Exercícios-Problemas (EP)

Semana 7:

https://www.facom.ufu.br/~albertini/grafos/

EP17: UVa 00924 - Spreading the News

In a large organization, everyone knows a lot of colleagues. However, friendship relations are kept with only a few of them, to whom news are told. Suppose that whenever an employee knows of a piece of news, he tells it to all his friends on the following day. So, on the first day, the source of the information tells it to his friends; on the second day, the source's friends tell it to their friends; on the third day, the friends of the source's friends' tell it to their friends; and so on. The goal is to determine: • the maximum daily boom size, which is the largest number of employees that, on a single day, hear the piece of news for the first time; and • the first boom day, which is the first day on which the maximum daily boom size occurs. Write a program that, given the friendship relations between the employees and the source of a piece of news, computes the maximum daily boom size and the first boom day of that information spreading process.

Input

The first line of the input contains the number E of employees ($1 \leq E \leq 2500$). Employees are numbered from 0 to $E - 1$. Each of the following E lines specifies the set of friends of an employee's (from employee 0 to employee $E - 1$). A set of friends contains the number of friends N ($0 \leq N < 15$), followed by N distinct integers representing the employee's friends. All integers are separated by a single space. The next line contains an integer T ($1 \leq T < 60$), which is the number of test cases. Each of the following T lines contains an employee, which represents the (unique) source of the piece of news in the test case.

