Hands On 3: Karp-Rabin fingerprinting on strings

Algorithm Design

Ferraro Domenico
559813

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, $lce(i,j) = max\{l \geq 1 : S[i \dots i + l - 1] = S[j \dots j + l - 1]\}$.

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.

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

2 Solution

Let's start with the algorithm to compute the $lce(i,\ j)$ in $O(log\ n)$ time. It is important to note that the result can at most be equal to the minimum value between $j - i$ and $n - j$. Let's say that the highest possible $lce(i,\ j)$ is equal to h. We can follow a binary search strategy: check the equality of the substrings $S[i \dots\ i + \frac{h}{2}]\ and\ S[j \dots\ j + \frac{h}{2}]$, if they are equal then it means that the initial substring of length $\frac{h}{2}$ is part of the lce so we can count that value and we can recursively apply the same strategy starting from that initial substring, otherwise that value is ignored and the algorithm continues recursively to the half of the initial substring.

The following is the pseudocode:

```
LCE(i, j, l) {
  if (l <= 0) return 0;
  if (equal(i, j, l)) return l + LCE(i+l, j+l, l);
  return LCE(i, j, l/2);
}
```

Checking the equality, as requested, will cost constant time. Because of that and because of the binary search-like approach, the overall time complexity is $O(log\ n)$.

Let's now talk about how to compute $equal(i, j, l)$ in constant time. We will start with the baseline which is not good in terms of space usage but then we will improve it.

Given S, preprocess in advance every possible fingerprint and store them in a matrix. That means that we will have precomputed the fingerprint for each possible substring, and we will be able to answer for the equality in constant time. The following is an example of a matrix that stores all the fingerprints for a string S with length $n = 5$.

	0	*1*	*2*	*3*	*4*
0	F_{00}	F_{01}	F_{02}	F_{03}	F_{04}
1	-	F_{11}	F_{12}	F_{13}	F_{14}
2	-	-	F_{22}	F_{23}	F_{24}
3	-	-	-	F_{33}	F_{34}
4	-	-	-	-	F_{44}

F_{ij} is the fingerprint for the substring $S[i \dots j]$.

Such matrix of course takes $O(n^2)$ space. We can formalize the Karp-Rabin fingerprint F_{ij} of the substring $S[i \dots j] = s_i s_{i+1} \dots s_{j-1} s_j$ as the following

$$F_{ij} = h(S[i \dots j]) = \sum_{k=i}^{j} val(S_k) q^{k-i} \quad (mod\ p)$$

Where $val(S_k)$ is a function that gives the numeric value of the character of S in position k, and p is a prime number. To achieve a better space usage, note that the fingerprint $F_{a,b}$ can be computed in constant time if we know $F_{0,a-1}$ and $F_{0,b}$:

$$F_{a,b} = \frac{(F_{0,b} - F_{0,a-1})}{q^a}$$

To give the demonstration, we can see that

$$\begin{aligned} F_{0,b} &= val(s_0) + val(s_1)q + \cdots + val(s_b)q^b \\ F_{0,a-1} &= val(s_0) + val(s_1)q + \cdots + val(s_{a-1})q^{a-1} \end{aligned}$$

Then, because $a \leq b$, $F_{0,b}$ can be also written as

$$\begin{aligned} F_{0,b} &= val(s_0) + val(s_1)q + \cdots + val(s_{a-1})q^{a-1} + \cdots + val(s_b)q^b \\ &= F_{0,a-1} + \cdots + val(s_b)q^b \end{aligned}$$

So, we can compute $F_{0,b} - F_{0,a-1}$

$$F_{0,b} - F_{0,a-1} = val(s_a)q^a + \cdots + val(s_b)q^b$$

And finally

$$\begin{aligned} F_{a,b} &= val(s_a) + val(s_{a+1})q + \cdots + val(s_b)q^{b-a} \\ &= \frac{(F_{0,b} - F_{0,a-1})}{q^a} \end{aligned}$$

So, we only need to store the fingerprints of all the substrings that starts from the first character. To subtract two fingerprints, it will take constant time but computing q^a is not done in constant time. To also improve that we can precompute and store all the powers of q from 1 to n. Finally, considering all the optimizations, the space usage will be $O(n)$.

The query $equal(i, j, l)$ will just have to compute $F_{i,i+l}$ and $F_{j,j+l}$ and then return the comparison between them. Of course, such operations take constant time.

2.1 Error analysis

We have an error if the substrings are not equal, but the fingerprints are.

$$\Pr\left[S[a \ldots b] \neq S[c \ldots d] \wedge h\left(F_{a,b}\right) = h\left(F_{c,d}\right)\right] = \frac{\#\, bad\ primes}{\#\, all\ possible\ primes} \leq \frac{n}{\tau / \ln \tau}$$

If we choose $\tau \sim n^{c+1} \ln n$ for a constant $c > 0$, we have

$$\Pr[error] < \frac{1}{n^c}$$

The error probability for $equal$ is then $\frac{1}{n^c}$, while for lce is $\frac{1}{n^c} \log n$.