Grundlæggende Algoritmer og Datastrukturer

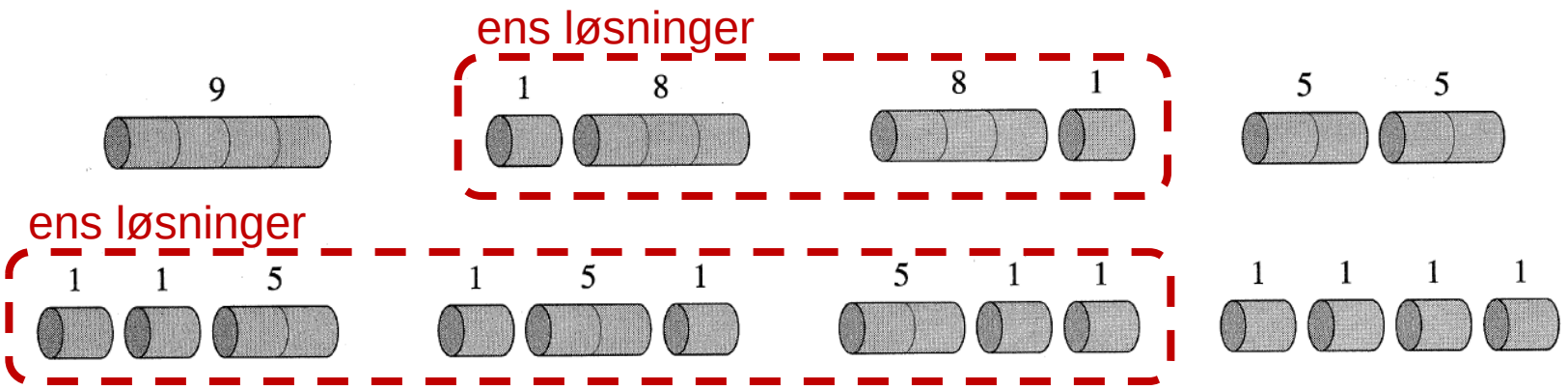
Dynamisk Programmering
[CLRS 15]

Dynamisk Programmering

- **Generel algoritmisk teknik** – virker for mange (men langt fra alle) problemer
- ***Krav***: ”Optimal delstruktur” – en løsning til problemet kan konstrueres ud fra optimale løsninger til ”delproblemer”
- **Rekursive løsning:**
  - Typisk eksponentiel tid
- **Dynamisk Programmering:**
  - Beregn delløsninger systematisk
  - Typisk polynomiel tid

Dynamisk Programmering: Optimal opdeling af en stang

Problem: Opdel en stang i dele, hvor hver del har en pris, således at den resulterende sum er maksimereret



En stang af længde 4 kan opdeles 8 forskellige måder – dog kun 5 forskellige resultater

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

Maksimal værdi for en opdeling af en stang af længde 12?

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

a) 12
b) 30
c) 32
d) 34
e) 35
f) Ved ikke

Optimal opdeling af stænger af længde 1..10

Bedste pris	Opdeling
$r_1 = 1$	$1 = 1$ (no cuts),
$r_2 = 5$	$2 = 2$ (no cuts),
$r_3 = 8$	$3 = 3$ (no cuts),
$r_4 = 10$	$4 = 2 + 2$,
$r_5 = 13$	$5 = 2 + 3$,
$r_6 = 17$	$6 = 6$ (no cuts),
$r_7 = 18$	$7 = 1 + 6$ or $7 = 2 + 2 + 3$,
$r_8 = 22$	$8 = 2 + 6$,
$r_9 = 25$	$9 = 3 + 6$,
$r_{10} = 30$	$10 = 10$ (no cuts).

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

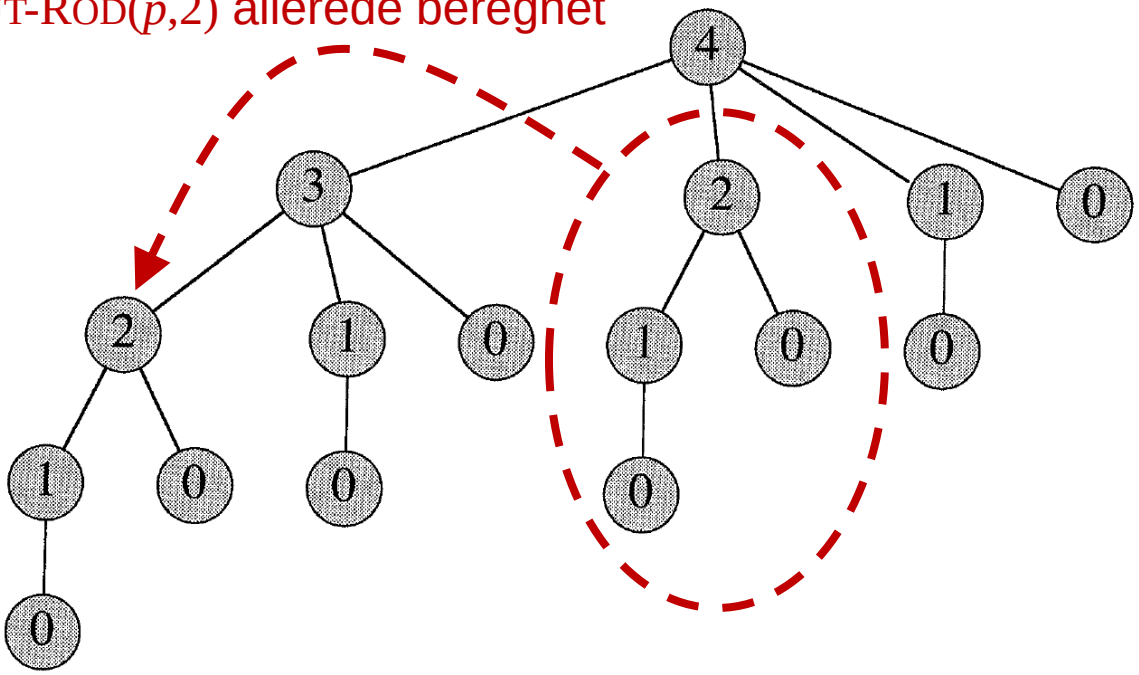
Optimal opdeling af en stang: Rekursiv løsning

$$r_n = \begin{cases} 0 & \text{hvis } n = 0 \\ \max_{i=1..n}(p_i + r_{n-i}) & \text{ellers} \end{cases}$$

```
CUT-ROD(p, n)
1  if n == 0
2      return 0
3  q = -∞
4  for i = 1 to n
5      q = max(q, p[i] + CUT-ROD(p, n - i))
6  return q
```

Optimal opdeling af en stang: Rekursiv løsning

CUT-ROD(p,2) allerede beregnet

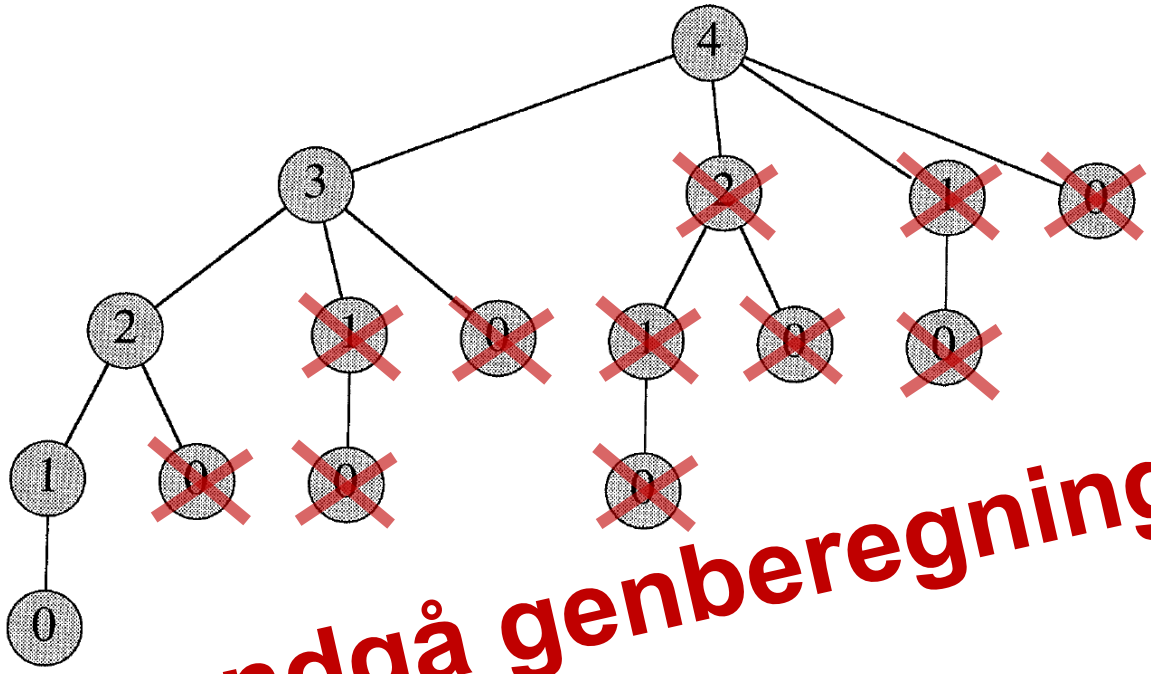



```
CUT-ROD(p, n)
1  if n == 0
2       return 0
3  q = -∞
4  for i = 1 to n
5       q = max(q, p[i] + CUT-ROD(p, n - i))
6  return q
```

$$r_n = \begin{cases} 0 & \text{hvis } n = 0 \\ \max_{i=1..n}(p_i + r_{n-i}) & \text{ellers} \end{cases}$$

Tid $O(2^n)$

Optimal opdeling af en stang: Rekursiv løsning



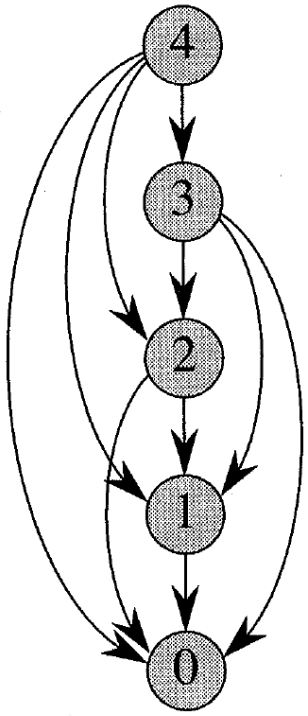
undgå genberegning ?

```
CUT-ROD(p, n)
1  if n == 0
2       return 0
3  q = -∞
4  for i = 1 to n
5       q = max(q, p[i] + CUT-ROD(p, n - i))
6  return q
```

$$r_n = \begin{cases} 0 & \text{hvis } n = 0 \\ \max_{i=1..n}(p_i + r_{n-i}) & \text{ellers} \end{cases}$$

Optimal opdeling af en stang: Rekursiv løsning + Memoization

```
MEMOIZED-CUT-ROD(p, n)
1  let r[0..n] be a new array
2  for i = 0 to n
3      r[i] = -∞
4  return MEMOIZED-CUT-ROD-AUX(p, n, r)

MEMOIZED-CUT-ROD-AUX(p, n, r)
1  if r[n] ≥ 0
2      return r[n]
3  if n == 0
4      q = 0
5  else q = -∞
6      for i = 1 to n
7          q = max(q, p[i] + MEMOIZED-CUT-ROD-AUX(p, n - i, r))
8  r[n] = q
9  return q
```



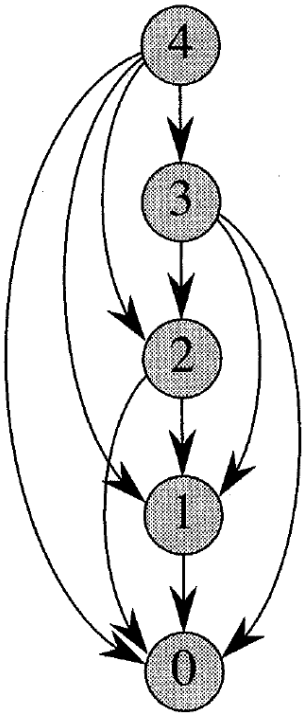
husk resultatet !

Tid $O(n^2)$

Optimal opdeling af en stang: Systematisk udfyldning

```
BOTTOM-UP-CUT-ROD(p, n)
1  let r[0..n] be a new array
2  r[0] = 0
3  for j = 1 to n
4      q = -∞
5      for i = 1 to j
6          q = max(q, p[i] + r[j - i])
7      r[j] = q
8  return r[n]
```

rækkefølgen vigtig !



Tid $O(n^2)$

Optimal opdeling af en stang: Udskrivning af løsningen

```
EXTENDED-BOTTOM-UP-CUT-ROD(p, n)
 1  let r[0..n] and s[0..n] be new arrays
 2  r[0] = 0
 3  for j = 1 to n
 4      q = -∞
 5      for i = 1 to j
 6          if q < p[i] + r[j - i]
 7              q = p[i] + r[j - i]
 8              s[j] = i
 9      r[j] = q
10  return r and s
```

i	0	1	2	3	4	5	6	7	8	9	10
$r[i]$	0	1	5	8	10	13	17	18	22	25	30
$s[i]$	0	1	2	3	2	2	6	1	2	3	10

```
PRINT-CUT-ROD-SOLUTION(p, n)
1  (r, s) = EXTENDED-BOTTOM-UP-CUT-ROD(p, n)
2  while n > 0
3      print s[n]
4      n = n - s[n]
```

Tid $O(n^2+n)$

Hvad udskrives der for $n = 7$?

```
PRINT-CUT-ROD-SOLUTION(p, n)
1  (r, s) = EXTENDED-BOTTOM-UP-CUT-ROD(p, n)
2  while n > 0
3      print s[n]
4      n = n − s[n]
```

i	0	1	2	3	4	5	6	7	8	9	10
$r[i]$	0	1	5	8	10	13	17	18	22	25	30
$s[i]$	0	1	2	3	2	2	6	1	2	3	10

a) 18 17

b) 17 18

c) 6 1

d) 1 6

e) 7 6

f) Ved ikke

Lad $X = x_1, \ldots, x_n$ og $W = w_1, \ldots, w_n$ være sekvenser af længde n, hvor der til hvert element x_i er knyttet en positiv vægt w_i.

I denne opgave ønsker vi at finde en voksende delsekvens af X som har maximal vægt. Det vil sige at vi ønsker at finde en delsekvens $i_1 < i_2 < \cdots < i_\ell$ af positionerne, hvor $x_{i_1} < x_{i_2} < \cdots < x_{i_\ell}$ og summen $w_{i_1} + w_{i_2} + \ldots + w_{i_\ell}$ er størst mulig.

i	1	2	3	4	5	6	7	8	9
x_i	6	8	7	$\underline{5}$	3	$\underline{9}$	4	$\underline{10}$	5
w_i	1	1	2	$\underline{4}$	1	$\underline{1}$	2	$\underline{4}$	3

Eksamensopgave 4, august 2008

Maximal vægt for en voksende delsekvens?

a) 1
b) 2
c) 3
d) 4
e) 5
f) 6
g) 7
h) 8
i) 9
j) Ved ikke

Vi lader $MEH(i)$ betegne *den maksimale vægt af en voksende delsekvens, hvor x_i er det sidste element i delsekvensen.*

$$MEH(i) = \begin{cases} w_i + \max\{MEH(j) \mid 1 \leq j < i \wedge x_j < x_i\} & \text{hvis der findes } 1 \leq j < i \wedge x_j < x_i \\ w_i & \text{ellers} \end{cases}$$

i	1	2	3	4	5	6	7	8	9
$MEH(i)$	**1**	**2**	**3**	**4**	**1**	**?**			

i	1	2	3	4	5	6	7	8	9
x_i	6	8	7	5	3	9	4	10	5
w_i	1	1	2	4	1	1	2	4	3

Eksamensopgave 4, august 2008

MEH(6) ?

(svar til spørgsmål b)

```
msf = −∞
for i = 1 to n
  m = 0
  for j = 1 to i − 1
    if x_j < x_i and MEH(j) > m then m = MEH(j)
  MEH(j) = m + w_i
  if MEH(j) > msf then msf = MEH(j)
return msf
```

a) 1
b) 2
c) 3
d) 4
e) 5
f) 6
g) 7
h) 8
i) 9
j) Ved ikke

Længste Fælles Delsekvens

Længste Fælles Delsekvens ?

