BỘ GIÁO DỤC VÀ ĐÀO TẠO

**TRƯỜNG ĐẠI HỌC NHA TRANG**

KHOA CÔNG NGHỆ THÔNG TIN



**THỰC TẬP CƠ SỞ CÔNG NGHỆ THÔNG TIN**

**THAO TÁC TRÊN CÂY NHỊ PHÂN TÌM KIẾM**

**Giảng viên hướng dẫn: ThS. Nguyễn Thị Hương Lý**

**Sinh viên thực hiện: Trần Quang Vinh**

**Lớp: 64.CNTT-1**

Khánh Hòa – 2024

BỘ GIÁO DỤC VÀ ĐÀO TẠO

**TRƯỜNG ĐẠI HỌC NHA TRANG**

KHOA CÔNG NGHỆ THÔNG TIN



**THỰC TẬP CƠ SỞ CÔNG NGHỆ THÔNG TIN**

**THAO TÁC TRÊN CÂY NHỊ PHÂN TÌM KIẾM**

GVHD: ThS. Nguyễn Thị Hương Lý

Sinh viên thực hiện: Trần Quang Vinh

Lớp: 64.CNTT-1

Khánh Hòa – 2024

**MỤC LỤC**

[LỜI NÓI ĐẦU 1](#_Toc93268350)

[1. Lý do chọn đề tài 2](#_Toc93268351)

[2. Mục tiêu của đề tài 3](#_Toc93268352)

[3. Phạm vi nghiên cứu 4](#_Toc93268353)

[4. Phương pháp nghiên cứu 5](#_Toc93268354)

[CHƯƠNG 1: CƠ SỞ LÝ THUYẾT 6](#_Toc93268355)

[1.1 Lý thuyết 7](#_Toc93268356)

[1.2 Tổng quan về thuật toán 8](#_Toc93268357)

[CHƯƠNG 2: ỨNG DỤNG THUẬT TOÁN 9](#_Toc93268358)

[2.1 GIỚI THIỆT THUẬT TOÁN 10](#_Toc93268359)

[2.1.1 Mô tả thuật toán 11](#_Toc93268360)

[2.1.2 Cài đặt thuật toán 12](#_Toc93268361)

[2.1.3 Thuật toán 13](#_Toc93268362)

[2.2 Phân tích thiết kế chương trình 14](#_Toc93268363)

[2.2.1 Thành phần của bài toán 15](#_Toc93268364)

[2.2.2 Cấu trúc dữ liệu bài toán 16](#_Toc93268365)

[2.3 Cài đặt chương trình 17](#_Toc93268366)

[2.3.1 Sơ đồ khái quát 18](#_Toc93268367)

[2.3.2 Minh họa thuật toán 19](#_Toc93268368)

[2.3.3 Kết quả chương trình 20](#_Toc93268369)

[KẾT LUẬN 21](#_Toc93268370)

[1.Các kết quả đạt được. 22](#_Toc93268371)

[2.Ưu và nhược điểm của đề tài 23](#_Toc93268372)

[3.Hướng phát triển 24](#_Toc93268373)

**LỜI NÓI ĐẦU**

Công nghệ thông tin và lập trình đã trở thành nền tảng quan trọng trong thời đại hiện nay, với ứng dụng rộng rãi trong mọi lĩnh vực từ giáo dục, y tế, tài chính cho đến sản xuất và quản lý. Trong đó, các cấu trúc dữ liệu và thuật toán đóng vai trò cốt lõi, giúp tối ưu hóa quá trình xử lý thông tin và giải quyết các bài toán một cách hiệu quả.

Cây nhị phân tìm kiếm (Binary Search Tree) là một trong những cấu trúc dữ liệu cơ bản, phổ biến và hữu ích trong lập trình. Với đặc tính sắp xếp tự nhiên và khả năng truy xuất dữ liệu nhanh chóng, cây nhị phân tìm kiếm thường được sử dụng trong các hệ thống cơ sở dữ liệu, trình biên dịch, và nhiều ứng dụng khác. Tuy nhiên, việc hiểu rõ nguyên lý hoạt động, cách cài đặt và trực quan hóa cây nhị phân tìm kiếm là một thách thức đối với nhiều lập trình viên mới bắt đầu.

