5/28/2023

Final Assignment Report

Data Structure and Algorithms COMP1002 Semester 1 2023

Lina Yeo

700043539(MIRI STUDENT ID) / 21204647(PERTH STUDENT ID)

Table of Contents

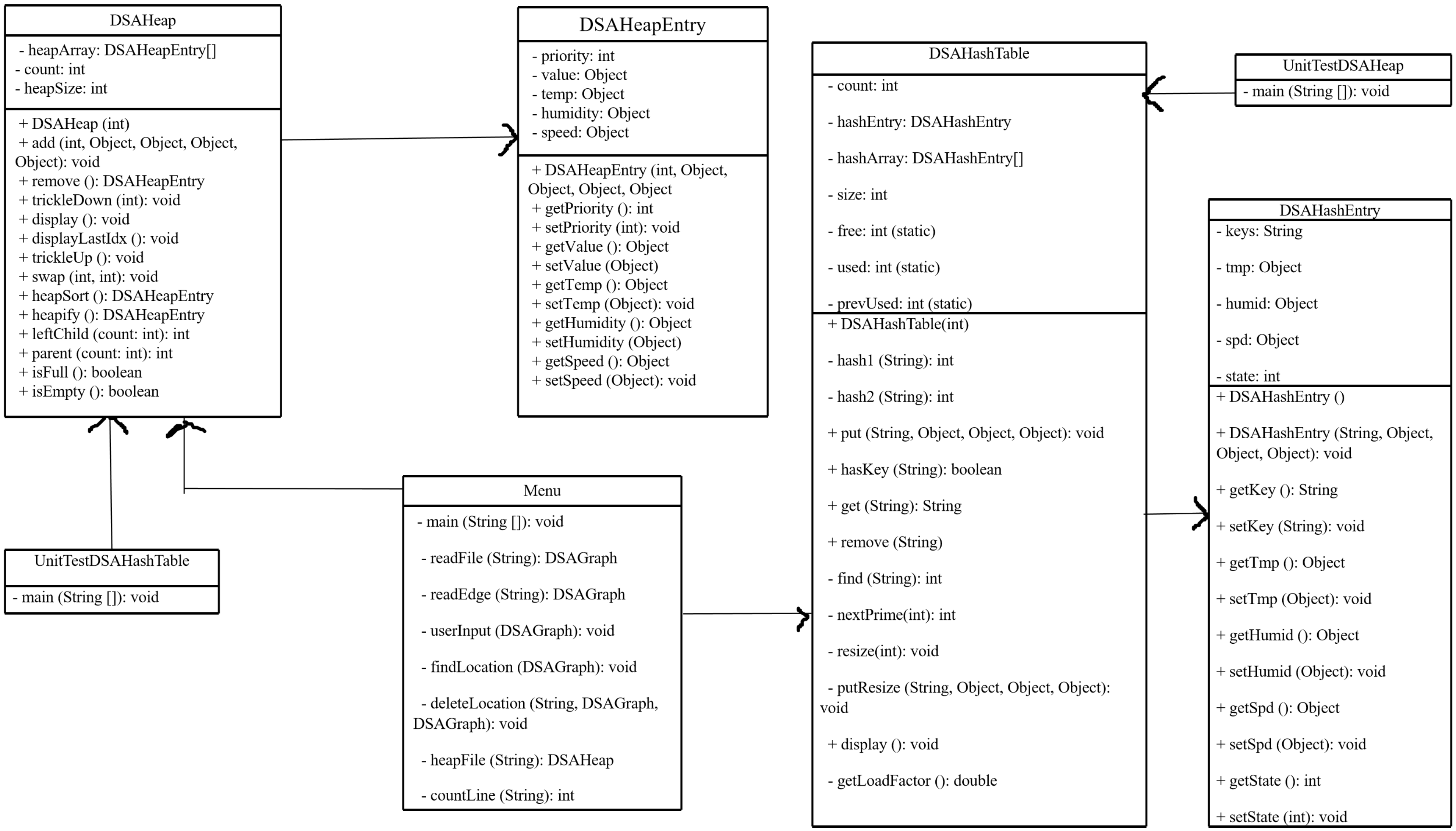
Design of Program

The program is designed by using of doubly linked list, graph, heap, and hash table algorithms to process the location and the location data collected by the UAVs. Data of the location collected by the UAVs will be added into the graph. The location will be the vertex in the graph. The temperature, humidity and wind speed will be the values of the vertex.

