Forelesning 7

Grådige algoritmer består av en serie med valg, og hvert valg tas *lokalt*: Algoritmen gjør alltid det som ser best ut her og nå, uten noe større perspektiv. Slike algoritmer er ofte enkle; utfordringen ligger i å finne ut om de gir rett svar.

Pensum

- [ ] Kap. 16. Greedy algorithms: Innledning og 16.1–16.3

Læringsmål

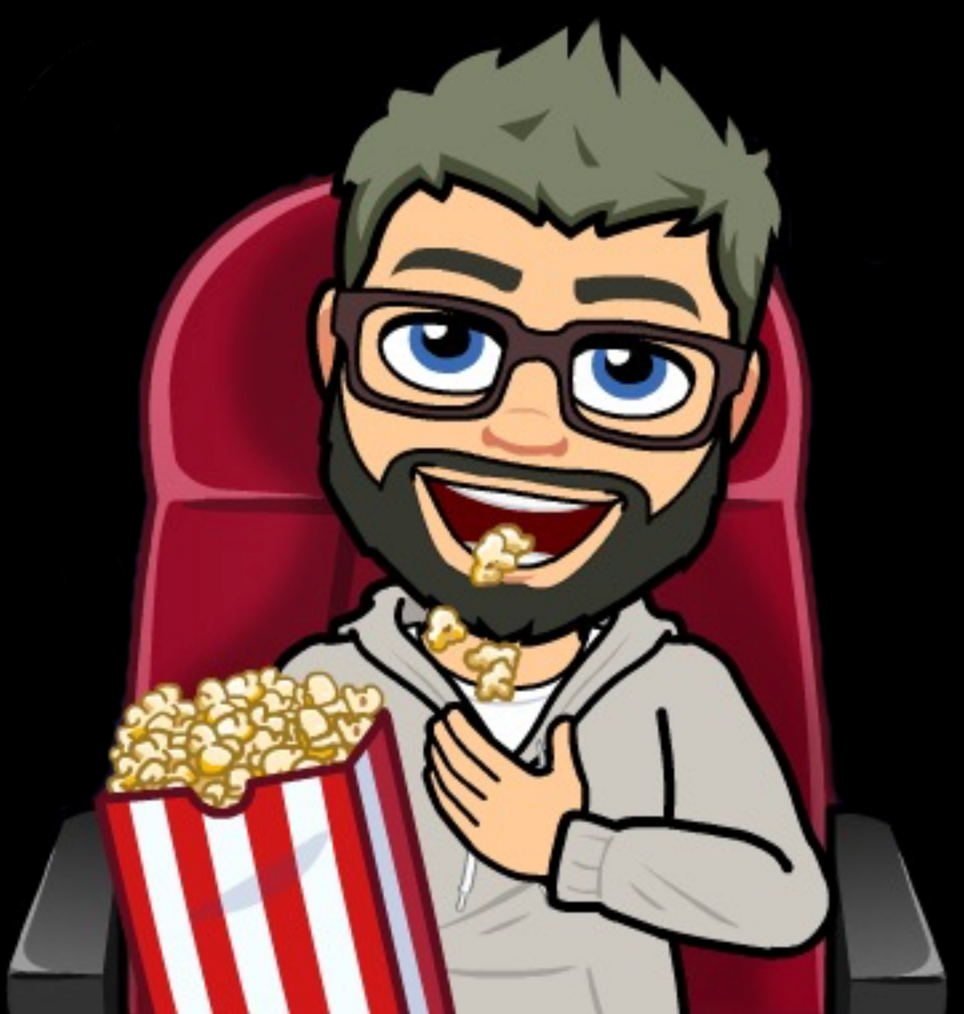
[G_1] Forstå designmetoden *grådighet*

[G_2] Forstå *grådighetsegenskapen* (*the greedy-choice property*)

[G_3] Forstå eksemplene *aktivitet-utvelgelse* og *det fraksjonelle ryggsekkproblemet*

[G_4] Forstå HUFFMAN og *Huffman-koder*

Forelesningen filmes



0:5

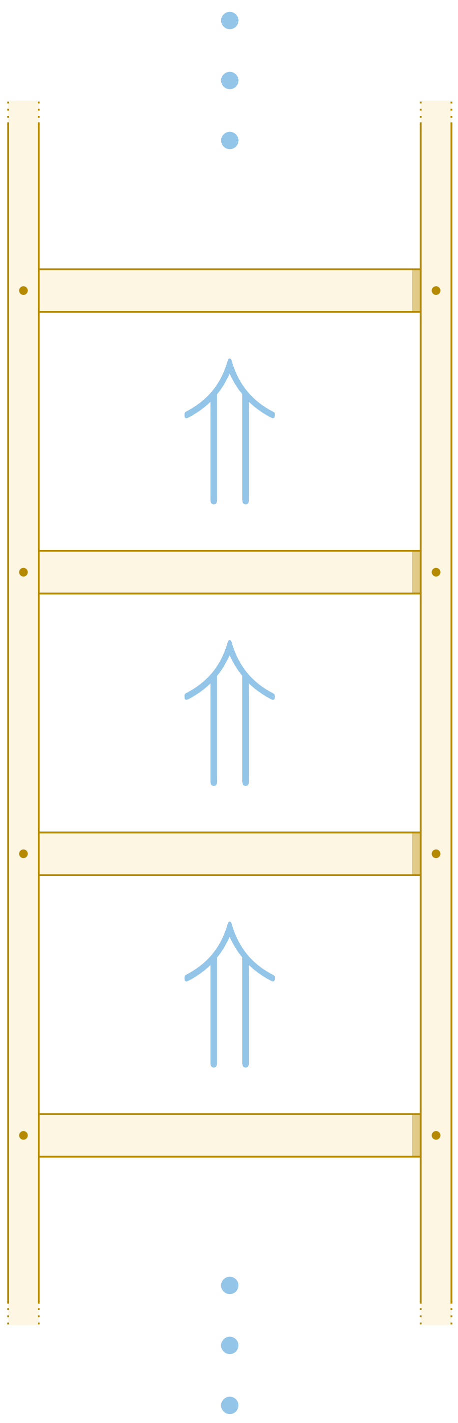
Mer DP, fra forrige gang

Optimal delstruktur

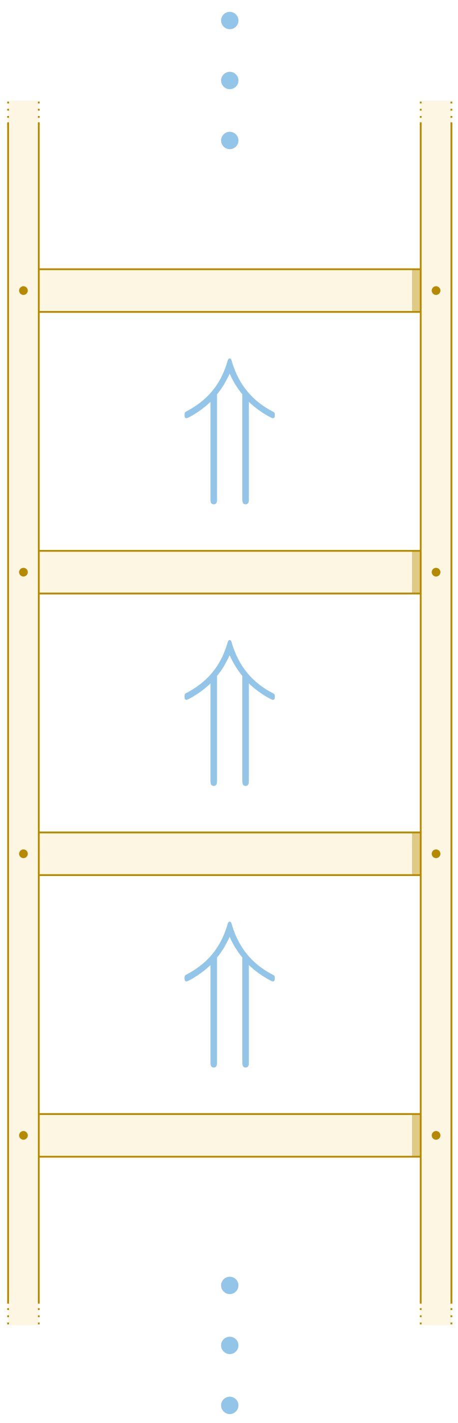
Det han omtaler som «remaining decisions» er det vi tenker på som delproblemer (selv om det høres litt bakvent ut). Hva vi gjr i vårt ene induktive trinn kommer an på optimale løsninger på delproblemene – og det forteller oss hva konsekvensene blir.

PRINCIPLE OF OPTIMALITY. *An optimal policy has the property that whatever the initial state and initial decisions are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decisions.*

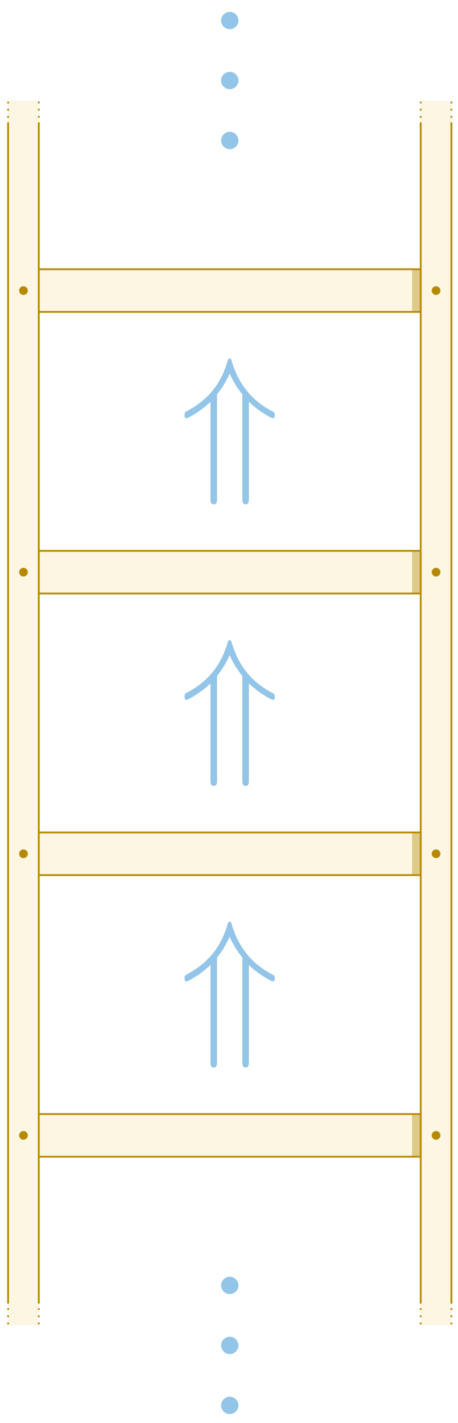
Richard Bellman, «The theory of dynamic programming», Bull. Amer. Math. Soc. 60 (1954), 503-515.



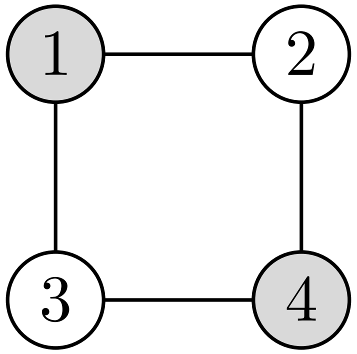
Opt. delstrukt.: Det finnes opt. løsning bestående av opt. delløsninger



Dette gir «smitte-effekten» vi trenger, dvs., det induktive trinnet!

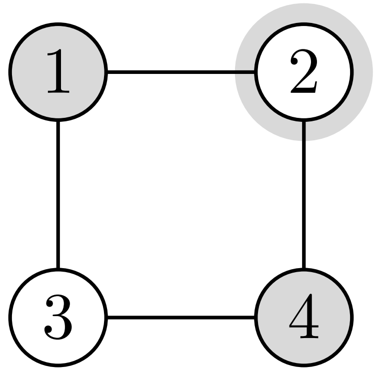


Finner optimale delløsninger; konstruerer optimal løsning

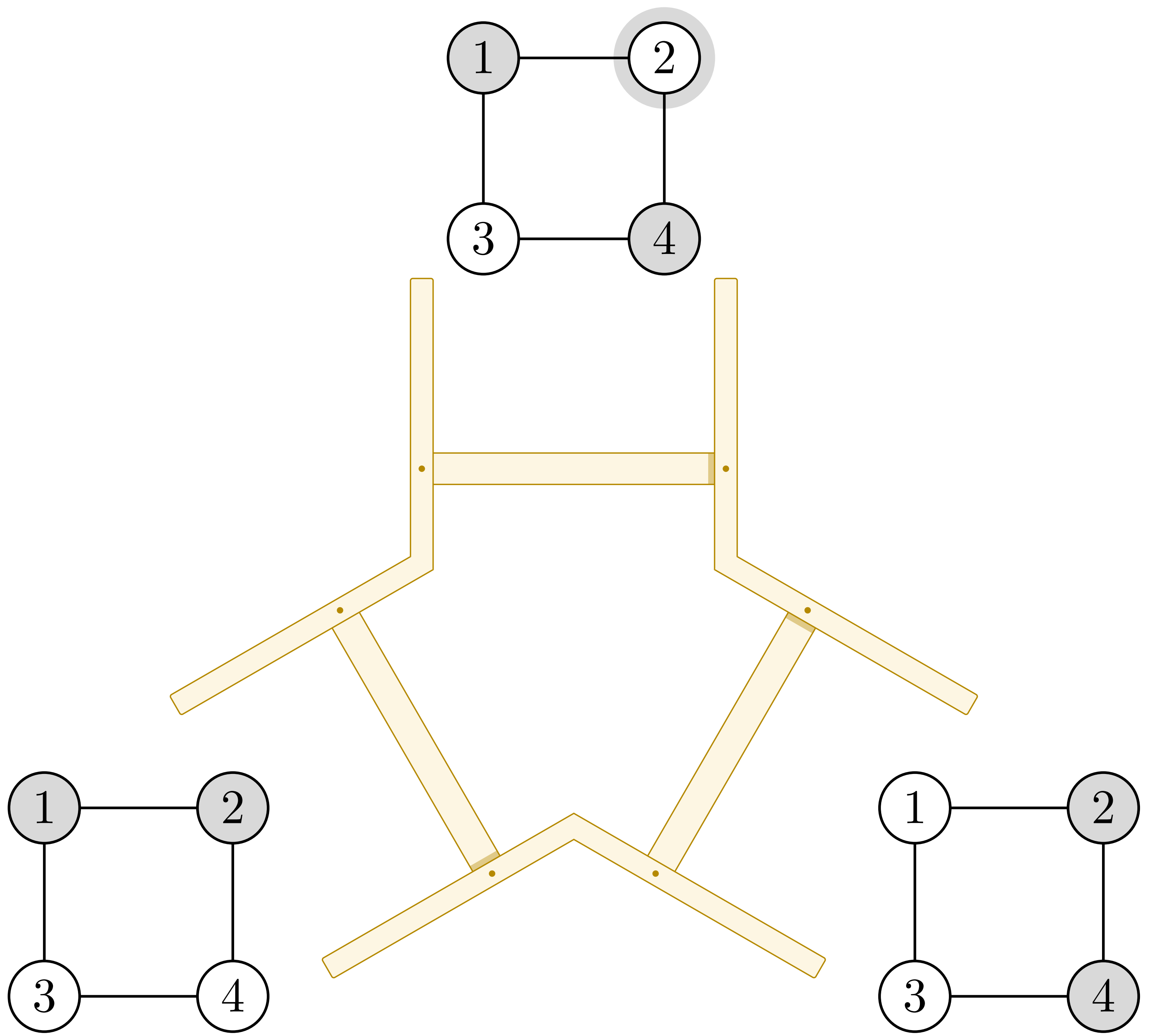


Vi vil finne korteste vei fra 1 til 4

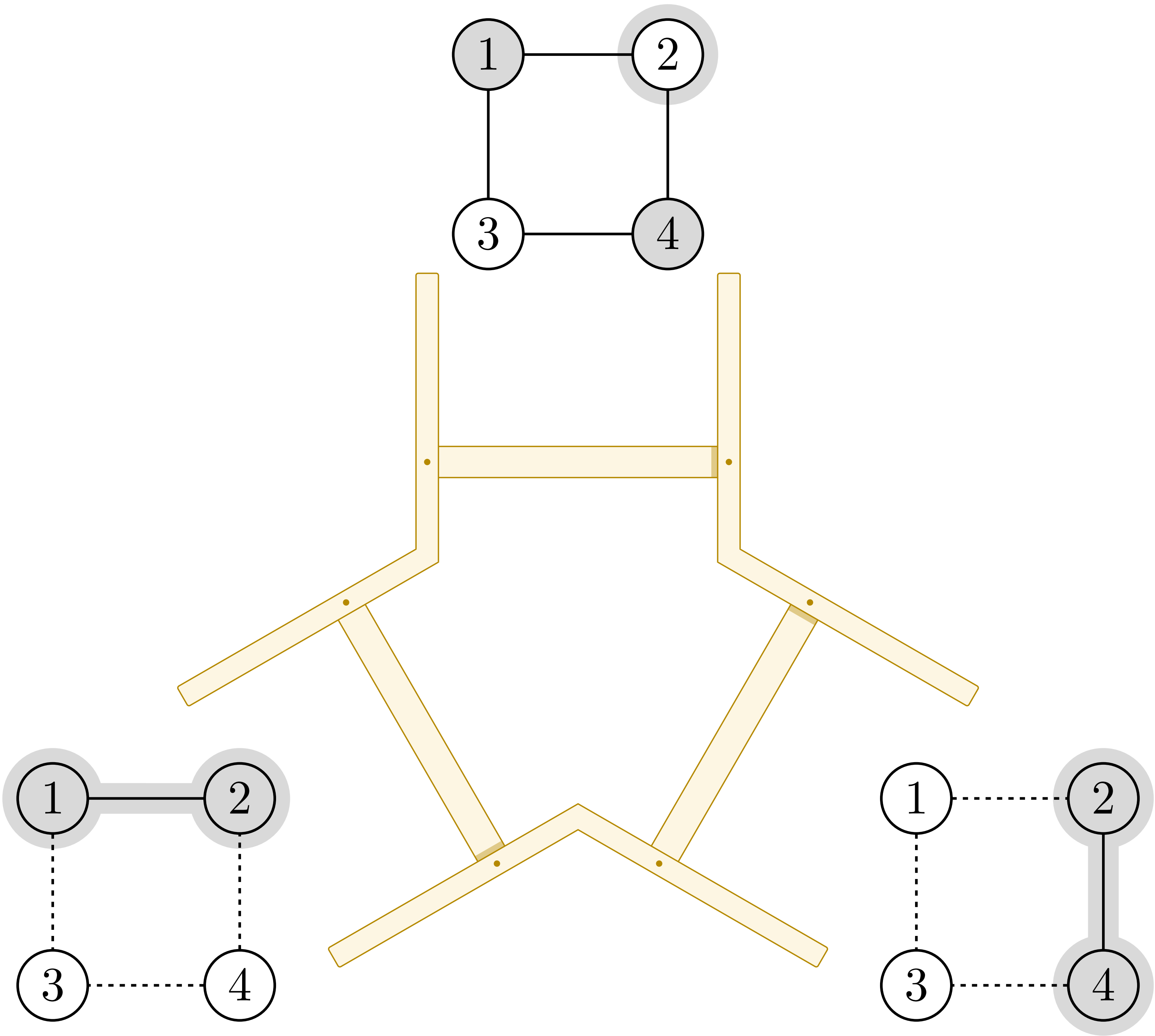
mlh@ntnu.no



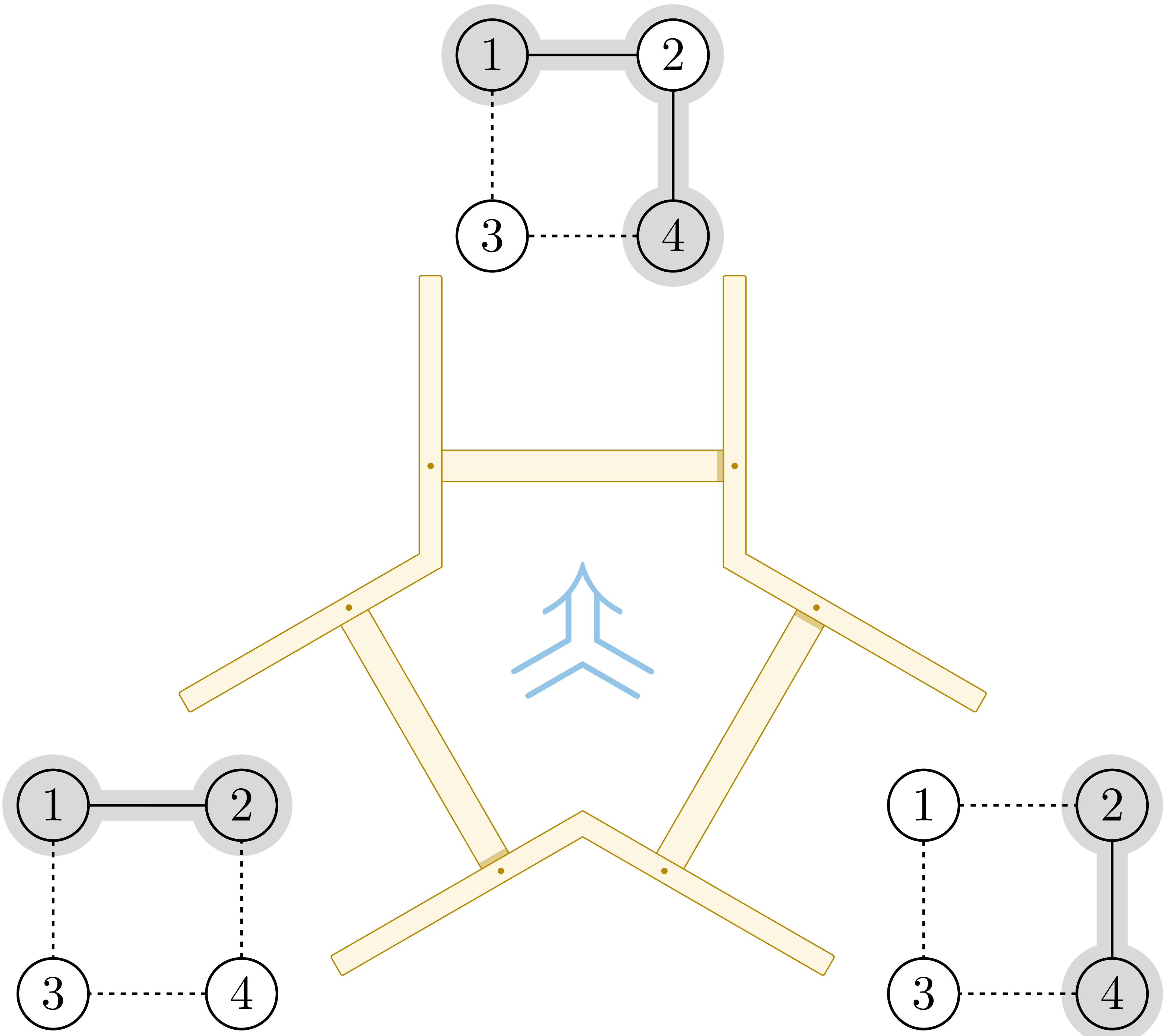
Vi velger tentativt å gå innom 2



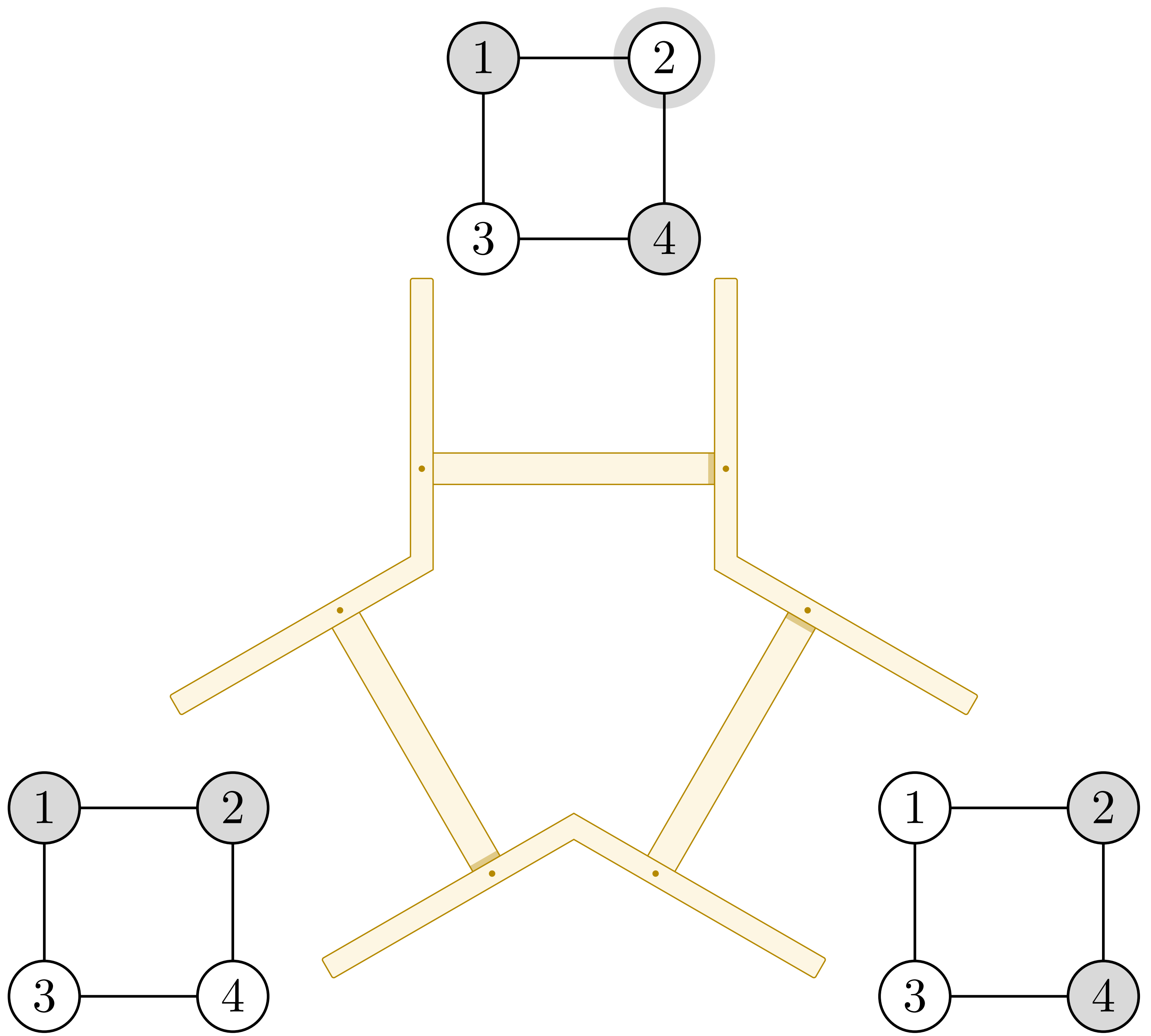
Kan dekomponere i korteste veier $1 \rightsquigarrow 2$ og $2 \rightsquigarrow 4$



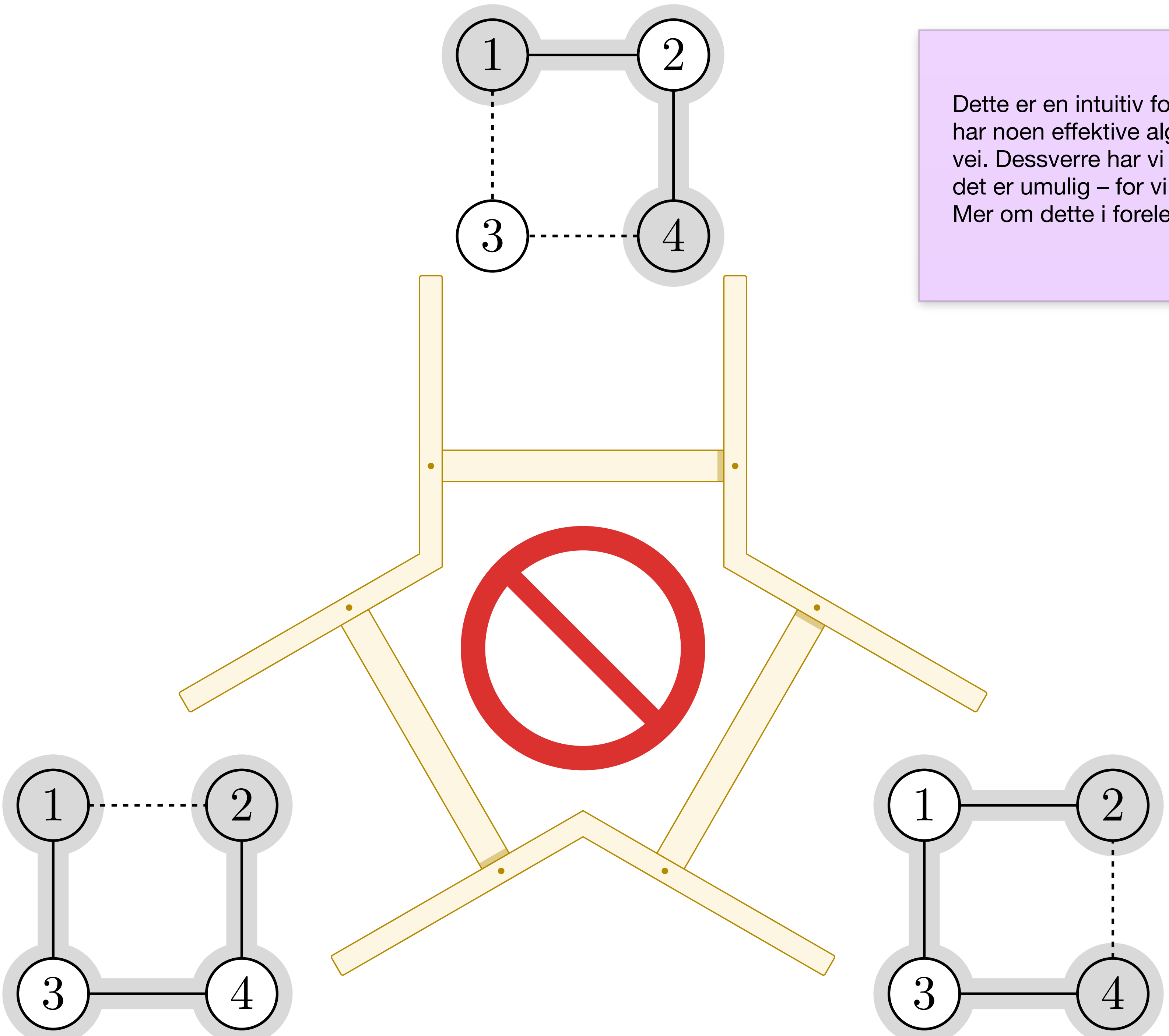
Om vi finner disse . . .



…så kan vi konstruere korteste vei $1 \rightsquigarrow 4$



Lengste vei: Prøver samme dekomponering



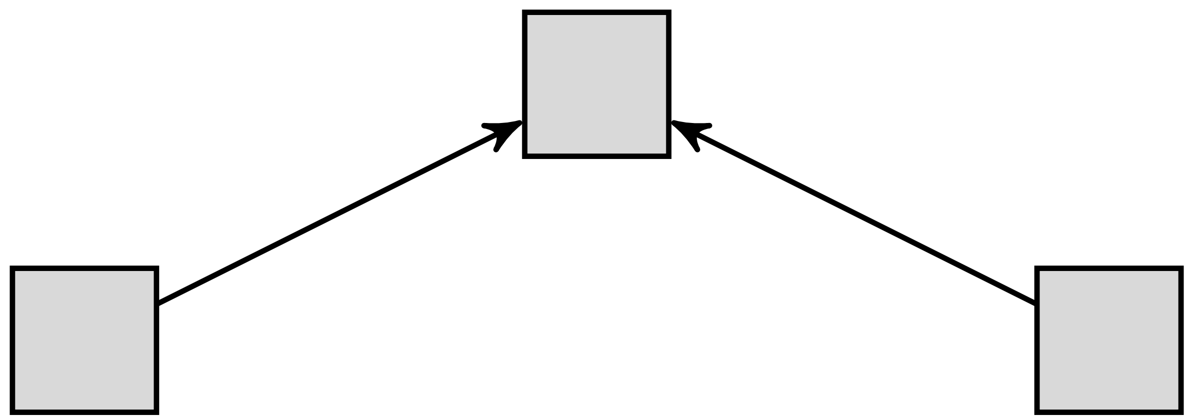
Dette er en intuitiv forklaring på hvorfor vi ikke har noen effektive algoritmer for å finne lengste vei. Dessverre har vi ingen forklaring på hvorfor det er umulig – for vi vet ikke om det *er* umulig! Mer om dette i forelesning 13!

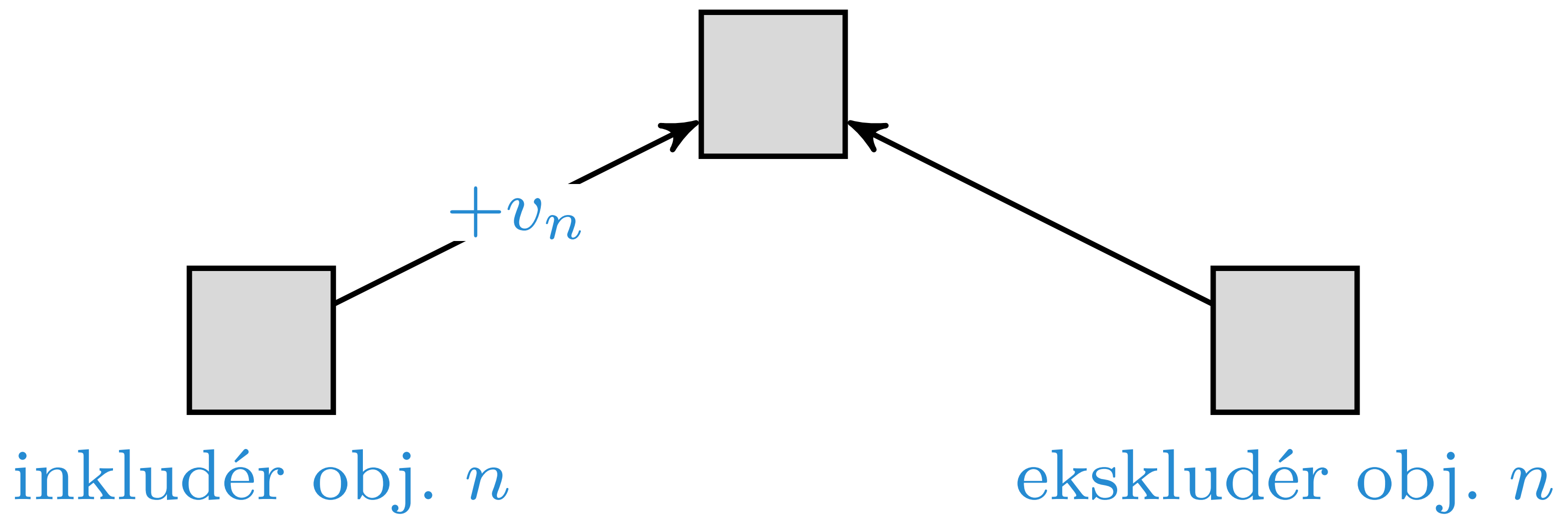
Fungerer ikke! En lengste vei kan ikke dekomponeres i lengste veier

Eksempel: Binær ryggsekk

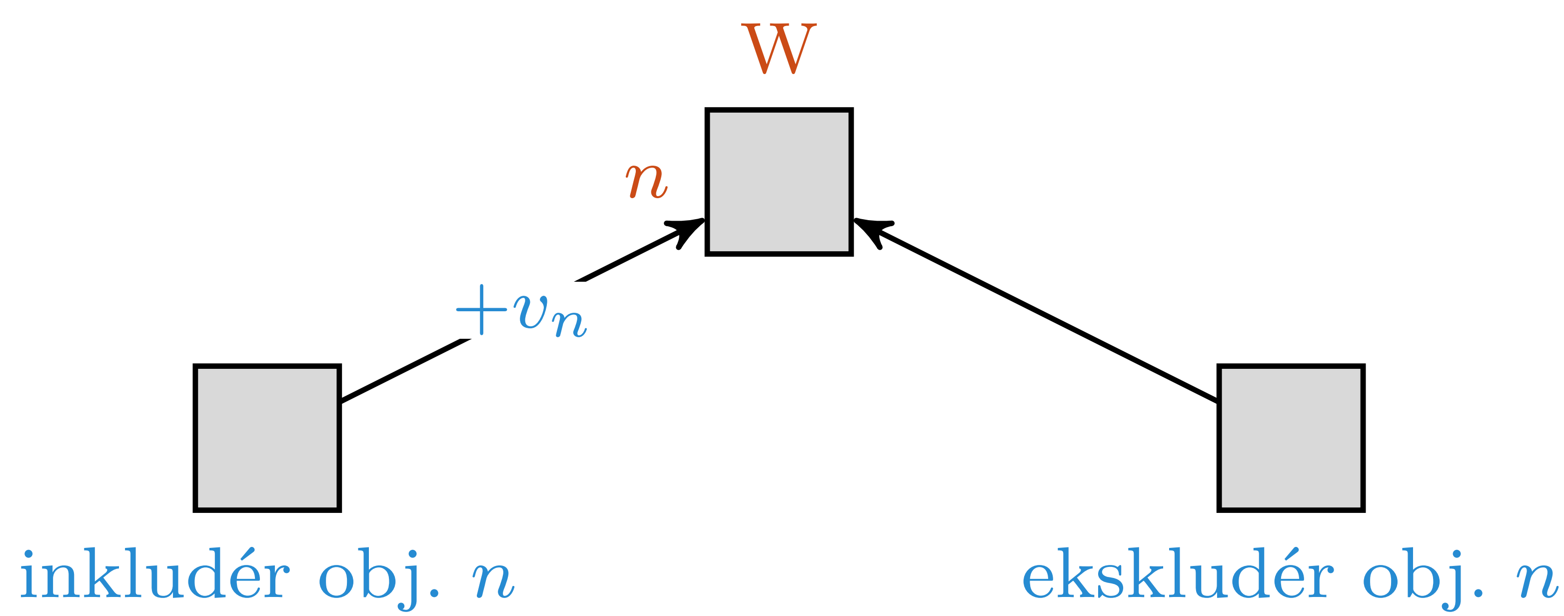
Input: Verdier $v_1, \ldots, v_n$, vekter $w_1, \ldots, w_n$ og en kapasitet W.

Output: Indekser $i_1, \ldots, i_k$ slik at $w_{i_1} + \cdots + w_{i_k} \leq$ W og totalverdien $v_{i_1} + \cdots + v_{i_k}$ er ma

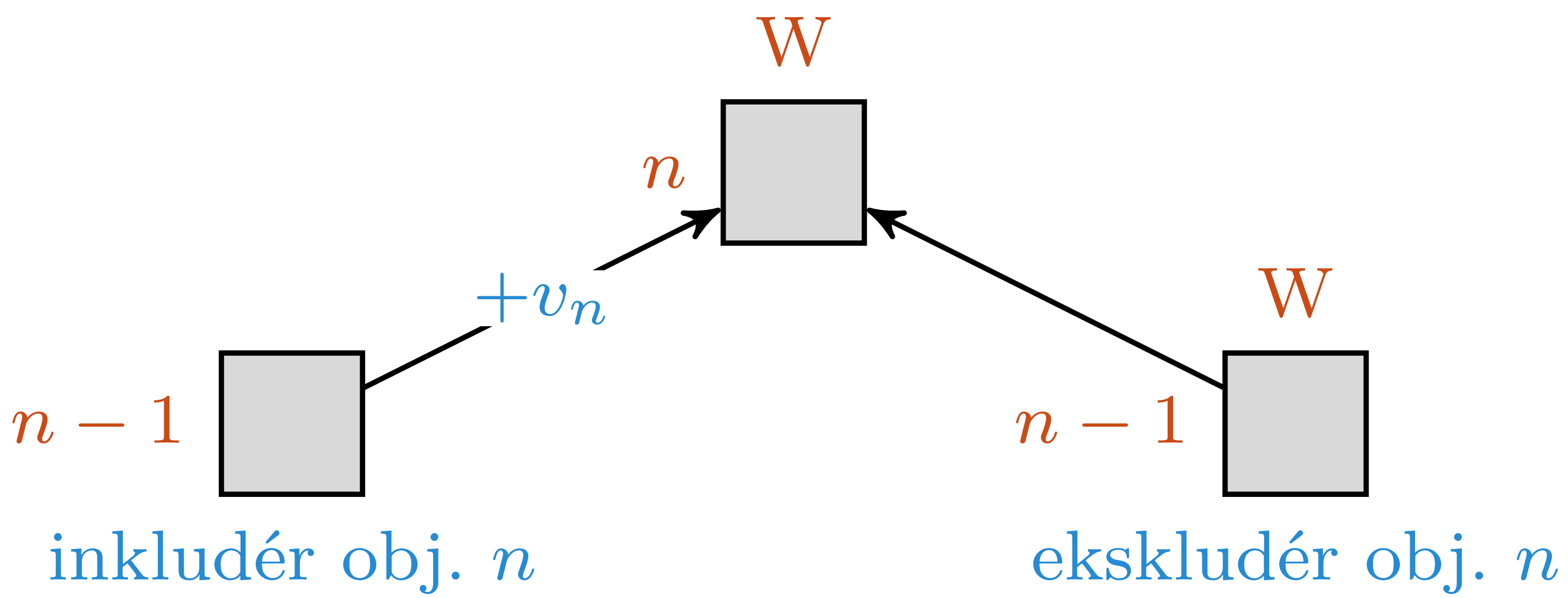




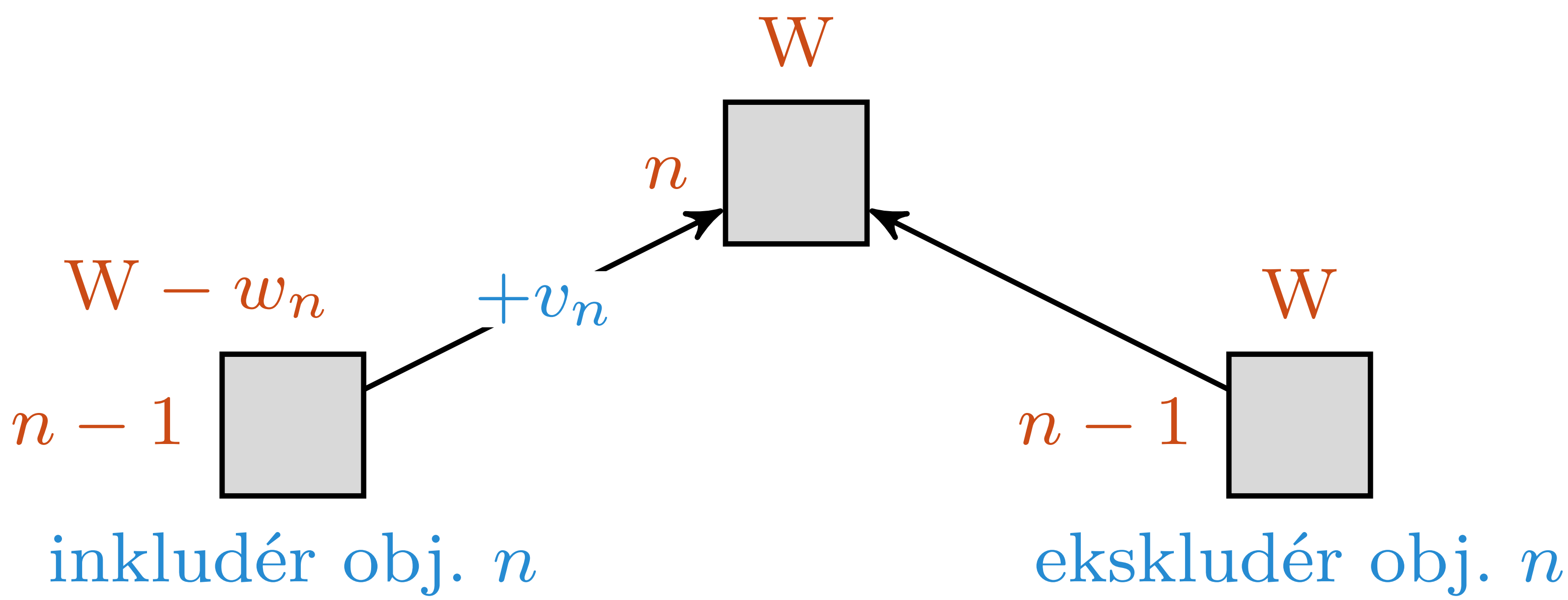
Objekt n bidrar med verdi v_n



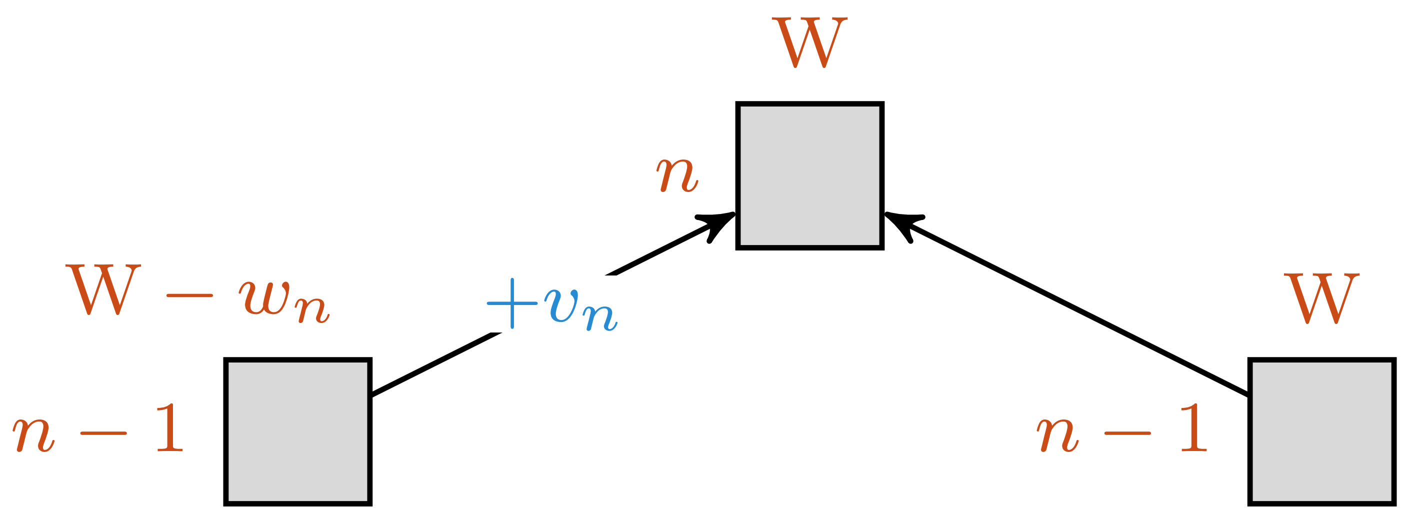
Vi parametriserer delproblemer vha. n og W

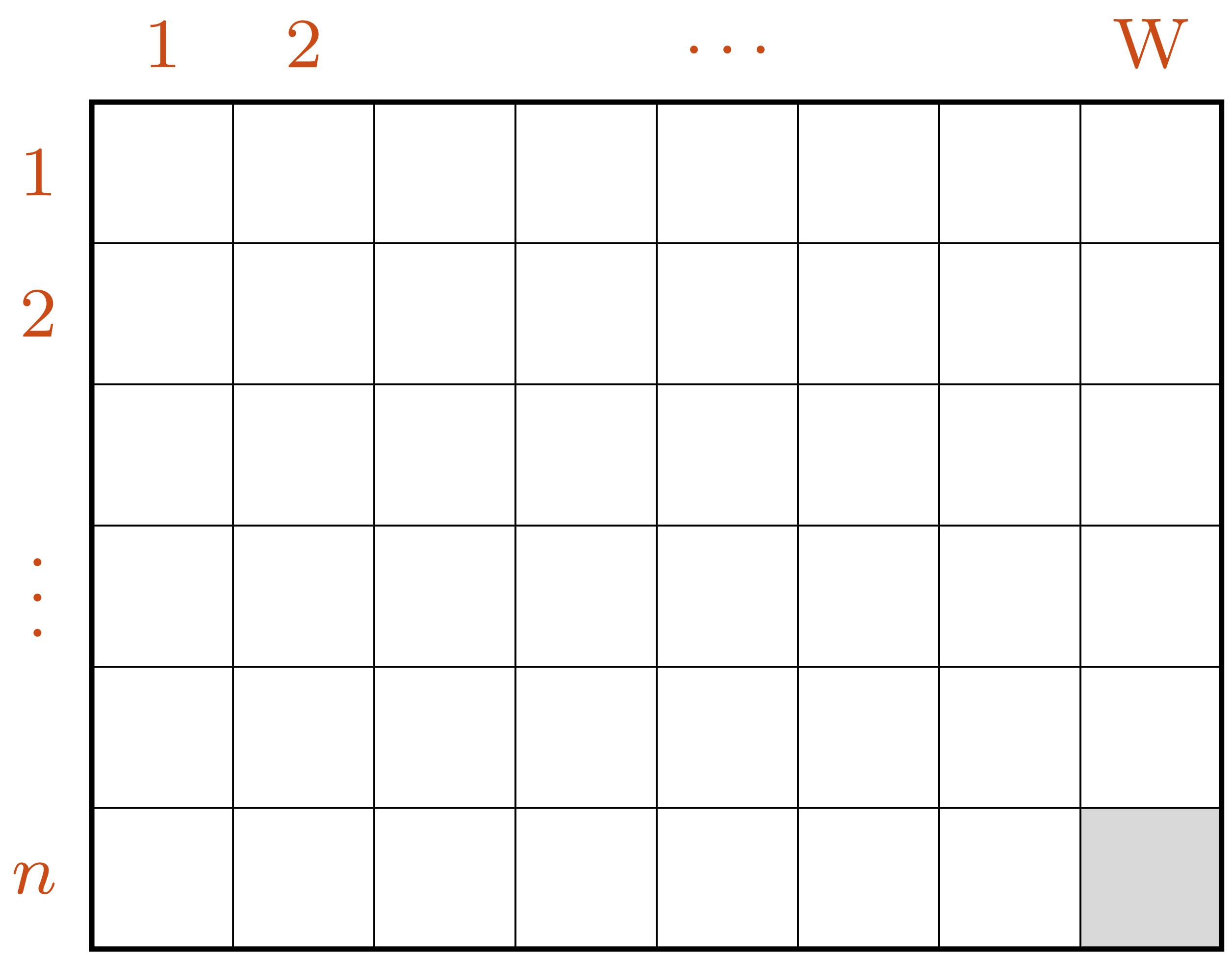


Ser nå bare på objekter $1 \ldots n-1$

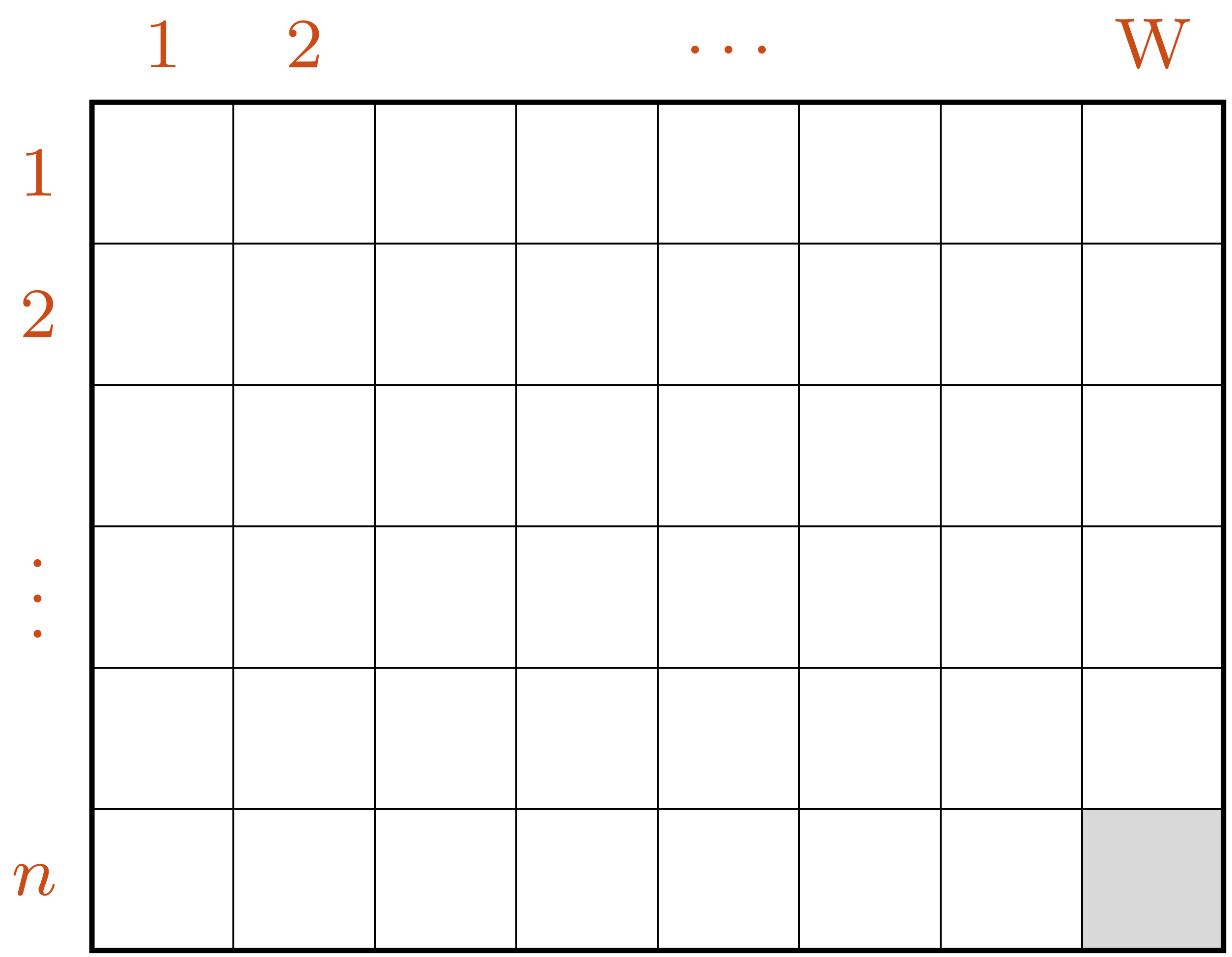


Objekt n bruker opp w_n av W

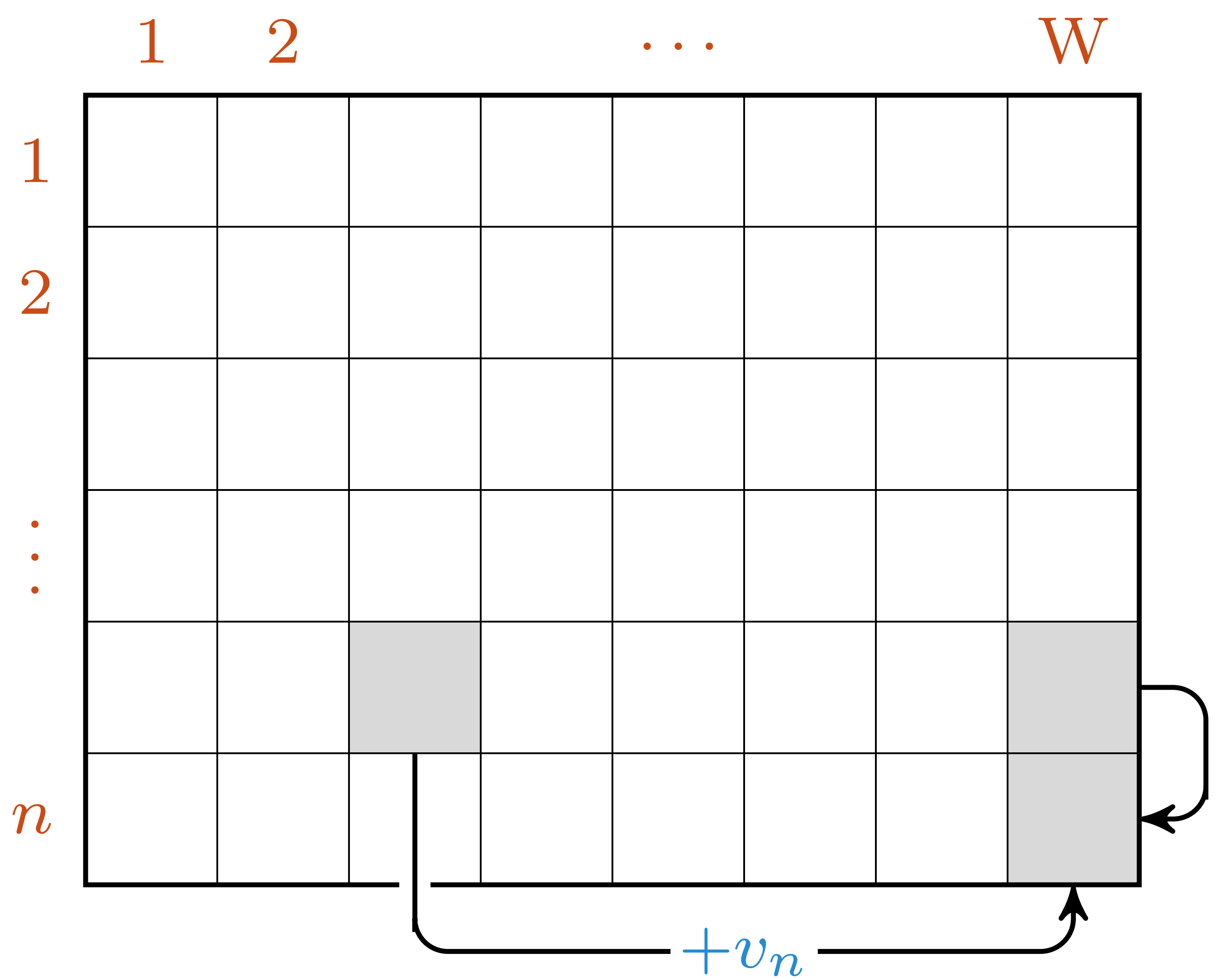

W
n
$W - w_n$
$+v_n$
$n-1$
W
$n-1$



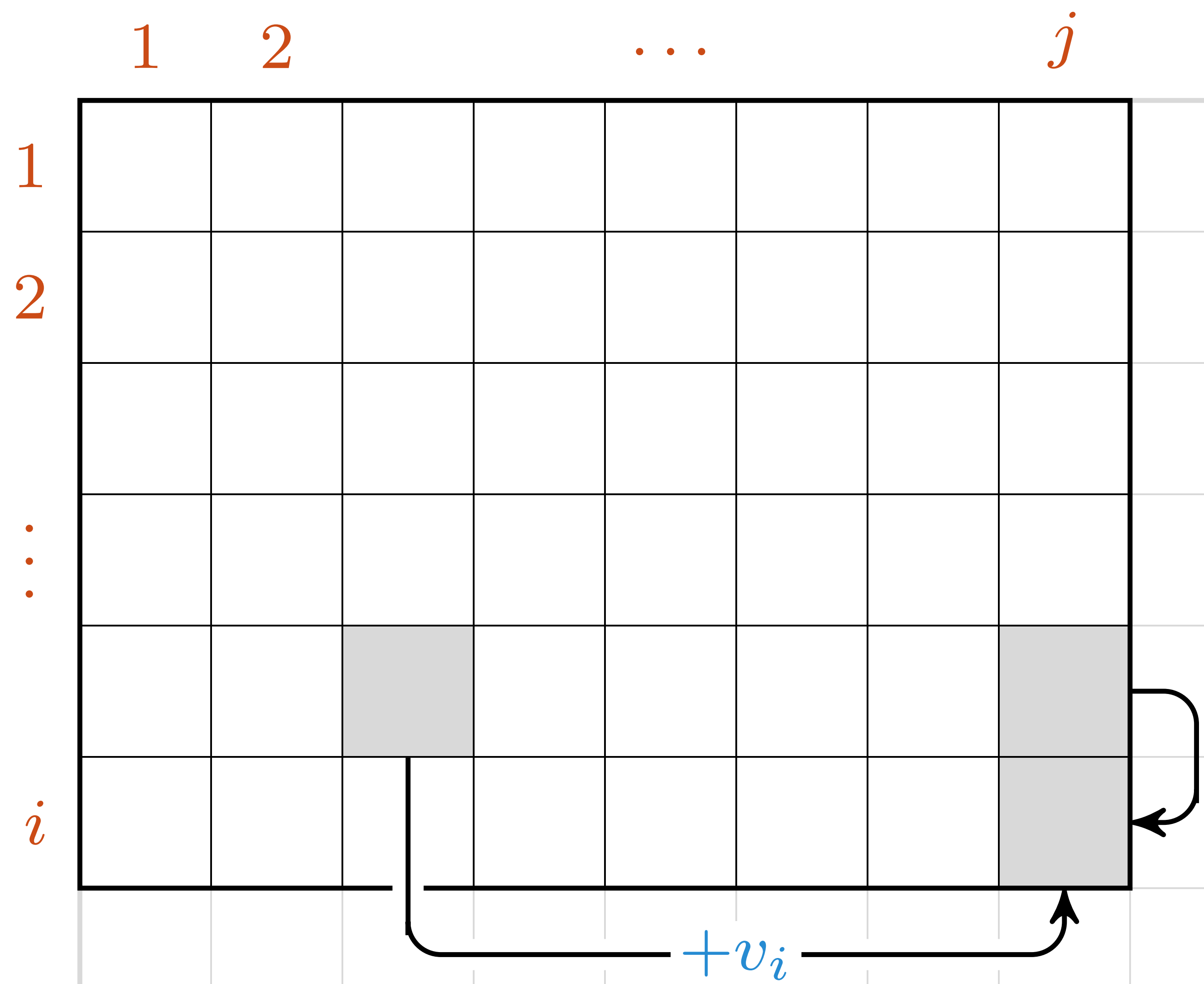
Lagre delløsninger i $n \times \mathrm{W}$-tabell



La f.eks. $w_n = 5$.



