Homework 2 (programming assignment due 2/12)

DS-563 / CS-543 @ Boston University

Spring 2024

Before you start...

Collaboration policy: You may verbally collaborate on programming assignments, however, you must write your code and final report independently, i.e., without seeing other students' solutions. If you choose to collaborate on a problem, you are allowed to discuss it with **at most three** other students currently enrolled in the class.

The header of each assignment you submit must include the field "Collaborators:" with the names of the students with whom you have had discussions concerning your solutions. A failure to list collaborators may result in a credit deduction.

You may use external resources such as textbooks, lecture notes, and videos to supplement your general understanding of the course topics. You may use references such as books and online resources for well known facts. However, you must always cite the source.

You may **not** look up solutions to a programming assignment in the published literature or on the web. You may **not** share written work with anyone else. If you wish to make your code public (for instance, by making it publicly available on GitHub or GitLab), please wait till the end of the semester. (If you want to publish it earlier for whatever reason, please check with us first.)

Submitting: Your solution is to be submitted via Gradescope (entry code: 6G4V6G). In particular, you should submit a pdf with your project report and a zip file with your code (or an equivalent of these as long as it's allowed by Gradescope). Don't forget to provide information how to obtain your data sets. More information on this is provided below.

Late submission policy: No extensions. Submitting a solution one day late may result in a deduction of 10% of points.

Your task

1. Implement the CountMin sketch. You can use an arbitrary programming language. Make the implementation flexible so that running various experiments described later is not difficult. In particular, the hash functions (or a generator of hash functions) should be one of the parameters to the data structure.

2. Test your implementation on one or more simple artificial data sets, for which you know the expected answers, to make sure that it works correctly.

3. Implement universal hashing for strings of fixed size as discussed in the first discussion section (see the notes on Piazza).

4. Run your implementation on your favorite real–world data set that involves strings, using your implementation of universal hashing. Run experiments to find out how well it performs. How much does it deviate from the exact values with what probability? To answer this question, you could, for instance, plot a graph that depicts the observed frequency of overestimating by a given amount.

   Note: If you need help finding a suitable data set, see the "Resources" post on Piazza, which contains links to many public data sets.

5. When deploying the CountMin sketch, you can choose both the number of buckets (let us denote it as K) and the number of independent instances from which you select the minimum (let us denote it as N). Fix the total number of buckets, i.e., $N \times K$, to something sufficiently large. (Think of this as fixing the total space used by the data structure.)

   (a) How does the performance of the algorithm change for various trade-offs between N and K? Run experiments on either real–world data or artificial data sets (or both).

   (b) What does theory—or at least our analysis—predict? Does the observed behavior match the theory predictions? Or is it significantly better?

6. Recall that in class we used Markov's inequality to bound the probability that we overestimate the number of occurrences of any item by $2S/K$ or more, where S is the total number of items and K is the number of buckets. We said that it is at most $1/2$. Is this analysis tight (if K is large)? What is the worst you can make this probability for some distribution? Run experiments to see how your distribution behaves experimentally.

7. How much time (approximately) did you spend on this homework? Was it too easy/too hard?

Deliverables

Your final submission should include:

- Code:
  - Provide your implementation of the CountMin sketch.
  - Provide your implementation of the universal hashing.
  - Provide any auxiliary tools you develop for preprocessing data or generating artificial data sets.
  - Make your code readable (use reasonable variable and function names, etc.).
- Report:
  - It should include all important implementation details and discuss implementation decisions. In particular, describe what hash functions you use in each experiment (e.g., universal hashing you implemented, or cryptographically strong hash functions, or hash functions provided by your programming language). Describe briefly how to run your code with an example.

  - It should address all questions from the previous section. Use tables and/or graphs to display numerical results of your experiments.

- Data: Describe your data sets and make sure we can access them. Think of what someone would need to reproduce your results.
  - For any public data set that you use, provide a link to where this data set can be obtained from and include code used for its preprocesssing (if needed).
  - For any artificial data set, provide code that generates it (try to make it deterministic, by for instance, fixing the random seed it uses) or share a link (Dropbox, Google Drive, etc.) that can be used to access the data set. If you provide a link, make sure it does not require logging in.

Inspiration for the final project

You can use this programming assignment as an inspiration for your final project. One option is to implement an algorithm—especially an algorithm that was not covered in class or a non-trivial extension of an algorithm covered in class—and run interesting experiments. A call for final project proposals, which we will publish soon, will list more suggestions and ideas.