# Đồ án cuối năm cây AVL nhóm 15

# I/ Phần định nghĩa:

### + Lí thuyết về cây AVL:

* Là cây tìm kiếm nhị phân tự cân bằng và cấu trúc dữ liệu đầu tiên có khả năng này do luôn đảm bảo điều kiện độ cao của các cây con bên trái và cây con bên phải và bảo đảm rằng hiệu số giữa chúng là không lớn hơn 1. **Hiệu số** này được gọi là **Balance Factor (Nhân tố cân bằng)**.
* Về những phần tử của cây AVL bao gồm:

+ Node gốc (Thường phần tử nhập vào đầu tiên trong cây).

+ Các node phần tử (liên kết với nhau bởi các con trỏ trái và phải với cấu trúc theo dạng xếp dần xuống dưới, 1 node có tối đa 2 cây trái và phải->Cấu trúc dạng cây) theo qui tắc.

+ Cây con trái (với những phần tử thêm vào sẽ được so sánh với node gốc->nếu giá trị node thêm vào nhỏ hơn node gốc sẽ được xếp vào cây con trái).

+ Cây côn phải (với những phần tử thêm vào sẽ được so sánh với node gốc->nếu giá trị node thêm vào lớn hơn node gốc sẽ được xếp vào cây con phải).

\*Note:Trong cây AVL không tồn tại phần tử bằng nhau

### + Mục đích sử dụng:

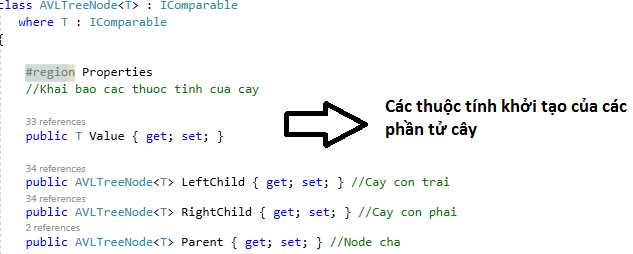
Giúp việc tìm kiếm phần tử dễ dàng hơn so với việc sử dụng mảng 1 chiều và danh sách liên kết đơn với thời gian làm việc là **O(log n)** với mội trường hợp dù là xấu nhất do việc có thể giảm bớt số phần tử khi duyệt cây (việc duyệt chỉ xảy ra với cây con trái hoặc phải->loại bỏ bớt trường hợp không cần thiết phải duyệt ngày từ đầu) và luôn đảm bảo tính đúng đắn, ổn định do rang buộc và tính tự cân bằng của cây.

## II/Phần đồ án

### 1/ Phần code cây AVL:

#### \* Phần khai báo thuộc tính và khởi tạo cây con cơ bản:

- Các thành phần cuả cây con được khởi tạo :cây con trái, cây con phải và node cha dưới dạng con trỏ

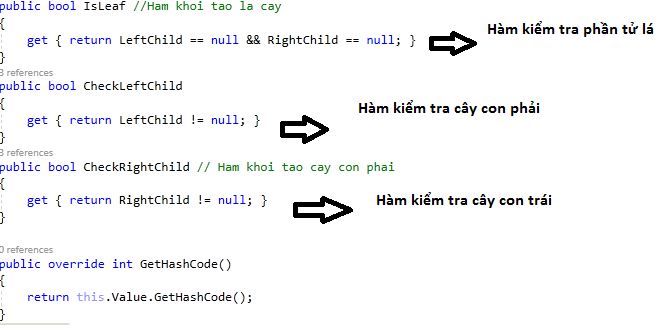


* Các hàm tương tác với phần tử cây con:

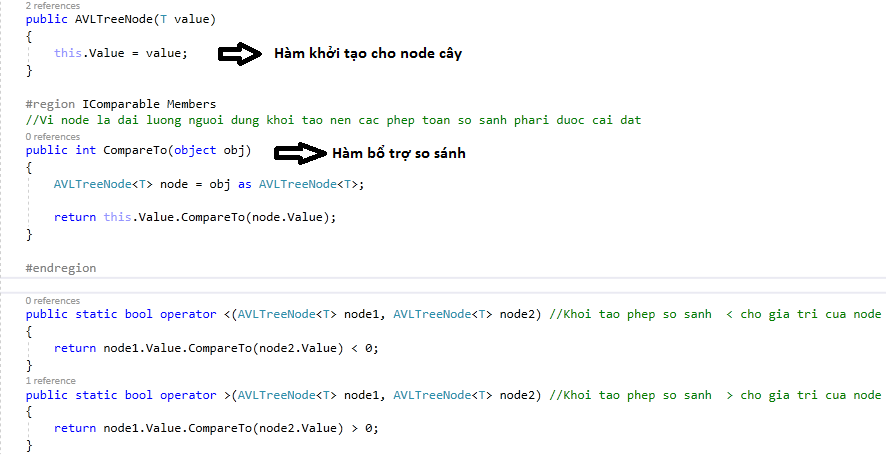
+ IsLeaf: trả về cây con trái và phải của node hiện hành là Null khi nó là phần tử lá

