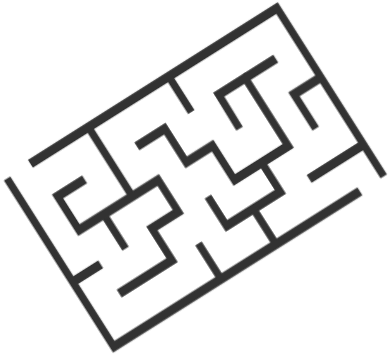
Methods to Find The Shortest Path of The Maze



Shoumya Singh

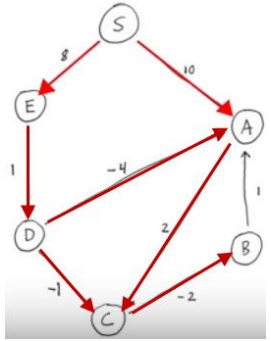
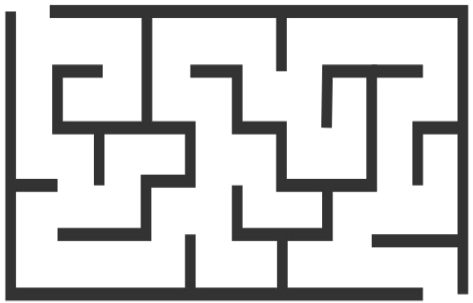


Table of Content

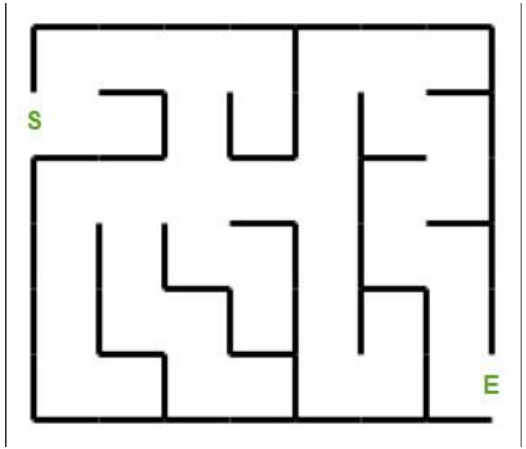
What is a Maze?



- A **maze** is a path or collection of paths, typically from an entrance to a goal.
- A maze can be viewed as a graph, if we consider each intersection in the maze to be a vertex, and we add edges to the graph between adjacent junctures that are not blocked by a wall.
- Our mazes will have random weights specified between any two adjacent intersection. These values can be thought of as the "cost" of traveling from one intersection to an adjacent one. The weights will be used for running Dijkstra's algorithm.

Table of Content

Design (Problem Statement)



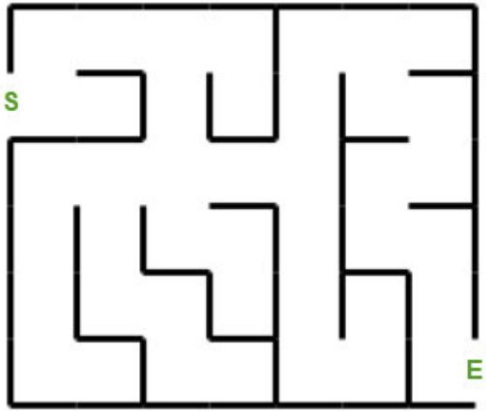
- Find the shortest path of the following maze .
  - Dijkstra's Algorithm
  - Bellman Ford's Algorithm
  - Prim's & Kruskal's minimum spanning tree Algorithm
- Convert the maze into a weighted Graph.
- Each node of the tree representation of the maze should be labeled sequentially and each edge should have a number indicating the distance.
- Apply the different Algorithms to find the shortest path in the weighted graph and compare .

Table of Content

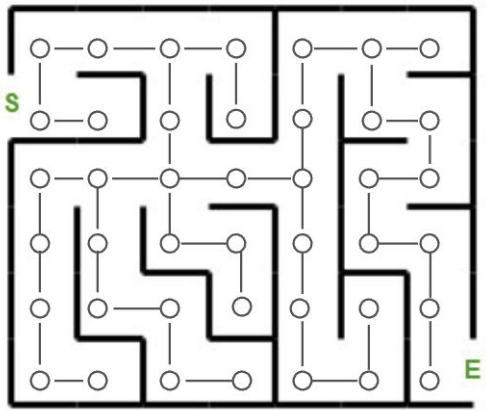
Implementation (Convert Maze to Weighted Graph)

- A maze can be viewed as a graph, if we consider each juncture (intersection) in the maze to be a vertex, and we add edges to the graph between adjacent junctures that are not blocked by a wall.

