Course: CS455 - Algorithms & Structured Programming
Faculty: Prof. Henry Chang

Comparison of Algorithms - Maze Project

Signature Project
Name: Haripriya A
ID: 19579

NORTHWESTERN POLYTECHNIC
UNIVERSITY

Table of Contents

Shortest Path

- The shortest path is about finding path between two vertices in a graph that is from source to the target destination vertex, So be the sum of all the edges weight is minimum or short compared to the complete graph.

- The path can be found using various algorithms but we are going to compare two algorithms which are as follows.
    - Dijkstra's Algorithm
    - Bellman-Ford Algorithm

Introduction to Dijkstra’s Algorithm

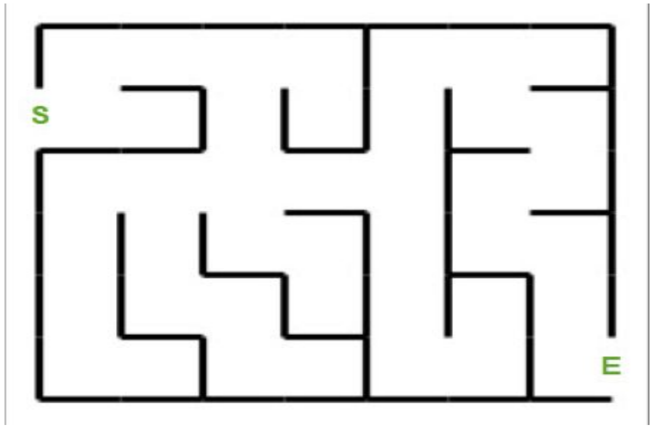
*Dijkstra’s algorithm solves the single-source shortest-paths problem on a directed weighted graph where all the nodes need to be non-negative.

*With the given maze we are going to implementing a tree of shortest paths from source vertex to the target vertex and find the shortest path of the following maze using Dijkstra’s algorithm.

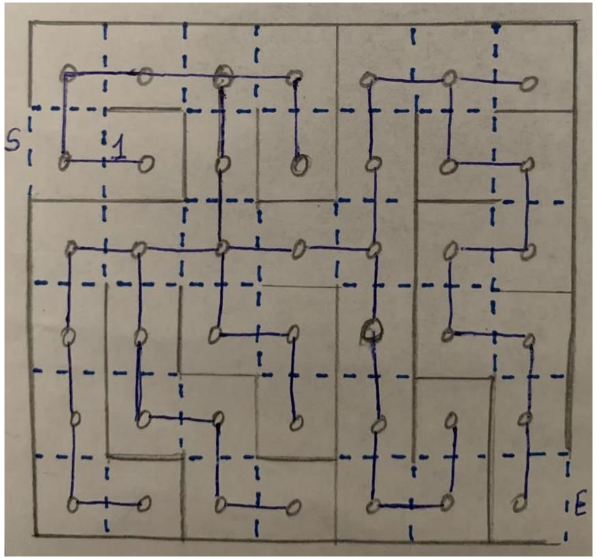
Steps for Implementing a tree from given Maze

A. Below are the detailed steps to implement a tree from source vertex S to the end E

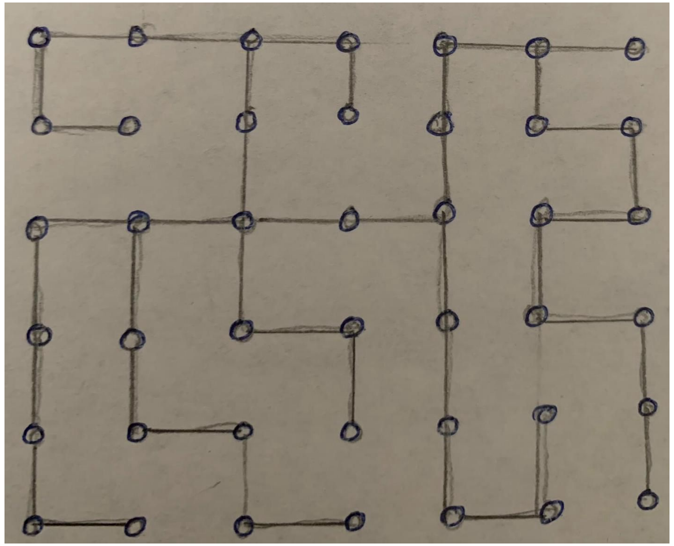
STEP - 1: Starting with the Initial Node S



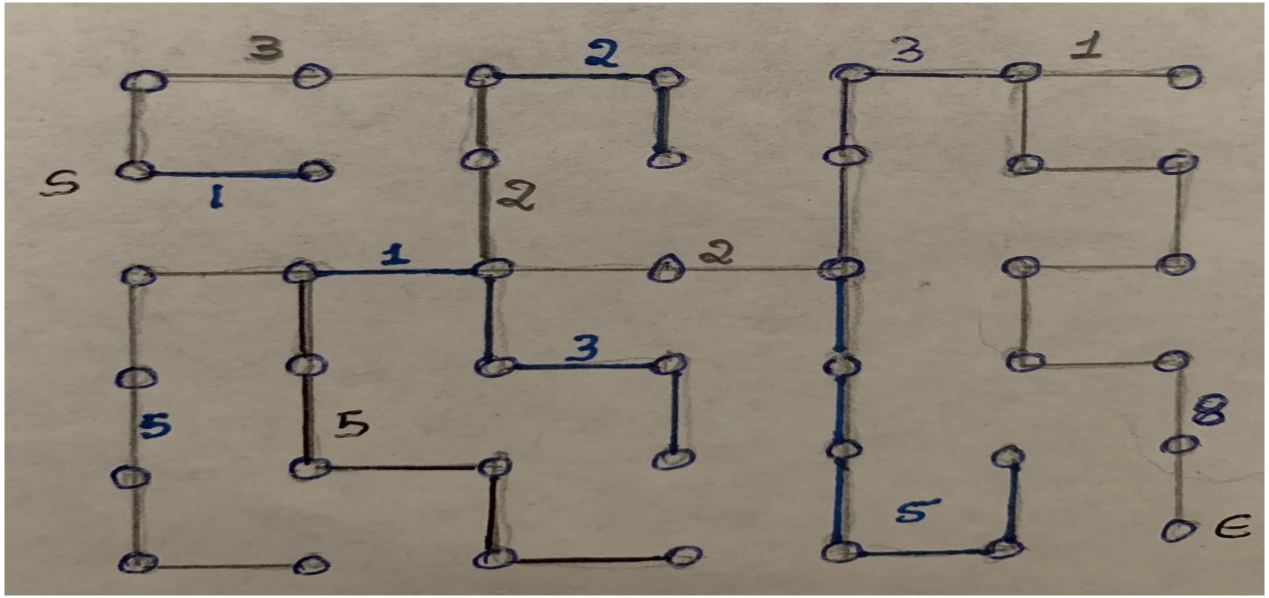
STEP - 2 : Draw the dotted lines and place the nodes in each box and draw the path from each node starting from S



STEP - 3 : Take out the maze and draw the nodes and path.

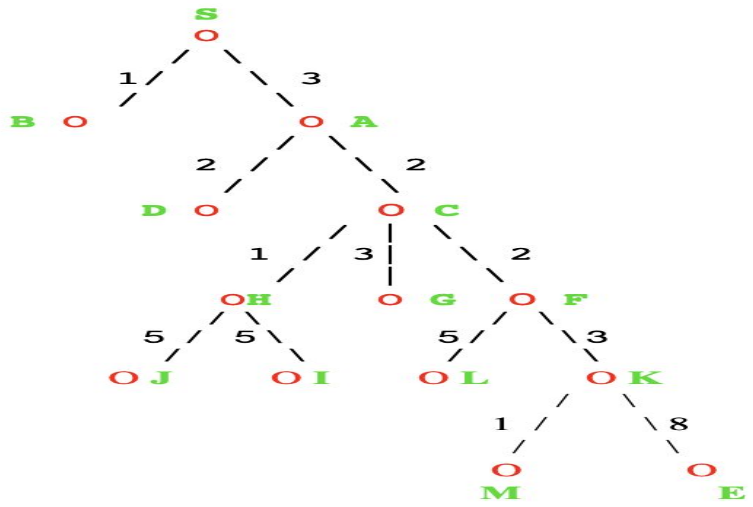


STEP - 4 : Identify the nodes end and find the distance to draw a tree



NORTHWESTERN POLYTECHNIC UNIVERSITY

STEP - 5: Implement tree with vertices and distance between the each vertices.



