CẤU TRÚC DỮ LIỆU
CÂY AVL

Bùi Tiến Lên

01/01/2017

KHOA CÔNG NGHỆ THÔNG TIN
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

Một số vấn đề của cây nhị phân tìm kiếm

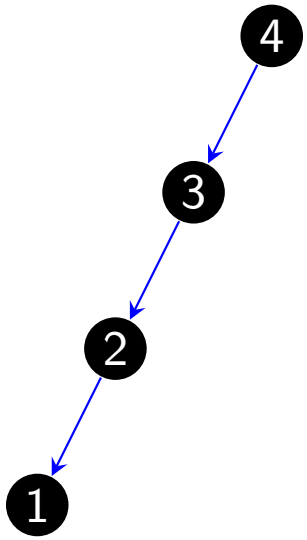
Vấn đề

Khi thực hiện các thao tác trên cây nhị phân tìm kiếm chẳng hạn như thêm, xóa có thể dẫn đến cây mất cân bằng. Dẫn đến cây nhị phân tìm kiếm không còn hiệu quả

Một số vấn đề của cây nhị phân tìm kiếm (cont.)

Ví dụ 1

Tạo cây nhị phân tìm kiếm từ dãy các số $\{4, 3, 2, 1\}$ ta sẽ được



Hình 1: Cây tuyến tính

Cây nhị phân tìm kiếm cân bằng

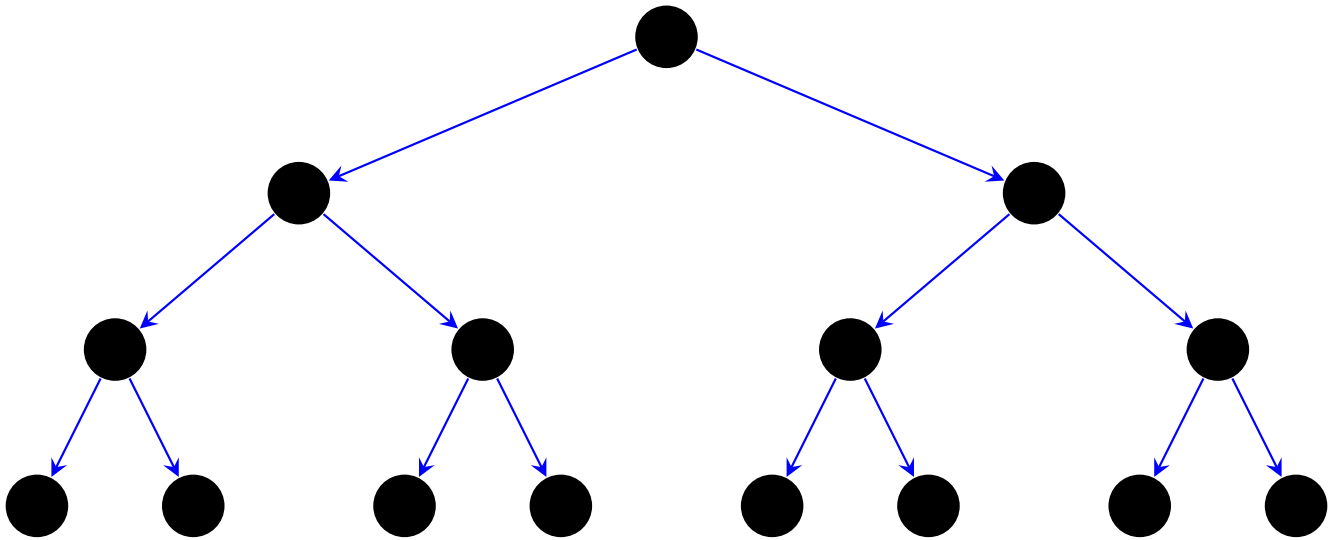
Định nghĩa 1

Cây cân bằng tối ưu (**perfect tree**) là cây có chiều cao $h = \log_2(n+1)$ với n là số nút của cây

Cây nhị phân tìm kiếm cân bằng (cont.)

Ví dụ 2

Cây hoàn chỉnh là một cây cân bằng tối ưu



Hình 2: Cây nhị phân tìm kiếm hoàn chỉnh

Cây nhị phân tìm kiếm cân bằng (cont.)

Ý tưởng

Có hai chiến lược cân bằng:

- Cân bằng theo chu kỳ hoạt động
- Cân bằng theo thao tác cập nhật

Đa số kỹ thuật sử dụng biến đổi xoay để cân bằng lại

Các phép biến đổi

Để duy trì được sự cân bằng trong cây T, các nhà lập trình thường sử dụng các phép biến đổi sau

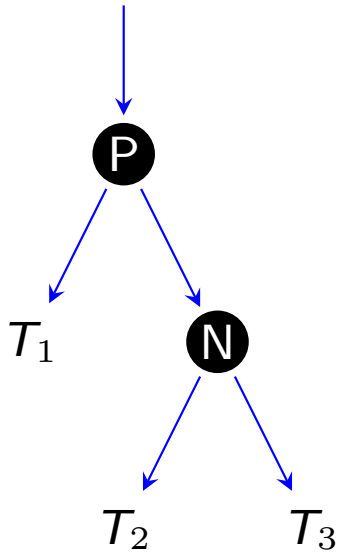
- Phép xoay trái (*left rotation*)
- Phép xoay phải (*right rotation*)

Định lý 1

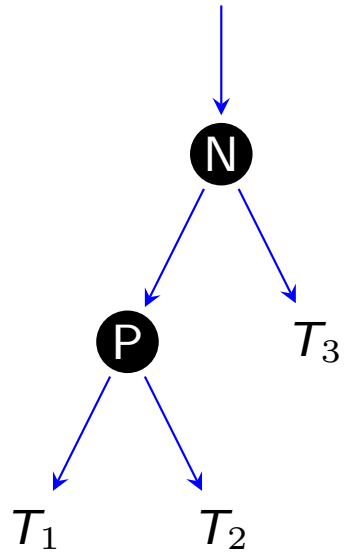
Các phép biến đổi xoay trái và xoay phải không làm mất đi tính chất "tìm kiếm" của cây

Các phép biến đổi (cont.)

