Homework 3

CSCE 411
Sahil Palnitkar

22.2-7)

Create a graph G which has each vertex is a wrestler and each edge is a rivalry. There are n vertices and r edges. We will perform BFS until all nodes are visited. We have to make sure that each edge only connects a babyface to a heel and not a heel to heel or babyface to babyface.

Assuming there are no wrestlers with no rivalries, and each rivalry has wrestler 1 and wrestler 2
Pseudocode:

```
(Choose a random point x as the primary point)
point x = babyface
for each point:
if (path_length from point to x = odd):
point = babyface
else:
point = heel
for each rivalry:
if ((rivalry.wrestler1 == babyface and rivalry.wrestler2 == babyface)
or (rivalry.wrestler1 == heel and rivalry.wrestler2 == heel):
return false (not possible to designate)
else:
return true
```

Correctness:
Since there are no points without a designation and no points that aren't a part of a rivalry, this algorithm will be correct.

Time Complexity:
O(n + r)

22.4-2)

We know that the graph is acyclic and we need to list the paths from one point to another.
Let us assume that each point has a paths variable which paths from that point to t and a children variable. Let V be the number of nodes and E be the number of edges.

Psuedocode:

```
algorithm(s,t):
   if s == t:
      return 1 //as it is the same point
   else:
      if !s.paths:
           s.paths = sum(algorithm(x,t) for x in s.children)
      return s.paths
```

Correctness:
We can assume that sum returns 0 if s has no children. So, using this recursive algorithm keeps track of the paths taken during computation. This will return all the paths from one point to another point.

Time Complexity:
O(V+E)

22-3)
a)
=> direction
Let a cycle in which a vertex is visited once at most be called a simple cycle. Let a cycle in which a vertex is visited more than once be called a complex cycle. All vertices in a simple cycle have indegree and outdegree = 1. Complex cycles are a combination of simple cycles. Therefore, a complex cycle has indegree = outdegree. This is the same for an Euler Tour. Therefore, all vertices in an Euler Tour have in-degree = out-degree.

<= direction
Assume the in-degree and out-degree for all vertices on the graph are equal.
Let C be the longest complex cycle. If C is not an Euler tour, there exists vertex x touched by C, such that not all edges in and out of x are exhausted by C. Make a cycle B, starting and ending at x by performing a walk in G-C. (As in-out degrees are equal). Therefore, the cycle B that starts at x and goes along the edges of B and then along the edges of C is a longer cycle than C. Therefore, C is not the longest complex cycle. Therefore, C is an Euler Tour with in-degree = out-degree.

b)
We can start at a random vertex and make a cycle that ends at the same vertex. We can start at a random vertex and follow each edge while removing them from the edge list. When the current vertex has no more out-edges, we can push it into another list, till the main list becomes empty. We do this recursively.

Pseudocode:
```
Algorithm1(G):
make edge list E
make result list S
picking random vertex v,
        Algorithm2(G,v,S)
return S

Algorithm(G,v,S):
for edge e in E:
delete e = (v,u) from E
Algorithm2(G,u,S)
add v to S
return S
```

Correctness:
To do an Euler Tour, we can remove vertices from S and follow the edges. This will ensure all edges are checked.

Time Complexity:
Since at most E calls are being made, the time complexity is O(E)

23.1-9)

Assume there is a cheaper tree than T'.

Therefore, we have T'' s.t. weight(T'')< weight(T').

Let S be the edges in T but not in T'. We can construct an MST of G by considering (S union T'').
This is a spanning tree as (S union T') is, and T'' connects all the vertices in V' like T' does.
But, we get w(S U T'') = w(s) + w(T'') < w(S) + w(T')
= w(S U T') = w(T).
This results in a spanning tree with a lower weight than the MST. This is a contradiction. Therefore, there cannot be a tree cheaper than T'. Therefore, T' is the MST.