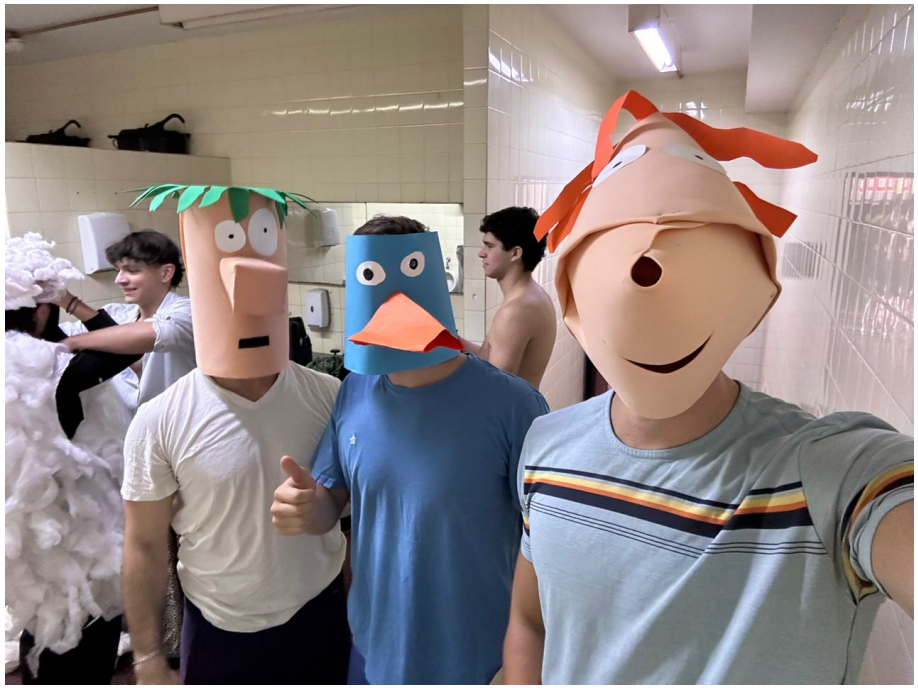
Notebook

UTN FRRO - HOLA

2024



Contents

1 Basics

1.1 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 all(v) v.begin(), v.end()
#define dbg(v) cerr << #v << ": " << v;
#define HOLA ios::sync_with_stdio(false); cin.tie(NULL); cout.tie(NULL);

using namespace std;
typedef long long ll;
typedef pair<int, int> ii;

int main() {
    HOLA
    #ifdef EBUG
        freopen("<ej>.txt", "r", stdin);
    #endif
    return 0;
}
```

1.2 Compilation

```
g++ -DEBUG <ej>.cpp -o a && time ./a
```

2 Math

2.1 Identidades

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

2.2 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`)
$10! = 3.628.800$	$21! = 51.090.942.171.709.400.000$

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

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

2.3 Reglas de divisibilidad

Nro	Regla	Ejemplo
1	Todos los nÃžmeros	5: porque si divides 5:1=5 y ese nÃžmero es un mÃžltiplo o divisor de cualquier nÃžmero.
2	El nÃžmero termina en una cifra par.	378: porque la Ãžltima cifra (8) es par.
3	La suma de sus cifras es un mÃžltiplo de 3.	480: porque 4+8+0=12 es mÃžltiplo de 3.
4	Sus Ãžltimos dos dÃŋgitos son 0 o un mÃžltiplo de 4.	300 y 516 son divisibles entre 4 porque terminan en 00 y en 16, respectivamente, siendo este Ãžltimo un mÃžltiplo de 4 (16=4*4).
5	La Ãžltima cifra es 0 o 5.	485: porque termina en 5.
7	Un nÃžmero es divisible entre 7 cuando, al separar la Ãžltima cifra de la derecha, multiplicarla por 2 y restarla de las cifras restantes la diferencia es igual a 0 o es un mÃžltiplo de 7. Otro sistema: Si la suma de la multiplicaciÃşn de los nÃžmeros por la serie 2,3,1,-2,-3,-1... da 0 o un mÃžltiplo de 7.	34349: separamos el 9, y lo duplicamos (18), entonces 3434-18=3416. Repetimos el proceso separando el 6 (341'6) y duplicÃąndolo (12), entonces 341-12=329, y de nuevo, 32'9, 9*2=18, entonces 32-18=14; por lo tanto, 34349 es divisible entre 7 porque 14 es mÃžltiplo de 7. Ejemplo mÃĺtodo 2: 34349: [(2*3)+(3*4)+(1*3)-(2*4)-(3*9)]= 6+12+3-8-27 = -14.8
8	Para saber si un nÃžmero es divisible entre 8 hay que comprobar que sus tres Ãžltimas cifras sean divisibles entre 8. Si sus tres Ãžltimas cifras son divisibles entre 8 entonces el nÃžmero tambiÃĺn es divisible entre 8.	Ejemplo: El nÃžmero 571.328 es divisible por 8 ya que sus Ãžltimas tres cifras (328) son divisibles por 8 (32 = 8*4 y 8 = 8*1). Realizando la divisiÃşn comprobamos que 571.328 : 8 = 71.416

Continúa		
Nro	**Regla**	**Ejemplo**
9	Un nÃžmero es divisible por 9 cuando al sumar todas sus cifras el resultado es mÃžltiplo de 9.	504: sumamos 5+0+4=9 y como 9 es mÃžltiplo de 9 504 es divisible por 9 5346: sumamos 5+3+4+6=18 y como 18 es mÃžltiplo de 9, 5346 es divisible por 9.
10	La Ãžltima cifra es 0.	4680: porque termina en 0
11	Sumando las cifras (del nÃžmero) en posiciÃşn impar por un lado y las de posiciÃşn par por otro. Luego se resta el resultado de ambas sumas obtenidas. Si el resultado es cero o un mÃžltiplo de 11, el nÃžmero es divisible entre este. Si el nÃžmero tiene solo dos cifras y estas son iguales serÃą mÃžltiplo de 11.	42702: 4+7+2=13 Âů 2+0=2 Âů 13-2=11 âĘŠ 42702 es mÃžltiplo de 11. 66: porque las dos cifras son iguales. Entonces 66 es mÃžltiplo de 11.
13	Un nÃžmero es divisible entre 13 cuando, al separar la Ãžltima cifra de la derecha, multiplicarla por 9 y restarla de las cifras restantes la diferencia es igual a 0 o es un mÃžltiplo de 13	3822: separamos el Ãžltimo dos (382'2) y lo multiplicamos por 9, 2ÃŮ9=18, entonces 382-18=364. Repetimos el proceso separando el 4 (36'4) y multiplicÃąndolo por 9, 4ÃŮ9=36, entonces 36-36=0; por lo tanto, 3822 es divisible entre 13.
17	Un nÃžmero es divisible entre 17 cuando, al separar la Ãžltima cifra de la derecha, multiplicarla por 5 y restarla de las cifras restantes la diferencia es igual a 0 o es un mÃžltiplo de 17	2142: porque 214'2, 2*5=10, entonces 214-10=204, de nuevo, 20'4, 4*5=20, entonces 20-20=0; por lo tanto, 2142 es divisible entre 17.
19	Un nÃžmero es divisible entre 19 si al separar la cifra de las unidades, multiplicarla por 2 y sumar a las cifras restantes el resultado es mÃžltiplo de 19.	3401: separamos el 1, lo doblamos (2) y sumamos 340+2= 342, ahora separamos el 2, lo doblamos (4) y sumamos 34+4=38 que es mÃžltiplo de 19, luego 3401 tambiÃľn lo es.

Continúa		
Nro	**Regla**	**Ejemplo**
20	Un nÃžmero es divisible entre 20 si sus dos Ãžltimas cifras son ceros o mÃžltiplos de 20. Cualquier nÃžmero par que tenga uno o mÃąs ceros a la derecha, es mÃžltiplo de 20.	57860: Sus 2 Ãžltimas cifras son 60 (Que es divisible entre 20), por lo tanto 57860 es divisible entre 20.
23	Un nÃžmero es divisible entre 23 si al separar la cifra de las unidades, multiplicar por 7 y sumar las cifras restantes el resultado es mÃžltiplo de 23.	253: separamos el 3, lo multiplicamos por 7 y sumamos 25+21= 46, 46 es mÃžltiplo de 23 asÃŋ que es divisible entre 23.
25	Un nÃžmero es divisible entre 25 si sus dos Ãžltimas cifras son 00, o en mÃžltiplo de 25 (25,50,75,...)	650: Es mÃžltiplo de 25 por lo cual es divisible. 400 tambiÃľn serÃą divisible entre 25.
29	Un nÃžmero es divisible entre 29 cuando, al separar la Ãžltima cifra de la derecha, multiplicarla por 3 y restarla de las cifras restantes la diferencia es igual a 0 o es un mÃžltiplo de 29	2436: separamos el 6 (243'6) y lo multiplicamos por 3, 6ÃŮ3=18, entonces 243-18=225. Repetimos el proceso separando el 5 (22'5) y multiplicÃąndolo por 3, 5ÃŮ3=15, entonces 22-15=7, que no es divisible entre 29.

2.4 Coprimos

Son aquellos nÃžmeros enteros a y b cuyo Ãžnico factor en comÃžn que tienen es 1. Equivalentemente son coprimos, si, y solo si, su mÃąximo comÃžn divisor (MCD) es igual a 1. Dos nÃžmeros coprimos no tienen por quÃľ ser primos absolutos de forma individual. 14 y 15 son compuestos, sin embargo son coprimos, pues: $gcd(14, 15) = 1$

2.5 Primos

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

#define forn(i, n) for (int i = 0; i < n; i++)
const int MAXN = 1e5+1;

// Si la criba[n] == 0 => n es primo
// si no criba[n] es el divisor mas grande de n
#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();
  for(int i = 2; i < MAXP+1; i++) if (!criba[i]) primos.push_back(i);
}

// O(sqrt(n))
// para buscar los divisores de un unico numero
vector<int> divisors(int n) {
    vector<int> div;
    for (int d = 1; d * d <= n; d++) {
        if (n%d == 0) {
            div.push_back(d);
            div.push_back(n/d);
        }
        if (d*d == n) div.pop_back(); // para cuadrados perfectos
    }
    sort(div.begin(), div.end());
    return div;
}

// precalcular divisores
// O(MAXN * lg(MAXN))
void pre_div() {
    vector<int> sum_div;
    vector<int> divs[MAXN];
    forn(i, MAXN) for(int j = i; j < MAXN; j+=i) {
        sum_div[j] += i;
        divs[j].push_back(i);
    }
}

// factorizar en O(sqrt(n))
map<ll, ll> fact_sqrt(int n) {
    map<ll, ll> F;
    for (int p = 2; p * p <= n; p++) {
        while (n%p == 0) {
            F[p]++;
            n /= p;
        }
    }
    if (n > 1) F[n]++;
    return F;
}

// factoriza bien numeros hasta MAXP^2
// O (cant primos)
// esta se podria hacer con la otra criba
map<ll,ll> fact(ll n) {
  map<ll,ll> ret; // factor, cantidad de veces
  for(auto p : primos){
    while(!(n % p)){ // mientras no sea divisor?
      ret[p]++; //divisor found
      n /= p;
    }
  }
  if(n > 1) ret[n]++;
  return ret;
}

// factoriza bien numeros hasta MAXP
map<ll,ll> fact2(ll n){ //O (lg n)
  map<ll,ll> ret;
  while (criba[n]){
    ret[criba[n]]++;
    n/=criba[n];
  }
  if(n>1) ret[n]++;
  return ret;
}
```

