Graphs

Mahdi Nazari

October 2022

Introduction

In this work we will present a new data structure which is called graph. Graph is a general data structure that contains a number of vertices which are connected to each other by edges. Edges either can be directed or bidirectional between two vertices in the graph. Trees and linked lists are different types of graphs. In this work we will look at the Swedens railroad as a graph where each node in graph is a city. We will implement an algorithm to find the shortest path and enhance it with some improvements to make the algorithm more efficient.

Graphs

To implement railroad system as a graph we need to construct different objects, such as City, Connections and Map. City is defined as an object with a name and an array of Connections(neighbours). Connections is defined as a City and an integer distance in minutes. To be able to add a new connection to the connections array of a city an add method was implemented in class City, which looks like:

```
int pt = 0;
public void add(City dest, int distance){
        this.neighbors[pt++] = new Connection(dest, distance);
    }
```

The object Map was defined with an array of cities(hash table) and a constant value modulo which was set to 541. To be able to define constructor of the Map, we need methods such as hash and lookup. Hash method was implemented so that it takes a string as argument and returns a hashed value of that string. Lookup method takes a city name as argument and take the hash value of that name by hash function. Then it will check if the city exist in the cities array(which is a hash table) of the Map if so it will return that city otherwise it will add that city to the cities array of the Map and then return that city.

Constructor of the Map was implemented so that it takes a String file as argument. It creates a hash table with size of 541(mod value). The string file is ordered in a number of raws and each raw has a city as start point and another city as destination and an integer number which shows how long the journey between these cities will be in minutes. Constructor will go through each raw of the file and will call to lookup to add new cities to the cities array of the Map and then add new connection to the connections array of each city. When the Maps constructor is implemented, we can go through different cities in the railway system and find a path between two cities.

Shortest path

To find the shortest path between two cities in the railway system(graph) we will use depth-first searching. It means that if we want find the shortest path between city A and city M we will start at city A and if A has direct connections to the B and C we need to check each path that goes through B and C and continue until we either reach null or we find the destination M. Since the graph can have cyclical path the risk is huge that we end up in an infinite loop with this algorithm. Therefore we set a limit for searching, namely a max value, this means if program ends up in an infinite loop this limit will stop that loop after it passes the max value and program should prove another direct connection.

A new class Naive was implemented and to find the shortest path between two cities a method which is called ”shortest” was implemented in this class. This method takes two objects of type City as start point and destination point and an integer value as max value. For example if we want find the shortest path between Malmö and Göteborg with a max value of 200 min. We will start at Malmö and since it has just one direct connection which is Lund, we travel to Lund and this journey takes 13 minutes then we have just 200 - 13 = 187 min left. Lund has four direct connections which are Malmö, Hässleholm, Åstorp and Helsingborg. Now we need to check each of these four directions with a max value of 187 min towards Götebog. This will be done recursively and either we find a path from Lund to Göteborg which is less than 187 minutes or the max value will be too law to be able to reach Göteborg which means we need to increase max value.

A new method was implemented in the class Naive which will be called trip. This method will call to the shortest method for finding the shortest path between two cities meanwhile it calculate the time it takes for shortest method to find the shortest path. This method is actually a rewriting of the main method given in the instruction written by Johan Montelius.

```
private static void trip(String fr, String t, int m){
        Map map = new Map("trains.csv");
        long t0 = System.nanoTime();
        Integer dist = shortest(map.lookup(fr), map.lookup(t), m);
        long time = (System.nanoTime() - t0)/1_000_000;
        if(dist == null){ System.out.println("NOT FOUND! Increase the max!");}
        else{ System.out.println("shorest: " + dist + " min (" + time + " ms)"); }
    }
```

Benchmark

A benchmark was performed to find the shortest path between a number of cities with this algorithm that we have implemented and the result is as in the table below:

From	**To**	**Journey time**	**Runtime**
Malmö	Göteborg	153 min	3 ms
Göteborg	Stockholm	211 min	2 ms
Malmö	Stockholm	273 min	4 ms
Stockholm	Sundsvall	327 min	44 ms
Stockholm	Umeå	Not Found	gave up
Göteborg	Sundsvall	511 min	98982 ms
Sundsvall	Umeå	190	13 us
Umeå	Göteborg	705 min	10 ms
Göteborg	Umeå	Not Found	gave up

Table 1: Finding shortest path between two cities with Naive algorithm.

As the result shows finding the shortest path from Umeå to Göteborg was done in appropriate runtime but finding the shortest path from Göteborg to Umeå had a very long runtime and we gave up to wait. This makes us wondering about the efficiency of the algorithm that we have implemented. The reason that it takes much longer time to find the shortest path from Göteborg to Umeå than the opposite direction is cities around Göteborg have more direct connections to other cities and because we search for the shortest path recursively then we have many long branches in the graph that should be explored recursively and this means that program will end up in many loops inside each others. But Umeå has cities around that have fewer direct connections to other cities and this means that in recursive searching we have shorter branches in the graph that should be explored recursively.

Detect loops

We know that one of factors that make this algorithm inefficient is that the program needs to handle many loops. Many of those loops are because of the cyclical paths in the graph. For example for finding the shortest path between A to M, program starts at A and if A has direct connections to B and C, we know that even B and C have direct connections to A. With this current algorithm program needs to check each direct connections even though it goes back to a city that it was before. To prevent these loops we need to keep track of all cities that the program has traversed through.

A new class Path was implemented. Path was defined with an array of cities which will be large enough to store all cities in the graph. Since we know that there are 52 nodes in the graph(railway system) we create this array with size of 54(with some marginal). Now in the shortest method we store all cities that we go through in the path array. Each time we want go to a direct connection (city) we check first if we have already been in that city or not if we have been there we just return and check another direct connection. This improvement will prevent many useless loops. Now we can check if we can find the shortest path between those cities that we were not able to find with the previous algorithm. The result is as below:

From	**To**	**Journey time**	**Runtime**
Stockholm	Umeå	517	12 ms
Göteborg	Umeå	705	17 ms

Table 2: Finding shortest path between two cities with Path algorithm.

Since we know that with this algorithm program will not end up in an endless loop. The max value is not longer the limit for avoiding an infinite loop. Now we want find the shortest pash from Malmö to Kiruna with a max value of 10000.

From	**To**	**Journey time**	**Runtime**
Malmö	Kiruna	1162 min	800 ms

Table 3: Finding shortest path between two cities with Path algorithm and a high number of max.

One more improvement can be to avoid an initial value for the max. Max can have an initial value of null until the program finds a path. Then that path will be know the max value. Every time a new path is found we will check if it is smaller than the current max, if so then the max is this new path. With this improvement we want find the shortest path from Malmö to Kiruna and the result is as below:

From	To	Journey time	Runtime
Malmö	Kiruna	1162 min	91 ms

Table 4: Finding shortest path between two cities with Path algorithm and max initialized with null.

As the result shows with this improvement, runtime decreased from 800 ms to 91 ms.

Things to ponder

Time complexity of our algorithm is proportional to an exponential expression. For example if we had n nodes and each node in average had two direct connections the time complexity would be proportional to the O(2^n). An algorithm with an exponential time complexity is very inefficient when the number of nodes (n) increases.

Finding shortest path from Malmö to Athene would depends on how many nodes are there in the Europe railroad system. If the number of nodes in Europa railroad is 800 and each node has two direct connections to other nodes then the time complexity of finding the shortest path from Malmö to Athene will be proportional to 2^{800} which is a very large number therefore our algorithm will not be able to find the shortest path from Malmö to Athene in an appropriate time.