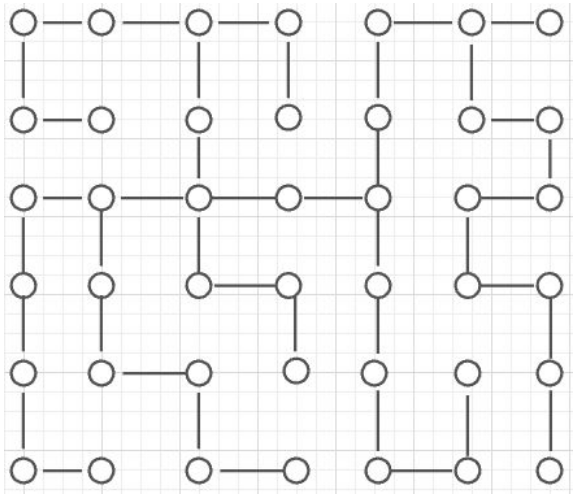
Step 1



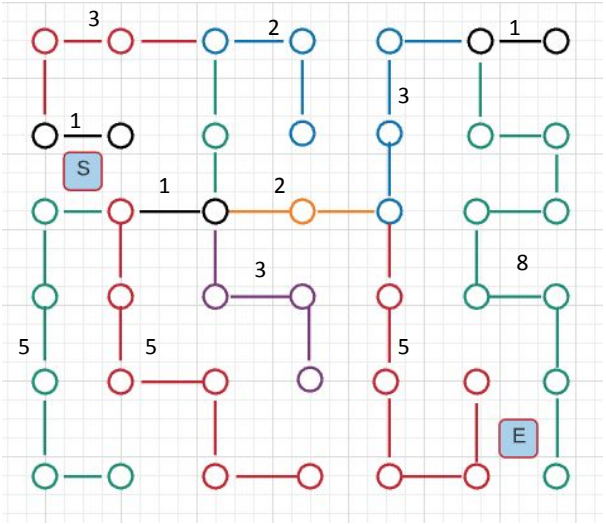
Step 2



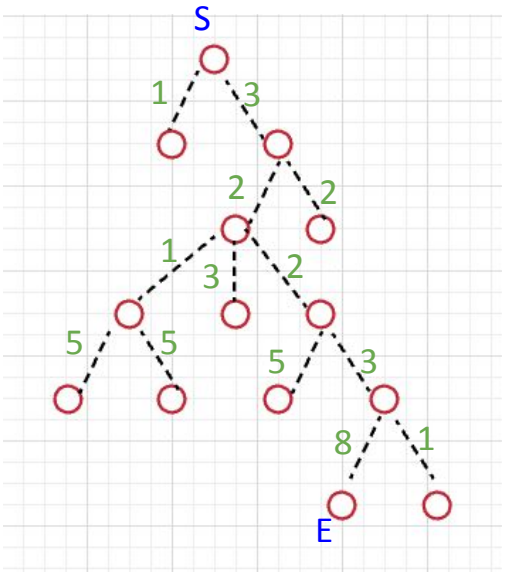
Step 3



Step 4



Step 5



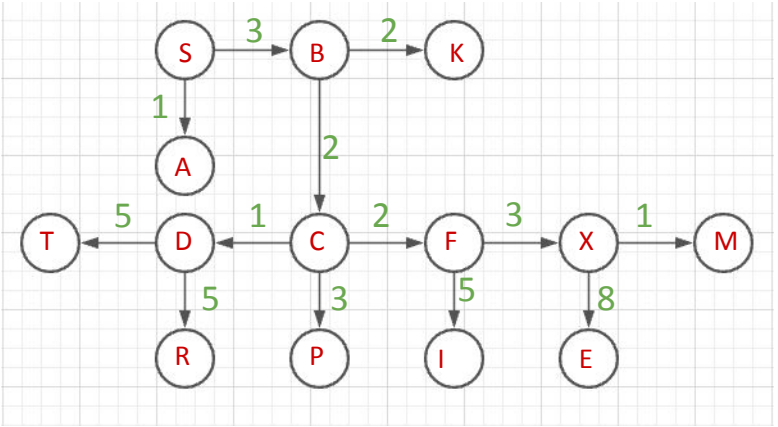


Table of Content

Dijkstra Algorithm to Weighted Graph

Dijkstra's Algorithm

- Dijkstra's algorithm solves the single-source shortest-paths problem on a directed weighted graph $G = (V, E)$, where all the edges are non-negative (i.e., $w(u, v) \geq 0$ for each edge $(u, v) \in E$).

