Tekstalgoritmer

Søk etter delstrenger i array

Definisjoner

Et **alfabet** er en endelig mengde tegn $A = \{a_1, a_2, \ldots, a_k\}$.

En (tekst)**streng** $S = S\,[0: n-1]$ med lengde n er en sekvens av tegn fra A.

Vi vil i programmene representer strengen S som en array $S[0:n-1]$.

Definisjoner

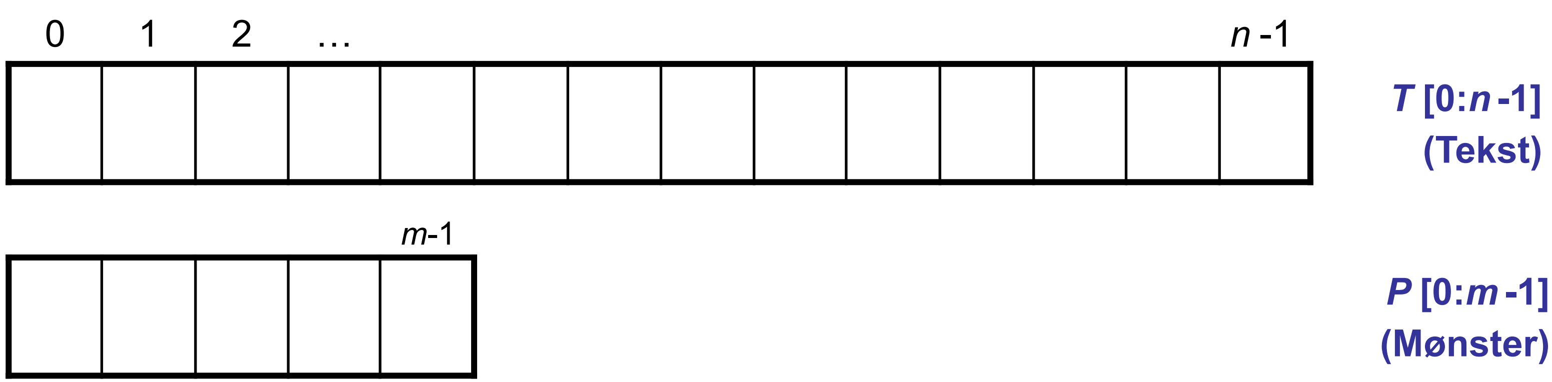
Et **alfabet** er en endelig mengde tegn $A = \{a_1, a_2, \ldots, a_k\}$.

En (tekst)**streng** $S = S\,[0{:}\ n\text{-}1]$ med lengde n er en sekvens av tegn fra A.

Vi vil i programmene representer strengen S som en array $S[0{:}n\text{-}1]$.

Søkeproblemet: Gitt to strenger, T (= Tekst) and P (= Pattern) mønster, hvor P er kortere enn T (vanligvis mye kortere).

Finn ut om P finnes som en (sammenhengende) substreng i T, og hvis ja, hvor i T.



Definisjoner

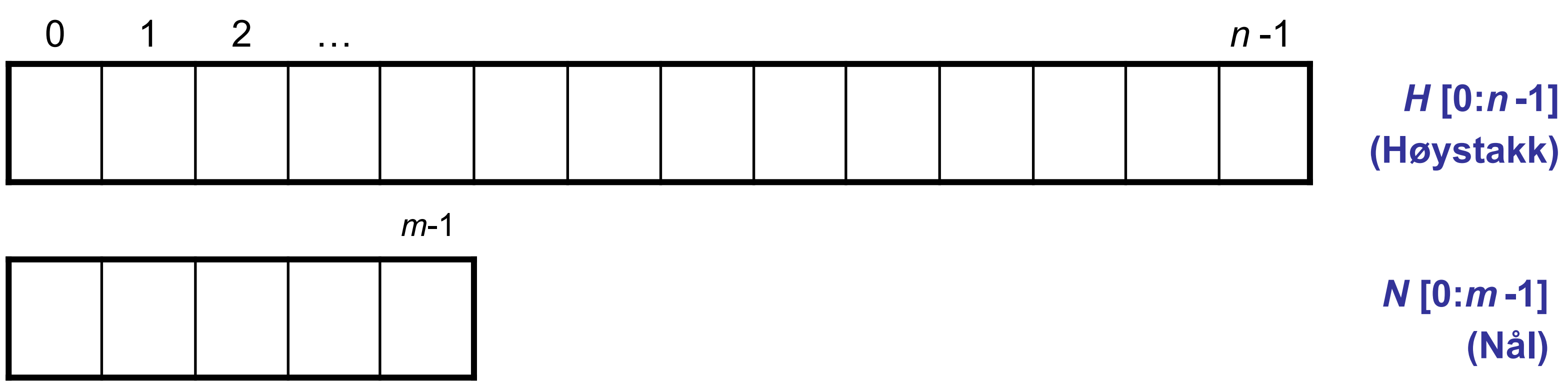
Et **alfabet** er en endelig mengde tegn $A = \{a_1, a_2, \ldots, a_k\}$.

En (tekst)**streng** $S = S\,[0{:}\ n\text{-}1]$ med lengde n er en sekvens av tegn fra A.

Vi vil i programmene representer strengen S som en array $S[0{:}n\text{-}1]$.

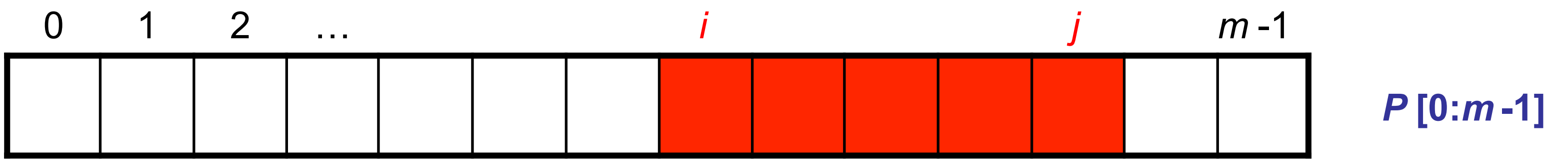
Søkeproblemet: Gitt to strenger, T (= Tekst) and P (= Pattern) mønster, hvor P er kortere enn T (vanligvis mye kortere).

Finn ut om P finnes som en (sammenhengende) substreng i T, og hvis ja, hvor i T.



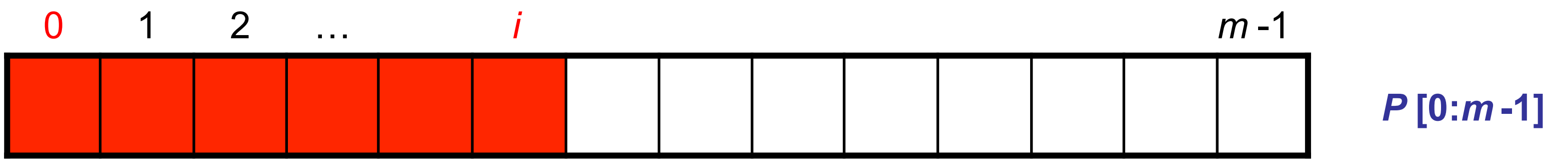
Definisjoner

- Let P be a string of size m
  - A substring $P[i \,..\, j]$ of P is the subsequence of P consisting of the characters with ranks between i and j
  - A prefix of P is a substring of the type $P[0 \,..\, i]$
  - A suffix of P is a substring of the type $P[i \,..m - 1]$



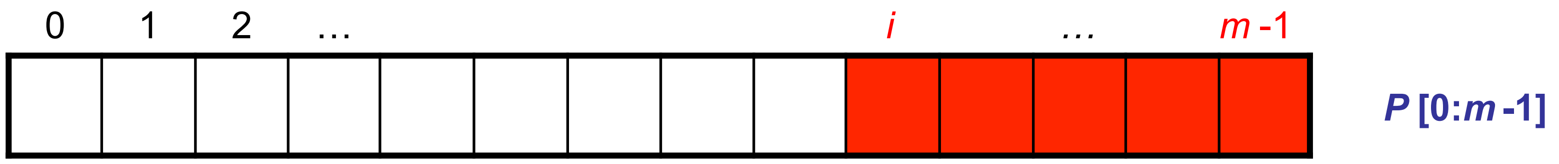
Definisjoner

- Let P be a string of size m
  - A substring $P[i \, .. \, j]$ of P is the subsequence of P consisting of the characters with ranks between i and j
  - A prefix of P is a substring of the type $P[0 \, .. \, i]$
  - A suffix of P is a substring of the type $P[i \, .. m - 1]$



Definisjoner

- Let P be a string of size m
  - A substring $P[i .. j]$ of P is the subsequence of P consisting of the characters with ranks between i and j
  - A prefix of P is a substring of the type $P[0 .. i]$
  - A suffix of P is a substring of the type $P[i .. m - 1]$



Class String

java.lang.Object
    java.lang.String

All Implemented Interfaces:

Serializable, CharSequence, Comparable<String>

```
public final class String
extends Object
implements Serializable, Comparable<String>, CharSequence
```

The `String` class represents character strings. All string literals in Java programs, such as `"abc"`, are implemented as instances of this class.

Strings are constant; their values cannot be changed after they are created. String buffers support mutable strings. Because String objects are immutable they can be shared. For example:

```
String str = "abc";
```

is equivalent to:

```
char data[] = {'a', 'b', 'c'};
String str = new String(data);
```

Examples of strings:

- Python program
- HTML document
- DNA sequence
- Digitized image

```
import math
print("Enter the coefficients of the form ax^3 + bx^2 + cx + d")
lst=[]
for i in range(0,4):
    a=int(input("Enter coefficient:"))
    lst.append(a)
x=int(input("Enter the value of x:"))
sum1=0
j=3
for i in range(0,3):
    while(j>0):
        sum1=sum1+(lst[i]*math.pow(x,j))
        break
    j=j-1
sum1=sum1+lst[3]
print("The value of the polynomial is:",sum1)
```

Examples of strings:

Python program
HTML document
DNA sequence
Digitized image

```
<!DOCTYPE html>
<html lang="no">
<head>
  <meta http-equiv="X-UA-Compatible" content="IE=edge" />
  <meta id="viewport" name="viewport" content="width=device-width, initial-scale=1" />
 <meta charset="utf-8" >
 <meta name="format-detection" content="telephone=no">
 <meta name="generator" content="Vortex" />
  <title>Obligatoriske innleveringer høsten 2018 - IN2010 - Høst 2018 - Universitetet i Oslo</title>
  <meta property="og:title" content="Obligatoriske innleveringer høsten 2018 - IN2010 - Høst 2018 - Universitetet i Oslo" />
  <meta name="twitter:card" content="summary" />
  <meta name="twitter:site" content="@unioslo" />
  <meta name="twitter:title" content="Obligatoriske innleveringer høsten 2018" />
   <meta name="twitter:description" content="Les denne saken på UiOs nettsider." />
   <meta name="twitter:url" content="https://www.uio.no/studier/emner/matnat/ifi/IN2010/h18/obligatoriske-innleveringer/index.html" />
  <meta property="og:url" content="https://www.uio.no/studier/emner/matnat/ifi/IN2010/h18/obligatoriske-innleveringer/index.html" />
  <meta property="og:type" content="website" />
   <link rel="shortcut icon" href="/vrtx/decorating/resources/dist/images/favicon.ico" >
   <link rel="apple-touch-icon-precomposed" href="/vrtx/decorating/resources/dist/images/apple-touch-icon.png" >
 <script><!--
  if (/iPad|Android 3/i.test(navigator.userAgent)) {
   var tabletVp = document.getElementById('viewport');
   tabletVp.setAttribute("content", "width=1020, user-scalable=yes");
  }
  if (/googlebot/i.test(navigator.userAgent)) {
   document.addEventListener("DOMContentLoaded", function(event) {
    var css = '@media only screen and (max-width: 16cm) and (orientation : portrait),' +
         '      only screen and (max-width: 19cm) and (orientation : landscape) { * { max-width: 420px; } }';
    var head = document.getElementsByTagName('head')[0];
    var s = document.createElement('style');
    s.setAttribute('type', 'text/css');
    if (s.styleSheet) {
     s.styleSheet.cssText = css;
    } else {
     s.appendChild(document.createTextNode(css));
    }
    head.appendChild(s);
   });
  }
```

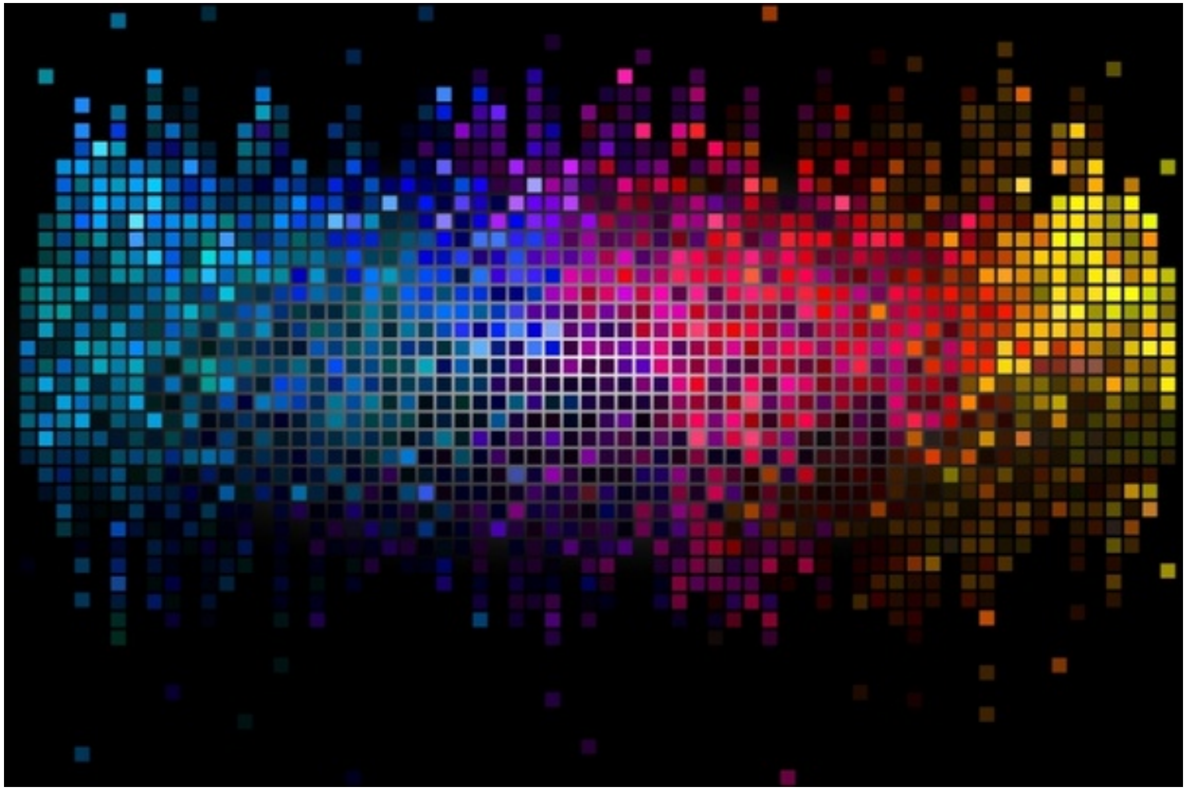
Examples of strings:

- Python program
- HTML document
- DNA sequence
- Digitized image

```
atgcgccgta ggacacttga ttacgaacaa tttctacaaa acacttgata ctgtatgagg
atacagtata attgcttcaa cagaacatat tgactatccg gcatgacagg agtaaaaatg
atggctatcg acgaaaacaa acagaaagcg ttggcggcag cactgggcca gattgagaaa
caatttggta aaggctccat catgcgcctg ggtgaagacc gttccatgga tgtggaaacc
atctctaccg gttcgctttc actggatatc gcgcttgggg caggtggtct gccgatgggc
cgtatcgtcg aaatctacgg accggaatct tccggtaaaa ccacgctgac gctgcaggtg
atcgccgcag cgcagcgtga aggtaaaacc tgtgcgttta tcgatgctga acacgcgctg
gacccaatct acgcacgtaa actgggcgtc gatatcgaca acctgctgtg ctcccagccg
gacaccggcg agcaggcact ggaaatctgt gacgccctgg cgcgttctgg cgcagtagac
gttatcgtcg ttgactccgt ggcggcactg acgccgaaag cggaaatcga aggcgaaatc
ggcgactctc acatgggcct tgcggcacgt atgatgagcc aggcgatgcg taagctggcg
ggtaacctga agcagtccaa cacgctgctg atcttcatca accagatccg tatgaaaatt
ggtgtgatgt tcggtaaccc ggaaaccact accggtggta acgcgctgaa attctacgcc
tctgttcgtc tcgacatccg tcgtatcggc gcggtgaaag agggcgaaaa cgtggtgggt
agcgaaaccc gcgtgaaagt ggtgaagaac aaaatcgctg cgccgtttaa acaggctgaa
ttccagatcc tctacggcga aggtatcaac ttctacggcg aactggttga cctgggcgta
aaagagaagc tgatcgagaa agcaggcgcg tggtacagct acaaaggtga gaagatcggt
cagggtaaag cgaatgcgac tgcctggctg aaagataacc cggaaaccgc gaaagagatc
gagaagaaag tacgtgagtt gctgctgagc aacccgaact caacgccgga tttctctgta
gatgatagcg aaggcgtagc agaaactaac gaagattttt aatcgtcttg tttgatacac
aagggtcgca tgtgcggccc ttttgttttt ttaagttgta aggatatgcc atgacacgat
```

Examples of strings:

- Python program
- HTML document
- DNA sequence
- Digitized image



Anvendelser

GREP(1) User Commands GREP(1)

NAME top

grep, egrep, fgrep - print lines that match patterns

SYNOPSIS top

grep [OPTION...] PATTERNS [FILE...]

grep [OPTION...] -e PATTERNS ... [FILE...]

grep [OPTION...] -f PATTERN_FILE ... [FILE...]

