Algoritmer og Datastrukturer 2

Gerth Stølting Brodal

Dynamisk Programmering
[CLRS 15]

AARHUS UNIVERSITET

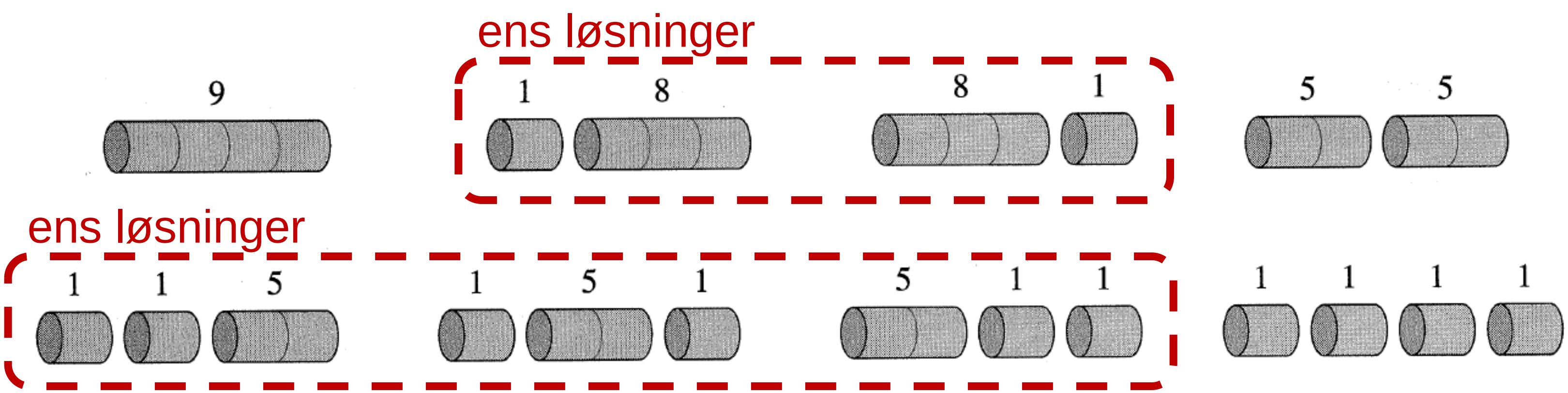
Dynamisk Programmering

- **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

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 maksimerereret



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

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) .

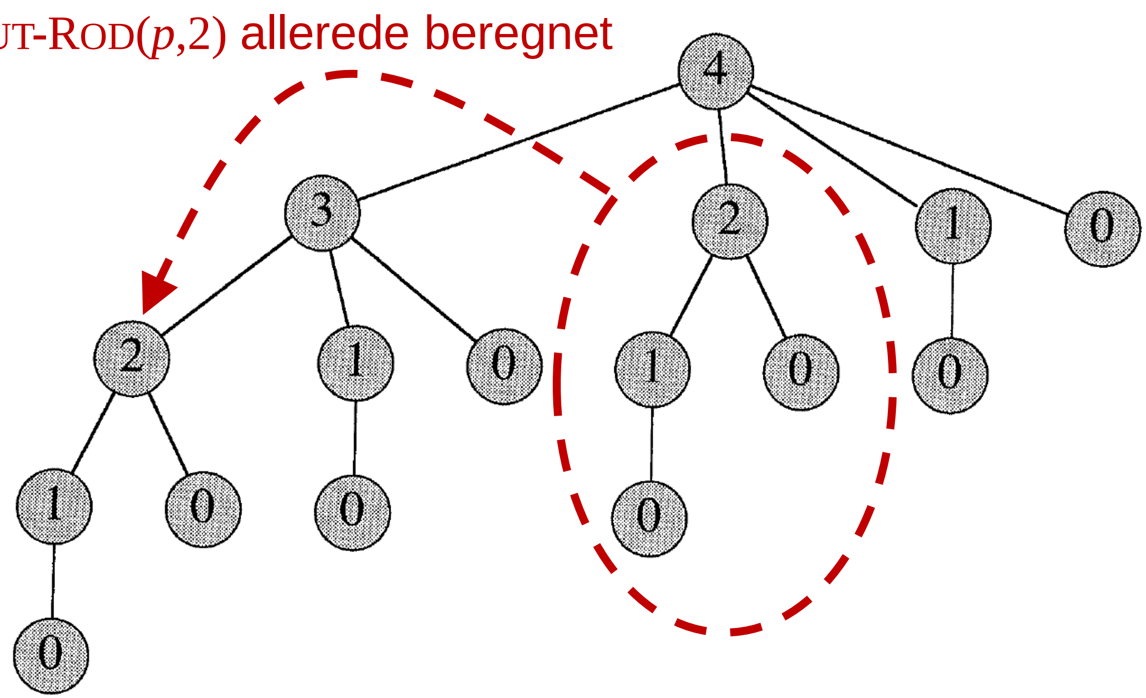
length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

