Beihang University
School of General Engineering

Beijing, Nov. 22th, 2019
Sebastian Wandelt, Pengfei Yi, Junyu Zhao

Computer Science and Programming
Lab Class 10

Task 1. ***Counting sort*** *(15 minutes)*

Write a function *counting_sort(L,k)* to sort a list L within linear time. Here, all elements in L are integers in the range 0 to k. Try your algorithm on $L = [2, 5, 3, 0, 2, 3, 0, 3]$, with $k = 6$.

Task 2. ***Binary Search Trees*** *(50 minutes)*

Please read and understand the following short code snippet for storing a tree in Python.

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

tree = Node(2, Node(1),Node(3))
```

Implement the following operations on the tree class:

1. Use the class structure above to create a new tree with at least 7 numbers (an example could be the tree in Page 23 in the lecture slide).
2. Implement a function that iterates through and print out each leafs in a given tree. That is, the function should access each leaf element in the tree and print them out one by one.
3. Implement a function that print all elements in a tree in infix-order traversal. Infix means that the left children are explored first, then the data of a node is printed, and finally the right children are explored further. For instance, in the case of the given tree above, the function should print out: 1,2,3.
4. Implement a function which returns the length of the longest path in the tree.
5. Implement a function that checks whether a tree is a binary search tree, i.e. a tree such that all left-descendants are smaller than a node's key and all right-descendants are larger than the key of a node.
6. Implement a member function which finds a given key in a binary search tree. If the key is not contained, your function should return *None*.
7. Implement a function which constructs a binary search tree from a given list L. That is, the binary search tree should contain all the elements in the list and therefore a user could use the binary search tree to accelerate the search process.

Task 3. ***Red-black Trees - Basic*** *(30 minutes)*

Please implement (extend) the tree class from the previous tasks to make it able to model red-black trees.

1. For each node, or element in the tree, add a color attribute assciated with it. Because in this case we are dealing with RBTs, the color attribute should be either red or black.
2. Implement a function which checks whether or not a given tree instance is a red-black tree (satisfies all five criteria introduced in Page 42 in the lecture slide).

Note that this task is only a beginning of the implementation of RBTs. If one is interested in implementing the whole structure, the class and functions built in this task could be a beginning.