Dijkstra's , Bellman's Ford,MST Algorithm: To calculate the shortest path using a MAZE

DIRECTED BY:
PROF.Henry Chang

DONE BY:
SAHITI EMANI
STUDENT ID:19556

TABLE OF CONTENTS:

INTRODUCTION:

Dijkstra's Algorithm: The shortest path between two vertices is a path with the shortest length (least number of edges). Call this the link-distance.

Time Complexity of Dijkstra's Algorithm is
O(V2)
but with min-priority queue it drops down to
O(V+ElogV)

.**Intuition behind Dijkstra's Algorithm**

The vertices in increasing order of their distance from the source vertex. Construct the shortest path tree edge by edge; at each step adding one new edge, corresponding to construction of shortest path to the current new vertex.

DESIGN AND PROCESS:

Use Dijkstra’s Algorithm to find the shortest path of the following maze:

STEP-1: Applying Dijkstra’s Algorithm to find the shortest path and join the dots and move forward accordingly,name the nodes for better understanding and draw the tree diagram .As shown below:

FIG:1

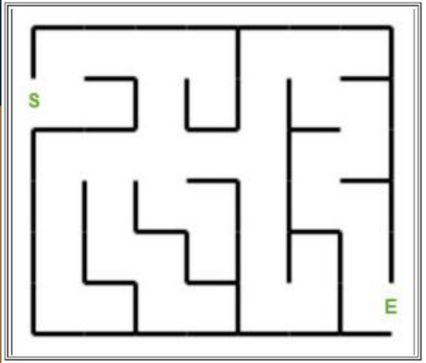


FIG:1.1

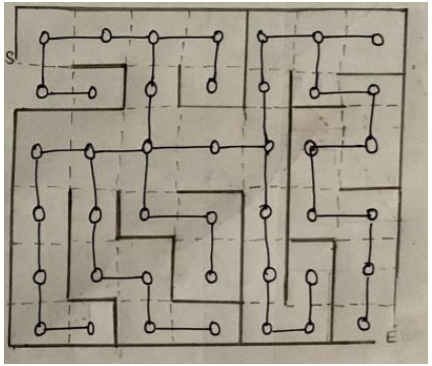


FIG:1.2

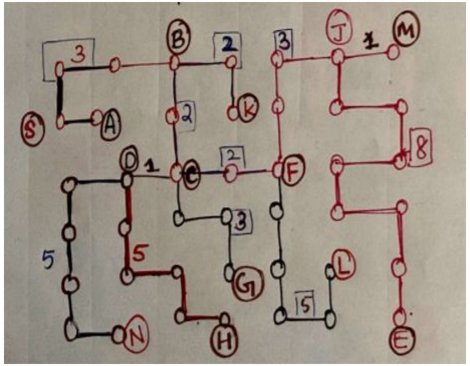
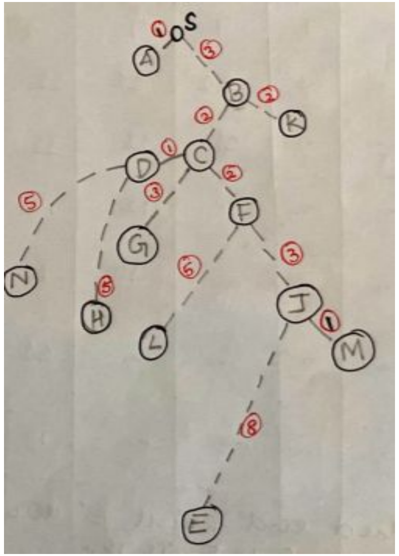


