


UNIVERSIDADE FEDERAL DE JUIZ DE FORA

ICE - INSTITUTO DE CIÊNCIAS EXATAS

DEPT. CIÊNCIA DE COMPUTAÇÃO

BACHARELADO EM SISTEMAS DE INFORMAÇÃO

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

DCC012 · ESTRUTURA DE DADOS ||

2° SEMESTRE DE 2020

TRABALHO
PARTE 2

Juiz de Fora

2021

UNIVERSIDADE FEDERAL DE JUIZ DE FORA

BACHARELADO EM SISTEMAS DE INFORMAÇÃO

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

DCC012 · ESTRUTURA DE DADOS ||

ORIENTADOR:

MARCELO CANIATO RENHE

COMPONENTES:

NOME	MATRÍCULA
BEATRIZ CUNHA RODRIGUES	201776038
IVANYLSON HONÓRIO GONÇALVES	201776002
JOÃO PEDRO SEQUETO NASCIMENTO	201776022

Juiz de Fora

2021

Resumo

r

Este projeto é a segunda parte da disciplina de Estrutura de Dados 2.

Sumário

● Introdução

Nesta fase do trabalho, foi necessário analisar um conjuntos de dados fornecido pelos arquivos “brazil_cities_coordinates.csv” e “brazil_covid19_cities_processado.csv” [9] , e comparar o desempenho de diferentes estruturas de dados baseados no conhecimento adquirido ao longo da disciplina. Com isso, fomos capazes de:

- Manipular o arquivo em modo texto e armazená-lo em uma quadtree;
- Implementar cada estrutura de dados;
- Analisar o desempenho das estruturas em diferentes situações;
- Apresentar os resultados obtidos.

● Detalhamento das atividades

Integrantes	Funções
BEATRIZ CUNHA RODRIGUES	Esteve responsável pela implementação da tabela hash, do módulo de testes e contribuiu para a implementação da análise das estruturas.
IVANYLSON HONÓRIO GONÇALVES	Esteve responsável pela implementação da árvore B, contribuiu para a implementação da árvore AVL e para análise das estruturas.
JOÃO PEDRO SEQUETO NASCIMENTO	Esteve responsável pela implementação do quad tree (pré processamento dos dados), pela implementação da árvore AVL e contribuiu para a análise das estruturas.

• Explicações das estruturas de dados

- Árvore AVL: Criada em 1962 pelos soviéticos Georgii Adelson-Velskii e Yevgeniy Landis, a árvore AVL é uma árvore binária de busca com balanceamento, ou seja, que possui mecanismos para balancear a árvore após inserções. Nesta estrutura, as alturas das subárvores não diferem em mais do que uma unidade, quando isso ocorre, acontecem rotações para que as alturas voltem a ter no máximo uma unidade de diferença. Dessa forma, a árvore é altamente balanceada, mantendo a altura da árvore em O (logn) e otimizando o processo de busca na árvore.[1,2,3,4 e 9].

- Árvore B: Proposta por Rudolf Bayer e Edward Meyers McCreight em 1972, a árvore B é uma estrutura de dados em árvore perfeitamente balanceada projetada de forma a permitir acesso eficiente em memória secundária, de forma a ser bastante utilizado em sistemas de banco de dados e arquivos. A árvore B é uma generalização da árvore binária de busca, porém armazenando mais do que uma chave em cada nó e podendo possuir mais de dois filhos em cada nó com exceção da raiz, sendo uma árvore Multiway. Dessa forma, a árvore B permite realizar um número menor de acessos a disco e permitindo um melhor desempenho nas operações de busca. [1,2,3,4 e 6].

- Árvore QuadTree: Criado por Raphael Finkel e Jon Louis Bentley, a árvore QuadTree é uma árvore implementada como uma generalização multidimensional da árvore binária de busca. No caso, a árvore QuadTree possui 4 filhos ao invés de 2 como na árvore binária de busca tradicional, sendo que os 4 filhos indicam 4 quadrantes (NW, NE, SW,SE) e a busca ocorre através de coordenadas. Árvore muito utilizada para representação de dados espaciais. [1,2,3,4 e 7].

- Tabela Hash: Estrutura de dados para realizar acesso direto com endereçamento na tabela ao armazenar e buscar informações. Utiliza de uma função para realizar transformações na chave ao indicar a posição de acesso na tabela e tratamento de colisões ao indicar posições que já estão ocupadas. [1,2,3,4 e 5].

