CS4102 - ALGORITHMS - SPRING 2022

DUE FRIDAY, MARCH 18, 2022 AT **11:30 pm** VIA GRADESCOPE B1 BASIC (WYZ5RGE)

Collaboration Policy: You are encouraged to collaborate with up to 3 other students, but all work submitted must be your own *independently* written solution. List the computing ids of all of your collaborators in the `collabs` command at the top of the tex file. Do not share written notes, documents (including Google docs, Overleaf docs, discussion notes, PDFs), or code. Do not seek published or online solutions for any assignments. If you use any published or online resources (which may not include solutions) when completing this assignment, be sure to cite by naming the book etc. or listing a website's URL. Do not submit a solution that you are unable to explain orally to a member of the course staff. Any solutions that share similar text/code will be considered in breach of this policy. Please refer to the syllabus for a complete description of the collaboration policy.

Collaborators: jhs8cue, kd5eyn, spj6s

Sources: Cormen, et al, Introduction to Algorithms. *(add others here)*

PROBLEM 1 *Short Questions on BFS*

A. What is the maximum number of vertices that can be on the queue at one time in a BFS search? Briefly explain the situation that would cause this number to be on the queue.

Solution: For a graph G with v nodes, a maximum of $v - 1$ vertices will be on the queue at one time. This situation would occur if starting vertex s is adjacent to all other vertices in the G

B. If you draw the BFS tree for an undirected graph G, some of the edges in G will not be part of the tree. Explain why it's not possible for one of these non-tree edges to connect two vertices that have a difference of depth that's greater than 1 in the tree.

Solution: Due to the "fanning out" nature of BFS, and BFS will process vertices of equal distance from starting vertex s. Assuming G is unweighted, if two vertices have a difference of depth greater than 1 in the BFS tree, then not only are they different distances from s, but they are also not adjacent. Because BFS finds the shortest path from s to all other vertices in unweighted G, BFS cannot connect two such vertices because no direct path exists between the two.

PROBLEM 2 *True or False. (You don't have to explain this in your submission, but you should understand the reason behind your answer.)*

A. If you use BFS to detect a cycle in an undirected graph, an edge that connects to a vertex that's currently on the queue or has been removed from the queue indicates a cycle as long as that vertex is not the parent of the current node.

Solution: True

B. If you use DFS-visit on a *directed* graph with $V > 1$ starting at vertex v_1, you will always visit the same number of vertices that you would if you started at another node v_2.

Solution: False

C. If you use DFS-visit on a connected *undirected* graph with $V > 1$ starting at vertex v_1, you will always visit the same number of vertices that you would if you started at another node v_2. (If it were not connected, would your answer change?)

Solution: True

PROBLEM 3 *Finding Cycles Using DFS*

In a few sentences, explain how to recognize a directed graph has a cycle in the DFS-visit algorithm's code we saw in class. How does this need to be modified if the graph is undirected?

Solution: The algorithm we discussed in class for DFS involved marking vertices as seen (grey) or processed (black). I.e., a vertex that has been visited is marked grey, and once all of its adjacent vertices (or children in the case of a digraph) are marked grey, it is marked black. When running DFS, if a vertex were marked grey (placed on the queue) and it points to another vertex already grey, that means that the vertex it points to is also on the queue. The only way this could happen is if one vertex is the ancestor of another, which would create a cycle.

PROBLEM 4 *Finding a path between two vertices*

Describe the modifications you would make to DFS-visit() given in class to allow it to find a path from a start node s to a target node t. The function should stop the search when it finds the target and return the path from s to t.

Solution: While building the DFS tree, maintain a separate stack. Every time the algorithm traverses a vertex, push it onto the stack. When a dead end is reached and DFS needs to back up, pop the vertex off the stack. Repeat this process of pushing and popping as DFS traverses until the target vertex t is found. Push the target onto the stack and end DFS searching. The path from s to t will be stored in the stack in reverse order, starting from the top. If we want the path to be in the correct order, then we can set up a while loop to pop each vertex off the stack into an array, or even another stack.

PROBLEM 5 *Labeling Nodes in a Connected Components*

In a few sentences, explain how you'd modify the DFS functions taught in class to assign a value $v.cc$ to each vertex v in an undirected graph G so that all vertices in the same connected component have the same cc values. Also, count the number of connected components in G.
In addition to your explanation, give the order-class of the time-complexity of your algorithm.

Solution: The approach to this problem is similar to the algorithm we discussed in class for finding SCCs in a digraph. First call DFS-Sweep on G, starting at a given vertex to find the discovery and finish times for each vertex. Instantiate an integer for cc and initialize it to 0. Instantiate a list as well. Call DFS on transpose G^T, starting at the vertex with the highest finish time, assigning cc to each node, until DFS reaches a seen node, indicating a cycle. Every time a vertex is assigned cc, increment the value of index cc in the list we created. After a cycle/SCC has been visited, increment cc and move DFS on to the vertex with the next highest finish time that hasn't been visited yet. Continue this process until no more unvisited vertices remain. We are left with a graph of vertices that all have a number assigned to them that indicates they are a member of an SCC with a corresponding number for identification, as well as a list of counts whose indices correspond to the SCC of the same number.

PROBLEM 6 *BFS and DFS Trees*

Consider the BFS tree T_B and the DFS tree T_D for the same graph G and same starting vertex s. In a few sentences, clearly explain why for every vertex v in G, the depth of v in the BFS tree cannot be greater than its depth in the DFS tree. That is:

$$\forall\, v \in G.V, depth(T_B, v) \leq depth(T_D, v)$$

(Here the depth of a node is the number of edges from the node to the tree's root node. Also, you can use properties of BFS and DFS that you've been taught in class. We're not asking you to prove those properties.)

Solution: A node in a BFS tree cannot have a depth greater than the same node in a DFS tree due to the "fanning out" behavior of BFS. Since BFS visits vertices in G of the same degree of adjacency from starting vertex s, the tree created by BFS will be wider than it is tall, whereas DFS traverses as far away from s as possible, only backing up as little as possible when encountering dead ends, which will create a tree taller than it is wide.

PROBLEM 7 *Gradescope Submission*

Submit a version of this `.tex` file to Gradescope with your solutions added, along with the compiled PDF. You should only submit your `.pdf` and `.tex` files.