NORTHWESTERN POLYTECHNIC
UNIVERSITY

Maze Project

Prepared For:

Mr. Henry Chang
Algorithms & Structured Programming CS455
Spring 2021
Northwestern Polytechnic University

Prepared By:

Ms. Nagalla, Santhi Sree ID:19568

Table of Contents

Introduction

- Maze solving is the act of finding a route through the maze from the start to finish. Some maze solving methods are designed to be used inside the maze by a traveler with no prior knowledge of the maze, whereas others are designed to be used by a person or computer program that can see the whole maze at once.

Maze-solving algorithms

- There are a number of different maze-solving algorithms, that is, automated methods for the solving of mazes.
  - ➢ Wall follower
  - ➢ Random mouse algorithm
  - ➢ Pledge algorithm
  - ➢ Trémaux's algorithm
  - ➢ Dead-end filling

Maze-solving algorithms - Continues

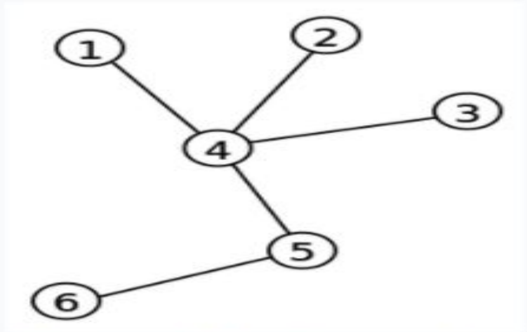
- ➢ Maze-routing algorithm
- ➢ Recursive algorithm
- ➢ Shortest path algorithm

- Mazes containing no loops are known as "simply connected", or "perfect" mazes, and are equivalent to a tree in graph theory. Thus many maze-solving algorithms are closely related to graph theory.

What is a Graph Theory?

- In graph theory, a tree is an undirected graph in which any two vertices are connected by exactly one path, or equivalently a connected acyclic undirected graph. A forest is an undirected graph in which any two vertices are connected by at most one path, or equivalently an acyclic undirected graph, or equivalently a disjoint union of trees.

Trees



A labeled tree with 6 vertices and 5 edges.

Vertices	v
Edges	$v - 1$
Chromatic number	2 if $v > 1$

Table of graphs and parameters

Shortest path algorithm

- When a maze has multiple solutions, the solver may want to find the shortest path from start to finish. One such algorithm finds the shortest path by implementing a breadth-first search, while another, the A* algorithm, uses a heuristic technique. The breadth-first search algorithm uses a queue to visit cells in increasing distance order from the start until the finish is reached.

Dijkstra’s Algorithm

- One algorithm for finding the shortest path from a starting node to a target node in a weighted graph is **Dijkstra's algorithm**.The algorithm creates a tree of shortest paths from the starting vertex, the source, to all other points in the graph.Each visited cell needs to keep track of its distance from the start or which adjacent cell nearer to the start caused it to be added to the queue. When the finish location is found, follow the path of cells backwards to the start, which is the shortest path. The breadth-first search in its simplest form has its limitations, like finding the shortest path in weighted graphs.

Bellman Ford’s Algorithm

- As with Dijkstra’s algorithm, the Bellman-Ford algorithm is one of the SSSP(Single Source Shortest Path) algorithms. Therefore, it calculates the shortest path from a starting source node to all the nodes inside a weighted graph. However, the concept behind the Bellman-Ford algorithm is different from Dijkstra’s.

Bellman Ford’s Algorithm - Analysis

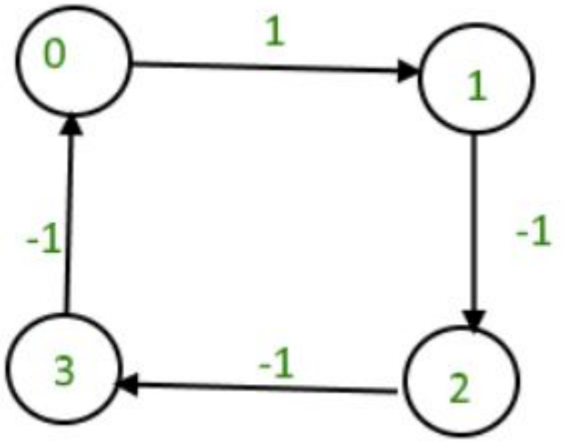
- ➢ The Bellman-Ford algorithm propagates correct distance estimates to all nodes in a graph in V-1 steps, unless there is a **negative weight cycle.**
- ➢ If there is a **negative weight cycle**, you can go on relaxing its nodes indefinitely.
- The process will stop when either of the following conditions is met:
  - The minimum cost is not changed for a cycle.
  - V-1 steps is reached.

Example for Negative cycle in a Graph

- A **negative weight cycle is** a **cycle** with **weights** that sum to a **negative** number.

```
Input : 4 4
        0 1 1
        1 2 -1
        2 3 -1
        3 0 -1

Output : Yes
  The graph contains a
  negative cycle.
```

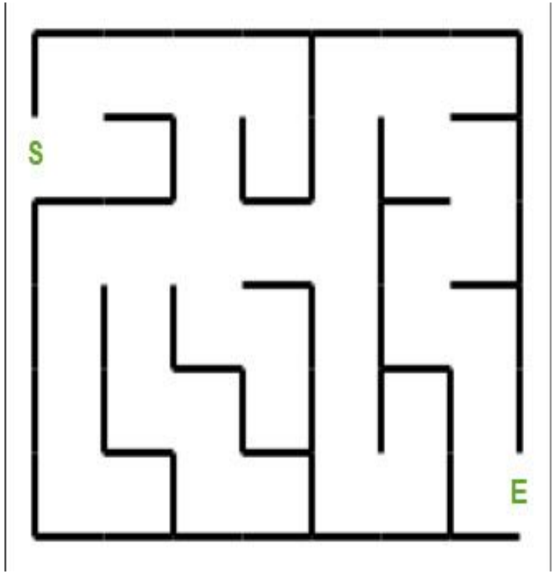


Difference Between Dijkstra’s and Bellman Ford’s Algorithm

- **Bellman-Ford** algorithm is a single-source shortest path algorithm, so when you have negative edge weight then it can detect negative cycles in a graph. The only **difference between** the two is that **Bellman-Ford** is also capable of handling negative weights whereas **Dijkstra** Algorithm can only handle positives.

