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

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

Lista de Exercícios 2

Professores: Amadeu Almeida e Thiago Rodrigues **Data de entrega:** 30 de Agosto de 2019

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

- Esta lista de exercícios é individual e vale 5 pontos.
- Submeta no SIGAA um arquivo compactado chamado **Lista2.zip**. Ele deverá conter:
  - As classes Java que implementam as soluções dos exercícios desta lista.
  - Um relatório em **PDF** com os gráficos dos experimentos e suas respectivas explicações.
- O prazo de entrega é até às 16:40 do dia 30 de agosto.
- Códigos fonte em PDF e submissões em atraso serão desconsiderados.
- O enunciado desta lista possui 3 exercícios e 4 páginas.

Exercício 1: Classe Item

Este exercício objetiva escrever os métodos da classe **Item** que implementa um tipo abstrato de dados utilizado para representar a chave de cada nó da árvore binária.

1. Abra um projeto chamado **Lista2**. Em seguida, crie um arquivo nomeado Item.java.
2. Implemente a classe **Item** conforme especificada abaixo.

```
package lista1;

public class Item {
    private int chave;
    public Item(int chave) {
        this.chave = chave;
    }
    public int compara(Item it) {
        Item item = it;
        if (this.chave < item.chave)
            return -1;
        else if (this.chave > item.chave)
            return 1;
        return 0;
    }

    public int getChave() {
        return chave;
    }
}
```

Exercício 2: Classe ÁrvoreSBB

Este exercício visa implementar uma classe que representa uma Árvore SBB aplicando os conceitos aprendidos durante a disciplina teórica.

1. No projeto **Lista 2**, crie um arquivo chamado ArvoreSBB.java.

2. Implemente a classe **No** conforme especificada abaixo. Essa classe deve ser inserida dentro da classe **ArvoreSBB** de modo que cada nó da árvore seja um objeto da classe **No**.

```
private static class No {
    Item reg;
    No esq, dir;
    byte incE, incD;
}
```

3. Inclua os métodos auxiliares **ee**, **ed**, **dd** e **de**. O objetivo destas funções é balancear a Árvore SBB durante as inserções das chaves.

```
private No ee(No ap) {
        No ap1 = ap.esq;
        ap.esq = ap1.dir;
        ap1.dir = ap;
        ap1.incE = Vertical;
        ap.incE = Vertical;
        ap = ap1;
        return ap;
    }

    private No ed(No ap) {
        No ap1 = ap.esq;
        No ap2 = ap1.dir;
        ap1.incD = Vertical;
        ap.incE = Vertical;
        ap1.dir = ap2.esq;
        ap2.esq = ap1;
        ap.esq = ap2.dir;
        ap2.dir = ap;
        ap = ap2;
        return ap;
    }

    private No dd(No ap) {
        No ap1 = ap.dir;
        ap.dir = ap1.esq;
        ap1.esq = ap;
        ap1.incD = Vertical;
        ap.incD = Vertical;
        ap = ap1;
        return ap;
    }

    private No de(No ap) {
        No ap1 = ap.dir;
        No ap2 = ap1.esq;
        ap1.incE = Vertical;
        ap.incD = Vertical;
        ap1.esq = ap2.dir;
        ap2.dir = ap1;
        ap.dir = ap2.esq;
        ap2.esq = ap;
        ap = ap2;
        return ap;
    }
```

4. Implemente os atributos da classe **ArvoreSBB**:

   - private No raiz
   - private long nosVisitados
   - private int quantidadeNiveisArvore

- private static final byte Horizontal = 0;
- private static final byte Vertical = 1;
- private boolean propSBB;

