



NTNU

Kunnskap for en bedre verden

TEKSTSØK, DATAKOMPRESJON

Helge Hafting

22. oktober 2021

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	<i>b</i>	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		<i>b</i>	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	<i>r</i>	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				<i>b</i>	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					<i>b</i>	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	<i>b</i>	r	a					
1		<i>b</i>	r	a				
2			b	<i>r</i>	a			
3				<i>b</i>	r	a		
4					<i>b</i>	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	<i>a</i>					
1			b	r	<i>a</i>			
2				b	r	<i>a</i>		
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 *nest siste* 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	<i>n</i>	e										
1				m	e	n	<i>e</i>							

- ▶ 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>

Run-length coding

- ▶ Enkleste form for datakompresjon
- ▶ En serie repetisjoner erstattes med et antall:
 - ▶ ABIIIIIIIIIIIBBBCDEFFFGH → AB12I3BCDE3FGH

Run-length coding

- ▶ Enkleste form for datakompresjon
- ▶ En serie repetisjoner erstattes med et antall:
 - ▶ ABIIIIIIIIIIIBBBCDEFFFGH → AB12I3BCDE3FGH
- ▶ I praksis litt mer komplisert
 - ▶ det kan jo være sifre i det vi komprimerer
 - ▶ ser vanligvis på «bytes», ikke «tekst»
 - ▶ må kunne skille mellom data og metadata

Run-length coding

- ▶ Enkleste form for datakompresjon
- ▶ En serie repetisjoner erstattes med et antall:
 - ▶ `ABIIIIIIIIIIIBBBCDEFFFGH` → `AB12I3BCDE3FGH`
- ▶ I praksis litt mer komplisert
 - ▶ det kan jo være sifre i det vi komprimerer
 - ▶ ser vanligvis på «bytes», ikke «tekst»
 - ▶ må kunne skille mellom data og metadata
- ▶ Eks., bruker negativ byte for ukomprimerte sekvenser:
 - ▶ `ABIIIIIIIIIIIBBBCDEFFFGH` → `[-2]AB[12]I[3]B[-3]CDE[3]F[-2]GH`
 - ▶ 25 byte ble redusert til 16

Run-length coding

- ▶ Enkleste form for datakompresjon
- ▶ En serie repetisjoner erstattes med et antall:
 - ▶ ABIIIIIIIIIIIBBBCDEFFFGH → AB12I3BCDE3FGH
- ▶ I praksis litt mer komplisert
 - ▶ det kan jo være sifre i det vi komprimerer
 - ▶ ser vanligvis på «bytes», ikke «tekst»
 - ▶ må kunne skille mellom data og metadata
- ▶ Eks., bruker negativ byte for ukomprimerte sekvenser:
 - ▶ ABIIIIIIIIIIIBBBCDEFFFGH → [-2]AB[12]I[3]B[-3]CDE[3]F[-2]GH
 - ▶ 25 byte ble redusert til 16
- ▶ Kan ikke komprimere ABABABABABAB...

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

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 peke 255 tegn bakover
 - ▶ 2 byte kan peke 65 536 tegn bakover
 - ▶ 3 byte kan peke 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 problemer²⁰. Alltid p³⁰roblemer!
⁴⁰Dette er d⁵⁰agens prob⁶⁰lem. Probl⁷⁰emet er
å ⁸⁰komprimere⁹⁰ problemat¹⁰⁰isk tekst.¹¹⁰
- ▶ 110 tegn, inkludert linjeskift og blanke.

Eksempel

- ▶ Eksempeltekst med avstander:
Problemer,¹⁰ problemer²⁰. Alltid p³⁰roblemer!
⁴⁰Dette er d⁵⁰agens prob⁶⁰lem. Probl⁷⁰emet er
å ⁸⁰komprimere⁹⁰ problemat¹⁰⁰isk 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%
- ▶ se også Lz-demo

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 $O(nm^2)$
- ▶ I praksis går det bedre, særlig hvis data varierer en del
- ▶ Kan bruke Boyer-Moore tekstsøk for bedre kjøretid.

- ▶ 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>

Kombinere LZ og Huffman

- ▶ LZ leser input, og skriver
 - ▶ bakover-referanser
 - ▶ sekvenser med ukomprimerte tegn
- ▶ ukomprimerte tegn telles opp, og komprimeres videre med Huffmannkoding

LZW – Lempel Ziv Welsh

- ▶ Ligner LZ. Teoretisk samme kompresjon. Lettere å speede opp.
- ▶ Leser ett og ett tegn
- ▶ Bygger en ordliste (dictionary) underveis
 - ▶ til å begynne med, alle 1-byte «ord»
- ▶ Finn et (lengst mulig) ord, skriv ordnummeret (med færrest mulig bits!)
 - ▶ lagre nytt «ord» = dette ordet + neste tegn
- ▶ Kompresjon hvis ordene blir lengre enn numrene
- ▶ LZW+Huffman → Deflate (brukt i zip)

LZW – Lempel Ziv Welsh

- ▶ Ligner LZ. Teoretisk samme kompresjon. Lettere å speede opp.
- ▶ Leser ett og ett tegn
- ▶ Bygger en ordliste (dictionary) underveis
 - ▶ til å begynne med, alle 1-byte «ord»
- ▶ Finn et (lengst mulig) ord, skriv ordnummeret (med færrest mulig bits!)
 - ▶ lagre nytt «ord» = dette ordet + neste tegn
- ▶ Kompresjon hvis ordene blir lengre enn numrene
- ▶ LZW+Huffman → Deflate (brukt i zip)
- ▶ Se eksempel «lzw»

Kombinere LZW og Huffman

- ▶ LZW
 - ▶ leser input,
 - ▶ bygger en dictionary,
 - ▶ skriver «ordnumre»
- ▶ Noen «ord» forekommer oftere enn andre
- ▶ Programmet finner antall (frekvenser) for ulike ordnumre,
 - ▶ skriver Huffmankoder i stedet for ordnumre
 - ▶ ord som forekommer ofte, får kortere koder

BZip2 blokk-komprimering

- ▶ Komprimerer mer enn LZ-algoritmene
 1. run-length coding
 2. Burrows-Wheeler transformasjon (hoveddel)
 3. Move-To-Front transformasjon (MFT)
 4. run-length coding igjen
 5. Huffmannkoding

Burrows Wheeler transformasjonen (BWT)

- ▶ Hoveddelen av BZ2 (blokksorteringen)
- ▶ Dette steget komprimerer ikke selv, men transformerer en blokk (typisk 900kB)
- ▶ Transformerer repeterte sekvenser (som ord) til repeterte tegn
- ▶ Repeterte *tegn* er lettere å komprimere videre!
- ▶ Transformasjonen er reversibel (for dekomprimering)

Eksempel, Burrows-Wheeler Transformasjon

- ▶ BWT på ordet «refererer●». Tegnet «●» markerer slutten

Rotasjoner

Sortert

refererer● efererer●r

●refererer ererer●ref

r●referere erer●refer

er●referer er●referer

- ▶ rer●refere fererer●re

erer●refer refererer●

rerer●refe rerer●refe

ererer●ref rer●refere

fererer●re r●referere

efererer●r ●refererer

- ▶ BWT er siste kolonne med tegn fra sortert liste, «rfrre●eer»

