El Mufoso

UTN FRSF - anarap+1

2022

Contents

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*

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

length	**f e g a**	**c s [ ] [ˆ]**	**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 representacíion más corta: %e ó %f	392.65
G	Utilizar la representacíion 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

length	d i	u o x X
		Continuación
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

```cpp
#include <bits/stdc++.h>
#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 sz(c) ((int)c.size())
#define rsz resize
#define pb push_back
#define mp make_pair
#define lb lower_bound
#define ub upper_bound
#define fst first
#define snd second

#ifdef ANARAP
//local
#else
//judge
#endif

using namespace std;

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

int main()
{
  // agregar g++ -DANARAP en compilacion
  #ifdef ANARAP
    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 Ordered set

Esto solo compila desde C++11.

```cpp
#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

3.2.1 Classic DSU

```cpp
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;
  }
};
```

3.2.2 DSU with rollbacks

```cpp
struct dsu_save {
    int v, rnkv, u, rnku;
    dsu_save() {}
    dsu_save(int _v, int _rnkv, int _u, int _rnku)
        : v(_v), rnkv(_rnkv), u(_u), rnku(_rnku) {}
};
struct dsu_with_rollbacks {
    vector<int> p, rnk;
    int comps;
    stack<dsu_save> op;
    dsu_with_rollbacks() {}
    dsu_with_rollbacks(int n) {
        p.rsz(n);rnk.rsz(n);
        forn(i,n) {p[i] = i; rnk[i] = 0;}
        comps = n;
    }
    int find_set(int v) {return (v == p[v]) ? v : find_set(p[v]);}
    bool unite(int v, int u) {
        v = find_set(v); u = find_set(u);
        if (v == u) return false;
        comps--;
        if (rnk[v] > rnk[u]) swap(v, u);
        op.push(dsu_save(v, rnk[v], u, rnk[u]));
        p[v] = u;
        if (rnk[u] == rnk[v]) rnk[u]++;
        return true;
    }
    void rollback() {
        if (op.empty()) return;
        dsu_save x = op.top();
        op.pop(); comps++;
        p[x.v] = x.v; rnk[x.v] = x.rnkv;
        p[x.u] = x.u; rnk[x.u] = x.rnku;
    }
};
```

3.3　Hash Table

```cpp
//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

```cpp
//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

```cpp
// 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().

```cpp
//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

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

```
 4  const tipo neutro=0;
 5  struct RMQ {
 6    int sz;
 7    tipo t[4*MAXN];
 8    tipo &operator[](int p) {return t[sz+p];}
 9    void init(int n) {//O(nlgn)
10      sz = 1 << (32 - __builtin_clz(n));
11      forn(i, 2*sz) t[i] = neutro;
12    }
13    void updall() {dforn(i, sz) t[i]=operacion(t[2*i], t[2*i+1]);} //O(n)
14    tipo get(int i, int j) {return get(i,j,1,0,sz);}
15    tipo get(int i, int j, int n, int a, int b) {//O(lgn), [i, j)
16      if(j<=a || i>=b) return neutro;
17      if(i<=a && b<=j) return t[n];//n = node of range [a,b)
18      int c = (a+b)/2;
19      return operacion(get(i, j, 2*n, a, c), get(i, j, 2*n+1, c, b));
20    }
21    void set(int p, tipo val) {//O(lgn)
22      p += sz;
23      while(p>0 && t[p]!=val) {
24        t[p] = val;
25        p /= 2;
26        val = operacion(t[p*2], t[p*2+1]);
27      }
28    }
29  }; //Use: definir operacion tipo neutro y MAXN,
30  //cin >> n; rmq.init(n); forn(i, n) cin >> rmq[i]; rmq.updall();
```

```
22        }
23        dirty[n] = neutro2;
24      }
25    }
26    Elem get(int i, int j, int n, int a, int b) {//O(lgn)
27      if(j<=a || i>=b) return neutro;
28      push(n, a, b);//corrige el valor antes de usarlo
29      if(i<=a && b<=j) return t[n];//n = node of range [a,b)
30      int c = (a+b)/2;
31      return operacion(get(i, j, 2*n, a, c), get(i, j, 2*n+1, c, b));
32    }
33    Elem get(int i, int j) {return get(i,j,1,0,sz);}
34    //altera los valores en [i, j) con una alteracion de val
35    void alterar(Alt val, int i, int j, int n, int a, int b) {//O(lgn)
36      push(n, a, b);
37      if(j<=a || i>=b) return;
38      if(i<=a && b<=j){
39        dirty[n] += val; // modificar segun el problema
40        push(n, a, b);
41        return;
42      }
43      int c = (a+b)/2;
44      alterar(val, i, j, 2*n, a, c), alterar(val, i, j, 2*n+1, c, b);
45      t[n] = operacion(t[2*n], t[2*n+1]);//por esto es el push de arriba
46    }
47    void alterar(Alt val, int i, int j){alterar(val,i,j,1,0,sz);}
48  };//Use: definir operacion, neutros, Alt, Elem, uso de dirty, MAXN
49  //cin >> n; rmq.init(n); forn(i,n) cin >> rmq[i]; rmq.updall()
```

3.6.3   ST lazy

```
 1  //Dado un arreglo y una operacion asociativa con neutro
 2  #define operacion(x,y) ((x)+(y))
 3  const Elem neutro=0; const Alt neutro2=0;
 4  #define MAXN 100010//Cambiar segun el N del problema
 5  struct RMQ{
 6    int sz;
 7    Elem t[4*MAXN];
 8    Alt dirty[4*MAXN];//las alteraciones pueden ser de tipo distinto a Elem
 9    Elem &operator[](int p) {return t[sz+p];}
10    void init(int n){//O(nlgn)
11      sz = 1 << (32-__builtin_clz(n));
12      forn(i, 2*sz) t[i] = neutro;
13      forn(i, 2*sz) dirty[i] = neutro2;
14    }
15    void updall() {dforn(i, sz) t[i]=operacion(t[2*i], t[2*i+1]);} //O(n)
16    void push(int n, int a, int b) {//propaga el dirty a sus hijos
17      if(dirty[n] != neutro2) {// n = node of range [a,b)
18        t[n] += dirty[n]*(b-a);//altera el nodo, modificar segun el problema
19        if(n<sz){
20          dirty[2*n] += dirty[n];
21          dirty[2*n+1] += dirty[n];
```

3.6.4   ST persistente

```
 1  typedef int tipo;
 2  tipo oper(const tipo &a, const tipo &b){
 3      return a+b;
 4  }
 5  struct node{
 6    tipo v; node *l, *r;
 7    node(tipo v) : v(v), l(NULL), r(NULL) {}
 8    node(node *l, node *r) : l(l), r(r) {
 9        if(!l) v = r->v;
10        else if(!r) v = l->v;
11        else v = oper(l->v, r->v);
12    }
13  };
14  node *build(tipo *a, int tl, int tr) {//build a partir de un arreglo
15    if(tl+1 == tr) return new node(a[tl]);
16    int tm = (tl+tr)>>1;
17    return new node(build(a, tl, tm), build(a, tm, tr));
18  }
19  node *update(int pos, int new_val, node *t, int tl, int tr) {
20    if(tl+1 == tr) return new node(new_val);
```

```
21    int tm = (tl+tr)>>1;
22    if(pos < tm) return new node(update(pos, new_val, t->l, tl, tm), t->r);
23    else return new node(t->l, update(pos, new_val, t->r, tm, tr));
24  }
25  tipo get(int l, int r, node *t, int tl, int tr) {
26      if(l==tl && tr==r) return t->v;
27    int tm = (tl+tr)>>1;
28      if(r <= tm) return get(l, r, t->l, tl, tm);
29      else if(l >= tm) return get(l, r, t->r, tm, tr);
30    return oper(get(l, tm, t->l, tl, tm), get(tm, r, t->r, tm, tr));
31  }
32  //node t represents range [tl, tr). For roots always use tl = 0, tr = size
```

3.6.5   ST 2d

```
1   #define operacion(x, y) max(x, y)
2   int n, m;
3   int a[MAXN][MAXN], st[2*MAXN][2*MAXN];
4   void build() { //O(n*m)
5     forn(i, n) forn(j, m) st[i+n][j+m] = a[i][j];
6     forn(i, n) dforn(j, m) //build st of row i+n (each row independently)
7       st[i+n][j] = operacion(st[i+n][j<<1], st[i+n][j<<1|1]);
8     dforn(i, n) forn(j, 2*m) //build st of ranges of rows
9       st[i][j] = operacion(st[i<<1][j], st[i<<1|1][j]);
10  }
11  void upd(int x, int y, int v) { //O(logn * logm)
12    st[x+n][y+m] = v;
13    for(int j=y+m; j>1; j>>=1)//update the ranges that contains y+m in row x+n
14      st[x+n][j>>1] = operacion(st[x+n][j], st[x+n][j^1]);
15    for(int i=x+n; i>1; i>>=1)//in each range that contains row x+n
16      for(int j=y+m; j; j>>=1) //update the ranges that contains y+m
17        st[i>>1][j] = operacion(st[i][j], st[i^1][j]);
18  }
19  int query(int x0, int x1, int y0, int y1) { //O(logn * logm)
20    int r = NEUT;
21    //start at the bottom and move up each time
22    for(int i0=x0+n, i1=x1+n; i0<i1; i0>>=1, i1>>=1) {
23      int t[4], q=0;
24      //if the whole segment of row node i0 is included, then move right
25      if(i0&1) t[q++] = i0++;
26      //if the whole segment of row node i1-1 is included, then move left
27      if(i1&1) t[q++] = --i1;
28      forn(k, q) for(int j0=y0+m, j1=y1+m; j0<j1; j0>>=1, j1>>=1) {
29        if(j0&1) r = operacion(r, st[t[k]][j0++]);
30        if(j1&1) r = operacion(r, st[t[k]][--j1]);
31      }
32    }
33    return r;
34  }
```

3.7   Treap

```
1   typedef struct item *pitem;
2   struct item {
3       //pr = randomized priority, key = BST value, cnt = size of subtree
4       int pr, key, cnt;
5       pitem l, r;
6       item(int key) : key(key), pr(rand()), cnt(1), l(NULL), r(NULL) {}
7   };
8   int cnt(pitem node) {return node ? node->cnt : 0;}
9   void upd_cnt(pitem node) {if(node) node->cnt = cnt(node->l)+cnt(node->r)+1;}
10  //splits t in l and r - l: <= key, r: > key
11  void split(pitem node, int key, pitem& L, pitem& R){ // O(log)
12      if(!node) L = R = 0;
13      // if cur > key, go left to split and cur is part of R
14      else if(key < node->key) split(node->l, key, L, node->l), R = node;
15      // if cur <= key, go right to split and cur is part of L
16      else split(node->r, key, node->r, R), L = node;
17      upd_cnt(node);
18  }
19  //1) go down the BST following the key of the new node (x), until
20  // you reach NULL or a node with lower pr than the new one.
21  //2.1) if you reach NULL, put the new node there
22  //2.2) if you reach a node with lower pr, split the subtree rooted at that
23  //node, put the new one there and put the split result as children of it
24  void insert(pitem& node, pitem x) { // O(log)
25      if(!node) node = x;
26      else if(x->pr > node->pr) split(node, x->key, x->l, x->r), node = x;
27      else insert(x->key <= node->key ? node->l : node->r, x);
28      upd_cnt(node);
29  }
30  //Assumes that the key of every element in L <= to the keys in R
31  void merge(pitem& result, pitem L, pitem R) { // O(log)
32      //If one of the nodes is NULL, the merge result is the other node
33      if(!L || !R) result = L ? L : R;
34      //if L has higher priority than R, put L and update it's right child
35      //with the merge result of L->r and R
36      else if(L->pr > R->pr) merge(L->r, L->r, R), result = L;
37      //if R has higher priority than L, put R and update it's left child
38      //with the merge result of L and R->l
39      else merge(R->l, L, R->l), result = R;
40      upd_cnt(result);
41  }
42  //go down the BST following the key to erase. When the key is found,
43  //replace that node with the result of merging it children
44  void erase(pitem& node, int key) {// O(log), (erases only 1 repetition)
45      if(node->key == key) merge(node, node->l, node->r);
46      else erase(key < node->key ? node->l : node->r, key);
47      upd_cnt(node);
48  }
49  //union of two treaps
50  void unite(pitem &t, pitem L, pitem R) { // O(M*log(N/M))
51      if(!L || !R) {t = L ? L : R; return;}
```

```
52      if(L->pr < R->pr) swap(L, R);
53      pitem p1, p2; split(R, L->key, p1, p2);
54      unite(L->l, L->l, p1); unite(L->r, L->r, p2);
55      t = L; upd_cnt(t);
56  }
57  pitem kth(pitem t, int k) { // element at "position" k
58      if(!t) return 0;
59      if(k == cnt(t->l)) return t;
60      return k < cnt(t->l) ? kth(t->l, k) : kth(t->r, k - cnt(t->l) - 1);
61  }
62  pair<int,int> lb(pitem t, int key) { // position and value of lower_bound
63      if(!t) return {0,1<<30}; // (special value)
64      if(key > t->key){
65          auto w = lb(t->r,key); w.fst += cnt(t->l)+1; return w;
66      }
67      auto w = lb(t->l,key);
68      if(w.fst == cnt(t->l)) w.snd = t->key;
69      return w;
70  }
```

