Algoritmer og Datastrukturer 2

Gerth Stølting Brodal

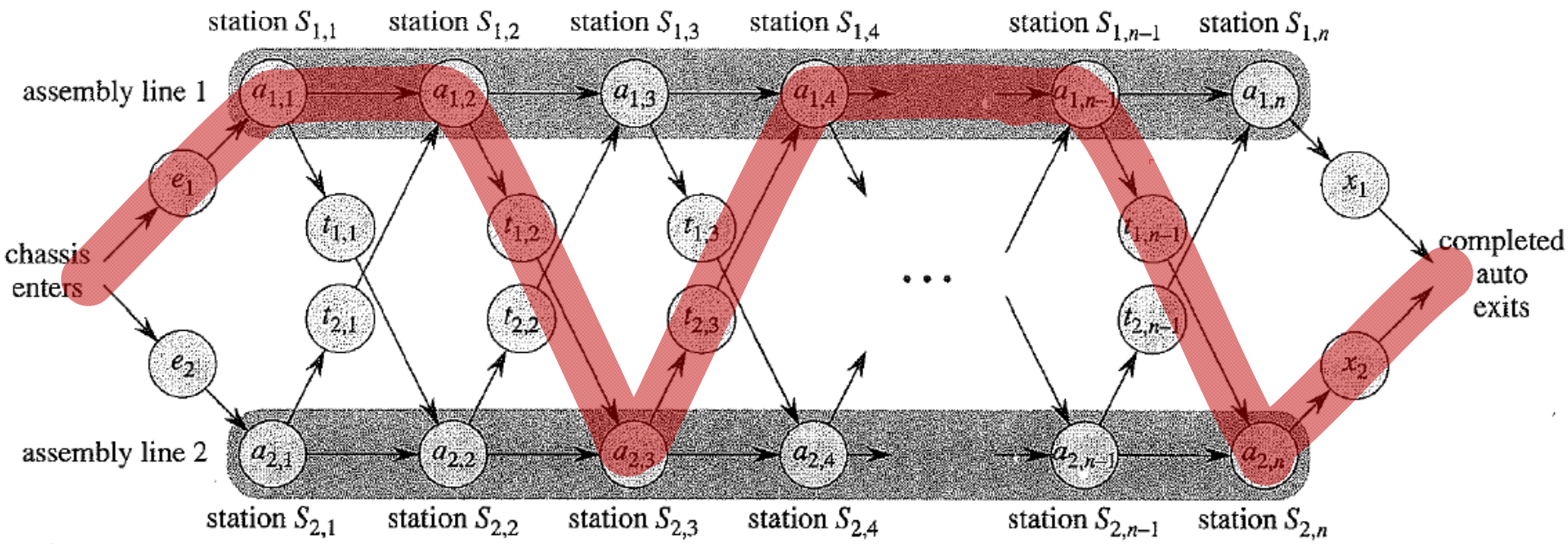
Dynamisk Programmering
[CLRS 15]

