Algorithms and Data Structures

Graph Traversal, MST & SPT Algorithm Assignment

Name: Hemant Sundarrajan

Student number: C22440886

For various parts of the code and debugging, I worked in a small group

1. Introduction

For my Algorithms and Data structures assignment I was required to implement and show my knowledge on a range of algorithms and data structures. I was required to implement Depth First Traversal, Breadth first traversal, Prims Algorithm, Dijkstra's Algorithm and Kruskal's Algorithm. I had 1 file for Prims, Dijkstra's, BFS and DFS, and another file for Kruskal's. I had to implement these algorithms on a provided, weighted, and undirected graph. I am to start from node L in the execution of all my code, and display of how the algorithms work. This meant I had to create a file which displayed how many edges and vertices were in the graph, and their relationships between each other. For example, the line "1 2 1" essentially says "A connects to B with a weight of 1", with A being 1, B being 2, and 1 being the weight of the distance between them. Then using the provided code for the assignment, I read the text file in, and was able to use the graphs' edges and vertices in my code.

2. Adjacency List Diagram

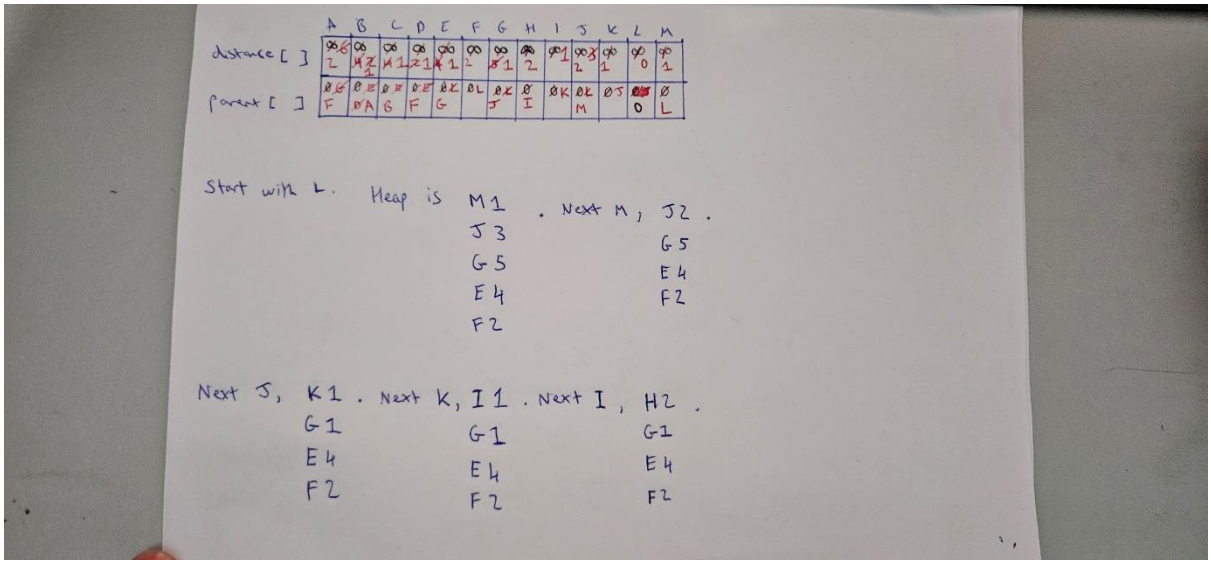
Graph Representation of sample graph:

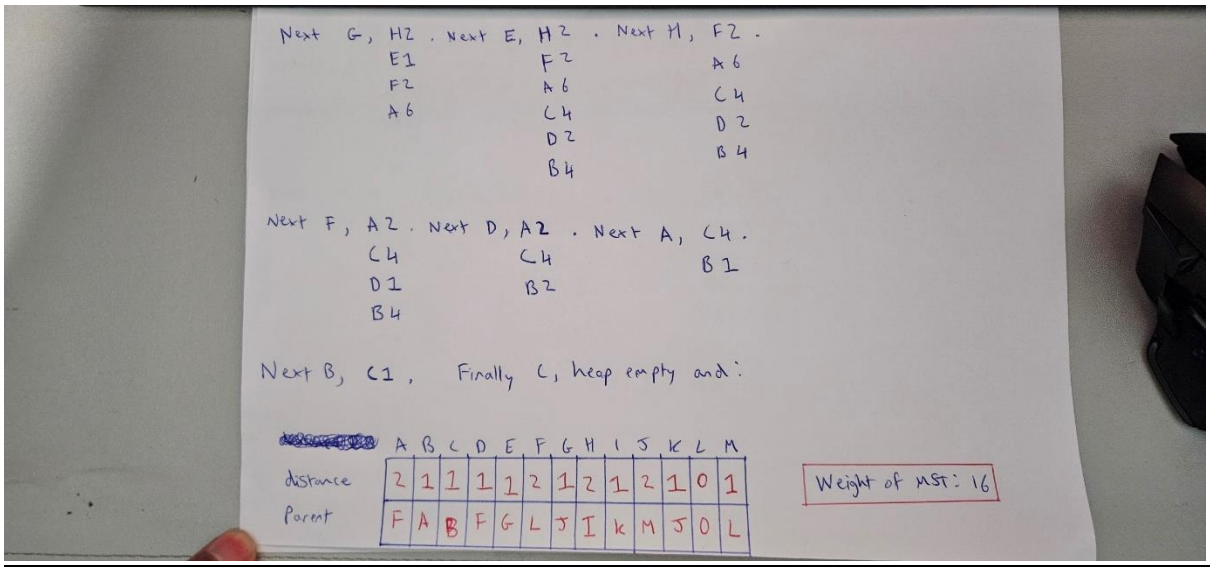
```
adj[A] -> |G | 6| -> |F | 2| -> |B | 1| ->
adj[B] -> |E | 4| -> |D | 2| -> |C | 1| -> |A | 1| ->
adj[C] -> |E | 4| -> |B | 1| ->
adj[D] -> |F | 1| -> |E | 2| -> |B | 2| ->
adj[E] -> |L | 4| -> |G | 1| -> |F | 2| -> |D | 2| -> |C | 4| -> |B | 4| ->
adj[F] -> |L | 2| -> |E | 2| -> |D | 1| -> |A | 2| ->
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| ->
```

