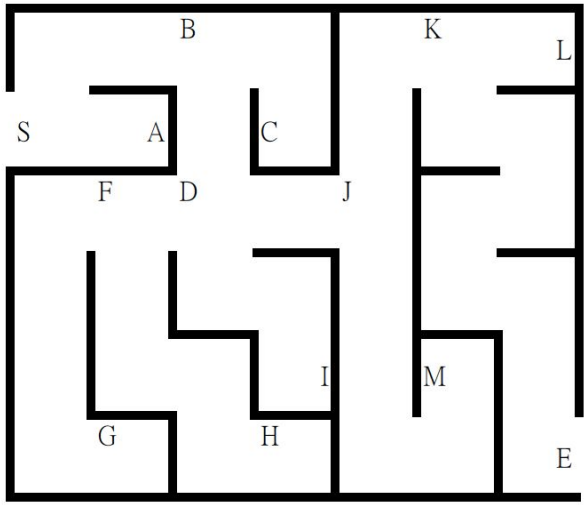
GET OUT OF A MAZE

Find the shortest path

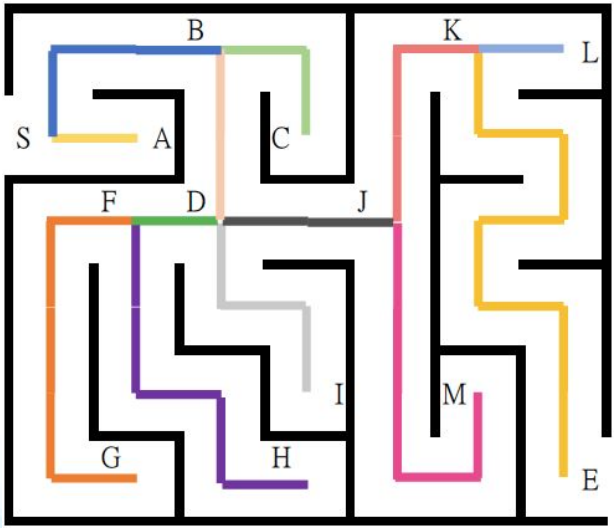
What's a maze

- A maze must includes at least one entry and one exit(goal).
- A maze can be find a path or paths from entries to exits.
- Look at the diagram in the right side, it is a maze with Entry-S and Exit-E.



How many path in the maze?

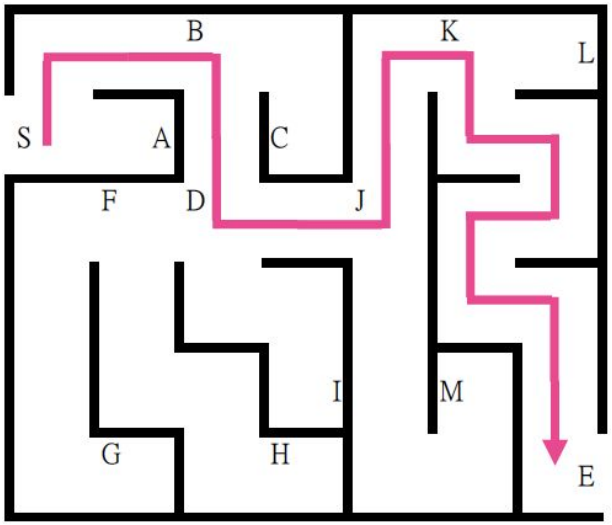
Take the maze the maze below for example:



This maze creates 13 paths which are SA, SB, BC, BD, DF, DI, DJ, FG, FH, JK, JM, KL, KE.

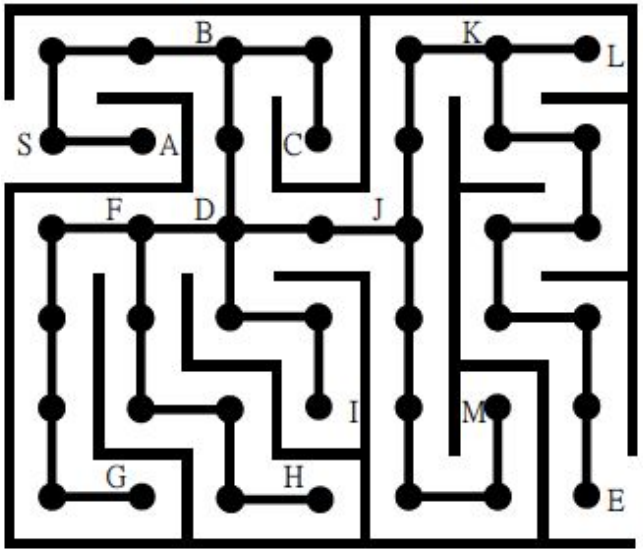
How to go from S to E?

- Take the paths SB, BD, DJ, JK, KE is the way from S to E.
- This simple example is easy for humans but if it's a complicated maze we will need computer to help.

