

Informe de los temas repasados en los meses de Junio y Julio por el equipo Codex

Juan David García Acevedo

Camilo Castañeda Yepes

Nicolás Ceballos Brito

Universidad Católica de Pereira

Ingeniería en sistemas y telecomunicaciones

Juan Carlos Blandón Andrade

Semillero de programación

1 de Agosto de 2023

## Programación Dinámica

La programación dinámica sirve para resolver problemas combinando las soluciones de los subproblemas superpuestos y subestructuras óptimas con el fin de reducir el tiempo de ejecución, normalmente es usada para resolver problemas de optimización.

PD: Técnica matemática que resuelve una serie de decisiones secuenciales, cada una de las cuales afecta las decisiones futuras.

La programación dinámica se puede dividir en dos tipos de algoritmos base los cuales son:

- ❖ **Memoización:** *top down*, para algoritmos recursivos. (+memoria -tiempo de ejecución).
- ❖ **Tabulación:** *bottom up*, para algoritmos iterativos. (la tabulación no aplica recursión por lo que es un poco más eficiente).

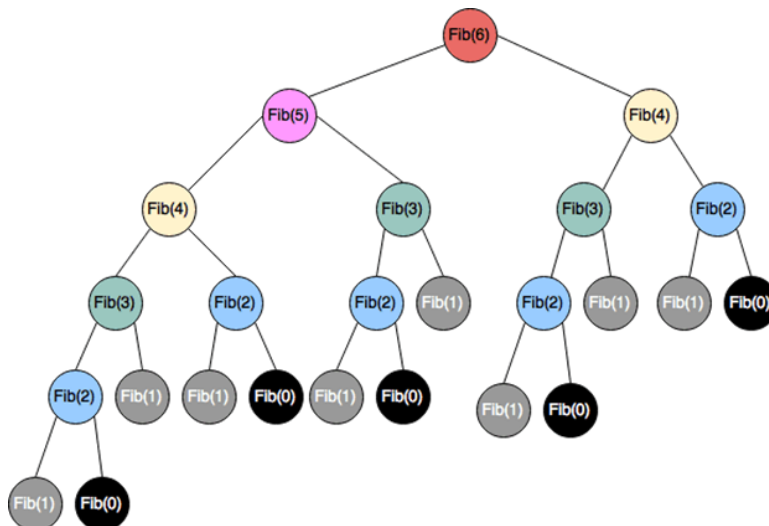
Las condiciones para poder implementar programación dinámica al resolver un problema son:

- **Subestructura óptima:** se trata de fraccionar un problema grande en varios pequeños para dar soluciones óptimas a esos subproblemas.
- **Superposición de problemas:** para tener una solución óptima se implicaría realizar muchas veces un mismo subproblema por lo que debe existir una superposición.

**Programación dinámica** = **recursión** + **memorización**

Ejercicios:

### 1. Sucesión de Fibonacci con recursividad.



## 2. Sucesión de Fibonacci con programación dinámica.

Ahora el algoritmo es mucho más eficiente, reduciendo enormemente el número de operaciones innecesarias para valores altos de  $n$ . Pero surge un nuevo problema. La complejidad espacial es de  $O(n)$  también. Hemos sacrificado espacio por tiempo, ahora tenemos un array de  $n$  posiciones cuando en realidad solo necesitábamos la posición  $(n)$  y las demás podemos desecharlas.

## 3. Sucesión de Fibonacci reduciendo costo espacial.

Como el algoritmo ÚNICAMENTE necesita de los dos valores anteriores para obtener el siguiente, podemos evitar el array, y así reducir su espacio ocupado.

Ahora la complejidad temporal sigue siendo de orden  $O(n)$  pero la complejidad espacial se reduce a un orden  $O(1)$ .

## 4. Fibonacci matemáticamente exacto.

Esta función es muy eficiente, pero su orden depende de la manera de implementar la potenciación. Una potenciación por cuadrados como en el caso anterior, arroja un costo de  $O(\log 2n)$ . Pero se debe tener mucho cuidado en su implementación, pues cada lenguaje tiene diferentes maneras de implementar las funciones matemáticas, y muy posiblemente sea necesario redondear el número para dar la respuesta exacta.

$$f_n = \frac{1}{\sqrt{5}} \left( \frac{1 + \sqrt{5}}{2} \right)^n - \frac{1}{\sqrt{5}} \left( \frac{1 - \sqrt{5}}{2} \right)^n$$

## 5. Fibonacci Modified.

### Fibonacci Modified

Implement a modified Fibonacci sequence using the following definition:

Given terms  $t[i]$  and  $t[i+1]$  where  $i \in (1, \infty)$ , term  $t[i+2]$  is computed as:

$$t(i+2) = t_i + t(i+2)^2$$

Given three integers, t1, t2, and n, compute and print the nth term of a modified Fibonacci sequence.

Example

- t1 = 0
- t2 = 1
- n = 6

t3 = 0+1^2 = 1  
t4 = 1+1^2 = 2  
t5 = 1+2^2 = 5  
t6 = 2+5^2 = 27  
Return 27.

Fibonacci Modified has the following parameter(s):

int t1: an integer

int t2: an integer

int n: the iteration to report

Returns

int: the nth number in the sequence

**Note:** The value of t[n] may far exceed the range of a 64-bit integer. Many submission languages have libraries that can handle such large results but, for those that don't (e.g., C++), you will need to compensate for the size of the result.

Input Format

A single line of three space-separated integers, the values of t1, t2, and n.

Constraints

- $0 \leq t1, t2 \leq 2$
- $3 \leq n \leq 20$
- tn may far exceed the range of a 64-bit integer.

## 6. Factorial estándar.

El factorial estándar es una operación matemática aplicada a un número entero no negativo “n”, denotado como “n!”. El factorial de un número "n" es el producto de todos los números enteros positivos desde 1 hasta "n". Matemáticamente, se expresa de la siguiente manera:

$$n! = n \times (n - 1) \times (n - 2) \times \dots \times 2 \times 1$$

Calcular el factorial estándar es un proceso sencillo que implica la multiplicación en serie de los números enteros en orden decreciente. Por ejemplo, el factorial de 5 (5!) se calcula como:

$$5 \times 4 \times 3 \times 2 \times 1 = 120.$$

## 7. Factorial “recursivo”.

El cálculo del factorial de forma recursiva implica definir la operación factorial en función de sí misma. En este enfoque, el factorial de un número “n” se expresa en términos del factorial de un número menor. Matemáticamente, esto se expresa como:

$$n! = n \times (n - 1)!$$

Este enfoque se basa en la idea de que el factorial de “n” puede ser calculado en términos del factorial de (n-1), lo que eventualmente llevará a la base del caso recursivo donde (0!=1). Sin embargo, este método puede llevar a una pila de llamadas recursivas profundas, lo que podría causar desbordamiento de pila en casos de números grandes.

## 8. Factorial recursivo (optimizado) Recursión de Cola.

Para abordar el problema de desbordamiento de pila en la recursión factorial, se puede utilizar la optimización de la recursión de cola. En este enfoque, se calcula el factorial de manera similar al método recursivo estándar, pero se realiza una acumulación en el proceso. Cada vez que se realiza una llamada recursiva, se actualiza el acumulador en lugar de generar una nueva llamada en la pila.

Esta técnica de recursión de cola elimina la necesidad de mantener múltiples llamadas recursivas en la pila de llamadas, evitando así el desbordamiento de pila en números grandes. La acumulación gradual del resultado a medida que se resuelve la recursión asegura una utilización eficiente de recursos.

## Grafos

**Los grafos son estructuras de datos que consisten en un conjunto de nodos (vértices) conectados entre sí mediante aristas. Son ampliamente utilizados en programación competitiva para resolver una variedad de problemas. Los grafos pueden ser explícitos, cuando se definen directamente mediante listas de adyacencia o matrices de adyacencia; o implícitos, cuando se representan en una matriz o cuadrícula.**

**Definición de 1 un grafo usando tuplas:** Se utiliza cuando se requiere información adicional de las aristas

```
typedef tuple < int, int, int > ti;
```

```
typedef vector < ti > edgeList;
```

```
int main ( ) {
```

```

int n, e;
cin >> n >> e;
while (e--) {
    int a, b, w;
    cin >> a >> b >> w;
    edgeList.push_back ( make_tuple ( a, b, w ) );
}
for (const auto& edge : edgeList) {
    int a, b, w;
    tie(a, b, w) = edge; // El tie se usa para desempaquetar los elementos de la tupla
//                               // Para poder imprimirlos
    cout << a << " " << b << " " << w << "\n";
}
return 0;
}

```

```

typedef pair <int, int> ii;
typedef vector<ii> vii;
vector <vii> ady;//Vector de vector de pares

int main(){
    int n, e;
    cin>>n>>e;
    //resize(), assign(), clear()
    ady.assign(n, vii(0,ii{0,0}));
    while(e--){//nosotros guardamos es las conexiones
        int a, b, w;
        cin>>a>>b>>w;
        //no dirigido
        // 1 2 3
        // a=1 b=2 w=3 weight
        ady[a].push_back(ii{b, w}); //1 --->2 = 3
        ady[b].push_back(ii{a, w}); //2 --- >1 = 3
        //dirigido
        //ady[a].push_back(ii{b, w});
    }
    for(int i=0; i<ady.size(); i++){
        //cout<<i<<": "<<ady[i].size()<<"\n";
        for(int j=0; j<ady[i].size(); j++){
            cout<<ady[i][j].first<<" = "<<ady[i][j].second<<"\n";
        }
    }

    return 0;
}

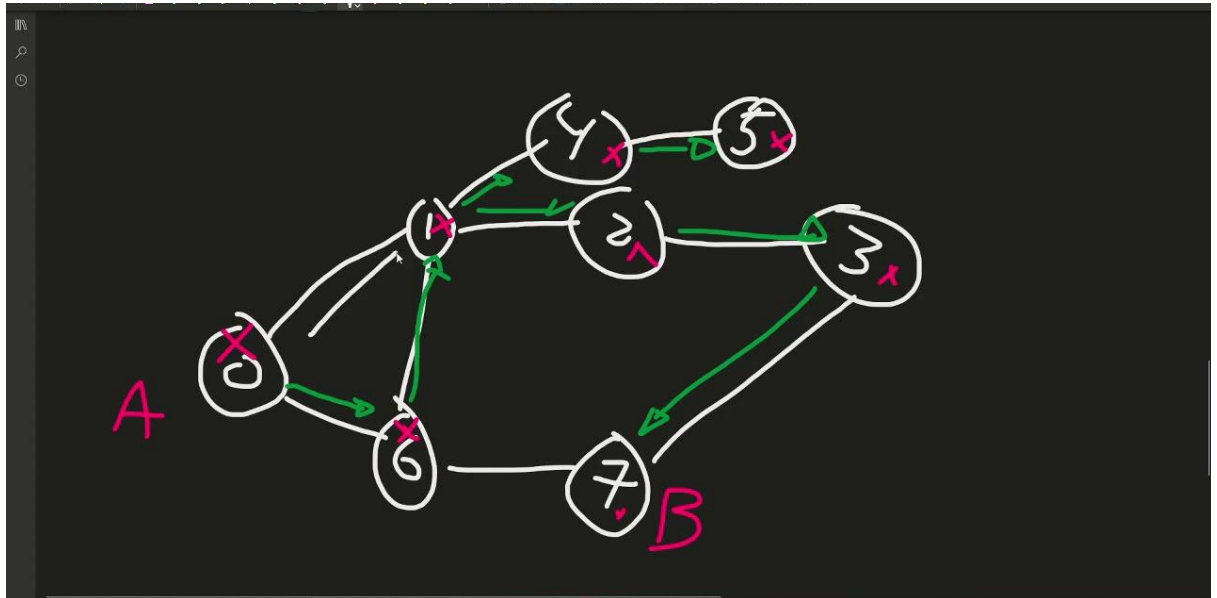
```

## Grafos Explícitos

### 0. Grafo Transversal:

- El Grafo Transversal es un concepto simple pero importante que consiste en visitar todos los nodos del grafo exactamente una vez. Una manera común de lograr esto es utilizando un algoritmo de búsqueda en profundidad (DFS) o búsqueda en anchura (BFS) para recorrer el grafo.

