Dept. of Electronics and Electrical Communication Engineering

Indian Institute of Technology Kharagpur

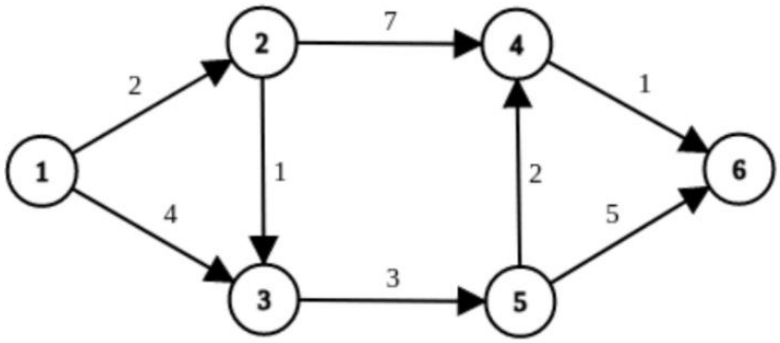
ALGORITHMS
(EC31205)

CODING TASK-2

AUTHOR: JAYA KISHNANI
ROLL NUMBER: 20EC30020
DATE: 18/09/2022
THEME: GRAPH ALGORITHMS

Problem 1: Dijkstra's Algorithm for directed graph

For the given directed graph



Let the source vertex be node 1 and its reference cost be 0.
The cost table for the above graph is shown below with yellow marked node being the smallest node's cost for the cost update

node 2	node 3	node 4	node 5	node 6
2	4	∞	∞	∞
2	3	∞	∞	∞
2	3	∞	6	∞
2	3	8	6	11
2	3	8	6	9

So by the above table we can say that the distance for nodes 2, 3, 4, 5, 6 is 2, 3, 8, 6, 9 respectively from the source node 1.

The time complexity for the dijkstra algorithm is O(E + VlogV), where E and V are number of edges and vertices respectively.

This can be verified from the output of the code while running Dijkstra's algorithm.
Output

```
DIJKSTRA'S ALGORITHM


We found the following best path to reach node-2 with the cost as 2.
1 -> 2

We found the following best path to reach node-3 with the cost as 3.
1 -> 2 -> 3

We found the following best path to reach node-4 with the cost as 8.
```

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

We found the following best path to reach node-5 with the cost as 6.
1 -> 2 -> 3 -> 5

We found the following best path to reach node-6 with the cost as 9.
1 -> 2 -> 3 -> 5 -> 4 -> 6
```

Problem 2: Bellman Ford's algorithm for shortest path

a) We know that by introducing negative weights in the graph dijkstra's algorithm may or may not be satisfactory. So the bellman-ford algorithm is one of the most classical way to deal with graphs containing negative edges. The algorithm is less efficient than dijkstra as it takes **O(V.E)** time whereas dijkstra takes **O(E+VlogV)** time, where E and V are number of edges and vertices respectively.

The above graph is defined as following in the code.

```
nodes = ["1", "2", "3", "4", "5", "6"]

init_graph = {}
for node in nodes:
    init_graph[node] = {}

init_graph["1"]["2"] = 2
init_graph["2"]["4"] = -7
init_graph["4"]["6"] = 1
init_graph["5"]["6"] = 5
init_graph["5"]["4"] = 2
init_graph["1"]["3"] = 4
init_graph["3"]["5"] = -3
init_graph["2"]["3"] = 1
```

The output of running bellman-ford algorithm and comparing it with dijkstra's algorithm is as follows:

```
BELLMAN FORD ALGORITHM

We found the following best path to reach node-2 with the cost as 2.
1 -> 2

We found the following best path to reach node-3 with the cost as 3.
1 -> 2 -> 3
```

```
We found the following best path to reach node-4 with the cost as -5.
1 -> 2 -> 4

We found the following best path to reach node-5 with the cost as 0.
1 -> 2 -> 3 -> 5