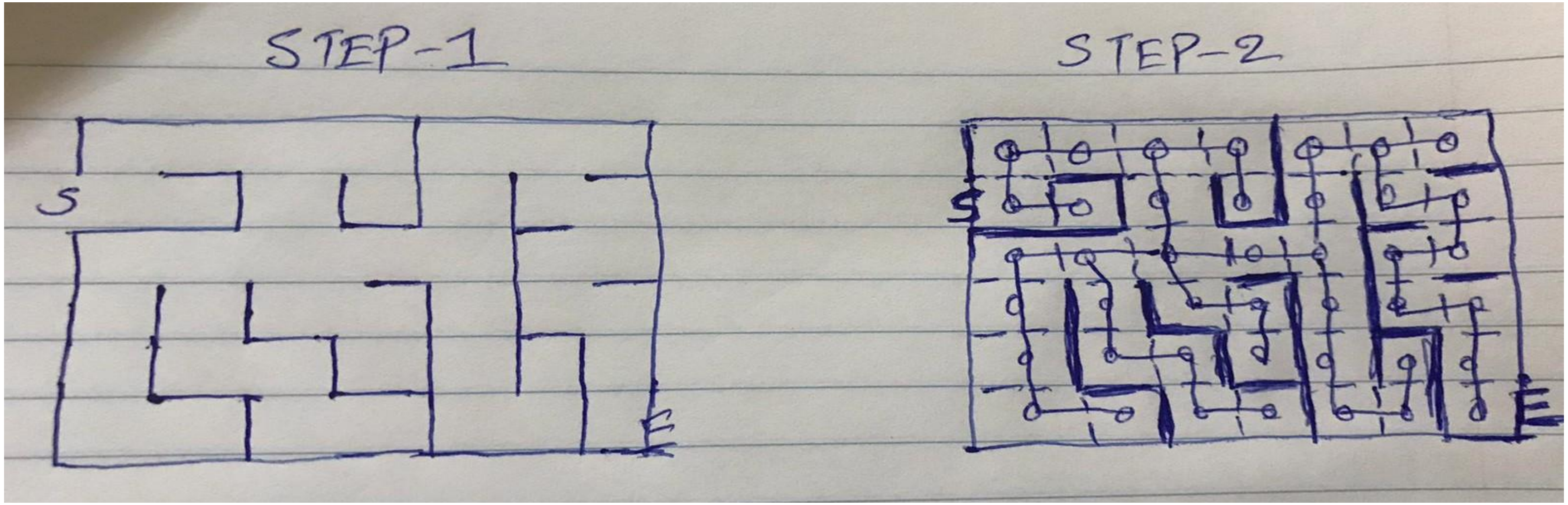
Steps to follow Shortest-Path

- ➢ Initialize distances according to the **algorithm**.
- ➢ Pick first node and **calculate** distances to adjacent nodes.
- ➢ Pick next node with minimal **distance** repeat adjacent node **distance** calculations.
- ➢ Final result of **shortest-path** tree.



Convert Maze to Graph

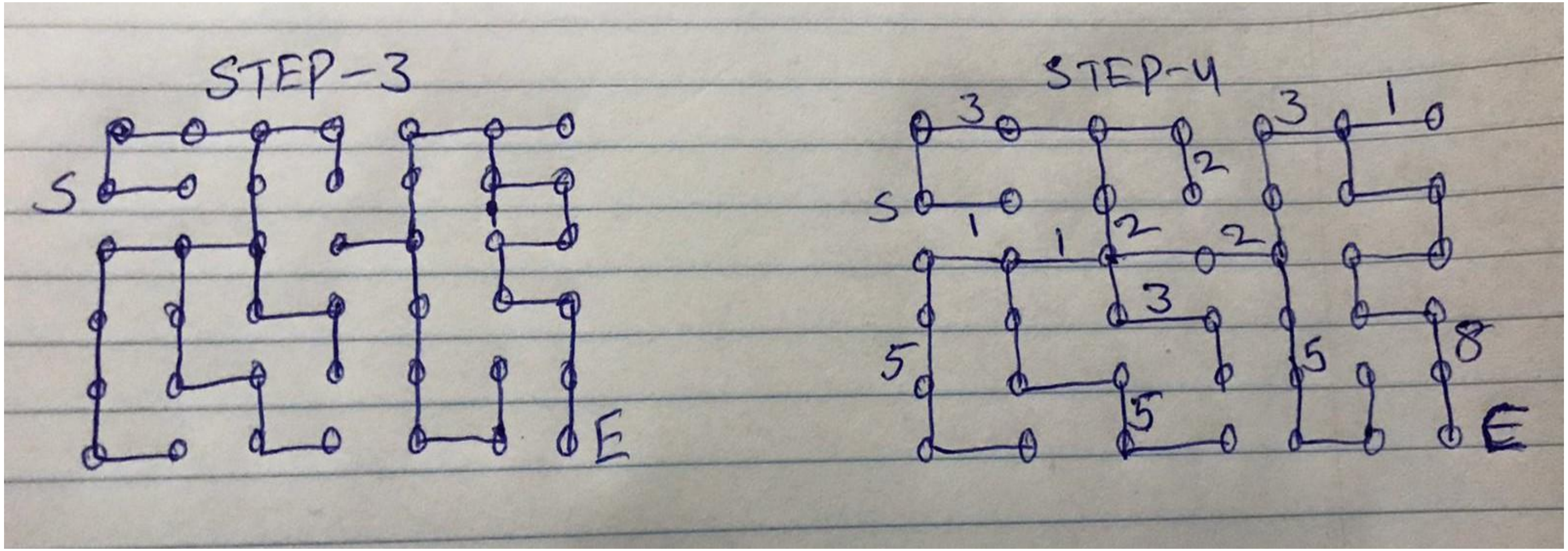
Step 1: Initialize distances according to the algorithm.



