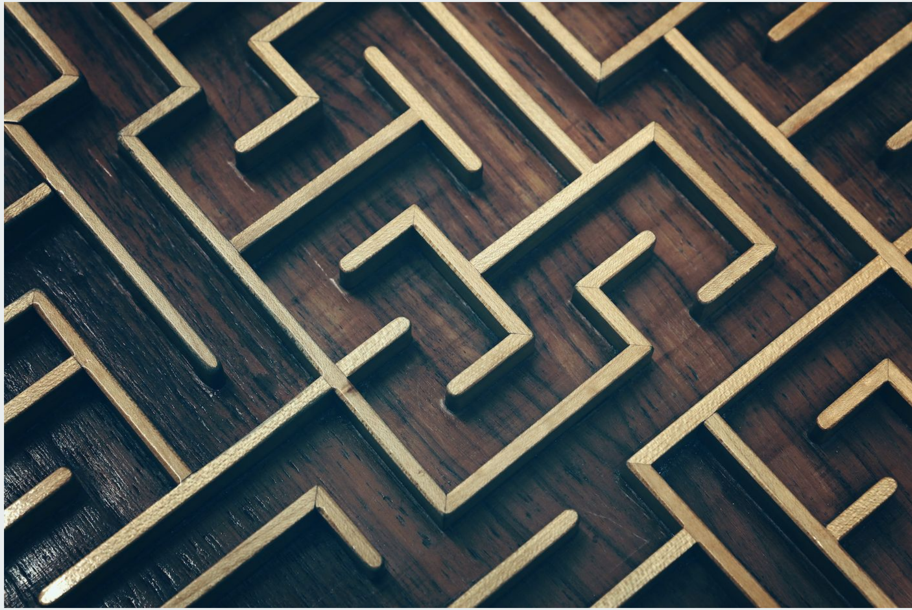
Maze Project



Dan Liu
March 2021

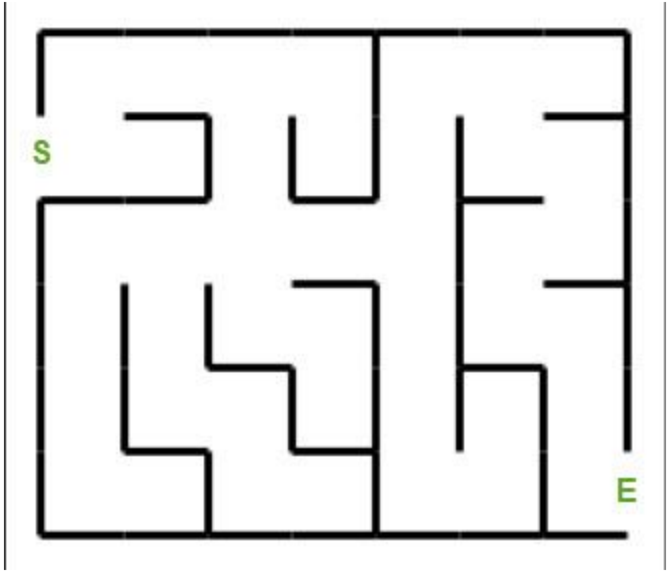
Subjects

Introduction

Challenge:

1.How to find the shortest path of the maze?

2. How to find the minimum spanning tree of the maze?



Design

Challenge 1: Shortest path

Two ways to solve single-source shortest path problem:

1. Dijkstra's Algorithm

   On a directed weighted graph *G = (V, E)*, where all the edges are non-negative.

   The time complexity of **Dijkstra's algorithm** dependents on the implementation of extract-minimum. The simplest version stores vertices as array or linked list, and use a linear search through all vertices. The running time is $\mathbf{O(|E| + |V|^2)}$, |E| is the number of edges and |V| is the number of vertices. This algorithm can be implemented more efficiently using min-heap to implement extract- minimum function.

Challenge 1: Shortest path

Two ways to solve single-source shortest path problem:

2. Bellman Ford's Algorithm

   On a directed graph *G* = *(V, E)*, where the edge weights may be negative.

   Bellman-Ford's algorithm relaxes all the edges and runs |V - 1| times, the time complexity of this algorithm is **O(|V| . |E|)**. |E| is the number of edges and |V| is the number of vertices.

Challenge 2: Minimum Spanning Tree

Definition: MST is a subset of edges of a connected weighted undirected graph which connects all the vertices with the minimum possible total edge weight.

