COMP261 Assignment 3 - 300492683

The Auckland Road System program allows the user to search for articulation points/critical intersections of a road graph. The search will show all the connected nodes of the node to find the points of the connected section of the graph. Two buttons are provided to allow the user to also search the disconnected graph to find all the articulation points and to do the MST, Minimum Spanning Tree search. The program has attempted up to challenge, using the iterative version and implemented Kruskal's algorithm, I have attempted only Question's 1 and 2 of the handout.

The articulation search does not consider one way roads and the weight of edges, the articulation points are highlighted in red shown on the graph. I used the iterative version as this was a much safer method as it can expand its stack size without creating a stackOverflow, if I were to use the recursive method. Two helper methods are used to begin the search, one method for the connected search and the other for the disconnected search.

The Minimum Spanning Tree can also be searched on the graph. To find the MST, I used Kruskal's algorithm using a disjoint set with inverted trees. This is the forest of grouped edges, this makes the algorithm work faster as it does not need to search all the sets of sets. The MST search only considers the edge weight and stores the shortest edge if there are multiple of the same. The MSTNode stores two nodes, which is the edge weight and the edge. At the beginning of the search all nodes are added to the fringe. Once the search is complete, the MST tree is displayed with all the highlighted segments inside the tree and the nodes it goes through.

Testing:

To test this program I used a few checks to make sure everything seemed correct. To check the articulation point search is working as intended, I printed the number of nodes found of the search to compare with the number of articulation points provided in the handout. 240 articulation points for the small graph and 10853 for the large graph, which is both correct. Other than this, I also used assert statements within the program. For example, within findMSTTree function to check that there must only be one shortest segment of each neighbour node of a given node. This made sure that the search for articulation points is functioning as expected.

Question 1:

a)

A, 0, 0
B, 1, 0
C, 2, 2
E, 2, 0
D, 3, 0
G, 4, 4
H, 4, 4
I, 5, 4
F, 6, 4
J, 7, 7

b)

- B is an articulation point as the reachBack value of C is greater than the depth of B. So it has no way back onto the graph which means it is disconnected if B is removed.

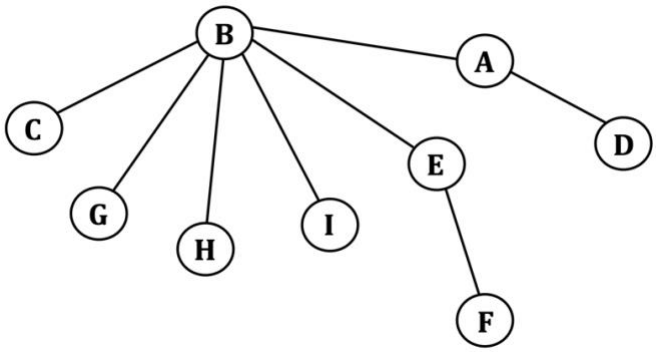
- D is an articulation point as the reachBack value of G is greater than the depth of D. This means that if D was removed then G would be disconnected from the graph, as well as H and all its children will be disconnected as the reachBack of them is not lower than the depth of D.
- F is an articulation point as the reachBack value of the child J is greater than the depth of F. Removing F will disconnect J from the graph as it has no alternative way to connect to the graph.
- H is an Articulation point as its child node cannot reachBack onto the graph, and it's lower than its depth. This means if H was removed, I and all its children will be disconnected. I gets its reachBack off child F which also can't reachBack further than the depth of H.

Question 2:

a)

AD, 2
DE, 6
EF, 3
BE, 4
BG, 1
BC, 2
CH, 4
HI, 6

b)



START PSEUDO CODE
FUNCTION findArticulationPoints
TAKES IN a rootNode
INIT Set of Nodes articulationPoints
CALL searchReset
INIT rootNode to a random node

CAll setHighlightedNodes from graph class passing it articulationPoints
Print out articulationPoints size
END findArticulationPoints

FUNCTION findDisconnected
TAKES IN no Parameters

```
        INIT SET of Nodes articulationPoints
        CALL searchReset
        INIT rootNode to a random node
        INIT boolean complete to false

WHILE not complete
        CALL rootSearch(rootNode, articulationPoints)
        SET complete to true
                FOR each Node in graph
                        IF Node depth is equal to -1
                                SET complete to false
                                SET rootNode to Node

        CAll setHighlightedNodes from graph class passing it articulationPoints
        Print out articulationPoints size
END findDisconnected


FUNCTION rootSearch
        TAKES IN a rootNode
        INIT SET of Nodes articulationPoints
        INIT subTrees to 0
        Set rootNode depth to 0
        FOR each Node in graph
                SET nodes depth to -1
        FOR each Node neighbour of root
        IF depth of neighbour is -1
        CALL articulationSearch passing (neighbour, root, 1, articulationPoints)
                        INCREMENT subTrees
                IF numOfSubTrees is greater than 1
                        ADD rootNode to articulationPoints
END rootSearch


NESTED STATIC CLASS articulationObjects
        FIELDS currNode, parent, depth
END articulationObjects


FUNCTION articulationSearch
        TAKES IN the firstNode to search, the root node it comes from and the firstNode
depth, the articulationPoints Set
        INIT Stack callled searchStack
        PUSH new articulationSearch for firstNode(firstNode, root, firstDepth)

        WHILE stackSearch is NOT empty
                PEEK the top articulationSearch
                        SET currNode depth to depth
                        SET currNode currSearchNode to currNode
                        SET  currParent to parent
                IF currSearch depth is equal to -1
```

SET currNode depth to depth
SET reachBack to currNode
FILL children of currNode without parent

ELSE IF currNode children is NOT empty
SET child to the next Node in currNode children

IF child is visited
SET currNode reachBack to the minimum between current reachBack and child depth
ELSE
PUSH new articulationObjects (child, currNode, currNode depth + 1)

ELSE
IF currNode is NOT the firstNode
SET currNode's parents reachBack to the minimum between the currNode reachBack and its reachBack
IF currNode reachBack is greater than or equal to the parents reachBack
ADD parent to articulationPoints
REMOVE currNode from searchStack
END articulationSearch

Function findMSTTree
INIT Set of Nodes: forest
INIT PriorityQUeue of MSTNode called fringe sorted
CALL unHighlightAll from Graph
FOR each Node in graph
ADD new Set of Nodes with this node to forest
ASSERT if Node neighbours is NOT empty
ASSERT neighbour size to the shortest segments

INIT MSTTree Set of Segment of searchMST passing forest and fringe
FOR each Segment in MSTTree
CALL addHighlightedSegment passing start currSegment
CALL addHighlightedNode passing end segment for both nodes
END findMSTTree

Function searchMST
TAKES IN SET of nodes: forest , PriorityQueue of MSTNode called fringe
INIT Set of Segments called tree

WHILE forest size is greater than one and the fringe is NOT empty
GET MSTNode currMst from top of fringe and removing it
IF node1 and node2 are not in same sets in forest
MERGE these two sets in forest
ADD segment to tree
RETURN tree
END searchMST
END PSEUDO CODE