Convert Maze to graph with weights

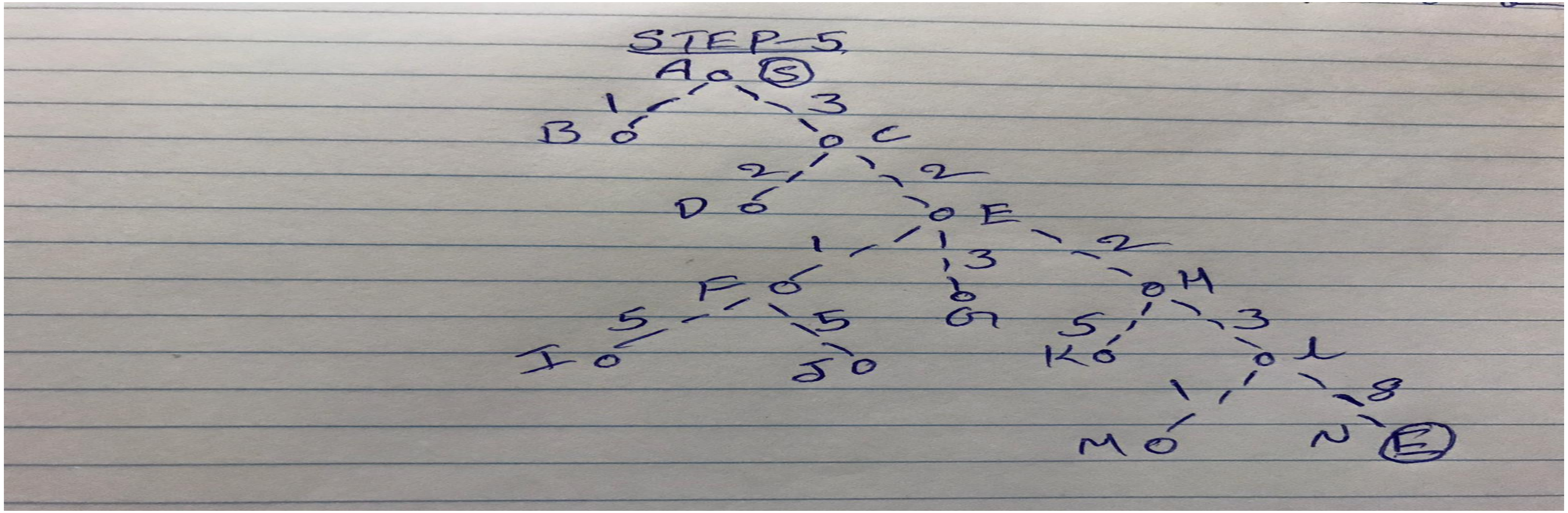
Step 2: Pick first node and calculate distances to adjacent nodes.

Step 3: Pick next node with minimal distance repeat adjacent node distance calculations.



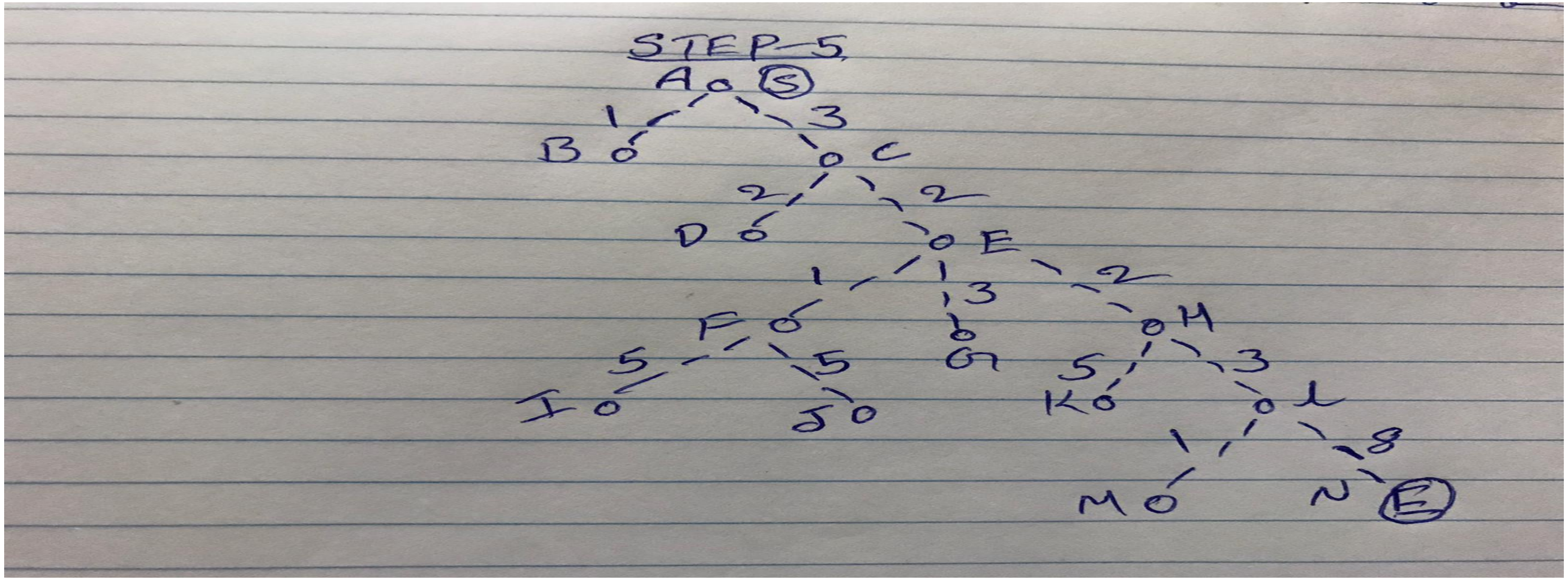
Tree or Graph

- A tree is an undirected graph in which any two vertices are connected by exactly one path, or equivalently a connected acyclic undirected graph

