BALANCED TREES

Nhóm 6:

1. Dương Hiển Lê Hoàng, MSSV: 20127503

2. Trần Nhật Tân, MSSV: 20127322

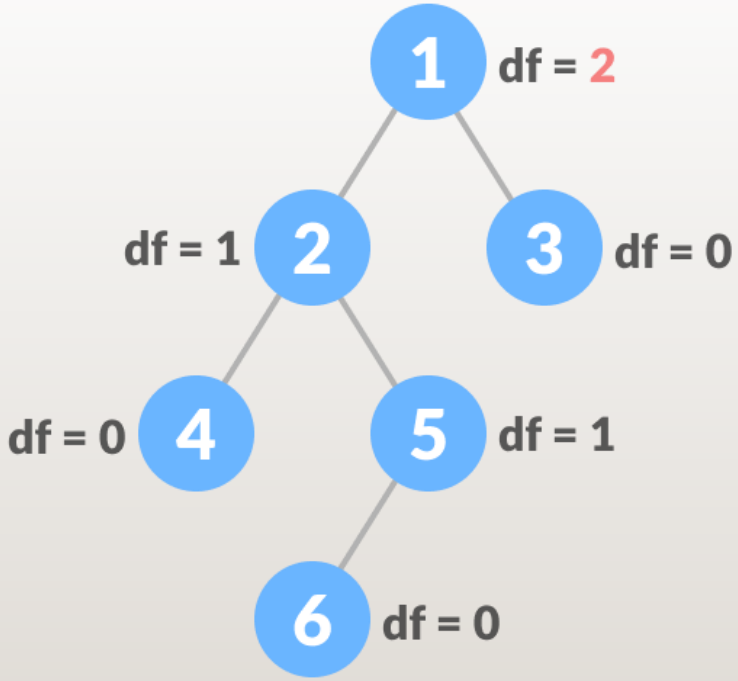
3. Nguyễn Văn Đạt, MSSV; 20127132

Balanced Trees

Định nghĩa

-Là một cấu trúc dữ liệu dạng cây tự cân bằng nhằm duy trì dữ liệu đã được sắp xếp và cho phép tìm kiếm, truy cập tuần tự, chèn và xóa theo thời gian logarit.

-Là một dạng tổng quát của cây tìm kiếm nhị phân

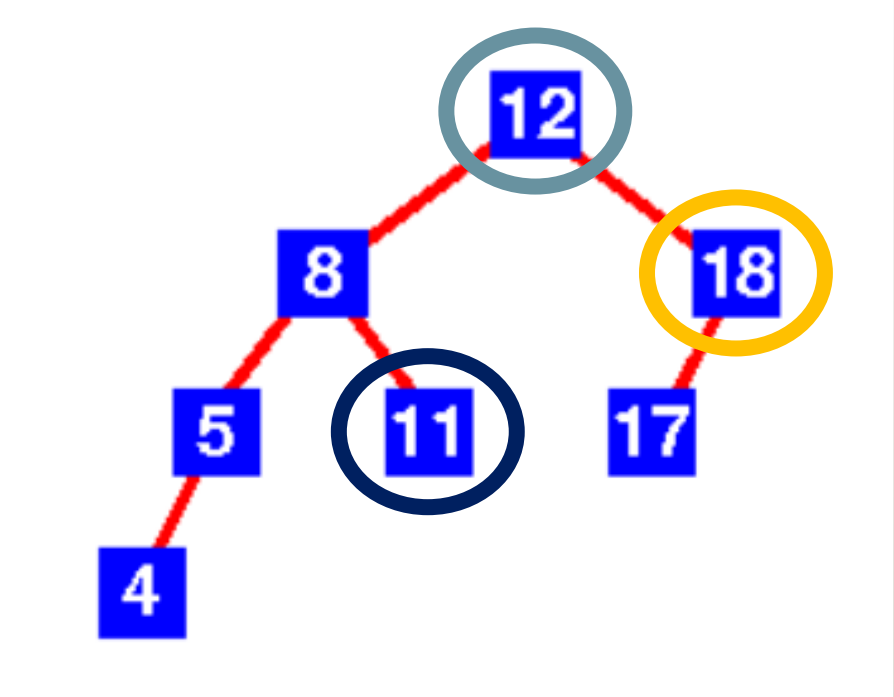


Balanced Trees

Định nghĩa

-Các nút:

- Nút gốc
- Nút lá
- Nút nội bộ (nút nhánh)



Balanced Trees

Độ cao

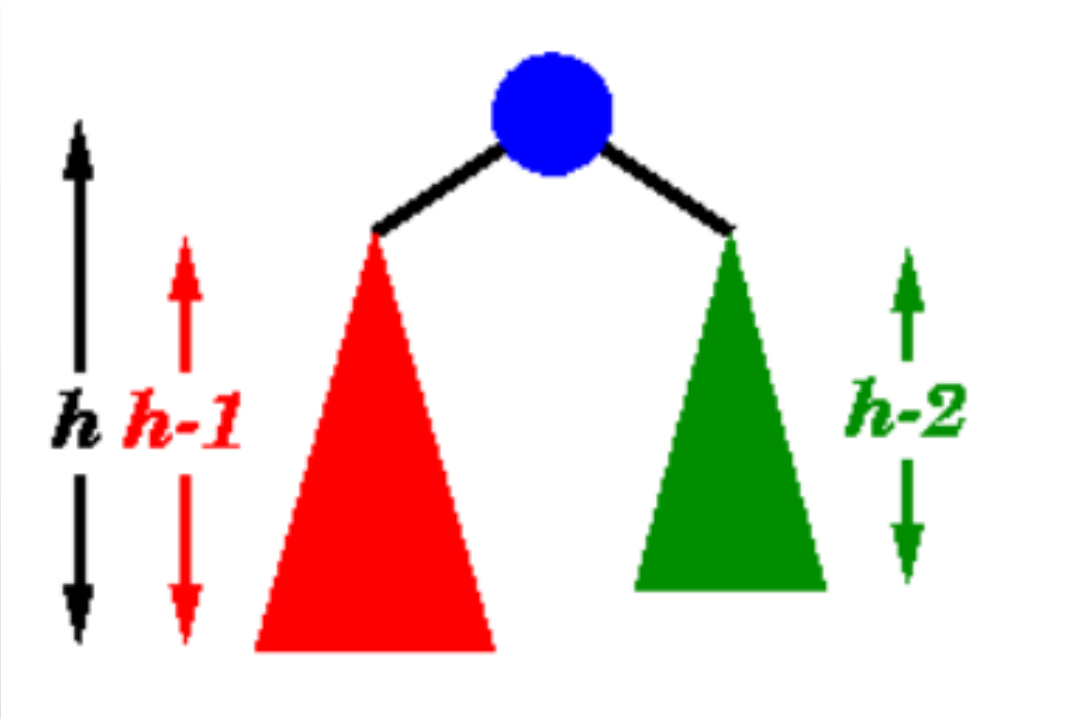
-Độ cao của cây ảnh hưởng rất lớn đến thời gian chạy của chương trình

-Độ cao tốt nhất: $\Theta(\ln(n))$

-Độ cao xấu nhất: $\Theta(n)$

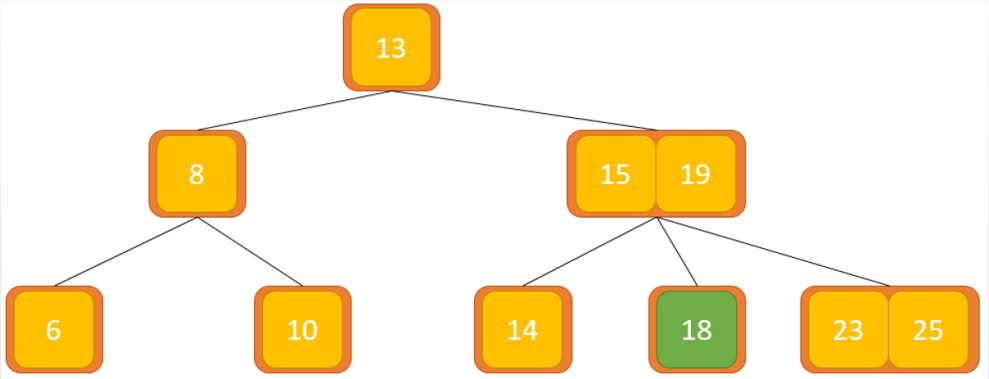
-Độ cao trung bình: $\Theta(\ln(n))$

-Tuy nhiên trong khi xóa và thêm node độ cao trung bình có thể tăng lên: $\Theta\ (\sqrt{n})$

