

datastructuren en algoritmen  
2010 10 18 (slides van 2010-2011 dus vorig jaar!)  
college 12

## schema

- brute-force pattern matching
- Boyer-Moore pattern matching
- Knuth-Morris-Pratt pattern matching
- preprocessing de tekst
- Huffman codering
- extra materiaal

## schema

- brute-force pattern matching
- Boyer-Moore pattern matching
- Knuth-Morris-Pratt pattern matching
- preprocessing de tekst
- Huffman codering
- extra materiaal

## string

- **string**: rijtje karakters ter lengte  $m$   
voorbeeld:  $S = acaabca$
- **substring**: string van de vorm  $S[i \dots j]$   
voorbeeld:  $abc$  is een substring van  $S$
- **prefix**: substring  $S[0 \dots j]$  die begint bij het begin  
voorbeeld:  $aca$  is een prefix van  $S$
- **suffix**: substring  $S[i \dots m]$  die eindigt bij het eind  
voorbeeld:  $ca$  is een suffix van  $S$

## string/pattern matching

- **probleem:**  
gegeven tekst  $T$  en patroon  $P$   
vind substring  $P$  van  $T$
- **toepassingen:**  
text editors, search engines, bioinformatica
- **voorbeeld**  
zoek *bra* in *adacadabra*

## brute-force algoritme

- **voorbeeld worst-case:**  
 $T = \text{aaaaa} \dots \text{aaaaab}$  en  $P = \text{aaab}$
- **worst-case tijdscomplexiteit:**  
buitenste loop  $n - m + 1$  keer, binnenste  $m$  keer  
 $\mathcal{O}(n \cdot m)$   
(lengte tekst keer lengte patroon)

## brute-force algoritme

loop van links naar rechts met  $P$  van lengte  $m$  door  $T$  van lengte  $n$

bruteForceMatch( $T, P$ ):

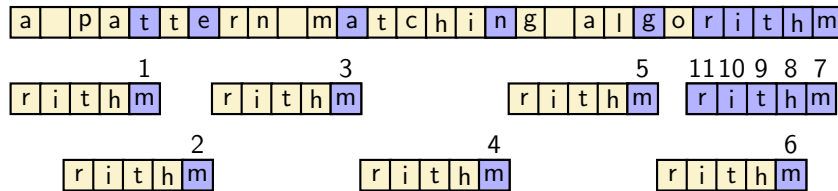
```
for pos = 0 to n - m do
  match = true
  for i = 0 to m - 1 do
    if P[i] != T[pos + i] then
      match = false
      break the for-i-loop
  if match then return pos (match at position pos)
return -1 (no match)
```

## schema

- brute-force pattern matching
- Boyer-Moore pattern matching
- Knuth-Morris-Pratt pattern matching
- preprocessing de tekst
- Huffman codering
- extra materiaal

## Boyer-Moore algoritme: idee

- vergelijk  $P$  met substring van  $T$  van achter naar voren
- character-jump als mismatch op  $T[i] = c$   
als  $c$  niet in  $P$  dan: schuif begin  $P$  naar  $T[i + 1]$   
als  $c$  wel in  $P$  dan: schuif laatste  $c$  in  $P$  naar  $T[i]$



## last-occurrence functie

- vaak gegeven als array;  
voorbeeld met  $\Sigma = \{a, b, c, d\}$  en  $P = \text{"abacab"}$

c	a	b	c	d
$L(c)$	4	5	3	-1

- geeft voor patroon per letter de index van laatste voorkomen met  $-1$  als helemaal geen voorkomen
- worst-case tijdscomplexiteit:  
 $\mathcal{O}(m + s)$   
met  $m$  de lengte van patroon  $P$  en  $s$  de grootte van alfabet

