Árvore de Pesquisa Binária

Nome:

Pedro Tian Xi Fruck Liu

Introdução:

Este relatório documenta a implementação de uma Árvore de Pesquisa Binária em Python, seguindo as diretrizes do trabalho. Após estudar os algoritmos e referências indicadas, foi desenvolvida a árvore implementando alguns métodos essenciais. Testes foram realizados para verificação de seu funcionamento.

Estrutura:

São utilizados três arquivos:

- Node.py: define a classe ‘Node’, que representa um nó na árvore de pesquisa binária. Cada nó possui um valor (key), referências para o nó pai, filho da esquerda e filho da direita. A classe tem métodos que retornam ou modificam o valor, nó pai, nó filho da direita e nó filho da esquerda.
- ArvPesqBinaria.py: O arquivo contém a classe ‘ArvPesqBinaria’, que representa a árvore de pesquisa binária. Possui métodos que incluem inserção de elementos, verificação se a árvore está vazia, obtenção da raiz, busca de elemento na árvore, contagem total de nós, altura, identificação de pai de algum elemento e diferentes caminhamentos para visualização.
- main.py: Arquivo para instanciar as árvores e possui interface para interagir com elas através do terminal. A interface aciona métodos da classe ‘ArvPesqBinaria’.

Algoritmo e Funcionamento:

Node.py:

A classe Node é responsável por representar os nós que compõem a Árvore de Pesquisa Binária (APB). Cada nó contém uma chave, uma referência para o nó pai, uma referência para o nó à direita e uma referência para o nó à esquerda. O construtor recebe como parâmetro apenas o valor do nó(‘key’).

Métodos get:

Os métodos get apenas retornam o valor de cada atributo e não recebem nenhum parâmetro.

```
def getKey(self):
    return self.key
```

Métodos set:

Os métodos set atualizam o valor de cada atributo, o novo valor é recebido através de seu parâmetro.

```
def setKey(self, key):
    self.key = key
```

ArvPesqBinaria:

A classe ArvPesqBinaria implementa uma Árvore de Pesquisa Binária (APB), uma estrutura de dados que facilita a organização e busca eficiente de elementos. A árvore é composta por nós que mantêm uma relação de ordem de acordo com suas chaves.

Método is_empty:

Este método é utilizado para fazer a verificação se a árvore está vazia ou não, por meio da existência ou não da raiz, retornando verdadeiro ou falso.

Método get_root:

Apenas retorna a raiz da árvore de pesquisa binária.

Método insert:

O método recebe de parâmetro o valor do nó que será inserido. Primeiramente, é criado o nó com o valor recebido do parâmetro, e então, efetuado a verificação se a árvore está vazia, se estiver, o nó virá a raiz da árvore; se não, será efetuado um laço de repetição para alocar o nó no lugar certo. Antes de começar a repetição, são criadas duas variáveis(nó pai e nó atual) para que seja possível o percorrer na árvore.

```
while True:
    if (nodo_atual != None):
        nodo_pai = nodo_atual
        if (nodo.getKey() < nodo_atual.getKey()):
            nodo_atual = nodo_atual.getLeft()
        else:
            nodo_atual = nodo_atual.getRight()
```

Dentro do laço, primeiro se verifica se o nó atual, que é a raiz, existe ou não; se sim, ocorre outra validação, se o valor do nó a ser inserido for menor

que o nó atual da repetição, este assume o valor do filho menor(à esquerda), se for maior, o nó atual assume o valor do filho maior(à direita).

Método search:

Realiza uma busca na árvore por uma chave específica. O método recebe nos parâmetros o valor do nó a ser buscado. Primeiro verifica-se se a árvore está vazia. Caso contrário, percorre a árvore de maneira eficiente(conforme a lógica de uma árvore binária) verificando se o valor do nó é igual ao valor procurado.

Método size:

Percorre a árvore utilizando recursividade e conta quantos nós existem na árvore, retornando este valor final. A condição de encerramento da recursão acontece quando a variável que representa o nó atual é None.

```
else:
    return 1 + self.size(nodo_atual.getLeft()) + self.size(nodo_atual.getRight())
```

Vale ressaltar que o '+ 1' faz com que a cada chamada recursiva seja somado 1(exceto na condição de encerramento), trazendo o resultado final no término da operação.

