TCS-503: Design and Analysis of Algorithms

Unit II

Advanced Data Structures

Unit II

- Advanced Data Structures:
  - Red-Black Trees
  - Augmenting Data Structure
  - B-Trees
  - Binomial Heaps
  - Fibonacci Heaps
  - Data Structure for Disjoint Sets

Why should we learn Red-Black Tree?

A Binary Search Tree of height h

can implement

SEARCH, PREDECESSOR, SUCCESSOR, MINIMUM, MAXIMUM, INSERT, DELETE operations in time O(h).

These operations are fast

if the height of the search tree is small;

but

if its height is large, their performance may be no better than with a linked list.

Why should we learn Red-Black Tree?

Red-Black trees are approximately balanced

and

guarantee that SEARCH, PREDECESSOR, SUCCESSOR, MINIMUM, MAXIMUM, INSERT, DELETE operations take O(lgn) time in the worst case.

Red-Black Trees

Is Binary search tree

with an additional attribute for its nodes:

color

which can be red or black

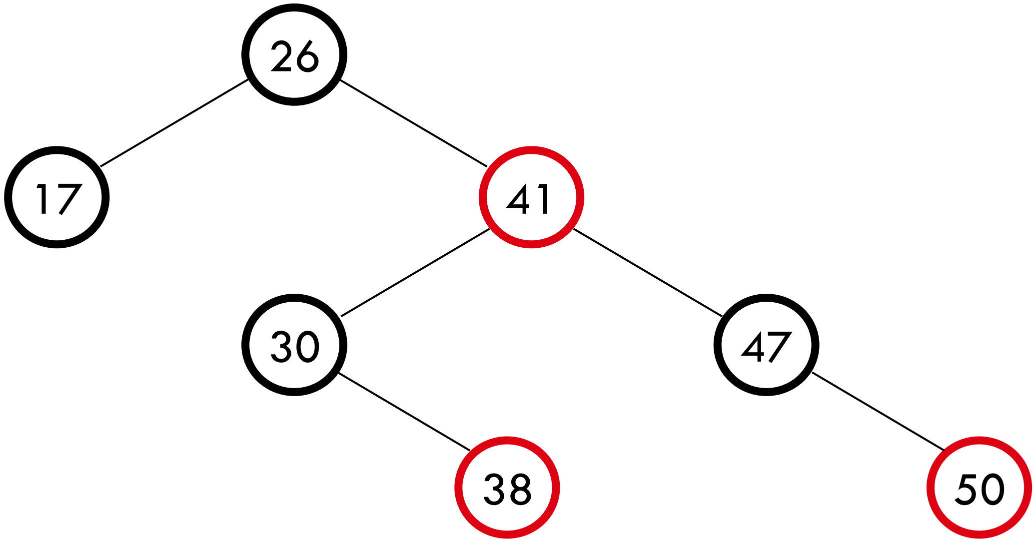
Ensures that

no path is more than twice

as long as any other path

$\Rightarrow$ the tree is balanced

Red-Black Trees

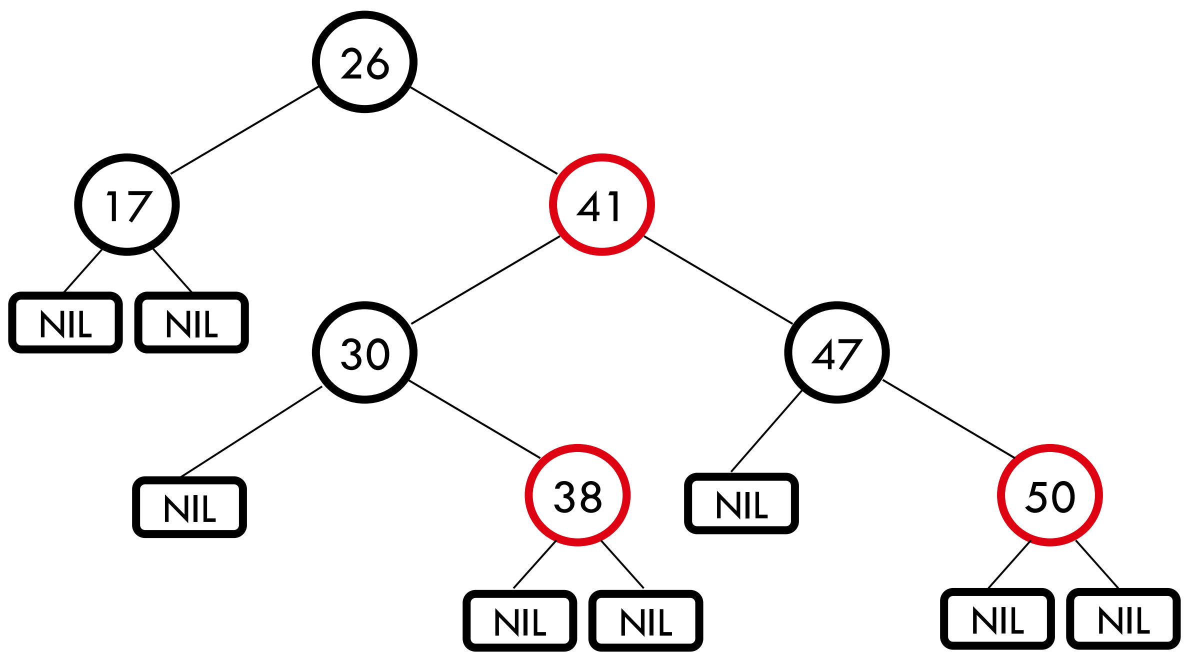


Red-Black Trees

For convenience we use a sentinel NIL[T] to represent all the NIL nodes at the leafs