## 1. DFS (Búsqueda en Profundidad):



```

here x *DFS.cpp x
1  #include<bits/stdc++.h>
2  using namespace std;
3  #define INF -1
4  typedef pair<int,int>ii;
5  typedef vector<int> vi;
6  typedef vector<ii> vii;
7  vectot<vi> ady;
8  vi dist;
9  int origen, destino, n, e;
10 void dfs(int a){
11     dist[a]=1;
12     for(int i=0; i<ady[a].size();i++){
13         int b=ady[a][i];
14         if(dist[b]==INF)
15             dfs(b);
16     }
17 }
18 // CodeX Junio y Julio
19 int main(){

```

```

}
// CodeX Junio y Julio
int main() {
    int t;
    cin >> t;
    while (t--) {
        cin >> n >> e;
        if (n == 0 && e == 0)
            break;
        ady.assign(n, vi(0, 0));
        dist.assign(n, INF);

        int a, b;
        cin >> a >> b;
        ady[a].push_back(b);
        ady[b].push_back(a);

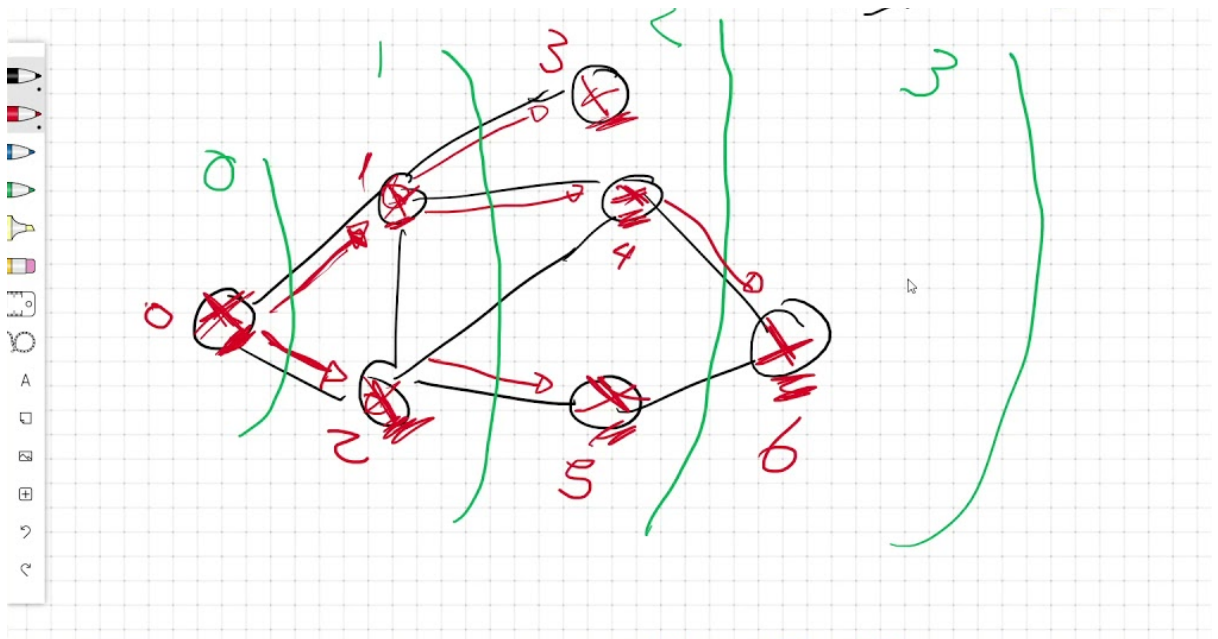
        dfs(a);
        int movements = 0;
        for (int i = 0; i < n; i++) {
            if (dist[i] == 1)
                movements++;
        }
        cout << movements << endl;
    }
    return 0;
}

```

- DFS es un algoritmo para recorrer todos los nodos de un grafo, explorando tan lejos como sea posible a lo largo de cada rama antes de retroceder. Esto se puede implementar mediante una función recursiva o mediante una pila explícita. Es útil para encontrar componentes conectados y puntos de articulación.

**BFS (Búsqueda en Anchura):**





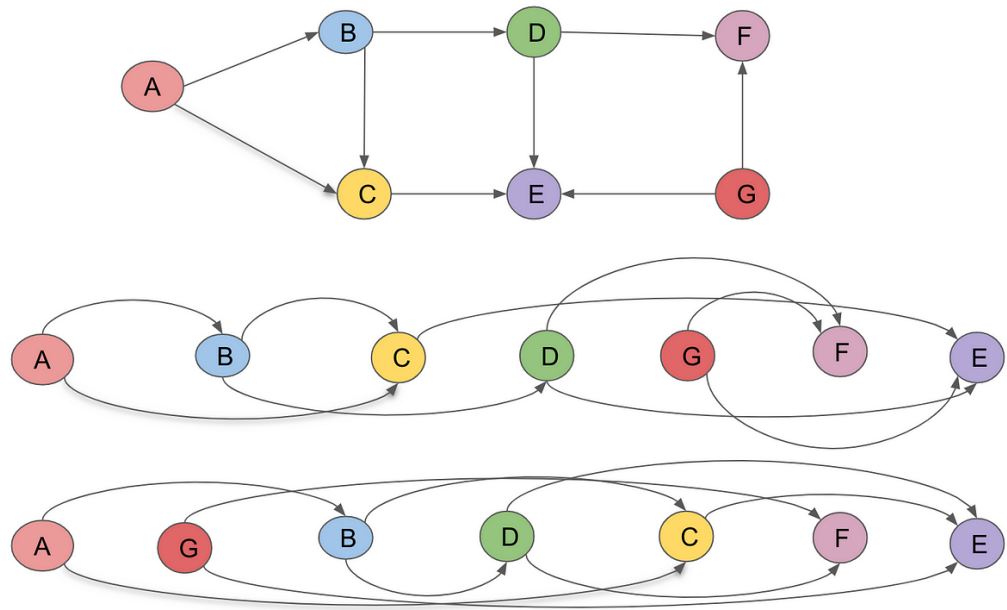
```

1  #include <bits/stdc++.h>
2  using namespace std;
3  #define INF -1
4  typedef vector<int> vi; //Vector base para adjList sin pesos
5  vector<vi> adjList;
6  vi dist;
7  int source, n, e;
8  void BFS(){
9      dist[source]=0; //El bfs guarda las distancias que hay del source a los demás nodos, por lo que el nodo de origen comienza mismo no hay dis
10     queue<int> q; q.push(source); //Iniciamos un 0, tenes en cuenta que la cola es FIFO, first in, first out
11     while(!q.empty()){
12         int u = q.front(); q.pop();
13         for(int j = 0; j < (int)adjList[u].size(); j++){
14             if(dist[j]!=INF) continue;
15             dist[j]=dist[u]+1;
16             q.push(j);
17         }
18     }
19 }
20 // CodeX Junior y Julio
21 int main(){
22     //int n, e;
23     cin>>n>>e;
24     adjList.assign(0, vi(0, 0));
25     dist.assign(n, INF);
26     source=0;
27     while(e--){
28         int a, b;
29         adjList[a].push_back(b);
30     }
31     BFS();
32     return 0;
33 }
34

```

- **BFS es un algoritmo para recorrer todos los nodos de un grafo, explorando los vecinos de un nodo antes de moverse al siguiente nivel de vecinos. Puede implementarse utilizando una cola explícita. Es útil para encontrar el camino más corto desde un nodo fuente hasta todos los demás nodos en grafos sin pesos.**

**Orden Topológico (Topological Sort):**



Topological Sort

```
#include <bits/stdc++.h>

using namespace std;

#define VISITED 1
#define UNVISITED -1
typedef vector <int> vi;
vector <vi> adjList;
vi inDegree; //controla el InDegree de cada nodo
vi topSort;
vi dfs_num;
vi inDegreeAux;

void topologicalBFS_queue(){
    queue<int> q; //controla el paso de los nodos y sus vecinos, variación del bfs
    for(int i=0; i<inDegree.size(); i++){
        if(inDegree[i]==0) q.push(i); //Se insertan los indegree 0 ya que son los primeros que se visitan
    }
    while(!q.empty()){//posible: q.size()
        int u = q.front(); q.pop();
        cout<<u<<" ";
        for(int v: adjList[u]){
            inDegree[v]--;
            if(inDegree[v]==0) q.push(v);
        }
    }
}

void topologicalBFS_Pqueue(){//nos da un topological_sort ordenado ascendentemente
    priority_queue<int, vector<int>, greater<int>>> q; //controla el paso de los nodos de manera ascendente y sus vecinos,
    for(int i=0; i<inDegree.size(); i++){
        if(inDegree[i]==0) q.push(i); //Se insertan los indegree 0 ya que son los primeros que se visitan
    }
    while(!q.empty()){//posible: q.size()
        int u = q.top(); q.pop();
        cout<<u<<" ";
        for(int v: adjList[u]){
            inDegree[v]--;
            if(inDegree[v]==0) q.push(v);
        }
    }
}

}
```

```

void topologicalDFS(int u){
    dfs_num[u]=VISITED;
    for(int j=0; j<adjList[u].size(); j++){
        int v = adjList[u][j];
        if(dfs_num[v]==UNVISITED){
            topologicalDFS(v);
        }
    }
    toposort.push_back(u);
}

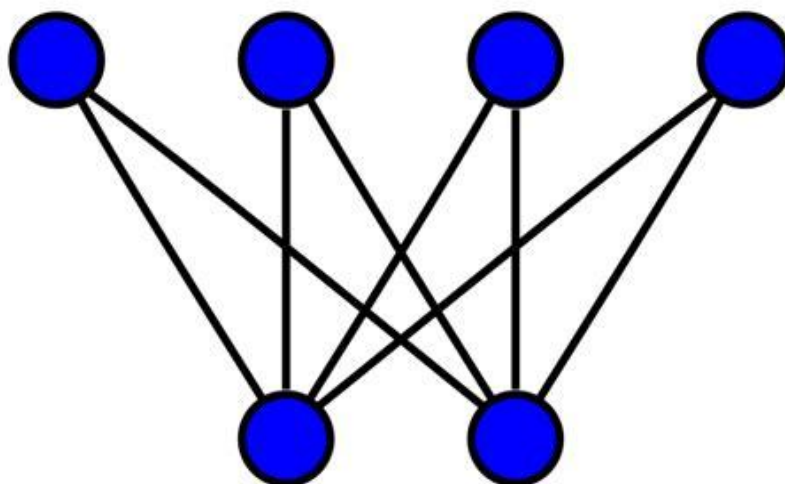
int main(){
    int n, e;
    cin>>n>>e;
    adjList.assign(n, vi());
    inDegree.assign(n, 0);
    inDegreeAux.assign(n, 0);
    dfs_num.assign(n, UNVISITED);
    while(e--){
        int a, b;
        cin>>a>>b;
        adjList[a].push_back(b);
        inDegree[b]++; //Controla el inDegree de cada nodo
        inDegreeAux[b]++; //fines prácticos
    }
    topologicalBFS_queue();
    cout<<"\n";
    inDegree.assign(inDegreeAux.begin(), inDegreeAux.end());
    topologicalBFS_Pqueue();
    cout<<"\n";
    for(int i=0; i<dfs_num.size(); i++){
        if(dfs_num[i]==UNVISITED) topologicalDFS(i);
    }
    reverse(toposort.begin(), toposort.end());
    //recorrido normal
    //sino, recorrerlo al revés
    for(int i=0; i<toposort.size(); i++) cout<<toposort[i]<<" ";
    cout<<"\n";

    return 0;
}

```

- El orden topológico es un orden lineal de los nodos de un grafo dirigido, donde para cada arista dirigida  $uv$ , el nodo  $u$  aparece antes que el nodo  $v$  en la secuencia. Se utiliza en problemas que requieren una secuencia de eventos o tareas.

**Grafo Bipartito (Bipartite Graph Check):**



- Un grafo es bipartito si sus nodos se pueden dividir en dos conjuntos disjuntos, de manera que todas las aristas vayan de un conjunto al otro. Se puede verificar usando DFS o BFS con marcado de colores.

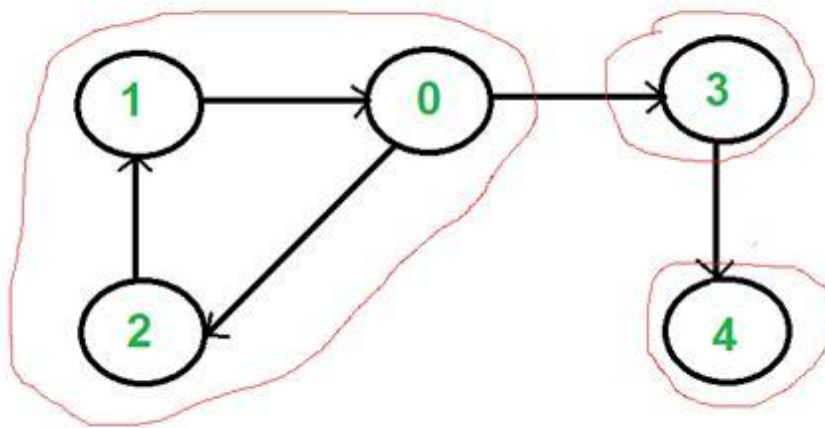
#### **Propiedad de Chequeo de Aristas de los Grafos mediante el Árbol de Expansión:**

- DFS puede utilizarse para encontrar árboles de expansión (como el árbol DFS), y esta propiedad se puede utilizar para clasificar las aristas de un grafo en tres categorías: aristas de avance, aristas de retroceso y aristas de cruce. Estas categorías son útiles para resolver problemas relacionados con grafos.

#### **Encontrar Puntos de Articulación y Puentes:**

- Un punto de articulación es un nodo cuya eliminación aumenta el número de componentes conectados en el grafo. Los puentes son aristas cuya eliminación aumenta el número de componentes conectados. Estos se pueden encontrar utilizando algoritmos de búsqueda en profundidad.

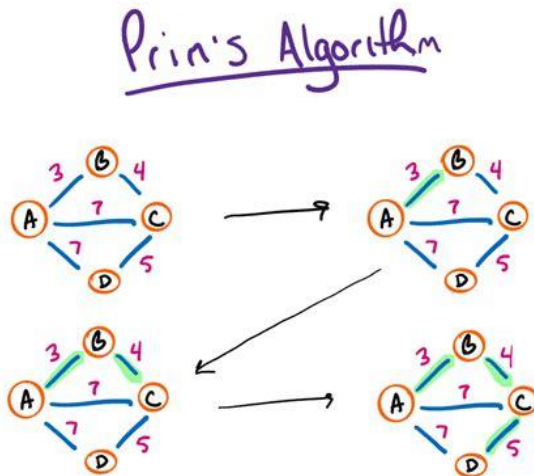
#### **Encontrando los Componentes Fuertemente Conectados (Strongly Connected Components - SCC):**



- Los SCC son subconjuntos de nodos de un grafo dirigido en los que cada nodo puede alcanzar a cualquier otro nodo dentro del subconjunto. El algoritmo de Kosaraju o el algoritmo de Tarjan se pueden usar para encontrar SCC.

