

**TRƯỜNG ĐẠI HỌC BẠC LIÊU  
KHOA CÔNG NGHỆ THÔNG TIN**



**NIÊN LUẬN 1**

**TÊN ĐỀ TÀI**

**CÂY TÌM KIẾM NHỊ PHÂN TỰ CÂN BẰNG**

*Sinh viên thực hiện*  
**Trương Trung Hiếu**

*Mã sinh viên*  
**17D480201013**

*Giảng viên hướng dẫn*

**Ths. Triệu Vĩnh Viêm**

**HỌC KỲ 1, 2019 – 2020**

**TRƯỜNG ĐẠI HỌC BẠC LIÊU  
KHOA CÔNG NGHỆ THÔNG TIN**



**NIÊN LUẬN 1**

**TÊN ĐỀ TÀI**

**CÂY TÌM KIẾM NHỊ PHÂN TỰ CÂN BẰNG**

*Sinh viên thực hiện*  
**Trương Trung Hiếu**

*Mã sinh viên*  
**17D480201013**

*Giảng viên hướng dẫn*

**Ths. Triệu Vĩnh Viêm**

**HỌC KỲ 1, 2019 – 2020**

## NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN

[illegible]

**Bạc Liêu, ngày    tháng    năm 201**

**Giáo viên hướng dẫn**

**Ký tên**

# ĐÁNH GIÁ KẾT QUẢ THỰC HIỆN NIÊN LUẬN 1

(Học kỳ 1, Niên khóa 2019 – 2020)

**TÊN ĐỀ TÀI:** Cây tìm kiếm nhị phân tự cân bằng

**GIÁO VIÊN HƯỚNG DẪN:**

STT	HỌ VÀ TÊN	MSCB
1	Triệu Vĩnh Viêm	

**CÁC SINH VIÊN THỰC HIỆN:**

STT	HỌ VÀ TÊN	MSSV	THƯỜNG (Tối đa 1,0 điểm)	ĐIỂM
1	Trương Trung Hiếu	17D480201013		

<b>I. HÌNH THỨC</b> (Tối đa 0,5 điểm)	
<b>Bìa</b> (tối đa 0,25 điểm)	
<ul style="list-style-type: none"><li>Các tiêu đề: Trường ĐHBL, Khoa CNTT</li><li>Loại niên luận: 1</li><li>Tên đề tài</li><li>Tên nhóm</li><li>Thông tin về các sinh viên thực hiện: họ tên, mã số sinh viên</li><li>Giáo viên hướng dẫn: chức danh, họ tên</li><li>Học kỳ, niên khóa thực hiện</li></ul>	
<b>Bố cục</b> (tối đa 0,25 điểm)	
<ul style="list-style-type: none"><li>Nhận xét, đánh giá của giảng viên chấm</li><li>Mục lục: cấu trúc chương, mục và tiểu mục</li><li>Phụ lục (nếu có)</li><li>Tài liệu tham khảo</li></ul>	
<b>II. NỘI DUNG</b> (Tối đa 3,5 điểm)	
<b>Tổng quan</b> (tối đa 0,5 điểm)	
<ul style="list-style-type: none"><li>Mô tả bài toán, mục tiêu cần đạt được (0,25 điểm)</li><li>Hướng giải quyết và kế hoạch thực hiện (0,25 điểm)</li></ul>	
<b>Lý thuyết</b> (tối đa 0,5 điểm)	
<ul style="list-style-type: none"><li>Các khái niệm sử dụng trong đề tài (0,25 điểm)</li><li>Kết quả vận dụng lý thuyết vào đề tài (0,25 điểm)</li></ul>	
<b>Ứng dụng</b> (tối đa 2,0 điểm)	
<ul style="list-style-type: none"><li>Phân tích yêu cầu bài toán, xây dựng các cấu trúc dữ liệu cần thiết (tối đa 0,5 điểm)</li><li>Giải thuật (Lưu đồ-Ngôn ngữ giả) (1,0 điểm)</li><li>Giới thiệu chương trình (0,5 điểm)</li></ul>	
<b>Kết luận</b> (tối đa 0,5 điểm)	
<ul style="list-style-type: none"><li>Nhận xét kết quả đạt được</li></ul>	

<ul style="list-style-type: none"> <li>▪ Hạn chế</li> <li>▪ Hướng phát triển</li> </ul>	
<b>III. CHƯƠNG TRÌNH DEMO</b> ( <i>Tối đa 5,0 điểm</i> )	
<b>Giao diện thân thiện với người dùng</b> ( <i>1,0 điểm</i> )	
<b>Hướng dẫn sử dụng</b> ( <i>0.5 điểm</i> )	
<b>Kết quả thực hiện đúng với kết quả của phần ứng dụng</b> ( <i>3,5 điểm</i> )	

**Ghi chú:**

1. Điểm trong khung “sinh viên thực hiện” là điểm kết quả cuối cùng của từng sinh viên trong quá trình thực hiện niên luận 1.
2. Nếu sinh viên demo chương trình và trả lời vấn đáp không đạt yêu cầu của giáo viên hướng dẫn thì sinh viên sẽ nhận điểm F cho học phần này.

Bạc Liêu, ngày ... tháng ... năm 2019

**GIÁO VIÊN CHẤM**

# LỜI MỞ ĐẦU



Xã hội ngày càng phát triển cuộc sống con người mỗi lúc được cải thiện hơn do nhu cầu sống, làm việc và giải trí của con người ngày càng tăng cao. Để đáp ứng nhu cầu đó của xã hội, nhiều công nghệ tiên tiến mới ra đời, và con người đã áp dụng nó trong cuộc sống một cách hiệu quả nhất. Nói đến công nghệ thông tin là nói đến sự tiện lợi và nhanh chóng của những lợi ích mà nó đem lại cho con người. Nó trở thành một phần không thể thiếu trong cuộc sống. Công nghệ có thể thay thế ta làm những việc mà con người chưa làm được và giúp ta giải quyết những vấn đề đầu đầu nhất như tính toán những con số lớn và lưu trữ dữ liệu khổng lồ một cách dễ dàng.

Hiện nay, công nghệ thông tin được xem là một ngành mũi nhọn của quốc gia, đặc biệt là các nước đang phát triển, tiến hành công nghiệp hóa hiện đại hóa như nước ta. Sự bùng nổ thông tin và sự phát triển mạnh mẽ của công nghệ kỹ thuật số, muốn phát triển thì phải áp dụng tin học hóa vào tất cả các ngành các lĩnh vực. Cùng với sự phát triển nhanh chóng về phần cứng máy tính, các phần mềm càng trở nên đa dạng, phong phú, hoàn thiện hơn và hỗ trợ hiệu quả cho con người. Các phần mềm hiện nay ngày càng hỗ trợ cho người dùng thuận tiện sử dụng, thời gian xử lý nhanh chóng, và một số nghiệp vụ được tự động hóa cao.

## *Đôi nét về đề tài nghiên cứu*

Đề tài này được thực hiện nhằm đạt được mục tiêu là hiểu rõ, sâu sắc hơn về phép quay trên cây tìm kiếm nhị phân, ứng dụng của thuật toán vào bài toán tự cân bằng của cây. Tìm hiểu cách để tối ưu thời gian chèn và xóa của một cây tìm kiếm nhị phân.

## *Về cơ sở lý thuyết*

- Toán học
- Cây nhị phân

## *Về đối tượng và phạm vi nghiên cứu*

- Ngôn ngữ lập trình C#.
- Phép quay trên cây nhị phân.

## *Về phương pháp nghiên cứu*

- Tìm hiểu thông tin trên mạng internet, sách, báo, tạp chí...
- Thông qua sự hướng dẫn của thầy cô giáo và nghiên cứu những tài liệu tham khảo liên quan.

# MỤC LỤC

Chương 1: TỔNG QUAN .....	1
1.1. Mô tả bài toán .....	1
1.2. Mục tiêu cần đạt được .....	1
1.3. Hướng giải quyết .....	1
1.4. Kế hoạch thực hiện .....	1
Chương 2: LÝ THUYẾT .....	2
2.1 Cây nhị phân (Binary tree) .....	2
2.1.1 Khái niệm.....	2
2.2 Cây tìm kiếm nhị phân (Binary search tree) .....	3
2.2.1 Khái niệm.....	3
2.2.2 Ưu và nhược điểm của quá trình xử lý trên cây BST .....	4
2.3 Cây nhị phân tự cân bằng (AVL TREE).....	4
2.3.1 Khái niệm.....	4
2.3.2 Khái niệm về các phép quay trên cây AVL.....	7
2.3.3 So sánh về độ tối ưu của cây AVL so với cây BST.....	10
2.4 Cơ sở lý thuyết về ngôn ngữ lập trình C# .....	11
Chương 3: ỨNG DỤNG TỰ CÂN BẰNG TRÊN CÂY NHỊ PHÂN.....	12
3.1. Phân tích yêu cầu bài toán, xây dựng các cấu trúc dữ liệu .....	12
3.1.1 Cấu trúc dữ liệu lưu trữ .....	12
3.1.2 Đặt vấn đề .....	12
3.2 THIẾT KẾ GIẢI THUẬT .....	15
3.2.1 Các ký hiệu dùng trong lưu đồ thuật toán .....	15
3.3 GIỚI THIỆU CHƯƠNG TRÌNH TỰ CÂN BẰNG CÂY TÌM KIẾM NHỊ PHÂN .....	17
3.3.1 Dùng ngôn ngữ C# trên công cụ lập trình Visual Studio .....	17
Chương 4: KẾT LUẬN.....	21
4.1 Kết quả đạt được .....	21

