Lab 12 Graphs

Weichen Chai

Fall 2022

Introduction

In this assignment, we are asked to work with "Graphs", which is a data structure that stores connected data, it contains nodes and edges, with edges being connected to different nodes to form. In this assignment, we will use graphs to solve problems regarding the shortest paths between two cities.

Background

A simple graph is a tree, which is a special graph that has root, direct edges, where only one node is the destination for one edge. It is a simple non-circular graph structure. Graphs are basically made up by nodes and edges, which connects nodes to one another to represent paths.

Tasks

A train from Malmö

In this task, we will the descriptions given and turn them into a graph that can be used to find the shortest path between cities. The file we refer to is given in the task as a CSV file.

the graph

The format in the CSV file is : "from,to,minutes". The requirement for this task is to present a city as a structure with name and neighbors stored in an array holding connections. Connection is a city and distance in minutes.

```
String name;
Connection[] connections;
Integer p = 0;

public City(String name) {
    this.name = name;
```

```
        this.connections = new Connection[0];
    }

    ...

    public class Connection {
     City neighbors;
    Integer distance;

    public Connection(City neighbors, Integer distance) {
        this.neighbors = neighbors;
        this.distance = distance;
    }
```

hashing to our rescue

In this assignment, we will use a hash function that takes in a string as argument. By following the assignment, the code we receive is.

```
  private Integer hash(String name) {
        Integer hash = 7;
        for (int i = 0; i < name.length(); i++) {
            hash = (hash * 31 % mod) + name.charAt(i);
        }

        return hash % mod;
    }
```

We can utilize hashing in the lookup function , which will be used later in maps.

```
public City lookup(String name) {
        Integer look = hash(name);
        while (cities[look] != null) {
            if (cities[look].name.equals(name)) { // lookup
                return cities[look];
            }
...
```

The Map

In order to finish the skeleton code given to us in the task of map, we should first implement hash and lookup, which has been shown above. This section has a requirements which states that we should lookup the two cities and add a connection to each. In order to perform this operation, we use lookup function for the first city and the second city, then connect city one to city two, and city two to city one.

```
City one = lookup(row[0]);
City two = lookup(row[1]);
Integer distance  = Integer.valueOf(row[2]);
one.connect(two, distance);
two.connect(one, distance);
```

Shortest path from A to B

In this section we will be discussing the shortest path between two cities. This can be done by doing a depth first search technique, where the algorithm explores as far as possible down each branch before moving on to another. The skeleton code has been given in the assignment, therefore, it will not be shown here.

We will have a max value, which is set to avoid being stuck in infinite loops. The max value is the maximum amount of moves to have. In addition, we will

```
...
        Connection conn = from.connections[i];
        Integer distance = shortest(to, from, max - conn.distance);
        if ((distance != null){
            distance += conn.distance;
            if((shrt == null) || (distance.compareTo(shrt) > 0)) {
                 shrt = distance;
            }
        }
   return shrt;
 ...
```

some benchmarks

The tasks asks for the shortest path, the distance, as well as the time required to find these distances. The table is as following. with travel time to those places in minutes, and time used to find these in millisecond.

In the table above, the benchmark indicates that a path can be found from Umeå to Göteborg, but the path from Göteborg to Umeå takes way to long to find, to the point where I have just put ”not found” in the table. Even though technically the time traveled should be also 705 minutes, but

Path	**Travel (Min)**	**Found(ms)**
Malmö, Göteborg	153	3
Göteborg, Stockholm	211	5
Malmö, Stockholm	273	6
Stockholm, Sundsvall	327	89
Stockholm, Umeå	517	83094
Göteborg, Sundsvall	515	30050
Sundsvall, Umeå	190	3
Umeå, Göteborg	705	5
Göteborg, Umeå	Not found	Not found

it can not be found when ran. This might be due to the fact that Göteborg has connections to many other cities, and in this case, we will be looping a lot in comparison to Umeå which is connected to a small amount of cities, therefore, it will be much quicker.

detect loops

Since looping is a problem we had be in the previous task, we will be finding a way to avoid them. We will make a an array called : path, which is big should hold hold any paths. The skeleton code is given in the task, we we would insert the code we have implemented previously in Naive.java into this.

Path	**Travel (Min)**	**Found(ms)**
Malmö, Göteborg	153	402
Göteborg, Stockholm	211	231
Malmö, Stockholm	273	256
Stockholm, Sundsvall	327	202
Stockholm, Umeå	517	294
Göteborg, Sundsvall	515	263
Sundsvall, Umeå	190	593
Umeå, Göteborg	705	307
Göteborg, Umeå	705	329
Malmö, Kiruna	1162	862

As shown above, the implementation allows us to find the path that we were not able to before, this new benchmark table shows a more average run time in comparison to before. The ones that were quick when performing the first benchmark are drastically slower, this may be due to the fact that a traversal of the list of already checked cities have to be performed each search, however, this allows for us to solve the issue with the previous benchmark, where the found times are much longer.

In order to improve upon this, we use the max value, where we can limit the shortest time according to the task descriptions. The result are below.

Path	**Travel (Min)**	**Found(ms)**
Malmö, Göteborg	153	0.6
Göteborg, Stockholm	211	0.7
Malmö, Stockholm	273	0.5
Stockholm, Sundsvall	327	6.8
Stockholm, Umeå	517	15.8
Göteborg, Sundsvall	515	8.7
Sundsvall, Umeå	190	51.1
Umeå, Göteborg	705	1.3
Göteborg, Umeå	705	45.6
Malmö, Kiruna	1162	178.5

The time was decreased from the second benchmark, the search times are improved dramatically.

Things to ponder

Graph structures we are dealing with in this assignment is where a node have around 2 links to other nodes, it can be argued that the structure will be a line or in a circular structure. It cannot however, be in a tree, as that has 3 direct links. Since it will still be a linked list structure, the time complexity is O(n), The circular linked list will also have the complexity of O(n). So the time complexity for both will be O(n).