- In simple words, 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.

Vertex(accumulated path)	Initial	Step1 S	Step 2 (S,B)	Step 3 (S,B,C)	Step 4 (S,B,C,D)	Step 5 (S,B,C,D,F)	Step 6 (S,B,C,D,F,X)	Step 7 (S,B,C,D,F,X,E)
	Next Step S	Next Step B	Next Step C	Next Step D	Next Step F	Next Step X	Next Step E	
S	0	0	0	0	0	0	0	0
B	In	3	3	3	3	3	4	4
A	In	1	1	1	1	1	1	1
C	In	In	5	5	5	5	5	5
K	In	In	5	5	5	5	5	5
P	In	In	In	8	8	8	8	8
D	In	in	In	6	6	6	6	6
F	In	In	In	7	7	7	7	7
R	In	In	In	In	11	11	11	11
T	In	In	In	In	11	11	11	11
I	In	In	In	In	In	12	12	12
X	In	In	In	In	In	10	10	10
M	In	In	In	In	In	In	11	11
E	In	In	In	In	In	In	18	18

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

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

S is selected as the starting point for Step 1.

- From S, one can go to S or A or B
  - The accumulated cost on S is not changed. It is still 0.
  - The accumulated cost on A is 1.
  - The accumulated cost on B is 3.
  - 1 is smaller than 3.
    - But, B is selected as the starting point for Step 2 because A do have any further intersection.
- Stop if the destination node E is reached you will find the minimum distance of **E** from **S** is **18**.

The Shortest path is **S>B>C>D>F>X>E**

Bellman Ford’s Algorithm to Weighted Graph

Bellman Ford’s Algorithm

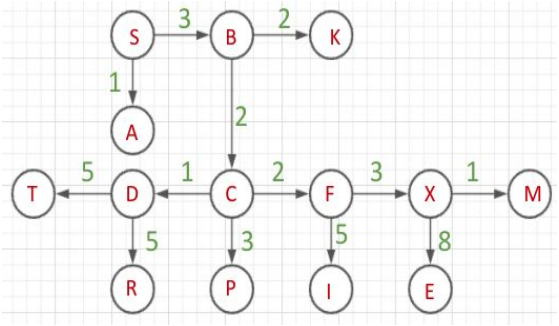
- This algorithm solves the single source shortest path problem of a directed graph G = (V, E) in which the edge weights may be negative.

- The Bellman-Ford algorithm is a graph search algorithm that finds the shortest path between a given source vertex and all other vertices in the graph. This algorithm can be used on both weighted and unweighted graphs.

Step 1: Initialization

Next node to visit => B

S	A	B	C	K	P	D	F	R	T	I	X	M	E
0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞



Step 2: 0+1 < ∞. Change the value of A to 1
0+3 < ∞. Change the value of B to 3

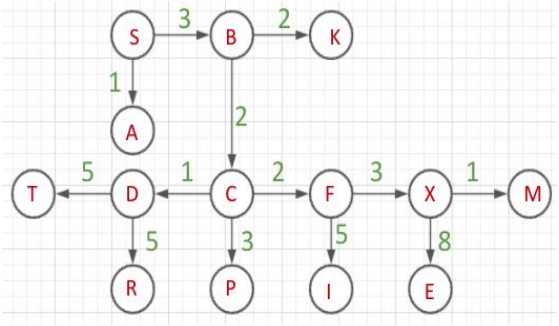
Next node to visit => B

S	A	B	C	K	P	D	F	R	T	I	X	M	E
0	1	3	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞

Step 3: 3+2=5 < ∞. Change the value of k to 5
3+2=5 < ∞. Change the value of C to 5

Next node to visit => C

S	A	B	C	K	P	D	F	R	T	I	X	M	E
0	1	3	5	5	∞	∞	∞	∞	∞	∞	∞	∞	∞



Step 4: 5+1=6 < ∞. Change the value of D to 6
5+2=7 < ∞. Change the value of F to 7
5+3=8 < ∞. Change the value of P to 8

Next node to visit => D

S	A	B	C	K	P	D	F	R	T	I	X	M	E
0	1	3	5	5	8	6	7	∞	∞	∞	∞	∞	∞

Step 5: 5+6=11 < ∞. Change the value of T to 11
5+6=11 < ∞. Change the value of R to 11