4.2 Hạn chế.....	21
4.3 Hướng phát triển.....	21



# CHƯƠNG 1: TỔNG QUAN

## 1.1. MÔ TẢ BÀI TOÁN

Cây tìm kiếm nhị tự cân bằng, hay còn gọi là cây AVL là cây mà tại mỗi nút phải đảm bảo tính chất chiều cao của hai cây con sai khác nhau không quá một. Hiệu quả là các phép chèn (insertion), và xóa (deletion) luôn chỉ tốn thời gian  $O(\log n)$  trong cả trường hợp trung bình và trường hợp xấu nhất. Phép bổ sung và loại bỏ có thể cần đến việc tái cân bằng bằng một hoặc nhiều phép quay. Cây AVL được gọi theo tên của 2 người đề xuất là G.M. Adelson-Velsky và E.M. Landis. Cây AVL thường được so sánh với cây đỏ đen vì chúng hỗ trợ các phép toán như nhau và cùng tốn thời gian  $O(\log n)$  cho các phép toán cơ sở. Cây AVL thường thi hành tốt hơn cây đỏ đen đối với các ứng dụng sâu sắc.

## 1.2. MỤC TIÊU CẦN ĐẠT ĐƯỢC

- Cài đặt cây tìm kiếm nhị phân tự cân bằng.
- Thêm/xóa nút cho cây, nếu mất cân bằng thì tái cân bằng với một hoặc nhiều phép quay.
- Hiểu và cài đặt thuật toán đã được yêu cầu bằng ngôn ngữ C# và sử dụng thao tác và hàm trong C#.
- Phát triển tính năng đồ họa cho bài toán.

## 1.3. HƯỚNG GIẢI QUYẾT

### 1.3.1 Về lý thuyết

- Dùng phép quay trên cây để giải quyết bài toán.

### 1.3.2 Về chương trình

- Dùng ngôn ngữ lập trình C# trên công cụ Visual Studio.

## 1.4. KẾ HOẠCH THỰC HIỆN

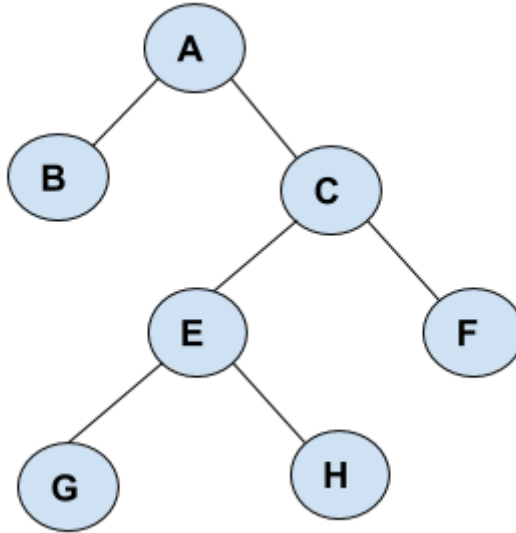
- |   |        |
|---|--------|
| ○ Tìm hiểu lý thuyết                      | 2 tuần |
| ○ Xây dựng giải thuật                     | 2 tuần |
| ○ Thiết kế giao diện                      | 1 tuần |
| ○ Viết chương trình                       | 2 tuần |
| ○ Viết báo cáo và hoàn chỉnh chương trình | 1 tuần |

## CHƯƠNG 2: LÝ THUYẾT

### 2.1 CÂY NHỊ PHÂN (BINARY TREE)

#### 2.1.1 Khái niệm

Trong khoa học máy tính, cây nhị phân (tiếng Anh: binary tree) là một cấu trúc dữ liệu cây mà mỗi nút có nhiều nhất hai nút con, được gọi là con trái (left child) và con phải (right child).



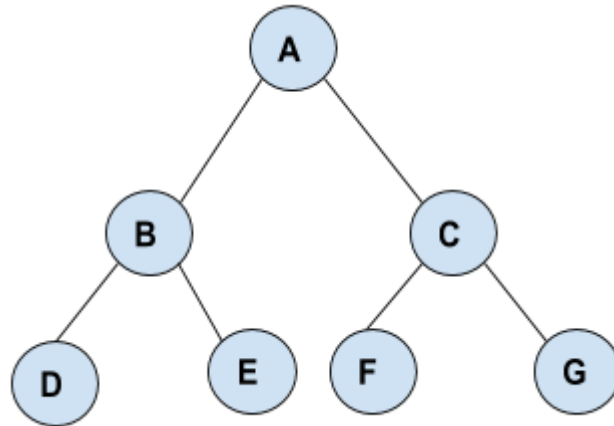
Gọi  $N$  là số node,  $H$  là chiều cao, và  $E$  là hệ số cân bằng của một cây nhị phân.

Với chú ý rằng:

- $H = -1$  trong trường hợp cây rỗng.
- $H = 0$  trong trường hợp cây có duy nhất 1 node.

Quan hệ giữa $N$ và $H$	Mô tả
$H_{max} = N - 1$	Trường hợp này xảy ra khi cây nhị phân suy biến thành cây dạng tuần tự - giống với danh sách liên kết.
$H_{min} = \log_2(N + 1) - 1$	Trong trường hợp cây nhị phân hoàn thiện, ở mỗi mức đều có số lượng tối đa các node.
$N_{min} = H + 1$	Trường hợp này xảy ra khi cây nhị phân suy biến thành cây dạng tuần tự - giống với danh sách liên kết.
$N_{max} = 2^{H+1} - 1$	Trong trường hợp cây nhị phân hoàn thiện, ở mỗi mức đều có số lượng tối đa các node.
$E = H_L - H_R$	Là sự chênh lệch chiều cao của cây con trái và cây con phải. Một cây được coi là cân bằng khi $E = 0$ hoặc -1 hoặc 1.

Cây nhị phân hoàn thiện là cây mà mỗi node của nó có đúng 2 con (trừ node lá), và các node lá có cùng mức, Ta cũng sẽ thấy cây này có hệ số  $E = 0$ , tức là độ chênh lệch ở mọi node trên cây đều bằng 0.



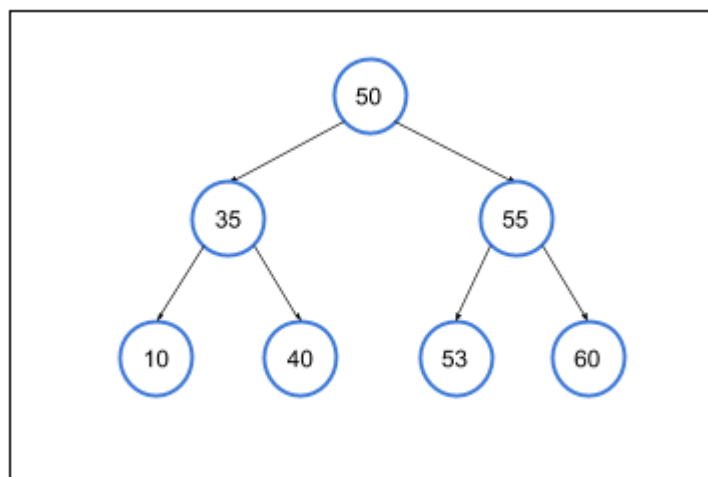
## 2.2 CÂY TÌM KIẾM NHỊ PHÂN (BINARY SEARCH TREE)

### 2.2.1 Khái niệm

Cây tìm kiếm nhị phân là cây nhị phân có các thuộc tính sau:

- Giá trị phần dữ liệu của mỗi node thuộc cây con bên trái của một node nhỏ hơn giá trị phần dữ liệu của chính node đó.
- Giá trị phần dữ liệu của mỗi node thuộc cây con bên phải của một node lớn hơn giá trị phần dữ liệu của chính node đó.

Xét ví dụ cụ thể của cây tìm kiếm nhị phân như sau, với phần dữ liệu của các node là số nguyên.



Như vậy cây tìm kiếm nhị phân là một trường hợp đặc biệt của cây nhị phân, và nó có cấu trúc tối ưu cho việc tìm kiếm cũng như cập nhật dữ liệu một cách nhanh chóng.

## 2.2.2 Ưu và nhược điểm của quá trình xử lý trên cây BST

### 2.2.2.1 Ưu điểm

Có được ưu điểm của danh sách liên kết như là : Thao tác hủy hoặc chèn rất nhanh, có thể cấp phát thêm nút hoặc thu hồi bộ nhớ đã cấp phát cho nút và kích thước cây không bị giới hạn bởi 64 KB.

Có thêm ưu điểm khác là tìm kiếm nhanh (trong khi danh sách liên kết tìm kiếm chậm).