Dekomponering som før; kan løses radvis



Samme dekomponering for alle delproblemer

$\textsc{Knapsack}(n, W)$

n	antall
W	kapasitet

Utvalg: Maksimal total verdi, med vekt-kapasitet W

KNAPSACK(n, W)
1 let $\mathrm{K}[0\,..\,n, 0\,..\,\mathrm{W}]$ be a new array

n antall
W kapasitet
K memo

$\mathrm{K}[i, j]$ er optimum blant objekter 1 til i med kapasitet j

Knapsack(n, W)

let K$[0\,..\,n, 0\,..\,$W$]$ be a new array

for $j = 0$ **to** W

n	antall
W	kapasitet
K	memo
j	kapasitet

For hver mulig kapasitet . . .

$\textsc{Knapsack}(n, \mathrm{W})$

1 let $\mathrm{K}[0\,..\,n, 0\,..\mathrm{W}]$ be a new array
2 **for** $j = 0$ **to** W
3 $\quad \mathrm{K}[0, j] = 0$

n	antall
W	kapasitet
K	memo
j	kapasitet

Ingen objekter, ingen verdi

KNAPSACK(n, W)

let $\mathrm{K}[0\,..\,n, 0\,..\mathrm{W}]$ be a new array
for $j = 0$ **to** W
    $\mathrm{K}[0, j] = 0$
for $i = 1$ **to** n

n	antall
W	kapasitet
K	memo
i	objekt
j	kapasitet

Vurdér objekt i: Velg eller forkast?

Knapsack(n, W)

let $\mathrm{K}[0\,..\,n, 0\,..\,\mathrm{W}]$ be a new array
for $j = 0$ **to** W
    $\mathrm{K}[0, j] = 0$
for $i = 1$ **to** n
    for $j = 0$ **to** W

n	antall
W	kapasitet
K	memo
i	objekt
j	kapasitet

Anta at kapasiteten er j

Knapsack(n, W)

let K[0 . . n, 0 . . W] be a new array
for $j = 0$ **to** W
    K[0, j] = 0
for $i = 1$ **to** n
    for $j = 0$ **to** W
        $x = \mathrm{K}[i-1, j]$

n	antall
W	kapasitet
K	memo
i	objekt
j	kapasitet
x	uten i

Det beste vi fikk til med objekter 1 til $i-1$

Knapsack(n, W)

let $\mathrm{K}[0\,..\,n, 0\,..\,\mathrm{W}]$ be a new array
for $j = 0$ **to** W
    $\mathrm{K}[0, j] = 0$
for $i = 1$ **to** n
    for $j = 0$ **to** W
        $x = \mathrm{K}[i-1, j]$
        if $j < w_i$

n	antall
W	kapasitet
K	memo
i	objekt
j	kapasitet
w_i	vekt
x	uten i

Er objekt i tyngre enn hele kapasiteten?

```
KNAPSACK(n, W)
let K[0..n, 0..W] be a new array
for j = 0 to W
    K[0, j] = 0
for i = 1 to n
    for j = 0 to W
        x = K[i − 1, j]
        if j < w_i
            K[i, j] = x
```

n	antall
W	kapasitet
K	memo
i	objekt
j	kapasitet
w_i	vekt
x	uten i

Dropp objekt i, og behold forrige løsning

KNAPSACK(n, W)

let $K[0\,..\,n, 0\,..\,W]$ be a new array
for $j = 0$ **to** W
    $K[0, j] = 0$
for $i = 1$ **to** n
    for $j = 0$ **to** W
        $x = K[i-1, j]$
        if $j < w_i$
            $K[i, j] = x$
        else $y = K[i-1, j-w_i] + v_i$

n	antall
W	kapasitet
K	memo
i	objekt
j	kapasitet
w_i	vekt
v_i	verdi
x	uten i
y	med i

Ellers: Ta med i, og løs resten «rekursivt»

```
KNAPSACK(n, W)
let K[0..n, 0..W] be a new array
for j = 0 to W
    K[0, j] = 0
for i = 1 to n
    for j = 0 to W
        x = K[i − 1, j]
        if j < w_i
            K[i, j] = x
        else y = K[i − 1, j − w_i] + v_i
            K[i, j] = max(x, y)
```

n	antall
W	kapasitet
K	memo
i	objekt
j	kapasitet
w_i	vekt
v_i	verdi
x	uten i
y	med i

Velg det beste av å ta med i eller ikke

```
Knapsack(n, W)
let K[0..n, 0..W] be a new array
for j = 0 to W
    K[0, j] = 0
for i = 1 to n
    for j = 0 to W
        x = K[i − 1, j]
        if j < w_i
            K[i, j] = x
        else y = K[i − 1, j − w_i] + v_i
            K[i, j] = max(x, y)
```

	0	1	2	3	4	5
0						
1						
2						
3						
4						
5						
6						

w	v
1	1
2	5
1	4
3	3
1	2
2	6

$i, j, x, y \; = \; -, -, -, -$

```
KNAPSACK(n, W)
let K[0 . . n, 0 . . W] be a new array
for j = 0 to W
    K[0, j] = 0
for i = 1 to n
    for j = 0 to W
        x = K[i − 1, j]
        if j < w_i
            K[i, j] = x
        else y = K[i − 1, j − w_i] + v_i
            K[i, j] = max(x, y)
```

$i, j, x, y \;=\;$ -, -, -, -

	0	1	2	3	4	5
0	0	0	0	0	0	0
1						
2						
3						
4						
5						
6						

w	v
1	1
2	5
1	4
3	3
1	2
2	6

```
Knapsack(n, W)
let K[0..n, 0..W] be a new array
for j = 0 to W
    K[0, j] = 0
for i = 1 to n
    for j = 0 to W
        x = K[i − 1, j]
        if j < w_i
            K[i, j] = x
        else y = K[i − 1, j − w_i] + v_i
            K[i, j] = max(x, y)
```

$i, j, x, y \;=\; 1, \text{-}, \text{-}, \text{-}$

	0	1	2	3	4	5
0	0	0	0	0	0	0
1						
2						
3						
4						
5						
6						

w	v
1	1
2	5
1	4
3	3
1	2
2	6

```
Knapsack(n, W)
let K[0..n, 0..W] be a new array
for j = 0 to W
    K[0, j] = 0
for i = 1 to n
    for j = 0 to W
        x = K[i − 1, j]
        if j < w_i
            K[i, j] = x
        else y = K[i − 1, j − w_i] + v_i
            K[i, j] = max(x, y)
```

	0	1	2	3	4	5
0	0	0	0	0	0	0
1						
2						
3						
4						
5						
6						

w	v
1	1
2	5
1	4
3	3
1	2
2	6

$i, j, x, y \;=\; 1, 0, \text{-}, \text{-}$

```
KNAPSACK(n, W)
let K[0..n, 0..W] be a new array
for j = 0 to W
        K[0, j] = 0
for i = 1 to n
        for j = 0 to W
                x = K[i − 1, j]
                if j < w_i
                        K[i, j] = x
                else y = K[i − 1, j − w_i] + v_i
                        K[i, j] = max(x, y)
```

	0	1	2	3	4	5
0	0	0	0	0	0	0
1						
2						
3						
4						
5						
6						

w	v
1	1
2	5
1	4
3	3
1	2
2	6

$i, j, x, y \;=\; 1, 0, 0, \text{-}$

```
KNAPSACK(n, W)
let K[0..n, 0..W] be a new array
for j = 0 to W
    K[0, j] = 0
for i = 1 to n
    for j = 0 to W
        x = K[i − 1, j]
        if j < w_i
            K[i, j] = x
        else y = K[i − 1, j − w_i] + v_i
            K[i, j] = max(x, y)
```

$i, j, x, y \;=\; 1, 0, 0, -$

	0	1	2	3	4	5
0	0	0	0	0	0	0
1						
2						
3						
4						
5						
6						

w	v
1	1
2	5
1	4
3	3
1	2
2	6

```
Knapsack(n, W)
let K[0 . . n, 0 . . W] be a new array
for j = 0 to W
    K[0, j] = 0
for i = 1 to n
    for j = 0 to W
        x = K[i − 1, j]
        if j < w_i
            K[i, j] = x
        else y = K[i − 1, j − w_i] + v_i
            K[i, j] = max(x, y)
```

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0					
2						
3						
4						
5						
6						

w	v
1	1
2	5
1	4
3	3
1	2
2	6

$i, j, x, y \;=\; 1, 1, \text{-}, \text{-}$

```
KNAPSACK(n, W)
let K[0 . . n, 0 . . W] be a new array
for j = 0 to W
    K[0, j] = 0
for i = 1 to n
    for j = 0 to W
        x = K[i − 1, j]
        if j < w_i
            K[i, j] = x
        else y = K[i − 1, j − w_i] + v_i
            K[i, j] = max(x, y)
```

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0					
2						
3						
4						
5						
6						

w	v
1	1
2	5
1	4
3	3
1	2
2	6

$i, j, x, y \;=\; 1, 1, 0, -$

```
KNAPSACK(n, W)
let K[0..n, 0..W] be a new array
for j = 0 to W
    K[0, j] = 0
for i = 1 to n
    for j = 0 to W
        x = K[i − 1, j]
        if j < w_i
            K[i, j] = x
        else y = K[i − 1, j − w_i] + v_i
            K[i, j] = max(x, y)
```

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0					
2						
3						
4						
5						
6						

w	v
1	1
2	5
1	4
3	3
1	2
2	6

$i, j, x, y \;=\; 1, 1, 0, -$

Knapsack(n, W)

```
let K[0 . . n, 0 . . W] be a new array
for j = 0 to W
    K[0, j] = 0
for i = 1 to n
    for j = 0 to W
        x = K[i − 1, j]
        if j < w_i
            K[i, j] = x
        else y = K[i − 1, j − w_i] + v_i
            K[i, j] = max(x, y)
```

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0					
2						
3						
4						
5						
6						

w	v
1	1
2	5
1	4
3	3
1	2
2	6

$i, j, x, y \ = \ 1, 1, 0, 1$

```
KNAPSACK(n, W)
let K[0..n, 0..W] be a new array
for j = 0 to W
    K[0, j] = 0
for i = 1 to n
    for j = 0 to W
        x = K[i − 1, j]
        if j < w_i
            K[i, j] = x
        else y = K[i − 1, j − w_i] + v_i
            K[i, j] = max(x, y)
```

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	1				
2						
3						
4						
5						
6						

w	v
1	1
2	5
1	4
3	3
1	2
2	6

$i, j, x, y \;=\; 1, 2, \text{-}, \text{-}$

```
KNAPSACK(n, W)
let K[0..n, 0..W] be a new array
for j = 0 to W
    K[0, j] = 0
for i = 1 to n
    for j = 0 to W
        x = K[i − 1, j]
        if j < w_i
            K[i, j] = x
        else y = K[i − 1, j − w_i] + v_i
            K[i, j] = max(x, y)
```

$i, j, x, y \;=\; 2, 2, \text{-}, \text{-}$

	0	1	2	3	4	5	w	v
0	0	0	0	0	0	0		
1	0	1	1	1	1	1	1	1
2	0	1					2	5
3							1	4
4							3	3
5							1	2
6							2	6

KNAPSACK(n, W)

let $\mathrm{K}[0\,..\,n, 0\,..\,\mathrm{W}]$ be a new array
for $j = 0$ **to** W
    $\mathrm{K}[0, j] = 0$
for $i = 1$ **to** n
    for $j = 0$ **to** W
        $x = \mathrm{K}[i-1, j]$
        if $j < w_i$
            $\mathrm{K}[i, j] = x$
        else $y = \mathrm{K}[i-1, j-w_i] + v_i$
            $\mathrm{K}[i, j] = \max(x, y)$

$i, j, x, y \;=\; 2, 2, 1, -$

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	1	1	1	1	1
2	0	1				
3						
4						
5						
6						

w	v
1	1
2	5
1	4
3	3
1	2
2	6

```
KNAPSACK(n, W)
let K[0..n, 0..W] be a new array
for j = 0 to W
    K[0, j] = 0
for i = 1 to n
    for j = 0 to W
        x = K[i − 1, j]
        if j < w_i
            K[i, j] = x
        else y = K[i − 1, j − w_i] + v_i
            K[i, j] = max(x, y)
```

$i, j, x, y \;=\; 2, 2, 1, \text{-}$

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	1	1	1	1	1
2	0	1				
3						
4						
5						
6						

w	v
1	1
2	5
1	4
3	3
1	2
2	6

```
KNAPSACK(n, W)
let K[0..n, 0..W] be a new array
for j = 0 to W
        K[0, j] = 0
for i = 1 to n
        for j = 0 to W
                x = K[i - 1, j]
                if j < w_i
                        K[i, j] = x
                else y = K[i - 1, j - w_i] + v_i
                        K[i, j] = max(x, y)
```

	0	1	2	3	4	5	w	v
0	0	0	0	0	0	0		
1	0	1	1	1	1	1	1	1
2	0	1					2	5
3							1	4
4							3	3
5							1	2
6							2	6

$i, j, x, y \ = \ 2, 2, 1, 5$

```
Knapsack(n, W)
let K[0..n, 0..W] be a new array
for j = 0 to W
    K[0, j] = 0
for i = 1 to n
    for j = 0 to W
        x = K[i − 1, j]
        if j < w_i
            K[i, j] = x
        else y = K[i − 1, j − w_i] + v_i
            K[i, j] = max(x, y)
```

	0	1	2	3	4	5		w	v
0	0	0	0	0	0	0			
1	0	1	1	1	1	1		1	1
2	0	1	5					2	5
3								1	4
4								3	3
5								1	2
6								2	6

$i, j, x, y \;=\; 2, 3, \text{-}, \text{-}$

```
Knapsack(n, W)
let K[0..n, 0..W] be a new array
for j = 0 to W
    K[0, j] = 0
for i = 1 to n
    for j = 0 to W
        x = K[i − 1, j]
        if j < w_i
            K[i, j] = x
        else y = K[i − 1, j − w_i] + v_i
            K[i, j] = max(x, y)
```

$i, j, x, y \;=\; 2, 3, 1, \text{-}$

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	1	1	1	1	1
2	0	1	5			
3						
4						
5						
6						

w	v
1	1
2	5
1	4
3	3
1	2
2	6

```
KNAPSACK(n, W)
let K[0..n, 0..W] be a new array
for j = 0 to W
    K[0, j] = 0
for i = 1 to n
    for j = 0 to W
        x = K[i − 1, j]
        if j < w_i
            K[i, j] = x
        else y = K[i − 1, j − w_i] + v_i
            K[i, j] = max(x, y)
```

	0	1	2	3	4	5	w	v
0	0	0	0	0	0	0		
1	0	1	1	1	1	1	1	1
2	0	1	5				2	5
3							1	4
4							3	3
5							1	2
6							2	6

$i, j, x, y \;=\; 2, 3, 1, -$

```
KNAPSACK(n, W)
let K[0..n, 0..W] be a new array
for j = 0 to W
    K[0, j] = 0
for i = 1 to n
    for j = 0 to W
        x = K[i − 1, j]
        if j < w_i
            K[i, j] = x
        else y = K[i − 1, j − w_i] + v_i
            K[i, j] = max(x, y)
```

$i, j, x, y \;=\; 2, 3, 1, 6$

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	1	1	1	1	1
2	0	1	5			
3						
4						
5						
6						

w	v
1	1
2	5
1	4
3	3
1	2
2	6

```
KNAPSACK(n, W)
let K[0..n, 0..W] be a new array
for j = 0 to W
        K[0, j] = 0
for i = 1 to n
        for j = 0 to W
                x = K[i − 1, j]
                if j < w_i
                        K[i, j] = x
                else y = K[i − 1, j − w_i] + v_i
                        K[i, j] = max(x, y)
```

	0	1	2	3	4	5	w	v
0	0	0	0	0	0	0		
1	0	1	1	1	1	1	1	1
2	0	1	5	6			2	5
3							1	4
4							3	3
5							1	2
6							2	6

$i, j, x, y \;=\; 2, 4, \text{-}, \text{-}$

```
KNAPSACK(n, W)
let K[0..n, 0..W] be a new array
for j = 0 to W
    K[0, j] = 0
for i = 1 to n
    for j = 0 to W
        x = K[i − 1, j]
        if j < w_i
            K[i, j] = x
        else y = K[i − 1, j − w_i] + v_i
            K[i, j] = max(x, y)
```

$i, j, x, y \;=\; 2, 4, 1, \text{-}$

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	1	1	1	1	1
2	0	1	5	6		
3						
4						
5						
6						

w	v
1	1
2	5
1	4
3	3
1	2
2	6

```
KNAPSACK(n, W)
let K[0..n, 0..W] be a new array
for j = 0 to W
        K[0, j] = 0
for i = 1 to n
        for j = 0 to W
                x = K[i − 1, j]
                if j < w_i
                        K[i, j] = x
                else y = K[i − 1, j − w_i] + v_i
                        K[i, j] = max(x, y)
```

$i, j, x, y \; = \; 2, 4, 1, \text{-}$

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	1	1	1	1	1
2	0	1	5	6		
3						
4						
5						
6						

w	v
1	1
2	5
1	4
3	3
1	2
2	6

```
Knapsack(n, W)
let K[0..n, 0..W] be a new array
for j = 0 to W
    K[0, j] = 0
for i = 1 to n
    for j = 0 to W
        x = K[i − 1, j]
        if j < w_i
            K[i, j] = x
        else y = K[i − 1, j − w_i] + v_i
            K[i, j] = max(x, y)
```

$i, j, x, y \;=\; 2, 4, 1, 6$

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	1	1	1	1	1
2	0	1	5	6		
3						
4						
5						
6						

w	v
1	1
2	5
1	4
3	3
1	2
2	6

```
KNAPSACK(n, W)
let K[0..n, 0..W] be a new array
for j = 0 to W
    K[0, j] = 0
for i = 1 to n
    for j = 0 to W
        x = K[i − 1, j]
        if j < w_i
            K[i, j] = x
        else y = K[i − 1, j − w_i] + v_i
            K[i, j] = max(x, y)
```

$i, j, x, y \;=\; 2, 5, \text{-}, \text{-}$

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	1	1	1	1	1
2	0	1	5	6	6	
3						
4						
5						
6						

w	v
1	1
2	5
1	4
3	3
1	2
2	6

```
KNAPSACK(n, W)
let K[0..n, 0..W] be a new array
for j = 0 to W
    K[0, j] = 0
for i = 1 to n
    for j = 0 to W
        x = K[i − 1, j]
        if j < w_i
            K[i, j] = x
        else y = K[i − 1, j − w_i] + v_i
            K[i, j] = max(x, y)
```

$i, j, x, y \;=\; 2, 5, 1, \text{-}$

	0	1	2	3	4	5	w	v
0	0	0	0	0	0	0		
1	0	1	1	1	1	1	1	1
2	0	1	5	6	6		2	5
3							1	4
4							3	3
5							1	2
6							2	6

```
KNAPSACK(n, W)
let K[0 . . n, 0 . . W] be a new array
for j = 0 to W
        K[0, j] = 0
for i = 1 to n
        for j = 0 to W
                x = K[i − 1, j]
                if j < w_i
                        K[i, j] = x
                else y = K[i − 1, j − w_i] + v_i
                        K[i, j] = max(x, y)
```

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	1	1	1	1	1
2	0	1	5	6	6	
3						
4						
5						
6						

w	v
1	1
2	5
1	4
3	3
1	2
2	6

$i, j, x, y \;=\; 2, 5, 1, \text{-}$

```
KNAPSACK(n, W)
let K[0..n, 0..W] be a new array
for j = 0 to W
    K[0, j] = 0
for i = 1 to n
    for j = 0 to W
        x = K[i − 1, j]
        if j < w_i
            K[i, j] = x
        else y = K[i − 1, j − w_i] + v_i
            K[i, j] = max(x, y)
```

	0	1	2	3	4	5	w	v
0	0	0	0	0	0	0		
1	0	1	1	1	1	1	1	1
2	0	1	5	6	6		2	5
3							1	4
4							3	3
5							1	2
6							2	6

$i, j, x, y \;=\; 2, 5, 1, 6$

```
KNAPSACK(n, W)
let K[0..n, 0..W] be a new array
for j = 0 to W
    K[0, j] = 0
for i = 1 to n
    for j = 0 to W
        x = K[i − 1, j]
        if j < w_i
            K[i, j] = x
        else y = K[i − 1, j − w_i] + v_i
            K[i, j] = max(x, y)
```

$i, j, x, y \;=\; 3, 0, \text{-}, \text{-}$

	0	1	2	3	4	5	w	v
0	0	0	0	0	0	0		
1	0	1	1	1	1	1	1	1
2	0	1	5	6	6	6	2	5
3							1	4
4							3	3
5							1	2
6							2	6

```
KNAPSACK(n, W)
let K[0..n, 0..W] be a new array
for j = 0 to W
    K[0, j] = 0
for i = 1 to n
    for j = 0 to W
        x = K[i − 1, j]
        if j < w_i
            K[i, j] = x
        else y = K[i − 1, j − w_i] + v_i
            K[i, j] = max(x, y)
```

$i, j, x, y \;=\; 3, 0, 0, \text{-}$

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	1	1	1	1	1
2	0	1	5	6	6	6
3						
4						
5						
6						

w	v
1	1
2	5
1	4
3	3
1	2
2	6

```
KNAPSACK(n, W)
let K[0..n, 0..W] be a new array
for j = 0 to W
    K[0, j] = 0
for i = 1 to n
    for j = 0 to W
        x = K[i − 1, j]
        if j < w_i
            K[i, j] = x
        else y = K[i − 1, j − w_i] + v_i
            K[i, j] = max(x, y)
```

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	1	1	1	1	1
2	0	1	5	6	6	6
3						
4						
5						
6						

w	v
1	1
2	5
1	4
3	3
1	2
2	6

$i, j, x, y \;=\; 3, 0, 0, \text{-}$

```
KNAPSACK(n, W)
let K[0 . . n, 0 . . W] be a new array
for j = 0 to W
    K[0, j] = 0
for i = 1 to n
    for j = 0 to W
        x = K[i − 1, j]
        if j < w_i
            K[i, j] = x
        else y = K[i − 1, j − w_i] + v_i
            K[i, j] = max(x, y)
```

$i, j, x, y \;=\; 3, 1, \text{-}, \text{-}$

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	1	1	1	1	1
2	0	1	5	6	6	6
3	0					
4						
5						
6						

w	v
1	1
2	5
1	4
3	3
1	2
2	6

```
KNAPSACK(n, W)
let K[0..n, 0..W] be a new array
for j = 0 to W
    K[0, j] = 0
for i = 1 to n
    for j = 0 to W
        x = K[i − 1, j]
        if j < w_i
            K[i, j] = x
        else y = K[i − 1, j − w_i] + v_i
            K[i, j] = max(x, y)
```

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	1	1	1	1	1
2	0	1	5	6	6	6
3	0					
4						
5						
6						

w	v
1	1
2	5
1	4
3	3
1	2
2	6

$i, j, x, y \;=\; 3, 1, 1, -$

```
KNAPSACK(n, W)
let K[0..n, 0..W] be a new array
for j = 0 to W
        K[0, j] = 0
for i = 1 to n
        for j = 0 to W
                x = K[i − 1, j]
                if j < w_i
                        K[i, j] = x
                else y = K[i − 1, j − w_i] + v_i
                        K[i, j] = max(x, y)
```

	0	1	2	3	4	5		w	v
0	0	0	0	0	0	0			
1	0	1	1	1	1	1		1	1
2	0	1	5	6	6	6		2	5
3	0							1	4
4								3	3
5								1	2
6								2	6

$i, j, x, y \;=\; 3, 1, 1, \text{-}$

```
KNAPSACK(n, W)
let K[0..n, 0..W] be a new array
for j = 0 to W
    K[0, j] = 0
for i = 1 to n
    for j = 0 to W
        x = K[i - 1, j]
        if j < w_i
            K[i, j] = x
        else y = K[i - 1, j - w_i] + v_i
            K[i, j] = max(x, y)
```

$i, j, x, y \;=\; 3, 1, 1, 4$

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	1	1	1	1	1
2	0	1	5	6	6	6
3	0					
4						
5						
6						

w	v
1	1
2	5
1	4
3	3
1	2
2	6

```
KNAPSACK(n, W)
let K[0..n, 0..W] be a new array
for j = 0 to W
    K[0, j] = 0
for i = 1 to n
    for j = 0 to W
        x = K[i − 1, j]
        if j < w_i
            K[i, j] = x
        else y = K[i − 1, j − w_i] + v_i
            K[i, j] = max(x, y)
```

$i, j, x, y \;=\; 3, 2, \text{-}, \text{-}$

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	1	1	1	1	1
2	0	1	5	6	6	6
3	0	4				
4						
5						
6						

w	v
1	1
2	5
1	4
3	3
1	2
2	6

```
KNAPSACK(n, W)
let K[0 . . n, 0 . . W] be a new array
for j = 0 to W
    K[0, j] = 0
for i = 1 to n
    for j = 0 to W
        x = K[i − 1, j]
        if j < w_i
            K[i, j] = x
        else y = K[i − 1, j − w_i] + v_i
            K[i, j] = max(x, y)
```

$i, j, x, y \;=\; 3, 2, 5, \text{-}$

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	1	1	1	1	1
2	0	1	5	6	6	6
3	0	4				
4						
5						
6						

w	v
1	1
2	5
1	4
3	3
1	2
2	6

```
KNAPSACK(n, W)
let K[0..n, 0..W] be a new array
for j = 0 to W
    K[0, j] = 0
for i = 1 to n
    for j = 0 to W
        x = K[i − 1, j]
        if j < w_i
            K[i, j] = x
        else y = K[i − 1, j − w_i] + v_i
            K[i, j] = max(x, y)
```

$i, j, x, y \;=\; 3, 2, 5, \text{-}$

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	1	1	1	1	1
2	0	1	5	6	6	6
3	0	4				
4						
5						
6						

w	v
1	1
2	5
1	4
3	3
1	2
2	6

```
KNAPSACK(n, W)
let K[0..n, 0..W] be a new array
for j = 0 to W
    K[0, j] = 0
for i = 1 to n
    for j = 0 to W
        x = K[i − 1, j]
        if j < w_i
            K[i, j] = x
        else y = K[i − 1, j − w_i] + v_i
            K[i, j] = max(x, y)
```

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	1	1	1	1	1
2	0	1	5	6	6	6
3	0	4				
4						
5						
6						

w	v
1	1
2	5
1	4
3	3
1	2
2	6

$i, j, x, y \;=\; 3, 2, 5, 5$

```
KNAPSACK(n, W)
let K[0..n, 0..W] be a new array
for j = 0 to W
    K[0, j] = 0
for i = 1 to n
    for j = 0 to W
        x = K[i − 1, j]
        if j < w_i
            K[i, j] = x
        else y = K[i − 1, j − w_i] + v_i
            K[i, j] = max(x, y)
```

$i, j, x, y \;=\; 3, 3, \text{-}, \text{-}$

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	1	1	1	1	1
2	0	1	5	6	6	6
3	0	4	5			
4						
5						
6						

w	v
1	1
2	5
1	4
3	3
1	2
2	6

Knapsack(n, W)

```
let K[0..n, 0..W] be a new array
for j = 0 to W
    K[0, j] = 0
for i = 1 to n
    for j = 0 to W
        x = K[i - 1, j]
        if j < w_i
            K[i, j] = x
        else y = K[i - 1, j - w_i] + v_i
            K[i, j] = max(x, y)
```

$i, j, x, y \;=\; 3, 3, 6, \text{-}$

	0	1	2	3	4	5	w	v
0	0	0	0	0	0	0		
1	0	1	1	1	1	1	1	1
2	0	1	5	6	6	6	2	5
3	0	4	5				1	4
4							3	3
5							1	2
6							2	6

```
KNAPSACK(n, W)
let K[0..n, 0..W] be a new array
for j = 0 to W
    K[0, j] = 0
for i = 1 to n
    for j = 0 to W
        x = K[i − 1, j]
        if j < w_i
            K[i, j] = x
        else y = K[i − 1, j − w_i] + v_i
            K[i, j] = max(x, y)
```

$i, j, x, y \;=\; 3, 3, 6, -$

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	1	1	1	1	1
2	0	1	5	6	6	6
3	0	4	5			
4						
5						
6						

w	v
1	1
2	5
1	4
3	3
1	2
2	6

```
KNAPSACK(n, W)
let K[0..n, 0..W] be a new array
for j = 0 to W
    K[0, j] = 0
for i = 1 to n
    for j = 0 to W
        x = K[i − 1, j]
        if j < w_i
            K[i, j] = x
        else y = K[i − 1, j − w_i] + v_i
            K[i, j] = max(x, y)
```

$i, j, x, y \;=\; 3, 3, 6, 9$

	0	1	2	3	4	5	w	v
0	0	0	0	0	0	0		
1	0	1	1	1	1	1	1	1
2	0	1	5	6	6	6	2	5
3	0	4	5				1	4
4							3	3
5							1	2
6							2	6

```
KNAPSACK(n, W)
let K[0..n, 0..W] be a new array
for j = 0 to W
    K[0, j] = 0
for i = 1 to n
    for j = 0 to W
        x = K[i − 1, j]
        if j < w_i
            K[i, j] = x
        else y = K[i − 1, j − w_i] + v_i
            K[i, j] = max(x, y)
```

	0	1	2	3	4	5	w	v
0	0	0	0	0	0	0		
1	0	1	1	1	1	1	1	1
2	0	1	5	6	6	6	2	5
3	0	4	5	9			1	4
4							3	3
5							1	2
6							2	6

$i, j, x, y \;=\; 3, 4, \text{-}, \text{-}$

```
KNAPSACK(n, W)
let K[0..n, 0..W] be a new array
for j = 0 to W
    K[0, j] = 0
for i = 1 to n
    for j = 0 to W
        x = K[i − 1, j]
        if j < w_i
            K[i, j] = x
        else y = K[i − 1, j − w_i] + v_i
            K[i, j] = max(x, y)
```

$i, j, x, y \;=\; 6, 3, \text{-}, \text{-}$

	0	1	2	3	4	5	w	v
0	0	0	0	0	0	0		
1	0	1	1	1	1	1	1	1
2	0	1	5	6	6	6	2	5
3	0	4	5	9	10	10	1	4
4	0	4	5	9	10	10	3	3
5	0	4	6	9	11	12	1	2
6	0	4	6				2	6

```
KNAPSACK(n, W)
let K[0..n, 0..W] be a new array
for j = 0 to W
    K[0, j] = 0
for i = 1 to n
    for j = 0 to W
        x = K[i − 1, j]
        if j < w_i
            K[i, j] = x
        else y = K[i − 1, j − w_i] + v_i
            K[i, j] = max(x, y)
```

$i, j, x, y \ = \ 6, 3, 9, -$

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	1	1	1	1	1
2	0	1	5	6	6	6
3	0	4	5	9	10	10
4	0	4	5	9	10	10
5	0	4	6	9	11	12
6	0	4	6			

w	v
1	1
2	5
1	4
3	3
1	2
2	6

```
Knapsack(n, W)
let K[0..n, 0..W] be a new array
for j = 0 to W
    K[0, j] = 0
for i = 1 to n
    for j = 0 to W
        x = K[i − 1, j]
        if j < w_i
            K[i, j] = x
        else y = K[i − 1, j − w_i] + v_i
            K[i, j] = max(x, y)
```

$i, j, x, y \;=\; 6, 3, 9, \text{-}$

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	1	1	1	1	1
2	0	1	5	6	6	6
3	0	4	5	9	10	10
4	0	4	5	9	10	10
5	0	4	6	9	11	12
6	0	4	6			

w	v
1	1
2	5
1	4
3	3
1	2
2	6

```
Knapsack(n, W)
let K[0..n, 0..W] be a new array
for j = 0 to W
    K[0, j] = 0
for i = 1 to n
    for j = 0 to W
        x = K[i − 1, j]
        if j < w_i
            K[i, j] = x
        else y = K[i − 1, j − w_i] + v_i
            K[i, j] = max(x, y)
```

$i, j, x, y \;=\; 6, 3, 9, 10$

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	1	1	1	1	1
2	0	1	5	6	6	6
3	0	4	5	9	10	10
4	0	4	5	9	10	10
5	0	4	6	9	11	12
6	0	4	6			

w	v
1	1
2	5
1	4
3	3
1	2
2	6

```
KNAPSACK(n, W)
let K[0..n, 0..W] be a new array
for j = 0 to W
    K[0, j] = 0
for i = 1 to n
    for j = 0 to W
        x = K[i − 1, j]
        if j < w_i
            K[i, j] = x
        else y = K[i − 1, j − w_i] + v_i
            K[i, j] = max(x, y)
```

$i, j, x, y \;=\; 6, 4, \text{-}, \text{-}$

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	1	1	1	1	1
2	0	1	5	6	6	6
3	0	4	5	9	10	10
4	0	4	5	9	10	10
5	0	4	6	9	11	12
6	0	4	6	10		

w	v
1	1
2	5
1	4
3	3
1	2
2	6

```
Knapsack(n, W)
let K[0..n, 0..W] be a new array
for j = 0 to W
    K[0, j] = 0
for i = 1 to n
    for j = 0 to W
        x = K[i − 1, j]
        if j < w_i
            K[i, j] = x
        else y = K[i − 1, j − w_i] + v_i
            K[i, j] = max(x, y)
```

$i, j, x, y \;=\; 6, 4, 11, -$

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	1	1	1	1	1
2	0	1	5	6	6	6
3	0	4	5	9	10	10
4	0	4	5	9	10	10
5	0	4	6	9	11	12
6	0	4	6	10		

w	v
1	1
2	5
1	4
3	3
1	2
2	6

```
Knapsack(n, W)
let K[0..n, 0..W] be a new array
for j = 0 to W
    K[0, j] = 0
for i = 1 to n
    for j = 0 to W
        x = K[i − 1, j]
        if j < w_i
            K[i, j] = x
        else y = K[i − 1, j − w_i] + v_i
            K[i, j] = max(x, y)
```

	0	1	2	3	4	5	w	v
0	0	0	0	0	0	0		
1	0	1	1	1	1	1	1	1
2	0	1	5	6	6	6	2	5
3	0	4	5	9	10	10	1	4
4	0	4	5	9	10	10	3	3
5	0	4	6	9	11	12	1	2
6	0	4	6	10			2	6

$i, j, x, y \;=\; 6, 4, 11, -$

```
KNAPSACK(n, W)
let K[0 . . n, 0 . . W] be a new array
for j = 0 to W
    K[0, j] = 0
for i = 1 to n
    for j = 0 to W
        x = K[i − 1, j]
        if j < w_i
            K[i, j] = x
        else y = K[i − 1, j − w_i] + v_i
            K[i, j] = max(x, y)
```

	0	1	2	3	4	5	w	v
0	0	0	0	0	0	0		
1	0	1	1	1	1	1	1	1
2	0	1	5	6	6	6	2	5
3	0	4	5	9	10	10	1	4
4	0	4	5	9	10	10	3	3
5	0	4	6	9	11	12	1	2
6	0	4	6	10			2	6

$i, j, x, y \ = \ 6, 4, 11, 12$

Knapsack(n, W)

```
let K[0 . . n, 0 . . W] be a new array
for j = 0 to W
        K[0, j] = 0
for i = 1 to n
        for j = 0 to W
                x = K[i − 1, j]
                if j < w_i
                        K[i, j] = x
                else y = K[i − 1, j − w_i] + v_i
                        K[i, j] = max(x, y)
```

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	1	1	1	1	1
2	0	1	5	6	6	6
3	0	4	5	9	10	10
4	0	4	5	9	10	10
5	0	4	6	9	11	12
6	0	4	6	10	12	

w	v
1	1
2	5
1	4
3	3
1	2
2	6

$i, j, x, y \;=\; 6, 5, \text{-}, \text{-}$

```
KNAPSACK(n, W)
let K[0 . . n, 0 . . W] be a new array
for j = 0 to W
    K[0, j] = 0
for i = 1 to n
    for j = 0 to W
        x = K[i − 1, j]
        if j < w_i
            K[i, j] = x
        else y = K[i − 1, j − w_i] + v_i
            K[i, j] = max(x, y)
```

$i, j, x, y \;=\; 6, 5, 12, \text{-}$

	0	1	2	3	4	5	w	v
0	0	0	0	0	0	0		
1	0	1	1	1	1	1	1	1
2	0	1	5	6	6	6	2	5
3	0	4	5	9	10	10	1	4
4	0	4	5	9	10	10	3	3
5	0	4	6	9	11	12	1	2
6	0	4	6	10	12		2	6

```
KNAPSACK(n, W)
let K[0..n, 0..W] be a new array
for j = 0 to W
    K[0, j] = 0
for i = 1 to n
    for j = 0 to W
        x = K[i − 1, j]
        if j < w_i
            K[i, j] = x
        else y = K[i − 1, j − w_i] + v_i
            K[i, j] = max(x, y)
```

	0	1	2	3	4	5	w	v
0	0	0	0	0	0	0		
1	0	1	1	1	1	1	1	1
2	0	1	5	6	6	6	2	5
3	0	4	5	9	10	10	1	4
4	0	4	5	9	10	10	3	3
5	0	4	6	9	11	12	1	2
6	0	4	6	10	12		2	6

$i, j, x, y \;=\; 6, 5, 12, \text{–}$

```
KNAPSACK(n, W)
let K[0..n, 0..W] be a new array
for j = 0 to W
    K[0, j] = 0
for i = 1 to n
    for j = 0 to W
        x = K[i − 1, j]
        if j < w_i
            K[i, j] = x
        else y = K[i − 1, j − w_i] + v_i
            K[i, j] = max(x, y)
```

	0	1	2	3	4	5	w	v
0	0	0	0	0	0	0		
1	0	1	1	1	1	1	1	1
2	0	1	5	6	6	6	2	5
3	0	4	5	9	10	10	1	4
4	0	4	5	9	10	10	3	3
5	0	4	6	9	11	12	1	2
6	0	4	6	10	12		2	6

$i, j, x, y \ = \ 6, 5, 12, 15$

Svaret er altså i siste rute, K[6,5], dvs., 15.

```
KNAPSACK(n, W)
let K[0..n, 0..W] be a new array
for j = 0 to W
    K[0, j] = 0
for i = 1 to n
    for j = 0 to W
        x = K[i − 1, j]
        if j < w_i
            K[i, j] = x
        else y = K[i − 1, j − w_i] + v_i
            K[i, j] = max(x, y)
```

$i, j, x, y \ = \ -, -, -, -$

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	1	1	1	1	1
2	0	1	5	6	6	6
3	0	4	5	9	10	10
4	0	4	5	9	10	10
5	0	4	6	9	11	12
6	0	4	6	10	12	15

w	v
1	1
2	5
1	4
3	3
1	2
2	6

Vi kan spore oss tilbake til hvilke elementer som er med på samme måte som i LCS, hvis vi tar vare på valget som gjøres av max(x,y) i hver iterasjon.

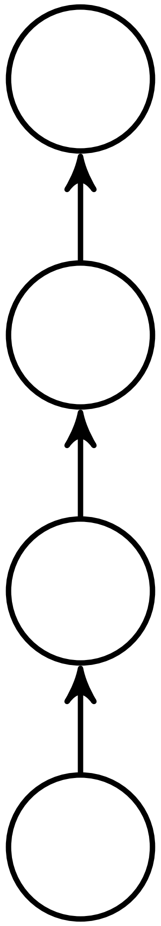
Forelesning 7

Grådighet

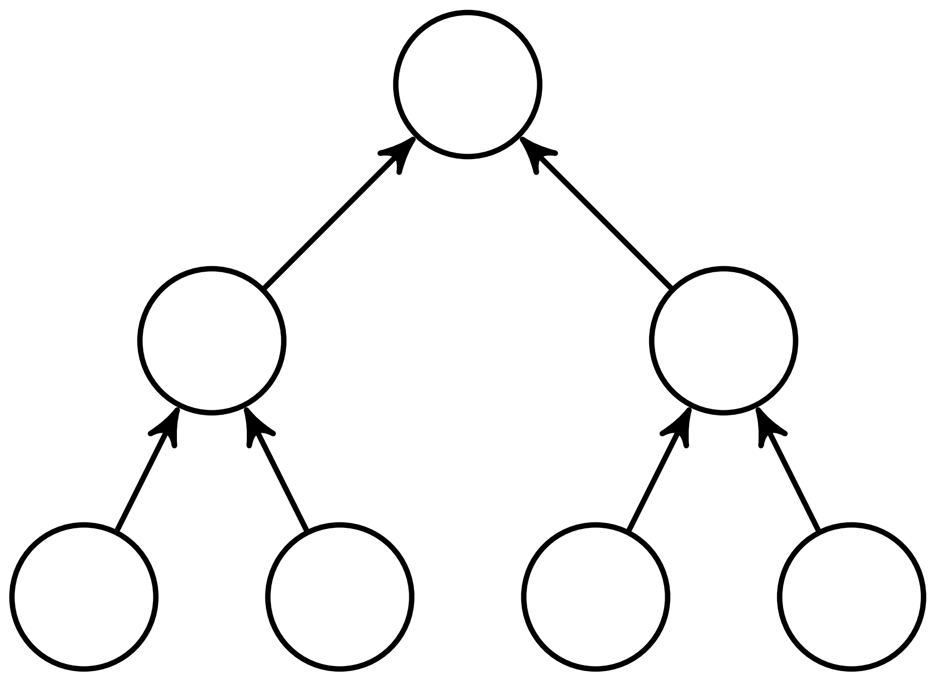
1. **Grådighet › hva er det?**
2. **Eksempel: Ryggsekk**
3. **Eksempel: Aktivitetsutvalg**
4. **Eksempel: Huffman**

1:4

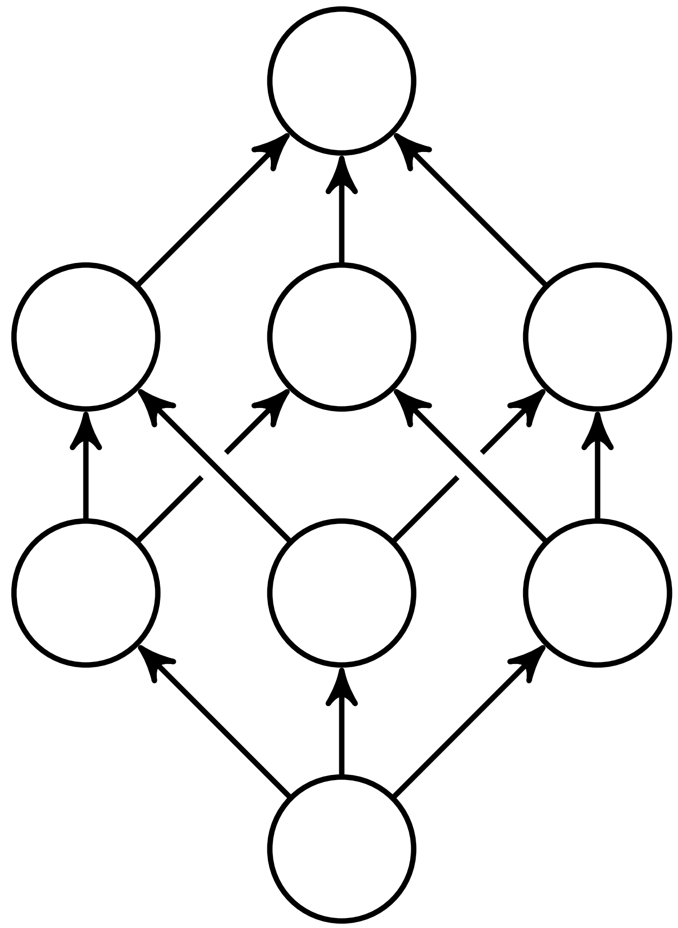
Grådighet › Hva er det?



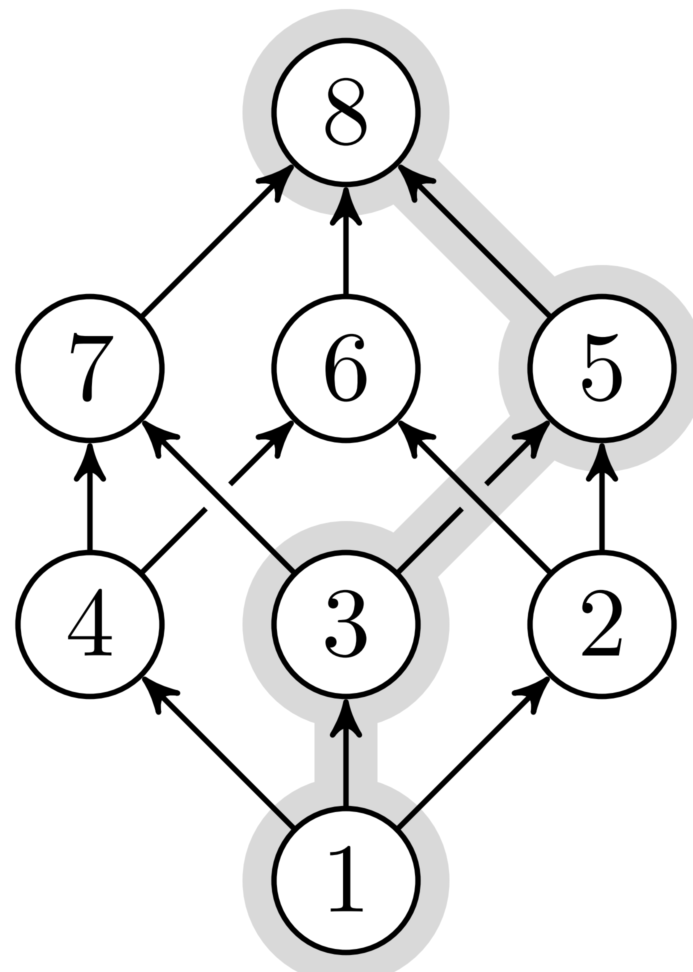
Inkrementell design



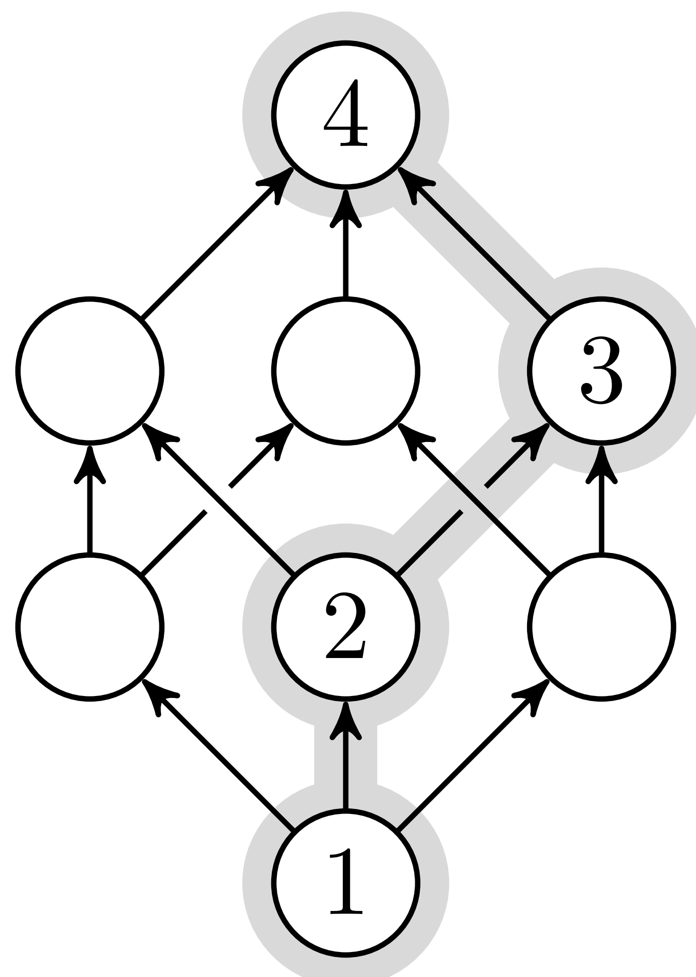
Uavhengige delproblemer: Splitt og hersk



Overlappende delproblemer: Dynamisk programmering

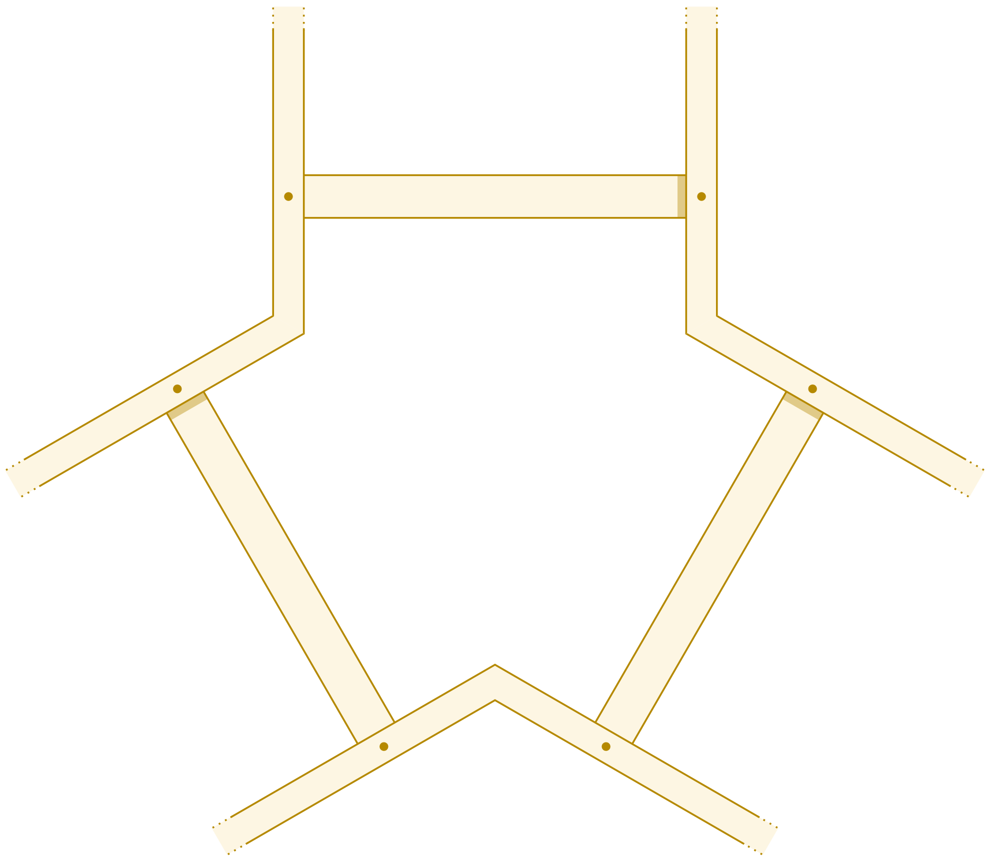


Hvis DP tar valg: Løs delproblemer først

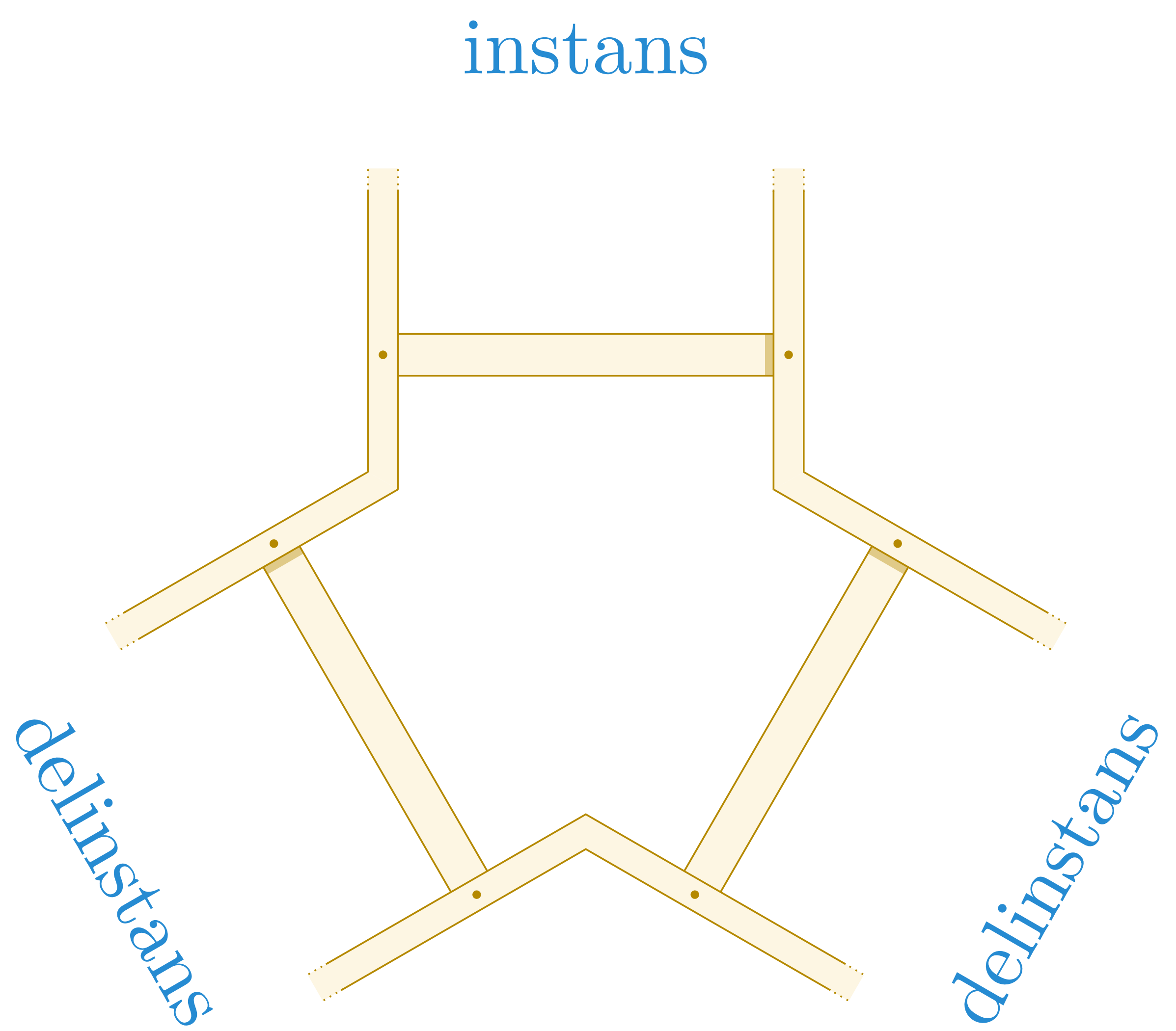


Grådighet: Velg med én gang!

