ÁRVORES AVL

Gustavo Carvalho
(ghpc@cin.ufpe.br)

Universidade Federal de Pernambuco
Centro de Informática, 50740-560, Brazil

Agenda

Introdução

Motivação: bom desempenho da BST depende do balanceamento

- AVL: uma BST auto balançeável
- Criadores (1962): G. M. Adelson-Velsky e E. M. Landis

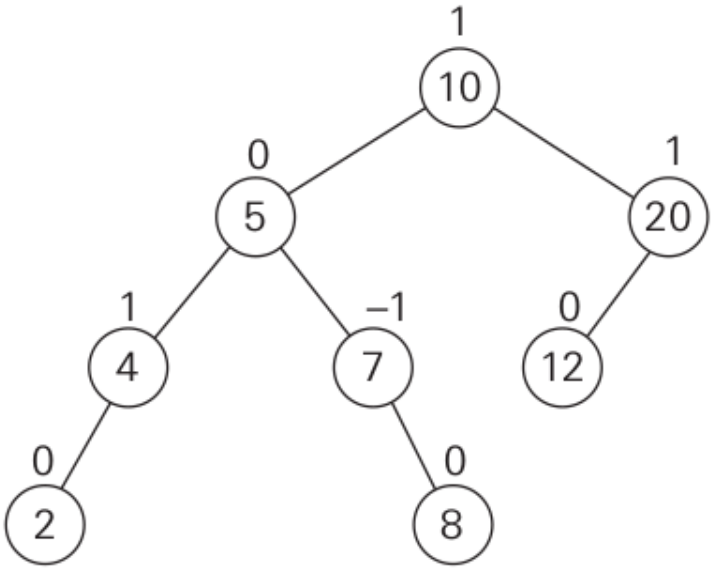
AVL: para todo vértice, o módulo da diferença das alturas das sub-árvores nunca pode ser maior do que 1

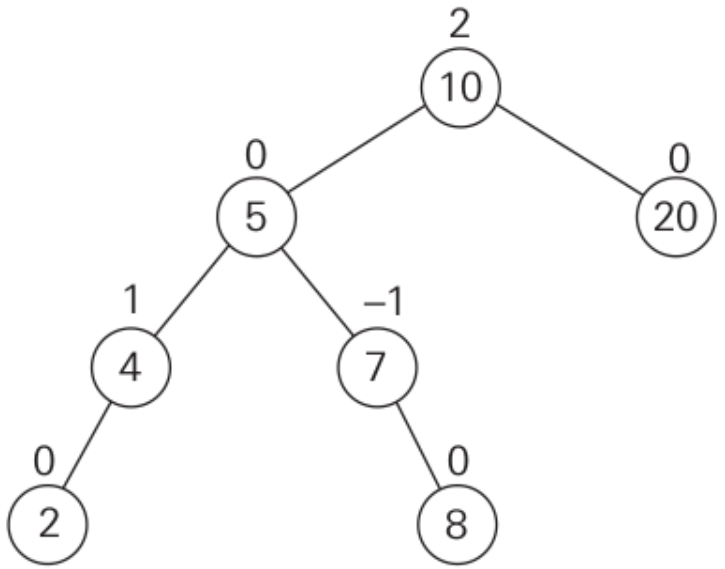
- Fator de balanceamento: -1, 0 ou +1.
- Altura da árvore vazia: -1

Rotação: transformação para balancear a árvore

- Após inserção, quando o fator de um nó é -2 ou +2
- No vértice -2 ou +2 mais próximo do ponto de inserção

Introdução





[1]

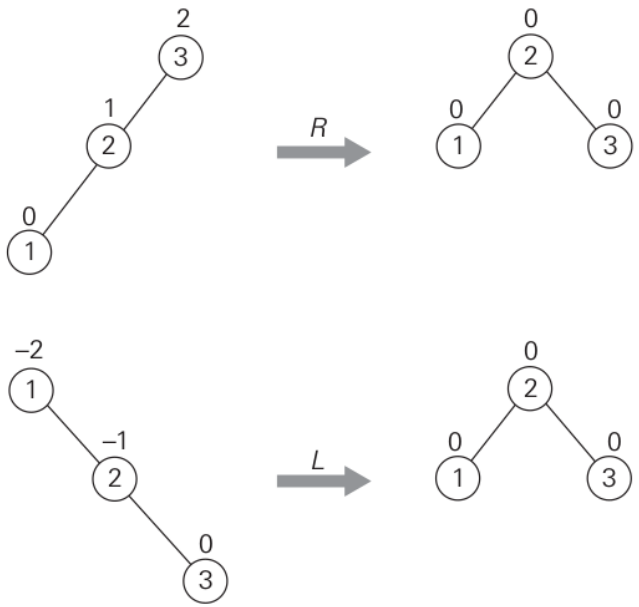
[1] Fonte: A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