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

Optimal opdeling af en stang: Rekursiv løsning

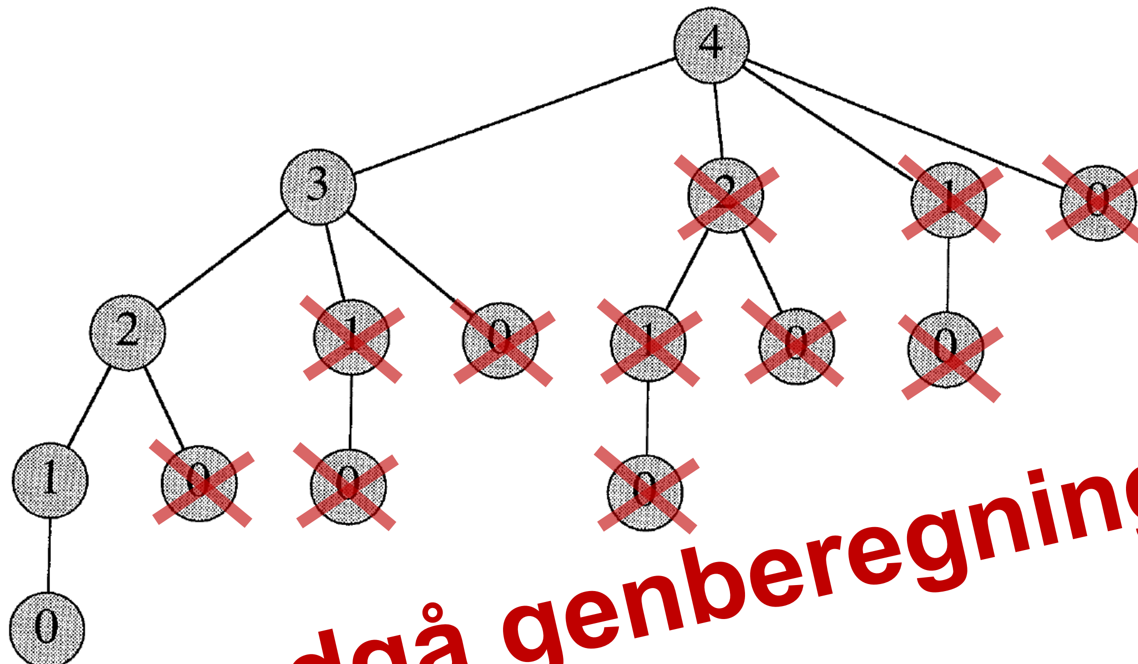



```
CUT-ROD(p, n)
    if n == 0
        return 0
    q = −∞
    for i = 1 to n
        q = max(q, p[i] + CUT-ROD(p, n − i))
    return q
```

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

Tid $O(2^n)$

Optimal opdeling af en stang: Rekursiv løsning



undgå genberegning ?

CUT-ROD(p, n)

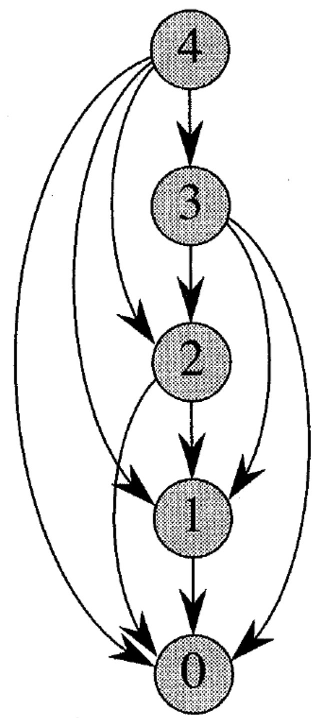
```
if n == 0
    return 0
q = −∞
for i = 1 to n
    q = max(q, p[i] + CUT-ROD(p, n − i))
return q
```

