**TRƯỜNG ĐẠI HỌC NHA TRANG**

**KHOA CÔNG NGHỆ THÔNG TIN**

Logo, icon

Description automatically generated

**BÁO CÁO THỰC TẬP CƠ SỞ**

**MÔ PHỎNG AVL TREE**

**Giảng viên hướng dẫn: GV. Nguyễn Mạnh Cương**

**Sinh viên thực hiện: Phạm Tấn Hải**

**Mã số sinh viên: 63133878**

Khánh Hòa - 2024

BỘ GIÁO DỤC VÀ ĐÀO TẠO

**TRƯỜNG ĐẠI HỌC NHA TRANG**

**KHOA CÔNG NGHỆ THÔNG TIN**

Logo, icon

Description automatically generated

**BÁO CÁO THỰC TẬP CƠ SỞ**

**MÔ PHỎNG AVL TREE**

GVHD: GV. Nguyễn Mạnh Cương

SVTH: Phạm Tấn Hải

MSSV: 63133878

Khánh Hòa, Tháng 1/2024

LỜI CAM ĐOAN

Tôi xin cam đoan mọi kết quả của đề tài “MÔ PHỎNG” là công trình nghiên cứu của cá nhân tôi và chưa từng được công bố trong bất cứ công trình khoa học nào khác cho tới thời điểm này.

Khánh Hòa, ngày 03 tháng 01 năm 2024

Tác giả báo cáo

*(ký và ghi rõ họ tên)*

LỜI CẢM ƠN

Trong suốt thời gian thực hiện đề tài, tôi đã nhận được sự giúp đỡ, hướng dẫn tận tình của thầy Nguyễn Mạnh Cương đã giúp tôi hoàn thành tốt đề tài. Qua đây, tôi xin gửi lời cảm ơn sâu sắc đến sự giúp đỡ này

Cuối cùng tôi xin gửi lời cảm ơn chân thành đến gia đình và tất cả bạn bè đã giúp đỡ, động viên tôi trong suốt quá trình học tập và thực hiện đề tài.

Tôi xin chân thành cảm ơn!

Khánh Hòa, ngày 03 tháng 01 năm 2024

Tác giả báo cáo

*(ký và ghi rõ họ tên)*

# **MỤC LỤC**

[LỜI CAM ĐOAN i](#_Toc155716359)

[LỜI CẢM ƠN ii](#_Toc155716360)

[MỤC LỤC iii](#_Toc155716361)

[DANH MỤC HÌNH ẢNH v](#_Toc155716362)

[DANH MỤC BẢNG BIỂU vi](#_Toc155716363)

[PHẦN MỞ ĐẦU 1](#_Toc155716364)

[Chương 1. TỔNG QUAN VỀ MÔ PHỎNG AVL TREE 2](#_Toc155716365)

[Chương 2. CƠ SỞ LÝ THUYẾT 3](#_Toc155716366)

[2.1 CÂY NHỊ PHÂN TÌM KIẾM 3](#_Toc155716367)

[2.1.1 Định nghĩa và khái niệm về cây nhị phân 3](#_Toc155716368)

[2.1.2 Cây nhị phân tìm kiếm 3](#_Toc155716369)

[2.1.2.1 Định nghĩa và tính chất 3](#_Toc155716370)

[2.1.2.2 Giải thuật tìm kiếm 4](#_Toc155716371)

[2.1.2.3 Giải thuật bổ sung 4](#_Toc155716372)

[2.1.2.3 Giải thuật loại bỏ 4](#_Toc155716373)

[2.1.2.3 Phân tích đánh giá 5](#_Toc155716374)

[2.1.3 Cây nhị phân tìm kiếm cân bằng (AVL) 5](#_Toc155716375)

[2.1.3.1 Định nghĩa 5](#_Toc155716376)

[2.1.3.2 Các trường hợp gây mất cân bằng trên cây AVL 5](#_Toc155716377)

[2.1.3.3 Giải thuật bổ sung trên cây AVL 7](#_Toc155716378)

[2.1.3.4 Giải thuật loại bỏ trên cây AVL 8](#_Toc155716379)

[2.1.3.5 Đánh giá 8](#_Toc155716380)

[Chương 3. MÔ PHỎNG DỰ ÁN 9](#_Toc155716381)

[3.1 LÝ THUYẾT MÔ PHỎNG 9](#_Toc155716382)

[3.1.1 Định nghĩa mô phỏng thuật toán 9](#_Toc155716383)

[3.1.2 Mục đích của mô phỏng thuật toán 9](#_Toc155716384)

[3.1.3 Yêu cầu về mô phỏng thuật toán 9](#_Toc155716385)

[3.1.3.1 Phản ánh đúng nội dung của thuật toán 9](#_Toc155716386)

[3.1.3.2 Có thể thực hiện giải thuật theo từng bước một để theo dõi giá trị của các biến và các đối tương trong bài toán 9](#_Toc155716387)

[3.1.3.3 Có hình ảnh động (có thể có âm thanh khi cần) để mô tả trực tiếp quá trình thi hành của thuật toán 9](#_Toc155716388)

[3.1.3.4 Có thể kiểm định thuật toán trong trường hợp ngẫu nhiên, trường hợp xấu nhất, trường hợp tốt nhất 10](#_Toc155716389)

[3.2 MÔ PHỎNG AVL TREE 10](#_Toc155716390)

[3.2.1 Cấu trúc dữ liệu lưu trữ 10](#_Toc155716391)

[3.2.1.1 Ngôn ngữ lập trình được sử dụng 10](#_Toc155716392)

[3.2.1.2 Phân tích giải thuật và đưa ra cấu trúc dữ liệu 10](#_Toc155716393)

[3.2.2 Xây dựng mô hình mô phỏng dữ liệu vào ra 11](#_Toc155716394)

[3.2.2.1 Xây dựng mẫu dữ liệu vào 11](#_Toc155716395)

