The Rabin-Karp algorithm

Rabin and Karp have proposed a string-matching algorithm that performs well in practice and that also generalizes to other algorithms for related problems, such as two-dimensional pattern matching. The worst-case running time of the Rabin-Karp algorithm is $O((n - m + 1)m)$, but it has a good average-case running time.

This algorithm makes use of elementary number-theoretic notions such as the equivalence of two numbers modulo a third number. You may want to refer to Section 33.1 for the relevant definitions.

For expository purposes, let us assume that $\Sigma = \{0, 1, 2, . . . , 9\}$, so that each character is a decimal digit. (In the general case, we can assume that each character is a digit in radix-d notation, where $d = |\Sigma|$.) We can then view a string of k consecutive characters as representing a length-k decimal number. The character string 31415 thus corresponds to the decimal number 31,415. Given the dual interpretation of the input characters as both graphical symbols and digits, we find it convenient in this section to denote them as we would digits, in our standard text font.

Given a pattern $P[1 . . m]$, we let p denote its corresponding decimal value. In a similar manner, given a text $T[1 . . n]$, we let t_s denote the decimal value of the length-m substring $T[s + 1 . . s + m]$, for $s = 0, 1, . . . , n - m$. Certainly, $t_s = p$ if and only if $T[s + 1 . . s + m] = P[1 . . m]$; thus, s is a valid shift if and only if $t_s = p$. If we could compute p in time $O(m)$ and all of the t_i values in a total of $O(n)$ time, then we could determine all valid shifts s in time $O(n)$ by comparing p with each of the t_s's. (For the moment, let's not worry about the possibility that p and the t_s's might be very large numbers.)

We can compute p in time $O(m)$ using Horner's rule (see Section 32.1):

$$p = P[m] + 10\ (P[m - 1] + 10(P[m - 2] + . . . + 10(P[2] + 10P[1]) . . .)).$$

The value t_0 can be similarly computed from $T[1 . . m]$ in time $O(m)$.

To compute the remaining values $t_1, t_2, . . . , t_{n-m}$ in time $O(n - m)$, it suffices to observe that t_{s+1} can be computed from t_s in constant time, since

$$t_s + 1 \quad = \quad 10(t_s - 10^m - 1T[s + 1]) + T[s + m + 1].$$

(34.1)

For example, if $m = 5$ and $t_s = 31415$, then we wish to remove the high-order digit $T[s + 1] = 3$ and bring in the new low-order digit (suppose it is $T[s + 5 + 1] = 2$) to obtain

```
t_s+1 = 10(31415 - 10000.3) + 2
= 14152 .
```

Subtracting 10^{m}-1 $T[s+1]$ removes the high-order digit from t_s, multiplying the result by 10 shifts the number left one position, and adding $T[s + m + 1]$ brings in the appropriate low-order digit. If the constant 10^{m-1} is precomputed (which can be done in time $O(1g\ m)$ using the techniques of Section 33.6, although for this application a straightforward $O(m)$ method is quite adequate), then each execution of equation (34.1) takes a constant number of arithmetic operations. Thus, p and $t_0, t_1, \ldots, t_{n}$-m can all be computed in time $O(n + m)$, and we can find all occurrences of the pattern $P[1 \ldots m]$ in the text $T[1 \ldots n]$ in time $O(n + m)$.

The only difficulty with this procedure is that p and t_s may be too large to work with conveniently. If P contains m characters, then assuming that each arithmetic operation on p (which is m digits long) takes "constant time" is unreasonable. Fortunately, there is a simple cure for this problem, as shown in Figure 34.4 : compute p and the t_s's modulo a suitable modulus q. Since the computation of p, t_0, and the recurrence (34.1) can all be performed modulo q, we see that p and all the t_s's can be computed modulo q in time $O(n + m)$. The modulus q is typically chosen as a prime such that $10q$ just fits within one computer word, which allows all of the necessary computations to be performed with single-precision arithmetic.