Thực hiện xoay trái giữa hai nút P và N; trong đó, N là nút con phải của P



(a) trước khi xoay

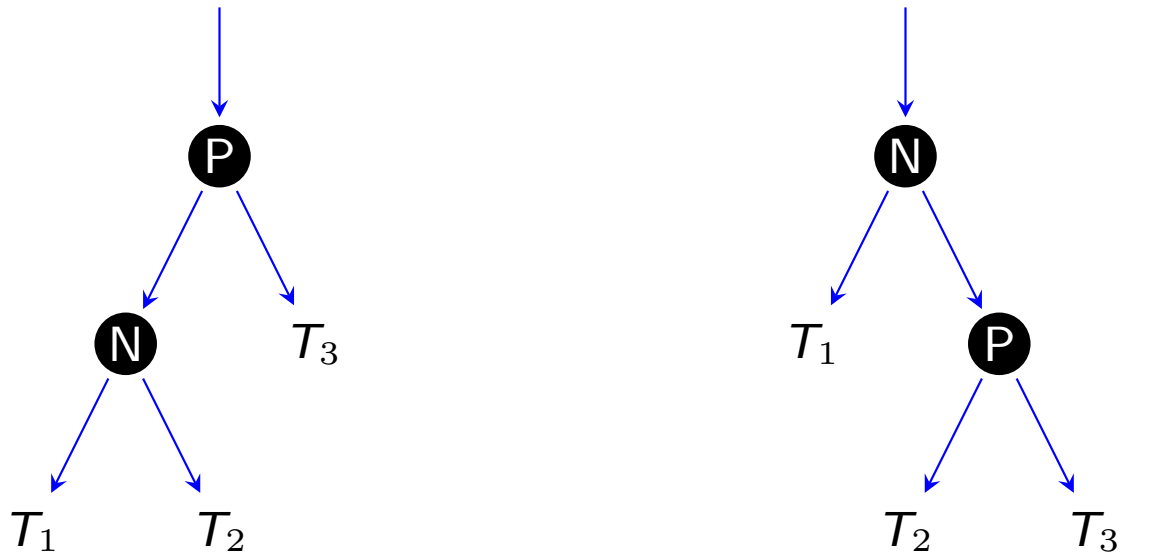


(b) sau khi xoay

Hình 3: Thao tác xoay trái

Các phép biến đổi (cont.)

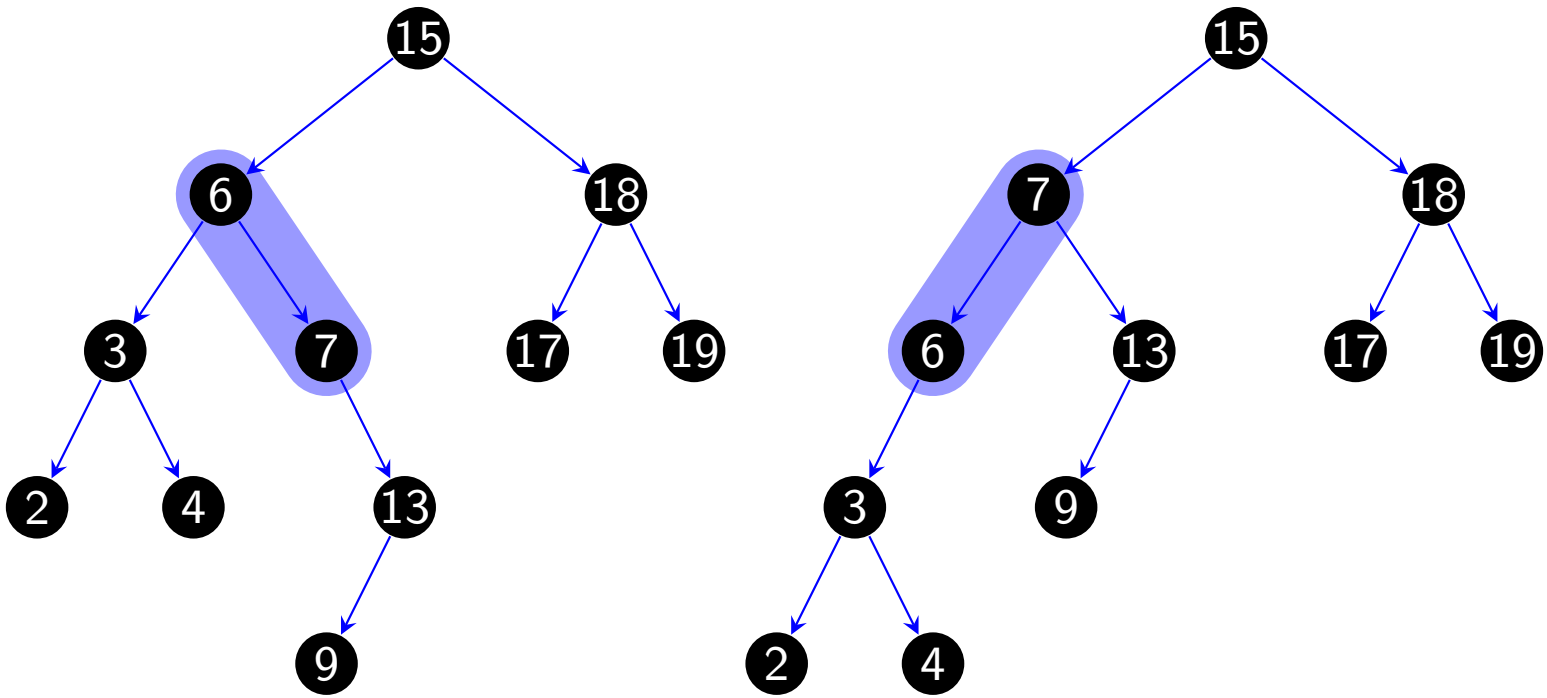
Thực hiện xoay phải giữa hai nút P và N; trong đó, N là nút con trái của P



