



Estruturas de Dados: Árvores

Professor: Elton Sarmanho¹

E-mail: eltonss@ufpa.br



¹Faculdade de Sistemas de Informação - UFPA/CUTINS

19 de outubro de 2025

Roteiro da Aula

Introdução às Árvores

- Objetivos de Aprendizagem
- Conceitos Fundamentais
- Nomenclatura Essencial

Árvore Binária

- Conceito e Estrutura
- Tipos de Árvores Binárias
- Árvore Binária de Busca (ABB)
- Operações em ABB
- Algoritmos de Percurso
- Implementação

Árvore AVL

- O Problema do Desbalanceamento
- Conceito de Árvore AVL
- Rotações em Árvores AVL

Conclusões

Referências Bibliográficas



Objetivos de Aprendizagem

- ▶ Ao final desta aula, você será capaz de responder às seguintes perguntas:
 - ▶ O que é uma árvore e como ela se difere de estruturas de dados lineares?
 - ▶ O que caracteriza uma árvore binária e uma árvore binária de busca?
 - ▶ Em que cenários uma árvore binária de busca é uma estrutura de dados eficiente?
 - ▶ Quais são os três principais algoritmos de percurso em árvores binárias?
 - ▶ O que é uma árvore AVL e por que o balanceamento de altura é importante?
 - ▶ Como funcionam as rotações para manter o balanceamento em uma árvore AVL?
 - ▶ O que é e onde podemos aplicar uma árvore de decisão?



Objetivos de Aprendizagem

- ▶ Ao final desta aula, você será capaz de responder às seguintes perguntas:
 - ▶ O que é uma árvore e como ela se difere de estruturas de dados lineares?
 - ▶ O que caracteriza uma árvore binária e uma árvore binária de busca?
 - ▶ Em que cenários uma árvore binária de busca é uma estrutura de dados eficiente?
 - ▶ Quais são os três principais algoritmos de percurso em árvores binárias?
 - ▶ O que é uma árvore AVL e por que o balanceamento de altura é importante?
 - ▶ Como funcionam as rotações para manter o balanceamento em uma árvore AVL?
 - ▶ O que é e onde podemos aplicar uma árvore de decisão?



Objetivos de Aprendizagem

- ▶ Ao final desta aula, você será capaz de responder às seguintes perguntas:
 - ▶ O que é uma árvore e como ela se difere de estruturas de dados lineares?
 - ▶ O que caracteriza uma árvore binária e uma árvore binária de busca?
 - ▶ Em que cenários uma árvore binária de busca é uma estrutura de dados eficiente?
 - ▶ Quais são os três principais algoritmos de percurso em árvores binárias?
 - ▶ O que é uma árvore AVL e por que o balanceamento de altura é importante?
 - ▶ Como funcionam as rotações para manter o balanceamento em uma árvore AVL?
 - ▶ O que é e onde podemos aplicar uma árvore de decisão?



Objetivos de Aprendizagem

- ▶ Ao final desta aula, você será capaz de responder às seguintes perguntas:
 - ▶ O que é uma árvore e como ela se difere de estruturas de dados lineares?
 - ▶ O que caracteriza uma árvore binária e uma árvore binária de busca?
 - ▶ Em que cenários uma árvore binária de busca é uma estrutura de dados eficiente?
 - ▶ Quais são os três principais algoritmos de percurso em árvores binárias?
 - ▶ O que é uma árvore AVL e por que o balanceamento de altura é importante?
 - ▶ Como funcionam as rotações para manter o balanceamento em uma árvore AVL?
 - ▶ O que é e onde podemos aplicar uma árvore de decisão?



Objetivos de Aprendizagem

- ▶ Ao final desta aula, você será capaz de responder às seguintes perguntas:
 - ▶ O que é uma árvore e como ela se difere de estruturas de dados lineares?
 - ▶ O que caracteriza uma árvore binária e uma árvore binária de busca?
 - ▶ Em que cenários uma árvore binária de busca é uma estrutura de dados eficiente?
 - ▶ Quais são os três principais algoritmos de percurso em árvores binárias?
 - ▶ O que é uma árvore AVL e por que o balanceamento de altura é importante?
 - ▶ Como funcionam as rotações para manter o balanceamento em uma árvore AVL?
 - ▶ O que é e onde podemos aplicar uma árvore de decisão?



Objetivos de Aprendizagem

- ▶ Ao final desta aula, você será capaz de responder às seguintes perguntas:
 - ▶ O que é uma árvore e como ela se difere de estruturas de dados lineares?
 - ▶ O que caracteriza uma árvore binária e uma árvore binária de busca?
 - ▶ Em que cenários uma árvore binária de busca é uma estrutura de dados eficiente?
 - ▶ Quais são os três principais algoritmos de percurso em árvores binárias?
 - ▶ O que é uma árvore AVL e por que o balanceamento de altura é importante?
 - ▶ Como funcionam as rotações para manter o balanceamento em uma árvore AVL?
 - ▶ O que é e onde podemos aplicar uma árvore de decisão?



Objetivos de Aprendizagem

- ▶ Ao final desta aula, você será capaz de responder às seguintes perguntas:
 - ▶ O que é uma árvore e como ela se difere de estruturas de dados lineares?
 - ▶ O que caracteriza uma árvore binária e uma árvore binária de busca?
 - ▶ Em que cenários uma árvore binária de busca é uma estrutura de dados eficiente?
 - ▶ Quais são os três principais algoritmos de percurso em árvores binárias?
 - ▶ O que é uma árvore AVL e por que o balanceamento de altura é importante?
 - ▶ Como funcionam as rotações para manter o balanceamento em uma árvore AVL?
 - ▶ O que é e onde podemos aplicar uma árvore de decisão?