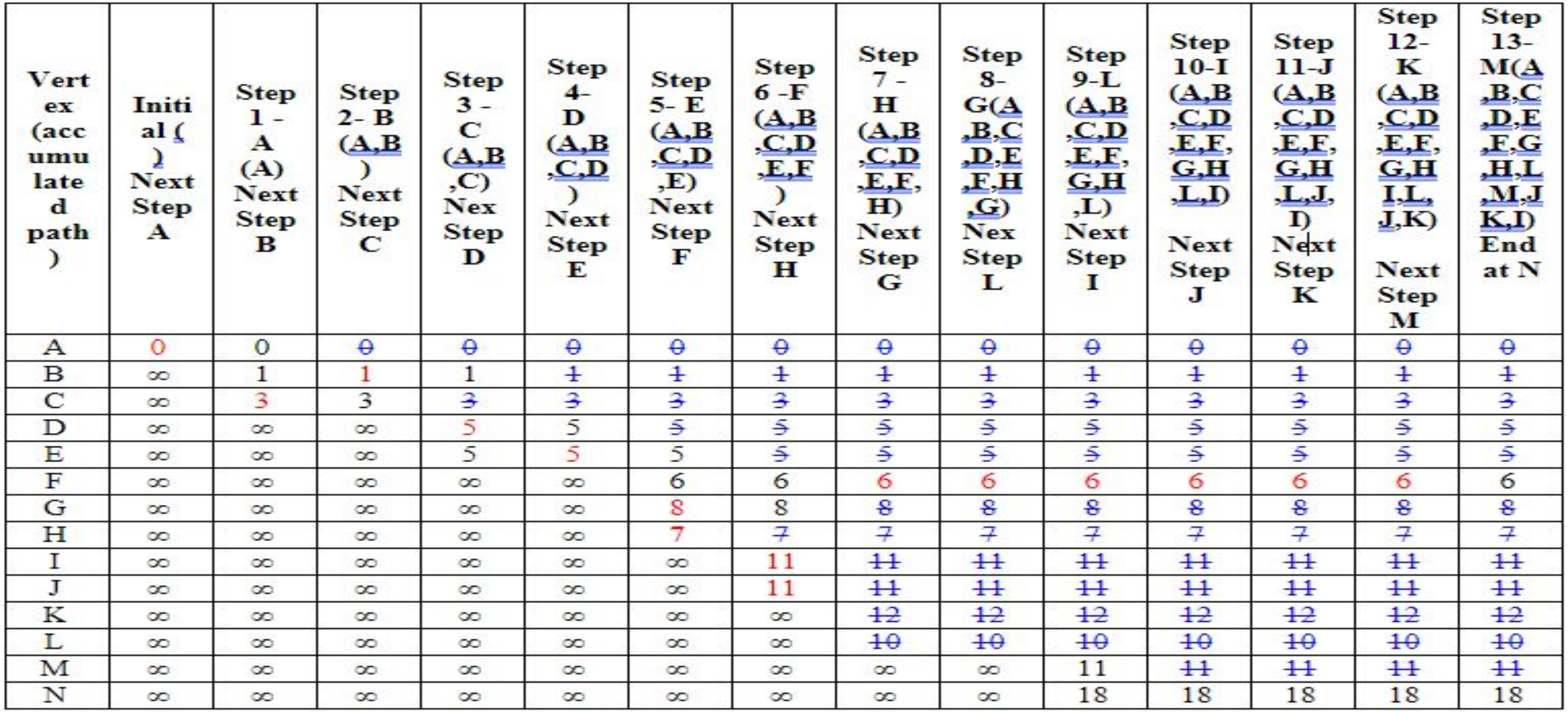


Find the Path and Distance using Tree

- This path is determined based on predecessor information.



Dijkstra’s - Minimum distance using Table



Vertex (accumulated path)	Initial ( ) Next Step A	Step 1 – A (A) Next Step B	Step 2- B (A,B) Next Step C	Step 3 – C (A,B,C) Nex Step D	Step 4- D (A,B,C,D) Next Step E	Step 5- E (A,B,C,D,E) Next Step F	Step 6 -F (A,B,C,D,E,F) Next Step H	Step 7 – H (A,B,C,D,E,F, H) Next Step G	Step 8- G(A,B,C,D,E,F,H,G) Nex Step L	Step 9-L (A,B,C,D,E,F, G,H,L) Next Step I	Step 10-I (A,B,C,D,E,F, G,H,L,I) Next Step J	Step 11-J (A,B,C,D,E,F, G,H,L,J, I) Next Step K	Step 12-K (A,B,C,D,E,F, G,H,I,L, J,K) Next Step M	Step 13-M(A,B,C,D,E,F,G,H,L,M,J,K,I) End at N
A	0	0	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~
B	∞	1	1	1	~~1~~	~~1~~	~~1~~	~~1~~	~~1~~	~~1~~	~~1~~	~~1~~	~~1~~	~~1~~
C	∞	3	3	~~3~~	~~3~~	~~3~~	~~3~~	~~3~~	~~3~~	~~3~~	~~3~~	~~3~~	~~3~~	~~3~~
D	∞	∞	∞	5	5	~~5~~	~~5~~	~~5~~	~~5~~	~~5~~	~~5~~	~~5~~	~~5~~	~~5~~
E	∞	∞	∞	5	5	5	~~5~~	~~5~~	~~5~~	~~5~~	~~5~~	~~5~~	~~5~~	~~5~~
F	∞	∞	∞	∞	∞	6	6	6	6	6	6	6	6	6
G	∞	∞	∞	∞	∞	8	8	~~8~~	~~8~~	~~8~~	~~8~~	~~8~~	~~8~~	~~8~~
H	∞	∞	∞	∞	∞	7	~~7~~	~~7~~	~~7~~	~~7~~	~~7~~	~~7~~	~~7~~	~~7~~
I	∞	∞	∞	∞	∞	∞	11	~~11~~	~~11~~	~~11~~	~~11~~	~~11~~	~~11~~	~~11~~
J	∞	∞	∞	∞	∞	∞	11	~~11~~	~~11~~	~~11~~	~~11~~	~~11~~	~~11~~	~~11~~
K	∞	∞	∞	∞	∞	∞	∞	~~12~~	~~12~~	~~12~~	~~12~~	~~12~~	~~12~~	~~12~~
L	∞	∞	∞	∞	∞	∞	∞	~~10~~	~~10~~	~~10~~	~~10~~	~~10~~	~~10~~	~~10~~
M	∞	∞	∞	∞	∞	∞	∞	∞	∞	11	~~11~~	~~11~~	~~11~~	~~11~~
N	∞	∞	∞	∞	∞	∞	∞	∞	∞	18	18	18	18	18