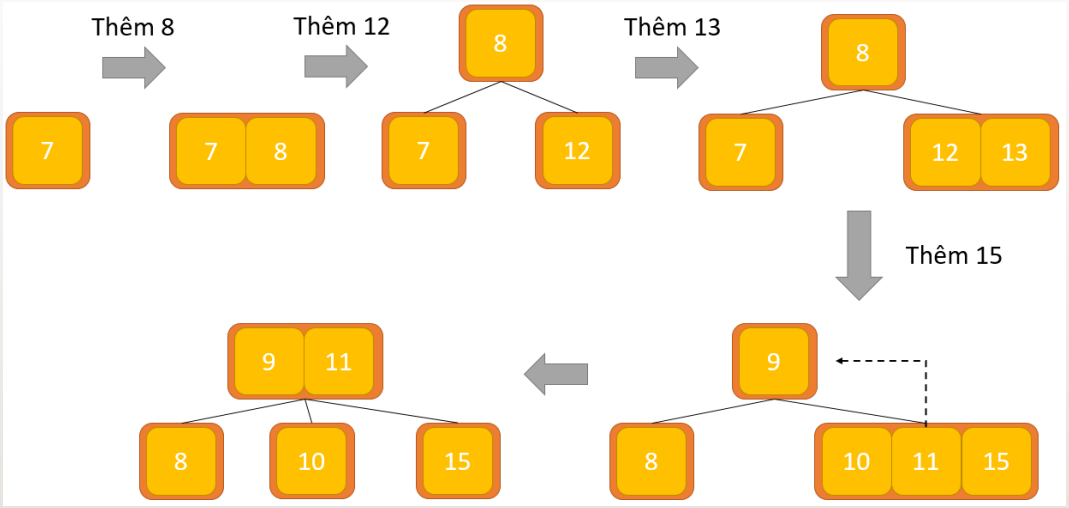


Balanced Trees

Các thuật toán



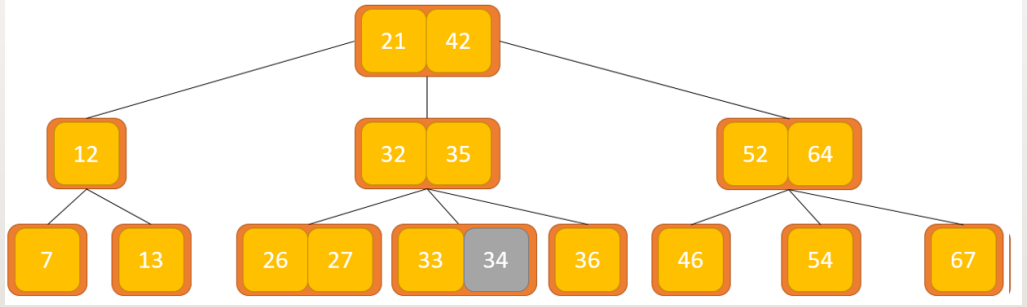
Tìm Kiếm



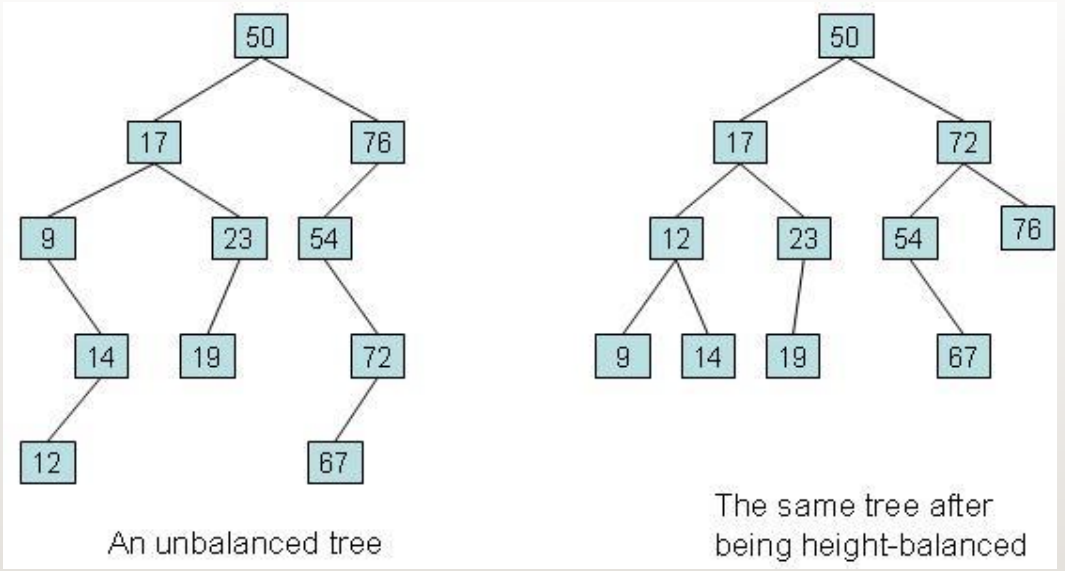
Thêm vào

Balanced Trees

Các thuật toán



Xóa

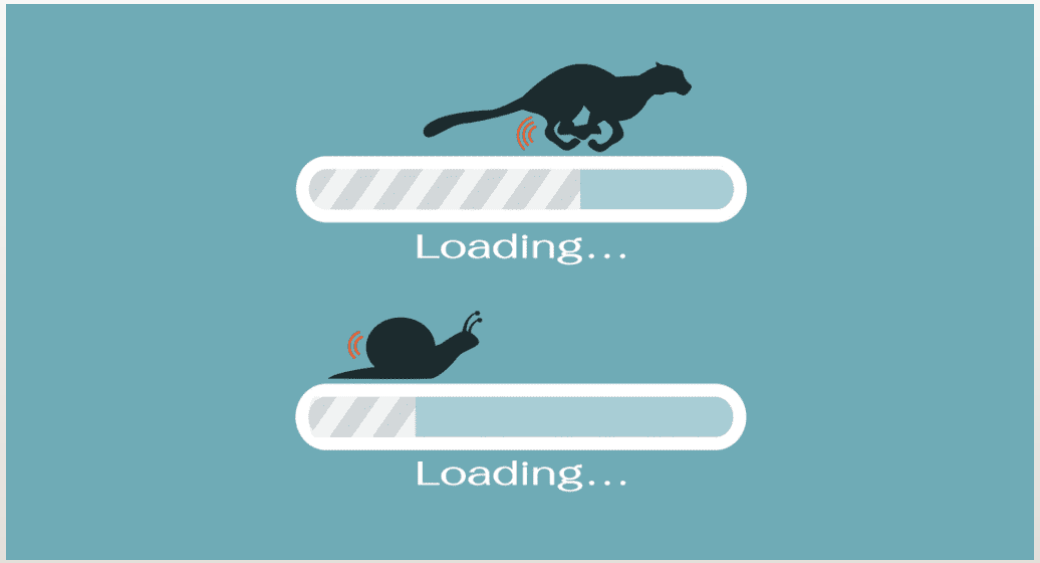


Tái cân bằng

Balanced Trees

Ưu điểm

- Giữ các khóa theo thứ tự được sắp xếp để duyệt tuần tự
- Sử dụng chỉ mục phân cấp để giảm thiểu số lần đọc đĩa
- Cho phép thực hiện nhanh các thao tác chèn và xóa
- Cho phép tìm các phần tử gần nhất



Balanced Trees

Yêu Cầu

- Để thời gian chạy chương trình không bao giờ vượt quá $\omega(\ln(n))$ thì phải duy trì độ cao không vượt quá $\Theta(\ln(n))$

