Algorithms Assignment

Name: Neil Gibson

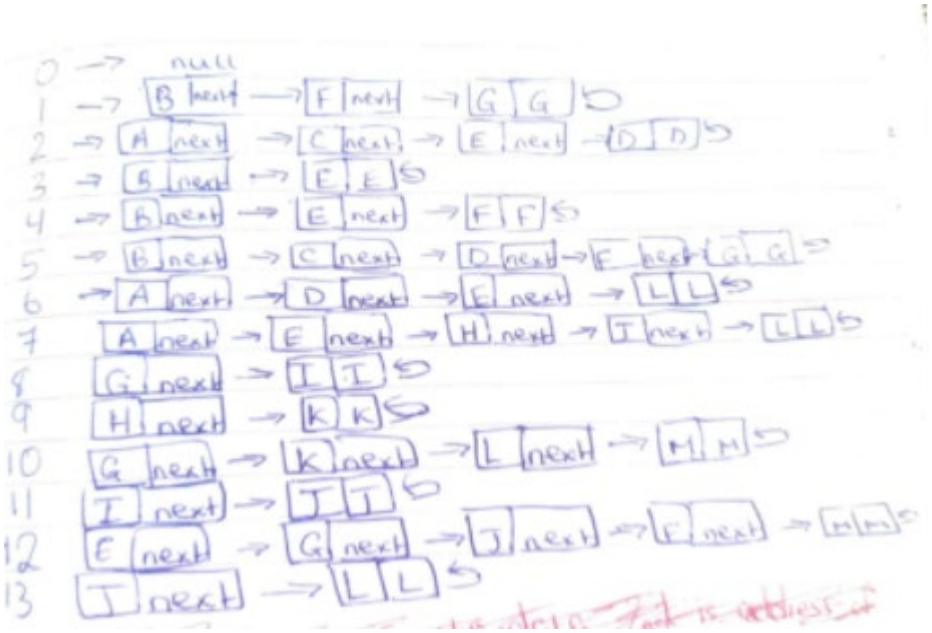
Student number: C17438292

Introduction

In this report I will be going through how I implemented Kruskal's and Prims's algorithm on a given graph. Each program prompts the for the name of a text file, and then for prims it also asks for a vertex to start at on the graph. Kruskal and Prim are both algorithms to find the minimum spanning tree of a graph. A minimum spanning tree (MST) is a subset of edges that connects all vertices on a weighted undirected graph without cycles and in the minimum amount of weight/distance. Prim starts from a single point and picks the next shortest edge that can be connected to the current tree of vertices, whereas Kruskal just picks the next shortest edge every time from anywhere on the graph (Does not need to be connected to any vertex to be chosen next).

Adjacency List graph representation

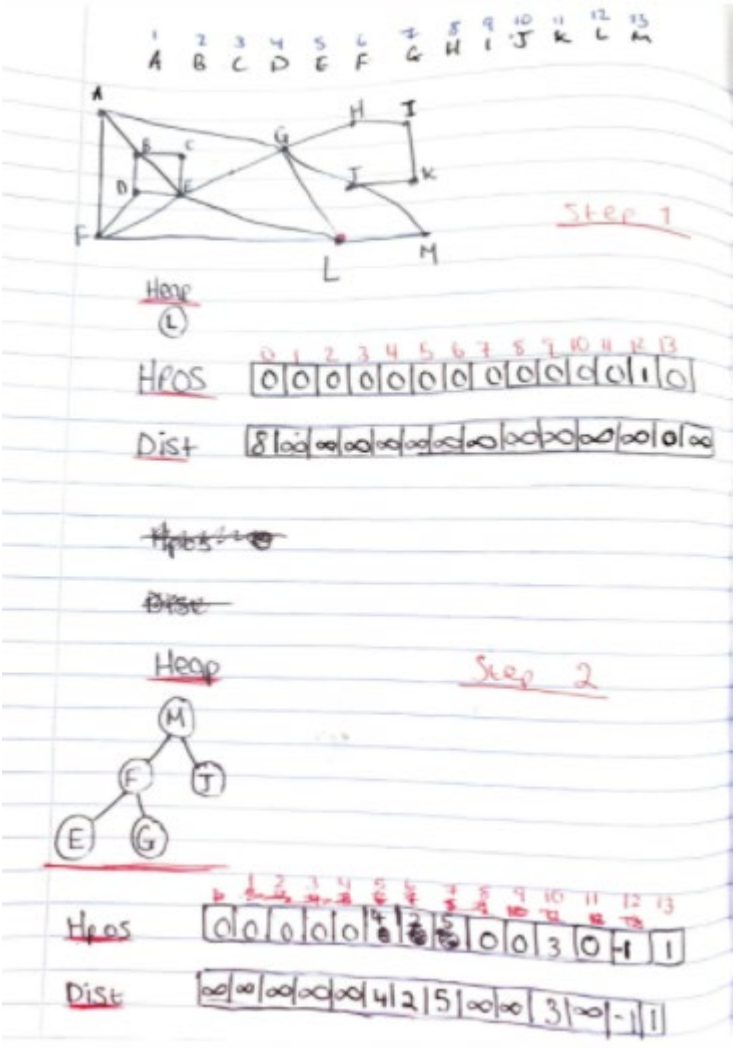
Below is a picture of how the adjacency list will look for the graph. This array goes from 0 to 13 using spaces 1 to 13 (A to M). Each element of the array contains its own linked list of all vertices it connects to in the graph. Each node of the linked list contains a data field (the letter of the node) and a pointer field which points to the next node. The last node on each linked list points back to itself (sentinel node) which you can see in the picture below.

