Andrew Letz
CIS 315
Professor Wilson
Assignment #1

Question 1

The node with in-degree n-1 and out-degree 0 will be represented by a row with all 0s and its respective column being all 1s except for M(n, n).

Pseudo-code:

```
M = n by n matrix representing graph G starting at node 0
N = [0 … n]
while len(N) > 1:
      if M[N[0]][N[1]] == 1: // if there is an outgoing edge from N[0]
            del N[0]    // remove it from candidates
      else:
            del N[1]    // if there is no ingoing edge to N[1], remove
                        // it from candidates
// This first while loop takes (n - 1) time, now check if last element is
// what we are looking for
target = N[0] // get the only element left in the list
for x in range(n):
      if M[target][x] != 0:
            return false      // not every entry in the row is 0
      if M[x][target] != 1:
            if (x != target):
                  return false // not every entry in the col is 1 (except
                               // for the target)
return true       // all checks passed
```

The overall complexity of this algorithm is O(n-1) + O(n) which is equivalent to O(n).

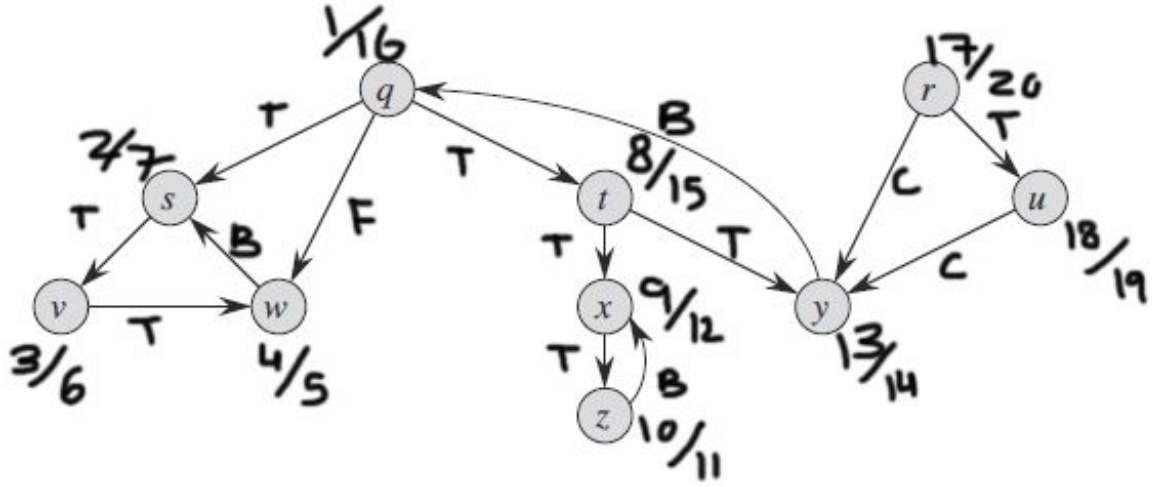
Question 2 (exercise 22.2-7)

If wrestlers are represented by nodes on a graph G where rivalries are edges E, then this graph would need to be two-colorable (ie bipartite) in order to only have rivalries between babyfaces and heels.

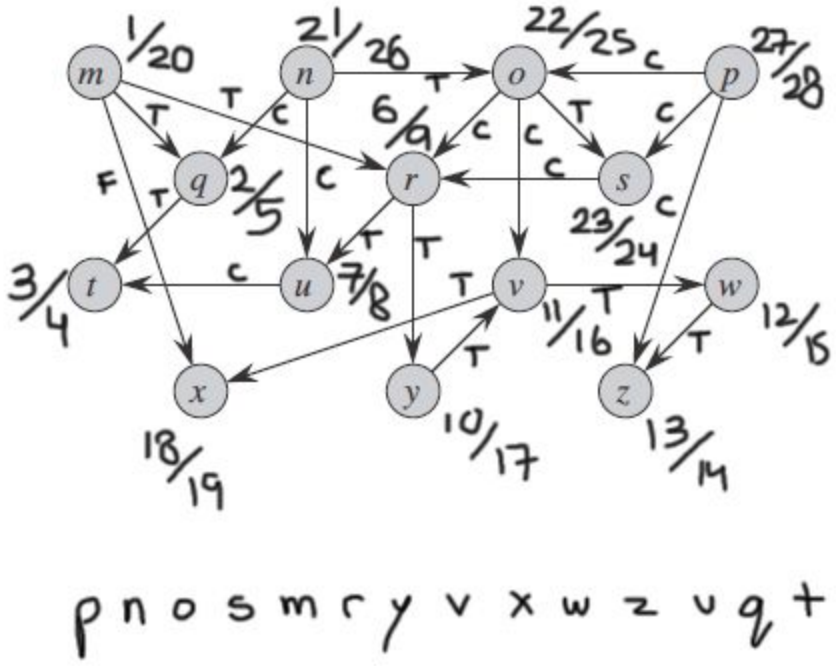
In order to determine whether a graph is bipartite, one can start by performing a BFS on each component of the graph. In addition to the base algorithm, however, instead of only giving a color of black to processed nodes each step will alternate between colors red and black. This way, if a neighbor is ever found to be the same color as the currently selected node, it is known instantly that the graph is not bipartite and therefore a rivalry must exist between a babyface and babyface or heel and heel.

As the BFS algorithm takes O(|V| + |E|) time, this would take O(n + r) time where n is the number of wrestlers and r the number of pairs (edges).

Question 3 (exercise 22.3-2)



Question 4 (exercise 22.4-1)



Question 5

node = class representing a node with default distance value -1. holds a list representing all adjacent nodes titled adj_list.

DAG = list of nodes representing topological ordering of graph G

```
def longestPath(DAG, n):
      DAG[0].d = 0      // set distance of first node to 0
      for node in DAG:
            for adj_node in node.adj_list:
                  // if the adjacent node is unupdated, give it a new d value
                  if (node.d + 1) > adj_node.d:
                        adj_node.d = node.d + 1
      return DAG[n].d
```