● Decisões de Implementação

- Árvore AVL: Com relação a implementação da árvore AVL, foi realizada a implementação generalizada da árvore binária de busca, com uma estrutura para a árvore e uma estrutura para nós. Inserimos as rotações para realizar o balanceamento após as inserções. A implementação não utiliza da indicação do nó pai ao realizar a rotação, apenas utilizando recursão e retornando a subárvore gerada pela rotação para o nó “pai”. Nos nós, apenas fica guardado uma informação do tipo inteiro que na etapa de análise, é a chave gerada pela função da tabela de espelhamento, sendo que para acessar as outras informações do registro é necessário acessar na tabela Hash.

- Árvore B: Com relação a implementação da árvore B Tree, foi realizada a implementação foram feitos da seguinte forma: Todas as folhas estão no mesmo nível.Uma árvore B é definida pelo termo grau mínimo 't'. O valor de t depende do tamanho do bloco de disco. Cada nó, exceto raiz, deve conter pelo menos (teto) ([t-1] / 2) chaves. A raiz pode conter no mínimo 1 chave. Todos os nós (incluindo a raiz) podem conter no máximo t - 1 chaves. O número de filhos de um nó é igual ao número de chaves nele mais 1. Todas as chaves de um nó são classificadas em ordem crescente. O filho entre duas chaves k1 e k2 contém todas as chaves no intervalo de k1 e k2. A árvore B cresce e diminui a partir da raiz, o que é diferente da árvore de pesquisa binária. As árvores de busca binária crescem para baixo e também diminuem para baixo. Como outras árvores de pesquisa binárias equilibradas, a complexidade de tempo para pesquisar, inserir e excluir é O (log n). Apenas fica guardado uma informação do tipo inteiro que, na etapa de análise, é a chave gerada pela função da tabela de espelhamento, sendo que para acessar as outras informações do registro é necessário acessar na tabela Hash.

- Árvore QuadTree: Na implementação da árvore QuadTree, também foi realizado uma implementação de uma árvore binária de busca, porém com 4 filhos, e também é utilizado dois campos que indicam a coordenada X e Y para indicar em qual nó filho realizar o avanço na busca. Nos nós, são armazenados um TAD que representa a cidade com informações gerais da cidade e de suas coordenadas, de forma que a busca retorna esse registro e não somente uma informação sobre a cidade.

- Tabela Hash: A tabela Hash nesse projeto foi implementado com um vetor de ponteiros que apontam para uma lista dos registros do arquivo “brazil_covid19_cities_processado.csv”. Essa Lista é implementada de forma encadeada, sendo a forma do tratamento de colisão o encadeamento aberto. Para gerar a posição de acesso na tabela, a função Hash faz uma união do código da cidade com a data do registro, transformando as duas informações em inteiros e, posteriormente, somando esses valores. A soma resultante da chave é elevada ao quadrado e passa por um método de divisão de forma a gerar uma posição de acesso válida, dessa forma, utilizando dois métodos: o de divisão e o do meio quadrado.

● Resultados

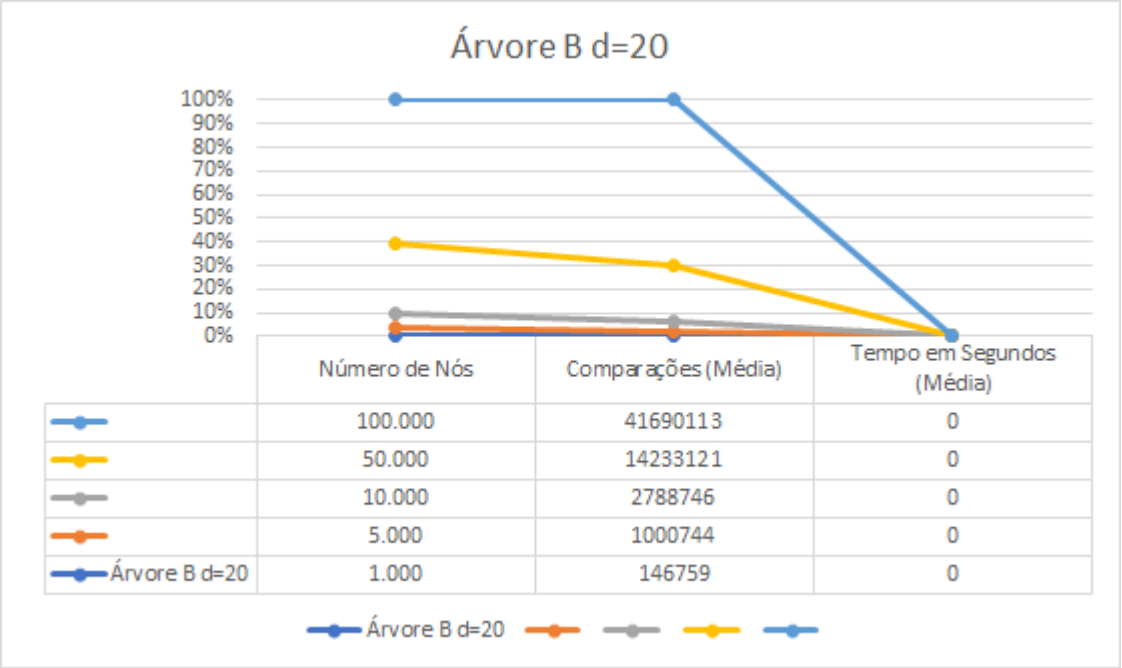
Para este relatório foram realizados testes com os algoritmos selecionados ordenando vetores de diferentes tamanhos e com ordenação aleatória. Foram utilizados vetores de 5 tamanhos diferentes: 10000, 50000, 100000, 500000 e 1000000. Cada vetor foi ordenado 5 vezes e o valor utilizado no relatório foi a média obtida das 5 interações. As métricas definidas para avaliar o resultado foram a quantidade de movimentações entre chaves, quantidade de comparações entre chaves e tempo de execução de cada algoritmo. As tabelas abaixo mostram os valores obtidos com os testes realizados para cada um dos algoritmos.

