

Rondineli Seba Salomão Matrícula: 20200022063  
Gabriel Belo Pereira dos Reis Matrícula: 2022029760

O relatório abaixo foi elaborado de acordo com a atividade, seguindo a orientação dos padrões *generics* em java e utilizando os fundamentos apresentados pelo CORMEN. Neste relatório, faremos uma breve discussão, apresentando principais detalhes da logica utilizado nos códigos, incluindo comentários das principais funções a análise da complexidade de tempo (Big O).

## PROGRAMA 1 – INDICE REMISSIVO

### 1. DESCRIÇÃO GERAL DO CÓDIGO

O projeto Índice Remissivo em Java implementa uma estrutura de dados que armazena palavras e suas respectivas ocorrências em arquivos de texto. O índice remissivo utiliza três estruturas de dados diferentes: Tabela Hash, Árvore AVL e Árvore Rubro-Negra. Cada uma dessas estruturas possui suas próprias características e vantagens, conforme detalhado abaixo.

### 2. ESTRUTURA DO PROJETO

```
Codigos_Trabalhos CP/  
└─ Projetos_Java/  
    └─ Trabalho_2_ED2/  
        ├── .vscode/  
        │   ├── launch.json  
        │   └── settings.json  
        ├── arquivos/  
        │   ├── texto1.txt  
        │   └── texto2.txt  
        ├── src/  
        │   ├── EstruturaDeDados.java  
        │   ├── IndiceRemissivo.java  
        │   ├── TabelaHash.java  
        │   ├── ArvoreAVL.java  
        │   └── ArvoreRB.java
```

### 3. FUNÇÕES PRINCIPAIS E COMPLEXIDADE DE TEMPO

**Classe Principal: IndiceRemissivo.java**

- **processarArquivos(int x, EstruturaDeDados estrutura)**

- Complexidade de Tempo:  $O(n)$ , onde  $n$  é o número de linhas nos arquivos de texto.

Rondineli Seba Salomão Matrícula: 20200022063  
Gabriel Belo Pereira dos Reis Matrícula: 2022029760

- Lê os arquivos de texto e processa cada linha para identificar as palavras que devem ser armazenadas no índice remissivo.

- **processarLinha(String linha, int numeroLinha, int x, EstruturaDeDados estrutura)**

- Complexidade de Tempo:  $O(m)$ , onde  $m$  é o número de palavras em cada linha.

- Divide a linha em palavras, verifica o comprimento de cada palavra e insere as palavras válidas na estrutura de dados escolhida.

**Estrutura de Dados: TabelaHash.java**

- **inserir(String palavra, int linha)**

- Complexidade de Tempo:  $O(1)$  no melhor caso,  $O(n)$  no pior caso devido a colisões.

- Insere uma palavra e a linha correspondente na tabela hash, utilizando endereçamento aberto para resolver colisões.

- **buscar(String palavra)**

- Complexidade de Tempo:  $O(1)$  no melhor caso,  $O(n)$  no pior caso.

- Busca uma palavra na tabela hash e retorna as linhas onde ela ocorre.

- **remover(String palavra)**

- Complexidade de Tempo:  $O(1)$  no melhor caso,  $O(n)$  no pior caso.

- Remove uma palavra da tabela hash, marcando a entrada como livre.

- **imprimir()**

- Complexidade de Tempo:  $O(n)$ , onde  $n$  é o tamanho da tabela hash.

- Imprime todas as palavras e suas ocorrências armazenadas na tabela hash.

```
1 import java.io.*;
2 import java.util.*;
3
4 public class IndiceRemissivo {
5
6     public static void main(String[] args) throws IOException {
7         Scanner scanner = new Scanner(System.in);
8
9         System.out.println("Digite o valor de X (número mínimo de caracteres para considerar uma palavra:");
10        int x = scanner.nextInt();
11
12        System.out.println("Escolha a estrutura de dados para o índice remissivo:");
13        System.out.println("1 - Hash");
14        System.out.println("2 - Árvore AVL");
15        System.out.println("3 - Árvore Rubro-Negra");
16        int escolhaEstrutura = scanner.nextInt();
17
18        int escolhaColisao = 0;
19        if (escolhaEstrutura == 1) {
20            System.out.println("Escolha a estratégia de tratamento de colisões:");
21            System.out.println("1 - Tentativa Linear");
22            System.out.println("2 - Tentativa Quadrática");
23            escolhaColisao = scanner.nextInt();
24        }
25
26        EstruturaDeDados estrutura;
27        if (escolhaEstrutura == 1) {
28            estrutura = new TabelaHash(escolhaColisao);
29        } else if (escolhaEstrutura == 2) {
```

Rondineli Seba Salomão Matrícula: 20200022063  
Gabriel Belo Pereira dos Reis Matrícula: 2022029760

### Estrutura de Dados: ArvoreAVL.java

#### - inserir(String palavra, int linha)

- Complexidade de Tempo:  $O(\log n)$ , onde  $n$  é o número de nós na árvore.
- Insere uma palavra na árvore AVL, garantindo que a árvore permaneça balanceada após a inserção.