### 2.2.2.2 Nhược điểm

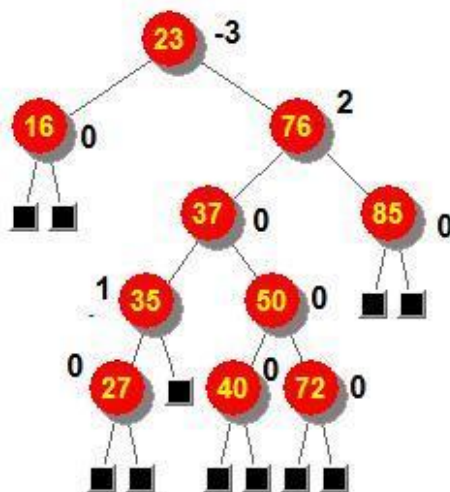
Tốn thêm bộ nhớ để lưu trữ địa chỉ nút con trái, con phải.

## 2.3 CÂY NHỊ PHÂN TỰ CÂN BẰNG (AVL TREE)

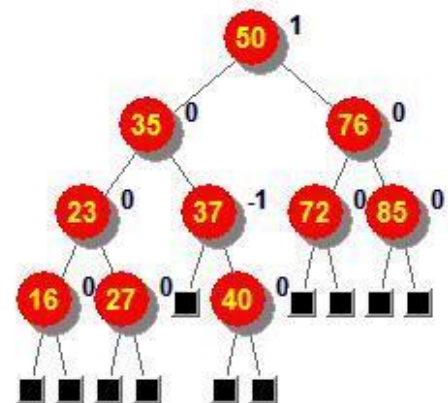
### 2.3.1 Khái niệm

#### 2.3.1.1 Khái niệm về cây AVL

Trong khoa học máy tính, một cây AVL là một cây tìm kiếm nhị phân tự cân bằng, và là cấu trúc dữ liệu đầu tiên có khả năng này. Trong một cây AVL, tại mỗi nút chiều cao của hai cây con sai khác nhau không quá một. Hiệu quả là các phép chèn (insertion), và xóa (deletion) luôn chỉ tốn thời gian  $O(\log n)$  trong cả trường hợp trung bình và trường hợp xấu nhất. Phép bổ sung và loại bỏ có thể cần đến việc tái cân bằng bằng một hoặc nhiều phép quay. Ta có thể theo dõi ví dụ về cây AVL bên dưới.<sup>1</sup>



1. Cây tìm kiếm nhị phân không là cây AVL



2. Cây AVL

Hai cây được tạo từ cùng dãy khóa

23;76;37;85;50;40;72;35;16;27;

<sup>1</sup> Bách khoa toàn thư mở Wikipedia, Cây AVL, Wikipedia, [https://vi.wikipedia.org/wiki/C%C3%A2y\\_AVL](https://vi.wikipedia.org/wiki/C%C3%A2y_AVL), tham khảo ngày 2/11/2019

Một cây áp dụng các thuật toán giúp định dạng của cây được cân bằng. Được gọi là cây cân bằng. Ta sẽ đi vào tìm hiểu kỹ hơn về cây cân bằng và các thuật toán giúp cây cân bằng ngay dưới đây.

Có vài thuật toán giúp cho cây cân bằng, trong đó thuật toán cơ bản và nổi tiếng nhất là AVL. AVL là viết tắt của hai tác giả đề xuất ra thuật toán này G.M. Adelson-Velsky và E.M. Landis.

Cây AVL là cây nhị phân tìm kiếm mà sự khác biệt về chiều cao giữa cây con trái và cây con bên phải không vượt quá 1. Các thao tác cơ bản với cây AVL là thêm node hoặc xóa node khỏi cây cũng giống với cây nhị phân tìm kiếm, chỉ có sự khác biệt là ta cần duy trì sự cân bằng của cây (sự khác biệt về chiều cao giữa cây con bên trái và cây con bên phải của các node không được vượt quá 1).

**Ưu điểm:** Hiệu năng tìm kiếm luôn được đảm bảo kể cả trong trường hợp xấu nhất (bằng  $O(\log N)$ ).

**Nhược điểm:** Khi chèn phần tử hoặc xóa phần tử khỏi cây thì cần thêm một thao tác đó là giữ cho cây cân bằng.

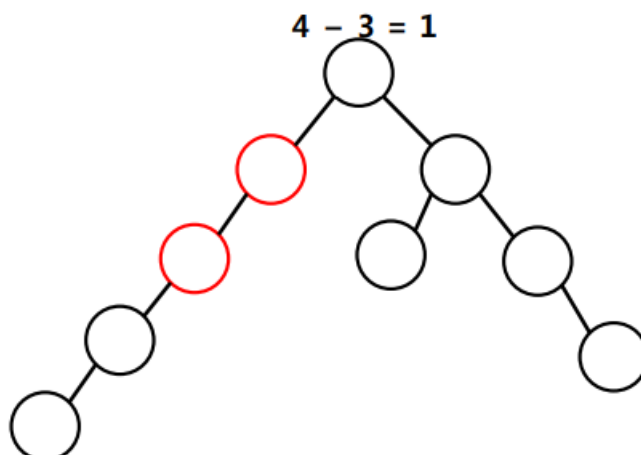
### 2.3.1.2 Khái niệm về hệ số cân bằng (Balance Factor)

Hệ số cân bằng của cây T là hiệu số giữa các chiều cao của cây con trái và cây con phải của nó. Ký hiệu hệ số cân bằng của cây con gốc u là  $balance(u)$ . Hệ số cân bằng của cây T là  $balance(T)$ .

$$balance(u) = height(u.left) - height(u.right)$$

Nếu với mọi đỉnh u của T ta có  $balance(u) = 0$  thì T được gọi là cây cân bằng hoàn toàn; Nếu  $balance(T) > 0$ , nghĩa là cây con trái cao hơn cây con phải T được gọi là cây lệch trái; Nếu  $balance(T) < 0$ , nghĩa là cây con phải cao hơn cây con trái T được gọi là cây lệch phải.

Hệ số cân bằng của các node ở mức cuối cùng (node lá) luôn luôn bằng không.



Ta thấy trên cây lúc này, chiều cao của cây con trái là 4 và chiều cao của cây con phải là 3. Vậy hệ số cân bằng lúc này của cây sẽ là  $4 - 3 = 1$ .  $\Rightarrow$  Cây đang bị lệch trái với hệ số là 1.

Sau đây là đoạn mã giả mà tôi đã tìm hiểu và nghiên cứu trên internet về cách tính hệ số cân bằng và tái cân bằng trên cây AVL.

### Tính hệ số cân bằng trên cây AVL

```
CALCULATE-BALANCE(u)2
// Tính hệ số cân bằng và chiều cao cây con tại đỉnh u.
if u.right = Null and u.left=Null then
    begin
        height(u)=1;
        balance(u)=0
    end;
else
    if u.right = Null and u.left<>Null then
        begin
            height(u)=height(u.left)+1;
            balance(u)=height(u)
        end
    else
        if u.left = Null and u.right<>Null then
            begin
                height(u)=height(u.right)+1;
                balance(u)=-height(u)
            end;
        else
            begin
                height(u)=max(height(u.left),height(u.right))+1;
                balance(u)=height(u.left)-
                height(u.right);
            end;
```

### Thủ tục tái cân bằng trên cây AVL

```
REBALANCE(v)3
// Thủ tục này tái cân bằng AVL (nếu bị mất cân bằng)
các một đỉnh tiền bối của v bị mất cân bằng AVL.
// Sau đó các bậc tiền
bối trên nữa sẽ không thay đổi hệ số cân bằng so với
trước khi đỉnh v được chèn vào.
// v là lá
```

<sup>2</sup> Đỗ Xuân Lôi. *Cấu trúc dữ liệu và giải thuật*. NXB Đại học Quốc gia Hà Nội, 2006

<sup>3</sup> Đỗ Xuân Lôi. *Cấu trúc dữ liệu và giải thuật*. NXB Đại học Quốc gia Hà Nội, 2006

```

//Tìm đỉnh tiền bối v sao cho v có tri số tuyệt đối của
v
u= v
while u<> T.root and abs(balance(u)) ≤1 do
    begin
        CALCULATE-BALACE(u)
        u=u.parent;
    end;
if balance(u)>2 then
    if balance(u.left)>0 then
        RIGHT-ROTATE(u)
    else
        begin
            LEFT-ROTATE(u.left);
            RIGHT-ROTATE(u);
        end;
else
    if balance(u)<-2 then
        if balance(u.right)< 0 then
            LEFT-ROTATE(u)
        else
            begin
                RIGHT-ROTATE(u.right);
                LEFT-ROTATE(u);
            end;
end;

```

### 2.3.2 Khái niệm về các phép quay trên cây AVL

Khi có một yếu tố gây ra sự mất cân bằng của cây, AVL sẽ thực hiện quá trình tự điều chỉnh để lấy lại sự cân bằng bởi các phép quay, bao gồm:

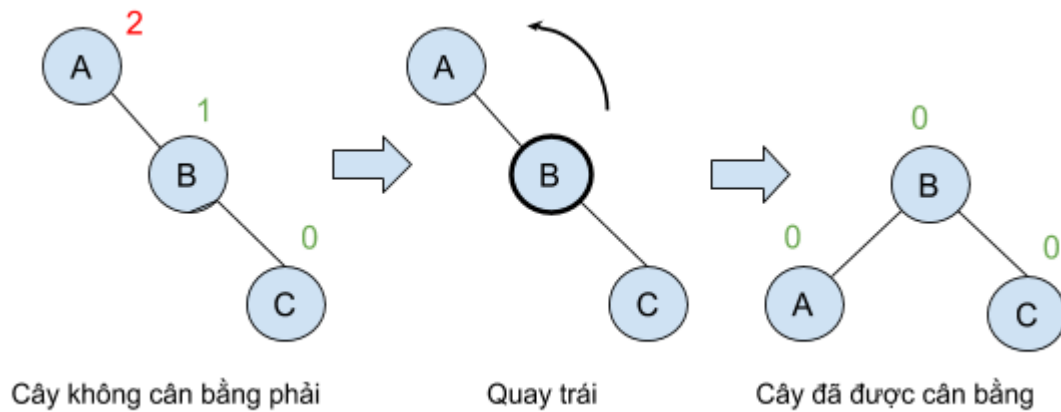
- Quay trái.
- Quay phải.
- Quay trái - phải.
- Quay phải – trái.

