COMP2112 Data Structures and Algorithms
Fall 2023, Project 3
Due Date: January 14 2024, 23:55

IŞIK UNIVERSITY
COMPUTER SCIENCE AND ENGINEERING

In this project we will model the connections between some fictional cities. The data is available on the blackboard (graph.txt).

Our weighted graph will represent this by edge-weights equal to a fictional distance value among some fictional cities.

Remember that the city names are String. So, you need to use a hashtable to map the city names into graph vertices: converting strings to integers as vertex numbers.

More explanation on Hashing:

Implement a method or a piece of code using hashing techniques which stores some strings as names (e.g., "ece", "araz", "ali", "sara", …) in an array after generating a hashcode (index) for each name. For example, if hashcode ("sara") = 8, it should be stored in index 8. If index 8 is already full, collision management techniques should be used. The length of hash table (array) is up to you, but do not choose it too big. The initial list of names should be read from an input file that contains 353 lines. In this file, each line contains two names as vertices, and an integer value as edge weight between two vertices. Your program should also be able to search a given string in the array in constant time using hashing techniques (not linear search) and return the index of array containing this name, or -1 if the name does not exist in the array.

Note: We will not use a different input file to test your code but we will enter different input values as parameters of the methods that you are asked to implement (given below).

ReadGraphFromFile(): loads graph data a from the file into an adjacency matrix. Before constructing this matrix, a hash table should be also created.

IsThereAPath(String v1, String v2): Returns true if there is a path between vertex v1 and vertex v2 or false, otherwise.

BFSfromTo(String v1, String v2): prints the sequence of vertices (names of the vertices) and edges (weights of the edges) between the vertices while starting a BFS from v1 until reaching v2.

DFSfromTo(String v1, String v2): prints the sequence of vertices (names of the vertices) and edges (weights of the edges) between while starting a DFS from v1 until reaching v2.

WhatIsShortestPathLength(String v1, String v2): returns the length of shortest path from vertex v1 to vertex v2. It prints "v1 --x-- v2" if there is no path from v1 to v2.

NumberOfSimplePaths(String v1, String v2): returns the number of paths from v1 to v2.

Neighbors(String v1): returns the names of the neighbor vertices of v1.

HighestDegree(): returns the name of the vertex with the highest degree. If there is more then one, it returns the names of all.

IsDirected(): returns true if the graph is directed, otherwise false.

AreTheyAdjacent(String v1, String v2): returns true if v1 and v2 are adjacent, otherwise false.

IsThereACycle(String v1): returns true if there is a cycle path which starts and ends on v1, otherwise false.

NumberOfVerticesInComponent(String v1): prints the number of vertices that exist in the component that contains name1.

Main method: in this method, prepare a menu to select any of the methods mentioned above. For each method, the input parameters should be received from the user interactively and the result of the method must be printed to the console.

NOTE 1: In all the methods mentioned above, no vertex can be repeated in the same path.

NOTE 2: In BFS, the edge order while visiting the children of a vertex, starts from edge with minimum weight and continues to the edge with maximum weight

NOTE 3: For the graph representation and algorithms of this project, use your own implementations and ideas. Any usages of premade-classes, libraries or known algorithms will be graded as 0. For example, if you use any graph related class in Java, your grade will be 0. Another example, for the solution of ***WhatIsShortestPathLength*** method, if you use any known solution such as Djisktra's algorithm, your grade will be 0. Please use your own ideas, solutions and implementations. For your overall project, this rule is **not applied only for the DFS and BFS** functions.

NOTE 4: File structure is different than what you saw in the lecture. The number of the vertices and the edges are not specified in the file. You can calculate them if you think that you need them. The content of the file is similar to the adjacency list representation. Each row of the file defines the edges for a single vertex. For example, the first for of the file defines the edges of vertex A. In each row, after the "->" sign, edges are listed and separated with ",". Each edge definition consists of two elements, separated with ":". The first elements is the end-point and the second element is the weight of the edge. For example, the first edge that is defined for the vertex A is "B: 3". This means there is an edge from A to B of weight 3. This different file structure means, the graph generation code that you saw in the lecture is no use to you as it is, yet it can give you an idea about how to read the file. Feel free to modify that code or to implement your own idea. Please note that you are not allowed to use anything other than basic string operations such as ***split***. (For the string operations you are allowed to use basic methods of the String class in Java)

NOTE 5: The final due date of the project is **January 14 2024, 23:55**. There will be no extension.

This is a **group** assignment. That means, you are **only** allowed to study with your designated groupmate. You are **not** allowed to take a peek at any solutions, including **online resources**, and you are not allowed to share your answers with **anyone**, including your classmates. You are only allowed to use your lecture notes and the textbook. Failure to follow this rule will result in an F for the course grade, in the best case.

Compress your .java and upload your java files through BlackBoard.