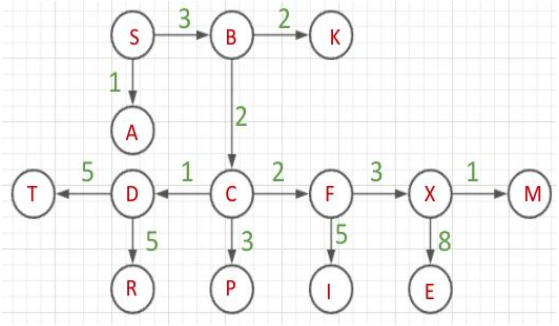
Next node to visit => F

S	A	B	C	K	P	D	F	R	T	I	X	M	E
0	1	3	5	5	8	6	7	11	11	∞	∞	∞	∞

Step 6: 7+5=12 < ∞. Change the value of I to 12
7+3=10 < ∞. Change the value of X to 10

Next node to visit => X

S	A	B	C	K	P	D	F	R	T	I	X	M	E
0	1	3	5	5	8	6	7	11	11	12	10	∞	∞



Step 7: 10+1=11 < ∞. Change the value of M to 11
10+8=18 < ∞. Change the value of E to 18

S	A	B	C	K	P	D	F	R	T	I	X	M	E
0	1	3	5	5	8	6	7	11	11	12	10	11	18

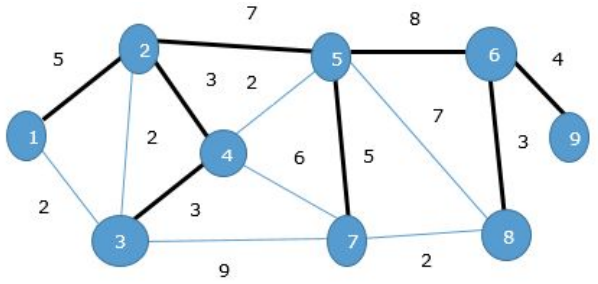
The process ends at cycle one as there are no vertices to change.

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

Prim’s & Kruskal’s minimum spanning tree Algorithm to Weighted Graph

Spanning Tree

- A **spanning tree** is a subset of an undirected Graph that
  - has all the vertices connected by minimum number of edges.
  - A spanning tree does not have any cycle.
  - Any vertex can be reached from any other vertex.
- If all the vertices are connected in a graph, then there exists at least one spanning tree.
- In the following graph, we have shown a spanning tree though it's not the minimum spanning tree. The cost of this spanning tree is (5 + 7 + 3 + 3 + 5 + 8 + 3 + 4) = 38.

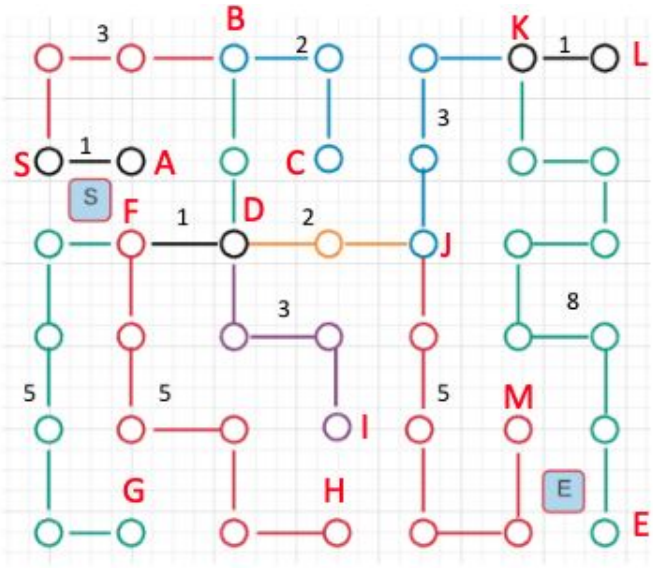


Minimum Spanning Tree

- 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.
- To derive an MST, Prim's algorithm or Kruskal's algorithm can be used.
- 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.**

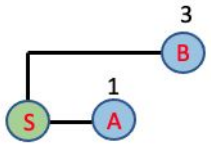
Prim’s Minimum Spanning Tree

A. Create a set mstSet that keeps track of vertices already included in MST.
B. Assign a key value to all vertices in the input graph.
    a. Initialize all key values as INFINITE.
    b. Assign key value as 0 for the first vertex so that it is picked first.
C. While mstSet doesn’t include all vertices
    a. Pick a vertex u which is not there in mstSet and has minimum key value.
    b. Include u to mstSet.
    c. Update key value of all of u's adjacent vertices which are not in mstSet.