[3.2.2.2 Xây dựng mẫu dữ liệu ra 11](#_Toc155716396)

[3.2.3 Sản phẩm mẫu 13](#_Toc155716397)

[3.2.4 Đánh giá ý tưởng phát triển 13](#_Toc155716398)

[Chương 4. KẾT LUẬN VÀ KIẾN NGHỊ 14](#_Toc155716399)

[4.1 KẾT LUẬN 14](#_Toc155716400)

[4.2 KIẾN NGHỊ 14](#_Toc155716401)

[TÀI LIỆU THAM KHẢO 15](#_Toc155716402)

# **DANH MỤC HÌNH ẢNH**

[Hình 2. 1 Cây nhị phân 5](#_Toc155703091)

[Hình 2. 2 Cây nhị phân tìm kiếm 5](#_Toc155703092)

[Hình 2. 3 Cây nhị phân tìm kiếm cân bằng 7](#_Toc155703093)

[Hình 2. 4 Cây lệch trái 7](#_Toc155703094)

[Hình 2. 5 Cây lệch phải 7](#_Toc155703095)

[Hình 2. 6 Các trường hợp cân bằng 8](#_Toc155703096)

[Hình 2. 7 Lệch trái trái 9](#_Toc155703097)

[Hình 2. 8 Lệch trái phải 9](#_Toc155703098)

[Hình 3. 1 Hàm OnPaint() 28](#_Toc155713211)

[Hình 3. 2 Winform chính 29](#_Toc155713212)

# **DANH MỤC BẢNG BIỂU**

[Bảng 3. 1 Thuộc tính Node 26](#_Toc155713247)

[Bảng 3. 2 Thuộc tính NodeInfo 26](#_Toc155713248)

[Bảng 3. 3 Code cân bằng 27](#_Toc155713249)

PHẦN MỞ ĐẦU

Trong thế kỷ 21, sự phát triển nhanh chóng của công nghệ thông tin và ứng dụng của nó trong nhiều lĩnh vực đòi hỏi sự hiệu quả và linh hoạt trong quản lý dữ liệu. Cây nhị phân cân bằng (AVL Tree), đóng góp rất lớn trong việc tối ưu hóa thời gian truy xuất dữ liệu. Mặc dù đã tồn tại nhiều cách tiếp cận, việc mô phỏng cây AVL vẫn giữ vững giá trị nghiên cứu để cải thiện và tối ưu hóa hiệu suất giáo dục trong quá trình giảng dạy của giảng viên, cũng như quá trình học tập của học sinh, sinh viên.

Cây AVL (viết tắt của tên các nhà phát minh Adelson, Velski và Landis) [1] là cây tìm kiếm nhị phân có độ cân bằng cao. Cây có các chức năng cơ bản tương tự cây nhị phân tìm kiếm (Binary Search Tree) thêm và xóa các giá trị biến động, tạo thành một cây có gốc và phát triển rộng theo chiều ngang. Tuy nhiên trong quá trình phát triển, cây nhị phân tìm kiếm có khuynh hướng suy biến, việc sửa lại cây cho cân đối nếu được tiến hành thường xuyên sẽ gây tổn phí khá nhiều về thời gian và công sức. Vì vậy sự ra đời của cây nhị phân tìm kiếm cân bằng là một bước phát triển rất lớn trong việc tối ưu hóa quá trình quản lý và lưu trữ dữ liệu. Việc mô phỏng cây AVL là điều cần thiết với sự chi tiết và hiểu biết sâu sắc về cơ chế cân bằng của nó. Nghiên cứu sẽ đánh giá hiệu suất của cây AVL trong nhiều điều kiện và tình huống khác nhau để cung cấp cái nhìn toàn diện về tính linh hoạt và khả năng ứng dụng trong thực tế. Đồng thời, phạm vi mở rộng đến việc áp dụng kiến thức từ mô phỏng vào các lĩnh vực ứng dụng như cơ sở dữ liệu và hệ thống quản lý.

Trong quá trình nắm bắt kiến thức về các giải thuật và đặc biệt là môn cấu trúc dữ liệu, chúng ta thường nhận thức rằng nhiều giải thuật mang tính phức tạp, trừu tượng và khó hình dung vấn đề. Điều này thường làm cho quá trình học trở nên khó khăn. Vì vậy, luôn tồn tại mong muốn trong quá trình học giải thuật có sự hỗ trợ từ những mô phỏng trực quan, giúp chúng ta nắm bắt kiến thức một cách dễ dàng hơn. Sự hiểu biết tốt về giải thuật không chỉ mang lại lợi ích trong việc tư duy giải thuật mà còn hỗ trợ phát hiện vấn đề một cách nhanh chóng. Điều quan trọng là, việc hiểu giải thuật có thể giúp chúng ta học tốt các môn có tính logic cao, tạo thuận lợi cho quá trình nắm vững kiến thức. Mặc dù mong muốn này tồn tại, nhưng thực tế cho thấy việc học giải thuật không phải là điều dễ dàng với nhiều người. Vì vậy lý do để chọn đề tài này là để giúp người học tiếp thu kiến thức giải thuật một cách dễ dàng hơn. Những công cụ mô phỏng này không chỉ giúp học viên hình dung rõ hơn về cách giải thuật hoạt động mà còn tạo ra môi trường thực tế, hỗ trợ quá trình học tập và nắm bắt kiến thức một cách hiệu quả. Tuy nhiên, việc xây dựng ứng dụng mô phỏng cây nhị phân cân bằng không vì mục đích cải tiến hay nâng cấp so với các dự án đã có. Dự án này vì mong muốn hiểu rõ hơn thuật toán cây nhị phân cân bằng và thành thạo hơn công cụ được sử dụng để phát triển dự án.

