Graph Representation in Programming Assignments

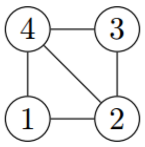
In programming assignments, graphs are given as follows. The first line contains non-negative integers n and m – the number of vertices and the number of edges respectively. The vertices are always numbered from 1 to n. Each of the following m lines defines an edge in the format u v where $1 \leq u, v \leq n$ are endpoints of the edge. If the problem deals with an undirected graph this defines an undirected edge between u and v, In case of a directed graph this defines a directed edge from u to v. If the problem deals with a weighted graph then each edge is given as u v w where u and v are vertices and w is a weight.

It is guaranteed that a given graph is simple. That is, it does not contain self-loops (edge going from a vertex to itself) and parallel edges.

Examples:

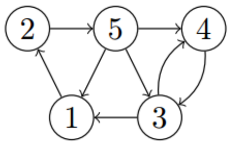
- An undirected graph with four vertices and five edges:

```
4 5
2 1
4 3
1 4
2 4
3 2
```



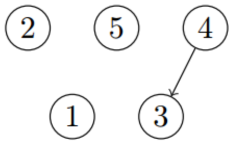
- A directed graph with five vertices and eight edges:

```
5 8
4 3
1 2
3 1
3 4
2 5
5 1
5 4
5 3
```



- A directed graph with five vertices and one edge:

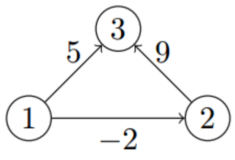
```
5 1
4 3
```



  Note that the vertices $1, 2,$ and 5 are isolated (have no adjacent edges), but they are still present in the graph.

- A weighted directed graph with three vertices and three edges:

```
3 3
2 3 9
1 3 5
1 2 -2
```



1 Computing the Minimum Number of Flight Segments

Problem Introduction

You would like to compute the minimum number of flight segments to get from one city to another one. For this, you construct the following undirected graph: vertices represent cities, there is an edge between two vertices whenever there is a flight between the corresponding two cities. Then, it suffices to find a shortest path from one of the given cities to the other one.

Problem Description

Task. Given an *undirected* graph with n vertices and m edges and two vertices u and v, compute the length of a shortest path between u and v (that is, the minimum number of edges in a path from u to v).

Input Format. A graph is given in the standard format. The next line contains two vertices u and v.

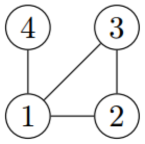
Constraints. $2 \leq n \leq 10^5$; $0 \leq m \leq 10^5$; $1 \leq u, v \leq n$; $u \neq v$.

Output Format. Output the minimum number of edges in a path from u to v, or -1 if there is no path.

Sample 1.

Input:

```
4 4
1 2
4 1
2 3
3 1
2 4
```



Output:

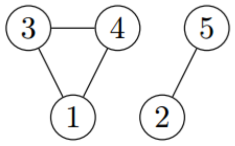
```
2
```

There is a unique shortest path between vertices 2 and 4 in this graph: $2 - 1 - 4$.

Sample 2.

Input:

```
5 4
5 2
1 3
3 4
1 4
3 5
```



Output:

```
-1
```

There is no path between vertices 3 and 5 in this graph.

Starter Files

The starter solutions for this problem read the input data from the standard input, pass it to a blank procedure, and then write the result to the standard output. You are supposed to implement your algorithm in this blank procedure if you are using `C++`, `Java`, or `Python3`. For other programming languages, you need to implement a solution from scratch. Filename: `bfs`

What To Do

To solve this problem, it is enough to implement carefully the corresponding algorithm covered in the lectures.

2 Checking whether a Graph is Bipartite

Problem Introduction

An undirected graph is called *bipartite* if its vertices can be split into two parts such that each edge of the graph joins two vertices form different parts. Bipartite graphs arise naturally in applications where a graph is used to model connections between objects of two different types (say, boys and girls; or students and dormitories).

An alternative definition is the following: a graph is bipartite if its vertices can be colored with two colors (say, black and white) such that endpoints of each edge have different colors.

Problem Description

Task. Given an undirected graph with n vertices and m edges, check whether it is bipartite.

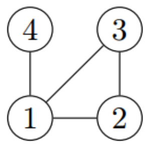
Constraints. $1 \leq n \leq 10^5$; $0 \leq m \leq 10^5$.

Output Format. Output 1 if the graph is bipartite and 0 otherwise.

Sample 1.

Input:

```
4 4
1 2
4 1
2 3
3 1
```



Output:

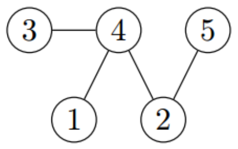
```
0
```

This graph is not bipartite. To see this assume that the vertex 1 is colored white. Then the vertices 2 and 3 should be colored black since the graph contains the edges {1,2} and {1,3}. But then the edge {2,3} has both endpoints of the same color.