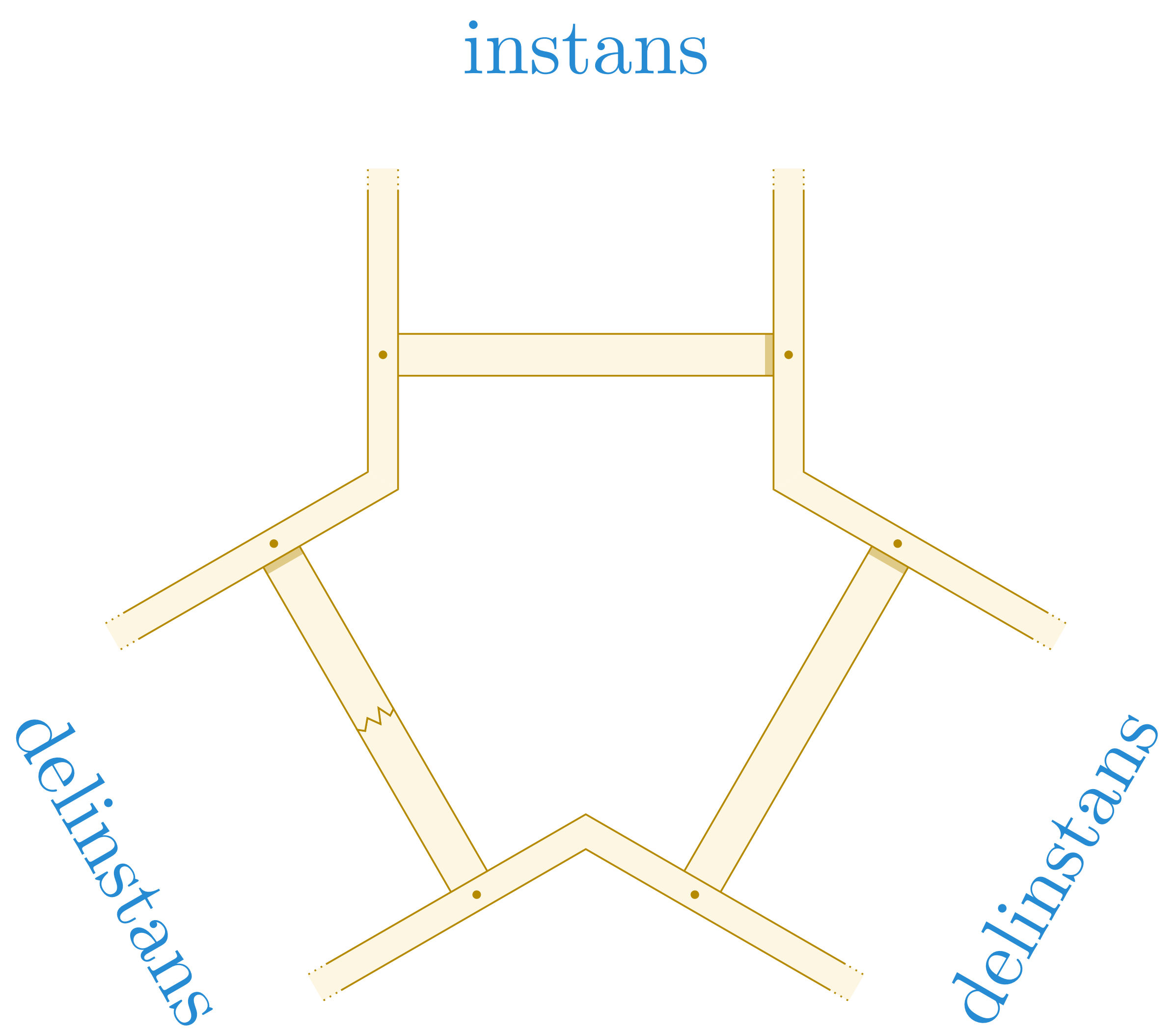
instans



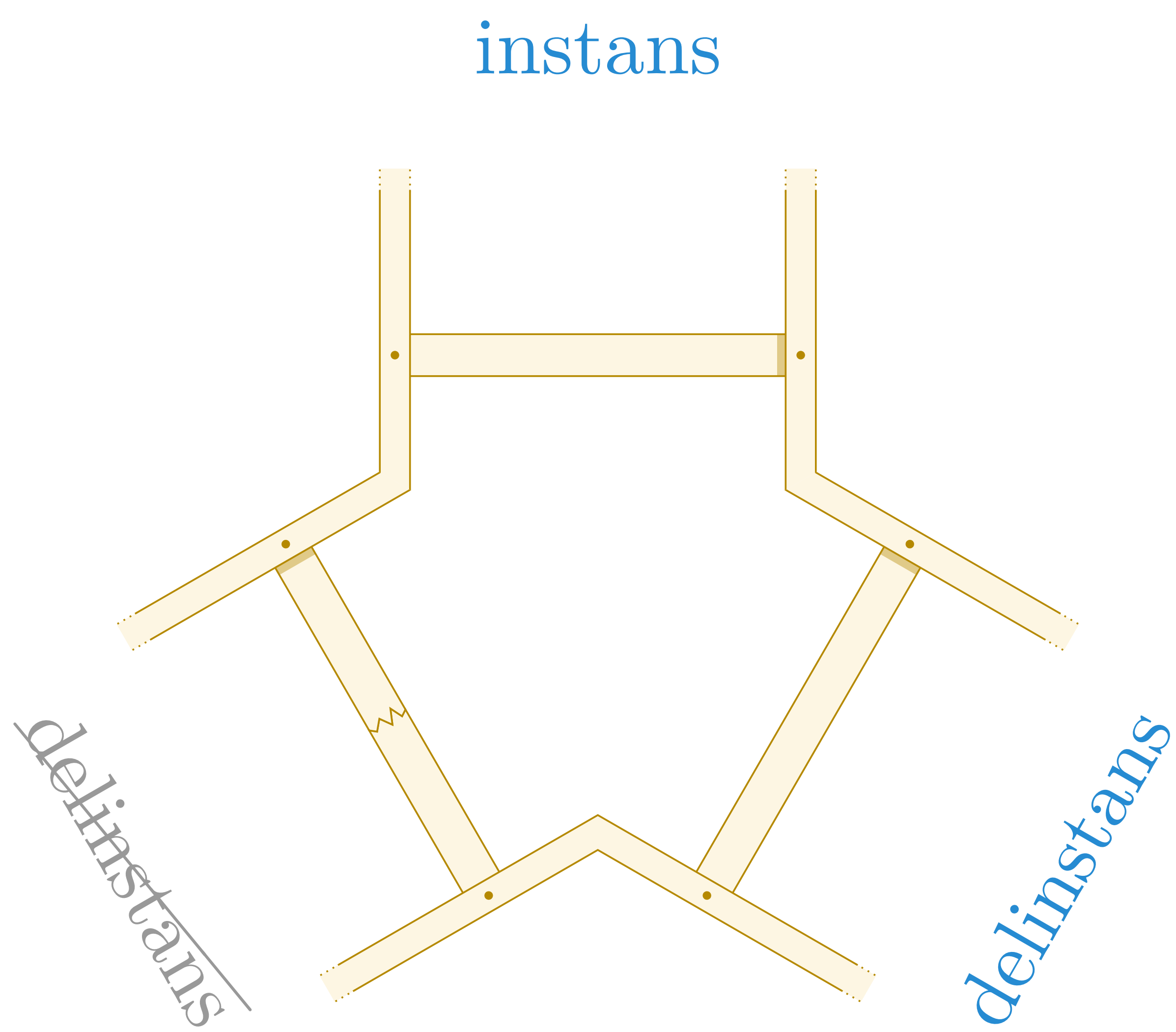
Vi bryter instansen ned i delinstanser



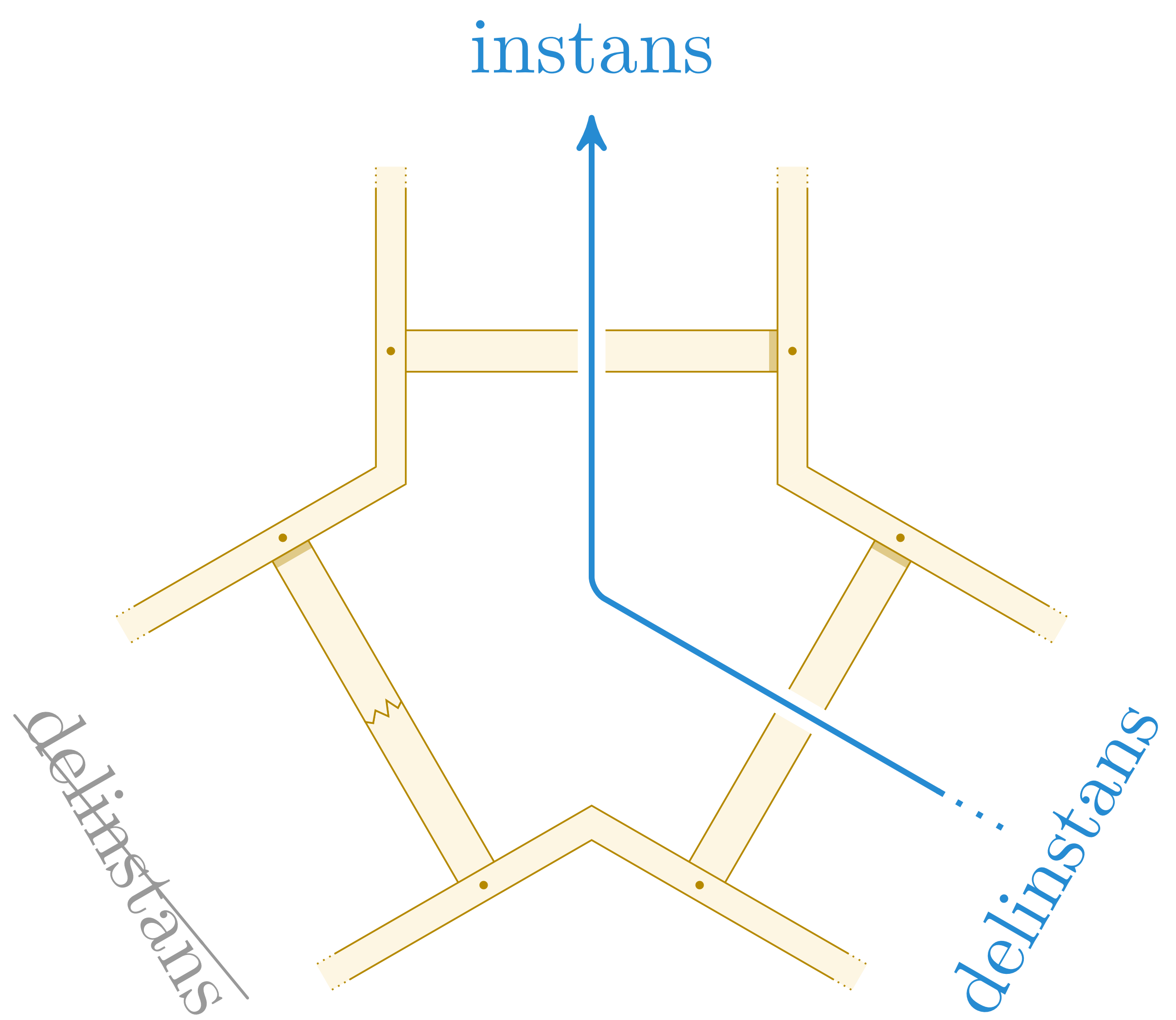
Optimal løsning bygger på én av delløsningene



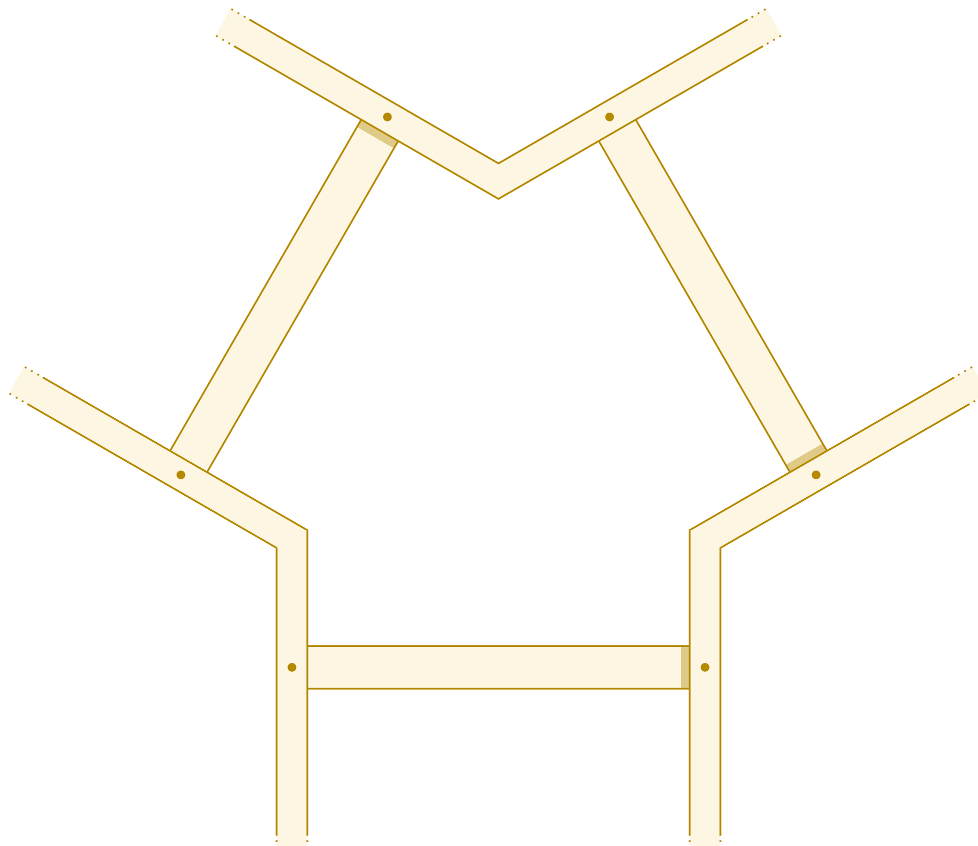
Før vi løser noen: Hvilken delinstans er mest lovende?



Vi løser kun den mest lovende, og baserer oss på den!

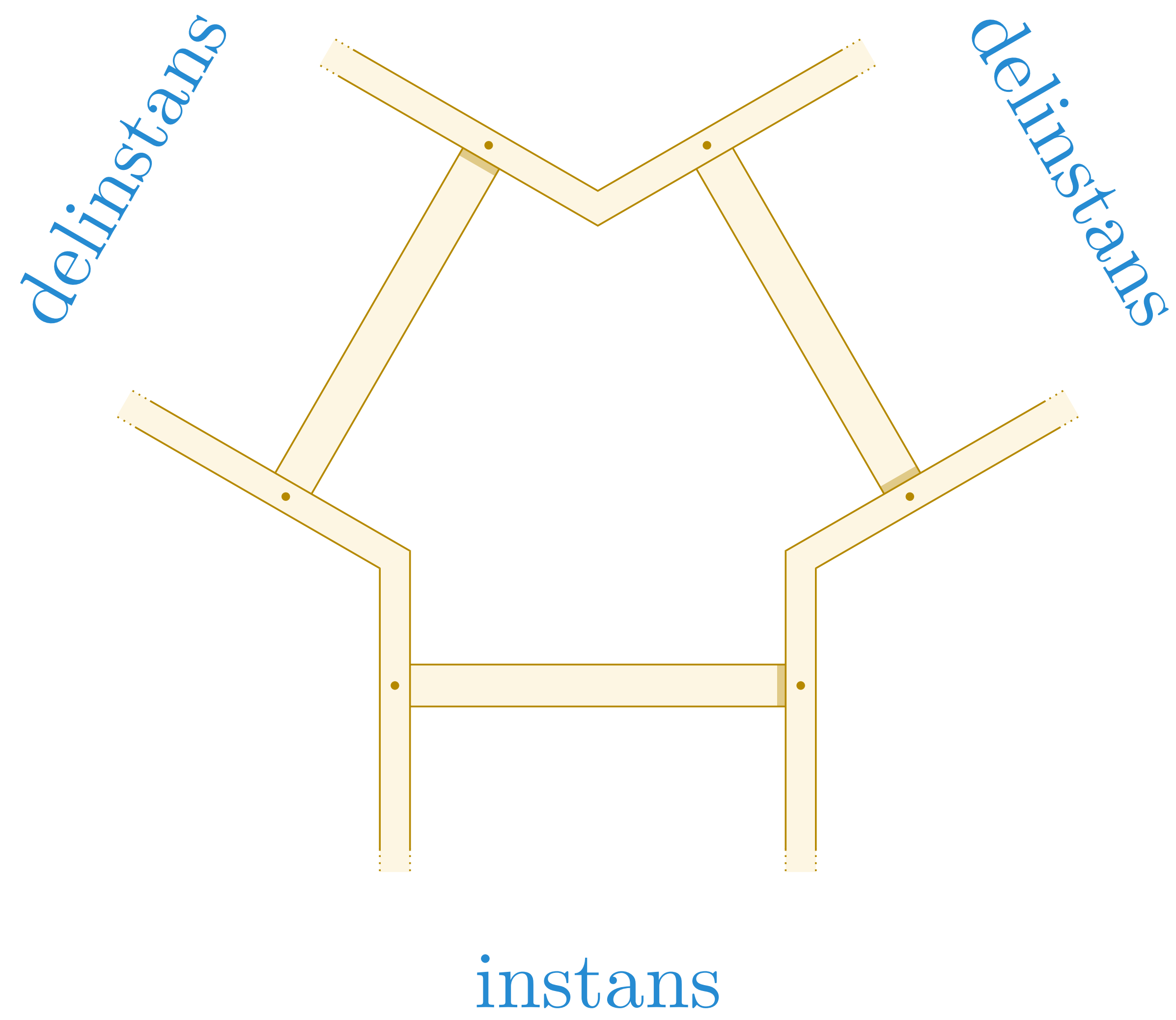


Vi løser kun den mest lovende, og baserer oss på den!

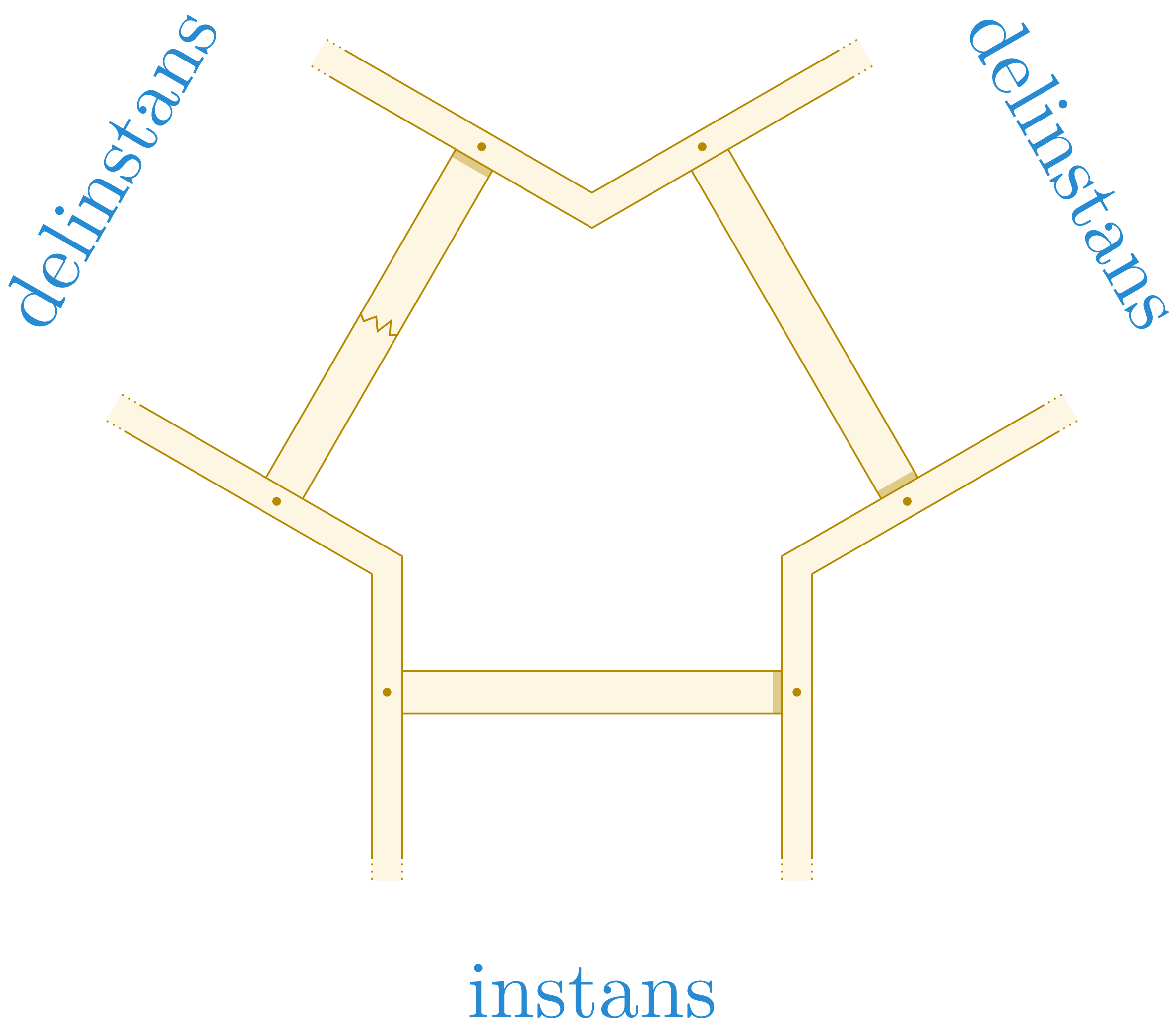


instans

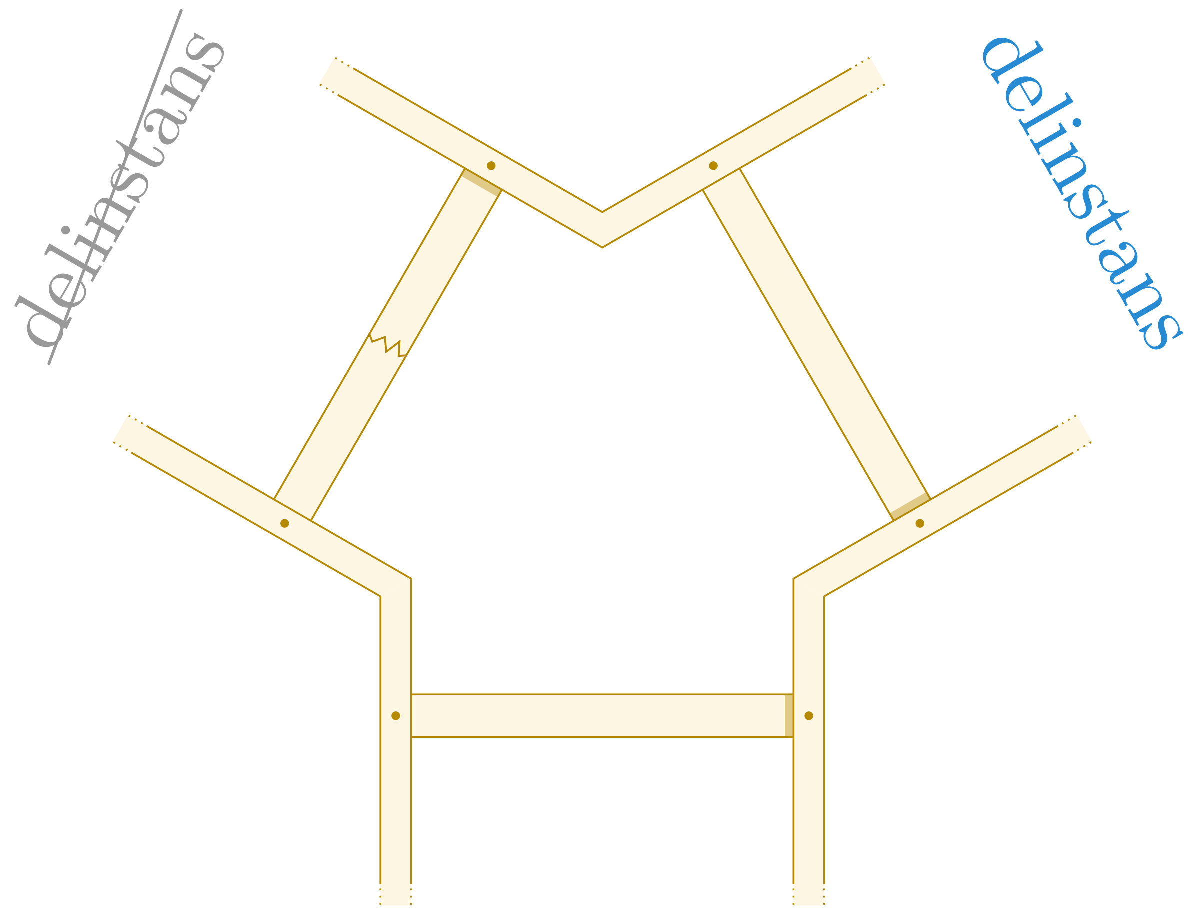
Annet perspektiv: Se på «veien videre» heller enn «veien hit»



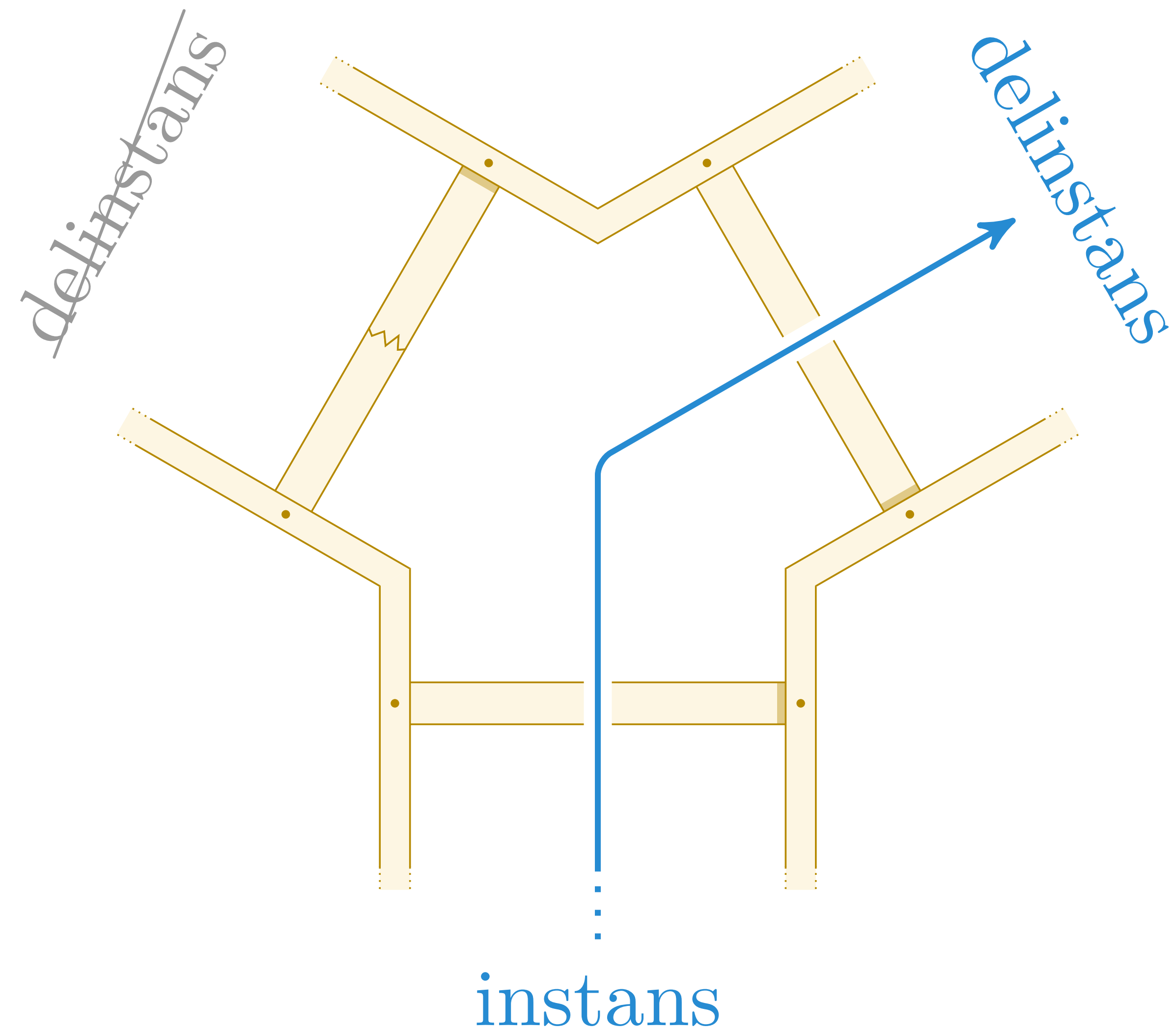
Optimal løsning bygger videre i én av retningene



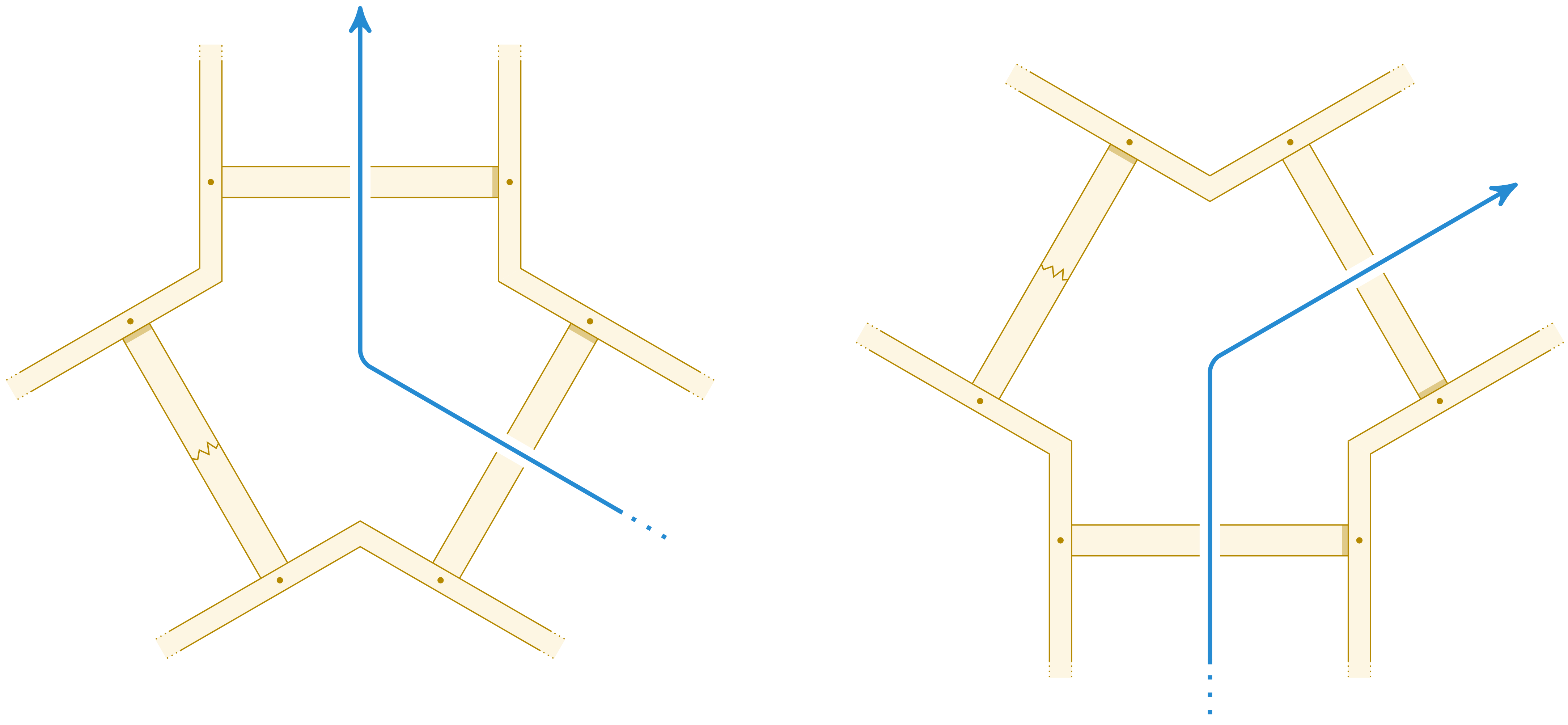
Før vi bygger videre: Hvilken retning er mest lovende?



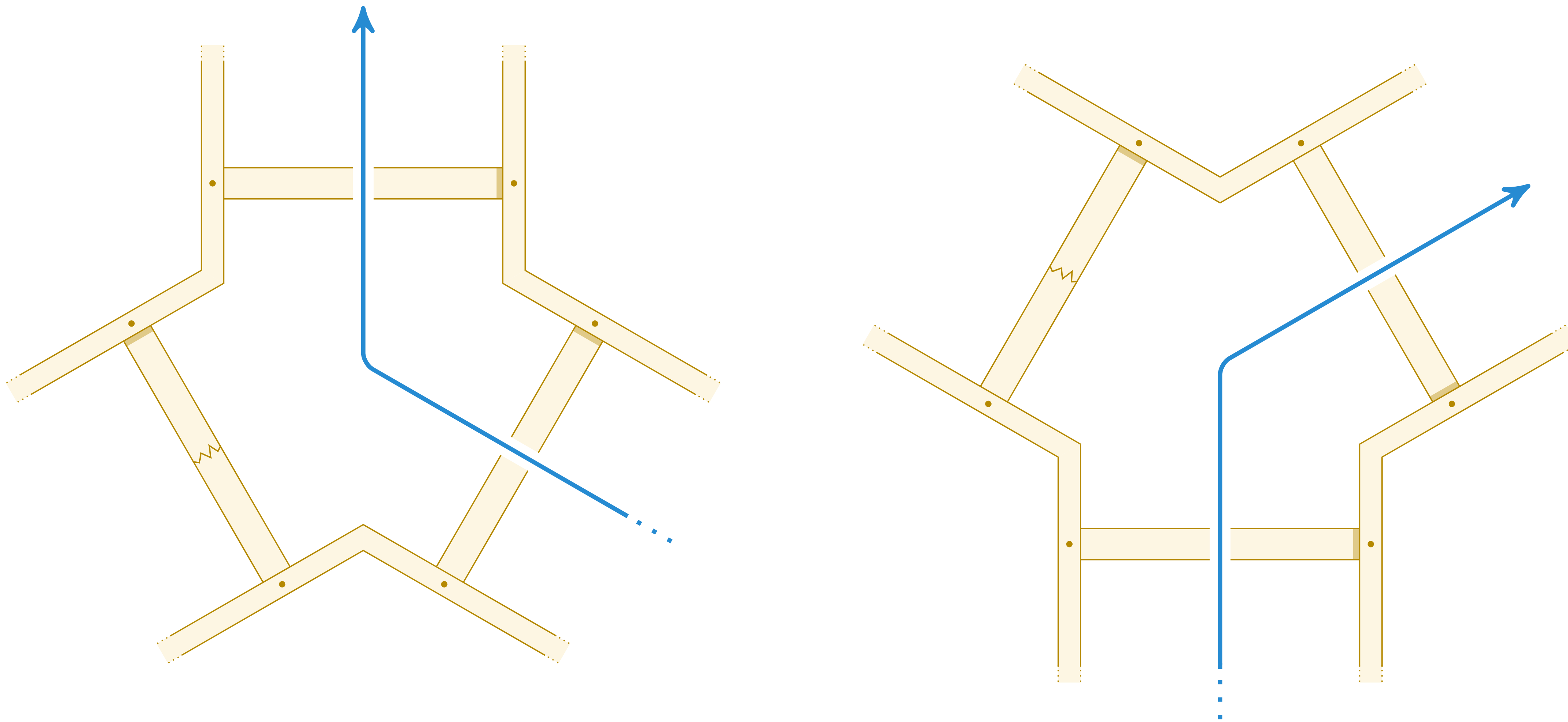
Vi bygger kun videre i den mest lovende retningen



Vi bygger <u>kun</u> videre i den mest lovende retningen



De to perspektivene er helt ekvivalente . . .



…men begge kan være nyttige

- **Dynamisk programmering:**
  - **Løs delproblemer rekursivt**
  - **Bygg løsning på beste delløsning**
- **Grådighet**
  - **Løs det mest lovende delproblemet rekursivt**
  - **Bygg løsning på denne delløsningen**
- **DP vil fortsatt fungere – akkurat som for D&C**

Ting å vise

Med «grådighetsegenskapen» mener jeg «the greedy-choice property». Merk at det kun beskriver at det *første* grådige valget er trygt – ikke at vi kan *fortsette* å velge grådig.

1. **Grådighetsegenskapen**
   Kan velge det som ser best ut her og nå <u>uten å skyte oss i foten</u>

2. **Optimal delstruktur**
   Kan fortsette på samme måte: Opt. løsning bygger på opt. delløsninger

Dvs.

Grådig valg + optimal delløsning gir optimal løsning

Om vi ikke har …

1. **Grådighetsegenskapen**
   vil et grådig valg kunne gjøre at vi ikke lenger har håp om optimalitet

2. **Optimal delstruktur**
   vil vi kunne måtte løse ting på en helt annen måte etter første valg

1. **Formuler som opt.-problem der vi tar et valg så ett delproblem gjenstår**

2. **Vis at det alltid finnes en optimal løsning som tar det grådige valget**

3. **Vis at optimal løsning på grådig valgt delproblem gir globalt optimal løsning**

2:4

Kontinuerlig

Eksempel: Ryggsekk

Input: Verdier $v_1, \ldots, v_n$, vekter $w_1, \ldots, w_n$ og en kapasitet W.

Output: Indekser $i_1, \ldots, i_k$ og en fraksjon $0 \leq \epsilon \leq 1$ slik at $w_{i_1} + \cdots + w_{i_{k-1}} + \epsilon \cdot w_{i_k} \leq$ W og totalverdien $v_{i_1} + \cdots + v_{k-1} + \epsilon \cdot v_{i_k}$ er maksimal.

- **Fra {0,1} til brøk**
  - **Velg alltid det med høyest kilopris**
  - **Begge har optimal substruktur**
  - **{0,1}-varianten kan ikke løses grådig**

Prøv et såkalt «utvekslingsargument» (exchange argument), der du antar en annen løsning, og endrer til den grådige: Tenk deg at du tar med mindre enn mest mulig av det med høyest kilopris. Du kunne da byttet ut noe av det som har lavere pris med litt mer av det med høyest kilopris og fått en bedre totalpris!

- **Grådighetsegenskapen**: Det finnes en optimal løsning der vi tar med mest mulig av det dyreste

- **Optimal delstruktur**: Om vi tar med noe, må resten av sekken fortsatt fylles optimalt

Samme logikk: Tenk deg at resten ikke løses optimalt: Da kunne du jo ha fått en bedre løsning ved å løse resten optimalt. (Rimelig opplagt, i dette tilfellet…)

3:4

Eksempel: Aktivitetsutvalg

Input: Intervaller $[s_1, f_1), \ldots, [s_n, f_n)$.

Output: Flest mulig ikke-overlappende intervaller.

- **Skal velge størst mulig delmengde av ikke-overlappende intervaller**
- **Delproblem: Intervaller innenfor et gitt område**
- **Valg: Et intervall som skal bli med**
  - **Løs begge delproblemer rekursivt og legg til 1**
- **Men: Vi trenger ikke se på alle disse delproblemene!**
  - **Det vil alltid lønne seg å ta med intervallet som slutter først!**

- **Grådighetsegenskapen**:
  Det finnes en optimal løsning som inkluderer første intervall

- **Optimal delstruktur**:
  Om vi velger første intervall, må resten fortsatt løses optimalt

$\textsc{Rec-Act-Sel}(s, f, k, n)$

$s[i]$	start, a_i
$f[i]$	slutt, a_i
a_k	forrige
n	antall

Velg flest mulig ikke-overlappende akt. fra $a_{k+1} \ldots a_n$

Rec-Act-Sel(s, f, k, n)

1 $m = k + 1$

$s[i]$	start, a_i
$f[i]$	slutt, a_i
a_k	forrige
a_m	neste
n	antall

Vi vil ha det første lovlige (sortert etter f)

REC-ACT-SEL(s, f, k, n)
1 $m = k + 1$
2 **while** $m \leq n$ and $s[m] < f[k]$

$s[i]$	start, a_i
$f[i]$	slutt, a_i
a_k	forrige
a_m	neste
n	antall

Begynner a_m før a_k slutter?

$\textsc{Rec-Act-Sel}(s, f, k, n)$

$m = k + 1$

while $m \leq n$ and $s[m] < f[k]$

    $m = m + 1$

$s[i]$	start, a_i
$f[i]$	slutt, a_i
a_k	forrige
a_m	neste
n	antall

Prøv neste aktivitet i stedet

Rec-Act-Sel(s, f, k, n)

```
m = k + 1
while m ≤ n and s[m] < f[k]
      m = m + 1
if m ≤ n
```

$s[i]$	start, a_i
$f[i]$	slutt, a_i
a_k	forrige
a_m	neste
n	antall

Fant vi en a_m som startet etter $f[k]$?

Rec-Act-Sel(s, f, k, n)

$m = k + 1$
while $m \leq n$ and $s[m] < f[k]$
    $m = m + 1$
if $m \leq n$
    $\mathrm{S} =$ Rec-Act-Sel(s, f, m, n)

$s[i]$	start, a_i
$f[i]$	slutt, a_i
a_k	forrige
a_m	neste
n	antall
S	delsvar

Finn resten av løsningen rekursivt

mlh@ntnu.no

Rec-Act-Sel(s, f, k, n)

$m = k + 1$

while $m \leq n$ and $s[m] < f[k]$

  $m = m + 1$

if $m \leq n$

  $\mathrm{S} =$ Rec-Act-Sel(s, f, m, n)

  return $\{a_m\} \cup \mathrm{S}$

$s[i]$	start, a_i
$f[i]$	slutt, a_i
a_k	forrige
a_m	neste
n	antall
S	delsvar

Vi har alt valgt a_m; kombinér med resten

$\textsc{Rec-Act-Sel}(s, f, k, n)$

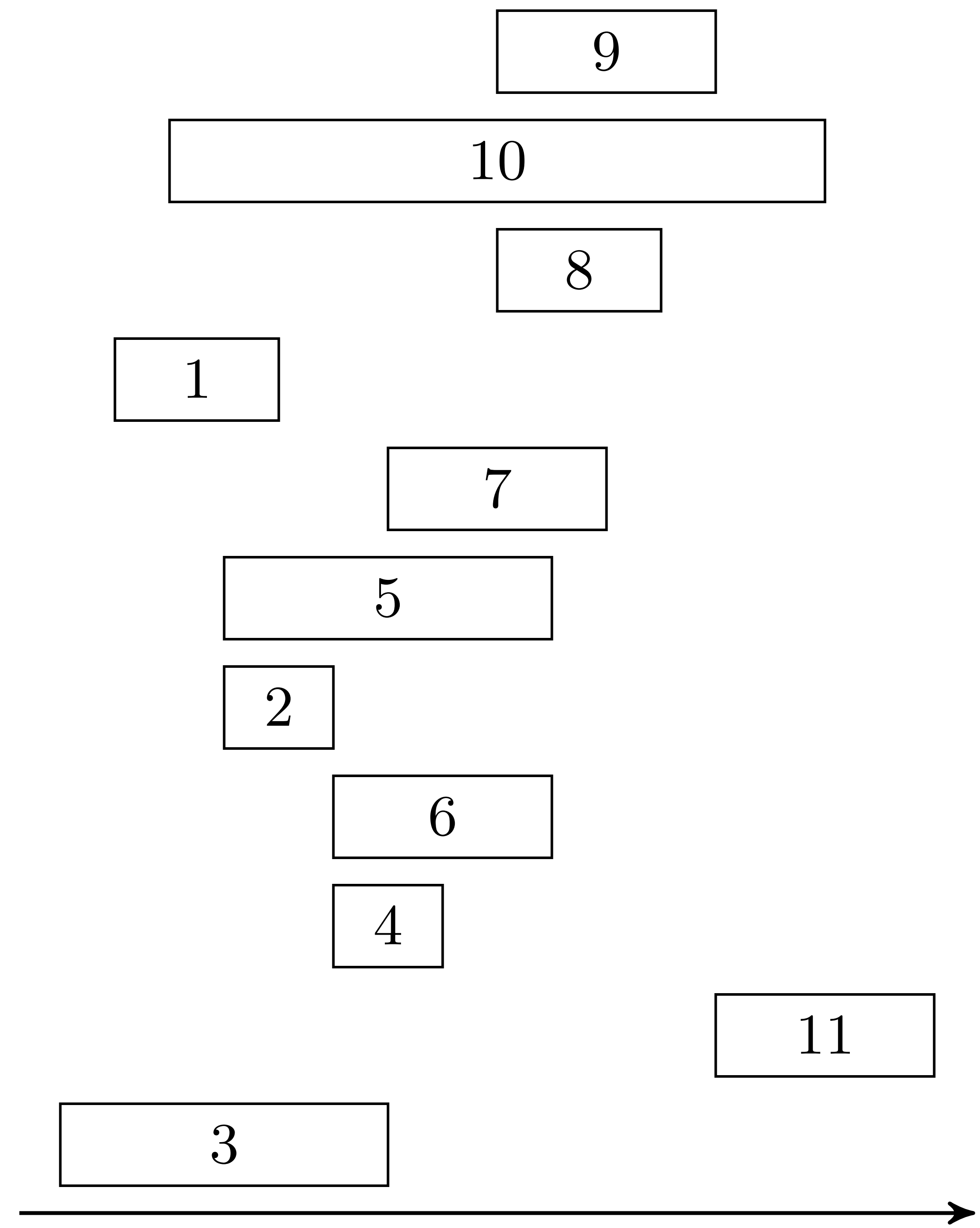
$m = k + 1$
while $m \leq n$ and $s[m] < f[k]$
    $m = m + 1$
if $m \leq n$
    $\mathrm{S} = \textsc{Rec-Act-Sel}(s, f, m, n)$
    return $\{a_m\} \cup \mathrm{S}$
else return $\emptyset$

$s[i]$	start, a_i
$f[i]$	slutt, a_i
a_k	forrige
a_m	neste
n	antall
S	delsvar

Vi fant ingen gyldige intervaller

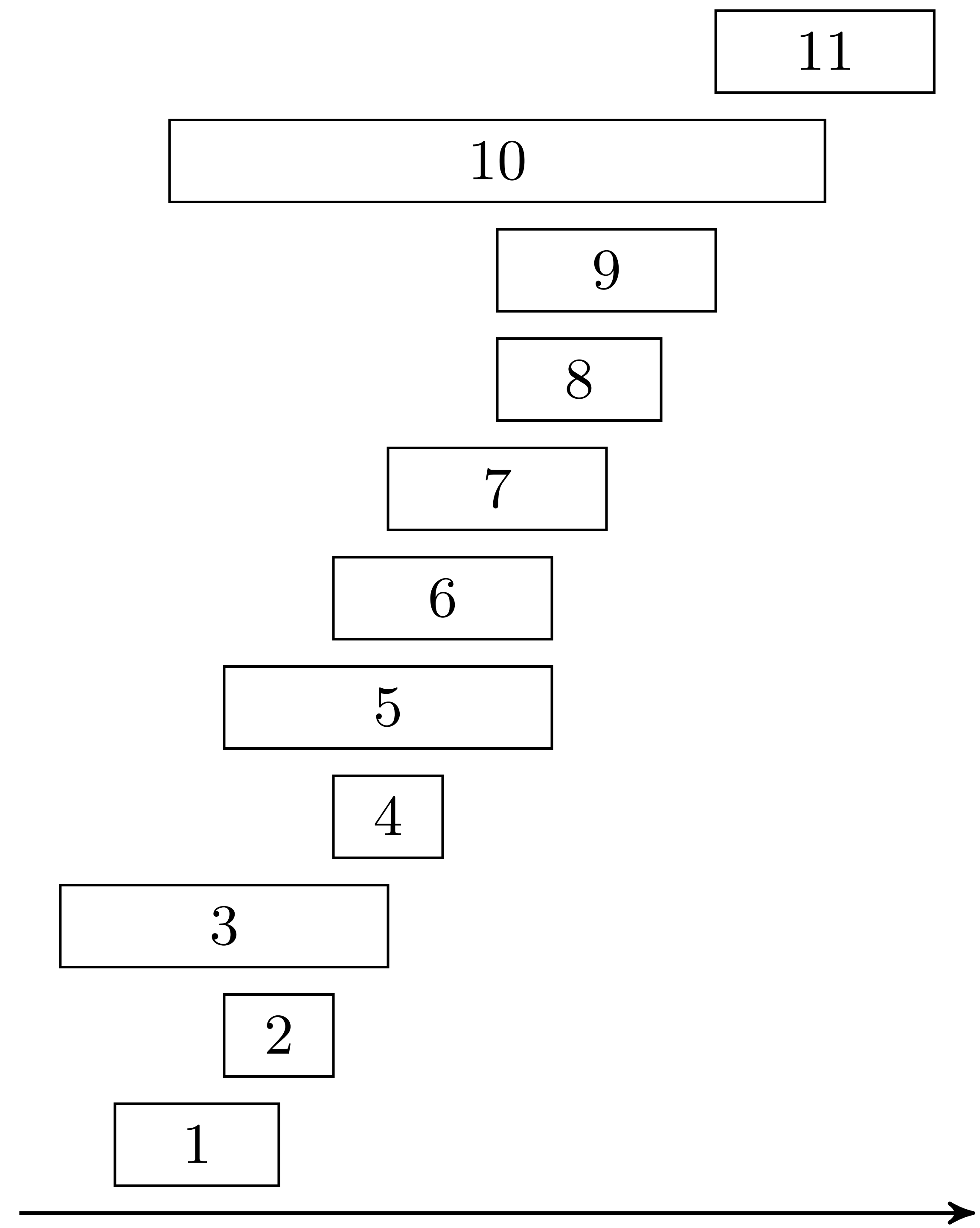
mlh@ntnu.no

```
REC-ACT-SEL(s, f, k, n)
m = k + 1
while m ≤ n and s[m] < f[k]
      m = m + 1
if m ≤ n
      S = REC-ACT-SEL(s, f, m, n)
      return {a_m} ∪ S
else return ∅
```



Rec-Act-Sel(s, f, k, n)

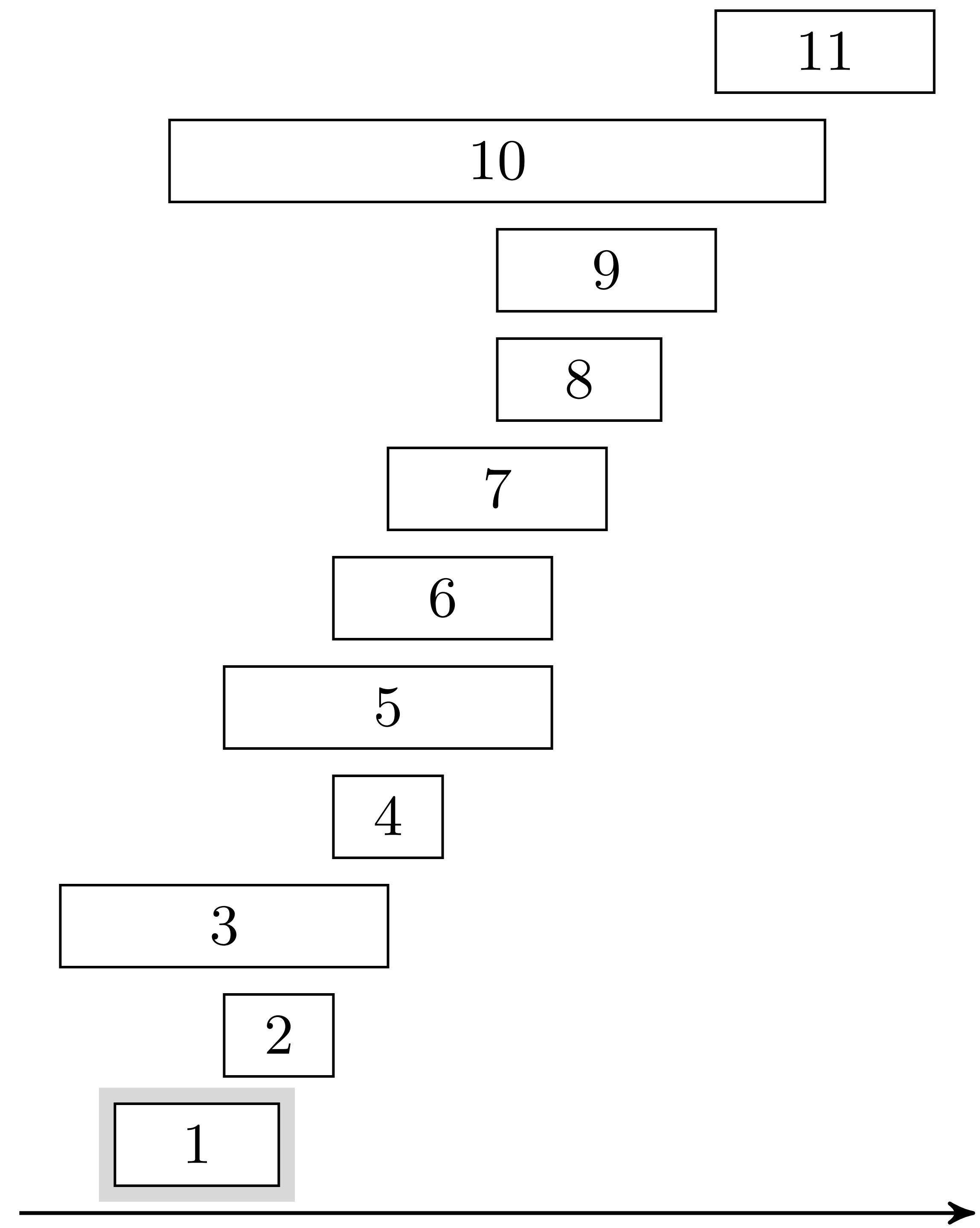
$m = k + 1$
while $m \leq n$ and $s[m] < f[k]$
  $m = m + 1$
if $m \leq n$
  $S =$ Rec-Act-Sel(s, f, m, n)
  return $\{a_m\} \cup S$
else return $\emptyset$



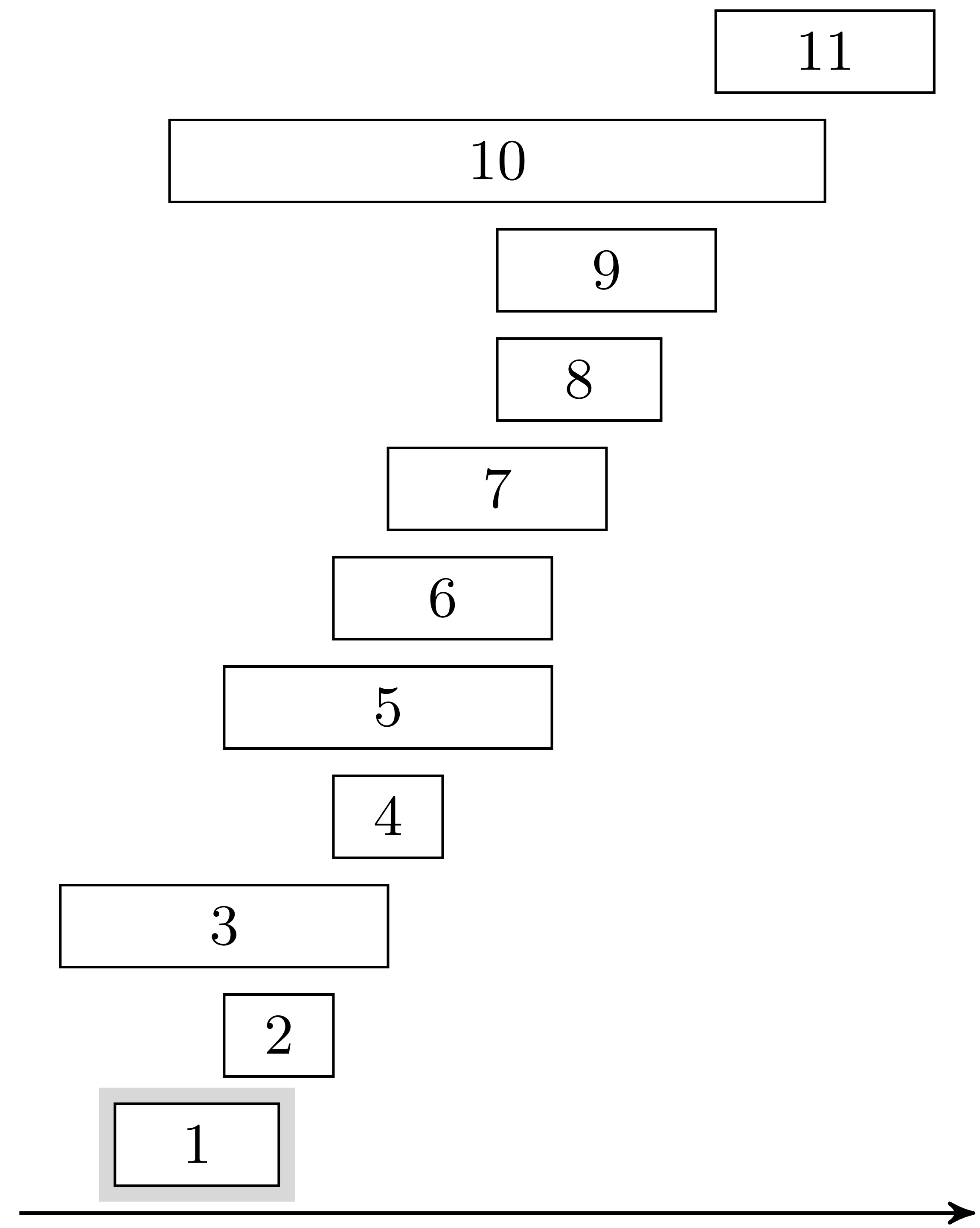
$k, m \;=\; 0, -$

```
REC-ACT-SEL(s, f, k, n)
  m = k + 1
  while m ≤ n and s[m] < f[k]
      m = m + 1
  if m ≤ n
      S = REC-ACT-SEL(s, f, m, n)
      return {a_m} ∪ S
  else return ∅
```