3.8   Implicit treap

```
1   // An array represented as a treap, where the "key" is the index.
2   // However, the key is not stored explicitly, but can be calculated as
3   // the sum of the sizes of the left child of the ancestors where the node
4   // is in the right subtree of it.
5   // (commented parts are specific to range sum queries and other problems)
6   // rng = random number generator, works better than rand in some cases
7   mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
8   typedef struct item *pitem;
9   struct item {
10    int pr, cnt, val;
11    bool rev;
12  //  int sum; // (paramters for range query)
13  //  int add; // (parameters for lazy prop)
14    pitem l, r;
15  //  pitem p; // ptr to parent, for getPos
16    item(int val) : pr(rng()), cnt(1), val(val), rev(false),/* sum(val),
17           add(0),*/ l(NULL), r(NULL), /*p(NULL)*/{}
18  };
19  void push(pitem node) {
20    if(node){
21      if(node->rev) {
22        swap(node->l, node->r);
23        if(node->l) node->l->rev ^= true;
24        if(node->r) node->r->rev ^= true;
25        node->rev = false;
26      }
27      /*node->val += node->add; node->sum += node->cnt * node->add;
28      if(node->l) node->l->add += node->add;
29      if(node->r) node->r->add += node->add;
30      node->add = 0;*/
31    }
32  }
33  int cnt(pitem t) {return t ? t->cnt : 0;}
34  // int sum(pitem t) {return t ? push(t), t->sum : 0;}
35  void upd_cnt(pitem t) {
36    if(t) {
37      t->cnt = cnt(t->l) + cnt(t->r) + 1;
38      //t->sum=t->val+sum(t->l)+sum(t->r); // for range sum
39      /*if(t->l) t->l->p = t; // for getPos
40      if(t->r) t->r->p = t;
41      t->p = NULL;*/
42    }
43  }
44  void split(pitem node, pitem& L, pitem& R, int sz) {// sz: wanted size for L
45    if(!node) {L = R = 0; return;}              // O(log)
46    push(node);
47    //If node's left child has at least sz nodes, go left
48    if(sz <= cnt(node->l)) split(node->l, L, node->l, sz), R = node;
49    //Else, go right changing wanted sz
50    else split(node->r, node->r, R, sz-1-cnt(node->l)), L = node;
51    upd_cnt(node);
52  }
53  void merge(pitem& result, pitem L, pitem R) {// O(log)
54    push(L); push(R);
55    if(!L || !R) result = L ? L : R;
56    else if(L->pr > R->pr) merge(L->r, L->r, R), result = L;
57    else merge(R->l, L, R->l), result = R;
58    upd_cnt(result);
59  }
60  void insert(pitem& node, pitem x, int pos) {// 0-index O(log)
61    pitem l,r;
62    split(node, l, r, pos);
63    merge(l, l, x);
64    merge(node, l, r);
65  }
66  void erase(pitem& node, int pos) {// 0-index O(log)
67    if(!node) return;
68    push(node);
69    if(pos == cnt(node->l)) merge(node, node->l, node->r);
70    else if(pos < cnt(node->l)) erase(node->l, pos);
71    else erase(node->r, pos-1-cnt(node->l));
72    upd_cnt(node);
73  }
74  void reverse(pitem &node, int L, int R) {//[L, R] O(log)
75    pitem t1, t2, t3;
76    split(node, t1, t2, L);
77    split(t2, t2, t3, R-L);
78    t2->rev ^= true;
79    merge(node, t1, t2);
80    merge(node, node, t3);
81  }
82  /*void add(pitem &node, int L, int R, int x) {//[L, R) O(log)
```

```
 83    pitem t1, t2, t3;
 84    split(node, t1, t2, L);
 85    split(t2, t2, t3, R-L);
 86    t2->add += x;
 87    merge(node, t1, t2);
 88    merge(node, node, t3);
 89  }*/
 90  /*int get(pitem &node, int L, int R) {//[L, R) O(log)
 91    pitem t1, t2, t3;
 92    split(node, t1, t2, L);
 93    split(t2, t2, t3, R-L);
 94    push(t2);
 95    int ret = t2->sum;
 96    merge(node, t1, t2);
 97    merge(node, node, t3);
 98    return ret;
 99  }*/
100  /*/void push_all(pitem t){
101    if(t->p)push_all(t->p);
102    push(t);
103  }
104  pitem getRoot(pitem t, int& pos){ // get root and position for node t
105    push_all(t);
106    pos=cnt(t->l);
107    while(t->p){
108      pitem p=t->p;
109      if(t==p->r)pos+=cnt(p->l)+1;
110      t=p;
111    }
112    return t;
113  }*/
114  void output(pitem t){ // useful for debugging
115      if(!t)return;
116    push(t);
117    output(t->l);cout << ' ' << t->val;output(t->r);
118  }
```

3.9   STL rope

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

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

3.10   BIGInt

```
 1  #define BASEXP 6
 2  #define BASE 1000000
 3  #define LMAX 1000
 4  struct bint{
 5      int l;
 6      ll n[LMAX];
 7      bint(ll x=0){
 8          l=1;
 9          forn(i, LMAX){
10              if (x) l=i+1;
11              n[i]=x%BASE;
12              x/=BASE;
13
14          }
15      }
16      bint(string x){
17      l=(x.size()-1)/BASEXP+1;
18          fill(n, n+LMAX, 0);
19          ll r=1;
20          forn(i, sz(x)){
21              n[i / BASEXP] += r * (x[x.size()-1-i]-'0');
22              r*=10; if(r==BASE)r=1;
23          }
24      }
25      void out(){
26      cout << n[l-1];
27      dforn(i, l-1) printf("%6.6llu", n[i]);//6=BASEXP!
28    }
29    void invar(){
30      fill(n+l, n+LMAX, 0);
31      while(l>1 && !n[l-1]) l--;
32    }
33  };
34  bint operator+(const bint&a, const bint&b){
35    bint c;
36      c.l = max(a.l, b.l);
37      ll q = 0;
38      forn(i, c.l) q += a.n[i]+b.n[i], c.n[i]=q %BASE, q/=BASE;
39      if(q) c.n[c.l++] = q;
40      c.invar();
41      return c;
42  }
43  pair<bint, bool> lresta(const bint& a, const bint& b)   // c = a - b
44  {
45    bint c;
46      c.l = max(a.l, b.l);
```

```
47        ll q = 0;
48        forn(i, c.l) q += a.n[i]-b.n[i], c.n[i]=(q+BASE) %BASE, q=(q+BASE)/BASE
             -1;
49        c.invar();
50        return make_pair(c, !q);
51 }
52 bint& operator-= (bint& a, const bint& b){return a=lresta(a, b).first;}
53 bint operator- (const bint&a, const bint&b){return lresta(a, b).first;}
54 bool operator< (const bint&a, const bint&b){return !lresta(a, b).second;}
55 bool operator<= (const bint&a, const bint&b){return lresta(b, a).second;}
56 bool operator==(const bint&a, const bint&b){return a <= b && b <= a;}
57 bint operator*(const bint&a, ll b){
58     bint c;
59     ll q = 0;
60     forn(i, a.l) q += a.n[i]*b, c.n[i] = q %BASE, q/=BASE;
61     c.l = a.l;
62     while(q) c.n[c.l++] = q %BASE, q/=BASE;
63     c.invar();
64     return c;
65 }
66 bint operator*(const bint&a, const bint&b){
67     bint c;
68     c.l = a.l+b.l;
69     fill(c.n, c.n+b.l, 0);
70     forn(i, a.l){
71         ll q = 0;
72         forn(j, b.l) q += a.n[i]*b.n[j]+c.n[i+j], c.n[i+j] = q %BASE, q/=
             BASE;
73         c.n[i+b.l] = q;
74     }
75     c.invar();
76     return c;
77 }
78 pair<bint, ll> ldiv(const bint& a, ll b){// c = a / b ; rm = a % b
79   bint c;
80   ll rm = 0;
81   dforn(i, a.l){
82           rm = rm * BASE + a.n[i];
83           c.n[i] = rm / b;
84           rm %= b;
85       }
86     c.l = a.l;
87     c.invar();
88     return make_pair(c, rm);
89 }
90 bint operator/(const bint&a, ll b){return ldiv(a, b).first;}
91 ll operator%(const bint&a, ll b){return ldiv(a, b).second;}
92 pair<bint, bint> ldiv(const bint& a, const bint& b){
93   bint c;
94     bint rm = 0;
95     dforn(i, a.l){
96         if (rm.l==1 && !rm.n[0])
97             rm.n[0] = a.n[i];
98         else{
99             dforn(j, rm.l) rm.n[j+1] = rm.n[j];
100            rm.n[0] = a.n[i];
101            rm.l++;
102        }
103        ll q = rm.n[b.l] * BASE + rm.n[b.l-1];
104        ll u = q / (b.n[b.l-1] + 1);
105        ll v = q /  b.n[b.l-1] + 1;
106        while (u < v-1){
107            ll m = (u+v)/2;
108            if (b*m <= rm) u = m;
109            else v = m;
110        }
111        c.n[i]=u;
112        rm-=b*u;
113    }
114  c.l=a.l;
115    c.invar();
116    return make_pair(c, rm);
117 }
118 bint operator/(const bint&a, const bint&b){return ldiv(a, b).first;}
119 bint operator%(const bint&a, const bint&b){return ldiv(a, b).second;}
```

4   Algoritmos

4.1   Longest Increasing Subsecuence

```
1 //Para non-increasing, cambiar comparaciones y revisar busq binaria
2 //Given an array, paint it in the least number of colors so that each color
     turns to a non-increasing subsequence.
3 //Solution:Min number of colors=Length of the longest increasing subsequence
4 int N, a[MAXN];//secuencia y su longitud
5 ii d[MAXN+1];//d[i]=ultimo valor de la subsecuencia de tamanio i
6 int p[MAXN];//padres
7 vector<int> R;//respuesta
8 void rec(int i){
9   if(i==-1) return;
10  R.push_back(a[i]);
11  rec(p[i]);
12 }
13 int lis(){//O(nlogn)
14   d[0] = ii(-INF, -1); forn(i, N) d[i+1]=ii(INF, -1);
15   forn(i, N){
16     int j = upper_bound(d, d+N+1, ii(a[i], INF))-d;
17     if (d[j-1].first < a[i]&&a[i] < d[j].first){
18       p[i]=d[j-1].second;
19       d[j] = ii(a[i], i);
20     }
21   }
22   R.clear();
```

```
23    dforn(i, N+1) if(d[i].first!=INF){
24      rec(d[i].second);//reconstruir
25      reverse(R.begin(), R.end());
26      return i;//longitud
27    }
28    return 0;
29 }
```

4.2   Mo's

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

```
1  //Commented code should be used if updates are needed
2  int n,sq,nq; // array size, sqrt(array size), #queries
3  struct Qu{ //[l, r)
4    int l,r,id;
5    //int upds; // # of updates before this query
6  };
7  Qu qs[MAXN];
8  ll ans[MAXN]; // ans[i] = answer to ith query
9  /*struct Upd{
10   int p, v, prev; // pos, new_val, prev_val
11 };
12 Upd vupd[MAXN];*/
13
14 //Without updates
15 bool qcomp(const Qu &a, const Qu &b){
16     if(a.l/sq!=b.l/sq) return a.l<b.l;
17     return (a.l/sq)&1?a.r<b.r:a.r>b.r;
18 }
19
20 //With updates
21 /*bool qcomp(const Qu &a, const Qu &b){
22   if(a.l/sq != b.l/sq) return a.l<b.l;
23   if(a.r/sq != b.r/sq) return a.r<b.r;
24   return a.upds < b.upds;
25 }*/
26
27 //Without updates: O(n^2/sq + q*sq)
28 //with sq = sqrt(n): O(n*sqrt(n) + q*sqrt(n))
29 //with sq = n/sqrt(q): O(n*sqrt(q))
30 //
31 //With updates: O(sq*q + q*n^2/sq^2)
32 //with sq = n^(2/3): O(q*n^(2/3))
33 //with sq = (2*n^2)^(1/3) may improve a bit
34 void mos(){
35     forn(i,nq)qs[i].id=i;
36     sq=sqrt(n)+.5; // without updates
37     //sq=pow(n, 2/3.0)+.5; // with updates
38     sort(qs,qs+nq,qcomp);
39     int l=0,r=0;
40     init();
```

```
41     forn(i,nq){
42         Qu q=qs[i];
43         while(l>q.l)add(--l);
44         while(r<q.r)add(r++);
45         while(l<q.l)remove(l++);
46         while(r>q.r)remove(--r);
47         /*while(upds<q.upds){
48       if(vupd[upds].p >= l && vupd[upds].p < r) remove(vupd[upds].p);
49       v[vupd[upds].p] = vupd[upds].v; // do update
50       if(vupd[upds].p >= l && vupd[upds].p < r) add(vupd[upds].p);
51       upds++;
52     }
53     while(upds>q.upds){
54       upds--;
55       if(vupd[upds].p >= l && vupd[upds].p < r) remove(vupd[upds].p);
56       v[vupd[upds].p] = vupd[upds].prev; // undo update
57       if(vupd[upds].p >= l && vupd[upds].p < r) add(vupd[upds].p);
58     }*/
59         ans[q.id]=get_ans();
60     }
61 }
```

5   Strings

5.1   Z function

```
1  //z[i] = length of longest substring starting from s[i] that is prefix of s
2  //z[i] = max k: s[0,k) == s[i,i+k)
3  vector<int> zFunction(string &s) {
4    int l=0, r=0, n=s.size();
5    vector<int> z(n,0);
6    forr(i, 1, n) {
7      if(i<=r) z[i] = min(r-i+1, z[i-l]);
8      while(i+z[i]<n && s[z[i]]==s[i+z[i]]) z[i]++;
9      if(i+z[i]-1>r) l=i, r=i+z[i]-1;
10   }
11   return z;
12 }
13 void match(string &T,string &P) { //Text, Pattern -- O(|T|+|P|)
14   string s = P+'$'+T;//'$' should be a character that is not present in T
15   vector<int> z = zFunction(s);
16   forr(i, P.size()+1, s.size())
17     if(z[i] == P.size()); //match found, idx = i-P.size()-1
18 }
```

5.2   KMP

```
1  // b[i] = longest border of t[0,i) = length of the longest prefix of
2  // the substring P[0..i-1) that is also suffix of the substring P[0..i)
3  // For "AABAACAABAA", b[i] = {-1, 0, 1, 0, 1, 2, 0, 1, 2, 3, 4, 5}
4  vector<int> kmppre(string& P) { //
5    vector<int> b(P.size()+1); b[0]=-1;
6    int j=-1;
7    forn(i, P.size()) {
8      while(j>=0 && P[i]!=P[j]) j = b[j];
9      b[i+1] = ++j;
10   }
11   return b;
12 }
13 void kmp(string& T, string& P) { //Text, Pattern -- O(|T|+|P|)
14   int j = 0;
15   vector<int> b = kmppre(P);
16   forn(i, T.size()) {
17     while(j>=0 && T[i]!=P[j]) j = b[j];
18     if(++j == P.size())
19     {
20       //Match at i-j+1
21       j=b[j];
22     }
23   }
24 }
```

