String Matching Using the Rabin-Karp Algorithm

String Matching Problem

- We assume that the text is an array T [1..N] of length n and that the pattern is an array P [1..M] of length m, where m << n. We also assume that the elements of P and T are characters in the finite alphabet Σ.

(e.g., Σ = {a,b} We want to find *P* = ‘aab’ in *T* = ‘abbaabaaaab’)

String Matching Problem (Continued)

- The idea of the string matching problem is that we want to find all occurrences of the pattern P in the given text T.
- We could use the brute force method for string matching, which utilizes iteration over T. At each letter, we compare the sequence against P until all letters match of until the end of the alphabet is reached.
- The worst case scenario can reach O(N*M)

Definition of Rabin-Karp

- A string search algorithm which compares a string's hash values, rather than the strings themselves.

- For efficiency, the hash value of the next position in the text is easily computed from the hash value of the current position.

How Rabin-Karp works

- Let characters in both arrays T and P be digits in radix-Σ notation. ($\Sigma = (0,1,\ldots,9)$)
- Let p be the value of the characters in P
- Choose a prime number *q* such that fits within a computer word to speed computations.
- Compute (p mod q)
  - The value of p mod q is what we will be using to find all matches of the pattern P in T.

How Rabin-Karp works (continued)

- Compute (T[s+1, .., s+m] mod q) for s = 0 .. n-m
- Test against P only those sequences in T having the same (mod q) value
- (T[s+1, .., s+m] mod q) can be incrementally computed by subtracting the high-order digit, shifting, adding the low-order bit, all in modulo q arithmetic.

A Rabin-Karp example

- Given T = 31415926535 and P = 26
- We choose q = 11
- P mod q = 26 mod 11 = 4

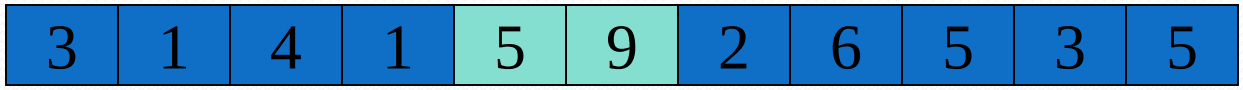
31 mod 11 = 9 not equal to 4

14 mod 11 = 3 not equal to 4

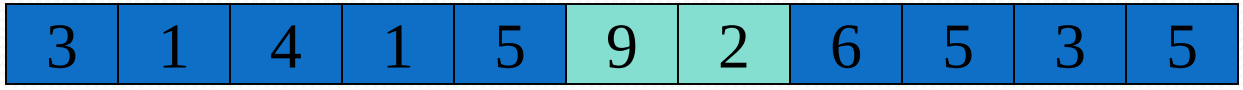
41 mod 11 = 8 not equal to 4

Rabin-Karp example continued

15 mod 11 = 4 equal to 4 -> spurious hit



59 mod 11 = 4 equal to 4 -> spurious hit



92 mod 11 = 4 equal to 4 -> spurious hit

26 mod 11 = 4 equal to 4 -> an exact match!!

65 mod 11 = 10 not equal to 4

Rabin-Karp example continued

53 mod 11 = 9 not equal to 4

35 mod 11 = 2 not equal to 4

As we can see, when a match is found, further testing is done to insure that a match has indeed been found.

$$m = 4$$

$$6378 = 8 + 7 \times 10 + 3 \times 10^2 + 6 \times 10^3$$

$$= 8 + 10\,(7 + 10\,(3 + 10(6)))$$

$$= 8 + 70 + 300 + 6000$$

$$p = P[m] + 10(P[m\text{-}1] + 10(P[m\text{-}2] + \ldots + 10(P[2] + 10(P[1]))$$

t_{s+1} **can be computed from** t_s **in constant time.**

$$t_{s+1} = 10(t_s - 10^{m-1} T[s+1]) + T[s+m+1]$$

Example : $T = 314152$

$t_s = 31415$, $s = 0$, $m = 5$ **and** $T[s+m+1] = 2$

$$t_{s+1} = 10(31415 - 10000*3) + 2 = 14152$$

Thus p and $t_0, t_1, \ldots, t_{n-m}$ **can all be computed in O(**$n+m$**) time. And all occurences of the pattern** $P[1..m]$ **in the text** $T[1..n]$ **can be found in time** $O(n+m)$**.**

However, p **and** t_s **may be too large to work with conveniently.**

Do we have a simple solution!!

