NORTHWESTERN POLYTECHNIC
UNIVERSITY

Maze Project

Prepared For:

Mr. Henry Chang
Algorithms & Structured Programming CS455
Spring 2021
Northwestern Polytechnic University

Prepared By:

Ms. Nagalla, Santhi Sree ID:19568

Table of Contents

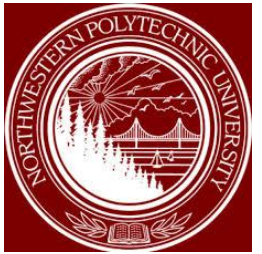
Introduction

- Maze solving is the act of finding a route through the maze from the start to finish. Some maze solving methods are designed to be used inside the maze by a traveler with no prior knowledge of the maze, whereas others are designed to be used by a person or computer program that can see the whole maze at once.

Maze-solving algorithms

- There are a number of different maze-solving algorithms, that is, automated methods for the solving of mazes.
  - ➢ Wall follower
  - ➢ Random mouse algorithm
  - ➢ Pledge algorithm
  - ➢ Trémaux's algorithm
  - ➢ Dead-end filling

Maze-solving algorithms - Continues



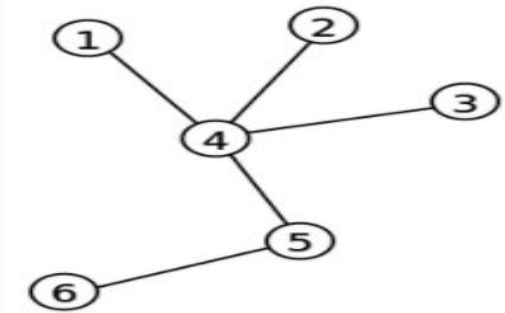
- ➢ Maze-routing algorithm
- ➢ Recursive algorithm
- ➢ Shortest path algorithm

- Mazes containing no loops are known as "simply connected", or "perfect" mazes, and are equivalent to a tree in graph theory. Thus many maze-solving algorithms are closely related to graph theory.

What is a Graph Theory?

- In graph theory, a tree is an undirected graph in which any two vertices are connected by exactly one path, or equivalently a connected acyclic undirected graph. A forest is an undirected graph in which any two vertices are connected by at most one path, or equivalently an acyclic undirected graph, or equivalently a disjoint union of trees.

Trees



A labeled tree with 6 vertices and 5 edges.

Vertices	v
Edges	$v - 1$
Chromatic number	2 if $v > 1$

Table of graphs and parameters

Shortest path algorithm

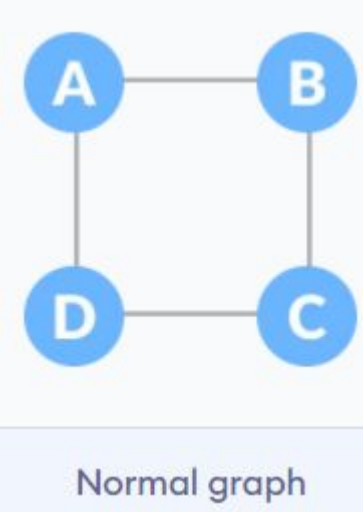
- When a maze has multiple solutions, the solver may want to find the shortest path from start to finish. One such algorithm finds the shortest path by implementing a breadth-first search, while another, the A* algorithm, uses a heuristic technique. The breadth-first search algorithm uses a queue to visit cells in increasing distance order from the start until the finish is reached.

Spanning Tree

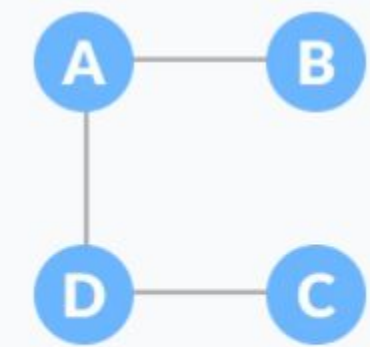
- A spanning tree is a sub-graph of an undirected connected graph, which includes all the vertices of the graph with a minimum possible number of edges. If a vertex is missed, then it is not a spanning tree.
- The edges may or may not have weights assigned to them.
- The total number of spanning trees with n vertices that can be created from a complete graph is equal to n^(n-2).