Ta sẽ đi vào tìm hiểu chi tiết cách hoạt động của từng phép quay. Để đơn giản hóa vấn đề ta sẽ xét những cây có hai node.

#### 2.3.2.1 Phép quay trái

Trong trường hợp cây ở trạng thái không cân bằng khi một node được chèn vào làm con phải của một cây có một node gốc và một node con phải. Lúc này ta phải thực hiện phép quay trái để đưa cây trở lại trạng thái cân bằng.

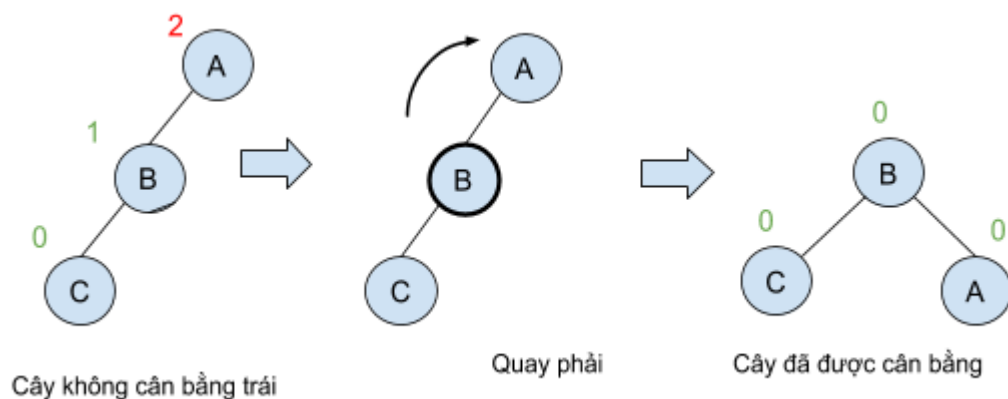
Trong trường hợp cây ở trạng thái không cân bằng khi một node được chèn vào làm con phải của một cây có một node gốc và một node con phải. Lúc này ta phải thực hiện phép quay trái để đưa cây trở lại trạng thái cân bằng.



Ở ví dụ trên, node A trở thành node không cân bằng khi có một node chèn vào cây con bên phải của nó. Tôi thực hiện phép quay trái bởi đưa node A trở thành con trái của node B, để đưa cây về trạng thái cân bằng.

### 2.3.2.2 Phép quay phải

Trong trường hợp cây ở trạng thái không cân bằng khi một node được chèn vào làm con trái của một cây có một node gốc và một node con trái. Lúc này ta phải thực hiện phép quay phải để đưa cây trở lại trạng thái cân bằng.



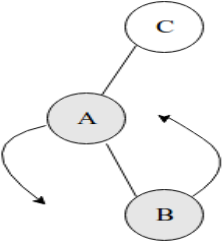
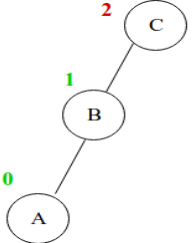
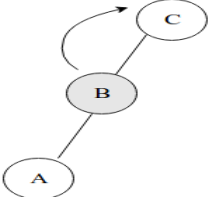
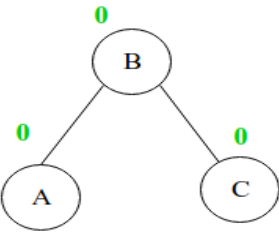
Ở ví dụ trên, khi thực hiện phép quay phải ta đưa node A (node mất cân bằng) trở thành con phải của node B.

### 2.3.2.3 Phép quay trái - phải (Trường hợp cây con trái lệch phải)

Phép quay kết hợp có đôi chút phức tạp hơn so với các phép quay trái và quay phải ở trên. Với phép quay trái-phải, ta sẽ thực hiện việc quay trái trước và quay phải sau đó.

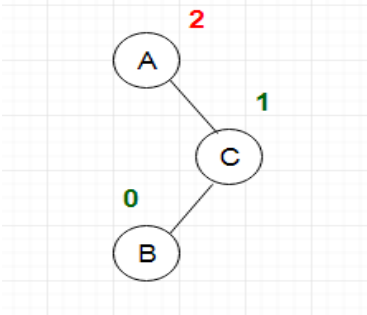
Trạng thái	Thao tác
	<p>Khi một node được chèn vào làm con phải của cây con trái. Điều này làm cho node C bị mất cân bằng. Trong tình huống như vậy ta phải thực hiện phép quay kết hợp trái - phải, để đưa cây trở về trạng thái cân bằng.</p>

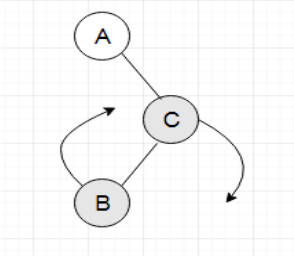
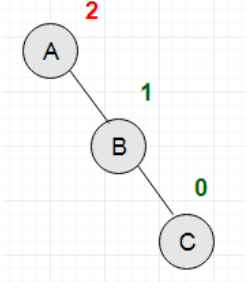
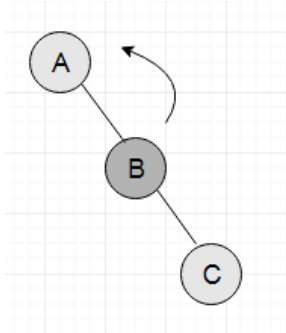
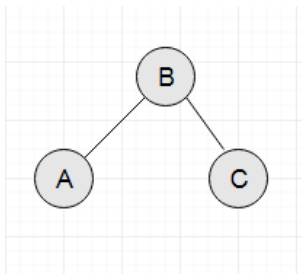


	<p>Đầu tiên, trên cây con trái của node C, ta sẽ thực hiện phép quay trái để khiến A trở thành con trái của B.</p>
	<p>Sau khi thực hiện xong một phép quay trái, cây vẫn chưa ở trạng thái cân bằng. Để đưa cây trở về trạng thái cân bằng, ta cần tiếp tục thực hiện một phép quay phải.</p>
	<p>Phép quay phải sẽ đưa B trở thành node gốc mới. Node C trở thành con phải của B.</p>
	<p>Sau phép quay kết hợp, cây đã có trạng thái cân bằng.</p>

#### 2.3.2.4 Phép quay phải – trái (Trường hợp cây con phải lệch trái)

Phép quay này là kết hợp của phép quay phải và theo sau bởi một phép quay trái.

Trạng thái	Thao tác
	<p>Khi một node được chèn vào làm con trái của cây con phải. Điều này làm cho node A bị mất cân bằng. Trong tình huống như vậy ta phải thực hiện phép quay kết hợp phải - trái, để đưa cây trở về trạng thái cân bằng.</p>

	<p>Đầu tiên, trên cây con phải của node A, ta sẽ thực hiện phép quay phải để khiến C trở thành con phải của B.</p>
	<p>Sau khi thực hiện xong một phép quay phải, cây vẫn chưa ở trạng thái cân bằng. Để đưa cây trở về trạng thái cân bằng, ta cần tiếp tục thực hiện một phép quay trái.</p>
	<p>Phép quay trái tiếp theo sẽ đưa B trở thành node gốc mới. Node A trở thành con trái của B.</p>
	<p>Sau phép quay kết hợp, cây đã có trạng thái cân bằng.</p>

### 2.3.3 So sánh về độ tối ưu của cây AVL so với cây BST

	Cây tìm kiếm nhị phân (BST)	Cây tìm kiếm nhị phân cân bằng (AVL)
Search()	$O(\log 2n) \sim O(n)$	$O(\log n)$
Insert()	$O(n)$	$O(\log n)$

Remove()	$O(n)$	$O(\log n)$
----------	--------	-------------

Ta có thể thấy, thời gian tìm kiếm trung bình của cây BST sẽ là  $O(\log 2n)$  và sẽ là  $O(n)$  khi cây ở trường hợp tệ nhất.

Còn trong khi đó thì các thao tác trên cây tìm kiếm nhị phân cân bằng sẽ là  $O(\log n)$  cho cả trường hợp trung bình và trường hợp tệ nhất.

