PPGCA - Algoritmos e Programação

PROJETO 2

Estrutura de Dados - Árvores

Autor Fabricio Zillig

Este projeto tem como objetivo **comparar o desempenho** de busca e inserção de dois diferentes algoritmos de árvore

- Binary Search Tree (BST)
- AVL Tree

Deveríamos escolher uma base de dados e escrever rotinas para ler o arquivo e mapear os dados nas duas estrutura de dados distintas.

Foram feitos dois testes de inserção: Com os dados aleatórios e ordenados do menor id para o maior id

Metodologia

Metodologia utilizada para realizar os testes e coletar os resultados

Equipamento Utilizado

O programa foi executado em um computador com as seguintes configurações:

- Macbook Pro 2021 com Apple **M1 Pro**
  - CPU de **8 núcleos** (6 de desempenho e 2 de eficiência)
  - GPU de 14 núcleos.
- **16 GB** de memória RAM
- MacOS - Sonoma 14.6.1
- Python 3.12.5

Massa de Dados

Para a comparação dos TAD (Tipo Abstrato de Dado):

- Foi escolhido um dataset contendo informações de filmes do IMDB (site com registro de filmes)
- O dataset contém um total de **1.071.607 registro**
- As informações utilizadas para registro foram:

statistic (str)	id (f64)	titulo (str)	nota_media (f64)	numero_votos (f64)	ano_lancamento (f64)	diretor (str)	genero (str)
count	1.071.607	1071607	1.071.607	1.071.607	921.447	1071607	1071607
null_count	0	0	0	0	150.160	0	0

Algoritmos

Para garantir a precisão do teste algumas medidas foram tomadas:

- Os algoritmos utilizados para a ordenação dos vetores foram extraídos do repositório TheAlgorithms e ajustados com ChatGPT.
- O tempo de execução foi medido utilizando a biblioteca `time` do Python em milissegundos.
- As inserções aleatórias foram executadas **10 vezes** com ordenação aleatória diferentes**.** O tempo de execução considerado foi a média dos 10 testes.
- Foram realizadas **100 buscas** (de chaves existentes aleatórias) **para cada iteração** de inserção
- Para a inserção ordenada houve apenas uma iteração e o teste foi realizado com **10% da base** (**107.159 registros**)

https://github.com/z-fab/ppgca/tree/master/programacao-algoritmos/projeto-arvore

Resultados

Resultados obtidos através dos testes realizados

Resultados – Inserção Aleatória

Descrição dos resultados

- Em média a inserção na BST levou **4,7 segundos** e na AVL levou **10,6 segundos**
- A busca, em média, levou **0,008 ms** na BST e **0,02 ms** na AVL
- Em média as BST ficaram com uma altura de **48** (variando de 46 até 50) enquanto a AVL de **23** (mesma altura em todas iterações)

