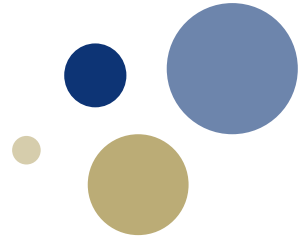
NTNU

Kunnskap for en bedre verden



Tekstsøk, Datakompresjon

Helge Hafting

IIE

27. oktober 2016

Anvendelser for tekstsøk

- Fritekstsøk i dokumenter, nettsider og lignende
- Fritekstsøk i databaser
- Søkemotorer
- Søke etter repeterte strenger for datakompresjon
- DNA-matching

En enkel naiv algoritme

Tekst: rabarbra (lengde n)
Søkeord: bra (lengde m)

Skyv søkeordet langs teksten, se om det passer

- tegn som passer, vises med **fet skrift**
- første feil med *kursiv*
- dobbeltløkke for $n - m$ posisjoner, og m tegn i søkeordet.

Forsøk	r	a	b	a	r	b	r	a
0	*b*	r	a					

En enkel naiv algoritme

Tekst: rabarbra (lengde n)
Søkeord: bra (lengde m)

Skyv søkeordet langs teksten, se om det passer

- tegn som passer, vises med **fet skrift**
- første feil med *kursiv*
- dobbeltløkke for $n - m$ posisjoner, og m tegn i søkeordet.

Forsøk	r	a	b	a	r	b	r	a
1		*b*	r	**a**				

En enkel naiv algoritme

Tekst: rabarbra (lengde n)
Søkeord: bra (lengde m)

Skyv søkeordet langs teksten, se om det passer

- tegn som passer, vises med **fet skrift**
- første feil med *kursiv*
- dobbeltløkke for $n - m$ posisjoner, og m tegn i søkeordet.

Forsøk	r	a	b	a	r	b	r	a
2			**b**	*r*	a			

En enkel naiv algoritme

Tekst: rabarbra (lengde n)
Søkeord: bra (lengde m)

Skyv søkeordet langs teksten, se om det passer

- tegn som passer, vises med **fet skrift**
- første feil med *kursiv*
- dobbeltløkke for $n - m$ posisjoner, og m tegn i søkeordet.

Forsøk	r	a	b	a	r	b	r	a
3				*b*	**r**	a		

En enkel naiv algoritme

Tekst: rabarbra (lengde n)
Søkeord: bra (lengde m)

Skyv søkeordet langs teksten, se om det passer

- tegn som passer, vises med **fet skrift**
- første feil med *kursiv*
- dobbeltløkke for $n - m$ posisjoner, og m tegn i søkeordet.

Forsøk	r	a	b	a	r	b	r	a
4					*b*	r	a	

En enkel naiv algoritme

Tekst: rabarbra (lengde n)
Søkeord: bra (lengde m)

Skyv søkeordet langs teksten, se om det passer

- tegn som passer, vises med **fet skrift**
- første feil med *kursiv*
- dobbeltløkke for $n - m$ posisjoner, og m tegn i søkeordet.

Forsøk	r	a	b	a	r	b	r	a
5						**b**	**r**	**a**

En enkel naiv algoritme

Tekst: rabarbra (lengde n)
Søkeord: bra (lengde m)

Hele greia, $O(n \cdot m), \Omega(n)$

Forsøk	r	a	b	a	r	b	r	a
0	*b*	r	a					
1		*b*	r	**a**				
2			**b**	*r*	a			
3				*b*	**r**	a		
4					*b*	r	a	
5						**b**	**r**	**a**

Boyer-Moore

— Se på *siste* tegn i søketeksten først
— Hvis det ikke passer, flytt søketeksten *så langt vi kan*

	r	a	b	a	r	b	r	a
0	b	r	*a*					
1			b	r	*a*			
2				b	r	*a*		
3						**b**	**r**	**a**

— Hvis det passer, se på nestsiste osv.

