Hamming One
$\mathcal{O}(nl \log n)$

Dawid Kożykowski

2021/2022

Problem statement

Given n binary sequences ($0 < n \leq 10^5$), each sequence of lenght l ($0 < l \leq 1000$), find all pairs of sequences such that the Hamming distance between them equals 1.

Hamming distance between two binary sequences of equal lenght equals the number of positions on which those sequences differ, e.g.: distance between 1101 and 1100 equals 1, and between 0011 and 1100 equals 4.

GPU solution

Let's define constants:

$P_1 = 29$
$P_2 = 41$
$M_1 = 100000004917 \approx 10^{11}$
$M_2 = 99999981101 \approx 10^{11}$

By $a_i[j]$ we denote j-th element of i-th sequence, i.e. for $a_i = 10101$ we have $a_i[0] = 1$.

For each sequence a_i we calculate two numbers according to the following formulas:

$$hash1_i = \left(\sum_{j=0}^{l-1} (P_1)^j \cdot a_i[j] \right) \% M_1$$

$$hash2_i = \left(\sum_{j=0}^{l-1} (P_2)^j \cdot a_i[j] \right) \% M_2$$

In order to avoid exceeding the range of numeric types in the implementation, we convert the above formulas equivalently using the modulo congruence to formulas that can be calculated dynamically.

$hash1_i = 0$
$pot1 = P_1$

For $j = 0, 1, 2, \ldots, l-1$:
$hash1_i = (hash1_i + a_i[j] \cdot pot1)\%M1$
$pot1 = (pot1 \cdot P1)\$M1$

Similarly for $hash2_i$.

Then we put calculated values as tuples ($hash1_i$, $hash2_i$, i) in an array. We sort the array comparing first values of each tuple and in the case of conflicts - subsequent ones.

Note that for the sequences a_i and a_j, for which the Hamming distance is exactly 1, there is a position with index k, which is the only position on which those strings differ. In this case, hashes of those sequences also differ by the value calculated for the k -th element, i.e.

$hash1_i = (hash1_j + g(a_j[k]) \cdot (P_1)^k)\%M_1$, where:
$g(x) = 1 - 2x$, so $g()$ returns -1 for $x = 1$ and 1 for $x = 0$

Similarly for $hash2_i$.

Using the above conclusions , we can compute $hash1$ and $hash2$ in constant complexity for a sequence that differs at k th position. In order to find all sequences in distance equal to 1 from a_i, consider all $k = 0, 1, 2, ..., l-1$, i.e. calculate $hash1$ and $hash2$ for the sequence with the changed bit on a k-th position, then check if there is a tuple in the array containing the same value of $hash1$ and $hash2$ and list the matching pairs.

In order to efficiently check whether there is such a tuple, the *lowerbound* algorithm should be used .

Complexity

The first stage, i.e. string hashing, is performed in the complexity $\mathcal{O}(nl)$ (n strings, each of l length).

The next step, sorting the tuple array, is $\mathcal{O}(n\ logn)$ (sorting n tuples).

The last stage, that is searching for pairs using the *lowerbound* algorithm, is performed in the complexity $\mathcal{O}(nl\ logn)$ (for n sequences, considering l different positions of the replaced bit and finding the matching pair using the *lowerbound* algorithm with has the complexity of a single query equal to $\mathcal{O}(\log n)$).

Hence, the final algorithm complexity is $\mathcal{O}(nl \log n)$.

Correctness

An algorithm written in this way is a probabilistic algorithm - it is possible that a hash collision will occur and two different sequences will get the same $hash1$ and $hash2$ values, which the program deems sufficient grounds to believe that they are the same sequences. However, the probability of this happening is $\frac{n}{M_1 \cdot M_2} \approx 10^{-17}$, so it is so low that it can be considered to work properly in almost all situations.