Spanning Tree - Example

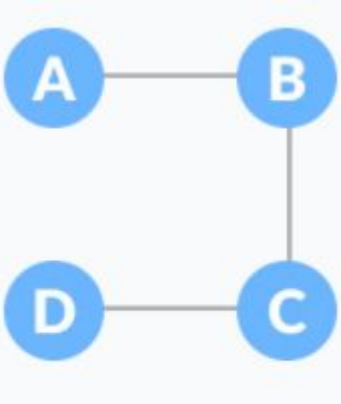
- If we have n = 4, the maximum number of possible spanning trees is equal to 4^4-2 = 16. Thus, 16 spanning trees can be formed from a complete graph with 4 vertices. Examples -



Normal graph



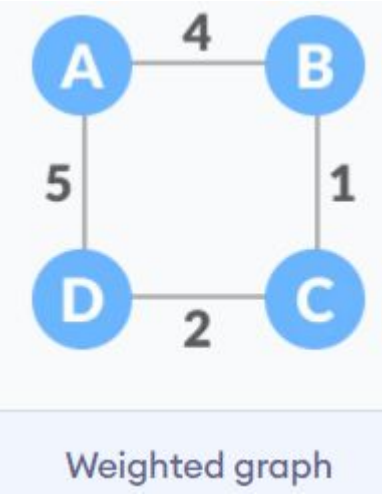
A spanning tree

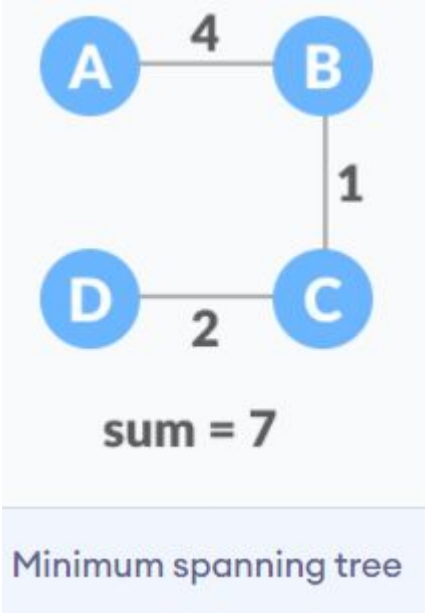


A spanning tree

Minimum Spanning Tree

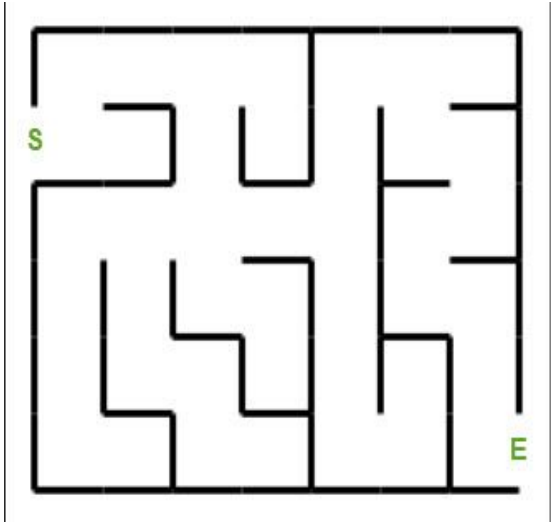
- A minimum spanning tree is a spanning tree in which the sum of the weight of the edges is as minimum as possible.





Find the minimum length spanning tree

- The weight of a tree is just the sum of weights of its edges. Obviously, different trees have different lengths. The problem: how to find the minimum length spanning tree?This problem can be solved by many different algorithms.
- In graph theory, there are below algorithms for calculating the minimum spanning tree (MST):
  - ➢ Kruskal's algorithm
  - ➢ Prim's algorithm
  - ➢ Boruvka's algorithm



Prim's Algorithm

- Prim’s start from one vertex and keep adding edges with the lowest weight until we reach our goal.

