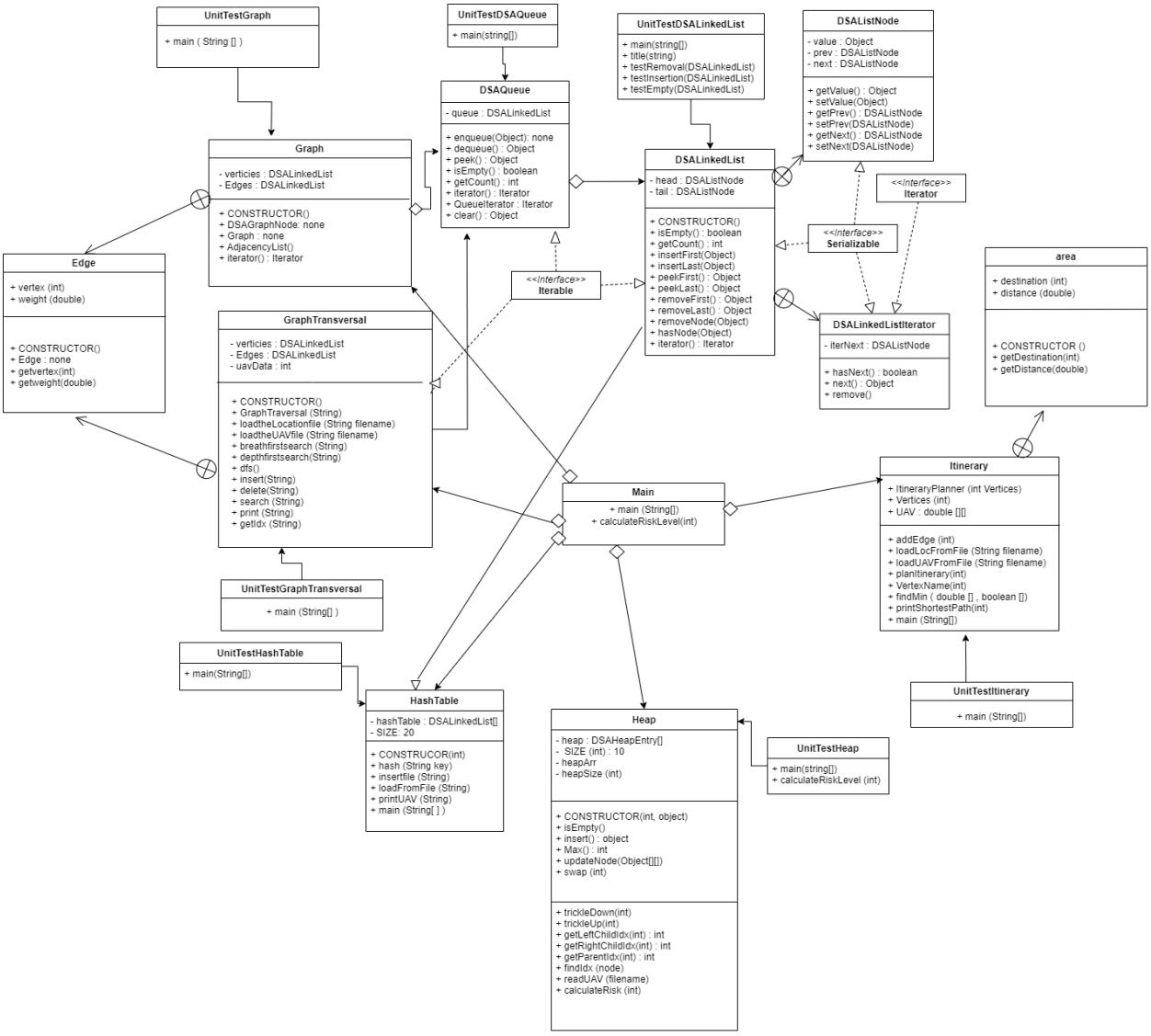
REPORT

Hesara Yasaswin Pathirana

20928386

UML,



Graph.java,

This is the implementation of task 1,

The graph for the location.txt file will be read here.

the graph will be printed as an adjacency list.