B. Below are the detailed steps to find the shortest path from source vertex S to the target vertex E by developing a table

Here we need to identify the shortest path between two vertices from source vertex which is shortest distance

Let G = (V, E) and vertices or nodes denoted as v or u. Here an edge is denoted as (u,v), and the distance or weight is denoted as w(u,v)

G=(V, E) be a non-negative edges (i.e., $w(u, v) \geq 0$ for each edge $(u, v) \in E$)

By using the function Extract-Min() we extract the node with the smallest key.

Algorithm: Dijkstra's-Algorithm (G, w, s)

```
for each vertex v Є G.V
   v.d := ∞
   v.∏ := NIL
s.d := 0
S := Φ
Q := G.V
while Q ≠ Φ
   u := Extract-Min (Q)
   S := S U {u}
   for each vertex v Є G.adj[u]
      if v.d > u.d + w(u, v)
         v.d := u.d + w(u, v)
         v.∏ := u
```

The complexity of this algorithm is fully dependent on the implementation of Extract-Min function. If extract min function is implemented using linear search, The complexity of this algorithm is $\boldsymbol{O(V^2 + E)}$.

For developing a table let us consider vertex S and E as the start and target destination vertex respectively.

Initially, all the vertices except the start vertex S are marked by ∞ and the start vertex S is marked by 0.

STEPS to implement table and find the shortest path

We are going to determine the path based on predecessor information.
Initial : 0 is smallest cost on Initial step. Thus, S is selected as the starting point for Step 1.

Step-1: S is selected as the starting point for Step 1.

- From S, one can go to B or A
  - The accumulated cost on S is not changed. It is still 0.
  - The accumulated cost on B is 1.
  - The accumulated cost on A is 3.
  - 1 is smaller than 3.
    - Thus, B is selected as the starting point for Step 2.

Step-2: B is selected as the starting point for Step 2.

- From B, one can go to A as the B is already visited and does not have any further extended path
  - The accumulated cost on S is not changed. It is still 0.
  - The accumulated cost on B is 1.
  - The accumulated cost on A is 3.
  - 3 is smaller as B is already visited and remaining nodes are infinity.
    - Thus, A is selected as the starting point for Step 3.

Step-3: A is selected as the starting point for Step 3.

- From A, one can go to D or C
  - The accumulated cost of D is 5 (S => A => D).
  - The accumulated cost on C is 5(S => A => C).
  - Here both D and C are 5. We can select any of the node and we are taking D
    - Thus, D is selected as the starting point for Step 4.

Step-4: D is selected as the starting point for Step 4.

- From D, cannot go to any other node as none is point away from D so selected C as in the table only C is small and all other are visited nodes.
    - The accumulated cost on C is 5(S => A => C).
    - Here C is the next smallest having 5.
        - Thus, C is selected as the starting point for Step 5.

Step-5: C is selected as the starting point for Step 5.

- From C, one can go to H or G or F
    - The accumulated cost of H is 6(S => A => C => H).
    - The accumulated cost on G is 8(S => A => C => G).
    - The accumulated cost on F is 7(S => A => C => F).
    - Here H is the smallest having 6 among G and F.
        - Thus, H is selected as the starting point for Step 6.

Step-6: H is selected as the starting point for Step 6.

- From H, one can go to I or J
    - The accumulated cost of I is 11(S => A => C => H => I).
    - The accumulated cost on J is 11(S => A => C => H => J).
    - The accumulated cost on G is 8(S => A => C => G).
    - The accumulated cost on F is 7(S => A => C => F).
    - We have F as the next smallest node in the table having 7.
        - Thus, F is selected as the starting point for Step 7.

Step-7: F is selected as the starting point for Step 7.

- From F, one can go to L or K
  - The accumulated cost of I is 11(S => A => C => H => I).
  - The accumulated cost on J is 11(S => A => C => H => J).
  - The accumulated cost on G is 8(S => A => C => G).
  - The accumulated cost on L is 12(S => A => C => F => L).
  - The accumulated cost on K is 10(S => A => C => F => K)
  - We have G as the next smallest node in the table having 8.
    i. Thus, G is selected as the starting point for Step 8.

Step-8: G is selected as the starting point for Step 8.

- From G, one cannot go to any other node as none is point away from G so selected K as in the table only K is small among all unvisited nodes.
  - The accumulated cost of I is 11(S => A => C => H => I).
  - The accumulated cost on J is 11(S => A => C => H => J).
  - The accumulated cost on L is 12(S => A => C => F => L).
  - The accumulated cost on K is 10(S => A => C => F => K)
  - We have K as the next smallest node in the table having 10.
    - Thus, K is selected as the starting point for Step 9.

Step-9: K is selected as the starting point for Step 9.

- From K, one can go to M or E.
  - The accumulated cost of I is 11(S => A => C => H => I).
  - The accumulated cost on J is 11(S => A => C => H => J).
  - The accumulated cost on L is 12(S => A => C => F => L).
  - The accumulated cost on M is 11(S => A => C => F => K => M)
  - The accumulated cost on E is 18(S => A => C => F => K => E)
  - We have M, I, J as the next smallest node in the table having 11. We can select any of the node and we are taking M.
    - Thus, M is selected as the starting point for Step 10.

Step-10: M is selected as the starting point for Step 10.

- From M, one cannot go to any other node as none is point away from M, we can select I or J, as in the table I and J are small among all unvisited nodes but alphabetically we are selecting I.
  - The accumulated cost of I is 11(S => A => C => H => I).
  - The accumulated cost on J is 11(S => A => C => H => J).
  - The accumulated cost on L is 12(S => A => C => F => L).
  - The accumulated cost on E is 18(S => A => C => F => K => E)
  - We have I, J as the next smallest nodes in the table having 11. We can select any of the node and we are taking I.
    - Thus, I is selected as the starting point for Step 11.

Step-11: I is selected as the starting point for Step 11.

- From M, one cannot go to any other node as none is point away from I so selected J, as in the table only J is small among all unvisited nodes.
  - The accumulated cost on J is 11(S => A => C => H => J).
  - The accumulated cost on L is 12(S => A => C => F => L).
  - The accumulated cost on E is 18(S => A => C => F => K => E)
  - J is the next smallest nodes in the table having 11. We can select J.
    - Thus, J is selected as the starting point for Step 12.

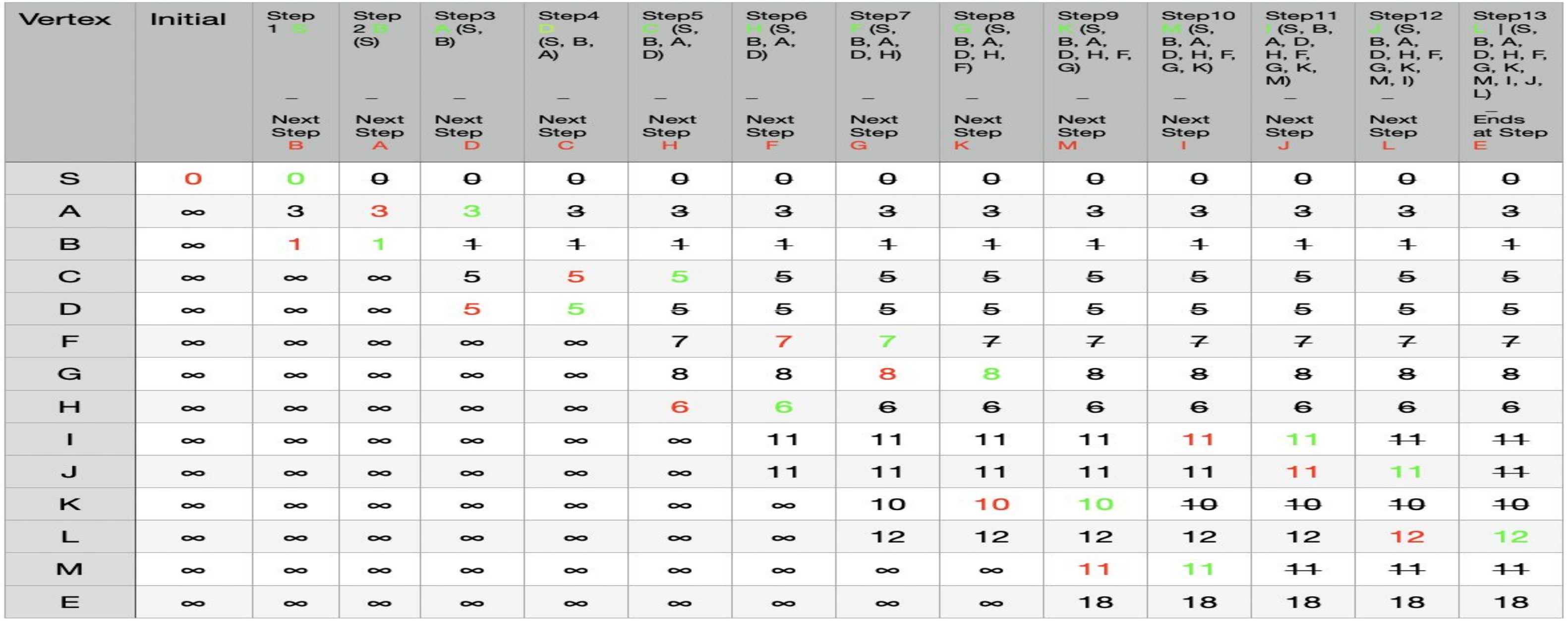
Step-12: J is selected as the starting point for Step 12.

