COMP90056 Stream Computing and Applications
Assignment B
Second (Spring) Semester 2019

Posted on LMS: Monday, 30 September 2019
Due: Monday, 21 October 2019 [07:30]

Important: Each student must submit their own code and report, written individually.

This Assignment contributes 15% towards your total mark for this subject. As a reminder, there is a hurdle on the non-final-examination component for this subject.

Part 1: Theory question

*This Part is worth **2** marks.*

In class, we showed that, for all x, the estimated count, $\hat{c}(x)$, of the frequency of x, f_x, in the Misra-Gries summary, satisfies $f_x - m/k \leq \hat{c}(x) \leq f_x$. In fact, we can show that

$$f_x - \frac{m - \tau}{k} \leq \hat{c}(x) \leq f_x\,, \tag{1}$$

where τ is the sum of the counters of the tracked items.

In this question, you don't need to prove inequality (1), you can assume it to be true. Instead, we apply inequality (1) to show how to combine the outcomes of Misra-Gries on two separate streams.

Suppose we run the Misra-Gries algorithm, with parameter k (looking for items of relative frequency $> 1/k$), on two separate streams, σ_1 and σ_2. For each stream, we obtain a *candidate* set of frequent elements, respectively, S_1 and S_2. For each element x in S_1, denote the counter by $\hat{c}_1(x)$, and for each y in S_2 denote it by $\hat{c}_2(y)$.

From S_1 and S_2, we determine a candidate set for the combined stream, i.e., the stream comprising σ_1 followed by σ_2, via the following method:

- For each z in $S_1 \cup S_2$, let the counter $\hat{c}(z)$ be

$$\begin{cases} \hat{c}_1(z), & \text{if } z \text{ is in } S_1, \text{ but not } S_2; \\ \hat{c}_2(z), & \text{if } z \text{ is in } S_2, \text{ but not } S_1; \\ \hat{c}_1(z) + \hat{c}_2(z), & \text{if } z \text{ is in both } S_1 \text{ and } S_2. \end{cases}$$

- If there are more than $k-1$ items in $S_1 \cup S_2$:
  - Let a be the value of the k^{th} largest counter.
  - Subtract a from $\hat{c}(z)$ for each z in $S_1 \cup S_2$. Consequently at most $k-1$ items have positive counters.
- Return S, the items with positive counters.

Prove that for all $z \in S$ returned by this method, inequality (1) holds, for the combined stream, $\sigma_1 \circ \sigma_2$.

Part 2: ℓ_0-sampling and sparse recovery

*This Part is worth **13** marks.*

So far, you have some experience implementing reservoir sampling, which is an elegant technique to sample a family of k distinct item instances from a stream of items. If you focus on sampling just one item (i.e., $k = 1$) then this approach is an example of ℓ_1-sampling.

In this Assignment, we change a few aspects of the sampling scenario. First, we assume a stream comprising a sequence of $\langle$item, count-increment$\rangle$ updates, where the count increment might in fact be negative (in some sense, a *decrement*). Second, we focus on ℓ_0-sampling; here, every item with non-zero frequency has an equal chance of being sampled, and no item with frequency zero is sampled. That is, if T is the set of items x with $f_x \neq 0$, then the probability of sampling an item y should be:

$$\begin{cases} 1/|T| & \text{if } y \in T\,; \\ 0 & \text{if } y \notin T\,. \end{cases}$$

We sample *uniformly* from the items with non-zero frequency. For this Assignment, we offer the Cormode-Firmani paper (as well as our own teaching materials) as a blueprint for your implementations[1].

Details

Given a universe U and a stream $S = \langle (s_1, \triangle_1), \ldots, (s_m, \triangle_m) \rangle$, with $s_i \in U$, the frequency of item x is defined as:

$$f_x = \sum_{i:s_i=x} \triangle_i. \tag{2}$$

The Assignment is broken down into *four* component tasks.

(a) ℓ_0-sampling in an insert-only stream Suppose items are only *added* in the input stream, never deleted, that is, $\triangle_i > 0$ for all $i \in [m]$. Your task is to implement, test, and analyze an efficient ℓ_0-sampler for this context. Each item that occurs in the stream must be equally likely to be selected, regardless of how many times it occurs in the stream. *Hint:* A randomly chosen hash function will help! Completing this part correctly would earn you at most **3 marks**.

