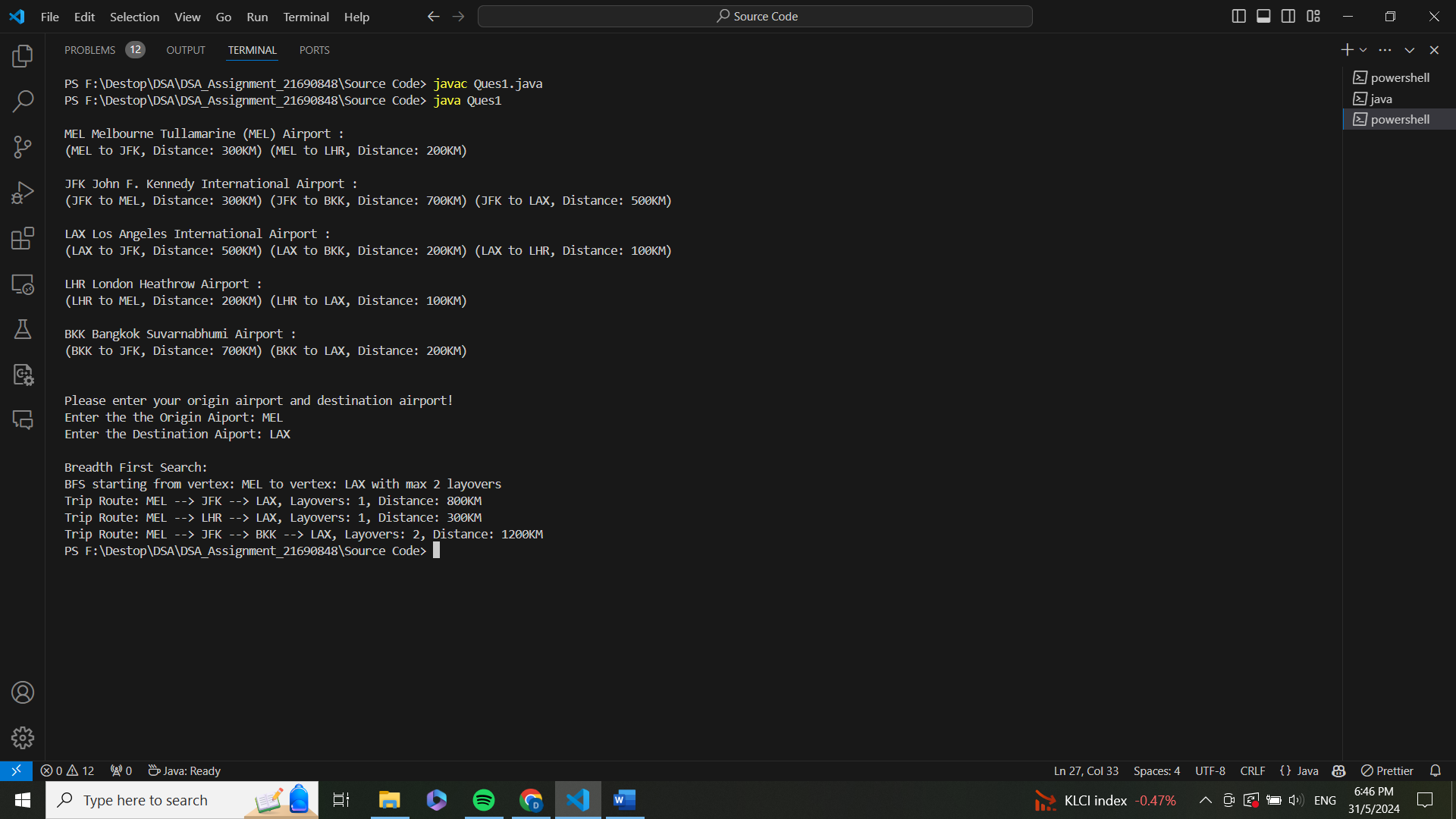
**Report**

**Problem 1: Graph Representation**

* Classes involved in this part to implement the breadth-first search will be DSAGraph, DSAGraphEdge, DSAGraphVertex, DSALinkedList, and DSAListNode.
* DSAGraph is the main class that combines all the other classes to represent the graph structure by storing vertices in a linked list. The graph allows you to add vertices and edges, check for their existence, retrieve vertices and their adjacent vertices, and remove them.
* The class presents a breadth-first search (BFS) algorithm for determining the path between two vertices, with a maximum of two layovers per path.
* The BFS implementation assures that pathways are identified effectively within the stated layover limit, making it appropriate for applications such as routing in the Airplane Management System.