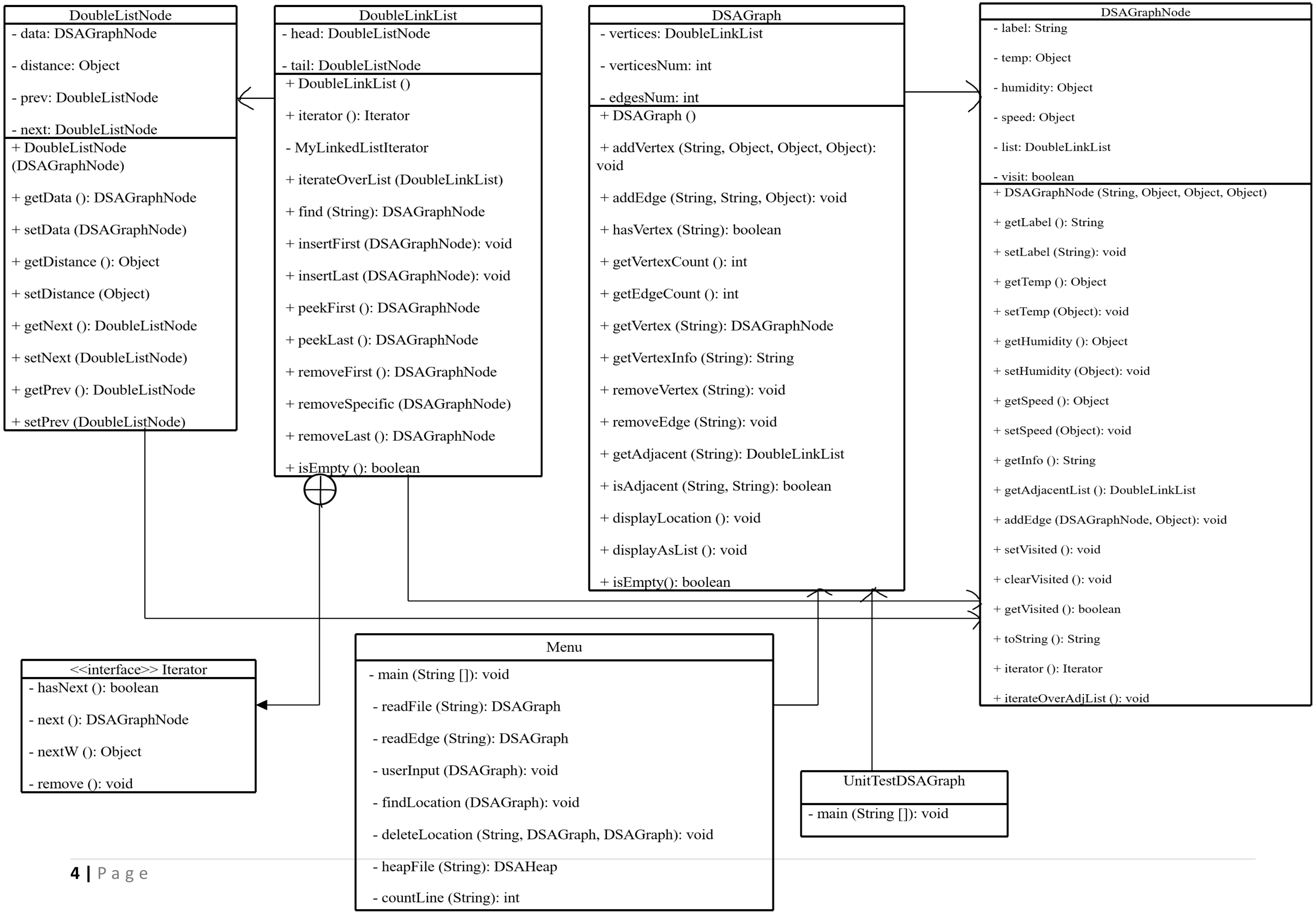
The doubly linked list is implemented in the program as the graph is approaches by using adjacency list. The location that are adjacent to each other will be connected by implementing the doubly linked list. The distance between the location will be the weight carried by the edge.

Hash Table is implemented to efficiently store and retrieve the data gathered by the UAVs. The location will be set as the key at the hash table to get the index to insert the data collected by the UAVs.

Heap is implemented in the program to process the data gathered by the UAVs. The data will be sorted based on the risk of bushfires and need attention. The priority of the areas will be determined by the temperature, humidity and the wind speed of the area.


DSAHeap
- heapArray: DSAHeapEntry[]
- count: int
- heapSize: int
+ DSAHeap (int)
+ add (int, Object, Object, Object, Object): void
+ remove (): DSAHeapEntry
+ trickleDown (int): void
+ display (): void
+ displayLastIdx (): void
+ trickleUp (): void
+ swap (int, int): void
+ heapSort (): DSAHeapEntry
+ heapify (): DSAHeapEntry
+ leftChild (count: int): int
+ parent (count: int): int
+ isFull (): boolean
+ isEmpty (): boolean
DSAHeapEntry
- priority: int
- value: Object
- temp: Object
- humidity: Object
- speed: Object
+ DSAHeapEntry (int, Object, Object, Object, Object
+ getPriority (): int
+ setPriority (int): void
+ getValue (): Object
+ setValue (Object)
+ getTemp (): Object
+ setTemp (Object): void
+ getHumidity (): Object
+ setHumidity (Object)
+ getSpeed (): Object
+ setSpeed (Object): void
DSAHashTable
- count: int
- hashEntry: DSAHashEntry
- hashArray: DSAHashEntry[]
- size: int
- free: int (static)
- used: int (static)
- prevUsed: int (static)
+ DSAHashTable(int)
- hash1 (String): int
- hash2 (String): int
+ put (String, Object, Object, Object): void
+ hasKey (String): boolean
+ get (String): String
+ remove (String)
- find (String): int
- nextPrime(int): int
- resize(int): void
- putResize (String, Object, Object, Object): void
+ display (): void
- getLoadFactor (): double
UnitTestDSAHeap
- main (String []): void
DSAHashEntry
- keys: String
- tmp: Object
- humid: Object
- spd: Object
- state: int
+ DSAHashEntry ()
+ DSAHashEntry (String, Object, Object, Object): void
+ getKey (): String
+ setKey (String): void
+ getTmp (): Object
+ setTmp (Object): void
+ getHumid (): Object
+ setHumid (Object): void
+ getSpd (): Object
+ setSpd (Object): void
+ getState (): int
+ setState (int): void
Menu
- main (String []): void
- readFile (String): DSAGraph
- readEdge (String): DSAGraph
- userInput (DSAGraph): void
- findLocation (DSAGraph): void
- deleteLocation (String, DSAGraph, DSAGraph): void
- heapFile (String): DSAHeap
- countLine (String): int
UnitTestDSAHashTable
- main (String []): void

