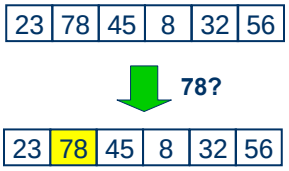
Chương VII : Tìm kiếm

Tìm kiếm – Phần I

- Nội dung
  1. Tìm kiếm tuần tự và tìm kiếm nhị phân
  2. Tìm kiếm trên cây nhị phân
     1. Cây nhị phân tìm kiếm
        1. Đặc điểm của cây nhị phân tìm kiếm
        2. Thao tác bổ sung trên cây nhị phân tìm kiếm
        3. Thao tác loại bỏ trên cây nhị phân tìm kiếm
     2. Cây nhị phân tìm kiếm cân bằng (AVL)
        1. Khôi phục tính cân bằng khi thực hiện bổ sung và loại bỏ

Bài toán Tìm kiếm

- Tìm kiếm là thuật toán tìm 1 phần tử có giá trị cho trước trong một tập các phần tử



- Khóa tìm kiếm: Một bộ phận của các phần tử trong tập mà giá trị của nó được sử dụng để so sánh và tìm kiếm

Tìm kiếm tuần tự

- Tìm kiếm tuần tự
  - Các phần tử trong tập đầu vào không được sắp xếp theo khóa tìm kiếm
  - Mô tả
    - Duyệt dãy (danh sách, hàng đợi , v…v) chứa các phần tử trong tập
    - So sánh với khóa cần tìm tới khi tìm thấy khóa hoặc duyệt qua hết mảng mà chưa tìm thấy
    - Trả lại chỉ số phần tử trong dãy (nếu thấy)

Tìm kiếm tuần tự

```
Function SEQUENTIAL(A, n, key)
{tìm phần tử có khóa key trong mảng A gồm n phần tử. Kết quả
    trả ra: -1 nếu không tìm thấy phần tử có khóa key, chỉ số của
    phần tử nếu tìm thấy}
1. i:= 1;
2. while (i <= n ) and (A[i] <> key) do
        i:= i + 1;
3. if (i> n) then return -1 { không thấy};
4. else
        return i{tìm thấy tại vị trí i}
```

Tìm kiếm tuần tự

- Độ phức tạp :
  - Trường hợp tốt nhất: O(1)
  - Trường hợp tồi nhất: O(n)
  - Trường hợp trung bình : O(n)

Tìm kiếm nhị phân

- Tìm kiếm nhị phân
  - Sử dụng cho việc tìm kiếm trên mảng đã được sắp xếp
  - Mô tả
    - Chọn phần tử "ở giữa" dãy – A[k] để thực hiện so sánh với giá trị cần tìm
    - Nếu key = A[k] thì tìm thấy , kết thúc
    - Nếu key < A[k] thì tìm trên nửa đầu của mảng đã cho
    - Nếu key > A[k] thì tìm trên nửa sau của mảng đã cho

Tìm kiếm nhị phân

```
Function BINARY-SEARCH(A,l, r, key)
1. If (l> r) return -1;
2. m = (l+r) /2 ;
3. If (A[m] = key ) return m ;
4. Else if (A[m] > key) return BINARY-SEARCH(A, l, m-1, key);
5. Else return BINARY-SEARCH(A, m+1, r, key);
```

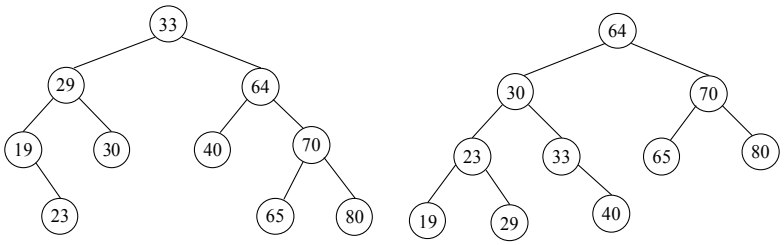
Tìm kiếm nhị phân

```
Function BINARY-SEARCH(A,n,key)
1. l:=1 ; r := n ; { l, r lần lượt là biến chỉ số sử dụng để ghi nhận chỉ số của phần tử
   đầu và phần tử cuối của mảng mà chúng ta đang tìm kiếm trên đó}
2. while l <= r do begin
        {Tìm chỉ số của phần tử giữa} m:= (l+r) / 2;
        if key < A[m] then r:= m-1;
        else
            if key > A[m] then l:= m+1
            else return m;
    end;
3. { Không tìm thấy } return -1;
```

Cây nhị phân tìm kiếm
Binary Search Tree (BST)

- Cây tìm kiếm nhị phân ứng với 1 dãy gồm n khóa a_1, a_2, …, a_n là một cây nhị phân thỏa mãn tính chất sau
  - Mọi giá trị thuộc cây con trái của một nút đều nhỏ hơn giá trị tại nút đó
  - Mọi giá trị thuộc cây con phải của một nút đều lớn hơn giá trị tại nút đó
  - Mỗi cây con của một nút cũng đều là cây nhị phân tìm kiếm
- Với một tập khóa có thể xác định được nhiều cây nhị phân tìm kiếm

Cây nhị phân tìm kiếm

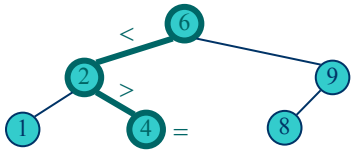


Cây nhị phân tìm kiếm

- Các thao tác trên cây nhị phân tìm kiếm
  - Duyệt cây nhị phân tìm kiếm
  - Tìm kiếm nút có giá trị x
  - Thêm một nút mới có giá trị x
  - Xóa một nút có giá trị x

Tìm kiếm trên cây nhị phân tìm kiếm

- Cách thực hiện
  - Nếu cây rỗng: không tìm thấy
  - Nếu cây không rỗng:
    - So sánh giá trị cần tìm kiếm với các giá trị khóa tìm kiếm ở nút gốc
      - Nếu = → Tìm thấy
      - Nếu < → Đi xuống tìm kiếm trong cây con trái
      - Nếu > → Đi xuống tìm kiếm trong cây con phải



Tìm kiếm trên cây nhị phân tìm kiếm

- Giải thuật đệ qui

Algorithm BST-Recursive(T, key)

{T là con trỏ trỏ tới gốc của cây; key là giá trị cần tìm, trả ra con trỏ trỏ tới nút chứa giá trị cần tìm }

1. If (T = NULL) then return NULL;
2. If (key < INFO(T)) return BST-Recursive(LPTR(T), key);
3. Else if (key > INFO(T)) return BST-Recursive(RPTR(T), key);
4. Else return T;

Tìm kiếm trên cây nhị phân tìm kiếm

Giải thuật không đệ qui

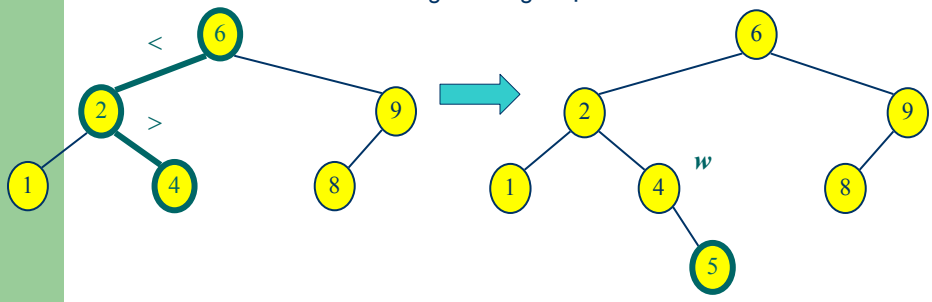
```
Algorithm BST(T, key)
1.  q=  T ; {Khởi tạo biến con trỏ để duyệt cây}
2.  while q < > NULL do begin
            if (INFO(q) = key then return q;
            else begin
                        if (INFO(q) < key) then q = RPTR(q);
                        else q = LPTR(q);
            end.
    end.
3. Return NULL;
```

Bổ sung trên cây nhị phân tìm kiếm

- Cách thực hiện thêm một nút có giá trị x vào cây nhị phân tìm kiếm
  - Tìm nút có giá trị x
  - Nếu tìm thấy, không cần thêm
  - Nếu không tìm thấy
    - Giả sử gọi w là nút lá mà ta chạm đến trong quá trình tìm kiếm
    - Tạo một nút mới có giá trị x và biến nút này thành nút con của w (con trái hay con phải phụ thuộc vào việc so sánh x với giá trị lưu trong w)

Bổ sung trên cây nhị phân tìm kiếm

Bổ sung nút có giá trị 5



Bổ sung trên cây nhị phân tìm kiếm

Algorithm Insert_BST_Recursive(T, x)

{Tìm hoặc bổ sung nút có giá trị x trên cây nhị phân tìm kiếm.

