CS358: Applied Algorithms

Homework 4: Streaming (due 10/19/2024 10PM)

Instructor: Sam McCauley

Instructions

All submissions are to be done through github. This process is detailed in the handout "Handing In Assignments" on the course website. Answers to the questions below should be submitted by editing this document. All places where you are expected to fill in solution are marked in comments with "FILL IN."

Please contact me at srm2@cs.williams.edu if you have any questions or find any problems with the assignment materials.

Problem Description

In this problem, we will be analyzing a very long novel: "In Search of Lost Time," by Marcel Proust.[1] This novel contains about a million words (including duplicates), and the text file given for this assignment is about 7MB. Nonetheless, we will be using very small streaming data structures to analyze this file with just a single pass over the data—the first using a handful of kilobytes of space, the second using just 32 bytes.

In this assignment, you will be building two data structures.

First, you will build a Count-Min Sketch data structure. All words in "In Search of Lost Time" will be inserted into the Count-Min Sketch. At the end, your data structure will be queried with some of the most common words in the novel: how many times does this word appear? The testing program will compare your output to the actual count of each word; your data structure should always overestimate the count, but give reasonably similar values.

Second, you will build a HyperLogLog data structure. Again, all words in "In Search of Lost Time" will be inserted into it. At the end, your data structure will be queried to find out approximately how many unique words occurred in the novel. HyperLogLog uses an incredibly small amount of space, so it is likely that your data structure will have some error. However, it should usually be reasonably close to the correct value.

INPUT: `test.out` is given three arguments. The first is a text document in ASCII format.[2] The second is a text document, where each line contains a word from the first text document, followed by a space, followed by the number of times that word appears in the

[1] This novel is, I understand, very popular and well-regarded. However, it was chosen for this class mostly because its copyright has expired.

[2] As with last time, this means there are no accented characters.

first document. The final argument is an integer denoting the number of unique words in the original text document.

To run your program on "In Search of Lost Time," you would use the following input:

```
./test.out proust.txt words.txt 36372
```

The following input may be useful for testing:

```
./test.out proustShort.txt wordsShort.txt 125
```

Output: This assignment is unique in this course in that a single answer is not usually marked as correct or incorrect.[3] Instead, the testing program will output, for each word in the second text file, the actual number of occurrences of the word in the text compared to the number output by your Count-Min Sketch. Furthermore, the testing program will output the number of unique words predicted by your HyperLogLog data structure, compared to the actual number of unique words.

Interpreting the output: For the large output, the CMS should generally answer most word queries within 1000 of the correct value. Almost all CMS answers should be within 1500 of the correct value. The HLL overall estimation of the number of words should almost always be between 25000 and 50000.

You should use several different seeds to check that your answers satisfy these bounds.

Functions: This assignment is, broadly, structured much like the last assignment. The functions for both data structures already exist, and you must fill them in. The code in `test.c` will perform the above tests using the functions you provide.

`cms.c` and `cms.h` contain the code for the Count-Min Sketch data structure. `hll.c` and `hll.h` contain the code for the HyperLogLog data structure.
Here is a list of the functions and how they are used:

- `void cms_instantiate(Cms* cms)`

  This function is called before any other calls to the cms functions. You can think of it like a constructor. It should set constants and allocate memory. `cms` is a pointer to a struct that I found useful (you can change this to not use a struct if you wish). You do not need to edit this function if you don't want to; the version in the assignment is the version I used.

- `void cms_insert(char* word, int length, Cms* cms)`

  This function inserts a new word (given by `word`) into the filter. `length` is the length of the word, and `cms` is a pointer to the Count-Min Sketch we want to insert to.

  `test.c` will insert each word in the first document into the Count-Min Sketch by calling this function. These inserts will all occur before any call to `filter_lookup`.

[3] This choice is due to two concerns. First, we're using randomness, so some error is to be expected. Second, these structures are fairly inconsistent: for example, a cuckoo filter will almost always have approximately the same false positive rate on a large dataset; a CMS or HLL may not.

