

Velkommen

- **Forelesere:**

Dino Karabeg, Stein Krogdahl, Petter Kristiansen
dino@ifi.uio.no steinkr@ifi.uio.no pettkr@ifi.uio.no

- **Lærebok:**

Algorithms: Sequential, Parallel, and Distributed,
Kenneth A. Berman and Jerome L. Paul.
Til salgs i ABC-fagbokhandel (studierabatt).



(Pluss kanskje litt vi plukker andre steder og legger ut.)

Velkommen

- **Gruppelærere:**

Karianne Ekern, Patricia Aas
karianho@student.matnat.uio.no psmaas@ifi.uio.no

- **Obliger:**

To stykker, som må godkjennes.
(Pluss en frivillig konkurranse.)

- **Andre, ”nærliggende” kurs:**

INF-MAT 3/4370 Lineær optimering
INF-MAT 5360 Matematisk optimering
INF 5340 Algoritmer i bioinformatikk

Kvalitetssikring ved Ifi

- Som student har du rett og plikt til å bidra til kvalitetssikringen av studiet ditt. Dette gjør du først og fremst gjennom å delta i undervisningsevaluering. Faglærer vil ta initiativ til å sette i gang undervisningsevalueringen for hvert enkelt emne.
- Undervisningsevalueringen gir deg mulighet til å komme med tilbakemeldinger og innspill på undervisningen i løpet av semesteret, slik at forbedringer kan gjøres underveis.
- Du finner mer informasjon om dette på hovedsiden til Institutt for informatikk, under "Annet" – "Kvalitetssikring", eller ved å følge denne linken:
<http://www.ifi.uio.no/studinf/kvalitetssikring/studenter>

Undervisningsplan

01/09	Søking i strenger (kap. 20)
08/09	Dynamisk programmering (kap. 9)
15/09	Trær (kap. 21) (Forelesning holdes i PO-bygget.)
22/09	
29/09	
06/10	
13/10	Undervisningsfri

Søking i strenger

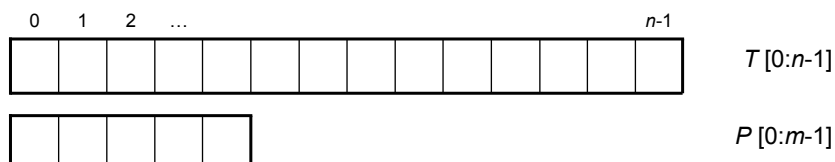
- **Generelle søkealgoritmer**
 - Naiv algoritme
 - Knuth-Morris-Pratt-algoritmen
 - Boyer-Moore-algoritmen
 - Karp-Rabin-algoritmen
- **Indeksering av tekst**
 - Trie-trær
 - Suffix-trær

Naiv algoritme

Et **alfabet** er en mengde symboler $A = \{a_1, a_2, \dots, a_k\}$

En **streng** $S = S[0:n-1]$ av lengde n er en sekvens av symboler fra A (gjerne med repetisjoner)

(Vi kan se på strengen S både som et array $S[0:n-1]$ og som en sekvens av symboler $S = s_1 s_2 \dots s_{n-1}$.)

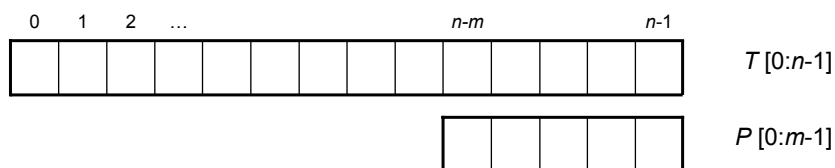


Naiv algoritme

Et **alfabet** er en mengde symboler $A = \{a_1, a_2, \dots, a_k\}$

En **streng** $S = S[0:n-1]$ av lengde n er en sekvens av symboler fra A (gjerne med repetisjoner)

(Vi kan se på strengen S både som et array $S[0:n-1]$ og som en sekvens av symboler $S = s_1 s_2 \dots s_{n-1}$.)