To explain how the diagram works an example is adj[A]. adj[A] has 3 connections, it connects to G with a weight of 6, F with a weight of 2, and B with a weight of 1. This diagram shows how the graph is represented.

3. Step by Step Construction of MST using Prim's Algorithm from vertex L

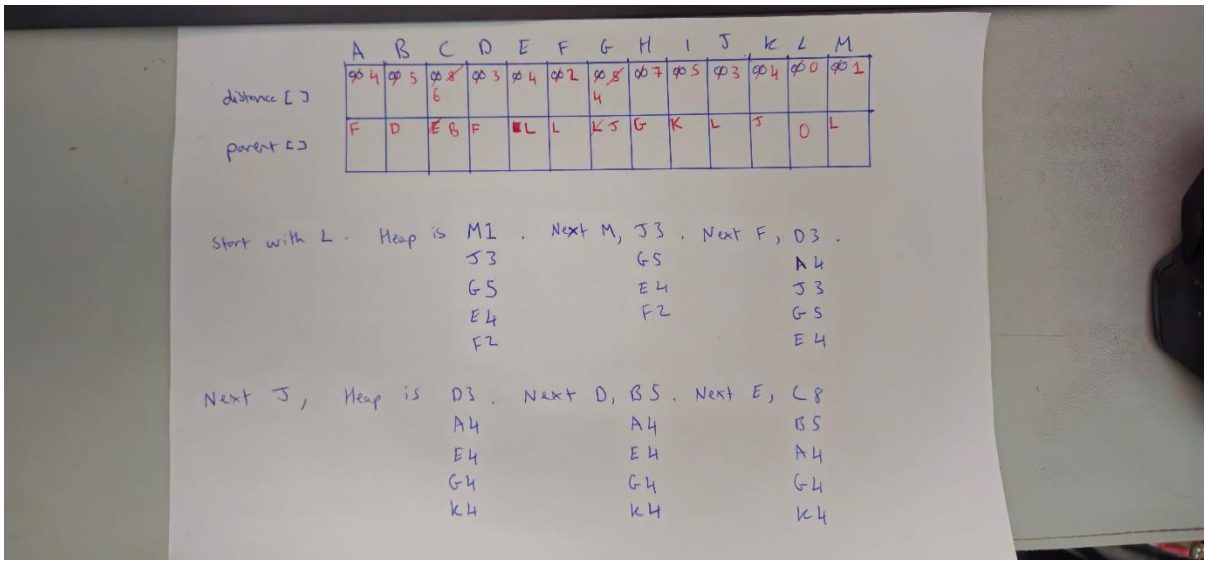
This construction of the MST clearly shows how Prims Algorithm is utilized on the graph provided. It shows the heap, parent[] and distance[] at exactly each step, starting from Vertex L, till the completion of the algorithm.