UML Diagram


DoubleListNode
- data: DSAGraphNode
- distance: Object
- prev: DoubleListNode
- next: DoubleListNode
+ DoubleListNode (DSAGraphNode)
+ getData (): DSAGraphNode
+ setData (DSAGraphNode)
+ getDistance (): Object
+ setDistance (Object)
+ getNext (): DoubleListNode
+ setNext (DoubleListNode)
+ getPrev (): DoubleListNode
+ setPrev (DoubleListNode)
DoubleLinkList
- head: DoubleListNode
- tail: DoubleListNode
+ DoubleLinkList ()
+ iterator (): Iterator
- MyLinkedListIterator
+ iterateOverList (DoubleLinkList)
+ find (String): DSAGraphNode
+ insertFirst (DSAGraphNode): void
+ insertLast (DSAGraphNode): void
+ peekFirst (): DSAGraphNode
+ peekLast (): DSAGraphNode
+ removeFirst (): DSAGraphNode
+ removeSpecific (DSAGraphNode)
+ removeLast (): DSAGraphNode
+ isEmpty (): boolean
DSAGraph
- vertices: DoubleLinkList
- verticesNum: int
- edgesNum: int
+ DSAGraph ()
+ addVertex (String, Object, Object, Object): void
+ addEdge (String, String, Object): void
+ hasVertex (String): boolean
+ getVertexCount (): int
+ getEdgeCount (): int
+ getVertex (String): DSAGraphNode
+ getVertexInfo (String): String
+ removeVertex (String): void
+ removeEdge (String): void
+ getAdjacent (String): DoubleLinkList
+ isAdjacent (String, String): boolean
+ displayLocation (): void
+ displayAsList (): void
+ isEmpty(): boolean
DSAGraphNode
- label: String
- temp: Object
- humidity: Object
- speed: Object
- list: DoubleLinkList
- visit: boolean
+ DSAGraphNode (String, Object, Object, Object)
+ getLabel (): String
+ setLabel (String): void
+ getTemp (): Object
+ setTemp (Object): void
+ getHumidity (): Object
+ setHumidity (Object): void
+ getSpeed (): Object
+ setSpeed (Object): void
+ getInfo (): String
+ getAdjacentList (): DoubleLinkList
+ addEdge (DSAGraphNode, Object): void
+ setVisited (): void
+ clearVisited (): void
+ getVisited (): boolean
+ toString (): String
+ iterator (): Iterator
+ iterateOverAdjList (): void
<<interface>> Iterator
- hasNext (): boolean
- next (): DSAGraphNode
- nextW (): Object
- remove (): void
Menu
- main (String []): void
- readFile (String): DSAGraph
- readEdge (String): DSAGraph
- userInput (DSAGraph): void
- findLocation (DSAGraph): void
- deleteLocation (String, DSAGraph, DSAGraph): void
- heapFile (String): DSAHeap
- countLine (String): int
UnitTestDSAGraph
- main (String []): void

Data Structure and Algorithm used

The data structure and algorithm used in the program are as below:

- Linked List
- Iterators
- Graphs
- Heaps
- Hash Tables

Graphs is implemented in the program because some location is connected to each other. The location will be the vertex of the graph and the data of the location will be the value of the vertex. When the location is connected to other location, the location will then be added as the edge of the other location and vice versa, as the graph implemented for this program is undirected graph. The graph is approaches by using the adjacency list for this program. Adjacency list use linked list, to connect one node to another node. The node in this case is the location or the vertex of the graph. For this program, each node will hold a value which is the distance between each of the location. Iterator is implemented in the linked list to iterate through the list and obtain the location and the location data.

In this program, Hash Table is also implemented to compare the efficiency of using hash table than searching through the list of arrays. The key that will be used to get index of the array for this program will be the location. Each of the key will have the values which are the temperature, humidity, and the wind speed of the location.

Moreover, Heap data structure is implemented in the program to get the location with the high risk of bushfires and need attention, the value used to decide the priority of location is obtained by adding up the risk values for the temperature, humidity, and the wind speed of the location. The risk values are set as below:

Temperature (25 – 48 degrees Celsius)	Humidity (15 – 60%)	Wind Speed (30 – 100 km/h)	Risk Value
25 – 32 degrees Celsius (low)	>50% (low)	<40 km/h (low)	1
33 – 40 degrees Celsius (medium)	31 – 50% (medium)	41 – 55 km/h (medium)	2
>48 degrees Celsius (high)	<30% (high0	>55 km/h (high)	3

Descriptions of Classes

The java class exist in the program are as below:

- Menu
- DoubleLinkList
- DoubleListNode
- DSAGraph
- DSAGraphNode
- DSAHashEntry
- DSAHashTable
- DSAHeap
- DSAHeapEntry
- UnitTestDSAGraph
- UnitTestDSAHeap
- UnitTestDSAHashTable

Menu class consist of the main method which will display the menu selection and received the selection chosen by the user. With the selection obtained, main method will call the method to run the selection. Below is the method inside main class and their function:

Method	Function
readFile	Receive the name of the file, for this program is the UAVdata.txt, then read through the file. Insert the location and the data of the location into the graph.
readEdge	Receive the name of the file, for this program is the location.txt, then read through the file. Insert the edge of the location and the weight(distance) between each location into the graph.
userInput	Ask the user to insert new location, temperature, humidity, and the wind speed in that location. Received the information and insert it into the graph.
findLocationGraph	Ask the user to insert the location they would like to check. Received the input and search through the graph to looks for the

	location information. Then, displayed the location information for the user.
deleteLocation	Ask the user for the location they would like to delete. Received the input and find the location in the graph, then remove the vertex from the graph and the adjacency list.
heapFile	Receive the name of the file, for this program is the UAVdata.txt, then read through the file. Insert the location and the data of the location into the heap table.
findLocationHeap	Ask the user to insert the location they would like to check. Received the input and find the key in the heap table. Then, displayed the location information for the user.
countLine	Take in the file name, for this program is the UAVdata.txt and count the line of the file.

Next, DoubleLinkList class, linked list is applied as the graph is approaches by using adjacency list. Below is the method inside DoubleLinkList class and their function:

Method	Function
iterator	For the private class iterator.
iterateOverList	Iterate through the linked list and print out the linked list.
find	Find and return the graph node in the linked list.
insertFirst	Add the graph node as the first node in the list or the head in the linked list.
InsertLast	Add the graph node as the last node in the list or the tail in the linked list.
peekFirst	Get the first node in the linked list.
peekLast	Get the last node in the linked list
removeFirst	Remove the first node in the linked list.
removeLast	Remove the last node in the linked list.
removeSpecific	Remove specific node in the linked list, so this method will take in the graph node, that the user would like to delete. It will go

	through the list to get the specific node and remove the node from the list.
isEmpty	Check if the linked list is empty.