Dự án “Mô phỏng trực quan cây nhị phân tìm kiếm” ra đời với mục tiêu cung cấp một công cụ hữu ích để học tập và nghiên cứu. Thông qua ứng dụng, người dùng có thể tạo ra cây nhị phân tìm kiếm, thêm hoặc xóa các node, và quan sát trực quan cách thức cây thay đổi theo từng thao tác. Đây không chỉ là một bài toán lập trình mà còn là một bước tiến trong việc kết nối lý thuyết với thực hành.

Hy vọng rằng, với dự án này, người đọc sẽ nắm vững hơn về cấu trúc cây nhị phân tìm kiếm, đồng thời có thêm cảm hứng khám phá các thuật toán khác trong lĩnh vực khoa học máy tính. Dự án cũng là một bước đệm quan trọng để tiến xa hơn trong các nghiên cứu và ứng dụng thực tiễn sau này.

***Xin chân thành cảm ơn !***

**1. Lý do chọn đề tài**

Cây nhị phân tìm kiếm là một trong những cấu trúc dữ liệu quan trọng, được áp dụng trong nhiều lĩnh vực như xử lý dữ liệu, tìm kiếm và quản lý hệ thống. Tuy nhiên, việc học và hình dung cách hoạt động của BST thường gặp khó khăn do thiếu sự minh họa trực quan.

**2. Mục tiêu của đề tài**

* Xây dựng ứng dụng trực quan hóa BST với các tính năng cơ bản: thêm, xóa và hiển thị cây.
* Tích hợp giao diện người dùng thân thiện và dễ sử dụng.

**3. Phạm vi nghiên cứu**

* Ứng dụng sử dụng HTML, CSS và JavaScript.
* Hỗ trợ các chức năng cơ bản của BST, bao gồm: chèn (insert), xóa (delete), và duyệt cây (in-order, level-order).

**4. Phương pháp nghiên cứu**

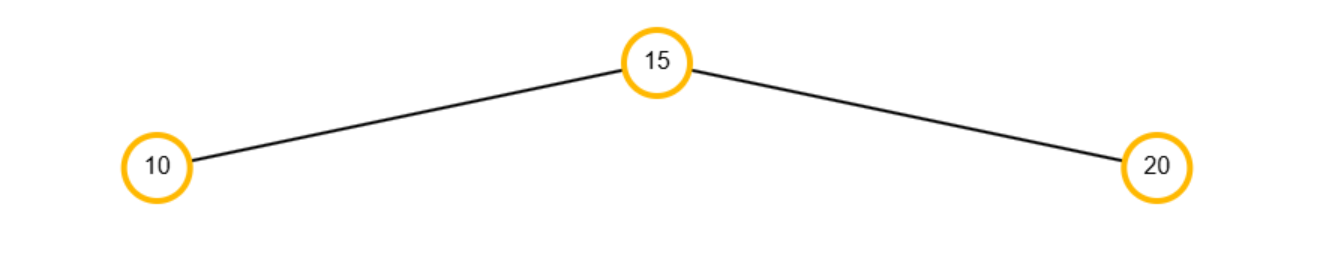
* Sử dụng phương pháp lập trình hướng đối tượng trong JavaScript để mô hình hóa cây nhị phân.
* Thiết kế giao diện web sử dụng HTML và CSS để hiển thị kết quả trực quan.

**CHƯƠNG 1: CƠ SỞ LÝ THUYẾT**

**1.1 Lý thuyết**

Cây nhị phân tìm kiếm (BST) là một cấu trúc dữ liệu cây, trong đó:

1. Mỗi nút có tối đa hai con: con trái và con phải.



1. Giá trị của các nút trong cây thỏa mãn:

* Giá trị của nút con bên trái nhỏ hơn giá trị của nút cha.
* Giá trị của nút con bên phải lớn hơn giá trị của nút cha.