(b) 1-sparse recovery For this task, we seek to detect whether the stream has exactly one item with non-zero frequency, i.e., when the set T described above has cardinality one, $|T| = 1$. In this case, the 1-sparse exact recovery data structure should, with high probability, return the item (in T) and its frequency. As shown in the slides, Ganguly's approach achieves this in a strict turnstile stream.

Your task is to implement, test, and analyze a 1-sparse exact recovery system that reports that the input has:

[1] G. Cormode, D. Firmani. A unifying framework for ℓ_0-sampling algorithms. *Distributed and Parallel Databases* 32(3): 315-335 (2014).

- more than one item in T; or
- zero items in T; or
- a single item in T, reporting the item and its frequency.

Completing this part correctly for a strict turnstile stream ($f_x \geq 0$, for all x) would earn you at most **2 marks**; completing it correctly for a general stream would earn at most **3 marks**.

(c) s-sparse recovery Now suppose we wish to detect whether the stream comprises at most s items with non-zero frequency, i.e., $|T| \leq s$. Following Ganguly's approach (also mentioned in slides), the aim is to construct a data structure that, with high probability, reveals an s-sparse input. The sparsity parameter can be provided at the start of the input stream.

Your task is to implement, test, and analyze an exact s-sparse recovery system that reports that the input has:

- more than s items in T; or
- zero items in T; or
- $1 \leq |T| \leq s$, returning the item-frequency pairs.

Completing this part correctly for a strict turnstile stream ($f_x \geq 0$, for all x) would earn you at most **2 marks**; completing it correctly for a general stream would earn at most **3 marks**.

(d) ℓ_0-sampling in dynamic/general streams This task is the culmination of the project components. It is to implement, test, and analyze an ℓ_0-sampler for a general stream: allowing for arbitrary increments and decrements, and for negative frequencies. We recommend combining the techniques from (b) and (c), with scheme for generating efficiently a nested family of sets, to produce an ℓ_0-sampler for general stream. Completing this part correctly would earn you at most **3 marks**.

(e) The thirteenth mark In the Cormode and Firmani article, which is the backbone of this project, they describe in Section 2.5.3 that there are alternative solutions to ℓ_0-sampling, which involve pairwise independence. The task is to implement, test, and analyze a variant ℓ_0-sampler, which incorporates a different level of hash-function-family independence from part (d). **Recommendation:** Only attempt this if you feel very confident that you have scored highly on *all* the other parts of this Assignment.

Implementation However, your implementation can be in the programming language of your choice. We will not be testing code or allocating marks for program quality. If you prefer to use a language like C or C++, for greater control over memory allocation, you are encouraged to do so.

Preparing the Report

Submit a multi-page report (around 1500 words) that provides a rich evaluation of the ℓ_0-sampler and its primitives. It is your job to convince the reader of the effectiveness of your ℓ_0-sampling data structure on general/turnstile streams.

To assist in presenting your results in a clear and organised manner, we recommend breaking the report down into five sections: Introduction, Theory, Implementation, Experimental Set-up (including data sets), Results and Discussion.

Introduction Establish the aims and purpose of the report. Provide some context and include a use-case for the ℓ_0-sampler.

Theory Introduce the data structures that form the study. Offer some intuition behind the recovery structures and their use in the ℓ_0-sampling algorithm. Compare, in a theoretical sense, the complexity of any relevant sampling data structures. You could do the latter in the form of a table:

	ℓ_0 insert-only	ℓ_0 turnstile	ℓ_0 general	baseline
Memory				
Update time				
Query time				
Fail rate				
Closeness to uniformity				

A good theory section will help your discussion. For example, your experiments may 'reflect' what is stated in the complexity table.

Implementation Detail some of the key decisions you made in your implementation. Which programming language did you use and why? Provide a justification for your choice of hash function. We also highly recommend that you implement a *baseline* solution for each of your data structures: a space-hungry, but simple-to-implement solution.

Experimental Set-up Detail your data-collection process. Did you generate your own data? If so, how and why? Did you use actual-world data-sets? State the names and attributes of the data-sets used in the experiments. Describe the metrics and processes used for measuring memory, time and accuracy.

Results and Discussion Present the results to support your discussion points, for example, via plots and tables. *Discuss* the results. Do your results express any trade-offs[2]? Are your results consistent with what you stated in the theory section?

You may deviate from this template if you think a different structure suits your arguments and comparisons. Regardless, your report *must* be in the following format: A4 page size, minimum 11-point font, minimum 2cm margins, and must be a `pdf` file!

