Algoritmer og Sandsynlighed
Kompendium

Kevin Vinther

January 6, 2024

Contents

1 Basic Counting Problems **2**

2 Inclusion Exclusion **2**

3 Discrete Probability **2**

4 Randomized Algorithms **2**

5 Probabilistic Analysis **2**

6 Indicator Random Variables **2**

7 Universal Hashing **2**

8 String Matching **2**
8.1 Notation 2
8.1.1 Køretids Overview 3
8.2 Naive Algoritme 4
8.2.1 Køretid 4
8.3 Rabin-Karp 5
8.4 Finite Automaton Based 6

9 Flows **7**

10 Min-Cut **7**

1 Basic Counting Problems

2 Inclusion Exclusion

3 Discrete Probability

4 Randomized Algorithms

5 Probabilistic Analysis

6 Indicator Random Variables

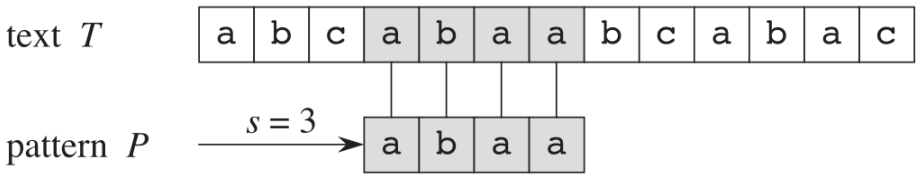
7 Universal Hashing

8 String Matching

8.1 Notation

Jeg tænker ikke at der skal snakkes om det her til eksamen, men følgende er en liste af notation der er nødvendige for forforståelse:

- **Strenge**: Arrays med karakterer (ligesom i programmeringssprog)
- **Shift**: Hvor langt inde i en streng
- $P[1..m]$: Mønster med længde m
- $T[1..n]$: Tekst med længde n
- $T[1..n-m]$: Den tekst vi leder efter. Vi er ikke interesseret i de sidste m, da de er længere end mønsterstrengen.
- **P forekommer med shift s**: Du finder mønstret s karakterer inde i teksten.
- **Validt shift** et shift hvor mønsteret P forekommer
- **Invalidt shift** et shift hvor mønsteret P **ikke** forekommer.



- Σ^* (Sigma-Stjerne) er sættet af alle endelige strenge der kan bliver lavet fra karaktererne i Σ.
- ε, den **tomme streng**, er strengen uden noget indhold. Den er også en del af Σ^*.
- $|x|$ er længden af streng x.
- **Concatenation** af to strenge x og y, skrevet xy har længde $|x| + |y|$ og er karaktererne i x efterfulgt af karaktererne i y.
- **Præfiks** af streng x, denoted $w \sqsubset x$, gælder hvis $x = wy$ hvor $y \in \Sigma^*$, altså, w er en del af streng x i starten af strængen. y er den resterende del af streng x, som ikke er w.
- **Suffiks**: denoted $w \sqsupset x$ omvendt.

Lemma 1 (31.1 (Overlapping-suffix lemma) (Cormen))**.** *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 $x = y$.*

Proof. Se Figur 1 □

Vi antager at tiden det tager for at finde ligheden mellem to strenge er $\Theta(t+1)$ hvor t er størrelsen af den længste streng. $+1$, til hvis $t = 0$.