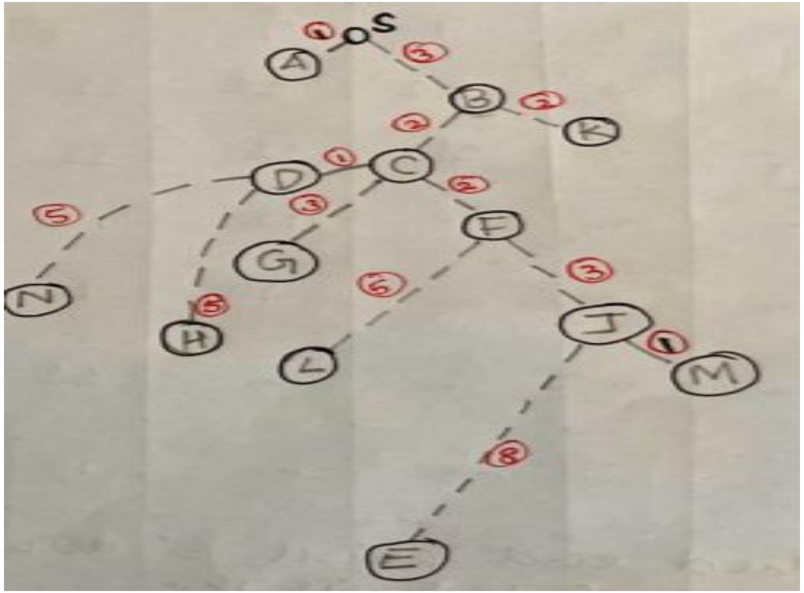
FIG:1.3



IMPLEMENTATION:

From,the maze diagram we can see two nodes that indicate S and E,they are indicated as the Start(“S”) and End(Destination”E”).One important rule to start the implementation of the djikstra’s Algorithm i.e,STARTING WITH S as”0” i.e considered to the INITIAL STEP to start the process and the other vertices are considered to start initially,with “INFINITY”.

Now, we start implementing the maze with required steps and find the minimum distance.



IMPLEMENTATION:Cont’d

Start with “**S**” **and form it in a tabular form for better understanding and continue till you find the shortest distance until you reach the final destination “E” and we stop after that.**

Starting Point.	I Initial step→S	Step 1 B	Step 2 C	Step 3 D	Step-4 F	Step-5 I	Step-6 E	Step-7
S	0	0	0	0	0	0	0	0
B	∞	3	3	3	3	3	4	4
A	∞	1	1	1	1	1	1	1
C	∞	∞	5	5	5	5	5	5
K	∞	∞	5	5	5	5	5	5
G	∞	∞	∞	8	8	8	8	8
D	∞	∞	∞	6	6	6	6	6
F	∞	∞	∞	7	7	7	7	7
H	∞	∞	∞	∞	11	11	11	11

IMPLEMENTATION:Cont'd

Since,We continued the process until we reached **"E".Hence, we got the shortest path from S to E i.e:18.**

N	∞	∞	∞	∞	11	11	11	11
L	∞	∞	∞	∞	∞	12	12	12
J	∞	∞	∞	∞	∞	10	10	10
M	∞	∞	∞	∞	∞	∞	11	11
E	∞	∞	∞	∞	∞	∞	18	18

Since, we reached end point 'E' the shortest path S to E is 18.

The shortest path.

∴ $\boxed{S \rightarrow B \rightarrow C \rightarrow D \rightarrow F \rightarrow J \rightarrow E}$

TESTING:

Initial

- S is smallest cost on Initial step.
  - Thus, S is slected as the starting point for Step 1.

Step 1

- S is selected as the starting point for Step 1.
  - From **S**$_1$, one can go to **A**$_1$ or **B** .
    - The accumulated cost on **S**$_1$ is not changed. It is still 0.
    - The accumulated cost on **A** is **1**.
    - The accumulated cost on **B** is **2**.
    - 1 is smaller than 3.
      - Thus, **A** is selected as the starting point but there no following path from A so the values remain same and we will choose B for starting **STEP 2**.

Step 2

- **B** is selected as the starting point for Step 2.
  - From B$_3$, one can go to C or K.
    - The accumulated cost on **B** is not changed from **3**.
    - The accumulated cost on **C** is changed. It is till **5**.
    - The accumulated cost on K is changed to 5 .
    - Comparing C,K have Same value.So,choose Cas no further path from K.
      - Thus, the next smallest number is picked, and C is selected as the starting point on Step 3.

TESTING:Cont’d

Step 3

- C is selected as the starting point for Step 3.
  - From C, one can go to **D**,**G** or **F**.
    - The accumulated cost on **B** is not changed. It is still 5.
    - The accumulated cost on **D** is 6.
    - The accumulated cost on G is 8.
    - The accumulated cost on **F** is 7.
    - Comparing,6,7,8 value i.e 6 is smaller..
      - Thus, the next smallest number 6,is picked, and D is selected as the starting point on Step 4.