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

Optimal opdeling af en stang: Rekursiv løsning + Memoization

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

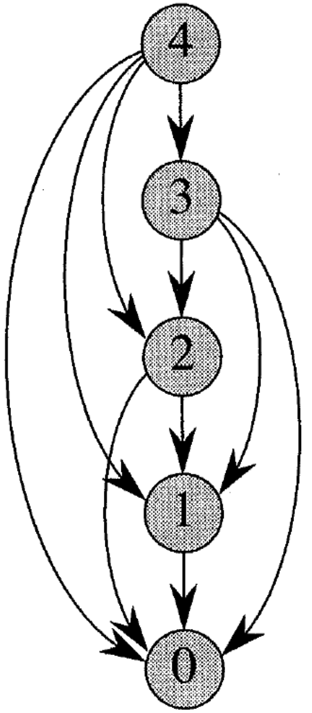
```
MEMOIZED-CUT-ROD-AUX(p, n, r)
if r[n] ≥ 0
    return r[n]
if n == 0
    q = 0
else q = −∞
    for i = 1 to n
        q = max(q, p[i] + MEMOIZED-CUT-ROD-AUX(p, n − i, r))
r[n] = q          ← husk resultatet !
return q
```



Tid $O(n^2)$

Optimal opdeling af en stang: Systematisk udfyldning

```
BOTTOM-UP-CUT-ROD(p, n)
let r[0..n] be a new array
r[0] = 0
for j = 1 to n            ← rækkefølgen vigtig !
    q = −∞
    for i = 1 to j
        q = max(q, p[i] + r[j − i])
    r[j] = q
return r[n]
```



Tid $O(n^2)$

Optimal opdeling af en stang: Udskrivning af løsningen

```
EXTENDED-BOTTOM-UP-CUT-ROD(p, n)
let r[0..n] and s[0..n] be new arrays
r[0] = 0
for j = 1 to n
    q = −∞
    for i = 1 to j
        if q < p[i] + r[j − i]
            q = p[i] + r[j − i]
            s[j] = i
    r[j] = q
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)
(r, s) = EXTENDED-BOTTOM-UP-CUT-ROD(p, n)
while n > 0
    print s[n]
    n = n − s[n]
```

Tid $O(n^2+n)$

Matrix Multiplikation

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

$$(A \cdot B) \cdot C \quad \text{eller} \quad A \cdot (B \cdot C) \quad ?$$

Matrix multiplikation er associativ (kan sætte paranteser som man vil) 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 $A_i \cdot \cdots \cdot A_j$