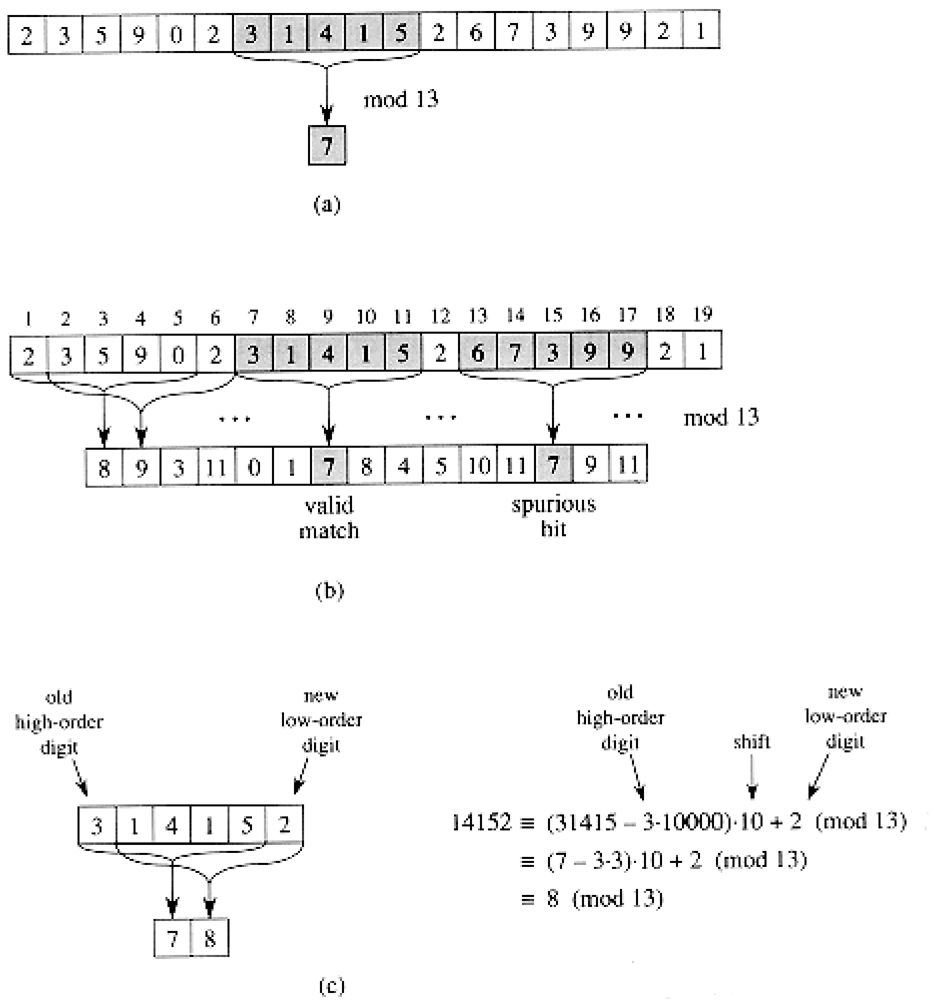

Figure 34.4 The Rabin-Karp algorithm. Each character is a decimal digit, and we compute values modulo 13. (a) A text string. A window of length 5 is shown shaded. The numerical value of the shaded number is computed modulo 13, yielding the value 7. (b) The same text string with values computed modulo 13 for each possible position of a length-5 window. Assuming the pattern P = 31415, we look for windows whose value modulo 13 is 7, since 31415 ≡ 7 (mod 13). Two such windows are found, shown shaded in the figure. The first, beginning at text position 7, is indeed an occurrence of the pattern, while the second, beginning at text position 13, is a spurious hit. (c) Computing the value for a window in constant time, given the value for the previous window. The first window has value 31415. Dropping the high-order digit 3, shifting left (multiplying by 10), and then adding in the low-order digit 2 gives us the new value 14152. All computations are performed modulo 13, however, so the value for the first window is 7, and the value computed for the new window is 8.

In general, with a *d*-ary alphabet {0, 1, . . . ,*d* - 1}, we choose *q* so that *d q* fits within a computer word and adjust the recurrence equation (34.1) to work modulo *q*, so that it becomes

$$t_s+1 = (d(t_s - T[s + 1]h) + T[s + m + 1]) \bmod q ,$$

(34.2)

where $h \equiv d^m$-1 (mod q) is the value of the digit "1" in the high-order position of an m-digit text window.