Regelen om upassende tegn

— Hvis tegnet ikke fins i søketeksten, kan vi flytte *m* steg frem:

	m	e	t	e	o	r	i	t	t	s	t	e	i	n
0	s	t	e	i	*n*									
1						s	t	e	i	*n*				
2										**s**	**t**	**e**	**i**	**n**

— Hvis tegnet fins til venstre i søkeordet, kan vi flytte ordet så det passer med teksten

— Vi har vi en tabell for hvor mye vi kan flytte

— I praksis en tabell for hele alfabetet, hvor de fleste tegn gir et flytt på *m*. (Regel om «upassende tegn»)

— Tabellen lager vi ved å pre-prosessere søketeksten

— Tegn som fins i søketeksten, gir kortere flytt

- En «s» i siste posisjon gir flytt på $m - 1$, fordi ordet starter på «s»

— $\Omega(n/m)$ for søket. Mye bedre!

Upassende tegn, fortsatt

— Hvis tegnet ikke fins i søketeksten, kan vi flytte *m* steg frem,

- hvis mismatch var på *siste* tegn i søketeksten
- med mismatch på *nestsiste* tegn kan vi flytte $m - 1$ steg
- ved mismatch på nestnestsiste, flytter vi $m - 2$ steg osv.

	m	e	t	e	o	r	i	t	t	s	t	e	i	n
0	m	e	*n*	e										
1				m	e	n	*e*							

— Vi trenger altså en todimensjonal tabell:

- En indeks er det upassende tegnet
- Den andre indeksen er posisjonen i søketeksten
- Verdien i cellen er hvor langt vi kan flytte fremover

Upassende tegn, lage tabellen

```
For hver posisjon p i søketeksten
   For hvert tegn x i alfabetet
      let mot start i søketeksten fra p
      hvis vi finner x etter i steg,
      sett Tab[p][x] = i
      hvis vi ikke finner x, Tab[p][x]=p+1
```

Regel om passende endelse

	r	e	n	n	e	n	e	
0	e	n	*e*					
1		e	n	*e*				
2			*e*	**n**	**e**			
					e	**n**	**e**	

— 0,1: Når siste posisjon treffer «n», kan vi bare flytte ett steg
— 2: Feil i første posisjon
  - Regel om «upassende tegn» lar oss bare flytte ett hakk
— Regel om «passende endelse» lar oss flytte to hakk her
— «ne» passet, og «ene» overlapper med seg selv
— Vi slår opp både «upassende tegn» og passende endelse», og bruker regelen som gir det lengste hoppet.

Passende endelse, tabell

— Tabellen for «passende endelse»
  - index er hvor mange tegn som passet
  - verdien i cellen er hvor langt vi kan flytte

— Lages ved å prøve ut om søketeksten overlapper med seg selv
  - ofte gjør den ikke det, og vi får lange hopp!

Galil sin regel

— Hvis vi søker etter «aaa» i «aaaaaa…», har vi dessverre $O(n \cdot m)$
  - søkeordet passer overalt, de samme a-ene sjekkes flere ganger

— Galil fant en måte å unngå unødvendige sammenligninger:
  - Når vi flytter søkeordet kortere enn den delen av søkeordet vi allerede har sjekket, trenger vi ikke sjekke det overlappende området omigjen.
  - Korte flytt skjer fordi søkeordet delvis matcher seg selv. Hvis det ikke hadde passet, hadde vi flyttet lenger.