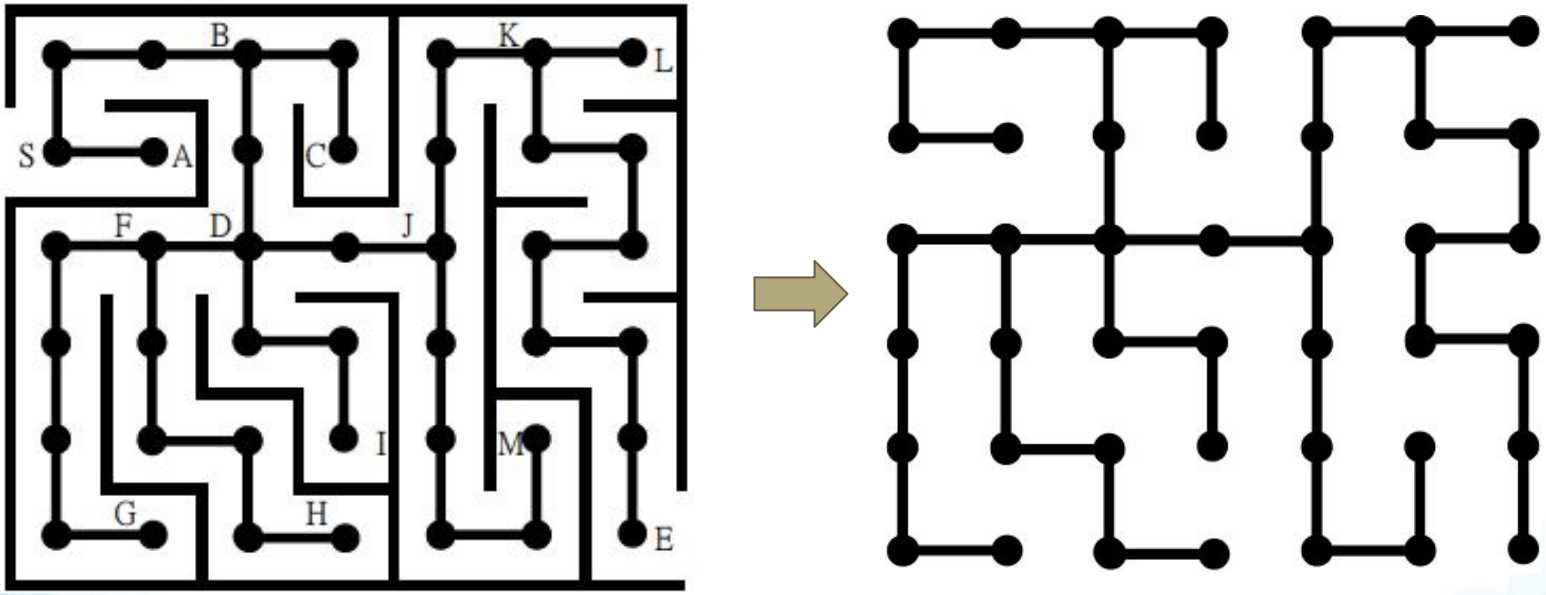


Find the node

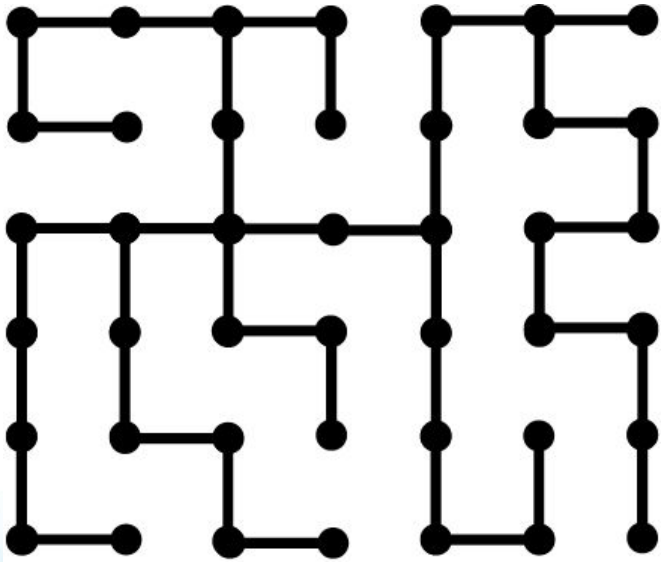
- First step is to find out the node of this maze, there are 42 nodes in this maze.