**1.2 Tổng quan về thuật toán**

1. Chèn nút: Duyệt cây từ gốc để tìm vị trí phù hợp cho nút mới.
2. Xóa nút: Phụ thuộc vào số lượng con của nút cần xóa:

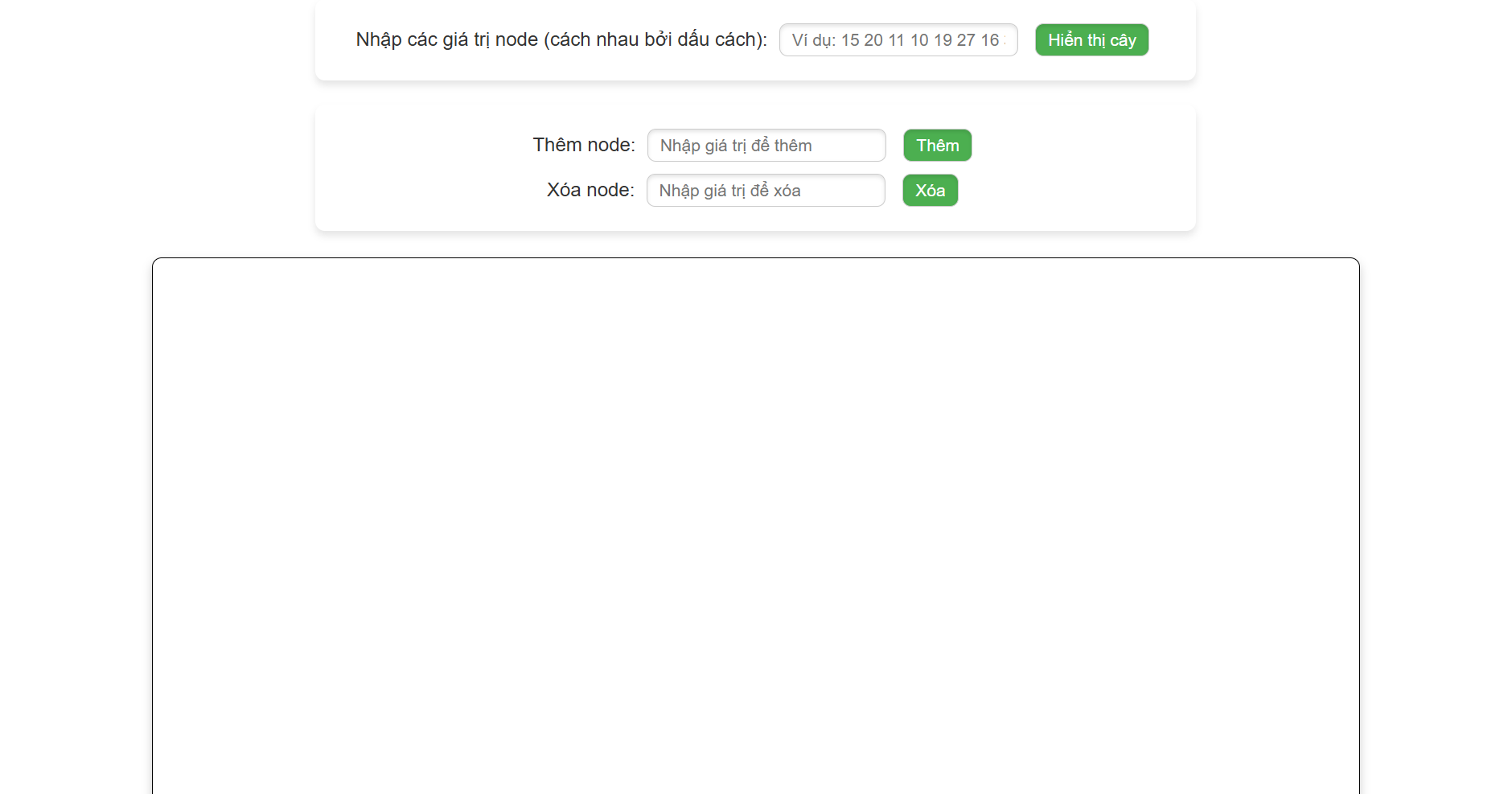
* Không có con: Xóa trực tiếp.
* Có một con: Thay thế bằng con hiện có.
* Có hai con: Tìm giá trị nhỏ nhất của cây con phải và thay thế.

