CS4102 - ALGORITHMS - SPRING 2022

DUE FRIDAY, MARCH 18, 2022 AT **11:30 pm** VIA GRADESCOPE B1 BASIC(RTG5XKH)

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:

Sources: Cormen, et al, Introduction to Algorithms.

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: V - 1. If the start vertex has a path to every other vertex in the graph, each of those remaining vertices would be marked "gray" and added to the queue. Only once the start vertex is marked black would those remaining vertices start getting dequeued.

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: If any edge connects two vertices in an undirected, unweighted graph, then that edge represents the minimum distance between those two vertices (distance = 1 edge). Thus, if such an edge exists between any two vertices in the graph, when BFS is executed it will find that edge and represent it in the tree by putting one of the vertices at some depth and putting the other vertex at a depth adjacent to the first vertex.

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
If a vertex has already been seen, it is on the queue. if it can be seen through multiple paths in the graph, there is a cycle

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
If a v1 has an edge to v2 and v2 does not have an edge to v1, then DFS-visit(v1) will visit at least one more vertex than DFS-visit(v2) will.

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
Since the graph is connected and undirected, each vertex will have access to every other vertex in the graph through some path, thus no matter where we start DFS-visit will always visit every vertex.

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: When performing DFS-visit, we can add a check to see if, at any time in the algorithm's search, it visits a vertex that it has previously seen and that is not yet finished with its DFS-visit (thus the vertex will be marked gray). We can do this by adding a check in the for loop:

```
if v.color == GRAY:
    cycle = True
```

For an undirected graph, it would work similarly, however we would also need to check that the gray vertex the algorithm might come across is not the parent of the vertex DFS-visit was just called on. The code within the for loop would look like this:

```
if v.color == GRAY and u.π != v:
    cycle = True
```

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: First, I would store a current path as a list of vertices. In each call to DFS-visit, vertex u will be added to the path. this path will be passed as another parameter of DFS-visit to ensure that it is maintained through the recursive calls. in the for loop, I would check if v == target. If so, I would return the current path list and terminate the search.
If the for loop completes without the target being found, I would remove the last vertex of the current path list (at the same time as when the vertex u is set to BLACK). This will ensure that the path is accurate even when the search must traverse multiple adjacent paths that do not contain the target vertex.

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: In DFS-Sweep, I would initialize a variable 'count' at the very beginning to keep track of the number of strongly connected components. Count will be used to store both the number

of strongly connected components and to provide indices for the cc values. In the second for loop of DFS-sweep (the one that iterates through the unseen vertices and calls DFS-Visit), I would set u.cc = count, and I would also pass count into DFS-Visit. Any vertex v that DFS-Visit comes across here should also get cc set, so v.cc = count within DFS-Visit's for loop. After the for loop in DFS-sweep completes, count should be incremented. This way, each vertex in the graph will get cc correctly set and the total number of SCCs will be stored in count after the function completes. This is because each iteration of DFS-Sweep's for loop represents a new SCC, and we increment count at each of these iterations.

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: In the BFS tree, the depth difference for any 2 vertices is minimized. That is, the depth difference of any 2 vertices in the BFS tree will represent the minimum number of edges in a path between them in the graph. A DFS graph will make a path out as far as it can, before backtracking and making more paths that extend as far as the graph "reaches". Thus, the DFS tree may or may not, but often will not, represent minimum distances between vertices in the graph. And it could never contain a depth of a vertex in the tree that is greater the depth of that vertex in the BFS tree because if it did, the tree would simply be incorrect. If there existed a connection between a given vertex in the BFS tree such that the vertex could have been placed higher, the BFS tree would have found it and placed that vertex higher in the tree.

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.