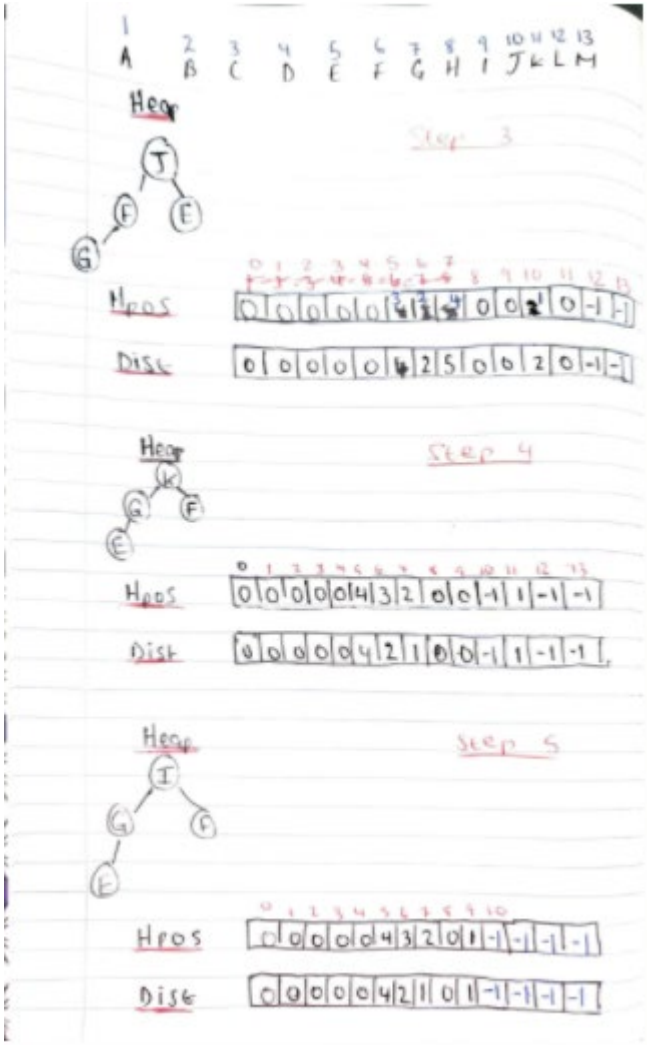


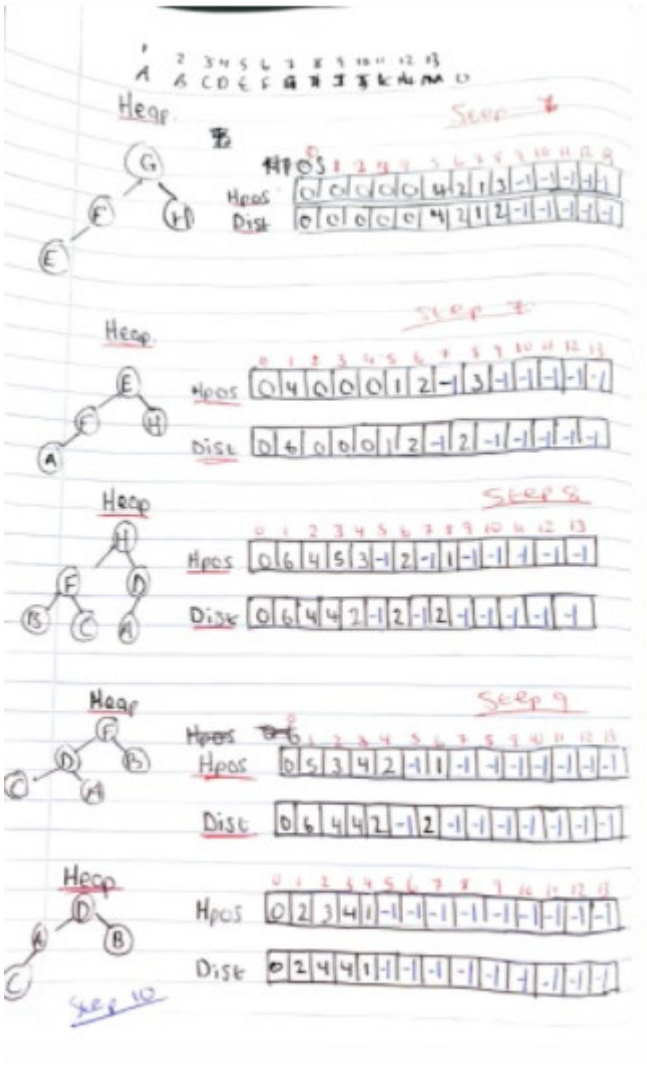
```
adj[A] -> |G | 6| -> |B | 1| -> |F | 2| ->
adj[B] -> |D | 2| -> |E | 4| -> |C | 1| -> |A | 1| ->
adj[C] -> |E | 4| -> |B | 1| ->
adj[D] -> |F | 1| -> |E | 2| -> |B | 2| ->
adj[E] -> |L | 4| -> |G | 1| -> |F | 2| -> |D | 2| -> |C | 4| -> |B | 4| ->
adj[F] -> |L | 2| -> |E | 2| -> |D | 1| -> |A | 2| ->
adj[G] -> |H | 3| -> |J | 1| -> |L | 5| -> |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| ->
```

Prim step by step representation

For reference the symbol in distance is infinity. From step 3 onwards infinity is just represented as 0 in distance array. Whenever a vertex has been visited its data in HPOS and DIST array are changed to '-1' to show that the vertex has been visited to avoid revisiting a vertex. Prim works by updating the distance of a vertex in DIST whenever the vertex is available to travel to, or the new way of getting there is smaller than the current distance. HPOS represents the position a vertex is on the heap, if it is 0 it means it is not on the heap yet. The heaps priority is based on a vertex's current smallest distance and is updated every time a new vertex is added onto the heap to reflect this. Here are how the heap, HPOS and DIST array look after every step. (Starting from vertex 'L').