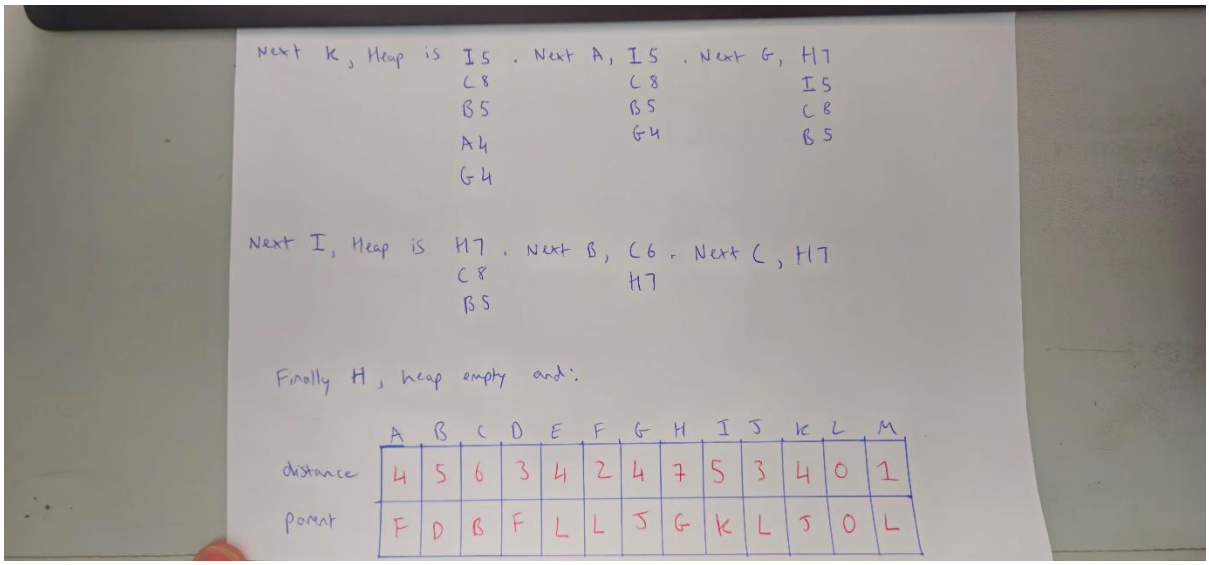




4. A step-by-step construction of the SPT using Dijkstra's Algorithm

This construction shows the SPT of the graph provided utilizing Dijkstra's Algorithm. It begins at Vertex L and iterates through the graph until completion. It clearly shows the state of the heap, distance[] and parent[] from start to finish. When a distance is updated for example, I strike it out and write the new distance. Of course, this also means that the parent gets updated in accordance with the distance. If you follow the heap you can see how certain distances change.





5. a step-by-step construction of the MST for Kruskal

Also shows step-by-step the union-find partition and set representation for Kruskal at each step.

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

Clearly shows the sets, and how they are all interlinked. Shows step-by-step how the vertexes all connect to create the MST.

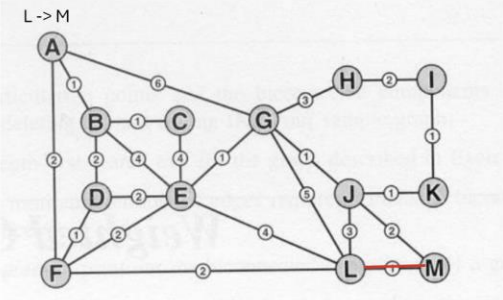
The MST:

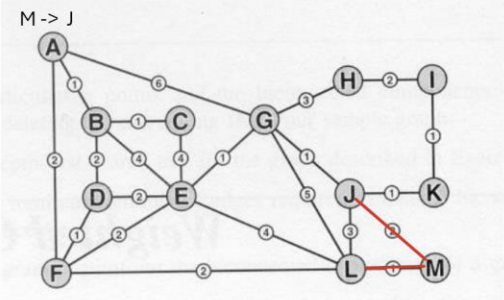
```
Minimum spanning tree build from following edges:
Edge A--1--B
Edge B--1--C
Edge D--1--F
Edge I--1--K
Edge K--1--J
Edge E--1--G
Edge L--1--M
Edge G--1--J
Edge D--2--E
Edge H--2--I
Edge J--2--M
Edge A--2--F
```

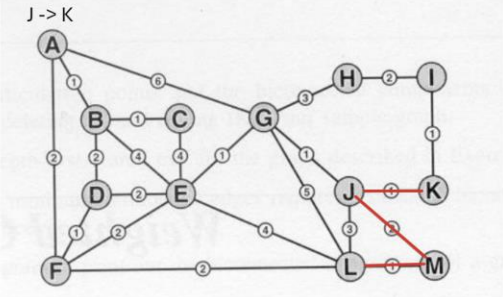
6. A diagram showing the MST superimposed on the graph

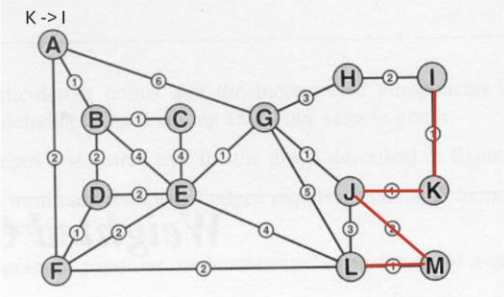
All Diagrams move from left to right. Prims Algorithm Starts at Vertex L.