$k, m \;=\; 0, 1$



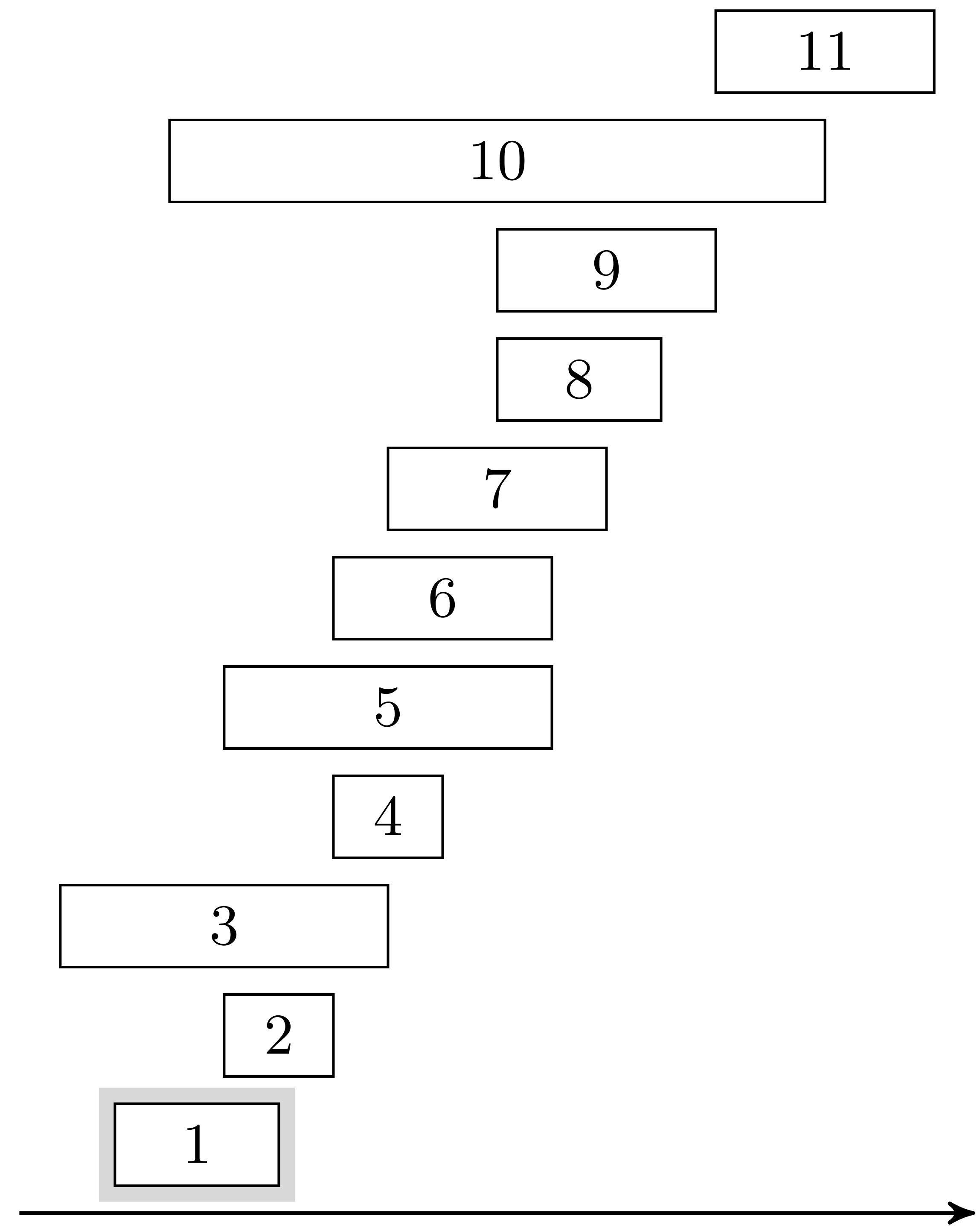
```
Rec-Act-Sel(s, f, k, n)
m = k + 1
while m ≤ n and s[m] < f[k]
        m = m + 1
if m ≤ n
        S = Rec-Act-Sel(s, f, m, n)
        return {a_m} ∪ S
else return ∅
```



$k, m \;=\; 0, 1$

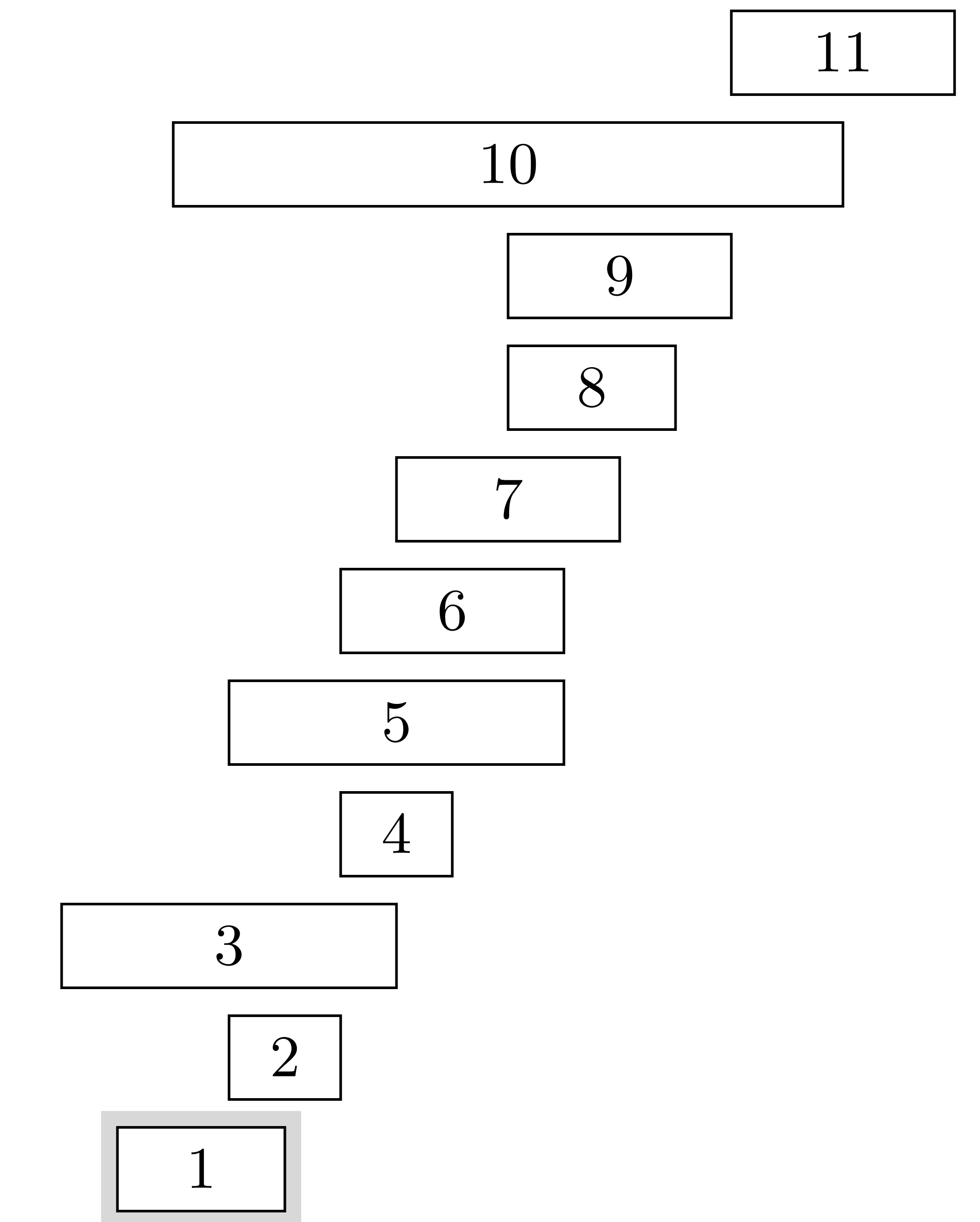
```
REC-ACT-SEL(s, f, k, n)
  m = k + 1
  while m ≤ n and s[m] < f[k]
        m = m + 1
  if m ≤ n
        S = REC-ACT-SEL(s, f, m, n)
        return {a_m} ∪ S
  else return ∅
```

$k, m \ = \ 0, 1$



Rec-Act-Sel(s, f, k, n)

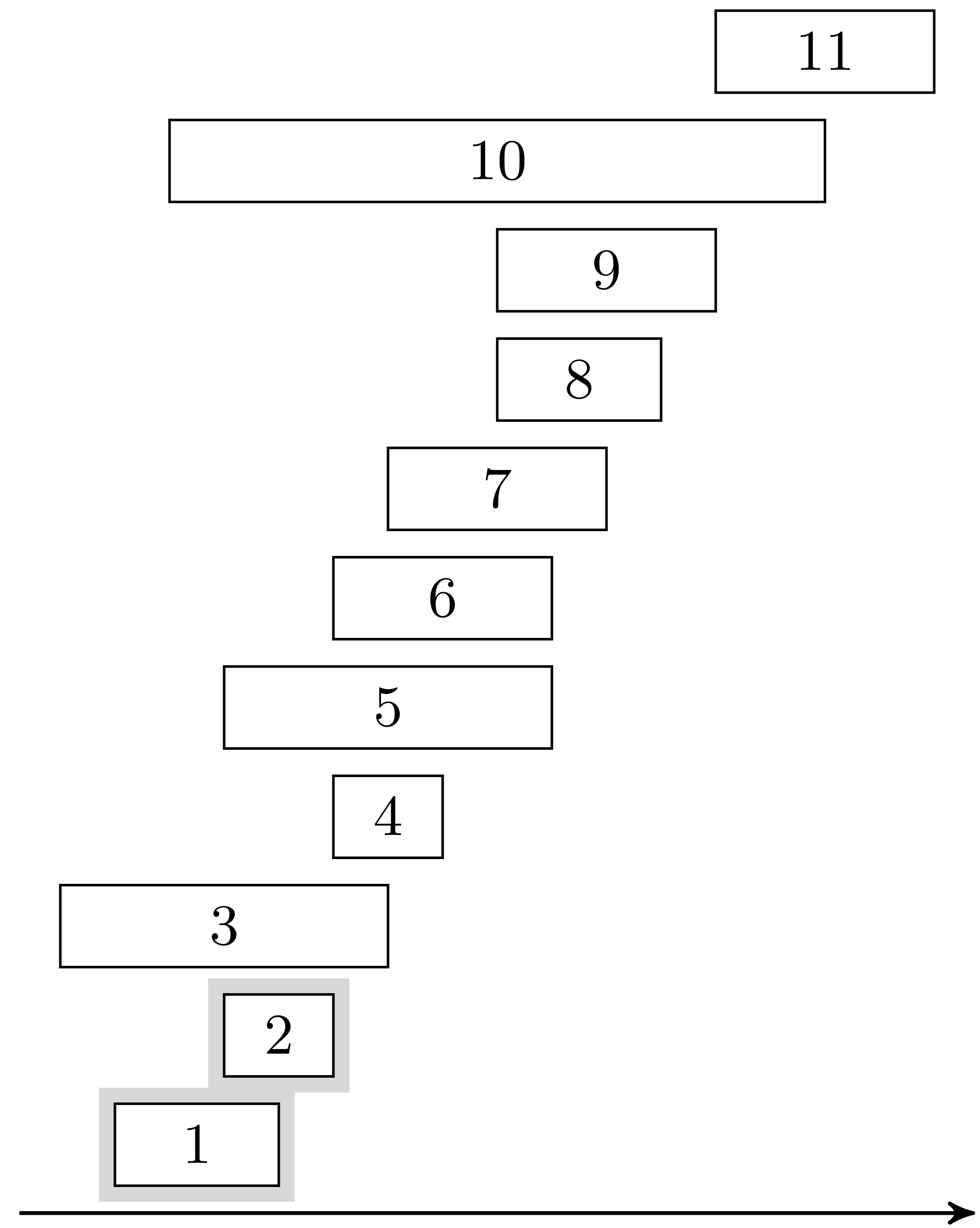
```
m = k + 1
while m ≤ n and s[m] < f[k]
    m = m + 1
if m ≤ n
    S = Rec-Act-Sel(s, f, m, n)
    return {a_m} ∪ S
else return ∅
```



$k, m \;=\; 0, 1 \;›\; 1, \text{-}$

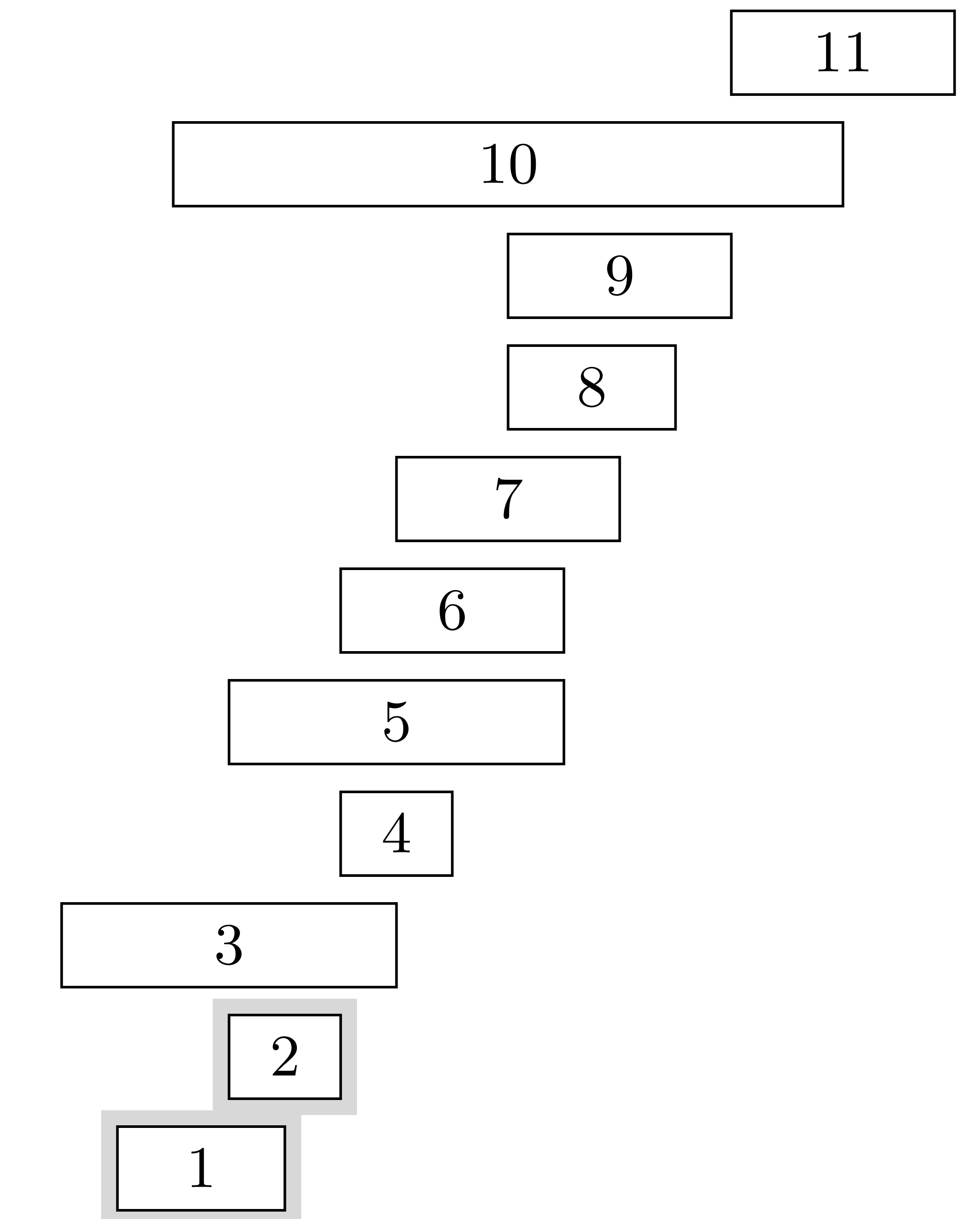
```
Rec-Act-Sel(s, f, k, n)
m = k + 1
while m ≤ n and s[m] < f[k]
        m = m + 1
if m ≤ n
        S = Rec-Act-Sel(s, f, m, n)
        return {a_m} ∪ S
else return ∅
```

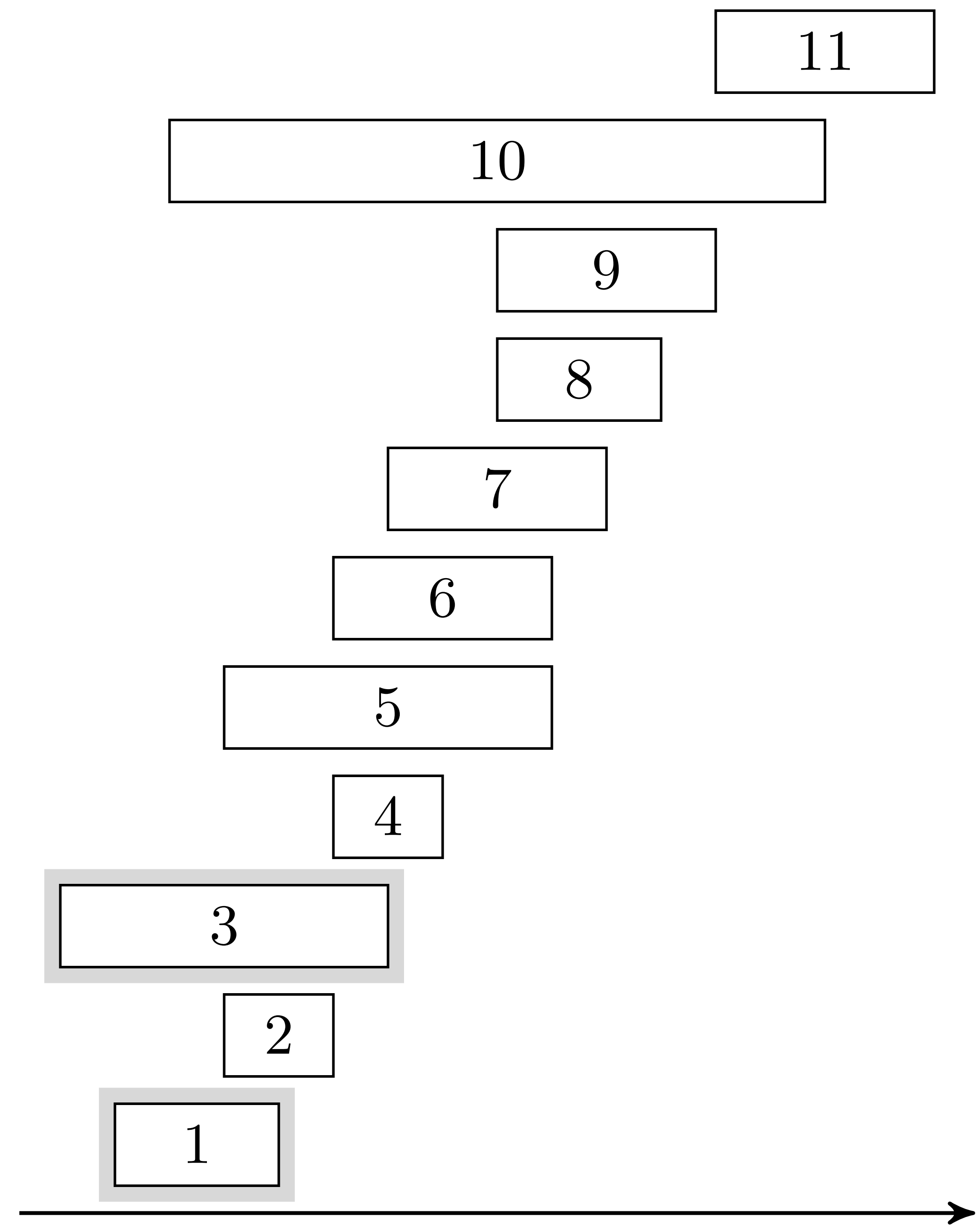
$k, m \;=\; 0, 1 \;›\; 1, 2$



```
Rec-Act-Sel(s, f, k, n)
m = k + 1
while m ≤ n and s[m] < f[k]
        m = m + 1
if m ≤ n
        S = Rec-Act-Sel(s, f, m, n)
        return {a_m} ∪ S
else return ∅
```

$k, m \;=\; 0, 1 \;›\; 1, 2$



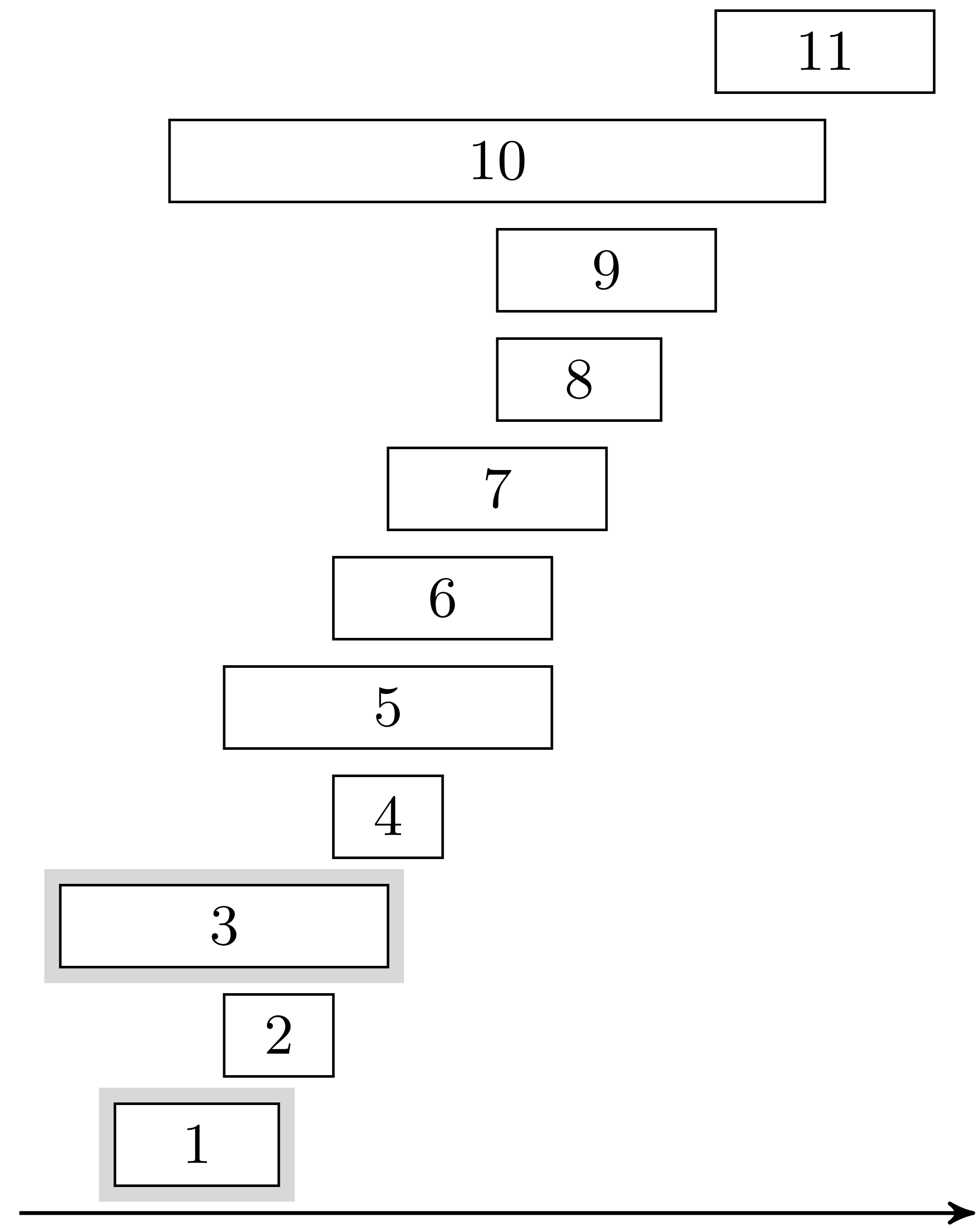


$\textsc{Rec-Act-Sel}(s, f, k, n)$

```
m = k + 1
while m ≤ n and s[m] < f[k]
      m = m + 1
if m ≤ n
      S = Rec-Act-Sel(s, f, m, n)
      return {a_m} ∪ S
else return ∅
```

$k, m \;=\; 0, 1 \;›\; 1, 3$

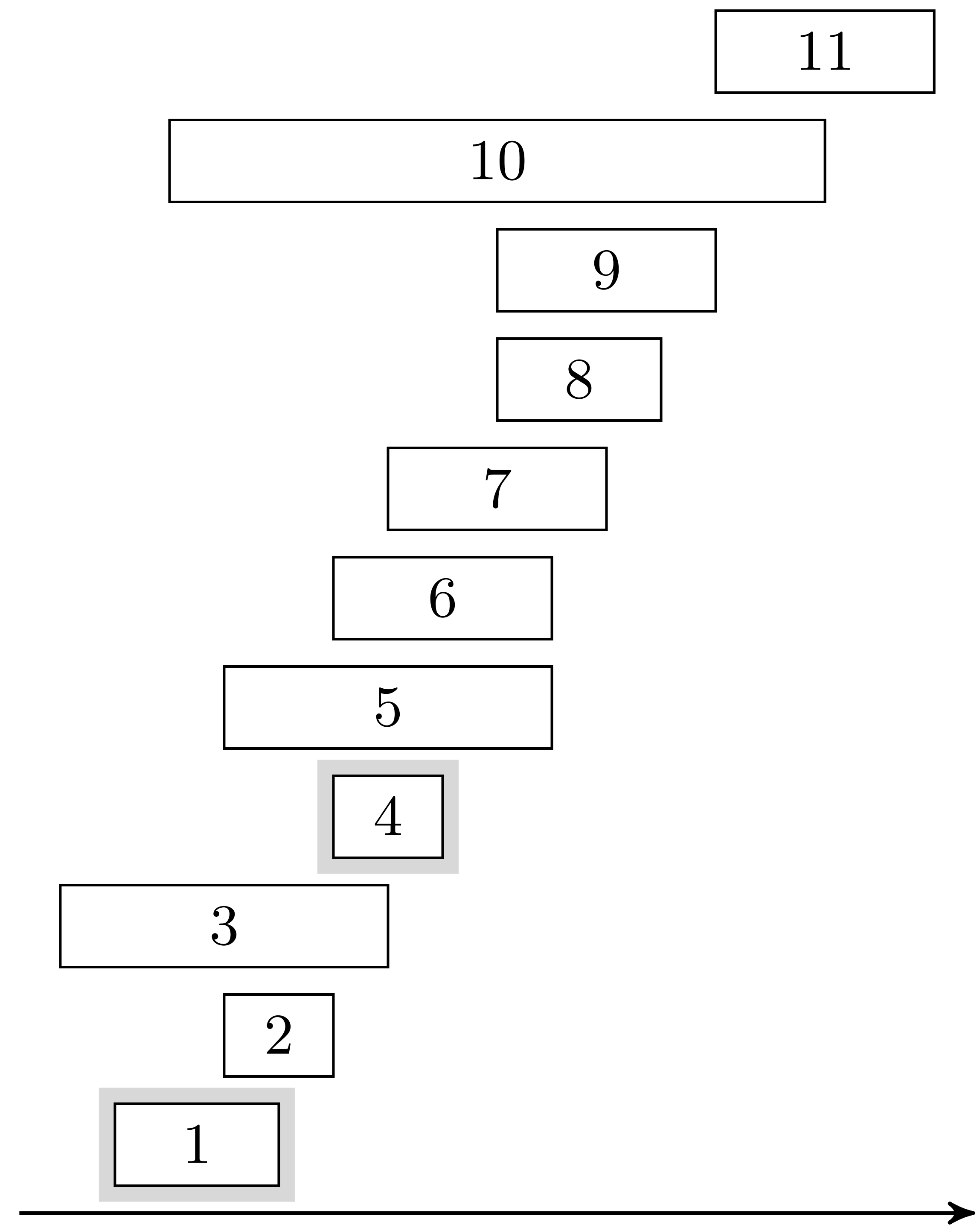
```
Rec-Act-Sel(s, f, k, n)
m = k + 1
while m ≤ n and s[m] < f[k]
        m = m + 1
if m ≤ n
        S = Rec-Act-Sel(s, f, m, n)
        return {a_m} ∪ S
else return ∅
```



$k, m \;=\; 0, 1 \;›\; 1, 3$

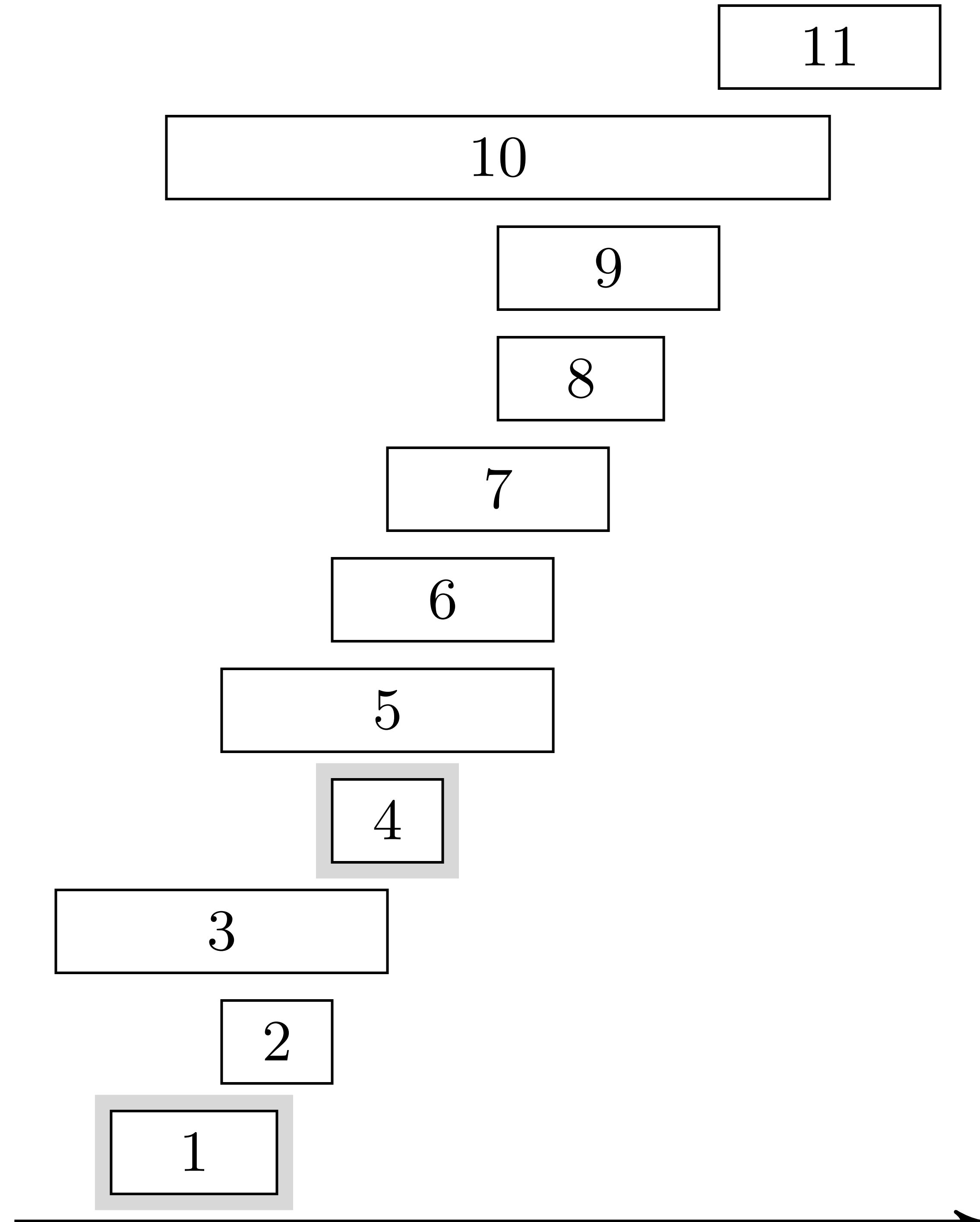
Rec-Act-Sel(s, f, k, n)

$m = k + 1$
while $m \leq n$ and $s[m] < f[k]$
    $m = m + 1$
if $m \leq n$
    $S = $ Rec-Act-Sel(s, f, m, n)
    return $\{a_m\} \cup S$
else return $\emptyset$



$k, m \;=\; 0, 1 \;›\; 1, 4$

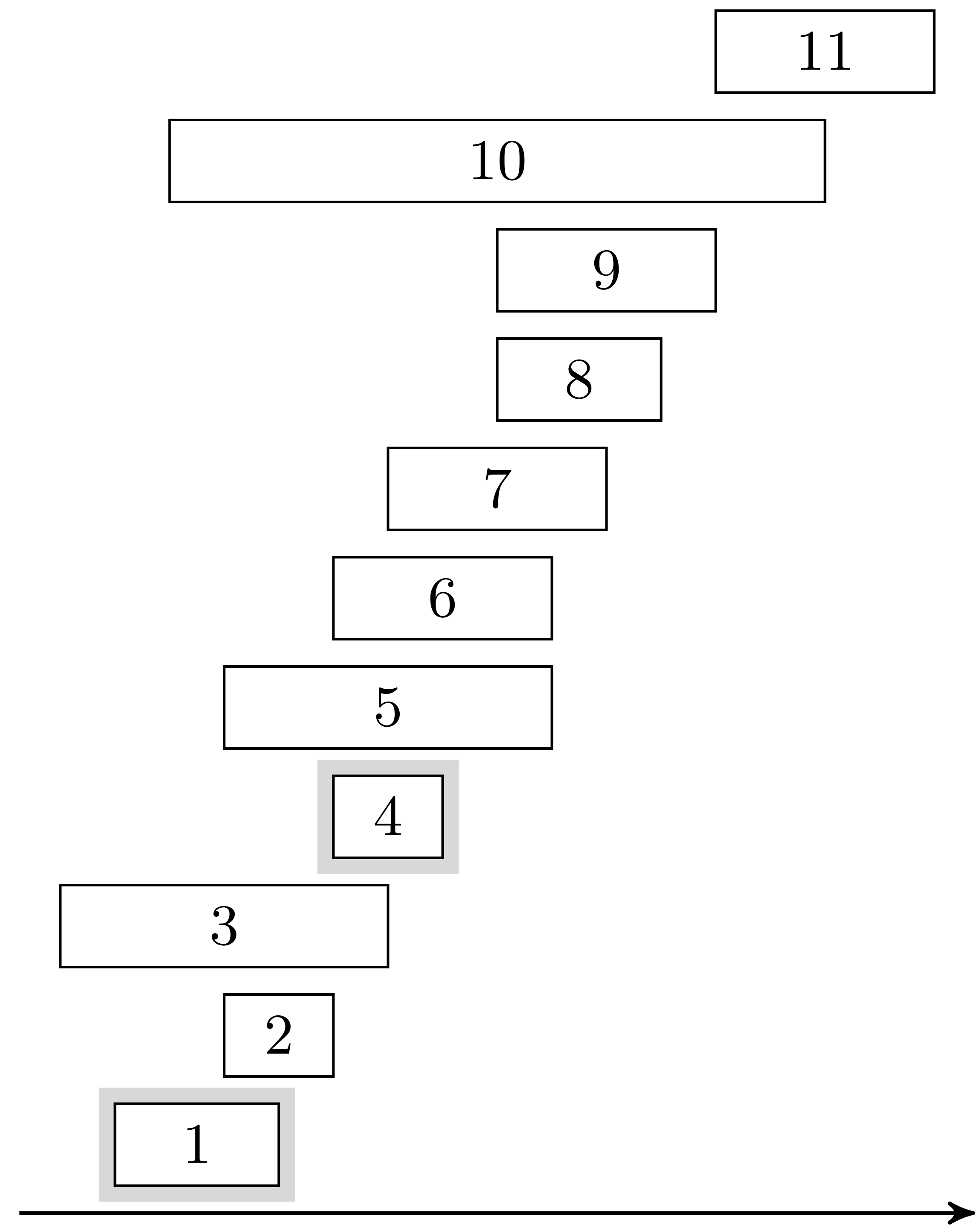
```
REC-ACT-SEL(s, f, k, n)
m = k + 1
while m ≤ n and s[m] < f[k]
      m = m + 1
if m ≤ n
      S = REC-ACT-SEL(s, f, m, n)
      return {a_m} ∪ S
else return ∅
```



$k, m \;=\; 0, 1 \;›\; 1, 4$

Rec-Act-Sel(s, f, k, n)

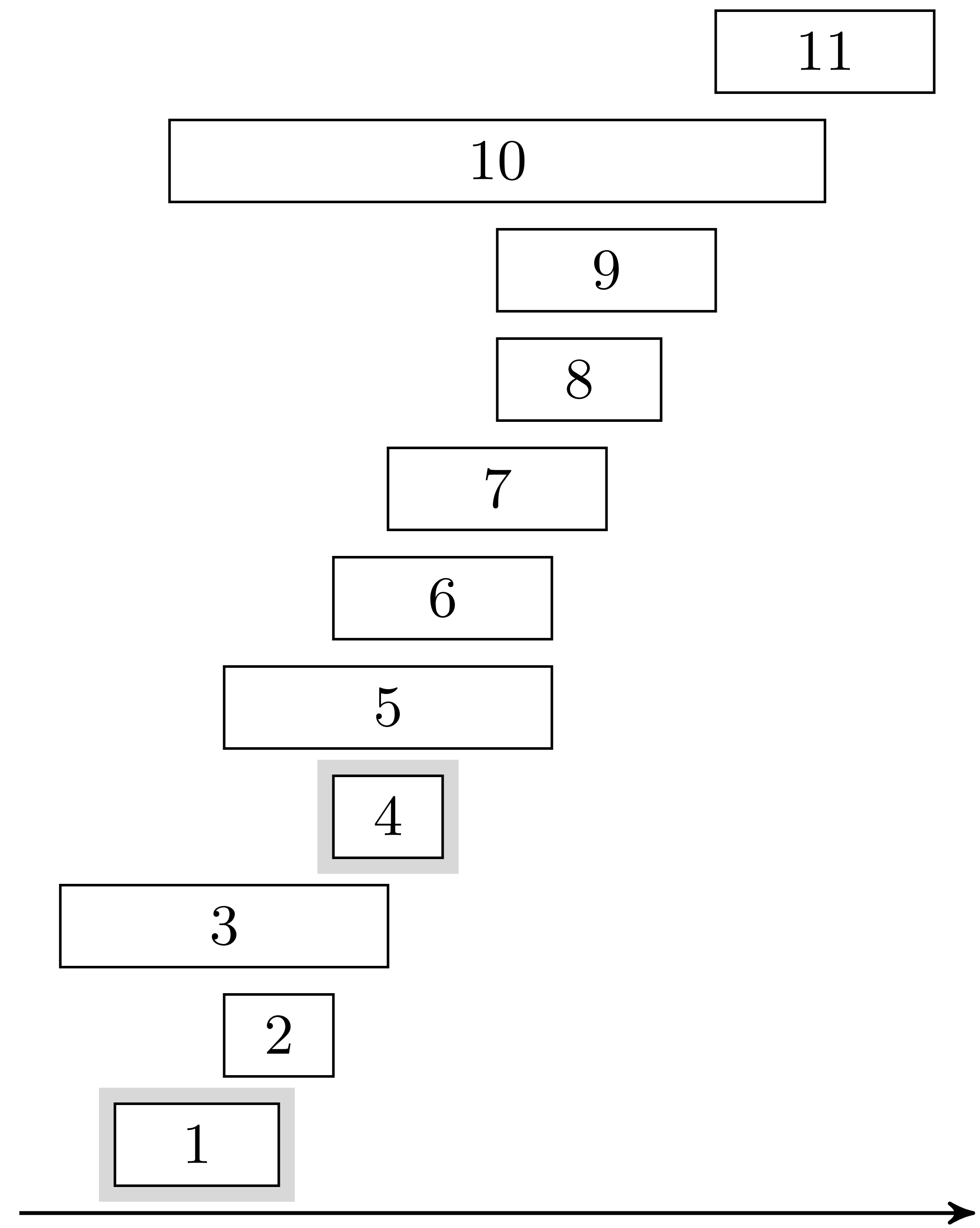
$m = k + 1$
while $m \leq n$ and $s[m] < f[k]$
    $m = m + 1$
if $m \leq n$
    $S = $ Rec-Act-Sel(s, f, m, n)
    return $\{a_m\} \cup S$
else return $\emptyset$



$k, m \;=\; 0, 1 \;›\; 1, 4$

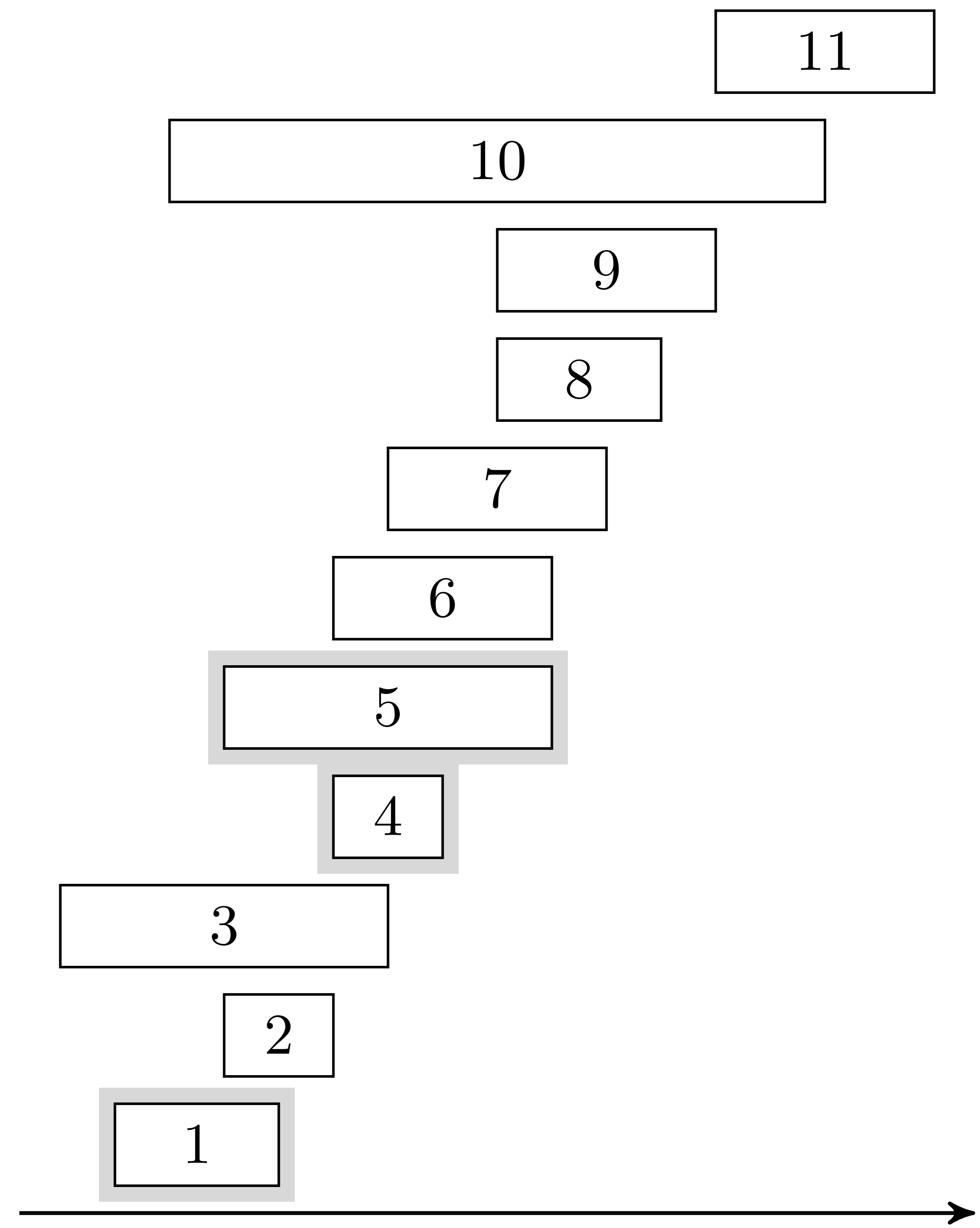
Rec-Act-Sel(s, f, k, n)

$m = k + 1$
while $m \leq n$ and $s[m] < f[k]$
    $m = m + 1$
if $m \leq n$
    $\mathrm{S} =$ Rec-Act-Sel(s, f, m, n)
    return $\{a_m\} \cup \mathrm{S}$
else return $\emptyset$



$k, m \;=\; 0, 1 \;›\; 1, 4 \;›\; 4,$ -

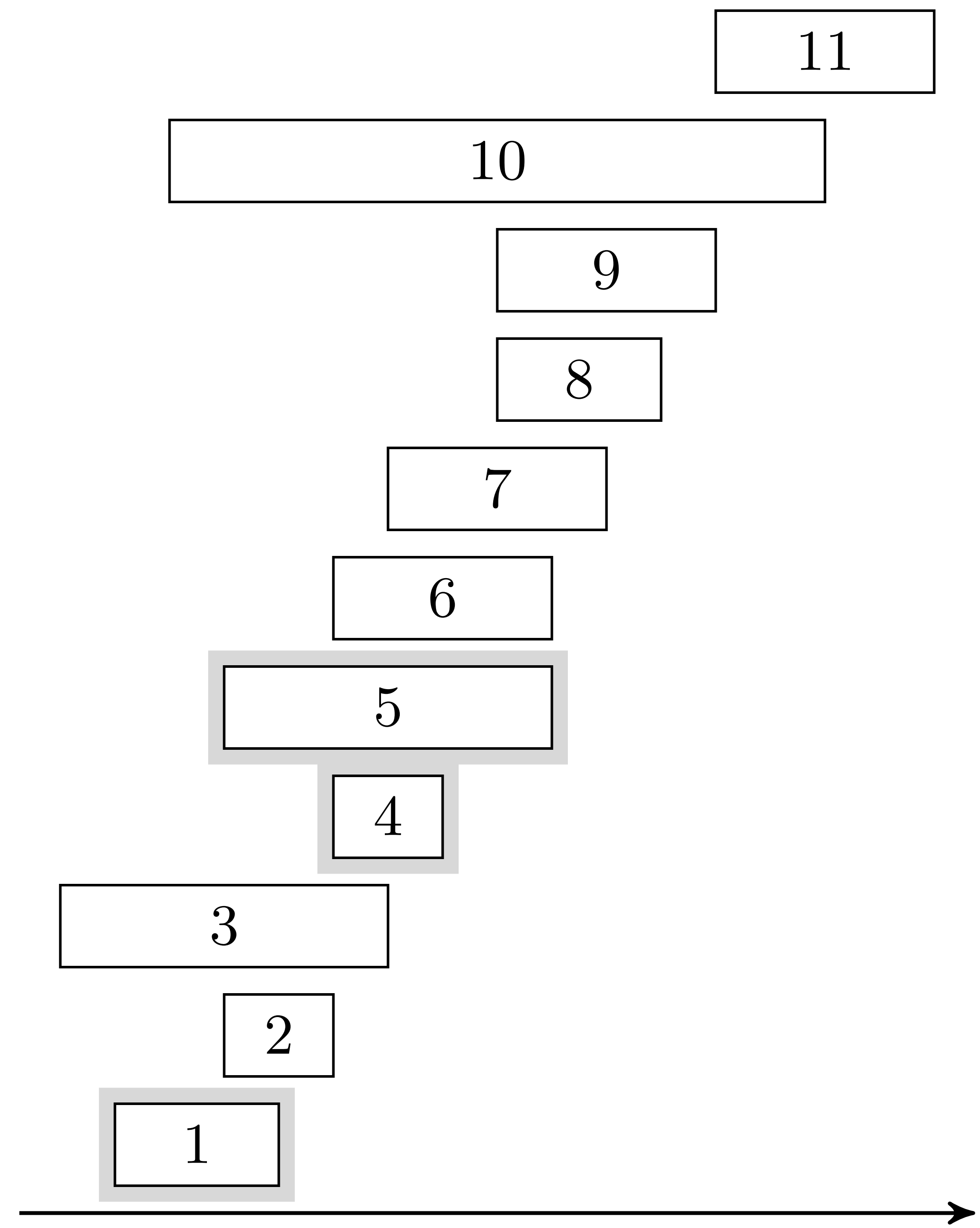
```
REC-ACT-SEL(s, f, k, n)
m = k + 1
while m ≤ n and s[m] < f[k]
    m = m + 1
if m ≤ n
    S = REC-ACT-SEL(s, f, m, n)
    return {a_m} ∪ S
else return ∅
```



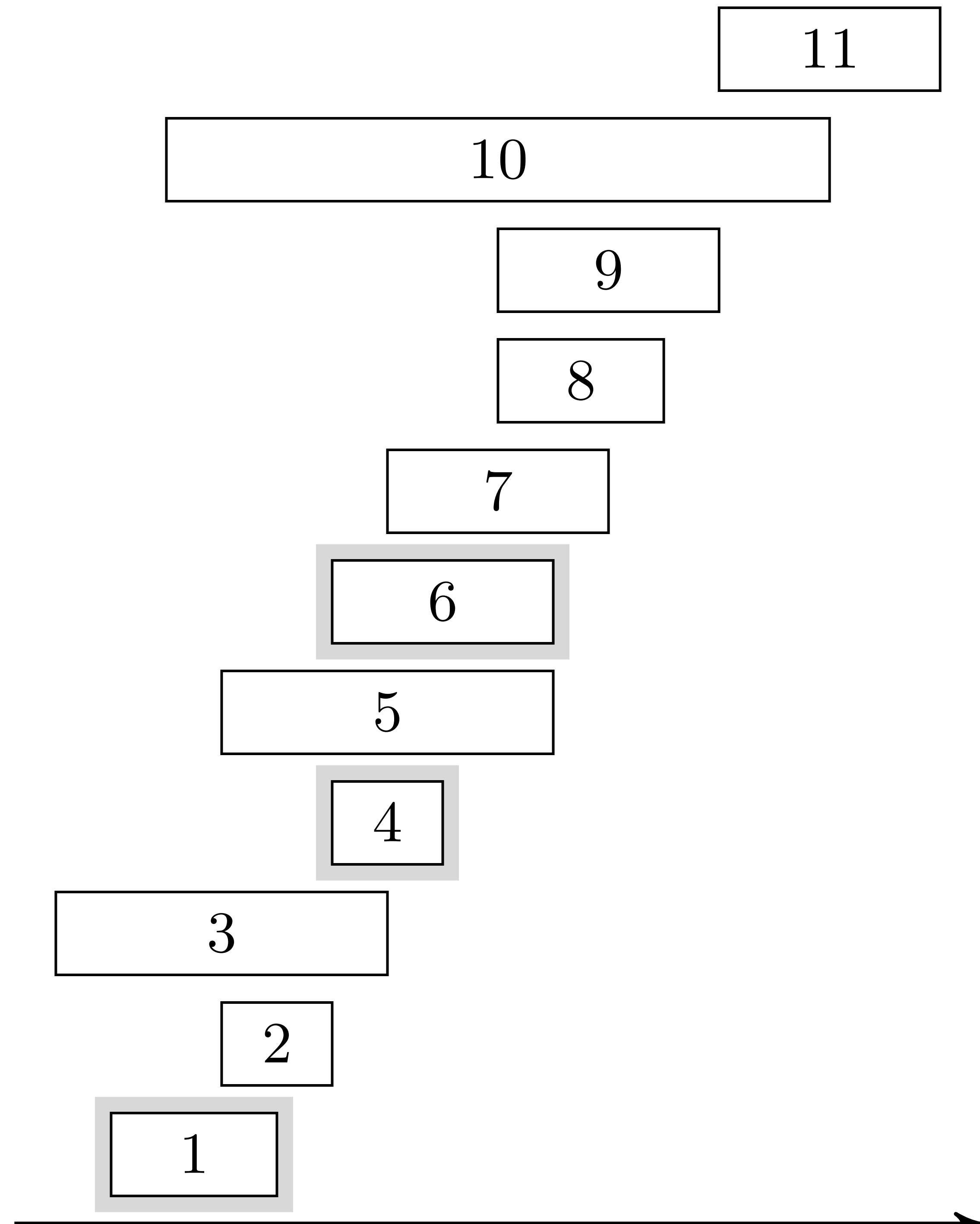
$k, m \;=\; 0, 1 \;›\; 1, 4 \;›\; 4, 5$

```
REC-ACT-SEL(s, f, k, n)
m = k + 1
while m ≤ n and s[m] < f[k]
        m = m + 1
if m ≤ n
        S = REC-ACT-SEL(s, f, m, n)
        return {a_m} ∪ S
else return ∅
```

$k, m \;=\; 0, 1 \;›\; 1, 4 \;›\; 4, 5$

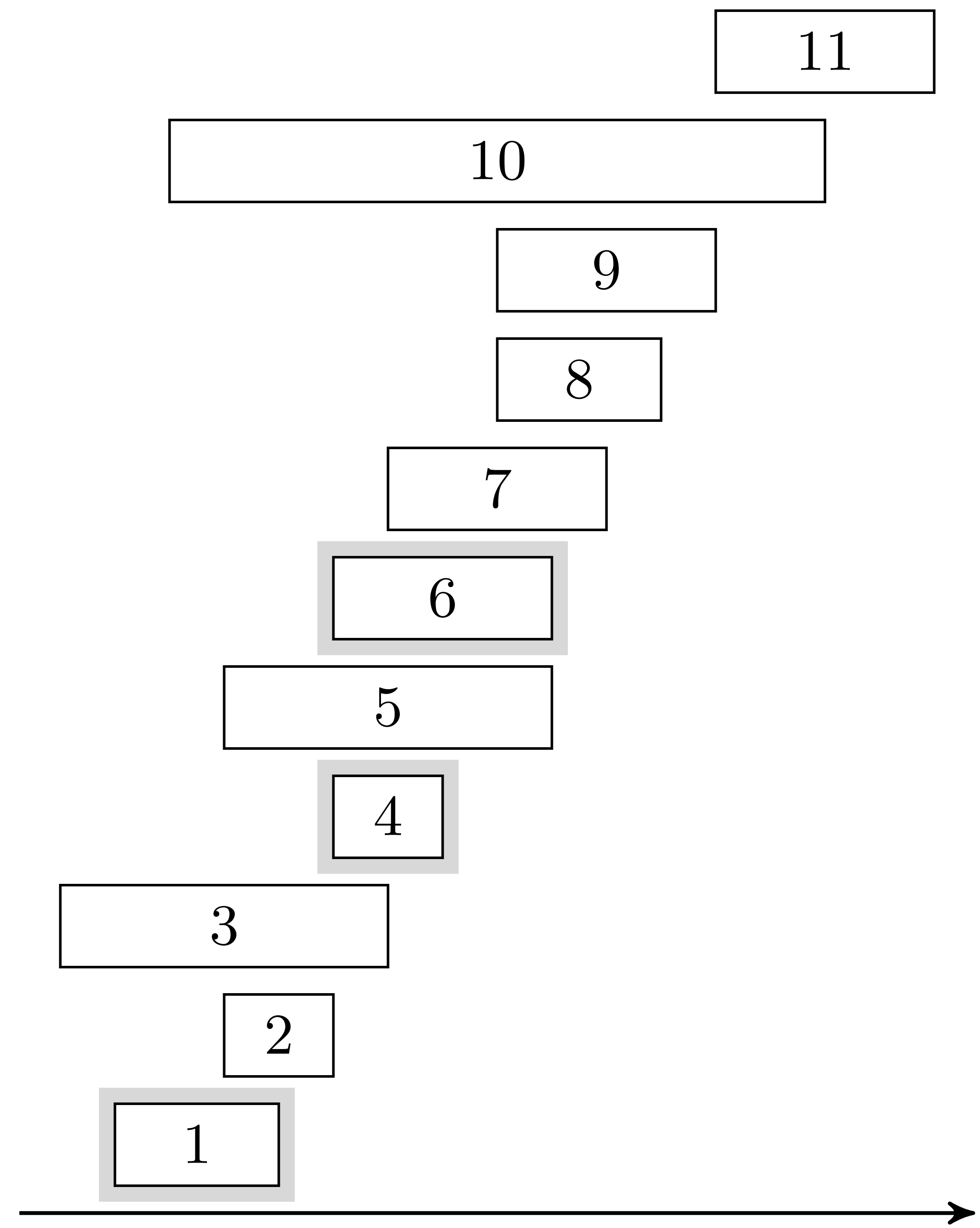


```
REC-ACT-SEL(s, f, k, n)
1  m = k + 1
2  while m ≤ n and s[m] < f[k]
3       m = m + 1
4  if m ≤ n
5       S = REC-ACT-SEL(s, f, m, n)
6       return {a_m} ∪ S
7  else return ∅
```



$k, m \;=\; 0,1 \;›\; 1,4 \;›\; 4,6$

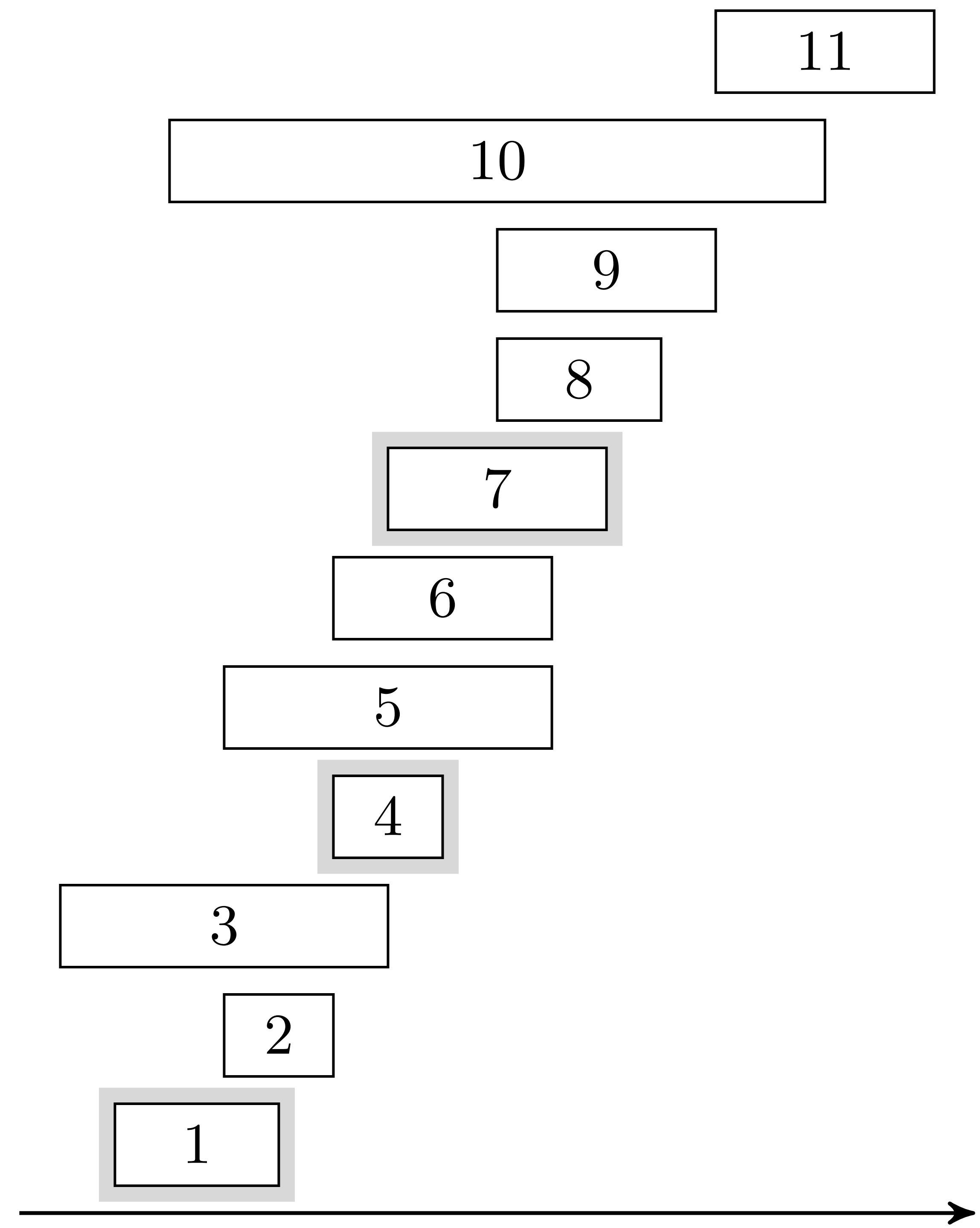
```
Rec-Act-Sel(s, f, k, n)
m = k + 1
while m ≤ n and s[m] < f[k]
    m = m + 1
if m ≤ n
    S = Rec-Act-Sel(s, f, m, n)
    return {a_m} ∪ S
else return ∅
```



