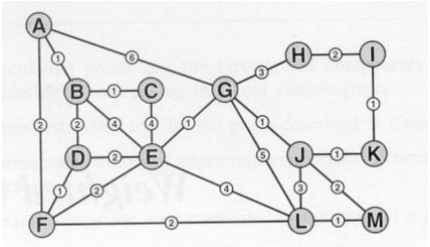
Graph Traversal, MST & SPT Algorithm Assignment

Introduction

This is my report for the Graph Traversal, MST & SPT Algorithm assignment. During the assignment, I implemented Depth First Traversal, Breadth First Traversal, Prim's algorithm for finding minimum spanning tree (MST) and Dijkstra's shortest path tree (SPT) all in java. My program prompts the user for the name of a text file which contains a graph and then prompts the user to enter what vertex they want to start at. Once this is done the graph in the text file will be read and the graph will be represented by an adjacency lists data structure. All of the methods for the four algorithms above are then called. Depth first traversal is Cormen's version and uses recursion. Breadth first traversal is Cormen's version and uses a queue, more specifically, a circular queue. Prim's algorithm and Dijkstra's algorithm both use a heap. I tested all of my code/algorithms on the sample graph below and converted it into the text file below. I started from vertex L in all my methods. All the algorithms I demonstrate in this report will be shown on the sample graph and starting from vertex L.



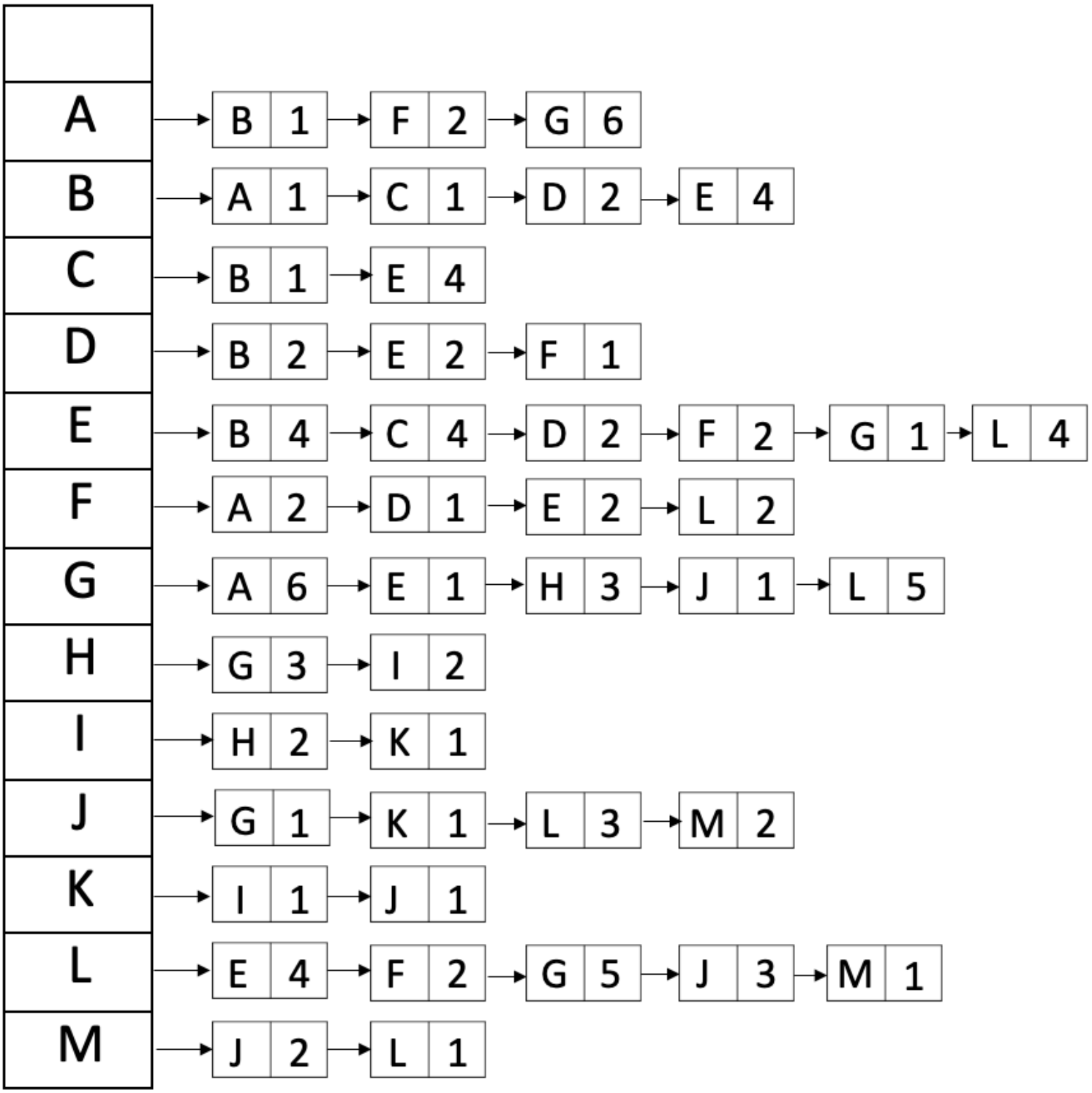

```
13 22
1 2 1
1 6 2
1 7 6
2 3 1
2 4 2
2 5 4
3 5 4
4 5 2
4 6 1
5 6 2
5 7 1
5 12 4
6 12 2
7 8 3
7 10 1
7 12 5
8 9 2
9 11 1
10 11 1
10 12 3
10 13 2
12 13 1
```

The report is structured as follows

1. Adjacency lists diagram
2. Step by Step Construction of MST using Prim's Algorithm.
3. Step by Step Construction of SPT using Dijkstra's Algorithm
4. Graph diagrams with MST, SPT, Depth first search and Breadth first search superimposed on them
5. Screen captures of my programs/methods executing on the sample graph
6. Challenging world graph and how Dijkstra performed on it
7. Analysis/Reflection on what I learned/found useful in the assignment

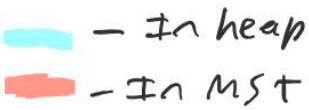
Adjacency Lists Diagram

This is an adjacency lists diagram showing the graph representation of the sample graph. This is how the graph was represented in the program as an adjacency list that is sorted.

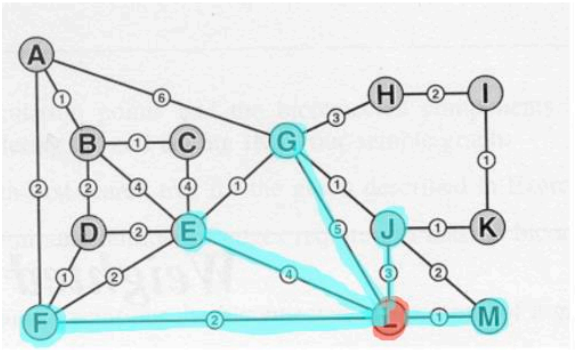


A step-by-step construction of the MST using Prim's algorithm

It starts from vertex L shows the heap, parent[] and dist[] arrays alongside the MST superimposed on the graph for each step in prim. In the code the parent array would hold integers that represent the numbers while here I have diagrammed it with the letters.

