Distinct elements using hashing

Martin Aumüller and Holger Dell

Hand-in date: November 6, 2019, 13:59

Format of Hand-in This assignment can be solved in groups of up to three people. (You can choose a new group if you want.) Please submit a .zip file on learnIT, containing:

1. your code in a subdirectory src/, including all necessary non-standard libraries and classes required to compile your code, and a script (e.g., `run.sh` or `run.py`) that we can use to re-run your experiments.
2. a single `report.pdf` file containing a report of at most 2 pages written text (12pt font, 1in margins, a4paper). The sections must be called "Exercise 1" to "Exercise 4". Plots, tables, and short segments of code do not count towards the page limit.
3. a file `authors.txt` containing a list of the group members and email addresses in the following format (and nothing else):

   ```
   Martin Aumüller <maau@itu.dk>
   Holger Dell <hold@itu.dk>
   ```

A note on Python Be aware that we're assuming integers to be 32-bit signed integers. If you plan on using Python, you have to manually convert hash values to signed integers to get the same output as reported by CodeJudge.

1 Hash Functions

The unsigned binary representation of an integer x is a sequence of bits $x_b, \ldots, x_1$ such that $x = \sum_{i=1}^{b} x_i 2^{i-1}$ holds. The number b of bits is typically 32 or 64 depending on the integer data type used. Let A be a matrix (or, if you prefer, a two-dimensional array) with k rows and b columns, and suppose each of the $k \cdot b$ entries of A is either 0 or 1. In the first exercise, we will use a specific matrix A to define a hash function h. The matrix will have dimensions $b = k = 32$. This means that each row of A consists of 32 bits; thus, each of the 32 rows of A can be represented by a 32-bit integer. In Exercise 1, we'll use the 32-bit integers in Appendix A as the 32 rows of A.

Now, we define a hash function h based on the matrix A: The function h takes a b-bit integer x and produces a k-bit integer $h(x)$. For each $i \in \{1, \ldots, k\}$, the ith bit of $h(x)$ is defined via:

$$h(x)_i = \sum_{j=1}^{b} A_{i,j} x_j \bmod 2$$

where mod 2 is the operator that computes the parity of the sum.

Exercise 1. Let us fix the values $b = k = 32$. Implement an efficient function that is given an integer x and returns $h(x)$. In your function, you should represent the matrix A as an array of 32-bit integers with values as specified in Appendix A. You should make use of bit-wise

operations and a built-in method that counts the number of 1s in the binary representation of an integer (e.g. `Integer.bitCount` in Java or the `popcnt` intrinsics in C++). Add your code for this function to your report (it can be quite short and elegant).

To test your solution to Exercise 1 on CodeJudge, your implementation should read a list of integers from `stdin` and output their hash values (as integers).

The data structure for counting distinct elements will make use of a certain function ρ, which we will empirically analyze in Exercise 2. More precisely, we consider the function $\rho\colon \{0,1\}^k \to \mathbb{N}$ defined by

$$\rho(y) = \min\{i \mid y_{k-i} = 1\}. \tag{1}$$

This function identifies the position of the first 1 in the binary representation of a number y, read from left to right. (Note that $\rho(y)$ is undefined for $y = 0$.) In the following exercise, we wonder what happens if we plug in the k-bit output of the hash function h from Exercise 1 into (1).

Exercise 2. Let h be the hash function from Exercise 1, and let ρ be the function from (1). Perform an experiment to determine the distribution of $\rho(h(x))$ for one million hash values with $x \in \{1, \ldots, 10^6\}$. Create a plot of the results and add it to your report. Discuss in how far the results of your experiment support the claim that the distribution of the hash values of ρ satisfies the relation $\Pr(\rho(y) = i) = 2^{-i}$ for all i from 1 to k for random $y \in \{0,1\}^k$.

2 HyperLogLog Counter

We consider the following algorithm[1] for estimating the number of distinct integers in a sequence of integers $y_1, \ldots, y_n$ using an array `M` of m integers, where m is a power of 2 with m much smaller than n. If we want to keep track of the exact number of distinct elements, we need to basically store all n integers in the sequence, but if we allow for an estimation error then we can get away with storing only m integers.

```
for i:=0 to m-1 do M[i]:=0
for i:=1 to n do
   j := f(y[i])
   M[j] := max(M[j],rho(h(y[i])))
end
Z := 1/(2^(-M[0])+...+2^(-M[m-1]))
V := |{i | M[i]=0}|
E := m*m*Z*0.7213/(1 + 1.079/m)
if (E < 2.5*m and V > 0) then E:= m * ln(m/V)
return E
```

The algorithm uses two distinct hash functions: f maps integers to numbers in $\{0, \ldots, m-1\}$ and h maps integers to k-bit numbers. The algorithm also makes use of the natural logarithm `ln`, as well as the function ρ (`rho`) defined in (1).

Exercise 3. Implement this algorithm with $m = 1024$ and $k = 32$. Test it on the input sequence $10^6, \ldots, 2 \cdot 10^6 - 1$ with 1 million distinct items: How many distinct elements are reported by your implementation? (Add this number to your report.)

While you are free to use different hash functions, it is suggested that you use h from Exercise 1 and as f use the following heuristic hash function:

[1] From *HyperLogLog: the analysis of a near-optimal cardinality estimation algorithm* by P. Flajolet, É. Fusy, O. Gandouet, and F. Meunier, 2008.

`f(x) = ((x*0xbc164501) & 0x7fe00000) >> 21.`

In any case, add the choices you made to the report.

To test on CodeJudge: Your implementation should read a list of integers from `stdin`. The first integer is special and should be stored, say, in a variable `threshold`. Your program should determine if the number of distinct integers is above or below `threshold`, and output either `above` or `below`. Doing this successfully requires an estimation error of less than about 10%. Using $m = 1024$ should suffice. Your program is space constrained, so using a naïve solution will not work on large instances.

Experimental Evaluation of the Estimation Error

As alluded to earlier, the space usage of the HyperLogLog counter influences its estimation error. In the last exercise, we want to experimentally find out how strong this influence is.

Exercise 4. Write an input generator that takes as input an integer n and a seed `seed` on standard input and outputs a list of n random 32-bit integers from a random integer generator using seed `seed`. Describe and run an experiment that gives a graphical representation of the connection of m and the estimation error. A recommended representation is a histogram plot over the distinct element count reported by the algorithm. Try at least 3 different values of m. Report in a table for each m and n the fraction of runs that reported a value in $n(1 \pm \sigma)$ and $n(1 \pm 2\sigma)$ for $\sigma = 1.04/\sqrt{m}$.

A List of 32-bit integers, hexadecimal format

```
0x21ae4036
0x32435171
0xac3338cf
0xea97b40c
0x0e504b22
0x9ff9a4ef
0x111d014d
0x934f3787
0x6cd079bf
0x69db5c31
0xdf3c28ed
0x40daf2ad
0x82a5891c
0x4659c7b0
0x73dc0ca8
0xdad3aca2
0x00c74c7e
0x9a2521e2
0xf38eb6aa
0x64711ab6
0x5823150a
0xd13a3a9a
0x30a5aa04
0x0fb9a1da
0xef785119
0xc9f0b067
0x1e7dde42
0xdda4a7b2
0x1a1c2640
0x297c0633
0x744edb48
```

```
0x19adce93
```