CS 130A
Assignment II - The Hottest of Them All

Assigned: October 18^{th}, 2018
Due On Demo Day: November 2^{nd}, 2018

PLEASE NOTE:

- Solutions have to be your own.
- No collaboration or cooperation among students is permitted.
- 5% of the points will be deducted for each day the assignment was late, with a maximum of 4 days
- Requests for a regrade must be submitted within seven days from the day when we return the assignment.

1 Introduction

The *heavy hitter* or *top-k* problem is a famous problem in computer science. Typically used in settings where a stream of data (sensor data, click stream, advertisment requests, etc) is being analyzed to detect the most popular items. An easy solution is to keep a counter for each item in the stream, and increment the counter every time the corresponding item is received. Unfortunately, often the domain of items is too large to maintain a counter for each element, eg, the domain might be all IP addresses. In this case, we typically maintain a smaller *finite* set of counters. So, for each new item, we create a counter which correspond to that item. The counter is incremented every time this item is encountered. This works fine until the set of counters is full. We need to eliminate one counter and add a new one. Usually, the element with the smallest count is deleted. Different strategies have been proposed regarding the initialization of the new counter. One very successful strategy is to initialize the new counter with the value of the deleted (smallest count) counter. Of course the resulting top-k elements are an approximation of the real top-k, but it has been shown to be quite accurate for realistic distributions. In this assignment, we will use this strategy.

2 Basic Data Structures

In this assignment, you will implement a simple version of the heavy hitter problem, where we would like you to find the most popular words in a document. You will be given a .txt file, which is an article containing words. You will read the words one by one from the file and at the end identify an approximation of the most popular, ie, most frequent, 15 words. Given that we often need to retrieve (and delete) the word with the smallest frequency, a min heap is a natural choice. However, finding an item in a min heap is not easy. Therefore, you need an additional data structure to easily retrieve each element in the heap. For this, use a Hash table.

A binary min heap is a complete binary tree which satisfies the following min heap ordering invariant.

- ***the min heap invariant***: the value of each node is greater than or equal to the value of its parent, with the minimum-value element at the root.

A min heap can be uniquely represented by storing its ***level order traversal*** in an array as shown in Figure 1. Consider the kth element of the array, its left child is located at index 2*k,its right child is located at index 2*k+1 and its parent is located at index k/2.

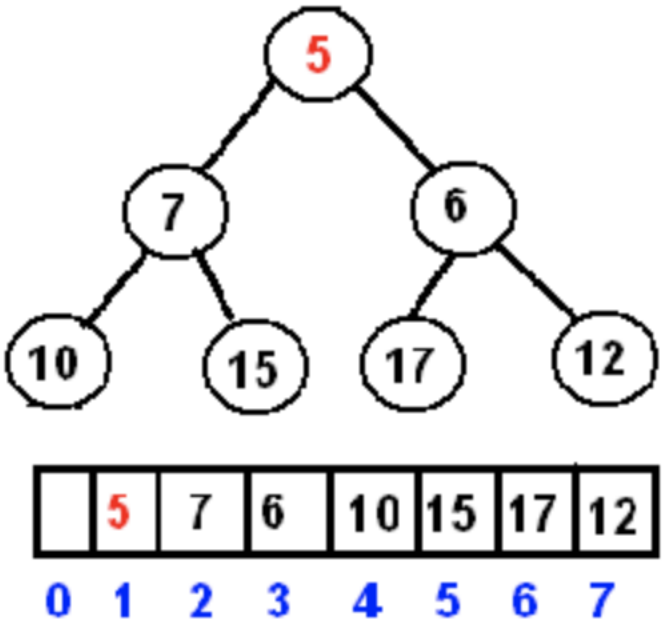

Figure 1: min heap

In this assignment, you will implement a ***min heap*** using an ***array object*** where each item in the heap contains the frequency of a word.

In order to keep track of the words corresponding to the frequencies in the min heap, you will implement a ***hash table***. Each entry in the hash table contains the word hashed to that location as well as a pointer to the corresponding entry in the min heap (since the

min-heap is implemented using an array, this is simply an index in the array). We leave it to you to decide whether to use a *chaining* or a *probing* hash table, as well as the hash function and the details of collision handling. However, since the minimum word in the min heap will be deleted and replaced, you need to support deletion in the hash table (as well as insert). Every time a word is read, it is looked up in the hash table, there are three cases to consider:

1. If the string already exists, increment its frequency by **one** in the min heap and percolate down to the correct place in the min heap while updating all the corresponding pointers in hash table.

2. If the word does not exist in the hash table **and the min heap is not full**, the new word is inserted into the min heap and is initialized to a frequency of **one**. Furthermore, the word is inserted into the hash table with a pointer to the corresponding frequency in the min heap (the root in this case).