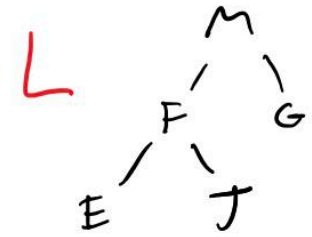


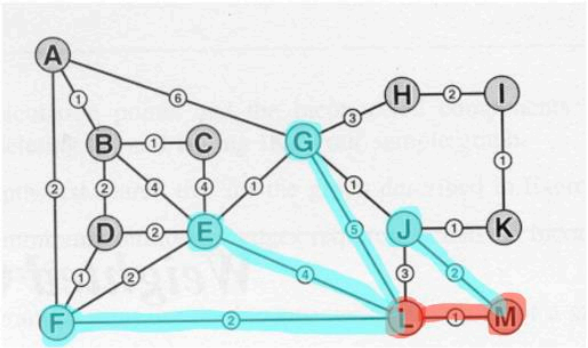
	A	B	C	D	E	F	G	H	I	J	K	L	M
	0	1	2	3	4	5	6	7	8	9	10	11	12
Parent		0	0	0	0	0	0	0	0	0	0	0	0
Dist		∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	0	∞



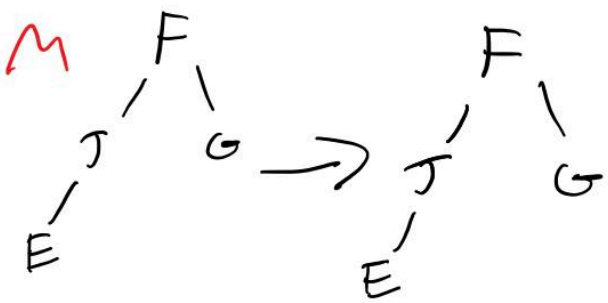
	A	B	C	D	E	F	G	H	I	J	K	L	M
	0	1	2	3	4	5	6	7	8	9	10	11	12
Parent		0	0	0	0	L	L	L	0	0	L	0	0	L
Dist		∞	∞	∞	∞	4	2	5	∞	∞	3	∞	0	1

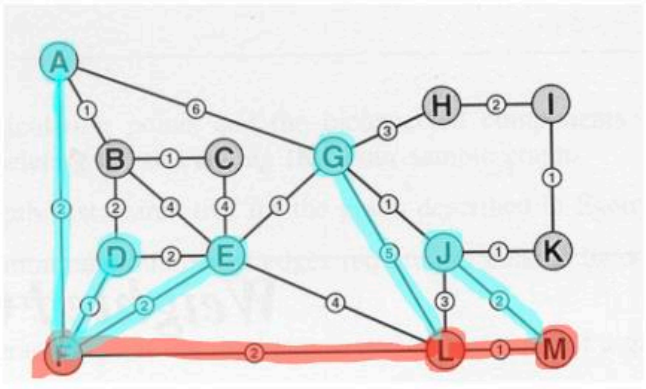
Heap





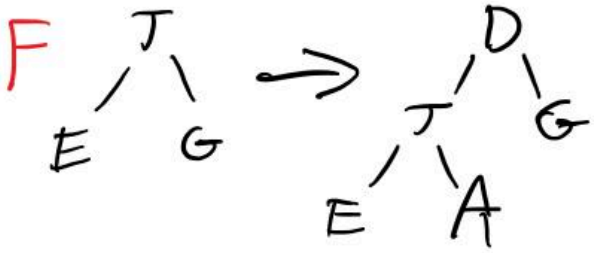
Heap

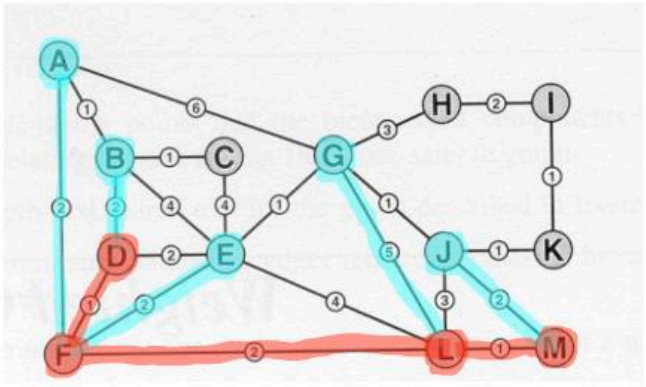




	A	B	C	D	E	F	G	H	I	J	K	L	M
	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Parent		F	0	0	F	F	L	L	0	0	M	0	0	L
Dist		~~∞~~ 2	∞	∞	~~∞~~ 1	~~4~~ 2	2	5	∞	∞	2	∞	0	1

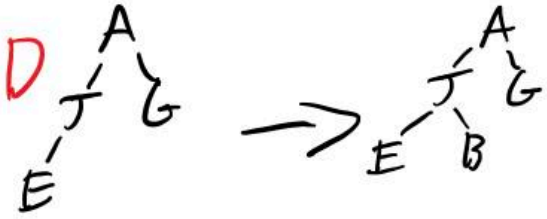
Heap

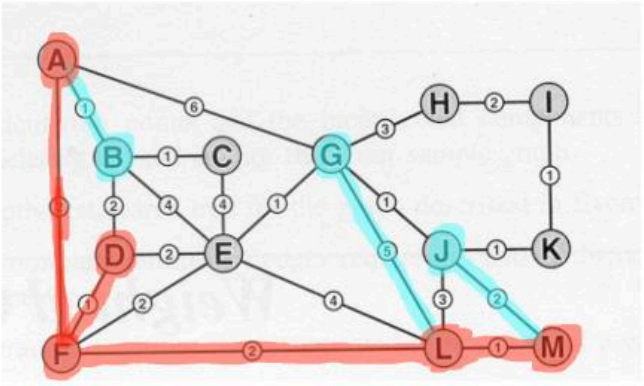




	A	B	C	D	E	F	G	H	I	J	K	L	M
	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Parent		F	~~D~~	0	F	F	L	L	0	0	M	0	0	L
Dist		2	~~∞~~ 2	∞	1	2	2	5	∞	∞	2	∞	0	1