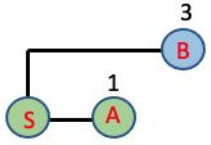


Prim’s Minimum Spanning Tree

STEP 1 :



STEP 2 :

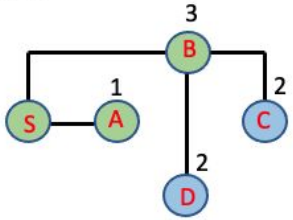


S->A

1

Starting from S, the tree can link to two nodes then choose the one with the smallest weight which is A and then go to STEP 2.

STEP 3 :

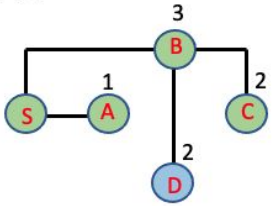


S->A->B

1+3=4

Since A doesn’t connect to other nodes, go to next smallest node which is B .

STEP 4 :



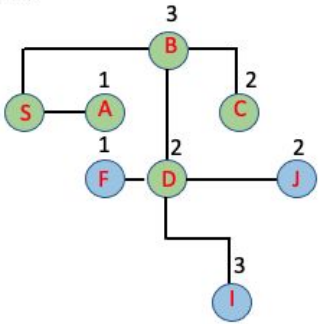
S->A->B->C

1+3+2=6

Now repeat the rule like step 1,2 and keep choosing the smallest nodes till all the vertices are visited.

Prim's Minimum Spanning Tree

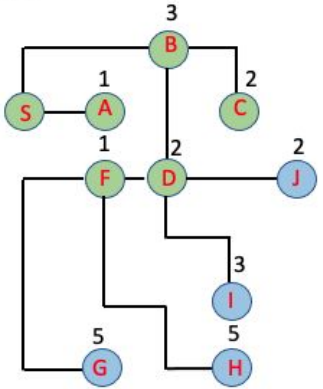
STEP 5 :



S->A->B->C->D

1+3+2+2=8

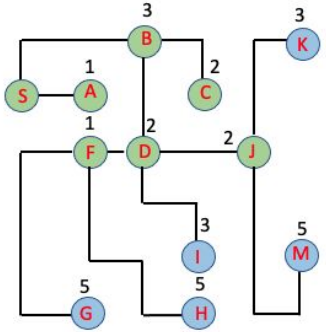
STEP 6 :



S->A->B->C->D->F

1+3+2+2+1=9

STEP 7 :

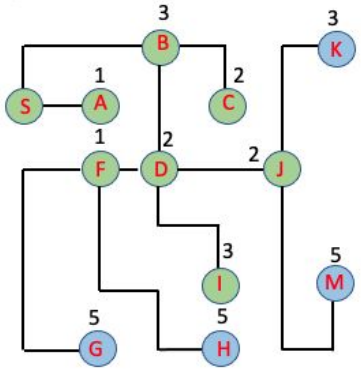


S->A->B->C->D->F->J

1+3+2+2+1+2=11

Prim’s Minimum Spanning Tree

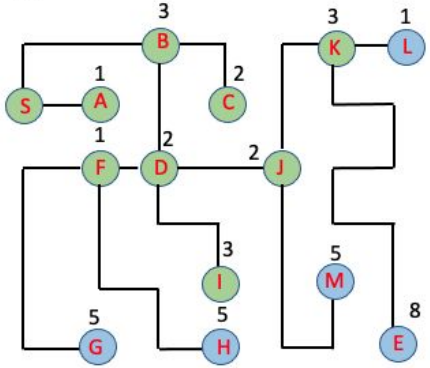
STEP 8 :



S->A->B->C->D->F->J->I

1+3+2+2+1+2+3=14

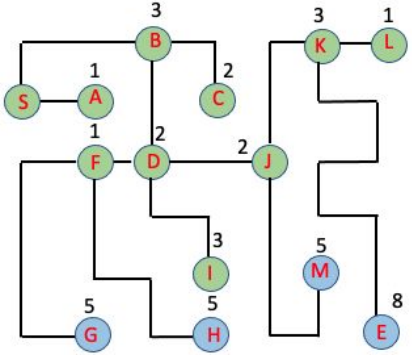
STEP 9 :



S->A->B->C->D->F->J->I->K

1+3+2+2+1+2+3+3=17

STEP 10 :

