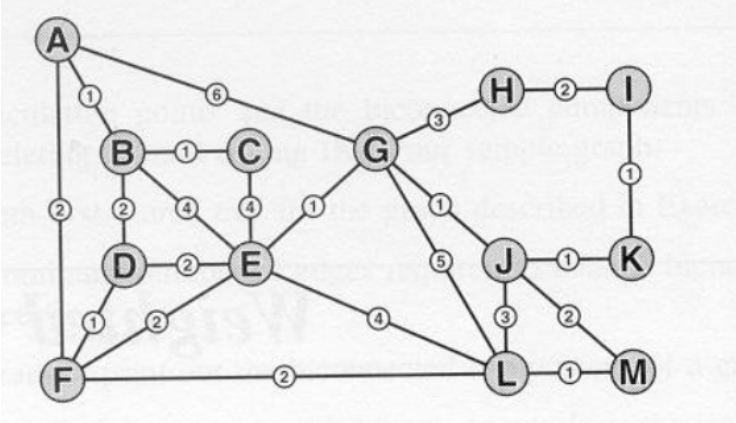
Algorithm Assignment

Aaron Connolly - C22410766

Introduction/Explanation

The program will ask the user for the name of the the text file containing a sample graph which should be “wGraph1.txt”. Then the user will be asked for the starting vertex that each algorithm will start from. A menu will then be printed asking the user which algorithm they wish to be displayed. The options are 1. Prim’s MST, 2. Dijkstra SPT, 3. Depth First Search, 4. Breadth First Search.

A separate program Krystal.java will construct an MST using Kruskal’s algorithm. The reason this is in a separate program is because both Kruskal and Prim use heaps, but Kruskal’s algorithm uses an array of edges.

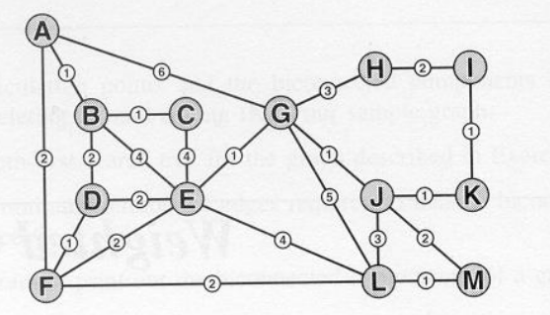


Adjacency List

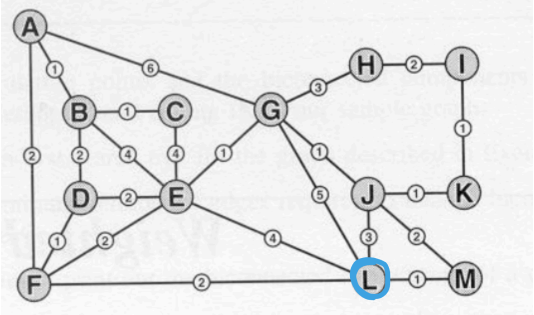
```
adj[A] -> |G | 6| -> |F | 2| -> |B | 1| ->
adj[B] -> |E | 4| -> |D | 2| -> |C | 1| -> |A | 1| ->
adj[C] -> |E | 4| -> |B | 1| ->
adj[D] -> |F | 1| -> |E | 2| -> |B | 2| ->
adj[E] -> |L | 4| -> |G | 1| -> |F | 2| -> |D | 2| -> |C | 4| -> |B | 4| ->
adj[F] -> |L | 2| -> |E | 2| -> |D | 1| -> |A | 2|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

Construction of MST using Prim's algorithm

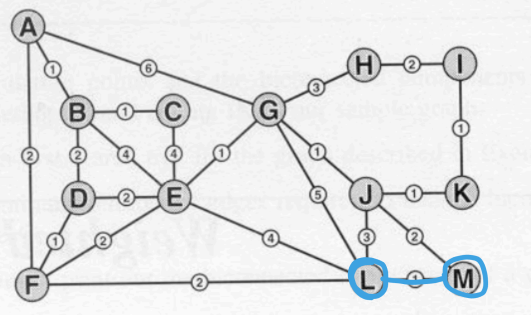
```
Starting vertex: L
Vertex M is connected to Vertex L with edge weight = 1
Vertex F is connected to Vertex L with edge weight = 2
Vertex D is connected to Vertex F with edge weight = 1
Vertex A is connected to Vertex F with edge weight = 2
Vertex B is connected to Vertex A with edge weight = 1
Vertex 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 E is connected to Vertex G with edge weight = 1
Vertex H is connected to Vertex I with edge weight = 2