# **Chương 1. TỔNG QUAN VỀ MÔ PHỎNG AVL TREE**

Trong lĩnh vực khoa học máy tính và cấu trúc dữ liệu, cây AVL đóng vai trò quan trọng trong việc quản lý và tối ưu hóa dữ liệu. Được giới thiệu bởi Adelson-Velsky và Landis vào năm 1962 [1], cây AVL là một cấu trúc dữ liệu cây nhị phân cân bằng, nơi mỗi node trong cây có sự cân bằng chiều cao giữa các nhánh con trái và phải.

Hầu hết các thao tác trên cây tìm kiếm nhị phân (BST), bao gồm tìm kiếm, tìm giá trị lớn nhất, tìm giá trị nhỏ nhất, thêm, xóa và các thao tác khác, đều yêu cầu thời gian O(h), trong đó h là chiều cao của cây BST. Trong trường hợp cây nhị phân không cân bằng, chi phí của những thao tác này có thể tăng lên O(n). Chúng ta có thể cung cấp một giới hạn trên là O(log(n)) cho tất cả các thao tác này nếu chúng ta đảm bảo rằng chiều cao của cây luôn là O(log(n)) sau mỗi thêm và xóa. Chiều cao của cây AVL luôn là O(log(n)), trong đó n là số lượng nút trong cây. [2]

Ưu điểm của AVL Tree là Chiều cao của cây AVL không bao giờ vượt quá log(N), trong đó N là tổng số nút trong cây, do chiều cao luôn được cân bằng. So với cây tìm kiếm nhị phân thông thường, nó cung cấp một độ phức tạp thời gian tìm kiếm cao cấp. Cây AVL có khả năng tự cân bằng. Trong thực tế AVL Tree được sử dụng nhiều trong quản lý dữ liễu với quy mô nhỏ: hầu hết các tập hợp và từ điển trong bộ nhớ được lưu trữ bằng cách sử dụng cây AVL. Ứng dụng cơ sở dữ liệu, nơi thêm và xóa dữ liệu ít nhưng việc tra cứu dữ liệu thường xuyên là cần thiết, cũng thường sử dụng cây AVL. Ngoài các ứng dụng cơ sở dữ liệu, nó được sử dụng trong các ứng dụng khác đòi hỏi khả năng tìm kiếm tốt hơn. Nó chủ yếu được sử dụng trong các lĩnh vực kinh doanh nơi cần phải giữ bản ghi về nhân viên làm việc và sự thay đổi ca làm việc của họ. [2]

Từ những điều trên cho thấy việc xây dựng ứng dụng mô phỏng cây AVL thự sự quan trong cho việc tiếp cận và thấu hiểu thuật toán, nhằm thích ứng với nhiều ứng dụng mà cây AVL đem lại. Hiện nay đã có rất nhiều dự án mô phỏng cây nhị phân tìm kiếm cân bằng với nhiều ngôn ngữ và công cụ khác nhau. Với cùng thuật toán, nhưng tối ưu hơn về mặt giao diện giúp người dùng dễ dàng hình dung được quy trình cũng như hiểu được các bước tự cân bằng của cây nhị phân cân bằng khi cấu trúc cây thay đổi.

# **Chương 2. CƠ SỞ LÝ THUYẾT**

## **2.1 CÂY NHỊ PHÂN TÌM KIẾM**

### **2.1.1 Định nghĩa và khái niệm về cây nhị phân**

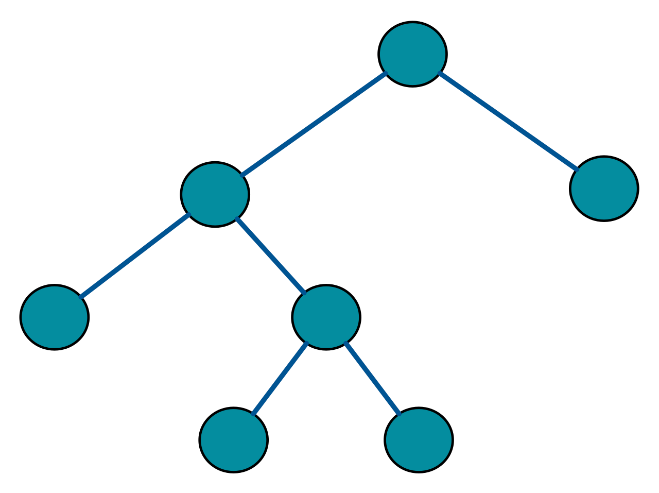
Cây nhị phân là cây mà các nút chỉ có tối đa 2 con. Đối với cây con có một nút thì người ta phân biệt cây con trái và cây con phải. Vì vây cây nhị phân là cây có thứ tự.[3]

-Số nút ở mức i <= 2i.

-Số nút ở mức lá <= 2h-1, với h là chiều cao của cây.

-Chiều cao của cây h >= log2(số nút trong cây).

-Số nút trong cây <= 2h-1.

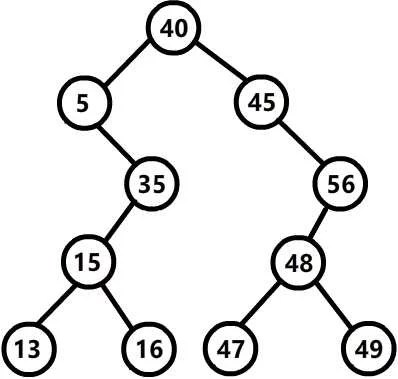


**Hình 2. 1 Cây nhị phân**

### **2.1.2 Cây nhị phân tìm kiếm**

#### **2.1.2.1 Định nghĩa và tính chất**

Cây nhị phân tìm kiếm là cây nhị phân trong đó tại mỗi nút, khóa của nút đang xét lớn hơn khóa của tất cả các nút thuộc cây con trái và nhỏ hơn khóa của tất cả các nút thuộc cây con phải.