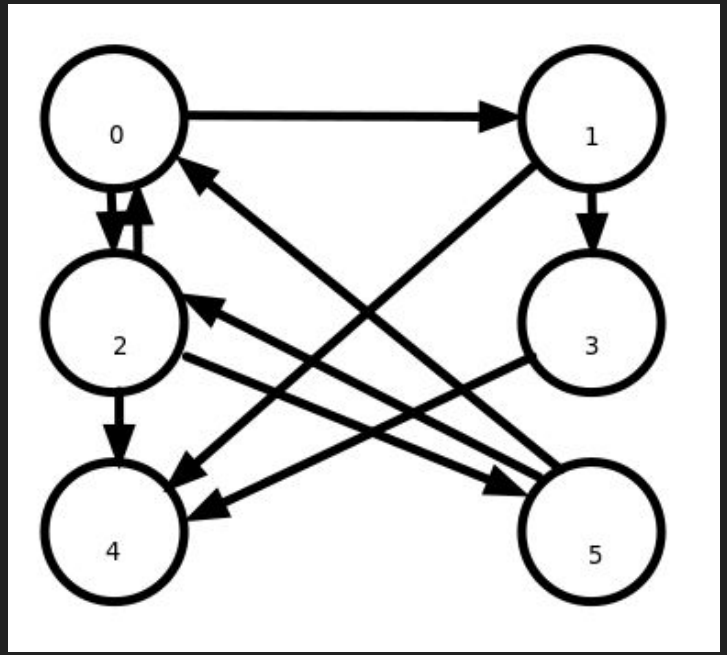
EP17: UVa 00924 - Spreading the News

Output

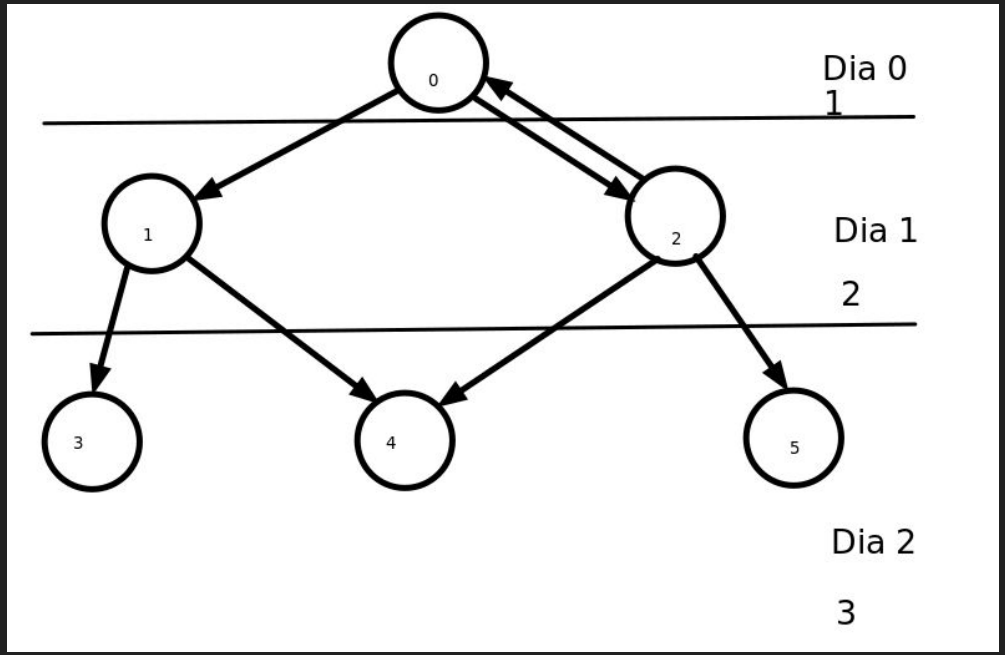
The output consists of T lines, one for each test case. If no employee (but the source) hears the piece of news, the output line contains the integer ‘0’. Otherwise, the output line contains two integers, M and D, separated by a single space, where M is the maximum daily boom size and D is the first boom day.

Sample Input	Sample Output
6 2 1 2 2 3 4 3 0 4 5 1 4 0 2 0 2 3 0 4 5	3 2 0 2 1

Rede de amigos



Dispersão da informação com source = 0




```
Camadas {0: 1, 1: 2, 2: 3}
```

```python
def main():
    E = int(input())  # Leitura do numero de funcionarios

    grafo = Grafo()  # Inicialização do grafo

    for i in range(E):  # Criando arestas que representam as amizades
        grafo.add_vertice(i)
        amigos = list(map(int, input().split()))
        amigos.pop(0)
        for amigo in amigos:
            grafo.add_aresta(i, amigo)

    T = int(input())  # Leitura do numero de casos

    results = []
    for _ in range(T):
        source = int(input())  # Leitura da fonte das novidades
        M, D = grafo.bfs(source)
        results.append((M, D))

    for M, D in results:
        if D == 0 or M == 0:  # Somente source ouviu a informação
            print(0)
        else:
            print(M, D)

if __name__ == "__main__":
    main()
```

- Leitura dos inputs
- Inicialização do grafo
- Criação das arestas
- Chamada da bfs
- Print dos resultados

- Estrutura do grafo: lista de adjacência
- Grafo direcional

```python
class Grafo:
    def __init__(self):
        self.adj = {}  # Dicionário para armazenar listas de adjacência

    def add_vertice(self, v):
        if v not in self.adj:
            self.adj[v] = []  # Adiciona um novo vértice com uma lista vazia de vizinhos

    def add_aresta(self, v1, v2):
        self.adj[v1].append(v2)  # Adiciona uma aresta direcionada de A -> B
```

```python
def bfs(self, v):
    fila = deque()
    fila.append((v, 0))  # Tupla (vertice, camada) -> source da novidade é a camada 0
    visitados = set()
    camadas = {}  # Quantidade de nos em cada camada

    while fila:
        vertice, camada = fila.popleft()

        if camada not in camadas:
            camadas[camada] = 1
        else:
            camadas[camada] += 1

        visitados.add(vertice)

        for vizinho in self.adj[vertice]:
            if vizinho not in visitados:
                fila.append((vizinho, camada + 1))
                visitados.add(vizinho)

    D = max(camadas, key=camadas.get) # First boom day
    M = camadas[D] # Maximum daily boom size

    return M, D
```