Weight of MST = 16
```



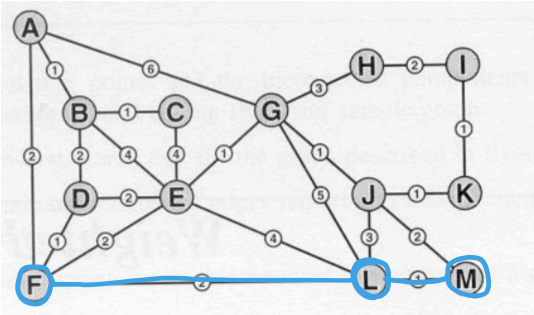
Heap[L]
Parent[0,0,0,0,0,0,0,0,0,0,0,0,0]
Dist[∞,∞,∞,∞,∞,∞,∞,∞,∞,∞,∞,∞,∞]



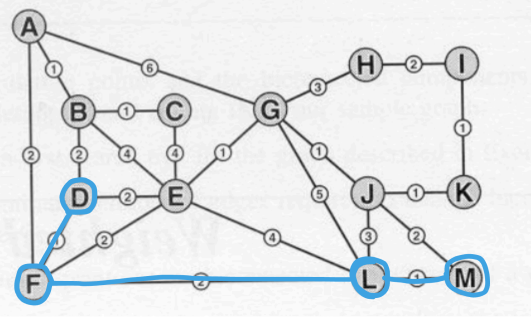
Heap[E,F, G, J, M]
Parent[0,0,0,0,0,0,0,0,0,0,0,0,0]
Dist[∞,∞,∞,∞,∞,∞,∞,∞,∞,∞,∞,0,∞]



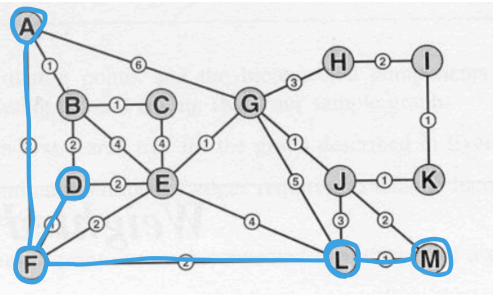
Heap[E, J, G, F]
Parent[0,0,0,0,0,0,0,0,0,0,0,0,12]
Dist[∞,∞,∞,∞,∞,∞,∞,∞,∞,∞,∞,0,1]



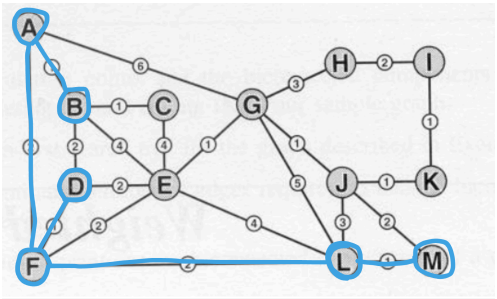
Heap[A, D, E, G, J]
Parent[0,0,0,0,0,12,0,0,0,0,0,0,12]
Dist[∞,∞,∞,∞,∞,2,∞,∞,∞,∞,∞,0,1]



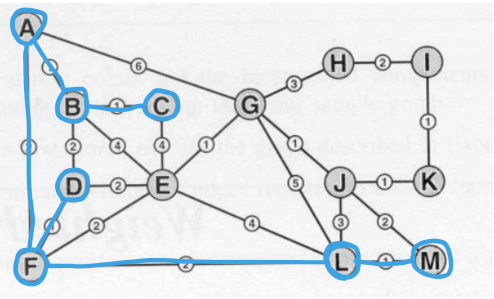
Heap[A, B, E, G, J]
Parent[0,0,0,6,0,12,0,0,0,0,0,0,12]
Dist[∞,∞,∞,1,∞,2,∞,∞,∞,∞,∞,0,1]



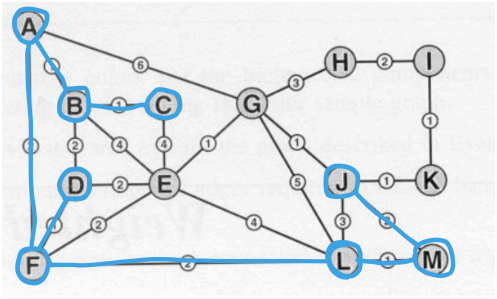
