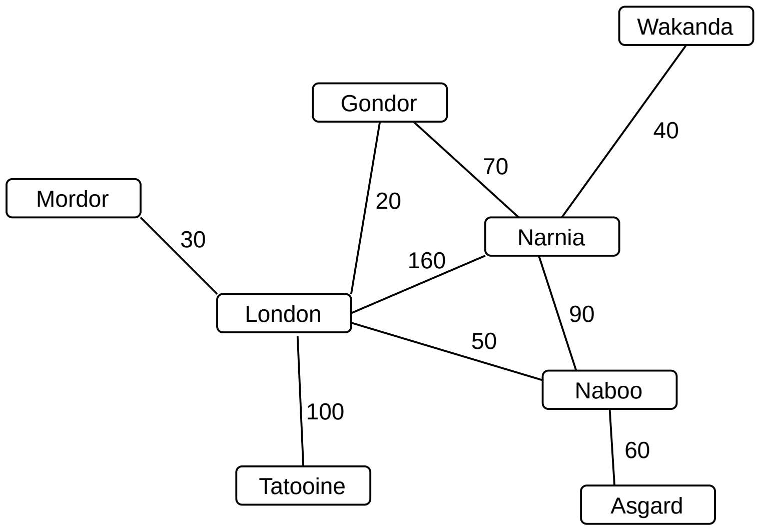
COMP 201 -Computer Systems and Programming Assignment 3(Due Date: 27.11.2020)

Contact TA: Muhammad Aditya Sasongko
msasongko17@ku.edu.tr

In this asignment, you are expected to find the shortest path between two cities in a given road network using the **Floyd-Warshall** algorithm.

Each road in this road network connects exactly two cities, and the roads in the network have various lengths. This road network can be considered as an undirected graph since each road can be used in either direction, and has the same length in both directions. Between any two cities, there exist at least one path that connects them, though this path might not be exactly a road that directly connects the two cities.

You can see an example of such a road network.



Given the road network above, if the question is

What is shortest path between Mordor and Narnia, and how long is it?

the answer is:

Mordor - London - Gondor - Narnia
Length: 120

1. Program Description

- Your program reads the road network information from a file whose name is provided as a command line argument to your program. The file name is passed to the program as follow.

./your_program input_file.txt

- The input file for your program consists of lines, each of which describes a road that connects any two cities. Each line has three entries, and the entries are separated by blank spaces. The first and second entries are the names of the cities connected by the road, and the third entry shows the length of the road in kilometer.

  Here is an example of a file that can be an input to your program. This program is based on the figure of road network depicted above.

```
Mordor London 30
London Tatooine 100
London Gondor 20
London Narnia 160
London Naboo 50
Gondor Narnia 70
Narnia Naboo 90
Naboo Asgard 60
Narnia Wakanda 40
```

Listing 1: example of input file

  The file will not include the number of lines that it contains.

- After reading the input file, the program will prompt its user to enter the names of two cities whose shortest distance and path are to be searched.

  Program: Enter the cities:

  User: Mordor Narnia

  The names of the cities are separated by a whitespace blank.

- If a shortest path is found between the two cities, the output is printed in two lines. The first line shows the sequence of cities in the path, and the second line displays the length of the path in kilometer. If there are more than one shortest path, any one of them can be a valid output. For the problem in the example, the output is:

  Mordor London Gondor Narnia
  120

  If it was found that there exists no path, the output is a single line which is as follow.