+ CheckLeftChild: trả về cây con trái là khác Null (khởi tạo sẵn phần tử để chứa giá trị)

+ CheckRightChild: trả về cây con phải là khác Null (khởi tạo sẵn phần tử để chứa giá trị)

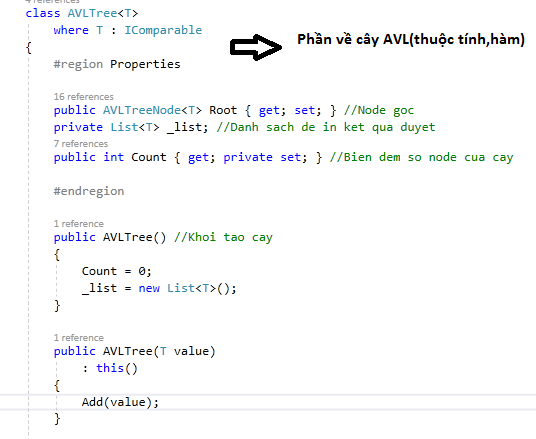


* Các hàm bổ trợ (phụ không nhất thiết phải có): là các hàm so sánh giá trị do người dùng nhập vào với giá trị (value) của node hiện hành trong quá trình thực hiện các thuật toán để tránh sự lặp lại của code.
* Các hàm bổ trợ này chỉ giúp thuận tiện trong quá trình viết không nhất thiết phải có.



#### • Phần khai báo thuộc tính,hàm xử lí và khởi tạo cây AVL cơ bản:

- Các thành phần của cây AVL được khởi tạo:node gốc, cây (quản lí các node dưới dạng list trong C#), biến count (quản lí các biến số về chiều cao của cây trong quá trình tương tác với cây)



* Phần hằng số cân bằng của cây:

Vì đây là phần quan trọng liên quan tới hầu hết các quá trình cân bằng cây trong các thao tác hàm nên được trình bày trước:

* Hệ số cân bằng của cây T là hiệu số giữa các chiều cao của cây con trái và cây con phải của nó.

Đây là điểm tạo nên sự khác biệt giữa cây nhị phân và cây AVL.

#### 

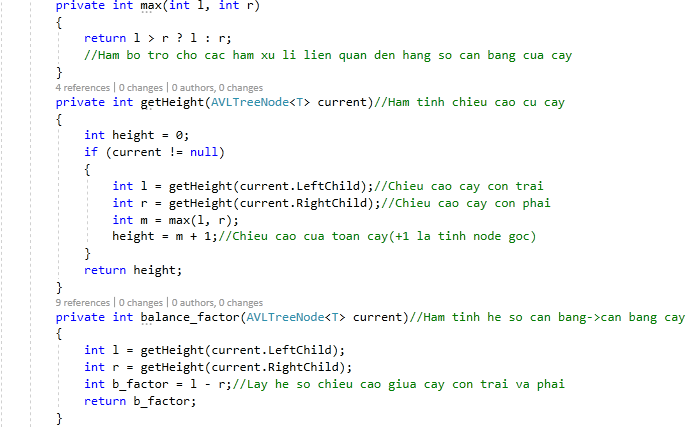
* Cơ chế hoạt động của hằng số cân bằng: ký hiệu hệ số cân bằng của cây con gôc *u* là **balance**(*u*). Hệ số cân bằng của cây T là **balance**(*T*).

**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.

Cây T được gọi là cân bằng AVL nếu tại mỗi nút u của nó hệ số cân bằng có trị số tuyệt đối không vượt quá 1. Điều đó cũng có nghĩa là với mọi nút u của T, **balance**(u) chỉ nhận một trong ba giá trị -1, 0, 1. Khi đó cây T cũng được gọi là cây AVL. Nếu cây con gốc tại đỉnh u là cân bằng AVL, ta cũng gọi đỉnh u là cân bằng AVL. Như vậy các lá là cân bằng AVL, cây chỉ gồm một nút gốc là cây AVL, cây chỉ gồm 2 nút là cây AVL. Cây gồm 3 nút có thể cân bằng AVL, cũng có thể không.