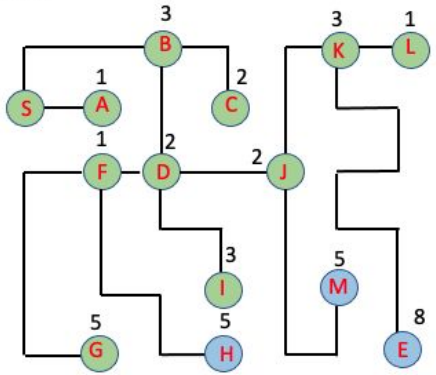


S->A->B->C->D->F->J->I->K->L

1+3+2+2+1+2+3+3+1=18

Prim’s Minimum Spanning Tree

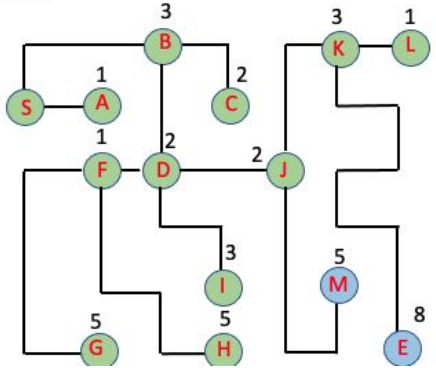
STEP 11 :



S->A->B->C->D->F->J->I->K->L->G

1+3+2+2+1+2+3+3+1+5=23

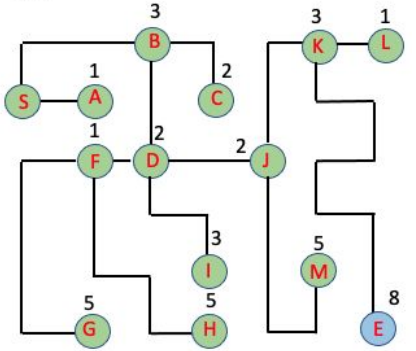
STEP 12 :



S->A->B->C->D->F->J->I->K->L->G->H

1+3+2+2+1+2+3+3+1+5+5=28

STEP 13 :

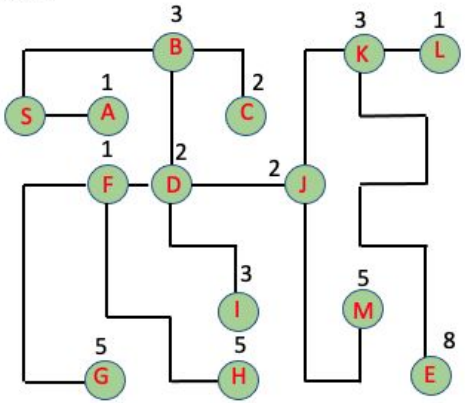


S->A->B->C->D->F->J->I->K->L->G->H->M

1+3+2+2+1+2+3+3+1+5+5+5=33

Prim’s Minimum Spanning Tree

STEP 14 :



The shortest path of the given maze with Prim’s minimum spanning tree is as follows:

S->A->B->C->D->F->J->I->K->L->G->H->M->E

1+3+2+2+1+2+3+3+1+5+5+5+8=41

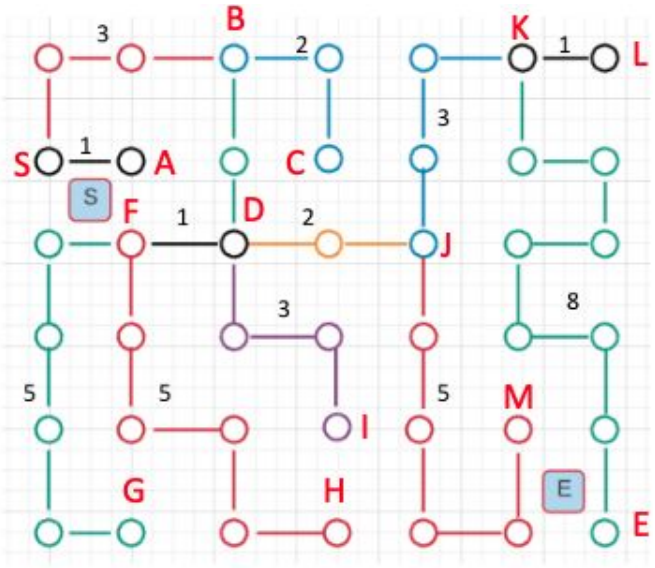
Total steps 14

Time Complexity :

O((v + E)logV)

Kruskal's Minimum Spanning Tree

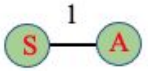
1. Sort all the edges in non-decreasing order of their weight.
2. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.
3. Repeat step#2 until there are (V-1) edges in the spanning tree.