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

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

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

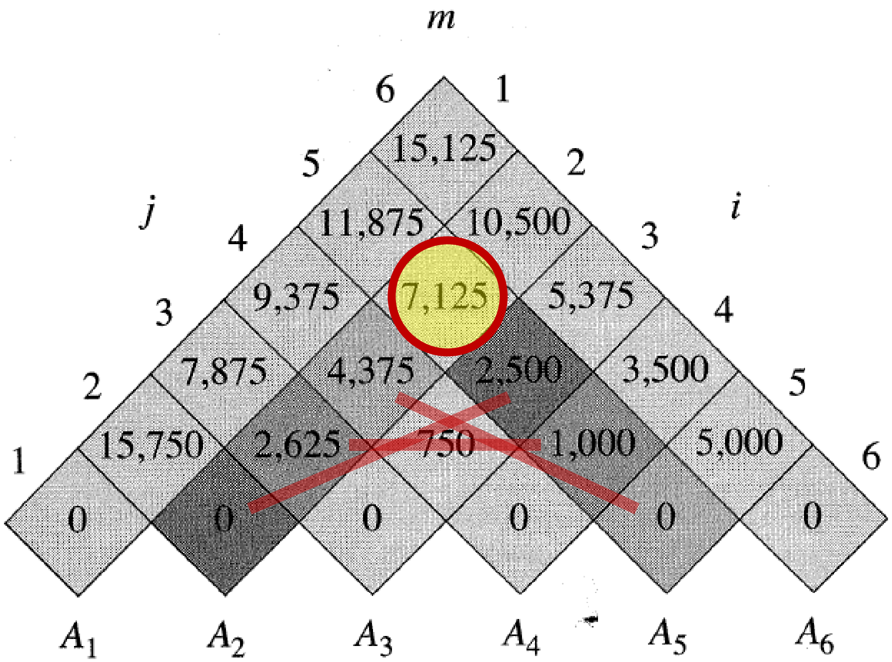
Tid $\Omega(4^n/n^{3/2})$

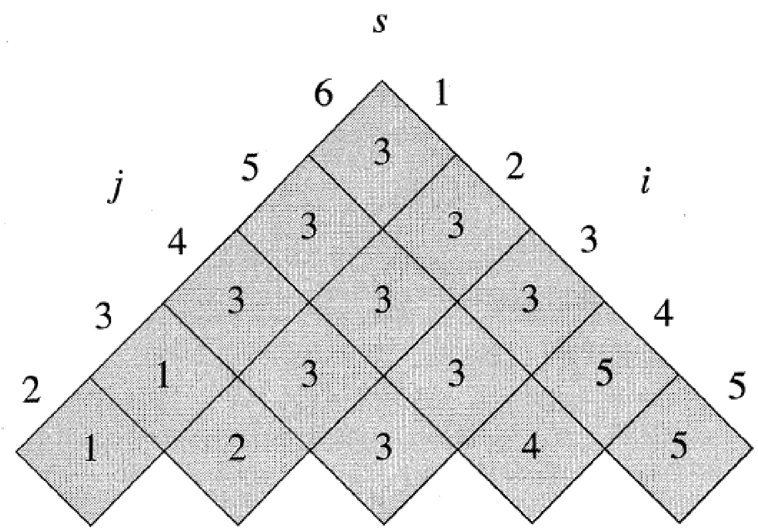
Matrix-kæde Multiplikation

```
MATRIX-CHAIN-ORDER(p)
n = p.length − 1
let m[1..n, 1..n] and s[1..n − 1, 2..n] be new tables
for i = 1 to n
    m[i, i] = 0
for l = 2 to n                    // l is the chain length
    for i = 1 to n − l + 1
        j = i + l − 1
        m[i, j] = ∞
        for k = i to j − 1
            q = m[i, k] + m[k + 1, j] + p_{i−1} p_k p_j
            if q < m[i, j]
                m[i, j] = q
                s[i, j] = k
return m and s
```

Tid $O(n^3)$

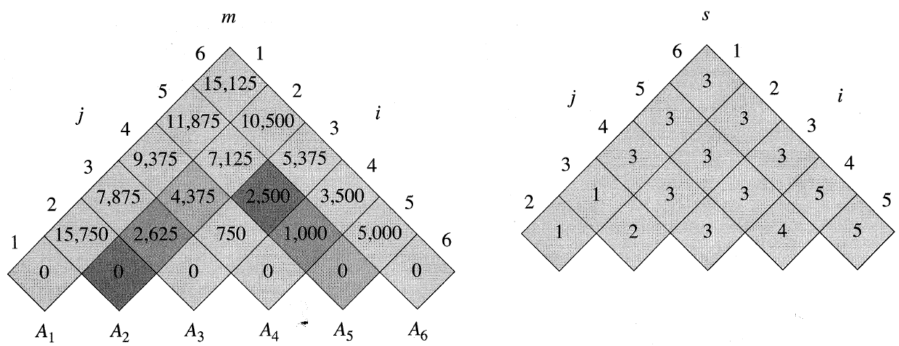
Matrix-kæde Multiplikation





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

Tid O(n)

