Árvores AVL

Estruturas de Dados e Algoritmos – Ciência da Computação

Prof. Daniel Saad Nogueira Nunes

IFB – Instituto Federal de Brasília, Campus Taguatinga

INSTITUTO FEDERAL
Brasília

Sumário

Sumário

Árvores AVL

- As árvores AVL foram as primeiras BST balanceadas da literatura.
- Nomeada de acordo com os seus desenvolvedores: Georgy **A**delson-**V**elsky e Evgenii **L**andis.
- Possuem tempo de consulta/inserção/remoção limitado a $\Theta(\lg n)$.
- São estruturas que são auto-balanceáveis, tentando deixar as alturas dos nós de cada nível próximas.

Árvore AVL

- Antes de expor os algoritmos que atuam sobre esta estrutura de dados, é necessário formalizar alguns conceitos.

Sumário

Árvore AVL

Definição ($h(x)$)

- Seja T uma árvore binária com raiz no nó x e seja uma função $ht(v)$ que nos dá a altura de um nó $v \in T$.
- A função $h(T)$ corresponde à seguinte definição:

$$h(T) = \begin{cases} 0, & \text{se } T \text{ é uma árvore vazia} \\ ht(x) + 1, & \text{caso contrário} \end{cases}$$

Árvore AVL

Definição (Fator de Equilíbrio)

Sejam T_1 e T_2 as subárvores da esquerda e da direita de uma árvore com raiz no nó x.

O fator de equilíbrio (*balance factor*) de x é dado como:

$$bf(x) := h(T_1) - h(T_2)$$

Árvore AVL

Definição (Árvore AVL)

Seja T uma árvore binária de pesquisa. T também é uma árvore AVL se e somente se:

- T é vazia; ou
- $-1 \leq bf(x) \leq 1, \forall x \in T$

Árvore AVL

figuras/AVL-tree.pdf

Árvore AVL

Fator de Equilíbrio

- Em árvores AVL, o fator de equilíbrio de cada nó está limitado a três possíveis valores: $\{-1, 0, 1\}$.
- Esta restrição que possibilita operações de inserção/remoção/busca serem efetuadas em tempo $\Theta(\lg n)$.

Árvore AVL

figuras/AVL-tree.pdf

Sumário

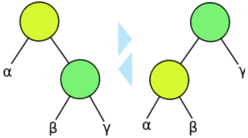
Operações em Árvores AVL

- Como árvores AVL são especializações de BST, os procedimentos de inserção, remoção e busca são bem parecidos.
- A diferença é que, após uma inserção e remoção, a árvore pode ficar desbalanceada, isto é, o fator de equilíbrio de algum nó está fora do intervalo $\{-1, 0, 1\}$.
- Assim, é necessário rebalancear a árvore.

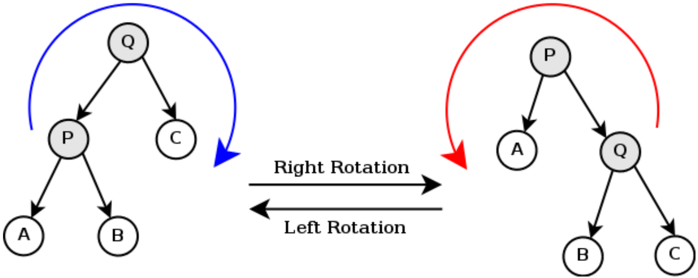
Balanceamento em Árvores AVL

- O rebalanceamento é feito através de rotações, que podem ser:
  - Rotações para a esquerda.
  - Rotações para a direita.
- Uma rotação não interfere na propriedade de BST, isto é, a subárvore da esquerda continua os elementos com chaves menores do que a nova raiz e a subárvore da direita continua com os elementos com chaves maiores que a da nova raiz.

Rotações



Rotações



Árvores AVL

Tree Rotation

Árvores AVL

- Ao final de cada inserção/remoção, a árvore deve ser atualizada.
- Para cada nó ao longo do caminho percorrido, deve-se computar:
  - A nova altura.
  - O novo fator de equilíbrio.
- Rotações para a esquerda/direita são feitas de movo a preservar os fatores de balanceamento no intervalo $\{-1, 0, 1\}$.
- Elas aumentam a altura de algumas subárvores enquanto diminuem a de outras.

