Algorithm Design 21/22

Hands On 3 - Karp-Rabin Fingerprinting on strings

Federico Ramacciotti

1 Problem

Given a string $S \equiv S[0 \dots n-1]$, and two positions $0 \le i < j \le n-1$, the longest common extension $lce(i,j)$ is the length of the maximal run of matching characters from those positions, namely: if $S[i] \neq S[j]$ then $lce(i,j) = 0$; otherwise, $lce(i,j) = max\{l \ge 1 : S[i \dots i+l-1] = S[j \dots j+l-1]\}$. For example, if $S = abracadabra$, then $lce(1,2) = 0$, $lce(0,3) = 1$, and $lce(0,7) = 4$. Given S in advance for pre-processing, build a data structure for S based on the Karp-Rabin fingerprinting, in $O(n \log n)$ time, so that it supports subsequent online queries of the following two types:

- $lce(i,j)$: it computes the longest common extension at positions i and j in $O(\log n)$ time.
- $equal(i,j,l)$: it checks if $S[i \dots i+l-1] = S[j \dots j+l-1]$ in constant time.

Analyze the cost and the error probability. The space occupied by the data structure can be $O(n \log n)$ but it is possible to use $O(n)$ space.

[Note: in this exercise, a one-time pre-processing is performed, and then many online queries are to be answered on the fly.]

2 Solution

A data structure can be built as follows: using prefix sums in an array P, in $P[i]$ store the hash value between $[0,i]$ using Karp-Rabin's rolling hash, so `P[i]=KR_hash(S[0...i])`. We can compute the hash value of $P[i,j]$ by doing $P[j] - P[i]$ and normalize it by dividing it by Σ^i (Σ = length of the alphabet). In order to compute Σ^i in constant time we can use fast exponentiation, since we work modulo p prime, and store all the values for any $0 \le i \le n$ to have access to them in constant time.

The space used is $O(n)$ for both the P and the Σ arrays.

2.1 Equal

Define

$$equal(i,j,l) = \begin{cases} \text{True} & \text{if } \left(\frac{P[i+l-1]-P[i]}{\Sigma^i} = \frac{P[j+l-1]-P[j]}{\Sigma^i}\right) \mod p \\ \text{False} & \text{otherwise} \end{cases}$$

In order to divide by Σ^i we can find the multiplicative inverse modulo p, since it is prime. It computes the solution in constant time and with $Pr[error] = \frac{1}{n^c}$, given by the probability that two different numbers have the same Karp-Rabin's hash value.

2.2 Longest Common Extension

$lce(i,j)$ uses exponential search in parallel on the portions of the arrays that start from i and from j, comparing $\frac{P[i+k]-P[i]}{\Sigma^i}$ and $\frac{P[j+k]-P[j]}{\Sigma^i}$. As long as the hash values are equal (i.e. the substrings are equal w.h.p.), it searches to the right of that interval with $equal(i,j,k)$; when the two values are different, it searches in the latest interval found (e.g. in $k = 16$ the two hashes are equal and at $k = 32$ they're different: now the algorithm searches the interval $[16, 32]$ with binary search, following the same method). The algorithm stops when the interval is one character long.

This algorithm computes in $O(\log n)$ and, since it makes $\log n$ comparisons each with error $\frac{1}{n^c}$ (given by the probability that two different numbers have the same Karp-Rabin's hash value), the total error probability is $\frac{1}{n^c} \log n$.