DSAGraph class in this program will have the location as the vertex and temperature, humidity, and wind speed of the location as the values of the vertex. Below is the method inside DSAGraph class and their function:

Method	Function
addVertex	Adding the vertex and the vertex values to the graph.
addEdge	Adding the edge to the vertex.
hasVertex	Check if the vertex exists in the list.
getVertexCount	Return the vertex count.
getEdgeCount	Return the edge count.
getVertex	Get the vertex from the linked list.
getVertexInfo	Get the vertex values in the graph.
removeVertex	Remove the vertex from the graph and called removeEdge method to remove the vertex from the linked list.
removeEdge	Remove the vertex from the linked list.
getAdjacent	Take in a vertex and return the adjacency list of that vertex.
isAdjacent	Check if the vertex is adjacent to each other.
displayLocation	Iterate through linked list and display the location(vertex) and the location data(value) in the graph.
displayAsList	Iterate through linked list and display the adjacency list of the vertex in the graph.
isEmpty	Check if the linked list is empty.

DSAHashTable class in this program will have the location as the key and temperature, humidity, and wind speed of the location as the values of the key. The key will be used to find the elements which are temperature, humidity, and the wind speed of the location for this program. Below is the method inside DSAHashTable class and their function:

Method	Function
hash1	Convert the key to the index to put in the data in the Hash Table.
hash2	Convert the key to the index to put in the data in the Hash Table when the index get from hash1 is filled already.
put	Called resize, hash1 and hash2 method. Get the index to put in the element into the Hash Table from the hash1 and hash2 method. Resize the table when the size of the table is too big or too small for the data inserted.
hasKey	Check if the key exists in the Hash Table.
get	Receive the key, called find method to get the index of the key, and return the data of the key.
remove	Receive the key, called find method to get the index of the key, and remove the key and the data from the Hash Table.
find	Find the index of the key received.
nextPrime	Get and calculate the next prime number of the table size.
resize	Calculate and set a new size for the Hash Table when it is too empty or too filled.
putResize	Put back the key into the Hash Table after resizing the table.
display	Display the key and the data inside the Hash Table.
getLoadFactor	Measure and calculate how filled the Hash Table is.

DSAHeap class in this program will take in the score, location, temperature, humidity, and the wind speed of the location. The score received will be the values that decided the priority of the location. Below is the method inside DSAHeap class and their function:

Method	Function
add	Get the data and called trickleUp method when there is more than one data in the array.

remove	Remove the data in the first index of the array, called the trickleDown method when there is more than two array filled.
trickleDown	Go through the array from the first to the last array filled, to check if there is any value more prior.
display	Display all the data inside the heap array.
displayLastIndex	Only display the last index that is filled inside the array.
trickleUp	Go through the array from the last to the first array filled, to check if there is any value more prior.
swap	Swap the data when the value of that data is more prior or higher than the other data.
heapSort	Return the data from the least prior to the most prior.
heapify	Organise the array into a heap in increasing order.
leftChild	Get the left child index of array. For the right child index, it is the left child index plus by one.
parent	Get the parent index of array
isFull	Check if the heap array is full.
isEmpty	Check if the heap array is empty.

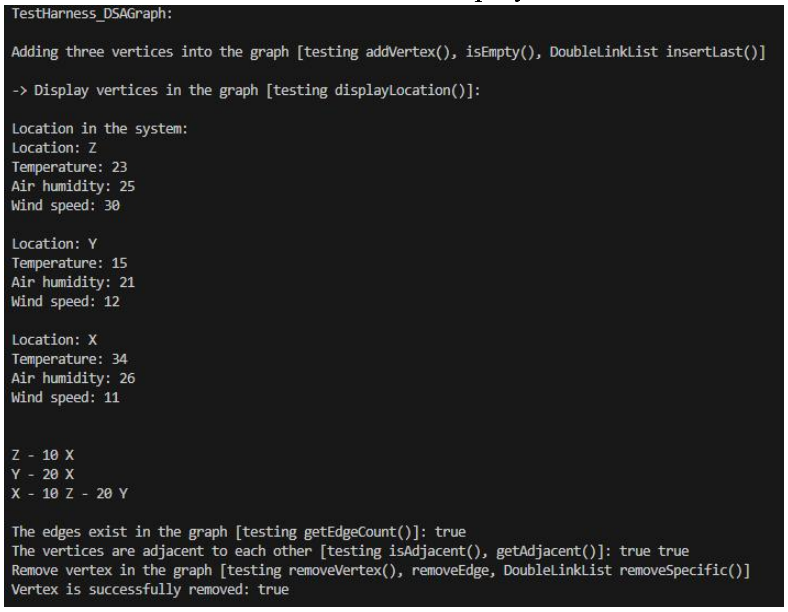
The other class which is private inner class that exist inside the DoubleLinkList class is the MyLinkedListIterator which implements iterator, method present in the class are hasNext, next, nextW, and remove method. Method hasNext, queries if more items exist in the list, method next, move the cursor to get vertex in the list, method nextW, move the cursor to get the vertex weight (location distance) in the list, and lastly remove method is to remove the current item.

While DoubleListNode, DSAGraphNode, DSAHashEntry, DSAHeapEntry is the class which have the setter and getter method for the DoubleLinkList, DSAGraph, DsaHashTable, and DSAHeap class respectively. UnitTestDSAGraph, UnitTestDSAHashTable and UnitTestDSAHeap is the test harness class to test DSAGraph, DSAHashTable and DSAHeap respectively.