  The steps for implementing Prim's algorithm are as follows:

  - ➢ Initialize the minimum spanning tree with a vertex chosen at random.
  - ➢ Find all the edges that connect the tree to new vertices, find the minimum and add it to the tree
  - ➢ Keep repeating step 2 until we get a minimum spanning tree

Kruskal's algorithm

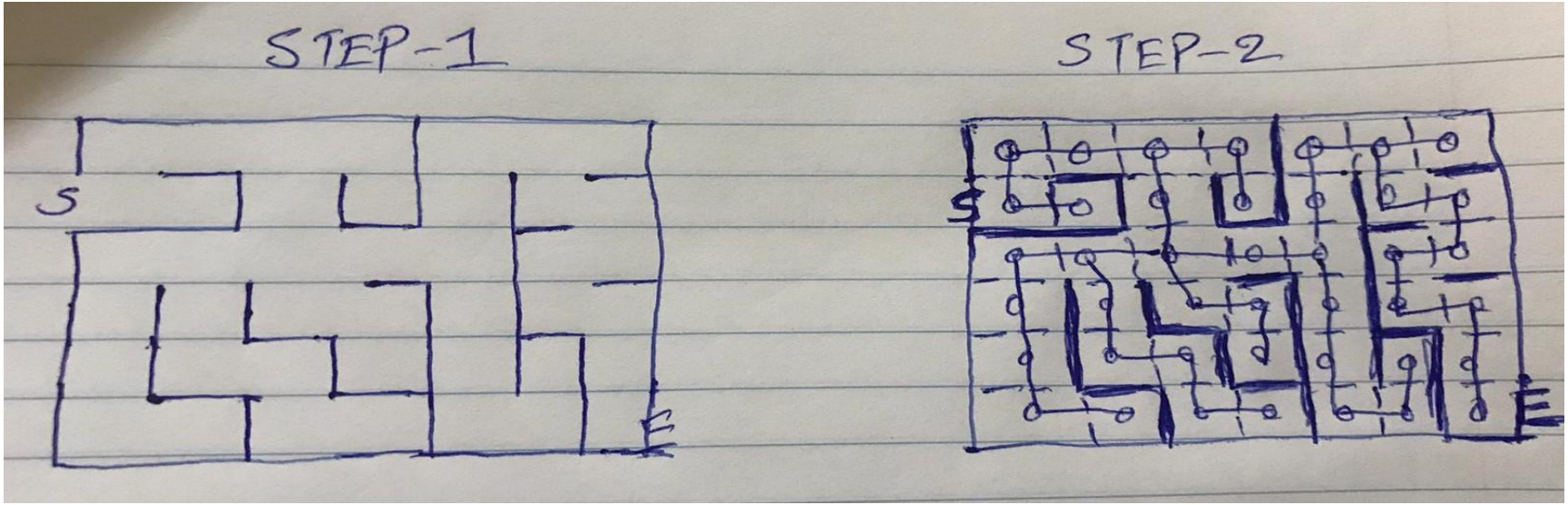
- Kruskal start from the edges with the lowest weight and keep adding edges until we reach our goal.

  The steps for implementing Kruskal's algorithm are as follows:

  - ➢ Sort all the edges from low weight to high
  - ➢ Take the edge with the lowest weight and add it to the spanning tree. If adding the edge created a cycle, then reject this edge.
  - ➢ Keep adding edges until we reach all vertices.