ABABGACBABAD

BACABAABABC

a) BACBABA
b) BABBAB
c) BAABABA
d) BACABAB
e) Ved ikke

Længste Fælles Delsekvens

$$Y = \quad B\ D\ C\ A\ B\ A \qquad (j)$$

$$X = A\ B\ C\ B\ D\ A\ B \qquad (i)$$

$$c[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0, \\ c[i-1, j-1] + 1 & \text{if } i, j > 0 \text{ and } x_i = y_j, \\ \max(c[i, j-1], c[i-1, j]) & \text{if } i, j > 0 \text{ and } x_i \neq y_j. \end{cases}$$

længden af en længste fælles delsekvens af $x_1 x_2 \cdots x_i$ og $y_1 y_2 \cdots y_j$

c[5,3] ?

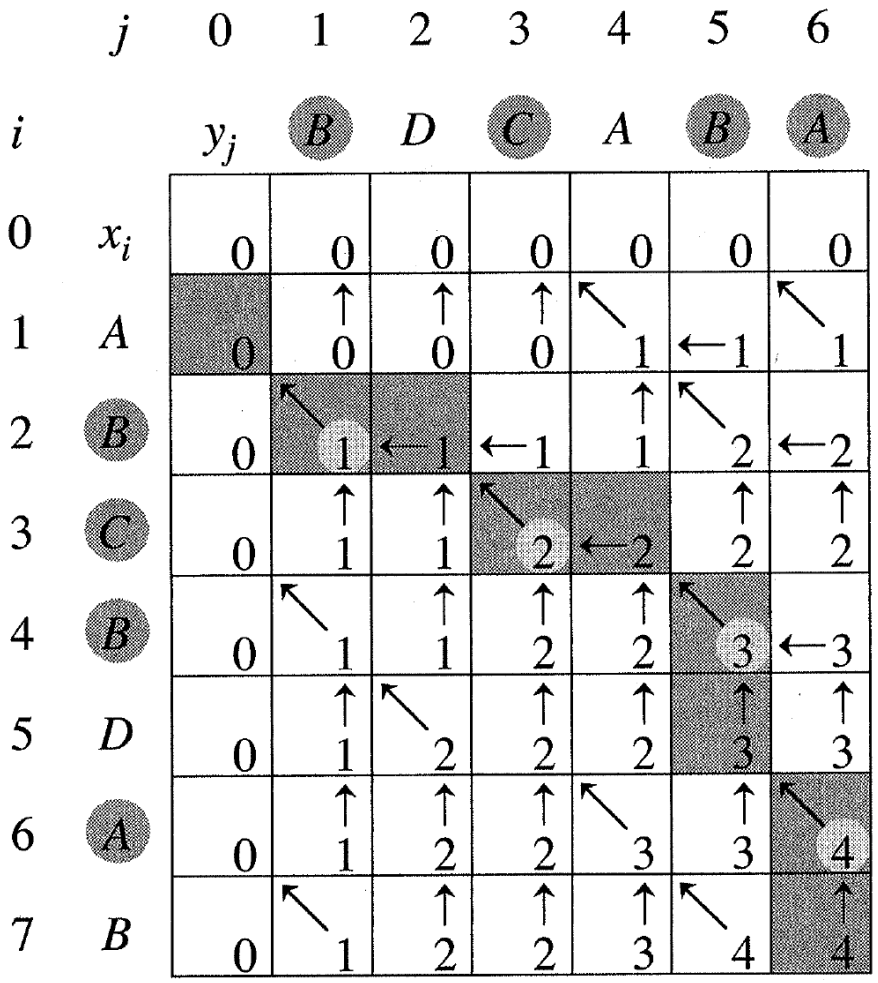
Y = ABABGACBABAD

X = BACABAABABC

a) 1
b) 2
c) 3
d) 4
e) 5
f) 6
g) 7
h) ved ikke

$$c[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0, \\ c[i-1, j-1] + 1 & \text{if } i, j > 0 \text{ and } x_i = y_j, \\ \max(c[i, j-1], c[i-1, j]) & \text{if } i, j > 0 \text{ and } x_i \neq y_j. \end{cases}$$

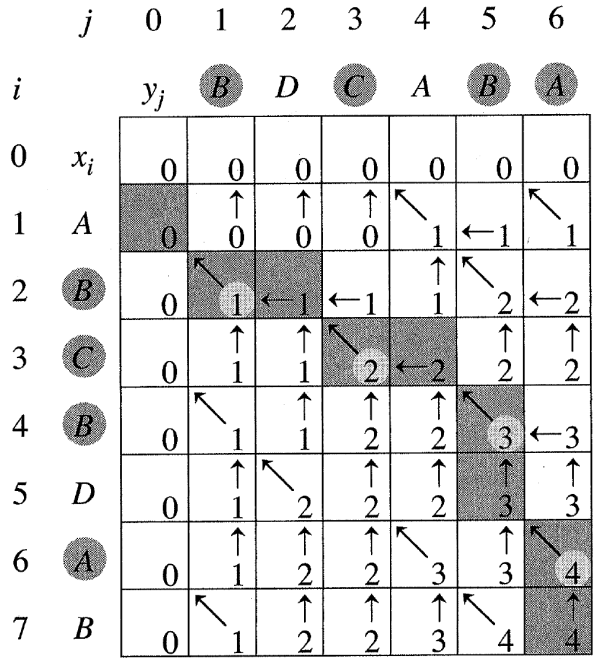
Længste Fælles Delsekvens



$$c[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0, \\ c[i-1, j-1] + 1 \nwarrow & \text{if } i, j > 0 \text{ and } x_i = y_j, \\ \max(c[i, j-1] \leftarrow, c[i-1, j] \uparrow) & \text{if } i, j > 0 \text{ and } x_i \neq y_j. \end{cases}$$

Længste Fælles Delsekvens

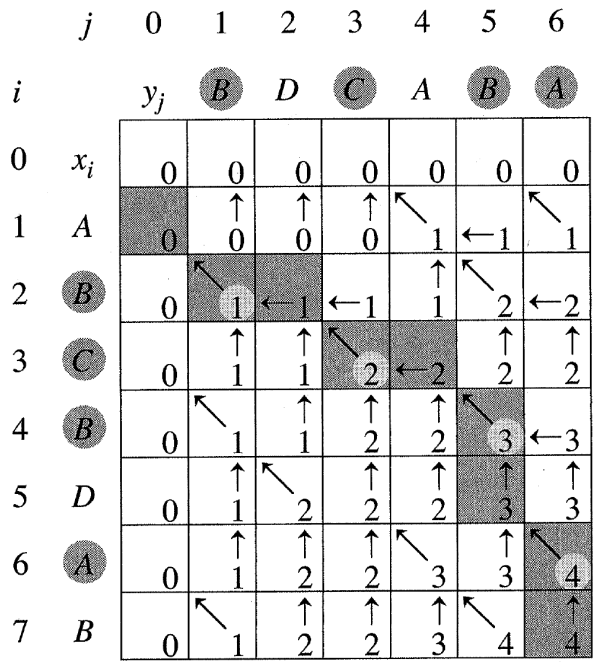
```
LCS-LENGTH(X, Y)
 1  m = X.length
 2  n = Y.length
 3  let b[1..m, 1..n] and c[0..m, 0..n] be new tables
 4  for i = 1 to m
 5      c[i, 0] = 0
 6  for j = 0 to n
 7      c[0, j] = 0
 8  for i = 1 to m
 9      for j = 1 to n
10          if x_i == y_j
11              c[i, j] = c[i − 1, j − 1] + 1
12              b[i, j] = "↖"
13          elseif c[i − 1, j] ≥ c[i, j − 1]
14              c[i, j] = c[i − 1, j]
15              b[i, j] = "↑"
16          else c[i, j] = c[i, j − 1]
17              b[i, j] = "←"
18  return c and b
```



Tid O(nm)

Længste Fælles Delsekvens

```
PRINT-LCS(b, X, i, j)
1  if i == 0 or j == 0
2       return
3  if b[i, j] == "↖"
4       PRINT-LCS(b, X, i − 1, j − 1)
5       print x_i
6  elseif b[i, j] == "↑"
7       PRINT-LCS(b, X, i − 1, j)
8  else PRINT-LCS(b, X, i, j − 1)
```