Example Output:



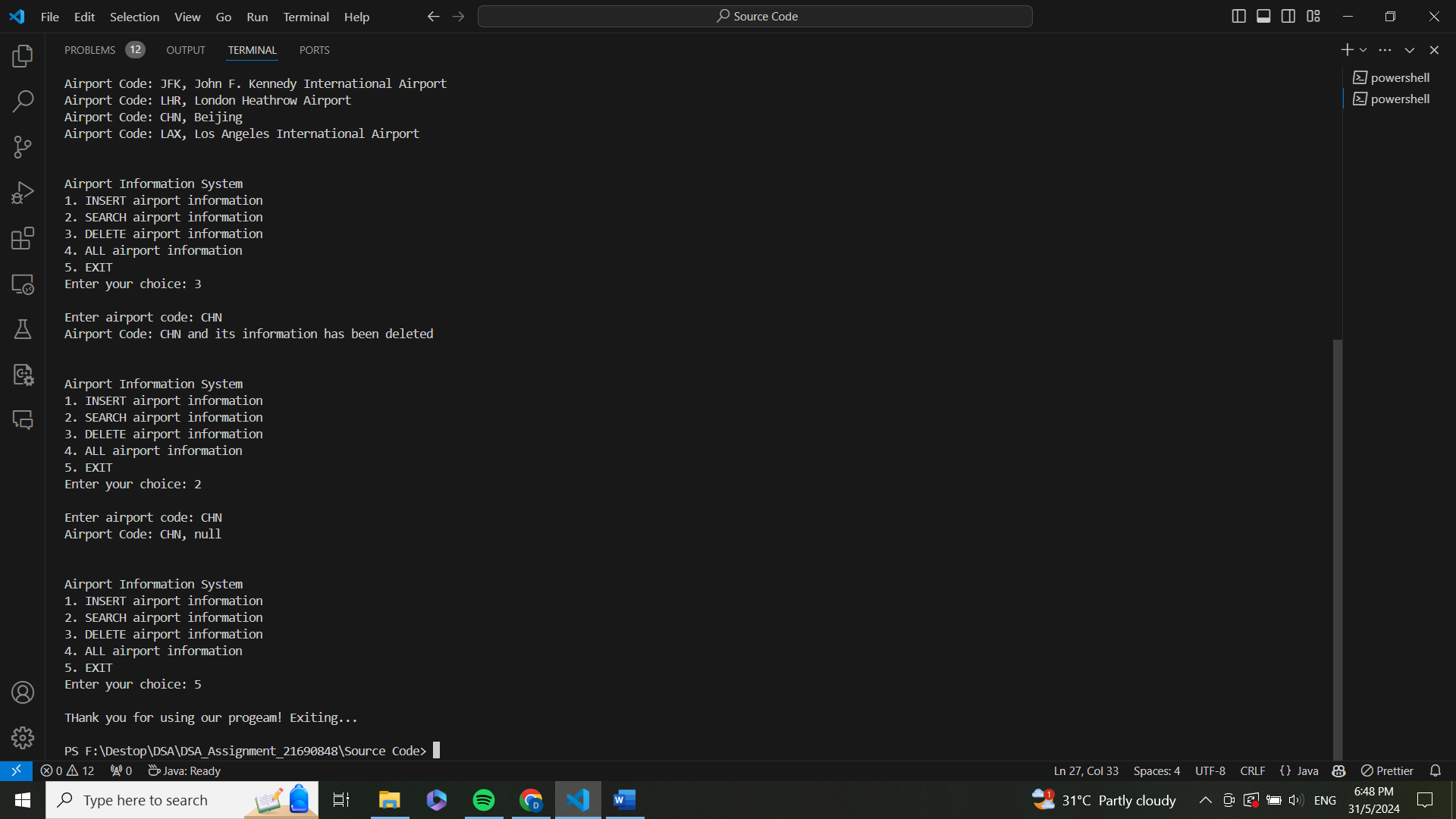
1. Show test data for the Graph of the airports with weight edges of the distance.
2. The Airport will be added to the vertex of the graph.
3. The Airports are then connected by adding an Edge with its distance on the edge.
4. The adjacent of each airport will be displayed to the user.
5. The user can choose to see the routes that are available between two airports.

**Problem 2: Hashing for Efficient Airport Lookup**

* Classes involved in this part to implement the hashing will be DSAHashTable and DSAHashEntry.
* DSAHashTable is the primary class that combines DSAHashEntry to create a hash table with dynamic resizing. It employs linear probing for collision resolution and enables effective space utilization by keeping the load factor within predefined limits.
* The class provides functions to insert, retrieve, and delete key-value pairs, as well as display hash table contents.
* The hash function and linear probing improve access efficiency, while dynamic resizing depending on load factors optimizes space and speed. It efficiently manages insertion, retrieval, deletion, and display actions while keeping a balanced load factor.

