# **Assignment Report: DFS for Disconnected and Connected Graphs with Prim’s Algorithm.**

Algorithms and Data Structure Assignment Report for 15th May 2020

Name: Salman AlSamiri

Student No: D18124618

Program/Year:DT228/2

Module Code: CMPU 2001

Collaboration with: Alexander Bowes.

“I declare that this assignment report is my own work”

Introduction:

The Assignment focuses on Depth First Search/Traversal for connected and disconnected graphs and will also be using a minimum spanning Tree (MST) to calculate the weight of the graph, which all be implemented using object-oriented programming (OOP) language Java.

The algorithm chosen to calculate the MST weight of the graphs is Prim’s algorithm, the solutions to the problem with MST have been made and worked on since the 1920s.

The algorithm was developed by Vojtěch Jarník a Czech mathematician back in 1930 based on Boruvka’s algorithm, in the 1950s it was found, translated and repurposed by few scientists among them Robert C. Prim an American Computer Scientist who have manged to redevelop around the year 1957.

Prim managed to make the algorithm to be considered a “greedy” algorithm as it works for more than just connected graphs but also for undirected graphs, and until this day his algorithm has been used many even developed by Edsger W. Dijkstra not long after Prim’s worked on it.

The Graphs:

Sedgwick’s Graph

A picture containing necklace

Description automatically generated 13 Vertex and 22 edge.

A close up of a map

Description automatically generatedmyGraph.txt made of 14 Vertex and 23 edge.A close up of text on a white background

Description automatically generated

myDisconnectedGraph.txt made of three graphs 15 Vertex and 15 Edge.

The Algorithm Explanation:

Depth First Traversal/Search ( DF(T/S) ) methods:

For all the Graphs DFS/T methods has been used to translate lines from the text files (myGraph, sGraph and myDisGraph) to nodes to help connect and display it as lines of output that connects to recreate the graph within the files. Connected graphs uses Prim’s MST algorithm to find the shortest path that connects the nodes and add up to the smallest possible weight with the help of using Heap algorithm to minimise the time complexity of the graph from V^2 to V log V, unless the graph is too dense then it would turn it into V^2 log V which is it result in more time consumed.

DFRecursive(int s),dfVisit(int prev, int v):

The Depth First algorithm start at a node and when it moves from it does not go back to the node it has been to unless it ends up with no other following nodes to visit, then it will start by accessing previous node and checking other connected node until it hit a null or in this case the sentinel node, which is a dummy node that helps in decreasing the lines of code written and act as stopping and/or starting point for any node related algorithm that is used.

DFRecursive is merely the method to set up the DF Recurring (dfVisit), first it start by initialising the visited array size and all its elements to zero along with the variable used for checking if the nodes matching with the element position in the array has been accessed or not. The method dfVisit where most of the recurring happen get passed with the starting points a 0 and a vertex, it uses a variable to be set as the current vertex after visiting the first vertex and having it swapped to be set as the previously visited (prev), and the node it uses follow through the adjacent array to keep track on which node has been visited and the weight of the edge in between the vertices. The for loop is used to access the next node to current vertex in the adjacent array and setup the variable for the next current vertex(u), but if this u has been visited it will not be going recurring through the method again, as it keep going through the adjacent array and checks if all the matching positions to the nodes has been set to 1 which indicate that they have been visited, otherwise it will run the if statement until all is visited and it run into the last node in the adjacent list.

A screenshot of a cell phone

Description automatically generatedA picture containing clock

Description automatically generatedA picture containing clock

Description automatically generatedA picture containing clock

Description automatically generatedA picture containing clock

Description automatically generatedA picture containing clock

Description automatically generatedA picture containing clock

Description automatically generatedA picture containing clock

Description automatically generatedA picture containing clock

Description automatically generatedA picture containing clock

Description automatically generatedA picture containing clock

Description automatically generatedA picture containing clock

Description automatically generatedA close up of a device

Description automatically generated

DFIterative(int s):

This method uses stack to visit each of the vertices via the adjacent array, similar to the Recursive DF algorithm the chosen node get passed in to start along with initialising the visited array for this method, the variables for previous/current and current/coming vertices and a node to traverse/search through the adjacent list. After initialising the elements used in the method it starts by pushing the chosen/starting node/vertex into the stack and then after satisfying the while loop to check if the stack is empty or not. The algorithm in the method starts by popping/removing the pushed vertex from the stack, then it output it has been visited and increment its position in visited array to state that this vertex has been visited. The following for loop moves to the vertices adjacent to the popped vertex and from the array push the ones that has not been visited into the stack

First of after choosing starting Vertex to go DFSA close up of text on a white background

