Phill-DS-0117

Leetcode 104

計算一棵二元樹的最深深度

- linked list
- 以下面的程式為基礎

```
#include <iostream>
using namespace std;

struct TreeNode{
  int data;
  TreeNode* llink; //左子樹 link
  TreeNode* rlink;

  TreeNode(int value): data(value), llink(NULL), rlink(NULL){}
};

class BinaryTree{
  private:
    TreeNode* root;

  public:
    BinaryTree() : root(NULL){}

    void insert(int data){
      if(root ==NULL){ //樹是空的
        root = new TreeNode(data);
        return;
      }

      TreeNode* temp = root;
      while(true){
        if(data < temp->data){ //插入的值比魁儡變數值小 -> 向左走
          if(temp->llink==NULL){ //左是空
            temp->llink = new TreeNode(data); //直接assign成左noc
            return;
          }
          temp = temp->llink; //推移魁儡變數
        }
        else{
          if(temp->rlink == NULL){
            temp->rlink = new TreeNode(data);
            return;
          }
          temp = temp->rlink;
        }
      }
    }

    TreeNode *getRoot(){ //get root 實體 -> getter
      return root;
    }

    ~BinaryTree(){
      destroyTree(root);
    }

  private:
    void destroyTree(TreeNode *node){
      if(node != NULL){ //不要 free 掉空tree
        destroyTree(node->llink);
        destroyTree(node->rlink);
        delete node;
      }
    }
};

int main()
```

```
{
    BinaryTree tree;

    tree.insert(5);
    tree.insert(3);
    tree.insert(7);
    tree.insert(2);
    tree.insert(4);
    tree.insert(6);
    tree.insert(8);
    return 0;
}
```

Phill

```
private:

      int  countNodes(TreeNode *node){
        if(node !=NULL){
        int leftNum= 1+countNodes(node->llink);
        int rightNum= 1+countNodes(node->rlink);
        int treeNum= max(leftNum,rightNum);
        return treeNum;

        }
      }
    public:
      int MaxDepth(TreeNode *node){

        return countNodes(node);
        }
};
```

Traverse Tree

- in-order traverse → left subtree, self(root), right subtree
- preorder → self, left, right
- postorder → left, right, self

DFS → search based on Depth-First traverse

- 印出 DFS traverse 結果
- DFS_inorder(), DFS_preorder(), DFS_postorder()

```
public:

      int  traversal_inorder(TreeNode *node){
        if(node !=NULL){

        traversal_inorder(node->llink);
        cout << node->data;
        traversal_inorder(node->rlink);

        }
      }
       int  traversal_preorder(TreeNode *node){
        if(node !=NULL){
        cout << node->data;
        traversal_inorder(node->llink);
        traversal_inorder(node->rlink);

        }
      }

      int  traversal_postorder(TreeNode *node){
        if(node !=NULL){

        traversal_inorder(node->llink);
        traversal_inorder(node->rlink);
        cout << node->data;
        }
      }

======

int main()
{
    BinaryTree tree;
    tree.insert(5);
    tree.insert(3);
    tree.insert(7);
    tree.insert(2);
    tree.insert(4);
    tree.insert(6);
    tree.insert(8);
    cout << "\n Inorder traversal of binary tree is \n";
    tree.traversal_inorder(tree.getRoot());
    cout << "\n preorder traversal of binary tree is \n";
    tree.traversal_preorder(tree.getRoot());
    cout << "\n postorder traversal of binary tree is \n";
    tree.traversal_postorder(tree.getRoot());
    return 0;
}
```