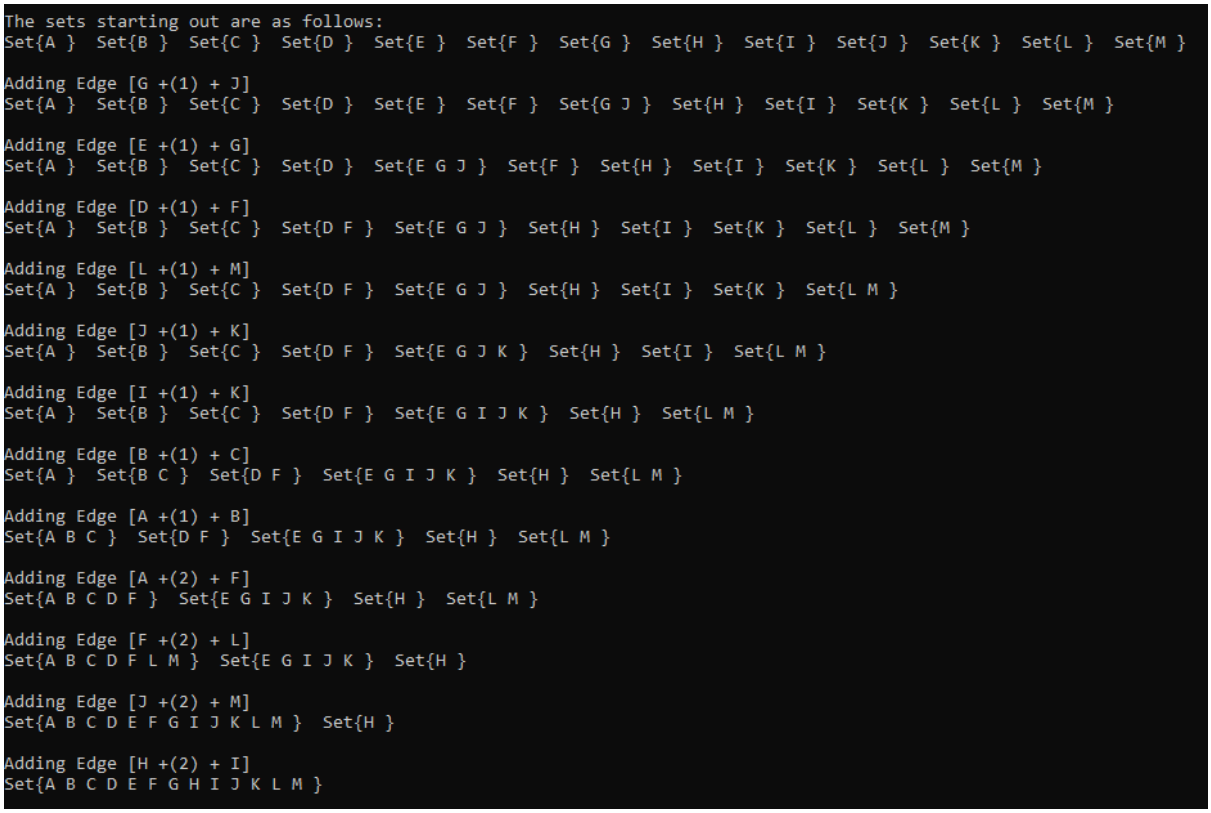

1 2 3 4 5 6 7 8 9 10 11 12 13
A B C D E F G H I J K L M
Heap
Step 3
Heos
Dist
Heap
Step 4
Heos
Dist
Heap
Step 5
Heos
Dist


Heap
Step 6
Hpos
Dist
Heap
Step 7
Hpos
Dist
