CMPT307: Red-Black Trees

Week 3-1

Xian Qiu

Simon Fraser University

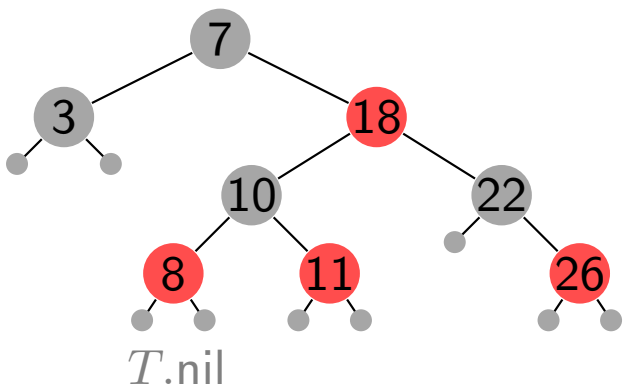
xianq@sfu.ca

Red-Black Tree

binary search tree with color property

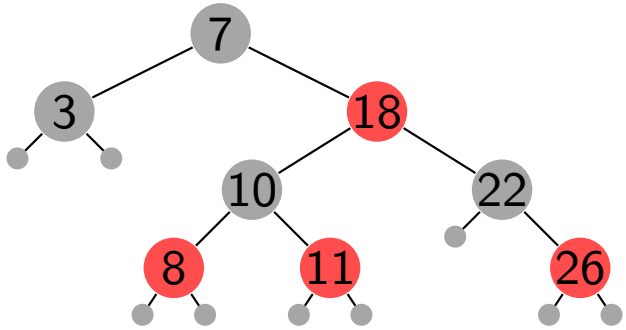
Red-black properties

1. every node is either red or black
2. the root and all leaves (nil) are black
3. the children of a red node are black
4. for each node, all simple paths from the node to descendant leaves contain the same number of black nodes — black-height property



$h = 4$

Intuition



- ▷ each leaf has the same depth w.r.t. black nodes
- ▷ # red nodes $\leq$ # black nodes
- ▷ is it possible to have only black nodes?

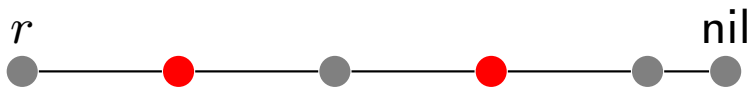
Black-height: bh(x)

bh(x) denotes the number of black nodes (excluding x) of a simple path from x to a descendant leaf.

- ▷ tree height $h \leq 2\text{bh}(r)$, where $r = T.\text{root}$

Tree Height

Lemma. $h \leq 2\text{bh}(r)$, where $r = T.\text{root}$.



$h = \#\text{black} - 1 + \#\text{red} \leq 2\text{bh}(r)$

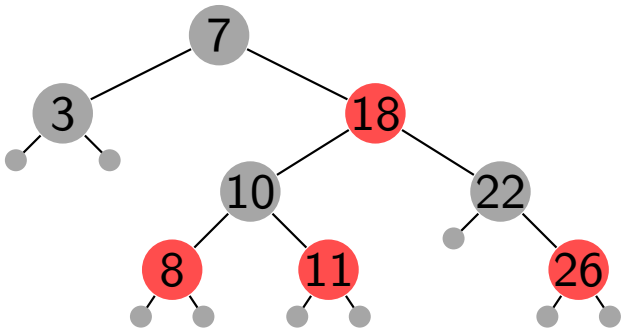
Theorem

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

- any subtree rooted at x has at least $2^{\text{bh}(x)} - 1$ nodes
- $2^{\text{bh(r)}} - 1 \leq n$ yields $\text{bh}(r) \leq \log(n+1)$
- since $h \leq 2\text{bh}(r)$, we get $h \leq 2\log(n+1)$

Tree Height

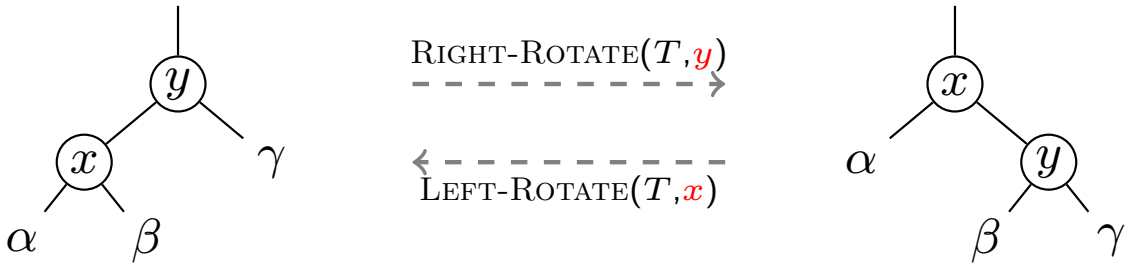
given a red-black tree T, count its black nodes



