Testes e Classe ArvoreBinaria

O desenvolvimento da biblioteca de árvores binárias foi todo orientado a testes usando o framework
Catch2, sendo este dependência obrigatória para compilar o módulo de testes.
O módulo de testes é representado pelo arquivo **testa_arvore.cpp**

Todos os métodos e dados relativos à manipulação das árvores binárias estão contidos na classe **ArvoreBinaria**
definida principalmente no arquivo **arvores.hpp**.

O usuário da biblioteca não tem acesso direto aos ponteiros que compoem a árvore, mas pode ler, inserir, remover
e modificar valores das células presentes na árvore através dos métodos públicos.

As funções públicas da classe **ArvoreBinaria** retornam um numero inteiro que representa um código de erro
utilizado para sinalizar caso algumas das rotinas não seja executada corretamente dentro das funções.

A definição dos sinais de erro está no início do arquivo **arvore.hpp**:

```
enum CodigosErro {EXITO = 2, FALHA}; //Definindo os codigos d
e erro para as funcoes
```

Para evitar a repeticao de **REQUIRE** ao longo do código, foi usada uma função que verifica os retornos dos métodos
usando os sinais **EXITO** e **FALHA**:

```
 //Definindo funcoes de teste de falhas/exitos para evitar re
peticoes pelo codigo
 void inline TesteExito(int codigo_erro) {
   REQUIRE(codigo_erro == arvores::EXITO);
 }

 void inline TesteFalha(int codigo_erro) {
   REQUIRE(codigo_erro == arvores::FALHA);
 }
```

Listagem dos Membros da Classe ArvoreBinaria

Métodos Públicos:

```
 ArvoreBinaria(void);
 bool EstaVazia(void);
 int InserirCelula(const uint nivel, const uint posicao, cons
t T);
 int LerCelula(const uint nivel, const uint posicao, T& saida
_dado);
 int MudarValorCelula(const uint nivel, const uint posicao, c
```

```
onst T);
 int LerDoArquivo(const std::string diretorio_arquivo);
 int SalvarNoArquivo(const std::string diretorio_arquivo);
 int ApagarCelula(const uint nivel, const uint posicao);
 bool TemFilho(const uint nivel, const uint posicao);
```

Tipos de Dados Privados:

```
 struct CelulaArvore;
 typedef std::shared_ptr<CelulaArvore> PtrCelulaArvore;
 PtrCelulaArvore raiz;
```

Métodos Privados:

```
 bool CelulaNula(PtrCelulaArvore celula_arvore);
 void ErroInsercaoCelulaInvalida(const uint nivel, const uint
 posicao);
 void ErroInsercaoCelulaExiste(const uint nivel, const uint p
osicao);
 void ErroBuscaCelulaNaoExiste(const uint nivel, const uint p
osicao);
 void ErroNaoPodeAbrirArquivo(const std::string diretorio_arq
uivo);
 void ErroArquivoInvalido(const std::string diretorio_arquivo
);
 void ErroTipoLeituraDeArquivo(void);
```

```
Uma descrição mais detalhada das subrotinas está presente em
forma de comentários ao longo do código.
```

Descrição dos Testes:

Teste de Inicializacao

O primeiro teste presente no arquivo **testa_arvore.cpp** é um teste simples para verificar
se a classe **ArvoreBinaria** foi instanciada e inicializada corretamente.

Para isso, são declaradas duas árvores, uma de inteiros e outra de strings e em seguida é verificado
se ambas as árvores estão vazias utilizando o método **EstaVazia**, este método retorna *true* se
a raíz da árvore for um *nullptr* indicando que a árvore ainda não foi inicializada.

Função EstaVazia:

```
bool EstaVazia(void);
```

Casos de Teste:

```
TEST_CASE("Testando inicializadores", "[arvore]") {
 arvores::ArvoreBinaria<int> arvore_int;
```

```
 arvores::ArvoreBinaria<std::string> arvore_string;
 REQUIRE(arvore_int.EstaVazia());
 REQUIRE(arvore_int.EstaVazia());
}
```

Teste de Conversão de Decimal/Binario

Este teste verifica se a função **DecimalParaBinario** membra do namespace **arvores**, faz a conversão das entradas corretamente.

Este método deve estar funcionando antes da implementação dos métodos da classe já que para inserir,
remover e modificar as células da árvore é utilizado um sistema de coordenadas **nível/posição** que mapeia o caminho
a ser percorrido na árvore através da representação binária dos números que representam a **posição**.

A função recebe 2 parâmetros, sendo o primeiro o número em base 10 a ser convertido e o segundo o número
de bits que serão usados para representá-lo.

Função DecimalParaBinario:

```
std::string DecimalParaBinario(const int decimal, const int n
umero_de_bits);
```

Casos de Teste:

```
TEST_CASE("Testando funcao de conversao de decimal para binar
io", "[arvore]") {
 REQUIRE(arvores::DecimalParaBinario(100, 7) == "1100100");
 REQUIRE(arvores::DecimalParaBinario(128, 8) == "10000000");
 REQUIRE(arvores::DecimalParaBinario(127, 10) == "0001111111"
);
 }
```

Teste de Inserção das Células na Árvore

Este teste procura verificar se as células estão sendo inseridas corretamente na árvore binária,
para fazer essa inserção, será usado o método **InserirCelula**.

O método **InserirCelula** recebe como parametro as coordenadas que representam o local na árvore
em que o dado deve ser inserido e uma cópia desse dado.

