HW9

Daniel Ginsburg

KT 3.4

Essentially, we can model this problem similar to Testing Bipartiness, which is based on BFS and proven to be O(m+n).

<u>**Set Up:**</u>

Let each butterfly represent a node, thus there are n butterflies and n nodes. Each butterfly will be given a number 1 through n.

Let each comparison (i,j) represent an edge, however each edge has an additional property of true for same or false for different.

Use a Boolean array “Group” with n+1 nodes (we won’t use zero) where each cell is either true (for group A) or false (for group B).

<u>**Algorithm:**</u>

Pick any node at random (we will use 1 for simplicity sake) and label Group[1] true. Now do a BFS. When you get to a new edge (i,j):

If Group[j] is unlabeled then label it according to the edge’s property. If edge is true then Group[j] = Group[i], if edge is false then Group[j] = !Group[i]. Group[i] will always have a value do to the nature of BFS.

If Group[j] is not labeled, then simply check if its status of true or false matches with the edge property. (Ex. If edge is true then does Group[j] == Group[i]). If so continue, if not then j doesn’t fit because it is both in group A and group B which means that the judgements are not consistent. Return false, and exit.

Continue the BFS until completion, if it completes then the judgements are consistent, so return true.

Special case: if the graph is not connected (ex. There is a edge connecting (1,2) and (3,4) but nothing connecting 1 or 2 to 3 or 4.

When the BFS gets to a new node (3 or 4 in the example), simply label the node true and continue normally. It is arbitrary whether true or false is chosen because it only impact the specific nodes that are connected and nothing else.

Analysis:

Again, because this algorithm mimics BFS with O(1) additional work for each step (checking the edge property, labeling the node if unlabeled, or comparing the nodes if labeled) the runtime is the same as BFS which is O(m+n).