5.3   Hashing

5.3.1   Simple hashing (no substring hash)

```
1  struct Hash {
2    //P must be a prime number, could be randomly generated,
3    //sometimes is good to make it close to alphabet size
4    int P=1777771, MOD[2], PI[2];
5    ll h[2];
6    vector<ll> vp[2];
7    deque<char> x;
8    Hash(vector<char>& s) {
9      forn(i,sz(s)) x.pb(s[i]);
10     //MOD[i] must be a prime of this order, could be randomly generated
11     MOD[0]=999727999; MOD[1]=1070777777;
12     //PI[i] = P^-1 % MOD[i]
13     PI[0]=325255434; PI[1]=10018302;
14     forn(k, 2)
15       vp[k].rsz(s.size()+1);
16     forn(k, 2) {
17       h[k] = 0; vp[k][0] = 1;
18       ll p=1;
19       forr(i, 1, s.size()+1) {
20         h[k] = (h[k] + p*s[i-1]) % MOD[k];
21         vp[k][i] = p = (p*P) % MOD[k];
```

```
22     }
23   }
24 }
25 //Put the value val in position pos and update the hash value
26 void change(int pos, int val) {
27   forn(i,2)
28     h[i] = (h[i] + vp[i][pos] * (val - x[pos] + MOD[i])) % MOD[i];
29   x[pos] = val;
30 }
31 //Add val to the end of the current string
32 void push_back(int val) {
33   int pos = sz(x);
34   x.pb(val);
35   forn(k, 2)
36   {
37     assert(pos <= sz(vp[k]));
38     if(pos == sz(vp[k])) vp[k].pb(vp[k].back()*P%MOD[k]);
39     ll p = vp[k][pos];
40     h[k] = (h[k] + p*val) % MOD[k];
41   }
42 }
43 //Delete the first element of the current string
44 void pop_front() {
45   assert(sz(x) > 0);
46   forn(k,2)
47   {
48     h[k] = (h[k] - x[0] + MOD[k]) % MOD[k];
49     h[k] = h[k] * PI[k] % MOD[k];
50   }
51   x.pop_front();
52 }
53 ll getHashVal() {return (h[0]<<32)|h[1];}
54 };
```

5.3.2   Classic hashing (with substring hash)

```
1  struct Hash {
2    //P must be a prime number, could be randomly generated,
3    //sometimes is good to make it close to alphabet size
4    int P=1777771, MOD[2], PI[2];
5    vector<int> h[2], pi[2];
6    vector<ll> vp[2]; //Only used if getChanged is used (delete it if not)
7    Hash(string& s) {
8      //MOD[i] must be a prime of this order, could be randomly generated
9      MOD[0]=999727999; MOD[1]=1070777777;
10     //PI[i] = P^-1 % MOD[i]
11     PI[0]=325255434; PI[1]=10018302;
12     forn(k, 2)
13       h[k].rsz(s.size()+1), pi[k].rsz(s.size()+1), vp[k].rsz(s.size()+1);
14     forn(k, 2) {
15       h[k][0] = 0; pi[k][0] = vp[k][0] = 1;
```

```
16      ll p=1;
17      forr(i, 1, s.size()+1) {
18        h[k][i] = (h[k][i-1] + p*s[i-1]) % MOD[k];
19        pi[k][i] = (1LL * pi[k][i-1] * PI[k]) % MOD[k];
20        vp[k][i] = p = (p*P) % MOD[k];
21      }
22    }
23  }
24  ll get(int s, int e) { // get hash value of the substring [s, e)
25    ll H[2];
26    forn(i, 2) {
27      H[i] = (h[i][e] - h[i][s] + MOD[i]) % MOD[i];
28      H[i] = (1LL * H[i] * pi[i][s]) % MOD[i];
29    }
30    return (H[0]<<32)|H[1];
31  }
32  //get hash value of [s, e) if origVal in pos is changed to val
33  //Assumes s <= pos < e. If multiple changes are needed,
34    //do what is done in the for loop for every change
35    ll getChanged(int s, int e, int pos, int val, int origVal) {
36        ll hv = get(s,e), h[2];
37        h[1] = hv & ((1LL<<32)-1);
38        h[0] = hv >> 32;
39        forn(i, 2)
40          h[i] = (h[i] + vp[i][pos] * (val - origVal + MOD[i])) % MOD[i];
41        return (h[0]<<32)|h[1];
42    }
43  };
```

5.3.3   Hashing 128 bits

```
1  #define bint __int128 // needs gcc compiler?
2  struct Hash {
3    bint MOD=212345678987654321LL, P=1777771, PI=1069557741089659571LL;
4    vector<bint> h, pi;
5    Hash(string& s) {
6      assert((P*PI)%MOD == 1);
7      h.resize(s.size()+1); pi.resize(s.size()+1);
8      h[0]=0; pi[0]=1;
9      bint p=1;
10     forr(i, 1, s.size()+1) {
11       h[i] = (h[i-1] + p*s[i-1]) % MOD;
12       pi[i] = (pi[i-1] * PI) % MOD;
13       p = (p*P) % MOD;
14     }
15   }
16   ll get(int s, int e){ // get hash value of the substring [s, e)
17     return (((h[e]-h[s]+MOD)%MOD)*pi[s])%MOD;
18   }
19 };
```

5.4   Trie

```
1  struct trie{
2    map<char, trie> m;
3    void add(const string &s, int p=0)
4    {
5      if(s[p]) m[s[p]].add(s, p+1);
6    }
7    void dfs()
8    {
9      //Do stuff
10     forall(it, m)
11     it->second.dfs();
12   }
13 };
```

5.5   Aho Corasick

```
1  struct vertex {
2    map<char,int> next,go;
3    int p,link,leafLink;
4    char pch;
5    vector<int> leaf;
6    vertex(int p=-1, char pch=-1):p(p),link(-1),leafLink(-1),pch(pch){}
7  };
8  vector<vertex> t;
9  void aho_init(){ //do not forget!!
10   t.clear();t.pb(vertex());
11 }
12 void add_string(string s, int id){
13   int v=0;
14   for(char c:s){
15     if(!t[v].next.count(c)){
16       t[v].next[c]=t.size();
17       t.pb(vertex(v,c));
18     }
19     v=t[v].next[c];
20   }
21   t[v].leaf.pb(id);
22 }
23 int go(int v, char c);
24 int get_link(int v){
25   if(t[v].link<0)
26     if(!v||!t[v].p)t[v].link=0;
27     else t[v].link=go(get_link(t[v].p),t[v].pch);
28   return t[v].link;
29 }
30 int go(int v, char c){
31   if(!t[v].go.count(c))
32     if(t[v].next.count(c))t[v].go[c]=t[v].next[c];
33     else t[v].go[c]=v==0?0:go(get_link(v),c);
```

```
34    return t[v].go[c];
35  }
36  int get_leaf_link(int v){
37    if(t[v].leafLink < 0){
38      if(!v||!t[v].p) t[v].leafLink = 0;
39      else
40      {
41        if(t[get_link(v)].leaf.empty()) t[v].leafLink = get_leaf_link(t[v].
                link);
42        else t[v].leafLink = t[v].link;
43      }
44    }
45    return t[v].leafLink;
46  }
```

5.6   Manacher

```
1   int d1[MAXN];//d1[i] = max odd palindrome centered on i
2   int d2[MAXN];//d2[i] = max even palindrome centered on i
3   //s  aabbaacaabbaa
4   //d1 1111117111111
5   //d2 0103010010301
6   void manacher(string &s) { // O(|S|) - find longest palindromic substring
7     int l=0, r=-1, n=s.size();
8     forn(i, n) { // build d1
9       int k = i>r? 1 : min(d1[l+r-i], r-i);
10      while(i+k<n && i-k>=0 && s[i+k]==s[i-k]) k++;
11      d1[i] = k--;
12      if(i+k > r) l=i-k, r=i+k;
13    }
14    l=0, r=-1;
15    forn(i, n) { // build d2
16      int k = (i>r? 0 : min(d2[l+r-i+1], r-i+1))+1;
17      while(i+k<=n && i-k>=0 && s[i+k-1]==s[i-k]) k++;
18      d2[i] = --k;
19      if(i+k-1 > r) l=i-k, r=i+k-1;
20    }
21  }
```

5.7   Suffix array

5.7.1   Slow version O(n*logn*logn)

```
1   pair<int, int> sf[MAXN];
2   bool sacomp(int lhs, int rhs) {return sf[lhs] < sf[rhs];}
3   vector<int> constructSA(string& s) { // O(n log^2(n))
4     int n = s.size();                     // (sometimes fast enough)
5     vector<int> sa(n), r(n);
```

```
6     forn(i,n) r[i] = s[i]; //r[i]: equivalence class of s[i..i+m]
7     for(int m=1; m<n; m*=2) {
8       //sf[i] = {r[i], r[i+m]}, used to sort for next equivalence classes
9       forn(i,n) sa[i] = i, sf[i] = {r[i], i+m<n ? r[i+m] : -1};
10      stable_sort(sa.begin(), sa.end(), sacomp); //O(n log(n))
11      r[sa[0]] = 0;
12      //if sf[sa[i]] == sf[sa[i-1]] then same equivalence class
13      forr(i,1,n) r[sa[i]] = sf[sa[i]]!=sf[sa[i-1]] ? i : r[sa[i-1]];
14    }
15    return sa;
16  }
```

5.7.2   Fast version O(n*logn)

```
1   #define RB(x) (x<n ? r[x] : 0)
2   void csort(vector<int>& sa, vector<int>& r, int k){ //counting sort O(n)
3       int n = sa.size();
4       vector<int> f(max(255,n),0), t(n);
5       forn(i, n) f[RB(i+k)]++;
6       int sum = 0;
7       forn(i, max(255,n)) f[i] = (sum+=f[i]) - f[i];
8       forn(i, n) t[f[RB(sa[i]+k)]++] = sa[i];
9       sa = t;
10  }
11  vector<int> constructSA(string& s){ // O(n logn)
12      int n = s.size(), rank;
13      vector<int> sa(n), r(n), t(n);
14      forn(i,n) sa[i] = i, r[i] = s[i];//r[i]: equivalence class of s[i..i+k]
15      for(int k=1; k<n; k*=2) {
16          csort(sa, r, k); csort(sa, r, 0); //counting sort, O(n)
17          t[sa[0]] = rank = 0; //t : equivalence classes array for next size
18          forr(i, 1, n) {
19              //check if sa[i] and sa[i-1] are in te same equivalence class
20              if(r[sa[i]]!=r[sa[i-1]] || RB(sa[i]+k)!=RB(sa[i-1]+k)) rank++;
21              t[sa[i]] = rank;
22          }
23          r = t;
24          if(r[sa[n-1]] == n-1) break;
25      }
26      return sa;
27  }
```

5.8   Longest common prefix (LCP)

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

```
6    int n = s.size(), L = 0;
7    vector<int> lcp(n), plcp(n), phi(n);
8    phi[sa[0]] = -1;
9    forr(i, 1, n) phi[sa[i]] = sa[i-1];
10   forn(i, n) {
11     if(phi[i]<0) {plcp[i] = 0; continue;}
12     while(s[i+L] == s[phi[i]+L]) L++;
13     plcp[i] = L;
14     L = max(L-1, 0);
15   }
16   forn(i,n) lcp[i] = plcp[sa[i]];
17   return lcp; // lcp[i]=LCP(sa[i-1],sa[i])
18 }
```

5.9   Suffix automaton

```
1  //The substrings of S can be decomposed into equivalence classes
2  //2 substr are of the same class if they have the same set of endpos
3  //Example: endpos("bc") = {2, 4, 6} in "abcbcbc"
4  //Each class is a node of the automaton.
5  //Len is the longest substring of each class
6  //Link in state X is the state where the longest suffix of the longest
7  //substring in X, with a different endpos set, belongs
8  //The links form a tree rooted at 0
9  //last is the state of the whole string after each extention
10 struct state {int len, link; map<char,int> next;}; //clear next!!
11 state st[MAXN];
12 int sz, last;
13 void sa_init() {
14   last = st[0].len = 0; sz = 1;
15   st[0].link = -1;
16 }
17 void sa_extend(char c) {
18   int k = sz++, p; //k = new state
19   st[k].len = st[last].len + 1;
20   //while c is not present in p assign it as edge to the new state and
21   //move through link (note that p always corresponds to a suffix state)
22   for(p=last; p!=-1 && !st[p].next.count(c); p=st[p].link) st[p].next[c]=k;
23   if(p == -1) st[k].link = 0;
24   else {
25     //state p already goes to state q through char c. Then, link of k
26     //should go to a state with len = st[p].len + 1 (because of c)
27     int q = st[p].next[c];
28     if(st[p].len+1 == st[q].len) st[k].link = q;
29     else {
30       //q is not the state we are looking for. Then, we
31       //create a clone of q (w) with the desired length
32       int w = sz++;
33       st[w].len = st[p].len + 1;
34       st[w].next = st[q].next; st[w].link = st[q].link;
35       //go through links from p and while next[c] is q, change it to w
```

```
36       for(; p!=-1 && st[p].next[c]==q; p=st[p].link) st[p].next[c] = w;
37       //change link of q from p to w, and finally set link of k to w
38       st[q].link = st[k].link = w;
39     }
40   }
41   last = k;
42 }
43 //  input: abcbcbc
44 //  i,link,len,next
45 //  0 -1 0 (a,1) (b,5) (c,7)
46 //  1 0 1 (b,2)
47 //  2 5 2 (c,3)
48 //  3 7 3 (b,4)
49 //  4 9 4 (c,6)
50 //  5 0 1 (c,7)
51 //  6 11 5 (b,8)
52 //  7 0 2 (b,9)
53 //  8 9 6 (c,10)
54 //  9 5 3 (c,11)
55 //  10 11 7
56 //  11 7 4 (b,8)
```

