Kaleb Kim
12/2/24

CSDS 310 Assignment 5

Note: Arrays are zero-indexed.

Problem 1

Pseudocode

```
procedure DOMINATION_FACTOR(n, m, rankings, games):
    G ← empty dictionary with default value []
    for (winner, loser) in games:
        G[winner].append(loser)
    Z ← array of size n initialized to ∞
    for 1 ≤ i ≤ n:
        visited ← empty set
        S.append(i)
        best_rank ← rankings[i]
        while S is not empty:
            v ← stack.pop()
            if v ∈ visited:
                continue
            visited.add(v)
            best_rank ← min(best_rank, rankings[v])
            for u ∈ G[v]:
                if u ∉ visited:
                    S.append(u)
            Z[i] ← best_rank
    return O
```

Proof

- **Base Case:** We have $n = 1$. If the graph has only one vertex (team) v, then there are no edges (games) and $z_i = \text{rank}(v)$. In other words, v dominates itself.
- **Inductive Hypothesis**: Assume that for a graph with $n = k$ vertices, the algorithm correctly computes the domination factor z_i for each vertex i.
- **Inductive Step**: Consider a graph G with $n = k + 1$ vertices. Adding a new vertex u with edges does not affect the domination factors of other vertices, as their reachable nodes remain the same. For u, the DFS traversal explores all vertices reachable from u and updates $z[u]$ to the minimum rank of these vertices. By the inductive hypothesis, the algorithm computes z_i for all $i \neq u$ correctly. For u, DFS reaches the correct set of dominated vertices due to the connected structure of the graph. Thus, the algorithm computes z_i for all vertices in G with $n = k + 1$, proving the algorithm correct.

Runtime

Constructing the graph G takes $O(m)$ time to build. Traversing with DFS takes $O(n)$ for the outer loop and processes $O(m)$ edges. Graph G requires $O(n+m)$ space, with the visited set taking $O(n)$. Thus, have have:

$$\text{Time complexity: } O(n+m)$$
$$\text{Space complexity: } O(n+m)$$

Problem 2

a) True. If $uv \in E$ and $v.f > u.f$ in a DFS tree, it means that DFS finishes v after u, implying that u is an ancestor of v in the DFS tree. This creates a contradiction since u should finish before v if it were an ancestor. Therefore, the edge uv must form a cycle, as u is reachable from v, which creates a back edge. Thus, uv lies on a cycle.

b) True. Let u and v be two vertices with $u \neq v$. In DFS, a cross edge occurs when u and v belong to different DFS trees, and the edge uv points from a finished vertex u, (i.e. $u.f < v.d$) to a vertex v in a different subtree. However, if there exists a path from v to u, then u must be visited or revisited during the traversal of v's subtree. As a result, u cannot be finished before v is discovered. The existence of a path from v to u guarantees that u and v are in the same DFS tree. Therefore, uv cannot be classified as a cross edge since the condition for cross edges, $u.f < v.d$, is violated. Awesome

Problem 3

Pseudocode

```
1  procedure WRESTLE(W, R, n, r):
2      G ← empty dictionary with default value []
3      for each (wrestler1, wrestler2) ∈ R:
4          G[wrestler1].append(wrestler2)
5          G[wrestler2].append(wrestler1)
6      C ← empty dictionary with default value NONE
7      Q ← empty queue (implemented with a doubly linked list)
8      for each w ∈ W:
9          if C[w] != NONE:
10             continue
11         C[w] ← BABYFACE
12         Q.enqueue(w)
13         while Q is not empty:
14             v ← Q.dequeue()
15                 for each u ∈ G[v]:
16                     if C[u] is NONE:
17                         C[u] ← HEEL if C[v] = BABYFACE else BABYFACE
18                         Q.enqueue(u)
19                     else if C[u] = C[v]:
20                         return False
21     return C
```

Proof

The algorithm for assigning wrestler types babyface or heel is based on a BFS traversal of the rivalry graph. A graph is bipartite if and only if it can be two-colored, such that each vertex can be assigned one of two colors such that no two adjacent vertices have the same color. This means to prove the algorithm is true, we must prove the graph is bipartite.

The algorithm creates an adjacency list representation, G, of the rivalry graph. It then iterates through each uncolored (NONE) wrestler, and performs a BFS traversal starting from that wrestler.

If the algorithm encounters a wrestler who has the same color as their neighbor, then it means that G is not bipartite. We know then it is impossible to assign wrestler types such that each rivalry is between a babyface and a heel. So, the algorithm returns false.

On the other hand, if the BFS completes without any color conflicts, then the graph is bipartite. In this case, the algorithm returns the color array of the wrestlers.

Runtime

Constructing the graph G takes $O(r)$ time to build, and constructing color dictionary C takes $O(n)$ time to build. Traversing with BFS takes $O(n)$ for the outer loop, and each edge is visited at most twice–hence$O(n) + O(r) = O(n + r)$ time.

For space, we have the graph G containing a space of $O(n + r)$. C is $O(n)$. Thus, we have:

$$\text{Time complexity: } O(n + r)$$
$$\text{Space complexity: } O(n + r)$$

Problem 4