

Notas: todo es $O(!N \cdot N^2)$.

```
#define NQUEEN 8
#define abs(x) ((x)<0?(-(x)):(x))

int board[NQUEEN];
void inline init(){for(int i=0;i<NQUEEN;++i)board[i]=i;}
bool check(){
    for(int i=0;i<NQUEEN;++i)
        for(int j=i+1;i<NQUEEN;++j)
            if(abs(i-j)==abs(board[i]-board[j]))
                return false;
    return true;
}
//en main
init();
do{
    if(check()){
        //process solution
    }
}while(next_permutation(board,board+NQUEEN));
```

11. Utils

11.1. Convertir string a num e viceversa

```
#include <sstream>
string num_to_str(int x){
  ostringstream convert;
  convert << x;
  return convert.str();
}

int str_to_num(string x){
  int ret;
  istringstream (x) >> ret;
  return ret;
}
```

11.2. Truquitos para entradas/salidas

```
//Cantidad de decimales
cout << setprecision(2) << fixed;
//Rellenar con espacios(para justificar)
cout << setfill(' ') << setw(3) << 2 << endl;
//Leer hasta fin de linea
//  hacer cin.ignore() antes de getline()
while(getline(cin, line)){
  istringstream is(line);
  while(is >> X)
    cout << X << " ";
  cout << endl;
}
```

11.3. Comparación de Double

```
const double EPS = 1e-9;
x == y <=> fabs(x-y) < EPS
x > y <=> x > y + EPS
x >= y <=> x > y - EPS
```

11.4. Iterar subconjuntos

```
for(int sbm=bm; sbm; sbm=(sbm-1)&bm)
```

11.5. Limites

```
#include <limits>
numeric_limits<T>
  ::max()
  ::min()
  ::epsilon()
```

11.6. Mejorar Lectura de Enteros

```
//Solo para enteros positivos
inline void Scanf(int& a)
{
  char c = 0;
  while(c<33) c = getc(stdin);
  a = 0;
  while(c>33) a = a*10 + c - '0', c = getc(stdin);
}
```

11.7. Tablita de relacion de Complejidades

n	Peor AC Complejidad	Comentario
$\leq$ [10.,11]	$O(n!), O(n^6)$	ej. Enumerar permutaciones
$\leq$ [15.,18]	$O(2^n \times n^2)$	ej. DP TSP
$\leq$ [18.,22]	$O(2^n \times n)$	ej. DP con mascara de bits
≤ 100	$O(n^4)$	ej. DP con 3 dimensiones $+O(n)$ loops
≤ 400	$O(n^3)$	ej. Floyd Warshall
$\leq 2K$	$n^2 \log_2 n$	ej. 2 loops anidados + una busqueda en arbol en una estructura de datos
$\leq 10K$	$O(n^2)$	ej. Ordenamiento Burbuja/Selección/Inserción
$\leq 1M$	$O(n \log_2 n)$	ej. Merge Sort, armar Segment Tree
$\leq 100M$	$O(n), O(\log_2 n), O(1)$	La mayoría de los problemas de contest tiene $n \leq 1M$ (cuello de botella en I/O)

11.8. Compilar C++11 con g++

Dos opciones, útil en Linux.

```
g++ -std=c++11 {file} -o {filename}

g++ -std=c++0x {file} -o {filename}
```

11.9. Build de C++11 para Sublime Text

```
{
  "shell_cmd": "g++ -std=c++0x \"${file}\" -o \"${file_path}/${
      file_base_name}\"",
  "file_regex": "^(..[^:]*):([0-9]+):?([0-9]+)?:? (.*)$",
  "working_dir": "${file_path}",
  "selector": "source.cpp, source.c++",
  "variants":
  [{
    "name": "Run",
    "shell_cmd": "g++ -std=c++0x \"${file}\" -o \"${file_path}/${
        file_base_name}\" && \"${file_path}/${file_base_name}\""
}]}
```

11.10. Funciones Utiles

Algo	Params	Función
`fill, fill_n`	`f, l / n, elem`	*void* llena [f, l) o [f,f+n) con elem
`lower_bound, upper_bound`	`f, l, elem`	*it* al primer ultimo donde se puede insertar elem para que quede ordenada
`copy`	`f, l, resul`	hace resul+i=f+i $\forall i$
`find, find_if, find_first_of`	`f, l, elem` / pred / f2, l2	*it* encuentra i $\in$[f,l) tq. i=elem, pred(i), i$\in$[f2,l2)
`count, count_if`	`f, l, elem/pred`	cuenta elem, pred(i)
`search`	`f, l, f2, l2`	busca [f2,l2) $\in$ [f,l)
`replace, replace_if`	`f, l, old` / pred, new	cambia old / pred(i) por new
`lexicographical_compare`	`f1,l1,f2,l2`	*bool* con [f1,l1]¡[f2,l2]
`accumulate`	`f,l,i,[op]`	$T = \sum$/oper de [f,l)
`inner_product`	`f1, l1, f2, i`	T = i + [f1, l1) . [f2, ...)
`partial_sum`	`f, l, r, [op]`	r+i = $\sum$/oper de [f,f+i] $\forall i$ $\in$[f,l)
`__builtin_ffs`	`unsigned int`	Pos. del primer 1 desde la derecha
`__builtin_clz`	`unsigned int`	Cant. de ceros desde la izquierda.
`__builtin_ctz`	`unsigned int`	Cant. de ceros desde la derecha.
`__builtin_popcount`	`unsigned int`	Cant. de 1's en x.
`__builtin_parity`	`unsigned int`	1 si x es par, 0 si es impar.
`__builtin_XXXXXXll`	`unsigned ll`	= pero para long long's.