## Boyer-Moore algoritme

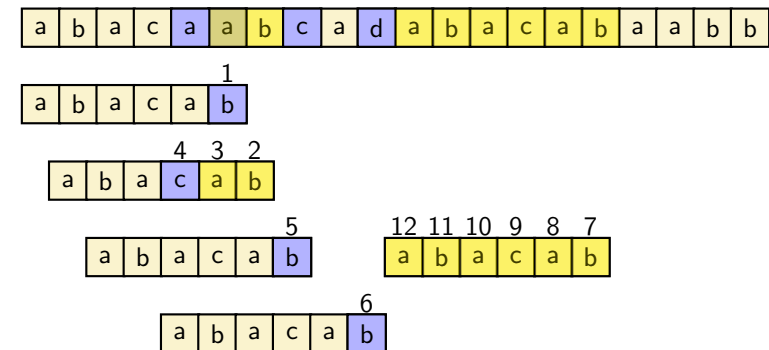
```

BoyerMooreMatch( $T, P, \Sigma$ ):
   $L = \text{lastOccurrenceFunction}(P, \Sigma)$ 
   $pos = 0$ 
  while  $pos \leq n - m$  do
    match = true
    for  $i = m - 1$  downto 0 do
      if  $P[i] \neq T[pos + i]$  then
        match = false
        (align last occurrence, but move at least 1)
         $pos = pos + \max(1, i - L(T[pos + i]))$ 
        break the for- $i$ -loop
    done
    if match then return  $pos$  (match at position  $pos$ )
  done
  return  $-1$  (no match)
  
```

## Boyer-Moore algoritme: voorbeeld

- $\Sigma = \{a, b, c, d\}$
- $P = \text{"abacab"}$

c	a	b	c	d
$L(c)$	4	5	3	-1



## Boyer-Moore algoritme: tijdscomplexiteit

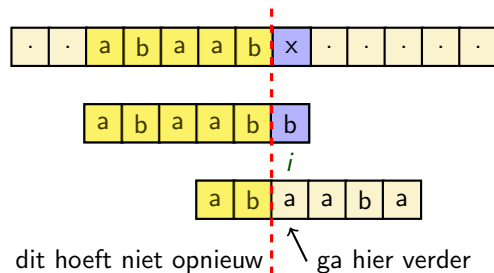
- **voorbeeld worst-case:**  
 $T = aaa \dots aaa$  en  $P = ba$   
 niet waarschijnlijk in Engelse tekst;  
 kan best in bio-toepassingen
- **worst-case tijdscomplexiteit:**  
 $O(m + s) + O(n \cdot m) = O(n \cdot m + s)$   
 met  $n$  lengte  $T$ , en  $m$  lengte  $P$ , en  $s$  grootte alfabet

## schema

- brute-force pattern matching
- Boyer-Moore pattern matching
- Knuth-Morris-Pratt pattern matching
- preprocessing de tekst
- Huffman codering
- extra materiaal

## Knuth-Morris-Pratt algoritme: idee

- vergelijk patroon  $P$  met tekst  $T$  van links naar rechts
- als mismatch  $T[pos] \neq P[i]$ : wat is de maximale opschuif?  
 het grootste prefix  $P[0 \dots i]$  dat is suffix  $P[1 \dots i-1]$   
 oftewel:  $f(i)$  geeft het aantal karakters dat je kunt hergebruiken bij mismatch op  $P[i+1]$

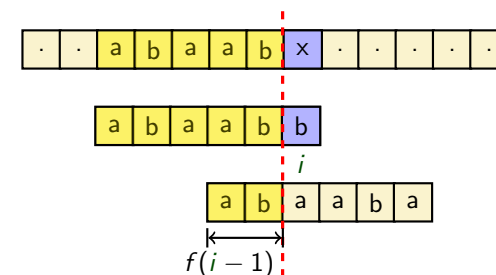


## KMP algoritme: failure functie

- bereken  $f(i)$  voor elke positie in patroon  $P$
- $f(i)$ : grootste  $j$  zdd  $P[0 \dots j]$  is suffix van  $P[1 \dots i]$

$P = \text{"abaaba"}$

$i$	0	1	2	3	4	5
$P[i]$	a	b	a	a	b	a
$f(i)$	0	0	1	1	2	3



## Knuth-Morris-Pratt algoritme

KMPMatch( $T, P$ ):