Step 4

- C is selected as the starting point for Step 3.
  - From D, one can go to **N** or **H**.
    - The accumulated cost on **D** is not changed. It is still 6.
    - The accumulated cost on N is 11.
    - The accumulated cost on H is 11.
    - Comparing N,H have same value i.e 11.
      - Thus, the next smallest number is picked,but here the value is same and F is selected as the starting point on Step 5 because F is smaller than G..

TESTING:Cont'd

Step 5

- F is selected as the starting point for Step 5.
  - From F, one can go to **L AND J**.
    - The accumulated cost on **F** is not changed. It is still 7.
    - The accumulated cost on **L** is 12.
    - The accumulated cost on J is 10.
    - Comparing,10,12 value i.e 10 is smaller..
      - Thus, the next smallest number 10,is picked, and J is selected as the starting point on Step 6.

Step 6

- J is selected as the starting point for Step 6.
  - From J, one can go to **M** or **E**.
    - The accumulated cost on **J** is **10** as of now..
    - The accumulated cost on M is changed to 11.
    - The accumulated cost on E is changed to 18.
    - Comparing N,H have same value i.e 11.
      - Thus, the next smallest number is picked,but here the value is same and M is selected as the starting point on Step 7 because M is smaller than E..

TESTING:Cont'd

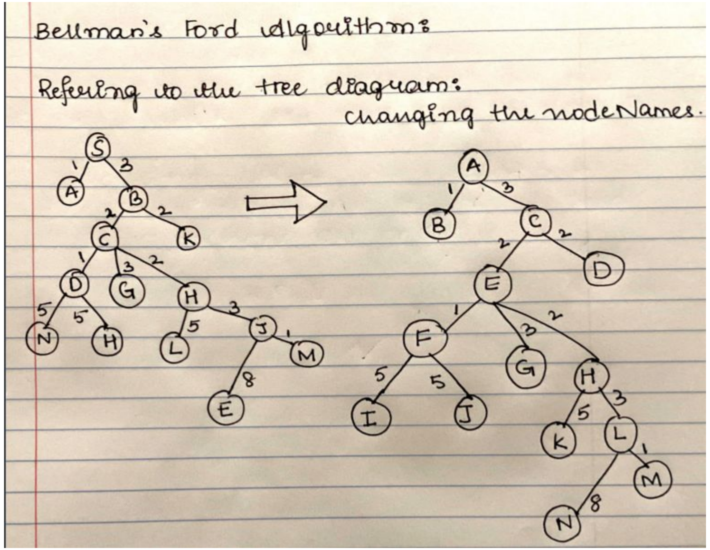
Step 7

- M is selected as the starting point for Step 7.
    - From M, one can go to **no further path is connected so the value remains same i.e 11.**.
        - Now,choose the nodes thar not visited during these steps.
        - The non-visited nodes are L and G.
        - The accumulated cost on L is 12.
        - The accumulated cost on G is 8.
        - Comparing L,G remains as 8 no more path from there after that we will go to L with the value as **12**.
    - Hence, we reached the final destination i.e **E**.
    - Shortest Path is **18**.
    - So,the direction of the shortest path is **S-B-C-D-F-J-E.**

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 single source shortest path algorithm (for arbitrary weight positive or negative) is also known Bellman-Ford algorithm is used to find minimum distance from source vertex to any other vertex.



BELLMAN's ALGORITHM FOR MAZE:

Cycle 1:

Step 1: Let us consider 'A' as our first node.

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 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	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞

Step 2: Now, Let's take 'B' as our Node.

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

Note:

- Since, From B there are no further nodes avaliable. So, the Value of 'B' is unchanged.

Step 3: Let's take 'C' as our next Node.

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 5.
$3+2=5<\infty$, E's Value is Changed to 5.

So,

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

BELLMAN's ALGORITHM:Cont'd

Step4: Now, let's select 'D' as fourth node

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

- Since, From 'D' no other nodes avaliable So, D value is not changed any futher.