- ▶ Nå har vi mange like tegn ved siden av hverandre,

- ▶ lettere å komprimere med run-length coding

- ▶ Se også bw brukt på diverse filer

Reversere Burrows-Wheeler transformasjonen

- ▶ Hvordan gå fra «rfrre•eer» til «referere•»?
- ▶ Vet at «rfrre•eer» er siste kolonne i sortert liste
- ▶ Lista bestod av ulike *rotasjoner* av *samme* ord
 - ▶ alle kolonner inneholder de samme tegnene
- ▶ Lista var sortert
 - ▶ første kolonne må altså ha de samme tegnene, sortert
 - ▶ altså «eeeefrrr•»
- ▶ Vi har nå to kolonner, i ei liste over rotasjoner
 - ▶ kan rotere sidelengs, så siste kolonne blir første, og første blir andre
 - ▶ dette er fortsatt en del av løsningen
 - ▶ sorterer vi dette, har vi de *to første* kolonnene
 - ▶ så kan vi legge på siste kolonne igjen
 - ▶ vi har nå tre kolonner. Repeter til vi har alle!
- ▶ Riktig rad er den som har «•» på siste plass

Animasjon, reversere Burrows-Wheeler transformasjonen

- ▶ Hvordan gå fra «rfrre•eeer» til «referere•»?
- ▶ Legg til siste

r
f
r
r
e
•
e
e
e
r

Animasjon, reversere Burrows-Wheeler transformasjonen

- ▶ Hvordan gå fra «rfrre•eeer» til «referere•»?
- ▶ Rotere mot høyre

r
f
r
r
e
•
e
e
e
r

Animasjon, reversere Burrows-Wheeler transformasjonen

- ▶ Hvordan gå fra «rfrre•eeer» til «referere•»?
- ▶ Sortere

e

e

e

e

f

r

r

r

r

•

Animasjon, reversere Burrows-Wheeler transformasjonen

- ▶ Hvordan gå fra «rfrre•eeer» til «referere•»?
- ▶ Legg til siste

e	r
e	f
e	r
e	r
f	e
r	•
r	e
r	e
r	e
•	r

Animasjon, reversere Burrows-Wheeler transformasjonen

- ▶ Hvordan gå fra «rfrre•eeer» til «referere•»?
- ▶ Rotere mot høyre

re

fe

re

re

ef

•r

er

er

er

r•

Animasjon, reversere Burrows-Wheeler transformasjonen

- ▶ Hvordan gå fra «rfrre•eeer» til «referere•»?
- ▶ Sortere

ef

er

er

er

fe

re

re

re

r•

•r

Animasjon, reversere Burrows-Wheeler transformasjonen

- ▶ Hvordan gå fra «rfrre•eeer» til «referere•»?
- ▶ Legg til siste

ef	r
er	f
er	r
er	r
fe	e
re	•
re	e
re	e
r•	e
•r	r

Animasjon, reversere Burrows-Wheeler transformasjonen

- ▶ Hvordan gå fra «rfrre•eeer» til «referere•»?
- ▶ Rotere mot høyre

ref
fer
rer
rer
efe
•re
ere
ere
er•
r•r

Animasjon, reversere Burrows-Wheeler transformasjonen

- ▶ Hvordan gå fra «rfrre•eeer» til «referere•»?
- ▶ Sortere

e f e

e r e

e r e

e r •

f e r

r e f

r e r

r e r

r • r

• r e

Animasjon, reversere Burrows-Wheeler transformasjonen

- ▶ Hvordan gå fra «rfrre•eeer» til «referere•»?
- ▶ Legg til siste

e f e	r
e r e	f
e r e	r
e r •	r
f e r	e
r e f	•
r e r	e
r e r	e
r • r	e
• r e	r

Animasjon, reversere Burrows-Wheeler transformasjonen

- ▶ Hvordan gå fra «rfrre•eeer» til «referere•»?
- ▶ Rotere mot høyre

refe
fere
rere
rer•
efer
•ref
erer
erer
er•r
r•re

Animasjon, reversere Burrows-Wheeler transformasjonen

- ▶ Hvordan gå fra «rfrre•eeer» til «referere•»?