Testing Methodology

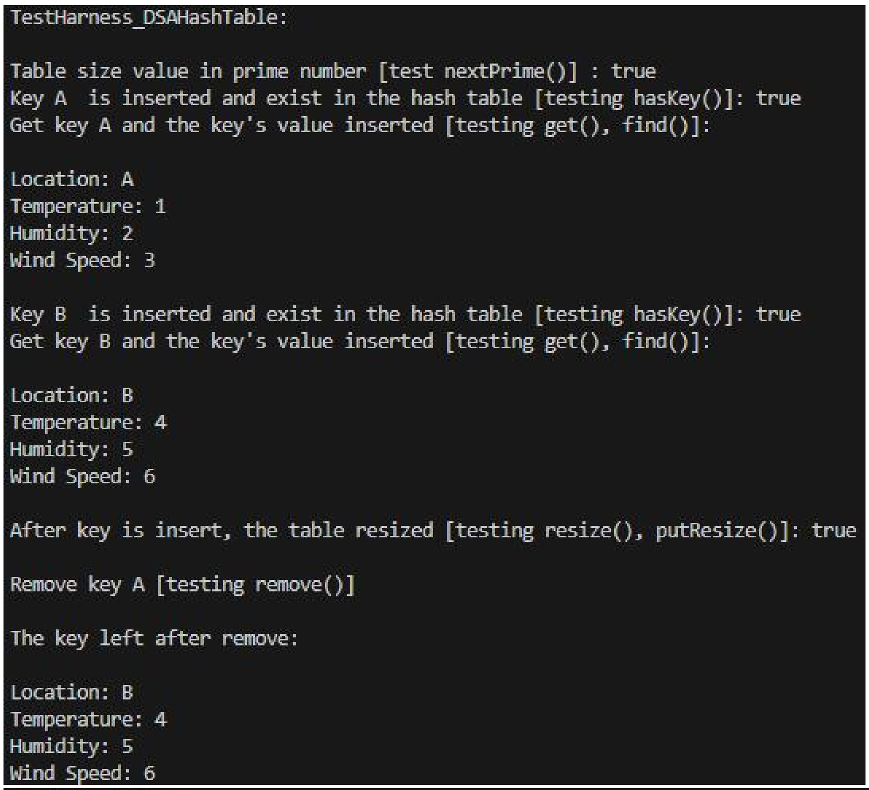
For testing the program, VMware Workstation can be used, make sure all the java class as stated previously and the files, location.txt and UAVdata.txt are already downloaded before testing the program. The steps to test the program are as below:

1. Before running the menu selection, to check and make sure each class can function perfectly, the user can run the test harness first, UnitTestDSAGraph.java, UnitTestDSAHeap.java, UnitTestDSAHashTable.java.
2. Compile the class by using command "javac javaclass.java". For example, to compile UnitTestDSAGraph.java, type command "javac UnitTestDSAGraph.java". in the terminal.
3. After compiling the code, run the code by using command "java javaclass". For running the UnitTestDSAGraph.java, type command "java UnitTestDSAGraph". The outcome as below should then be displayed on the terminal.

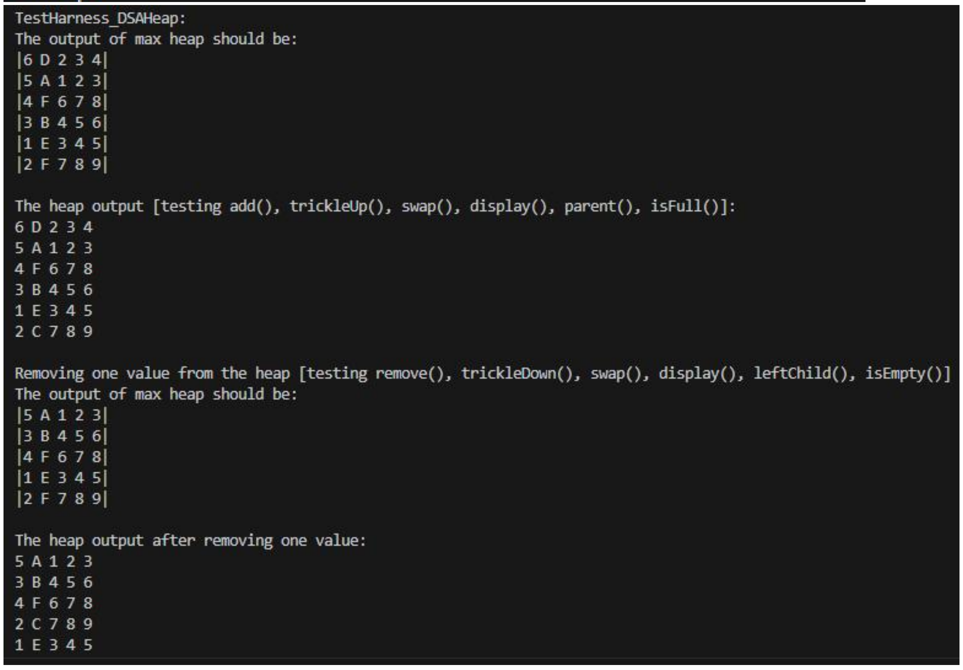


```
TestHarness_DSAGraph:

Adding three vertices into the graph [testing addVertex(), isEmpty(), DoubleLinkList insertLast()]

-> Display vertices in the graph [testing displayLocation()]:

Location in the system:
Location: Z
Temperature: 23
Air humidity: 25
Wind speed: 30

Location: Y
Temperature: 15
Air humidity: 21
Wind speed: 12

Location: X
Temperature: 34
Air humidity: 26
Wind speed: 11


Z - 10 X
Y - 20 X
X - 10 Z - 20 Y

The edges exist in the graph [testing getEdgeCount()]: true
The vertices are adjacent to each other [testing isAdjacent(), getAdjacent()]: true true
Remove vertex in the graph [testing removeVertex(), removeEdge, DoubleLinkList removeSpecific()]
Vertex is successfully removed: true
```

