COMP90056 Stream Computing and Applications
Assignment 1 (15 marks) — Analysis of Distinct Count Algorithms
Start date: 09:00 Monday 7 September 2020
Due date: 17:00 Monday 21 September 2020

Background

Estimating the number of distinct elements of a stream is an important problem in Computer Science, having wide-ranging applicability. In this assignment you will implement and compare the performance of two algorithms (BJKST I and HyperLogLog), reporting on your findings and providing a recommendation on their suitability for varying applications.

Task 1: Implementation

There is no specified programming language for this assignment. You may use whichever programming language you wish.

Implement each of the algorithms and performance metrics below. The hash functions may be based on the Hash.java implementation from Tutorial 5.

- A baseline algorithm for the *distinct count* problem—this should *not* be probabilistic, but instead give a correct answer every time. You may use existing data structures (such as HashMap in Java).
- BJKST I algorithm as presented in Theorem 1 of [1] (also in the lecture notes):
  - Include a priority queue that tracks the t smallest hash values.
  - Include inputs ϵ and δ that give desired theoretical performance by running multiple copies of the algorithm and outputting the median value.
- The HyperLogLog algorithm as presented in Fig. 3 of [2].
  - It should be able to work with $m = 2^b$ substreams, where $b \in \{4, 5, \ldots, 16\}$.
  - The algorithm makes use of adjustments to the estimate E. Note that the small-range correction is the linear counting algorithm discussed in lectures. The large-range correction is made when estimates exceed 2^{32}, but there will not be a need to explore the algorithm in this range.
- A stream of integers as input for the algorithm where the range of values is user-defined. The more types of input streams used for algorithm testing the better.
- Define relative error E_{rel} of an estimate $\hat{F}_0$ compared to its true value F_0 as $|\hat{F}_0 - F_0|/F_0$.
- Compute the speed in which the *distinct count* problem is solved.
- Be able to calculate the memory usage of your implementations. If possible use a pre-existing tool (function) specific to your programming language that measures runtime consumption.

Make use of a random seed(s) for the data and algorithms so that you are able to apply different algorithms to the same input data and to ensure your results are reproducible.

Task 2: Report

Complete a short report analysing the two algorithms including the following points. Since you will be considering probabilistic algorithms, steps will need to be taken to make results more definitive. Plots should be well described and clear to visualise. Refer to Section 4 of [3] for a good example of analysis including plots to describe the performance of algorithms (your report is not expected to be as detailed).

- *Set-up*—A description of the experimental set-up to aid reproducibility. For example, include the CPU frequency, memory, operating system, programming language, compiler (if applicable). **[1 mark]**
- A description of steps taken to gain confidence in the implementations (i.e., testing) and ensuring a fair comparison between the algorithms. **[1 mark]**
- Verification that the hash function has uniformly distributed output values. **[1 mark]**

- An investigation of the effect of the small-range correction to the estimate in the HyperLogLog algorithm. Plot average absolute error vs exact distinct count (up to 10^8 or more) to compare these effects and comment on any differences observed. **[2 marks]**
- Histograms for the BJKST and HyperLogLog algorithms similar to those given in Figure 4 of [2] showing the distribution of estimates for values of $n = 10^4$ and 10^6 and for each case specify parameters guaranteeing an estimate within ±5% of the true value with probability 0.9. **[2 marks]**
- A plot of average relative error vs exact distinct count comparing the BJKST and HyperLogLog algorithms with a choice of parameters to enable a fair comparison. **[2 marks]**
- A plot of average memory consumption (kB) as a function of actual distinct count (up to 10^8) comparing the three algorithms. (If unable to compute memory usage for your program you may do a theoretical analysis for half marks.) Use a legend to combine plots having multiple input parameters. Point out where the algorithms differ and possible reasons for this. **[2 marks]**
- A plot of average accuracy vs runtime comparing the two algorithms. Use a legend to incorporate multiple input parameters. Point out where the algorithms differ and possible reasons for this. **[1 mark]**
- *Practical considerations*—point out any differences between practice and theory and possible explanations for the difference. **[1 mark]**
- *Conclusion and recommendation*—which would be your recommended algorithm for practical use? Indicate how your answer depends on the application (e.g., speed, accuracy or memory requirements). **[2 marks]**

Marking Scheme

The marks for each component in the report are as shown. The grading will be based on the clarity and rigour of your description or analysis for each part. The code will not be assessed but should be presented and documented well enough that others could reproduce the whole experimental study of your report and obtain similar results.

Submissions

You should lodge your submission for Assignment 1 via the LMS (i.e., Canvas). 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. Solutions generated directly on a computer are of course acceptable. Submit two files:

- A report.pdf file comprising your report for the experimental study.
- A code.zip file containing all your source files of the implementations for the experiments.

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 Matters

Late Work

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 Chaitanya Rao when you submit late. If you make both on-time and late submissions, please consult the subject instructor 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 penalised, 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! Frequently asked questions about the Assignment will be answered in the LMS discussion group.

References

[1] Bar-Yossef, Z., Jayram, T.S., Kumar, R., Sivakumar, D., and Trevisan, L. *Counting distinct elements in a data stream.* Proc. 6th International Workshop on Randomization and Approximation Techniques, pp. 1–10, 2002.

[2] Flajolet, P., Fusy, É., Gandouet, O., and Meunier, F. *Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm.* Discrete Mathematics and Theoretical Computer Science Proceedings, pp. 127–146, 2007.

[3] Luo, G., Wang, L., Yi, K., and Cormode, G. *Quantiles over data streams: experimental comparisons, new analyses, and further improvements.* The VLDB Journal vol. 25, pp. 449–472, 2016.

COMP90056 team
September 2020