3.23

Andrew Lee

March 15, 2017

Give an efficient algorithm that takes as input a directed acyclic graph G = (V, E), and two verticess,t $\in V$, and outputs the number of different directed paths from s to t in G .

```
graph = {1: set([2, 3]),
         2: set([4, 5]),
         3: set([6]),
         4: set([7]),
         5: set([7]),
         6: set([5, 7]),
         7: set([])}

def dfs(graph, root, target, path=None):
    if path is None:
        path = [root]
    if root == target:
        yield path
    for i in [x for x in graph[root] if x not in path]:
        for each_path in dfs(graph, i, target, path + [i]):
            yield each_path

print(list(dfs(graph, 1, 7)))
```

Uses a modified DFS to accomplish the task. Hence, T(n) = $\odot(n)$.