**Hình 2. 2 Cây nhị phân tìm kiếm**

Nhờ ràng buộc về khóa trên cây nhị phân tìm kiếm, việc tìm kiếm trở nên có định hướng. Hơn nữa, do cấu trúc cây việc tìm kiếm trở nên nhanh đáng kể. Nếu số nút trên cây là N thì chi phí tìm kiếm trung bình chỉ khoảng log2(N). Trong thực tế, khi xét đến cây nhị phân chủ yếu người ta xét cây nhị phân tìm kiếm.[3]

#### **2.1.2.2 Giải thuật tìm kiếm**

Giả sử, ta muốn biết liệu trên cây tìm kiếm nhị phân có nút nào chứa khoá K hay không. Ta sẽ bắt đầu duyệt từ nút gốc của cây (Nút gốc có khoá N). Nếu K > N, thi chuyển sang nhánh phải và tiếp tục quá trình so sánh. Nếu K < N, thì chuyển sang nhánh trái và tiếp tục quá trình so sánh. Quá trình tìm kiếm sẽ dừng lại, khi xảy ra một trong hai trường hợp sau:

-K = N. Tức là tìm thấy nút có giá trị khoa bằng K.

-Con trỏ trỏ đến Null. Tức là, trên cây tìm kiếm nhị phân không có nút nào có giá trị khoá bằng K.[4]

#### **2.1.2.3 Giải thuật bổ sung**

Dựa vào giá trị khoá của nút cần chèn để xác định vị trí chính xác của nút đó. Giả sử nút cần chèn có giá trị khoá là V, nút gốc của cây có giá trị khoá là N. Nếu V > N thì ta đi theo nhánh phải và tiếp tục quá trình so sánh. Nếu V < N thì ta đi theo nhánh trái và tiếp tục quá trình so sánh. Quá trình này sẽ dừng lại khi xảy ra một trong hai trường hợp sau:

-V = N. Trong trường hợp này, dữ liệu cần chèn đã có trong cây. Vì vậy, ta không cần chèn thêm nút mới.

-Con trỏ trò đến Null. Tức là ta đã tìm đến vị trí chính xác cho nút mới.[4]

#### **2.1.2.3 Giải thuật loại bỏ**

Giả sử ta muốn xóa một nút có giá trị khoá là X, ta tiến hành tìm kiểm trên cây bắt đầu từ nút gốc: nếu X lớn hơn nhẫn của nút gốc thì ta tìm sang cây con bên phải, ngược lại thì ta sẽ tìm sang cây con bên trái. Nếu không tìm thấy thì giải thuật kết thúc.

Nếu tìm gặp thì có 3 trường hợp sau:

-Nếu X là lá thì ta thay x bằng Null.

-Nếu X chỉ có một nút con thì ta thay X bằng nút con của nó.

-Nếu X có 2 con thì ta thay X bằng nút lớn nhất trên cây con bên trái (nút cực phải của cây trái) hoặc nút bé nhất trên cây con bên phải của X (nút cực trái của cây phải).[4]

#### **2.1.2.3 Phân tích đánh giá**

Tất cả các thao tác tìm kiếm, bổ sung, loại bỏ trên cây nhị phân tìm kiếm đều có độ phức tạp trung bình O(h), với h là chiều cao của cây

Trong trong trường hợp tốt nhất, cây nhị phân tìm kiếm có n nút sẽ có độ cao h = log2(n). Chi phí tìm kiếm khi đó sẽ tương đương tìm kiếm nhị phân trên mảng có thứ tự.

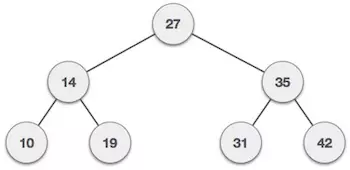
Tuy nhiên, trong trường hợp xấu nhất, cây có thể bị suy biển thành 1 dạnh sách liên kết (khi mà mỗi nút đều chỉ có 1 con trừ nút lá). Lúc đó các thao tác trên sẽ có độ phức tạp O(n). Vì vậy cần có cải tiến cấu trúc của cây nhị phân tìm kiếm để đạt được chi phí cho các thao tác là log2(n).

### **2.1.3 Cây nhị phân tìm kiếm cân bằng (AVL)**

#### **2.1.3.1 Định nghĩa**

Cây nhị phân tìm kiếm cân bằng là cây mà tại mỗi nút của nó độ cao của cây con trái và của cây con phải chênh lệch không quá một.

Cây nhị phân tìm kiếm mà luôn có dạng cân đối AVL, thì chi phi tìm kiếm đối với nó ngay trong trường hợp xấu nhất vẫn là O(log2n).[1]

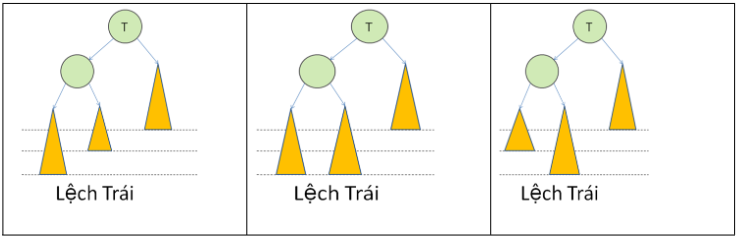


**Hình 2. 3 Cây nhị phân tìm kiếm cân bằng**

Từ khi được giới thiệu, cây AVL đã nhanh chóng tìm thấy ứng dụng trong nhiều bài toán khác nhau. Vì vậy, nó mau chóng trở nên thịnh hành và thu hút nhiều nghiên cứu. Từ cây AVL, người ta đã phát triển thêm nhiều loại cấu trúc dữ liệu hữu dụng khác như cây đỏ-đen (Red-Black Tree), B-Tree,…