* Code xử lí về hẳng số cân bằng:



+ Hàm max sẽ xét chiều cao của cây con trái và cây con phải và trả về độ cao lớn hơn (int) để giúp tính toán chiều cao thực của cây

+ Hàm getHeight sẽ trả về chiều cao của cây thông qua việc duyệt đến tận cùng của các cây con(biến l cho cây con trái và biến r duyệt cây con phải)->Trả về chiều con thực của cây +1(node gốc)

+ Hàm balance\_facetor trả về hằng số cân bằng của node hiện hành(int) bằng cách tính độ chênh lệch của 2 cây con trái và phải.

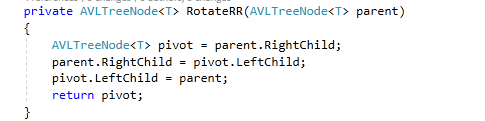
Từ hệ số cân bằng người ta sẽ phát hiện ra các trường hợp mất cân bằng của cây và giải thuật khắc phục:

Khi chèn thêm hoặc xóa đi 1 phần tử v cho nút u cây AVL cân bằng sẽ dẫn sinh các trường hợp:Nếu với mọi đỉnh u của T ta có **balance**(u)= 0 thì T được gọi là cây cân bằng hoàn toàn; do v chỉ có thể được chèn vào đúng một trong hai cây con của **u** nên nhiều nhất là **v** có thể làm tăng chiều cao của một trong hai cây con đó 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.Cụ thể:

* Trước khi chèn cây con gốc u lệch trái và v làm tăng chiều cao của cây con trái.

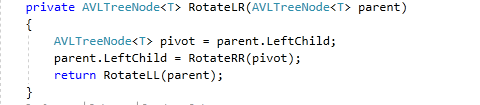
+ Sau khi chèn cây con trái lệch trái (Case LL): cách giải quyết sử dụng kĩ thuật quay phải.Mô tả:



+ Code kĩ thuật quay phải:

+ Sau khi chèn cây con trái lệch phải (Case LR): cách giải quyết sử dụng phép quay trái-phải.Mô tả:

|  |  |
| --- | --- |
| Trạng thái | Hành động |
|  | Một nút đã được chèn vào cây con bên phải của cây con bên trái.Điều này làm nút C trở nên không cân bằng |
|  | Đầu tiên, thực hiện phép quay trái trên cây con bên trái của **C**. Điều này làm cho **A** trở thành cây con bên trái của **B**. |
|  | Bây giờ nút **C** vẫn không cân bằng, đó là do xuất hiện cây con bên trái của cây con bên trái. |
|  | Bây giờ chúng ta sẽ thực hiện kỹ thuật quay phải để làm **B** trở thành nút gốc mới của cây này. Nút **C** bây giờ trở thành cây con bên phải của chính cây con bên trái của nó. |
|  | Cây trở lại trạng thái cân bằng |

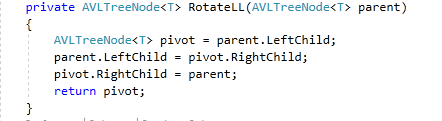
+ Code kĩ thuật quay trái-phải:

* Trước khi chèn cây con gốc u lệch phải và v làm tăng chiều cao của cây con phải.

+ Sau khi chèn cây con phải lệch phải (Case RR): cách giải quyết sử dụng kĩ thuật quay trái:



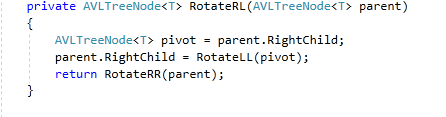
+ Code kĩ thuật quay trái:



+ Sau khi chèn cây con phải lệch trái. (Case RL): cách giải quyết sử dụng kĩ thuật quay phải-trái:

|  |  |
| --- | --- |
| Trạng thái | Hành động |
|  | Một nút đã được chèn vào trong cây con bên trái của cây con bên phải. Điều này làm nút **A** trở nên không cân bằng |
|  | Đầu tiên, chúng ta thực hiện kỹ thuật quay phải với nút **C**, làm cho **C** trở thành cây con bên phải của chính cây con bên trái **B**. Bây giờ, nút **B** trở thành cây con bên phải của nút **A**. |
|  | Bây giờ nút **A** vẫn không cân bằng bởi vì xuất hiện cây con bên phải của cây con bên phải của nó. Do đó cần phải thực hiện một kỹ thuật quay trái. |
|  | Một kỹ thuật quay trái được thực hiện làm cho **B** trở thành nút gốc mới của cây con. Nút **A** trở thành cây con bên trái của cây con **B** bên phải của nó. |
|  | Một kỹ thuật quay trái được thực hiện làm cho **B** trở thành nút gốc mới của cây con. Nút **A** trở thành cây con bên trái của cây con **B** bên phải của nó. |

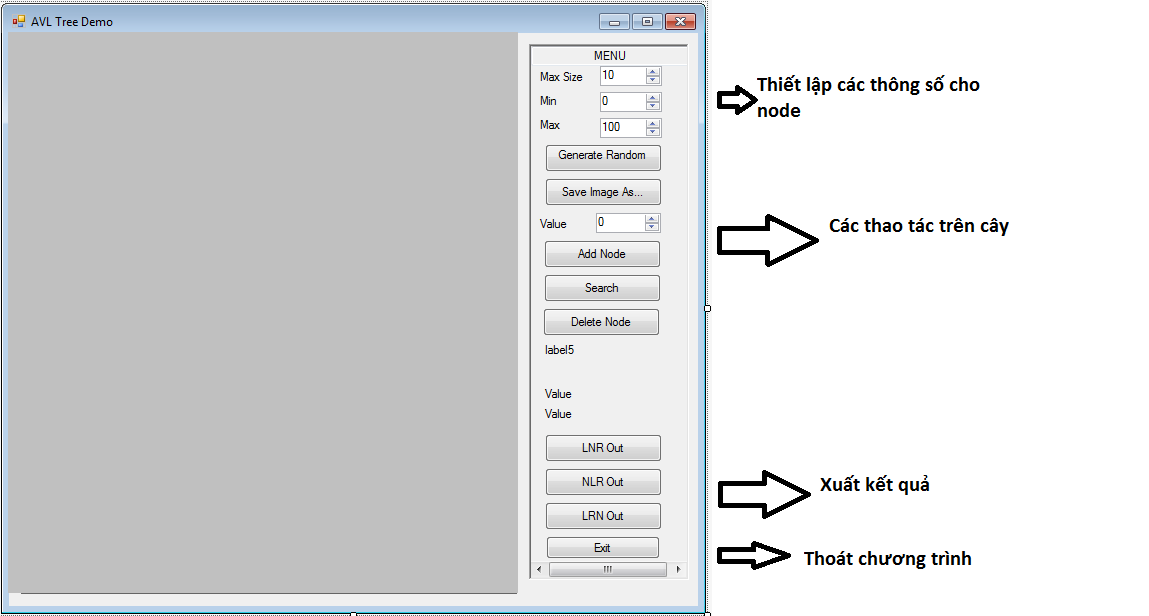
+ Code kĩ thuật quay phải trái:



### 2/ Phần xây dựng giao diện winform:

### + Giao diện người dung:

Xây dựng trên panel:



* Các thiết lập cho node: Max Size(int):người dung nhập để qui định số node tối đa mà cây khởi tạo

+ Min/max: thiết lập phạm vi giá trị (int) cho node, ví dụ như hình Min=0, Max=100 các node tạo thành sẽ có giá trị từ 0->100