Balanceamento de Árvores AVL

- O balanceamento das árvores AVL concentram-se em 4 casos:
  - Rotação para esquerda (L).
  - Rotação para direita (R).
  - Rotação para esquerda seguida de rotação para direita (LR).
  - Rotação para direita seguida de rotação para esquerda (RL).

Balanceamento de Árvores AVL

Caso 1 (L)

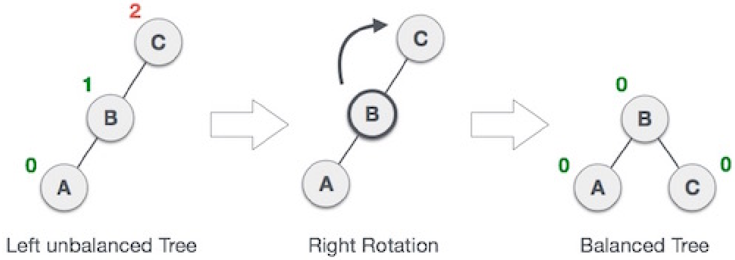
- Se a raiz da árvore tem fator de equilíbrio -2 e seu filho da direita possui fator de equilíbrio ≤ 0, uma rotação à esquerda é suficiente para balancear a árvore.

figuras/avl-tree-case-1.jpg

Balanceamento de Árvores AVL

Caso 2 (R)

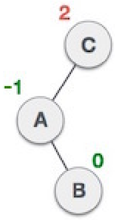
- Se a raiz da árvore tem fator de equilíbrio 2 e seu filho da esquerda possui fator de equilíbrio ≥ 0, uma rotação à direita é suficiente para balancear a árvore.



Balanceamento de Árvores AVL

Caso 3 (LR)

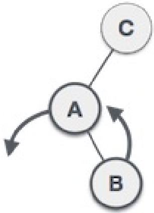
- Se a raiz da árvore tem fator de equilíbrio 2 e seu filho da esquerda possui fator de equilíbrio < 0, uma rotação à esquerda seguida de uma a direita é suficiente para balancear a árvore.



Balanceamento de Árvores AVL

Caso 3 (LR)

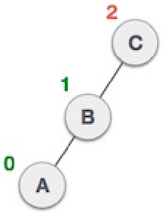
- Se a raiz da árvore tem fator de equilíbrio 2 e seu filho da esquerda possui fator de equilíbrio < 0, uma rotação à esquerda seguida de uma a direita é suficiente para balancear a árvore.



Balanceamento de Árvores AVL

Caso 3 (LR)

- Se a raiz da árvore tem fator de equilíbrio 2 e seu filho da esquerda possui fator de equilíbrio < 0, uma rotação à esquerda seguida de uma a direita é suficiente para balancear a árvore.



Balanceamento de Árvores AVL

Caso 3 (LR)

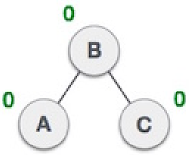
- Se a raiz da árvore tem fator de equilíbrio 2 e seu filho da esquerda possui fator de equilíbrio < 0, uma rotação à esquerda seguida de uma a direita é suficiente para balancear a árvore.

figuras/avl-tree-case-3-4.jpg

Balanceamento de Árvores AVL

Caso 3 (LR)

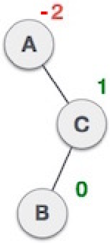
- Se a raiz da árvore tem fator de equilíbrio 2 e seu filho da esquerda possui fator de equilíbrio < 0, uma rotação à esquerda seguida de uma a direita é suficiente para balancear a árvore.



Balanceamento de Árvores AVL

Caso 4 (RL)

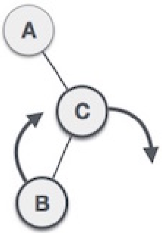
- Se raiz da árvore tem fator de equilíbrio -2 e seu filho da esquerda possui fator de equilíbrio > 0, uma rotação à direita seguida de uma à esquerda é suficiente para balancear a árvore.



Balanceamento de Árvores AVL

Caso 4 (RL)

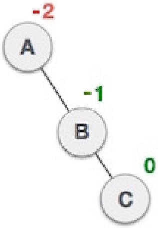
- Se raiz da árvore tem fator de equilíbrio -2 e seu filho da esquerda possui fator de equilíbrio > 0, uma rotação à direita seguida de uma à esquerda é suficiente para balancear a árvore.