## 2.4 CƠ SỞ LÝ THUYẾT VỀ NGÔN NGỮ LẬP TRÌNH C#

Trong dự án niên luận 1 này, tôi quyết định sử dụng ngôn ngữ C# bởi các lý do sau:

Ngôn ngữ C# khá đơn giản, chỉ khoảng 80 từ khóa và hơn mười kiểu dữ liệu được xây dựng sẵn. Tuy nhiên, ngôn ngữ C# có ý nghĩa cao khi nó thực thi những khái niệm lập trình hiện đại. C# bao gồm tất cả những hỗ trợ cho cấu trúc, thành phần component, lập trình hướng đối tượng. Những tính chất đó hiện diện trong một ngôn ngữ lập trình hiện đại và được phát triển bởi Microsoft, là phần khởi đầu cho kế hoạch .NET của họ. Tên của ngôn ngữ bao gồm ký tự thẳng theo Microsoft nhưng theo ECMA là C#, chỉ bao gồm dấu số thường. Microsoft phát triển C# dựa trên C++ và Java. C# được miêu tả là ngôn ngữ có được sự cân bằng giữa C++, Visual Basic, Delphi và Java.

C#, theo một hướng nào đó, là ngôn ngữ lập trình phản ánh trực tiếp nhất đến .NET Framework mà tất cả các chương trình .NET chạy, và nó phụ thuộc mạnh mẽ vào Framework này. Các loại dữ liệu cơ sở là những đối tượng, hay được gọi là garbage-collected, và nhiều kiểu trừu tượng khác chẳng hạn như **class**, delegate, interface, exception, v.v, phản ánh rõ ràng những đặc trưng của .NET runtime.

So sánh với C và C++, ngôn ngữ này bị giới hạn và được nâng cao ở một vài đặc điểm nào đó, nhưng không bao gồm các giới hạn sau đây:

Các con trỏ chỉ có thể được sử dụng trong chế độ không an toàn. Hầu hết các đối tượng được tham chiếu an toàn, và các phép tính đều được kiểm tra tràn bộ đệm. Các con trỏ chỉ được sử dụng để gọi các loại kiểu giá trị; còn những đối tượng thuộc bộ thu rác (*garbage-collector*) thì chỉ được gọi bằng cách tham chiếu.

- + Các đối tượng không thể được giải phóng tường minh.
- + Chỉ có đơn kế thừa, nhưng có thể cài đặt nhiều interface trừu tượng (abstract interfaces). Chức năng này làm đơn giản hóa sự thực thi của thời gian thực thi.
- + C# thì an-toàn-kiểu (*typesafe*) hơn C++.
- + Cú pháp khai báo mảng khác nhau ("int[] a = new int[5]" thay vì "int a[5]").
- + Kiểu thứ tự được thay thế bằng tên miền không gian (*namespace*).
- + C# không có tiêu bản.
- + Có thêm Properties, các phương pháp có thể gọi các Properties để truy cập dữ liệu.
- + Có reflection.

## CHƯƠNG 3: ỨNG DỤNG TỰ CÂN BẰNG TRÊN CÂY NHỊ PHÂN

### 3.1. PHÂN TÍCH YÊU CẦU BÀI TOÁN, XÂY DỰNG CÁC CẤU TRÚC DỮ LIỆU

#### 3.1.1 Cấu trúc dữ liệu lưu trữ

Mỗi nút của cây là một đối tượng có tên là Node gồm các thuộc tính sau:

Thuộc tính/Hàm	Ý nghĩa	Thuộc tính/Hàm	Ý nghĩa
x.Left	Nút con trái	x.Right	Nút con phải
Hight()	Trả về chiều cao của cây	getMin()	Trả về giá trị nút nhỏ nhất
Balance()	Kiểm tra cân bằng cây	getMax()	Trả về giá trị nút lớn nhất
Remove()	Xoá 1 nút vào cây	Insert()	Chèn 1 nút vào cây
RotateLeft()	Xoay trái	RotateRight()	Xoay phải
root	Nút gốc		

#### 3.1.2 Đặt vấn đề

**Đặt vấn đề:** Trường hợp hệ số cân bằng của nút bằng -2 và 2 là các trường hợp cây bị mất cân đối. Cần phải tái cân đối lại cây bằng các phép quay.

**Giải quyết vấn đề:** Ta xây dựng phương thức **RotateLeft(x)** và **RotateRight(x)**.

#### Phép quay cây trên cây AVL

1	<code>private Node&lt;T&gt; RotateLeft(Node&lt;T&gt; x)</code>
2	<code>{</code>
3	<code>Node&lt;T&gt; y = x.Right; //Gán 1 biến tạm y có giá trị x.Right</code>
4	<code>x.Right = y.Left; //Đổi vị trí</code>
5	<code>y.Left = x;</code>
6	<code>x.HeightNode = 1 + Math.Max(Height(x.Left), Height(x.Right));</code>
7	<code>y.HeightNode = 1 + Math.Max(Height(y.Left), Height(y.Right));</code>
8	<code>return y;</code>
9	
10	<code>}</code>
11	
12	
13	<code>private Node&lt;T&gt; RotateRight(Node&lt;T&gt; x)</code>
14	<code>{</code>
15	<code>Node&lt;T&gt; y = x.Left;</code>
16	<code>x.Left = y.Right;</code>
17	<code>y.Right = x;</code>
18	<code>x.HeightNode = 1 + Math.Max(Height(x.Left), Height(x.Right));</code>
19	<code>y.HeightNode = 1 + Math.Max(Height(y.Left), Height(y.Right));</code>
20	<code>return y;</code>
21	<code>}</code>

### **Đặt vấn đề:**

Cách để cây tự cân bằng sau mỗi lần thêm/xoá nút

### **Giải quyết vấn đề:**

#### **Thêm, Xoá nút trên cây**

##### **Thêm nút:**

1	public void Insert(Node<T> node)
2	{
3	if (node == null)
4	{
5	return;
6	}
7	root = Insert(root, node);
8	}
9	
10	private Node<T> Insert(Node<T> x, Node<T> key)
11	{
12	if (x == null)
13	{
14	return key;
15	}
16	
17	int cmp = key.CompareTo(x);
18	if (cmp < 0)
19	x.Left = Insert(x.Left, key);
20	else if (cmp > 0)
21	x.Right = Insert(x.Right, key);
22	else
23	x.Data = key.Data;
24	x = Balance(x);
25	
26	x.HeightNode = Height(x);
27	return x;
	}

##### **Chú thích:**

Dòng 1: Hàm xây dựng cho sự kiện Insert

Dòng 17-23: Khai báo biến cmp lưu trữ giá trị so sánh. Nếu  $cmp < 0$  (Nút x nhỏ hơn nút cha của x) thì chèn vào cây con bên trái,  $cmp > 0$  (Nút x lớn hơn nút cha của x) thì chèn vào cây con bên phải.

Dòng 24: Gán x = hệ số cân bằng của nó (Sử dụng hàm Balance() )

Dòng 26: Lấy chiều cao của x

## Xoá nút

1	<code>public bool Remove(Node&lt;T&gt; node)</code>
2	<code>{</code>
3	<code>    if (node == null)</code>
4	<code>    {</code>
5	<code>        return false;</code>
6	<code>    }</code>
7	<code>    return Remove(node.Data);</code>
8	<code>}</code>
9	
10	<code>private Node&lt;T&gt; Remove(Node&lt;T&gt; x, T key)</code>
11	<code>{</code>
12	<code>    if (x == null) return null;</code>
13	<code>    int cmp = key.CompareTo(x.Data);</code>
14	<code>    if (cmp &lt; 0)</code>
15	<code>        x.Left = Remove(x.Left, key);</code>
16	<code>    else if (cmp &gt; 0)</code>
17	<code>        x.Right = Remove(x.Right, key);</code>
18	<code>    else</code>
19	<code>    {</code>
20	<code>        if (x.Right == null)</code>
21	<code>            return x.Left;</code>
22	<code>        if (x.Left == null)</code>
23	<code>            return x.Right;</code>
24	<code>        Node&lt;T&gt; t = x;</code>
25	<code>        x.Data = GetMin(t.Right).Data;</code>
26	<code>        x.Right = RemoveMin(t.Right);</code>
27	<code>        x.Left = t.Left;</code>
28	<code>    }</code>
29	<code>    x = Balance(x);</code>
30	<code>    return x;</code>
31	<code>}</code>

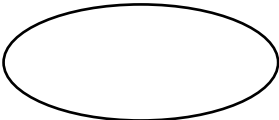

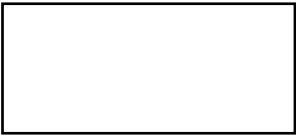
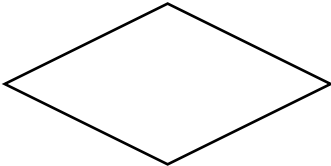