Dijkstra’s - Steps to follow Minimum Distance

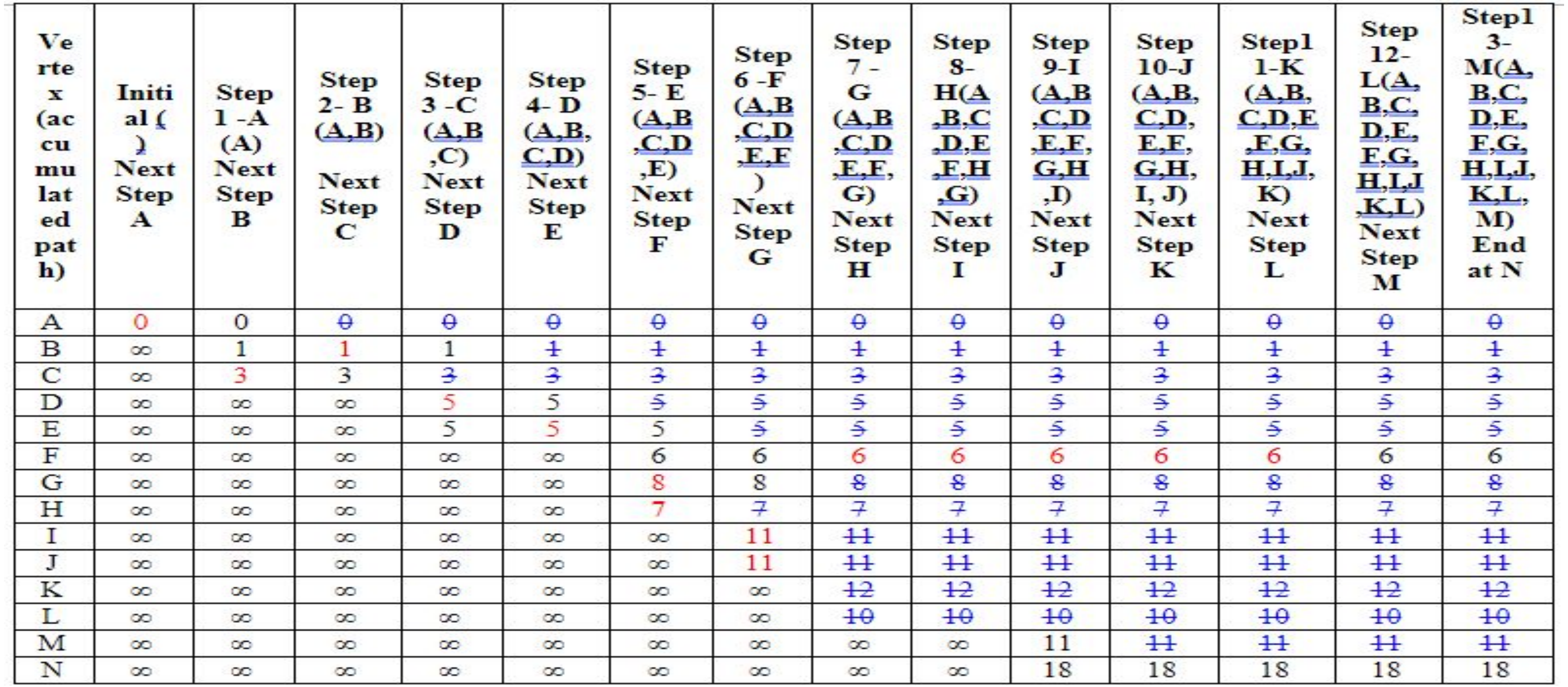
- Let us consider vertex **A** and **N** as the start and destination vertex respectively.
- Initially, all the vertices except the start vertex **A** are marked by ∞ and the start vertex **A** is marked by ***0***.
  - **Step X** - represents: the current visiting node
  - **Next Step** represents: the next node to visit
  - ~~V~~: this node has been visited

  - Initial
    - 0 is smallest cost on Initial step.
      - Thus, **A** is selected as the starting point for Step 1.

Steps - Continues

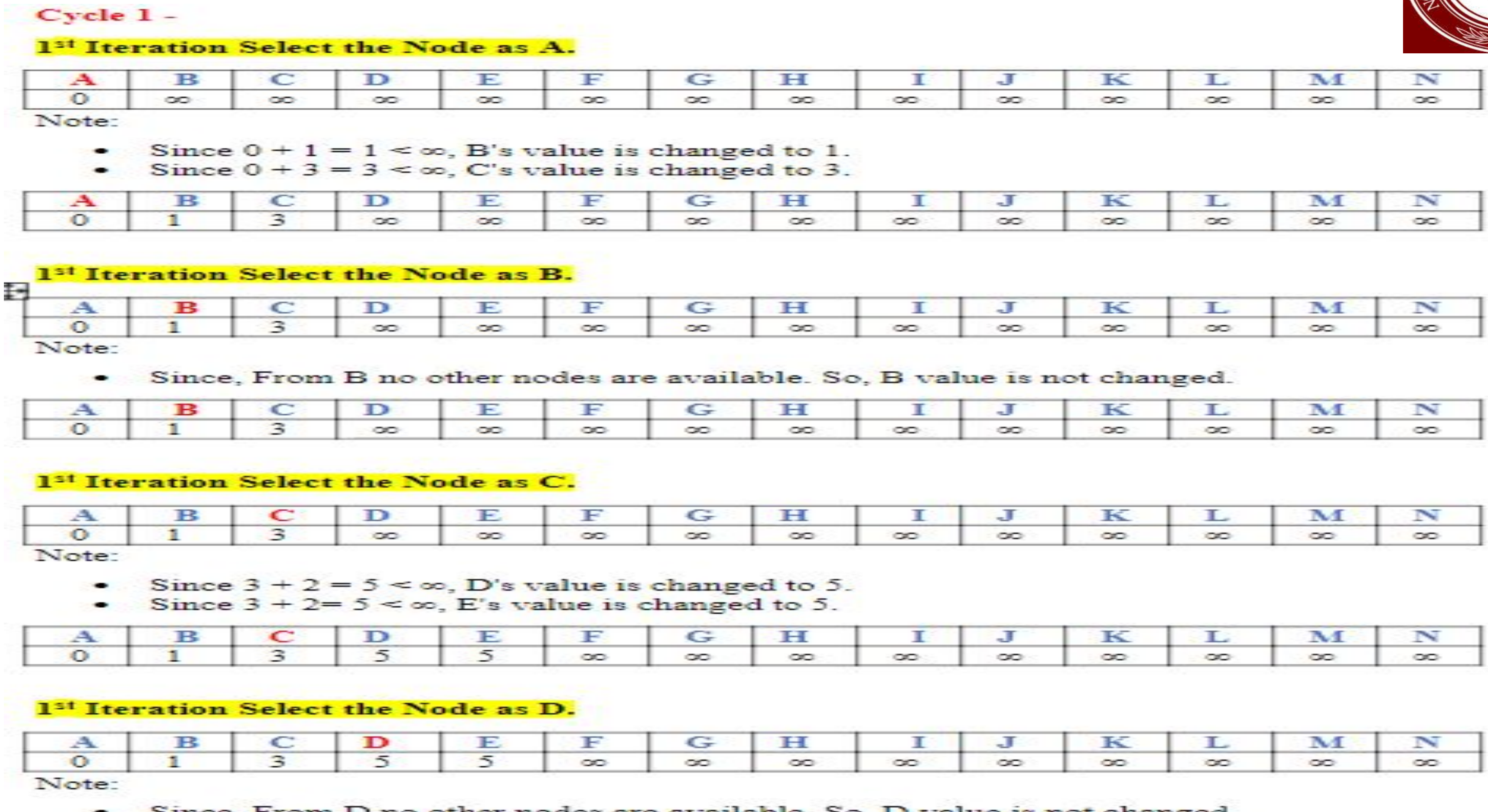
- Step 1
    - **A** is selected as the starting point for Step 1.
        - From **A**, one can go to **A or B or C**
            - The accumulated cost on **A** is not changed. It is still 0.
            - The accumulated cost on **B** is **1.**
            - The accumulated cost on **C** is **3.**
            - **1** is smaller than **3**.
                - Thus, **B** is selected as the starting point for Step 2.
    - Stop if the destination node **N** is reached
- Continue this process, you will find the minimum distance of **N** from ***A*** is **18**. And the path is **3→ 2→ 2→ 3→ 8**