- Ta sẽ có một số ý tưởng

Balanced Trees

Yêu Cầu

Đối với một perfect tree, tất cả các nút có cùng số lượng con cháu ở mỗi bên

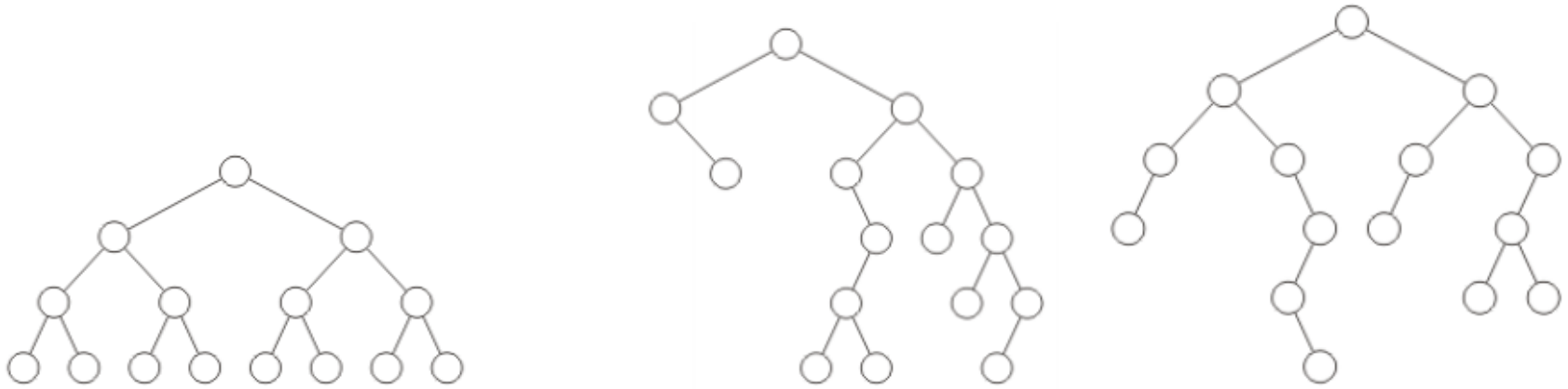


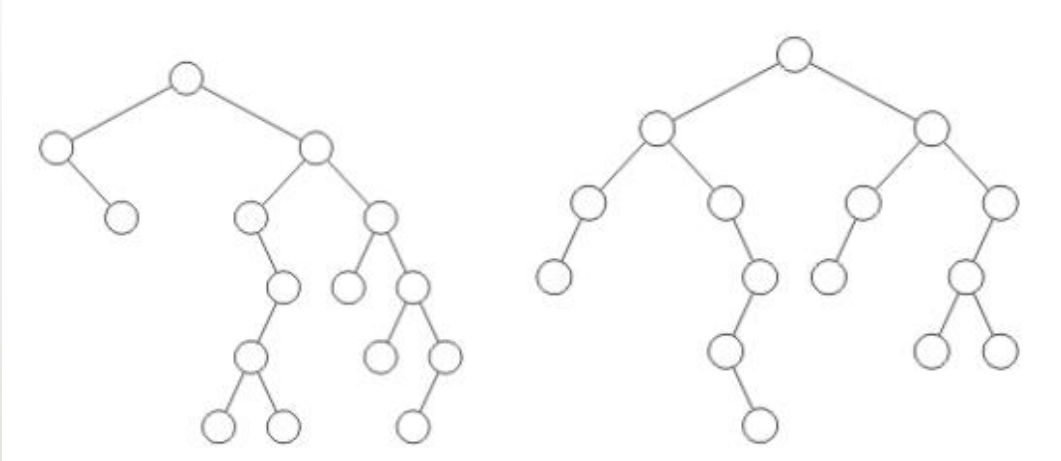
Figure 1. Three binary trees with $n = 15$ nodes.

Balanced Trees

Yêu Cầu

Nhìn vào cây thứ hai và thứ ba, không cân đối ở gốc, nhưng ở cây thứ hai, cây phụ bên phải ít nhất về số lượng (con bên phải của gốc có 5 node con ở cây phụ bên trái và 6 node con ở cây phụ bên phải).

Ở cây thứ ba, gốc có 7 node con ở cả hai cây phụ; tuy nhiên, các cây con trong số này về cơ bản là danh sách được liên kết.

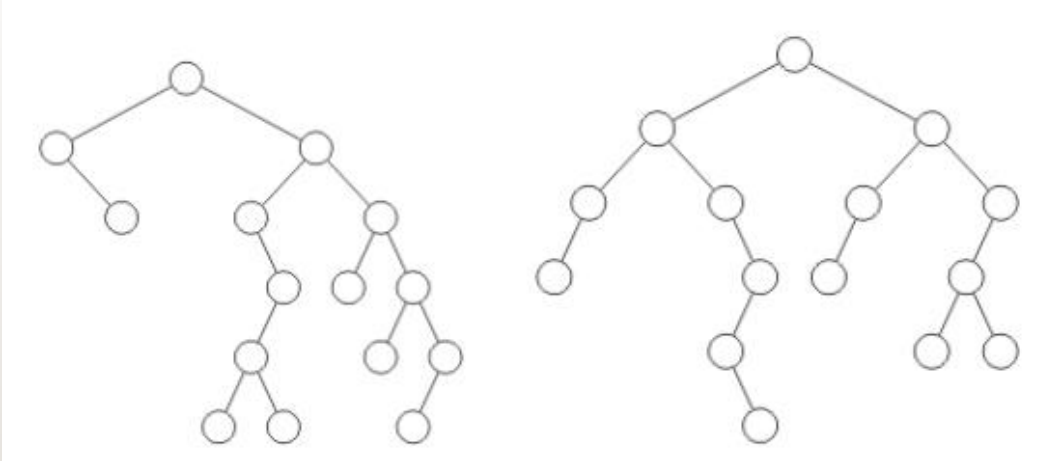


Balanced Trees

Yêu Cầu

Do đó, cân bằng phải là một khái niệm được định nghĩa trong toàn bộ cây và chúng ta sẽ xem xét hai định nghĩa có thể có:

1. Cân bằng chiều cao: xác định sự cân bằng bằng cách so sánh chiều cao của hai cây phụ.



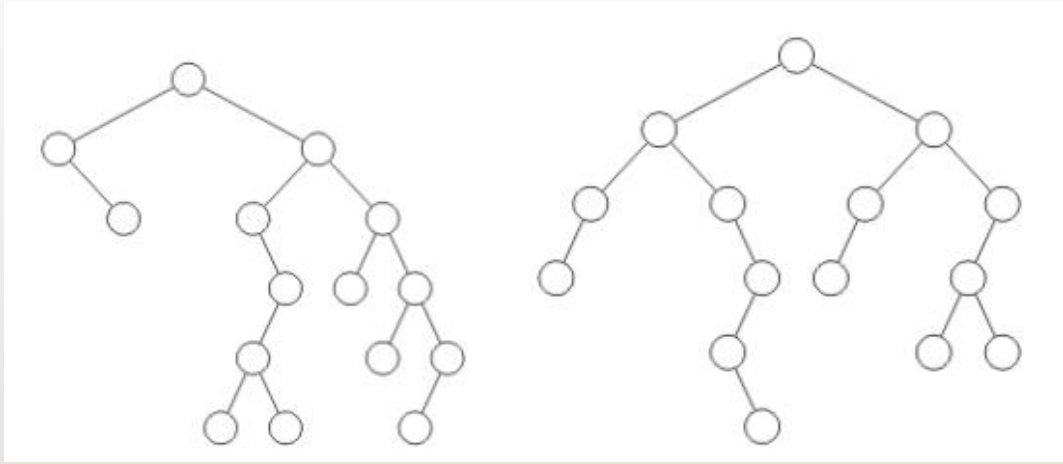
Balanced Trees

Yêu Cầu

Do đó, cân bằng phải là một khái niệm được định nghĩa trong toàn bộ cây và chúng ta sẽ xem xét hai định nghĩa có thể có:

2. Cân bằng null-path length : xác định số dư bằng cách so sánh null-path length của mỗi cây trong số hai cây con.

2. Cân bằng null-path length : xác định số dư bằng cách so sánh null-path length của mỗi cây trong số hai cây con.

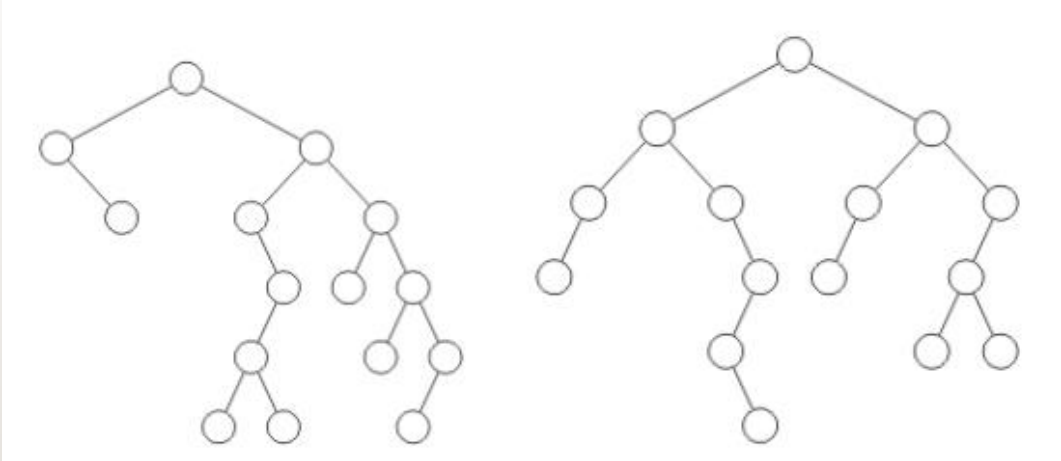


Balanced Trees

Yêu Cầu

Do đó, cân bằng phải là một khái niệm được định nghĩa trong toàn bộ cây và chúng ta sẽ xem xét hai định nghĩa có thể có:

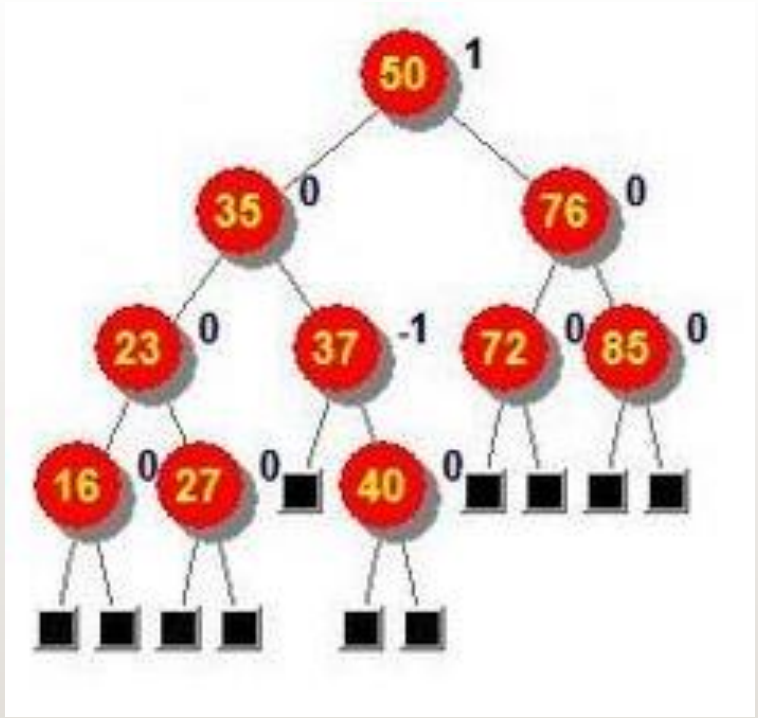
3. Cân bằng số node: xác định sự cân bằng bằng cách so sánh số lượng node trống trong mỗi cây trong số hai cây con. Khi chúng ta tạo ra một định nghĩa về sự cân bằng, chúng ta cũng sẽ chứng minh bằng toán học rằng bất kỳ cây nào có thuộc tính đó đều có chiều cao $\Theta(\ln(n))$.



AVL Trees

-Là cây nhị phân cân bằng

| độ cao node trái – độ cao node phải | ≤ 1

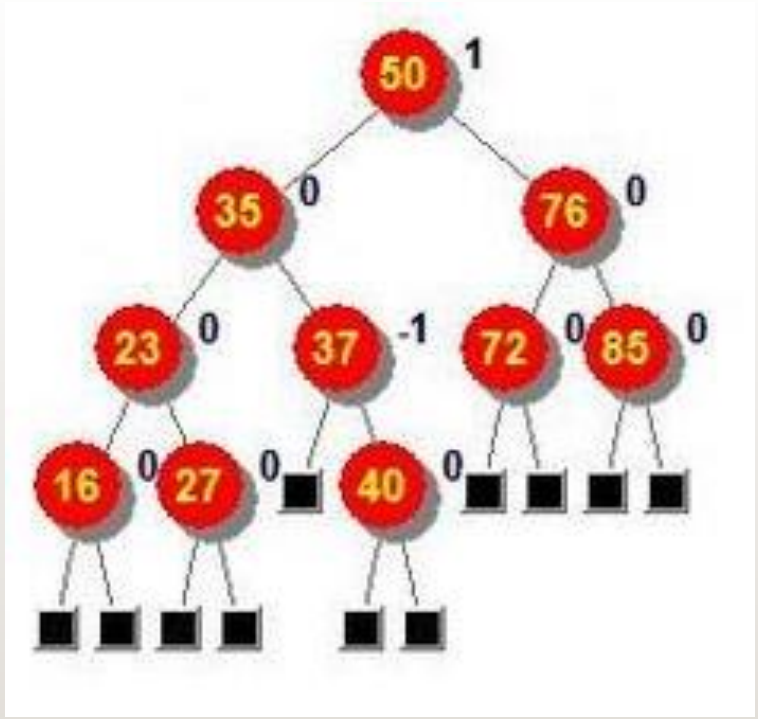


AVL Trees

-Để xác định chiều cao tối đa mà cây AVL với N nút có thể có, thay vào đó chúng ta có thể hỏi số lượng nút tối thiểu mà một cây AVL chiều cao h có thể có (gọi là cây AVL F_h).

$$|F_h| = |F_{h-1}| + |F_{h-2}| + 1$$

khi $|F_0| = 1$ and $|F_1| = 2$



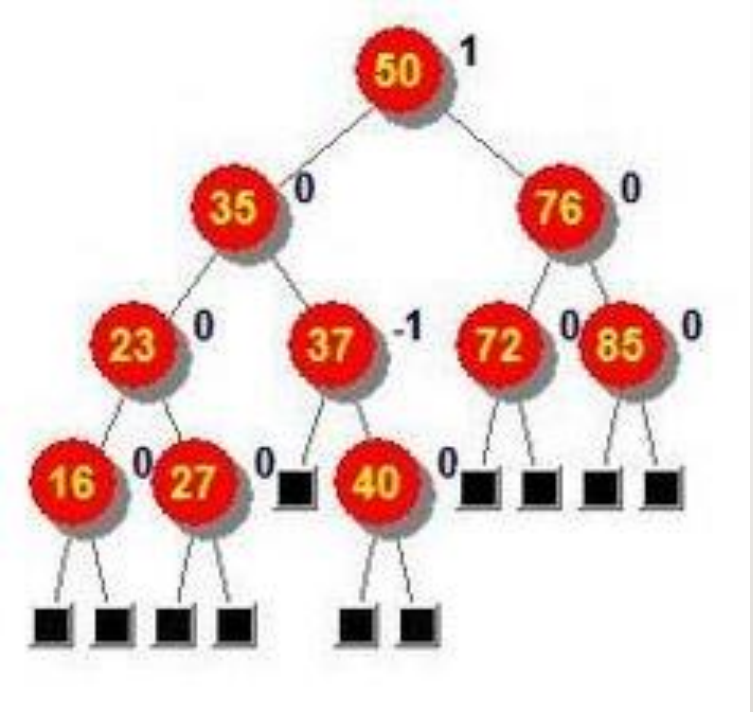
AVL Trees

Suy ra:

$$|F_h| + 1 \approx \frac{1}{\sqrt{5}} \left[\frac{1+\sqrt{5}}{2} \right]^{h+2}$$