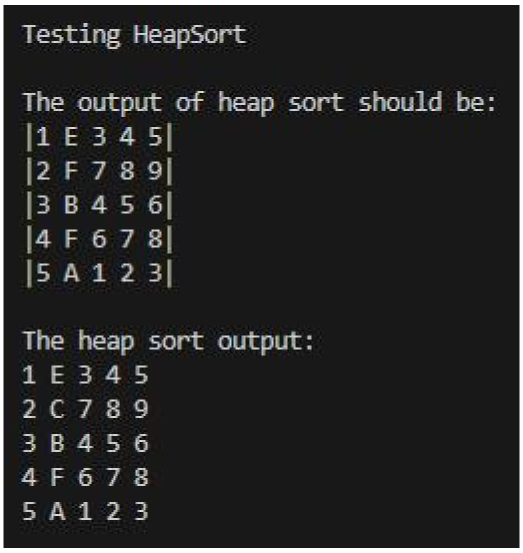
4. Repeat step 2 and 3 with UnitTestDSAHeap.java and UnitTestDSAHashTable.java. The outcome as below should then be displayed on the terminal for UnitTestDSAHeap.java and UnitTestDSAHashTable.java respectively.



```
TestHarness_DSAHashTable:

Table size value in prime number [test nextPrime()] : true
Key A  is inserted and exist in the hash table [testing hasKey()]: true
Get key A and the key's value inserted [testing get(), find()]:

Location: A
Temperature: 1
Humidity: 2
Wind Speed: 3

Key B  is inserted and exist in the hash table [testing hasKey()]: true
Get key B and the key's value inserted [testing get(), find()]:

Location: B
Temperature: 4
Humidity: 5
Wind Speed: 6

After key is insert, the table resized [testing resize(), putResize()]: true

Remove key A [testing remove()]

The key left after remove:

Location: B
Temperature: 4
Humidity: 5
Wind Speed: 6
```

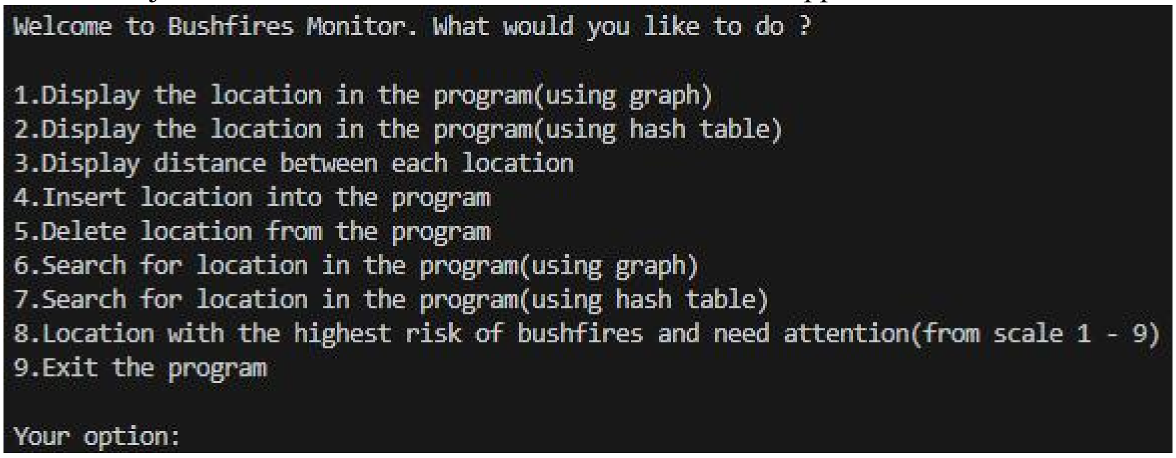


```
TestHarness_DSAHeap:
The output of max heap should be:
|6 D 2 3 4|
|5 A 1 2 3|
|4 F 6 7 8|
|3 B 4 5 6|
|1 E 3 4 5|
|2 F 7 8 9|

The heap output [testing add(), trickleUp(), swap(), display(), parent(), isFull()]:
6 D 2 3 4
5 A 1 2 3
4 F 6 7 8
3 B 4 5 6
1 E 3 4 5
2 C 7 8 9

Removing one value from the heap [testing remove(), trickleDown(), swap(), display(), leftChild(), isEmpty()]
The output of max heap should be:
|5 A 1 2 3|
|3 B 4 5 6|
|4 F 6 7 8|
|1 E 3 4 5|
|2 F 7 8 9|

The heap output after removing one value:
5 A 1 2 3
3 B 4 5 6
4 F 6 7 8
2 C 7 8 9
1 E 3 4 5
```



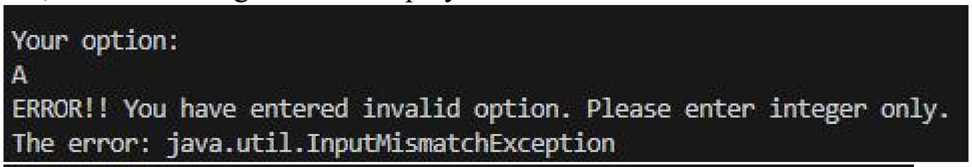
```
Testing HeapSort

The output of heap sort should be:
|1 E 3 4 5|
|2 F 7 8 9|
|3 B 4 5 6|
|4 F 6 7 8|
|5 A 1 2 3|

The heap sort output:
1 E 3 4 5
2 C 7 8 9
3 B 4 5 6
4 F 6 7 8
5 A 1 2 3
```

5. Now, the user can proceed to the menu, and same as before, make sure to compile the code, write command "javac Menu.java". After compiling, run the code by writing the command "java Menu". The menu selection as below should appear:

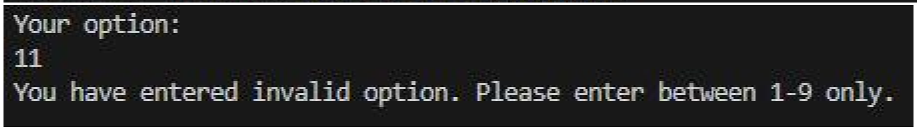


```
Welcome to Bushfires Monitor. What would you like to do ?

1.Display the location in the program(using graph)
2.Display the location in the program(using hash table)
3.Display distance between each location
4.Insert location into the program
5.Delete location from the program
6.Search for location in the program(using graph)
7.Search for location in the program(using hash table)
8.Location with the highest risk of bushfires and need attention(from scale 1 - 9)
9.Exit the program

Your option:
```

6. The user should then type in their option in integer. If the user inserts other data type, such as char, error message will be displayed. While if the integer insert is not in the list, default message will be displayed.

