Implementação de uma Árvore Binária de Pesquisa com Interface Gráfica em PyQt

Lucca Quintas Pedro H. Alcântara Marcos Tirelo

1 Introdução

As árvore binárias de pesquisa (BST) são estruturas de dados fundamentais para a computação. São utilizadas para organização e busca eficiente de dados. Neste artigo iremos explorar a implementação de uma UI (User Interface) que consiga simular uma BST e ao mesmo tempo aprender como podemos exibir visualmente a sua estrutura.

2 Implementação

O projeto consiste em três arquivos principais:

2.1 main.py

Este arquivo contém o código responsável pela inicialização da aplicação em PyQt, criação da janela e instanciamento da classe `BinaryTreeApp`.

```
# Codigo do main.py
import sys
from UI import QApplication, BinaryTreeApp

if __name__ == '__main__':
    app = QApplication(sys.argv)
    window = BinaryTreeApp()
    window.show()
    sys.exit(app.exec_())
```

2.2 UI.py

Aqui está a implementação da interface gráfica em PyQt, incluindo widgets para inserção, exclusão, listagem e importação de elementos na árvore binária de pesquisa. Também utiliza o matplotlib para exibir visualmente a árvore.

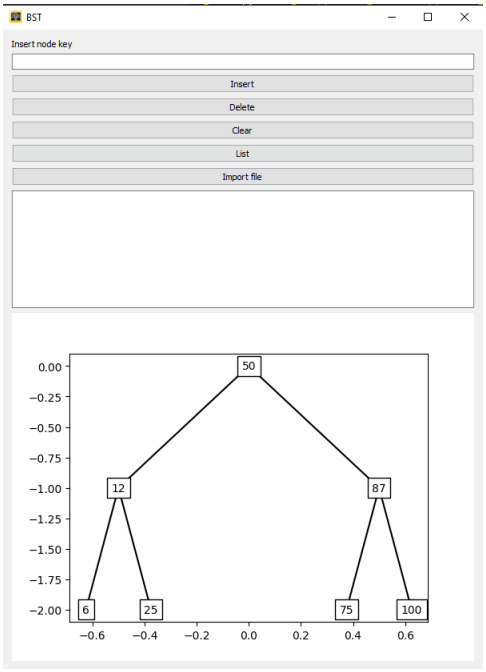

Figura 1: UI em funcionamento

2.2.1 Criação do layout com PyQt

Dentro da classe `BinaryTreeApp` iremos fazer as configurações padrões para a criação de uma janela utilizando `setWindowTitle()`, `setGeometry()` e `setWindowIcon(QIcon())` para dimensionar, estilizar e intitular a janela da sua UI da forma que preferir.

```
class BinaryTreeApp(QWidget):
    def __init__(self):
        super().__init__()

        self.tree = None
        self.initUI()

    def initUI(self):
        self.setWindowTitle('Arvore Binaria de Pesquisa')
        self.setGeometry(100, 100, 600, 800)

        icon_path = 'icon.png'
        self.setWindowIcon(QIcon(icon_path))
```

2.2.2 Botões e caixas de texto

É preciso criar um *input* de nós usando `QLineEdit()`. Feito isso iremos criar os nossos botões com `QPushButton()` e por fim iremos definir o nosso layout usando `QVBoxLayout` e `addWidget()` para adicioná-los ao layout

```
self.label = QLabel('Insira o valor do no:', self)
self.node_input = QLineEdit(self)

self.insert_button = QPushButton('Inserir', self)
self.delete_button = QPushButton('Excluir', self)
self.list_button = QPushButton('Listar', self)
self.import_button = QPushButton('Importar de arquivo', self)

self.text_output = QTextEdit(self)
self.text_output.setReadOnly(True)

self.layout = QVBoxLayout()
self.layout.addWidget(self.label)
self.layout.addWidget(self.node_input)
self.layout.addWidget(self.insert_button)
self.layout.addWidget(self.delete_button)
self.layout.addWidget(self.list_button)
self.layout.addWidget(self.import_button)
self.layout.addWidget(self.text_output)
self.setLayout(self.layout)



# Mapeando funcoes aos botoes
self.insert_button.clicked.connect(self.insert_node_func)
self.delete_button.clicked.connect(self.delete_node_func)
self.list_button.clicked.connect(self.list_tree)
self.import_button.clicked.connect(self.import_tree)
self.setWindowIcon(QIcon(icon_path))
```

