

# Streng matching

Mads Ohm Larsen

6. april 2008

## Problemet

Vi søger at finde alle forekomster af mønsteret  $P[1..m]$  i teksten  $T[1..n]$ , hvor  $P$  og  $T$  består af tegn fra alfabetet  $\Sigma$ . Dette gøres ved at finde alle "skift" hvor  $P$  matcher. Skift vil være 0-index.

## Naïve Algoritme

$T[1..n] = \text{abaabac}$

$P[1..m] = \text{baa}$

Kører alle skift igennem og tjekker om den passer her.

## Køretid

$O((n - m + 1)m)$ , da vi skal tjekke alle værdier af  $s$  for  $P[1..m] = T[s + 1..s + m]$ , altså  $n - m + 1$  mulige værdier af  $s$ .

Denne grænse er tæt, da vi i værstetilfælde har  $T = a^n$  (n a'er ved siden af hinanden) som skal sammenlignes med  $a^m$ , altså matcher den alle steder. Vi foretager altså  $m$  sammenligninger på alle  $s \Rightarrow \Theta((n - m + 1)m)$ . Hvis  $m = \frac{n}{2}$  giver dette  $\Theta(n^2)$ .

## Rabin-Karp

Hele ideen bag Rabin-Karp algoritmen er at vi sammenligner hele  $P[1..m]$  med  $T[s + 1..s + m]$ .

Vi går til at starte med udfra at  $\Sigma = \{0, 1, 2..9\}$ , således at en streng på  $k$  tegn kan opfattes som et tal på  $k$  cifre.

Vi laver derfor  $P[1..m] = p$  om til dens tilsvarende tal værdi. Det samme er gældende for  $T$ , her skal vi bare lave sub-strengen  $t_s = T[s + 1..s + m]$  for  $s = 0, 1..n - m$  om.

Vi har altså nu at der er et gyldigt skift når  $t_s = p$ .

Vi kan udregne  $p$  i linær tid ( $\Theta(m)$ ), vha. Horner's regl:

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

$t_0$  kan regnes på samme måde. Da  $t_0 = T[1..n]$  bliver også dette i  $\Theta(m)$  tid.

De resterende  $t_1..t_{n-m}$  kan regnes i konstant tid vha. følgende sætning:

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

med andre ord så tager vi det vi er nået til ( $t_s$ ) og trækker det mest betyden-  
de tal (det længst til venstre) fra. Så skifter vi hele tallet til venstre (ganger  
med 10) og ligger det næste tal i rækken til. (Dette gælder selvfølgelig kun for  
( $\Sigma = \{0, 1..9\}$ ))

Vi har altså nu at der kræves  $\Theta(m)$  i forberedelses tid og  $\Theta(n - m + 1)$  i sam-  
menligningstid. (Vi laver sammenligninger for  $t_0..t_{n-m}$ )

Det eneste problem der viser sig nu er, at vi kan have tal der er så store at de  
ikke kan være i et computerord. Arithmetiske operationer vil ikke kunne udføres  
i konstant tid på sådanne tal. Et andet problem, som vi løser med det samme,  
er at vi kun kan bruge et begrænset  $\Sigma$ . Laver vi vores forgående sætning om til

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

hvor  $d = |\Sigma|$  og  $h \equiv d^{m-1} \pmod{q}$ , hvor  $q$  er et godt valgt primtal. Vi har nu to  
tal vi kan sammenligne.  $t_s \pmod{q}$  og  $p \pmod{q}$ . Desværre medfører  $t_s \equiv p \pmod{q}$   
 $\not\Rightarrow t_s = p$  ikke. Men vi kan bruge det til noget alligevel, for vi ved at følgende  
gælder  $t_s \not\equiv p \pmod{q} \Rightarrow t_s \neq p$ . Vi bliver altså nødt til at teste alle  $t_s \equiv p \pmod{q}$   
for at være sikker på at det er et **hit** og ikke bare et **spurious hit** (*missed*).

### Køretid

I værstetilfælde virker **Rabin-Karp**-algoritmen som den naive algoritme. F.eks.  
kan vi kigge på det samme eksempel som ved den naive. Hvis  $P = a^m$  og  $T = a^n$   
skal vi tjekke om alle hits faktisk er hits, altså får vi  $\Theta((n - m + 1)m)$ .

I praksis forventer vi ikke så mange hits, f.eks. et konstant antal  $k$ . I sådanne  
tilfælde vil køretiden være  $O((n - m + 1) + km) = O(n + m)$ . Risikoen for at  
få et spurious hit ( $t_s \equiv p \pmod{q}$ ) er  $\frac{1}{q}$ , altså kan vi forventer i størrelses ordnen  
 $O\left(\frac{n}{q}\right)$  spurious hits. Da vi i værstetilfælde bruger  $O(m)$  lige meget om det er  
et spurious hit eller et rigtigt hit og der er lineært antal steder hvor  $p = t_s$  fejler.  
Dermed vil vores algoritme køre  $O(n) + O\left(m\left(v + \frac{n}{q}\right)\right)$ , hvor  $v$  er antallet af  
reale hits. Vælger vi  $q \geq |m|$  er køretiden  $O(n)$  hvis  $v = O(1)$ . Altså har vi en  
køretid på  $O(n + m)$ , men da  $n \geq m$  vil dette være  $O(n)$ .

