During week 3, we covered the linked list and how can it be implemented in Python. A binary search tree can be considered as an extended version of a linked list. Instead of one link a BST node has two links: left and right.

```
# BST Node
class Node:
    def __init__(self, key: int):
        self.key = key
        self.left = self.right = None
```

Like previously we create a separate class for BST itself which stores the actual tree maintaining the binary search tree property.

```
class BST:
    def __init__(self):
        self.root = None
```

The property of BST is that all the nodes in the right subtree are greater than the root and all the nodes in the left subtree are smaller than the root. Due to this property, searching a key from the BST can be done using recursion.

```
def search(self, key):
    return self.search_help(self.root, key)

# help function for 'search'.
def search_help(self, node, key):
    if not node:
        return False
    elif node.key > key:
        return self.search_help(node.left, key)
    elif node.key < key:
        return self.search_help(node.right, key)
    # None of the conditions were true. That
    # means the node stores the key.
    return True
```

Recursion is commonly used for algorithms in recursive data structures such as BST. However some algorithms for BST can be also implemented iteratively. Preorder traversal is an algoritmh which can be implemented in both ways. In iterative version we store the order of next nodes to visit in a stack.

```
# order: 1. root, 2. left, 3. right
def preorder(self):
    stack = [self.root]     # start: stack with root only
    while stack != []:
        next = stack.pop(-1) # pop the node from the top of the stack
        # push right and left children of 'next' into the stack
        if next.right != None:
            stack.append(next.right)
        if next.left != None:
            stack.append(next.left)
        print(next.key, end = " ")  # visit 'next'
    print()
```

Note that in the programming assignment you will implement the preorder traversal recursively. See the background material covering binary trees in general for a recursive version of preorder traversal.

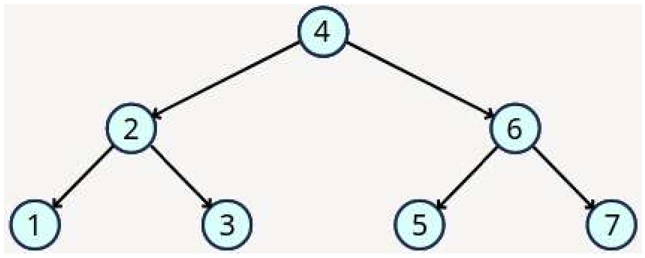
Example 1.1: The BST in action

```
class Node:
    def __init__(self, key: int):
        self.key = key
        self.left = self.right = None


class BST:
    def __init__(self):
        self.root = None

    # inserts new key to the BST
    def insert(self, key):
        # ...

    def search(self, key):
        # ...

    def preorder(self):
        # ...


if __name__ == "__main__":
    tree = BST()
    keys = [4, 2, 6, 1, 3, 5, 7]
    for key in keys:
        tree.insert(key)
    tree.preorder()
    print(tree.search(5))
    print(tree.search(8))
```

Example 1.2: The output

```
$ python3 example.py
4 2 1 3 6 5 7
True
False
```

Example 1.3: Structure of the stored binary search tree



Last modified: Thursday, 26 September 2024, 1:49 PM