Tree class

Create a method with the signature **public void printTree(int level)**

that performs a depth-first printing of a tree including the identification of each node and the level where the node is located; use exercises 3, 4, and 5 of the file **Course Notes Unit 5 1 Data Structures II.pdf**

Create a method with the signature **public TNode findNode(String partialKey, int where)**

that returns a pointer to the node whose identification starts with partialKey; in a sample tree with identifications from "0000" to "4500", the 'where' parameter will have this effect:

if the parameter 'where' is 0, findNode(String, int) will return a pointer to the node with smallest alphabetical identification that starts with partialKey. For example: findNode("13", 0) will return a pointer to the node with id "1300"

if parameter 'where' is 1, findNode(String, int) will return a pointer to the node with the largest alphabetical identification that starts with partialKey. For example: findNode("13", 1) will return a pointer to the node with id "1399"

if no node in the tree has an identification that starts with partialKey, then **null** is returned

suggested algorithm is on the page below

algorithm

```
    public TNode findNode(String partialKey, int where) {
        if (partialKey is a full length key)
            return findNode(partialKey)
        else if (tree is empty)
            return null
        else {
            n = length of partialKey

            if (partialKey has a lower value than the first n characters of the root's identification then) {
                declare and instantiate a tree whose root is the left child of root
                call findNode(String, int) recursively with this new tree and same parameters
            }
            else if (partialKey has a higher value than the first n characters of the root's identification then) {
                declare and instantiate a tree whose root is the right child of root
                call findNode(String, int) recursively with this new tree and same parameters
            }
            else {
                // a node has been found whose identification starts with partialKey; now we need to find the node whose identification
                // starts with partialKey and is the smallest or the largest (depending on the value of 'where')

                let p point to the root
                if (where == 0) {      // we move down the left subtree to find the smallest key that contains partialKey
                    let q point to the left child of p
                    while (q != null) {
                        if (q's identification contains partialKey)
                            let p point to the node that q points to
                            let q point to the left child of q
                        }
                        else {
                            let q point to the right child of q
                        }
                    }
                }
                else {
                    let q point to the right child of p
                    while (q != null) {
                        if (q's identification contains partialKey) {
                            let p point to the node that q points to
                            let q point to the right child of q
                        }
                        else {
                            let q point to the left child of q
                        }
                    }
                }
                return p;
            }
        }
    }
```