EECS 340: Algorithms **(Due: July 26, 2020)**

Assignment #7

Instructor: Orhan Ozguner *Name:* Philippe Boulas, Gabriella Koh, *CaseID:* pjb123, gbk27

Problem 1

(a)

Let us assume there exists a graph, G, where nodes are teams and edges represents who beats who: edge uv represents u beats v. There are n nodes representing n teams and m edges representing m number of games played.

Let us assume the transpose of this graph, G^T, also exists where edge uv represents u lost to v.

Let us assume set of vertices, V, is ordered by previous year's rank

Let us assume previous-rank is a given list with all n teams ordered by the rank from the previous year. Calling previous-rank(u) will return u's rank from the previous year. This will work in the same way that the counter "root" does in office hour examples. DFS will visit vertices in ascending order from top ranked teams (given ordered V), each vertex visited will supply the previous year's rank - similar to the way the root counter did in the office hour example. Choosing to use previous-rank(u) instead of root counter was because it seemingly answered the question more directly as domination factor or z_i is defined as " the rank of the best team (that is, the highest ranked team according to last year's rankings) that is dominated by team i."

The field u.r will hold the domination factor, z_i, of each team.

ALGORITHM DESCRIPTION

Given the graph, G, where nodes are teams and edges represents who beats who: edge ij represents i beats j. There are n nodes representing n teams and m edges representing m number of games played. Let there be a team j that is dominated by i. This means that in graph, G, j is a descendant of i.

The goal of this problem is to find the maximum rank of all teams that i dominates. Let us take the transpose of the graph G such that edge ij represents i lost to j. This means that in G^T, i is the descendant of j. We begin the algorithm by initializing each vertex's rank as infinity, and the rank as zero.

The for loop in DFS will begin visiting nodes from the top ranked team (because set of vertices V is sorted by previous year rank). Rank will become the previous year rank of the node it is starting at and DFS-VISIT will be done on G^T, u. Because V is ordered, this means at the beginning the previous year rank will be 1 - this is why using previous-year(u) works the same way as the root counter. As DFS visits each node, rank increments by 1 (the same way root counter does), because, once again, V is ordered by first ranked team, second ranked, and so on.

DFS-VISIT will visit all nodes adjacent to u in G^T. This means that it will find all descendants of u, which means it will find out every team that u dominates. The domination factor is found by the maximum rank of all the teams dominated by u. At this point, let us remember, rank is assigned the previous rank of the top ranked team (at first iteration rank is 1). The domination factor (u.r) of every adjacent node will then also be assigned that rank.

DFS-VISIT will then visit every node in ascending order of previous year rankings (the top ranked team visited first, second ranked team visited second, and so on). RESULT: Each node will be assigned u.r, or their domination factor. If a node's domination factor is ∞ that means that it did not dominate any other team.

Algorithm 1: DFS(G^T)

```
for each u ∈ V do
  u.color ← WHITE
  u.r ← ∞
end for
rank ← 0
for each u ∈ V do
  if u.color is WHITE then
    rank ← previous-rank(u)
    DFS-VISIT(G^T, u)
  end if
end for
```

Algorithm 2: DFS-VISIT(G^T, u)

```
if rank < u.r then
  u.r ← rank
end if
u.color ← GRAY
for each v ∈ Adj(u) do
  if v.color is WHITE then
    DFS-VISIT(G^T,v)
  end if
end for
u.color ← BLACK
```

The runtime of this algorithm will be $O(m + n)$ because all m edges are explored exactly once and all n nodes are explored exactly once.

(b) Assume $z(x) = r(v)$ for some v.
CASE 1 Single path from v to x:

- There $\exists$ a path from v to x in the graph and no other vertex on that path has a better ranking factor than v. If there exists a node, u, with a better ranking than v, then x will get u's ranking.

CASE 2 If there are multiple paths to x, we prove why v is not the best ranking:

- There $\exists$ a path from v to x in the graph and no other vertex on that path has a better ranking factor than v. If there exists a node, u, with a better ranking than v, then x will get u's ranking.
- At time u.d, the path from v to x will be white because we visit vertices in ranking order.
- From white path theorem, we know that x is the descendant of v or the vertex before x is a descendant of v.
- If there is a node s such s is a ancestor of u and has a better ranking than v, u will get the rank of s instead of the rank of v.
- This causes a contradiction

Problem 2

(a) Let G = (V,E) be a directed graph and $uv \in E$. For some run of DFS on G, if $v.f > u.f$ then prove uv must be a cycle. If $v.f > u.f$, then $v.d < u.d$ because if u was discovered first, by White Path Theorem, v would be a descendant of u. We know this because $uv \in E$. If v were to be a descendant of u, then $v.f < u.f$. Therefore, if $v.f > u.f$, we know v must be discovered first.