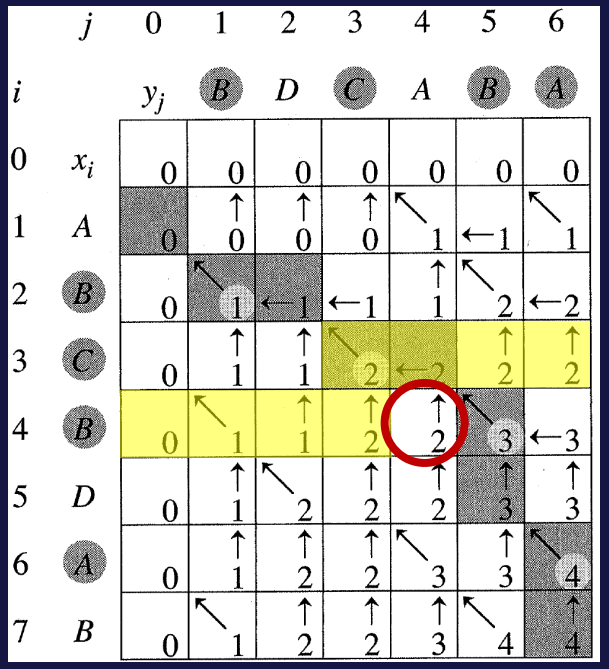
Tid $O(n+m)$

Pladsforbruget for LCS algoritmen til at finde en LCS ?

a) O($n + m$)

b) O($n \cdot m$)

c) Ved ikke



$m = |X|$ $n = |Y|$

Beregning af en LCS i O(*n* + *m*) plads

Idé

Beregn stien **uden at huske** matricen

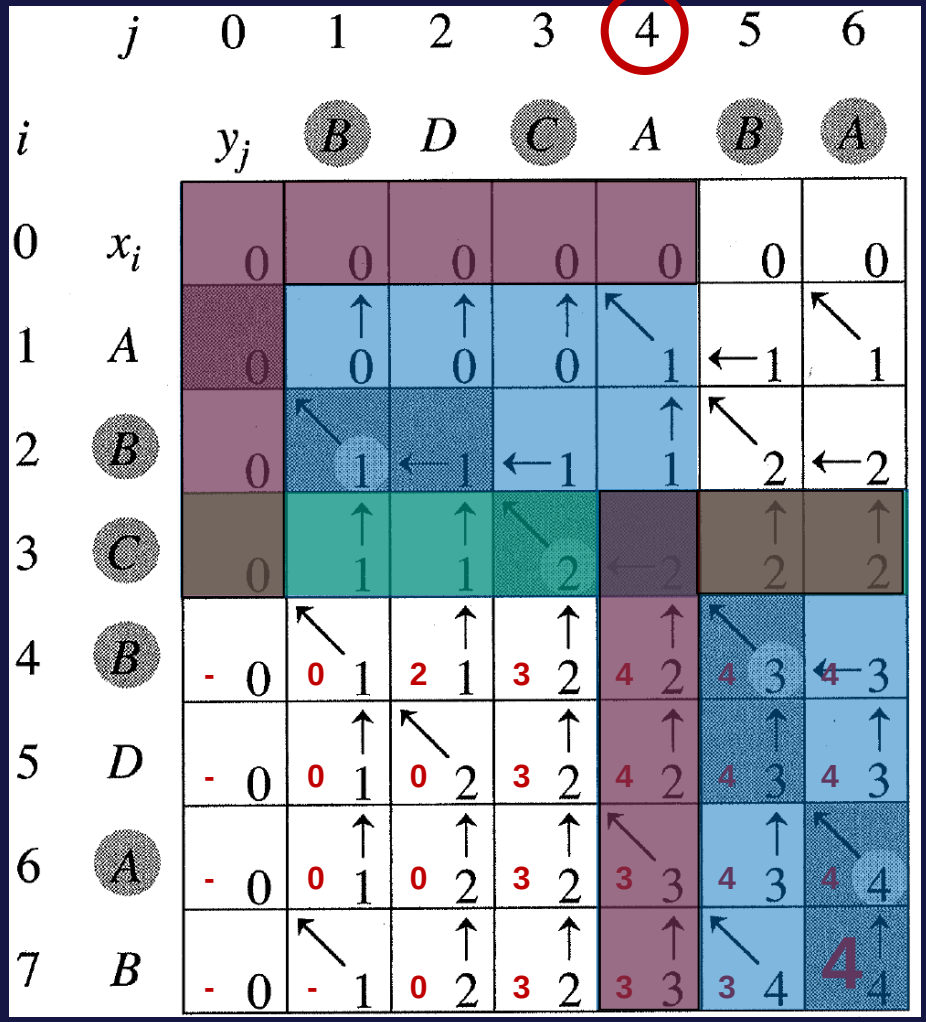
1) Beregn hvor stien krydser den midterste række

2) Beregn rekursivt de to del-stier

Tid O(*n*·*m*)

Plads O(*n*+*m*)

Eksempel på **del-og-kombiner** anvendt i en **dynamisk programmerings** algoritme



$m = |X|$ $n = |Y|$

Hirschberg 1975

Længste Fælles Delsekvens

$O(n \cdot m)$ tid	dynamisk programmering
$O(n+m)$ plads	dynamisk programmering og del og kombiner [Hirschberg 1975]

Åben problem

Kan man løse længste fælles delsekvens problemet for to strenge af længde n i tid $O(n^{2-\varepsilon})$?

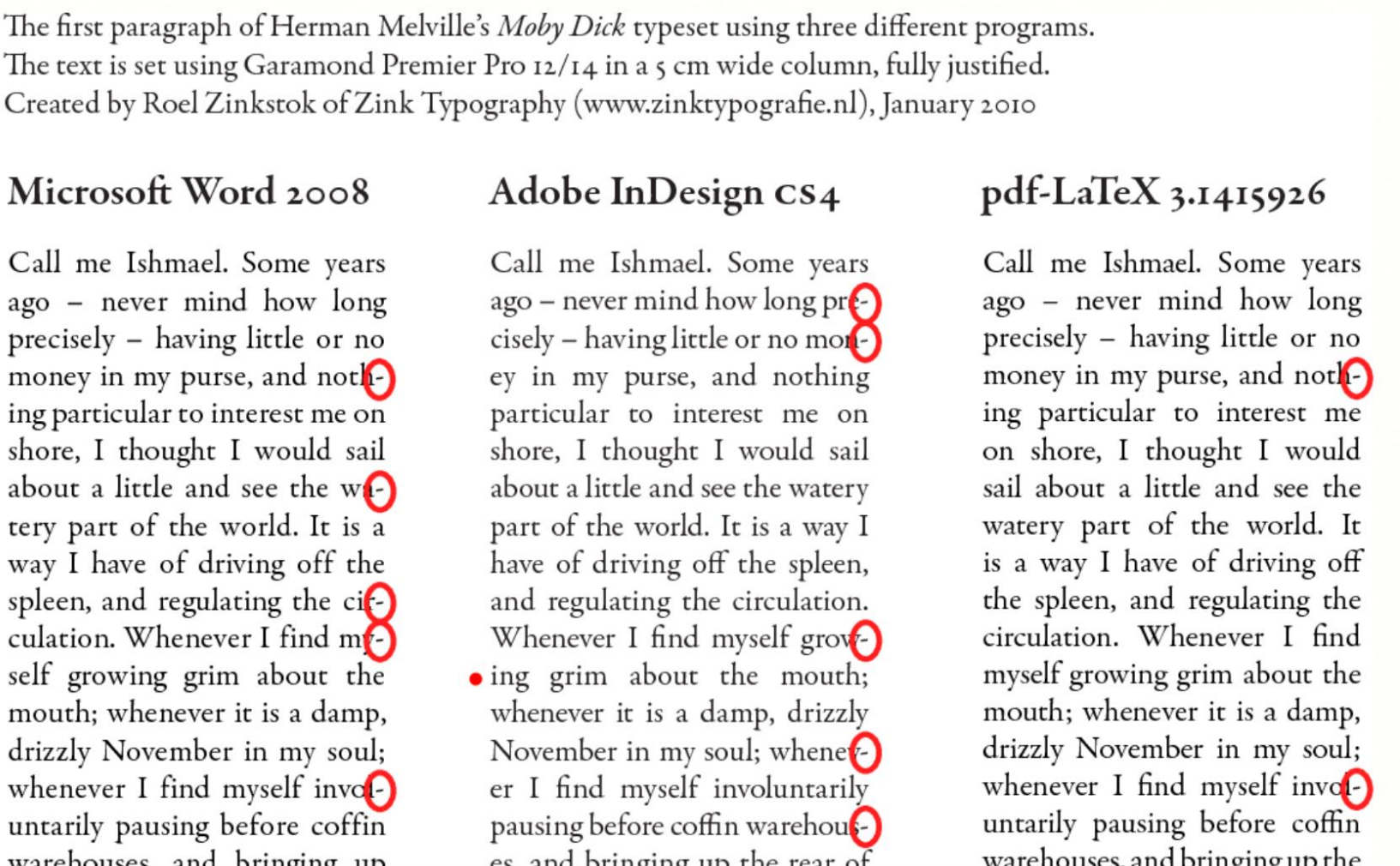
– Det ville bryde med den såkaldte Strong Exponential Time Hypothesis

Abboud, Backurs, Williams. Tight Hardness Results for LCS and Other Sequence Similarity Measures.
In *Proc. 56th Annual IEEE Symposium on Foundations of Computer Science (FOCS'15)*, pages 59–78, 2015.