#### - buscar(String palavra)

- Complexidade de Tempo:  $O(\log n)$ .
- Busca uma palavra na árvore AVL e retorna as linhas onde ela ocorre.

#### - remover(String palavra)

- Complexidade de Tempo:  $O(\log n)$ .
- Remove uma palavra da árvore AVL, realizando as rotações necessárias para manter o balanceamento.

#### - imprimir()

- Complexidade de Tempo:  $O(n)$ , onde  $n$  é o número de nós na árvore.
- Imprime todas as palavras e suas ocorrências armazenadas na árvore AVL.

```
1 import java.util.*;
2
3 public class ArvoreAVL implements EstruturaDeDados {
4     private No raiz;
5
6     private class No {
7         String palavra;
8         List<Integer> linhas;
9         int altura;
10        No esquerda;
11        No direita;
12
13        No(String palavra, int linha) {
14            this.palavra = palavra;
15            this.linhas = new ArrayList<>();
16            this.linhas.add(linha);
17            this.altura = 1;
18        }
19    }
20
21    @Override
22    public void inserir(String palavra, int linha) {
23        raiz = inserir(raiz, palavra, linha);
24    }
25
26    private No inserir(No no, String palavra, int linha) {
27        if (no == null) {
28            return new No(palavra, linha);
29        }
30    }
```

Estrutura da classe ArvoreAVL

### Estrutura de Dados: ArvoreRB.java

#### - inserir(String palavra, int linha)

- Complexidade de Tempo:  $O(\log n)$ , onde  $n$  é o número de nós na árvore.
- Insere uma palavra na árvore Rubro-Negra, garantindo que a árvore permaneça balanceada após a inserção.

#### - buscar(String palavra)

Rondineli Seba Salomão Matrícula: 20200022063  
Gabriel Belo Pereira dos Reis Matrícula: 2022029760

- Complexidade de Tempo:  $O(\log n)$ .
- Busca uma palavra na árvore Rubro-Negra e retorna as linhas onde ela ocorre.
- **remover(String palavra)**
  - Complexidade de Tempo:  $O(\log n)$ .
  - Remove uma palavra da árvore Rubro-Negra, realizando as rotações e ajustes de cores necessários para manter o balanceamento.
- **imprimir()**
  - Complexidade de Tempo:  $O(n)$ , onde  $n$  é o número de nós na árvore.
  - Imprime todas as palavras e suas ocorrências armazenadas na árvore Rubro-Negra.

```
1 import java.util.*;
2
3 public class ArvoreRB implements EstruturaDeDados {
4     private final boolean RED = true;
5     private final boolean BLACK = false;
6
7     private class No {
8         String palavra;
9         List<Integer> linhas;
10        No esquerda, direita, pai;
11        boolean cor;
12
13        No(String palavra, int linha, boolean cor) {
14            this.palavra = palavra;
15            this.linhas = new ArrayList<>();
16            this.linhas.add(linha);
17            this.cor = cor;
18        }
19    }
20
21    private No raiz;
22
23    @Override
24    public void inserir(String palavra, int linha) {
25        raiz = inserir(raiz, palavra, linha);
26        raiz.cor = BLACK;
27    }
28
29    private No inserir(No h, String palavra, int linha) {
```

Estrutura da classe ArvoreRB

## CONCLUSÃO

O projeto de Índice Remissivo implementado em Java apresenta uma solução eficiente para armazenar e buscar palavras em arquivos de texto, utilizando três estruturas de dados diferentes: Tabela Hash, Árvore AVL e Árvore Rubro-Negra. Cada estrutura oferece suas próprias vantagens em termos de complexidade de tempo e adequação para diferentes tipos de operações e tamanhos de dados.

- Tabela Hash: Ideal para operações rápidas de inserção e busca em grandes conjuntos de dados, desde que as colisões sejam minimamente controladas.
- Árvore AVL: Garante operações balanceadas de busca, inserção e remoção, mantendo a altura da árvore em  $O(\log n)$ .

UNIVERSIDADE FEDERAL DO MARANHÃO  
Centro de Ciências Exatas e Tecnológicas  
Ciências da Computação  
Estrutura de Dados 2 – Prof. Dr. João Dalyson

Rondineli Seba Salomão Matrícula: 20200022063  
Gabriel Belo Pereira dos Reis Matrícula: 2022029760

- Árvore Rubro-Negra: Similar à árvore AVL, mas com uma implementação de balanceamento mais flexível, utilizando propriedades de coloração.

```
Digite o valor de X (número mínimo de caracteres para considerar uma palavra):  
5  
Escolha a estrutura de dados para o índice remissivo:  
1 - Hash  
2 - Árvore AVL  
3 - Árvore Rubro-Negra  
1  
Escolha a estratégia de tratamento de colisões:  
1 - Tentativa Linear  
2 - Tentativa Quadrática
```

Execução do código e tela de seleção para criação do índice.