Chiều cao của cây AVL trong trường hợp xấu nhất là:

$$h \approx 1.44 \log_2 |F_h| = 1.44 \log_2 N$$

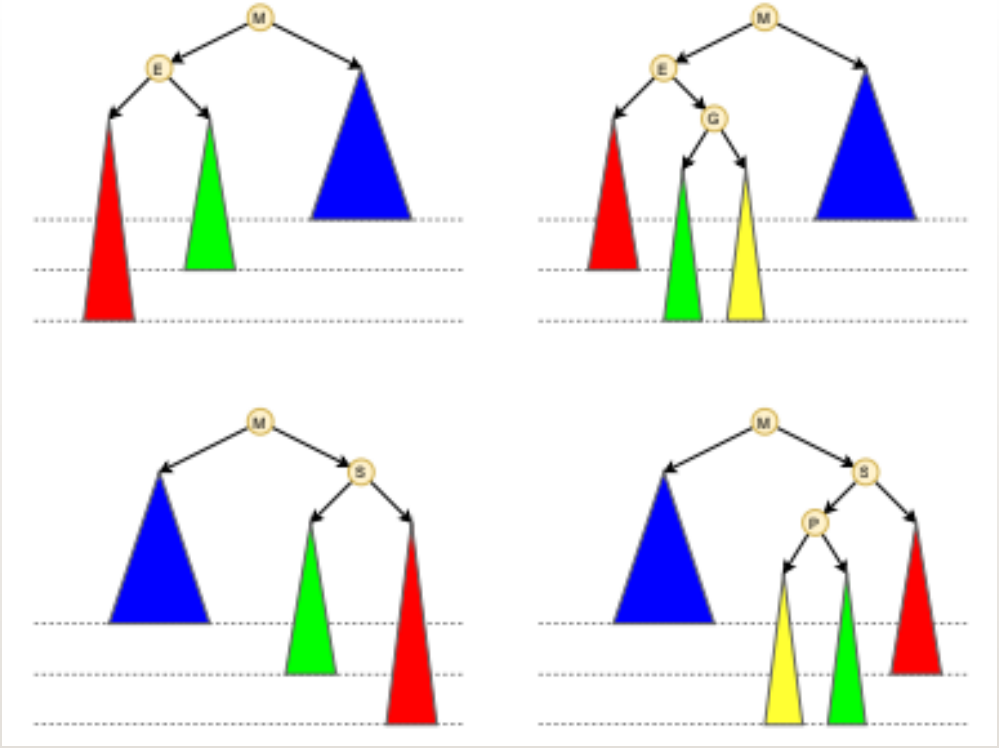


AVL Trees

Tái cân bằng:

-Khi thêm hoặc xóa, cây sẽ không còn cân bằng, độ cao của node có số dư không còn thỏa mãn điều kiện của cây AVL.

-Có 4 trường hợp mất cân bằng: Trái trái, Phải phải, Phải trái, Trái phải.

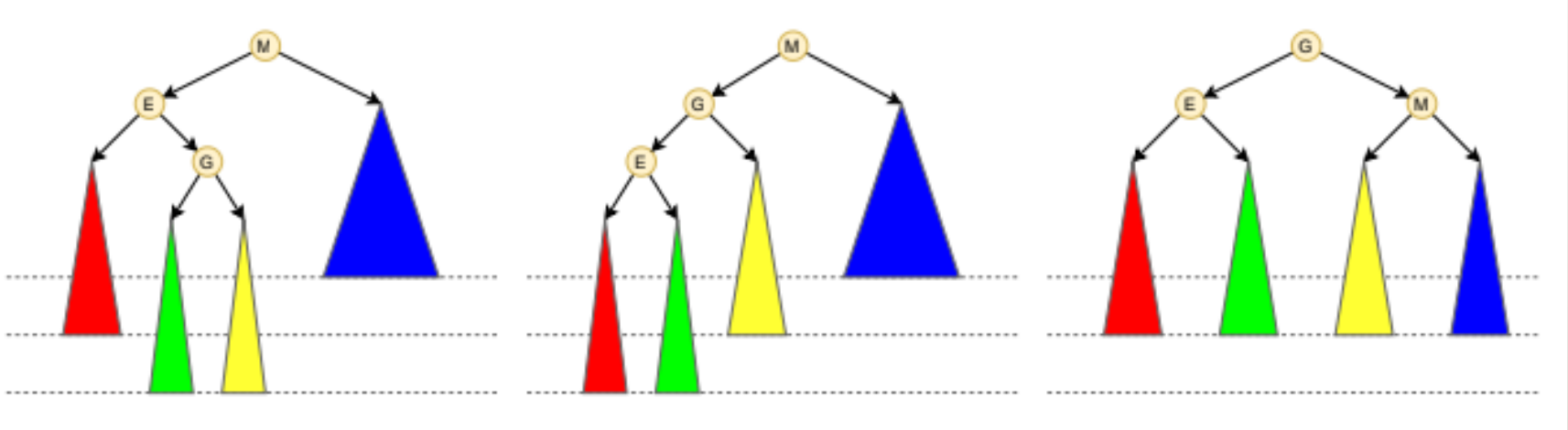


AVL Trees

Tái cân bằng:

-Trường hợp 1: Mất cân bằng Trái phải

(xoay trái phải)

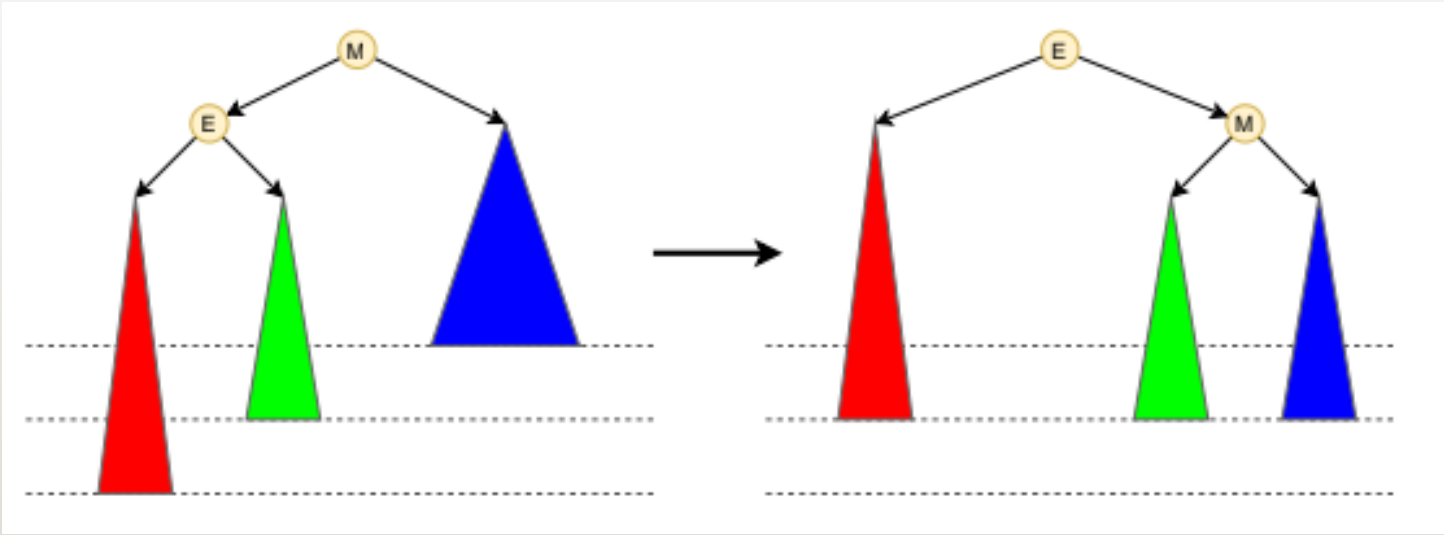


AVL Trees

Tái cân bằng:

-Trường hợp 2: Mất cân bằng Trái trái (xoay phải phải)

Các trường hợp còn lại thì ngược lại.



Red-Black Trees

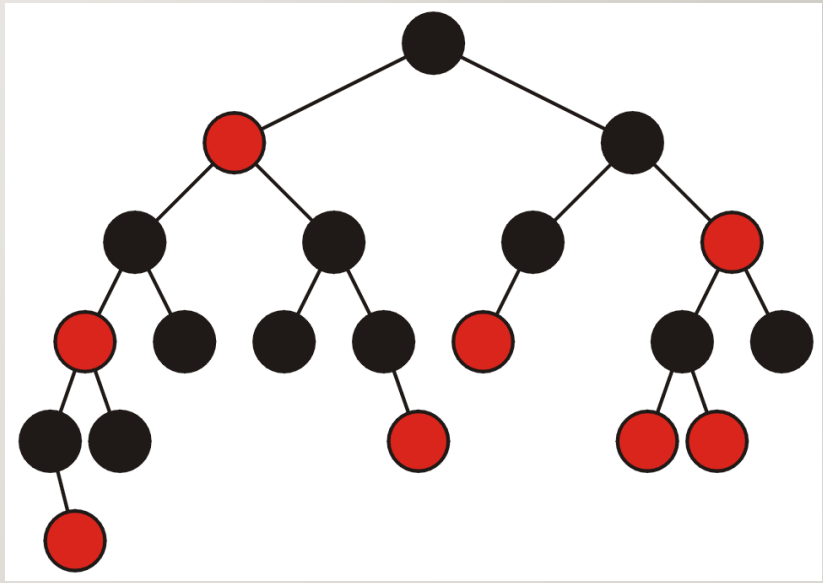
Red-black trees duy trì sự cân bằng bằng cách: Tất cả các nút được tô màu đỏ hoặc đen (0 hoặc 1)