Bringmann and Künnemann. Quadratic Conditional Lower Bounds for String Problems and Dynamic Time Warping.
In *Proc. 56th Annual IEEE Symposium on Foundations of Computer Science (FOCS'15)*, pages 79–97, 2015.

Tekstformatering - Linieskift

https://tex.stackexchange.com/questions/120271/alternatives-to-latex/120279#120279



The first paragraph of Herman Melville's *Moby Dick* typeset using three different programs.
The text is set using Garamond Premier Pro 12/14 in a 5 cm wide column, fully justified.
Created by Roel Zinkstok of Zink Typography (www.zinktypografie.nl), January 2010

Microsoft Word 2008

Call me Ishmael. Some years ago – never mind how long precisely – having little or no money in my purse, and noth-ing particular to interest me on shore, I thought I would sail about a little and see the wa-tery part of the world. It is a way I have of driving off the spleen, and regulating the cir-culation. Whenever I find my-self growing grim about the mouth; whenever it is a damp, drizzly November in my soul; whenever I find myself invol-untarily pausing before coffin

Adobe InDesign CS4

Call me Ishmael. Some years ago – never mind how long pre-cisely – having little or no mon-ey in my purse, and nothing particular to interest me on shore, I thought I would sail about a little and see the watery part of the world. It is a way I have of driving off the spleen, and regulating the circulation. Whenever I find myself grow-ing grim about the mouth; whenever it is a damp, drizzly November in my soul; whenev-er I find myself involuntarily pausing before coffin warehous-

pdf-LaTeX 3.1415926

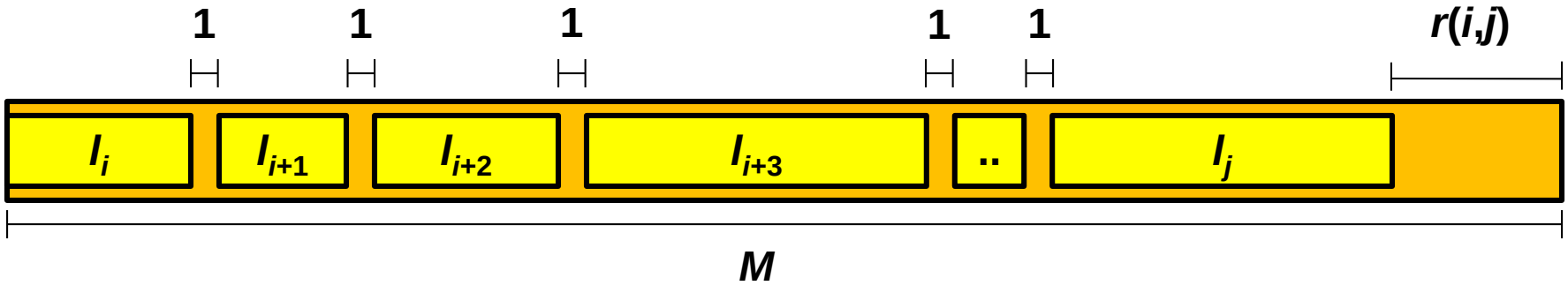
Call me Ishmael. Some years ago – never mind how long precisely – having little or no money in my purse, and noth-ing particular to interest me on shore, I thought I would sail about a little and see the watery part of the world. It is a way I have of driving off the spleen, and regulating the circulation. Whenever I find myself growing grim about the mouth; whenever it is a damp, drizzly November in my soul; whenever I find myself invol-untarily pausing before coffin

Problem 15-4 Printing neatly

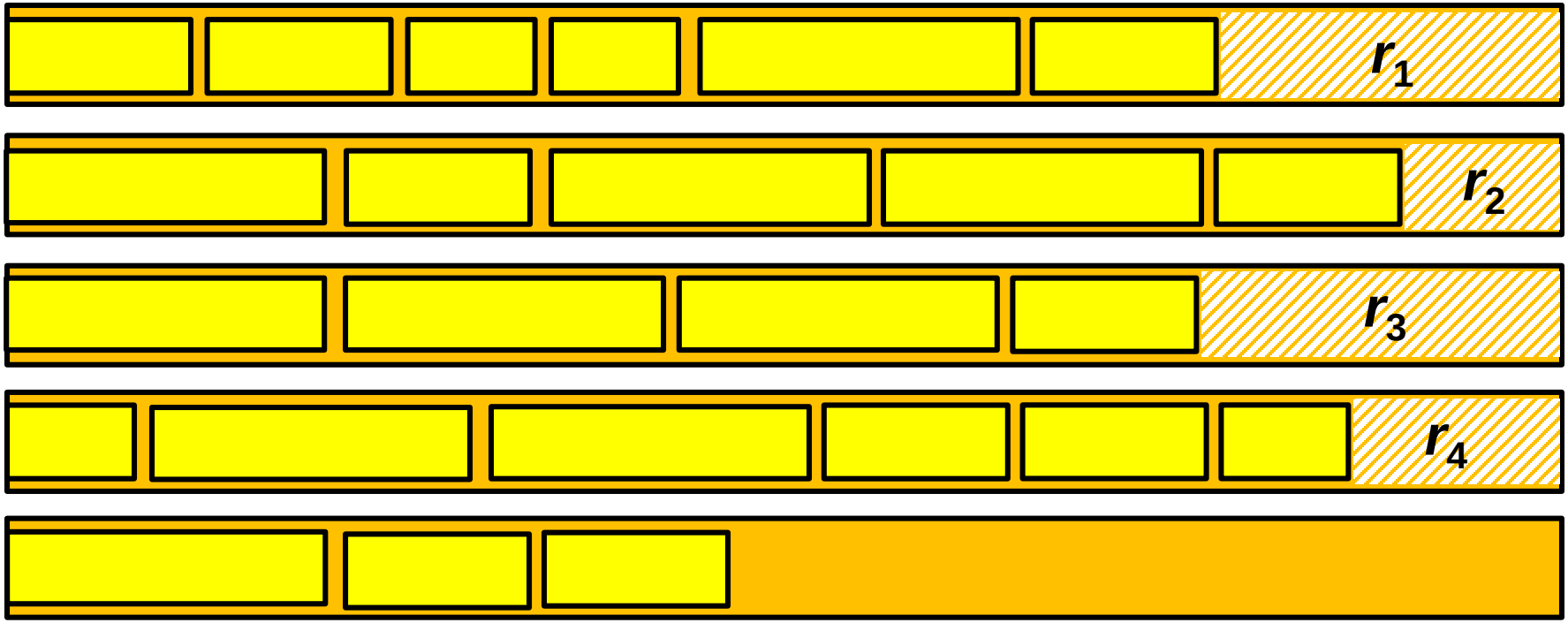
“Consider the problem of neatly printing a paragraph with a monospaced font (all characters having the same width) on a printer. The input text is a sequence of n words of lengths l_1, l_2,..., l_n, measured in characters. We want to print this paragraph neatly on a number of lines that hold a maximum of M characters each. Our criterion of “neatness” is as follows. If a given line contains words i through j, where $i \le j$, and we leave exactly one space between words, the number of extra space characters at the end of the line is

$$M - j + i - \sum_{k=i..j} l_k,$$

which must be nonnegative so that the words fit on the line. We wish to **minimize** the sum, over all lines except the last, of **the cubes of the numbers of extra space characters at the ends of lines**. Give a dynamic-programming algorithm to print a paragraph of n words neatly on a printer. Analyze the running time and space requirements of your algorithm.“



$$r(i,j) = M - j + i - \Sigma_{k=i..j}\, l_k$$



$$\text{pris} = r_1^3 + r_2^3 + r_3^3 + r_4^3$$

Problem 15-4 Printing neatly

Pris for at udskrive de første j ord på hele linier:

$$C[j] = \begin{cases} 0 & \text{hvis } j = 0 \\ \min\{C[i-1] + r(i,j)^3 \mid 1 \le i \le j \ \wedge \ r(i,j) \ge 0\} & \text{hvis } j > 0 \end{cases}$$

