# Cartilla de código piola

UTN FRSF - El Rejunte

2017

## Índice

# 1. C/C++

## 1.1. I/O

### 1.1.1. scanf Format Strings

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

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

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

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

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

### 1.1.2. printf Format Strings

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

| specifier | Descripción | Ejemplo |
|---|---|---|
| d or i | Entero decimal con signo | 392 |
| u | Entero decimal sin signo | 7235 |
| o | Entero octal sin signo | 610 |
| x | Entero hexadecimal sin signo | 7fa |
| X | Entero hexadecimal sin signo (mayúsculas) | 7FA |
| f | Decimal punto flotante (minúsculas) | 392.65 |
| F | Decimal punto flotante (mayúsculas) | 392.65 |
| e | Notación científica (mantisa/exponente), (minúsculas) | 3.9265e+2 |
| E | Notación científica (mantisa/exponente), (mayúsculas) | 3.9265E+2 |
| g | Utilizar la representaciíon más corta: %e ó %f | 392.65 |
| G | Utilizar la representaciíon 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.<br>Para a, A, e, E, f, F: número de dígitos a imprimir después de la coma (default 6).<br>Para g, G: Número máximo de cifras significativas a imprimir.<br>Para s: Número máximo de caracteres a imprimir. Trunca. |
| .* | No se especifica precision pero se agrega un argumento entero precediendo al argumento a ser formateado. |

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

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

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

# 2. Template del Rejunte

```cpp
#include <bits/stdc++.h>
#define sqr(a) ((a)*(a))
#define rsz resize
#define forr(i,a,b) for(int i=(a);i<(b);i++)
#define forn(i,n) forr(i,0,n)
#define dforn(i,n) for(int i=n-1;i>=0;i--)
#define forall(it,v) for(auto it=v.begin();it!=v.end();it++)
#define sz(c) ((int)c.size())
#define zero(v) memset(v, 0, sizeof(v))
#define pb push_back
#define mp make_pair
#define lb lower_bound
#define ub upper_bound
#define fst first
#define snd second
#define PI 3.14159265358979323846

using namespace std;

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

int main()
{
  //freopen("input","r",stdin);
  //freopen("output","w",stdout);
```

```
  ios::sync_with_stdio(false);
  cin.tie(NULL);
  cout.tie(NULL);
  return 0;
}
```

# 3. Estructuras de datos

## 3.1. Set Mejorado

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

## 3.2. Union Find

```
struct UnionFind{
  vector<int> f,setSize; //the array f contains the parent of each node
  int cantSets;
  void init(int n)
  {
    f.clear(); setSize.clear();
    cantSets=n;
    f.rsz(n,-1);
    setSize.rsz(n,1);
  }
  int comp(int x){return (f[x]==-1?x:f[x]=comp(f[x]));}//O(1)
  bool join(int i,int j)
  {
    bool con=comp(i)==comp(j);
    if(!con)
    {
      cantSets--;
      f[comp(i)]=comp(j);
      setSize[comp(j)]+=setSize[comp(i)];
      setSize[comp(i)]=setSize[comp(j)]; //no suma, solo asigna
    }
    return con;
  }
};
```

## 3.3. Hash Table

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

## 3.4. RMQ

### 3.4.1. RMQ (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().

```
struct RMQ{
  #define LVL 10
  tipo vec[LVL][1<<(LVL+1)];
  tipo &operator[](int p){return vec[0][p];}
  tipo get(int i, int j) {//intervalo [i,j)
    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)]);
  }};
```

### 3.4.2. RMQ (dynamic)

```
//Dado un arreglo y una operacion asociativa con neutro, get(i, j) opera
    sobre el rango [i, j).
#define MAXN 100000
#define operacion(x, y) max(x, y)
```

