GRAPH TRAVERSAL & MST ALGORITHM

Algorithms and Data Structures | CMPU 2001

Aayush Gaur | C20396243

Algorithm Assignment

C20396243 Aayush Gaur

Introduction

Depth First Traversal

Depth first traversal starts at a root node and explores as far as possible before backtracking. It may use a stack or recursion. After visiting a vertex it adds its children onto the stack. This way the algorithm keeps going until the vertex no longer has a child, then it backtracks.

Breadth First Traversal

It will give the shortest path spanning tree on an unweighted graph. It uses a queue, by adding the children of a vertex into the queue. Hence it visits all the vertices on the same level before proceeding to a deeper level.

Prim's Algorithm

Prim's algorithm is a greedy algorithm that finds the minimum spanning tree for a connected weighted undirected graph. It maintains 2 sets of vertices. The first set contains the vertices included in the MST, the second contains the vertices not yet included. Every step it considers all the edges that connect the two sets (1 vertex from each set), and picks the edge with the lowest weight. After picking the edge, it moves the vertex from the second set to the set containing MST. This way it forms a Minimum Spanning Tree.

Kruskal's MST Algorithm

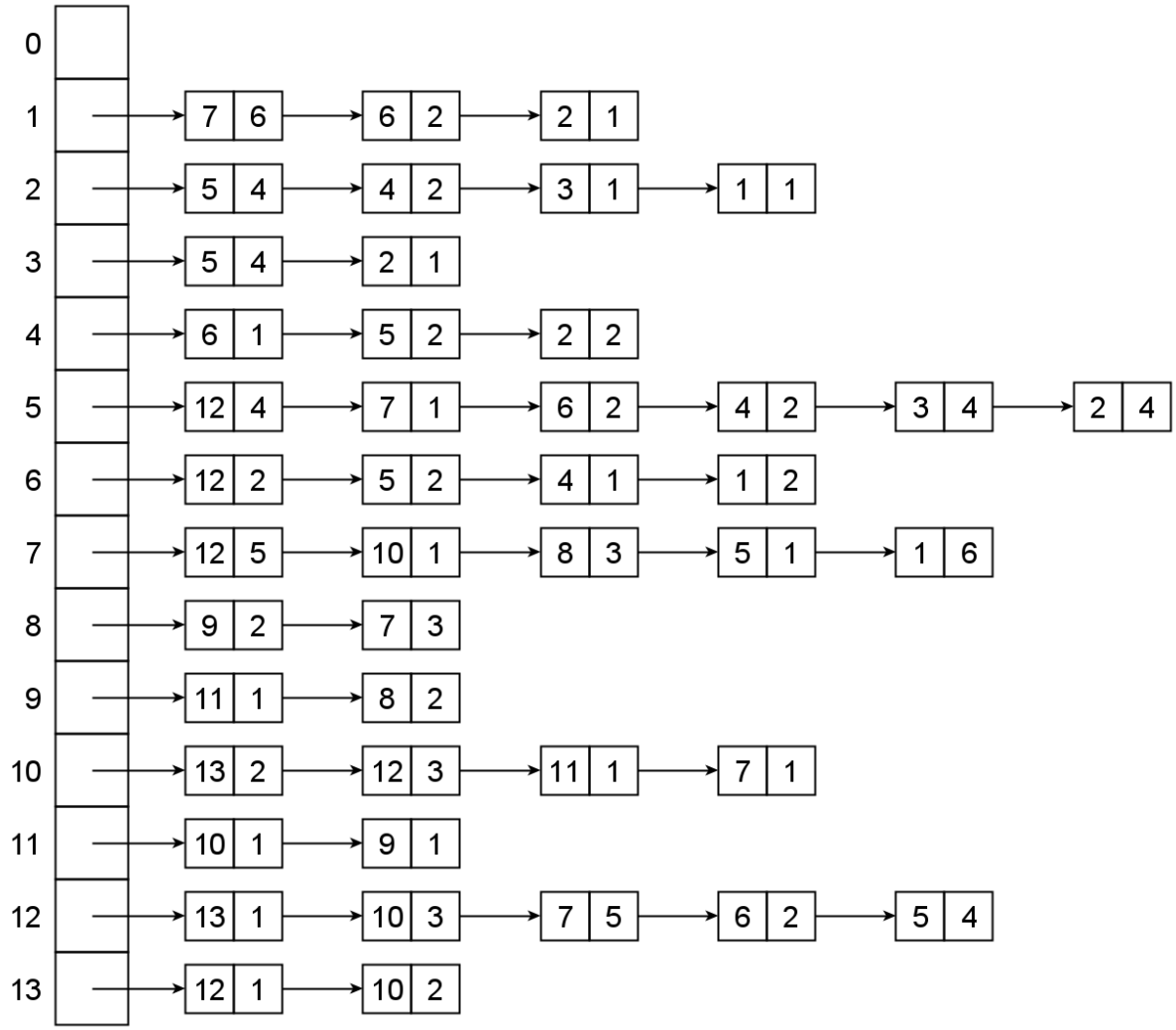
It is an algorithm to find the minimum spanning tree for a connected weighted graph. It sorts all the edges in an ascending order of their weight. This can be done using a heap or any type or a sorting method. It then picks the edge with the minimum edge (remove from a minimum heap). Checks if the edge forms a cycle. If it does not, then it is added to the MST, else it is rejected.

