Data Structures and Algorithms II

HOMEWORK REPORT

Duygu Yesiloglu ID: 43018881148

CMPE 224 - SEC 03

Instructor: Tolga Kurtuluş Çapın

Problem Statement and Code Design

For Q1, the purpose of this code is to find the shortest path from X to Y which are our starting and target city. Time data's are significant for both in the state change and in the time spent on each route. The lexicographically smallest route amongst the shortest paths is to be decided. I have also used graph theory algorithms, for this question BFT, and determined the lexicographically smallest path.

For Q2, a tour guide has to create roads of islands N and M, create the routes, and then go back to the starting point from that path. For the output, it should be lexicographically smallest path that satisfies the requirements.

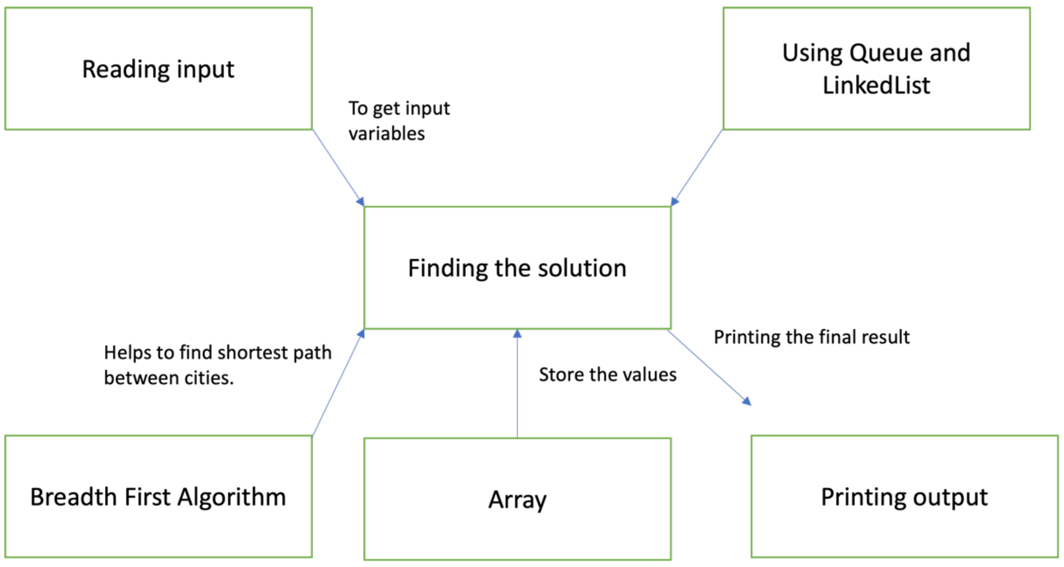
Also, our codes include several subs. These sub modules are basically defined using a structure chart below.

Implementation, Functionality

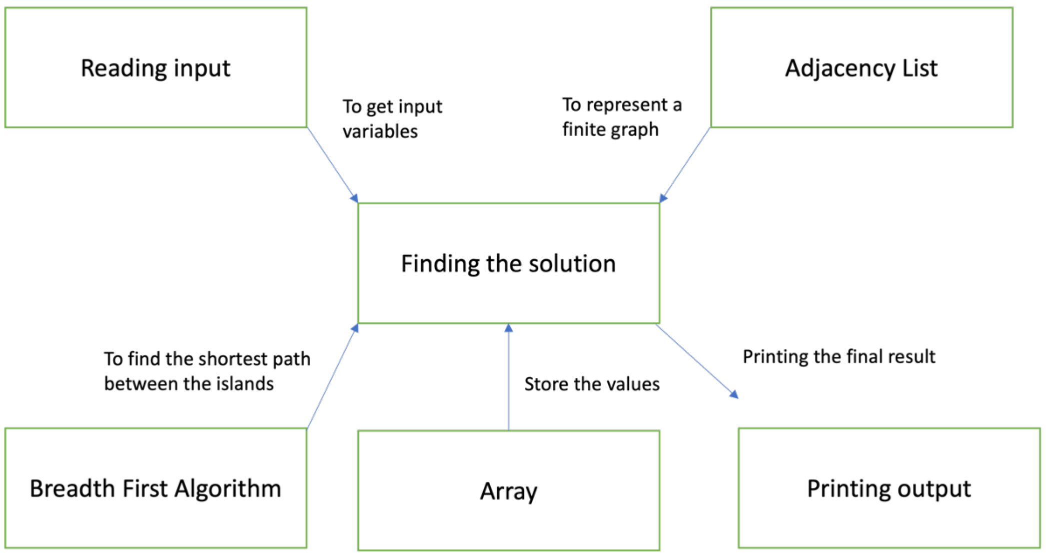
For all questions, the priority was to read input values from user. Then, I have defined the Manage() method for both, it initialize arrays which are road connections between cities, distance and parent of each city, cities have been visited and roads per city. I wrote methods according to the BFS search algorithm structure. Queue and LinkedList have been created for the algorithm.