### Árbol de Expansión Mínima (Minimum Spanning Tree - MST)

Algoritmo de Prim:



```

#include<bits/stdc++.h>
using namespace std;
typedef pair<int,int> ii;
typedef vector<ii> vii;
vector<vii> adjList;

int prim(const vector<vii>& adjList, int vI){//Enviar el grafo, Y el nodo inicial
    int n=adjList.size();
    vector<bool> eA (n, false);//arbol espacion minima
    vector<int> minDist (n, INT_MAX);
    priority_queue<ii, vii, greater<ii>> pq;
    int ans=0;
    minDist[vI]=0;//minimaDistancia
    pq.push(ii{0,vI});
    while(!pq.empty()){
        int actV = pq.top().second;//vertice actual
        int actP = pq.top().first;//peso actual
        pq.pop();
        if(eA[actV])continue;
        eA[actV]=true;
        ans += actP;
        for(const auto& con: adjList[actV]){
            int vertice = con.first;
            int peso = con.second;
            if(!eA[vertice] && peso < minDist[vertice]){//Si en el arbol eA[vertice] no es true. y peso es menor que la mīr
                minDist[vertice]=peso;
                pq.push(ii{peso,vertice});
            }
        }
    }
    return ans;
}

vector<int> Prim(const vector<vii>& adjList, int startNode) {
    int n = adjList.size(); // Número de nodos
    vector<int> dist(n, INT_MAX); // Distancias mínimas desde el MST
    vector<bool> visited(n, false); // Nodos visitados
    vector<int> parent(n, -1); // Padre de cada nodo en el MST
    // Usamos una cola de prioridad para seleccionar las aristas de menor peso
    priority_queue<ii, vector<ii>, greater<ii>> pq;
    dist[startNode] = 0; // La distancia al nodo inicial es 0
    pq.push(make_pair(0, startNode)); // Agregamos el nodo inicial a la cola
    while (!pq.empty()) {
        int u = pq.top().second; // Obtener el nodo de menor distancia
        pq.pop();
        visited[u] = true; // Marcar el nodo como visitado
        // Recorrer los vecinos del nodo actual
        for (const auto& neighbor : adjList[u]) {
            int v = neighbor.first;
            int weight = neighbor.second;
            // Si el vecino no ha sido visitado y la distancia es menor a la mínima actual
            if (!visited[v] && weight < dist[v]) {
                dist[v] = weight; // Actualizar la distancia mínima
                pq.push(make_pair(dist[v], v)); // Agregar el vecino a la cola de prioridad
                parent[v] = u; // Establecer el padre del vecino en el MST
            }
        }
    }
    return parent; // Retornar el arreglo de padres del MST
}

int main(){
    return 0;
}

```

```

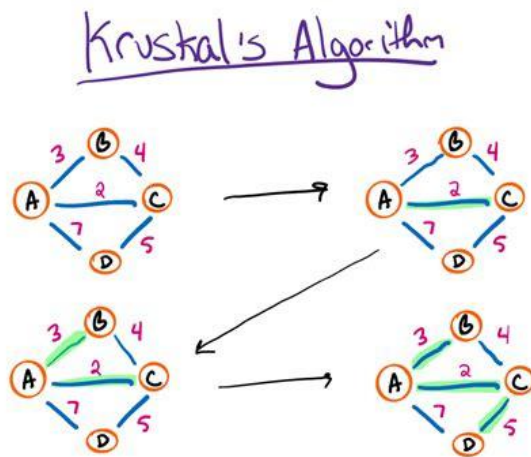
vector<int> Prim(const vector<vii>& adjList, int startNode) {
    int n = adjList.size(); // Número de nodos
    vector<int> dist(n, INT_MAX); // Distancias mínimas desde el MST
    vector<bool> visited(n, false); // Nodos visitados
    vector<int> parent(n, -1); // Padre de cada nodo en el MST
    // Usamos una cola de prioridad para seleccionar las aristas de menor peso
    priority_queue<ii, vector<ii>, greater<ii>> pq;
    dist[startNode] = 0; // La distancia al nodo inicial es 0
    pq.push(make_pair(0, startNode)); // Agregamos el nodo inicial a la cola
    while (!pq.empty()) {
        int u = pq.top().second; // Obtener el nodo de menor distancia
        pq.pop();
        visited[u] = true; // Marcar el nodo como visitado
        // Recorrer los vecinos del nodo actual
        for (const auto& neighbor : adjList[u]) {
            int v = neighbor.first;
            int weight = neighbor.second;
            // Si el vecino no ha sido visitado y la distancia es menor a la mínima actual
            if (!visited[v] && weight < dist[v]) {
                dist[v] = weight; // Actualizar la distancia mínima
                pq.push(make_pair(dist[v], v)); // Agregar el vecino a la cola de prioridad
                parent[v] = u; // Establecer el padre del vecino en el MST
            }
        }
    }
    return parent; // Retornar el arreglo de padres del MST
}

int main(){
    return 0;
}

```

- Prim es un algoritmo que encuentra un árbol de expansión mínima en un grafo no dirigido y ponderado. Comienza con un nodo arbitrario y expande el árbol al conectarse con el nodo más cercano.

Algoritmo de Kruskal:



Implementación de Kruskal Usando los Conjuntos Disjuntos también es posible:

```

#include<bits/stdc++.h>
using namespace std;
struct DS {
    vector<int> padre;
    DS(int tam) {
        padre.resize(tam);
        for (int i = 0; i < tam; i++) {
            padre[i] = i;
        }
    }
    int find(int a) {
        if (padre[a] != a) {
            padre[a] = find(padre[a]);
        }
        return padre[a];
    }
    void unionSets(int a, int b) {
        int raizA = find(a);
        int raizB = find(b);
        if (raizA != raizB) {
            padre[raizA] = raizB;
        }
    }
};

struct Edge {
    int origen;
    int destino;
    int pesoE;
    bool operator<(const Edge& other) const {
        return pesoE < other.pesoE;
    }
};

vector<Edge> kruskal(const vector<Edge>& edges, int numV) {
    vector<Edge> mst;
    DS ds(numV);
    vector<Edge> sorteado = edges;
    sort(sorteado.begin(), sorteado.end());
    for (const Edge& e : sorteado) {
        int raizOrigen = ds.find(e.origen);
        int raizDestino = ds.find(e.destino);
        if (raizOrigen != raizDestino) {
            mst.push_back(e);
            ds.unionSets(raizOrigen, raizDestino);
        }
    }
    return mst;
}

```



```

struct Edge {
    int origen;
    int destino;
    int pesoE;
    bool operator<(const Edge& other) const {
        return pesoE < other.pesoE;
    }
};

vector<Edge> kruskal(const vector<Edge>& edges, int numV) {
    vector<Edge> mst;
    DS ds(numV);
    vector<Edge> sorteado = edges;
    sort(sortado.begin(), sorteado.end());
    for (const Edge& e : sorteado) {
        int raizOrigen = ds.find(e.origen);
        int raizDestino = ds.find(e.destino);
        if (raizOrigen != raizDestino) {
            mst.push_back(e);
            ds.unionSets(raizOrigen, raizDestino);
        }
    }
    return mst;
}

int main(){
    int n, e, i = 0;
    while (cin >> n >> e && (n != 0 || e != 0)) {
        vector<Edge> edges(e);
        for (i = 0; i < e; i++) {
            cin >> edges[i].origen >> edges[i].destino >> edges[i].pesoE;
        }
        vector<Edge> mst = kruskal(edges, n);
    }
    return 0;
}

```

Otra versión de Kruskal más rápida es usando arreglos, y el mst como un entero, ejercicio de Beecrowd 1152:

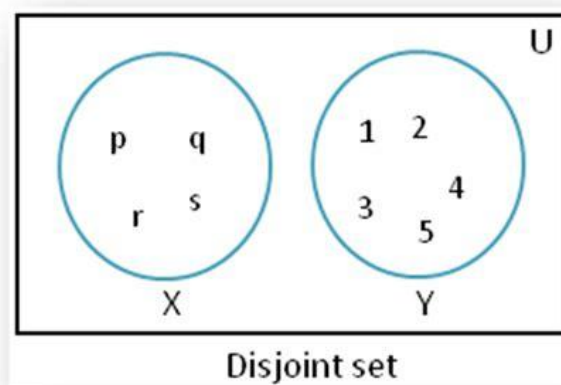
```

1  #include<bits/stdc++.h>
2  using namespace std;
3  #define INF 1000001
4  struct Edge{
5      int origen, destino;
6      int pesoE;
7  }edge[INF];
8  int padre[INF];
9  inline int find(int a);
0  int kruskal(int N, int E);
1  // CodeX Junio y Julio
2  int main(){
3      int kruskal(int N, int E){
4          for (int i=0; i<=N; i++)
5              padre[i] = i;
6          sort(edge, edge + E, [](const Edge& a, const Edge& b)->bool{return a.pesoE < b.pesoE;});
7          int e, cost = 0;
8          for (int i=0, e=0; i<E&&e<N-1; e++,i++){
9              while (find(edge[i].origen) == find(edge[i].destino)) i++;
10             padre[find(edge[i].origen)] = find(edge[i].destino);
11             cout<<"costAntes: "<<cost<<"\n";
12             cout<<"pesoEdgeAntes: "<<edge[i].pesoE<<"\n";
13             cost += edge[i].pesoE;
14             cout<<"costDespues: "<<cost<<"\n";
15             cout<<"pesoEdgeDespues: "<<edge[i].pesoE<<"\n";
16         }
17         return cost;
18     }
19     int find(int a){
20         return a == padre[a] ? a : (padre[a] = find(padre[a]));
21     }
22 }

```

- Kruskal es otro algoritmo para encontrar un árbol de expansión mínima en un grafo no dirigido y ponderado. Ordena las aristas por peso y las selecciona en orden ascendente, siempre que no formen un ciclo.

### Conjuntos Disjuntos:



- Para implementar los algoritmos de MST, es común utilizar la estructura de conjuntos disjuntos (Disjoint Set Union - DSU) para mantener un seguimiento de los componentes conexos y evitar ciclos.

## Camino más corto desde un Origen Único

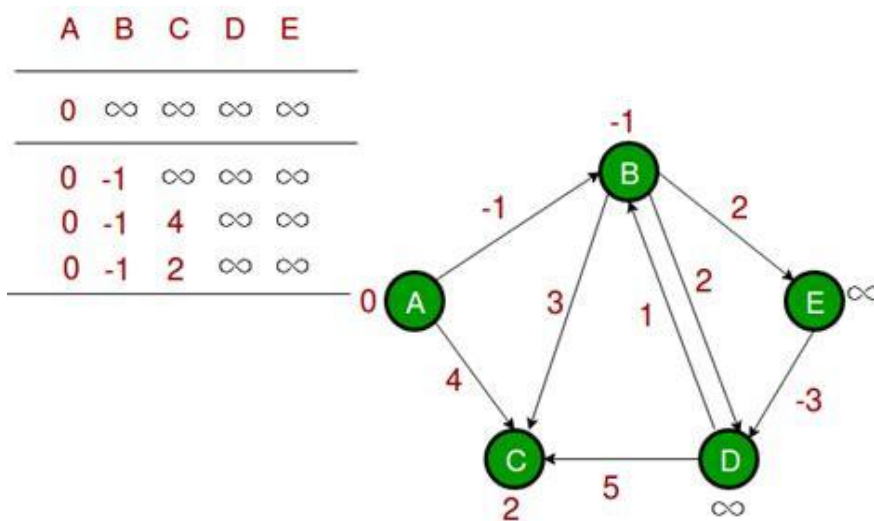
### SSSP en Grafos sin Pesos:

- En grafos sin pesos, el camino más corto desde un nodo fuente a cualquier otro nodo se puede encontrar utilizando BFS.

### SSSP en Grafos con Pesos:

- En grafos con pesos no negativos, el algoritmo de Dijkstra se puede usar para encontrar el camino más corto desde un nodo fuente a todos los demás nodos.

### Algoritmo de Bellman Ford:



```

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

#define INF INT_MAX
typedef vector<int> vi;
typedef pair<int,int> ii;
typedef vector<ii> vii;
vector<vii> ady;
vi dist;
int origen, destino, n, e;

void bellmanFord(int src) {
    dist.assign(n+1, INF);
    dist[src] = 0;
    bool chk = true;
    for(int i=1; i<=n && chk; i++) {
        chk = false;
        for(int u=1; u<=n; u++) {
            for(auto v : ady[u]) {
                int adj = v.first;
                int weight = v.second;
                if(dist[u] != INF && dist[u] + weight < dist[adj]) {
                    dist[adj] = dist[u] + weight;
                    chk = true;
                }
            }
        }
    }
}

int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    while(true) {
        cin >> n >> e;
        if(n == 0 && e == 0)
            break;
        ady.assign(n+1, vii());
        dist.assign(n+1, INF);
        origen = 0;
        destino = 0;
        while(e--) {
            int a, b, w;
            cin >> a >> b >> w;

```

```

int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    while(true) {
        cin >> n >> e;
        if(n == 0 && e == 0)
            break;
        ady.assign(n+1, vii());
        dist.assign(n+1, INF);
        origen = 0;
        destino = 0;
        while(e--) {
            int a, b, w;
            cin >> a >> b >> w;
            a--, b--;
            ady[a].push_back(ii(b, w));
            ady[b].push_back(ii(a, w));
        }
        int k;
        cin >> k;
        while(k--) {
            cin >> origen >> destino;
            origen--, destino--; // Adaptación para empezar en 0-index
            bellmanFord(origen);
            if(dist[destino] == INF) {
                cout << "Nao e possivel entregar a carta\n";
            }
            else {
                cout << dist[destino] << "\n";
            }
        }
        cout << "\n";
    }
    return 0;
}

```

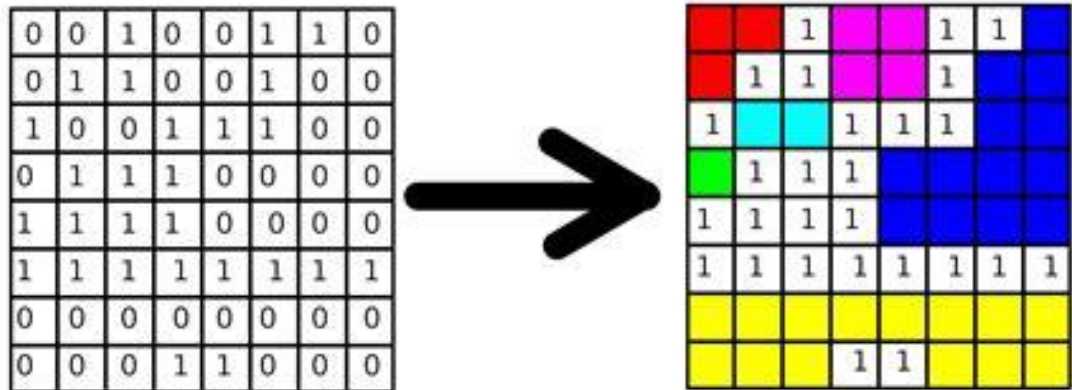
- Bellman Ford es un algoritmo que puede manejar grafos con aristas de peso negativo, pero también puede detectar ciclos negativos en el grafo.