BellMan Ford’s - Minimum distance using Table



Vertex (accumulated path)	Initial ( ) Next Step A	Step 1 -A (A) Next Step B	Step 2- B (A,B) Next Step C	Step 3 -C (A,B ,C) Next Step D	Step 4- D (A,B, C,D) Next Step E	Step 5- E (A,B ,C,D ,E) Next Step F	Step 6 -F (A,B ,C,D ,E,F) Next Step G	Step 7 - G (A,B ,C,D ,E,F, G) Next Step H	Step 8- H(A ,B,C ,D,E ,F,H ,G) Next Step I	Step 9-I (A,B ,C,D ,E,F, G,H ,I) Next Step J	Step 10-J (A,B, C,D, E,F, G,H, I, J) Next Step K	Step11-K (A,B, C,D,E ,F,G, H,I,J, K) Next Step L	Step 12-L(A, B,C, D,E, F,G, H,I,J ,K,L) Next Step M	Step13-M(A, B,C, D,E, F,G, H,I,J, K,L, M) End at N
A	0	0	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~
B	∞	1	1	1	~~1~~	~~1~~	~~1~~	~~1~~	~~1~~	~~1~~	~~1~~	~~1~~	~~1~~	~~1~~
C	∞	3	3	~~3~~	~~3~~	~~3~~	~~3~~	~~3~~	~~3~~	~~3~~	~~3~~	~~3~~	~~3~~	~~3~~
D	∞	∞	∞	5	5	~~5~~	~~5~~	~~5~~	~~5~~	~~5~~	~~5~~	~~5~~	~~5~~	~~5~~
E	∞	∞	∞	5	5	5	~~5~~	~~5~~	~~5~~	~~5~~	~~5~~	~~5~~	~~5~~	~~5~~
F	∞	∞	∞	∞	∞	6	6	6	6	6	6	6	6	6
G	∞	∞	∞	∞	∞	8	8	~~8~~	~~8~~	~~8~~	~~8~~	~~8~~	~~8~~	~~8~~
H	∞	∞	∞	∞	∞	7	~~7~~	~~7~~	~~7~~	~~7~~	~~7~~	~~7~~	~~7~~	~~7~~
I	∞	∞	∞	∞	∞	∞	11	~~11~~	~~11~~	~~11~~	~~11~~	~~11~~	~~11~~	~~11~~
J	∞	∞	∞	∞	∞	∞	11	~~11~~	~~11~~	~~11~~	~~11~~	~~11~~	~~11~~	~~11~~
K	∞	∞	∞	∞	∞	∞	∞	~~12~~	~~12~~	~~12~~	~~12~~	~~12~~	~~12~~	~~12~~
L	∞	∞	∞	∞	∞	∞	∞	~~10~~	~~10~~	~~10~~	~~10~~	~~10~~	~~10~~	~~10~~
M	∞	∞	∞	∞	∞	∞	∞	∞	∞	11	~~11~~	~~11~~	~~11~~	~~11~~
N	∞	∞	∞	∞	∞	∞	∞	∞	∞	18	18	18	18	18

Bellman-Ford Algorithm Steps to find Shortest Path -



Cycle 1 -

1st Iteration Select the Node as A.

A	B	C	D	E	F	G	H	I	J	K	L	M	N
0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞

Note:

- Since $0 + 1 = 1 < \infty$, B's value is changed to 1.
- Since $0 + 3 = 3 < \infty$, C's value is changed to 3.

A	B	C	D	E	F	G	H	I	J	K	L	M	N
0	1	3	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞

1st Iteration Select the Node as B.

A	B	C	D	E	F	G	H	I	J	K	L	M	N
0	1	3	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞

Note:

- Since, From B no other nodes are available. So, B value is not changed.

A	B	C	D	E	F	G	H	I	J	K	L	M	N
0	1	3	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞

1st Iteration Select the Node as C.

A	B	C	D	E	F	G	H	I	J	K	L	M	N
0	1	3	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞

Note:

- Since $3 + 2 = 5 < \infty$, D's value is changed to 5.
- Since $3 + 2= 5 < \infty$, E's value is changed to 5.

A	B	C	D	E	F	G	H	I	J	K	L	M	N
0	1	3	5	5	∞	∞	∞	∞	∞	∞	∞	∞	∞

1st Iteration Select the Node as D.

A	B	C	D	E	F	G	H	I	J	K	L	M	N
0	1	3	5	5	∞	∞	∞	∞	∞	∞	∞	∞	∞

Note:

- Since, From D no other nodes are available. So, D value is not changed.

1st Iteration Select the Node as E.

A	B	C	D	E	F	G	H	I	J	K	L	M	N
0	1	3	5	5	∞	∞	∞	∞	∞	∞	∞	∞	∞

Note:

- Since $5 + 1 = 6 < \infty$, F's value is changed to 6.
- Since $5 + 3 = 8 < \infty$, G's value is changed to 8.
- Since $5 + 2 = 7 < \infty$, H's value is changed to 7.

A	B	C	D	E	F	G	H	I	J	K	L	M	N
0	1	3	5	5	6	8	7	∞	∞	∞	∞	∞	∞

1st Iteration Select the Node as F.

