Red-Black Tree

Balanced Binary Search Tree

Hiago Lopes Cavalcante
João Pedro Brito Tomé
John Davi Dutra Canuto Pires
Luana Júlia Nunes Ferreira
Ruan Heleno Correa da Silva

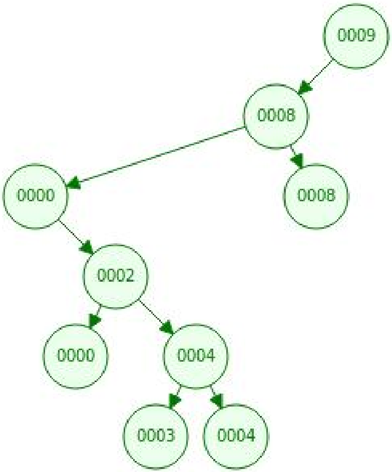
Universidade Federal de Alagoas
Instituto de Computação
https://github.com/ferreiraluana/Data-Structure-Project

25 de março de 2019

Outline

Previous in BST classes...

Let's consider adding theses numbers: 9, 8, 8, 0, 2, 0, 4, 4, 3.



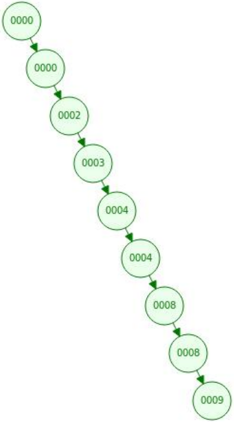
Troubles start to show up...

What if we add the same numbers in descending order?

Low efficiency:

- Search;
- Insert;
- Delete;

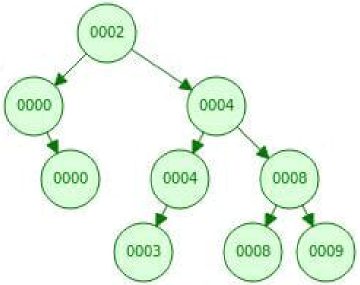
It runs $O(n)$!



AVL tree

What do we need? BALANCE!
And here it comes: AVL Tree!

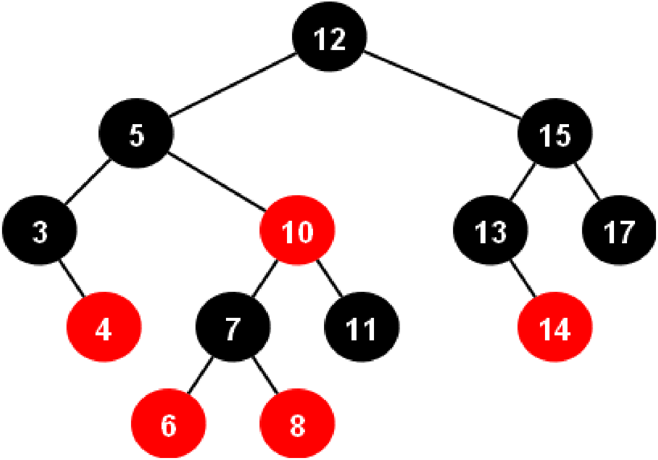
- 4 kinds of rotations (L-L, R-R, R-L, L-R);
- Balance Factor: 0, -1 or 1;
- Searching in AVL is close to O(log n).



But...

AVL needs too many rotations!!!

The Red-Black Tree



Definition

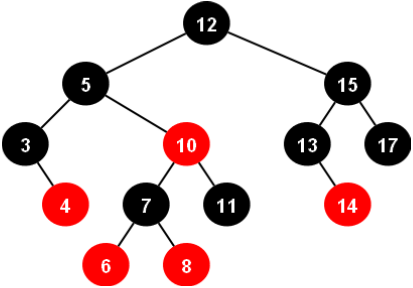
- A Binary Search Tree with an extra bit to hold the color:

$$\mathbf{RED = 1}$$
$$\mathbf{BLACK = 0}$$

- It ensures the tree remains balanced.

