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

Posted on LMS: Monday, 2 September 2019
Due: Monday, 16 September 2019 [07:30]

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

This Assignment contributes 10% 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 questions

*This Part is worth **1.5** marks.*

(a) [1.5 marks] **Bloom Filter variant**. The analysis of the Bloom filter in the slides assumes that the available hash functions are uniform and *fully independent*. It is known that a hash function with the latter properties requires $\mathcal{O}(n \log n)$ bits to store, making it larger in size than the Bloom filter. In a theoretical sense, the solution can be seen as "incorrect". That being said, we do have access to *other* hash functions that fall within the memory bound. This leads to the following Bloom filter variant. To store a subset S, with $|S| = m$, of a universe $U = [n]$, initialise a bitmap of width r and choose a *single* hash function drawn from a **2-universal** hash family. The update and output procedures remain the same. With what width r should we initialise the Bloom filter bitmap so that the false positive rate is no more than a parameter $\varepsilon > 0$?

Part 2: Count-min sketch and its variations

*This Part is worth **8.5** marks.*

The *aim* of this part of the Assignment is to perform an experimental evaluation of the count-min sketch and its variants. We focus on the *trade-offs* expressed by these variants of the scheme. Since this is a rather involved task, we have broken it up into several components, with marks available for partial progress.

Background

Formally, 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:

$$f_x = \sum_{i:s_i=x} \triangle_x. \tag{1}$$

The count-min sketch is a data structure that supports estimates of f_x, $\forall x \in U$, on turnstile streams. It provides the following guarantee: for parameters $\varepsilon, \delta > 0$, on query x, the count-min sketch returns $\hat{f}_x$ such that:

$$f_x \leq \hat{f}_x \leq f_x + \varepsilon F_1, \quad \text{with probability } (1 - \delta). \tag{2}$$

Pseudo-code is provided in Algorithm 1.

Algorithm 1: CountMinSketch

Require: $f_x \geq 0, \forall x \in [n]$

Procedure `initialise`(n, ε, δ)
- $w \leftarrow 2/\varepsilon^2$; $d \leftarrow \log 1/\delta$;
- initialise pairwise independent hash functions $h_1, \ldots, h_d$ on $[n] \rightarrow [w]$;
- create a $d \times w$ matrix of counters C initialised to 0;
- **return**;

Procedure `update`$(x, \triangle)$
- **for** $i \in [d]$ **do**
  - $C[i][h_i(x)] \leftarrow C[i][h_i(x)] + \triangle$;
- **return**;

Procedure `output`(x)
- **return** $\min\{C[i][h_i(x)] \mid i \in [d]\}$;

Algorithm 2: CMSConservativeUpdate

Require: $f_x \geq 0, \forall x \in [n]$

Procedure `update`$(x, \triangle)$
- $\hat{f}_x \leftarrow$ `output`(x); `// get the current estimate for` x
- **for** $i \in [d]$ **do**
  - $C[i][h_i(x)] \leftarrow \max\{C[i][h_i(x)], \hat{f}_x + \triangle\}$;
- **return**;

The tasks

You are required to complete the implementation of *two* variants of the count-min sketch. Most importantly, you need to analyse the variants and the trade-offs. We will provide a basic implementation of the count-min sketch in Java, and scaffolds of the two respective variants. Each variant expresses at least one kind of trade-off.

You must *test* your implementations in a *comparative* experimental environment and write a report that discusses the results. The report, rather than the code, is the **main** piece of assessment. We will also provide a scaffold for the report (see below), to give you more time to focus on the analysis and discussion. But first, we introduce the variants.

Count-min sketch variants

(a) conservative update The *conservative update* variant is designed to mitigate the effect of *collisions*. It only changes the `update` procedure: `initialise` and `output` remain the same. The pseudo-code is provided in Algorithm 2.

(b) Morris counters A count-min sketch that replaces ordinary frequency counts with Morris counters as its counting primitive.

Template We will provide you with code templates to get you started. This includes a full count-min sketch program in Java and scaffold code for both the conservative update and Morris counter variants. 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 brief report (around 1200 words) that provides a rich comparison of the three data structures (Count-min sketch and its two variants). 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 count-min sketch (and variants).

Theory Introduce the three data-structures that form the study. Offer some intuition behind the conservative update and Morris counter variants. Compare, in a theoretical sense, the *accuracy* of each data structure and provide a complexity analysis. You could do the latter in the form of a table:

	Standard	Conservative Update	Morris Counter
Memory			
Update time			
Query time			

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.

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[1]? Are your results consistent with what you stated in the theory section?

[1] Hint: they should!

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!

Criteria

There are **8.5** 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? [1.5 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? [1.5 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? [1.5 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? [1 mark]
  - 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? [1 mark]
  - 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? [1.5 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? [0.5 marks]
  - Is independent thinking demonstrated?

Expectations

We want you to **demonstrate** that you understand the 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 `cms.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 1, 2019