Prims Algorithm superimposed on the graph

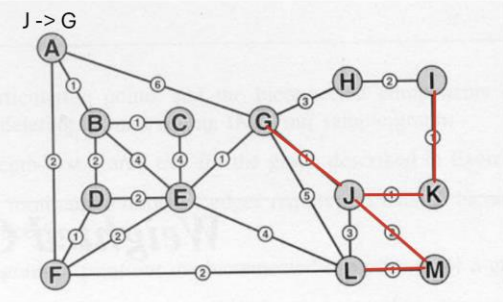




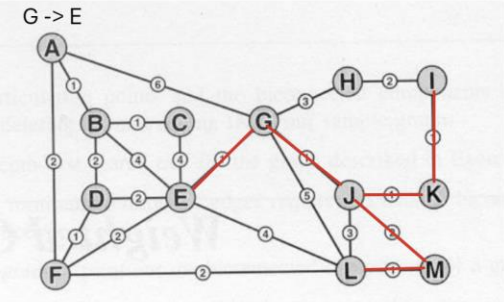




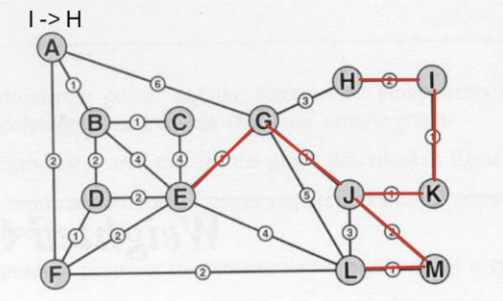
J -> G



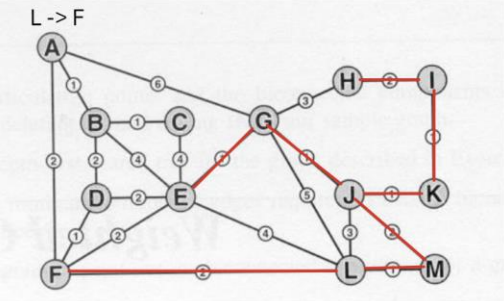
G -> E



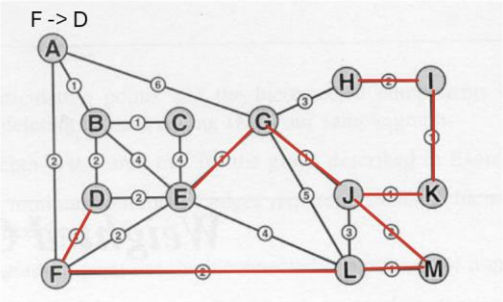
I -> H



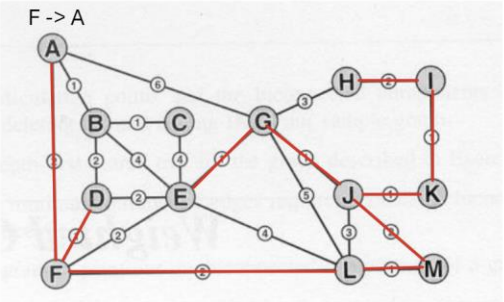
L -> F



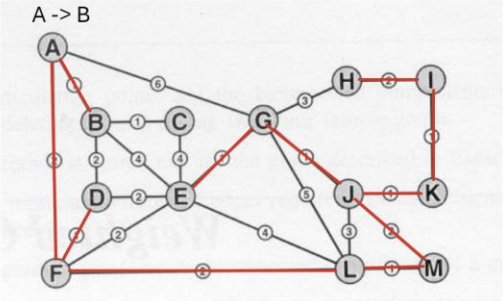
F -> D



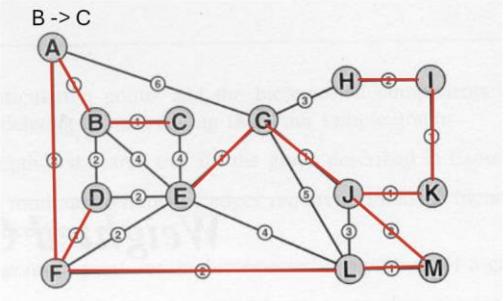
F -> A



A -> B

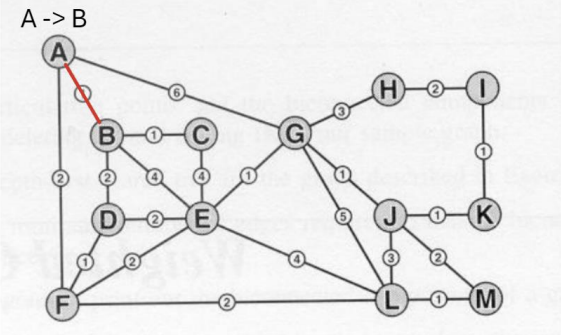