Heap[B, C, E, G, J]
Parent[6,0,0,6,0,12,0,0,0,0,0,0,12]
Dist[2,∞,∞,1,∞,2,∞,∞,∞,∞,∞,0,1]



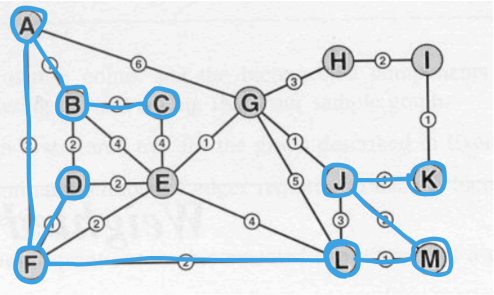
Heap[C, E, G, J]
Parent[6,1,0,6,0,12,0,0,0,0,0,0,12]
Dist[2,1,∞,1,∞,2,∞,∞,∞,∞,∞,0,1]



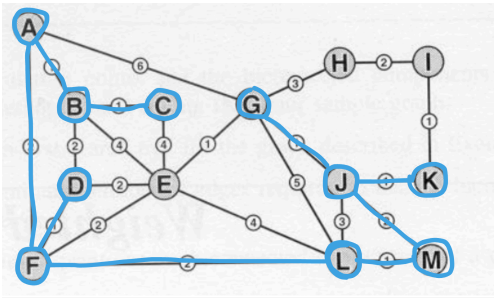
Heap[E, G, J]
Parent[6,1,2,6,0,12,0,0,0,0,0,0,12]
Dist[2,1,1,1,∞,2,∞,∞,∞,∞,∞,0,1]



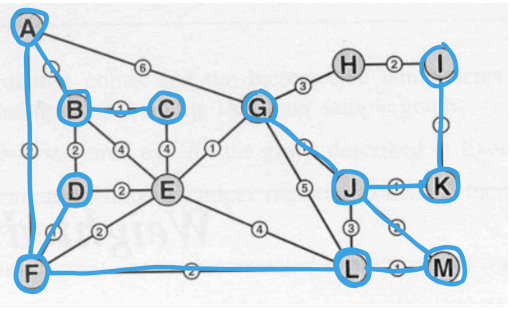
Heap[E, G, K]
Parent[6,1,2,6,0,12,0,0,0,13,0,0,12]
Dist[2,1,1,1,∞,2,∞,∞,∞,2,∞,0,1]



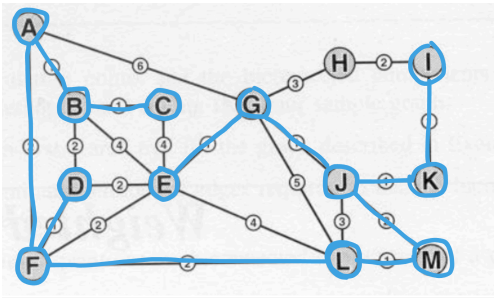
Heap[E, G, I]
Parent[6,1,2,6,0,12,0,0,0,13,11,0,12]
Dist[2,1,1,1,∞,2,∞,∞,∞,2,1,0,1]

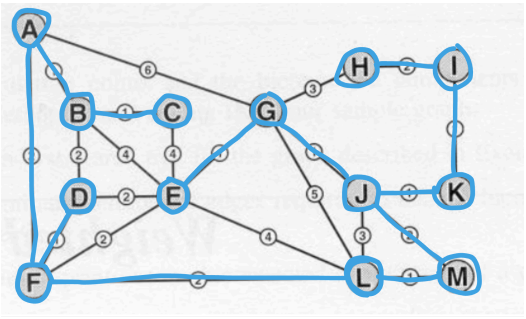


Heap[E, H, I]
Parent[6,1,2,6,0,12,10,0,0,13,11,0,12]
Dist[2,1,1,1,∞,2,1,∞,∞,2,1,0,1]