Objetivos de Aprendizagem

- ▶ Ao final desta aula, você será capaz de responder às seguintes perguntas:
 - ▶ O que é uma árvore e como ela se difere de estruturas de dados lineares?
 - ▶ O que caracteriza uma árvore binária e uma árvore binária de busca?
 - ▶ Em que cenários uma árvore binária de busca é uma estrutura de dados eficiente?
 - ▶ Quais são os três principais algoritmos de percurso em árvores binárias?
 - ▶ O que é uma árvore AVL e por que o balanceamento de altura é importante?
 - ▶ Como funcionam as rotações para manter o balanceamento em uma árvore AVL?
 - ▶ O que é e onde podemos aplicar uma árvore de decisão?



Por que usar Árvores?

- ▶ Estruturas de dados como **listas**, **pilhas** e **filas** são **lineares**. Elas organizam os dados em uma sequência única.
- ▶ No entanto, muitos problemas do mundo real envolvem **hierarquia**, como:
 - ▶ Estrutura de diretórios de um sistema de arquivos.
 - ▶ Organograma de uma empresa.
 - ▶ Genealogia de uma família.
- ▶ Uma **árvore** é uma estrutura de dados não linear, ideal para modelar relações hierárquicas.

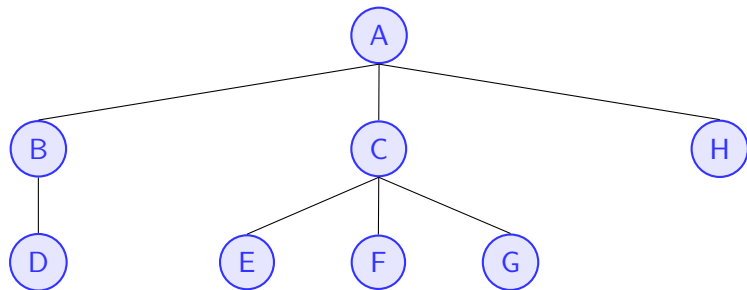


Definição Formal de uma Árvore

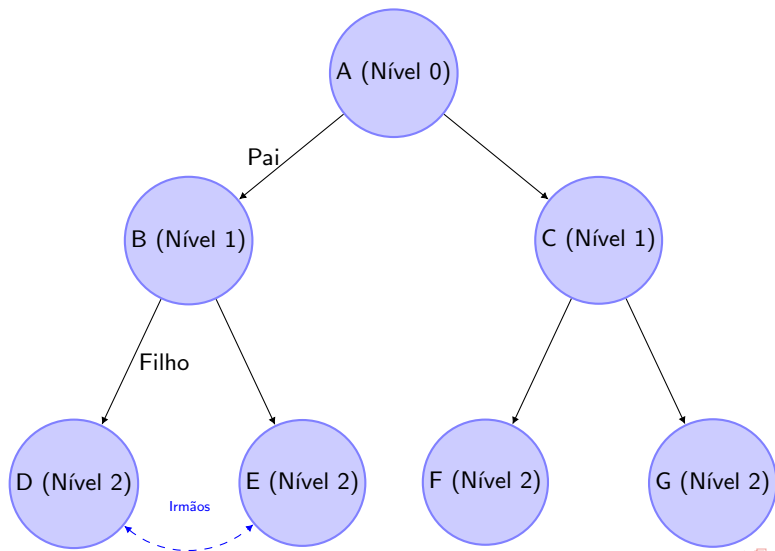
- ▶ Uma árvore pode ser definida recursivamente como um conjunto de nós.
- ▶ Ou a árvore está vazia, ou ela consiste em um nó especial chamado **raiz** e zero ou mais subárvores, onde cada raiz de subárvore é conectada à raiz principal por uma aresta.
- ▶ Formalmente: Uma árvore $T = (r, S)$, onde r é um nó (a raiz) e S é um conjunto de árvores disjuntas $\{T_1, T_2, \dots, T_n\}$, chamadas de subárvores de r .



