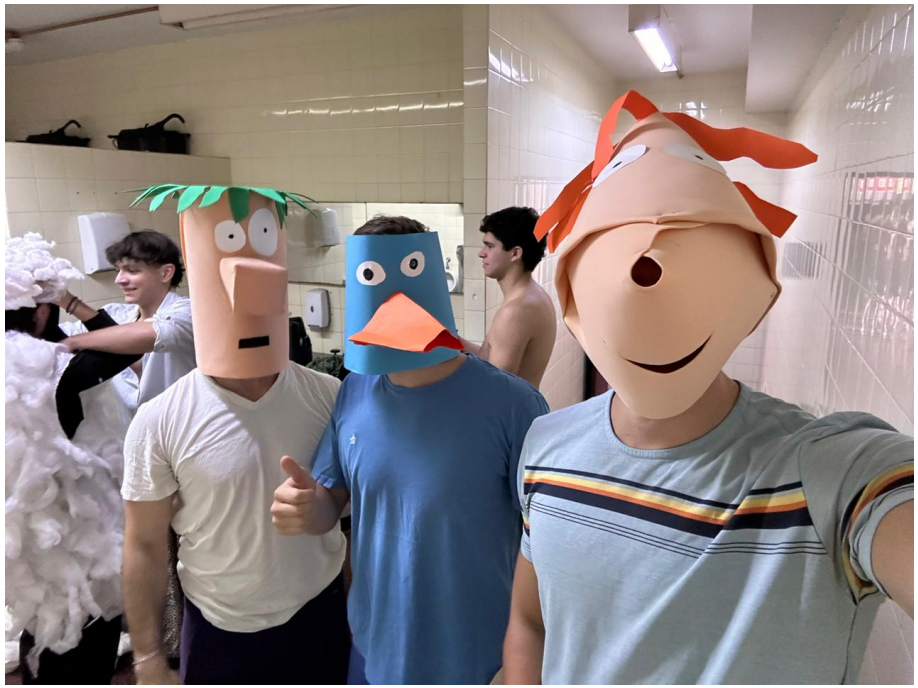
Notebook

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 forall(it, v) for (auto it = v.begin(); it != v.end(); it++)

#ifdef EBUG
// local
#else
// judge
#endif

using namespace std;

int main() {
#ifdef EBUG
    freopen("input.txt", "r", stdin);
#endif

    ios::sync_with_stdio(false);
    cin.tie(NULL);
    cout.tie(NULL);
    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 números	5: porque si divides 5:1=5 y ese número es un múltiplo o divisor de cualquier número.
2	El número termina en una cifra par.	378: porque la última cifra (8) es par.
3	La suma de sus cifras es un múltiplo de 3.	480: porque 4+8+0=12 es múltiplo de 3.
4	Sus últimos dos dígitos son 0 o un múltiplo de 4.	300 y 516 son divisibles entre 4 porque terminan en 00 y en 16, respectivamente, siendo este último un múltiplo de 4 (16=4*4).
5	La última cifra es 0 o 5.	485: porque termina en 5.
7	Un número es divisible entre 7 cuando, al separar la última cifra de la derecha, multiplicarla por 2 y restarla de las cifras restantes la diferencia es igual a 0 o es un múltiplo de 7. Otro sistema: Si la suma de la multiplicación de los números por la serie 2,3,1,-2,-3,-1... da 0 o un múltiplo de 7.	34349: separamos el 9, y lo duplicamos (18), entonces 3434-18=3416. Repetimos el proceso separando el 6 (341'6) y duplicándolo (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 múltiplo de 7. Ejemplo método 2: 34349: [(2*3)+(3*4)+(1*3)-(2*4)-(3*9)]= 6+12+3-8-27 = -14.8
8	Para saber si un número es divisible entre 8 hay que comprobar que sus tres últimas cifras sean divisibles entre 8. Si sus tres últimas cifras son divisibles entre 8 entonces el número también es divisible entre 8.	Ejemplo: El número 571.328 es divisible por 8 ya que sus últimas tres cifras (328) son divisibles por 8 (32 = 8*4 y 8 = 8*1). Realizando la división comprobamos que 571.328 : 8 = 71.416
9	Un número es divisible por 9 cuando al sumar todas sus cifras el resultado es múltiplo de 9.	504: sumamos 5+0+4=9 y como 9 es múltiplo de 9 504 es divisible por 9 5346: sumamos 5+3+4+6=18 y como 18 es múltiplo de 9, 5346 es divisible por 9.
10	La última cifra es 0.	4680: porque termina en 0
11	Sumando las cifras (del número) en posición impar por un lado y las de posición par por otro. Luego se resta el resultado de ambas sumas obtenidas. Si el resultado es cero o un múltiplo de 11, el número es divisible entre este. Si el número tiene solo dos cifras y estas son iguales será múltiplo de 11.	42702: 4+7+2=13 · 2+0=2 · 13-2=11 → 42702 es múltiplo de 11. 66: porque las dos cifras son iguales. Entonces 66 es múltiplo de 11.
13	Un número es divisible entre 13 cuando, al separar la última cifra de la derecha, multiplicarla por 9 y restarla de las cifras restantes la diferencia es igual a 0 o es un múltiplo de 13	3822: separamos el último dos (382'2) y lo multiplicamos por 9, 2×9=18, entonces 382-18=364. Repetimos el proceso separando el 4 (36'4) y multiplicándolo por 9, 4×9=36, entonces 36-36=0; por lo tanto, 3822 es divisible entre 13.
17	Un número es divisible entre 17 cuando, al separar la última cifra de la derecha, multiplicarla por 5 y restarla de las cifras restantes la diferencia es igual a 0 o es un múltiplo 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 número es divisible entre 19 si al separar la cifra de las unidades, multiplicarla por 2 y sumar a las cifras restantes el resultado es múltiplo 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 múltiplo de 19, luego 3401 también lo es.