**CHƯƠNG 2: ỨNG DỤNG THUẬT TOÁN**

**2.1 GIỚI THIỆU THUẬT TOÁN**

**2.1.1 Mô tả thuật toán:**

Ứng dụng sử dụng các thuật toán chèn, xóa và duyệt cây để hiển thị cây BST trên giao diện đồ họa.



**2.1.2 Cài đặt thuật toán**

Trong quá trình xây dựng chương trình mô phỏng cây nhị phân tìm kiếm (Binary Search Tree), việc cài đặt thuật toán không chỉ dừng lại ở thao tác quản lý dữ liệu mà còn cần một cách tiếp cận để trực quan hóa các node và cấu trúc cây trên giao diện.

**Cấu trúc dữ liệu:**

1. Lớp Node:

Mỗi nút trong cây được biểu diễn bởi lớp Node, với các thuộc tính cơ bản bao gồm:

* value: Giá trị của nút.
* x, y: Tọa độ để xác định vị trí của nút trên canvas (phục vụ việc vẽ).
* left, right: Tham chiếu đến các nút con bên trái và bên phải.

1. Lớp ActionBST:

Lớp này đảm nhiệm vai trò quản lý cây nhị phân tìm kiếm, bao gồm các thao tác:

* Thêm nút (insert): Thêm giá trị mới vào cây, đồng thời đảm bảo tuân thủ quy tắc của cây nhị phân tìm kiếm.
* Xóa nút (delete): Loại bỏ một giá trị khỏi cây và tái cân bằng cấu trúc cây nếu cần.
* Duyệt cây (inOrder, levelOrder): Lấy danh sách các nút theo thứ tự tăng dần hoặc theo từng cấp bậc (dùng để ánh xạ tọa độ khi vẽ).

**Vẽ cây nhị phân tìm kiếm trên canvas:**

Việc trực quan hóa cây nhị phân tìm kiếm được thực hiện trên thẻ <canvas>. Quá trình vẽ bao gồm các bước chính sau:

1. Ánh xạ vị trí ngang (x):

Sử dụng phương pháp duyệt cây theo thứ tự trung thứ (inOrder) để lấy danh sách các nút.

Tính toán khoảng cách ngang giữa các nút bằng cách chia chiều rộng của canvas cho tổng số nút.

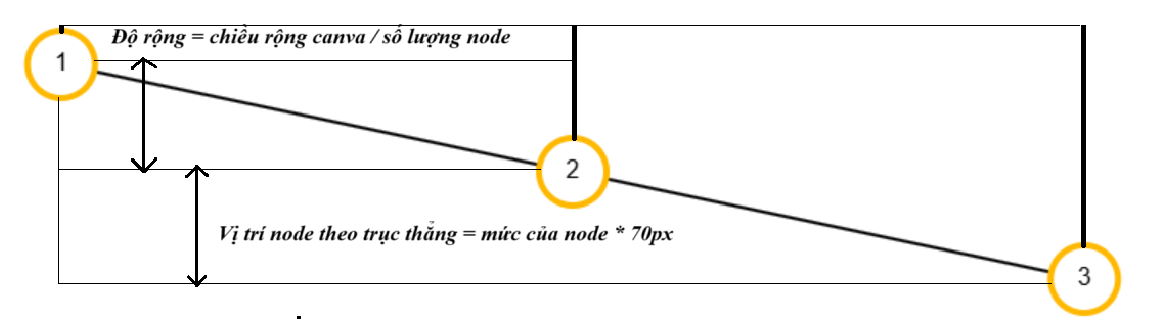
Gán giá trị x cho từng nút dựa trên thứ tự xuất hiện trong danh sách.

1. Ánh xạ vị trí dọc (y):

Dựa trên thứ bậc (level) của mỗi nút trong cây.

Sử dụng duyệt theo cấp (levelOrder) để chia các nút thành từng cấp bậc.

Gán giá trị y cho các nút dựa trên thứ bậc, với khoảng cách dọc cố định giữa các cấp bậc.



***Hình ảnh: Ánh xạ các nút dọc và ngang ( cách đều nhau )***

1. Vẽ các nút (drawNode):

Sử dụng phương thức arc của canvas để vẽ các hình tròn đại diện cho các nút.

