TRƯỜNG ĐẠI HỌC SÀI GÒN

KHOA CÔNG NGHỆ THÔNG TIN



BÁO CÁO

HỌC PHẦN: CẤU TRÚC DỮ LIỆU & GIẢI THUẬT

Bài tập: Cây Nhị Phân Và Tìm Kiếm

|  |  |
| --- | --- |
| Sinh viên thực hiện | MSSV |
| Vũ Bình Phước | 3121411171 |

Giảng viên môn học: ĐỖ NHƯ TÀI

**Bài 1:**

**Test Case 1: Cây có cả số dương và số âm**

**Input:**

BinaryTree tree;

tree.insert(5);

tree.insert(-3);

tree.insert(8);

tree.insert(-7);

tree.insert(2);

tree.insert(-1);

tree.insert(10);

**Output mong đợi:**

Trung bình cộng tất cả các nút: 2.0

Trung bình cộng các số dương: 6.25

Trung bình cộng các số âm: -3.67

Tỉ số R (tổng số dương / tổng số âm): 2.36

**Test Case 2: Cây chỉ có số dương**

**Input:**

BinaryTree tree;

tree.insert(1);

tree.insert(3);

tree.insert(5);

tree.insert(7);

tree.insert(9);

**Output mong đợi:**

Trung bình cộng tất cả các nút: 5.0

Trung bình cộng các số dương: 5.0

Trung bình cộng các số âm: 0

Tỉ số R (tổng số dương / tổng số âm): 0

**Test Case 3: Cây chỉ có số âm**

**Input:**

BinaryTree tree;

tree.insert(-2);

tree.insert(-4);

tree.insert(-6);

tree.insert(-8);

**Output mong đợi:**

Trung bình cộng tất cả các nút: -5.0

Trung bình cộng các số dương: 0

Trung bình cộng các số âm: -5.0

Tỉ số R (tổng số dương / tổng số âm): 0

**Test Case 4: Cây có cả số dương và âm nhưng số dương lớn hơn**

**Input:**

BinaryTree tree;

tree.insert(10);

tree.insert(20);

tree.insert(30);

tree.insert(-5);

tree.insert(-15);

**Output mong đợi:**

Trung bình cộng tất cả các nút: 8.0

Trung bình cộng các số dương: 20.0

Trung bình cộng các số âm: -10.0

Tỉ số R (tổng số dương / tổng số âm): 2.0

**Test Case 5: Cây chỉ có một phần tử**

**Input:**

BinaryTree tree;

tree.insert(7);

**Output mong đợi:**

Trung bình cộng tất cả các nút: 7.0

Trung bình cộng các số dương: 7.0

Trung bình cộng các số âm: 0

Tỉ số R (tổng số dương / tổng số âm): 0

**Bài 2:**

**Test Case 1: Cây từ Hình 1 → Hình 2 (Cây chuẩn theo đề bài)**

**Input:**

BinaryTree tree;

tree.buildTree();

tree.transformTree();

**Output mong đợi:**

Cây ban đầu: 3 5 7 9 10 12 15 18 20

Cây sau khi biến đổi: 12 15 18 20 10 3 5 7 9

**Test Case 2: Cây có ít nút (chỉ có 3 nút)**

**Input:**

BinaryTree tree;

tree.insert(tree.root, 10);

tree.insert(tree.root, 5);

tree.insert(tree.root, 15);

tree.transformTree();

**Output mong đợi:**

Cây ban đầu: 5 10 15

Cây sau khi biến đổi: 15 10 5

**Test Case 3: Cây chỉ có một nút duy nhất (gốc không thay đổi)**

**Input:**

BinaryTree tree;

tree.insert(tree.root, 10);

tree.transformTree();