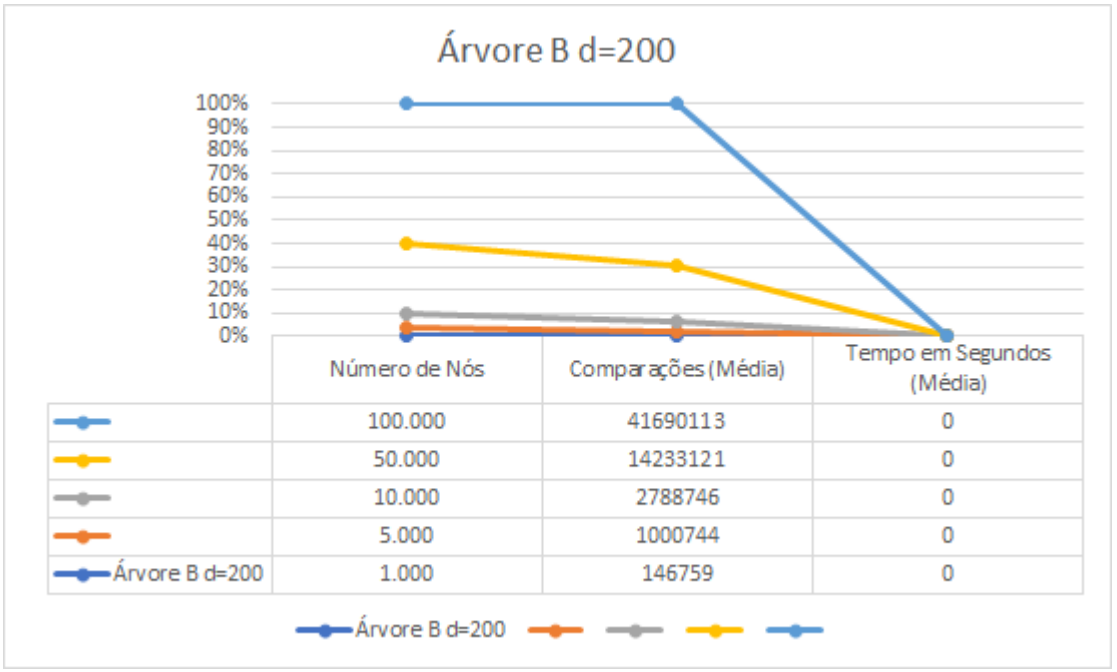
	Número de Nós	Comparações (Média)	Tempo em Segundos (Média)
Árvore B d=20	1.000	146759	0
	5.000	1000744	0.004
	10.000	2788746	0.011
	50.000	14233121	0.06
	100.000	41690113	0.182

	Número de Nós	Comparações (Média)	Tempo em Segundos (Média)
Árvore B d=200	1.000	146759	0.002
	5.000	1000744	0.004
	10.000	2788746	0.01
	50.000	14233121	0.121
	100.000	41690113	0.331

