ALGORITMOS E ESTRUTURAS DE DADOS - IF969
2019.2

LISTA DE MONITORIA III

Deste exercício em diante, sempre que forem solicitados a implementar uma classe em Python, implementem também os métodos especiais pertinentes a sua classe. Isto é, os métodos __str__, __repr__, __getitem__, __setitem__, __iter__, __next__, __dict__ e assim por diante. Deixarei ao fim desta lista material extra caso queiram se aprofundar nos métodos especiais.

Na lista III trabalharemos com árvores, em especial árvores binárias e suas derivações.

Q1. Implemente a classe árvore binária, assim como seus métodos especiais e particularidades de acordo com as diretrizes a seguir:

- 1. Implemente o método especial __bool__. Esse método converte a estrutura em um valor booleano e, no caso da nossa árvore, essa conversão é bem simples: se a árvore estiver vazia o método deve retornar False, se não, deve retornar True;
- 2. O método __str__ deve imprimir os elementos da árvore do caminhamento em ordem;
- 3. O método __repr__ deve ser capaz de inicializar uma instância de árvore de igual valor e, por tanto, deve dispor os elementos impressos em pré-ordem;
- 4. O método __iter__ também deve iterar pela árvore em pré-ordem;
- 5. O método pós-ordem é normalmente usado para a remoção um-a-um dos nós da árvore e, nesse caso, será implementada em um método personalizado chamado 'reiniciar'. Esse método deve retornar um iterador que, depois de retornar o valor guardado em um dado nó, deve remover aquele nó da árvore;
- 6. Além dos métodos __getitem__ e __setitem__, teremos na nossa árvore binária o método __delitem__. Esse método é responśavel pelo comportamento da estrutura quando uma expresão do tipo 'del arvore[0]' é chamada. Para implementar a remoção de forma apropriada em árvores binárias, você deve optar pela adoção da seleção de antecessor ou sucessor na remoção, independente do escolhido, ambos os métodos antecessor e sucessor devem ser implementados;
- 7. Por fim, sua árvore deve também implementar o método __contains__. Com este método, dado uma chave, o método retorna True ou False para determinar se a chave está contida na árvore. Esse é o método responsável pelo comportamento de comandos como "'chave' in arvore";

Por fim, há também algumas diretrizes à implementação da classe auxiliar nó:

- 1. A classe nó deve guardar um par chave-valor, sendo assim indexada pela chave. Embora normalmente qualquer objeto possa ser usado para indexar um nó, na nossa implementação de árvore binária considere que as chaves serão sempre do tipo string.
- 2. Implemente os métodos comparativos, são eles __lt__ (<), __gt__, (>), __le__ (<=),

__ge__ (>=), __eq__ (==) e __ne__ (!=). Esses métodos devem receber como parâmetro um outro nó e comparar os valores (não as chaves) dos dois. Caso a comparação seja entre um nó e um objeto de qualquer outro tipo, seu método deve levantar uma exceção do tipo TypeError;

Por último, implemente um método ‘chaves’ e um método ‘valores’ que varra a árvore e retorne somente as chaves ou somente os valores nela presentes.

Q2. Usando o conceito de herança, implemente a classe AVL como herança direta da sua árvore binária. Assim como para árvore binária há algumas restrições e cuidados a serem tomados na implementação da AVL:

- 1. Implemente um método estático “balanceada” que recebe como parâmetro uma árvore binária e retorna True ou False dependendo do nível de balanceamento da árvore.
- 2. Implemente um outro método estático “balancear” que receba como parâmetro uma árvore binária e faça o balanceamento da árvore existente. Esse método não deve retornar uma nova árvore, mas sim balancear o objeto passado como parâmetro e, por tanto não deve possuir retorno.
- 3. Implemente o método “coef_balanceamento”, este deve receber como parâmetro uma subárvore e retornar o coeficiente de balanceamento desta. Implemente o método de forma recursiva.
- 4. Implemente os métodos “girar_esquerda” e “girar_direita”.
- 5. Por fim faça a sobrescreva os métodos de inserção e remoção para que sua AVL se mantenha balanceada.

Uma dica para a questão 2. Sempre que implementar uma herança, lembre-se de chamar o método __init__ da classe parente e implementar no método __init__ da subclasse apenas as peculiaridades desta.

Árvores 2-3 e B são estruturas de dados bastante complexas de se implementar e, por isso, para a próxima questão, eu aconselho que sigam o passo a passo ilustrado na mesma. Lembre-se, em caso de dúvidas, não tenham medo de entrar em contato conosco.

Q3. Implemente uma árvore 2-3 usando uma nova estrutura auxiliar nó. Para esta árvore, se preocupe apenas com o método de inserção e o método __str__. Seguem algumas dicas:

- 1. Você pode usar listas de Python para armazenar as chaves e as referências para os filhos de cada nó, ao invés de usar um atributo para cada chave (como self.chave_menor e self.chave_maior). Se quiser, implemente uma estrutura auxiliar “Chaves” e uma estrutura auxiliar “Lista de Filhos” para usar dentro da estrutura Nó;
- 2. Implemente o método especial __add__ para realizar a concatenação entre duas instâncias de Nós;
- 3. Faça as inserções recursivamente, dessa forma você elimina a necessidade de uma entidade observadora externa.

Para essa lista, vocês devem implementar cada uma das questões em scripts individuais. Quando necessário, vocês podem importar os scripts uns nos outros para facilitar o reuso de código. Caso queiram implementar casos de testes, usem a expressão “if __name__ == ‘__main__’” para isolar o caso de testes no seu script e não importá-lo junto para outros scripts.

Referências Métodos especiais:
https://pythonhelp.wordpress.com/2013/03/11/os-metodos-magicos-de-python/
http://ptcomputador.com/P/python-programming/94049.html
https://docs.python.org/3/reference/datamodel.html

Métodos estáticos:
https://www.youtube.com/watch?v=QP4vJg8q-ZA
__name__ == ‘__main__’:
http://www.devfuria.com.br/python/entenda-__name__-__main__/