CSCA48 Exercise 5

Due: February 23, 2018. 5:00 pm

In this exercise you will be practicing working with heaps. The main goal of this exercise is to optimize the first function of exercise 3. So read exercise 3 again, if you've forgotten what was it about.

Marzieh and Nick realised that it takes n^2 operations to create a sorted list of students if the number of students in their class is n. But, if they use a heap instead of a double linked list, they can reduce the number of operations to $log_2{}^n$, which is much better than previous method[1]. So this time, they both created a heap of their students' surname, where the highest priority item has a minimum key[2].

Your job this week is to write two functions called `merge_heap` and `first_and_last`.

1. `merge_heap` takes two heaps as its input parameters and returns a heap that contains all the names in both Marzieh's and Nick's class. To write this code you need to download and import class heap from https://mathlab.utsc.utoronto.ca/courses/csca48/Codes/week6_heap.py [3]. In this exercise the efficiency of running the code is important for us. So choose the heap's methods wisely to make sure this function works with minimum number of operations possible.
2. `first_and_last` takes a heap as an input parameter and returns a tuple containing the surname of the first and last student in alphabetical order.

What we know/ don't know

- We don't know how many students registered for each class therefore you should expect that the number of nodes in the heaps are not the same.
- We know that there a few individuals that have the same surname. None of them should be excluded.
- The only ADT available to solve this exercise is heaps.

---

[1] Read lecture notes at https://mathlab.utsc.utoronto.ca/courses/csca48/Lectures/week6_BT_Heap.pdf if you're not sure how this number of operations is calculated.

[2] Keys here are students' surname and for simplicity, we just store the keys.

[3] Even students in my class should look at this url again, because I have added some auxiliary functions, which are useful for this exercise.