AARHUS UNIVERSITET

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

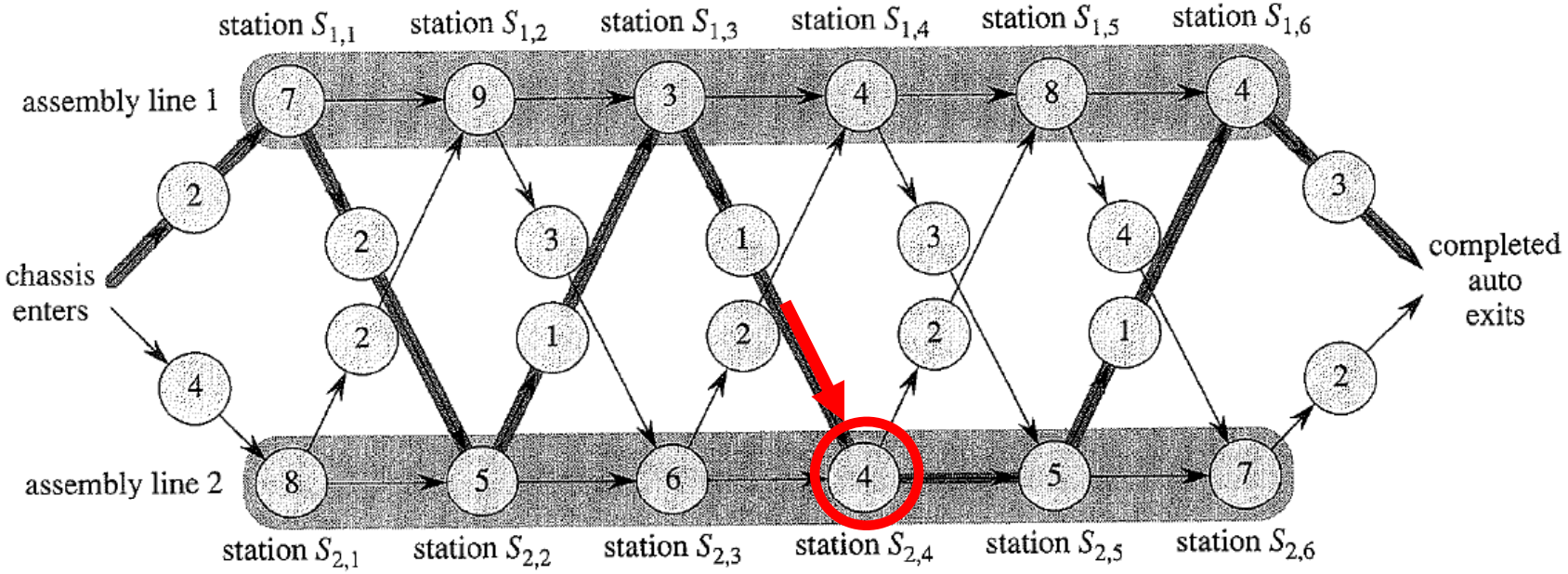
”Colonel Motors Corporation”



Mål Find en hurtigste vej fra ”Chassis enters” til ”completed auto” – knuderne angiver hvor lang tid de forskellige ting tager

Naive algoritme Prøv alle 2^n forskellige veje – tid $\Omega(2^n)$

”Colonel Motors Corporation”



(a)

j	1	2	3	4	5	6
$f_1[j]$	9	18	20	24	32	35
$f_2[j]$	12	16	22	25	30	37

$f^* = 38$

Korteste tid til og med stationerne

(b)

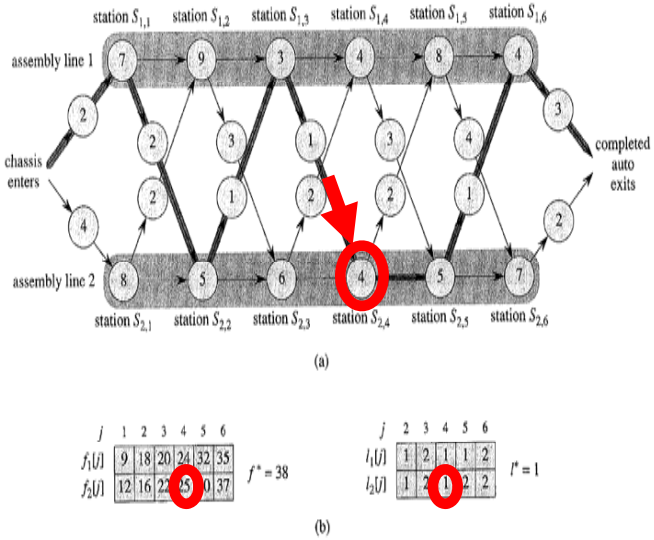
j	2	3	4	5	6
$l_1[j]$	1	2	1	1	2
$l_2[j]$	1	2	1	2	2

$l^* = 1$

Forrige station for hurtigste løsning

”Colonel Motors Corporation”

```
FASTEST-WAY(a, t, e, x, n)
 1  f1[1] ← e1 + a1,1
 2  f2[1] ← e2 + a2,1
 3  for j ← 2 to n
 4       do if f1[j − 1] + a1,j ≤ f2[j − 1] + t2,j−1 + a1,j
 5            then f1[j] ← f1[j − 1] + a1,j
 6                 l1[j] ← 1
 7            else f1[j] ← f2[j − 1] + t2,j−1 + a1,j
 8                 l1[j] ← 2
 9          if f2[j − 1] + a2,j ≤ f1[j − 1] + t1,j−1 + a2,j
10            then f2[j] ← f2[j − 1] + a2,j
11                 l2[j] ← 2
12            else f2[j] ← f1[j − 1] + t1,j−1 + a2,j
13                 l2[j] ← 1
14  if f1[n] + x1 ≤ f2[n] + x2
15     then f* = f1[n] + x1
16          l* = 1
17     else f* = f2[n] + x2
18          l* = 2
```

