Doga Uras

300062160

CSI 2510- Devoir 2

Report

You must submit a short report (suggested max 2 pages) in file report.pdf that contains:

• A brief description of your classes and methods.

• A detailed description of the removeMin/removeMax algorithm as you implemented it.

• Optional: any additional information to be seen by the marker.

The HeapPriorityQueue class implements the PriorityQueue interface that has the insert(K key, V value) method which inserts a new Entry object having key and value as its parameters. Here, the key point is to check whether the buffer is empty or not. In case the buffer is empty, the new entry is placed in the buffer. If not, then the new Entry gets associated to the buffer. Depending on the values of the buffer and the new Entry; the one with smaller key gets placed in the minheap while the one with bigger key gets placed in maxheap. Finally upheap functions are called on both heaps to re-establish the heap conditions.

The second function to be implemented in HeapPriorityQueue class was removeMin() which removes and returns an entry with minimal key. We know that the minimum key is always found in the root of the minheap (if the buffer is empty). Here, the important part is to figure out how to remove its associate from the maxheap and this depends on whether the buffer is empty or not:

- If the buffer is empty, then the minimum key has to be the root of minheap. In order to remove the root, the entry found in tail is moved to the root. Now, removing its associate from maxheap: there are two possibilities, if the associate is found in tail then the tail is simply set to null. If not, then the associate and maxheap tail get swapped, then the maxheap tail is set to null. After that, the minheap tail is set to null and tail is decremented by one.
- If the buffer is not empty, then there's a possibility that the buffer contains a smaller value than that of the root of minheap. If the buffer key is smaller than the root of minheap, then buffer is simply set to null. If not, then there's a new association that needs to be established between buffer and the associate of the minimum key (found in maxheap). If the buffer is smaller than the associate, then the buffer is placed straight into the root of minheap. Else, the associate and the buffer exchange their values. Now, the entry in the buffer gets placed in the root of minheap. Finally, calling down heap on both heaps allows to re-establish the heaps.

The third function removeMax() is implemented in the exact same way as removeMin(), but this time it operates from the maxheap. Again, if the buffer is empty, the tail of maxheap is moved to its root and its associate from minheap is now stored in the buffer. If the buffer is not empty, then the buffer is either the max value in which case it is simply set to null, or the buffer gets associated to the associate of the max key (found in minheap) and they get placed in the heaps depending on their values. Finally the heap conditions are re-established by calling down heap on both heaps.

Up heap and down heap methods are also modified in order to be able to call the functions on both heaps (min heap and max heap). maxUpHeap does up heap on the maxheap while minUpHeap does down heap on the minheap. Same applies to the down heap method.