Tô màu nền và viền, sau đó hiển thị giá trị nút ở vị trí trung tâm hình tròn.

1. Vẽ các đường nối (drawLine):

Với mỗi nút, nếu có con trái hoặc con phải, vẽ đường thẳng từ tâm nút hiện tại đến tâm nút con tương ứng.

Sử dụng phương thức lineTo để nối các điểm.

**2.2 PHÂN TÍCH THIẾT KẾ CHƯƠNG TRÌNH**

**2.2.1 Thành phần của bài toán**

1. HTML:

Cung cấp giao diện nhập liệu và các nút tương tác.

Canvas để vẽ cây.

1. CSS:

Tạo bố cục và định dạng giao diện người dùng.

1. JavaScript:

Xử lý logic cây BST và vẽ cây trên Canvas.

**2.2.2 Cấu trúc dữ liệu bài toán**

Cấu trúc cây nhị phân được mô tả bằng lớp Node, trong đó:

1. value: Giá trị của nút.
2. left, right: Trỏ đến con trái và con phải.
3. x, y: Vị trí tọa độ trên canvas.

class Node {

    constructor(value, x = 0, y = 0) {

        this.value = value;

        this.left = null;

        this.right = null;

        this.x = x;

        this.y = y;

    }

}

Các xử lý logic của cây nhị phân

1. Thêm node:

insert(value) {

        const newNode = new Node(value);

        if (this.root == null) {

            this.root = newNode;

        } else {

            this.#insertNode(this.root, newNode);

        }

    }

    #insertNode(node, newNode) {

        if (newNode.value === node.value) return;

        if (newNode.value < node.value) {

            if (node.left === null) {

                node.left = newNode;

            } else {

                this.#insertNode(node.left, newNode);

            }

        } else {

            if (node.right === null) {

                node.right = newNode;

            } else {

                this.#insertNode(node.right, newNode);

            }

        }

    }

2. Xóa node

delete(value) {

        this.root = this.#deleteNode(this.root, value);

    }

    #deleteNode(node, value) {

        if (node === null) return null;

        if (value < node.value) {

            node.left = this.#deleteNode(node.left, value);

            return node;

        } else if (value > node.value) {

            node.right = this.#deleteNode(node.right, value

            return node;

        } else {

            if (node.left === null && node.right === null) {

                return null;

            } else if (node.left === null) {

                return node.right;

            } else if (node.right === null) {

                return node.left;

            } else {

                const minRightValue

                    = this.#findMinNode(node.right).value;

                node.value = minRightValue;

                node.right = this.#deleteNode

                            (node.right, minRightValue);

                return node;