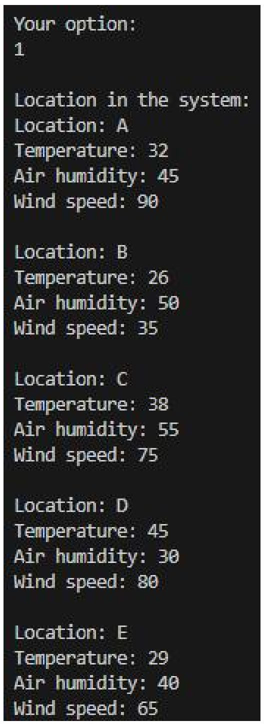


```
Your option:
A
ERROR!! You have entered invalid option. Please enter integer only.
The error: java.util.InputMismatchException
```

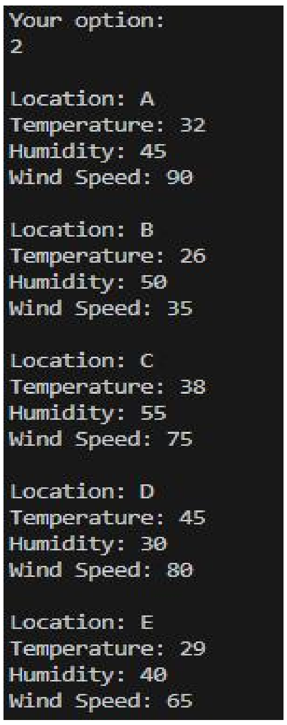


```
Your option:
11
You have entered invalid option. Please enter between 1-9 only.
```

7. For option 1 and 2, both will display the location and the data of the location in the UAVdata.txt. The difference is option 1 is implemented by using graph and option 2 is implemented by using hash table. (Disclaimer: Data shown below is not the complete data.)

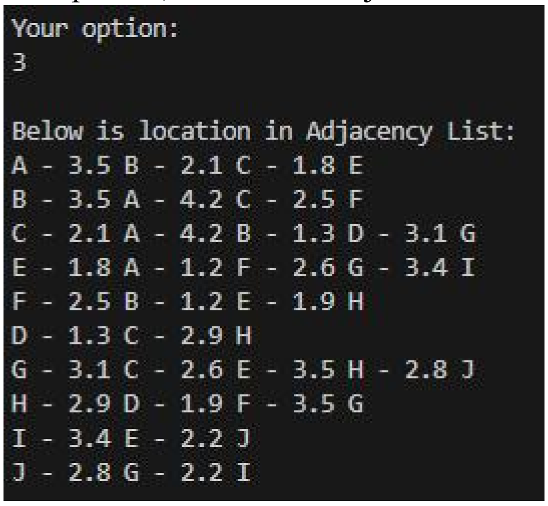


```
Your option:
1

Location in the system:
Location: A
Temperature: 32
Air humidity: 45
Wind speed: 90

Location: B
Temperature: 26
Air humidity: 50
Wind speed: 35

Location: C
Temperature: 38
Air humidity: 55
Wind speed: 75

Location: D
Temperature: 45
Air humidity: 30
Wind speed: 80

Location: E
Temperature: 29
Air humidity: 40
Wind speed: 65
```



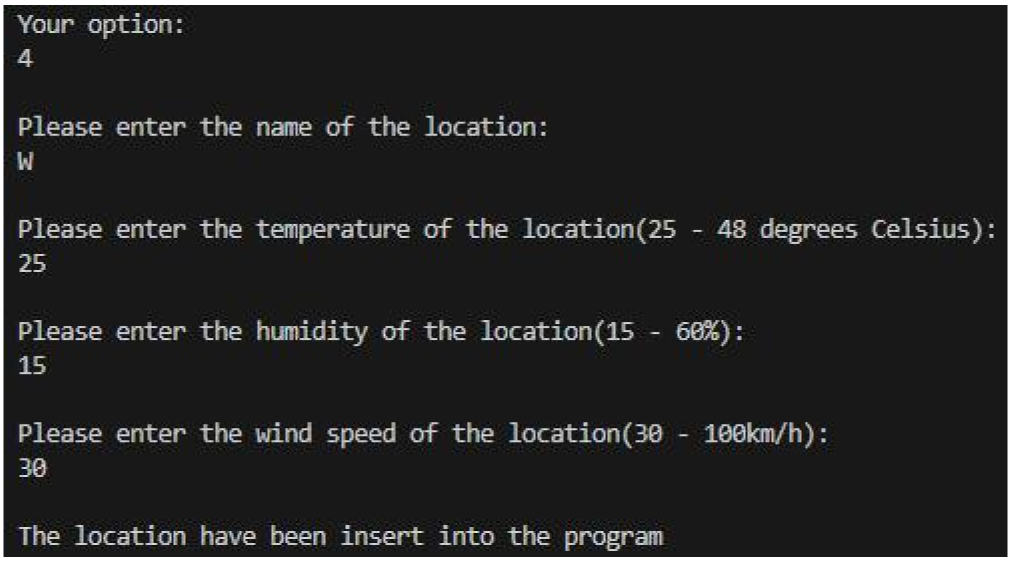
```
Your option:
2

Location: A
Temperature: 32
Humidity: 45
Wind Speed: 90

Location: B
Temperature: 26
Humidity: 50
Wind Speed: 35

Location: C
Temperature: 38
Humidity: 55
Wind Speed: 75

Location: D
Temperature: 45
Humidity: 30
Wind Speed: 80

Location: E
Temperature: 29
Humidity: 40
Wind Speed: 65
```

8. For option 3, the location adjacent to each other and their distance will be displayed.

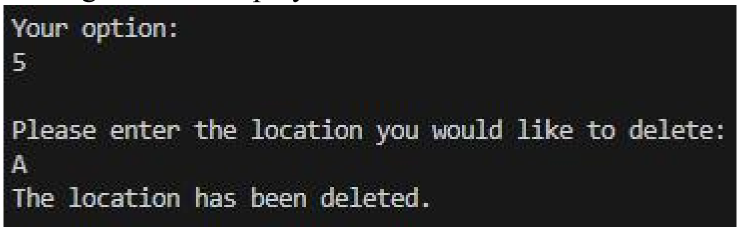


```
Your option:
3

Below is location in Adjacency List:
A - 3.5 B - 2.1 C - 1.8 E
B - 3.5 A - 4.2 C - 2.5 F
C - 2.1 A - 4.2 B - 1.3 D - 3.1 G
E - 1.8 A - 1.2 F - 2.6 G - 3.4 I
F - 2.5 B - 1.2 E - 1.9 H
D - 1.3 C - 2.9 H
G - 3.1 C - 2.6 E - 3.5 H - 2.8 J
H - 2.9 D - 1.9 F - 3.5 G
I - 3.4 E - 2.2 J
J - 2.8 G - 2.2 I
```