Rondineli Seba Salomão Matrícula: 20200022063  
Gabriel Belo Pereira dos Reis Matrícula: 2022029760

## PROGRAMA 2 – ARVORE B FUNCIONAL

### DESCRIÇÃO GERAL DO CÓDIGO

O projeto Árvore B em Java implementa uma estrutura de dados que suporta operações de inserção, exclusão e busca. A árvore B utiliza a memória secundária simulada (arquivos de texto) para armazenar seus nós. Além disso, foi desenvolvido um sistema de gerenciamento de usuários que permite inserir, buscar e excluir usuários, bem como visualizar a estrutura da árvore.

### 1. ESTRUTURA DO PROJETO

#### FUNÇÕES PRINCIPAIS E COMPLEXIDADE DE TEMPO

##### Classe Principal: Main.java

- **processarArquivos(int x, EstruturaDeDados estrutura)**
  - Inicializa a árvore B e executa o menu de operações.
  - Complexidade de Tempo:  $O(1)$
- **inserirUsuario(OperacoesArvoreB<Usuario> operacoesArvoreB)**
  - Insere um novo usuário na árvore B. - Complexidade de Tempo:  $O(t \log_t n)$
- **buscarUsuario(OperacoesArvoreB<Usuario> operacoesArvoreB)**
  - Busca um usuário na árvore B base no ID fornecido pelo usuário..
  - Complexidade de Tempo:  $O(t \log_t n)$
- **excluirUsuario(OperacoesArvoreB<Usuario> operacoesArvoreB)**
  - Exclui um usuário da árvore B base no ID fornecido pelo usuário..
  - Complexidade de Tempo:  $O(t \log_t n)$
- **visualizarEstrutura(ArvoreB<Usuario> arvoreB)**
  - Visualiza a estrutura da árvore B, mostrando todos os nós e suas chaves..
  - Complexidade de Tempo:  $O(n)$

##### Estrutura de Dados: ArvoreB.java

- **ArvoreB(Class<T> clazz)**
  - Construtor da árvore B, inicializa uma árvore vazia.
  - Complexidade de Tempo:  $O(1)$

Parâmetros:

clazz: Classe dos objetos armazenados na árvore.

Rondineli Seba Salomão Matrícula: 20200022063  
Gabriel Belo Pereira dos Reis Matrícula: 2022029760

**- ArvoreB(NoArvoreB<T> raiz, Class<T> clazz)**

- Construtor da árvore B com nó raiz já existente.

Complexidade de Tempo:  $O(1)$

Parâmetros:

raiz: Nó raiz da árvore.

clazz: Classe dos objetos armazenados na árvore.

- Getters e Setters

- Inclui métodos para obter e definir a raiz da árvore.

Complexidade de Tempo:  $O(1)$

**Estrutura de Dados: NoArvoreB.java**

**- ArvoreB <T extends Comparable<T> & Serializable>**

Construtor do nó da árvore B. Inicializa um nó como folha ou interno e aloca espaço para as chaves e filhos.

Complexidade de Tempo:  $O(1)$

Parâmetros:

folha: booleano indicando se o nó é uma folha.

Inicializa os arrays de chaves e filhos com base no grau mínimo t.

- Getters e Setters

- Inclui métodos para obter e definir o número de chaves, a lista de chaves, a lista de filhos e se o nó é folha.

- Complexidade de Tempo:  $O(1)$

**Gerenciamento de Arquivos: GerenciadorArquivo.java**

salvarArvore(String nomeArquivo, NoArvoreB<T> raiz)

Salva a árvore B em um arquivo.

Complexidade de Tempo:  $O(n)$  onde n é o número de nós.

- Parâmetros:

- nomeArquivo: Nome do arquivo onde a árvore será salva.

- raiz: Nó raiz da árvore a ser salva.

- carregarArvore(String nomeArquivo, Class<T> clazz)

- Carrega a árvore B de um arquivo.

- Complexidade de Tempo:  $O(n)$

- Parâmetros:

- nomeArquivo: Nome do arquivo de onde a árvore será carregada.

- clazz: Classe dos objetos armazenados na árvore.

Rondineli Seba Salomão Matrícula: 20200022063  
Gabriel Belo Pereira dos Reis Matrícula: 2022029760

**Classe Auxiliar: OperacoesArvoreB.java**

OperacoesArvoreB(ArvoreB<T> arvoreB)

Construtor das operações da árvore B, recebe uma instância de ArvoreB.

Complexidade de Tempo:  $O(1)$

Parâmetros:

arvoreB: Instância da árvore B onde as operações serão realizadas.

inserir(T chave)

Insere uma chave na árvore B, lidando com a possível divisão de nós se necessário, e salva os dados.

Complexidade de Tempo:  $O(t \log_t n)$

Parâmetros:

chave: Chave a ser inserida na árvore.

**inserirNaoCheio(NoArvoreB<T> x, T chave)**

Insere uma chave em um nó que não está cheio, localizando a posição correta para a chave.

Complexidade de Tempo:  $O(t \log_t n)$

Parâmetros:

x: Nó onde a chave será inserida.

chave: Chave a ser inserida.

**dividirFilho(NoArvoreB<T> x, int i, NoArvoreB<T> y)**

Divide um nó filho quando ele está cheio, criando um novo nó e redistribuindo as chaves.

Complexidade de Tempo:  $O(t \log_t n)$

Parâmetros:

x: Nó pai do nó cheio.

i: Índice do nó filho que será dividido.

y: Nó filho que será dividido.

**excluir(T chave)**

Exclui uma chave da árvore B, lidando com a possível fusão de nós se necessário, e salva os dados.

Complexidade de Tempo:  $O(t \log_t n)$

Parâmetros:

chave: Chave a ser excluída.

**excluir(NoArvoreB<T> x, T chave)**

Exclui uma chave de um nó específico, garantindo que a árvore continue balanceada.

Complexidade de Tempo:  $O(t \log_t n)$

Parâmetros:



Rondineli Seba Salomão Matrícula: 20200022063  
Gabriel Belo Pereira dos Reis Matrícula: 2022029760

x: Nó de onde a chave será excluída.

chave: Chave a ser excluída.

**buscarChave( $\text{NoArvoreB} < T > x, T \text{ chave}$ )**

Busca a posição de uma chave em um nó específico.

Complexidade de Tempo:  $O(t \log_t n)$

Parâmetros:

x: Nó onde a busca será realizada.

chave: Chave a ser buscada.

**removeDeFolha( $\text{NoArvoreB} < T > x, \text{int idx}$ )**

Remove uma chave de um nó folha, deslocando as chaves para manter a ordem.

Complexidade de Tempo:  $O(t \log_t n)$

Parâmetros:

x: Nó folha de onde a chave será removida.

idx: Índice da chave a ser removida.

**removeDeNaoFolha( $\text{NoArvoreB} < T > x, \text{int idx}$ )**

Remove uma chave de um nó que não é folha, lidando com o predecessor ou sucessor da chave.

Complexidade de Tempo:  $O(t \log_t n)$

Parâmetros:

x: Nó de onde a chave será removida.

idx: Índice da chave a ser removida.

**getPredecessor( $\text{NoArvoreB} < T > x, \text{int idx}$ )**

Obtém o predecessor de uma chave, que é a maior chave no subárvore à esquerda.

Complexidade de Tempo:  $O(t \log_t n)$

Parâmetros:

x: Nó de onde será obtido o predecessor.

idx: Índice da chave cujo predecessor será obtido.

**getSucessor( $\text{NoArvoreB} < T > x, \text{int idx}$ )**

Obtém o sucessor de uma chave, que é a menor chave no subárvore à direita.

Complexidade de Tempo:  $O(t \log_t n)$

Parâmetros:

x: Nó de onde será obtido o sucessor.

idx: Índice da chave cujo sucessor será obtido.

**juntar( $\text{NoArvoreB} < T > x, \text{int idx}$ )**

Junta dois nós filhos em um único nó, reduzindo o número de filhos de um nó pai.

Rondineli Seba Salomão Matrícula: 20200022063  
Gabriel Belo Pereira dos Reis Matrícula: 2022029760

Complexidade de Tempo:  $O(t \log_t n)$

Parâmetros:

x: Nó pai cujos filhos serão juntados.

idx: Índice do nó filho que será juntado com o próximo.

preencher( $\text{NoArvoreB}<T> x, \text{int idx}$ )

Preenche um nó que tem menos do que t chaves, garantindo que ele tenha pelo menos t chaves.

Complexidade de Tempo:  $O(t \log_t n)$

Parâmetros:

x: Nó que será preenchido.

idx: Índice do nó filho que será preenchido.

pegarDoAnterior( $\text{NoArvoreB}<T> x, \text{int idx}$ )

Pega uma chave do nó anterior (à esquerda) para preencher o nó atual.

Complexidade de Tempo:  $O(t \log_t n)$

Parâmetros:

x: Nó que será preenchido.

idx: Índice do nó filho que pegará a chave do anterior.

pegarDoProximo( $\text{NoArvoreB}<T> x, \text{int idx}$ )

Pega uma chave do próximo nó (à direita) para preencher o nó atual.

Complexidade de Tempo:  $O(t \log_t n)$

Parâmetros:

x: Nó que será preenchido.

idx: Índice do nó filho que pegará a chave do próximo.

buscar( $T \text{ chave}$ )

Busca uma chave na árvore B, começando pela raiz.

Complexidade de Tempo:  $O(t \log_t n)$

Parâmetros:

chave: Chave a ser buscada.

buscar( $\text{NoArvoreB}<T> \text{no}, T \text{ chave}$ )

Busca uma chave em um nó específico da árvore.

Complexidade de Tempo:  $O(t \log_t n)$

Parâmetros:

no: Nó onde a busca será realizada.

chave: Chave a ser buscada.

salvarDados()

Rondineli Seba Salomão Matrícula: 20200022063  
Gabriel Belo Pereira dos Reis Matrícula: 2022029760

Salva a estrutura da árvore B em um arquivo binário, persistindo os dados.