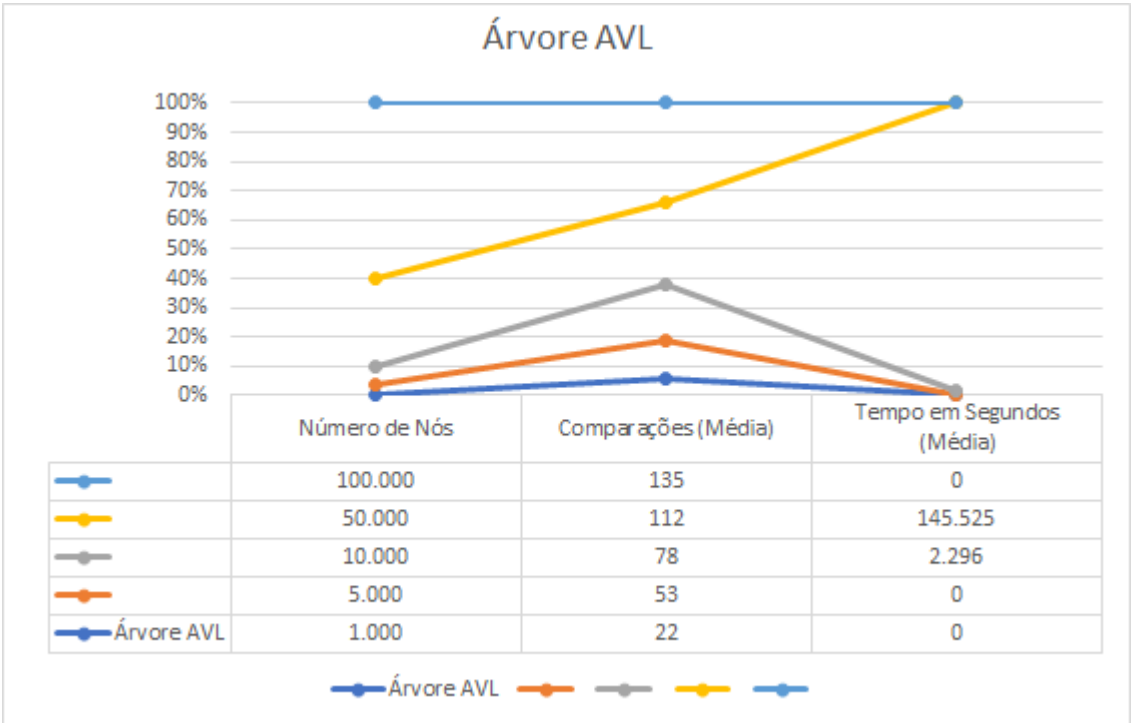
	Número de Nós	Comparações (Média)	Tempo em Segundos (Média)
Árvore AVL	1.000	22	0.15
	5.000	53	0.796
	10.000	78	2.296
	50.000	112	145.525
	100.000	135	762.3

Gráficos

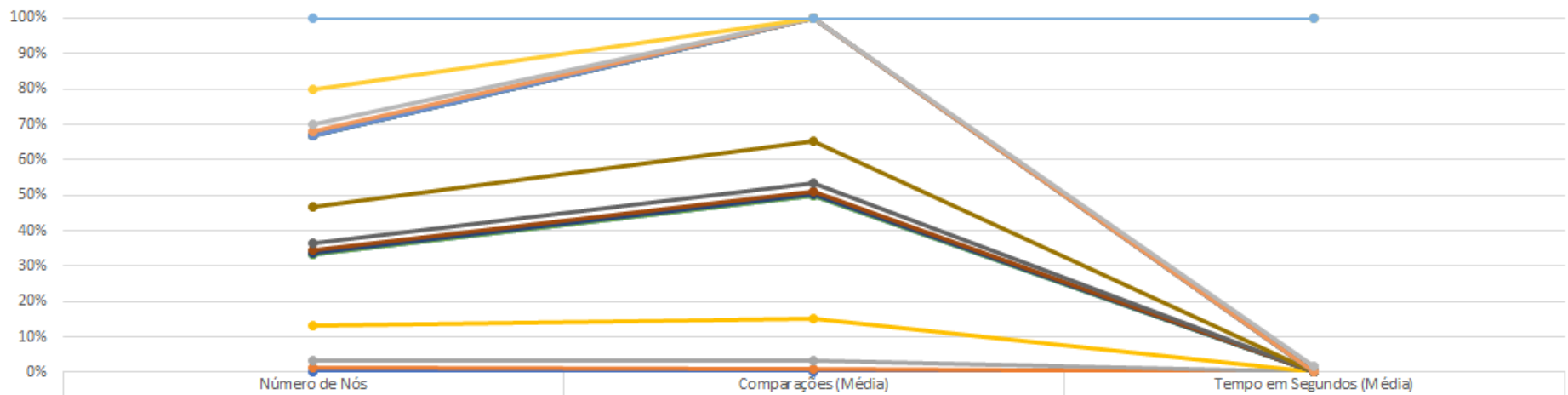




	Número de Nós	Comparações (Média)	Tempo em Segundos (Média)
	100.000	41690113	0
	50.000	14233121	0
	10.000	2788746	0
	5.000	1000744	0
Árvore B d=200	1.000	146759	0