Agenda

Rotações

Existem quatro casos:

- Rotação simples à direita (R): valor inserido na sub-árvore à esquerda do filho à esquerda do nó desbalanceado (+1 antes)
- Rotação simples à esquerda (L): valor inserido na sub-árvore à direita do filho à direita do nó desbalanceado (-1 antes)



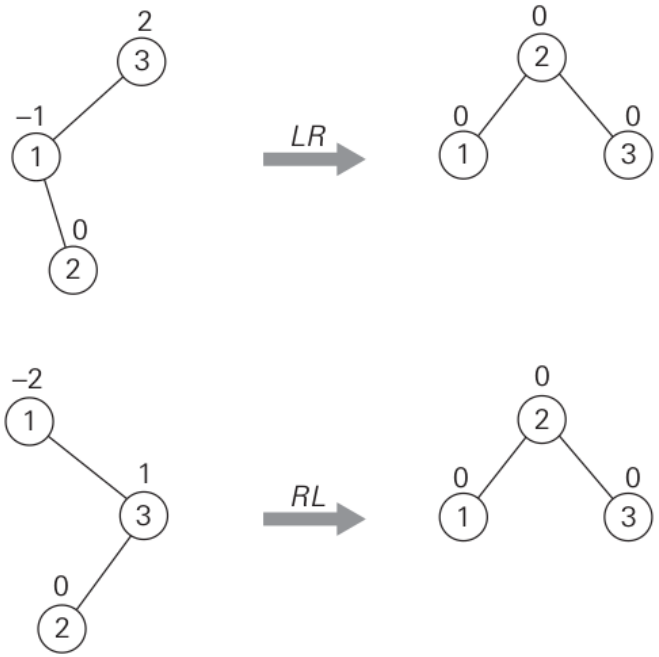
[2]

[2] Fonte: A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

Rotações

Existem quatro casos:

- Rotação dupla à direita (LR): valor inserido na sub-árvore à direita do filho à esquerda do nó desbalanceado (+1 antes)
- Rotação dupla à esquerda (RL): valor inserido na sub-árvore à esquerda do filho à direita do nó desbalanceado (-1 antes)

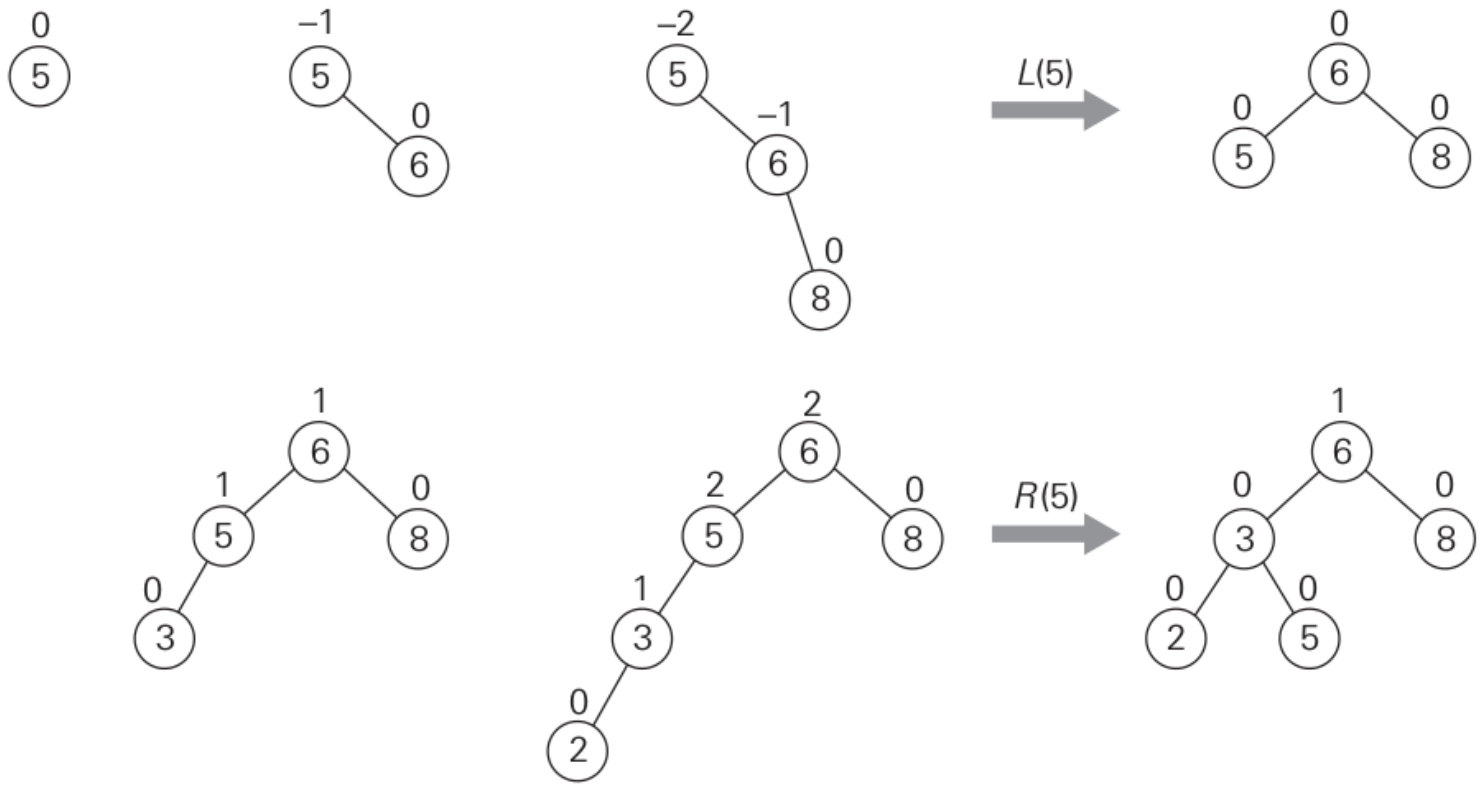


3

3

Rotações

Inserindo valores: 5, 6, 8, 3, 2, 4, e 7

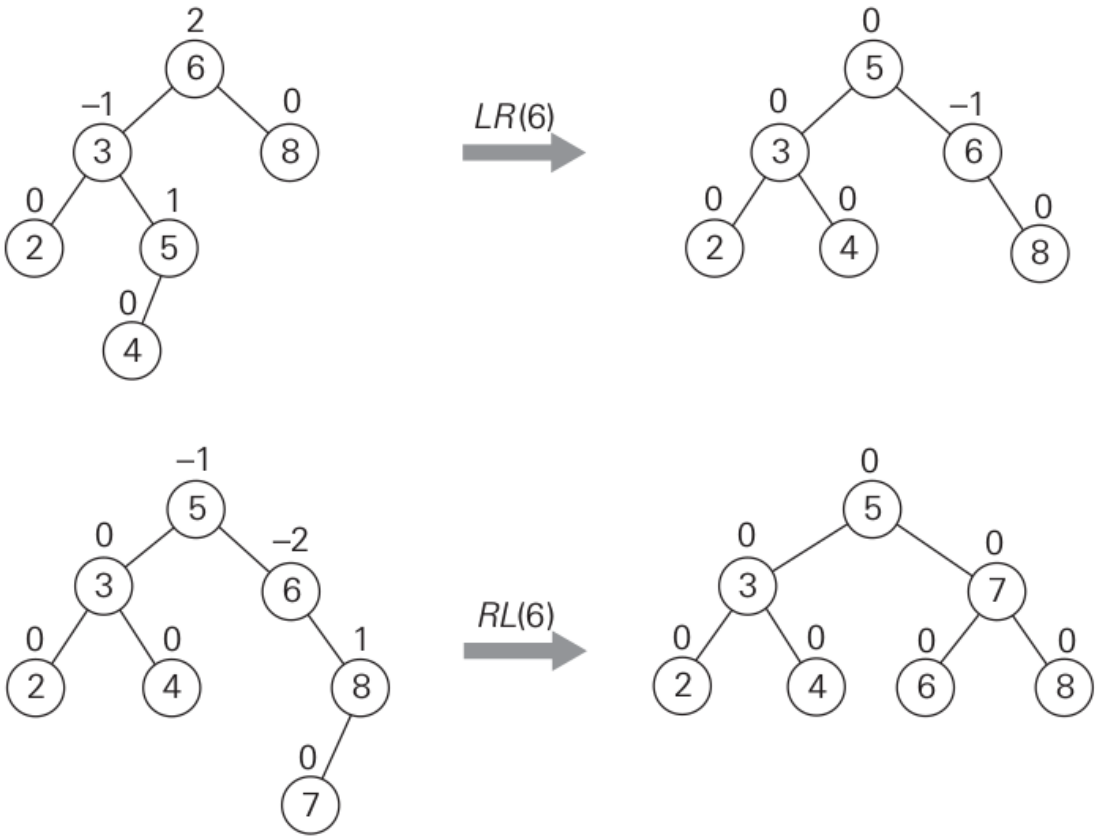


4

[4] Fonte: A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

Rotações

Inserindo valores: 5, 6, 8, 3, 2, 4, e 7

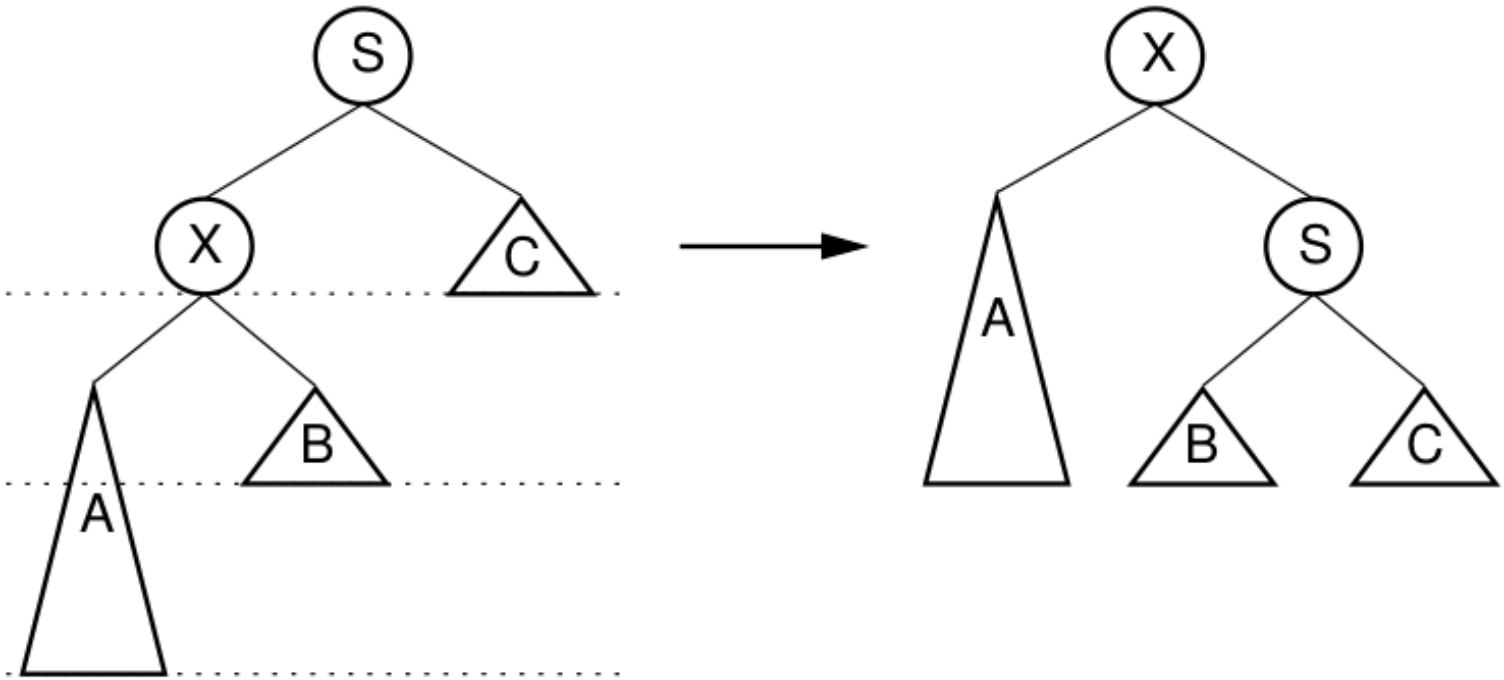


5

[5] Fonte: A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

Rotações

Rotação simples (R): de uma forma mais geral



6

6 Fonte: A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

Rotações

Algoritmo: BSTNode inserthelp(BSTNode rt, Key k, E e)

```
1  if rt = NULL then return create_bstnode(k, e) ;
2  if rt.key > k then
3  |    rt.left ← inserthelp(rt.left, k, e);
4  else
5  |    rt.right ← inserthelp(rt.right, k, e);
6  rt.height ← 1 + max(height(rt.left), height(rt.right));
7  int balance ← getBalance(rt);
8  if balance > 1 ∧ k < rt.left.key then return rigthRotate(rt) ;
9  if balance < −1 ∧ k ≥ rt.right.key then return leftRotate(rt) ;
10 if balance > 1 ∧ k ≥ rt.left.key then
11 |    rt.left ← leftRotate(rt.left);
12 |    return rigthRotate(rt);
13 if balance < −1 ∧ k < rt.right.key then
14 |    rt.right ← rightRotate(rt.right);
15 |    return leftRotate(rt);
