**Centro Federal de Educação Tecnológica de Minas Gerais - Departamento de Computação**

**Laboratório de Algoritmos e Estruturas de Dados II (2ECOM.025)**

# Lista de Exercícios 3

*Professores:* Amadeu Almeida e Thiago Rodrigues **Data de entrega:** 06 de Setembro de 2019

Instruções: leia com atenção **todas** as orientações abaixo antes de começar a lista.

- Os exercícios deverão ser feitos individualmente e valem 5 pontos.
- Submeta no SIGAA um arquivo compactado chamado **Lista3.zip**. Ele deverá conter:
  - As classes Java que implementam as soluções das tarefas.
  - Um relatório em **PDF** com os gráficos dos experimentos e suas respectivas explicações.
- O prazo de entrega é até às 16:40 do dia 6 de setembro.
- Códigos fonte em PDF e submissões em atraso serão desconsiderados.
- O enunciado desta lista possui 2 exercícios e 4 páginas.

## Exercício 1: Classe ArvoreB

Este exercício objetiva implementar uma classe que cria e manipula uma Árvore B aplicando os conceitos estudados durante a disciplina teórica.

1. Abra um projeto chamado **Lista3**. Em seguida, crie um arquivo nomeado ArvoreB.java.
2. Implemente a classe **Pagina** conforme especificado abaixo. Ela deve ser inserida dentro da classe **ArvoreB**, de modo que cada página da Árvore B seja um objeto da classe **Pagina**.

```
private static class Pagina {

    // Numero de itens que a pagina contem atualmente
    int numeroCorrenteItens;

    // Vetor que armazena todos os itens da pagina
    int itensPagina[];

    // Vetor que indica quem sao as paginas filhas (ele aponta para todas as
       filhas)
    Pagina paginasFilhas[];

    // Metodo construtor da classe pagina
    public Pagina(int qtdemaximaRegistros) {
        this.numeroCorrenteItens = 0;
        this.itensPagina = new int[qtdemaximaRegistros];
        this.paginasFilhas = new Pagina[qtdemaximaRegistros + 1];
    }
}
```

3. Transcreva o método **insereNaPagina**. Seu objetivo é inserir um registro em uma das páginas na árvore B.

```
private void insereNaPagina(Pagina paginaAtual, int registro, Pagina
    filhaDireita) {
    int k = paginaAtual.numeroCorrenteItens - 1;

    while (k >= 0 && (registro - paginaAtual.itensPagina[k]) < 0) {
```

```
            paginaAtual.itensPagina[k + 1] = paginaAtual.itensPagina[k];
            paginaAtual.paginasFilhas[k + 2] = paginaAtual.paginasFilhas[k + 1];
            k--;
        }

        paginaAtual.itensPagina[k + 1] = registro;
        paginaAtual.paginasFilhas[k + 2] = filhaDireita;
        paginaAtual.numeroCorrenteItens++;
    }
```

4. Implemente os atributos da classe **ArvoreB**:
   - private Pagina pagRaiz;
   - private final int minimoRegistrosPagina, maximoRegistrosPagina;
   - private boolean arvoreDesbalanceada;
   - private int regRetorno;
   - private int paginasVisitadas;
   - private int numeroComparacoes;
5. Inclua os métodos construtor e de inserção dos registros da árvore conforme o modelo abaixo.

```
public ArvoreB(int qtdeminimaRegistros) {
    this.pagRaiz = null;
    this.minimoRegistrosPagina = qtdeminimaRegistros;
    this.maximoRegistrosPagina = 2 * qtdeminimaRegistros;
    arvoreDesbalanceada = false;
    regRetorno = -1;
    paginasVisitadas = 0;
    numeroComparacoes = 0;
}

private Pagina insere(int registro, Pagina paginaAtual) {
    Pagina paginaRetorno = null;

    if (paginaAtual == null) {
        arvoreDesbalanceada = true;
        regRetorno = registro;
    } else {

        int i = 0;

        while ((i < paginaAtual.numeroCorrenteItens - 1) && (registro -
            paginaAtual.itensPagina[i] > 0)) {
            i++;
        }

        if (registro == paginaAtual.itensPagina[i]) {
            System.out.println("Erro: Registro ja existente");
            arvoreDesbalanceada = false;
        } else {
            if (registro - paginaAtual.itensPagina[i] > 0) {
                i++;
            }
            paginaRetorno = insere(registro, paginaAtual.paginasFilhas[i]);
            if (arvoreDesbalanceada) {
                if (paginaAtual.numeroCorrenteItens < this.maximoRegistrosPagina
                    ) {
                    this.insereNaPagina(paginaAtual, regRetorno, paginaRetorno);
                    arvoreDesbalanceada = false;
                    paginaRetorno = paginaAtual;
                }
                else {
```

```
                        Pagina apTemp = new Pagina(this.maximoRegistrosPagina);
                        apTemp.paginasFilhas[0] = null;

                        if (i <= this.minimoRegistros) {
                            this.insereNaPagina(apTemp, paginaAtual.itensPagina[this
                                .maximoRegistros - 1], paginaAtual.paginasFilhas[this
                                .maximoRegistros]);
                            paginaAtual.numeroCorrenteItens--;
                            this.insereNaPagina(paginaAtual, regRetorno,
                                paginaRetorno);
                        } else {
                            this.insereNaPagina(apTemp, regRetorno, paginaRetorno);
                        }
                        for (int j = this.minimoRegistrosPagina + 1; j < this.
                            maximoRegistrosPagina; j++) {
                            this.insereNaPagina(apTemp, paginaAtual.itensPagina[j],
                                paginaAtual.paginasFilhas[j + 1]);
                            paginaAtual.paginasFilhas[j + 1] = null;
                        }
                        paginaAtual.numeroCorrenteItens = this.minimoRegistrosPagina
                            ;
                        apTemp.paginasFilhas[0] = paginaAtual.paginasFilhas[this.
                            minimoRegistros + 1];
                        regRetorno = paginaAtual.itensPagina[this.minimoRegistros];
                        paginaRetorno = apTemp;
                    }
                }
            }
        }
        return (arvoreDesbalanceada ? paginaRetorno : paginaAtual);
    }
```