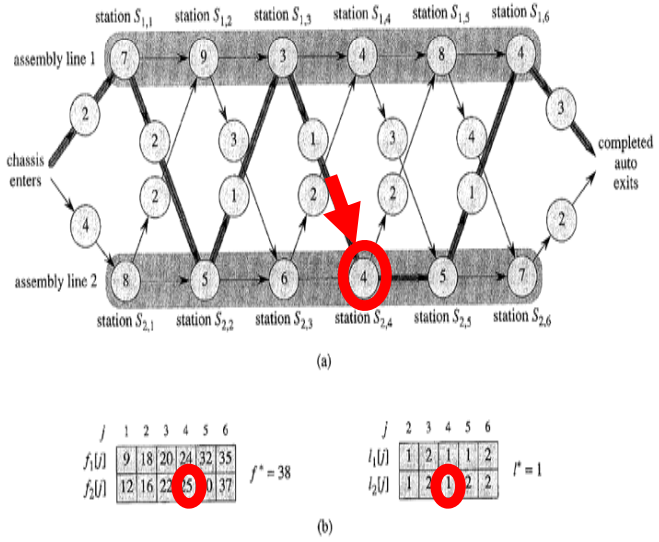


***Længden* af den hurtigste vej findes i tid** $O(n)$

”Colonel Motors Corporation”

```
PRINT-STATIONS(l, n)
1  i ← l*
2  print “line ” i “, station ” n
3  for j ← n downto 2
4      do i ← l_i[j]
5         print “line ” i “, station ” j − 1
```

line 1, station 6
line 2, station 5
line 2, station 4
line 1, station 3
line 2, station 2
line 1, station 1



Gå ”baglæns” igennem de beregnede værdier, og find løsningen

Matrix-kæde Multiplikation

```
MATRIX-MULTIPLY(A, B)
1  if columns[A] ≠ rows[B]
2     then error "incompatible dimensions"
3     else for i ← 1 to rows[A]
4               do for j ← 1 to columns[B]
5                     do C[i, j] ← 0
6                        for k ← 1 to columns[A]
7                           do C[i, j] ← C[i, j] + A[i, k] · B[k, j]
8           return C
```

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

Matrix-kæde Multiplikation

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

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

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 $A_i \cdot \ldots \cdot A_j$

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

```
RECURSIVE-MATRIX-CHAIN(p, i, j)
1  if i = j
2     then return 0
3  m[i, j] ← ∞
4  for k ← i to j − 1
5        do 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              then m[i, j] ← q
8  return m[i, j]
```

