Módulo 9 - Laboratório prático – Exercícios

Em cada módulo, os exercícios assinalados com ♣ são mais importantes. Os exercícios extra apresentam treino extra ou aplicações importantes dos TAD do módulo. São exercícios cuja prática complementa os conceitos e ajuda a perceber como os aplicar.

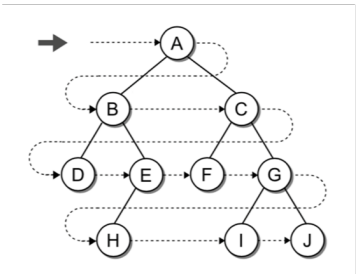
Exercícios:

1. ♣ Considere a classe BinaryTree, que implementa um TAD Árvore Binária com uma classe privada (_BTreeNode).

```
# Classe para nó de árvore binaria
class BSTree:
        def __init__(self):
                self._root = None
            ( … )
        class _BTreeNode:
                def __init__(self, elem):
                    self.data = elem
                    self.left = None
                    self.right = None
```

Desenvolva um método para a classe BinaryTree:

```
def breadth_first_traversal():
          '''Este método imprime todas as
         chaves da árvore por nível'''
```



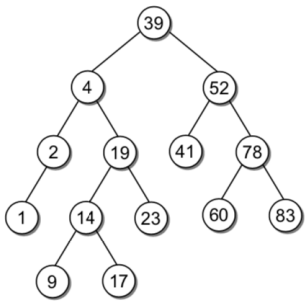
source: Data Structures and Algorithms, R. Necaise

que imprime todas as chaves dos nós da árvore por nível, isto é, na orientação da sequência apresentada na figura ao lado. Neste exemplo, o primeiro elemento a imprimir será "A", depois "B", e assim sucessivamente.

O método desenvolvido só precisa de se socorrer dos objetos da classe _ BTreeNode, dos seus atributos e do nó raiz para fazer a travessia.

Nota: Deve usar uma estrutura Fila (Queue), nomeadamente métodos para inserir novo elemento no fim da estrutura; outra que devolve o elemento no início da estrutura; e a que retira o elemento no início da estrutura e o devolve.

2. ♣ Considere a árvore AVL seguinte e responda às alíneas abaixo indicadas.



Nota: Deve desenhar todas as alterações efetuadas, desde a colocação do elemento na árvore até ao reequilíbrio final.

a. Desenhe a árvore AVL que resulta das inserções seguintes na árvore da figura acima:
   i. um elemento com chave 88;
   ii. um elemento com chave 95.

b. Desenhe a árvore AVL que resulta da remoção do elemento com chave 23 na árvore da figura acima.

3. ♣ Implemente o TAD Árvore AVL como subclasse da classe Árvore Binária de Pesquisa (ABP) desenvolvida na aula anterior de tal modo que:

c. Crie um método **recursivo**, `balanced()`, que indica se a Árvore está equilibrada.

d. Reimplemente os métodos de inserção e remoção próprios para uma Árvore AVL, ou seja, de modo a manter a propriedade de equilíbrio, implementando ainda todos os métodos auxiliares que necessitar.

Exercício Extra

1. ♣ Defina uma classe Árvore que armazena nós implementando o TAD de Árvore geral (n-ary) não ordenada. A sua classe deve implementar a seguinte interface:

Método - assinatura	**Descrição**
len()	Devolve a quantidade de nós na árvore.
root()	Devolve o nó raiz da árvore.
parent(X)	Devolve o Node pai do nó com valor (de chave) X e devolve None se X estiver na raiz. Nota: lança uma exceção se não encontrar o valor indicado.
height()	Devolve a altura da árvore.
create(X)	Cria uma estrutura de árvore, criando o nó raiz com valor X.
add_child (Parent, X)	Cria um nó na árvore como descendente do nó Parent e com valor X. Caso a árvore esteja vazia ou Parent não exista, lança uma exceção.
children(X)	Devolve um iterador sobre todos os filhos do nó com valor X.
is_node(X)	Indica se X está num nó da árvore ou não.
get_node(X)	Devolve o ponteiro para o nó onde está armazenada a chave X na árvore. Se não existir um nó com valor de chave X, devolve None.

Nota: Caso existam nós com chaves duplicadas, a operações de interface que usam a pesquisa da chave devolvem/indicam o primeiro nó que apresenta a chave procurada.

Sugere-se utilizar a seguinte estrutura para a implementação da classe: