Sarp Arslan - 11458145526

CMPE 224 – SEC 03 / Programming Assignment 2

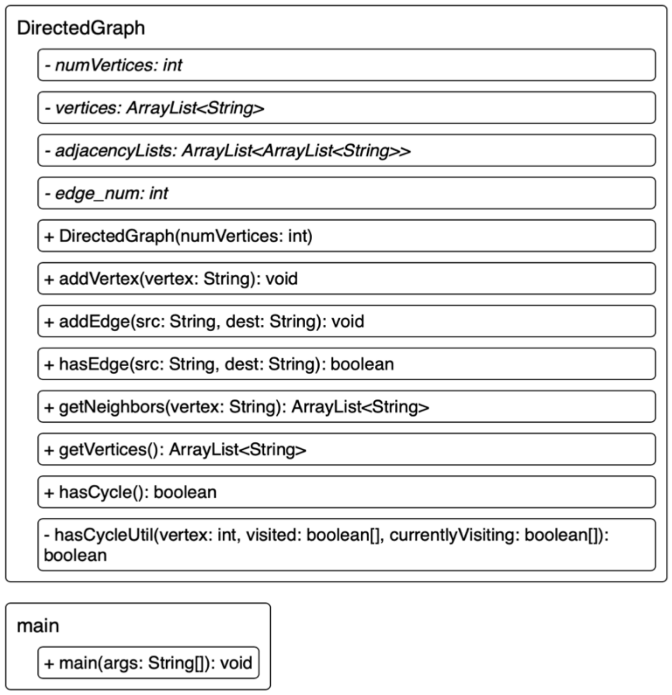
Question 1

Problem Statement and Code Design

In this assignment, the task is to prove whether you can keep company's ride network in a tree structure. The program asks the user for the number of taxi pickups and rides, and reads in the source and destination vertices for each ride. It keeps track of the number of times each taxi pickup location appears and adds the directed edge to the graph. Then, it prints out the adjacency list representation of the graph and checks if it contains a cycle or can be represented as a tree. Finally, it outputs a message indicating whether the ride network can be kept in a tree structure or not.

Implementation, Functionality

This application works by using direct graph structure, and tests whether it includes graph cycle or not. Finally prints the result with vertices and edges.


DirectedGraph
- numVertices: int
- vertices: ArrayList<String>
- adjacencyLists: ArrayList<ArrayList<String>>
- edge_num: int
+ DirectedGraph(numVertices: int)
+ addVertex(vertex: String): void
+ addEdge(src: String, dest: String): void
+ hasEdge(src: String, dest: String): boolean
+ getNeighbors(vertex: String): ArrayList<String>
+ getVertices(): ArrayList<String>
+ hasCycle(): boolean
- hasCycleUtil(vertex: int, visited: boolean[], currentlyVisiting: boolean[]): boolean
main
+ main(args: String[]): void

Testing

Testing this question was simpler than the 2nd question due to the small size of the inoputs. I did not encounter any errors. I just didn't know that I had to write the links of the graph in my hand upside down, so for a while I did not get the output Emek: Yenimahalle Kızılay, instead of that my program’s output was Emek: Kızılay Yenimahalle. I was able to solve this problem with my instructor drawing me attention to this issue in office hour.