$k, m \ = \ 0, 1 \ › \ 1, 4 \ › \ 4, 6$

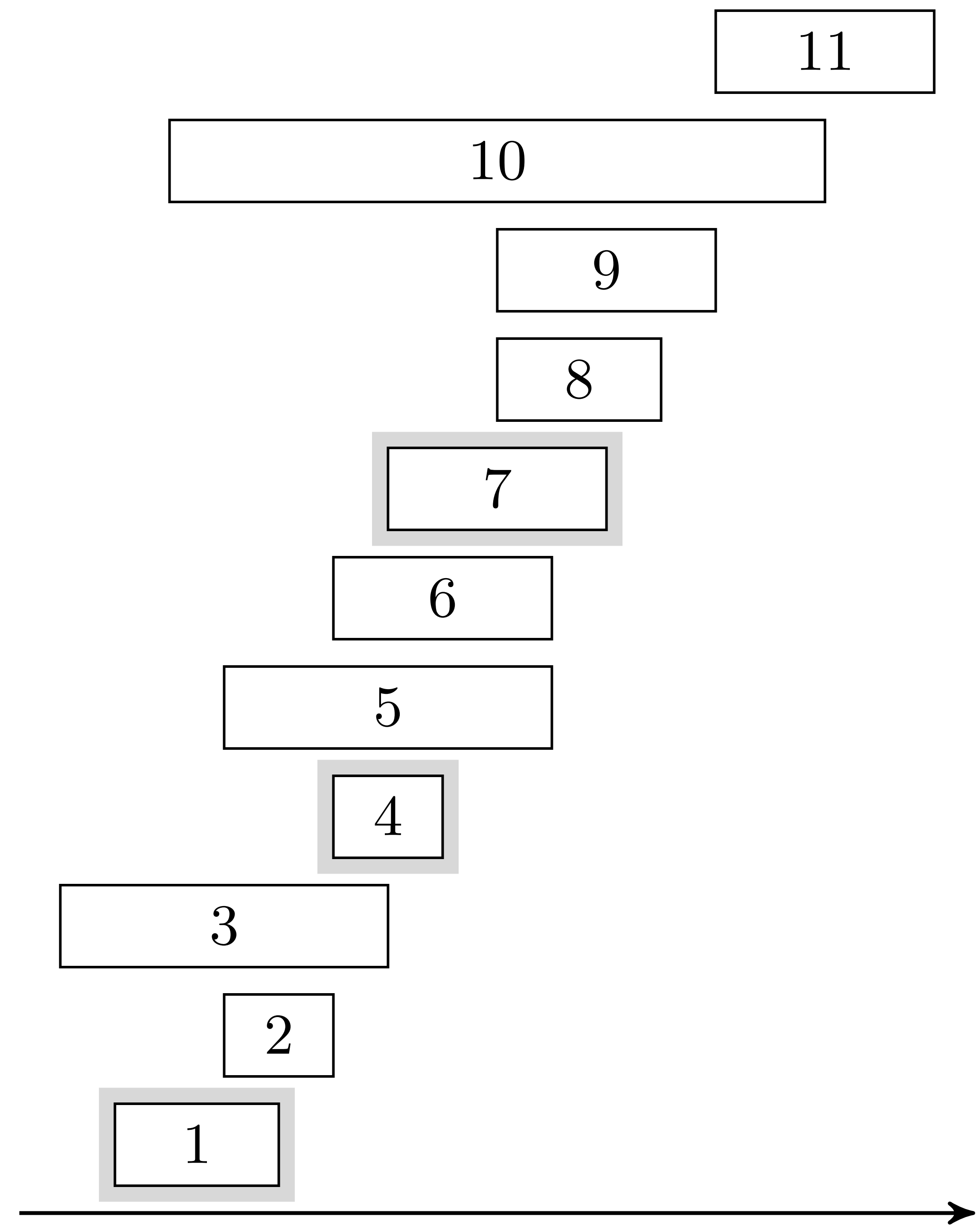
```
REC-ACT-SEL(s, f, k, n)
m = k + 1
while m ≤ n and s[m] < f[k]
        m = m + 1
if m ≤ n
        S = REC-ACT-SEL(s, f, m, n)
        return {a_m} ∪ S
else return ∅
```

$k, m \; = \; 0, 1 \;›\; 1, 4 \;›\; 4, 7$



Rec-Act-Sel(s, f, k, n)

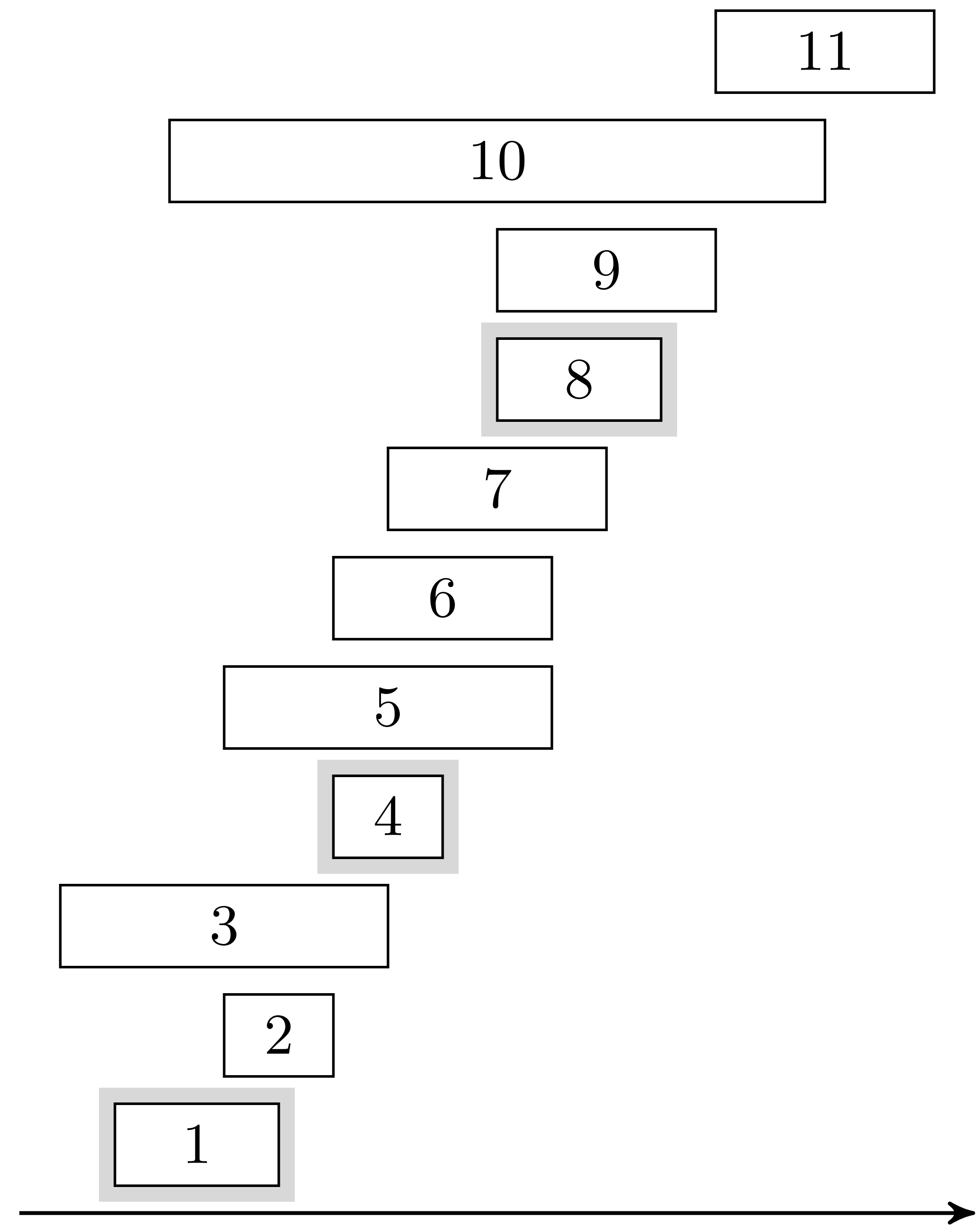
$m = k + 1$
while $m \leq n$ and $s[m] < f[k]$
    $m = m + 1$
if $m \leq n$
    S = Rec-Act-Sel(s, f, m, n)
    return $\{a_m\} \cup$ S
else return $\emptyset$



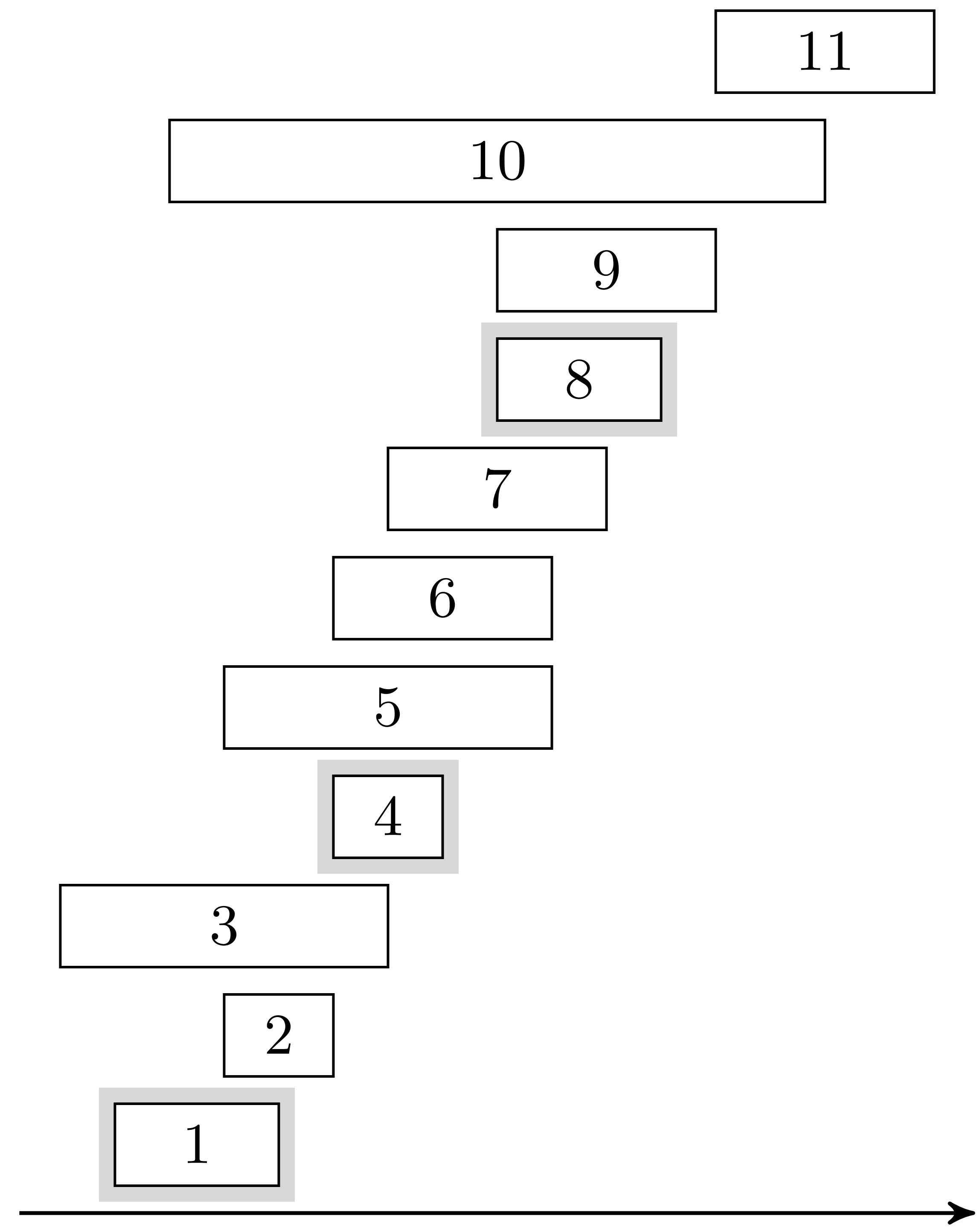
$k, m \;=\; 0, 1 \;›\; 1, 4 \;›\; \mathbf{4, 7}$

```
Rec-Act-Sel(s, f, k, n)
m = k + 1
while m ≤ n and s[m] < f[k]
        m = m + 1
if m ≤ n
        S = Rec-Act-Sel(s, f, m, n)
        return {a_m} ∪ S
else return ∅
```

$k, m \; = \; 0, 1 \;›\; 1, 4 \;›\; 4, 8$

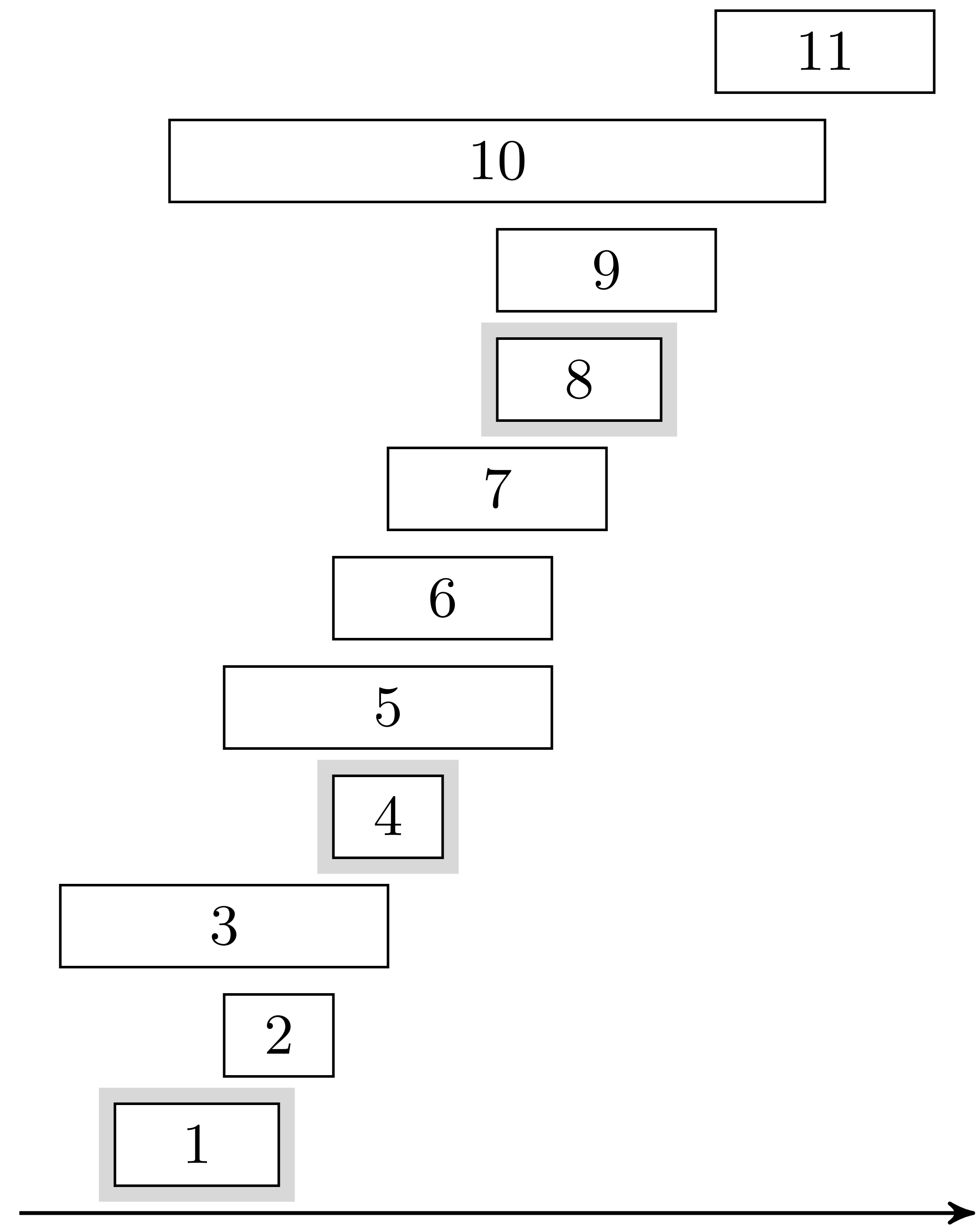


```
Rec-Act-Sel(s, f, k, n)
  m = k + 1
  while m ≤ n and s[m] < f[k]
        m = m + 1
  if m ≤ n
        S = Rec-Act-Sel(s, f, m, n)
        return {a_m} ∪ S
  else return ∅
```



$k, m \;=\; 0, 1 \;\rangle\; 1, 4 \;\rangle\; 4, 8$

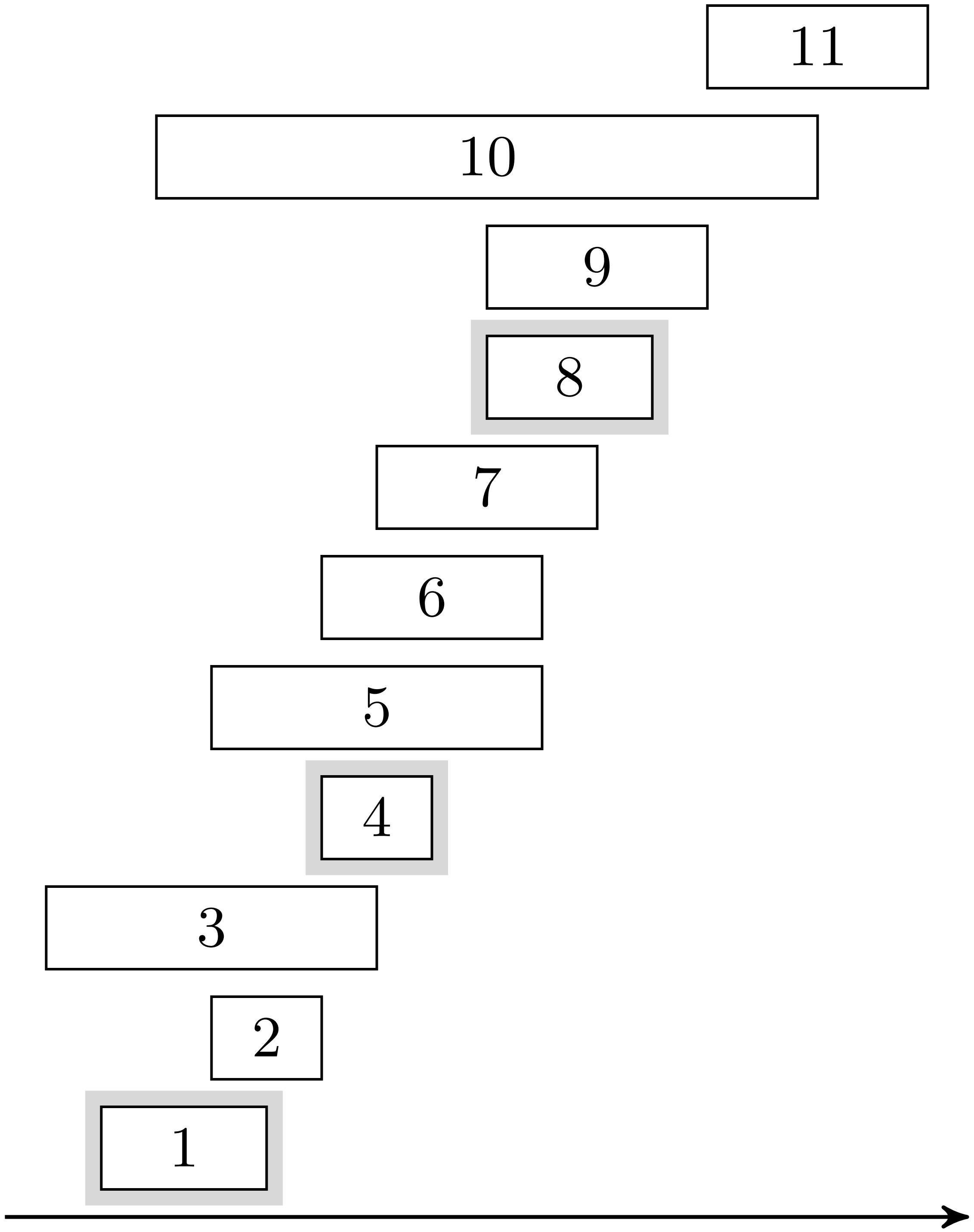
```
Rec-Act-Sel(s, f, k, n)
    m = k + 1
    while m ≤ n and s[m] < f[k]
        m = m + 1
    if m ≤ n
        S = Rec-Act-Sel(s, f, m, n)
        return {a_m} ∪ S
    else return ∅
```



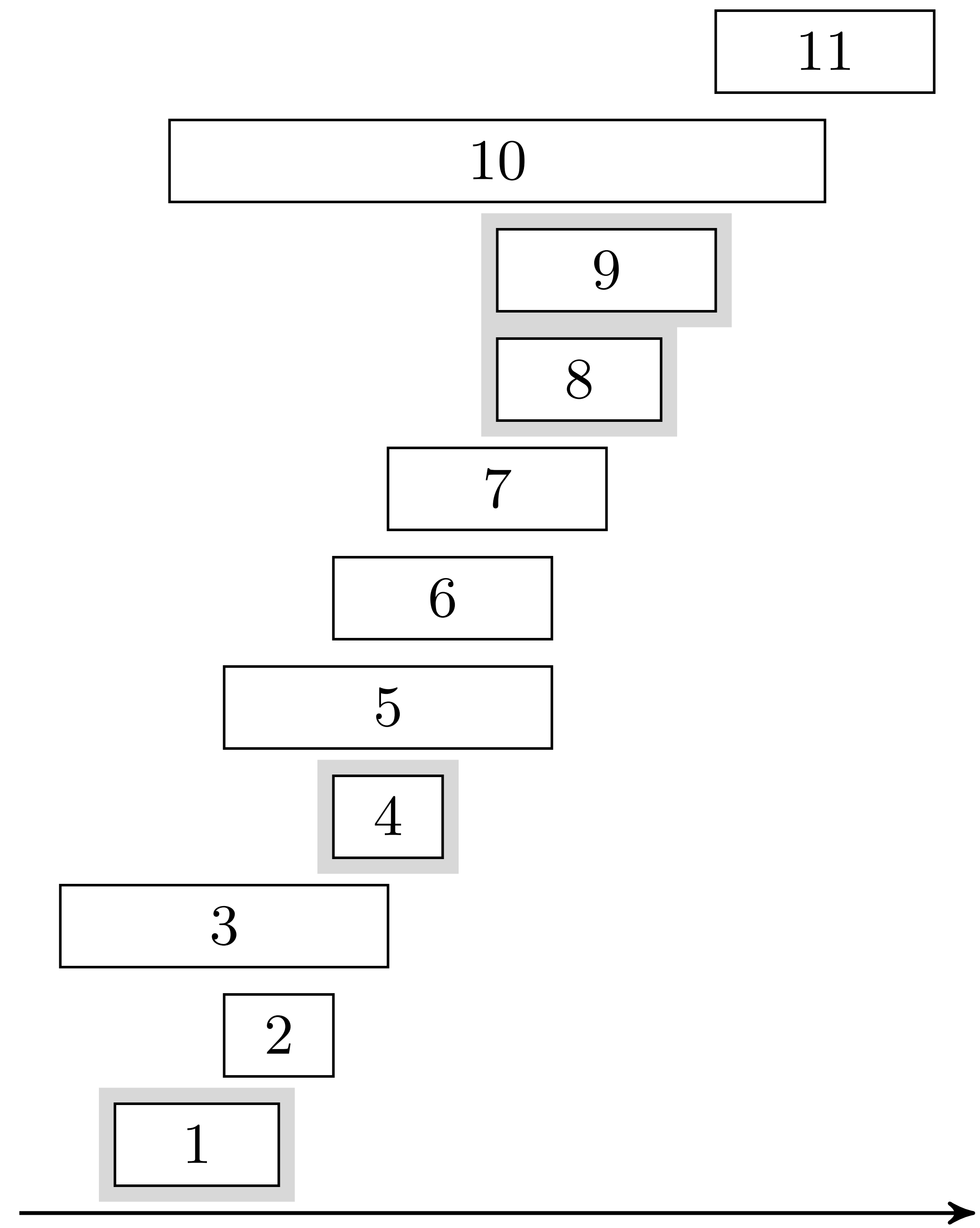
$k, m \;=\; 0, 1 \;›\; 1, 4 \;›\; 4, 8$

```
Rec-Act-Sel(s, f, k, n)
m = k + 1
while m ≤ n and s[m] < f[k]
    m = m + 1
if m ≤ n
    S = Rec-Act-Sel(s, f, m, n)
    return {a_m} ∪ S
else return ∅
```

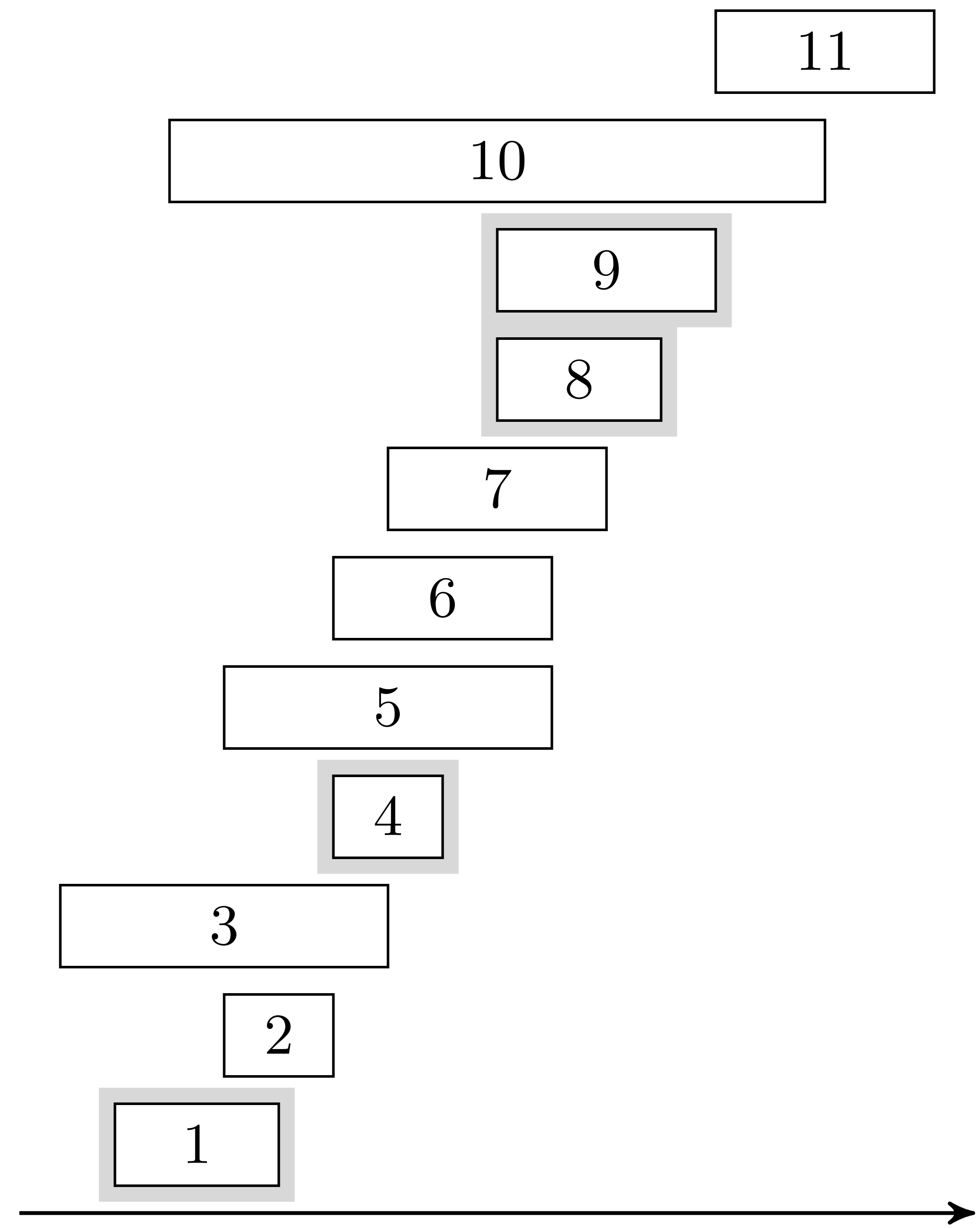
$k, m \ = \ 0, 1 \ › \ 1, 4 \ › \ 4, 8 \ › \ 8, \text{-}$



```
Rec-Act-Sel(s, f, k, n)
m = k + 1
while m ≤ n and s[m] < f[k]
      m = m + 1
if m ≤ n
      S = Rec-Act-Sel(s, f, m, n)
      return {a_m} ∪ S
else return ∅
```



$k, m \;=\; 0, 1 \;›\; 1, 4 \;›\; 4, 8 \;›\; 8, 9$



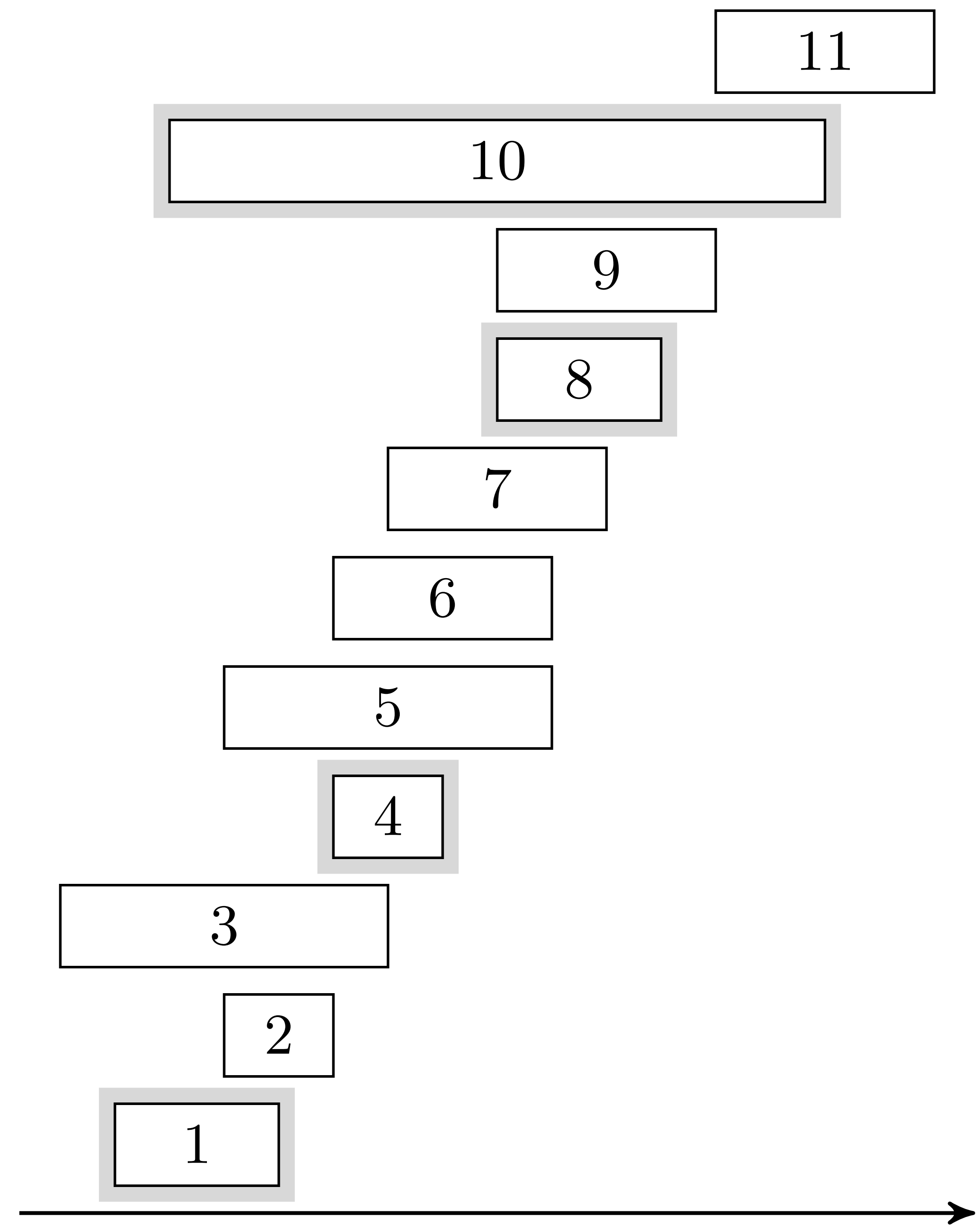
Rec-Act-Sel(s, f, k, n)

$m = k + 1$
while $m \le n$ and $s[m] < f[k]$
    $m = m + 1$
if $m \le n$
    S = Rec-Act-Sel(s, f, m, n)
    return $\{a_m\} \cup \mathrm{S}$
else return $\emptyset$

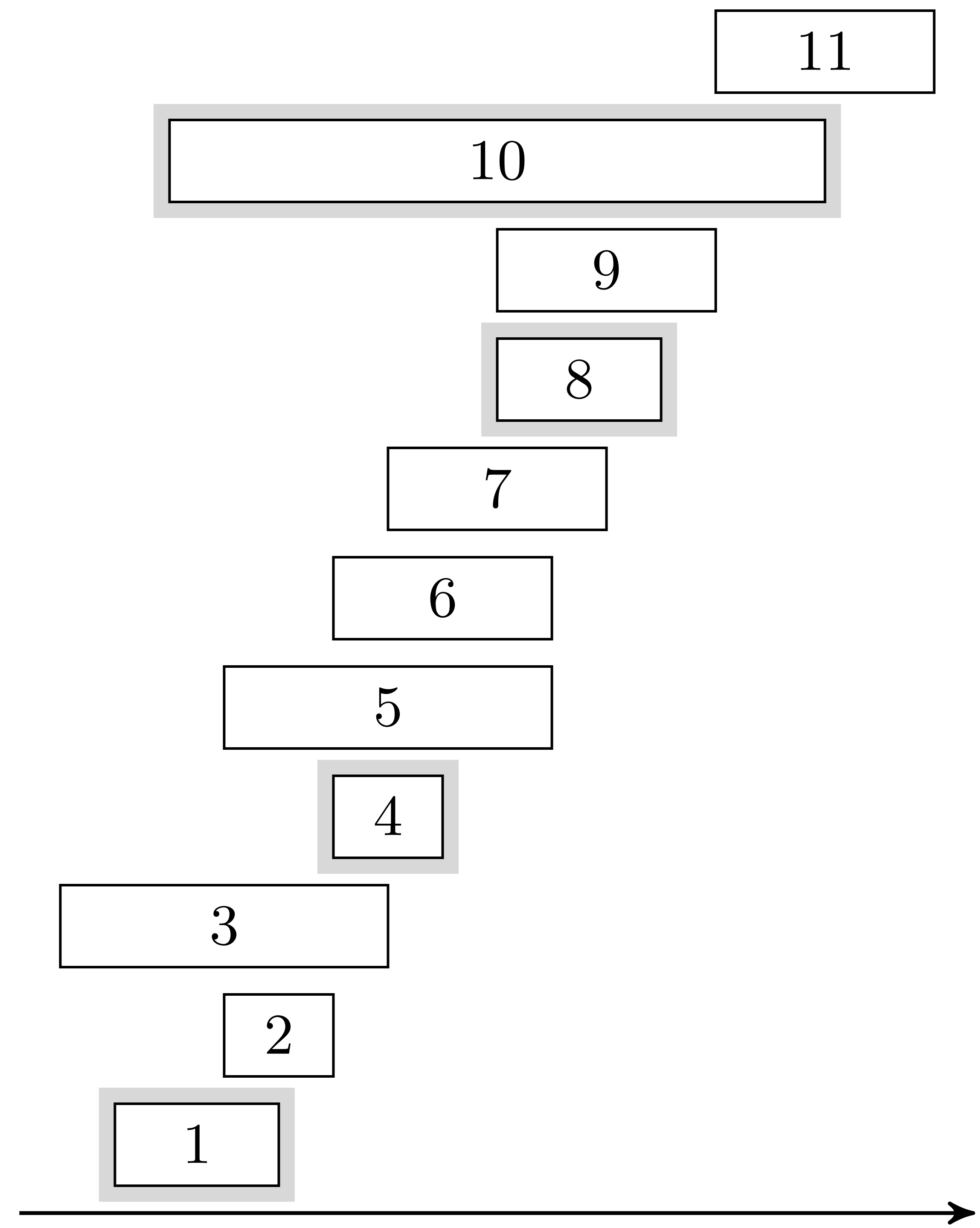
$k, m \;=\; 0, 1 \;›\; 1, 4 \;›\; 4, 8 \;›\; 8, 9$

```
Rec-Act-Sel(s, f, k, n)
m = k + 1
while m ≤ n and s[m] < f[k]
        m = m + 1
if m ≤ n
        S = Rec-Act-Sel(s, f, m, n)
        return {a_m} ∪ S
else return ∅
```

$k, m \;=\; 0, 1 \;›\; 1, 4 \;›\; 4, 8 \;›\; 8, 10$

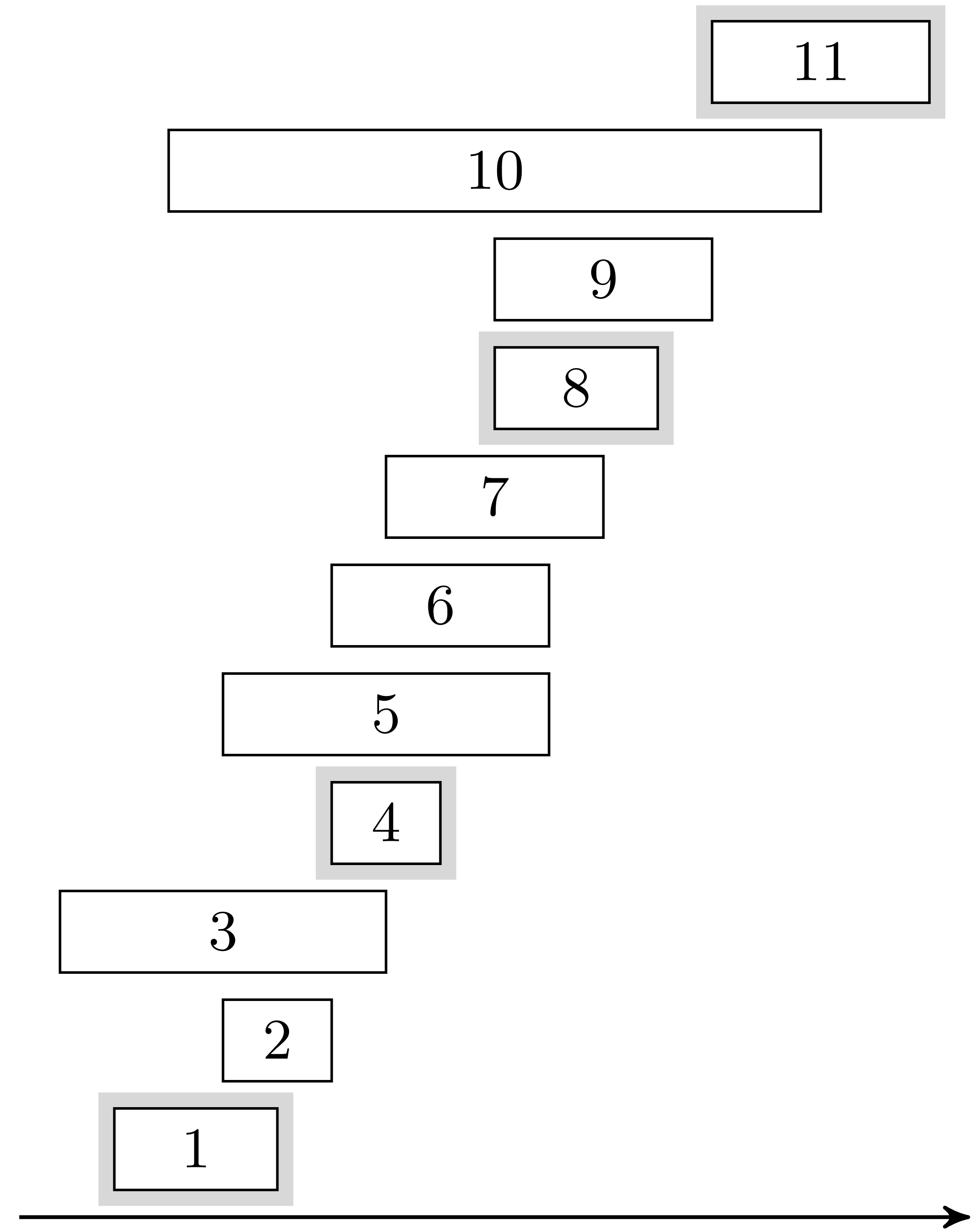


```
Rec-Act-Sel(s, f, k, n)
m = k + 1
while m ≤ n and s[m] < f[k]
      m = m + 1
if m ≤ n
      S = Rec-Act-Sel(s, f, m, n)
      return {a_m} ∪ S
else return ∅
```



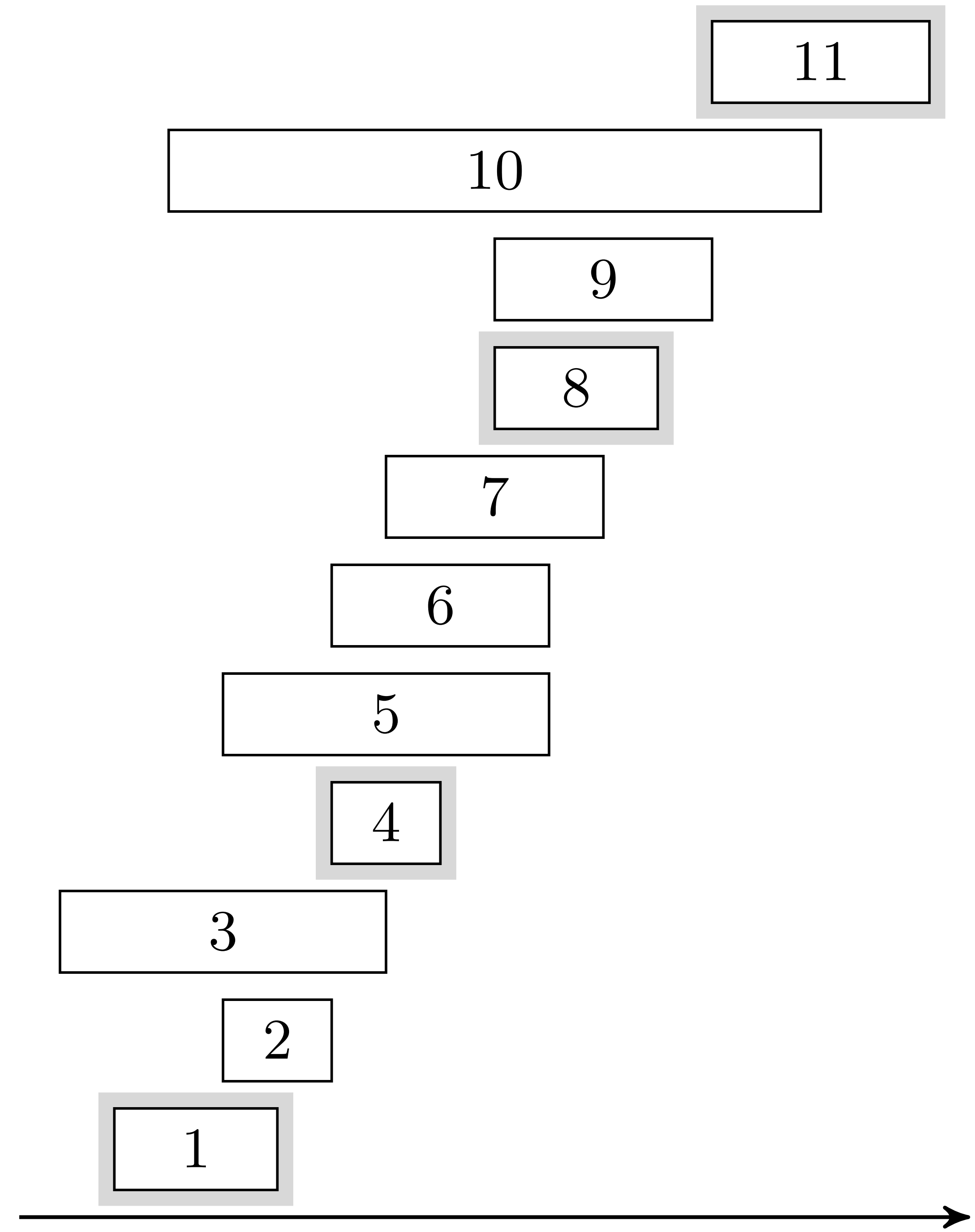
$k, m \;=\; 0, 1 \;›\; 1, 4 \;›\; 4, 8 \;›\; 8, 10$

```
Rec-Act-Sel(s, f, k, n)
  m = k + 1
  while m ≤ n and s[m] < f[k]
        m = m + 1
  if m ≤ n
        S = Rec-Act-Sel(s, f, m, n)
        return {a_m} ∪ S
  else return ∅
```



$k, m \;=\; 0, 1 \,›\, 1, 4 \,›\, 4, 8 \,›\, 8, 11$

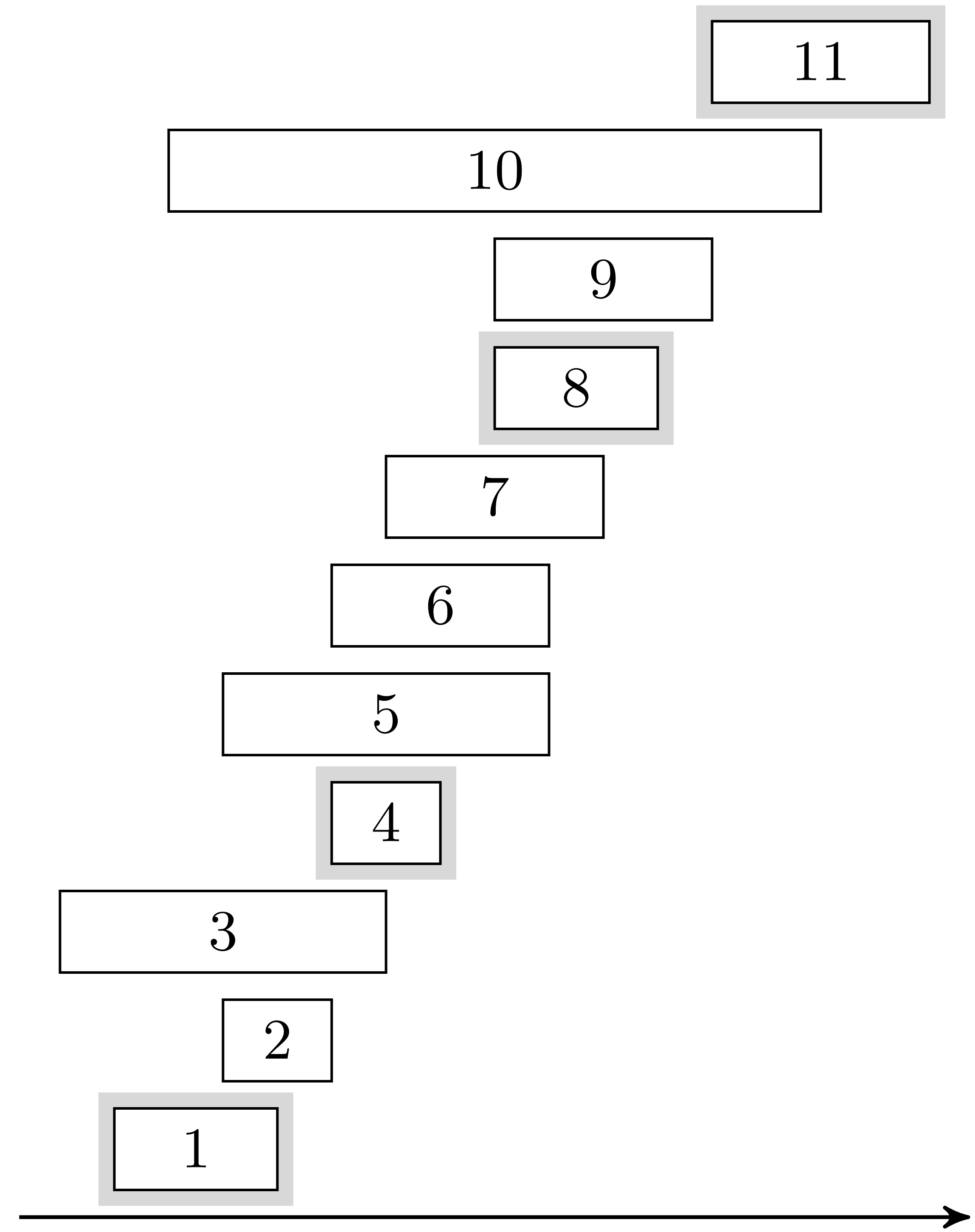
```
REC-ACT-SEL(s, f, k, n)
m = k + 1
while m ≤ n and s[m] < f[k]
        m = m + 1
if m ≤ n
        S = REC-ACT-SEL(s, f, m, n)
        return {a_m} ∪ S
else return ∅
```



$k, m \;=\; 0, 1 \;›\; 1, 4 \;›\; 4, 8 \;›\; 8, 11$

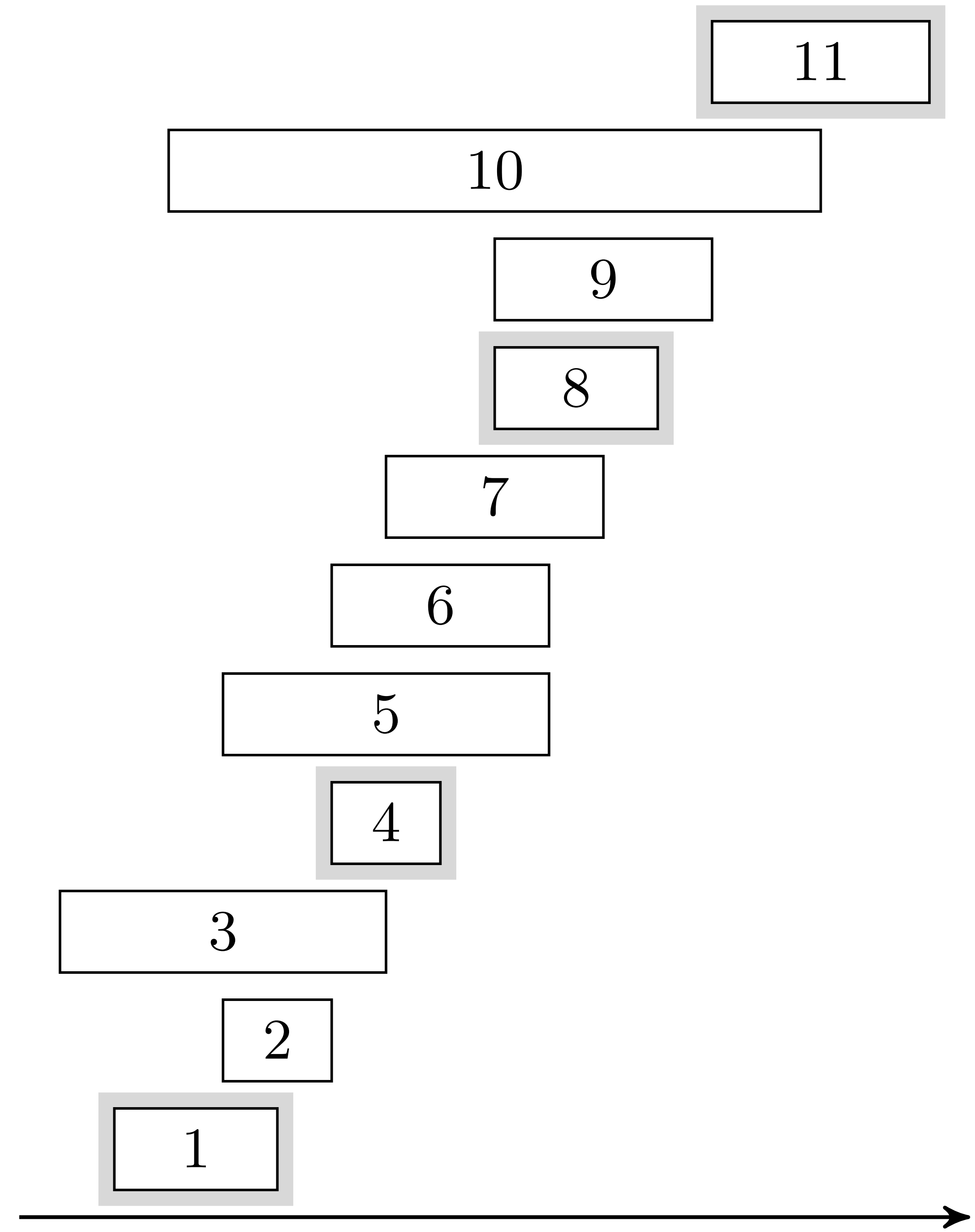
Rec-Act-Sel(s, f, k, n)

$m = k + 1$
while $m \leq n$ and $s[m] < f[k]$
    $m = m + 1$
if $m \leq n$
    $S = \text{Rec-Act-Sel}(s, f, m, n)$
    return $\{a_m\} \cup S$
else return $\emptyset$