3. If the word does not exist in the hash table **and the min heap is full**, retrieve the existing word with the minimum frequency, delete it from the hash table and replace it with the new word while keeping the frequency as before. The new word is also inserted in the hash table with a pointer to the corresponding frequency in the min heap.

NOTE

- Your algorithm should be case insensitive, meaning that, for example, "he" and "He" should be treated as the same string.

- We only care about strings that are words in the file, meaning that, blank space and commas and etc, should not be considered as strings.

- The algorithm will give an approximation of the most frequent 15 strings.

- The size of the heap(array) should always be 16 because we keep the index 0 empty.

- Figure 2 shows a high level sketch of the two data structures used in this assignment and how the hash table needs to have pointers to the corresponding entries in the heap.

3 Implementation details

As a part of this homework, you will implement 4 functions as explained below:

- *Insert:*
  This function will take a string as an input.

  - If the newly inserted string already exists in the hashtable, then first locate the position of the string in the min heap using hashtable, then update the frequency of the string in the min heap, percolate the element to its correct place in the array, and lastly update the hashtable to point to the updated position.

  - If the newly inserted string does not exist in the hashtable, check if the minheap is full or not. If it is full, then simply replace the root entry of the heap with the newly inserted word and keep the frequency, and then update the hashtable (delete the old word and insert the new word in the correct place using the hash function). If the minheap is not full, then insert the string to the heap, and then update the hashtable. (This is essentially achieved by implementing the *ReplaceMin* function as explained below.)

- *ReplaceMin:* This function will be called when the newly inserted string does not exist in the hashtable and the minheap is full. Replace the first element of the array (index one of the array), which has the lowest frequency, with the newly inserted string and update the hashtable (i.e, you have to locate the string in the hashtable, and then delete the entry, and use the hash function to place the newly inserted word in the correct place).

- *PrintHeap:* This function will print out the most frequent 15 words associated with their corresponding frequencies.

- *PrintHashTable:* This function will print out the current hash table.

3.1 Program Flow

NOTE: We do not want any front end UI for this project. Your project will be run on the terminal and the input/output for the demo will use stdio. The file name will be provided as an input to your program. After running your program, we will ask you to call the *PrintHeap* function, which will print out the **15** most frequent strings associated with its corresponding frequencies, and *PrintHashTable* function, which will print out the whole hash table. And then we will interact with your program (i.e. we will let you call insert(tree)), and then you will be asked to call *PrintHeap* and *PrintHashTable*. This process might be repeated multiple times during the demo.

3.2 Extra Credit and Sanity Check

Given that there are about 250,000 words in English, it is not so unreasonable to maintain an array of size 250,000, where each entry maintains the frequency of the corresponding word in the file you are analyzing. As extra credit, you can try and figure out a way to maintain the frequency of all words in the given file and then retrieve the most frequent 15. Compare them to what your approximate fixed size min heap solution gives.

4 Demo

We will have a short demo for each project. It will be on **November 2nd, 2018** in CSIL. Time details will be announced later. Please be ready with the working program at the time of your demo.

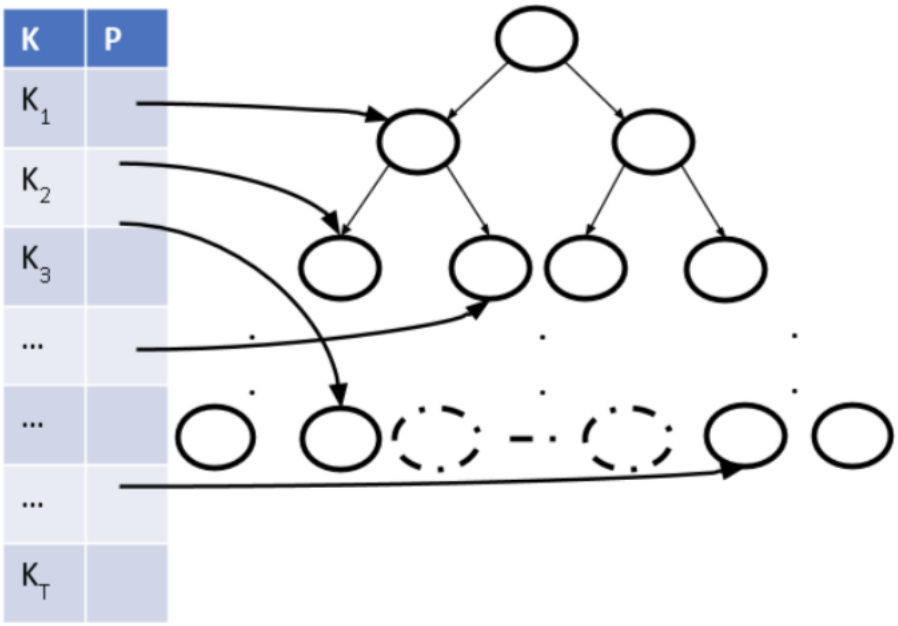

Figure 2: example