Algorithms and Data Structures Assignment

Name: Fatima Alubaidi
Student number: C22305656

The purpose of this assignment is to implement different algorithms for graph traversal, minimum spanning tree (MST) construction, and shortest path tree (SPT) construction. I will be implementing Prim's algorithm, Kruskal's algorithm, and Dijkstra's algorithm in Java. These algorithms are important for things such as computer networking and transport. I will implement BFS AND DFS.

Prim's Minimum Spanning Tree:

Prim's algorithm is a way to find the smallest collection of edges in a graph that connects all the vertices. It starts with just one vertex and keeps adding edges with the smallest weights until all vertices are connected. The goal is to make sure all vertices are linked while keeping the total weight of the edges as low as possible.

The program I created uses a text file (Graph1.txt) to create a graph. It starts from a chosen (entered) vertex and keeps adding edges with the lowest weights until all vertices are connected. It does this by picking the next best edge each time until all vertices are covered.

Here are the steps:

1. Determine the starting vertex of the MST.
2. Search for all edges connecting the current tree to new vertices.
3. Select the edge with the minimum weight and include it in the MST, ensuring the new vertex is different from the previous one.
4. Keep repeating step 2 and 3 until all new vertices and edges together form the complete MST.

Kruskal's Minimum Spanning Tree:

Kruskal's algorithm is a way to find the smallest possible tree that connects all the vertices in a graph with weights on its edges. This tree is called a minimum spanning tree. If the graph isn't just one connected piece, Kruskal's can find the smallest tree for each piece.

To use Kruskal's algorithm, we need a way to see which elements belong to which groups and to join groups together. We call this a Union-Find

data structure. There are different ways to do this, but we're using something called disjoint-set trees.

Union-Find keeps track of a bunch of elements that are split into different groups. In Kruskal's algorithm, we use Union-Find to see if adding an edge would create a loop. Each group in Union-Find is like a tree, with a main root. Every element connected to that root is in the same group.

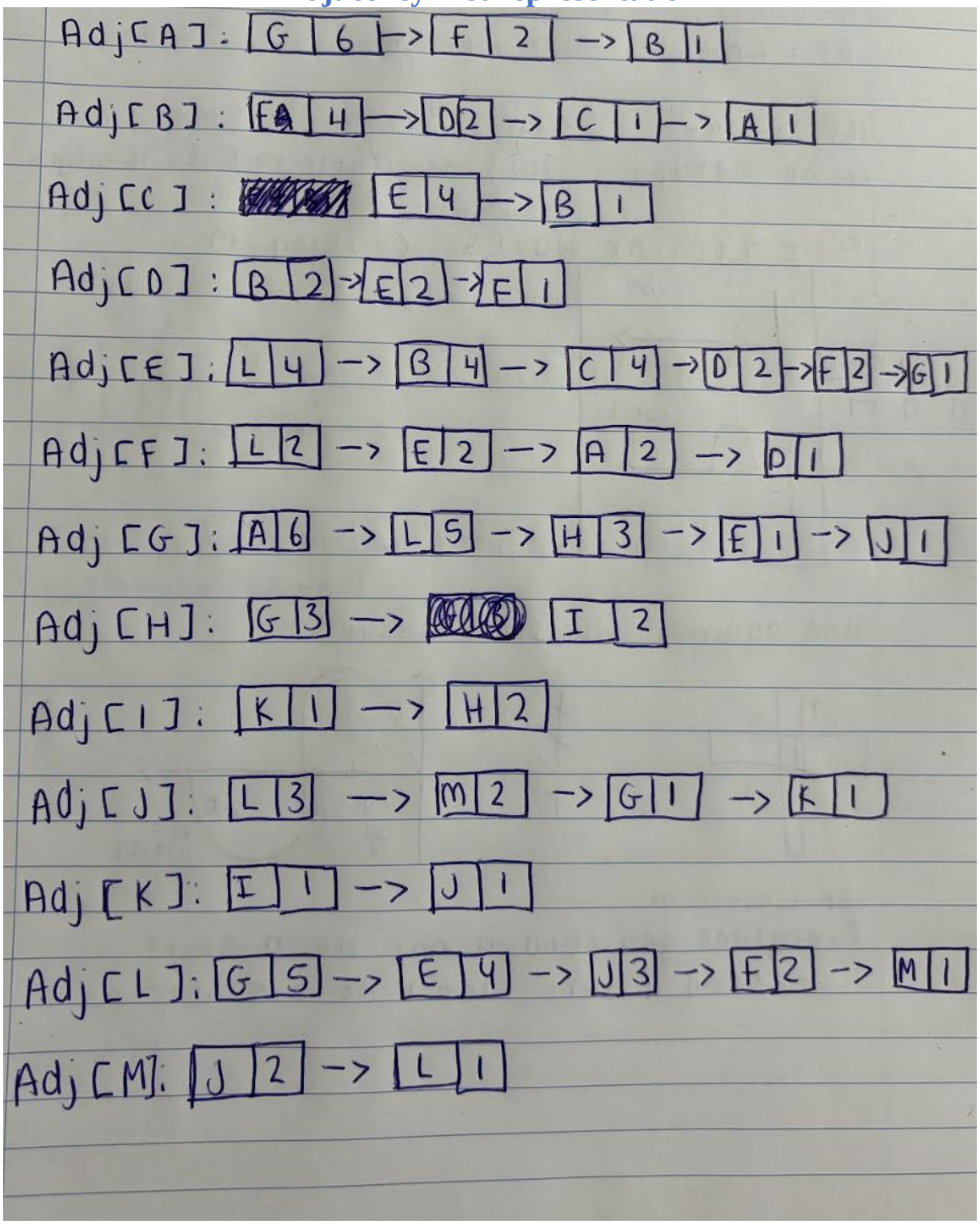
Steps of Kruskals:

1. Start with an empty set of edges (this will be the MST).
2. Sort all the edges in the graph in ascending order based on their weights - smallest to largest.
3. Iterate through each edge from the sorted list.
4. Check if adding the edge creates a cycle in the current set of edges. If not, add it to the minimum spanning tree.
5. Continue adding edges until all vertices are connected or there are no more edges left.
6. The resulting set of edges is the minimum spanning tree of the graph.

Adjacency Matrix:

	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**
Adj(A)	@	1	0	0	0	2	6	0	0	0	0	0	0
Adj(B)	1	@	1	2	4	0	0	0	0	0	0	0	0
Adj(C)	0	1	@	0	4	0	0	0	0	0	0	0	0
Adj(D)	0	2	0	@	2	1	0	0	0	0	0	0	0
Adj(E)	0	4	4	2	@	2	1	0	0	0	0	4	0
Adj(F)	2	0	0	1	2	@	0	0	0	0	0	2	0
Adj(G)	6	0	0	0	1	0	@	3	0	1	0	5	0
Adj(H)	0	0	0	0	0	0	3	@	2	0	0	0	0
Adj(I)	0	0	0	0	0	0	0	2	@	0	1	0	0
Adj(J)	0	0	0	0	0	0	1	0	0	@	1	3	2
Adj(K)	0	0	0	0	0	0	0	0	1	1	@	0	0
Adj(L)	0	0	0	0	2	2	5	0	0	3	0	@	1
Adj(M)	0	0	0	0	0	0	0	0	0	2	0	1	@

Adjacency List representation



Construction of the MST using Prim's algorithm:

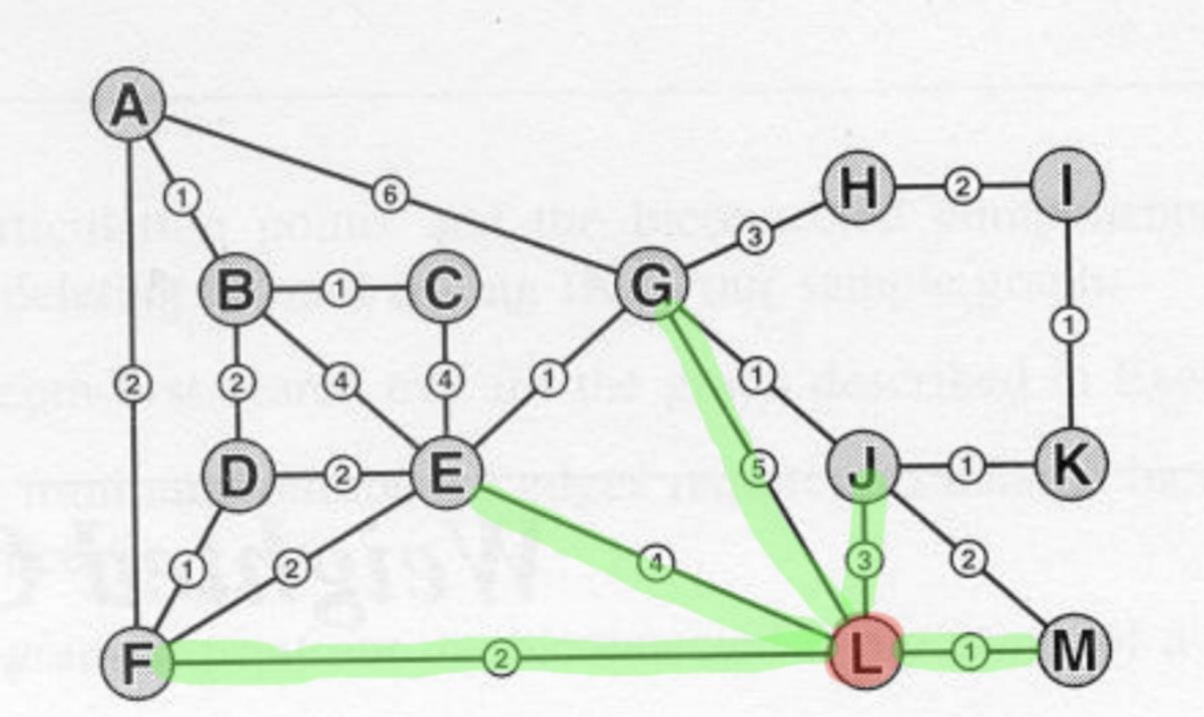
Initial state from vertex L:

		A	B	C	D	E	F	G	H	I	J	K	L	M
	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Parent[]		0	0	0	0	0	0	0	0	0	0	0	0	0
Dist[]		0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞

Heap:
M1
F2
J3
E4
G5

		A	B	C	D	E	F	G	H	I	J	K	L	M
	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Parent[]		0	0	0	0	L	L	L	0	0	L	0	@	L
Dist[]		∞	∞	∞	∞	4	2	5	∞	∞	3	∞	0	1

Graph representation:



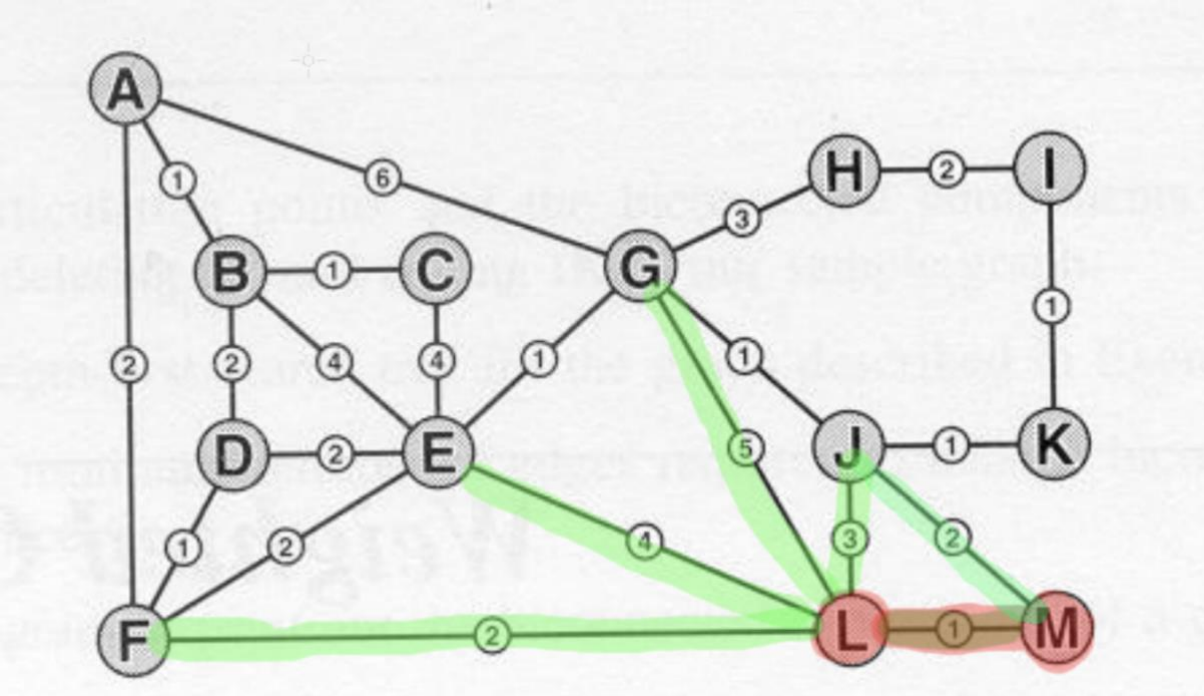
Step 1: Next up is M

Heap:
F2
J2
E4
G5

Parent and Distance Arrays:

		A	B	C	D	E	F	G	H	I	J	K	L	M
	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Parent[]		0	0	0	0	L	L	L	0	0	~~L~~ M	0	@	L
Dist[]		∞	∞	∞	∞	4	2	5	∞	∞	~~3~~ 2	∞	0	1

Graph Representation



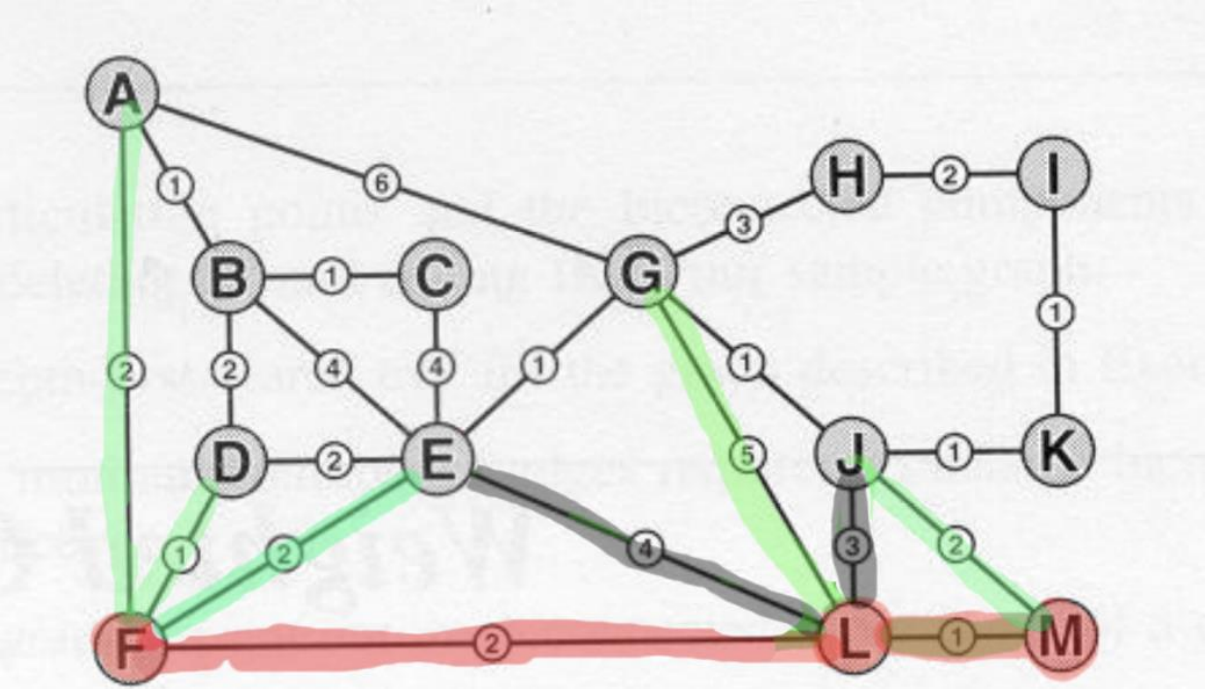
Step 2: Next up is F

Heap:
D1
A2
E2
J2
E4
G5

Parent and Distance Arrays:

		A	B	C	D	E	F	G	H	I	J	K	L	M
	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Parent[]		0	0	0	0	~~L~~ F	L	L	0	0	~~L~~ M	0	@	L
Dist[]		∞	∞	∞	∞	~~4~~ 2	2	5	∞	∞	~~3~~ 2	∞	0	1

Graph Representation:



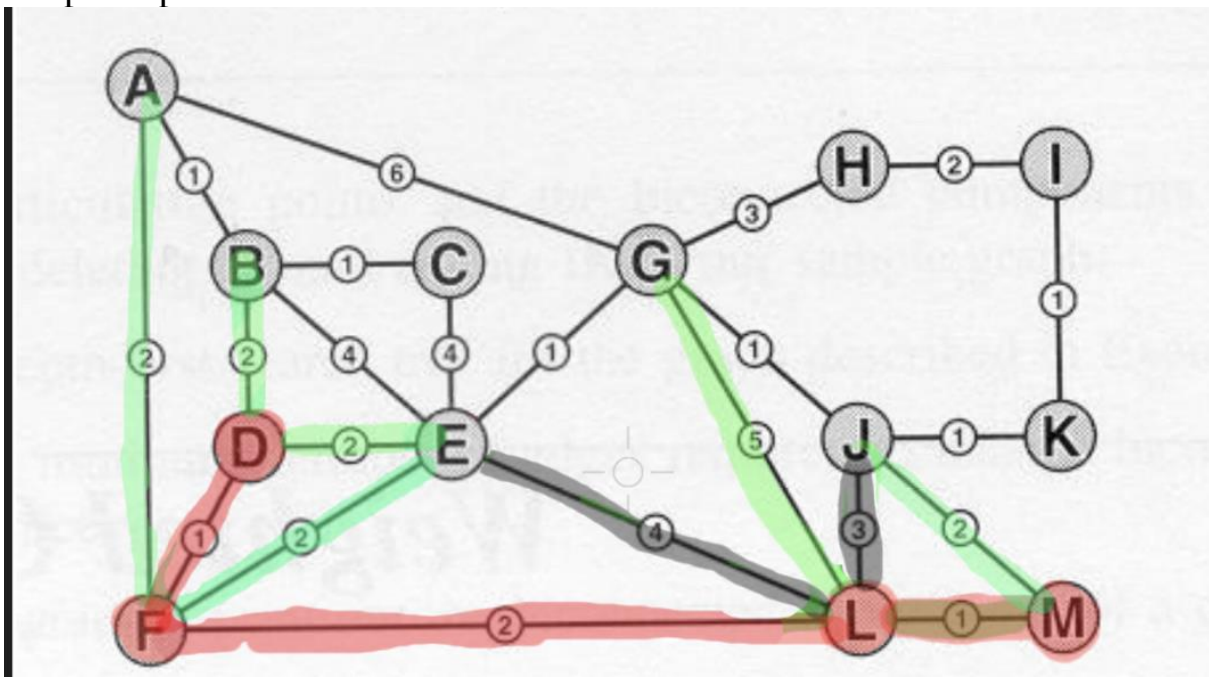
Step 3: Next up is D

Heap:
A2
B2
E2
J2
E4
G5

Parent and Distance Arrays:

		A	B	C	D	E	F	G	H	I	J	K	L	M
	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Parent[]		F	D	0	0	~~L~~ F	L	L	0	0	~~L~~ M	0	@	L
Dist[]		2	2	∞	∞	~~4~~ 2	2	5	∞	∞	~~3~~ 2	∞	0	1

Graph Representation:



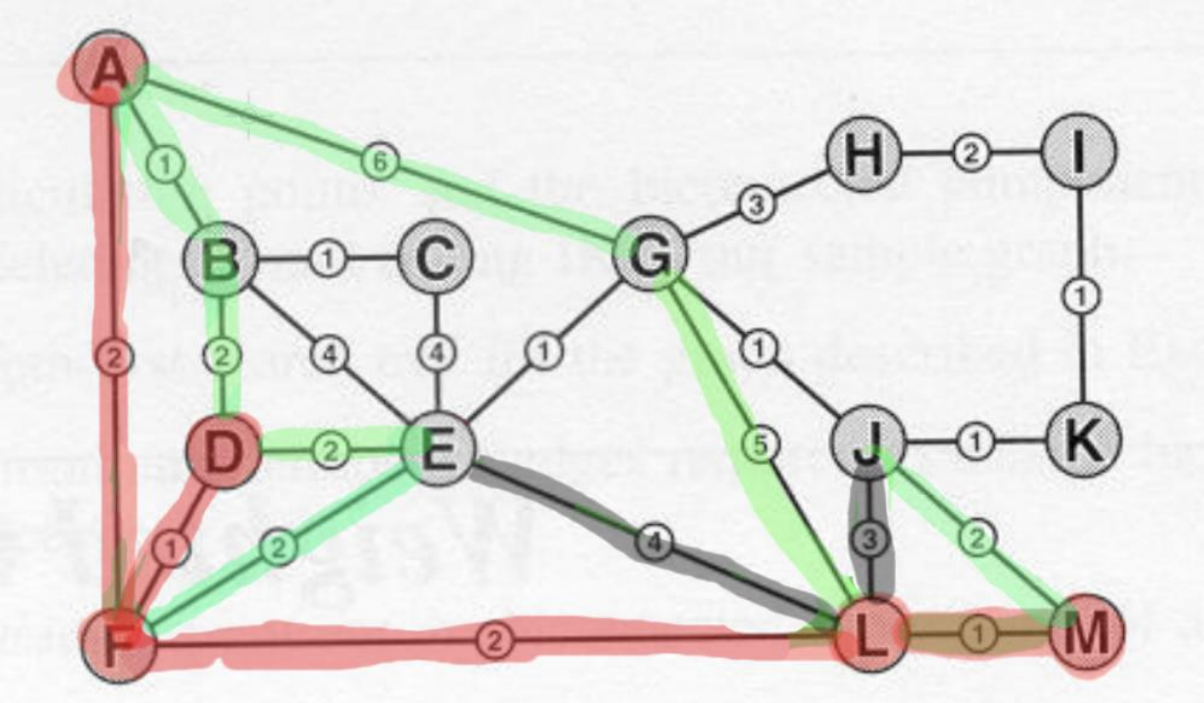
Step 4: Next up is A

Heap:
B1
E2
J2
E4
G5

Parent and Distance Arrays:

		A	B	C	D	E	F	G	H	I	J	K	L	M
	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Parent[]		F	~~D~~ A	0	0	~~L~~ F	L	L	0	0	~~L~~ M	0	@	L
Dist[]		2	~~2~~ 1	∞	∞	~~4~~ 2	2	5	∞	∞	~~3~~ 2	∞	0	1

Graph Representation:



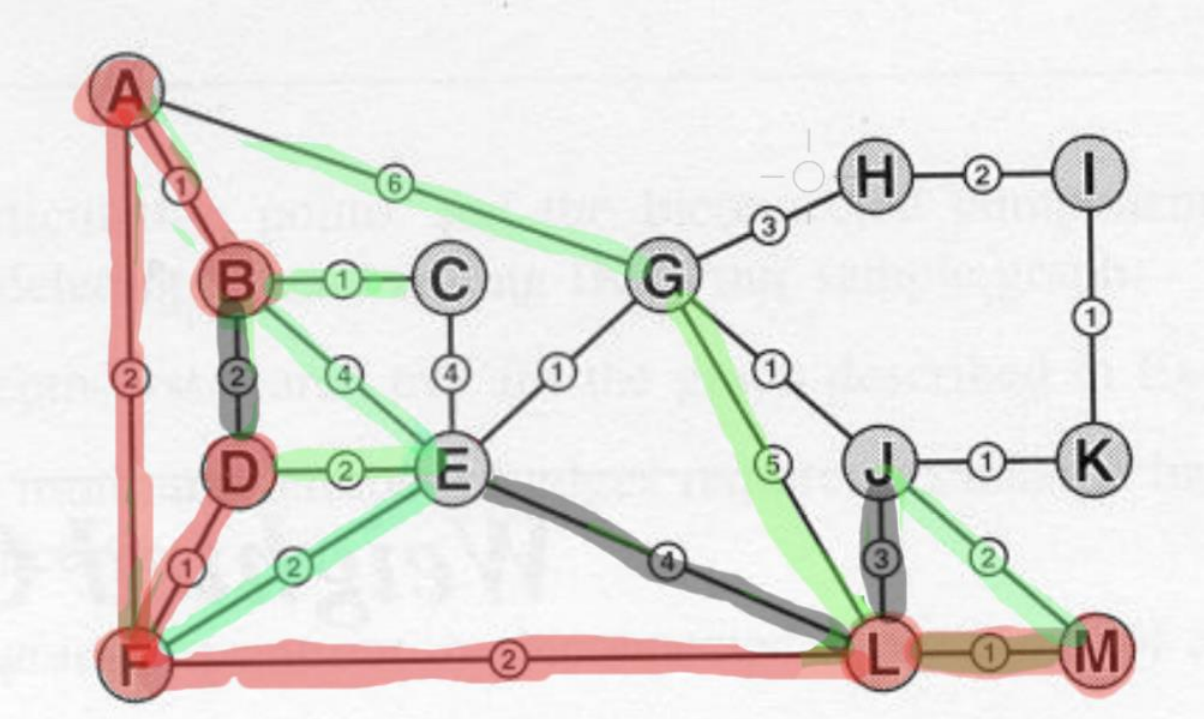
Step 5: Next up is B

Heap:
C1
E2
G5
J2

Parent and Distance Arrays:

		A	B	C	D	E	F	G	H	I	J	K	L	M
	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Parent[]		F	~~D~~ A	B	F	~~L~~ F	L	L	0	0	~~L~~ M	0	@	L
Dist[]		2	~~2~~ 1	1	1	~~4~~ 2	2	5	∞	∞	~~3~~ 2	∞	0	1

Graph representation:



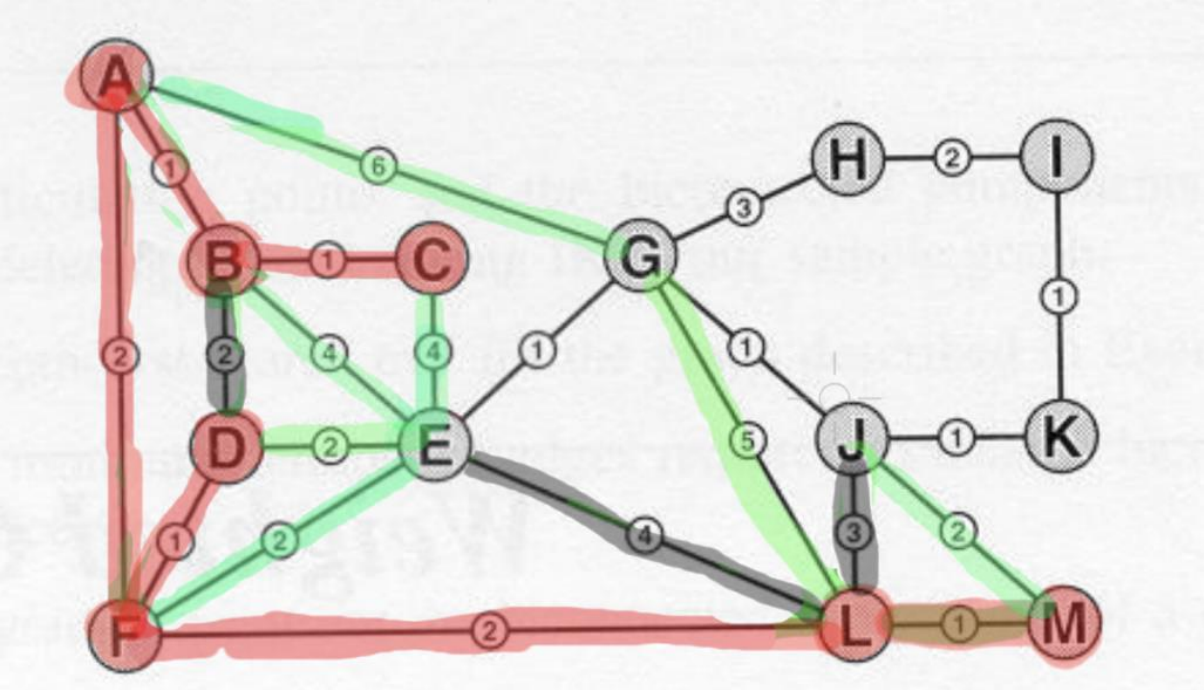
Step 6: Next up is C

Heap:
J2
E2
G5

Parent and Distance Arrays:

		A	B	C	D	E	F	G	H	I	J	K	L	M
	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Parent[]		F	~~D~~ A	B	F	~~L~~ F	L	L	0	0	~~L~~ M	0	0	L
Dist[]		2	~~2~~ 1	1	1	~~4~~ 2	2	5	∞	∞	~~3~~ 2	∞	0	1

Graph Representation:



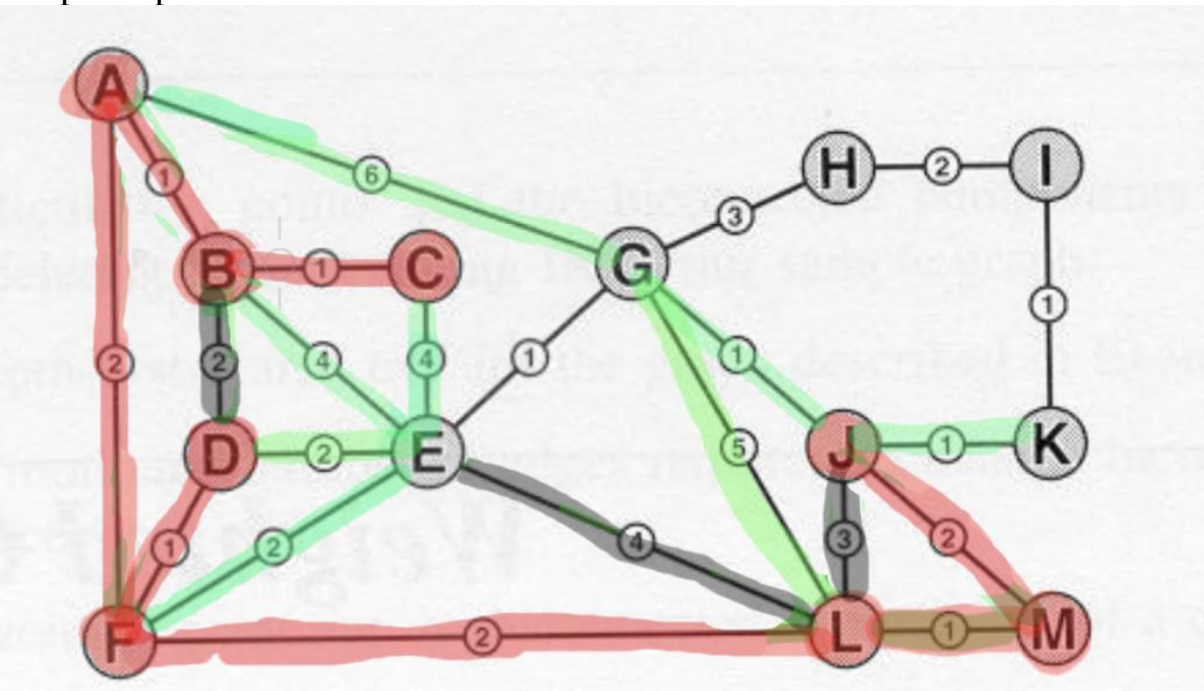
Step 7: Next up is J

Heap:
K1
E2
G5

Parent and Distance Arrays:

		A	B	C	D	E	F	G	H	I	J	K	L	M
	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Parent[]		F	~~D~~ A	B	F	~~L~~ F	L	~~L~~ J	0	0	~~L~~ M	0	0	L
Dist[]		2	~~2~~ 1	1	1	~~4~~ 2	2	~~5~~ 1	∞	∞	~~3~~ 2	∞	0	1

Graph Representation:



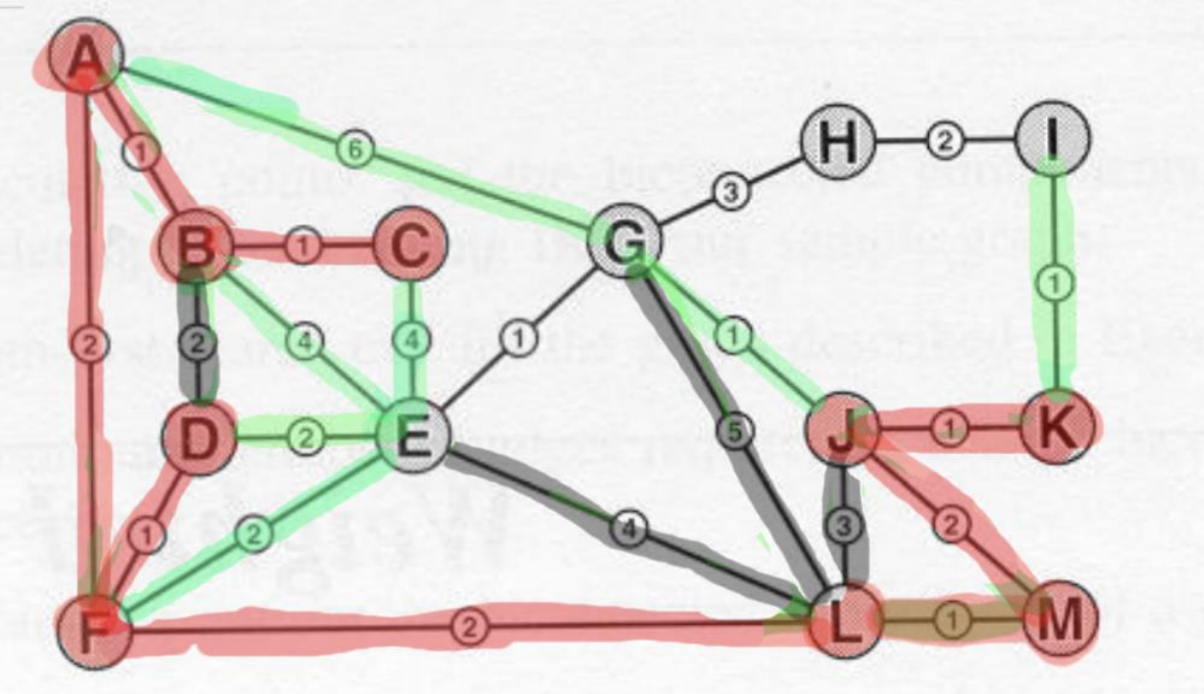
Step 8: Next up is K

Heap:
G1
E2
I1

Parent and Distance Arrays:

		A	B	C	D	E	F	G	H	I	J	K	L	M
	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Parent[]		F	~~D~~ A	B	F	~~L~~ F	L	~~L~~ J	0	0	~~L~~ M	J	0	L
Dist[]		2	~~2~~ 1	1	1	~~4~~ 2	2	~~5~~ 1	∞	∞	~~3~~ 2	1	∞	1

Graph Representation:



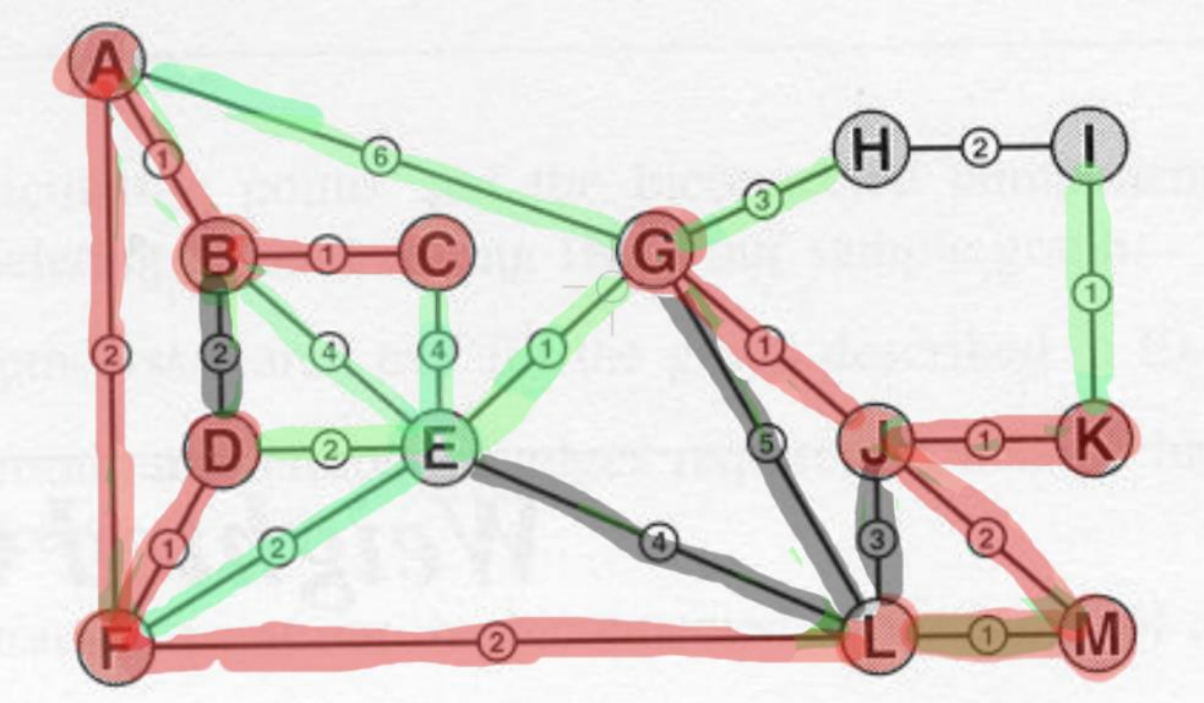
Step 9: Next up is G

Heap:
E1
I1
H3

Parent and Distance Arrays:

		A	B	C	D	E	F	G	H	I	J	K	L	M
	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Parent[]		F	~~D~~ A	B	F	~~L~~ ~~F~~ G	L	~~L~~ J	G	0	~~L~~ M	J	0	L
Dist[]		2	~~2~~ 1	1	1	~~4~~ ~~2~~ 1	2	~~5~~ 1	3	∞	~~3~~ 2	1	∞	1

Graph representation:



Step 10: Next up is I

Heap:

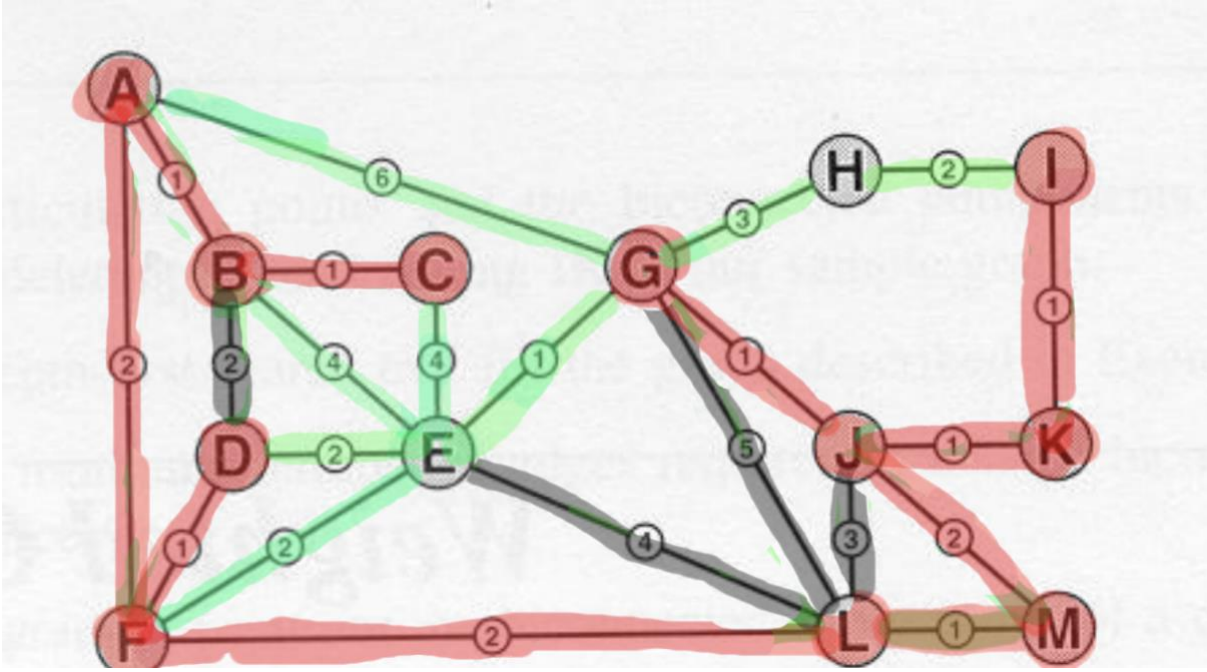
E1

H2

Parent and Distance Arrays:

		A	B	C	D	E	F	G	H	I	J	K	L	M
	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Parent[]		F	~~D~~ A	B	F	~~L~~ ~~F~~ G	L	~~L~~ J	~~G~~ I	K	~~L~~ M	J	0	L
Dist[]		2	~~2~~ 1	1	1	~~4~~ ~~2~~ 1	2	~~5~~ 1	~~3~~ 2	1	~~3~~ 2	1	∞	1

Graph Representation:



Step 11: Next up is E

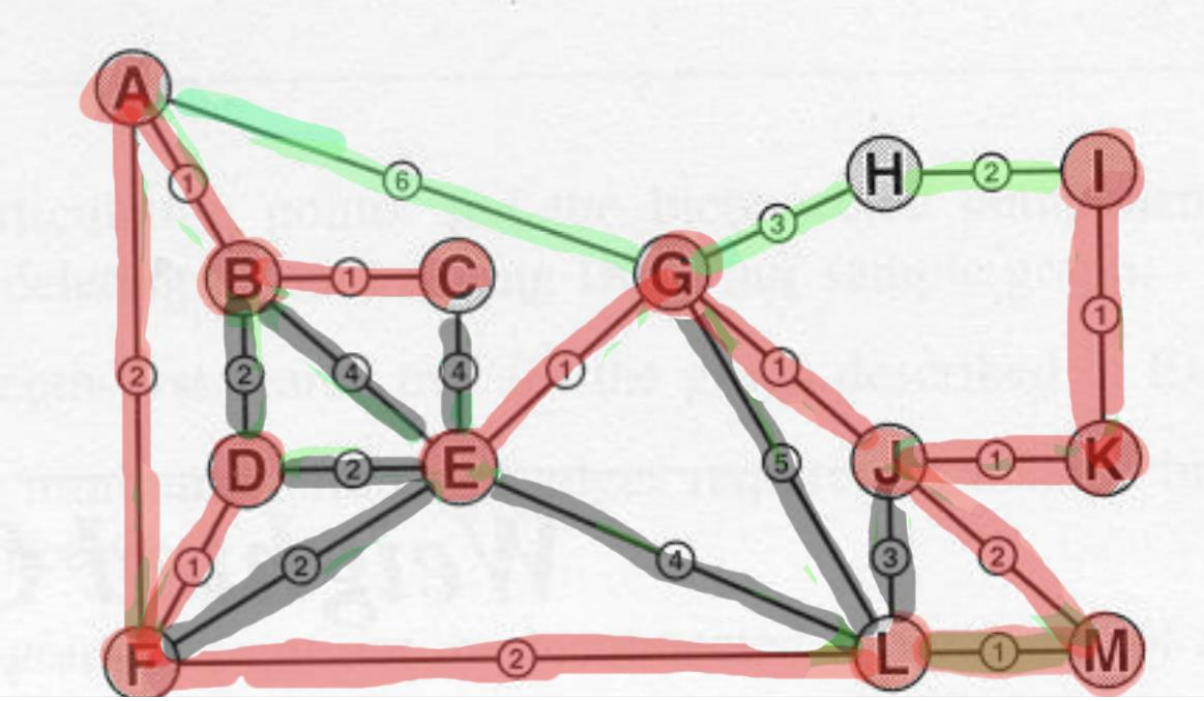
Heap:

H2

Parent and Distance Arrays:

		A	B	C	D	E	F	G	H	I	J	K	L	M
	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Parent[]		F	~~D~~ A	B	F	~~L~~ ~~F~~ G	L	~~L~~ J	~~G~~ I	K	~~L~~ M	J	0	L
Dist[]		2	~~2~~ 1	1	1	~~4~~ ~~2~~ 1	2	~~5~~ 1	~~3~~ 2	1	~~3~~ 2	1	∞	1

Graph Representation:



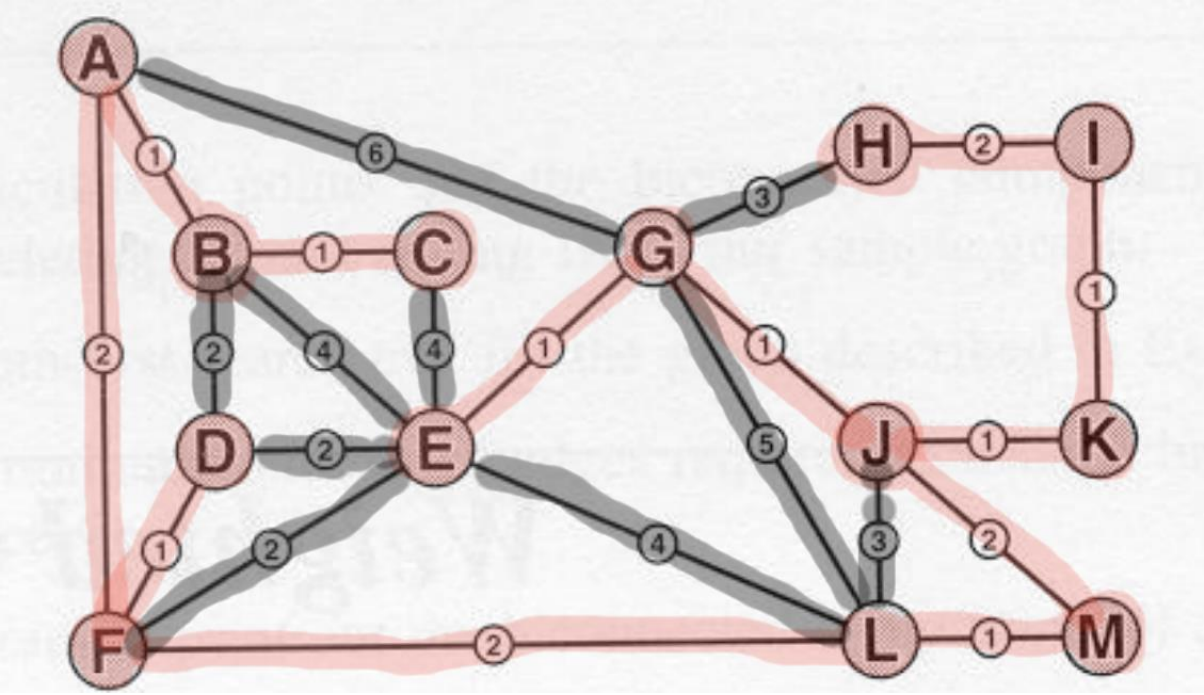
Step 12: Next up is H

Heap:
The heap is now empty

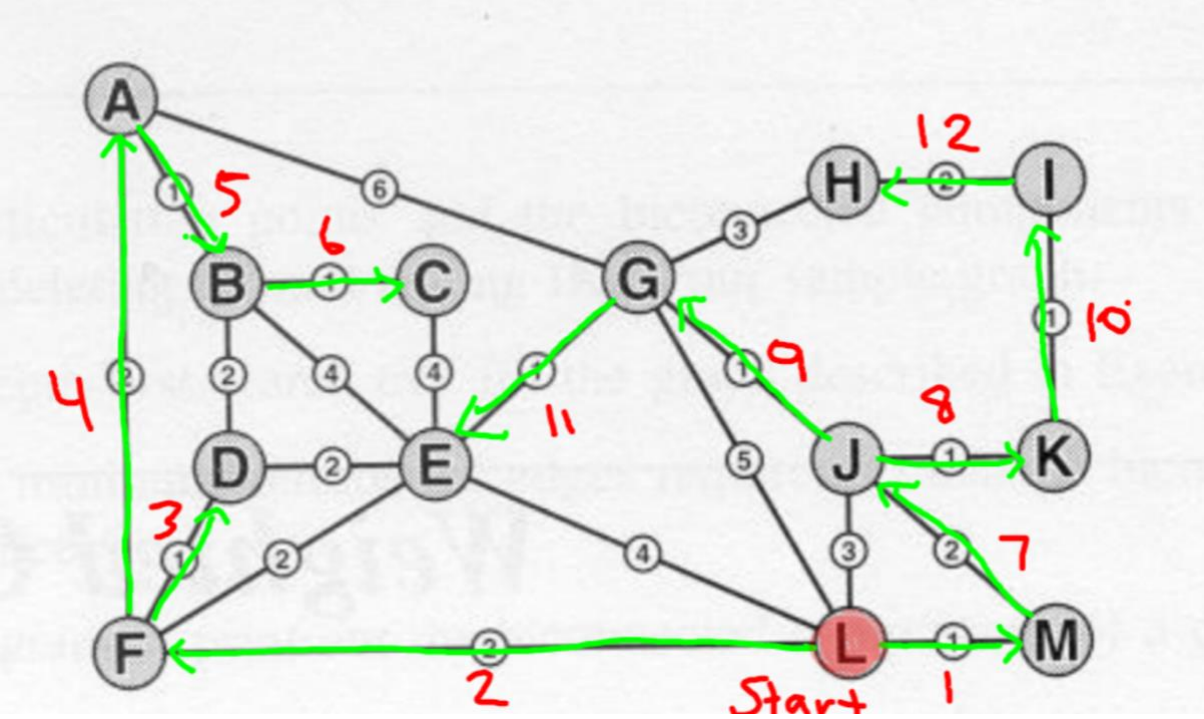
Parent and Distance Arrays:

		A	B	C	D	E	F	G	H	I	J	K	L	M
	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Parent[]		F	~~D~~ A	B	F	~~L~~ ~~F~~ G	L	~~L~~ J	~~G~~ I	K	~~L~~ M	J	0	L
Dist[]		2	~~2~~ 1	1	1	~~4~~ ~~2~~ 1	2	~~5~~ 1	~~3~~ 2	1	~~3~~ 2	1	∞	1

Graph Representation:



Final step by step prims MST superimposed on graph:



Construction of the SSP using Dijkstra's algorithm:

Initial state from vertex L:

		A	B	C	D	E	F	G	H	I	J	K	L	M
	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Parent[]		0	0	0	0	0	0	0	0	0	0	0	0	0
Dist[]		∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞

Start on vertex L:

		A	B	C	D	E	F	G	H	I	J	K	L	M
	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Parent[]		0	0	0	0	0	0	0	0	0	0	0	0	0
Dist[]		∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	0	∞

STEP 1: L -> A

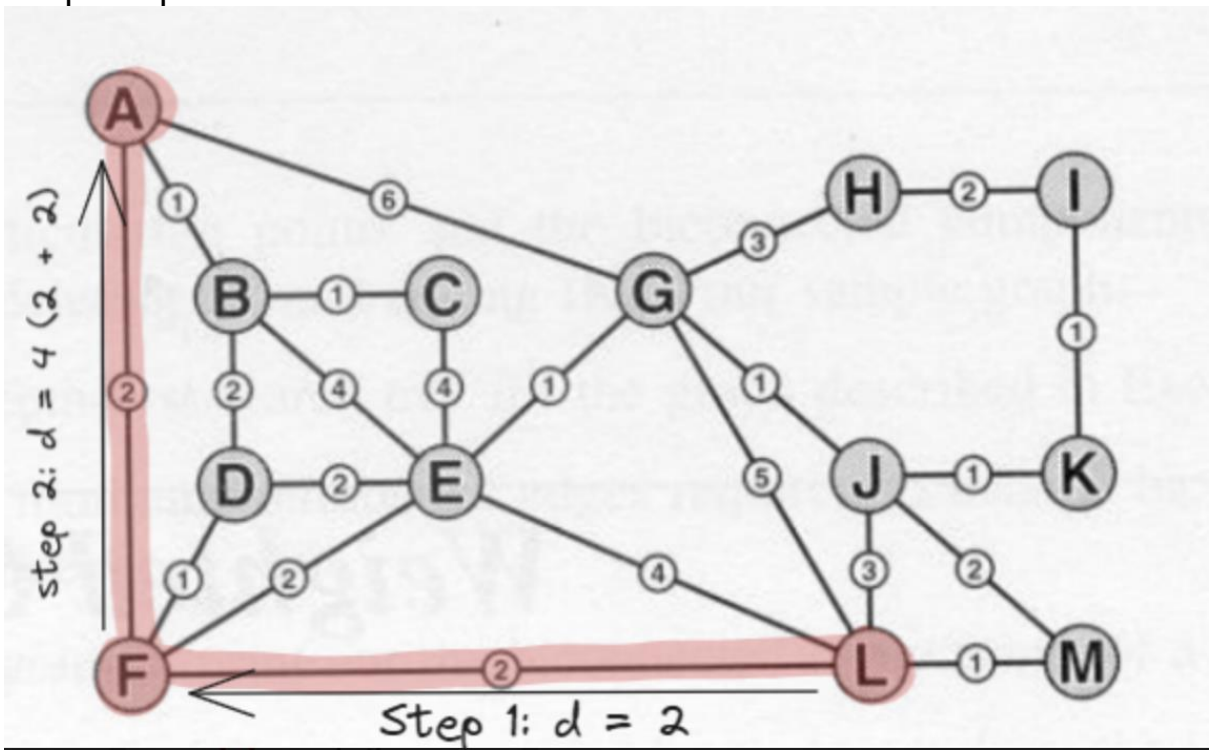
Shortest Path: L -> F -> A

Distance from L = 4

Heap: B C D E F G H I J K L M

		A	B	C	D	E	F	G	H	I	J	K	L	M
	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Parent[]		L	0	0	0	0	0	0	0	0	0	0	0	0
Dist[]		4	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	0	∞

Graph Representation:



STEP 2: L -> B

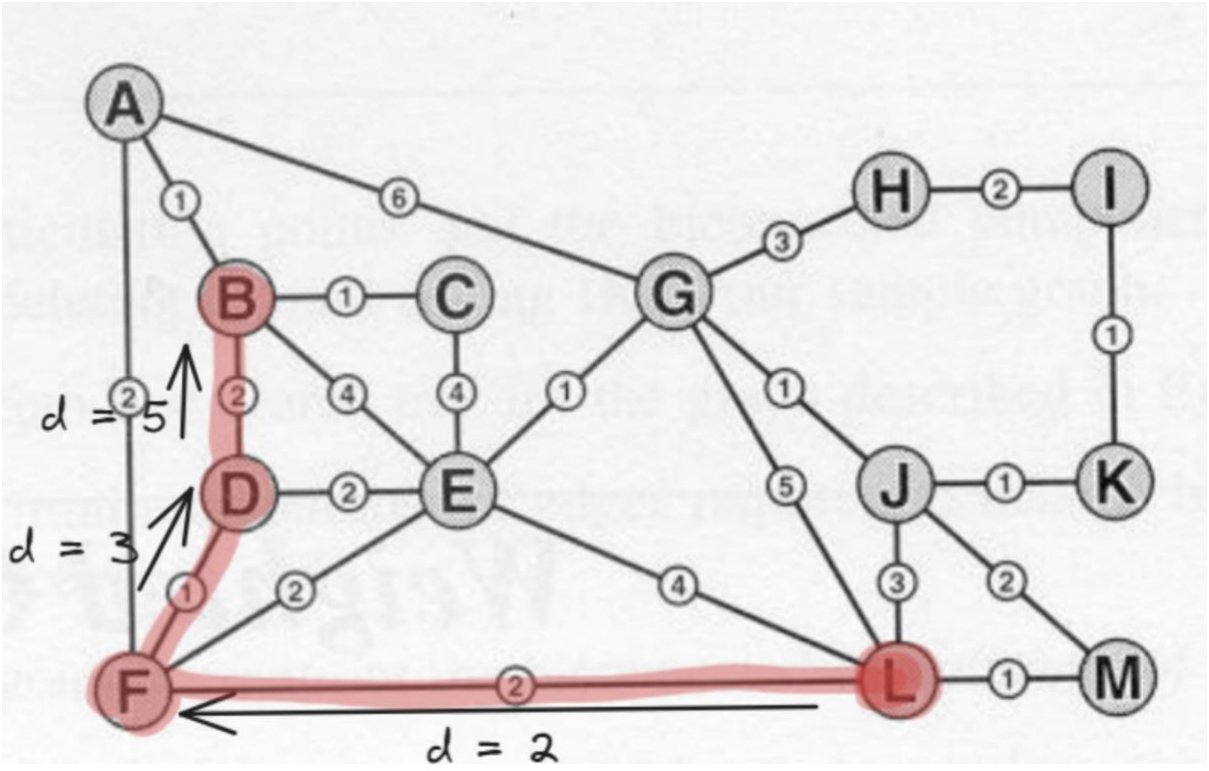
Shortest Path: L -> F -> D -> B

Distance from L = 5

Heap: C D E F G H I J K L M

		A	B	C	D	E	F	G	H	I	J	K	L	M
	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Parent[]		0	L	0	0	0	0	0	0	0	0	0	0	0
Dist[]		∞	5	∞	∞	∞	∞	∞	∞	∞	∞	∞	0	∞

Graph Representation:



STEP 3: L -> C

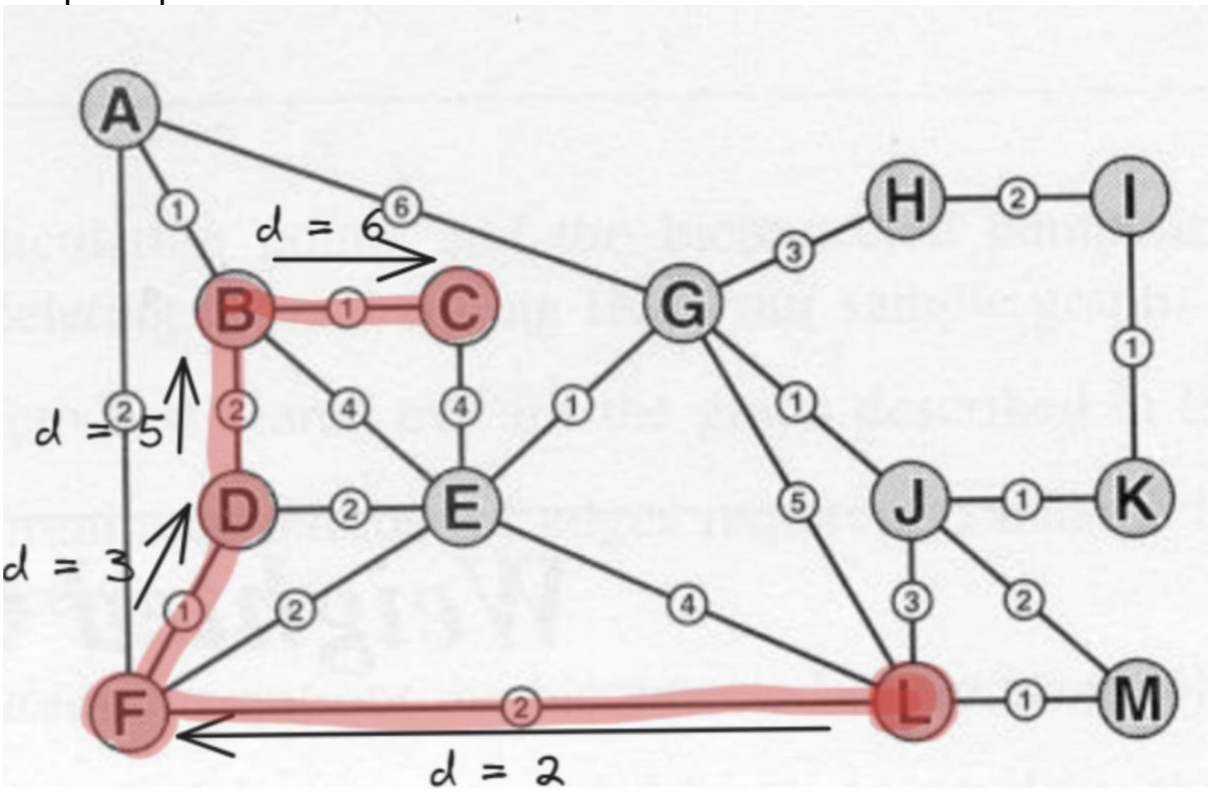
Shortest Path: L -> F -> D -> B -> C

Distance from L = 6

Heap: D E F G H I J K L M

		A	B	C	D	E	F	G	H	I	J	K	L	M
	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Parent[]		0	0	L	0	0	0	0	0	0	0	0	0	0
Dist[]		∞	∞	6	∞	∞	∞	∞	∞	∞	∞	∞	0	∞

Graph Representation:



STEP 4: L -> D

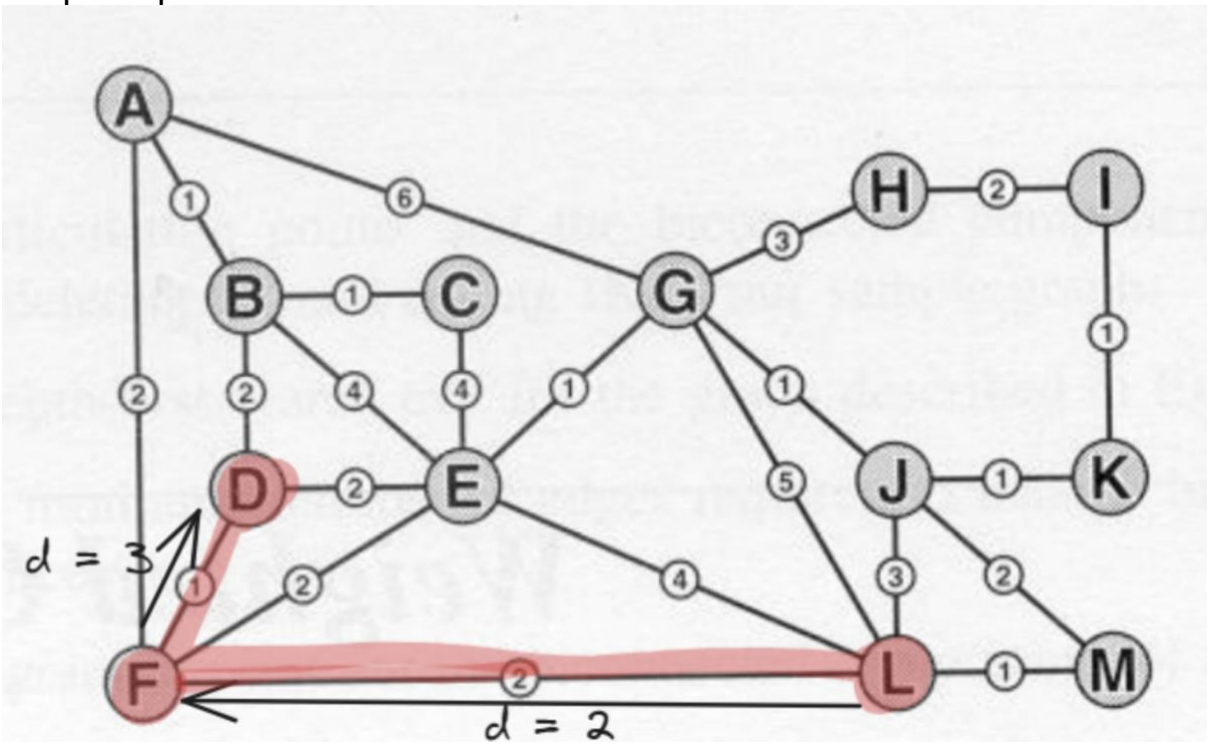
Shortest Path: L -> F -> D

Distance from L = 3

Heap: E F G H I J K L M

		A	B	C	D	E	F	G	H	I	J	K	L	M
	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Parent[]		0	0	0	L	0	0	0	0	0	0	0	0	0
Dist[]		∞	∞	∞	3	∞	∞	∞	∞	∞	∞	∞	0	∞

