Lab10

Lab10

EE 599 Lab10
Spring 2019 Nazarian **80 + extra 80 points**

Student ID: ___________________ **Name: ____________________**

Last Update: 4/6 @ 5:56pm:

Assigned: Saturday March 30, 2019

Due: Wednesday April 10 at 11:59pm (the majority on Sat April 6 class asked for extension. No extra credit for early submission.

Late submissions will be accepted for two days after the deadline with a maximum penalty of 15% per day. For each day, submissions between 12am and 1am: 2% penalty; between 1 and 2am: 4%; 2-3am: 8%; and after 3am: 15%. No submissions accepted after Friday at 12 am.

Notes:

This assignment is based on individual work. No collaboration is allowed (Discussing the technical details of each problem with other students before submitting your work, copying from students of current or previous semesters is not permitted in our lab and HW assignments). You may post your questions on the discussion forums and/or use the office hours. We may pick some students in random to demonstrate their design and simulations. Please refer to the syllabus and the first lecture regarding the USC and our policies, including the penalties for any violation. If you have any doubts about what is allowed or prohibited in this course, please contact the instructor.

What You Will Practice

In this lab, you will practice graph related algorithms.

Part I is mandatory.

Parts II and III are extra credit, i.e., skipping them would not negatively impact your grade.

Background

A graph G is simply a way of encoding pairwise relationships among a set of objects: it consists of a collection V of *nodes* and a collection E of *edges*, each of which “joins” two of the nodes. We thus represent an edge $e \in E$ as a two-element subset of V: $e = \{u, v\}$ for some $u, v \in V$, where we call u and v the *ends* of e.

Graphs can be used to model many types of relations and processes in physical, biological, social and information systems. Many practical problems can be represented by graphs.

For example, graphs can be used to model social network problems. Given any collection of people who interact (the employees of a company, the students in a high school, or the residents of a small town), we can define an undirected graph where the graph's vertices represent people, with an edge joining u and v if they are friends with one another. We could have the edges mean a number of different things instead of friendship: the undirected edge (u, v) could mean that u and v have had a romantic relationship or a financial relationship; the directed edge (u, v) could mean that u seeks advice from$v$, or that u lists v in his or her e-mail address book.

Part I - Graph Valid Tree Checker (80pts)

Problem Description

We say that an undirected graph is a *tree* if it is connected and does not contain a cycle. Trees are the simplest kind of connected graphs. Note that deleting any edge from a tree will disconnect it. Trees are fundamental objects in computer science, because they encode the notion of a hierarchy. For example, consider a company, where the graph's vertices represent employee, and the graph's edges represent the reporting relationships. We can imagine the tree in Figure 1 as corresponding to the organizational structure of a tiny nine-person company; employees 3 and 4 report to employee 2; employees 2, 5, and 7 report to employee 1; and so on.

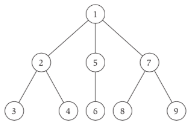

Figure 1

To check if an undirected graph is a tree, we need to check to properties, 1) there is no cycle, and 2) the graph is connected.

Task:

Given n nodes labeled from 0 to n-1 and a list of undirected edges (each edge is a pair of nodes), design a checker called the Graph Valid Tree Checker, which examines whether the graph is a tree. Hint: For an undirected graph we can either use BFS or DFS to detect above two properties.

a. Your compiled C/C++ program should be able to read a graph described in an external text file and check if this given undirected graph is a tree. The file name of the graph should be an input parameter of your complied program. Suppose your compiled C/C++ program is a.out. A command of running your program can be:

"./a.out input1.txt",

where "input1" is the name of the file that describes a graph.

b. The external file should describe the graph in the above example as follows:

```
5

0,1, 0,2, 0,3, 1,4
```

Where the first line is the number of nodes, the second line is the edges. For example, there are four edges in the given graph, i.e., edge from node 0 to node 1, edge from node 0 to node 2, and so on. You can assume that no duplicate edges will appear in edges. Since all edges are undirected, 0,1 is the same as 1,0 and thus will not appear together in edges.

c. Print "Graph is a valid tree" if the given graph is a valid tree. Otherwise, print "Graph is not a valid tree".

d. Write a Python code that automatically generates 10 randomly generated undirected graph samples. Some of them could be trees, some could be forests, and some connected with cycles. The sizes of graphs are yours to choose. Please, in your readme file, briefly explain your idea of random generation of graphs. Please add a system call in your Python to also call the Graph Valid Checker you had implemented earlier on each one of the 10 graphs.

