Dalton Altstaetter

360C

10/1/14

Program 1

**Part 1 - Pseudo-Code:**

---> Iterate over all tics and all tacs creating objects and storing the information in each tic and tac reference

---> From the tics object data construct the “CBG” by determining the tacs it can reach and storing them in a set. Do this by checking the upper and lower bounds against the tac ID’s

---> Finding MCM. Do so by recursively iterating over all possible matchings as long as there is a tac that is unmatched (one of my base cases). As you go, capture the current cardinality by counting the number of matched tacs. If at anytime the current number of tacs matched is greater than the previous max. Update the new max cardinality to the current number of matched tics.

---> Finding MWMCM. Next I will just find the weight of the MWMCM, which can be achieved by using the same recursive method as above except instead of capturing the maximum cardinality, capture the maximum weight. The only difference is that now we only update the maximum weight from a temporaryWeight variable when the current cardinality is equal to the max cardinality.

---> Finding MWMCM Pairs. Now that I have my maximum cardinality and maximum weight, I need to find all possible solutions that have weight and cardinality equal to my current (stored) maximums from the previous two steps. This is done using the same recursive method as in MCM. Except now I continue until I have my MWMCM conditions. When I have a current set of matchings that meet my conditions, I store all my pairings into a data structure. I continue through all possible permutations as long as there is a tac available and a tic that I haven’t checked.

---> I now have the valid matchings. To sort the orderings lexicographically I used a TreeSet which automatically orders my strings in lexicographical order. I add all valid matchings to my TreeSet to get the ordering correct.

---> Output them to the <output file>.out

---> Now it is a matter of going through all the input blocks provided repeating the procedure.

**Part 2 – Worst Case Complexity**

Since I have to find all possible matchings and there isn’t necessarily a perfect matching my algorithm is brute force recursion. The worst case occurs when every tic can be matched with every tac. That is because my recursive method returns when all tacs have been matched or I have gone through all my tics, whichever occurs first (the latter case is for when I have more tacs than tics).

Because of this my alogorithm has 3n! operations, => my worst case complexity is of order O(n!)