Complexidade de Tempo:  $O(n)$

salvarDadosTexto()

Salva a estrutura da árvore B em um arquivo de texto, facilitando a visualização.

Complexidade de Tempo:  $O(n)$

salvarNoTexto(BufferedWriter writer, NoArvoreB<T> no, int nivel, String posicao)

Salva um nó específico em um arquivo de texto, incluindo detalhes sobre a posição do nó.

Complexidade de Tempo:  $O(n)$

Parâmetros:

writer: Escritor do arquivo de texto.

no: Nó a ser salvo.

nivel: Nível do nó na árvore.

posicao: Posição do nó (Raiz, Folha Esquerda, Folha Direita, Interno).

noParaString(NoArvoreB<T> no)

Converte um nó em string para exibição.

Complexidade de Tempo:  $O(n)$

Parâmetros:

no: Nó a ser convertido em string.

Classe Usuário: Usuario.java

Usuario(int id\_usuario, String login, String nome, String email, Date data\_nascimento, String foto)

Construtor do usuário, inicializa todos os campos.

Complexidade de Tempo:  $O(1)$

Parâmetros:

id\_usuario: ID do usuário.

login: Login do usuário.

nome: Nome do usuário.

email: Email do usuário.

data\_nascimento: Data de nascimento do usuário.

foto: Foto do usuário.

Getters e Setters

Inclui métodos para obter e definir os atributos do usuário.

Complexidade de Tempo:  $O(1)$

compareTo(Usuario other)

Compara este usuário com outro usuário com base no ID.

Rondineli Seba Salomão Matrícula: 20200022063  
Gabriel Belo Pereira dos Reis Matrícula: 2022029760

Complexidade de Tempo:  $O(1)$

Parâmetros:

other: Outro usuário para comparação.

toString()

Converte o usuário em string para exibição.

Complexidade de Tempo:  $O(1)$

## CONCLUSÃO

O projeto de Árvore B implementado em Java apresenta uma solução eficaz para armazenar e buscar dados de usuários utilizando uma estrutura de árvore B que suporta inserção, exclusão e busca. A implementação utiliza memória secundária para persistência dos dados, garantindo que a árvore seja mantida entre execuções do programa.

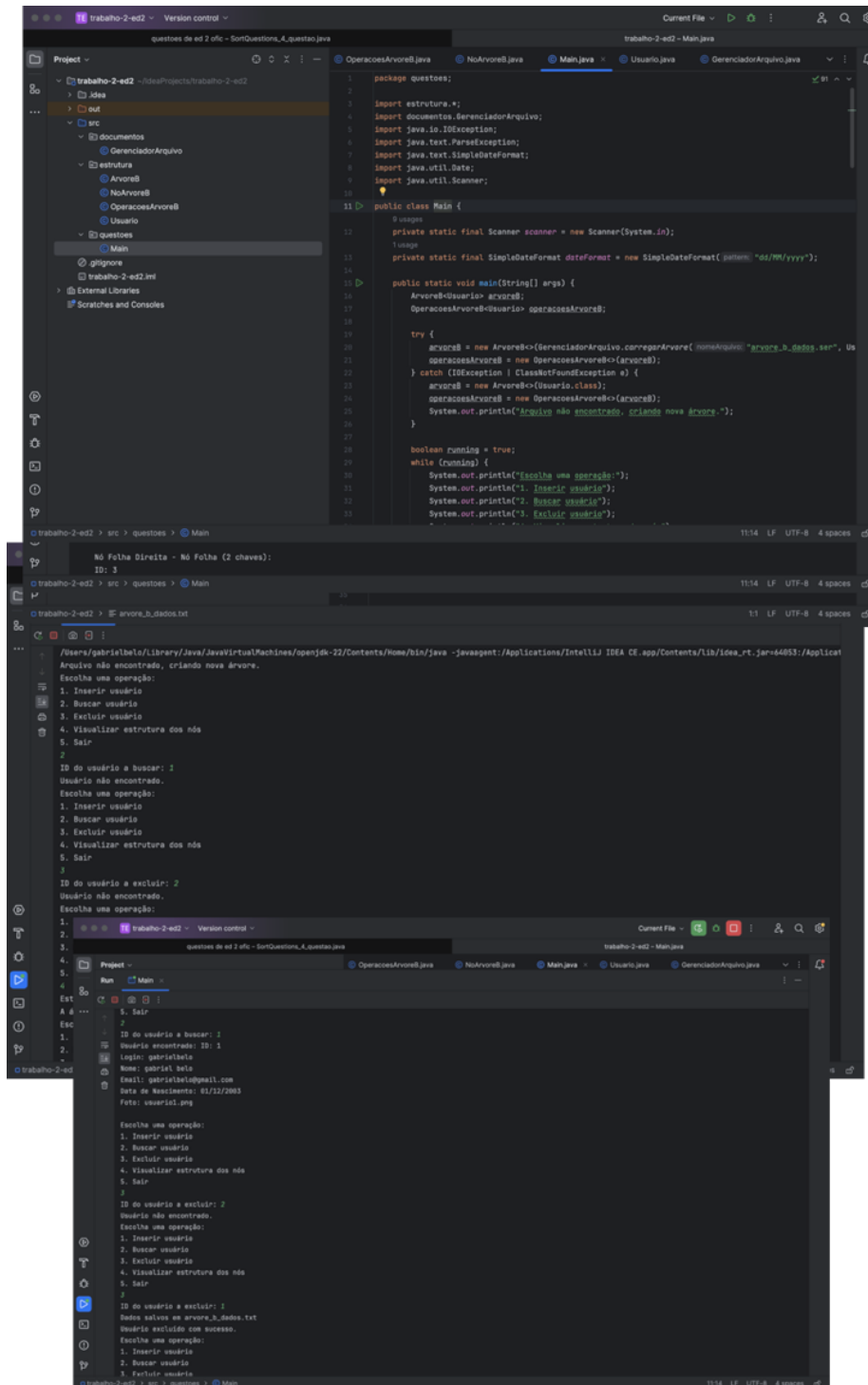
Árvore B: Garante operações balanceadas de busca, inserção e remoção, mantendo a altura da árvore em  $O(\log_t n)$ .

