ÁRVORE AVL

DCE792 - AEDs II (Prática)

Atualizado em: 12 de outubro de 2023

Iago Carvalho

Departamento de Ciência da Computação

ÁRVORE AVL

Uma árvore AVL é um exemplo de árvore balanceada

Ela tem seu nome devido a seus criadores

- Georgy **A**delson-**V**elsky
- Yevgeniy **L**andis

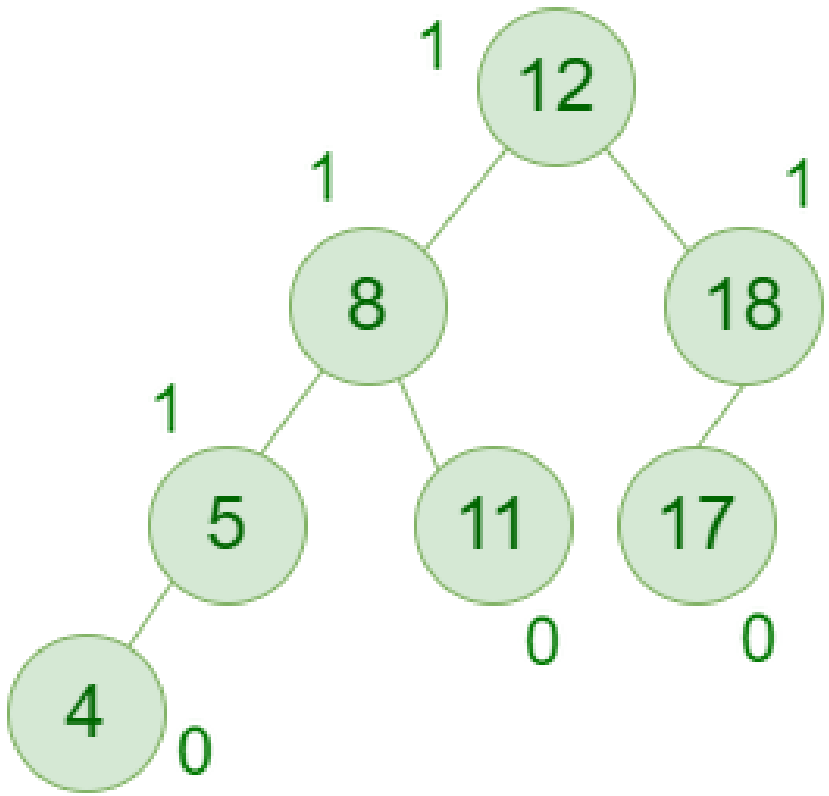
Estas árvores garantem seu balanceamento utilizando a diferença entre a altura da sub-árvore a esquerda e da sub-árvore a direita

- Esta diferença nunca pode ser maior do que 1
- Diferença garantida na inserção e remoção de items

ÁRVORE AVL BALANCEADA

Abaixo podemos ver o exemplo de uma árvore AVL balanceada

- Notem que a diferença entre as alturas da sub-árvore a esquerda e da sub-árvore a direita nunca é maior do que 1



COMPLEXIDADE DAS OPERAÇÕES EM UMA AVL

Em uma árvore binária não-balanceada, temos que o pior caso de inserção e remoção tem complexidade $\mathcal{O}(n)$

- Casos onde a árvore se degenera em uma lista encadeada

Uma árvore AVL não possui este pior caso devido ao balanceamento de suas sub-árvores

	Caso médio	Pior caso
Espaço	$\mathcal{O}(n)$	$\mathcal{O}(n)$
Inserção	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$
Remoção	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$
Pesquisa	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$

APLICAÇÕES, VANTAGENS E DESVANTAGENS DA AVL

Aplicações

- Úteis para armazenar chaves de busca de grandes bancos de dados
- Utilizados em estruturas de dados como conjuntos, dicionários e multi-conjuntos
- Indexação de dados onde inserção e remoção não são muito comuns, mas a pesquisa é frequente
  - Caso contrário, utilizar árvores rubro-negras
- Softwares que precisam de pesquisa otimizada

Vantagens

- Auto-balanceada
- Pesquisa rápida
- Altura máxima garantida
  - $O(\log n)$

Desvantagens

- Difícil implementação
- Inserções e remoções são mais caras
  - Muitas rotações são necessárias