## Hàm tự cân bằng cho cây AVL bằng phép quay

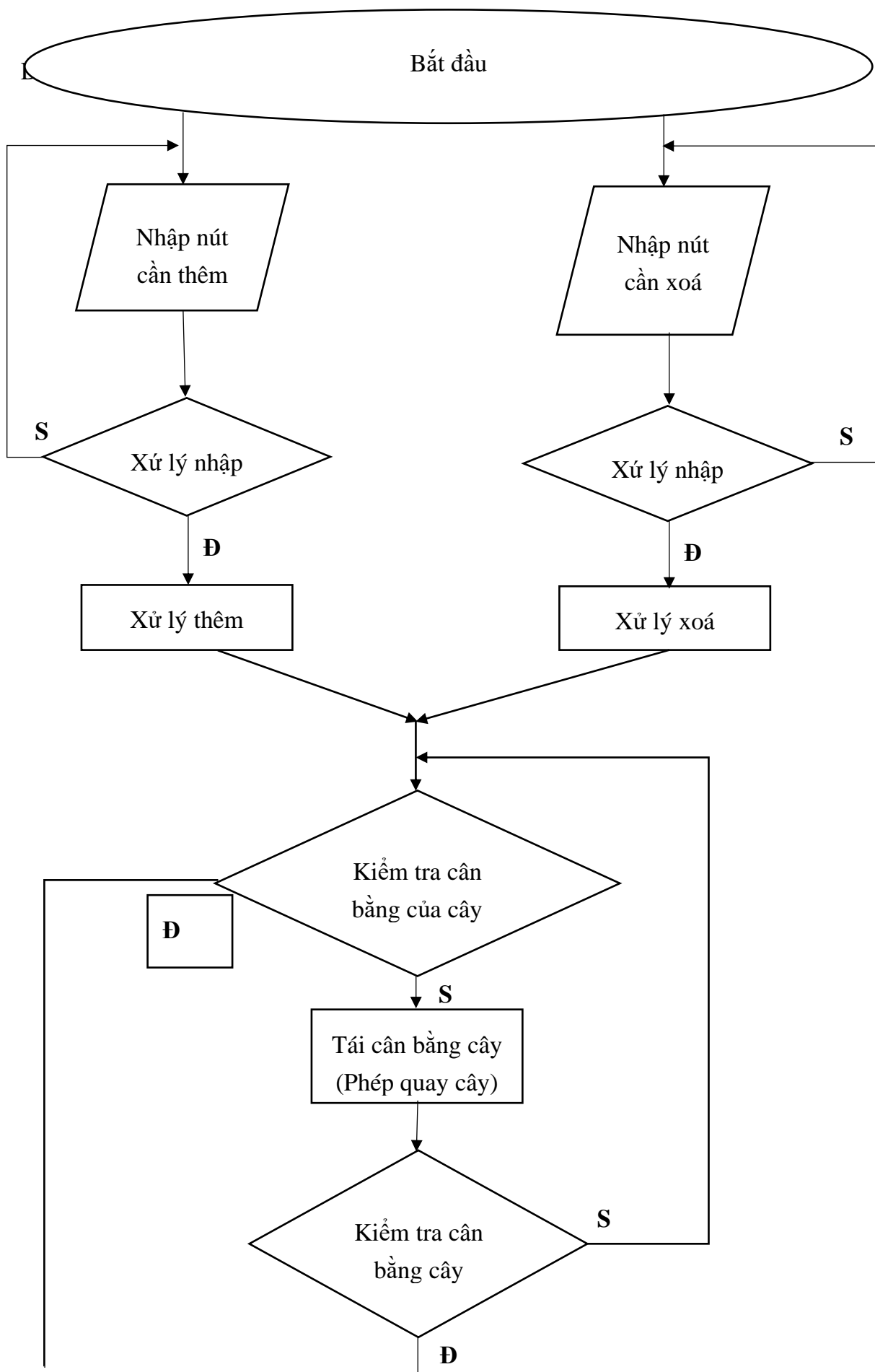
1	<code>private Node&lt;T&gt; Balance(Node&lt;T&gt; x)</code>
2	<code>{</code>
3	<code>    if (CheckBalance(x) &lt; -1)</code>
4	<code>    {</code>
5	<code>        if (CheckBalance(x.Right) &gt; 0)</code>
6	<code>        {</code>
7	<code>            x.Right = RotateRight(x.Right);</code>
8	<code>        }</code>
9	<code>        x = RotateLeft(x);</code>
10	<code>    }</code>

11	<code>else if (CheckBalance(x) &gt; 1)</code>
12	<code>{</code>
13	<code>if (CheckBalance(x.Left) &lt; 0)</code>
14	<code>{</code>
15	<code>    x.Left = RotateLeft(x.Left);</code>
16	<code>}</code>
17	<code>    x = RotateRight(x);</code>
18	
19	<code>}</code>
20	<code>return x;</code>
21	<code>}</code>
22	
23	<code>private int CheckBalance(Node&lt;T&gt; x)</code>
24	<code>{</code>
25	<code>    return Height(x.Left) - Height(x.Right);</code>
26	<code>}</code>

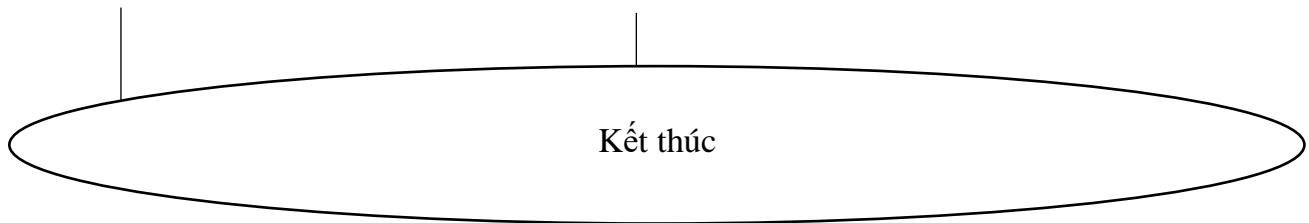
## 3.2 THIẾT KẾ GIẢI THUẬT

### 3.2.1 Các ký hiệu dùng trong lưu đồ thuật toán

STT	KÝ HIỆU	MÔ TẢ
1		Điểm bắt đầu hay chấm dứt thuật toán
2		Thao tác nhập hay xuất dữ liệu
3		Khối xử lý công việc
4		Khối quyết định chọn lựa
6		Dòng tính toán, thao tác của chương trình



