Task Design Discussion

There are two main scenarios which motivates two different implementations.

Scenario 1

The number of appearance of new elements in the stream is not large. Consequently, most of the time we update the count of the already seen elements in the stream.

Solution :

Maintain a hashTable where the key is the String representation of the element, and the value is Integer counting the number of appearance of the its key in the stream.

Since the task description did not specify the type of elements in the stream, a string is more convenient. Moreover, we restrict the number of entries in the HashTable to “k”, a given parameter.

The memory complexity in this case is O(k). Each search, modify, remove operation is constant time. Though, each new element appears after the first “k” distinct elements requires O(k) time to retrieve the entry with the least count, and replace its key with the new element.

Scenario 2

The number of appearance of new elements in the stream is **large**. Consequently, most of the time we search for the item with minimum count, to be replaced with the new element. Relying on HashTable in this case leads to more Linear processing time searching for the minimum counted item.

To avoid such overhead, I propose to use MIN HEAP data structure. The Java implementation is PriorityQueue. Using MIN HEAP leads to Constant O (1) time for replacing the minimum counted item with the new one.

The downside of HEAP implementation is that each update of already seen item requires two linear iterations, one to **find** the item and another one to **remove** it. Afterwards, we update the count and insert it again into the HEAP. Modifying the items inside HEAP without re-insertion lead to unexpected behavior, since the HEAP call a MinHeapify procedure on each insertion operation to guarantee the head of the HEAP is the minimum element. Such downside can be reduced with a custom implementation, which remove the item and retrieve at the same time.

Since PriorityQueue is a generic class, I have implemented a Pair type, to maintains each item as a string key, and its value as the count. A custom Comparator relies on the Value only is used when constructing the PriorityQueue to force the sorting on the Count, and that the head of the Queue is the item with minimum count at any moment of time. Also I have overridden   .equals() method because each remove or contains operation of a custom type, relies on this method for finding the target item.

To fasten the processing, a HashSet is used to check for the existence of an item into the HEAP. Hashing is constant time, while searching on the queue is Linear. This advance in the processing time comes in exchange to additional space complexity of O(k) for the HashSet.