Two ways can be used to find a MST:

1. ## Prim's Algorithm

   The time complexity is O((V + E) * logV), V is the number of vertices. Each vertex is inserted in the priority queue only once, the time spent on insertion is logarithmic time. This algorithm runs faster in dense graphs.

2. ## Kruskal's Algorithm

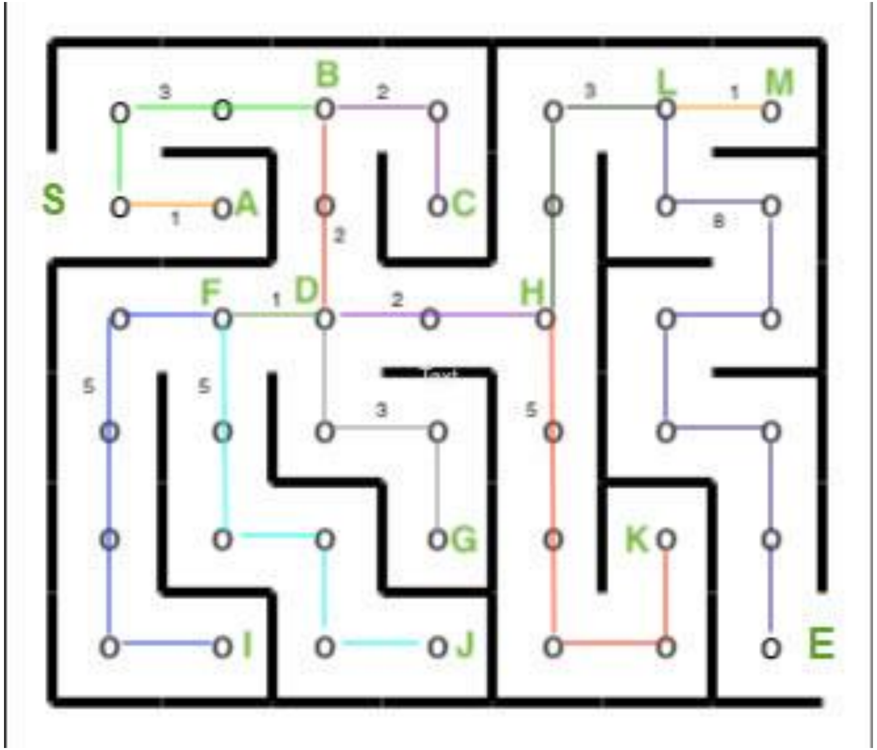
   The time complexity is O(E * logV), V is the number of vertices. Most time is consumed on sorting. This algorithm runs faster in sparse graphs.

Implementation

Challenge 1: Shortest path

Dijkstra's Algorithm

Step 1:
Draw the maze route, mark the number of edges on each route and label nodes with alphabet.

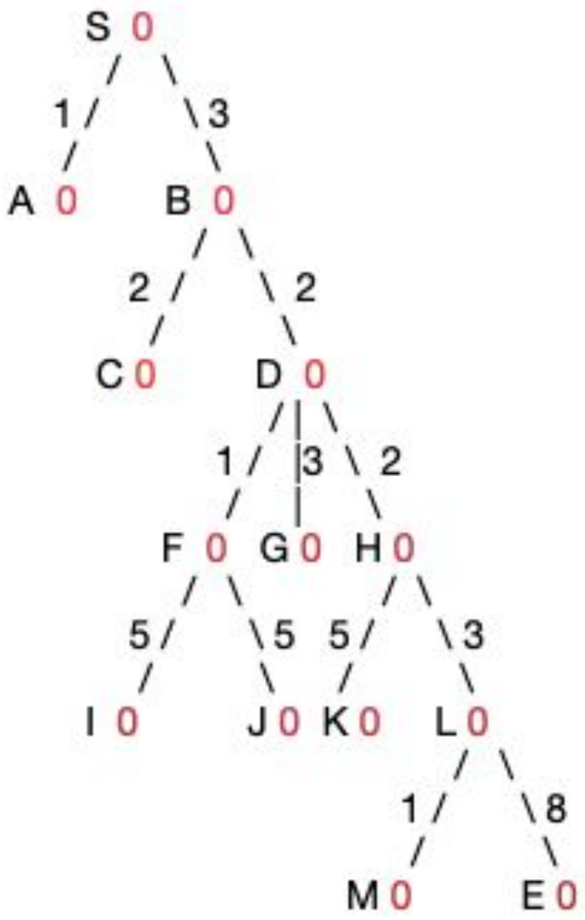


Challenge 1: Shortest path

Dijkstra's Algorithm

Step 2:

1. Draw the tree representation of the maze;
2. Label each node of the tree sequentially with alphabets;
3. Mark each edge with a number indicating the distance.