            }

        }

    }

**2.3 CÀI ĐẶT CHƯƠNG TRÌNH**

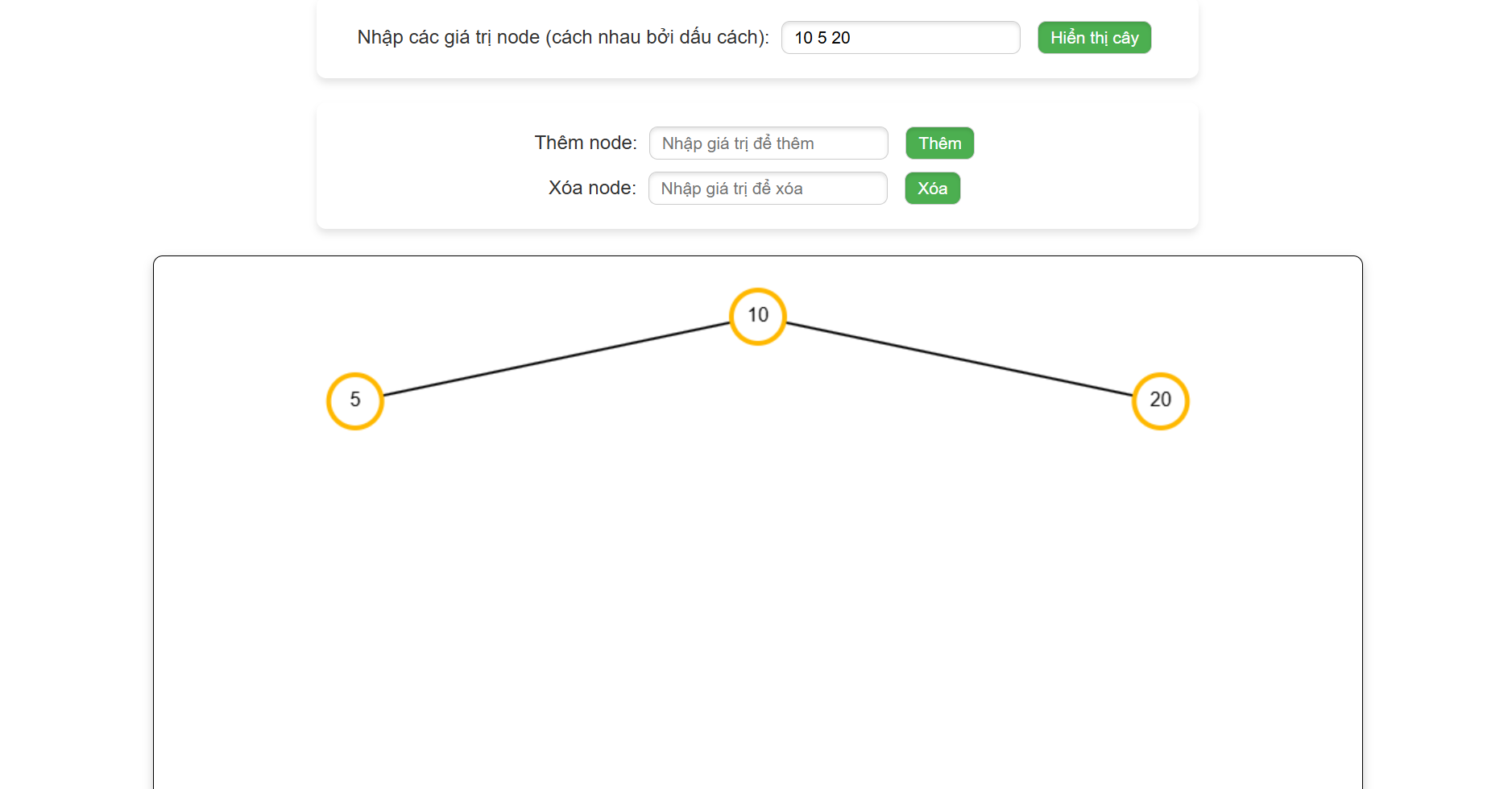
**2.3.1 Sơ đồ khái quát**

Ứng dụng bao gồm các bước chính:

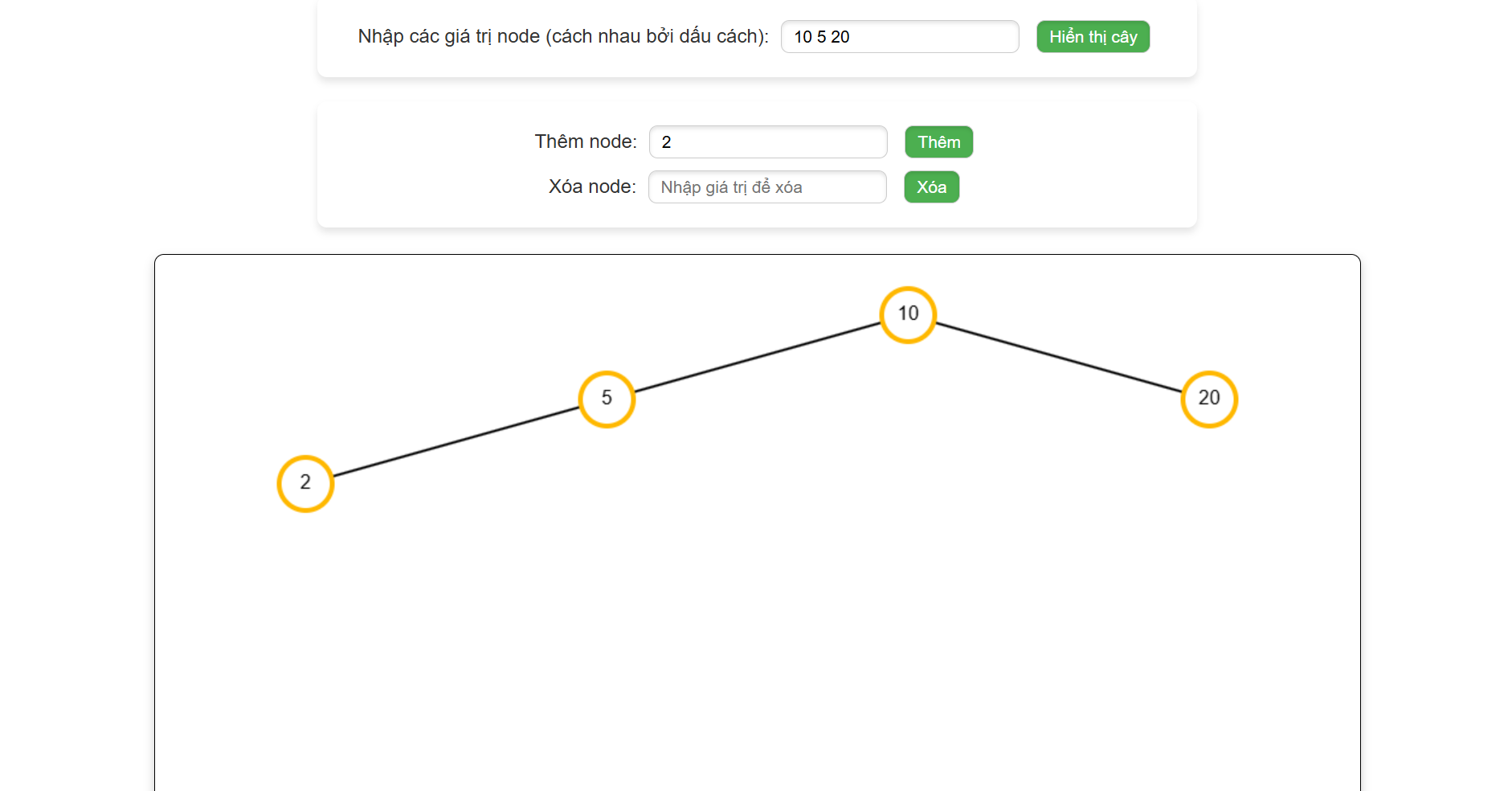
1. Nhập dữ liệu từ người dùng.
2. Cập nhật cấu trúc cây theo thao tác thêm/xóa.
3. Tính toán vị trí các nút.
4. Vẽ cây trên canvas.

