Reference Manual

Requirements

In order to compile, this application must be a version of Java 1.5 or higher. As much memory should be allocated to the jvm as possible for faster run times.

Algorithms and Data Structures

This project implemented a binary search tree as part of the graph representation. The Strongly-connected-Components algorithm, which also uses Depth-First-Search as well as Depth-First-Visit, was also implemented. All algorithms are based on psuedocode found in Introduction to Algorithms, Third Edition by Cormen et al. HashMaps, HashSets, and ArrayLists were used as part of the java.util library.

Organization of modules

All files required to run the program is the code package. JUnit test files are found in the test package. The directed graph was implemented using a binary search tree (BTree.java) as well as Vertex.java for vertices as well as Arc.java for arcs. An adjacency list was implemented with AList.java. All output needed was computed in Metrics.java. .vna files are parsed with the VNAParser.java file. Driver.java handles user interaction with the graphs. Data.java represents data that can be stored in arcs or vertices. Utils.java, VertexComparable.java, and VertexListComparable.java are helper classes for Metrics.java.

Javadoc

The javadoc for this project can be found in Project 2/doc. To see it, open index.html in a web browser.

Run-time Analysis of Implementation

The graph ADT is implemented with a HashMap, with the keys being the vertices and the values a BinarySearch tree that represents the edge from the key vertex to another vertex. Therefore, any method that processes arcs will of , or (search, delete, and insert are all Any method that accesses an object’s attributes is The following is the analysis of the implementation, sorted by method:

number of vertices

The run time is because it is just returning the size of the HashMap.

number of arcs

The run time is because it is accessing the counter of arcs.

vertices (iterator)

The run time is because it iterates over all of the HashMap keys.

arcs (iterator)

The run time is because the method iterates through all of the vertices, then goes through all of the arcs outgoing from that vertex and in the end it will have iterated through every arc.

getVertex

The run time is because for the worst case it will iterate through all of the keys of the HashMap until it finds it.

getArc

The start vertex will be used to get the binary search tree for that vertex, which is , and the search operation on the tree will be However this search operation will overshadow the (1) and therefore the run time is

getVertexData

The run time is since the Data field is a Vertex attribute.

getArcData

The run time is since the Data field is an Arc attribute.

inDegree

The run time is because in the AList class, it iterates through every vertex and checks to see if the wanted vertex is in the HashSet of the iterated vertex.

outDegree

The run time is because it retrieves the HashSet for the wanted vertex and returns the size of the HashSet.

inAdjacentVertices (iterator)

This method goes through all of the vertices giving a run time of , and it will get a list representation of the tree of arcs. Then it will check each arc in the list to see if their end vertex matches the input vertex, giving a run time of . Therefore the total running time will be .

outAdjacentVertices (iterator)

This method will iterate through all of the HashMap’s keys for that input vertex which is , and then it will be loop through all of the arcs outgoing from that vertex, which is also . Therefore the total running time will be

origin

Since this method is just an attribute of Arc the run time will be .

destination

Since this method is just an attribute of Arc the run time will be .

insertVertex

This method is just inserting a Vertex into the HashMap, which is .

insertArc

This method first finds the appropriate tree by getting the vertex from the HashMap, which is . It then inserts an arc into the BinarySearch Tree, which is. However, the gets overshadowed by the and therefore the run time is .

setVertexData

This method is setting a field of the Vertex, therefore the run time is .

setArcData

This method is setting a field of the Arc, therefore the run time is .

removeVertex

For each BinarySearch tree it searches, which is , if that vertex exists in one of its arcs (search takes ) it then deletes its . Therefore the total running time is .

removeArc

This method iterates through all of the vertices which results in , then the arc is deleted from the tree which is . Therefore the total running time is

reverseDirection

This method reverses the edge in the adjacency which is since a HashMap and a HashSet are implemented which results in . Getting the Arc tree to modify is also . Inserting the tree is and putting that tree back is . Therefore the total running time is .

setAnnotation

This method is modifying an arc or vertex annotation which is an attribute, therefore the run time is .

getAnnotation

This method is retrieving an arc or vertex annotation which is a attribute, therefore the run time is .

removeAnnotation

This method retrieves the annotation from the HashMap, making it .

clearAnnotations

This method clears the HashMap, and therefore the run time is