Heap[E, H]
Parent[6,1,2,6,0,12,10,0,11,13,11,0,12]
Dist[2,1,1,1,∞,2,1,∞,1,2,1,0,1]



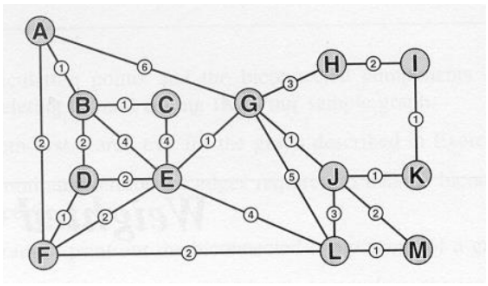


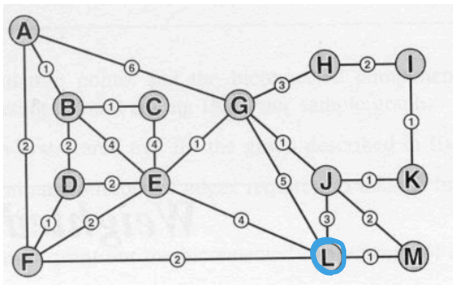
Heap[H]
Parent[6,1,2,6,7,12,10,0,11,13,11,0,12]
Dist[2,1,1,1,1,2,1,∞,1,2,1,0,1]

Heap[ ]
Parent[6,1,2,6,7,12,10,9,11,13,11,0,12]
Dist[2,1,1,1,1,2,1,2,1,2,1,0,1]

The last diagram is MST superimposed on the graph

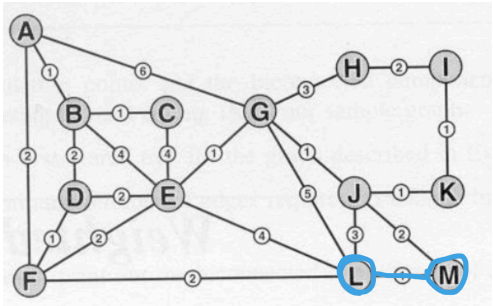
Construction of SST using Dijkstra's algorithm

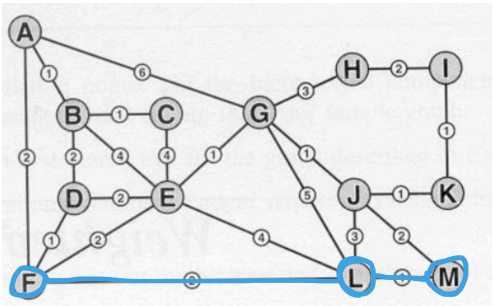




Heap[L (0)]

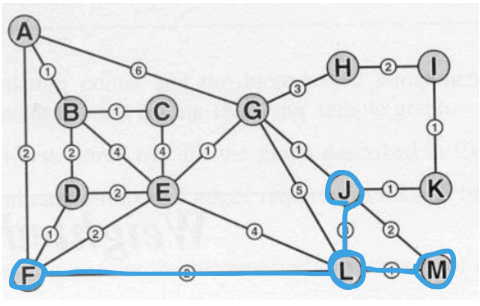
Heap[E (4), F (2), G (5), J (3), M (1)]

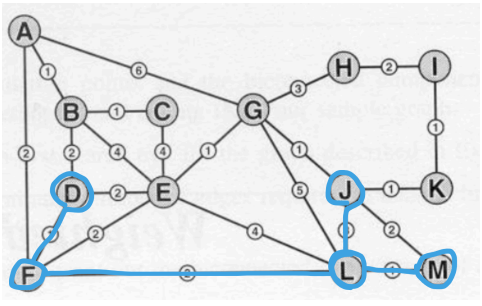




Heap[F (2), E (4), J (3), G (5)]