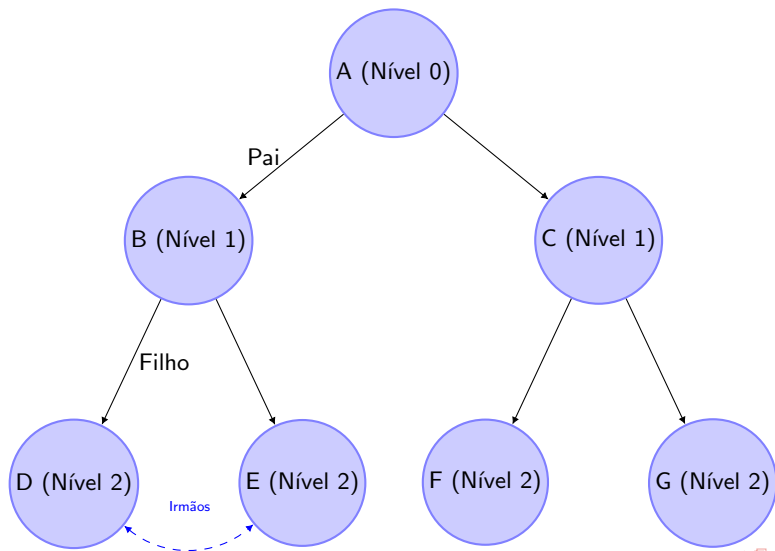
Definição Formal de uma Árvore



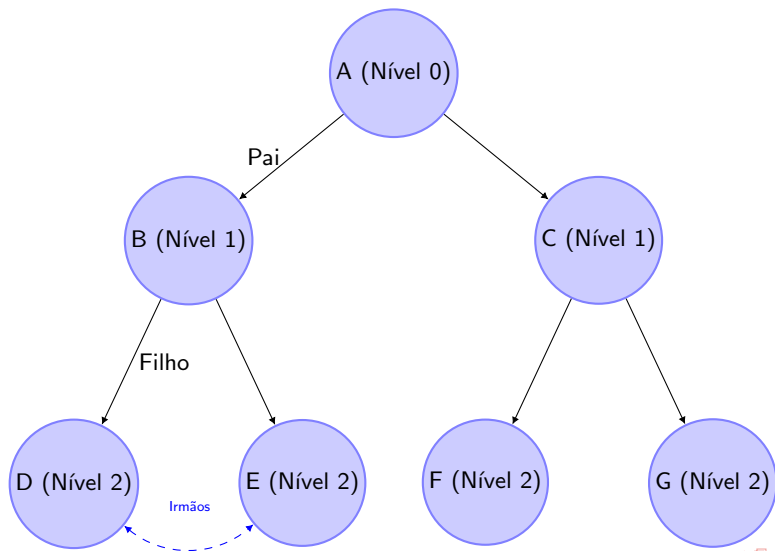
- **Nó Raiz:** O único nó que não possui um "pai". (A)



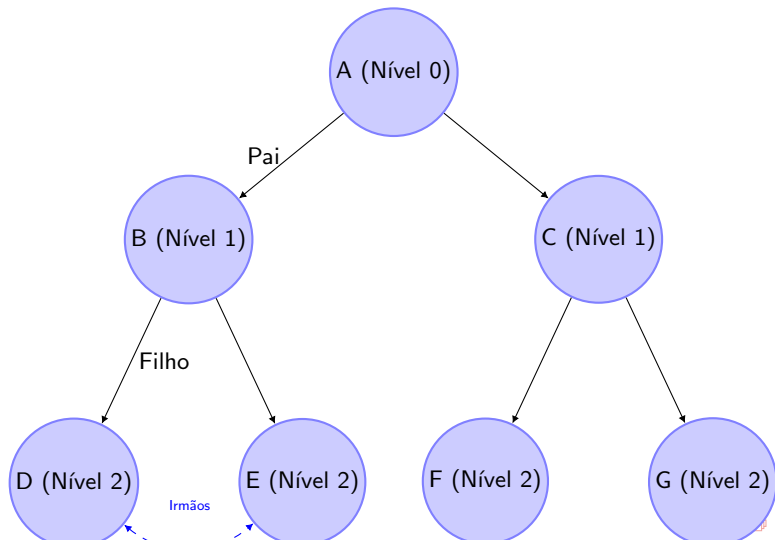
- **Nó Folha (ou Terminal):** Um nó que não possui filhos. (D, E, F, G)



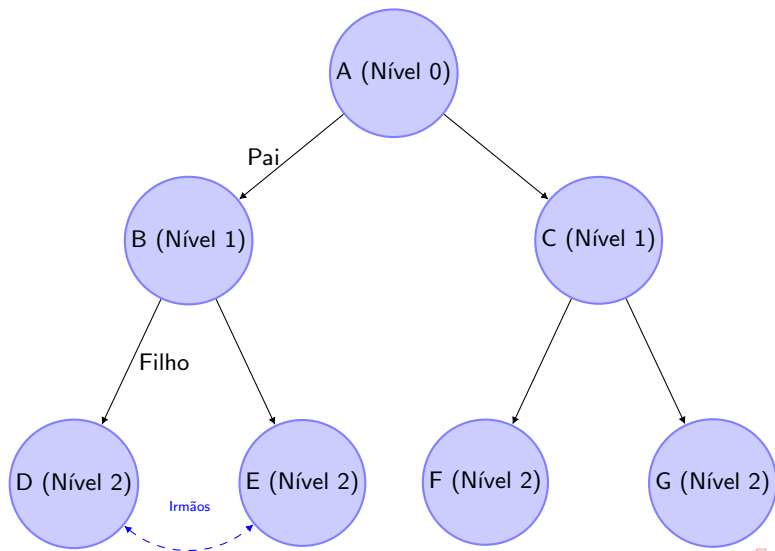
- **Nó Interno:** Um nó que possui pelo menos um filho. (A, B, C)



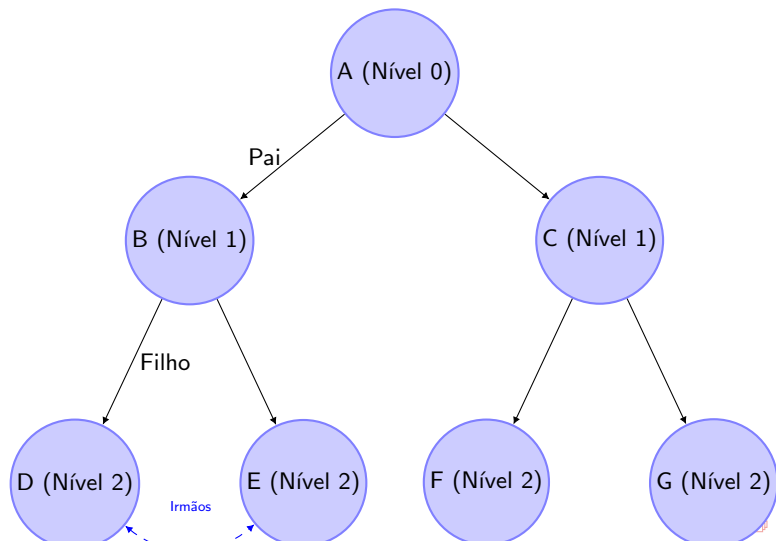
- **Pai e Filho:** Se o nó X está diretamente conectado ao nó Y abaixo dele, X é pai de Y. (B é pai de D e E)