The details about each submodule for all questions are described in the following section, in the order in which the logic of the programs work:

Q1:



Q2:



For Q1;

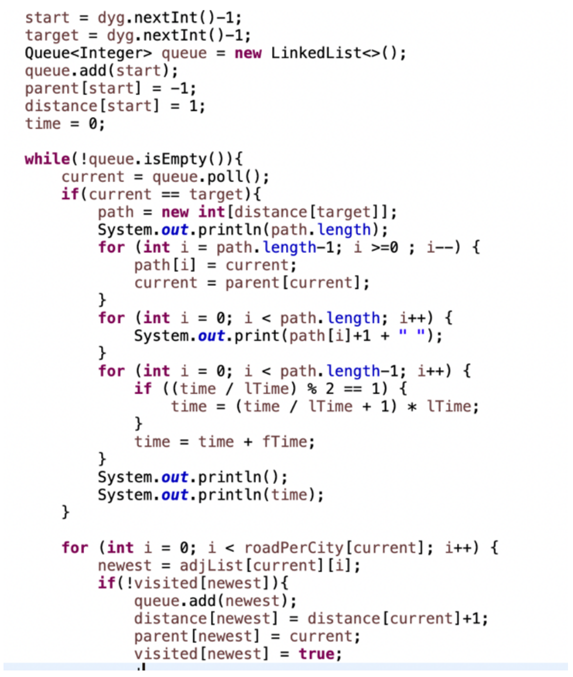
1. **Scanner:** To get our inputs from user, we have to use Scanner object. This will be helping to access user inputs which includes number of cities, roads between cities, loading time and the flight time between cities.
2. **Manage Method:** This method is the skeleton of the code. Manage() creates and initializes various arrays for the purpose of representing the connections between cities through roads, which cities have been visited, distance and parent of each city, and number of roads per city. One of these arrays is an adjacency list.

3. **Queue:** Using queue for selecting the starting city and target city. I set starting point to -1 and the distance of staring city is 1. After that, loop continues until the queue is empty. The loop checks to see if there is a target city each time a node arrives. In this case, the shortest path is found and saved to a new array after returning from the original array.
4. **Arrays**: I determined ‘adjlist’ which is 2d array and represent adjacency list of the graph, ‘visited’ array is a Boolean and it follows node using breadth algorithm. Path is an int array and stores all the cities, it helps to find shortest way between the nodes. Distance array stores distance between start and target city. Last one is roadPercity it stores number of roads connect to another.
5. **Breadth-First Algorithm:** I used an adjacency list and a breadth first search algorithm to find the shortest path, it prints and get to find total travel time. It starts by intituling queue and adding start node to this. I set the distance and parent of start node by 0 and -1.

For Q2:

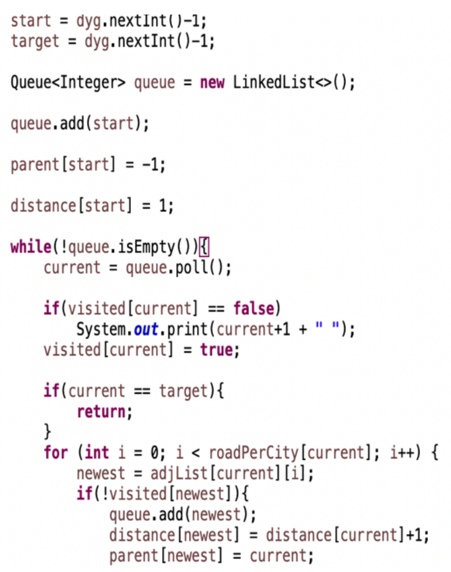
1. **Scanner:** Reading values from user which was inputted and create adjacency list to represent connections between the island M and N.
2. **BFS**: There can be several roads and paths, to find all possible paths I user Breadth First Search algorithm.
3. **Arrays:** It helps the store the values like path, distance and parent.
4. **Manage Method:** It is works same with Q1.I applied BFS as well.
5. **Printing:** After sorting the list of paths lexicographically order, I printed the first path from the list.

