Algorithms Homework 1

Question 1.)

a.) Prove or disprove:

$$log_2 f(n) \in \theta(log_2 g(n))$$

Given:

$$f(n) \in \theta(g(n)) \iff \lim_{n \to \infty} \frac{f(n)}{g(n)} = c$$

This is also true:

$$c * f(n) \leq g(n) \leq c * f(n)$$

If we assume is true:

$$log_2 f(n) \in \theta(log_2 g(n))$$

We can assert that:

$$log_2 f(n) \in \theta(log_2 g(n)) \iff \lim_{n \to \infty} \frac{log_2 f(n)}{log_2 g(n)} = c$$

If we assume that the limit approaches a constant, by g(n) must be an upper bound as well as a lower bound. Both equations below must be true.

$$log_2 f(n) \leq c * log_2 g(n)$$

$$log_2 f(n) \geq c * log_2 g(n)$$

Since both are true due to the property of logarithmic growth

$$f(n) \in O(g(n))$$

and

$$f(n) \in \Omega(g(n))$$

Thus, by squeeze theorem:

$$log_2 f(n) = \theta(log_2(g(n))$$

b.) Prove or disprove:

$$2^{f(n)} \in \theta(2^{g(n)})$$

Given:

$$f(n) \in \theta(g(n)) \iff \lim_{n \to \infty} \frac{f(n)}{g(n)} = c$$

This is also true:

$$c * f(n) \leq g(n) \leq c * f(n)$$

If we assume this is true:

$$log_2 f(n) \in \theta(log_2 g(n))$$

We can assert:

$$2^{f(n)} \in \theta(2^{g(n)}) \iff \lim_{n \to \infty} \frac{2^{f(n)}}{2^{g(n)}} = c$$

This means that this must be true:

$$c * 2^{f(n)} \leq 2^{g(n)} \leq c * 2^{f(n)}$$

However, if we assume:

$$f(n) = 2n$$

$$g(n) = 3n$$

We can now rewrite the inequality as

$$c * 2^{2^n} \le 2^{3^n} \le c * 2^{2^n}$$

$$= c * 4^n \le 8^n \le c * 4^n$$

Observing this we can see that a function with base 8 grows exponentially bigger than base 4. As a result there is no constant ‘c’ that can make 4^n have a larger growth rate than 8^n as n approaches infinity.

Question 2.) Prove any connected acyclic graph with at least two vertices has at least two vertices of degree one.

Base case:

$$n = 2$$

Observe this graph is both acyclic and connected. Both nodes in this graph also have a degree of 1.

Given:

$$\text{Connected} + \text{Acyclic} \Rightarrow (n-1)\text{ Edges}$$

This is also true:

$$\text{Acyclic} + (n-1)\text{ Edges} \Rightarrow \text{Connected}$$

For n + 1 nodes we must have a graph with 2 vertices and 3 nodes as this will keep the theorem above true.

Suppose n’ = n-1 where the node removed has degrees d > 1:

$$\text{number of edges } E < (n' - 1)$$

Because E is not equal to n’ - 1:

$$\text{Acyclic} + \text{E} \not\Rightarrow \text{Connected}$$

Now suppose n’ = n-1 where node removed has degree d = 1

$$E == (n' - 1)$$

Thus,

$$\text{Acyclic} + \text{E} \not\Rightarrow \text{Connected}$$

For graph G to be connected, n number of nodes must be a at least n-1 in order to have enough resources to connect all nodes in G. Acyclic also be true for n-1 edges to optimize to connectivity.