#### Algoritmo de Floyd Warshall:

- Floyd Warshall es un algoritmo que encuentra los caminos más cortos entre todos los pares de nodos en un grafo, incluso si las aristas tienen pesos negativos.

## Grafos Implícitos

### FloodFill:



```
#include <bits/stdc++.h>

using namespace std;

int dr[]={1,1,0,-1,-1,-1,0,1}; //Controla la coordenada del inundamiento en las filas
int dc[]={0,1,1,1,0,-1,-1,-1}; //Controla la coordenada del inundamiento en la columnas
char grid[100][100]; //Matriz cuadrada, siempre dejar espacio de más, por ejemplo, si son 105
int R,C; //R=filas (rows), C=columnas (columns), controlar los limites de la matriz real

int floodfill(int r, int c, char c1, char c2){ //r= es la fila en donde se encuentra el puntero
//c1=char que se va a contar, buscar y reemplazar, c2=es el caracter por el que se va a reemplazar
if(r<0 || r>=R || c<0 || c>=C) return 0;
if(grid[r][c]!=c1) return 0;
int ans=1;
grid[r][c]=c2; //esta linea no se toca, porque queda infinito
for(int d=0; d<8; d++){ //control de la inundación mediante de los vectores
ans+=floodfill(r+dr[d], c+dc[d], c1, c2); //inundación recursiva mediante dfs
}
return ans;
}

int main(){
R=6;
C=4;
for(int i=0; i<R; i++){
for(int j=0; j<C; j++){
cin.get(grid[i][j]);
}
getchar();
}
cout<<"ANTES: \n";
for(int i=0; i<R; i++){
for(int j=0; j<C; j++){
cout<<grid[i][j];
}
cout<<"\n";
}
int aux=floodfill(0, 0, '*', '|');
cout<<"DESPUÉS: \n";
for(int i=0; i<R; i++){
for(int j=0; j<C; j++){
cout<<grid[i][j];
}
cout<<"\n";
}
}
```

- FloodFill es un algoritmo que se utiliza para rellenar áreas conectadas en una cuadrícula o matriz con el mismo color. Se puede implementar utilizando DFS o BFS en la cuadrícula.

Ejercicios:

## SPOJ: Sphere Online Judge

### Toposort - Topological Sorting

<https://www.spoj.com/problems/TOPOSORT/> topological sort

```
#include <bits/stdc++.h>
using namespace std;
typedef vector<int> vi;
vector<vi> ady;
vi inDegree;
vi toposort;

bool topologicalSort(int n) {
    priority_queue<int, vector<int>, greater<int>> pq;
    for (int i=1; i<=n; i++)
        if (inDegree[i] == 0)
            pq.push(i);
    while (!pq.empty()){
        int u = pq.top();
        pq.pop();
        toposort.push_back(u);
        for (int v : ady[u]){
            inDegree[v]--;
            if (inDegree[v] == 0) {
                pq.push(v);
            }
        }
    }
    return toposort.size() == n;
}

int main(){
    int n, e;
    cin >> n >> e;
    ady.resize(n + 1);
    inDegree.resize(n + 1);
    for (int i=0; i<e; i++){
        int a, b;
        cin >> a >> b;
        ady[a].push_back(b);
        inDegree[b]++;
    }
    if(!topologicalSort(n)){
        cout << "Sandro fails." << "\n";
    }else{
        for (int i=0; i<n; i++){
            cout << toposort[i] << " ";
        }
        cout << "\n";
    }
    return 0;
}
```

## CSES:

### Course Schedule

<https://cses.fi/problemset/task/1679> topological sort

```

#include <bits/stdc++.h>
using namespace std;
typedef vector<int> vi;
vector<vi> ady;
vi inDegree;
vi toposort;

bool topologicalSort(int n) {
    queue<int> q; //controla el paso de los nodos y sus vecinos, variación del bfs
    for(int i=1; i<inDegree.size(); i++){
        if(inDegree[i]==0) q.push(i); //Se insertan los indegree 0 ya que son los primeros que se visitan
    }
    while(!q.empty()){//posible: q.size()
        int u = q.front(); q.pop();
        toposort.push_back(u);
        for(int v: ady[u]){
            inDegree[v]--;
            if(inDegree[v]==0) q.push(v);
        }
    }
    return toposort.size() == n;
}

int main(){
    int n, e;
    cin >> n >> e;
    ady.resize(n + 1);
    inDegree.resize(n + 1);
    for (int i=0; i<e; i++){
        int a, b;
        cin >> a >> b;
        ady[a].push_back(b);
        inDegree[b]++;
    }
    if(!topologicalSort(n)){
        cout << "IMPOSSIBLE" << "\n";
    }else{
        for (int i=0; i<n; i++){
            cout << toposort[i] << " ";
        }
        cout << "\n";
    }
    return 0;
}

```

## Weird Algorithm

<https://cses.fi/problemset/task/1068> Operaciones Bitwise

```

#include <bits/stdc++.h>
using namespace std;
int main() {
    long long n;
    cin >> n;
    while (n != 1) {
        cout << n << " ";
        if (n & 1) {
            // n es impar
            n = (n << 1) + n + 1;
        } else {
            // n es par
            n >>= 1;
        }
    }
    cout << 1;
    return 0;
}

```



Beecrowds,

## 1026 - To Carry or not to Carry

<https://www.beecrowd.com.br/judge/en/problems/view/1026> Operaciones Bitwise

### SUBMISSION # 32456744

PROBLEM: 1026 - To Carry or not to Carry  
ANSWER: **Accepted**  
LANGUAGE: C++17 (g++ 7.3.0, -std=c++17 -O2 -lm) [+0s]  
RUNTIME: 0.378s  
FILE SIZE: 369 Bytes  
MEMORY: -  
SUBMISSION: 3/21/23, 9:20:48 PM

### SOURCE CODE

```
1 #include<bits/stdc++.h>
2 using namespace std;
3
4 int main(){
5     unsigned int x=0, y=0, suma=0;
6     while(cin>>x>>y){
7         suma = x ^ y;
8         cout << suma <<"\n";
9     }
10    return 0;
11 }
```

1152

<https://www.beecrowd.com.br/judge/en/problems/view/1152> Kruskal disjunto o MST

```

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

typedef pair<int, int> ii;
typedef vector<ii> vii;

struct DS {
    vector<int> padre;
    DS(int tam) {
        padre.resize(tam);
        for (int i = 0; i < tam; i++) {
            padre[i] = i;
        }
    }

    int find(int a) {
        if (padre[a] != a) {
            padre[a] = find(padre[a]);
        }
        return padre[a];
    }

    void unionSets(int a, int b) {
        int raizA = find(a);
        int raizB = find(b);
        if (raizA != raizB) {
            padre[raizA] = raizB;
        }
    }
};

struct Edge {
    int origen;
    int destino;
    int pesoE;
    bool operator<(const Edge& other) const {
        return pesoE < other.pesoE;
    }
};

vector<Edge> kruskal(const vector<Edge>& edges, int numV) {
    vector<Edge> mst;
    DS ds(numV);
    vector<Edge> sorteado = edges;

```

```

vector<Edge> kruskal(const vector<Edge>& edges, int numV) {
    vector<Edge> mst;
    DS ds(numV);
    vector<Edge> sorteado = edges;
    sort(sorteado.begin(), sorteado.end());
    for (const Edge& e : sorteado) {
        int raizOrigen = ds.find(e.origen);
        int raizDestino = ds.find(e.destino);
        if (raizOrigen != raizDestino) {
            mst.push_back(e);
            ds.unionSets(raizOrigen, raizDestino);
        }
    }
    return mst;
}

int main() {
    int n, e, i = 0;
    while (cin >> n >> e && (n != 0 || e != 0)) {
        vector<Edge> edges(e);
        for (i = 0; i < e; i++) {
            cin >> edges[i].origen >> edges[i].destino >> edges[i].pesoE;
        }
        vector<Edge> mst = kruskal(edges, n);
        int pesoTotal = 0;
        for (const Edge& edge : mst) {
            pesoTotal += edge.pesoE;
        }
        int ahorro = 0;
        for (const Edge& edge : edges) {
            ahorro += edge.pesoE;
        }
        int gobiernoSaving = ahorro - pesoTotal;
        cout << gobiernoSaving << "\n";
    }
    return 0;
}

```

```

#include<bits/stdc++.h>
using namespace std;
#define INF 1000001
struct Edge{
    int origen, destino;
    int pesoE;
}edge[INF];
int padre[INF];
inline int find(int a);
int kruskal(int N, int E);
int main(){
    int m, n;
    while (cin>>m>>n && (m != 0 || n != 0)) {
        for (int i=0; i<n; i++) {
            cin >> edge[i].origen >> edge[i].destino >> edge[i].pesoE;
        }

        long long pesoTotal = kruskal(m, n);
        cout<<"peso:"<<pesoTotal<< "\n";
        long long ahorro = 0;
        for (int i = 0; i < n; i++) {
            cout<<"ahorrosAntes:"<<ahorro<<"\n";
            ahorro += edge[i].pesoE;
            cout<<"ahorrosADespués:"<<ahorro<<"\n";
        }
        cout<<"edge:\n";
        for (int i=0; i<n; i++) {
            cout<<"\npara i:"<<i<<"\n";
            cout<<"origen: "<< edge[i].origen <<"\ndestino: "<<edge[i].destino <<"\npesoE: "<< edge[i].pesoE<<"\n";
        }
        cout<<"padre:\n";
        for (int i=0; i<n; i++) {
            cout<<"para i:"<<padre[i]<<"\n";
        }
        long long ans = ahorro - pesoTotal;
        cout << ans << "\n";
    }

    return 0;
}
int kruskal(int N, int E){
    for (int i=0; i<=N; i++)
        padre[i] = i;
    sort(edge, edge + E, [](const Edge& a, const Edge& b)->bool{return a.pesoE < b.pesoE;});
    int e, cost = 0;
    for (int i=0, e=0; i<E&&e<N-1; e++,i++){
        while (find(edge[i].origen) == find(edge[i].destino)) i++;
        padre[find(edge[i].origen)] = find(edge[i].destino);
        cout<<"costAntes: "<<cost<< "\n";
        cout<<"pesoEdgeAntes: "<<edge[i].pesoE<< "\n";
        cost += edge[i].pesoE;
        cout<<"costDespués: "<<cost<< "\n";
        cout<<"pesoEdgeDespués: "<<edge[i].pesoE<< "\n";
    }
    return cost;
}
int find(int a){
    return a == padre[a] ? a : (padre[a] = find(padre[a]));
}

```

Arista {0, 3, 5} (origen: 0, destino: 3, peso: 5)

Arista {2, 4, 5} (origen: 2, destino: 4, peso: 5)

Arista {3, 5, 6} (origen: 3, destino: 5, peso: 6)

Arista {1, 4, 7} (origen: 1, destino: 4, peso: 7)

Arista {0, 1, 7} (origen: 0, destino: 1, peso: 7)

Esta es la iteración mágica : Arista {1, 3, 9} (origen: 1, destino: 3, peso: 9)

La arista {0, 1, 7} forma un ciclo con las aristas {0, 3, 5} y {3, 5, 6} entonces no se agregan al mst.

Yo estuve desde ayer sentado sacando eso en el cuaderno mío y apenas di con él porque daba esa iteración tan rara la 39, me depure la lógica y estaba en lo correcto yo.

Arista  $\{4, 5, 8\}$ : Esta arista conecta los vértices 4 y 5 y ya existe una conexión entre estos vértices en el MST:

Vértice 4 ya está conectado con el vértice 1 en el MST.

Vértice 5 ya está conectado con el vértice 3 en el MST.

Entonces, agregar esta arista formaría un ciclo en el MST y no se incluiría.

Arista  $\{4, 6, 9\}$ : Esta arista conecta los vértices 4 y 6 y ya existe una conexión entre estos vértices en el MST:

Vértice 4 ya está conectado con el vértice 1 en el MST.

Vértice 6 no está conectado a ningún vértice en el MST.

Agregar esta arista no formaría un ciclo en el MST, pero el peso de esta arista (9) es mayor que el peso de la arista  $\{1, 3, 9\}$  (que ya está en el MST y tiene un peso de 9). Entonces, no se incluye esta arista.

Arista  $\{5, 6, 11\}$ : Esta arista conecta los vértices 5 y 6 y ya existe una conexión entre estos vértices en el MST:

Vértice 5 ya está conectado con el vértice 3 en el MST.

Vértice 6 no está conectado a ningún vértice en el MST.

Agregar esta arista no formaría un ciclo en el MST, pero el peso de esta arista (11) es mayor que el peso de la arista  $\{1, 3, 9\}$  (que ya está en el MST y tiene un peso de 9). Entonces, no se incluye esta arista.

En resumen, las aristas restantes ( $\{4, 5, 8\}$ ,  $\{4, 6, 9\}$ ,  $\{5, 6, 11\}$ ) no se incluyen en el MST porque formarían ciclos o tienen pesos más altos en comparación con las aristas ya presentes en el MST.

Este tiene que ser de los grafos más sencillos pero enredados hasta ahora.

