3.16

Andrew Lee

March 13, 2017

Suppose a CS curriculum consists of n courses, all of them mandatory. The prerequisite graph G has a node for each course, and an edge from course v to course w if and only if v is a prerequisite for w. Find an algorithm that works directly with this graph representation, and computes the minimum number of semesters necessary to complete the curriculum (assume that a student can take any number of courses in one semester). The running time of your algorithm should be linear.

The first thing to note is that this will be a directed graph since we need to specify a pre-requisite, class u for class v, it is not true that class v will be a prerequisite for class u. Furthermore, since "a student can take any number of courses in one semester", we can start at all the classes that do not have a requirement or rather vertices that are only sources. vertex, and simultaneously traverse all the adjacency lists of vertices at every stage.

The specific search algorithm for this problem will be breadth first search, which specializes in being able to find all the possible vertices within n steps (edges) from the root (or roots). This behavior is desirable because of the condition that we can traverse {multiple edges at once, and the speed at which BFS works becomes better more edges can be traversed.

```
#!/usr/bin/python
from sys import exit

def findkey(dictionary, stack):
    returnlist = []
    if len(stack) == 0:
        newmin = 100000
        for i in dictionary.keys():
            newmin = min(newmin, i)
        returnlist.append(newmin)
    else:
        for i in stack:
            returnlist.append(i)
    return returnlist

def findMinimumSemesters(dictionary, totalstack = [], stack = [], steps
    = 0):
    if len(stack) == len(dictionary):
        print("Taking all classes will take at minimum", steps, "
            semesters.")
        exit();
    keys = findkey(dictionary, stack)
    for i in keys:
        for p in dictionary[i]:
            stack.append(p)
    for i in stack:
        if i not in totalstack:
            totalstack.append(i)
        else:
            stack.remove(i)
    steps+=1
    findMinimumSemesters(dictionary, totalstack, stack, steps)






dictionary = {121:[241, 251],
241:[273, 276],
251:[259, 263, 273, 276, 284, 300, 315, 333, 336, 350, 390],
259:[390],
263:[390],
273:[390],
276:[390],
284:[390],
300:[390],
315:[390],
333:[390],
336:[390],
350:[390],
390:[]}

findMinimumSemesters(dictionary)
```

The reason why this program is linear is because we will wait at most N steps for the computation to complete, but because we are allowed to take “multiple classes” this will most likely be significantly faster than just taking one step at a time. In fact, although T(n)=$\bigotimes(n)$, $T(n) = \Omega(1)$ because in the best case scenario, with no prerequisites for classes, we can add all the classes in one “step”.