2.6 Combinatoria

3 Estructuras

3.1 vector

FunciÃşn	ExplicaciÃşn	O
(constructor)	Construct vector	O(n)
(destructor)	Vector destructor	O(n)
operator=	Assign content	O(n)
begin	Return iterator to beginning	O(1)
end	Return iterator to end	O(1)
rbegin	Return reverse iterator to reverse beginning	O(1)
rend	Return reverse iterator to reverse end	O(1)
cbegin	Return const_iterator to beginning	O(1)
cend	Return const_iterator to end	O(1)
crbegin	Return const_reverse_iterator to reverse beginning	O(1)
crend	Return const_reverse_iterator to reverse end	O(1)
size	Return size	O(1)
resize	Change size	O(n)
empty	Test whether vector is empty	O(1)
operator[]	Access element	O(1)
at	Access element	O(1)
front	Access first element	O(1)
back	Access last element	O(1)
assign	Assign vector content	O(n)
push_back	Add element at the end	O(1)
pop_back	Delete last element	O(1)
insert	Insert elements	O(n)
erase	Erase elements	O(n)
swap	Swap content	O(1)
clear	Clear content	O(n)
emplace	Construct and insert element	O(1)
emplace_back	Construct and insert element at the end	O(1)

3.1.1 emplace

```cpp
vector<int> myvector = {10,20,30};

auto it = myvector.emplace ( myvector.begin()+1, 100 );
myvector.emplace ( it, 200 );
myvector.emplace ( myvector.end(), 300 );

for (auto& x: myvector)
    cout << ' ' << x;
// 10 200 100 20 30 300
```

3.1.2 resize

Resizes the container so that it contains n elements.

```cpp
void resize(size_type n, const value_type& val);
```

```cpp
vector<int> myvector;
for (int i=1; i<10; i++) myvector.push_back(i);

myvector.resize(5);
myvector.resize(8, 100);
myvector.resize(12);

for (int i=0;i<myvector.size();i++)
    cout << ' ' << myvector[i];
//  1 2 3 4 5 100 100 100 0 0 0 0
```

3.1.3 assign

Assigns new contents to the vector, replacing its current contents, and modifying its size accordingly.

```cpp
void assign(size_type n, const value_type& val);
```

```cpp
vector<int> first;
vector<int> second;
vector<int> third;

first.assign (7,100);             // 7 ints with a value of 100

vector<int>::iterator it;
it=first.begin()+1;

second.assign (it,first.end()-1); // the 5 central values of first

int myints[] = {1776,7,4};
third.assign (myints,myints+3);   // assigning from array.

first.size() // -> 7
second.size() // -> 5
third.size()) // -> 3
```

3.2 unordered set

FunciÃşn	ExplicaciÃşn	O
(constructor)	Construct unordered_set	-
(destructor)	Destroy unordered_set	-
operator=	Assign content	O(n)
empty	Test whether container is empty	O(1)
size	Return container size	O(1)
max_size	Return maximum size	O(1)
begin	Return iterator to beginning	O(1)
end	Return iterator to end	O(1)
find	Get iterator to element	O(n)
count	Count elements with a specific key Returns 0 or 1	O(n)
equal_range	Get range of elements with a specific key	O(n)
emplace	Construct and insert element	O(n)
emplace_hint	Construct and insert element with hint	O(n)
insert	Insert elements	O(n)
erase	Erase elements	O(n)
clear	Clear content	O(n)
swap	Swap content	O(1)
reserve	Request a capacity change	O(n)
key_eq	Get key equivalence predicate	O(1)

```cpp
unordered_set<int> s;
s.insert(3);
s.insert(5);
s.erase(3);
s.count(3); // -> 0
s.count(5); // -> 1
```

3.3 Iterators

```cpp
sort(v.begin(), v.end());
reverse(v.begin(), v.end());
random_shuffle(v.begin(), v.end());

sort(v.begin(), v.end(), sortbysec);
```

sort complexity = $O(Nlog_2N)$.

Ordenar un vector de pair por su segunda componente

```cpp
vector<pair<int, int>> v;

bool sortbysec(const pair<int,int> &a, const pair<int,int> &b){
  return (a.second < b.second);
}
```

TambiÃ¡n es posible ordenar un array normal:

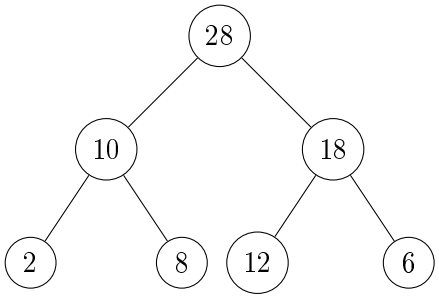
```cpp
sort(a, a+n);
reverse(a, a+n);
random_shuffle(a, a+n)
```

3.4 Indexed Set

```cpp
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
using namespace std;

template<typename T>
using indexed_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;
//indexed_set<int> s;
//s.insert(5);
//s.insert(1);
//s.insert(10);
//s.insert(3);

// Elementos ordenados: [1, 3, 5, 10]

//cout << *s.find_by_order(0) << "\n"; // 1
```

```
//cout << *s.find_by_order(2) << "\n"; // 5

//cout << s.order_of_key(5) << "\n";   // 2 (hay dos menores que 5)
//cout << s.order_of_key(6) << "\n";   // 3 (1,3,5 < 6)
//s.insert(8);
//cout << *s.find_by_order(4) << "\n";
//s.erase(3);
//cout << s.order_of_key(5) << "\n";   // 1 (quedaron 1 y 5 y 10)
```

3.5 Segment Tree




```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
#define dforn(i,n) for(int i=n-1; i>=0; i--)
#define fora(p, i, n) for(int i = p; i < n; i++)
#define forn(i, n) for(int i = 0; i < n; i++)
#define fori(i, n) for(int i = n - 1; i <= 0; i--)
#define forall(it, v) for (auto it = v.begin(); it != v.end(); it++)
#define MAXN 200000
#define operacion(x, y) make_pair(max(x.first, x.second + y.first), x.
    second + y.second)
#define neutro make_pair(0,0)
#define tipo pair<ll, ll>

struct RMQ{
    int sz;
    tipo t[4*MAXN];
    tipo &operator[](int p){return t[sz+p];}
    void init(int n){//O(nlgn)
        sz = 1 << (32-__builtin_clz(n));
        forn(i, 2*sz) t[i] = neutro;
    }
    void updall(){
        dforn(i, sz) t[i] = operacion(t[2*i], t[2*i+1]);
    }
    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){
        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){
        for(p+=sz; p>0 && t[p] != val;){
            t[p]=val;
            p/=2;
            val = operacion(t[2*p],t[2*p+1]);
        }
    }
}mx;

//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 eneutro=0; const Alt aneutro=0;
#define MAXN 100000
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]=eneutro;
    forn(i, 2*sz) dirty[i]=aneutro;
  }
    void updall(){
        dforn(i, sz) t[i] = operacion(t[2*i], t[2*i+1]);
    }
  void push(int n, int a, int b){//propaga el dirty a sus hijos
    if(dirty[n]!=aneutro){
      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]=aneutro;
    }
  }
  Elem get(int i, int j, int n, int a, int b){//O(lgn)
    if(j<=a || i>=b) return eneutro;
    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; //ver esto gordo
      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;

void inicializate(){
  vector<Elem> v;
  mx.init(v.size()); forn(i, v.size()) rmq[i] = v[i]; mx.updall();
}
```

3.6 Array

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

int main(){
    int dp[50][50];
    // llenar una lista para DP
    memset(dp,-1, sizeof(dp));

    for(int i=0; i<50;i++){
        for(int j=0; j<50;j++){
            cout << dp[i][j] << " ";
        }
        cout << endl;
    }

}
```

3.7 BitSet

FunciÃşn	ExplicaciÃşn	O
Template parameters		
N	nÃžmero de bits	O(1)
Member functions		
(constructor)	construye el ‘bitset‘	O(N)
Element access		
operator[]	accede a un bit especÃıfico	O(1)
all, any, none	todos, algÃžn o ningÃžn bit estÃą en true	O(N)
count	nÃžmero de bits establecidos en true	O(N)
size	nÃžmero de bits que contiene	O(1)
Modifiers		
operator&= operator|= operator^= operator~= operator«= operator»= operator« operator»		O(N)
set() set(int pos)	pone todos los bits en el valor dado	O(N) O(1)
reset() reset(int pos)	establece bits en ‘false‘	O(N) O(1)
flip() flip(int pos)	alterna los valores de los bits	O(N) O(1)
Conversions		
to_string	representaciÃşn en cadena	

4 Flujo

4.1 Dinic

```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 pb push_back
#define sz(v) ((int)v.size())
#define INF 1e18

using namespace std;
typedef long long ll;

// ojo con el max
const int MAX = 501;
// Corte minimo: vertices con dist[v]>=0 (del lado de src) VS.  dist[v]==-1 (del lado del dst)
// Para el caso de la red de Bipartite Matching (Sean V1 y V2 los conjuntos mas proximos a src y dst respectivamente):
// Reconstruir matching: para todo v1 en V1 ver las aristas a vertices de V2 con it->f>0, es arista del Matching
// Min Vertex Cover: vertices de V1 con dist[v]==-1 + vertices de V2 con dist[v]>0
// Max Independent Set: tomar los vertices NO tomados por el Min Vertex Cover
// Max Clique: construir la red de G complemento (debe ser bipartito!) y encontrar un Max Independet Set
// Min Edge Cover: tomar las aristas del matching + para todo vertices no cubierto hasta el momento, tomar cualquier arista de el
int nodes, src, dst;
int dist[MAX], q[MAX], work[MAX];
struct Edge {
    int to;
    int rev; // indice en en la red residual
    ll f; // flow
    ll cap; // capacity
    Edge(int to, int rev, ll f, ll cap) : to(to), rev(rev), f(f), cap(cap) {}
};

vector<Edge> G[MAX];
void addEdge(int s, int t, ll cap){
    G[s].pb(Edge(t, sz(G[t]), 0, cap));
    G[t].pb(Edge(s, sz(G[s])-1, 0, 0));
}

bool dinic_bfs(){
    fill(dist, dist+nodes, -1), dist[src]=0;
    int qt=0; q[qt++]=src;
    for(int qh=0; qh<qt; qh++){
        int u = q[qh];
        forall(e, G[u]){
            int v=e->to;
            if(dist[v]<0 && e->f < e->cap)
                dist[v]=dist[u]+1, q[qt++]=v;
        }
    }
    return dist[dst]>=0;
}
ll dinic_dfs(int u, ll f){
    if(u == dst) return f;
    for(int &i=work[u]; i<sz(G[u]); i++){
        Edge &e = G[u][i];
        if(e.cap <= e.f) continue;
        int v = e.to;
        if(dist[v]==dist[u]+1){
                ll df = dinic_dfs(v, min(f, e.cap-e.f));
                if(df>0){
                    e.f+=df, G[v][e.rev].f-= df;
                    return df;
                }
        }
    }
    return 0;
}

ll maxFlow(int _src, int _dst){
    src=_src, dst=_dst;
    ll result = 0;
    while(dinic_bfs()){
        fill(work, work+nodes, 0);
        while(ll delta=dinic_dfs(src,INF))
            result += delta;
    }
    // todos los nodos con dist[v]!=-1 vs los que tienen dist[v]==-1 forman el min-cut
    return result;
}

bool vis[501][501];
void corteMinimo(){
    forn(i, nodes){
        if(dist[i] >= 0){
            for(auto j : G[i]){
                if(dist[j.to] == -1 && !vis[i][j.to]){
                    vis[i][j.to] = true;
                    cout << i+1 << " " << j.to+1 << endl;
                }
            }
        }
    }
}

int capa1, capa2, aristas;
void matchingMaximo(){
    forr(i,1, capa1+1){
        for(auto j : G[i]){
            if(j.f > 0){
```