(a) trước khi xoay **(b)** sau khi xoay

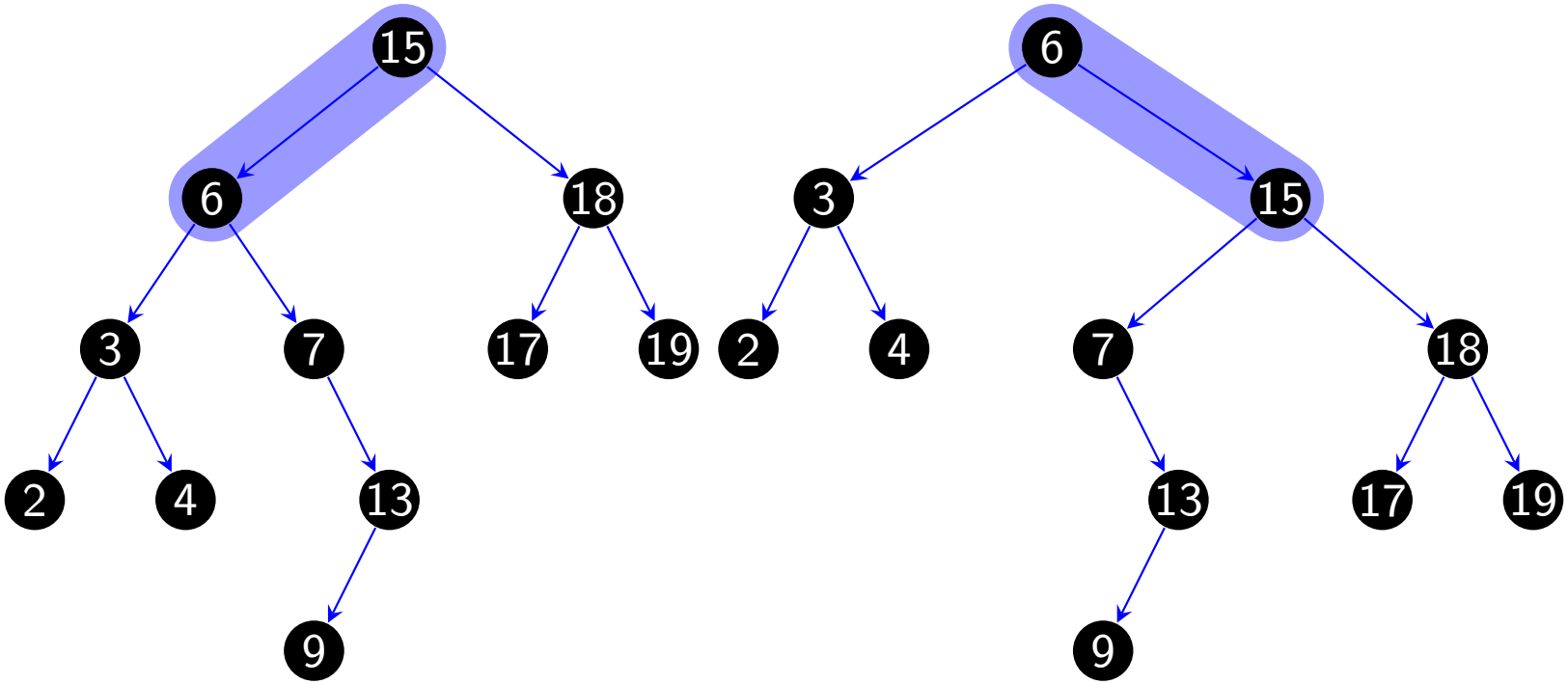
Hình 4: Thao tác xoay phải

Các phép biến đổi (cont.)



Hình 5: Xoay trái 6 và 7

Các phép biến đổi (cont.)



Hình 6: Xoay phải 6 và 15

Thuật toán DSW

Ý tưởng

Thuật toán được Quentin F. Stout và Bette L. Warren đề xuất dựa trên công trình của Colin Day. Đây là thuật toán được hoạt động theo từng chu kỳ hoạt động

- Biến đổi cây BST thành cây **backbone** có dạng như danh sách liên kết
- Cây **backbone** kết quả sẽ được xoay liên tục để trở thành cây hoàn chỉnh

Thuật toán DSW (cont.)

```
createBackbone(root)
  p ← root
  while(p ≠ null)
    if p có con nút con trái c
      xoay p với c và hoán đổi vai trò
    else
      di chuyển p đến nút con phải
```

Thuật toán DSW (cont.)

```
createPerfectTree()
  n ← số lượng nút
  m ← 2^⌊log2 n+1⌋ − 1
  thực hiện xoay trái n − m lần bắt đầu
   từ đỉnh của cây backbone dọc theo hướng phải
  while (m > 1)
    m ← m/2
    thực hiện xoay trái m lần bắt đầu từ đỉnh
     của cây backbone dọc theo hướng phải
```

Đánh giá

Việc duy trì một cây cân bằng tối ưu đòi hỏi chi phí rất lớn. Do đó, trong thực tế các loại cây cân bằng theo từng thao tác cập nhật phổ biến hơn vì chi phí để duy trì ít tốt kém hơn

- Cây AVL
- Cây Đỏ - Đen
- Cây AA

Cây nhị phân cân bằng AVL

Lịch sử

- Cấu trúc cây cân bằng AVL do hai nhà khoa học Xô Viết G. M. Adelson-Velskii và E. M. Landis đề xuất vào năm 1962 [Sedgewick, 2002]
- Đây là một cấu trúc cây cân bằng đầu tiên

Cây nhị phân cân bằng AVL (cont.)

Định nghĩa 2

Cây cân bằng AVL là cây nhị phân tìm kiếm sao cho

- Mỗi nút p của cây đều có tính chất "chiều cao của cây con trái và cây con phải không chênh lệch quá 1"

$$\forall p : |h(p \to left) - h(p \to right)| \leq 1 \quad (1)$$

Cấu trúc dữ liệu động cho nút cây AVL

Nút của cây AVL có thể được biểu diễn bằng một lớp như sau

```
1  template <class T>
2  struct AVLNode
3  {
4      T data;
5      int key;
6      int balance;
7      AVLNode<T> *left;
8      AVLNode<T> *right;
9  };
```

Cấu trúc dữ liệu động cho nút cây AVL (cont.)