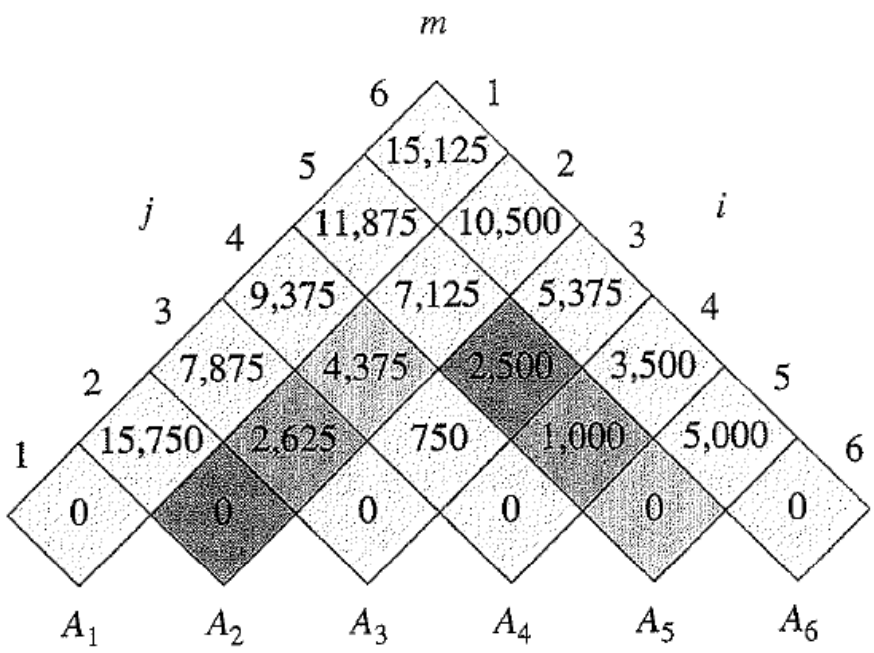
Tid $\Omega(4^n/n^{3/2})$

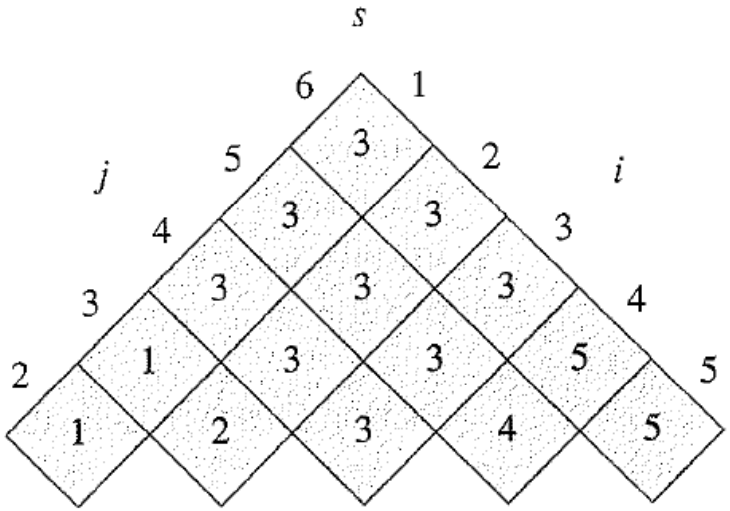
Matrix-kæde Multiplikation

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

Tid $O(n^3)$

Matrix-kæde Multiplikation





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

Matrix-kæde Multiplikation

```
PRINT-OPTIMAL-PARENS(s, i, j)
1  if i = j
2      then 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 $O(n)$

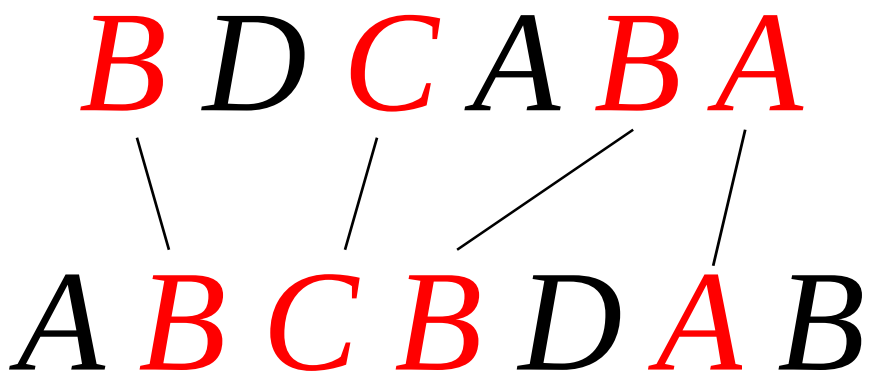
”Memoized”
Matrix-kæde Multiplikation

```
MEMOIZED-MATRIX-CHAIN(p)
1  n ← length[p] − 1
2  for i ← 1 to n
3        do for j ← i to n
4              do m[i, j] ← ∞
5  return LOOKUP-CHAIN(p, 1, n)