Convert Maze to Graph

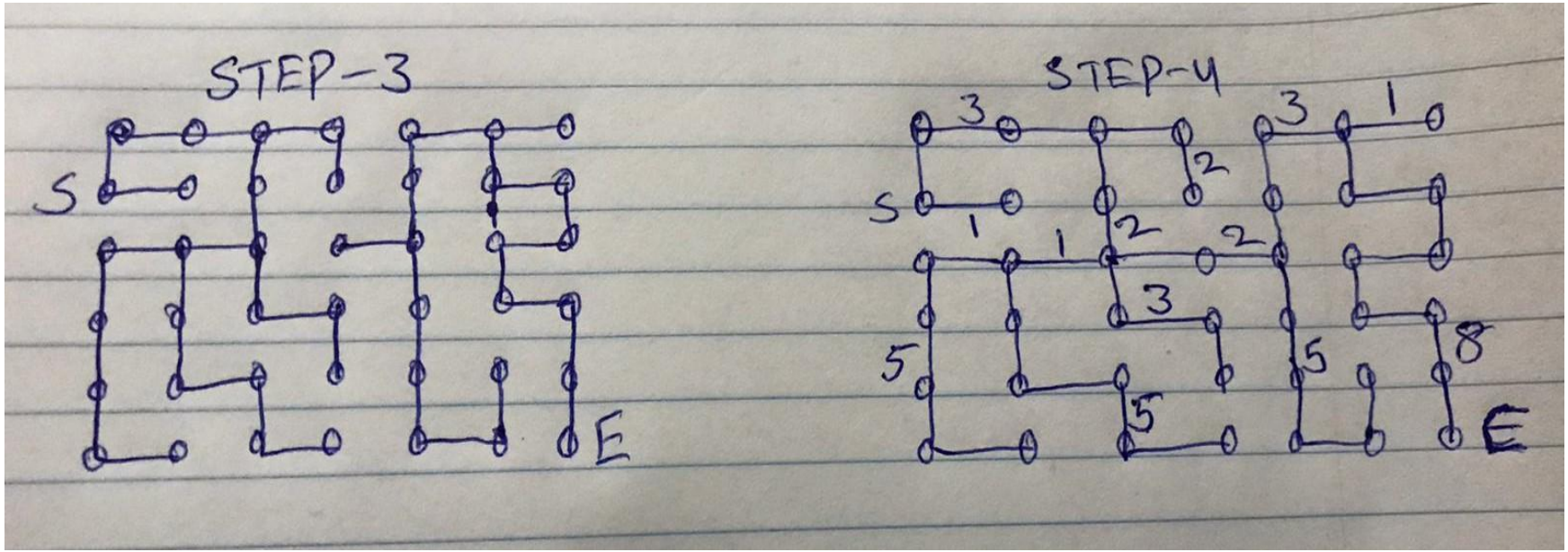
Step 1: Initialize distances according to the algorithm.



Convert Maze to graph with weights

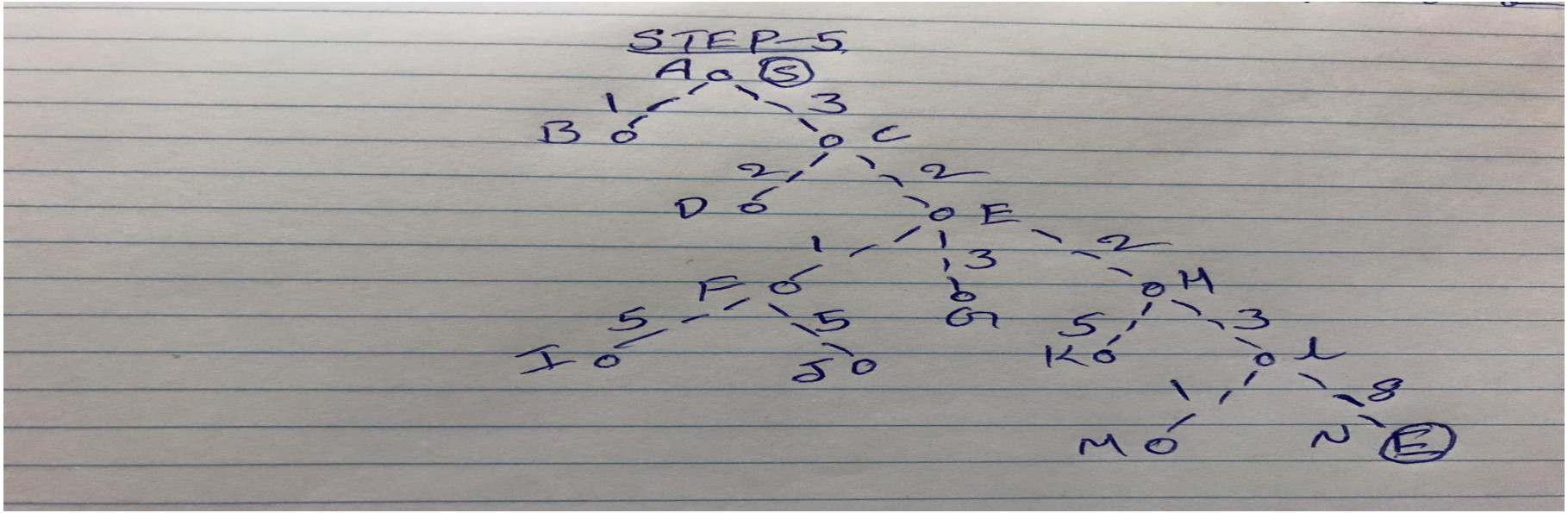
Step 2: Pick first node and calculate distances to adjacent nodes.