- ▷ replace a red node by one of its black rooted subtree
- ▷ discard the other one
- ▷ "balanced" binary search tree T' of height $\mathsf{bh}(r) - 1$

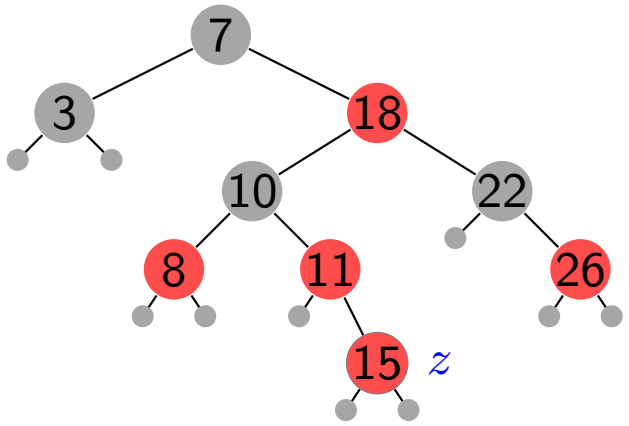
$$|V(T)| \geq |V(T')| = 1 + 2 + \cdots + 2^{\mathsf{bh}(r)-1} = 2^{\mathsf{bh}(r)} - 1$$

Rotations



- use RIGHT-ROTATE to move up left-branch
  use LEFT-ROTATE to move up right-branch
- rotations preserve binary-search-tree property
- running time $O(1)$

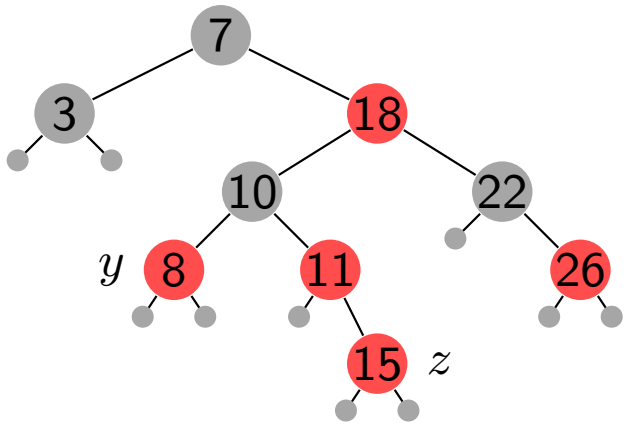
Insertion



- color z red or black?
- idea: color red and move the violation up (if possible)
- z's uncle y is critical
  1. y is red: move up the violation
  2. y is black and z is a left child: RIGHT-ROTATE
  3. y is black and z is a right child: LEFT-ROTATE

