CS 180 Homework 2

Name: Wang, Zheng CS 180 Dis 1E

UID: 404855295 Time: F 10:00-11:50

TA: Orpaz Goldstein

**Question 1 (Exercise 5, Page 108):**

Here, the base case is a tree with only one vertex. In this case, there is only one leaf; also, there is no node with two children. Thus, the number of nodes with two children, zero, is one less than the number of leaves.

Now, assume that for a binary tree with nodes, it has number of nodes with 2 children and leaves. If we want to add a node to this tree, then we connect to a previous-existing node on the tree.

There are two cases of for which can be attached.

* Firstly, if already has one child, then add to will then create one new node with two children and one new leaf.

Thus, for any binary tree with nodes and fall in to this case, the number of nodes with two children is , and number of leaves is .

* Secondly, if is a leaf, then add to will eliminate one leaf and create a new leaf. Notice that in this case, we won’t impact the number of nodes with two children.

Thus, for any binary tree with nodes and fall into this case, the number of nodes with two children is , and the number of leaves is .

Overall, for any binary tree with nodes, the number of nodes with two children is exactly one less than the number of leaves.

Thus, by induction, for all binary trees, the number of nodes with two children is exactly one less than the number of leaves.

**Question 2 (Exercise 6, Page 108):**

Suppose towards contradiction that there exists some that is an edge of but is not an edge of .

Now, since is obtained by doing DFS on , to must be a DFS tree, and therefore, one of and must be the ancestor of the other. Without loss of generality, let’s assume that is an ancestor of .

Since is also a BFS tree, as is an edge of , thus if belongs to layer and belongs to layer , and cannot be differ more than 1.

Thus, is an ancestor of only one layer above, which means that is ’s parent. Thus, is an edge of , a contradiction.

**Question 3 (Exercise 7, Page 108):**

Suppose towards contradiction that there exist vertices and in such that they are not connected.

Thus, and must each connect with a **disjoint** set of nodes. However, this is not possible since there are only nodes left for and to connect to but the two disjoint set mentioned above will need nodes.

Thus, by pigeon-hole principle, there exist some nodes such that is connected with and at the same time, is connected with . So, the claim made is true.

**Question 4 (Exercise 9, Page 110):**

*Proof.*

Since the distance between and is strictly greater than thus, if we run BFS with as a root, then it is necessary that the resulting three has the last layer , where .

Claim: If we run a BFS starting at , then there exists some layer in the resulting BFS-tree such that there is only one node on .

Proof of the claim: suppose that for all layer , there are larger or equal to 2 nodes. Then, we have two nodes, and , and nodes in layer to . This gives us totally nodes, which is a contradiction. Therefore, there must exist some layer of the BFS-tree where there is only one node.

Claim: Let the node on mentioned above , then deleting will result in destruction of all path from to .

Proof of the claim: Since all nodes in layers to must either be connect with another node in layers to or connect with the in (since if is an edge of , then and must result in layers that are in maximum one layer apart). But since is not on layers to , so for , which is in layer , to be connected with , some node in layers to must be connected to first. But since is the only node on and it has been removed, so no nodes in layers to can be connected with . Thus, there is no path from to after is removed.

Algorithm:

* Run BFS with root on to obtain a BFS tree.
* Initialize a queue. First push to the queue and count the number of its children.
* Then pop from the queue and push all ’s children to the queue. Then count the total number of children of all elements in the queue.
* Repeat this process until we obtain that the total number of children of all elements in the queue is 1
* Then go to that child and remove it

Since run BFS will take ( is the number of edges and is the number of nodes), then to look for the layer with only one node, we will visit each node once (when we pop the nodes from the queue) and visit each edge twice (when we count the number of children and push children to the queue), so that will take as well. Overall, this algorithm will run with

**Question 5 (Exercise 12, Page 112):**

The algorithm goes as follows:

* For each person , create node and , where the first one represents the event of birth of and the second represents the death of .
* For all , add a directed edge from to . i.e. add an edge
* Then go through the fact list.
  + If we encounter the event of the form “For some and , person died before person was born.”, draw a directed edge from to . i.e. add edge .
  + If we encounter the event of the form “For some and , the life spans of and overlapped at least partially.”, draw a directed edge from to ; also draw a directed edge from to . i.e. add edge and .

This will then give us a directed graph containing all information of the birth and death of all people in the set. Denote the graph .

Then, run topological sort on this graph. Suppose that all information is consistent, then topological sorting will produce an ordering of all nodes of such that they are consistent with the information collected. Since there will be nodes in the , we take to be the difference of the time between two consecutively ordered nodes. We then calculate the date of all events in the sorted list of nodes by adding to the beginning year the 200-year interval.

Suppose then that the information is not internally consistent. Then when we go down the time line, we must have encountered some event that has already happened in the past. In this case, we will end up with a cycle in and topological sort will stop before expiring all the nodes. Then, if the topological sort fails, then we have information that is not internally consistent.

Before we analyze the algorithm, assume that is the number of edges of and is the number of people in the set. Thus, construct to for all will take steps and adding all the edges will take steps. Then construction of the graph will run with order Since the topological sort will also run at time. Thus, the overall algorithm will run with order time.

**Question 6:**

Introduction of idea:

The goal of this question is to find if there is a way from the start cell to the end. We regard the maze as a graph with each cell in the grid representing a node and each edge connecting a pair of touching cells where neither is in the range of alarm. Given this graph representation of the maze, we can run a DFS rooted on the start cell to see if it is connected with the end.

Algorithm:

This algorithm uses a stack to do DFS search on a “imaginary” graph of the maze. (The graph is “imaginary” since there will be no explicit steps used to construct the graph.)

So, the DFS algorithm here will be slightly modified: instead of directly following an edge (which we haven’t explicitly made), check three condition to see if there is an edge between a pair of touching cells:

1. Does not trigger an alarm
2. It is actually inside the grid.

For the following explanation of the algorithm, assume a cell’s location is , and returning TRUE means there is a solution, and returning FALSE means there is no solution.

Then the exact algorithm goes like this:

* Initialize an matrix and fill every place with ‘\*‘
* Initialize a stack
* Push the Start cell into it mark the Start cell (in ) ‘V’
* While the stack is not empty
  + Pop the cell with location from the top of stack
  + If it is the End cell
    - Return **TRUE**
  + If the cell does not trigger alarm, and is inside the grid, and is not marked ‘V’
    - Push cell to the stack and mark it as ‘V’ in
  + If the cell does not trigger alarm, and is inside the grid, and is not marked ‘V’
    - Push cell to the stack and mark it as ‘V’ in
  + If the gird does not trigger alarm, and is inside the grid, and is not marked ‘V’
    - Push cell to the stack and mark it as ‘V’ in
  + If the gird does not trigger alarm, and is inside the grid, and is not marked ‘V’
    - Push cell to the stack and mark it as ‘V’ in
* Endwhile
* Return **FALSE**

This algorithm will run with time complexity . The reasons are:

1. Initialize will take steps
2. Initialize a stack will run with
3. Mark the Start cell takes 1 step
4. The while loop will run at time nodes. This is because we are going through all of the nodes in the grid and inside each loop, we are doing a constant number of operations

Therefore, the overall complexity is .