We found the following best path to reach node-6 with the cost as -4.
1 -> 2 -> 4 -> 6

DIJKSTRA ALGORITHM


We found the following best path to reach node-2 with the cost as 2.
1 -> 2

We found the following best path to reach node-3 with the cost as 3.
1 -> 2 -> 3

We found the following best path to reach node-4 with the cost as -5.
1 -> 2 -> 4

We found the following best path to reach node-5 with the cost as 0.
1 -> 2 -> 3 -> 5

We found the following best path to reach node-6 with the cost as -4.
1 -> 2 -> 4 -> 6

Dijkstra's algorithm works equivalently as bellman ford’s
```

Bellman-ford’s algorithm applies ***dynamic programming approach*** whereas dijkstra algorithm applies ***greedy approach***.

But as mentioned the dijkstra algorithm might not work always with negative edges.
So the following changes were made in the graph

```
nodes = ["1", "2", "3", "4", "5", "6"]

init_graph = {}
for node in nodes:
    init_graph[node] = {}

init_graph["1"]["2"] = 2
init_graph["2"]["4"] = 7
init_graph["4"]["6"] = 1
init_graph["5"]["6"] = 5
init_graph["5"]["4"] = -5
init_graph["1"]["3"] = 4
init_graph["3"]["5"] = 9
init_graph["2"]["3"] = 1
```

The following output was observed:

```
BELLMAN FORD ALGORITHM


We found the following best path to reach node-2 with the cost as 2.
1 -> 2

We found the following best path to reach node-3 with the cost as 3.
1 -> 2 -> 3

We found the following best path to reach node-4 with the cost as 7.
1 -> 2 -> 3 -> 5 -> 4

We found the following best path to reach node-5 with the cost as 12.
1 -> 2 -> 3 -> 5

We found the following best path to reach node-6 with the cost as 8.
1 -> 2 -> 3 -> 5 -> 4 -> 6

DIJKSTRA ALGORITHM


We found the following best path to reach node-2 with the cost as 2.
1 -> 2

We found the following best path to reach node-3 with the cost as 3.
1 -> 2 -> 3

We found the following best path to reach node-4 with the cost as 7.
1 -> 2 -> 3 -> 5 -> 4

We found the following best path to reach node-5 with the cost as 12.
1 -> 2 -> 3 -> 5

We found the following best path to reach node-6 with the cost as 10.
1 -> 2 -> 3 -> 5 -> 4 -> 6

DIJKSTRA ALGORITHM FAILED!!
```

We see that the dijkstra's algorithm failed in this case.

b) The bellman ford algorithm is not an ideal algorithm it may also fail when a negative weight cycle is detected in the graph.

The changes were made in the above graph are as follows:

```
nodes = ["1", "2", "3", "4", "5", "6"]

init_graph = {}
for node in nodes:
    init_graph[node] = {}

init_graph["1"]["2"] = 2
init_graph["2"]["4"] = 7
init_graph["4"]["6"] = 1
init_graph["5"]["6"] = 5
init_graph["5"]["4"] = 2
init_graph["1"]["3"] = 4
init_graph["3"]["5"] = 3
init_graph["2"]["3"] = 1
init_graph["4"]["3"] = -7
```

The output is as follows:

```
The shortest path cost to the target node is given in the form of
dictionary

After n-1 iterations (where n is the number of vertices)

{'1': 0, '2': 2, '3': -2, '4': 4, '5': 2, '6': 6}

After n iterations (where n is the number of vertices)

{'1': 0, '2': 2, '3': -3, '4': 3, '5': 1, '6': 5}

After n+1 iterations (where n is the number of vertices)

{'1': 0, '2': 2, '3': -4, '4': 2, '5': 0, '6': 4}

BELLMAN FORD ALGORITHM FAILED!!
```

So we observe that even after n-1 iterations (where n is the number of vertices of the graph) the cost of the nodes are getting updated, whereas it shouldn't. ***So here the bellman ford algorithm fails in case of negative weight cycles.***