Example

insert $z = 15$

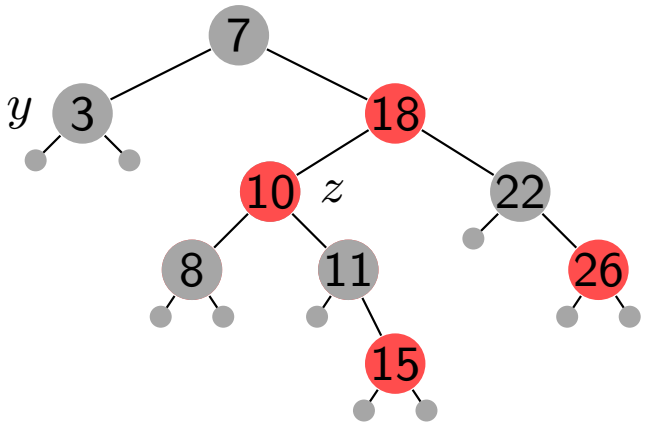


case 1: y is red

color y and z.p black; color y.p red

Example

insert $z = 15$



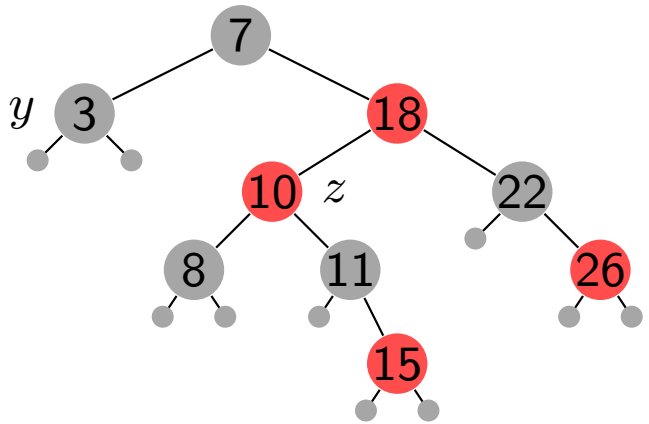
case 1: y is red

color y and z.p black; color y.p red

update z, y

Example

insert $z = 15$



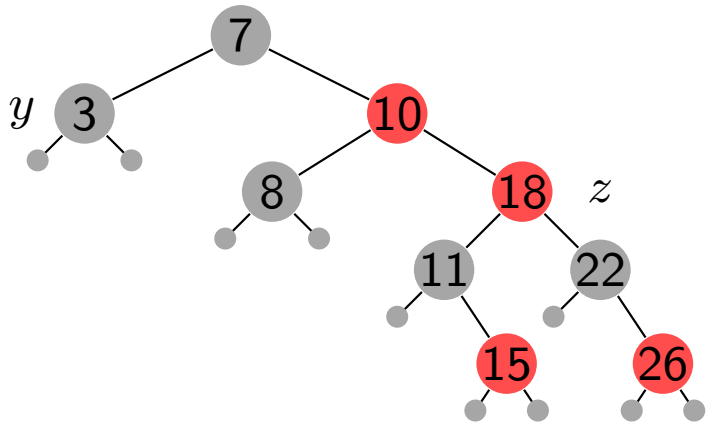
case 2: y is black and z is a left child

RIGHT-ROTATE(T,z.p);

update z, y;

Example

insert $z = 15$



case 2: y is black and z is a left child

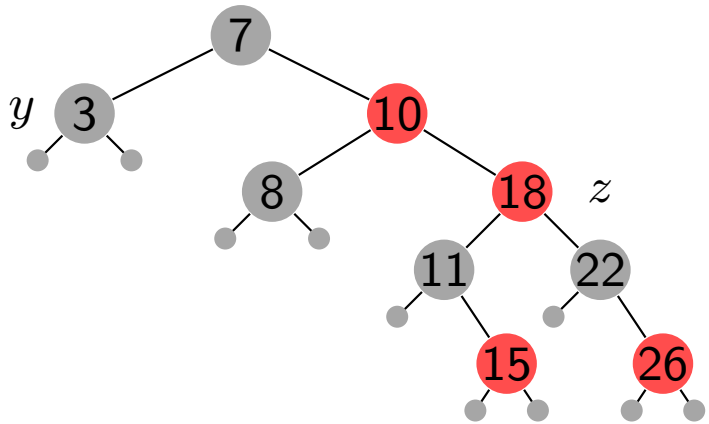
RIGHT-ROTATE(T,z.p);

update z, y;

go to case 3

Example

insert $z = 15$

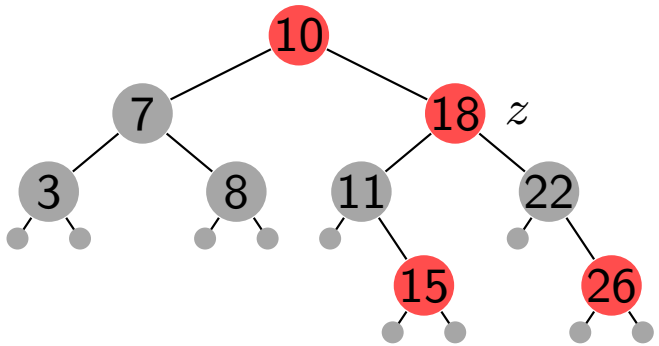


case 3: y is black and z is a right child

LEFT-ROTATE(T,z.p.p);

Example

insert $z = 15$

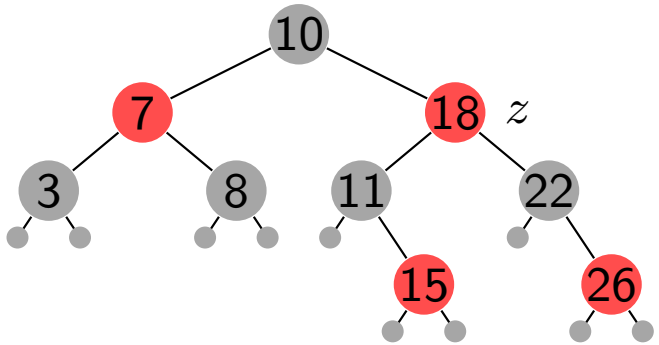


case 3: y is black and z is a right child

LEFT-ROTATE(T,z.p.p);

Example

insert $z = 15$



case 3: y is black and z is a right child

LEFT-ROTATE(T,z.p.p);

recolor z.p and z.p.left;