16 return rt;
```

Rotações

Algoritmo: int getBalance(BSTNode rt)

```
1  if rt = NULL then return 0 ;
2  return height(rt.left) − height(rt.right);
```

Algoritmo: int height(BSTNode rt)

```
1  if rt = NULL then return −1 ;
2  return rt.height;
```

Rotações

Algoritmo: BSTNode rightRotate(BSTNode rt)

```
1  BSTNode l ← rt.left;
2  BSTNode lr ← l.right;
3  l.right ← rt;
4  rt.left ← lr;
5  rt.height ← max(height(rt.left), height(rt.right)) + 1;
6  l.height ← max(height(l.left), height(l.right)) + 1;
7  return l;
```

Rotações

Algoritmo: BSTNode leftRotate(BSTNode rt)

```
1  BSTNode r ← rt.right;
2  BSTNode rl ← r.left;
3  r.left ← rt;
4  rt.right ← rl;
5  rt.height ← max(height(rt.left), height(rt.right)) + 1;
6  r.height ← max(height(r.left), height(r.right)) + 1;
7  return r;
```

Remoção: análoga à remoção em BST + rotações

- A partir do nível da operação *deletemin*

Custo computacional

Relembrando:

Data Structure	Time Complexity								Space Complexit
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	O(1)	O(n)	O(n)	O(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Queue	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Singly-Linked List	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Skip List	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log(n))