- Cấu trúc nút AVL tương tự như nút BST
- Ngoài ra tại mỗi nút có thêm thuộc tính `balance` mô tả tả trạng thái cân bằng tại nút đó
  - Nếu `balance=-1` nút bị lệch về trái; nghĩa là cây con trái cao hơn cây con phải
  - Nếu `balance=0` nút cân bằng chiều cao hai cây con bằng nhau
  - Nếu `balance=+1` nút bị lệch về phải cây con phải cao hơn cây con trái

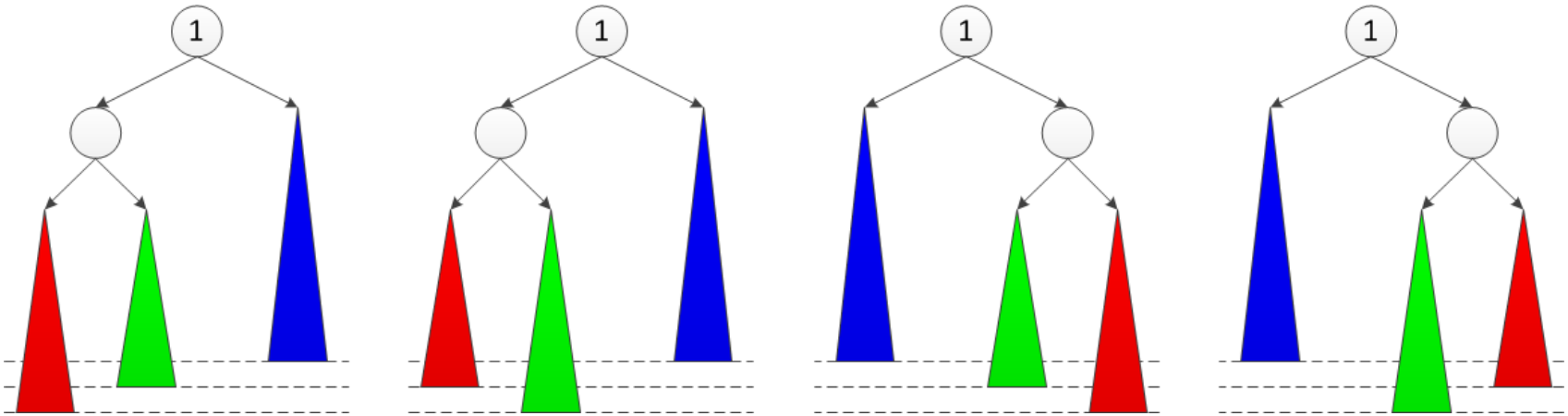
Thao tác thêm một nút

- Sử dụng kỹ thuật thêm một nút vào cây nhị phân tìm kiếm để thêm nút. Khi thêm một nút vào cây AVL có thể làm cây mất cân bằng. Do đó cần thực hiện
- Duyệt từ nút vừa thêm ngược về gốc
- Nếu tìm thấy nút p bị mất cân bằng thì tiến hành hiệu chỉnh cây tại nút này

Thuật toán cân bằng lại cây AVL

Giả sử tại nút ❶ của cây AVL xảy ra tình trạng mất cân bằng. Sẽ xảy ra một trong bốn trường hợp sau tương ứng với bốn hình vẽ:

- Trường hợp 1: tại nút ❶ cây lệch về bên trái
- Trường hợp 2: tại nút ❶ cây lệch về bên trái
- Trường hợp 3: tại nút ❶ cây lệch về bên phải
- Trường hợp 4: tại nút ❶ cây lệch về bên phải



Hình 7: Bốn trường hợp mất cân bằng tại nút ❶

Thuật toán cân bằng lại cây AVL (cont.)

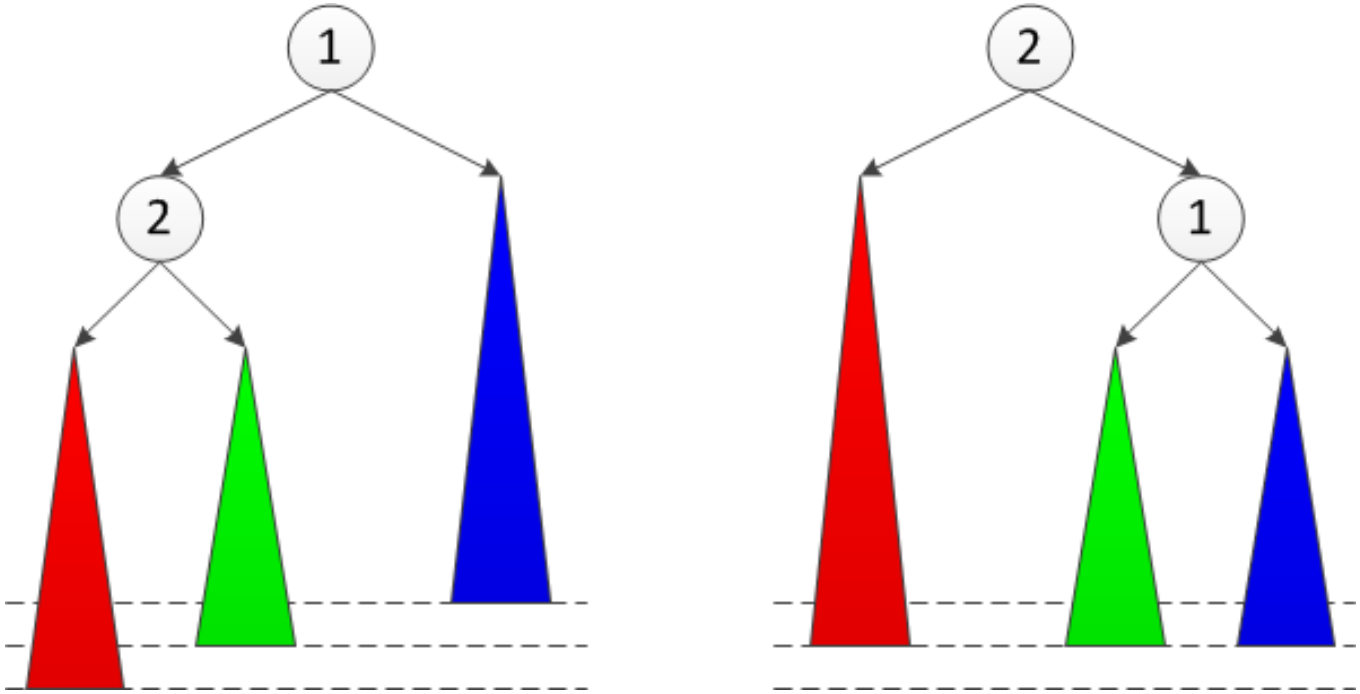
Nhận xét

Trường hợp 3 đối xứng của trường hợp 1, trường hợp 4 là đối xứng của trường hợp 2

Thuật toán cân bằng lại cây AVL (cont.)