Heap
Step 8
Hpos
Dist
Heap
Step 9
Hpos
Dist
Heap
Hpos
Dist
Step 10

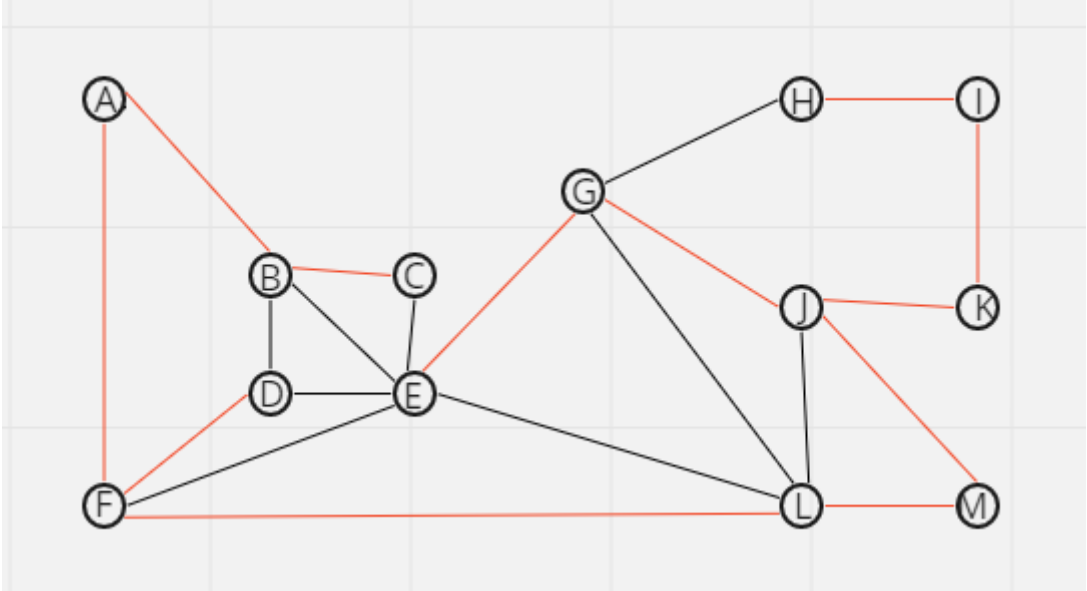

Step 11
Heap
A
B
C
Hpos
Dist
Heap
B
C
Hpos
Dist
Step 12
Heap
C
Step 13
Hpos
Dist
Heap
Hpos
Dist
Step 14