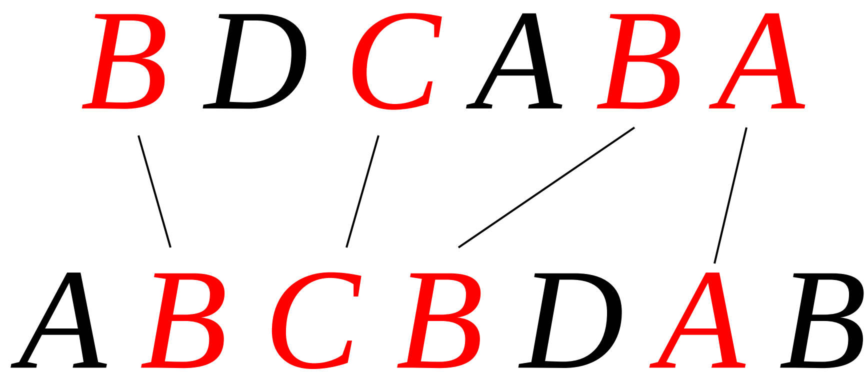
"Memoized" Matrix-kæde Multiplikation

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

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

Tid $O(n^3)$

Længste Fælles Delsekvens



Længste Fælles Delsekvens

j

$$Y = B\ D\ C\ A\ B\ A$$

$$X = A\ B\ C\ B\ D\ A\ B$$

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 $y_1 y_2 \cdots y_j$

Længste Fælles Delsekvens

i		j: 0	1	2	3	4	5	6
		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

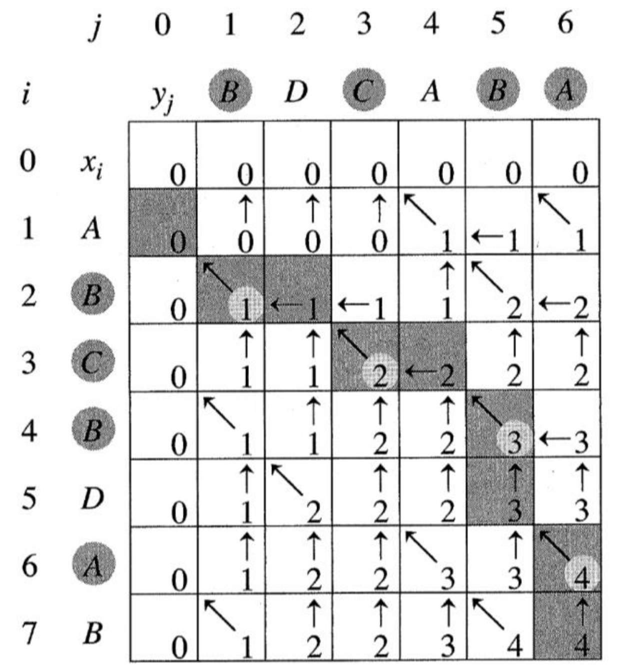
B D C A B A

A B C B D A B

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

Længste Fælles Delsekvens

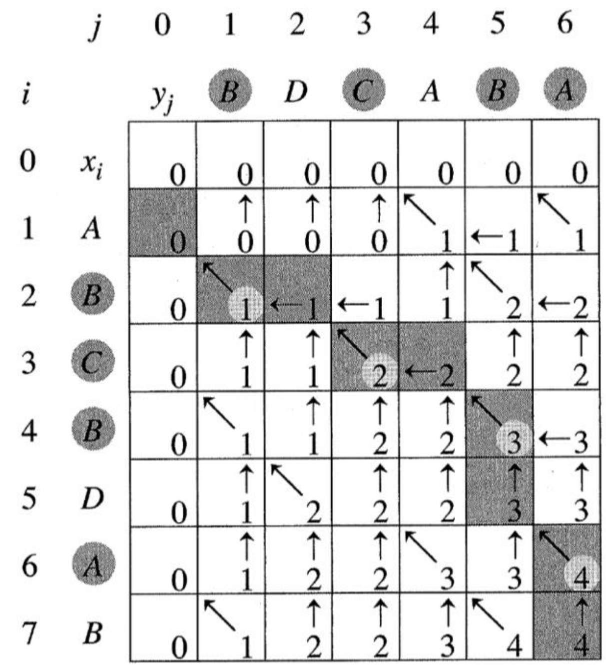
```
LCS-LENGTH(X, Y)
m = X.length
n = Y.length
let b[1..m, 1..n] and c[0..m, 0..n] be new tables
for i = 1 to m
    c[i, 0] = 0
for j = 0 to n
    c[0, j] = 0
for i = 1 to m
    for j = 1 to n
        if x_i == y_j
            c[i, j] = c[i − 1, j − 1] + 1
            b[i, j] = "↖"
        elseif c[i − 1, j] ≥ c[i, j − 1]
            c[i, j] = c[i − 1, j]
            b[i, j] = "↑"
        else c[i, j] = c[i, j − 1]
            b[i, j] = "←"
return c and b
```



Tid O(*nm*)