NIL[T] has the same fields as an ordinary node

Color[NIL[T]] = BLACK

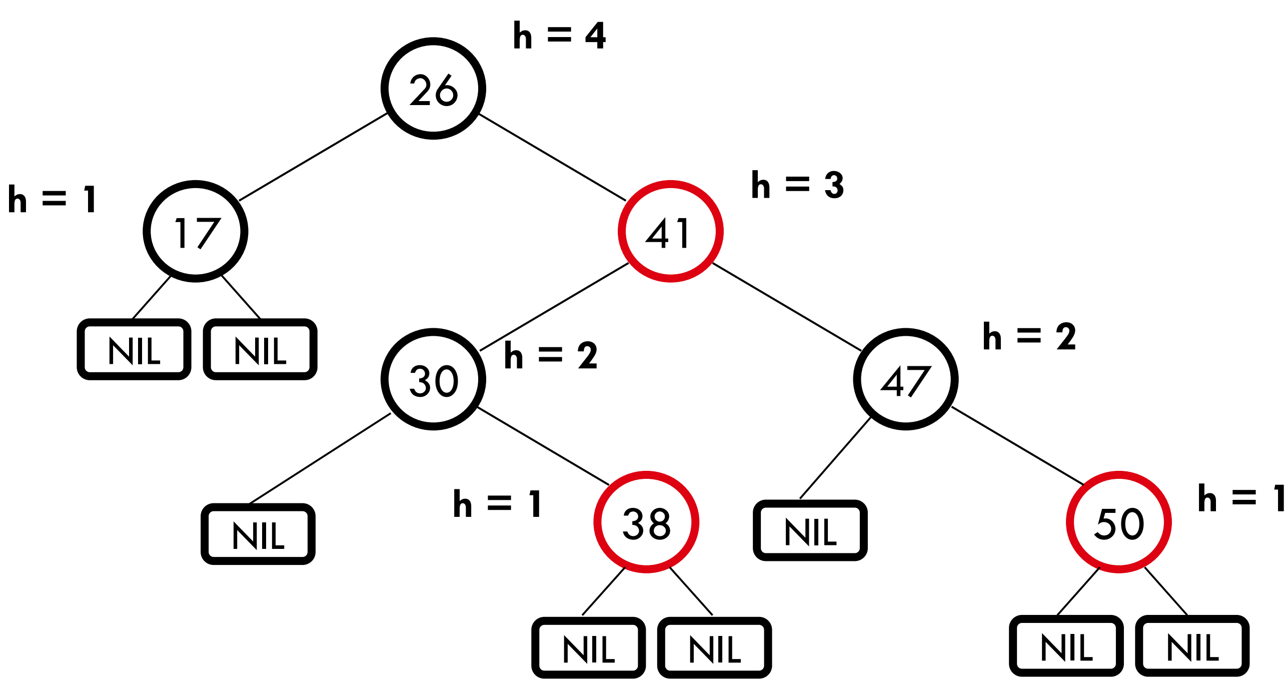


Learn DAA : From B K Sharma

Red-Black Trees

Height and Black-Height of a Node

Height of a node: the number of edges in the longest path to a leaf

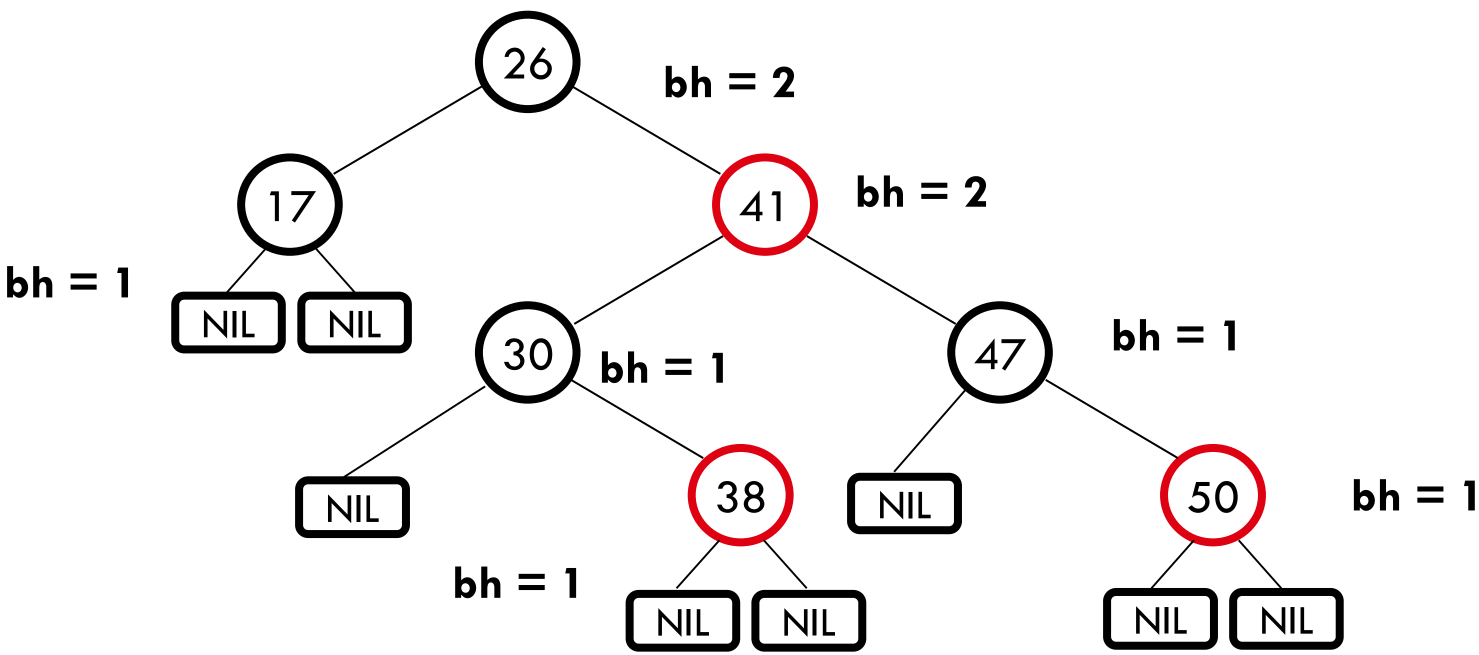


Red-Black Trees

