String Matching

Spørgsmål 8 fra Exam Questions

Kevin Vinther

January 6, 2024

Table of Contents i

String Matching

- String Matching er, simpelt, et problem hvor vi er givet en tekst, og vil finde alle forekomster af en streng i teksten. Formelt:
- Vi antager at teksten er en array $T[1..n]$ af længde n og at strengen vi leder efter er en array $P[1..m]$ hvor $m \leq n$.
- Endvidere antager vi at elementer af P og T er karakterer fra alfabetet Σ. (husk til(bage til) formelle sprog.)
- Arraysne P og T kaldes ofte(st) strenge af karakterer.
- Vi siger at et "mønster" (den streng vi leder efter) P forekommer med *shift* s i teksten T.

String Matching ii

- (Eller, ækvivalent, at mønsteret P begynder at forekomme i position $s+1$ i tekst T)
- Dette gælder så længe at $0 \leq s \leq n-m$ og $T[s+1..s+m] = P[1..m]$ (altså, hvis $T[s+j] = P[j]$, for $1 \leq j \leq m$)

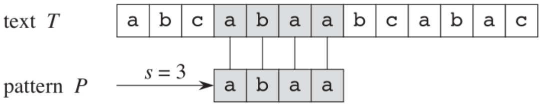


Figure 32.1 An example of the string-matching problem, where we want to find all occurrences of the pattern P = abaa in the text T = abcabaabcabac. The pattern occurs only once in the text, at shift $s = 3$, which we call a valid shift. A vertical line connects each character of the pattern to its matching character in the text, and all matched characters are shaded.

String Matching iii

- Hvis mønsteret P forekommer med shift s i T, kalder vi s et **valid shift**, og ellers et **invalid shift**.
- String matching problemet er problemet om at finde **alle** valide shifts givet et mønster P der forekommer i tekst T.

Oversigt over string-matching algoritmer

Algorithm	**Preprocessing Time**	**Matching Time**
Naive	0	$O((n-m+1)m)$
Rabin-Karp	$\Theta(m)$	$O((n-m+1)m)$
Finite Automaton	$O(m\|\Sigma\|)$	$\Theta(n)$
Knuth-Morris-Pratt	$\Theta(m)$	$\Theta(n)$

Preprocessing i

- Alle algoritmer med undtagelse af den naive laver noget preprocessing baseret på et mønster og finder så alle valide shifts.
- At finde de valide shifts kalder vi “matching”
- Rabin Karp har en langt bedre average-case på trods af at dens worst case er det samme som naiv.

Terminologi i

- Vi betegner sættet af alle endelige strenge formet af alfabetet Σ som værende Σ^*
- Bemærk at ε (den tomme streng) også er en del af Σ^*.
- Længden af en streng, x bliver betegnet som $|x|$
- Concatenation bliver betegnet som xy og har længden $|x| + |y|$. Der er, simpelt, karaktererne i x efterfulgt af karaktererne i y.
- Vi siger at ω er et præfix af en streng x, betegner $\omega \sqsubset x$, hvis $x = \omega y, y \in \Sigma^*$.
- Bemærk at hvis $\omega \sqsubset x$, så $|w| \leq |x|$.

Terminologi ii

- Endvidere siger vi at ω er et suffix af en streng x, betegnet $x \sqsupset x$, hvis $x = y\omega$ for en $y \in \Sigma^*$.
- Som med et præcis, hvis $w \sqsupset x$ så $|\omega| \leq |x|$.
- Bemærk også at $\sqsubset$ og $\sqsupset$ er transitive relationer.

Lemma (31.1 (Overlapping-suffic lemma))

Suppose that x, y, and z are strings such that $x \sqsupset z$ and $y \sqsupset z$. If $|x| \leq |y|$, then $x \sqsupset y$. If $|x| \geq |y|$, then $y \sqsupset x$. If $|x| = |y|$, then $x = y$.

- Vi bruger et grafisk bevis:

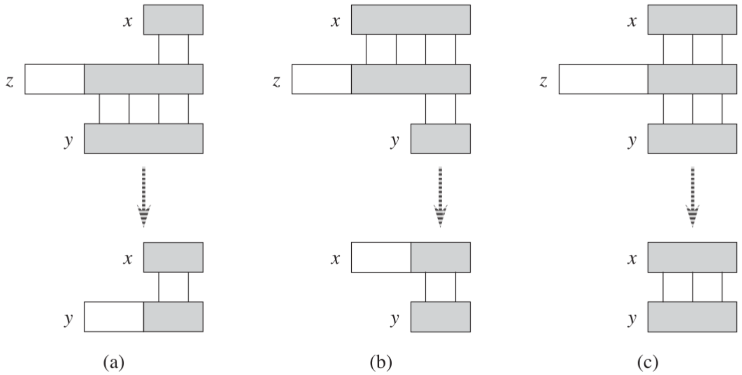


Figure 32.3 A graphical proof of Lemma 32.1. We suppose that $x \sqsupset z$ and $y \sqsupset z$. The three parts of the figure illustrate the three cases of the lemma. Vertical lines connect matching regions (shown shaded) of the strings. **(a)** If $|x| \leq |y|$, then $x \sqsupset y$. **(b)** If $|x| \geq |y|$, then $y \sqsupset x$. **(c)** If $|x| = |y|$, then $x = y$.