In my program, this is done by using sets. First it creates a forest of singleton sets. Then it removes the lowest edge from the heap and check if the 2 vertices are of different sets or not. If they are of different sets, combine the sets into one set. This is done until all the vertices are added to the MST.

Path Compression: The idea is to flatten the tree. This is done when *find()* is called, by making the found root as the parent of the vertex. This way we don't have to traverse all the intermediate nodes.

Union by Rank: Here we assign a rank value to the tree. This rank can be the size of the tree. The idea is to always attach the smaller depth tree to the root of the deeper tree.

Adjacency Lists Diagram

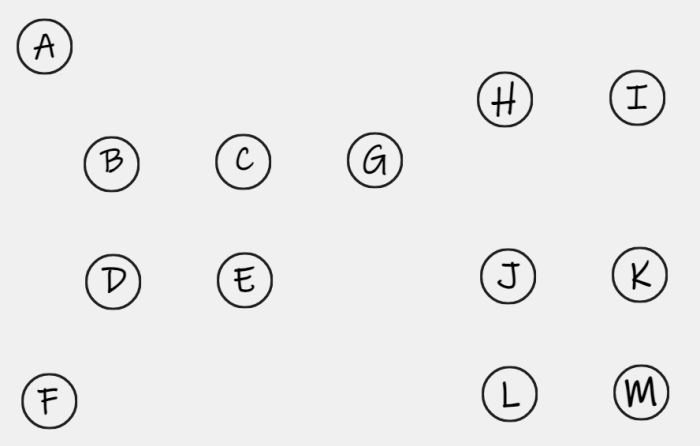


MST using Prim's Algorithm

	0	A	B	C	D	E	F	G	H	I	J	K	L	M
parent[]		~~0~~ F	~~0~~ ~~D~~ A	~~0~~ B	~~0~~ F	~~0~~ ~~L~~ ~~F~~ G	~~0~~ L	~~0~~ ~~L~~ J	~~0~~ ~~G~~ I	~~0~~ K	~~0~~ ~~L~~ M	~~0~~ J	0	~~0~~ L
distance[]		~~∞~~ 2	~~∞~~ ~~2~~ 1	~~∞~~ 1	~~∞~~ 1	~~∞~~ ~~4~~ ~~2~~ 1	~~∞~~ 2	~~∞~~ ~~5~~ 1	~~∞~~ ~~3~~ 2	~~∞~~ 1	~~∞~~ ~~3~~ 2	~~∞~~ 1	0	~~∞~~ 1

Heap	L	M1 F2 J3 E4 G5	→	M	F2 J2 E4 G5	→	F	D1 A2 J2 E2 G5	→	D	A2 B2 J2 E2 G5	→	A	B1 J2 E2 G5
→	B	C1 J2 E2 G5	→	C	J2 E2 G5	→	J	K1 G1 E2	→	K	G1 I1 E2	→	G	I1 E1 H3
→	I	E1 H2	→	E	H2									