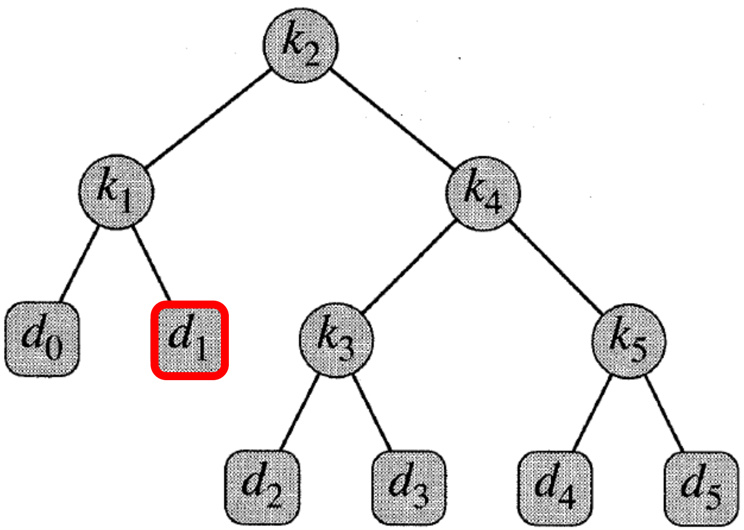
Længste Fælles Delsekvens

```
PRINT-LCS(b, X, i, j)
    if i == 0 or j == 0
        return
    if b[i, j] == "↖"
        PRINT-LCS(b, X, i − 1, j − 1)
        print x_i
    elseif b[i, j] == "↑"
        PRINT-LCS(b, X, i − 1, j)
    else PRINT-LCS(b, X, i, j − 1)
```

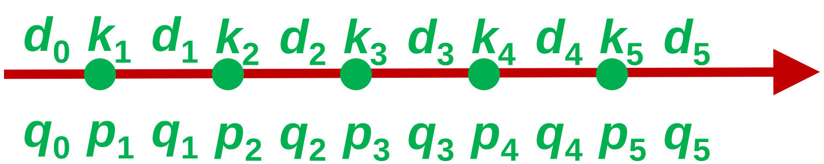


Tid O(n+m)

Optimale Binære Søgetræer



Forventet søgetid 2.80

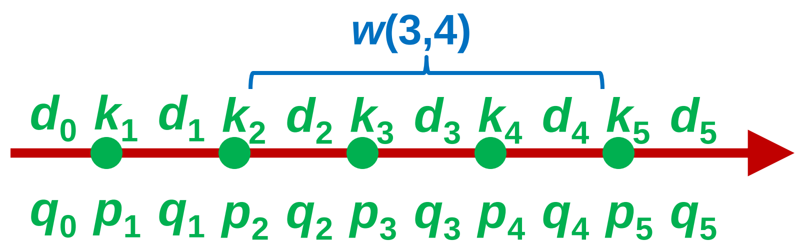


node	depth	probability	contribution
k_1	1	0.15	0.30
k_2	0	0.10	0.10
k_3	2	0.05	0.15
k_4	1	0.10	0.20
k_5	2	0.20	0.60
d_0	2	0.05	0.15
d_1	(2 + 1) x	0.10 =	0.30
d_2	3	0.05	0.20
d_3	3	0.05	0.20
d_4	3	0.05	0.20
d_5	3	0.10	0.40
Total			2.80

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

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



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

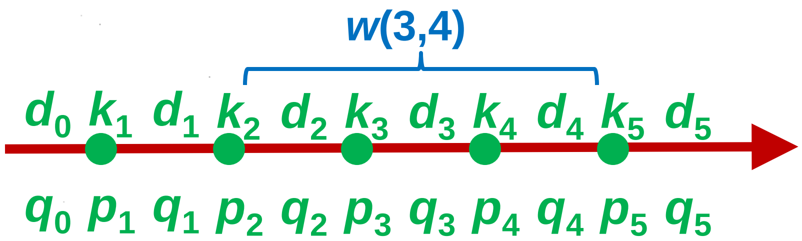
$$e[i,j] = \begin{cases} q_{i-1} & \text{if } j = i - 1\,, \\ \min\limits_{i \le r \le j} \{e[i, r-1] + e[r+1, j] + w(i,j)\} & \text{if } i \le j\,. \end{cases}$$