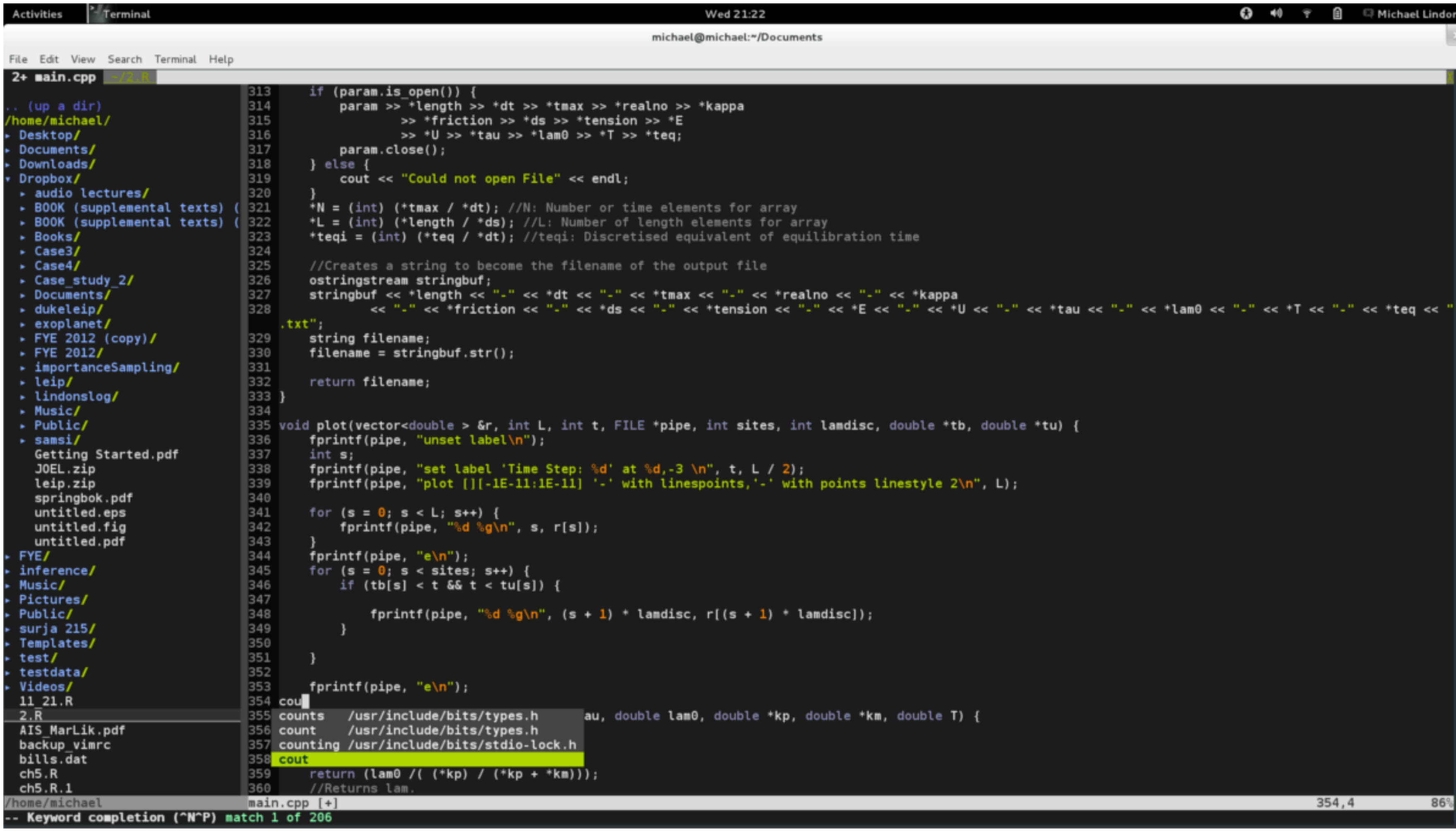
DESCRIPTION top

grep searches for PATTERNS in each FILE. PATTERNS is one or patterns separated by newline characters, and grep prints each line that matches a pattern.

A FILE of “-” stands for standard input. If no FILE is given, recursive searches examine the working directory, and nonrecursive searches read standard input.

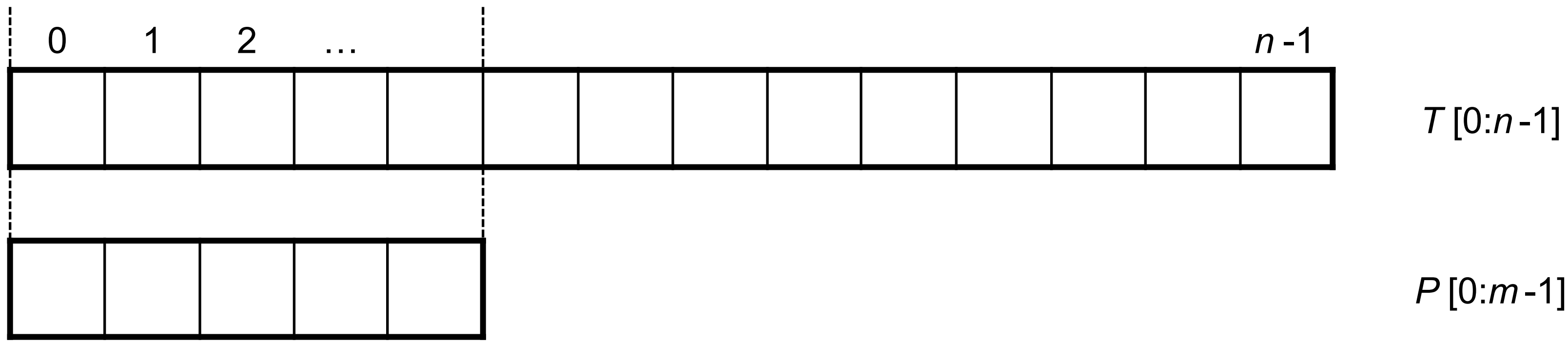
In addition, the variant programs egrep and fgrep are the same as grep -E and grep -F, respectively. These variants are deprecated, but are provided for backward compatibility.

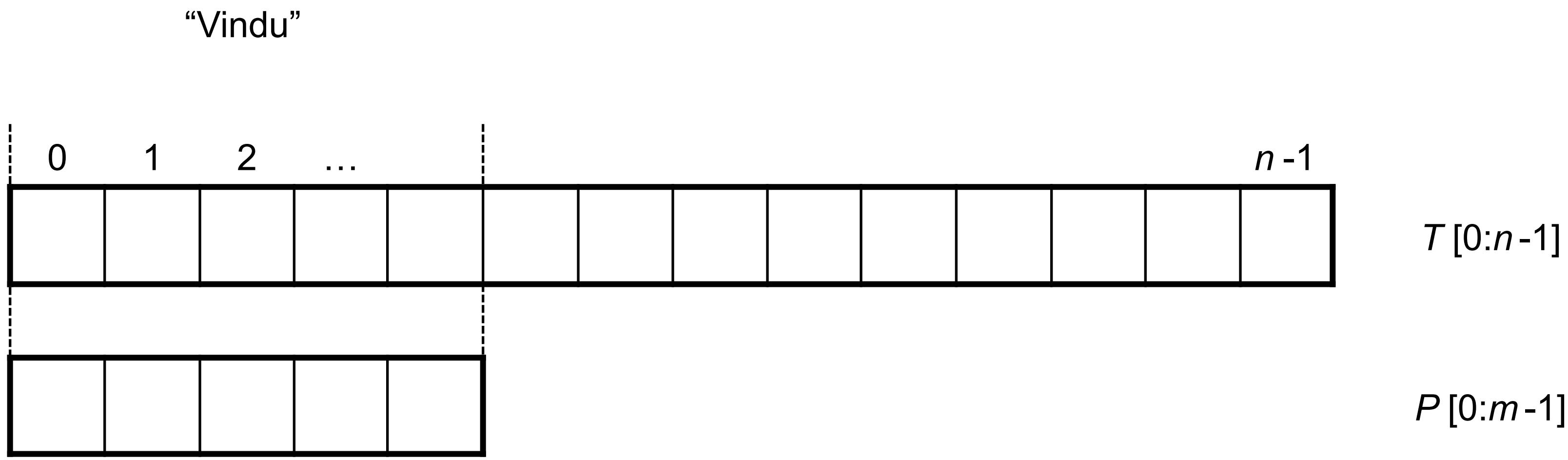
Anvendelser



Anvendelser

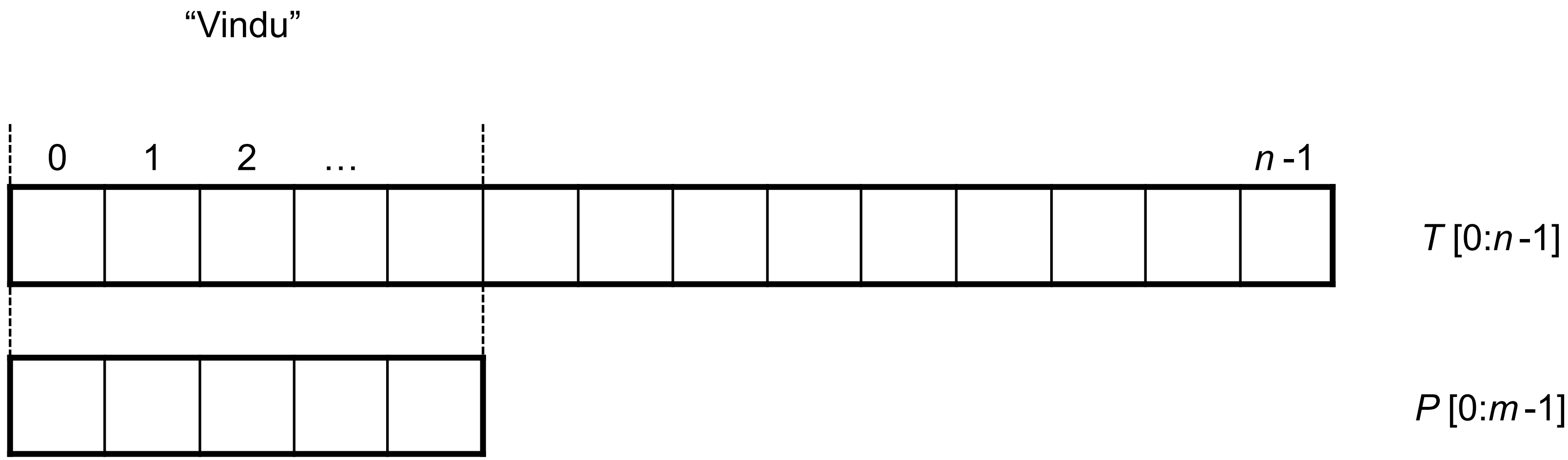
Anvendelser


0
1
2
…
n -1
T [0:n -1]
P [0:m -1]



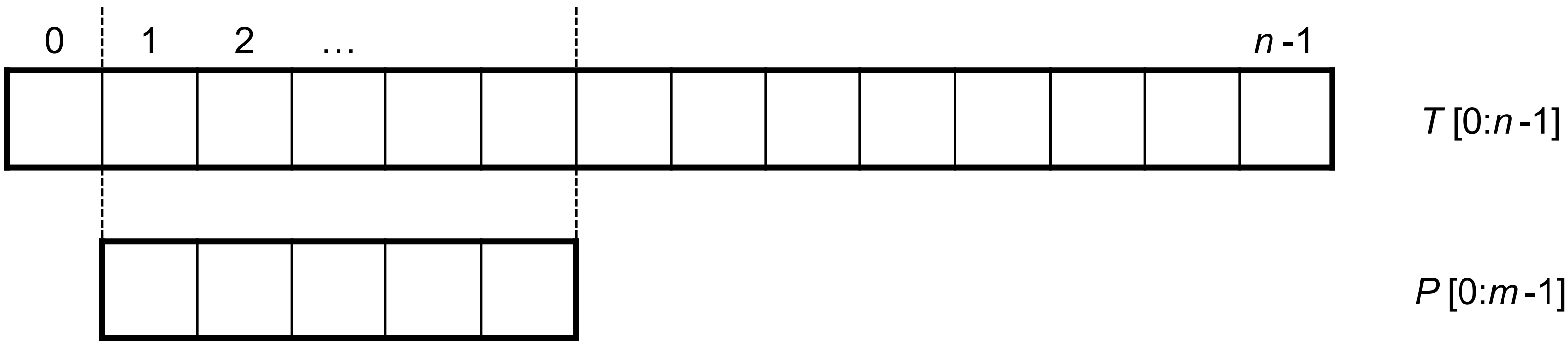
Hvordan vil *du* gjøre dette?

Beskriv en algoritme!



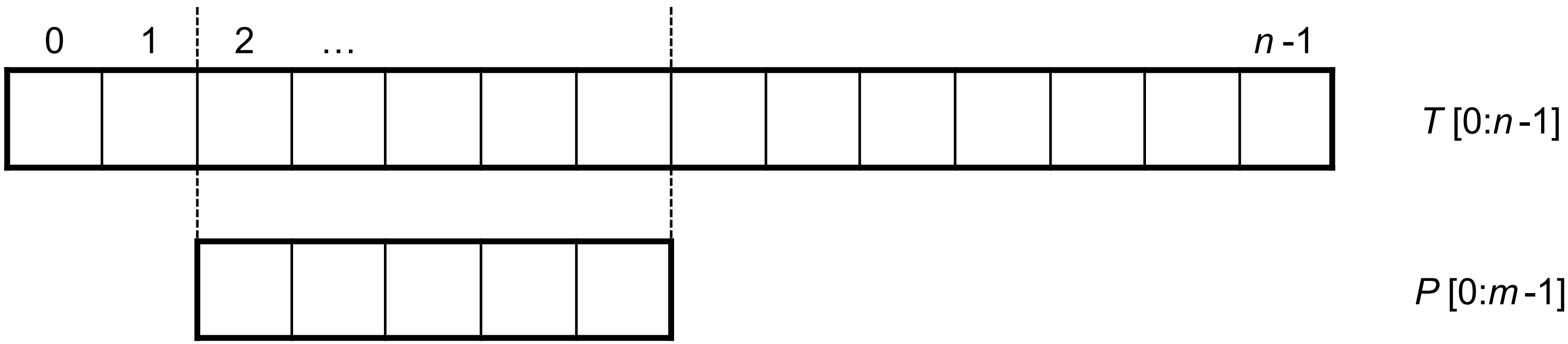
Sammenlign Vinduet med P:

Hvis mismatch, flytt vinduet ett hakk forover



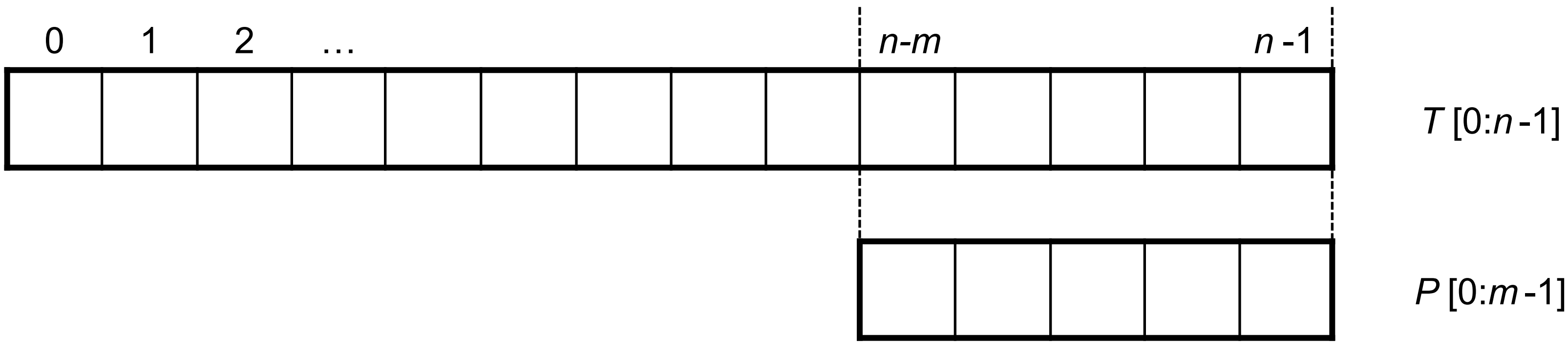
Sammenlign Vinduet med P:

Hvis mismatch, flytt vinduet ett hakk forover



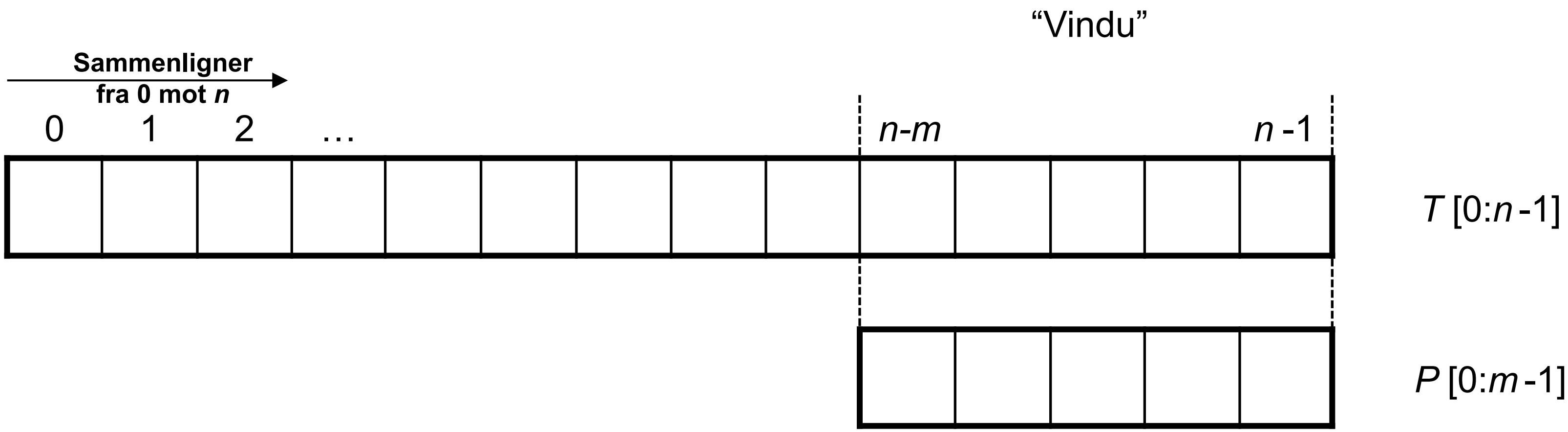
Sammenlign Vinduet med P:

Hvis mismatch, flytt vinduet ett hakk forover



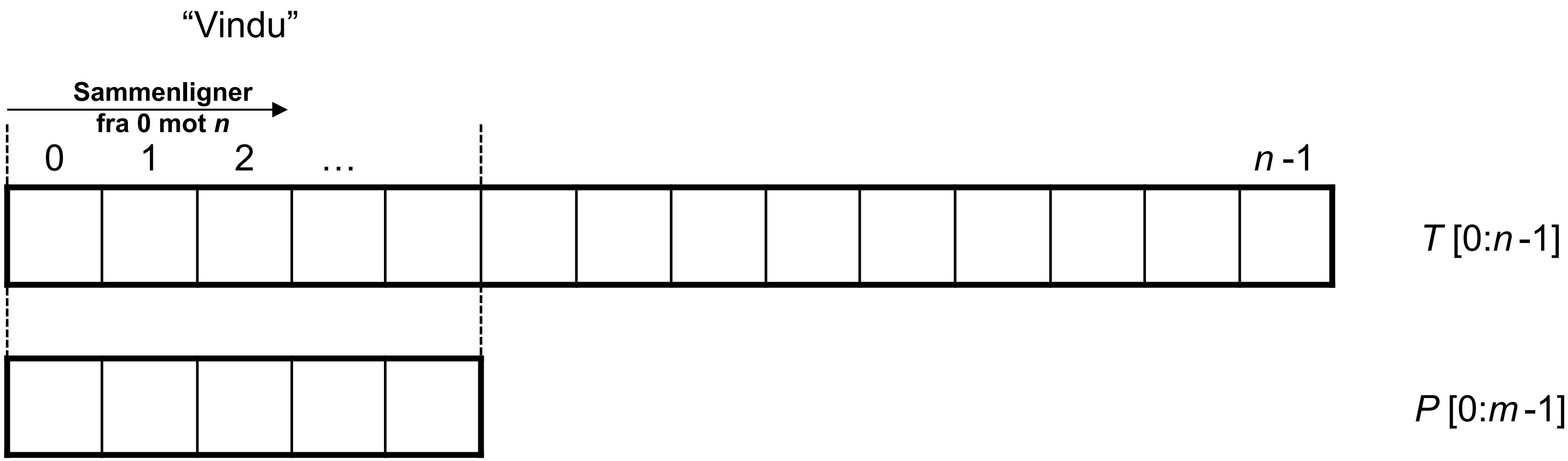
Sammenlign Vinduet med P:

Hvis mismatch, P finnes ikke i T (som substreng)

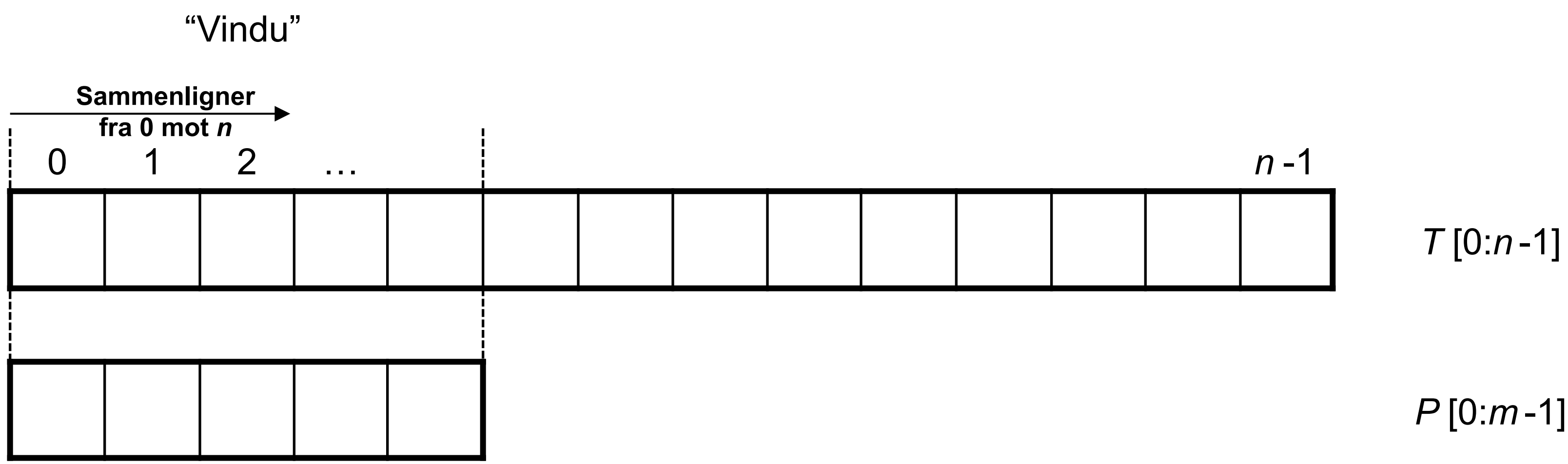



```
for i ← 0 to n - m do
    if T [i : i+m-1] = P then          // er vindu = P ?
        return(i)
    endif
endfor
return(-1)
```

Den naive algoritmen for strengsammenligning



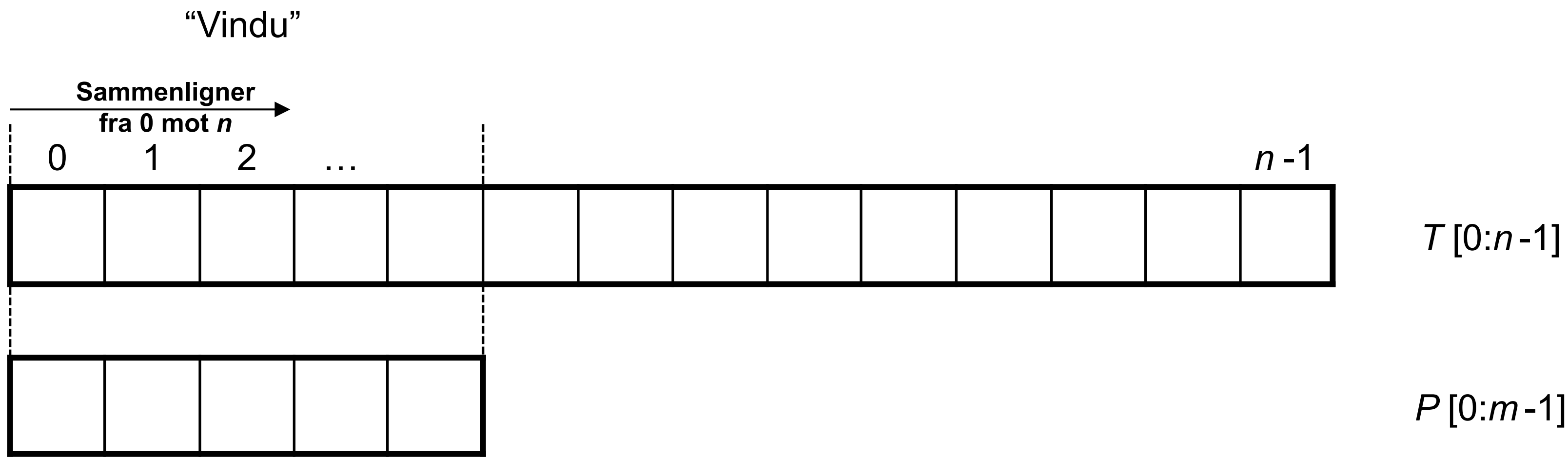
Den naive algoritmen for strengsammenligning



Brute force (rå kraft) brukes ofte synonymt med

- unødvendig tungvindt
- dårlig, men korrekt
- treg
- enkel
- lite gjennomtenkt
- nødløsning
- lite effektiv

Den naive algoritmen for strengsammenligning

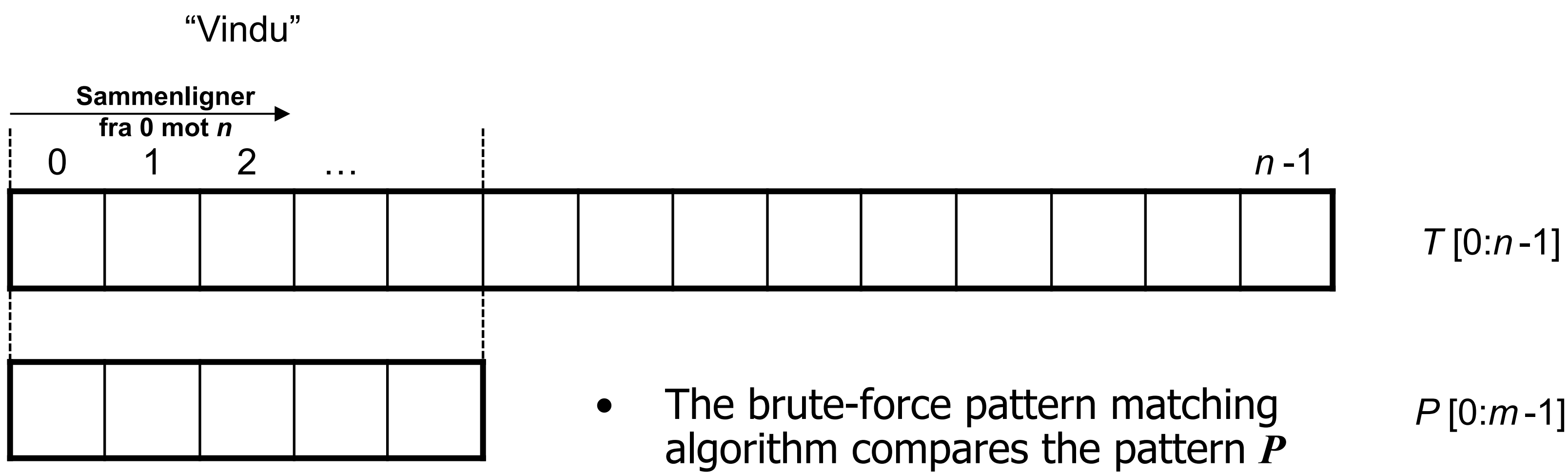


Brute force (rå kraft) brukes ofte synonymt med

- unødvendig tungvindt
- dårlig, men korrekt
- treg
- enkel
- lite gjennomtenkt
- nødløsning
- lite effektiv

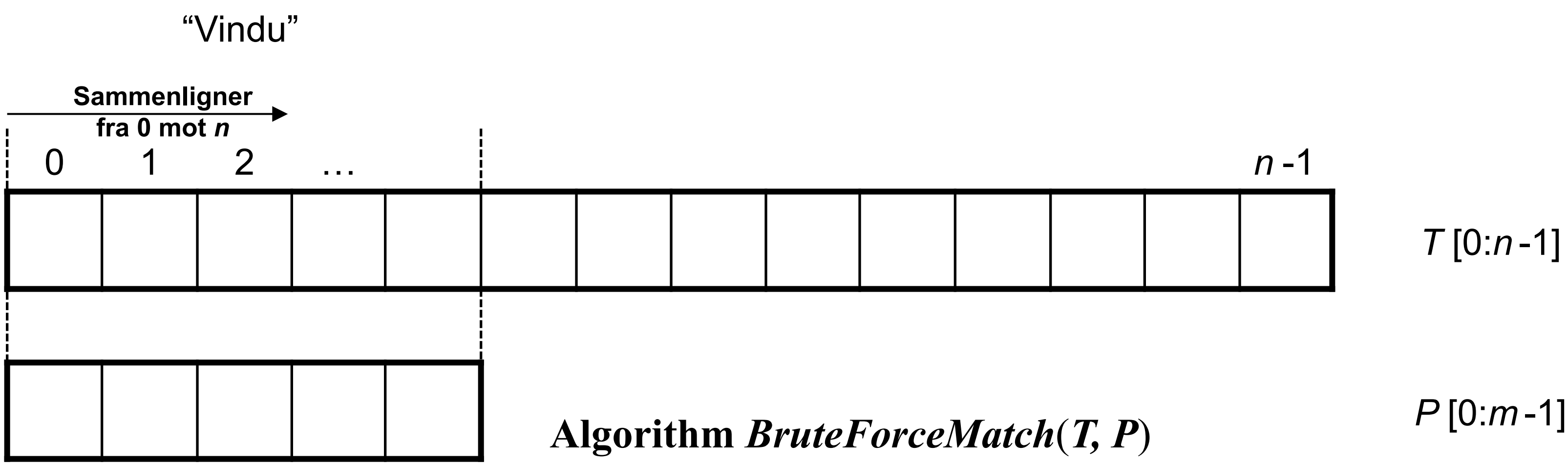
men er noen ganger nødvendig
brute force løsninger er typisk den første ideen vi får
i IN2010 søker vi som oftest mer *effektive* algoritmer

Den naive algoritmen for strengsammenligning



- The brute-force pattern matching algorithm compares the pattern P with the text T for each possible shift of P relative to T, until either
  - a match is found, or
  - all placements of the pattern have been tried
- Brute-force pattern matching runs in time $O(nm)$
- Example of worst case:
  - $T = aaa \ldots ah$
  - $P = aaah$
  - may occur in images and DNA sequences
  - unlikely in English text

Den naive algoritmen for strengsammenligning




```
Algorithm BruteForceMatch(T, P)
    Input text T of size n and pattern
        P of size m
    Output starting index of a
        substring of T equal to P or −1
        if no such substring exists
    for  i ← 0 to n − m
        { test shift i of the pattern }
        j ← 0
        while j < m ∧ T[i + j] = P[j]
            j ← j + 1
        if  j = m
            return  i {match at i}
        else
            break while loop {mismatch}
    return  -1 {no match anywhere}
```

Boyer-Moore

a		p	a	t	t	e	r	n		m	a	t	c	h	i	n	g		a	l	g	o	r	i	t	h	m

r	i	t	h	m

Boyer-Moore

a		p	a	t	t	e	r	n		m	a	t	c	h	i	n	g		a	l	g	o	r	i	t	h	m

r	i	t	h	m

Boyer-Moore

a		p	a	t	t	e	r	n		m	a	t	c	h	i	n	g		a	l	g	o	r	i	t	h	m

r	i	t	h	m

Boyer-Moore

a		p	a	t	t	e	r	n		m	a	t	c	h	i	n	g		a	l	g	o	r	i	t	h	m

r	i	t	h	m

Boyer-Moore

a		p	a	t	t	e	r	n		m	a	t	c	h	i	n	g		a	l	g	o	r	i	t	h	m

r	i	t	h	m

Boyer-Moore

a		p	a	t	t	e	r	n		m	a	t	c	h	i	n	g		a	l	g	o	r	i	t	h	m

r	i	t	h	m

Boyer-Moore

a		p	a	t	t	e	r	n		m	a	t	c	h	i	n	g		a	l	g	o	r	i	t	h	m

r	i	t	h	m

Boyer-Moore

a		p	a	t	t	e	r	n		m	a	t	c	h	i	n	g		a	l	g	o	r	i	t	h	m

r	i	t	h	m

Boyer-Moore

a		p	a	t	t	e	r	n		m	a	t	c	h	i	n	g		a	l	g	o	r	i	t	h	m

r	i	t	h	m

Boyer-Moore

a		p	a	t	t	e	r	n		m	a	t	c	h	i	n	g		a	l	g	o	r	i	t	h	m
					r	i	t	h	m																		

Boyer-Moore

a		p	a	t	t	e	r	n		m	a	t	c	h	i	n	g		a	l	g	o	r	i	t	h	m
						r	i	t	h	m																	

Boyer-Moore

a		p	a	t	t	e	r	n		m	a	t	c	h	i	n	g		a	l	g	o	r	i	t	h	m

r	i	t	h	m

Boyer-Moore

a		p	a	t	t	e	r	n		m	a	t	c	h	i	n	g		a	l	g	o	r	i	t	h	m

r	i	t	h	m

Boyer-Moore

a		p	a	t	t	e	r	n		m	a	t	c	h	i	n	g		a	l	g	o	r	i	t	h	m

r	i	t	h	m

Boyer-Moore

a		p	a	t	t	e	r	n		m	a	t	c	h	i	n	g		a	l	g	o	r	i	t	h	m

r	i	t	h	m

Boyer-Moore

a		p	a	t	t	e	r	n		m	a	t	c	h	i	n	g		a	l	g	o	r	i	t	h	m

r	i	t	h	m

Boyer-Moore

a		p	a	t	t	e	r	n		m	a	t	c	h	i	n	g		a	l	g	o	r	i	t	h	m
																	r	i	t	h	m						

Boyer-Moore

a		p	a	t	t	e	r	n		m	a	t	c	h	i	n	g		a	l	g	o	r	i	t	h	m
																						r	i	t	h	m	

Boyer-Moore

a		p	a	t	t	e	r	n		m	a	t	c	h	i	n	g		a	l	g	o	r	i	t	h	m
																						r	i	t	h	m	

Boyer-Moore

a		p	a	t	t	e	r	n		m	a	t	c	h	i	n	g		a	l	g	o	r	i	t	h	m

r	i	t	h	m

Boyer-Moore

a		p	a	t	t	e	r	n		m	a	t	c	h	i	n	g		a	l	g	o	r	i	t	h	m

r	i	t	h	m

Boyer-Moore

a		p	a	t	t	e	r	n		m	a	t	c	h	i	n	g		a	l	g	o	r	i	t	h	m
																							r	i	t	h	m