Graph Representation:



STEP 5: L -> E

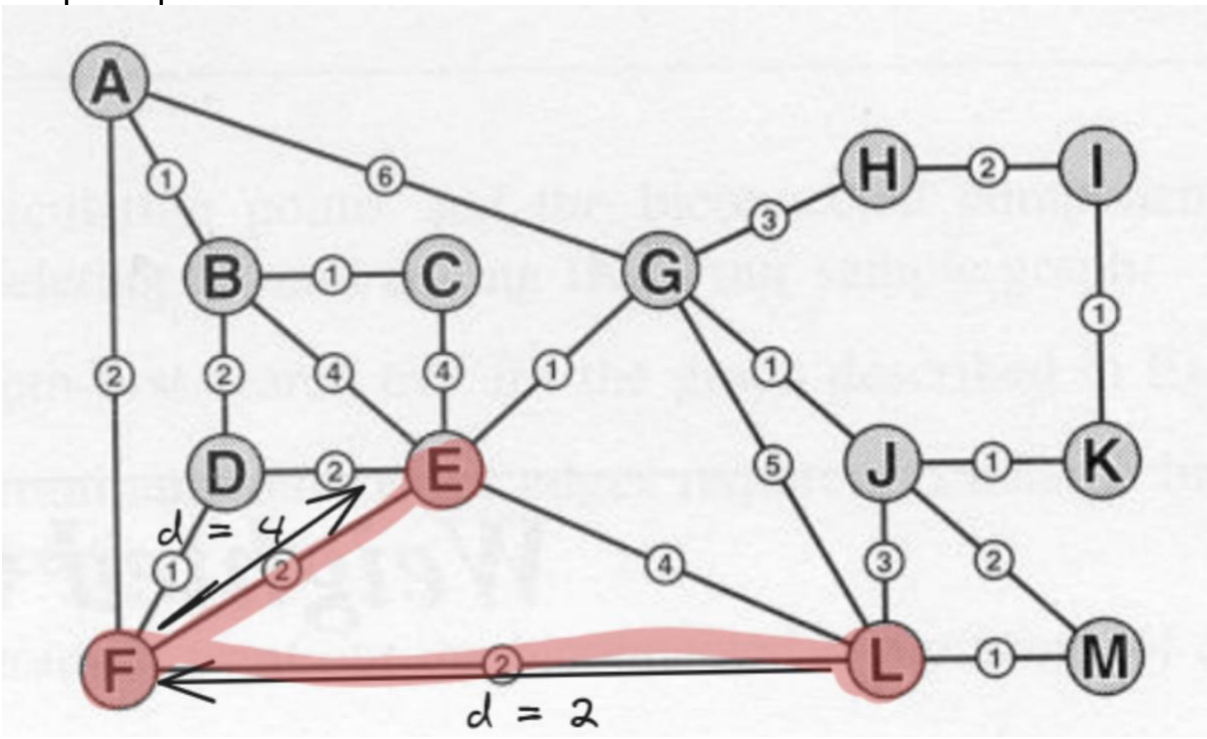
Shortest Path: L -> F -> E

Distance from L = 4

Heap: F G H I J K L M

		A	B	C	D	E	F	G	H	I	J	K	L	M
	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Parent[]		0	0	0	0	L	0	0	0	0	0	0	0	0
Dist[]		∞	∞	∞	∞	4	∞	∞	∞	∞	∞	∞	0	∞

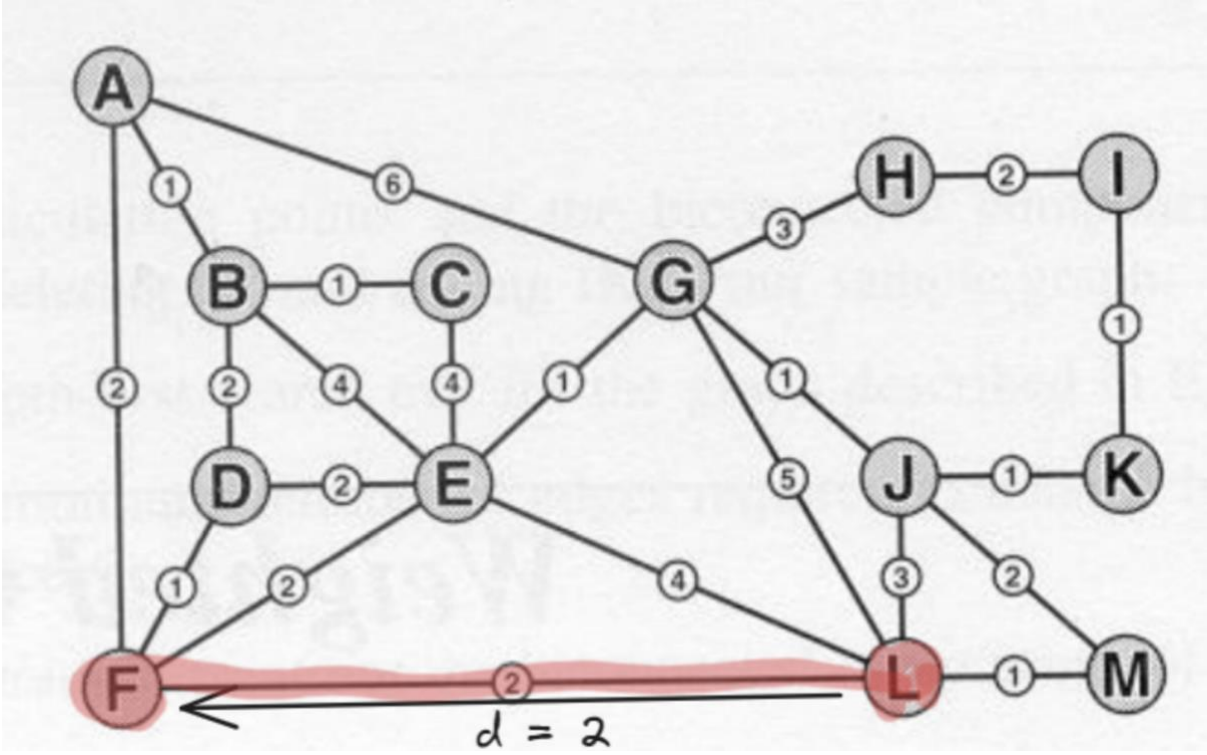
Graph Representation:



STEP 6 : L -> F
Shortest Path: L -> F
Distance from L = 2
Heap: G H I J K L M

		A	B	C	D	E	F	G	H	I	J	K	L	M
	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Parent[]		0	0	0	0	0	L	0	0	0	0	0	0	0
Dist[]		∞	∞	∞	∞	∞	2	∞	∞	∞	∞	∞	0	∞

Graph Representation:



STEP 7: L -> G

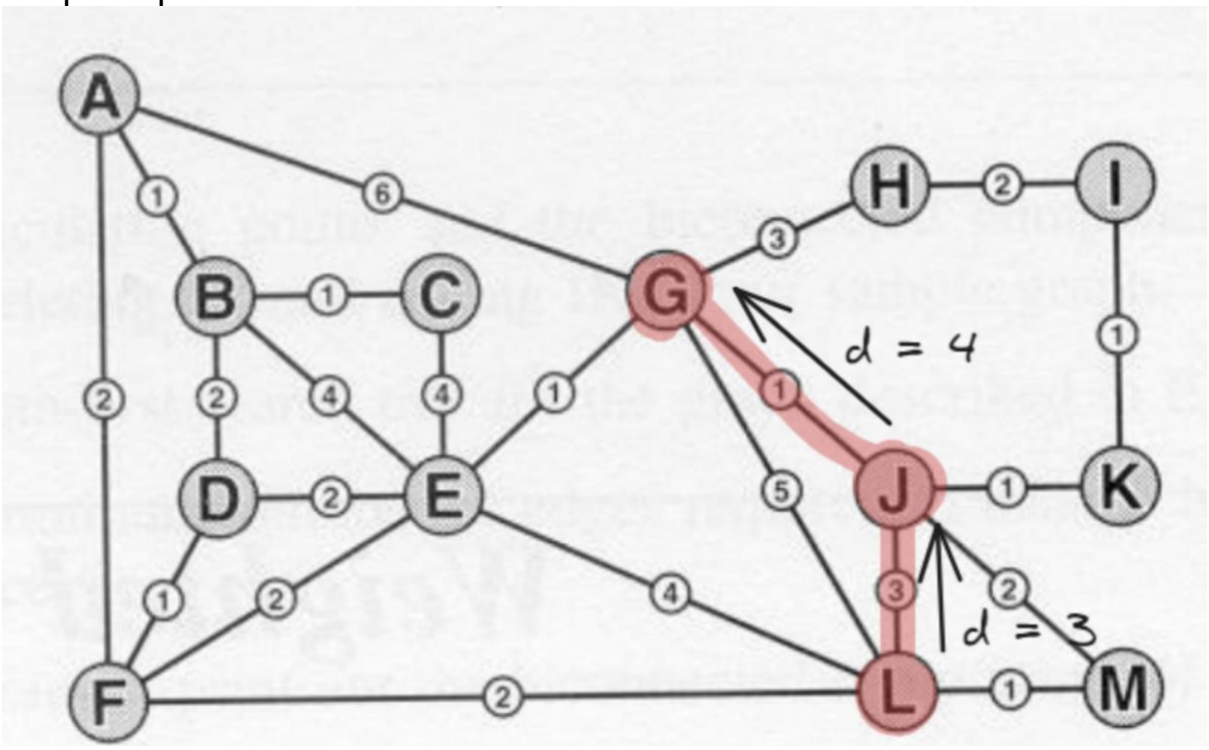
Shortest Path: L -> J -> G

Distance from L = 4

Heap: H I J K L M

		A	B	C	D	E	F	G	H	I	J	K	L	M
	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Parent[]		0	0	0	0	0	0	L	0	0	0	0	0	0
Dist[]		∞	∞	∞	∞	∞	∞	4	∞	∞	∞	∞	0	∞

Graph Representation:



STEP 8: L -> H

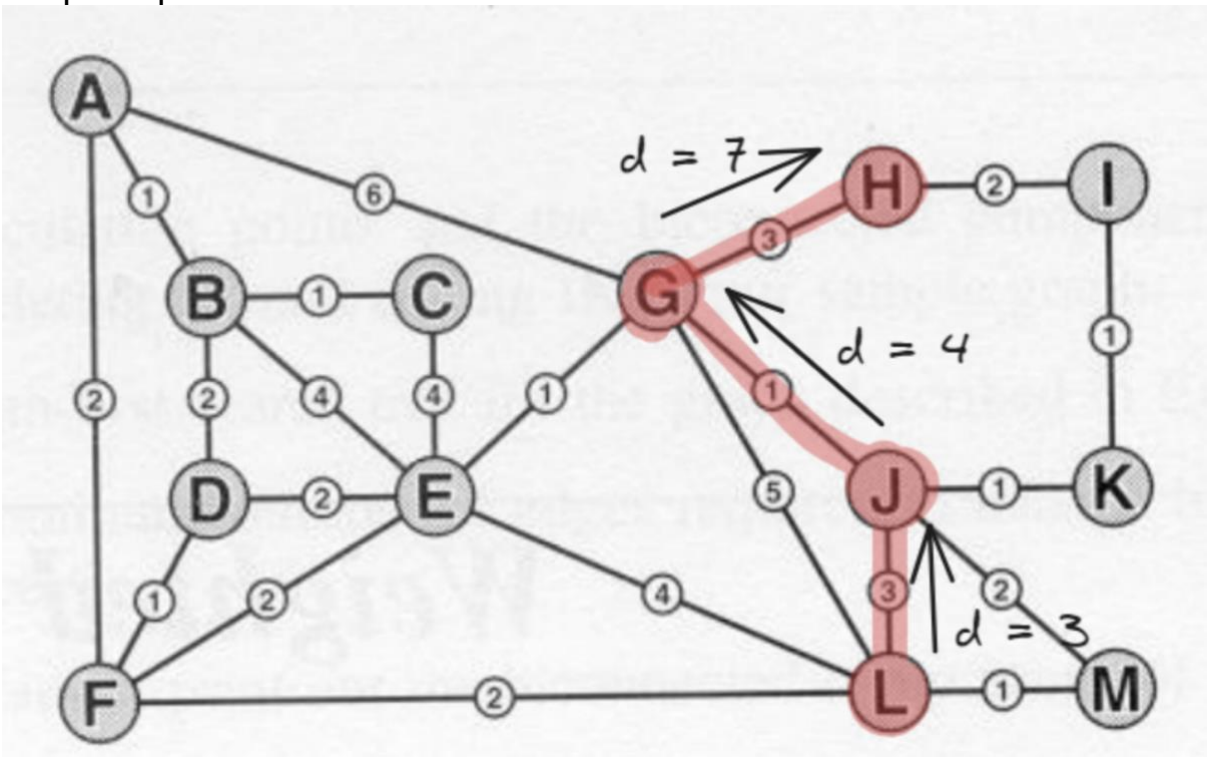
Shortest Path: L -> J -> G -> H

Distance from L = 7

Heap: I J K L M

		A	B	C	D	E	F	G	H	I	J	K	L	M
	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Parent[]		0	0	0	0	0	0	0	L	0	0	0	0	0
Dist[]		∞	∞	∞	∞	∞	∞	∞	7	∞	∞	∞	0	∞

Graph Representation:



STEP 9: L -> I

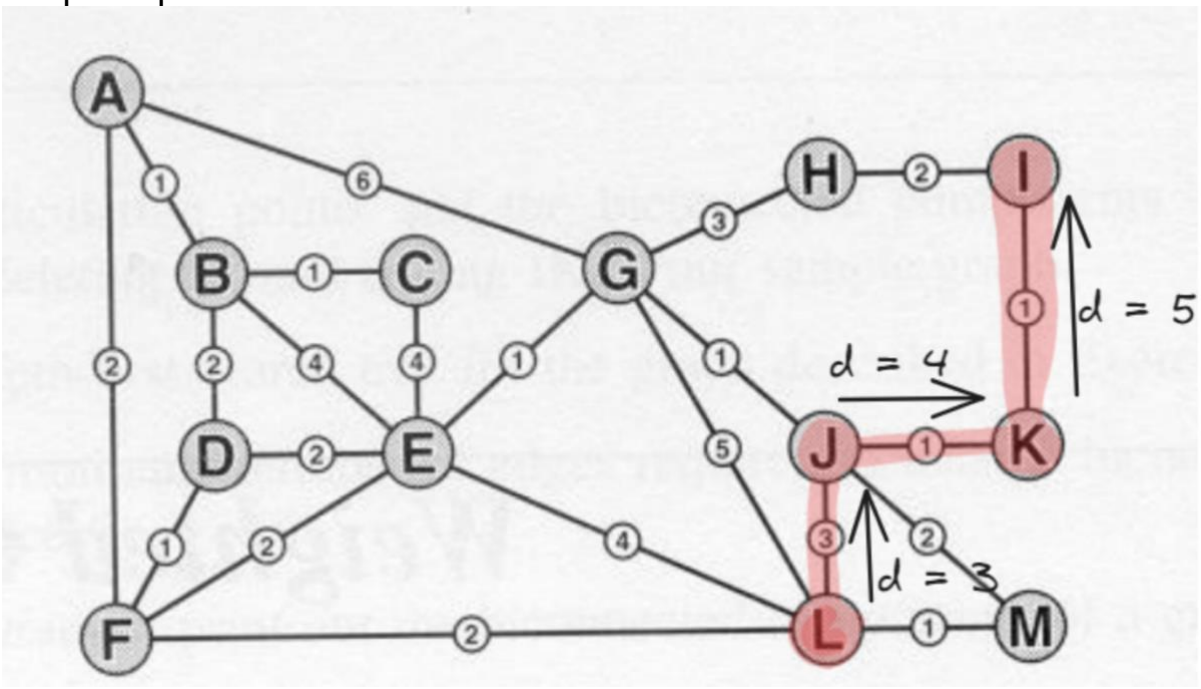
Shortest Path: L -> J -> K -> I

Distance from L = 5

Heap: J K L M

		A	B	C	D	E	F	G	H	I	J	K	L	M
	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Parent[]		0	0	0	0	0	0	0	0	L	0	0	0	0
Dist[]		∞	∞	∞	∞	∞	∞	∞	∞	5	∞	∞	0	∞