Trả ra cây sau khi bổ sung hoặc trả ra nút có chứa x }

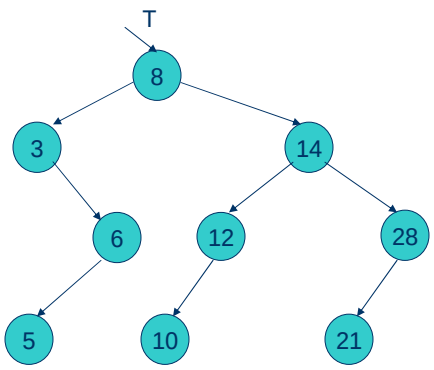
1. If (T = null) then
    1. Call New (p) ; {Xin bộ nhớ cho nút mới}
    2. INFO(p) := x; LPTR(p): = RPTR(p) := NULL;
    3. T = p ;
2. If (key < INFO(T)) then
    1. LPRT(T) := Insert_BST(LPTR(T), x) ;
3. Else if (key > INFO(T)) then begin
    1. RPTR(T) := Insert_BST(RPTR(T), x) ;
4. return T;

Bổ sung trên cây nhị phân tìm kiếm

```
Algorithm Insert_BST(T, x)
{Bổ sung nút mới có giá trị x vào cây, trả ra con trỏ trỏ tới nút mới, hoặc trả ra con
trỏ trỏ tới một nút trong cây nếu trong cây đã có nút chứa khóa x }
1. q=  T ; {Khởi tạo biến con trỏ để duyệt cây}
2. while q < > NULL do begin
     if (INFO(q) = key) then return q; // Tìm thấy, kết thúc giải thuật
     else begin
            if (INFO(q) < key) then begin p= q; q = RPTR(q); end;
            else begin p=q; q = LPTR(q);end;
     end.
end.
3. Call New(q); INFO(q) = x; LPTR(q) = RPTR(q) = NULL;
     if (T= null ) then T = q;
     else if x < INFO(p) then LPTR(p) = q;
     else RPTR(p) = q;
```

Dựng cây nhị phân tìm kiếm

- Ví dụ: Dựng cây nhị phân tìm kiếm sử dụng phép bổ sung cho ở trên với dãy số {8,3,14,6, 12, 28, 10,21,5}

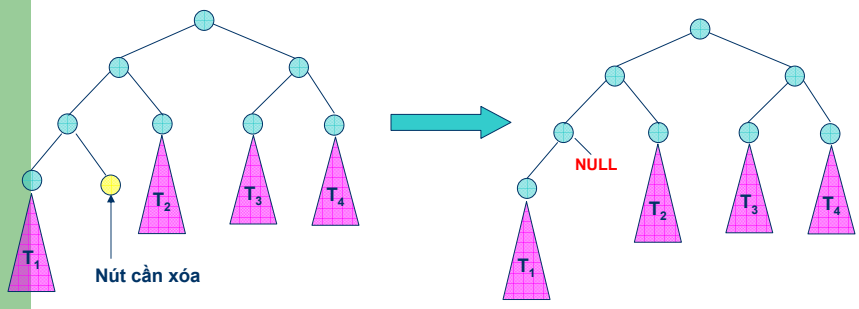


Xóa nút trên cây nhị phân tìm kiếm

- Các trường hợp :
  - Nút loại bỏ là nút lá: Xóa ngay lập tức
  - Nút loại bỏ là nút nhánh và chỉ có một cây con (trái hoặc phải) : Thay nút cần xóa bằng nút con
  - Nút loại bỏ là nút nhánh và có 2 cây con: Thay nút cần xóa bằng nút cực phải của cây con trái hoặc nút cực trái của cây con phải