**Output mong đợi:**

Cây ban đầu: 10

Cây sau khi biến đổi: 10

**Test Case 4: Cây có nhiều nút hơn nhưng vẫn áp dụng hoán đổi gốc**

**Input:**

BinaryTree tree;

tree.insert(tree.root, 30);

tree.insert(tree.root, 20);

tree.insert(tree.root, 40);

tree.insert(tree.root, 10);

tree.insert(tree.root, 25);

tree.insert(tree.root, 35);

tree.insert(tree.root, 50);

tree.transformTree();

**Output mong đợi:**

Cây ban đầu: 10 20 25 30 35 40 50

Cây sau khi biến đổi: 35 40 50 30 10 20 25

**Bài 3:**

1. Xác định thứ tự các nút được thêm vào cây nhị phân tìm kiếm (BST)

10

/ \

5 15

/ \ / \

3 7 12 18

\ \

9 20

- Gốc cây là 10.

- Các nút nhỏ hơn 10 được thêm vào bên trái, các nút lớn hơn 10 được thêm vào bên phải.

- Tương tự áp dụng cho các nút con.

Thứ tự thêm vào có thể là

1. 10 (gốc)

2. 5 (nhỏ hơn 10, thêm vào trái)

3. 15 (lớn hơn 10, thêm vào phải)

4. 3 (nhỏ hơn 10, nhỏ hơn 5, thêm vào trái của 5)

5. 7 (nhỏ hơn 10, lớn hơn 5, thêm vào phải của 5)

6. 12 (lớn hơn 10, nhỏ hơn 15, thêm vào trái của 15)

7. 18 (lớn hơn 10, lớn hơn 15, thêm vào phải của 15)

8. 9 (nhỏ hơn 10, lớn hơn 5, lớn hơn 7, thêm vào phải của 7)

9. 20 (lớn hơn 10, lớn hơn 15, lớn hơn 18, thêm vào phải của 18)

Thứ tự thêm vào 10, 5, 15, 3, 7, 12, 18, 9, 20.

2. Xác định phép duyệt cây

Kết quả duyệt cây là: 3, 7, 9, 5, 12, 20, 18, 15, 10.

- In-order (LNR): Duyệt trái - gốc - phải. Kết quả sẽ là dãy tăng dần (đối với BST). Không phù hợp.

- Pre-order (NLR): Gốc - trái - phải. Kết quả bắt đầu bằng gốc (10). Không phù hợp.

- Post-order (LRN): Duyệt trái - phải - gốc. Kết quả sẽ kết thúc bằng gốc (10). Phù hợp với dãy đã cho.

- Level-order: Duyệt theo từng tầng. Kết quả sẽ bắt đầu bằng gốc (10). Không phù hợp.

Kết luận: Phép duyệt được sử dụng là Post-order (LRN).

***BÀI 4:***

**Test Case 1: Cây BST bình thường**

/\*

10

/ \

5 15

\ / \

9 12 20

/

18

\*/

Node\* root = new Node(10);

root->left = new Node(5);

root->right = new Node(15);

root->left->right = new Node(9);

root->right->left = new Node(12);

root->right->right = new Node(20);

root->right->right->left = new Node(18);

Kết quả mong đợi:

Min: 5

Max: 20

```

**Test Case 2: Cây BST chỉ có một nút**

/\*

42

\*/

Node\* root = new Node(42);

Kết quả mong đợi:

Min: 42

Max: 42

```

**Test Case 3: Cây BST lệch trái (left-skewed)**

/\*

50

/

30

/

20

/

10

\*/

Node\* root = new Node(50);

root->left = new Node(30);

root->left->left = new Node(20);

root->left->left->left = new Node(10);

Kết quả mong đợi:

Min: 10

Max: 50

```

**Test Case 4: Cây BST lệch phải (right-skewed)**

/\*

10

\

20

\

30

\

40

\*/

Node\* root = new Node(10);

root->right = new Node(20);

root->right->right = new Node(30);

root->right->right->right = new Node(40);

Kết quả mong đợi:

Min: 10

Max: 40

```

**Test Case 5: Cây BST rỗng**

Node\* root = nullptr;

Kết quả mong đợi:

Chương trình sẽ in thông báo "Cây rỗng!" và trả về -1 (hoặc throw exception)

```

**Test Case 6: Cây BST với các giá trị âm**

/\*

-10

/ \

-20 0

\ / \

-5 -1 10

\*/

Node\* root = new Node(-10);

root->left = new Node(-20);

root->right = new Node(0);

root->left->right = new Node(-5);

root->right->left = new Node(-1);

root->right->right = new Node(10);

Kết quả mong đợi:

Min: -20

Max: 10

```

***BÀI 5 :***

**Test Case 1**: Cây có nhiều nút một con.

1

/ \

2 3

\ \

4 5

/

6

**Kết quả mong đợi**: 3

**Test Case 2**: Cây chỉ có một nút gốc.

10

**Kết quả mong đợi**: 0

**Test Case 3**: Cây có tất cả các nút đều có hai con.

1

/ \

2 3

/ \ / \

4 5 6 7

**Kết quả mong đợi**: 0

**Test Case 4**: Cây rỗng.

(Không có nút nào)

**Kết quả mong đợi**: 0

**Test Case 5**: Cây có dạng chuỗi (tất cả các nút chỉ có một con phải).

1

\

2

\

3

**Kết quả mong đợi**: 3

**Test Case 6**: Cây có dạng chuỗi (tất cả các nút chỉ có một con trái).

1

/

2

/

3

\*\*Kết quả mong đợi\*\*: `3`

***Bài 6:***

**Test Case 1: Cây có nhiều nút một con**

Cây:

1

/ \

2 3

\ \

4 5

/

6

**Số nút có một con**: 3

**Số nút trong cây**: 6

**Tổng giá trị các nút**: 1 + 2 + 3 + 4 + 5 + 6 = 21

**Test Case 2: Cây chỉ có một nút gốc**

Cây:

10

**Số nút có một con**: 0

**Số nút trong cây**: 1

**Tổng giá trị các nút**: 10

**Test Case 3: Cây có tất cả các nút đều có hai con**

Cây:

1

/ \

2 3

/ \ / \

4 5 6 7

**Số nút có một con**: 0

**Số nút trong cây**: 7

**Tổng giá trị các nút**: 1 + 2 + 3 + 4 + 5 + 6 + 7 = 28

**Test Case 4: Cây rỗng**

Cây:

(Không có nút nào)

**Số nút có một con**: 0

**Số nút trong cây**: 0

**Tổng giá trị các nút**: 0

**Test Case 5: Cây có dạng chuỗi (tất cả các nút chỉ có một con phải)**

Cây:

1

\

2

\

3

\

4

**Số nút có một con**: 3

**Số nút trong cây**: 4

**Tổng giá trị các nút**: 1 + 2 + 3 + 4 = 10

**Test Case 6: Cây có dạng chuỗi (tất cả các nút chỉ có một con trái)**

Cây:

1

/

2

/

3

/

4

**Số nút có một con**: 3

**Số nút trong cây**: 4

**Tổng giá trị các nút**: 1 + 2 + 3 + 4 = 10

***Bài 7:***

Giống nhau:

CTDL động.

Các thao tác cơ bản Thêm, Xóa, Cập Nhật được thực hiện một

cách linh hoạt.

Khác nhau

Dữ liệu trên cây NPTK được tổ chức và dslk đơn thì không.

Chi phí tìm kiếm, thêm trên cây nhanh hơn trên dslk đơn.