5.10   Suffix tree

```
1  //The SuffixTree of S is the compressed trie that would result
2  //after inserting every suffix of S.
3  //As it is a COMPRESSED trie, some edges may correspond to strings, instead
4  //of chars, and the compression is done in a way that every vertex that
5  //doesn't correspond to a suffix and has only one descendent, is omitted.
6  struct SuffixTree {
7    char s[MAXN];
8    map<int,int> to[2*MAXN];//key is fst char of substring on edge to value
9    //s[fpos[i], fpos[i]+len[i]) is the substring on the edge between
10   //i's father and i.
11   //link[i] goes to the node that corresponds to the substring that result
12   //after "removing" the first character of the substring that i represents
13   //note that link is only defined for every internal (non-leaf) node.
14   int len[2*MAXN] = {INF}, fpos[2*MAXN], link[2*MAXN];
15   int node = 0, pos = 0, sz = 1, n = 0;
16   int make_node(int p, int l) {
17     fpos[sz] = p; len[sz] = l; return sz++;}
18   void go_edge() {
19     while(pos > len[to[node][s[n-pos]]]) {
20       node = to[node][s[n-pos]];
21       pos -= len[node];
22     }
23   }
24   void add(char c) {
25     s[n++] = c; pos++;
26     int last = 0;
27     while(pos > 0) {
```

```
28        go_edge();
29        int edge = s[n-pos];
30        int& v = to[node][edge];
31        int t = s[fpos[v]+pos-1];
32        if(v == 0) {
33          v = make_node(n-pos, INF);
34          link[last] = node; last = 0;
35        }
36        else if(t == c) {link[last] = node; return;}
37        else {
38          int u = make_node(fpos[v], pos-1);
39          to[u][c] = make_node(n-1, INF);
40          to[u][t] = v;
41          fpos[v] += pos-1; len[v] -= pos-1;
42          v = u; link[last] = u; last = u;
43        }
44        if(node == 0) pos--;
45        else node = link[node];
46      }
47    }
48  };
```

6   Geometría

6.1   Punto

```
1  struct pto{
2    double x, y;
3    pto(double x=0, double y=0):x(x),y(y){}
4    pto operator+(pto a){return pto(x+a.x, y+a.y);}
5    pto operator-(pto a){return pto(x-a.x, y-a.y);}
6    pto operator+(double a){return pto(x+a, y+a);}
7    pto operator*(double a){return pto(x*a, y*a);}
8    pto operator/(double a){return pto(x/a, y/a);}
9    //dot product, producto interno:
10   double operator*(pto a){return x*a.x+y*a.y;}
11   //module of the cross product or vectorial product:
12   //if a is less than 180 clockwise from b, a^b>0
13   double operator^(pto a){return x*a.y-y*a.x;}
14   //returns true if this is at the left side of line qr
15   bool left(pto q, pto r){return ((q-*this)^(r-*this))>0;}
16   bool operator<(const pto &a) const{return x<a.x-EPS || (abs(x-a.x)<EPS &&
        y<a.y-EPS);}
17 bool operator==(pto a){return abs(x-a.x)<EPS && abs(y-a.y)<EPS;}
18   double norm(){return sqrt(x*x+y*y);}
19   double norm_sq(){return x*x+y*y;}
20 };
21 double dist(pto a, pto b){return (b-a).norm();}
22 typedef pto vec;
23
```

```
24 double angle(pto a, pto o, pto b){
25   pto oa=a-o, ob=b-o;
26   return atan2(oa^ob, oa*ob);}
27
28 //rotate p by theta rads CCW w.r.t. origin (0,0)
29 pto rotate(pto p, double theta){
30   return pto(p.x*cos(theta)-p.y*sin(theta),
31     p.x*sin(theta)+p.y*cos(theta));
32 }
```

6.2   Orden Radial de Puntos

```
1  struct Cmp{//orden total de puntos alrededor de un punto r
2    pto r;
3    Cmp(pto r):r(r) {}
4    int cuad(const pto &a) const{
5      if(a.x > 0 && a.y >= 0)return 0;
6      if(a.x <= 0 && a.y > 0)return 1;
7      if(a.x < 0 && a.y <= 0)return 2;
8      if(a.x >= 0 && a.y < 0)return 3;
9      assert(a.x ==0 && a.y==0);
10     return -1;
11   }
12   bool cmp(const pto&p1, const pto&p2)const{
13     int c1 = cuad(p1), c2 = cuad(p2);
14     if(c1==c2) return p1.y*p2.x<p1.x*p2.y;
15         else return c1 < c2;
16   }
17   bool operator()(const pto&p1, const pto&p2) const{
18     return cmp(pto(p1.x-r.x,p1.y-r.y),pto(p2.x-r.x,p2.y-r.y));
19   }
20 };
```

6.3   Linea

```
1  int sgn(ll x){return x<0? -1 : !!x;}
2  struct line{
3    line() {}
4    double a,b,c;//Ax+By=C
5  //pto MUST store float coordinates!
6    line(double a, double b, double c):a(a),b(b),c(c){}
7    // TO DO chequear porque paso problema metiendo negativo el C (-(todo el
        calculo como esta))
8    line(pto p, pto q): a(q.y-p.y), b(p.x-q.x), c(a*p.x+b*p.y) {}
9    int side(pto p){return sgn(ll(a) * p.x + ll(b) * p.y - c);}
10 };
11 bool parallels(line l1, line l2){return abs(l1.a*l2.b-l2.a*l1.b)<EPS;}
12 pto inter(line l1, line l2){//intersection
```

```
13   double det=l1.a*l2.b-l2.a*l1.b;
14   if(abs(det)<EPS) return pto(INF, INF);//parallels
15   return pto(l2.b*l1.c-l1.b*l2.c, l1.a*l2.c-l2.a*l1.c)/det;
16 }
```

6.4 Segmento

```
1  struct segm{
2    pto s,f;
3    segm(pto s, pto f):s(s), f(f) {}
4    pto closest(pto p) {//use for dist to point
5        double l2 = dist_sq(s, f);
6        if(l2==0.) return s;
7        double t =((p-s)*(f-s))/l2;
8        if (t<0.) return s;//not write if is a line
9        else if(t>1.)return f;//not write if is a line
10       return s+((f-s)*t);
11   }
12     bool inside(pto p){return abs(dist(s, p)+dist(p, f)-dist(s, f))<EPS;}
13 };
14
15 //NOTA: Si los segmentos son coolineales solo devuelve un punto de
       interseccion
16 pto inter(segm s1, segm s2){
17     if(s1.inside(s2.s)) return s2.s; //Fix cuando son colineales
18     if(s1.inside(s2.f)) return s2.f; //Fix cuando son colineales
19   pto r=inter(line(s1.s, s1.f), line(s2.s, s2.f));
20     if(s1.inside(r) && s2.inside(r)) return r;
21   return pto(INF, INF);
22 }
```

6.5 Rectangulo

```
1  struct rect{
2    //lower-left and upper-right corners
3    pto lw, up;
4  };
5  //returns if there's an intersection and stores it in r
6  bool inter(rect a, rect b, rect &r){
7    r.lw=pto(max(a.lw.x, b.lw.x), max(a.lw.y, b.lw.y));
8    r.up=pto(min(a.up.x, b.up.x), min(a.up.y, b.up.y));
9  //check case when only a edge is common
10   return r.lw.x<r.up.x && r.lw.y<r.up.y;
11 }
```

6.6 Circulo

```
1  vec perp(vec v){return vec(-v.y, v.x);}
2  line bisector(pto x, pto y){
3    line l=line(x, y); pto m=(x+y)/2;
4    return line(-l.b, l.a, -l.b*m.x+l.a*m.y);
5  }
6  struct Circle{
7    pto o;
8    double r;
9    Circle(pto x, pto y, pto z){
10     o=inter(bisector(x, y), bisector(y, z));
11     r=dist(o, x);
12   }
13   pair<pto, pto> ptosTang(pto p){
14     pto m=(p+o)/2;
15     tipo d=dist(o, m);
16     tipo a=r*r/(2*d);
17     tipo h=sqrt(r*r-a*a);
18     pto m2=o+(m-o)*a/d;
19     vec per=perp(m-o)/d;
20     return make_pair(m2-per*h, m2+per*h);
21   }
22 };
23 //finds the center of the circle containing p1 and p2 with radius r
24 //as there may be two solutions swap p1, p2 to get the other
25 bool circle2PtsRad(pto p1, pto p2, double r, pto &c){
26         double d2=(p1-p2).norm_sq(), det=r*r/d2-0.25;
27         if(det<0) return false;
28         c=(p1+p2)/2+perp(p2-p1)*sqrt(det);
29         return true;
30 }
31 #define sqr(a) ((a)*(a))
32 #define feq(a,b) (fabs((a)-(b))<EPS)
33 pair<tipo, tipo> ecCuad(tipo a, tipo b, tipo c){//a*x*x+b*x+c=0
34   tipo dx = sqrt(b*b-4.0*a*c);
35   return make_pair((-b + dx)/(2.0*a),(-b - dx)/(2.0*a));
36 }
37 pair<pto, pto> interCL(Circle c, line l){
38   bool sw=false;
39   if((sw=feq(0,l.b))){
40   swap(l.a, l.b);
41   swap(c.o.x, c.o.y);
42   }
43   pair<tipo, tipo> rc = ecCuad(
44   sqr(l.a)+sqr(l.b),
45   2.0*l.a*l.b*c.o.y-2.0*(sqr(l.b)*c.o.x+l.c*l.a),
46   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
47   );
48   pair<pto, pto> p( pto(rc.first, (l.c - l.a * rc.first) / l.b),
49           pto(rc.second, (l.c - l.a * rc.second) / l.b) );
50   if(sw){
51   swap(p.first.x, p.first.y);
52   swap(p.second.x, p.second.y);
53   }
```

```
54    return p;
55 }
56 pair<pto, pto> interCC(Circle c1, Circle c2){
57   line l;
58   l.a = c1.o.x-c2.o.x;
59   l.b = c1.o.y-c2.o.y;
60   l.c = (sqr(c2.r)-sqr(c1.r)+sqr(c1.o.x)-sqr(c2.o.x)+sqr(c1.o.y)
61   -sqr(c2.o.y))/2.0;
62   return interCL(c1, l);
63 }
```

6.7   Area de poligono

```
1 double area(vector<pto> &p){//O(sz(p))
2   double area=0;
3   forn(i, sz(p)) area+=p[i]^p[(i+1)%sz(p)];
4   //if points are in clockwise order then area is negative
5   return abs(area)/2;
6 }
7 //Area ellipse = M_PI*a*b where a and b are the semi axis lengths
8 //Area triangle = sqrt(s*(s-a)(s-b)(s-c)) where s=(a+b+c)/2
```

6.8   Punto en poligono

```
1 //checks if v is inside of P, using ray casting
2 //works with convex and concave.
3 //excludes boundaries, handle it separately using segment.inside()
4 bool inPolygon(pto v, vector<pto>& P) {
5   bool c = false;
6   forn(i, sz(P)){
7     int j=(i+1)%sz(P);
8     if((P[j].y>v.y) != (P[i].y > v.y) &&
9   (v.x < (P[i].x - P[j].x) * (v.y-P[j].y) / (P[i].y - P[j].y) + P[j].x))
10       c = !c;
11   }
12   return c;
13 }
```

6.9   Punto en Poligono Convexo

$$O(\log n)$$

```
1 void normalize(vector<pto> &pt) //delete collinear points first!
2 {
3   //this makes it clockwise:
4   if(pt[2].left(pt[0], pt[1])) reverse(pt.begin(), pt.end());
5   int n=sz(pt), pi=0;
```

```
6    forn(i, n)
7      if(pt[i].x<pt[pi].x || (pt[i].x==pt[pi].x && pt[i].y<pt[pi].y))
8        pi=i;
9    vector<pto> shift(n);//puts pi as first point
10   forn(i, n) shift[i]=pt[(pi+i)%n];
11   pt.swap(shift);
12 }
13 bool inPolygon(pto p, const vector<pto> &pt)
14 {
15   //call normalize first!
16   if(p.left(pt[0], pt[1]) || p.left(pt[sz(pt)-1], pt[0])) return false;
17   int a=1, b=sz(pt)-1;
18   while(b-a>1)
19   {
20     int c=(a+b)/2;
21     if(!p.left(pt[0], pt[c])) a=c;
22     else b=c;
23   }
24   return !p.left(pt[a], pt[a+1]);
25 }
```

6.10   Chequeo de Convex

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

6.11   Convex Hull

```
1 //stores convex hull of P in S, CCW order
2 //left must return >=0 to delete collinear points!
3 void CH(vector<pto>& P, vector<pto> &S){
4   S.clear();
5   sort(P.begin(), P.end());//first x, then y
6   forn(i, sz(P)){//lower hull
7     while(sz(S)>= 2 && S[sz(S)-1].left(S[sz(S)-2], P[i])) S.pop_back();
8     S.pb(P[i]);
9   }
10   S.pop_back();
11   int k=sz(S);
12   dforn(i, sz(P)){//upper hull
13     while(sz(S) >= k+2 && S[sz(S)-1].left(S[sz(S)-2], P[i])) S.pop_back();
```

```
14        S.pb(P[i]);
15    }
16    S.pop_back();
17 }
```

6.12   Convex Hull Dinamico

```
1  // Superior
2  struct Semi_Chull {
3      set<pto> pts; // this set maintains the convex hull without collinears
              points
4      //in case that we want to have them on the set, make the changes
              commented below
5      bool check(pto p) {
6          if(pts.empty()) return false;
7          if(*pts.rbegin() < p) return false;
8          if(p < *pts.begin()) return false;
9          auto it = pts.lower_bound(p);
10         if(it->x == p.x) return p.y <= it->y; //ignore it to take in count
                 collinears points too
11         pto b = *it;
12         pto a = *prev(it);
13         return ((b-p)^(a-p))+EPS >= 0; //> 0 to take in count collinears
                 points too
14     }
15
16     void add(pto p) {
17         if(check(p)) return;
18         pts.erase(p); pts.insert(p);
19         auto it = pts.find(p);
20
21         while(true) {
22             if(next(it) == pts.end() || next(next(it)) == pts.end()) break;
23             pto a = *next(it);
24             pto b = *next(next(it));
25             if(((b-a)^(p-a))+EPS >= 0) { //> 0 to take in count collinears
                     points too
26                 pts.erase(next(it));
27             } else break;
28         }
29
30         it = pts.find(p);
31
32         while(true) {
33             if(it == pts.begin() || prev(it) == pts.begin()) break;
34             pto a = *prev(it);
35             pto b = *prev(prev(it));
36             if(((b-a)^(p-a))-EPS <=0) { //< 0 to take in count collinears
                     points too
37                 pts.erase(prev(it));
38             } else break;
39         }
40     }
41 };
42
43 struct Chull {
44     Semi_Chull sup, inf;
45     void add(pto p) {
46         sup.add(p);
47         pto np = {-1*p.x, -1*p.y};
48         inf.add(np);
49     }
50     bool check(pto p) {
51     pto np = {-1*p.x, -1*p.y};
52         return sup.check(p) && inf.check(np);
53     }
54 };
```

