Outstanding Work Option: Implement Min-Heap

Eckel, TJHSST AI1, Fall 2020

Background & Explanation

Usually, heaps are covered in Data Structures; this year, they were skipped. A good explanation video is at https://www.youtube.com/watch?v=WCm3TqScBM8. This information is also on the course website, but I've copied it here for easy reference as well:

Unlike the video, Python uses min-heaps, which means that the minimum value is always at index 0. Python doesn't use a separate data structure for heaps - it simply stores the heap in a list, just like the video above explains. In our code, we'll start with a blank list, and then use heap-specific commands to add elements to the list maintaining the heap invariant and to remove the minimum value element from the list, which will again cause a rearrangment to maintain the heap invariant. Both of these are already implemented by Python, and work exactly the way the video describes, but for minimum instead of maximum values. This means that removing the minimum value and adding a new value to the min-heap are both O(log n) time. Specifically:

- To use heaps, at the top of your code type "`from heapq import heappush, heappop, heapify`"
- Then, make a blank list, say for instance "`heap_list = []`"
- Each time an element is added to the list, use the command "`heappush(heap_list, new_item)`"
- To remove the minimum-valued element, use the command "`temp = heappop(heap_list)`" and the minimum value will be removed and stored in `temp`
- To simply look at the minimum-valued element, just look at `heap_list[0]`!
- And finally, it is extremely unlikely you'll need to use this (in fact, it will probably be a sign of bad understanding of our search algorithms!) but if you have a list of stuff and you want to turn it into a heap, you can use the command "`heapify(some_list)`", which will heapify the list in place (that is to say it doesn't make a new data structure, it simply turns "`some_list`" into a heap containing the same values)

At no point should the usual append or remove commands be used.

Task

You can get an Outstanding Work credit for implementing heaps this year in Python. To be specific, the above commands should perform the requisite heap functions WITHOUT importing anything from heapq. You should have **your own function definitions and code** for heappush, heappop, and heapify which should work exactly as specified above. In particular, it should be a **min-heap,** NOT the max-heap presented in the video.

(Note: I think the video linked above has a sequel where he talks about writing heapify in linear time; feel free to watch if you like to get a feel for that algorithm.)

Test your commands on a variety of test cases yourself. In particular, be sure to account for the fact that the last node may be the *only* child of its parent node in both pushing and popping.

Finally, be sure to **comment your code as you write** on this assignment! Since these commands already exist in Python's included libraries, I'll be reading your code to check this assignment, not just checking output. This is also stated in the specification below, but a good way to approach this is to assume I'm a reasonable person who knows how this works and I just want to see that your implementation is correct; ***comments are necessary*** but essays are ***not***.

Prepare Your Code to Turn In

Once your commands are written, prepare your code to turn in by implementing the following test protocol.

Your code should take an arbitrary number of command line arguments. Every argument will be either an integer, "A", or "R". There will be a large number of initial integers, followed by some number of A-integer pairs or standalone Rs.

- The large number of initial integers should be converted to ints and put in a list in the order they were written. Stop when you reach an "A" or "R". **Once this list is made, print it to the console.**
- Then, heapify the list with a call to your heapify function. **Output the heapified list to the console.**
- Then, for each "A" command, add the following integer to the heap using your heappush command. **Output the new list to the console.**
- For each "R" command, remove the lowest-valued element from the heap using your heappop command. **Output the removed value and the new list to the console.**

An example input would be: `>python yourfile.py 3 -1 5 17 8 -4 19 3 6 A 7 A -8 A 5 R R R R A 2`

This creates an initial list (the integers from 3 to 6), then adds 7, adds -8, adds 5, removes four values in a row, then adds 2. Sample output for this console input follows, but note that your lists *might not be identical* to the ones that follow. They must maintain the heap invariant (each parent is less than either of its children) but the precise order does not need to match. Note that the item at index 0 is always the minimum, however.

Sample output:

```
Initial list: [3, -1, 5, 17, 8, -4, 19, 3, 6]
Heapified list: [-4, -1, 3, 3, 8, 5, 19, 17, 6]
Added 7 to heap: [-4, -1, 3, 3, 7, 5, 19, 17, 6, 8]
Added -8 to heap: [-8, -4, 3, 3, -1, 5, 19, 17, 6, 8, 7]
Added 5 to heap: [-8, -4, 3, 3, -1, 5, 19, 17, 6, 8, 7, 5]
Popped -8 from heap: [-4, -1, 3, 3, 5, 5, 19, 17, 6, 8, 7]
Popped -4 from heap: [-1, 3, 3, 6, 5, 5, 19, 17, 7, 8]
Popped -1 from heap: [3, 3, 5, 6, 5, 8, 19, 17, 7]
Popped 3 from heap: [3, 5, 5, 6, 7, 8, 19, 17]
Added 2 to heap: [2, 3, 5, 5, 7, 8, 19, 17, 6]
```

Specification:

Submit your Python file to the link given on the course website.

This assignment is **complete** if:

- Your code behaves as described above.
- Your functions contain logical variable names and comments as necessary for me to be able to read them and follow your algorithms. (Assume I'm a reasonable person who knows how this works and I just want to see that your implementation is correct; ***comments are necessary*** but essays are ***not***.)
- The "Name" field on the Dropbox submission form contains your **class period**, then your **last name**, then your **first name**, in that order.

For **resubmission**:

- Correctly complete the specification.