Lab 8

ID: 19201089
Sec: 9

Task 1

Here, we have to find the minimum spanning tree cost. First, the vertices are sorted in ascending order of cost. Then, we have to iterate through this ~~lif~~ list of sorted vertices. For the two nodes in every connection, we find their parents. This is done by calling the function 'getParent' that recursively finds the parent of the node. If the two parents are different, this means that ~~they~~ the two nodes can be joined as there is no fear of cycle. So, we set the parent of node 2 as parent 1 and parent of the node 2's parent as parent 1. Then, we append this vertice to the graph and add the cost to a totalCost variable. After the loop ends, the result is this totalCost.

Task 2

Here, we have used the concept of fibonacci to recursively get the result. We have also used the concept of caching to ensure faster execution. For every resultant number, after calculating the fibonacci, the result is stored into a dictionary. so that it didn't need to be calculated again.

Task 3

We have implemented the coin change algorithm here. First, a table has been declared with row = number of coins, or denominations and column = target+1. Then we iterate through the matrix. ~~For every~~ row ~~column, we check if the coin is~~ For the first row, if the coin is 1, simply set value of the columns to the numberings. If not, set them to the quotient ~~if~~ of numbering/coin if it is divisible. otherwise, it is infinity. For all other rows, the columns are just copies of the upper columns as long as numbering of column < coin. otherwise, it follows this formula: min(upper column, 1+ samerow[column - coin]). The last box of the whole table is the number of coins needed to make up the target.