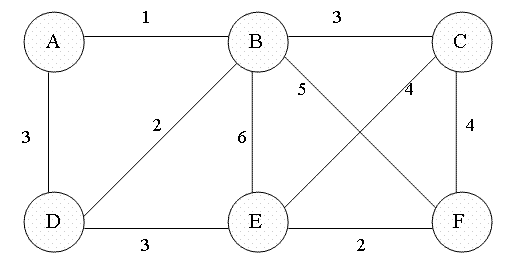
## Program Functionality

The programs order of execution

1. Prompt the user for the name of a text file.
2. Prompt the user for a starting vertex.
3. Enter the name of the text file.
4. Enter the vertex as a number.
5. Program will read the text file.
6. Program will construct Graph Data Structure.
7. Program will call method to Compute the graph’s Shortest Path & Minimal Spanning Tree.
   1. Program will print workings of Dijkstra’s Shortest Path Tree.
   2. Program will print workings of Prim’s Minimum Spanning Tree.
   3. Program will print workings of Kruskal’s Minimum Spanning Tree.
8. Program will output Dijkstra’s Shortest Path Tree.
9. Program will output Prim’s Minimum Spanning Tree.
10. Program will output Kruskal’s Minimum Spanning Tree.
11. Program will call method to Compute the Graph’s Depth First Traversal using Recursion.
    1. Program will output result of Graph’s Depth First Traversal using Recursion.
12. Program will call method to Compute the Graph’s Breadth First Traversal using a queue.
    1. Program will output result of Graph’s Depth First Traversal using Recursion.
13. Program will End.

## Algorithms to be implemented.

### Prim’s Algo Minimal Spanning Tree.



PrimMST():

# Initialize an empty set to hold the vertices in the minimum spanning tree

mst = empty set

# Select the first vertex to start the tree

startVertex = first vertex in graph

mst.add(startVertex)

# Initialize the set of edges to consider

edges = edges connected to startVertex

# Iterate until all vertices are in the minimum spanning tree

while mst has fewer vertices than graph:

# Find the minimum edge in the set of edges

minEdge, minWeight = findMinEdge(edges)

# Add the vertex to the minimum spanning tree

mst.add(minEdge)

# Add the edges connected to the vertex to the set of edges to consider

for edge in edges connected to minEdge:

if edge is not in mst:

edges.add(edge)

# Remove the minimum edge from the set of edges to consider

edges.remove(minEdge)

# Return the minimum spanning tree as an array

return mst as an array

### Kruskal’s Algo Minimal Spanning Tree.

A picture containing graphical user interface

Description automatically generated

KruskalMST():

A = ∅

For each vertex v ∈ G.V:

MAKE-SET(v)

For each edge (u, v) ∈ G.E ordered by increasing order by weight(u, v):

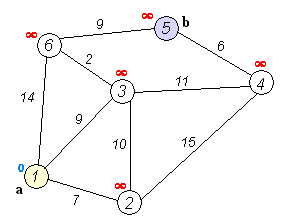
if FIND-SET(u) ≠ FIND-SET(v):

A = A ∪ {(u, v)}

UNION(u, v)

return A

### Dijkstra’s Algo Shortest Path Tree.



1. Create a set of unvisited nodes and initialize all distances to infinity

2. Set the distance of the start node to 0 and add it to a priority queue

3. While the priority queue is not empty:

a. Get the node with the smallest distance from the priority queue

b. For each neighbor of the current node:

i. Calculate the tentative distance from the start node

ii. If the tentative distance is less than the current distance of the neighbor, update its distance

iii. If the neighbor is unvisited, add it to the priority queue

c. Mark the current node as visited

4. Return the distances from the start node to all other nodes

### Depth First Traversal using Recursion.

### Breadth First Traversal using Queue.

## Implementation

* 4 Algo in 1 Java source code file.
  + Prim/Dijkstra/DF/BF.
* Kruskal’s in 1 java source code file.
* Represent the Graph using an adjacency list data structure.
* Kruskal’s Algo Minimal Spanning Tree uses an array of edges.
* Kruskal’s Algo Minimal Spanning Tree MAY use a Heap.
* Kruskal’s Algo Minimal Spanning Tree Alternatively you can use Quicksort or Heapsort to sort edges instead of using a heap.
* Kruskal’s Algo Minimal Spanning Tree Optimizations:
  + Union by Rank.
  + Path Compression.
* Prim’s Algo Minimal Spanning Tree uses Adjacency lists AND a Priority Queue OR Heap X
* Dijkstra’s Algo Minimal Spanning Tree uses Adjacency lists AND a Priority Queue OR Heap