Step 5: Let's select the Node to be '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.
$5+3=8<\infty$, G's value is changed to 8.
$5+2=7<\infty$, 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	∞	∞	∞	∞	∞	∞

Step6: Let's Select 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 Now, 11
$6+5=11<\infty$, J's Value is Now, 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	∞	∞	∞	∞

Step 7:- Let's select 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 avaliable any further. So, the Value of 'G' is unchanged

BELLMAN's ALGORITHM:Cont'd

Step8:-Let's select 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: $7+5=12<\infty$ K's is changed to 12.
$7+3=10<\infty$ L's Value is 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	∞	∞

BELLMAN's ALGORITHM:Cont'd

Step9:- Let's select the Node is '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, I has no further Nodes, So the 'I' value remains unchanged.

Step10: Let's 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' there no further nodes available.

Step 11:- Let's select the next node '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	∞	∞

Since, there are no further nodes are available. So, the Value of 'K' is unchanged.

BELLMAN's ALGORITHM:Cont'd

Step12: select the node is '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 11 now.
since, $10+8=11<\infty$, N's value is 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

Step13: Select the Node is '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: M's value is not changed so there are no further nodes.

Step14: Select the Node is '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: 'N' no nodes that are available.

BELLMAN's ALGORITHM:Cont'd

cycle 2:-

Now, we start the 2nd iteration.

Step 1:- Let's select the Node i.e '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 not changed
- since, 0+3 = 3 C's value is not changed

Step 2:- Select the Node i.e 'B'.

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

Note: since, the value of 'B' is unchanged.

Step 3:- select the Node i.e '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.

BELLMAN’s ALGORITHM:Cont’d

So,in this way we will visit all the nodes until the last node i.e”N”.

The steps and the values of the nodes remain unchanged because they are positive directions only provided in the maze diagram.

there fore,there is no further change in the values because bellman’s ford algorithm is more about finding the shortest distance only when there are negative direction.But in our case we have only positive directions and therefore we will not find any change in the nodes and values even if we continue till 13 cycles.

Since we have 14 vertices=13 iterations the values remain unchanged.

TIME COMPLEXITY: DIJKSTRA'S AND BELLMAN'S FORD ALGORITHM

The complexity of Dijkstra's algorithm is O(V+E.logV) , where is the number of nodes, and is the number of edges in the graph.
when working with dense graphs, where **E** is close to **V^2 ,** if we need to calculate the shortest path between any pair of nodes, using Dijkstra's algorithm is not a good option.

The reason for this is that Dijkstra's time complexity is **O(V+E.logV)** . Since **E** equals almost **V^2** , the complexity becomes **(O(V+V^2 log(V))).**

In the **Bellman-Ford algorithm**, we begin by initializing all the distances of all nodes with , except for the source node, which is initialized with zero. Next, **we perform V-1 steps**.

After **V-1** steps, all the nodes will have the correct distance, and we stop the algorithm.

The Bellman-Ford algorithm's time complexity is O(V.E), where **V** is the number of vertices, and **E** is the number of edges inside the graph. The reason for this complexity is that we perform **V** steps. In each step, we visit all the edges inside the graph.

SPANNING TREE:PRIM’S AND KRUSKAL’S(MST)

A **spanning tree** is a subset of an undirected Graph that has all the vertices connected by minimum number of edges.

If all the vertices are connected in a graph, then there exists at least one spanning tree. In a graph, there may exist more than one spanning tree.

Properties

- A spanning tree does not have any cycle.
- Any vertex can be reached from any other vertex.

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.

PRIM’S:Prim’s algorithm is also a Greedy algorithm. It starts with an empty spanning tree. The idea is to maintain two sets of vertices.

KRUSKAL’S:Sort all the edges in non-decreasing order of their weight.

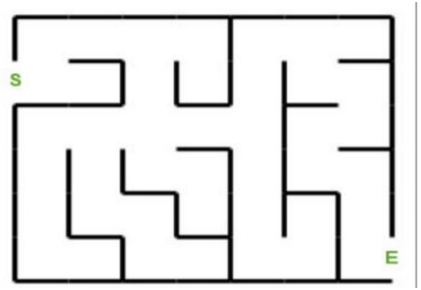
1. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.
2. Repeat step#2 until there are (V-1) edges in the spanning tree.

PRIM'S SPANNING TREE:

Use Prim's Minimum Spanning Tree algorithm to find the shortest path of a maze.

