Due: Oct. 29, 2021 at 11:59pm

Overview

Create a max-heap structure and provide it with enough functionality to implement a Priority Queue.

Data Structures

You will primarily focus on implementing a binary Max-heap using a list L. Recall, if a node is stored as the k^{th} element of an array ($L[k]$), then its left child is in $L[2k+1]$, and its right child is in $L[2k+2]$.
Note: We will be using Python lists for this (which support random access) but in other languages you would use a fixed-length array.

You should fill out all of the methods in the provided skeleton code **mheap.py**. You may add additional private methods, but should not add any additional public methods or public fields. Like before, you should not alter any names for any of the classes, methods, or files. The provided **pqueue.py** provides a priority queue class that acts as a wrapper for the functions you develop in **mheap.py**.

Max-heap

`insert(self, data)` : Insert data into the heap. If the result violates the condition of a max-heap you should swap relevant nodes until a max-heap is achieved.

`peek(self)` : Return maximum value in the heap. This should not alter the heap itself.

`extract_max(self)` : Remove and return the maximum value in the heap.

`__heapify(self, curr_index, list_length=None)` : Given a node at `curr_index` check the left and right child and swap nodes as needed to reach a maximum heap.

`build_heap(self)` : Builds a maximum heap from the binary tree present in your heap structure.

heap_sort

Outside of the max_heap class you will also need to fill in the function `heap_sort(L)` which takes a list `L` and returns a new list which contains the elements of `L` given in ascending order.

Testing

Some sample test cases have been provided in **test_lab2.py** but this provided list is not exhaustive. You should provide test cases for the following:

— Check if a heap is built correctly after calling `build_heap()`

— Check if an `IndexError` is raised when inserting into a heap that is already full.

— Check if a `KeyError` is raised when `extract_max()` is called on an empty heap.

— Check if an object of `pqueue` is still a valid heap after the following series of `insert()` and `extract_max()` calls on the same `pqueue` object.

In addition to the above cases, you should add at least 6 additional test cases. More are encouraged.

Grading

The assignment will be graded as follows:

— AutoGrader - [70pts]

— Style - [10pts]

— Additional Test Cases - [15pts]

-2pt- **Test Case:** Check that `build_heap()` properly builds a max heap.

-2pt- **Test Case:** Check that an `IndexError` is raised when inserting to a heap that is already full.

-2pt- **Test Case:** Check that a `KeyError` is raised when `extract_max()` is called on an empty heap.

-4pt- **Test Case:** Check that an object of texttttpqueue is still a valid maximum heap after calling `insert()` and `extract_max()` on the same `pqueue` object.

-5pt- **Test Case:** Write 6 additional test cases of your own to **test_lab2.py**.

— DocStrings - [5pts]

-3pt- **DocString:** Write a docstring for the class `max_heap`. You should include the class attributes as well as the class methods and a brief summary of what the method does. See **Lab1.py** which shows a solid docstring for the `node` class.

-1pt- **DocString:**Provide a docstring for `__heapify()`. This one can be just a single line summary if you'd like.

-1pt- **DocString:**Provide a docstring for `build_heap()`. This one can be just a single line summary if you'd like.

To earn points for style, your code must be clear enough for us to understand. The rubrics for different functions will be provided in Gradescope.