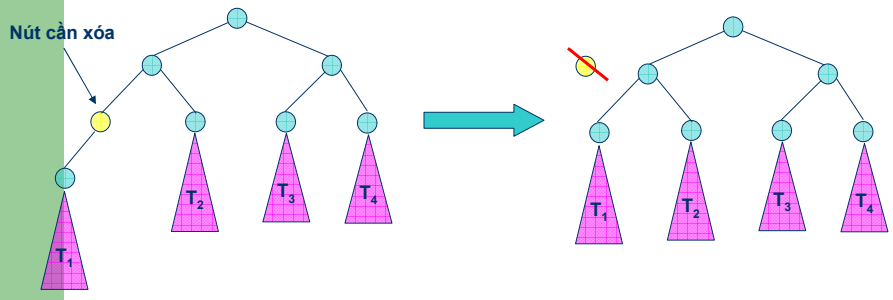
Xóa một nút lá trên cây

- Trường hợp nút cần xóa là nút lá
  - Xóa nút này
  - Gán liên kết từ cha của nó trở thành NULL



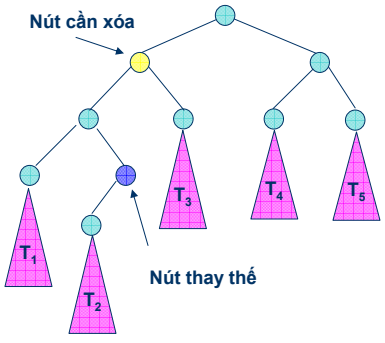
Xóa nút nhánh có 1 con

- Trường hợp nút cần xóa là nút nhánh có 1 con
  - Gắn cây con của nút cần xóa vào cha



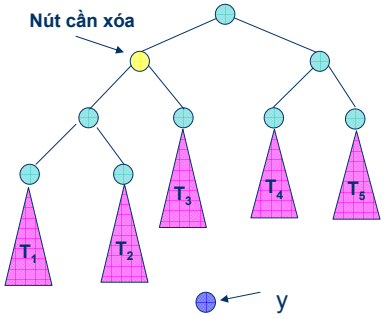
Xóa nút nhánh có đầy đủ 2 con

- Trường hợp nút cần xóa là nút có 2 con
  - Bước 1: Xác định nút thay thế
    - Nút thay thế là nút cực phải của cây con trái hoặc nút cực trái của cây con phải



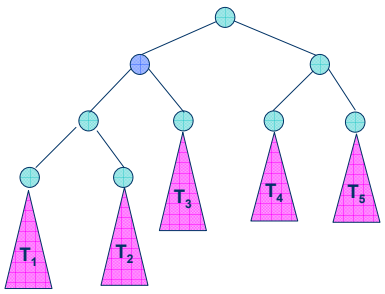
Xóa nút nhánh có đầy đủ 2 con

- Trường hợp nút cần xóa là nút có 2 con
  - Bước 2: Có nút thay thế là y
    - Gỡ y ra khỏi cây
    - Gắn con trái của y vào cha của y



Xóa nút nhánh có đầy đủ 2 con

- Trường hợp nút cần xóa là nút có 2 con
  - Bước 3: Có nút thay thế là y
    - Gắn y vào vị trí của nút cần xóa



Xóa nút trên cây nhị phân tìm kiếm

```
Algorithm BSTDEL(key, nut_xoa, nut_cha)
{Thực hiện việc xóa nút trỏ bởi con trỏ nut_xoa , biết con trỏ nut_cha trỏ tới nút cha
    của nút xóa, biết giá trị key của nút cần xóa}
1.  If (LPTR(nut_xoa) = null && RPTR(nut_xoa) = null) then begin
            if ( key < INFO(nut_cha)) then LPTR(nut_cha): = null;
            else RPTR(nut_cha) := null; call dispose(nut_xoa) ; end;
2.  If (LPTR(nut_xoa) = null || RPTR(nut_xoa) = null) then begin
            if LPTR(nut_xoa) = NULL then nut_thay := RPTR(P);
            else if RPTR(nut_xoa) = NULL then nut_thay := LPTR(P);
            if (key < INFO(nut_cha)) then LPTR(nut_cha) := nut_thay;
            else RPTR(nut_cha) := nut_thay;
            call dispose(nut_xoa); end;
```

