Centro Federal de Educação Tecnológica de Minas Gerais
Departamento de Computação
Laboratório de Algoritmos e Estruturas de Dados II
Prof. Thiago de Souza Rodrigues
2018-03-16

Prática 02 – Implementação em JAVA do TAD *Symetric Binary B-Tree* (SBB)

- **Data de Entrega: 30/03/2018**
- **Deve ser entregue um relatório no moodle contendo:**
  - **Nome, Data de entrega, Número da Prática e Título da Prática;**
  - **Gráfico dos tópicos C e D do item 4;**
  - **Item 5;**
- **Deve ser entregue, via moodle, projeto contendo o código fonte comentado;**
- **Pode ser entregue em grupos de até 03 peassoas;**

1) Utilizando o Netbeans, crie um projeto chamado *Prática02*;

2) Implemente a classe ***Item***, como especificada abaixo para ser utilizada no T.A.D.;

```
package Item;
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;
    }
}
```

3) Implemente uma classe chamada ***ArvoreSBB*** para manipular uma árvore binária de pesquisa onde os nós da árvore são objetos da classe ***No***, especificada abaixo;

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

A classe ArvoreBinaria deve conter os seguintes métodos:

- **`public ArvoreSBB():`** para inicializar o nó raiz;
- **`public void insere(Item reg):`** para inserir o elemento ***`reg`*** passado por parâmetro;
- **`public Item pesquisa(Item reg):`** para realizar a busca do elemento ***`reg`*** passado por parâmetro;
- **`public void pre_order():`** para imprimir os nós da árvore utilizando o percurso Pre-Order;

Cada método deve estar comentado;

Obs.: os métodos auxiliares para realizar as movimentações necessárias durante a inserção devem ser implementadas;

```
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) Realizar os seguintes experimentos:

a) gerar árvores a partir de **n** elementos ***ORDENADOS***, com **n** variando de 10.000 até 100.000, com intervalo de 10.000.

   Em cada árvore gerada pesquisar por um elemento ***não existente*** e verificar o número de comparações realizadas e o tempo gasto (dica: função **`System.nanoTime()`**) na pesquisa em cada árvore;

b) gerar árvores a partir de **n** elementos ***ALEATÓRIOS*** (**`long j = obj.nextInt()`**), com **n** variando de 10.000 até 100.000, com intervalo de 10.000.

   Em cada árvore gerada pesquisar por um elemento ***não existente*** e verificar o número de comparações realizadas e o tempo gasto na pesquisa em cada árvore;

c) Fazer um único gráfico de **`n x número de comparações`** levando em consideração as árvores geradas com inserções ordenadas e aleatórias;

d) Fazer um único gráfico de **`n x tempo gasto`** levando em consideração as árvores geradas com inserções ordenadas e aleatórias;

5) Explique o comportamento dos gráficos gerados;

6) Compare com o resultados obtidos na prática 01;