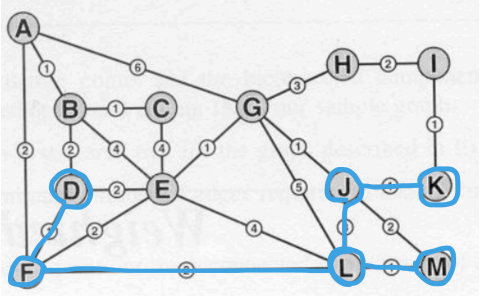
Heap[E (4), J (3), G (5), D (3), A(4)]



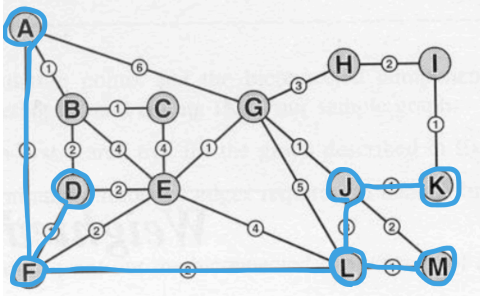


Heap[E (4), G (4), D (3), A(4), K(3)]

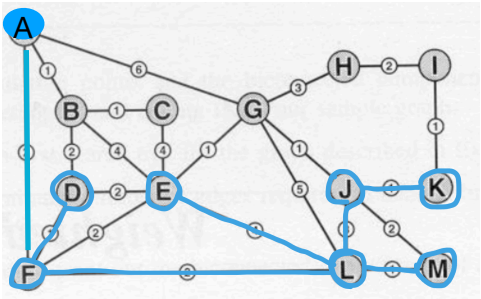
Heap[E (4), G (4), A(4), K(3), B(5)]



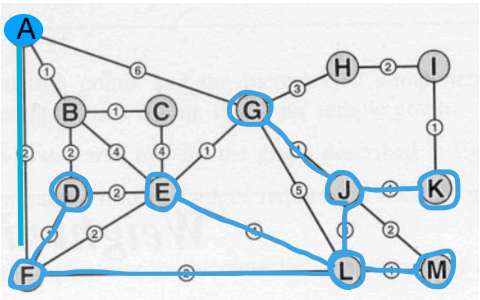
Heap[E (4), G (4), A(4), B(5), I(4)]



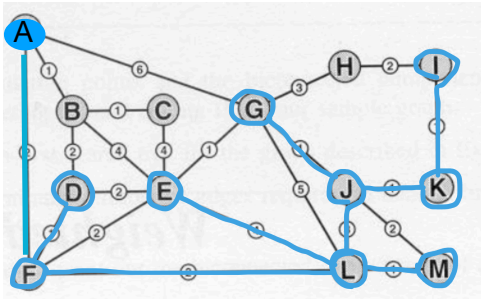
Heap[E (4), G (4), B(5), I(4)]



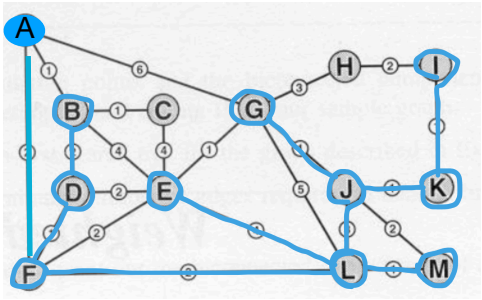
Heap[G(4), B(5), I(4), C(8)]



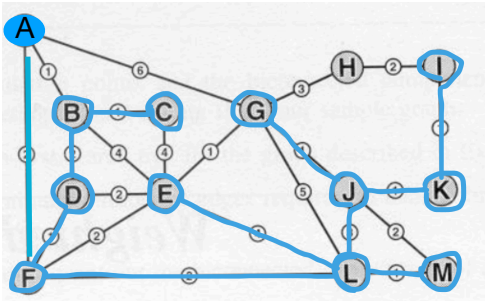
Heap [B(5), I(4), C(8), H(7)]



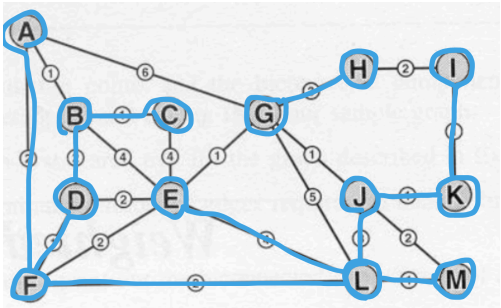
Heap[B(5), C(8), H(6)]



Heap[C(6), H(6)]



Heap[H(6)]

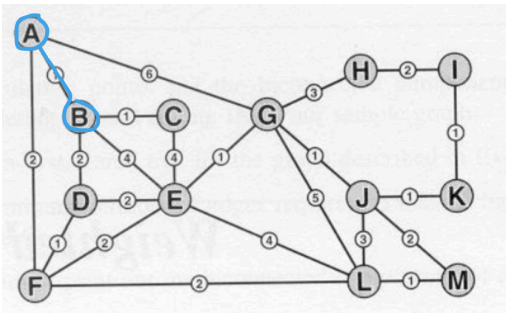
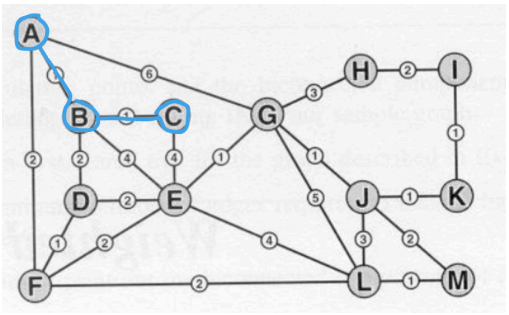
