Báo cáo bài tập tuần 7

Họ và tên: Trần Hoàng Nhi

MSSV: 24120113

1 Binary Tree

Each node in a binary tree is defined with the following structure:

```
struct NODE{ int key;
      NODE* p_left;
      NODE* p_right;
};
```

Students are required to implement the following functions:

1. Initialize a NODE from a given value:
    - NODE* createNode(int data)

    Cấp phát bộ nhớ cho một `NODE`.

    Gán khóa `key = data`.

    Khởi tạo con trái và phải là `NULL`.

    Trả về con trỏ nút mới tạo.

2. Các hàm duyệt cây (NLR, LNR, LRN)

Sử dụng đệ quy, theo thứ tự:

NLR: Xử lý nút hiện tại, đệ quy vào con trái, đệ quy vào con phải.

LNR: Đệ quy con trái, xử lý nút, đệ quy con phải.

LRN: Đệ quy con trái, đệ quy con phải, xử lý nút.

Mỗi lần xử lý nút, thêm key vào vector kết quả.

3. Perform a level-order traversal:
    - vector<vector<int>> LevelOrder(NODE* pRoot)

Sử dụng queue để duyệt rộng theo chiều rộng.

Đưa nút gốc vào queue.

Lặp qua từng tầng:

Lấy số lượng nút ở tầng hiện tại.

Duyệt từng nút, lấy giá trị, thêm con trái và phải (nếu có) vào queue.

Lưu giá trị các nút tầng hiện tại vào vector kết quả.

4. Calculate the number of NODEs in a binary tree:
    - int countNode(NODE* pRoot)

Đệ quy: nếu nút NULL trả về 0.

Nếu không, trả về 1 cộng số nút con trái và con phải.

5. Calculate the sum of all NODE values in a binary tree:
    - int sumNode(NODE* pRoot)

Đệ quy tương tự hàm đếm nút.

Trả về key nút hiện tại cộng tổng con trái và con phải.

6. Calculate the height of a NODE with a given value: *(return -1 if not found)*
    - heightNode(NODE* pRoot, int value)
7. * Calculate the level of a given NODE:
    - int Level(NODE* pRoot, NODE* p)

Sử dụng hàng đợi (queue) để duyệt theo chiều rộng (Level Order).

Tạo biến level ban đầu bằng 1 để đánh dấu tầng hiện tại.

Đưa root vào queue, sau đó thực hiện vòng lặp

Với mỗi tầng, lấy số lượng node hiện có trong queue (size).

Duyệt qua các node ở tầng đó:

Nếu node hiện tại có giá trị bằng x, trả về level.

Nếu có con trái/phải, đẩy vào queue để duyệt ở tầng sau.

Sau mỗi vòng lặp, tăng level lên 1.

Nếu không tìm thấy x trong cây, trả về -1.* Count the number of leaf nodes in a binary tree:

- int countLeaf(NODE* pRoot)

Sử dụng duyệt theo tầng bằng `queue`.

Đưa nút gốc vào queue.

Lặp từng tầng, nếu gặp nút có giá trị cần tìm, trả về tầng hiện tại.

2 Binary Tree - Binary Search Tree (BST)

Each node in a binary (search) tree is defined with the following structure:

```
struct NODE{ int key;
        NODE* p_left;
        NODE* p_right;
};
```

Students are required to implement the following functions:

1. Find and return a NODE with a specified value from a binary search tree

    - NODE* Search(NODE* pRoot, int x)

Dùng vòng lặp hoặc đệ quy:

Nếu root là NULL hoặc root->key == x, trả về root.

Nếu x nhỏ hơn root->key, tìm ở cây con trái.

Ngược lại tìm ở cây con phải.

2. Add a NODE with a specified value to a binary search tree:

    - void Insert(NODE* &pRoot, int x)

Nếu cây rỗng, tạo nút mới.

Nếu x < root->key, đệ quy chèn vào con trái.

Nếu x > root->key, đệ quy chèn vào con phải.

Nếu x trùng, không thêm (BST không chứa khóa trùng).

3. Delete a NODE with a given value from a binary search tree:

    - void Remove(NODE* &pRoot, int x)

Tìm nút cần xóa bằng tìm kiếm.

Nếu nút là lá, xóa trực tiếp.

Nếu nút có 1 con, thay nút đó bằng con đó.

Nếu nút có 2 con, thay thế giá trị nút bằng giá trị nhỏ nhất trong cây con phải (hoặc lớn nhất cây con trái), rồi xóa nút thay thế.

4. Initialize a binary search tree from a given array:

    - NODE* createTree(int a[], int n)

Khởi tạo nút gốc và dùng hàm insert để thêm từng nút từ mảng vào cây

5. Calculate the height of a binary search tree:

    - int Height(NODE* pRoot)

Gọi lại hàm maxDepth, trả về độ sâu lớn nhất từ root đến lá.

6. * Hàm đếm số nút nhỏ hơn/ lớn hơn 1 giá trị:

Đệ quy:

Nếu root == NULL, trả về 0.

So sánh root->key với x, đếm nếu thỏa mãn.

Đệ quy ở con trái và phải.

7. * Determine if a binary tree is a binary search tree:

   - bool isBST(NODE* pRoot)

Đệ quy với phạm vi giá trị hợp lệ (minValue, maxValue).

Nếu nút nằm ngoài phạm vi, trả về false.

Kiểm tra con trái với phạm vi mới (min, key - 1).

Kiểm tra con phải với phạm vi mới (key + 1, max).

8. * Check if a binary tree is a full binary search tree (BST):

   - bool isFullBST(NODE* pRoot)

Đệ quy:

Nếu nút là lá, trả về true.

Nếu nút có cả con trái và phải, đệ quy kiểm tra hai cây con.

Nếu nút chỉ có một con, trả về false.

isFullBST kết hợp kiểm tra isBST và isFullTree.

3 AVL Tree

Each node of an AVL tree is defined as follows:

```
struct NODE{ int key;
    NODE* p_left;
```

```
    NODE* p_right;
    int height;
};
```

---

1. Insert a NODE with a given value into a given AVL tree (Note that the given value may already exist):
    - void Insert(NODE* &pRoot, int x)

Thêm nút mới như BST.

Sau khi thêm, tính hệ số cân bằng của nút.

Nếu mất cân bằng, tùy thuộc vào 4 trường hợp:

Left Left (quay phải).

Left Right (quay trái + quay phải).

Right Right (quay trái).

Right Left (quay phải + quay trái).

Trả về nút đã cân bằng.

2. Delete a NODE with a given value from a given AVL tree (Note that the value may not exist):
    - void Remove(NODE* &pRoot, int x)

Xóa nút như BST.

Cập nhật chiều cao.

Kiểm tra và cân bằng lại tương tự hàm Insert.

3. * Determine if a binary tree is an AVL tree:
    - bool isAVL(NODE* pRoot)

Kiểm tra đồng thời:

Cây là BST (dùng isBST).

Mọi nút đều có hệ số cân bằng nằm trong [-1, 1].

Đệ quy kiểm tra toàn bộ cây.

4 Git/Github