### Endelige Automater

Smart fordi vi kun behøver at kigge på hvert tegn én gang og vi bruge konstant  
( $O(1)$ ) ved hvert tegn, altså  $O(n)$  for  $n$  tegn. Problemet er at vi skal bygge en  
først.

En endelig automat  $M$  består af 5 ting:

- $Q$ : et endeligt antal tilstande
- $q_0$ : start tilstand
- $A \subseteq Q$ : accepterende tilstand
- $\Sigma$ : et endeligt alfabet
- $\delta$ : Transaktionsfunktion, går fra en tilstand til en anden

Den har desuden også en funktion  $\phi$ , kaldet *final-state function*. Denne funktion mapper fra  $\Sigma^*$  over i  $Q$ , sådan at  $\phi(w)$  er den tilstand  $M$  ender op i, efter at have skannet strengen  $w$ .  $\phi$  er defineret ved:

$$\begin{aligned}\phi(\epsilon) &= q_0 \\ \phi(wa) &= \delta(\phi(w), a)\end{aligned}$$

### String-matching Automat

Der findes en endelig automat til alle forekomster af  $P$ . Denne automat skal laves i et forberedelseskridt, før vi kan bruge den til at søge i  $T$ . Vi kan f.eks. have en streng-automat der ser således ud:

Problemet er nu at finde den funktion der sender os fra en tilstand til en anden. Her vil vi definere *suffix funktionen*  $\sigma(x)$ , der er længden af det længste prefix af  $P$ , som også er et suffix til  $x$ .

$$\sigma(x) = \max\{k : P_k \supset x\}$$

Eksempler på dette kunne være et  $P = \mathbf{ab}$ , her ville vi have  $\sigma(\epsilon) = 0$ ,  $\sigma(\mathbf{ccaca}) = 1$  og  $\sigma(\mathbf{ccab}) = 2$ . Vi ved nu at for et mønster på længde  $m$  vil  $\sigma(x) = m$  hvis og kun hvis  $P \supset x$

Vi kan nu definere vores streng-matching automat til at have et sæt tilstande,  $Q \{0, 1, \dots, m\}$ . Start tilstanden er  $q_0$  og den eneste accepterende tilstand er  $T_m$ . For at vise at algoritmen fungerer korrekt, vil vi, for en funktion  $\phi$  for et mønster  $P$  og en tekst  $T[1..n]$  bevise at

$$\phi(T_i) = \sigma(T_i)$$

for  $i = 0, 1, \dots, n$ . Dette vil vi bevise ved induktion på  $i$ . Lad os antage at  $\phi(T_i) = \sigma(T_i)$  og derved bevise at  $\phi(T_{i+1}) = \sigma(T_{i+1})$ . Basis tilfældet, med  $i = 0$  er trivielt, da, hvis vi ikke har skannet nogle tegn, så er  $T_0 = \epsilon \Rightarrow \phi(T_0) = 0 = \sigma(T_0)$ . Lader vi  $\phi(T_i) = q$  og  $a = T[i+1]$ .

$$\begin{aligned}\phi(T_{i+1}) &= \phi(T_i a) \\ &= \delta(\phi(T_i), a) \\ &= \delta(q, a) \\ &= \sigma(P_q a) \\ &= \sigma(T_i a) \\ &= \sigma(T_{i+1})\end{aligned}$$

Hvis vi går ind i tilstand  $q$ , så vil  $q$  være den største værdi således at  $P_q \supset T_i$ . Altså har vi at hvis  $q = m$  hvis og kun hvis mønsteret  $P$  lige er blevet skannet.

En måde at bygge en endelig automat på, er at løbe alfabetet igennem, og tjekke hvor de forskellige tegn vil føre en hen, når man antager at man er kommet  $k$  vej i teksten. F.eks. med  $\Sigma\{\mathbf{a}, \mathbf{b}\}$  og  $m = \mathbf{abaaba}$  vil vi kigge på hvorhen vi kommer fra plads 0. Med et  $\mathbf{a}$  ville vi komme til plads 1, mens vi med et  $\mathbf{b}$  ville blive i 0. Hvis vi bruger denne fremgangsmåde vil vi løbe hele  $\Sigma$  igennem for hver plads i  $m \Rightarrow O(m|\Sigma|)$ . For hver gang vi gør dette risikere vi at skulle gå  $m + 1$  tegn tilbage i rækken for at finde ud af hvor vi skal hoppe hen, og vi risikere at skulle sammenligne  $m$  gange, altså:

$$O(m|\Sigma| \cdot (m + 1) \cdot m) = O(m^3|\Sigma| + m^2|\Sigma|) = O(m^3|\Sigma|)$$

Vi har altså  $O(m^3|\Sigma|)$  forberedelses tid og  $\Theta(n)$  matching tid. Dette kan gøres bedre, med *Knuth-Morris-Pratt-Algoritmen*, men dette er udenfor pensum.