**1148**

<https://www.beecrowd.com.br/judge/en/problems/view/1148> Dijkstra

```

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

#define INF INT_MAX
typedef vector<int> vi;
typedef pair<int,int> ii;
typedef vector<ii> vii;
vector<vii> ady;
vi dist;
int origen, destino, n, e;

void bellmanFord(int src) {
    dist.assign(n+1, INF);
    dist[src] = 0;
    bool chk = true;
    for(int i=1; i<=n && chk; i++) {
        chk = false;
        for(int u=1; u<=n; u++) {
            for(auto v : ady[u]) {
                int adj = v.first;
                int weight = v.second;
                if(dist[u] != INF && dist[u] + weight < dist[adj]) {
                    dist[adj] = dist[u] + weight;
                    chk = true;
                }
            }
        }
    }
}

int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    while(true) {
        cin >> n >> e;
        if(n == 0 && e == 0)
            break;
        ady.assign(n+1, vii());
        dist.assign(n+1, INF);
        origen = 0;
        destino = 0;
        while(e--) {
            int a, b, w;
            cin >> a >> b >> w;

```

```

int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    while(true) {
        cin >> n >> e;
        if(n == 0 && e == 0)
            break;
        ady.assign(n+1, vii());
        dist.assign(n+1, INF);
        origen = 0;
        destino = 0;
        while(e--) {
            int a, b, w;
            cin >> a >> b >> w;
            a--, b--;
            ady[a].push_back(ii(b, w));
            ady[b].push_back(ii(a, w));
        }
        int k;
        cin >> k;
        while(k--) {
            cin >> origen >> destino;
            origen--, destino--; // Adaptación para empezar en 0-index
            bellmanFord(origen);
            if(dist[destino] == INF) {
                cout << "Nao e possivel entregar a carta\n";
            }
            else {
                cout << dist[destino] << "\n";
            }
        }
        cout << "\n";
    }
    return 0;
}

```

Otra solución es con el algoritmo Dijkstra:

```

#include<bits/stdc++.h>
using namespace std;
#define INF 1001001001
typedef pair<int, int> ii;

int dijkstra(vector<ii> ady[], int origen, int destino, int tam){
    priority_queue<ii> pq;
    int dist[tam + 1];
    fill(dist, dist+tam, INF);
    dist[origen] = 0;
    pq.push(ii(0,origen));
    while(!pq.empty()){
        int a = pq.top().first;
        int b = pq.top().second;
        pq.pop();
        if(a != dist[b])continue;
        for(int j = 0; j < (int)ady[b].size(); j++){
            ii v = ady[b][j];
            int peso = v.first;
            int adj = v.second;
            if(a + peso < dist[adj]){
                dist[adj] = a+peso;
                pq.push(ii(dist[adj], adj));
            }
        }
    }
    return dist[destino] == INF ? -1 : dist[destino];
}

int main(){
    int n, e;
    while((cin >> n >> e) && n){
        vector<ii> graph[n];
        bool has_path[502][502] = {{},{}};
        for(int i=0; i<e; i++){
            int a, b, w;
            cin >> a >> b >> w;
            a--, b--;
            graph[a].push_back(ii(w, b));
            has_path[a][b] = true;
            if(has_path[b][a]){
                graph[a][((int)graph[a].size()-1).first] = 0;
                int j = 0;
                while(graph[b][j].second != a) j++;
                graph[b][j].first = 0;
            }
        }
    }
}

```



```

int main(){
    int n, e;
    while((cin >> n >> e) && n){
        vector<ii> graph[n];
        bool has_path[502][502] = {{},{}};
        for(int i=0; i<e; i++){
            int a, b, w;
            cin >> a >> b >> w;
            a--, b--;
            graph[a].push_back(ii(w, b));
            has_path[a][b] = true;
            if(has_path[b][a]){
                graph[a][((int)graph[a].size()-1).first] = 0;
                int j = 0;
                while(graph[b][j].second != a) j++;
                graph[b][j].first = 0;
            }
        }
        int t;
        cin >> t;
        while(t--){
            int a, b;
            cin >> a >> b;
            a--, b--;
            int ans = dijkstra(graph, a, b, n);
            (ans == -1) ? cout << "Nao e possivel entregar a carta\n" :
                        cout << ans << endl;
        }
        puts("");
    }
}

```

puts ( " " ) agrega una linea vacia, esta util para este tipo de outputs

0

6

6

0

Nao e possivel entregar a carta

<-----Aquí entra el puts cuando termina un test case

10

Nao e possivel entregar a carta

0

1076

<https://www.beecrowd.com.br/judge/es/problems/view/1076> Dfs

```

#include<bits/stdc++.h>
using namespace std;
#define INF -1
typedef pair<int,int> ii;
typedef vector<int> vi;
typedef vector<ii> vii;
vector<vi> ady;
vi dist;
int mov;
int dfs(int a, int b) {
    int cont = 0;
    dist[a] = 1;
    for (int i=0; i<b; i++) {
        if (ady[a][i] && !dist[i]) {
            cont += dfs(i, b) + 1;
        }
    }
    return cont;
}
int main() {
    int t, n, a, b, origen, destino;
    cin >> t;
    while (t--) {
        cin >> n;
        cin >> a >> b;
        ady.assign(a, vi(a, 0));
        dist.assign(a, 0);
        while (b--) {
            cin >> origen >> destino;
            ady[origen][destino] = 1;
            ady[destino][origen] = 1;
        }
        mov = dfs(n, a) * 2;
        cout << mov << "\n";
    }
    return 0;
}

```

3171

<https://www.beecrowd.com.br/judge/es/problems/view/3171> Dfs o Dfs o Disjunto

```

#include<bits/stdc++.h>
using namespace std;
vector<int> padre;
int find(int a){
    if(padre[a]==a){
        return a;
    }
    return padre[a]=find(padre[a]);
}
void unionSets(int a, int b){
    int raizA = find(a);
    int raizB = find(b);
    padre[raizA]=raizB;
}
int main(){
    int n,e;
    cin >> n >> e;
    padre.resize(n+1);
    for(int i=1; i<=n; i++){
        padre[i] = i;
    }
    while(e--){
        int a,b;
        cin >> a >> b;
        unionSets(a,b);
    }
    bool completo = true;
    for(int i=2; i<=n; i++){
        if(find(i)!=find(1)){
            completo = false;
            break;
        }
    }
    if (completo) {
        cout << "COMPLETO\n";
    } else {
        cout << "INCOMPLETO\n";
    }

    return 0;
}

```

```

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

typedef vector<int> vi;
vector<vi> ady;
vi dist;
int origen, destino, n, e;

void bfs() {
    dist[origen] = 0;
    queue<int> q;
    q.push(origen);
    while (!q.empty()) {
        int a = q.front();
        q.pop();
        for (int i = 0; i < (int)ady[a].size(); i++) {
            int b = ady[a][i];
            if (dist[b] == -1) {
                dist[b] = dist[a] + 1;
                q.push(b);
            }
        }
    }
}

int main() {
    cin >> n >> e;
    ady.assign(n + 1, vi());
    dist.assign(n + 1, -1);
    origen = 1;
    destino = n;
    while (e--) {
        int a, b;
        cin >> a >> b;
        ady[a].push_back(b);
        ady[b].push_back(a);
    }
    bfs();
    if (dist[destino] != -1) {
        cout << "COMPLETO\n";
    } else {
        cout << "INCOMPLETO\n";
    }
    return 0;
}

```

1128

<https://www.beecrowd.com.br/judge/es/problems/view/1128> Dfs o Bfs

```

#include<bits/stdc++.h>
using namespace std;
#define INF -1
typedef pair<int,int> ii;
typedef vector<int> vi;
typedef vector<ii> vii;
vector<vi> ady;
vi dist;
int origen,destino,n,e,cont;
int bfs(){
    dist[origen]=0;
    queue<int> q;
    q.push(origen);
    while(!q.empty()){
        int a=q.front();
        q.pop(), cont++;
        for(int i=0; i<ady[a].size();i++){
            int v = ady[a][i];
            if (dist[v] == INF) {
                dist[v] = dist[a]+1;
                q.push(v);
            }
        }
    }
    return cont;
}
int main(){
    while(cin>>n>>e && n && e){
        ady.assign(n,vi(0,0));
        dist.assign(n,INF);
        while(e--){
            int a,b,w;
            cin>>a>>b>>w;
            a--;b--;
            ady[a].push_back(b);
            if(w==2)ady[b].push_back(a);
        }
        bool ans = true;
        int ss=0;
        for(int i=0; i<n; i++){
            cont=0;
            origen = i;
            fill(dist.begin(),dist.end(),INF);
            ss = bfs();
            if(ss < n){
                ans = false;
            }
        }
    }
}

```

```

int main(){
    while(cin>>n>>e && n && e){
        ady.assign(n,vi(0,0));
        dist.assign(n,INF);
        while(e--){
            int a,b,w;
            cin>>a>>b>>w;
            a--;b--;
            ady[a].push_back(b);
            if(w==2)ady[b].push_back(a);
        }
        bool ans = true;
        int ss=0;
        for(int i=0; i<n; i++){
            cont=0;
            origen = i;
            fill(dist.begin(),dist.end(),INF);
            ss = bfs();
            if(ss < n){
                ans = false;
                break;
            }
        }
        cout<<ans<<"\n";
    }
    return 0;
}

```

1082

<https://www.beecrowd.com.br/judge/es/problems/view/1082> Dfs o Bfs

```

#include <bits/stdc++.h>
using namespace std;
typedef vector<int> vi;
typedef vector<vi> vii;
void dfs(int n, const vii& ady, vi& visitado, vi& componente) {
    visitado[n] = true;
    componente.push_back(n);
    for (int vecino : ady[n]) {
        if (!visitado[vecino]) {
            dfs(vecino, ady, visitado, componente);
        }
    }
}
int find(const vii& ady) {
    int n = ady.size();
    vi visitado(n, false);
    int conectado=0;
    for (int i=0; i<n; i++) {
        if (!visitado[i]) {
            vi componente;
            dfs(i, ady, visitado, componente);
            conectado++;
        }
    }
    return conectado;
}
int main() {
    int t;
    cin >> t;
    int c=1;
    do {
        int n, e;
        cin >> n >> e;
        vii ady(n);
        for (int i=0; i<e; i++) {
            char a, b;
            cin >> a >> b;
            ady[a - 'a'].push_back(b - 'a');
            ady[b - 'a'].push_back(a - 'a');
        }
        cout << "Case #" << c << ":\n";
        vi visitado(n, false);
        for (int i=0; i<n; i++) {
            if (!visitado[i]) {
                vi componente;
                dfs(i, ady, visitado, componente);
            }
        }
    } while (t--);
}

```

```

int main() {
    int t;
    cin >> t;
    int c=1;
    do {
        int n, e;
        cin >> n >> e;
        vii ady(n);
        for (int i=0; i<e; i++) {
            char a, b;
            cin >> a >> b;
            ady[a - 'a'].push_back(b - 'a');
            ady[b - 'a'].push_back(a - 'a');
        }
        cout << "Case #" << c << ":\n";
        vi visitado(n, false);
        for (int i=0; i<n; i++) {
            if (!visitado[i]) {
                vi componente;
                dfs(i, ady, visitado, componente);
                sort(componente.begin(), componente.end());
                for (int e : componente) {
                    cout << (char)('a'+e) << ",";
                }
                cout << "\n";
            }
        }
        int conectado = find(ady);
        cout << conectado << " connected components\n\n";
        c++;
    } while (--t > 0);
    return 0;
}

```

también se puede resolver con kruskal:



```

#include <bits/stdc++.h>
using namespace std;
typedef vector<int> vi;
typedef vector<vi> vii;
struct Edge {
    int origen, destino;
};
struct DS {
    vi padre, rango;
    DS(int n) {
        padre.resize(n);
        rango.resize(n, 0);
        for (int i=0; i<n; i++) {
            padre[i] = i;
        }
    }
    int find(int a) {
        if (a != padre[a]) {
            padre[a] = find(padre[a]);
        }
        return padre[a];
    }
    void unionSets(int a, int b) {
        int raizOrigen = find(a);
        int raizDestino = find(b);
        if (raizOrigen == raizDestino) {
            return;
        }
        if (rango[raizOrigen] < rango[raizDestino]) {
            padre[raizOrigen] = raizDestino;
        } else if (rango[raizOrigen] > rango[raizDestino]) {
            padre[raizDestino] = raizOrigen;
        } else {
            padre[raizDestino] = raizOrigen;
            rango[raizOrigen]++;
        }
    }
};
vii kruskal(int n, const vector<Edge>& edges) {
    DS ds(n);
    for (const Edge& edge : edges) {
        ds.unionSets(edge.origen, edge.destino);
    }
    vii componentes(n);
    for (int i=0; i<n; i++) {
        componentes[ds.find(i)].push_back(i);
    }
}

```


```

vii kruskal(int n, const vector<Edge>& edges) {
    DS ds(n);
    for (const Edge& edge : edges) {
        ds.unionSets(edge.origen, edge.destino);
    }
    vii componentes(n);
    for (int i=0; i<n; i++) {
        componentes[ds.find(i)].push_back(i);
    }
    componentes.erase(remove_if(componentes.begin(), componentes.end(), [](const vector<int>& component) { return component.
return componentes;
}

int main() {
    int t;
    cin >> t;
    int c=1;
    do {
        int n, e;
        cin >> n >> e;
        vector<Edge> edges(e);
        for (int i = 0; i < e; i++) {
            char a, b;
            cin >> a >> b;
            edges[i].origen = a - 'a';
            edges[i].destino = b - 'a';
        }
        vii componentes = kruskal(n, edges);
        cout << "Case #" << c << ":\n";
        for (const auto& component : componentes) {
            for (int e : component) {
                cout << (char)('a' + e) << ",";
            }
            cout << "\n";
        }
        cout << componentes.size() << " connected components\n\n";
        c++;
    } while (--t > 0);
    return 0;
}

```

pero produce un error en un caso de prueba que no pude resolver:



**SOURCE CODE**
EDIT & SUBMIT

VISUALIZE THE SOURCE CODE OF YOUR SUBMISSION, PLUS SOME EXTRA DETAILS.

**SUBMISSION # 34422994**

PROBLEM:	1082 - Connected Components
ANSWER:	<b>Wrong answer (0%)</b>
LANGUAGE:	C++17 (g++ 7.3.0, -std=c++17 -O2 -lm) [+0s]
RUNTIME:	0.016s
FILE SIZE:	2.11 KB
MEMORY:	-
SUBMISSION:	7/3/23, 2:46:51 PM

**DIFFERENCE**
LINE
SIDE


 1082-a.out → 1082-a.sol
 VIEWED

	@@ -1,12 +1,12 @@
1 1	Case #1:
2 2	a,
3	- d,
3	+ d,
4	- Other differences were found in this test case file!

You can check out the differences we found between **your output** and the **expected solution**! Notice that the difference we display is **limited**. This information is available for a period of **seven days**, after that it will be automatically removed. This should be used for **debug** purposes only.