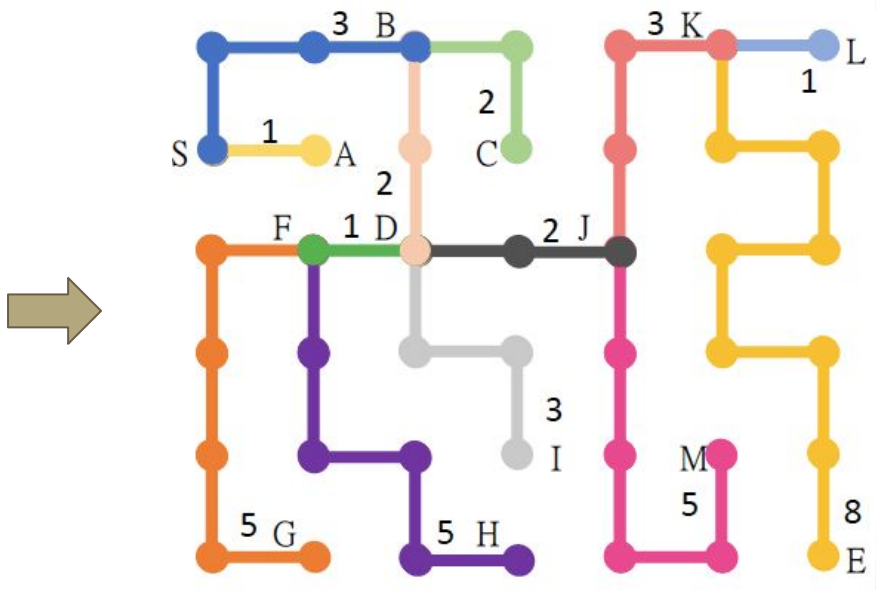


Simple the maze into graphs

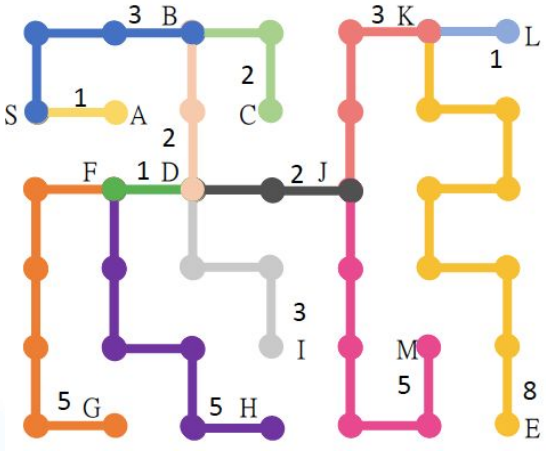


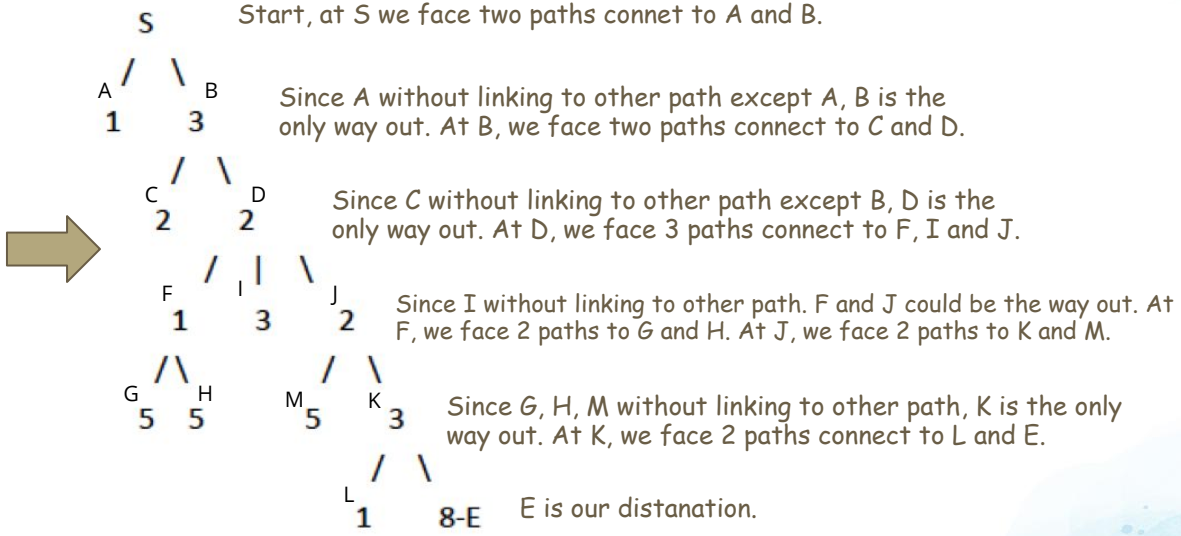
Find out the distance





Create the tree





Start, at S we face two paths connet to A and B.

Since A without linking to other path except A, B is the only way out. At B, we face two paths connect to C and D.

Since C without linking to other path except B, D is the only way out. At D, we face 3 paths connect to F, I and J.

Since I without linking to other path. F and J could be the way out. At F, we face 2 paths to G and H. At J, we face 2 paths to K and M.

Since G, H, M without linking to other path, K is the only way out. At K, we face 2 paths connect to L and E.

E is our distanation.

Dijkstra's Algorithm

- Dijkstra's algorithm - an algorithm for finding the shortest paths. It solves the single-source problem on a directed weighted graph G = (V, E), where all the edges are non-negative.

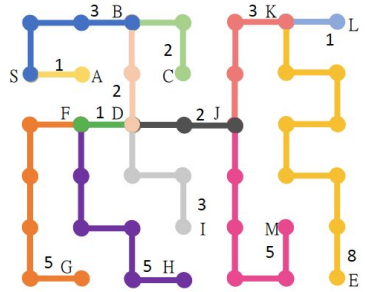
- The complexity of this algorithm is O(V2 + E).

Dijkstra's Algorithm Table

2.

	Initial	Step1	Step2	Step3	Step4	Step5	Step6	Step7	Step8	Step9	Step10	Step11	Step12	Step13
		S	A	B	C	D	F	J	I	K	G	H	L	M
1.	Next	Next	Next	Next	Next	Next	Next	Next	Next	Next	Next	Next	Next	Next
	S	A	B	C	D	F	J	I	K	G	H	L	M	E
S	0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	∞	1	1	1	1	1	1	1	1	1	1	1	1	1
B	∞	3	3	3	3	3	3	3	3	3	3	3	3	3
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	∞	∞	∞	∞	∞	6	6	6	6	6	6	6	6	6
G	∞	∞	∞	∞	∞	∞	11	11	11	11	11	11	11	11
H	∞	∞	∞	∞	∞	∞	11	11	11	11	11	11	11	11
I	∞	∞	∞	∞	∞	8	8	8	8	8	8	8	8	8
J	∞	∞	∞	∞	∞	7	7	7	7	7	7	7	7	7
K	∞	∞	∞	∞	∞	∞	∞	10	10	10	10	10	10	10
L	∞	∞	∞	∞	∞	∞	∞	∞	∞	11	11	11	11	11
M	∞	∞	∞	∞	∞	∞	∞	12	12	12	12	12	12	12
E	∞	∞	∞	∞	∞	∞	∞	∞	∞	18	18	18	18	18

3.