```
                cout << i << " " << j.to - capa1 << endl;
            }
        }
    }
}
vector<int> path;
void recrearUnCamino(int i){ //solo te recrea un camino del flujo
    path.push_back(i);
    if(i == nodes-1)return;
    for(auto j: G[i]){
        if(j.f>0 && !vis[i][j.to]){
            vis[i][j.to] = true;
            path.push_back(j.to+1);
            recrearUnCamino(j.to);
            break;
        }
    }
}


int main() {
    // Example usage
    int m;
    cin >> nodes >> m;
    forn(i, m){
        int a, b, c;
        cin >> a >> b >> c;
        addEdge(a-1, b-1, (ll)c);
    }
    ll flow = maxFlow(0, nodes-1);
    cout << flow << endl;
    return 0;
}
```

4.2 Hungarian

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

#define forr(i,a,b) for(int i =(a);i<int(b); ++i)
#define forn(i,n) forr(i,0,n)
#define dforr(i,a,b) for(int i =(b)-1;i>=int(a);--i)
#define dforn(i,n) dforr(i,0,n)
#define db(v) cerr << #v << " = " << (v) << "\n";
#define sz(v) (int((v).size()))
#define all(v) (v).begin(), (v).end()
#define pb push_back
#define pp pop_back
#define fst first
#define snd second

using ll = long long;
using ull = unsigned ll;
using ld = long double;
using pii = pair<int,int>;
using pll = pair<ll,ll>;
using vll = vector<ll>;

const ll MAXN = 110;
//const ll INF = 1e18 +100;
const ll MOD = 1e9+7;
const ld EPS = 1e-9;
const ld PI = acosl(-1);


//matching minimo con pesos en las aristas pai
// Minima asignacion en O(n^3)
// Lado izquierdo [0..n), lado derecho [0..m). OJO: n <= m
typedef long double td; typedef vector<int> vi; typedef vector<td> vd;
const td INF = 1e100;    // Para max asignacion, INF = 0, y negar costos
bool zero(td x) {return fabs(x) < 1e-9;}  // Para int/ll: return x==0;
struct Hungarian{
    vector<pii> ans;      //Guarda las aristas usadas en la asignacion (
        cada arista es [0..n)x[0..m))
    int n; vector<vd> cs; vi L, R;
  Hungarian(int N, int M) : n(max(N,M)), cs(n,vd(n)), L(n), R(n){
    forn(x, N) forn(y, M) cs[x][y] = INF;
  }
  void set(int x, int y, td c) { cs[x][y] = c; }
  td assign(){
    int mat = 0; vd ds(n), u(n), v(n); vi dad(n), sn(n);
    forn(i, n) u[i] = *min_element(all(cs[i]));
    forn(j, n){
      v[j] = cs[0][j]-u[0];
      forr(i, 1, n) v[j] = min(v[j], cs[i][j] - u[i]);
    }
    L = R = vi(n, -1);
    forn(i, n) forn(j, n) if(R[j] == -1 and zero(cs[i][j] - u[i] - v[j]))
        {
      L[i] = j; R[j] = i; mat++; break;
    }
    for(; mat < n; mat ++){
      int s = 0, j = 0, i;
      while(L[s] != -1) s++;
      fill(all(dad), -1); fill(all(sn), 0);
      forn(k, n) ds[k] = cs[s][k]-u[s]-v[k];
      while(true){
        j = -1;
        forn(k, n) if(!sn[k] and (j == -1 or ds[k] < ds[j])) j = k;
        sn[j] = 1; i = R[j];
        if(i == -1) break;
        forn(k, n) if(!sn[k]){
          auto new_ds = ds[j] + cs[i][k] - u[i]-v[k];
          if(ds[k] > new_ds) ds[k]=new_ds, dad[k]=j;
        }
      }
      forn(k, n) if(k!=j and sn[k]) { auto w = ds[k]-ds[j]; v[k] += w, u[
          R[k]] -= w; }
      u[s] += ds[j];
```

```
      while(dad[j] >= 0) { int d = dad[j]; R[j] = R[d]; L[R[j]] = j; j =
          d; }
      R[j] = s; L[s] = j;
    }
    td value = 0; forn(i, n) value += cs[i][L[i]], ans.pb({i, L[i]});
    return value;
  }
};

bool f(pii x, pii y){
    return x.snd < y.snd;
}
```

5 Grafos

5.1 Recorrer Grafos

Dado un Grafo como lista de adjacencias

```
#include <bits/stdc++.h>
#define MAXN 100000
using namespace std;

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

Podemos recorrerlo con DFS o con BFS.

5.1.1 DFS

```
void dfs(int nodo){
    if(visited[nodo]) return;
    visited[nodo] = true;

    for(auto it : G[nodo]){
        if(!visited[it]){
            // logic
        }
    }
}
```

5.1.2 BFS

```
void bfs(int nodo){
    cola.push(nodo);

    while(!cola.empty()){
        Nodo actual = cola.front();
        cola.pop();

        for (int vecino : G[actual]) {
            if (!visited[vecino]) {
                visited[vecino] = true;
                cola.push(vecino);
            }
        }
    }
}
```

5.2 Camino MÃŋnimo

5.2.1 Bellman-Ford

El algoritmo de Bellman-Ford encuentra el camino desde un nodo de origen a todos los nodos del grafo.

Complejidad = O(nm)

```cpp
#include <bits/stdc++.h>
#define INF 100000000
using namespace std;

vector<tuple<int, int, int>> edges;
int n;

#define op min

// acepta negativos
// se puede cambiar a buscar el camino maximo cambiando
// min por max
void bellman_ford(int x){
    vector<int> dist(n, INF);
    dist[x] = 0;

    for (int i = 1; i <= n-1; i++) {
        for (auto e : edges) {
            int a, b, w;
            tie(a, b, w) = e;
            dist[b] = min(dist[b], dist[a]+w);
        }
    }

    // Buscar ciclos negativos
    // Un ciclo que se reduce infinitamente
    unordered_set<int> in_cycle;
    for (auto [a, b, w] : edges) {
        if (dist[a]+w < dist[b]) {
            in_cycle.insert(b);
        }
        dist[b] = min(dist[b], dist[a]+w);
    }
}
```

5.2.2 Ciclos negativos

El algoritmo es capaz de detectar ciclos negativos. Para eso Se debe correr una vez m'as

5.2.3 Dijkstra

El algorimo necesita que todos los pesos sean 0

Complejidad = O(n + m log m)

```cpp
#include <bits/stdc++.h>
#define MAXN 100000
using namespace std;
typedef pair<int, int> ii;

vector<pair<int, int>> G[MAXN]; //Lista de pares, dest, peso
bool visited[MAXN];
int n;

void dijkstra(int x){
    // La PQ esta ordenada de menor a mayor
    priority_queue<ii, vector<ii>, greater<ii>> q;
    vector<int> distance(n, MAXN);
    distance[x] = 0; // x es el punto de inicio
    q.push({0, x});

    while (!q.empty()) {
        int a = q.top().second; q.pop();
        if (visited[a]) continue;
        visited[a] = true;

        for (auto [b, w] : G[a]) {
            if (distance[a]+w < distance[b]) {
                distance[b] = distance[a]+w;
                q.push({distance[b], b});
            }
        }
    }
}
```

5.2.4 Floyd-Warshall

```

/* minima distancia entre cada par de nodos en un grafo dirigido.
    O(n^3)
*/
int dist[MAX_N][MAX_N]; //Distancia de i a j

// llenar la matriz de adjayencia
// tmb se podria armar a medida que lees la entrada
for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= n; j++) {
        if (i == j) dist[i][j] = 0;
        else if (adj[i][j]) dist[i][j] = adj[i][j];
        else dist[i][j] = INF;
    }
}

/* After this, the shortest dists can be found as follows: */
for (int k = 1; k <= n; k++) {
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            dist[i][j] = min(dist[i][j], dist[i][k]+dist[k][j]);
        }
    }
}
```

5.3 Spanning Tree

5.3.1 UnionFind

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

struct UnionFind{
  vector<int> f;//the array contains the parent of each node
  void init(int n){f.clear(); f.insert(f.begin(), 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) f[comp(i)] = comp(j);
    return con;
  }
} uf;
```

Tambien podemos guardar el tamaÃśo de los sets:

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

struct UnionFind{
  vector<int> f;//the array contains the parent of each node
    vector<int> size;
    int largest;
  void init(int n){
        f.clear(); f.insert(f.begin(), n, -1);
        largest = 1;
        size.insert(size.begin(), n, 1);
    }
  int comp(int x){return (f[x]==-1?x:f[x]=comp(f[x]));}//O(1)
    bool join(int i, int j) {
        int rootI = comp(i);
        int rootJ = comp(j);

        if (rootI == rootJ) return true; // Ya estan en el mismo
            componente

        if (size[rootI] < size[rootJ]) {
            swap(rootI, rootJ);
        }

        f[rootJ] = rootI;
        size[rootI] += size[rootJ];
        largest = max(largest, size[rootI]); // Actualizar la componente
            mas grande
        return false;
    }
    bool same(int a, int b) {
        return comp(a) == comp(b);
    }
} uf;
```

5.3.2 Kruskal

Obtener el spanning tree de costo minimo dado un grafo ponderado no dirigido

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

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

int edges_used = 0;
int kruskal(){
    int cost=0;
    sort(E.begin(), E.end()); //Ordenar aristas de menor a mayor
    uf.init(n);
    for(Ar it : E){
        if(uf.comp(it.a) != uf.comp(it.b)){ //Si no estan conectados
            uf.join(it.a, it.b); //Conectar
            cost += it.w;
            edges_used++;
        }
    }
    return cost;
}
```

5.4 Aplicaciones comunes

5.4.1 Chequear si es conexo

Tirar un dfs desde un nodo cualquiera v es conexo si alcanzamos todos los otros nodos del grafo

5.4.2 Bipartito

```cpp
#include <bits/stdc++.h>
using namespace std;
#define forn(i, n) for(int i = 0; i < n; i++)
#define DBG(x) cerr << #x << ": " << x << endl;
#define MAXN 1000000

/* Comprobar si el grafo es bipartito
 * BFS pintando intercaladamente
 * */

int main(){
    int n, m;
    // Leer el grafo
    vector<vector<int>> G(n);

    queue<int> q;
    vector<int> color(n, -1);

    forn(i, n){
        if (color[i] != -1) continue;

        q.push(i); color[i] = 0;
        while(!q.empty()){
            int n = q.front(); q.pop();

            for (int adj : G[n]) {
                if (color[adj] == -1){
                    color[adj] = color[n] ^ 1; // invertir, si es 1 => 0, 0 => 1
                    q.push(adj);
                }else if (color[adj] == color[n]){
                    cout << "IMPOSSIBLE\n"; // no es bipartito
                    return 0;
                }
            }
        }
    }

    forn(i, n) {
        cout << color[i] + 1 << " ";
    }
    cout << '\n';

    return 0;
}
```

5.4.3 Ciclos

```cpp
// https://cses.fi/problemset/task/1669/
#include <bits/stdc++.h>
using namespace std;
#define forn(i, n) for(int i = 0; i < n; i++)
#define DBG(x) cerr << #x << ": " << x << endl;
#define MAXN 1000000

vector<vector<int>> G;
vector<bool> visited;
vector<int> parent;

// buscamos algun ciclo
int dfs(int nodo, int prev = 0){
    parent[nodo] = prev;
    if(visited[nodo]) return nodo;
    visited[nodo] = true;

    for(int adj : G[nodo]){
        if (adj == prev) continue; // Para no volver para atras
        int cycle = dfs(adj, nodo);
        if (cycle != -1) return cycle; // si hay ciclo
    }
    return -1; // si no ahay ciclo devuelve -1
}

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

    int n, m;

    // leer grafo

    int cycle_start = -1;
    forn(i, n){
        if (visited[i]) continue; // si ya esta visitado, ya forma parte
            d un ciclo
        cycle_start = dfs(i);
        if (cycle_start == -1){
            cout << "IMPOSSIBLE\n";
            return 0;
        }
    }

    // reconstruir camino
    vector<int> path;
    int next = parent[cycle_start];
    path.push_back(cycle_start+1);
    while(next != cycle_start){
        path.push_back(next+1);
        next = parent[next];
    }
    path.push_back(cycle_start+1);

    // mostrar path
}
```

