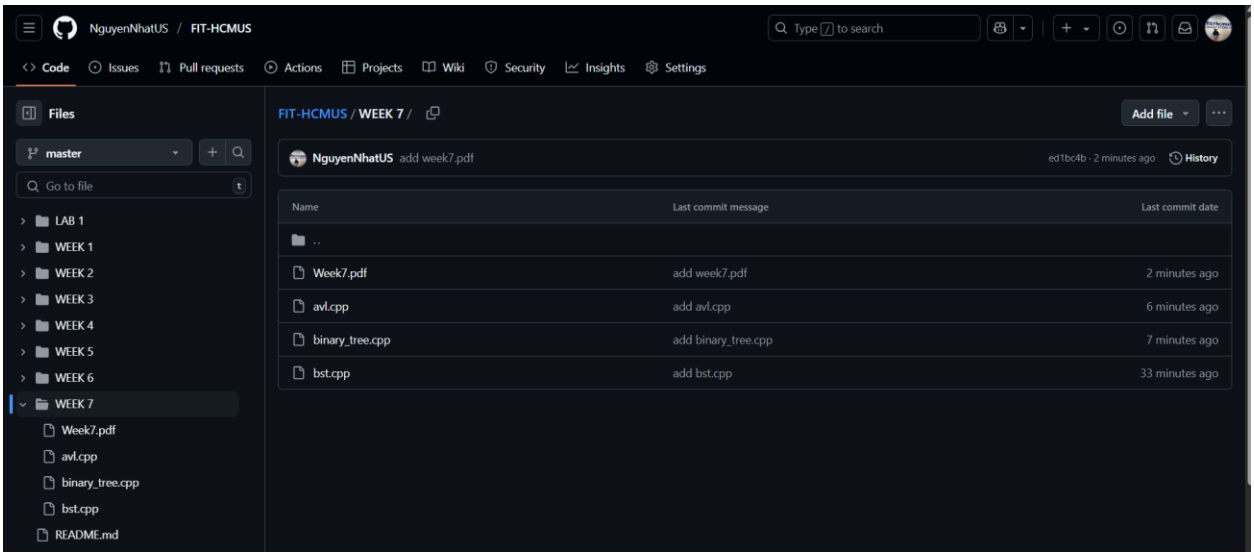
Nguyễn Lê Đức Nhật

24120403



Phần 1: Binary Tree

1, Node* createNode(int data)

Cấp phát động newNode

Gán newNode->key = data

newNode->left = newNode->right = NULL;

Return newNode;

2, vector<int> NLR(Node* root_

Duyệt **trước (NLR)**: Gốc → Trái → Phải

Đệ quy: lưu giá trị nút hiện tại trước, sau đó đệ quy trái và phải.

3, vector<int> LNR(Node* root)

Duyệt giữa (LNR): Trái → Gốc → Phải

Dùng đệ quy: vào trái, lưu nút hiện tại, sang phải.

4, vector<int> LRN(Node* root)

Duyệt sau (LRN): Trái → Phải → Gốc

Đệ quy trái rồi phải, cuối cùng lưu nút hiện tại.

5, vector<vector<int>> LevelOrder(Node* root)

Duyệt cây theo từng tầng (bfs).

Dùng hàng đợi queue<Node*> để duyệt từng tầng từ trái sang phải.

Mỗi tầng tạo 1 vector<int> và thêm vào res.

6, int countNode(Node* root)

Trả về tổng số nút trong cây.

Đệ quy: mỗi nút tính là 1 + tổng nút bên trái + tổng nút bên phải.

7, int sumNode(Node* root)

Trả về tổng key của tất cả các nút.

Đệ quy: nút hiện tại + tổng trái + tổng phải.

8, int getHeight(Node* root)

Trả về chiều cao của cây.

Chiều cao = 1 + max(chiều cao trái, chiều cao phải).

Nếu cây rỗng, trả về -1 (nhiều tài liệu dùng -1 thay vì 0).

9, int heightNode(Node* root, int value)

Tìm nút có key == value, rồi trả về chiều cao của cây tại nút đó.

Duyệt theo tầng đến khi gặp nút đúng thì gọi getHeight(node)

10, int Level(Node* root, Node* p)

Trả về cấp (level) của nút p trong cây (gốc = 0).