Challenge 1: Shortest path

Dijkstra's Algorithm

Step 3:
Using Dijkstra's Algorithm to find the minimum distance of E from S.

The shortest path from S to E is:
S -> B -> D -> H -> L -> E

Total distance is 18.

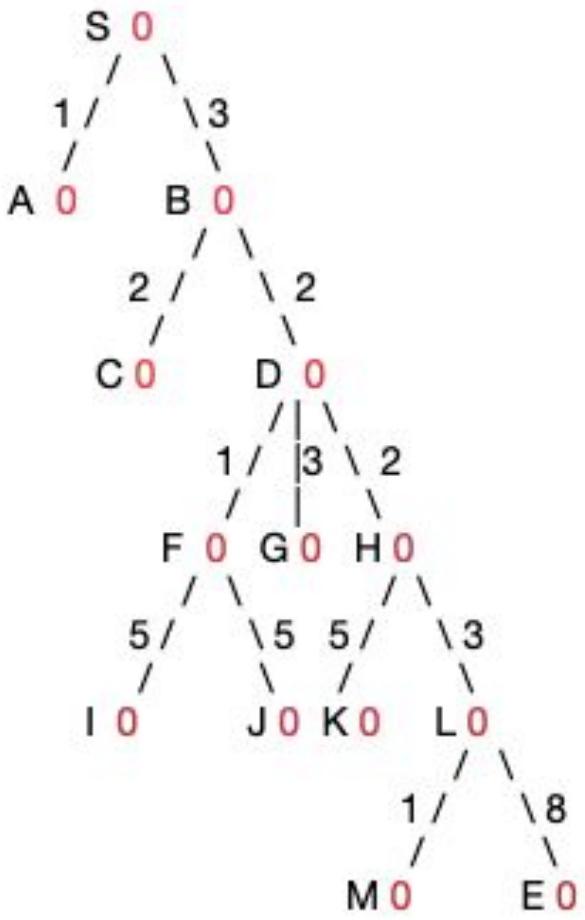
Vertex	Initial	Step 1	Step 2	Step 3	Step 4	Step 5	Step 6	Step 7	Step 8	Step 9	Step 10	Step 11	Step 12	Step 13
		S	A	B	C	D	F	H	G	L	I	J	M	K
	Next	Next	Next	Next	Next	Next	Next	Next	Next	Next	Next	Next	Next	End
	S	A	B	C	D	F	H	G	L	I	J	M	K	E
S	0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	∞	1	1	1	1	1	1	1	1	1	1	1	1	1
B	∞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

Challenge 1: Shortest path

Bellman Ford's Algorithm

Step 1:

1. Use the same tree representation of the maze;
2. Get started: 14 vertices = 13 iterations



Challenge 1: Shortest path

Bellman Ford's Algorithm

Step 2:

1. Iterate every vertice in the tree in each cycle;
2. The process ends at cycle 2 when none of the vertices is changed.

Cycle 1:

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

Cycle 2:

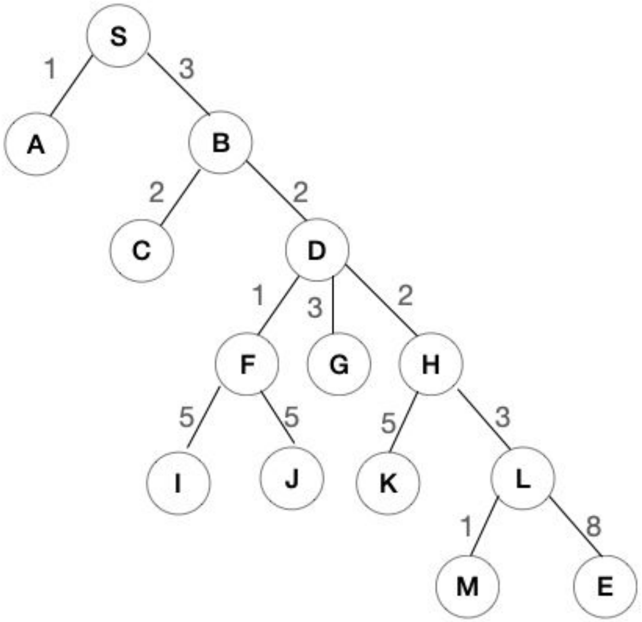
Current node	S	A	B	C	D	F	G	H	I	J	K	L	M	E
S	0	1	3	5	5	6	8	7	11	11	12	10	11	18
A	0	1	3	5	5	6	8	7	11	11	12	10	11	18
B	0	1	3	5	5	6	8	7	11	11	12	10	11	18
C	0	1	3	5	5	6	8	7	11	11	12	10	11	18
D	0	1	3	5	5	6	8	7	11	11	12	10	11	18
F	0	1	3	5	5	6	8	7	11	11	12	10	11	18
G	0	1	3	5	5	6	8	7	11	11	12	10	11	18
H	0	1	3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

