Assignment 5.1: Binary Search Tree (3 points)

Implement a binary seach tree in Python. The tree stores only integers (`int`).

Create following classes:

- `Node`: stores the integer value (`int`) and links to its left and right child
- `BST`: maintains the binary search tree built using `Node` objects.

Create following methods for class `BTS`:

- `preorder()`: prints the contents of the search tree in preorder.
- `insert(key: int)` (**1 pt**): inserts a `key` to the search tree, ignores duplicates.
- `search(key: int)` (**1 pt**): searches `key` from the search tree and returns boolean True if found, False otherwise.
- `remove(key: int)` (**1 pt**): removes `key` from the search tree, maintains the BST property. Implement `remove` using the maximum node principle.

A code template with an example program:

```
class Node:
    # TODO


class BST:
    # TODO


if __name__ == "__main__":
    Tree = BST()
    keys = [5, 9, 1, 3, 7, 7, 4, 6, 2]

    for key in keys:
        Tree.insert(key)

    Tree.preorder()

    print(Tree.search(6))
    print(Tree.search(8))

    Tree.remove(1)
    Tree.preorder()
    Tree.remove(9)
    Tree.preorder()
    Tree.remove(3)
    Tree.preorder()
```

Output:

```
$ python bintree.py
5 1 3 2 4 9 7 6
True
False
5 3 2 4 9 7 6
5 3 2 4 7 6
5 2 4 7 6
```

Submit your solution in CodeGrade as `bintree.py`.

Assignment 5.2: More Traversals (3 points)

Update your class `BST` with three new traversal methods:

- `postorder()` (**1 pt**): prints the content of the search tree in postorder.
- `inorder()` (**1 pt**): prints the content of the search tree in inorder.
- `breadthfirst()` (**1 pt**): prints the content of the search tree in breadth-first-order.

Breadth-First enumeration presents the nodes of the search tree level by level (or depth), unlike other impemented methods which traverses the left subtree first before the right subtree.

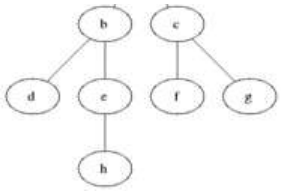

Figure 1: Breadth-First enumeration for a binary tree. Black: explored, grey: queued to be explored later on (source: wikipedia.org).

An example program:

```
# bintree.py


if __name__ == "__main__":
    Tree = BST()
    keys = [5, 9, 1, 3, 7, 7, 4, 6, 2]

    for key in keys:
        Tree.insert(key)

    Tree.postorder()
    Tree.inorder()
    Tree.breadthfirst()
```

Output:

```
$ python bintree.py
2 4 3 1 6 7 9 5
1 2 3 4 5 6 7 9
5 1 9 3 7 2 4 6
```

Submit your updated version of `bintree.py` in CodeGrade.

Assignment 5.3: Mirroring The BST (3 points)

Implement a new method `mirror()` to your class `BST`. The method mirrors the search tree along the root node, i.e. each nodes' left and right child nodes will swap places.

When the tree is mirrored, the traversal methods displays the tree in mirrored order (see example). However, remember that the BST property must remain whether the tree is mirrored or not!

An example program:

```
if __name__ == "__main__":
    Tree = BST()
    keys = [5, 9, 1, 3, 7, 7, 4, 6, 2]

    for key in keys:
        Tree.insert(key)

    Tree.preorder()
    Tree.mirror()
    Tree.preorder()

    Tree.insert(8)
    Tree.remove(3)
    print(Tree.search(2))
    Tree.preorder()
    Tree.mirror()
    Tree.preorder()
```

Output:

```
$ python bintree.py
5 1 3 2 4 9 7 6
5 9 7 6 1 3 4 2
True
5 9 7 8 6 1 2 4
5 1 2 4 9 7 6 8
```

Submit your updated version of `bintree.py` in CodeGrade.

Last modified: Wednesday, 26 June 2024, 9:21 AM