Graph Representation:



STEP 10: L -> J

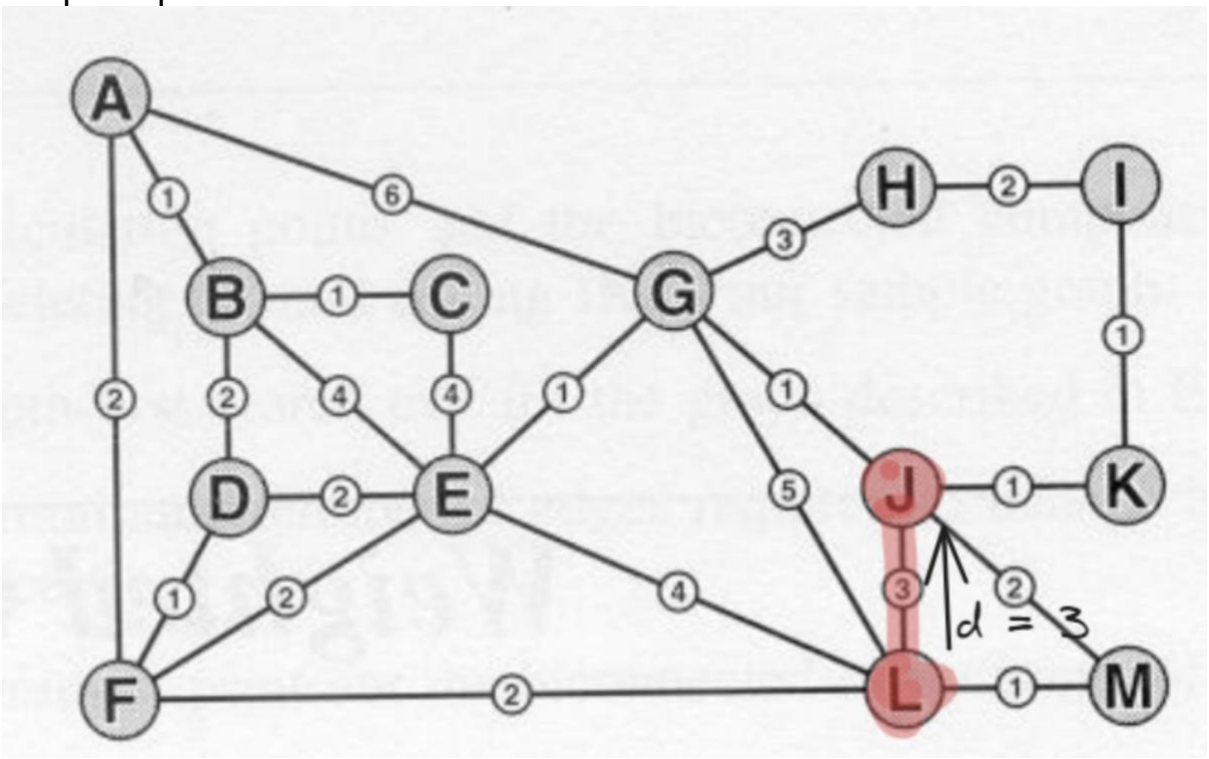
Shortest Path: L -> J

Distance = 3

Heap: K L M

		A	B	C	D	E	F	G	H	I	J	K	L	M
	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Parent[]		0	0	0	0	0	0	0	0	0	L	0	0	0
Dist[]		∞	∞	∞	∞	∞	∞	∞	∞	∞	3	∞	0	∞

Graph Representation:



STEP 11: L -> K

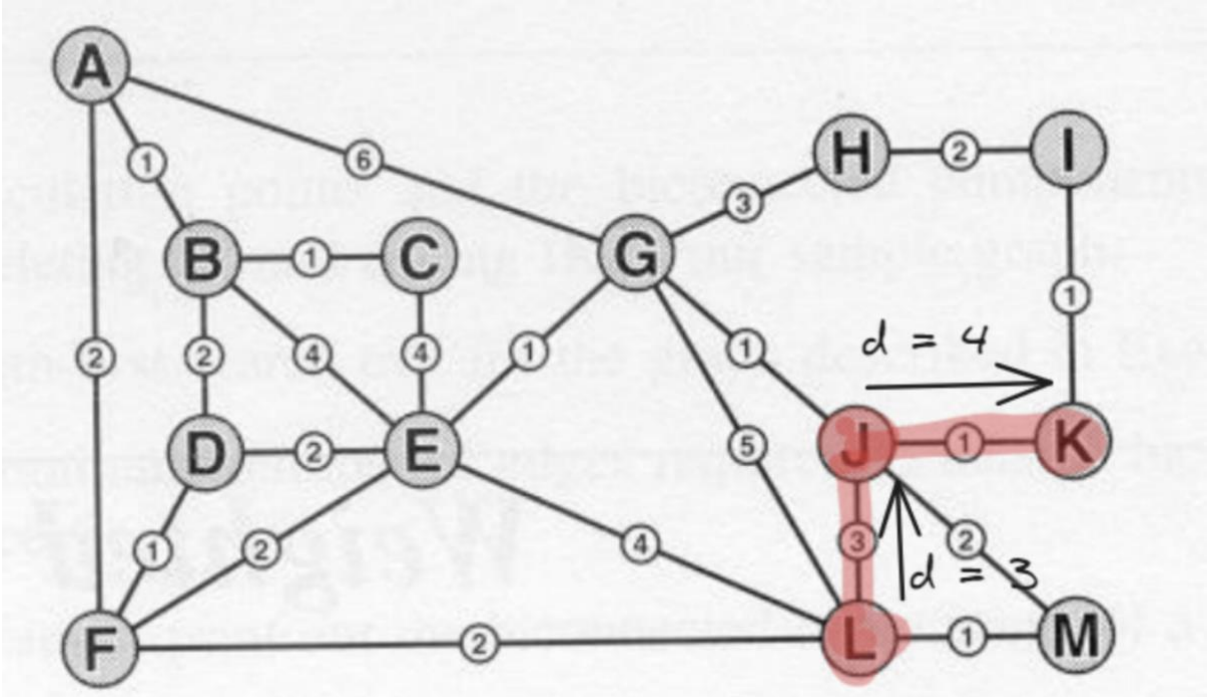
Shortest Path: L -> J -> K

Distance from L = 4

Heap: M

		A	B	C	D	E	F	G	H	I	J	K	L	M
	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Parent[]		0	0	0	0	0	0	0	0	0	0	L	0	0
Dist[]		∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	4	0	∞

Graph Representation:



STEP 12: L -> M

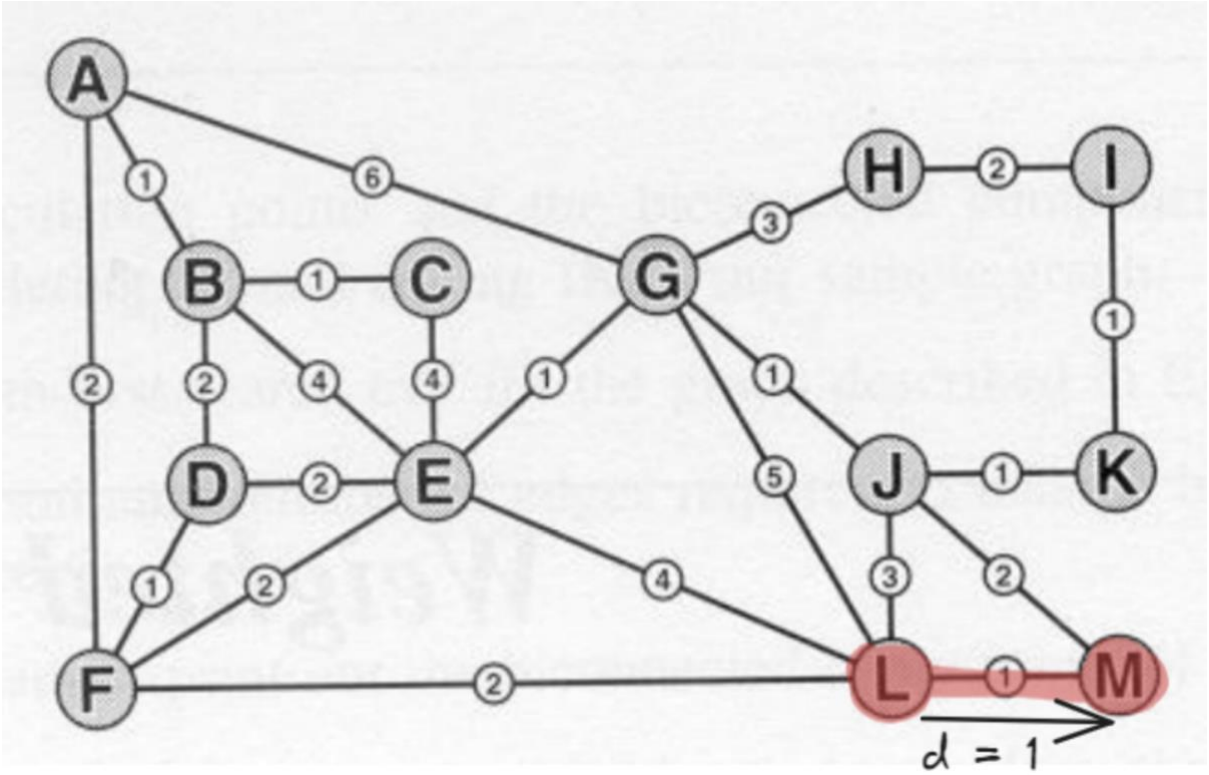
Shortest Path: L ->M

Distance from L = 1

Heap: Empty

		A	B	C	D	E	F	G	H	I	J	K	L	M
	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Parent[]		0	0	0	0	0	0	0	0	0	0	0	0	L
Dist[]		∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	0	1

Graph Representation:



Construction of MST using Kruskal's algorithm

Initial state

Set Representation :

{ A } { B } { C } { D } { E } { F } { G } { H } { I } { J } { K } { L } { M }

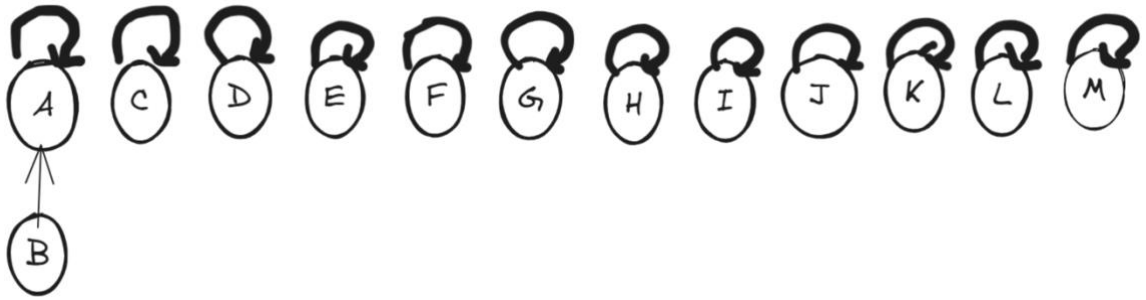
Union-find Partition:

Step 1 : A -- 1 -> B

Set Representation :

{ A, B} { C } { D } { E } { F } { G } { H } { I } { J } { K } { L } { M }

Union-find Partition:

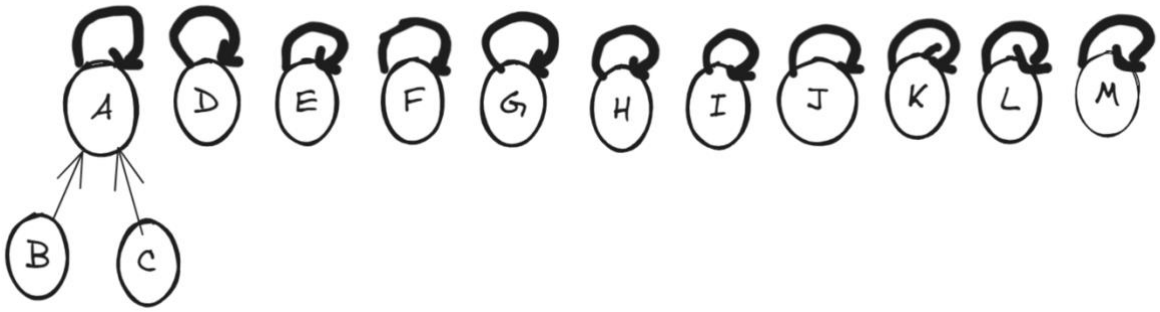


Step 2 : B -- 1 -> C

Set Representation :

{ A, B, C} { D } { E } { F } { G } { H } { I } { J } { K } { L } { M }

Union-find Partition:

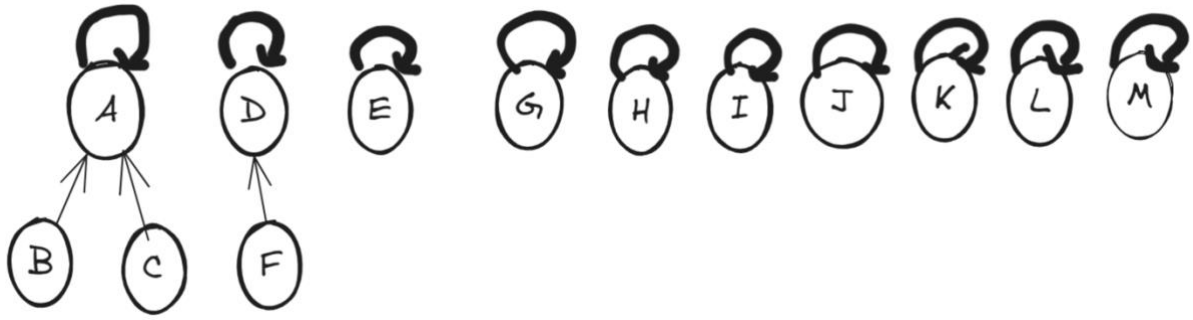


Step 3 : D -- 1 -> F

Set Representation :

{ A, B, C} { D, F} { E } { G } { H } { I } { J } { K } { L } { M }

Union-find Partition:

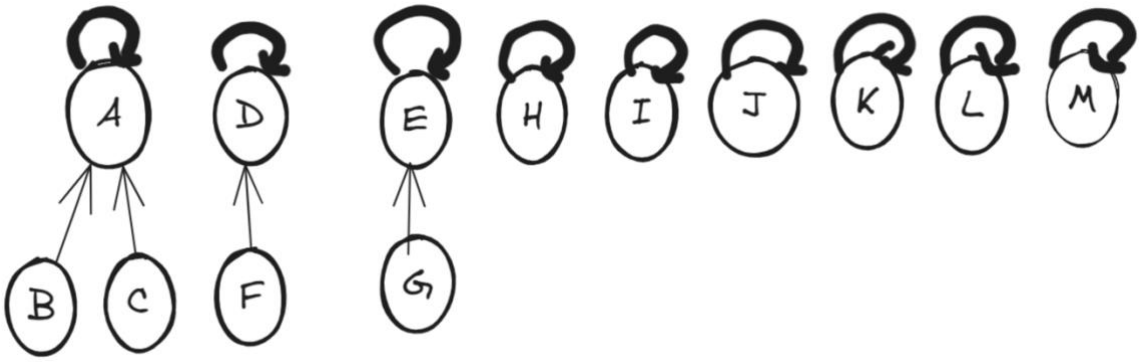


Step 4 : E -- 1 -> G

Set Representation :

{ A, B, C} { D, F} { E, G} { H } { I } { J } { K } { L } { M }

Union-find Partition:

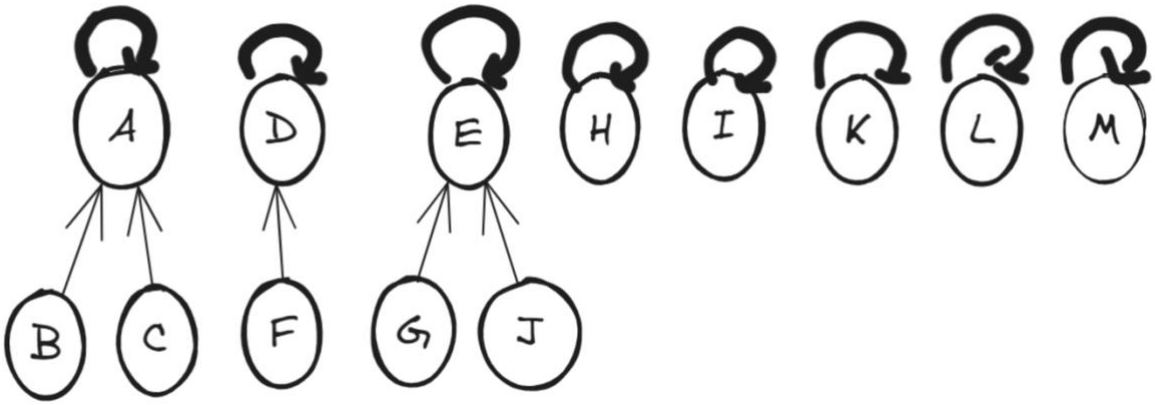


Step 5 : G -- 1 -> J

Set Representation :

{ A, B, C} { D, F} { E, G, J} { H } { I } { K } { L } { M }

Union-find Partition:

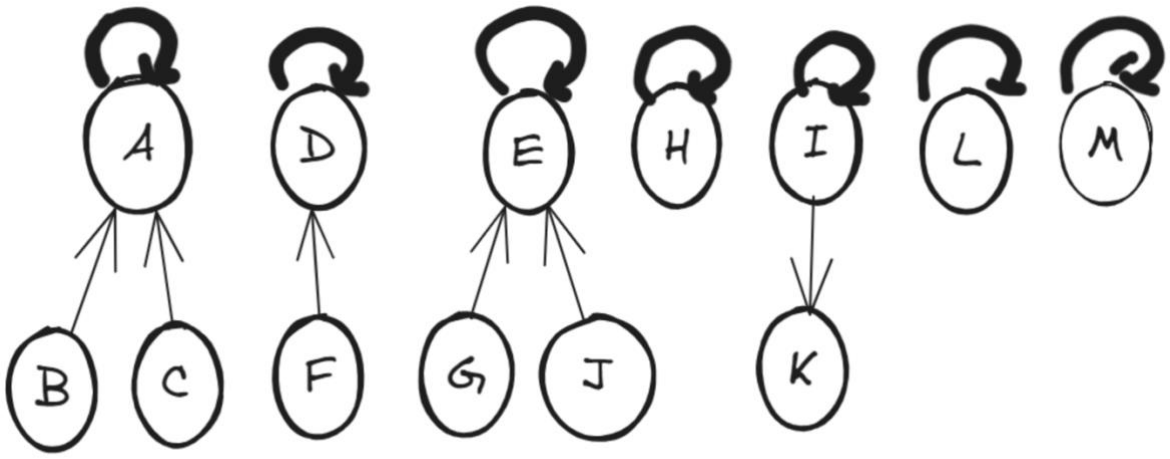


Step 6 : K -- 1 -> I

Set Representation :

{ A, B, C} { D, F} { E, G, J} { H } { I, K} { L } { M }

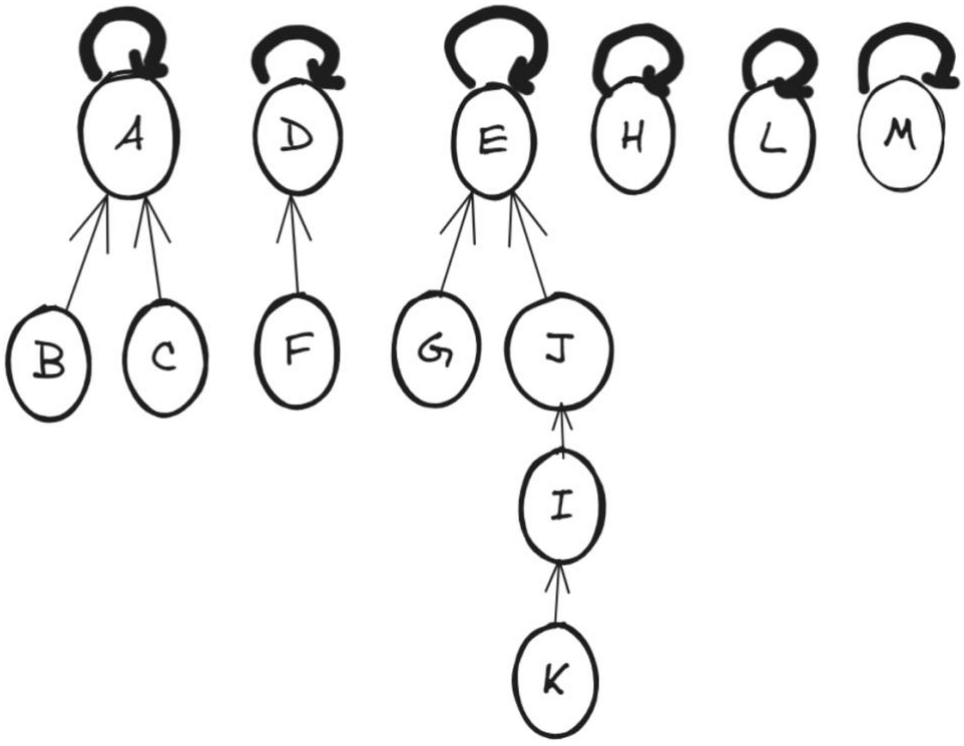
Union-find Partition:



Step 7 : K -- 1 -> J

Set Representation : { A, B, C} { D, F} { E, G, J, I, K} { H } { L } { M }

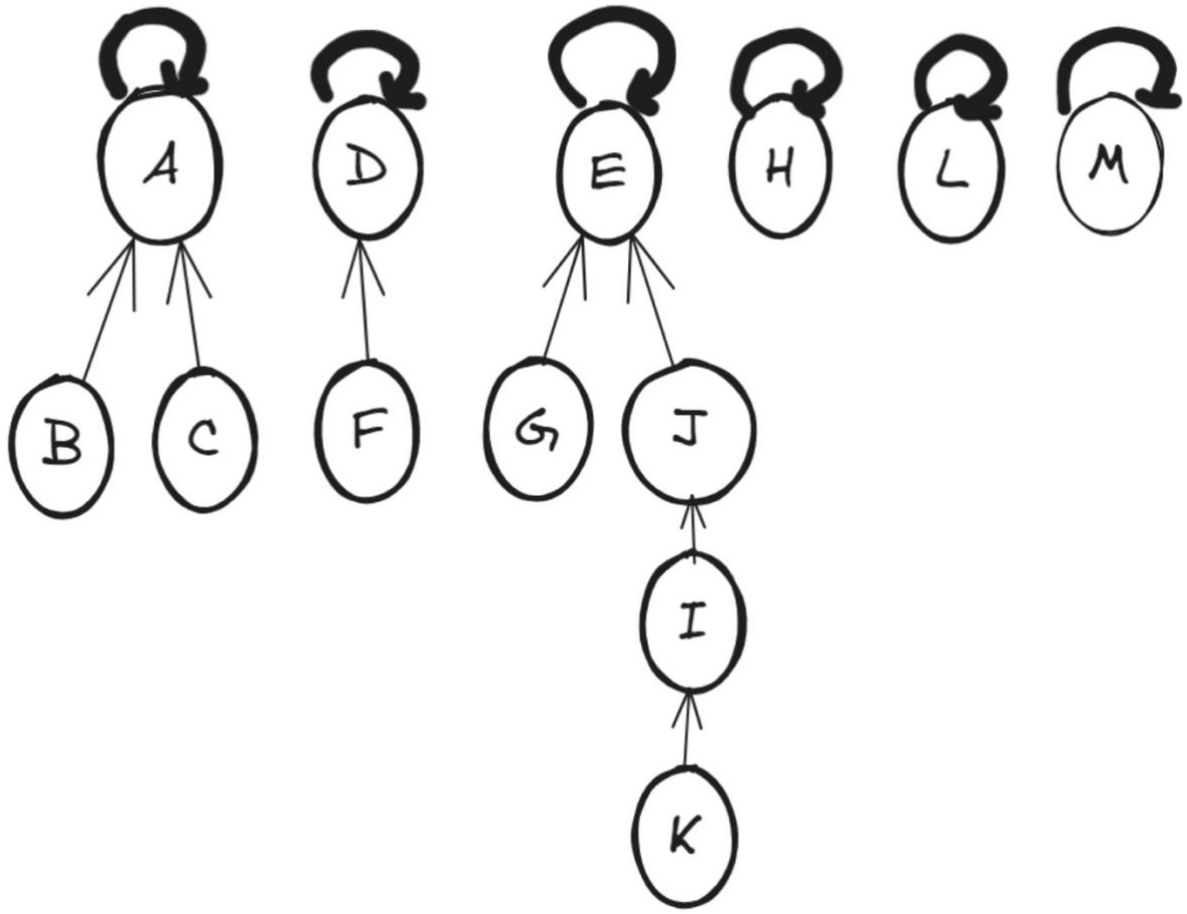
Union-find Partition:



Step 8 : L -- 1 -> M

Set Representation : { A, B, C} { D, F} { E, G, J, I, K} { H } { L, M }

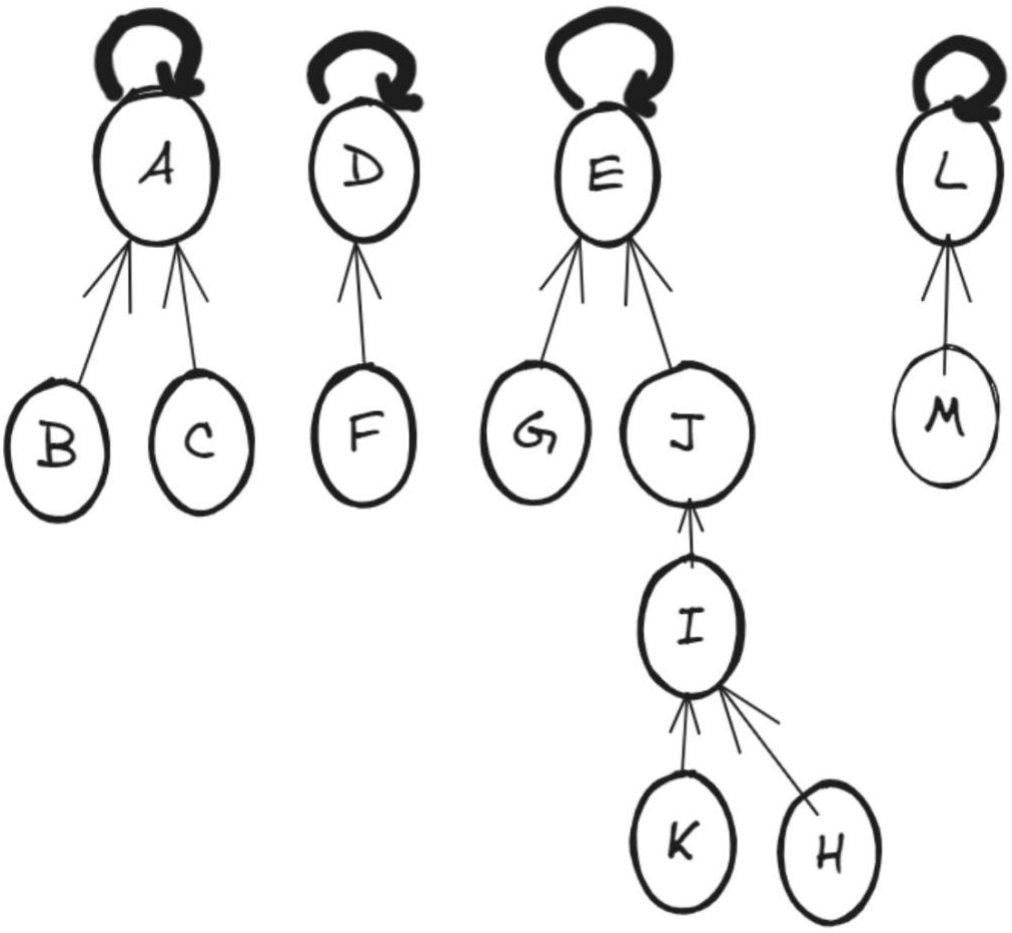
Union-find Partition:



Step 9 : I -- 2 -> H

Set Representation : { A, B, C} { D, F} { E, G, J, I, K, H} { L, M }

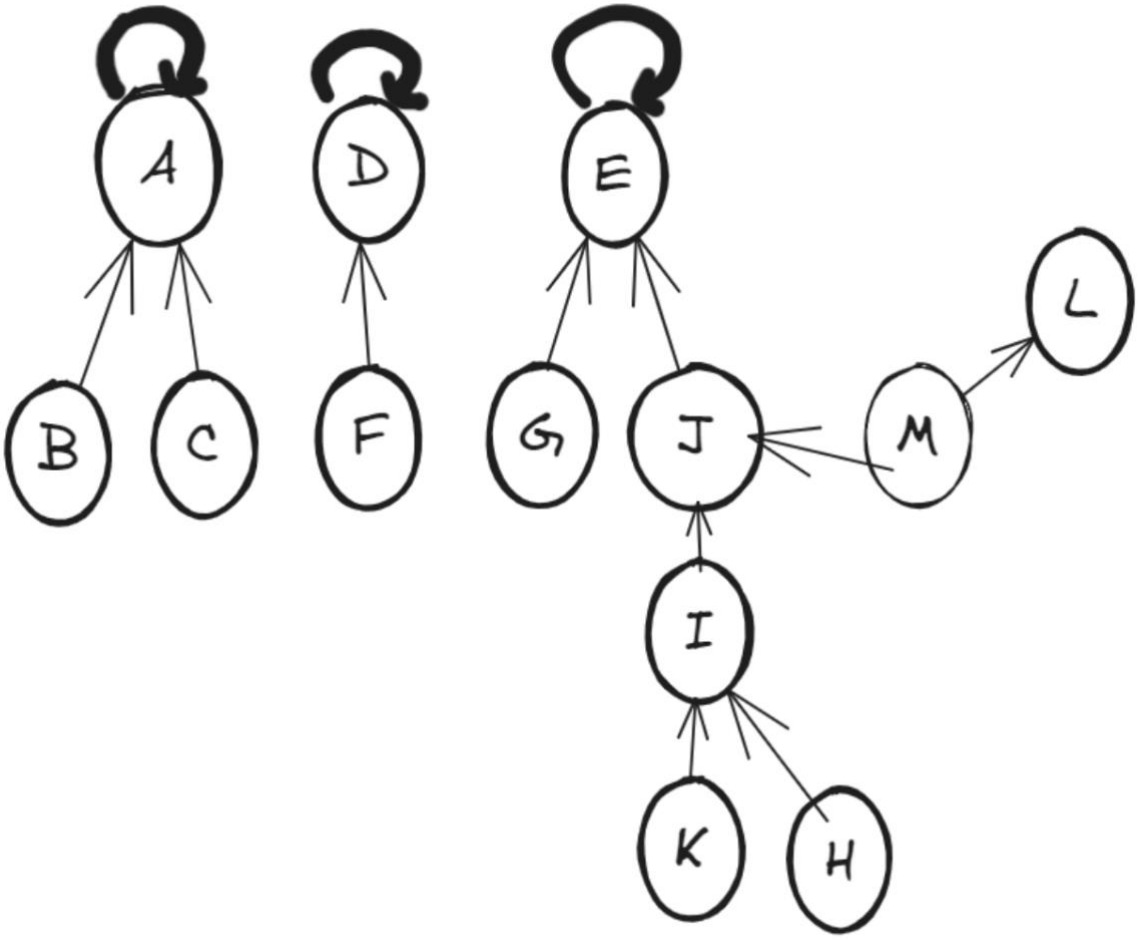
Union-find Partition:



Step 10 : J -- 2 -> M

Set Representation : { A, B, C} { D, F} { E, G, J, I, K, H, L, M }

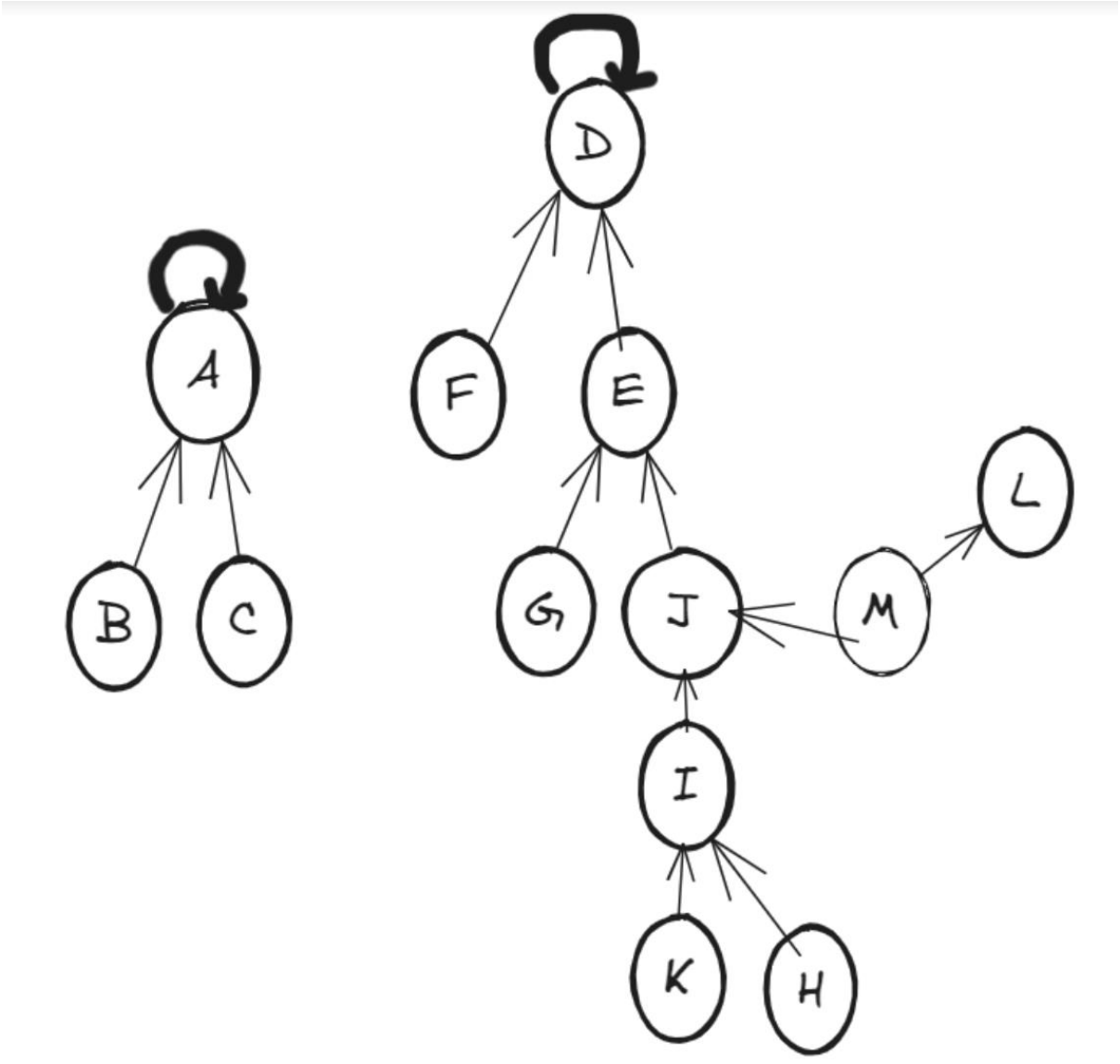
Union-find Partition:



Step 11 : E -- 2 -> D

Set Representation : { A, B, C} { E, G, J, I, K, H, L, M, D, F }

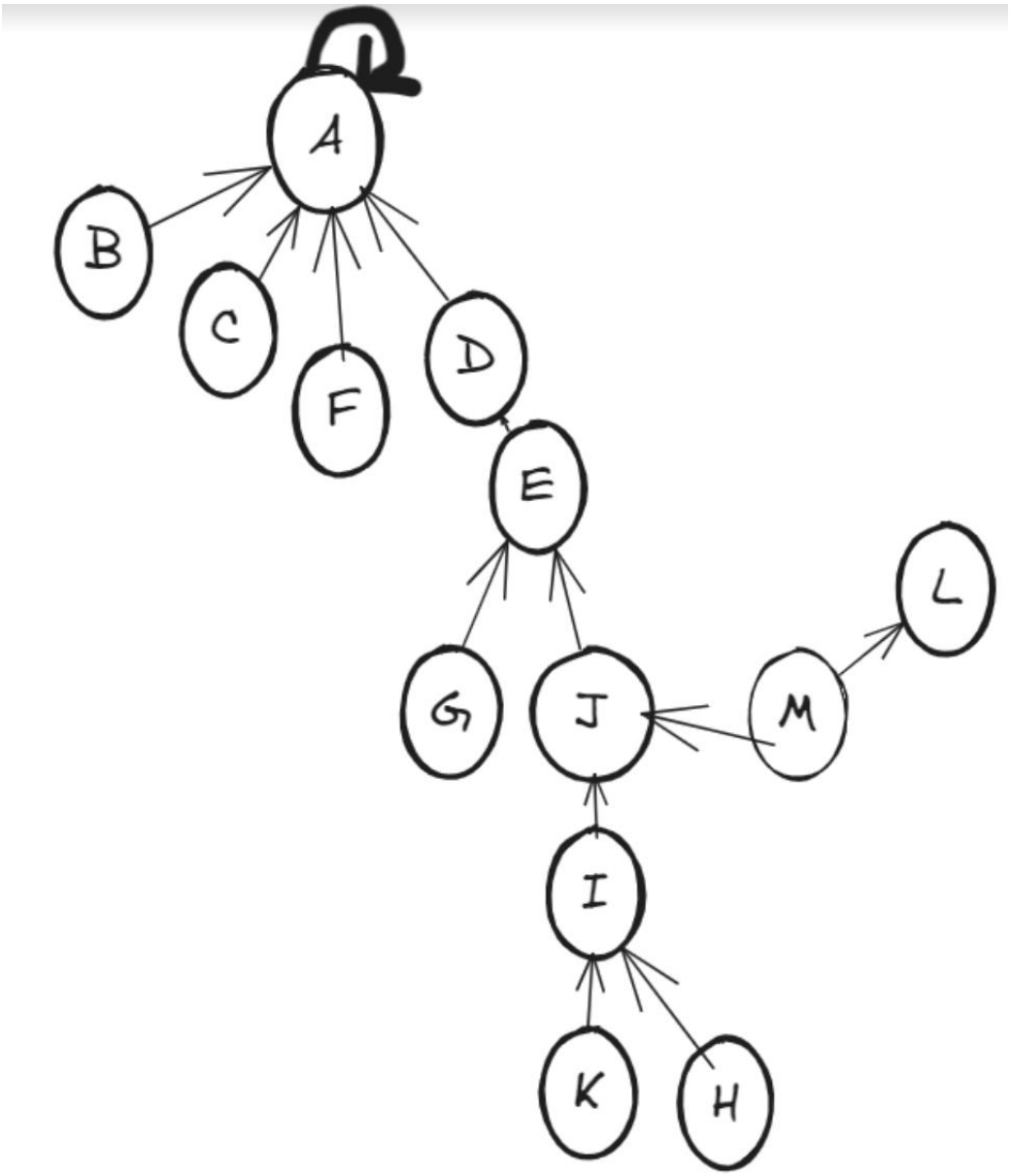
Union-find Partition:



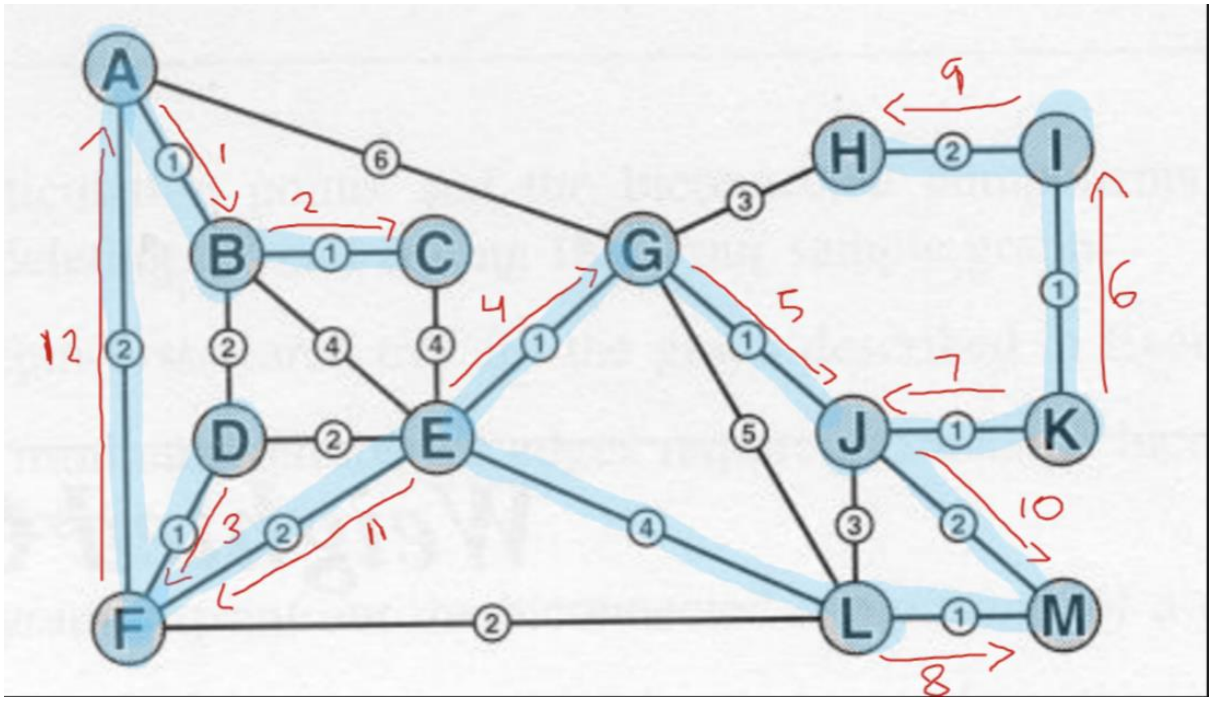
Step 12 : F -- 2 -> A

Set Representation : { A, B, C, E, G, J, I, K, H, L, M, D, F }

Union-find Partition:



Kruskal’s MST superimposed on graph:



Output for GraphLists.java (from vertex L)

```
Enter the name of the text file which has the graph: wGraph1.txt
Enter the beginning vertex (as a number): 12

Depth First Search beiginning from vertex L:
L M J K I H G E F D B C A

Breadth First Search starting from vertex L:
L M J G F K H E A D I C B

Prim's Minimum spanning tree beginning from vertex L:

Step 1: Vertex L added to MST
Step 2: Edge L -> M added to MST
Step 3: Edge L -> F added to MST
Step 4: Edge F -> D added to MST
Step 5: Edge F -> A added to MST
Step 6: Edge A -> B added to MST
Step 7: Edge B -> C added to MST
Step 8: Edge M -> J added to MST
Step 9: Edge J -> K added to MST
Step 10: Edge J -> G added to MST
Step 11: Edge K -> I added to MST
Step 12: Edge G -> E added to MST
Step 13: Edge I -> H added to MST

Total weight of MST = 16


Minimum Spanning tree parent array is:
A --> F
B --> A
C --> B
D --> F
E --> G
F --> L
G --> J
H --> I
I --> K
J --> M
K --> J
L --> ?
M --> L


Dijkstra's SPT beginning from vertex L:
Vertex          Distance                Pathway
A               4               L --> F --> A
B               5               L --> F --> D --> B
C               6               L --> F --> D --> B --> C
D               3               L --> F --> D
E               4               L --> F --> E
F               2               L --> F
G               4               L --> J --> G
H               7               L --> J --> G --> H
I               5               L --> J --> K --> I
J               3               L --> J
K               4               L --> J --> K
M               1               L --> M
(base) fatima@Fatimas-Air Algorithms Assignment %
```

Output of KruskalTrees.java

```
(base) fatima@Fatimas-Air Algorithms Assignment % cd "/Users/fatima/Desktop/Algorithms Assignment/" && javac KruskalTrees.java && java KruskalT
s
Parts[] = 13 21
Reading edges from text file
Edge A--(1)--B
Edge A--(2)--F
Edge A--(6)--G
Edge B--(1)--C
Edge B--(2)--D
Edge B--(4)--E
Edge C--(4)--E
Edge D--(2)--E
Edge D--(1)--F
Edge E--(2)--F
Edge E--(1)--G
Edge F--(2)--L
Edge G--(3)--H
Edge G--(1)--J
Edge G--(5)--L
Edge H--(2)--I
Edge I--(1)--K
Edge J--(1)--K
Edge J--(3)--L
Edge J--(2)--M
Edge L--(1)--M
```

```
Edge L--(1)--M

Steps of Kruskal:

A-1-B
Set: {A B }  Set: {C }  Set: {D }  Set: {E }  Set: {F }  Set: {G }  Set: {H }  Set: {I }  Set: {J }  Set: {K }  Set: {L }  Set: {M }

B-1-C
Set: {A B C }  Set: {D }  Set: {E }  Set: {F }  Set: {G }  Set: {H }  Set: {I }  Set: {J }  Set: {K }  Set: {L }  Set: {M }

I-1-K
Set: {A B C }  Set: {D }  Set: {E }  Set: {F }  Set: {G }  Set: {H }  Set: {I K }  Set: {J }  Set: {L }  Set: {M }

D-1-F
Set: {A B C }  Set: {D F }  Set: {E }  Set: {G }  Set: {H }  Set: {I K }  Set: {J }  Set: {L }  Set: {M }

J-1-K
Set: {A B C }  Set: {D F }  Set: {E }  Set: {G }  Set: {H }  Set: {I J K }  Set: {L }  Set: {M }

L-1-M
Set: {A B C }  Set: {D F }  Set: {E }  Set: {G }  Set: {H }  Set: {I J K }  Set: {L M }

E-1-G
Set: {A B C }  Set: {D F }  Set: {E G }  Set: {H }  Set: {I J K }  Set: {L M }

G-1-J
Set: {A B C }  Set: {D F }  Set: {E G I J K }  Set: {H }  Set: {L M }

A-2-F
Set: {A B C D F }  Set: {E G I J K }  Set: {H }  Set: {L M }

H-2-I
Set: {A B C D F }  Set: {E G H I J K }  Set: {L M }

J-2-M
Set: {A B C D F }  Set: {E G H I J K L M }

E-2-F
Set: {A B C D E F G H I J K L M }

Minimum spanning tree from following edges:
Edge A - 1 - B
Edge B - 1 - C
Edge I - 1 - K
Edge D - 1 - F
Edge J - 1 - K
Edge L - 1 - M
Edge E - 1 - G
Edge G - 1 - J
Edge A - 2 - F
Edge H - 2 - I
Edge J - 2 - M
Edge E - 2 - F

Weight of MST: 16
(base) fatima@Fatimas-Air Algorithms Assignment %
```

Reflection / what I have learned

Implementing Prim's, Kruskal's, and Dijkstra's algorithms was a great learning experience in graph theory and algorithm design. These algorithms play vital roles in computer science, especially in optimising networks and planning routes.

Prim's algorithm constructs a minimum spanning tree by selecting the smallest edge connected to the current tree at each step. Kruskal's algorithm, on the other hand, incrementally adds edges to the MST while ensuring there are no cycles. Dijkstra's algorithm efficiently finds the shortest path from a single source to all other vertices in the graph.

To implement these algorithms effectively, I needed to be comfortable with various data structures like arrays, priority queues, and union-find structures. Additionally, choosing the appropriate graph representation, whether adjacency lists or matrices, was crucial for optimising performance.

By analysing the time and space complexity of these algorithms, I gained insights into their scalability.

I learnt that these algorithms have practical applications in real world scenarios such as optimising transportation networks or minimising costs in communication networks. Understanding these practical implications deepened my problem-solving skills.

In conclusion, implementing these algorithms provided valuable insights into graph theory, algorithmic thinking, and their practical applications in various domains.