Middle East Technical University | Department of Computer Engineering

CENG 310

ALGORITHMS AND DATA STRUCTURES WITH PYTHON

Spring 2022-2023

Homework 3 - Linked Lists

Due Date: 24/05/2023 23:59

Instructions. In this assignment you will implement two tasks by using the **given** Binary Search Tree and Linked List structure. You will first complete the Linked List code and then solve 2 tasks by using your Linked List implementation. Note that you **can not** change the type of Binary Search Tree and Linked List structure you are given.

1 Implementation of Linked List

A template file *hw3template.py* will be provided to you in which you can find the skeleton code regarding Linked List. In the template you can find a class named LinkedList, an inner class named Node and each with their own member functions and variables. LinkedList class contains:

- __init__(self) function where it initializes LinkedList class.
- insert_last(self, e) is a function that adds a node with element e to the end of LinkedList class.
- print_contents(self) is a function that prints the current data of the nodes of the Linked List. Do not change anything in this function.
- head is a link that refers to the head node of the Linked List.
- size denotes the node count of the Linked List.

Node class is a simple class with only __init__(self, element, next) function with two member variables which are called data and next respectively.

Another function called create_lists(filename, taskNo) is provided to you. You need to fill out this function to read input from files and create Linked List objects. Your Linked List objects (and their content) can be tested during testing to see whether you created the desired Linked Lists.

You need to implement the given function inside the Linked List class and optionally define more helper functions. However, do not try to change the Singly Linked List structure (e.g, any solutions with Doubly Linked Lists **will not** be accepted).

2 Implementation of Binary Search Tree

In the same file (*hw3template.py*) you will also find the incomplete code for Binary Search Tree class which you need to complete. In the template, you can find a class named BST, an inner class named BSTNode and with their member functions and variables. It contains:

- __init__(self) function where it initializes BST class.
- insert(self, e) is a function that adds a BSTNode into the correct position of tree.
- print_inorder(self) prints contents of tree with inorder traversal. When printing print only the content and put a newline after each value.
- root is a link that refers to the root of the BST.

BSTNode class is a simple class with only __init__(self, data, left, right) function with three member variables which are called data left and right respectively.

On top of implementing insert and print_inorder functions you may want to implement other helper functions. However, do not try to change the definitions of classes and names of member variables.

3 Programming Task 1

In this task, you will implement a function named optional_concatenate(L1, L2, listToReverse) that takes two Linked Lists and a string that denotes which list to be reversed as input and combines these lists in a new Linked List. The function creates a new Linked List and then returns that list. Below you can find the instructions about how to perform this task.

- You will get the contents of each list from a txt file. A sample txt file will be provided to you.
- After creating each Linked List you need to call the function.
- Assume that the given L1 is 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7, L2 is 7 -> 8 -> 9 and listToReverse is "L2" you need to combine these as follows:
  - You need to combine these lists by starting from the beginning for L1 and starting from the back of L2. While combining put first element from L1 and second from L2 third from L1 and fourth from L2 and so on.
  - If at any point one of the lists become empty (or lists can be empty from the start) rest of the elements of nonempty list should be added on to new list.
  - New list, therefore, should be 1 -> 9 -> 2 -> 8 -> 3 -> 7 -> 4 -> 5 -> 6 -> 7. Note that you **do not** need to print the new list. Your function **must** return the new Linked List object.
- Make sure that optional concatenate operations are performed by your function. Do not just create a Linked List object with desired data simply by reading input file and reversing contents with list manipulation. Your function **must** perform Linked List operations to complete the task. If you do not create the Linked Lists given to you we can easily detect it from the output of create_lists function.

4 Programming Task 2

In this task, you will be given a singly Linked List and convert it into a Binary Search Tree (BST). The data of the Linked List will be in random order. A sample Linked List and its conversion to a BST can be seen below.

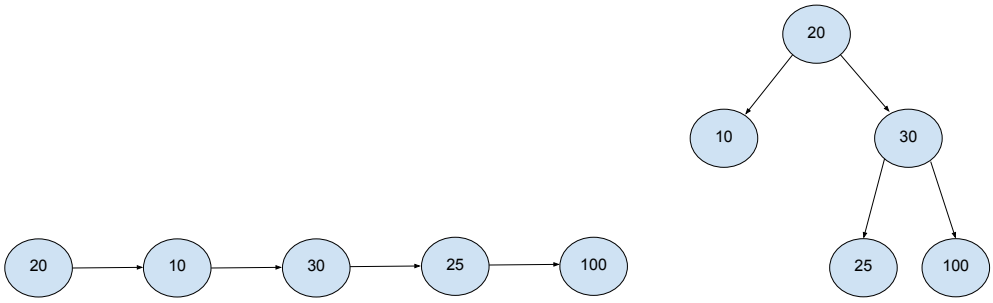

Figure 1: Linked List and corresponding BST

Depending on which order you implement the function you may get different BST structures. However, regardless of the structure inorder traversal will give the same result unless there is a mistake in your implementation.

5 Submission & Grading

- Grading will be done for each parts separately. Both tasks will be 50 points each.
- You will be given a template file *hw3template.py* which contains the Linked List class Binary Search Tree class and other functions that you **must** implement.
- Do not change the function and class definitions. You can not change the Singly Linked List structure.
- **Do not** leave your codes with unnecessary function codes and print statements. We will import your code and then call necessary functions ourselves to get the outputs. Any code that does not follow this rule will get penalized.
- You need to submit a single file with a *.py* extension.
- Be sure that you use the exact names for the classes and functions that you will implement. You can implement other helper functions to use inside of the functions you are provided.
- You should not worry about the run time of your program as long as it gives the desired output. However, extraordinarily long (in terms of time) solutions may result in getting no credit from particular test case.
- You will be provided sample Input and Output files for each task. Normally, we do not want you to use print_contents function. However, in order to show what your functions should do we provided output files. For example output file of task 1 is created by calling:

```
L1, L2 = create_lists("task1.inp")
L3 = optional_concatenate(L1, L2, "L2")
L3.print_contents()
```

However, this is printed just to show you the contents of the output of optional_concatenate which simply returns a new Linked List object (L3 in this case).