Rule 1. The total distance is the distance from start to current stop add distance from current stop to target stop, and it needs to be compared to the distance it already had which one is smaller.

Rule 2. Choose the smallest distance spot as the next stop.

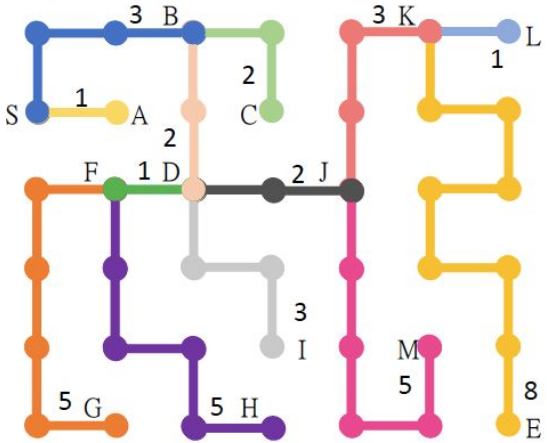
Rule 3. Next stop can't be the one which have been choosed.

1. Start from S, therefore, distance from S to S 0.
2. There are 2 paths connect to S, pass one node to A or pass three nodes to B, choose the nearest one, then A is the next stop.
3. From B, there are 2 paths to go, 2 nodes to C and 2 nodes to D,from S to B is 3 nodes, therefore from S to C or D is 3+2=5 nodes.

Shortest path to the exit by Dijkstra's Algorithm

	Initial	Step1	Step2	Step3	Step4	Step5	Step6	Step7	Step8	Step9	Step10	Step11	Step12	Step13
		S	A	B	C	D	F	J	I	K	G	H	L	M
	Next	Next	Next	Next	Next	Next	Next	Next	Next	Next	Next	Next	Next	Next
	S	A	B	C	D	F	J	I	K	G	H	L	M	E
S	0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	∞	1	1	1	1	1	1	1	1	1	1	1	1	1
B	∞	3	3	3	3	3	3	3	3	3	3	3	3	3
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	∞	∞	∞	∞	∞	6	6	6	6	6	6	6	6	6
G	∞	∞	∞	∞	∞	∞	11	11	11	11	11	11	11	11
H	∞	∞	∞	∞	∞	∞	11	11	11	11	11	11	11	11
I	∞	∞	∞	∞	∞	8	8	8	8	8	8	8	8	8
J	∞	∞	∞	∞	∞	7	7	7	7	7	7	7	7	7
K	∞	∞	∞	∞	∞	∞	∞	10	10	10	10	10	10	10
L	∞	∞	∞	∞	∞	∞	∞	∞	∞	11	11	11	11	11
M	∞	∞	∞	∞	∞	∞	∞	12	12	12	12	12	12	12
E	∞	∞	∞	∞	∞	∞	∞	∞	∞	18	18	18	18	18

- Calculate by Dijkstra's Algorithm, the smallest path from S to E needs to pass 18 nodes.



Bellman-Ford Algorithm

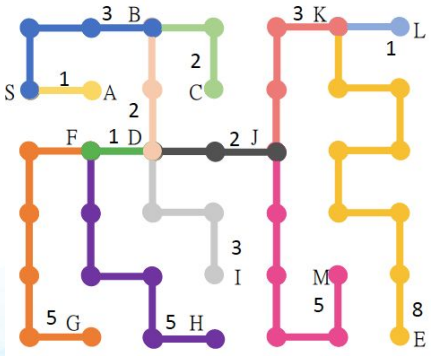
- This algorithm is also an algorithm for finding shortest paths from single source, it' graph G = (V, E) in which the edge weights may be negative.

- The first for loop is used for initialization, therefore, it runs in O(V) times.
- The next for loop runs |V - 1| passes over the edges, which takes O(E) times.
  - Hence, Bellman-Ford algorithm runs in O(V, E) time.

Bellman-Ford Algorithm Table

1. Since 0+1 < ∞, A's value changed to 1.

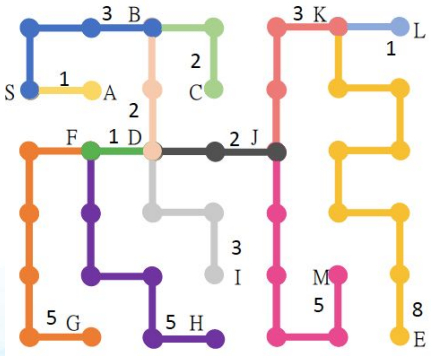
Cycle 1



	S	A	B	C	D	F	G	H	I	J	K	L	M	E
	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
Step 1	0	1	3	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
Step 2	0	1	3	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
Step 3	0	1	3	5	5	∞	∞	∞	∞	∞	∞	∞	∞	∞
Step 4	0	1	3	5	5	∞	∞	∞	∞	∞	∞	∞	∞	∞
Step 5	0	1	3	5	5	6	∞	∞	8	7	∞	∞	∞	∞
Step 6	0	1	3	5	5	6	11	11	8	7	∞	∞	∞	∞
Step 7	0	1	3	5	5	6	11	11	8	7	∞	∞	∞	∞
Step 8	0	1	3	5	5	6	11	11	8	7	∞	∞	∞	∞
Step 9	0	1	3	5	5	6	11	11	8	7	∞	∞	∞	∞
Step 10	0	1	3	5	5	6	11	11	8	7	10	∞	12	∞
Step 11	0	1	3	5	5	6	11	11	8	7	10	11	12	18
Step 12	0	1	3	5	5	6	11	11	8	7	10	11	12	18
Step 13	0	1	3	5	5	6	11	11	8	7	10	11	12	18
Step 14	0	1	3	5	5	6	11	11	8	7	10	11	12	18

Bellman-Ford Algorithm Table-2

Cycle 2



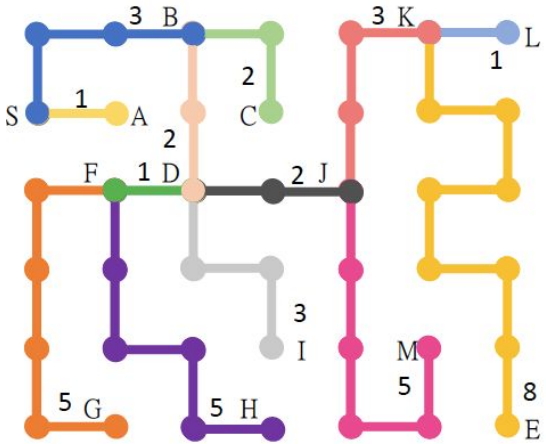
	S	A	B	C	D	F	G	H	I	J	K	L	M	E
	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
Step 15	0	1	3	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
Step 16	0	1	3	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
Step 17	0	1	3	5	5	∞	∞	∞	∞	∞	∞	∞	∞	∞
Step 18	0	1	3	5	5	∞	∞	∞	∞	∞	∞	∞	∞	∞
Step 19	0	1	3	5	5	6	∞	∞	8	7	∞	∞	∞	∞
Step 20	0	1	3	5	5	6	11	11	8	7	∞	∞	∞	∞
Step 21	0	1	3	5	5	6	11	11	8	7	∞	∞	∞	∞

	S	A	B	C	D	F	G	H	I	J	K	L	M	E
Step 22	0	1	3	5	5	6	11	11	8	7	∞	∞	∞	∞
Step 23	0	1	3	5	5	6	11	11	8	7	∞	∞	∞	∞
Step 24	0	1	3	5	5	6	11	11	8	7	10	∞	12	∞
Step 25	0	1	3	5	5	6	11	11	8	7	10	11	12	18
Step 26	0	1	3	5	5	6	11	11	8	7	10	11	12	18
Step 27	0	1	3	5	5	6	11	11	8	7	10	11	12	18
Step 28	0	1	3	5	5	6	11	11	8	7	10	11	12	18

Shortest path to the exit by Bellman-Ford Algorithm

Step 22	0	1	3	5	5	6	11	11	8	7	∞	∞	∞	∞
	S	A	B	C	D	F	G	H	I	J	K	L	M	E
Step 23	0	1	3	5	5	6	11	11	8	7	∞	∞	∞	∞
	S	A	B	C	D	F	G	H	I	J	K	L	M	E
Step 24	0	1	3	5	5	6	11	11	8	7	10	∞	12	∞
	S	A	B	C	D	F	G	H	I	J	K	L	M	E
Step 25	0	1	3	5	5	6	11	11	8	7	10	11	12	18
	S	A	B	C	D	F	G	H	I	J	K	L	M	E
Step 26	0	1	3	5	5	6	11	11	8	7	10	11	12	18
	S	A	B	C	D	F	G	H	I	J	K	L	M	E
Step 27	0	1	3	5	5	6	11	11	8	7	10	11	12	18
	S	A	B	C	D	F	G	H	I	J	K	L	M	E
Step 28	0	1	3	5	5	6	11	11	8	7	10	11	12	18
	S	A	B	C	D	F	G	H	I	J	K	L	M	E

- The process ends at Cycle 2 because none of the vertices is changed.
- Calculate by Bellman-Ford Algorithm, the smallest path from S to E needs to pass 18 nodes.



Spanning Tree

- A spanning tree is a Graph that has all the vertices(nodes) connected by edges and does not have any cycle.
- A Minimum Spanning Tree (MST) is a 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.

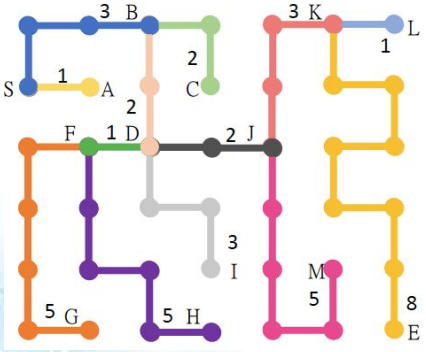
Prim's Minimun Spanning Tree (MST)

Assign a key value(weight) to all vertices in the input graph.

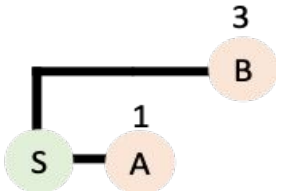
1. Initialize all key values as INFINITE.
2. Assign key value as 0 for the first vertex so that it is picked first.

While picked-set doesn't include all vertices

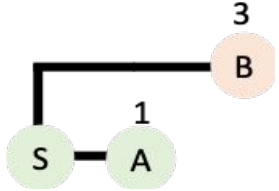
1. Pick a vertex which is not in set and has minimum key value.
2. Update key value of all of picked-node's adjacent vertices which are not in set.



Step 1:



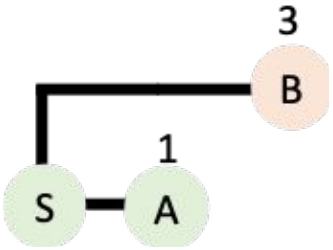
Step 2:



Start from S, the tree can link to two nodes, choose the one with smallest weight (which is A) to be the next node, and then go to Step 2.