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

### 3.4.3. RMQ (lazy)

```
//Dado un arreglo y una operacion asociativa con neutro, get(i, j) opera
    sobre el rango [i, j).
typedef int Elem;//Elem de los elementos del arreglo
typedef int Alt;//Elem de la alteracion
#define operacion(x,y) x+y
const Elem neutro=0; const Alt neutro2=0;
#define MAXN 100000//Cambiar segun el N del problema
struct RMQ{
  int sz;
  Elem t[4*MAXN];
  Alt dirty[4*MAXN];//las alteraciones pueden ser de distinto Elem
  Elem &operator[](int p){return t[sz+p];}
  void init(int n){//O(nlgn)
    sz = 1 << (32-__builtin_clz(n));
    forn(i, 2*sz) t[i]=neutro;
    forn(i, 2*sz) dirty[i]=neutro2;
  }
  void push(int n, int a, int b){//propaga el dirty a sus hijos
    if(dirty[n]!=0){
      t[n]+=dirty[n]*(b-a);//altera el nodo
      if(n<sz){
        dirty[2*n]+=dirty[n];
        dirty[2*n+1]+=dirty[n];
      }
      dirty[n]=0;
    }
  }
  Elem get(int i, int j, int n, int a, int b){//O(lgn)
    if(j<=a || i>=b) return neutro;
    push(n, a, b);//corrige el valor antes de usarlo
    if(i<=a && b<=j) return t[n];
    int c=(a+b)/2;
    return operacion(get(i, j, 2*n, a, c), get(i, j, 2*n+1, c, b));
  }
  Elem get(int i, int j){return get(i,j,1,0,sz);}
  //altera los valores en [i, j) con una alteracion de val
  void alterar(Alt val, int i, int j, int n, int a, int b){//O(lgn)
    push(n, a, b);
    if(j<=a || i>=b) return;
    if(i<=a && b<=j){
      dirty[n]+=val;
      push(n, a, b);
      return;
    }
    int c=(a+b)/2;
    alterar(val, i, j, 2*n, a, c), alterar(val, i, j, 2*n+1, c, b);
    t[n]=operacion(t[2*n], t[2*n+1]);//por esto es el push de arriba
  }
  void alterar(Alt val, int i, int j){alterar(val,i,j,1,0,sz);}
}rmq;
```

### 3.4.4. RMQ (persistente)

```
typedef int tipo;
tipo oper(const tipo &a, const tipo &b){
    return a+b;
}
struct node{
  tipo v; node *l,*r;
  node(tipo v):v(v), l(NULL), r(NULL) {}
    node(node *l, node *r) : l(l), r(r){
        if(!l) v=r->v;
        else if(!r) v=l->v;
        else v=oper(l->v, r->v);
    }
};
node *build (tipo *a, int tl, int tr) {//modificar para que tome tipo a
  if (tl+1==tr) return new node(a[tl]);
  int tm=(tl + tr)>>1;
  return new node(build(a, tl, tm), build(a, tm, tr));
}
```

```
node *update(int pos, int new_val, node *t, int tl, int tr){
  if (tl+1==tr) return new node(new_val);
  int tm=(tl+tr)>>1;
  if(pos < tm) return new node(update(pos, new_val, t->l, tl, tm), t->r);
  else return new node(t->l, update(pos, new_val, t->r, tm, tr));
}
tipo get(int l, int r, node *t, int tl, int tr){
    if(l==tl && tr==r) return t->v;
  int tm=(tl + tr)>>1;
    if(r<=tm) return get(l, r, t->l, tl, tm);
    else if(l>=tm) return get(l, r, t->r, tm, tr);
  return oper(get(l, tm, t->l, tl, tm), get(tm, r, t->r, tm, tr));
}
```

# 4. Strings

## 4.1. KMP

```
vector<int> b; //back table b[i] maximo borde de [0..i)
void kmppre(string &P) //by gabina with love
{
  b.clear();
  b.rsz(P.size());
  int i =0, j=-1; b[0]=-1;
  while(i<sz(P))
  {
    while(j>=0 && P[i] != P[j]) j=b[j];
    i++, j++;
    b[i] = j;
  }
}
void kmp(string &T,string &P) //Text, Pattern -- O(|T|+|P|)
{
  kmppre(P);
  int i=0, j=0;
  while(i<sz(T))
  {
    while(j>=0 && T[i]!=P[j]) j=b[j];
    i++, j++;
    if(j==sz(P))
    {
      //P encontrado en T empezando en [i-j,i)
      j=b[j];
    }
  }
}
```