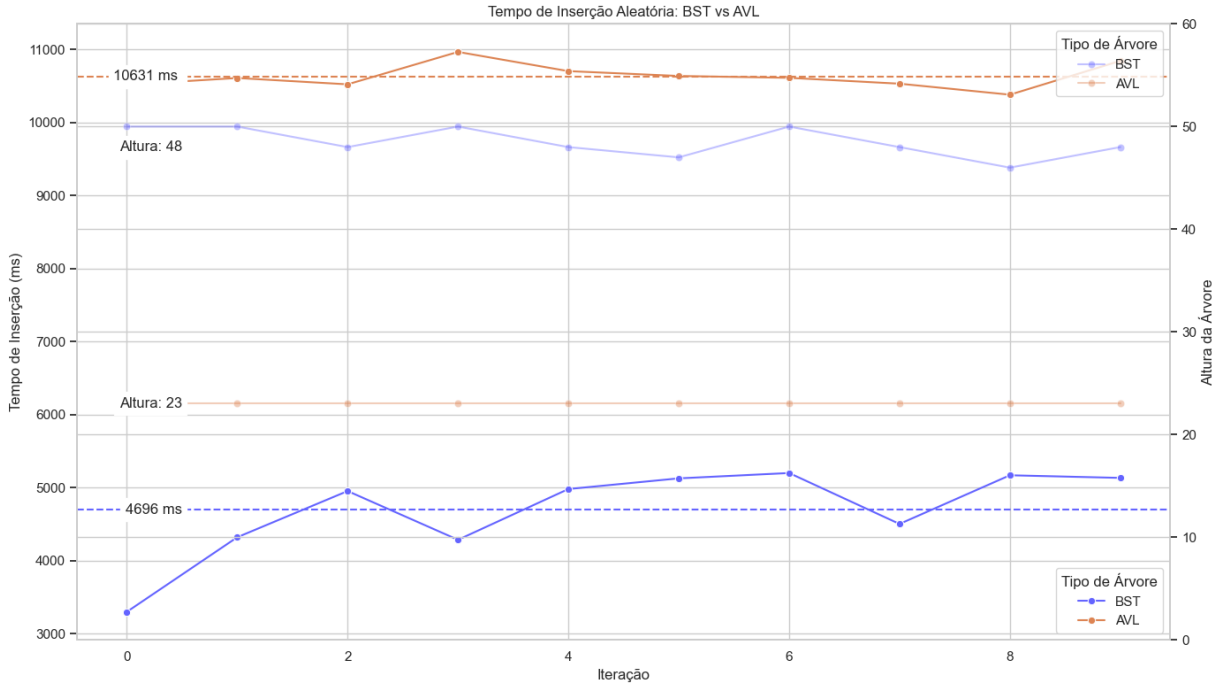
statistic (str)	n_iter (f64)	tempo_insercao_bst (f64)	tempo_insercao_avl (f64)	altura_bst (f64)	altura_avl (f64)
count	10	10	10	10	10
null_count	0	0	0	0	0
mean	4	4.696	10.631	48	23
std	3	605	170	1	0
min	0	3.298	10.380	46	23
25%	2	4.319	10.521	48	23
50%	5	4.980	10.610	48	23
75%	7	5.132	10.702	50	23
max	9	5.199	10.965	50	23

statistic (str)	n_iter (f64)	n_search (f64)	tempo_busca_avl (f64)	tempo_busca_bst (f64)	altura_bst (f64)	altura_avl (f64)
count	1.000,0000	1.000,0000	1.000,0000	1.000,0000	1.000,0000	1.000,0000
null_count	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000
mean	4,5000	100,0000	0,0177	0,0080	48,5000	23,0000
std	2,8737	0,0000	0,0912	0,0041	1,3608	0,0000
min	0,0000	100,0000	0,0069	0,0029	46,0000	23,0000
25%	2,0000	100,0000	0,0110	0,0060	48,0000	23,0000
50%	5,0000	100,0000	0,0119	0,0072	48,0000	23,0000
75%	7,0000	100,0000	0,0131	0,0088	50,0000	23,0000
max	9,0000	100,0000	2,8632	0,0532	50,0000	23,0000

