CMPE 224/343 HW2 REPORT

I) INFORMATION

ID: 47113751436 **Assignment number: 02**

Name: Doğukan Us **Section: 03**

II) PROBLEM STATEMENT AND CODE DESIGN

Task 1:

My first task is about a flight network problem. With the given input of user, I was asked to create a directed graph and insert the cities to the vertices and connect them with directed edges according to the input.

Structure chart of Task 1 is shown in *Figure 1*.

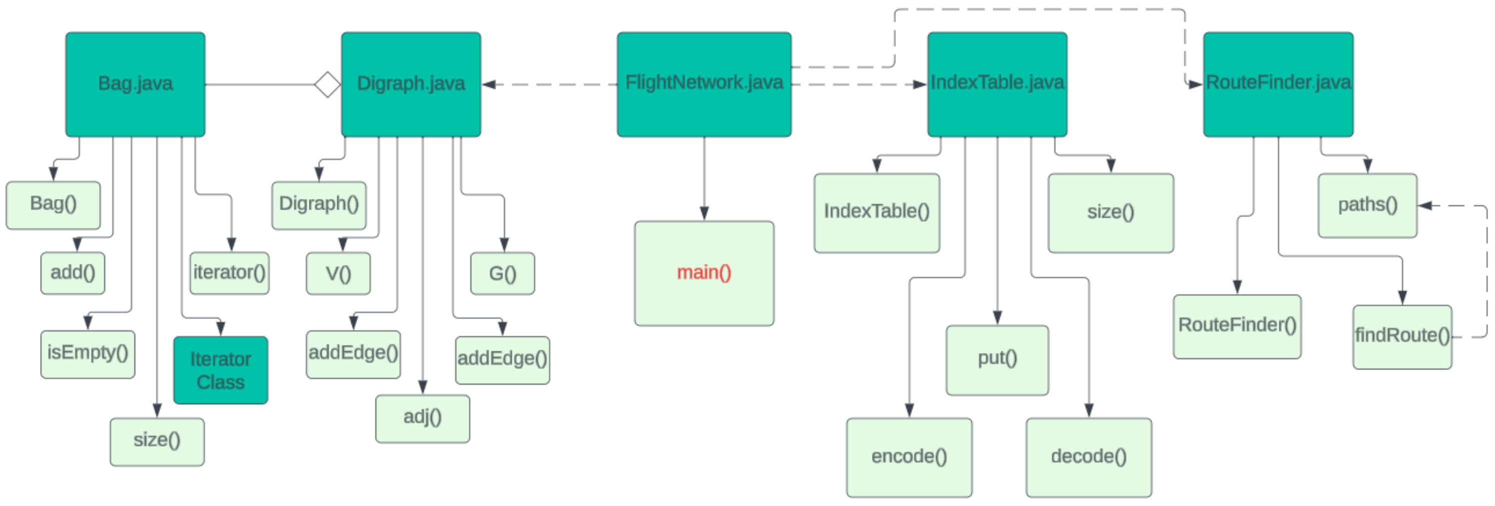

Figure 1 (Structure chart of Task 1)

Task 2:

My second task is about a maze solving problem. The mazes are given in an input file, and I was asked to create a maze by using directed graph, connecting the not-wall characters if they are adjacent to each other by using directed edges, and find all paths through targets which are called treasures.

Structure chart of Task 2 is shown in *Figure 2*.