+ Generate Random: tạo cây với các giá trị node được khởi tạo random int

+ Save Image As: chụp lại kết quả chạy mà người dung mong muốn

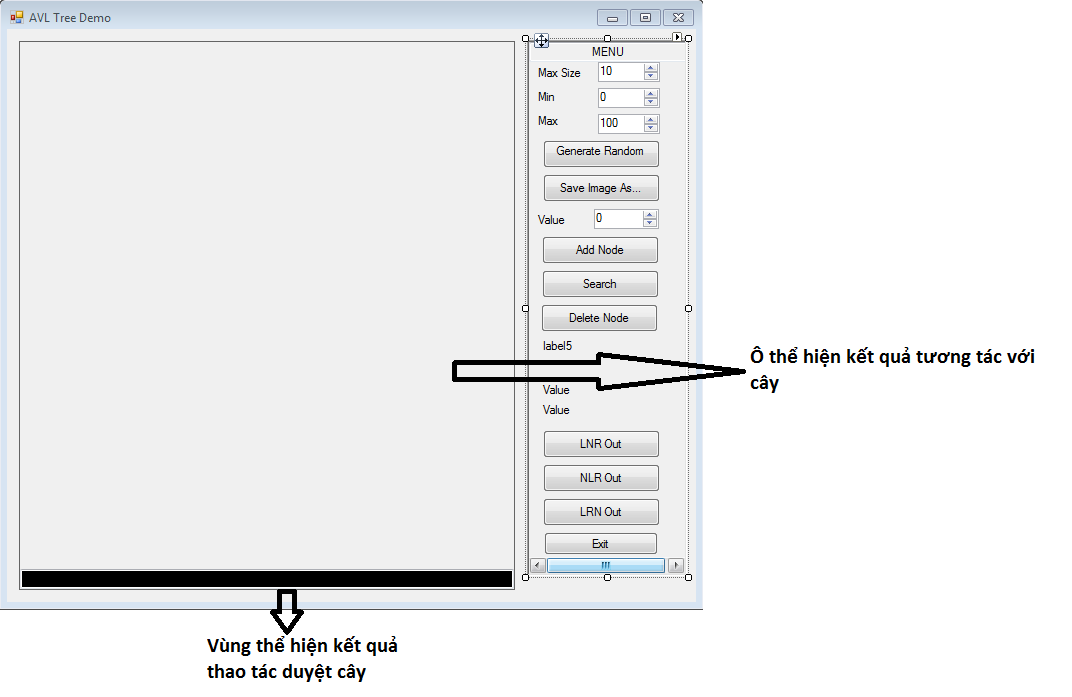
Với các thao tác tiếp theo: cần nhập giá trị mà người dung muốn thực hiện vào label Value và có thể thực hiện các thao tác sau:  
+ Addnode: thêm giá trị Value vào cây,trùng sẽ báo lỗi

+ Deletenode: xóa node có giá trị Value khỏi cây, nếu không tồn tại giá trị Value trên cây sẽ thông báo.

+ Search: tìm node có giá trị Value trên cây, nếu không tồn tại giá trị Value sẽ thông báo.

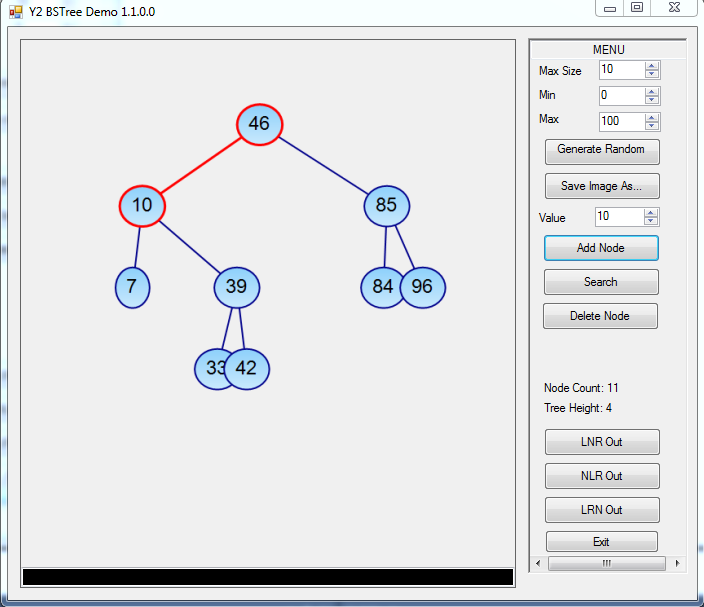
* Các thao tác khác: Exit: thoát khỏi winform hiện thời.

### + Giao diện quan sát các thao tác:

Xây dựng trên 1 Picturebox:

+ Ô thể hiện kết quả tương tác với cây: hiển thị các quá trình vẽ cây và các tương tác add, search, delete node:

Ví dụ: thao tác Add Node 10 vào cây hiện hành



## III/Các tài liệu sử dụng:

Các thao tác vẽ winform: <https://yinyangit.wordpress.com/2011/01/21/algorithm-c%E1%BA%ADp-nh%E1%BA%ADt-thu%E1%BA%ADt-toan-v%E1%BA%BD-cay-nh%E1%BB%8B-phan/>

Kiến thức cơ bản về winform: Wikipedia