

# Laboratório de Programação II

## Árvore Binária de Busca

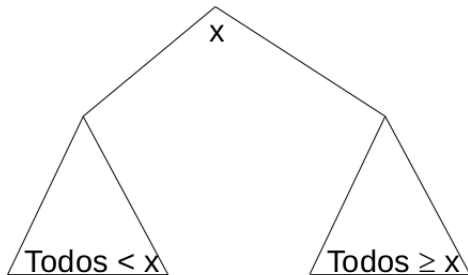
Universidade Federal de Juiz de Fora  
Departamento de Ciência da Computação

# Aula de Hoje

- ▶ TAD Árvore Binária de Busca (ABB)
  - ▶ Revisão ABB
  - ▶ Exercícios

# Árvore Binária de Busca

- ▶ Definição: uma árvore binária de busca (ABB) é uma árvore binária na qual cada nó possui uma chave comparável e que satisfaz a seguinte restrição: a chave em qualquer nó é
  - ▶ maior do que as chaves de todos os nós da sub-árvore à esquerda; e
  - ▶ menor (ou igual) às chaves de todos os nós da sub-árvore à direita.



## TAD Nó (NoArv)

- ▶ TAD Nó para árvore binária de busca.
- ▶ Idêntico ao nó para árvore binária (TAD ArvBin).

```
class NoArv
{
    public:
        NoArv()                { };
        ~NoArv()               { };
        void setEsq(NoArv *p)  { esq = p; };
        void setInfo(int val)  { info = val; };
        void setDir(NoArv *p)  { dir = p; };
        NoArv* getEsq()        { return esq; };
        int  getInfo()          { return info; };
        NoArv* getDir()        { return dir; };

    private:
        NoArv *esq; // ponteiro para o filho a esquerda
        int  info;  // informacao do No (int)
        NoArv *dir; // ponteiro para o filho a direita
};
```

# TAD ArvBinBusca

```
class ArvBinBusca
{
public:
    ArvBin();
    ~ArvBin();
    bool vazia();
    bool busca(int val);
    void imprime();
    void remove(int val);

private:
    NoArv* raiz;
    bool auxBusca(NoArv *p, int val);
    NoArv* auxInsere(NoArv *p, int val);
    NoArv* auxRemove(NoArv *p, int val);
    NoArv* libera(NoArv *p);
    // etc ...
};
```

## TAD ArvBinBusca

- ▶ Lembre-se, para algumas operações que utilizam um algoritmo recursivo para realizar alguma tarefa é preciso criar uma função auxiliar que recebe como parâmetro um ponteiro para um nó.
- ▶ Na primeira chamada a função passa a raiz.
- ▶ Em seguida a função auxiliar trabalha de forma recursiva para implementar o algoritmo desejado.

```
class ArvBinBusca
{
    public:
        // ....
        void insere(int val);

    private:
        // ...
        void auxInsere (NoArv *p, int val);
};
```

# Exercícios

1. Fazer uma operação para encontrar, e retornar, o **maior elemento** de uma árvore binária de busca.

```
int ArvBinBusca::maior();
```

2. Fazer uma operação para encontrar, e retornar, o **menor elemento** de uma árvore binária de busca.

```
int ArvBinBusca::menor();
```

3. Fazer uma operação para **remover o maior elemento** de uma árvore binária de busca.

```
void ArvBinBusca::removeMaior();
```

4. Fazer uma operação para **remover o menor elemento** de uma árvore binária de busca.

```
void ArvBinBusca::removeMenor();
```

## Exercícios

5. Desenvolver uma operação para contar **quantos nós são pares** no caminho que vai da raiz até o nó de valor  $x$ .

```
int ArvBinBusca::contaParesCaminho(int x);
```

6. **Alterar a função de remoção** de nó com dois filhos considerando, agora, **o maior elemento da sub-árvore à esquerda** como o elemento a ser trocado com o nó a ser removido.

```
NoArv* ArvBinBusca::maiorSubArvEsq(NoArv *p);
```

7. Desenvolver uma função recursiva para ser usado no programa principal para **preencher uma árvore binária de busca**, isto é, de tal forma que ela fique completa. Os valores dos nós devem ser inteiros no intervalo de  $p$  a  $q$ . Os parâmetros do procedimento serão uma árvore vazia e os valores inteiros  $p$  e  $q$ .

```
void preencheABB(ArvBinBusca *a, int p, int q);
```