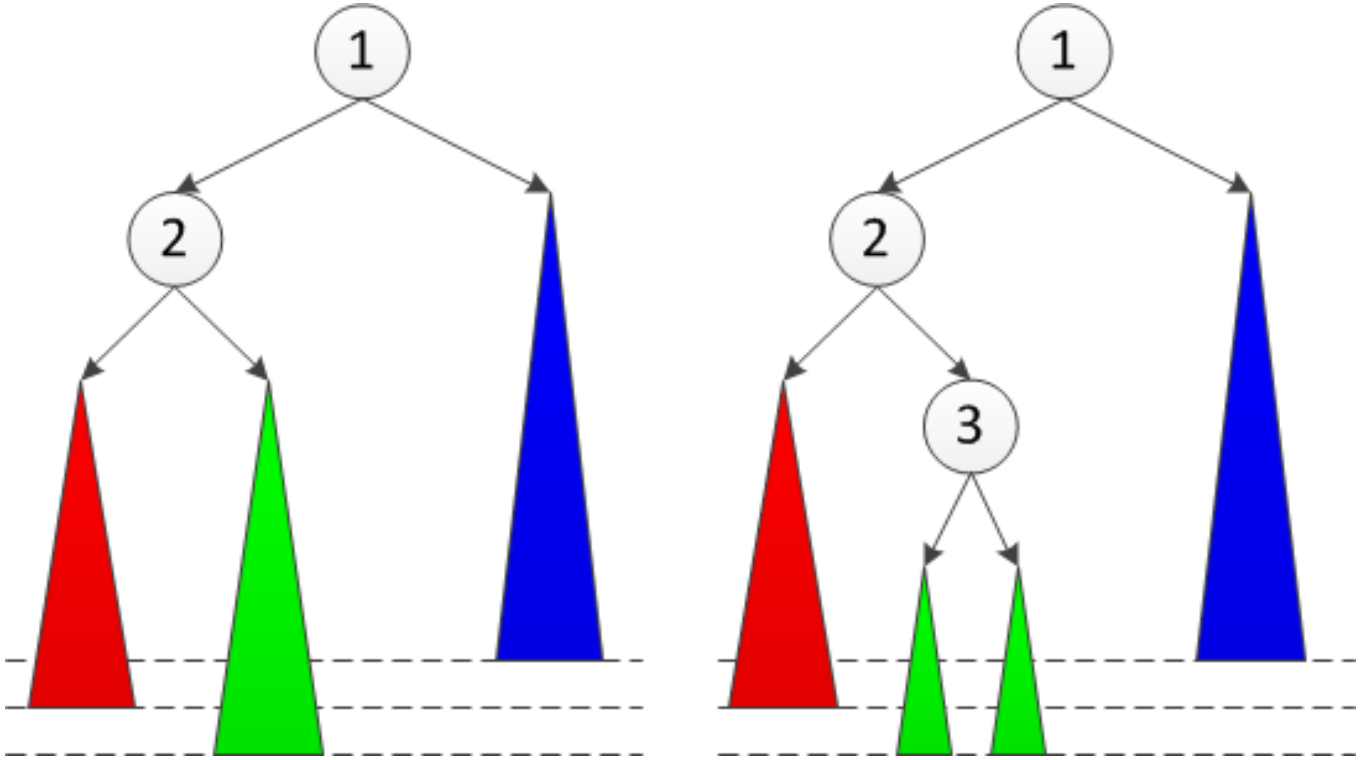
Hiệu chỉnh cân bằng cho trường hợp 1

- Thực hiện xoay nút ❶ và nút ❷



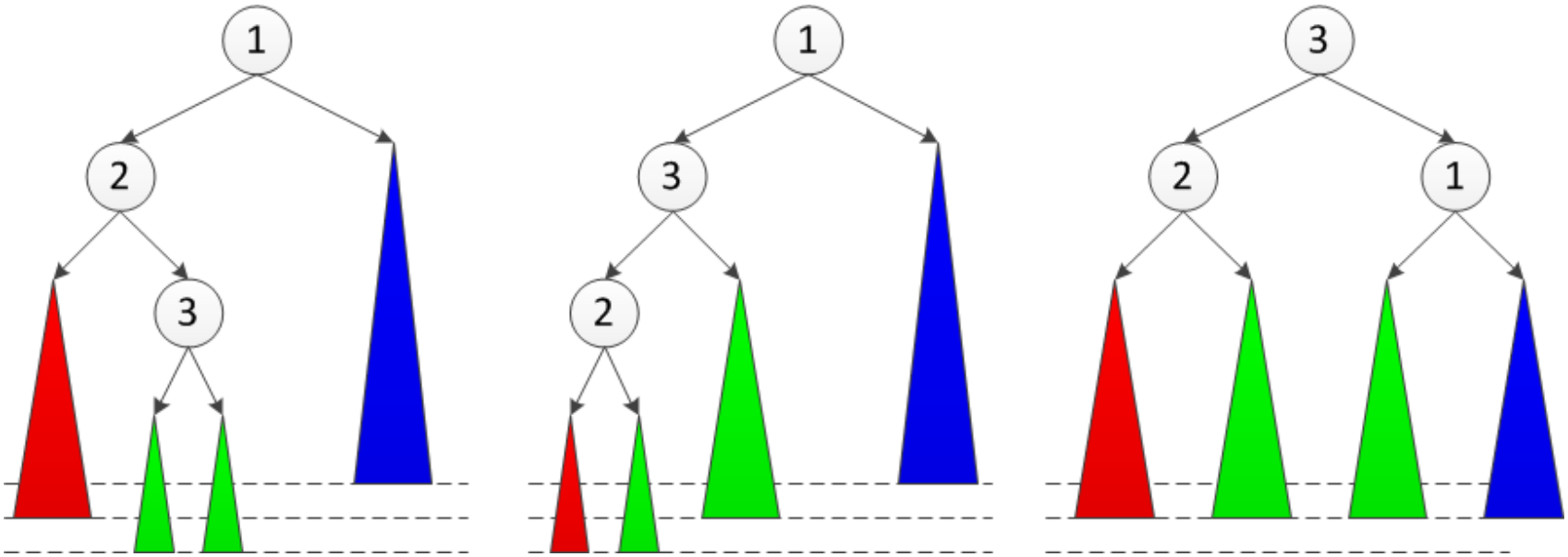
Thuật toán cân bằng lại cây AVL (cont.)