5.4.4 Componentes de un grafo

```cpp
#include <bits/stdc++.h>
using namespace std;
#define forn(i, n) for(int i = 0; i < n; i++)
#define MAXN 1000000

/* Buscar todas las componentes del grafo
 * De cada componente agarrar un elemento cualquiera
 * */

vector<int> G[MAXN];
int components[MAXN]; //LLenar en -1

int dfs(int n, int component=1){
    if (components[n] != -1) return components[n];
    components[n] = component;

    for (auto adj : G[n]) {
        if (components[n] == -1) {
            dfs(adj, component);
        }
    }
    return component;
}

int main(){
    int n, m; //Leer el grafo

    forn(i, n){
        dfs(i, i);
    }

    return 0;
}
```

5.4.5 Dijkstra modificado

```cpp
#include <bits/stdc++.h>
using namespace std;
#define forn(i, n) for(int i = 0; i < n; i++)

typedef unsigned long long ll;
typedef tuple<ll, int, bool> e;

/* Dijkstra modificado
 * Camino minimo pero tenes un cupon de descuento q usas una vez, q
    reduce el precio a w/2
 * Tenemos en cuenta dos estados diferentes para cada ciudad, uno
    habiendo usado el cupon y otro sin
```

```
 * Para cada vuelo, si no usamos el cupÃşn, tenemos dos opciones:
    - Usar el vuelo al costo original y no usar el cupÃşn.
    - Usar el cupÃşn en ese vuelo, reduciendo su costo a âŇŁw/2âŇŃ.
 * */

int main(){
    int n, m;
    vector<vector<pair<int, ll>>> G(n);
    vector<vector<bool>> visited(n, vector<bool>(2, false));

    // leer el grafo
    priority_queue<e, vector<e>, greater<e>> q;

    vector<vector<ll>> distance(2, vector<ll>(n, LONG_LONG_MAX));
    // distance[0][i] => sin usar el cupon, distance[1][i] => usando el
        cupon
    distance[0][0] = 0;
    q.push({0, 0, false}); // distance, v, si ya aplique el descuento

    while (!q.empty()) {
        auto [dist, a, discount] = q.top(); q.pop();
        if (visited[a][discount]) continue;
        visited[a][discount] = true;

        for (auto [b, w] : G[a]) {
            // Si ya aplique el descuento
            if (distance[discount][a]+w < distance[discount][b]) {
                distance[discount][b] = distance[discount][a]+w;
                q.push({distance[discount][b], b, discount});
            }
            if (!discount){
                ll w2 = w/2;
                if (distance[0][a]+w2 < distance[1][b]) {
                    distance[1][b] = distance[0][a]+w2;
                    q.push({distance[1][b], b, true});
                }
            }
        }
    }
    cout << min(distance[0][n-1], distance[1][n-1]) << '\n';
}
```

5.4.6 Max edge in path

Para grafos dirigidos

```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 1e5 + 1;
const int MAXM = 2e5 + 1;
const int LOGN = 18;

tuple<int, int, int> vet[MAXM];
int n, m, q, comp[MAXN], anc[MAXN][LOGN], val[MAXN][LOGN], nvl[MAXN];
unordered_map<int, int> cost[MAXN];
vector<int> adj[MAXN], custo[MAXN];

int find(int a){
  return a == comp[a] ? a : comp[a] = find(comp[a]);
}

void merge(int a, int b){
  comp[find(a)] = find(b);
}

void dfs(int v, int p){
  for(int i = 0; i < (int)adj[v].size(); i++){
    if(adj[v][i] != p){
      int u = adj[v][i], c = custo[v][i];
      nvl[u] = nvl[v]+1;
      anc[u][0] = v;
      val[u][0] = c;
      for(int j = 1; j < LOGN; j++){
        anc[u][j] = anc[anc[u][j - 1]][j - 1];
        val[u][j] = max(val[u][j - 1], val[anc[u][j - 1]][j - 1]);
      }
      dfs(u,v);
    }
  }
}

int path_max(int u, int v){
  if(nvl[u] < nvl[v])
    swap(u,v);
  int ret = 0;
  for(int i = LOGN-1; i >= 0; i--){
    if(nvl[u] - (1<<i) >= nvl[v]){
      ret = max(ret, val[u][i]);
      u = anc[u][i];
    }
  }
  if(u != v){
    for(int i = LOGN-1; i >= 0; i--){
      if(anc[u][i] != anc[v][i]){
        ret = max(ret, val[u][i]);
        ret = max(ret, val[v][i]);
        u = anc[u][i];
        v = anc[v][i];
      }
    }
    ret = max(ret, val[u][0]);
    ret = max(ret, val[v][0]);
  }
  return ret;
}

int main(){
  iota(comp, comp + MAXN, 0);
  scanf("%d%d", &n, &m);
  for(int i = 0; i < m; i++){
    int u, v, c;
```

```
    scanf("%d%d%d", &u, &v, &c);
    u--, v--;
    cost[u][v] = cost[v][u] = c;
    vet[i] = make_tuple(c, u, v);
  }
  sort(vet, vet + m);

  long long ans = 0;
  for(int i = 0; i < m; i++){
    int c, u, v;
    tie(c, u, v) = vet[i];
    if(find(u) != find(v)){
      merge(u, v);
      adj[u].emplace_back(v);
      adj[v].emplace_back(u);
      custo[u].emplace_back(c);
      custo[v].emplace_back(c);
      ans += c;
    }
  }
  dfs(0, 0);
  scanf("%d", &q);
  while(q--){
    int u, v;
    scanf("%d%d", &u, &v);
    u--, v--;
    if(anc[u][0] == v || anc[v][0] == u){
      printf("%lld\n", ans);
      continue;
    }
    printf("%lld\n", ans - path_max(u, v) + cost[u][v]);
  }
}
```

5.4.7 Max path

```
// Se resuelve con un Bellman-Ford
```

5.4.8 Strong Connectivity

Para grafos dirigidos

```
#include <bits/stdc++.h>
using namespace std;
#define forn(i, n) for(int i = 0; i < n; i++)

typedef long long ll;
const int MAXN = 1e5+1;

/* Checkear si el grafo es fuertemente conexo
 * DFS para el grafo normal desde una arista cualquiera v (en este caso
     1)
 * DFS en el grafo invertido desde la misma artista
 * Si existe arista sin visitar 'x' en algun dfs => "NO" v x
 * */

int n, m;

vector<int> G[MAXN];
vector<int> G_inv[MAXN];
vector<bool> visited(MAXN, false);

void dfs(vector<int> *graph, int x){
    visited[x] = true;
    for(int i : graph[x]){
        if(!visited[i]) dfs(graph, i);
    }
}

int main(){
    // - Leer grafo G y el grafo inverso G_inv

    dfs(G, 0);
    forn(i, n){
        if (!visited[i]){
            cout << "NO\n";
            cout << 1 << ' ' << i+1 << endl;
            return 0;
        }
    }
    fill(visited.begin(), visited.end(), false);
    dfs(G_inv, 0);
    forn(i, n){
        if (!visited[i]){
            cout << "NO\n";
            cout << i+1 << ' ' << 1 << endl;
            return 0;
        }
    }

    cout << "YES\n";
    return 0;
}
```

5.4.9 Topological Sort

```
#include <bits/stdc++.h>
using namespace std;
#define forn(i, n) for(int i = 0; i < n; i++)

typedef long long ll;
const int MAXN = 1e5;

/*  Topological Sort
 * */

```

```cpp
vector<int> state; //Tres estados 0, 1, 2
                   //0 -> no visitado, 1 -> visitando, 2 -> totalmente visitado
vector<int> G[MAXN];
vector<int> order;

bool dfs(int nodo){ //Devolvemos si true si encontramos un ciclo
    if(state[nodo] == 2) return false;
    state[nodo] = 1;

    bool cicle = false;
    for(auto it : G[nodo]){
        if(state[it] == 0){
            cicle = dfs(it);
        }else if(state[it] == 1){
            return true;
        }
    }
    state[nodo] = 2;
    order.push_back(nodo);
    return cicle;
}

int main(){
    int n, m;
    // Leer grafo

    state.assign(n+1, 0);

    bool cicle = false;
    forn(i, n){
        if (state[i] == 0){
            cicle = dfs(i);
        }
        if (cicle) break;
    }

    if (cicle){
        cout << "IMPOSSIBLE\n";
    }else{
        reverse(order.begin(), order.end());
        for(int i : order){
            cout << i+1 << ' ';
        }
        cout << '\n';
    }

    return 0;
}
```

5.5 Kosaraju

Encontrar todas las componentes fuertemente conexas en un grafo dirigido

```cpp
#include <bits/stdc++.h>
using namespace std;
#define forr(i, a, b) for(int i = (int) a; i < (int) b; i++)
#define forn(i, n) forr(i, 0, n)
#define dforn(i, n) for(int i = (int)(n-1); i >= 0; i--)
#define pb push_back

// Sacado del olaf
// Not tested, not even a bit
struct Korasaju {
  static const int default_sz = 1000001;
  int n;
  vector< vector<int> > G, revG, C, ady; // ady is the condensed graph
  vector<int> used, where;
  Korasaju(int sz = default_sz){
    n = sz;
    G.assign(sz, vector<int>());
    revG.assign(sz, vector<int>());
    used.assign(sz, 0);
    where.assign(sz, -1);
  }
  void addEdge(int a, int b){ G[a].pb(b); revG[b].pb(a); }
  void dfsNormal(vector<int> &F, int v){
    used[v] = true;
    forn(i, G[v].size()) if(!used[ G[v][i] ])
      dfsNormal(F, G[v][i]);
    F.pb(v);
  }
  void dfsRev(vector<int> &F, int v){
    used[v] = true;
    forn(i, revG[v].size()) if(!used[ revG[v][i] ])
      dfsRev(F, revG[v][i]);
    F.pb(v);
  }
  void build(){
    vector<int> T;
    fill(used.begin(), used.end(), 0);
    forn(i, n) if(!used[i]) dfsNormal(T, i);
    reverse(T.begin(), T.end());
    fill(used.begin(), used.end(), 0);
    forn(i, T.size()) if(!used[ T[i] ]){
      vector<int> F;
      dfsRev(F, T[i]);
      forn(i, F.size()) where[ F[i] ] = C.size();
      C.pb(F);
    }
    ady.resize(C.size()); // Create edges between condensed nodes
    forn(i, n) forn(j, G[i].size()){
      if(where[i] != where[ G[i][j] ]){
        ady[ where[i] ].pb( where[ G[i][j] ] );
      }
    }
```

```
    forn(v, C.size()){
      sort(ady[v].begin(), ady[v].end());
      ady[v].erase(unique(ady[v].begin(), ady[v].end()), ady[v].end());
    }
  }
};
```

6 Strings

6.1 Hash

```
#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++)

typedef long long ll;

using namespace std;
struct Hash {
    ll MOD = 999727999;
    ll BASE = 325255434;
    //ll MOD = 97;
    //ll BASE = 3;
    vector<ll> h, pot;
    string str;

    Hash(string& s) {
        int n = s.size();
        str = s;
        h.resize(n + 1);
        pot.resize(n + 1);
        h[0] = s[0];
        pot[0] = 1;
        for (int i = 0; i < n; i++) {
            h[i + 1] = (h[i] * BASE + s[i]) % MOD;
            pot[i + 1] = (pot[i] * BASE) % MOD;
        }
    }

    // Devuelve el hash del substring s[s..e] en O(1)
    ll get(int s, int e) {
        ll hash_val = (h[e + 1] - h[s] * pot[e - s + 1]) % MOD;
        if (hash_val < 0) hash_val += MOD;
        return hash_val;
    }

    // devuelve el hash del string cambiando en la iesima posicion de c1
        a c2.
    // s: inicio del string, e: fin, i: posicion a cambiar, c1: old char,
         c2: new char
    ll get_change(int s, int e, int i, int c1, int c2) {
        ll original_hash = get(s, e);
        int exp = e - i;
        ll diff = ((c2 - c1) * pot[exp]) % MOD;
        ll changed_hash = (original_hash + diff) % MOD;
        if (changed_hash < 0) changed_hash += MOD;
        return changed_hash;
    }
};
```