Analysis

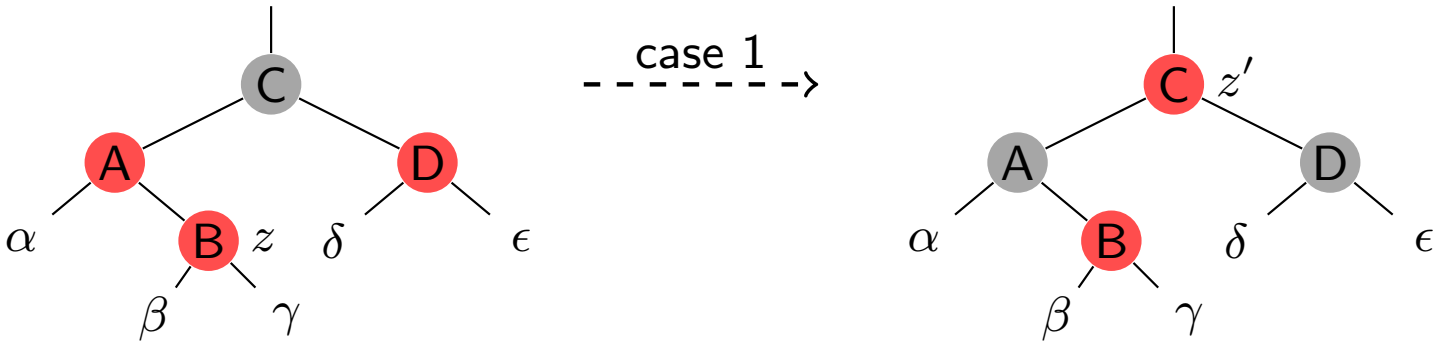
Loop invariant

1. node z is red
2. if z.p is the root, then z.p is black
3. it violates at most one red-black property at any time

   violation 1: z is the root and is red
   violation 2: both z and z.p are red

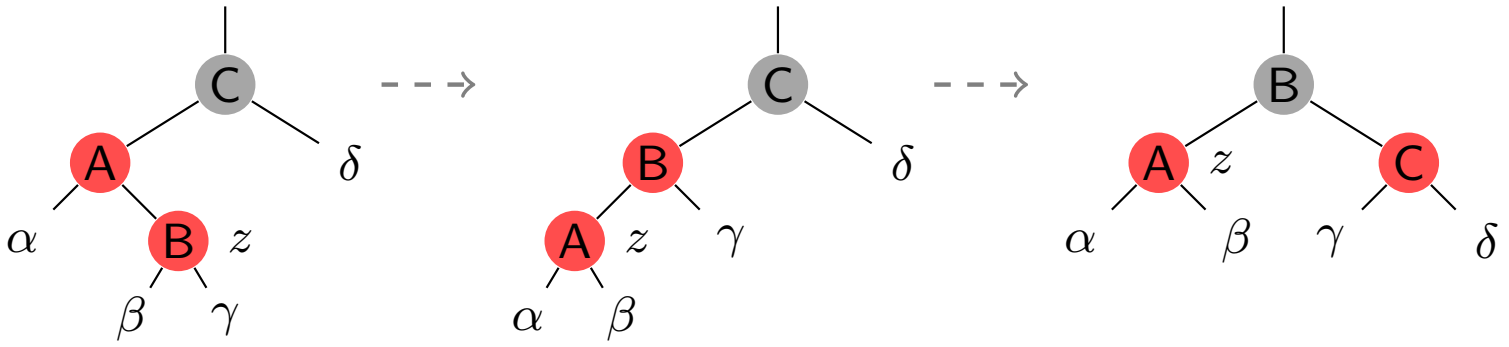
- ▷ initialization satisfies the loop invariant
- ▷ termination: z.p is black (root is also colored black)
- ▷ to show: loop invariant holds for each loop
  need check the black-height property

Case 1



- z.p $=$ z.p.p.left (another case can be discussed analogously)
- black-height property holds
- if z' is the root, then violation 1 occurs
- else if z'.p is red, violation 2 occurs

Case 2 & 3



case 2 case 3

- case 2: black-height property holds, violation 2
- case 3: all red-black properties will hold at termination
- running time of insertion?

Pseudocode

RB-INSERT-FIXUP(T,z)

```
1  while z.p.color == RED do
2      if z.p = z.p.p.left then
3          y = z.p.p.right ;                                   // uncle y
4          if y.color == RED then
5              z.p.color = BLACK; y.color = BLACK ;            // case 1
6              z.p.p.color = RED; z = z.p.p;
7          else if z == z.p.right then
8              z = z.p; LEFT-ROTATE(T, z) ;                    // case 2
9          z.p.color = BLACK; z.p.p.color = RED ;              // case 3
10         RIGHT-ROTATE(T, z.p.p);
11     else
12         ... same as above, with "left" and "right" exchanged
13 T.root.color = BLACK;
```