Resultados – Inserção Aleatória

Desempenho Inserção

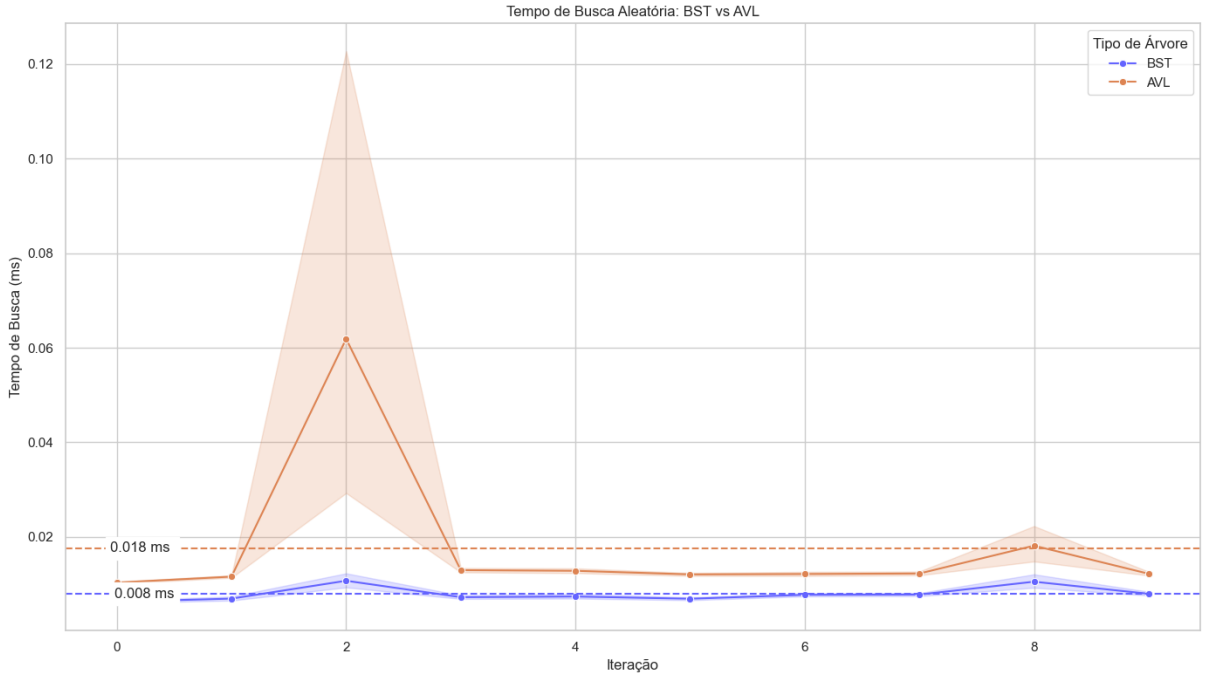
- Podemos perceber que o tempo da AVL variou menos que a BST. Isso comprova que a ordem de inserção interfere muito mais na BST que na AVL
- A Altura final da BST também depende diretamente da ordem de inserção, enquanto da AVL não



Resultados – Inserção Aleatória

Desempenho Busca

- Os tempos de buscas, apesar da diferença de altura das árvores, se manteve muito semelhante nos testes
- Houve uma variação maior no tempo das buscas na segunda iteração. Um caso atípico



Resultados – Inserção Ordenada

Descrição dos resultados

- A inserção na BST levou **402.466 segundos** (> 6 min). Na AVL levou **0,6 segundos**
- A busca, em média, levou **3,9 ms** na BST e **0,01 ms** na AVL
- A BST ficou com uma altura de **107.159** (altura do total de registros) enquanto a AVL de **16**

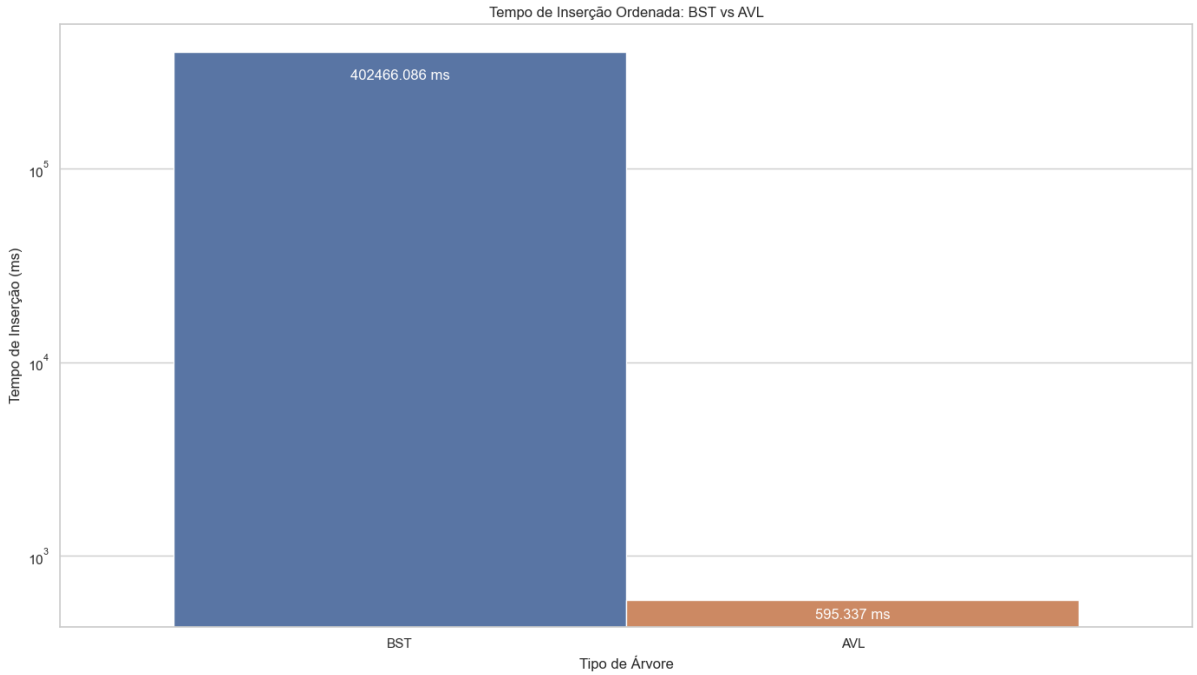
statistic (str)	n_iter (f64)	tempo_insercao_bst (f64)	tempo_insercao_avl (f64)	altura_bst (f64)	altura_avl (f64)
count	1	1	1	1	1
null_count	0	0	0	0	0
mean	0	402.466	595	107.159	16
std	null	null	null	null	null
min	0	402.466	595	107.159	16
25%	0	402.466	595	107.159	16
50%	0	402.466	595	107.159	16
75%	0	402.466	595	107.159	16
max	0	402.466	595	107.159	16