1583

<https://www.beecrowd.com.br/judge/es/problems/view/1583> Floodfill

```
#include<bits/stdc++.h>
using namespace std;
int dr[] = {-1, 1, 0, 0}; //arriba y abajo (fila)
int dc[] = {0, 0, -1, 1}; //izquierda y derecha (columna)
char grid[100][100];
int R,C;
int floodfill(int r, int c, char c1, char c2){
    if(r<0 || r>=R || c<0 || c>=C)return 0;
    if(grid[r][c] != c1)return 0;
    int ans=1;
    grid[r][c]=c2;
    for(int i=0; i<4;i++){
        ans += floodfill(r+dr[i],c+dc[i],c1,c2);
    }
    return ans;
}
int main(){
    while(true){
        cin>>R>>C;
        cin.ignore();//Arregla el bug que no lee la ultima columna
        if(R==0 && C==0)break;
        for(int i=0; i<R; i++){
            for(int j=0; j<C; j++){
                cin>>grid[i][j];
            }
            getchar();
        }
        for(int i=0; i<R; i++){
            for(int j=0; j<C; j++){
                if(grid[i][j]=='T'){
                    grid[i][j]='A';
                    floodfill(i,j,'A','T');
                }
            }
        }
        for(int i=0; i<R; i++){
            for(int j=0; j<C; j++){
                cout<<grid[i][j];
            }
            cout<<"\n";
        }
        cout<<"\n";
    }
    return 0;
}
```

1907

<https://www.beecrowd.com.br/judge/en/problems/view/1907> Floodfill en Bfs

```

#include <bits/stdc++.h>
using namespace std;
typedef tuple<int, int, int> tii;
int R, C;
char grid[1026][1026];
int floodfill(int r, int c, char c1, char c2) {
    if (r < 0 || r >= R || c < 0 || c >= C) return 0;
    if (grid[r][c] != c1) return 0;
    int ans = 1;
    grid[r][c] = c2;
    queue<tii> fila;
    fila.push(make_tuple(r, c, 0));
    while (!fila.empty()) {
        int x, y, cor;
        tie(x, y, cor) = fila.front();
        fila.pop();
        if (grid[x - 1][y] == c1) {
            fila.push(make_tuple(x - 1, y, cor));
            grid[x - 1][y] = c2;
        }
        if (grid[x + 1][y] == c1) {
            fila.push(make_tuple(x + 1, y, cor));
            grid[x + 1][y] = c2;
        }
        if (grid[x][y - 1] == c1) {
            fila.push(make_tuple(x, y - 1, cor));
            grid[x][y - 1] = c2;
        }
        if (grid[x][y + 1] == c1) {
            fila.push(make_tuple(x, y + 1, cor));
            grid[x][y + 1] = c2;
        }
    }
    return ans;
}

```

```

int main() {
    cin >> R >> C;
    cin.ignore();
    for (int i = 0; i < R; i++) {
        for (int j = 0; j < C; j++) {
            cin >> grid[i][j];
        }
    }
    int ans = 0;
    for (int i = 0; i < R; i++) {
        for (int j = 0; j < C; j++) {
            if (grid[i][j] == '.') {
                ans++;
                floodfill(i, j, '.', 'k');
            }
        }
    }
    cout << ans << "\n";
    return 0;
}

```

**UVA ONLINE JUDGE**

## 11172 - Relational Operator

[https://onlinejudge.org/index.php?option=com\\_onlinejudge&Itemid=8&category=23&page=show\\_problem&problem=2113](https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=23&page=show_problem&problem=2113) Manipulación de Bits y Bitwise

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    int t;
    cin >> t; // Leer el número de conjuntos de entradas
    for (int i = 0; i < t; i++) {
        int a, b;
        cin >> a >> b; // Leer los dos números de cada conjunto
        // Obtener el bit más significativo de a y b
        int bitA = (a >> 31) & 1;
        int bitB = (b >> 31) & 1;
        if (bitA > bitB) {
            cout << ">" << endl;
        } else if (bitA < bitB) {
            cout << "<" << endl;
        } else {
            // Comparar los bits restantes
            if ((a & 0x7FFFFFFF) > (b & 0x7FFFFFFF)) {
                cout << ">" << endl;
            } else if ((a & 0x7FFFFFFF) < (b & 0x7FFFFFFF)) {
                cout << "<" << endl;
            } else {
                cout << "=" << endl;
            }
        }
    }
    return 0;
}
```

## 469 - Wetlands of Florida

[https://onlinejudge.org/index.php?option=com\\_onlinejudge&Itemid=8&category=667&page=show\\_problem&problem=410](https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=667&page=show_problem&problem=410) floodfill

```

#include<bits/stdc++.h>
using namespace std;
int dr[] = {1, 1, 0, -1, -1, -1, 0, 1};
int dc[] = {0, 1, 1, 1, 0, -1, -1, -1};
char grid[100][100];
int R, C, n, m;
int floodfill(int r, int c, char c1, char c2) {
    if (!(0 <= r && r < n && 0 <= c && c < m)) return 0;
    if (grid[r][c] != c1) return 0;
    grid[r][c] = c2;
    int area = 1;
    for (int i = 0; i < 8; i++) {
        area += floodfill(r + dr[i], c + dc[i], c1, c2);
    }
    return area;
}
int main() {
    int t;
    cin >> t;
    cin.ignore(1);
    string s;
    getline(cin, s);
    while (t--) {
        n = 0;
        m = 0;
        while (getline(cin, s), (s[0] == 'L' || s[0] == 'W')) {
            for (int i = 0; i < s.size(); i++) {
                grid[n][i] = s[i];
            }
            n++;
            m = s.size();
        }
        stringstream ss(s);
        ss >> R >> C;
        cout << floodfill(R - 1, C - 1, 'W', '.') << '\n';
        floodfill(R - 1, C - 1, '.', 'W');
        while (getline(cin, s), s.compare("") != 0) {
            ss.clear();
            ss.str(s);
            ss >> R >> C;
            cout << floodfill(R - 1, C - 1, 'W', '.') << '\n';
            floodfill(R - 1, C - 1, '.', 'W');
        }
        if (t) cout << '\n';
    }
    return 0;
}

```

## 572 - Oil Deposits

[https://onlinejudge.org/index.php?option=com\\_onlinejudge&Itemid=8&category=667&page=show\\_problem&problem=513](https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=667&page=show_problem&problem=513) **floodfill**

```

#include<bits/stdc++.h>
using namespace std;
int dr[]={1,1,0,-1,-1,-1,0,1};
int dc[]={0,1,1,1,0,-1,-1,-1};
char grid[100][100];
int R,C;
int floodfill(int r, int c, char c1, char c2){
    if(r<0 || r>=R || c<0 || c>=C)return 0;
    if(grid[r][c]!=c1)return 0;
    int ans=1;
    grid[r][c]=c2;
    for(int i=0; i<8; i++){
        ans += floodfill(r+dr[i],c+dc[i],c1,c2);
    }
    return ans;
}
int main(){
    while(true){
        cin>>R>>C;
        cin.ignore();
        if(R==0 && C==0)break;
        for(int i=0; i<R; i++){
            for(int j=0; j<C; j++){
                cin>>grid[i][j];
            }
        }
        int ans=0;
        for(int i=0; i<R; i++){
            for(int j=0; j<C; j++){
                if (grid[i][j] == '@') {
                    if (floodfill(i, j, '@', '.') >= 1) {
                        ans++;
                    }
                }
            }
        }
        cout<<ans<<"\n";
    }
    return 0;
}

```

## 10336 - Rank the Languages

[https://onlinejudge.org/index.php?option=com\\_onlinejudge&Itemid=8&category=667&page=show\\_problem&problem=1277](https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=667&page=show_problem&problem=1277) **floodfill**

```

#include <bits/stdc++.h>
using namespace std;
int dr[] = {1, 0, -1, 0};
int dc[] = {0, 1, 0, -1};
map<char, int> mci;
char grid[100][100];
int R, C, t, c = 1;

void floodfill(int r, int c, char c1, char c2) {
    if (r < 1 || r > R || c < 1 || c > C)
        return;
    if (grid[r][c] != c1)
        return;
    grid[r][c] = c2;
    for (int p = 0; p < 4; ++p)
        floodfill(r+dr[p], c+dc[p], c1, c2);
}

struct Compare {
    bool operator()(const char& a, const char& b) const {
        if (mci[a] != mci[b])
            return mci[a] > mci[b];
        return a < b;
    }
};

int main() {
    cin >> t;
    while (t--) {
        mci.clear();
        cin >> R >> C;
        for (int i = 1; i <= R; ++i)
            for (int j = 1; j <= C; ++j)
                cin >> grid[i][j];
        for (int letra = 97; letra <= 122; letra++){
            int ans = 0;
            for (int i = 1; i <= R; ++i){
                for (int j = 1; j <= C; ++j){
                    if (grid[i][j] == (char)letra){
                        ++ans;
                        floodfill(i, j, (char)letra, '.');
                    }
                }
            }
            if (ans > 0)
                mci[(char)letra] = ans;
        }
    }
}

```



```

int main() {
    cin >> t;
    while (t--) {
        mci.clear();
        cin >> R >> C;
        for (int i = 1; i <= R; ++i)
            for (int j = 1; j <= C; ++j)
                cin >> grid[i][j];
        for (int letra = 97; letra <= 122; letra++){
            int ans = 0;
            for (int i = 1; i <= R; ++i){
                for (int j = 1; j <= C; ++j){
                    if (grid[i][j] == (char)letra){
                        ++ans;
                        floodfill(i, j, (char)letra, '.');
                    }
                }
            }
            if(ans > 0)
                mci[(char)letra] = ans;
        }
        cout << "World #" << c++ << "\n";
        map<char, int, Compare> res(mci.begin(), mci.end());
        for (const auto& e : res) {
            cout << e.first << ": " << e.second << "\n";
        }
    }
    return 0;
}

```

## 11244 - Counting Stars

[https://onlinejudge.org/index.php?option=com\\_onlinejudge&Itemid=8&category=667&page=show\\_problem&problem=2201](https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=667&page=show_problem&problem=2201) **floodfill**

```

#include<bits/stdc++.h>
using namespace std;
int dr[]={1,1,0,-1,-1,-1,0,1};
int dc[]={0,1,1,1,0,-1,-1,-1};
char grid[100][100];
int R,C;
int floodfill(int r, int c, char c1, char c2){
    if(r<0 || r>=R || c<0 || c>=C)return 0;
    if(grid[r][c]!=c1)return 0;
    int ans=1;
    grid[r][c]=c2;
    for(int i=0; i<8; i++){
        ans += floodfill(r+dr[i],c+dc[i],c1,c2);
    }
    return ans;
}
int main(){
    while(true){
        cin>>R>>C;
        cin.ignore();
        if(R==0 && C==0)break;
        for(int i=0; i<R; i++){
            for(int j=0; j<C; j++){
                cin>>grid[i][j];
            }
        }
        int ans=0;
        for(int i=0; i<R; i++){
            for(int j=0; j<C; j++){
                if (grid[i][j] == '*' && floodfill(i, j, '*', '.') == 1) {
                    grid[i][j] = '.';
                    ans++;
                }
            }
        }
        /*cout<<"DESPUES: \n";
        for(int i=0; i<R; i++){
            for(int j=0; j<C; j++){
                cout<<grid[i][j];
            }
            cout<<"\n";
        }*/
        cout<<ans<<"\n";
    }
    return 0;
}

```

## 908 - Reconnecting computer Sites

[https://onlinejudge.org/index.php?option=com\\_onlinejudge&Itemid=8&category=673&page=show\\_problem&problem=849](https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=673&page=show_problem&problem=849) **Kruskal disjointo**

```

#include<bits/stdc++.h>
using namespace std;
#define MAX 1000001
struct Edge{
    int v1, v2;
    int weight;
}edge[MAX];
int p[MAX];
inline int find(int a);
int kruskal(int N, int E);
int main(){
    int N;
    bool first = true;
    while (cin >> N){
        int a, b, w;
        int old_cost = 0, new_cost;
        for (int i=1; i<N; i++){
            cin >> a >> b >> w;
            old_cost += w;
        }
        int K, M;
        cin >> K;
        for (int i=0; i<K; i++)
            cin >> edge[i].v1 >> edge[i].v2 >> edge[i].weight;
        cin >> M;
        for (int i=0; i<M; i++)
            cin >> edge[i + K].v1 >> edge[i + K].v2 >> edge[i + K].weight;
        new_cost = kruskal(N, K + M);
        if (first)
            first = false;
        else
            putchar('\n');
        cout << old_cost << '\n' << new_cost << '\n';
    }
    return 0;
}
int kruskal(int N, int E){
    for (int i=0; i<= N; i++)
        p[i] = i;
    sort(edge, edge + E, [](const Edge& a, const Edge& b)->bool{return a.weight < b.weight;});
    int e, cost = 0;
    for (int i=0, e=0; i<E&&e<N-1; e++, i++){
        while (find(edge[i].v1) == find(edge[i].v2))i++;
        p[find(edge[i].v1)] = find(edge[i].v2);
        cost += edge[i].weight;
    }
}

```

```

int main(){
    int N;
    bool first = true;
    while (cin >> N){
        int a, b, w;
        int old_cost = 0, new_cost;
        for (int i=1; i<N; i++){
            cin >> a >> b >> w;
            old_cost += w;
        }
        int K, M;
        cin >> K;
        for (int i=0; i<K; i++)
            cin >> edge[i].v1 >> edge[i].v2 >> edge[i].weight;
        cin >> M;
        for (int i=0; i<M; i++)
            cin >> edge[i + K].v1 >> edge[i + K].v2 >> edge[i + K].weight;
        new_cost = kruskal(N, K + M);
        if (first)
            first = false;
        else
            putchar('\n');
        cout << old_cost << '\n' << new_cost << '\n';
    }
    return 0;
}

int kruskal(int N, int E){
    for (int i=0; i<= N; i++)
        p[i] = i;
    sort(edge, edge + E, [](const Edge& a, const Edge& b)->bool{return a.weight < b.weight;});
    int e, cost = 0;
    for (int i=0, e=0; i<E&&e<N-1; e++, i++){
        while (find(edge[i].v1) == find(edge[i].v2))i++;
        p[find(edge[i].v1)] = find(edge[i].v2);
        cost += edge[i].weight;
    }
    return cost;
}