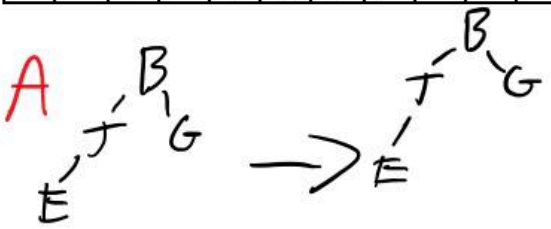
Heap

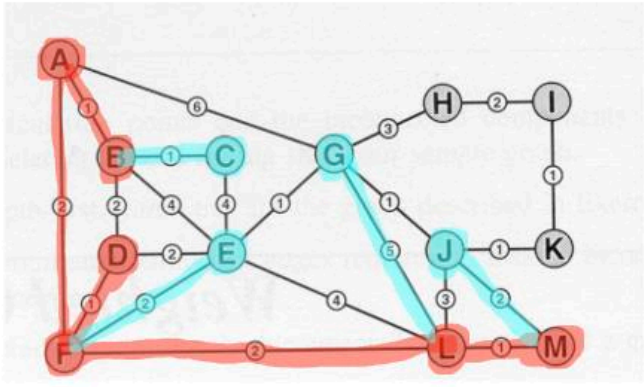




	A	B	C	D	E	F	G	H	I	J	K	L	M
	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Parent		F	~~A~~ B	0	F	F	L	L	0	0	M	0	0	L
Dist		2	~~2~~ 1	∞	1	2	2	5	∞	∞	2	∞	0	1

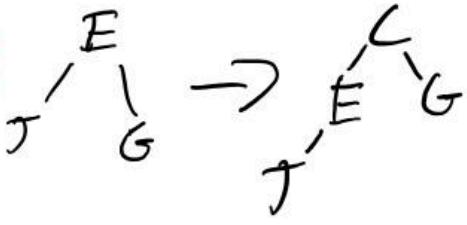
Heap

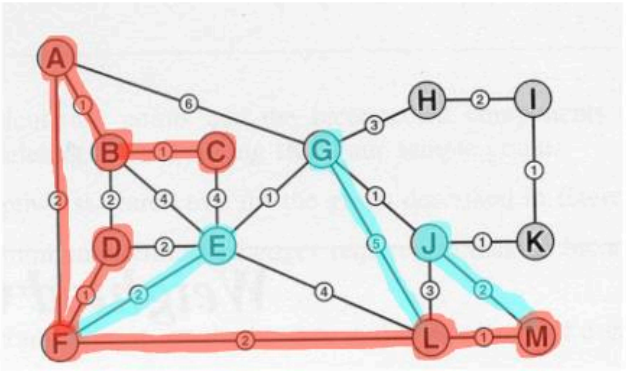




	A	B	C	D	E	F	G	H	I	J	K	L	M
	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Parent		F	A	~~B~~ 0	F	F	L	L	0	0	M	0	0	L
Dist		2	1	~~∞~~ 0	1	2	2	5	∞	∞	2	∞	0	1

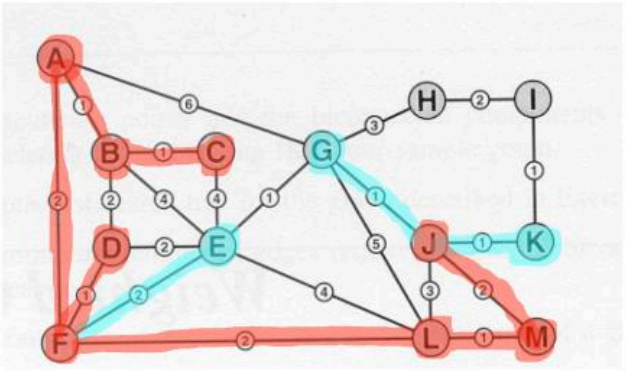
Heap





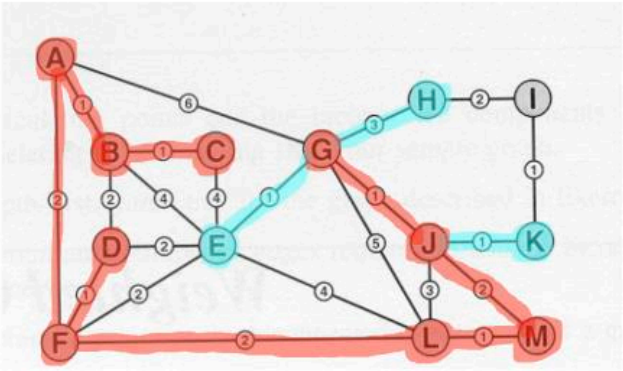
	A	B	C	D	E	F	G	H	I	J	K	L	M
	0	1	2	3	4	5	6	7	8	9	10	11	12
Parent		F	A	B	F	F	L	C	O	O	M	O	O	L
Dist		2	1	1	1	2	2	5	∞	∞	2	∞	0	1

Heap



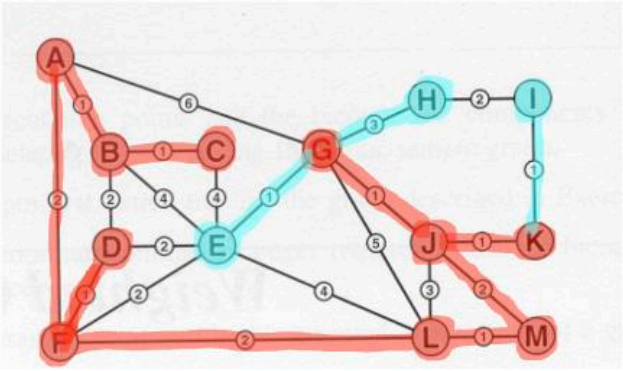
	A	B	C	D	E	F	G	H	I	J	K	L	M
	0	1	2	3	4	5	6	7	8	9	10	11	12
Parent		F	A	B	F	F	L	J	O	O	M	J	O	L
Dist		2	1	1	1	2	2	5	∞	∞	2	∞	0	1

Heap



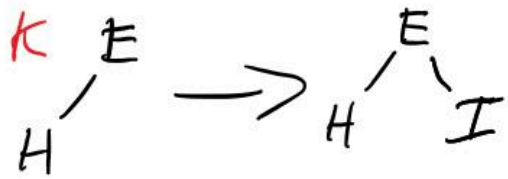
	A	B	C	D	E	F	G	H	I	J	K	L	M
	0	1	2	3	4	5	6	7	8	9	10	11	12
Parent		F	A	B	F	G	L	J	G	O	M	J	O	L
Dist		2	1	1	1	1	2	1	3	∞	2	1	0	1

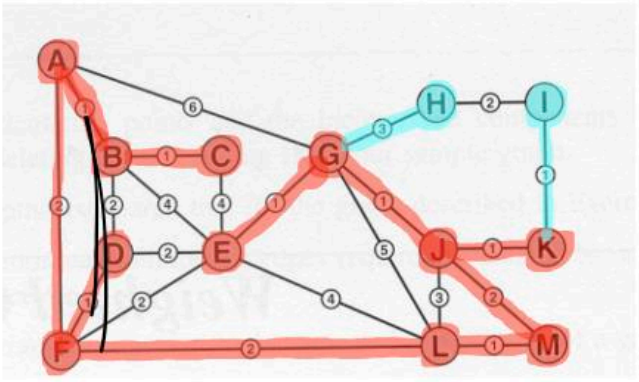
Heap



	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
Parent		F	A	B	F	G	L	J	G	K	M	J	O	L
Dist		2	1	1	1	1	2	1	3	∞	2	1	0	1