- ▶ Sortere

efer

erer

erer

er•r

fere

refe

rere

rer•

r•re

•ref

Animasjon, reversere Burrows-Wheeler transformasjonen

- ▶ Hvordan gå fra «rfrre•eeer» til «referere•»?
- ▶ Legg til siste

efer	r
erer	f
erer	r
er•r	r
fere	e
refe	•
rere	e
rer•	e
r•re	e
•ref	r

Animasjon, reversere Burrows-Wheeler transformasjonen

- ▶ Hvordan gå fra «rfrre•eere» til «referere•»?
- ▶ Rotere mot høyre

refer
ferer
rerer
rer•r
efere
•refe
erere
erer•
er•re
r•ref

Animasjon, reversere Burrows-Wheeler transformasjonen

- ▶ Hvordan gå fra «rfrre•eeer» til «referere•»?
- ▶ Sortere

efere

erere

erer•

er•re

ferer

refer

rerer

rer•r

r•ref

•refe

Animasjon, reversere Burrows-Wheeler transformasjonen

- ▶ Hvordan gå fra «rfrre•eeer» til «referere•»?
- ▶ Legg til siste

efere	r
erere	f
erer•	r
er•re	r
ferer	e
refer	•
rerer	e
rer•r	e
r•ref	e
•refe	r

Animasjon, reversere Burrows-Wheeler transformasjonen

- ▶ Hvordan gå fra «rfrre•eere» til «referere•»?
- ▶ Rotere mot høyre

refere
ferere
rerer•
rer•re
eferer
•refer
ererer
erer•r
er•ref
r•refe

Animasjon, reversere Burrows-Wheeler transformasjonen

- ▶ Hvordan gå fra «rfrre•eeer» til «referere•»?
- ▶ Sortere

eferer

ererer

erer•r

er•ref

ferere

refere

rerer•

rer•re

r•refe

•refer

Animasjon, reversere Burrows-Wheeler transformasjonen

- ▶ Hvordan gå fra «rfrre•eeer» til «referere•»?
- ▶ Legg til siste

eferer	r
ererer	f
erer•r	r
er•ref	r
ferere	e
refere	•
rerer•	e
rer•re	e
r•refe	e
•refer	r

Animasjon, reversere Burrows-Wheeler transformasjonen

- ▶ Hvordan gå fra «rfrre•eeer» til «referere•»?
- ▶ Rotere mot høyre

referer
fererer
rerer•r
rer•ref
eferere
•refere
ererer•
erer•re
er•refe
r•refer

Animasjon, reversere Burrows-Wheeler transformasjonen

- ▶ Hvordan gå fra «rfrre•eeer» til «referere•»?
- ▶ Sortere

eferere
ererer•
erer•re
er•refe
fererer
referer
rerer•r
rer•ref
r•refer
•refere

Animasjon, reversere Burrows-Wheeler transformasjonen

- ▶ Hvordan gå fra «rfrre•eeer» til «referere•»?
- ▶ Legg til siste

eferere	r
ererer•	f
erer•re	r
er•refe	r
fererer	e
referer	•
rerer•r	e
rer•ref	e
r•refer	e
•refere	r

Animasjon, reversere Burrows-Wheeler transformasjonen

- ▶ Hvordan gå fra «rfrre•eerr» til «referere•»?
- ▶ Rotere mot høyre

referere
fererer•
rerer•re
rer•refe
efererer
•referer
ererer•r
erer•ref
er•refer
r•refere

Animasjon, reversere Burrows-Wheeler transformasjonen

- ▶ Hvordan gå fra «rfrre•eeer» til «referere•»?
- ▶ Sortere

efererer
ererer•r
erer•ref
er•refer
fererer•
referere
rerer•re
rer•refe
r•refere
•referer

Animasjon, reversere Burrows-Wheeler transformasjonen

- ▶ Hvordan gå fra «rfrre•eeer» til «referere•»?
- ▶ Legg til siste

```
efererer r  
ererer•r f  
erer•ref r  
er•refer r  
fererer• e  
referere •  
rerer•re e  
rer•refe e  
r•refere e  
•referer r
```