Kruskal's Minimum Spanning Tree

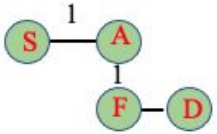
After Sorting		
Weight	**Source**	**Destination**
1	S	A
1	D	F
1	K	L
2	B	C
2	B	D
2	D	J
3	S	B
3	D	I
3	J	K
5	F	G
5	F	H
5	J	M
8	K	E

STEP 1 :

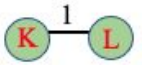


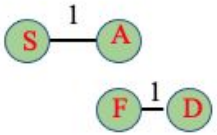
The smallest weight is 1 which edge is S – A.

STEP 2 :



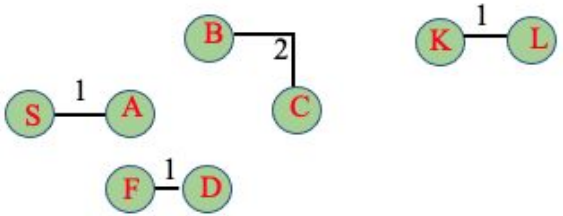
STEP 3 :



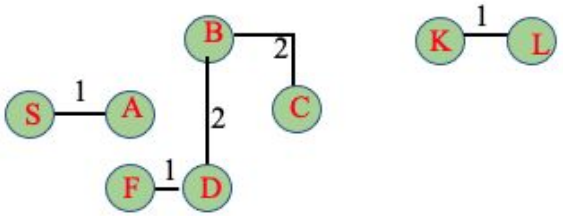


Kruskal's Minimum Spanning Tree

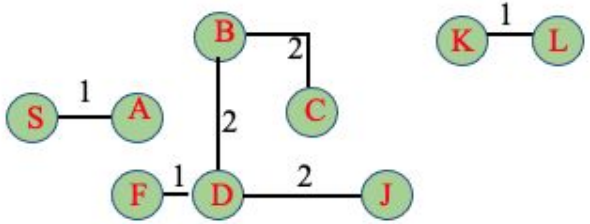
STEP 4 :



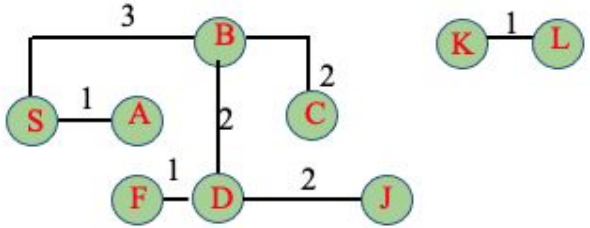
STEP 5 :



STEP 6 :

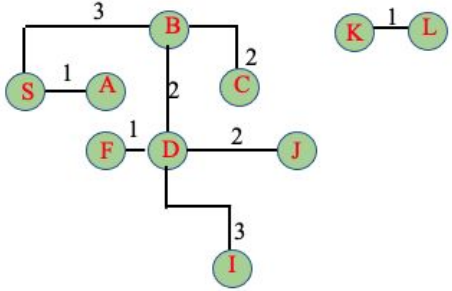


STEP 7 :

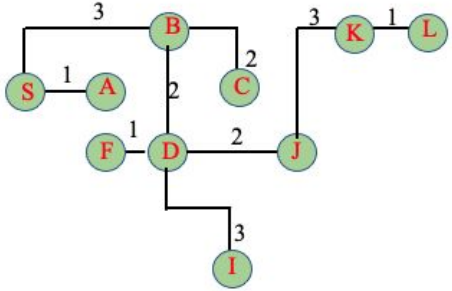


Kruskal's Minimum Spanning Tree

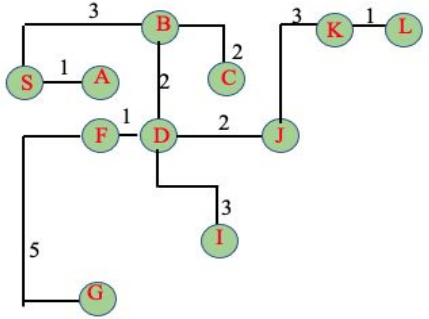
STEP 8 :