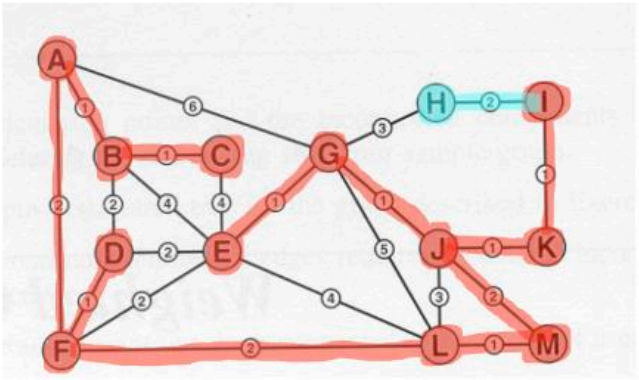
Heap





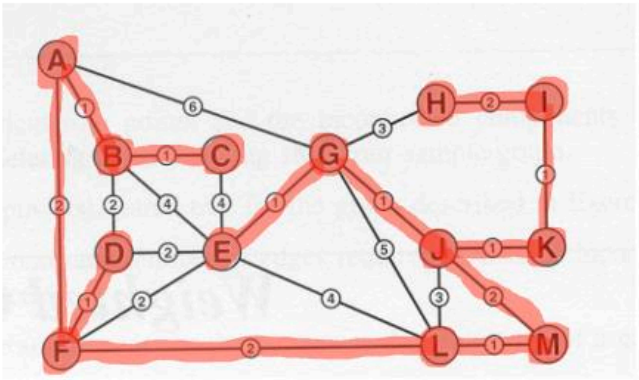
	A	B	C	D	E	F	G	H	I	J	K	L	M
	0	1	2	3	4	5	6	7	8	9	10	11	12
Parent		F	A	B	F	G	L	J	G	K	M	J	O	L
Dist		2	1	1	1	1	2	1	3	1	2	1	0	1

Heap E I H



	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
Parent		F	A	B	F	G	L	J	~~G~~ F	K	M	J	O	L
Dist		2	1	1	1	1	2	1	~~3~~ 2	1	2	1	0	1

Heap I H



	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
Parent		F	A	B	F	G	L	J	I	K	M	J	O	L
Dist		2	1	1	1	1	2	1	2	1	2	1	0	1

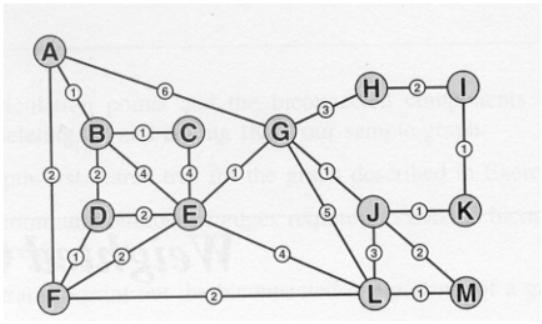
Heap H Heap is now empty

MST complete

Total weight of MST is 16

A step-by-step construction of SPT using Dijkstra's algorithm

It starts from vertex L shows the heap, parent[] and dist[] arrays. In the code the parent array would hold integers that represent the numbers while here I have diagrammed it with the letters.

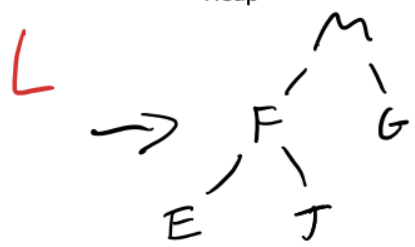


	A	B	C	D	E	F	G	H	I	J	K	L	M
	0	1	2	3	4	5	6	7	8	9	10	11	12
Parent		0	0	0	0	0	0	0	0	0	0	0	0
Dist		∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	0	∞

Heap

L

Heap



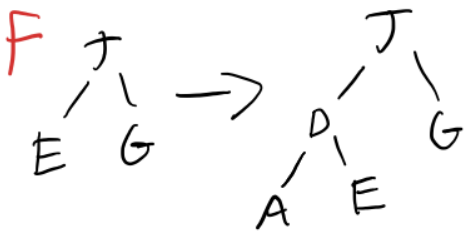
	A	B	C	D	E	F	G	H	I	J	K	L	M
	0	1	2	3	4	5	6	7	8	9	10	11	12
Parent		0	0	0	0	L	L	L	0	0	L	0	0	L
Dist		∞	∞	∞	∞	4	2	5	∞	∞	3	∞	0	1

Heap

M F J G E

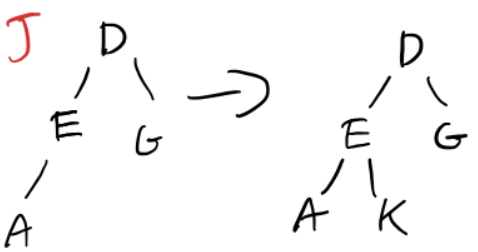
	A	B	C	D	E	F	G	H	I	J	K	L	M
	0	1	2	3	4	5	6	7	8	9	10	11	12
Parent		0	0	0	0	L	L	L	0	0	L	0	0	L
Dist		∞	∞	∞	∞	4	2	5	∞	∞	3	∞	0	1

Heap



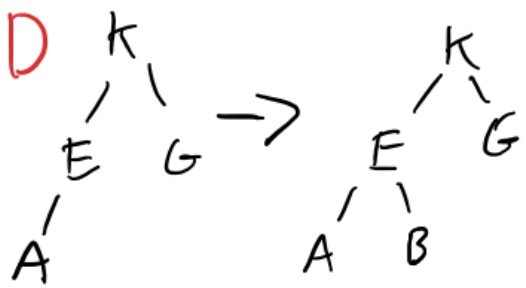
	A	B	C	D	E	F	G	H	I	J	K	L	M
	0	1	2	3	4	5	6	7	8	9	10	11	12
Parent		F	0	0	F	L	L	L	0	0	L	0	0	L
Dist		4	∞	∞	3	4	2	5	∞	∞	3	∞	0	1

Heap



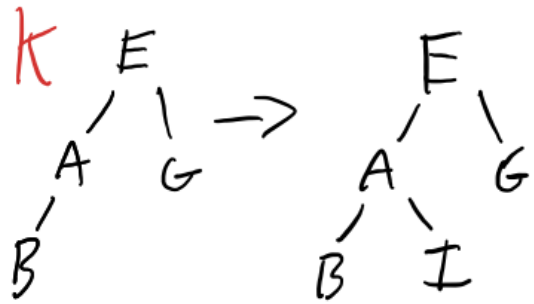
	A	B	C	D	E	F	G	H	I	J	K	L	M
	0	1	2	3	4	5	6	7	8	9	10	11	12
Parent		F	0	0	F	L	L	J	0	0	L	0	0	L
Dist		4	∞	∞	3	4	2	5	∞	∞	3	∞	0	1