Height and Black-Height of a Node

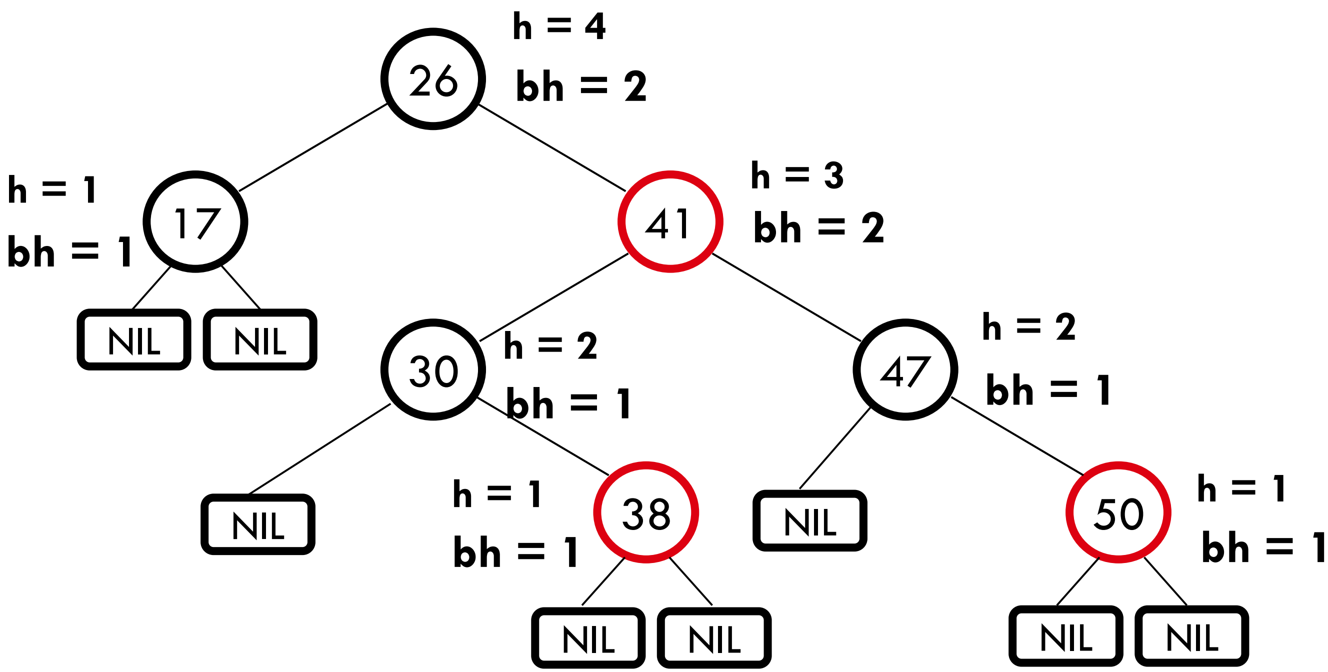
Black-height of a node x:

bh(x) is the number of black nodes(including NIL) on the path from x to a leaf, not counting x



Red-Black Trees

Height and Black-Height of a Node



Properties of Red-Black Trees

1. Every node is either red or black
2. The root is black
3. Every leaf (NIL) is black
4. If a node is red, then both its children are black
5. For each node, all paths from the node to descendant leaves contain the same number of black nodes.

Most important property of Red-Black-Trees

A red-black tree with n internal nodes has height at most $2\lg(n + 1)$.

That is $h(x) \le 2\lg(n + 1)$

Proof:

$$n \geq 2^b - 1 \geq 2^{h/2} - 1$$

number n of internal nodes

since $b \ge h/2$

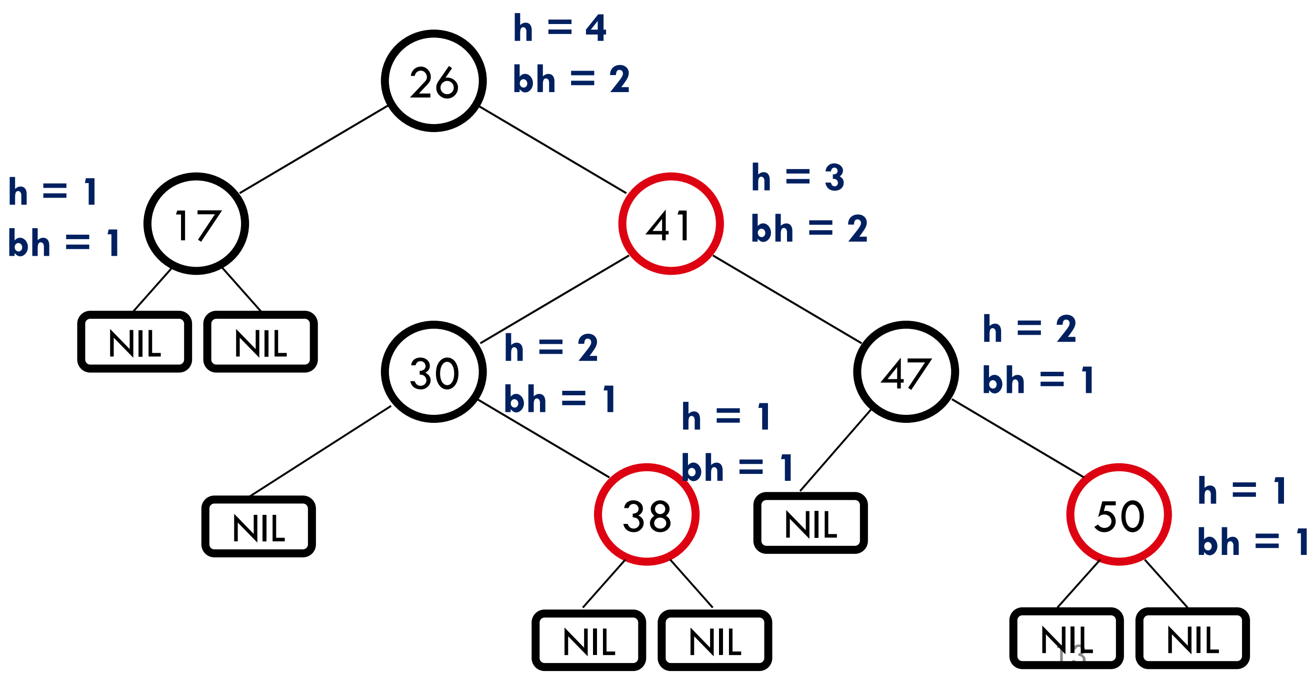
Add 1 to both sides and then take logs:

$$n + 1 \geq 2^b \geq 2^{h/2}$$

$$\lg(n + 1) \geq h/2 \Rightarrow h \leq 2\lg(n + 1)$$

Claim 1:

Any node x with height h(x) has bh(x) ≥ h(x)/2

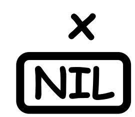


Claim 2:

The subtree rooted at any node x contains at least $2^{bh(x)} - 1$ internal nodes: $n(x) \geq 2^{bh(x)} - 1$

Let x is a leaf Node

$h[x] = 0$



$bh(x) = 0$

Number of internal nodes: $n(x) \geq 2^0 - 1 = 0$

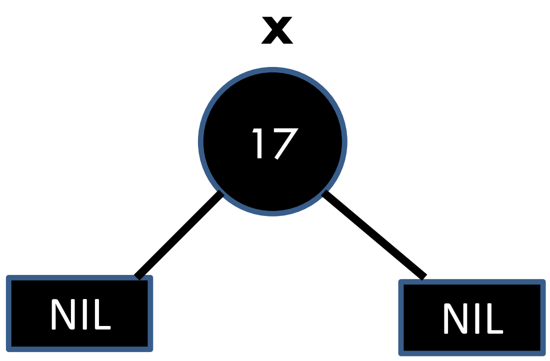
Claim 2:

The subtree rooted at any node x contains at least $2^{bh(x)} - 1$ internal nodes: $n(x) \geq 2^{bh(x)} - 1$

Let x is internal Node

$h[x]=1$

$b[x]=1$



Number of internal nodes: $n(x) \geq 2^1 - 1 = 1$