6.1.1 Examples

```
/**
 * Given a string and a pattern, your task is to count the number of
     positions
 * where the pattern occurs in the string.
 *
Input:
saippuakauppias
pp

Output:
2
**/

#include <bits/stdc++.h>
#include "hashing.h"
#define forn(i,n) for(ll i = 0; i < n; i++)
#define fora(p, i,n) for(ll i = p; i < n; i++)

using namespace std;

#define pb push_back
typedef long long ll;
#define MAXN 1000000010

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

    string s,t;
    cin >> s >> t;
    Hash h(s);
    Hash ha(t);
    int n = s.size(), res=0;

    forn(i, n-t.size()+1){
        if(t.size() > s.size()) break;
        if(h.get(i,i+t.size()-1) == ha.get(0,t.size()-1)) res++;
    }

    cout << res << endl;
    return 0;
}
```

6.1.2 Examples

```

#include <bits/stdc++.h>
#include <bitset>
#include <iostream>
#include <string>
using namespace std;
#define forn(i,n) for(ll i = 0; i < n; i++)
#define fora(p, i,n) for(ll i = p; i < n; i++)
#define pb push_back
typedef long long ll;
#define MAXN 1000000010
#ifdef EBUG
//local
#else
//judge
#endif

#define forr(i,s,n) for(int i=s; i<n; i++)
struct Hash{
    int P=1777771, MOD[2], PI[2];
    vector<int> h[2], pi[2];
    vector<ll> primos[2];
    Hash(string& s){
        MOD[0]=999727999; MOD[1]=1070777777;
        PI[0]=325255434; PI[1]=10018302;
        forr(k,0,2){
           h[k].resize(s.size()+1), pi[k].resize(s.size()+1), primos[k].
               resize(s.size()+1);
        }
        forr(k,0,2){
            h[k][0]=0;
            pi[k][0]=1;
            ll p=1;
            primos[0][0]=p;
            primos[1][0]=p;
            forr(i,1,s.size()+1){
                h[k][i]=(h[k][i-1]+p*s[i-1])%MOD[k];
                pi[k][i]=(1LL* pi[k][i-1]* PI[k])%MOD[k];
                p=(p*P)%MOD[k];
                primos[k][i]=p;
            }
        }
    }
    ll get(int s, int e){
        ll h0=(h[0][e]-h[0][s]+MOD[0])%MOD[0];
        h0 =(1LL*h0*pi[0][s])%MOD[0];
        ll h1=(h[1][e]-h[1][s]+MOD[1])%MOD[1];
        h1 =(1LL*h1*pi[1][s])%MOD[1];
        return (h0<<32)|h1;
    }

    //devuelve el hash del string cambiando en la iesima posicion a c1
        por c2.
    // s: inicio del string, e: fin, i: posicion a cambiar, c1: old char,
         c2: new char
    ll get_change(int s, int e, int i, int c1, int c2){

        ll h0=(h[0][e]-h[0][s]+MOD[0])%MOD[0];
        h0 =(1LL*h0*pi[0][s])%MOD[0];
        h0 = ((h0 - c1*primos[0][i] % MOD[0]) + MOD[0])%MOD[0];
        h0 = (h0 + c2*primos[0][i])%MOD[0];

        ll h1=(h[1][e]-h[1][s]+MOD[1])%MOD[1];
        h1 =(1LL*h1*pi[1][s])%MOD[1];
```

```
        h1=( (h1 - c1*primos[1][i])%MOD[1] + MOD[1])%MOD[1];
        h1=(h1 + c2*primos[1][i])%MOD[1];
        return (h0<<32)|h1;
    }

    void set_change(int s, int e, int i, int c1, int c2) {
        for (int k = 0; k < 2; ++k) {
            h[k][e] = (h[k][e] - c1 * primos[k][i] % MOD[k] + MOD[k]) %
                MOD[k];
            h[k][e] = (h[k][e] + c2 * primos[k][i]) % MOD[k];
        }
    }
};

int main(){
    #ifdef EBUG
        freopen("input.txt", "r", stdin);
    #endif
    ios :: sync_with_stdio(false);
    cin.tie(NULL);
    cout.tie(NULL);


    int n;
    cin >> n;
    string s;
    cin >> s;
    int r, m;
    cin >> r >> m;

    unordered_map<ll,ll> tabla;
    string abecedario = "abcdefghijklmnopqrstuvwxyz,._";
    forn(i,r){
        string palabra;
        cin >> palabra;
        Hash p(palabra);
        tabla[p.get(0,palabra.size())]++;

        forn(j,m){
            forn(k, abecedario.size()){
                if(palabra[j] == abecedario[k])continue;
                tabla[p.get_change(0, m , j, palabra[j], abecedario[k])
                    ]++;
            }
        }

    }

    Hash secuencia(s);
    ll ans = 0;
    for(int i=0 ; i+m <= s.size() ; i++){
        ll valor = secuencia.get(i,i+m);

        if(tabla.count(valor)>0){
            ans += tabla[valor];
        }
    }
    cout << ans << endl;


    return 0;
}
```

7 Geometría

7.1 Punto

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

// si la entrada es entero cambiar a long long.
typedef long long T; // double could be faster but less precise
typedef long double ld;
const T EPS = 1e-9; // if T is integer, set to 0
const T INF = 1e18;

struct pto{
  T x, y;
    //constructor. pto a = (0,0)
  pto() : x(0), y(0) {}
  pto(T _x, T _y) : x(_x), y(_y) {}

    //operadores basicos
  pto operator+(pto b) { return pto(x+b.x, y+b.y); }
  pto operator-(pto b) { return pto(x-b.x, y-b.y); }
  pto operator+(T k) { return pto(x+k, y+k); }
  pto operator*(T k) { return pto(x*k, y*k); }
  pto operator/(T k) { return pto(x/k, y/k); }

  // dot product
    /*  a Âů b = |a| * |b| * cos(Îÿ)
        proy_a_b = ( (aÂůb) / |b|Âš ) * b
        a â§Ć b  âĞŤ  aÂůb = 0  (o |aÂůb| < EPS en flotantes)
    */
  T operator*(pto b) { return x*b.x + y*b.y; }

  // cross product
  T operator^(pto b) { return x*b.y - y*b.x; }
    /*
        |a ÃŮ b| = |a| * |b| * sin(Îÿ)
        > 0  âĘŠ  b estÃą a la izquierda de a  (rotaciÃşn CCW menor a 180 Âř)
        < 0  âĘŠ  b estÃą a la derecha de a    (rotaciÃşn CW menor a 180 Âř)
        = 0  âĘŠ  a y b son colineales.
        Es el Ãąrea con signo del paralelogramo formado por a y b.
        El Ãąrea del triÃąngulo definido por a y b es |a ÃŮ b| / 2.
    */
  // vector projection of this above b
  pto proj(pto b) { return b*((*this)*b) / (b*b); }

  T norm_sq() { return x*x + y*y; }
  ld norm() { return sqrtl(x*x + y*y); }
  ld dist(pto b) { return (b - (*this)).norm(); }

  //rotate by theta rads CCW w.r.t. origin (0,0)
  pto rotate(T ang) {
    return pto(x*cosl(ang) - y*sinl(ang), x*sinl(ang) + y*cosl(ang));
  }

  // true if this is at the left side of line ab
  bool left(pto a, pto b) { return ((a-*this) ^ (b-*this)) > 0; }
  bool operator<(const pto &b) const {
    return x < b.x-EPS || (abs(x - b.x) <= EPS && y < b.y-EPS);
  }
  bool operator==(pto b){ return abs(x-b.x)<=EPS && abs(y-b.y)<=EPS; }

};
pto perp(pto a) {return pto(-a.y, a.x);}

ld angle(pto a, pto o, pto b) {
  pto oa = a-o, ob = b-o;
  return atan2l(oa^ob, oa*ob);
}
ld angle(pto a, pto b) { // smallest angle bewteen a and b
  ld cost = (a*b) / a.norm() / b.norm();
  return acosl(max(ld(-1.), min(ld(1.), cost)));
}
```

7.2 Line

```
#include "pto.cpp"

int sgn(T x) { return x < 0 ? -1 : !!x; }
struct line {
  T a, b, c;  // Ax+By=C
  line() {}
  line(T a_, T b_, T c_) : a(a_), b(b_), c(c_) {}
  // TO DO: check negative C (multiply everything by -1)
  line(pto u, pto v) : a(v.y - u.y), b(u.x - v.x), c(a * u.x + b * u.y)
      {}
  int side(pto v) { return sgn(a * v.x + b * v.y - c); }
  bool inside(pto v) { return abs(a * v.x + b * v.y - c) <= EPS; }
  bool parallel(line v) { return abs(a * v.b - v.a * b) <= EPS; }
  pto inter(line v) {
    T det = a * v.b - v.a * b;
    if (abs(det) <= EPS) return pto(INF, INF);
    return pto(v.b * c - b * v.c, a * v.c - v.a * c) / det;
  }
};
```

7.3 Segment

```
#include "pto.cpp"
#include "line.cpp"

struct segment {
    pto s, e;
```

```
    segment(pto s_, pto e_) : s(s_), e(e_) {}

    //devuelve el punto del segmento mÃąs cercano a b
    pto closest(pto b) {
        pto bs = b - s, es = e - s;
        ld l = es * es;
        if (abs(l) <= EPS) return s;
        ld t = (bs * es) / l;
        if (t < 0.) return s;        // comment for lines
        else if (t > 1.) return e;  // comment for lines
        return s + (es * t);
    }
    bool inside(pto b) { //Return true if pto b is inside the segment
        return abs(s.dist(b) + e.dist(b) - s.dist(e)) < EPS;
    }

    // si los puntos son muy grandes puede dar overflow. Usar este inside
  bool inside2(pto b) {
    return ((s - b) ^ (e - b)) == 0 &&
        min(s.x, e.x) <= b.x && b.x <= max(s.x, e.x) &&
        min(s.y, e.y) <= b.y && b.y <= max(s.y, e.y);
  }

    pto inter(segment b) {  // if a and b are collinear, returns one
        point
        if ((*this).inside(b.s)) return b.s;
        if ((*this).inside(b.e)) return b.e;
        pto in = line(s, e).inter(line(b.s, b.e));
        if ((*this).inside(in) && b.inside(in)) return in;
        return pto(INF, INF);
    }


    // cuando no importa el punto
    bool intersects(segment b) {
        pto a1 = s, a2 = e, b1 = b.s, b2 = b.e;
        auto cross1 = (a2 - a1) ^ (b1 - a1);
        auto cross2 = (a2 - a1) ^ (b2 - a1);
        auto cross3 = (b2 - b1) ^ (a1 - b1);
        auto cross4 = (b2 - b1) ^ (a2 - b1);
        if ((cross1 > 0 && cross2 < 0 || cross1 < 0 && cross2 > 0) &&
            (cross3 > 0 && cross4 < 0 || cross3 < 0 && cross4 > 0))
            return true;
        return inside(b1) || inside(b2) || b.inside(a1) || b.inside(a2);
    }

};
```

7.4 Polar sort

```
#include "pto.cpp"

/*
funcionamiento:
    vector<pto> puntos = {pto(1, 2), pto(2, 1), pto(-1, -1), pto(0, 2)};
    pto referencia(1, 1); // punto de referencia
    sort(puntos.begin(), puntos.end(), Cmp(referencia));
*/

