UNIVERSITÀ DI PISA

Third hands-on: Karp-Rabin fingerprinting on strings

Algorithm Design (2021/2022)

Gabriele Pappalardo
Email: g.pappalardo4@studenti.unipi.it
Department of Computer Science

March 2022

1 Introduction

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 preprocessing, 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.

2 Solution

2.1 Baseline solution

Before giving the requested solution, we start from the baseline one. The problem can be solved in $O(n^2)$ space with $O(1)$ time to compute the `equal(i, j, l)` and $O(\log n)$ time for the `lce(i, j)`. The baseline solution makes use of a matrix $M \in \mathbb{Z}_p^{n \times n}$ where each cell contains the Karp-Rabin fingerprint F_{ij}.

$$\begin{bmatrix} F_{00} & F_{01} & F_{02} & F_{03} & \dots & F_{0n} \\ \emptyset & F_{11} & F_{12} & F_{13} & \dots & F_{1n} \\ \emptyset & \emptyset & F_{22} & F_{23} & \dots & F_{2n} \\ \emptyset & \emptyset & \emptyset & F_{33} & \dots & F_{3n} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & F_{nn} \end{bmatrix} \tag{1}$$

Figure 1: Where F_{ij} is the Karp-Rabin of the substring $S[i \dots j]$

To compute the `equal(i, j, l)` we will use the function `equal(M, i, j, l)` shown in the pseudo-Python code list 1.

```
def equal(M, i, j, l):

    # n is the length of the string
    if (i + l - 1) >= n or (j + l - 1) >= n:
            return False

    fi = M[i][i + l - 1]
    fj = M[j][j + l - 1]

    return (fi == fj)
```

Listing 1: "Function to compute the equality."

To compute the `lce(i, j, l)` we will use the function `lce(M, i, j, l)` show in the pseudo-Python code list 2.

```
def lce(M, i, j, l):
    if l > 0:
        if equal(M, i, j, l):
            return l + lce(M, i + l, j + l, l)
        else:
            return lce(M, i, j, l / 2)
    else:
        return 0
```

Listing 2: "Function to compute the longest common extension."

These two algorithms respect the requested running time.

2.2 Improving space

To reduce the space used by the matrix M, we can use just one row, i.e. the first one. The Karp-Rabin fingerprint of a sub-string $S[i \dots j], i < j$ is computed using the equation 2.

$$F_{ij} = h(S) = \sum_{k=i}^{j} \operatorname{ord}(S_k)\sigma^{k-i} \mod p \tag{2}$$

Where p is a prime number and $\operatorname{ord} : \Sigma \to \mathbb{N}$ is a function converting a character belonging to the string alphabet to a natural number (e.g. the `String.fromCharCode( ... )` function in JavaScript). The hash function can be viewed as a polynomial equation in $\mathbb{Z}[x]_p$ field, as shown in Equation 3 (using the Horner's method).

$$\begin{aligned} F_{0n} = h(S) &= \operatorname{ord}(S_0) + \operatorname{ord}(S_1)\sigma + \operatorname{ord}(S_2)\sigma^2 + \cdots + \operatorname{ord}(S_{n-1})\sigma^{n-1} \mod p & (3)\\ &= \operatorname{ord}(S_0) + \sigma(\operatorname{ord}(S_1) + \sigma(\operatorname{ord}(S_2) + \cdots + \sigma\operatorname{ord}(S_{n-1})))) \mod p & (4) \end{aligned}$$

We can exploit the properties of this rolling hash function to compute the missing fingerprints that we removed from the matrix. In fact, given two indices, i, j we can find its fingerprint F_{ij}, using only the first row of the matrix, in a mathematical way as shown in Equation 5.

$$F_{ij} = ((F_{0j} - F_{0(i-1)})/\sigma^i) \mod p \tag{5}$$

We show a numerical example. Suppose we have a string S (where $|S| > 6$), we are going to find the fingerprint F_{36} using the statement above. We are assuming $\operatorname{ord}(S_k) = S_k$ in order to ease computations.

$$F_{36} = (F_{06} - F_{02})/\sigma^3 \mod p \tag{6}$$

$$F_{36} = S_3 + S_4\sigma + S_5\sigma^2 + S_6\sigma^3 \mod p \tag{7}$$

We know already, F_{06} and F_{02} which are computed as shown below.

$$F_{06} = S_0 + S_1\sigma + S_2\sigma^2 + S_3\sigma^3 + S_4\sigma^4 + S_5\sigma^5 + S_6\sigma^6 \mod p \tag{8}$$

$$F_{02} = S_0 + S_1\sigma + S_2\sigma^2 \mod p \tag{9}$$

$$F_{06} - F_{02} = S_3\sigma^3 + S_4\sigma^4 + S_5\sigma^5 + S_6\sigma^6 \mod p \tag{10}$$

$$F_{36} = (F_{06} - F_{02})/\sigma^3 \mod p = S_3 + S_4\sigma + S_5\sigma^2 + S_6\sigma^3 \mod p \tag{11}$$

2.3 Error analysis

The error probability for the equal function is $\frac{1}{n^c}$. For the LCE function we have an error probability of $\frac{1}{n^c} \log n$.