# ESTRUTURAS DE DADOS

## Árvores (Implementação)

# Roteiro

- **Aplicações da Estrutura**
- **Estrutura do Nó**
- **Tipo Abstrato de Dados**
- **Detalhes de Implementação**

# Aplicações da Estrutura

**Árvores binárias de busca são estruturas fundamentais usadas para construir outras estruturas.**

**Em geral, podem ser usadas em qualquer situação em que queremos organizar os dados por meio de uma chave usada nas buscas.**

- **Quando inserções e remoções são frequentes, são melhores que arranjos ordenados.**

**Vamos supor  que queremos organizar alunos em uma estrutura e, posteriormente, fazer buscas pelo registro acadêmico (ra).**

- **A única informação que queremos é o nome.**

```
class Aluno{
private :
  int         ra;
  std::string nome;
public:
  Aluno();
  Aluno(int ra, std::string nome);
  string getNome() const;
  int getRa() const;
};
```

```
Aluno::Aluno(){
  this->ra   = -1;
  this->nome = "";
};
Aluno::Aluno(int ra, std::string nome){
  this->ra    = ra;
  this->nome  = nome;
}
string Aluno::getNome() const {
  return nome;
}
int Aluno::getRa() const{
  return ra;
}
```

# Estrutura do Nó

**Contém os dados e os endereços das subárvores.**

```
/*
  Estrutura usada para
  guardar a informação
  e os endereços das
  subárvores
 */
struct NodeType
{
  Aluno aluno;
  NodeType* esquerda;
  NodeType* direita;
};
```

# Tipo Abstrato de Dados

```
class SearchTree
{
 public:
  SearchTree() { root = NULL;  }
  ~SearchTree(){ destroyTree(root); }
  bool isEmpty() const;
  bool isFull() const;
  void retrieveAluno(Aluno& item, bool& found) const{
    retrieveAluno(root, item, found);
  }
  void insertAluno(Aluno item){ insertAluno(root, item); }
  void deleteAluno(int item){ deleteAluno(root, item); }
  void printPreOrder()  const { printPreOrder(root); }
  void printInOrder()   const { printInOrder(root);  }
  void printPostOrder() const { printPostOrder(root);}
```

**Interface pública invocando métodos recursivos privados.**

```
 private:
  void destroyTree(NodeType*& tree);
  void retrieveAluno(NodeType* tree,
                     Aluno& item,
                     bool& found) const;
  void insertAluno(NodeType*& tree, Aluno item);
  void deleteAluno(NodeType*& tree, int item);
  void deleteNode(NodeType*& tree);
  void getSuccessor(NodeType* tree, Aluno& data);
  void printTree(NodeType *tree) const;
  void printPreOrder(NodeType* tree)  const;
  void printInOrder(NodeType* tree)   const;
  void printPostOrder(NodeType* tree) const;
  //  Raiz da árvore binária de busca.
  NodeType* root;
};
```

# Detalhes de Implementação

**O método destroyTree efetua um caminhamento pós-ordem:**

```
void SearchTree::destroyTree(NodeType*& tree)
{
  if (tree != NULL)
    {
      destroyTree(tree->esquerda);
      destroyTree(tree->direita);
      delete tree;
    }
}
```

# Verificação de cheio ou vazio.

```
bool SearchTree::isEmpty() const {
  return root == NULL;
}
bool SearchTree::isFull() const {
  NodeType* location;
  try
    {
      location = new NodeType;
      delete location;
      return false;
    }
  catch(std::bad_alloc exception)
    {
      return true;
    }
}
```

# Implementando as ideias da aula passada.

- **Buscando aluno**

```
void SearchTree::retrieveAluno(NodeType* tree,
                               Aluno& aluno,
                               bool& found) const {
  if (tree == NULL)
    found = false;
  else if (aluno.getRa() < tree->aluno.getRa())
    retrieveAluno(tree->esquerda, aluno, found);
  else if (aluno.getRa() > tree->aluno.getRa())
    retrieveAluno(tree->direita, aluno, found);
  else {
    aluno = tree->aluno;
    found = true;
  }
}
```

# Implementando as ideias da aula passada.

- **Inserindo aluno**

```
void SearchTree::insertAluno(NodeType*& tree, Aluno aluno)
{
  if (tree == NULL)
    {
      tree = new NodeType;
      tree->direita  = NULL;
      tree->esquerda = NULL;
      tree->aluno    = aluno;
    }
  else if (aluno.getRa() < tree->aluno.getRa() )
    insertAluno(tree->esquerda, aluno);
  else
    insertAluno(tree->direita, aluno);
}
```

# Implementando as ideias da aula passada.

- **Imprimindo a lista na saída padrão**

```
void SearchTree::printPreOrder(NodeType* tree) const {
  if (tree != NULL) {
      std::cout << tree->aluno.getNome() << " , ";
      printPreOrder(tree->esquerda);
      printPreOrder(tree->direita);
    }
}
```

```
void SearchTree::printInOrder(NodeType* tree) const {
  if (tree != NULL) {
        printInOrder(tree->esquerda);
        std::cout << tree->aluno.getNome() << " , ";
        printInOrder(tree->direita);
    }
}
void SearchTree::printPostOrder(NodeType* tree) const {
  if (tree != NULL) {
        printPostOrder(tree->esquerda);
        printPostOrder(tree->direita);
        std::cout << tree->aluno.getNome() << " , ";
    }
}
```

# ESTRUTURAS DE DADOS

## Árvores (Implementação)