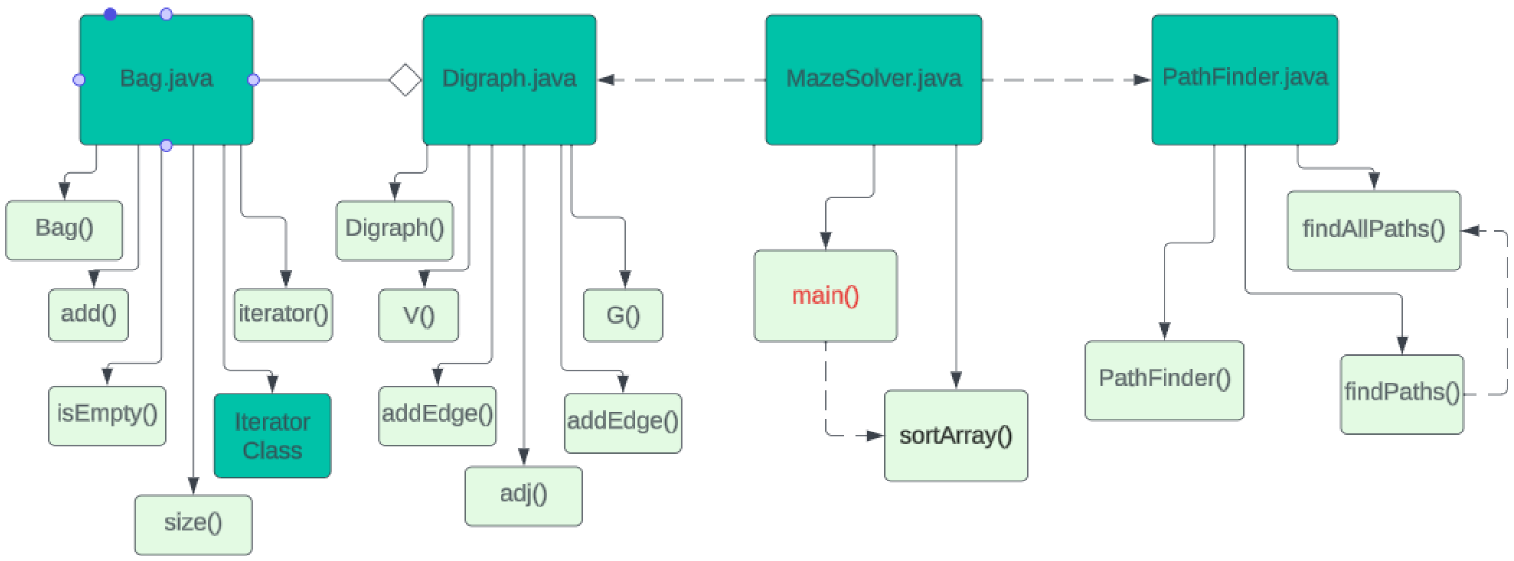

Figure 2 (Structure chart of Task 2)

III) IMPLEMENTATION, FUNCTIONALITY

Task 1:

Digraph Class:

I used the Digraph class inspiring from classes used in our slides and Sedgewick's book

IndexTable Class:

```
public IndexTable() {
```

- Constructor that initializes the table.

```
public void put(String s) {
```

- Puts the String s to the table if it's not already in the table.

```
public int encode(String s) {
```

- Returns the index of the string s from the list.

```
public String decode(int i) {
```

- Returns the String at given index

```
public int size() {
```

- Gives the size of the table which is private.

RouteFinder Class:

```
public RouteFinder(int h) {
```

- Constructor that initializes number of hops as given in parameter, route and routes listes

```
public ArrayList<int[]> findRoute(Digraph g, int v) {
```

- The function called from main function, uses the graph given in parameter and the start index v. Calls paths function to calculate all routes and returns the routes list.

```
public void paths(int v, Digraph g, int count) {
```

- Uses the directed graph g given in the parameter, v as the start index and count which is equals to number of hops. It calls itself recursively for the adjacent vertices of v with value of one incremented version of count, keeps track of the path. After count reaches to number of hops, it stores the path in routes list.

FlightNetwork (main) Class:

```
public static void main(String[] args) {
```

- This is the class includes main function. In the main function, a scanner object reads input from the user, reads it line by line and stores it in a list, creates an IndexTable object and puts the cities in the list to use their index number as the vertices of the graph. Then creates a directed graph, adds edges to the graph according to the list of the lines. Creates a RouteFinder object to call its functions. After that, gets the calculated routes and prints it to the console.

Task 2:

Digraph Class:

- I used the Digraph class inspiring from classes used in our slides and Sedgewick's book

PathFinder Class:

```
public PathFinder(ArrayList<Integer> t) {
```

- Constructor that initializes the treasures list by using the parameter t which is an array list.

```
public ArrayList<int[]> findPaths(Digraph g,int v) {
```

- The function called from main function. Uses parameter g as the digraph to work on, and parameter v as start vertex. It creates a Boolean array to track visited vertices and path list to track the current path it is searching on. Adds start vertex as the first element of the path list. For all elements of treasures list, calls findAllPaths() function by sending the graph g, the start vertex v, the target vertex e which is the treasure, and the Boolean array and the path list as parameter to calculate the paths. At the end, returns the paths list.

```
private void findAllPaths(Digraph g, int v, int e, boolean[] visited, ArrayList<Integer> path) {
```

- Uses the parameters sent by findPaths() function, iterates recursively over the adjacent vertices of the vertex v. Adds current vertex to the path and if cannot find the target destination vertex, removes it from the path. If the current vertex and the target destination vertex matches, it creates a temporary array to store the current version of the path list and adds it to the paths array.

MazeSolver (main) Class:

```
public static void main(String[] args) {
```

- This class contains the main method. In the main method, a scanner object reads input from the user, reads it line by line and stores each character of the line in the maze list. Creates digraph object and adds edges according to the neighbor vertices computed by modulo operation. If the neighbor vertex is not a wall character, it creates a directed edge between them. Creates a PathFinder object to call its function. After the function call, it stores the returned paths list in a String array, sends it as the parameter of sortArray() function and prints out the sorted paths to the console.

```
public static void sortArray(String[] s) {
```

- Gets a String array as parameters, starting from the 0th index, checks if the element has larger length from the next element or not. If so, switches the element.

IV) FINAL ASSESSMENTS

To conclude my experience about this assignment, it has some challenging problems. Understanding how the search algorithms work and implementing directed graph structure dynamically according to the user input was the two topics I had trouble with. Especially, getting the true outputs by the search algorithm to find desired paths was the most difficult part for me. But at the and of the day, this assignment really helped me about gaining a big understanding of searching algorithms and path finding.