Homework 1: DFS and Shortest Paths

Handed out Thu, Sep 14. Due Fri, Sep 22. (Submission instructions will be forthcoming.)

Problem 1. In this problem you will simulate the strong-components algorithm given in class on the digraph shown in Fig. 1.

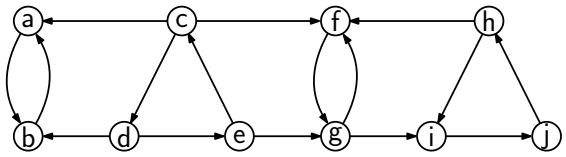

Figure 1: Problem 1.

(a) Draw the reverse graph G^R. (Be careful to note the directions of the edges.)

(b) Show the result of applying DFS to G^R. Label each vertex u with its discovery and finish times ($d[u]/f[u]$). You need only show the *final* DFS tree, not the intermediate results.

To make the grader's life easier please draw your DFS forest in the same manner as in Fig. 2(b) of Lecture 3[1] Also, whenever you have a choice of which vertex to visit next, select the lowest vertex in alphabetic order.

(c) Show the result of running DFS to G, where the main program selects the next vertex to visit based on decreasing order of finish times from part (b).

(d) Illustrate (e.g., by circling them or listing them out) the strong components of G.

Problem 2. In this problem you will simulate the bridge-detection algorithm given in class on the graph shown in Fig. 2.

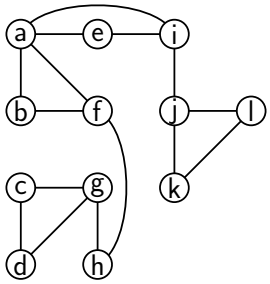

Figure 2: Problem 2.

[1] See `http://cs.umd.edu/class/fall2017/cmsc451-0101/Lects/lect03-strong-components.pdf`.

(a) Show the result of applying DFS to G. Label each vertex u with its discovery time and Low value ($d[u] : \text{Low}[u]$). You need only show the *final* DFS tree, not the intermediate results.

To make the grader's life easier please draw your DFS forest in the same manner as in Fig. 4(b) of Lecture 4[2] Also, whenever you have a choice of which vertex to visit next, select the lowest vertex in alphabetic order.

(b) Indicate which edges are bridges, and illustrate (e.g., by circling them or listing them out) the 2-edge connected components of G.

Problem 3. Recall that a vertex in an undirected graph is a *cut vertex* if its removal (along with any incident edges) causes the graph to become disconnected. In this problem we will (at a high level) adapt the algorithm for computing bridges to compute cut vertices, by describing the conditions under which a vertex is a cut vertex.

Let $G = (V, E)$ be a connected undirected graph, and suppose that you run DFS on G and (as in Lecture 4) you compute the value of $\text{Low}[u]$ for every vertex $u \in V$. Prove each of the following assertions:

(a) The root of the DFS tree is a cut vertex if and only if it has two or more children.

(b) No leaf of the DFS tree can be a cut vertex.

(c) A non-root, internal vertex u of the DFS tree is a cut vertex if and only if it has a child v such that $\text{Low}[v] \geq d[u]$. (Intuitively, no back edge of v's subtree goes to a vertex higher than u.)

Hint: Recall that in proving "A if and only if B", you need to show that "A implies B" and "B implies A".

Warning: The answer can be found on the Web or in text books, but I would encourage you to think about it on your own. If you do make use of any external sources in answering your question, you will not be penalized provided that your answer is correct and is stated in your own words. But you *must* cite whatever source(s) you use.

Problem 4. You are given a directed graph $G = (V, E)$ with (possibly negative) weighted edges but no negative cost cycles, in which the shortest path between any two vertices is guaranteed to have at most k edges. Give an algorithm that finds the shortest path between two vertices u and v in $O(km)$ time, where $m = |E|$. (Prove that your algorithm is correct and derive its running time. Prove correctness from first principles. You may refer to the proof presented in lecture notes, but please state the proof in your own words.)

Challenge Problem. Challenge problems count for extra credit points. These additional points are factored in only after the final cutoffs have been set, and can only increase your final grade.

Present an efficient algorithm that, given a connected, undirected graph $G = (V, E)$, determines whether it is possible to assign directions to the edges so that the resulting digraph

[2] See `http://cs.umd.edu/class/fall2017/cmsc451-0101/Lects/lect04-edge-connectivity.pdf`.

has no sources. If this is not possible, your algorithm should indicate this. If it is possible, your algorithm should provide such an orientation of the edges.

Recall that a vertex of a digraph is a *source* if it has no incoming edges. In other words, every vertex of your digraph should have at least one incoming edge (see Fig. 3).

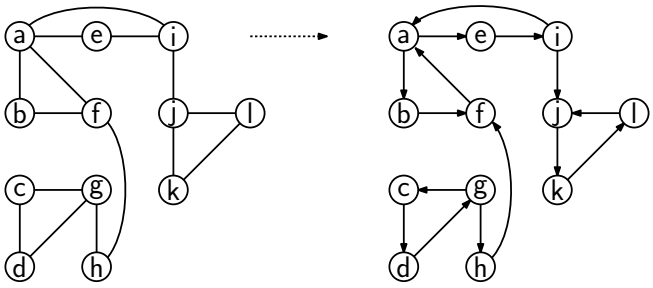

Figure 3: Challenge Problem.

Prove your algorithm's correctness and derive its running time. (Hint: This can be done in $O(n + m)$ time.)