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

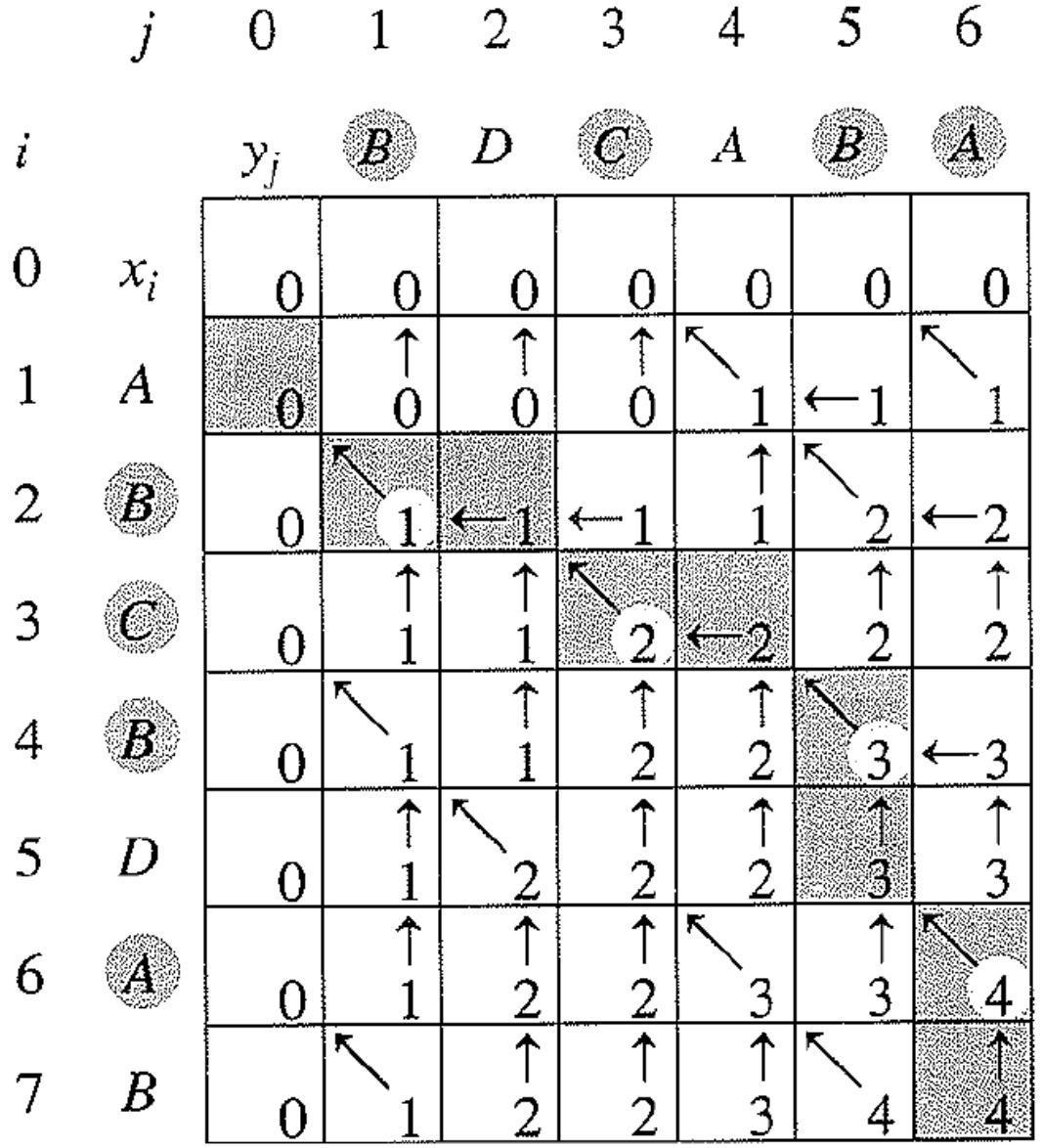
Tid $O(n^3)$

Længste Fælles Delsekvens



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

Længste Fælles Delsekvens



Længste Fælles Delsekvens

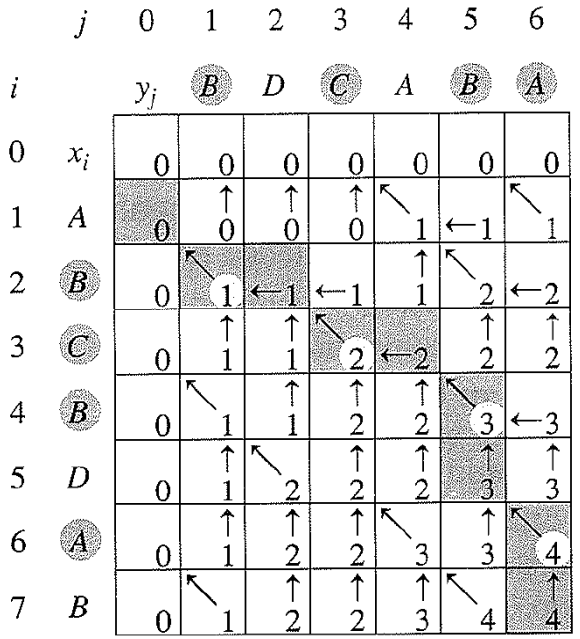
```
LCS-LENGTH(X, Y)
 1  m ← length[X]
 2  n ← length[Y]
 3  for i ← 1 to m
 4       do c[i, 0] ← 0
 5  for j ← 0 to n
 6       do c[0, j] ← 0
 7  for i ← 1 to m
 8       do for j ← 1 to n
 9            do if x_i = y_j
10                 then c[i, j] ← c[i − 1, j − 1] + 1
11                      b[i,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

	j	0	1	2	3	4	5	6
i		y_j	*B*	*D*	*C*	*A*	*B*	*A*
0	x_i	0	0	0	0	0	0	0
