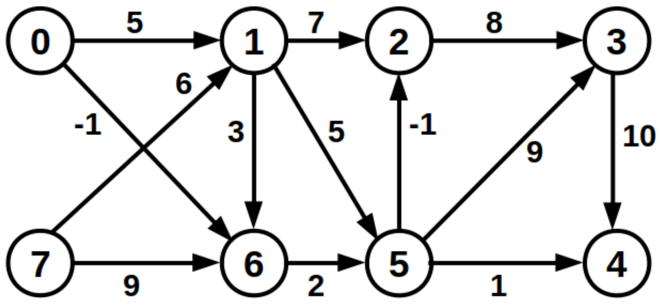
Algorithms Laboratory (CS29203)
Assignment 7: Graph algorithms
Department of CSE, IIT Kharagpur

9th November 2023

Question-1 (50 points)

In this question, we will be performing a variation of shorest path algorithm. Consider a weighted and directed graph. We are interested in finding the minimum cost of the shorest path between two specified vertices (i.e. from a source to a destination) where the path with consist of **exactly** n number of edges.

For example, consider the following graph:



Let source = 0, destination = 3, number of edges (n) = 4. The graph has 3 different paths from source 0 to destination 3 with 4 edges as follows:

- $0 \to 1 \to 5 \to 2 \to 3$ (cost = 17)
- $0 \to 1 \to 6 \to 5 \to 3$ (cost = 19)
- $0 \to 6 \to 5 \to 2 \to 3$ (cost = 8)

So the minimum cost is 8.

While you can use any approach to solve the problem, a modified BFS traversal can be performed to find the minimum cost. The BFS does not explore the already discovered vertices again, but here we can do the opposite. To cover all possible paths from source to destination, we can remove this check from BFS. However removing this check will cause the program to go into an infinite loop. We can easily handle that if we do not consider nodes having a BFS depth of more than n. We can maintain the following three attributes in a BFS queue node: current vertex number, current depth of BFS (i.e. how far the current node is from the source), and the cost of the current path chosen so far. Whenever we encounter any node whose cost of a path is more and required BFS depth is reached, we will update the result. The BFS will terminate when we have explored every path in the given graph or BFS depth exceeds n.

Your task is to implement the algorithm to solve the problem. The complexity of the algorithm should not exceed $O(V + E)$. You can assume that the input is taken as a series of tuple of edges: (source vertex, destination vertex, weight). For example the tuple $(7, 6, 9)$ denotes an edge that connects vertex # 7 and vertex # 6, and the edge weight is 9. The set of all edges can be stored in a multidimensional array.

Example:

```
(Input) Edges: [[0, 6, -1], [0, 1, 5], [1, 6, 3], [1, 5, 5], [1, 2, 7], [2, 3, 8], [3, 4, 10],
                [5, 2, -1], [5, 3, 9], [5, 4, 1], [6, 5, 2], [7, 6, 9], [7, 1, 6]]

(Output) Minimum cost: 8
```

Question-2 (50 points)

In this question we will be performing a variation of topological sort algorithm. Consider your course curriculum having n number of courses labelled from 0 to $n-1$. Each course (i) also has a prerequisite, which is stored in an array prereq$[i] = [a_i, b_i]$, which means that course b_i is the prerequisite of course a_i. For example, the pair $[0, 1]$ indicates that to take course #0 you have to first take course #1.

Given the total number of courses and the prerequisite array, you have to determine whether you can finish all the courses or not (return `true` if you can, otherwise return `false`).

For example, let $n = 2$, prereq $= [[1, 0]]$. Then the answer is `true`. There are a total of 2 courses you have to take. To take course 1 you should have finished course 0. So it is possible. However if $n = 2$, prereq $= [[1, 0], [0, 1]]$, then the answer is `false`. There are a total of 2 courses you have to take. To take course 1 you should have finished course 0, and to take course 0 you should also have finished course 1. So it is impossible. Your task is to implement the algorithm to solve the problem.

Basically the problem is essentially checking if there is a cycle in a directed graph. Each course can be represented as a node, and each prerequisite relationship can be represented as a directed edge from the prerequisite course to the target course. We can use a Topological Sort algorithm to detect if there is a cycle in the directed graph. The algorithm works by first finding all the courses that have no prerequisites and adding them to a queue. Then we remove these courses and their corresponding edges from the graph. We keep doing this until there are no courses with no prerequisites left. If we have removed all the courses successfully, then there is no cycle in the graph and we can finish all the courses.

If there is a cycle in the graph, there will always be at least one course that cannot be taken because it depends on another course that cannot be taken. This means that we will never be able to remove all the courses with no prerequisites, and we cannot finish all the courses.

Example 1:

```
(Input) n = 4,   prereq = [[1, 0], [2, 1], [3, 2]]

(Output) true
```

Example 2:

```
(Input) n = 2,   prereq = [[1, 0], [0, 1]]

(Output) false
```