MST Assignment

Student Name: Ryan Deguara

Student Number: C20309873

Introduction:

Graph is created by the user who is prompted to enter the text file containing the contents of a graph that will be used by both algorithms (Prim's MST and Kruskal's MST).

Prim's MST:

Algorithm that finds a minimum spanning tree (MST) for a connected weighted undirected graph. It finds the edges that forms a tree that includes every vertex where the total weight of all the edges in the tree is minimized. The algorithm continuously increases the size of a tree, one edge at a time, starting with a tree consisting of a single vertex, until it spans all vertices (It finds the local optimum in hopes of finding the global optimum). The program uses heaps and linked lists to form the graph matrix from the text file (Graph1.txt). Starts from chosen vertex and keeps adding edges with the lowest weight until all vertices have been spanned.

Steps:

1. Initializes the MST with the chosen vertex.
2. Finds all edges that connect the tree to new vertices, find the minimum and add it to the tree where the new vertex doesn't equal the old one.
3. Repeat step 2 until new vertices and edges describe the MST.

Kruskal's MST:

Kruskal's algorithm is an algorithm in graph theory that finds a minimum spanning tree for a connected weighted graph. This means it finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized. If the graph is not connected, then it

finds a minimum spanning forest (a minimum spanning tree for each connected component).

To implement this algorithm a way to represent sets, to find which set a vertex is in and to get the union of two sets is required. We refer to a data structure which supports this as a Union-Find data structure. Two standard implementations are available: Disjoint-set linked lists and Disjoint-set trees, We are using disjoint-set trees in this approach.

Union-Find is a data structure that keeps track of a set of elements partitioned into a number of disjoint (nonoverlapping) subsets. In the Kruskal's Algorithm, Union Find Data Structure is used as a subroutine to find the cycles in the graph, which helps in finding the minimum spanning tree. When trees used to represent the disjoint union-find sets, the tree root is used to label the set. All the vertices on a tree are considered to be in the same set.

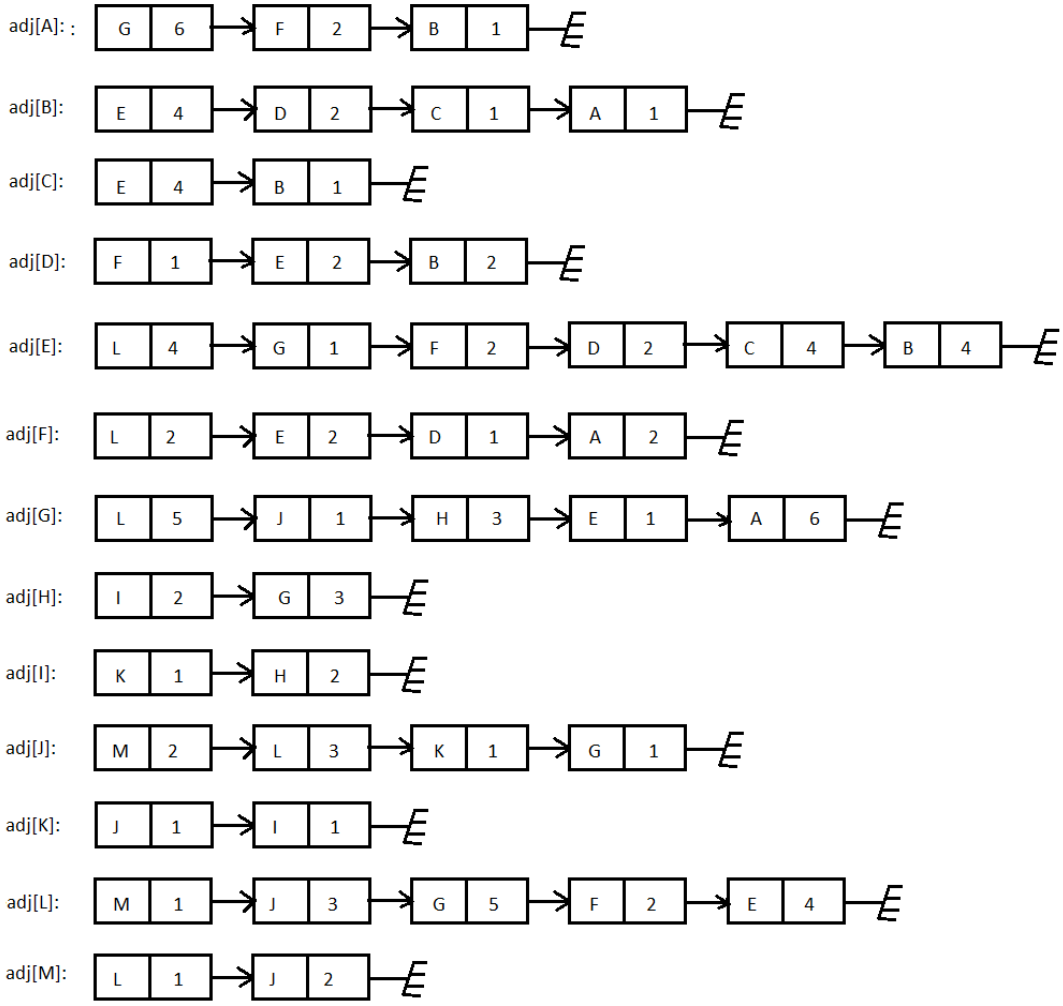
Two operations: findSet() – Determines which subset a particular element is in. Returns an item from this set that serves as its 'representative' by travelling back the tree from vertex u until the root is found.

Union() – joins two subsets into a single subset

Adjacency List:

	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	4	2	5	0	0	3	0	@	1
Adj[M]	0	0	0	0	0	0	0	0	0	2	0	1	@

Graph representation of adjacency list:



Construction of the MST using Prim's algorithm:

Initial state starting from vertex L:

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	11	12
Parent		0	0	0	0	0	0	0	0	0	0	0	0	0
Dist		0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞

Step 0: Start With L vertex

Heap:

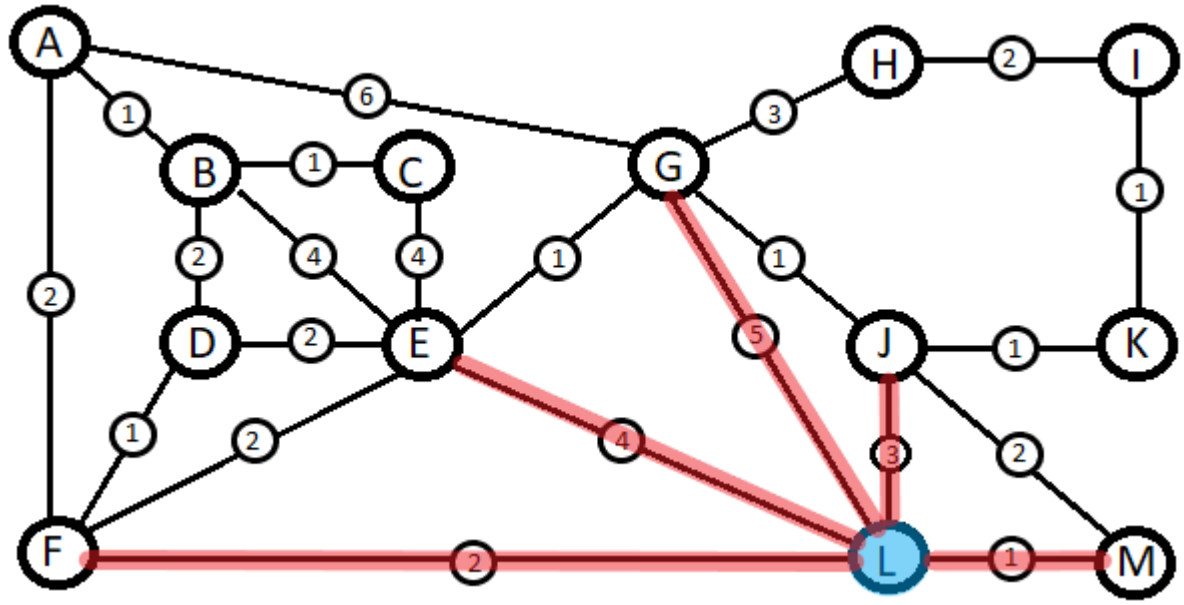
E4

F2
G5
J3
M1

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	11	12
Parent[]		0	0	0	0	L	L	L	0	0	L	0	@	L
Dist[]		0	∞	∞	∞	4	2	5	∞	∞	3	∞	0	1

