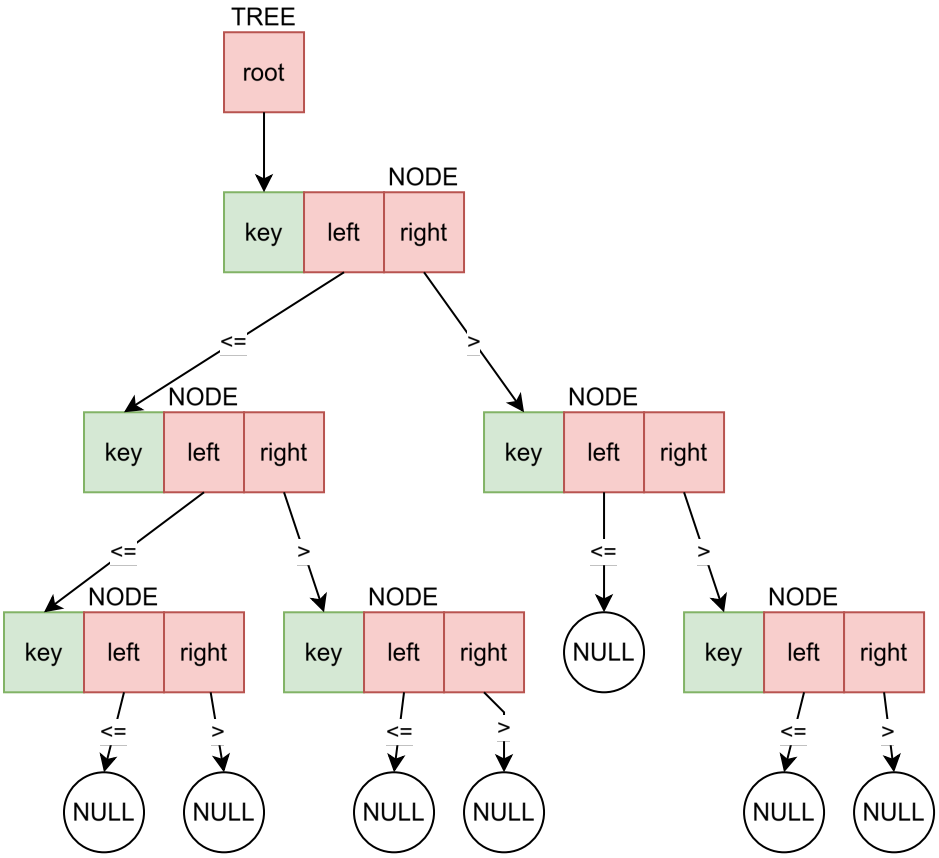
Lab 3 - Binary Search Trees

Implement a binary search tree using a structure similar to the following example:

```
typedef struct _NODE{
    int key;
    struct _NODE* left;
    struct _NODE* right;
} NODE;

typedef struct {
    NODE* root;
} BST;
```



Implement a function for each of the following operations:

1. Tree initialisation (create an empty tree) ★
2. Insert node ★
3. Search node by key. If the element is found return a pointer to the node, otherwise return 0. ★
4. Delete node by key ★★
5. Tree deinitialisation (free the memory allocated for the nodes and the tree) ★

6. Print the nodes in preorder ★
7. Print the nodes in inorder ★
8. Print the nodes in postorder ★
9. Get the number of nodes in the tree ★★
10. Get the number of leaf nodes in the tree ★★
11. Get the depth of a node given the node's key (the number of edges from the root node to the node) ★★
12. Get the height of a node given the node's key (the largest number of edges from the node to a leaf node) ★★
13. Find the n-th largest element in the tree ★★★
14. Find the n-th smallest element in the tree ★★★
15. Merge 2 trees ★★
16. Merge 2 trees without allocating any additional memory ★★★

Hard mode: Solve the lab problems using the containing record trick:

```
#define CONTAINING_RECORD(address, type, field) (\
    (type *)((char*)(address) - (size_t)(&((type *)0)->field)))
```

Note: Leave a comment with the text PB1, PB2, ... PB10 above every function that implements the respective lab task. (upper case text, no space between the text and the problem number)