Challenge 2: Minimum Spanning Tree

Prim's Algorithm

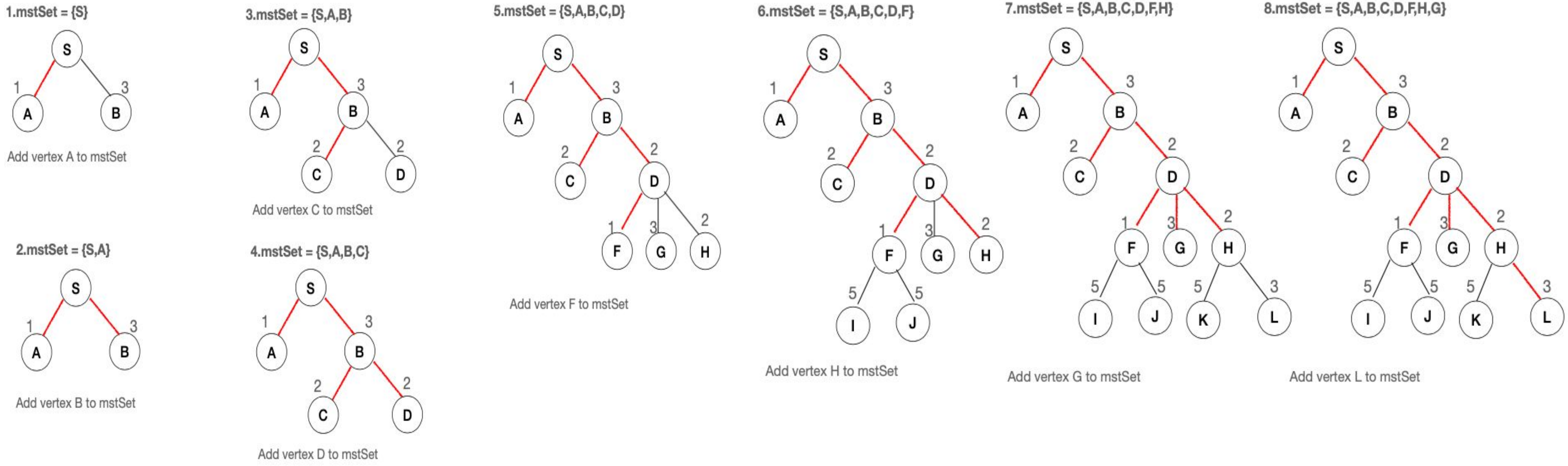
Steps:

1. Create a mstSet to keep track of vertices included in MST;
2. Assign a key value to all vertices in the graph and initialize them as INFINITE and pick the first vertex;
3. While mstSet doesn't include all vertices:
   Pick a minimum key value of vertex v which not exists in mstSet;
   Include vertex v to mstSet;
   Update the value of v's adjacent vertices which are not in mstSet.