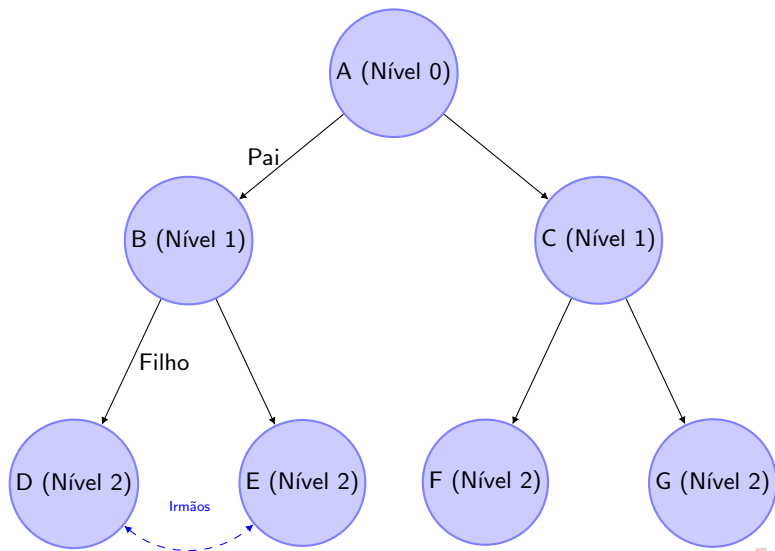
- **Irmãos:** Nós que compartilham o mesmo pai. (D e E são irmãos)



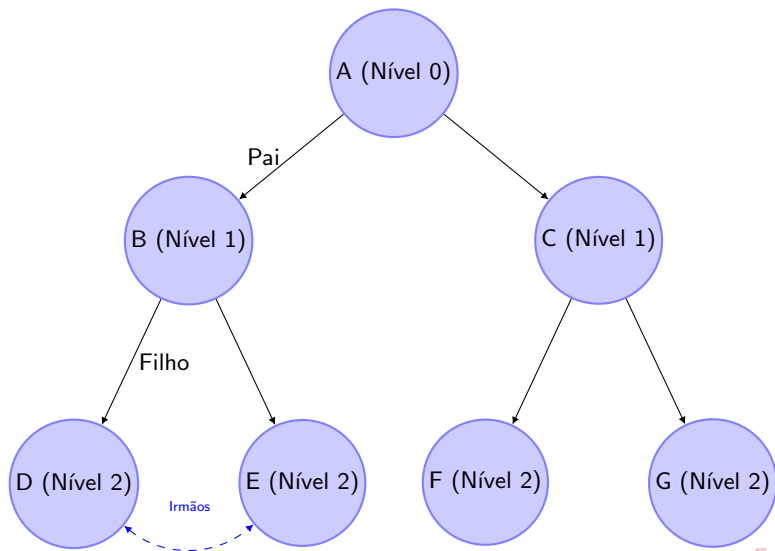
- **Ancestral e Descendente:** Um nó 'u' é ancestral de 'v' se 'u' está no caminho da raiz até 'v'. (A é ancestral de todos)



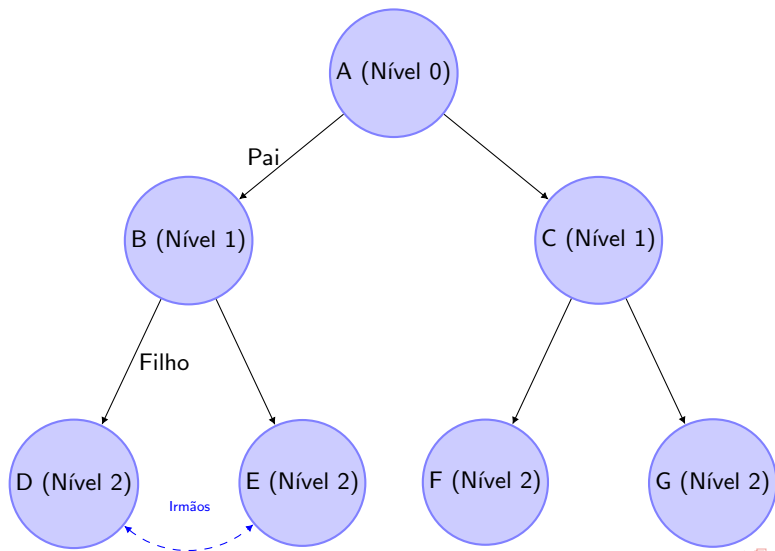
- **Grau de um Nó:** O número de filhos que um nó possui. (Grau de B é 2)



- **Nível de um Nó:** A distância da raiz até o nó. O nível da raiz é 0.

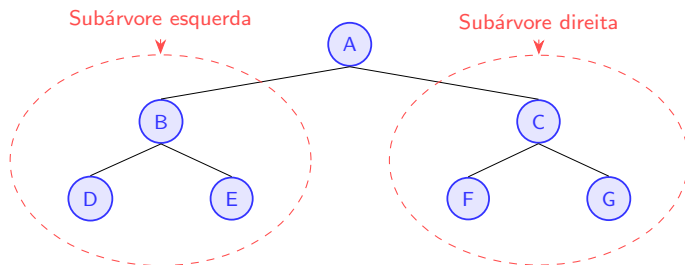


- **Altura da Árvore:** O maior nível entre todas as folhas.



Árvore Binária

- ▶ É um tipo especial de árvore onde cada nó interno pode ter **no máximo 2 filhos**.
- ▶ Esses filhos são distintos e referenciados como **LST** e **RST**.
- ▶ Formalmente, uma árvore binária $T = (x, L, R)$, onde x é a raiz, e L e R são árvores binárias disjuntas (subárvore esquerda e subárvore direita, respectivamente).