1	*A*	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	*B*	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	*C*	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	*B*	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	*D*	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	*A*	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	*B*	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

Tid O(nm)

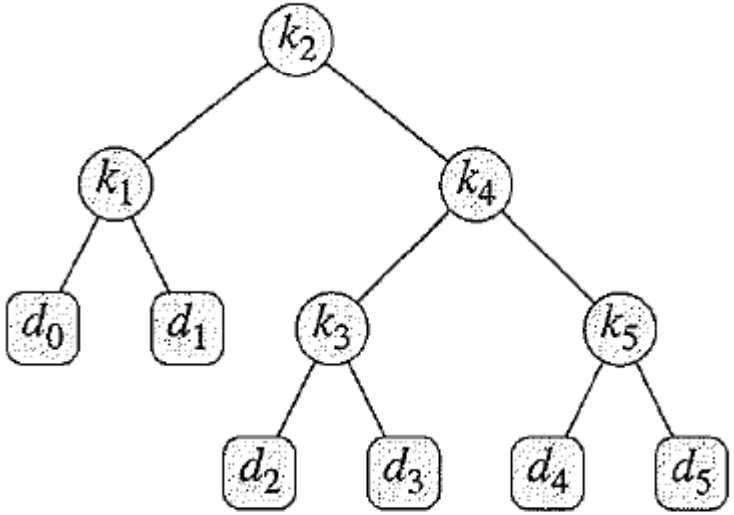
Længste Fælles Delsekvens

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

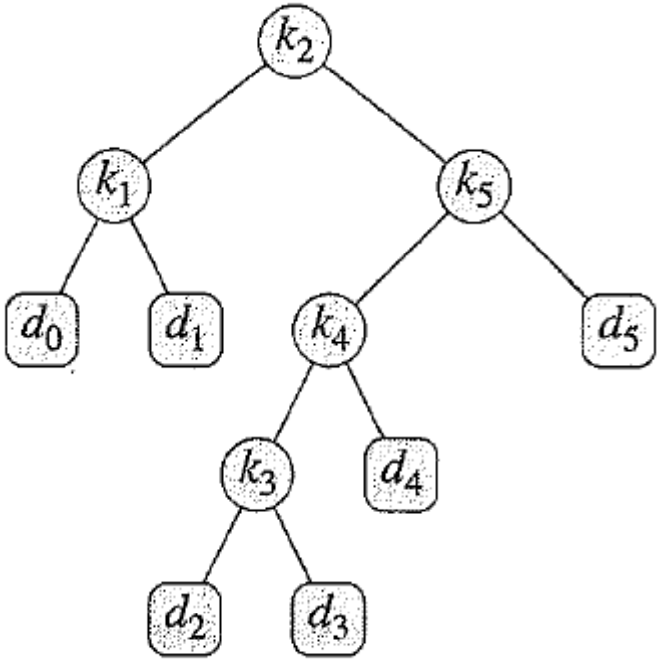


Tid $O(n)$

Optimale Binære Søgetræer



Forventet søgetid 2.80



Forventet søgetid 2.75

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

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

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

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

