CANTHO UNIVERSITY

Cấu trúc dữ liệu

CÂY AVL

(G.M. Adelson-Velsky và E.M. Landis)

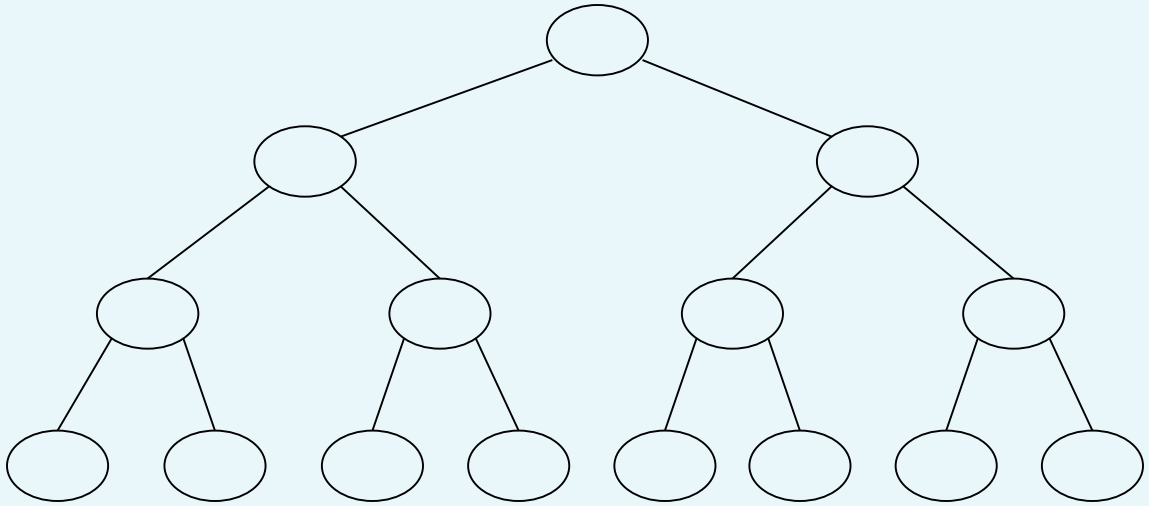
Bộ môn Công Nghệ Phần Mềm

Nội dung

- Kiến thức bổ sung (cây nhị phân)
- Khái niệm về cây tìm kiếm nhị phân cân bằng
- Khái niệm về cây AVL
- Các thuật toán trên cây AVL
- Ý tưởng cài đặt cây bằng con trỏ

CÂY NHỊ PHÂN ĐẦY ĐỦ (full binary tree)

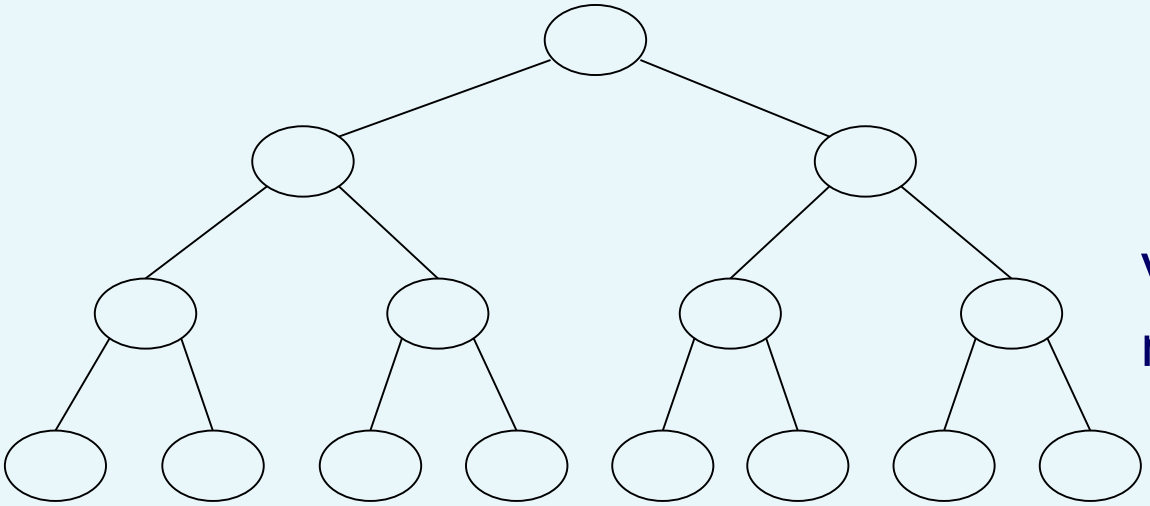
- Một cây nhị phân là "cây nhị phân đầy đủ" nếu và chỉ nếu
  - Mỗi nút không phải lá có chính xác 2 nút con.
  - Tất cả các nút lá có mức bằng nhau.



Ví dụ: một cây nhị phân đầy đủ

CÂY NHỊ PHÂN ĐẦY ĐỦ

- **Câu hỏi về cây nhị phân đầy đủ:**
  - Một cây nhị phân đầy đủ chiều cao h sẽ có bao nhiêu nút lá? 2^h
  - Một cây nhị phân đầy đủ chiều cao h sẽ có tất cả bao nhiêu nút? 2^0+2^1+…+2^h=2^(h + 1) -1



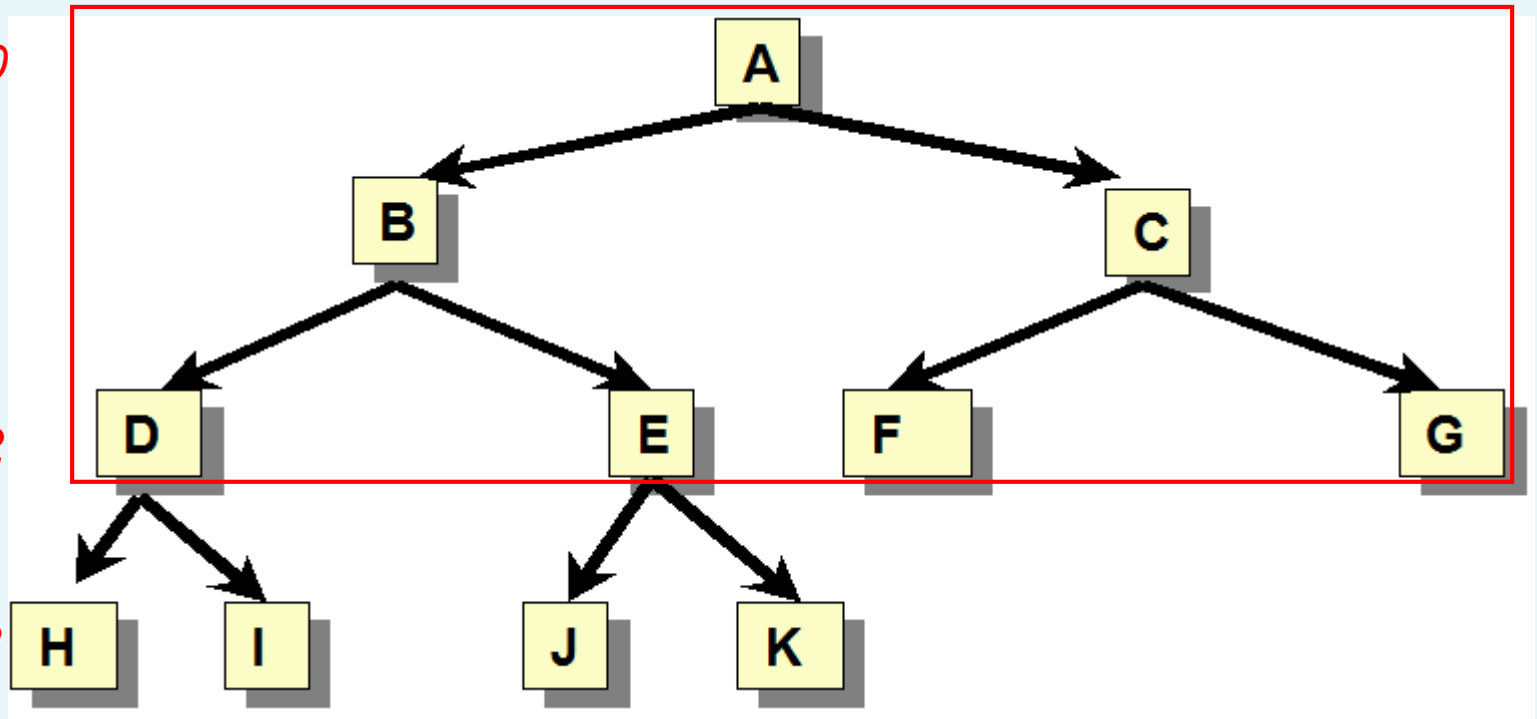
Ví dụ: một cây nhị phân đầy đủ

CÂY NHỊ PHÂN HOÀN CHỈNH (complete binary tree)

- Một cây nhị phân hoàn chỉnh (về chiều cao) thỏa mãn các điều kiện sau:
  - Mức 0 đến h-1 là trình bày một cây nhị phân đầy đủ chiều cao h-1.
  - Một hoặc nhiều nút ở mức h-1 có thể có 0, hoặc 1 nút con.
  - Nếu j, k là các nút ở mức h-1, khi đó j có nhiều nút con hơn k nếu và chỉ nếu j ở bên trái của k.

CÂY NHỊ PHÂN HOÀN CHỈNH

- Ví dụ



Một cây nhị phân hoàn chỉnh

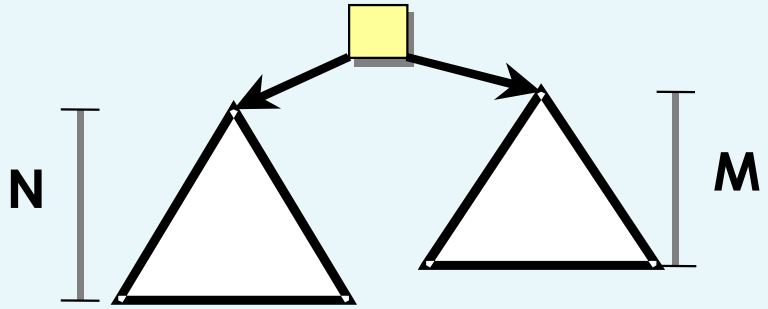
CÂY NHỊ PHÂN HOÀN CHỈNH

- Cho một tập hợp n nút, một cây nhị phân hoàn chỉnh của những nút này cung cấp số nút lá nhiều nhất - với chiều cao trung bình của mỗi nút là nhỏ nhất.
- Cây hoàn chỉnh n nút phải chứa ít nhất một nút có chiều cao là $\lfloor \log n \rfloor$.

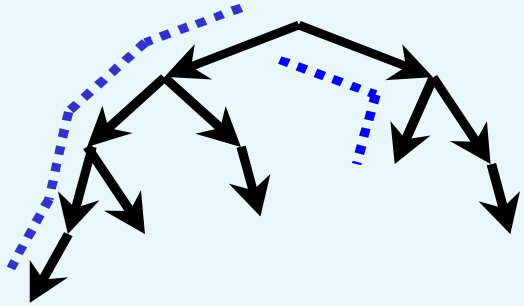
CÂY NHỊ PHÂN CÂN BẰNG VỀ CHIỀU CAO (Height-balanced Binary Tree)

- Một cây nhị phân cân bằng về chiều cao là một cây nhị phân như sau:
  - Chiều cao của cây con trái và phải của bất kỳ nút nào khác nhau không quá một đơn vị.
  - Chú ý: mỗi cây nhị phân hoàn chỉnh là một cây cân bằng về chiều cao.

CÂY NHỊ PHÂN CÂN BẰNG VỀ CHIỀU CAO



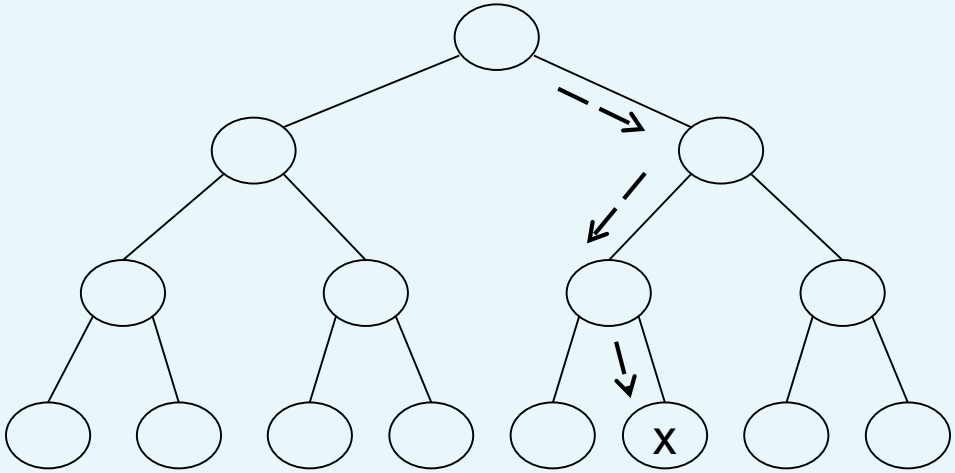
N-M<=1



Cân bằng về chiều cao là một thuộc tính cục bộ

CÂY NHỊ PHÂN CÂN BẰNG VỀ CHIỀU CAO

- Cây nhị phân cân bằng về chiều cao là cây "cân bằng".
- Thời gian tìm kiếm một nút trên cây N nút là O(logN).