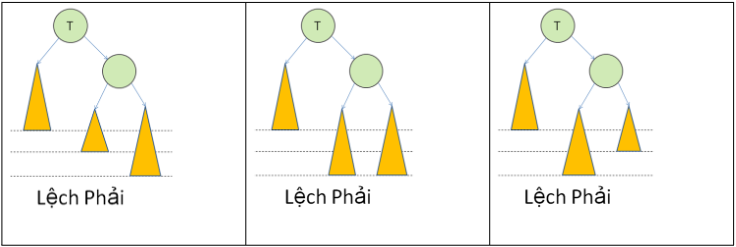
#### **2.1.3.2 Các trường hợp gây mất cân bằng trên cây AVL**

Trường hợp 1: cây lệch trái

****

**Hình 2. 4 Cây lệch trái**

Trường hợp 2: cây lệch phải



**Hình 2. 5 Cây lệch phải**

Ta có thể thấy rằng các trường hợp lệch về bên phải hoàn toàn đối xứng với các trường hợp lệch về bên trái. Vì vậy ta chỉ cần khảo sát trường hợp lệch về bên trái.

Phương pháp cân bằng cây khi cây bị mất cân bằng.[1]



**Hình 2. 6 Các trường hợp cân bằng**

#### **2.1.3.3 Giải thuật bổ sung trên cây AVL**

Việc đi theo đường tìm kiếm cây để thấy được khóa mới chưa có sẵn trên cây và chưa biết được chổ để bổ sung nó vào, tất nhiên được thực hiện tương tự như việc bổ sung một node vào trong cây nhị phân. Sau khi node mới được bổ sung, có ba tình huống có thể xảy ra với các node cha của nó. Để tiện trình bày, ta giả sử phép bổ sung được thực hiện vào phía trái.

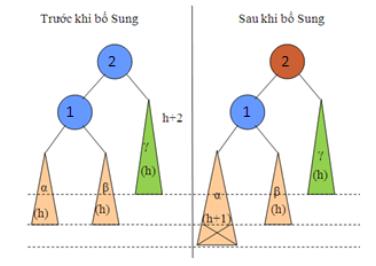
Như vậy ba tình huống đó có thể nêu cụ thể như sau:

-Tình huống 1: Cây con phải đã cao hơn 1 (lệch phải) sau phép bổ sung chiều cao hai cây con bằng nhau. Trường hợp này ta chỉ cần chỉnh lại hệ số cân bằng tại nút đang xét.

-Tình huống 2: Chiều cao của hai cây con vốn đã bằng nhau, sau phép bổ sung cây con trái cao hơn 1 (lệch trái). Trường hợp này chiều cao của cây gốc là node đang xét bị thay đồi, nên không chỉ phải chỉnh lý hệ số cân đối nút đang xét mà còn phải chỉnh lý hệ số cân đối ở các node cha của nó.

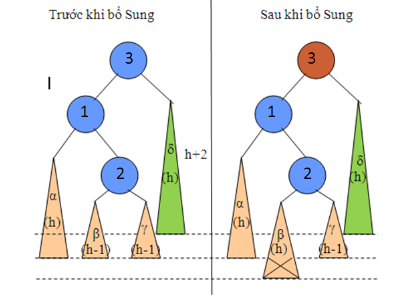
-Tình huống 3: Cây con trái đã cao hơn 1 (lệch trái), sau phép bổ sung nó cao hơn 2: tính “cân bằng AVL" bị phá vỡ vậy ta phải cân bằng lại bằng phép xoay. Có hai trường hợp phải xử lý khác nhau:

-TH1: Node mới bổ sung làm tăng chiều cao cây con trái của node con trái node bất thường. Tái cân bằng giống như trong trường hợp 1.



**Hình 2. 7 Lệch trái trái**

-TH1: Node mới bổ sung làm tăng chiều cao cây con phải của node con trái node bất thường. Tái cân bằng giống như trong trường hợp 3.[1]



**Hình 2. 8 Lệch trái phải**

#### **2.1.3.4 Giải thuật loại bỏ trên cây AVL**

Loại bỏ giống như giải thuật loại bỏ của cây nhị phân, chỉ khác là sau khi loại bỏ cây bị mất cân đối và phải tái cân đối bằng phép quay như đã làm khi bổ sung. Việc huỷ 1 nút có thể phải cân bằng dây chuyền các nút từ gốc cho đến phần tử bị huỷ trong khi thêm vào chỉ cần 1 lần cân bằng cục bộ.

#### **2.1.3.5 Đánh giá**

Cây AVL với chiều cao được khống chế sẽ cho phép thực thi các thao tác tìm thêm hủy với chi phí O(log2(n)) và bảo đảm không suy biển thành O(n).

# **Chương 3. MÔ PHỎNG DỰ ÁN**

## **3.1 LÝ THUYẾT MÔ PHỎNG**

### **3.1.1 Định nghĩa mô phỏng thuật toán**

Mô phỏng thuật toán là quá trình tách dữ liệu, thao tác vào tạo giao diện đồ họa mô phỏng cho quá trình đó.[5]

### **3.1.2 Mục đích của mô phỏng thuật toán**

Mô phỏng thuật toán sử dụng đồ hoạ để mô tả các cấu trúc dữ liệu bên trong của thuật toán được thực hiện trong chương trình và biểu diễn sự thay đổi của các cấu trúc dữ liệu trong mỗi trạng thái thực thi và các hoạt động của chương trình. Trong suốt quá trình mô tả, người sử dụng có thể thấy từng bước thực hiện của chương trình và có thể thấy được những chi tiết nhỏ của thuật toán và hiểu sâu hơn về nó. Vì thế, mô phòng thuật toán giúp cho người sử dụng hiểu thuật toán.