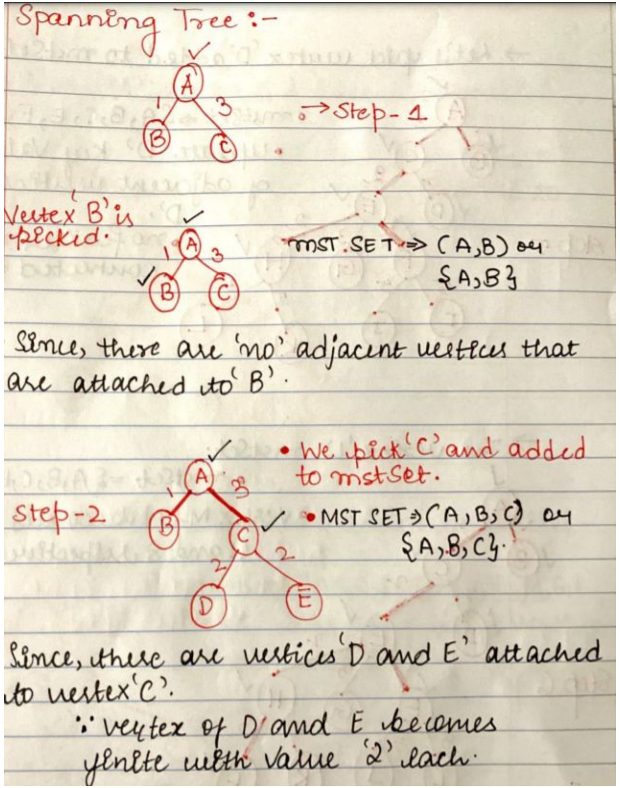
STEPS MENTIONED BELOW:

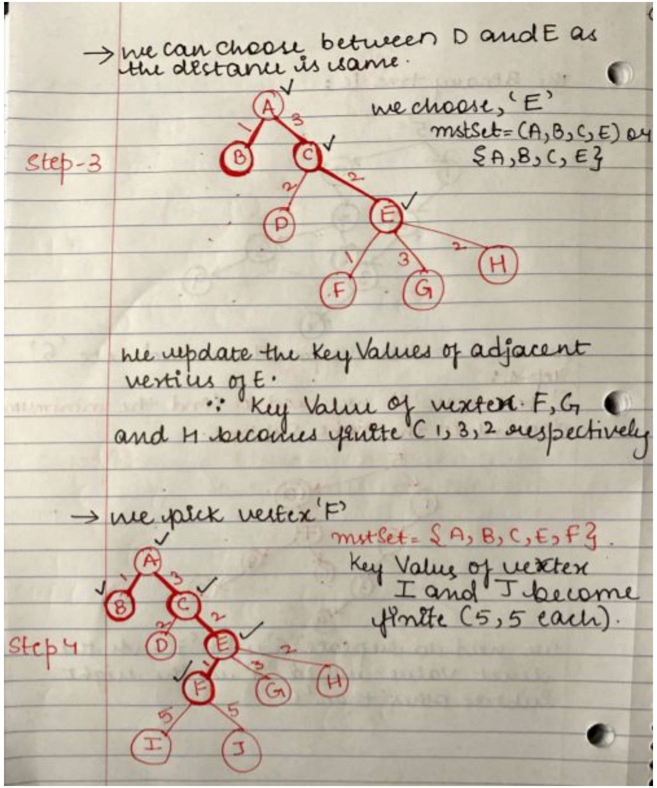
1. Create a set mstSet that keeps track of vertices already included in MST.
2. Assign a key value to all vertices in the input graph.
    1. Initialize all key values as INFINITE.
    2. Assign key value as 0 for the first vertex so that it is picked first.
3. While mstSet doesn't include all vertices
    1. Pick a vertex u which is not there in mstSet and has minimum key value.
    2. Include u to mstSet.
    3. 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