Teksten	.	.	.	O	l	a	l	a	.	.	.
Mismatch O/a			l	*a*	**l**	**a**	**l**	**a**			
Nytt forsøk					**l**	**a**	**l**	**a**	l	a	

— Programmet trenger ikke sjekke den oransje regionen omigjen

— Dermed: $O(n)$ og $\Omega(n/m)$ for tekstsøk

Lenker

- Boyer og Moore sin artikkel:
  `http://www.cs.utexas.edu/~moore/publications/fstrpos.pdf`
- Wikipedia:
  `https://en.wikipedia.org/wiki/Boyer_moore_string_search_algorithm`
- Animasjon (Fyll ut, og velg Boyer-Moore) Trenger java
  `http://www.cs.pitt.edu/~kirk/cs1501/animations/String.html`
- Demonstrasjon på Moore sin nettside:
  `http://www.cs.utexas.edu/users/moore/best-ideas/string-searching/fstrpos-example.html`

Lempel-Ziv kompresjon

- Leser gjennom fila
- Input kopieres til output
- Hvis en lang nok sekvens kommer omigjen:
  - dropp den, skriv heller en referanse til output
  - format: repeter X tegn, som vi har sett Y tegn tidligere
- Hjelper hvis sekvensen er lenger enn en slik referanse
- Søker bakover i et sirkulært buffer
- Output kan komprimeres videre med Huffman-koding (DEFLATE-algoritmen, som brukes i zip.)

Bakover-referanser

- Må være *kompakt*
  - ellers kan vi ikke referere til korte strenger
  - f.eks. 2–3 byte
- Å «se» langt bakover i datastrømmen, gir større sjanse for å finne repetisjoner.
  - men også lenger kjøretid
  - påvirker formatet på referansene våre
    - 1 byte kan gå 255 tegn bakover
    - 2 byte kan gå 65 536 tegn bakover
    - 3 byte kan gå 16 777 215 tegn bakover
- I blant kan vi ikke komprimere
  - Må derfor også ha en måte å si:
  - Her kommer X bytes ukomprimerte data
  - Slik informasjon tar også plass!

Hva kan komprimeres?

— Vurdering:
  - Skal dette være en del av en større ukomprimert blokk?
  - Evt. bakover-ref + header for kortere ukomprimert blokk

— Det vi komprimerer må altså være lenger enn samlet lengde for:
  - en bakover-referanse
  - header for en ukomprimert blokk

— Vi komprimerer ikke svært korte strenger, det hjelper ikke!

Eksempel

— Eksempeltekst:
Problemer, problemer. Alltid problemer!
Dette er dagens problem. Problemet er
å komprimere problematisk tekst.

— Eksempeltekst med avstander:
Problemer,[10] problemer[20]. Alltid p[30]roblemer!
[40]Dette er d[50]agens prob[60]lem. Probl[70]emet er
å [80]komprimere[90] problemat[100]isk tekst.[110]

— 110 tegn, inkludert linjeskift og blanke.

Eksempel

- Eksempeltekst med avstander:
  Problemer,[10] problemer[20]. Alltid p[30]roblemer!
  [40]Dette er d[50]agens prob[60]lem. Probl[70]emet er
  å [80]komprimere[90] problemat[100]isk tekst.[110]
- Komprimert:
  [12]Problemer, p[-11,8][8]. Alltid[-18,10][17]!
  Dette er dagens[-27,7][2]. [-65,8][17]t er
  å komprimere[-35,8][12]atisk tekst.
- Før komprimering, 110 tegn.
- Med 1 byte per tallkode, 84 tegn.
  Vi sparte 110-84=26 tegn, eller 23%

Kjøretid

— For hver tegnposisjon i input, må vi søke etter lengste match i bufferet.
— Fil med n tegn, sirkulært buffer med størrelse m.
— Teste alle posisjoner, i verste fall $O\left(nm^2\right)$
— I praksis går det bedre, særlig hvis data varierer en del
— Kan bruke Boyer-Moore tekstsøk for bedre kjøretid.

Lenker

— Lempel og Ziv sin artikkel:
`http://www.cs.duke.edu/courses/spring03/cps296.5/papers/ziv_lempel_1977_universal_algorithm.pdf`
— Wikipedia:
`https://en.wikipedia.org/wiki/Lempel%E2%80%93Ziv`