Red-Black Structure

```
struct redblack
{
    int item;
    int color;
    redblack *left;
    redblack *right;
}
```



Advantages of Red-Black

1. Rotations run in $O(1)$;
2. Searching, insertion and deletion run in $O(\log n)$;
3. In remotion, the RB tree rotates once (with single or double rotation), while the AVL tree can rotate $\log n$ times;

Properties

- Every node is either red or black;
- The root and leaves **(NIL's)** are black;
- If a node is **red**, then its parent is **black**;
- For each node, every path from the node to the descendant leaves contains the same number of **BLACK** nodes.

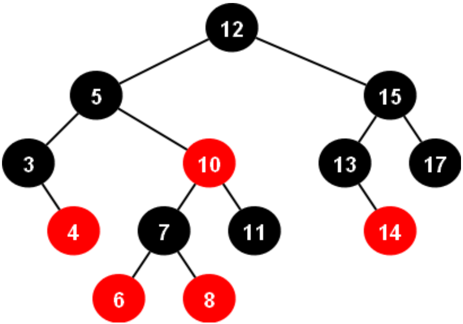
Properties

- There can't be two consecutive red nodes in a path from the root to a sub-tree;
- The properties are checked every time a operation is done in the RB tree;
- In case some property is not satisfied, rotations and color flips are done.

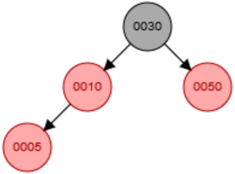
Black Height

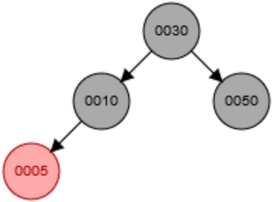
It is the number of **BLACK** nodes found until any descendant node. A red-black tree with n keys has height:

$$h \leq 2log(n+1)$$

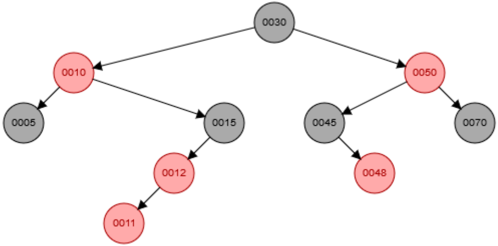


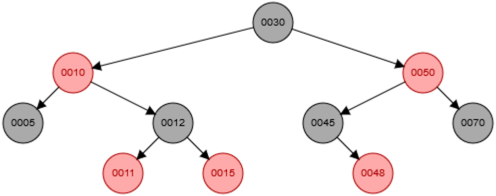
Color Flip



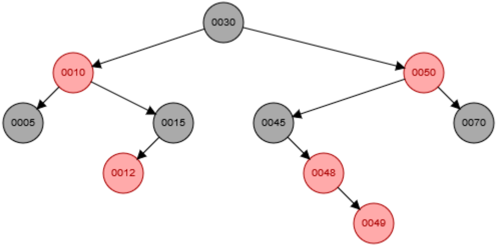


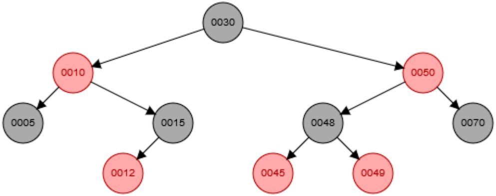
Right Rotation





Left Rotation





Abstract Data Type

```
btree* create_empty_btree();
btree* rotate_left(btree *bt);
btree* rotate_right(btree *bt);
btree* move_left_red(btree *bt);
btree* move_right_red(btree *bt);
btree* balance_factor(btree *bt);
btree* add_bt(btree *bt, int value);
btree* add_arv(btree *bt, int value);
btree* remove_bt(btree *bt, int value);
btree* remove_arv(btree *bt, int value);
int search(btree *bt, int value, int *flag);
int color(btree *bt);
void print_pre_order(btree *bt);
void color_swap(btree *bt);
```

Add

```
btree* add_bt(btree *bt, int value)
{
if(bt == NULL)
{
btree *new_btree = (btree*) malloc ( sizeof(btree) );
if(new_btree == NULL) return NULL;
new_btree → item = value;
new_btree → color = RED;
new_btree → left = NULL;
new_btree → right = NULL;
return new_btree;
}
if(value != bt → item)
{
if(value < bt → item)  bt → left = add_bt(bt → left, value);
else if(value > bt → item)  bt → right = add_bt(bt → right, value);
}
if(color(bt → right) == RED  &&  color(bt → left) == BLACK)  bt = rotate_left(bt);
if(color(bt → left) == RED  &&  color(bt → left → left) == RED)  bt = rotate_right(bt);
if(color(bt → left) == RED  &&  color(bt → right) == RED)
color_swap(bt);
return bt;
}
```

