CS4102 - ALGORITHMS - SPRING 2022

DUE FRIDAY, MARCH 18, 2022 AT **11:30 pm** VIA GRADESCOPE

B1 BASIC (SSB3VK)

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: sjb9qvc, hk2ku, bnh5fh

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: The maximum number of vertices that can be on the queue at one time in a BFS search are n-1 vertices, where n is equal to the total number of vertices in the graph. This can pretty much only occur when the starting node has a edge towards every other node in the graph and during the "for each v $\in$ G.Adj[u]" each v is added to the queue.

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: We have 4 types of edges:

- tree edge
- back edge
- cross edge
- descendent edge

The thing is, in the context of an undirected, BFS tree, tree edges, back edges, and descendent edges are all considered as the same thing; tree edges. The point of tree edges in a BFS tree is that they will *always* connect vertices that have a depth difference of 1, simply based off of the definition of BFS. A BFS tree is *guaranteed* to traverse a tree one "layer" of edges at a time. These non-tree edges are cross edges and will always connect vertices that have a depth difference of 0.

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? This is because the vertex in question is connected via a cross edge and this cross edge can create a triangular cycle (in the simplest case).

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? Think about the simplest two-node case where v_1 points to v_2.

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. Think about the case where v_1 is part of a massive tree and v_2 is all alone. But note this case can only happen in an non-connected graph.

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: We can recognize that a directed graph has a cycle via the DFS-visit algorithm if we check each v adjacent to u to not be gray. Ie if we find that an adjacent vertex v is gray, we've found a back edge, cross edge, or descendent edge and can conclude that we have a cycle.

In the case of an undirected graph, we can do pretty much the same thing but we have to keep track of each node's parent and ensure that the adjacent edge goes to a non-parent node (if we hit a gray node, then we have a cycle). We don't have to keep track of this in a directed graph because *any* back edge indicates a cycle. Whereas in an undirected graph, all nodes can access their parent nodes, therefore, true "cycle-indicating-back-edges" are only those edges that connect to grandparent and above nodes. All other types of "bad" edges (cross and descendent) indicate cycles. Also, edges that connect to black nodes indicate cycles as well.

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: I think there are two changes required. The first is that we need a method to output the path from the start to target node. The other modification is that we need to be able to terminate when we find the target node. I am also assuming (for the sake of simplicity) that we begin our DFS Algorithm on the start node so even if the target node exists in the same set, if there is no path from s to t, we won't have multiple calls of DFS-VISIT from DFS_sweep.

The best way to go about this is to add a termination condition between lines 7 and 8 of DFS-VISIT that terminates if node v is equal to t.

The other modification is that after line 7 in *DFS_sweep*, we have the following code:

while(t.pi != NIL or t.pi != s):
print(t)
t = t.pi

This will print out the path from the target to the start node. This could be printed from s to t trivially (using a LIFO) queue but I won't be doing that here.

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: I would add a parameter to DFS-Visit that sets the value of $u.cc$ to the value of cc from the original call to DFS-Visit in DFS_sweep. In addition, I would add a line somewhere that increments a value that counts the number of cc's. Coincidentally, this is the number of times we call DFS-VISIT from DFS_sweep.

My implementation assumes cc is the value of the node that is initially referenced in the DFS-VISIT call from DFS_sweep. This way we can use cc as an "id".

This would require the following changes:

- Between lines 6 and 7 in DFS_sweep add the code to increment a cc counter
- Change DFS-VISIT(G, u) to DFS-VISIT(G, u, cc)
- Betweeen lines 6 and 7 of DFS-VISIT, set v.cc to the cc argument passed into DFS-visit originally. Also, pass in the cc argument from the og DFS-visit call to DFS-visit in line 7.

In terms of time-complexity, these modifications don't make any major differences. We only do a factor amount of work greater than in the normal case Θ(V + V + E) instead of Θ(V + E). Therefore, the time complexity stays the same.

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: The only case where a BFS and DFS tree would differ is when cross edges exist. Consider the following case. The case of a three node undirected graph. Lets name the start node A and the first node selected in DFS to be B. In a BFS search of this graph, we would generate a tree where B and C are both children of A despite being connected themselves. Both of which would have a depth of 1. Consider the same graph but subject to a DFS search. We would then produce a tree with A, B, and C in sequence. To put it another way, A would be the parent of B and B the parent of C. This would lead to C having a depth of 2. This depth would be greater than that of the depth of C found by the BFS tree.

In a more abstract sense, a BFS search generates the shortest distance between the start node and all other nodes. Therefore, by definition, the depth of any node in a BFS tree will be less than or equal to the depth of the same node in a DFS tree. This is because unlike a BFS search, a DFS search has no such distance guarantee, it simply explores depth-first.

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.