String Matching

Spørgsmål 8 fra Exam Questions

Kevin Vinther

December 19, 2023

Table of Contents i

String Matching

String Matching i

- 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 $1 \leq j \leq m$)

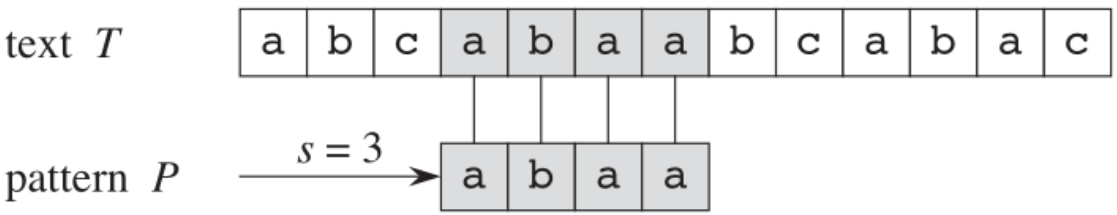

Figure 32.1 An example of the string-matching problem, where we want to find all occurrences of the pattern $P = \texttt{abaa}$ in the text $T = \texttt{abcabaabcabac}$. The pattern occurs only once in the text, at shift $s = 3$, which we call a valid shift. A vertical line connects each character of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

Terminologi iii

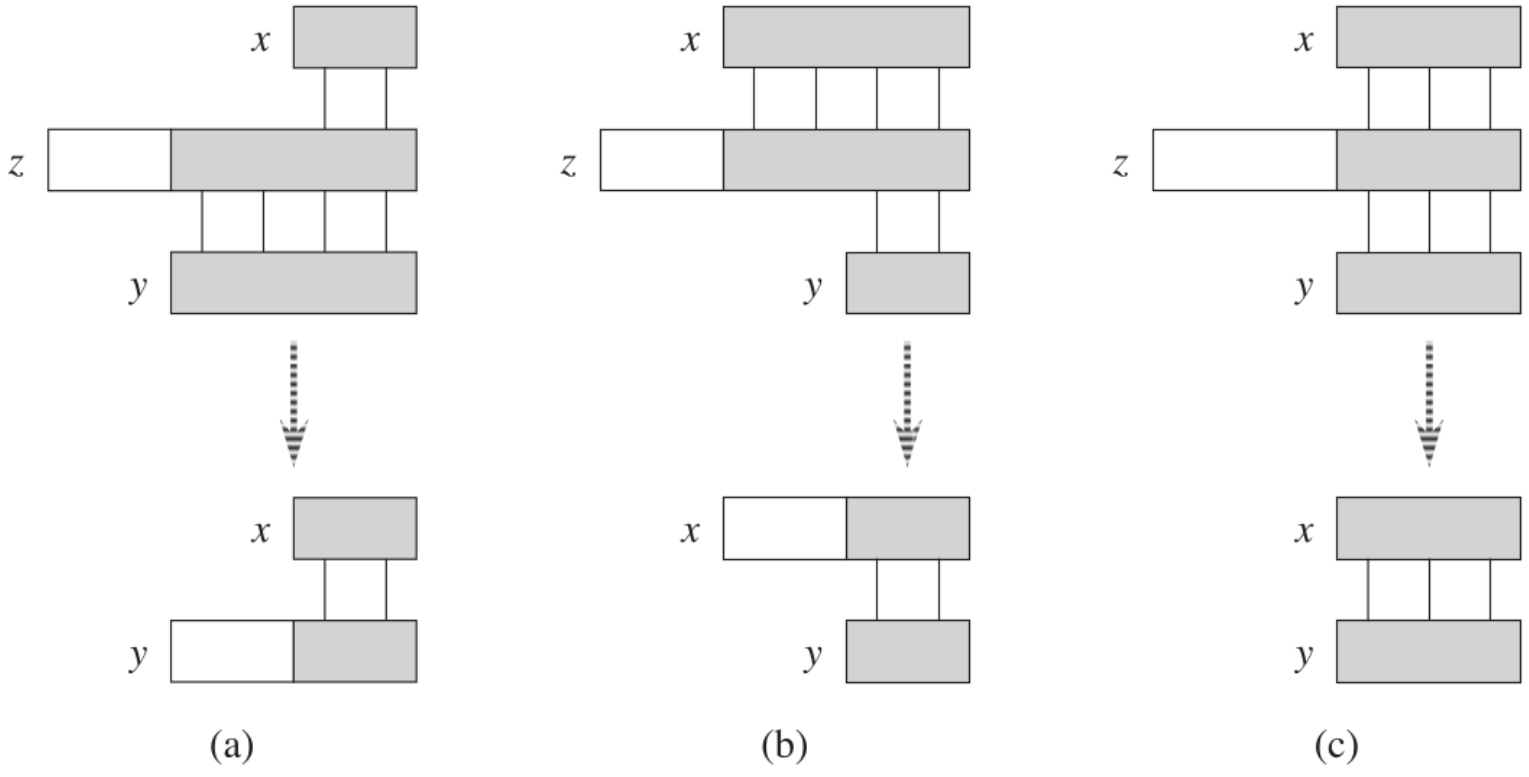

Figure 32.3 A graphical proof of Lemma 32.1. We suppose that $x \sqsupset z$ and $y \sqsupset z$. The three parts of the figure illustrate the three cases of the lemma. Vertical lines connect matching regions (shown shaded) of the strings. **(a)** If $|x| \leq |y|$, then $x \sqsupset y$. **(b)** If $|x|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

```
1  n = T.length
2  m = P.length
3  for s = 0 to n − m
4       if P[1..m] == T[s + 1..s + m]
5            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!

Rabink-Karp i

- Det var da godt nok lort, hva?
- Tid til Rabin-Karp som (måske nok) er bedre!