Bedste løsning er:

$$\min\{C[j] \mid 0 \le j < n \ \wedge \ r(j+1,n) \ge 0\}$$

ord j+1,j+2,...,n på sidste linie

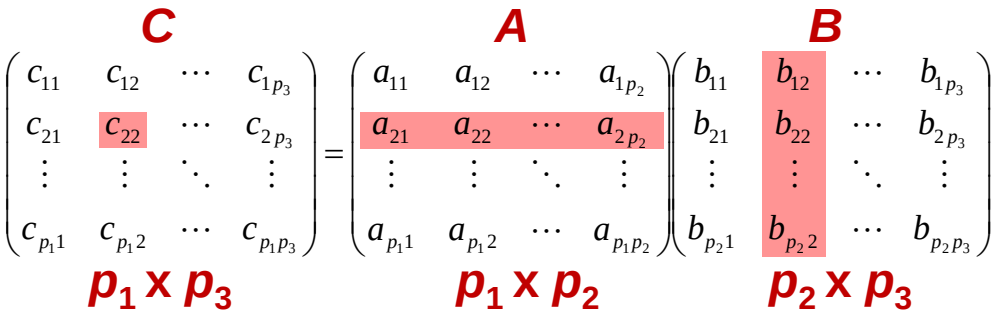
Printing neatly

```
 1 // beregn r
 2 for i=1 to n
 3   sum = 0
 4   for j=i to n
 5     sum = sum+l[j]
 6     r[i,j] = M-j+i-sum

 7 // beregn C
 8 C[0] = 0
 9 for j=1 to n
10  C[j] = +∞
11  for i = 1 to j
12    if r[i,j]>=0 and C[i-1]+r[i,j]^3<C[j] then
13      C[j] = C[i-1]+r[i,j]^3
14      I[j] = i
```

```
15 // udskriv bedste løsning
16 proc udskriv(j)
17   if j>0 then
18     i = I[j]
19     udskriv(i-1)
20     print ord[i]..ord[j] <newline>

21 k = n-1
22 for j=0 to n-2
23   if r[j+1,n]>=0 and C[j]<C[k] then
24     k = j
25 udskriv(k)
26 print ord[k+1]..ord[n] <newline>
```

Tid og plads $O(n^2)$

Matrix Multiplikation




```
MATRIX-MULTIPLY(A, B)
1  if A.columns ≠ B.rows
2      error "incompatible dimensions"
3  else let C be a new A.rows × B.columns matrix
4      for i = 1 to A.rows
5          for j = 1 to B.columns
6              c_ij = 0
7              for k = 1 to A.columns
8                  c_ij = c_ij + a_ik · b_kj
9      return C
```

Multiplikation af to matricer A og B af størrelse $p_1 \times p_2$ og $p_2 \times p_3$ tager tid $O(p_1 \cdot p_2 \cdot p_3)$

Matrix-kæde Multiplikation

$(A \cdot B) \cdot C$ eller $A \cdot (B \cdot C)$?

Matrix multiplikation er associativ (kan sætte paranteser som man vil) men ikke kommutative (kan ikke bytte rundt på rækkefølgen af matricerne)

Hvilken rækkefølge laver mindst (primitive) multiplikationer ?

a) $(A \cdot B) \cdot C$

b) $A \cdot (B \cdot C)$

c) Ved ikke

A **= 2 x 10** ***B*** **= 10 x 50** ***C*** **= 50 x 20**

Matrix-kæde Multiplikation

Problem: Find den bedste rækkefølge (paranteser) for at gange n matricer sammen

$$A_1 \cdot A_2 \cdots A_n$$

hvor A_i er en $p_{i\text{-}1}$ x p_i matrice

NB: Der er $\Omega(4^n/n^{3/2})$ mulige måder for paranteserne

Matrix-kæde Multiplikation

$m[i,j]$ = minimale antal (primitive) multiplikationer for at beregne $A_i \cdot \cdots \cdot A_j$

$(A_i \cdots A_k) \cdot (A_{k+1} \cdots A_j)$

$$m[i,j] = \begin{cases} 0 & \text{if } i = j\,, \\ \min\limits_{i \le k < j} \{m[i,k] + m[k+1,j] + p_{i-1} p_k p_j\} & \text{if } i < j\,. \end{cases}$$

```
RECURSIVE-MATRIX-CHAIN(p, i, j)
1  if i == j
2      return 0
3  m[i, j] = ∞
4  for k = i to j − 1
5      q = RECURSIVE-MATRIX-CHAIN(p, i, k)
           + RECURSIVE-MATRIX-CHAIN(p, k + 1, j)
           + p_{i−1} p_k p_j
6      if q < m[i, j]
7          m[i, j] = q
8  return m[i, j]
```

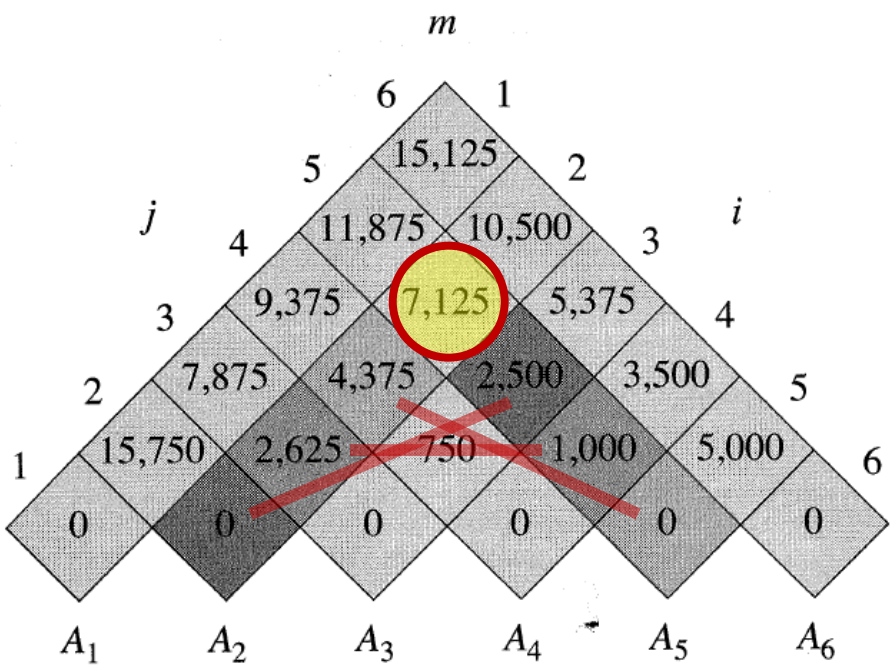
Tid $\Omega(4^n/n^{3/2})$

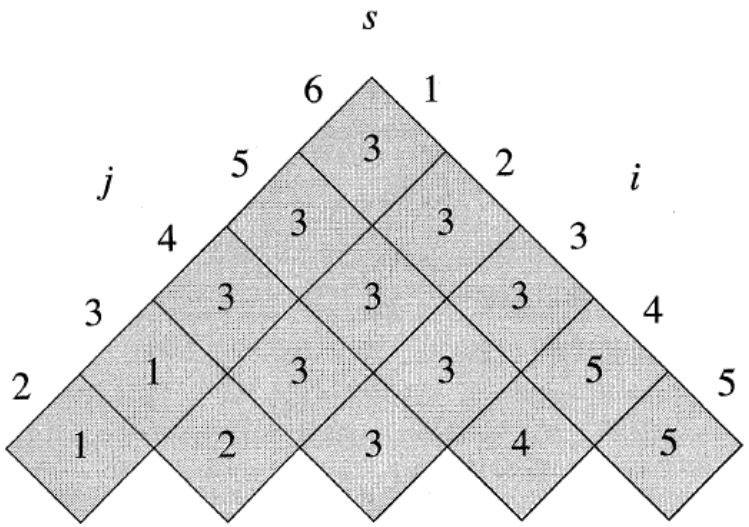
Matrix-kæde Multiplikation

```
MATRIX-CHAIN-ORDER(p)
 1  n = p.length − 1
 2  let m[1..n, 1..n] and s[1..n − 1, 2..n] be new tables
 3  for i = 1 to n
 4      m[i, i] = 0
 5  for l = 2 to n                    // l is the chain length
 6      for i = 1 to n − l + 1
 7          j = i + l − 1
 8          m[i, j] = ∞
 9          for k = i to j − 1
10              q = m[i, k] + m[k + 1, j] + p_{i−1} p_k p_j
11              if q < m[i, j]
12                  m[i, j] = q
13                  s[i, j] = k
14  return m and s
```