A	B	C	D	E	F	G	H	I	J	K	L	M	N
0	1	3	5	5	6	8	7	∞	∞	∞	∞	∞	∞

Note:

- Since $6 + 5 = 11 < \infty$, I's value is changed to 11.
- Since $6 + 5 =11 < \infty$, J's value is changed to 11.

A	B	C	D	E	F	G	H	I	J	K	L	M	N
0	1	3	5	5	6	8	7	11	11	∞	∞	∞	∞

1st Iteration Select the Node as G.

A	B	C	D	E	F	G	H	I	J	K	L	M	N
0	1	3	5	5	6	8	7	11	11	∞	∞	∞	∞

Note:

- Since, From G no other nodes are available. So, G value is not changed.

1st Iteration Select the Node as H.

A	B	C	D	E	F	G	H	I	J	K	L	M	N
0	1	3	5	5	6	8	7	11	11	∞	∞	∞	∞

Note:

- Since $7 + 5 = 12 < \infty$, K's value is changed to 12.
- Since $7 + 3 =10 < \infty$, L's value is changed to 10.

A	B	C	D	E	F	G	H	I	J	K	L	M	N
0	1	3	5	5	6	8	7	11	11	12	10	∞	∞

1st Iteration Select the Node as I.

A	B	C	D	E	F	G	H	I	J	K	L	M	N
0	1	3	5	5	6	8	7	11	11	12	10	∞	∞

Note:

- Since, From I no other nodes are available. So, I value is not changed.

1st Iteration Select the Node as J.

A	B	C	D	E	F	G	H	I	J	K	L	M	N
0	1	3	5	5	6	8	7	11	11	12	10	∞	∞

Note:

- Since, From J no other nodes are available. So, J value is not changed.

1st Iteration Select the Node as K.

A	B	C	D	E	F	G	H	I	J	K	L	M	N
0	1	3	5	5	6	8	7	11	11	12	10	∞	∞

Note:

- Since, From K no other nodes are available. So, K value is not changed.

1st Iteration Select the Node as L.

A	B	C	D	E	F	G	H	I	J	K	L	M	N
0	1	3	5	5	6	8	7	11	11	12	10	∞	∞

Note:

- Since $10 + 1 = 11 < \infty$, M's value is changed to 11.
- Since $10 + 8 = 18 < \infty$, N's(Nothing but Final Node) value is changed to 18.

A	B	C	D	E	F	G	H	I	J	K	L	M	N
0	1	3	5	5	6	8	7	11	11	12	10	11	18

1st Iteration Select the Node as M

A	B	C	D	E	F	G	H	I	J	K	L	M	N
0	1	3	5	5	6	8	7	11	11	12	10	11	18

Note:

- Since, From M no other nodes are available. So, M value is not changed.

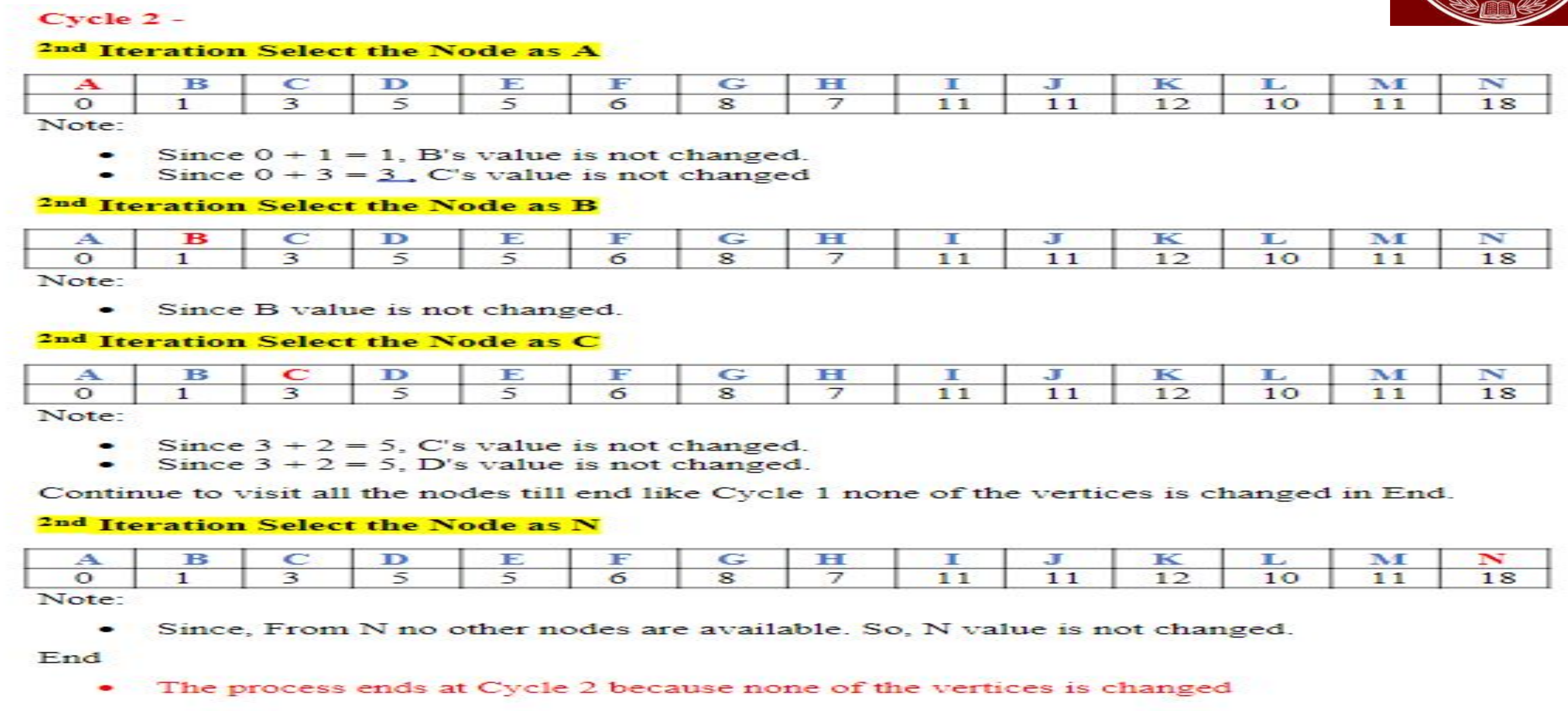
1st Iteration Select the Node as N

A	B	C	D	E	F	G	H	I	J	K	L	M	N
0	1	3	5	5	6	8	7	11	11	12	10	11	18

Note:

- Since, From N no other nodes are available. So, N value is not changed.