statistic (str)	n_iter (f64)	n_search (f64)	tempo_busca_avl (f64)	tempo_busca_bst (f64)	altura_bst (f64)	altura_avl (f64)
count	100,0000	100,0000	100,0000	100,0000	100,0000	100,0000
null_count	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000
mean	0,0000	100,0000	0,0131	3,8790	107.159,0000	16,0000
std	0,0000	0,0000	0,0046	2,3446	0,0000	0,0000
min	0,0000	100,0000	0,0069	0,0439	107.159,0000	16,0000
25%	0,0000	100,0000	0,0091	2,0831	107.159,0000	16,0000
50%	0,0000	100,0000	0,0131	3,7270	107.159,0000	16,0000
75%	0,0000	100,0000	0,0162	5,3811	107.159,0000	16,0000
max	0,0000	100,0000	0,0269	9,3820	107.159,0000	16,0000

Resultados – Inserção Ordenada

Desempenho Inserção

- Com os dados ordenados, o desempenho da BST é muito pior que AVL.
- A inserção na BST levou **402.466 segundos** (> 6 min). Na AVL levou **0,6 segundos**

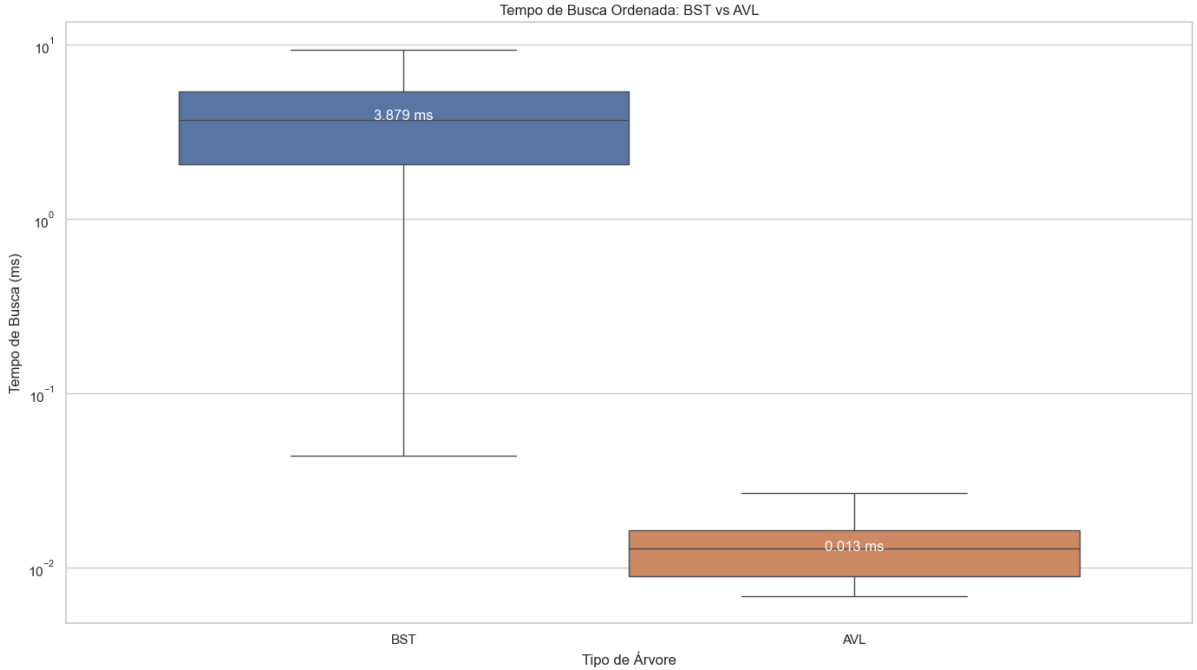


Resultados – Inserção Ordenada

Desempenho Busca

- A busca, em média, levou **3,9 ms** na BST e **0,01 ms** na AVL

- A variação do tempo também foi muito maior na BST já que ficamos na prática com uma lista ligada. Provavelmente houveram valores que estavam no começo da árvore e outros que estavam no final e necessitaram de muito mais passos até encontra-los



Resultados

Questões sobre os dados utilizados

Algumas perguntas que poderiam ser feitas sobre os dados:

1. Quantos filmes foram lançados em cada ano?
2. Qual foi o Ano com maior nota média?
3. Filmes mais antigos e filmes mais recentes na base de dados?

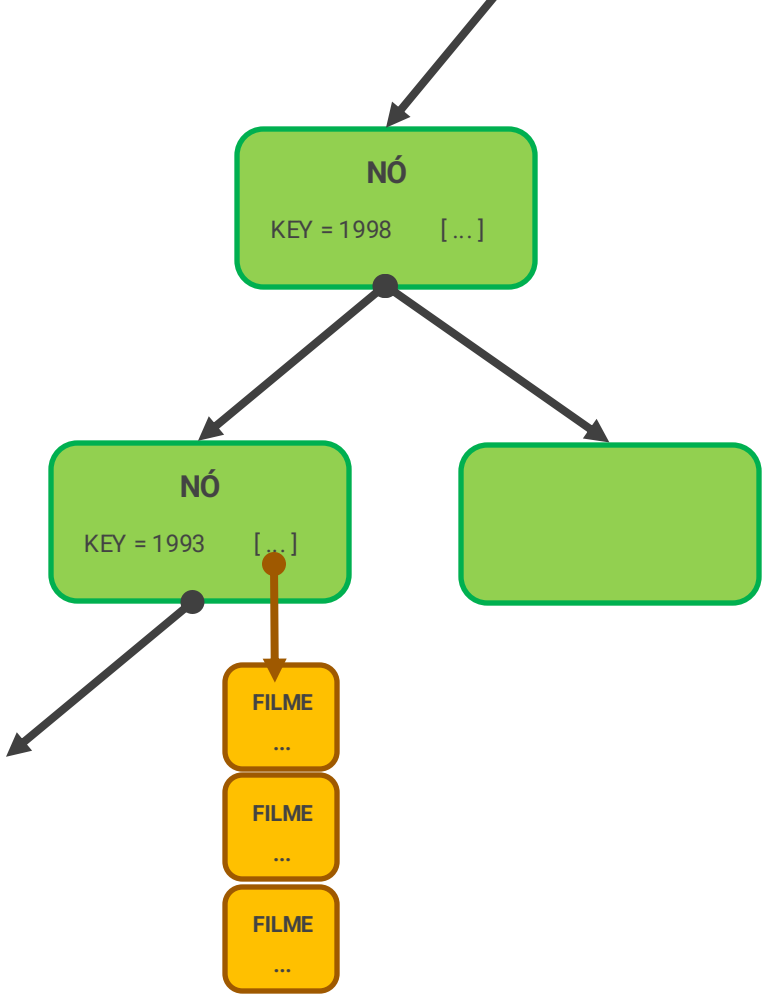
Resultados

Questões sobre os dados utilizados

Com a estrutura atual teríamos que percorrer toda a árvore para responder as perguntas.