6.13 Convex Hull Trick

```
1  struct Line{tipo m,h;};
2  //Coordenada X de la interseccion
3  tipo inter(Line a, Line b){
4      tipo x=b.h-a.h, y=a.m-b.m;
5      return x/y+(x%y?!((x>0)^(y>0)):0);//==ceil(x/y)
6  }
7  struct CHT {
8    vector<Line> c;
9    bool mx;
10   int pos;
11   CHT(bool mx=0):mx(mx),pos(0){}//mx=1 si las query devuelven el max
12   //Indexar de der a izq si se inserto con m/-m creciente (min/max)
13   inline Line acc(int i){return c[c[0].m>c.back().m? i : sz(c)-1-i];}
14   inline bool irre(Line x, Line y, Line z){
15       return c[0].m>z.m? inter(y, z) <= inter(x, y)
16                         : inter(y, z) >= inter(x, y);
17   }
18   void add(tipo m, tipo h) {//O(1), los m tienen que entrar ordenados
19       if(mx) m*=-1, h*=-1; //max f_i(x) == min -f_i(x)
20     Line l=(Line){m, h};
21     //Manejar caso igual pendiente
22       if(sz(c) && m==c.back().m) { l.h=min(h, c.back().h), c.pop_back();
                if(pos) pos--; }
23       while(sz(c)>=2 && irre(c[sz(c)-2], c[sz(c)-1], l)) { c.pop_back();
                if(pos) pos--; }
24       c.pb(l);
25   }
26   inline bool fbin(tipo x, int m) {return inter(acc(m), acc(m+1))>x;}
27   tipo eval(tipo x){
28     int n = sz(c);
29     //query con x no ordenados O(lgn)
30     int a=-1, b=n-1;
31     while(b-a>1) { int m = (a+b)/2;
32       if(fbin(x, m)) b=m;
33       else a=m;
34     }
35     return (acc(b).m*x+acc(b).h)*(mx?-1:1);
36         //query O(1), con x ordenado
37     while(pos>0 && fbin(x, pos-1)) pos--;
38     while(pos<n-1 && !fbin(x, pos)) pos++;
39     return (acc(pos).m*x+acc(pos).h)*(mx?-1:1);
40   }
41 } ch;
```

6.14 Convex Hull Trick Dinamico

```
1  const ll is_query = -(1LL<<62);
2  struct Line {
3      ll m, b;
```

```
4      mutable multiset<Line>::iterator it;
5      const Line *succ(multiset<Line>::iterator it) const;
6      bool operator<(const Line& rhs) const {
7          if (rhs.b != is_query) return m < rhs.m;
8          const Line *s=succ(it);
9          if(!s) return 0;
10         ll x = rhs.m;
11         return b - s->b < (s->m - m) * x;
12     }
13 };
14 struct HullDynamic : public multiset<Line>{ // will maintain upper hull for
       maximum
15     bool bad(iterator y) {
16         iterator z = next(y);
17         if (y == begin()) {
18             if (z == end()) return 0;
19             return y->m == z->m && y->b <= z->b;
20         }
21         iterator x = prev(y);
22         if (z == end()) return y->m == x->m && y->b <= x->b;
23         return (x->b - y->b)*(z->m - y->m) >= (y->b - z->b)*(y->m - x->m);
24     }
25     iterator next(iterator y){return ++y;}
26     iterator prev(iterator y){return --y;}
27     void insert_line(ll m, ll b) {
28         iterator y = insert((Line) { m, b });
29         y->it=y;
30         if (bad(y)) { erase(y); return; }
31         while (next(y) != end() && bad(next(y))) erase(next(y));
32         while (y != begin() && bad(prev(y))) erase(prev(y));
33     }
34     ll eval(ll x) {
35         Line l = *lower_bound((Line) { x, is_query });
36         return l.m * x + l.b;
37     }
38 }h;
39 const Line *Line::succ(multiset<Line>::iterator it) const{
40     return (++it==h.end()? NULL : &*it);}
```

6.15 Convex Hull Trick with Points

```
1  struct pto{
2    tipo x,y;
3    pto operator-(pto a){return {x-a.x, y-a.y};}
4    //dot product, producto interno:
5    tipo operator*(pto a){return x*a.x+y*a.y;}
6    //module of the cross product or vectorial product:
7    //if a is less than 180 clockwise from b, a^b>0
8    tipo operator^(pto a){return x*a.y-y*a.x;}
9    //returns true if this is at the left side of line qr
10   bool left(pto &q, pto &r){return ((q-*this)^(r-*this))>0;}
```

```
11 };
12 typedef pto vec;
13 struct CHT{
14   deque<pto> c;
15   bool mn;
16   int pos;
17   CHT(bool mn=0):mn(mn),pos(0){}//mn=1 si querys de min
18   void add(tipo m, tipo h) {//O(1)
19     //Querys max f_i, necesitamos capsula inferior
20     if(mn) m*=-1, h*=-1; //min f_i(x) == max -f_i(x)
21     pto p = {-m,-h}; //Line(m,h) == pto(-m,-h)
22     if(c.empty()) {c.pb(p); return;}
23     //el nuevo m tiene que ser el menor o el mayor (m "ordenados")
24     assert(p.x <= c[0].x || p.x >= c.back().x);
25     if(p.x <= c[0].x) {
26       if(p.x == c[0].x) p.y = min(p.y, c[0].y), c.pop_front(), pos--;
27       while(c.size() > 1 && !c[1].left(p, c[0])) c.pop_front(), pos--;
28       c.push_front(p);
29     }
30     else {
31       if(p.x == c.back().x) p.y = min(p.y, c.back().y), c.pop_back();
32       while(c.size() > 1 && !p.left(c[sz(c)-2], c[sz(c)-1])) c.pop_back();
33       c.pb(p);
34     }
35     pos = min(max(0,pos), sz(c)-1);
36   }
37   tipo eval(tipo x){ //max f_i(x) == max p_i*(-x,-1)  (dot prod)
38     vec vx = {-x,-1};
39     //querys O(log), para x desordenado
40     int L = 0,R = sz(c)-1, M;
41     while(L < R) {
42       M = (L+R)/2;
43       if(c[M+1]*vx > c[M]*vx) L = M+1;
44       else R = M;
45     }
46     return c[L]*vx*(mn?-1:1);
47     //querys O(1), para x "ordenados"
48     while(pos > 0 && c[pos-1]*vx > c[pos]*vx) pos--;
49     while(pos < sz(c)-1 && c[pos+1]*vx > c[pos]*vx) pos++;
50     return c[pos]*vx*(mn?-1:1);
51   }
52 };
```

6.16   Cortar poligono

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

```
7     if(left1>=0) P.pb(Q[i]);
8     if(left1*left2<0)
9       P.pb(inter(line(Q[i], Q[(i+1)%sz(Q)]), line(a, b)));
10   }
11 }
```

6.17   Intersección de Circulos

```
1 struct event {
2   double x; int t;
3   event(double xx, int tt) : x(xx), t(tt) {}
4   bool operator <(const event &o) const { return x < o.x; }
5 };
6 typedef vector<Circle> VC;
7 typedef vector<event> VE;
8 int n;
9 double cuenta(VE &v, double A,double B)
10 {
11   sort(v.begin(), v.end());
12   double res = 0.0, lx = ((v.empty())?0.0:v[0].x);
13   int contador = 0;
14   forn(i,sz(v))
15   { //interseccion de todos (contador == n), union de todos (contador > 0)
16     //conjunto de puntos cubierto por exacta k Circulos (contador == k)
17     if (contador == n) res += v[i].x - lx;
18     contador += v[i].t, lx = v[i].x;
19   }
20   return res;
21 }
22 // Primitiva de sqrt(r*r - x*x) como funcion double de una variable x.
23 inline double primitiva(double x,double r)
24 {
25   if (x >= r) return r*r*M_PI/4.0;
26   if (x <= -r) return -r*r*M_PI/4.0;
27   double raiz = sqrt(r*r-x*x);
28   return 0.5 * (x * raiz + r*r*atan(x/raiz));
29 }
30 double interCircle(VC &v)
31 {
32   vector<double> p; p.reserve(v.size() * (v.size() + 2));
33   forn(i,sz(v))  p.push_back(v[i].c.x + v[i].r), p.push_back(v[i].c.x - v[i].r);
34   forn(i,sz(v)) forn(j,i)
35   {
36     Circle &a = v[i], b = v[j];
37     double d = (a.c - b.c).norm();
38     if (fabs(a.r - b.r) < d && d < a.r + b.r)
39     {
40       double alfa = acos((sqr(a.r) + sqr(d) - sqr(b.r)) / (2.0 * d * a.r));
41       pto vec = (b.c - a.c) * (a.r / d);
42       p.pb((a.c + rotate(vec, alfa)).x), p.pb((a.c + rotate(vec, -alfa)).x);
```

```
43      }
44    }
45    sort(p.begin(), p.end());
46    double res = 0.0;
47    forn(i,sz(p)-1)
48    {
49      const double A = p[i], B = p[i+1];
50      VE ve; ve.reserve(2 * v.size());
51      forn(j,sz(v))
52      {
53        const Circle &c = v[j];
54        double arco = primitiva(B-c.c.x,c.r) - primitiva(A-c.c.x,c.r);
55        double base = c.c.y * (B-A);
56        ve.push_back(event(base + arco,-1));
57        ve.push_back(event(base - arco, 1));
58      }
59      res += cuenta(ve,A,B);
60    }
61    return res;
62 }
```

6.18   Rotar Matriz

```
1 //rotates matrix t 90 degrees clockwise
2 //using auxiliary matrix t2(faster)
3 void rotate()
4 {
5   forn(x, n) forn(y, n)
6     t2[n-y-1][x]=t[x][y];
7   memcpy(t, t2, sizeof(t));
8 }
```

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

$$a_0 T(n) + a_1 T(n-1) + ... + a_k T(n-k) = 0$$
$$p(x) = a_0 x^k + a_1 x^{k-1} + ... + a_k$$

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

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

```
1 //Chinese remainder theorem (special case): find z such that
2 //z % m1 = r1, z % m2 = r2.  Here, z is unique modulo M = lcm(m1, m2).
3 //Return (z, M).  On failure, M = -1.
4 ii chinese_remainder_theorem(int m1, int r1, int m2, int r2)
5 { //{xx,yy,d} son variables globales usadas en extendedEuclid
6   extendedEuclid(m1, m2);
7   if (r1%d != r2%d) return make_pair(0,-1);
8   return mp(sumMod(xx*r2*m1, yy*r1*m2, m1*m2) / d, m1*m2 / d);
9 }
10 //Chinese remainder theorem: find z such that z % m[i] = r[i] for all i.
```

```
11  //Note that the solution is unique modulo M = lcm_i (m[i]).
12  //Return (z, M). On failure, M = -1.
13  //Note that we do not require the a[i]'s to be relatively prime.
14  ii chinese_remainder_theorem(const vector<int> &m, const vector<int> &r)
15  {
16    ii ret=mp(r[0], m[0]);
17    forr(i,1,m.size())
18    {
19      ret=chinese_remainder_theorem(ret.snd, ret.fst, m[i], r[i]);
20      if (ret.snd==-1) break;
21    }
22    return ret;
23  }
```

7.4   GCD & LCM

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

7.5   Euclides Extendido

```
1  //ecuacion diofantica lineal
2  //sea d=gcd(a,b); la ecuacion a * x + b * y = c tiene soluciones enteras si
3  //d|c. La siguiente funcion nos sirve para esto. De forma general sera:
4  //x = x0 + (b/d)n      x0 = xx*c/d
5  //y = y0 - (a/d)n      y0 = yy*c/d
6  ll xx,yy,d;
7  void extendedEuclid(ll a, ll b)  //a * xx + b * yy = d
8  {
9    if (!b) {xx=1; yy=0; d=a; return;}
10    extendedEuclid (b,a%b);
11    ll x1=yy;
12    ll y1=xx-(a/b)*yy;
13    xx=x1; yy=y1;
14  }
```

7.6   Combinatoria

```
1  void cargarComb()//O(MAXN^2)
2  {
3    forn(i, MAXN+1) //comb[i][k]=i tomados de a k = i!/(k!*(i-k)!)
4    {
5      comb[0][i]=0;
6      comb[i][0]=comb[i][i]=1;
7      forr(k, 1, i) comb[i][k]=(comb[i-1][k-1]+comb[i-1][k]) %MOD;
8    }
```

```
9  }
10  ll lucas (ll n, ll k, int p)
11  { //Calcula (n,k)%p teniendo comb[p][p] precalculado.
12    ll aux = 1;
13    while (n + k)
14    {
15      aux = (aux * comb[n %p][k %p]) %p;
16      n/=p, k/=p;
17    }
18    return aux;
19  }
```

7.7   Exponenciación de Matrices y Fibonacci

```
1  #define SIZE 350
2  int NN;
3  void mul(double a[SIZE][SIZE], double b[SIZE][SIZE])
4  {
5    double res[SIZE][SIZE] = {{0}};
6    forn(i, NN) forn(j, NN) forn(k, NN) res[i][j]+=a[i][k]*b[k][j];
7    forn(i, NN) forn(j, NN) a[i][j]=res[i][j];
8  }
9  void powmat(double a[SIZE][SIZE], int n, double res[SIZE][SIZE])
10  {
11    forn(i, NN) forn(j, NN) res[i][j]=(i==j);
12    while(n)
13    {
14      if(n&1) mul(res, a), n--;
15      else mul(a, a), n/=2;
16    }
17  }
18
19  struct M22{      // |a b|
20    tipo a,b,c,d;// |c d| -- TIPO
21    M22 operator*(const M22 &p) const {
22    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};}
23  };
24  M22 operator^(const M22 &p, int n)
25  {//VER COMO SE PUEDE PONER DENTRO DEL STRUCT
26    if(!n) return (M22){1, 0, 0, 1};//identidad
27    M22 q=p^(n/2); q=q*q;
28    return n %2? p * q : q;
29  }
30
31  ll fibo(ll n)//calcula el fibonacci enesimo en O(logN)
32  {
33    M22 mat=(M22){0, 1, 1, 1}^n;
34    return mat.a*f0+mat.b*f1;//f0 y f1 son los valores iniciales
35  }
```