Xóa nút trên cây nhị phân tìm kiếm

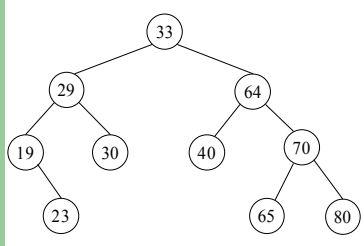
```
3. If (LPTR(nut_xoa) != null && RPTR(nut_xoa) != null) then begin
    nut_thay := LPTR(nut_xoa); {sang cây con trái}
    while RPTR(nut_thay) <> null do begin
        T := nut_thay; nut_thay := RPTR(nut_thay);
    end;{Kết thúc vòng lặp nut_thay trỏ đến nút cực phải của cây con trái, T:nút cha
    của nút thay}
    RPTR(nut_thay) := RPTR(nut_xoa); RPTR(T) := LPTR(nut_thay);
    LPTR(nut_thay) := LPTR(nut_xoa);
    if (key < INFO(nut_cha)) then LPTR(nut_cha) := nut_thay;
        else RPTR(nut_cha) := nut_thay;
        call dispose(nut_xoa);
End.
```

Cây nhị phân tìm kiếm

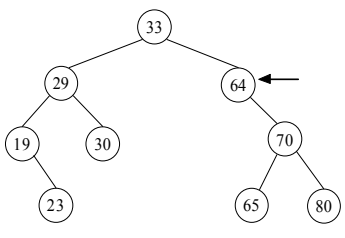
- Đánh giá giải thuật : tìm kiếm và loại bỏ
  - Thời gian thực hiện trung bình $T_{tb}(n) = O(\log_2 n)$
- Nhược điểm của cây tìm kiếm nhị phân:
  - Cây suy biến có thể được hình thành trong quá trình bổ sung, ảnh hưởng đến hiệu năng của việc sử dụng cây nhị phân trong tìm kiếm

Cây nhị phân cân đối AVL

- Cây nhị phân cân đối AVL (AVL balanced binary search tree)
  - Một cây nhị phân tìm kiếm được gọi là cây cân đối AVL nếu với mọi nút trên cây, chiều cao của 2 cây con tương ứng chỉ chênh nhau nhiều nhất là 1 đơn vị



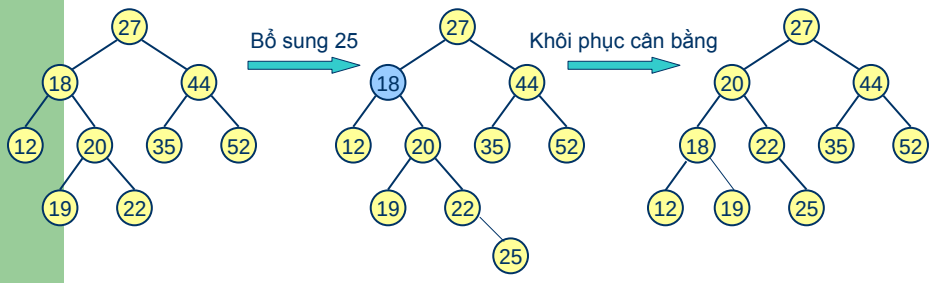
Cây nhị phân tìm kiếm cân đối AVL



Cây nhị phân tìm kiếm không cân đối

Cây nhị phân cân đối AVL

- Bổ sung trên cây AVL
  - Bổ sung theo nguyên tắc giống với cây nhị phân tìm kiếm
  - Việc bổ sung có thể làm vi phạm tính cân bằng của cây

