Unit: COMP1002

Name: Adivishva Mohan

Assessment: Final Assignment

Student ID: 20318330

A screenshot of a computer program

Description automatically generated with medium confidenceUML

Introduction

I have been assigned to create a program that monitors bushfires using unmanned aerial vehicles (UAVs). The design of my program is stated above the UML. All the Abstract Data Types classes such as DSALinkedQueue, DSAStack, DSALinkedList and DSAHashTable are already manually created files originating from my previous practicals such as practical 3, practical 4 and practical 7. These created ADTS are assumed to be included as part of my program therefore it does not appear included in the UML however the location files depend on these ADTs.

Classes and methods

This program contains an overall locationGraph class with 3 inner private classes such as locationGraphVertex, locationGraphEdge and locationDetails. Other classes such as locationFileIO, unitTestLocation, DSALinkedList, DSAStack, DSALinkedQueue and DSAHashTable are part of my programming design to create the graph and identifying locations.

locationGraph (public)

**fields:** vertices (DSALinkedList), edges (DSALinkedList), details (DSALinkedList)

**methods:** locationGraph(), insertVertex, insertDetails, displayDetails, existDetails, getVertexCount, getEdgeCount, getEdge, insertEdge, insertEdge, insertEdge, insertEdge, getAdjacent, getVertex, isAdjacent, listDisplay, printAllVertices, matrixDisplay, breadthFirstSearch, depthFirstSearch, getNextUnvisited

This class is used to create the graph of the locations as well as store information and details of the locations and contains methods that are used to modify the location information and update details. This public class contains three private inner classes such as locationGraphVertex, locationGraphEdge and locationDetails.

locationGraphVertex (private) (inner class of locationGraph class)

**fields:** label (String), visited (boolean), edges (DSALinkedList)

**methods:** locationGraphVertex, iterator, getLabel, getVisited, setVisited, clearVisited, isVisited, setLabel, getAdjacentEdge, getAdjacent, insertEdge, toString

This inner class stores vertex details such as label, value, whether it is visited, its adjacent vertices etc.

locationGraphEdge(private) (inner class of locationGraph class)

**fields:** from (locationGraphVertex), to (locationGraphVertex), value(Object), label (String) directed (boolean)

**methods:** locationGraphEdge, isDirected, getLabel, getValue, getFrom, getTo, setLabel, setValue, toString

This inner class stores attributes and information related to edges and edge weights such as inserting edges, finding whether edges between vertices exist or not, deleting edges etc.

locationDetails(private) (inner class of locationGraph class)

**fields:** temperature (double), windspeed (double), humidity (double), label(String)

**methods:** locationDetails, iterator, getTemperature, setTemperature, getWindSpeed, setWindSpeed, getHumidity, setHumidity, getLabel, setLabel

This inner class stores details of location like the temperature, windspeed, humidity and the name of the location which is the vertex label.

locationFileIO(public)

fields: **none**

**methods**: readLocations, readDetails

This class contains methods designed to read the file and add the locations and information depending the data on the file onto the graph or map.

unitTestLocation(public)

fields: none

methods: main()

This class in a separate used for testing the location classes and files and to test the data to validate the functionality of the program and to draw and print map here.

DSALinkedList (public)

**Fields:** head (DSAListNode), newNd (DSAListNode), currNode(DSAListNode), nodeValue(Object), prevNd (DSAListNode), tail(DSAListNode), count (int)

**Methods:** DSALinkedList, iterator, getCount, insertFirst, insertLast, isEmpty, peekFirst, peekLast, removeFirst, removeLast

This class is a linked list designed to assist in the operations relating to manipulating the location data etc.

DSALinkedListIterator (inner private) (inner class of DSALinkedList)

Field: iterNext (DSAListNode)

Methods: DSALinkedListIterator, hasNext, next, remove

This inner class is an iterating class which allows the iteration of linked list for effective traversal across linked list etc.

DSAListNode (public)

**Fields:** value (Object), next (DSAListNode), prev (DSAListNode)

**Methods:** DSAListNode, getValue, setValue, getNext, setNext, getPrev, setPrev

ListNode class contains all the values and attributes of each list node which is connected to linkedList class and this class is public.

DSALinkedQueue (public)

**Fields:** lists (DSALinkedList), DEFAULT\_CAPACITY (int), capacity(int)

**Methods:** DSALinkedQueue, DSALinkedQueue, iterator, getCount, isEmpty, isFull, enqueue, dequeue, peek, displayQueue

Linked Queue is a queue class that involves in pushing data to the queue from last and popping them at the front. Uses linked List to perform, enqueue, dequeue, peek of the queue. This linked queue is a type of ADT used for assisting in the location class files.

DSAStack (public)

**Fields:** DEFAULT\_CAPACITY(int), capacity (int), list(DSALinkedList)

**Methods:** DSAStack, iterator, DSAStack, getCount, isFull, push, pop, top, printStack

Stack is used for operating the breadthFirstSearch and DepthFirstSearch of the location graph and contains methods in manipulating what goes into the stack and what goes out. This is Stack is type of ADT.

DSAHashTable (public)

**Fields:** hashArray (DSAHashEntry[]), removeEntry (DSAHashEntry), count(int), hashPrint(DSAHashEntry)

**Methods:** DSAHashTable, put, get, hash, hasKey, remove, stepHash, printHashTable

This class contains all the operations to store data into the hash table and manipulate data by hashing it into the table with value and key and removing them by removing entry.

