   

##### 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

[Sumário...........................................................................................................................................4](#_Toc62908421)

[ Introdução...........................................................................................................................................5](#_Toc62908422)

[ Detalhamento das atividades......................................................................................................5](#_Toc62908422)

[ Decisões de implementação......................................................................................6](#_Toc62908423)

[ Resultados................................................................................................................. 11](#_Toc62908425)

[ Análise detalhada dos resultados obtidos............................................................. 14](#_Toc62908426)

[ Referência utilizada no desenvolvimento do trabalho.......................................... 15](#_Toc62908427)

# 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”, 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 quad tree;

# 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 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 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 a 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.

# 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 arvore 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.

# Árvore QuadTree: Criado por Raphael Finkel e Jon Louis Bentley, a ávore 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.

# 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.

# Decisões de Implementação

# Árvore AVL: Com relação a implementação da árvore AVL, foi realizado a implementação generalizada da árvore binária de busca, com uma estrutura para a árvore e uma estrutura para os nós. Inserimos as rotações para realizar 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 realizado 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 AVL | 10 000 | 120509 | 0.2 |
| 50 000 | 718487 | 1.6 |
| 100 000 | 1537208 | 3.2 |
| 500 000 | 8803616 | 16.8 |
| 1 000 000 | 18607749 | 34.8 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | Número de Nós | Comparações (Média) | Tempo em Segundos (Média) |
| Árvore AVL | 10 000 | 120509 | 0.2 |
| 50 000 | 718487 | 1.6 |
| 100 000 | 1537208 | 3.2 |
| 500 000 | 8803616 | 16.8 |
| 1 000 000 | 18607749 | 34.8 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | Número de Nós | Comparações (Média) | Tempo em Segundos (Média) |
| Árvore AVL | 10 000 | 120509 | 0.2 |
| 50 000 | 718487 | 1.6 |
| 100 000 | 1537208 | 3.2 |
| 500 000 | 8803616 | 16.8 |
| 1 000 000 | 18607749 | 34.8 |

# 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.

O primeiro ponto que foi observado foi o número de comparações e trocas entre o algoritmo do Mergesort e Quicksort. Foi possível verificar que o número de ambas operações foi similar nos dois algoritmos, porém ocorreu uma diferença no tempo de execução, onde o Mergesort levou um tempo bem acima do que o Quicksort. Os dois algoritmos utilizam a divisão do vetor para a ordenação, de acordo com o paradigma de divisão e conquista, porém o quicksort não utiliza um vetor auxiliar para realizar intercalação dos elementos e também realiza menos operações de trocas entre os elementos, o que pode determinar a redução no tempo de execução.

A próxima análise foi sobre o desempenho do algoritmo Shellsort nesse experimento. Foi possível verificar que o algoritmo demonstrou um tempo de execução melhor que o Mergesort e próximo do Quicksort. Analisando os dados obtidos com a execução do Shellsort, percebemos que o algoritmo realiza um número maior de operações de trocas, porém ocorre uma redução significativa do número de comparações, o que pode ser o motivo da redução de tempo em relação aos demais algoritmos.

Por fim, nesse experimento foi possível verificar uma similaridade de desempenho entre o Quicksort e o Shellsort para o grupo de dados selecionados. Ambos demonstraram um tempo médio de execução bem próximo entre si, e ambos foram mais eficientes do que o Mergesort para a ordenação dos registros.

# 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.