Challenge 2: Minimum Spanning Tree

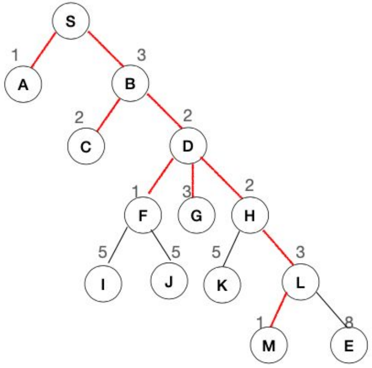
Prim's Algorithm



Challenge 2: Minimum Spanning Tree

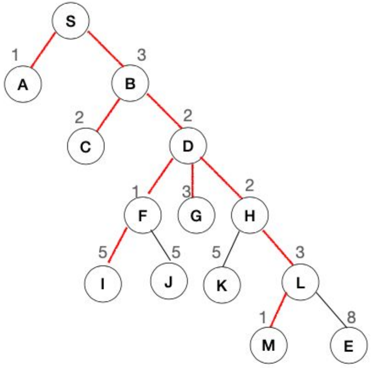
Prim's Algorithm

9.mstSet = {S,A,B,C,D,F,H,G,L}



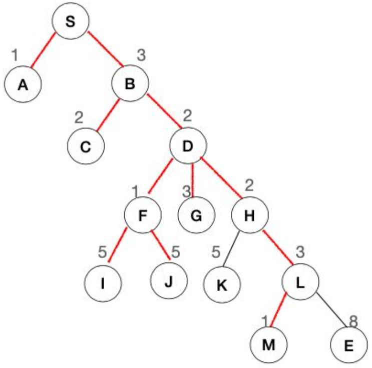
Add vertex M to mstSet

10.mstSet = {S,A,B,C,D,F,H,G,L,M}



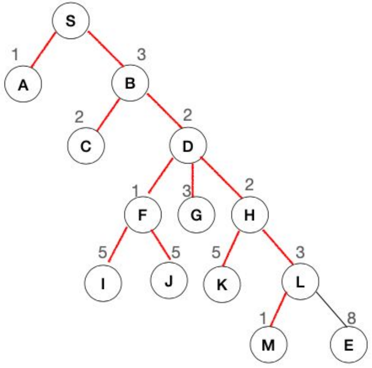
Add vertex I to mstSet

11.mstSet = {S,A,B,C,D,F,H,G,L,M,I}



Add vertex J to mstSet

12.mstSet = {S,A,B,C,D,F,H,G,L,M,I,J}

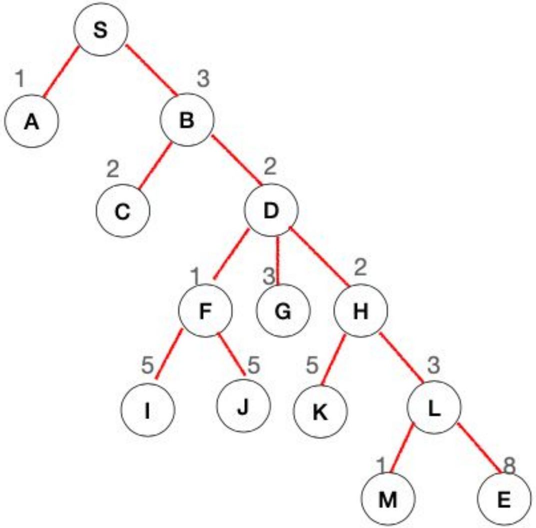


Add vertex K to mstSet

Challenge 2: Minimum Spanning Tree

Prim's Algorithm

13.mstSet = {S,A,B,C,D,F,H,G,L,M,I,J,K}



Add vertex E to mstSet

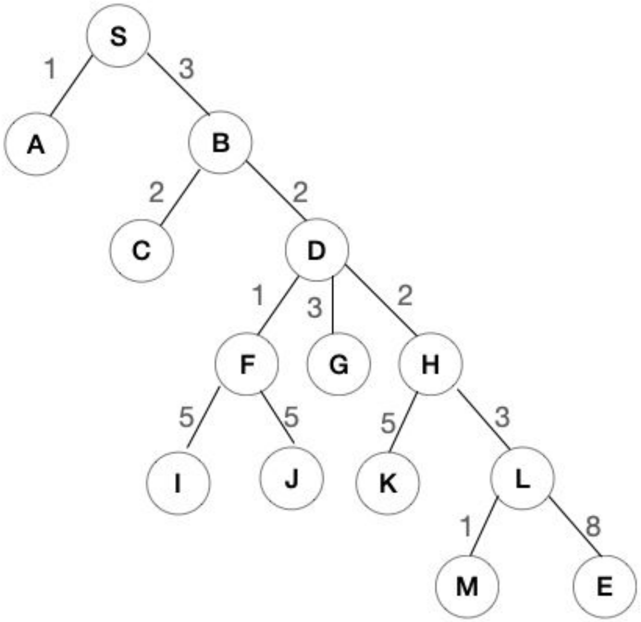
14.mstSet = {S,A,B,C,D,F,H,G,L,M,I,J,K,E}

Challenge 2: Minimum Spanning Tree

Kruskal's Algorithm

Steps:

1. Sort all the edges on their weight in non-decreasing order;
2. Pick the smallest edge. Check if it forms a cycle so far. If cycle is not formed, include the edge. Otherwise, discard it;
3. Repeat step 2 until find V-1 edges in the spanning tree.