Step 3: Pick next node with minimal distance repeat adjacent node distance calculations.



Tree or Graph

- A tree is an undirected graph in which any two vertices are connected by exactly one path, or equivalently a connected acyclic undirected graph

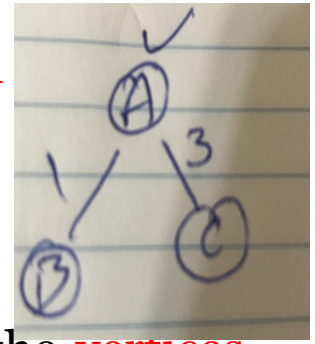


Prim's Minimum Spanning Tree (MST) - Steps

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.
    1. Initialize all key values as INFINITE.
    2. Assign key value as 0 for the first vertex so that it is picked first.
C. While mstSet doesn't include all vertices
    3. Pick a vertex u which is not there in mstSet and has minimum key value.
    4. Include u to mstSet.
    5. 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 algorithm to solve maze

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.

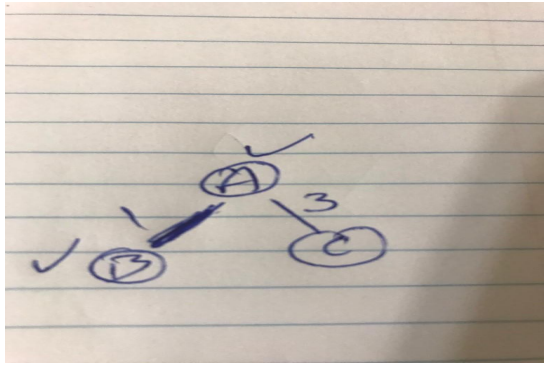


- Now pick the vertex with minimum key value. The vertex A include it in mstSet. So mstSet becomes {A}.
- After including to mstSet, update key values of adjacent vertices. Adjacent vertices of **A** are **B and C**.
- The key values of A and B are updated as 1 and 3.
- Following subgraph shows vertices and their key values, only the vertices with finite key values are shown. The vertices included in MST are shown in Tick Mark.

Steps - Continues

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

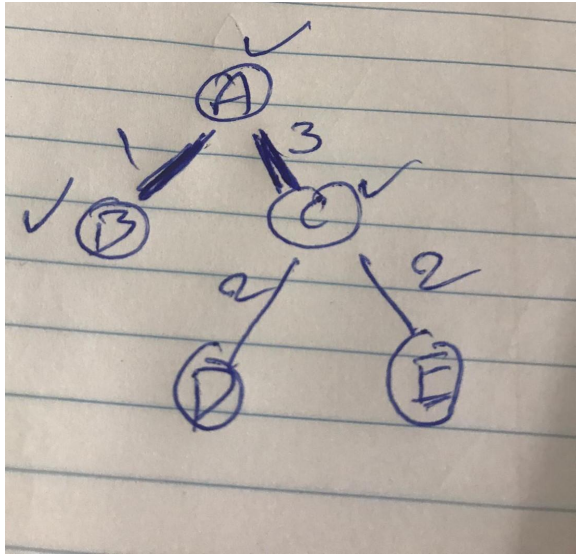
a. The vertex B is picked and added to mstSet.
b. So mstSet now becomes {A,B}.
c. Update the key values of adjacent vertices of B There are no adjacent vertices of vertex B.



