Advanced Topics in Comp Sci — Activity—MaxHeap

ASSIGNMENT OVERVIEW

A *maxHeap* is a binary tree-based structure that displays the *heap* structure, with each parent having a value greater than the value of either of its children. In this activity you'll write a `Person` class, a `BinaryHeap` class, and a `binary_heap_demo.py` main program that demonstrates using a binary heap to manage a list of Very Important Persons (VIPs).

This assignment is worth 40 points and is due on the *crashwhite.polytechnic.org* server at 23:59:59 on the date given in class.

BACKGROUND

A *minHeap* is a binary tree-based structure that displays the *heap* structure, with each parent having a value less than the value of either of its children. It's common to implement and test a *minheap* using integers, which are easy to store and manipulate.

The methods of the minHeap-based BinaryHeap class include:

- `BinaryHeap()` constructs a new empty binary heap
- `insert(value)` places a new value in the binary heap in an appropriate location
- `find_min()` returns the minimum value in the heap, but doesn't remove it
- `del_min()` returns the minimum value in the heap and removes it from the heap
- `is_empty()` returns True or False depending on the state of the heap
- `size()` returns the number of elements in the heap
- `build_heap(value_list)` creates a new heap from the given list of values

Based on what you've already learned about implementing a *minHeap* using a Python list of integers, in this assignment you'll be implementing a *maxHeap* containing Person objects, organized according to the "VIP value" of each person on the list people, where a person with a higher VIP value is more important, and will be located higher on the heap.

PROGRAM SPECIFICATION

Write the MaxHeap project as a set of three files including `person.py` (defining the `Person` class), `max_heap.py` (defining the `BinaryHeap` class acting as a maxheap), and `max_heap_demo.py` (with a `main` method that runs a demonstration of your BinaryHeap-based VIP list). This demo program will create a series of `Person` objects, place them in the `BinaryHeap`, and perform operations on the heap that indicate how it works. (See the end of this document for sample output.)

DELIVERABLES

`max_heap.zip`

This single file will contain a `MaxHeap` directory with the files `person.py`, `max_heap.py`, and `max_heap_demo.py` .

To submit your assignment for grading, copy your file to your directory in `/home/studentID/forInstructor/` at *crashwhite.polytechnic.org* before the deadline.

ASSIGNMENT NOTES

- Begin by writing and testing the `person.py` file containing a definition of the `Person` class. For simplicity, a `Person` object need only contain `name` and `VIPvalue` instance variables, accessible via a couple of "getter" methods. Don't forget to write a `__str__` method for your Person so they can be displayed in a user-friendly manner.
- Use your previous minheap-oriented `binary_heap` module as a starting point and convert it to a maxheap implementation. You'll need to alter this module some:
  - We're not storing integers in the list now—instead we're storing `Person`s.
  - We're not comparing `int` objects now—we're comparing the `VIP_value` from `Person` objects.
  - We're not basing the heap on minimum values, but on maximum values.
  - Some of the method names may even change: there should no longer be a `get_min` method, for example.
- At some point, while writing your new `binary_heap` module, you'll tire of testing it by hand and start to think about writing the `binary_heap_demo.py` program, which will have a `main` method that uses the `BinaryHeap` class to manage a collection of `Person` objects.
- Make sure your tester runs through all the common "failure" scenarios:
  - Does your heap actually exhibit maxheap properties?
  - Does the `insert` method work correctly?
  - Do the `find_max` and `del_max` methods work correctly?

GETTING STARTED

1. Ensure that your minHeap `binary_heap.py` file is working as it should, and that you have a good understanding of the principles involved in managing a heap.

2. On your computer, create the `max_heap` directory that will store the files associate with this project.

3. There are two changes being made to the B`inaryHeap` data structure: it's being rewritten so that it's a *maxHeap* instead of a *minHeap*, and it's being rewritten so that it compares VIP values for `Person` objects rather than integers. Which of those changes will you take on first? Which of those changes will be more challenging? A rookie mistake is to try to change everything all at once and then try to fix everything all at once. Instead, change one thing and make sure it works before moving on to changing something else.

4. If needed, take a look at the sample output to for ideas on how you might treat the output for this assignment.

EXTENSIONS

1. Up to this point we've manipulated heaps that don't have multiple `int`s of the same value. Does our algorithm need to be modified to work with duplicate values? Perform experiments to identify any issues. If you identify any, how might those issues be solved?

2. The `build_heap(list_of_keys)` strategy we used to create a heap from a list involved copying the list into the heap and then percolating down the items in only the first half of the list. This was a little tricky to understand, but is ostensibly faster than creating the heap by adding one key at a time.

Prove it. Using an integer-based minHeap or maxHeap, create a significantly large list of random keys (one million values, perhaps?) and time how long it takes to build a heap using each of the two strategies.

QUESTIONS FOR YOU TO CONSIDER (NOT HAND IN)

1. The functional equivalent of `VIP_status` is ubiquitous in society:
   1. Cellphone service plans with greater amounts of data
   2. More Instagram followers = great social status
   3. Cable/satellite TV services with greater numbers of channels
   4. More handsome/beautiful = faster entry into a dance club
   5. "FastPass" services on the freeway, at Disneyland, at Six Flags Magic Mountain

   Which of these policies are acceptable to you, and which are problematic from a fairness or equity perspective?

SAMPLE OUTPUT

```
rwhite@MotteRouge$ python max_heap_demo.py

Welcome to the BinaryHeap of the Stars!
Where you can get in line, but you may *never*
get to the top of the heap.

BinaryHeap[ Carrie:100 , Schmoke:23 , Zetlian:12 , Patty:10 , Dexter:17 ]
Adding a star
BinaryHeap[ Carrie:100 , Schmoke:23 , Tsai:30 , Patty:10 , Dexter:17 , Zetlian:12 ]
Most Important VIP (findMax()): Carrie:100
Admitting VIP (delMax()): Carrie:100
Remaining queue: BinaryHeap[ Tsai:30 , Schmoke:23 , Zetlian:12 , Patty:10 , Dexter:17 ]
Admitting remaining VIPs in order:
Tsai:30
Schmoke:23
Dexter:17
Zetlian:12
Patty:10
VIPs remaining: 0
```