- From J, one cannot go to any other node as none is point away from J so selected L, as in the table only L is small among all unvisited nodes.
  - The accumulated cost on L is 12(S => A => C => F => L).
  - The accumulated cost on E is 18(S => A => C => F => K => E)
  - L is the next smallest nodes in the table having 12. We can select L.
    - Thus, L is selected as the starting point for Step 13.

Step-13: L is selected as the starting point for Step 13.

- From L, one cannot go to any other node as none is point away from L so selected E, as in the table onlv E is left.
  - The accumulated cost on E is 18(S => A => C => F => K => E).

Implemented table to find shortest path



Vertex	Initial	Step 1 S — Next Step B	Step 2 B (S) — Next Step A	Step3 A (S, B) — Next Step D	Step4 D (S, B, A) — Next Step C	Step5 C (S, B, A, D) — Next Step H	Step6 H (S, B, A, D) — Next Step F	Step7 F (S, B, A, D, H) — Next Step G	Step8 G (S, B, A, D, H, F) — Next Step K	Step9 K (S, B, A, D, H, F, G) — Next Step M	Step10 M (S, B, A, D, H, F, G, K) — Next Step I	Step11 I (S, B, A, D, H, F, G, K, M) — Next Step J	Step12 J (S, B, A, D, H, F, G, K, M, I) — Next Step L	Step13 L | (S, B, A, D, H, F, G, K, M, I, J, L) — Ends at Step E
S	0	0	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~
A	∞	3	3	3	~~3~~	~~3~~	~~3~~	~~3~~	~~3~~	~~3~~	~~3~~	~~3~~	~~3~~	~~3~~
B	∞	1	1	~~1~~	~~1~~	~~1~~	~~1~~	~~1~~	~~1~~	~~1~~	~~1~~	~~1~~	~~1~~	~~1~~
C	∞	∞	∞	5	5	5	~~5~~	~~5~~	~~5~~	~~5~~	~~5~~	~~5~~	~~5~~	~~5~~
D	∞	∞	∞	5	5	~~5~~	~~5~~	~~5~~	~~5~~	~~5~~	~~5~~	~~5~~	~~5~~	~~5~~
F	∞	∞	∞	∞	∞	7	7	7	~~7~~	~~7~~	~~7~~	~~7~~	~~7~~	~~7~~
G	∞	∞	∞	∞	∞	8	8	8	8	~~8~~	~~8~~	~~8~~	~~8~~	~~8~~
H	∞	∞	∞	∞	∞	6	6	~~6~~	~~6~~	~~6~~	~~6~~	~~6~~	~~6~~	~~6~~
I	∞	∞	∞	∞	∞	∞	11	11	11	11	11	11	~~11~~	~~11~~
J	∞	∞	∞	∞	∞	∞	11	11	11	11	11	11	11	~~11~~
K	∞	∞	∞	∞	∞	∞	∞	10	10	10	~~10~~	~~10~~	~~10~~	~~10~~
L	∞	∞	∞	∞	∞	∞	∞	12	12	12	12	12	12	12
M	∞	∞	∞	∞	∞	∞	∞	∞	∞	11	11	~~11~~	~~11~~	~~11~~
E	∞	∞	∞	∞	∞	∞	∞	∞	∞	18	18	18	18	18

- V: the current visiting node
- V: the next node to visit
- ~~V~~: this node has been visited

Shortest Path - Dijkstra’s Algorithm

Hence, the minimum distance from S to E in the maze is 18
*The path is S $\rightarrow$ A$\rightarrow$ C$\rightarrow$ F$\rightarrow$ K$\rightarrow$ E
* Distance is 3 $\rightarrow$ 2 $\rightarrow$ 2 $\rightarrow$ 3 $\rightarrow$ 8

Note : The shortest path can be found by either Dijkstra’s Algorithm or Bellman Ford Algorithm.

- Dijkstra’s algorithm only works for the edges which are non-negative whereas Bellman Ford Algorithm works for both negative and non-negative edges.

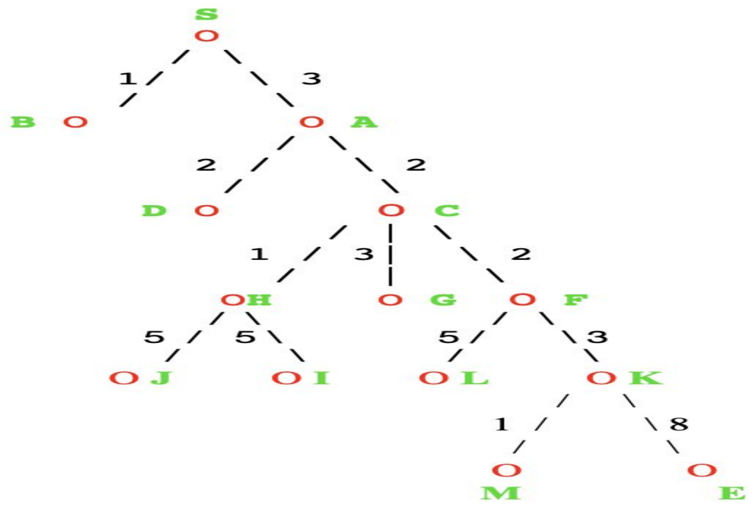
Introduction to Bellman-Ford Algorithm

- The Bellman Ford Algorithm is another way of finding the shortest path. Bellman Ford Algorithm is also a single source shortest path algorithm which can detect and work on the negative edge weight cycles in a graph.
- The Bellman Ford algorithm relaxes all the edges in a |V| - 1 times, V is the number of vertices in the given graph.
- Eventually, In every cycle all the vertices with calculated distances increase correctly

Algorithm: Bellman-Ford-Algorithm (G, w, s)

```
for each vertex v Є G.V
  v.d := ∞
  v.∏ := NIL
s.d := 0
for i = 1 to |G.V| - 1
  for each edge (u, v) Є G.E
    if v.d > u.d + w(u, v)
      v.d := u.d +w(u, v)
      v.∏ := u
for each edge (u, v) Є G.E
  if v.d > u.d + w(u, v)
    return FALSE
return TRUE
```

Get Started with the below tree to find shortest path



Steps for Implementing the shortest Path

Cycle 1:

- **S:**

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

Note:

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

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

- **A :**

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

Note:

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

- **C:**

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

Note:

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

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

- **D:**

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

- **F:**

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

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

- **H:**

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

Note:

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

- **I:**

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

- **J:**

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

- **K:**

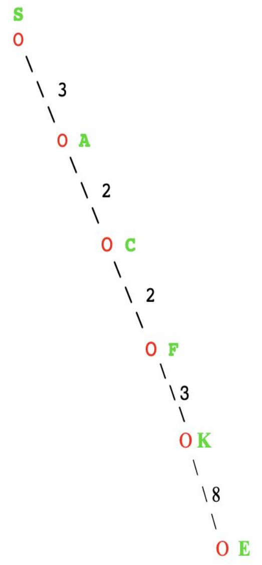
0	3	1	5	5	7	8	6	11	11	10	12	11	18
S	A	B	C	D	F	G	H	I	J	K	L	M	E

- **M:**

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

- **E:**

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



Cycle 2:

- **S:**

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

Note:

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

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

- **A :**

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

Note:

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

- **C:**

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

Note:

- ▪ Since 5 + 2 = 7, F's value is not changed.
- ▪ Since 5 + 3 = 8, G's value is not changed.
- ▪ Since 5 + 1 = 6, H's value is not changed .

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

- **D:**

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

- **F:**

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

- ▪ Since 7 + 3 = 10 < ∞, K's value is not changed.
- ▪ Since 7 + 5 = 12 < ∞, L's value is not changed.

- **H:**

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

Note:

- ■ Since 6 + 5 = 11, I's value is not changed.
- ■ Since 6 + 5 = 11, J's value is not changed.