Hiệu chỉnh cân bằng cho trường hợp 2

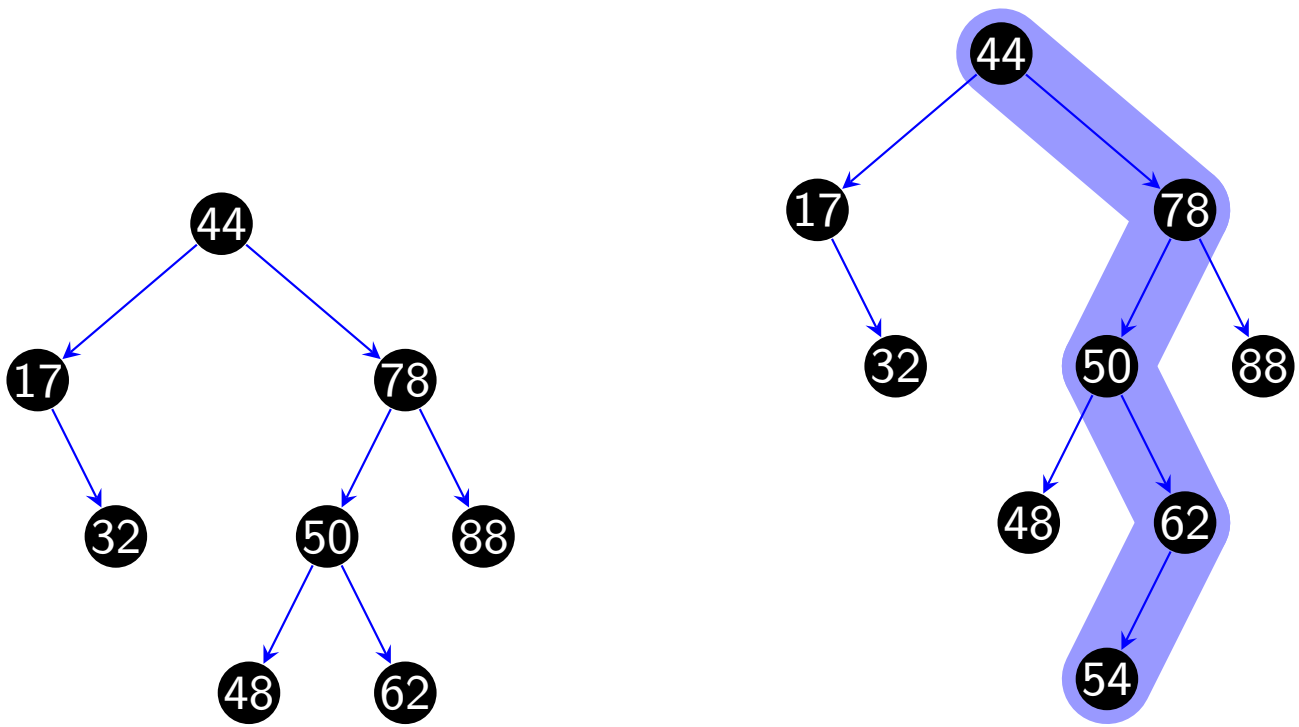


Thuật toán cân bằng lại cây AVL (cont.)

