Grundlæggende Algoritmer og Datastrukturer

Grådige Algoritmer
[CLRS 16.1-16.3]

Grådige Algoritmer

- Problemer hvor en løsning kan konstrueres ud fra en løsning for kun **ét mindre delproblem**

- Delproblemet kan identificeres effektivt

  (simplere end dynamisk programmering!)

Længste Fælles Delsekvens ?

ABABGACBABAD

BACABAABABC

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

Er *BACBABA* en delsekvens af *ABABGACBABAD* ?

Mulig forekomst ***ABABGACBABAD***

Observation Hvis der findes en forekomst, så findes også en forekomst der matcher det første ***B*** i strengen

ABABGACBABAD

Find rekursivt ***ACBABA***

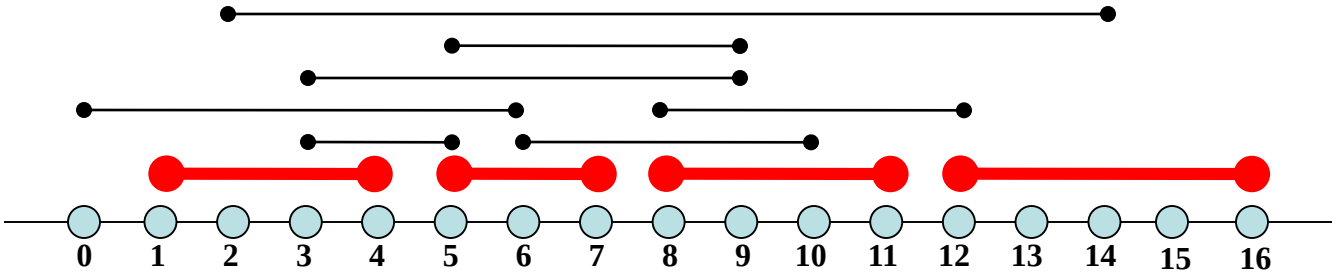
Tid $O(n)$

Hvilken løsning finder den grådige algoritme?

a) ***A<u>B</u>ABG<u>ACBABA</u>D***
b) ***A<u>BA</u>BGA<u>CBABA</u>D***
c) ***ABA<u>B</u>G<u>ACBABA</u>D***
d) Ved ikke

Udvælgelse af Aktiviteter

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

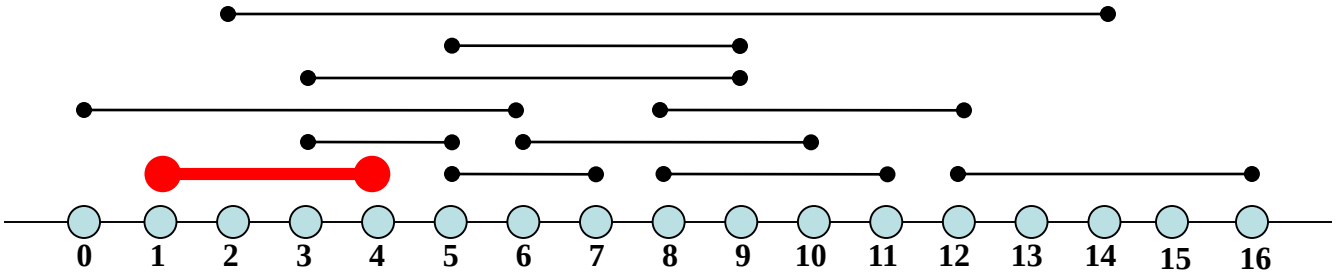


Input: n aktiviter med starttid s_i og sluttid f_i

Output: Maximal mængde ikke-overlappende aktiviteter

Udvælgelse af Aktiviteter

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16



Observation: Der findes altid en maximal løsning som indeholder aktiviteten med den *tidligste sluttid (grådig-valg egenskab)*