Uma solução mais eficiente seria utilizar o Ano como **chave** e em cada nó do ano guardar uma **lista** dos filmes pertencentes aquele ano.

Dessa forma conseguiríamos responder todas as perguntas de forma mais eficiente



Ajustes

Ajustes realizados para investigar o resultado da busca AVL

Resultados – Algoritmo Ajustado

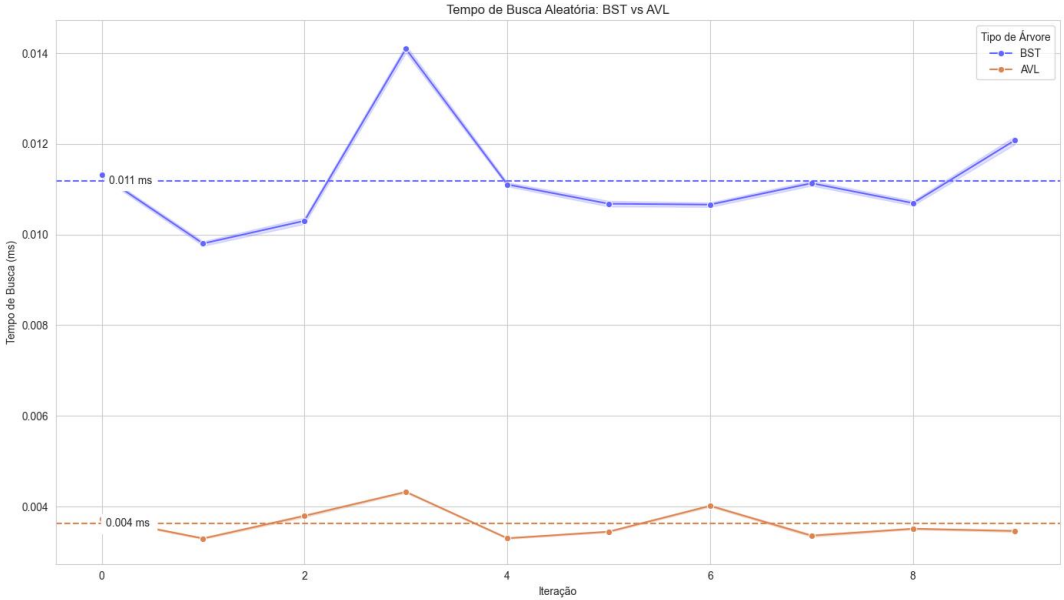
Alteração Algoritmo

- Foi percebido que, diferente do esperado, o tempo da AVL, na busca, apresentou um tempo maior que a árvore binária.
- Após essa análise o algoritmo usado para medição do tempo foi alterado: ao invés de construir as árvores e realizar as buscas, as buscas foram feitas logo em seguida da criação da respectiva árvore

Resultados – Algoritmo Ajustado

Resultado

- Com a alteração foi possível obter o comportamento esperado: o tempo de busca da AVL se mostrou mais eficiente que o tempo de busca na BST.
- O tempo médio de busca na BST foi de **0,01 ms** e na AVL foi de **0,004 ms**.
- Com isso, comprovamos a eficiência da AVL em buscas



Conclusão

Conclusões obtidas através do trabalho

- A AVL apresentou um **desempenho de inserção mais estável** em comparação com a BST, com menor variação de tempo entre as iterações.

- Na **busca**, o tempo médio foi **bastante similar** entre as duas árvores, porém, ao ajustar a ordem com que o teste foi executada chegamos ao resultado teórico esperado: AVL é mais rápida em busca ao comparar com a BST

- **A ordem de inserção** interfere muito mais no desempenho da BST do que da AVL, indicando a maior suscetibilidade da BST ao desbalanceamento.

- A escolha entre BST e AVL **deve levar em consideração o perfil dos dados e a frequência das operações** de **inserção** e **busca**, sendo a **AVL** mais indicada quando **se prioriza o balanceamento** e a **previsibilidade**, e a **BST** quando se **busca simplicidade e menor tempo de inserção** em dados que já estão distribuídos de forma favorável.

{ VALEU }

Fabricio Zillig

https://github.com/z-fab