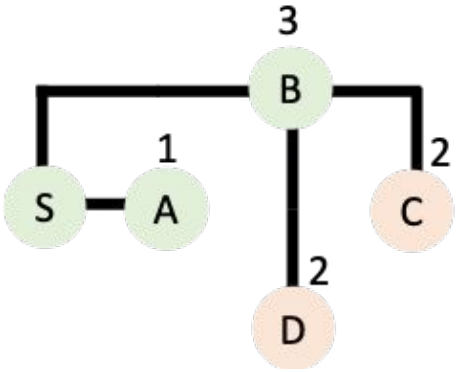
Prim's Minimun Spanning Tree Graph -1

Step 2:

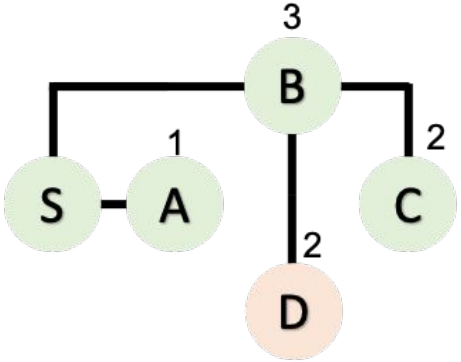


Since A doesn't connect to other nodes, find the smallest node from the nodes haven't chosen which is B to Step 3.

Step 3:



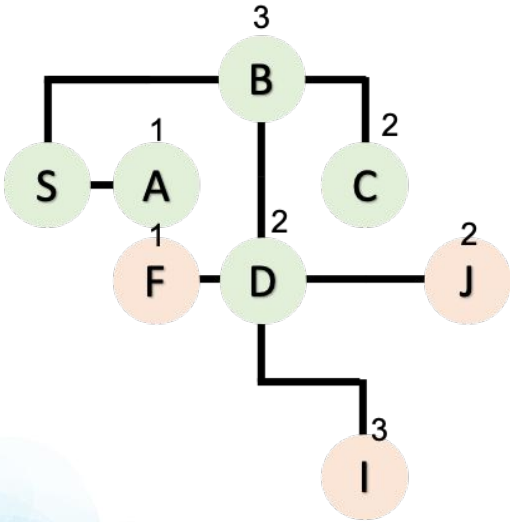
Step 4:



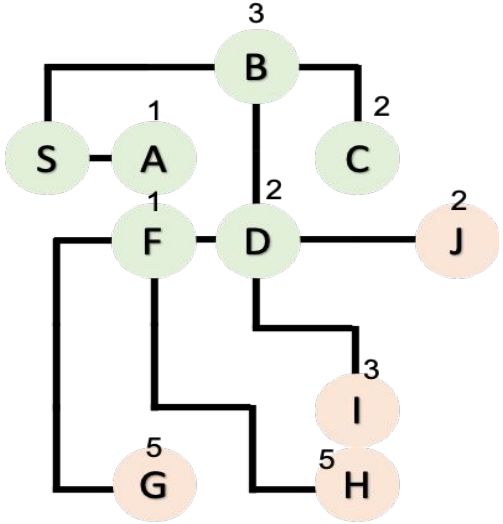
Repeat the rule like Step 1, 2 keep choose the smallest nodes.

Prim's Minimun Spanning Tree Graph -2

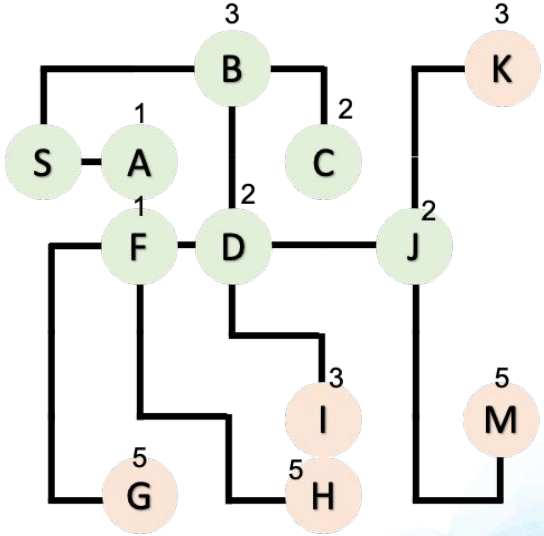
Step 5:



Step 6:

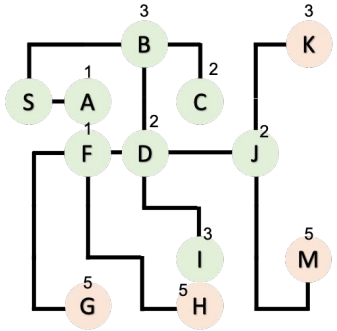


Step 7:

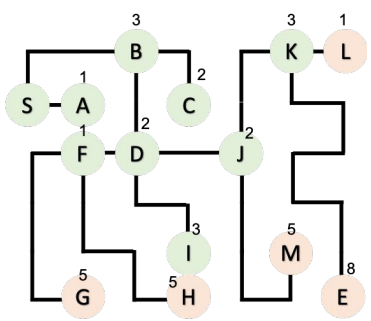


Prim's Minimun Spanning Tree Graph -3

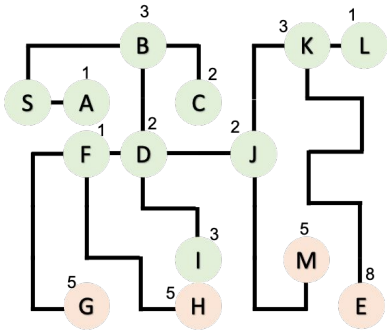
Step 8:



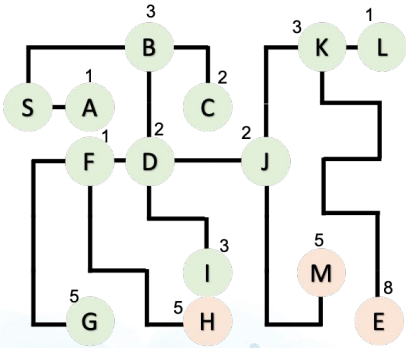
Step 9:



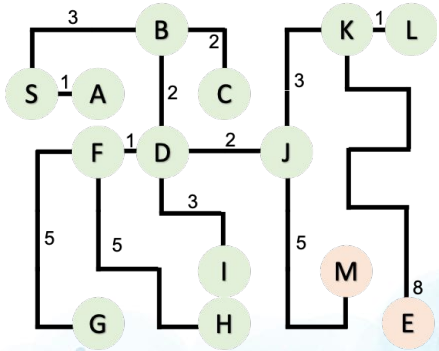
Step 10:



Step 11:

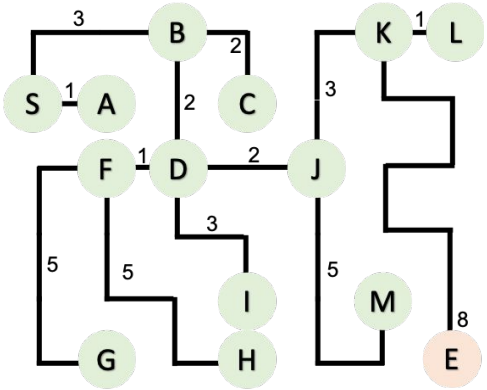


Step 12:

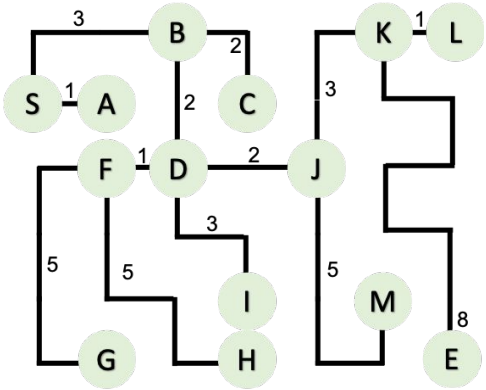


Prim's Minimun Spanning Tree Graph -4

Step 13:



Step 14:

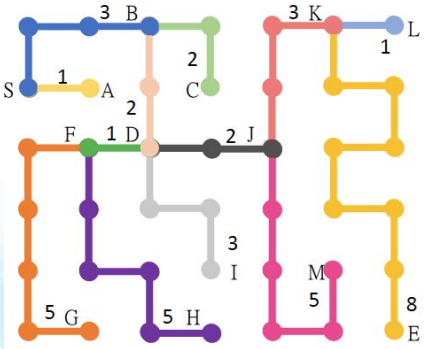


Process stop since edges = vertices-1

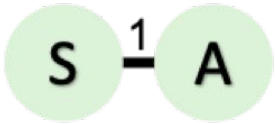
The time complexity of Prim's MST = O((v+E)logV).

Kruskal's Minimum Spanning Tree

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



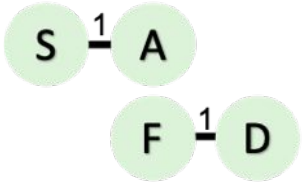
The smallest weight is 1 which edge is S-A, and then step 1 is like:



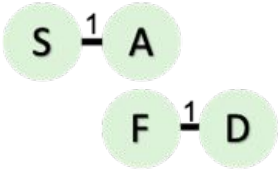
After sorting		
Weight	Src	Dest
1	S	A
1	D	F
1	K	L
2	B	C
2	B	D
2	D	J
3	S	B
3	D	I
3	J	K
5	F	G
5	F	H
5	J	M
8	K	E

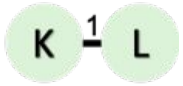
Kruskal's Minimum Spanning Tree Graph-1

Step 2:

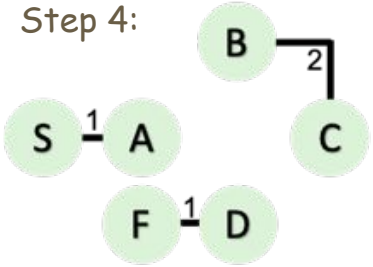


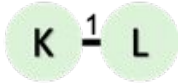
Step 3:



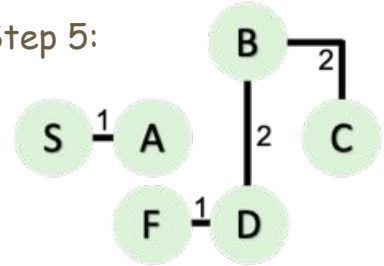


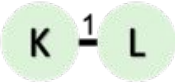
Step 4:





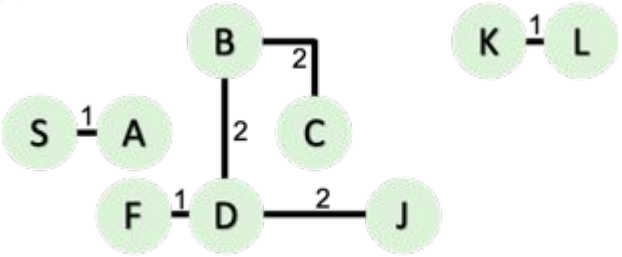
Step 5:



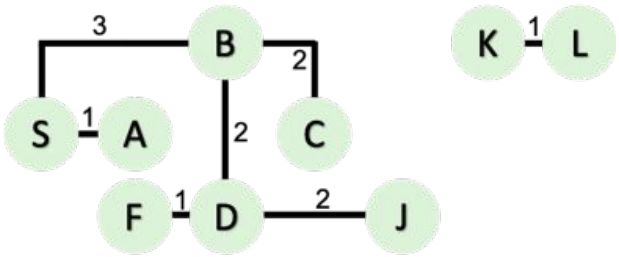


Kruskal's Minimum Spanning Tree Graph-2

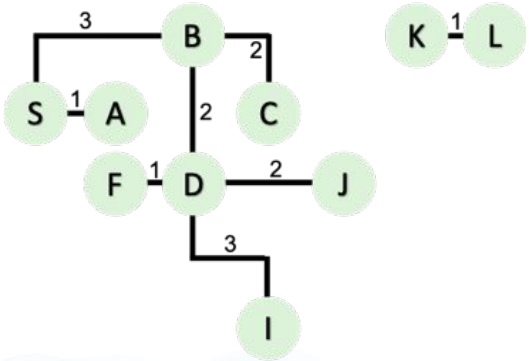
Step 6:



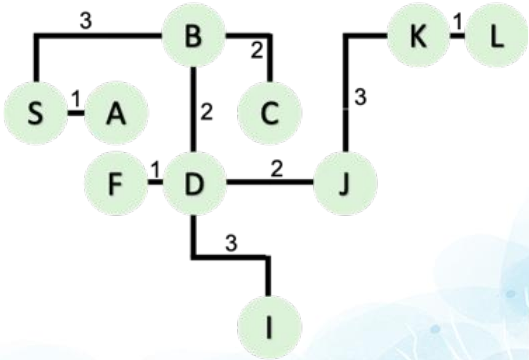
Step 7:



Step 8:

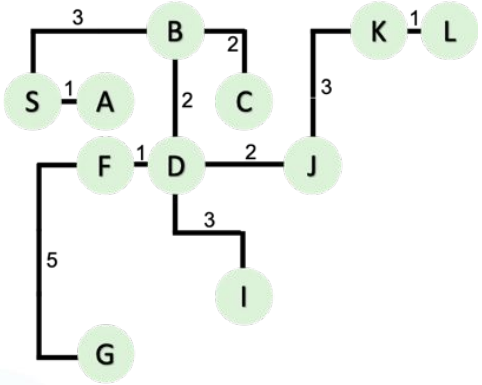


Step 9:

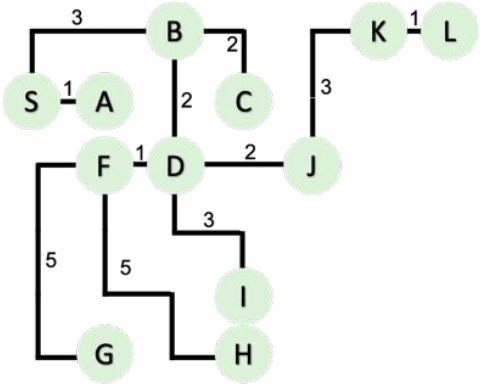


Kruskal's Minimum Spanning Tree Graph-3

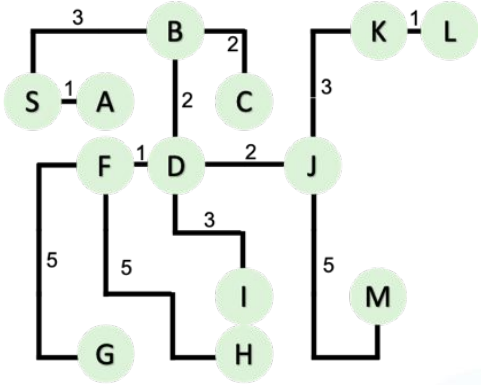
Step 10:



Step 11:

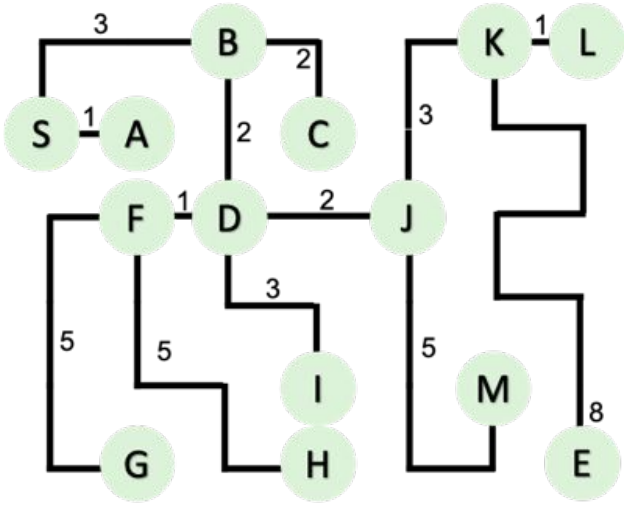


Step 12:



Kruskal's Minimum Spanning Tree Graph-4

Step 13:



Process stop since edges = vertices-1

The time complexity of Kruskal's MST = O(E*logV).

Thank you!