- **I:**

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

- **J:**

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

- **K:**

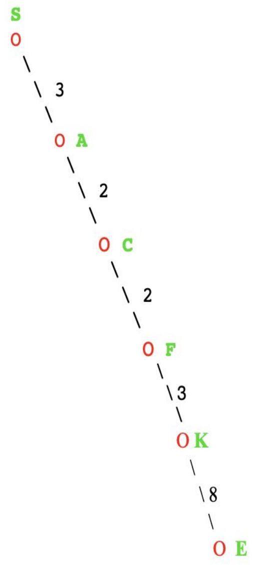
0	3	1	5	5	7	8	6	11	11	10	12	11	18
S	A	B	C	D	F	G	H	I	J	K	L	M	E

- **M:**

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

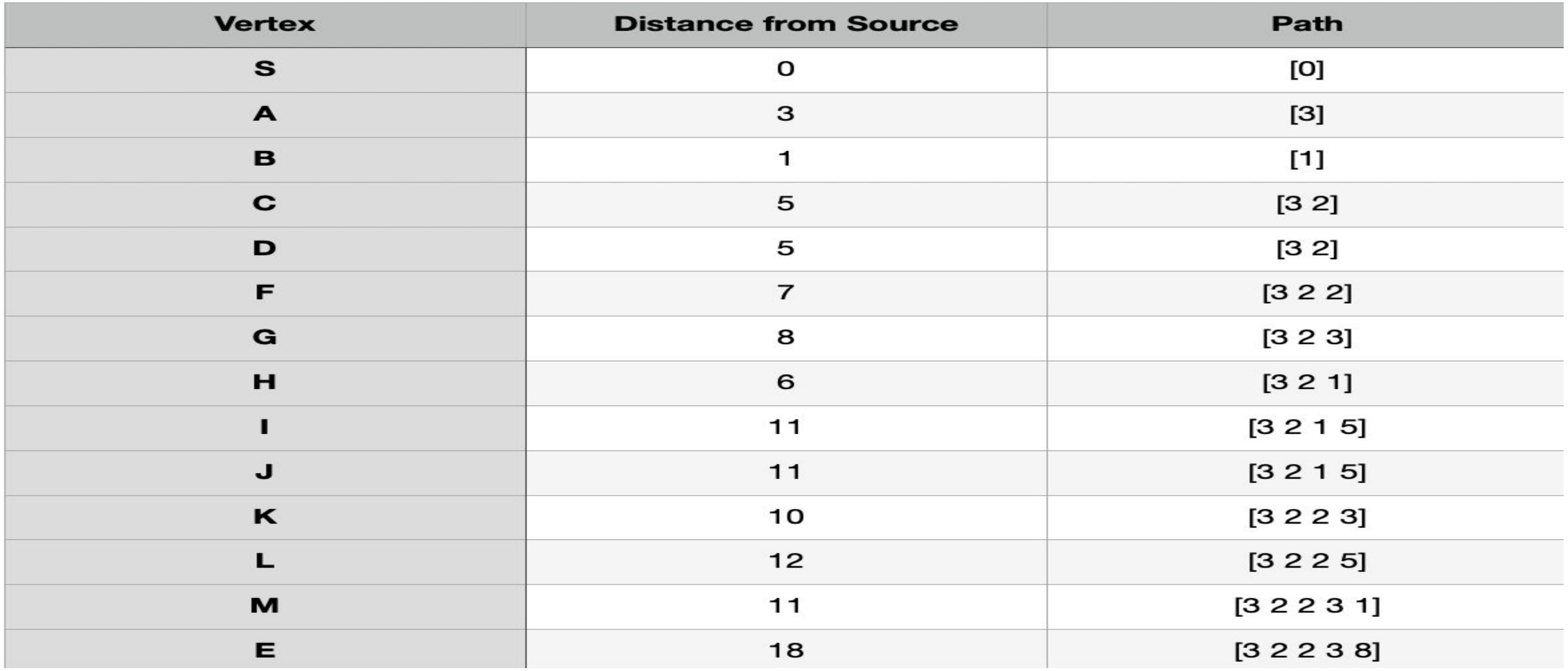
- **E:**

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



- The second cycle is also repeated with no changes after the end target destination E as we do not have any negative distances in the cycle.
- After second cycle none of the edges are relaxed and the algorithm ends at cycle 2

- Table of shortest path from source



Vertex	Distance from Source	Path
S	0	[0]
A	3	[3]
B	1	[1]
C	5	[3 2]
D	5	[3 2]
F	7	[3 2 2]
G	8	[3 2 3]
H	6	[3 2 1]
I	11	[3 2 1 5]
J	11	[3 2 1 5]
K	10	[3 2 2 3]
L	12	[3 2 2 5]
M	11	[3 2 2 3 1]
E	18	[3 2 2 3 8]

Shortest Distance of Bellman-Ford Algorithm

Hence, the minimum distance between vertex **S** and target vertex **E** is **18**.

Based on the predecessor information, the path is S → A→ C→ F→ K→ E

The first for loop is used for initialization, which runs in O(V) times. The next for loop runs |V - 1| passes over the edges, which takes O(E) times.

The time complexity of the algorithm Bellman-Ford is O(V*E), where V is the total number of vertices and E is the total number of edges in the graph, respectively.

Comparison Between Dijkstra’s Bellman-Ford algorithms

- The Dijkstra’s algorithm greedily selects the minimum weight node which is not visited and do the relaxation on the edge outgoing from source whereas, Bellman-Ford algorithm will relax edges in |V| - 1 times by eventually calculating the distance of all vertices.The Bellman-Ford algorithm is applied in wider inputs when compared to Dijkstra’s as it can read the negative cycles and method of calculating the distance.
- The Bellman-Ford algorithm works well with negative edges and is capable of detecting a negative edge cycle in a graph. However, for Dijkstra algorithm, it works well with positive weights in a directed or undirected graph.
- The Dijkstra’s algorithm considered better in the absence of the negative cycles.
- Bellman-Ford algorithm works good if the number of nodes in the given graph are small, the running time is better than Dijkstra algorithm. Though, Dijkstra algorithm has better running time when the number of given nodes in the graph are larger.
- The time complexity of Dijkstra’s is $\boldsymbol{O(V^2 + E)}$ where for Bellman-Ford it is $\boldsymbol{O(V*E)}$
- Finally, Dijkstra algorithm is faster and more efficient than Bellman-Ford with all positive cycles.

Spanning Tree

- A spanning tree is a subset of an undirected Graph that has all the vertices connected by minimum number of edges.
- The weight of a spanning tree is the sum of weights given to each edge of the spanning tree.
- A Minimum Spanning Tree (MST) is a subset of edges of a connected weighted undirected graph that connects all the vertices together with the minimum possible total edge weight.
- If there are n number of vertices, the spanning tree should have n - 1 number of edges.
- If there exist any duplicate weighted edges, the graph may have multiple minimum spanning tree.
- A minimum spanning tree has (V – 1) edges where V is the number of vertices in the given graph.
- To find Minimum Spanning Tree we can use below two algorithms
    - Prim's Minimum Spanning Tree (MST)
    - Kruskal's Minimum Spanning Tree Algorithm

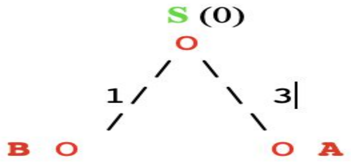
Prim’s Minimum Spanning Tree (MST)

- Create a set mstSet that keeps track of vertices already included in MST.
- Assign a key value to all vertices in the input graph
    - Initialize all key values as INFINITE.
    - Assign key value as 0 for the first vertex so that it is picked first.
- While mstSet doesn’t include all vertices
- Pick a vertex u which is not there in mstSet and has minimum key value.
- Include u to mstSet.
- Update key value of all of u's adjacent vertices which are not in mstSet. For every adjacent vertex v which are not in mstSet, if weight of edge u-v is less than the previous key value of v, update v's key value as weight of u-v.

Steps for Implementing the Minimum Spanning Tree

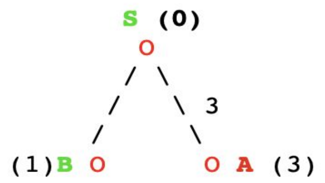
Step -1 : The set mstSet is initially empty and keys assigned to vertices are {0, INF, INF, INF, INF, INF, INF, INF} where INF indicates infinite.

a. Now pick the vertex with minimum key value. The vertex S is picked, include it in mstSet. So mstSet becomes {S}.
b. After including to mstSet, update key values of adjacent vertices. Adjacent vertices of S are A and B.
c. The key values of A and B are updated as 3 and 1.
d. Following subgraph shows vertices and their key values, only the vertices with finite key values are shown. The vertices included in MST are shown in green color.