Tid $O(n^3)$

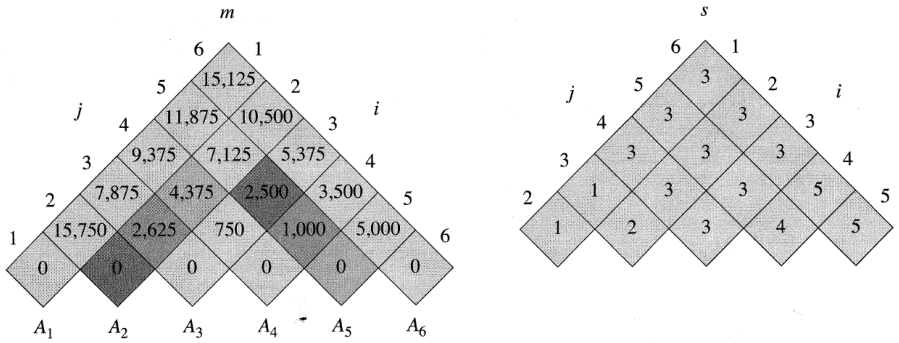
Matrix-kæde Multiplikation





matrix	A_1	A_2	A_3	A_4	A_5	A_6
dimension	30×35	35×15	15×5	5×10	10×20	20×25

Matrix-kæde Multiplikation

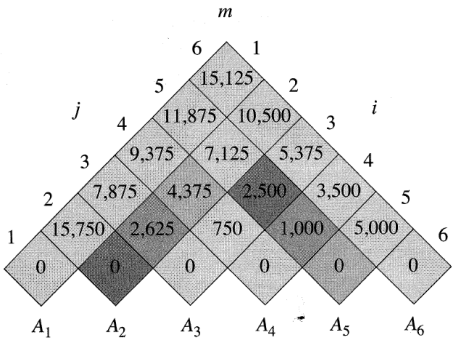


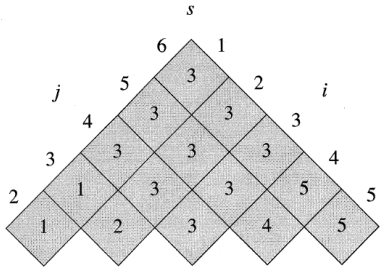

```
PRINT-OPTIMAL-PARENS(s, i, j)
1  if i == j
2      print "A"_i
3  else print "("
4      PRINT-OPTIMAL-PARENS(s, i, s[i, j])
5      PRINT-OPTIMAL-PARENS(s, s[i, j] + 1, j)
6      print ")"
```

Tid O(n)

Hvad udskrives der for $i = 1$ og $j = 6$?

```
PRINT-OPTIMAL-PARENS(s, i, j)
1  if i == j
2      print "A"_i
3  else print "("
4      PRINT-OPTIMAL-PARENS(s, i, s[i, j])
5      PRINT-OPTIMAL-PARENS(s, s[i, j] + 1, j)
6      print ")"
```





a) $((A_1A_2)(A_3(A_4A_5)A_6)))$

b) $((A_1((A_2A_3)((A_4A_5)A_6))))$

c) $((A_1(A_2A_3))((A_4A_5)A_6))$

d) $((A_1(A_2\,(A_3\,(A_4(A_5A_6))))))$

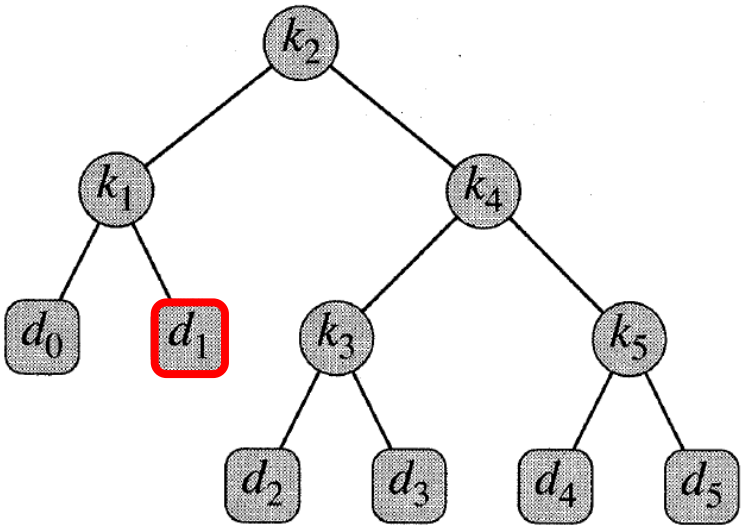
e) Ved ikke

"Memoized" Matrix-kæde Multiplikation

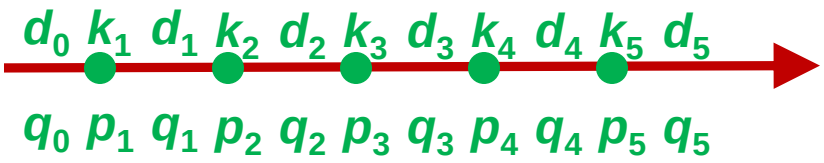
```
MEMOIZED-MATRIX-CHAIN(p)
1  n = p.length − 1
2  let m[1..n, 1..n] be a new table
3  for i = 1 to n
4      for j = i to n
5          m[i, j] = ∞
6  return LOOKUP-CHAIN(m, p, 1, n)

LOOKUP-CHAIN(m, p, i, j)
1  if m[i, j] < ∞
2      return m[i, j]
3  if i == j
4      m[i, j] = 0
5  else for k = i to j − 1
6          q = LOOKUP-CHAIN(m, p, i, k)
                + LOOKUP-CHAIN(m, p, k + 1, j) + p_{i−1} p_k p_j
7          if q < m[i, j]
8              m[i, j] = q
9  return m[i, j]
```

Tid $O(n^3)$

Optimale Binære Søgetræer



Forventet søgetid 2.80

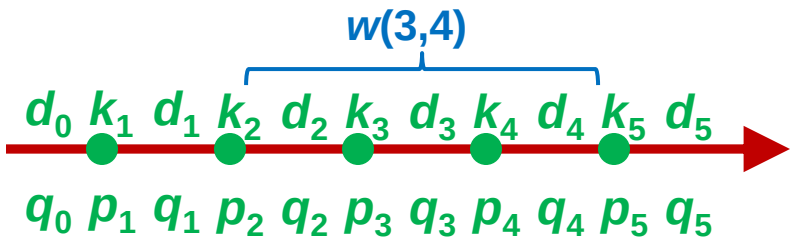


node	depth	probability	contribution
k_1	1	0.15	0.30
k_2	0	0.10	0.10
k_3	2	0.05	0.15
k_4	1	0.10	0.20
k_5	2	0.20	0.60
d_0	2	0.05	0.15
d_1	(2 + 1) x	0.10	= 0.30
d_2	3	0.05	0.20
d_3	3	0.05	0.20
d_4	3	0.05	0.20
d_5	3	0.10	0.40
Total			2.80

i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

Optimale Binære Søgetræer

$$\sum_{i=1}^{n} p_i + \sum_{i=0}^{n} q_i = 1 \qquad w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$



$$\mathrm{E}[\text{search cost in } T] = \sum_{i=1}^{n} (\mathrm{depth}_T(k_i) + 1) \cdot p_i + \sum_{i=0}^{n} (\mathrm{depth}_T(d_i) + 1) \cdot q_i$$

$$= 1 + \sum_{i=1}^{n} \mathrm{depth}_T(k_i) \cdot p_i + \sum_{i=0}^{n} \mathrm{depth}_T(d_i) \cdot q_i$$