5. Implemente os métodos da classe **ArvoreSBB**:
   - public ArvoreSBB(): inicializa o nó raiz.
   - private No insere(Item reg, No pai, No filho, boolean filhoEsq): insere uma chave na Árvore SBB e balanceia-a.
   - private Item pesquisa(Item reg, No p): busca uma chave na Árvore SBB.
   - public void insere(Item reg): inicializa o processo de inserção de uma chave à partir do nó raiz.
   - public void pesquisa(Item reg): inicia a busca por um elemento da árvore a partir do nó raiz.
   - public long getNosVisitados(): retorna o número de nós visitados durante a busca de um registro.
   - public int getQuantidadeNiveisArvore(): retorna a quantidade de níveis em uma Árvore SBB.
6. Transcreva os métodos que calculam a quantidade de níveis de uma árvore SBB.

```
private void calculaTamanho(No p, int nivel) {
    if (p == null) {
        return;
    }
    if (this.primeiraFolha) {
        if (this.tamanhoArvore < nivel) {
            this.tamanhoArvore = nivel;
        }
    }
    if (p.esq == null && p.dir == null) {
        if (this.primeiraFolha) {
            this.primeiraFolha = false;
        }
    }
    if (p.incE == Horizontal) {
        this.calculaTamanho(p.esq, nivel);
    } else {
        this.calculaTamanho(p.esq, nivel + 1);
    }
    if (p.incD == Horizontal) {
        this.calculaTamanho(p.dir, nivel);
    } else {
        this.calculaTamanho(p.dir, nivel + 1);
    }
}

public void retornaQuantidadeNiveis() {
    this.tamanhoArvore = 0;
    this.primeiraFolha = true;
    this.calculaTamanho(this.raiz, 1);
}
```

7. Explique o objetivo de cada um dos métodos implementados nos itens 3, 5 e 6 por meio de comentários no próprio código.

Exercício 3: Experimentos e gráficos

O objetivo deste exercício é testar a eficácia do método de pesquisa da classe **ArvoreSBB** em diferentes tamanhos de árvores.

1. Experimento 1:
   (a) Crie 10 árvores binárias 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 nós visitados.
   (c) Utilize o método **retornaQuantidadeNiveis** para calcular quantos níveis cada árvore gerada no exercício (a) possui.
2. Experimento 2:
   (a) Conceba 10 árvores binárias diferentes contendo N elementos **ALEATÓRIOS**, entre 1 e 500.000, 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 nós visitados.
   (c) Calcule a quantidade de níveis em cada árvore criada no exercício (a).
3. Experimento 3:
   (a) Gere 10 árvores binárias diferentes contendo N elementos **ALEATÓRIOS** entre 1 e 500.000, no qual N contém os seguintes valores: $\{5, 10, 50, 100, 500, 1.000, 5.000, 10.000, 50.000, 100.000\}$. Durante a construção da árvore, utilize a classe **Random** do pacote java.util para gerar os valores randômicos.
   (b) Compute quantos níveis tem em cada árvore concebida no exercício (a).
4. Faça três gráficos:
   - Um que mostre, para cada valor de N, o número de comparações feitas durante as buscas dos experimentos 1 e 2 desta lista. Neste gráfico, o eixo X refere-se ao número de elementos de cada árvore (N) enquanto o numero de comparações é retratado pelo eixo Y. Em seguida, indique porque os números de comparações são similares em ambos os casos.
   - Um que exponha, para cada valor de N, o número de comparações realizadas durante as pesquisas dos experimentos 1 das listas 1 e 2. O eixo X deste gráfico contem o número de elementos de cada árvore (N) e o eixo Y é composto pela quantidade de comparações. Depois, explique porque os números de comparações são consideravelmente maiores para um dos dois casos.
   - Um que relaciona cada valor de N com o tamanho de sua respectiva Árvore SBB obtido no experimento 3 desta lista. Neste gráfico, o eixo X refere-se ao número de elementos de cada árvore (N) enquanto o eixo Y descreve o tamanho da árvore. Em seguida, descreva a relação entre o tamanho da árvore e a complexidade do algoritmo de pesquisa em uma árvore SBB.