- Thực hiện xoay nút nút ❷ và nút ❸
- Sau đó, thực hiện xoay nút nút ❶ và nút ❸



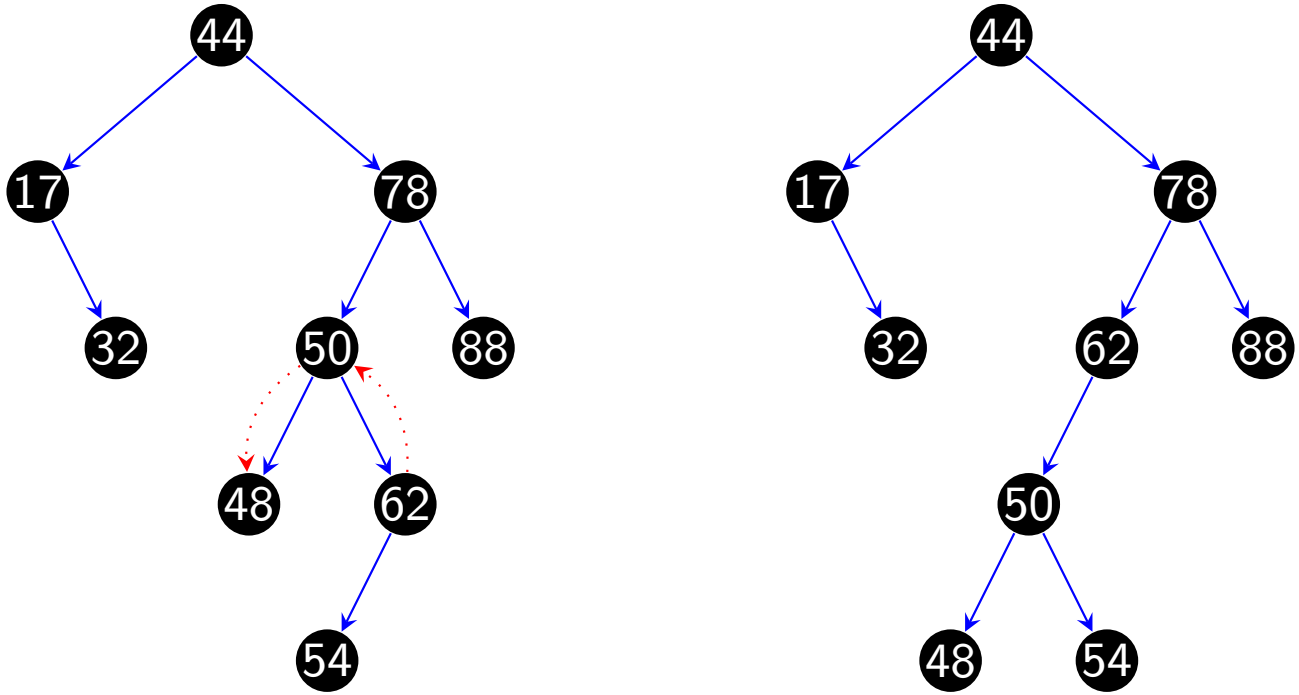
Minh họa thêm một nút



(a) trước khi thêm **(b)** sau khi thêm

Hình 8: Thêm nút 54 vào cây AVL

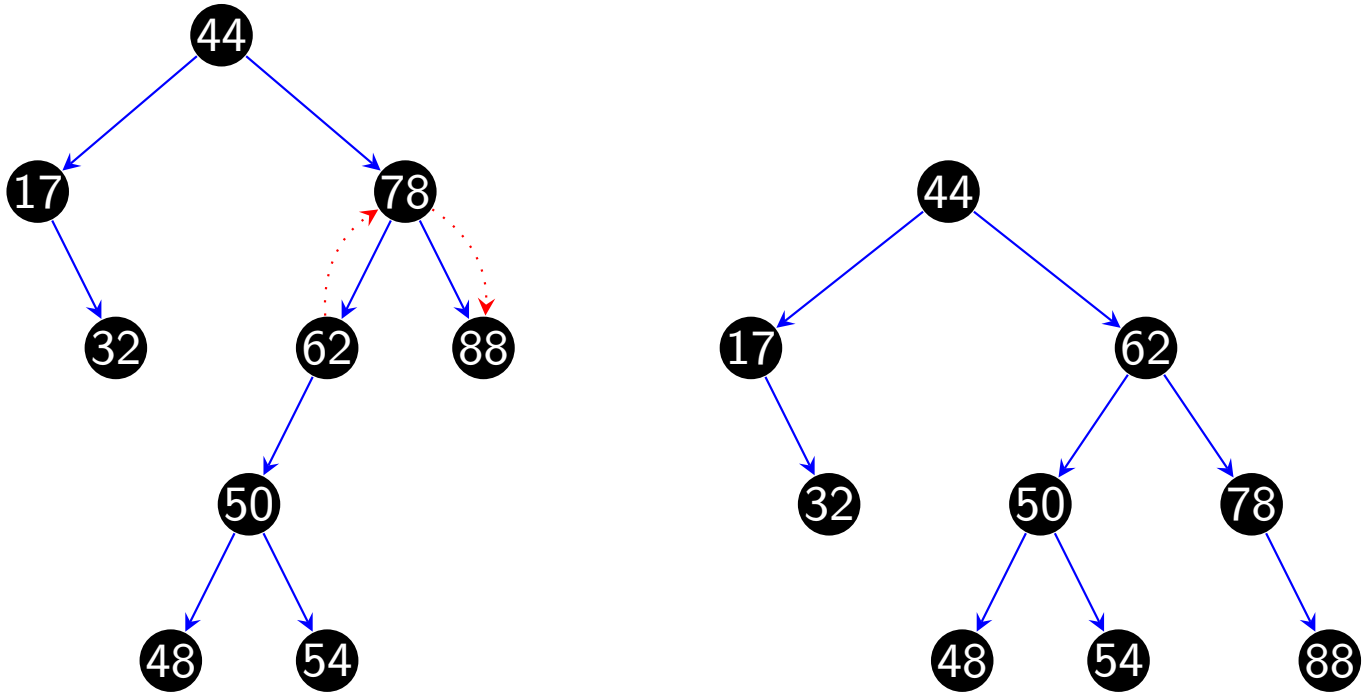
Minh họa thêm một nút (cont.)



(a) trước khi xoay **(b)** sau khi xoay

Hình 9: Mất cân bằng tại nút 78 → xoay nút 50 và 62

Minh họa thêm một nút (cont.)



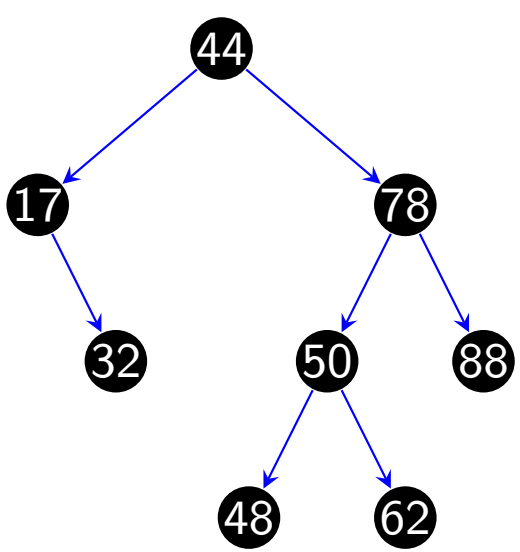
(a) trước khi xoay **(b)** sau khi xoay

Hình 10: Mất cân bằng tại nút 78 → xoay nút 62 và 78

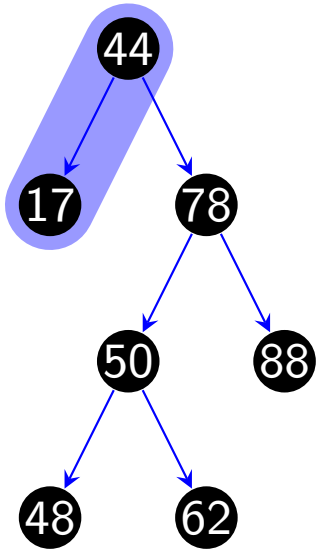
Thao tác xóa một nút

- Sử dụng kỹ thuật xóa một nút của cây nhị phân tìm kiếm. Khi xóa một nút của cây AVL có thể làm cây mất cân bằng. Vậy cần phải cân bằng lại
- Duyệt từ nút bị xóa trở về gốc
- Tìm xem có nút p nào bị mất cân bằng nếu có thì hiệu chỉnh lại