- Início da fila é o source da informação
- Contador de camadas (dias)
- Cada camada contém uma quantidade de vértices (amigos que receberam informação)
- Retorna a camada com maior quantidade vértices (amigos com informação)
- Número da camada -> First boom day
- Quantidade -> Maximum daily boom size

EP18: UVa 00949 - Getaway

Man-hat'em is a very well organized city. Its streets are layed out in a grid and the distance between each crossroad is always the same. However some streets are one way only and some don't even allow cars. Besides having to make a quick getaway Selma and Louis have one other problem.
The city has recently installed a surveillance system. Cameras have been installed in some crossroads but only one camera is monitored at each given time. The success of their getaway is drastically reduced if they are spotted during their escape so their escape route must avoid any crossing that is being monitored. Luckily, their friend Tony managed to get the surveillance system scheduling plans.
Your job will be to create a program that, given a certain partial map of the city and the surveillance system scheduling, will compute the optimum escape route. The following can be assumed to be always true:

- The robbery will always take place at the northwest of the map;
- The hideaway will always be at the southeast of the map;
- There is always a possible escape route;
- The time needed to get from one crossroad to the next is always the same, and, for simplicity sake, can be considered as being one time unit.
- The escape route must never pass a crossing that is being monitorized at that time;
- To avoid getting caught on camera Selma and Louis can wait any units of time at a given crossroad

EP18: UVa 00949 - Getaway

Input

The input le contains several test cases, each of them as describes below.
• The input starts by stating the number of vertical roads (nv) and horizontal roads (nh) in a single line. With 1 ≤ nv, nh ≤ 100.
• The following line will contain the number (r) of traffic restrictions in the city. With 0 ≤ r ≤ 500.
• Each of the following r lines will contain a restriction of the form: x1 y1 x2 y2. Meaning that traffic isn't allowed from the crossroad with coordinates (x1, y1) to the crossroad with coordinates (x2, y2).
• The next line will contain the number (m) of scheduled crossroad monitorizations. With 0 ≤ m ≤ 500.
• Each of the following m lines will contain the monitorization information in the form: t x y. Where t is the time counting from the start of the getaway and (x, y) are the coordinates of thecrossroad being monitored. All these m lines will have different values for t. With 0 ≤ t ≤ 500.

Output

For each test case, the output will contain a single line stating the minimum number of time units needed for a perfect getaway.
Explanation of the example output:

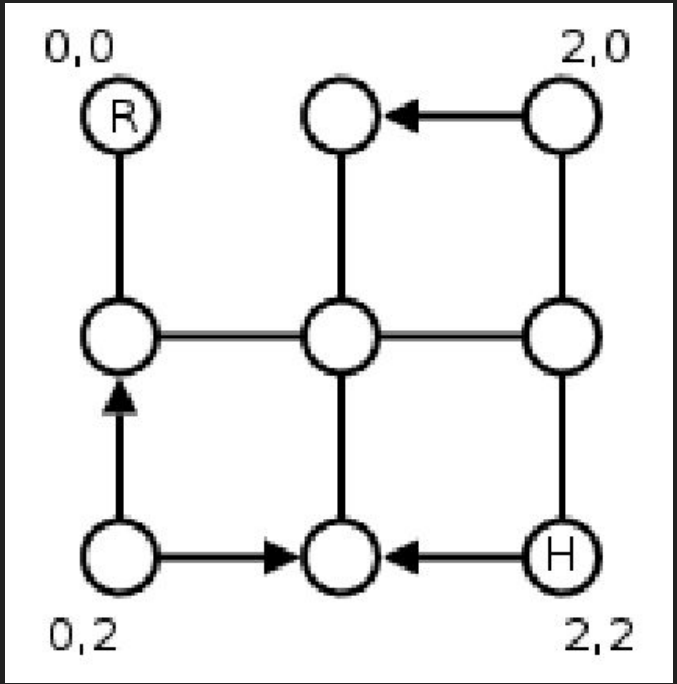
The plan starts at (0,0) at time 0. They then proceed to (0,1) where they intended to turn left heading to (1,1). Thanks to Tony, they know someone will be monitoring them at (1,1) by the time they get there, so they wait for one time unit. They arrive at (1,1) at time 3, where they wait another time unit as they know they will be monitored at (2,1) at time 4. After this second wait, they proceed to (2,1) and then to (2,2) where they arrive at time 6.

