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 - 07 - Árvores N-árias

1 Objetivo

O objetivo desta atividade prática em laboratório é implementar árvores N-árias (árvores com número variável de sub-árvores) conforme visto em sala de aula (os slides estão disponíveis no Goggle Classroom.

2 Implementação orientada a objetos

Siga as orientações desta seção se você vai implementar sua árvore N-ária em uma linguagem orientada a objetos. O diagrama de classes da figura 1 mostra a implementação sugerida. Além destas classes, você também deve implementar um programa principal para testar e demonstrar sua implementação.

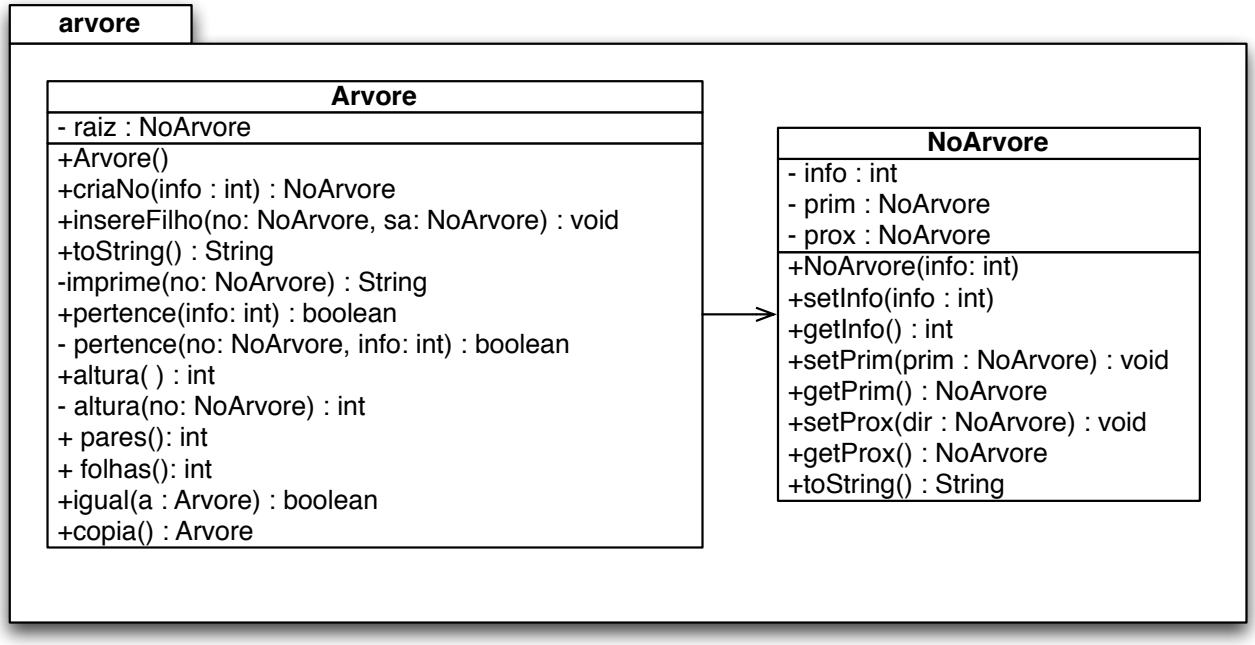

Figura 1: Diagrama de classes do pacote arvore

2.1 Observações importantes

A classe `NoArvoreBinaria` deve ter os seguintes atributos privados:

- `int info`: é o dado armazenado no nó. Para este exercício, vamos considerar que cada nó da árvore armazena um valor do tipo primitivo `int`;

- `NoArvoreBinaria prim`: é uma referência para o nó raiz da primeira sub-árvore filha do nó corrente e
- `prox`: referência para o nó raíz da sub-árvore "irmã".

A classe `Arvore` deve ter somente o atributo privado `raiz`, que armazena uma referência ao nó raiz principal da árvore.

2.2 Descrição dos métodos a serem implementados

1. `public Arvore()`: construtor da classe Arvore (que cria uma árvore vazia). **Observação:** nosso modelo teórico não prevê árvores vazias, mas por hora vamos deixar este método assim, e vamos considerar que só vamos chamar os outros métodos da árvore após termos criado pelo menos o nó raiz.
2. `criaNo`: método para inserir um novo nó folha na árvore.
3. `insereFilho`: método para inserir um uma sub-árvore como filha de um nó previamente criado.
4. `toString`: chama os métodos privado para impressão textual da árvore.
5. `imprime`: método privado (recursivo) para imprimir árvores.
6. `pertence`: métodos publico e privado para verificar se determinada informação está armazenada na árvore.
7. `altura`: métodos publico e privado para calcular e retornar a altura da árvore.

   Os métodos a seguir podem necessitar de implementação de método privados auxiliares não especificados na figura 1
8. `pares`: retorna a quantidade de números pares armazenados na árvore.
9. `folhas`: retorna a quantidade de nós do tipo folha na árvore.
10. `igual`: verifica se a árvore atual e a árvore `a` são iguais. Se voce já conhece melhor a linguagem Java, pode implementar o método `equals` ao invés do `igual`.
11. `copia`: cria e retorna uma cópia da árvore (novos objetos precisam ser instanciados).

3 Implementação em ANSI C

1. Inicialmente crie os seguintes arquivos fonte:
   - `principal.c`: função main com demonstração do funcionamento da árvore
   - `arvv.h`: declarações de estruturas, tipo e funções da árvore
   - `arvv.c`: implementação das funções da árvore
2. Estruturas a serem criadas no arquivo `arvv.h`:

```
/* tipo abstrato de dados Arvv */
struct arvv {
int info;
struct arvv *prim;
struct arvv *prox;
```

```
};
typedef struct arvv Arvv;

/* protótipos das funcoes para operacoes nas arvores */
Arvv* arvv_cria(int v);
Void  arvv_insere(Arvv*a, Arvv*sa);
Arvv* arvv_libera(Arvv* a);
int   arvv_pertence(Arvv* a, int n);
void  arvv_imprime(Arvv* a);
int   arvv_altura(Arvv* a);
int   arvv_pares(Arvv* a);
int   arvv_folhas(Arvv* a);
int   arvv_ivual(Arvv* a1, Arvv* a2);
Arvv* arvv_copia(Arvv* a);
```

3. As unções a serem criadas no arquivo `arvv.c`estão listadas acima e seguem descrição e funcionamento análogos à versão orientada a objetos deste exercício.