a. To solve this we need to start by setting the first pair to be arbitrarily a BabyFace and a Heel. We then can start a BFS at the BabyFace node. When we add a wrestler to the tree, we set them opposite to the parent. After adding a node, we check to see if any of its adjacent nodes have the same label (BabyFace or Heel), if they do, we return Impossible. Otherwise, we continue adding nodes from the list.

b. psudocode:

```
//read in list

int n; // number of wrestlers numbered 1..n

int m; // number of rivalries

int w1, w2;



create and fill matrix G with wrestlers and rivals



for (int j = 0; j < m; j++)

{

// 1st wrestler

cin >> w1;

// 2nd wrestler

cin >> w2;

w1--;

w2--;

// Add edges to adjacency matrix optional

G[w1][w2] = 1;

G[w2][w1] = 1;
```

```
}



bool result = babyfaceHeel(G, n);



if result is true, return "Possible"

else return "Impossible"

}



babyfaceheel(){

//BFS

//make vector to function as queue and another to store nodes

vector<visited> visitedNodes(n);

vector<int> queue;

//run loop to add nodes and run checks for the number in the lists

for (int i = 0; i < n; i++)

//set types and add nodes

struct visited pushin;

pushin.visit = true;



// set type opposite of current node

if (visitedNodes[current].wType == 1)

{

pushin.wType = 2;

}
```

```
else

{

pushin.wType = 1;

}

visitedNodes[i] = pushin;

queue.push_back(i);

//run check for babyface or heel affiliation

if (visitedNodes[i].wType == visitedNodes[current].wType)

return false;

}
```

c. Run time: $O(n + m)$