Animasjon, reversere Burrows-Wheeler transformasjonen

- ▶ Hvordan gå fra «rfrre•eeer» til «referere•»?
- ▶ Rotere mot høyre

refererer
fererer•r
rerer•ref
rer•refer
efererer•
•referere
ererer•re
erer•refe
er•refere
r•referer

Animasjon, reversere Burrows-Wheeler transformasjonen

- ▶ Hvordan gå fra «rfrre•eeer» til «referere•»?
- ▶ Sortere

efererer•
ererer•re
erer•refe
er•refere
fererer•r
refererer
rerer•ref
rer•refer
r•referer
•referere

Animasjon, reversere Burrows-Wheeler transformasjonen

- ▶ Hvordan gå fra «rfrre•eeer» til «referere•»?
- ▶ Legg til siste

efererer•r

ererer•ref

erer•refer

er•referer

fererer•re

refererer• ← Der

rerer•refe

rer•refere

r•referere

•refererer

Move-to-front transformasjonen

- ▶ Komprimerer ikke data, men forbereder
- ▶ Initialiserer en tabell med alle byte-verdier. $t[0]=0$, $t[1]=1$, $t[2]=2$, ...
- ▶ Leser ett og ett tegn fra input
 - ▶ finn tegnet i tabellen, skriv index til output
 - ▶ flytt tegnet vi fant til første plass i tabellen (move to front)
- ▶ input: caaaaacbbbbbabababab

inn:caaaaacbbbbbabababab

ut:21000012000021111111

tabell

0: aca....cb....abababab

1: bac....ac....babababa

2: cbb....ba....c.....

3: ddd....dd....d.....

- ▶ Alle repeterte tegn blir til nuller
- ▶ Korte repeterende sekvenser blir små tall
- ▶ Lett å gå andre veien ved utpakking

MTF gir mer effektiv Huffmankoding

► Eksempel

inn: caaaaacbbbbbbbbaaaabb

ut: 21000012000000200010

	Frekv. før		Frekv. etter
► a	9		0 14
b	9		1 3
c	2		2 3

► Før: like mange «a» som «b»

► Etter: overvekt av «0», som kan få kortere kode.

- ▶ Burrows-Wheeler sorterer så vi får mange repetisjoner
 - ▶ 900 kB blokkstørrelse
- ▶ Move-to-front gjør ulike repetisjoner om til nuller
- ▶ Deretter fungerer run-length coding veldig bra!
- ▶ Huffmannkoding av det som blir igjen

Adaptiv kompresjon

- ▶ Huffmankoding av ei fil, bruker samme koding for hele fila
- ▶ Ei fil kan bestå av ulike deler (f.eks. norsk+engelsk)
 - ▶ Ulike deler har ulik bokstavfordeling
 - ▶ De komprimeres best med ulike Huffman-trær

Kompresjon og AI

- ▶ Noen forskere mener datakompresjon og AI er samme problem
 - ▶ AI: det *korteste* programmet som oppfører seg intelligent
- ▶ Å oppdage repeterte mønstre (kan nyttes for kompresjon) krever intelligens
- ▶ Mer intelligens gir bedre kompresjon
- ▶ Desimalene i π er et vanskelig datasett å komprimere. (mye variasjon) Men:
 - ▶ vi kjenner rekkeutviklinger som genererer π .
 - ▶ Et program med endelig lengde, kan generere hele rekka. ∞ kompresjon!
- ▶ Ei zipfil er vanskelig å komprimere, selv om det fins bedre kompresjon enn zip
 - ▶ Hvis vi pakker ut zipfila, kan vi komprimere bedre med bz2
 - ▶ Å oppdage at noe er zip-komprimert, og dermed kan behandles slik, krever intelligens...