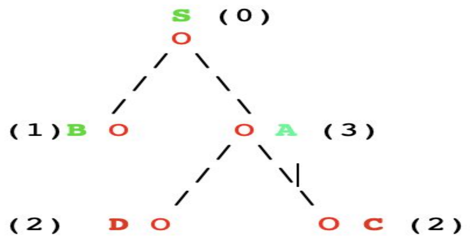
Step -2 : Pick the vertex with minimum key value and not already included in MST (i.e., not in mstSET).

a. The vertex B is picked and added to mstSet.
b. So mstSet now becomes {S, B}.
c. Update the key values of adjacent vertices of B is not present. So from S we pick A



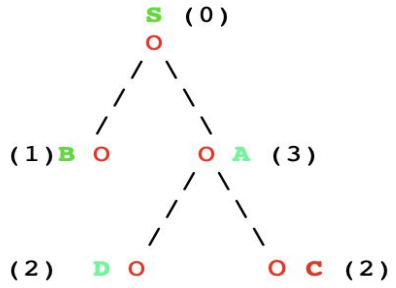
Step -3 : Pick the vertex with minimum key value and not already included in MST (i.e., not in mstSET).

a. The vertex A is picked and added to mstSet.
b. So mstSet now becomes {S, B, A}.
c. Update the key values of adjacent vertices of A. The key value of vertex D and C becomes finite(2 and 2 respectively).



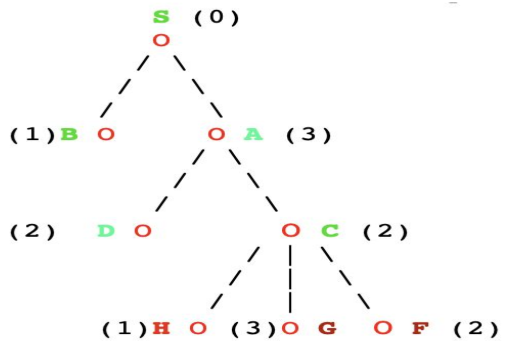
Step -4 : Pick the vertex with minimum key value and not already included in MST (i.e., not in mstSET).

a. The vertex D is picked and added to mstSet.
b. So mst Set now becomes {S, B, A, D}.
c. Update the key values of adjacent vertices of D is not present.



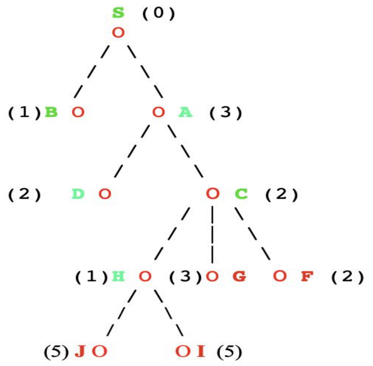
Step -5 : Pick the vertex with minimum key value and not already included in MST (i.e., not in mstSET).

a. The vertex C is picked and added to mstSet.
b. So mst Set now becomes {S, B, A, D, C}.
c. Update the key values of adjacent vertices of C.
d. The key value of vertex H, G and F becomes finite(1, 3 and 2 respectively).



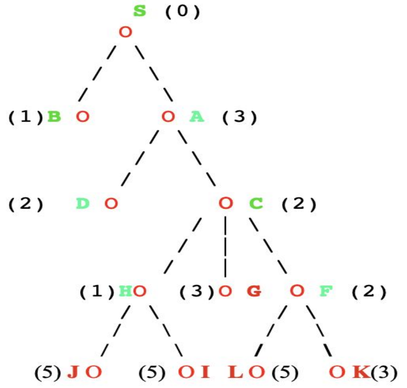
Step -6 : Pick the vertex with minimum key value and not already included in MST (i.e., not in mstSET).

a. The vertex H is picked and added to mstSet.
b. So mst Set now becomes {S, B, A, D, C, H}.
c. Update the key values of adjacent vertices of H.
d. The key value of vertex J and I becomes finite(5 and 5 respectively).



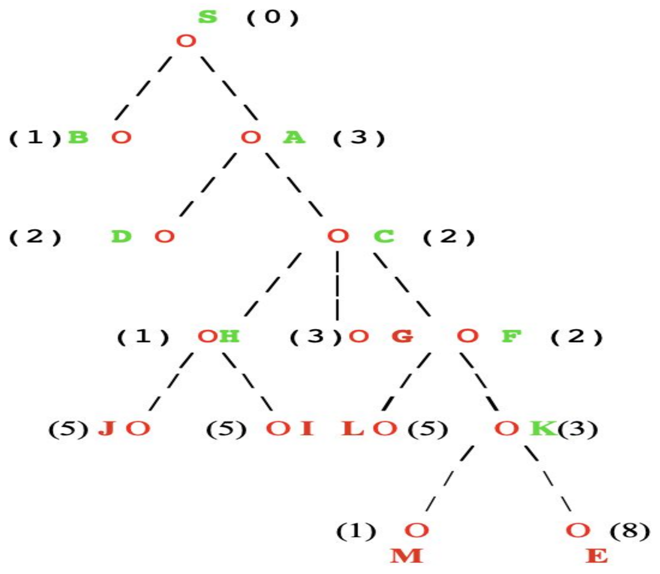
Step -7 : Pick the vertex with minimum key value and not already included in MST (i.e., not in mstSET).

a. The vertex F is picked and added to mstSet.
b. So mst Set now becomes {S, B, A, D, C, H, F}.
c. Update the key values of adjacent vertices of F.
d. The key value of vertex L and K becomes finite(3 and 5 respectively).



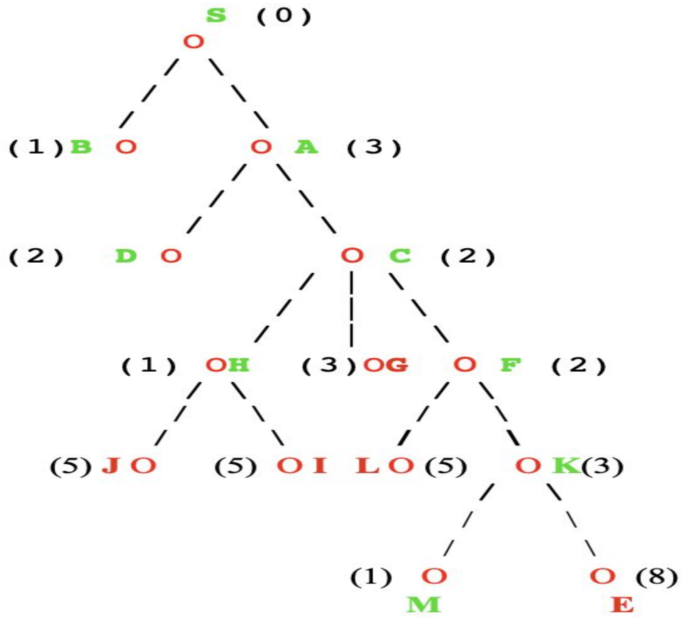
Step -8 : Pick the vertex with minimum key value and not already included in MST (i.e., not in mstSET).

a. The vertex K is picked and added to mstSet.
b. So mst Set now becomes {S, B, A, D, C, H, F, K}.
c. Update the key values of adjacent vertices of K. The key value of vertex L and K becomes finite(3 and 5 respectively).



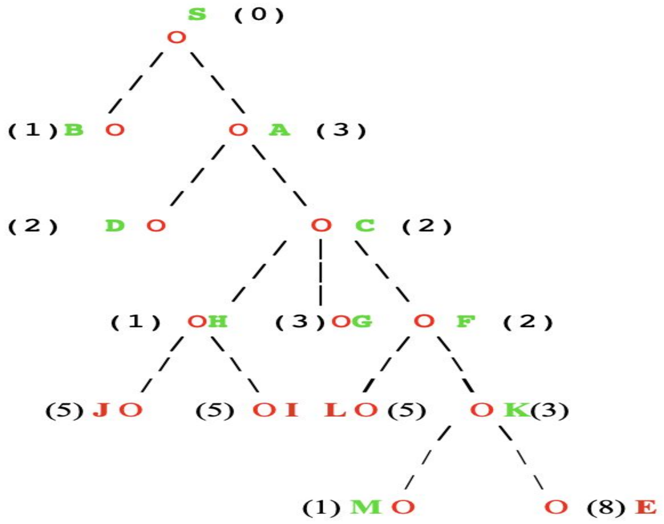
Step -9 : Pick the vertex with minimum key value and not already included in MST (i.e., not in mstSET).

a. The vertex M is picked and added to mstSet.
b. So mst Set now becomes {S, B, A, D, C, H, F, K, M}.
c. Update the key values of adjacent vertices of M and no edge is passing through M so we select the next unvisited vertex which is G.