- Cây có n nút.
- Thời gian tìm x trong trường hợp xấu nhất khi x là lá
  - Ta phải đi qua đường đi độ dài h để tìm x (h là chiều cao cây)
  - 2^(h + 1) -1 = n

    => h = log(n+1) -1

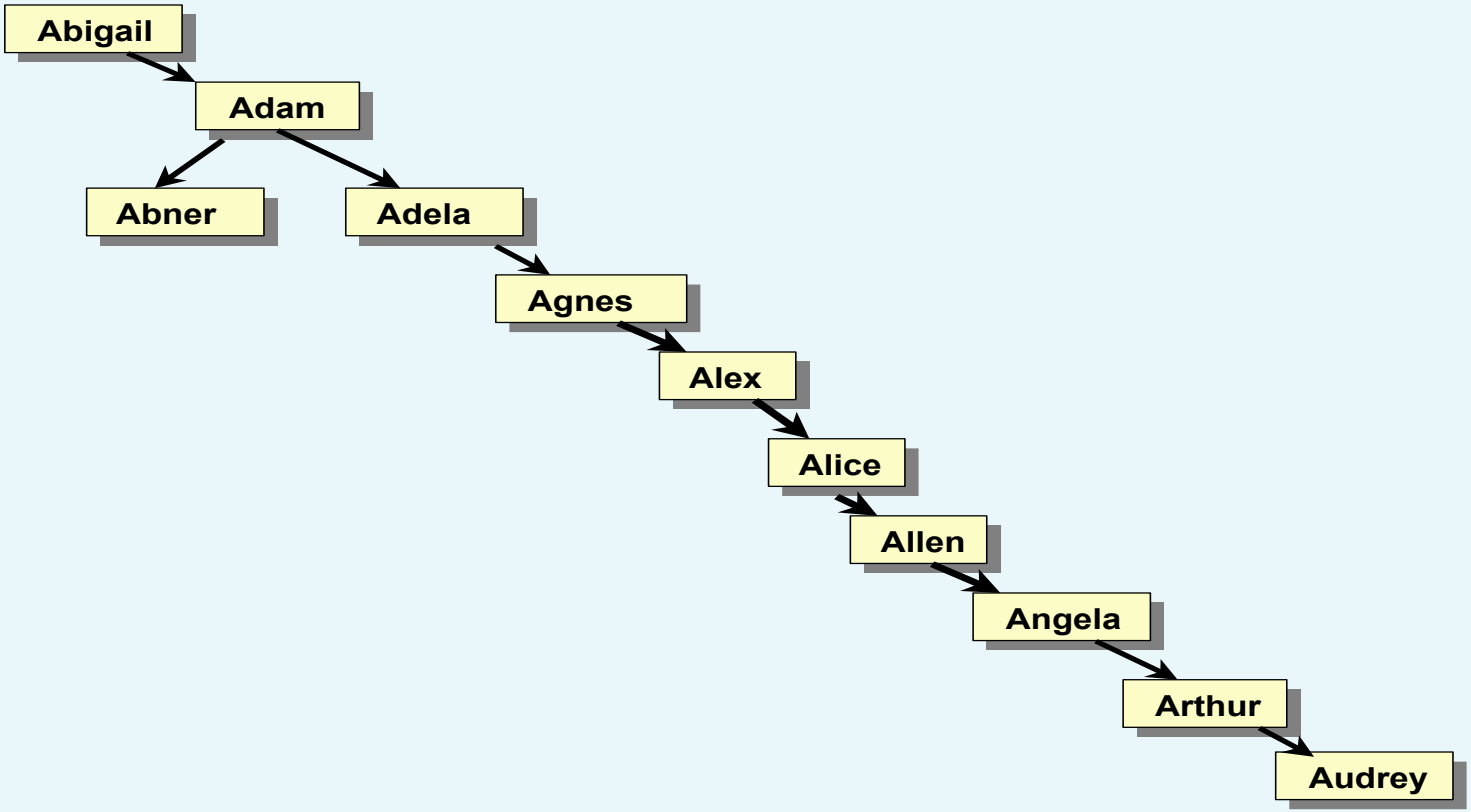
    => O(log(n+1) -1) ≈ O(log(n))

Nội dung

- Kiến thức bổ sung (cây nhị phân)
- Khái niệm về cây tìm kiếm nhị phân cân bằng
- Khái niệm về cây AVL
- Các thuật toán trên cây AVL
- Ý tưởng cài đặt cây bằng con trỏ

CÂY BST KHÔNG CÂN BẰNG

- Bên dưới là một cây BST "không cân bằng"



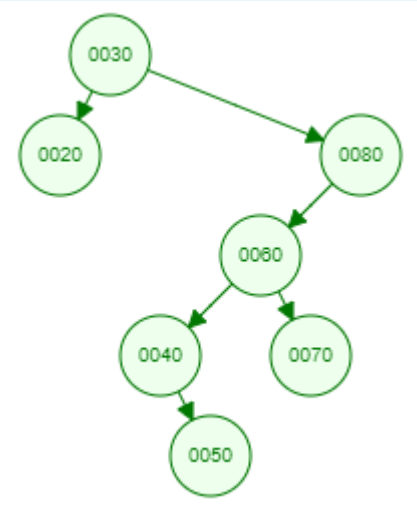
CÂY BST KHÔNG CÂN BẰNG

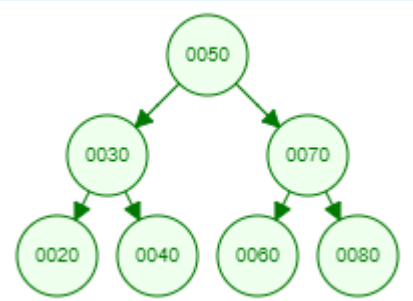
- Tìm kiếm nút có khóa "Audrey" là trường hợp xấu nhất
  - Ta phải duyệt qua hầu như toàn bộ n nút của cây.
  - Thời gian tìm kiếm: O(n).

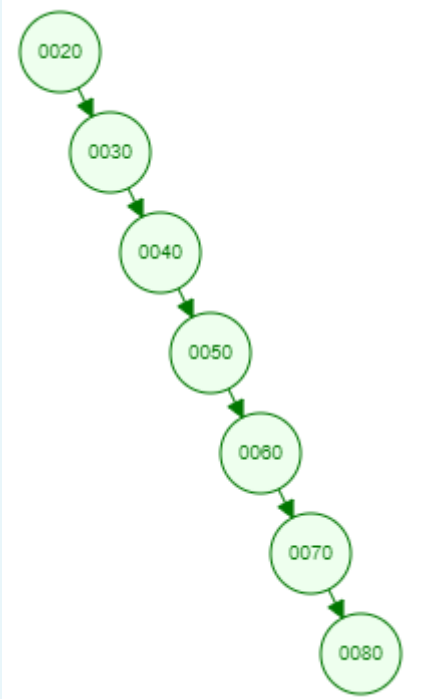
Khái niệm về cây BST cân bằng

- Cây BST được dựng từ các khóa:

  20, 30, 40, 50, 60, 70, 80







- Nhận xét: độ phức tạp trong các giải thuật trên cây tìm kiếm nhị phân trường hợp xấu nhất là O(n) và trung bình O(logn).

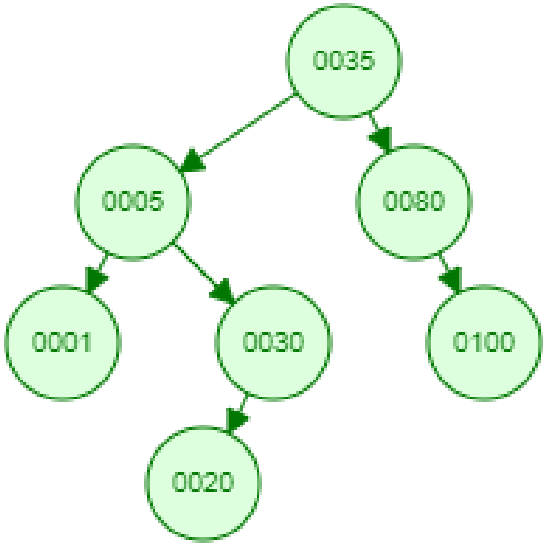
Khái niệm về cây BST cân bằng

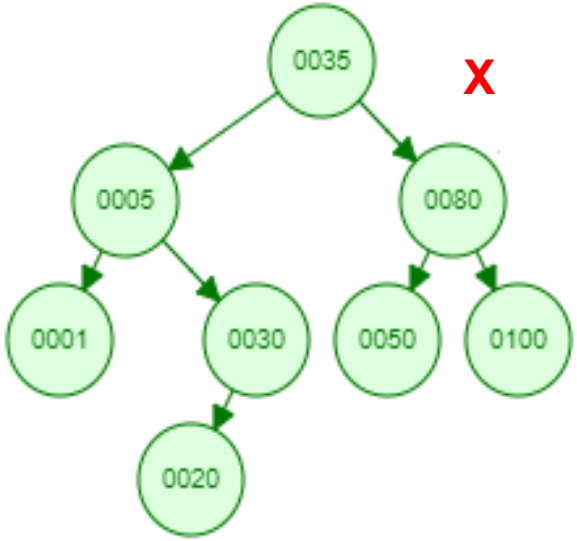
- Quá trình tìm kiếm khóa, thêm nút, xóa nút trên cây BST là *quá trình di chuyển từ nút gốc ra nút lá*. ➔ Cây càng cao thì giải thuật càng kém hiệu quả.

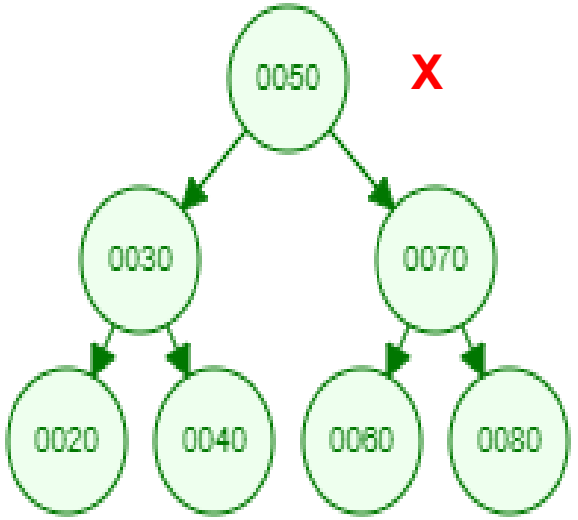
- Do vậy rất cần thiết phải xây dựng cây BST mà trong đó chiều cao của cây càng nhỏ càng tốt để cho các giải thuật được hiệu quả nhất. Cây ở dạng này được gọi là cây cân bằng.

Cây BST cân bằng hoàn toàn

- Cây BST cân bằng hoàn toàn là cây BST mà tại mỗi nút có tổng số nút của cây con trái và con phải lệch nhau **không quá một**.
- Cây nào là cây cân bằng hoàn toàn?

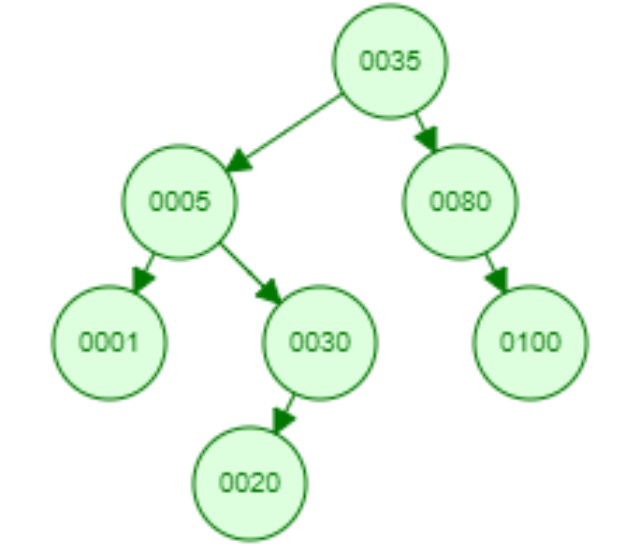


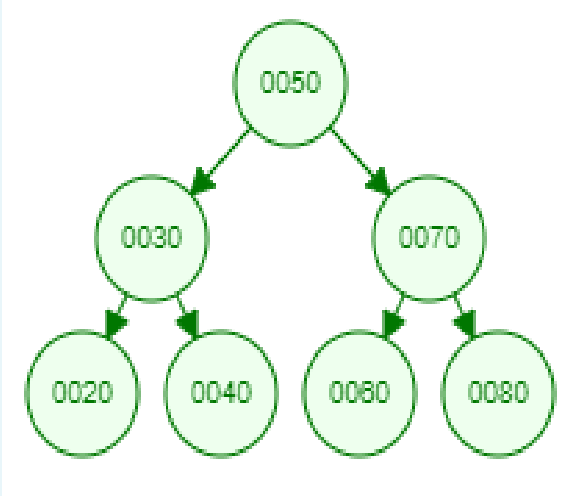




Cây BST cân bằng tương đối (cân bằng về chiều cao)

- Cây BST cân bằng **về chiều cao** là cây BST mà trong đó mỗi nút đều có chiều cao con trái và con phải lệch nhau tối đa là 1.
- Cây cân bằng về chiều cao còn gọi là cân bằng **tương đối**.
- Ví dụ:





Nội dung

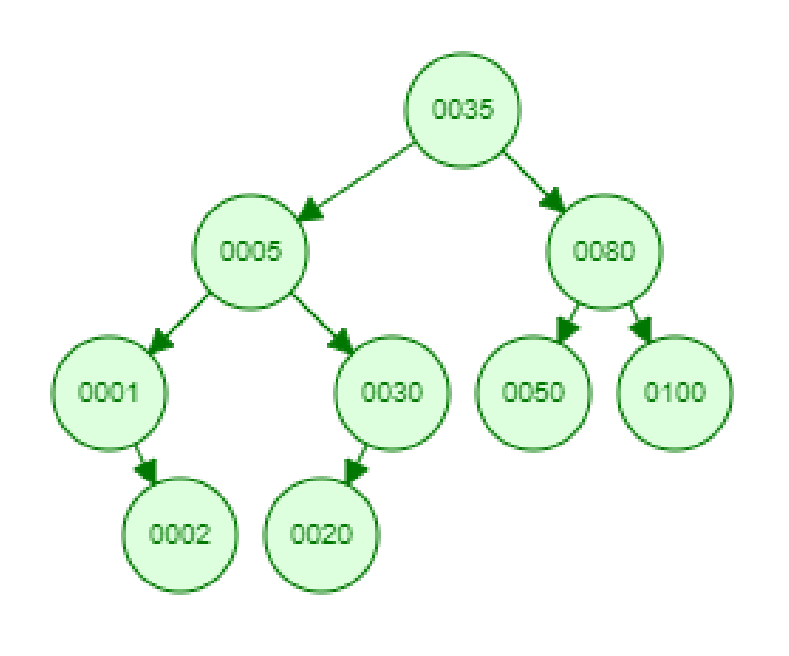
- Kiến thức bổ sung (cây nhị phân)
- Khái niệm về cây tìm kiếm nhị phân cân bằng
- Khái niệm về cây AVL
- Các thuật toán trên cây AVL
- Ý tưởng cài đặt cây bằng con trỏ

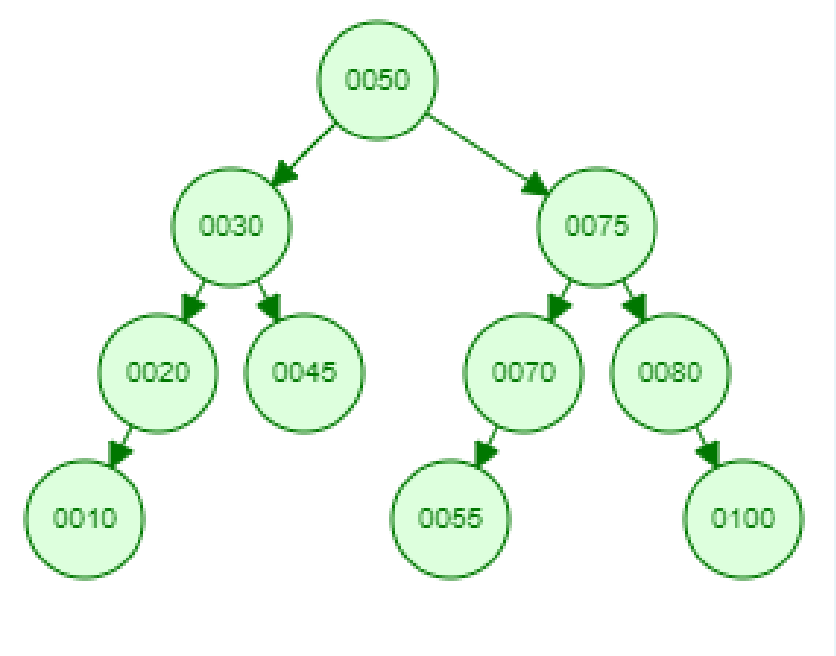
Khái niệm về cây AVL

- Cây AVL được gọi theo tên của hai người đề xuất chúng, G.M. Adelson-Velsky và E.M. Landis, được công bố trong bài báo của họ vào năm 1962: "An algorithm for the organization of information." (Một thuật toán về tổ chức thông tin)
- Cây AVL là cây BST mà chiều cao của hai cây con của mọi nút chênh lệch tối đa là 1.

