Homework 5 Solutions

Due: Tuesday 6 December 2022 by 11:59 PM

Instructions. Type your answers to the following questions and submit as a PDF on Gradescope by the due date and time listed above. (You may write your solutions by hand, but remember that it is at your own risk as illegible solutions will not receive credit.) Assign pages to questions when you submit. **For all questions, you must show your work and/or provide a justification for your answers.**

Note on Academic Dishonesty. Although you are allowed to discuss these problems with other people, your work must be entirely your own. It is considered academic dishonesty to read anyone else's solution to any of these problems or to share your solution with another student or to look for solutions to these questions online. See the syllabus for more information on academic dishonesty.

Grading. Some of these questions may be graded for completion and some for accuracy.

Question 1.
(a) A *bridge* is an edge in a graph that, if removed, would cause the graph to become disconnected. Describe and analyze an algorithm for determining if an edge *(u,v)* in a connected, undirected graph is a bridge or not. You do not have to write pseudocode.
(b)Prove that your algorithm works.

(a)
Run DFS on the graph starting at vertex *u*, but ignore *(u,v)* as if it was removed from the graph. If, at the end of the algorithm, *v* is marked as visited, then *(u,v)* is not a bridge, so return false. Otherwise, it is a bridge, so return true.

Runtime.
Assuming an adjacency list representation, we are running DFS once, so the worst-case runtime (such as when the the edge is not a bridge and the graph is connected), the runtime would be O(V+E).

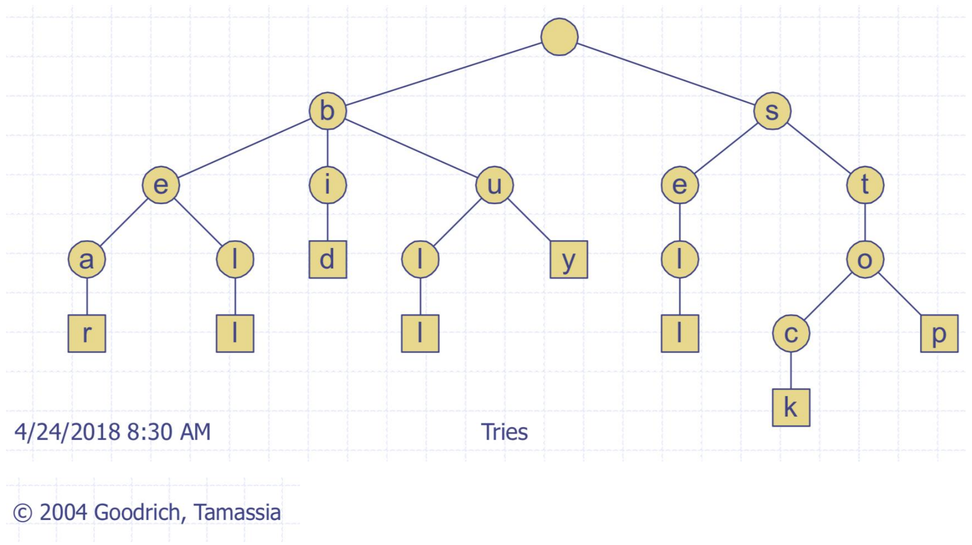
(b)
Proof.
Assume that the algorithm does not work correctly on all graphs. There are two possible cases.
Case 1. Assume that the algorithm returns true when (u,v) is not a bridge. If (u,v) is not a bridge, then removing it (or ignoring it) would not disconnect the graph, which means

that there must be another path from u to v in the graph, so when DFS is run starting at u, we would get a connected component that includes v. But that means v would be "visited" and the algorithm would return false.

Case 2. Assume that the algorithm returns false when (u, v) is a bridge. If (u, v), is a bridge, then removing it (or ignoring it) would disconnect the graph and u and v would be in separate components. Then when DFS is run on u, we would get a connected component of the graph that contains u but does not contain v and v would not be marked as "visited". But that means that the algorithm would return true.

Question 2.
A Trie is a tree structure that is built from a text and used for searching specific words. For example, the following is a Trie that is built from the following set of Strings:

```
      {bear, bell, bid, bull, buy, sell, stock, stop}
```



As you can see, the children of each node are in alphabetical order, which makes it easier to search for a specific character, and each path from a root to a leaf node corresponds to a word. Typically, the leaf node would contain information about where the word occurs in the text.

Write and analyze an algorithm that prints out each word in a trie in alphabetical order. For analysis purposes, let *N* be the total number of nodes in the Trie. You can assume that concatenating a character to the end of a String is O(1) and that printing a String is O(1).

```
Algorithm printWords
Input: A trie T
Output: prints the words in the trie in alphabetical order
```

```
printHelper(T.root, “”)

procedure printHelper(Node n, String s):
if n has no children:
     print s
     return
for each child c of n:
    s += node.val //add the character from that node to the end of s
    printHelper(c, s)
end for
```

Analysis. This algorithm is based on DFS. Since the individual commands are O(1), the runtime is dependent on the number of nodes in the Trie, as the algorithm basically goes through every node. So the total runtime is O(N).

