Stefan Emmons
COSC-3020-01
Lab 06
10-23-2019
Part 2: Runtime analysis

As with any runtime analysis, let's begin by looking at our code that does the majority of the work. We are not considering the helper function, is it only passes values to this function once during the traversal of one graph.

```
17  function depthFirstSearch(startNode, graph, isVisited) {
18      if(!isVisited[startNode]) {
19          isVisited[startNode] = true;
20          console.log((startNode + 1) + " -> ");
21
22      for(let j = 0; j < graph.length; j++) {
23          if(graph[startNode][j] == 1 && !isVisited[j]) {
24              depthFirstSearch(j, graph, isVisited);
25          }
26      }
27    }
28  }
```

Lines 18 – 20 → Constant, or θ(1)

Line 22 → θ(n)

Line 23, within the loop → θ(1)

Line 24, dependent on how many loop executions are carried out → T(n-1)

We can now begin building a recurrence relation based on this information. It should also be noted that our base case is implicit, as the recursion stops when all nodes have been visited (marked as true), or there is only one node.

$$T(n) = \begin{cases} 1 & if\ n \leq 1 \\ T(n-1) + n & if\ n > 1 \end{cases}$$

Solving through substitution:

$$T(n) = T(n-1) + n$$

$$= T(n^2 - 1) + 2n$$

$$= T(n^3 - 1) + 3n$$

Based on this noticeable pattern, we can begin to model this relation with "i" in the following fashion:

$$T(n^i - 1) + i * n$$

Solving for i:

$n^i - 1 = 1$

i = log(n)

Substitute "i" back into our original equation, and we can begin to find bounds for this function. However, we just want to know about the absolute worst-case for this algorithm overall. For the sake of covering all bases though, let's set the next equation up:

$$T(n^{\log(n)} - 1) + \log(n) * n$$

$$= T(n^{\log(n)}) + n * \log(n)$$

Now let's say we must analyze a fully connected graph, that is, all vertices and all edges must be visited by this function. This is a worst-case scenario because the main loop body will need to visit every single edge for every possible node. This means that since I am using adjacency matrices as my data structure, the function would need to run through all rows*columns. Also, because I am using an implicit stack with recursion, this doesn't necessarily mean I get to avoid visiting each adjacency relation. Especially in a worst case scenario where all nodes are connected, each neighbor will need to be checked to see if it must be visited. This would lead to a worst-case time complexity of $\theta(|V|^2)$.