Heap



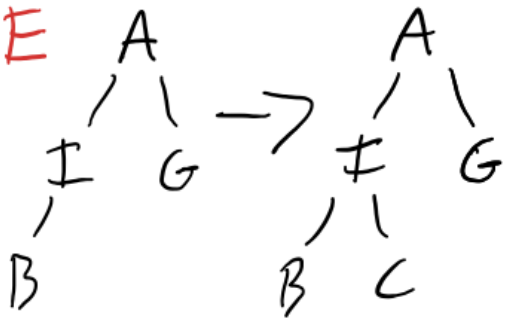
	A	B	C	D	E	F	G	H	I	J	K	L	M
	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Parent		F	D	∅	F	L	L	J	∅	∅	L	J	∅	L
Dist		4	~~∞~~ 5	∞	3	4	2	4	∞	∞	3	4	0	1

Heap



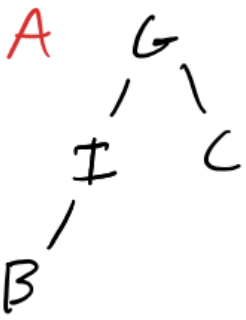
	A	B	C	D	E	F	G	H	I	J	K	L	M
	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Parent		F	D	∅	F	L	L	J	∅	K	L	J	∅	L
Dist		4	5	∞	3	4	2	4	∞	~~∞~~ 5	3	4	0	1

Heap



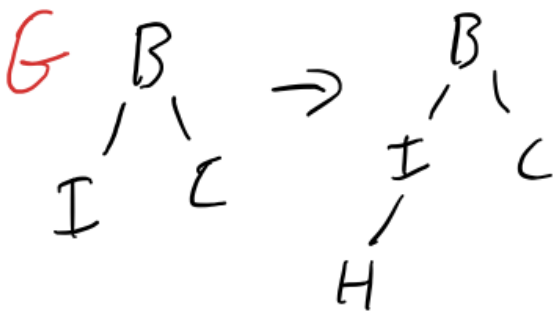
	A	B	C	D	E	F	G	H	I	J	K	L	M
	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Parent		F	D	~~∅~~ E	F	L	L	J	∅	K	L	J	∅	L
Dist		4	5	~~∞~~ 8	3	4	2	4	∞	5	3	4	0	1

Heap



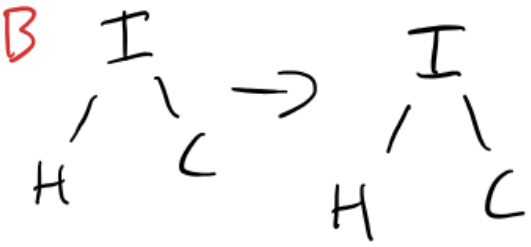
	A	B	C	D	E	F	G	H	I	J	K	L	M
	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Parent		F	D	E	F	L	L	J	∅	K	L	J	∅	L
Dist		4	5	8	3	4	2	4	∞	5	3	4	0	1

Heap



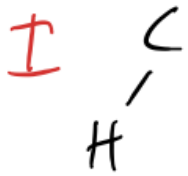
	A	B	C	D	E	F	G	H	I	J	K	L	M
	0	1	2	3	4	5	6	7	8	9	10	11	12
Parent		F	D	E	F	L	L	J	~~G~~	K	L	J	O	L
Dist		4	5	8	3	4	2	4	~~0~~ 7	5	3	4	0	1

Heap



	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
Parent		F	D	~~E~~ B	F	L	L	J	G	K	L	J	O	L
Dist		4	5	~~8~~ 6	3	4	2	4	7	5	3	4	0	1

Heap



	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
Parent		F	D	B	F	L	L	J	G	K	L	J	O	L
Dist		4	5	6	3	4	2	4	7	5	3	4	0	1

Heap

	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
Parent		F	D	B	F	L	L	J	G	K	L	J	O	L
Dist		4	5	6	3	4	2	4	7	5	3	4	0	1

Heap

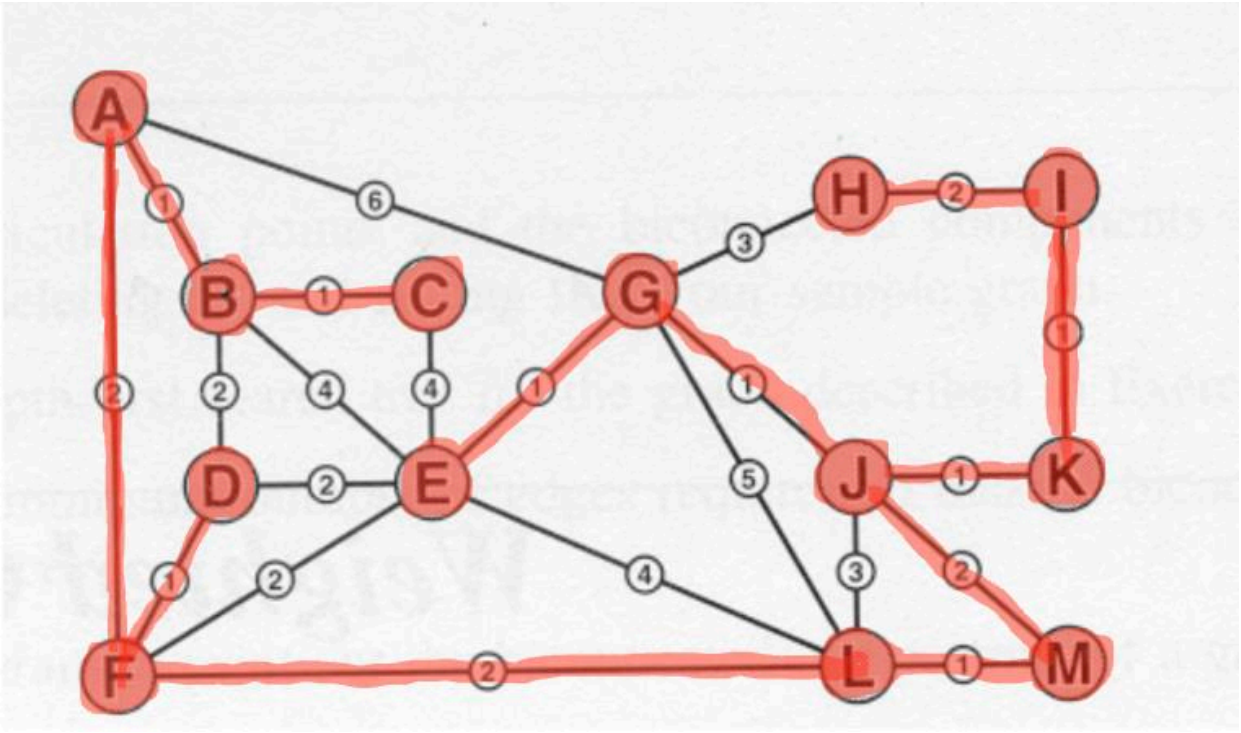
H Heap is now empty

	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
Parent		F	D	B	F	L	L	J	G	K	L	J	O	L
Dist		4	5	6	3	4	2	4	7	5	3	1	0	1