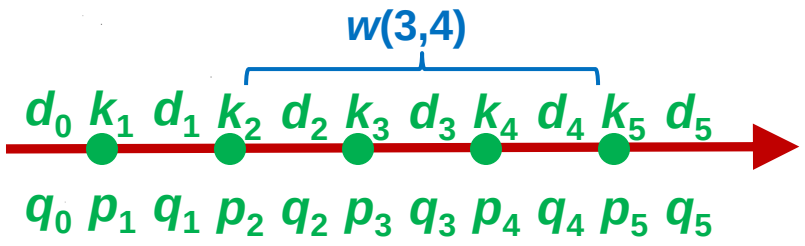
$$e[i, j] = \begin{cases} q_{i-1} & \text{if } j = i - 1, \\ \min\limits_{i \le r \le j} \{e[i, r-1] + e[r+1, j] + w(i, j)\} & \text{if } i \le j. \end{cases}$$

forventet optimal tid for et søgetræ indeholdende k_i...,k_j

Optimale Binære Søgetræer

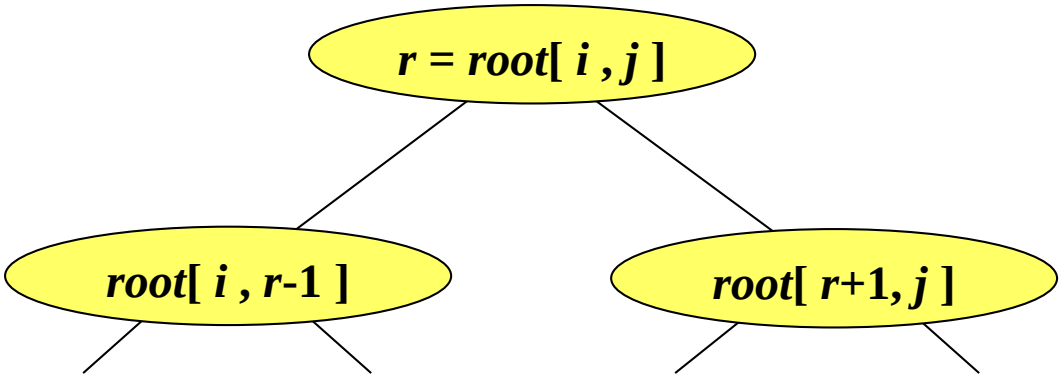
$$e[i,j] = \begin{cases} q_{i-1} & \text{if } j = i-1, \\ \min\limits_{i \le r \le j} \{e[i,r-1] + e[r+1,j] + w(i,j)\} & \text{if } i \le j. \end{cases}$$

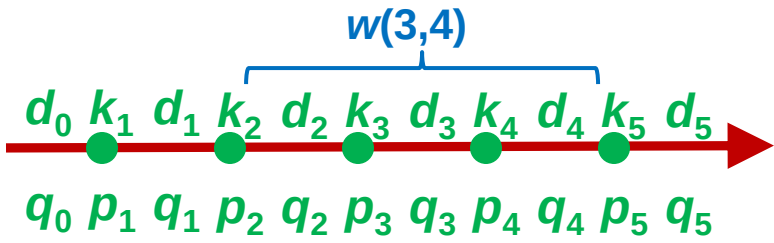
```
OPTIMAL-BST(p, q, n)
 1  let e[1..n+1, 0..n], w[1..n+1, 0..n],
          and root[1..n, 1..n] be new tables
 2  for i = 1 to n + 1
 3      e[i, i-1] = q_{i-1}
 4      w[i, i-1] = q_{i-1}
 5  for l = 1 to n
 6      for i = 1 to n - l + 1
 7          j = i + l - 1
 8          e[i, j] = ∞
 9          w[i, j] = w[i, j-1] + p_j + q_j
10          for r = i to j
11              t = e[i, r-1] + e[r+1, j] + w[i, j]
12              if t < e[i, j]
13                  e[i, j] = t
14                  root[i, j] = r
15  return e and root
```

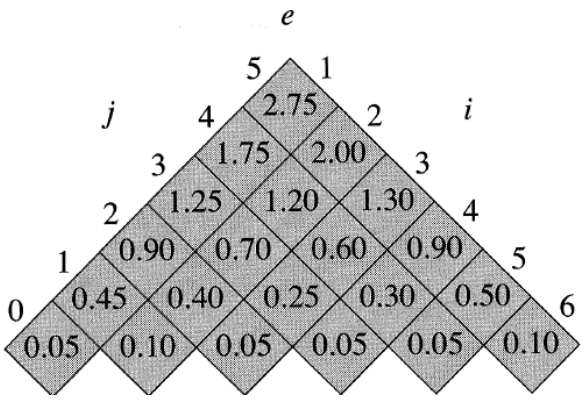


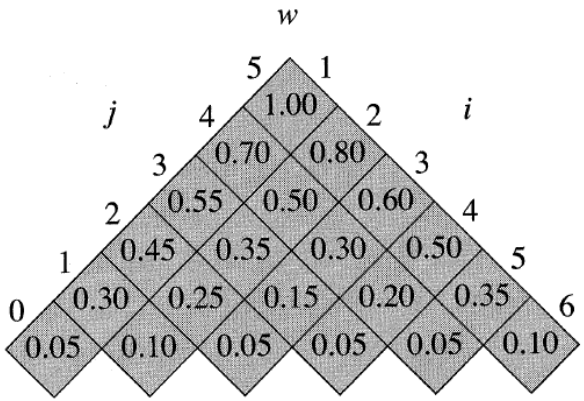
Tid $O(n^3)$

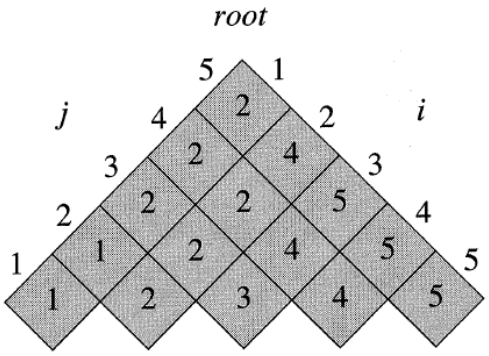
Konstruktion af Optimalt Binært Søgetræ





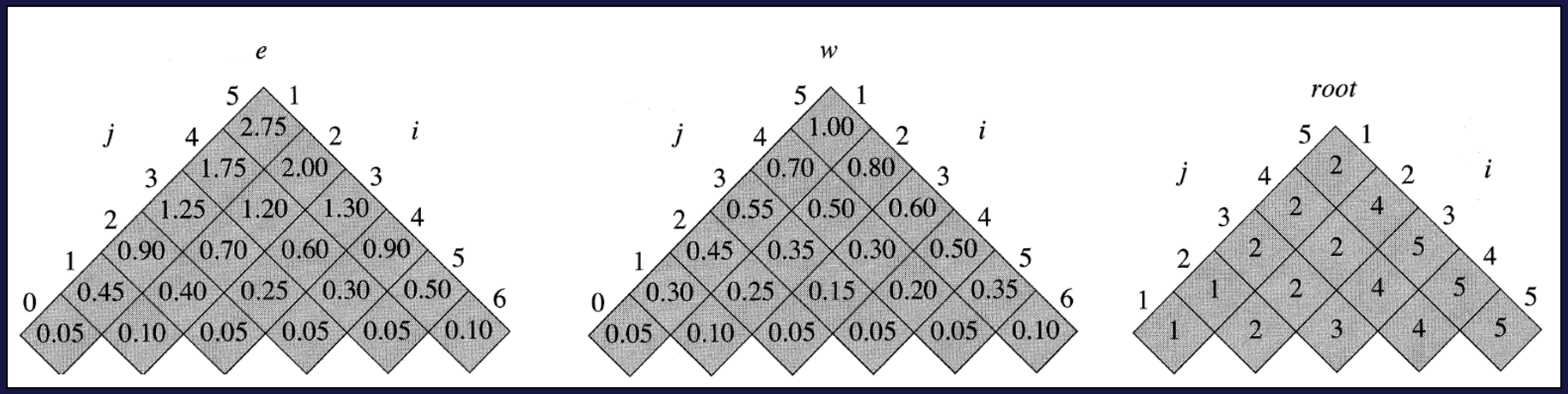






Hvad er roden i et optimalt binært søgetræ ?

a) k_1
b) k_2
c) k_3
d) k_4
e) k_5
f) Ved ikke



Dynamisk Programmering

- **Generel algoritmisk teknik**
- ***Krav***: ”Optimal delstruktur” – en løsning til problemet kan konstrueres ud fra optimale løsninger til ”delproblemer”
- **Rekursionsligning**
- **Eksempler**
  - Stang opdeling
  - Matrix-kæde multiplikation
  - Længste fælles delsekvens
  - Optimale søgetræer