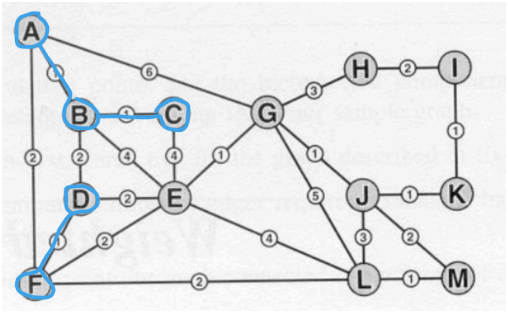
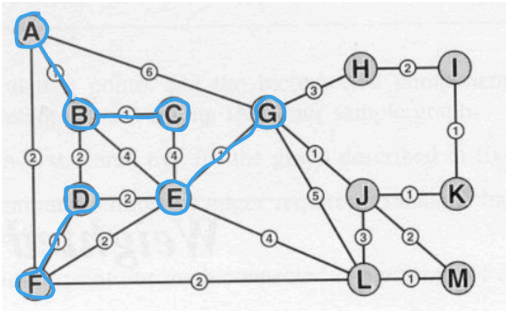
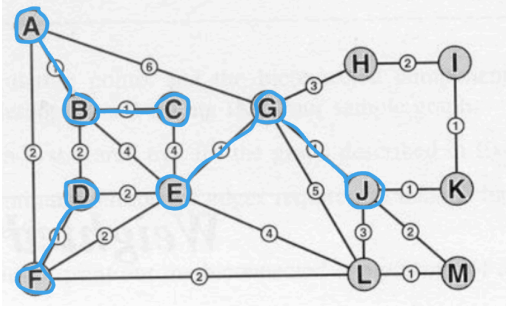
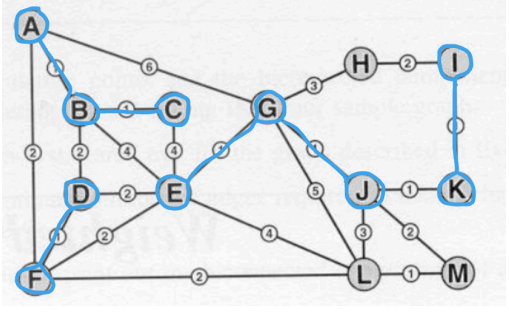


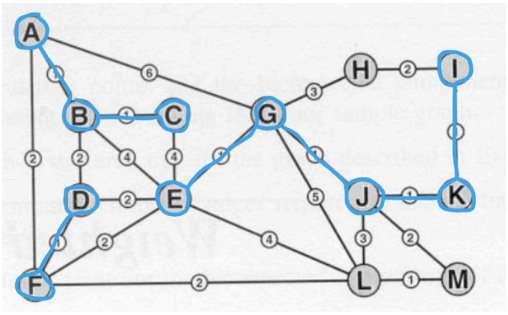
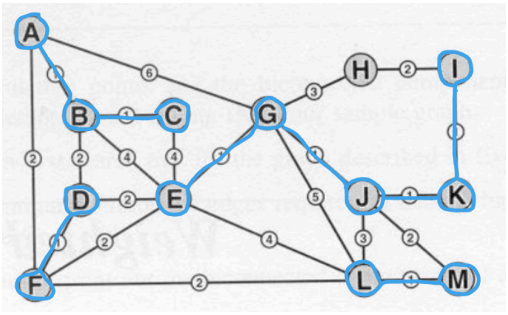
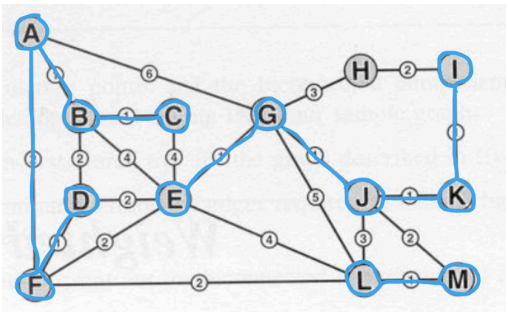
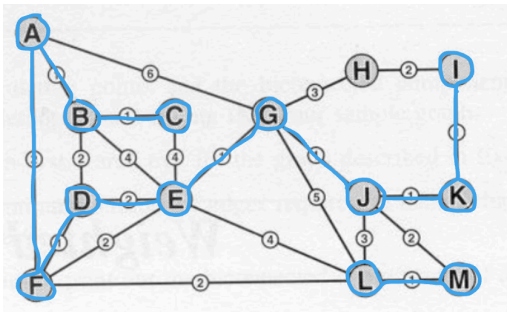
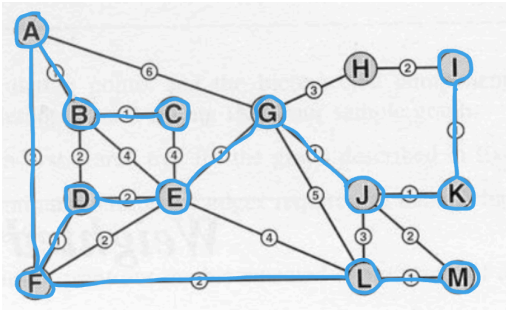
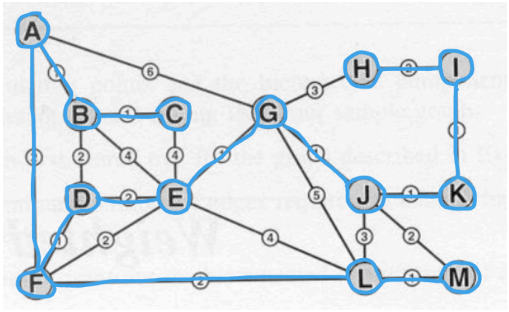
Heap[ ]

The last diagram is SPT superimposed on the graph

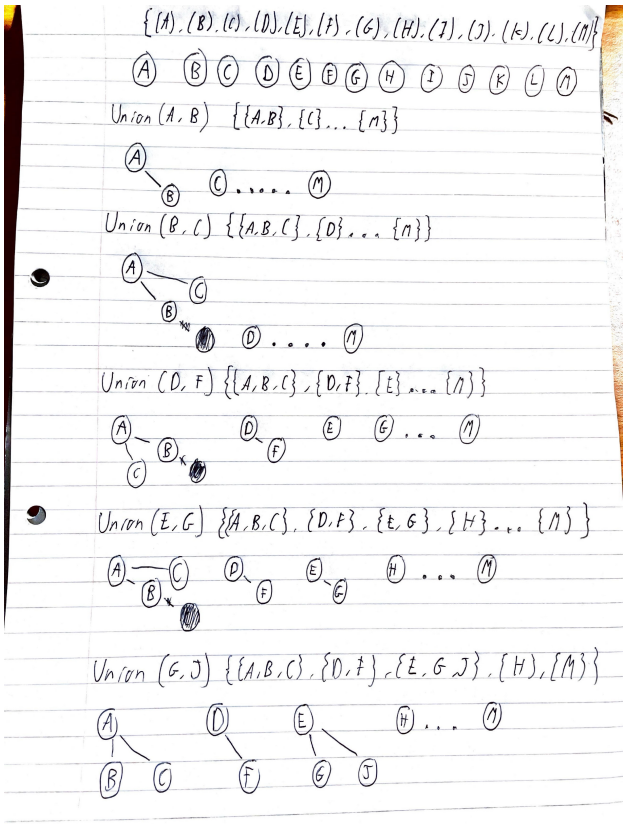
```
Current Vertex: 12
dist: 1[0, 2147483647, 2147483647, 2147483647, 2147483647, 4, 2, 5, 2147483647, 2147483647, 3, 2147483647, 0, 1]
parent: [0, -1, -1, -1, -1, 12, 12, 12, -1, -1, 12, -1, -1, 12]