Árvore Estritamente Binária

Definição

Uma árvore é dita **estritamente binária** se todo nó possui 0 filhos (sendo uma folha) ou exatamente 2 filhos. Nenhum nó possui apenas 1 filho.

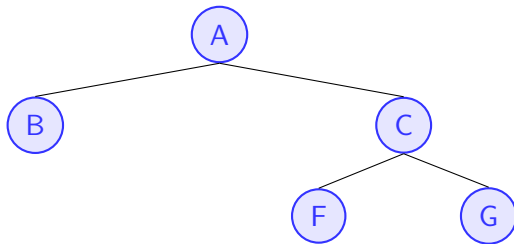


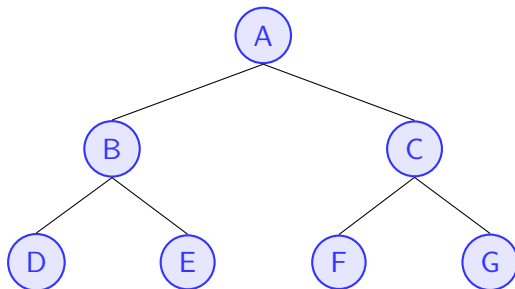
Figura: Neste exemplo, todos os nós têm 0 ou 2 filhos, satisfazendo a condição de uma árvore estritamente binária.



Árvore Binária Cheia (ou Perfeita)

Definição

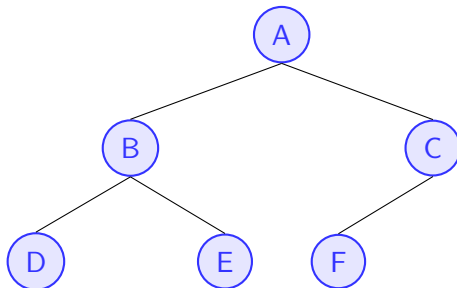
Se todos os seus nós internos possuem exatamente 2 filhos e todas as suas folhas estão no mesmo nível. Uma árvore perfeita de altura h possui $2^{h+1} - 1$ nós



Árvore Binária Completa

Definição

Uma árvore binária é **completa** se todos os seus níveis, exceto possivelmente o último, estão completamente preenchidos, e todos os nós no último nível estão posicionados o mais à esquerda possível.



└ Árvore Binária

└ Árvore Binária de Busca (ABB)

Árvore Binária de Busca (ABB)

- ▶ A ABB (ou BST - Binary Search Tree) é uma árvore binária com uma propriedade de ordem fundamental:

Para qualquer nó x :

- ▶ Todos os valores na subárvore **esquerda** de x são **menores** que o valor de x .
- ▶ Todos os valores na subárvore **direita** de x são **maiores ou iguais** ao valor de x .
- ▶ Essa propriedade torna a busca por elementos extremamente eficiente.

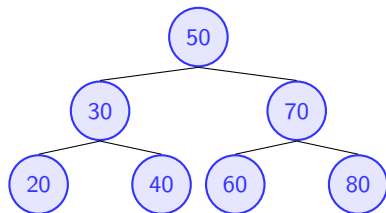


Figura: Exemplo de ABB. Note que para a raiz (50), todos os nós à esquerda (20, 30, 40) são menores, e todos à direita (60, 70, 80) são maiores.



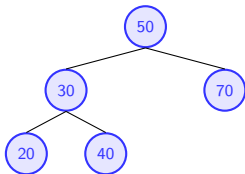
Operação de Inserção em ABB

- ▶ A inserção sempre começa na raiz e respeita a propriedade da ABB.
- ▶ O novo nó é sempre inserido como uma nova folha.
- ▶ **Processo:**
 1. Compare o valor a ser inserido com o nó atual.
 2. Se o valor for menor, desça para a subárvore esquerda.
 3. Se o valor for maior ou igual, desça para a subárvore direita.
 4. Repita até encontrar uma posição nula, onde o novo nó será inserido.
- ▶ **Complexidade:**
 - ▶ **Caso Médio (árvore balanceada):** $O(\log n)$
 - ▶ **Pior Caso (árvore degenerada):** $O(n)$



Inserção em Árvore Binária de Busca (ABB)

1. Árvore Inicial



2. Objetivo

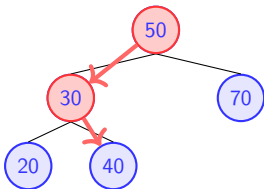
- ▶ Inserir um novo nó com o valor **45** na árvore.
- ▶ A propriedade da ABB deve ser mantida.

3. Processo

- ▶ O processo sempre começa pela raiz.
- ▶ Comparamos o valor a ser inserido com o nó atual para decidir se vamos para a subárvore esquerda ou direita.



Inserção do Nó 45: Passos 1 e 2



Passo 1: Comparar com a Raiz

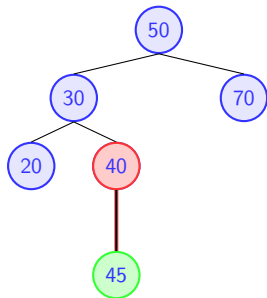
- ▶ Valor a inserir: **45**
- ▶ Nó atual: **50**
- ▶ Como $45 < 50$, descemos para a **subárvore esquerda**.

Passo 2: Comparar com o Nó 30

- ▶ Nó atual: **30**
- ▶ Como $45 > 30$, descemos para a **subárvore direita**.



Inserção do Nó 45: Passos 3 e 4



Passo 3: Comparar com o Nó 40

- ▶ Nó atual: **40**
- ▶ Como $45 > 40$, tentamos descer para a **subárvore direita**.

