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

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

Lista de Exercícios 1

Professores: Amadeu Almeida e Thiago Rodrigues **Data de entrega:** 23 de Agosto de 2019

Instruções: Leia com atenção todas as orientações abaixo antes de começar a trabalhar nesta lista.

- Esta lista de exercícios é individual e vale 5 pontos.
- Submeta no Moodle o arquivo **Lista1_SeuNome.zip** (Exemplo: Lista1_Amadeu.zip) que contenha: as classes JAVA, os gráficos do experimento e suas respectivas explicações. Os gráficos e as explicações devem estar respectivamente nos formatos **PDF** e **txt**.
- O prazo de entrega é até às 14:40 do dia 23 de Agosto.
- Códigos fonte em PDF e submissões em atraso serão desconsiderados.
- O enunciado desta lista possui 3 exercícios e 2 páginas.

Exercício 1: Classe Item

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

1. Faça um projeto chamado **Lista1** no NetBeans. 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 Árvore Binária

Este exercício visa implementar uma classe que representa uma árvore binária de pesquisa não balanceada, aplicando os conceitos aprendidos durante a disciplina teórica.

1. No projeto **Lista 1**, crie um arquivo chamado ArvoreBinaria.java.
2. Implemente a classe **No** conforme especificada abaixo. Esta classe deve ser implementada dentro da classe **ArvoreBinaria** de modo que cada nó da árvore seja um objeto da classe **No**.

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

3. Implemente os atributos da classe **ArvoreBinaria**:
   - private No raiz
   - private long comparacoes
4. Implemente os atributos da classe **ArvoreBinaria**:
   - public ArvoreBinaria(): inicializa o nó raiz.
   - private No insere(Item reg, No p): insere o elemento reg, passado como parâmetro, na árvore.
   - private Item pesquisa(Item reg, No p): busca o elemento reg, passado como parâmetro, na árvore.
   - 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 pesquisa de uma chave à partir do nó raiz.
   - public long getComparacoes(): retorna o número de comparações realizadas durante uma pequisa na árvore.
5. Explique o objetivo de cada um dos métodos acima 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 **ArvoreBinaria** em diferentes tipos de árvores.

1. Experimento 1:
   (a) Gere 10 árvores binárias diferentes contendo N elementos **ORDENADOS** (ou seja, inseridos em ordem crescente), na qual N varia em intervalos de 1.000, entre os números 1.000 e 9.000.
   (b) Para cada uma das árvores geradas no exercício (a), pesquise por um elemento não pertencente à árvore e verifique o número de comparações realizadas (utilizando o atributo comparações) e o tempo gasto (usando a função System.nanoTime()) durante a pesquisa.
2. Experimento 2:
   (a) Gere 10 árvores binárias diferentes contendo N elementos **ALEATÓRIOS**, na qual N varia em intervalos de 1.000, entre os números 1.000 e 9.000. Durante a construção da árvore, utilize a classe **Random** do pacote java.util para gerar os N elementos aleatórios da árvore.
   (b) Para cada uma das árvores geradas no exercício (a), pesquise por um elemento não pertencente à árvore e verifique o número de comparações realizadas (utilizando o atributo comparações) e o tempo gasto (usando a função System.nanoTime()) durante a pesquisa.
3. Faça dois gráficos:
   - Um que exponha, para cada valor de N, o número de comparações realizadas durante as pesquisas dos experimentos 1 e 2. O eixo X deste gráfico contem o número de elementos de cada árvore (N) e o eixo Y é composto pelo número de comparações. Em seguida, explique porque o número de comparações é consideravelmente maior para um dos dois casos.
   - Um que mostre, para cada valor de N, o tempo gasto durante cada pesquisa feita nos experimentos 1 e 2. Neste gráfico, o eixo X refere-se ao número de elementos de cada árvore (N) e o tempo gasto é representado pelo eixo Y. Em seguida, explique porque o algoritmo de pesquisa demora mais tempo em um dos dois casos.