Vidulash Rajaratnam
8190398

REPORT

Detailed Description

removeMin algorithm: In this algorithm, we first check if the double-ended priority queue (DEPQ) is empty. This includes both heaps and the buffer. If it is empty, it will return null. If it is not empty, it will proceed. It saves the Entry value of the root of the minheap, which is the smallest in the minheap, as 'ret'. Then, it saves the Entry value for the root's associated value from the maxheap as 'ret_ref' along with its index as 'associate_index'.

If buffer was empty:

If the buffer is empty, it first removes the associate from the maxheap. If the associated value was the tail of the maxheap, it simply removes it (no restore required). If associated value was not the tail, it will remove it and then restore maxheap using downHeapMax on the maxheap. Once the associated value is removed, it will set the currently empty buffer as the saved value of the associate 'ret_ref' and make its index -1. Now, the root can be removed from the minheap and restored. If the min heap is only 1 element, it will simply make it empty. If the minheap size was larger than 1, it will swap root with tail, remove the tail, and downHeapMin the minheap to restore the minheap. The removed min value is returned.

If buffer was NOT empty:

- **If buffer was smaller than root**

If the buffer was not empty, it first compares the root of the minheap to the buffer. If buffer was smaller, creates a temporary value for buffer called 'temp' and saves it. Buffer is then cleared, and the temporary value, which is the min value, is returned.

- **If buffer was larger than root**

If the buffer was not empty and the root was smaller, it will swap root with tail, remove the tail, and downHeapMin the minheap to restore the minheap. Now the associated value in the maxheap will get associated, using setAssociate, with the current buffer, vice-versa. The index of the associated value is saved as well. The associated value is now removed from the maxheap. If the associated value was the tail of the maxheap, simply removes it. If associated value was not the tail, it will remove it and then restore maxheap using downHeapMax on the maxheap. Now the saved associated value 'ret_ref' is compared to the buffer. If it is smaller than the buffer, it gets inserted at the tail of the minheap and upheaped using upHeapMin to restore minheap. While buffer is added at the tail of the maxheap and upheaped using upHeapMax to restore maxheap. If it is larger than the buffer, it gets inserted at the tail of the maxheap and upheaped using upHeapMax to restore maxheap. While buffer is added at the tail of the minheap and upheaped using upHeapMin to restore maxheap. The removed min value is returned.

removeMax algorithm: In this algorithm, we first check if the double-ended priority queue (DEPQ) is empty. This includes both heaps and the buffer. If it is empty, it will return null. If it is not empty, it will proceed. It saves the Entry value of the root of the maxheap, which is the largest in the maxheap, as 'ret'. Then, it saves the Entry value for the root's associated value from the minheap as 'ret_ref' along with its index as 'associate_index'.

If buffer was empty:

If the buffer is empty, it first removes the associate from the minheap. If the associated value was the tail of the minheap, it simply removes it (no restore required). If associated value was not the tail, it will remove it and then restore minheap using downHeapMin on the minheap. Once the associated value is removed, it will set the currently empty buffer as the saved value of the associate 'ret_ref' and make its index -1. Now, the root can be removed from the maxheap and restored. If the maxheap is only 1 element, it will simply make it empty. If the maxheap size was larger than 1, it will swap root with tail, remove the tail, and downHeapMax the maxheap to restore the maxheap. The removed max value is returned.

If buffer was NOT empty:

- **If buffer was larger than root**

If the buffer was not empty, it first compares the root 'ret' of the maxheap to the buffer. If buffer was larger, creates a temporary value for buffer called 'temp' and saves it. Buffer is then cleared, and the temporary value, which is the max value, is returned.

- **If buffer was smaller than root**

If the buffer was not empty and the root was larger, it will swap root with tail, remove the tail, and downHeapMax the maxheap to restore the maxheap. Now the associated value in the minheap will get associated, using setAssociate, with the current buffer, vice-versa. The index of the associated value is saved as well. The associated value is now removed from the minheap. If the associated value was the tail of the minheap, simply removes it. If associated value was not the tail, it will remove it and then restore minheap using downHeapMin on the minheap. Now the saved associated value 'ret_ref' is compared to the buffer. If it is smaller than the buffer, it gets inserted at the tail of the minheap and upheaped using upHeapMin to restore minheap. While buffer is added at the tail of the maxheap and upheaped using upHeapMax to restore maxheap. If 're_ref" is larger than the buffer, it gets inserted at the tail of the maxheap and upheaped using upHeapMax to restore maxheap. While buffer is added at the tail of the minheap and upheaped using upHeapMin to restore maxheap. The removed max value is returned.

Description of Classes/Methods:

1. **Class Entry:** Type of element used to implement min and max heap used for DEPQ. Has a key integer value that is used to organize heaps, has a value for identification, an integer index for positioning and another Entry associate for the reference element in the other heap.
2. **Interface PriorityQueue:** Interface used by the heap priority queue abstract data type.
3. **Class HeapPriorityQueueTest:** Used as a test class to test HeapPriorityQueue. Forms randomized DEPQ and removes min and max key elements.
4. **Class HeapPriorityQueue implements PriorityQueue:**
    a. **Instance variables:**
        - **Entry[] minheap:** The minheap itself in array form containing Entry elements
        - **Entry[] maxheap:** The maxheap itself in array form containing Entry elements
        - **Int minTail:** Index of the last element in the minheap
        - **Int maxTail:** Index of the last element in the maxheap
        - **Entry buffer:** The buffer
    b. **Methods:**
        - **Public HeapPriorityQueue(int size):** Constructor that will initialize min and max heaps with size amount of elements. Will initialize the buffer as empty. And initialize min and max tail as -1.
        - **Public int size():** Returns the number of items in priority queue, including the buffer.
        - **Public Boolean isEmpty():** Returns true if both min and max heap and buffer are empty. Else it will return false.
        - **Public Entry insert(K value, V value):** Inserts a key-value pair of K and V as Entry in double-ended priority queue. If buffer is empty, inserts it there. Otherwise, associates' buffer with new Entry and inserts lowest value in minheap and highest value in maxheap.
        - **Public Entry min():** If empty, returns null. If buffer is not empty, compares it first with root of minheap to see which one is smaller. Returns but doesn't remove the smallest value with the minimal key.
        - **Public Entry max():** same as min() but for maximum key.
        - **Private void upHeapMax(int location):** Algorithm to place element after insertion at the tail, specifically in the maxheap.
        - **Private void upHeapMin(int location):** same as upHeapMax but for minheap.
        - **Private void downHeapMax(int location):** Algorithm to place element after removal of the root and tail element is placed at the root, specifically used in maxheap.
        - **Private void downHeapMin(int location):** same as downHeapMax but for minheap.
        - **Private int parent(int location):** Finds the parent of the given location. If location at root, returns root.
        - **Private swapMax(int location1, int location2):** Swaps element at location1 with element at location2, specifically in maxheap.
        - **Private swapMin(int location1, int location2):** Same as swapMax() but for minheap.
        - **Public void print():** Prints information on DEPQ size, minHeap/maxHeap size along with all the elements in the heaps.