B -> C

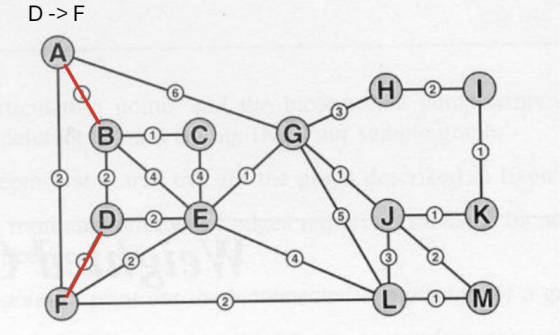


Kruskal's Algorithm superimposed on the graph

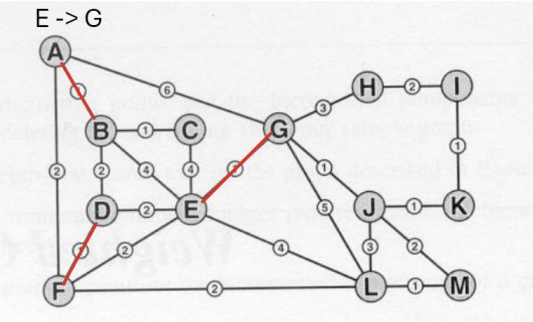
A -> B



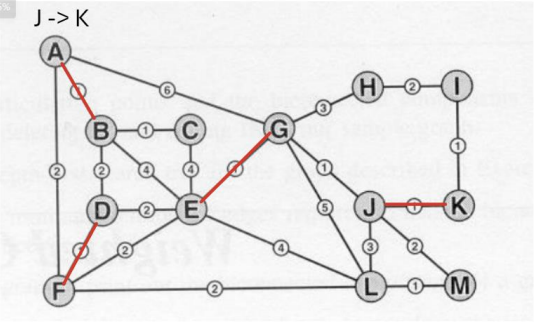
D -> F



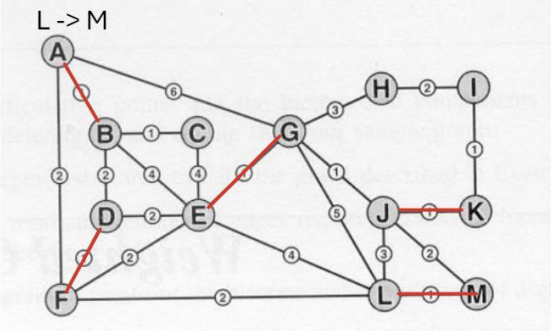
E -> G



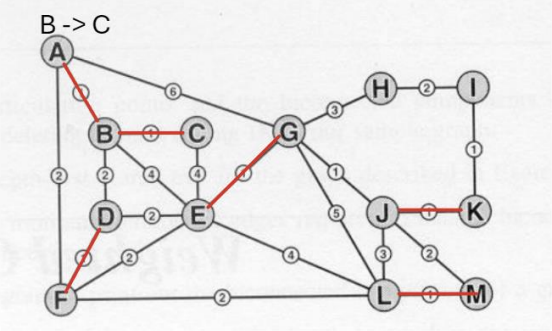
J -> K



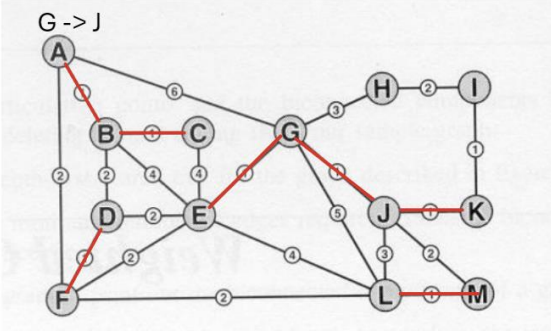
L -> M



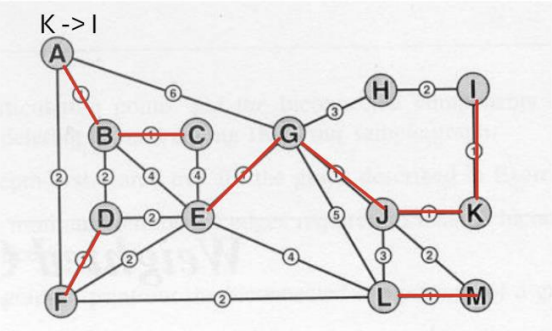
B -> C

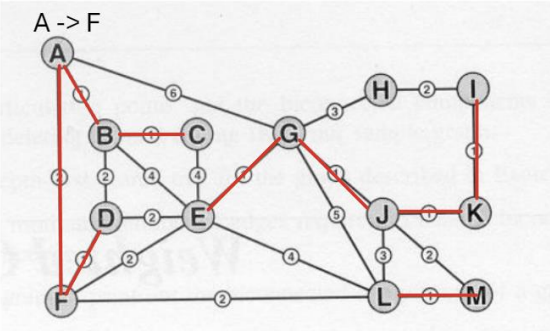