2.2.3 Widget do `matplotlib`

Para finalizar sua janela tudo que falta é adicionar ao layout o widget do matplotlib para exibir o grafo.

```
# Adiciona um widget do matplotlib para exibir o grafo
self.figure = Figure()
self.canvas = FigureCanvas(self.figure)
self.layout.addWidget(self.canvas)

self.setLayout(self.layout)
```

2.2.4 Explorando as Funções da Interface

A seguir, apresentamos uma explicação breve de cada função implementada na interface gráfica da aplicação:

- **list_tree(self):**

```
def list_tree(self):
    if self.tree:
        self.text_output.clear()
        self.text_output.append('Em-ordem:')
        self.text_output.append(',-'.join(self.tree.print_in_order()))
        self.text_output.append('Pre-ordem:')
        self.text_output.append(',-'.join(self.tree.print_pre_order()))
        self.text_output.append('Pos-ordem:')
        self.text_output.append(',-'.join(self.tree.print_post_order()))
```

Esta função é responsável por listar os elementos da árvore binária de pesquisa (BST) em diferentes ordens, incluindo em ordem, pré-ordem e pós-ordem. Primeiro, verifica se a árvore não está vazia. Em seguida, limpa o campo de saída de texto e adiciona os elementos da árvore nas três ordens mencionadas, separados por vírgula. A saída é exibida na interface para visualização do usuário.

- **delete_node_func(self):**

```
def delete_node_func(self):
    if self.node_input.text().isdigit():
        key = int(self.node_input.text())
        if key:
            delete_node(root=self.tree, key=key)

    self.plot_tree()
    self.node_input.clear()
```

Esta função é acionada quando o usuário deseja excluir um nó da árvore. Verifica se o valor inserido é um número inteiro válido. Se for, converte para inteiro e chama a função de exclusão do nó correspondente na árvore. Após a exclusão, a função de atualização da visualização da árvore é chamada para refletir as alterações na interface.

- **insert_node_func(self):**

```
def insert_node_func(self):
    # Insert node in tree
    if self.node_input.text().isdigit():
        key = int(self.node_input.text())

        if key is not None:
            if self.tree is None:
                self.tree = TreeNode(key)
            else:
```

```
                self.tree.insert(key)
        self.plot_tree()
        self.node_input.clear()
```

Responsável pela inserção de um novo nó na árvore quando o usuário realiza essa ação. Verifica se o valor inserido é um número inteiro válido. Se for, converte para inteiro e insere o novo nó na árvore existente. Caso a árvore ainda não exista, cria uma nova árvore com o nó inserido. Após a inserção, a função de atualização da visualização da árvore é chamada para refletir as alterações na interface.

- **import_tree(self):**

```
def import_tree(self):
    # Imports the tree from the txt

    options = QFileDialog.Options()
    options |= QFileDialog.ReadOnly
    file_name, _ = QFileDialog.getOpenFileName(self, "Selecionar Arquivo", "",
    "Arquivos de Texto (*.txt);;Todos os Arquivos (*)", options=options)

    if file_name:
        with open(file_name) as file:
            data = file.readlines()
            entry = [int(i.replace("\n", "")) for i in data if i.strip().isdigit()]

        if self.tree is None:
            self.tree = TreeNode(entry[0])
            for i in entry[1:]:
                self.tree.insert(int(i))
        else:
            for i in entry:
                self.tree.insert(int(i))

    self.plot_tree()
```

Ativada quando o usuário deseja importar elementos de um arquivo de texto para a árvore. Abre uma janela para seleção do arquivo, lê os dados do arquivo e os processa para inserção na árvore. Se a árvore não existir, cria uma nova árvore com os elementos do arquivo. Caso contrário, apenas insere os elementos na árvore existente. Após a importação, a função de atualização da visualização da árvore é chamada para refletir as alterações na interface.

2.2.5 Matplotlib em ação

A representação visual da árvore binária de pesquisa (BST) é realizada utilizando a biblioteca Matplotlib. A implementação das funções `plot_tree` e `plot_tree_recursive` demonstra como os nós da árvore são posicionados e conectados graficamente.

```
def plot_tree(self):
```

```
        self.figure.clear()
        ax = self.figure.add_subplot(111)
        self.plot_tree_recursive(ax, self.tree)
        self.canvas.draw()

def plot_tree_recursive(self, ax, node, x=0, y=0, espaco_horizontal=1, altura=1):
    if node:
        ax.text(x, y, str(node.key), ha='center', va='center', bbox=dict(facecolor='whi
        if node.left:
            x_left = x - espaco_horizontal / 2 ** altura
            y_left = y - 1
            ax.plot([x, x_left], [y, y_left], 'k-')
            self.plot_tree_recursive(ax, node.left, x_left, y_left, espaco_horizontal /
        if node.right:
            x_right = x + espaco_horizontal / 2 ** altura
            y_right = y - 1
            ax.plot([x, x_right], [y, y_right], 'k-')
            self.plot_tree_recursive(ax, node.right, x_right, y_right, espaco_horizonta
```

