Curtin
University of Technology

Design and Analysis of Algorithms

Lecture 08

String Matching Algorithms

Topics

- **Basics of Strings**
- **Brute-force String Matcher**
- **Rabin-Karp String Matching Algorithm**

Read Chapter 32 (34) new (old) Cormen

Curtin University of Technology

String matching problem

- Find the occurrence of a pattern in a text
- These problems find applications in text processing, text-editing, computer security, and DNA sequence analysis
- **Text**: $T[1..n]$ of length n $\rightarrow$ T = abracadabraabracadabra
- **Pattern**: $P[1..m]$ of length m $\rightarrow$ P = ada
- Elements of P & T are characters from a finite **alphabet** set Σ
- For example $\Sigma=\{0,1\}$ or $\Sigma=\{a,b, . . . , z\}$, or $\Sigma=\{c, g, a, t\}$ c=cytosine; g=guanine; a=adenosine; t=thymine

Curtin University of Technology

String matching problem

- The character arrays of P and T are also referred to as strings of characters
- Pattern P is said to occur with **shift** s in text T if:

$$T[s+1..s+m] = \mathrm{P}[1..m] \text{ or } T[s+j] = P[j]$$

$$\text{for } 0 \leq s \leq n-m \text{ and } 1 \leq j \leq m,$$

such a shift is called a **valid shift**.

T = abrac**<u>ada</u>**braabrac**ada**bra; P = **ada**; s=5

- The string-matching problem:
  - find all valid shifts with which a given pattern P occurs in a given text T

Curtin University of Technology

T = a c a a b c a c a a b c ; n = 12

a a b a a b ; m = 3

a c a a b c

a a b

a c a a b c matches; s=2

a a b

Curtin University of Technology

Brute Force String Search

BF_String_Matcher(T, P)

```
1.    n = length [T];
2.    m = length[P];
3.    for s = 0 to n - m  // O(n-m+1)
4.          do if P[1..m] = T[s+1..s+m]  // O(m)
5.                then shift s is valid
```

Worst case: a text string a^n and a pattern a^m

e.g., T = aaaaaa …aaa; P=aaaa

This algorithm takes $\Theta((n - m + 1)m)$ in the worst case

Best case: Line 4: $O(1)$

Rabin-Karp Algorithm

- Let $\Sigma = \{0,1,2, \ldots,9\}$
- We can view a string of k consecutive characters as representing a length-k decimal number
- Let p denote the decimal number for $P[1..m]$
- Let t_s denote the decimal value of the length-m substring $T[s+1..s+m]$ of T for $s = 0, 1, \ldots, n - m$.

$t_s = p$ iff $T[s+1..s+m] = P[1..m]$, and s is a valid shift.

Horner's rule:

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

Rabin-Karp Example

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

$m = 4$

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

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

$= 8 + 70 + 300 + 6000$

We can compute p in $O(m)$ time

We can compute t_0 from $T[1..m]$ in $O(m)$ time.

How to compute t_{s+1}?

t_{s+1} can be computed from t_s in constant time as follow:

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

- 10^{m-1} is pre-computed
- Subtracting $(10^{m-1}\, T[s+1])$ removes the highest order digit of t_s
- Adding $T[s+m+1]$ brings in the low order digit

Example:

$T = 314152$

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

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

Rabin-Karp - Time Complexity

- p can be computed in $O(m)$ time.
- t_{s+1} can be computed from t_s in $O(1)$
- $t_0, t_1, \ldots, t_{n-m}$ can all be computed in $O(n\text{-}m+1)$ time; there are n-m windows of m-digits.
- All occurrences of the pattern $P[1..m]$ in the text $T[1..n]$ can be found in time $O(n)$.

Curtin University of Technology

Too Good to be True?

Assuming a RAM model of computation, there is a maximum word size: 2^w

What if the numbers become bigger than 2^w?

e.g., If Σ =ASCII, $|\Sigma| = 256$, what is the biggest pattern on a 32-bit machine?

one character pattern $p \in [0,255]$
two character pattern $p \in [0,2^{16}]$
three character pattern $p \in [0,2^{24}]$
…
Ex: P=ABCD $\rightarrow$ $68+67*2^8+66*2^{16}+65*2^{24}$

Curtin University of Technology

Solution

Computation of p and t_0 and the recurrence is done mod q ; choose a prime number for q

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

The recurrence equation can be rewritten from

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

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

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

Solution

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

However, if t_s is not equivalent to $p \bmod q$,

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

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

Curtin University of Technology

Example

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

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

Example: d=10

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

p = 26 mod 11 = 4; $h = 10^{2-1}$ mod 11 = 10

t_0 = 31 mod 11 = 9

t_1 = (10 (9 – 3.10) + 4) mod 11 = -206 mod 11 = 3

Note: 206 mod 11 = 206 - 198 = 8

-206 mod 11 = -206 - (-209) = 3.

Curtin University of Technology

Example (cont.)

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

$p = 26 \bmod 11 = 4$; $h = 10^{2-1} \bmod 11 = 10$

$t_0 = 31 \bmod 11 = 9$

<u>OR</u>

$t_1 = 14 \bmod 11 = 3$, which can also be:

$= ((31 - 3.10)\ 10 + 4) \bmod 11$

$(10\ (9 - 8) + 4) \bmod 11 = 14 \bmod 11 = 3$

Curtin University of Technology

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

```
n = length[T]
m = length[P]
h = d^(m-1) mod q
p = 0
t_0 = 0
for i = 1 to m
      do  p = (d*p + P[i]) mod q
          t_0 = (d*t_0 + T[i]) mod q
for s = 0 to n-m
      do if p = t_s
            then if P[1..m] = T[s+1..s+m]
                  then "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
```

Comments on Rabin-Karp Algorithm

- All characters are interpreted as radix-d digits
- h is initiated to the value of high order digit position of an m-digit window
- p and t_0 are computed in $O(m+m) = O(m)$ time
- The loop in lines 6-8 takes $O(m)$ time
- The loop line 9 takes $\Theta((n-m+1)m)$
- The overall running time is $O((n-m+1)m)$

The End