Current Vertex: 13
dist: 2[0, 2147483647, 2147483647, 2147483647, 2147483647, 4, 2, 5, 2147483647, 2147483647, 3, 2147483647, 0, 1]
parent: [0, -1, -1, -1, -1, 12, 12, 12, -1, -1, 12, -1, -1, 12]

Current Vertex: 6
dist: 3[0, 4, 2147483647, 2147483647, 3, 4, 2, 5, 2147483647, 2147483647, 3, 2147483647, 0, 1]
parent: [0, 6, -1, -1, 6, 12, 12, 12, -1, -1, 12, -1, -1, 12]

Current Vertex: 10
dist: 4[0, 4, 2147483647, 2147483647, 3, 4, 2, 4, 2147483647, 2147483647, 3, 4, 0, 1]
parent: [0, 6, -1, -1, 6, 12, 12, 10, -1, -1, 12, 10, -1, 12]

Current Vertex: 4
dist: 5[0, 4, 5, 2147483647, 3, 4, 2, 4, 2147483647, 2147483647, 3, 4, 0, 1]
parent: [0, 6, 4, -1, 6, 12, 12, 10, -1, -1, 12, 10, -1, 12]

Current Vertex: 11
dist: 6[0, 4, 5, 2147483647, 3, 4, 2, 4, 2147483647, 5, 3, 4, 0, 1]
parent: [0, 6, 4, -1, 6, 12, 12, 10, -1, 11, 12, 10, -1, 12]

Current Vertex: 1
dist: 7[0, 4, 5, 2147483647, 3, 4, 2, 4, 2147483647, 5, 3, 4, 0, 1]
parent: [0, 6, 4, -1, 6, 12, 12, 10, -1, 11, 12, 10, -1, 12]

Current Vertex: 5
dist: 8[0, 4, 5, 8, 3, 4, 2, 4, 2147483647, 5, 3, 4, 0, 1]
parent: [0, 6, 4, 5, 6, 12, 12, 10, -1, 11, 12, 10, -1, 12]

Current Vertex: 7
dist: 9[0, 4, 5, 8, 3, 4, 2, 4, 7, 5, 3, 4, 0, 1]
parent: [0, 6, 4, 5, 6, 12, 12, 10, 7, 11, 12, 10, -1, 12]

Current Vertex: 9
dist: 10[0, 4, 5, 8, 3, 4, 2, 4, 7, 5, 3, 4, 0, 1]
parent: [0, 6, 4, 5, 6, 12, 12, 10, 7, 11, 12, 10, -1, 12]

Current Vertex: 2
dist: 11[0, 4, 5, 6, 3, 4, 2, 4, 7, 5, 3, 4, 0, 1]
parent: [0, 6, 4, 2, 6, 12, 12, 10, 7, 11, 12, 10, -1, 12]