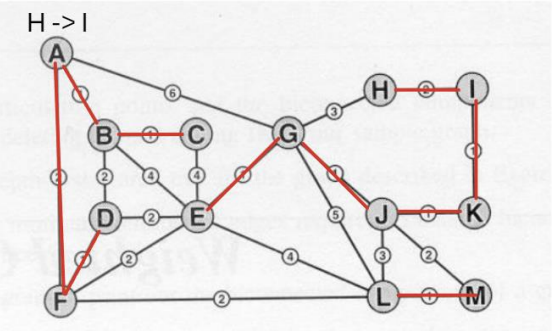
G -> J

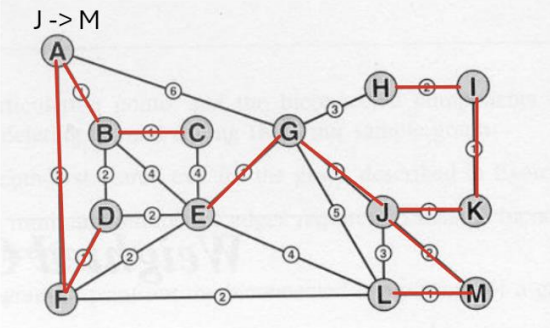


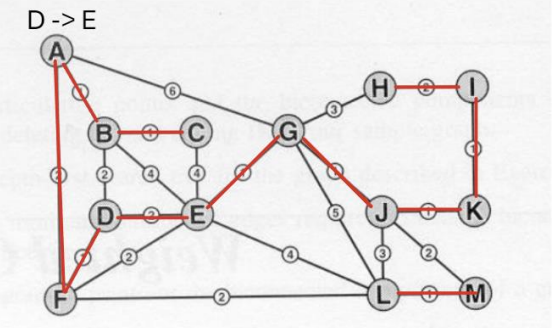
K -> I









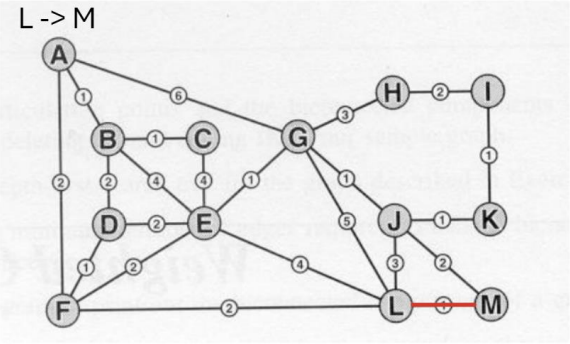


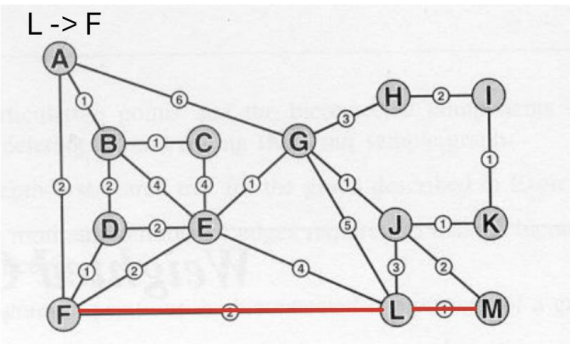
---

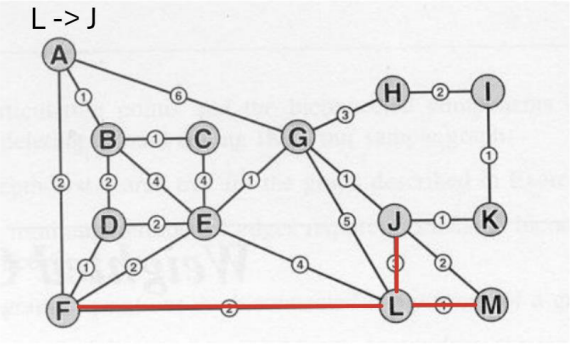
7. A diagram showing the SPT superimposed on the graph

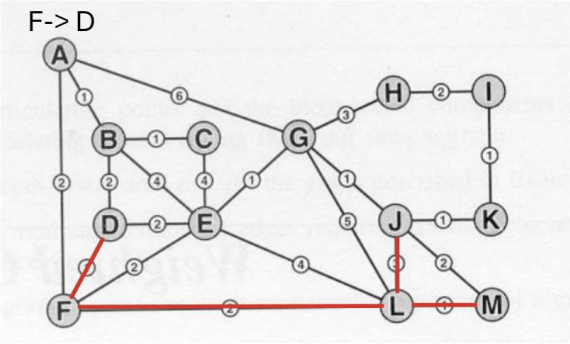
The Algorithm begins at Vertex L. Pictures move left to right.

Dijkstra's Algorithm superimposed on a graph:

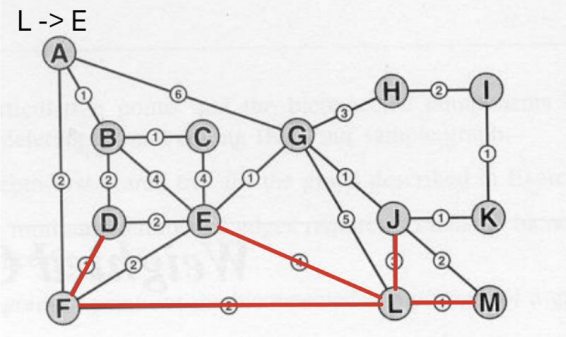




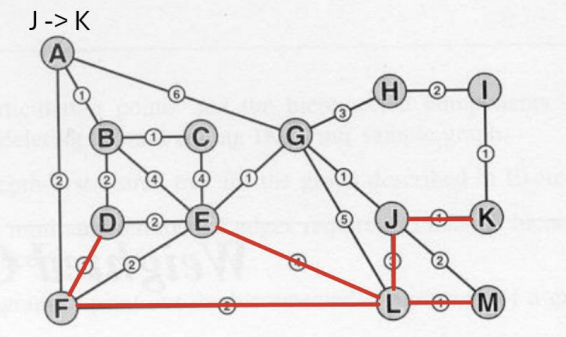




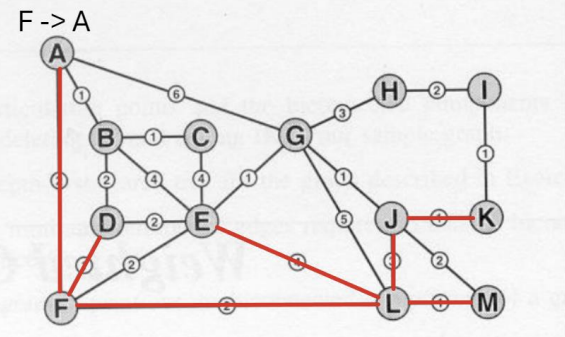
L -> E



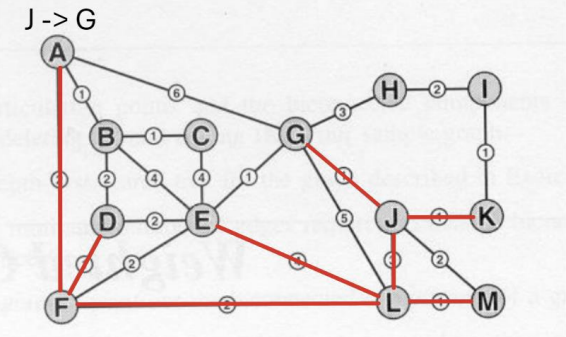
J -> K



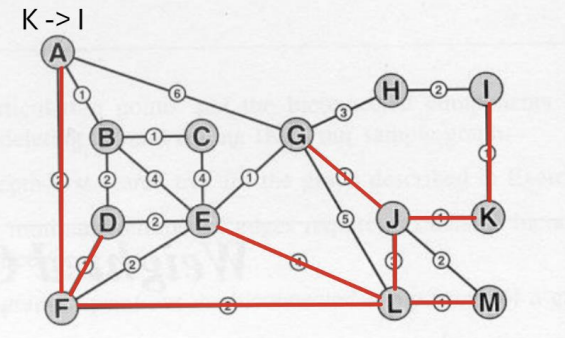
F -> A



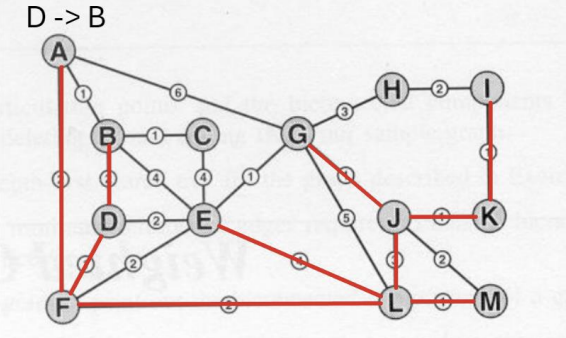
J -> G



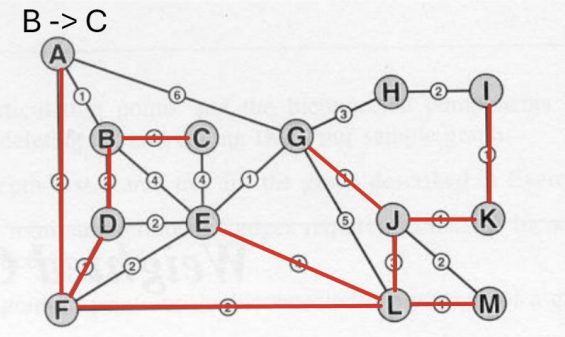
K -> I



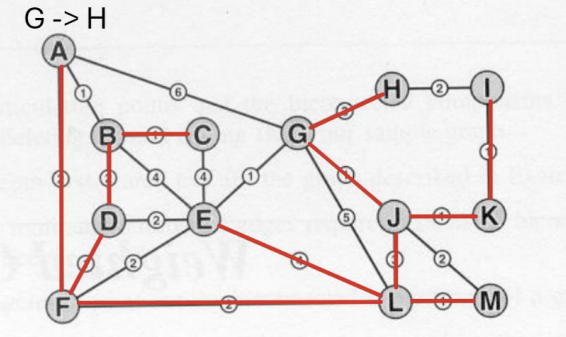
D -> B



B -> C



G -> H



---

8. screen captures showing all the programs executing

Depth first traversal:

```
Input name of file with graph definition: wGraph1.txt

Enter the vertex you want to start at (I.E A = 1, B = 2 etc): 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 K--(1)--J
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| ->

DFS using Recursion:
Visiting Vertex [L] from Vertex [@]
Visiting Vertex [M] from Vertex [L]
Visiting Vertex [L] from Vertex [M]
Visiting Vertex [J] from Vertex [L]
Visiting Vertex [K] from Vertex [J]
Visiting Vertex [I] from Vertex [K]
Visiting Vertex [H] from Vertex [I]
Visiting Vertex [G] from Vertex [H]
Visiting Vertex [E] from Vertex [G]
Visiting Vertex [F] from Vertex [E]
Visiting Vertex [D] from Vertex [F]
Visiting Vertex [B] from Vertex [D]
Visiting Vertex [C] from Vertex [B]
Visiting Vertex [A] from Vertex [B]
PS C:\Users\heman\OneDrive\Documents\Sem2SecYear\Alogrithms\Assignment\Prims>
```