Boyer-Moore

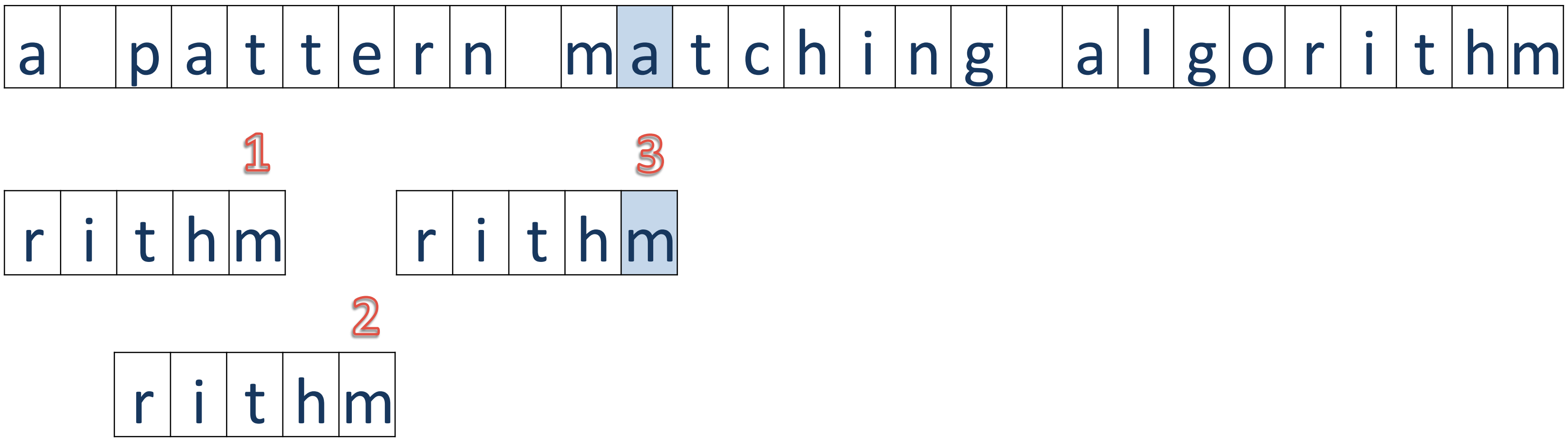
a		p	a	t	t	e	r	n		m	a	t	c	h	i	n	g		a	l	g	o	r	i	t	h	m

				1
r	i	t	h	m

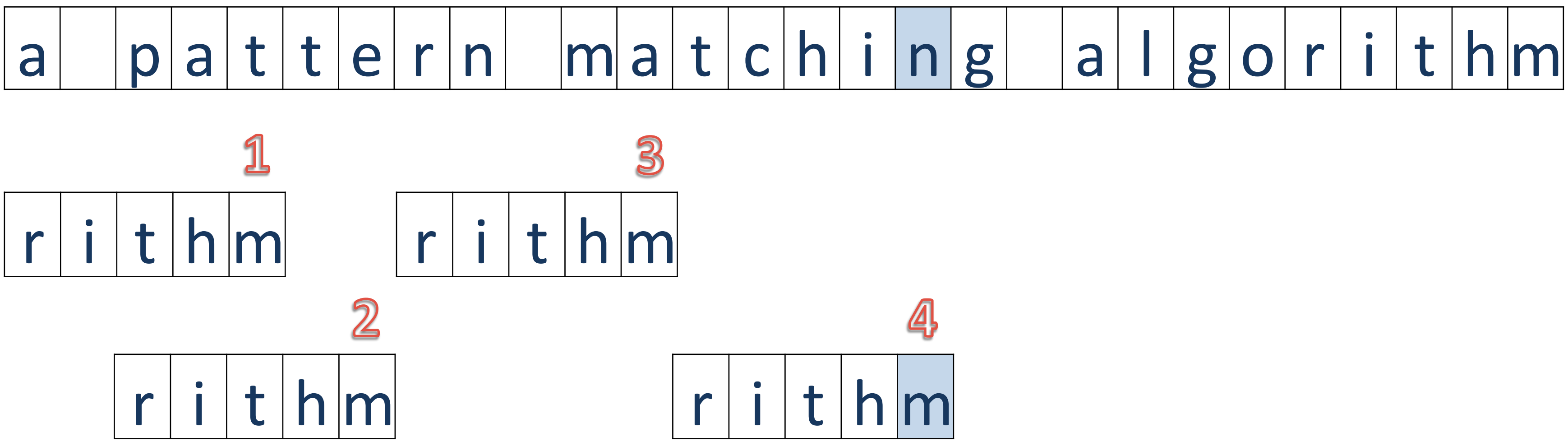
Boyer-Moore

a		p	a	t	t	e	r	n		m	a	t	c	h	i	n	g		a	l	g	o	r	i	t	h	m
				1																							
r	i	t	h	m																							
						2																					
		r	i	t	h	m																					

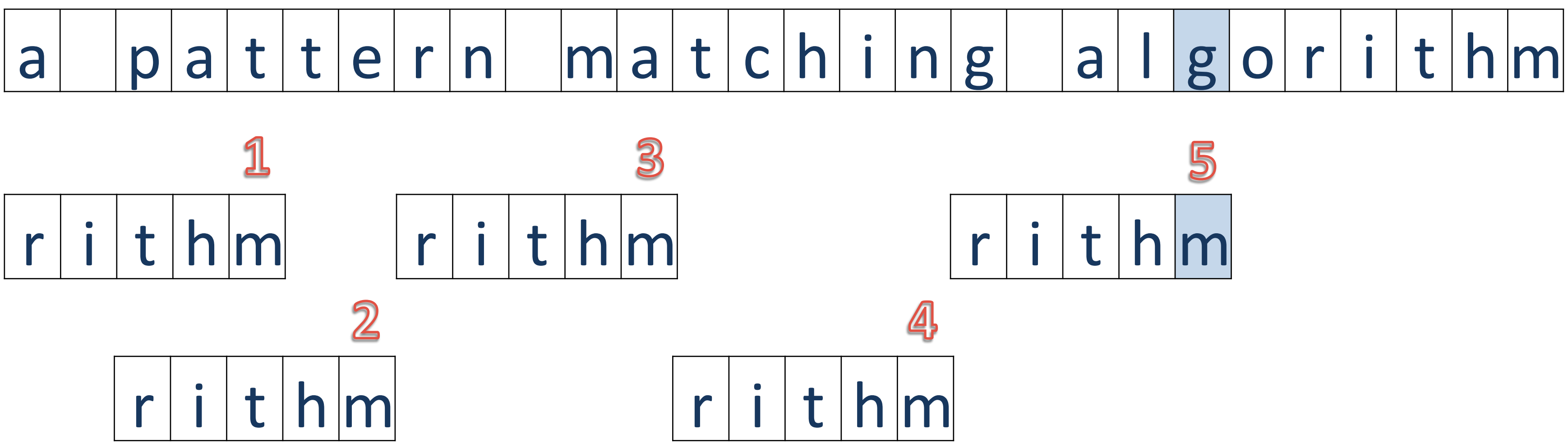
Boyer-Moore



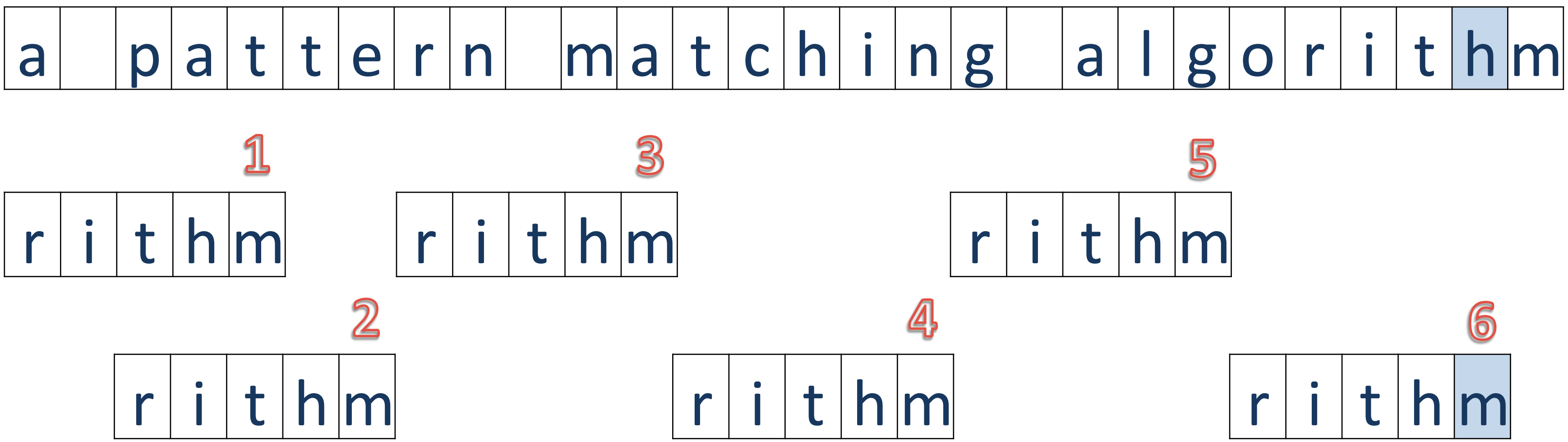
Boyer-Moore



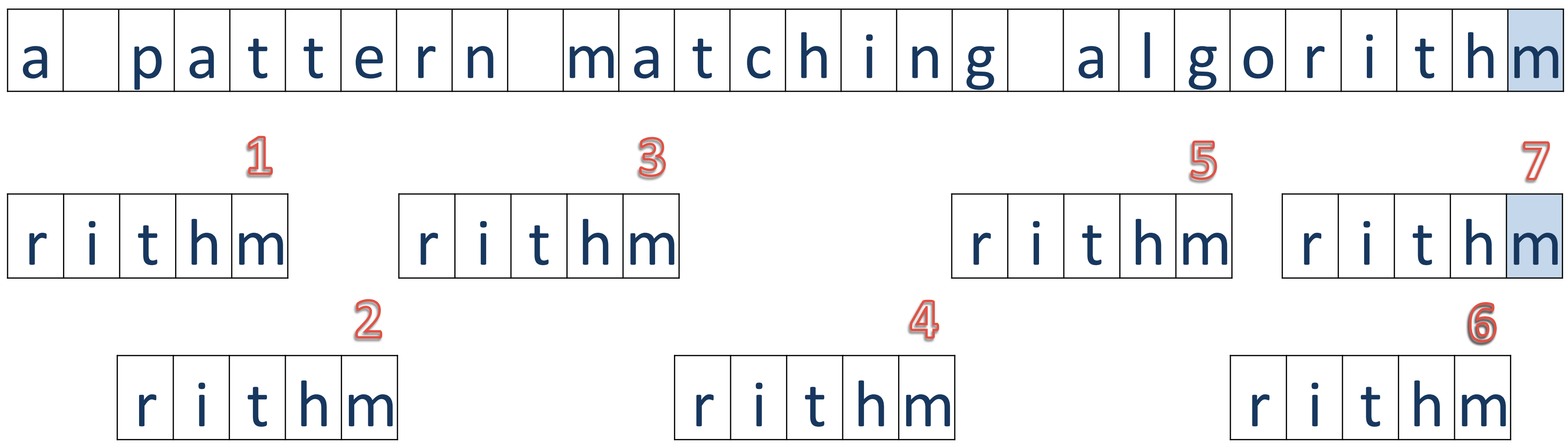
Boyer-Moore


a p a t t e r n m a t c h i n g a l g o r i t h m
1
r i t h m
2
r i t h m
3
r i t h m
4
r i t h m
5
r i t h m

Boyer-Moore

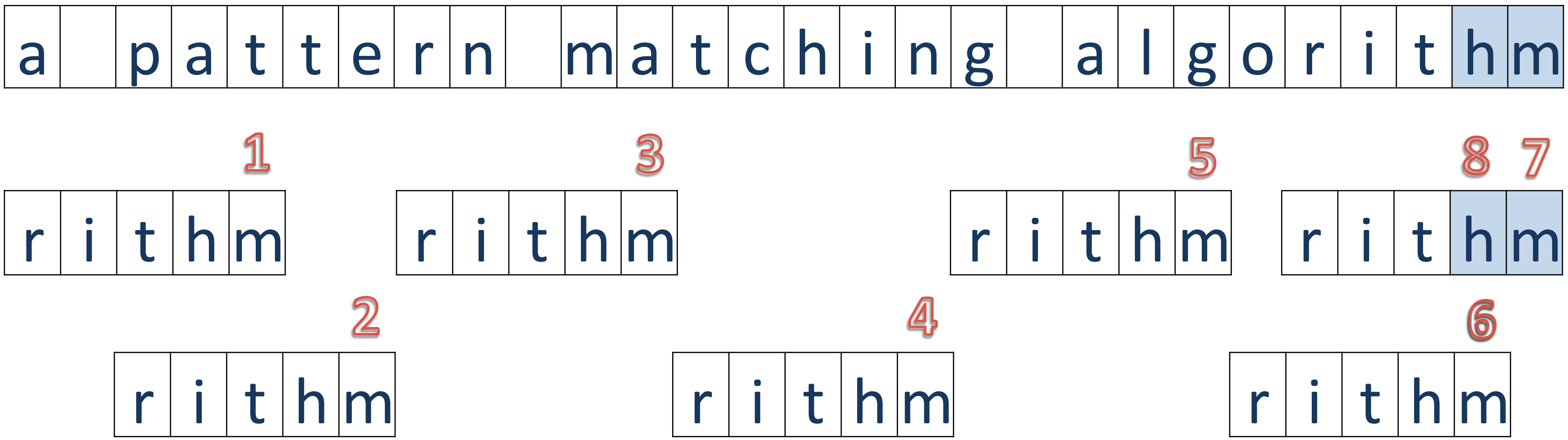

a pattern matching algorithm
1
3
5
rithm
rithm
rithm
2
4
6
rithm
rithm
rithm

Boyer-Moore

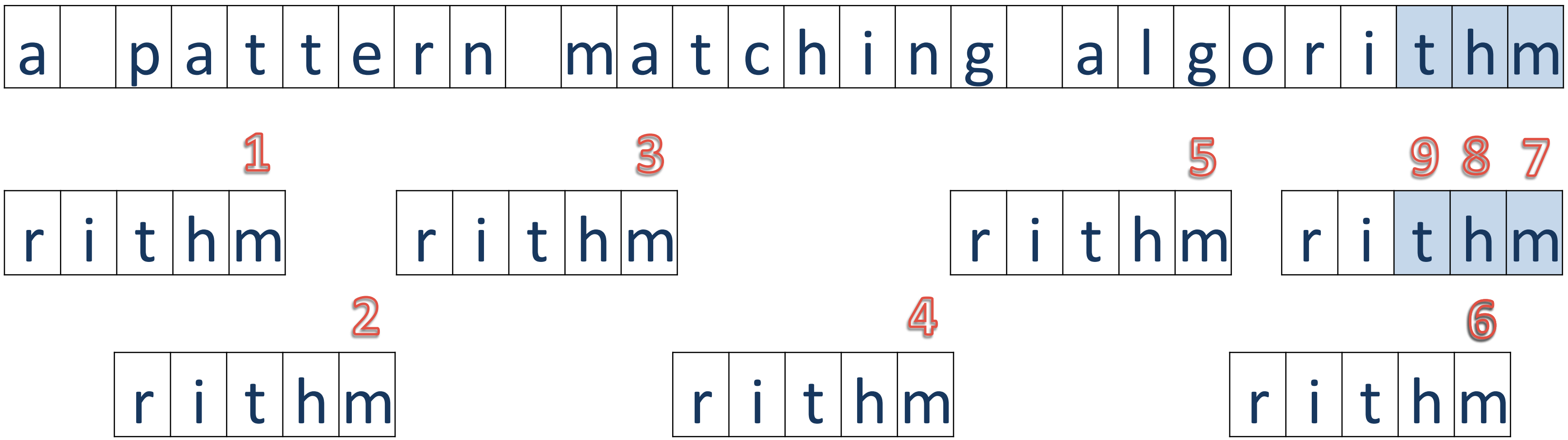


a pattern matching algorithm
1
r i t h m
2
r i t h m
3
r i t h m
4
r i t h m
5
r i t h m
6
r i t h m
7
r i t h m

Boyer-Moore

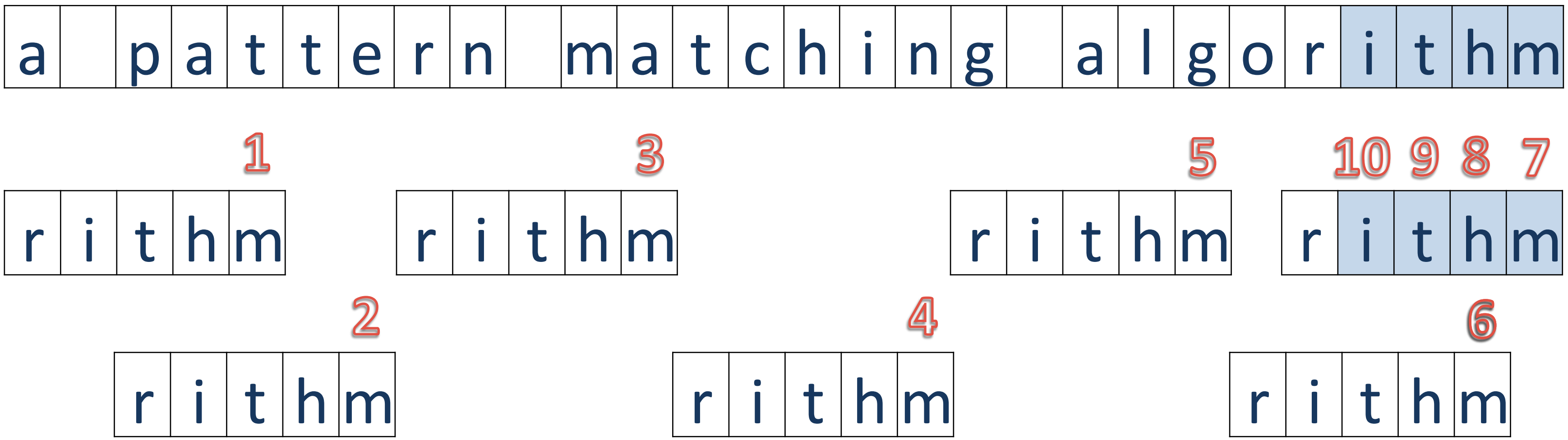

a pattern matching algorithm
1
r i t h m
3
r i t h m
5
r i t h m
8 7
r i t h m
2
r i t h m
4
r i t h m
6
r i t h m