  *** NO PATH ***

- To find the shortest path in the problem, you are expected to utilize **Floyd-Warshall** algorithm. This algorithm finds and calculates the length of the shortest paths between all pairs of vertices in a graph. In your code, this algorithm is executed only once. However, the shortest paths that it finds can be queried repeatedly by a entering city names in the command prompt. The algorithm is used in your code as follow.

```
cities // array of vertices; each array element contains a city name and
    its index becomes the city's numeric id
roads // array of edges; each array element contains the ids of two
    cities connected directy by a road and the length of the road
city_graph // a two-dimensional array that shows the length of the
    shortest path between any two cities
shortest_paths // a two-dimensional array that shows the direction to the
     shortest path between any two cities

void floydWarshall() {
  for (int k = 0; k < cities.city_count; k++) {
  for (int i = 0; i < cities.city_count; i++) {
    for (int j = 0; j < cities.city_count; j++) {
    if ((city_graph[i][k] == INF) || (city_graph[k][j] == INF))  {
      continue;
    }
    if (city_graph[i][j] > (city_graph[i][k] + city_graph[k][j])) {
      city_graph[i][j] = city_graph[i][k] + city_graph[k][j];
      shortest_paths[i][j] = shortest_paths[i][k];
    }
    }
  }
  }
}

int main(int argc, char *argv[]) {
  // Allocate memory regions dynamically to cities array and roads array.
  // Read and parse the input file. Insert the city names and ids to
    cities array.
  // Insert city ids and road lengths to roads array.
  // Allocate memory regions dynamically to city_graph, distance, and
    shortest_paths arrays.
  // Initialize the values in city_graph array with road lengths, such
    that the value in city_graph[i][j] is the road length between cities
    i and j if these cities are directly connected by a road. For cities
    m and n that are not connected directly by a road, the value in
    city_graph[m][n] will be INF, which is a large value like 10M, that
    is assumed to be infinite.
  initialise();
  floydWarshall();
  while(1) {
    // prompt user to enter two city names
    // print the shortest path between the two cities
    // print the length of the path
  }
  return 0;
}
```

Listing 2: FloydWarshall Algorithm

- A template code is provided that will help you create your final program. Please read the comments in the template code carefully before working on your code.

- You are allowed to use **ONLY** dynamic memory allocation to create any array in the code.
- When a dynamically allocated array is no longer used, the memory regions allocated for it have to be **freed**.

2. Work Instruction

1. Accept the assignment link: https://classroom.github.com/a/g_YOXrS3, and open the generated private Github repository.
2. Clone the repository locally to your machine or import it to REPL.it.
3. Work on the floyd_warshall.c template code by modifying the `insert_to_cities`, `printPath`, and `main` functions to add the missing parts that are necessary to make the code work properly.
4. To ensure that your code works correctly, you can use the provided `input.txt` file as an input to your program, and compare your output with the content of the `expected-output.txt` file. In the `expected-output.txt` file, there is a list of inputs and outputs that your code is supposed to produce when the input file is the `input.txt` file.
5. To ensure that all dynamically allocated memory regions have been freed by the end of your code, you can use Valgrind to check if there is still unfreed allocated memory.
6. After you finish working on your code, commit and push your code to your repository.

3. Evauation Criteria

You will be graded out of 50 maximal points. The total grade comprises the following criteria.

- 25 points for code correctness

  This criterion covers correctness of printed output, and the code being free from syntax errors, segmentation faults, stack smashing, infinite loop/recursion, and other logical errors. Please make sure that your code can handle input files that have **much higher** number of cities and roads than the provided `input.txt` file.

- 10 points for using only dynamic memory allocations for creating arrays

  You must not declare arrays statically such as "int array1[100]", but, instead, use functions like `malloc`, `calloc`, and `realloc` in allocating memory regions for the arrays in your code.

- 10 points for freeing all dynamically allocated memory before the end of your code
- 5 points for coding style and comments

4. Oral Assessment

Important Note: We use automated plagiarism detection to compare your assignment submission with others and also the code repositories on GitHub and similar sites. Moreover, we plan to ask randomly selected 10% of students to explain their code verbally after the assignments are graded. And one may lose full credit if he or she fails from this oral part.

5. Late Submission Policy

- You may use up to 5 grace days (in total) over the course of the semester for the assignments. That is you can submit your solutions without any penalty if you have free grace days left.
- Any additional unapproved late submission will be punished (1 day late: 20% off, 2 days late: 40% off) and no submission after 2 days will be accepted.

6. Coding Style Evaluation

We have reserved 5 points for a subjective evaluation of the style of your solutions and your commenting. Your solutions should be as clean and straightforward as possible. Your comments should be informative, but they need not be extensive.

7. Academic Integrity

All work on assignments must be done individually unless stated otherwise. You are encouraged to discuss with your classmates about the given assignments, but these discussions should be carried out in an abstract way. That is, discussions related to a particular solution to a specific problem (either in actual code or in the pseudocode) will not be tolerated. In short, turning in someone else's work, in whole or in part, as your own will be considered as a violation of academic integrity. Please note that the former condition also holds for the material found on the web as everything on the web has been written by someone else. See Koç University - Student Code of Conduct.

8. Acknowledgement

This assignment is adapted from Middle East Technical University Course C programming assignment.