Função InserirCelula:

```
int InserirCelula(const uint nivel, const uint posicao, const
 T);
```

Casos de Teste:

```
TEST_CASE("Testando insercao de celulas nas arvores", "[arvore]") {
arvores::ArvoreBinaria<double> arvore_double;

//Testando insercoes na origem e em posicoes invalidas
TesteExito(arvore_double.InserirCelula(0, 0, 1.5));

TesteFalha(arvore_double.InserirCelula(0, 0, 3.10)); // Tentando inserir uma celula em lugar ja ocupado
TesteFalha(arvore_double.InserirCelula(2, 100, 0.500001)); // Tentando inserir uma celula em posicao invalida (2,100)
TesteFalha(arvore_double.InserirCelula(3, 8, 1.5678987)); // Tentando inserir umar celular em posicao invalida (3,8)
TesteFalha(arvore_double.InserirCelula(10, 2, 10.8)); // Tentando inserir um no em um nivel ainda nao preenchido

TesteExito(arvore_double.InserirCelula(1, 0, 0.000001));
TesteExito(arvore_double.InserirCelula(1,1,1));

arvores::ArvoreBinaria<uint> arvore_uint;
TesteFalha(arvore_uint.InserirCelula(1, 0, 100)); // Tentando inserir um no antes de incializar a raiz da arvore
TesteExito(arvore_uint.InserirCelula(0, 0, 0));
TesteFalha(arvore_uint.InserirCelula(10, 2, 10)); // Tentando inserir um no em um nivel ainda nao preenchido
```

```
TesteExito(arvore_uint.InserirCelula(1, 0, 3));
TesteExito(arvore_uint.InserirCelula(1,1,1));
}
```

Testes de Busca e Leitura das Células da Árvore

Estes testes verificam se a leitura das células inseridas está de acordo com o esperado, isto é, se o método
LerCelula consegue recuperar os valores inseridos na árvore binária por **InserirCelula**.

A função **LerCelula** recebe como parâmetros a **posição** e **nível** da célula que se deseja ler e a
variável **saida_dado** em que se deseja salvar os valores obtidos. O parâmetro **saida_dado** é passado
como referência para que seja possível modificá-lo dentro da função de leitura.

Função LerCelula:

```
 int LerCelula(const uint nivel, const uint posicao, T& saida_dado);
```

Casos de Teste:

```
TEST_CASE("Testando busca de celulas pela arvore", "[arvore]"
) {
arvores::ArvoreBinaria<float> arvore_float;
float saida_arvore;
float saida_esperada;

//Inserindo celulas para serem testadas
arvore_float.InserirCelula(0, 0, 0.1);
arvore_float.InserirCelula(1, 1, 0.001);
arvore_float.InserirCelula(2, 2, 100);

//Verificando se a celula 0,0 foi inserida corretamente e o v
alor retornado corresponde ao valor esperado (0.1)
saida_esperada = 0.1;
TesteExito(arvore_float.LerCelula(0, 0, saida_arvore));
REQUIRE(saida_arvore == saida_esperada);

//Verificando se a celula 1,1 foi inserida corretamente
saida_esperada = 0.001;
TesteExito(arvore_float.LerCelula(1, 1, saida_arvore));
REQUIRE(saida_arvore == saida_esperada);

saida_esperada = 100;
TesteExito(arvore_float.LerCelula(2, 2, saida_arvore));
REQUIRE(saida_arvore == saida_esperada);

//Verificando a leitura de celulas nao existentes
```

```
TesteFalha(arvore_float.LerCelula(10, 10, saida_arvore));
TesteFalha(arvore_float.LerCelula(0, 10, saida_arvore));
}
```

Testes de Mudança de Dados das Células

Este grupo de teste visa garantir que a função **MudarValorCelula** consegue alterar com sucesso o membro
dados presente nas células da árvore para um valor especificado pelo usuário.

Para fazer essa verificaçao são inseridas 3 células na árvore com o método **InserirCelula**, em seguida
a função **LerCelula** é chamada para garantir que os dados foram inseridos com sucesso. Logo após, são
feitas tentativas de modificação nas células com **MudarValorCelula** e os dados sao lidos novamente
para verificar se a mudança ocorreu.

O método **MudarValorCelula** recebe como parâmetro as coordenadas **nivel** e **posicao** e uma cópia do
novo dado que será inserido na célula especificada.

Função MudarValorCelula:

```
int MudarValorCelula(const uint nivel, const uint posicao, co
nst T);
```

Casos de Teste:

```
TEST_CASE("Testando busca de celulas pela arvore", "[arvore]"
) {
arvores::ArvoreBinaria<float> arvore_float;
float saida_arvore;
float saida_esperada;

//Inserindo celulas para serem testadas
arvore_float.InserirCelula(0, 0, 0.1);
arvore_float.InserirCelula(1, 1, 0.001);
arvore_float.InserirCelula(2, 2, 100);

//Verificando se a celula 0,0 foi inserida corretamente e o v
alor retornado corresponde ao valor esperado (0.1)
saida_esperada = 0.1;
TesteExito(arvore_float.LerCelula(0, 0, saida_arvore));
REQUIRE(saida_arvore == saida_esperada);

//Verificando se a celula 1,1 foi inserida corretamente
saida_esperada = 0.001;
TesteExito(arvore_float.LerCelula(1, 1, saida_arvore));
REQUIRE(saida_arvore == saida_esperada);
```

```
saida_esperada = 100;
TesteExito(arvore_float.LerCelula(2, 2, saida_arvore));
REQUIRE(saida_arvore == saida_esperada);

//Verificando a leitura de celulas nao existentes
TesteFalha(arvore_float.LerCelula(10, 10, saida_arvore));
TesteFalha(arvore_float.LerCelula(0, 10, saida_arvore));
}
```