Kruskal step by step representation

This is step by step how the sets look after every edge is added to the minimum spanning tree and the order in which the edges are added to the MST each time. Kruskal works by picking the edge with the smallest distance every time, checking if the 2 vertices are already in the same set. If they are this means joining them will end up in a cycle, if not we can add the edge to the MST and join/unionize the 2 sets. You will see the sets gradually getting bigger after each step in the picture below due to this.



MST

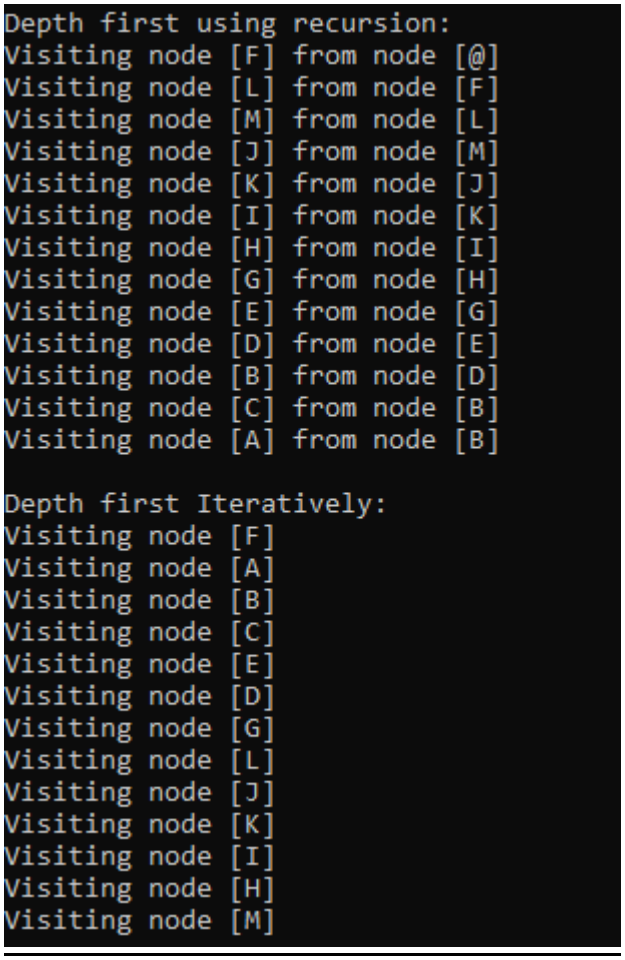


Screen capture of programmes running

Reading the edges in for Prim's and displaying the adjacency list of all vertices and their edges:

```
D:\Algo-Ass>java PrimLists

Input name of file with graph definition: wgraph1.txt

Input the number of the vertex you want to start at: 12

Parts[] = 13 22
Reading edges from text file
Edge A--(2)--F
Edge A--(1)--B
Edge A--(6)--G
Edge B--(1)--C
Edge B--(4)--E
Edge B--(2)--D
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--(5)--L
Edge G--(1)--J
Edge G--(3)--H
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| -> |B | 1| -> |F | 2| ->
adj[B] -> |D | 2| -> |E | 4| -> |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] -> |H | 3| -> |J | 1| -> |L | 5| -> |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| ->
```

This is the output when doing depth first traversal from a specified point using a recursive approach and then an iterative approach. Recursive approach obviously uses recursion and the iterative approach uses a stack to decide which node it will visit next:



This is the breadth first traversal of the graph which uses a queue to decide which node it will visit next:

```
Breadth First Traversal:
Currently visiting [L]
Currently visiting [M]
Currently visiting [J]
Currently visiting [G]
Currently visiting [F]
Currently visiting [E]
Currently visiting [K]
Currently visiting [H]
Currently visiting [A]
Currently visiting [D]
Currently visiting [C]
Currently visiting [B]
Currently visiting [I]
```

