1 Main Idea

In this problem, there are two types of professional wrestlers: babyfaces ("good guys") and heels ("bad guys"), and between any two wrestlers, there may or may not be a rivalry. Given n professional wrestlers and a list of r pairs of wrestlers with rivalries, we want to determine whether it is possible to designate some of the wrestlers as babyfaces and the remainder as heels such that each rivalry is between a babyface and a heel in $O(n + r)$ time.

The main idea of the solution is to represent the wrestlers as vertices in a graph, and the rivalries as edges between the vertices. Then, we use DFS to traverse the graph and assign babyface and heel labels to the vertices. In the end, if we can assign labels to all the vertices, then we return the labels; otherwise, we return "No".

2 Pseudocode

In this pseudocode, we represent the wrestlers as numbers from 1 to n. Arrays are indexed starting from 1. The graph G is represented as an adjacency list, where $G[i]$ is the list of vertices adjacent to vertex i. We assume that r is a list of pairs of wrestlers with rivalries, where each pair is represented as a tuple (u, v). We have an array of labels *labels* where the index i corresponds to the label of wrestler i.

Algorithm 1: Assign Labels to Wrestlers

```
Input: A list of $n$ professional wrestlers and $r$ pairs of wrestlers with
       rivalries
Output: A list of labels for the wrestlers or "No" if no such
        assignment exists
$G$ = [[] for $i = 1$ to $n$] ;                      // Adjacency list
$labels$ = ["" for $i = 1$ to $n$] ;                 // Array of labels
for $(w_1, w_2)$ in $r$ do
    $G[w_1]$.append($w_2$);
    $G[w_2]$.append($w_1$);
end
for $i = 1$ to $n$ do
    if $labels[i]$ = "" then
        if not DFS($G, labels, i$, "babyface") then
            return "No";
        end
    end
end
$babyfaces$ = [i for $i = 1$ to $n$ if $labels[i]$ = "babyface"];
$heels$ = [i for $i = 1$ to $n$ if $labels[i]$ = "heel"];
return $babyfaces, heels$;
```

Algorithm 2: DFS

```
Input: Reference to the graph $G$, reference to the array of labels
       $labels$, wrestler $w$, label $l$
Output: True if the labels can be assigned, False otherwise
$labels[w] = l$;
for $rival$ in $G[w]$ do
    if $labels[rival] = l$ then
        return False;
    end
    if $labels[rival] =$ “” then
        $opposite$ = “babyface” if $l$ = “heel” else “heel”;
        if not DFS($G, labels, rival, opposite$) then
            return False;
        end
    end
    return True;
end
```

3 Proof of Correctness

To prove the correctness of our algorithm, we need to show the following:

1. If a valid labeling exists, our algorithm will find it.
2. If our algorithm returns a labeling, it is valid.
3. If no valid labeling exists, our algorithm will return ”No”.

First, suppose that a valid labeling exists. Our DFS traversal is guaranteed to visit every connected component of the graph. For each vertex, we assign the opposite label of its parent in the DFS tree. This ensures that all rivalries are between a babyface and a heel. If a valid labeling exists, this process will find it, since it is basically the same as 2-coloring the graph. Now suppose that our algorithm returns a labeling. The DFS function only returns `True` if it can correctly assign labels to all vertices in the connected component, i.e., if a valid labeling exists. A conflict occurs when two rivals have the same label, which makes the function immediately return `False`. The main function only returns labels if DFS succeeds for all components. Therefore, if our algorithm returns a labeling, it is valid. Finally, suppose that no valid labeling exists. In this case, at some point during the DFS, there must be some point where we try to label two rivals with the same label. This will cause DFS to immediately return `False`, and the main function will return ”No”. Thus, the algorithm is correct.

4 Time Complexity Analysis

Initializing the adjacency list and label array takes $O(n)$ time. Populating the adjacency list takes $O(r)$ time. The dominant operation here is the DFS traversal. In the worst case, the DFS loop will visit all n vertices and r edges once. Each vertex is processed once when it is first discovered, and each edge processed twice (once for each endpoint). Both are $O(1)$ operations, done in $O(n)$ and $O(r)$ time, respectively. Therefore, the whole loop takes $O(n + r)$ time. In the end, we construct the lists of babyfaces and heels, which takes $O(n)$ time. Thus, the total time complexity of our algorithm is $O(n + r)$.