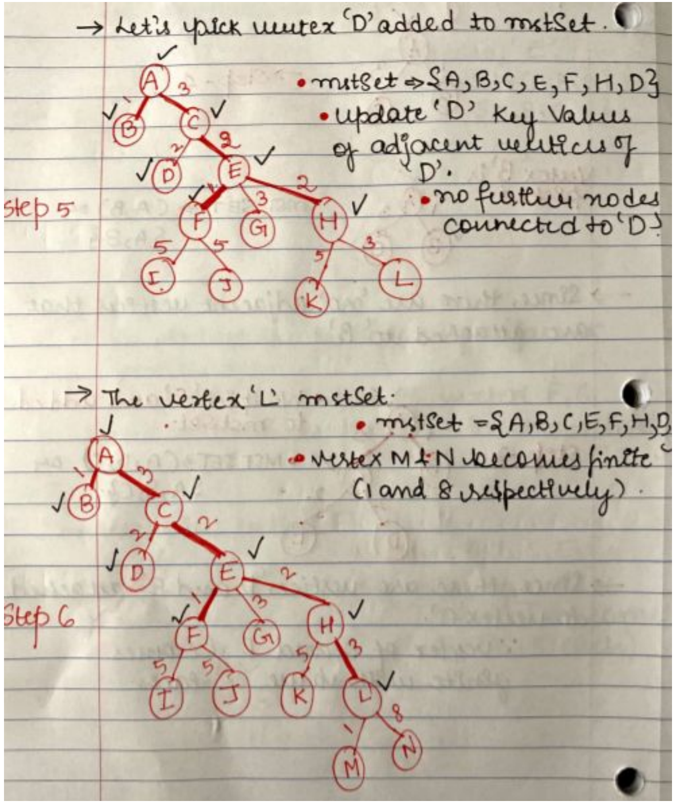
PRIM'S SPANNING TREE:Cont'd

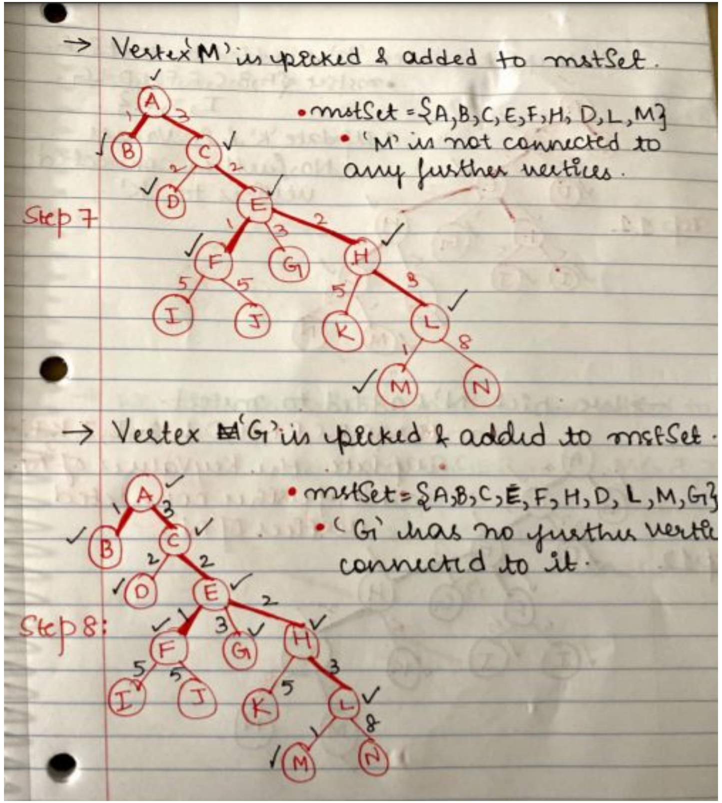
Solving MST for the MAZE:



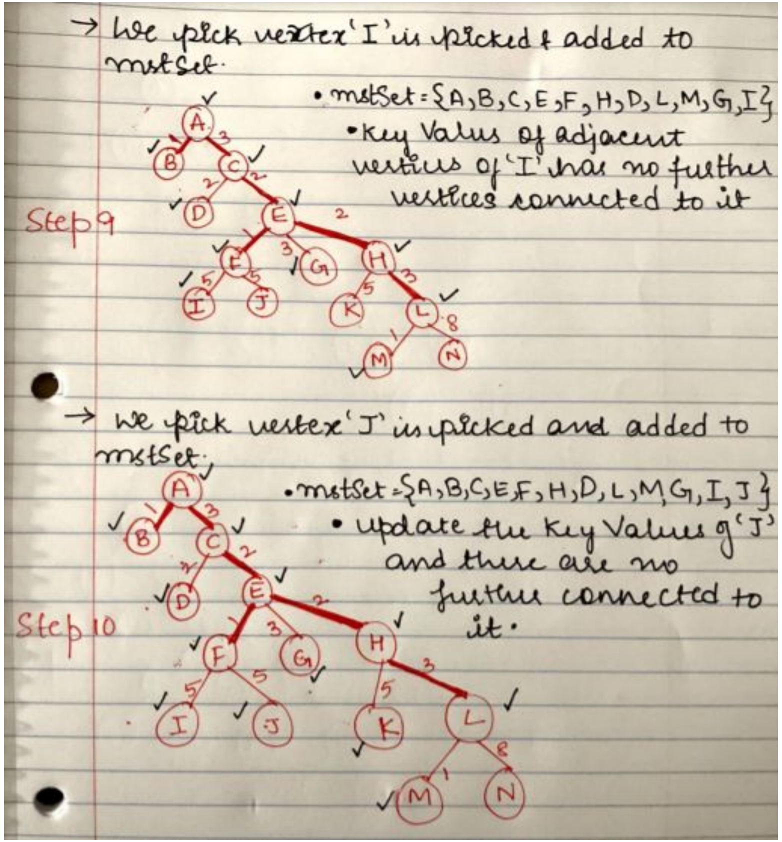


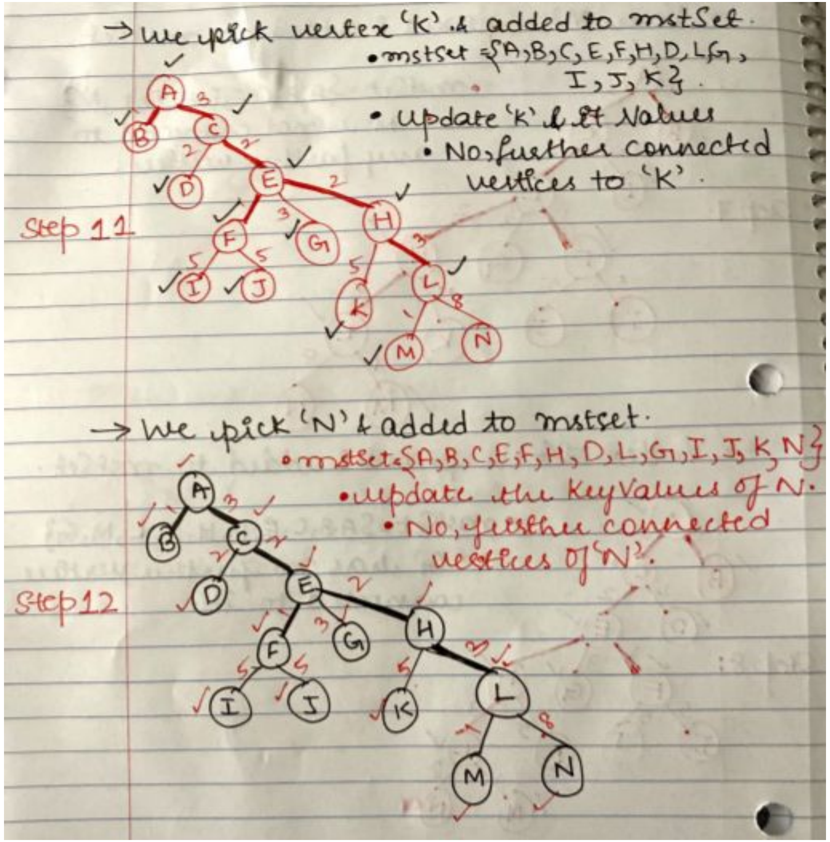
PRIM'S SPANNING TREE:Cont'd





PRIM'S SPANNING TREE:Cont'd





PRIM’S SPANNING TREE:Cont’d

As all the vertices(nodes) are visited, now the algorithm stops.

The cost of the spanning tree is (1+ 3 +2 +1 + 2 + 2+ 3 + 1 +3 + 5+ 5+5+ 8) = 41.

