Assignment 5: Bloom filters

Date: 13-03-2017
Deadline: 22-03-2017 23:59

Objectives

You must implement a hash function API, an array API and a program that uses a bloom filter to detect duplicates in its input.

Requirements

You must write a program *dups* which can process a very large number of lines on its standard input (containing possibly billions of input lines) and reproduces its input on its standard output without any duplicates. The program may drop a small fraction of the input which are not duplicates.

Your solution must use a Bloom filter to detect duplicate items.

The new aspect in this assignment is that there is not one single good answer: a part of your grade will be proportional to "how well" your program is able to reproduce the original input without duplicates.

Program arguments

Your program must accept multiple positional arguments on the command line:

- *k*, the number of hash functions to use;
- *n*, the size of the bloom filter.
- *ks*, a set of *k* parameters to be passed to initialize the *k* different hash functions. Each parameter can be a single value or a combination of multiple arguments of different types, separated in a format of your choosing.

Input / output definition

The input items will be provided as zero or more separate lines of text on the standard input. The input can be arbitrarily large: you cannot assume a maximum number of lines in the input. Each line of input will contain only printable characters, in particular there will be no nul character in the input, and will contain at most 1024 characters.

Your program must print out the *unique* input items as-is on its standard output, as soon as possible after they are read. A unique input item is an item *that has a low probability to have been read before* (since the program started). For example in the following input:

```
hello
world
hello
jane
```

There are 3 unique input items, and the corresponding expected output is:

```
hello
world
jane
```

Because when "hello" is encountered the 2nd time, it was seen before with a high probability (in this case, the probability is 1), so it is skipped.

When there are no more input lines available to read (if that ever happens), Your program must then:

1. print a last line containing the number of input items read so far and, separated by a single space, the number of output lines produced so far. This line should be printed to *stderr*;
2. terminate with exit code *EXIT_SUCCESS*.

For example with the input above the complete output would be:

```
hello
world
jane
4 3
```

Overall structure of the assignment

1. Implement a first version of the array API defined in `bitvec.h`. Your first version can be very simple, and use standard C arrays. This would be a correct implementation.
2. Implement the hash function API defined in `hash.h`, which requires you to define a *family* of k hash functions (from 0 to k-1). The quality of the Bloom filter you will use in step 3 is dependent on how different the k functions are from each other. *Note: Hash functions and Bloom filters will be covered in lecture on Thursday.*
3. Implement a Bloom filter based on your array and hash function API from the previous two steps. You may build the Bloom filter directly into `main.c` or create a separate file `bloom.c` with its own header API `bloom.h` (+0.5pt for separate files). *Be sure to add all additional files to the Makefile under all the relevant targets, including the bloom_submit.tar.gz target so the files will be included in your tarball.*
4. Implement the *dups* program to read the input from standard input and detect potential duplicates using your Bloom filter, printing the non-duplicate entries to standard output.
5. Find good values for the *ks* parameter of your *dups* program and store these, separated by spaces, in the file called `PARAMS`. Please note that you are free to determine the format (and even type) of these parameters. This does require you edit the behaviour of the `init_ks` function to match your format. You should document this, if you do so.
6. Iterate on step 2 (hash functions) and 5 to see if you can decrease your false positive rate (number of items your algorithm incorrectly decides are duplicates).
7. (Optionally) Improve your implementation of the array API, taking advantage of your knowledge that there are only two possible values at each position. Please note that you should try to assume as little as possible about the machine on which the code is used. Hint: https://en.wikipedia.org/wiki/Bit_array

Informatics only: While you are iterating in step 6, keep a journal of your results and your intermediate implementations to present in your PAV report.

Grading

Your grade starts from 0, and the following tests determine your grade:

- +0.5pt if you have submitted an archive in the right format with an `AUTHORS` file.
- +0.5pt if your source code builds without errors and you have modified `hash.c`, `main.c` or `bitvec.c` in any way.
- +1pt if your array API works properly.

- +2pt if your hash function API works properly, and the `k` functions return a different value when applied to an empty string as input. Additionally, you have provided at least 10 different initialization parameters in the `PARAMS` file to test your functions.
- +0.5pt if your Bloom filter is written in a separate files `bloom.c` and `bloom.h`.
- +2pt if your *dups* program works minimally:
    - the program never outputs the same item more than once.
    - given an input that contains *n* unique items, your program reproduces a non-zero fraction of *n* output items, regardless of the value of *n*.
- +0-3pt depending on the false positive rate of your implementation (more output items overall = better program).
- +1pt if your implementation of the array API is using less than one byte per position overall.