**Erifeoluwa O. Jamgbadi**

**C18387973**

**Algorithms Design and Data Structures Assignment**

**CMPU2001**

**Year 2**

**Algorithms Design and Data Structures**

This report is my own work

1. introduction/explanation of what you are going to do and a bit about the history of prim’s algorithm.

I created three graphs txt files, two are connected graphs and one is disconnected. These graphs are going to be later used to test the code of my Prims and modification of DF algorithms. I created a DF algorithm using both iteration and recursion. I created the DF file separately with recursion and separately with iteration. Then I put them together into one file. For the second question, I had to modify the Df to be able to count the number of connected components in a disconnected graph. Lastly, I had to implements Prims algorithm for finding the minimum spanning tree for a weighted graph, using heap. In all the program, I used adjacency lists.

***History of prim’s algorithm***

Vijtech Jarnik, a Czech mathematician first discovered prim’s algorithm is 1930, then later it was rediscovered in 1957 by Robert C. Prim. It again rediscovered by Edsger Wybe Dijkstra in 1959. Therefore, prim’s algorithm an also be called DJP Algorithm, Jarnik Algorithm or Prim-Jarnik Algorithm. Prim’s algorithm is a greedy algorithm that finds a minimum spanning tree for a connected graph. This means it finds a subset of the edges that forms a tree, that includes vertex, where the total weight of all the edges in the tree is minimized.

1. diagrams of graphs you constructed yourself. Try to use a drawing tool.

***myGraph***

A bunch of yellow flowers

Description automatically generated

***sGraph***

A picture containing table, bunch, different, yellow

Description automatically generated

***myDiscGraph***

***A picture containing yellow, ball, water, field

Description automatically generated***

1. adjacency lists diagrams showing the graph representation of both your sample graphs.

***myDiscGraph***

A close up of text on a white background

Description automatically generated

***myGraph***

A close up of text on a whiteboard

Description automatically generated

1. a section with detailed explanations showing how all three algorithms that you have been asked to implement will work on the graphs you constructed yourself. For MST, it should include the contents of parent[] and dist[] arrays as well as the heap. Diagrams are important here, even though they will be time consuming to draw.

***Modification of depth first search***

1. In df method, all vertices will be marked as not visited.
2. Then the recursive function is called, which is the method dfVisit. This will make sure that all the vertices in the graph are visited
3. In dfVisit method when it is called, the vertex sent from the df method is marked as visited and is printed out
4. It will then loop through the vertex in the adjacent list and visit all the vertex in the adjacent list
5. There will be an extra for and if loop in the df method, which will make sure that even the nodes, that are not connected will be accounted for. So, if the vertex is not visited, it makes sure that the recursive function is called again, so all vertices will be visited.
6. A count connected nodes variables is declared and this will be put in the dfVisit method, so that the count connected node variable can be incremented by 1, whenever a vertex as been visited along an edge and this is printed out in the df method about how many connected nodes are present.

A picture containing text, whiteboard

Description automatically generated

***Depth First Search Recursion and Iteration Algorithm***

For recursion:

1. In df method, all vertices will be marked as not visited.
2. Then the recursive function is called, which is the method dfVisit. This will make sure that all the vertices are visited
3. In dfVisit method when it is called, the vertex sent from the df method is marked as visited and is printed out
4. It will then loop through the vertex in the adjacent list and visit all the vertex in the adjacent list

A close up of text on a white background

Description automatically generated ***sGraph***

A close up of a map

Description automatically generated ***myGraph***

For iteration:

1. Firstly, all the vertices in the graph will be marked as not visited
2. create a new stack.
3. Push the first node in the stack
4. Loop until the stack is not empty

Pop the top node of the stack s and check whether s is visited or not, if it is not visited, we print the node and mark as visited

Traverse through the entire adjacency list of , if we get a not visited node, we push it into the stack

A picture containing text, whiteboard

Description automatically generated ***sGraph***

A picture containing text

Description automatically generated ***myGraph***

***Prim’s Algorithm***

**For heap:**

First, the heap will need to convert array to a heap and then afterwards, the heap will do the remove and siftdown function. The heap around the node key values or the contents of the heap. These key values are compared against one another.

When I’m inserting a node into the heap, the insert function will add the node at the bottom of the tree and is sifted up to correct position. The sift up function, sifts the vertex up, if the distance of the current vertex is smaller than the distance of the vertex above. It will, then change the position of the vertex, when the current vertex takes the place of the distance of the vertex that is bigger. The vertex that is bigger will than change to the position of the, that the current vertex was in before it moved up. It works hand in hand with the insert function, which when inserting a new element into the heap, it will then call on the siftup to sort out the position, by comparing the distances together.

