TRƯỜNG ĐẠI HỌC SÀI GÒN

**KHOA CÔNG NGHỆ THÔNG TIN**

**Môn: Cấu trúc dữ liệu và giải thuật**

Tên: Đặng Hữu Đăng Tâm

MSSV: 3124411265

**CÂY NHỊ PHÂN TÌM KIẾM**

Câu 1&4:

**1. Ý Tưởng và Chức Năng:**

Chương trình này được thiết kế để thao tác với cây nhị phân tìm kiếm (Binary Search Tree - BST) các số nguyên. Nó cung cấp các chức năng sau:

* **Tạo Cây Nhị Phân:** Cho phép người dùng nhập các số nguyên để tạo một cây nhị phân tìm kiếm.
* **Đếm và Tính Tổng Nút:** Tính số lượng nút và tổng giá trị của tất cả các nút trong cây.
* **Tính Trung Bình Cộng:** Tính trung bình cộng của tất cả các nút trong cây.
* **Thống Kê Nút Dương/Âm:** Đếm và tính tổng các nút dương và âm riêng biệt.
* **Tính Trung Bình Cộng Nút Dương/Âm:** Tính trung bình cộng của các nút dương và âm.
* **Tính Tỷ Số Tổng Dương/Âm:** Tính tỷ số giữa tổng các nút dương và tổng các nút âm.
* **Tìm Nút Nhỏ Nhất/Lớn Nhất:** Tìm và hiển thị giá trị của nút nhỏ nhất và lớn nhất trong cây.

**Ý tưởng chính:**

* Sử dụng cấu trúc dữ liệu cây nhị phân tìm kiếm để lưu trữ dữ liệu.
* Sử dụng đệ quy để duyệt cây và thực hiện các phép tính.
* Phân loại các nút thành dương và âm để thực hiện các phép tính riêng biệt.

**2. Input và Output:**

**Input:**

* + Số lượng nút của cây.
  + Giá trị của từng nút (số nguyên).

**Output:**

* + Số lượng nút trong cây.
  + Tổng giá trị của các nút.
  + Trung bình cộng của các nút.
  + Số lượng và tổng các nút dương.
  + Trung bình cộng của các nút dương.
  + Số lượng và tổng các nút âm.
  + Trung bình cộng của các nút âm.
  + Tỷ số giữa tổng các nút dương và âm.
  + giá trị nhỏ nhất và lớn nhất trong cây.

**3. Test Case:**

Dưới đây là một số test case để kiểm tra chương trình:

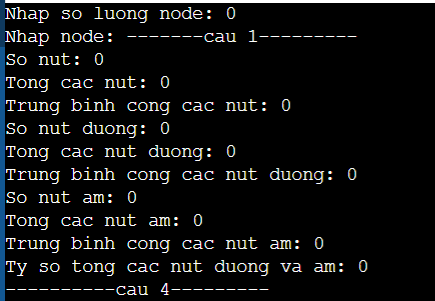
**Test Case 1: Cây Rỗng**

**Input:**

* + Số lượng nút: 0

**Output:**

* + Số nút: 0
  + Tổng các nút: 0
  + Trung bình cộng các nút: 0
  + Số nút dương: 0
  + Tổng các nút dương: 0
  + Trung bình cộng các nút dương: 0
  + Số nút âm: 0
  + Tổng các nút âm: 0
  + Trung bình cộng các nút âm: 0
  + Tỷ số tổng các nút duong va am: 0
  + phan tu nho nhat: error
  + phan tu lon nhat: error



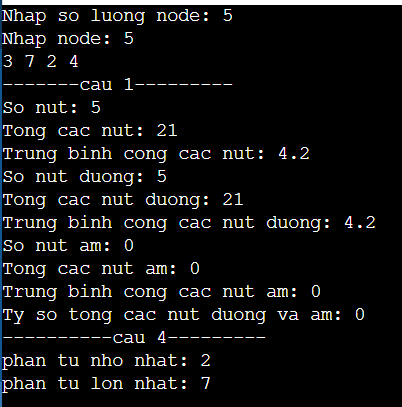
**Test Case 2: Cây với các nút dương**

**Input:**

* + Số lượng nút: 5
  + Giá trị nút: 5 3 7 2 4

**Output:**

* + Số nút: 5
  + Tổng các nút: 21
  + Trung bình cộng các nút: 4.2
  + Số nút dương: 5
  + Tổng các nút dương: 21
  + Trung bình cộng các nút dương: 4.2
  + Số nút âm: 0
  + Tổng các nút âm: 0
  + Trung bình cộng các nút âm: 0
  + Tỷ số tổng các nút duong va am: 0
  + phan tu nho nhat: 2
  + phan tu lon nhat: 7