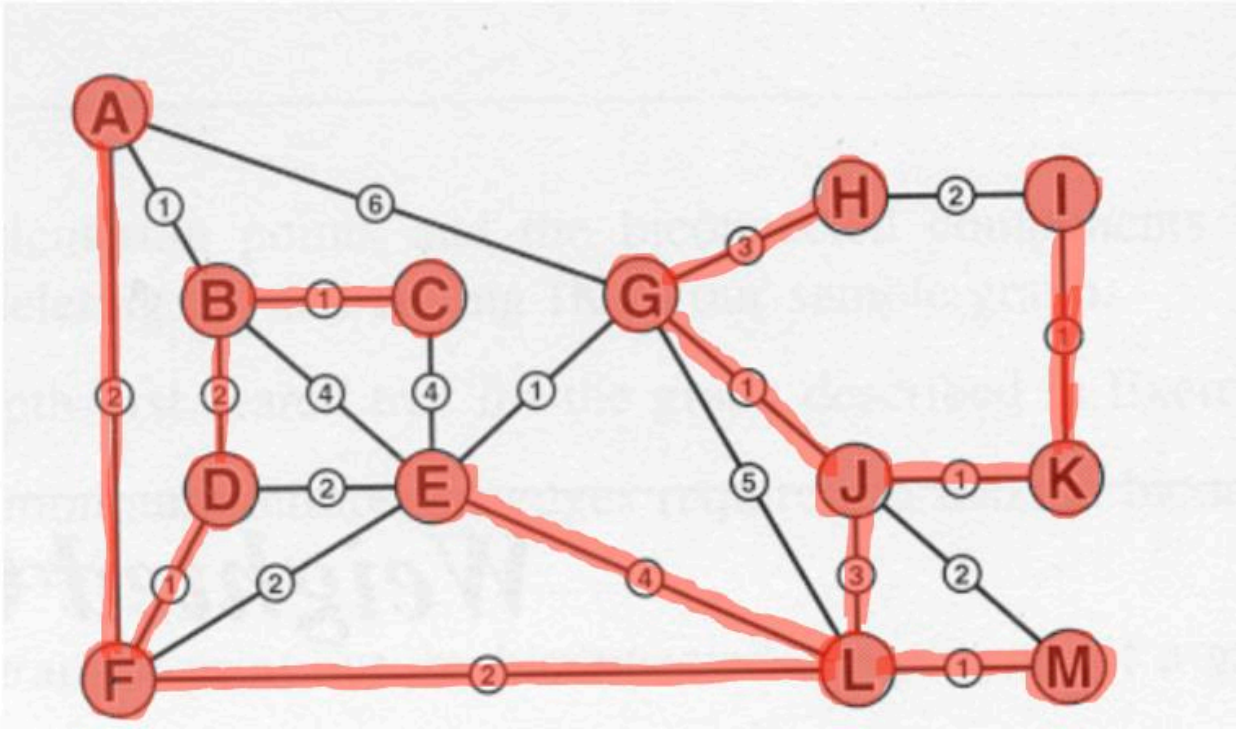
Spt complete

Graph diagrams with MST, SPT, Depth first search and Breadth first search superimposed on them