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

7.5 Convex Hull

```
#include "pto.cpp"
using namespace std;
#define pb push_back
#define dforn(i,n) for(int i=n-1; i>=0; i--)
#define fora(p, i, n) for(int i = p; i < n; i++)
#define forn(i, n) for(int i = 0; i < n; i++)

// returns convex hull of p in CCW order
// left must return >=0 to delete collinear points
vector<pto> CH(vector<pto>& p) {
    if (p.size() < 3) return p;  // edge case, keep line or point
    vector<pto> ch;
    sort(p.begin(), p.end());
    forn(i, p.size()) {  // lower hull
        while (ch.size() >= 2 && ch[ch.size() - 1].left(ch[ch.size() -
            2], p[i]))
            ch.pop_back();
        ch.pb(p[i]);
    }
    ch.pop_back();
    int k = ch.size();
    dforn(i, p.size()) {  // upper hull
        while (ch.size() >= k + 2 && ch[ch.size() - 1].left(ch[ch.size()
            - 2], p[i]))
            ch.pop_back();
        ch.pb(p[i]);
```

```
    }
    ch.pop_back();
    return ch;
}
```

7.6 Area poligono

```
#include "pto.cpp"
#define forn(i, n) for(int i = 0; i < n; i++)
double area(vector<pto> &p){//O(p.size())
  double area=0;
  forn(i, p.size()) area+=p[i]^p[(i+1)%p.size()];
  //if points are in clockwise order then area is negative
  return abs(area)/2;
  // si los puntos son enteros abs(area) es siempre entero
}
```

7.7 Poligono

```
#include "pto.cpp"
#include "line.cpp"
#include "circle.cpp"
#include "convex-hull.cpp"

#define sz(v) (int(v.size()))
#define forn(i, n) for(int i = 0; i < n; i++)
#define forr(i, a, b) for (int i = (a); i < (b); i++)

#define pb push_back

struct poly{
  vector<pto> pt;
  poly(){}
  poly(vector<pto> pt_) : pt(pt_) {}
  void delete_collinears() { // delete collinear points
    deque<pto> nxt; int len = 0;
    forn(i,sz(pt)) {
      if(len>1 && abs((pt[i]-nxt[len-2])^(nxt[len-1]-nxt[len-2])) <= EPS)
        nxt.pop_back(), len--;
      nxt.pb(pt[i]); len++;
    }
    if(len>2 && abs((nxt[1]-nxt[len-1])^(nxt[0]-nxt[len-1])) <= EPS)
      nxt.pop_front(), len--;
    if(len>2 && abs((nxt[len-1]-nxt[len-2])^(nxt[0]-nxt[len-2])) <= EPS)
      nxt.pop_back(), len--;
    pt.clear(); forn(i,sz(nxt)) pt.pb(nxt[i]);
  }

  // asegura sentido horario, elimina colineales, rota el vector para
      que pt[0] sea el primero
  void normalize() {
    delete_collinears();
    if(pt[2].left(pt[0], pt[1])) reverse(pt.begin(), pt.end()); //make it
        CW
    int n = sz(pt), pi = 0;
    forn(i, n)
      if(pt[i].x<pt[pi].x || (pt[i].x==pt[pi].x && pt[i].y<pt[pi].y))
        pi = i;
    rotate(pt.begin(), pt.begin()+pi, pt.end());
  }


  bool is_convex() { // delete collinear points first O(n)
    int N = sz(pt);
    if(N < 3) return false;
    bool isLeft = pt[0].left(pt[1], pt[2]);
    forr(i, 1, sz(pt))
      if(pt[i].left(pt[(i+1)%N], pt[(i+2)%N]) != isLeft)
        return false;
    return true;
  }

  // for convex or concave polygons
  // excludes boundaries, check it manually
  bool inside(pto p) { // O(n)
    bool c = false;
    forn(i, sz(pt)) {
      int j = (i+1)%sz(pt);
      if((pt[j].y>p.y) != (pt[i].y > p.y) &&
      (p.x < (pt[i].x-pt[j].x)*(p.y-pt[j].y)/(pt[i].y-pt[j].y)+pt[j].x))
        c = !c;
    }
    return c;
  }

  bool inside_convex(pto p) { // O(lg(n)) normalize first
    if(p.left(pt[0], pt[1]) || p.left(pt[sz(pt)-1], pt[0])) return false;
    int a = 1, b = sz(pt)-1;
    while(b-a > 1){
      int c = (a+b)/2;
      if(!p.left(pt[0], pt[c])) a = c;
      else b = c;
    }
    return !p.left(pt[a], pt[a+1]);
  }

  // cuts this along line ab and return the left side
  // (swap a, b for the right one)
  poly cut(pto a, pto b) { // O(n)
    vector<pto> ret;
    forn(i, sz(pt)) {
      ld left1 = (b-a)^(pt[i]-a), left2 = (b-a)^(pt[(i+1)%sz(pt)]-a);
      if(left1 >= 0) ret.pb(pt[i]);
      if(left1*left2 < 0)
        ret.pb(line(pt[i], pt[(i+1)%sz(pt)]).inter(line(a, b)));
    }
    return poly(ret);
```

```
  }


  // addition of convex polygons
  poly minkowski(poly p) { // O(n+m) n=|this|,m=|p|
    this->normalize(); p.normalize();
    vector<pto> a = (*this).pt, b = p.pt;
    a.pb(a[0]); a.pb(a[1]);
    b.pb(b[0]); b.pb(b[1]);
    vector<pto> sum;
    int i = 0, j = 0;
    while(i < sz(a)-2 || j < sz(b)-2) {
      sum.pb(a[i]+b[j]);
      T cross = (a[i+1]-a[i])^(b[j+1]-b[j]);
      if(cross <= 0 && i < sz(a)-2) i++;
      if(cross >= 0 && j < sz(b)-2) j++;
    }
    return poly(sum);
  }

    // busca el punto mas lejano en una direccion dada
  pto farthest(pto v) { // O(log(n)) for convex polygons
    if(sz(pt)<10) {
      int k=0;
      forr(i,1,sz(pt)) if(v * (pt[i] - pt[k]) > EPS) k = i;
      return pt[k];
    }
    pt.pb(pt[0]);
    pto a=pt[1] - pt[0];
    int s = 0, e = sz(pt)-1, ua = v*a > EPS;
    if(!ua && v*(pt[sz(pt)-2]-pt[0]) <= EPS){ pt.pop_back(); return pt
        [0];}
    while(1) {
      int m = (s+e)/2; pto c = pt[m+1]-pt[m];
      int uc = v*c > EPS;
      if(!uc && v*(pt[m-1]-pt[m]) <= EPS){ pt.pop_back(); return pt[m];}
      if(ua && (!uc || v*(pt[s]-pt[m]) > EPS)) e = m;
      else if(ua || uc || v*(pt[s]-pt[m]) >= -EPS) s = m, a = c, ua = uc;
      else e = m;
      assert(e > s+1);
    }
  }

    //hasta aca.

  ld inter_circle(circle c){ // area of intersection with circle
    ld r = 0.;
    forn(i,sz(pt)) {
      int j = (i+1)%sz(pt); ld w = c.inter_triangle(pt[i], pt[j]);
      if(((pt[j]-c.o)^(pt[i]-c.o)) > 0) r += w;
      else r -= w;
    }
    return abs(r);
  }
  // area ellipse = M_PI*a*b where a and b are the semi axis lengths
  // area triangle = sqrt(s*(s-a)(s-b)(s-c)) where s=(a+b+c)/2
  ld area(){ // O(n)
    ld area = 0;
    forn(i, sz(pt)) area += pt[i]^pt[(i+1)%sz(pt)];
    return abs(area)/ld(2);
  }
  // returns one pair of most distant points
  pair<pto,pto> callipers() { // O(n), for convex  poly, normalize first
    int n = sz(pt);
    if(n <= 2) return {pt[0], pt[1%n]};
    pair<pto,pto> ret = {pt[0], pt[1]};
    T maxi = 0; int j = 1;
    forn(i,sz(pt)) {
      while(((pt[(i+1)%n]-pt[i])^(pt[(j+1)%n]-pt[j]))<-EPS)j=(j+1)%sz(pt)
          ;
      if(pt[i].dist(pt[j]) > maxi+EPS)
        ret = {pt[i], pt[j]}, maxi = pt[i].dist(pt[j]);
    }
    return ret;
  }
  pto centroid(){ // (barycenter, mass center, needs float points)
    int n = sz(pt);
    pto r(0,0); ld t=0;
    forn(i,n) {
      r = r + (pt[i] + pt[(i+1)%n]) * (pt[i] ^ pt[(i+1)%n]);
      t += pt[i] ^ pt[(i+1)%n];
    }
    return r/t/3;
  }
};
// Dynamic convex hull trick (based on poly struct)
vector<poly> w;
void add(pto q) { // add(q), O(log^2(n))
  vector<pto> p = {q};
  while(!w.empty() && sz(w.back().pt) < 2*sz(p)){
    for(pto v : w.back().pt) p.pb(v);
    w.pop_back();
  }
  w.pb(poly(CH(p))); // CH = convex hull, must delete collinears
}
T query(pto v) { // max(q*v:q in w), O(log^2(n))
  T r = -INF;
  for(auto& p : w) r = max(r, p.farthest(v)*v);
  return r;
}
```

7.8 Puntos mas cercanos

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
#define dforn(i,n) for(int i=n-1; i>=0; i--)
#define forn(i, n) for(int i = 0; i < n; i++)
#define fori(i, n) for(int i = n - 1; i <= 0; i--)
#define forall(it, v) for (auto it = v.begin(); it != v.end(); it++)
```

```cpp
#define forr(i, a, b) for (int i = (a); i < (b); i++)
#define sz(v) (int(v.size()))
#define pb push_back

typedef long long T; // double could be faster but less precise
typedef long double ld;
//const T EPS = 1e-9; // if T is integer, set to 0
const T EPS = 0; // if T is integer, set to 0
const T INF = 1e18;

struct pto{
  T x, y;
    //constructor. pto a = (0,0)
  pto() : x(0), y(0) {}
  pto(T _x, T _y) : x(_x), y(_y) {}

    //operadores basicos
  pto operator+(pto b) { return pto(x+b.x, y+b.y); }
  pto operator-(pto b) { return pto(x-b.x, y-b.y); }
  pto operator+(T k) { return pto(x+k, y+k); }
  pto operator*(T k) { return pto(x*k, y*k); }
  pto operator/(T k) { return pto(x/k, y/k); }

  // dot product
    /*  a Âů b = |a| * |b| * cos(Îÿ)
        proy_a_b = ( (aÂůb) / |b|Âš ) * b
        a â§Ć b  âĞŤ  aÂůb = 0  (o |aÂůb| < EPS en flotantes)
    */
  T operator*(pto b) { return x*b.x + y*b.y; }

  // cross product
  T operator^(pto b) { return x*b.y - y*b.x; }
    /*
        |a ÃŮ b| = |a| * |b| * sin(Îÿ)
        > 0  âĘŠ  b estÃą a la izquierda de a  (rotaciÃşn CCW menor a 180 Âř)
        < 0  âĘŠ  b estÃą a la derecha de a    (rotaciÃşn CW menor a 180 Âř)
        = 0  âĘŠ  a y b son colineales.
        Es el Ãąrea con signo del paralelogramo formado por a y b.
        El Ãąrea del triÃąngulo definido por a y b es |a ÃŮ b| / 2.
    */
  // vector projection of this above b
  pto proj(pto b) { return b*((*this)*b) / (b*b); }

  T norm_sq() { return x*x + y*y; }
  ld norm() { return sqrtl(x*x + y*y); }
  ld dist(pto b) { return (b - (*this)).norm(); }
    ll dist2(pto b) { return (b - (*this)).norm_sq(); }