When the remove function is called, the node is removed and the siftdown is used to sort the heap nodes into the correct positions. The sift down function works hand in hand with the remove function, so when an element is being removed from the heap, siftdown is called from the remove function, so that it will sift the elements in the heap down. It makes checks between the distances, e.g making sure that the heap size is not exceeded and if the distance of the vertex is smaller, than the previous vertex. It will the change the positions accordingly.

**For MST\_Prim:**

1. Starts from vertex c, then select unvisited vertex which is adjacent of the visited vertices with minimum weight.
2. Add the weight of the vertex to the now visited vertex and add that to the weight sum total.
3. Keep repeating, until all vertices are visited.

The parent, hPos and dist is initialized to by V + 1. It loops through the variables for each vertex. For each vertex, it sets the distance to infinite, the array position of the vertex is set to 0 and the parent is set to 0, because we want to start afresh for all the vertex, so that it can find the minimum spanning tree of the graph and fill in the details as it goes along. The while loop checks that the heap is not empty. It puts the vertex in the spanning tree and then it calculates the distance from the previous vertex to the current vertex. It checks to see if the weight of the vertex is bigger than the distance of the two vertices, it will then change the distance of the vertex to the weight of the vertex, the parent of the vertex to the chosen adjacent vertex and adds the weight of the vertices to the value of the total weight before. If the position of the vertex is equal to 0, it inserts the vertex in the heap because it wasn’t there before. If it’s not equal to 0, that means the vertex is already in the heap, so it sifts the position of the vertex up.

***A close up of text on a whiteboard

Description automatically generatedmyGraph, weight of MST=39***

***A close up of text on a whiteboard

Description automatically generatedsGraph, weight of MST= 19***

1. a section on implementation. No need to include whole programs in report, just some relevant part with explanatory text.

***DF***

This is the iteration part. It is where the stack is created. All the vertex is marked as not visited. The vertex is pushed into the stack. While the stack is not empty, it keeps visiting adjacent of each vertex and moving on to the next adjacent of the vertex. If the vertex has no adjacency, the vertex is popped out. This continues until all the vertex are visited. While doing this, it prints out all the vertex, it had visited.

***A screenshot of a social media post

Description automatically generated***

This is the recursive part of the depth first search. The vertex is marked as visited. It iterates through the loop, until the for loop visits every vertex. It returns the previous vertex it visits and the current vertex each time to the depth first function.

***A screenshot of a social media post

Description automatically generated***

This is the df visit function, which calls the recursive function. It sets all the vertices as not visited, so that the recursive function will know, which vertex has been visited or not.

***A screenshot of a social media post

Description automatically generated***

***Modification DF***

This is the df visit function, which calls the recursive function. It sets all the vertices as not visited, so that the recursive function will know, which vertex has been visited or not. I added a new variable called countConnectedNodes, which basically was used to count all the connected nodes in the program and the number of connected nodes it printed in this function.

***A screenshot of a social media post

Description automatically generated***

This part is imported for this algorithm because the variable to count the connected nodes is declared here.

***A screen shot of a computer

Description automatically generated***

This is the recursive part of the depth first search. The vertex is marked as visited. It iterates through the loop, until the for loop visits every vertex. It returns the previous vertex it visits and the current vertex each time to the depth first function. There is a countConnectedNodes variable, that is incremented by 1, everytime a connected vertex is visited. This helps to keep count of how many connected vertex, there are.

***A screenshot of a social media post

Description automatically generated***

***Prim’s Algorithm – MST***

The parent, hPos and dist is initialized to by V + 1. It loops through the variables for each vertex. For each vertex, it sets the distance to infinite, the array position of the vertex is set to 0 and the parent is set to 0, because we want to start afresh for all the vertex, so that it can find the minimum spanning tree of the graph and fill in the details as it goes along. The while loop checks that the heap is not empty. It puts the vertex in the spanning tree and then it calculates the distance from the previous vertex to the current vertex. It checks to see if the weight of the vertex is bigger than the distance of the two vertices, it will then change the distance of the vertex to the weight of the vertex, the parent of the vertex to the chosen adjacent vertex and adds the weight of the vertices to the value of the total weight before. If the position of the vertex is equal to 0, it inserts the vertex in the heap because it wasn’t there before. If it’s not equal to 0, that means the vertex is already in the heap, so it sifts the position of the vertex up.

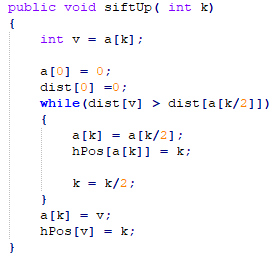
A screenshot of a social media post

Description automatically generated

A screenshot of a cell phone

Description automatically generated

This is the sift up function, it sifts the vertex up, if the distance of the current vertex is smaller than the distance of the vertex above. It will, then change the position of the vertex, when the current vertex takes the place of the distance of the vertex that is bigger. The vertex that is bigger will than change to the position of the, that the current vertex was in before it moved up. It works hand in hand with the insert function, which when inserting a new element into the heap, it will then call on the siftup to sort out the position, by comparing the distances together.