***BÀI 8:***

**Test Case 1: Cây nhỏ với 3 nút**

Cấu trúc cây:

2.5

/ \

1.2 3.8

* **Dữ liệu xuất ra (LNR - Tăng dần):** 1.2 2.5 3.8
* **Tập tin đầu ra:** data1.out

**Test Case 2: Cây lớn hơn với 5 nút**

Cấu trúc cây:

5.0

/ \

2.3 7.8

/ \

1.1 3.4

* **Dữ liệu xuất ra (LNR - Tăng dần):** 1.1 2.3 3.4 5.0 7.8
* **Tập tin đầu ra:** data2.out

***BÀI 9:***

**Test Case 1: Cây nhỏ với 3 nút**

Cấu trúc cây:

2.5

/ \

1.2 3.8

* **Thứ tự duyệt NLR (Pre-order)**: 2.5 1.2 3.8
* **Tập tin đầu ra**: data1.out

**Test Case 2: Cây lớn hơn với 5 nút**

Cấu trúc cây:

5.0

/ \

2.3 7.8

/ \

1.1 3.4

* **Thứ tự duyệt NLR (Pre-order)**: 5.0 2.3 1.1 3.4 7.8
* **Tập tin đầu ra**: data2.out

***BÀI 10:***

**Test Case 1: Cây nhỏ với 3 nút**

Cấu trúc cây:

2.5

/ \

1.2 3.8

* **Thứ tự duyệt LRN (Post-order)**: 1.2 3.8 2.5
* **Tập tin đầu ra**: data1.out

**Test Case 2: Cây lớn hơn với 5 nút**

Cấu trúc cây:

5.0

/ \

2.3 7.8

/ \

1.1 3.4

* **Thứ tự duyệt LRN (Post-order)**: 1.1 3.4 2.3 7.8 5.0
* **Tập tin đầu ra**: data2.out

**BÀI 11:**

Duyệt cây theo phương pháp LNR ta sẽ được các giá trị tang dần.

**BÀI 12:**

Duyệt cây theo phương pháp RNL ta sẽ được các giá trị giảm dần.

***BÀI 13:***

**Test Case 1: Cây nhỏ với 3 nút**

Cấu trúc cây:

2.5

/ \

1.2 3.8

* **Thứ tự lưu vào file (NLR - Pre-order)**: 2.5 1.2 -1.0 -1.0 3.8 -1.0 -1.0
* **Tập tin đầu ra**: tree1.bin

**Test Case 2: Cây lớn hơn với 5 nút**

Cấu trúc cây:

5.0

/ \

2.3 7.8

/ \

1.1 3.4

* **Thứ tự lưu vào file (NLR - Pre-order)**: 5.0 2.3 1.1 -1.0 -1.0 3.4 -1.0 -1.0 7.8 -1.0 -1.0
* **Tập tin đầu ra**: tree2.bin

***BÀI 14:***

**Test Case 1: Cây nhỏ với 3 nút**

Cấu trúc cây:

5.0

/ \

3.0 7.0

* **Danh sách liên kết mong đợi (giảm dần)**: 7.0 -> 5.0 -> 3.0

**Test Case 2: Cây lớn hơn với 6 nút**

Cấu trúc cây:

10.0

/ \

5.0 15.0

/ \ / \

2.0 7.0 12.0 18.0

* **Danh sách liên kết mong đợi (giảm dần)**: 18.0 -> 15.0 -> 12.0 -> 10.0 -> 7.0 -> 5.0 -> 2.0

***BÀI 15:***

• Giống nhau

• Dữ liệu được tổ chức.

• Chi phí tìm kiếm một phần tử trên cả hai ctdl là như nhau.

• Khác nhau

• Chi phí thêm và xoá phần tử vào mảng lớn hơn chi phí cây nhị

phân tìm kiếm.

***BÀI 16:***

**Test Case 1: Xóa phần tử có nhiều lần xuất hiện**

Cây ban đầu:

10(2)

/ \

5(1) 15(1)

**Thao tác thực hiện**: Xóa 10

**Kết quả mong đợi**:

10(1)

/ \

5(1) 15(1)

(Vì 10 có số lần xuất hiện là 2, sau khi xóa, nó giảm xuống còn 1)

**Test Case 2: Xóa phần tử có một lần xuất hiện**

Cây ban đầu:

20(1)

/ \

10(1) 30(1)

/ \

5(1) 15(1)

* **Thao tác thực hiện**: Xóa 10
* **Kết quả mong đợi**:

20(1)