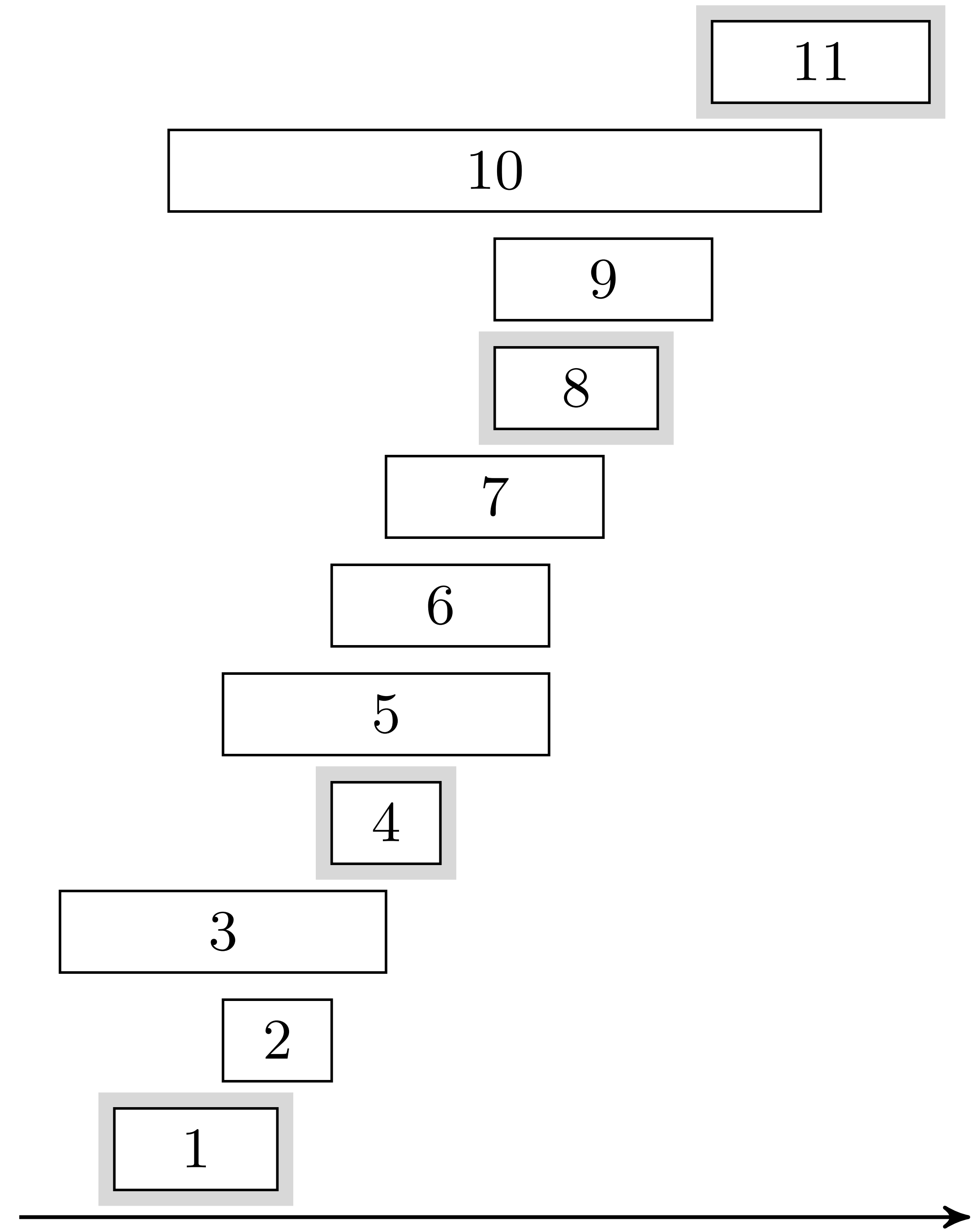
$k, m \;=\; 0, 1 \;›\; 1, 4 \;›\; 4, 8 \;›\; 8, 11$

```
Rec-Act-Sel(s, f, k, n)
m = k + 1
while m ≤ n and s[m] < f[k]
    m = m + 1
if m ≤ n
    S = Rec-Act-Sel(s, f, m, n)
    return {a_m} ∪ S
else return ∅
```



k, m = 0, 1 › 1, 4 › 4, 8 › 8, 11 › **11, –**

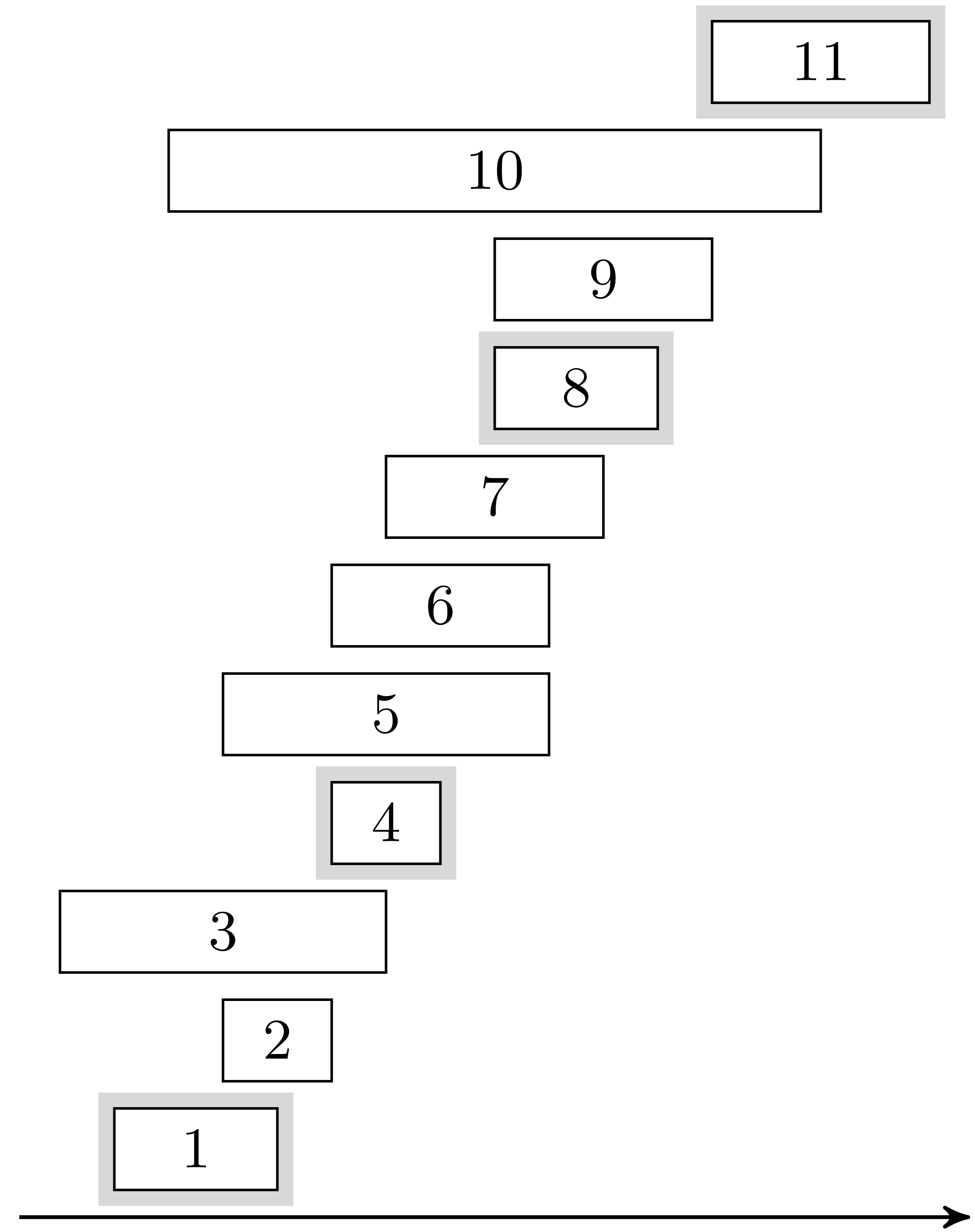
```
Rec-Act-Sel(s, f, k, n)
    m = k + 1
    while m ≤ n and s[m] < f[k]
        m = m + 1
    if m ≤ n
        S = Rec-Act-Sel(s, f, m, n)
        return {a_m} ∪ S
    else return ∅
```



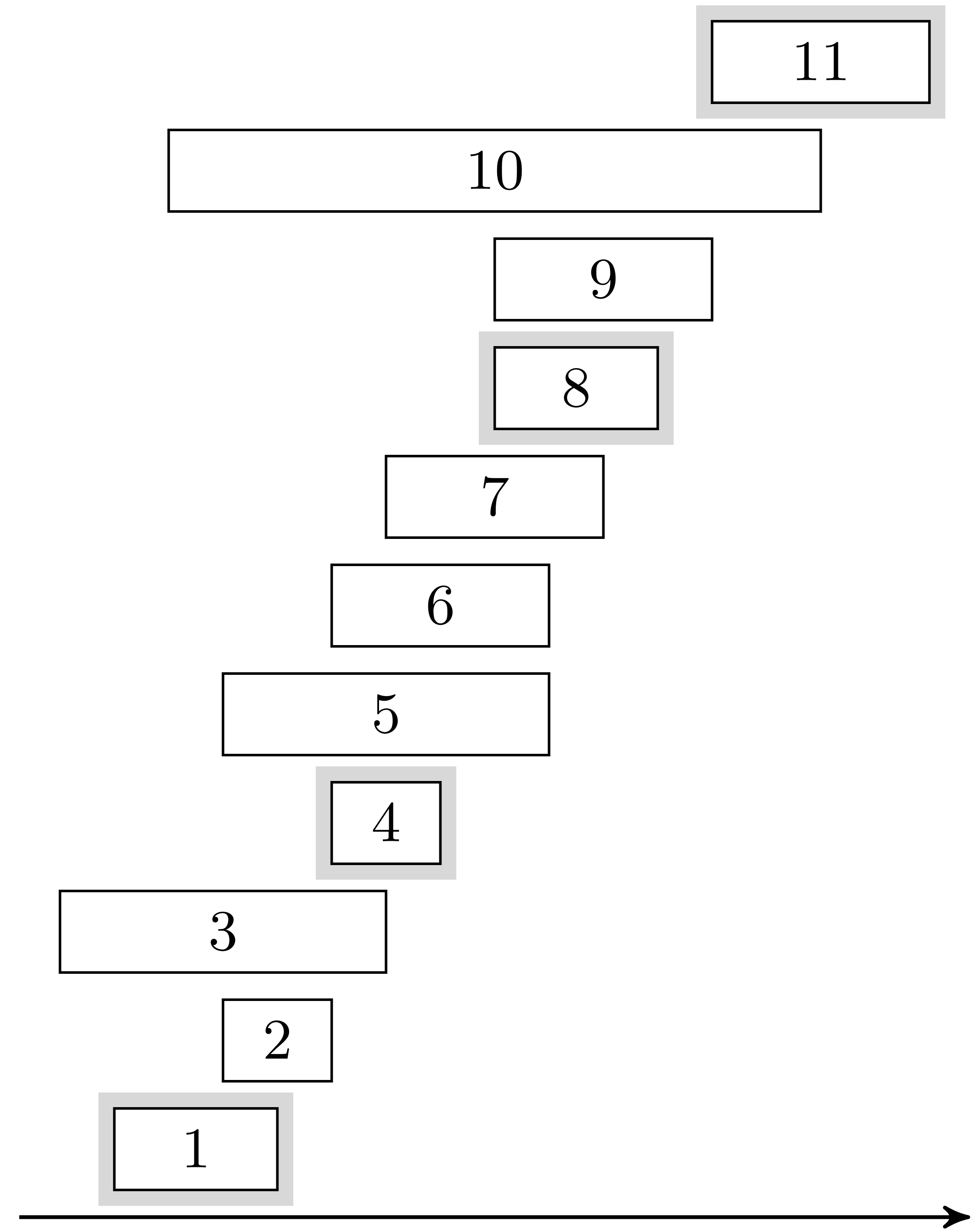
$k, m \;=\; 0, 1 \;›\; 1, 4 \;›\; 4, 8 \;›\; 8, 11 \;›\; 11, 12$

```
Rec-Act-Sel(s, f, k, n)
m = k + 1
while m ≤ n and s[m] < f[k]
        m = m + 1
if m ≤ n
        S = Rec-Act-Sel(s, f, m, n)
        return {a_m} ∪ S
else return ∅
```

$k, m \;=\; 0,1 \;›\; 1,4 \;›\; 4,8 \;›\; 8,11 \;›\; 11,12$

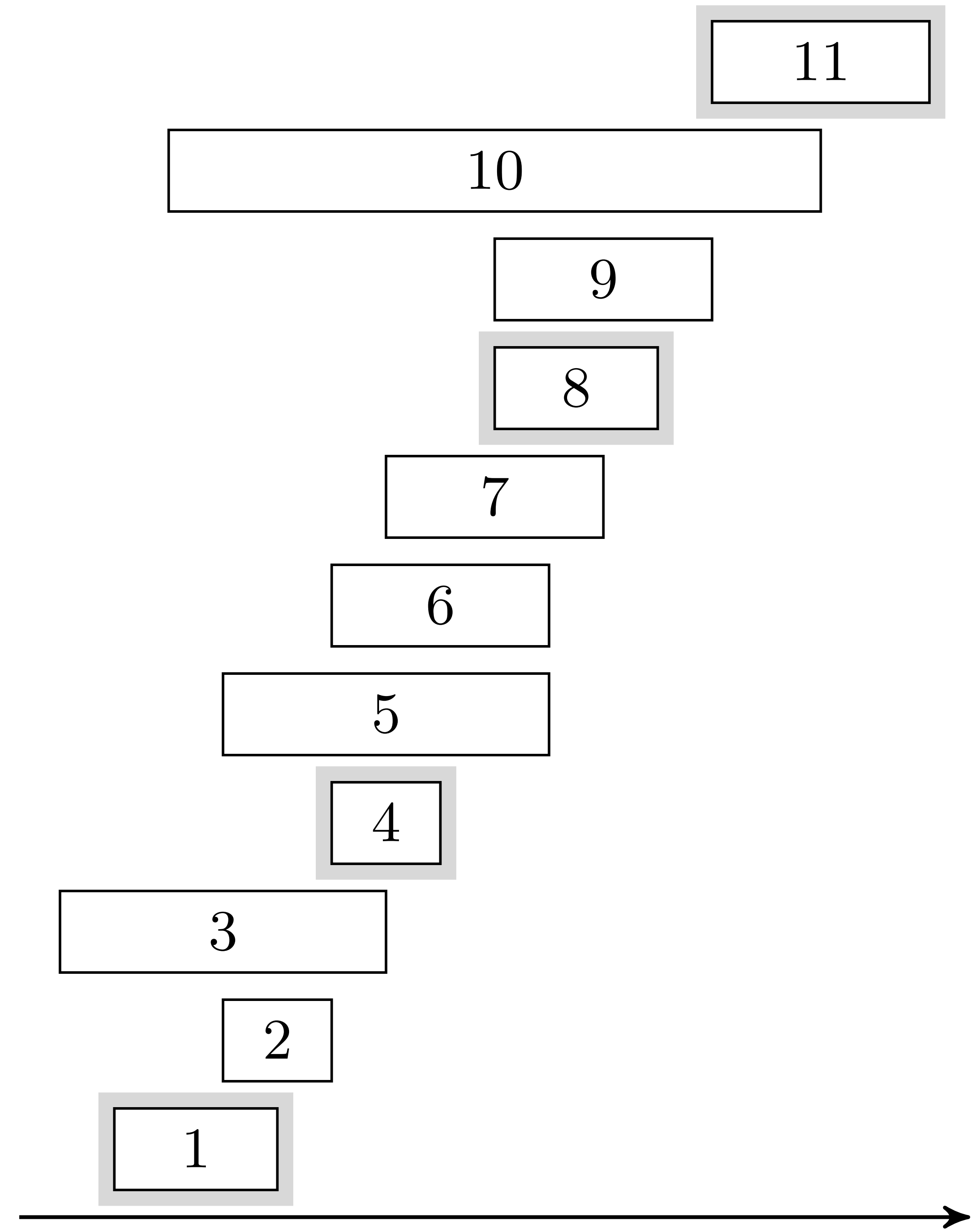


```
REC-ACT-SEL(s, f, k, n)
m = k + 1
while m ≤ n and s[m] < f[k]
      m = m + 1
if m ≤ n
      S = REC-ACT-SEL(s, f, m, n)
      return {a_m} ∪ S
else return ∅
```



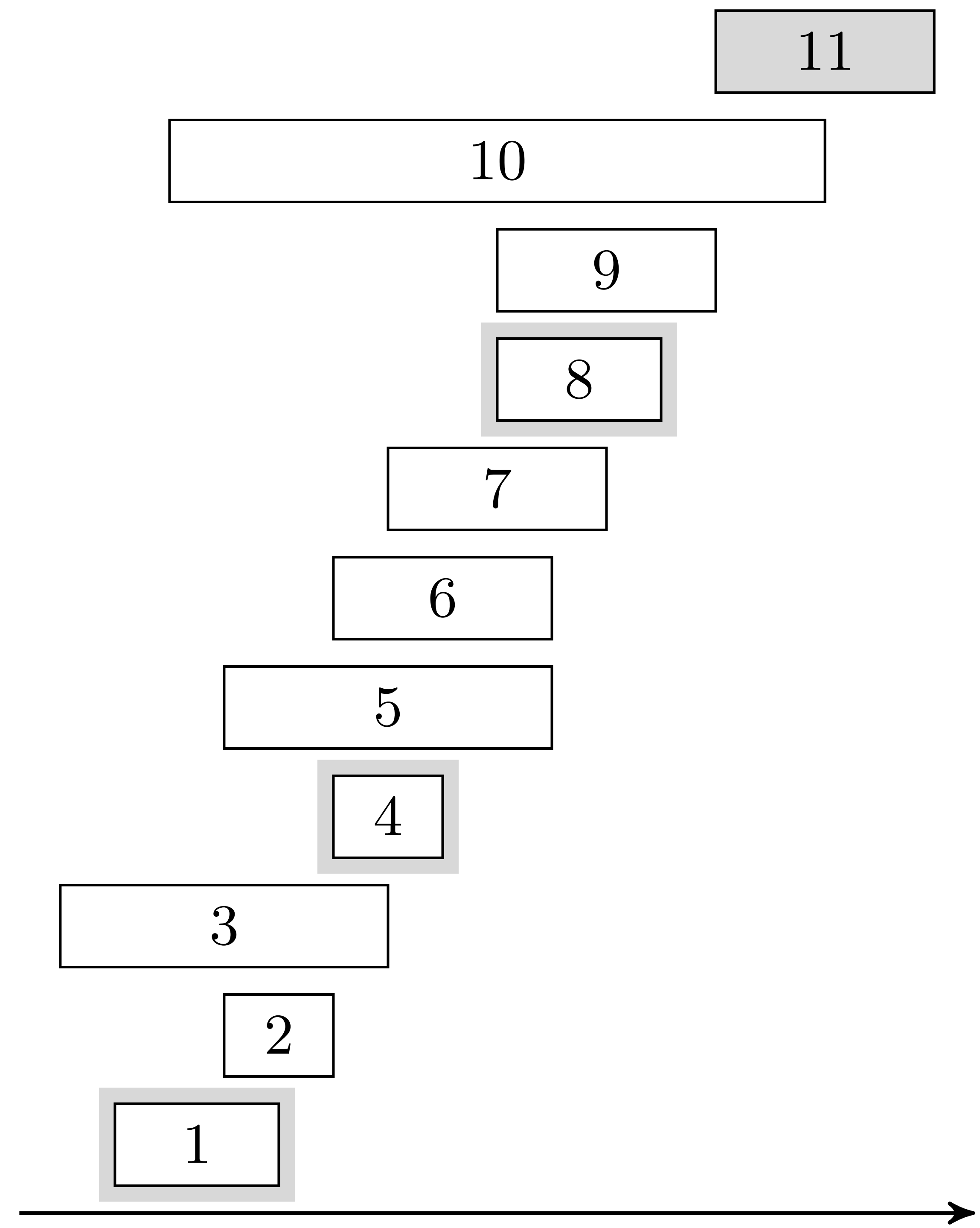
$k, m \; = \; 0,1 \;›\; 1,4 \;›\; 4,8 \;›\; 8,11 \;›\; 11,12$

```
REC-ACT-SEL(s, f, k, n)
m = k + 1
while m ≤ n and s[m] < f[k]
    m = m + 1
if m ≤ n
    S = REC-ACT-SEL(s, f, m, n)
    return {a_m} ∪ S
else return ∅
```

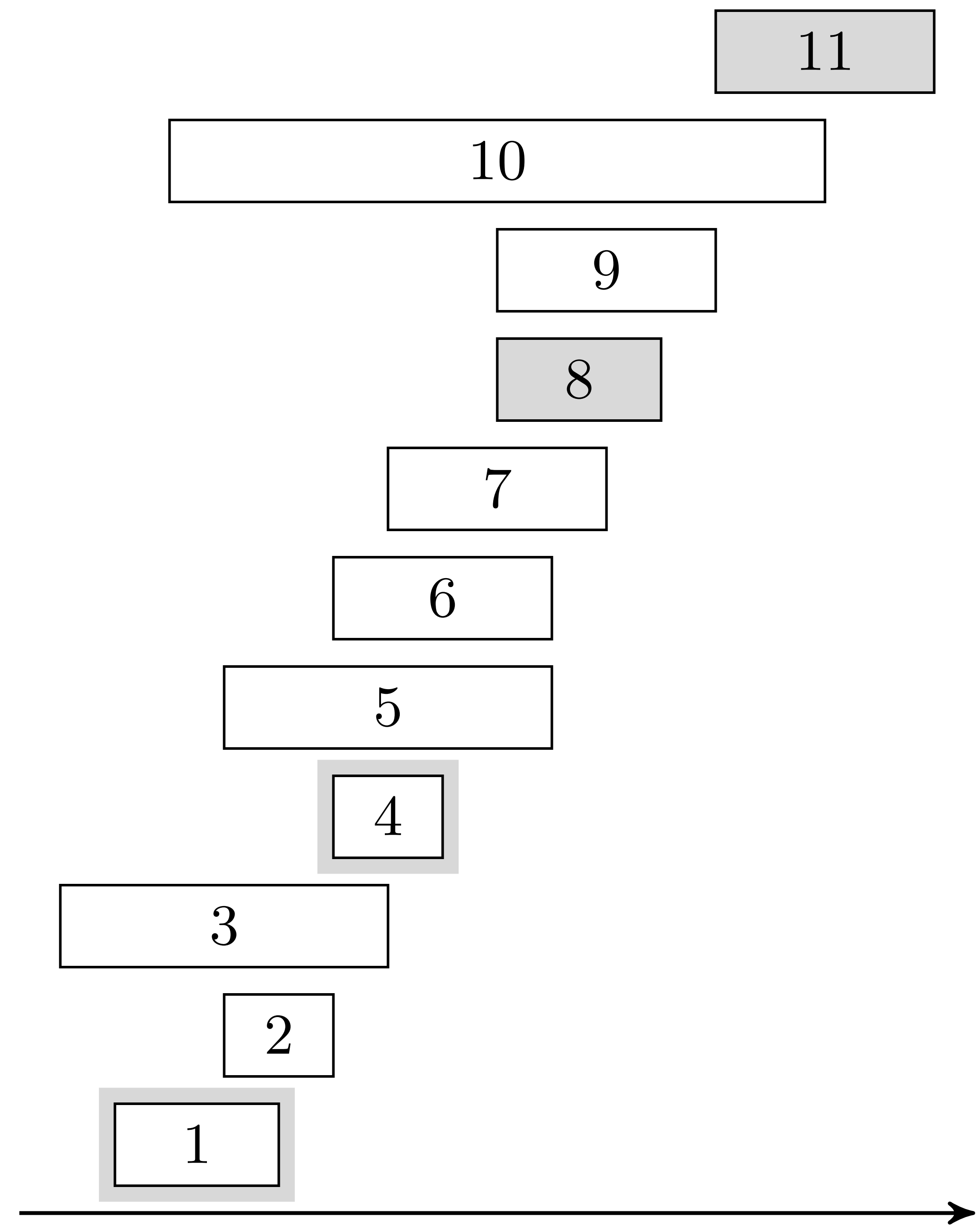


$k, m \;=\; 0, 1 \;›\; 1, 4 \;›\; 4, 8 \;›\; 8, 11$

```
Rec-Act-Sel(s, f, k, n)
m = k + 1
while m ≤ n and s[m] < f[k]
        m = m + 1
if m ≤ n
        S = Rec-Act-Sel(s, f, m, n)
        return {a_m} ∪ S
else return ∅
```



$k, m \;=\; 0, 1 \;›\; 1, 4 \;›\; 4, 8$

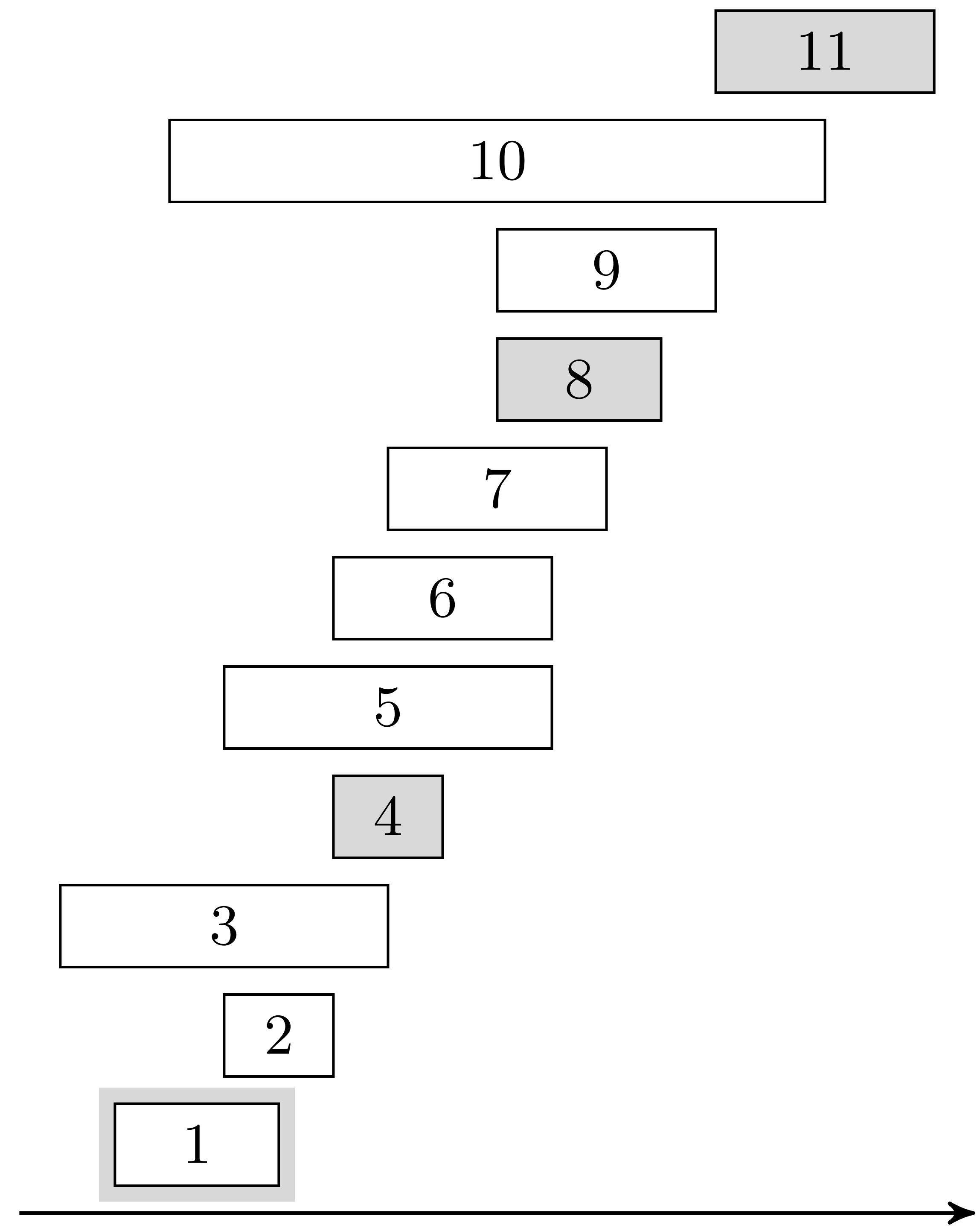



```
Rec-Act-Sel(s, f, k, n)
m = k + 1
while m ≤ n and s[m] < f[k]
    m = m + 1
if m ≤ n
    S = Rec-Act-Sel(s, f, m, n)
    return {a_m} ∪ S
else return ∅
```

$k, m \;=\; 0, 1 \;›\; 1, 4$

```
REC-ACT-SEL(s, f, k, n)
  m = k + 1
  while m ≤ n and s[m] < f[k]
        m = m + 1
  if m ≤ n
        S = REC-ACT-SEL(s, f, m, n)
        return {a_m} ∪ S
  else return ∅
```

$k, m \ = \ 0, 1$



Rec-Act-Sel(s, f, k, n)

$m = k + 1$

while $m \leq n$ and $s[m] < f[k]$

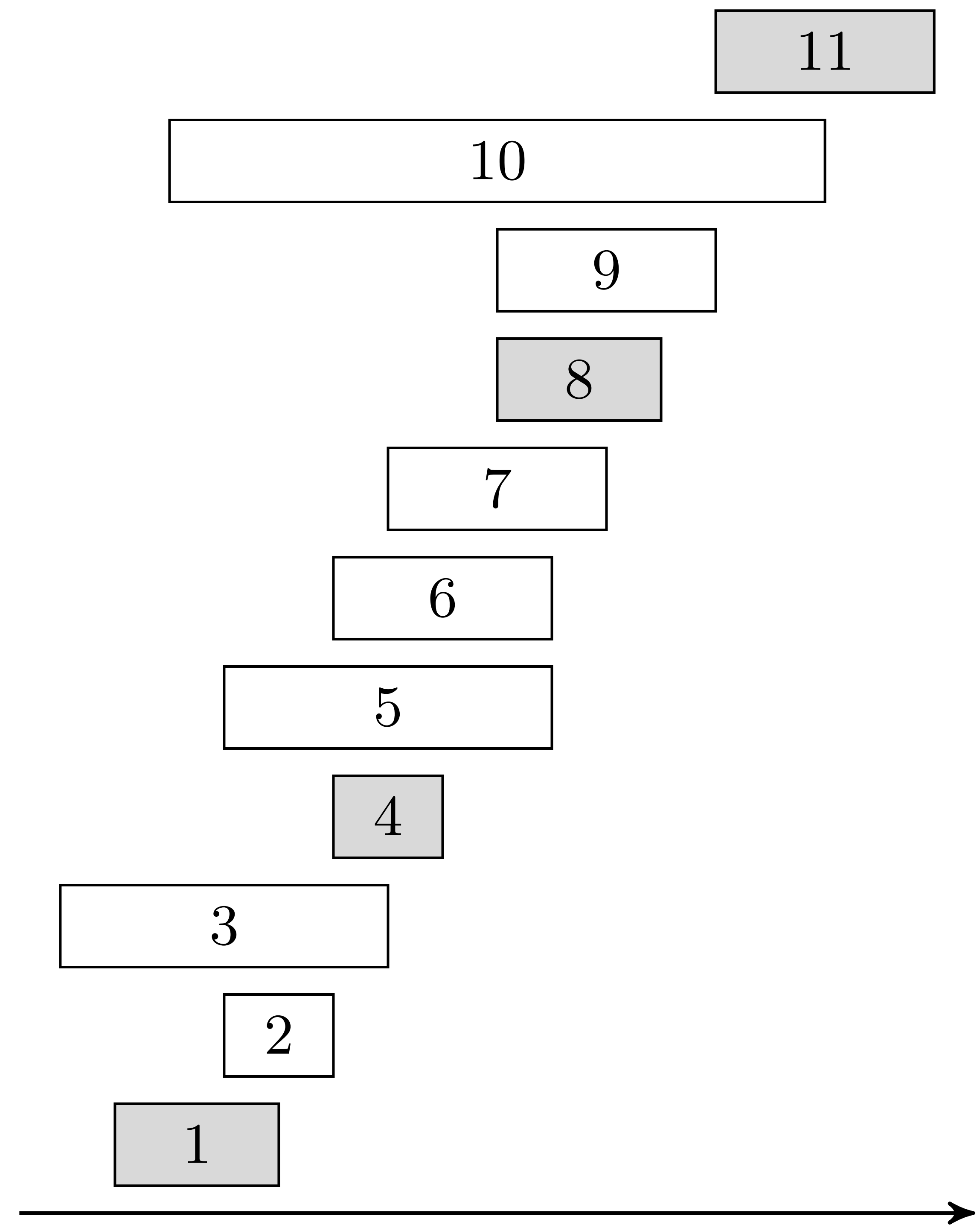
  $m = m + 1$

if $m \leq n$

  S = Rec-Act-Sel(s, f, m, n)

  return $\{a_m\} \cup$ S

else return $\emptyset$



GREEDY-ACTIVITY-SELECTOR(s, f)

$s[i]$ start, a_i
$f[i]$ slutt, a_i

Iterativ omskriving av REC-ACT-SEL

GREEDY-ACTIVITY-SELECTOR(s, f)
1 $n = s.length$

$s[i]$ start, a_i
$f[i]$ slutt, a_i

n antall

GREEDY-ACTIVITY-SELECTOR(s, f)
1 $n = s.length$
2 $A = \{a_1\}$

$s[i]$ start, a_i
$f[i]$ slutt, a_i

n antall
A løsning

Det lønner seg alltid å ta med første element

$\textsc{Greedy-Activity-Selector}(s, f)$

$n = s.length$

$\mathrm{A} = \{a_1\}$

$k = 1$

$s[i]$	start, a_i
$f[i]$	slutt, a_i
a_k	forrige
n	antall
A	løsning

mlh@ntnu.no

Greedy-Activity-Selector(s, f)

$n = s.length$

$A = \{a_1\}$

$k = 1$

for $m = 2$ **to** n

$s[i]$	start, a_i
$f[i]$	slutt, a_i
a_k	forrige
a_m	neste
n	antall
A	løsning

Finn første lovlige (sortert etter f)

GREEDY-ACTIVITY-SELECTOR(s, f)

```
n = s.length
A = {a_1}
k = 1
for m = 2 to n
        if s[m] ≥ f[k]
```

$s[i]$	start, a_i
$f[i]$	slutt, a_i
a_k	forrige
a_m	neste
n	antall
A	løsning

Vi fant et lovlig intervall

GREEDY-ACTIVITY-SELECTOR(s, f)

```
1  n = s.length
2  A = {a_1}
3  k = 1
4  for m = 2 to n
5      if s[m] ≥ f[k]
6          A = A ∪ {a_m}
```

$s[i]$	start, a_i
$f[i]$	slutt, a_i
a_k	forrige
a_m	neste
n	antall
A	løsning

Legg det til i løsningen

GREEDY-ACTIVITY-SELECTOR(s, f)

$n = s.length$
$\mathrm{A} = \{a_1\}$
$k = 1$
for $m = 2$ **to** n
    if $s[m] \geq f[k]$
        $\mathrm{A} = \mathrm{A} \cup \{a_m\}$
        $k = m$

$s[i]$	start, a_i
$f[i]$	slutt, a_i
a_k	forrige
a_m	neste
n	antall
A	løsning

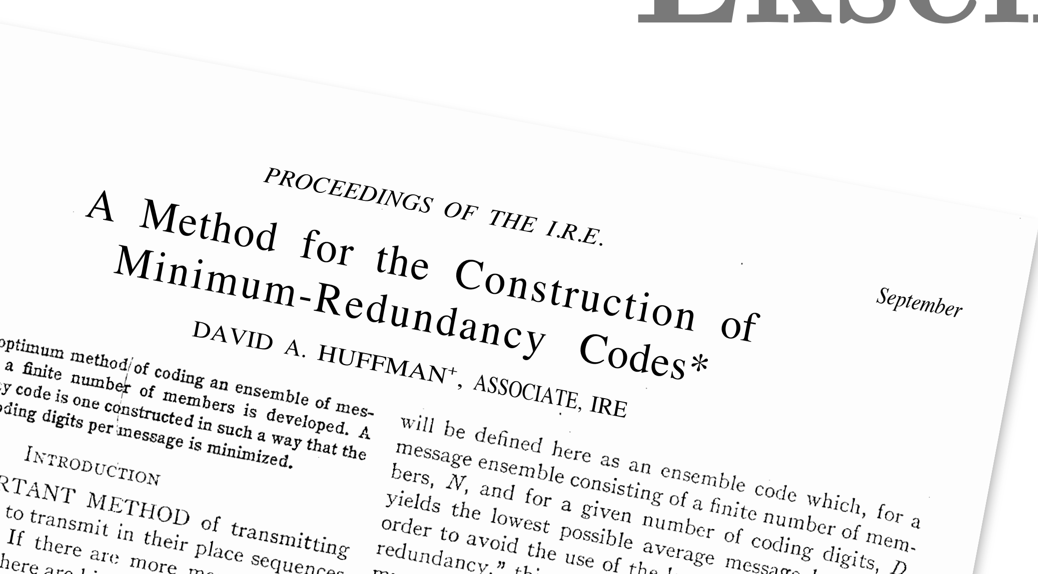
Dette er nå det som slutter sist av de valgte

```
GREEDY-ACTIVITY-SELECTOR(s, f)
n = s.length
A = {a_1}
k = 1
for m = 2 to n
        if s[m] ≥ f[k]
                A = A ∪ {a_m}
                k = m
return A
```

$s[i]$	start, a_i
$f[i]$	slutt, a_i
a_k	forrige
a_m	neste
n	antall
A	løsning

4:4

Eksempel: Huffman



PROCEEDINGS OF THE I.R.E. September

A Method for the Construction of Minimum-Redundancy Codes*

DAVID A. HUFFMAN+, ASSOCIATE, IRE

INTRODUCTION

Fra 1952

Input: Alfabet $\mathrm{C} = \{c, \ldots\}$ med frekvenser $c.\mathit{freq}$.

Output: Binær koding som minimerer forventet kodelengde $\sum_{c \in \mathrm{C}} (c.\mathit{freq} \cdot \mathit{length}(\mathit{code}(c)))$.

- **Vil lage binære koder for tegn**
- **Tegnene har frekvenser**
- **Kodene kan ha varierende lengde**
- **Vil minimere forventet kodelengde**
- **Prefiks-kode: Ingen koder er prefiks av andre. Kan representeres som stier i binærtre, med tegn som løvnoder**

- **La oss prøve å lage en grådig algoritme**
- **Vi kan «slå sammen» to partielle løsninger ved å la én bit velge mellom dem**
- **Grådighet: Slå alltid sammen de sjeldneste, siden den ekstra bit-en da koster minst**

HUFFMAN(C)

C frekvenser

Finn kode for tegn i C som minimerer forventet tekstlengde

$\textsc{Huffman}(C)$

1 $n = |C|$

C frekvenser

n antall

Huffman(C)

$n = |C|$
$Q = C$

C frekvenser
n antall
Q pri-kø

Q er en prioritetskø, basert på frekvens: Sjeldne tegn først!

```
HUFFMAN(C)
n = |C|
Q = C
for i = 1 to n − 1
```

C	frekvenser
n	antall
Q	pri-kø
i	iterasjon

Vi har n løvnoder; trenger $n - 1$ interne

HUFFMAN(C)
1 $n = |C|$
2 $Q = C$
3 **for** $i = 1$ **to** $n - 1$
4 allocate a new node z

C frekvenser
n antall
Q pri-kø
i iterasjon
z ny rot

HUFFMAN(C)

$n = |C|$

$Q = C$

for $i = 1$ **to** $n - 1$

  allocate a new node z

  $x =$ EXTRACT-MIN(Q)

C	frekvenser
n	antall
Q	pri-kø
i	iterasjon
z	ny rot
x	v. barn

HUFFMAN(C)

$n = |\mathrm{C}|$
$\mathrm{Q} = \mathrm{C}$
for $i = 1$ **to** $n - 1$
    allocate a new node z
    $x = \textsc{Extract-Min}(\mathrm{Q})$
    $y = \textsc{Extract-Min}(\mathrm{Q})$

C	frekvenser
n	antall
Q	pri-kø
i	iterasjon
z	ny rot
x	v. barn
y	h. barn

HUFFMAN(C)

$n = |C|$
$Q = C$
for $i = 1$ **to** $n - 1$
    allocate a new node z
    $x = \text{EXTRACT-MIN}(Q)$
    $y = \text{EXTRACT-MIN}(Q)$
    $z.left, z.right = x, y$

C	frekvenser
n	antall
Q	pri-kø
i	iterasjon
z	ny rot
x	v. barn
y	h. barn

Slå sammen de to minste; gi dem felles foreldrenode

Huffman(C)

$n = |\mathrm{C}|$
$\mathrm{Q} = \mathrm{C}$
for $i = 1$ **to** $n - 1$
    allocate a new node z
    $x = \textsc{Extract-Min}(\mathrm{Q})$
    $y = \textsc{Extract-Min}(\mathrm{Q})$
    $z.\mathit{left}, z.\mathit{right} = x, y$
    $z.\mathit{freq} = x.\mathit{freq} + y.\mathit{freq}$

C frekvenser
n antall
Q pri-kø
i iterasjon
z ny rot
x v. barn
y h. barn

Vi later som om deltreet er ett tegn

HUFFMAN(C)

$n = |C|$
$Q = C$
for $i = 1$ **to** $n - 1$
    allocate a new node z
    $x = \text{EXTRACT-MIN}(Q)$
    $y = \text{EXTRACT-MIN}(Q)$
    $z.\mathit{left}, z.\mathit{right} = x, y$
    $z.\mathit{freq} = x.\mathit{freq} + y.\mathit{freq}$
    $\text{INSERT}(Q, z)$

C	frekvenser
n	antall
Q	pri-kø
i	iterasjon
z	ny rot
x	v. barn
y	h. barn

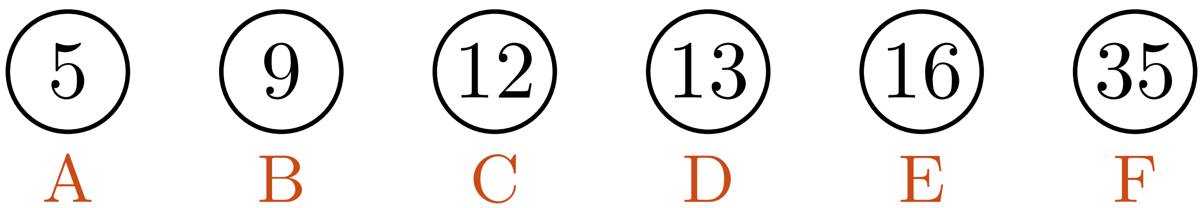
Sett dette «tegnet» tilbake i køen

```
HUFFMAN(C)
n = |C|
Q = C
for i = 1 to n − 1
    allocate a new node z
    x = EXTRACT-MIN(Q)
    y = EXTRACT-MIN(Q)
    z.left, z.right = x, y
    z.freq = x.freq + y.freq
    INSERT(Q, z)
return EXTRACT-MIN(Q)
```

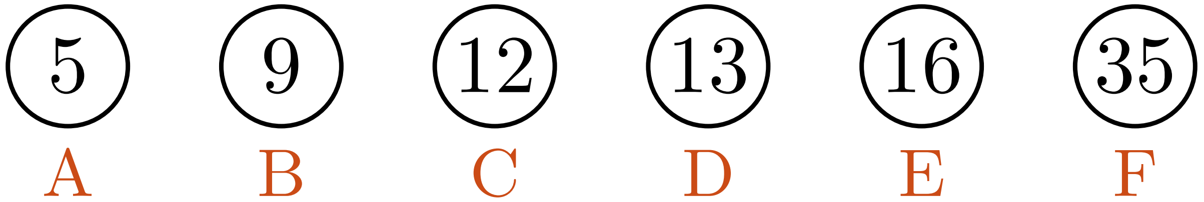
C frekvenser
n antall
Q pri-kø
i iterasjon
z ny rot
x v. barn
y h. barn

Til slutt har vi bare én node i Q: Rota i treet

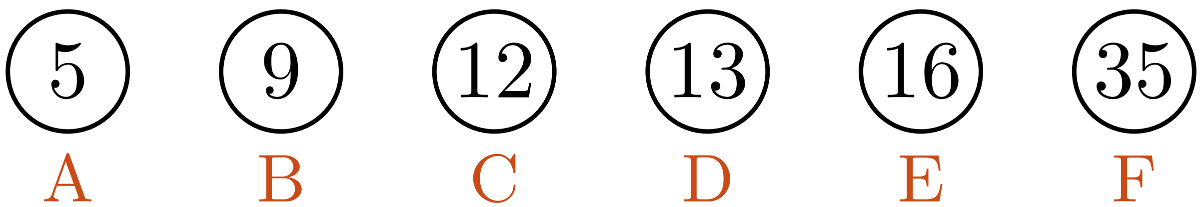
```
Huffman(C)
n = |C|
Q = C
for i = 1 to n − 1
        allocate a new node z
        x = Extract-Min(Q)
        y = Extract-Min(Q)
        z.left, z.right = x, y
        z.freq = x.freq + y.freq
        Insert(Q, z)
return Extract-Min(Q)
```



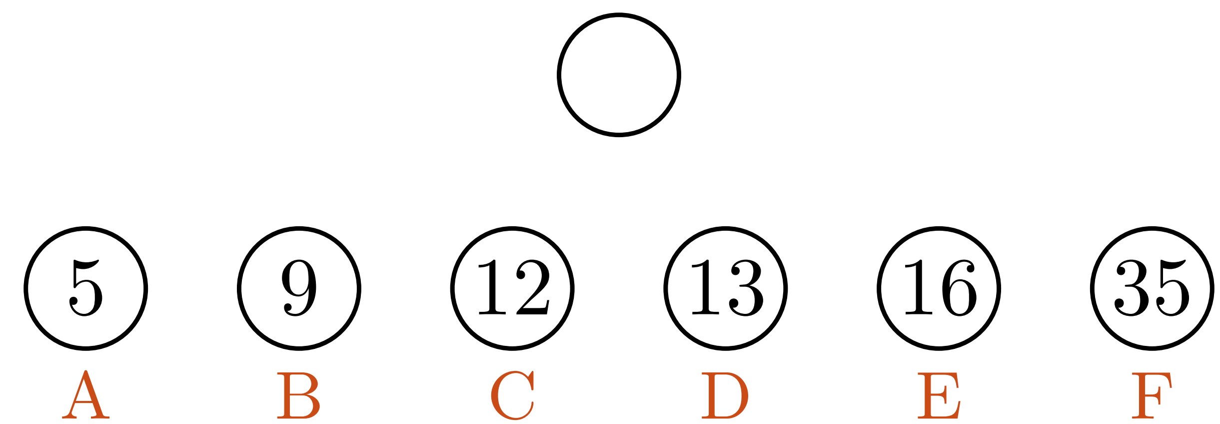
```
HUFFMAN(C)
n = |C|
Q = C
for i = 1 to n − 1
    allocate a new node z
    x = EXTRACT-MIN(Q)
    y = EXTRACT-MIN(Q)
    z.left, z.right = x, y
    z.freq = x.freq + y.freq
    INSERT(Q, z)
return EXTRACT-MIN(Q)
```



```
Huffman(C)
n = |C|
Q = C
for i = 1 to n − 1
    allocate a new node z
    x = Extract-Min(Q)
    y = Extract-Min(Q)
    z.left, z.right = x, y
    z.freq = x.freq + y.freq
    Insert(Q, z)
return Extract-Min(Q)
```

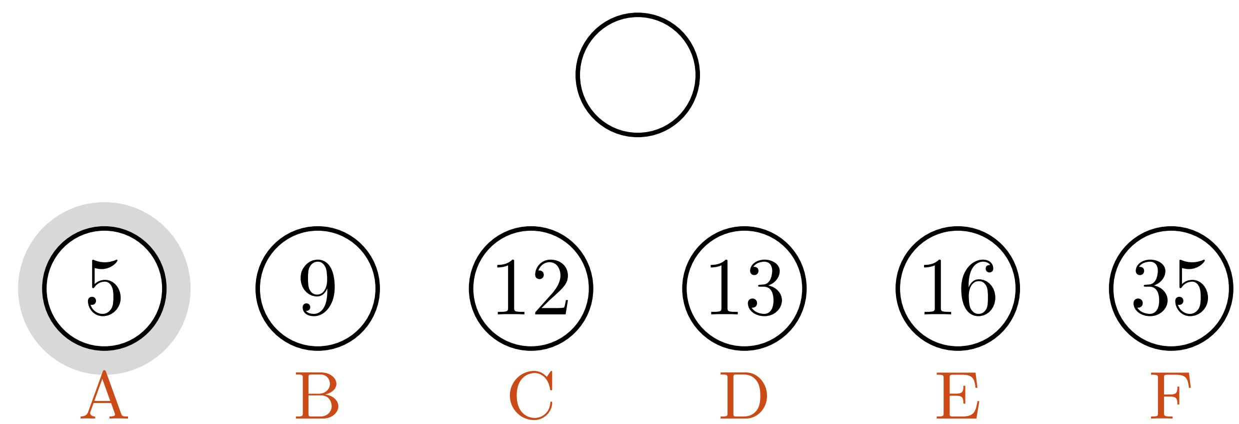


```
Huffman(C)
n = |C|
Q = C
for i = 1 to n − 1
    allocate a new node z
    x = Extract-Min(Q)
    y = Extract-Min(Q)
    z.left, z.right = x, y
    z.freq = x.freq + y.freq
    Insert(Q, z)
return Extract-Min(Q)
```

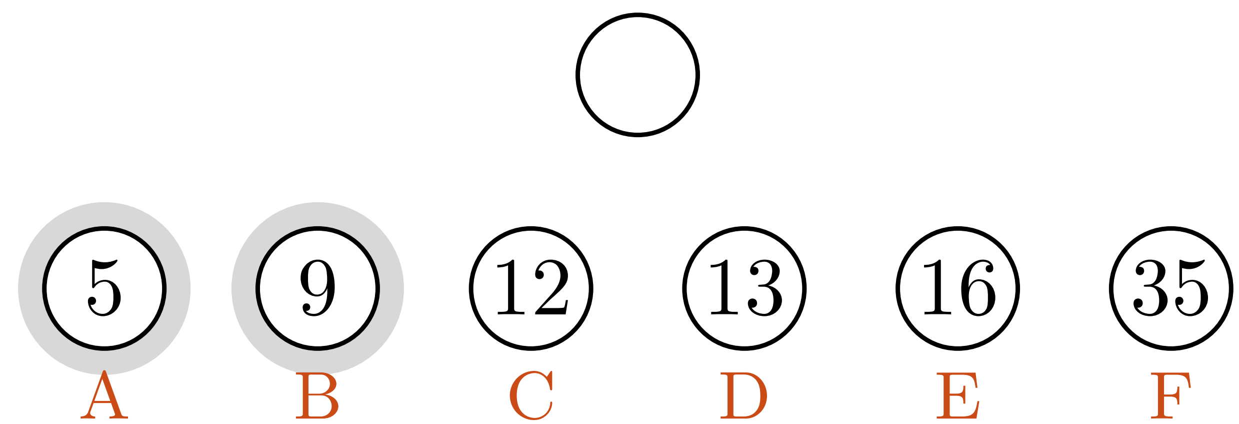


HUFFMAN(C)

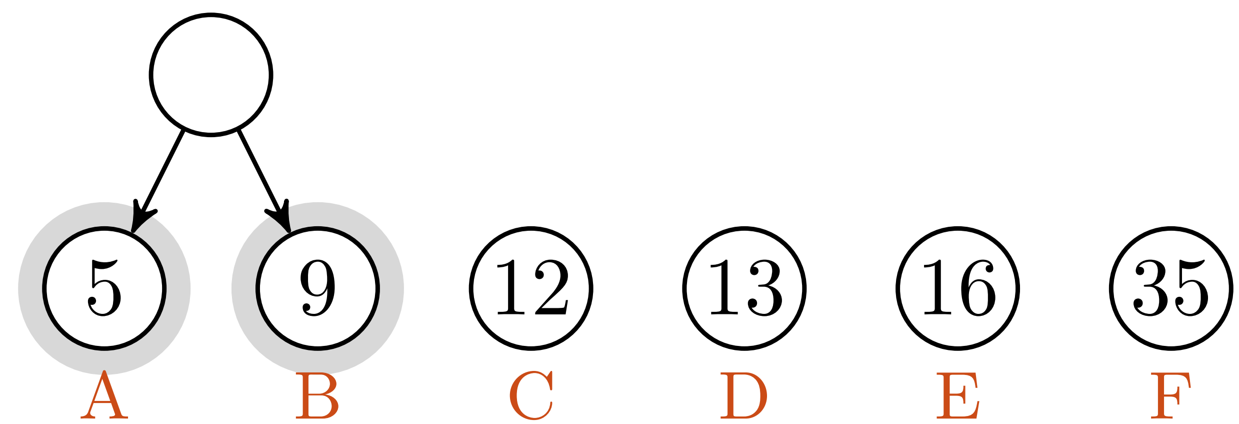
$n = |C|$
$Q = C$
for $i = 1$ **to** $n - 1$
    allocate a new node z
    $x = \text{EXTRACT-MIN}(Q)$
    $y = \text{EXTRACT-MIN}(Q)$
    $z.\mathit{left}, z.\mathit{right} = x, y$
    $z.\mathit{freq} = x.\mathit{freq} + y.\mathit{freq}$
    $\text{INSERT}(Q, z)$
return $\text{EXTRACT-MIN}(Q)$



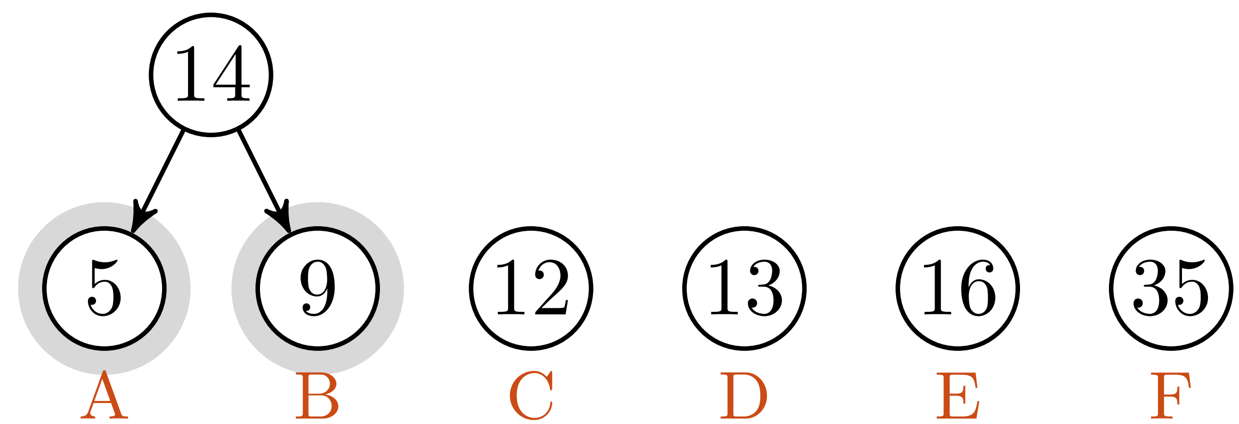
```
HUFFMAN(C)
n = |C|
Q = C
for i = 1 to n − 1
    allocate a new node z
    x = EXTRACT-MIN(Q)
    y = EXTRACT-MIN(Q)
    z.left, z.right = x, y
    z.freq = x.freq + y.freq
    INSERT(Q, z)
return EXTRACT-MIN(Q)
```



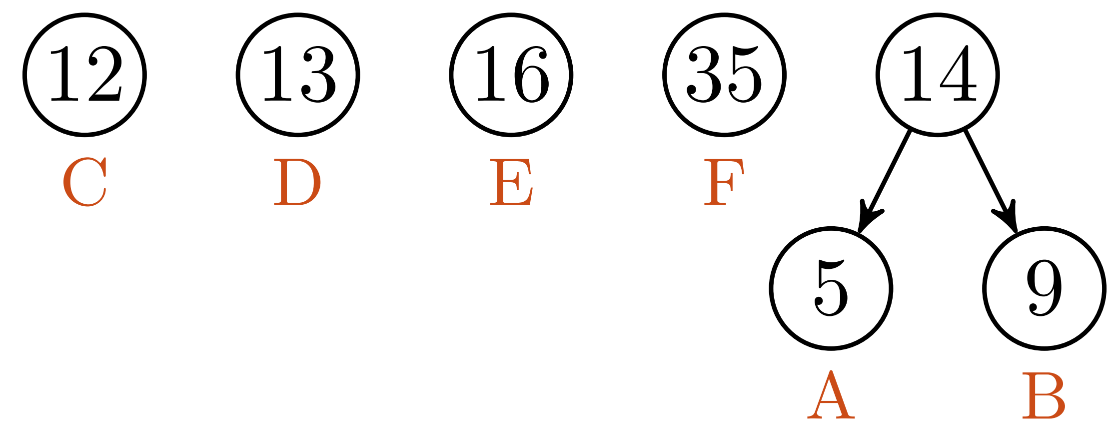
```
HUFFMAN(C)
n = |C|
Q = C
for i = 1 to n − 1
    allocate a new node z
    x = EXTRACT-MIN(Q)
    y = EXTRACT-MIN(Q)
    z.left, z.right = x, y
    z.freq = x.freq + y.freq
    INSERT(Q, z)
return EXTRACT-MIN(Q)
```



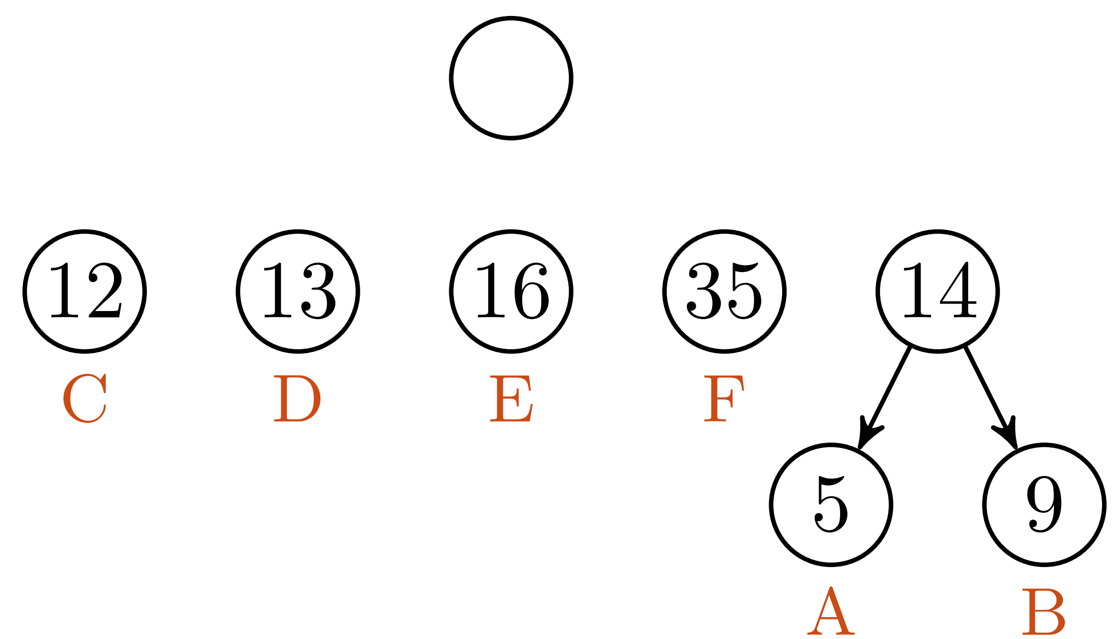
```
HUFFMAN(C)
n = |C|
Q = C
for i = 1 to n − 1
    allocate a new node z
    x = EXTRACT-MIN(Q)
    y = EXTRACT-MIN(Q)
    z.left, z.right = x, y
    z.freq = x.freq + y.freq
    INSERT(Q, z)
return EXTRACT-MIN(Q)
```



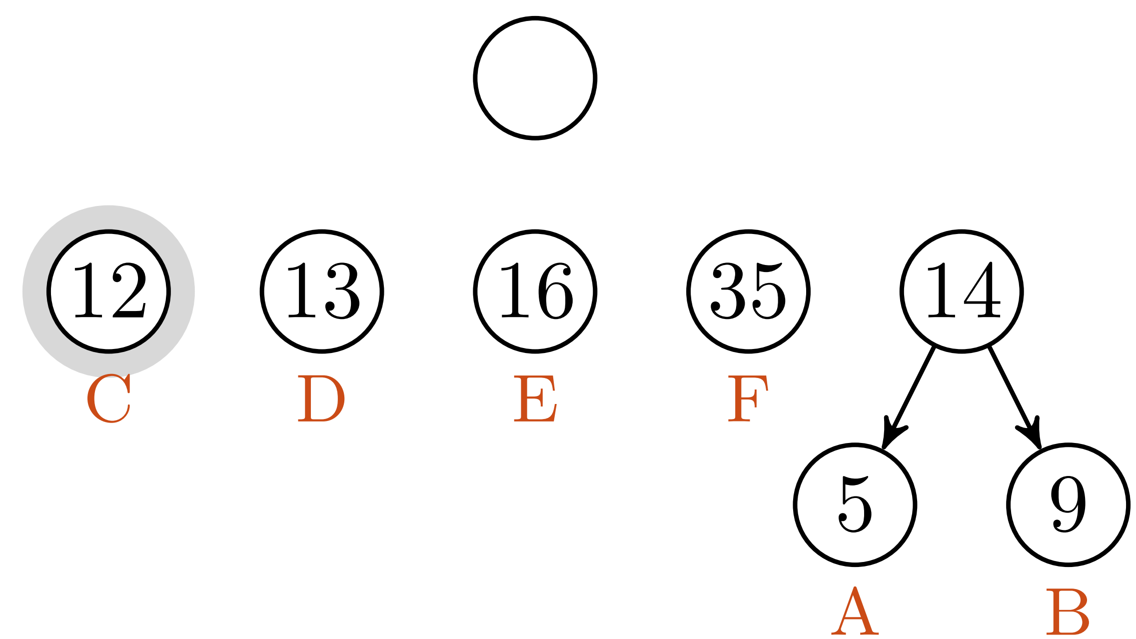
```
HUFFMAN(C)
n = |C|
Q = C
for i = 1 to n − 1
    allocate a new node z
    x = EXTRACT-MIN(Q)
    y = EXTRACT-MIN(Q)
    z.left, z.right = x, y
    z.freq = x.freq + y.freq
    INSERT(Q, z)
return EXTRACT-MIN(Q)
```



```
HUFFMAN(C)
n = |C|
Q = C
for i = 1 to n − 1
    allocate a new node z
    x = EXTRACT-MIN(Q)
    y = EXTRACT-MIN(Q)
    z.left, z.right = x, y
    z.freq = x.freq + y.freq
    INSERT(Q, z)
return EXTRACT-MIN(Q)
```