Naiv algoritme

```
function NaiveStringMatcher (P [0:m -1], T [0:n -1])
  for s ← 0 to n - m do
    if T [s:s + m -1] = P then
      return(s)
    endif
  endfor
  return(-1)
end NaiveStringMatcher
```

for-løkken eksekveres $n - m + 1$ ganger.
Hver sjekk $T [s:s + m -1] = P$ gjør inntil m symbolsammenlikninger. } $O(nm)$

Total kjøretid (worst case)

Knuth-Morris-Pratt-algorithmen

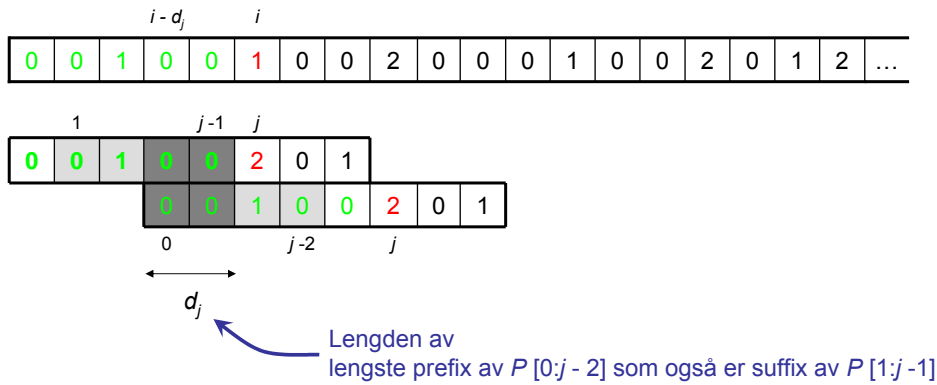
0	0	1	0	0	1	0	0	2	0	0	0	1	0	0	2	0	1	2	...
0	0	1	0	0	2	0	1												

Knuth-Morris-Pratt-algorithmen

0	0	1	0	0	1	0	0	2	0	0	0	1	0	0	2	0	1	2	...
			0	0	1	0	0	2	0	1									

← 3 →

Knuth-Morris-Pratt-algoritmen



Vi kan vet nå at vi kan flytte P maksimalt $j-d_j$ steg.
Og vi vet at $P[0:d_j-1]$ matcher T , så vi kan starte å sammenlikne med $P[d_j:m-1]$

Knuth-Morris-Pratt-algoritmen

```

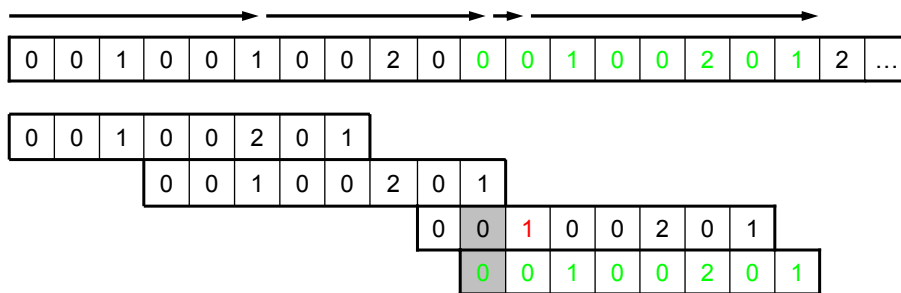
function KMPStringMatcher ( $P[0:m-1]$ ,  $T[0:n-1]$ )
   $i \leftarrow 0$ 
   $j \leftarrow 0$ 
  CreateNext( $P[0:m-1]$ , Next [ $n-1$ ])
  while  $i < n$  do
    if  $P[j] = T[i]$  then
      if  $j = m-1$  then
        return( $i - m + 1$ )
      endif
       $i \leftarrow i + 1$ 
       $j \leftarrow j + 1$ 
    else
       $j \leftarrow \text{Next}[j]$ 
      if  $j = 0$  then
        if  $T[i] \neq P[0]$  then
           $i \leftarrow i + 1$ 
        endif
      endif
    endif
  endif
  endwhile
  return(-1)
end KMPStringMatcher
  
```

$O(2n)$

```

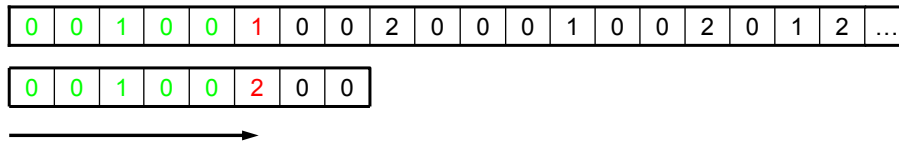
function CreateNext (P [0:m -1], Next [0:m -1])
  Next [ 0 ] ← Next [ 1 ] ← 0
  i ← 2
  j ← 0
  while i < m do
    if P [j] = P [i - 1] then
      Next [ i ] ← j + 1
      i ← i + 1
      j ← j + 1
    else
      if j > 0 then
        j ← Next [j - 1]
      else
        Next [ i ] ← 0
        i ← i + 1
      endif
    endif
  endwhile
end CreateNext
  