Sample 2.

Input:

```
5 4
5 2
4 2
3 4
1 4
```



Output:

```
1
```

This graph is bipartite: assign the vertices 4 and 5 the white color, assign all the remaining vertices the black color.

Starter Files

The starter solutions for this problem read the input data from the standard input, pass it to a blank procedure, and then write the result to the standard output. You are supposed to implement your algorithm in this blank procedure if you are using `C++`, `Java`, or `Python3`. For other programming languages, you need to implement a solution from scratch. Filename: `bipartite`

What To Do

Adapt the breadth-first search to solve this problem.

3 Computing the Minimum Cost of a Flight

Problem Introduction

Now, you are interested in minimizing not the number of segments, but the total cost of a flight. For this, you construct a weighted graph: the weight of an edge from one city to another one is the cost of the corresponding flight.

Problem Description

Task. Given a *directed* graph with positive edge weights and with n vertices and m edges as well as two vertices u and v, compute the weight of a shortest path between u and v (that is, the minimum total weight of a path from u to v).

Input Format. A graph is given in the standard format. The next line contains two vertices u and v.

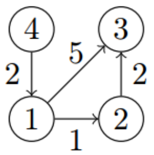
Constraints. $1 \leq n \leq 10^4$; $0 \leq m \leq 10^5$; $1 \leq u, v \leq n$; $u \neq v$. Edge weights are non-negative integers not exceeding 10^3.

Output Format. Output the minimum weight of a path from u to v, or -1 if there is no path.

Sample 1.

Input:

```
4 4
1 2 1
4 1 2
2 3 2
1 3 5
1 3
```



Output:

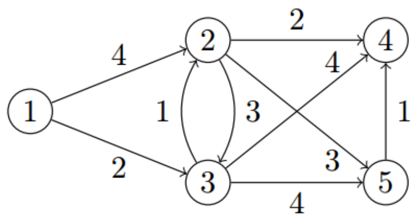
```
3
```

There is a unique shortest path from vertex 1 to vertex 3 in this graph ($1 \to 2 \to 3$), and it has weight 3.

Sample 2.

Input:

```
5 9
1 2 4
1 3 2
2 3 2
3 2 1
2 4 2
3 5 4
5 4 1
2 5 3
3 4 4
1 5
```



Output:

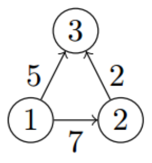
```
6
```

There are two paths from 1 to 5 of total weight 6: $1 \to 3 \to 5$ and $1 \to 3 \to 2 \to 5$.

Sample 3.

Input:

```
3 3
1 2 7
1 3 5
2 3 2
3 2
```



Output:

```
-1
```

There is no path from 3 to 2.

Starter Files

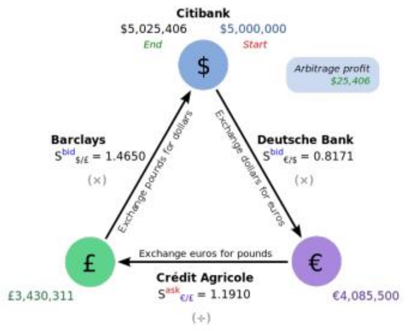
The starter solutions for this problem read the input data from the standard input, pass it to a blank procedure, and then write the result to the standard output. You are supposed to implement your algorithm in this blank procedure if you are using `C++`, `Java`, or `Python3`. For other programming languages, you need to implement a solution from scratch. Filename: `dijkstra`

What To Do

To solve this problem, it is enough to implement carefully the corresponding algorithm covered in the lectures.

4 Detecting Anomalies in Currency Exchange Rates

Problem Introduction



You are given a list of currencies $c_1, c_2, \dots, c_n$ together with a list of exchange rates: r_{ij} is the number of units of currency c_j that one gets for one unit of c_i. You would like to check whether it is possible to start with one unit of some currency, perform a sequence of exchanges, and get more than one unit of the same currency. In other words, you would like to find currencies $c_{i_1}, c_{i_2}, \dots, c_{i_k}$ such that $r_{i_1,i_2} \cdot r_{i_2,i_3} \cdot \dots \cdot r_{i_{k-1},i_k} \cdot r_{i_k,i_1} > 1$. For this, you construct the following graph: vertices are currencies $c_1, c_2, \dots, c_n$, the weight of an edge from c_i to c_j is equal to $-\log r_{ij}$. Then it suffices to check whether there is a negative cycle in this graph. Indeed, assume that a cycle $c_i \to c_j \to c_k \to c_i$ has negative weight. This means that $-(\log r_{ij} + \log r_{jk} + \log r_{ki}) < 0$ and hence $\log r_{ij} + \log r_{jk} + \log r_{ki} > 0$. This, in turn, means that

$$r_{ij} r_{jk} r_{ki} = 2^{\log r_{ij}} 2^{\log r_{jk}} 2^{\log r_{ki}} = 2^{\log r_{ij} + \log r_{jk} + \log r_{ki}} > 1.$$

Problem Description

Task. Given a *directed* graph with possibly negative edge weights and with n vertices and m edges, check whether it contains a cycle of negative weight.

