INSTITUTO FEDERAL Catarinense Campus Blumenau

CCA0916 - Estruturas de dados I - BCC

Bacharelado em Ciência da Computação

Prof. Dr. Paulo César Rodacki Gomes

Lista de exercícios - 09 - Árvores Binárias de Busca

1 Objetivo

O objetivo desta atividade prática em laboratório é implementar a estrutura de árvore binária de busca e suas operações conforme visto em sala de aula (os slides estão disponíveis no Google Classroom. Você pode implementar o exercício em uma linguagem orientada a objetos ou em ANSI C. Para esta atividade vamos ordenar arrays com valores do tipo int.

Observação 1: as descrições das assinaturas dos métodos estão em sintaxe Java. Você deve adaptar para a linguagem orientada a objetos de sua escolha.

Observação 2: se você optar por implementar o exercício em ANSI C, a descrição está no final deste documento.

2 Instruções – implementação em linguagem orientada a objetos

O objetivo desta atividade prática em laboratório é implementar uma estrutura de árvore binária de busca para armazenar e buscar números inteiros de acordo com o diagrama de classes da figura 1.

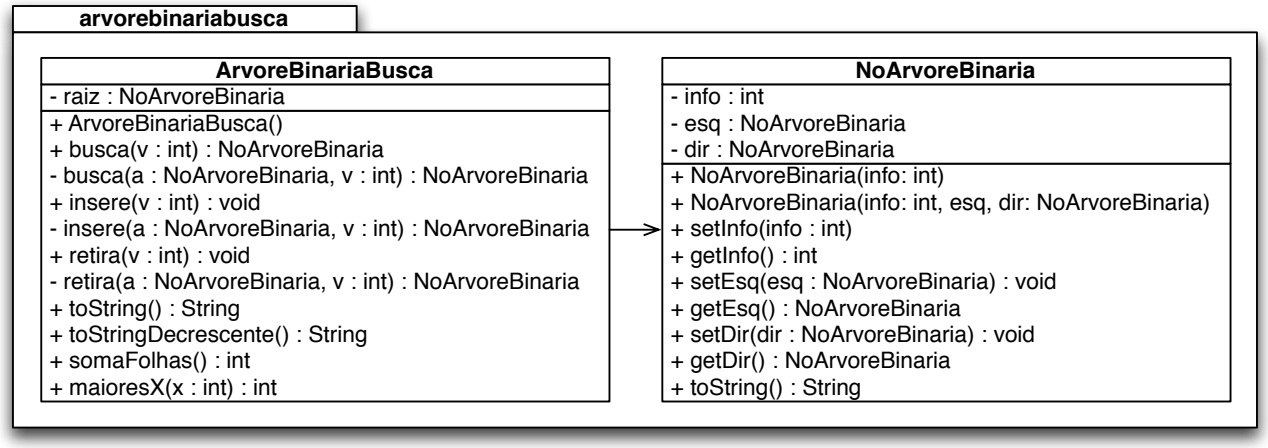

Figura 1: Diagrama de classes do pacote arvorebinariabusca

2.1 Descrição dos métodos a serem implementados

1. `public ArvoreBinariaBusca()`: construtor da classe ArvoreBinariaBusca, deve criar uma árvore vazia atribuindo `null` para a raiz.

2. `public NoArvoreBinaria busca(int v)`: método para buscar um elemento na árvore, deve retornar o nó da árvore que contÈm o elemento de valor v.

3. `private NoArvoreBinaria busca(NoArvoreBinaria a, int v)`: método privado recursivo para buscar um elemento na árvore, deve retornar o nó da árvore que contém o elemento de valor v.

4. `public void insere(int v)`: método para inserir um elemento na árvore, respeitando a ordenação de seus elementos.

5. `private NoArvoreBinaria insere(NoArvoreBinaria a, int v)`: método privado recursivo para inserir um elemento na árvore, respeitando a ordenação de seus elementos, deve retornar o endereço do eventual novo nó raiz da (sub)-árvore.

6. `public void retira(int v)`: método para retirar um elemento com valor `v` da árvore.

7. `private NoArvoreBinaria retira(NoArvoreBinaria a, int v)`: método privado recursivo para retirar um elemento com valor `v` da árvore, deve retornar o endereÁo do eventual novo nó raiz da (sub)-árvore.

8. `public String toString()`:imprime o conteúdo de toda a árvore, em ordem crescente. Observação: se você estiver implementando em Python, este deve ser o método `__str__`.

9. Implemente também uma classe contendo método `main` para testar sua implementação de árvore Binária de Busca.

3 Instruções – implementação em ANSI C

Considere uma árvore binária de busca que armazena valores inteiros. Neste tipo de árvore, os valores associados aos nós da sub-árvore à esquerda são menores que o valor associado à raiz e que os valores dos nós da sub-árvore à direita são iguais ou maiores.

1. Inicialmente crie os seguintes arquivos fonte:

   - `principal.c`: função main com demonstração do funcionamento da árvore.
   - `arvore_busca_binaria.h`: declarações de estruturas, tipos de dados e protótipos das funções.
   - `arvore_busca_binaria.c`: implementação das funções da árvore.

2. Estruturas a serem criadas no arquivo `arvore_busca_binaria.h`:

```
struct arvbb {
  int info;
  struct arvbb* esq;
  struct arvbb* dir;
};
typedef struct arvbb Arvbb;
```

3. Funções a serem implementadas no arquivo `arvore_busca_binaria.c`:

   (a) `Arvbb* arvbb_cria (void)`: função de inicialização da árvore. Esta função deve criar uma árvore vazia atribuindo `NULL` para o ponteiro para a árvore.

   (b) `void arvbb_imprime (Arvbb *a)`: função para imprimir os elementos da árvore em ordem crescente;

(c) `Arvbb* arvbb_busca (Arvbb *r, int v)`: função para buscar um elemento na árvore, de acordo com o protótipo definido a seguir. A função deve retornar o endereço do nó da árvore que contem o elemento de valor v;

(d) `Arvbb* arvbb_insere (Arvbb* a, int v)`:função para inserir um valor na árvore, respeitando a ordenação de seus elementos. A função deve retornar o endereço do eventual novo nó raiz da árvore (ou sub-árvore);

(e) `Arvbb* arvbb_retira (Arvbb* a, int v)`:implemente a operação que retire um elemento da árvore. A função deve retornar o endereço do eventual novo nó raiz da (sub)-árvore;

(f) `void arvbb_imprime_decrescente (Arvbb *a)`: função para imprimir os elementos da árvore em ordem decrescente.

Observação: após implementar a árvore, implemente um programa principal para testar e demonstrar o funcionamento do código implementada.