Balanceamento de Árvores AVL

Caso 4 (RL)

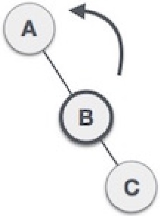
- Se raiz da árvore tem fator de equilíbrio -2 e seu filho da esquerda possui fator de equilíbrio > 0, uma rotação à direita seguida de uma à esquerda é suficiente para balancear a árvore.



Balanceamento de Árvores AVL

Caso 4 (RL)

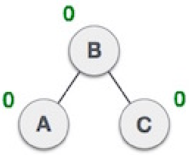
- Se raiz da árvore tem fator de equilíbrio -2 e seu filho da esquerda possui fator de equilíbrio > 0, uma rotação à direita seguida de uma à esquerda é suficiente para balancear a árvore.



Balanceamento de Árvores AVL

Caso 4 (RL)

- Se raiz da árvore tem fator de equilíbrio -2 e seu filho da esquerda possui fator de equilíbrio > 0, uma rotação à direita seguida de uma à esquerda é suficiente para balancear a árvore.



Balanceamento de Árvores AVL

AVL Applet

Sumário

Sumário

2 Implementação
- Definição
- Inicialização
- Funções auxiliares
- Busca
- Inserção
- Remoção
- Limpeza
- Análise

Definição

```c
typedef struct avl_node_t {
    int data;
    size_t height;
    struct avl_node_t *left;
    struct avl_node_t *right;
} avl_node_t;
```

Definição

```
typedef struct avl_tree_t {
    struct avl_node_t *root;
    size_t size;
} avl_tree_t;
```

Sumário

Inicialização

```c
void avl_tree_initialize(avl_tree_t **t) {
    (*t) = mallocx(sizeof(avl_tree_t));
    (*t)->root = NULL;
    (*t)->size = 0;
}
```

Sumário

Funções auxiliares

Devolve o tamanho (número de nós) em uma árvore AVL.

```
size_t avl_tree_size(avl_tree_t *t) {
    return t->size;
}
```

Funções auxiliares

Cria um novo nó.

```c
static avl_node_t *avl_new_node(int data) {
    avl_node_t *new_node = mallocx(sizeof(avl_node_t));
    new_node->height = 1;
    new_node->left = NULL;
    new_node->right = NULL;
    new_node->data = data;
    return new_node;
}
```

Funções auxiliares

Deleta um nó.

```
static void avl_tree_delete_node(avl_node_t *t) {
    free(t);
}
```

Funções auxiliares

Retorna a altura de uma árvore com raiz em v.

```
static size_t avl_node_get_height(avl_node_t *v) {
    if (v == NULL) {
        return 0;
    }
    return v->height;
}
```

Funções auxiliares

Calcula a altura de uma árvore com raiz em v a partir dos seus filhos.

```
static size_t avl_calculate_height(avl_node_t *v) {
    size_t hl, hr;
    if (v == NULL) {
        return 0;
    }
    hl = avl_node_get_height(v->left);
    hr = avl_node_get_height(v->right);
    return hl > hr ? hl + 1 : hr + 1;
}
```

Funções auxiliares

Obtém o fator de equilíbrio da árvore com raiz em v.

```
static int avl_node_get_balance(avl_node_t *v) {
    if (v == NULL) {
        return 0;
    }
    return ((int)avl_node_get_height(v->left) - avl_node_get_heig
}
```

Funções auxiliares

Realiza a rotação para a esquerda da árvore com raiz em x e devolve a nova raiz.

```
static avl_node_t *avl_left_rotate(avl_node_t *x) {
    assert(x != NULL);
    avl_node_t *y = x->right;
    assert(y != NULL);
    x->right = y->left;
    y->left = x;
    x->height = avl_calculate_height(x);
    y->height = avl_calculate_height(y);
    return y;
}
```

Funções auxiliares

Realiza a rotação para a direita da árvore com raiz em y e devolve a nova raiz.

```
static avl_node_t *avl_right_rotate(avl_node_t *y) {
    assert(y != NULL);
    avl_node_t *x = y->left;
    assert(x != NULL);
    y->left = x->right;
    x->right = y;
    y->height = avl_calculate_height(y);
    x->height = avl_calculate_height(x);
    return x;
}
```