Balance Factor

btree* balance_factor(btree *bt);
{
if(color(bt $\rightarrow$ $right) == RED) \quad bt = rotate\_left(bt)$;
if(bt $\rightarrow$ $left! = NULL \quad \&\& \quad color(bt \rightarrow right) == RED \quad \&\&$ color(bt $\rightarrow$ $left \rightarrow left) == RED) \quad bt = rotate\_right(bt)$;
if(color(bt $\rightarrow$ $left) == RED \quad \&\& \quad color(bt \rightarrow right) == RED) \quad color\_swap(bt)$;
return bt;
}

Rotate Left and Rotate Right

btree* rotate_left(btree *bt)
{
btree *aux $= \mathrm{bt} \rightarrow right$;
bt $\rightarrow right = aux \rightarrow left$;
aux $\rightarrow left = bt$;
aux $\rightarrow color = bt \rightarrow color$;
bt $\rightarrow color = RED$;
return aux;
}

btree* rotate_right(btree *bt)
{
btree *aux $= \mathrm{bt} \rightarrow left$;
bt $\rightarrow left = aux \rightarrow right$;
aux $\rightarrow right = bt$;
aux $\rightarrow color = bt \rightarrow color$;
bt $\rightarrow color = RED$;
return aux;
}

Move Left Red and Move Right Red

```
btree* move_left_red(btree *bt)
{
color_swap(bt);
if(color(bt->right->left) == RED)
{
bt->right = rotate_right(bt->right);
bt = rotate_left(bt);
color_swap(bt);
}
return bt;
}
```

```
btree* move_right_red(btree *bt)
{
color_swap(bt);
if(color(bt->left->left) == RED)
{
bt = rotate_right(bt);
color_swap(bt);
}
return bt;
}
```

Remove

btree* remove_bt(btree *bt, int value)
{
if(value $<$ bt $\rightarrow$ $item$)
{
if(color(bt $\rightarrow$ $left$) $== BLACK \quad \&\& \quad color(bt \rightarrow left \rightarrow left) == BLACK) \quad bt = move\_left\_RED(bt)$;
bt $\rightarrow$ $left = remove\_bt(bt \rightarrow left, value)$; }
else {
if(color(bt $\rightarrow$ $left$) $== RED) \quad bt = rotate\_right(bt)$;
if(value $==$ bt $\rightarrow$ $item \quad \&\& \quad bt \rightarrow right == NULL) \quad \{free(bt); returnNULL; \}$
if(color(bt $\rightarrow$ $right$) $== BLACK \quad \&\& \quad color(bt \rightarrow right \rightarrow left) == BLACK) \quad bt = move\_right\_RED(bt); if(value == bt \rightarrow item)$
{
btree *aux $=$ minor_search(bt $\rightarrow$ $right$);
bt $\rightarrow$ $item = aux \rightarrow item$;
bt $\rightarrow$ $right = minor\_remove(bt \rightarrow right)$;
} else bt $\rightarrow$ $right = remove\_bt(bt \rightarrow right, value)$;
}
return balance_factor(bt);
}

Instituto de Computação

Search

int search(btree *bt, int value, int *flag)
{
if(bt $!=$ NULL)
{
if(bt $\rightarrow item == value) \quad *flag = 1;$
search(bt $\rightarrow left, value, flag);$
search(bt $\rightarrow right, value, flag);$
}
return *flag;
}

Animation

bit.ly/gifredblack
imgur.com/vV1RDz5

Conclusion

Red-Black Trees can be very useful!

1. Running time: $O(\log n)$;
2. Rotations: $O(1)$;

References

- E.Demaine, **"Introduction to algorithms"**, Lecture 10, Massachusetts Institute of technology Open Course, 2015;
- S. W. Song, **"Árvore Rubro-Negra"**, Estruturas de Dados, Universidade de São Paulo - IME/USP, 2008;
- T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, **"Introduction to Algorithms"**, 2º edition, MIT Press & McGraw-Hill, 2001

Thank you!