Data Structures
Homework #4

Due: Dec 5, 2023 (before class)

1. Suppose that we need to merge k sorted sequences into one sorted sequence and the total number of elements in these k sequences is n. A straightforward method is to select repeatedly the smallest one from the elements which are the smallest (at the first position) in each of the k sorted sequences. To find the smallest one in these k smallest elements from these k sequences needs $O(k)$ time. Because there are n elements, the worst case of the approach is $O(nk)$. Now, we would like to have a method that is better than the straightforward one and can be done in $O(n \log k)$ time. The idea of the method is depicted in Figure 1, where 8 sorted sequences are given and a data structure is needed to help the merge process. Your job is to provide a binary tree structure that can help merging these k sequences in $O(n \log k)$ time. Please describe the structure you provide and the process for merging the sorted sequences. In addition, please also explain why $O(n \log k)$ time can be achieved.

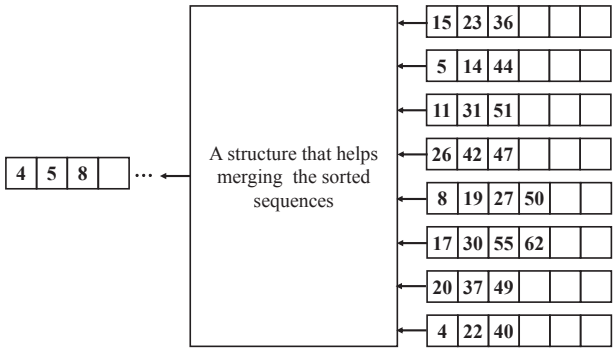

Figure 1: An example of merging 8 given sorted sequences into 1 sorted sequence

2. Show how to use the ***Euler tour*** traversal to compute the level number of each node in a binary tree T and give a pseudo-code for this approach. What is the running time of the approach?

3. Let T be an ordered tree with more than one node. Is it possible that the preorder traversal of T visits the nodes in the same order as the postorder traversal of T? If so, give an example; otherwise, argue why this cannot occur. Likewise, is it possible that the preorder traversal of T visits the nodes in the reverse order of the postorder traversal of T? If so, give an example; otherwise, argue why this cannot occur.

4. (15 pts) Let T be a tree with n nodes. Define the *lowest common ancestor* (LCA) between two nodes v and w as the lowest node in T that has both v and w as descendants (where we allow a node to be a descendent of itself). Given two nodes v and w, describe an

efficient algorithm for finding the LCA of v and w. What is the running time of your algorithm?

5. **Programming (Binary Search Trees)**
In this problem, you need to implement an ***array-based binary search tree*** T with the following associated operations(methods) using Python. We assume that each node is represented by a key value which is an integer. The operations include:

   (a) `insert(x)`: insert a node with key x into T.
   (b) `delete(x)`: delete a node with key x from T.
   (c) `searchkey(x)`: perform the search process from the root; if the search fails, print "NOT found"; otherwise, print "x is found".
   (d) `print()`: print out the properties of tree T, including the height and size of T.
   (e) `list_pre()`: list all the nodes using a preorder traversal.
   (f) `list_post()`: list all the nodes using a postorder traversal.
   (g) `list_in()`: list all the nodes using a inorder traversal.

   The binary search tree will be built or defined as a class, `BinarySearchTree`, and the operations mentioned above are defined as the methods. Of course, you can have other auxiliary methods. Initially, the binary search tree is empty. Your program allows the user to search, insert, and delete a node with a key. Please run `T.list_pre()`, `T.list_post()`, and `T.list_in()` for the same binary search tree to see what you can observe.

   Execution Example

```
RESTART: E:/binaryTree _using_Array-v2.py
>>> T=BinarySearchTree()
>>> T.searchkey(9)
NOT found
>>> T.insert(35)
>>> T.insert(45)
>>> T.insert(40)
>>> T.insert(43)
>>> T.insert(41)
>>> T.insert(42)
>>> T.delete(30)
Key 30 does not exist >>> T.insert(38)
>>> T.delete(38)
>>> T.print()
The height of T is 5 and the size of T is 6.
>>> T.list_pre()
35, 45, 40, 43, 41, 42
>>> T.list_in()
35, 40, 41, 42, 43, 45
>>> T.list_post()
42, 41, 43, 40, 45, 35
```