6. Implemente todos os métodos a seguir:
   - private int pesquisa(int reg, Pagina ap) - busca um registro na Árvore SBB.
   - public int pesquisa(int reg) - inicia a busca por um elemento da árvore a partir do nó raiz.
   - public void insere(int reg) - inicializa o processo de inserção de uma chave à partir do nó raiz.
   - public int getPaginasVisitadas() - retorna o número de páginas visitadas durante a busca por um registro.
   - public int getNumeroComparacoes() - obtém a quantidade de comparações necessárias para encontrar uma chave ou determinar que ela não está na árvore.
7. Explique o objetivo de cada um dos métodos implementados nos itens 3, 5 e 6 por meio de comentários no próprio código.

**Exercício 2: Experimentos computacionais**

As tarefas deste exercício visam testar a eficácia do método de pesquisa da classe **ArvoreB** para diferentes tamanhos de árvores e páginas.

1. Experimento 1:
   (a) Crie 10 **Árvores B de ordem 2** diferentes contendo $N$ elementos **ORDENADOS**, ou seja, inseridos em ordem crescente, no qual $N$ varia em intervalos de 10.000, entre os números 10.000 e 100.000.
   (b) Para cada uma das árvores geradas no exercício (a), pesquise por um elemento não pertencente à árvore e retorne o número de páginas visitadas e de comparações feitas em cada pesquisa.
2. Experimento 2:
   (a) Conceba 10 **Árvores B de ordem 4** distintas contendo $N$ elementos **ORDENADOS**, ou seja, adicionados em ordem crescente, no qual $N$ varia em intervalos de 10.000, entre os números 10.000 e 100.000.

(b) Pesquise por um elemento não pertencente a cada uma das árvores geradas no exercício (a) e retorne o número de páginas visitadas e de comparações realizadas em cada pesquisa.

3. Experimento 3:

   (a) Gere 10 **Árvores B de ordem 8** diferentes contendo $N$ elementos **ORDENADOS**, ou seja, incluídos em ordem crescente, no qual $N$ varia em intervalos de 10.000, entre os números 10.000 e 100.000.

   (b) Para cada uma das árvores geradas no exercício (a), pesquise por um elemento não pertencente à árvore e retorne o número de páginas visitadas e de comparações feitas em cada pesquisa.

4. Experimento 4:

   (a) Crie 10 **Árvores B de ordem 16** distintas contendo $N$ elementos **ORDENADOS**, ou seja, incorporados em ordem crescente, no qual $N$ varia em intervalos de 10.000, entre os números 10.000 e 100.000.

   (b) Pesquise por um elemento não pertencente a cada uma das árvores criadas no exercício (a) e retorne o número de páginas visitadas e de comparações realizadas em cada pesquisa.

5. Faça dois gráficos:

   - Um que apresente, para cada valor de $N$, o número de páginas visitadas e de comparações feitas durante as buscas dos experimentos 1, 2, 3 e 4 desta lista. Neste gráfico, o eixo X refere-se a quantidade de elementos de cada árvore ($N$) enquanto o numero de comparações e de páginas investigadas são retratados pelo eixo $Y$. Em seguida, explique relação entre o número páginas visitadas e o de comparações feitas.
   - Um que compare, para cada valor de $N$, o número de nós visitados durante as pesquisas do experimento 1 da lista 2 e o de páginas investigadas no decorrer das buscas do experimento 3 da lista 3. O eixo X deste gráfico contem a quantidade de elementos de cada árvore ($N$) e o eixo $Y$ é composto pelo número de nós/páginas visitados. Depois, esclareça as razões pelas quais a quantidade de nós visitados na **Árvore SBB** é consideravelmente maior que o número de páginas investigadas na **Árvore B**.