CRIANDO UMA ÁRVORE BALANCEADA

Existe um simples algoritmo recursivo para esta tarefa

- Também existe uma versão iterativa, mas é um pouquinho mais complicada

O algoritmo tem, como ideia, pegar o elemento central do vetor e setar como raiz

- Particionar o vetor restante em duas partes
- A primeira metade será a árvore a esquerda da raiz
- A segunda metade será a árvore a direita da raiz

O procedimento acima é realizado de forma recursiva até que a árvore inteira esteja montada

REPRESENTAÇÃO DE UMA ÁRVORE AVL

```
 5    struct Node {
 6        int valor;
 7        struct Node *esquerda;
 8        struct Node *direita;
 9        int altura;
10    };
```

valor: Elemento da árvore

esquerda: Apontador para o filho à esquerda

direita: Apontador para o filho à direita

altura: Altura da sub-árvore enraizada neste nó

OPERAÇÕES POSSÍVEIS COM ÁRVORES AVL

- struct node* novo_no(<valor>);
  - Cria um novo nó na árvore
- struct node* inserir(<valor>);
  - Insere um novo item na árvore
- struct node* buscar(<valor>);
  - Verifica se existe um determinado item na árvore
- struct node* remover(<valor>);
  - Remove um determinado item na árvore
- int verifica_balanceamento(<struct node*>);
  - Computa o balanceamento da árvore enraizada em *node*
- int altura(<struct node*>);
  - Obtem a altura da árvore enraizada em *node*

BUSCA E NOVO_NO

Operações de busca e alocação (criação de novo nó) são realizadas de maneira extremamente que em árvores binárias não balanceadas

Busca é exatamente igual

- Em ordem, pré-ordem ou pós-ordem

Alocação de novo nó precisa definir sua altura

- Como o novo nó sempre é uma folha, a altura sempre é 1

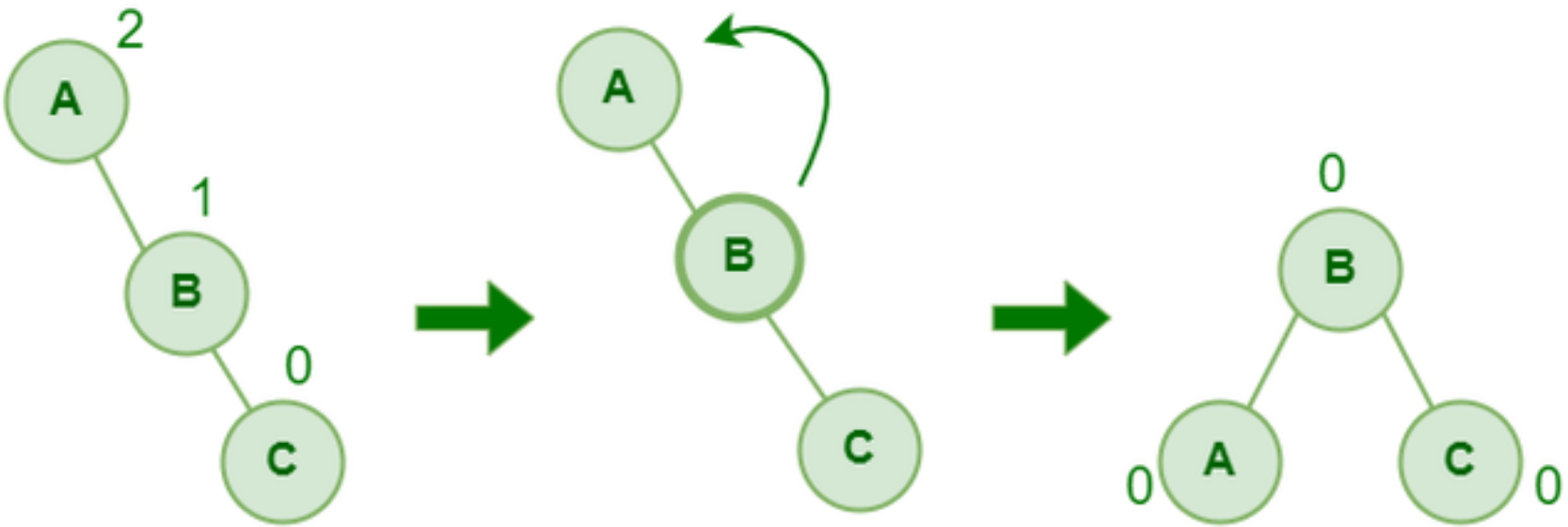
INSERÇÃO DE NOVO NÓ

Operação complexa, pois necessita de diversas rotações

1. Insere um nó utilizando o algoritmo de árvores binárias não balanceadas
2. Faz uma busca reversa (do nó inserido até a raiz) recomputando o fator de balanceamento dos nós
3. Caso exista um nó não-balanceado
   - Executa as rotações necessárias
   - Quatro casos diferentes
   - Pode-se resumir a rotações a direita e a esquerda
4. Caso todos os nós estejam balanceados
   - Inserção concluída com sucesso