$f = \text{failureFunction}(P)$

$pos = 0$

$i = 0$

**while**  $pos \leq n - m$  **do**

**if**  $P[i] \neq T[pos + i]$  **then**

**if**  $i > 0$  **then**

$pos = pos + i - f(i - 1)$

**else**

$pos = pos + 1$

$i = f(i - 1)$

**else**

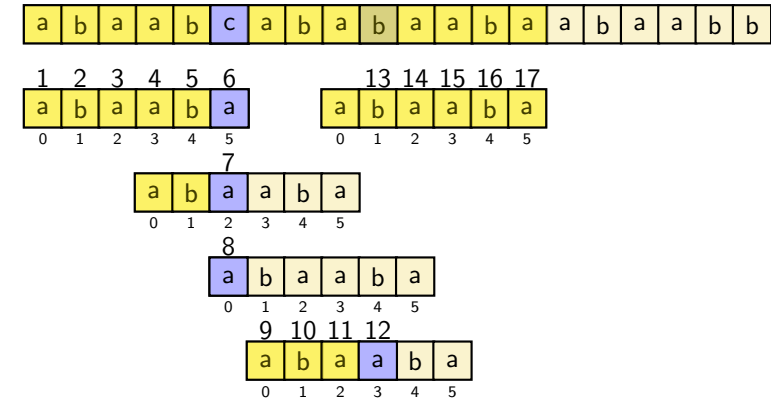
$i = i + 1$

**if**  $i == m$  **then return**  $pos$  (match at position  $pos$ )

**done**

**return**  $-1$  (no match)

## KMP algoritme: voorbeeld



$i$	0	1	2	3	4	5
$f(i)$	0	0	1	1	2	3

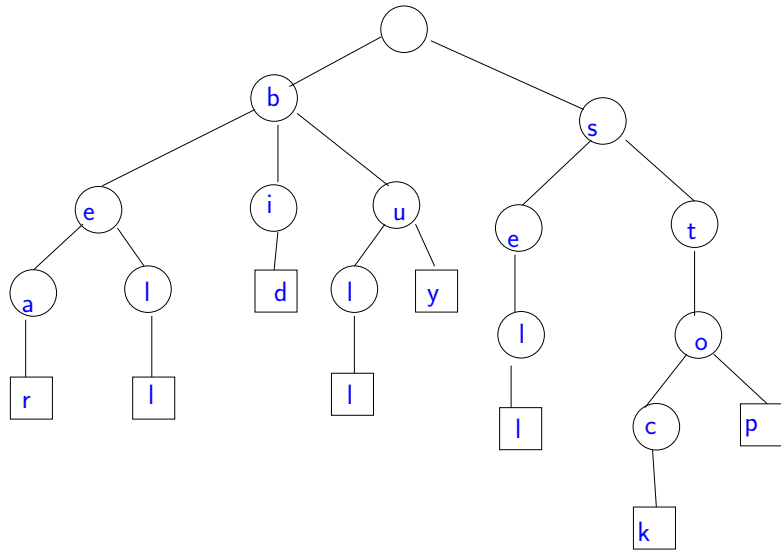
## Knuth-Morris-Pratt: tijdscomplexiteit

- berekening failure-functie:  
in  $\mathcal{O}(m)$  met  $m$  lengte patroon  $P$
- in iedere iteratie while-loop:  
of  $i$  wordt 1 groter  
of  $pos$  wordt ten minste zoveel groter als  $i$  kleiner wordt
- while-loop ten hoogste  $2n$  keer
- algoritme in  $\mathcal{O}(n + m)$

## schema

- brute-force pattern matching
- Boyer-Moore pattern matching
- Knuth-Morris-Pratt pattern matching
- preprocessing de tekst
- Huffman codering
- extra materiaal

## standard trie: voorbeeld



## standard trie

- boom om strings in op te slaan
- een string is niet de prefix van een andere string
- wortel leeg
- 1 karakter per knoop

## standard trie: eigenschappen

$d$  grootte alfabet,  $s$  strings,  $n$  totale lengte

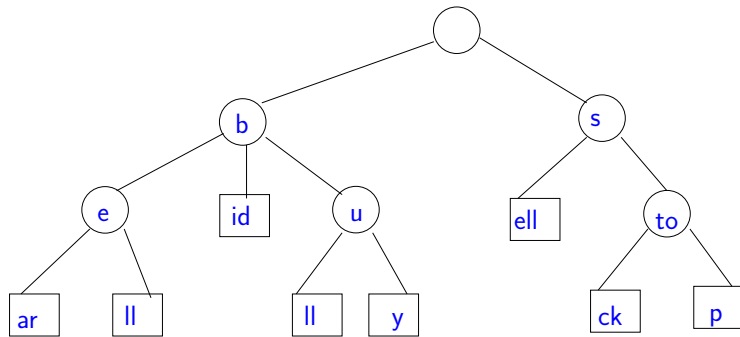
- interne knoop heeft ten hoogste  $d$  opvolgers
- voor elke string precies 1 externe knoop
- hoogte = lengte langste string
- aantal knopen in  $\mathcal{O}(n)$