Yêu cầu:

- Gốc phải có màu đen

- Tất cả các nút con của nút đỏ phải có màu đen

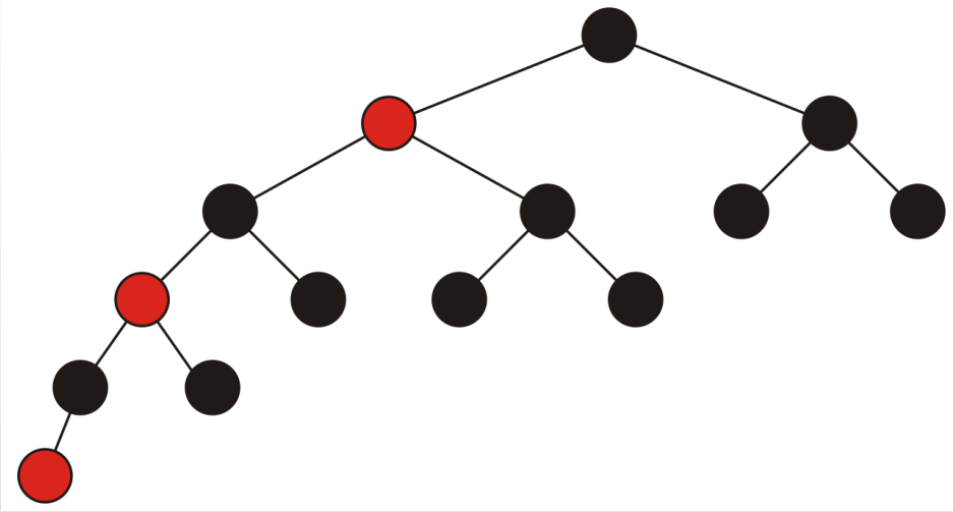
- Mọi đường đi từ gốc đến nút trống phải có cùng số nút đen



Red-Black Trees

Chiều cao của BST đỏ đen có N nút không quá $2\log_2 N$. Có nghĩa là rằng chiều cao của cây Red-Black trong trường hợp xấu nhất là:

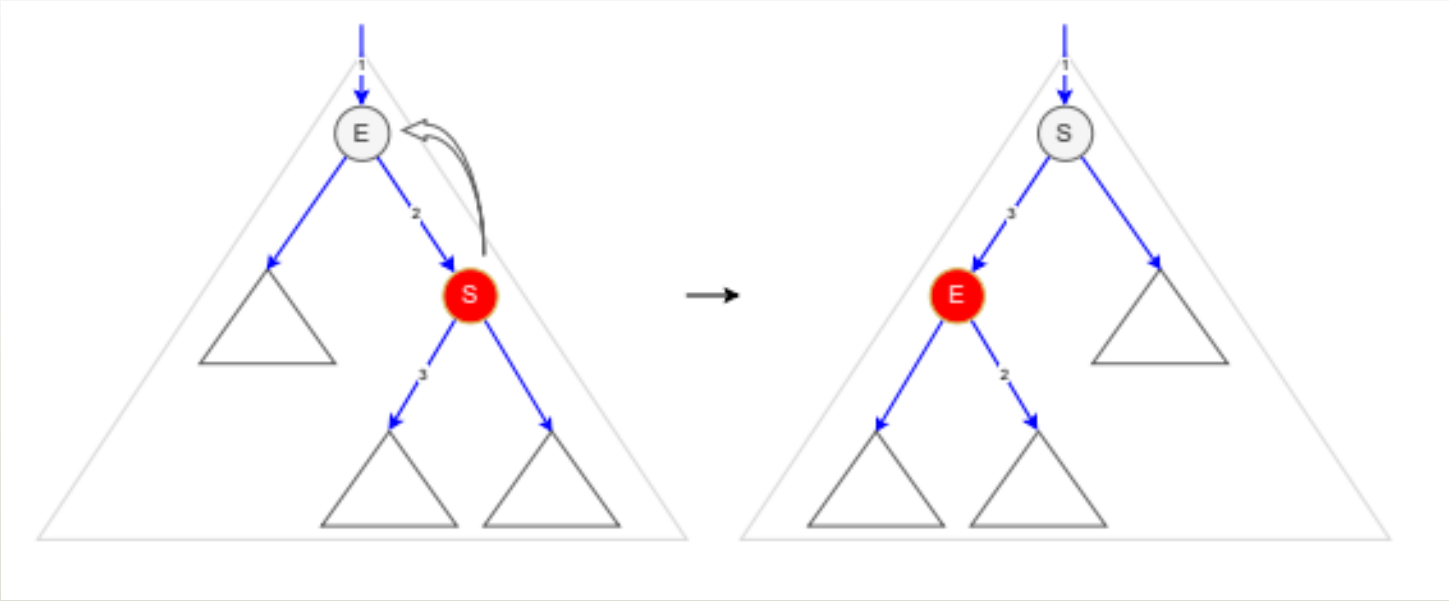
$$h \leq 2 \log_2 N$$



Red-Black Trees

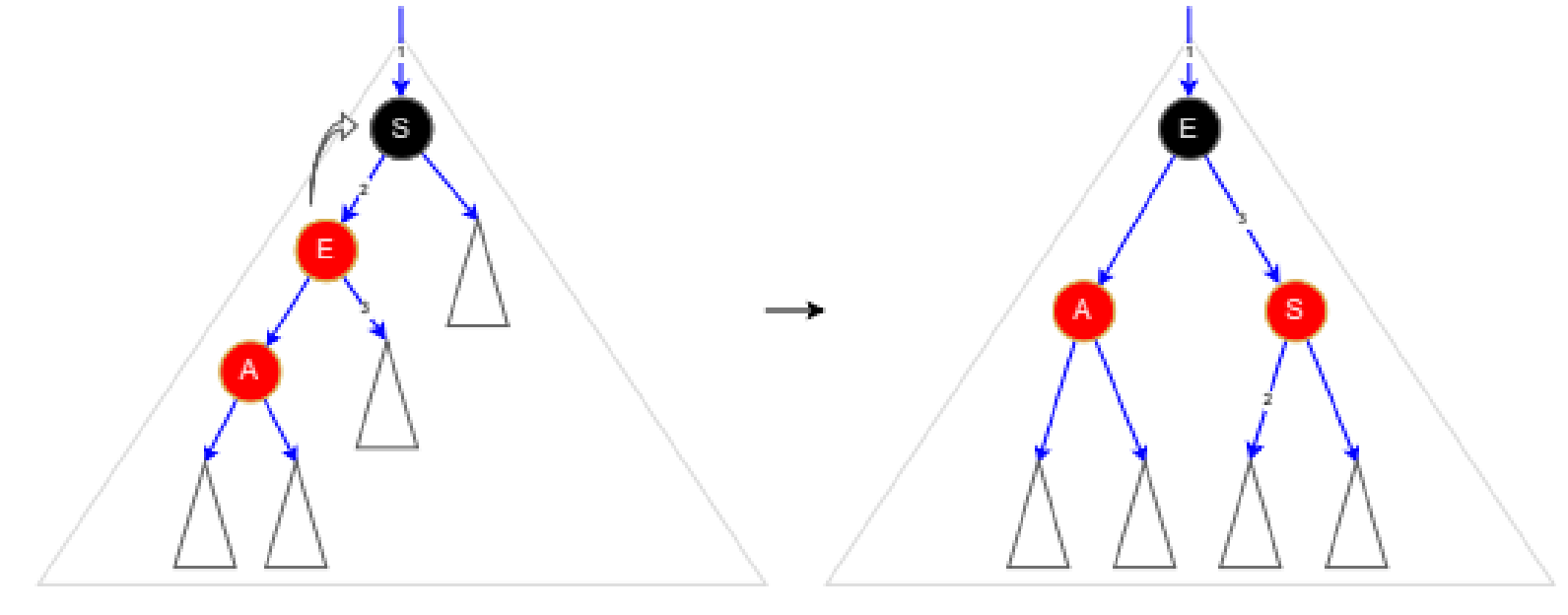
Trường hợp 1:

Xoay trái để đưa nút đỏ phải sang nút đỏ trái



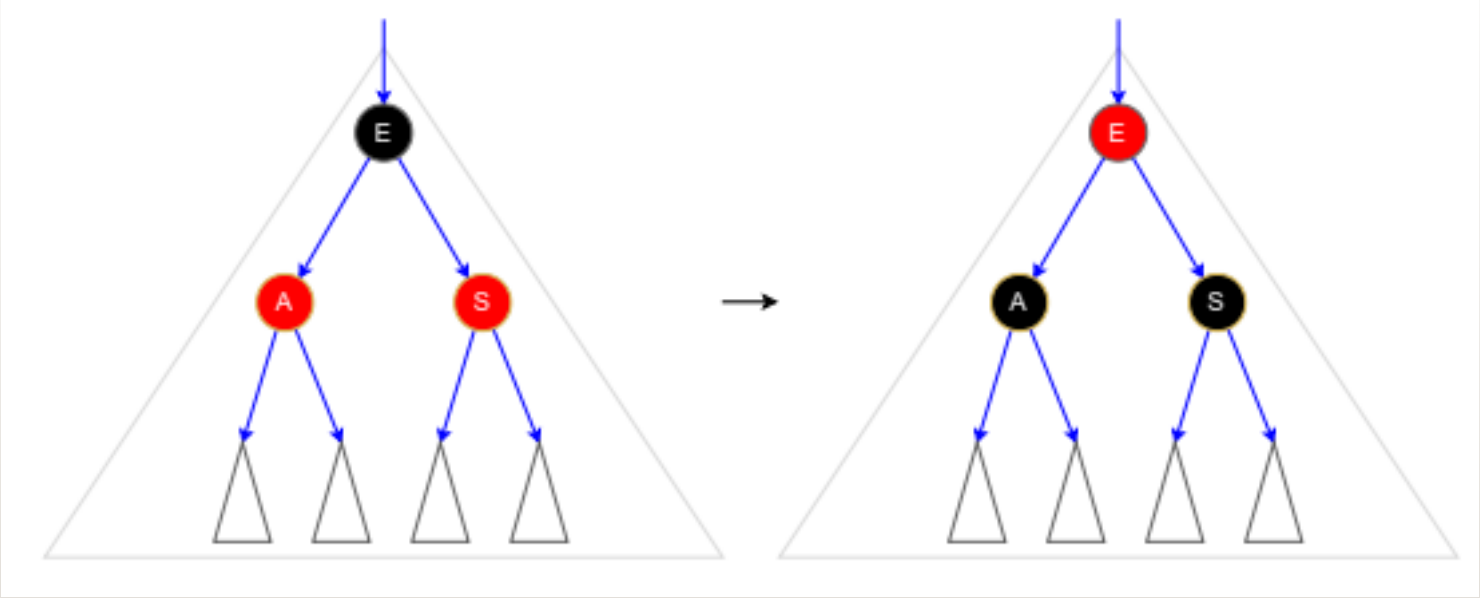
Red-Black Trees

Trường hợp 2: Xoay phải



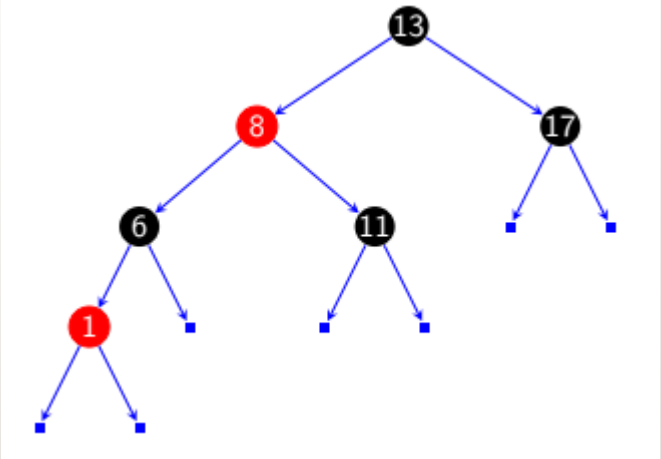
Red-Black Trees

Trường hợp 3: Xoay màu

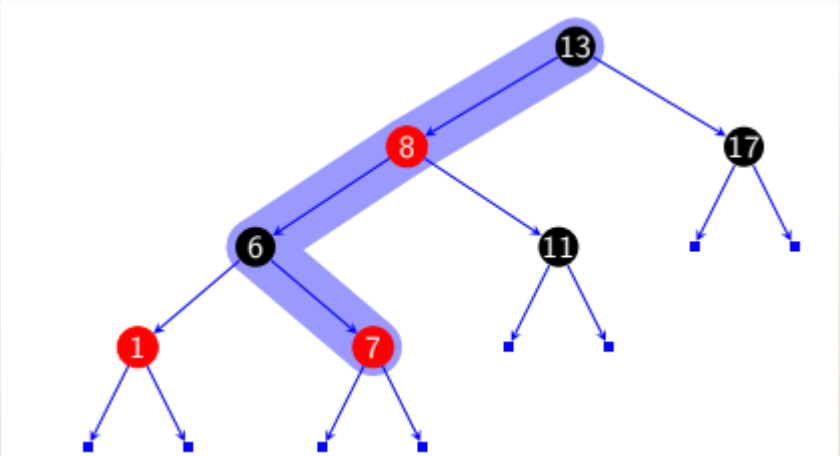


Red-Black Trees

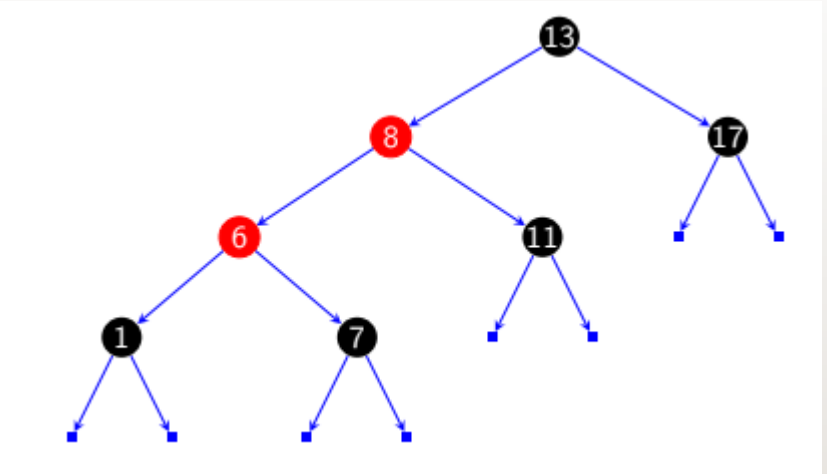
Thuật toán: Thêm Node



Cây ban đầu



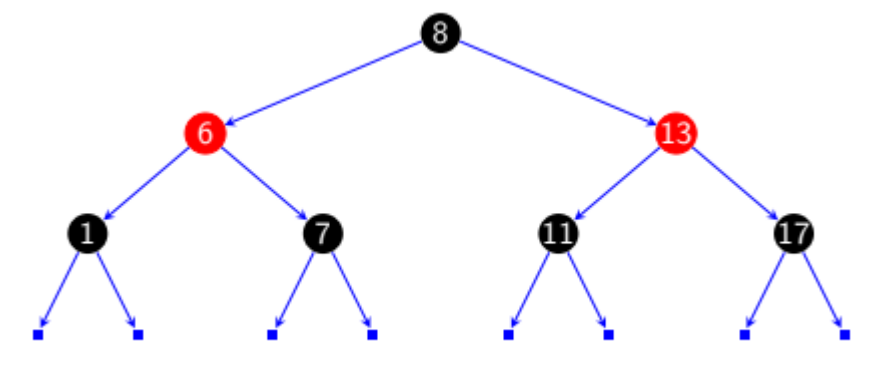
Thêm 7 vào cây



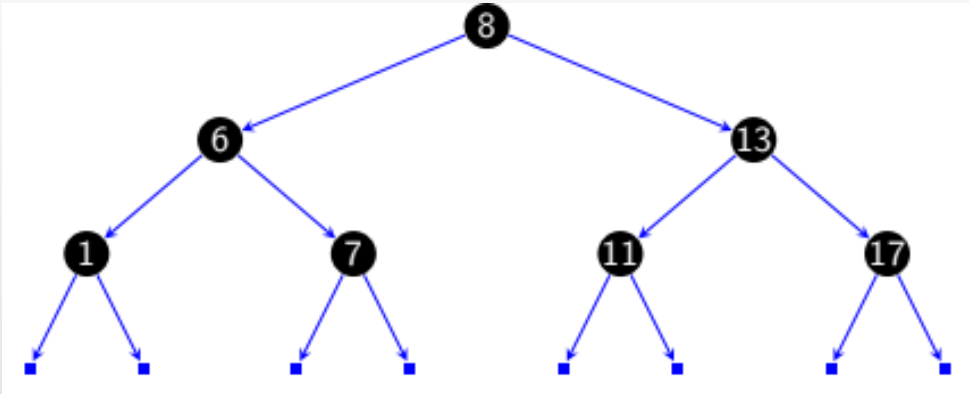
Đổi màu node 6

Red-Black Trees

Thuật toán: Thêm Node



Thêm 13

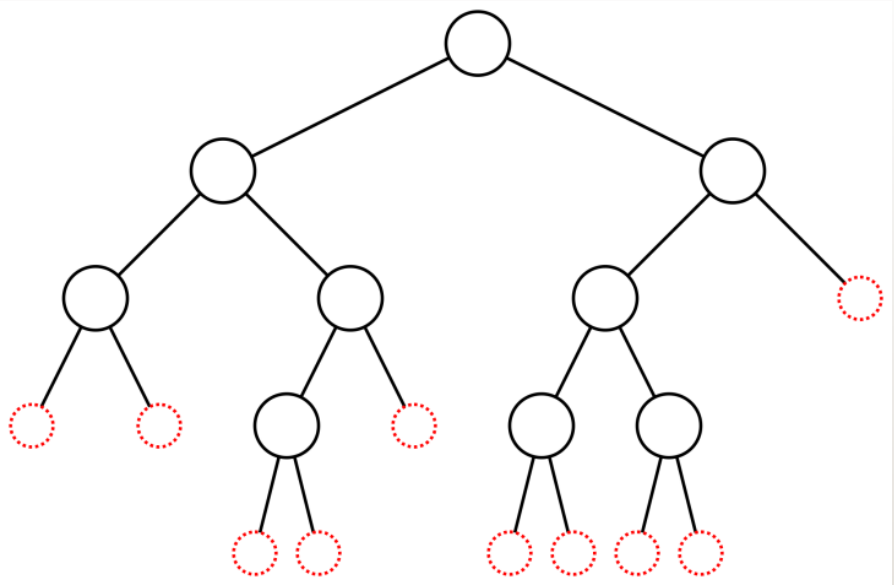


Đổi thành node đen

Weight-Balanced Trees

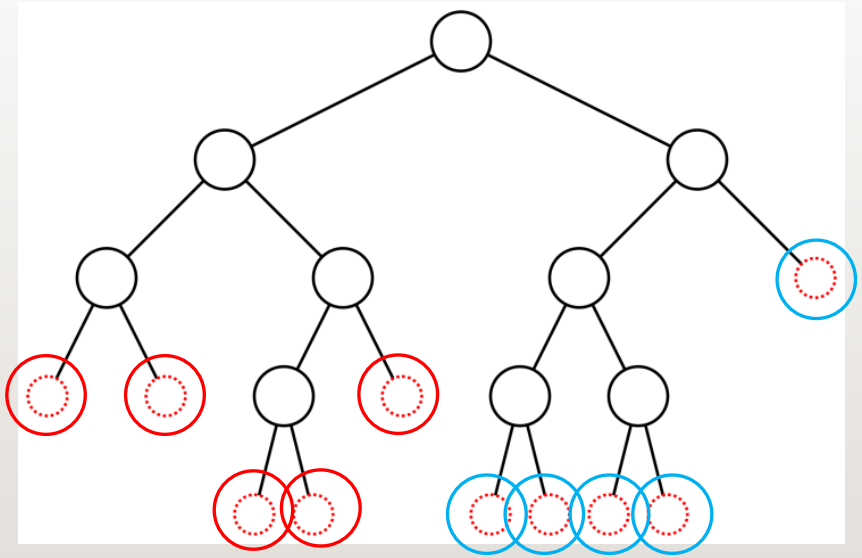
Nhắc lại: một nút trống / cây con null thì bất kỳ vị trí nào trong cây nhị phân có thể được lấp đầy bằng lần chèn tiếp theo:

Cây này có 9 nút và 10 nút trống:



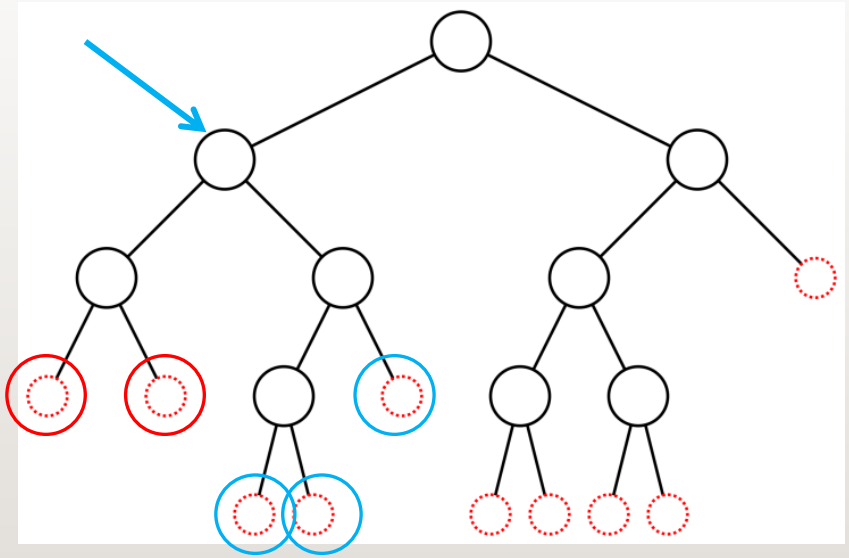
Weight-Balanced Trees

Tỷ lệ của các nút trống ở nút gốc là 5/10 và 5/10



Weight-Balanced Trees

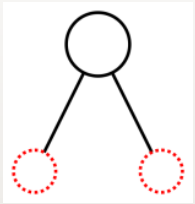
Tỷ lệ của các nút trống ở nút gốc là 2/5 và 3/5



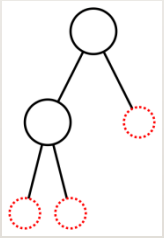
Weight-Balanced Trees