- Edge – inner class
- Graph – to read the file given
- AdjacencyList – prints the adjacency list

GraphTraverasal.java,

Here I implemented the GraphTraversal class for both task 2 and task 3 (Implementing BFS and DFS, Insert, Delete, and Searching Operations

- Edge (inner class): This class represents an edge in the graph.
- loadtheLocationfile - this method loads the location.txt, it reads the file and creates an adjacency list.
- loadtheUAVfile - this method loads the UAVdata file and reads the data for each location and stores it in the array representing temperature, humidity, and wind speed values for each location.
- breathfirstsearch - this method performs the Breadth-First Search (BFS) traversal on the graph to find the shortest path between two locations. The starting and ending positions are entered at the beginning of this algorithm. In order to discover the shortest path, it uses a queue to methodically explore the graph in a breadth-first fashion, keeping track of the vertices that have been visited and the parent vertex for each vertices. It uses the parent array to retrace its steps from the end to the beginning once it arrives at the destination, then it to reconstruct the shortest path. The quickest path is then displayed by printing it out.
- depthfirstsearch - this method performs a Depth-First Search (DFS) traversal on the graph starting from a specified location. This method focuses depth-first method and uses a recursive method to probe into the graph from a given start point. As it moves forward, a record of the vertices visited is recorded. technique produces a depth-first search traversal of the graph by returning the vertices in the order they are encountered.
- Insert(): inserts a location according to user's prompt.
- Delete(): deletes the location according to user's prompt.
- Search(): searches for the location according to user's prompt.
- print - prints the UAV data for a specific location.

HashTable.java,

Here I implemented the HashTable class for task 4,

- HashTable() – this is the constructor of the class. The table size is set to a value to handle collisions each element of the array is initialized as a new DSALinkedList.
- insertfile - this method inserts UAV data to the hash table. The hash () method uses the specified location as input to generate a distinct value known as the hash value. Then a string representation of this value is created. the calculated hash value's corresponding index for this string representation is added to the hashTable array.
- loadFromFile - This method loads UAV data from a file and inserts it into the hash table each line is splitted into tokens using a space delimiter which is converted to their appropriate data types.
- hash - It performs by summing the numerical values assigned by the ASCII system to each character in the key. It adds up these numbers, divides the total by a predetermined number, SIZE, and gives us the leftover amount which we refer to as the hash value.
- printUAV - this method retrieves and prints the UAV data for a given location. If the match is found it will print the required data if not an error message will be displayed saying " location not found"

Heap.java,

Here I implemented the Heap class for task 5,

Node – inner class

- isEmpty() - checks if the heap is empty by verifying the heapsize is 0.
- insert - it inserts a new node to the heap but if the size if full it throws an exception if not it adds the node to the heaparray and increments the heapsize by 1.
- Max () - first it checks if the heap is empty by the isEmpty() method , then if it is not empty the method will proceeds to store the node with the maximum risk level.it then swaps the root node of the heap with the last node in the heap, then the size of the heapsize is decremented by 1 then it performs the trickle up method and compare them.
- updateNode - this method updates a node in the heap with a new node. If the new node has a higher risk level than the old node it performs the trickleUp or trickleDown methods.
- swap - I created a swap method instead of always creating the swapping between indices in the heaparray.
- trickleUp - it updates the node with the higher risk level if the current node is higher. if the node has a higher risk than its parent node it switches through a sequence of swaps and it continues to do so till it finds it position.
- trickleDown - it updates the node with the lower risk level.
- getParentIdx - calculates and returns the index of the parent node.
- getLeftChildIdx - calculates and returns the index of the left child node.

- getRightChildIdx - calculates and returns the index of the right child node.
- findIdx - compares and finds the specific node in the heap.
- readUAV - reads the data from the UAVdata.txt file and it splits the line with the data types.
- calculateRisk - method calculates the risk level based on the given temperature, humidity, and wind speed values. If high = 3, medium =2 , low = 1.

Itinenary.java,

Here I implemented the Itinerary class for task 6,

area – inner class

- Itinerary () - private class area - inner class to represent 2 fields the destination and the distance.
- addEdge - adds an edge to the graph.it then adds the edge to the source vertex and constructs a edge object with the destination vertex and distance.
- planItinerary - implements Dijkstra's algorithm to find the shortest way from the start vertex to any other vertices from the loaction.txt
- MinimumDistance- finds the shorted path from the start vertex.
- print - prints the shortest path from the start vertex to a given destination vertex. it takes the destination vertex and an array of previous vertices from the plan Itinerary method here the Dijkstra's algorithm as implemented.
- Shortestpath – the shortest path to travel
- LoadFile - loads the locations and the distances from the files and build the graph.

Testing methodology and results.

Feature / Function	Requirements	Design	Test
Graph functions			
the graph from the given file should be read	the graph is read	Graph() in Graph.java class and tested in a main() method in UnitTestGraph.java	[SUCCESSFULLY PASSED] the graph from the given is read.
display the adjacency list of the graph from the given file	adjacency list should be displayed if a graph is given	AdjacencyList() in Graph.java class and tested in a main() method in UnitTestGraph.java	[SUCCESSFULLY PASSED] the adjacency list is displayed
loads the location.txt and UAVdata.txt	the files should be provided in the same directory.	loadLocation() and loadUAV() in the GraphTraversal.java class and tested in a main() method in UnitTestGraphTraversal.java	[SUCCESSFULLY PASSED] the files are loaded successfully.
breathfirstsearch is performed	the files should be provied in the same directory. the bfs method is done	breathfirstsearch() in GraphTraversal.java and tested in a main() method in UnitTestGraphTraversal.java	[SUCCESSFULLY PASSED] the bfs is method is done as expected.
depthfirstsearch is performed	the files should be provied in the same directory. the dfs method is done	depthfirstsearch() in GraphTraversal.java and tested in a main() method in UnitTestGraphTraversal.java	[SUCCESSFULLY PASSED] the dfs is method is done as expected.
insert node	the node should be inserted to the graph by user inputs.	insert() is in GraphTraversal.java and tested in a main() method in UnitTestGraphTraversal.java	[SUCCESSFULLY PASSED] the node is added.
delete node	the node should be deleted according to the user inputs	delete() is in GraphTraversal.java and tested in a main() method in UnitTestGraphTraversal.java	[SUCCESSFULLY PASSED] the node is deleted.
search node	the node should be searched according to the user inputs	search() is in GraphTraversal.java and tested in a main() method in UnitTestGraphTraversal.java	[SUCCESSFULLY PASSED] the node is searched.

Hash Table Functions			
HashTable() is used to handle collisions	the files should be provided in the same directory.	HashTable() in HashTable.java and tested in a main() method in UnitTestHashTable.java	[SUCCESSFULLY PASSED]
load the "UAVdata.txt" and enter it to hash table	the files should be provided in the same directory. the loading method is done	loadFromFile() in HashTable.java and tested in a main() method in UnitTestHashTable.java	[SUCCESSFULLY PASSED]
Heap Functions			
to calculate the risk level	the UAVdata.txt file should be provided	calculateRiskLevel() in Heap.java and tested in a main() method in UnitTestHeap.java	[SUCCESSFULLY PASSED] the risk level is calculated depending on the user input for the temperature, humidity and wind speed.
to find the areas with the highest risk of bushfires	the UAVdata.txt file should be provided.	HighRiskAreas() in Heap.java and tested in a main() method in UnitTestHeap.java	[SUCCESSFULLY PASSED] the risk levels are showed in order of highest to lowest.
inserting the new node to the heap	the user should input locations and UAVdata	insert() in Heap.java and tested inside a main() method in UnitTestHeap.java but not for inserting a new node itself	[SUCCESSFULLY PASSED]

Max heap	the user should input locations and UAVdata	Max() in Heap.java and tested inside a main() method in UnitTestHeap.java	[SUCCESSFULLY PASSED]
Itinerary Functions			
plan Itinerary	to find the shortest way from the start vertex to any other vertices from the loaction.txt	planItinerary() in ItineraryPlanner.java and tested in a main() method in UnitTestItineraryPlanner.java	[SUCCESSFULLY PASSED] the shortest way is found and displayed.
load location file	the test data files should be in the same directory.	loadLocationsFromFile() in ItineraryPlanner.java and tested in a main() method in UnitTestItineraryPlanner.java	[SUCCESSFULLY PASSED] - the itinerary from any locations will be shown.