Trong quá trình mô phòng, người sử dụng có thể thấy chương trình của họ được thi hành như thế nào, đánh giá sự thay đổi của dữ liệu qua mỗi bước và nó sẽ ảnh hưởng đến bước tiếp theo như thế nào. Nó giúp người sử dụng phát hiện ra lỗi của chương trình (nếu có), từ đó có thể tìm ra lỗi và sửa lại chương trình để nó chạy chính xác và ổn định hơn.[5]

### **3.1.3 Yêu cầu về mô phỏng thuật toán**

#### **3.1.3.1 Phản ánh đúng nội dung của thuật toán**

Thuật toán được mô phòng phải chính xác, các bước của thuật toán phải trực quan và phản ánh đứng nội dung của thuật toán để thể hiện tính đúng đắn của thuật toán.

#### **3.1.3.2 Có thể thực hiện giải thuật theo từng bước một để theo dõi giá trị của các biến và các đối tương trong bài toán**

Quá trình mô phỏng có thể diễn ra liên tục, biểu diễn thuật toán từ đầu đến cuối. Tuy nhiên, trong quá trình mô phỏng thuật toán chúng ta đều có nhu cầu theo dõi các bước của giải thuật xem nó chạy đúng hay chưa, các biến và các đối tượng thay đổi như thế nào. Do đó trong khi thiết kế giải thuật chương trình cần phải có nút tạm dừng để dùng chương trình và nút tiếp tục để tiếp tục quá trình mô phỏng.

#### **3.1.3.3 Có hình ảnh động (có thể có âm thanh khi cần) để mô tả trực tiếp quá trình thi hành của thuật toán**

Quá trình mô tả trực quan quá trình thì hành của thuật toán là để biết bản chất bên trong của vấn đề là gì, điều này đồng nghĩa với việc phải có hình ảnh động để mô tả thuật toán hoạt động như thế nào, các biên thay đổi ra sao. từ đó mới kích thích tư duy sáng tạo của học sinh và thu hút sự chú ý của người học. Bên cạnh đó để cho quá trình mô phỏng thêm sinh động chúng ta có thể chèn thêm âm thanh vào phần mềm, có các nút chỉnh âm thanh để lúc nào cần chúng ta có thể bật hay tắt một cách chủ động.

#### **3.1.3.4 Có thể kiểm định thuật toán trong trường hợp ngẫu nhiên, trường hợp xấu nhất, trường hợp tốt nhất**

Để kiểm định thuật toán thì ta phải thử với các bộ dữ liệu vào ngẫu nhiên hoặc các bộ dữ liệu mẫu hoặc các bộ dữ liệu do người dùng nhập vào. Nếu kết quả chạy chương trình vẫn ổn định và thuật toán vẫn đúng đắn thì khi đó chương trình mới được đánh giá cao.[5]

## **3.2 MÔ PHỎNG AVL TREE**

### **3.2.1 Cấu trúc dữ liệu lưu trữ**

#### **3.2.1.1 Ngôn ngữ lập trình được sử dụng**

C# (C Sharp) là một ngôn ngữ lập trình hướng đối tượng đa năng, mạnh mẽ kết hợp với Windows Forms (WinForms) là thư viện lớp đồ họa (GUI) mã nguồn mở và miễn phí được bao gồm như một phần của Microsoft.NET Framework [6]. Tạo ra một ứng dụng chạy trên giao diện Windown phù hợp cho quá trình học tập và giảng dạy.

#### **3.2.1.2 Phân tích giải thuật và đưa ra cấu trúc dữ liệu**

Cây AVL được cài đặt bằng con trỏ. Mỗi nút của cây là một đối tượng có tên là **Node** gồm các thuộc tính sau:

|  |  |
| --- | --- |
| Thuộc tính | Ý nghĩa |
| **Value** | Giá trị của Node |
| **Left** | Con trái của Node |
| **Right** | Con phải của Node |

**Bảng 3. 1 Thuộc tính Node**

Trong trường hợp không có con trái / phải thì **Left** và **Right** trỏ vào Null.

Mỗi **Node** sẽ đi kèm với một **NodeInfo** chứa thông tin vị trí được tính toán dựa trên giá trị so sánh với **Node** cha và **Node** kề với nó.

|  |  |  |  |
| --- | --- | --- | --- |
| Thuộc tính | Ý nghĩa | Thuộc tính | Ý nghĩa |
| **Value** | Giá trị của Node | **IsRightChild** | Node là con phải |
| **Height** | Chiều cao của Node | **Post** | Vị trí Node |
| **IsLeaf** | Node là lá | **LeftChildPost** | Vị trí con trái |
| **IsLetfChild** | Node là con trái | **RightChildPost** | Vị trí con phải |

**Bảng 3. 2 Thuộc tính NodeInfo**

**IsLeaf** sẽ bằng **true** nếu không có cả con trái và con phải.

**IsLeftChild / IsRightChild** sẽ bằng **true** nếu là con trái / phải của **Node** cha trước đó.

**LeftChildPost và RightChildPost** được tính dựa vào **Post** bằng độ lệch của đường kính hình tròn, sao cho vị trí của các **Node** tương đối trên giao diện đồ họa.

Với giá trị của **NodeInfo,** các thuộc tính vị trí được tính toán lại mỗi khi cây thay đổi cấu trúc và được cân bằng lại, từ những vị trí đã được tính toán hàm **DrawBox : PictureBox** sẽ đồ họa được hình ảnh của **Node** trên giao diện người dùng một cách linh hoạt.

Mỗi **Node** sẽ luôn được tính toán lại giá trị cân bằng, hàm **GetBalanceFactor**() và được cân bằng lại bằng hàm **BalanceTree()**.