Graph representation:



Step 1: Next is M

Heap:

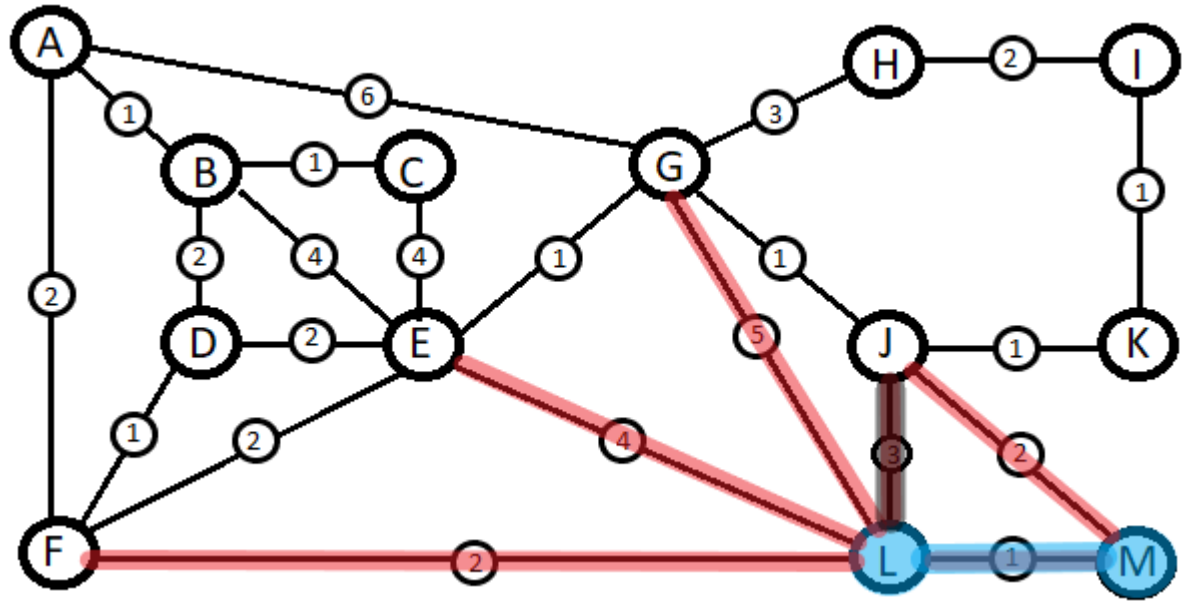
E4
F2
G5
J2

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	11	12
Parent[]		0	0	0	0	L	L	L	0	0	~~L~~ M	0	@	L

Dist[]		0	∞	∞	∞	4	2	5	∞	∞	~~3~~ 2	∞	0	1

Graph representation:



Step 2: Next is F

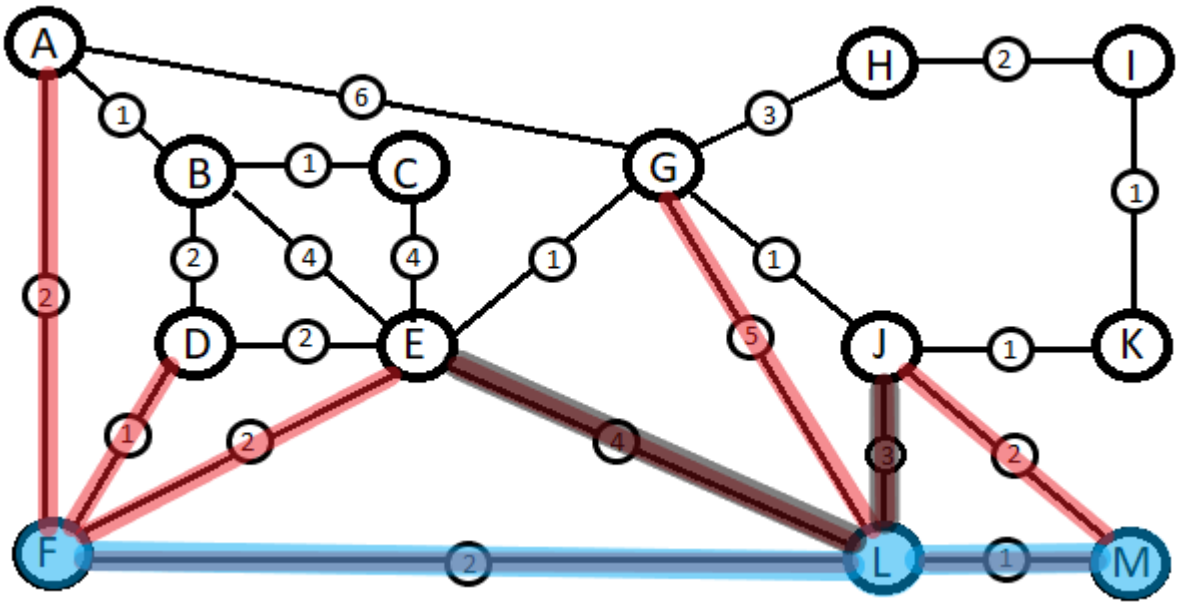
Heap:

A2
D1
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	11	12
Parent[]		F	0	0	F	~~L~~ F	L	L	0	0	~~L~~ M	0	@	L
Dist[]		2	∞	∞	1	~~4~~ 2	2	5	∞	∞	~~3~~ 2	∞	0	1

Graph representation:



Step 3: Next is D

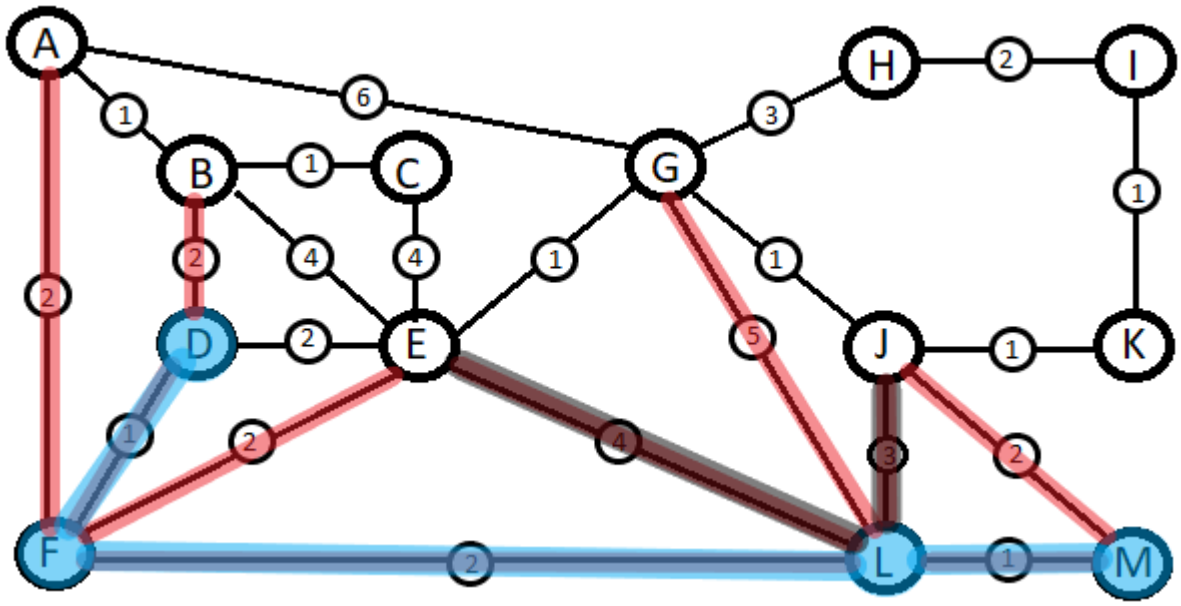
Heap:

A2
B2
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	11	12
Parent[]		F	D	0	F	~~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 4: Next is A