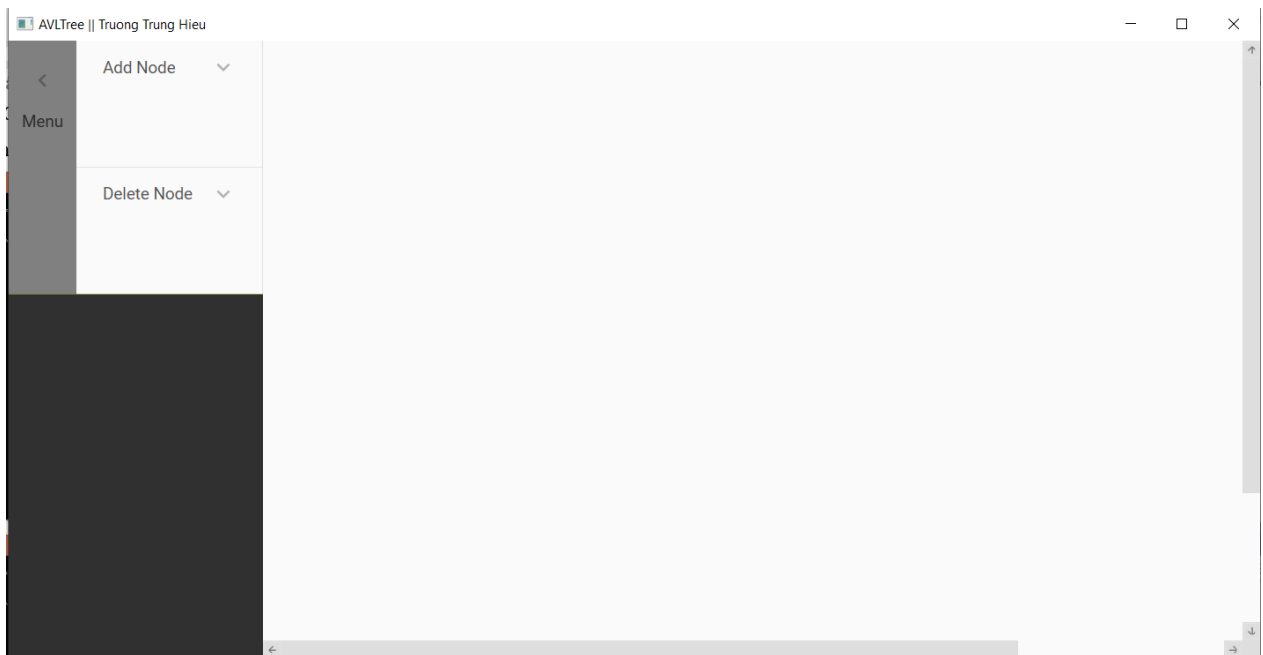




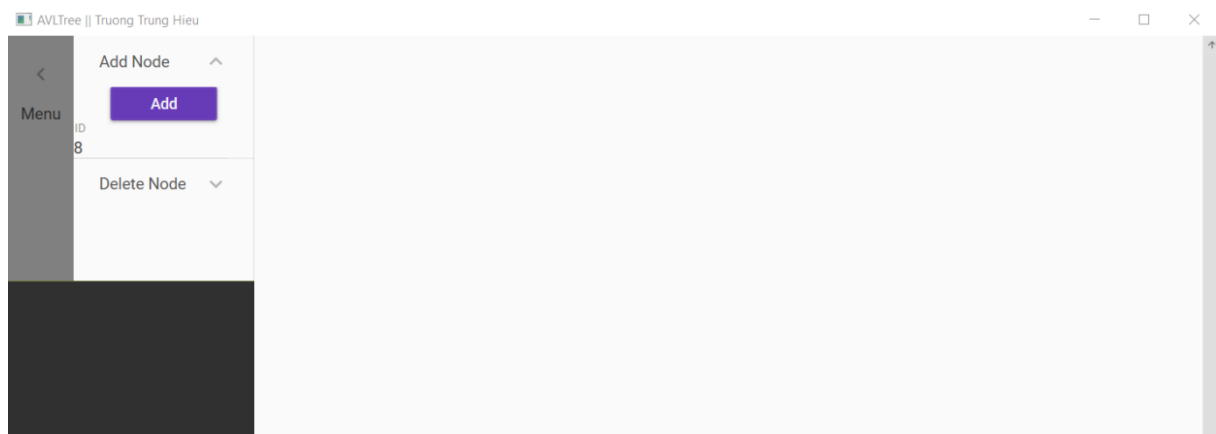
### 3.3 GIỚI THIỆU CHƯƠNG TRÌNH TỰ CÂN BẰNG CÂY TÌM KIẾM NHỊ PHÂN

#### 3.3.1 Dùng ngôn ngữ C# trên công cụ lập trình Visual Studio

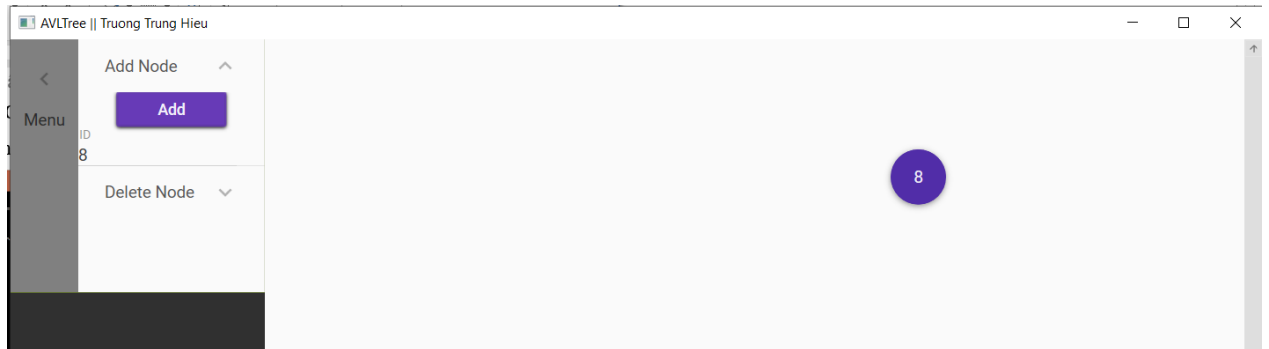
*Chọn Menu và chọn chức năng muốn sử dụng:*



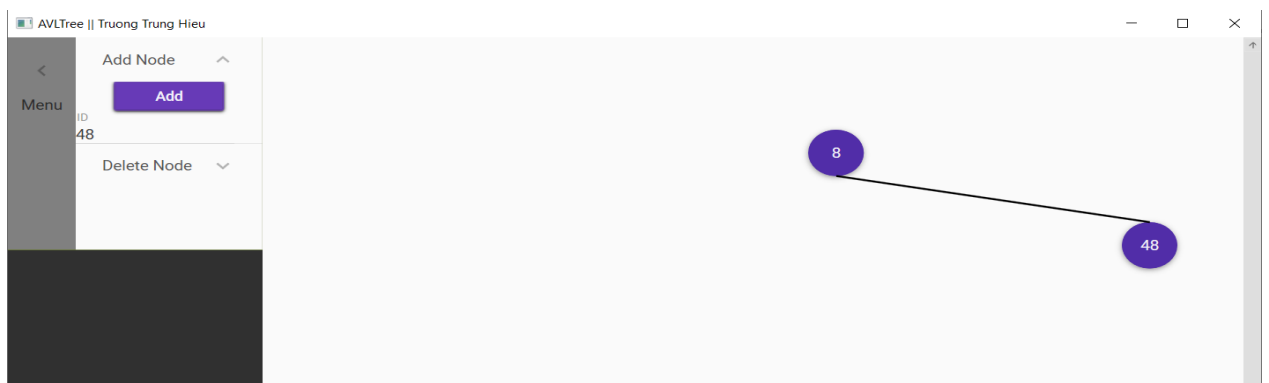
*Bấm vào Thanh menu để hiện cửa sổ chức năng*



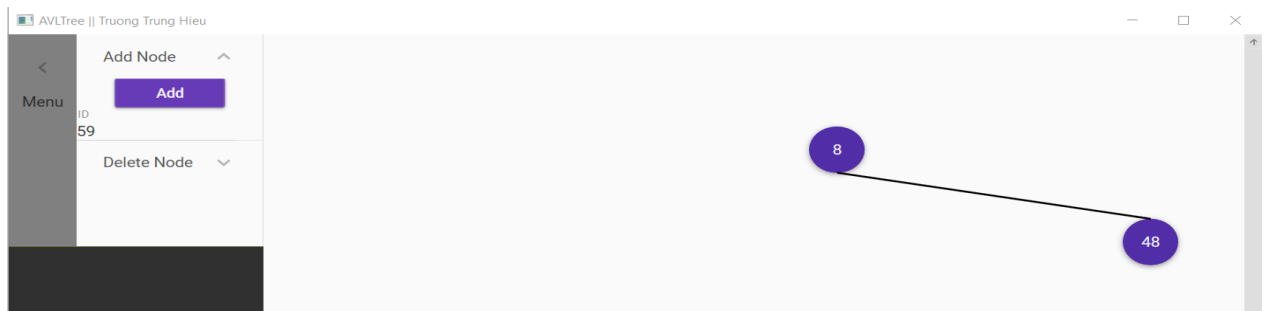
*Ấn phím Add để chèn nút vào cây*



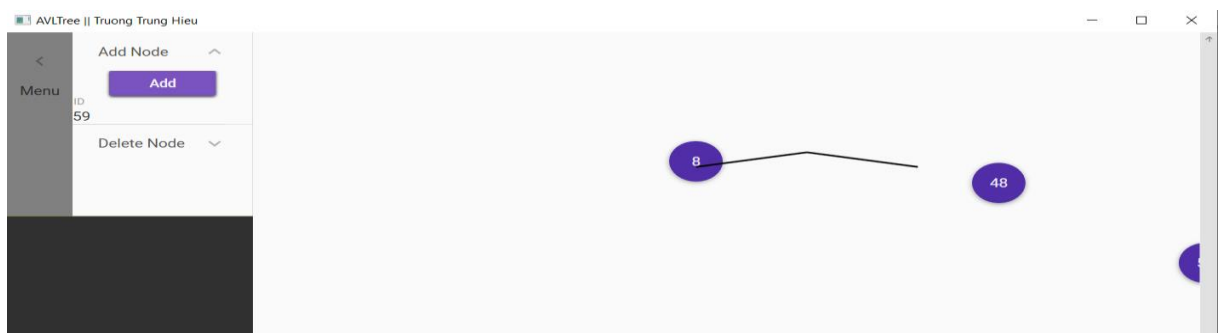
*Sau khi chèn thêm nút 48*



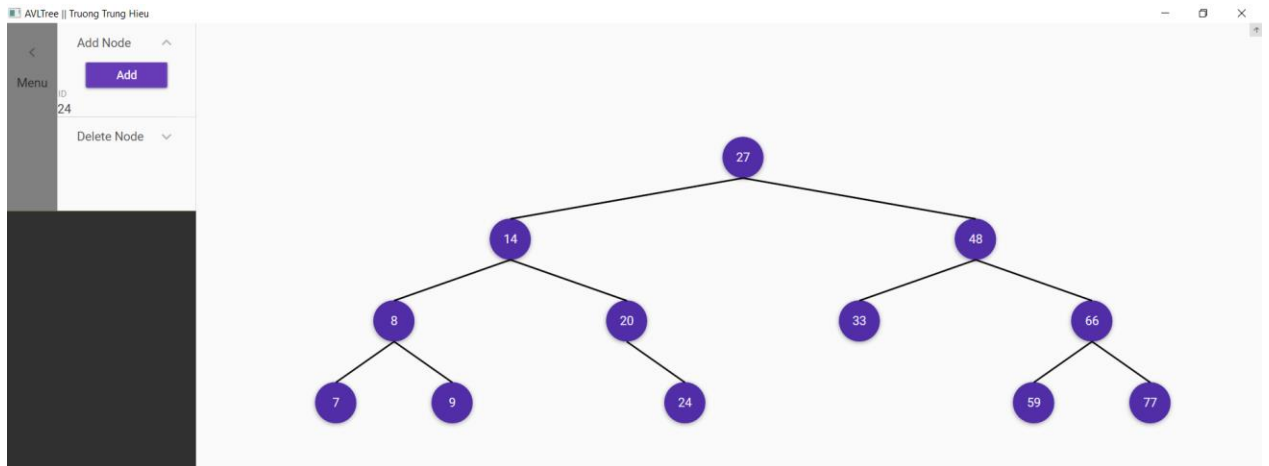
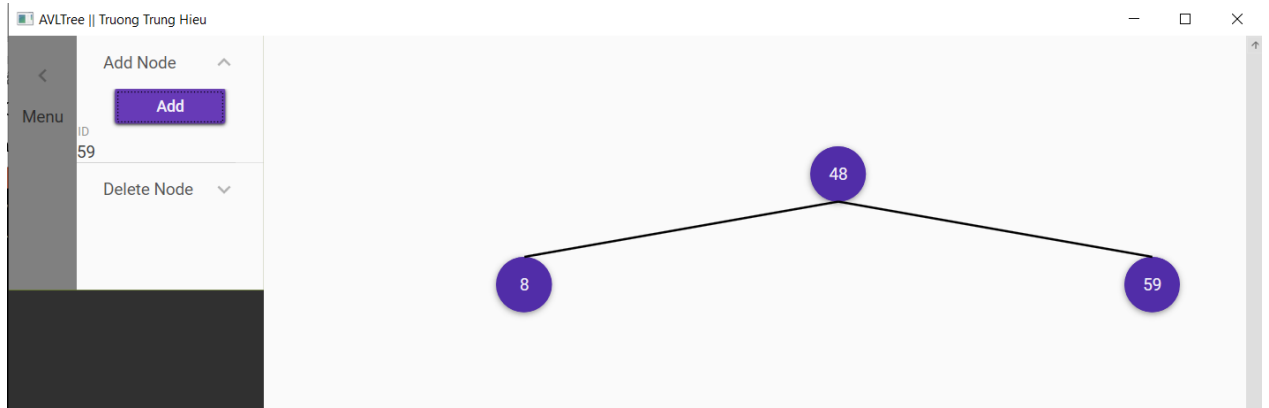
*Trước khi chèn vào nút 59*