8.1.1 Køretids Overview

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	$\Theta(m)$	$\Theta(n)$

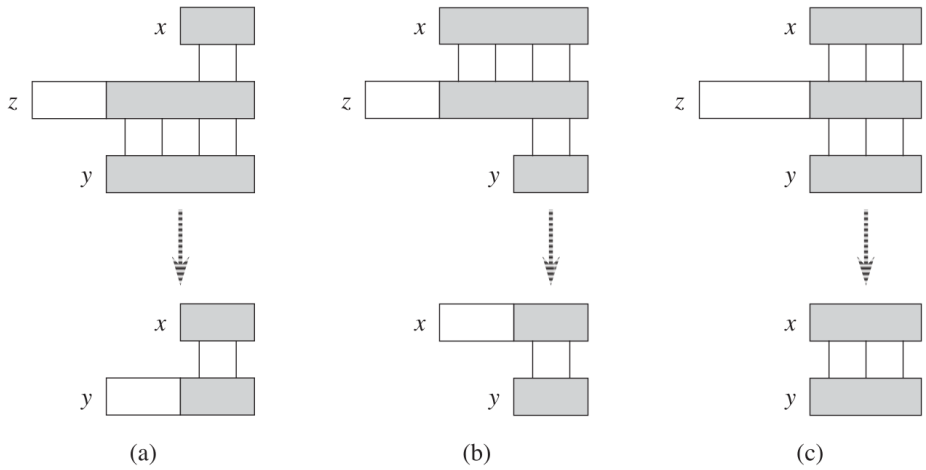


Figure 1: Vi antager at $x \sqsupset z$ og $y \sqsupset z$. De tre dele af figuren illustrerer de tre cases af lemmaet. **(a)** Hvis $|x| \leq |y|$, så $x \sqsupset y$. **(b)** Hvis $|x| \geq |y|$, så $y \sqsupset x$. **(c)** Hvis $|x| = |y|$, så er $x = y$.

8.2 Naive Algoritme

- Hvad er den?
- Hvorfor er den dårlig?
- Hvad er worst-case?

Den naive algoritme er virkelig det, naiv. **Source Code**:

```
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
```

8.2.1 Køretid

Den er virkelig skrald. Køretiden er $O((n-m+1)m)$. Dens worst case sker hvis teksten er a^n og mønsteret der ledes efter er a^n (begge er mængder af a'er, på længde hhv. m og n. I dette tilfælde finder den matches hver gnag, og der tager dermed $O(n^2)$ tid.

Der er **ingen** preprocessing tid, da der ikke skal gøres noget før algoritmen kører.

8.3 Rabin-Karp

- Hvad er hovedidéen?
- Hvorfor er den bedre end naive?

Trods at Rabin-Karp har en worst-case køretid på $\Theta((n-m+1)m)$ er dens gennemsnitlige køretid bedre.

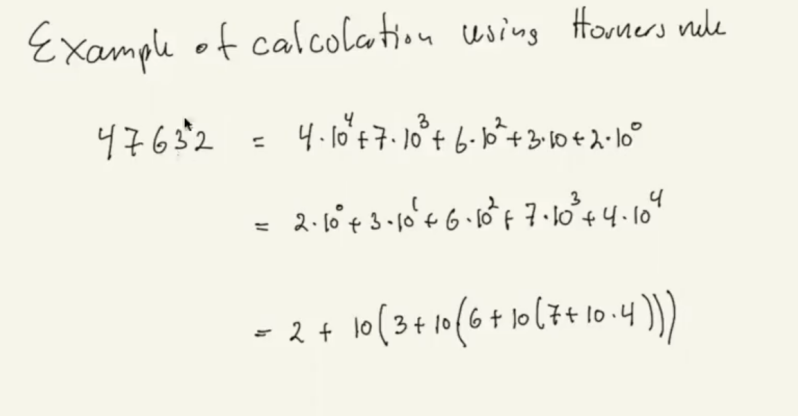
Algoritmen konverterer bogstaverne til tal, i radix-d notation, hvor d er størrelsen på alfabetet, $|\Sigma|$.