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

```

## 1208 - Oreon

[https://onlinejudge.org/index.php?option=onlinejudge&Itemid=8&page=show\\_problem&problem=3649](https://onlinejudge.org/index.php?option=onlinejudge&Itemid=8&page=show_problem&problem=3649)

### 49 Kruskal

```

#include<bits/stdc++.h>
using namespace std;
struct Edge {
    int origen, destino;
    int pesoE;
};
const int MAX = 26;
int padre[MAX];
int find(int a) {
    return a == padre[a] ? a : (padre[a] = find(padre[a]));
}
int kruskal(int N, int E, Edge edge[]) {
    for (int i = 0; i < N; i++)
        padre[i] = i;
    sort(edge, edge + E, [](const Edge& a, const Edge& b) -> bool {
        return a.pesoE < b.pesoE;
    });
    int e, cost = 0;
    for (int i = 0, e = 0; i < E && e < N - 1; e++, i++) {
        while (find(edge[i].origen) == find(edge[i].destino))
            i++;
        padre[find(edge[i].origen)] = find(edge[i].destino);
        cost += edge[i].pesoE;
        char origen = 'A' + edge[i].origen;
        char destino = 'A' + edge[i].destino;
        printf("%c-%c %d\n", origen, destino, edge[i].pesoE);
    }
    return cost;
}
int main() {
    int t, ciudades, c = 0, tmp;
    char coma;
    scanf("%d", &t);
    do{
        scanf("%d", &ciudades);
        Edge edges[MAX];
        int e = 0;
        for(int i=0; i<ciudades; i++) {
            for(int j=0; j<ciudades; j++) {
                scanf("%d%c", &tmp, &coma);
                if(i<j && tmp) {
                    edges[e].origen = i;
                    edges[e].destino = j;
                    edges[e].pesoE = tmp;
                    e++;
                }
            }
        }
    } while(t--);
}

```

```

int main() {
    int t, ciudades, c = 0, tmp;
    char coma;
    scanf("%d", &t);
    do{
        scanf("%d", &ciudades);
        Edge edges[MAX];
        int e = 0;
        for(int i=0; i<ciudades; i++) {
            for(int j=0; j<ciudades; j++) {
                scanf("%d%c", &tmp, &coma);
                if(i<j && tmp) {
                    edges[e].origen = i;
                    edges[e].destino = j;
                    edges[e].pesoE = tmp;
                    e++;
                }
            }
        }
        printf("Case %d:\n", ++c);
        kruskal(ciudades, e, edges);
        t--;
    }while(t>0);
    return 0;
}

```

## 10305 - Ordering tasks

[https://onlinejudge.org/index.php?option=com\\_onlinejudge&Itemid=8&category=668&page=show\\_problem&problem=1246](https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=668&page=show_problem&problem=1246) **topological sort** Nota: El resultado real del caso de prueba es: 4 1 5 2 3

```

#include <bits/stdc++.h>
using namespace std;
typedef vector<int> vi;
typedef pair<int, int> ii;
vector<vector<ii>> ady;
vi visited, topo;
int n, e;
void dfs(int u){
    visited[u] = 1;
    for (ii temp : ady[u]){
        int v = temp.first;
        if (!visited[v])
            dfs(v);
    }
    topo.push_back(u);
}
int main(){
    while ((cin >> n >> e) && n){
        ady.assign(n + 1, vector<ii>());
        visited.assign(n + 1, 0);
        topo.clear();
        for (int i=0; i<e; ++i){
            int a, b;
            cin >> a >> b;
            ady[a].push_back(make_pair(b, 1));
        }
        for (int i = 1; i <= n; ++i){
            if (!visited[i])
                dfs(i);
        }
        reverse(topo.begin(), topo.end());
        for (int i=0; i<n; ++i){
            if (i > 0)
                cout << " ";
            cout << topo[i];
        }
        cout << "\n";
    }
    return 0;
}

```

## 200 - Rare Order

[https://onlinejudge.org/index.php?option=com\\_onlinejudge&Itemid=8&category=668&page=show\\_problem&problem=136](https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=668&page=show_problem&problem=136) **topological sort**

```

#include <bits/stdc++.h>
using namespace std;
typedef vector<int> vi;
typedef vector<vi> vii;
vii ady;
vi inDegree;
vi dfs_num;
vi toposort;
void topologicalDFS(int u) {
    dfs_num[u] = 1;
    for (int v : ady[u]) {
        if (dfs_num[v] == 0) {
            topologicalDFS(v);
        }
    }
    toposort.push_back(u);
}
int main() {
    vector<string> palabras;
    string letra;
    while (getline(cin, letra)) {
        if (letra == "#") {
            break;
        }
        palabras.push_back(letra);
    }
    map<char, set<char>> origen;
    set<char> caracteres;
    for (const string& letra : palabras) {
        for (char c : letra) {
            caracteres.insert(c);
        }
    }
    for (int i=0; i<palabras.size()-1; i++) {
        const string& actual = palabras[i];
        const string& siguiente = palabras[i + 1];
        int j=0;
        while (j<actual.size()&&j<siguiente.size()&&actual[j]==siguiente[j]) {
            j++;
        }
        if (j<actual.size()&&j<siguiente.size()){
            origen[actual[j]].insert(siguiente[j]);
        }
    }
    int nodos = caracteres.size();
    map<char, int> mapa;
    .
    .
    .

```



```

for (int i=0; i<palabras.size()-1; i++) {
    const string& actual = palabras[i];
    const string& siguiente = palabras[i + 1];
    int j=0;
    while (j<actual.size()&&j<siguiente.size()&&actual[j]==siguiente[j]) {
        j++;
    }
    if (j<actual.size()&&j<siguiente.size()){
        origen[actual[j]].insert(siguiente[j]);
    }
}
int nodos = caracteres.size();
map<char, int> mapa;
int raiz = 0;
for (char c : caracteres) {
    mapa[c] = raiz++;
}
ady.resize(nodos);
inDegree.assign(nodos, 0);
for (const auto& e : origen) {
    char nodo = e.first;
    int a = mapa[nodo];
    for (char siguiente : e.second) {
        int b = mapa[siguiente];
        ady[a].push_back(b);
        inDegree[b]++;
    }
}
dfs_num.assign(nodos, 0);
for (int i=0; i<nodos; i++) {
    if (dfs_num[i] == 0) {
        topologicalDFS(i);
    }
}
reverse(toposort.begin(), toposort.end());
for (int i : toposort) {
    for (const auto& e : mapa) {
        if (e.second == i) {
            cout << e.first;
        }
    }
}
cout << "\n";
return 0;
}

```

## 929 - Number Maze

[https://onlinejudge.org/index.php?option=com\\_onlinejudge&Itemid=8&category=678&page=show\\_p  
roblem&problem=870](https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=678&page=show_problem&problem=870) **Dijkstra**

```

#include <bits/stdc++.h>
using namespace std;
typedef vector<int> vi;
typedef pair<int, pair<int, int>> pii;

int dijkstra(vector<vi>& maze, int N, int M) {
    vector<vi> dist(N, vi(M, INT_MAX));
    priority_queue<pii, vector<pii>, greater<pii>> pq;
    dist[0][0] = maze[0][0];
    pq.push({maze[0][0], {0, 0}});
    while (!pq.empty()) {
        int cost = pq.top().first;
        int a = pq.top().second.first;
        int b = pq.top().second.second;
        pq.pop();
        if (cost > dist[a][b]) continue;
        int dr[] = {0, 0, 1, -1};
        int dc[] = {1, -1, 0, 0};
        for (int i=0; i<4; i++) {
            int na = a + dr[i];
            int nb = b + dc[i];
            if (na >= 0 && na < N && nb >= 0 && nb < M) {
                int resultado = cost + maze[na][nb];
                if (resultado < dist[na][nb]) {
                    dist[na][nb] = resultado;
                    pq.push({resultado, {na, nb}});
                }
            }
        }
    }
    return dist[N - 1][M - 1];
}

int main() {
    int t;
    cin >> t;
    while (t--) {
        int N, M;
        cin >> N >> M;
        vector<vi> maze(N, vi(M));
        for (int i=0; i<N; i++)
            for (int j=0; j<M; j++)
                cin >> maze[i][j];

        int ans = dijkstra(maze, N, M);
        cout << ans << "\n";
    }
}

```

Como soporte de todos los códigos y su origen, todas la capturas de códigos son directamente sacadas de la bitácora que llevamos como equipo CodeX en la plataforma de Discord donde subimos todos los códigos tanto de las RPCS y de los jueces online, aqui pondre algunas capturas para probar que el contenido es netamente nuestro :

Nicolás

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

int main(){

    ios::sync_with_stdio(0);cin.tie(0);
    int n=0,c=0,r=0;
    cin>>n;
    char s[n+1];
    for(int i=0;i<n;++i){
        cin>>s[i];
        if(s[i]=='1')c=3;
        if(c>0)r++;
        c--;
    }
    cout<<r<<"\n";

    return 0;
}
```

^ Contraer ↗

C.cpp 1 KB ⬇ &lt;&gt;

```
#include <iostream>
#include <string>

using namespace std;

bool isDickensian(const string& word) {
```

v Expandir ↗

D.cpp 1 KB ⬇ &lt;&gt;

15:50 Nicolás

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

int main(){

    ios::sync_with_stdio(0);cin.tie(0);
```



Enviar mensaje a # -Codex

# -Codex

Nicolás .

Nicolás

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

int main() {
    int c,r;
    while(cin>>r>>c&&r&&c){
```

▼ Expandir ↗sideWaysSorting.cpp 1 KB ⬇ <>

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

int main(){
    int X;
    vector<int> values;
```

▼ Expandir ↗uva\_10107.cpp 1 KB ⬇ <>

```
#include <bits/stdc++.h>
using namespace std;
vector<string> v;
set<string> sss;

int main() {
```

▼ Expandir ↗beecrowd\_1215.cpp 1 KB ⬇ <>

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

struct Cup {
    string color;
    double radius;
```

▼ Expandir ↗Kattis\_StackingCups.cpp 2 KB ⬇ <>

+ Enviar mensaje a # -Codex

17:05 Nicolás

```
#include<bits/stdc++.h>
#define maxn 100010
using namespace std;

char s[maxn];
int b[maxn];
```

▼ Expandir ↗ J.cpp 2 KB ⬇ <>

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

using namespace std;

int main() {
```

▼ Expandir ↗ A.cpp 1 KB ⬇ <>

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

const int maxn = 760;
const int lim = 20000000;
const int max_p = 1400000;
```

▼ Expandir ↗ B.cpp 3 KB ⬇ <>

```
#include<bits/stdc++.h>
#define maxn 200010
using namespace std;

int lowbit(int x){
    return x & (-x);
```

▼ Expandir ↗ C.cpp 2 KB ⬇ <>



Enviar mensaje a # 📄 -Codex

12 de junio de 2023

18:49 **juan**

```
int gcd(int a, int b) {  
    return !b ? a : gcd(b, a % b);  
}
```

13 de junio de 2023

10:41 **juan**

```
#include <bits/stdc++.h>  
using namespace std;  
  
typedef vector<int> vi;  
vector<vi> ady;  
vi dist;
```

▼ Expandir ↗ b-3171\_Con\_grafos\_bfs.cpp 1 KB ⬇ <>

**juan**

```
#include <bits/stdc++.h>  
using namespace std;  
  
#define INF INT_MAX  
typedef vector<int> vi;  
typedef pair<int,int> ii;
```

▼ Expandir ↗ b-1148.cpp 2 KB ⬇ <>

**juan**

```
#include<bits/stdc++.h>  
using namespace std;  
#define INF -1  
typedef pair<int,int> ii;  
typedef vector<int> vi;  
typedef vector<ii> vii;
```



Enviar mensaje a # -Codex

11:45 **juan** esto pa la plantilla

```
juan #define INF -1
typedef pair<int,int> ii;
typedef vector<int> vi;
typedef vector<ii> vii;
vector<vi> ady;
vi dist;
int origen, destino, n, e;
```

12:45 **juan**

```
#include <bits/stdc++.h>

using namespace std;

int dr[]={1,1,0,-1,-1,-1,0,1}; //Controla la coordenada del inundamiento en las filas
int dc[]={0,1,1,1,0,-1,-1,-1}; //Controla la coordenada del inundamiento en la columnas
```

Expandir ↗

Floodfill.cpp 2 KB ⬇ <>

16:35 **juan**

```
#include<bits/stdc++.h>
using namespace std;
int dr[] = {-1, 1, 0, 0}; //arriba y abajo (fila)
int dc[] = {0, 0, -1, 1}; //izquierda y derecha (columna)
char grid[100][100];
int R,C;
```

Expandir ↗

b-1583.cpp 2 KB ⬇ <>

17:42 **juan**

```
#include<bits/stdc++.h>
using namespace std;
int dr[]={1,1,0,-1,-1,-1,0,1};
int dc[]={0,1,1,1,0,-1,-1,-1};
char grid[100][100];
```

+ Enviar mensaje a # -Codex

➔

▼ Expandir ↗ rpc-a.cpp 1 KB ⬇ <>

13:40 Camilo

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

int main() {
    int d1, m1, y1, n1;
```

▼ Expandir ↗ B.cpp 1 KB ⬇ <>

14:18 Camilo

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

bool isValid(string& binaryString, int k) {
    int consecutiveZeros = 0;
    int consecutiveOnes = 0;
```