Boyer-Moore

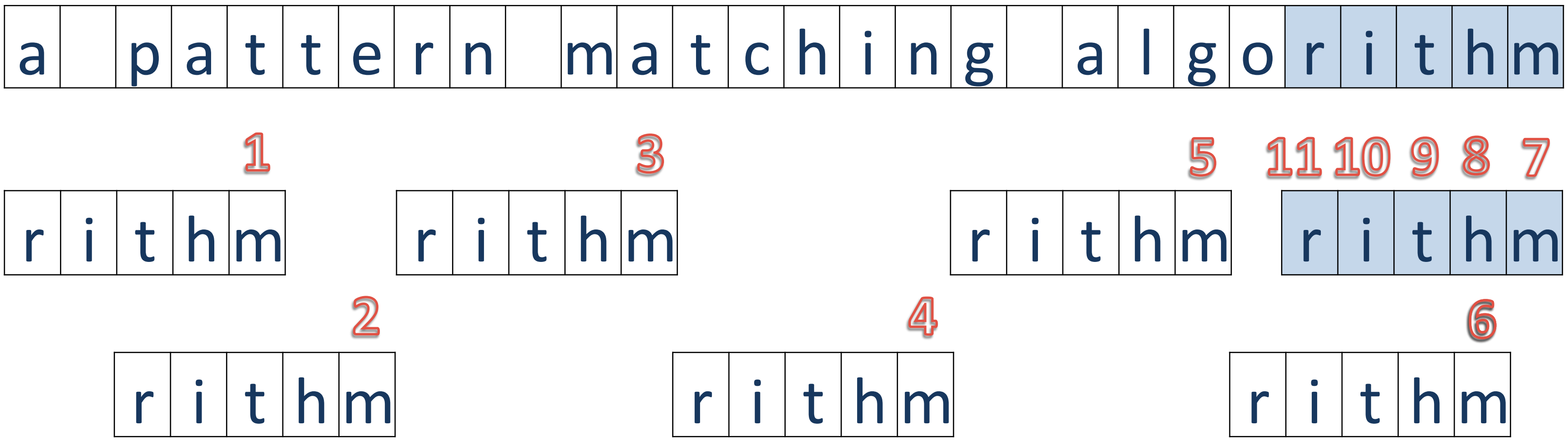


a pattern matching algorithm
1
3
5
9 8 7
rithm
rithm
rithm
rithm
2
4
6
rithm
rithm
rithm

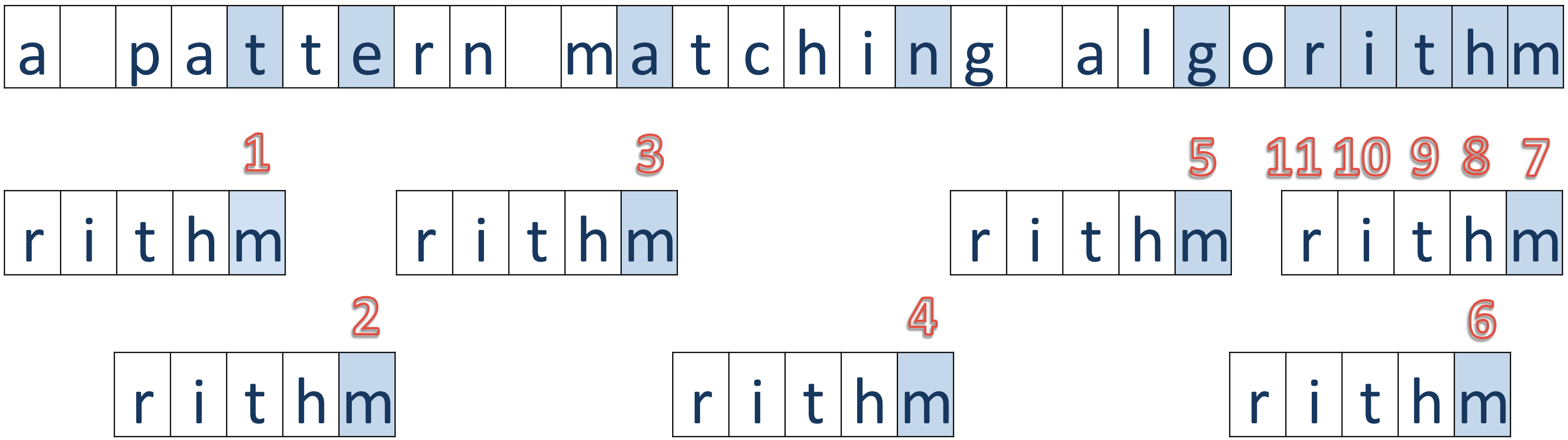
Boyer-Moore


a pattern matching algorithm
1
r i t h m
2
r i t h m
3
r i t h m
4
r i t h m
5
r i t h m
6
r i t h m
10 9 8 7
r i t h m

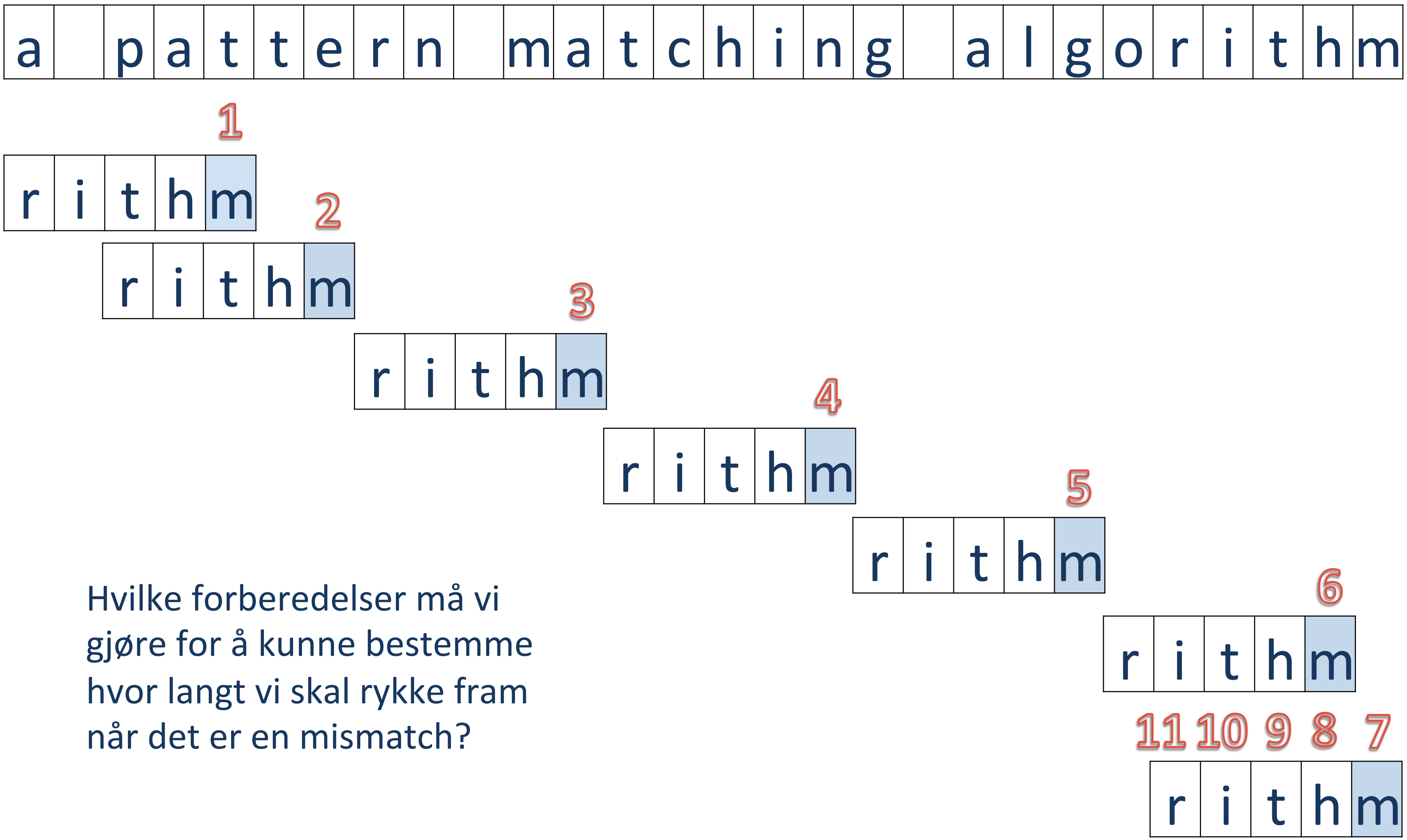
Boyer-Moore



Boyer-Moore

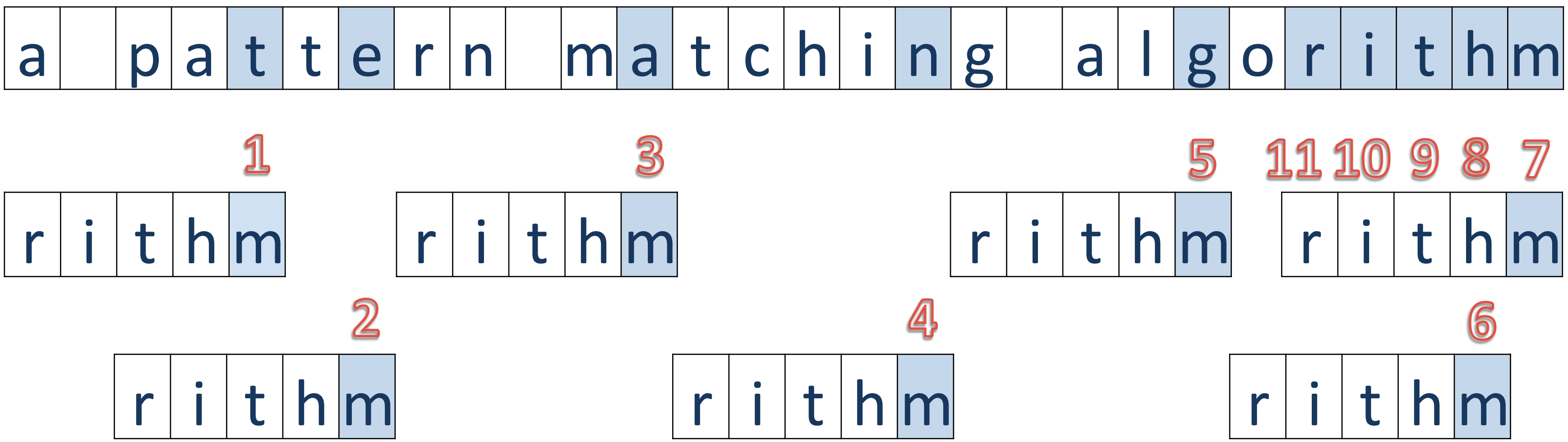

a pattern matching algorithm
1
rithm
3
rithm
5
rithm
11 10 9 8 7
rithm
2
rithm
4
rithm
6
rithm

Boyer-Moore



Hvilke forberedelser må vi gjøre for å kunne bestemme hvor langt vi skal rykke fram når det er en mismatch?

Boyer-Moore

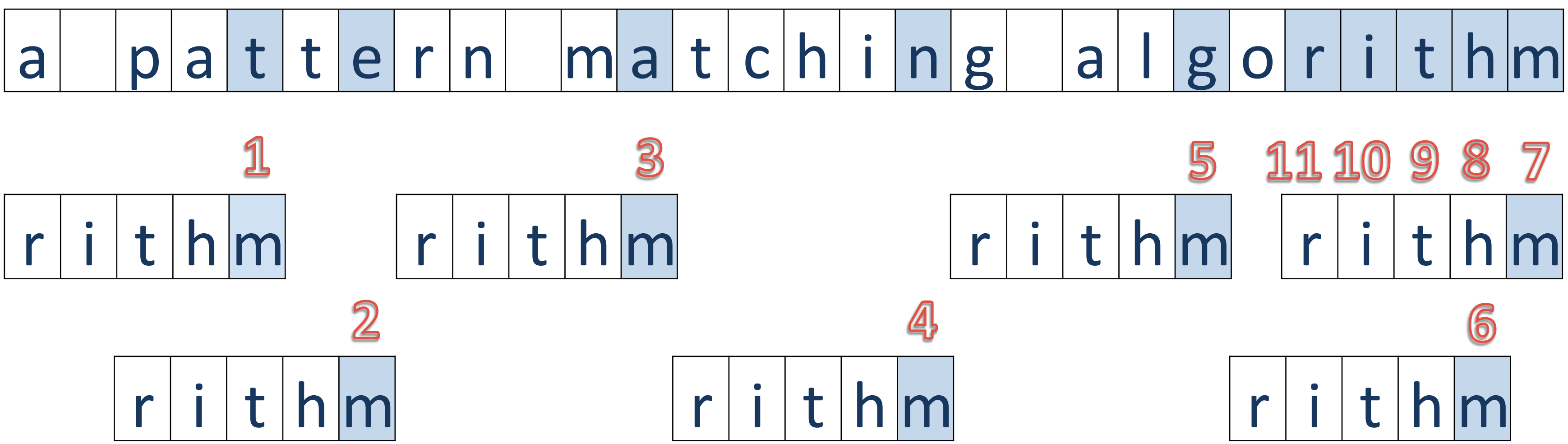


Character-jump heuristic:

When a mismatch occurs at $T[i] = c$

- If P contains c, shift P to align the last occurrence of c in P with $T[i]$
- Else, shift P to align $P[0]$ with $T[i + 1]$

Boyer-Moore



Character-jump heuristic:

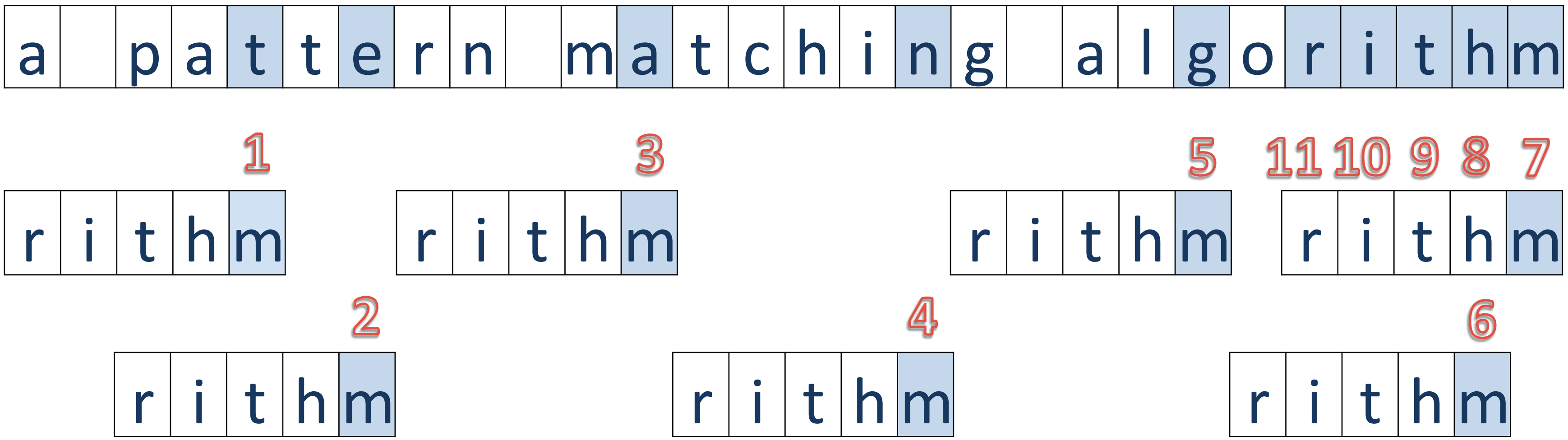
When a mismatch occurs at $T[i] = c$

- If P contains c, shift P to align the last occurrence of c in P with $T[i]$
- Else, shift P to align $P[0]$ with $T[i + 1]$

Vi må for alle tegn c i alfabetet vite:

Finnes c i mønsteret P? Hvis ja, må vi vite hvor *siste* forekomst er.

Boyer-Moore



c	a	b	c	d	e	f	g	h	i	j	k
last(c)	-1	-1	-1	-1	-1	-1	-1	3	1	-1	-1

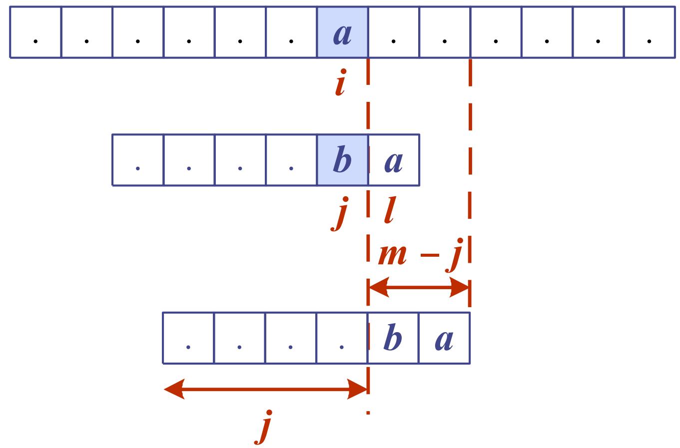
Vi må for alle tegn c i alfabetet vite:

Finnes c i mønsteret P? Hvis ja, må vi vite hvor *siste* forekomst er.

