String Matching

Module 3

AoA-Even 2021-22

Introduction

- Naïve String Matching Algorithm
- String Matching with Finite Automata
- Knuth Morris Pratt Algorithm

Naïve String Matching Algorithm

- String matching or pattern recognition is a problem for searching a pattern to be searched within a text under certain conditions and find out all occurrences of it.
- Pattern and text will be in form of an array of characters drawn from finite alphabet Σ.
- Pattern is denoted as P[1…m] and Text as T[1…n] where m and n are their respective length such that $n \geq m \geq 1$.
- If pattern P occurs in Text after ***s*** shifts then $\boldsymbol{P[1 \ldots m] = T[s+1 \ldots s+m]}$ where $\boldsymbol{n - m \geq s \geq 0}$.
- If P occurs after finite shift ***s*** in T, then we can say ***s*** is a valid shift, otherwise invalid shift

Naïve String Matching Algorithm

- Example

Position
s+1=3

Position
s + m=7

$\Sigma=\{a, b, c\}$

Text T

a	b	c	a	a	b	c	c	b	a	a	b

n = 12

Pattern P

s=2

c	a	a	b	c

m = 5

Naïve String Matching Algorithm

```
NAIVE-STRING-MATCHER(T, P)
n = T.length
m = P.length
for s = 0 to n − m
    if P[1..m] == T[s + 1..s + m]
        print "Pattern occurs with shift" s
```

String Matching Algorithm

Algorithm	Preprocessing time	Matching time
Naive	0	$O((n - m + 1)m)$
Rabin-Karp	$\Theta(m)$	$O((n - m + 1)m)$
Finite automaton	$O(m \lvert\Sigma\rvert)$	$\Theta(n)$
Knuth-Morris-Pratt	$\Theta(m)$	$\Theta(n)$

String-matching algorithms, their preprocessing and matching times

String Matching with Finite Automata

Finite Automata

A finite automaton **M** is a 5-tuple $(Q, \Sigma, \delta, s, F)$:

- $\mathbf{Q}$: the finite set of states
- $\boldsymbol{\Sigma}$: the finite input alphabet
- $\boldsymbol{\delta}$: the "transition function of **M**" from $Q \times \Sigma$ to Q
- $\mathbf{s} \in \mathbf{Q}$: the start state
- $\mathbf{F} \subset \mathbf{Q}$: the set of final (accepting) states

KMP Algorithm

- KMP is the first linear time algorithm for string matching.
- Prevents re examination of previously matched characters.
- This algorithm avoids computing the transition function δ altogether, and its matching time is $\theta(n)$ using just an auxiliary function π.
- $\pi[q]$ (Prefix Table ot LPS Table) stores information that is needed to compute transition function $\delta(q,a)$ but that does not depend on a.
- array $\pi[q]$ has only m entries, whereas δ has $\theta(m|\sum|)$.

KMP Algorithm

```
KMP-MATCHER(T, P)
n = T.length
m = P.length
π = COMPUTE-PREFIX-FUNCTION(P)
q = 0                                    // number of characters matched
for i = 1 to n                           // scan the text from left to right
    while q > 0 and P[q + 1] ≠ T[i]
        q = π[q]                         // next character does not match
    if P[q + 1] == T[i]
        q = q + 1                        // next character matches
    if q == m                            // is all of P matched?
        print "Pattern occurs with shift" i − m
        q = π[q]                         // look for the next match
```

KMP Algorithm

COMPUTE-PREFIX-FUNCTION(P)

$m = P.length$
let $\pi[1\,..\,m]$ be a new array
$\pi[1] = 0$
$k = 0$
for $q = 2$ **to** m
    while $k > 0$ and $P[k+1] \neq P[q]$
        $k = \pi[k]$
    if $P[k+1] == P[q]$
        $k = k + 1$
    $\pi[q] = k$
return π

The running time of COMPUTE-PREFIX-FUNCTION is $\Theta(m)$