  //rotate by theta rads CCW w.r.t. origin (0,0)
  pto rotate(T ang) {
    return pto(x*cosl(ang) - y*sinl(ang), x*sinl(ang) + y*cosl(ang));
  }

  // true if this is at the left side of line ab
  bool left(pto a, pto b) { return ((a-*this) ^ (b-*this)) > 0; }
  bool operator<(const pto &b) const {
    return x < b.x-EPS || (abs(x - b.x) <= EPS && y < b.y-EPS);
  }
  bool operator==(pto b){ return abs(x-b.x)<=EPS && abs(y-b.y)<=EPS; }
};

struct CmpY {
    bool operator()(const pto& a, const pto& b) const {
        if (a.y == b.y) return a.x < b.x;
        return a.y < b.y;
    }
};

ll closest_pair(vector<pto>& pts) {
    sort(pts.begin(), pts.end(), [](auto &a, auto &b) {
        return a.x < b.x;
    });

    set<pto, CmpY> active;
    ll d = LLONG_MAX;
    int j = 0;

    for (int i = 0; i < (int)pts.size(); i++) {
        pto p = pts[i];

        while (j < i && (p.x - pts[j].x)*(p.x - pts[j].x) > d) {
            active.erase(pts[j]);
            j++;
        }

        // vecinos en [p.y - sqrt(d), p.y + sqrt(d)]
        ll lim = (ll) sqrtl((long double)d) + 1;
        pto low = {p.x, p.y - lim};
        pto high = {p.x, p.y + lim};

        auto itlow = active.lower_bound(low);
        auto ithigh = active.upper_bound(high);

        for (auto it = itlow; it != ithigh; ++it) {
            d = min(d, p.dist2(*it));
        }

        active.insert(p);
    }
    return d;
}

int main(){
    #ifdef EBUG
        freopen("input.txt", "r", stdin);
    #endif
    ios :: sync_with_stdio(false);
    cin.tie(NULL);
    cout.tie(NULL);

```

```
    int n;
    cin >> n;
    vector<pto> a(n);
    forn(i,n){
        cin >> a[i].x >> a[i].y;
    }
    cout << closest_pair(a) << endl;

    return 0;
}
```

7.9 Puntos mas lejanos

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
#define dforn(i,n) for(int i=n-1; i>=0; i--)
#define forn(i, n) for(int i = 0; i < n; i++)
#define fori(i, n) for(int i = n - 1; i <= 0; i--)
#define forall(it, v) for (auto it = v.begin(); it != v.end(); it++)
#define forr(i, a, b) for (int i = (a); i < (b); i++)
#define sz(v) (int(v.size()))
#define pb push_back


int main(){
    #ifdef EBUG
        freopen("input.txt", "r", stdin);
    #endif
    ios :: sync_with_stdio(false);
    cin.tie(NULL);
    cout.tie(NULL);

   int n;
    cin >> n;
    vector<ll> mn(4, LLONG_MAX), mx(4, LLONG_MIN);

    for (int i = 0; i < n; i++) {
        ll x, y; cin >> x >> y;

        ll vals[4] = {x+y, x-y, -x+y, -x-y};

        for (int k = 0; k < 4; k++) {
            mn[k] = min(mn[k], vals[k]);
            mx[k] = max(mx[k], vals[k]);
        }

        ll ans = 0;
        for (int k = 0; k < 4; k++) {
            ans = max(ans, mx[k] - mn[k]);
        }
        cout << ans << endl;
    }

    return 0;
}
```

7.10 Puntos enteros

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
#define dforn(i,n) for(int i=n-1; i>=0; i--)
#define forn(i, n) for(int i = 0; i < n; i++)
#define fori(i, n) for(int i = n - 1; i <= 0; i--)
#define forall(it, v) for (auto it = v.begin(); it != v.end(); it++)
#define forr(i, a, b) for (int i = (a); i < (b); i++)
#define sz(v) (int(v.size()))
#define pb push_back


// si la entrada es entero cambiar a long long.
typedef long long T; // double could be faster but less precise
typedef long double ld;
//const T EPS = 1e-9; // if T is integer, set to 0
const T EPS = 0; // if T is integer, set to 0
const T INF = 1e18;

struct pto{
  T x, y;
    //constructor. pto a = (0,0)
  pto() : x(0), y(0) {}
  pto(T _x, T _y) : x(_x), y(_y) {}

    //operadores basicos
  pto operator+(pto b) { return pto(x+b.x, y+b.y); }
  pto operator-(pto b) { return pto(x-b.x, y-b.y); }
  pto operator+(T k) { return pto(x+k, y+k); }
  pto operator*(T k) { return pto(x*k, y*k); }
  pto operator/(T k) { return pto(x/k, y/k); }

  // dot product
    /*  a Âů b = |a| * |b| * cos(Îÿ)
        proy_a_b = ( (aÂůb) / |b|Âš ) * b
        a â§Ć b  âĞŤ  aÂůb = 0  (o |aÂůb| < EPS en flotantes)
    */
  T operator*(pto b) { return x*b.x + y*b.y; }

  // cross product
  T operator^(pto b) { return x*b.y - y*b.x; }
    /*
        |a ÃŮ b| = |a| * |b| * sin(Îÿ)
        > 0  âĘŠ  b estÃą a la izquierda de a  (rotaciÃşn CCW menor a 180 Âř)
        < 0  âĘŠ  b estÃą a la derecha de a    (rotaciÃşn CW menor a 180 Âř)
        = 0  âĘŠ  a y b son colineales.
        Es el Ãąrea con signo del paralelogramo formado por a y b.
```

```
        El Ãąrea del triÃąngulo definido por a y b es |a ÃŰ b| / 2.
    */
  // vector projection of this above b
  pto proj(pto b) { return b*((*this)*b) / (b*b); }

  T norm_sq() { return x*x + y*y; }
  ld norm() { return sqrtl(x*x + y*y); }
  ld dist(pto b) { return (b - (*this)).norm(); }

  //rotate by theta rads CCW w.r.t. origin (0,0)
  pto rotate(T ang) {
    return pto(x*cosl(ang) - y*sinl(ang), x*sinl(ang) + y*cosl(ang));
  }

  // true if this is at the left side of line ab
  bool left(pto a, pto b) { return ((a-*this) ^ (b-*this)) > 0; }
  bool operator<(const pto &b) const {
    return x < b.x-EPS || (abs(x - b.x) <= EPS && y < b.y-EPS);
  }
  bool operator==(pto b){ return abs(x-b.x)<=EPS && abs(y-b.y)<=EPS; }
};

ll area(vector<pto> &p){//O(p.size())
  ll area=0;
  forn(i, p.size()) area+=p[i]^p[(i+1)%p.size()];
  //if points are in clockwise order then area is negative
  return abs(area)/2;
  // si los puntos son enteros abs(area) es siempre entero
}


int main(){
    #ifdef EBUG
        freopen("input.txt", "r", stdin);
    #endif
    ios :: sync_with_stdio(false);
    cin.tie(NULL);
    cout.tie(NULL);

    int n;
    cin >> n;
    vector<pto> a(n);
    forn(i,n){
        cin >> a[i].x >> a[i].y ;
    }
    ll B=0;
    forn(i,n){
        B += __gcd(abs(a[i].x - a[(i+1)%n].x), abs(a[i].y - a[(i+1)%n].y)
            );
    }
    ll A = area(a);

    cout << (A-(B/2)+1) << " " << B << endl;

    return 0;
}
```

7.11 Circle

```
#include <bits/stdc++.h>
#include "polarsort.cpp"

#define ll long long
using namespace std;
#define dforn(i,n) for(int i=n-1; i>=0; i--)
#define forn(i, n) for(int i = 0; i < n; i++)
#define fori(i, n) for(int i = n - 1; i <= 0; i--)
#define forall(it, v) for (auto it = v.begin(); it != v.end(); it++)
#define forr(i, a, b) for (int i = (a); i < (b); i++)
#define sz(v) (int(v.size()))
#define pb push_back

#define sqr(a) ((a)*(a))
pto perp(pto a){return pto(-a.y, a.x);}
line bisector(pto a, pto b){
    line l = line(a, b); pto m = (a+b)/2;
    return line(-l.b, l.a, -l.b*m.x+l.a*m.y);
}

struct circle{
    pto o; T r;
    circle(){}
    circle(pto a, pto b, pto c) {
        o = bisector(a, b).inter(bisector(b, c));
        r = o.dist(a);
    }
    bool inside(pto p) { return (o-p).norm_sq() <= r*r+EPS; }
    bool inside(circle c) { // this inside of c
        T d = (o - c.o).norm_sq();
        return d <= (c.r-r) * (c.r-r) + EPS;
    }
    // circle containing p1 and p2 with radius r
    // swap p1, p2 to get snd solution
    circle* circle2PtoR(pto a, pto b, T r_) {
        ld d2 = (a-b).norm_sq(), det = r_*r_/d2 - ld(0.25);
        if(det < 0) return nullptr;
        circle *ret = new circle();
        ret->o = (a+b)/ld(2) + perp(b-a)*sqrt(det);
        ret->r = r_;
        return ret;
    }
    pair<pto, pto> tang(pto p) {
        pto m = (p+o)/2;
        ld d = o.dist(m);
        ld a = r*r/(2*d);
        ld h = sqrtl(r*r - a*a);
        pto m2 = o + (m-o)*a/d;
        pto per = perp(m-o)/d;
        return make_pair(m2 - per*h, m2 + per*h);
    }
    vector<pto> inter(line l) {
        ld a = l.a, b = l.b, c = l.c - l.a*o.x - l.b*o.y;
        pto xy0 = pto(a*c/(a*a + b*b), b*c/(a*a + b*b));
```

```
        if(c*c > r*r*(a*a + b*b) + EPS) {
            return {};
        }else if(abs(c*c - r*r*(a*a + b*b)) < EPS) {
            return { xy0 + o };
        }else{
            ld m = sqrtl((r*r - c*c/(a*a + b*b))/(a*a + b*b));
            pto p1 = xy0 + (pto(-b,a)*m);
            pto p2 = xy0 + (pto(b,-a)*m);
            return { p1 + o, p2 + o };
        }
    }
    vector<pto> inter(circle c) {
        line l;
        l.a = o.x - c.o.x;
        l.b = o.y - c.o.y;
        l.c = (sqr(c.r)-sqr(r)+sqr(o.x)-sqr(c.o.x)+sqr(o.y)-sqr(c.o.y))
            /2.0;
        return (*this).inter(l);
    }
    ld inter_triangle(pto a, pto b) { // area of intersection with oab
        if(abs((o-a)^(o-b)) <= EPS) return 0.;
        vector<pto> q = {a}, w = inter(line(a,b));
        if(sz(w) == 2) forn(i,sz(w)) if((a-w[i])*(b-w[i]) < -EPS) q.pb(w[
            i]);
        q.pb(b);
        if(sz(q) == 4 && (q[0]-q[1])*(q[2]-q[1]) > EPS) swap(q[1], q[2]);
        ld s = 0;
        forn(i, sz(q)-1){
            if(!inside(q[i]) || !inside(q[i+1])) {
                s += r*r*angle((q[i]-o),q[i+1]-o)/T(2);
            }
            else s += abs((q[i]-o)^(q[i+1]-o)/2);
        }
        return s;
    }
};
vector<ld> inter_circles(vector<circle> c){
    vector<ld> r(sz(c)+1); // r[k]: area covered by at least k circles
    forn(i, sz(c)) {       // O(n^2 log n) (high constant)
        int k = 1;
        Cmp s(c[i].o);
        vector<pair<pto,int>> p = {
            {c[i].o + pto(1,0)*c[i].r, 0},
            {c[i].o - pto(1,0)*c[i].r, 0}};
        forn(j, sz(c)) if(j != i) {
            bool b0 = c[i].inside(c[j]), b1 = c[j].inside(c[i]);
            if(b0 && (!b1 || i<j)) k++;
            else if(!b0 && !b1) {
                vector<pto> v = c[i].inter(c[j]);
                if(sz(v) == 2) {
                    p.pb({v[0], 1}); p.pb({v[1], -1});
                    if(s(v[1], v[0])) k++;
                }
            }
        }
        sort(p.begin(), p.end(), [&](pair<pto,int> a, pair<pto,int> b) {
            return s(a.first,b.first); });
        forn(j,sz(p)) {
            pto p0 = p[j? j-1: sz(p)-1].first, p1 = p[j].first;
            ld a = angle(p0 - c[i].o, p1 - c[i].o);
            r[k]+=(p0.x-p1.x)*(p0.y+p1.y)/ld(2)+c[i].r*c[i].r*(a-sinl(a))
                /ld(2);
            k += p[j].second;
        }
    }
    return r;
}
```

8 DP

8.1 Game