Preprocessering: Sorter aktiviteterne efter sluttidspunktet

Udvælgelse af Aktiviteter

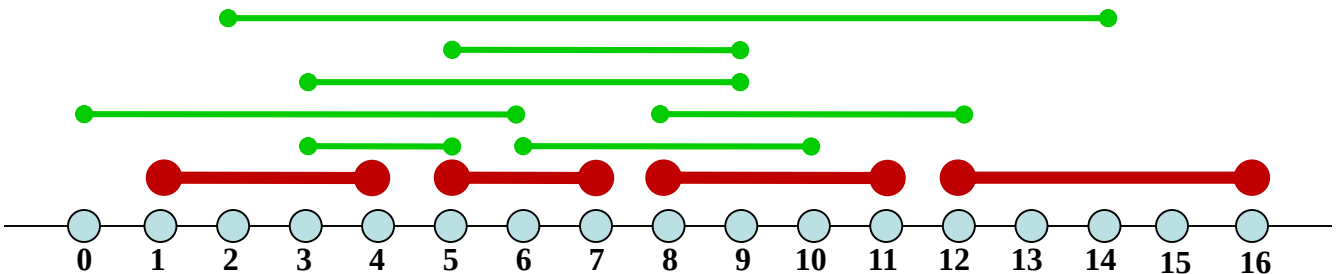
i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