Minh họa xóa một nút



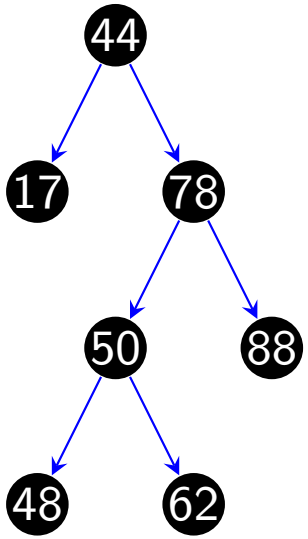
(a) trước khi xóa



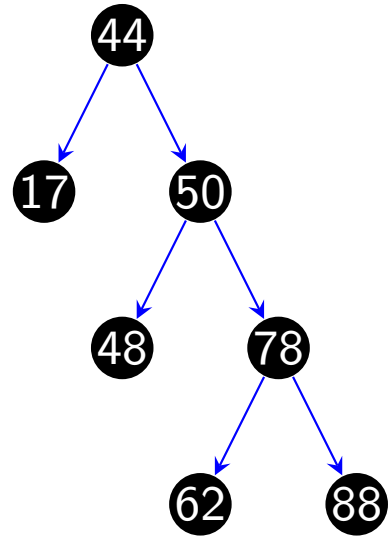
(b) sau khi xóa

Hình 11: Xóa nút 32 khỏi cây AVL

Minh họa xóa một nút (cont.)



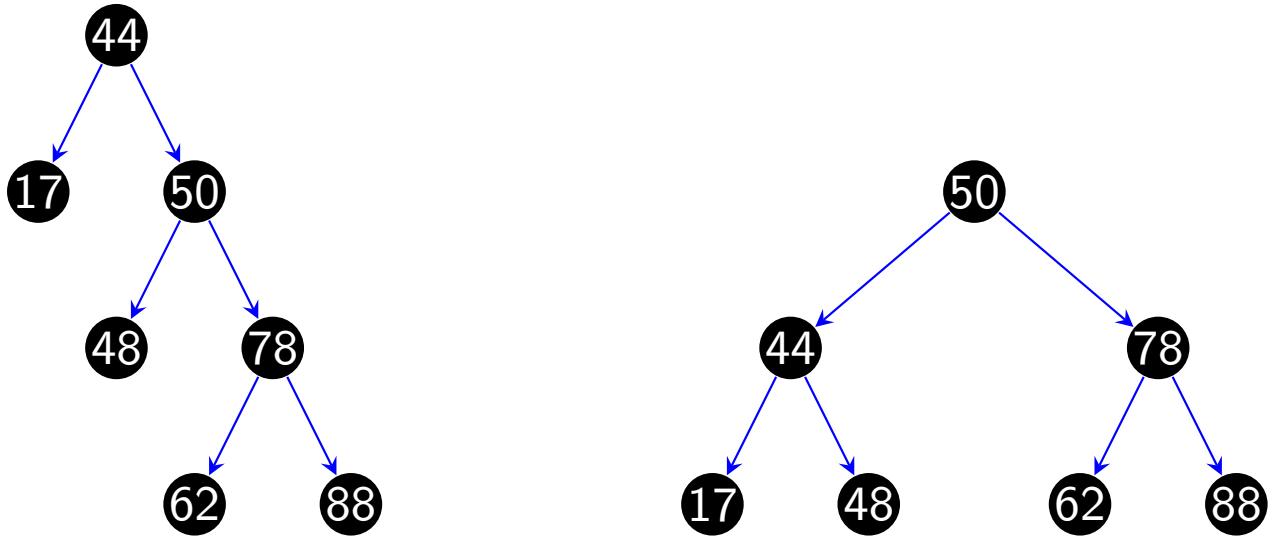
(a) trước khi xoay



(b) sau khi xoay

Hình 12: Cây mất cân bằng tại nút 44 $\rightarrow$ xoay 50 với 78

Minh họa xóa một nút (cont.)



(a) trước khi xoay **(b)** sau khi xoay

Hình 13: Cây mất cân bằng tại nút 44 $\rightarrow$ xoay 50 với 44

Bài luyện tập

Ví dụ 3

Hãy xây dựng cây AVL từ dãy {4,3,5,1,2,7,9,8,15,11,12,16}

- Xóa các nút 8, 16
- Thêm các nút 6, 17

Định lý về chiều cao cây AVL

Định lý 2

Một cây AVL có chiều cao là h thì sẽ có ít nhất $F_{h+3} - 1$ nút, với F_i là số Fibonacci thứ i.

Chứng minh

- Gọi S_h là kích cỡ của cây AVL nhỏ nhất với chiều cao h
- Rõ ràng, $S_0 = 1$ và $S_1 = 2$
- Hơn nữa, $S_h = S_{h-1} + S_{h-2} + 1$
- Do đó, $S_h = F_{h+3} - 1$

■

Định lý về chiều cao cây AVL (cont.)

Định lý 3

Hãy chứng minh điều sau

$$F_{h+3} - 1 \geq \left(\frac{3}{2}\right)^h$$

Chứng minh

Sử dụng công thức số Fibonacci

$$F_n = \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right]$$

■

Định lý về chiều cao cây AVL (cont.)

Định lý 4

Chiều cao của cây AVL gần với chiều cao tối ưu

$$h_{AVL} \leq 1.44 \log_2 (n+1) \tag{2}$$

Chứng minh

Sinh viên hãy tự chứng minh ■

Đánh giá cây AVL

Phân tích chi phí thực hiện theo n (số lượng nút của cây)

	xấu nhất	**trung bình**	**tốt nhất**
tìm một phần tử	?	?	?
thêm một phần tử	?	?	?
xóa một phần tử	?	?	?

Phân tích chi phí bộ nhớ theo n (số lượng nút của cây)

Tài liệu tham khảo


Sedgewick, R. (2002).
Algorithms in Java, Parts 1-4, volume 1.
Addison-Wesley Professional.