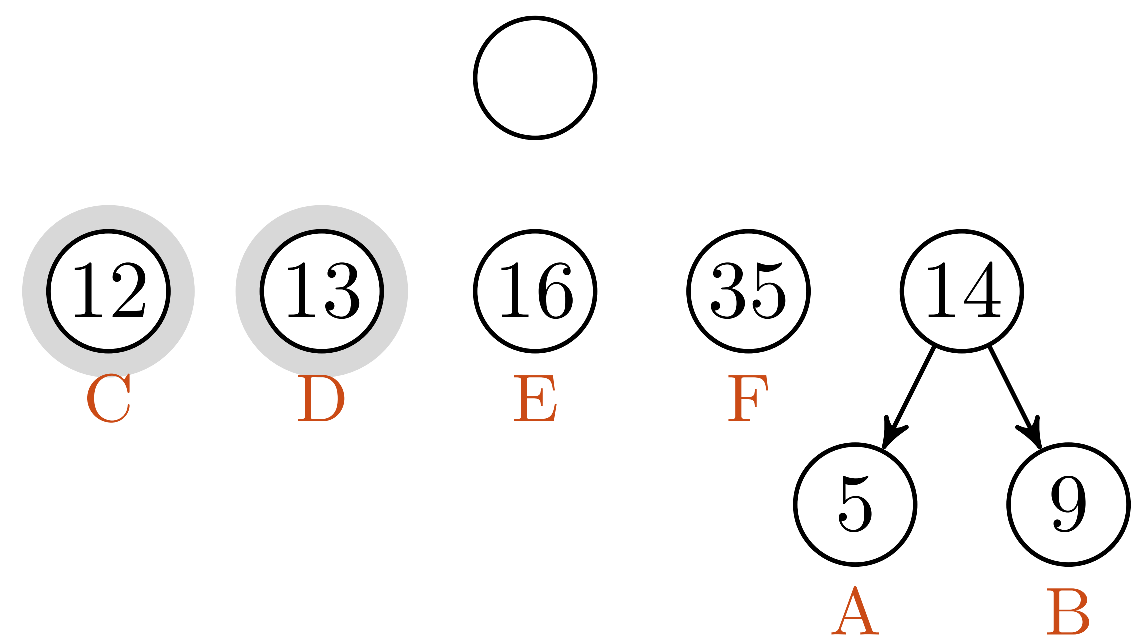


```
HUFFMAN(C)
n = |C|
Q = C
for i = 1 to n − 1
    allocate a new node z
    x = EXTRACT-MIN(Q)
    y = EXTRACT-MIN(Q)
    z.left, z.right = x, y
    z.freq = x.freq + y.freq
    INSERT(Q, z)
return EXTRACT-MIN(Q)
```

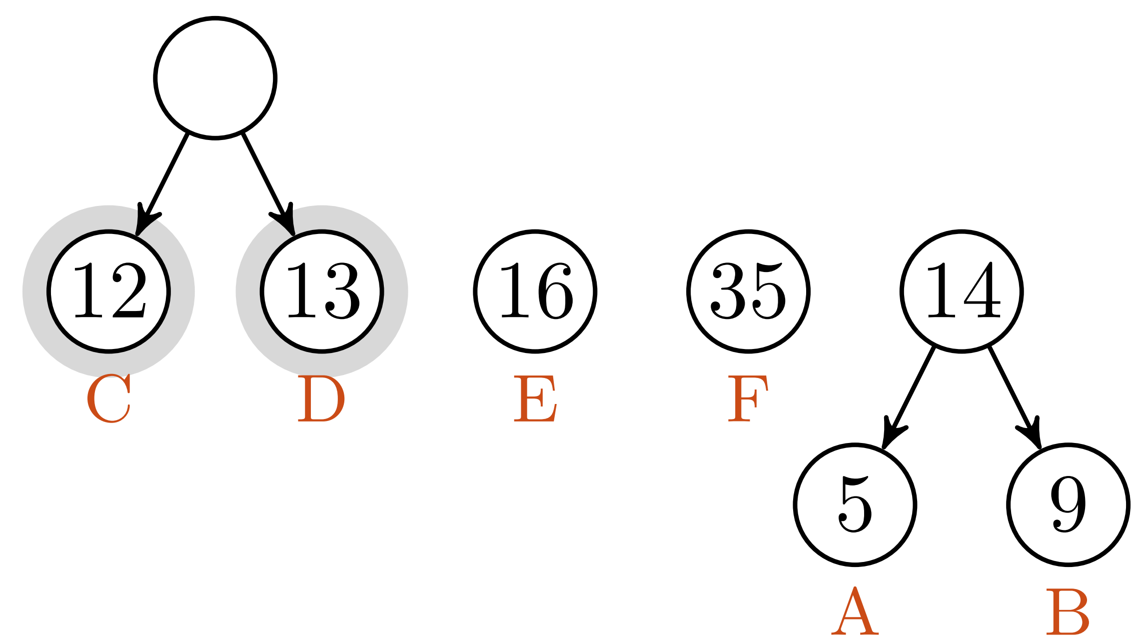


HUFFMAN(C)

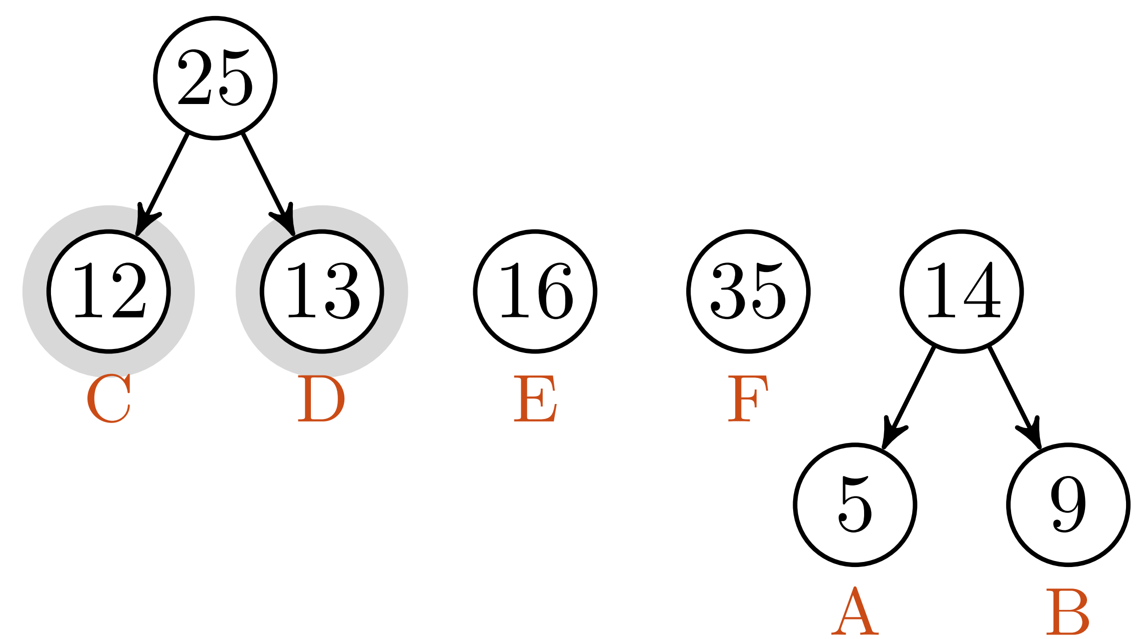
```
n = |C|
Q = C
for i = 1 to n − 1
    allocate a new node z
    x = EXTRACT-MIN(Q)
    y = EXTRACT-MIN(Q)
    z.left, z.right = x, y
    z.freq = x.freq + y.freq
    INSERT(Q, z)
return EXTRACT-MIN(Q)
```



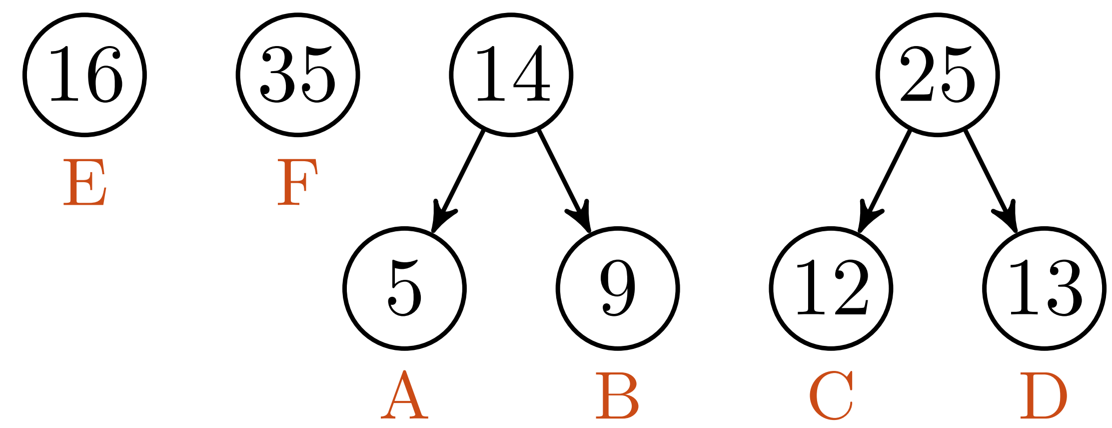
```
HUFFMAN(C)
  n = |C|
  Q = C
  for i = 1 to n − 1
      allocate a new node z
      x = EXTRACT-MIN(Q)
      y = EXTRACT-MIN(Q)
      z.left, z.right = x, y
      z.freq = x.freq + y.freq
      INSERT(Q, z)
  return EXTRACT-MIN(Q)
```



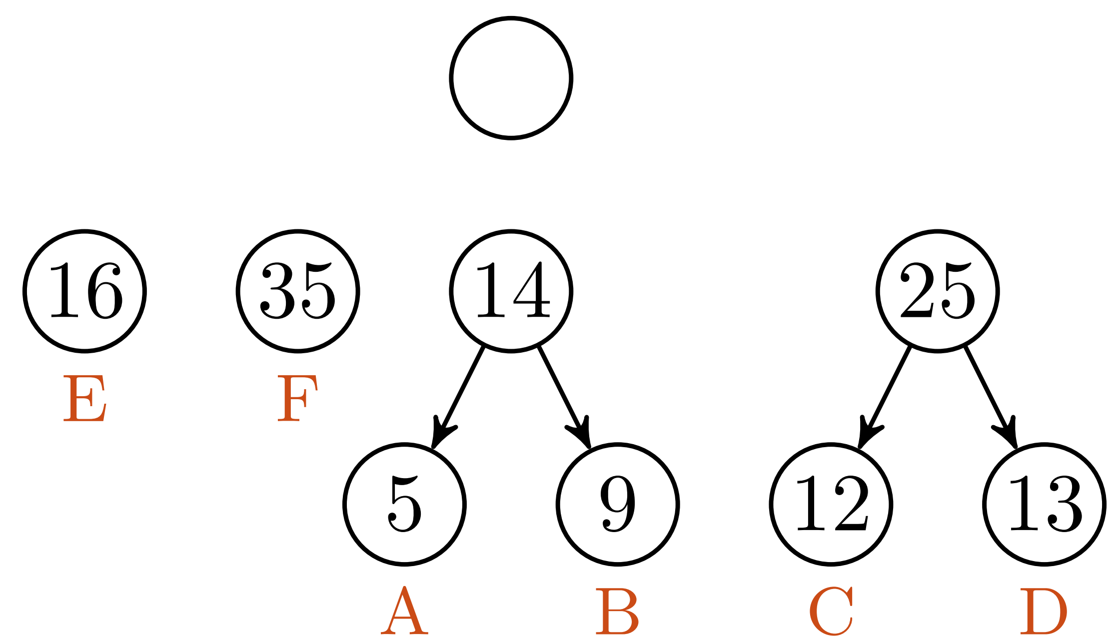
```
Huffman(C)
n = |C|
Q = C
for i = 1 to n − 1
        allocate a new node z
        x = Extract-Min(Q)
        y = Extract-Min(Q)
        z.left, z.right = x, y
        z.freq = x.freq + y.freq
        Insert(Q, z)
return Extract-Min(Q)
```



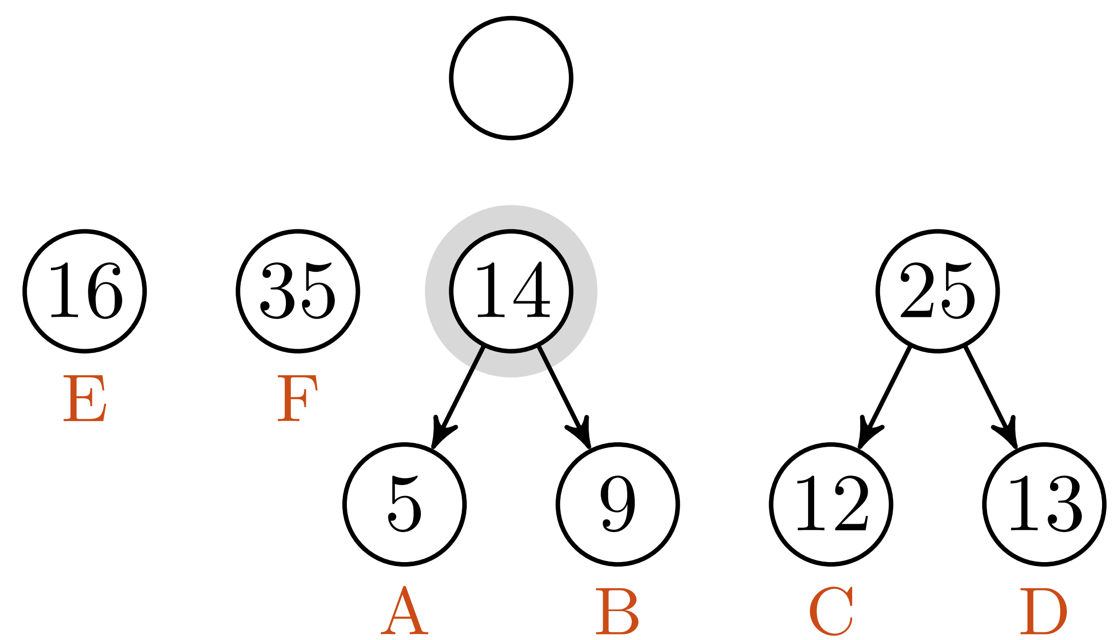
```
HUFFMAN(C)
n = |C|
Q = C
for i = 1 to n − 1
    allocate a new node z
    x = EXTRACT-MIN(Q)
    y = EXTRACT-MIN(Q)
    z.left, z.right = x, y
    z.freq = x.freq + y.freq
    INSERT(Q, z)
return EXTRACT-MIN(Q)
```



```
HUFFMAN(C)
n = |C|
Q = C
for i = 1 to n − 1
    allocate a new node z
    x = EXTRACT-MIN(Q)
    y = EXTRACT-MIN(Q)
    z.left, z.right = x, y
    z.freq = x.freq + y.freq
    INSERT(Q, z)
return EXTRACT-MIN(Q)
```

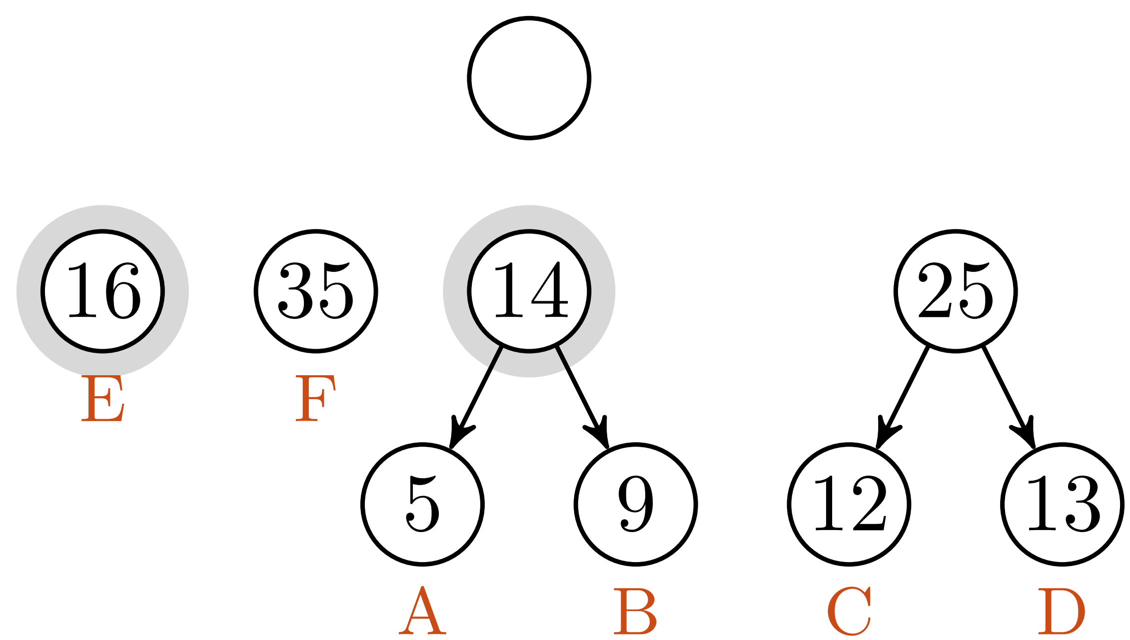


```
Huffman(C)
 1  n = |C|
 2  Q = C
 3  for i = 1 to n − 1
 4      allocate a new node z
 5      x = Extract-Min(Q)
 6      y = Extract-Min(Q)
 7      z.left, z.right = x, y
 8      z.freq = x.freq + y.freq
 9      Insert(Q, z)
10  return Extract-Min(Q)
```

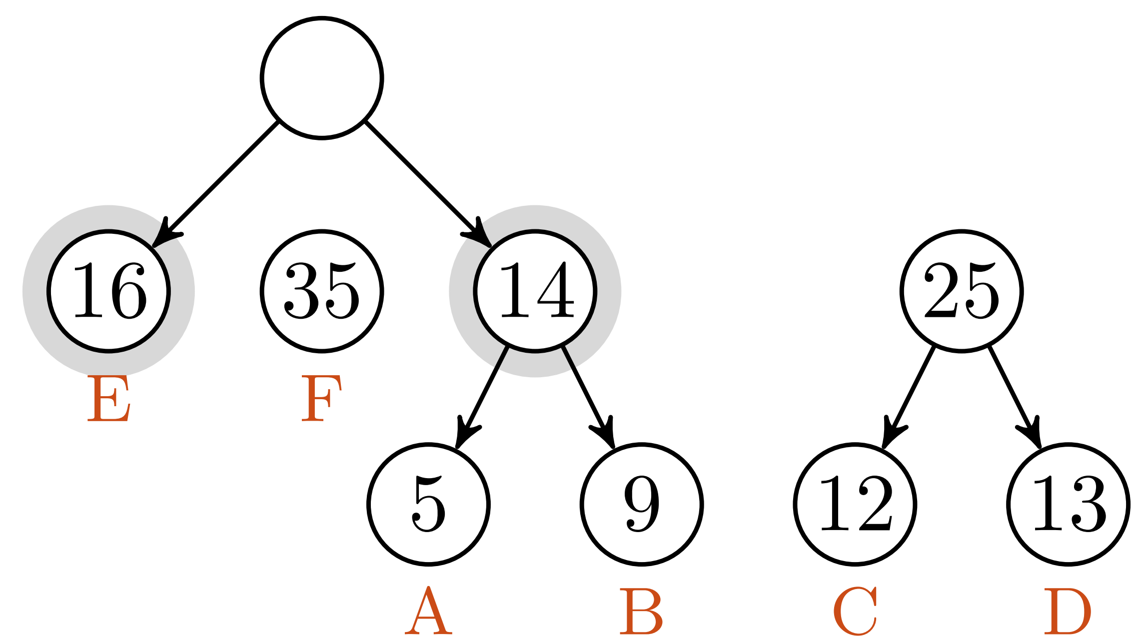


HUFFMAN(C)

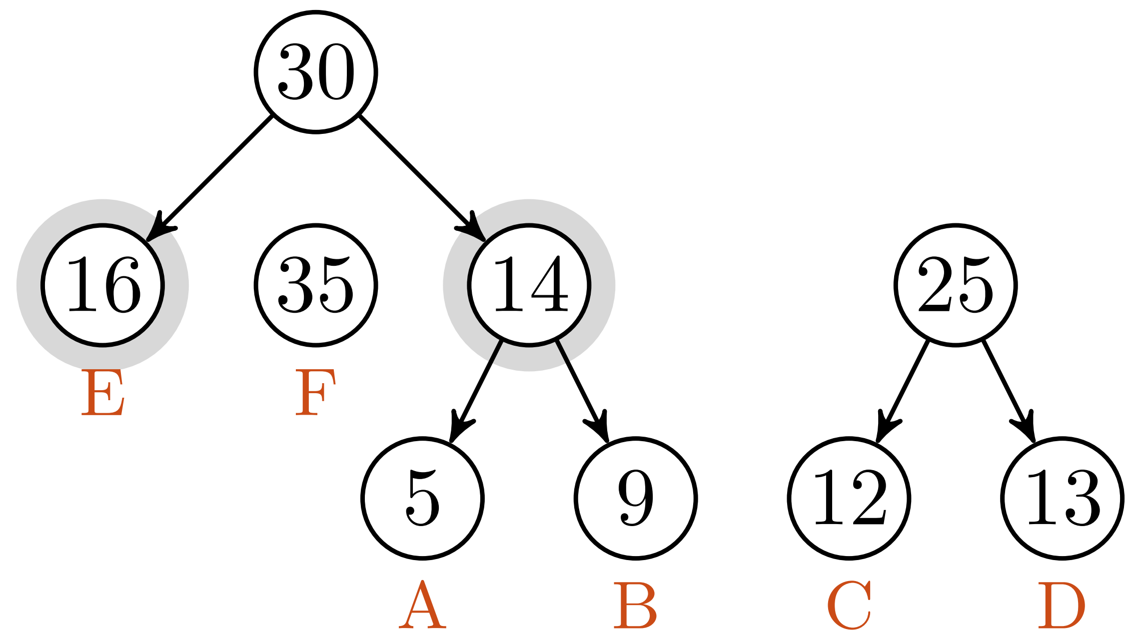
```
n = |C|
Q = C
for i = 1 to n − 1
    allocate a new node z
    x = EXTRACT-MIN(Q)
    y = EXTRACT-MIN(Q)
    z.left, z.right = x, y
    z.freq = x.freq + y.freq
    INSERT(Q, z)
return EXTRACT-MIN(Q)
```



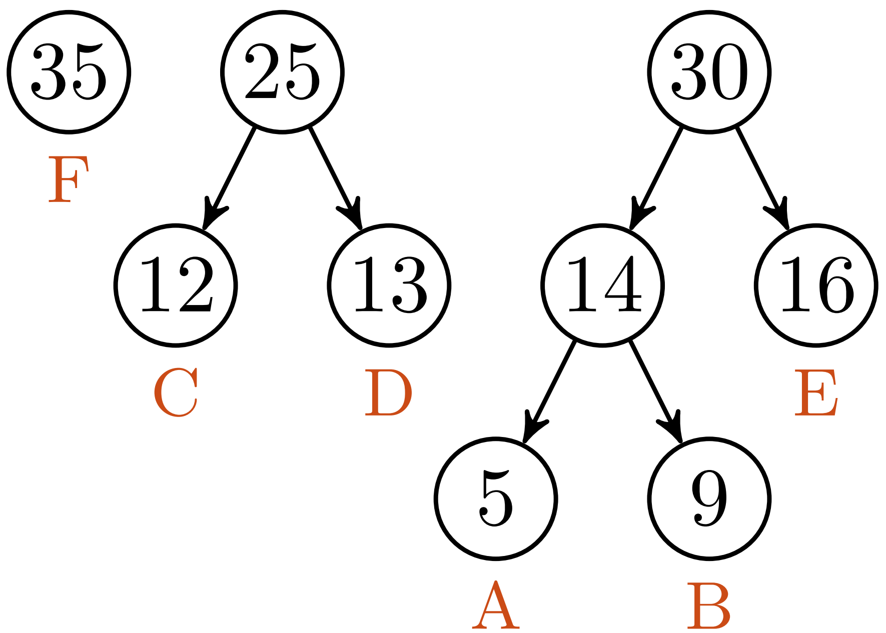
```
HUFFMAN(C)
n = |C|
Q = C
for i = 1 to n − 1
    allocate a new node z
    x = EXTRACT-MIN(Q)
    y = EXTRACT-MIN(Q)
    z.left, z.right = x, y
    z.freq = x.freq + y.freq
    INSERT(Q, z)
return EXTRACT-MIN(Q)
```



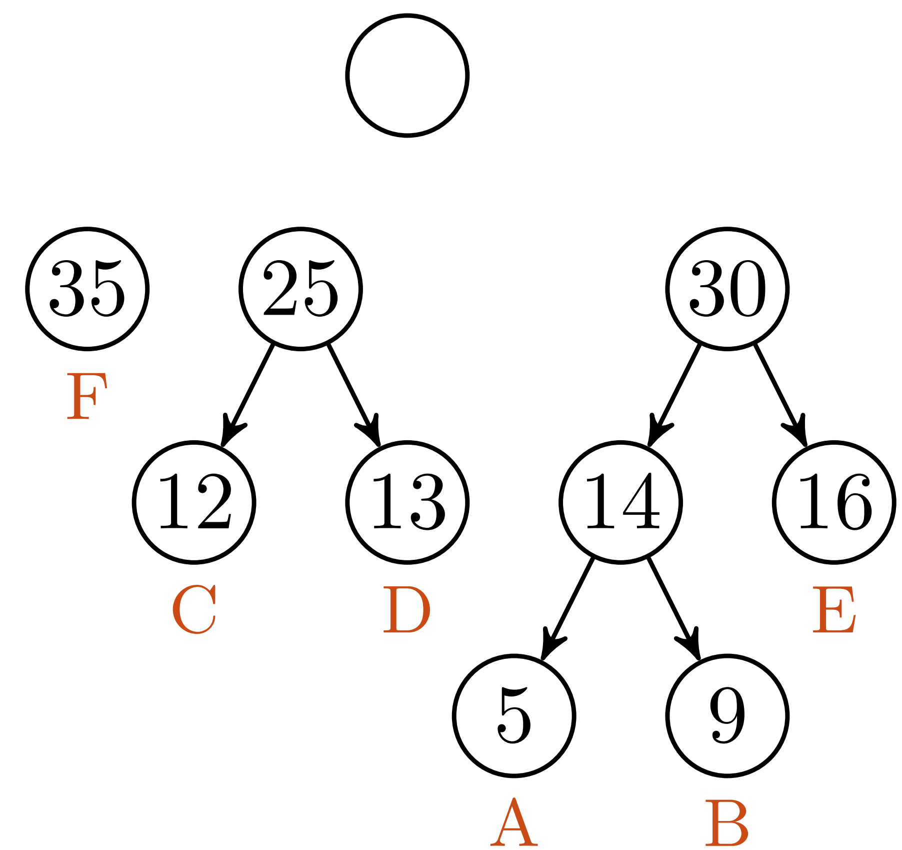
```
HUFFMAN(C)
n = |C|
Q = C
for i = 1 to n − 1
    allocate a new node z
    x = EXTRACT-MIN(Q)
    y = EXTRACT-MIN(Q)
    z.left, z.right = x, y
    z.freq = x.freq + y.freq
    INSERT(Q, z)
return EXTRACT-MIN(Q)
```



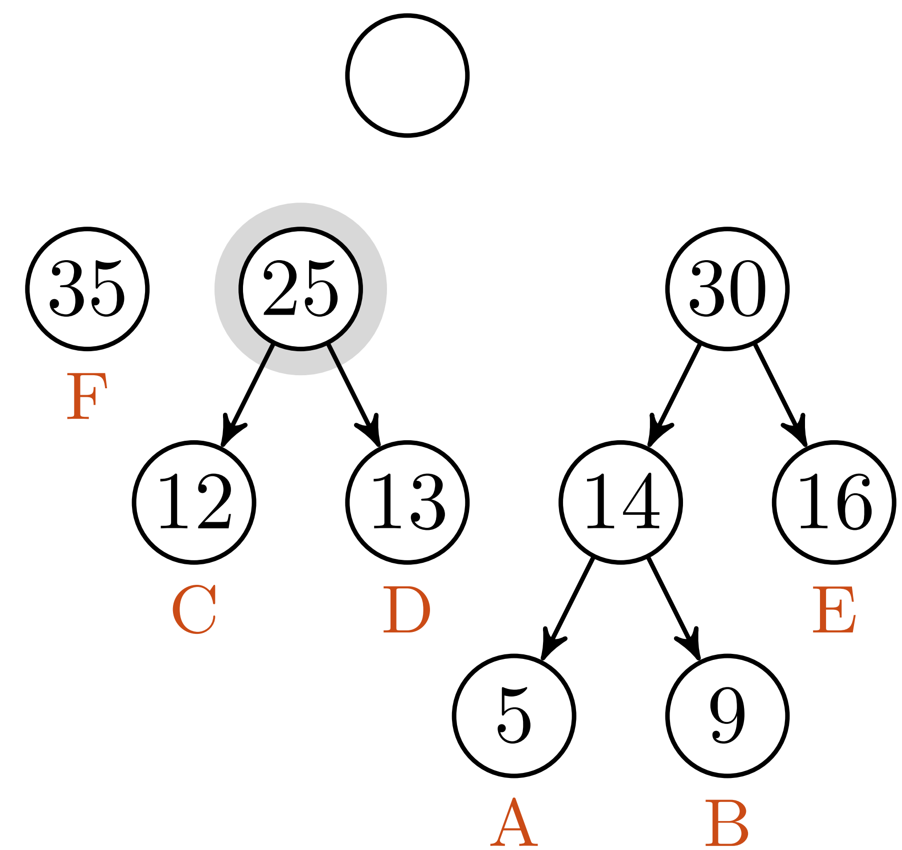
```
HUFFMAN(C)
n = |C|
Q = C
for i = 1 to n − 1
    allocate a new node z
    x = EXTRACT-MIN(Q)
    y = EXTRACT-MIN(Q)
    z.left, z.right = x, y
    z.freq = x.freq + y.freq
    INSERT(Q, z)
return EXTRACT-MIN(Q)
```



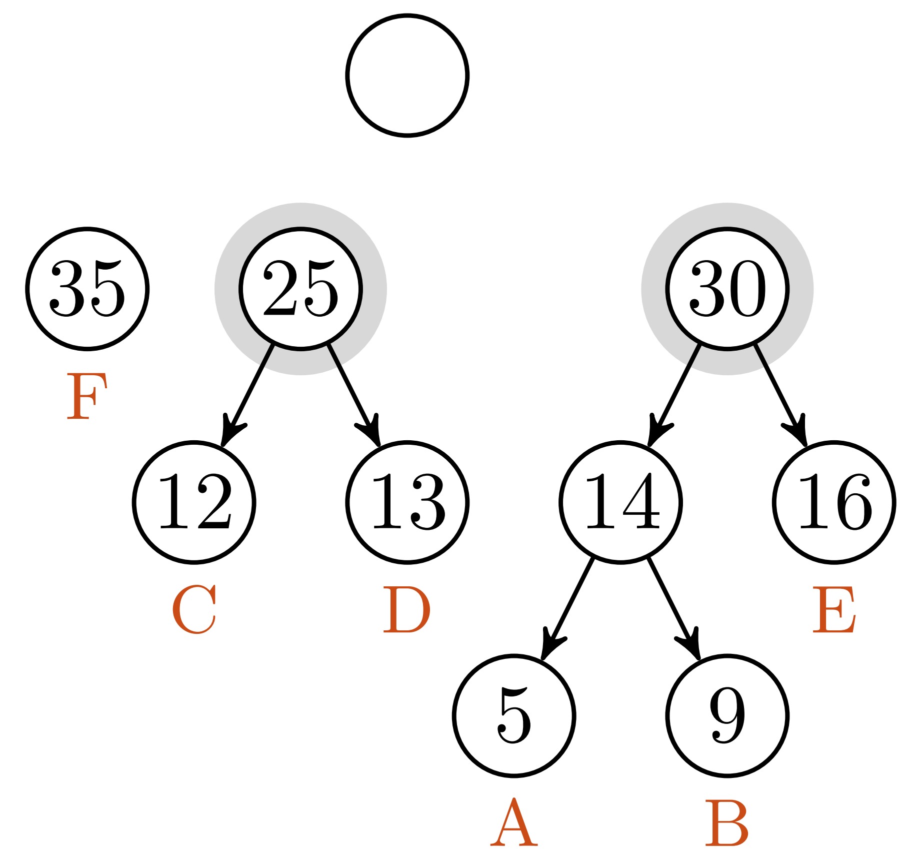
```
HUFFMAN(C)
n = |C|
Q = C
for i = 1 to n − 1
    allocate a new node z
    x = EXTRACT-MIN(Q)
    y = EXTRACT-MIN(Q)
    z.left, z.right = x, y
    z.freq = x.freq + y.freq
    INSERT(Q, z)
return EXTRACT-MIN(Q)
```



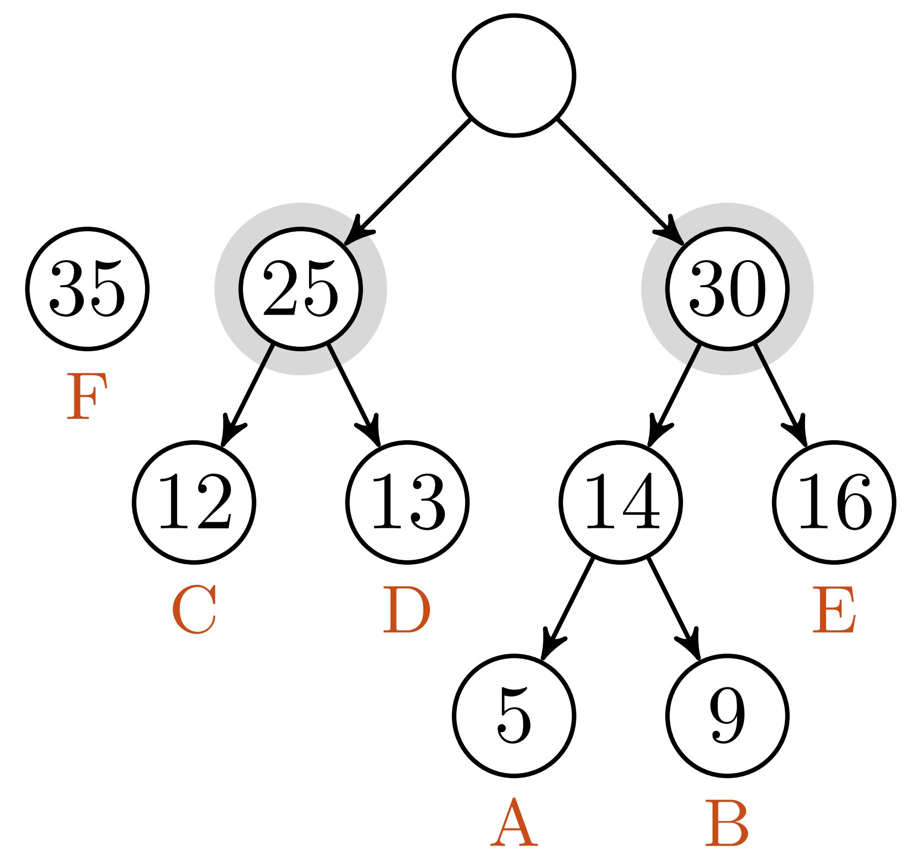
```
HUFFMAN(C)
n = |C|
Q = C
for i = 1 to n − 1
    allocate a new node z
    x = EXTRACT-MIN(Q)
    y = EXTRACT-MIN(Q)
    z.left, z.right = x, y
    z.freq = x.freq + y.freq
    INSERT(Q, z)
return EXTRACT-MIN(Q)
```



```
HUFFMAN(C)
n = |C|
Q = C
for i = 1 to n − 1
        allocate a new node z
        x = EXTRACT-MIN(Q)
        y = EXTRACT-MIN(Q)
        z.left, z.right = x, y
        z.freq = x.freq + y.freq
        INSERT(Q, z)
return EXTRACT-MIN(Q)
```

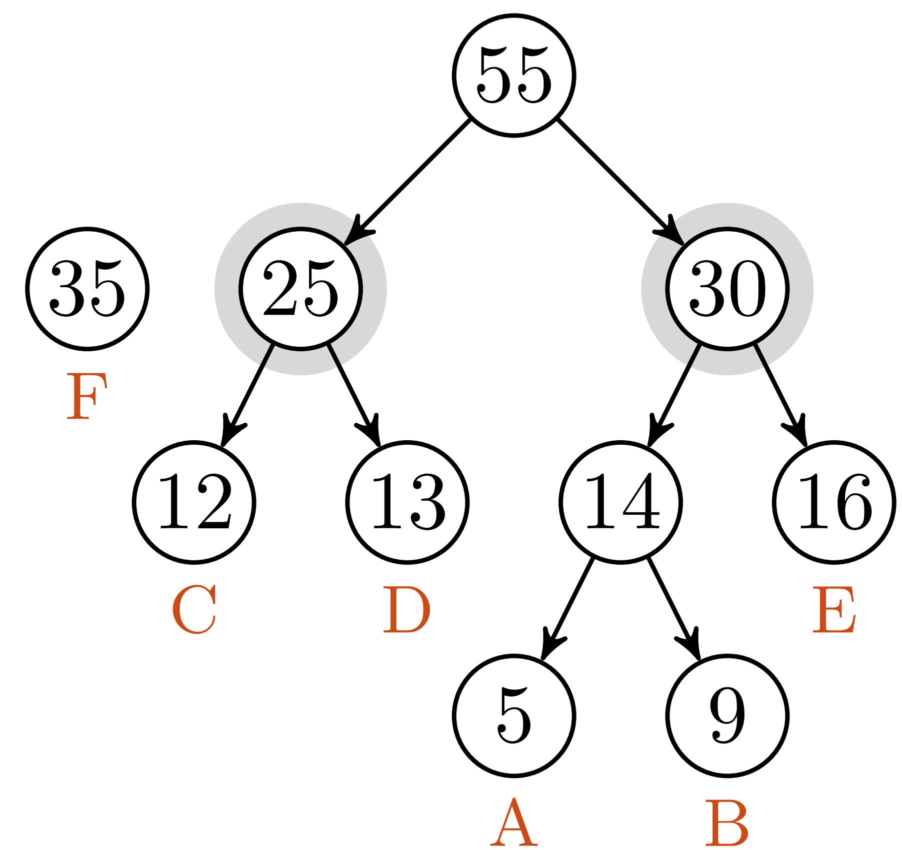


```
HUFFMAN(C)
n = |C|
Q = C
for i = 1 to n − 1
    allocate a new node z
    x = EXTRACT-MIN(Q)
    y = EXTRACT-MIN(Q)
    z.left, z.right = x, y
    z.freq = x.freq + y.freq
    INSERT(Q, z)
return EXTRACT-MIN(Q)
```



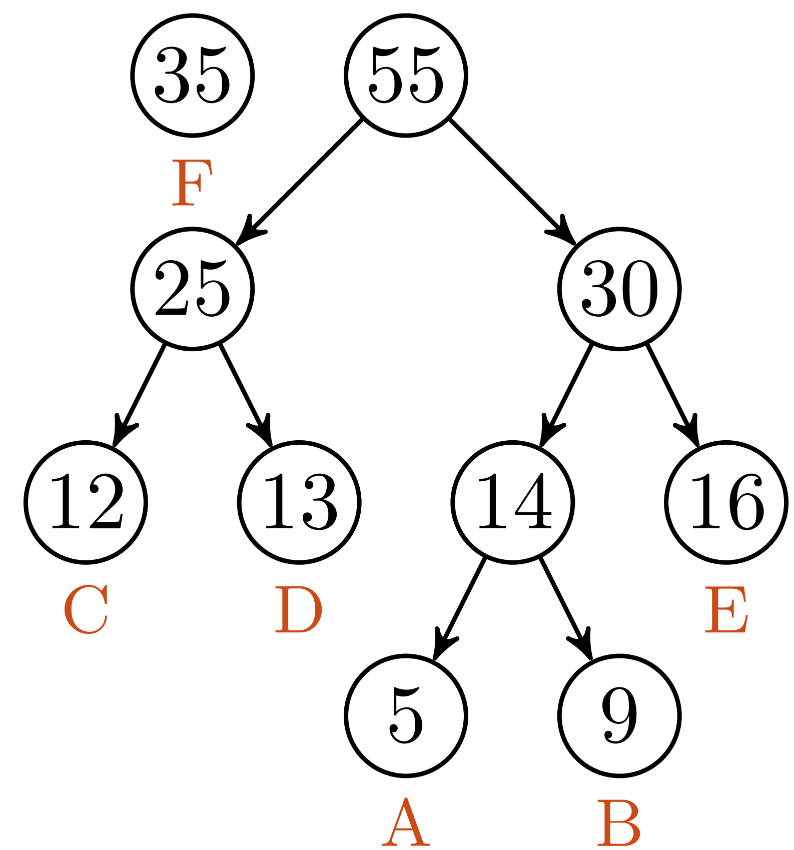
HUFFMAN(C)

$n = |C|$
$Q = C$
for $i = 1$ **to** $n - 1$
    allocate a new node z
    $x = \text{EXTRACT-MIN}(Q)$
    $y = \text{EXTRACT-MIN}(Q)$
    $z.\mathit{left}, z.\mathit{right} = x, y$
    $z.\mathit{freq} = x.\mathit{freq} + y.\mathit{freq}$
    $\text{INSERT}(Q, z)$
return $\text{EXTRACT-MIN}(Q)$

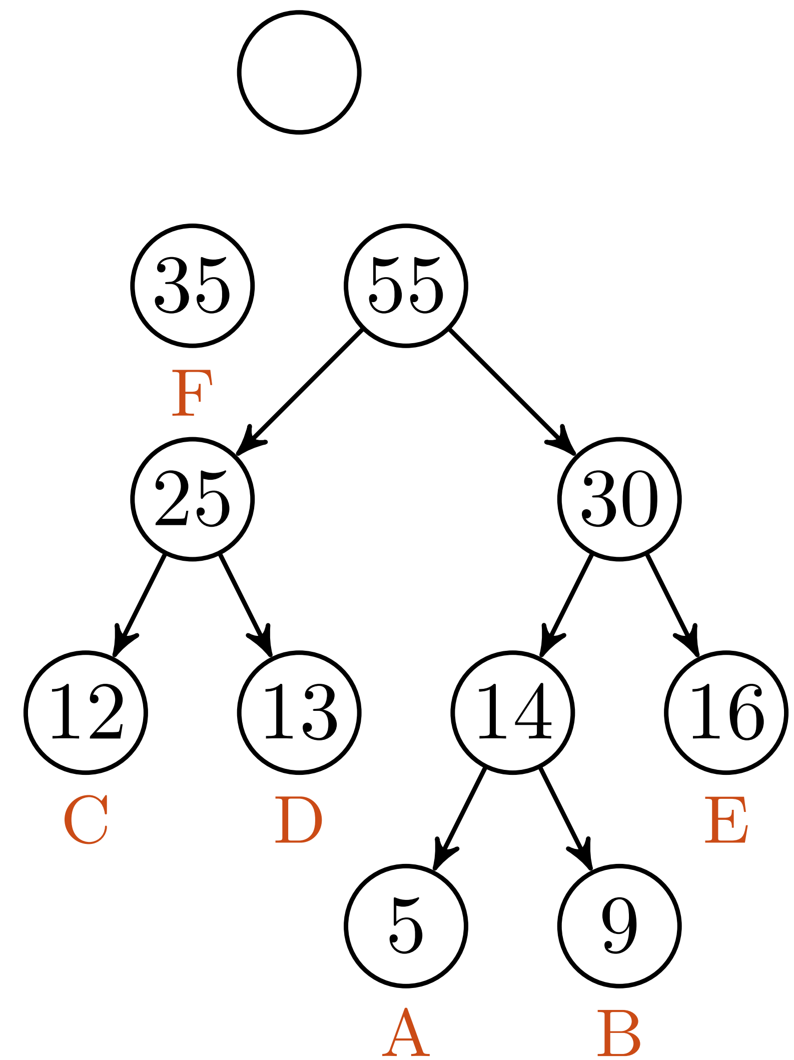


HUFFMAN(C)

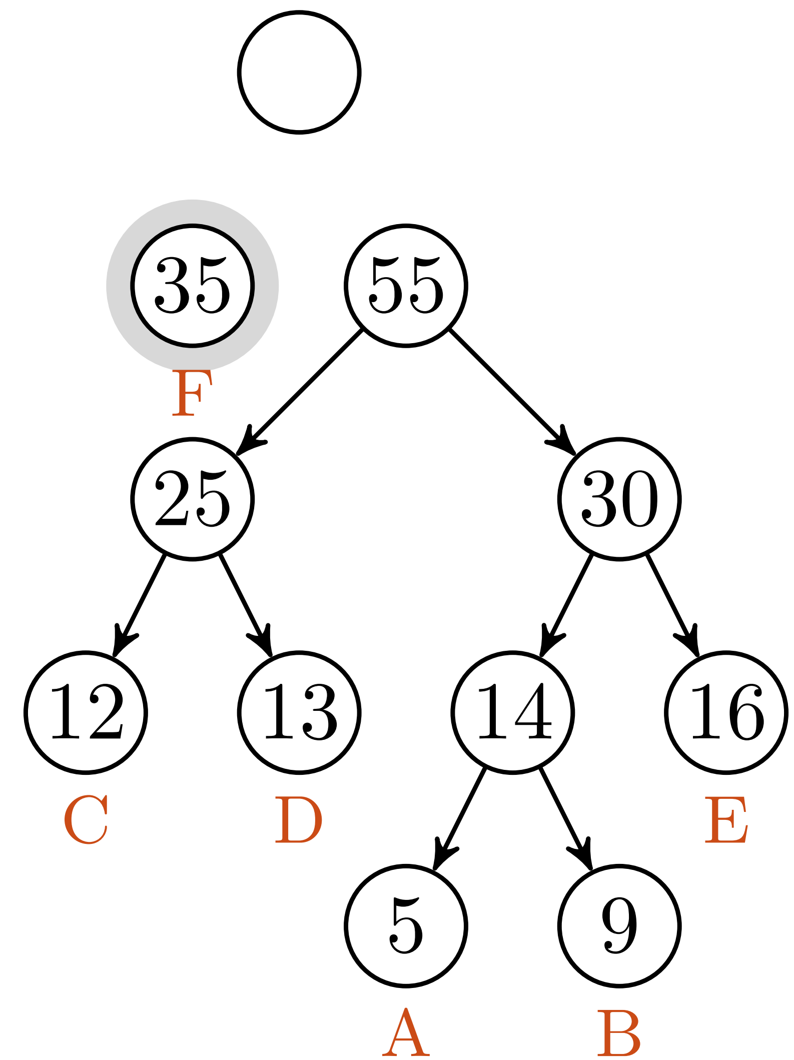
$n = |\mathrm{C}|$
$\mathrm{Q} = \mathrm{C}$
for $i = 1$ **to** $n - 1$
    allocate a new node z
    $x = \textsc{Extract-Min}(\mathrm{Q})$
    $y = \textsc{Extract-Min}(\mathrm{Q})$
    $z.\mathit{left}, z.\mathit{right} = x, y$
    $z.\mathit{freq} = x.\mathit{freq} + y.\mathit{freq}$
    $\textsc{Insert}(\mathrm{Q}, z)$
return $\textsc{Extract-Min}(\mathrm{Q})$



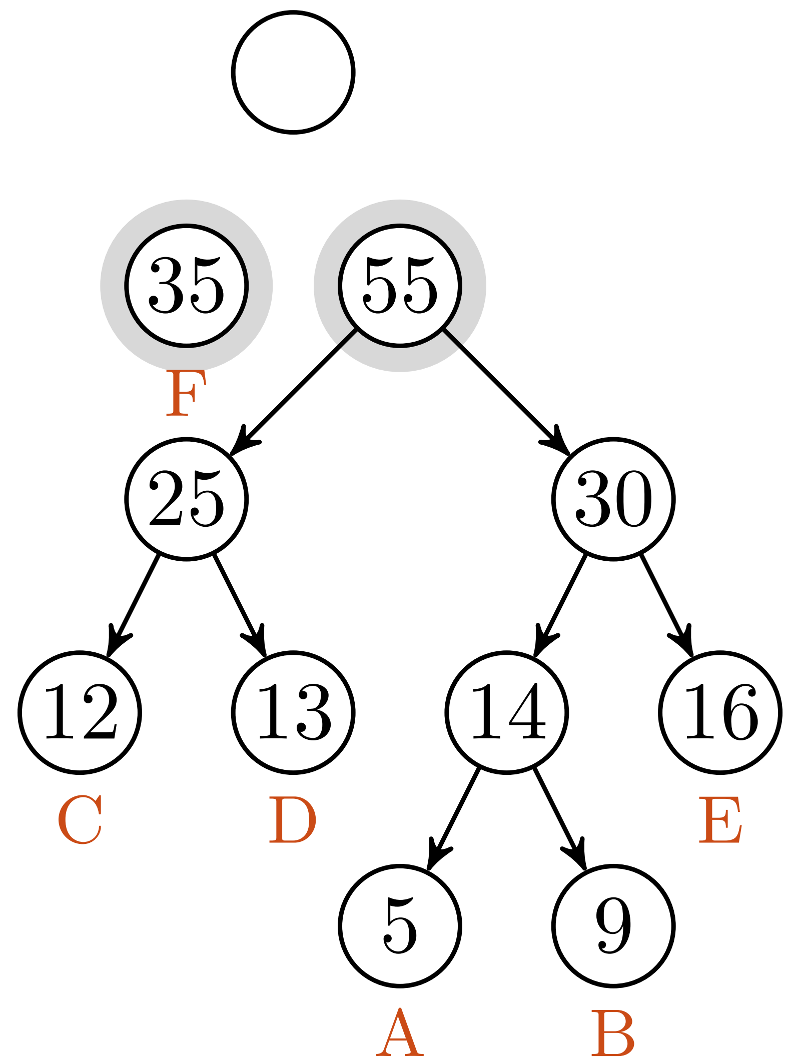
```
HUFFMAN(C)
n = |C|
Q = C
for i = 1 to n − 1
    allocate a new node z
    x = EXTRACT-MIN(Q)
    y = EXTRACT-MIN(Q)
    z.left, z.right = x, y
    z.freq = x.freq + y.freq
    INSERT(Q, z)
return EXTRACT-MIN(Q)
```



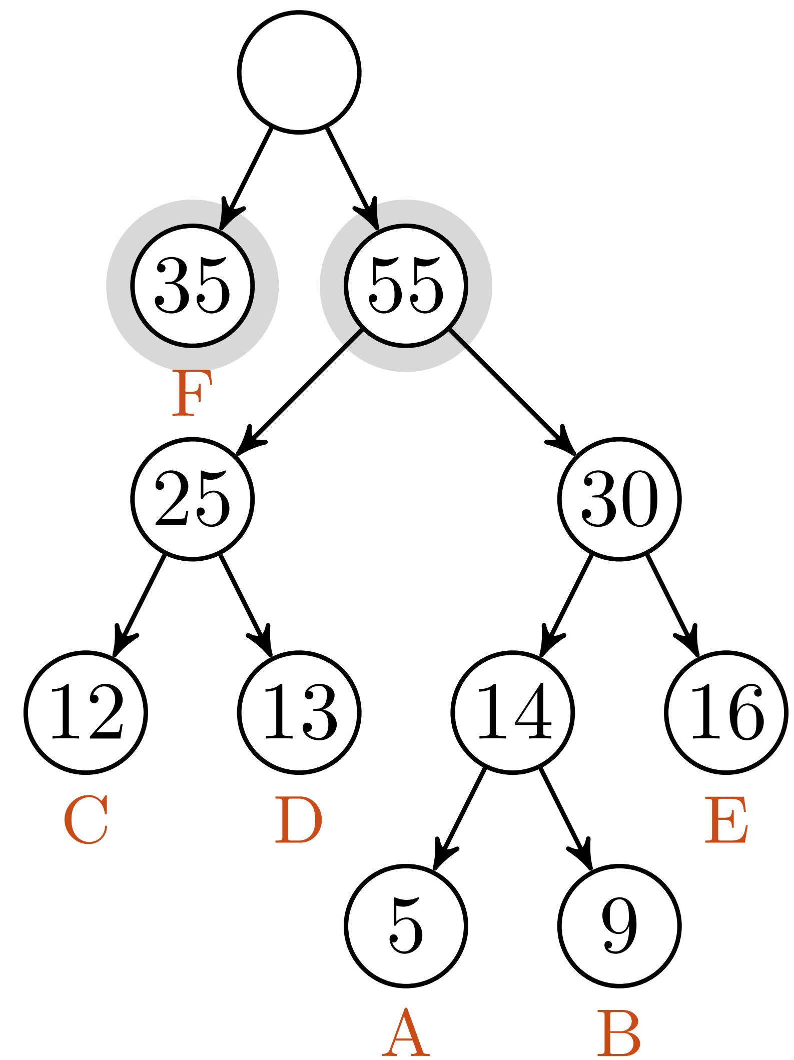
```
HUFFMAN(C)
n = |C|
Q = C
for i = 1 to n − 1
    allocate a new node z
    x = EXTRACT-MIN(Q)
    y = EXTRACT-MIN(Q)
    z.left, z.right = x, y
    z.freq = x.freq + y.freq
    INSERT(Q, z)
return EXTRACT-MIN(Q)
```



```
HUFFMAN(C)
n = |C|
Q = C
for i = 1 to n − 1
    allocate a new node z
    x = EXTRACT-MIN(Q)
    y = EXTRACT-MIN(Q)
    z.left, z.right = x, y
    z.freq = x.freq + y.freq
    INSERT(Q, z)
return EXTRACT-MIN(Q)
```

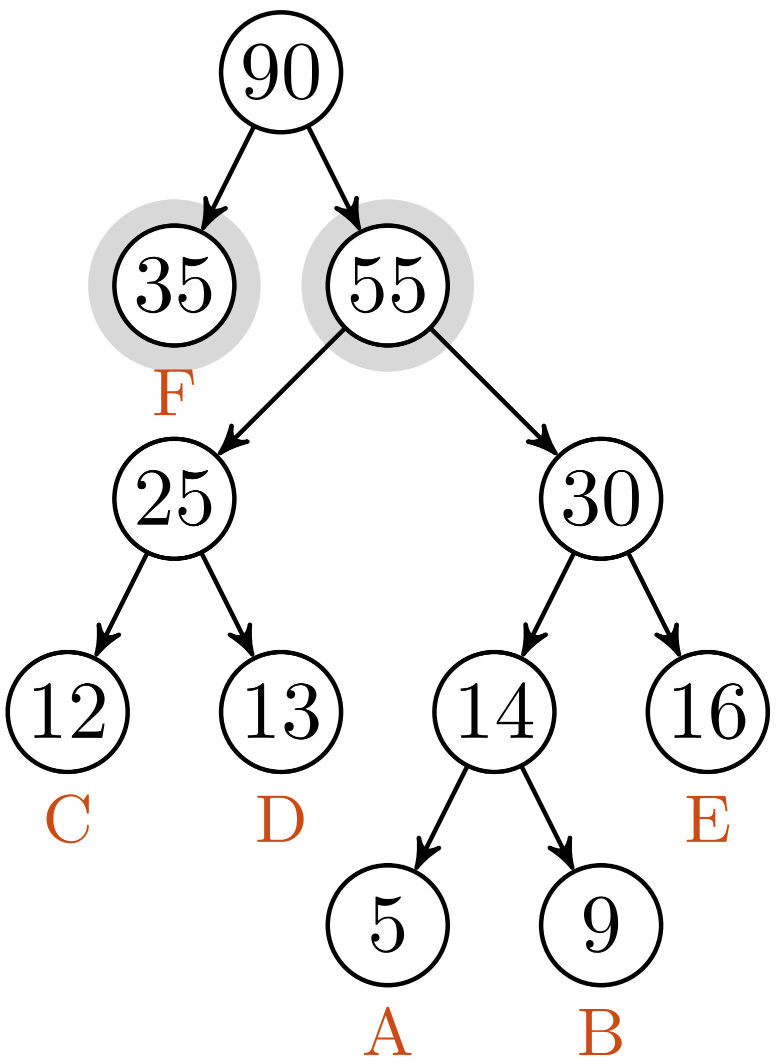


```
HUFFMAN(C)
n = |C|
Q = C
for i = 1 to n − 1
    allocate a new node z
    x = EXTRACT-MIN(Q)
    y = EXTRACT-MIN(Q)
    z.left, z.right = x, y
    z.freq = x.freq + y.freq
    INSERT(Q, z)
return EXTRACT-MIN(Q)
```