Q1:

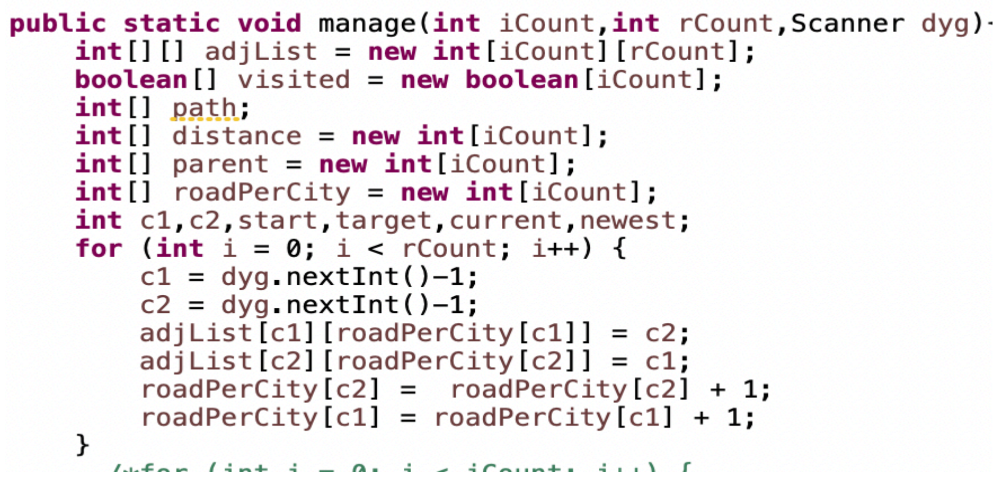


```
start = dyg.nextInt()-1;
target = dyg.nextInt()-1;
Queue<Integer> queue = new LinkedList<>();
queue.add(start);
parent[start] = -1;
distance[start] = 1;
time = 0;

while(!queue.isEmpty()){
    current = queue.poll();
    if(current == target){
        path = new int[distance[target]];
        System.out.println(path.length);
        for (int i = path.length-1; i >=0 ; i--) {
            path[i] = current;
            current = parent[current];
        }
        for (int i = 0; i < path.length; i++) {
            System.out.print(path[i]+1 + " ");
        }
        for (int i = 0; i < path.length-1; i++) {
            if ((time / lTime) % 2 == 1) {
                time = (time / lTime + 1) * lTime;
            }
            time = time + fTime;
        }
        System.out.println();
        System.out.println(time);
    }

    for (int i = 0; i < roadPerCity[current]; i++) {
        newest = adjList[current][i];
        if(!visited[newest]){
            queue.add(newest);
            distance[newest] = distance[current]+1;
            parent[newest] = current;
            visited[newest] = true;
```

Q2:



```
start = dyg.nextInt()-1;
target = dyg.nextInt()-1;

Queue<Integer> queue = new LinkedList<>();

queue.add(start);

parent[start] = -1;

distance[start] = 1;

while(!queue.isEmpty()){
    current = queue.poll();

    if(visited[current] == false)
        System.out.print(current+1 + " ");
    visited[current] = true;

    if(current == target){
        return;
    }
    for (int i = 0; i < roadPerCity[current]; i++) {
        newest = adjList[current][i];
        if(!visited[newest]){
            queue.add(newest);
            distance[newest] = distance[current]+1;
            parent[newest] = current;
```



```java
public static void manage(int iCount,int rCount,Scanner dyg)
    int[][] adjList = new int[iCount][rCount];
    boolean[] visited = new boolean[iCount];
    int[] path;
    int[] distance = new int[iCount];
    int[] parent = new int[iCount];
    int[] roadPerCity = new int[iCount];
    int c1,c2,start,target,current,newest;
    for (int i = 0; i < rCount; i++) {
        c1 = dyg.nextInt()-1;
        c2 = dyg.nextInt()-1;
        adjList[c1][roadPerCity[c1]] = c2;
        adjList[c2][roadPerCity[c2]] = c1;
        roadPerCity[c2] =  roadPerCity[c2] + 1;
        roadPerCity[c1] = roadPerCity[c1] + 1;
    }
      /*for (int i = 0; i < iCount; i++) {
```

TESTING

I had to use nodes and other data structures to find the shortest path between the cities and islands. After understanding the details and usage of these structures, I started writing the code. The significant part of this questions is understanding the question correctly. I had calculation errors because I misunderstood, and in VPL the tests did not pass. Then, by constantly checking my loops, the hidden and expected outputs came out. The first question was 100% successful and the second question was 66% successful.

FINAL ASSESSMENTS

1. Thanks to this assignment, I learned Breadth First search algorithm better. I got some solving complexities in my codes, but it made me do research and got an idea about depth first algorithm as well. BFT is a good method for finding shortest path.
2. Neither of them were very easy problems but both were very instructive. My knowledge of algorithms increased and I gained experience.