Funções auxiliares

Balanceia uma árvore com raiz em v.

```
static avl_node_t *balance(avl_node_t *v) {
    int balance = avl_node_get_balance(v);
    if (balance > 1 && avl_node_get_balance(v->left) >= 0) {
        v = avl_right_rotate(v);
    } else if (balance < -1 && avl_node_get_balance(v->right) <= 0) {
        v = avl_left_rotate(v);
    } else if (balance > 1 && avl_node_get_balance(v->left) < 0) {
        v->left = avl_left_rotate(v->left);
        v = avl_right_rotate(v);
    } else if (balance < -1 && avl_node_get_balance(v->right) > 0) {
        v->right = avl_right_rotate(v->right);
        v = avl_left_rotate(v);
    }
    return v;
}
```

Sumário

Busca

```
bool avl_tree_find(avl_tree_t *t, int data) {
    return avl_tree_find_helper(t->root, data);
}
```

Busca

```c
static bool avl_tree_find_helper(avl_node_t *v, int data) {
    if (v == NULL) {
        return false;
    }
    if (data < v->data) {
        return avl_tree_find_helper(v->left, data);
    } else if (data > v->data) {
        return avl_tree_find_helper(v->right, data);
    }
    return true;
}
```

Sumário

Inserção

Insere o dado estipulado na árvore. Premissa: o dado não existe como chave na árvore.

```
void avl_tree_insert(avl_tree_t *t, int data) {
    t->root = avl_tree_insert_helper(t->root, data);
    t->size++;
}
```

Inserção

Insere o dado estipulado na árvore. Premissa: o dado não existe como chave na árvore.

```
avl_node_t *avl_tree_insert_helper(avl_node_t *v, int data) {
    if (v == NULL) {
        v = avl_new_node(data);
        v->height = avl_calculate_height(v);
        return v;
    }
    assert(v->data != data);
    if (data < v->data) {
        v->left = avl_tree_insert_helper(v->left, data);
    } else {
        v->right = avl_tree_insert_helper(v->right, data);
    }
    v->height = avl_calculate_height(v);
    v = balance(v);
    return v;
}
```

Sumário

Remoção

Remove o dado estipulado da árvore. Premissa: o dado existe como chave na árvore.

```
void avl_tree_remove(avl_tree_t *t, int data) {
    t->root = avl_tree_remove_helper(t->root, data);
    t->size--;
}
```

Remoção

Remove o dado estipulado da árvore. Premissa: o dado existe como chave na árvore.

```
avl_node_t *avl_tree_remove_helper(avl_node_t *v, int data) {
    assert(v != NULL);
    if (data < v->data) {
        v->left = avl_tree_remove_helper(v->left, data);
    } else if (data > v->data) {
        v->right = avl_tree_remove_helper(v->right, data);
    } else { /*remoção do nó*/
        if (v->left == NULL) {
            avl_node_t *tmp = v->right;
            avl_tree_delete_node(v);
            return tmp;
        } else if (v->right == NULL) {
            avl_node_t *tmp = v->left;
            avl_tree_delete_node(v);
            return tmp;
        } else {
            avl_node_t *previous_v = avl_tree_find_rightmost(v->left);
            int aux = v->data;
            v->data = previous_v->data;
            previous_v->data = aux;
            v->left = avl_tree_remove_helper(v->left, previous_v->data);
        }
    }
```

Remoção

Remove o dado estipulado da árvore. Premissa: o dado existe como chave na árvore.

```
    if (v != NULL) {
        v->height = avl_calculate_height(v);
        v = balance(v);
    }
    return v;
}
```

Sumário

Limpeza

Deleta a árvore.

```
void avl_tree_delete(avl_tree_t **t) {
    avl_tree_delete_helper((*t)->root);
    free(*t);
    (*t) = NULL;
}
```

Limpeza

Deleta a árvore.

```
static void avl_tree_delete_helper(avl_node_t *v) {
    if (v != NULL) {
        avl_tree_delete_helper(v->left);
        avl_tree_delete_helper(v->right);
        avl_tree_delete_node(v);
    }
}
```

Sumário

Análise

	Busca	Inserção	Remoção
BST	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$
AVL	$\Theta(\lg n)$	$\Theta(\lg n)$	$\Theta(\lg n)$

Sumário

Considerações

- A árvore AVL aumenta a BST ao incluir uma estratégia de balanceamento por altura.
- Para alcançar esse objetivo, utiliza de rotações à esquerda e à direita, que conservam a propriedade de BST.
- Com esta estratégia, a altura máxima da árvore é $\Theta(\lg n)$.
- Implementá-la é um pouco mais complicado que implementar uma BST, mas o esforço vale a pena a longo prazo.