← sorteret

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}
7          k = m
8  return A
```

Tid $O(n \cdot \log n)$



Huffman koder

Ascii Tabel

$83_{10} = 53_{16} = 1010011_2$

	0	1	2	3	4	5	6	7
0	NUL	DLE	space	0	@	P	`	p
1	SOH	DC1 XON	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3 XOFF	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	del

Strengen ”AU” som binær?

a) 00010010101101
b) 11000011110101
c) 10000011010101
d) 00011100101110
e) Ved ikke

	0	1	2	3	4	5	6	7
0	NUL	DLE	space	0	@	P	`	p
1	SOH	DC1 XON	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3 XOFF	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	del

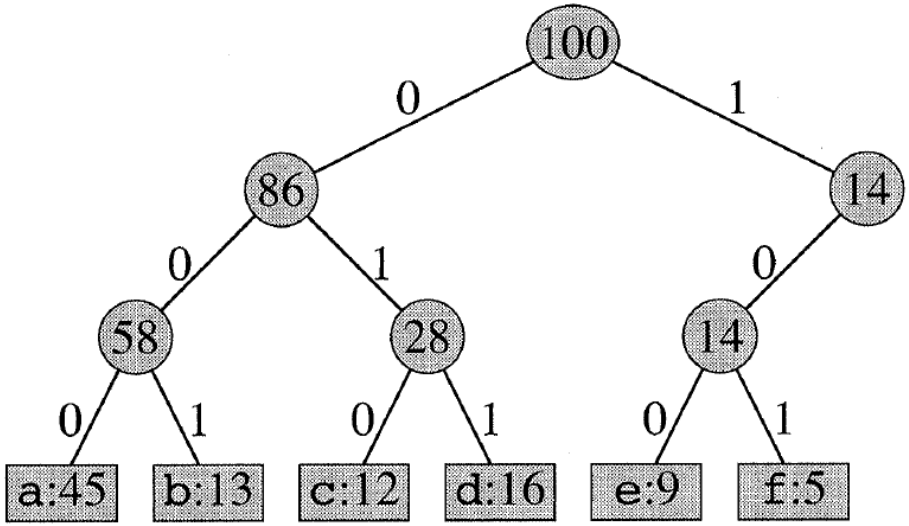
Komprimering

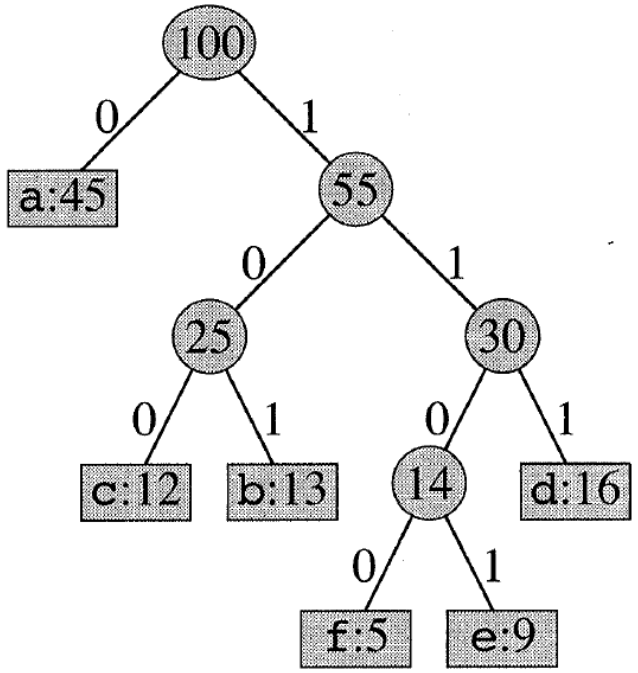
Givet frekvensen af symbolerne i input, erstat input symbolerne med **kortere bitstrenge** (fixed-længde eller variabel-længde)

	a	b	c	d	e	f
Frequency (in thousands)	45	13	12	16	9	5
Fixed-length codeword	000	001	010	011	100	101
Variable-length codeword	0	101	100	111	1101	1100

NB: Variabel-længde skal være prefix-fri

Fixed-længde vs variabel-længde





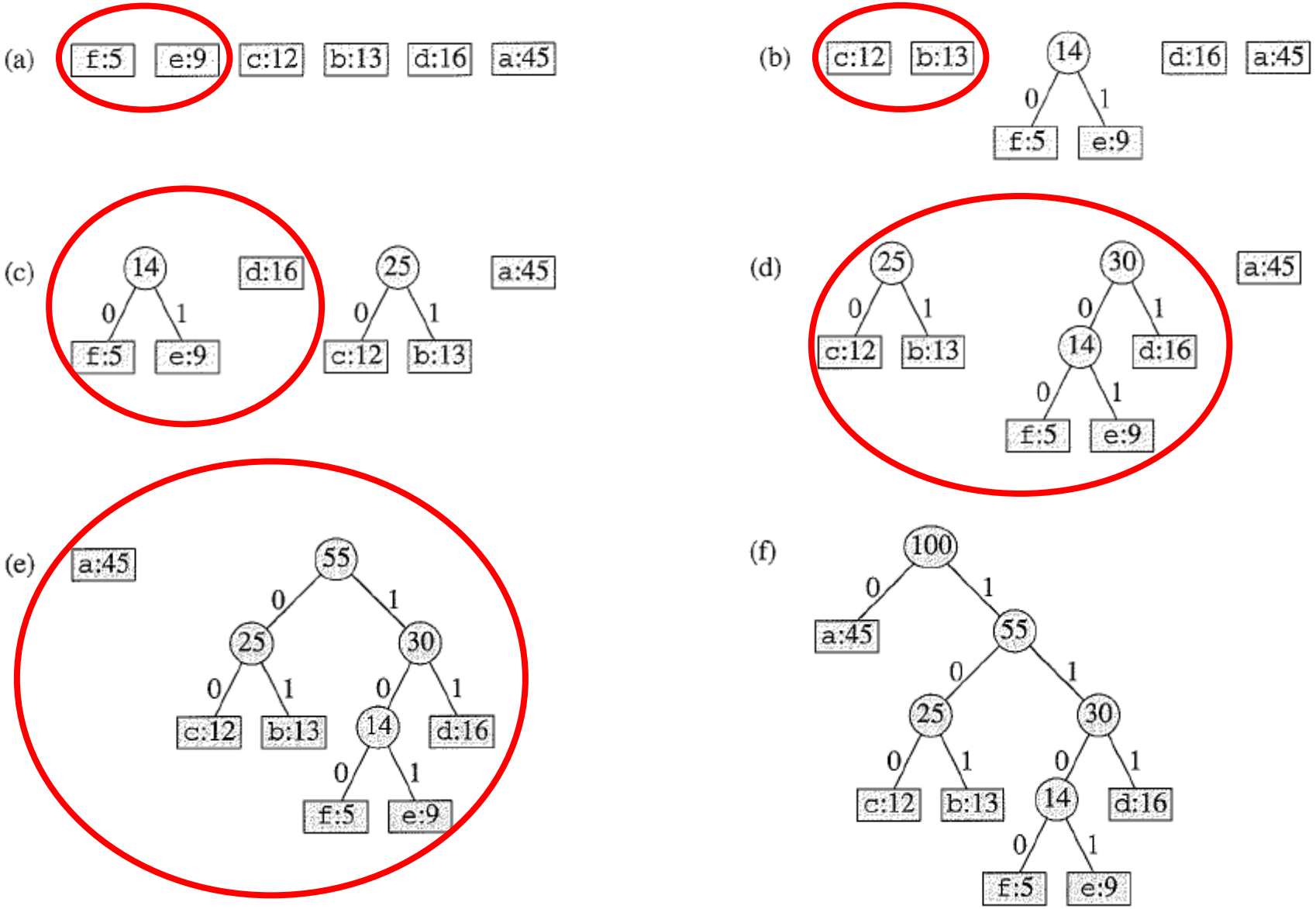
	a	b	c	d	e	f
Frequency (in thousands)	45	13	12	16	9	5
Fixed-length codeword	000	001	010	011	100	101
Variable-length codeword	0	101	100	111	1101	1100

Huffman Koder