This is the sift down function. It works hand in hand with the remove function, so when an element is being removed from the heap, siftdown is called from the remove function, so that it will sift the elements in the heap down. It makes checks between the distances, e.g making sure that the heap size is not exceeded and if the distance of the vertex is smaller, than the previous vertex. It will the change the positions accordingly.



1. diagrams showing the output of DF and MST superimposed on the graph diagrams and discuss if it corresponds with what you expected from step 4.

***DF and MST – sGraph***

***A picture containing text

Description automatically generated***

***A picture containing apron, curtain, bag

Description automatically generated***

***A close up of text on a whiteboard

Description automatically generated***

No, I would the diagram did not correspond with what I expected from step 4. The thing that didn’t correspond with step 4 was the weight of the MST. For the weight of the MST, I expected 19, but the output gave me 22. The other thing that didn’t correspond is that, vertex A is connected to vertex F as well and the line/edge between C and E is not meant to be there. Vertex E is no meant to connect to vertex L. it’s meant to be from vertex E to G to H to I to K to L and vertex M for the minimum spanning tree.

***DF and MST – myGraph***

A close up of text on a white background

Description automatically generated

A picture containing text, map

Description automatically generated

***A close up of text on a whiteboard

Description automatically generated***

I would say that majority of the diagram corresponds with what I expected from step 4 because vertex E was visited by vertex F not vertex G. Then vertex E connects to vertex G with an edge distance of 2. Other than that, everything else looked the same for the minimum spanning tree. The weight is the same as what I expected.

***Modification DF - myDiscGraph***

***A close up of a whiteboard

Description automatically generated***

No, it is not the same because the output says it’s visited vertex H and I through the edge that connects H to C and I to C, but H and I are not meant to be connected to C. Vertex H and I are meant to be connected together. If it didn’t say that and connected vertex H and I together, then it would correspond to what I expected from step 4.

1. screen captures showing the output of for all implementations for the above graph and your sample graphs.

***DF - myDiscGraph***

***A screenshot of a computer

Description automatically generated***

***MST – sGraph***

***A screenshot of a cell phone

Description automatically generated***

***A picture containing computer, lit, black, keyboard

Description automatically generated***

***A screen shot of a computer

Description automatically generated***

***MST – myGraph***

***A screenshot of a computer

Description automatically generated***

***A screen shot of a computer

Description automatically generated A close up of a logo

Description automatically generated***

1. a discussion/analysis/reflection on what you learned or found useful in the assignment.

I found a lot of interesting things, when doing the assignment. I was able to increase and gain more knowledge about the algorithms e.g. depth first search using recursion and depth first search using iteration, prim’s algorithm. I couldn’t remember recursion as much as iteration from first year, so doing this assignment allowed me to refresh my memory.

I also thought this assignment was very useful to me because I learn materials from school by doing things myself, so I didn’t really understand those algorithm’s, when going through it in class and I was confused on a lot of things, but doing this assignment helped me to understand the things I was confused about in class. I didn’t understand some parts of the heap and recursion kept going over my head(I just was not understanding it), and usually I would have just left it and moved on, but this assignment forced me out of my comfort zone and after multiple, a lot of Youtube videos, I finally understand recursion.

I know I put in a lot of work in this assignment, so seeing this assignment finally finished, is such relief, plus the patience I learnt in this assignment was a lot because I kept leaving the assignment because I would sometimes be frustrated with the code not compiling or when I solved some of the problems, it would lead to more problems. Therefore, I learnt a lot of patience from this assignment.

Another thing that I learnt was algorithm complexity/ BigO. Big O is used to describe the worst-case scenario of an algorithm. I learnt that after writing code, it is good to find out the BigO/complexity of my code because it will tell me how efficient my code and the execution time is. Prims’s algorithm for an adjacency list graph with heap is O(V+ V log2V + Elog2V), which is the equivalent to O(E log2 V) and because we are using a sparse graph, it is O(V^2log2V) [because E is equivalent to V^2].

The modification of the depth first traversal in theory the time complexity is meant to be O(V+E) because it visits every vertex and edge in the graph, but because it goes through every vertex and edge again because it’s a disconnected graph, the time complexity is O(b^d) because the depth first search is run repeatedly, until it finds the nodes that are disconnected. Therefore, the time complexity of both recursion and iteration is

O((b^d) + (V + E)) .

For the depth first traversal, it would be O(V+E) in theory because it is visiting every vertex and edge.

In conclusion, I learnt about time complexity/ BigO, patience, recursion and iteration, more about prim and depth first search algorithm from doing this assignment.