## standard tries: hoe duur?

- construeren van een standard trie:  
toevoegen string van lengte  $m$  in  $\mathcal{O}(dm)$   
construeren hele trie in  $\mathcal{O}(dn)$
- zoeken van string van lengte  $m$  in standard trie:  
bezoek ten hoogste  $m + 1$  knopen,  
per knoop ten hoogste  $d$  checks  
in  $\mathcal{O}(dm)$

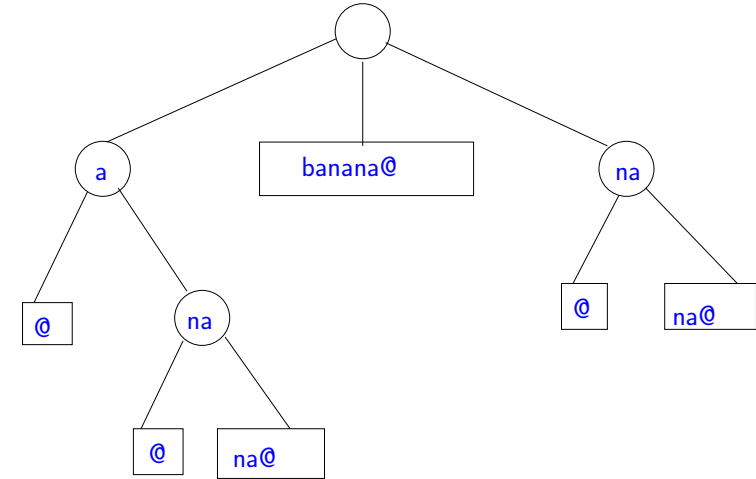
## gecomprimeerde trie: voorbeeld

vervang sliert naar blad door blad met groter label



## suffix tree: voorbeeld

voor snelle implementatie van string operaties



## schema

- brute-force pattern matching
- Boyer-Moore pattern matching
- Knuth-Morris-Pratt pattern matching
- preprocessing de tekst
- **Huffman codering**
- extra materiaal

## binaire codering voor karakters

- **fixed-length code:**  
elk karakter krijgt code (0-en en 1-en) in vaste lengte  
voorbeeld: ASCII en Unicode  
als  $c(A) = 00$ ,  $c(B) = 01$ ,  $c(C) = 11$   
dan  $ACBA = 00110100$
- **variable-length code:**  
elk karakter krijgt code maar mogelijk verschillende lengtes  
voorbeeld: Huffman codering  
als  $c(A) = 0$ ,  $c(B) = 10$ ,  $c(C) = 11$   
dan  $ACBA = 011100$

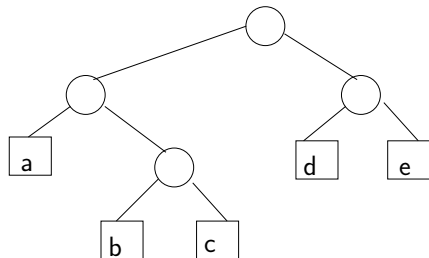
## codering: prefix-free

- voorbeeld ambigue codering:  
 $c(A) = 10$ ,  $c(B) = 01$ ,  $c(C) = 0$   
dan  $BC = 010$  en ook  $CA = 010$
- codering is prefix-free als:  
voor  $A \neq B$  geldt  $c(A)$  is geen prefix van  $c(B)$

## coderingsboom

- elk blad bevat karakter
- pad van wortel naar blad geeft codewoord
- met 0 voor links en 1 voor rechts

00	010	011	10	11
a	b	c	d	e



## Huffman codering: idee

- gebruikt voor data compressie
- variable-length code  
frequentere karakters krijgen kortere code
- binaire boom voor decodering

## Huffman codering

- gegeven: alfabet van karakters elk met frequentie
- gezocht: codering met minimale verwachte lengte

## Huffman's algoritme construeert coderingsboom

- maak van elk karakter een boom
- kies twee bomen met kleinste frequentie-som en voeg die samen
- ga door totdat 1 boom over is
- voorbeeld:  $X = abracadabra$

a	b	c	d	r
5	2	1	1	2

## schema

- brute-force pattern matching
- Boyer-Moore pattern matching
- Knuth-Morris-Pratt pattern matching
- preprocessing de tekst
- Huffman codering
- extra materiaal

## Huffman's algoritme

- greedy algoritme: kies steeds twee bomen met laagste key
- in  $\mathcal{O}(n + (d \log d))$   
 $n$  lengte van woord,  $d$  aantal verschillende karakters

## extra materiaal

- Boyer
- Moore
- Knuth
- Morris
- Pratt
- grep