Passo 4: Posição Vazia Encontrada

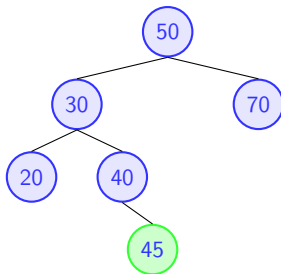
- ▶ A subárvore direita do nó 40 está vazia (NULL).
- ▶ **Inserimos o novo nó 45**



Resultado Final da Inserção

Árvore Após Inserir o Nó 45

O nó 45 foi inserido como filho direito do nó 40, e a propriedade da Árvore Binária de Busca foi mantida em toda a estrutura.



Operação de Remoção em ABB

A remoção é a operação mais complexa, pois a propriedade da ABB deve ser mantida. Existem três casos:

1. **O nó a ser removido é uma folha:** Simplesmente remova o nó.
2. **O nó a ser removido tem um único filho:** Substitua o nó por seu único filho.
3. **O nó a ser removido tem dois filhos:**
 - ▶ Encontre o **sucessor em ordem** do nó (o menor valor na subárvore direita) ou o **predecessor em ordem** (o maior valor na subárvore esquerda).
 - ▶ Copie o valor do sucessor/predecessor para o nó que está sendo removido.
 - ▶ Remova o nó sucessor/predecessor (que agora se torna um problema mais simples, caso 1 ou 2).

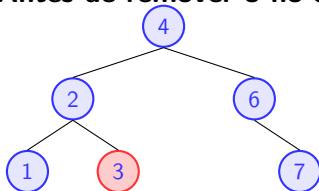


Remoção em ABB: Caso 1 (Nó Folha)

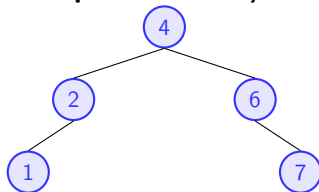
Regra

Se o nó a ser removido não tem filhos (é uma folha), ele é simplesmente desconectado de seu pai. Esta é a remoção mais simples.

Antes de remover o nó 3



Depois da remoção

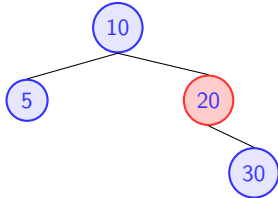


Remoção em ABB: Caso 2 (Um Filho)

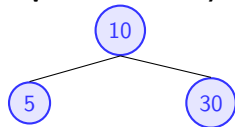
Regra

Se o nó a ser removido tem um único filho, a ligação de seu "avô" é redirecionada para seu "neto", pulando o nó removido.

Antes de remover o nó 20



Depois da remoção



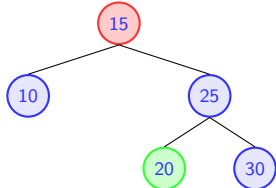
Remoção em ABB: Caso 3 (Dois Filhos)

Regra

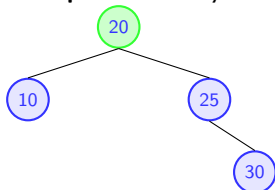
Se o nó a ser removido tem dois filhos, o processo é mais complexo:

1. Encontre o **sucessor em ordem** do nó (o menor valor na subárvore direita).
2. Copie o valor do sucessor para o nó que está sendo removido.
3. Remova o nó sucessor original (que agora será um problema do Caso 1 ou 2).

Antes de remover o nó 15



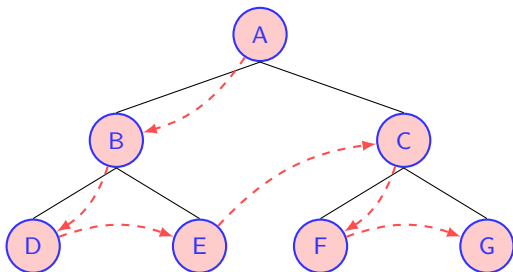
Depois da remoção



Percurso em Pré-Ordem (Raiz, Esquerda, Direita)

Regra

1. Visita a **Raiz**
2. Percorre a subárvore **Esquerda**
3. Percorre a subárvore **Direita**



Resultado

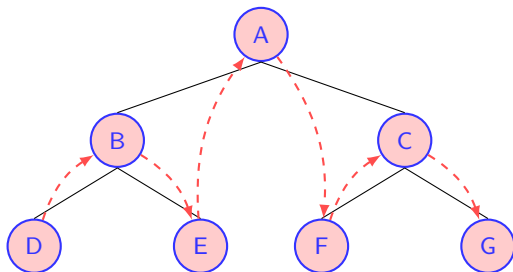
Ordem: A, B, D, E, C, F, G



Percurso em Ordem (Esquerda, Raiz, Direita)

Regra

1. Percorre a subárvore **Esquerda**
2. Visita a **Raiz**
3. Percorre a subárvore **Direita**



Resultado

Ordem: D, B, E, A, F, C, G



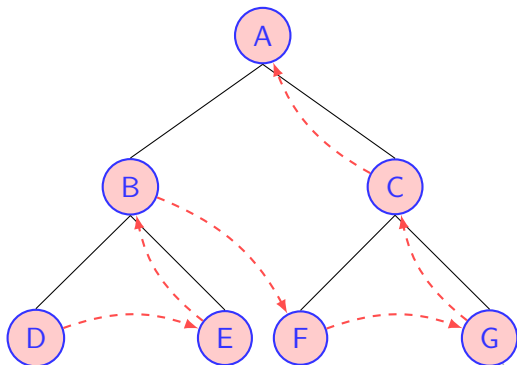
Percurso em Pós-Ordem (Esquerda, Direita, Raiz)

Regra

1. Percorre a subárvore **Esquerda**
2. Percorre a subárvore **Direita**
3. Visita a **Raiz**

Resultado

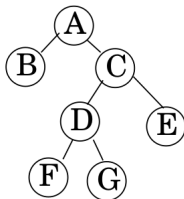
Ordem: D, E, B, F, G, C, A



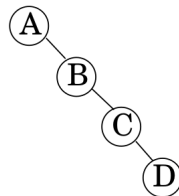
Exercício C

- Descreve Percurso pré-ordem, Ordem e Pós-ordem das árvores

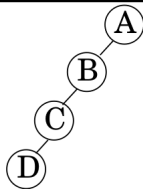
1.