```
HUFFMAN(C)
1  n = |C|
2  Q = C
3  for i = 1 to n − 1
4       allocate a new node z
5       z.left = x = EXTRACT-MIN(Q)
6       z.right = y = EXTRACT-MIN(Q)
7       z.freq = x.freq + y.freq
8       INSERT(Q, z)
9  return EXTRACT-MIN(Q)      // return the root of the tree
```

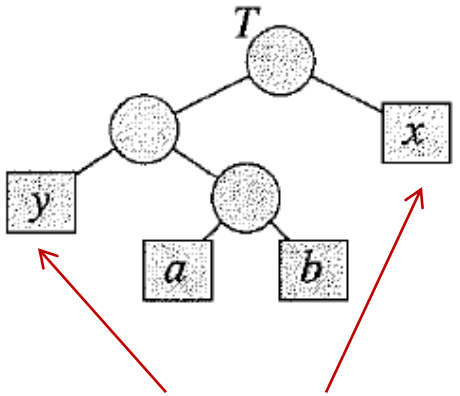
Tid $O(n \cdot \log n)$

Huffman Koder

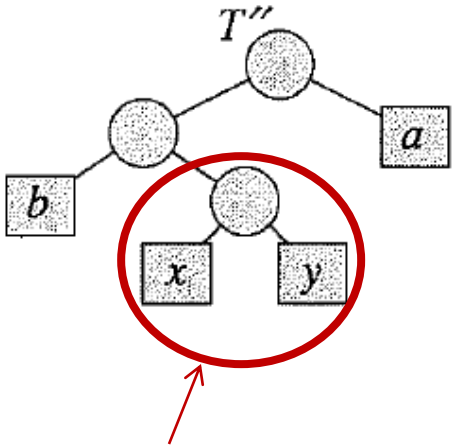


Korrektheden af Huffman Koder

Sætning Der findes altid en optimal prefix kode hvor de to mindst hyppige symboler (x og y) har samme kodelængde og kun adskiller sig i sidste bit



x og y mindst hyppige

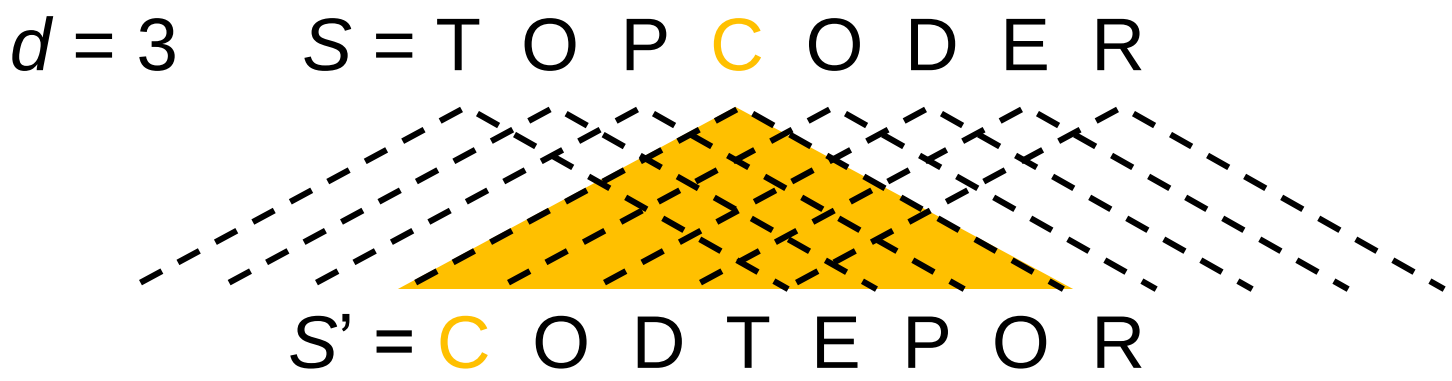