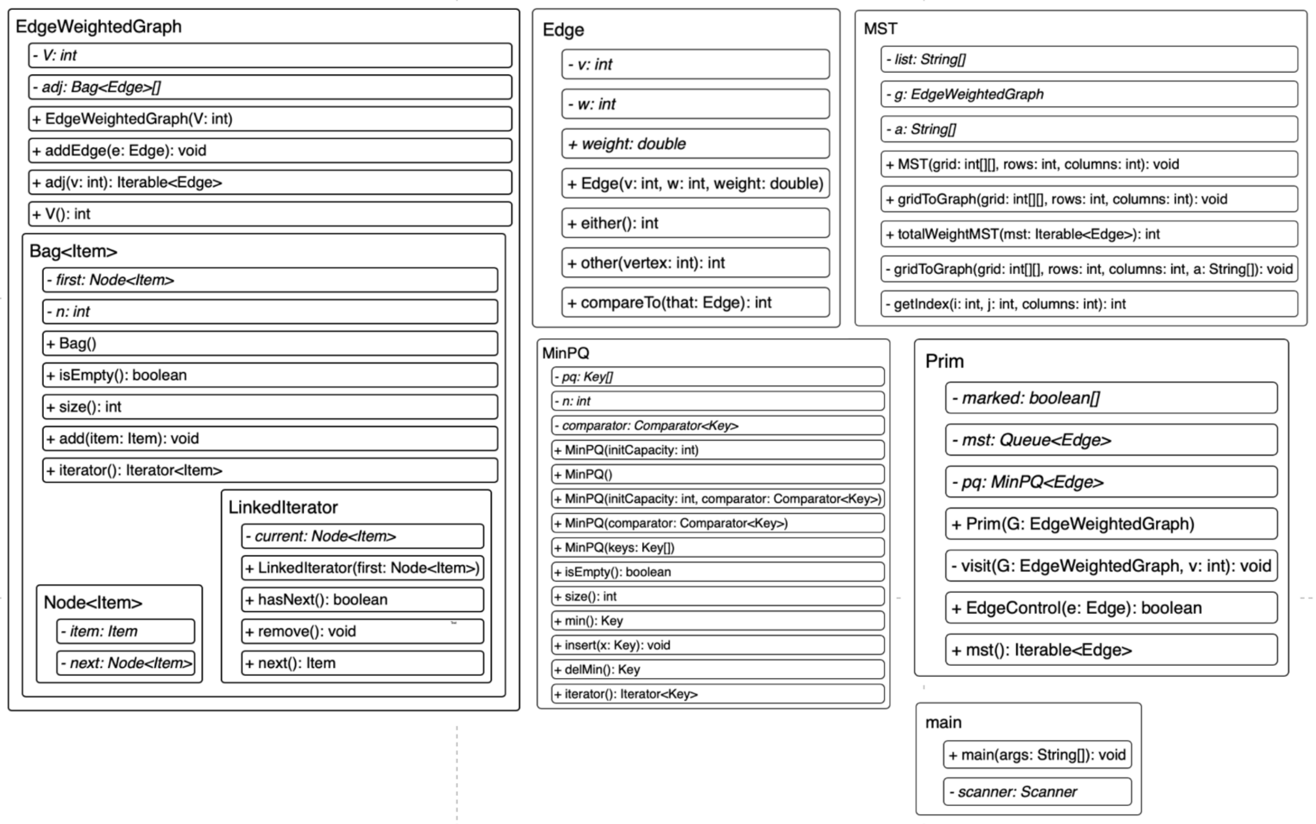
Question 2

Problem Statement and Code Design

The code defines a main class that reads input from the user via Scanner. It starts by getting the number of test cases and then loops through each test case. For each test case, it gets the number of rows and columns in the grid, as well as the starting and ending cells. It then creates a 2D array to represent the grid and fills it with integers. Finally, it creates an instance of the MST class, uses Prime Algorithm and passes the grid, rows, and columns to it.

Implementation, Functionality

This program works with Edge, EdgeWeightedGraph, Minimum spanning tree, minimum priority queye structures and Prim algorithm. It uses Minimum spanning tree and premium algorithm to find the shortest path, finds its connections with the help of the edge class, and the painful ones with the help of edgeWeightedGraph


EdgeWeightedGraph
- V: int
- adj: Bag<Edge>[]
+ EdgeWeightedGraph(V: int)
+ addEdge(e: Edge): void
+ adj(v: int): Iterable<Edge>
+ V(): int
Bag<Item>
- first: Node<Item>
- n: int
+ Bag()
+ isEmpty(): boolean
+ size(): int
+ add(item: Item): void
+ iterator(): Iterator<Item>
LinkedIterator
- current: Node<Item>
+ LinkedIterator(first: Node<Item>)
+ hasNext(): boolean
+ remove(): void
+ next(): Item
Node<Item>
- item: Item
- next: Node<Item>
Edge
- v: int
- w: int
+ weight: double
+ Edge(v: int, w: int, weight: double)
+ either(): int
+ other(vertex: int): int
+ compareTo(that: Edge): int
MST
- list: String[]
- g: EdgeWeightedGraph
- a: String[]
+ MST(grid: int[][], rows: int, columns: int): void
+ gridToGraph(grid: int[][], rows: int, columns: int): void
+ totalWeightMST(mst: Iterable<Edge>): int
- gridToGraph(grid: int[][], rows: int, columns: int, a: String[]): void
- getIndex(i: int, j: int, columns: int): int
MinPQ
- pq: Key[]
- n: int
- comparator: Comparator<Key>
+ MinPQ(initCapacity: int)
+ MinPQ()
+ MinPQ(initCapacity: int, comparator: Comparator<Key>)
+ MinPQ(comparator: Comparator<Key>)
+ MinPQ(keys: Key[])
+ isEmpty(): boolean
+ size(): int
+ min(): Key
+ insert(x: Key): void
+ delMin(): Key
+ iterator(): Iterator<Key>
Prim
- marked: boolean[]
- mst: Queue<Edge>
- pq: MinPQ<Edge>
+ Prim(G: EdgeWeightedGraph)
- visit(G: EdgeWeightedGraph, v: int): void
+ EdgeControl(e: Edge): boolean
+ mst(): Iterable<Edge>
main
+ main(args: String[]): void
- scanner: Scanner

Testing

It was very difficult to test the program in this question, especially since the 3rd test case was too long, I had a hard time seeing the error, and thanks to this application, I remembered that a correctly written algorithm will work correctly regardless of the number of inputs.

Final Assessments

For the first question, the trouble points in completing this assignment are understanding the concept of a directed graph, designing an algorithm to check if a directed graph can be a tree, implementing the algorithm using appropriate data structures such as symbol chart, and validating the correctness of the algorithm through testing. The most challenging parts of this assignment may be understanding the concept of a directed graph and implementing the algorithm to check for cycles in the graph.

For the second question, the most challenging part of this assignment is probably implementing the MST algorithm and optimizing the algorithm's performance. It involves calculating the cost of the edges, constructing the MST, and calculating the cost of the good trip.

What I liked about this assignment is that it provides a good opportunity to practice implementing a Prime Algorithm.

I liked the real-world application of the problem in the context of a ride-sharing company, which made it more interesting and relatable. This assignment also helped me learn about directed graphs, trees, using Prime Algorithm and algorithms to check for cycles in a graph.