7.8   Operaciones Modulares

```
1  ll mulMod(ll a,ll b,ll m=MOD) //O(log b)
2  { //returns (a*b) %c, and minimize overfloor
3    ll x=0, y=a%m;
4    while(b>0)
5    {
6      if(b%2==1) x=(x+y)%m;
7      y=(y*2)%m;
8      b/=2;
9    }
10   return x%m;
11 }
12 ll expMod(ll b,ll e,ll m=MOD) //O(log b)
13 {
14   if(!e) return 1;
15   ll q=expMod(b,e/2,m);
16   q=mulMod(q,q,m);
17   return e%2? mulMod(b,q,m) : q;
18 }
19 ll sumMod(ll a,ll b,ll m=MOD)
20 {
21   a%=m;
22   b%=m;
23   if(a<0) a+=m;
24   if(b<0) b+=m;
25   return (a+b)%m;
26 }
27 ll difMod(ll a,ll b,ll m=MOD)
28 {
29   a%=m;
30   b%=m;
31   if(a<0) a+=m;
32   if(b<0) b+=m;
33   ll ret=a-b;
34   if(ret<0) ret+=m;
35   return ret;
36 }
37 ll divMod(ll a,ll b,ll m=MOD)
38 {
39   return mulMod(a,inverso(b),m);
40 }
```

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

```
1  #define MAXP 100000 //no necesariamente primo
2  int criba[MAXP+1];
3  void crearCriba()
4  {
5    int w[] = {4,2,4,2,4,6,2,6};
6    for(int p=25;p<=MAXP;p+=10) criba[p]=5;
7    for(int p=9;p<=MAXP;p+=6) criba[p]=3;
8    for(int p=4;p<=MAXP;p+=2) criba[p]=2;
9    for(int p=7,cur=0;p*p<=MAXP;p+=w[cur++&7]) if (!criba[p])
10     for(int j=p*p;j<=MAXP;j+=(p<<1)) if(!criba[j]) criba[j]=p;
11 }
12 vector<int> primos;
13 void buscarPrimos()
14 {
15   crearCriba();
16   forr (i,2,MAXP+1) if (!criba[i]) primos.push_back(i);
17 }
18
19 //factoriza bien numeros hasta MAXP^2
20 void fact(ll n,map<ll,ll> &f) //O (cant primos)
21 { //llamar a buscarPrimos antes
22   forall(p, primos){
23     while(!(n %*p))
24     {
25       f[*p]++;//divisor found
26       n/=*p;
27     }
28   }
29   if(n>1) f[n]++;
30 }
31
32 //factoriza bien numeros hasta MAXP
33 void fact2(ll n,map<ll,ll> &f) //O (lg n)
34 { //llamar a crearCriba antes
35   while (criba[n])
36   {
37     f[criba[n]]++;
38     n/=criba[n];
39   }
40   if(n>1) f[n]++;
41 }
42
43 //Usar asi: divisores(fac, divs, fac.begin()); NO ESTA ORDENADO
44 void divisores(map<ll,ll> &f,vector<ll> &divs,map<ll,ll>::iterator it,ll n
       =1)
45 {
46   if(it==f.begin()) divs.clear();
47   if(it==f.end())
48   {
```

7.10   Phollard's Rho

```cpp
      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;
}
```

```cpp
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)-1);//si mod no es primo(sacar a mano)
  return expMod(x, MOD-2);//si mod es primo
}
```

7.12   Guass-Jordan

```
// https://cp-algorithms.com/linear_algebra/linear-system-gauss.html
const double EPS = 1e-9;
const int INF = 2; // a value to indicate infinite solutions

int gauss (vector < vector<double> > a, vector<double> & ans) {
    int n = (int) a.size();
    int m = (int) a[0].size() - 1;

    vector<int> where (m, -1);
    for (int col=0, row=0; col<m && row<n; ++col) {
        int sel = row;
        for (int i=row; i<n; ++i)
            if (abs (a[i][col]) > abs (a[sel][col]))
                sel = i;
        if (abs (a[sel][col]) < EPS)
            continue;
        for (int i=col; i<=m; ++i)
            swap (a[sel][i], a[row][i]);
        where[col] = row;

        for (int i=0; i<n; ++i)
            if (i != row) {
                double c = a[i][col] / a[row][col];
                for (int j=col; j<=m; ++j)
                    a[i][j] -= a[row][j] * c;
            }
        ++row;
    }

    ans.assign (m, 0);
    for (int i=0; i<m; ++i)
        if (where[i] != -1)
            ans[i] = a[where[i]][m] / a[where[i]][i];
    for (int i=0; i<n; ++i) {
```

```
        double sum = 0;
        for (int j=0; j<m; ++j)
            sum += ans[j] * a[i][j];
        if (abs (sum - a[i][m]) > EPS)
            return 0;
    }

    for (int i=0; i<m; ++i)
        if (where[i] == -1)
            return INF;
    return 1;
}
```

7.13   Guass-Jordan modular

```
// inv -> modular inverse function
// disclaimer: not very well tested, but got AC on a problem with this
int gauss (vector < vector<int> > a, vector<int> & ans) {
    int n = (int) a.size();
    int m = (int) a[0].size() - 1;

    vector<int> where (m, -1);
    for (int col=0, row=0; col<m && row<n; ++col) {
        int sel = row;
        for (int i=row; i<n; ++i)
            if (a[i][col] > a[sel][col])
                sel = i;
        if (a[sel][col] == 0)
            continue;
        for (int i=col; i<=m; ++i)
            swap (a[sel][i], a[row][i]);
        where[col] = row;

        for (int i=0; i<n; ++i)
            if (i != row) {
                int c = (a[i][col] * inv(a[row][col]))%MOD;
                for (int j=col; j<=m; ++j)
                    a[i][j] = (a[i][j] - a[row][j]*c%MOD + MOD)%MOD;
            }
        ++row;
    }
  ans.clear();
    ans.rsz(m, 0);
    for (int i=0; i<m; ++i)
        if (where[i] != -1)
            ans[i] = (a[where[i]][m] * inv(a[where[i]][i]))%MOD;
    for (int i=0; i<n; ++i) {
        int sum = 0;
        for (int j=0; j<m; ++j)
            sum = (sum + ans[j] * a[i][j])%MOD;
        if ((sum - a[i][m] + MOD)%MOD != 0)
```

```
37          return 0;
38      }
39
40      for (int i=0; i<m; ++i)
41          if (where[i] == -1)
42              return INF;
43      return 1;
44 }
```

7.14   Guass-Jordan with bitset

```
1  // https://cp-algorithms.com/linear_algebra/linear-system-gauss.html
2  // special case of gauss_jordan_mod with mod=2, bitset for efficiency
3  int gauss (vector < bitset<N> > a, int n, int m, bitset<N> & ans) {
4      vector<int> where (m, -1);
5      for (int col=0, row=0; col<m && row<n; ++col) {
6          for (int i=row; i<n; ++i)
7              if (a[i][col]) {
8                  swap (a[i], a[row]);
9                  break;
10             }
11         if (! a[row][col])
12             continue;
13         where[col] = row;
14
15         for (int i=0; i<n; ++i)
16             if (i != row && a[i][col])
17                 a[i] ^= a[row];
18         ++row;
19     }
20     // The rest is very similar to the modular version
21 }
```

7.15   Fracciones

```
1  struct frac{
2    int p,q;
3    frac(int p=0,int q=1):p(p),q(q) {norm();}
4    void norm()
5    {
6      int a=gcd(q,p);
7      if(a) p/=a, q/=a;
8      else q=1;
9      if (q<0) q=-q, p=-p;
10   }
11   frac operator+(const frac& o)
12   {
13     int a=gcd(o.q,q);
```

```
14     return frac(p*(o.q/a)+o.p*(q/a),q*(o.q/a));
15   }
16   frac operator-(const frac& o)
17   {
18     int a=gcd(o.q,q);
19     return frac(p*(o.q/a)-o.p*(q/a),q*(o.q/a));
20   }
21   frac operator*(frac o)
22   {
23     int a=gcd(o.p,q), b=gcd(p,o.q);
24     return frac((p/b)*(o.p/a),(q/a)*(o.q/b));
25   }
26   frac operator/(frac o)
27   {
28     int a=gcd(o.q,q), b=gcd(p,o.p);
29     return frac((p/b)*(o.q/a),(q/a)*(o.p/b));
30   }
31   bool operator<(const frac &o) const{return p*o.q < o.p*q;}
32   bool operator==(frac o){return p==o.p&&q==o.q;}
33 };
```

7.16   Simpson

```
1  double integral(double a, double b, int n=10000) //O(n), n=cantdiv
2  {
3    double area=0, h=(b-a)/n, fa=f(a), fb;
4    forn(i, n)
5    {
6      fb=f(a+h*(i+1));
7      area+=fa+ 4*f(a+h*(i+0.5)) +fb, fa=fb;
8    }
9    return area*h/6.;
10 }
```

7.17   Tablas y cotas (Primos, Divisores, Factoriales, etc)

Factoriales

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

$\pi(10^1) = 4$; $\pi(10^2) = 25$; $\pi(10^3) = 168$; $\pi(10^4) = 1229$; $\pi(10^5) = 9592$

$\pi(10^6) = 78.498$; $\pi(10^7) = 664.579$; $\pi(10^8) = 5.761.455$; $\pi(10^9) = 50.847.534$

$\pi(10^{10}) = 455.052,511$; $\pi(10^{11}) = 4.118.054.813$; $\pi(10^{12}) = 37.607.912.018$

7.18   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

7.18.1   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

```
1  #define INF 1e9
2  int N;
3  #define MAX_V 250001
4  vector<ii> G[MAX_V];
5  //To add an edge use
6  #define add(a, b, w) G[a].pb(make_pair(w, b))
7  ll dijkstra(int s, int t){//O(|E| log |V|)
8    priority_queue<ii, vector<ii>, greater<ii> > Q;
9    vector<ll> dist(N, INF); vector<int> dad(N, -1);
10   Q.push(make_pair(0, s)); dist[s] = 0;
11   while(sz(Q)){
12     ii p = Q.top(); Q.pop();
13     if(p.snd == t) break;
14     forall(it, G[p.snd])
15       if(dist[p.snd]+it->first < dist[it->snd]){
16         dist[it->snd] = dist[p.snd] + it->fst;
17         dad[it->snd] = p.snd;
18         Q.push(make_pair(dist[it->snd], it->snd));  }
19   }
20   return dist[t];
21   if(dist[t]<INF)//path generator
22     for(int i=t; i!=-1; i=dad[i])
23       printf("%d%c", i, (i==s?'\n':' '));}
```

8.2   Bellman-Ford

```
1  //Mas lento que Dijsktra, pero maneja arcos con peso negativo
2  vector<ii> G[MAX_N];//ady. list with pairs (weight, dst)
3  int dist[MAX_N];
4  void bford(int src){//O(VE)
5    dist[src]=0;
6    forn(i, N-1) forn(j, N) if(dist[j]!=INF) forall(it, G[j])
7      dist[it->snd]=min(dist[it->snd], dist[j]+it->fst);
8  }
9
10 bool hasNegCycle(){
11   forn(j, N) if(dist[j]!=INF) forall(it, G[j])
12     if(dist[it->snd]>dist[j]+it->fst) return true;
13   //inside if: all points reachable from it->snd will have -INF distance(do
       bfs)
14   return false;
15 }
```

8.3   Floyd-Warshall

```
1  // Camino minimo en grafos dirigidos ponderados, en todas las parejas de
      nodos.
2  //G[i][j] contains weight of edge (i, j) or INF
3  //G[i][i]=0
4  int G[MAX_N][MAX_N];
5  void floyd(){//O(N^3)
6  forn(k, N) forn(i, N) if(G[i][k]!=INF) forn(j, N) if(G[k][j]!=INF)
7    G[i][j]=min(G[i][j], G[i][k]+G[k][j]);
8  }
9  bool inNegCycle(int v){
10   return G[v][v]<0;}
11 //checks if there's a neg. cycle in path from a to b
12 bool hasNegCycle(int a, int b){
13   forn(i, N) if(G[a][i]!=INF && G[i][i]<0 && G[i][b]!=INF)
14     return true;
15   return false;
16 }
```

8.4   Kruskal

```
1  struct UF{
2      void init(int n){}
3      void unir(int a, int v){}
4      int comp(int n){return 0;}
5  }uf;
6  vector<ii> G[MAXN];
7  int n;
8
9  struct Ar{int a,b,w;};
10 bool operator<(const Ar& a, const Ar &b){return a.w<b.w;}
11 vector<Ar> E;
12
13 // Minimun Spanning Tree in O(e log e)
14 ll kruskal(){
15     ll cost=0;
16     sort(E.begin(), E.end());//ordenar aristas de menor a mayor
17     uf.init(n);
18     forall(it, E){
19         if(uf.comp(it->a)!=uf.comp(it->b)){//si no estan conectados
20             uf.unir(it->a, it->b);//conectar
21             cost+=it->w;
22         }
23     }
24     return cost;
25 }
```

8.5   Prim

```
1  vector<ii> G[MAXN];
2  bool taken[MAXN];
3  priority_queue<ii, vector<ii>, greater<ii> > pq;//min heap
4  void process(int v){
5      taken[v]=true;
6      forall(e, G[v])
7          if(!taken[e->second]) pq.push(*e);
8  }
9  // Minimun Spanning Tree in O(n^2)
10 ll prim(){
11     zero(taken);
12     process(0);
13     ll cost=0;
14     while(sz(pq)){
15         ii e=pq.top(); pq.pop();
16         if(!taken[e.second]) cost+=e.first, process(e.second);
17     }
18     return cost;
19 }
```

8.6   Kosaraju SCC

Componente Fuertemente Conexa

```
1  #define MAXN 1000000
2  vector<int> G[MAXN],gt[MAXN]; //Limpiar si se corre mas de una vez
3  //nodos 0...N-1 ; componentes 0...cantcomp-1
4  int comp[MAXN],N,cantcomp,used[MAXN];
5  stack<int> pila;
6  void add(int a, int b){ G[a].pb(b);gt[b].pb(a);}
7  void dfs1(int nodo)
8  {
9    used[nodo]=1;
10   forall(it,G[nodo]) if(!used[*it]) dfs1(*it);
11   pila.push(nodo);
12 }
13 void dfs2(int nodo)
14 {
15   used[nodo]=2;
16   comp[nodo]=cantcomp-1;
17   forall(it,gt[nodo]) if(used[*it]!=2) dfs2(*it);
18 }
19 void kosaraju()
20 {
21   cantcomp=0;
22   memset(used,0,sizeof(used));
23   forn(i,N) if(!used[i]) dfs1(i);
24   while(!pila.empty())
25   {
```

```
26     if(used[pila.top()]!=2)
27     {
28         cantcomp++;
29         dfs2(pila.top());
30     }
31     pila.pop();
32   }
33 }
```

