Algorithm idea:  
  
 With input of n specimen and m definitive judgments generate 2 dictionarys. One is the Graph as an adjacencylist (Note that we only connect the definitive judgments, ignore the ambiguous judgements). Another one is edgelabel dictionary which key is edge and hold value of same or different. Also we need to initialize a dictionary to specify whether or not a node is explored. Let's assume the start\_node a label "butterfly" which also can be done by generating another dictionary. Then run BFS from our start\_node to search all connected nodes. Each time we explore a node check the edgeslabel between two nodes, if it is same, give this node the same label as the node which explored this node. Otherwise give this node the label (butterfly or moth) opposite to the node explored this node. In addition, if the node found by BFS is explored before, check both edgelabel and nodelabel between this two nodes, if the target's nodelabel is same as edgelabel expect it to be. Else if the target's nodelabel violates the edgelabel, output Kiran's judgments are inconsistent.  
  
Algorithm detail:  
 # assume we have a dictionary of edgelabel which take a tuple as a key and its value to be "Same" and "Different" as input.  
 # Step 1: initalize everything in order to run BFS  
 Graph = { }  
 nodelabel = { } <--dictionary of node and its label  
 explored = { }  
 for edge in edgelabel: <-- connect the Graph with Kiran's definitive judgement (O(m))  
  
 Graph[edge[0]]=edge[1] or append edge[1]  
 Graph[edge[1]]=edge[0] or append edge[0]  
  
  
 # Step 2: BFS  
 for bug in spiecmen: <-- initialize all bug to be unexplored  
 explored[bug]=False  
 queue.append[edgelabel[0][0]] <--edgelabel[0][0] is a bug we chose as the start\_node   
 explored[edgelabel[0][0]] = True  
 nodelabel[edgelabel[0][0]] = 1 <-- label start\_node 1 (Note: 1 is Butterfly, -1 is Moth)  
  
 while queue is not empty:  
 temp = queue.pop()  
 for node in Graph[temp]:  
 if explored[node]==False:  
 queue.append(node)  
 explored[node]= True  
 if edgelabel[(temp, node)]== "Same":  
 nodelabel[node]=nodelabel[temp]  
 else:  
 nodelabel[node]=nodelabel[temp]\*-1   
 else:  
 if (nodelabel[node] == nodelabel[temp] and edgelabel[(node,temp)]== "Different")  
 or (nodelabel[node] != nodelabel[temp] and edgelabel[(node,temp)]== "Same"):  
 return "Kiran's judgements are inconsistent"  
 return Kiran's judgments are consistent  
  
Proof of Correctness idea:  
 We know that BFS can search all connected node from a node, so in the end we can go through all the node that are in the graph. With that, we can eliminate the possibility of miss a specific node. Furthermore, as I go through the graph, I give every explored node a specific label in order to clarify the type of the bugs. If there exist any violation between edgelabel and two nodes' label, we can make a final output that Kiran's judgments are inconsistent. Otherwise, her judgments is consistent.  
  
Proof Detail:  
 In order to determine Kiran’s judgments are consistent or not, I use BFS algorithm to go through all nodes that are part of the graph which mean I ignore the ambiguous judgments because that might interrupt my labelling process. As above, I have the ability go through all the definitive judgments and label the nodes just found according to the edgelabel between two nodes. So all the node should just label once that implies no node can accidentally change its label. As a consequence, once I find a node that was explored before and contradicted with the edgelabel between them, we can instantly stop the searching algorithm and give an output that Kiran’s judgments are inconsistent. Else if the algorithm run to the ending point, means her judgments are consistent since we keep track of all node’s label and didn’t find any connections are violating the edgelabel.

Runtime Analysis:  
 Step1: generate graph: O(m) time since we go through the set of edgelabel  
 Step2: because we use a pretty formatted BFS, so we know that take O(n+m) time  
 Overall: the total runtime will be O(m)+O(n+m) which is still O(n+m)