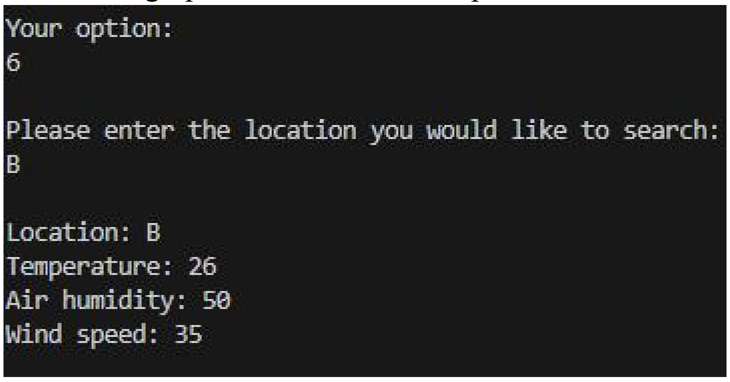
9. Option 4 will ask the user to insert new location, the temperature, humidity, and wind speed of that location into the graph. If the location inserts already exist, error message will be displayed. Error message will also be displayed, if the temperature, humidity and wind speed enter is out of the boundaries, and the program will keep on looping or repeating until the valid value is inserted.


```
Your option:
4

Please enter the name of the location:
W

Please enter the temperature of the location(25 - 48 degrees Celsius):
25

Please enter the humidity of the location(15 - 60%):
15

Please enter the wind speed of the location(30 - 100km/h):
30

The location have been insert into the program
```

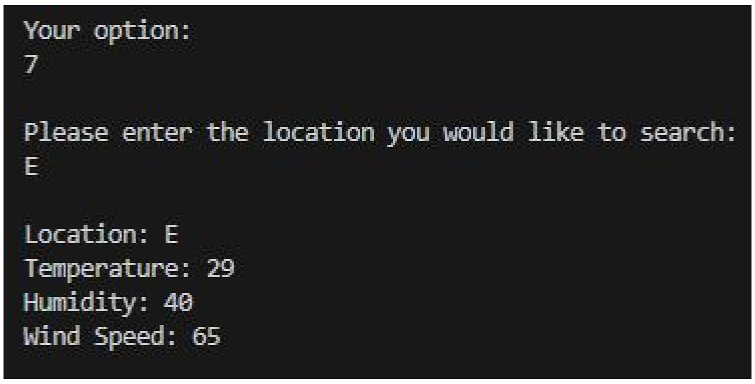
10. Option 5 will delete the location in the graph. If the location inserts do not exist, error message will be displayed.


```
Your option:
5

Please enter the location you would like to delete:
A
The location has been deleted.
```

11. Option 6 will show the certain location and the data location chosen or inserted by the user in the graph. Below is the example of the outcome:

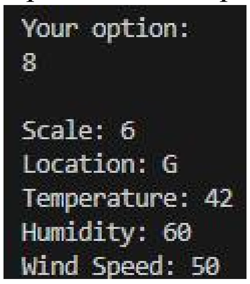

```
Your option:
6

Please enter the location you would like to search:
B

Location: B
Temperature: 26
Air humidity: 50
Wind speed: 35
```

12. Option 7 will show the certain location and the data location chosen or inserted by the user in the hash table. Below is the example of the outcome:



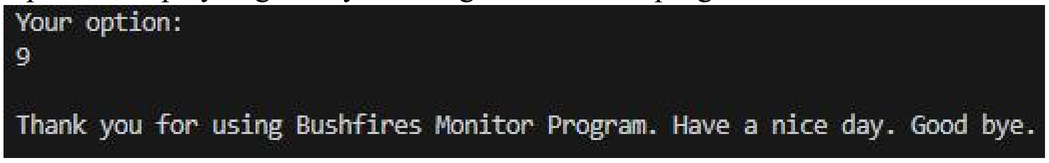
```
Your option:
7

Please enter the location you would like to search:
E

Location: E
Temperature: 29
Humidity: 40
Wind Speed: 65
```

13. Option 8 will display the location with the high risk of bushfires and need attention.



```
Your option:
8

Scale: 6
Location: G
Temperature: 42
Humidity: 60
Wind Speed: 50
```

14. Option 9 displayed good bye message and exit the program.



```
Your option:
9

Thank you for using Bushfires Monitor Program. Have a nice day. Good bye.
```

Result

The program implements the use of graph to store and retrieve the location and the data of the location gathered by the UAVs. Adjacency list approach for the graph is the right decision as some the location are connected to each other and the distance between the location is set as the weight of the edge. The distance and connection of the location can be shown clearly by adding the location into the linked list.

Based on the program, the user can check the location and the data of the location collected by the UAVs. Moreover, the user can get and see the distance between each location that are connected. The users can also delete and find specific data of the location in the program. New location and the data of the new location can also be added into the system. The used of heap allow the user to get the location with the high risk and need attention.

Not only using graph, hash table data structure is also applied to store and retrieve the location and the data collected by the UAVs. The time complexity to access the data by using the hash table is 0(1) while the time complexity to search through a list is 0(N). When using the hash table, from the key received the index of the array can be obtained, and the program can directly go to that index array to obtain the data, less time is required to obtain the data. While when searching through a list or array, the program needs to check the list and the array one by one, until it obtain the data the user want, this can be a waste of time if the data is at the end of the list or array, the program will need to read through every list and array. With this deduction, it is learnt that the use of hash table is more efficient compared to searching through a list or array.