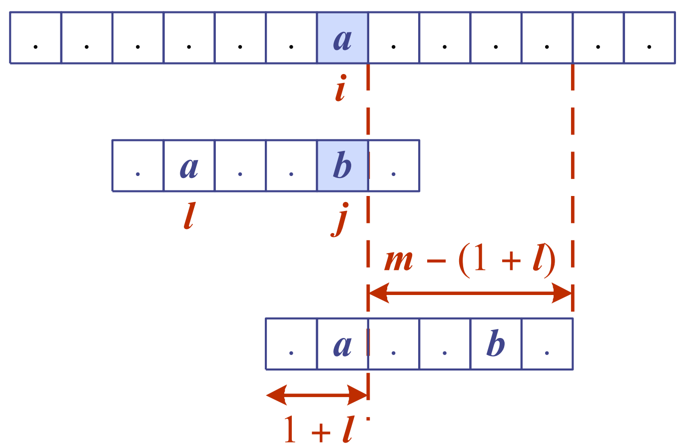
Boyer-Moore

```
Algorithm BoyerMooreMatch(T, P, Σ)
    L ← lastOccurenceFunction(P, Σ )
    i ← m − 1
    j ← m − 1
    repeat
        if T[i] = P[j]
            if  j = 0
                return  i   { match at i }
            else
                i ← i − 1
                j ← j − 1
        else
            { character-jump }
            l ← L[T[i]]
            i ← i + m – min(j, 1 + l)
            j ← m − 1
    until  i > n − 1
    return  −1  { no match }
```

Case 1: $j \leq 1 + l$

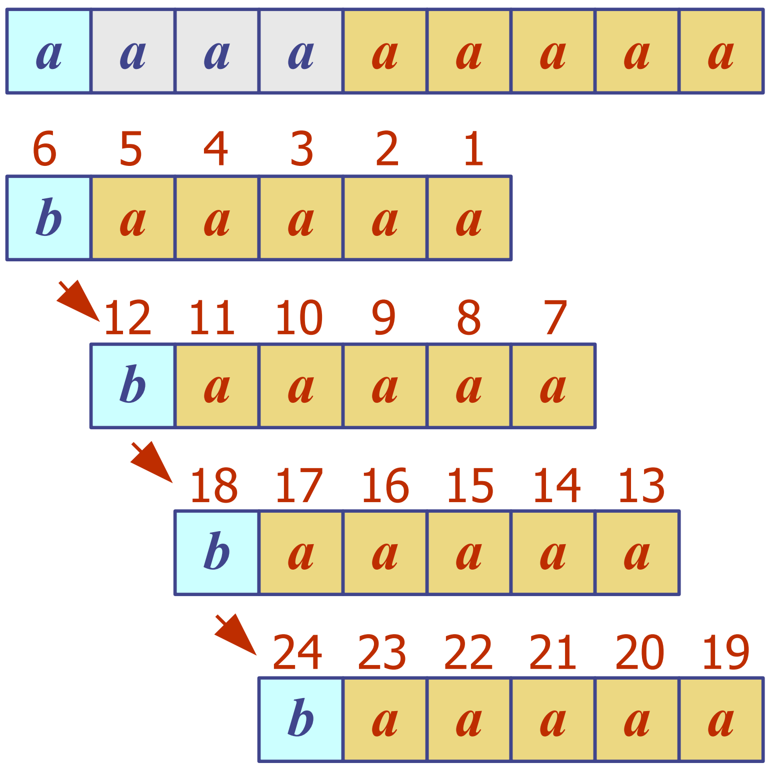


Case 2: $1 + l \leq j$



Analyse

- Kjøretiden for Boyer-Moores algoritme er $O(nm + s)$
- Worst case-eksempel:
  - $T = aaa \ldots a$
  - $P = baaa$
- Kan oppstå i bilder og DNA-sekvenser men er usannsynlig i norsk/engelsk tekst
- Algoritmen er signifikant raskere enn brute-force på norsk/engelsk tekst



Boyer-Moore

a	b	a	c	a	a	b	a	c	c	a	b	a	c	a	b	a	a	b	b

a	b	a	c	a	b

c	a	b	c
L(c)	4	5	3

Algorithm *BoyerMooreMatch*(T, P, Σ)

    $L \leftarrow$ *lastOccurenceFunction*(P, Σ)
    $i \leftarrow m - 1$
    $j \leftarrow m - 1$
    repeat
        if $T[i] = P[j]$
            if $j = 0$
                return i { match at i }
            else
                $i \leftarrow i - 1$
                $j \leftarrow j - 1$
        else
            { character-jump }
            $l \leftarrow L[T[i]]$
            $i \leftarrow i + m - \min(j, 1 + l)$
            $j \leftarrow m - 1$
    until $i > n - 1$
    return -1 { no match }

Boyer-Moore

$i = 5$

a	b	a	c	a	a	b	a	c	c	a	b	a	c	a	b	a	a	b	b

a	b	a	c	a	b

$j = 5$

$l \leftarrow L[T[i]]$
$i \leftarrow i + m - \min(j, 1 + l)$
$j \leftarrow m - 1$

c	a	b	c
L(c)	4	5	3

Algorithm ***BoyerMooreMatch***(T, P, Σ)
    $L \leftarrow lastOccurenceFunction(P, \Sigma)$
    $i \leftarrow m - 1$
    $j \leftarrow m - 1$
    repeat
        if $T[i] = P[j]$
            if $j = 0$
                return i { match at i }
            else
                $i \leftarrow i - 1$
                $j \leftarrow j - 1$
        else
            { character-jump }
            $l \leftarrow L[T[i]]$
            $i \leftarrow i + m - \min(j, 1 + l)$
            $j \leftarrow m - 1$
    until $i > n - 1$
    return -1 { no match }

Boyer-Moore

$i = 5$

a	b	a	c	a	a	b	a	c	c	a	b	a	c	a	b	a	a	b	b

a	b	a	c	a	b

$j = 5$

$l \leftarrow L[T[5]]$
$i \leftarrow i + m - \min(j, 1 + l)$
$j \leftarrow m - 1$

c	a	b	c
L(c)	4	5	3

Algorithm $BoyerMooreMatch(T, P, \Sigma)$
    $L \leftarrow lastOccurenceFunction(P, \Sigma)$
    $i \leftarrow m - 1$
    $j \leftarrow m - 1$
    repeat
        if $T[i] = P[j]$
            if $j = 0$
                return i { match at i }
            else
                $i \leftarrow i - 1$
                $j \leftarrow j - 1$
        else
            { character-jump }
            $l \leftarrow L[T[i]]$
            $i \leftarrow i + m - \min(j, 1 + l)$
            $j \leftarrow m - 1$
    until $i > n - 1$
    return -1 { no match }

Boyer-Moore

$i = 5$

a	b	a	c	a	a	b	a	c	c	a	b	a	c	a	b	a	a	b	b

a	b	a	c	a	b

$j = 5$

$l \leftarrow L[\mathrm{a}]$
$i \leftarrow i + m - \min(j, 1 + l)$
$j \leftarrow m - 1$

c	a	b	c
L(c)	4	5	3

Algorithm *BoyerMooreMatch*(T, P, Σ)
  $L \leftarrow$ *lastOccurenceFunction*(P, Σ)
  $i \leftarrow m - 1$
  $j \leftarrow m - 1$
  repeat
    if $T[i] = P[j]$
      if $j = 0$
        return i { match at i }
      else
        $i \leftarrow i - 1$
        $j \leftarrow j - 1$
    else
      { character-jump }
      $l \leftarrow L[T[i]]$
      $i \leftarrow i + m - \min(j, 1 + l)$
      $j \leftarrow m - 1$
  until $i > n - 1$
  return -1 { no match }

Boyer-Moore

$i = 5$

a	b	a	c	a	a	b	a	c	c	a	b	a	c	a	b	a	a	b	b

a	b	a	c	a	b

$j = 5$

$l \leftarrow 4$
$i \leftarrow i + m - \min(j, 1 + l)$
$j \leftarrow m - 1$

c	a	b	c
L(c)	4	5	3

Algorithm *BoyerMooreMatch*(T, P, Σ)
    $L \leftarrow lastOccurenceFunction(P, \Sigma)$
    $i \leftarrow m - 1$
    $j \leftarrow m - 1$
    repeat
        if $T[i] = P[j]$
            if $j = 0$
                return i { match at i }
            else
                $i \leftarrow i - 1$
                $j \leftarrow j - 1$
        else
            { character-jump }
            $l \leftarrow L[T[i]]$
            $i \leftarrow i + m - \min(j, 1 + l)$
            $j \leftarrow m - 1$
    until $i > n - 1$
    return -1 { no match }

Boyer-Moore

$i = 5$

a	b	a	c	a	a	b	a	c	c	a	b	a	c	a	b	a	a	b	b

a	b	a	c	a	b

$j = 5$

$l \leftarrow 4$
$i \leftarrow 5 + 6 - \min(5, 1 + 4)$
$j \leftarrow m - 1$

c	a	b	c
L(c)	4	5	3

```
Algorithm BoyerMooreMatch(T, P, Σ)
    L ← lastOccurenceFunction(P, Σ )
    i ← m − 1
    j ← m − 1
    repeat
        if T[i] = P[j]
            if  j = 0
                return  i  { match at i }
            else
                i ← i − 1
                j ← j − 1
        else
            { character-jump }
            l ← L[T[i]]
            i ← i + m – min(j, 1 + l)
            j ← m − 1
    until  i > n − 1
    return  −1  { no match }
```

Boyer-Moore

$i = 5$

a	b	a	c	a	a	b	a	c	c	a	b	a	c	a	b	a	a	b	b

a	b	a	c	a	b

$j = 5$

$l \leftarrow 4$

$i \leftarrow 5 + 6 - \min(5, \quad 5 \quad)$

$j \leftarrow m - 1$

c	a	b	c
L(c)	4	5	3

```
Algorithm BoyerMooreMatch(T, P, Σ)
    L ← lastOccurenceFunction(P, Σ )
    i ← m − 1
    j ← m − 1
    repeat
        if T[i] = P[j]
            if j = 0
                return i { match at i }
            else
                i ← i − 1
                j ← j − 1
        else
            { character-jump }
            l ← L[T[i]]
            i ← i + m – min(j, 1 + l)
            j ← m − 1
    until i > n − 1
    return −1 { no match }
```

Boyer-Moore

$i = 5$

a	b	a	c	a	a	b	a	c	c	a	b	a	c	a	b	a	a	b	b

a	b	a	c	a	b

$j = 5$

$l \leftarrow 4$
$i \leftarrow 5 + 6 - 5$
$j \leftarrow m - 1$

c	a	b	c
L(c)	4	5	3

```
Algorithm BoyerMooreMatch(T, P, Σ)
    L ← lastOccurenceFunction(P, Σ )
    i ← m − 1
    j ← m − 1
    repeat
        if T[i] = P[j]
            if j = 0
                return i  { match at i }
            else
                i ← i − 1
                j ← j − 1
        else
            { character-jump }
            l ← L[T[i]]
            i ← i + m – min(j, 1 + l)
            j ← m − 1
    until i > n − 1
    return −1 { no match }
```

Boyer-Moore

$i = 5$

a	b	a	c	a	a	b	a	c	c	a	b	a	c	a	b	a	a	b	b

a	b	a	c	a	b

$j = 5$

$l \leftarrow 4$
$i \leftarrow 6$
$j \leftarrow m - 1$

c	a	b	c
L(c)	4	5	3

Algorithm *BoyerMooreMatch*(T, P, Σ)
    $L \leftarrow lastOccurenceFunction(P, \Sigma)$
    $i \leftarrow m - 1$
    $j \leftarrow m - 1$
    repeat
        if $T[i] = P[j]$
            if $j = 0$
                return i { match at i }
            else
                $i \leftarrow i - 1$
                $j \leftarrow j - 1$
        else
            { character-jump }
            $l \leftarrow L[T[i]]$
            $i \leftarrow i + m - \min(j, 1 + l)$
            $j \leftarrow m - 1$
    until $i > n - 1$
    return -1 { no match }

Boyer-Moore

$i = 5$

a	b	a	c	a	a	b	a	c	c	a	b	a	c	a	b	a	a	b	b

a	b	a	c	a	b

$j = 5$

$l \leftarrow 4$
$i \leftarrow 6$
$j \leftarrow 5$

c	a	b	c
L(c)	4	5	3

Algorithm *BoyerMooreMatch*(T, P, Σ)
  $L \leftarrow$ *lastOccurenceFunction*(P, Σ)
  $i \leftarrow m - 1$
  $j \leftarrow m - 1$
  repeat
    if $T[i] = P[j]$
      if $j = 0$
        return i { match at i }
      else
        $i \leftarrow i - 1$
        $j \leftarrow j - 1$
    else
      { character-jump }
      $l \leftarrow L[T[i]]$
      $i \leftarrow i + m - \min(j, 1 + l)$
      $j \leftarrow m - 1$
  until $i > n - 1$
  return -1 { no match }

Boyer-Moore

$i = 6$

a	b	a	c	a	a	b	a	c	c	a	b	a	c	a	b	a	a	b	b

a	b	a	c	a	b

$j = 5$

$l \leftarrow 4$
$i \leftarrow 6$
$j \leftarrow 5$

c	a	b	c
L(c)	4	5	3

Algorithm *BoyerMooreMatch*(T, P, Σ)
  $L \leftarrow$ *lastOccurenceFunction*(P, Σ)
  $i \leftarrow m - 1$
  $j \leftarrow m - 1$
  repeat
    if $T[i] = P[j]$
      if $j = 0$
        return i { match at i }
      else
        $i \leftarrow i - 1$
        $j \leftarrow j - 1$
    else
      { character-jump }
      $l \leftarrow L[T[i]]$
      $i \leftarrow i + m - \min(j, 1 + l)$
      $j \leftarrow m - 1$
  until $i > n - 1$
  return -1 { no match }

Boyer-Moore

$i = 6$

a	b	a	c	a	a	b	a	c	c	a	b	a	c	a	b	a	a	b	b

a	b	a	c	a	b

$j = 5$

c	a	b	c
L(c)	4	5	3

Algorithm *BoyerMooreMatch*(T, P, Σ)

- $L \leftarrow lastOccurenceFunction(P, \Sigma)$
- $i \leftarrow m - 1$
- $j \leftarrow m - 1$
- **repeat**
  - **if** $T[i] = P[j]$
    - **if** $j = 0$
      - **return** i { match at i }
    - **else**
      - $i \leftarrow i - 1$
      - $j \leftarrow j - 1$
  - **else**
    - { character-jump }
    - $l \leftarrow L[T[i]]$
    - $i \leftarrow i + m - \min(j, 1 + l)$
    - $j \leftarrow m - 1$
- **until** $i > n - 1$
- **return** -1 { no match }

Boyer-Moore

$i = 4$

a	b	a	c	a	a	b	a	c	c	a	b	a	c	a	b	a	a	b	b

a	b	a	c	a	b

$j = 3$

Algorithm *BoyerMooreMatch*(T, P, Σ)
  $L \leftarrow lastOccurenceFunction(P, \Sigma)$
  $i \leftarrow m - 1$
  $j \leftarrow m - 1$
  repeat
    if $T[i] = P[j]$
      if $j = 0$
        return i { match at i }
      else
        $i \leftarrow i - 1$
        $j \leftarrow j - 1$
    else
      { character-jump }
      $l \leftarrow L[T[i]]$
      $i \leftarrow i + m - \min(j, 1 + l)$
      $j \leftarrow m - 1$
  until $i > n - 1$
  return -1 { no match }

c	a	b	c
L(c)	4	5	3

Boyer-Moore

$i = 4$

a	b	a	c	a	a	b	a	c	c	a	b	a	c	a	b	a	a	b	b

a	b	a	c	a	b

$j = 3$

c	a	b	c
L(c)	4	5	3

```
Algorithm BoyerMooreMatch(T, P, Σ)
    L ← lastOccurenceFunction(P, Σ )
    i ← m − 1
    j ← m − 1
    repeat
        if T[i] = P[j]
            if  j = 0
                return  i   { match at i }
            else
                i ← i − 1
                j ← j − 1
        else
            { character-jump }
            l ← L[T[i]]
            i ← i + m – min(j, 1 + l)
            j ← m − 1
    until  i > n − 1
    return  −1  { no match }
```

Boyer-Moore

i = 7

a	b	a	c	a	a	b	a	c	c	a	b	a	c	a	b	a	a	b	b

a	b	a	c	a	b

j = 5

c	a	b	c
L(c)	4	5	3

```
Algorithm BoyerMooreMatch(T, P, Σ)
    L ← lastOccurenceFunction(P, Σ )
    i ← m − 1
    j ← m − 1
    repeat
        if T[i] = P[j]
            if j = 0
                return i  { match at i }
            else
                i ← i − 1
                j ← j − 1
        else
            { character-jump }
            l ← L[T[i]]
            i ← i + m – min(j, 1 + l)
            j ← m − 1
    until i > n − 1
    return −1 { no match }
```

Boyer-Moore

$i = 7$

a	b	a	c	a	a	b	a	c	c	a	b	a	c	a	b	a	a	b	b

a	b	a	c	a	b

$j = 5$

c	a	b	c
L(c)	4	5	3

Algorithm *BoyerMooreMatch*(T, P, Σ)

$L \leftarrow lastOccurenceFunction(P, \Sigma)$
$i \leftarrow m - 1$
$j \leftarrow m - 1$
repeat
  if $T[i] = P[j]$
    if $j = 0$
      return i { match at i }
    else
      $i \leftarrow i - 1$
      $j \leftarrow j - 1$
  else
    { character-jump }
    $l \leftarrow L[T[i]]$
    $i \leftarrow i + m - \min(j, 1 + l)$
    $j \leftarrow m - 1$
until $i > n - 1$
return -1 { no match }

Boyer-Moore

$i = 8$

a	b	a	c	a	a	b	a	c	c	a	b	a	c	a	b	a	a	b	b

a	b	a	c	a	b

$j = 5$

c	a	b	c
L(c)	4	5	3

Algorithm *BoyerMooreMatch*(T, P, Σ)
    $L \leftarrow lastOccurenceFunction(P, \Sigma)$
    $i \leftarrow m - 1$
    $j \leftarrow m - 1$
    repeat
        if $T[i] = P[j]$
            if $j = 0$
                return i { match at i }
            else
                $i \leftarrow i - 1$
                $j \leftarrow j - 1$
        else
            { character-jump }
            $l \leftarrow L[T[i]]$
            $i \leftarrow i + m - \min(j, 1 + l)$
            $j \leftarrow m - 1$
    until $i > n - 1$
    return -1 { no match }

Boyer-Moore

$i = 8$

a	b	a	c	a	a	b	a	c	c	a	b	a	c	a	b	a	a	b	b

a	b	a	c	a	b

$j = 5$

c	a	b	c
L(c)	4	5	3

Algorithm *BoyerMooreMatch*(T, P, Σ)
$\quad L \leftarrow lastOccurenceFunction(P, \Sigma)$
$\quad i \leftarrow m - 1$
$\quad j \leftarrow m - 1$
$\quad$ **repeat**
$\quad\quad$ **if** $T[i] = P[j]$
$\quad\quad\quad$ **if** $j = 0$
$\quad\quad\quad\quad$ **return** i { match at i }
$\quad\quad\quad$ **else**
$\quad\quad\quad\quad i \leftarrow i - 1$
$\quad\quad\quad\quad j \leftarrow j - 1$
$\quad\quad$ **else**
$\quad\quad\quad$ { character-jump }
$\quad\quad\quad l \leftarrow L[T[i]]$
$\quad\quad\quad i \leftarrow i + m - \min(j, 1 + l)$
$\quad\quad\quad j \leftarrow m - 1$
$\quad$ **until** $i > n - 1$
$\quad$ **return** -1 { no match }