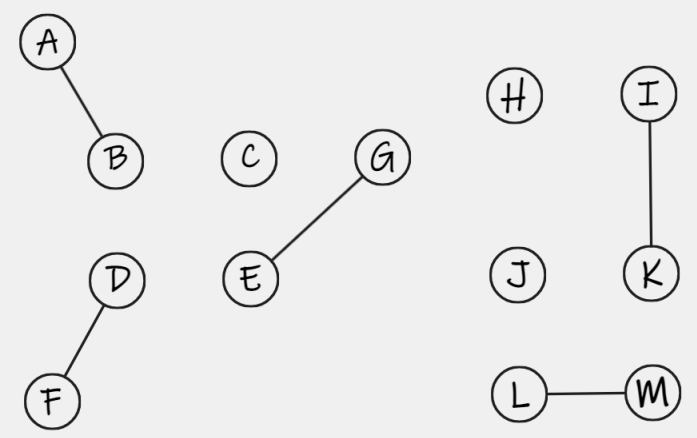
MST using Kruskal's Algorithm



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

Adding following edges to the MST:

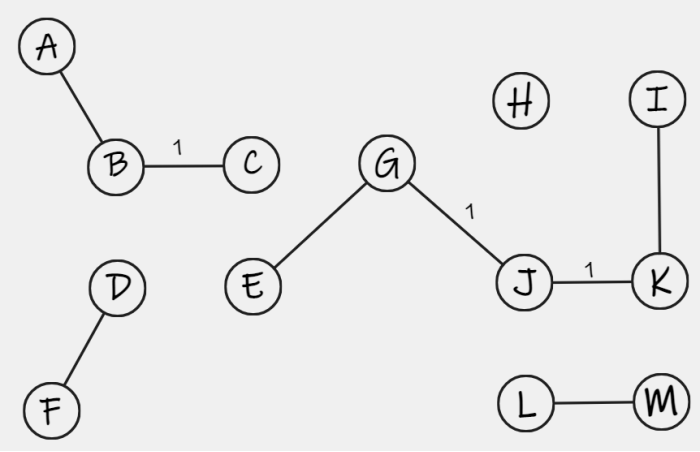
A-1-B
D-1-F
E-1-G
I-1-K
L-1-M



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

Adding following edges to the MST:

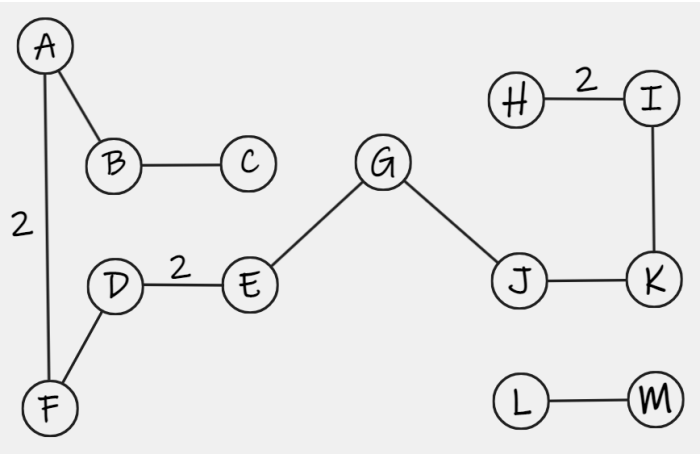
B-1-C
G-1-J
J-1-K



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

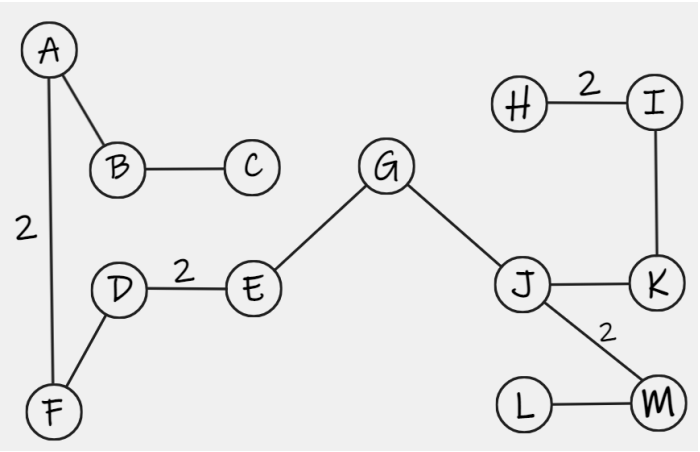
Adding following edges to the MST:

A-2-F
D-2-E
H-2-I



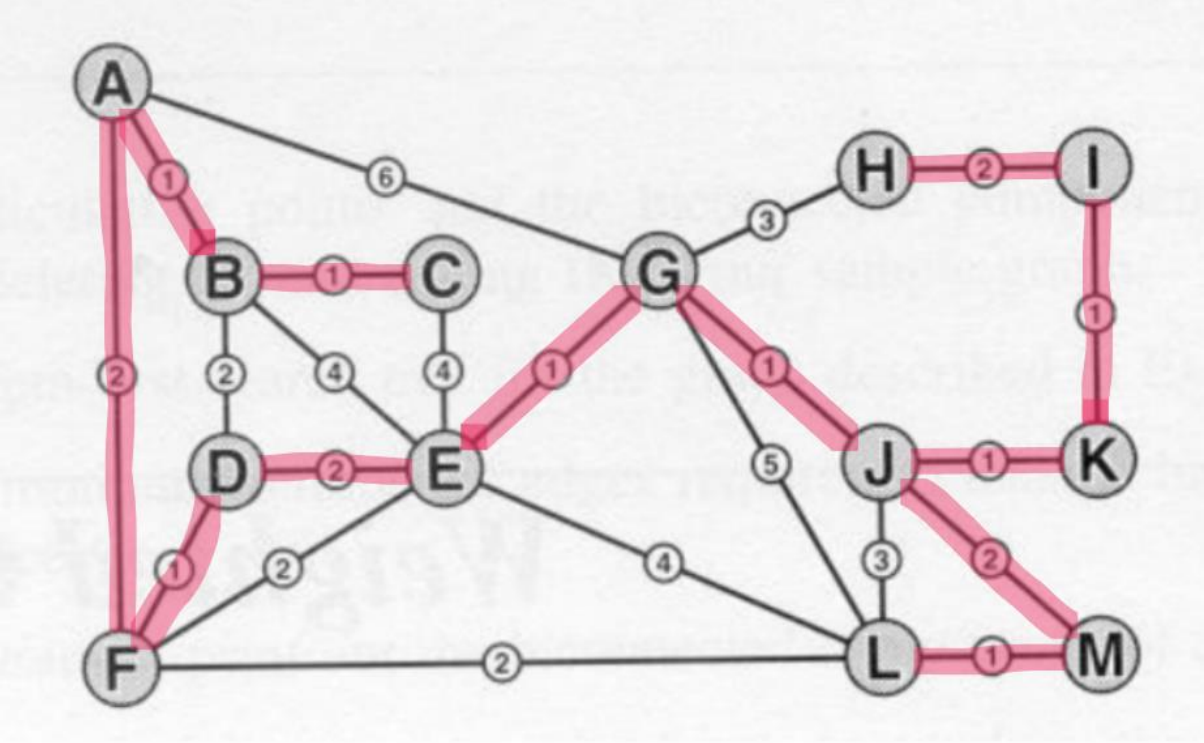
Sets: {A, B, C, D, E, F, G, H, I, J, K} {L, M}

Adding the last edge J-2-M:



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

MST Superimposed on the Graph



Screen Captures of Program Executing

Graph Traversals and Prim's Algorithm

```
rimLists.java } ; if ($?) { java PrimLists }


Graph traversal and Prim's Algorithm


Enter .txt filename: wGraph1.txt

Enter root 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


Displaying adjacency list:

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

```
Displaying 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| ->

Depth First Graph Traversal
Starting with Vertex L

  DF just visited vertex L along @--L
  DF just visited vertex M along L--M
  DF just visited vertex J along M--J
  DF just visited vertex K along J--K
  DF just visited vertex I along K--I
  DF just visited vertex H along I--H
  DF just visited vertex G along H--G
  DF just visited vertex E along G--E
  DF just visited vertex F along E--F
  DF just visited vertex D along F--D
  DF just visited vertex B along D--B
  DF just visited vertex C along B--C
  DF just visited vertex A along B--A


Breadth First Graph Traversal
Starting with Vertex L

  BF just visited vertex L
  BF just visited vertex M
  BF just visited vertex J
  BF just visited vertex G
  BF just visited vertex F
  BF just visited vertex E
  BF just visited vertex K
  BF just visited vertex H
```

```
  BF just visited vertex K
  BF just visited vertex H
  BF just visited vertex A
  BF just visited vertex D
  BF just visited vertex C
  BF just visited vertex B
  BF just visited vertex I

Prim's Algorithm:

Adding to MST: Edge @--(0)--L
Adding to MST: Edge L--(1)--M
Adding to MST: Edge L--(2)--F
Adding to MST: Edge F--(1)--D
Adding to MST: Edge F--(2)--A
Adding to MST: Edge A--(1)--B
Adding to MST: Edge B--(1)--C
Adding to MST: Edge M--(2)--J
Adding to MST: Edge J--(1)--K
Adding to MST: Edge J--(1)--G
Adding to MST: Edge K--(1)--I
Adding to MST: Edge G--(1)--E
Adding to MST: Edge I--(2)--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
```

Kruskal's Algorithm

```
ruskal.java } ; if ($?) { java Kruskal }


Graph traversal and Prim's Algorithm


Enter .txt filename: wGraph1.txt
Parts[] = 13 22
Reading edges from text file
Edge A--(1)--B
Edge A--(2)--F
Edge A--(6)--G
Edge B--(1)--C
Edge B--(2)--D
Edge B--(4)--E
Edge C--(4)--E
Edge D--(2)--E
Edge D--(1)--F
Edge E--(2)--F
Edge E--(1)--G
Edge E--(4)--L
Edge F--(2)--L
Edge G--(3)--H
Edge G--(1)--J
Edge G--(5)--L
Edge H--(2)--I
Edge I--(1)--K
Edge J--(1)--K
Edge J--(3)--L
Edge J--(2)--M
Edge L--(1)--M

Sets before Kruskal's:
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 }

Inserting edge to MST: Edge A--1--B
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 }
Tree of vertices:
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

Inserting edge to MST: Edge B--1--C
Set{A B C }  Set{D }  Set{E }  Set{F }  Set{G }  Set{H }  Set{I }  Set{J }  Set{K }  Set{L }  Set{M }
Tree of vertices:
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

Inserting edge to MST: Edge D--1--F
Set{A B C }  Set{D F }  Set{E }  Set{G }  Set{H }  Set{I }  Set{J }  Set{K }  Set{L }  Set{M }
Tree of vertices:
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

Inserting edge to MST: Edge I--1--K
Set{A B C }  Set{D F }  Set{E }  Set{G }  Set{H }  Set{I K }  Set{J }  Set{L }  Set{M }
Tree of vertices:
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

Inserting edge to MST: Edge J--1--K
Set{A B C }  Set{D F }  Set{E }  Set{G }  Set{H }  Set{I J K }  Set{L }  Set{M }
Tree of vertices:
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

Inserting edge to MST: Edge E--1--G
Set{A B C }  Set{D F }  Set{E G }  Set{H }  Set{I J K }  Set{L }  Set{M }
Tree of vertices:
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

Inserting edge to MST: Edge L--1--M
Set{A B C }  Set{D F }  Set{E G }  Set{H }  Set{I J K }  Set{L M }
Tree of vertices:
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

Inserting edge to MST: Edge G--1--J
Set{A B C }  Set{D F }  Set{E G I J K }  Set{H }  Set{L M }
Tree of vertices:
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
```

```
Inserting edge to MST: Edge G--1--J
Set{A B C }  Set{D F }  Set{E G I J K }  Set{H }  Set{L M }
Tree of vertices:
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

Inserting edge to MST: Edge D--2--E
Set{A B C }  Set{D E F G I J K }  Set{H }  Set{L M }
Tree of vertices:
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

Inserting edge to MST: Edge H--2--I
Set{A B C }  Set{D E F G H I J K }  Set{L M }
Tree of vertices:
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

Inserting edge to MST: Edge J--2--M
Set{A B C }  Set{D E F G H I J K L M }
Tree of vertices:
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

Inserting edge to MST: Edge A--2--F
Set{A B C D E F G H I J K L M }
Tree of vertices:
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

Sets after Kruskal's:
Set{A B C D E F G H I J K L M }

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

Weight of MST = 16
```

Analysis / Reflection

I learned Depth First and Breadth First Traversals. I can also make the Minimum Spanning Tree of a weighted graph. I learned that MST are useful in real life situations such as wiring.

Doing this project also showed me that the basic data structures like stack, queue and heap can be used in numerous different places for a completely new algorithm. These basic structures help in building complex algorithms later.

I also learned that there are different ways you can store a graph, like adjacency lists and matrix (arrays). It can be as simple as a text file with the graph edges.

While I was doing the **Union by Rank** and **Path Compression**, I learned how to make my program efficient by reducing the tree height.