```cpp
#include <bits/stdc++.h>
#include <cstdio>
#define ll long long
using namespace std;

int N, K, turno;
ll A[3000];
ll dp[3000][3000];

ll juegoOptimo(int inicio, int fin)
{
    if (inicio > fin)
        return 0;

    if (dp[inicio][fin] != -1)
        return dp[inicio][fin];

    if ((fin - inicio + 1) % 2 == turno)
        dp[inicio][fin] = max(A[inicio] + juegoOptimo(inicio + 1, fin), A
            [fin] + juegoOptimo(inicio, fin - 1));
    else
        dp[inicio][fin] = min(-A[inicio] + juegoOptimo(inicio + 1, fin),
            -A[fin] + juegoOptimo(inicio, fin - 1));

    return dp[inicio][fin];
}

int main(){
    freopen("input.txt", "r", stdin);

    cin >> N;
    memset(dp,-1, sizeof(dp));
    for (int i = 0; i < N; i++){
        cin >> A[i];
    }

    if(N%2==0) turno=0;
    else turno=1;

    cout << juegoOptimo(0,N-1) << endl;

    return 0;
}
```

8.2 Long common subsecuence

```cpp
/*
    Print one longest string that is a subsequence of both s and t.
    axyb
    abyxb
    output: axb
*/

#include <bits/stdc++.h>
using namespace std;
#define ll long long

string s,t;
int dp[3000][3000];

int subsecuencia(int i, int j){
    if( i == s.size() || j == t.size() ) return 0;

    if( dp[i][j] != -1 ) return dp[i][j];

    if ( s[i] == t[j] ) {
        dp[i][j] = 1 + subsecuencia(i+1,j+1);
        return dp[i][j];
    }
    else {
        dp[i][j] = max(subsecuencia(i+1,j), subsecuencia(i,j+1));
        return dp[i][j];
    }
}

string respuesta = "";
void sol(int i, int j){
    if(i == s.size() || j == t.size() ) return;

    if(s[i] == t[j]){
        respuesta += s[i] , sol(i+1, j+1);
    }else{
        if(dp[i+1][j] > dp[i][j+1]) sol(i+1, j);
        else sol(i, j+1);
    }
}

int main(){
    cin >> s >> t;
    memset(dp, -1, sizeof(dp));

    subsecuencia(0, 0);
    sol(0, 0);
    cout <<  respuesta << endl;

    return 0;
}
```

8.3 Matching mask

```cpp
#include <bits/stdc++.h>
using namespace std;
#define forn(i, n) for(int i = 0; i < n; i++)
```

```
#define fori(i, n) for(int i = n - 1; i <= 0; i--)
#define mos(v) forn(auto i : v) cout << i << " ";
#define pb push_back
typedef long long ll;
const int mod = 1e9+7;
int n;
ll dp[22][2097152];
ll a[22][22];

ll solve(int i, ll mask, int sum){
    if(i==n){
        if(sum==n){
            return 1;
        }
        else return 0;
    }
    if(dp[i][mask]!=-1) return dp[i][mask];

    ll ans = 0;
    forn(j,n){
        if(a[i][j]==1){
            ll aux = 0;
            aux = (1LL << j);
            if((aux&mask)==0){
                ans = ((ans%mod) + (solve(i+1,mask^aux,sum+1)%mod) %mod);
            }
        }
    }

    dp[i][mask] = ans;
    return dp[i][mask];
}

int main(){
    cin >> n;
    int valor;
    forn(i,n){
        forn(j,n){
            cin >> valor;
            a[i][j]= valor;
        }
    }
    forn(i,22){
        forn(j,2097152){
            dp[i][j]=-1;
        }
    }

    ll mask = 0;
    cout << (solve(0,mask,0)%mod) << endl;
}
```

8.4 DP rangos

```
#include <bits/stdc++.h>
using namespace std;
#define forn(i, n) for(int i = 0; i < n; i++)
#define fori(i, n) for(int i = n - 1; i <= 0; i--)
#define mos(v) forn(auto i : v) cout << i << " ";
#define ll long long
#define ld double
#define pb push_back
#define MAXN 410
int n;
ll dp[MAXN][MAXN];
vector<ll> A;
ll suma(int a, int b){
    if(a==0) return A[b];

    return (A[b]-A[a-1]);
}
ll sol(int a, int b){
    if(a==b) return 0;
    if(a>b) return 0;
    if(dp[a][b]!=-1) return dp[a][b];
    ll ans=1e18;
    for(int k=a;k<b;k++){
        ans = min(ans, sol(a,k) + sol(k+1,b) + suma(a,b));
    }

    return dp[a][b]=ans;
}

int main(){
    //freopen("input.txt", "r", stdin);
    cin >> n;
    forn(i,n){
        ll valor;
        cin >> valor;
        A.pb(valor);
    }
    for(ll i=1;i<n;i++) {
        A[i] += A[i-1];
    }
    memset(dp,-1, sizeof(dp));

    cout << sol(0,n-1) << endl;

}
```

9 Utils

9.1 Binary Search

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

bool ok(int a){
    // funcion que hace algo
    return a>5;
}
int bs(vector<int> &v, int val){
    int l = 0; // algo que siempre sea False
    int r = v.size() - 1; // algo que siempre sea True
    int mid = (l+r)/2;
    while((r-l)>1){
        mid = (l+r)/2;
        if(ok(v[mid])){
            r=mid;
        }else l=mid;
    }
    // FFFFFFFFF TTTTTT
    //         l r
    // r == l+1
    // l == ultimo que No cumple. Ultimo FALSE
    // r == primero que Si cumple. Primer TRUE

    return mid;
}
```

9.2 Sort

Ordenar un vector de pair por su segunda componente

```cpp
vector<pair<int, int>> v;

bool sortbysec(const pair<int,int> &a, const pair<int,int> &b){
  return (a.second < b.second);
}

sort(v.begin(), v.end(), sortbysec);
```

9.3 Cout para doubles

```cpp
cout << fixed << setprecision(20) << ans << endl;
```

9.4 Longest increasing subsequence

```cpp
#include <bits/stdc++.h>
using namespace std;
#define ll long long
ll INF = 1;

int lis(vector<ll> const& a) { //longest increasing subsequence
    int n = a.size();
    vector<ll> d(n+1, INF);
    d[0] = -INF;

    for (int i = 0; i < n; i++) {
        ll l = upper_bound(d.begin(), d.end(), a[i]) - d.begin();
        if (d[l-1] <= a[i] && a[i] <= d[l] && a[i] >= 0)
            d[l] = a[i];
    }
    ll ans = 0;
    for (int l = 0; l <= n; l++) {
        if (d[l] < INF)
            ans = l;
    }
    return ans-1;
}
```

9.5 Prev permutation

```cpp
#include <bits/stdc++.h>
using namespace std;
int main () {
    vector<int> dias = {1,2,3,4,5,6,7};
    vector<bool> mask(dias.size(), false);
    //combinaciones de D elementos.
    int d = 3;
    fill(mask.begin(), mask.begin() + d, true); //1110000

    do {
        for (int i = 0; i < dias.size(); ++i) {
            if (mask[i]) {
                // permutacion actual
                cout << dias[i] << " ";
            }
        }
        cout << endl;
    } while (prev_permutation(mask.begin(), mask.end()));

    /* Salida
        1 2 3
        1 2 4
        1 2 5
        1 2 6
        1 2 7
        1 3 4
        1 3 5
        1 3 6
```

```
        1 3 7
        ..
        5 6 7
    */
  return 0;
}
```

9.6 MO

```
#include <bits/stdc++.h>
using namespace std;
#define forn(i, n) for(int i = 0; i < n; i++)

#define MAXN 500010

int res = 0;
vector<int> v, r;
void add(int i){ //modificar
    r[v[i]]++;
    if(r[v[i]] == 1) res++;
}
void remove(int i){//modificar
    r[v[i]]--;
    if(r[v[i]] == 0) res--;
}

int get_ans(){//modificar
    return res;
}
int n,sq,nq; // array size, sqrt(array size), #queries
struct qu{int l,r,id;}; // O((n+nq)*sqrt(n)*update)
qu qs[MAXN];
int ans[MAXN]; // ans[i] = answer to ith query
bool qcomp(const qu &a, const qu &b){
    if(a.l/sq!=b.l/sq) return a.l<b.l;
    return (a.l/sq)&1?a.r<b.r:a.r>b.r;
}
void mos(){
    forn(i,nq)qs[i].id=i;
    sq=sqrt(n)+.5;
    sort(qs,qs+nq,qcomp);
    int l=0,r=0;
    forn(i,nq){
        qu q=qs[i];
        while(l>q.l)add(--l);
        while(r<q.r)add(r++);
        while(l<q.l)remove(l++);
        while(r>q.r)remove(--r);
        ans[q.id]=get_ans();
    }
}
```

9.7 Criba

Nros primos hasta maxp

```
#include <bits/stdc++.h>
using namespace std;
#define dprint(v) cerr << #v"=" << v << endl //;)
#define forr(i,a,b) for(int i=(a); i<(b); i++)
#define forn(i,n) forr(i,0,n)
#define forall(it,v) for(typeof(v.begin()) it=v.begin();it!=v.end();++it)
#define sz(c) ((int)c.size())
#define zero(v) memset(v, 0, sizeof(v))
typedef long long ll;
typedef pair<int,int> ii;

#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);
}


int main() {
  buscarprimos();
  cout << '{';
  bool first=true;
  forall(it, primos){
    if(first) first=false;
    else cout << ',';
    cout << *it;
    }
    cout << "};\n";
    return 0;
}
```

9.8 Subconjuntos

Subconjuntos distintos de un conjunto

```
#include <bits/stdc++.h>
#include <bitset>
#include <iostream>
#include <string>
using namespace std;
#define forn(i,n) for(ll i = 0; i < n; i++)
```

```cpp
#define fora(p, i,n) for(ll i = p; i < n; i++)
#define pb push_back
typedef long long ll;
#define MAXN 1000000010
#ifdef EBUG
//local
#else
//judge
#endif


void subconjuntos(vector<int> &nums, vector<int> &vacio, int pos) {
   if(pos==nums.size()){
        for(auto i: vacio){
            cout << i << " ";
        }
        cout << endl;
        return;
   }
   vacio.pb(nums[pos]);
   subconjuntos(nums, vacio, pos+1);
   vacio.pop_back();
   while(pos<nums.size()-1 && nums[pos+1]==nums[pos])pos++;
   subconjuntos(nums, vacio, pos+1);
}


int main(){
    #ifdef EBUG
        freopen("input.txt", "r", stdin);
    #endif
    ios :: sync_with_stdio(false);
    cin.tie(NULL);
    cout.tie(NULL);



    vector<int> nums{1,3,6,9};
    vector<int> vacio;
    subconjuntos(nums, vacio, 0);

    /*
        Salida:
            1 3 6 9
            1 3 6
            1 3 9
            1 3
            1 6 9
            1 6
            1 9
            1
            3 6 9
            3 6
            3 9
            3
            6 9
            6
            9
    */
    return 0;
}
```