A função `plot_tree` é responsável por limpar a figura atual, adicionar um novo subplot e chamar a função recursiva `plot_tree_recursive` para desenhar a árvore. Por sua vez, a função `plot_tree_recursive` posiciona os nós da árvore de forma hierárquica, conectando os nós pais aos seus nós filhos e desenhando os nós com seus respectivos valores.

A utilização de coordenadas x e y para posicionar os nós, juntamente com a recursividade para percorrer todos os nós da árvore, permite a representação visual precisa e organizada da BST. Os parâmetros *espaco_horizontal* e *altura* são utilizados para controlar o espaçamento horizontal entre os nós e a altura da árvore, garantindo uma disposição adequada dos elementos no gráfico.

Este trecho de código exemplifica a integração eficiente entre a biblioteca Matplotlib e a aplicação em PyQt, possibilitando a visualização gráfica da estrutura da árvore binária de pesquisa de forma clara e informativa para o usuário.

2.3 treenode.py

O arquivo `treenode.py` contém a implementação da classe `TreeNode`, que representa os nós da árvore binária de pesquisa (BST). Além disso, inclui funções para inserção, exclusão e listagem de elementos na árvore.

- **TreeNode**: Esta classe possui um construtor que inicializa um nó com suas propriedades (`right`, `left` e `key`). Ela também possui métodos para inserir um novo nó na árvore (`insert`), imprimir a árvore em ordem (`print_in_order`), pré-ordem (`print_pre_order`) e pós-ordem (`print_post_order`).

- **insert**: Esta função recursiva recebe um nó raiz (`root`) e uma chave (`key`) a ser inserida na árvore. Ela insere o nó com a chave especificada na posição correta de acordo com as regras da BST.

- **delete_node**: Esta função recursiva recebe um nó raiz (`root`) e uma chave (`key`) a ser removida da árvore. Ela realiza a exclusão do nó com a chave especificada e reorganiza a

árvore de acordo com as regras da BST.

3 Funcionalidades

A aplicação desenvolvida oferece diversas funcionalidades para interação do usuário com a árvore binária de pesquisa:

- **Inserção de Elementos**: O usuário pode inserir novos elementos na árvore utilizando o campo de entrada e o botão "Inserir". O código associado a esta funcionalidade verifica se o valor inserido é um número inteiro e realiza a inserção na árvore.
- **Exclusão de Elementos**: A exclusão de elementos é realizada através do botão "Excluir", onde o usuário informa o valor a ser removido. O código verifica se o valor inserido é um número inteiro e executa a exclusão do nó correspondente na árvore.
- **Listagem da Árvore**: A aplicação permite listar os elementos da árvore em diferentes ordens: em ordem, pré-ordem e pós-ordem. Ao clicar no botão "Listar", o código associado a esta funcionalidade exibe os elementos da árvore na ordem desejada.
- **Importação de Dados**: O usuário pode importar elementos de um arquivo de texto para a árvore. Ao clicar no botão "Importar de arquivo", o sistema abre uma janela para seleção do arquivo. O código lê os dados do arquivo, verifica se são números inteiros e realiza a inserção na árvore.

4 Conclusão

A implementação de uma árvore binária de pesquisa com interface gráfica em PyQt apresenta uma aplicação prática dos conceitos de estruturas de dados e programação orientada a objetos. A visualização da árvore facilita a compreensão de seu funcionamento, enquanto a interface intuitiva permite uma interação amigável com o usuário.

Ao explorar as funcionalidades de inserção, exclusão, listagem e importação de elementos na árvore, o usuário pode aprender e experimentar com o funcionamento da BST de forma interativa. Isso contribui para uma melhor compreensão dos conceitos de árvores binárias de pesquisa e suas aplicações na computação.

Por fim, a combinação da eficiência da BST na organização e busca de dados com a usabilidade proporcionada pela interface gráfica em PyQt resulta em uma ferramenta versátil e educativa para estudantes e profissionais da área de computação.