BB(α) trees ($0 < \alpha \leq 1/3$) duy trì cân bằng trọng lượng yêu cầu không bên nào có tỷ lệ nút trống α ít hơn, tức là cả hai tỷ lệ đều rơi vào [α, 1 - α]

-Với một nút, cả hai đều là 0,5



-Với hai, tỷ lệ là 1/3 và 2/3



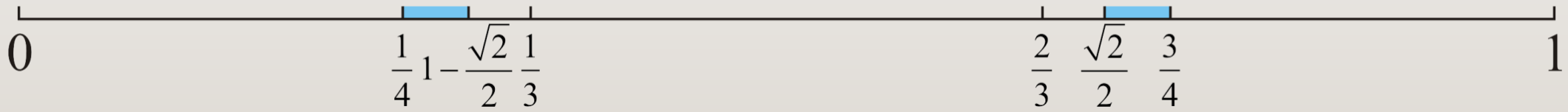
Weight-Balanced Trees

Nếu α bị ràng buộc bởi: $0.25 = \frac{1}{4} \leq \alpha \leq 1 - \frac{\sqrt{2}}{2} \approx 0.2929$

thì có thể thực hiện tất cả các hoạt động trong thời gian Θ(ln (n))

- Nếu a nhỏ hơn 0,25 (khoảng lớn hơn) thì chiều cao của cây có thể là $\omega(\ln(n))$

- Nếu a lớn hơn $1 - \frac{\sqrt{2}}{2}$, các phép toán cần thiết để duy trì sự cân bằng có thể là $\omega(\ln(n))$



Balanced Trees

So sánh thuật toán

implementation	worst case			average case			key
	search	insert	remove	search hit	insert	remove	
unordered list	N	1	N	$N/2$	1	$N/2$	equal
ordered list	N	N	N	$N/2$	$N/2$	$N/2$	compare
ordered array	$\log_2 N$	N	N	$\log_2 N$	$N/2$	$N/2$	compare
BST	N	N	N	$c \log_2 N$	$c \log_2 N$	$\sqrt{N}$	compare
AVL	$c_a \log_2 N$	-	-	$\log_2 N$	-	-	compare
RB	$c_r \log_2 N$	-	-	$\log_2 N$	-	-	compare
goal?							

Note: $c = 1.39, c_a = 1.44, c_r = 2.0$


THE END
Thank you for listening!