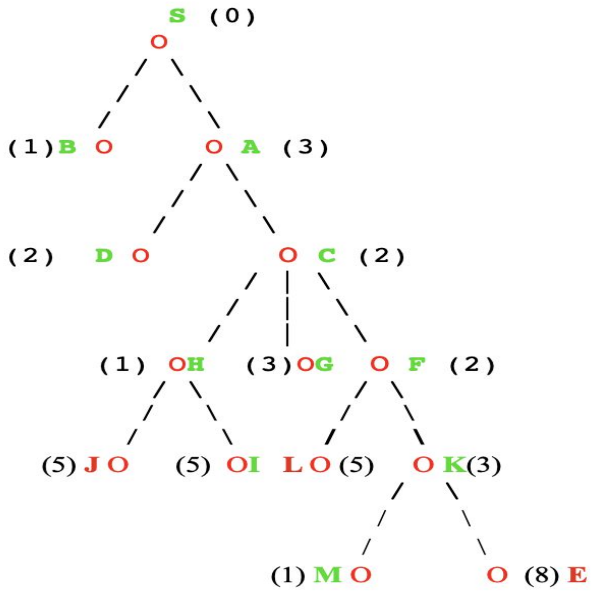
Step -10 : Pick the vertex with minimum key value and not already included in MST (i.e., not in mstSET).

a. The vertex G is picked and added to mstSet.
b. So mst Set now becomes {S, B, A, D, C, H, F, K, M, G}.
c. Update the key values of adjacent vertices of G and no edge is passing through M so we select the unvisited vertex which is I.
d. We repeat the above steps until mstSet includes all vertices of given graph. Finally, we get the following graph.



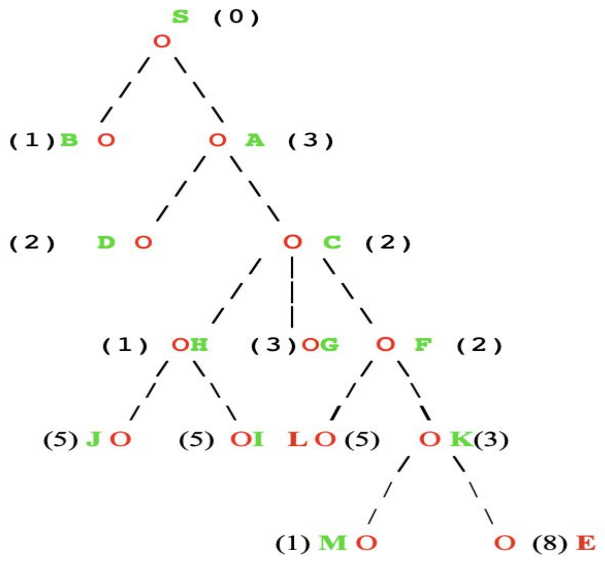
Step -11 : Pick the vertex with minimum key value and not already included in MST (i.e., not in mstSET).

a. The vertex I is picked and added to mstSet.
b. So mst Set now becomes {S, B, A, D, C, H, F, K, M, G, I}.
c. Update the key values of adjacent vertices of I and no edge is passing through I so we select the next unvisited vertex which is J.



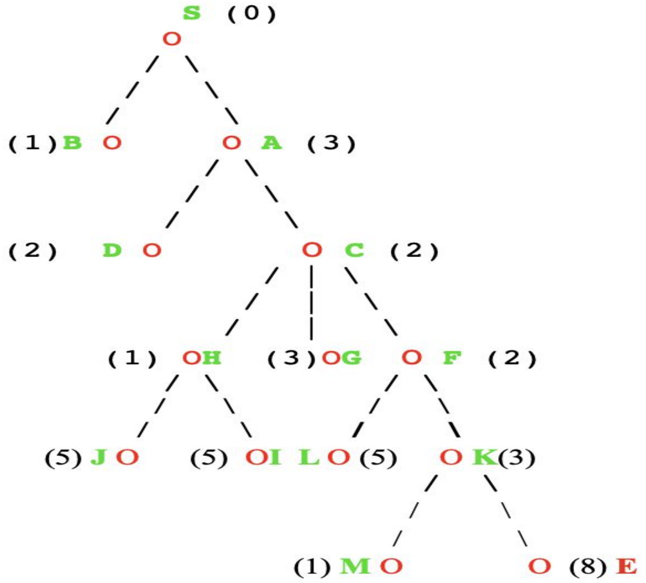
Step -12 : Pick the vertex with minimum key value and not already included in MST (i.e., not in mstSET).

a. The vertex J is picked and added to mstSet.
b. So mst Set now becomes {S, B, A, D, C, H, F, K, M, G, I, J}.
c. Update the key values of adjacent vertices of J and no edge is passing through J so we select the next unvisited vertex which is L.



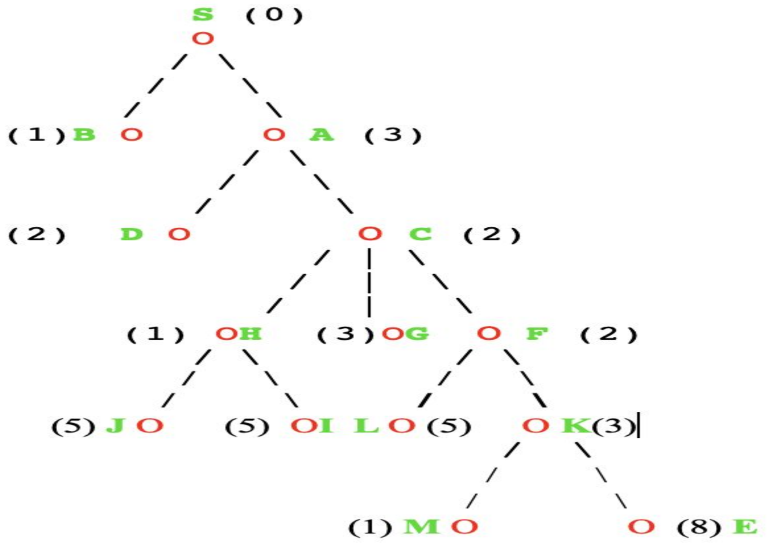
Step -13 : Pick the vertex with minimum key value and not already included in MST (i.e., not in mstSET).

a. The vertex L is picked and added to mstSet.
b. So mst Set now becomes {S, B, A, D, C, H, F, K, M, G, I, J, L}.
c. Update the key values of adjacent vertices of L and no edge is passing through L so we select the next unvisited vertex which is E.



Step -14 : Pick the vertex with minimum key value and not already included in MST (i.e., not in mstSET).

a. The vertex L is picked and added to mstSet.
b. So mst Set now becomes {S, B, A, D, C, H, F, K, M, G, I, J, L}.
c. Update the key values of adjacent vertices of L and no edge is passing through L so we select the next unvisited vertex which is E.



MST - Prim's Minimum Spanning Tree

- We have all the nodes visited and the process is completed.
- The MST set is {S, B, A, D, C, H, F, K, M, G, I, J, L}.
- The cost of the minimum spanning tree is (0+1+3+2+2+1+2+3+1+5+5+5+8) = 41, There is no more spanning tree in this graph with cost less than **41**.
- Time Complexity of Prim's algorithm is O((v + E)logV)

Kruskal’s Minimum Spanning Tree (MST)

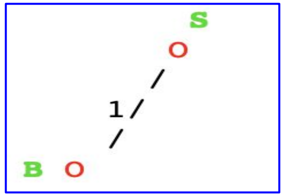
- Sort all the edges in non-decreasing order of their weight.
- Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.
- Repeat step#2 until there are (V-1) edges in the spanning tree.
- The graph contains 14 vertices and 13 edges. So, the minimum spanning tree formed will be having (14 – 1) = 13 edges.
- The algorithm is a Greedy Algorithm. The Greedy Choice is to pick the smallest weight edge that does not cause a cycle in the MST constructed so far. Let us understand it with an example: Consider the below input graph.

- After sorting:

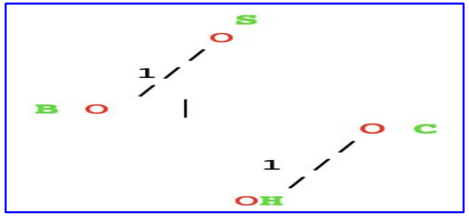
Weight	Source	Destination
1	S	B
1	C	H
1	K	M
2	A	C
2	A	D
2	C	F
3	S	A
3	C	G
3	F	K
5	H	I
5	H	J
5	F	L
8	K	E

- Now pick all edges one by one from sorted list of edges

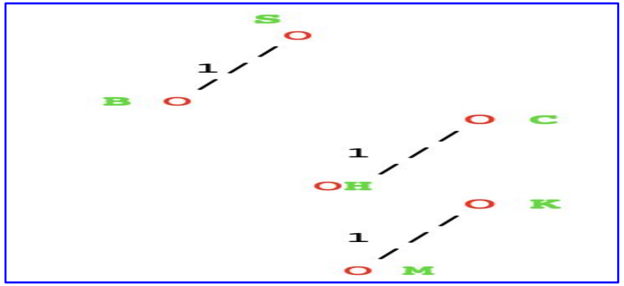
1. Pick edge S-B: No cycle is formed, include it.