Boyer-Moore

$i = 8$

a	b	a	c	a	a	b	a	c	c	a	b	a	c	a	b	a	a	b	b

a	b	a	c	a	b

$j = 5$

c	a	b	c
L(c)	4	5	3

Algorithm *BoyerMooreMatch*(T, P, Σ)

$L \leftarrow lastOccurenceFunction(P, \Sigma)$
$i \leftarrow m - 1$
$j \leftarrow m - 1$
repeat
  if $T[i] = P[j]$
    if $j = 0$
      return i { match at i }
    else
      $i \leftarrow i - 1$
      $j \leftarrow j - 1$
  else
    { character-jump }
    $l \leftarrow L[T[i]]$
    $i \leftarrow i + m - \min(j, 1 + l)$
    $j \leftarrow m - 1$
until $i > n - 1$
return -1 { no match }

Boyer-Moore

$i = 8$

a	b	a	c	a	a	b	a	c	c	a	b	a	c	a	b	a	a	b	b

a	b	a	c	a	b

$j = 5$

Algorithm *BoyerMooreMatch*(T, P, Σ)

```
L ← lastOccurenceFunction(P, Σ )
i ← m − 1
j ← m − 1
repeat
    if T[i] = P[j]
        if j = 0
            return i   { match at i }
        else
            i ← i − 1
            j ← j − 1
    else
        { character-jump }
        l ← L[T[i]]
        i ← i + m – min(j, 1 + l)
        j ← m − 1
until i > n − 1
return −1 { no match }
```

c	a	b	c
L(c)	4	5	3

Boyer-Moore

$i = 10$

a	b	a	c	a	a	b	a	c	c	a	b	a	c	a	b	a	a	b	b

a	b	a	c	a	b

$j = 5$

Boyer-Moore

$i = 10$

a	b	a	c	a	a	b	a	c	c	a	b	a	c	a	b	a	a	b	b

a	b	a	c	a	b

$j = 5$

Boyer-Moore

a	b	a	c	a	a	b	a	c	c	a	b	a	c	a	b	a	a	b	b

a	b	a	c	a	b

Boyer-Moore

$i = 8$

a	b	a	c	a	a	b	a	c	c	a	b	a	c	a	b	a	a	b	b

a	b	a	c	a	b

$j = 2$

c	a	b	c
L(c)	4	5	3

Boyer-Moore

$i = 12$

a	b	a	c	a	a	b	a	c	c	a	b	a	c	a	b	a	a	b	b

a	b	a	c	a	b

$j = 5$

c	a	b	c
L(c)	4	5	3

Boyer-Moore

$i = 12$

a	b	a	c	a	a	b	a	c	c	a	b	a	c	a	b	a	a	b	b

a	b	a	c	a	b

$j = 5$

c	a	b	c
L(c)	4	5	3

Boyer-Moore

$i = 12$

a	b	a	c	a	a	b	a	c	c	a	b	a	c	a	b	a	a	b	b

a	b	a	c	a	b

$j = 5$

c	a	b	c
L(c)	4	5	3

Boyer-Moore

$i = 13$

a	b	a	c	a	a	b	a	c	c	a	b	a	c	a	b	a	a	b	b

a	b	a	c	a	b

$j = 5$

c	a	b	c
L(c)	4	5	3

Boyer-Moore

$i = 15$

a	b	a	c	a	a	b	a	c	c	a	b	a	c	a	b	a	a	b	b

a	b	a	c	a	b

$j = 5$

c	a	b	c
L(c)	4	5	3

Boyer-Moore

$i = 10$

a	b	a	c	a	a	b	a	c	c	a	b	a	c	a	b	a	a	b	b

a	b	a	c	a	b

Boyer-Moore

a	b	a	c	a	a	b	a	c	c	a	b	a	c	a	b	a	a	b	b

a	b	a	c	a	b

Men hva er svakheten til Boyer-More ?

Boyer-Moore

$i = 8$

a	b	a	c	a	a	b	a	c	c	a	b	a	c	a	b	a	a	b	b

a	b	a	c	a	b

$j = 2$

Boyer-Moore

$i = 12$

a	b	a	c	a	a	b	a	c	c	a	b	a	c	a	b	a	a	b	b

a	b	a	c	a	b

$j = 5$

Boyer-Moore

$i = 8$

a	b	a	c	a	a	b	a	c	c	a	b	a	c	a	b	a	a	b	b

a	b	a	c	a	b

$j = 2$

Boyer-Moore

$i = 8$

a	b	a	c	a	a	b	a	c	c	a	b	a	c	a	b	a	a	b	b

a	b	a	c	a	b

$j = 2$

Boyer-Moore

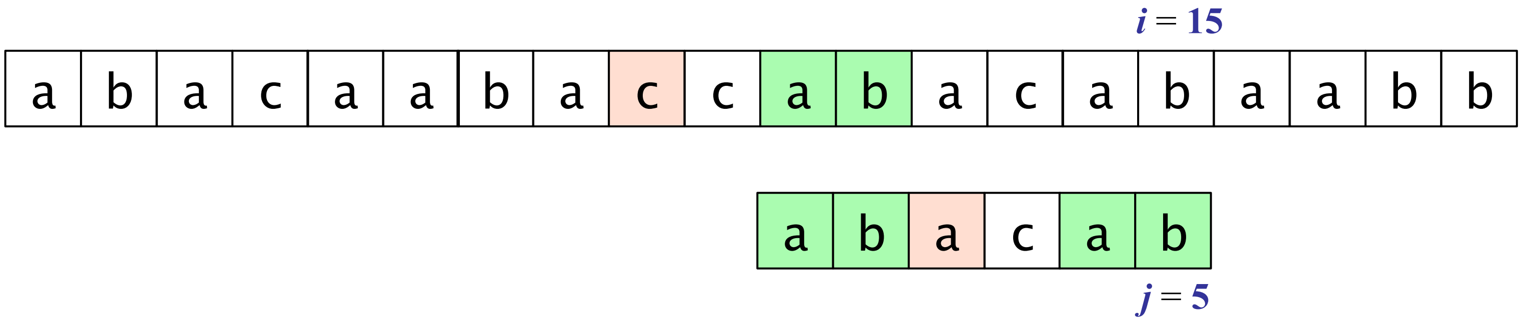
$i = 8$

a	b	a	c	a	a	b	a	c	c	a	b	a	c	a	b	a	a	b	b

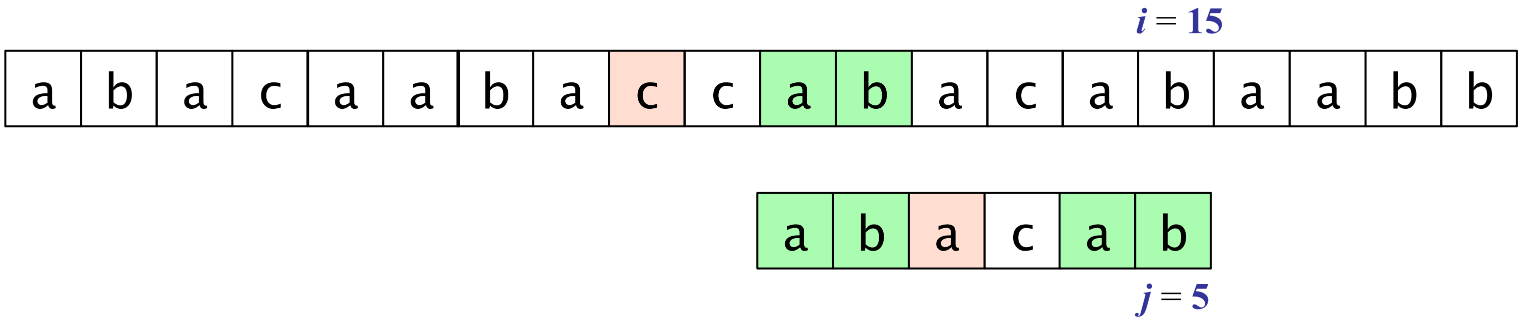
a	b	a	c	a	b

$j = 2$

Boyer-Moore



Boyer-Moore med good suffix shift



Boyer-Moore med good suffix shift

$i = 8$

a	b	a	c	a	a	b	a	c	c	a	b	a	c	a	b	a	a	b	b

a	b	a	c	a	b

$j = 2$

Boyer-Moore med good suffix shift

$i = 15$

a	b	a	c	a	a	b	a	c	c	a	b	a	c	a	b	a	a	b	b

a	b	a	c	a	b

$j = 5$

Knuth-Morris-Pratt-algoritmen

a	b	a	c	a	a	b	a	c	c	a	b	a	c	a	b	a	a	b	b

a	b	a	c	a	b

Knuth-Morris-Pratt-algoritmen

i

a	b	a	c	a	a	b	a	c	c	a	b	a	c	a	b	a	a	b	b

j

a	b	a	c	a	b

Knuth-Morris-Pratt-algoritmen

i

a	b	a	c	a	a	b	a	c	c	a	b	a	c	a	b	a	a	b	b

j

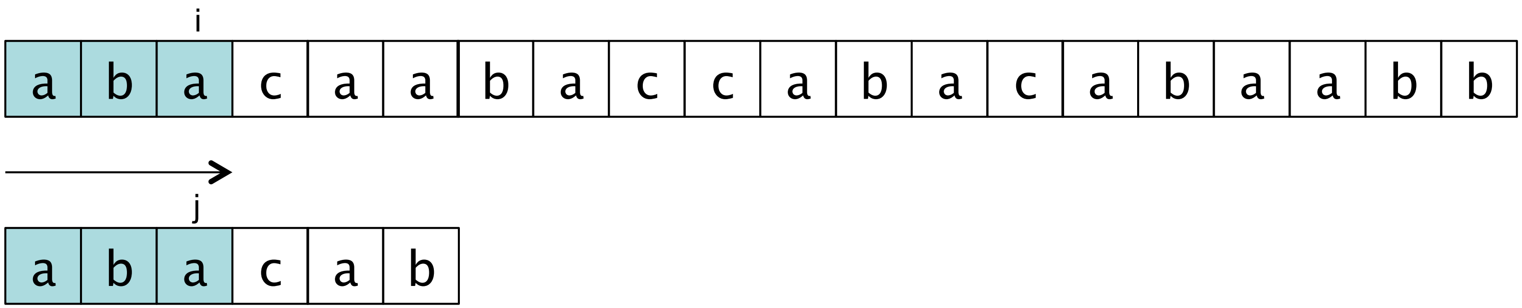
a	b	a	c	a	b

Knuth-Morris-Pratt-algoritmen

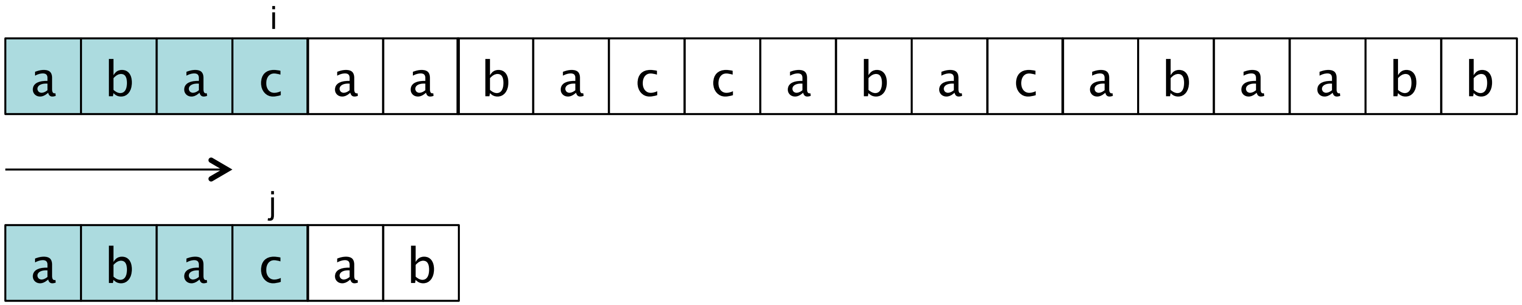
	i																		
a	b	a	c	a	a	b	a	c	c	a	b	a	c	a	b	a	a	b	b