DSAHashEntry (private inner class) (inner class of hash table)

Fields: key (String), value (Object), state(int)

Methods: DSAHashEntry, DSAHashEntry, getKey, getValue, getState, setKey, setValue, setState

This private contains information about entries of the DSAHashTable, it contains key, value, state and operations manipulating the entries of hash table.

**Testing**

|  |  |  |  |
| --- | --- | --- | --- |
| Method to Test | Data tested | Expected Result | Actual Result |
| readLocation(String file) | (1) file = “loc”  (2) file = “”  (3) file = location.al | (1)“File does not exist”  (2)”File does not exist”  (3) readLocations Passed | 1. File does not exist 2. File does not exist 3. readLocations passed |
| readDetails(String fileName, locationGraph mapFile) | (1) fileName = “U”  (2) fileName = “UAVdata.al” | (1) File does not exist  (2) readDetails Passed | (1) File does not exist  (2) readDetails Passed |
| insertVertex(“A”) – after A is already existence  insertVertex(“L”) | (1) label = “A”  (2) label = “L” | (1) Error A already exists  (2) Pass | (1) Error A already exists  (2) Pass |
| insertEdge(“J”, “L”, 2.8) | (1) l1 = “J” ; l2 = “L” ; value = 2.8 | (1) PASS | (2) PASS |
| getVertexCount | Count == match  Count = 11  Match = getVertexCount | (1) PASSED | (1) PASSED |
| dfsQueue= depthFirstSearch(String label);  dfsQueue.dequeue();  DFS = {"AB", "BC", "CD", "DH", "HF", "FE", "EG", "GJ", "JI", "JL"} | (1)label = “A”  (2)label = “B” | (1) PASS  DFS == dfsQueue  (2) PASS  DFS == dfsQueue | (1) PASS  (2) FAIL  - because my array DFS was NOT rearranged manually to suit what the dfsQueue was printing out. Nothing wrong with my depthFirstSearch method. |
| dfsQueue= depthFirstSearch(String label);  dfsQueue.dequeue();  DFS = {"BA", "AC", "CD", "DH", "HF", "FE", "EG", "GJ", "JI", "JL"}; | (1)label = “B” | (1) PASS  DFS == dfsQueue | (1) PASS  DFS == dfsQueue |
| BFS = { "A","B", "C", "E", "F", "D", "G", "I", "H", "J", "L"}  breadthFirstSearch("A","E");  Object too = bfsQueue.dequeue(); | (1) initialLabel = “A”; destLabel = “E”; | (1) PASS  BFS == bfsQueue | (1) PASS – however does not give shortest path between two vertices. |
| existVertex("B")  existVertex("D")  existVertex("F")  existVertex("J") | (1) label = “B”  (2) label = “D”  (3) label = “F”  (4) label = “J” | (1) PASS  (2) PASS  (3) PASS  (4) PASS | (1) PASS  (2) PASS  (3) PASS  (4) PASS |
| listDisplay() | file = “location.al” | Displays list | Display list |
| matrixDisplay() | file = “location.al” | Displays matrix | Displays matrix |
| displayDetails() | file = “UAVdata.al” | Displays Details | Display Details |
| isAdjacent(String l1,String l2) | String l1 = “J”; String l2 = “L” | PASS | PASS |
| isAdjacent(String l1, String l2) | String l1 = “A”; String l2 = “F” | “Oh no no edge between A and F exists” | Program crashes!!   * Due to null pointer exception |
| isAdjacent(String l1, String l2) | String l1 = “A”; String l2 = “B” | PASS | PASS |
| isAdjacent(String l1, String l2) | String l1 = “C”; String l2 = “G” | PASS | PASS |
| isAdjacent(String l1, String l2)   * Method is edited and algorithms are modified | String l1 = “A”; String l2 = “F” | “Oh no no edge between A and F exists” | “Oh no no edge between A and F exists” |
| isAdjacent(String l1, String l2) | String l1 = “C”; String l2 = null | “There must be two labels and must exist” | “There must be two labels and must exist” |
|  |  |  |  |

Implementation

The design of location reading program includes all the files as stated above however in the locationGraph class file, the breadthFirstSearch is only able to traverse through the entire list rather than find the shortest locations however there is an attempted implementation of the breadthFirstSearch to perform the shortest path between two vertices. Also, my program uses hashTable only in the breadthFirstSearch algorithm of locationGraph class however it is limited and the use of hashTable and heaps for task 4 and 5 could have been used more and efficiently to make my program more versatile and better. Also, the implementation of locationFileIO class file allows the reading and processing of location and UAV data and calls the locationGraph class file to insert vertices, edges and details and lets the locationGraph to create the graph and display. To test the functionality of the locationGraph and locationFileIO and other files mentioned above, the unitTestLocation class file is used to test each function of locationGraph and locationFileIO classes and ensure that run time errors as well as algorithmic errors have been neutralized. ADT classes files like DSALinkedQueue, DSAStack, DSALinkedList, DSAHashTable and DSAListNode are dependent on each other for example the DSALinkedList calls DSAListNode to create and manipulate the nodes of the linked lists and DSALinkedList and DSAListNode are both dependent on each other. The DSALinkedList and DSAListNode are also used by DSALinkedQueue, DSAStack to efficiently manage the size of the data and manipulate the insertion and removal of data. Overall, my program could have been more efficient and more developed however the program is partially incomplete and may not be able to delete vertices however the program can insert vertices and edges, check if the vertices exists or not and run BFS and DFS despite BFS not performing as expected.