/ \

15(1) 30(1)

/

5(1)

(`10` bị xóa, `15` thay thế vị trí của nó)

***BÀI 17:***

**Test Case 1: Xóa một nút có hai con**

Cây ban đầu:

20

/ \

10 30

/ \ \

5 15 40

* **Thao tác thực hiện**: Xóa 20
* **Kết quả mong đợi**:

30

/ \

10 40

/ \

5 15

(Nút 30 là phần tử nhỏ nhất trong cây con phải của 20 và sẽ thay thế 20)

**Test Case 2: Xóa một nút có hai con nhưng có nhiều lần xuất hiện**

Cây ban đầu:

25(2)

/ \

10(1) 30(1)

/ \

5(1) 15(1)

* **Thao tác thực hiện**: Xóa 25
* **Kết quả mong đợi**:

25(1)

/ \

10(1) 30(1)

/ \

5(1) 15(1)

(Vì 25 có số lần xuất hiện là 2, sau khi xóa, nó giảm xuống còn 1 và vẫn còn tồn tại trong cây)

***BÀI 18:***

**Test Case 1: Mảng chưa sắp xếp có số phần tử lẻ**

**Input:**

vector<int> arr = {10, 4, 20, 2, 8, 15, 25};

**Cây BST được tạo:**

10

/ \

4 20

/ \ / \

2 8 15 25

**Kết quả mong đợi sau khi sắp xếp:**

2 4 8 10 15 20 25

**Test Case 2: Mảng có số phần tử chẵn với số âm**

**Input:**

vector<int> arr = {-3, 7, -1, 5, 2, 0, -5, 10};

**Cây BST được tạo:**

-3

/ \

-5 7

\ / \

-1 5 10

/

2

**Kết quả mong đợi sau khi sắp xếp:**

-5 -3 -1 0 2 5 7 10

***BÀI 33:***

**Test Case 1: Cây có nhiều mức**

**Input mảng:** {10, 5, 15, 3, 7, 13, 18}  
**Cấu trúc cây BST sau khi xây dựng:**

10

/ \

5 15

/ \ / \

3 7 13 18

**Kết quả mong đợi:**

* **Duyệt theo NLR (Preorder, dùng stack):** 10 5 3 7 15 13 18
* **Duyệt theo mức (Level Order, dùng queue):** 10 5 15 3 7 13 18

**Test Case 2: Cây có giá trị âm và số lượng phần tử chẵn**

**Input mảng:** {0, -10, 10, -5, -15, 5, 15}  
**Cấu trúc cây BST sau khi xây dựng:**

0

/ \

-10 10

/ \ / \

-15 -5 5 15

**Kết quả mong đợi:**

* **Duyệt theo NLR (Preorder, dùng stack):** 0 -10 -15 -5 10 5 15
* **Duyệt theo mức (Level Order, dùng queue):** 0 -10 10 -15 -5 5 15

***BÀI 37:***

**Test Case 1: Cây cân bằng**

**Input mảng:** {10, 5, 15, 3, 7, 13, 18}  
**Cấu trúc cây BST sau khi xây dựng:**

10

/ \

5 15

/ \ / \

3 7 13 18

**Kết quả mong đợi:**

* **Duyệt theo NLR (Preorder, dùng stack):** 10 5 3 7 15 13 18
* **Duyệt theo mức (Level Order, dùng queue):** 10 5 15 3 7 13 18
* **Cây có cân bằng không?** Có

**Test Case 2: Cây không cân bằng (nghiêng trái)**

**Input mảng:** {30, 20, 10, 5, 3, 2, 1}  
**Cấu trúc cây BST sau khi xây dựng:**

30

/

20

/

10

/

5

/

3

/

2

/

1

**Kết quả mong đợi:**

* **Duyệt theo NLR (Preorder, dùng stack):** 30 20 10 5 3 2 1
* **Duyệt theo mức (Level Order, dùng queue):** 30 20 10 5 3 2 1
* **Cây có cân bằng không?** Không