Khái niệm về cây AVL

- Ví dụ:





Nội dung

- Kiến thức bổ sung (cây nhị phân)
- Khái niệm về cây tìm kiếm nhị phân cân bằng
- Khái niệm về cây AVL
- Các thuật toán trên cây AVL
- Ý tưởng cài đặt cây bằng con trỏ

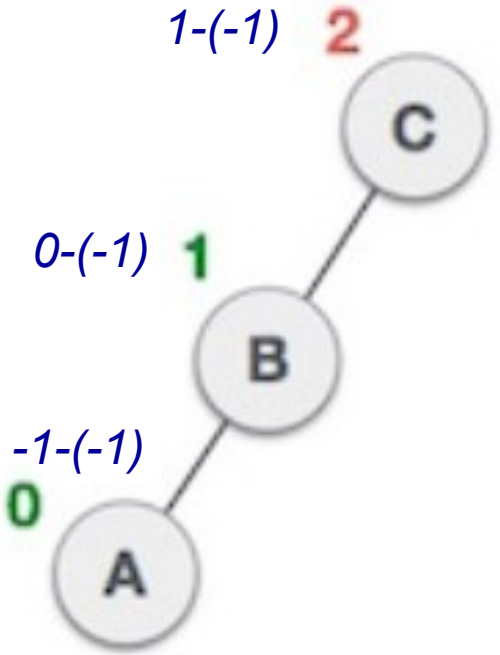
Các thao tác trên cây AVL

- Cây AVL là cây tìm kiếm nhị phân nên *các thao tác cơ bản trên cây AVL tương tự như các thao các cơ bản trên cây BST*.
- Trong đó, thao tác thêm và xóa nút trên cây có thể làm cây mất cân bằng nên *phải có thao tác cân bằng lại cây* sau khi thực hiện thêm hoặc xóa nút.

Các trường hợp cây mất cân bằng

- Hệ số cân bằng

BalanceFactor = height(left-subtree)
- height(right-subtree)

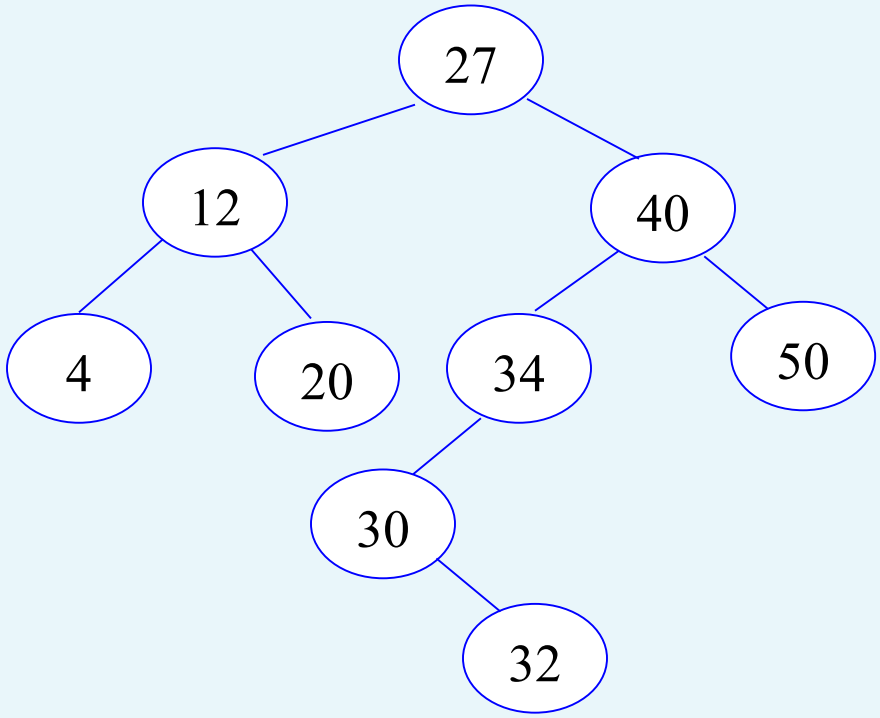



```
int getHeight(Tree T) {
    if (T == NULL)
        return -1;
    else
        return 1+max(getHeight(T->Left),
                     getHeight(T->Right));
}
```

Các trường hợp cây mất cân bằng

- Hệ số cân bằng

BalanceFactor = height(left-subtree)
- height(right-subtree)

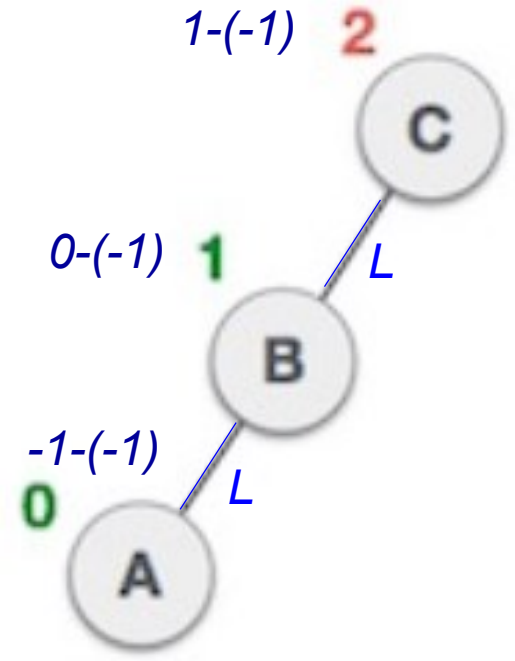


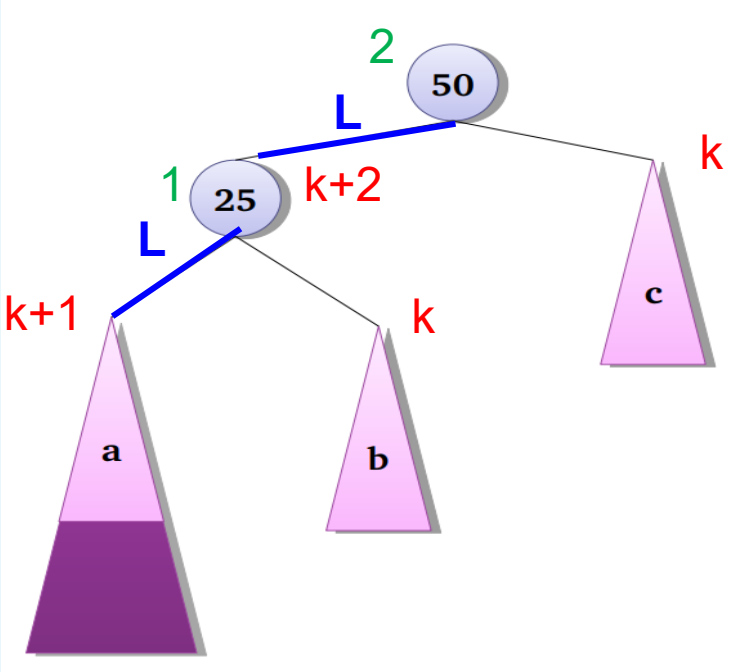
Các trường hợp cây mất cân bằng

- Trường hợp 1: Cây mất cân bằng bên trái của con trái (L-L)
- Trường hợp 2: Cây mất cân bằng bên phải của con phải (R-R)
- Trường hợp 3: Cây mất cân bằng bên phải của con trái (R-L)
- Trường hợp 4: Cây mất cân bằng bên trái của con phải (L-R)

Các trường hợp cây mất cân bằng

- Trường hợp 1: cây mất cân bằng bên trái của con trái (L-L)



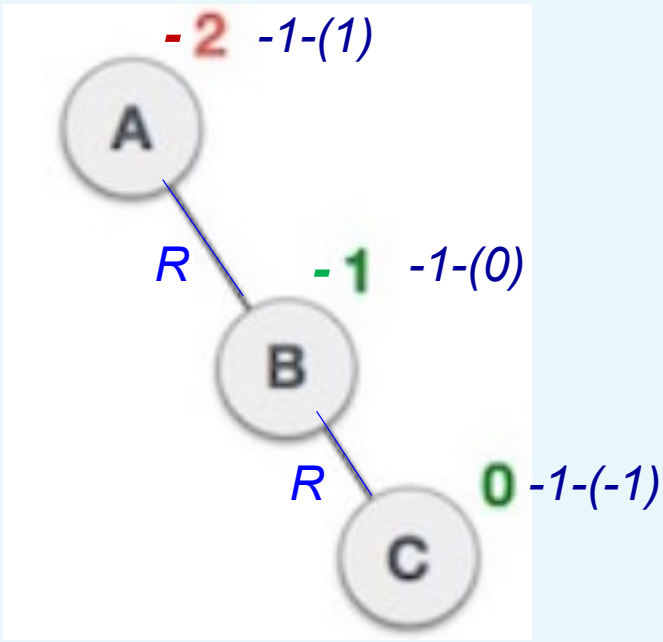


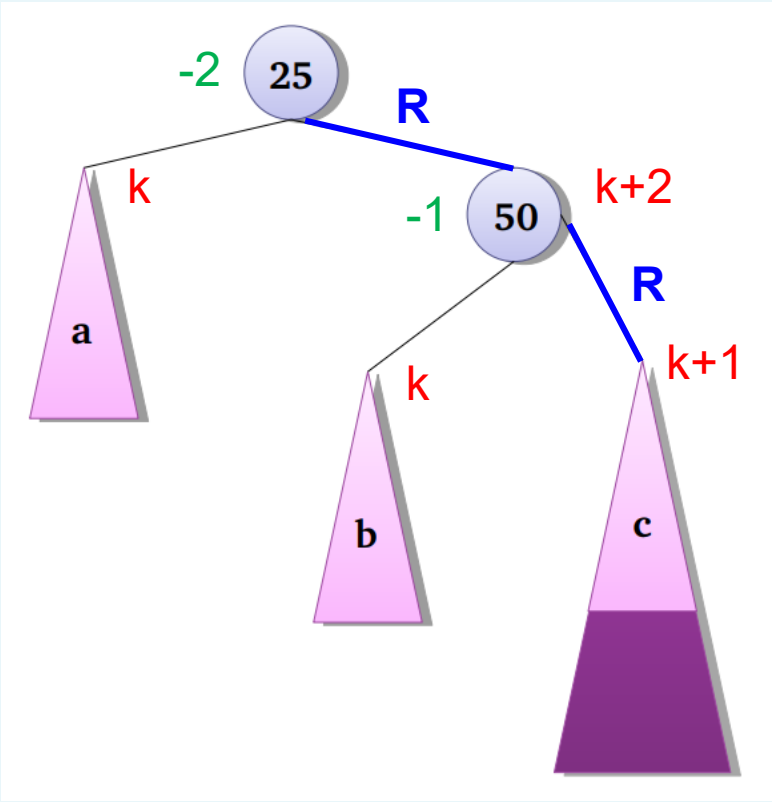
BalanceFactor =height(left-subtree)
- height(right-subtree)

k, k+1, k+2*: chiều cao các cây con*

Các trường hợp cây mất cân bằng

- Trường hợp 2: cây mất cân bằng bên phải của con phải (R-R)

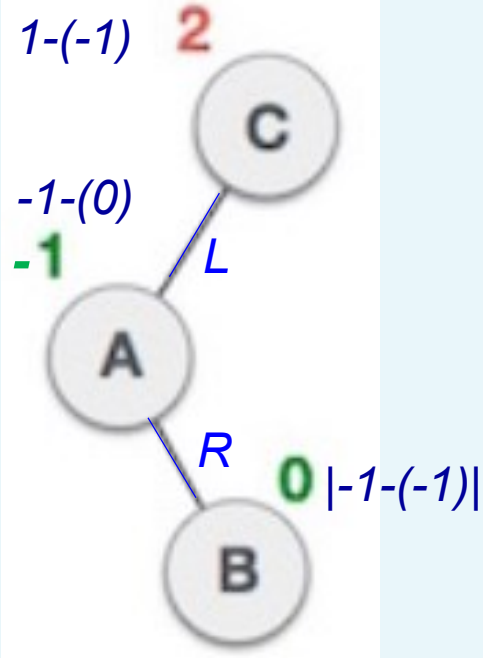


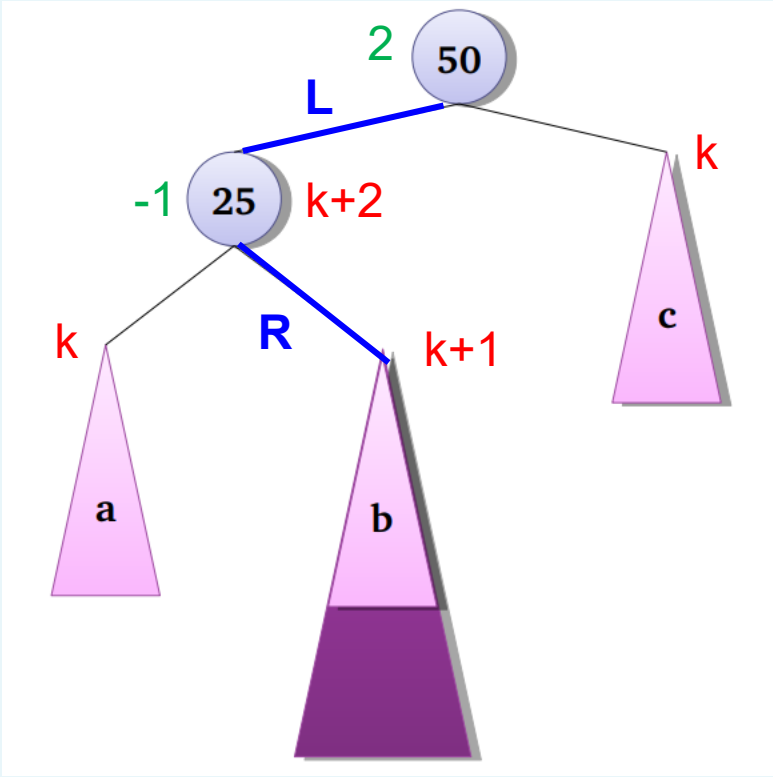


***k, k+1, k+2**: chiều cao các cây con*

Các trường hợp cây mất cân bằng

- Trường hợp 3: cây mất cân bằng bên phải của con trái (R-L)