Part II – Find shortest path using Dynamic Programming (Extra 40pts)

Problem Description

Graphs are often used to model networks in which one travels from one point to another, e.g., traversing a sequence of highways through interchanges, or traversing a sequence of communication links through intermediate routers. As a result, a basic algorithmic problem is to determine the shortest path between nodes in a graph. For example, a computer network can be modeled as a weighted directed graph (digraph) G=(V,E) where V is the set of nodes representing the routers and computers, and E is the set of directed edges representing the links that connect different nodes. A length metric can be assigned to each edge to represent the transmission delay, failing probability, monetary cost of each link, etc.. Thus, the routing decision is indeed the problem of finding a shortest path in a graph, which is a classical problem in algorithm and graph theory.

Tasks:

Given a graph and a source vertex 0 in graph, use Bellman-Ford algorithm to find shortest paths from 0 to all vertices in the given graph. The graph may contain negative weight edges.

a. Your compiled C/C++ program should be able to read a graph described in an external text file and find the shortest paths from node 0 to all the other nodes. The file name of the graph should be an input parameter of your complied program. Suppose your compiled C/C++ program is a.out. A command of running your program can be:

"./a.out input2.txt",

where "input2" is the name of the file that describes a graph.

b. The external file should describe the graph in the above example as follows:

```
0,1,*,2,*

*,0,4,2,*

*,*,0,*,2

*,*,4,0,2

*,*,*,*,0
```

where the first line is the lengths of the edges from node 0 to nodes 0-4, the second line is the lengths of the edges from node 1 to nodes 0-4; and so on so forth. Note that by conversion, the length from one node to itself is zero. We use * to represent that there is no edge from one node to another, i.e., the length is infinity.

c. Your code should be aware of negative loops.

d. The output of your program should also be a file, where the first line should output the distance from node 0 to all other nodes; the second line should be the shortest path from node 0 to node 0, which is trivial; the third line should be the shortest path from node 0 to node 1; and so on so forth. For example, if the output of your program for the above example should be:

```
0,1,5,2,4
0
0→1
0→1→2
0→3
0→3→4
```

If there is a negative loop:

Output: Negative Loop Detected

Part III – Find the maximum clique of a graph (Extra 40pts)

Problem Description

The clique problem has many real-world applications. E.g., consider a social network, where the graph's vertices represent people, and the graph's edges represent mutual acquaintance. Then a clique represents a subset of people who all know each other, and algorithms for finding cliques can be used to discover these groups of mutual friends.

Given an undirected graph G = (V, E), a clique S is a subset of V such that for any two elements u, v ∈ S, (u, v) ∈ E. A **maximum clique** of a graph, *G*, is a clique, such that there is no clique with more vertices. Moreover, the **clique number** ω(*G*) of a graph *G* is the number of vertices in a maximum clique in *G*. Finding the maximum clique in a graph is an NP-hard problem, called the maximum clique problem (MCP).

Tasks:

Find the maximum clique in a given graph.

a. Your compiled C/C++ program should be able to read a graph described in an external text file and find the maximum clique in the graph. The file name of the graph should be an input parameter of your complied program. Suppose your compiled C/C++ program is a.out. A command of running your program can be:

"./a.out input3.txt"

where "input3" is the name of the file that describes a graph.

b. The external file should describe the graph in the above example as follows:

```
5
0 1 0 0 1
1 0 1 0 1
0 1 0 1 0
0 0 1 0 1
1 1 0 1 0
```

Where the first line is the number of nodes, the first line is the edges from node 2 to nodes 1-5, the second line is the edges from node 2 to nodes 1-5; and so on so forth. The graph given in this part is unweighted. We use "1" to represent there is an edge and use "0" to represent that there is no edge from one node to another.
The output the clique number and the vectors in the maximum clique. For example, if the output of your program for the above example should be:

```
Clique number (3): 1 2 5
```

Submission

Please push all files you wrote into the GitHub repo.
https://classroom.github.com/a/6lkIQVTz
Don’t forget to submit a readme file:

- **Your readme file should be named as readme_<STUDENT-ID>.txt, please replace <STUDENT-ID> with your own 10-digit USC-ID.**
- Any non-working part should be clearly stated.
- The citations should be done carefully and clearly, e.g.: “to write my code, lines 27 to 65, I used the Djkstra's shortest path algorithm C++ code from the following website: www.SampleWebsite.com/...”

Example input files

Attached Files: input1.txt (21 B)
input2.txt (53 B)
input3.txt (53 B)

Discussion slide

Attached Files: discussion_Lab10.pdf (1.133 MB)