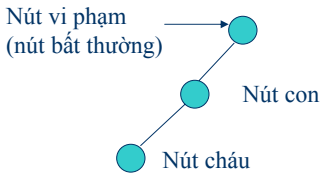


Cây nhị phân cân đối AVL

- Khôi phục tính cân bằng của cây
  - Kiểm tra tính cân bằng của các nút nằm trên đường đi từ nút gốc đến nút mới được bổ sung
  - Xác định nút vi phạm gần nhất với nút mới
  - Thực hiện các phép quay với nút vi phạm mà không cần thực hiện phép quay nào khác tại tổ tiên của nút đó
    - Tùy vào vị trí nút mới so với nút vi phạm có 4 loại phép quay khác nhau

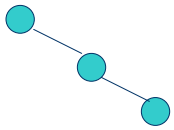
Cây nhị phân cân đối AVL

- Xác định các phép quay cần sử dụng
  - Bước 1: Xác định nút vi phạm gần nhất
  - Bước 2: Quan sát vị trí của nút con và nút cháu của nút vi phạm trên đường đi xác định vị trí bổ sung
    - Trường hợp 1: Quay đơn phải

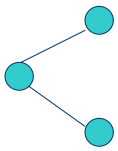


Cây nhị phân cân đối AVL

- Trường hợp 2: Quay đơn trái (single left rotation)

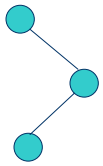


- Trường hợp 3: Quay kép phải (double right rotation) : quay trái với cây con trái rồi quay phải với cây có nút vi phạm và con trái của nó



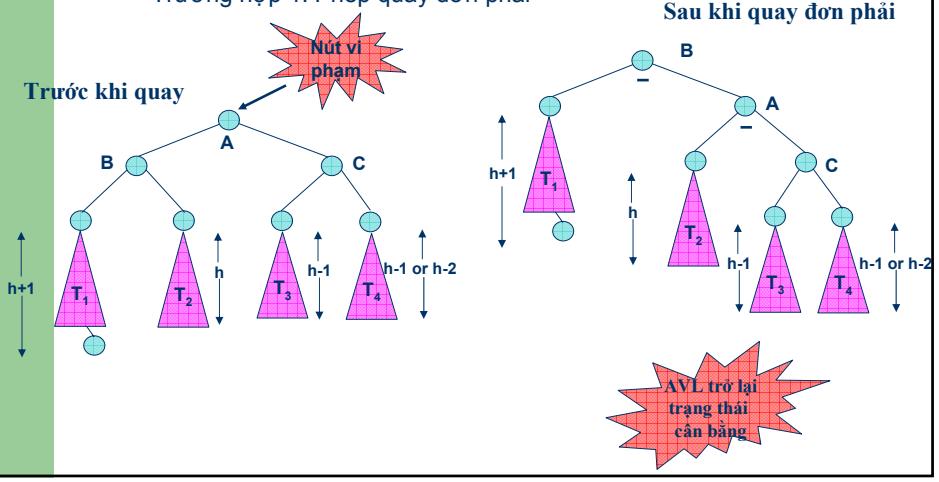
Cây nhị phân cân đối AVL

- Trường hợp 4: Quay kép trái (double left rotation) : quay phải với cây con phải, rồi quay trái với cây có nút vi phạm và con phải của nó