Có 4 trường hợp cây lệch cần cân bằng, gồm 2 trường hợp quay đơn (trái trái, phải phải) và 2 trường hợp quay đôi (trái phải, phải trái). Vì lệch con trái và phải đối xứng nhau nên chỉ cần ghi nhớ một trường hợp.

|  |  |
| --- | --- |
| Quay phải phải | Code |
|  | private Node<TValue> RotateRR(Node<TValue> parent)  {  Node<TValue> pivot = parent.Right;  parent.Right = pivot.Left;  pivot.Left = parent;  return pivot;  } |
| Quay trái trái |  |
|  | private Node<TValue> RotateLL(Node<TValue> parent)  {  Node<TValue> pivot = parent.Left;  parent.Left = pivot.Right;  pivot.Right = parent;  return pivot;  } |

**Bảng 3. 3 Code cân bằng**

Hàm **Insert()** dùng để thêm một **Node** mới vào cây. Hàm **Remove**() dùng để xóa một **Node** đã có ra khỏi cây. Hai hàm này đều đi kèm với hàm **BalanceTree()** để cân bằng cây khi thêm hay xóa **Node** bất kì.

### **3.2.2 Xây dựng mô hình mô phỏng dữ liệu vào ra**

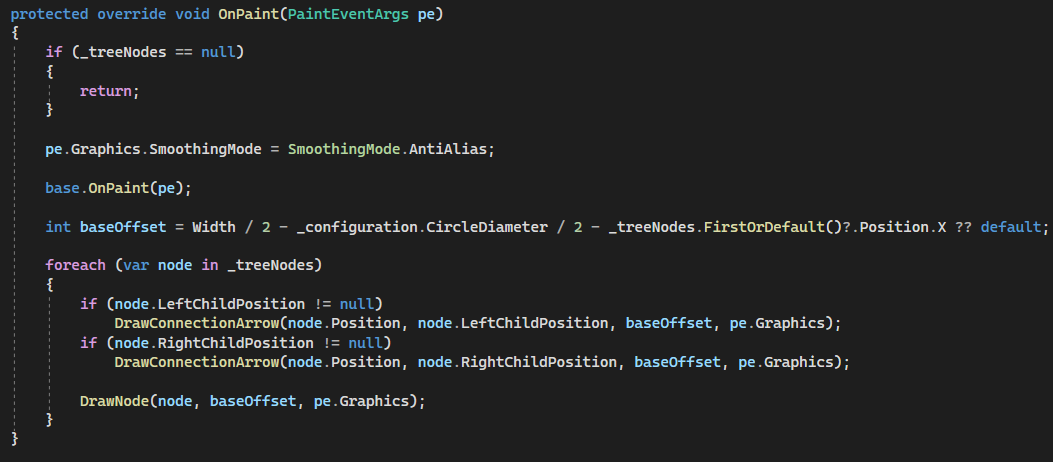
#### **3.2.2.1 Xây dựng mẫu dữ liệu vào**

Dữ liệu nhập trực tiếp: dữ liệu do người dùng nhập vào từ bàn phím, đa số thì dạng dữ liệu này thường được sử dụng vì người dùng muốn kiểm định dữ liệu do chính mình nhập vào.

#### **3.2.2.2 Xây dựng mẫu dữ liệu ra**

Sau khi thêm **Node** vào cây, **NodeInfo** sẽ được tính toán để phục vụ cho việc vẽ lên màn hình.

Khi bắt tính hiện thêm hoặc xóa **Node, DrawBox** sẽ kích thíc hàm **OnPaint()** để vẽ **Node** bằng hàm **DrawNode()** và vector nối giữa **Node** cha và con bằng hàm **DrawConnectionArrow()**.



**Hình 3. 1 Hàm OnPaint()**

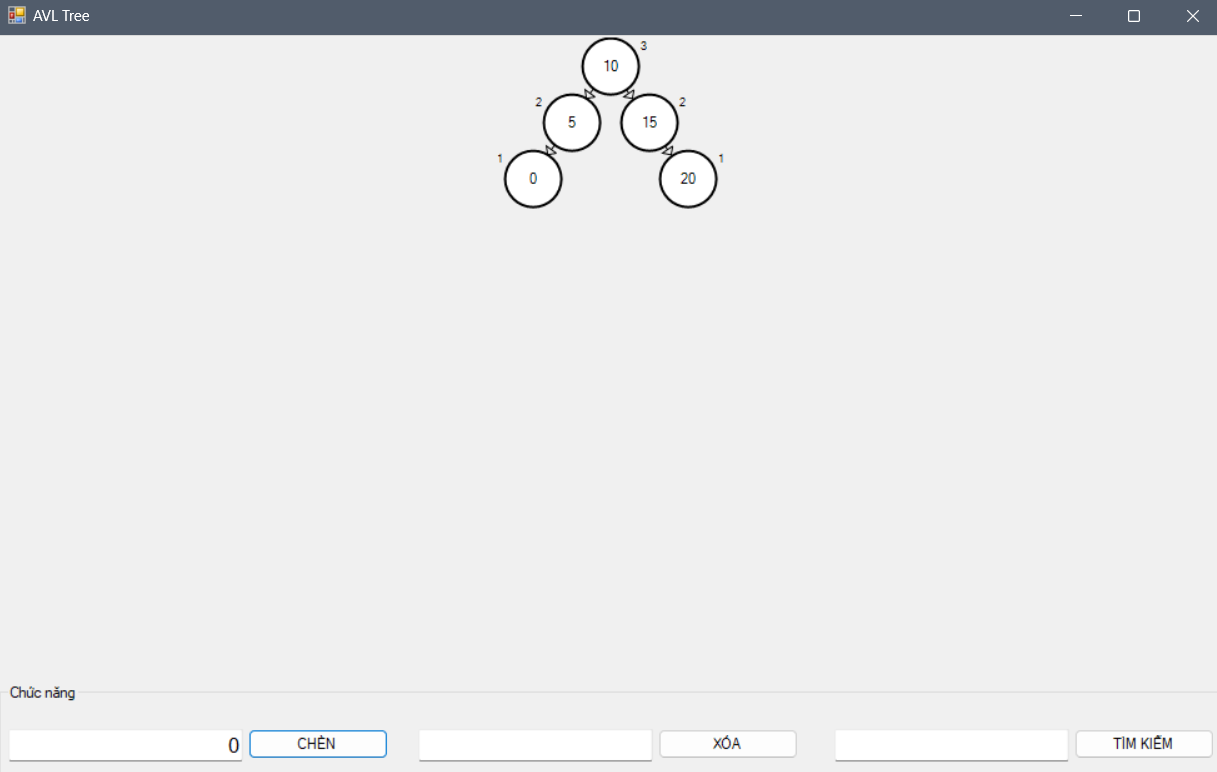
Hàm **if** đầu tiên dùng để kiểm tra cây có **Node** hay không để tiến hành vẽ.