Terminologi i

- For lethed af notation, betegner vi en k-karakters præfix $P[1..k]$ af møsteret $P[1..m]$ af P_k. Dermed $P_0 = \varepsilon$, $P_m = P = P[1..M]$ (så længe man går ud fra at længden af P er m.)
- Ved brug af denne notation kan vi sige at string-matching problemet er det hvor vi skal finde alle shifts s i rangen $0 \leq s \leq n - m$ således at $P \sqsupset T_{s+m}$.
- I den pseudokode vi kommer til at bruge, tillader vi to strenge på samme længde til at blive sammenlignet som en primitiv operation (som $+$, - etc).

Terminologi ii

- Hvis strengene bliver sammenlignet venstre til højre og sammenlignningen stopper når et mismatch er fundet, kan vi assume at tiden taget er en lineær funktion af antallet af sammenlignet karakterer fundet.
- Så, for at være præcis, testen `x == y` er antaget at tage tiden $\Theta(t+1)$, hvor t er længden af den længste streng z, således at $z \sqsubset x$, og $z \sqsubset y$. (Vi skriver $\Theta(t+1)$ i stedet for $\Theta(t)$ for at kunne tage den case hvor $t = 0$)

The Naive String-Matching Algorithm

Den Naive String-Matching Algorithme

- Den naive (brute-force) string-matching algoritme finder alle valide shifts ved brug at et loop.
- Loopet tjekker condition $P[1..m] = T[s+1..s+m]$ for hver linje af de $n - m + 1$ mulige værdier a s.

```
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
```

Naive Approach i

- Køretiden på denne algoritme er $O((n - m + 1)m)$ hvilket er ret lort.
- Det kan blive $\Theta(n^2)$ hvis vi har en streng a^n og et mønster a^m, og $m = \lfloor n/2 \rfloor$.
- Til gengæld ingen preprocessing!

Rabin-Karp i

- Randomiseret algoritme, dog bliver den først beskrevet uden randomisering
- Idé: Tænk på P og T som værende tal og sammenlign P og $T_{s+1..s+m}$ ved at sammenligne tallene
- Vi kører i radix-d hvor $d = |\Sigma|$.
- Det betyder at vi bruger tallene i base-d.
- Mere notation!:
- p (lille p): Decimalværdien af $P[1..m]$
- t_s: Decimalværdien af $T[s+1..s+m]$
- Dermed, hvis $p = t_s$ har vi et **match**!

Rabin-Karp ii

- **Antagelse**: Størrelsen af tallene p og t er ikke så store at deres lighed ikke kan findes i konstant tid.
- Hvordan finder vi så værdien af p?

Horner's Rule og udregning af tal i

- Horners rule er en regneregel til at finde værdien af store tal hurtigt, specielt hurtigt i computere.

Horner's Rule og udregning af tal ii

Example of calcolation using Horners rule

$$47632 = 4\cdot 10^4 + 7\cdot 10^3 + 6\cdot 10^2 + 3\cdot 10 + 2\cdot 10^0$$

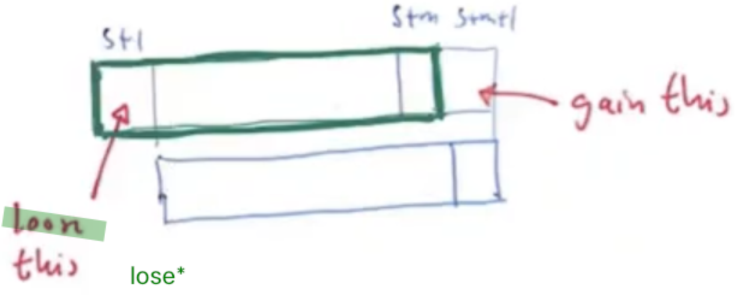
$$= 2\cdot 10^0 + 3\cdot 10^1 + 6\cdot 10^2 + 7\cdot 10^3 + 4\cdot 10^4$$

$$= 2 + 10(3 + 10(6 + 10(7 + 10\cdot 4)))$$

- Dermed:
  $p = P[m] + 10(P[m-1] + 10(P[m-2] + \cdots + 10(P[i])\cdots))$

Horner's Rule og udregning af tal iii

- På samme måde er t_0 udregnet fra $T[1..m]$ i $\Theta(m)$
- Vi kan hurtigt udregne t_{s+1} fra t_s da
  $t_{s+1} = 10(t_s - 10^{m-1}T[s+1]) + T[s+m+1]$



Horner's Rule og udregning af tal iv

- Det tager $O(1)$ konstant tid er udregne det næste t_{s+1}, når 10^{m-1} er udregnet tidligere, som tager $O(m)$ tid (men, som vi ser senere, kan blive gjort i $O(\log m)$ tid.