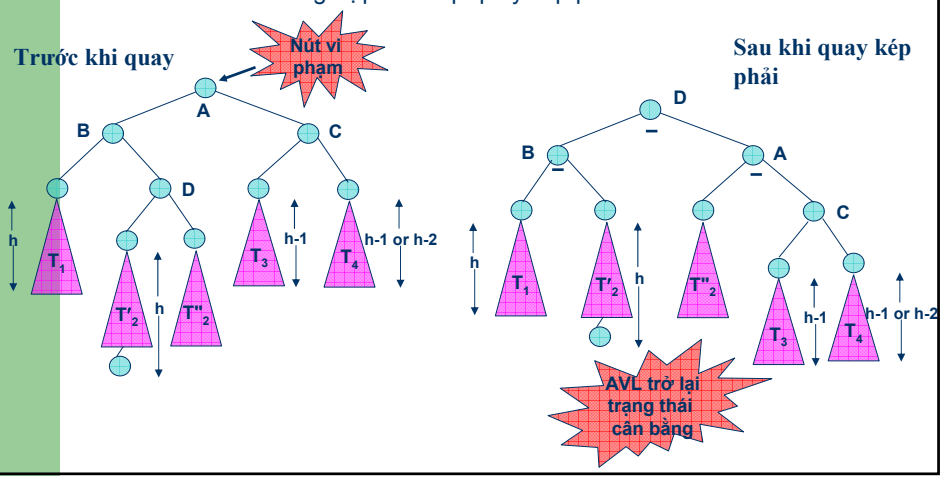
Cây nhị phân cân đối AVL

- Trường hợp 1: Phép quay đơn phải



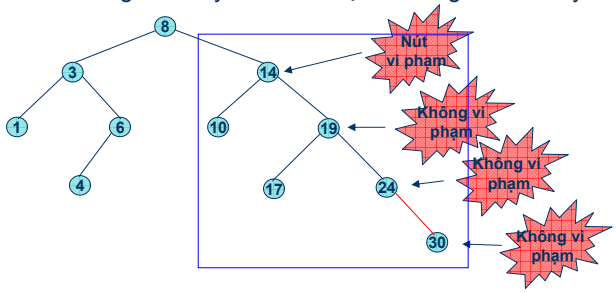
Cây nhị phân cân đối AVL

– Trường hợp 3: Phép quay kép phải



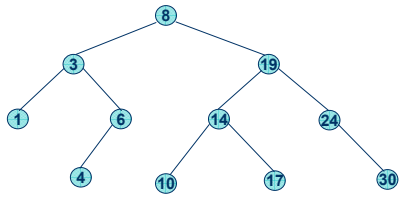
Cây nhị phân cân đối AVL

– Bổ sung trên cây AVL – Ví dụ: Bổ sung 30 vào cây s



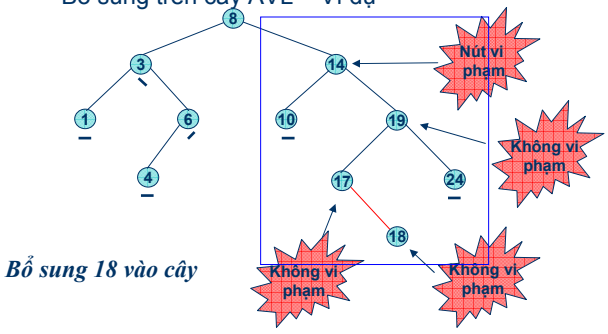
Cây nhị phân cân đối AVL

- Để sửa đổi lại cây, quay cây có gốc tại 14: Quay từ phải sang trái với cây con phải (nút 14 và 19) – Phép quay này gọi là phép quay đơn trái



Cây nhị phân cân đối AVL

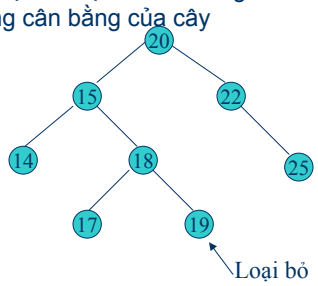
- Bổ sung trên cây AVL – Ví dụ



Bổ sung 18 vào cây

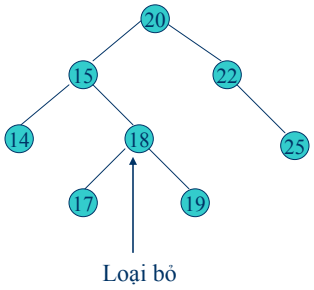
Cây nhị phân cân đối AVL

- Loại bỏ trên cây AVL cũng có thể dẫn đến tình trạng mất cân đối của cây, tương tự như trong phép bổ sung, ta cũng sẽ thực hiện phép quay để tái cân bằng lại cây.
  - Ví dụ: Loại bỏ một nút lá không làm ảnh hưởng đến tình trạng cân bằng của cây



