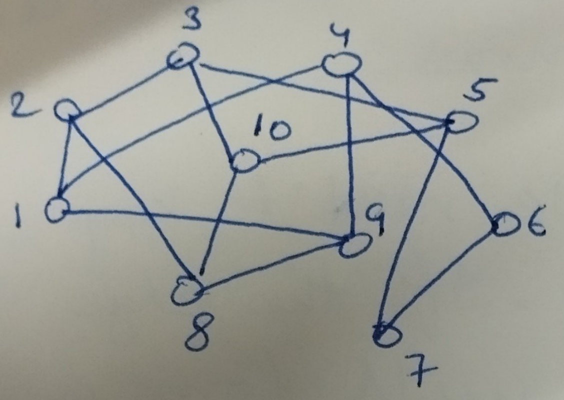
DSU

finding MCST:



(edge have wt.s too)

After staring for 15 mins:

Observation :- The edge with least wt. will always be there in the final MCST!

extension of thought :- The 2nd smallest edge will also be in the final MCST!

extend even more :- The 3rd smallest edge will be present in the final MCST only if it doesn't make a cycle with the first and second smallest edges

So.... Algorithm becomes.... :-

Go through all the edges, from least to ~~most~~ highest weights, keep the edge if it doesn't form a cycle and don't if it does.

Sub-Algorithm :- check if a given edge will ~~make~~ form a cycle.

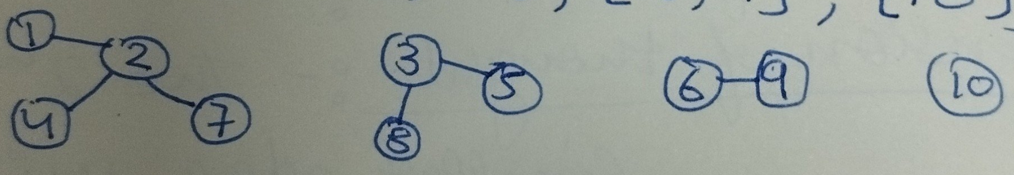
(Brute force will be bad!)

After staring for 5 mins :

Observation :- We need to keep track of the sub trees. If the edge in a sub-tree then it will form a cycle and if it's across two subtrees then it won't.

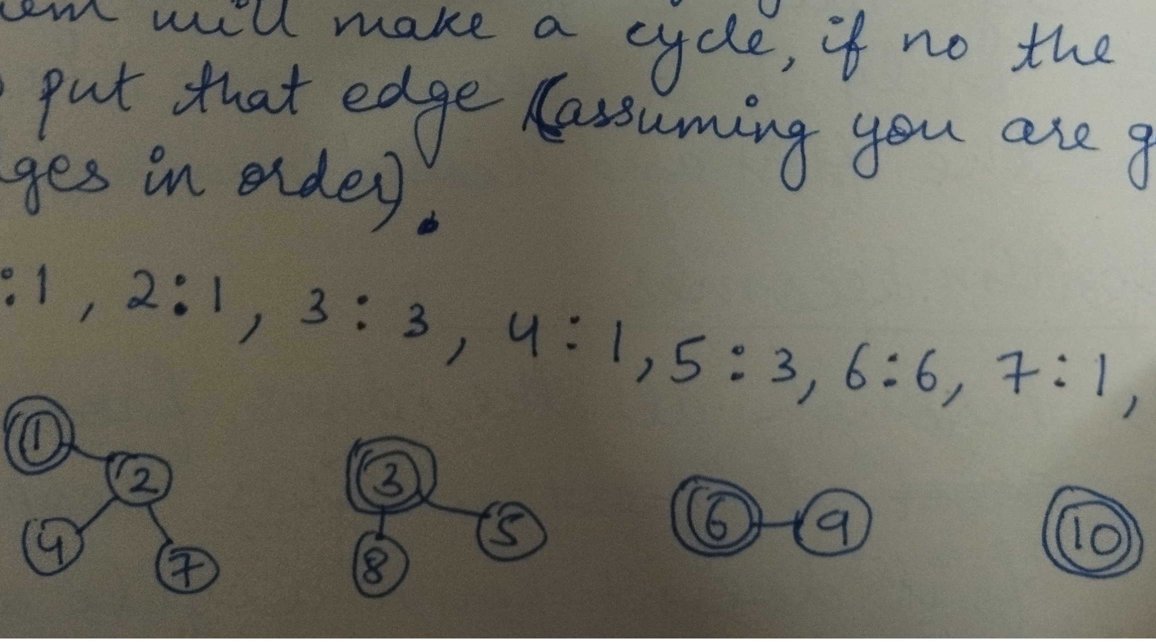
Keeping track of sub-trees :

[[1, 2, 4, 7], [3, 5, 8], [6, 9], [10]]

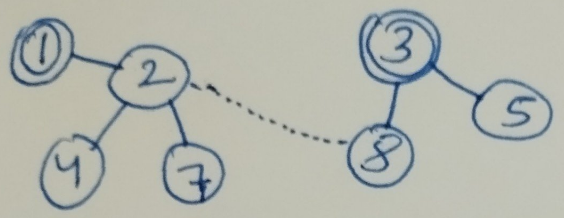


Better way :- have a sub-tree representative and just check if 2 nodes have same tree representative, if yes then the edge b/w them will make a cycle, if no the it's safe to put that edge (assuming you are going through edges in order).

{1:1, 2:1, 3:3, 4:1, 5:3, 6:6, 7:1, 8:3, 9:6, 10:10}



Joining two sub-trees :-



① Let's make node ① the representative of the new sub-tree (bcoz node ①'s sub-tree was bigger)

② updating dictionary :-

{1:1, 2:1, 3:1, 4:1, 5:1, 6:6, 7:1, 8:1, 9:6, 10:10}

Better way :-

have another dictionary to make the above updation and finding size of a sub-tree faster.

{1: [1, 2, 4, 7], 3: [3, 8, 5], 6: [6, 9], 10: [10]}

[len (value (1)) > len (value (3))]

updated : {1: [1, 2, 3, 4, 5, 7, 8], 6: [6, 9], 10: [10]}