Sample Input

3 3
6
0 0 1 0
1 0 0 0
1 0 2 0
0 1 0 2
1 2 0 2
1 2 2 2
2
2 1 1
4 2 1

Sample Output

6



The plan starts at (0,0) at time 0. They then proceed to (0,1) where they intended to turn left heading to (1,1). Thanks to Tony, they know someone will be monitoring them at (1,1) by the time they get there, so they wait for one time unit. They arrive at (1,1) at time 3, where they wait another time unit as they know they will be monitored at (2,1) at time 4. After this second wait, they proceed to (2,1) and then to (2,2) where they arrive at time 6.

```java
public class Main {
    Run | Debug
    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);
        int N = sc.nextInt();
        int M = sc.nextInt();
        Grafo grafo = new Grafo(N*M);
        GrafoTempo[] grafotempo = new GrafoTempo[500];
        for(int i = 0;i <N ; i++){
            for(int j = 0;j<M ; j++){
                if(j!=0){
                grafo.addAresta(i,j,i,j-1);
                }if(i!=0){
                    grafo.addAresta(i,j,i-1,j);
                }

            }
        }
        int casos = sc.nextInt();
        for(int i=0;i<casos;i++){
            grafo.removerAresta(sc.nextInt(),sc.nextInt(),sc.nextInt(),sc.nextInt());
        }
        casos = sc.nextInt();
        for(int i=0;i<casos;i++){
            grafotempo[i]= new GrafoTempo(sc.nextInt(),sc.nextInt(),sc.nextInt());
        }

        grafo.BFS(x1:0, y1:0, N-1, M-1,grafotempo,casos);
    }
}
```

Monta a matriz.

Remove as arestas dadas pelo exercício.

Armazena as arestas com câmeras numa estrutura separada.

chama BFS.

```java
// Encontra o menor caminho a partir das coordenadas (x1, y1) para (x2, y2) usando BFS
void BFS(int x1, int y1, int x2, int y2, GrafoTempo[] grafotempo,int casos) {
    String coordOrigem = x1 + "," + y1;
    String coordDestino = x2 + "," + y2;

    int origem = coordenadas.get(coordOrigem);
    int destino = coordenadas.get(coordDestino);

    boolean visitados[] = new boolean[V]; // Vetor de visitados
    int distancia[] = new int[V]; // Vetor de distâncias

    // Inicializa todos os vértices como não visitados e com distância infinita
    for (int i = 0; i < V; ++i) {
        visitados[i] = false;
        distancia[i] = Integer.MAX_VALUE;
    }

    // Marca o vértice de origem como visitado e com distância 0
    visitados[origem] = true;
    distancia[origem] = 0;

    Queue<Integer> fila = new LinkedList<>(); // Fila para BFS
    fila.add(origem);

    while (!fila.isEmpty()) {
        int u = fila.poll();

        // Itera sobre todos os vértices adjacentes a 'u'
        for (Integer v : adj.get(u)) {
            // Se 'v' não foi visitado, marca como visitado, atualiza a distância e coloca na fila
            if (!visitados[v]) {
                visitados[v] = true;
                for(int i=0;i<casos;i++){
                    int testa = coordenadas.get(grafotempo[i].coord1);
                    if (testa == u && grafotempo[i].tempo == distancia[u]){
                        distancia[u]++;
                        break;
                    }
                }
                distancia[v] = distancia[u] + 1;
                fila.add(v);
            }
        }
    }

    // A distância até o destino é armazenada em distancia[destino]
    System.out.print( distancia[destino]);
```