Anooj Pai Homework 5

1)
```
def bipartite(graph):
    c1 = [-1] * len(graph)
    c1[0] = 0
    q = deque()
    q.append(0)
    while q:
        element = q.pop()
        for v in graph[element]:
            if c1[v] == -1:
                c1[v] = 1 - c1[element]
                q.append(v)
            elif c1[v] == c1[element]:
                return False
    return True
```

If the graph is bipartite then the function returns True. Otherwise it returns False. The run time of this function is $O(|v| + |E|)$ where v is the number of verticies and E is the number of edges. This is shown in the algo because it goes through the v and E one time.

2) a) Prove a non empty DAG must have at least 1 source

A DAG has directed edges and no directed cycles. Because there are no cycles, there must be a vertacie with no incoming edges. this is because, if there was an edge incomming there it can be traced back making it a directed cycle. the vertecie with no incoming edges it is called a source. This proves that a non empty DAG must have at least 1 source otherwise there will be a directed cycle which would make the graph not a DAG.

b) The time complexity of the algorithm would be $O(|V|^2)$. V is equal to the number of vertacies in the graph.

Describe the algorithm:

It would itterate through the matrix looking for no incoming edges, or 0, how it is represented. When the row is found it will return the index of the row, or if it is not found it will not return anything meaning there are no sources.

c) Time complexity of finding the source in a directed graph: $O(|V|+|E|)$. This is when V is the number of vertacies and e is the number of edges in the graph.

Description of algorithm:

it starts by creating an array of size v for the degrees of each vertacie. Then initialize each of them to 0 until it loops through each of the vertacies. for each neighbor, increase degrees by 1. then itterate though the array until you find the vertex with 0 then return the index. If none is found then return nothing.

3) To start you initalize an array of size v with v is the number of vertacies in the graph. Then intitiaze each index to 0. Next itterate through the graph and for each vertex go through and count the number of neighbors of it and its neighors and assign it to the element. Once you iterate through all of the vertacies you return the array. The time complexity of this algorithm is also $O(|V|+|E|)$.

4) a) DAG:

first, create an array of size v, where v is the number of vertacies in the graph. set each element to the minimum value. Then sort the array from smallest to largest. For each element check if the value is 0, or it has no outgoing edges, then set the reachability-weight(v) to weight(v). If it has an outgoing edge then set the weight(v) and reachability-weight(v) to the number of its outgoing neighbors. After iterating through all the verticies return the reachability-weight array.

Time complexity: $O(|V|+|E|)$

b) General directed Graph:

first, initialize an array of size v where v is the number of vertacies in the graph. Initialize each element to the minimum value. then ick a vertex, lets call this x. Set the reachability weight to its weight. Iterate through the rest of the vertacies and repeat the process. If the reacability-weight is greater than $-\infty$ then set the reacability-weight to the sum of the current vertex and its weight. then return the reachability-weight array.

Time complexity $= O(|V| \cdot |E|)$