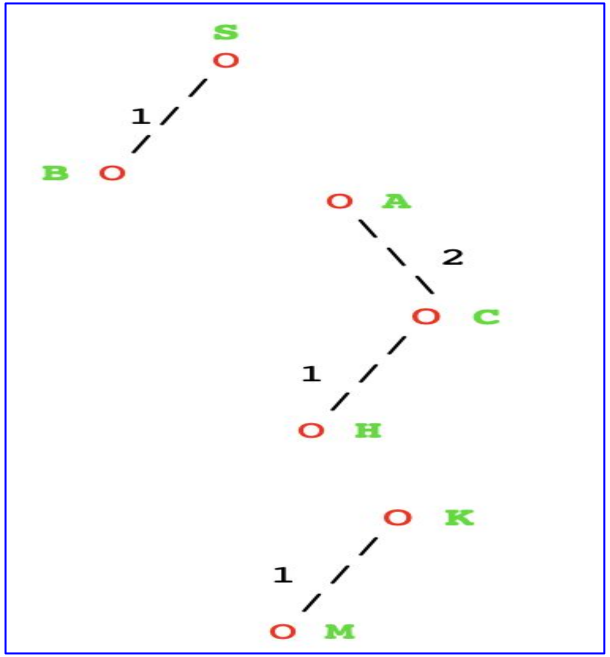
2. Pick edge C-H: No cycle is formed, include it.



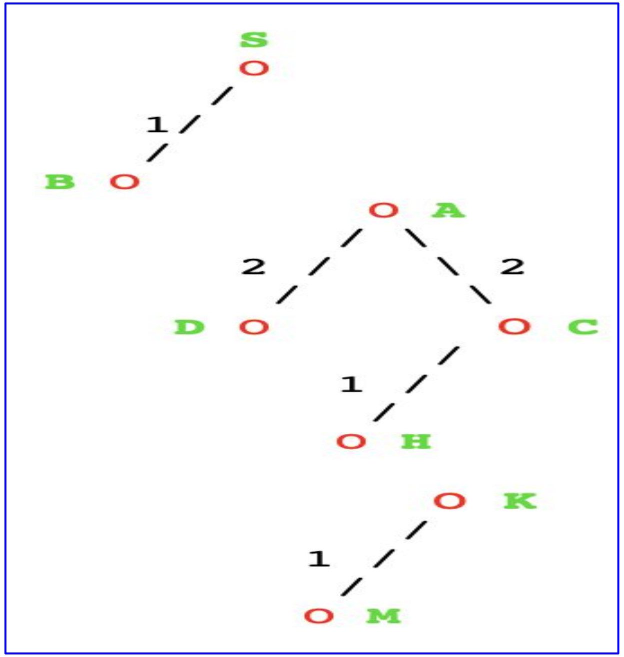
3. Pick edge K-M: No cycle is formed, include it.



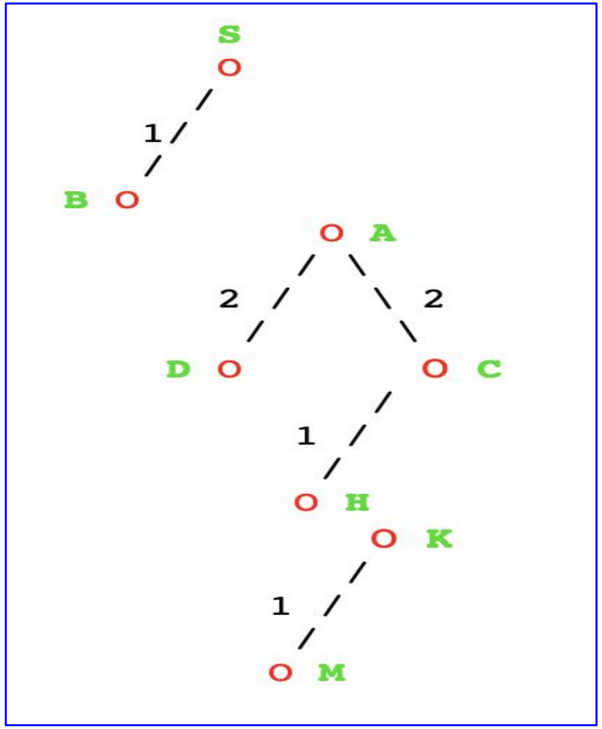
4. *Pick edge* A-C*:* No cycle is formed, include it.



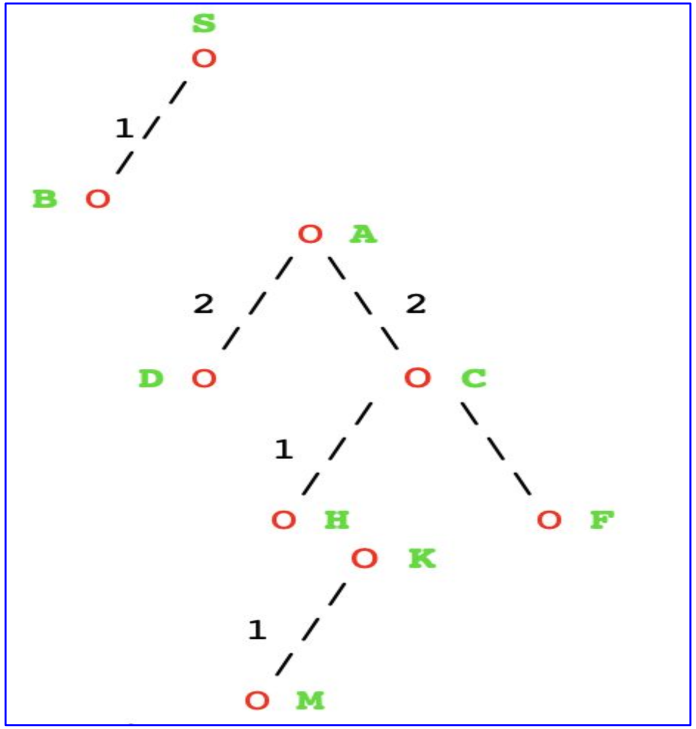
5. *Pick edge* A-D*:* No cycle is formed, include it.



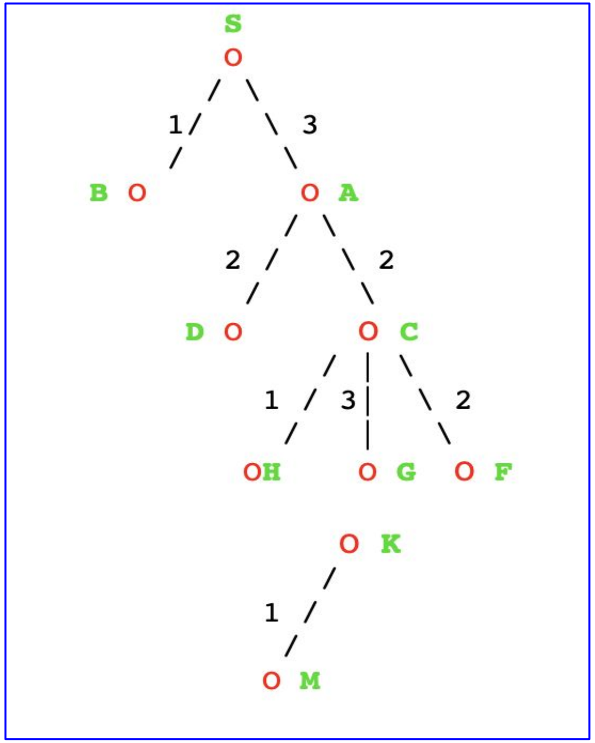
5. *Pick edge* A-D*:* No cycle is formed, include it.