**2.3.2 Minh họa thuật toán**

1. Thêm nút:  
   Khi người dùng nhập giá trị và nhấn "Thêm", thuật toán tìm vị trí thích hợp và cập nhật cây.

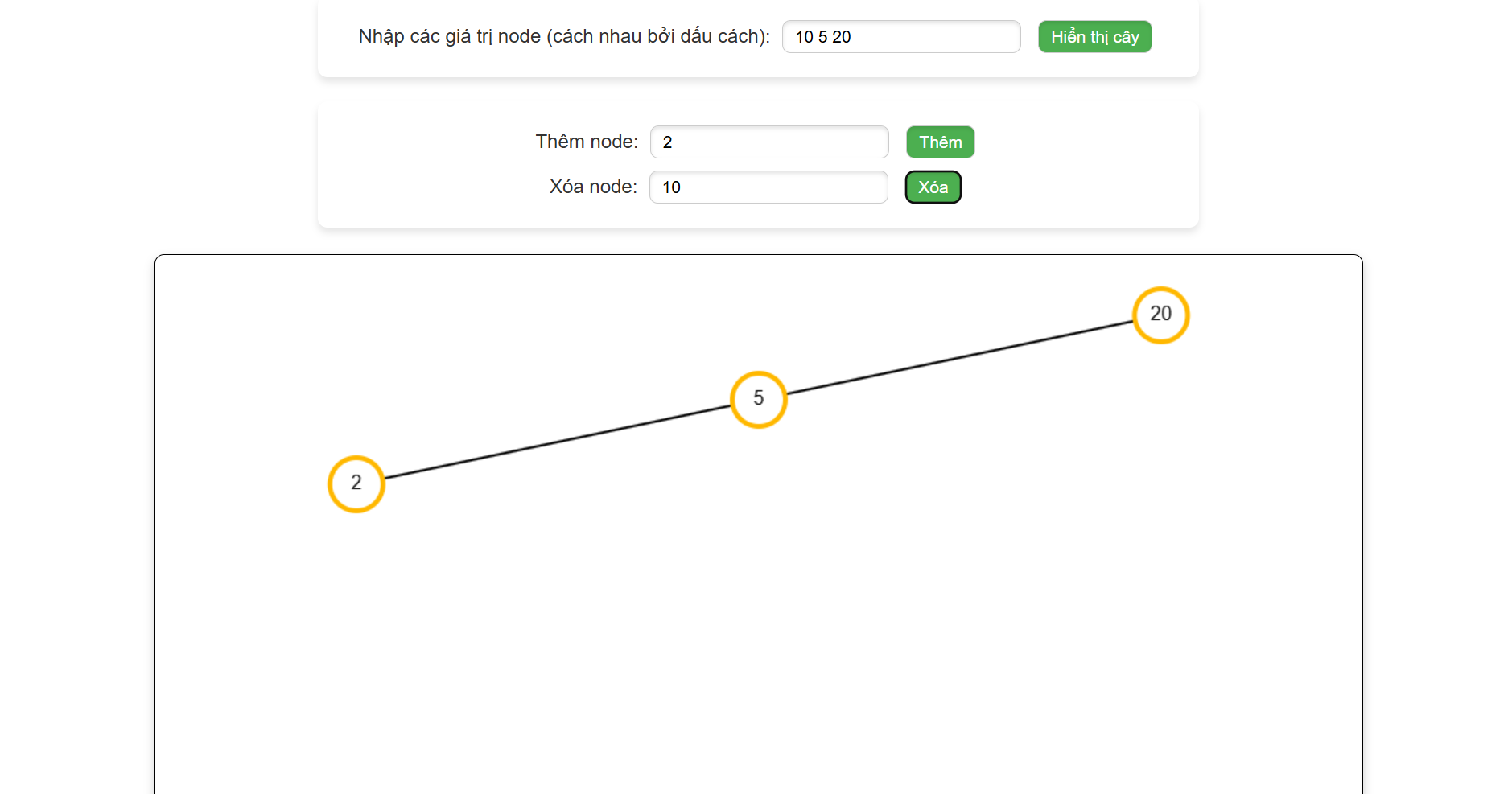


***Hình ảnh: Một cây nhị phân với giá trị ban đầu là: 10, 5, 20***



***Hình ảnh: Sau khi thêm một node có giá trị là 2***

1. Xóa nút:  
   Thuật toán tìm và xóa nút theo quy tắc đã đề cập ở phần lý thuyết.



***Hình ảnh: Sau khi đã xóa node có giá trị là 10***

**2.3.3 Kết quả chương trình**

Ứng dụng hiển thị trực quan cây BST với các nút được vẽ tại đúng vị trí, kèm theo các đường nối thể hiện mối quan hệ cha-con.

**KẾT LUẬN**

**1. Các kết quả đạt được**

* Xây dựng thành công ứng dụng trực quan hóa cây BST với các chức năng cơ bản.
* Tích hợp giao diện thân thiện, dễ sử dụng.

**2. Ưu và nhược điểm của đề tài**

Ưu điểm:

* Trực quan, dễ tiếp cận.
* Hiệu năng ổn định với dữ liệu nhỏ và trung bình.

Nhược điểm:

* Chưa hỗ trợ các chức năng phức tạp như cân bằng cây (AVL, Red-Black Tree).
* Hiệu năng giảm khi dữ liệu lớn.

**3. Hướng phát triển**

* Tối ưu hóa hiệu năng cho dữ liệu lớn.
* Thêm các thuật toán khác như cân bằng cây, xoay cây.
* Hỗ trợ xuất/nhập dữ liệu cây dưới dạng file JSON/XML.