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

2.4 Coprimos

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

3 Estructuras

3.1 vector

Función	**Explicación**	**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.

```
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.

```
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

Función	Explicación	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

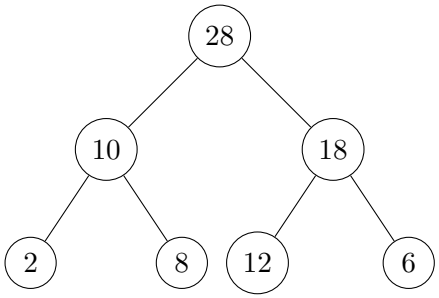
```cpp
vector<pair<int, int>> v;

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

También es posible ordenar un array normal:

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

3.4 Segment Tree




```cpp
#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;

mx.init(v.size()); forn(i, v.size()) mx[i] = v[i]; mx.updall();

ll searchST(int v, int tl, int tr,  int m){ //buscar el primer elemento mayor a un numero
   if( tl == tr){
       return tl;
   }
   int tm = (tl + tr)/2;
   if(t[2*v] >= m) return searchST(2*v, tl, tm, m);
   return searchST(2*v+1, tm+1, tr, m);

}
```

3.5 Array

```cpp
#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;
    }

}
```

3.6 BitSet

Función	Explicación	O
Template parameters		
N	número de bits	O(1)
Member functions		
(constructor)	construye el ‘bitset‘	O(N)
Element access		
operator[]	accede a un bit específico	O(1)
all, any, none	todos, algún o ningún bit está en true	O(N)
count	número de bits establecidos en true	O(N)
size	número de bits que contiene	O(1)
Modifiers		
operator&= operator—= operatorˆ= 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	representación en cadena	

4 Grafos

4.1 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.

4.1.1 DFS

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

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

4.1.2 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);
            }
        }
    }
}
```

4.2 Camino Mínimo

4.2.1 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;

void bellman_ford(int x){
    vector<int> distance(n, INF);
    distance[x] = 0;

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

4.2.2 Ciclos negativos

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

4.2.3 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;
    q.push({0,x});

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

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

4.2.4 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
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]);
        }
    }
}
```

4.3 Spanning Tree

4.3.1 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 tamaño 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;
```

4.3.2 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;
}
```

4.4 Aplicaciones comunes

4.4.1 Chequear si es conexo

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

4.4.2 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<bool> color(n, false);
    vector<bool> visited(n, false);

    forn(i, n){
        if (visited[i]) continue;
        q.push(i);
        while(!q.empty()){
            int n = q.front(); q.pop();
            visited[n] = true;

            for (int adj : G[n]) {
                if (!visited[adj]){
                    color[adj] = !color[n];
                    q.push(adj);
                }else if (color[adj] == color[n]){
                    cout << "IMPOSSIBLE\n";
                    return 0;
                }
            }
        }
    }

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

    return 0;
}
```

4.4.3 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;
}
```

4.4.4 Topological Sort

```cpp
#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
 * */

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;
}
```

4.4.5 Strong Connectivity

Para grafos dirigidos

```cpp
#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;
}
```

4.4.6 Max edge in path

Para grafos dirigidos

```cpp
#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]);
  }
}
```

4.5 Kosaraju

Encontrar todas las componentes fuertemente conexas en un grafo dirigido

```
#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());
    }
  }
};
```

5 Strings

5.1 Hash

```

#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;
}
```

6 Geometría

6.1 Punto

```
#include <bits/stdc++.h>
#define ll long long
#define ld double

struct pto {
    ll x, y;
    pto() : x(0), y(0) {} //Constructor pto a = pto(); ==> ) a.x = 0, a.y = 0
    pto(ll _x, ll _y) : x(_x), y(_y) {}
    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+(ll k) { return pto(x+k, y+k); }
    pto operator*(ll k) { return pto(x*k, y*k); }
    pto operator/(ll k) { return pto(x/k, y/k); }
    ll operator*(pto b) { return x*b.x + y*b.y; }
    pto proj(pto b) { return b*((*this)*b) / (b*b); }
    ll operator^(pto b) { return x*b.y - y*b.x; }
    ld norm() { return sqrt(x*x + y*y); }
    ld dist(pto b) { return (b - (*this)).norm(); }
};
```

6.2 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;
    }
};
```

6.3 Segment

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

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

    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;
    }

    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);
    }
};
```

6.4 Circle

```cpp
#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, pto(1,0));
        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.fst,b.fst); });
        forn(j,sz(p)) {
            pto p0 = p[j? j-1: sz(p)-1].fst, p1 = p[j].fst;
            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].snd;
        }
    }
    return r;
}
```

6.5 Polar sort

```
/*
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 DP

7.1 Game

```
#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;
}
```

7.2 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;
}
```

7.3 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;
}
```

7.4 DP rangos

```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 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;

}
```

8 Utils

8.1 Binary Search

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

int bs(vector<int> &v, int val){
    int l = 0, r = v.size() - 1, mid = (l+r)/2;
    while(l <= r){
        if(val < v[mid]){
            r = mid - 1;
        }else{
            l = mid + 1;
        }
        mid = (l+r)/2;
    }
    if(val < v[mid]){
        mid --;
    }

    return mid;
}
```

8.2 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);
```

8.3 Cout para doubles

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

8.4 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;
}
```

8.5 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();
    }
}
```