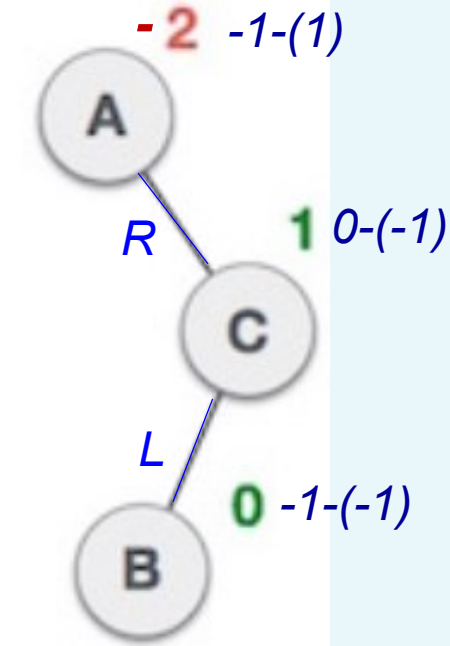


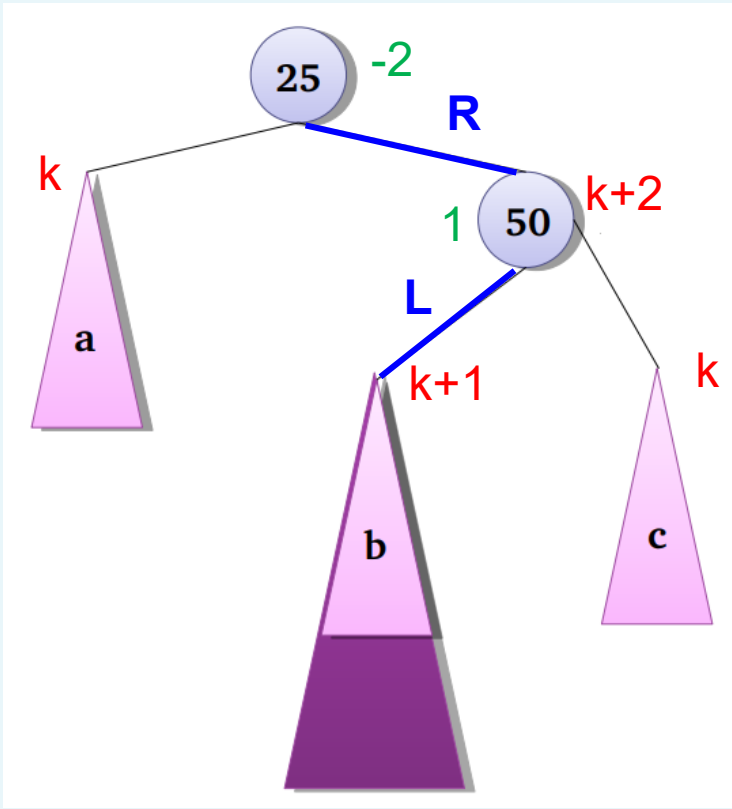


***k, k+1, k+2**: chiều cao các cây con*

Các trường hợp cây mất cân bằng

- Trường hợp 4: cây mất cân bằng bên trái của con phải (L-R)





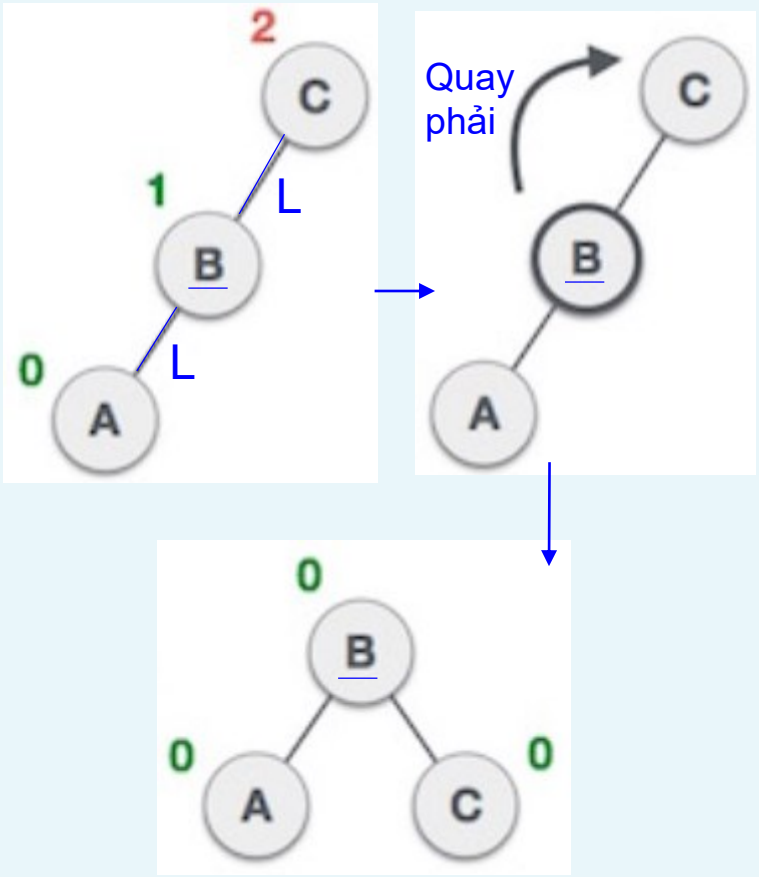
***k, k+1, k+2**: chiều cao các cây con*

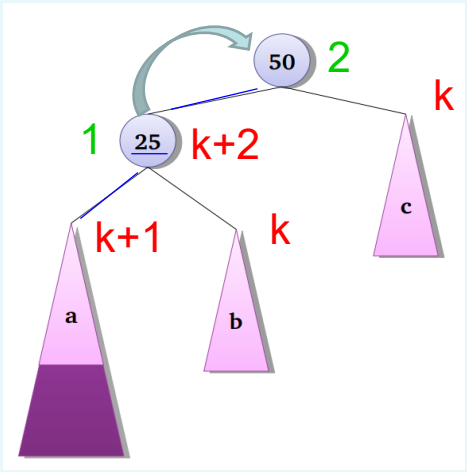
Các quy tắc xử lý khi cây mất cân bằng

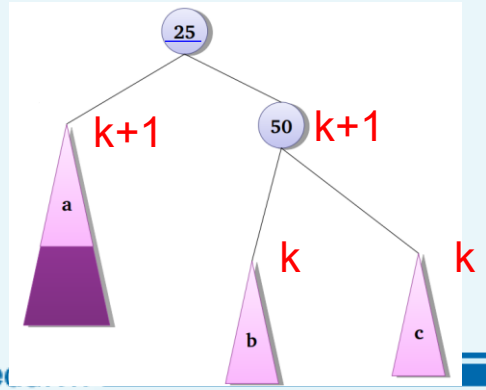
- Trường hợp 1 (L-L): thực hiện quay đơn qua phải (right rotate)
- Trường hợp 2 (R-R): thực hiện quay đơn qua trái (left rotate)
- Trường hợp 3 (R-L): ta thực hiện quay kép trái-phải (left-right rotate)
- Trường hợp 4 (L-R): ta thực hiện quay kép phải-trái (right-left rotate)

Các quy tắc xử lý khi cây mất cân bằng

- Trường hợp 1 (L-L): ta thực hiện quay đơn qua phải (right rotate) như sau:

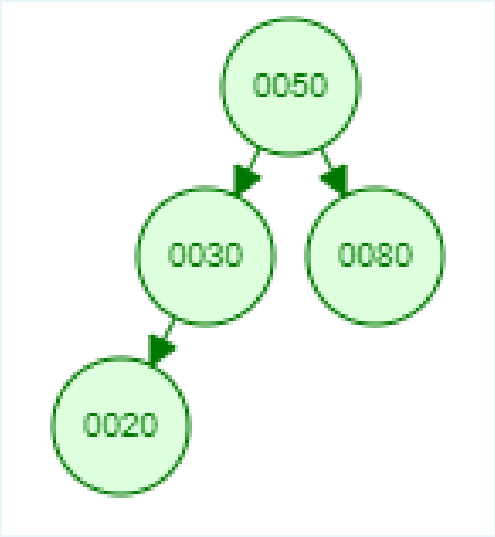




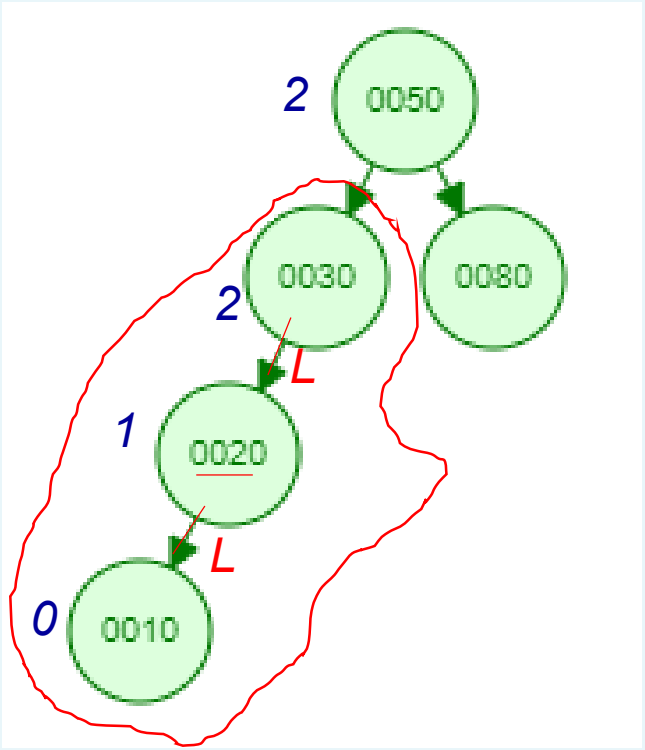


Ví dụ

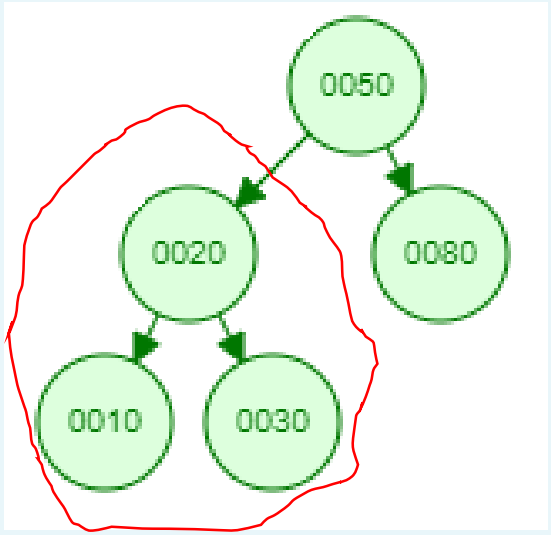
- Thêm nút 10



Mất cân bằng bên trái của con trái

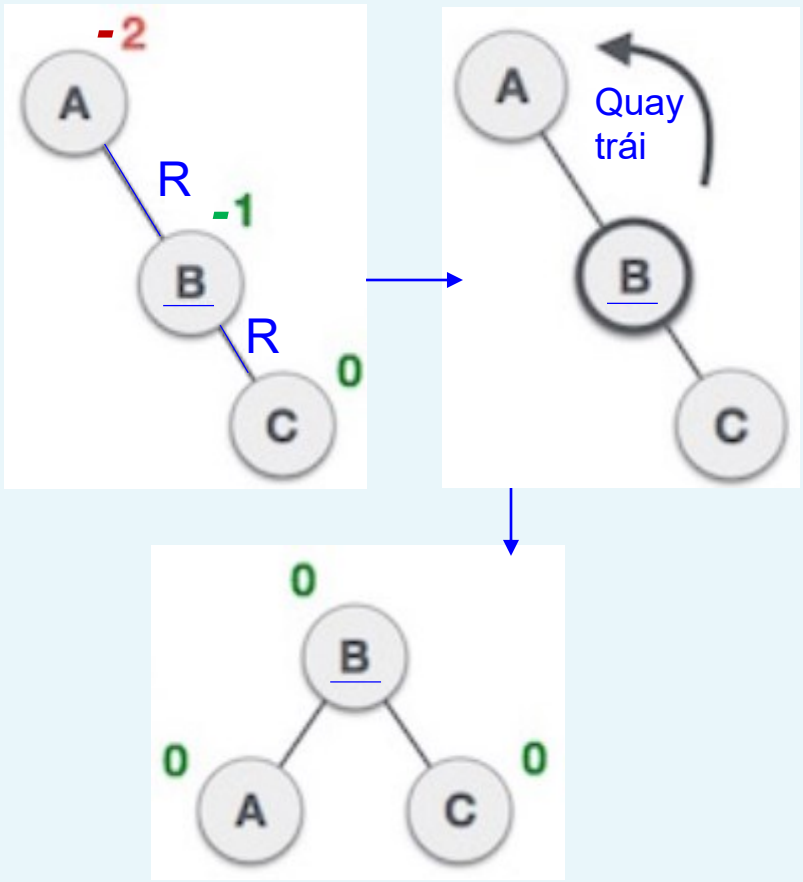


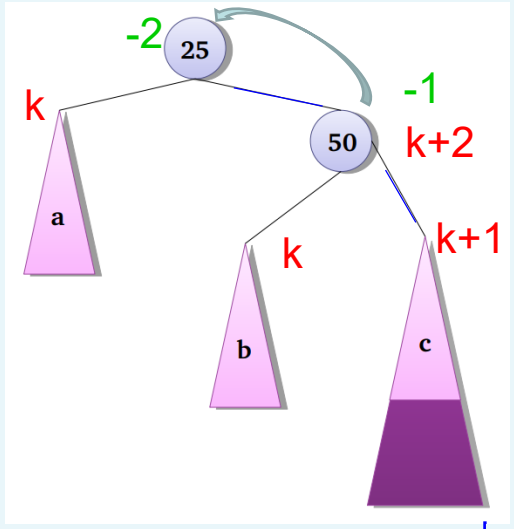
Quay qua phải

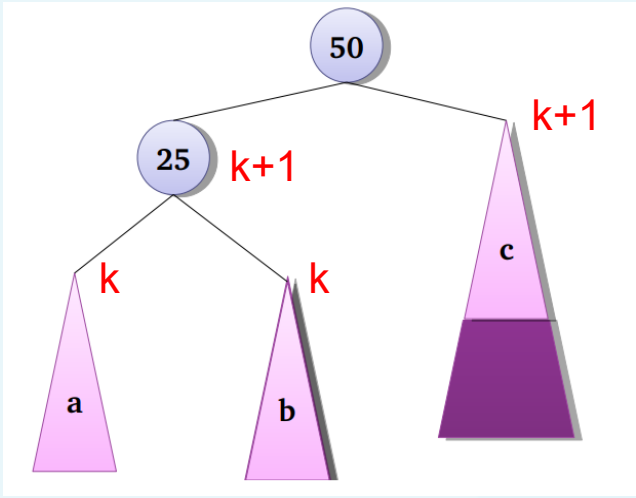


Các quy tắc xử lý khi cây mất cân bằng

- Trường hợp 2 (R-R): ta thực hiện quay đơn qua trái (left rotate) như sau:

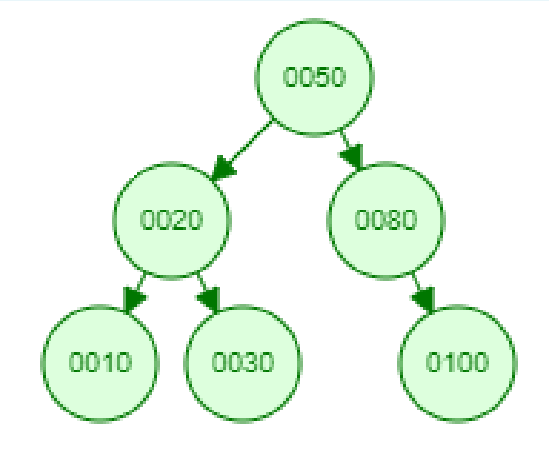




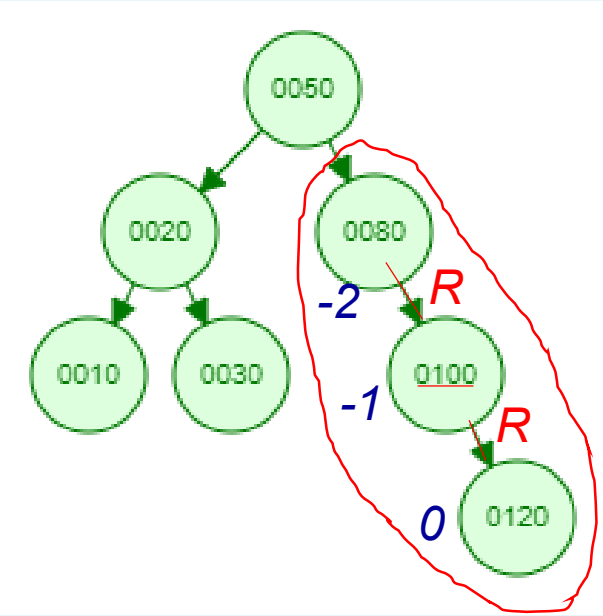


Ví dụ

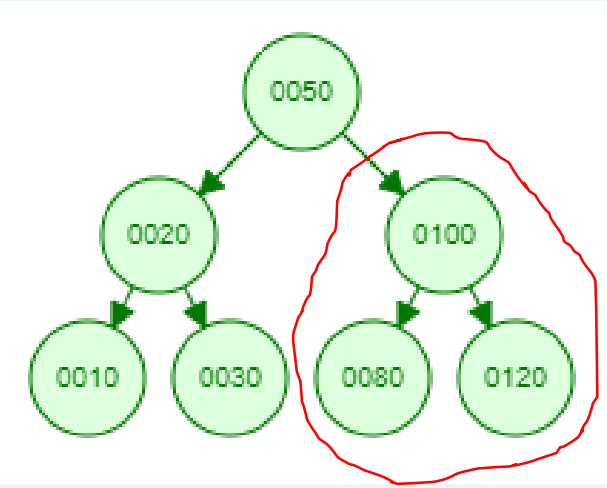
- Thêm nút 120



Mất cân bằng bên phải của con phải

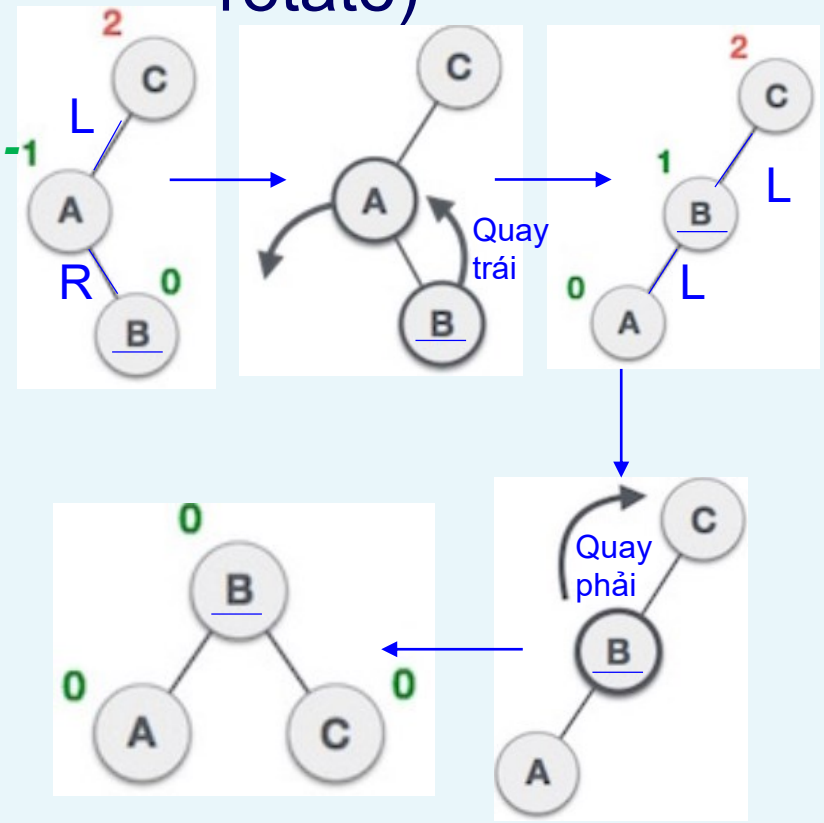


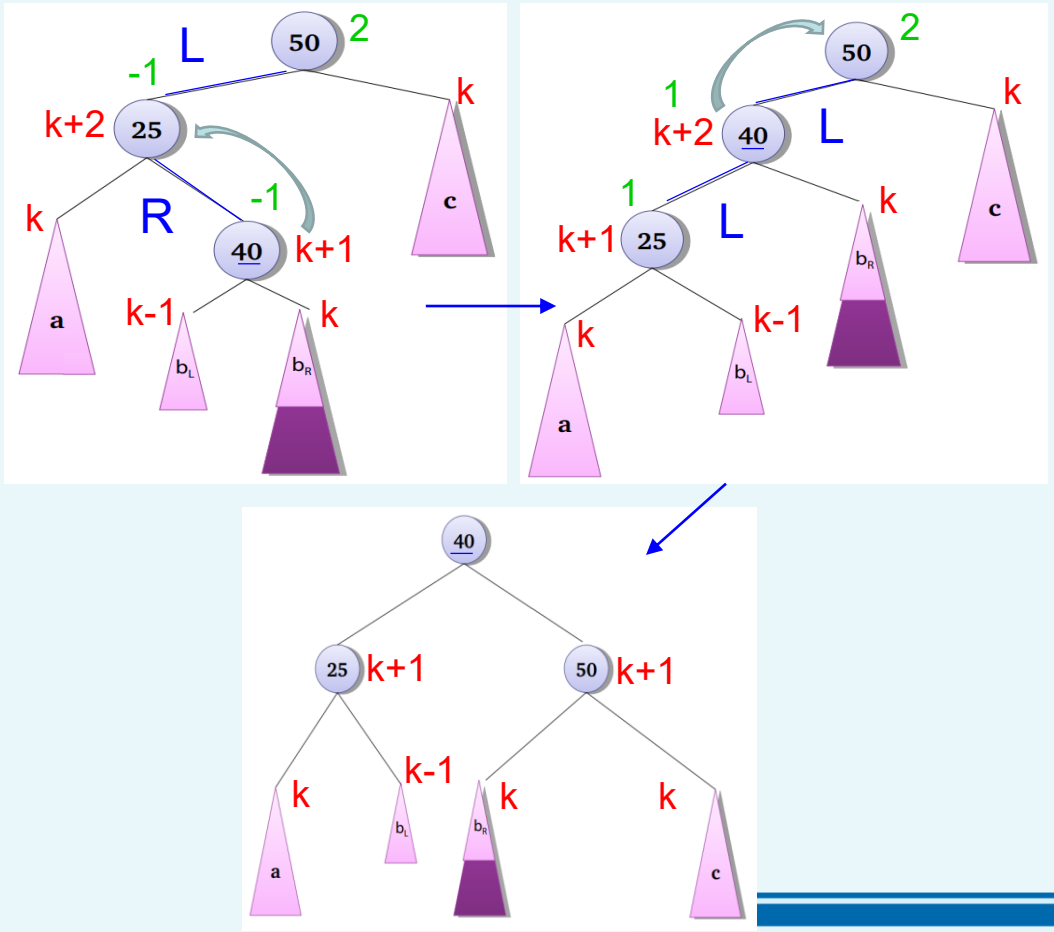
Quay qua trái



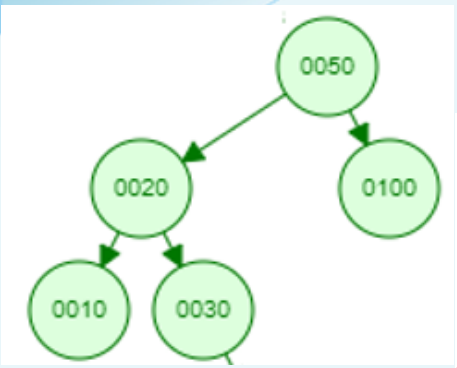
Các quy tắc xử lý khi cây mất cân bằng

- Trường hợp 3 (R-L): cây mất cân bằng bên phải của con trái ta thực hiện quay kép trái-phải (left-right rotate)