forventet optimal tid for et søgetræ indeholdende $k_i, \ldots, k_j$

Optimale Binære Søgetræer

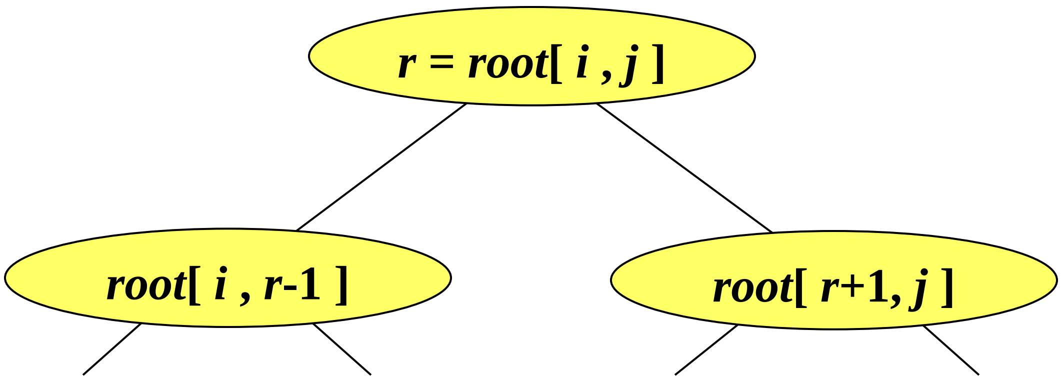
$$e[i,j] = \begin{cases} q_{i-1} & \text{if } j = i-1\,, \\ \min\limits_{i \le r \le j} \{e[i,r-1] + e[r+1,j] + w(i,j)\} & \text{if } i \le j\,. \end{cases}$$

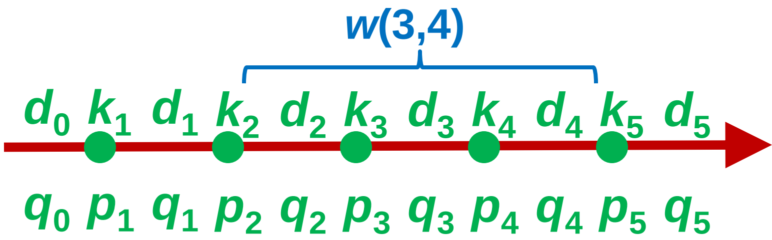
```
OPTIMAL-BST(p, q, n)
let e[1..n+1, 0..n], w[1..n+1, 0..n],
        and root[1..n, 1..n] be new tables
for i = 1 to n + 1
    e[i, i − 1] = q_{i−1}
    w[i, i − 1] = q_{i−1}
for l = 1 to n
    for i = 1 to n − l + 1
        j = i + l − 1
        e[i, j] = ∞
        w[i, j] = w[i, j − 1] + p_j + q_j
        for r = i to j
            t = e[i, r − 1] + e[r + 1, j] + w[i, j]
            if t < e[i, j]
                e[i, j] = t
                root[i, j] = r
return e and root
```

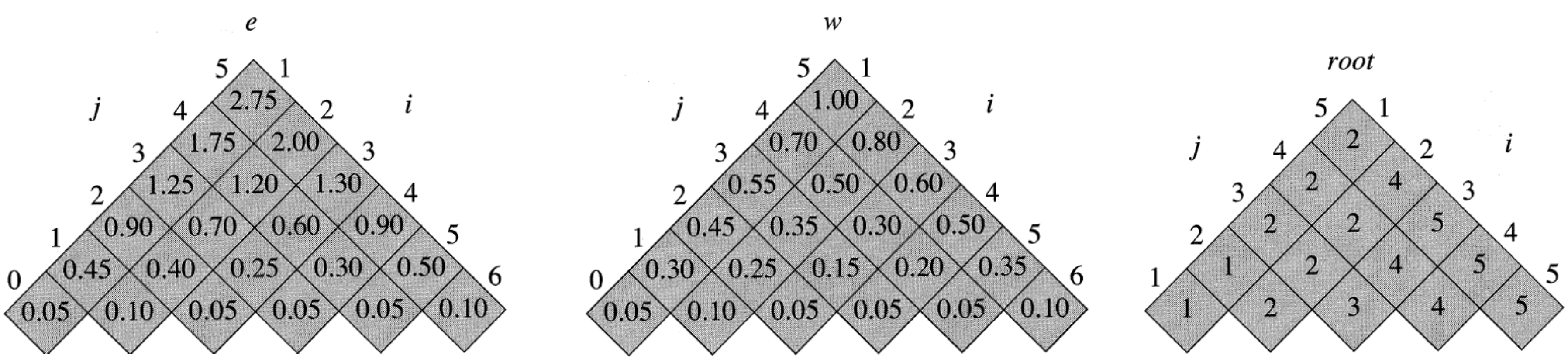


Tid $O(n^3)$

Konstruktion af
Optimalt Binært Søgetræ







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