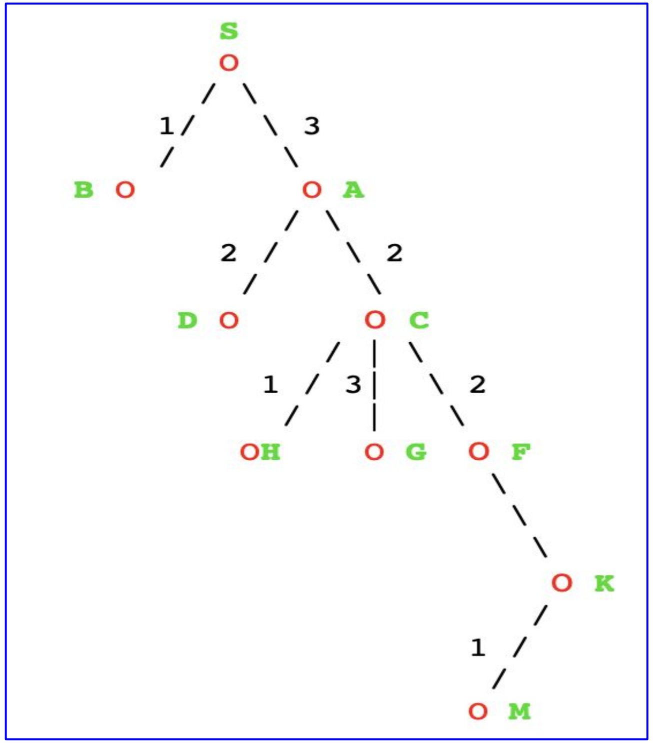
6. *Pick edge* C-F*:* No cycle is formed, include it.



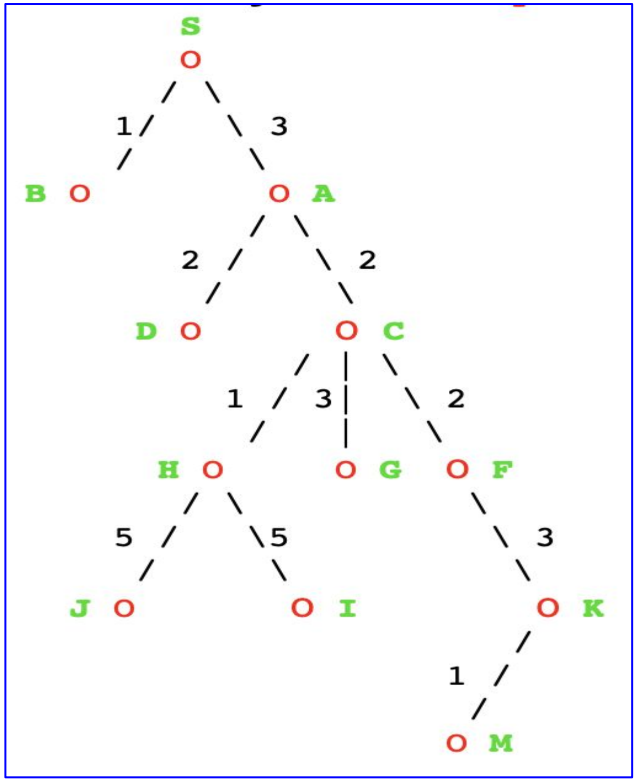
7. *Pick edge* S-A*:* No cycle is formed, include it.



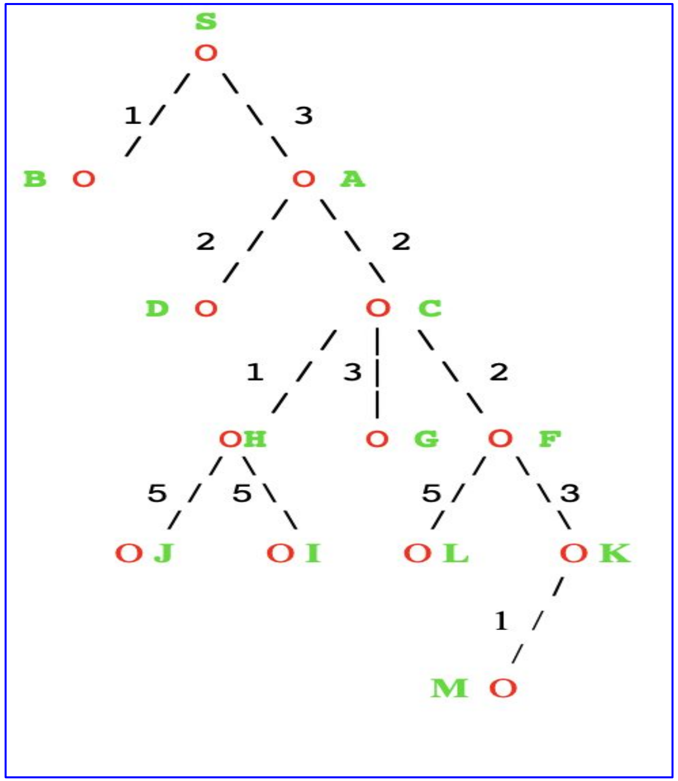
8**.** *Pick edge* F-K*:* No cycle is formed, include it.



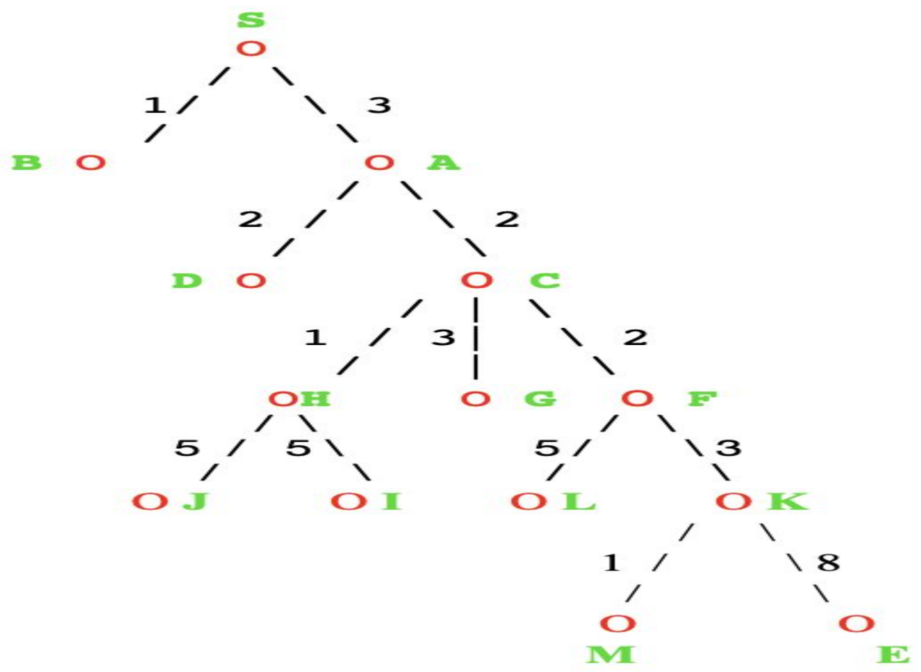
9. *Pick edge* H-J*:* No cycle is formed, include it.



10. *Pick edge* F-L*:* No cycle is formed, include it.



11. *Pick edge* K-E*:* No cycle is formed, include it.



MST - Kruskal’s Minimum Spanning Tree

- Since the number of edges equals to (V – 1), the algorithm stops here.
- The graph contains 14 vertices and 13 edges. So, the minimum spanning tree formed will be having (14 – 1) = 13 edges.
- Sorting all edges by their weights takes 13 edges.
- The time Complexity is O(E * logV)

Comparison between Prim’s and Kruskal’s Minimum Spanning Tree

- Prim's algorithm is significantly faster in the limit for dense graph with more edges than vertices. Kruskal performs better for simpler graphs.
- Time Complexity of Prim’s algorithm is O((v + E)logV) and The time complexity of Kruskal’s algorithm is O(E * logV).
- Prim’s algorithm gives connected component as well as it works only on connected graph whereas Kruskal’s work on disconnected components

Conclusion

1. Shortest Path
    - For finding the Maze shortest path the space complexity of Dijkstra algorithm is O(v^2) and Bellman-Ford is O(v^2)
    - For finding the Maze shortest path the time complexity of Dijkstra algorithm is $O(V^2 + E)$ and ellman-Ford is O(V*E)
    - The Maze shortest path can be found using Dijkstra algorithm more efficiently than Bellman-Ford as we do not have any negative edges.

2. Spanning Tree
    - Kruskal’s algorithm will work faster when the nodes already sorted. For the given Maze sorting is required to be performed before the output.
    - In finding the Maze shortest Time Complexity of Prim’s algorithm is O((v + E)logV) as priority queue take logarithmic time and The time complexity of Kruskal’s algorithm is O(E * logV).
    - The prim’s algorithm is complex and works faster on the give Maze but as we have many nodes which are not having the next passing edge from them it is difficult to choose the next node in such cases the Kruskal’s algorithm is working more efficiently.

References

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

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

Google slides URL :

https://docs.google.com/presentation/d/1JbDMnSiinAii8PDmttBEmT9bl0mP7SUp18YGTKBCWYw/edit?usp=sharing