**1. Lý thuyết:**

*a. Khái niệm:* Binary Search Tree (BST - Cây tìm kiếm nhị phân) là một cấu trúc dữ liệu dạng cây, trong đó mỗi nút có dữ liệu riêng và tối đa hai nút con: một nút con bên trái (left child) và một nút con bên phải (right child). BST được tổ chức sao cho việc tìm kiếm, chèn, hoặc xóa các giá trị trở nên hiệu quả.

*b. Thuộc tính:* Thuộc tính chính của BST là giá trị của tất cả các nút trong cây con bên trái luôn nhỏ hơn giá trị của nút gốc (root), trong khi giá trị của các nút trong cây con bên phải luôn lớn hơn giá trị của nút gốc. Quy tắc này được áp dụng đệ quy cho mọi nút trong cây, giúp duy trì trật tự.

*c. Tính chất:* BST hỗ trợ tìm kiếm, chèn, và xóa dữ liệu với hiệu suất trung bình là O(log n), nếu cây được cân bằng. Điều này khiến BST trở thành một công cụ mạnh mẽ để làm việc với các tập hợp dữ liệu lớn, nơi tốc độ truy xuất là yếu tố quan trọng.

*d. Ứng dụng:* BST được ứng dụng rộng rãi trong nhiều lĩnh vực, bao gồm tìm kiếm nhanh trong tập dữ liệu lớn, xây dựng bảng từ điển, quản lý lịch sử duyệt web, và hệ thống quản lý tệp tin. Ngoài ra, BST còn là nền tảng cho các cấu trúc cây nâng cao như AVL tree và Red-Black tree.

### 

### **2. Bài tập:**

**a. Khai báo cấu trúc dữ liệu:**

Đoạn code đầu tiên định nghĩa cấu trúc House, đại diện cho thông tin một ngôi nhà. Mỗi ngôi nhà bao gồm các thuộc tính: tên chủ sở hữu (**ownerName**), tuổi chủ sở hữu (**ownerAge**), và giá nhà (**price**). Ba toán tử so sánh (<, >, ==) được định nghĩa để so sánh hai ngôi nhà dựa trên giá của chúng. Ngoài ra, cấu trúc Node mô tả một nút trong cây tìm kiếm nhị phân (BST), chứa dữ liệu House và hai con trỏ trỏ đến các nút con trái (**left**) và phải (**right**). Cấu trúc BST lưu gốc của cây.

### **b. Khởi tạo cây BST**

### Hàm **initBST** khởi tạo một cây BST bằng cách gán con trỏ gốc (root) của cây bằng NULL, biểu thị cây rỗng.

### **c. Tạo nút mới**

Hàm createNode được sử dụng để tạo một nút mới chứa dữ liệu của một ngôi nhà (**House**). Hàm này kiểm tra xem việc cấp phát bộ nhớ có thành công hay không, sau đó khởi tạo các con trỏ trái và phải là NULL.

### **d. Thêm phần tử vào cây - Đệ quy:**

### Hàm **insertNodeRecursion** thêm một phần tử vào cây BST bằng cách sử dụng đệ quy. Nếu cây rỗng, hàm tạo nút mới và trả về nó. Nếu cây không rỗng, hàm so sánh giá trị mới với nút hiện tại: nếu nhỏ hơn thì thêm vào cây con bên trái, nếu lớn hơn thì thêm vào cây con bên phải. Hàm được gọi đệ quy cho đến khi tìm được vị trí thích hợp.

### **e. Thêm phần tử vào cây - Không dùng đệ quy**

Hàm insertNodeNonRecursion thêm phần tử mới vào cây BST mà không sử dụng đệ quy. Hàm tìm vị trí thích hợp để thêm phần tử bằng cách duyệt qua cây từ gốc đến lá, cập nhật các con trỏ cha (par) và nút hiện tại (cur). Sau đó, nó chèn phần tử mới vào vị trí thích hợp bên trái hoặc bên phải của nút cha.

### **f. Xóa phần tử khỏi cây - Đệ quy**

Hàm **deleleNodeRecursion** xóa một phần tử khỏi cây BST bằng cách sử dụng đệ quy. Nếu không tìm thấy phần tử, hàm trả về nút hiện tại. Nếu tìm thấy, ba trường hợp xảy ra:

1. Nút không có con -> xóa trực tiếp.
2. Nút có một con -> thay thế nút bằng con của nó.
3. Nút có hai con -> tìm nút nhỏ nhất trong cây con bên phải, thay thế dữ liệu của nút hiện tại, và xóa nút nhỏ nhất này.

### **g. Tìm chiều cao của cây**

Hàm **findHeight** xác định chiều cao của cây BST. Hàm tính chiều cao của cây con bên trái và bên phải bằng cách đệ quy, sau đó trả về giá trị lớn hơn cộng thêm 1 (tính cả nút gốc).

### g. Hàm **main**

Hàm main là nơi thực hiện chương trình, nhưng hiện tại chỉ có đoạn khung cơ bản mà chưa gọi các hàm hoặc thực hiện thao tác cụ thể nào trên cây BST. Đây là nơi bạn có thể thử nghiệm chèn, xóa, và truy vấn các nút trong cây BST.