- `int cms_lookup(char* word, int length, Cms* cms)`

  This function looks up a word (given by `word`) in the sketch pointed to by `cms`. It returns an estimate of how often `word` (which has length `length`) occurs in the first document.

- `void hll_instantiate(Hll* hll)`

  This function is called before any other calls to the hll functions. You can think of it like a constructor. It should set constants and allocate memory. `hll` is a pointer to a struct that I found useful[4] (you can change this to not use a struct if you wish). You do not need to edit this function if you don't want to; the version in the assignment is the version I used.

- `void hll_insert(char* word, int length, Hll* hll)`

  This function inserts a new word (given by `word`) into the HyperLogLog data structure `hll`. `length` is the length of the word, and `hll` is a pointer to the structure we want to insert to.

  `test.c` will insert each word in the first document into the HyperLogLog data structure by calling this function.

- `int hll_estimate(Hll* hll)`

  This function asks for an estimate of how many unique words have been inserted into `hll`.

Count-Min Sketch Parameters: Your CMS should have 4 rows, each of 300 entries. Each entry should be of 32 bits.[5]

HyperLogLog Parameters: Your HyperLogLog data structure should keep track of 32 counters, each of length 8 bits. For 32 counters, the bias constant is .697. (The bias constants are included in `hll.h`.)

Questions

Code (50 points). Implement a Count-Min Sketch and a HyperLogLog counter, each as described above. You do not need to describe your implementation.

Problem 1 (20 points)**.** Let X be a positive random variable (i.e. X only takes on values that are ≥ 0). For simplicity, assume that X only takes on integer values (i.e. the probability that X is not an integer is 0). Show that

$$\Pr\left[X \geq e \cdot \mathrm{E}[X]\right] \leq 1/e.$$

[4] This is unnecessary here, unlike in the CMS and cuckoo filter. You could easily have the seed as a global constant and just pass around the table of counters rather than storing this struct.

[5] This is wasteful! Unfortunately, 16 bit integers are JUST small enough to barely overflow on this data. Using 18 or 19 bit entries would almost certainly be ideal, but are much more difficult to work with.

Hint: Use the definition of expectation (and the assumption that X is always positive). It may also be useful to write $\Pr[X \geq e \cdot \mathrm{E}[X]] = \sum_{i=e\cdot\mathrm{E}[X]}^{\infty} \Pr[X = i]$.

Solution.

Problem 2 (10 points)**.** Let's say I create a HyperLogLog structure H_1 for a stream $a_1, \ldots a_n$ and a second HyperLogLog structure H_2 for a stream $b_1, \ldots b_n$. Assume that all a_i and b_i are in the same universe U, and assume that identical parameters and hash functions are used to create H_1 and H_2.

Describe how to use H_1 and H_2 to create a new HyperLogLog structure H_3 that can estimate the number of unique items in the concatenation of the two streams: $a_1, \ldots a_n, b_1, \ldots, b_n$. (You should build H_3 using only H_1 and H_2, without seeing the stream again.)

Solution.

Problem 3 (20 points)**.** The following short questions ask about different situations in which you can use a Count-Min Sketch. For each, say how many entries you would need in each row (in other words, what value of ε you would need asymptotically) to obtain the correct answer in a single row with constant probability. In each case, assume that there are N total insertions into the CMS.

For each explanation, please give a brief explanation ($\approx$ one sentence) as to why—and bear in mind that I am only expecting an asymptotic answer.

(a) A stream consists of two kinds of items: items of the first kind appear once each, and items of the second kind appear twice each. Let's consider a Count-Min Sketch data structure to determine if a given item appeared once or twice. How many entries would you need in each row of the CMS in order for it to provide the correct answer with constant probability?

Solution.

(b) In a stream of items, one appears the majority of the time (more than half the items are the same). Let's consider a Count-Min Sketch data structure to determine the majority item. How many entries would you need in each row of the CMS in order for it to provide the correct answer with constant probability?

Solution.