Abaixo segue a execução do código em um exemplo prático com as operações básicas e a visualização da estrutura da árvore em diferentes estados.

Exemplo de Execução do Código:

UNIVERSIDADE FEDERAL DO MARANHÃO  
Centro de Ciências Exatas e Tecnológicas  
Ciências da Computação  
Estrutura de Dados 2 – Prof. Dr. João Dalyson

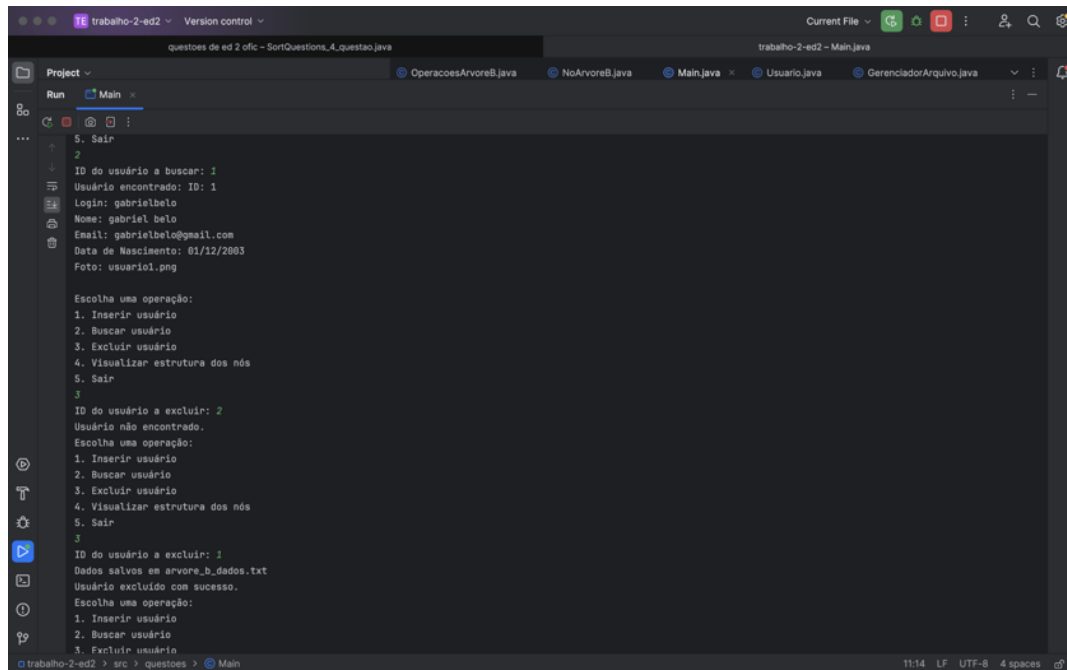
Rondineli Seba Salomão Matrícula: 20200022063  
Gabriel Belo Pereira dos Reis Matrícula: 2022029760



Esta imagem mostra a estrutura do projeto em IntelliJ IDEA. No painel de projeto, vemos a organização das pastas e arquivos do projeto. A pasta src contém subpastas para documentos, estrutura, questoes, e classes como Main.java, ArvoreB.java, NoArvoreB.java, OperacoesArvoreB.java, e Usuario.java.