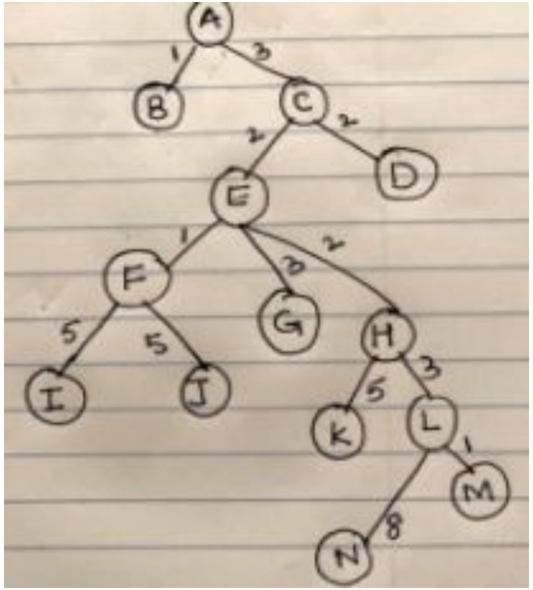
There is no more spanning tree in this graph with cost less than 41.

NOTE:

- The idea of using key values is to pick the minimum weight edge from cut.
- The key values are used only for vertices which are not yet included in MST, the key value for these vertices indicate the minimum weight edges connecting them to the set of vertices included in MST.

KRUSKAL'S 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 SPANNING TREE:

Kruskal's Minimum Spanning Tree:-

After Sorting:

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

KRUSKAL'S SPANNING TREE:Cont'd

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

1. Pick edge A-B: No cycle is formed, include it.
2. Pick edge E-F: No cycle is formed, include it.
3. Pick edge L-M: No cycle is formed, include it.
4. Pick edge C-D: No cycle is formed, include it.
5. Pick edge C-E: No cycle is formed, include it.
6. Pick edge E-H: No cycle is formed, include it.
7. Pick edge A-C: No cycle is formed, include it.
8. Pick edge E-G: No cycle is formed, include it.
9. Pick edge H-L: No cycle is formed, include it.
10. Pick edge F-I: No cycle is formed, include it.
11. Pick edge F-J: No cycle is formed, include it.
12. Pick edge H-K: No cycle is formed, include it.
13. Pick edge L-N: No cycle is formed, include it. Since the number of edges equals to (V – 1)=>(14-1) =>13, the algorithm stops here.

TIME COMPLEXITY:PRIMS'S &KRUSKAL'S

Prim's Algorithm	Kruskal's Algorithm
It starts to build the Minimum Spanning Tree from any vertex in the graph.	It starts to build the Minimum Spanning Tree from the vertex carrying minimum weight in the graph.
It traverses one node more than one time to get the minimum distance.	It traverses one node only once.
Prim's algorithm has a time complexity of $O(V^2)$, V being the number of vertices and can be improved up to $O(E + \log V)$ using Fibonacci heaps.	Kruskal's algorithm's time complexity is $O(E \log V)$, V being the number of vertices.
Prim's algorithm gives connected component as well as it works only on connected graph.	Kruskal's algorithm can generate forest(disconnected components) at any instant as well as it can work on disconnected components
Prim's algorithm runs faster in dense graphs.	Kruskal's algorithm runs faster in sparse graphs.

CONCLUSION:

Dijkstra's Algorithm can only work with graphs that have positive weights, during the process, the weights of the edges have to be added to **find the shortest path**.

Dijkstra's algorithm is **BFS** with a priority queue.

But,its not applicable for negative weights.As an explanation i would like to mention in a form of an example,

It is difficult to comprehend if we consider an edge represents the distance between two cities. However, it makes sense if an edge represents the cost (negative number)or profit (positive number) for a business task.We can also use the edge to present the speed driving from one city to another one. Driving above an average speed is positive, below an average speed is negative.

Hence,we calculated the shortest path for the shortest path of the given maze i.e:with value as “S” to”E”==>18.

S==>B==>C==>D==>F==>J==>E

Bellman’s Ford Algorithm:is one of the SSSP algorithms to check for the existence of negative cyclesit 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.

The calculation of the maze is explained thoroughly with mentioned steps and clear difference is provided in the time complexity of both the algorithms.

BIBLIOGRAPHY:

https://www.hackerearth.com/practice/algorithms/graphs/shortest-path-algorithms/tutorial/

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

https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/

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

https://www.baeldung.com/cs/dijkstra-vs-bellman-ford

https://www.geeksforgeeks.org/kruskals-minimum-spanning-tree-algorithm-greedy-algo-2/

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