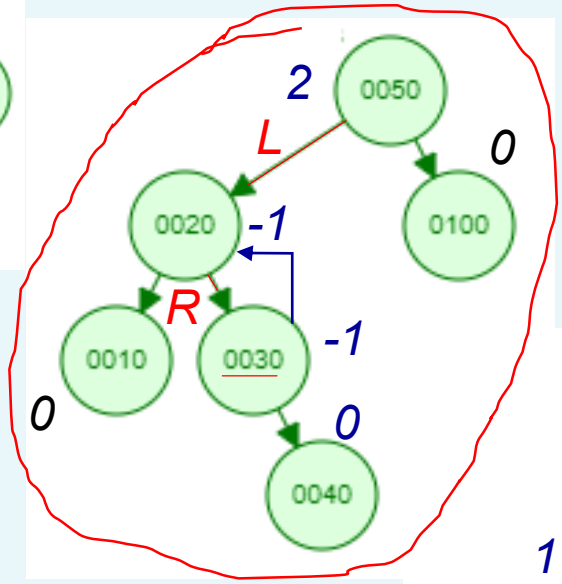




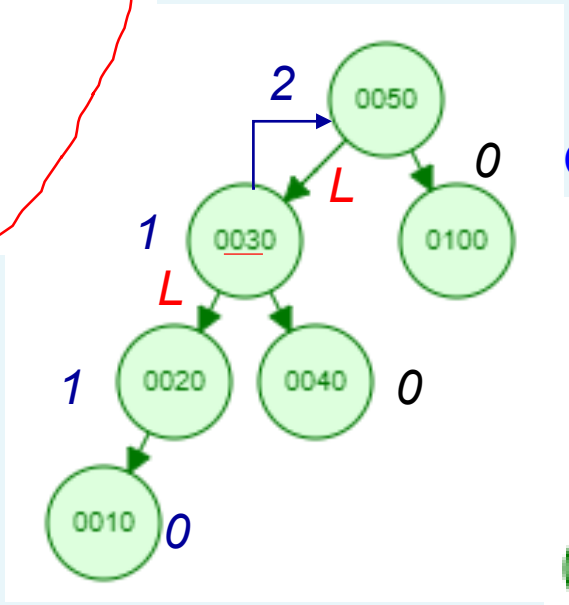
Ví dụ: Thêm nút 40 vào cây



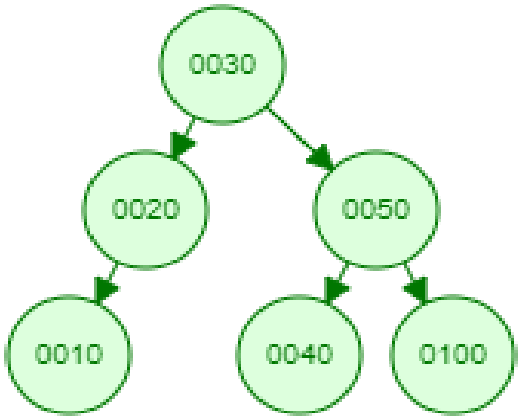
Mất cân bằng bên phải của con trái



Quay qua trái

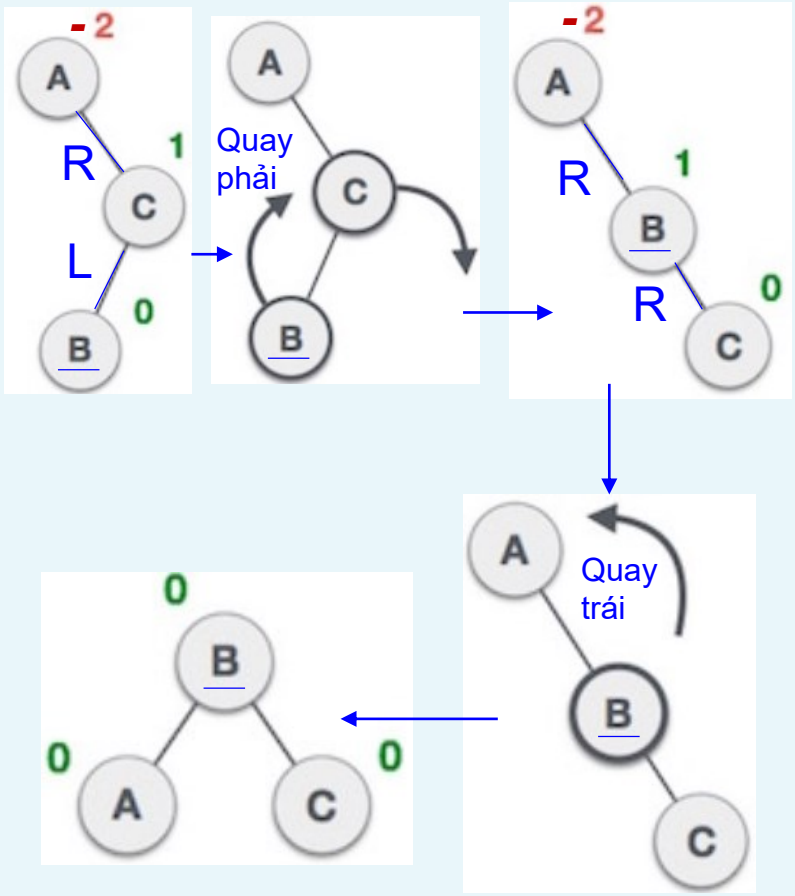


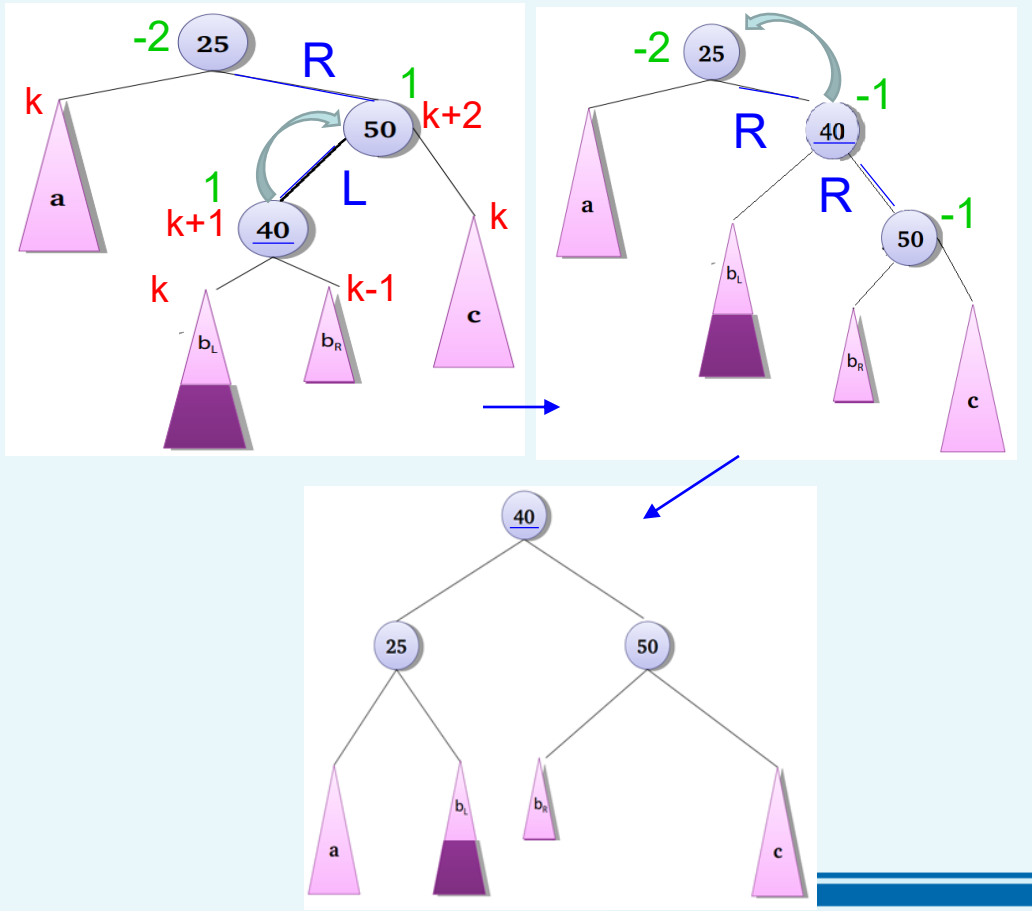
Quay qua phải



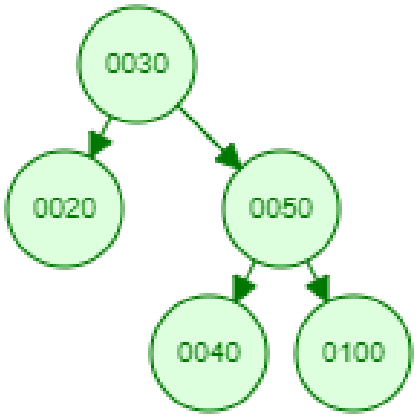
Các quy tắc xử lý khi cây mất cân bằng

- Trường hợp 4 (L-R): cây mất cân bằng bên trái của con phải. Trường hợp này ta thực hiện quay kép phải-trái