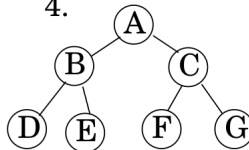
3.



2.



4.



- └ Árvore Binária

- └ Implementação

Implementação em Python (ABB)

Python Code

C Code

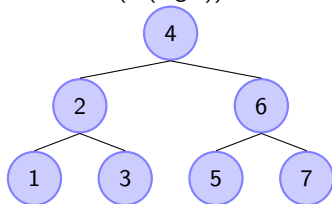


└ Árvore AVL

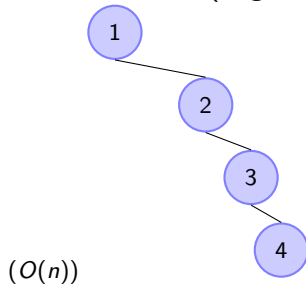
└ O Problema do Desbalanceamento

- ▶ O grande benefício da ABB, a busca em tempo $O(\log n)$, depende da árvore ser **balanceada**.
- ▶ Se inserirmos elementos em ordem crescente (ex: 1, 2, 3, 4, 5), a árvore se torna uma **lista encadeada**.

Árvore Balanceada
($O(\log n)$)



Árvore Desbalanceada (Degenerada)



Solução

Usar árvores de busca binária autobalanceáveis, como a Árvore AVL.



└ Árvore AVL

└ Conceito de Árvore AVL

Árvore AVL: Definição

- ▶ Uma árvore AVL (Adelson-Velskii e Landis) é uma Árvore Binária de Busca com uma propriedade de balanceamento adicional.
- ▶ **Propriedade AVL:** Para qualquer nó na árvore, a **diferença de altura** entre suas subárvores esquerda e direita é no máximo 1.
- ▶ Essa diferença é chamada de **Fator de Balanceamento (FB)**.

$$FB(no) = altura(subarvore_direita) - altura(subarvore_esquerda)$$

- ▶ Em uma árvore AVL, o FB de todo nó deve ser **-1, 0 ou 1**.
- ▶ Se, após uma inserção ou remoção, o FB de algum nó se torna -2 ou 2, a árvore precisa ser **rebalanceada** através de operações chamadas **rotações**.



Rebalanceamento com Rotações

As rotações são ajustes estruturais locais para restaurar a propriedade de balanceamento da AVL. Existem casos de desbalanceamento, que são resolvidos com dois tipos de rotações (simples e dupla).

Tabela de rotação

Diferença de altura de um nodo	Diferença de altura do nodo filho do nodo desbalanceado	Tipo de Rotação
2	1	Simples à esquerda
	0	Simples à esquerda
	-1	Dupla com filho para direita e pai para esquerda
-2	1	Dupla com filho para esquerda e pai para direita
	0	Simples à direita
	-1	Simples à direita



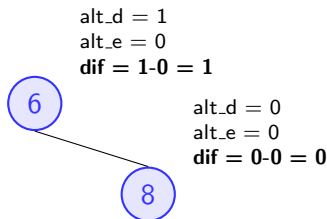
└ Árvore AVL

└ Rotações em Árvores AVL

Rotação Simples à Esquerda: 1. Estado Inicial

Situação

A árvore está balanceada. O fator de balanceamento (*dif*) de cada nó está dentro do limite de $[-1, 1]$. Uma nova inserção irá quebrar essa propriedade.



Árvore Balanceada



└ Árvore AVL

└ Rotações em Árvores AVL

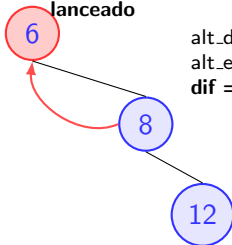
Rotação Simples à Esquerda: 2. Desbalanceamento

Problema

A inserção do nó **12** na subárvore direita do nó **8** faz com que o fator de balanceamento da raiz (**6**) se torne **2**, o que caracteriza um desbalanceamento.

$alt_d = 2$
 $alt_e = 0$
 $dif = 2 - 0 = 2$

Nó desbalanceado



$alt_d = 1$
 $alt_e = 0$
 $dif = 1 - 0 = 1$

$alt_d = 0$
 $alt_e = 0$
 $dif = 0 - 0 = 0$



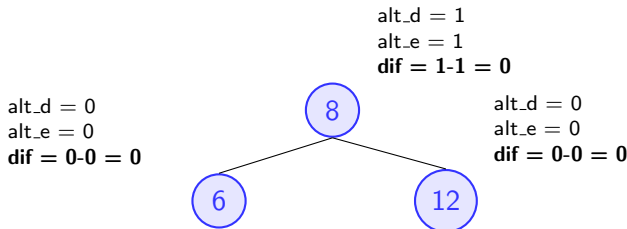
└ Árvore AVL

└ Rotações em Árvores AVL

Rotação Simples à Esquerda: 3. Resultado Final

Solução

Uma **rotação simples para a esquerda** é aplicada no nó desbalanceado (6). O seu filho à direita (8) "sobe" para se tornar a nova raiz, e o nó 6 se torna seu filho à esquerda. A árvore volta a estar balanceada.