## 4.2. Z function

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

## 4.3. Trie

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

# 5. Geometría

# 6. Matemática

## 6.1. Identidades

$\sum_{i=0}^{n} \binom{n}{i} = 2^n$
$\sum_{i=0}^{n} i\binom{n}{i} = n * 2^{n-1}$
$\sum_{i=m}^{n} i = \frac{n(n+1)}{2} - \frac{m(m-1)}{2} = \frac{(n+1-m)(n+m)}{2}$
$\sum_{i=0}^{n} i = \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$
$\sum_{i=0}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} = \frac{n^3}{3} + \frac{n^2}{2} + \frac{n}{6}$
$\sum_{i=0}^{n} i(i-1) = \frac{8}{6}(\frac{n}{2})(\frac{n}{2}+1)(n+1)$ (doubles) $\rightarrow$ Sino ver caso impar y par
$\sum_{i=0}^{n} i^3 = \left(\frac{n(n+1)}{2}\right)^2 = \frac{n^4}{4} + \frac{n^3}{2} + \frac{n^2}{4} = [\sum_{i=1}^{n} i]^2$
$\sum_{i=0}^{n} i^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30} = \frac{n^5}{5} + \frac{n^4}{2} + \frac{n^3}{3} - \frac{n}{30}$
$\sum_{i=0}^{n} i^p = \frac{(n+1)^{p+1}}{p+1} + \sum_{k=1}^{p} \frac{B_k}{p-k+1}\binom{p}{k}(n+1)^{p-k+1}$
$r = e - v + k + 1$
Teorema de Pick: (Area, puntos interiores y puntos en el borde)
$A = I + \frac{B}{2} - 1$

## 6.2. Ec. Caracteristica

$a_0T(n) + a_1T(n-1) + ... + a_kT(n-k) = 0$
$p(x) = a_0x^k + a_1x^{k-1} + ... + a_k$
Sean $r_1, r_2, ..., r_q$ las raíces distintas, de mult. $m_1, m_2, ..., m_q$
$T(n) = \sum_{i=1}^{q} \sum_{j=0}^{m_i-1} c_{ij}n^j r_i^n$
Las constantes $c_{ij}$ se determinan por los casos base.

## 6.3. Teorema Chino del Resto

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

## 6.4. GCD & LCM

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

## 6.5. Euclides Extendido

```
void extendedEuclid (ll a, ll b){ //a * x + b * y = d
  if (!b) {x=1; y=0; d=a; return;}
  extendedEuclid (b,a%b);
  ll x1=y;
  ll y1=x-(a/b)*y;
  x=x1; y=y1;
}
```

## 6.6. Combinatoria

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

## 6.7. Exponenciación de Matrices y Fibonacci

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

```
    else mul(a, a), n/=2;
  }
}

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

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

## 6.8. Operaciones Modulares

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

## 6.9. Funciones de Primos

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

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

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

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

//Usar asi: divisores(fac, divs, fac.begin()); NO ESTA ORDENADO
```

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

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

## 6.10. Phollard's Rho

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

## 6.11. Inversos

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

## 6.12. Fracciones

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

## 6.13. Simpson

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

## 6.14. 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. Grafos

# 8. Flow

## 8.1. 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;
}
```

## 8.2. Push Relabel

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

```

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

# 9. Juegos

## 9.1. Ajedrez

### 9.1.1. Non-Attacking N Queen

**Utiliza:** <algorithm>

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

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

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

# 10. Utils

## 10.1. 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;
}
```

## 10.2. Contar bits

```
int x = 5328; // 00000000000000000001010011010000

//cuenta los zeros a la izquierda del numero
__builtin_clz(x); // 19

//cuenta los zeros a la derecha del numero
__builtin_ctz(x); // 4

//cuenta los unos en el numero
__builtin_popcount(x); // 5

//cuenta la paridad de cantidad de unos
__builtin_parity(x); // 1
```