By IH, Connected & Acyclic must be true in order to all graphs with nodes n ≥ 2 to have atleast 2 nodes with degree 1 to exist in graph G

Question 3.) French Flag Run Pseudocode

By using DFS, we traverse the graph and keep track of the previous color. Based on the previous color, we look for a specific color in order to choose our next node to visit. As a result we only go down paths that follow the pattern “red, white, blue”. For example if we were at a node that took a white edge, it will go down a path with a blue edge and when coming back to this parent node, it will look again for another blue edge to go down. This is consistent throughout all colors.

```
/*
    bag → Needs to be an empty stack that take nodes from graph G
    previousColor →String that holds the name of the previous color
    targetColor →String that holds the name of the target color
    G → This is a graph in the form of an adjacency list
    visitedArray →Array used to mark each node as visited as you visit it.
*/
flagRunDfs( bag, G, previousColor, targetColor, visitedArray)
    IF no child or visited in visitedArray
        pop from bag
        RETURN
    ELSE
        IF previousColor is red
            set targetColor to white
        ELSEIF targetColor is white
            set targetColor to blue
        ELSE
            set targetColor to red
        ENDIF
        FOR all adjacent nodes
            IF edge color is targetColor
                set previousColor to targetColor
                call flagRunDfs(bag,G,previousColor, targetColor
        RETURN
    ENDIF
```

Question 4.) Number Maze

Use BFS to go through each potential tile connected to the current tile and enqueue them. Then as you dequeue them, you enqueue the adjacent tiles the a distance away based on the number on the current tile. You then move across the board while incrementing the current tile node until you reach the target.

/*

G → provided nxn grid

TILE →Node type that contains the x location, y location, distance (amount of tiles you can travel) and counter(move counter)

currentNode→This is the currently visited node

tempNumber→current distance you can travel

nodeQueue → Queue containing the current starting node

visitedArray → This is where a node is marked once it has been visited

NULL → no solution

targetNode → Node created with the attributes of the x and y of the target spot on the grid

*/

mazeBfs(currentNode, nodeQueue, G, visitedArray, targetNode)

WHILE nodeQueue is not empty

SET currentNode AS DEQUEUED node FROM nodeQueue

SET tempNumber AS distance attribute FROM currentNode

IF currentNode has been visited

CONTINUE

ELSE

MARK currentNode as visited in visitedArray

IF tempNumber to the right of curentNode is not greater than dimension of the grid

SET tempNode AS constructed new node with attributes of type TILE

SET tempNode counter attribute TO currentNode counter ADD 1

UPDATE x attribute OF tempNode

ENQUEUE tempNode INTO nodeQueue

ELSEIF tempNumber to the left of currentNode is not less than dimension of grid

SET tempNode AS constructed new node with attributes of type TILE

SET tempNode counter attribute TO currentNode counter ADD 1

UPDATE x attribute OF tempNode

ENQUEUE tempNode INTO nodeQueue

ELSEIF tempNumber to the bottom of currentNode is not less than dimension of grid

SET tempNode AS constructed new node with attributes of type TILE

SET tempNode counter attribute TO currentNode counter ADD 1

UPDATE y attribute OF tempNode

ENQUEUE tempNode INTO nodeQueue

ELSEIF G spot tempNumber to the top is not a negative number

SET tempNode AS constructed new node with attributes of type TILE

SET tempNode counter attribute TO currentNode counter ADD 1

UPDATE y attribute OF tempNode

ENQUEUE tempNode INTO nodeQueue

IF currentNode is targetNode

RETURN counter attribute of currentNode

RETURN NULL

Question 5.) Soccer Tournament

In order to find a cycle, use DFS by mark each node as visited while traversing and keeping track of the current path with a hashset. If a node has been visited and is in a hashset then there will be a cycle. We use a hashset because if we use a stack, the time complexity to check if it is in the current stack path is O(n). With a hashset we can optimize it to O(1) time.

```
/*
    hashSet →Needs to be defined before hand as an empty set
    visitedArray →Needs to be defined beforehand as an emptyArray
    G → This is a graph in the form of an adjacency list
*/
soccerDfs(hashSet, G, visitedArray)
    IF node has been visited
        IF hashSet contains node
            RETURN False
        ENDIF
    ELSE
        ADD node into hashSet
        mark node about to be visited as visited in visitedArray
    ENDIF
    FOR all adjacent nodes to current node in graph G
        IF call soccerDfs(hashSet, G, visitedArray) is False
            RETURN False
        ENDIF
    Remove node from hashSet
    RETURN True
```

SOURCES:

My Big Brain Cells