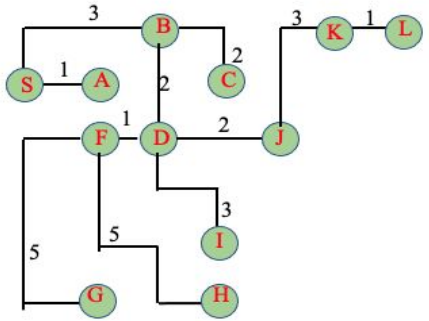
STEP 9 :



STEP 10 :

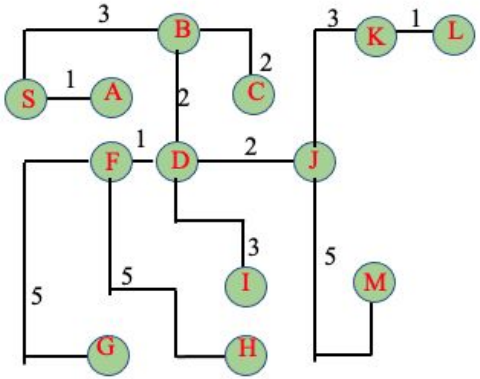


STEP 11 :

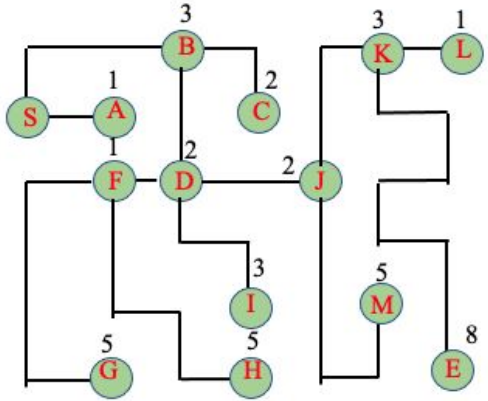


Kruskal's Minimum Spanning Tree

STEP 12 :



STEP 13 :



The graph contains 14 vertices . So, the minimum spanning tree formed will be having (14 – 1) = 13 edges.

Since the number of edges equals to (V – 1), the algorithm stops here.

Total weight = 41

Total steps 13

Time Complexity : O(E * logV)

Table of Content

Applications

- **Dijkstra's Algorithm** has several real-world use cases, some of which are as follows:
  - Digital Mapping Services in Google Maps
  - Social Networking Applications
  - IP routing to find Open shortest Path First
  - Robotic Path
  - Designate file server
  - Telephone Networks

- A version of **Bellman-Ford** is used in the distance-vector routing protocol. This protocol decides how to route packets of data on a network.

- **Minimum spanning trees** have direct applications in the design of networks, including computer networks, **telecommunications** networks, transportation networks, water supply networks, and electrical grids

Table of Content

Conclusion

- The only difference between the two is that **Bellman-Ford** is also capable of handling negative weights whereas **Dijkstra** Algorithm can only handle positives.**Bellman-Ford** algorithm is a single-source shortest path algorithm, so when we have negative edge weight then it can detect negative cycles in a graph.
- The complexity of **Bellman-Ford** algorithm with respect to time is slower **than** the algorithm of **Dijkstra**.
- Dijkstra's algorithm can work on both directed and undirected graphs, but Prim's algorithm only works on undirected graphs
- Prim's algorithm gives connected component as well as it works only on connected graph.Prim's algorithm runs faster in dense graphs.
- Kruskal's algorithm can generate forest(disconnected components) at any instant as well as it can work on disconnected components.Kruskal's algorithm runs faster in sparse graphs.

Bibliography / References

1. https://en.wikipedia.org/wiki/Bellman%E2%80%93Ford_algorithm
2. https://en.wikipedia.org/wiki/Edsger_W._Dijkstra
3. https://brilliant.org/wiki/dijkstras-short-path-finder/
4. https://npu85.npu.edu/~henry/npu/classes/algorithm/tutorialpoints_daa/slide/shortest_paths.html
5. https://www.lucidchart.com/pages/
6. http://www.cs.umd.edu/class/spring2019/cmsc132-020X-040X/Project8/proj8.html
7. https://en.wikipedia.org/wiki/Maze#Solving_mazes
8. https://brilliant.org/wiki/bellman-ford-algorithm/#:~:text=The%20Bellman%2DFord%20algorithm%20is,both%20weighted%20and%20unweighted%20graphs.
9. https://www.geeksforgeeks.org/applications-of-dijkstras-shortest-path-algorithm/
10. https://npu85.npu.edu/~henry/npu/classes/algorithm/tutorialpoints_daa/slide/spanning_tree.html