```

Knuth-Morris-Pratt-algoritmen

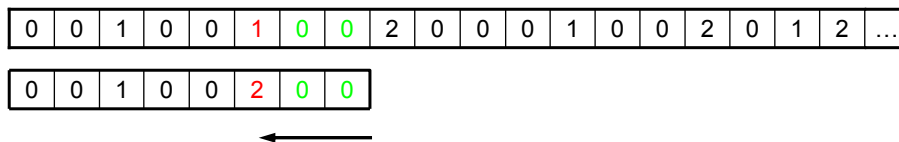


Lineær algoritme

Knuth-Morris-Pratt-algorithmen



Boyer-Moore-algorithmen



Karp-Rabin-algoritmen

- Vi antar at strengene våre kommer fra et k -ært alfabet $A = \{0, 1, 2, \dots, k-1\}$.
- Hvert symbol i A kan sees på som et siffer i k -tallssystemet.
- Hver streng S i A^* kan sees på som tall S' i k -tallssystemet.

Eks:

$k = 10$, og $A = \{0, 1, 2, \dots, 9\}$ (Det vanlige 10-tallssystemet)

Strengen "6832355" kan sees på som tallet 6832355.

- Gitt en streng $P [0:m-1]$, kan vi beregne det korresponderende tallet med m multiplikasjoner og m addisjoner (Horners regel):

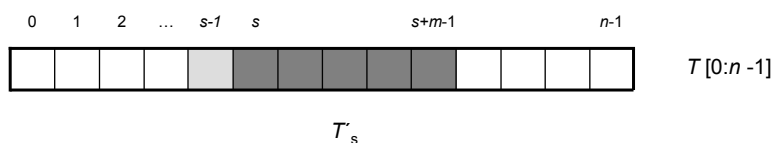
$$P' = P[m-1] + k(P[m-1] + k(P[m-2] + \dots + k(P[1] + kP[0])\dots))$$

Eks:

$$1234 = 4 + 10(3 + 10(2 + 10*1))$$

Karp-Rabin-algoritmen

- Gitt en tekststreng $T [0:n-1]$, og et heltall s (start-index), bruker vi T_s som betegnelse på delstrengen $T [s: s+m-1]$. (Vi antar at patternet vårt har lengde m .)
- En algoritme basert på Horners regel beregner T'_0, T'_1, T'_2, \dots og sammenlikner disse tallene med tallet P' for patternet P . (Tilsvarende den naive algoritmen.)
- Gitt T'_{s-1} og k^{m-1} , kan vi regne ut T'_s i konstant tid. **!**



Karp-Rabin-algoritmen

Grunnen til at vi kan beregne T'_s i konstant tid når vi har T'_{s-1} og k^{m-1} , er følgende rekurrensrelasjon:

$$T'_s = k(T'_{s-1} - k^{m-1} * T[s]) + T[s+m] \quad s = 1, \dots, n - m$$

Konstant, beregnes *en* gang, kan gjøres i tid $O(\log m)$

Eks:

$k = 10$, $A = \{0, 1, 2, \dots, 9\}$ (Det vanlige 10-tallssystemet) og $m = 7$.

$$T'_{s-1} = 7937245$$

$$T'_s = 9372458$$

$$T'_s = 10(7937245 - (1000000 * 7)) + 8$$

Kan gjøres i konstant tid. Bare multiplikasjon og addisjon, vi antar disse operasjonene kan gjøres i konstant tid.

Karp-Rabin-algoritmen

- Kan beregne T'_s i konstant tid når vi har T'_{s-1} og k^{m-1} .
- Altså kan vi beregne de $n - m + 1$ tallene T'_s , $s = 0, 1, \dots, n - m$ og P' i tid $O(n)$.
- Vi kan altså, i teorien, implementere en søkealgoritme med kjøretid $O(n)$.
- Dessverre vil tallene T'_s og P' i praksis være for store til at algoritmen blir praktisk anvendbar.
- Trikket er å bruke modulo-aritmetikk. Vi gjør alle beregninger modulo q (q et tilfeldig valgt primtall, slik at kq akkurat passer i et 32/64 bits register).

Karp-Rabin-algoritmen

- Vi beregner $T^{(q)}_s$ og $P^{(q)}$, hvor

$$\left. \begin{array}{l} T^{(q)}_s = T_s \bmod q, \\ P^{(q)} = P' \bmod q, \end{array} \right\} \begin{array}{l} \text{Resten i divisjonen, n\aa r vi deler p\aa } q: \\ \text{et tall i intervallet } \{0, 1, \dots, q-1\}. \end{array}$$

og sammenlikner.