Computation of p and t_0 and the recurrence is done using modulus q.

In general, with a d-ary alphabet {0,1,…,d-1}, q is chosen such that $d \times q$ fits within a computer word.

The recurrence equation can be rewritten as

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

where $h = d^{m-1} \pmod q$ is the value of the digit "1" in the high order position of an m-digit text window.

Note that $t_s \equiv p \bmod q$ does not imply that $t_s = p$.

However, if t_s is not equivalent to p mod q ,

then $t_s \neq p$, and the shift s is invalid.

We use $t_s \equiv p \bmod q$ as a fast heuristic test to rule out the invalid shifts.

Further testing is done to eliminate spurious hits.

- an explicit test to check whether

$$P[1..m] = T[s+1..s+m]$$

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

$$h = d^{m-1} (\bmod q)$$

Example :

T = 31415; P = 26, n = 5, m = 2, q = 11

p = 26 mod 11 = 4

t0 = 31 mod 11 = 9

t1 = (10(9 - 3(10) mod 11) + 4) mod 11

= (10 (9- 8) + 4) mod 11 = 14 mod 11 = 3

Procedure RABIN-KARP-MATCHER(T,P,d,q)

Input : Text *T*, pattern *P*, radix *d* (which is typically = $|\Sigma|$), and the prime *q*.

Output : valid shifts *s* where *P* matches

1. $n \leftarrow$ length[T];
2. $m \leftarrow$ length[P];
3. $h \leftarrow d^{m-1} \bmod q$;
4. $p \leftarrow 0$;
5. $t_0 \leftarrow 0$;
6. for $i \leftarrow 1$ to m
7. do $p \leftarrow (d \times p + P[i] \bmod q$;
8. $t_0 \leftarrow (d \times t_0 + T[i] \bmod q$;
9. for $s \leftarrow 0$ 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 $t_{s+1} \leftarrow (d(t_s - T[s+1]h) + T[s+m+1]) \bmod q$;

Code

RABIN-KARP-MATCHER(T, P, d, q)

```
n ← length[ T ]
m ← length[ P ]
h ← d^(m-1) mod q
p ← 0
t_0 ← 0
for i ← 1 to m                                   ► Preprocessing
      do   p ← ( d*p + P[ i ] ) mod q
           t_0 ← ( d*t_0 + T[ i ] ) mod q
for s ← 0 to n – m                               ► Matching
      do if p = t_s
           then if P[ 1..m ] = T[ s+1 .. s+m ]
                     then print "Pattern occurs with shift" s
      if s < n – m
         then t_(s+1) ← ( d * ( t_s – T[ s + 1 ] * h ) + T[ s + m + 1 ] ) mod q
```

Complexity

- The running time of the Rabin-Karp algorithm in the worst-case scenario is O(n-m+1)m but it has a good average-case running time.
- If the expected number of valid shifts is small O(1) and the prime q is chosen to be quite large, then the Rabin-Karp algorithm can be expected to run in time O(n+m) plus the time to required to process spurious hits.

Applications

- Bioinformatics
  - Used in looking for similarities of two or more proteins; i.e. high sequence similarity usually implies significant structural or functional similarity.

Example:

Hb A_human
GSAQVKGHGKKVADALTNAVAHVDDMPNALSALSDLHAHKL
G+ +VK+HGKKV A+++++++AH+ D++ ++ +++LS+LH KL
Hb B_human
GNPKVKAHGKKVLGAFSDGLAH LDNLKGTF ATLSELH CDKL
+ similar amino acids

Applications continued

- Alpha hemoglobin and beta hemoglobin are subunits that make up a protein called hemoglobin in red blood cells. Notice the similarities between the two sequences, which probably signify functional similarity.
- Many distantly related proteins have domains that are similar to each other, such as the DNA binding domain or cation binding domain. To find regions of high similarity within multiple sequences of proteins, local alignment must be performed. The local alignment of sequences may provide information of similar functional domains present among distantly related proteins.