If v is discovered before u and $v.f > u.f$, meaning that u is finished before v, then by Parenthesis Theorem, u must be an ancestor of v as u is encompassed between the discovery and finishing time of v.

By the WPT, there must exist a path from v to u if u is a descendant of v. Therefore, the edge uv will be a back edge. As uv is a back edge, then uv must also be a cycle.

(b) Let G be a directed graph with vertices a, u, v and 5 edges: uv, au, ua, av, va. All vertices are connected to each other except for the missing edge vu. There exists a path from from v to u in G: $< va, au >$. Let there be a run of DFS starting at vertex a that visits v first.

a visits v. From v, there is no vertex that v can discover and is White. Therefore, v is finished and the program returns to a. Then, a will discover u. From u, there is no White vertex that u can reach, so u reaches its finishing time. Then, the program returns to a where all other vertices have been discovered and finished, so a is finished.

In this run of DFS we have this order of times: $a.d < v.d < v.f < u.d < u.f < a.f$. As per the Parenthesis Theorem, there is no ancestor and descendant relationship between u and v, therefore, the edge uv must be a cross edge.

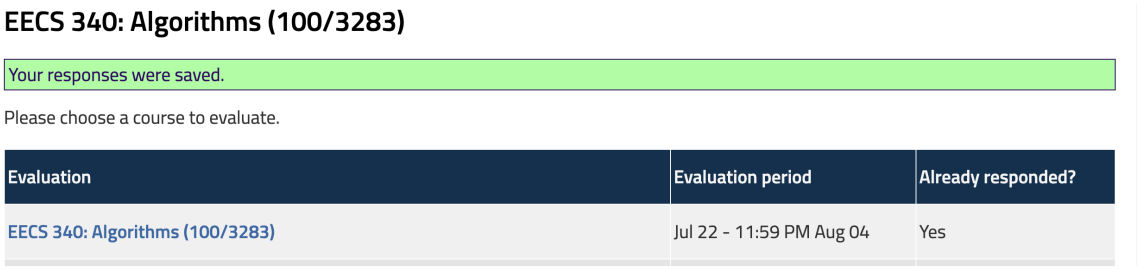
Problem 3

For this problem, we are not going to use adjacency lists, instead will use the bucket data structure. Given V vertices, we know that the maximum number of edges between two vertices is $V - 1$ and given that the max weight of one edge is W, we know that the maximum weight for any path is $(V - 1) * W$. Therefore let us create VW empty numbered buckets starting from 0 and increasing by 1. This will cover all possible weights for any path between two vertices. Let us also create a bucket, x, that holds all the undiscovered vertices with set distance ∞ away from the source.

The source vertex is added to the 0 labeled bucket as it is 0 distance away from itself. All other vertices are in the bucket x. The buckets are checked, starting from 0, in increasing order until it finds a non-empty bucket. A non-empty bucket can contain more than 1 vertex. At each vertex, i, all outgoing edges are checked. If the distance from the source to the outgoing vertex through i is less than the labeled distance on the vertex, then the distance is updated with the new distance through i and the vertex is placed in the bucket with that labeled distance and removed from its previous bucket. If the distance through i is greater than the distance already labeled, then nothing happens. Once all outgoing edges are checked, i is removed from its bucket with the distance from the source equal to the label on the bucket it was just in. Once i is removed from its bucket and permanently labeled, the algorithm finds the next smallest bucket that is occupying a vertex. Once a vertex is permanently labeled, i.e. no longer in any bucket, its distance is set and never changes. As the algorithm progresses, all vertices will be moved from bucket x into another bucket. The algorithm ends when all buckets are empty.

This algorithm sequentially checks the buckets from 0 to VW. At each bucket, these operations occur. It first checks if the bucket is empty. This is a constant time operation. Then, if it is not empty, it will check all outgoing edges and add each vertex to a new bucket if its distance is shortened. Otherwise, nothing occurs. Then, it removes the current vertex from the bucket. Therefore, it checks VW number of buckets and checks E edges during the process. Once the edge has been checked, it is never revisited. Therefore, the running time of this algorithm must be $O(VW + E)$.

Problem 4



EECS 340: Algorithms (100/3283)

Your responses were saved.

Please choose a course to evaluate.

Evaluation	Evaluation period	Already responded?
EECS 340: Algorithms (100/3283)	Jul 22 - 11:59 PM Aug 04	Yes

Figure 1: Philippe Boulas Course Evaluation

Evaluation	Evaluation period	Already responded?
EECS 340: Algorithms (100/3283)	Jul 22 - 11:59 PM Aug 04	Yes

Figure 2: Gabi Koh Course Evaluation