This is just some selective feedback (NOT ALL!!) of what is going on with the heap in terms of numbers being inserted and removed onto it while making the minimum spanning tree. These numbers then get sifted up and sifted down according to the edge with the lowest weights:

```
Inserting: 12
Removing: 12
Inserting: 13
Inserting: 10
Inserting: 7
Inserting: 6
Inserting: 5
Removing: 13
Removing: 6
Inserting: 4
Inserting: 1
Removing: 4
Inserting: 2
Removing: 1
Removing: 2
Inserting: 3
Removing: 3
Removing: 10
Inserting: 11
Removing: 11
Inserting: 9
Removing: 7
Inserting: 8
Removing: 9
Removing: 8
Removing: 5
```

This is the result of the minimum spanning tree for prim's, displaying the weight and the minimum spanning tree itself. L is visited from @ as it is the starting node that the user has picked in this case:

```
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
```

This is reading in the edges for kruskal's algorithm:

```
Input name of file with graph definition: wgraph1.txt
Parts[] = 13 22
Reading edges from text file
Edge A--(2)--F
Edge A--(1)--B
Edge A--(6)--G
Edge B--(1)--C
Edge B--(4)--E
Edge B--(2)--D
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--(5)--L
Edge G--(1)--J
Edge G--(3)--H
Edge H--(2)--I
Edge I--(1)--K
Edge J--(1)--K
Edge J--(3)--L
Edge J--(2)--M
Edge L--(1)--M
```

This is choosing the next edge each time based on their weights, and displaying the state of the sets/trees after every addition to the tree:

```
The sets starting out are as follows:
Set{A }  Set{B }  Set{C }  Set{D }  Set{E }  Set{F }  Set{G }  Set{H }  Set{I }  Set{J }  Set{K }  Set{L }  Set{M }

Adding Edge [G +(1) + J]
Set{A }  Set{B }  Set{C }  Set{D }  Set{E }  Set{F }  Set{G J }  Set{H }  Set{I }  Set{K }  Set{L }  Set{M }

Adding Edge [E +(1) + G]
Set{A }  Set{B }  Set{C }  Set{D }  Set{E G J }  Set{F }  Set{H }  Set{I }  Set{K }  Set{L }  Set{M }

Adding Edge [D +(1) + F]
Set{A }  Set{B }  Set{C }  Set{D F }  Set{E G J }  Set{H }  Set{I }  Set{K }  Set{L }  Set{M }

Adding Edge [L +(1) + M]
Set{A }  Set{B }  Set{C }  Set{D F }  Set{E G J }  Set{H }  Set{I }  Set{K }  Set{L M }

Adding Edge [J +(1) + K]
Set{A }  Set{B }  Set{C }  Set{D F }  Set{E G J K }  Set{H }  Set{I }  Set{L M }

Adding Edge [I +(1) + K]
Set{A }  Set{B }  Set{C }  Set{D F }  Set{E G I J K }  Set{H }  Set{L M }

Adding Edge [B +(1) + C]
Set{A }  Set{B C }  Set{D F }  Set{E G I J K }  Set{H }  Set{L M }

Adding Edge [A +(1) + B]
Set{A B C }  Set{D F }  Set{E G I J K }  Set{H }  Set{L M }

Adding Edge [A +(2) + F]
Set{A B C D F }  Set{E G I J K }  Set{H }  Set{L M }

Adding Edge [F +(2) + L]
Set{A B C D F L M }  Set{E G I J K }  Set{H }

Adding Edge [J +(2) + M]
Set{A B C D E F G I J K L M }  Set{H }

Adding Edge [H +(2) + I]
Set{A B C D E F G H I J K L M }
```

This is the minimum spanning tree displayed to the user:

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

Conclusion

I have learned a lot on this project, particularly about data structures and how they can be applied to real problems. I now feel a lot more comfortable in explaining and using queues, stacks, heaps and linked lists. I also realise that these are all important data structures to get to know inside and out. When doing this project I really found it useful to map out each individual step visually. This helped me better understand what was happening at each step and it was also a really useful tool when error checking as I would print results to the screen and compare with my drawn out work. I will now use this approach a lot more often. I found this project extremely helpful toward my programming skills due to all of these factors. I liked Kruskal's algorithm better as it was a lot easier to understand as it simply picks the next shortest edge on the graph and there was a lot less code. No need to keep track of or update distances as they should be in sorted order and the union find data structure was easy to implement. You are also not confined to starting at a certain point in a kind of depth first/breadth first manner. Prims was fine too but it felt like there was a lot more going on when accessing the DIST array through the HPOS array which referenced to the heap etc. It was like inception, if it was based on arrays.