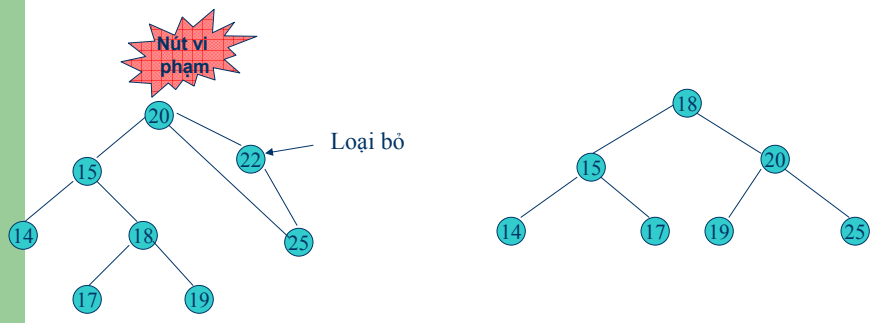
Cây nhị phân cân đối AVL

- Ví dụ: Loại bỏ một nút nhánh không làm ảnh hưởng đến tính cân bằng của cây



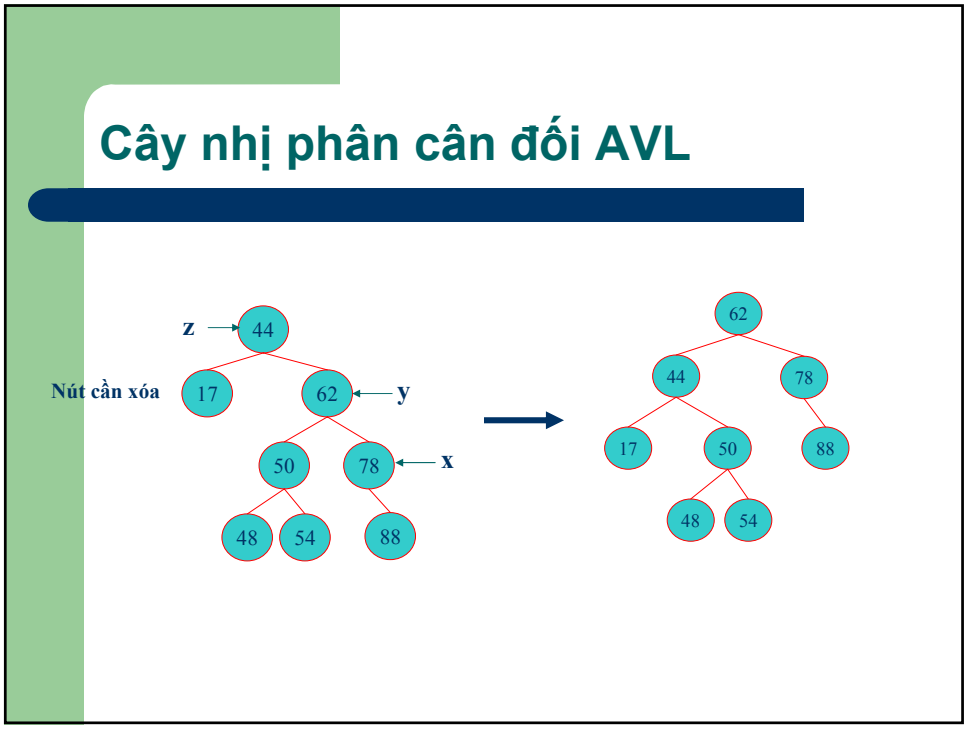
Cây nhị phân cân đối AVL

- Ví dụ: Loại bỏ một nút dẫn đến phải thực hiện phép quay để tái cân bằng cây



Cây nhị phân cân đối AVL

- Tái cân bằng lại cây sau khi loại bỏ trên cây AVL
  - Gọi z là nút đầu tiên không cân bằng trên đường đi từ vị trí của nút bị loại lên đến gốc cây.
  - Gọi y là nút con của z, y là nút con có chiều cao lớn hơn
  - Gọi x là nút con của y, x là nút con có chiều cao lớn hơn
  - Ta sẽ thực hiện phép quay tại nút z khi xét thêm cả y, x để tái cân bằng lại cây
  - Phép quay có thể ảnh hưởng đến tính cân bằng của các nút có chiều cao lớn hơn z trong cây (các nút tổ tiên của z trên đường đi đến gốc) vì vậy cần phải kiểm tra tính cân bằng của các nút đó cho đến khi chạm tới gốc.


Cây nhị phân cân đối AVL
z
44
Nút cần xóa
17
62
y
50
78
x
48
54
88
62
44
78
17
50
88
48
54