UNIVERSIDADE FEDERAL DO MARANHÃO  
Centro de Ciências Exatas e Tecnológicas  
Ciências da Computação  
Estrutura de Dados 2 – Prof. Dr. João Dalyson

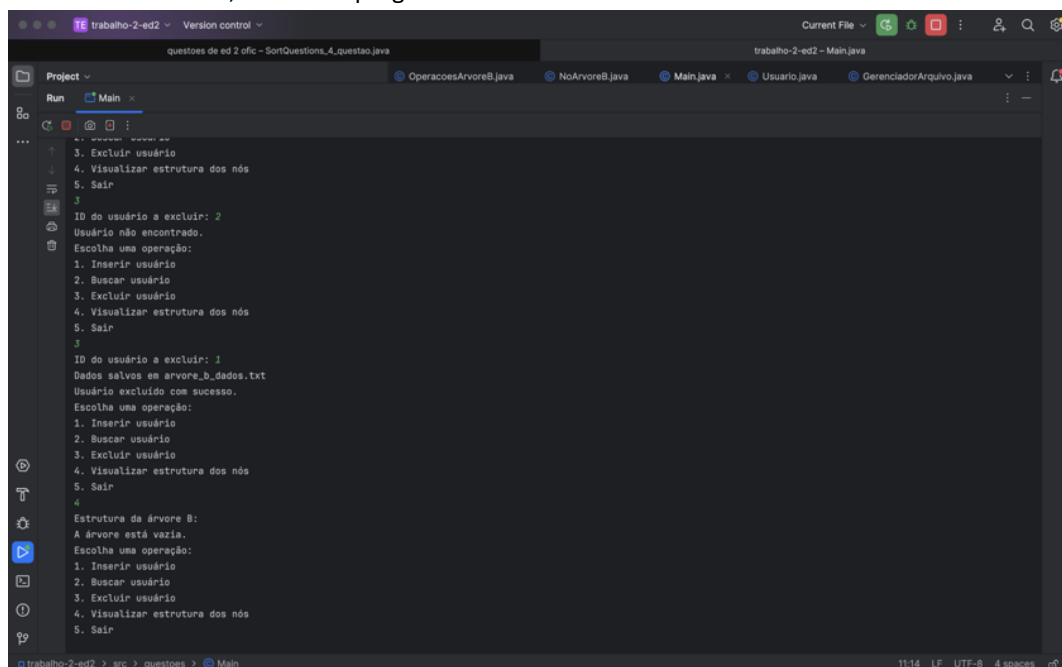
Rondineli Seba Salomão Matrícula: 20200022063  
Gabriel Belo Pereira dos Reis Matrícula: 2022029760



```
5. Sair
2
ID do usuário a buscar: 1
Usuário encontrado: ID: 1
Login: gabrielbelo
Nome: gabriel belo
Email: gabrielbelo@gmail.com
Data de Nascimento: 01/12/2003
Foto: usuario1.png

Escolha uma operação:
1. Inserir usuário
2. Buscar usuário
3. Excluir usuário
4. Visualizar estrutura dos nós
5. Sair
3
ID do usuário a excluir: 2
Usuário não encontrado.
Escolha uma operação:
1. Inserir usuário
2. Buscar usuário
3. Excluir usuário
4. Visualizar estrutura dos nós
5. Sair
3
ID do usuário a excluir: 1
Dados salvos em arvore_b_dados.txt
Usuário excluído com sucesso.
Escolha uma operação:
1. Inserir usuário
2. Buscar usuário
3. Excluir usuário
```

Esta imagem exibe a execução inicial do programa onde o arquivo da árvore B não é encontrado, resultando na criação de uma nova árvore. O menu de operações é mostrado com opções para inserir, buscar, excluir usuários, visualizar a estrutura dos nós, ou sair do programa.



```
3. Excluir usuário
4. Visualizar estrutura dos nós
5. Sair
3
ID do usuário a excluir: 2
Usuário não encontrado.
Escolha uma operação:
1. Inserir usuário
2. Buscar usuário
3. Excluir usuário
4. Visualizar estrutura dos nós
5. Sair
3
ID do usuário a excluir: 1
Dados salvos em arvore_b_dados.txt
Usuário excluído com sucesso.
Escolha uma operação:
1. Inserir usuário
2. Buscar usuário
3. Excluir usuário
4. Visualizar estrutura dos nós
5. Sair
4
Estrutura da Árvore B:
A árvore está vazia.
Escolha uma operação:
1. Inserir usuário
2. Buscar usuário
3. Excluir usuário
4. Visualizar estrutura dos nós
5. Sair
```

A terceira imagem mostra a tentativa de buscar e excluir usuários inexistentes. O sistema informa que o usuário não foi encontrado para as operações de busca e exclusão.

UNIVERSIDADE FEDERAL DO MARANHÃO  
Centro de Ciências Exatas e Tecnológicas  
Ciências da Computação  
Estrutura de Dados 2 – Prof. Dr. João Dalyson

Rondineli Seba Salomão Matrícula: 20200022063  
Gabriel Belo Pereira dos Reis Matrícula: 2022029760

```
1 Estrutura da Árvore B:  
2 Nó Raiz - Nó Interno (1 chaves):  
3 ID: 2  
4 Login: gabrielbelo2  
5 Nome: gabriel belo 2  
6 Email: gabrielbelo2@gmail.com  
7 Data de Nascimento: 01/12/2000  
8 Foto: usuario3.png  
9  
10 Nó Folha Esquerda - Nó Folha (1 chaves):  
11 ID: 1  
12 Login: gabrielbelo  
13 Nome: gabriel belo  
14 Email: gabrielbelo@gmail.com  
15 Data de Nascimento: 01/12/2003  
16 Foto: usuario1.png  
17  
18 Nó Folha Direita - Nó Folha (2 chaves):  
19 ID: 3  
20 Login: gabrielbelo3  
21 Nome: gabriel belo 3  
22 Email: gabrielbelo3  
23 Data de Nascimento: 01/12/1999  
24 Foto: usuario2.png  
25  
26 ID: 4  
27 Login: gabrielbelo4  
28 Nome: gabriel belo 4  
29 Email: gabrielbelo4@gmail.com  
30 Data de Nascimento: 01/12/2005  
31 Foto: usuario4.png  
32  
33  
34  
35
```

