Data Structure PA

電機四 B08901123 何景盛

思路:

我把 input file 內的 postorder 存為 vector<vector<int>> data = {[tree1 postorder], [tree2 postorder], …}來區分每棵 tree。這次主要為兩個 class Node 和 BST，Node 是一些 tree node 的 attribute，例如 left child, right child 和 key，BST 是用來建立 tree 或者是一些其他的操作。

```
class Node
{
public:
    Node *left;
    Node *right;
    int key;

    Node() : left(0), right(0), key(0){};
    Node(int a) : left(0), right(0), key(a){};
};
```

class BST 內的主要的五個 method:

```
Node* constructBST(vector<int> postorderArray, int start, int end){
    if (start > end) return NULL;
    Node* node = new Node(postorderArray[end]);
    int i;
    for (i = end; i >= start; i--){
        if (postorderArray[i] < node->key) break;
    }
    node->right = constructBST(postorderArray, i+1, end-1);
    node->left = constructBST(postorderArray, start, i);
    return node;
}
```

constructBST(vector<int> postorderArray, int start, int end):

此 method 是收到一個 postorder vector 後把它變成 binary search tree，start 和 end 是 index 代表 postorder vector 只看的範圍。一開始 postorder 最後的 element 是 root，然後就往前找第一個比這個 root 小的 element，在它和它之前的 elements 都是 left subtree，其他的 elements 是 right subtree，如此遞迴下去就會把 binary search tree 建起來。

```
void preorder(Node* root, vector<int>& preorderArray){
    if (root == NULL) return;
    preorderArray.push_back(root->key);
    preorder(root->left, preorderArray);
    preorder(root->right, preorderArray);
}
```

preorder(Node* root, vector<int>& preorderArray):
root 代表此 binary search tree 的根，而 preorderArray 是用來記錄 preorder 序列用的 vector。如果 root 是 NULL 就什麼都不做，如果不是的話，就先把 node 的 key push 進去 vector 裡，然後就遞迴 left subtree 和 right subtree。

```
int treeHeight(Node* root){
    if (root == NULL) return 0;
    int leftSubtreeHeight = treeHeight(root->left);
    int rightSubtreeHegiht = treeHeight(root->right);
    return max(leftSubtreeHeight, rightSubtreeHegiht) + 1;
}
```

treeHeight(Node* root):
tree height 為 left subtree height 和 right subtree height 兩者其中最大的+1，然後遞迴下去。

```
vector<int> findLevelMaxValue(Node* root){
    vector<int> ans;
    findLevelMaxValue_r(ans, root, 0);
    return ans;
}
```

```
void findLevelMaxValue_r(vector<int>& ans, Node* root, int h){
    if (root == NULL) return;
    if (ans.size() == h) ans.push_back(root->key);
    else{
        ans[h] = max(ans[h], root->key);
    }
    findLevelMaxValue_r(ans, root->left, h+1);
    findLevelMaxValue_r(ans, root->right, h+1);
}
```

findLevelMaxValue(Node* root), findLevelMaxValue_r(vector<int>& ans, Node* root, int h):
這兩個 method 是一起用的，findLevelMaxValue 是設定一些起始條件，findLevelMaxValue_r 則是進行遞迴。ans 是用來存每層最大值的 vector，index 為層數，所以 ans[h]為 h 層的最大值，h 是來記錄層數用。主要是用 preorder 的方式來看 tree 的每個 node，但在每次遞迴時都會把層數放到 input argument，所以我們可以比較新的 node key 和 ans 內原本的值，從而更新 ans

No collaborator

Reference:

https://www.geeksforgeeks.org/construct-a-binary-search-tree-from-given-postorder/

https://www.baeldung.com/cs/binary-tree-height