8.7   2-SAT + Tarjan SCC

```
1  //We have a vertex representing a var and other for his negation.
2  //Every edge stored in G represents an implication. To add an equation of
       the form a||b, use addor(a, b)
3  //MAX=max cant var, n=cant var
4  #define addor(a, b) (G[neg(a)].pb(b), G[neg(b)].pb(a))
5  vector<int> G[MAX*2];
6  //idx[i]=index assigned in the dfs
7  //lw[i]=lowest index(closer from the root) reachable from i
8  int lw[MAX*2], idx[MAX*2], qidx;
9  stack<int> q;
10 int qcmp, cmp[MAX*2];
11 //verdad[cmp[i]]=valor de la variable i
12 bool verdad[MAX*2+1];
13
14 int neg(int x) { return x>=n? x-n : x+n;}
15 void tjn(int v){
16   lw[v]=idx[v]=++qidx;
17   q.push(v), cmp[v]=-2;
18   forall(it, G[v]){
19     if(!idx[*it] || cmp[*it]==-2){
20       if(!idx[*it]) tjn(*it);
21       lw[v]=min(lw[v], lw[*it]);
22     }
23   }
24   if(lw[v]==idx[v]){
25     int x;
26     do{x=q.top(); q.pop(); cmp[x]=qcmp;}while(x!=v);
27     verdad[qcmp]=(cmp[neg(v)]<0);
28     qcmp++;
29   }
30 }
31 //remember to CLEAR G!!!
32 bool satisf(){//O(n)
33   memset(idx, 0, sizeof(idx)), qidx=0;
34   memset(cmp, -1, sizeof(cmp)), qcmp=0;
35   forn(i, n){
36     if(!idx[i]) tjn(i);
37     if(!idx[neg(i)]) tjn(neg(i));
38   }
39   forn(i, n) if(cmp[i]==cmp[neg(i)]) return false;
```

```
40    return true;
41  }
```

8.8   Puntos de Articulación

```
1  int N;
2  vector<int> G[1000000];
3  //V[i]=node number(if visited), L[i]= lowest V[i] reachable from i
4  int qV, V[1000000], L[1000000], P[1000000];
5  void dfs(int v, int f){
6    L[v]=V[v]=++qV;
7    forall(it, G[v])
8      if(!V[*it]){
9        dfs(*it, v);
10       L[v] = min(L[v], L[*it]);
11       P[v]+= L[*it]>=V[v];
12     }
13     else if(*it!=f)
14       L[v]=min(L[v], V[*it]);
15 }
16 int cantart(){ //O(n)
17   qV=0;
18   zero(V), zero(P);
19   dfs(1, 0); P[1]--;
20   int q=0;
21   forn(i, N) if(P[i]) q++;
22 return q;
23 }
```

8.9   Componentes Biconexas y Puentes

```
1  vector<int> G[MAXN];
2
3  struct edge{
4    int u,v, comp;
5    bool bridge;
6  };
7  vector<edge> e;
8  void addEdge(int u, int v)
9  {
10   G[u].pb(sz(e)), G[v].pb(sz(e));
11   e.pb((edge){u,v,-1,false});
12 }
13 //d[i]=id de la dfs
14 //b[i]=lowest id reachable from i
15 int d[MAXN], b[MAXN], t;
16 int nbc;//cant componentes
17 int comp[MAXN];//comp[i]=cant comp biconexas a la cual pertenece i
18 void initDfs(int n)
19 {
20   zero(G), zero(comp);
21   e.clear();
22   forn(i,n) d[i]=-1;
23   nbc = t = 0;
```

```
24 }
25 stack<int> st;
26 void dfs(int u,int pe)  //O(n + m)
27 {
28   b[u]=d[u]=t++;
29   comp[u]=(pe!=-1);
30   forall(ne,G[u]) if(*ne!=pe)
31   {
32     int v=e[*ne].u ^ e[*ne].v ^ u;
33     if(d[v]==-1)
34     {
35       st.push(*ne);
36       dfs(v,*ne);
37       if(b[v]>d[u]) e[*ne].bridge=true; // bridge
38       if(b[v]>=d[u])   // art
39       {
40         int last;
41         do
42         {
43           las=st.top(); st.pop();
44           e[last].comp=nbc;
45         }while(last!=*ne);
46         nbc++;
47         comp[u]++;
48       }
49       b[u]=min(b[u],b[v]);
50     }
51     else if(d[v]<d[u])   // back edge
52     {
53       st.push(*ne);
54       b[u]=min(b[u], d[v]);
55     }
56   }
57 }
```

8.10   Least Common Ancestor + Climb

```
1  const int MAXN=100001, LOGN=20;
2  //f[v][k] holds the 2^k father of v
3  //L[v] holds the level of v
4  int N, f[MAXN][LOGN], L[MAXN];
5  //call before build:
6  void dfs(int v, int fa=-1, int lvl=0){//generate required data
7    f[v][0]=fa, L[v]=lvl;
8    forall(it, G[v])if(*it!=fa) dfs(*it, v, lvl+1); }
9  void build(){//f[i][0] must be filled previously, O(nlgn)
10   forn(k, LOGN-1) forn(i, N) f[i][k+1]=f[f[i][k]][k];}
11 #define lg(x) (31-__builtin_clz(x))//=floor(log2(x))
12 int climb(int a, int d){//O(lgn)
13   if(!d) return a;
14   dforn(i, lg(L[a])+1) if(1<<i<=d) a=f[a][i], d-=1<<i;
```

```
15     return a;}
16 int lca(int a, int b){//O(lgn)
17   if(L[a]<L[b]) swap(a, b);
18   a=climb(a, L[a]-L[b]);
19   if(a==b) return a;
20   dforn(i, lg(L[a])+1) if(f[a][i]!=f[b][i]) a=f[a][i], b=f[b][i];
21   return f[a][0]; }
22 int dist(int a, int b) {//returns distance between nodes
23   return L[a]+L[b]-2*L[lca(a, b)];}
```

8.11   Heavy Light Decomposition

```
1  vector<int> g[MAXN];
2  int wg[MAXN],dad[MAXN],dep[MAXN]; // weight,father,depth
3  void dfs1(int x){
4    wg[x]=1;
5    for(int y:g[x])if(y!=dad[x]){
6      dad[y]=x;dep[y]=dep[x]+1;dfs1(y);
7      wg[x]+=wg[y];
8    }
9  }
10 int curpos,pos[MAXN],head[MAXN];
11 void hld(int x, int c){
12   if(c<0)c=x;
13   pos[x]=curpos++;head[x]=c;
14   int mx=-1;
15   for(int y:g[x])if(y!=dad[x]&&(mx<0||wg[mx]<wg[y]))mx=y;
16   if(mx>=0)hld(mx,c);
17   for(int y:g[x])if(y!=mx&&y!=dad[x])hld(y,-1);
18 }
19 void hld_init(){dad[0]=-1;dep[0]=0;dfs1(0);curpos=0;hld(0,-1);}
20 int query(int x, int y, RMQ &st){
21   int r=neutro;
22   while(head[x]!=head[y]){
23     if(dep[head[x]]>dep[head[y]])swap(x,y);
24     r=operacion(r,st.get(pos[head[y]],pos[y]+1));
25     y=dad[head[y]];
26   }
27   if(dep[x]>dep[y])swap(x,y); // now x is lca
28   r=operacion(r,st.get(pos[x]+1,pos[y]+1));
29   return r;
30 }
31 // for updating: rmq.upd(pos[x],v);
32 // queries on edges: - assign values of edges to "child" node
33 //                   - change pos[x] to pos[x]+1 in query (line 28)
```

8.12   Centroid Decomposition

```
1  vector<int> G[MAXN];
2  bool taken[MAXN];//poner todos en FALSE al principio!!
3  int padre[MAXN];//padre de cada nodo en el centroid tree
4
5  int szt[MAXN];
6  void calcsz(int v, int p) {
7    szt[v] = 1;
8    forall(it,G[v]) if (*it!=p && !taken[*it])
9      calcsz(*it,v), szt[v]+=szt[*it];
10 }
11 void centroid(int v=0, int f=-1, int lvl=0, int tam=-1) {//O(nlogn)
12   if(tam==-1) calcsz(v, -1), tam=szt[v];
13   forall(it, G[v]) if(!taken[*it] && szt[*it]>=tam/2)
14     {szt[v]=0; centroid(*it, f, lvl, tam); return;}
15   taken[v]=true;
16   padre[v]=f;
17   forall(it, G[v]) if(!taken[*it])
18     centroid(*it, v, lvl+1, -1);
19 }
```

8.13   Ciclo Euleriano - v1

```
1  int n,m,ars[MAXE], eq;
2  vector<int> G[MAXN];//fill G,n,m,ars,eq
3  list<int> path;
4  int used[MAXN];
5  bool usede[MAXE];
6  queue<list<int>::iterator> q;
7  int get(int v){
8    while(used[v]<sz(G[v]) && usede[ G[v][used[v]] ]) used[v]++;
9    return used[v];
10 }
11 void explore(int v, int r, list<int>::iterator it){
12   int ar=G[v][get(v)]; int u=v^ars[ar];
13   usede[ar]=true;
14   list<int>::iterator it2=path.insert(it, u);
15   if(u!=r) explore(u, r, it2);
16   if(get(v)<sz(G[v])) q.push(it);
17 }
18 void euler(){
19   zero(used), zero(usede);
20   path.clear();
21   q=queue<list<int>::iterator>();
22   path.push_back(0); q.push(path.begin());
23   while(sz(q)){
24     list<int>::iterator it=q.front(); q.pop();
25     if(used[*it]<sz(G[*it])) explore(*it, *it, it);
26   }
27   reverse(path.begin(), path.end());
28 }
```

```
29 void addEdge(int u, int v){
30   G[u].pb(eq), G[v].pb(eq);
31   ars[eq++]=u^v;
32 }
```

8.14   Ciclo Euleriano - v2

```
1  //In a connected graph where all the nodes have even degree
2  //finds a path that start and finish in the same node (SRC)
3  // and uses every edge once.
4  struct EulerianTour {
5    int N, M = 0, odd = 0;
6    vector<vector<ii>> E;
7    vector<int> deg;
8    vector<bool> vis;
9    EulerianTour(int N) : N(N), E(N), deg(N), vis(N) {}
10   void add_edge(int u, int v) {
11     int V[2] = {u, v};
12     for (auto t : {0, 1}) {
13       int v = V[t];
14       E[v].emplace_back(V[t ^ 1], M << 1 | t);
15       deg[v] += 1;
16       odd += (deg[v] % 2 ? +1 : -1);
17     }
18     ++M;
19   }
20   // returns eulerian tour by vertices and edges (reversed if first bit is
          1)
21   pair<vector<int>, vector<int>> find(int src) {
22     //run for every component if graph isn't connected
23     assert(odd == 0);
24     auto d = deg;
25     vector<bool> dead(M, false);
26     vector<int> ptr(N, 0), p, e;
27     stack<ii> stk;
28     stk.emplace(src, -1);
29     while (!stk.empty()) {
30       ii aux = stk.top();
31       int u=aux.fst, i = aux.snd;
32       vis[u] = true;
33       if (d[u] == 0) {
34         stk.pop();
35         p.push_back(u);
36         if (i != -1) e.push_back(i);
37       } else {
38         for (int& l = ptr[u]; l < deg[u]; ++l) {
39           ii aux2 = E[u][l];
40           int v=aux2.fst, j=aux2.snd;
41           if (!dead[j >> 1]) {
42             stk.emplace(v, j);
43             --d[u], --d[v], dead[j >> 1] = true;
```

```
44              break;
45            }
46          }
47        }
48      }
49      return {p, e};
50    }
51 };
```

8.15   Diametro Árbol

```
1  vector<int> G[MAXN]; int n,m,p[MAXN],d[MAXN],d2[MAXN];
2  int bfs(int r, int *d) {
3    queue<int> q;
4    d[r]=0; q.push(r);
5    int v;
6    while(sz(q)) { v=q.front(); q.pop();
7      forall(it,G[v]) if (d[*it]==-1)
8        d[*it]=d[v]+1, p[*it]=v, q.push(*it);
9    }
10   return v;//ultimo nodo visitado
11 }
12 vector<int> diams; vector<ii> centros;
13 void diametros(){
14   memset(d,-1,sizeof(d));
15   memset(d2,-1,sizeof(d2));
16   diams.clear(), centros.clear();
17   forn(i, n) if(d[i]==-1){
18     int v,c;
19     c=v=bfs(bfs(i, d2), d);
20     forn(_,d[v]/2) c=p[c];
21     diams.pb(d[v]);
22     if(d[v]&1) centros.pb(ii(c, p[c]));
23     else centros.pb(ii(c, c));
24   }
25 }
```

8.16   Hungarian

```
1  typedef long double td; typedef vector<int> vi; typedef vector<td> vd;
2  const td INF=1e100;//for maximum set INF to 0, and negate costs
3  bool zero(td x){return fabs(x)<1e-9;}//change to x==0, for ints/ll
4  struct Hungarian{
5    int n; vector<vd> cs; vi L, R;
6    Hungarian(int N, int M):n(max(N,M)),cs(n,vd(n)),L(n),R(n){
7      forn(x,N)forn(y,M)cs[x][y]=INF;
8    }
9    void set(int x,int y,td c){cs[x][y]=c;}
```

```
10   td assign() {
11     int mat = 0; vd ds(n), u(n), v(n); vi dad(n), sn(n);
12     forn(i,n)u[i]=*min_element(ALL(cs[i]));
13     forn(j,n){v[j]=cs[0][j]-u[0];forr(i,1,n)v[j]=min(v[j],cs[i][j]-u[i]);}
14     L=R=vi(n, -1);
15     forn(i,n)forn(j,n)
16       if(R[j]==-1&&zero(cs[i][j]-u[i]-v[j])){L[i]=j;R[j]=i;mat++;break;}
17     for(;mat<n;mat++){
18       int s=0, j=0, i;
19       while(L[s] != -1)s++;
20       fill(ALL(dad),-1);fill(ALL(sn),0);
21       forn(k,n)ds[k]=cs[s][k]-u[s]-v[k];
22       for(;;){
23         j = -1;
24         forn(k,n)if(!sn[k]&&(j==-1||ds[k]<ds[j]))j=k;
25         sn[j] = 1; i = R[j];
26         if(i == -1) break;
27         forn(k,n)if(!sn[k]){
28           auto new_ds=ds[j]+cs[i][k]-u[i]-v[k];
29           if(ds[k] > new_ds){ds[k]=new_ds;dad[k]=j;}
30         }
31       }
32       forn(k,n)if(k!=j&&sn[k]){auto w=ds[k]-ds[j];v[k]+=w,u[R[k]]-=w;}
33       u[s] += ds[j];
34       while(dad[j]>=0){int d = dad[j];R[j]=R[d];L[R[j]]=j;j=d;}
35       R[j]=s;L[s]=j;
36     }
37     td value=0;forn(i,n)value+=cs[i][L[i]];
38     return value;
39   }
40 };
```