Duyệt BFS, mỗi lần qua tầng thì tăng currLevel.

11, int countLeaf(Node* root)

Duyệt BFS, nếu 1 nút không có con trái và phải thì là lá.

Đếm số lượng nút thỏa mãn điều kiện này.

Phần 2: Binary Search Tree

1,. Node* Search(Node* root, int x)

Tìm kiếm phần tử x trong BST.

So sánh x với key:

Nếu x < key → tìm bên trái.

Nếu x > key → tìm bên phải.

Nếu bằng → trả về node.

2, void Insert(Node*& root, int x)

Duyệt cây theo nguyên tắc BST:

Nếu x < key → đi trái.

Nếu x > key → đi phải.

Nếu tới vị trí NULL thì gán node mới vào.

3, Node* getSucc(Node* root)

Trả về node nhỏ nhất bên cây con phải → successor.

Node* deleteNode(Node* root, int x)

Xóa node có giá trị x trong cây.

3 trường hợp:

Không có con → xóa node.

Một con → trả về node con.

Hai con → tìm successor, gán key, xóa successor.

void Remove(Node*& root, int x)

Gọi deleteNode và cập nhật lại root.

4, Node* createTree(int a[],int n)

Vòng lặp for chèn từng phần tử vào root rồi return root.

5, void removeTree(Node* root)

Duyệt cây theo postorder và xóa từng node.

6, int Height(Node* root)

Trả về chiều cao của cây, định nghĩa:

NULL → -1

node lá → 0

công thức: 1 + max(height(left), height(right))

7, int countNode(Node* root)

Đếm tổng số node trong cây.

Nếu là null thì return 0.

Công thức 1 + countNode(root->left) + countNode(root->right)

8, int countLess(Node* root, int x)

Đếm số node có giá trị nhỏ hơn x.

Duyệt theo inorder và kiểm tra nếu nó thỏa điều kiện thì + 1

9, int countGreater(Node* root, int x)

Đếm số node có giá trị lớn hơn x.

Duyệt theo inorder và kiểm tra nếu nó thỏa điều kiện thì + 1

10, bool isBST(Node* root)

Dựa vào tính chất 1 BST nếu duyệt theo inorder sẽ được mảng tăng dần, ta duyệt inorder và push vào 1 vector sau đó kiểm tra đây có phải 1 vector tăng chặt hay không

11, isFullBST(Node* root)

Là 1 BST và mỗi node chỉ có 0 hoặc 2 con, duyệt theo Level và kiểm tra

Phần 3: AVL Tree

1, Node* createNode(int data)

Trả về node mới có key = data, height = 1.

2, getHeight: Trả về chiều cao của node (0 nếu NULL).

3,getBalance: Trả về độ lệch chiều cao (left - right) tại node.

4, rightRotate(y)

Dùng khi cây lệch trái quá mức (Left-Left case).

Dịch cây con trái lên làm gốc, cập nhật chiều cao lại.

5, leftRotate(x)

Dùng khi cây lệch phải quá mức (Right-Right case).

Dịch cây con phải lên làm gốc.

6, Node* insert(Node*& root, int x)

Chèn node mới vào như BST.

Sau khi chèn xong:

Cập nhật height của node.

Tính balance.

Tùy từng trường hợp mất cân bằng, thực hiện xoay:

Left-Left (LL) → rightRotate

Right-Right (RR) → leftRotate

Left-Right (LR) → leftRotate rồi rightRotate

Right-Left (RL) → rightRotate rồi leftRotate

Insert(Node*& root, int x)

Gọi insert() và cập nhật lại gốc.

7, deleteNode(Node*& root, int x)

Tương tự BST:

Tìm node cần xoá.

Nếu có 0 hoặc 1 con → xoá thẳng.

Nếu có 2 con → tìm successor, gán key, xoá successor.

Sau khi xoá:

Cập nhật height.

Tính balance.

Thực hiện xoay nếu cần (như phần chèn).

Remove(Node*& root, int x)

Gọi deleteNode() để xoá và cập nhật gốc.

8,bool isAVL(Node* root)

Kiểm tra:

Có là BST không (isBST)

Có cân bằng chiều cao tại mọi node không (checkIfAVL)