Rabin-Karp

```
Compute p in time O(m)
Compute t_0 in time O(m) - report match if p = t_0
Compute t_1 in time O(1) - Report match if p = t_1
.
.
.
Compute t_(n-m) in time O(1) - Report match if p = t_(n-m)
```

Rabin-Karp i

- Total tid: $\Theta(m)$ preprocessing (konvertering til tal)
- $\Theta(n - m + 1)$ matching tid.
- Det er dog too-good-to-be-true:(
- Hvad hvis p er for stort til at vi kan sammenligne i konstant tid?

Løsning til køretidsproblemet i

- Vi løser dette problem ved at udregne p og t_s modulo q.
- Vi kan udregne $p \mod q$ i $\Theta(m)$, $t_0 \mod q$ i $\Theta(m)$
- **Hvordan finder vi en god værdi af q?**
- Så stort som muligt. $10q$ skal gerne være værdien af et `word`.
- $10q$ fordi vi gerne vil kunne lave udregningen v.h.a Horner's Rule.
- Ved et generelt alfabet:

Theorem

$\Sigma = \{0, 1, 2, \ldots, d-1\}$ *Chose q such that dq fits in one computer word.* $t_{s+1} = (d(t_s - T[s+1] \cdot h) + T[s+m+1]) \mod q$ *when* $h = d^{m-1} \mod q$

Løsning til køretidsproblemet ii

- Nyt problem:
- "Spurious Hits" $t_s \equiv p \mod q$ men $t_s \neq p$:
- Vi gider ikke de hits! ¿:(
- Vi skal derfor teste alle shifts med $t_s \equiv p \mod q$
- Det er $\Theta(m)$ per test.
- Dermed bliver worst case matching tid en del værre: $\Theta((n - m + 1)m)$

Forventede antal hits i

- Antag at mod q agerer som en tilfældig mapping fra alfabetet til hele tal base q, i.e., $\Sigma^* \to \mathbb{Z}_q$
- Hvis vi antager at alle værdierne mod q er lige sandsynlige, så $p(t_s \equiv p \mod q) = \frac{1}{q}$
- Så er antallet af "spurious hits", i.e., hits vi ikke vil have (falske), $\frac{O(n)}{q} = O(\frac{n}{q})$
- Nu når vi ved det, kan vi så finde den forventerede matchingtid (køretid).

Forventede køretid i

- $O(n) + O(m(v + \frac{n}{q}))$
- Hvor v er antallet af korrekte hits
- Og v er $O(1)$ og $q \geq m$.
- Tiden på dette er $O(n)$
- So den totale tid (med preprocessing og matches) er $O(n + m) = O(n)$ da $n \geq m$
- Yippeee!

- Vi kan også matche strings med DFA'er.
- Jeg går ikke over de helt basale definitioner.
- Vi siger at en endelig maskine M accepterer $w \in \Sigma^*$ hvis, når maskinen har kørt w igennem (spist, ifølge Jørgen), og den sidste state er en accept state.
- M accepterer **ikke** $w \in \Sigma^*$ hvis den ikke ender i en accepting state.
- Vi definerer ϕ til at være **final state funktionen**, som er den state M er i efter den har spist w.
- Så hvis $\phi(w) \in A$ hvor A er sættet af accepting states, så accepterer vi w.

String matching med DFA's ii

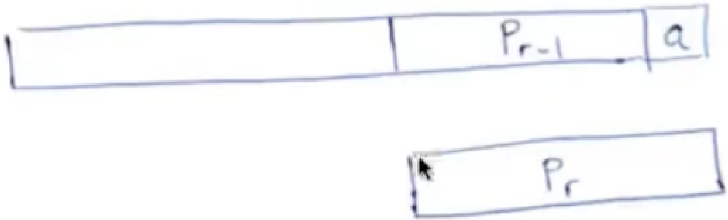
- Vi vil nu bygge en automata lavet til string matching, innovativt nok, kalder vi den for **string matching automata**.

String Matching Automata i

- **Mål**: M accepterer hvert præfix T_{s+m} af en tekst T hvor de sidste $m = |P|$ karakterer af T_{s+m} er lig p.
- Notationstid!
- **Suffix Funktion** for mønster p:
- $\sigma : \Sigma^* \rightarrow \{0, 1, 2 \ldots, m\}$
- $\sigma(x) = \max\{k | P_k \sqsupset x\}$
- σ er *well-defined* da $\varepsilon \sqsupset x \ \ \forall x \in \Sigma^*$
- Well-defined, da der er et værdi for hvert input.

Lemma (32.2)

$\forall x \in \Sigma^*, a \in \Sigma \qquad \sigma(xa) \leq \sigma(x) + 1$



- Så er det tid til at definere *string matching automaton*!
- $M = M(p) \quad p = P[1..m]$

String Matching Automata iii

- Givet at længden af mønstret p har længde m, i så fald vil M have states $Q = \{0, 1, 2, \ldots, m\}, q_0 = 0, A = \{m\}$
- $\sigma(q, a) = \sigma_q(P_a)$
- Vi vælger σ så, hvis automataen er i state q, så er den nuværrende streng T_i satisfier at $P_q \sqsupset T_i$ og $q = \sigma(T_i)$
- I.e., når der er spist T_i, så skal $P_q \sqsupset T_i$ og q den længste suffiks af T_i.