▼ Expandir ↗ D.cpp 1 KB ⬇ <>

14:30 Nicolás

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

int main() {
    int N;
    cin >> N;
```

▼ Expandir ↗ C.cpp 2 KB ⬇ <>

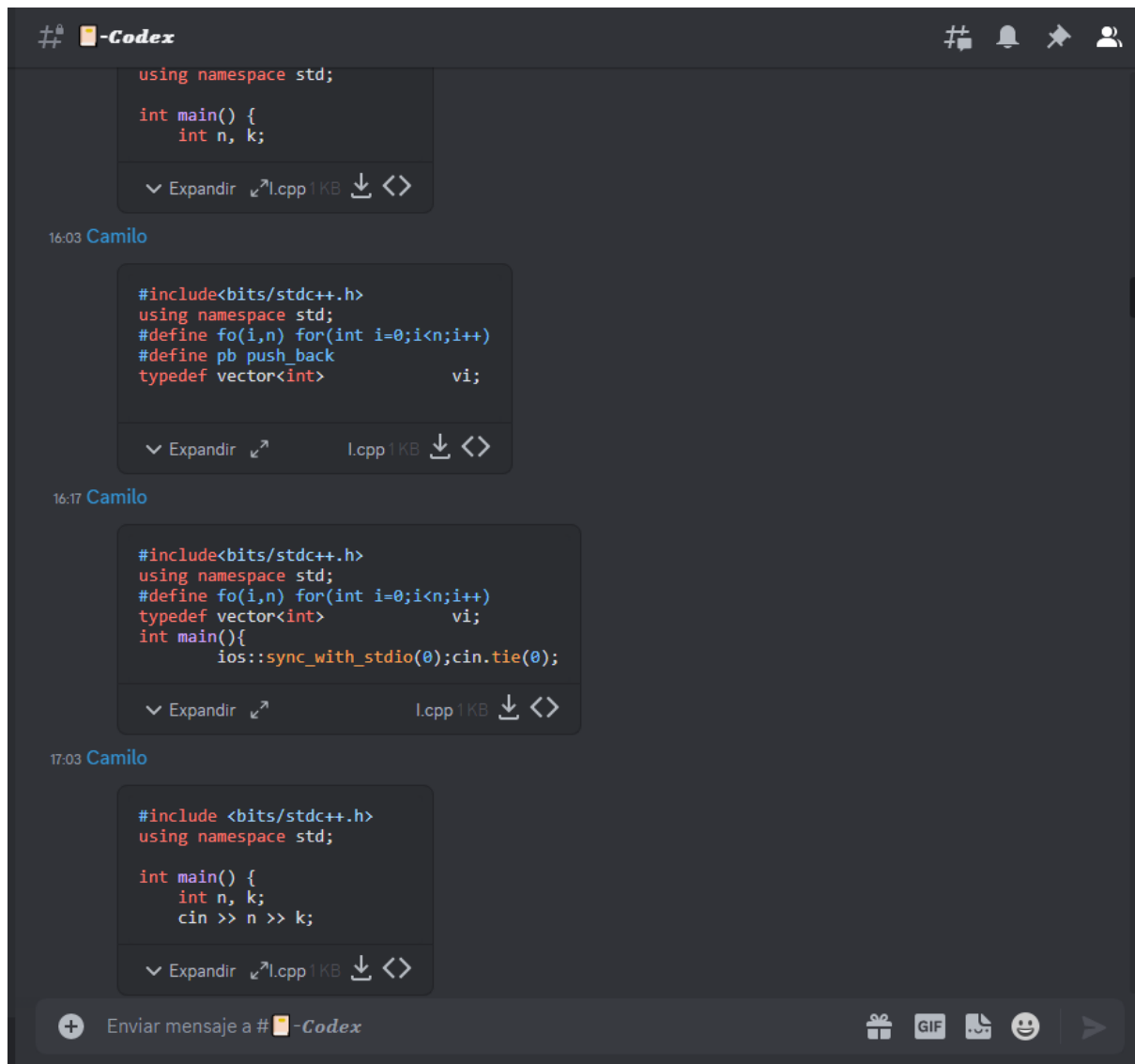
14:58 Nicolás .

Nicolás .

15:28 Camilo

+ Enviar mensaje a # -Codex





## Búsqueda binaria

La búsqueda binaria es una forma eficiente de encontrar un elemento en un arreglo, para su implementación es importante tener en cuenta los siguientes puntos:

1. La lista debe estar ordenada: La búsqueda binaria solo funciona en listas ordenadas. Si la lista no está ordenada, debes ordenarla antes de aplicar el algoritmo.
2. Eficiencia: La búsqueda binaria es especialmente útil cuando tienes una gran cantidad de elementos y deseas realizar búsquedas rápidas. La complejidad temporal de la búsqueda binaria es  $O(\log n)$ . En comparación, la búsqueda lineal tiene una complejidad  $O(n)$ , lo que significa que la búsqueda binaria es considerablemente más rápida para listas grandes.

3. Uso frecuente de búsqueda: Si sabes que necesitarás realizar múltiples búsquedas en la misma lista, puede ser beneficioso ordenar la lista una vez y luego realizar búsquedas binarias en lugar de realizar búsquedas lineales repetidas.

Algunas funciones utilizadas en búsqueda binaria en c++ son las siguientes:

- `Lower_bound`: La función `lower_bound` devuelve un iterador al primer elemento en una lista ordenada que es igual o mayor que un valor dado.

-Sintaxis: `lower_bound(first, last, value)`

-Parámetros:

`first`: Un iterador que señala al primer elemento de la lista.

`last`: Un iterador que señala al último elemento de la lista.

`value`: El valor que se desea buscar.

-Retorno: Un iterador al primer elemento mayor o igual que `value`. Si no se encuentra ningún elemento mayor o igual, se devuelve el iterador `last`.

- `Upper_bound`: La función `lower_bound` devuelve un iterador al primer elemento en una lista ordenada que es igual o mayor que un valor dado.

-Sintaxis: `lower_bound(first, last, value)`

-Parámetros:

`first`: Un iterador que señala al primer elemento de la lista.

`last`: Un iterador que señala al último elemento de la lista.

`value`: El valor que se desea buscar.

-Retorno: Un iterador al primer elemento mayor o igual que `value`. Si no se encuentra ningún elemento mayor o igual, se devuelve el iterador `last`.

- `Binary_search`: La función `lower_bound` devuelve un iterador al primer elemento en una lista ordenada que es igual o mayor que un valor dado.

-Sintaxis: `lower_bound(first, last, value)`

-Parámetros:

`first`: Un iterador que señala al primer elemento de la lista.

last: Un iterador que señala al último elemento de la lista.

value: El valor que se desea buscar.

-Retorno: Un iterador al primer elemento mayor o igual que value. Si no se encuentra ningún elemento mayor o igual, se devuelve el iterador last.

Todas las funciones anteriores tienen una complejidad de  $O(\log n)$ .

Algunos de los ejercicios realizados fueron extraídos del libro competitive programming 3 the new lower bound of programming contests de Steven Halim y Felix Halim. Todos los ejercicios mencionados en este apartado recibieron AC en su respectivo juez:

- [https://onlinejudge.org/index.php?option=com\\_onlinejudge&Itemid=8&page=show\\_problem&problem=898](https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=898)
- [https://onlinejudge.org/index.php?option=com\\_onlinejudge&Itemid=8&page=show\\_problem&problem=1998](https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=1998)
- [https://onlinejudge.org/index.php?option=com\\_onlinejudge&Itemid=8&page=show\\_problem&problem=1415](https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=1415)
- [https://onlinejudge.org/index.php?option=com\\_onlinejudge&Itemid=8&page=show\\_problem&problem=620](https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=620)
- [https://onlinejudge.org/index.php?option=com\\_onlinejudge&Itemid=8&page=show\\_problem&problem=1552](https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=1552)
- [https://onlinejudge.org/index.php?option=com\\_onlinejudge&Itemid=8&page=show\\_problem&problem=1508](https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=1508)
- Ejercicio G de la RPC 07 2023.

15:28 Camilo

```
#include<bits/stdc++.h>
using namespace std;
#define fo(i,n) for(int i=0;i<n;i++)
#define ll long long
#define pb push_back
typedef vector<int> vi;
```

▼ Expandir ↗

G.cpp 2 KB ⬇ <>

26 de junio de 2023

```
#include<bits/stdc++.h>
using namespace std;
#define fo(i,n) for(int i=0;i<n;i++)
#define ll long long
#define pb push_back
typedef vector<int> vi;
typedef vector<ll> vl;
ll gcd(ll a, ll b){if (b == 0)return a;return gcd(b, a % b);}
ll lcm(ll a, ll b){return(a/gcd(a,b))*b;}

ll contMult(ll lim, vl& primes){
    int k=primes.size();
    ll r=0;
    for(int i=1; i<(1<<k);++i){
        int cont = __builtin_popcount(i);
        ll mult=1;
        for(int j=0;j<k;++j){
            if(i&(1<<j)){
                mult=lcm(mult,primes[j]);
            }
        }
        if(cont%2==1)
            r+=lim/mult;
        else
            r-=lim/mult;
    }
    return r;
}

ll encontrarNth(ll n, vl& primes){
    ll menor=1;
    ll mayor=1e18;
    ll r=-1;

    while(menor<=mayor){
        ll mid=(menor+mayor)/2;
        ll cont= mid-contMult(mid,primes);
        if(cont>=n){
            r=mid;
            mayor=mid-1;
        }else{
            menor=mid+1;
        }
    }
    return r;
}
```

```



ll encontrarNth(ll n, vl& primes){
    ll menor=1;
    ll mayor=1e18;
    ll r=-1;

    while(menor<=mayor){
        ll mid=(menor+mayor)/2;
        ll cont= mid-contMult(mid,primes);
        if(cont>=n){
            r=mid;
            mayor=mid-1;
        }else{
            menor=mid+1;
        }
    }
    return r;
}

int main(){
    ios::sync_with_stdio(0);cin.tie(0);
    ll n;
    int k;
    cin>>n>>k;
    vl primes(k);
    fo(i,k) cin>>primes[i];

    ll r=encontrarNth(n,primes);
    cout<<r<<"\n";
    return 0;
}

```

G.cpp 2 KB  

De todo esto se puede reutilizar bastante código y adaptarlo a lo que se necesite:

```
int lowerBound(vector<int>& arr, int target) {
```

```
    int n = (int)arr.size();
```

```
    int left = 0, right = n - 1;
```

```
    int indx = -1;
```

```
    while(left <= right) {
```

```
        int mid = left + (right - left) / 2;
```

```
        if(arr[mid] == target) {
```

```
            indx = mid;
```

```
        }
```

```
        if(arr[mid] < target) {
```

```
            left = mid + 1;
```

```

    } else { // arr[mid] >= target
        right = mid - 1;
    }
}

return indx;
}

```

// different from C++ upper\_bound. return target's index inclusive

```

int upperBound(vector<int>& arr, int target) {
    int n = (int)arr.size();
    int left = 0, right = n - 1;
    int indx = -1;
    while(left <= right) {
        int mid = left + (right - left) / 2;
        if(arr[mid] == target) {
            indx = mid;
        }
        if(arr[mid] <= target) {
            left = mid + 1;
        } else { // arr[mid] > target
            right = mid - 1;
        }
    }
    return indx;
}

```

```

pair<int, int> equalRange(vector<int>& arr, int target) {

    int lbound = lowerBound(arr, target);

    int ubound = upperBound(arr, target);

    return pair<int, int>{lbound, ubound};

}

```

```

int binarySearch(vector<int>& arr, int target) {

    int lbound = lowerBound(arr, target);

    return lbound;

}

```

Eso sería todo con respecto a la búsqueda binaria. Además de esto también leí contenido de aritmética modular y teoría de números para la computación, este es un tema muy extenso y complejo ya que abarca desde teoría elemental de números hasta su implementación en algoritmos, criptografía, etc.

Este apartado sería un proyecto personal a futuro, para estas vacaciones me pareció más idóneo enfocarme en un solo tema y reforzar los conocimientos adquiridos hasta la fecha.

También se realizaron lecturas sobre búsqueda completa, combinatorias, números primos, factoriales, mínimo común múltiplo(lcm), máximo común divisor(gcd), procesamiento de strings y algunos temas raros como las bitmask. Me gustaría destacar que la lectura de estos temas sin su aplicación práctica como tal también han servido para orientarnos en algunos problemas de las RPCs realizadas en esta temporada vacacional, permitiendo identificar su posible implementación la cual requirió de una mayor investigación para resolver el problema en cuestión un ejemplo de esto puede ser el ejercicio G de la RPC 07/2023, para este ejercicio implementamos una bitmask para contar la cantidad de múltiplos de una serie de números primos, tanto el lcm como el gcd también fueron implementados.

Para finalizar este apartado personal de mi trabajo durante este receso, me gustaría compartir la plantilla que he estado usando la cual cuenta con algunos elementos que sirven para agilizar la codificación y no tener que escribir frecuentemente el mismo código.

**Template o plantilla**

```

#include<bits/stdc++.h>

using namespace std;

#define fo(i,n) for(i=0;i<n;i++)

#define ll long long

#define si(x) scanf("%d",&x)

#define sl(x) scanf("%lld",&x)

#define ss(s) scanf("%s",s)

#define pi(x) printf("%d\n",x)

#define pl(x) printf("%lld\n",x)

#define ps(s) printf("%s\n",s)

#define pb push_back

#define mp make_pair

#define F first

#define S second

#define sortall(x) sort(all(x))

#define PI 3.1415926535897932384626

typedef pair<int, int> pii;

typedef pair<ll, ll> pl;

typedef vector<int> vi;

typedef vector<ll> vl;

typedef vector<pii> vpii;

typedef vector<pl> vpl;

typedef vector<vi> vvi;

typedef vector<vl> vvl;

ll gcd(ll a, ll b){if (b == 0)return a;return gcd(b, a % b);}

```



```
ll lcm(ll a, ll b){return(a/gcd(a,b))*b;}
```

```
/*
```

```
ll moduloMultiplication(ll a,ll b,ll mod){ll res = 0;a %= mod;while (b){if (b & 1)res = (res + a) % mod;b >>= 1;}return res;}
```

```
ll powermod(ll x, ll y, ll p){ll res = 1;x = x % p;if (x == 0) return 0;while (y > 0){if (y & 1)res = (res*x) % p;y = y>>1;x = (x*x) % p;}return res;}*/
```

```
int main(){
```

```
ios::sync_with_stdio(0);cin.tie(0);
```

```
return 0;
```

```
}
```

Esta plantilla contiene en general algunos elementos que facilitan la implementación del código y algunos algoritmos reutilizables como el gcd y el lcm, en lo personal me parece muy útil y práctico el uso de la plantilla y se la recomiendo a mis compañeros.