Challenge 2: Minimum Spanning Tree

Kruskal's Algorithm

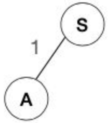
The graph contains 14 vertices and 13 edges. So minimum spanning tree formed will have 13 edges.

After sorting:

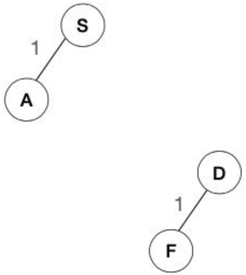
Weight	Source	Destination
1	S	A
1	D	F
1	L	M
2	B	C
2	B	D
2	D	H
3	S	B
3	D	G
3	H	L
5	F	I
5	F	J
5	H	K
8	L	E

Now pick all edges one by one from sorted edges.

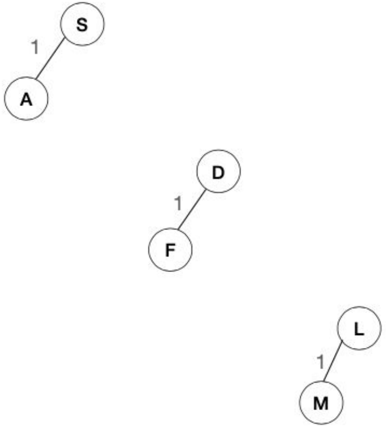
1.Pick edge S-A: No cycle is formed, include it.



2.Pick edge D-F: No cycle is formed, include it.



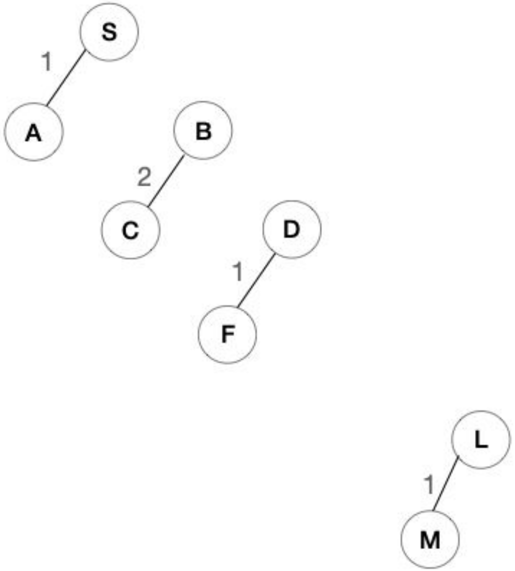
3.Pick edge L-M: No cycle is formed, include it.



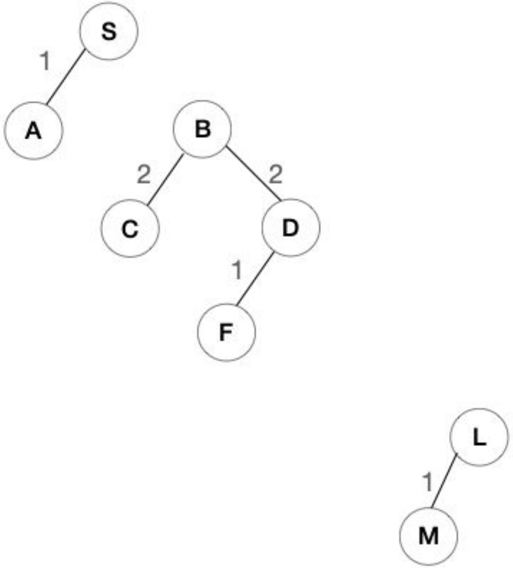
Challenge 2: Minimum Spanning Tree

Kruskal’s Algorithm

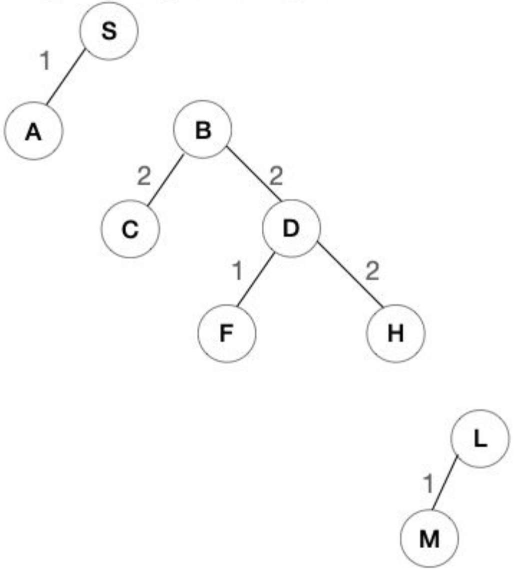
4.Pick edge B-C: No cycle is formed, include it.