Nesta imagem, o usuário foi inserido com sucesso na árvore B. A estrutura da árvore B é visualizada e indica que a árvore está vazia, pois nenhuma operação de inserção foi executada antes da visualização.

```
1 ID: 2  
2 Login: gabrielbelo2  
3 Nome: gabriel belo 2  
4 Email: gabrielbelo2@gmail.com  
5 Data de Nascimento: 01/12/2000  
6 Foto: usuario3.png  
7  
8 Escolha uma operação:  
9 1. Inserir usuário  
10 2. Buscar usuário  
11 3. Excluir usuário  
12 4. Visualizar estrutura dos nós  
13 5. Sair  
14 1  
15 ID do usuário: 3  
16 Login: gabrielbelo3  
17 Nome: gabriel belo 3  
18 Email: gabrielbelo3  
19 Data de nascimento (dd/MM/yyyy): 01/12/1999  
20 Foto: usuario2.png  
21 Dados salvos em arvore_b_dados.ser  
22 Dados salvos em arvore_b_dados.txt  
23 Usuário inserido com sucesso.  
24 Escolha uma operação:  
25 1. Inserir usuário  
26 2. Buscar usuário  
27 3. Excluir usuário  
28 4. Visualizar estrutura dos nós  
29 5. Sair
```

Apresenta a estrutura da árvore B após a inserção dos três primeiros usuários. A árvore é visualizada como um nó raiz com três chaves.

UNIVERSIDADE FEDERAL DO MARANHÃO  
Centro de Ciências Exatas e Tecnológicas  
Ciências da Computação  
Estrutura de Dados 2 – Prof. Dr. João Dalyson

Rondineli Seba Salomão Matrícula: 20200022063  
Gabriel Belo Pereira dos Reis Matrícula: 2022029760

Mostra a inserção do quarto usuário com ID 4. Detalhes do usuário são fornecidos, e a árvore B é salva, confirmando a inserção bem-sucedida do usuário.



UNIVERSIDADE FEDERAL DO MARANHÃO  
Centro de Ciências Exatas e Tecnológicas  
Ciências da Computação  
Estrutura de Dados 2 – Prof. Dr. João Dalyson

Rondineli Seba Salomão	Matrícula: 20200022063
Gabriel Belo Pereira dos Reis	Matrícula: 2022029760

UNIVERSIDADE FEDERAL DO MARANHÃO  
Centro de Ciências Exatas e Tecnológicas  
Ciências da Computação  
Estrutura de Dados 2 – Prof. Dr. João Dalyson

Rondineli Seba Salomão Matrícula: 20200022063  
Gabriel Belo Pereira dos Reis Matrícula: 2022029760

Exibe a estrutura da árvore B após a inserção do quarto usuário. A árvore é visualizada com um nó raiz interno e dois nós folha contendo os usuários.

Mostra a estrutura do projeto com foco nos arquivos do pacote estruturas. O código da classe ArvoreB está visível com as importações e a declaração da classe.

Exibe a execução do programa com a inserção de mais um usuário, seguida pela busca de um usuário inexistente (ID 2), resultando em um usuário não encontrado.

Rondineli Seba Salomão Matrícula: 20200022063  
Gabriel Belo Pereira dos Reis Matrícula: 2022029760

Mostra a inserção de mais um usuário com ID 4, seguida pela visualização da estrutura da árvore B. A árvore é visualizada com um nó raiz interno e dois nós folha contendo os usuários. Mostra a exclusão de um usuário (ID 2), resultando em um usuário não encontrado. Em seguida, a exclusão de outro usuário (ID 1) é bem-sucedida, e a estrutura da árvore B é visualizada como vazia.

Exibe a execução do programa com a inserção de mais um usuário, seguida pela exclusão de um usuário e a visualização da estrutura da árvore B. A árvore é visualizada como vazia após a exclusão. Mostra a execução do programa com a inserção de mais um usuário, seguida pela visualização da estrutura da árvore B. A árvore é visualizada com um nó raiz contendo duas chaves.

Exibe a execução do programa com a inserção de mais um usuário, seguida pela visualização da estrutura da árvore B. A árvore é visualizada com um nó raiz interno e dois nós folha contendo os usuários. Mostra a execução do programa com a exclusão de um usuário, seguida pela visualização da estrutura da árvore B. A árvore é visualizada com um nó raiz interno e dois nós folha contendo os usuários.