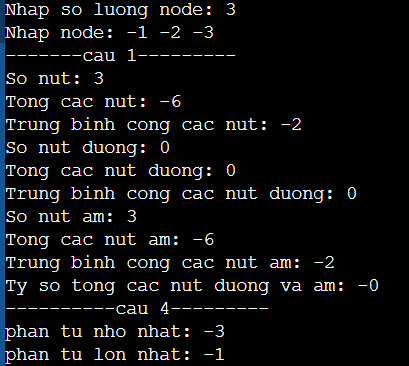
**Test Case 3: Cây với các nút âm**

**Input:**

* Số lượng nút: 3
* Giá trị nút: -1 -2 -3

**Output:**

* + Số nút: 3
  + Tổng các nút: -6
  + Trung bình cộng các nút: -2
  + Số nút dương: 0
  + Tổng các nút dương: 0
  + Trung bình cộng các nút dương: 0
  + Số nút âm: 3
  + Tổng các nút âm: -6
  + Trung bình cộng các nút âm: -2
  + Tỷ số tổng các nút duong va am: 0
  + phan tu nho nhat: -3
  + phan tu lon nhat: -1



**Test Case 4: Cây với cả nút dương và âm**

**Input:**

* + Số lượng nút: 7
  + Giá trị nút: 5 -3 7 -2 4 -1 6

**Output:**

* + Số nút: 7
  + Tổng các nút: 16
  + Trung bình cộng các nút: 2.28571
  + Số nút dương: 4
  + Tổng các nút dương: 22
  + Trung bình cộng các nút dương: 5.5
  + Số nút âm: 3
  + Tổng các nút âm: -6
  + Trung bình cộng các nút âm: -2
  + Tỷ số tổng các nút duong va am: -3.66667
  + phan tu nho nhat: -3
  + phan tu lon nhat: 7
  + phần tử lớn nhất: 5

Câu 8 & 9 & 10:

**1. Ý Tưởng và Chức Năng:**

Chương trình này được thiết kế để ghi dữ liệu từ một cây nhị phân tìm kiếm (BST) vào file nhị phân theo ba thứ tự duyệt khác nhau:

* **LNR (Left-Node-Right):** Duyệt cây theo thứ tự trái, gốc, phải.
* **NLR (Node-Left-Right):** Duyệt cây theo thứ tự gốc, trái, phải.
* **LRN (Left-Right-Node):** Duyệt cây theo thứ tự trái, phải, gốc.

**Ý tưởng chính:**

* Sử dụng cấu trúc dữ liệu cây nhị phân để lưu trữ dữ liệu số thực.
* Sử dụng đệ quy để thực hiện các phép duyệt cây theo ba thứ tự LNR, NLR, và LRN.
* Sử dụng hàm fwrite để ghi dữ liệu số thực vào file nhị phân.
* Sử dụng hàm con trỏ hàm để có thể truyền vào hàm Xuat các hàm duyệt cây khác nhau.
* Sử dụng hàm fread để đọc dữ liệu từ file nhị phân.

**2. Input và Output:**

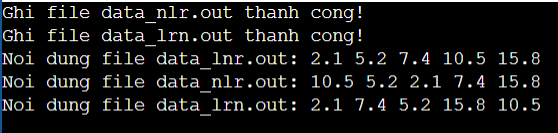
* **Input:**
  + Dữ liệu số thực được chèn vào cây nhị phân tìm kiếm.
* **Output:**
  + Ba file nhị phân: data\_lnr.out, data\_nlr.out, và data\_lrn.out, chứa dữ liệu từ cây nhị phân theo các thứ tự duyệt tương ứng.
  + In ra nội dung các file đã ghi vào màn hình.

**3. Test Case:**

Dưới đây là một số test case để kiểm tra chương trình:

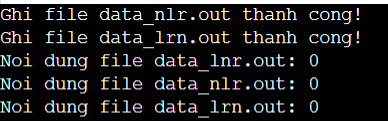
**Test Case 1: Cây Nhị Phân Đơn Giản**

* **Input:**
  + Chèn các giá trị: 10.5, 5.2, 15.8, 2.1, 7.4 vào cây nhị phân.
* **Output:**
  + data\_lnr.out: 2.1 5.2 7.4 10.5 15.8
  + data\_nlr.out: 10.5 5.2 2.1 7.4 15.8
  + data\_lrn.out: 2.1 7.4 5.2 15.8 10.5
  + In ra màn hình nội dung 3 file trên.