Description automatically generatedA picture containing clock

Description automatically generatedA picture containing clock

Description automatically generatedA picture containing clock

Description automatically generatedA picture containing clock

Description automatically generatedA picture containing clock

Description automatically generatedA picture containing clock

Description automatically generatedA picture containing clock

Description automatically generatedA picture containing clock

Description automatically generatedA picture containing clock

Description automatically generatedA picture containing clock

Description automatically generatedA picture containing clock

Description automatically generatedA picture containing clock

Description automatically generatedA picture containing clock

Description automatically generatedA picture containing clock

Description automatically generated

Prim’s MST:

The algorithms prim uses multiple lists each keep track of how the process go, nodes/vertices adjacent is used within the parent array and it insert all the adjacent nodes to it into the min heap to help track and add up the minimum spanning weight, while the distance array job is to hold on the weights/distance between the parent vertex and its adjacent vertices. The heap sift Up and sift Down take care of the adjacent vertices from the parent array as after first parent inserted it’s the adjacent vertices it would be inserted into the heap it will be always removing the top v, but after it moves in the smallest vertex in the heap and finds its adjacent to one of the previously inserted vertex it will sift up the vertex with the smaller weight and set its parent value to the current vertex, and after all vertices is visited then it removes all of the heap nodes and clear off the heap.

A picture containing clock

Description automatically generated The graph ‘’myGraph.txt” will start for example from vertex A and insert B, G and E into the heap, but it will not add up any weight until if it sift up the smallest value and move on the vertex that holds that weight/distance.

A picture containing clock

Description automatically generatedB is now the current vertex and it will insert in all its adjacent vertices that has not been visited yet.A picture containing clock

Description automatically generated G is the smallest vertex in the heap and so it’s the current Vertex now and The value of E that adjacent to G is smaller than A so it will get sift up while the other E sift down and get removed of the heap , but also the value of E’s parent will become G of E.A picture containing clock

Description automatically generatedE is The current vertex and follows on with what previously occurred and it will insert M, C and N.A picture containing clock

Description automatically generatedM is the current Vertex and it sift up the new weight for C and also insert I & D.A picture containing clock

Description automatically generatedI now has inserted L and did not become D’s Parent as the distance between it and D is bigger than between D and M.A picture containing clock

Description automatically generatedAs M-D is smallest in the heap D is now the current Vertex and inserting J and H.A picture containing clock

Description automatically generatedH were the smallest so it got to be the current vertex and it insert F into the Heap.A picture containing clock

Description automatically generatedNow F is the current but both J and C distances in the array are smaller than the ones between them and F so it will not be their parent.A picture containing clock

Description automatically generatedL is the current vertex and the vertices its adjacent to has been visited.Both J and C had similar weight but C were heighr in the heap so it got to be the current vertex.J now is the current vertex all the its adjacent has been visited and so all is left is to output N and K and add up their wieght.N is the current vertex and has no adjacent that need to be visited, so its weight get added up.A picture containing clock

Description automatically generatedThe last vertex K weight is added up to the minimum spanning tree weight and we get total of 39 made of 13 edge (out of 23 edge) and 14 vertex.A picture containing clock

Description automatically generated

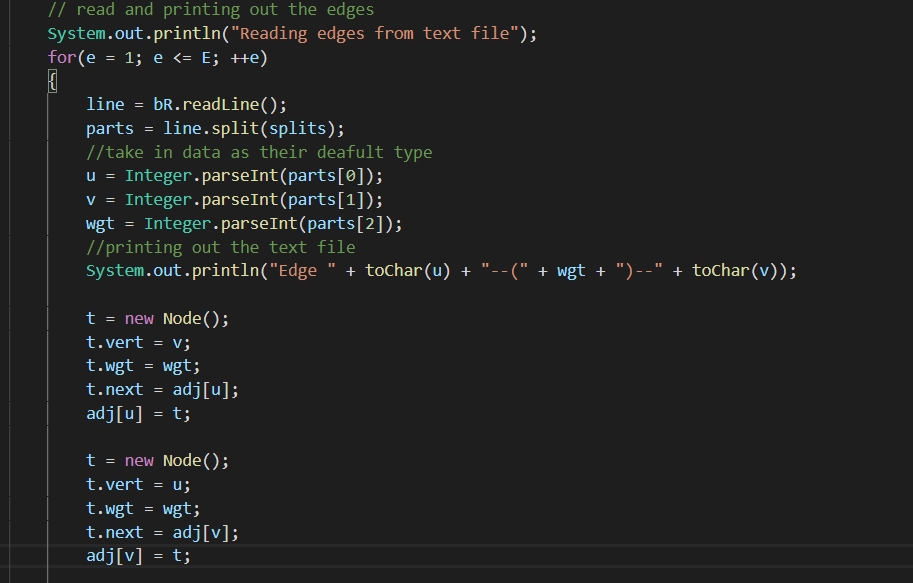
The MST of myGraph.txt after removing the other edges. As it seems the code meets my expectation on how the graph would turn out to be from its original paper sketch.