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



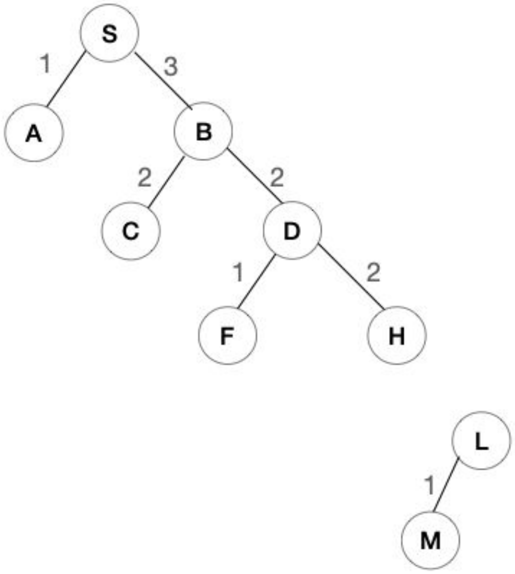
6.Pick edge D-H: No cycle is formed, include it.



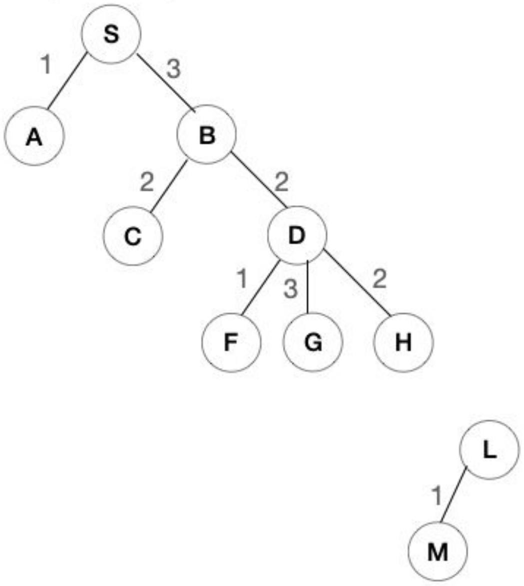
Challenge 2: Minimum Spanning Tree

Kruskal's Algorithm

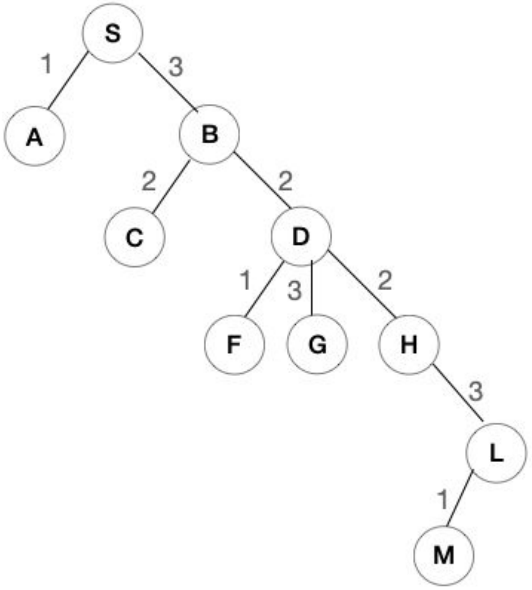
7.Pick edge S-B: No cycle is formed, include it.



8.Pick edge D-G: No cycle is formed, include it.



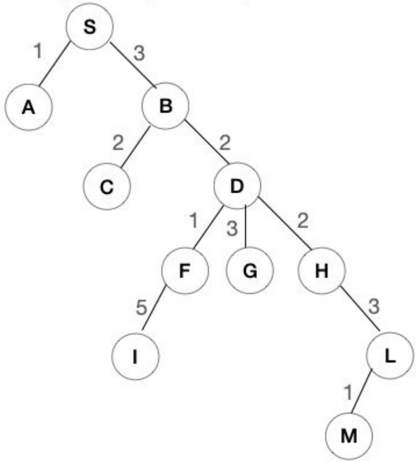
9.Pick edge H-L: No cycle is formed, include it.