	Número de Nós	Comparações (Média)	Tempo em Segundos (Média)
	100.000	135	0
	50.000	112	145.525
	10.000	78	2.296
	5.000	53	0
Árvore AVL	1.000	22	0



	Número de Nós	Comparações (Média)	Tempo em Segundos (Média)
	100.000	135	0
	50.000	112	145.525
	10.000	78	2.296
	5.000	53	0
Árvore AVL	1.000	22	0
	0	0	0
	100.000	41690113	0
	50.000	14233121	0
	10.000	2788746	0
	5.000	1000744	0
Árvore B d=200	1.000	146759	0
	0	0	0
	100.000	41690113	0
	50.000	14233121	0
	10.000	2788746	0
	5.000	1000744	0
Árvore B d=20	1.000	146759	0

● Análise detalhada dos resultados obtidos

Após a realização e cálculo da média das métricas selecionadas para a análise dos algoritmos, conseguimos observar algumas relações entre as operações de comparação e trocas com o tempo de execução e obter algumas conclusões.

A árvore AVL é muito útil pois executa as operações de inserção, busca e remoção em tempo O(log n) sendo inclusive mais rápida que a árvore rubro-negra para aplicações que fazem uma quantidade excessiva de buscas, porém esta estrutura é um pouco mais lenta para inserção e remoção. Isso se deve ao fato de as árvores AVL serem mais rígidamente balanceadas.

B-Tree tende a ser mais econômica. Outra vantagem é que a rotina de inserção é mais rápida, eventualmente exigindo splits de alguns nós. A rotina de deleção, embora pareça mais complicada.

Com as comparações e o tempo de execução retirados da etapa de análise, foi possível verificar que o tempo de inserção é significativamente maior na árvore AVL do que na árvore B, tanto de ordem 20 quanto 200. É possível que o tempo elevado em comparação com a árvore B seja devido às operações de balanceamento realizadas na inserção dos nós. Com relação ao número de comparações, o número de comparações na árvore B foi maior que na árvore AVL, tanto na inserção quanto na busca. Nos tempos de busca, as três árvores apresentaram um tempo parecido.

● Referência utilizada no desenvolvimento do trabalho

- [1] Souza, J. É. G., Ricarte, J. V. G., Lima, N. C. A., Algoritmos de Ordenação: Um estudo comparativo. In: ENCONTRO DE COMPUTAÇÃO DO OESTE POTIGUAR. 2017, Pau de Ferro, p. 163-173. <Acessado em Fev de 2021>
- [2] Drozdek, A. (2002) Estrutura de dados e algoritmos em C++, Thomson Learning, São Paulo <Acessado em Fev de 2021>
- [3] VIANA, Daniel. Conheça os principais algoritmos de ordenação. Brasil. 26 de Dezembro, 2016. Disponível em: https://www.treinaweb.com.br/blog/conheca-os-principais-algoritmos-de-ordenacao/ (acessado em 22 de Fev de 2021).
- [4] Souza, J. É. G., Ricarte, J. V. G. Lima, N. C. A. Algoritmos de Ordenação: Um estudo comparativo. In: ENCONTRO DE COMPUTAÇÃO DO OESTE POTIGUAR. 2017, Pau de Ferro, p. 169-170.
- [5] https://www.geeksforgeeks.org/hashing-data-structure/<Acessado em Fev de 2021>
- [6] https://www.geeksforgeeks.org/introduction-of-b-tree/<Acessado em Fev de 2021>
- [7] https://www.geeksforgeeks.org/quad-tree/ <Acessado em Fev de 2021>
- [8] https://www.geeksforgeeks.org/avl-tree-set-1-insertion/<Acessado em Fev de 2021>
- [9] FONTES, Rafael. Coronavirus (COVID 19) - Brazil Dataset. 2020. Disponível em: https://www.kaggle.com/unanimad/corona-virus-brazil?select=brazil_covid19_cities.csv (acessado em 08 de dezembro de 2020).