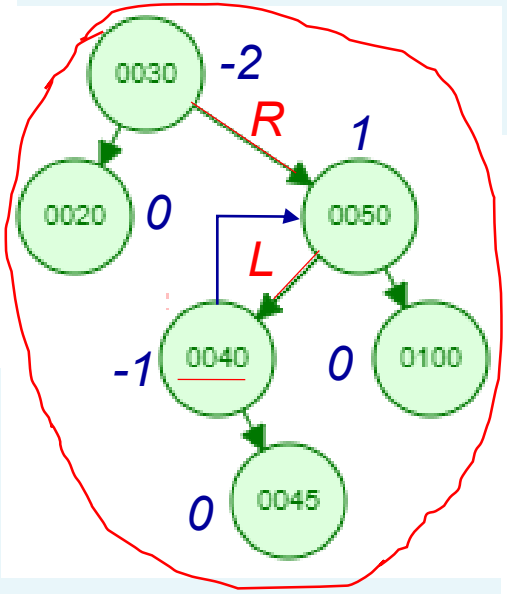




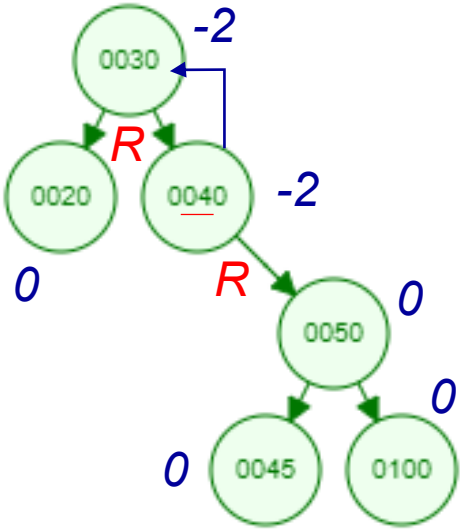
Ví dụ: Thêm nút 45



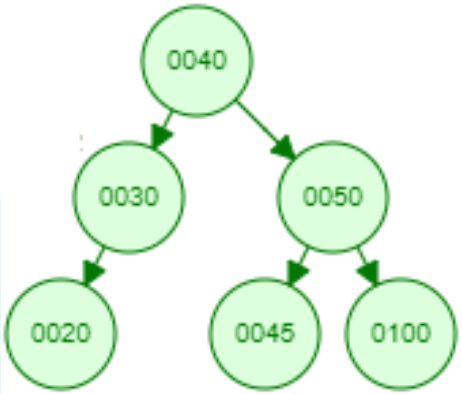
Mất cân bằng bên trái của con phải



Quay qua phải

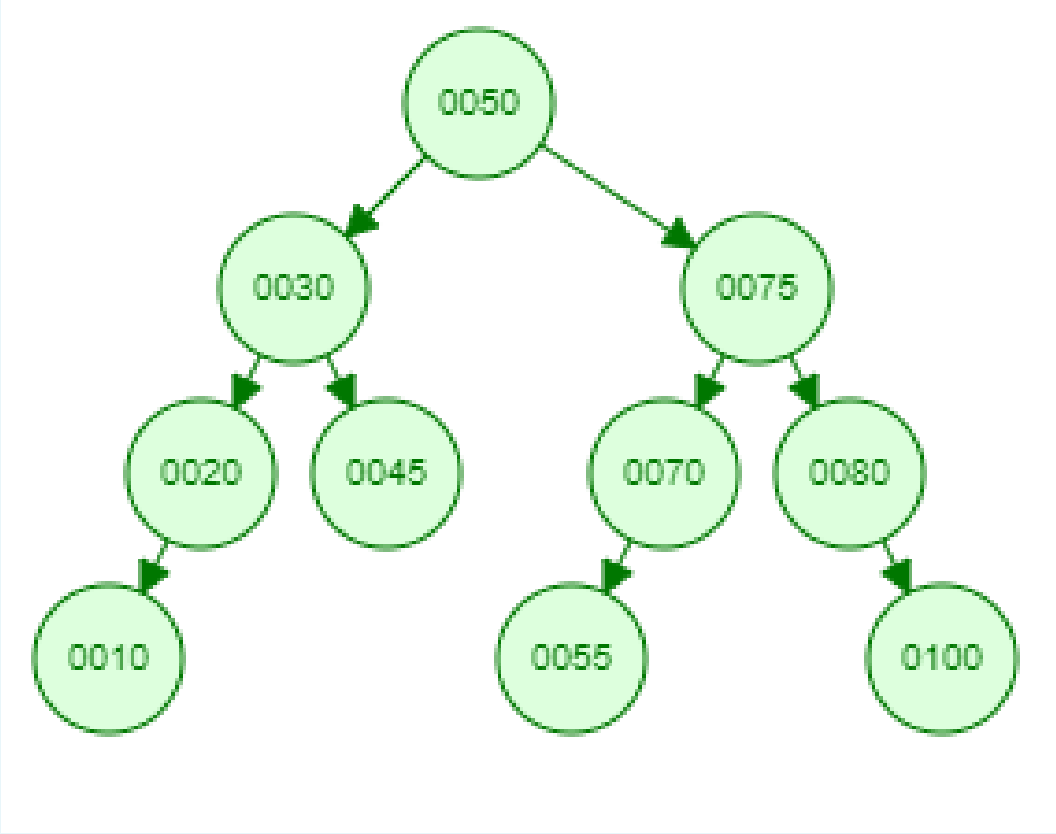


Quay qua trái

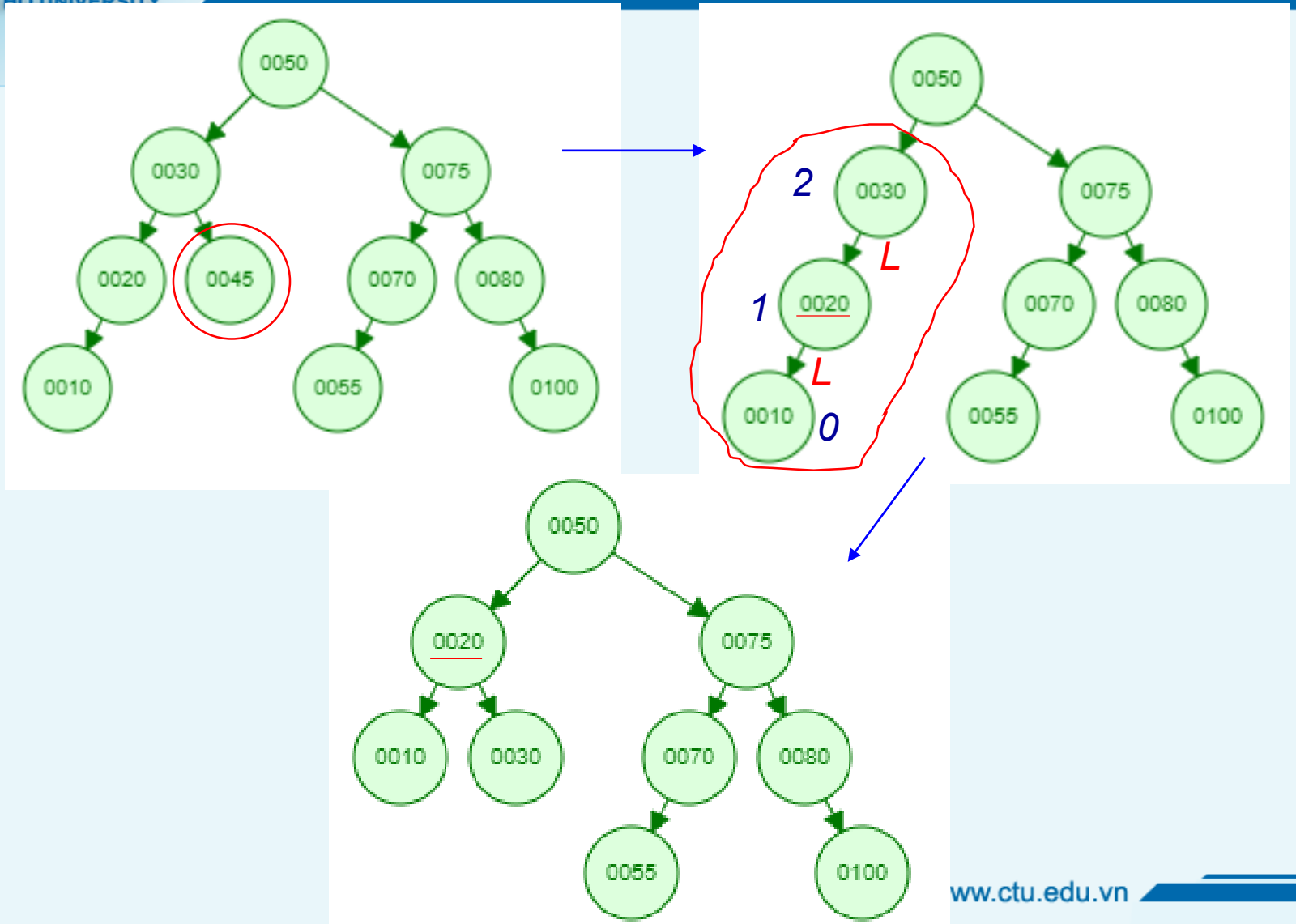


Vẽ cây AVL cho bởi danh sách

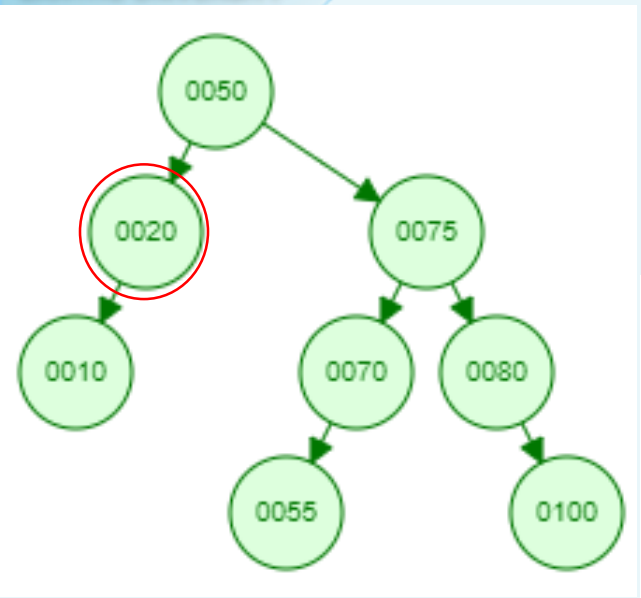
50, 70, 30, 10, 20, 45, 80, 75, 100, 55

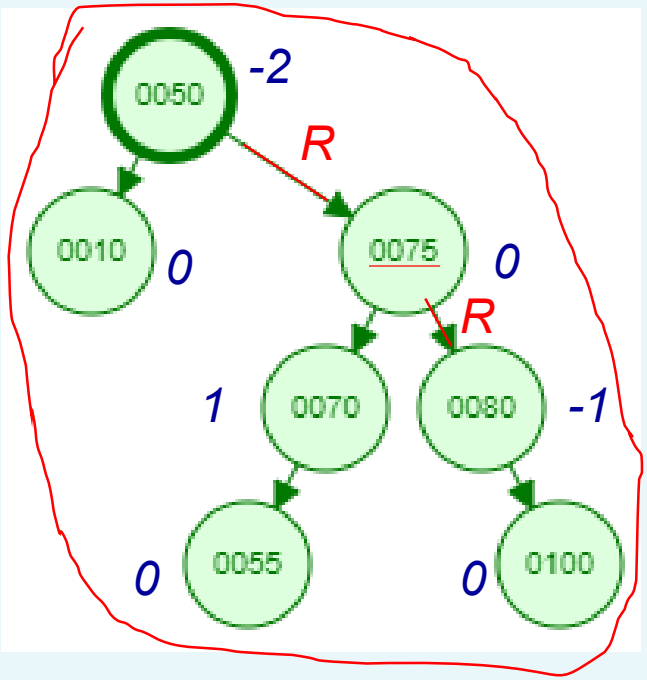


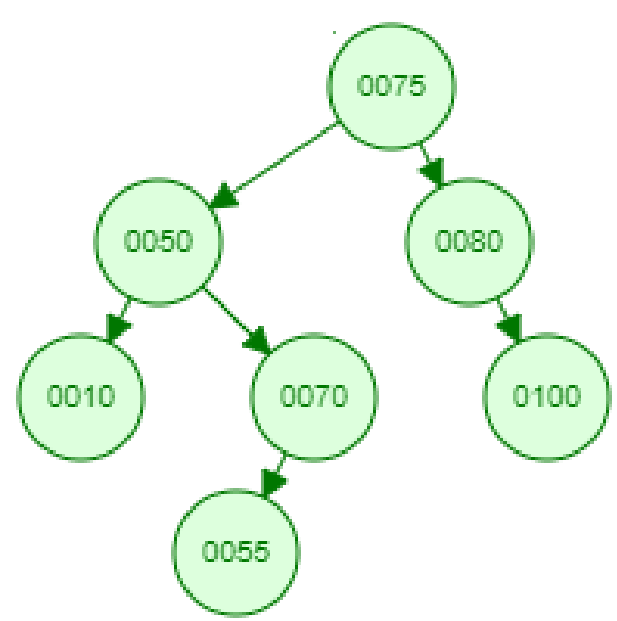
Xóa nút 45


0050
0030
0075
0020
0045
0070
0080
0010
0055
0100
2
L
1
L
0
0050
0030
0075
0020
0070
0080
0010
0055
0100
0050
0020
0075
0010
0030
0070
0080
0055
0100

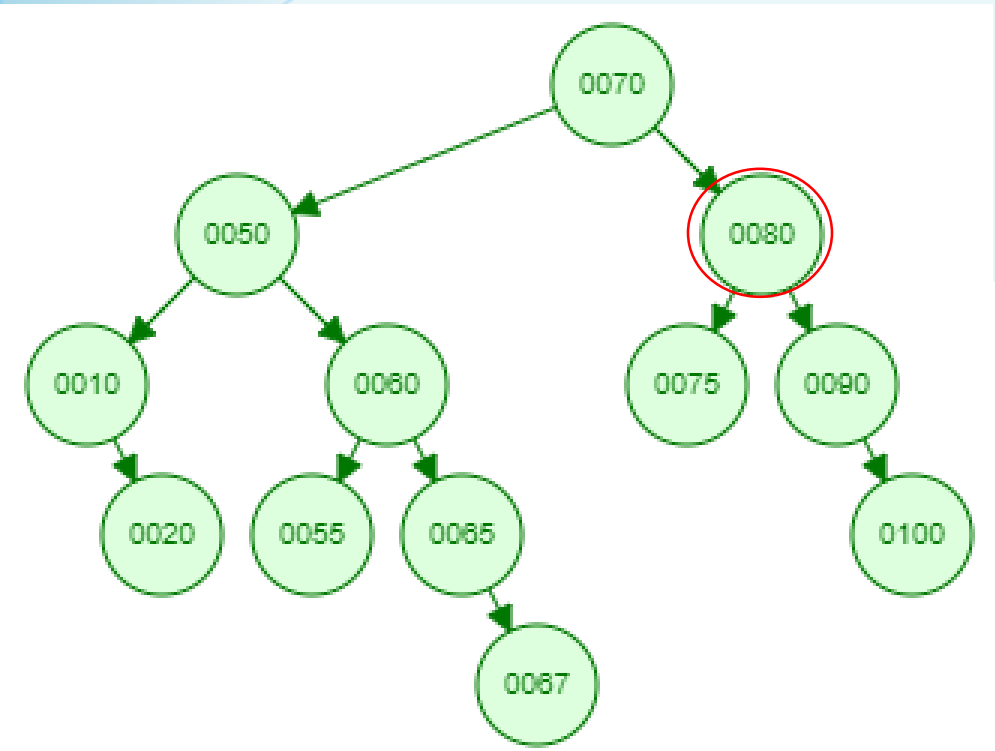
Xóa nút 20 trên cây

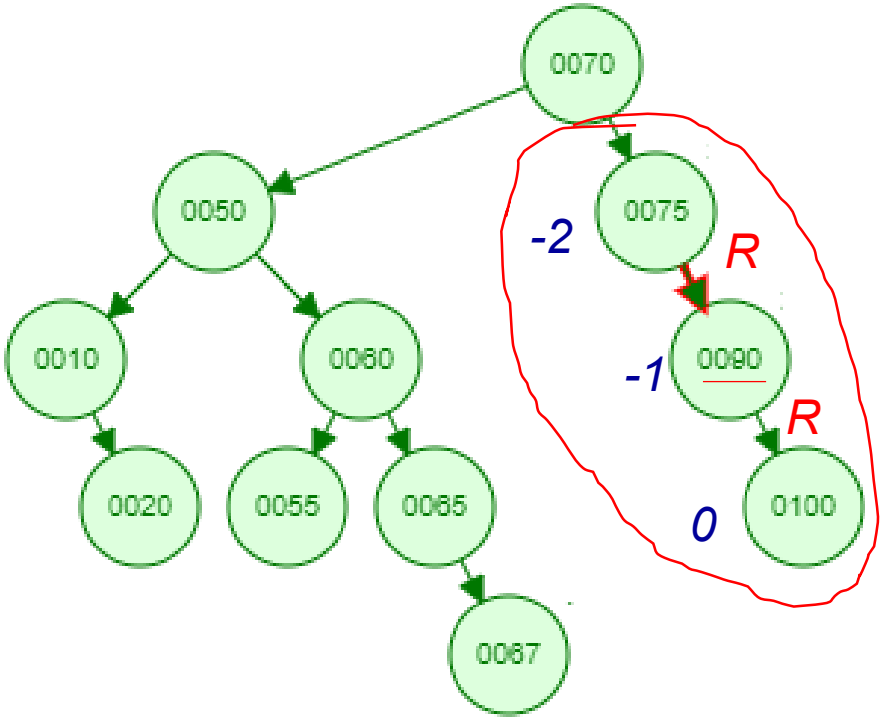




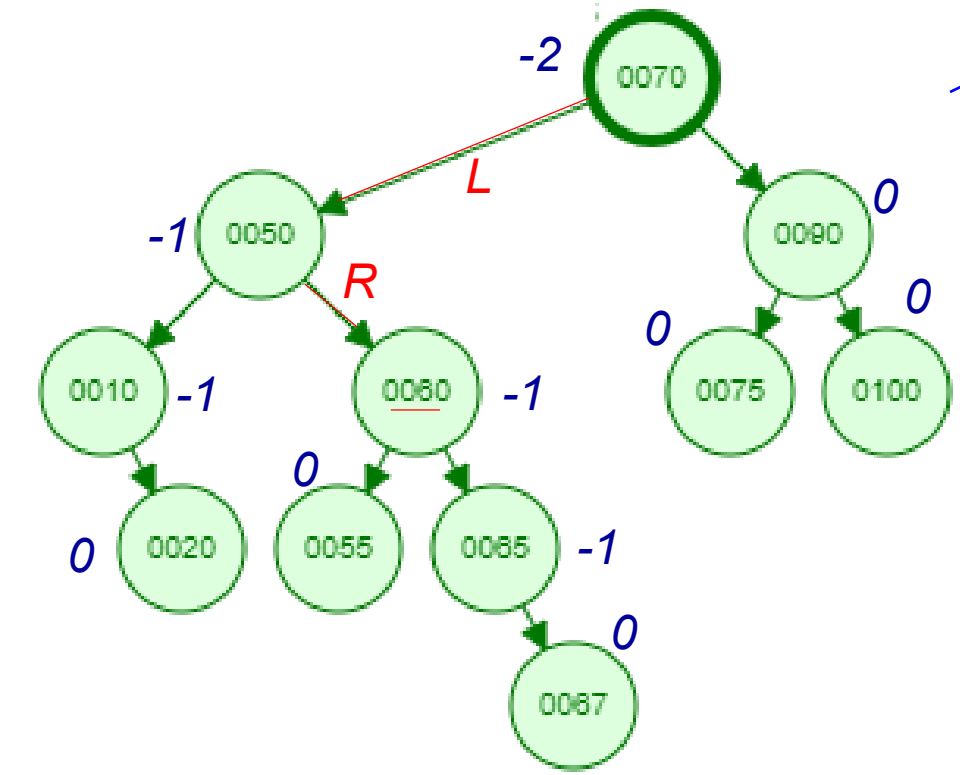


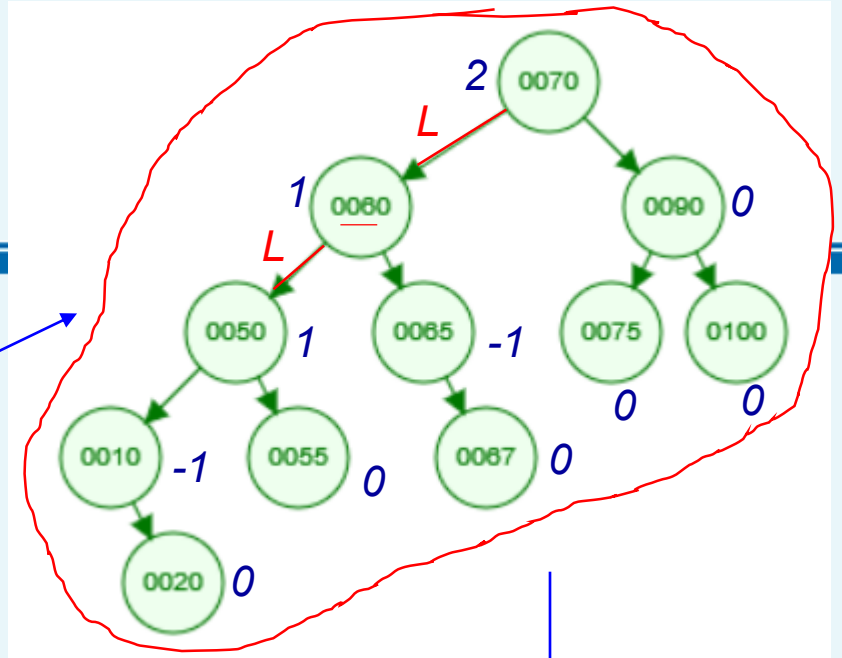
Xóa nút 80

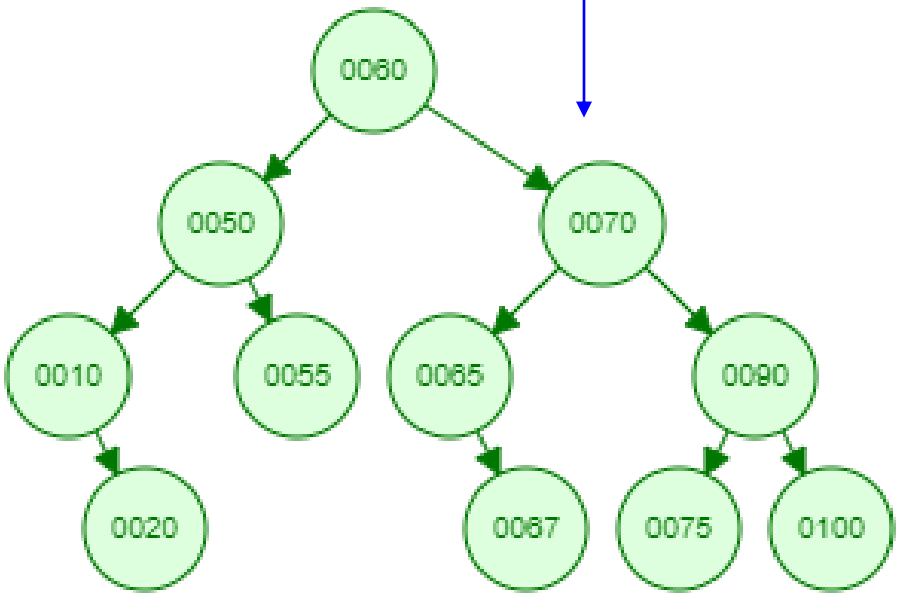

0070
0050
0080
0010
0060
0075
0090
0020
0055
0065
0100
0067


0070
0050
0075
-2
R
0010
0060
0090
-1
R
0020
0055
0065
0100
0
0067

Xóa nút 80


-2
0070
L
-1
0050
R
0090
0
0010
-1
0060
-1
0
0075
0100
0
0
0
0020
0055
0065
-1
0
0067


2
0070
L
1
0060
L
0090
0
0050
1
0065
-1
0075
0100
0
0
0010
-1
0055
0
0067
0
0020
0


0060
0050
0070
0010
0055
0065
0090
0020
0067
0075
0100

Ý tưởng cài đặt cây AVL

Mỗi nút trên cây cần có các thông tin:

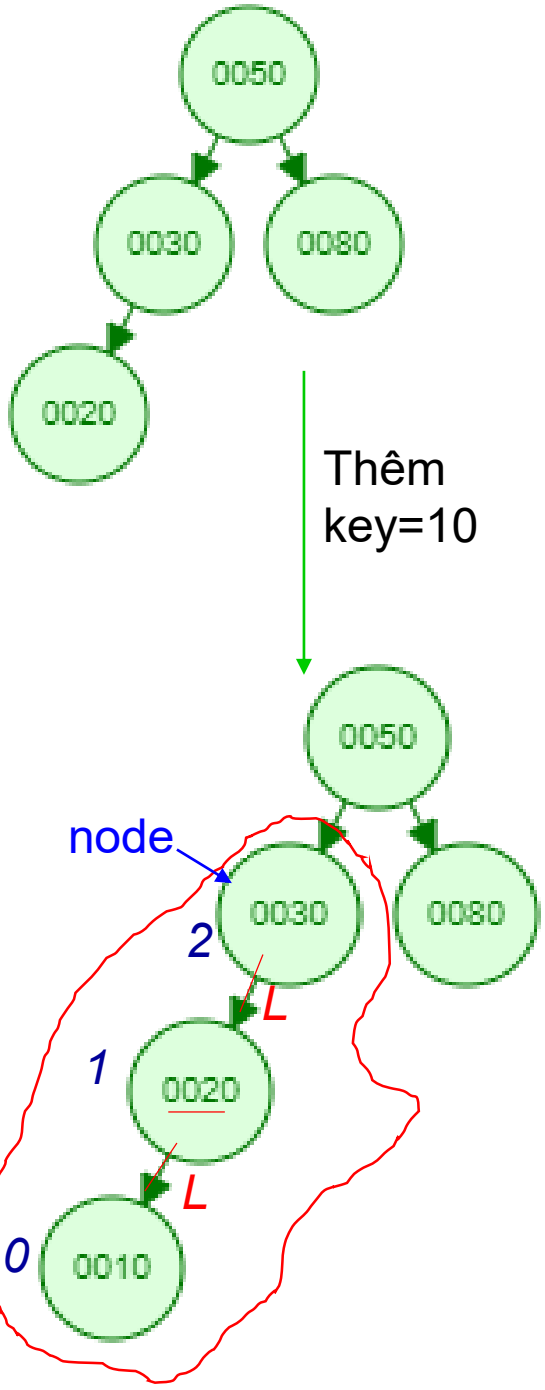
- Khóa của nút
- Chiều cao của nút
- Cây con trái
- Cây con phải

Khai báo cây AVL

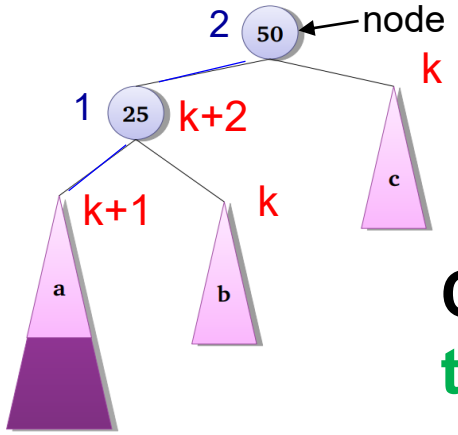
```
typedef <keytype> KeyType;
struct Node {
    KeyType         Key;
    int             Height;
    struct Node   *Left;
    struct Node   *Right;
};
typedef struct Node*   AVLTree;
```

Thêm nút vào cây AVL

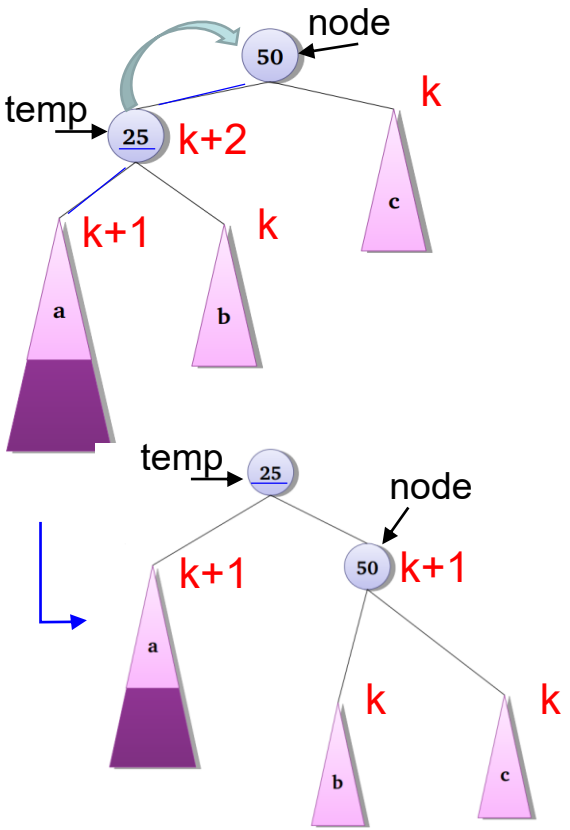
- Thêm nút vào AVL tương tự như thêm nút vào cây BST thông thường.
- Xét xem cây có bị mất cân bằng hay không để thực hiện cân bằng lại cây theo các quy tắc quay và cập nhật lại chiều cao của các nút.

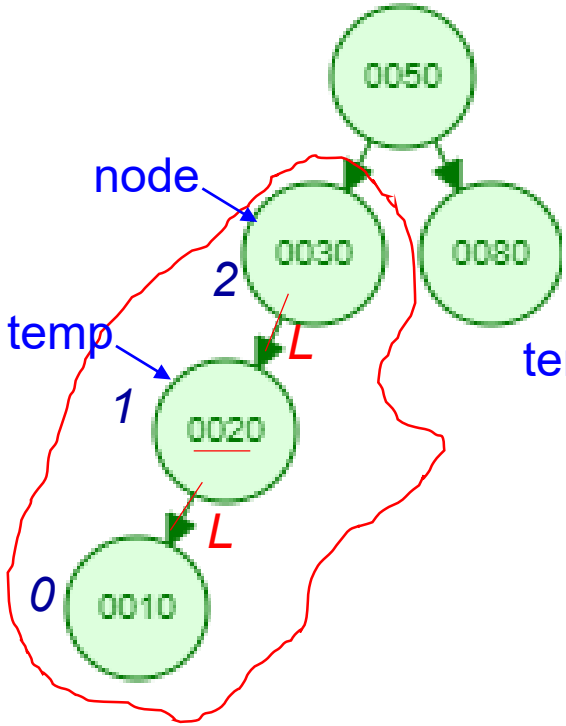


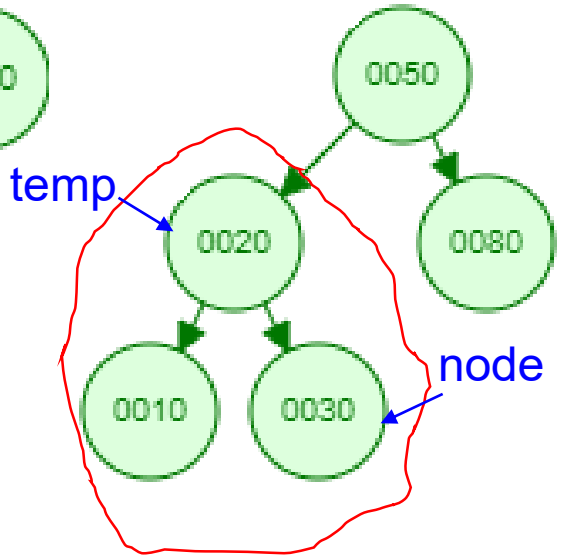

```
struct Node* insertNode(KeyType key,struct Node* node){
    int balance;
    if (node == NULL)   {
        ...
    }
    else if (key < node->Key)
        node->Left = insertNode(key,node->Left);
    else if (key > node->Key)
        node->Right = insertNode(key,node->Right);
    else
        return node; //Khong them
    node->Height = 1 + max(getHeight(node->Left),
                           getHeight(node->Right));
    balance = getBalance(node);
    //TH1: L-L
    if (balance > 1 && key < node->Left->Key)
        return rightRotate(node);
    ...
}
```



Cây mất cân bằng bên trái của con trái (L-L)

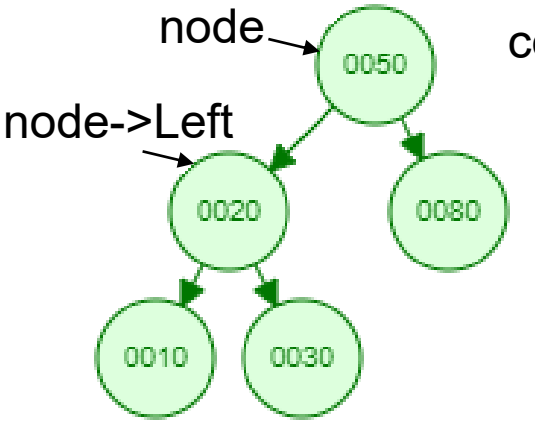




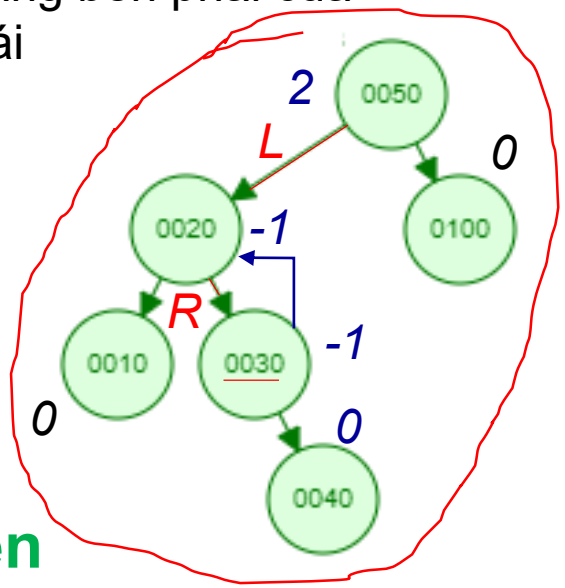


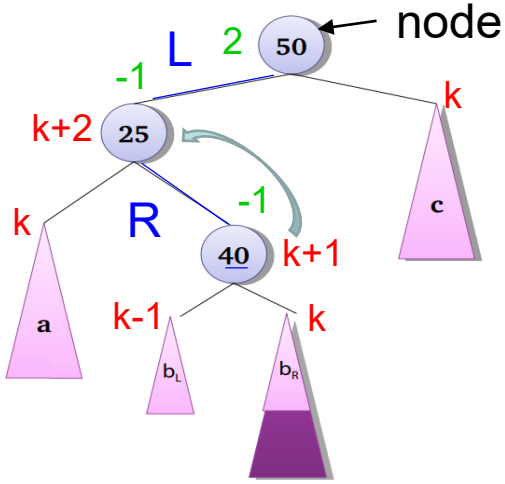
- TH1 (L-L): thực hiện quay đơn qua phải (right rotate)

```
struct Node* rightRotate(struct Node* node){
    struct Node* temp = node->Left;
    node->Left = temp->Right;
    temp->Right=node;
    node->Height = max(getHeight(node->Left),
                       getHeight(node->Right)) + 1;
    temp->Height = max(getHeight(temp->Left),
                       getHeight(temp->Right)) + 1;
    return temp;
}
```



Thêm key=40 => Mất cân bằng bên phải của con trái

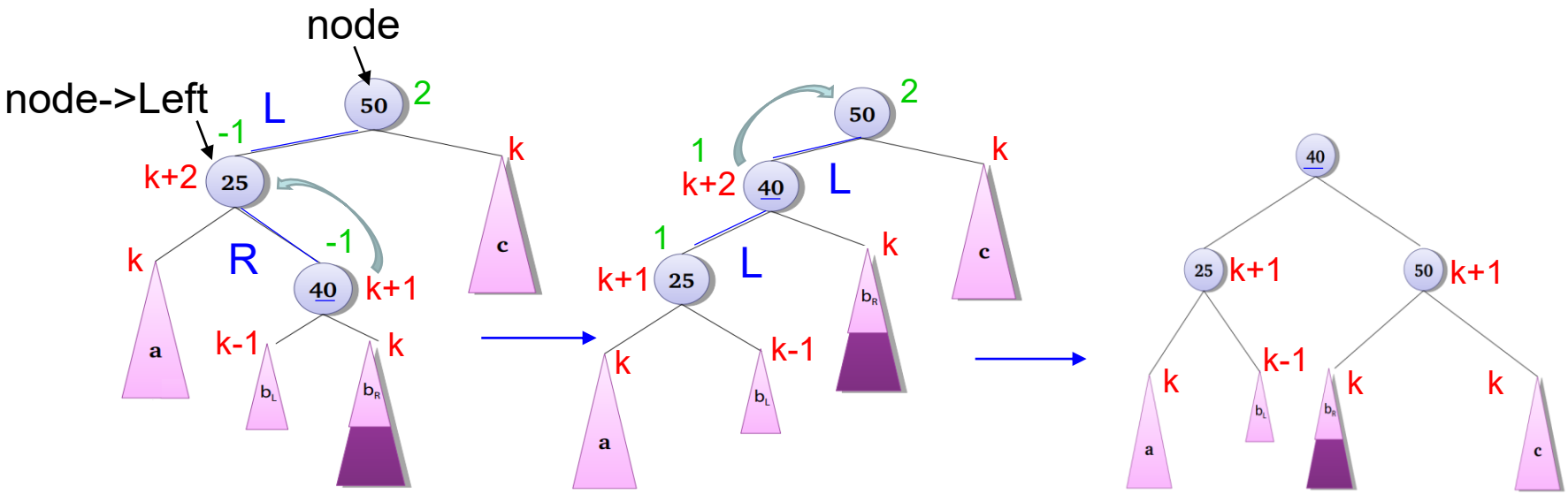




Cây mất cân bằng bên phải của con trái (R-L)

```
struct Node* insertNode(KeyType k,struct Node* node){
    ...
    //TH1: L-L
    if (balance > 1 && k < node->Left->Key)
        return rightRotate(node);
    ...
    //TH3: R-L
    if (balance > 1 && k > node->Left->Key)
        return leftrightRotate(node);
    ...
}
```

- TH3 (R-L): thực hiện quay kép trái-phải (left-right rotate)




```
struct Node* leftrightRotate(struct Node* node){
    node->Left = leftRotate(node->Left);
    return rightRotate(node);
}
```

Xóa nút ra khỏi cây AVL

- Xóa nút ra khỏi AVL tương tự như xóa nút ra khỏi cây BST thông thường.
- Xét xem cây có bị mất cân bằng hay không để thực hiện cân bằng lại cây theo các quy tắc quay và cập nhật lại chiều cao của các nút.
- Lưu ý: quá trình xóa nút có thể dẫn đến cây mất cân bằng dây chuyền liên tiếp đến các nút tiền bối của nó. Theo đó, nguyên tắc thực hiện là phải cân bằng từ nút con trước rồi đến lên nút cha.

CANTHO UNIVERSITY

Tài liệu tham khảo

- Adam Drozdek, Data structures and Algorithms Analysis in C++ 4th Edition, Cengage Learning, 2012
- Introduction to Algorithms, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest & Clifford Stein, MIT Press, 2012
- https://www.cs.usfca.edu/~galles/cs245S08/lecture/lecture23.pdf
- http://www.cse.chalmers.se/edu/year/2018/course/DAT037/slides/6c-avl-trees.pdf
- **https://www.cs.usfca.edu/~galles/visualization/AVLtree.html**
- https://www.tutorialspoint.com/data_structures_algorithms/avl_tree_algorithm.htm

Q&A?