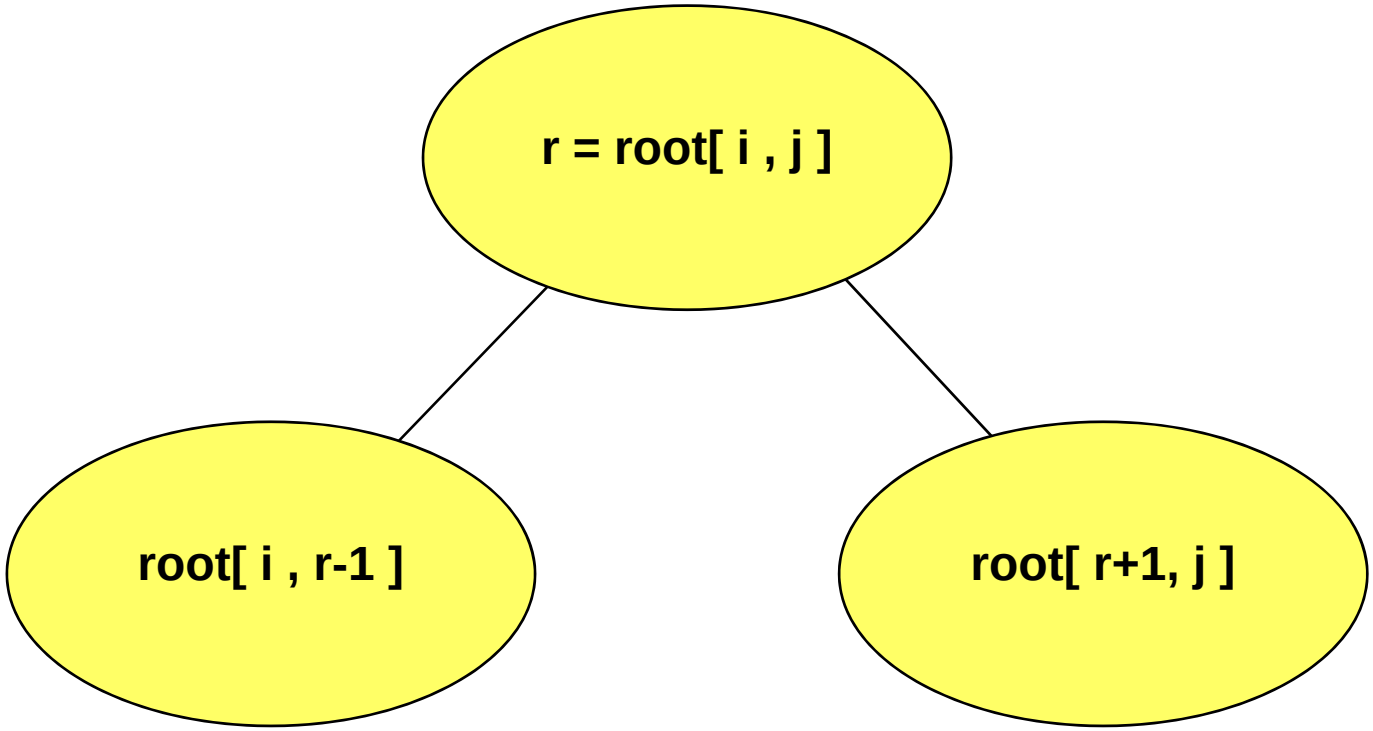
Optimale Binære Søgetræer

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

```
OPTIMAL-BST(p, q, n)
 1  for i ← 1 to n + 1
 2       do e[i, i − 1] ← q_{i−1}
 3          w[i, i − 1] ← q_{i−1}
 4  for l ← 1 to n
 5       do for i ← 1 to n − l + 1
 6             do j ← i + l − 1
 7                e[i, j] ← ∞
 8                w[i, j] ← w[i, j − 1] + p_j + q_j
 9                for r ← i to j
10                   do t ← e[i, r − 1] + e[r + 1, j] + w[i, j]
11                      if t < e[i, j]
12                         then e[i, j] ← t
13                              root[i, j] ← r
14  return e and root
```

Tid $O(n^3)$

Konstruktion af Optimalt Træ



Dynamisk Programmering

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