- Vi kan ha $T^{(q)}_s = P^{(q)}$, selv om $T_s \neq P'$, en s\aa kalt *spuri\aa s match*.
- Har vi $T^{(q)}_s = P^{(q)}$, m\aa vi alts\aa gj\aa re en n\aa yaktig sjekk av T_s og P .
- Med stor nok q , er sannsynligheten for spuri\aa se matcher lav.

Karp-Rabin-algoritmen

```

function KarpRabinStringMatcher (P [0:m -1], T [0:n -1], k, q)
  c ← km-1 mod q
  P(q) ← 0
  T(q)s ← 0

  for i ← 1 to m do
    P(q) ← (k * P(q) + P [ i ]) mod q
    T(q)0 ← (k * T(q)0 + T [ i ]) mod q
  endfor

  for s ← 0 to n - m do
    if s > 0 then
      T(q)s ← (k * ( T(q)s-1 - T [ s ] * c ) + T [ s + m ]) mod q
    endif
    if T(q)s = P(q) then
      if Ts = P then
        return(s)
      endif
    endif
  endfor
  return(-1)
end KarpRabinStringMatcher
  
```

Karp-Rabin-algoritmen

- Worst case-kjøretid for Karp-Rabin-algoritmen får vi når patternet P finnes helt i slutten av strengen T .
- Sannsynligheten for at $T^{(q)}_s$ antar en spesifikk verdi i intervallet $\{0, 1, \dots, q-1\}$ er uniform $1/q$. (Vi antar strengene er uniformt fordelte.)
- $T^{(q)}_s, s = 0, 1, \dots, n-m-1$ vil altså gi opphav til en spuriøs match med sannsynlighet $1/q$.
- La r være det forventede antall spuriøse matcher. Hver av disse innebærer (worst case) m sammenlikninger. I tillegg må vi sjekke $T^{(q)}_{n-m}$, hvor vi får match.
- Kjøretiden blir altså: $(r + 1)m + (n - m + 1)$

Karp-Rabin-algoritmen

- r er en binomialfordelt stokastisk variabel. (Hver skift er et "forsøk", med suksessansynlighet $1/q$, og vi gjør $n-m$ forsøk) [Nå anser vi plutselig spuriøse matcher som suksess...]
- $E[r] = (n-m)/q$. (Forventning av binomialfordelt variabel.)

- Totalt får vi altså $\left(\frac{n-m}{q} + 1\right)m + (n-m+1)$

Forventet kjøretid når matchen finnes helt til slutt i T .

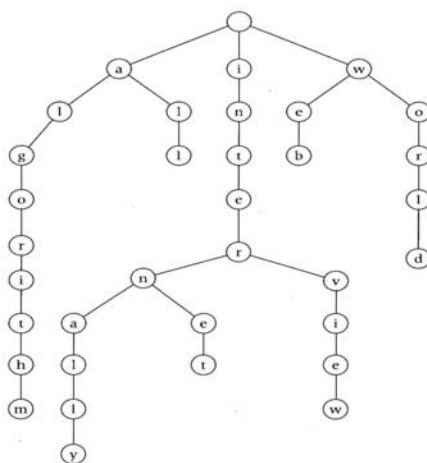
- Hvis $q < C$, hvor C er en konstant, blir kjøretiden $\Theta(nm)$.
- MEN det er rimelig å anta $q \gg m$, da blir kjøretiden $O(n)$

Søking i strenger

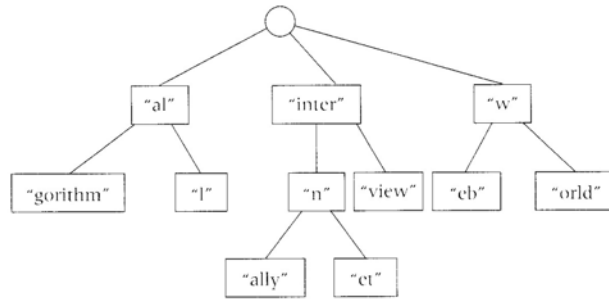
- **Generelle søkealgoritmer**

- Naiv algoritme $O(nm)$
- Knuth-Morris-Pratt-algoritmen $O(n)$
- Boyer-Moore-algoritmen $\Theta(nm)$
- Karp-Rabin-algoritmen $O(n)$

Trie-trær



Trie-trær



Suffix-trær

Suffix -tre for
"babbage"

