Com S 311: Hashing Application: Pattern Matchinh

February 13, 2018

1 Pattern Matching

We will apply the idea of hashing to design an efficient algorithm for pattern matching. Let x be a string of length n and y be a string of length m. We would like to know whether y appears as a substring in x or not. *Brute-force algorithm* for this problem is as follows:

1. Inputs: $x = x_1....x_n, y = y_1...y_m$
2. for i in range $[1, n - m + 1]$
   - if y equals $x_i x_{i+1}....x_{i+m-1}$, then return TRUE.
3. return FALSE.

What is the time taken by the above algorithm? In the worst-case, the loop is repeated $n - m$ times. Consider an iteration of a loop: Here we are comparing two m-character strings; this can be done in $O(m)$ time. Thus the total time taken by the above algorithm is $O(n - m)m$ which is $O(nm)$.

Can we reduce the run time? Recall that comparing integers takes less time than comparing Strings. This suggests the following idea: To compare two strings, first convert them into integers and then compare the integers. Let h be a function that maps strings to integers. Using this we arrive at the following *hash-based algorithm.*

1. Inputs: $x = x_1....x_n, y = y_1...y_m$
2. Compute $h(y)$. Denote this value with N_y
3. for i in range $[1, n - m + 1]$
   - Compute $h(x_i x_{i+1}....x_{i+m-1})$. Denote this value with N_i.
   - if N_y equals N_i, return TRUE.
4. return FALSE.

How much time does the above algorithm take? Consider an iteration of the loop. Time taken is: Time taken to compute h + time taken to compare N_y with N_i. How much time does it take to compute h? Recall our choice for h (from previous lecture): Let $z = c_1 c_2 \cdots c_m$ be a m-character String.

$$h(z) = c_m + c_{m-1}\alpha + c_{m-2}\alpha^2 + \cdots + c_2\alpha^{m-2} + c_1\alpha^{m-1}$$

It can be seen that the time taken to compute above function is $O(m)$ (Think about it). Thus the time taken by the above algorithm is: $O(nm)$ which is the same as the run-time of the Brute-Force algorithm. During each iteration of the above algorithm, we are computing h on a string. This computation takes $O(m)$ time. Can we reduce this computation time? We will see next that we can reduce this time.

To illustrate the idea to reduce the run time, let us consider the first two iterations of the above algorithm: In the first iteration, we compute h on $x_1, \cdots x_m$. In the second iteration, we compute h on $x_2, \cdots x_m + 1$. Naive approach for this takes $2m$ time. Can we do in less time? Let us look at the values of h on each of these strings. $h(x_1, \cdots x_m)$ is

$$x_m + x_{m-1}\alpha + x_{m-2}\alpha^2 + \cdots + x_2\alpha^{m-2} + x_1\alpha^{m-1}$$

Whereas $h(x_2, \cdots, x_{m+1})$ is

$$x_{m+1} + x_m\alpha + x_{m-1}\alpha^2 + \cdots + x_3\alpha^{m-2} + x_2\alpha^{m-1}$$

Note that

$$h(x_2, \cdots, x_{m+1}) = (h(x_1, \cdots x_m) - x_1\alpha^{m-1}) \times \alpha + x_{m+1}$$

Thus if we know $h(x_1, \cdots, x_m)$, then we can compute $h(x_2, \cdots, x_{m+1})$ easily (by doing few arithmetic operations). Thus we can compute both $h(x_1, \cdots x_m)$ and $h(x_2, \cdots, x_{m+1})$ in $m + constant$ time! (instead of $2m$ time).

This idea is known as *roll-over hashing*. It exploits the structure of the hash function along with the fact that the strings $x_1 \cdots x_m$ and $x_2 \cdots x_{m+1}$ share many characters, and thus it is easy to compute hash value of one string from the other string. Finally, using the idea of roll-over hashing, we arrive at the following algorithm. This algorithm is known as Karp-Rabin algorithm.

1. Input x, y.
2. Compute $N_y = h(y)$.
3. Compute $N_1 = h(x_1 \cdots x_m)$.
4. If N_y equals N_1, return TRUE.
5. for i in the range $[2, \cdots n - m + 1]$
   - $N_i = [(N_{i-1} - x_{i-1}\alpha^{m-1}) \times \alpha + x_{i+m-1}]]$
   - If N_i equals N_y, return TRUE.
6. Return FALSE.

The above algorithm can be implemented in $O(m + n)$ time. This is an improvement over $O(mn)$.

Remark. The above algorithm can give a wrong answer in the following scenario. We have that $y \neq x_i \cdots x_{i+m-1}$, and let N_y equals N_i. Think about how to fix it.