p7-ex2

To preprocess we use a slightly modified version of usual DFS, we want in fact for each node to take care of in-time, out-time and level

First of all we initialize a variable tree_time to zero, and root.level to zero as well.

What we have so far:

```
levels = {}    //dictionary to map each level of the tree into the array of nodes at that level.

levels[0] = [root]

tree_time = 0


DFS(u):

    tree_time += 1
    u.in_time = tree_time

    for v in Adj[u]

        v.level = u.level + 1
        DFS(v)

    tree_time += 1
    u.out_time = tree_time

    if u.level not in levels
        levels[u.level] = []
    levels[u.level].append(u)
```

So the actual preprocess will be

```
preprocess(tree):

    DFS(tree)

    for level in levels
        levels[level].sort
```

```
find_kth_ancestor(a,k):
    target_level = u.level - k

    if target_level < 0
        return none

    binary search in levels[target_level] for the node with greatest in_time smaller than a.in_time
```

OBS_1
Binary search will take O(log(n)), since we sorted levels[target_level] while preprocessing.

OBS_2
Why is this correct? First of all, ancestors in_time cannot be larger than current node in_time.
Secondly, let a be the current node, suppose the ancestor of a at target_level (let’s call it u) has not the greatest in_time within the nodes at target_level with in_time smaller than a in_time.
That would mean that there exists a sibiling of u visited after u itself, but before a: this is impossible, because DFS wouldn’t visit u sibilings untill all of u children were visited.