Método height:

Retorna a altura da subárvore ou árvore com raiz no nó fornecido. De parâmetro, recebe a raiz da subárvore. Com recursividade, percorre a árvore de maneira que achará o caminho em que há mais nós subsequentes, da maneira que é calculado a altura de uma árvore. Retorna este valor. A condição de parada é quando o nó da vez for None.

Método get_father:

Retorna e imprime o pai de um nó específico. O parâmetro deverá receber o valor do nó cujo pai será retornado. Percorre a árvore com um laço de repetição 'while' com algumas condições: se o valor do nó atual for igual ao nó atual, retorna o pai do mesmo ou se o valor do nó buscado for menor que o do atual, a busca segue pela esquerda ou se for maior, a direita, ou se não for nenhum destes casos, notifica que não foi encontrado.

Caminhamentos:

Os métodos 'pre_order', 'in_order' e 'post_order' realizam os caminhamentos em Pré-ordem, Em ordem e Pós-ordem, respectivamente. Recebe nos parâmetros o nó a partir do qual o caminhamento é realizado.

main.py:

Neste arquivo, apenas são instanciadas três árvores binárias de pesquisa e há uma interface para interagir com elas via terminal. Pela escolha que for feita perante a interface, é acionado métodos da classe ArvPesqBinaria. Cada interface interage apenas com uma árvore.

```
    interface = input("""
--------------------------------------------------------------------------------
                ÁRVORE 01
1. ADICIONAR elementos na árvore.
2. Retornar o PAI de um elemento.
3. Verificar qual é a ALTURA da árvore.
4. Verificar quantos ELEMENTOS tem na árvore.
5. Verificar se a árvore está VAZIA.
6. Retornar os elementos da árvore com CAMINHAMENTOS(em ordem, pré ordem e pós ordem).
7. Sair e ir para interface da árvore 02.

Digite a opção que você quer executar:""")
```

Testes

- Teste 1:
  - Instanciar a primeira árvore; Incluir os números 1,2,3,4,5,6,7,8,9, nesta ordem, na árvore; Apresentar a altura da árvore;

```
Digite a opção que você quer executar:3
9
```
Resultado correto.

  - Apresentar a quantidade de nodos da árvore;

```
Digite a opção que você quer executar:4
9
```
Resultado correto.

  - Apresentar o conteúdo da árvore usando um caminhamento;

```
Digite a opção que você quer executar:6

1 2 3 4 5 6 7 8 9

1 2 3 4 5 6 7 8 9

9 8 7 6 5 4 3 2 1
```
Resultado correto.

- Teste 2:
  - Instanciar a segunda árvore; Incluir os números 9,8,7,6,5,4,3,2,1, nesta ordem, na árvore; Apresentar a altura da árvore;

```
Digite a opção que você quer executar:3
9
```
Resultado correto.

  - Apresentar a quantidade de nodos da árvore;

```
Digite a opção que você quer executar:4
9
```
Correto esperado.

- Apresentar o conteúdo da árvore usando um caminhamento;

```
1 2 3 4 5 6 7 8 9

9 8 7 6 5 4 3 2 1

1 2 3 4 5 6 7 8 9
```

Correto esperado.

- Teste 3:
  - Instanciar a terceira árvore; Não incluir nenhum elemento; Apresentar a altura da árvore;

```
Digite a opção que você quer executar:3
0
```

Correto esperado.

  - Apresentar a quantidade de nodos da árvore;

```
Digite a opção que você quer executar:4
0
```

Correta esperado.

  - Apresentar o conteúdo da árvore usando um caminhamento;

```
Digite a opção que você quer executar:6
```

Não retorna nada pois não tem caminhamento para 0 elementos.

Conclusão

Em resumo, a implementação da Árvore de Pesquisa Binária em Python proporciona uma estrutura organizada para armazenamento e busca eficiente de elementos. A validação por meio de testes demonstrou a correta funcionalidade do código, atendendo aos requisitos propostos. O trabalho contribuiu para a compreensão dos conceitos relacionados a árvores binárias de pesquisa.

Referências

- Conteúdo de aula de Algoritmo e Estrutura de Dados, Senac-RS, Professora Drª. Eduarda Rodrigues Monteiro
- https://www.ime.usp.br/~pf/algoritmos/aulas/binst.html
- https://www.if.ufrgs.br/~thielo/arvores1.pdf