Section on implementation:

A screenshot of a computer screen

Description automatically generated

dfVisit keeps on implement the passed parameter starting point (0/@) and the chosen vertex for example (1/A), then after the output ‘int u’ is set to be the next node/vertex to the current vertex ‘v’ in the adjacent array, using recursive call helps in go through the graph and the nodes with less number of lines as the code will have each node go to the node next to it and if it was not visited it will go to the following one and keep on that, but if it all ‘visited[V+1]’ array elements is visited then it will be continuing on moving through the nodes until it there is no more and exit into following call in the call stack.



This is an importing part of the code implementation as line will read each line then turn each line into an array and saves it in parts, while it keep on repeating until reading the whole text file the date will be implemented as nodes and saved within them based on their use for the program.

A screenshot of a cell phone

Description automatically generated

The for loop implemented in the DFRecursive method is ensuring that after a graph vertices got visited it goes back and while the counter is less than the total amount of vertices, and based on the amount of times it call the recursive method dfVisit(int prev, int v) it increment ‘int connection’ and print out the amount of connected graphs within the disconnected graph.

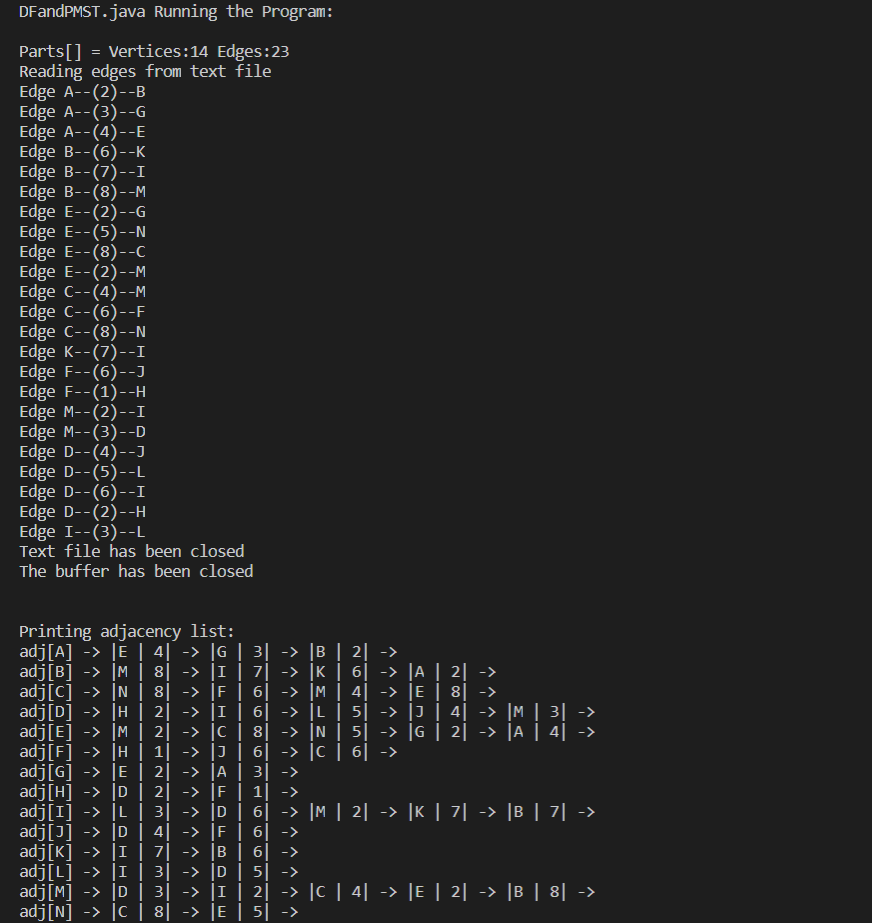
DFandPrim.java Code Output:

A screenshot of a cell phone

Description automatically generatedA picture containing computer

Description automatically generatedA screenshot of a cell phone

Description automatically generated

A close up of text on a black background

Description automatically generatedA screenshot of a cell phone

Description automatically generated

DiscDf.java Code Output:

A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generated

Analysis:

The Prim algorithm in MST gives the idea that a system would still follow plans even if the order seems to be all over and unorganised, and it would help in ordering and organising tasks based on their times and like the MST organise them based on the ones that would be quickest to be done with up to the tasks that consumes a lot of time. DF algorithm seems to be on bar when it comes to order as well as once it gone to a node it does not go back to the previous node until the node it’s on has no adjacent or connected node to it, then it go back and check for adjacent nodes until there is no other node to access to. This seems helpful when it comes to plans or systems that tend to branch out and with DF it will help in having tasks to stack up based on order of use, and it would benefit as well to understand these algorithms more as they seems to be helpful in organising more than just graphs but also daily tasks, plan’s phases and many different kind of data that would tend to branch out and need to be organised from smallest to largest with out having to repeat and cycle through them again.