Example Output:

A computer screen with many white text

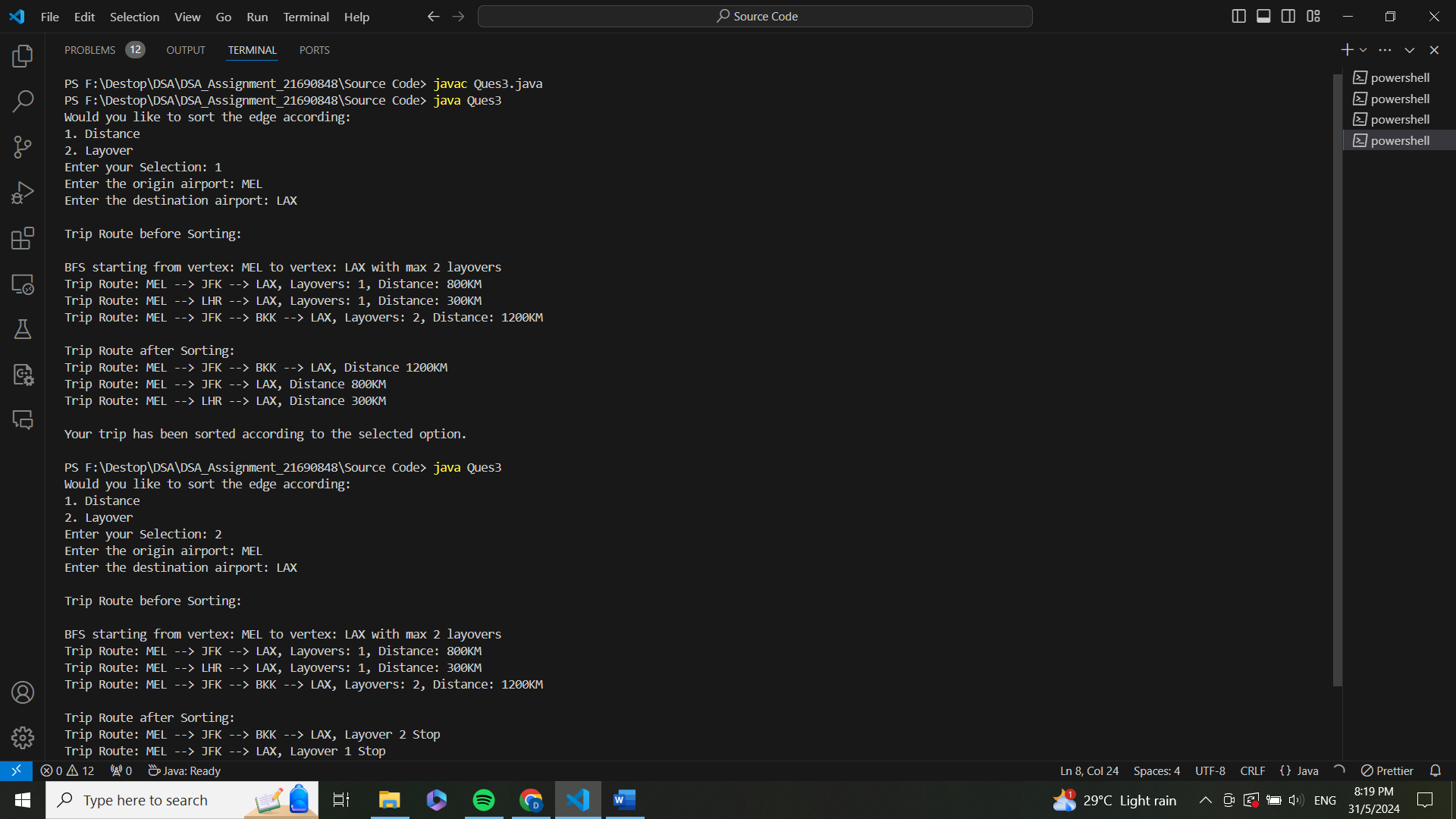
Description automatically generated

1. Shows a few test data for Hash Table with Airport Code and Airport Name.
2. The user can choose an option to add, search, delete, and show all airport information in the Hash Table.

**Problem 3: Heap Sort for Optimal Route Selection**

* Classes involved in this part to implement the Heap Sort will be DSAHeap, DSAHeapEntry and FileIO
* DSAHeap is the main class that combines DSAHeapEntry and FileIO to manage a binary heap. This data structure is used to efficiently perform priority queue operations.
* FileIO retrieves the BFS results and uses it to do sorting according to the user preferences.
* Functions include adding and removing elements, displaying the heap, converting an array to a heap, performing heap sort, and dynamically resizing the heap array.
* The heap function controls heap operations and maintains optimal performance by dynamically resizing. The heap sort algorithm uses the heap structure to sort elements in place, resulting in a stable and efficient sorting mechanism.
* **ATTENTION:** The HaepSort.txt must be deleted every time the program is run for Heap Sort to avoid using the previous programs results.

Example Output:



A screenshot of a computer

Description automatically generated

1. The user will be given two options to sort the routes by distance or layovers.
2. If the option chosen is sorted by distance, the routes displayed are in ascending order according to the distance of the trip.
3. If the option chosen is sorted by layovers, the routes displayed are in ascending order according to the number of layovers.

**Challenges**

The challenge that I faced in the assignment was on the heap sort. I was able to complete the heap sort successfully, but the instruction required is to include the BFS results from discovering all feasible routes into a heap sort to sort them by route trip distance or number of layovers. I couldn't figure out how to store the BFS findings. Finally, I came up with the concept of storing the findings in a txt file, from which I could retrieve the data that needed to be sorted using a heap by just reading the details.

**Potential Improvements**

Improve error handling to provide more descriptive error messages and deal with edge cases more gracefully. Implement more collision resolution techniques to handle collisions more effectively, particularly when the load factor is high.