CMPSC-132: Programming and Computation II
Spring 2020

Lab #7

Due Date: 04/04/2020, 11:59PM EST

Read the instructions carefully before starting the assignment. Make sure your code follows the stated guidelines to ensure full credit for your work.

Instructions:

- **As a REMINDER, you are not allowed to use the internet to search for a solution or to use code written by other students that have taken (or are currently taking) the class**
- The work in this lab must be completed alone and must be your own.
- **Download the starter code file from the LAB7 Assignment on Canvas. Do not change the function names or given started code on your script.**
- A doctest is provided as an example of code functionality. Getting the same result as the doctest does not guarantee full credit. You are responsible for debugging and testing your code with enough data, you can share ideas and testing code during your recitation class.
- Each function must return the output
- **Do not include test code outside any function in the upload. Printing unwanted or ill-formatted data to output will cause the test cases to fail. Remove all your testing code before uploading your file (You can also remove the doctest). Do not include the input() function in your submission.**

Goal:

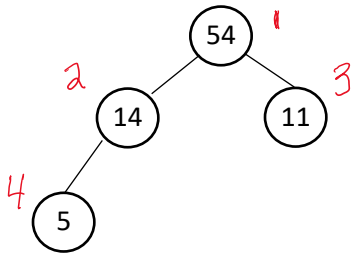
[10 pts] In Module 5 we talked about the Queue data structure and its FIFO property. For first come – first serve services, this could be a good data structure to use. However, when we want to process items based on precedence the queue might not be our best choice. Take for instance a job sent to a printer, each job is typically placed in a queue, but one job might be particularly more important, so it might be preferable to allow that job to run first as soon as the printer is available. A priority queue is a data structure that allows at least the following operations: insert (similar to enqueue) and deleteMin/Max (similar to dequeue). There are several ways to implement a priority queue, the most common one is using a Binary Heap. In Module 6 Part 1, we discussed the binary heap, which is a complete binary tree that satisfies the heap ordering property and can be implemented with an array. In a binary heap, we will always have the element with the maximum (or minimum) element in the root node of the heap.

Implement the class MaxPriorityQueue with the following operations:

- *parent*(index) returns the value of the parent of heap[*index*]. *index* is between 1 and the size of the heap. If index<=1 or index>size of the heap, it returns None.
- *leftChild*(index) returns the value of the left child of heap[*index*], *index* is between 1 and the size of the heap and returns None if there is no value
- *rightChild*(index) returns the value of the right child of heap[*index*], *index* is between 1 and the size of the heap and returns None if there is no value

*** The methods above are using the 1:n indexing. Note that as discussed in the video lecture, the index range of the array implementation of a heap is given from 1:n, NOT 0:n-1, so to return the correct value, you must subtract 1 from the video lecture formulas once you get the index (if you prefer, you can use the adjusted formulas for 0:n-1 indexing discussed during recitation, just account for that since input is 1 to n). For example using 1 to n formulas:

when heap = [54, 14, 11, 5]



0	1	2	3
54	14	11	5

	1:n index formula		Access to list
>>> parent(2) 54	k//2 = 2//2=**1**	value at node 1	heap[**1** -1] = heap[0]=54
>>> parent(4) 14	k//2 = 4//2=**2**	value at node 2	heap[**2** -1] = heap[1]=14
>>> leftChild(2) 5	2*k = 2*2 = **4**	value at node 4	heap[**4** -1] = heap[3]=5
>>> rightChild(1) 11	2*k+1 = 2*1+1= **3**	value at node 3	heap[**3** -1] = heap[2]=11

- *len*(heap_object) returns the number of items in the heap. It needs no parameters and returns an integer.
- *insert*(item) adds the item into the heap satisfying the max heap property. It modifies self.heap and returns nothing. Hint: use the first three methods to assist the process.
- *deleteMax*() removes the max element of the heap (the root node). It needs no parameters and **returns the value removed from the heap**. The heap satisfies the max heap property after deletion. (If you think on a recursive approach, you can create another method to help this method restore the max heap property, just don't forget to document it). Hint: use the first three methods to assist the process.

NOTE: You can't use heap functions from the Python library, otherwise no credit will be given. You are not allowed to modify the given constructor

Deliverables:

- Submit your code in a file name LAB7.py to the Lab7 GradeScope assignment before the due date