Current Vertex: 3
dist: 12[0, 4, 5, 6, 3, 4, 2, 4, 7, 5, 3, 4, 0, 1]
parent: [0, 6, 4, 2, 6, 12, 12, 10, 7, 11, 12, 10, -1, 12]

Current Vertex: 8
dist: 13[0, 4, 5, 6, 3, 4, 2, 4, 7, 5, 3, 4, 0, 1]
parent: [0, 6, 4, 2, 6, 12, 12, 10, 7, 11, 12, 10, -1, 12]
Vertex  Distance from Source    Path
```

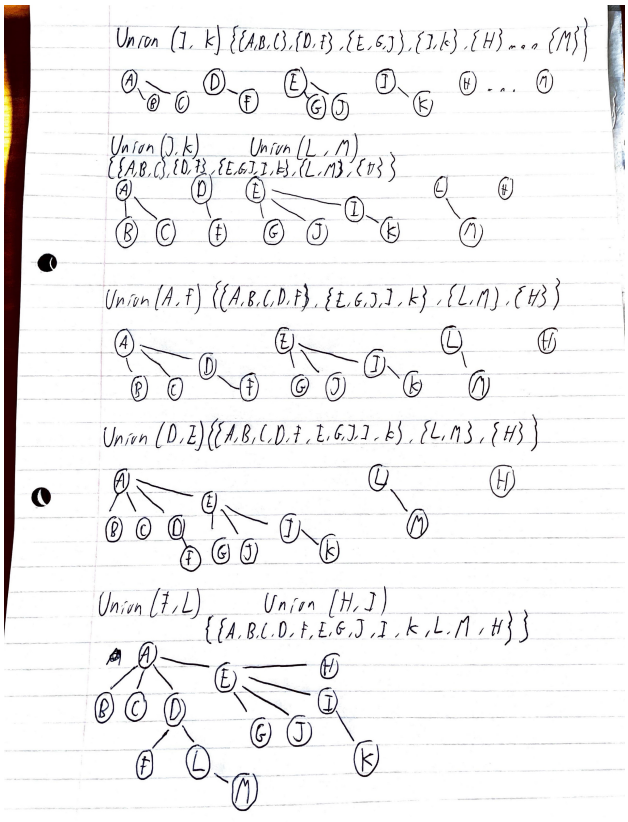
Step by step construction of Kruskal's MST

Union-find partition and set representations for each step





screen captures showing all my programs executing

```
aaronconnolly@Aarons-MacBook-Pro Assignment %  /usr/bin/env /Library/Java/JavaVirtualMachines/jdk-21.jdk/Contents/Ho
etailsInExceptionMessages -cp /Users/aaronconnolly/Library/Application\ Support/Code/User/workspaceStorage/c5888d001
Enter FileName
wGraph1.txt
Enter starting vertex;
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| ->

1. Prim's MST
2. Dijkstra SPT
3. Depth First
4. Breadth First
5. Exit
```

I decided to use a menu for easier representation of algorithms
Prim’s MST

```
1
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 E is connected to Vertex G with edge weight = 1
Vertex H is connected to Vertex I with edge weight = 2

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 -> L
M -> L
```

Dijkstra's SPT

```
Visited vertices in order: 12, 13, 6, 10, 4, 11, 1, 5, 7, 9, 2, 3, 8,
Vertex  Distance from Source    Path
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 -> 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
L               0               L
M               1               L -> M
```

Depth First Search

```
1. Prim's MST
2. Dijkstra SPT
3. Depth First
4. Breadth First
5. Exit
3
L E B A F D G H I K J M C
```

Breadth First Search

Kruskal's MST

```
aaronconnolly@Aarons-MacBook-Pro Kruskal %  /usr/bin/env /Library
ilsInExceptionMessages -cp /Users/aaronconnolly/Library/Applicati
Following are the edges in the constructed MST
A -- B == 1
B -- C == 1
D -- F == 1
E -- G == 1
G -- J == 1
I -- K == 1
J -- K == 1
L -- M == 1
A -- F == 2
D -- E == 2
F -- L == 2
H -- I == 2
Total weight of MST: 16
```

Conclusion

Overall, it was a fun challenging assignment. A lot of care and thought had to be put in translating the pseudocode to java and combining it with the skeleton code to create the MST and SPT algorithms. I found it a lot easier to translate the pseudocode into java code as time went on which I think is a great skill too have learnt. The given skeleton code was very useful along with the algorithms notes on Prim and Kruskal and Dijkstra that had outlined pseudocode.
The step by step constructions of each algorithm in the report were difficult to create at first but I now feel that I have a deep understanding on each algorithm, including the contents of the heap and dist/parent arrays at each step.