Steps - Continues

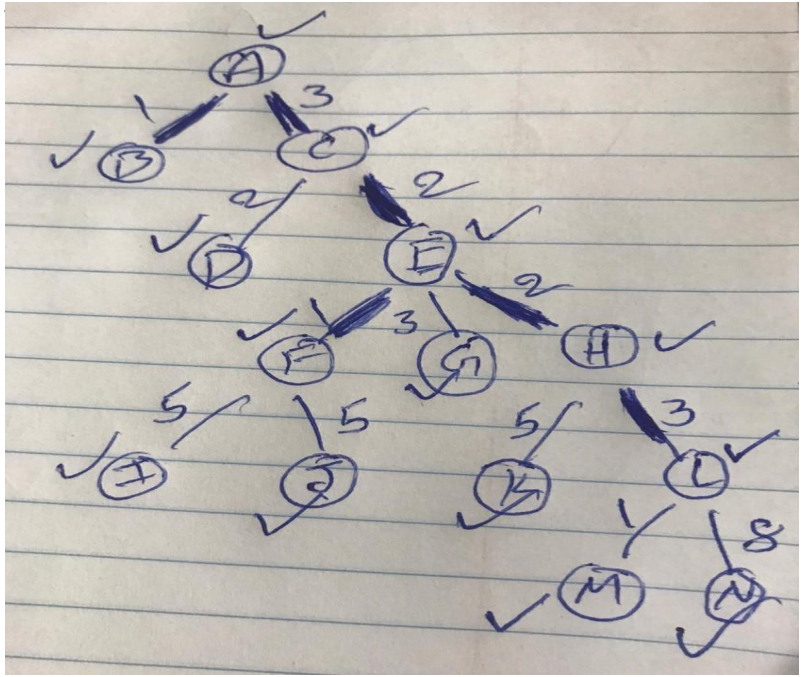
3. Pick the vertex with minimum key value and not already included in MST (i.e., not in mstSET).

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



Steps - Continues

- We repeat the above steps until mstSet includes all vertices of given graph. Finally, we get the following graph.
- As all the vertices are visited, now the algorithm stops.
- The cost of the spanning tree is **(3 + 2 + 2 + 3 + 8) = 18**. There is no more spanning tree in this graph with cost less than **18**.



Kruskal's Algorithm to solve maze

After Sorting -

Weight	Src	Dest
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

Sorted list of edges

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.

Kruskal Time Complexity

- Since the complexity is **O(E * log(V)),** the Kruskal algorithm is better used with sparse graphs, where we don't have lots of edges.

- However, since we are examining all edges one by one sorted on ascending order based on their weight, this allows us great control over the resulting MST. Since different MSTs come from different edges with the same cost, in the Kruskal algorithm, all these edges are located one after another when sorted.

Prim's Time Complexity

- In the beginning, we add the source node to the queue with a zero weight and without an edge. Also, we initialize the total cost with zero and mark all nodes as not yet included inside the MST.
- After that, we perform multiple steps. In each step, we extract the node with the lowest weight from the queue. For each extracted node, we increase the cost of the MST by the weight of the extracted edge. Also, in case the edge of the extracted node exists, we add it to the resulting MST.
- When we finish handling the extracted node, we iterate over its neighbors. In case the neighbor is not yet included in the resulting MST, we use the addOrUpdate function to add this neighbor to the queue. Also, we add the weight of the edge and the edge itself.
- The complexity of Prim's algorithm is **O(E + V log(V)),** where E is the number of edges and V is the number of vertices inside the graph.

Analysis

- The advantage of Prim's algorithm is its complexity, which is better than Kruskal's algorithm. Therefore, **Prim's algorithm is helpful when dealing with dense graphs that have lots of edges**.
- However, **Prim's algorithm doesn't allow us much control over the chosen edges when multiple edges with the same weight occur**. The reason is that only the edges discovered so far are stored inside the queue, rather than all the edges like in Kruskal's algorithm.
- Also, unlike Kruskal's algorithm, **Prim's algorithm is a little harder to implement**.

Comparision between Prim's & Kruskal

Kruskal algorithm is better to use regarding the easier implementation and the best control over the resulting MST. However, Prim's algorithm offers better complexity.

	Kruskal	**Prim**
Multiple MSTs	Offers a good control over the resulting MST	Controlling the MST might be a little harder
Implementation	Easier to implement	Harder to implement
Requirements	Disjoint set	Priority queue
Time Complexity	$O(E \cdot log(V))$	$O(E + V \cdot log(V))$

CONCLUSION

- Kruskal's has better running times if the number of edges is low, while Prim's has a better running time if both the number of edges and the number of nodes are low.
- So, of course, the best algorithm depends on the graph and if you want to bear the cost of complex data structures.

References:

- Maze

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

- Spanning Tree

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

- Google Slides URL -

  https://drive.google.com/file/d/1YUvHo-P8aX5kgTKTz_n6q-oUM9SRrBx5/view?usp=sharing