[2]Hint: they should!

Criteria

There are **13** marks available for Part 2 of this Assignment. The indicative marking criteria for this Part are:

- What is the standard of your testing regimen, as described in the report? [2 marks]
  - Have you tested your code at different scales (input sizes)?
  - Do your datasets vary in the distribution of items?
  - Do your datasets vary in the kinds of data they can handle?
  - Are you testing *corner*/tricky cases for your code?
- How have you demonstrated good design choices in line with the goals of stream computing? [2 marks]
  - Are your data structures compact?
  - Do they support fast update per stream input?
  - Do they support fast response to queries?
- Is the theory well understood and presented clearly? [2 marks]
  - Have you presented the ideas with consistent logic?
  - Have you comprehensively covered all aspects of your algorithms and data structures?
- Are the experimental results clearly presented and organized? [2 marks]
  - Can the experiments be replicated?
  - Have you used plots, graphics, tables appropriately? Don't just repeat what's in text: amplify and summarize it.
- Does the discussion align the results with the goals and purpose of the report? [2 marks]
  - Is there a systematic critical engagement and analysis of the design decisions and experimental results in line with the aims of streaming algorithms and data structures?
  - Were there relevant hypotheses and how were these reflected upon?
- Is the report well structured and written? [2 marks]
  - Are there appropriate paragraph and section headings?
  - Have all aspects of the task been covered?
  - Are arguments presented in a logical way and sequence?
- Does the report display some particular creativity or insight? [1 mark]
  - Is independent thinking demonstrated?
  - Is there an alternative approach to ℓ_0-sampling implemented and analyzed?

A submission with a reasonable quality report, that runs correctly on insert-only streams may be expected to earn at most **7** marks, notwithstanding the marks for each criterion above.

Expectations

We want you to **demonstrate** that you can implement and differences between the three data structures within a stream computing framework. This includes knowing how to test the implementation in a way that reveals these differences.

Submissions

You should lodge your submission for Assignment A via the LMS. *You must identify yourself in **each** of your source files and the report.* Poor-quality scans of solutions written or printed on paper will *not* be accepted. There are scanning facilities on campus, not to mention scanning apps for smartphones etc. Solutions generated directly on a computer are of course acceptable. Submit *three* files:

- A `part1.pdf` file containing your answer to the Part-1 theory question.
- A `report.pdf` file comprising your report for Part 2.
- A `sampler.zip` file containing all your source files for Part 2, but *not* including "standard" `.jar` files (refer to the section on Libraries).

Do not include the testing files, as these might be large. **REPEAT: DO NOT INCLUDE TESTING FILES!** It is very important, so that you can justify ownership of your work, that you detail your contributions in comments in your code, and in your report.

Administrative issues

When is late? What do I do if I am late? The due date and time are printed on the front of this document. The lateness policy is on the handout provided at the first lecture. As a reminder, the late penalty for non-exam assessment is *two* marks per day (or part thereof) overdue. Requests for extensions or adjustment must follow the University policy (the Melbourne School of Engineering "owns" this subject), including the requirement for appropriate evidence.

Late submissions should also be lodged via the LMS, but, as a courtesy, please also email Tony Wirth when you submit late. If you make both on-time and late submissions, please consult the subject coordinator as soon as possible to determine which submission will be assessed.

Individual work You are reminded that your submission for this Assignment is to be your own individual work. Students are expected to be familiar with and to observe the University's Academic Integrity policy `http://academicintegrity.unimelb.edu.au/`. For the purpose of ensuring academic integrity, *every* submission attempt by a student may be inspected, regardless of the number of attempts made.

Students who allow other students access to their work run the risk of also being penalized, even if they themselves are sole authors of the submission in question. **By submitting your work electronically, you are declaring that this is your own work.** Automated similarity checking software may be used to compare submissions.

You may re-use code provided by the teaching staff, and you may refer to resources on the Web or in published or similar sources. Indeed, you are encouraged to read beyond the standard teaching materials. However, *all* such sources *must* be cited fully and, apart from code provided by the teaching staff, you must *not* copy code.

Finally *Despite all these stern words, we are here to help!* There is information about getting help in this subject on the LMS pages. Frequently asked questions about the Assignment will be answered in the LMS discussion group.

William Holland and Tony Wirth, with assistance from the COMP90056 team.

September 30, 2019