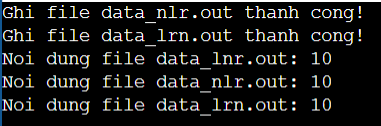
**Test Case 2: Cây Nhị Phân Rỗng**

* **Input:**
  + Không chèn giá trị nào vào cây nhị phân.
* **Output:**
  + Ba file nhị phân sẽ rỗng.
  + In ra màn hình nội dung 3 file rỗng.



**Test Case 3: Cây Nhị Phân Chỉ Có Gốc**

* **Input:**
  + Chèn giá trị 10.0 vào cây nhị phân.
* **Output:**
  + data\_lnr.out: 10.0
  + data\_nlr.out: 10.0
  + data\_lrn.out: 10.0
  + In ra màn hình nội dung 3 file.



Câu 16:

**1. Ý Tưởng và Chức Năng:**

Chương trình này được thiết kế để quản lý và thao tác với cây nhị phân tìm kiếm (BST) các số nguyên, với khả năng xử lý các phần tử trùng lặp bằng cách đếm số lần xuất hiện của chúng. Các chức năng chính bao gồm:

* **Tạo và Khởi Tạo Cây:** Tạo một cây BST rỗng.
* **Thêm Phần Tử:** Thêm một số nguyên vào cây. Nếu số đó đã tồn tại, tăng số lần xuất hiện của nó.
* **Xóa Phần Tử:** Xóa một số nguyên khỏi cây. Nếu số đó xuất hiện nhiều lần, giảm số lần xuất hiện.
* **Liệt Kê Cây:** Duyệt và in ra các phần tử trong cây theo thứ tự NLR (Node-Left-Right).

**Ý tưởng chính:**

* Sử dụng cấu trúc dữ liệu cây nhị phân tìm kiếm để lưu trữ dữ liệu.
* Mỗi nút trong cây lưu trữ một khóa (Key) và số lần xuất hiện (So\_lan).
* Thêm và xóa phần tử được thực hiện theo nguyên tắc của cây BST, với việc xử lý đặc biệt cho các phần tử trùng lặp.
* Duyệt cây NLR được sử dụng để in ra các phần tử theo thứ tự gốc, trái, phải.

**2. Input và Output:**

* **Input:**
  + Các số nguyên được thêm vào cây.
  + Số nguyên cần xóa khỏi cây.
* **Output:**
  + In ra các phần tử của cây theo thứ tự NLR, kèm theo số lần xuất hiện của mỗi phần tử.

**3. Test Case:**

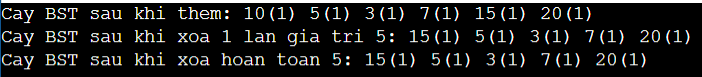
Dưới đây là một số test case để kiểm tra chương trình:

**Test Case 1: Cây Rỗng**

* **Input:**
  + Không thêm phần tử nào.
* **Output:**
  + "Cay BST sau khi them: " (không có gì được in ra).
  + "Cay BST sau khi xoa 1 lan gia tri 5: " (không có gì được in ra).
  + "Cay BST sau khi xoa hoan toan 5: " (không có gì được in ra).

**Test Case 2: Thêm và Xóa Phần Tử Đơn Giản**

* **Input:**
* Thêm: 10, 5, 15, 20, 3, 7.
* Xóa: 10.
* **Output:**
  + "Cay BST sau khi them: 10(1) 5(1) 3(1) 7(1) 15(1) 20(1) "
  + "Cay BST sau khi xoa 1 lan gia tri 5: 15(1) 5(0) 3(1) 7(1) 20(1)"
  + "Cay BST sau khi xoa hoan toan 5: 15(1) 3(1) 7(1) 20(1)"



**Test Case 3: Thêm Phần Tử Trùng Lặp**

* **Input:**
  + Thêm: 10, 5, 15, 5, 20, 3, 7.
  + Xóa 5 1 lần, sau đó xóa 5 lần 2
* **Output:**
  + "Cay BST sau khi them: 10(1) 5(2) 3(1) 7(1) 15(1) 20(1) "
  + "Cay BST sau khi xoa 1 lan gia tri 5: 10(1) 5(1) 3(1) 7(1) 15(1) 20(1) "
  + "Cay BST sau khi xoa hoan toan 5: 10(1) 3(1) 7(1) 15(1) 20(1) "

