DAT470/DIT065
Computational techniques for large-scale data

Assignment 5

Deadline: 2024-05-13 23:59

In this assignment, we will implement tabulation hashing, the HyperLogLog algorithm [1], and have a look at some of their properties.

Problem 1: Tabulation hashing (8 pts)

(a) Implement *tabulation hashing* such that you can hash unsigned 64-bit integer keys into 32-bit unsigned hash values. Use a 16-bit alphabet. Use the interface from the file `tabulation_hash.py`. Make sure that your implementation behaves predictably: constructing the function with the same random seed should always yield same results. Also, you *must* use bitwise operations; you are not allowed to convert the integers into strings or lists. (6 pts)

(b) Use the implementation you just crafted to hash the integers ~~1, 2, . . . , 999999999, 1000000000 (one billion or 10^9)~~ 1, 2, . . . , 999999, 10000000 (one million or 10^6). Plot a histogram of the hash values. (2 pts)

Problem 2: HyperLogLog (16 pts)

(a) Implement the function ρ using bitwise operations. (2 pts)

(b) Implement the HyperLogLog algorithm. Use the interface from the file `hyperloglog.py`. Note that you must not use any more memory for the registers than is allocated in the interface file (that is, for m registers, you may use at most m bytes). Use the tabulation hash you implemented in Problem 1 as your hash function h (pass the seed value to its constructor). For the hash function f, use a multiply-shift function with the following constant parameter $a = \mathrm{c863b1e7d63f37a3}_{16}$. (8 pts)

(c) Construct a sketch with $m = 1024$ registers, and feed into it the integers ~~1, 2, . . . , 10^9~~ 1, 2, . . . , 10^6. Report the estimate you get. (2 pts)

(d) Choose *two* other values of m beside $m = 1024$, and repeat 1000 times with different seeds for generating the data: construct ~~one million (10^6)~~ ten thousand (10^4) random integers, that is, estimate the cardinality of ~~$n = 10^6$~~ $n = 10^4$. Produce a table where you list on rows the present value of m, the average cardinality estimate, and the fraction of runs where the estimate was within $n(1 \pm \sigma)$ and within $n(1 \pm 2\sigma)$ where $\sigma = \frac{1.04}{\sqrt{m}}$.

That is, you should produce a table that has three rows (one for each value of m) and four columns. (4 pts)

Hints

- The experimental parts should be somewhat trivially parallelizable (e.g., with a process pool). As such, it might make sense to parallelize your code and run it on the cluster with a larger number of cores.
- Protip: Use a generator expression together with the `map` function of a multiprocessing `Pool` when performing evaluations; that way you do not need to create an explicit list of keys. Also, composition with the Python `map` and `range` may be useful.
- NumPy numerical types do not play along well with Python `int`s; you may need to explicitly convert integers and integer literals into unsigned NumPy integers, e.g., `np.uint32(x)`.
- Properly implemented HyperLogLog sketches can be trivially `merge`d which makes parallelization easy.
- Remember that m must be a power of two.

Returning your assignment

Return your assignment on Canvas. Your submission should consist of a report that answers all questions as PDF file (preferably typeset in LaTeX) called `assignment5.pdf`. In addition, you should provide the code you used in Problem 1a `assignment5_problem1.py` and Problems 2a and 2b in d `assignment5_problem2.py`. That is, you should produce the code for your implementation; you needn't produce your experiments. The code must match the interfaces of `tabulation_hash.py` and `hyperloglog.py`. Do *not* deviate from the requested filenames and do *not* produce the plots in these files; these files will be used for evaluating the quality of your implementations automatically.

References

[1] Philippe Flajolet et al. "Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm". In: *Proceedings of the 2007 Conference on Analysis of Algorithms (AofA 07)*. 2007.