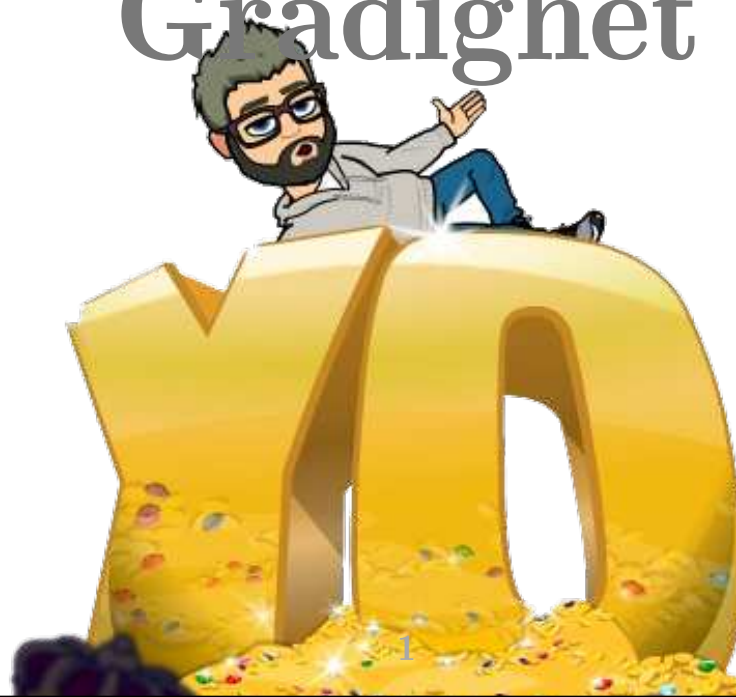
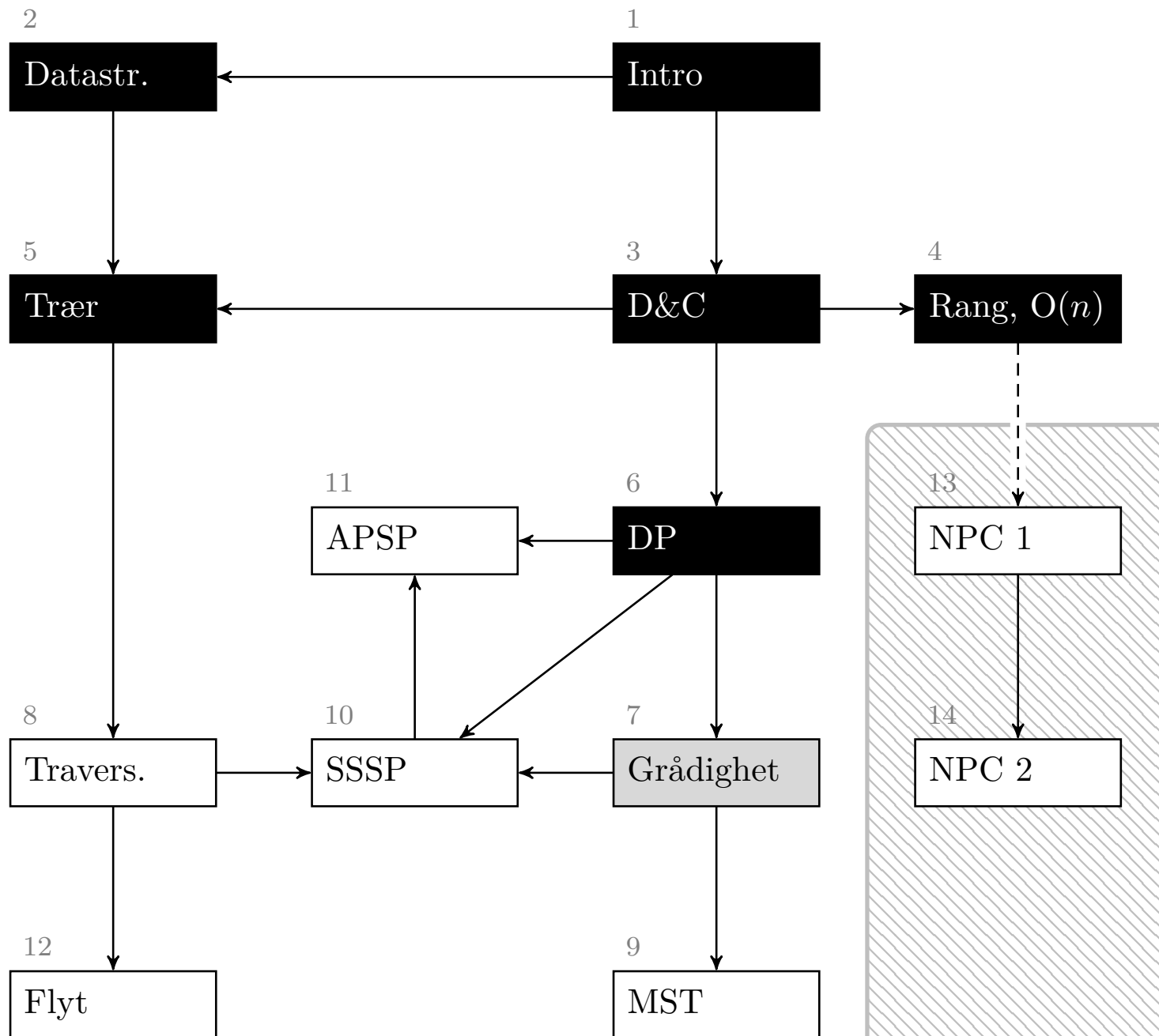


# Forelesning 7

Grådighet



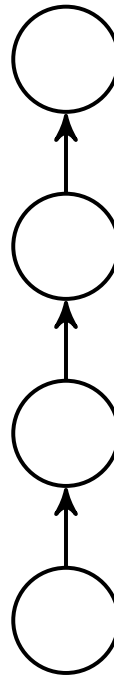


- 1. Grådighet › hva er det?**
- 2. Eksempel: Ryggsekk**
- 3. Eksempel: Aktivitetsutvalg**
- 4. Eksempel: Huffman**

# 1:4

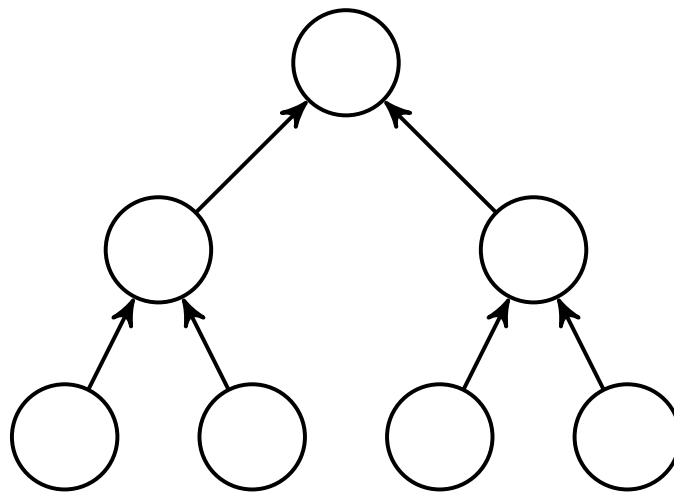
Grådighet › Hva er det?

grådighet › hva er det?



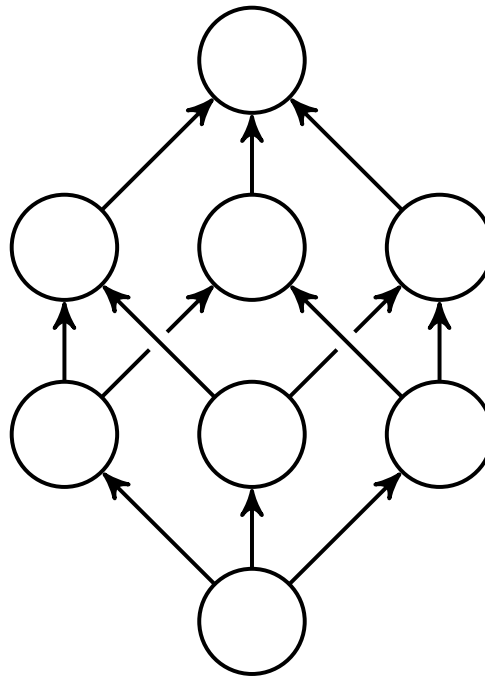
Inkrementell design

grådighet › hva er det?



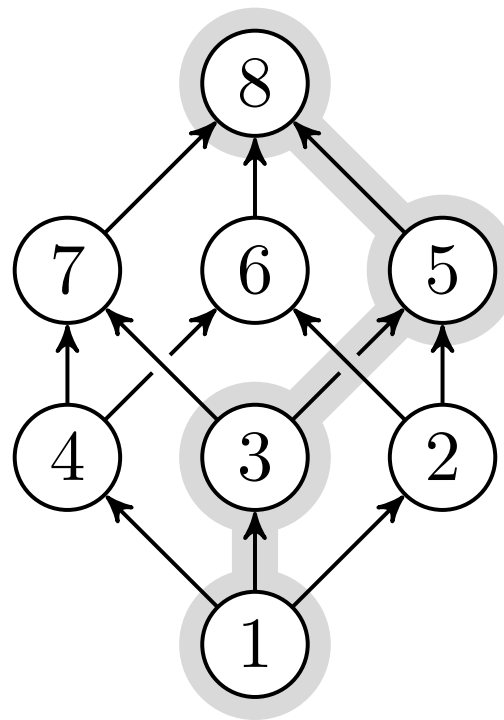
Uavhengige delproblemer: Splitt og hersk

grådighet › hva er det?



Overlappende delproblemer: Dynamisk programmering

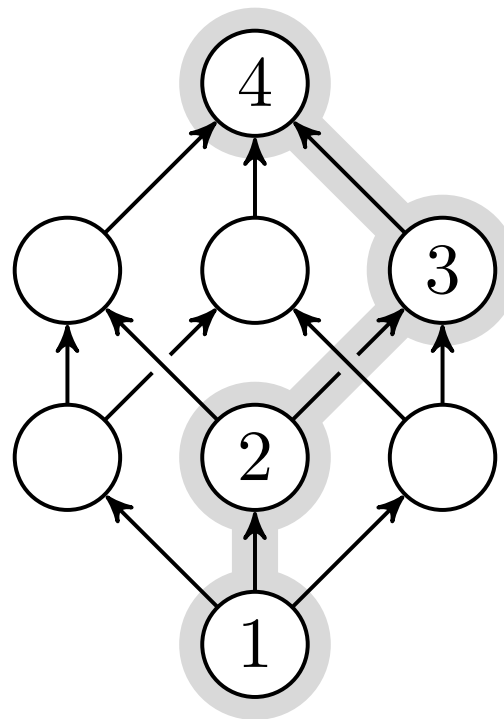
grådighet › hva er det?



Hvis DP tar valg: Løs delproblemer først



grådighet › hva er det?



Grådighet: Velg med én gang!

- **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 identifisere

1. Globalt optimalitetskriterium
2. Lokalt optimalitetskriterium
3. Konstruksjonstrinn

Algoritmen velger lokalt optimum i hvert konstruksjonstrinn. Vi må vise at det leder til et globalt optimum.

Se «The classification of greedy algorithms» av S. A. Curtis.

# Ting å vise

## 1. Grådighetsegenskapen

Vi kan velge det som ser best ut,  
her og nå

## 2. Optimal substruktur

En optimal løsning bygges av  
optimale delløsninger

**Dvs.**

**Grådig valg + optimal  
delløsning gir optimal løsning**

- 1. Formulér 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**

**Eksempel: Ryggsekk**

**Input:** Verdier  $v_1, \dots, v_n$ , vektorer  $w_1, \dots, w_n$  og en kapasitet  $W$ .

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





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

# 3:4

Eksempel: Aktivitetsutvalg

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

**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!

REC-ACT-SEL( $s, f, k, n$ )

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

REC-ACT-SEL( $s, f, k, n$ )

1  $m = k + 1$

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  $s[m] < f[k]$

Begynner  $a_m$  før  $a_k$  slutter?

REC-ACT-SEL( $s, f, k, n$ )

1  $m = k + 1$

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

3      $m = m + 1$

Prøv neste aktivitet i stedet



REC-ACT-SEL( $s, f, k, n$ )

1  $m = k + 1$

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

3      $m = m + 1$

4 **if**  $m \leq n$

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

REC-ACT-SEL( $s, f, k, n$ )

1  $m = k + 1$

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

3      $m = m + 1$

4 **if**  $m \leq n$

5      $S = \text{REC-ACT-SEL}(s, f, m, n)$

Finn resten av løsningen rekursivt

REC-ACT-SEL( $s, f, k, n$ )

1  $m = k + 1$

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

3      $m = m + 1$

4 **if**  $m \leq n$

5      $S = \text{REC-ACT-SEL}(s, f, m, n)$

6     **return**  $\{a_m\} \cup S$

Vi har alt valgt  $a_m$ ; kombinér med resten

```
REC-ACT-SEL( $s, f, k, n$ )  
1   $m = k + 1$   
2  while  $m \leq n$  and  $s[m] < f[k]$   
3       $m = m + 1$   
4  if  $m \leq n$   
5       $S = \text{REC-ACT-SEL}(s, f, m, n)$   
6      return  $\{a_m\} \cup S$   
7  else return  $\emptyset$ 
```

Vi fant ingen gyldige intervaller

grådighet › aktivitetsutvalg

REC-ACT-SEL( $s, f, k, n$ )

1  $m = k + 1$

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

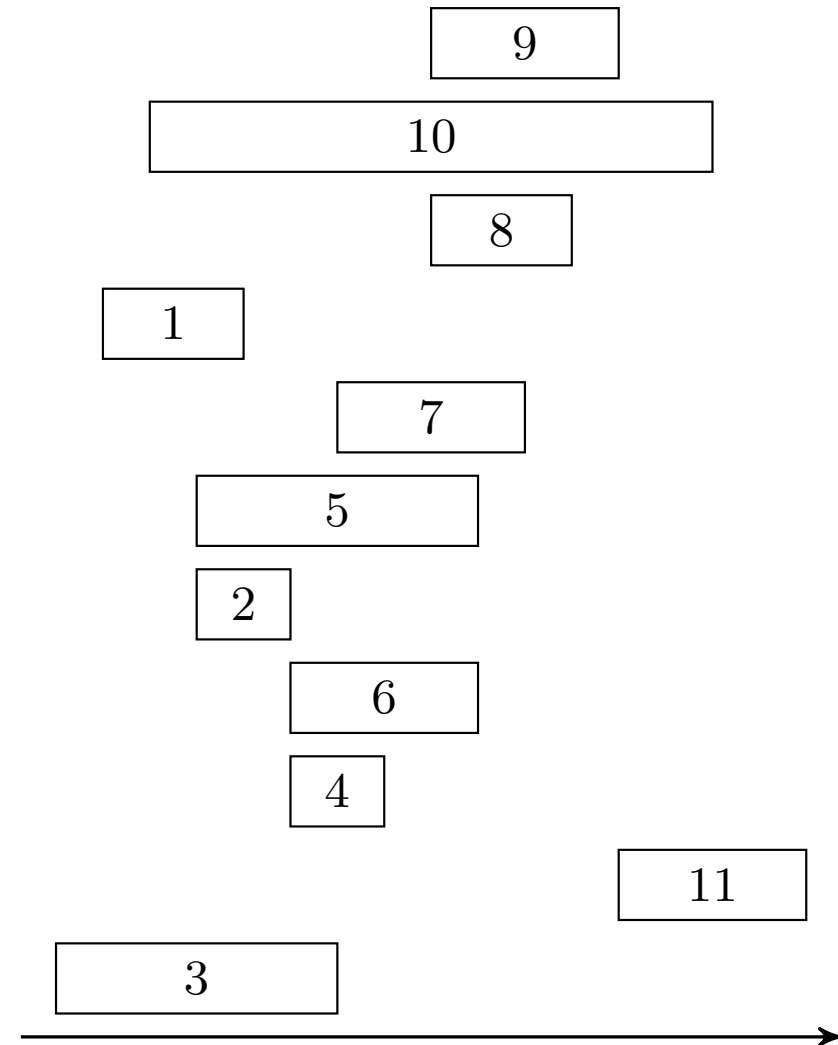
3      $m = m + 1$

4 **if**  $m \leq n$

5      $S = \text{REC-ACT-SEL}(s, f, m, n)$

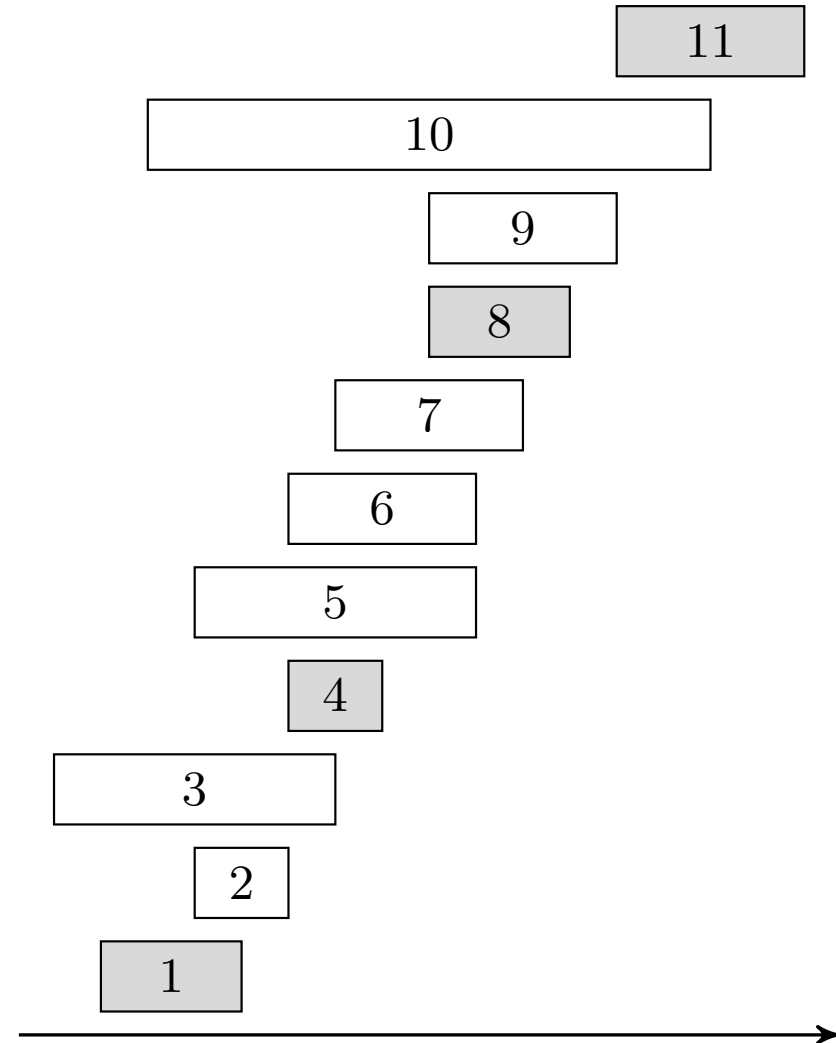
6     **return**  $\{a_m\} \cup S$

7 **else return**  $\emptyset$



REC-ACT-SEL( $s, f, k, n$ )

```
1  $m = k + 1$ 
2 while  $m \leq n$  and  $s[m] < f[k]$ 
3      $m = m + 1$ 
4 if  $m \leq n$ 
5      $S = \text{REC-ACT-SEL}(s, f, m, n)$ 
6     return  $\{a_m\} \cup S$ 
7 else return  $\emptyset$ 
```



GREEDY-ACTIVITY-SELECTOR( $s, f$ )

Iterativ omskriving av REC-ACT-SEL

GREEDY-ACTIVITY-SELECTOR( $s, f$ )

1  $n = s.length$

2  $A = \{a_1\}$

Det lønner seg alltid å ta med første element



GREEDY-ACTIVITY-SELECTOR( $s, f$ )

1  $n = s.length$

2  $A = \{a_1\}$

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

Finn første lovlige (sortert etter  $f$ )

GREEDY-ACTIVITY-SELECTOR( $s, f$ )

1  $n = s.length$

2  $A = \{a_1\}$

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

4     **if**  $s[m] \geq f[k]$

Vi fant et lovlig intervall

GREEDY-ACTIVITY-SELECTOR( $s, f$ )

1  $n = s.length$

2  $A = \{a_1\}$

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

4     **if**  $s[m] \geq f[k]$

5          $A = A \cup \{a_m\}$

Legg det til i løsningen

GREEDY-ACTIVITY-SELECTOR( $s, f$ )

1  $n = s.length$

2  $A = \{a_1\}$

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

4     **if**  $s[m] \geq f[k]$

5          $A = A \cup \{a_m\}$

6          $k = m$

Dette er nå det som slutter sist av de valgte

GREEDY-ACTIVITY-SELECTOR( $s, f$ )

1  $n = s.length$

2  $A = \{a_1\}$

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

4     **if**  $s[m] \geq f[k]$

5          $A = A \cup \{a_m\}$

6          $k = m$

7 **return**  $A$

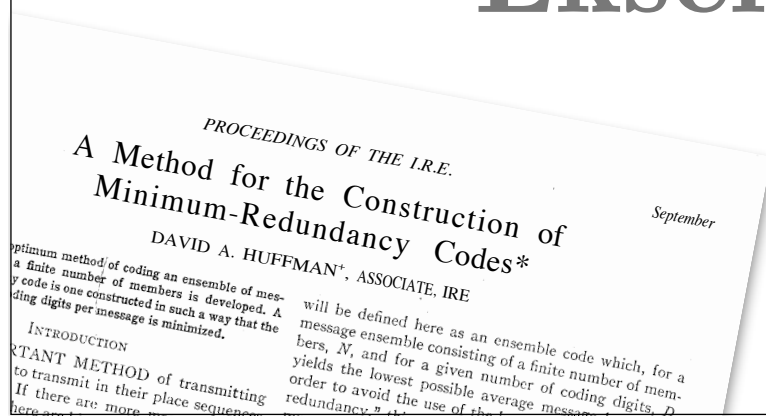
Gjerne kalt «Staying ahead» eller «Greedy stays ahead»

# Bevis ved forsprang

- Vis at grådighet er minst like bra som alle andre algoritmer for hvert trinn; det følger at den gir en optimal løsning
- Vis f.eks. induktivt at siste sluttidspunkt i grådig aktivitetsutvalg er minst like tidlig som enhver annen løsning

# 4:4

## Eksempel: Huffman



**Input:** Alfabet  $C = \{c, \dots\}$  med frekvenser  $c.freq$ .

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



- 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)

Finn kode for tegn i C som minimerer forventet tekstlengde

HUFFMAN(C)

1  $n = |C|$

2  $Q = C$

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

HUFFMAN( $C$ )

1  $n = |C|$

2  $Q = C$

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

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  $z$ 
5       $x = \text{EXTRACT-MIN}(Q)$ 
6       $y = \text{EXTRACT-MIN}(Q)$ 
7       $z.\text{left}, z.\text{right} = x, y$ 
```

Slå sammen de to minste; gi dem felles foreldrenode

HUFFMAN( $C$ )

1  $n = |C|$

2  $Q = C$

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

4     allocate a new node  $z$

5      $x = \text{EXTRACT-MIN}(Q)$

6      $y = \text{EXTRACT-MIN}(Q)$

7      $z.\text{left}, z.\text{right} = x, y$

8      $z.\text{freq} = x.\text{freq} + y.\text{freq}$

Vi later som om deltreet er ett tegn

```
HUFFMAN(C)
1   $n = |C|$ 
2   $Q = C$ 
3  for  $i = 1$  to  $n - 1$ 
4      allocate a new node  $z$ 
5       $x = \text{EXTRACT-MIN}(Q)$ 
6       $y = \text{EXTRACT-MIN}(Q)$ 
7       $z.\text{left}, z.\text{right} = x, y$ 
8       $z.\text{freq} = x.\text{freq} + y.\text{freq}$ 
9       $\text{INSERT}(Q, z)$ 
```

Sett dette «tegnet» tilbake i køen

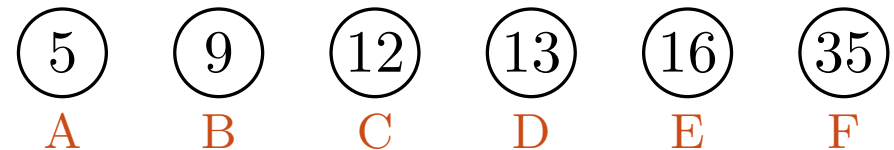


```
HUFFMAN(C)
1   $n = |C|$ 
2   $Q = C$ 
3  for  $i = 1$  to  $n - 1$ 
4      allocate a new node  $z$ 
5       $x = \text{EXTRACT-MIN}(Q)$ 
6       $y = \text{EXTRACT-MIN}(Q)$ 
7       $z.\text{left}, z.\text{right} = x, y$ 
8       $z.\text{freq} = x.\text{freq} + y.\text{freq}$ 
9       $\text{INSERT}(Q, z)$ 
10 return  $\text{EXTRACT-MIN}(Q)$ 
```

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

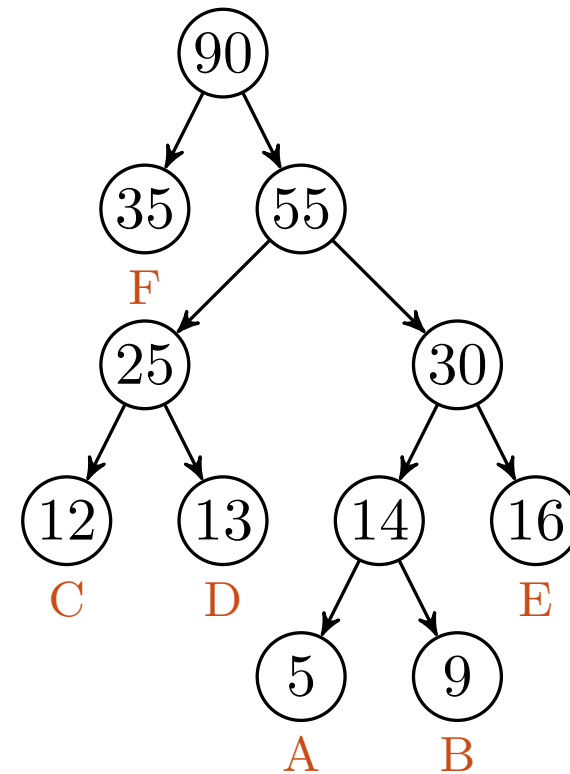
HUFFMAN( $C$ )

```
1   $n = |C|$ 
2   $Q = C$ 
3  for  $i = 1$  to  $n - 1$ 
4      allocate a new node  $z$ 
5       $x = \text{EXTRACT-MIN}(Q)$ 
6       $y = \text{EXTRACT-MIN}(Q)$ 
7       $z.\text{left}, z.\text{right} = x, y$ 
8       $z.\text{freq} = x.\text{freq} + y.\text{freq}$ 
9       $\text{INSERT}(Q, z)$ 
10 return  $\text{EXTRACT-MIN}(Q)$ 
```



HUFFMAN(C)

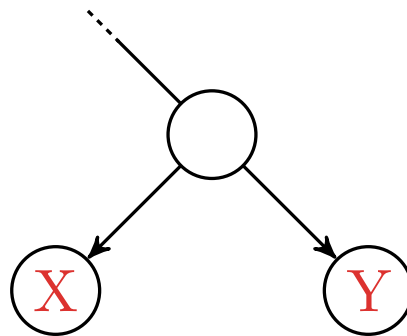
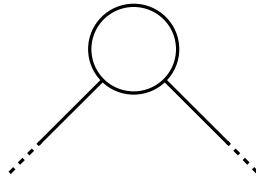
```
1   $n = |C|$ 
2   $Q = C$ 
3  for  $i = 1$  to  $n - 1$ 
4      allocate a new node  $z$ 
5       $x = \text{EXTRACT-MIN}(Q)$ 
6       $y = \text{EXTRACT-MIN}(Q)$ 
7       $z.\text{left}, z.\text{right} = x, y$ 
8       $z.\text{freq} = x.\text{freq} + y.\text{freq}$ 
9       $\text{INSERT}(Q, z)$ 
10 return  $\text{EXTRACT-MIN}(Q)$ 
```



Grådighet › Huffman ›

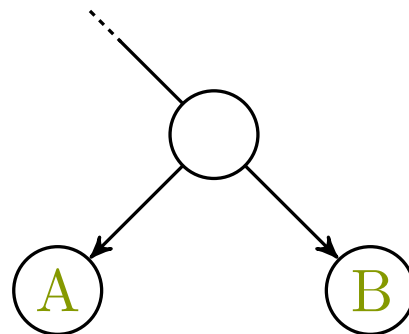
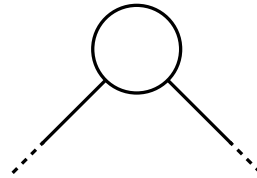
**Korrekthet**

grådighet › huffman › korrekthet



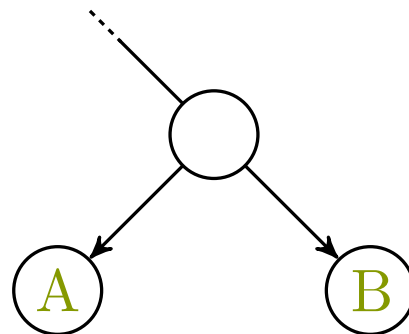
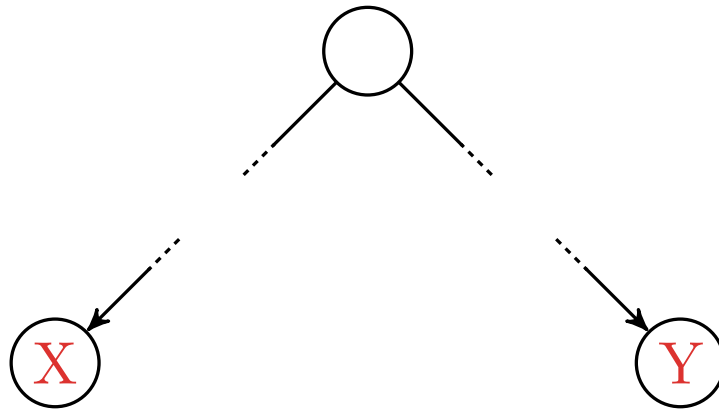
Vil vise: Sjeldneste sammen nederst lønner seg

grådighet › huffman › korrekthet

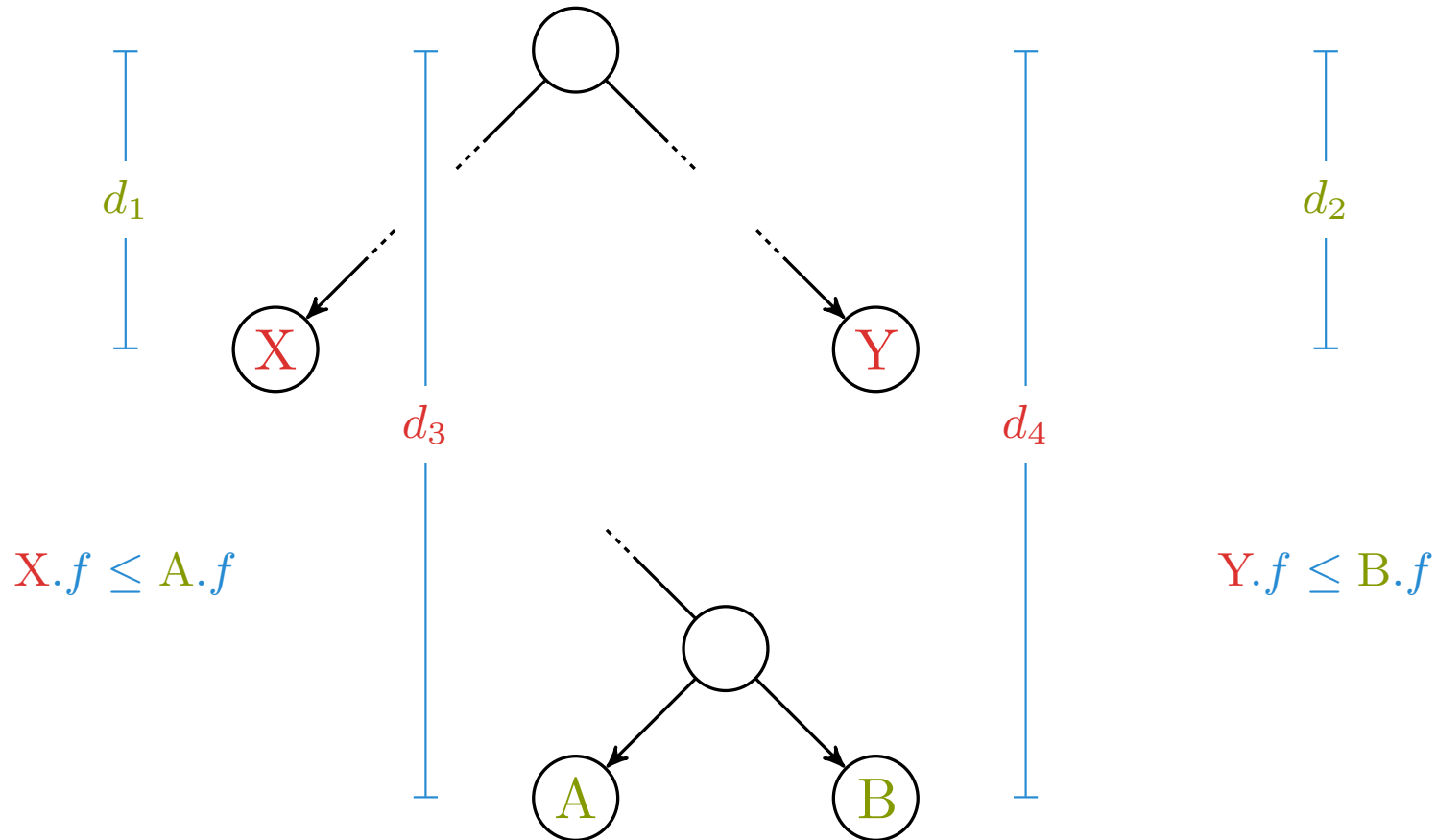


Anta en vilkårlig annen løsning ...

grådighet › huffman › korrekthet

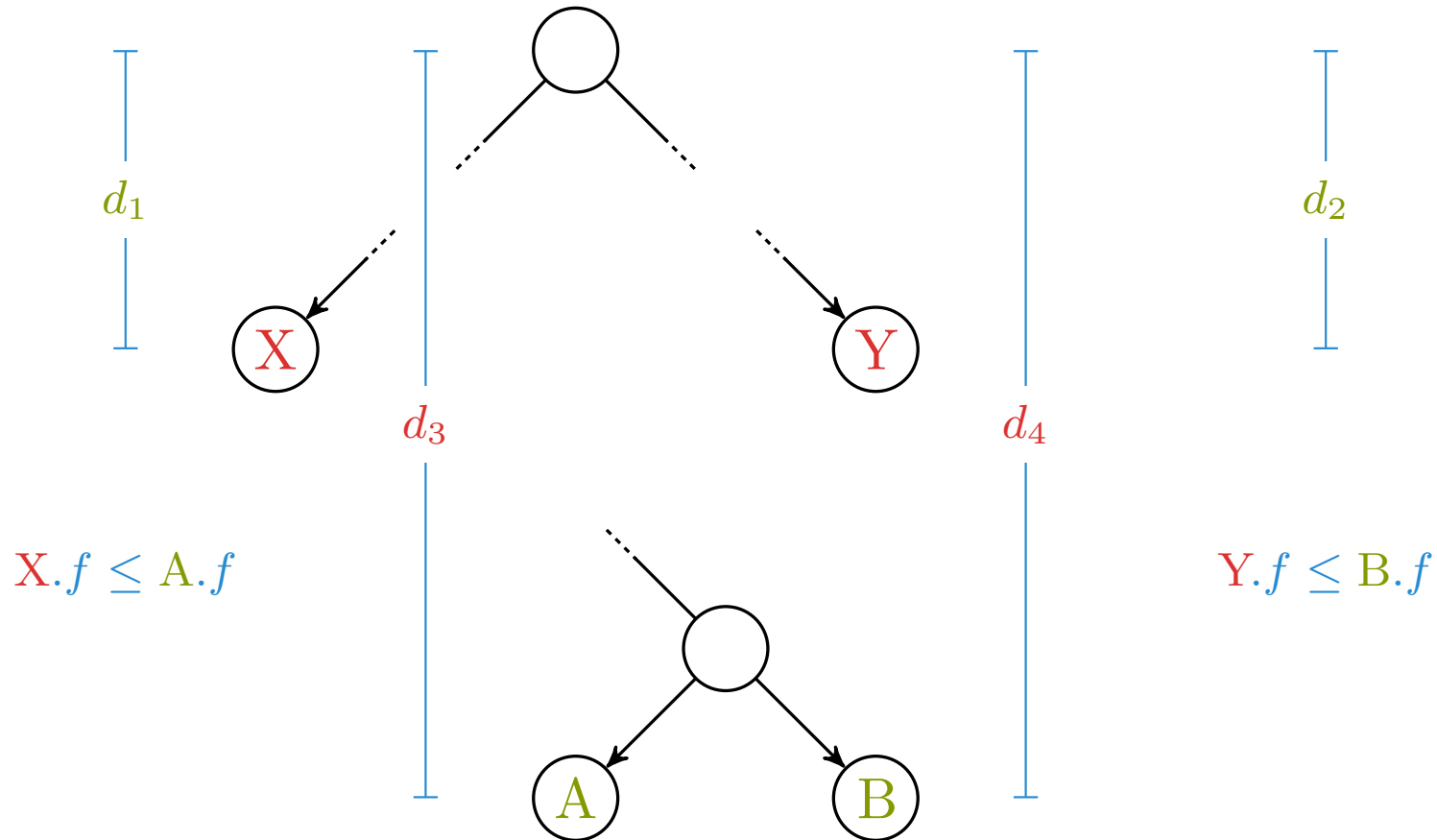


... med de sjeldneste lenger opp

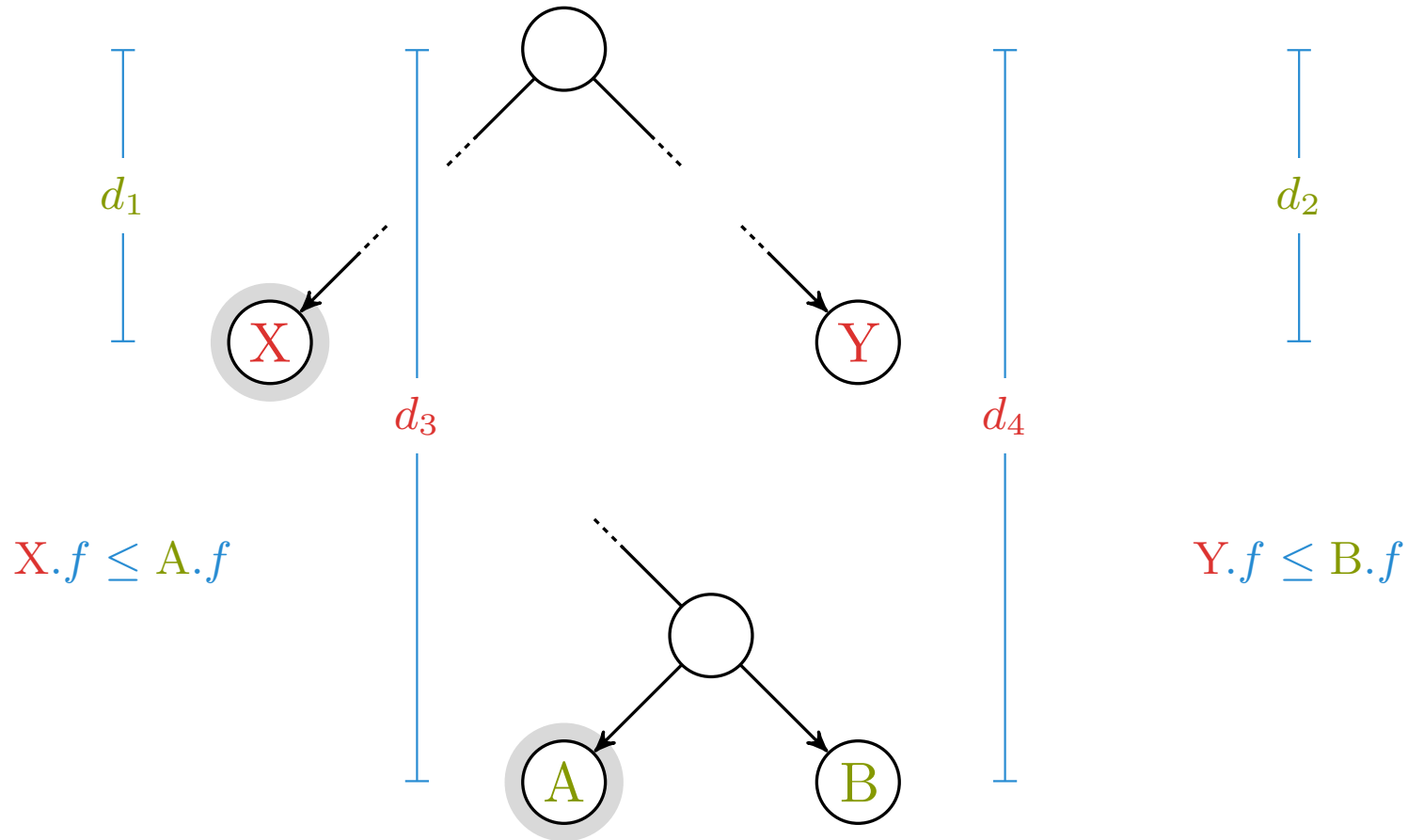


... med de sjeldneste lenger opp

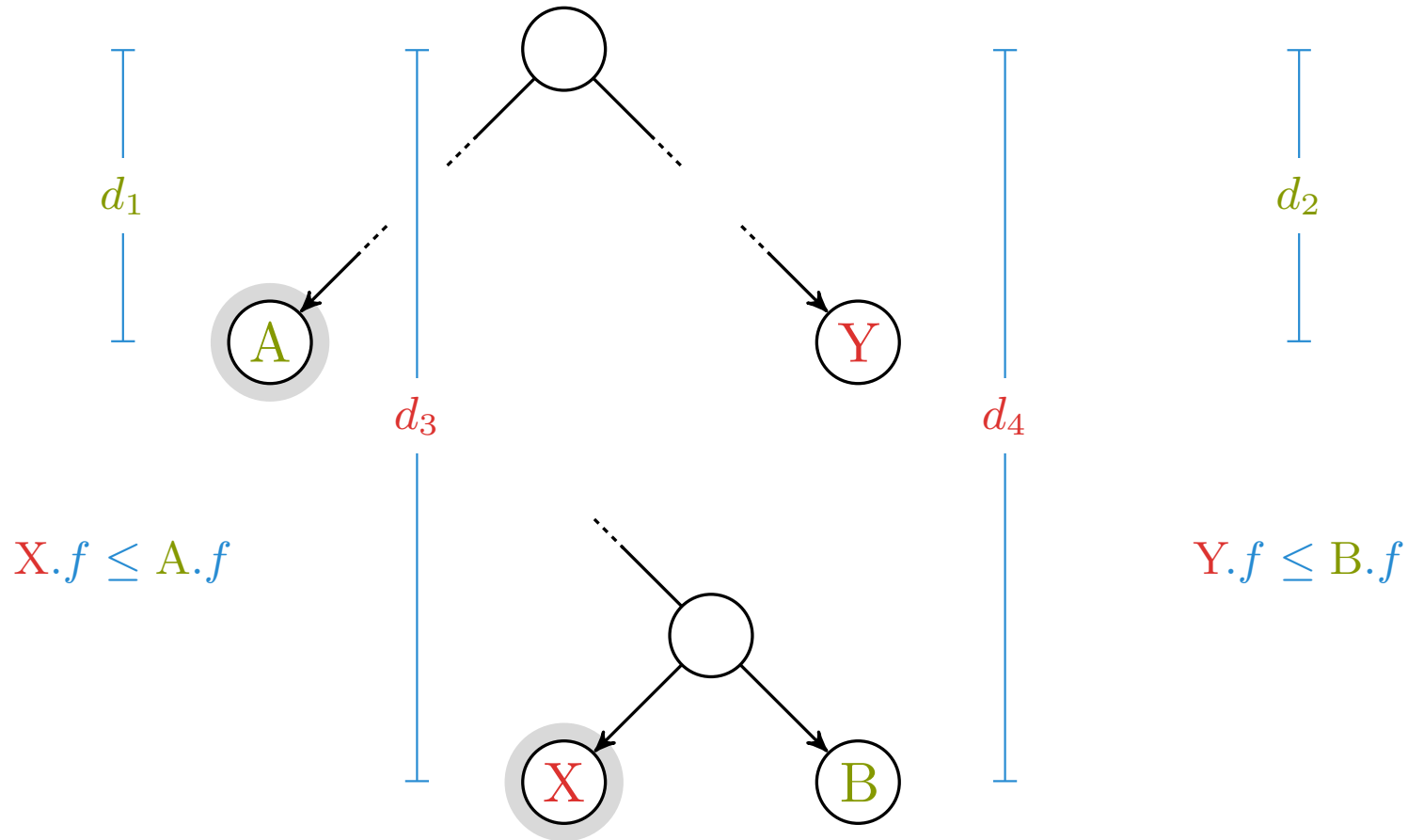




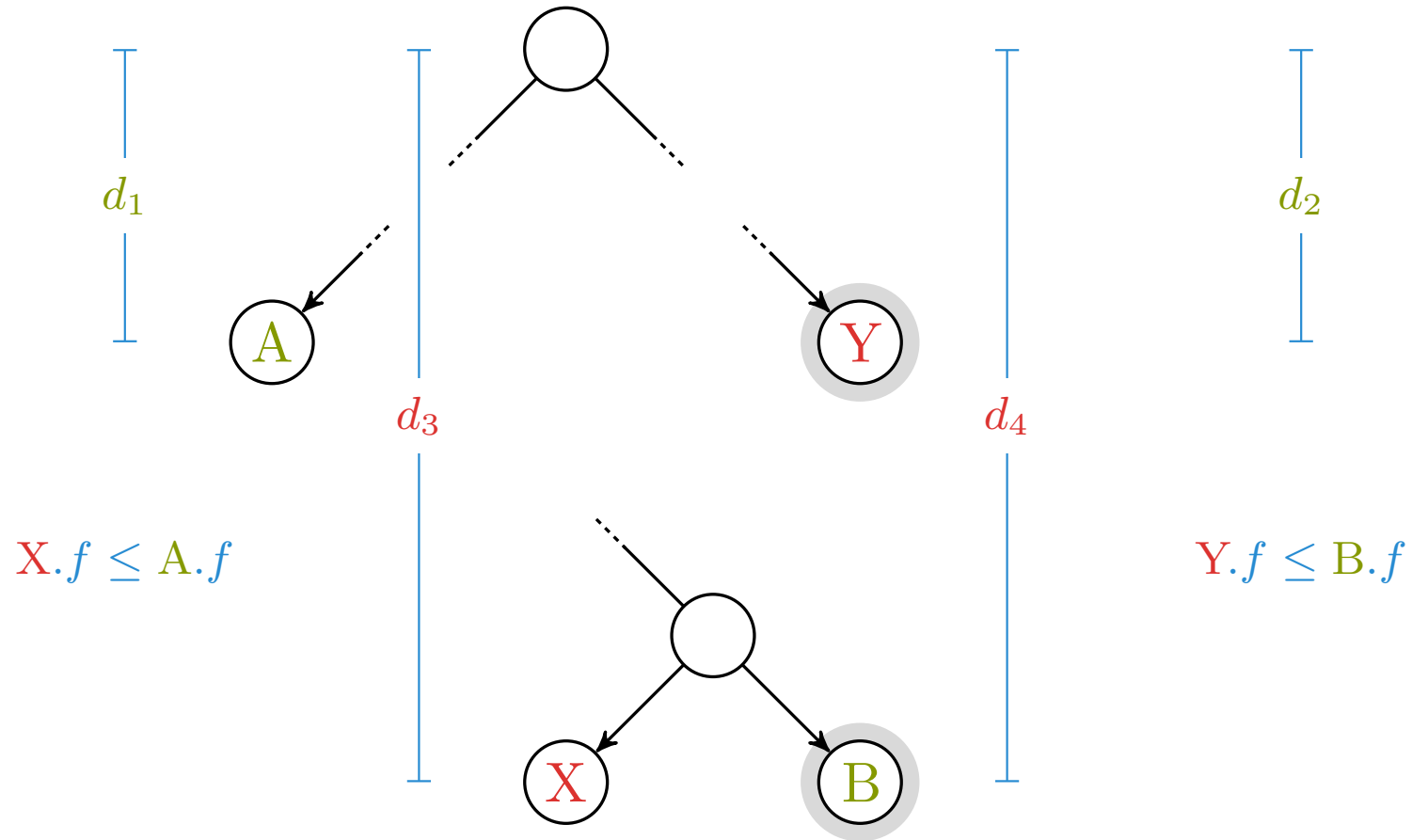
Bidrag fra  $X, Y, A, B$  er  $d_1 X.f + d_2 Y.f + d_3 A.f + d_4 B.f$



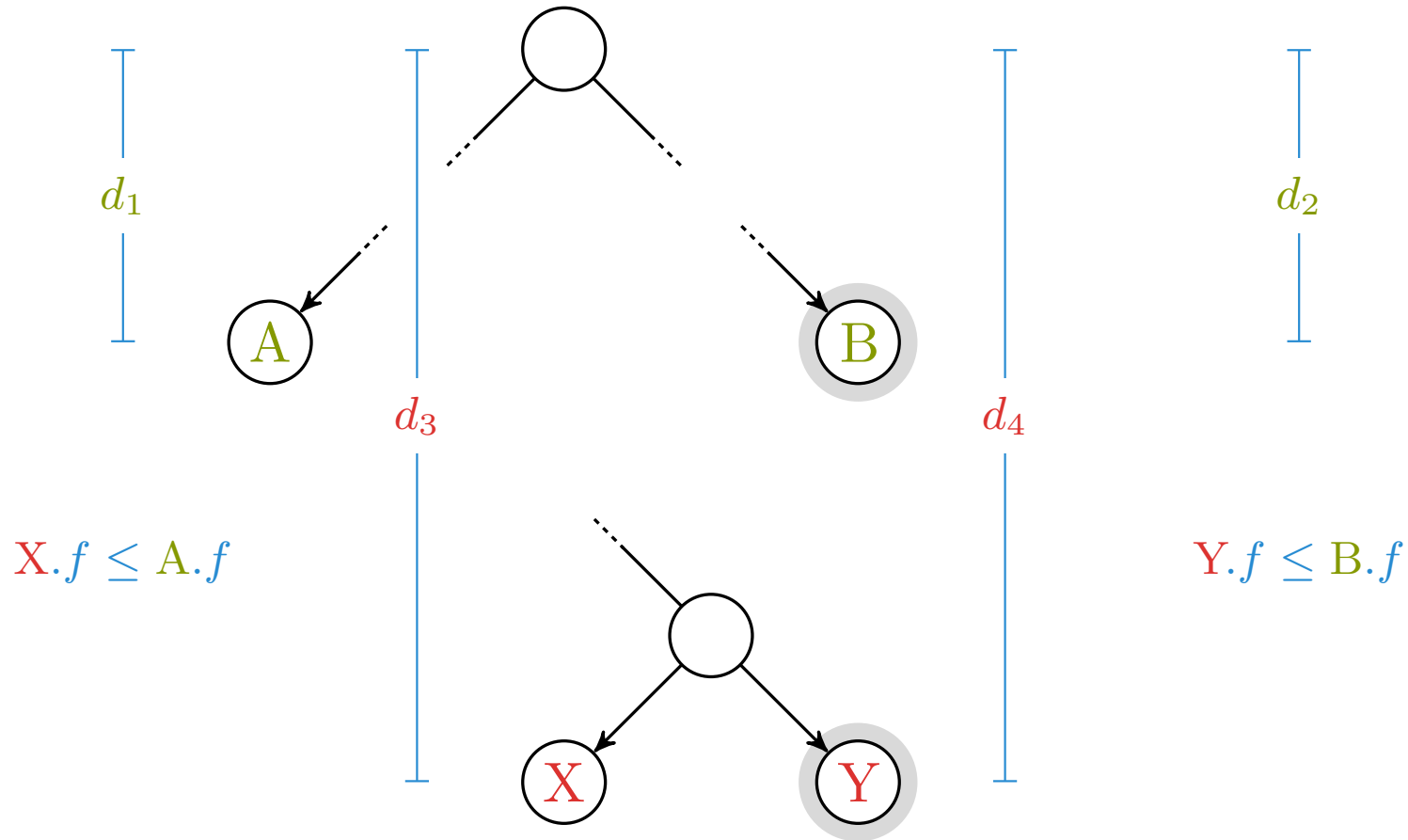
$$d_1X.f + d_3A.f$$



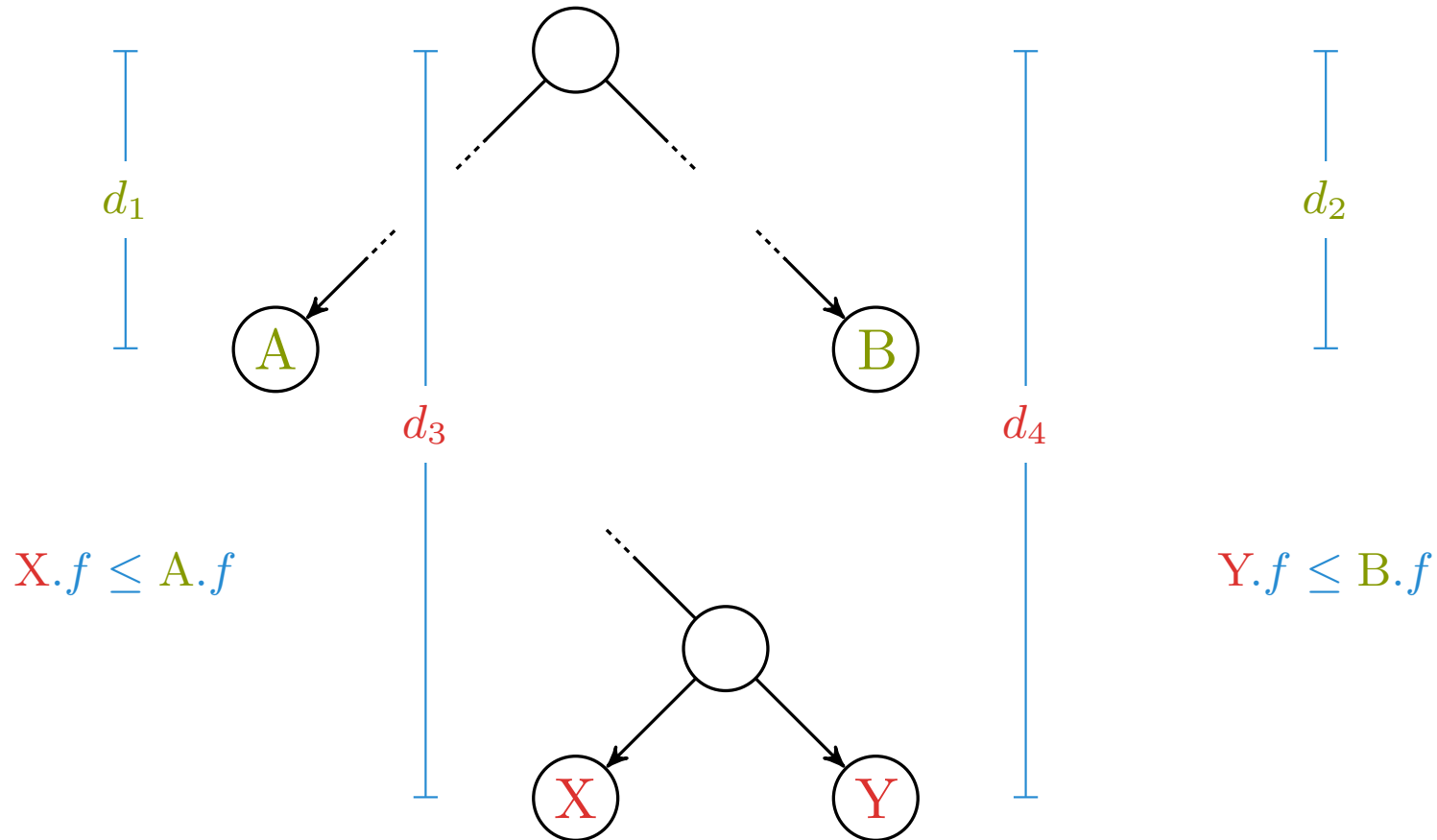
$$d_1 X.f + d_3 A.f \geq d_1 A.f + d_3 X.f$$



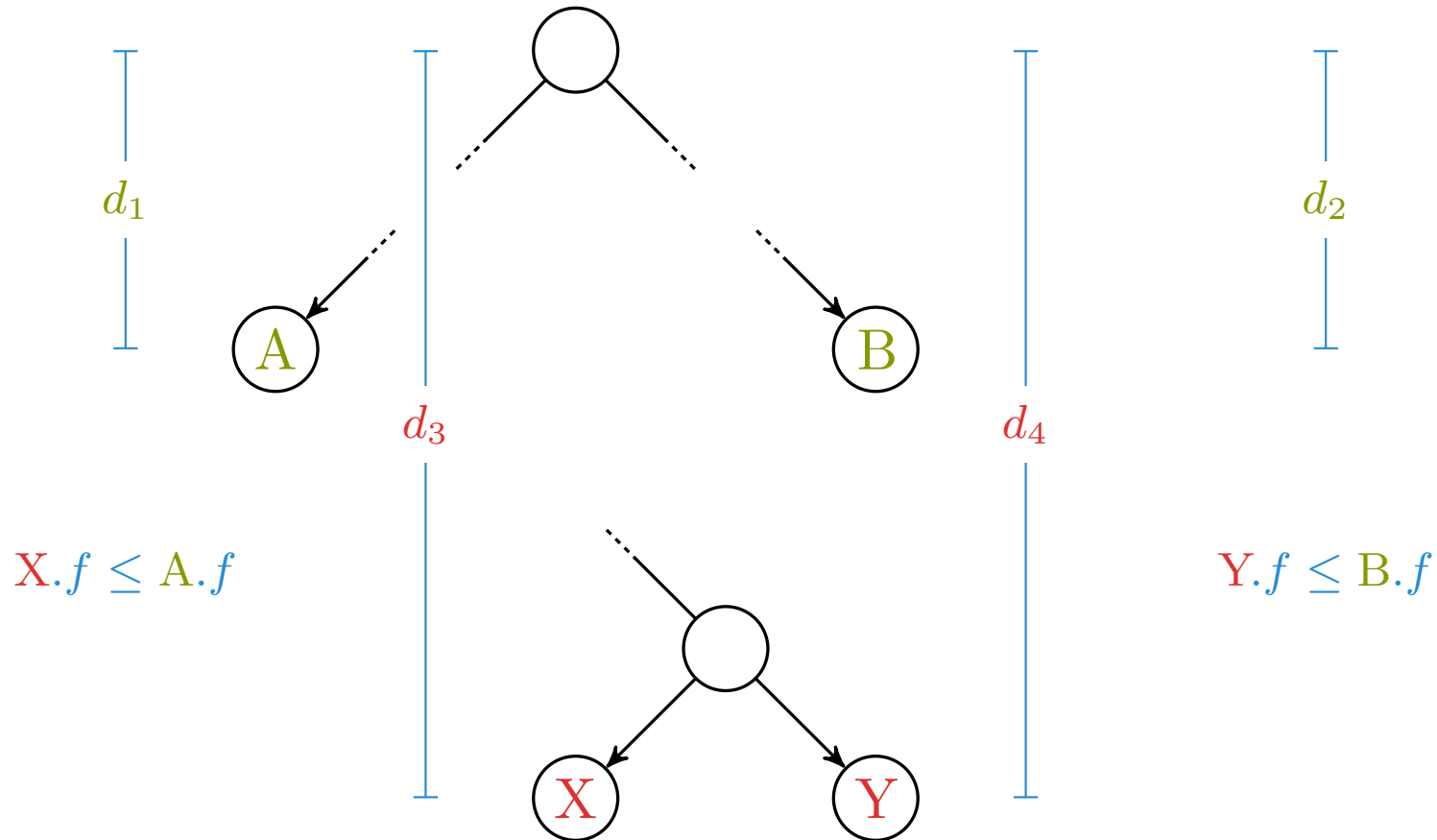
$$d_2 \text{Y}.f + d_4 \text{B}.f$$



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



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



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.

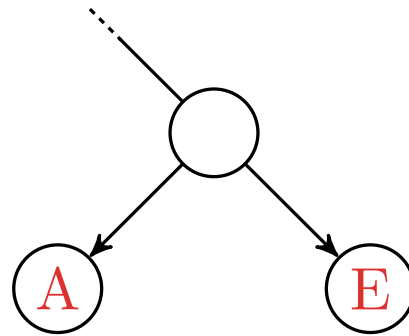


*Optimal substruktur?*

**Men: Kan vi fortsette  
på samme måte?**

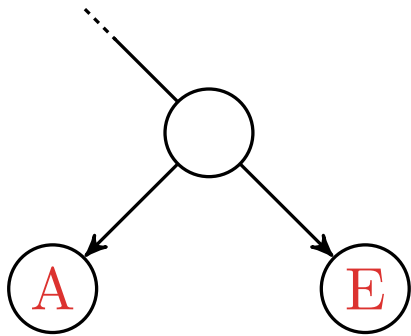
Det at det er trygt som  
et første trinn betyr  
ikke at vi bare kan  
fortsette sånn ... det må  
vi også bevise (optimal  
substruktur).

grådighet › huffman › korrekthet

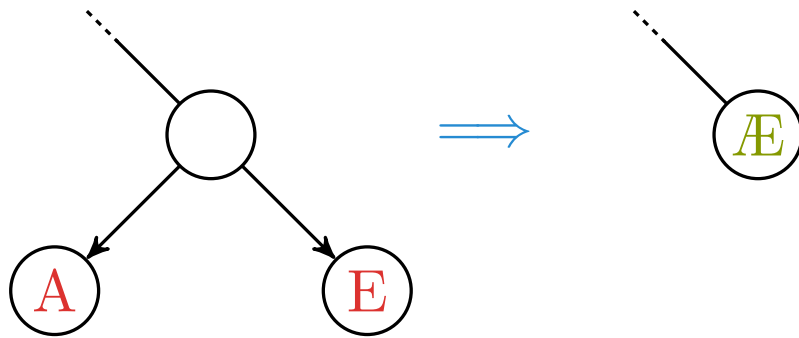


Bør resten bygges optimalt?

grådighet › huffman › korrekthet

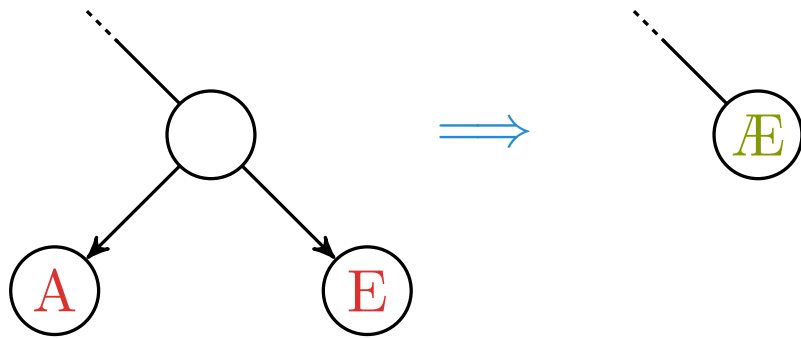


Konseptuelt: Behandle de to som ett tegn



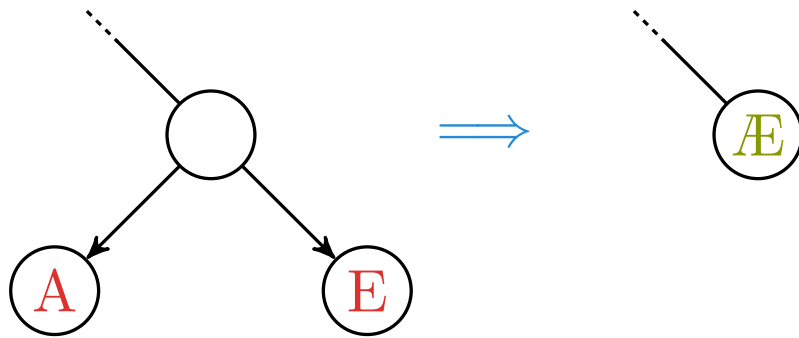
Konseptuelt: Behandle de to som ett tegn

grådighet › huffman › korrekthet



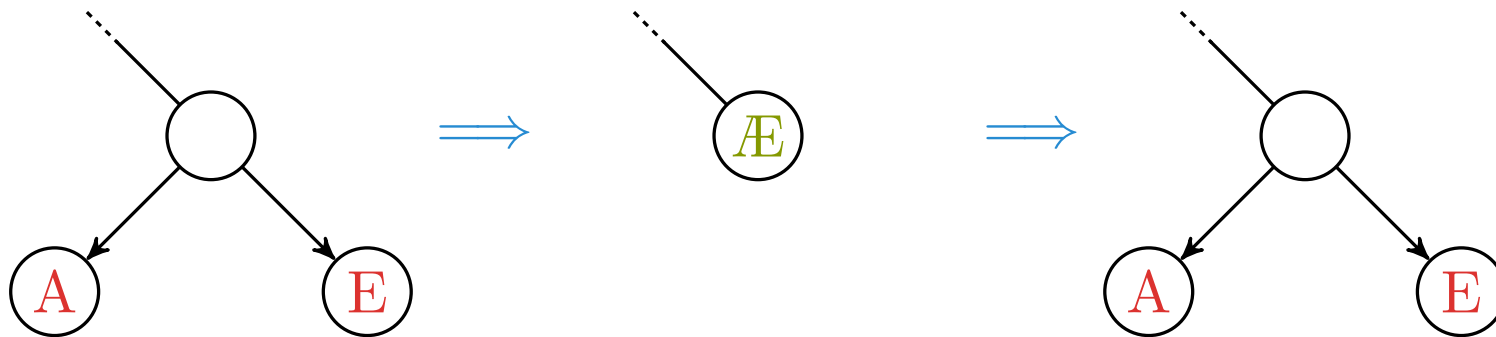
Løs det resulterende problemet

grådighet › huffman › korrekthet



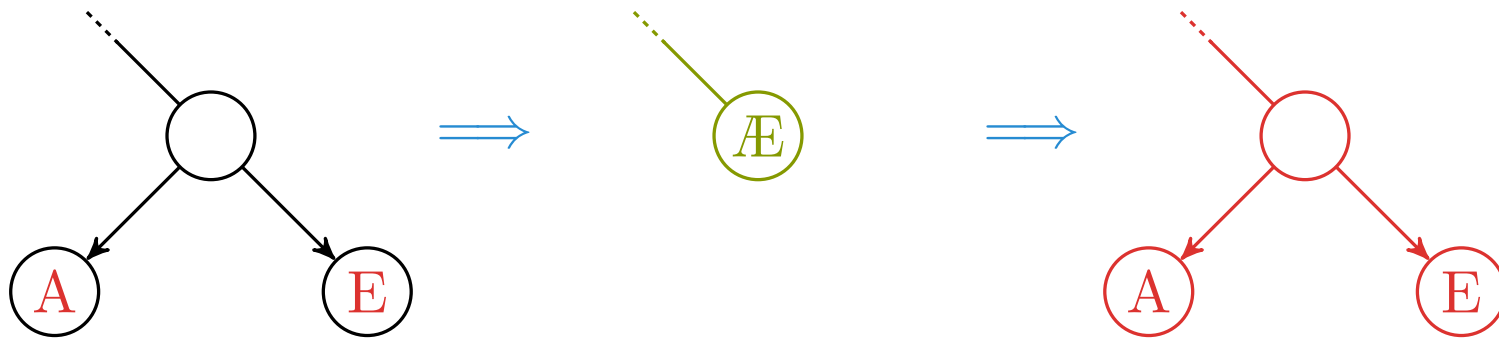
Del opp hybrid-tegnet igjen

grådighet › huffman › korrekthet



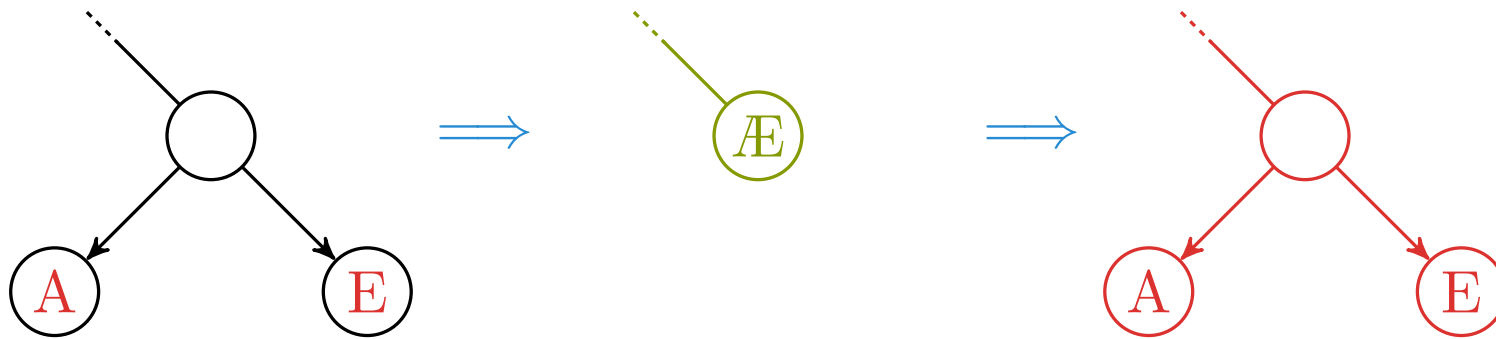
Del opp hybrid-tegnet igjen

grådighet › huffman › korrekthet



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!**

Ofte kalt «Exchange arguments»

# Bevis ved fortrinn

- Betrakt en vilkårlig løsning og transformér den gradvis til en grådig løsning, uten å senke kvaliteten. Den grådige løsningen må da være minst like god som enhver annen
- Prøv selv: Aktiviteter med varighet og deadline. Minimér største forsinkelse. Løsning: Velg hele tiden den med tidligst deadline
- Bevis optimalitet ved å starte med en optimal løsning, og så bytte om på elementer til du får den grådige. Hvordan unngår du å få en dårligere løsning?

Se <http://www.idi.ntnu.no/~mlh/algkon/greedy.pdf>

Optimum har ingen gaps. Hvis vi har en jobb som kommer senere men som har tidligere deadline, så vil vi også ha to jobber rett etter hverandre som er byttet om. Disse kan vi bytte, og få minst like bra resultat.

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

