CS358: Applied Algorithms

Assignment 4: Streaming (due 4/23/2020 10PM EDT)

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 sam@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 overall estimation of the number of words should almost always be between 30000 and 43000.

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. (One could run the program multiple times and perform statistical tests, but that’s very much contrary to the spirit of these structures.) 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 Paremeters 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

Problem 1 (70 points)**.** Your code is worth 70 points. **You are not required to write about your implementation to get credit for this assignment; the 70 points are for code alone.** But, you may fill in some information below if there is something you want me to know about your implementation.

Solution.

[4]This is a bit wasteful, unlike 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.

Problem 2 (14 points)**.** Consider a situation where I have a stream of 1,001,000 elements. 1,000,000 of the elements $a_1, \dots a_{1000000}$ only appear once, but one element, q, appears 1000 times throughout the stream.

For each of the following use cases, give the appropriate parameters[6] for a correct Count Min Sketch with the smallest possible space. Be sure to answer:

- How many rows should I have? (`numRepetitions`)
- How many entries should I have in each row? (`numEntries`)
- How large should each entry be in bits?[7] (The size of each entry in `table`, in bits)

(a) After inserting all stream elements into my CMS, I query an element a_i, and I receive an answer o_i, and I query q, and receive an answer o_q.

How should I set the parameters of my Count Min Sketch so that at least 90% of the time, $o_i \le o_q$? That is to say, how do I set my parameters so that 90% of the time, the CMS accurately returns that q was more common than a_i?

(b) After inserting all stream elements into my CMS, I query $a_1, \dots a_{1000000}$ to obtain answers $o_1, \dots o_{1000000}$.

How should I set the parameters of my Count Min Sketch so that at least 90% of the time, $\max_i o_i \le o_q$? That is to say, how do I set my parameters so that 90% of the time, the CMS accurately returns that q was the most common out of all elements queried?

Solution.

Problem 3 (10 points)**.** Let's say I create a HyperLogLog structure H_1 for a stream $a_1, \dots a_n$ and a second HyperLogLog structure H_2 for a stream $b_1, \dots b_n$. Assume that all a_i and b_i are in the same universe U.

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: $a_1, \dots a_n, b_1, \dots, b_n$.

Solution.

Problem 4 (6 points)**.** Is the Count-Min Sketch useful in each of the following situations? (In many of these situations, it is not the *most* effective data structure. I am asking if it is able to give guarantees on the answer to each question.) **Please give a brief answer (≈ one sentence) as to why**.

(a) Can the Count Min Sketch be used to approximate the number of unique items in a stream (in other words, can the CMS be used to provide a similar estimate to what HLL provides)?

[6] Here I am asking about the parameters you should use for your *code*: number table entries, number of rows, etc. Determining ε and δ is likely an important part of your answer, but it is not the final solution.

[7] Please give an exact answer, even if `C` does not support variables of this size.

(b) 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. Can a Count Min Sketch determine if a given item appeared once or twice?

(c) In a stream of items, one appears the majority of the time (more than half the items are the same). Can the Count Min Sketch determine the majority item?

Solution.