UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
INF01203 - ESTRUTURAS DE DADOS

João Vítor Fonseca Soares

Miguel Predebon Abichequer

# Árvores ABP e AVL

Relatório de execução

Porto Alegre
2. Semestre
2025

João Vítor Fonseca Soares
Miguel Predebon Abichequer

# Árvores ABP e AVL

Relatório de execução

Trabalho referente ao projeto final da disciplina INF01203 - Estruturas De Dados.

Porto Alegre
2. Semestre
2025

**SUMÁRIO**

# 1 INTRODUÇÃO

O presente trabalho visa analisar a eficiência de dois tipos distintos de árvores para tratar dados oriundos de arquivos estruturados. Para atingir tal fim, foram implementados os Tipos Abstratos de Dados (TAD) referentes às árvores AVL e ABP. Os critérios de avaliação envolvem a contagem de operações realizadas no decorrer da inserção dos elementos nos nodos e o tempo de execução necessário para isso.

# 2 FUNDAMENTAÇÃO TEÓRICA

As estruturas de árvore escolhidas, ABP e AVL, organizam os elementos em seus nodos da mesma forma: mantendo aqueles de maior valor sempre à direita. Em ambos os TADs, dados adicionados são comparados e então inseridos no nodo alocado na posição correta: aquela em que todos os valores à esquerda do elemento adicionado são menores do que ele e os à direita são maiores. Entretanto, essas estruturas de dados distinguem-se quanto à capacidade de rotação, que está presente na árvore AVL e ausente na árvore ABP. Devido a essa diferença, somente a AVL possui balanceamento dinâmico, ou seja, é capaz de se autobalancear sem a necessidade de intervenção por parte do programador, mantendo sempre o formato de menor altura, o que garante a eficiência logarítmica.

Para isso, a estrutura da árvore AVL contém um campo destinado ao fator de balanceamento (FB), um número inteiro dado pela diferença (subtração) entre as alturas das subárvores esquerda e direita de determinado nodo. Cada nodo possui somente um fator de balanceamento e ele é considerado ideal se o seu módulo é igual a um ou zero. Caso essa condição não seja satisfeita, o nodo é considerado desbalanceado e há, portanto, necessidade de rotação.

## 2.1 REVISÃO TEÓRICA DA ABP

Árvores binárias são uma estrutura de dados hierárquica cujos nodos podem ter apenas dois filhos: o esquerdo e o direito. As ABPs, por sua vez, são árvores binárias que armazenam informações de forma que seu ramo esquerdo sempre possua elementos menores que o direito. Para isso, é necessário que exista um

método de inserção capaz de compará-los com o elemento a ser adicionado, dessa forma o nodo é alocado na posição correta, respeitando a restrição imposta. A função é construída de maneira recursiva, prevendo a exceção contida no caso em que a árvore está inicialmente vazia, e tendo como critério de parada a devida inserção do nodo. Algoritmo para a função “inserir em uma árvore de pesquisa binária”, que recebe um valor a ser inserido e uma árvore e retorna a mesma árvore com o nodo adicional:

```
SE Raiz é NULO:
    RETORNE Novo No(X)
SE X < Raiz.Valor:
    Raiz.Esquerda = Inserir(Raiz.Esquerda, X)
SENÃO SE X > Raiz.Valor:
    Raiz.Direita = Inserir(Raiz.Direita, X)
RETORNE Raiz
```

No que tange à eficiência, a árvore binária de pesquisa dispõe de funções de busca muito mais velozes que as presentes em uma AB simples. Isso se deve à organização dos nodos da ABP, que facilita a procura por elementos específicos ao considerar sua lógica de ordenação.

## 2.2 REVISÃO TEÓRICA AVL

As árvores AVLs são também árvores binárias de pesquisa, mas que possuem quatro funções de rotação (à direita, à esquerda, dupla à direita e dupla à esquerda) com as quais organizam de maneira inteligente os nodos inseridos, fazendo com que sua altura seja sempre mínima. Para que isso seja garantido, as funções responsáveis por rotar subárvores são constantemente chamadas para restaurar a condição de altura mínima perdida devido à alocação de um novo nodo.