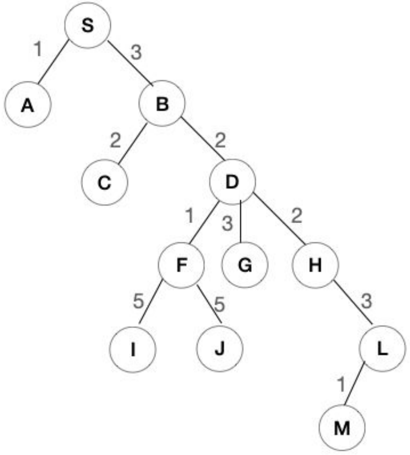
Challenge 2: Minimum Spanning Tree

Kruskal's Algorithm

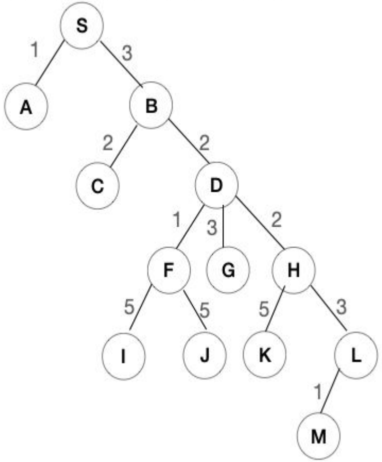
10.Pick edge F-I: No cycle is formed, include it.



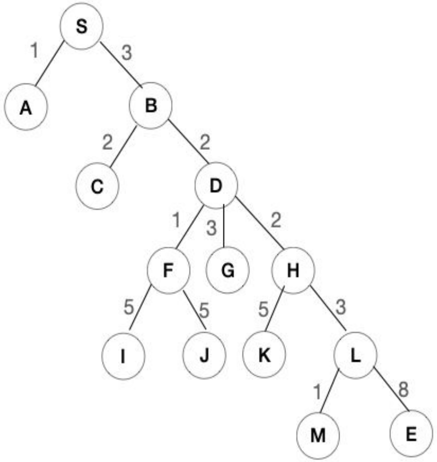
11.Pick edge F-J: No cycle is formed, include it.



12.Pick edge H-K: No cycle is formed, include it.



13.Pick edge L-E: No cycle is formed, include it.



Since the number of edges equals to 13 and all vertices have been visited, the algorithm stops.

Enhancement Ideas

Bellman Ford's Algorithm can be used to find shortest paths when there is no any negative weighted cycle. Besides, it can be applied in some applications since it may have negative edges. For example:

1. Imagine a logistic network where the weight w(i, j) of an edge i, j is the cost from vertex i to vertex j. In the situation of a business transportation with other companies, w(i, j) is a profit, you can interpret the weight as a negative cost.
2. Represent the speed driving from one place to another. The speed above average is positive, while below average is negative.

Conclusion

Shortest path

1. Dijkstra's Algorithm and Bellman Ford's Algorithm both can be used to find shortest paths.
2. The time complexity of Dijkstra's Algorithm dependents on the implementation of extract-minimum. The running time of simplest version is $O(|E| + |V|2)$.
   The time complexity of Bellman Ford's Algorithm is $O(|V| . |E|)$.
3. Bellman Ford Algorithm can have other applications because of its property of negative edge.

Minimum Spanning Tree

1. Prim's algorithm and Kruskal's algorithm both can be used to find a minimum spanning tree.
2. The time complexity of Prim's Algorithm is O((V + E) * logV), it has better performance in dense graphs.
3. The time complexity of Kruskal's Algorithm is O(E * logV), it has better performance in sparse graphs.

References

References

1. https://npu85.npu.edu/~henry/npu/classes/algorithm/graph_alg/slide/maze.html
2. https://npu85.npu.edu/~henry/npu/classes/algorithm/tutorialpoints_daa/slide/shortest_paths.html
3. https://www.cs.uah.edu/~rcoleman/CS221/Graphs/ShortestPath.html
4. https://npu85.npu.edu/~henry/npu/classes/algorithm/tutorialpoints_daa/slide/bf.html
5. https://npu85.npu.edu/~henry/npu/classes/algorithm/tutorialpoints/slide/index_slide.html