Hash Table	N/A	O(1)	O(1)	O(1)	N/A	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)
Cartesian Tree	N/A	O(log(n))	O(log(n))	O(log(n))	N/A	O(n)	O(n)	O(n)	O(n)
B-Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Red-Black Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Splay Tree	N/A	O(log(n))	O(log(n))	O(log(n))	N/A	O(log(n))	O(log(n))	O(log(n))	O(n)
AVL Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
KD Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)

7

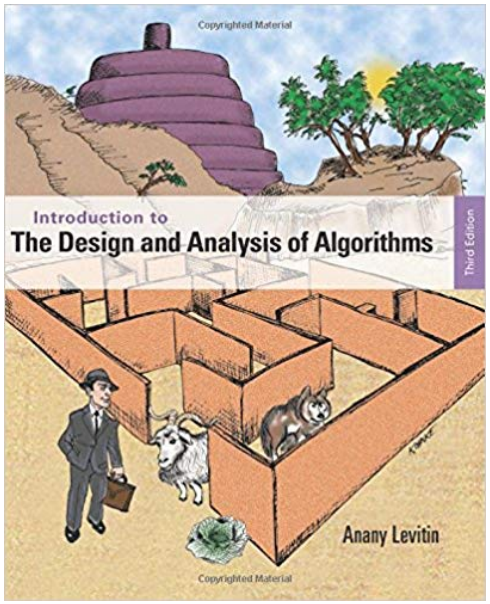
AVL: $\approx$ o mesmo número de comparações da busca binária

Desvantagens: rotações frequentes + armazenar fator por nó

[7] http://bigocheatsheet.com/

Agenda

Bibliografia + leitura recomendada



Capítulo 6 (pp. 218–223)
Anany Levitin.
Introduction to the Design and Analysis of Algorithms.
3a edição. Pearson. 2011.

Capítulo 13 (pp. 435–437)
Clifford Shaffer.
Data Structures and Algorithm Analysis.
Dover, 2013.

ÁRVORES AVL

Gustavo Carvalho
(ghpc@cin.ufpe.br)

Universidade Federal de Pernambuco
Centro de Informática, 50740-560, Brazil