Algoritmo para rotação à direita (em caso de desequilíbrio no ramo esquerdo):

*Nó RotaçãoDireita(Nó A):*

*Nó B = A.Esquerda*

*Nó C = B.Direita*

*B.Direita = A*

*A.Esquerda = C*

*RETORNE B*

Algoritmo para a rotação à esquerda:

*Nó RotaçãoEsquerda(Nó A):*

*Nó B = A.Direita*

*A.Direita = B.Esquerda*

*B.Esquerda = A*

*RETORNE B*

A complexidade das operações em uma árvore AVL é, no pior caso, significativamente menor do que a de árvores binárias de pesquisa não balanceadas. Já no caso médio, ambas as árvores apresentam eficiência similar. Apesar das operações de rotação, que acompanham toda a inserção de elementos nas árvores AVL, adicionarem chamadas extras às funções da árvore, o resultado balanceado é, no pior caso, muito mais eficiente que uma simples ABP. Isso é ilustrado na tabela abaixo, ABP x AVL:

**Tabela 1** – Complexidade de Tempo em Notação big O: ABP x AVL

| Estrutura | Caso médio | Pior caso |
|---|---|---|
| AVL | O(log n) | O(log n) |
| ABP | O(log n) | O(n) |

Fonte: Wikipédia (2025) adaptada.

## 3 BREVE DESCRIÇÃO DO CÓDIGO

O código recebe como argumentos os arquivos *dataset.csv* e *lista_jogador1.txt*, além do *path* de saída, sendo o primeiro a fonte de informações sobre os jogos e o segundo os jogos escolhidos pelo jogador. Primeiramente, na própria função principal, o arquivo .csv é lido e suas linhas são transformadas em dados, por meio de funções como de modificação de tipo e separação tendo em vista caracteres específicos. Após isso, os dados coletados são inseridos nas estruturas de árvore, então é medido o tempo necessário (quantidade contínua) para isso acontecer em cada caso. São também armazenados em contadores o número de rotações e comparações (quantidades discretas), além da altura da árvore e o número de nodos criados, para uma avaliação performática mais direta.

### 3.1 REFERÊNCIAS DOS CÓDIGOS

Visto que o grupo poderia utilizar códigos previamente disponibilizados em aula, as seguintes funções foram extraídas de PDFs disponibilizados no Moodle. Funções que não foram destacadas aqui e nem possuem referência associada nas respectivas .c são autorais.

- *InsereArvoreABP*: função extraída da aula 16 ABP (página 15) e adaptada
- *ConsultaABP*: função extraída do enunciado do trabalho e adaptada
- *AlturaABP*: função extraída do laboratório de balanceamento de árvores binárias
- *RotacaoDir*: função extraída da aula 18 AVL (página 31)
- *RotacaoEsq*: função extraída da aula 18 AVL (página 39)
- *RotacaoDuplaDir*: função extraída da aula 18 AVL (página 46)
- *RotacaoDuplaEsq*: função extraída da aula 18 AVL (página 46)

- *Caso1*: função extraída da aula 18 AVL (página 71)
- *Caso2*: função extraída da aula 18 AVL (página 72)
- *InsereArvoreAVL*: função extraída da aula 18 AVL (página 70) e adaptada
- *AlturaAVL*: função extraída do laboratório de balanceamento de árvores binárias
- *ConsultaAVL*: função extraída do enunciado do trabalho e adaptada
- *main*: código extraído parcialmente do exemplo de passagem de parâmetros disponibilizado no Moodle

### 3.2 RECURSO DE AGILIDADE

O arquivo .bat facilita a execução do código, uma vez que contém uma sequência de comandos para o sistema operacional. Em vez de escrever, um por um, todos os comandos necessários para executar a aplicação no prompt do Windows, o arquivo .bat agiliza esse processo, pois executa-os de forma automática. O presente trabalho utilizou disso para agilizar os testes.

## 4 RESULTADOS

De modo a comparar a performance das duas árvores, foram pensados quatro casos distintos que abordam situações extremas de dados: dados em ordem crescente, dados em ordem decrescente, busca por dados profundos e busca de elementos inexistentes. Dessa forma, será possível verificar um comparativo de performance sólido e pautado em uma análise quantitativa.

Todavia, antes de qualquer coisa, é importante verificar o que ocorre caso, sem nenhuma manipulação no .csv, lermos os arquivos de teste disponibilizados pelo professor. A tabela a seguir ilustra o que foi obtido:

**Tabela 2** – Resultado do *dataset.csv* e *lista_jogador1.txt*

| Árvore | Altura | Rotações | Comp.[1] | TMI[2] (ms) | nº Nodos |
|---|---|---|---|---|---|
| ABP | 28 | 0 | 3058 | 6 | 3598 |
| AVL | 14 | 1809 | 2152 | 5,9 | 3598 |

Fonte: Dados da pesquisa.