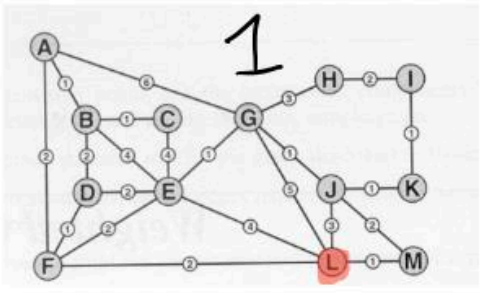
MST

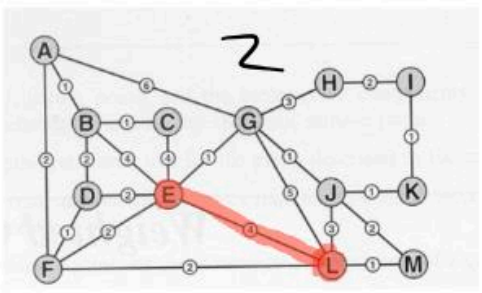


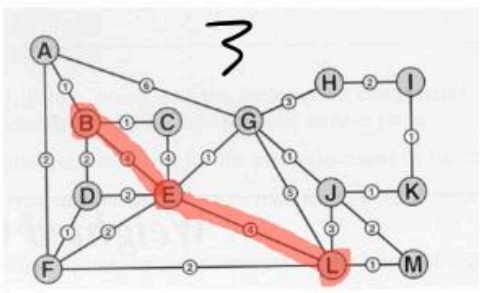
SPT

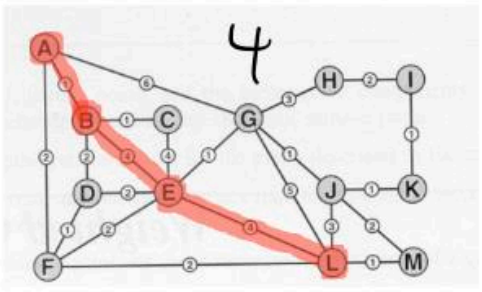


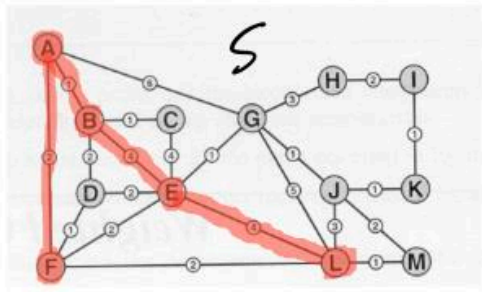
Depth First Search

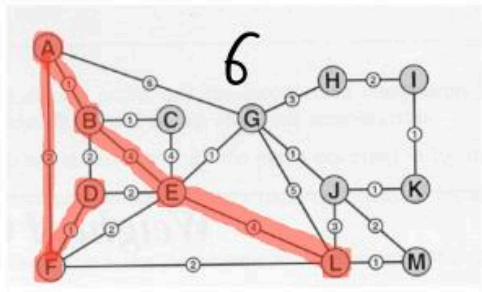


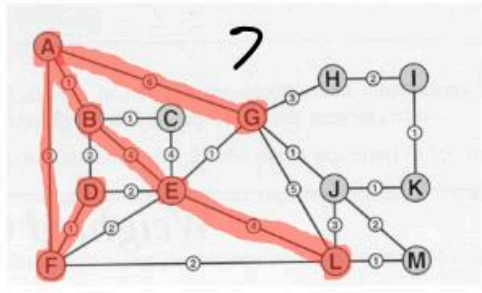


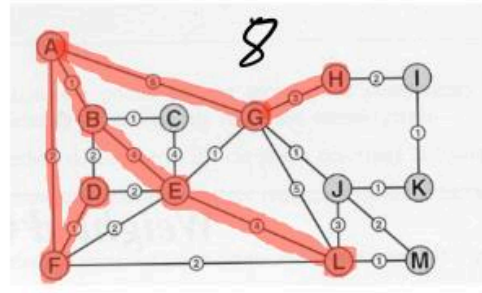


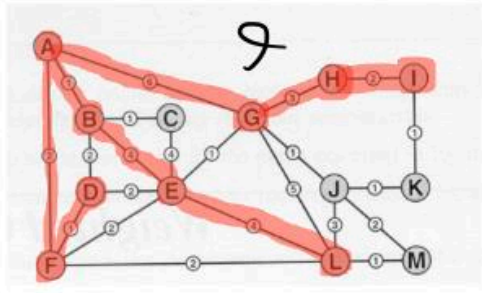


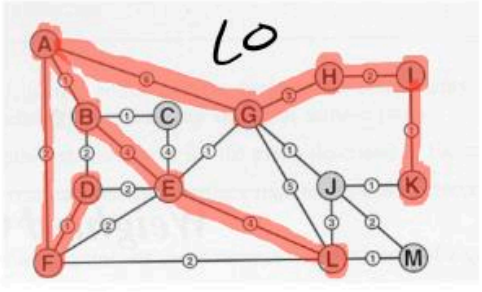


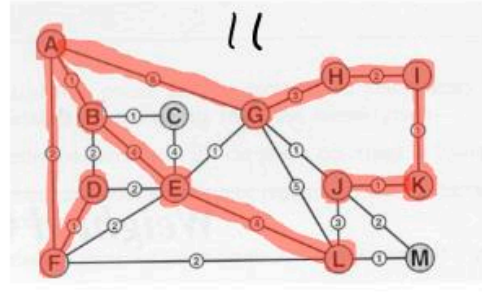


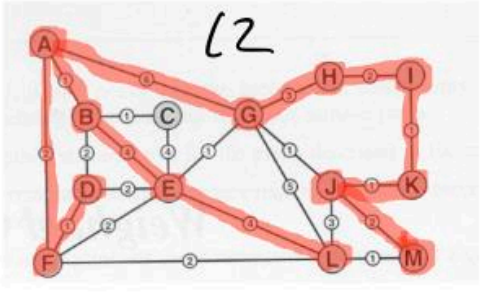


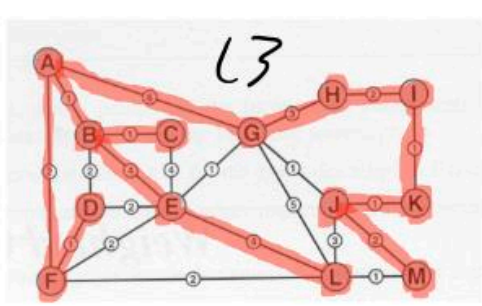




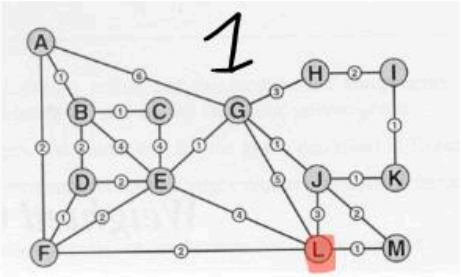


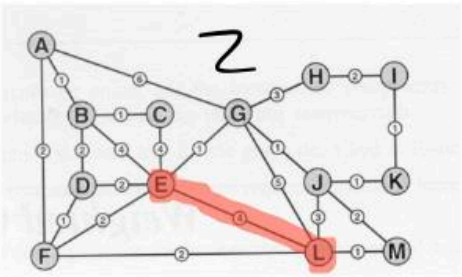


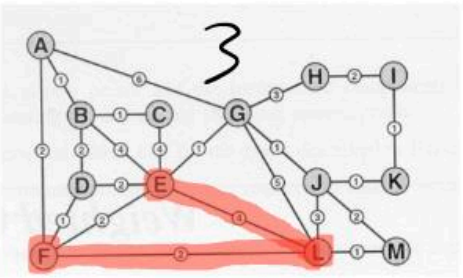


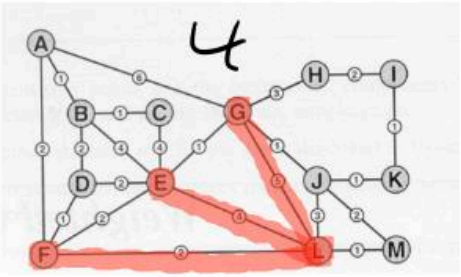


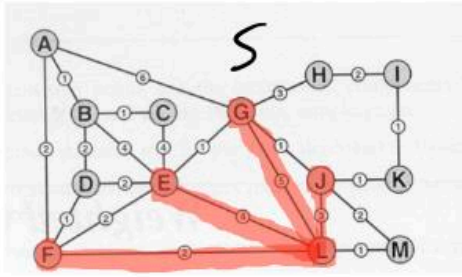
Breadth First Search

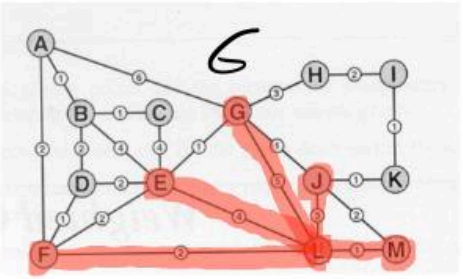


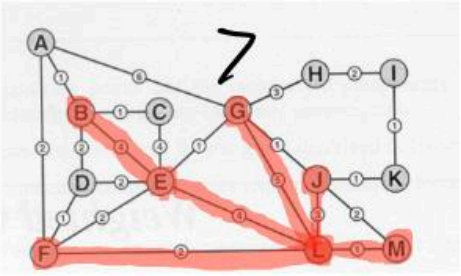


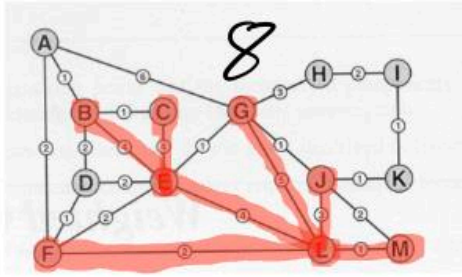


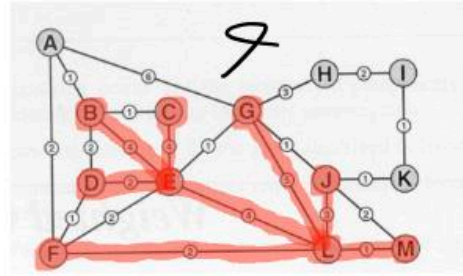


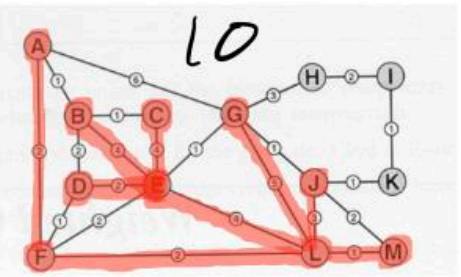


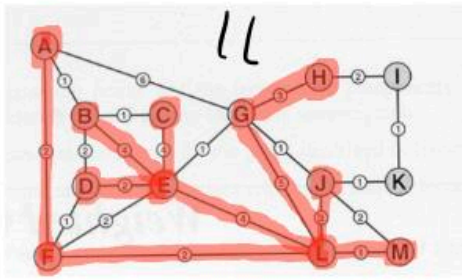


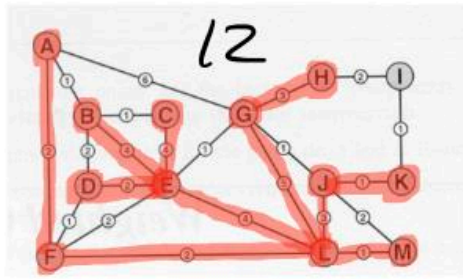


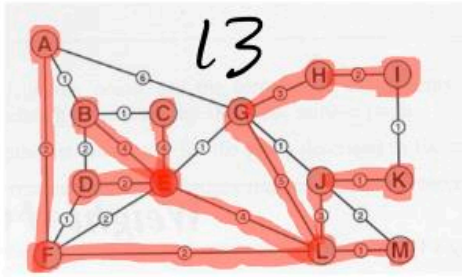












Screen Captures of all programs executing on the sample graph saved as wGraph1.txt graph starting from vertex L

```
(base) bencostello@MacBook-Pro Assignment % java GraphLists
Please enter the text file that contains a sample graph you want to use:
wGraph1.txt
Please enter the vertex you want to start at:
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] -> |B | 1| -> |F | 2| -> |G | 6| ->
adj[B] -> |A | 1| -> |C | 1| -> |D | 2| -> |E | 4| ->
adj[C] -> |B | 1| -> |E | 4| ->
adj[D] -> |B | 2| -> |E | 2| -> |F | 1| ->
adj[E] -> |B | 4| -> |C | 4| -> |D | 2| -> |F | 2| -> |G | 1| -> |L | 4| ->
adj[F] -> |A | 2| -> |D | 1| -> |E | 2| -> |L | 2| ->
adj[G] -> |A | 6| -> |E | 1| -> |H | 3| -> |J | 1| -> |L | 5| ->
adj[H] -> |G | 3| -> |I | 2| ->
adj[I] -> |H | 2| -> |K | 1| ->
adj[J] -> |G | 1| -> |K | 1| -> |L | 3| -> |M | 2| ->
adj[K] -> |I | 1| -> |J | 1| ->
adj[L] -> |E | 4| -> |F | 2| -> |G | 5| -> |J | 3| -> |M | 1| ->
adj[M] -> |J | 2| -> |L | 1| ->
```

```
------------------------------------
Depth First Graph Traversal
Starting with Vertex L

v = 12
DF just visited vertex L along edge @--L

v = 5
DF just visited vertex E along edge L--E

v = 2
DF just visited vertex B along edge E--B

v = 1
DF just visited vertex A along edge B--A

v = 6
DF just visited vertex F along edge A--F

v = 4
DF just visited vertex D along edge F--D

v = 7
DF just visited vertex G along edge A--G

v = 8
DF just visited vertex H along edge G--H

v = 9
DF just visited vertex I along edge H--I

v = 11
DF just visited vertex K along edge I--K

v = 10
DF just visited vertex J along edge K--J

v = 13
DF just visited vertex M along edge J--M

v = 3
DF just visited vertex C along edge B--C
```

```
------------------------------------
Breadth First Graph Traversal
Starting with Vertex L

v = 12
BF just visited vertex L along edge @--L

v = 5
BF just visited vertex E along edge L--E

v = 6
BF just visited vertex F along edge L--F

v = 7
BF just visited vertex G along edge L--G

v = 10
BF just visited vertex J along edge L--J

v = 13
BF just visited vertex M along edge L--M

v = 2
BF just visited vertex B along edge E--B

v = 3
BF just visited vertex C along edge E--C

v = 4
BF just visited vertex D along edge E--D

v = 1
BF just visited vertex A along edge F--A

v = 8
BF just visited vertex H along edge G--H

v = 11
BF just visited vertex K along edge J--K

v = 9
BF just visited vertex I along edge H--I
```

```