Question 3.
Determine how many topological orderings each graph below has and justify your answer.

(a)

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

There are 0 because the graph has a cycle: 0-1-4-3-0

(b)

0	1, 4
1	
2	
3	2, 4

4	1, 2

There are 4. They are: 3-0-4-2-1, 0-3-4-2-1, 3-0-4-1-2, and 0-3-4-1-2

(c) A graph with *n* vertices labeled $1, 2, \ldots, n$ where each of the vertices from 1 to $n - 1$ have a single outgoing edge that points to vertex *n* and vertex *n* has no outgoing edges.
There are (n-1)! because the order of the first n-1 vertices does not matter as long as they all come before the last vertex.

(d) A graph with *n* vertices labeled 1, 2, …, *n* where for all vertices from 2 to *n*, vertex *v* has a single edge pointing to vertex *v-1*. Vertex 1 has no outgoing edges.
There is only 1 and it is *n, n-1, n-2, …, 1.*

Question 4.
For each of the following graphs, (1) determine if the graph (G*) could be the transitive closure of another directed graph (G) and (2) if the answer is yes, describe the original graph (G) in as much detail as possible, and if the answer is no, explain why. Note that for describing the original graph, we want you to go in as much depth as you can. Don't just write easy things like (Vertex 0 can reach vertex 3, etc.). Think about *strong connectedness, strongly connected components, cycles, topological orderings, self loops,* etc.

0	1	1	0	1
1	0	0	1	0
1	0	1	1	0
0	1	1	1	0
1	0	0	0	1

No, this is not a transitive closure. Note that 0 can reach 1 and 1 can reach 0. This means that 0 is involved in a cycle, which would be indicated with a 1 at (0, 0) in a transitive closure graph, but here it is 0.

(b)

1	0	0	1	1
0	1	0	0	0
0	0	1	1	1

1	0	1	1	1
1	0	1	1	1

No. Note that 0 can reach 3 and 3 can reach 2. That means that 0 can reach 2, which would be indicated with a 1 at (0, 2) in a transitive closure graph. But here it is 0.

(c)

1	1	1	1	1
1	1	1	1	1
0	1	0	0	0
1	1	1	1	1
1	1	1	1	1

No. 3 can get to 2 and 2 can get everywhere else, which means that 3 can get to every other vertex, but there are 0's in that row.

(d)

0	1	0	0	0
0	0	0	0	0
1	1	0	0	0
1	1	1	0	0
1	1	1	1	0

Yes. This graph can be a transitive closure, and we can conclude the following about the original graph:

- The graph has no cycles and no self-loops as there are no 1's in the diagonal.
- There are no strongly connected components in the graph as there are no cycles.
- Note that 4 can reach every other vertex, 3 can reach vertices 0-2, 2 can reach vertices 0 and 1, 0 can reach 1, and 1 cannot reach any other vertex. That means that there is one topological ordering of the graph and it is 4-3-2-0-1.

Question 5.

Consider the following scenario.

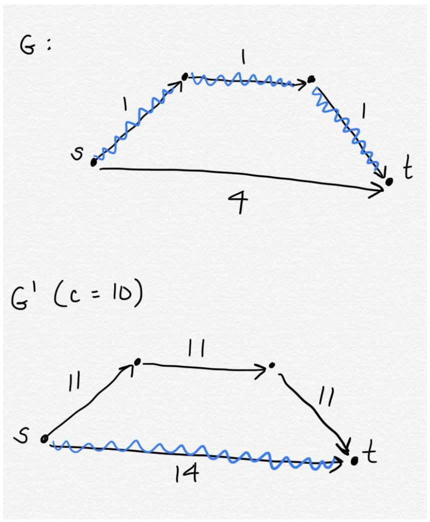
- G is a simple, weighted, undirected graph.

- S is a shortest path tree of G starting at vertex s.
- T is a minimum spanning tree of G.
- G’ is a graph that is basically a copy of G except that every edge weight has been increased by a constant *c*.

Prove or disprove each of the following statements.

(a) *The vertices and edges making up S in G also make up a shortest path tree in G’.*
(b) *The vertices and edges making up T in G also make up a minimum spanning tree in G’.*

(a) Proof by Counterexample



(b)
Assume we got the MST for G by running Kruskal’s Algorithm.
If you take the sorted edge list used to calculate the MST of G using Kruskal’s Algorithm and add *c* to each of the edges, the edges will still be in sorted order. (The same applies if we use a min PQ). Also, any set of edges in G that make a cycle will also make a cycle in G’ since we haven’t changed anything about the graph except the weights. So, when we run Kruskal’s Algorithm on G’, we will consider the edges in the same order that we did for G and since the relative weight orders haven’t changed and the same cycles will occur, we will accept and reject the exact same edges as we did in G, resulting in the same tree (with different weights).

Note: Another approach to proving this is to show that all MST’s of a graph have the same number of edges (exactly V-1 edges), so you will always be adding c(V-1) to the MST for G, which is much different than adding the weights to a shortest path, which can have any number of edges between 1 and V-1.