Os dados de TMI foram gerados a partir de 10 execuções do código, seguindo a tabela a seguir:

**Tabela 3** – TMI

| ABP (ms) | AVL (ms) |
|---|---|
| 6 | 5 |
| 5 | 6 |
| 7 | 5 |
| 6 | 5 |
| 5 | 6 |
| 5 | 7 |
| 5 | 9 |
| 9 | 6 |
| 8 | 4 |
| 4 | 6 |
| **Média: 6** | **5,9** |

Fonte: Dados da pesquisa.

Houve uma diferença quase que insignificante em tempo de inserção, mas cabe destacar que o conjunto de testes (dados de tempo) utilizado é muito pequeno para uma conclusão mais significativa. Todavia, dessa brevíssima testagem, estamos tendendo a afirmar que a AVL acelera o processamento. Os casos mais específicos destacados testarão os resultados no limite e serão uma prova mais cabível da esperada melhor performance da AVL em casos extremos. O que é irrefutável, até então, é que a árvore fica bem mais baixa sendo uma AVL, visto que tivemos alturas de 28 e 14 para ABP e AVL respectivamente.

[1] Comparações
[2] Tempo médio de inserção

## 4.1 DADOS EM ORDEM CRESCENTE

A primeira checagem consiste em reordenar os dados do *dataset.csv* de modo a forçar uma ordem lexicográfica crescente (isto é, está em ordem alfabética, de A até Z, respeitando que títulos iniciando por números se encontram antes do A). A tabela de resultados foi a seguir:

**Tabela 4** – Resultado do *dataset_crescente.csv* e *lista_jogador1.txt*

| Árvore | Altura | Rotações | Comp.[3] | TMI[4] (ms) | nº Nodos |
|---|---|---|---|---|---|
| ABP | 3585 | 0 | 354339 | 123,3 | 3598 |
| AVL | 12 | 3584 | 2155 | 1,3 | 3598 |

Fonte: Dados da pesquisa.

**Tabela 5** – TMI dados crescentes

| ABP (ms) | AVL (ms) |
|---|---|
| 128 | 1 |
| 106 | 1 |
| 137 | 2 |
| 128 | 2 |
| 122 | 1 |
| 119 | 1 |
| 120 | 1 |
| 128 | 2 |
| 120 | 1 |
| 125 | 1 |
| **Média: 123,3** | **1,3** |

Fonte: Dados da pesquisa.

Há uma discordância com a teoria na tabela de resultados, sendo fruto do uso da função *strcasecmp* e a presença de dados numéricos utilizados para a análise lexicográfica. No conjunto de dados, há, por exemplo, 140 e 10000000 como títulos, rearranjados pelo Excel como primeiro e segundo títulos na lista. Isto fará com que 140 seja a raiz da ABP. Na comparação, qualquer valor iniciado em 0 ou 1 e com segundo dígito menor do que 4 ficará à esquerda (que é o caso de 10000000), enquanto todos os demais ficarão à direita. Isso gera a discrepância

[3] Comparações
[4] Tempo médio de inserção

entre o número de nodos e a altura da ABP, que deveriam ter valores iguais, uma vez que o comportamento esperado de valores ordenados em uma ABP é uma lista.

Apesar desta observação, que foi considerada irrelevante frente à pequena diferença que resultou, observa-se que a ABP levou um tempo muito considerável a mais do que a AVL para inserir os dados, além de apresentar alturas com diferenças grotescas, exigindo um número muito maior de comparações para a ABP. Isso é o primeiro indício sólido de que a AVL é muito mais ágil do que a ABP para tratar dados ordenados.

## 4.2 DADOS EM ORDEM DECRESCENTE

O outro lado da moeda é o arranjo dos dados em ordem decrescente, o que deve gerar um resultado semelhante ao obtido no item 4.1, só que agora com lados invertidos na árvore. Os dados obtidos estão na tabela a seguir:

**Tabela 6** – Resultado do *dataset_decrescente.csv* e *lista_jogador1.txt*

| Árvore | Altura | Rotações | Comp. | TMI (ms) |
|---|---|---|---|---|
| ABP | 3594 | 0 | 352145 | 123,3 |
| AVL | 12 | 3584 | 2116 | 1,3 |

Fonte: Dados da pesquisa.

**Tabela 7** – TMI dados decrescentes

| ABP (ms) | AVL (ms) |
|---|---|
| 108 | 2 |
| 110 | 2 |
| 115 | 2 |
| 114 | 2 |
| 123 | 1 |
| 120 | 2 |
| 105 | 2 |
| 129 | 1 |
| 114 | 2 |
| 123 | 2 |
| **Média: 116,1** | **1,8** |

Fonte: Dados da pesquisa.