	j				
a	b	a	c	a	b

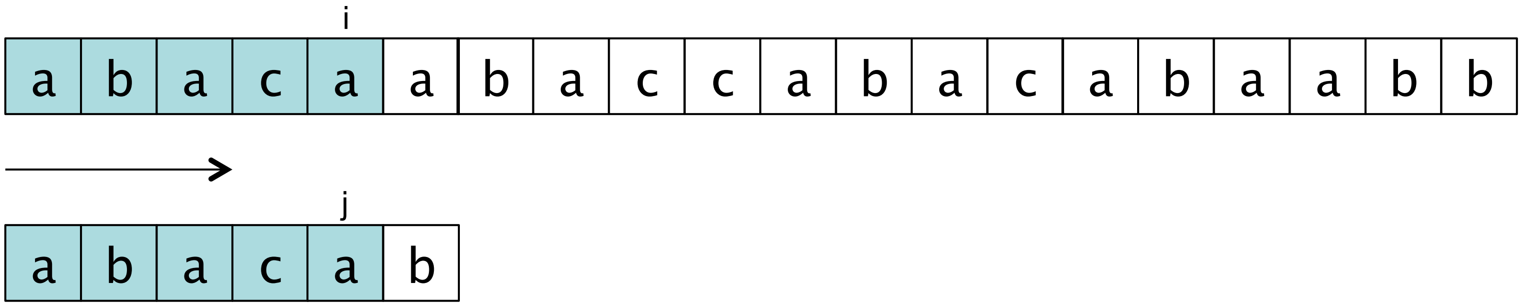
Knuth-Morris-Pratt-algoritmen



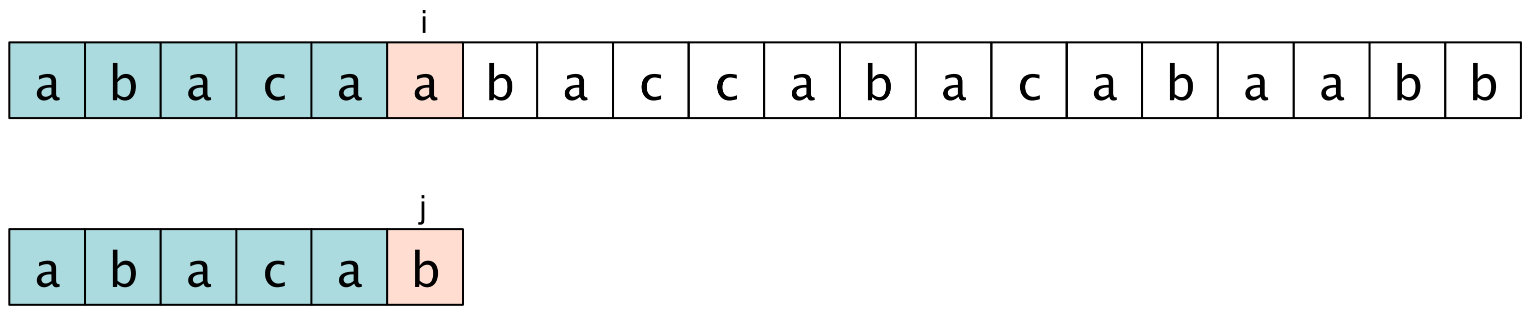
Knuth-Morris-Pratt-algoritmen



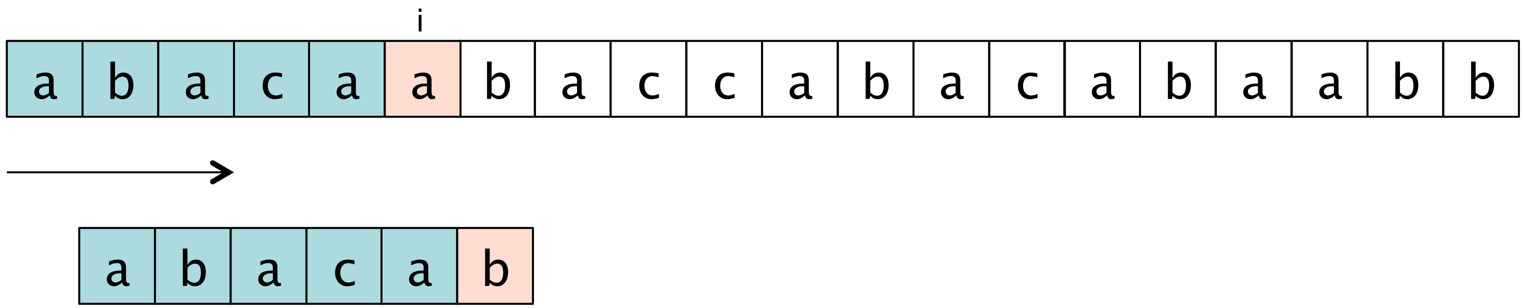
Knuth-Morris-Pratt-algoritmen



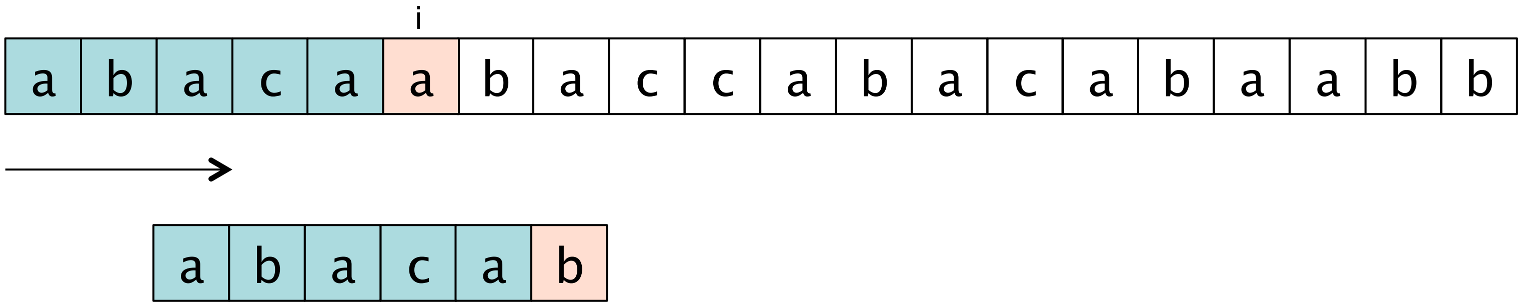
Knuth-Morris-Pratt-algoritmen



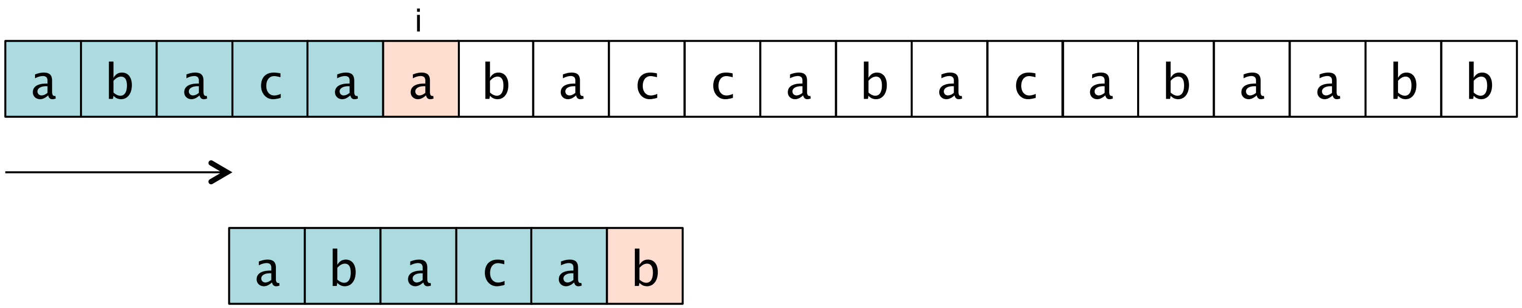
Knuth-Morris-Pratt-algoritmen



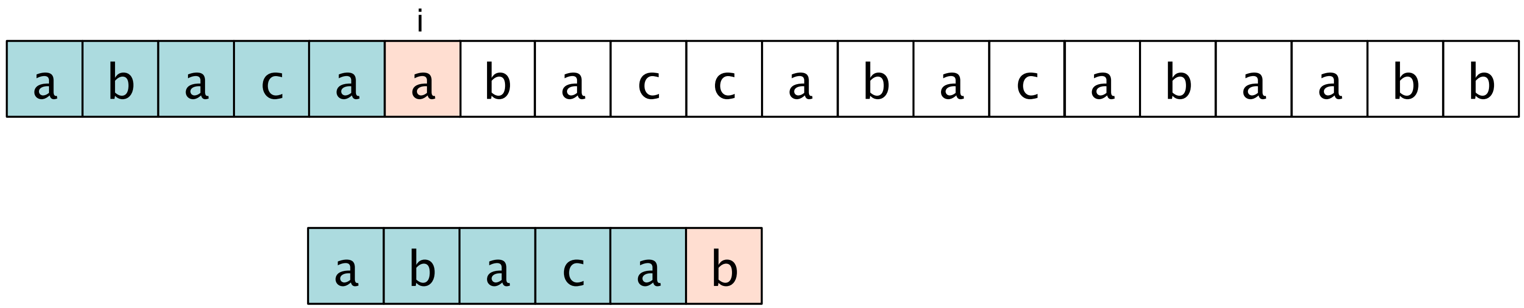
Knuth-Morris-Pratt-algoritmen



Knuth-Morris-Pratt-algoritmen



Knuth-Morris-Pratt-algoritmen



Knuth-Morris-Pratt-algoritmen

					i														
a	b	a	c	a	a	b	a	c	c	a	b	a	c	a	b	a	a	b	b

					j
a	b	a	c	a	b

Ingen sammenligning
nødvendig her

Knuth-Morris-Pratt-algoritmen

					i														
a	b	a	c	a	a	b	a	c	c	a	b	a	c	a	b	a	a	b	b
					j														
				a	b	a	c	a	b										

Første sammenligning
etter flytting av j
($j \leftarrow 1$)

Knuth-Morris-Pratt-algoritmen

					i														
a	b	a	c	a	a	b	a	c	c	a	b	a	c	a	b	a	a	b	b

	j				
a	b	a	c	a	b

Mismatch, og vi flytter ett hakk
($j \leftarrow 0$)

Knuth-Morris-Pratt-algoritmen

					i														
a	b	a	c	a	a	b	a	c	c	a	b	a	c	a	b	a	a	b	b

j					
a	b	a	c	a	b

Mismatch, og vi flytter ett hakk

($j \leftarrow 0$)

Knuth-Morris-Pratt-algoritmen

					i														
a	b	a	c	a	a	b	a	c	c	a	b	a	c	a	b	a	a	b	b

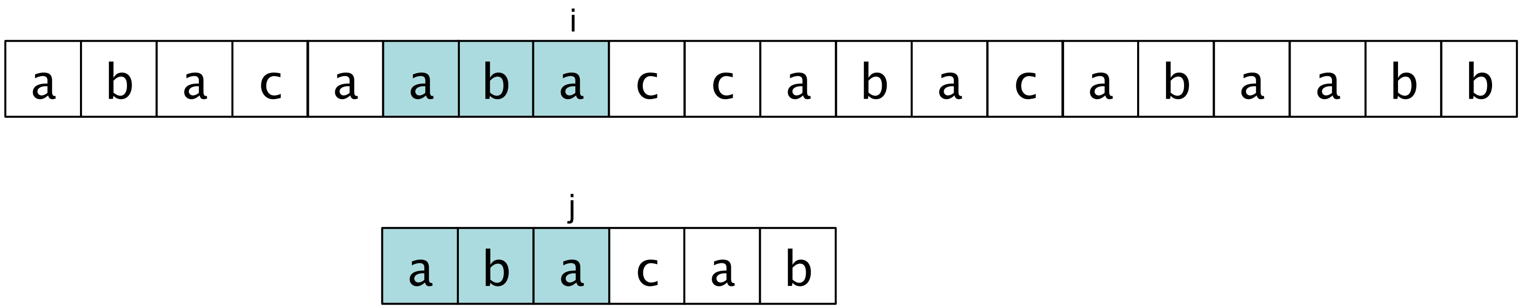
j					
a	b	a	c	a	b

Knuth-Morris-Pratt-algoritmen

						i													
a	b	a	c	a	a	b	a	c	c	a	b	a	c	a	b	a	a	b	b

	j				
a	b	a	c	a	b

Knuth-Morris-Pratt-algoritmen



Knuth-Morris-Pratt-algoritmen

								i											
a	b	a	c	a	a	b	a	c	c	a	b	a	c	a	b	a	a	b	b

			j		
a	b	a	c	a	b

Knuth-Morris-Pratt-algoritmen

									i										
a	b	a	c	a	a	b	a	c	c	a	b	a	c	a	b	a	a	b	b

				j	
a	b	a	c	a	b

Mismatch, og vi flytter 4 hakk
($j \leftarrow 0$)

Knuth-Morris-Pratt-algoritmen

									i										
a	b	a	c	a	a	b	a	c	c	a	b	a	c	a	b	a	a	b	b

j					
a	b	a	c	a	b

Mismatch, og vi flytter 4 hakk

($j \leftarrow 0$)

Knuth-Morris-Pratt-algoritmen

									i										
a	b	a	c	a	a	b	a	c	c	a	b	a	c	a	b	a	a	b	b
									j										
									a	b	a	c	a	b					

Mismatch, og vi flytter 1 hakk
(i ← 0)

Knuth-Morris-Pratt-algoritmen

										i									
a	b	a	c	a	a	b	a	c	c	a	b	a	c	a	b	a	a	b	b

j					
a	b	a	c	a	b

Knuth-Morris-Pratt-algoritmen

a	b	a	c	a	a	b	a	c	c	a	b	a	c	a	b	a	a	b	b
1	2	3	4	5	6														
a	b	a	c	a	b														
					7														
				a	b	a	c	a	b										
					8	9	10	11	12										
					a	b	a	c	a	b									
									13										
									a	b	a	c	a	b					
										14	15	16	17	18	19				
										a	b	a	c	a	b				

j	0	1	2	3	4	5
$P[j]$	*a*	*b*	*a*	*c*	*a*	*b*
$F(j)$	0	0	1	0	1	2

Knuth-Morris-Pratt-algoritmen
the failure function

Også kjent som «partial match-table»

j	0	1	2	3	4	5
$P[j]$	a	b	a	c	a	b
$F(j)$	0	0	1	0	1	2

Knuth-Morris-Pratt-algoritmen
the failure function

Også kjent som «partial match-table»

- Knuth-Morris-Pratt's algorithm preprocesses the pattern to find matches of prefixes of the pattern with the pattern itself

j	0	1	2	3	4	5
$P[j]$	***a***	***b***	***a***	***c***	***a***	***b***
$F(j)$	0	0	1	0	1	2

Knuth-Morris-Pratt-algoritmen
the failure function

Også kjent som «partial match-table»

- Knuth-Morris-Pratt's algorithm preprocesses the pattern to find matches of prefixes of the pattern with the pattern itself
- The failure function $F(j)$ is defined as the size of the largest prefix of $P[0..j]$ that is also a suffix of $P[1..j]$

j	0	1	2	3	4	5
$P[j]$	a	b	a	c	a	b
$F(j)$	0	0	1	0	1	2

Knuth-Morris-Pratt-algoritmen the failure function

Også kjent som «partial match-table»

- Knuth-Morris-Pratt's algorithm preprocesses the pattern to find matches of prefixes of the pattern with the pattern itself
- The failure function $F(j)$ is defined as the size of the largest prefix of $P[0..j]$ that is also a suffix of $P[1..j]$
- Knuth-Morris-Pratt's algorithm modifies the brute-force algorithm so that if a mismatch occurs at $P[j] \neq T[i]$ we set $j \leftarrow F(j-1)$

j	0	1	2	3	4	5
$P[j]$	a	b	a	c	a	b
$F(j)$	0	0	1	0	1	2

j	0	1	2	3	4	5
$P[j]$	a	b	a	c	a	b
$F(j)$	0					

$F(j)$ er definert som lengden av lengste prefikset i $P[0..j]$ som også er suffiks i $P[1..j]$

j	0	1	2	3	4	5
$P[j]$	***a***	***b***	***a***	***c***	***a***	***b***
$F(j)$	0					

$F(j)$ er definert som lengden av lengste prefikset i $P[0..j]$ som også er suffiks i $P[1..j]$

$P[0..1]$ = ab
$P[1..1]$ = b

Altså lengste prefiks i ‘ab’ som også er suffiks i ‘b’

j	0	1	2	3	4	5
$P[j]$	a	b	a	c	a	b
$F(j)$	0	0				

$F(j)$ er definert som lengden av lengste prefikset i $P[0..j]$ som også er suffiks i $P[1..j]$

$P[0..1]$ = ab
$P[1..1]$ = b

Altså lengste prefiks i ‘ab’ som også er suffiks i ‘b’

j	0	1	2	3	4	5
P[*j*]	***a***	***b***	***a***	***c***	***a***	***b***
F(*j*)	0	0				

F(***j***) er definert som lengden av lengste prefikset i ***P***[0..***j***] som også er suffiks i ***P***[1..***j***]

P[0..2] = aba
P[1..2] = ba

Altså lengste prefiks i 'aba' som også er suffiks i 'ba'

j	0	1	2	3	4	5
$P[j]$	*a*	*b*	*a*	*c*	*a*	*b*
$F(j)$	0	0				

$F(j)$ er definert som lengden av lengste prefikset i $P[0..j]$ som også er suffiks i $P[1..j]$

$P[0..2]$ = aba
$P[1..2]$ = ba

Altså lengste prefiks i 'ab' som også er suffiks i 'ba'

j	0	1	2	3	4	5
$P[j]$	a	b	a	c	a	b
$F(j)$	0	0	1			

$F(j)$ er definert som lengden av lengste prefikset i $P[0..j]$ som også er suffiks i $P[1..j]$

$P[0..2]$ = aba
$P[1..2]$ = ba

Altså lengste prefiks i 'ab' som også er suffiks i 'ba'

j	0	1	2	3	4	5
$P[j]$	a	b	a	c	a	b
$F(j)$	0	0	1			

$F(j)$ er definert som lengden av lengste prefikset i $P[0..j]$ som også er suffiks i $P[1..j]$

$P[0..3]$ = abac
$P[1..3]$ = bac

Altså lengste prefiks i ‘abac’ som også er suffiks i ‘bac’

j	0	1	2	3	4	5
$P[j]$	a	b	a	c	a	b
$F(j)$	0	0	1	0		

$F(j)$ er definert som lengden av lengste prefikset i $P[0..j]$ som også er suffiks i $P[1..j]$

$P[0..3]$ = abac
$P[1..3]$ = bac

Altså lengste prefiks i ‘abac’ som også er suffiks i ‘bac’

j	0	1	2	3	4	5
$P[j]$	a	b	a	c	a	b
$F(j)$	0	0	1	0		

$F(j)$ er definert som lengden av lengste prefikset i $P[0..j]$ som også er suffiks i $P[1..j]$

$P[0..4]$ = abaca
$P[1..4]$ = baca

Altså lengste prefiks i 'abaca' som også er suffiks i 'baca'

j	0	1	2	3	4	5
$P[j]$	*a*	*b*	*a*	*c*	*a*	*b*
$F(j)$	0	0	1	0		

$F(j)$ er definert som lengden av lengste prefikset i $P[0..j]$ som også er suffiks i $P[1..j]$

$P[0..4]$ = abaca
$P[1..4]$ = baca

Altså lengste prefiks i 'abaca' som også er suffiks i 'baca'

j	0	1	2	3	4	5
$P[j]$	***a***	***b***	***a***	***c***	***a***	***b***
$F(j)$	0	0	1	0	1	

$F(j)$ er definert som lengden av lengste prefikset i $P[0..j]$ som også er suffiks i $P[1..j]$

$P[0..4]$ = abaca
$P[1..4]$ = baca

Altså lengste prefiks i 'abaca' som også er suffiks i 'baca'

j	0	1	2	3	4	5
$P[j]$	***a***	***b***	***a***	***c***	***a***	***b***
$F(j)$	0	0	1	0	1	

$F(j)$ er definert som lengden av lengste prefikset i $P[0..j]$ som også er suffiks i $P[1..j]$

$P[0..5]$ = abacab
$P[1..4]$ = bacab

Altså lengste prefiks i 'abacab' som også er suffiks i 'bacab'

j	0	1	2	3	4	5
$P[j]$	a	b	a	c	a	b
$F(j)$	0	0	1	0	1	

$F(j)$ er definert som lengden av lengste prefikset i $P[0..j]$ som også er suffiks i $P[1..j]$

$P[0..5]$ = abacab
$P[1..4]$ = bacab

Altså lengste prefiks i 'abacab' som også er suffiks i 'bacab'

j	0	1	2	3	4	5
$P[j]$	***a***	***b***	***a***	***c***	***a***	***b***
$F(j)$	0	0	1	0	1	2

$F(j)$ er definert som lengden av lengste prefikset i $P[0..j]$ som også er suffiks i $P[1..j]$

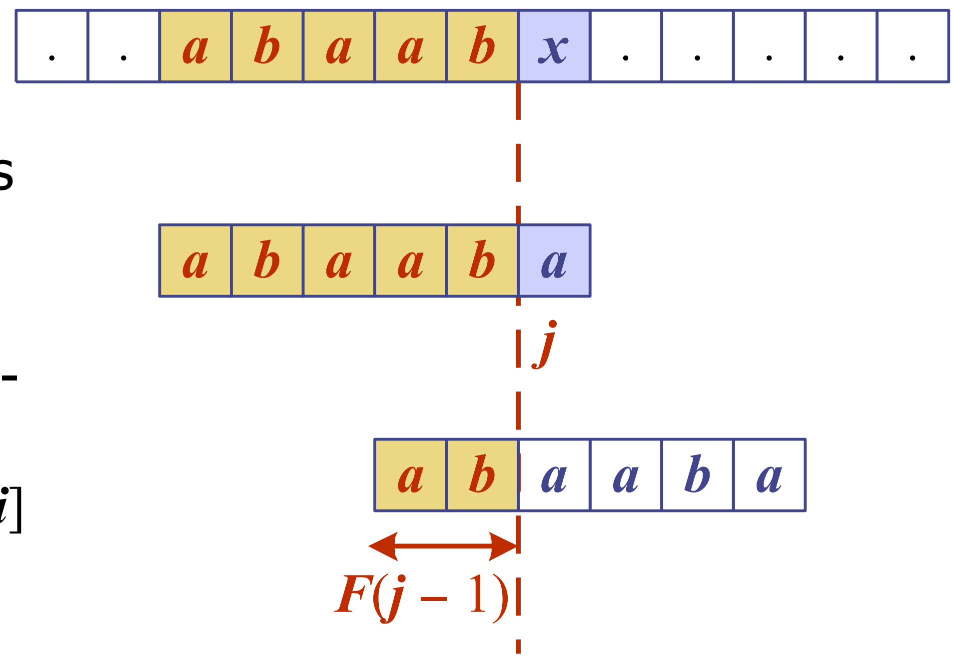
$P[0..5]$ = abacab
$P[1..4]$ = bacab

Altså lengste prefiks i 'abacab' som også er suffiks i 'bacab'

KMP Failure Function

- Knuth-Morris-Pratt's algorithm preprocesses the pattern to find matches of prefixes of the pattern with the pattern itself
- The failure function $F(j)$ is defined as the size of the largest prefix of $P[0..j]$ that is also a suffix of $P[1..j]$
- Knuth-Morris-Pratt's algorithm modifies the brute-force algorithm so that if a mismatch occurs at $P[j] \neq T[i]$ we set $j \leftarrow F(j-1)$

j	0	1	2	3	4	5
$P[j]$	a	b	a	a	b	a
$F(j)$	0	0	1	1	2	3



The KMP Algorithm

- The failure function can be represented by an array and can be computed in $O(m)$ time
- At each iteration of the while-loop, either
  - i increases by one, or
  - the shift amount $i - j$ increases by at least one (observe that $F(j - 1) < j$)
- Hence, there are no more than $2n$ iterations of the while-loop
- Thus, KMP's algorithm runs in optimal time $O(m + n)$

```
Algorithm KMPMatch(T, P)
    F ← failureFunction(P)
    i ← 0
    j ← 0
    while i < n
        if T[i] = P[j]
            if j = m − 1
                return  i − j { match }
            else
                i ← i + 1
                j ← j + 1
        else
            if j > 0
                j ← F[j − 1]
            else
                i ← i + 1
    return −1 { no match }
```