Jordan Allard
CSCI 264-01
Homework 5

HW5 Problem 3 Writeup

a) Verbal Description (Heart of the Solution)

This algorithm uses the principles behind topological sort to find the longest chain of prerequisites from a given list of courses. The solutions are calculated and stored as it travels through the values, in the style of dynamic programming. The algorithm uses a depth-first search to determine the order in which to calculate the values in the solution array.

Heart of the Solution for the dynamic programming portion of the algorithm:

I) WHAT: depths[j] = the longest possible chain of prerequisites ending with vertex j

II) HOW:

depths[j] = 1 if inDegree of j = 0

depths[j] = max(depths[k]) + 1, where k = all vertices with an edge that points to vertex j

III) WHERE: max(depths[j])

b) Pseudocode

```
DFS(v):
      For i = 1 to adjList[v].size():
            If depths[adjList[v][i]] == -1: DFS(v)
            If depths[adjList[v][i]] > currentMax:
                  currentMax = depths[adjList[v][i]]
      depths[v] = currentMax + 1

DFS-Main():
      Given a directed graph with n vertices and m edges
      Create and adjacency list adjList
      For i = 1 to n: depths[i] = -1
      For j = 1 to n:
            If depths[i] == -1: DFS(i)
      Return max(depths)
```

c) Proof of Correctness (Explanation of HOW)

The core of this algorithm is a dynamic programming approach that calculates the maximum prerequitie chain of each vertex and uses those values to calculate the maximum prerequisite chains for all the vertices that the others point to. If the in-degree of a vertex j is zero, then course j has no prerequisites and the longest chain must be 1. In any other cases, if the algorithm always chooses the prequisite path with the longest chain, and since it iterates through every vertex possible, it is guaranteed to eventually find maximum length chain.

In order to fill the solution array in a useful way, the algorithm must calculate the longest chains for each of the possible prerequisites of a course before being able to calculate the chain for the current course. This means that for every edge (u, v), where u is a prerequisite for v, u must be visited by the algorithm before v. This kind of traversal is the very defintion of a of a topological sort, and is therefore easily accomplished through a depth-first search. By nature, DFS travels to and "finishes" (which in this case would include the chain length calculation) the points that are furthest away, then climbs up the levels back towards the starting vertex. If DFS creates a topological ordering of the vertices, then the algorithm can guarantee that each prerequisite vertex will be finished before the vertices that they point to, thus creating an efficient way to find the maximum chain of prerequisites.

d) Running Time Estimate

O(n + m)

e) Running Time Estimate Reasoning

The running time for this algorithm can be broken down as follows:

- Building the adjacency list from collected input --> O(n) + O(m)
- Initializing the adjacenty list = O(n)
- DFS --> O(n + m)
- Everything else --> O(1)

Since this algorithm fills the solution array according to a depth-first search, the largest running time in the algorithm is O(n + m), the running time of the average DFS. Between the for loop and the recursion, each vertex and edge will be examined once and only once, creating the running time O(n + m). Thus the final running time of this algorithm is O(n) + O(m) + O(n) + O(n + m) + O(1) = O(n + m).