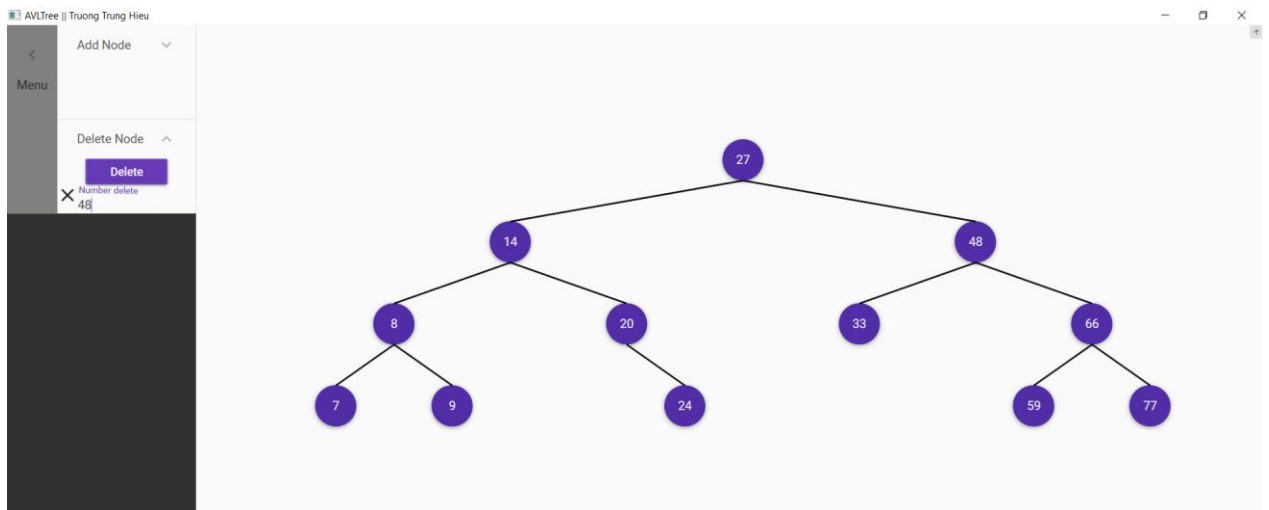
*Các nút tự di chuyển*



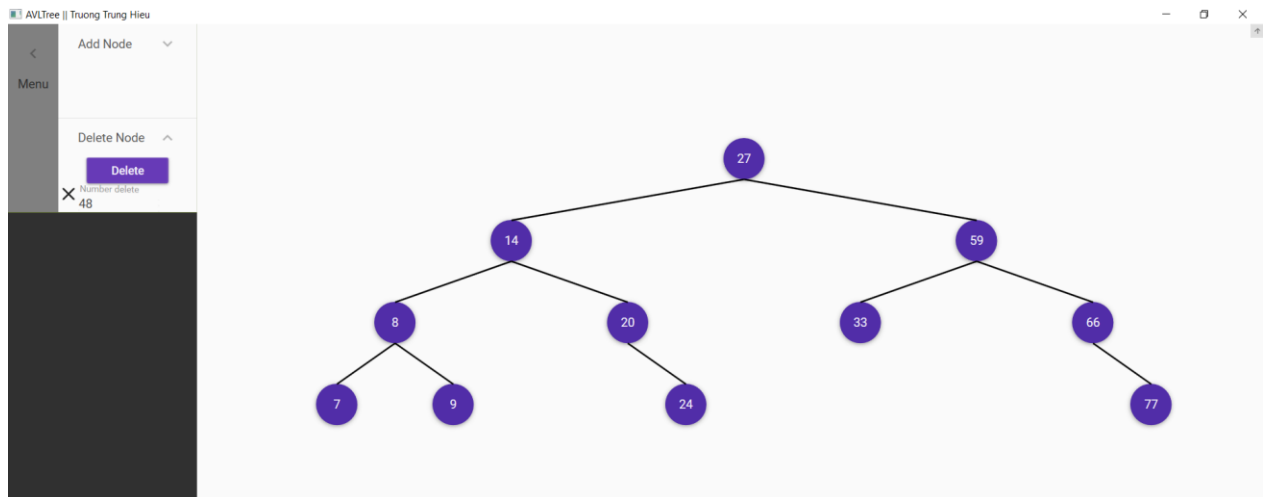
***Và kết quả:***



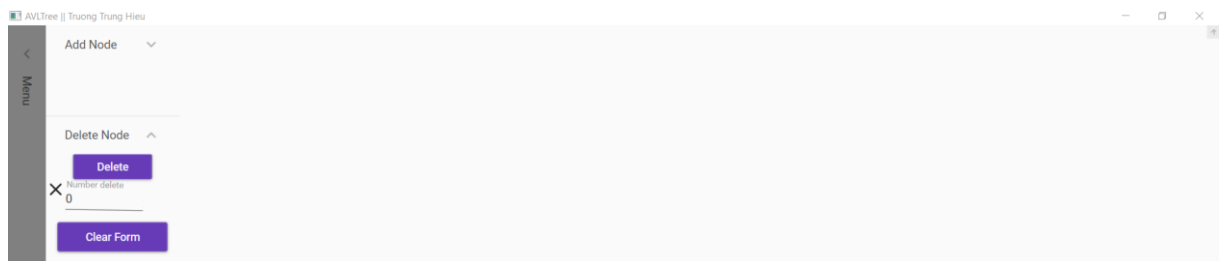
***Chức năng Delete nút 48:***



## *Sau khi Delete*



## *Button Clear Form*



*Với chức năng xoá tất cả các nút trên Panel, và thiết lập lại Panel.*

## CHƯƠNG 4: KẾT LUẬN

### 4.1 KẾT QUẢ ĐẠT ĐƯỢC

Sau thời gian nghiên cứu và tìm hiểu đề tài, cùng với sự hướng dẫn tận tình của thầy Triệu Vĩnh Viêm. Hôm nay, đề tài đã được hoàn thành và đạt được một số kết quả như sau:

- ✓ Tôi đã được đi sâu vào tìm hiểu cây nhị phân và giải thuật Quay cây một cách rõ ràng hơn.
- ✓ Hiểu và cài đặt thuật toán đã được yêu cầu bằng ngôn ngữ C# biết cách sử dụng các thao tác và các hàm trong C#.
- ✓ Chương trình chạy ổn định, giao diện thân thiện với người dùng và dễ sử dụng.
- ✓ Sau khi được thầy hướng dẫn, gợi ý thêm một số chức năng, thì tôi cũng đã làm thêm được chức năng Clear Form.
- ✓ Tích lũy thêm được kinh nghiệm, vốn hiểu biết trong quá trình làm niên luận, để hỗ trợ trong việc làm các niên luận sau.

### 4.2 HẠN CHẾ

- ❖ Có thể giao diện còn chưa đáp đầy đủ các chức năng người sử dụng yêu cầu.
- ❖ Kích thước Pane vẫn chưa được hoàn thiện với việc chèn hơn 15 nút.

### 4.3 HƯỚNG PHÁT TRIỂN

- ✚ Cải tiến chương trình đầy đủ và hoàn thiện hơn.
- ✚ Học tập và trau dồi thêm nữa về kiến thức lập trình C#.

## TÀI LIỆU THAM KHẢO

- [1] [https://vi.wikipedia.org/wiki/C%C3%A2y\\_AVL](https://vi.wikipedia.org/wiki/C%C3%A2y_AVL)
- [2] **Đỗ Xuân Lôi.** *Cấu trúc dữ liệu và giải thuật.* Chương 2. NXB Đại học Quốc gia Hà Nội, 2006.
- [3] <https://www.cs.usfca.edu/~galles/visualization/AVLtree.html>
- [4] **Steven S. Skiena,** The Algorithm Design Manual 2<sup>nd</sup>, Springer, 2008.
- [5] **Nguyễn Văn Linh.** *Giải thuật.* Trang 45-49. Đại học Cần Thơ, 2003.