-----------------------------------
Prim's Minimum Spanning Tree Algorithm
Starting with Vertex L

v = 12
MST_Prim just inserted L into MST along edge @--L

v = 13
MST_Prim just inserted M into MST along edge L--M

v = 6
MST_Prim just inserted F into MST along edge L--F

v = 4
MST_Prim just inserted D into MST along edge F--D

v = 1
MST_Prim just inserted A into MST along edge F--A

v = 2
MST_Prim just inserted B into MST along edge A--B

v = 3
MST_Prim just inserted C into MST along edge B--C

v = 10
MST_Prim just inserted J into MST along edge M--J

v = 7
MST_Prim just inserted G into MST along edge J--G

v = 11
MST_Prim just inserted K into MST along edge J--K

v = 5
MST_Prim just inserted E into MST along edge G--E

v = 9
MST_Prim just inserted I into MST along edge K--I

v = 8
MST_Prim just inserted H into MST along edge I--H


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 Shortest Path Tree Algorithm
Starting with Vertex L

Shortest Path Tree is:
Vertex   Parent    Distance from 12
   L        @              0
   M        L              1
   F        L              2
   J        L              3
   D        F              3
   K        J              4
   E        L              4
   A        F              4
   G        J              4
   B        D              5
   I        K              5
   C        B              6
   H        G              7
```

Challenging world graph and how Dijkstra performed on it

My challenging world graph is about the locations in Sweden which is the graph you gave us in .tsp format which didn't include the vertices and edge weights. As there wasn't a suitable example online and following on from my email, and the discussion I had with you, I amended the file with the assistance of ChatGPT in this instance, so it was in the correct text format. The text file includes the first 5000 vertices and all vertices have 1 to 3 edges connected to them. It has 5000 vertices and 6310 edges. In my test I decided to start from vertex 12. When I ran Dijkstra on it, I made a note of the running time and memory usage. Below are two outputs which I got.

```
The running of the SPT code used 10552 bytes_of memory  and took 24179583 nanoseconds
The running of the SPT code used 10664 bytes_of memory  and took 20465375 nanoseconds
```

I ran it 10 times and got an average memory usage of 10612 bytes and a running time of 22,410,350 nanoseconds.

Analysis/Reflection on what I learned/found useful in the assignment

During this assignment I learnt many new things. I found it very useful to implement breadth first, depth first, MST and SPT as it helped strengthen my understanding and makes me feel that I could implement them in other programs in the future. Alongside strengthening my understanding, I have learnt that there are many methods and ways to implement these algorithms. I also found it useful to create a step by step construction of MST and SPT for this report as from my code I had an understanding of what was going on but by doing the construction it reinforced what I knew. Furthermore, I feel my java has also improved. Getting to use java and build algorithms helped me understand how java works and how to code using it. I now know how to create queues, heaps, breadth first search, depth first search, SPT and MST using it. Though this assignment was done in java I now feel that I would be able to implement/translate the code from java to another coding language. Overall, this assignment has been very interesting and informative and has given me a very good overview of breadth first search, depth first search, MST and SPT.