Algorithms and Data Structures

Spring 2019

Assignment 11

Date: May 09, 2019

Taiyr Begeyev

Problem 11.1 Shortest Path Algorithm

Your friend (who has not taken an "Algorithms and Data Structures" course) asks you for help on implementing an algorithm for finding the shortest path between two nodes u and v in a directed graph (possibly containing negative edge weights). The friend proposes the following algorithm:

1. **Add a large constant to each edge weight such that all weights become positive.**
2. **Run Dijkstra's algorithm for the shortest path from u to v**

Prove or disprove the correctness of the above algorithm to find the shortest path (note that in order to disprove, you only need to give a counterexample)

It may work, but not in all cases. Consider the following counterexample:

$$a \xrightarrow{-2} b \xrightarrow{6} c$$
$$a \xrightarrow{-2} b \xrightarrow{2} d \xrightarrow{3} c$$

The shortest path would be the **second** one and the weight is 3. Since the smallest number is -2 in the graph, let's add 2 to all vertices, so we can use Dijkstra algorithm.

$$a \xrightarrow{0} b \xrightarrow{8} c$$
$$a \xrightarrow{0} b \xrightarrow{4} d \xrightarrow{5} c$$

Now the shortest path is the **first** one, whose weight is 8. This happens when we add some constant to all edges. Consequently, the weight depends on the number of vertices. Therefore, this approach is not correct. The correctness disproved ✓

Problem 11.2 Optimal Meeting Point

You are trying to meet your friend who lives in a different city than you and you want to meet in a city that is between where either of you live. Time is limited and you are trying to minimize the time spent traveling. So, where exactly should you meet?
You are given a graph G with nodes $V = \{0, ..., n-1\}$ that represent the cities and edges E that represent streets connecting the cities directly. The edges are associated with the distance $d(e)$, which is the time needed to travel between two cities. You are given your own city p and your friend's city q with $p, q \in \{0, ..., n-1\}$.
Implement an algorithm that finds the target city m in which you and your friend should meet in order to minimize travel time for both of you (you drive towards your meeting city simultaneously, so if you travel for x minutes and your friend travels for y minutes, then you will want to minimize $max(x, y)$). The graph is given to you with an adjacency matrix, where each entry x_{ij} represents the time (in minutes) that it takes to travel from city i to city j. Naturally, the indices are the nodes. The algorithm should return the target city $m \in \{0, ..., n-1\}$.
The prototype of the corresponding function should be similar to:

int find_meetup_city(int[][] adj_matrix, int your_city, int friend_city);

```
/*
__________           .__
\__    ___/ _____    |__|  ___.__. _______
  |    |    \__  \   |  | <   |  | \_  __ \
  |    |     / __ \_ |  |  \___  |  |  | \/
  |____|    (____  / |__|  / ____|  |__|
                \/        \/
*/
#include <iostream>
#include <climits>
#include <algorithm>
using namespace std;
```

```
// Prototype Declaration
void generalInfo();
int find_meetup_city(int**, int, int, int);

int main() {
    int n;
    cout << "Enter the number of cities: ";
    cin >> n;

    // dynamically allocate 2d array nxn
    int** cities = new int*[n];
    for (int i = 0; i < n; i++) {
        cities[i] = new int[n];
    }

    cout << "Fill out the adjacency matrix " << n << "x" << n
    << " representing time between cities:" << endl;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cin >> cities[i][j];
            if (cities[i][j] == -1)
                cities[i][j] = INT_MAX;
            if (i == j)
                // distance to itself is obviously zero
                cities[i][j] = 0;
        }
    }

    int p, q;
    cout << "Enter your city: ";
    cin >> p;
    cout << "Enter your friend's city: ";
    cin >> q;
    // call the function to find the optimal meetup point
    cout << "The optimal meetup point: " << find_meetup_city(cities, n, p, q) << endl;

    // deallocate memory
    for (int i = 0; i < n; i++) {
        delete cities[i];
    }
    delete[] cities;

    return 0;
}

/*
    @brief finds the target city min which you and your friendshould
    meet in order to minimize travel time for both of you
    (you drive towards yourmeeting city simultaneously, so if you travel
    for x minutes and your friend travels for y minutes, then you will want
    to minimize max(x, y)). The graph is given to youwith an adjacency  matrix,
    where each entry xij represents the time (in minutes) that it takes to
    travel from city i to city j
*/

int find_meetup_city(int** cities, int number_of_cities, int your_city, int friend_city) {
    // initialize the solution matrix as the input graph as a first step
    int dist[number_of_cities][number_of_cities];
    for (int i = 0; i < number_of_cities; i++) {
```

```
        for (int j = 0; j < number_of_cities; j++) {
            dist[i][j] = cities[i][j];
        }
    }

    // Floyd Warshall Algorithm
    for (int k = 0; k < number_of_cities; k++) {
        for (int i = 0; i < number_of_cities; i++) {
            for (int j = 0; j < number_of_cities; j++) {
                // If vertex k is on the shortest path from
                // i to j, then update the value of dist[i][j]
                if (dist[i][k] + dist[k][j] < dist[i][j])
                    dist[i][j] = dist[i][k] + dist[k][j];
            }
        }
    }

    /* Finding the optimal meeting city */
    int res = INT_MAX;
    int city;
    for(int i = 0; i < number_of_cities; i++) {
        if (res > max(dist[your_city][i], dist[friend_city][i])) {
            res = max(dist[your_city][i], dist[friend_city][i]);
            city = i;
        }
    }
    return city;
}
```

Problem 11.3 Number Maze

Consider a puzzle that consists of a $n{\times}n$ grid where each field contains a value $x_{ij} \in N$. Our player starts in the top left corner of the grid. The goal of the game is to reach the bottom right corner with the player. Rules of the game: On each turn, you may move your player up, down, left or right. The distance by which the player moves in a chosen direction is given by the number of its current cell. You must stay within the board (you cannot go off the edge of the board). Example: If your player is on a square with value 3, then you may move either three steps up, down, left or right (as long as you do not leave the board).

(a) Represent the problem as a graph problem. Formalize it by determining what is represented as the nodes and as the edges.

It is a simple graph shortest path finding problem. Mathematically, $G = (V, E)$, where $V = \{0, 1, 2, ...n^2 - 1\}$, $E = (v * M)$, $v \in V$ and $M = G.Adj[v]$.

(b) make

(c) make

Reference:

1. Wikipedia contributors, "Floyd–Warshall algorithm," Wikipedia, The Free Encyclopedia, https://en.wikipedia.org/wiki/Floyd%E2%80%93Warshall_algorithm (accessed May 9, 2019).