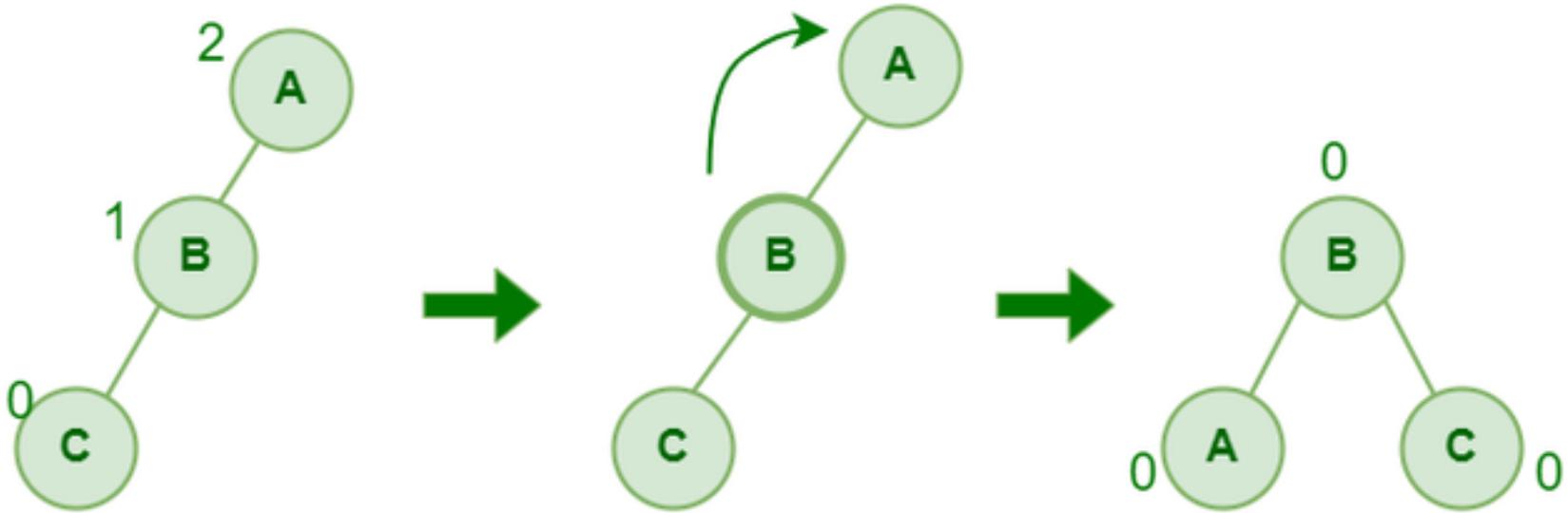
CASO ESQUERDA ESQUERDA

Inserção do nó *C* - Considere a ordem alfabética



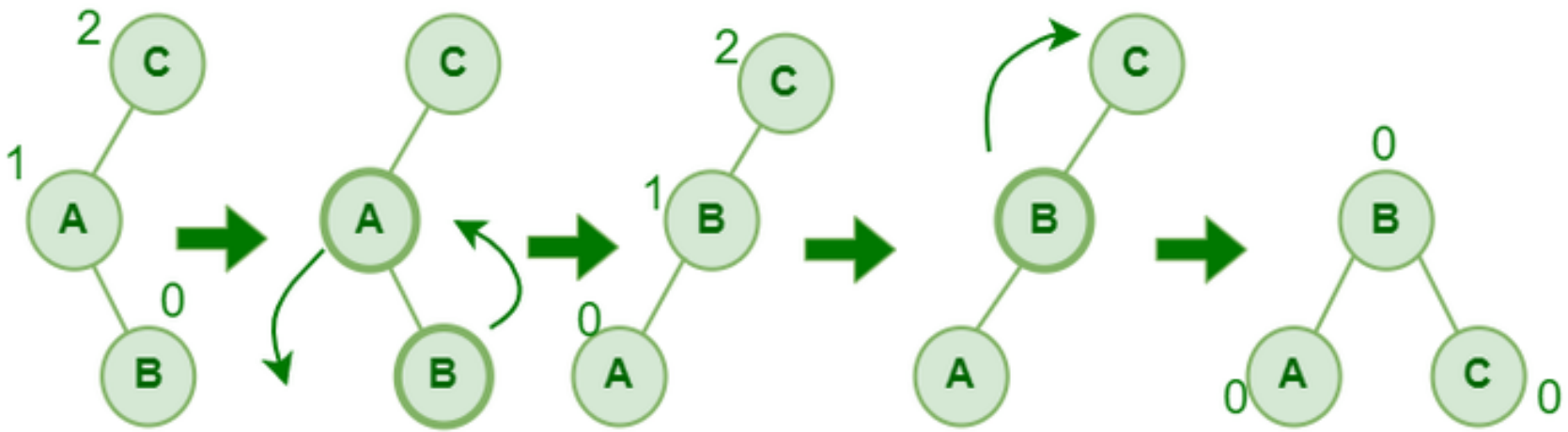
CASO DIREITA DIREITA

Inserção do nó *C* - Considere a ordem alfabética



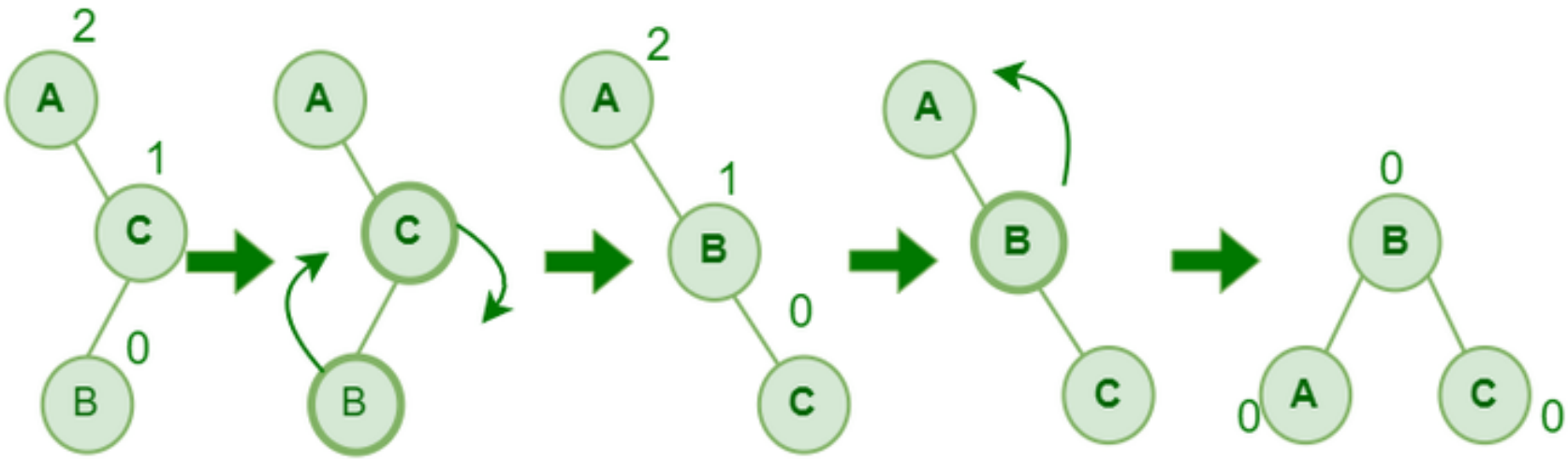
CASO ESQUERDA DIREITA

Inserção do nó *B* - Considere a ordem alfabética



CASO DIREITA ESQUERDA

Inserção do nó *B* - Considere a ordem alfabética



CÓDIGO DE ÁRVORES AVL

O código (quase) completo de uma árvore binária está disponível no Github

- Link

O algoritmo pode ser visualizado em Link

ATIVIDADE PRÁTICA

ATIVIDADE

O código disponibilizado só implementa a rotação a direita

Assim, deve-se implementar a rotação a esquerda

PRÓXIMA AULA:

REMOÇÃO DE NÓS EM ÁRVORES AVL