Como esperado, o resultado foi muito semelhante ao caso anterior, demonstrando a ineficiência da ABP para armazenar dados ordenados. A AVL se saiu muito bem, levando uma média de 1,8 milissegundos para inserir todos os dados no dataset agora ordenado de maneira decrescente. Mas então o que acontece se focarmos na lista de busca ao invés de somente olhar para o dataset?

## 4.3 DADOS PROFUNDOS

Há um jeito de deixar o ambiente de testes ainda mais extremo. Além de deixar o Dataset em ordem crescente, solicitar dados mais profundos na árvore tende a exigir muito mais (pelo menos para as ABPs). Para isso, foi realizado novamente o item 4.1, mas o arquivo lista_jogador1.txt agora recebe os 200 últimos títulos do dataset.csv ordenado (número de títulos selecionados próximo do arquivo de testes original). Dessa forma, estaremos buscando os últimos elementos adicionados, que tendem a ser os mais profundos na árvore. Os resultados foram os seguintes:

**Tabela 8** – Resultado do *dataset_crescente.csv* e *lista_jogador1.txt*

| Árvore | Altura | Rotações | Comp. |
|---|---|---|---|
| ABP | 3585 | 0 | 697100 |
| AVL | 12 | 3584 | 2040 |

Fonte: Dados da pesquisa.

É impossível não reparar que o número de comparações para a ABP praticamente dobrou em comparação com os resultados do teste da seção 4.1, enquanto a AVL obteve um número de comparações próximo do que tinha para uma lista de busca aleatória. Visto que já foi provado o ponto da TMI, ela não foi calculada para este caso.

### 4.4 DADOS INEXISTENTES

Por fim, o último item dos testes consiste em buscar por dados que não estão na árvore. Novamente, iremos utilizar o dataset ordenado, só que agora colocaremos 200 nomes fictícios inexistentes na árvore e realizaremos a busca por eles. Os resultados foram os seguintes:

**Tabela 9** – Resultado do *dataset_crescente.csv* e *lista_ficticia.txt*

| Árvore | Altura | Rotações | Comp. |
|---|---|---|---|
| ABP | 3585 | 0 | 365874 |
| AVL | 12 | 3584 | 2561 |

Fonte: Dados da pesquisa.

Novamente, os resultados demonstram que a AVL se mantém relativamente uniforme no número de comparações, enquanto a ABP chega a valores monstruosos de comparações caso esteja desbalanceada. Dessa forma, há a conclusão óbvia comprovada: a AVL é muito mais eficiente do que a ABP, apesar do custo computacional de realizar rotações no balanceamento.

## 5 CONSIDERAÇÕES FINAIS

O código foi revisado e exportado para o GitHub, disponível publicamente a partir do link: https://github.com/jv-fs/Arvores-ABP-e-AVL. Lá estará todo o projeto construído pelo CodeBlocks, juntamente da documentação explicando o uso. O item README.md está correto e explica um apanhado do que foi desenvolvido neste

relatório. Conforme solicitado, deixamos as referências dos códigos não autorais por todas as partes do projeto, tanto em código quanto no relatório na respectiva seção.

Além disso, optamos por não explicar de maneira tão concentrada no relatório sobre o funcionamento do código, apenas atentamos em destacar as conclusões acerca das diferenças performativas entre as duas estruturas observadas e a revisão teórica que deve justificar um pouco as diferenças. Foi a tentativa do grupo de juntar a teoria ao resultado prático.

## REFERÊNCIAS


DEVMEDIA. **Arquivo .BAT: introdução à programação em lotes**. [S.l.: s.n., 20--?]. Disponível em: https://www.devmedia.com.br/introducao-a-arquivos-bat-e-programacao-em-lotes/24800. Acesso em: 02 dez. 2025.

FEOFILOFF, Paulo. **Algoritmos em linguagem C**. Rio de Janeiro: Elsevier, 2009.

WIKIPÉDIA. **Árvore AVL**. In: WIKIPÉDIA, a enciclopédia livre. Flórida: Wikimedia Foundation, [2025]. Disponível em: https://pt.wikipedia.org/wiki/%C3%81rvore_AVL. Acesso em: 28 nov. 2025.

WIKIPÉDIA. **Árvore binária**. In: WIKIPÉDIA, a enciclopédia livre. Flórida: Wikimedia Foundation, [2025]. Disponível em: https://pt.wikipedia.org/wiki/%C3%81rvore_bin%C3%A1ria. Acesso em: 28 nov. 2025.