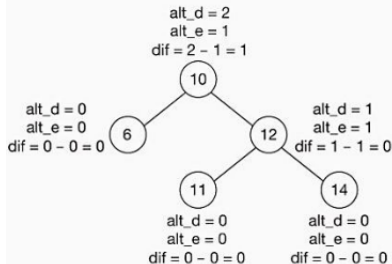


└ Árvore AVL

└ Rotações em Árvores AVL

Exercício

Rotação simples para a esquerda



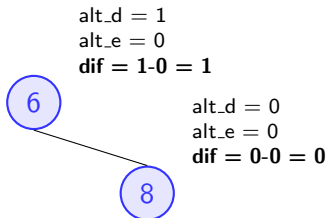
Árvore balanceada

Remoção
do nó nº 6

Rotação Dupla (Direita-Esquerda): 1. Estado Inicial

Situação

A árvore se encontra balanceada. Uma nova inserção será feita, criando um desbalanceamento do tipo "joelho", que exige uma rotação dupla para ser corrigido.



Árvore Balanceada



└ Árvore AVL

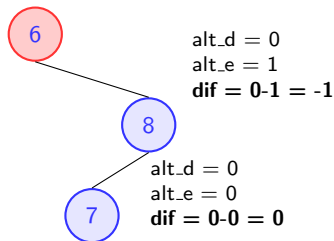
└ Rotações em Árvores AVL

Rotação Dupla (Direita-Esquerda): 2. Desbalanceamento

Problema

A inserção do nó **7** na subárvore **esquerda** do nó **8** cria um "joelho". O fator de balanceamento da raiz (**6**) se torna **+2** e o de seu filho (**8**) se torna **-1**. Esta combinação de sinais opostos **(+,-)** indica a necessidade de uma rotação dupla.

$alt_d = 2$
 $alt_e = 0$
 $dif = 2 - 0 = 2$



Árvore Desbalanceada



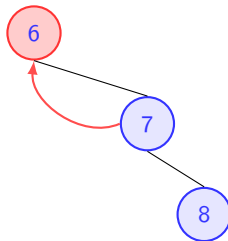
└ Árvore AVL

└ Rotações em Árvores AVL

Rotação Dupla: 3. Primeira Etapa (Rotação à Direita)

Solução - Passo 1

Primeiro, aplicamos uma **rotação simples à direita** no nó filho (8) para "desfazer o joelho" e alinhar os nós. A árvore continua desbalanceada na raiz, mas agora o problema se tornou um caso de rotação simples.



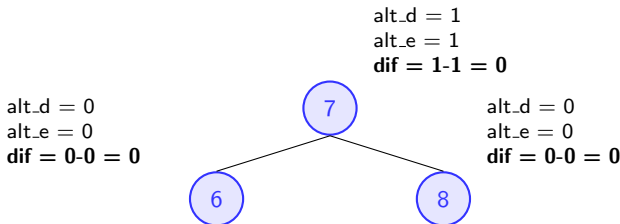
Árvore Alinhada (ainda desbalanceada)



Rotação Dupla: 4. Segunda Etapa e Resultado Final

Solução - Passo 2

Agora, com os nós alinhados, aplicamos uma **rotação simples à esquerda** no nó que estava originalmente desbalanceado (6). O nó 7 sobe, tornando-se a nova raiz, e a árvore volta a ficar perfeitamente balanceada.



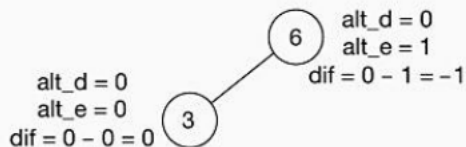
Árvore Balanceada



└ Árvore AVL

└ Rotações em Árvores AVL

Exercício






Árvore balanceada

Inserção
do nó n° 5

- ▶ Vantagens e Desvantagens de uma árvore Binária
 - ▶ A desvantagem é que ela pode se tornar muito desbalanceada, caso em que a pesquisa degenera um tempo de $O(n)$;
 - ▶ A vantagem é a eficiência que uma árvore binária oferece para inserções e exclusões;
- ▶ Vantagens e Desvantagens de uma árvore AVL
 - ▶ A vantagem de uma árvore AVL é que ela é sempre balanceada, garantindo a velocidade $O(\log n)$ do algoritmo de busca binária.
 - ▶ As desvantagens são as rotações complexas usadas pelos algoritmos de inserção e remoção necessários para manter o balanceamento da árvore.





Referências I

-  Lee K.D., Hubbard S. (2015) Trees. In: Data Structures and Algorithms with Python. Undergraduate Topics in Computer Science. Springer, Cham. Retrieved from https://doi.org/10.1007/978-3-319-13072-9_6
-  Hubbard, J. (2007). Schaum's Outlines of Data Structures with Java. Retrieved from <http://www.amazon.com/Schaums-Outline-Data-Structures-Java/dp/0071476989>
-  Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms, 3rd Edition. MIT Press.



Referências II

-  Ascencio, Ana Fernanda Gomes. (2010). Estrutura de dados: Algoritmos, análise da complexidade e implementações em Java e C++. São Paulo: Pearson Prentice Hall.
-  Szwarcfiter, Jayme Luiz & Markenzon, Lilian. (2015). Estruturas de dados e seus algoritmos. 3.ed. Rio de Janeiro: LTC.





Estruturas de Dados: Árvores

Professor: Elton Sarmanho¹

E-mail: eltonss@ufpa.br



¹Faculdade de Sistemas de Informação - UFPA/CUTINS

19 de outubro de 2025