The ointment of working modulo q now contains a fly, however, since $t_s \equiv p$ (mod q) does not imply that $t_s = p$. On the other hand, if $t_s \not\equiv p$ (mod q), then we definitely have that $t_s \neq p$, so that shift s is invalid. We can thus use the test $t_s \equiv p$ (mod q) as a fast heuristic test to rule out invalid shifts s. Any shift s for which $t_s \equiv p$ (mod q) must be tested further to see if s is really valid or we just have a ***spurious hit***. This testing can be done by explicitly checking the condition $P[1 . . m] = T[s + 1 . . s + m]$. If q is large enough, then we can hope that spurious hits occur infrequently enough that the cost of the extra checking is low.

The following procedure makes these ideas precise. The inputs to the procedure are the text T, the pattern P, the radix d to use (which is typically taken to be $|\Sigma|$), and the prime q to use.

```
RABIN-KARP-MATCHER(T, P, d, q)
1  n ← length[T]
2  m ← length[P]
3  h ← $d^{m-1}$ mod q
4  p ← 0
5  $t_0$ ← 0
6  for i ← 1 to m
7        do p ← (dp + P[i]) mod q
8           $t_0$ ← ($dt_0$ + T[i]) mod q
9  for s ← 0 to n - m
10       do if p = $t_s$
11            then if P[1 . . m] = T[s + 1 . . s + m]
12                    then "Pattern occurs with shift" s
13          if s < n - m
14            then $t_s$+1 ← (d($t_s$ - T[s + 1]h) + T[s + m + 1]) mod q
```

The procedure RABIN-KARP-MATCHER works as follows. All characters are interpreted as radix-d digits. The subscripts on t are provided only for clarity; the program works correctly if all the subscripts are dropped. Line 3 initializes h to the value of the high-order digit position of an m-digit window. Lines 4-8 compute p as the value of $P[1 . . m]$ mod q and t_0 as the value of $T[1 . . m]$ mod q. The **for** loop beginning on line 9 iterates through all possible shifts s. The loop has the following invariant: whenever line 10 is executed, $t_s = T[s + 1 . . s + m]$ mod q. If $p = t_s$ in line 10 (a "hit"), then we check to see if $P[1 . . m] = T[s + 1 . . s + m]$ in line 11 to rule out the possibility of a spurious hit. Any valid shifts found are printed out on line 12. If $s < n - m$ (checked in line 13), then the **for** loop is to be executed at least one more time, and so line 14 is first executed to ensure that the loop invariant holds when line 10 is again reached. Line 14 computes the value of t_{s+1} mod q from the value of t_s mod q in constant time using equation (34.2) directly.

The running time of RABIN-KARP-MATCHER is $\Theta((n - m + 1)m)$ in the worst case, since (like the naive string-matching algorithm) the Rabin-Karp algorithm explicitly verifies every valid shift. If $P = a^m$ and $T = a^n$, then the verifications take time $\Theta((n - m + 1)m)$, since each of the $n - m + 1$ possible shifts is valid. (Note also that the computation of $d^{m-1} \bmod q$ on line 3 and the loop on lines 6-8 take time $O(m) = O((n - m + 1)m)$.)

In many applications, we expect few valid shifts (perhaps $O(1)$ of them), and so the expected running time of the algorithm is $O(n + m)$ plus the time required to process spurious hits. We can base a heuristic analysis on the assumption that reducing values modulo q acts like a random mapping from Σ^* to $\mathbf{Z}_q$. (See the discussion on the use of division for hashing in Section 12.3.1. It is difficult to formalize and prove such an assumption, although one viable approach is to assume that q is chosen randomly from integers of the appropriate size. We shall not pursue this formalization here.) We can then expect that the number of spurious hits is $O(n/q)$, since the chance that an arbitrary t_s will be equivalent to p, modulo q, can be estimated as $1/q$. The expected amount of time taken by the Rabin-Karp algorithm is then

$$O(n) + O(m(v + n/q)),$$

where v is the number of valid shifts. This running time is $O(n)$ if we choose $q \geq m$. That is, if the expected number of valid shifts is small ($O(1)$) and the prime q is chosen to be larger than the length of the pattern, then we can expect the Rabin-Karp procedure to run in time $O(n + m)$.