I følgende eksempler vil vi gå ud fra at $d = 10$, og $\Sigma = \{0, 1, \ldots, 9\}$. Husk at $P[1..m]$ er mønsteret vi leder efter. Ved rabin-karp skelner vi mellem $P[1..m]$ og p, hvor p er dets decimalværdi. Dvs., hvis $P[1..m] = 1372$, så er $p = 1372$ i decimalværdi. Eksemplet virker forsimplet idet vores alfabet også er tal, men tænk hvis alfabetet var $\Sigma = \{a, b, ...j\}$, i dette tilfælde ville p ikke være ændret, men $P[1..m] = acgb$. Ydermere er teksten $T[1..n]$'s decimal counterpat t_s. Den bliver udregnet på samme måde. Hvis $t_s = p$ så $T[s+1..s+m] = P[1..m]$.

Vi vil gerne have en måde hvorpå vi kan lave alfabetet om til tal, som vi kan regne på. Hvis vi kan konverterer mønsteret $P[1..m]$ til p på $\Theta(m)$ tid, så kan vi konvertere t_s på $\Theta(n-m+1)$ tid. Til at gøre dette bruger vi **Horner's Rule**, som er meget vigtig at kende, se Definition 1.

Definition 1 (Horner's Rule)**.** Horner's Rule er en regel hvorpå du hurtigt (specielt for computere) kan udregne decimaltal. Dette gør du ved at tage det sidste tal der skal udregnes først, derefter tager du tallet på 10'ernes plads, ganger det med 10^1, etc. indtil du er ved d'ende plads, og ganger det med 10^{d1}. Se følgende billede.

[1] Dette gælder kun i base-10. Rabin-karp kører i base-b. Konverter dette til b^d



Noget af det smarteste med Horner's Rule, er at, når du går til næste værdi, så kan du udregne det hurtigt uden at tage det hele om igen. Dette giver køretid $\Theta(n-m)$:

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

Forklaring 2. *Skip dette hvis du ikke har meget tid.* $10^{m-1} \cdot T[s+1]$ *fjerner det højeste ciffer. Ved at gange det med 10 skifter du tallet til venstre med en cifferposition. Ved at tilføje* $T[s+m+1]$ *får du det nye, laveste ciffer.*

Problem! p og t_s er muligvis for storre til at de kan være i et computer `word`. Hvis dette er tilfældet, og P indeholder m karakterer, så tager vi tallet **modulo** q. $p \mod q$ bliver udregnet på $\Theta(m)$ tid (størrelsen af p.) Alle t_s værdier i $\Theta(n-m+1)$ tid.

Hvilken q **skal vi dog vælge?** Simpelt! Vælg et primtal således der er plads til $10q$ i én computer `word`. Derefter kan vi udføre alle udregninger simpelt. Ved at bruge modulo-udregning, ændrer Horner's udregning sig til at blive: $t_{s+1} = (d(t_s - T[s+1]h) + T[s+m+1]) \mod q$, hvor $h \equiv d^{m-1} (\mod q)$ er værdien af ciffret 1 i højeste position.

Problem igen! Hvad hvis $p \equiv t_s$, men $P[1..m] \neq T[s+1..n-m]$? Altså, tallene er ens, men de er strengene ikke grundet modulo? Dette kalder vi et **spurious hit**, og er pisse irriterende, men desværre end nødvendig onde. Derfor, når vi finder et **hit** om det er spurious eller ej, så tjekker vi også strengene.

8.4 Finite Automaton Based

- Hvordan laver man en DFA?

```
RABIN-KARP-MATCHER(T, P, d, q)
 1  n = T.length
 2  m = P.length
 3  h = d^(m-1) mod q
 4  p = 0
 5  t_0 = 0
 6  for i = 1 to m                    // preprocessing
 7       p = (dp + P[i]) mod q
 8       t_0 = (dt_0 + T[i]) mod q
 9  for s = 0 to n - m                // matching
10       if p == t_s
11            if P[1..m] == T[s + 1..s + m]
12                 print "Pattern occurs with shift" s
13       if s < n - m
14            t_(s+1) = (d(t_s - T[s + 1]h) + T[s + m + 1]) mod q
```

Figure 2: Rabin Karp Algoritmen

9 Flows

10 Min-Cut