Breadth First Traversal:

```
Input name of file with graph definition: wGraph1.txt

Enter the vertex you want to start at (I.E A = 1, B = 2 etc): 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 K--(1)--J
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| ->

BFS using a queue:
Visiting Vertex [L] (Starting vertex)
Visiting Vertex [M] from Vertex [L]
Visiting Vertex [J] from Vertex [L]
Visiting Vertex [G] from Vertex [L]
Visiting Vertex [F] from Vertex [L]
Visiting Vertex [E] from Vertex [L]
Visiting Vertex [K] from Vertex [J]
Visiting Vertex [H] from Vertex [G]
Visiting Vertex [A] from Vertex [G]
Visiting Vertex [D] from Vertex [F]
Visiting Vertex [C] from Vertex [E]
Visiting Vertex [B] from Vertex [E]
Visiting Vertex [I] from Vertex [K]
PS C:\Users\heman\OneDrive\Documents\Sem2SecYear\Alogrithms\Assignment\Prims>
```

Prim’s Minimum Spanning Tree:

```
Input name of file with graph definition: wGraph1.txt

Enter the vertex you want to start at (I.E A = 1, B = 2 etc): 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 K--(1)--J
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| ->

Prims Algorithm:
Starting vertex: L
Vertex M is connected to Vertex L with edge weight = 1
Vertex F is connected to Vertex L with edge weight = 2
Vertex D is connected to Vertex F with edge weight = 1
Vertex A is connected to Vertex F with edge weight = 2
Vertex B is connected to Vertex A with edge weight = 1
Vertex C is connected to Vertex B with edge weight = 1
Vertex J is connected to Vertex M with edge weight = 2
Vertex K is connected to Vertex J with edge weight = 1
Vertex G is connected to Vertex J with edge weight = 1
Vertex I is connected to Vertex K with edge weight = 1
Vertex H is connected to Vertex I with edge weight = 2
Vertex E is connected to Vertex G with edge weight = 1


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

PS C:\Users\heman\OneDrive\Documents\Sem2SecYear\Alogrithms\Assignment\Prims>
```

Dijkstra’s SPT:

```
Input name of file with graph definition: wGraph1.txt

Enter the vertex you want to start at (I.E A = 1, B = 2 etc): 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 K--(1)--J
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| ->

Dijkstras Algorithm:

Shortest Path Tree:
Vertex A is connected to Vertex F with edge weight = 4
Vertex B is connected to Vertex D with edge weight = 5
Vertex C is connected to Vertex B with edge weight = 6
Vertex D is connected to Vertex F with edge weight = 3
Vertex E is connected to Vertex L with edge weight = 4
Vertex F is connected to Vertex L with edge weight = 2
Vertex G is connected to Vertex J with edge weight = 4
Vertex H is connected to Vertex G with edge weight = 7
Vertex I is connected to Vertex K with edge weight = 5
Vertex J is connected to Vertex L with edge weight = 3
Vertex K is connected to Vertex J with edge weight = 4
Vertex L is connected to Vertex @ with edge weight = 0
Vertex M is connected to Vertex L with edge weight = 1
PS C:\Users\heman\OneDrive\Documents\Sem2SecYear\Alogrithms\Assignment\Prims>
```

Kruskal Minimum Spanning Tree:

```
Input name of file with graph definition: 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 K--(1)--J
Edge J--(3)--L
Edge J--(2)--M
Edge L--(1)--M

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

Minimum spanning tree build from following edges:
Edge A--1--B
Edge B--1--C
Edge D--1--F
Edge I--1--K
Edge K--1--J
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

PS C:\Users\heman\OneDrive\Documents\Sem2SecYear\Alogrithms\Assignment\Kruskals>
```

9. Discussion on what I learned or found useful in the assignment

Upon completion of the assignment, I found that I learnt a lot about the intricacies of the algorithms used. By taking an in depth look at how they work iteration on iteration, you can get a much deeper understanding of how the algorithms might work in real life. That's something I find quite interesting considering they seem simple in theory but have a wide range of utility.

By utilising algorithms and data structures, I was able to get a great understanding in this assignment on exactly how these different algorithms work, and their uniqueness in solving specific problems. Be it using union by rank in Kruskal's algorithm, or utilizing depth first search with recursion, there is so much to learn and absorb from these algorithms. I personally found that completing Dijkstra's algorithm, and Prims algorithm by hand to be very useful for me personally. Doing every iteration of an algorithm yourself is extremely helpful in understanding the basics of how they work. Had I not done this assignment, I would not have understood these algorithms to such an extent that. After analysing these algorithms in this assignment, I definitely learnt core skills in understanding the mechanisms behind algorithms in general.