Input Format. A graph is given in the standard format.

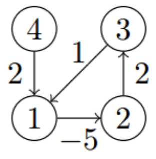
Constraints. $1 \le n \le 10^3$; $0 \le m \le 10^4$; edge weights are integers of absolute value at most 10^3.

Output Format. Output 1 if the graph contains a cycle of negative weight and 0 otherwise.

Sample 1.

Input:

```
4 4
1 2 -5
4 1 2
2 3 2
3 1 1
```



Output:

```
1
```

The weight of the cycle $1 \to 2 \to 3$ is equal to -2, that is, negative.

Starter Files

The starter solutions for this problem read the input data from the standard input, pass it to a blank procedure, and then write the result to the standard output. You are supposed to implement your algorithm in this blank procedure if you are using `C++`, `Java`, or `Python3`. For other programming languages, you need to implement a solution from scratch. Filename: `negative_cycle`

What To Do

To solve this problem, it is enough to implement carefully the corresponding algorithm covered in the lectures.

5 Advanced Problem: Exchanging Money Optimally

It is strongly recommended that you start solving advanced problems only when you are done with the basic problems (for some advanced problems, algorithms are not covered in the video lectures and require additional ideas to be solved; for some other advanced problems, algorithms are covered in the lectures, but implementing them is a more challenging task than for other problems).

Problem Introduction

Now, you would like to compute an optimal way of exchanging the given currency c_i into all other currencies. For this, you find shortest paths from the vertex c_i to all other vertices.

Problem Description

Task. Given a *directed* graph with possibly negative edge weights and with n vertices and m edges as well as its vertex s, compute the length of the shortest paths from s to all other vertices of the graph.

Constraints. $1 \leq n \leq 10^3$; $0 \leq m \leq 10^4$; $1 \leq s \leq n$; edge weights are integers of absolute value at most 10^9.

Output Format. For all vertices i from 1 to n output the following on a separate line:

- "*", if there is no path from s to i.
- "-", if there is a path from s to i, but there is no shortest path from s to i (that is, the distance from s to i is $-\infty$).
- the length of a shortest path otherwise.

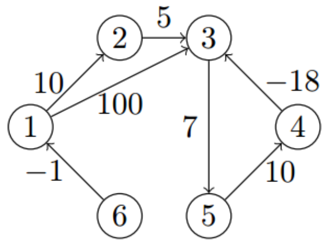
Sample 1.

Input:

```
6 7
1 2 10
2 3 5
1 3 100
3 5 7
5 4 10
4 3 -18
6 1 -1
1
```

Output:

```
0
10
-
-
-
*
```

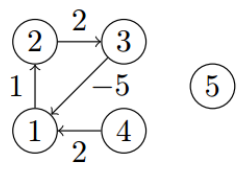


The first line of the output states that the distance from 1 to 1 is equal to 0. The second one shows that the distance from 1 to 2 is 10 (the corresponding path is $1 \to 2$). The next three lines indicate that the distance from 1 to vertices 3, 4, and 5 is equal to $-\infty$: indeed, one first reaches the vertex 3 through edges $1 \to 2 \to 3$ and then makes the length of a path arbitrarily small by making sufficiently many walks through the cycle $3 \to 5 \to 4$ of negative weight. The last line of the output shows that there is no path from 1 to 6 in this graph.

Sample 2.

Input:

```
5 4
1 2 1
4 1 2
2 3 2
3 1 -5
4
```



Output:

```
-
-
-
0
*
```

In this case, the distance from 4 to vertices 1, 2, and 3 is $-\infty$ since there is a negative cycle $1 \rightarrow 2 \rightarrow 3$ that is reachable from 4. The distance from 4 to 4 is 0. There is no path from 4 to 5.

Starter Files

The starter solutions for this problem read the input data from the standard input, pass it to a blank procedure, and then write the result to the standard output. You are supposed to implement your algorithm in this blank procedure if you are using `C++`, `Java`, or `Python3`. For other programming languages, you need to implement a solution from scratch. Filename: `shortest_paths`

What To Do

To solve this problem, it is enough to implement carefully the corresponding algorithm covered in the lectures.