løs problemet hvor dette er et blad

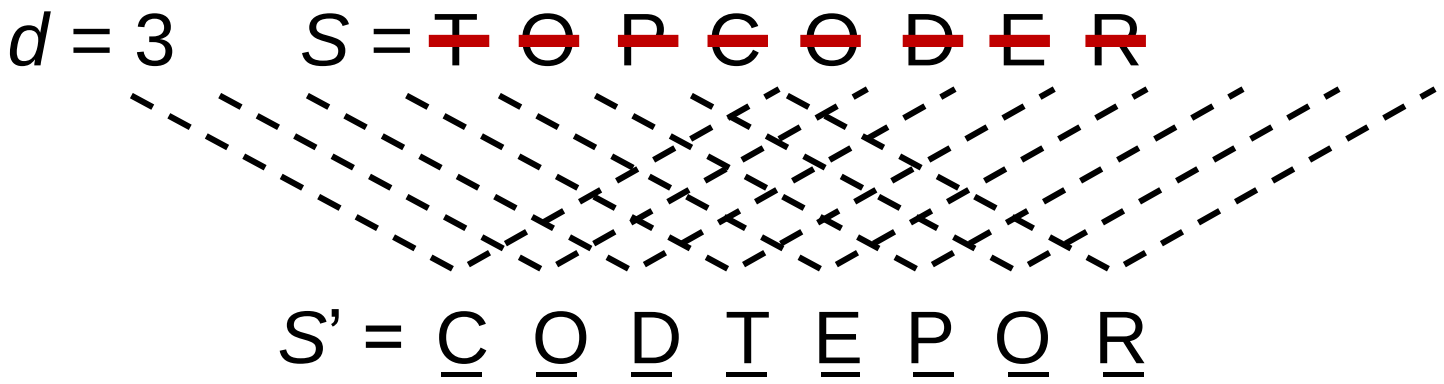
TopCoder Open 2014 – Round 1A (500 point)

Input: En streng *S*, afstand *d*

Ouput: En streng *S*' = leksikografisk mindste permutation af *S*, hvor ingen tegn flyttes mere end *d* positioner

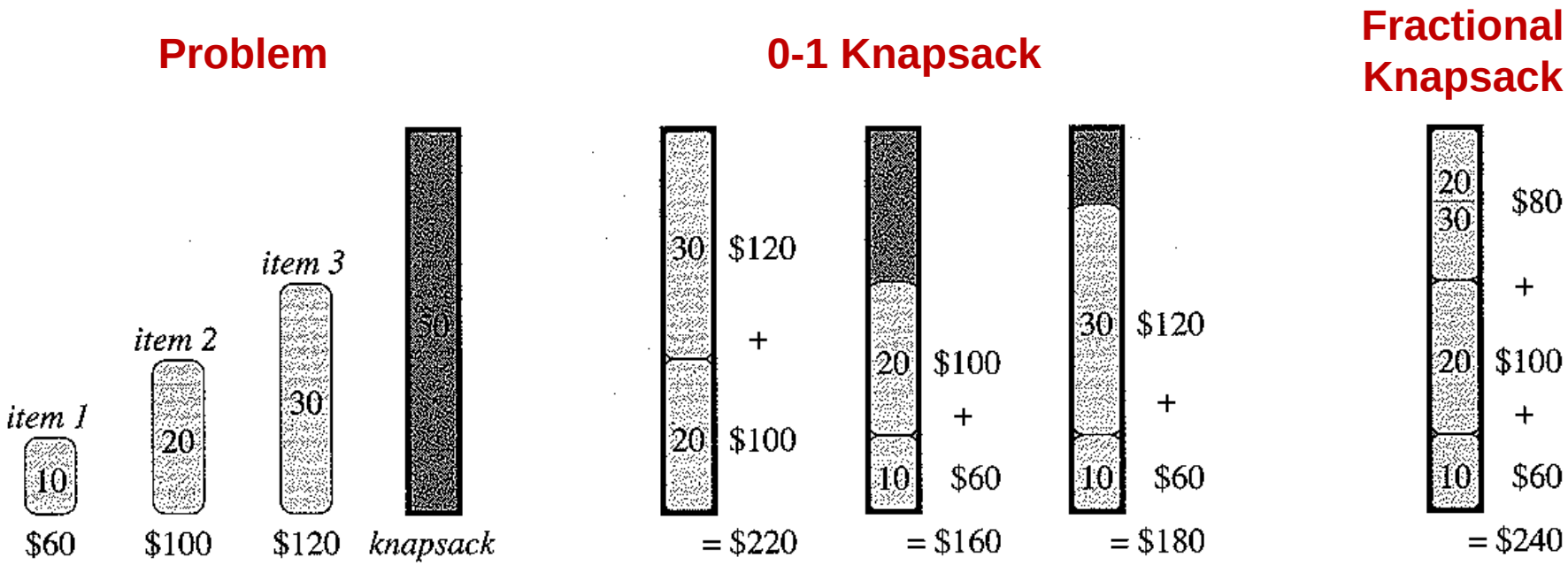


Grådig løsning



Algoritme Konstruer *S'* fra venstre mod højre
- tag det venstreste tegn i vinduet, hvis ubrugt
- ellers tag leksikografisk mindste tegn (venstreste hvis flere)

Dynamisk Programmering vs Grådig



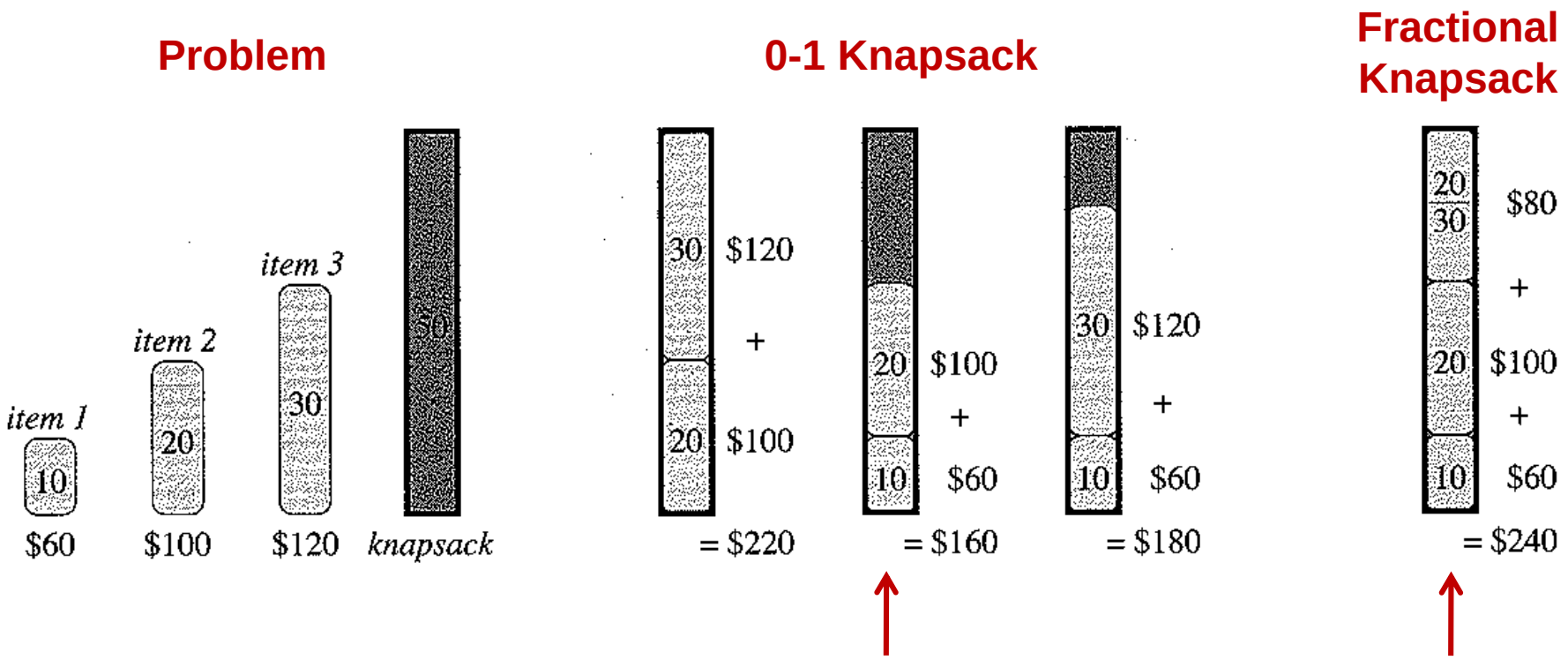
0-1 Knapsack

Maximal værdi hvis man har volume 11 ?

Volume	2	3	4	5
Værdi	3	5	4	6

a) 10
b) 11
c) 12
d) 13
e) 14
f) 15
g) Ved ikke

Dynamisk Programmering vs Grådig



c(k,v) = bedste værdi blandt k første objekter ved volume v tilbage

$$= \begin{cases} 0 & \text{hvis } k = 0 \\ c(k-1, v) & \text{hvis volume}_k > v \\ \max\{c(k-1, v), \text{værdi}_k + c(k-1, v - \text{volume}_k)\} & \text{hvis volume}_k \le v \end{cases}$$

Grådigt fyld i efter aftagende værdi/volume

Generelle Algoritmiske Design Tenikker

- **Del-og-Kombiner**
  - Disjunkte delproblemer
- **Dynamisk Programmering**
  - Overlappende delproblemer
  - Systematisk beregning af løsninger til alle mulige delproblemer (eller memoization)
- **Grådige Algoritmer**
  - Kun et delproblem