Dòng mã **pe.Graphics.SmoothingMode = SmoothingMode.AntiAlias**: được sử dụng để thiết lập chế độ làm mượt (**anti-aliasing**) cho đối tượng **Graphics** được sử dụng để vẽ trên một bề mặt.

Dòng **base.Onpaint(pe)**: để lắng nghe sự kiện thay đổi cây.

Biến **baseOffset**: để tính độ lệch từ vị trí của **Node** cha để vẽ vector tới **Node** con, đồng thời để vẽ **Node** con trong vòng lặp **foreach**.

### **3.2.3 Sản phẩm mẫu**

****

**Hình 3. 2 Winform chính**

### **3.2.4 Đánh giá ý tưởng phát triển**

Ưu điểm:

-Phần mềm mô phỏng đã cho người dùng thấy được tổng quan về giải thuật của cây nhị phân tự cân bằng AVL.

-Phần mềm đã tạo được giao diện đồ hoạ thân thiện, dễ sử dụng, Người học có thể tiếp thu một cách nhanh chóng các tư tưởng của giải thuật

Nhược điểm:

-Phần mềm mô phỏng chưa tiện dụng với người học. Chưa có thao tác kéo thả với từng đối tượng.

-Giao diện đồ học chưa sống động, mầu sắc chưa hài hoà. Phong cách lập trình chưa có tính chuyên nghiệp

Phát triển: Phần mềm sẽ có khả năng kéo thả đối với từng đối tượng, sẽ có hệ thống menu chuyên nghiệp hơn, giao diện bao gồm cả hình ảnh động và âm thanh.

# **Chương 4. KẾT LUẬN VÀ KIẾN NGHỊ**

4.1 KẾT LUẬN

Trong quá trình triển khai dự án mô phỏng cây AVL bằng WinForms C#, chúng ta đã xây dựng một ứng dụng giúp hiểu rõ và tương tác với cây AVL - một cấu trúc dữ liệu quan trọng trong lĩnh vực khoa học máy tính. Việc mô phỏng không chỉ giúp chúng ta hình dung cách cây AVL hoạt động mà còn tạo điều kiện thuận lợi cho việc học tập và nghiên cứu.

Ứng dụng đã giúp chúng ta theo dõi quá trình thêm, xóa, và duyệt cây AVL một cách trực quan, đồng thời hiển thị chiều cao cân bằng của cây. Việc tích hợp với giao diện WinForms cung cấp trải nghiệm người dùng thân thiện, làm cho quá trình tương tác với cây AVL trở nên dễ dàng hơn.

## **4.2 KIẾN NGHỊ**

Mở rộng tính năng: nâng cao tính năng mô phỏng bằng cách thêm khả năng tùy chỉnh thao tác, chẳng hạn như xoay node và hiển thị các thông số chi tiết của cây. Tích hợp tính năng tự động cập nhật khi thao tác thêm và xóa node, giúp người dùng theo dõi sự thay đổi của cây một cách nhanh chóng.

Hiệu suất và tối ưu hóa: tối ưu hóa hiệu suất của ứng dụng để xử lý các cây AVL lớn mà vẫn duy trì trải nghiệm người dùng mượt mà. Cân nhắc việc thêm tính năng đánh giá hiệu suất để người dùng có thể theo dõi thời gian thực hiện các thao tác trên cây.

Tài liệu và hỗ trợ người dùng: cung cấp tài liệu chi tiết và hướng dẫn sử dụng để giúp người dùng hiểu rõ về ứng dụng và cách tương tác với cây AVL. Xây dựng một hệ thống hỗ trợ hoặc diễn đàn để người dùng có thể chia sẻ kinh nghiệm và đặt câu hỏi.

Mở rộng ứng dụng vào các lĩnh vực khác: xem xét khả năng tích hợp với các ứng dụng khác, như làm phong phú hơn trong lĩnh vực giáo dục hoặc nghiên cứu khoa học máy tính. Tìm cách kết hợp ứng dụng với các dự án khác để tạo ra một bộ công cụ đầy đủ và linh hoạt.

# **TÀI LIỆU THAM KHẢO**

|  |  |
| --- | --- |
| [1] | H. Cầm, “Wikipedia,” 13 6 2008. [Trực tuyến]. Available: https://vi.wikipedia.org/wiki/C%C3%A2y\_AVL. |
| [2] | Javapoint, "Javapoint," 30 10 2021. [Online]. Available: https://www.javatpoint.com/avl-tree-applications. |
| [3] | H. Cầm, “Wikipedia,” 2 9 2006. [Trực tuyến]. Available: https://vi.wikipedia.org/wiki/C%C3%A2y\_nh%E1%BB%8B\_ph%C3%A2n#cite\_note-complete\_binary\_tree-12. |
| [4] | N. V. Hiếu, "blog.luyencode," 10 1 2020. [Online]. Available: https://blog.luyencode.net/cay-tim-kiem-nhi-phan-binary-search-tree/. |
| [5] | NTech, "spiderum," 25 10 2020. [Online]. Available: https://spiderum.com/bai-dang/Mo-phong-thuat-toan-tuong-de-ma-kho-s5k. [Accessed 30 12 2023]. |
| [6] | 115.78.14.65, "Wikipedia," 10 3 2021. [Online]. Available: https://vi.wikipedia.org/wiki/Windows\_Forms. |