- Hence, the minimum distance between vertex **A** and vertex **N** is **18**.Based on t predecessor information, the path is **A→ C→ E→ H→ L→ N.**



Cycle 2 -

2nd Iteration Select the Node as A

A	B	C	D	E	F	G	H	I	J	K	L	M	N
0	1	3	5	5	6	8	7	11	11	12	10	11	18

Note:

- Since 0 + 1 = 1, B's value is not changed.
- Since 0 + 3 = 3 , C's value is not changed

2nd Iteration Select the Node as B

A	B	C	D	E	F	G	H	I	J	K	L	M	N
0	1	3	5	5	6	8	7	11	11	12	10	11	18

Note:

- Since B value is not changed.

2nd Iteration Select the Node as C

A	B	C	D	E	F	G	H	I	J	K	L	M	N
0	1	3	5	5	6	8	7	11	11	12	10	11	18

Note:

- Since 3 + 2 = 5, C's value is not changed.
- Since 3 + 2 = 5, D's value is not changed.

Continue to visit all the nodes till end like Cycle 1 none of the vertices is changed in End.

2nd Iteration Select the Node as N

A	B	C	D	E	F	G	H	I	J	K	L	M	N
0	1	3	5	5	6	8	7	11	11	12	10	11	18

Note:

- Since, From N no other nodes are available. So, N value is not changed.

End

- The process ends at Cycle 2 because none of the vertices is changed

Bellman Ford’s steps to find a Shortest Path

- In the Bellman-Ford’s algorithm, we begin by initializing all the distances of all nodes with infinity, except for the source node, which is initialized with zero. Next, we perform
  V (number of nodes) **- 1** Steps **==> (14-1)** Steps **= 13 steps in each cycle**
- The above process ends at **Cycle 2** because none of the vertices is changed.So, Total Steps to complete Maze in 2 Cycles - Each Cycle(13 Steps) * 2 = **26 Steps**

Dijkstra steps to find a Shortest Path

- Dijkstra, is most likely found multiple other cheapest paths between our starting node and other nodes in the graph. However, if we wanted to know the shortest path between our starting node and all other nodes we would need to keep running the algorithm on all nodes that aren't visited yet. In the worst case scenario we'd need to run the algorithm ***NumberOfNodes - 1* times.**
- ***NumberOfNodes - 1* times => 14 - 1 => 13 Times/Steps.**

Dijkstra's algorithm - Big O

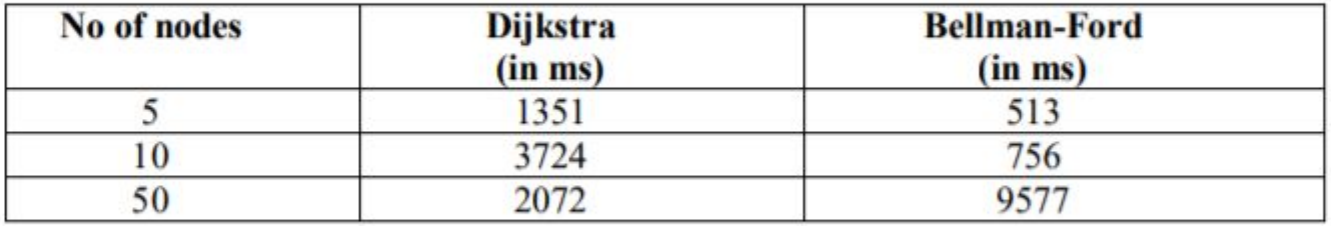
- The bottleneck of Dijkstra's algorithm is finding the next closest, unvisited node/vertex. Using LinkedList this has a complexity of *O(numberOfEdges)*, since in the worst case scenario we need to go through all the edges of the node to find the one with the smallest weight.
- If we didn't use PriorityQueue (like LinkedList) - The complexity would be =>O(V+E) *E => ***O(V² + E)***.
- This gives us a total of **O(V+E)log V)** complexity when using PriorityQueue Where V and E are the number of vertices and edges respectively.

Bellman Ford's Algorithm - Big O

- The Bellman–Ford algorithm simply relaxes *all* the edges, and does this |V|-1 times, where |V| is the number of vertices in the graph. In each of these repetitions, the number of vertices with correctly calculated distances grows, from which it follows that eventually all vertices will have their correct distances. This method allows the Bellman–Ford algorithm to be applied to a wider class of inputs than Dijkstra.
- Bellman–Ford runs in **O(|V|.|E|)** time, where |V| and |E| are the number of vertices and edges respectively.

Simulation of the two algorithms

- Bellman-Ford algorithm, when the number of nodes is small, the running time is better than Dijkstra algorithm. However, Dijkstra algorithm has better running time when the number of nodes is larger. For example, Dijkstra is most effective for a massive variety of nodes. In higher number of nodes, the Dijkstra’s algorithm is better and efficient. Also,Dijkstra’s algorithm emerged to produce shorter time in both small and big graphs.



No of nodes	Dijkstra (in ms)	Bellman-Ford (in ms)
5	1351	513
10	3724	756
50	2072	9577

CONCLUSION

- The comparative analysis in term of shortest path optimization between two algorithms. The two algorithms are compared which are Dijkstra and Bellman-Ford algorithms to conclude which of them is more efficient for finding the shortest path between two vertices. Our results show that the Dijkstra algorithm is much faster than the algorithm of the Bellman ford and commonly used in real-time applications.

CONCLUSION AND FUTURE WORK

- Maze solving problem used different algorithms to solve this problem. Using a good algorithm can get a high performance led to find a best shortest route in short time. The proposed technique based on the Flood fill algorithm works better and with less searching time. Using camera in this work avoids the algorithm step by executing them in the computer using image processing technique rather than using them by a robot.

References:

- Maze

  https://npu85.npu.edu/~henry/npu/classes/algorithm/graph_alg/slide/maze.html

- Shortest Path

  https://npu85.npu.edu/~henry/npu/classes/algorithm/tutorialpoints_daa/slide/shortest_paths.html

- Google Slides URL -

  https://drive.google.com/file/d/1yBcaI4g7Qdah4pj-vX3GxH9LUXQ2bfqI/view?usp=sharing