C343 Project 5 - Compression using Priority Queues

Due 11:59pm, October 31, 2014

1 Assignment Description

This week we will see an application of the greedy strategy in the problem of compressing strings. Suppose the string to be compressed contains 26 different characters from the English alphabet. We need to give each character a code. One example of such a code is ASCII. If we used a fixed length code like ASCII, each character can be represented using 5 bits and the total number of bits required is *string length * 5*. Instead we can take advantage of the fact that not all characters occur with the same frequency. A variable length code saves bits by assigning a smaller code to the most frequently occurring characters. For example, we might use just a single bit for the most frequently occurring character and more than 5 bits for the least frequent character. In this assignment, we use the variable length code called *Huffman Code*. The Huffman method builds up a tree whose leaves represent the characters in the string along with their frequencies. The code for a character is the path from the root to the leaf that represents that character. Huffman trees can be built using priority queues which internally use min-heaps.

2 Your Task

We have given you the Huffman Tree building code and the `PriorityQueue` class. The `PriorityQueue` class uses a heap internally and we need you to implement the following methods in the `heap` class.

1. `minimum` - returns the minimum element in the heap
2. `extract_min` - removes and returns the element with the smallest key from the heap
3. `insert(element)` - inserts `element` into the heap
4. `min_heapify(i)` - move the element at position at `i` to the correct position
5. `build_min_heap` - rearrange `data[]` attribute so that it represents the min-heap

3 Running Your Code

- To run, execute `python compress.py` This file runs a the exec huff method on a simple dictionary that maps each character to its frequrency in a string.
- To test You can run the provides test cases with `python compress.py test`.

4 Deliverables

Your forked repo's hw7 folder should contain all the files from the zip. These are the ones you need to modify:

- `heap.py` - containing your solution and tests
- `README.md` - where you explain your code.
- Hours - record the number of hours you spent on writing and debuging your code and, separately, the number of hours you spent on the textbook exercises. Put your answers in the `README.md`.

After that you can issue a pull request.

5 Testing

There are tests in `tests.txt` file which you can run by executing *run compress.py test*. We also encourage you to write your own unit tests. A good way to do this is to use **assert** statements. So if your function must return `True` for given inputs, the test case looks like

```
assert f(i) == True
```

You can then put all your tests inside an `if` statement.

```
if __name__ == "__main__":
  # my unit tests
```

The tests will be executed when you execute this python script but not when you include this file into another script and run that.