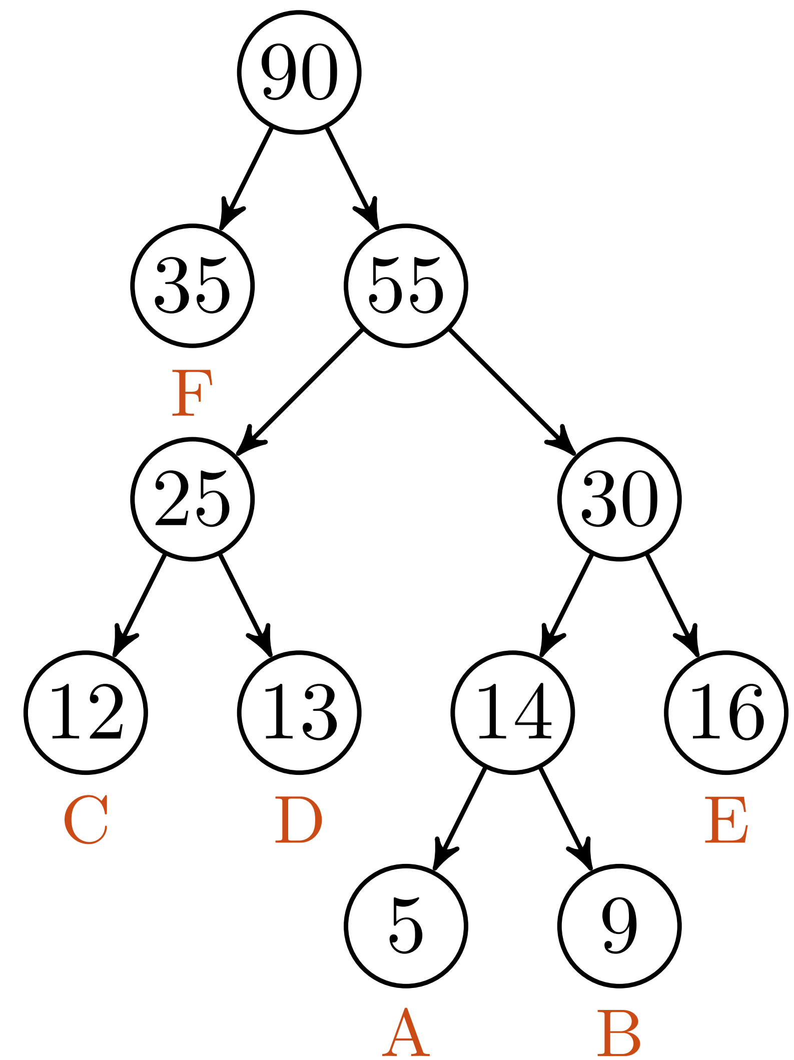


HUFFMAN(C)

```
n = |C|
Q = C
for i = 1 to n − 1
    allocate a new node z
    x = EXTRACT-MIN(Q)
    y = EXTRACT-MIN(Q)
    z.left, z.right = x, y
    z.freq = x.freq + y.freq
    INSERT(Q, z)
return EXTRACT-MIN(Q)
```

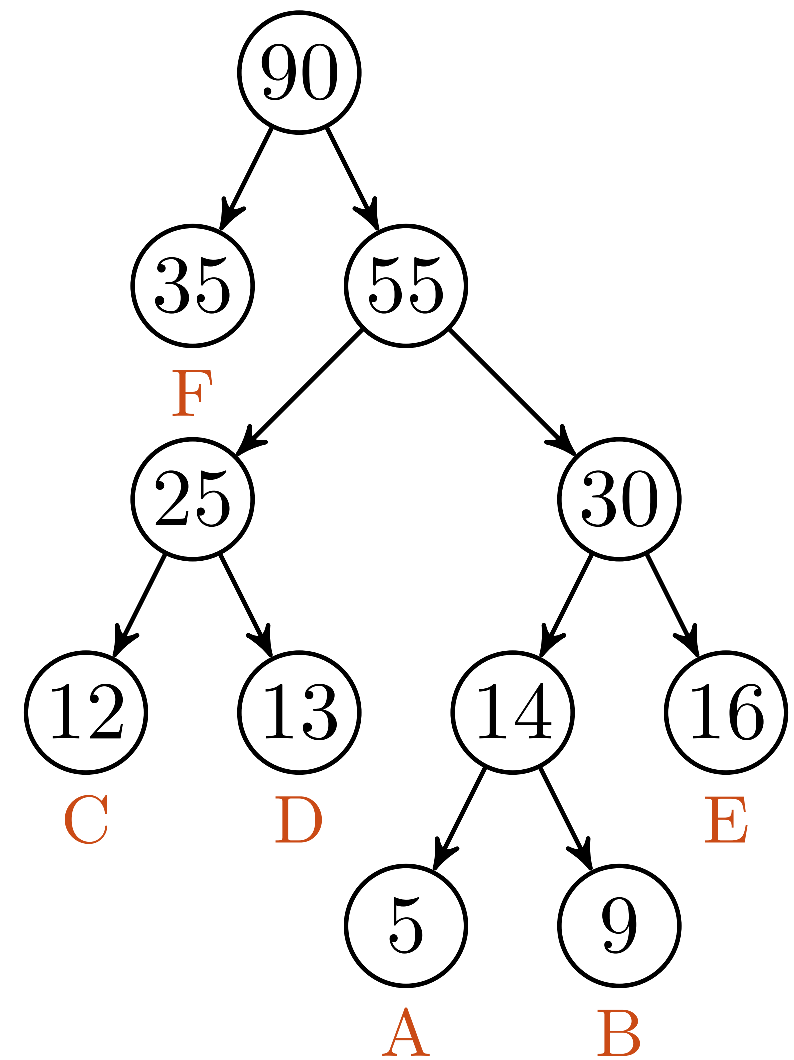


```
Huffman(C)
n = |C|
Q = C
for i = 1 to n − 1
    allocate a new node z
    x = Extract-Min(Q)
    y = Extract-Min(Q)
    z.left, z.right = x, y
    z.freq = x.freq + y.freq
    Insert(Q, z)
return Extract-Min(Q)
```



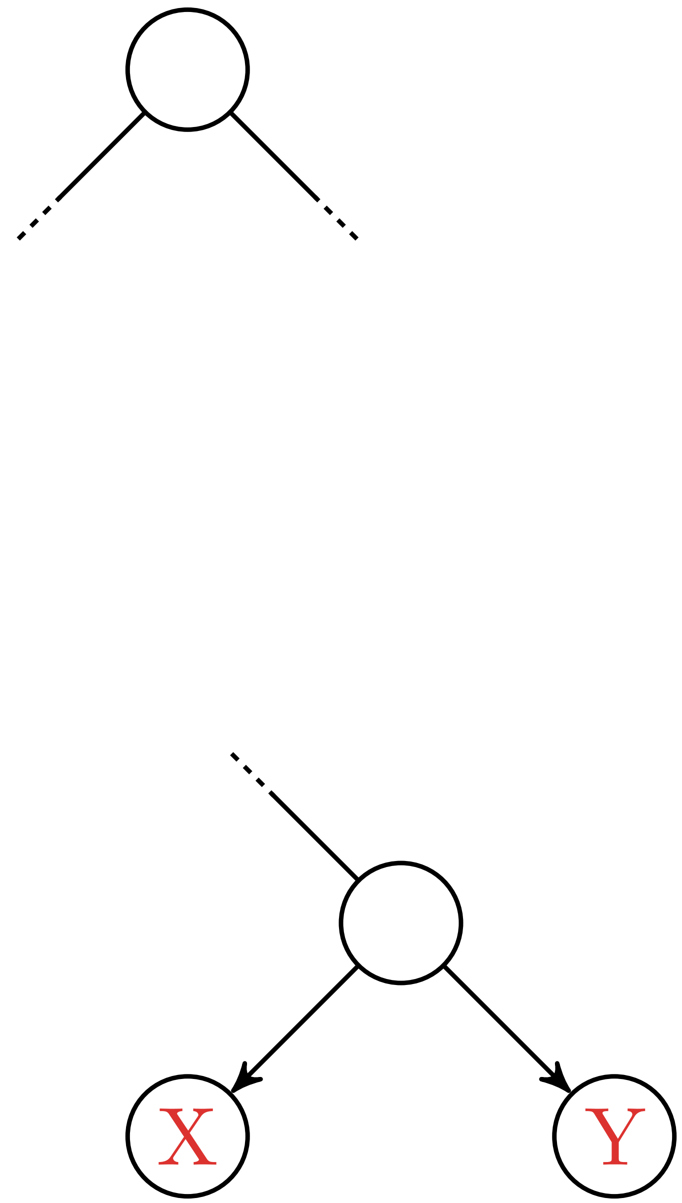
HUFFMAN(C)

$n = |C|$
$Q = C$
for $i = 1$ **to** $n - 1$
    allocate a new node z
    $x =$ EXTRACT-MIN(Q)
    $y =$ EXTRACT-MIN(Q)
    $z.left, z.right = x, y$
    $z.freq = x.freq + y.freq$
    INSERT(Q, z)
return EXTRACT-MIN(Q)

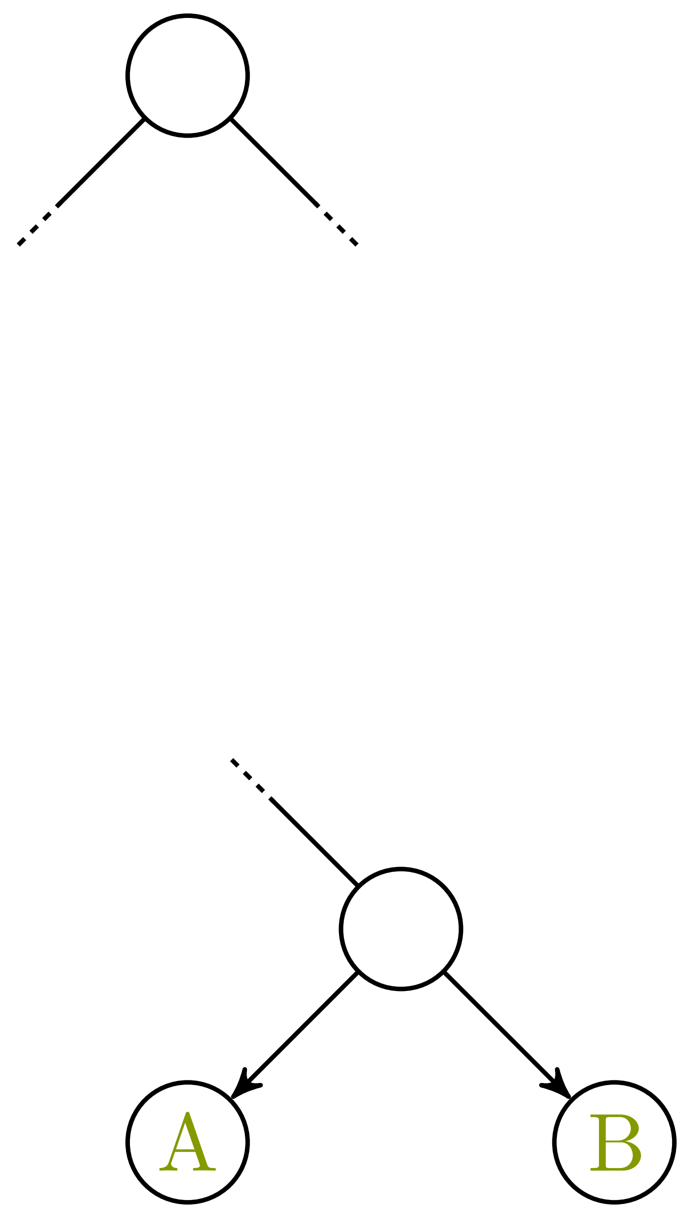


Grådighet › Huffman ›

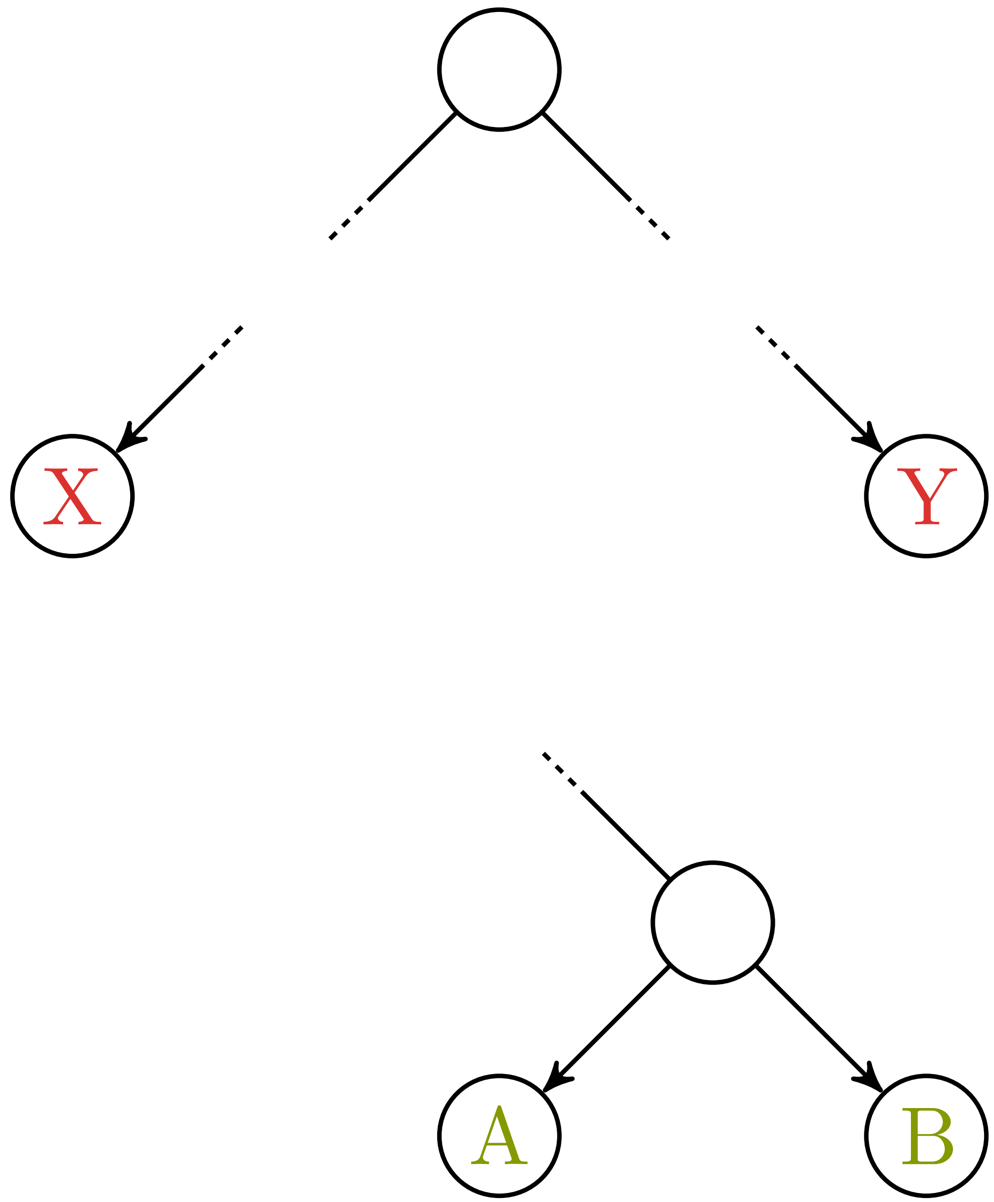
Korrekthet



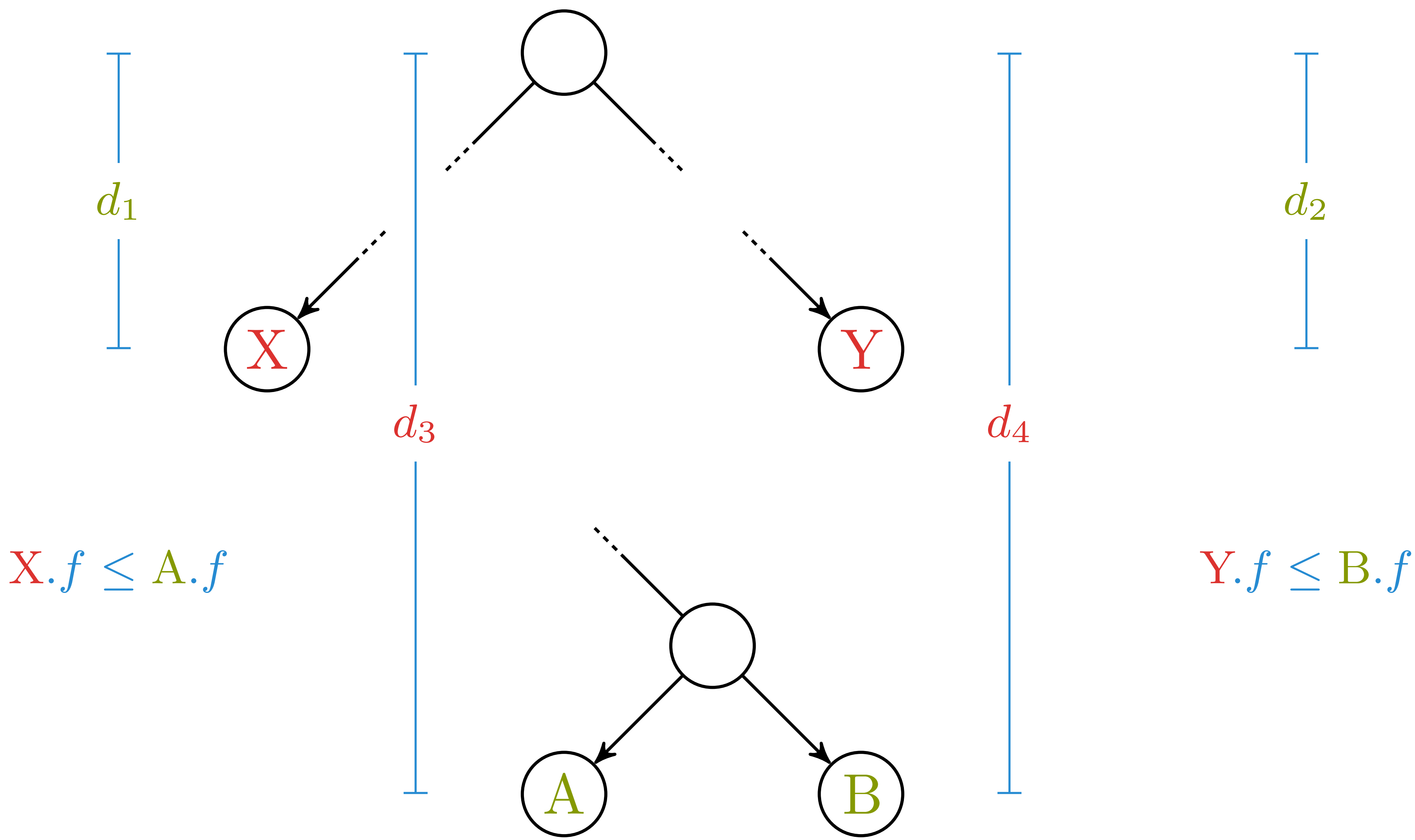
Vil vise: Sjeldneste sammen nederst lønner seg



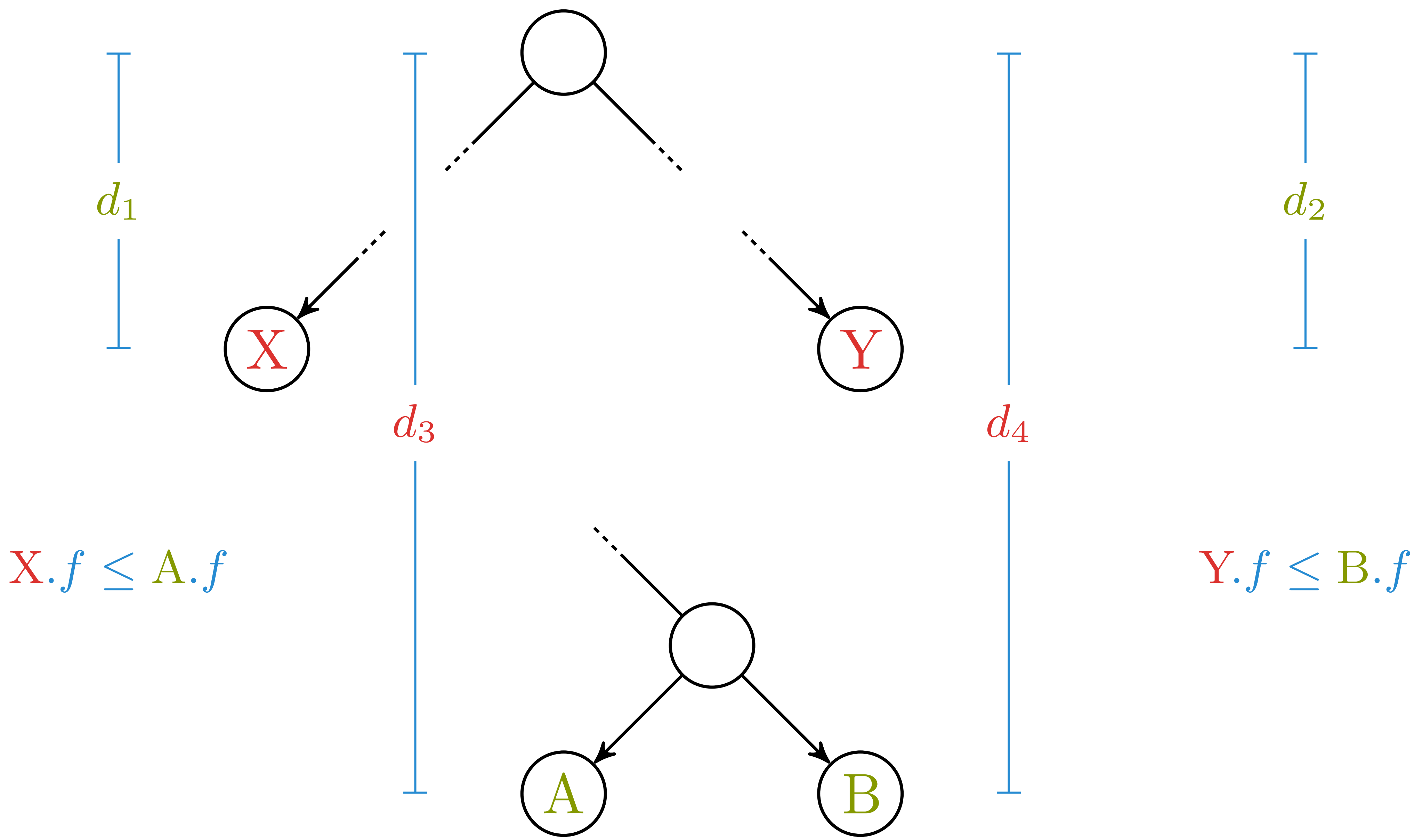
Anta en vilkårlig annen løsning . . .



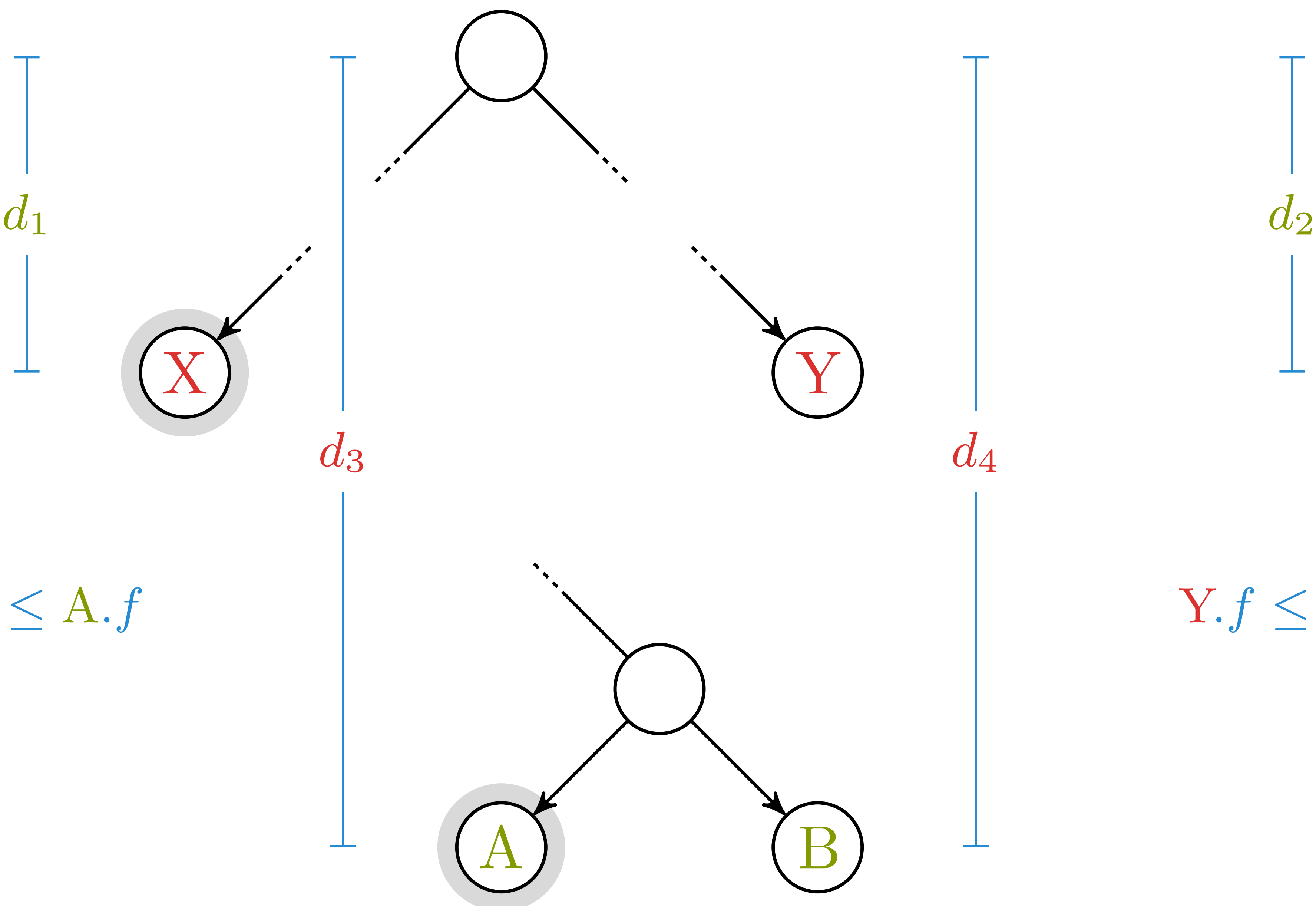
… med de sjeldneste lenger opp



… med de sjeldneste lenger opp



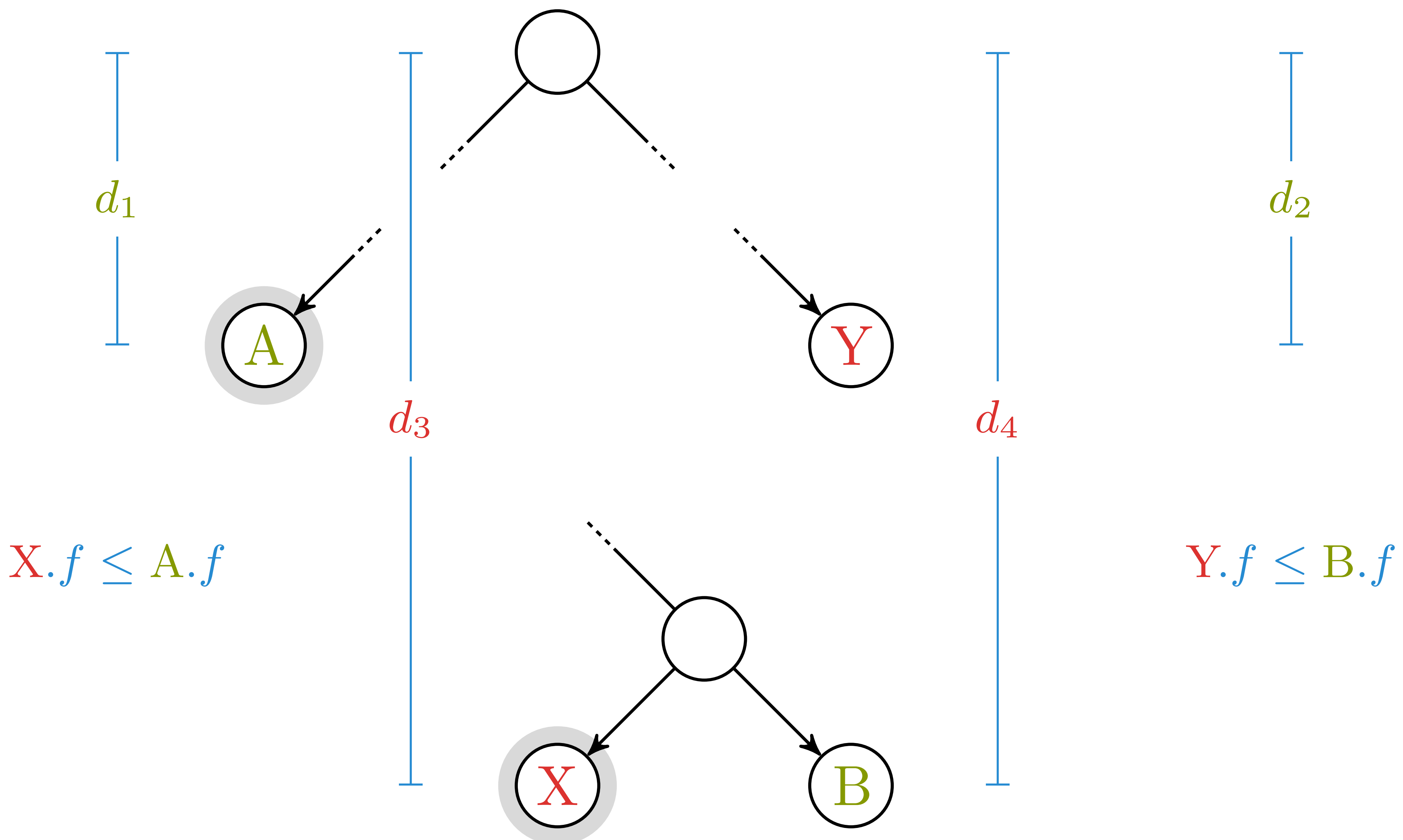
Bidrag fra $\boxed{\mathrm{X}, \mathrm{Y}, \mathrm{A}, \mathrm{B}}$ er $\boxed{d_1\mathrm{X}.f + d_2\mathrm{Y}.f + d_3\mathrm{A}.f + d_4\mathrm{B}.f}$



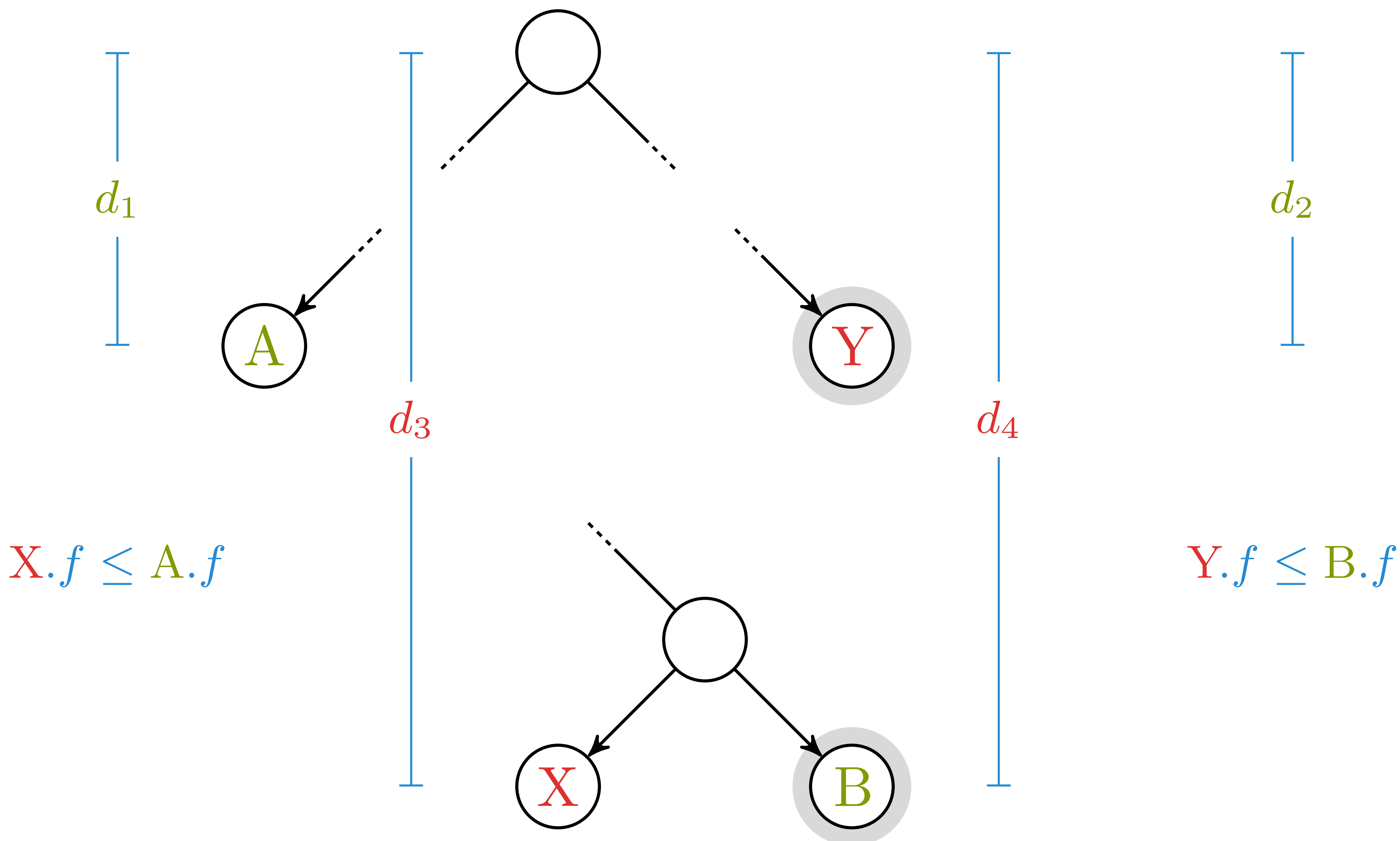
$\mathrm{X}.f \leq \mathrm{A}.f$

$\mathrm{Y}.f \leq \mathrm{B}.f$

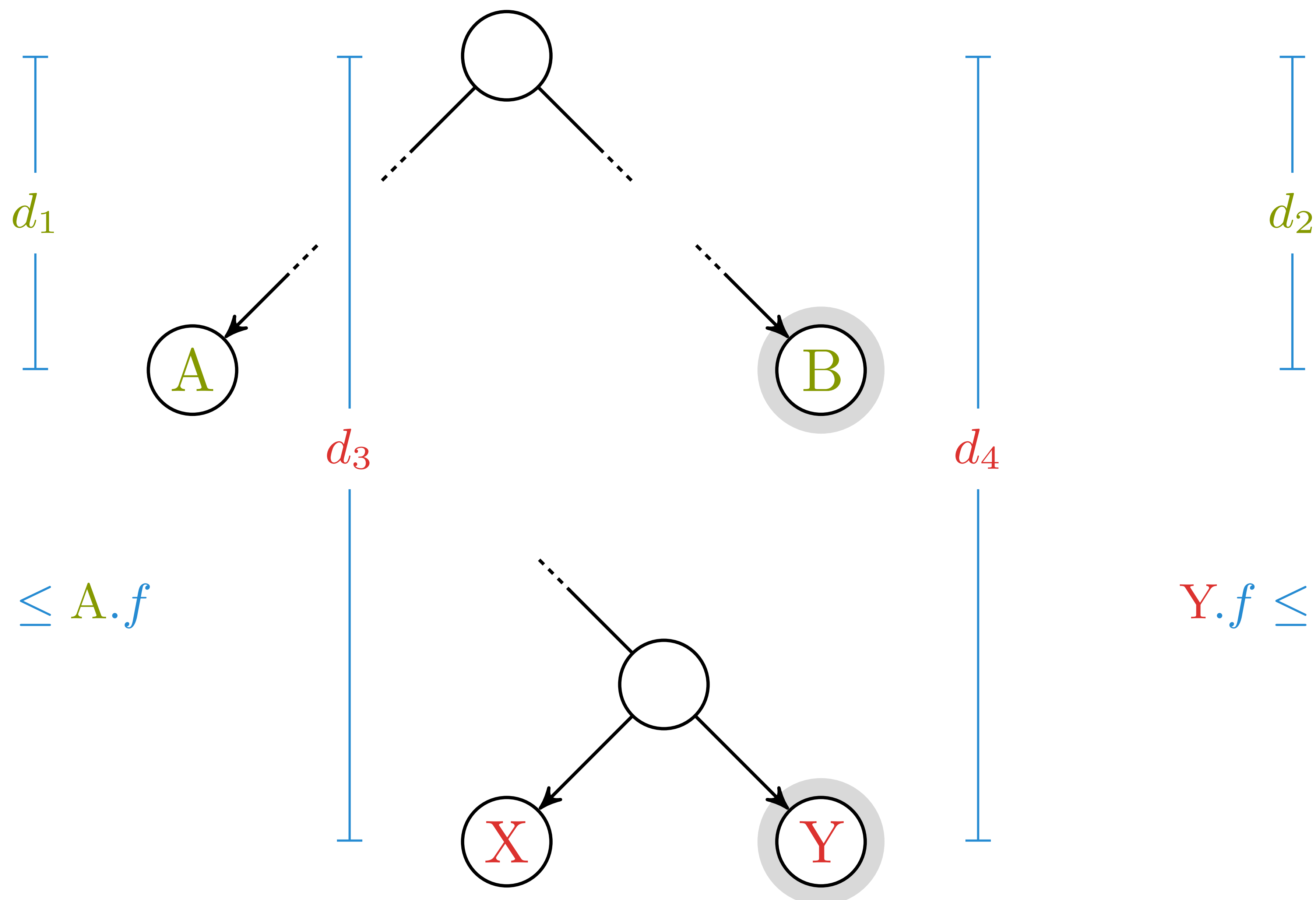
$\boxed{d_1\mathrm{X}.f + d_3\mathrm{A}.f}$



$$\boxed{d_1 X.f + d_3 A.f} \quad \ge \quad \boxed{d_1 A.f + d_3 X.f}$$



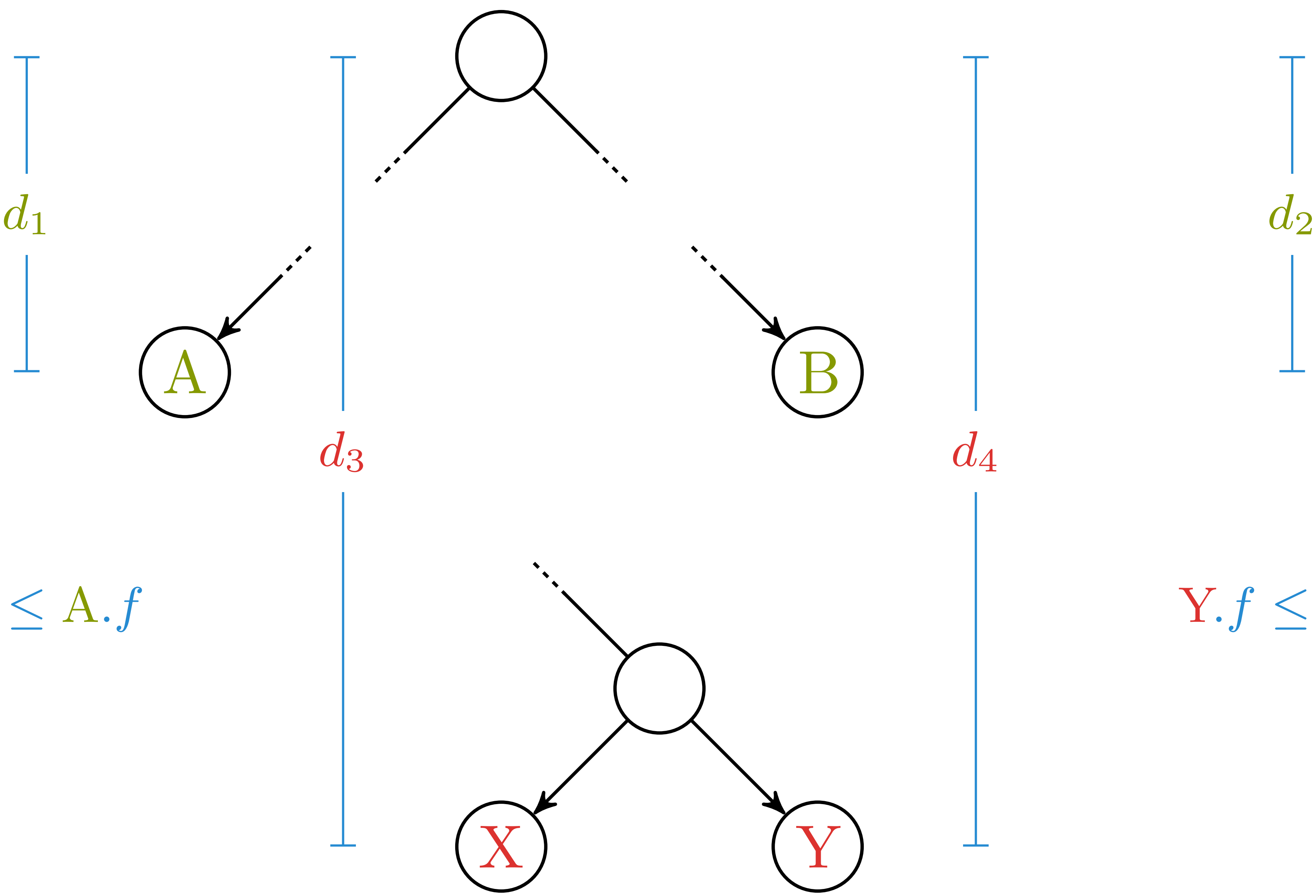
$$\boxed{d_2\mathrm{Y}.f + d_4\mathrm{B}.f}$$



$\mathrm{X}.f \leq \mathrm{A}.f$

$\mathrm{Y}.f \leq \mathrm{B}.f$

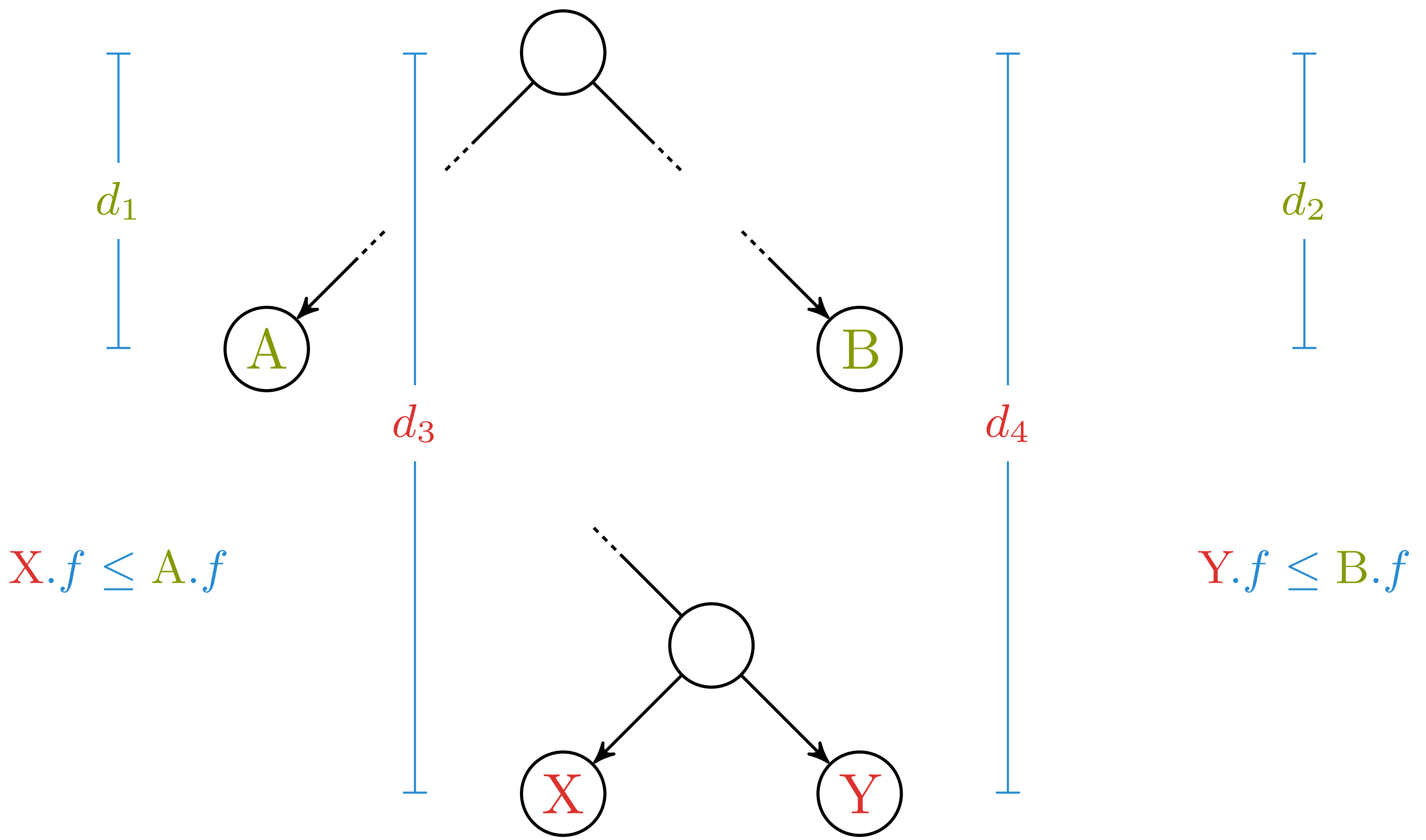
$$\boxed{d_2\mathrm{Y}.f + d_4\mathrm{B}.f} \quad \geq \quad \boxed{d_2\mathrm{B}.f + d_4\mathrm{Y}.f}$$



$X.f \leq A.f$

$Y.f \leq B.f$

$$d_1 X.f + d_2 Y.f + d_3 A.f + d_4 B.f \quad \geq \quad d_1 A.f + d_2 B.f + d_3 X.f + d_4 Y.f$$



Med andre ord: Vi taper ikke på å ha X og Y sammen nederst

Greedy choice!

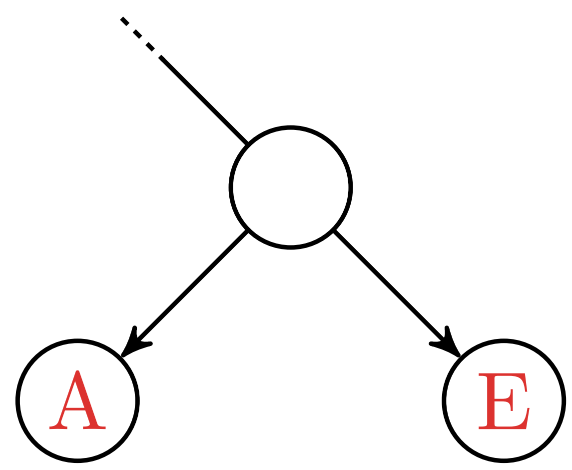
Altså: Å slå sammen X og Y er trygt som første trinn

Dvs., vi kan fortsatt få en optimal løsning dersom vi begynner med å slå sammen de to minste.

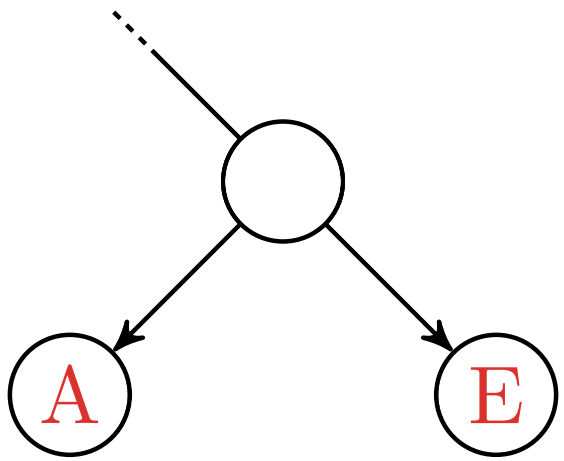
Optimal delstruktur?

Men: Kan vi fortsette på samme måte?

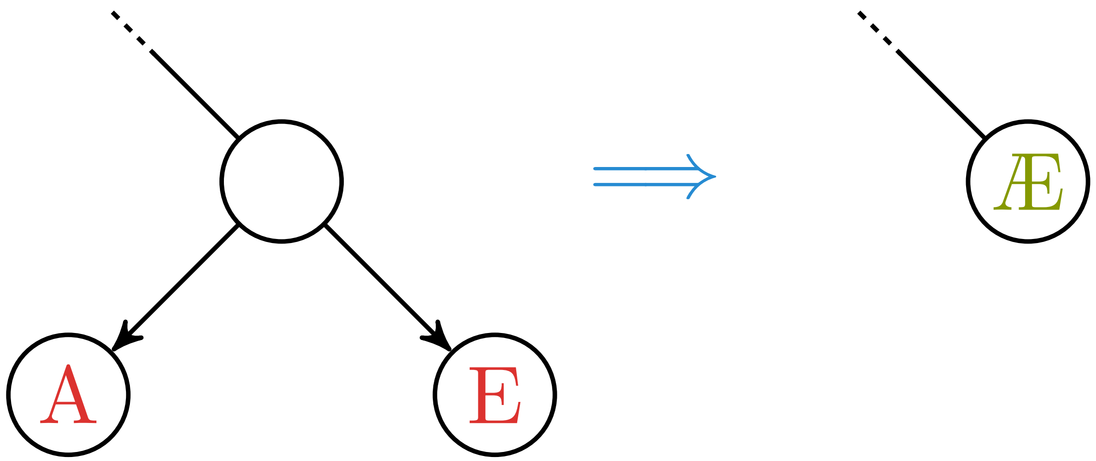
Det at det er trygt som et første trinn betyr ikke at vi bare kan fortsette sånn … det må vi også bevise (optimal substruktur).



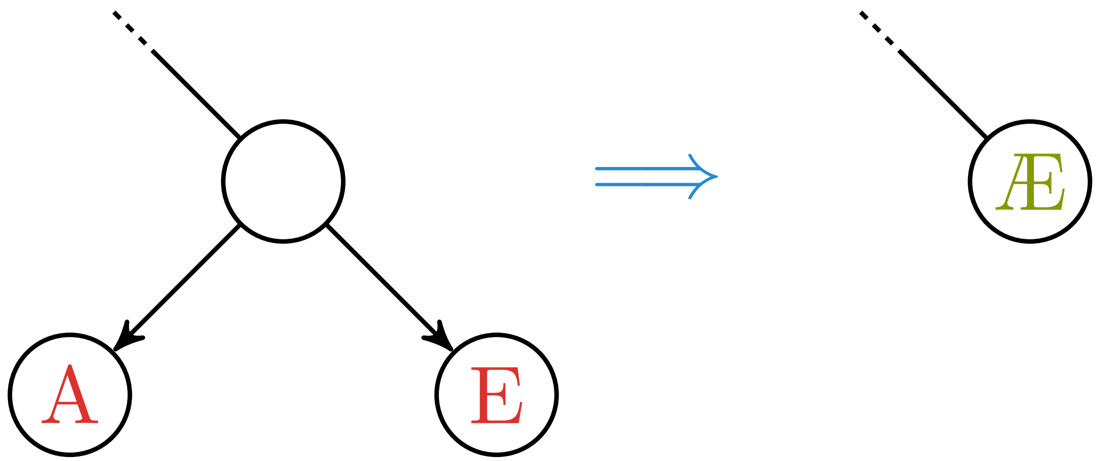
Bør resten bygges optimalt?



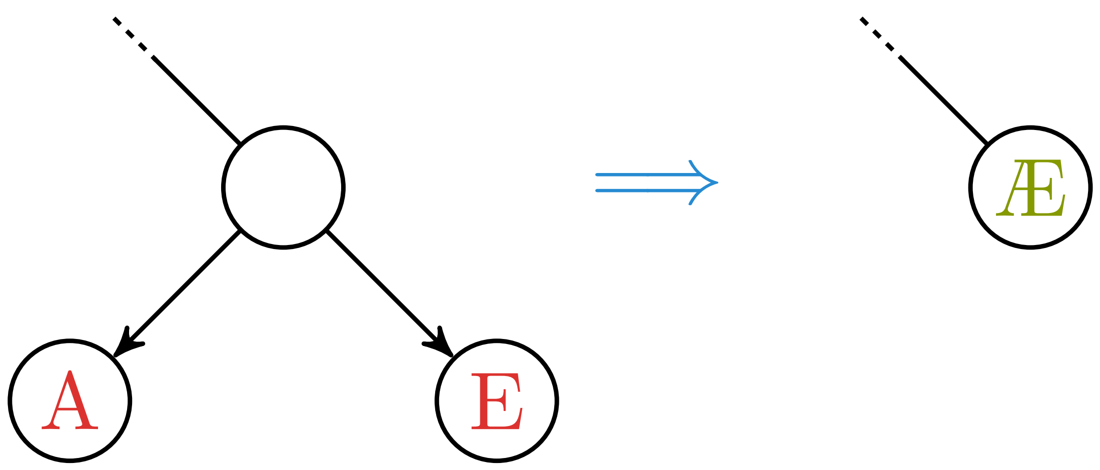
Konseptuelt: Behandle de to som ett tegn



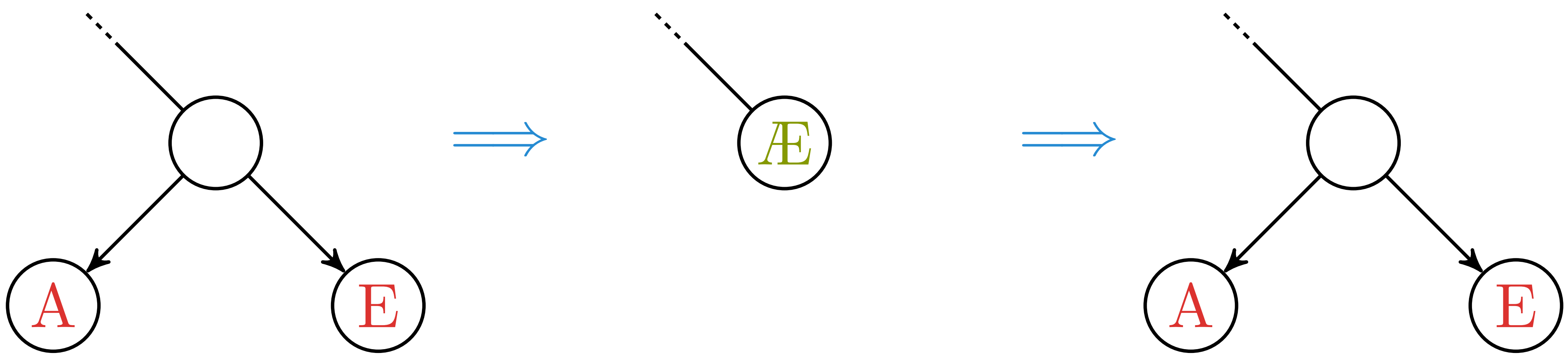
Konseptuelt: Behandle de to som ett tegn



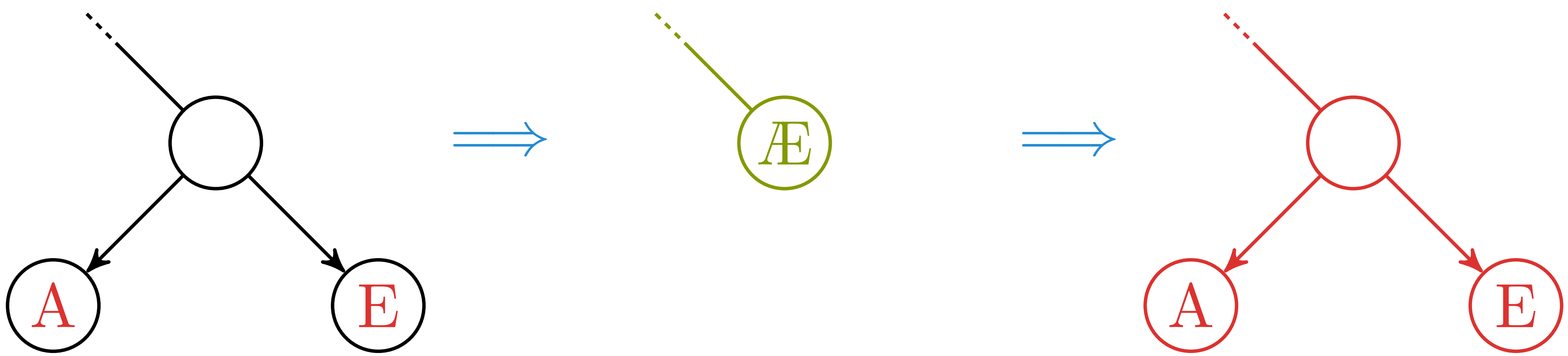
Løs det resulterende problemet



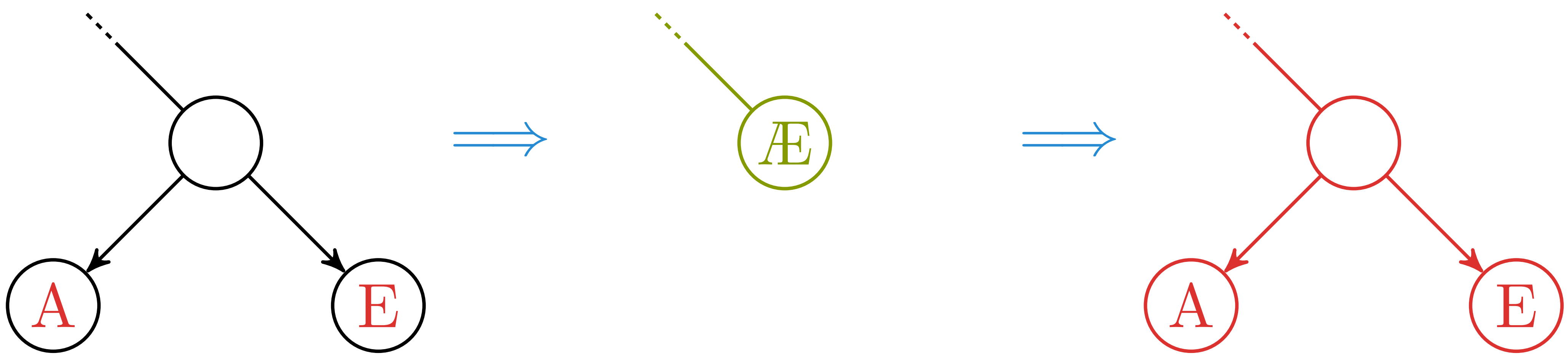
Del opp hybrid-tegnet igjen



Del opp hybrid-tegnet igjen



Suboptimal delløsning?



Suboptimal delløsning? Da kan vi forbedre løsningen!

- **Om vi velger grådig …**
- **… og løser resten optimalt …**
- **… så blir løsningen optimal.**
- **«Resten» har samme form som originalen**
- **Ved induksjon:**
  - **Vi kan velge grådig hele veien!**

1. **Grådighet › hva er det?**
2. **Eksempel: Ryggsekk**
3. **Eksempel: Aktivitetsutvalg**
4. **Eksempel: Huffman**