8.17   Dynamic Connectivity

```
1  struct UnionFind {
2    int n, comp;
3    vector<int> pre,si,c;
4    UnionFind(int n=0):n(n), comp(n), pre(n), si(n, 1) {
5      forn(i,n) pre[i] = i; }
6    int find(int u){return u==pre[u]?u:find(pre[u]);}
7    bool merge(int u, int v)
8    {
9      if((u=find(u))==(v=find(v))) return false;
10     if(si[u]<si[v]) swap(u, v);
11     si[u]+=si[v], pre[v]=u, comp--, c.pb(v);
12     return true;
13   }
14   int snap(){return sz(c);}
15   void rollback(int snap)
16   {
17     while(sz(c)>snap)
18     {
19       int v = c.back(); c.pop_back();
20       si[pre[v]] -= si[v], pre[v] = v, comp++;
21     }
22   }
23 };
24 enum {ADD,DEL,QUERY};
25 struct Query {int type,u,v;};
26 struct DynCon{//bidirectional graphs; create vble as DynCon name(cant_nodos)
27   vector<Query> q;
28   UnionFind dsu;
29   vector<int> match,res;
30   map<ii,int> last;//se puede no usar cuando hay identificador para cada
           arista (mejora poco)
31   DynCon(int n=0):dsu(n){}
32   void add(int u, int v) //to add an edge
33   {
34     if(u>v) swap(u,v);
35     q.pb((Query){ADD, u, v}), match.pb(-1);
36     last[ii(u,v)] = sz(q)-1;
37   }
38   void remove(int u, int v) //to remove an edge
39   {
40     if(u>v) swap(u,v);
41     q.pb((Query){DEL, u, v});
42     int prev = last[ii(u,v)];
43     match[prev] = sz(q)-1;
44     match.pb(prev);
45   }
46   void query() //to add a question (query) type of query
47   {
48     q.pb((Query){QUERY, -1, -1}), match.pb(-1);
49   }
50   void process() //call this to process queries in the order of q
51   {
52     forn(i,sz(q)) if (q[i].type == ADD && match[i] == -1) match[i] = sz(q);
53     go(0,sz(q));
54   }
55   void go(int l, int r)
56   {
57     if(l+1==r)
58     {
59       if (q[l].type == QUERY)//Aqui responder la query usando el dsu!
60         res.pb(dsu.comp);//aqui query=cantidad de componentes conexas
61       return;
62     }
63     int s=dsu.snap(), m = (l+r) / 2;
64     forr(i,m,r) if(match[i]!=-1 && match[i]<l) dsu.merge(q[i].u, q[i].v);
65     go(l,m);
66     dsu.rollback(s);
67     s = dsu.snap();
68     forr(i,l,m) if(match[i]!=-1 && match[i]>=r) dsu.merge(q[i].u, q[i].v);
69     go(m,r);
70     dsu.rollback(s);
71   }
72 };
```

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)
{
```

```
20    int amt = min(excess[a], ll(G[a][b]));
21    if(height[a] <= height[b] || amt == 0) return;
22    G[a][b]-=amt, G[b][a]+=amt;
23    excess[b] += amt, excess[a] -= amt;
24    enqueue(b);
25  }
26  void gap(int k)
27  {
28    forn(v, N)
29    {
30      if (height[v] < k) continue;
31      cuenta[height[v]]--;
32      height[v] = max(height[v], N+1);
33      cuenta[height[v]]++;
34      enqueue(v);
35    }
36  }
37  void relabel(int v)
38  {
39    cuenta[height[v]]--;
40    height[v] = 2*N;
41    forall(it, G[v])
42    if(it->snd) height[v] = min(height[v], height[it->fst] + 1);
43    cuenta[height[v]]++;
44    enqueue(v);
45  }
46  ll maxflow() //O(V^3)
47  {
48    zero(height), zero(active), zero(cuenta), zero(excess);
49    cuenta[0]=N-1; cuenta[N]=1;
50    height[SRC] = N;
51    active[SRC] = active[SNK] = true;
52    forall(it, G[SRC])
53    {
54      excess[SRC] += it->snd;
55      push(SRC, it->fst);
56    }
57    while(sz(Q))
58    {
59      int v = Q.front(); Q.pop();
60      active[v]=false;
61      forall(it, G[v]) push(v, it->fst);
62      if(excess[v] > 0)
63      cuenta[height[v]] == 1? gap(height[v]):relabel(v);
64    }
65    ll mf=0;
66    forall(it, G[SRC]) mf+=G[it->fst][SRC];
67    return mf;
68  }
```

9.4 Dinic

```
1   struct Edge {
2     int u, v;
3     ll cap, flow;
4     Edge() {}
5     Edge(int u, int v, ll cap): u(u), v(v), cap(cap), flow(0) {}
6   };
7   struct Dinic {
8     int N;
9     vector<Edge> E;
10    vector<vector<int>> g;
11    vector<int> d, pt;
12    Dinic(int N): N(N), E(0), g(N), d(N), pt(N) {} //clear and init
13    void addEdge(int u, int v, ll cap)
14    {
15      if (u != v)
16      {
17        E.emplace_back(Edge(u, v, cap));
18        g[u].emplace_back(E.size() - 1);
19        E.emplace_back(Edge(v, u, 0));
20        g[v].emplace_back(E.size() - 1);
21      }
22    }
23    bool BFS(int S, int T)
24    {
25      queue<int> q({S});
26      fill(d.begin(), d.end(), N + 1);
27      d[S] = 0;
28      while(!q.empty())
29      {
30        int u = q.front(); q.pop();
31        if (u == T) break;
32        for (int k: g[u])
33        {
34          Edge &e = E[k];
35          if (e.flow < e.cap && d[e.v] > d[e.u] + 1)
36          {
37            d[e.v] = d[e.u] + 1;
38            q.emplace(e.v);
39          }
40        }
41      }
42      return d[T] != N + 1;
43    }
44    ll DFS(int u, int T, ll flow = -1)
45    {
46      if (u == T || flow == 0) return flow;
47      for (int &i = pt[u]; i < g[u].size(); ++i)
48      {
49        Edge &e = E[g[u][i]];
50        Edge &oe = E[g[u][i]^1];
51        if (d[e.v] == d[e.u] + 1)
```

```
52        {
53          ll amt = e.cap - e.flow;
54          if (flow != -1 && amt > flow) amt = flow;
55          if (ll pushed = DFS(e.v, T, amt))
56          {
57            e.flow += pushed;
58            oe.flow -= pushed;
59            return pushed;
60          }
61        }
62      }
63      return 0;
64    }
65    ll maxFlow(int S,int T) //O(V^2*E), unit nets: O(sqrt(V)*E)
66    {
67      ll total = 0;
68      while(BFS(S, T))
69      {
70        fill(pt.begin(), pt.end(), 0);
71        while (ll flow = DFS(S, T)) total += flow;
72      }
73      return total;
74    }
75 };
```

9.5   Min cost - Max flow

```
 1 const int MAXN=10000;
 2 typedef ll tf;
 3 typedef ll tc;
 4 const tf INFFLUJO = 1e14;
 5 const tc INFCOSTO = 1e14;
 6 struct edge {
 7   int u, v;
 8   tf cap, flow;
 9   tc cost;
10   tf rem() { return cap - flow; }
11 };
12 int nodes; //numero de nodos
13 vector<int> G[MAXN]; // limpiar!
14 vector<edge> e;  // limpiar!
15 void addEdge(int u, int v, tf cap, tc cost) {
16   G[u].pb(sz(e)); e.pb((edge){u,v,cap,0,cost});
17   G[v].pb(sz(e)); e.pb((edge){v,u,0,0,-cost});
18 }
19 tc dist[MAXN], mnCost;
20 int pre[MAXN];
21 tf cap[MAXN], mxFlow;
22 //tf wantedFlow; //For fixed flow instead of max
23 bool in_queue[MAXN];
24 void flow(int s, int t) {// O(n^2 * m^2)
```

```
25   zero(in_queue);
26   mxFlow=mnCost=0;
27   while(1) {
28     fill(dist, dist+nodes, INFCOSTO); dist[s] = 0;
29     memset(pre, -1, sizeof(pre)); pre[s]=0;
30     zero(cap); cap[s] = INFFLUJO;
31     queue<int> q; q.push(s); in_queue[s]=1;
32     while(sz(q)) {//Fast bellman-ford
33       int u=q.front(); q.pop(); in_queue[u]=0;
34       for(auto it:G[u]) {
35         edge &E = e[it];
36         if(E.rem() && dist[E.v] > dist[u] + E.cost + 1e-9) {// ojo EPS
37           dist[E.v]=dist[u]+E.cost;
38           pre[E.v] = it;
39           cap[E.v] = min(cap[u], E.rem());
40           if(!in_queue[E.v]) q.push(E.v), in_queue[E.v]=1;
41         }
42       }
43     }
44     if (pre[t] == -1) break;
45     tf flow = cap[t];
46     //flow = min(flow, wantedFlow-mxFlow) //For fixed flow
47     mxFlow += flow;
48     mnCost += flow*dist[t];
49     for (int v = t; v != s; v = e[pre[v]].u) {
50       e[pre[v]].flow += flow;
51       e[pre[v]^1].flow -= flow;
52     }
53     //if(mxFlow == wantedFlow) break; //For fixed flow
54   }
55 }
```

9.6   Matching

```
 1 vector<int> g[MAXN]; // [0,n)->[0,m)
 2 int n, m;
 3 int mat[MAXM]; bool vis[MAXN];
 4 int match(int x) {
 5   if(vis[x]) return 0;
 6   vis[x] = true;
 7   for(int y:g[x])if(mat[y]<0||match(mat[y])){mat[y]=x;return 1;}
 8   return 0;
 9 }
10 vector<pair<int,int> > max_matching() { //O(V^2 * E)
11   vector<pair<int,int> > r;
12   memset(mat,-1,sizeof(mat));
13   forn(i,n) memset(vis,false,sizeof(vis)), match(i);
14   forn(i,m) if(mat[i]>=0) r.pb({mat[i],i});
15   return r;
16 }
```

9.7   Matching - Hopcroft Karp

```cpp
vector<int> g[MAXN]; // [0,n)->[0,m)
int n,m;
int mt[MAXN],mt2[MAXN],ds[MAXN];
bool bfs(){
  queue<int> q;
  memset(ds,-1,sizeof(ds));
  forn(i,n)if(mt2[i]<0)ds[i]=0,q.push(i);
  bool r=false;
  while(!q.empty()){
    int x=q.front();q.pop();
    for(int y:g[x]){
      if(mt[y]>=0&&ds[mt[y]]<0)ds[mt[y]]=ds[x]+1,q.push(mt[y]);
      else if(mt[y]<0)r=true;
    }
  }
  return r;
}
bool dfs(int x){
  for(int y:g[x])if(mt[y]<0||ds[mt[y]]==ds[x]+1&&dfs(mt[y])){
    mt[y]=x;mt2[x]=y;
    return true;
  }
  ds[x]=1<<30;
  return false;
}
int mm(){ //O(sqrt(V)*E)
  int r=0;
  memset(mt,-1,sizeof(mt));memset(mt2,-1,sizeof(mt2));
  while(bfs()){
    forn(i,n)if(mt2[i]<0)r+=dfs(i);
  }
  return r;
}
```

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 $O(!N \cdot N^2)$.

```cpp
#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 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);
}
```

11.4 Comparación de Double

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

11.5 Limites

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

// double inf
const double DINF=numeric_limits<double>::infinity();
```

11.6 Iterar subconjuntos

```
// Iterate over non empty subsets of bitmask
for(int s=m;s;s=(s-1)&m) // Decreasing order
for (int s=0;s=s-m&m;)   // Increasing order
```

11.7 Operaciones de bits

```
// Return the numbers the numbers of 1-bit in x
int __builtin_popcount (unsigned int x)
// Returns the number of trailing 0-bits in x. x=0 is undefined.
int __builtin_ctz (unsigned int x)
// Returns the number of leading 0-bits in x. x=0 is undefined.
int __builtin_clz (unsigned int x)
// x of type long long just add 'll' at the end of the function.
int __builtin_popcountll (unsigned long long x)
// Get the value of the least significant bit that is one.
v=(x&(-x))
```

11.8 Comparator for set

```
// Custom comparator for set/map
struct comp {
  bool operator()(const double& a, const double& b) const {
    return a+EPS<b;}
};
set<double,comp> w; // or map<double,int,comp>
```

11.9   Pragma

```
1  #pragma GCC optimize("O3,unroll-loops")
2  #pragma GCC target("avx2,bmi,bmi2,lzcnt,popcnt")
```

11.10   Randomization

```
1  // declaration (mt19937_64 for 64-bits version)
2  mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
3  // usage
4  rng()%k // random value [0,k)
5  shuffle(v.begin(), v.end(), rng); // vector random shuffle
```

11.11   Python utils

```
1  import sys, math
2  input = sys.stdin.readline
3
4  ########### ---- Input Functions ---- ###########
5  def inp():
6      return(int(input()))
7  def inlt():
8      return(list(map(int,input().split())))
9  def insr():
10     s = input()
11     return(list(s[:len(s) - 1]))
12 def invr():
13     return(map(float,input().split()))
14
15
16 n, k = inlt()
17 intpart = 0
18 while intpart*intpart <= n:
19   intpart += 1
20 intpart -= 1
21
22 if(k == 0):
23   print(intpart)
24 else:
25   L = 0
26   R = 10**k-1
27   aux = 10**k
28
29   while(L < R):
30     M = (L+R+1)//2
31     if intpart**2 * aux**2 + M**2 + 2*intpart*M*aux <= n * aux**2:
32       L = M
33     else:
```

```
34       R = M-1
35
36  decpart = str(L)
37  while(len(decpart) < k):
38    decpart = '0'+decpart
39  print(f"{intpart}.{decpart}")
```

11.12   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	$T = i + [f1, l1) . [f2, \dots)$
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.