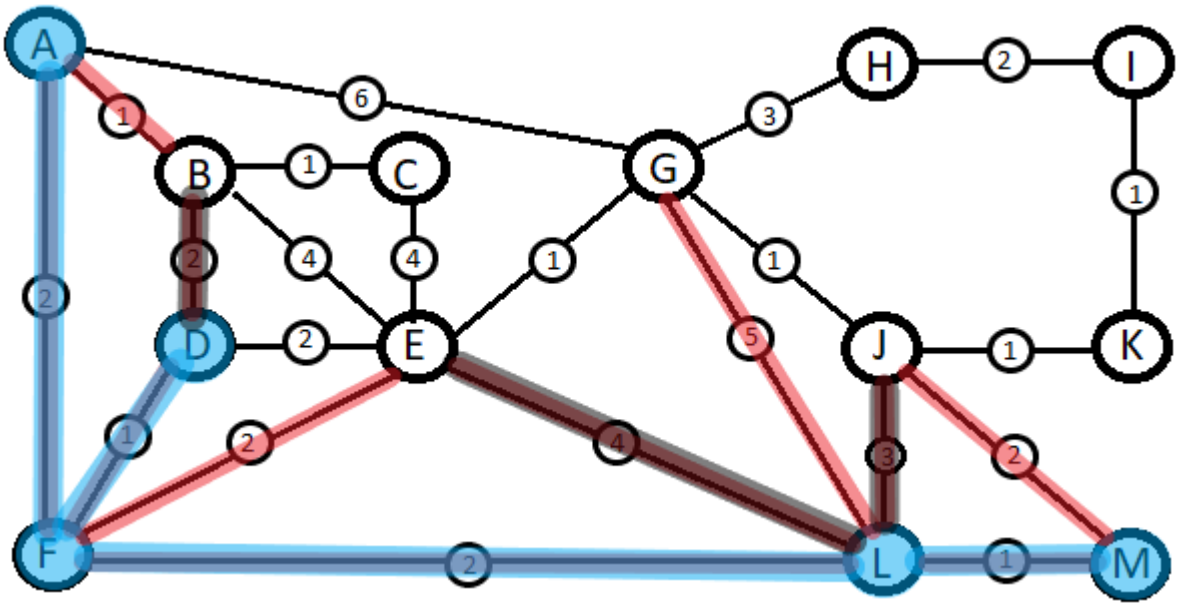
Heap:

B1
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	11	12
Parent[]		F	~~D~~ A	0	F	~~L~~ F	L	L	0	0	~~L~~ M	0	0	L
Dist[]		2	~~2~~ 1	∞	1	~~4~~ 2	2	5	∞	∞	~~3~~ 2	∞	∞	1

Graph representation:



Step 5: Next is B

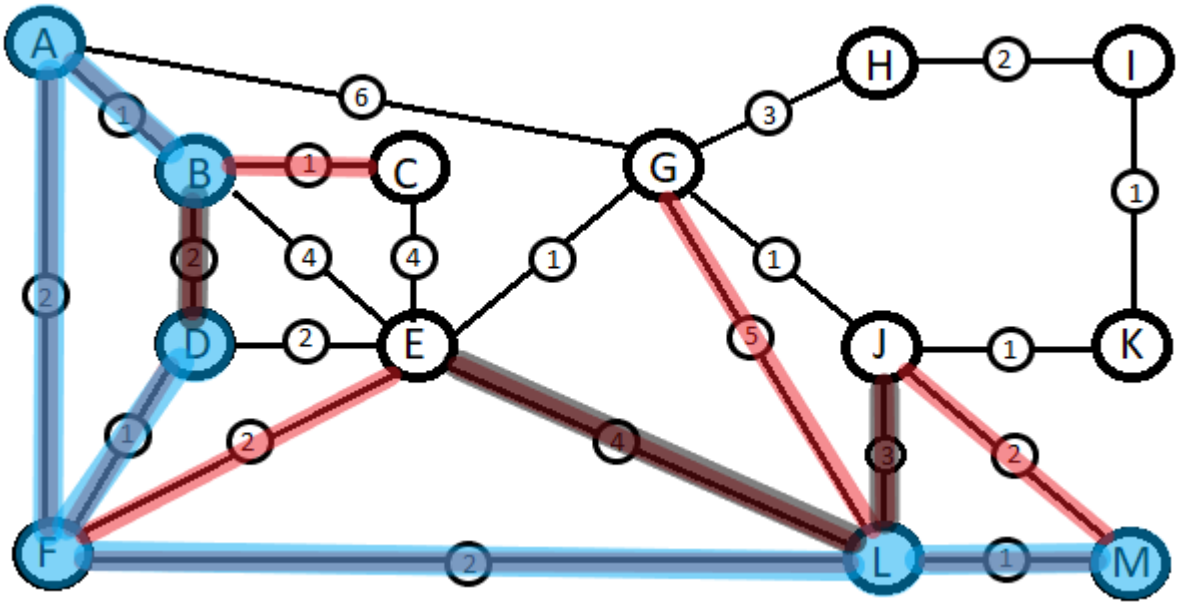
Heap:

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	11	12
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	∞	∞	1

Graph representation:



Step 6: Next is C

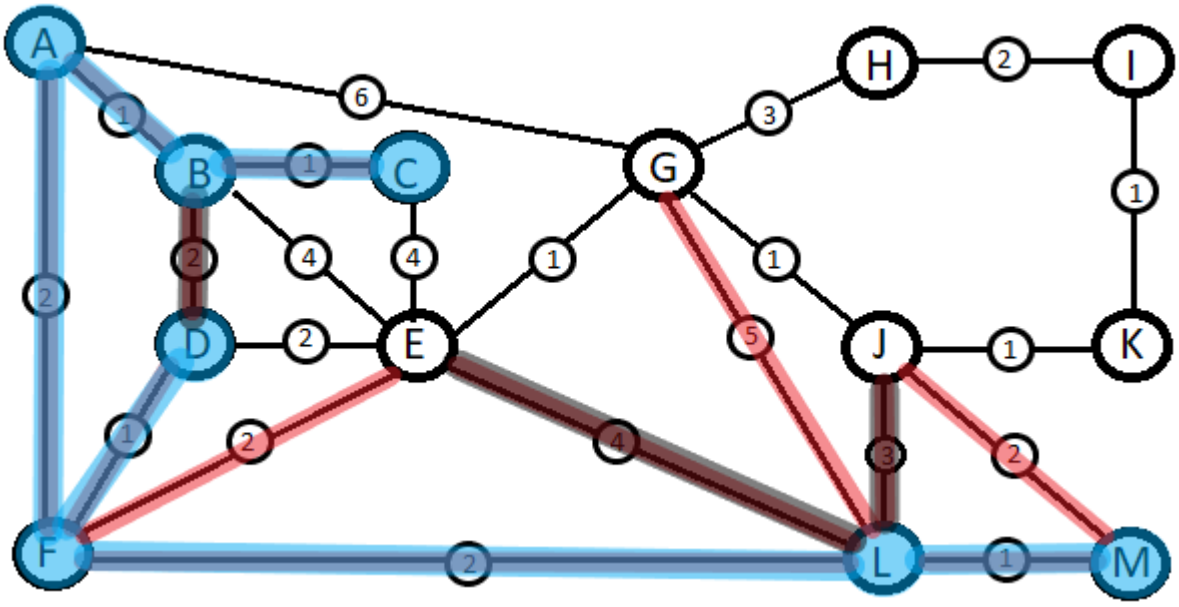
Heap:

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	11	12
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	∞	∞	1

Graph representation:



Step 7: Next is J

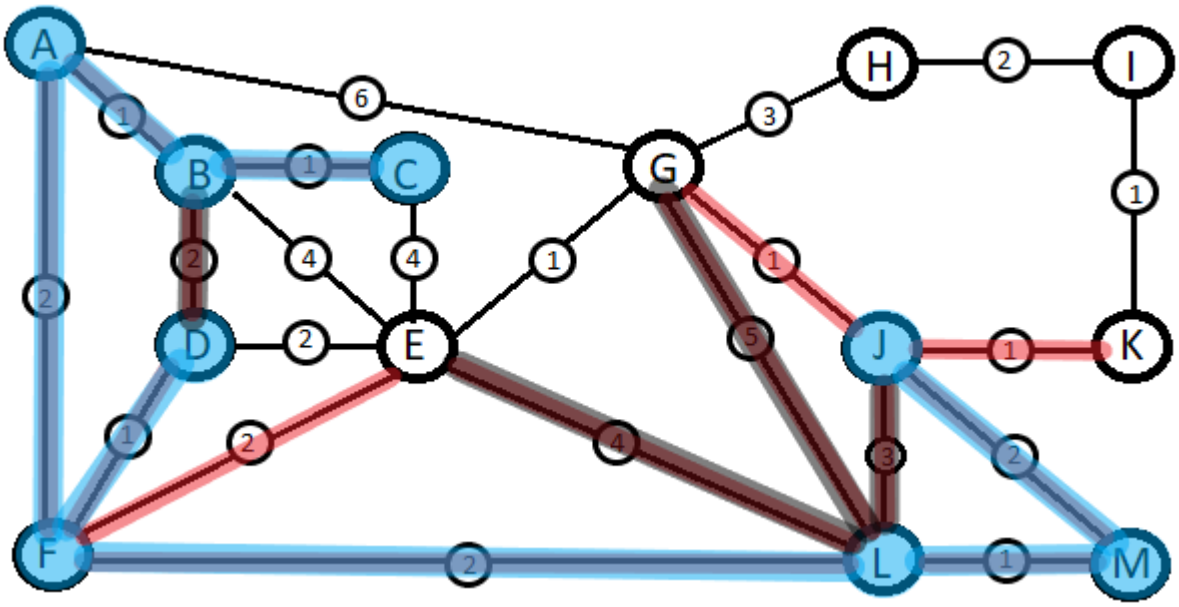
Heap:

E2
G1
K1

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	11	12
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 8: Next is K

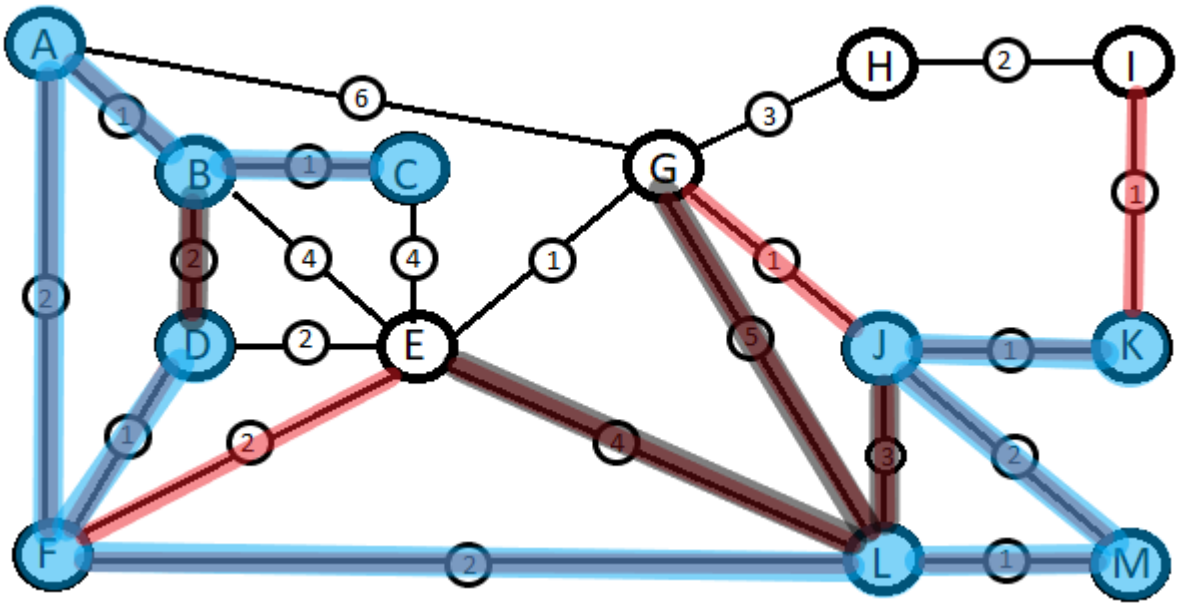
Heap:

E2
G1
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	11	12
Parent[]		F	~~D~~ A	B	F	~~L~~ F	L	~~L~~ J	0	K	~~L~~ M	J	0	L
Dist[]		2	~~2~~ 1	1	1	~~4~~ 2	2	~~5~~ 1	∞	1	~~3~~ 2	1	∞	1

Graph representation:



Step 9: Next is G

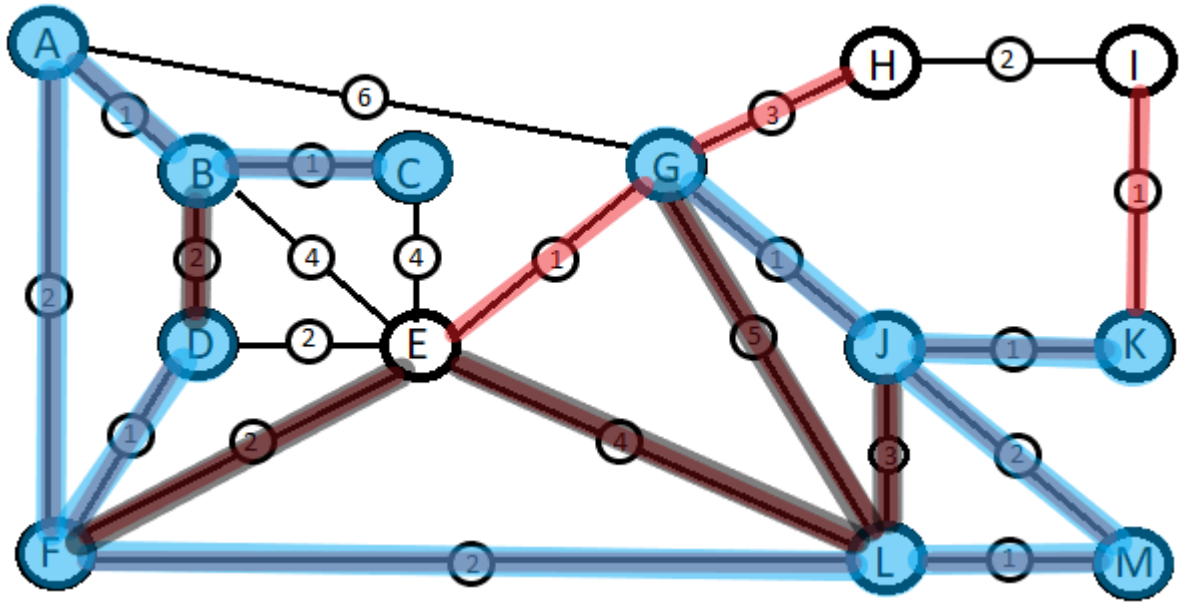
Heap:

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	11	12
Parent[]		F	~~D~~ A	B	F	~~L~~ ~~F~~ G	L	~~L~~ J	G	K	~~L~~ M	J	0	L
Dist[]		2	~~2~~ 1	1	1	~~4~~ ~~2~~ 1	2	~~5~~ 1	3	1	~~3~~ 2	1	∞	1

Graph representation:



Step 10: Next is I

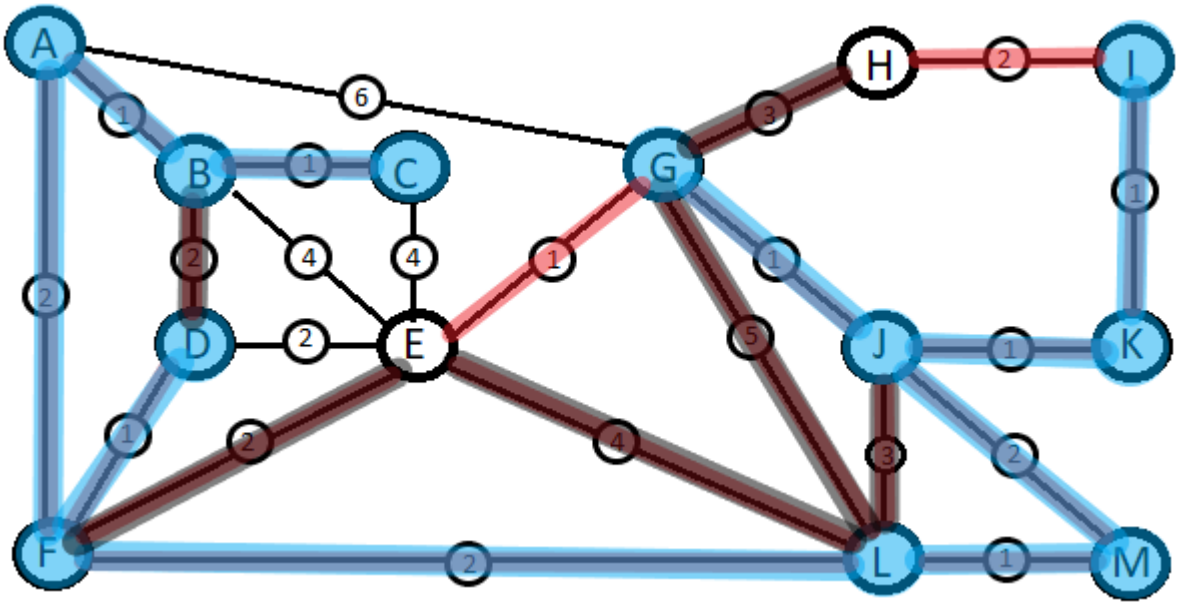
Heap:

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	11	12
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 is E

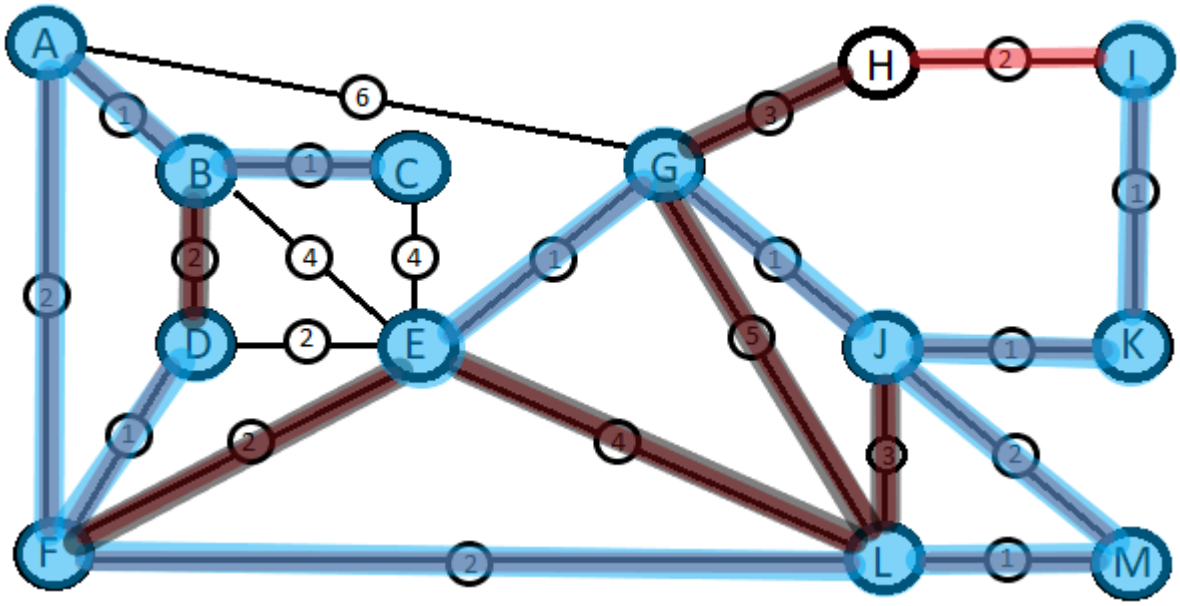
Heap:

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	11	12
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:



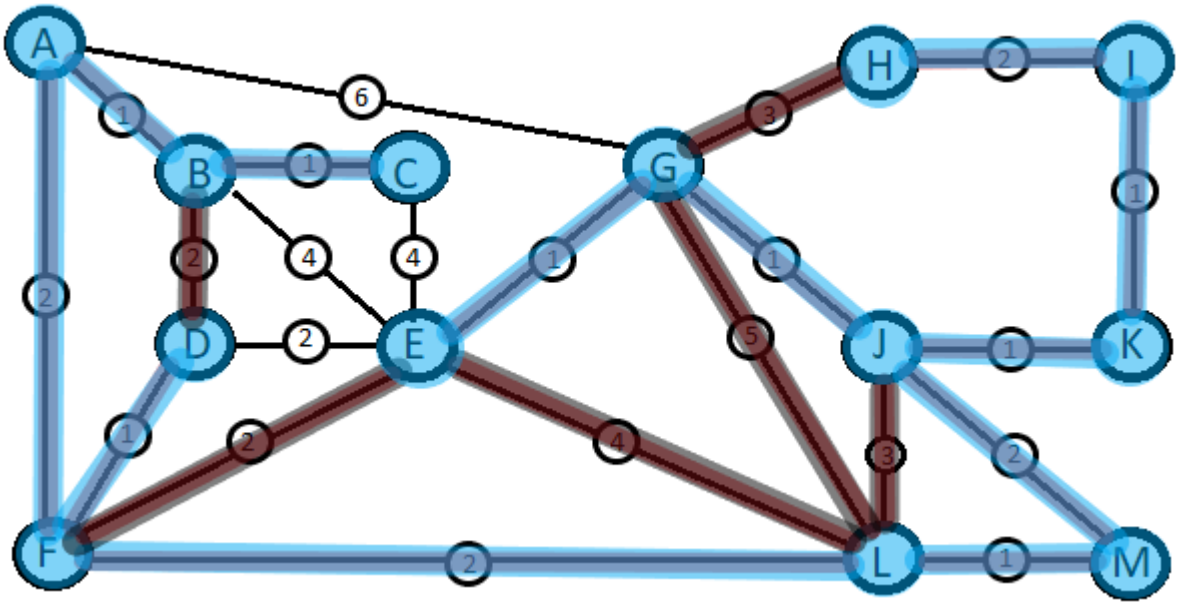
Step 12: Next is H

Heap 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	11	12
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

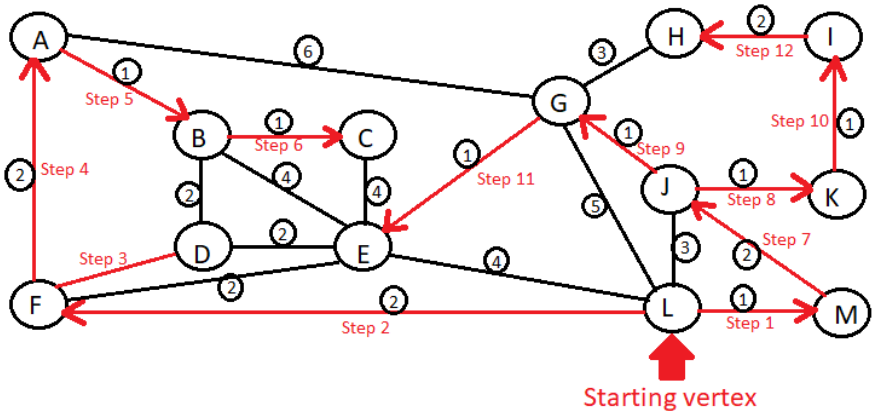
Graph representation:



Prim's MST final 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	11	12
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

Prim's MST graph diagram:



Construction of the MST using Kruskal algorithm:

Initial state:

Sets –

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

Union-find partition –

Step 1:

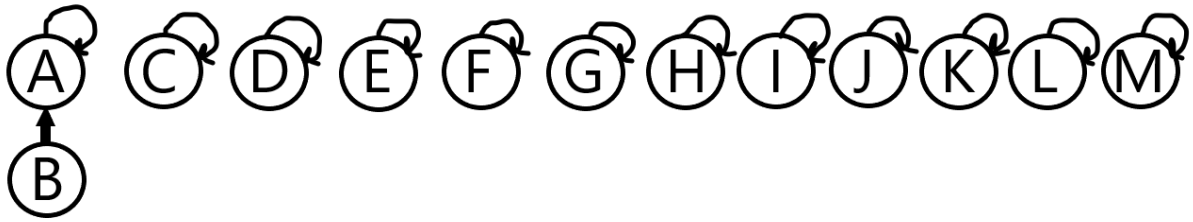
MST_Kruskal -

A-1-B

Sets –

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

Union-find partition –



Step 2:

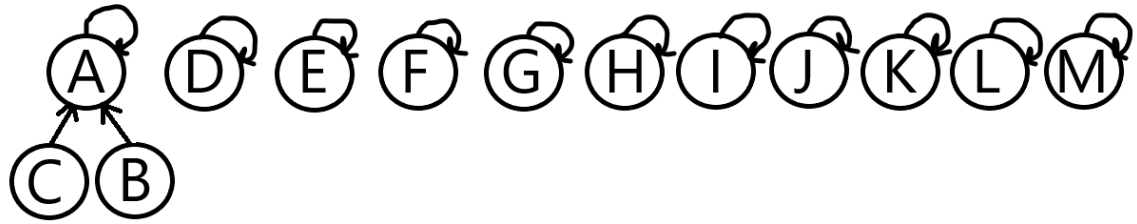
MST_Kruskal -

B-1-C

Sets –

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

Union-Find Partition –



Step 3:

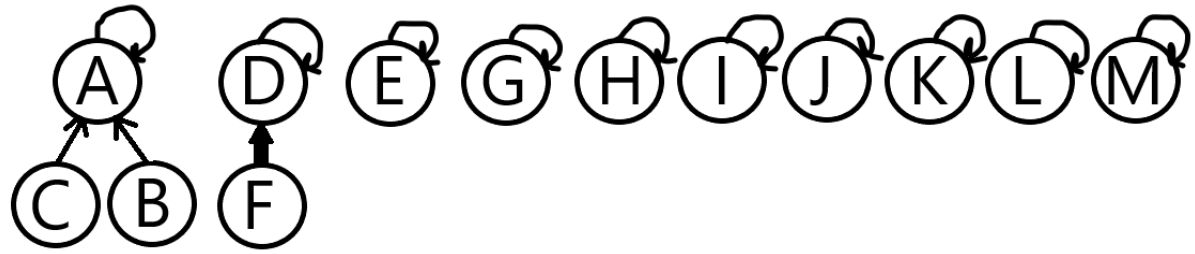
MST_Kruskal -

D-1-F

Sets –

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

Union-Find Partition –



Step 4:

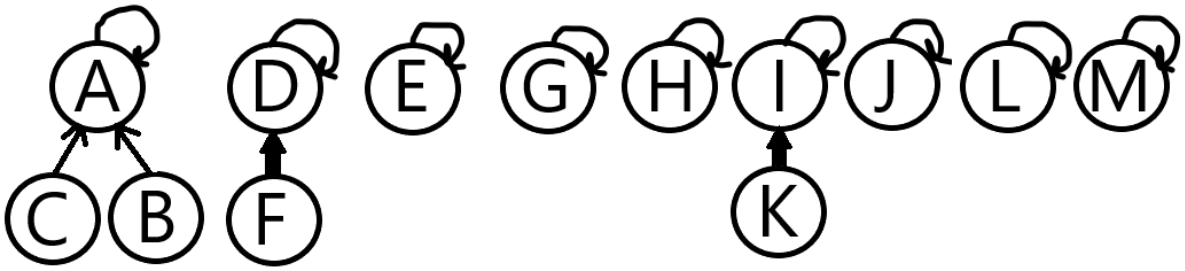
MST_Kruskal -

I-1-K

Sets –

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

Union-Find Partition –



Step 5:

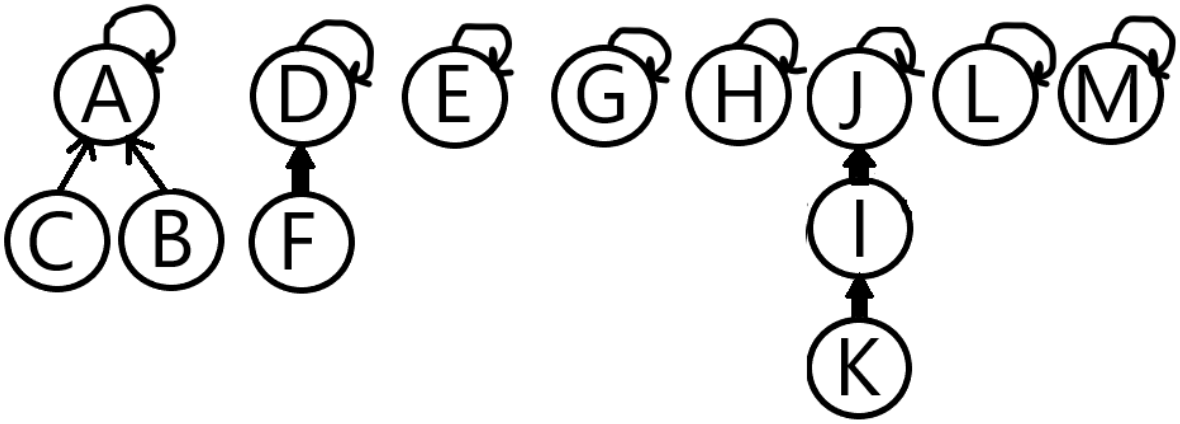
MST_Kruskal -

J-1-K

Sets –

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

Union-Find Partition –



Step 6:

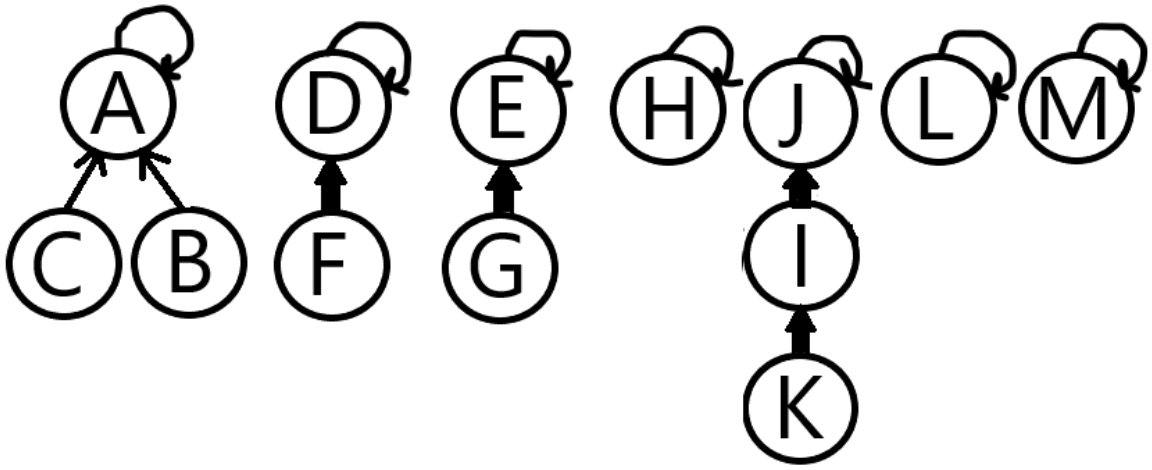
MST_Kruskal -

E-1-G

Sets –

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

Union-Find Partition –



Step 7:

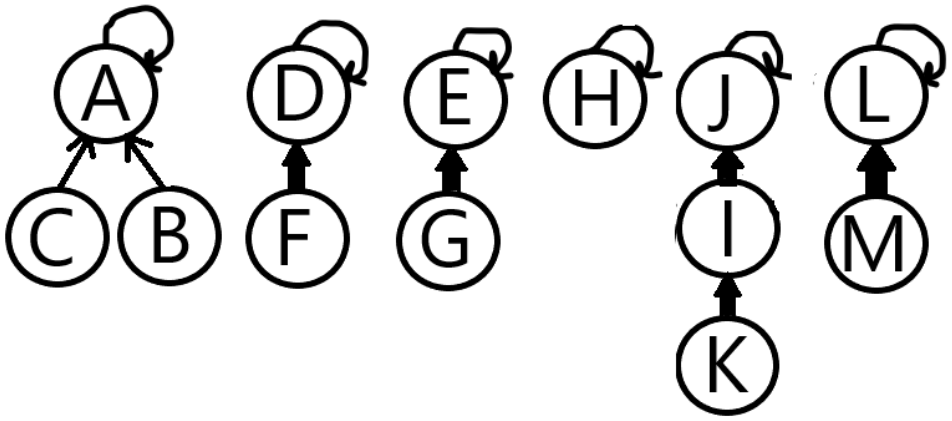
MST_Kruskal -

L-1-M

Sets –

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

Union-Find Partition –



Step 8:

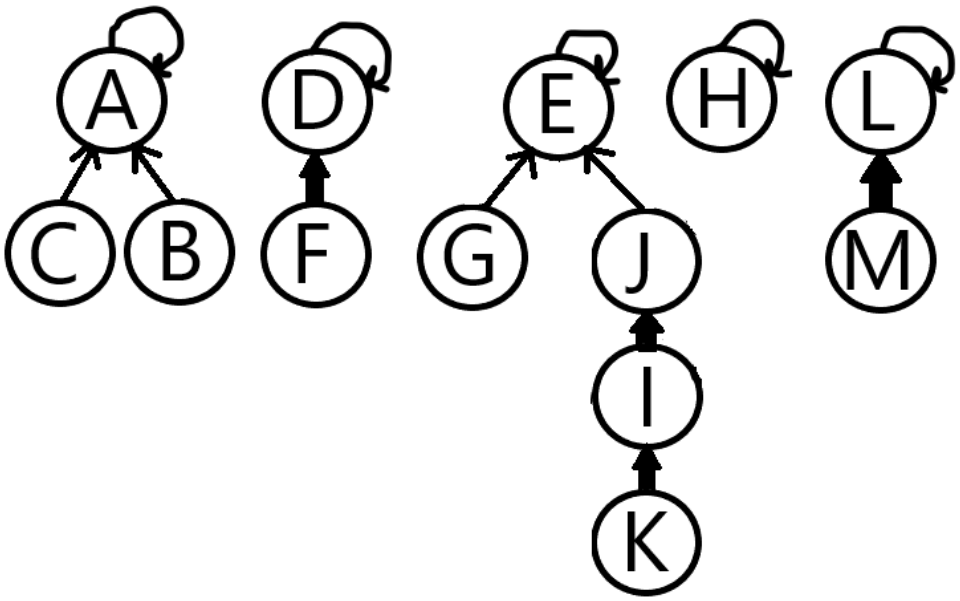
MST_Kruskal -

G-1-J

Sets –

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

Union-Find Partition –



Step 9:

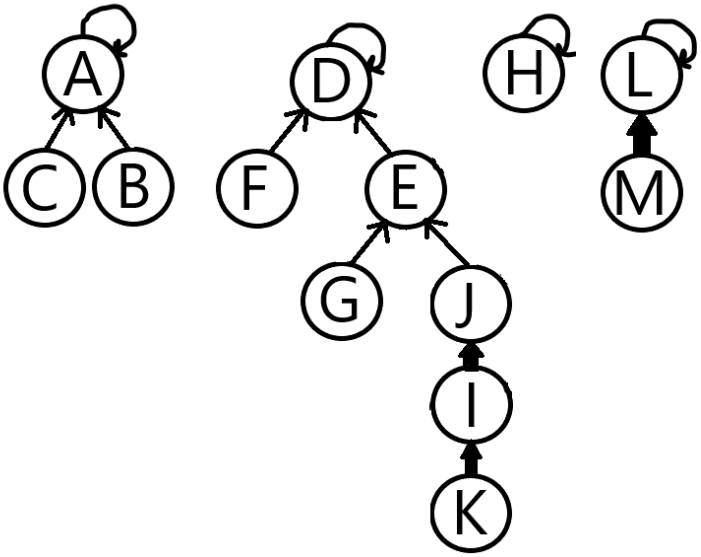
MST_Kruskal -

D-2-E

Sets –

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

Union-Find Partition –



Step 10:

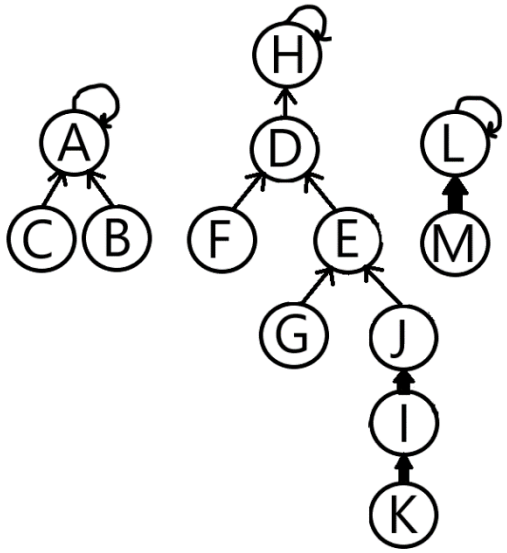
MST_Kruskal -

H-2-I

Sets –

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

Union-Find Partition –



Step 11:

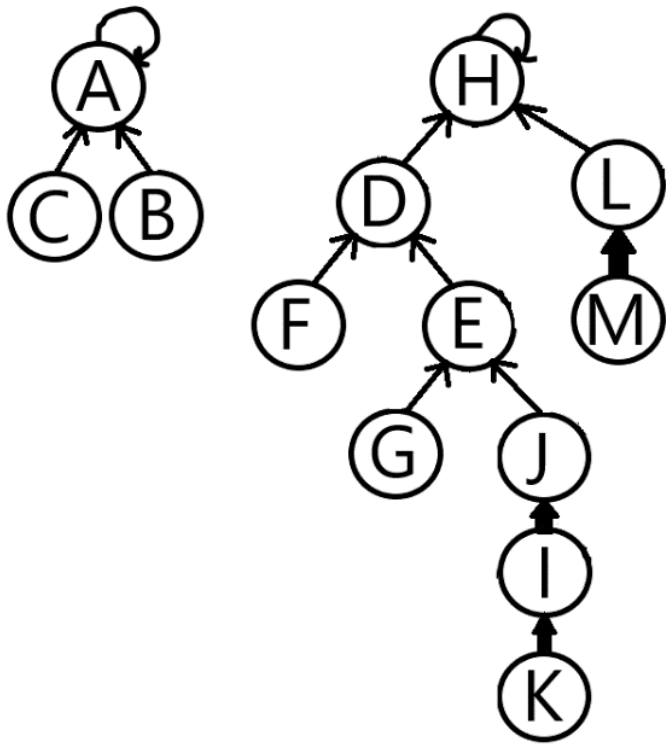
MST_Kruskal -

J-2-M

Sets –

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

Union-Find Partition –



Step 12:

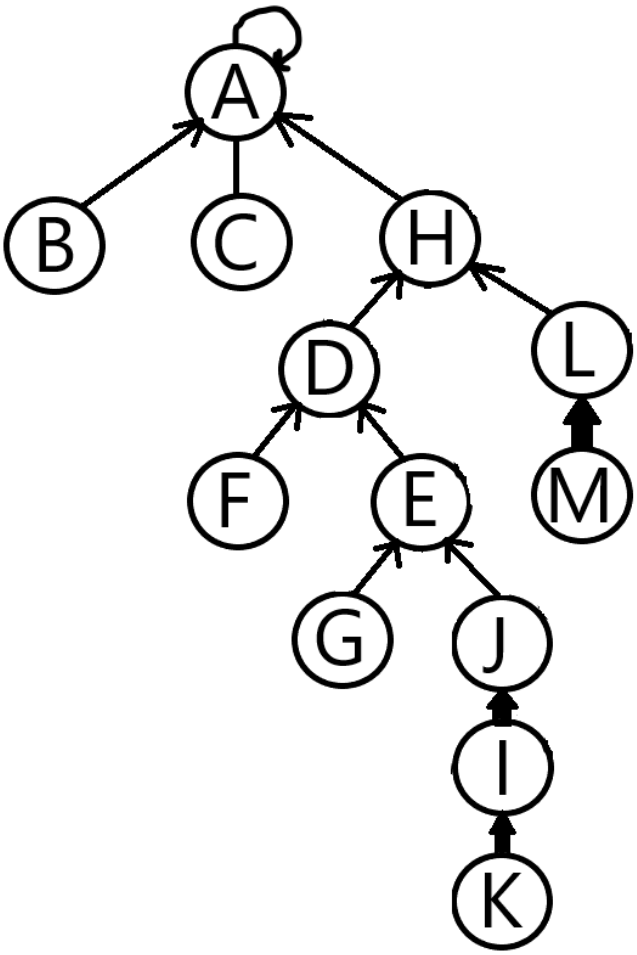
MST_Kruskal -

A-2-F

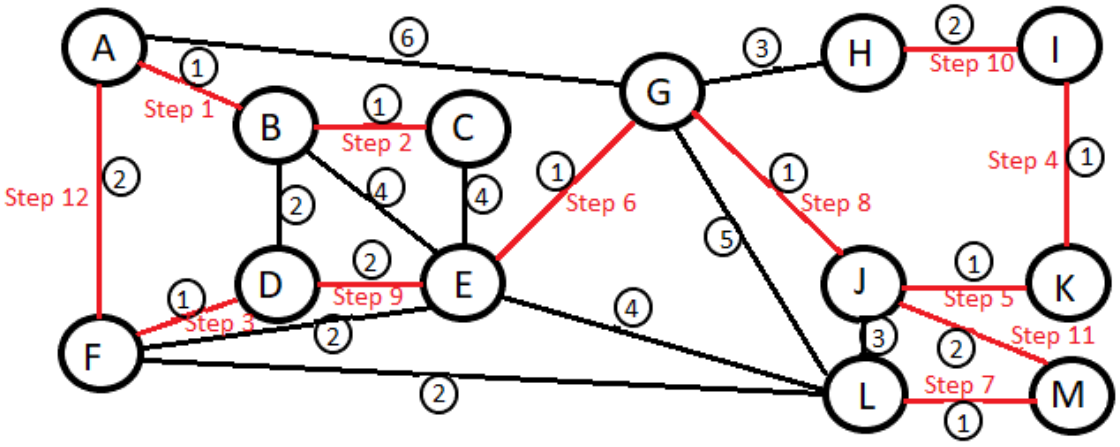
Sets –

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

Union-Find Partition –



Kruskal's MST graph diagram:



Output of Prim's algorithm:

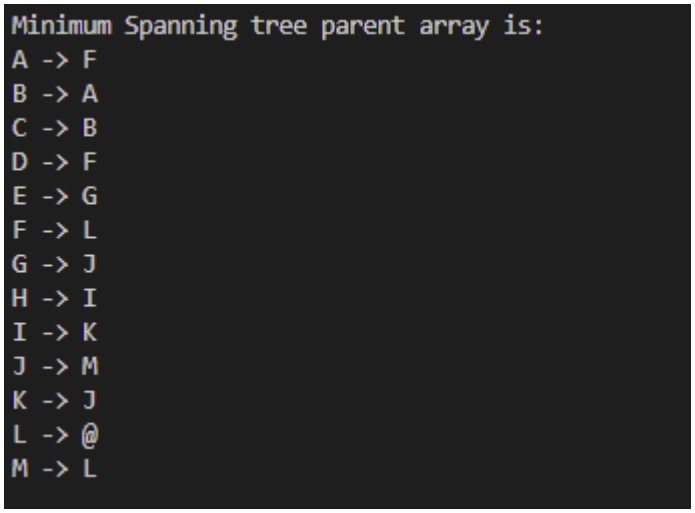
```
Enter the name of the file:
wGraph1.txt
Enter the starting vertex of the graph:
12
Parts[] = 13 22
Reading edges from text file
Edge A--(1)--B
Edge A--(2)--F
Edge A--(6)--G
Edge B--(1)--C
Edge B--(2)--D
Edge B--(4)--E
Edge C--(4)--E
Edge D--(2)--E
Edge D--(1)--F
Edge E--(2)--F
Edge E--(1)--G
Edge E--(4)--L
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

adj[A]:  -> [G(6)] -> [F(2)] -> [B(1)]
adj[B]:  -> [E(4)] -> [D(2)] -> [C(1)] -> [A(1)]
adj[C]:  -> [E(4)] -> [B(1)]
adj[D]:  -> [F(1)] -> [E(2)] -> [B(2)]
adj[E]:  -> [L(4)] -> [G(1)] -> [F(2)] -> [D(2)] -> [C(4)] -> [B(4)]
adj[F]:  -> [L(2)] -> [E(2)] -> [D(1)] -> [A(2)]
adj[G]:  -> [L(5)] -> [J(1)] -> [H(3)] -> [E(1)] -> [A(6)]
adj[H]:  -> [I(2)] -> [G(3)]
adj[I]:  -> [K(1)] -> [H(2)]
adj[J]:  -> [M(2)] -> [L(3)] -> [K(1)] -> [G(1)]
adj[K]:  -> [J(1)] -> [I(1)]
adj[L]:  -> [M(1)] -> [J(3)] -> [G(5)] -> [F(2)] -> [E(4)]
adj[M]:  -> [L(1)] -> [J(2)]
```

```
Depth First Search:
DFS visited vertex L along @--L
DFS visited vertex M along L--M
DFS visited vertex J along M--J
DFS visited vertex K along J--K
DFS visited vertex I along K--I
DFS visited vertex H along I--H
DFS visited vertex G along H--G
DFS visited vertex E along G--E
DFS visited vertex F along E--F
DFS visited vertex D along F--D
DFS visited vertex B along D--B
DFS visited vertex C along B--C
DFS visited vertex A along B--A

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

MST:
-------------------------
@       <-- (0) -->     L
L       <-- (1) -->     M
L       <-- (2) -->     F
F       <-- (1) -->     D
F       <-- (2) -->     A
A       <-- (1) -->     B
B       <-- (1) -->     C
M       <-- (2) -->     J
J       <-- (1) -->     K
J       <-- (1) -->     G
K       <-- (1) -->     I
G       <-- (1) -->     E
I       <-- (2) -->     H
-------------------------
MST Weight = 16
```



Output of Kruskal’s algorithm:

```
Input name of graph file: wGraph1.txt
Parts[] = 13 22
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 E--(4)--L
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

Kruskal's sets:
{ A } { B } { C } { D } { E } { F } { G } { H } { I } { J } { K } { L } { M }
Union: (A,B): { A B } { C } { D } { E } { F } { G } { H } { I } { J } { K } { L } { M }
Union: (B,C): { A B C } { D } { E } { F } { G } { H } { I } { J } { K } { L } { M }
Union: (D,F): { A B C } { D F } { E } { G } { H } { I } { J } { K } { L } { M }
Union: (I,K): { A B C } { D F } { E } { G } { H } { I K } { J } { L } { M }
Union: (J,K): { A B C } { D F } { E } { G } { H } { I J K } { L } { M }
Union: (E,G): { A B C } { D F } { E G } { H } { I J K } { L } { M }
Union: (L,M): { A B C } { D F } { E G } { H } { I J K } { L M }
Union: (G,J): { A B C } { D F } { E G I J K } { H } { L M }
Union: (D,E): { A B C } { D E F G I J K } { H } { L M }
Union: (H,I): { A B C } { D E F G H I J K } { L M }
Union: (J,M): { A B C } { D E F G H I J K L M }
Union: (A,F): { A B C D E F G H I J K L M }

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

Reflection:

- I have gained extensive knowledge to how Prim's MST algorithm is used with sparse graphs and adjacency lists and how Kruskal's MST algorithm is used with the union-find implementation using disjoint set trees.

- I have also gained knowledge of how the heaps and graphs work Prim's and Kruskal's algorithms.
- I have learnt how to construct a graph and perform algorithms from a text file that was inputted from the user.
- Have also learnt how to balance my time for multiple ongoing assignments while tackling a timely project like this project.
- This project has also allowed me to gain more knowledge at how to use java and an object oriented language while using different data structures and algorithms.
- This project will provide to be useful whenever I need to visualize how Prim's and Kruskal's work on different graphs by simply creating a new text file with the contents of the graph.