Grådige algoritmer

Grådige algoritmer

Et algoritme-konstruktionsprincip ("paradigme") for optimeringsproblemer.

Grådige algoritmer

Et algoritme-konstruktionsprincip ("paradigme") for optimeringsproblemer.

Dynamisk programmering:

1. Beregn først værdien af optimale løsninger for alle problemstørrelser bottom-up.
2. Opbyg så løsningen top-down ud fra denne information.

Grådige algoritmer

Et algoritme-konstruktionsprincip ("paradigme") for optimeringsproblemer.

Dynamisk programmering:

1. Beregn først værdien af optimale løsninger for alle problemstørrelser bottom-up.
2. Opbyg så løsningen top-down ud fra denne information.

Grådig algoritme:

1. Opbyg løsningen top-down ud fra hvad der lige nu ser ud som "bedste valg" (uden information om resten af løsningen).

Grådige algoritmer

Et algoritme-konstruktionsprincip ("paradigme") for optimeringsproblemer.

Dynamisk programmering:

1. Beregn først værdien af optimale løsninger for alle problemstørrelser bottom-up.
2. Opbyg så løsningen top-down ud fra denne information.

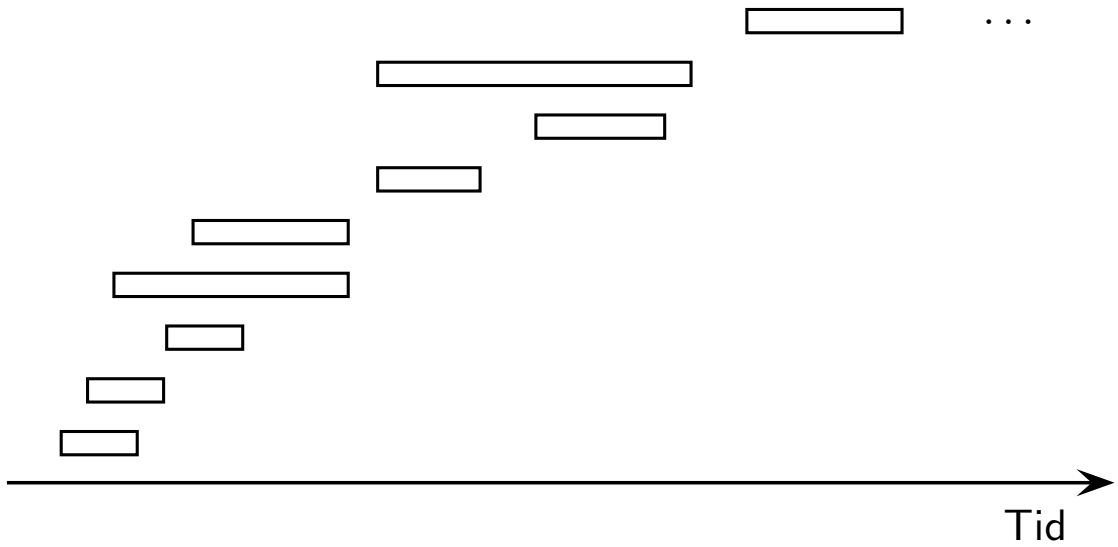
Grådig algoritme:

1. Opbyg løsningen top-down ud fra hvad der lige nu ser ud som "bedste valg" (uden information om resten af løsningen).

Dvs. simplere (og som regel hurtigere) end dynamisk programmering.

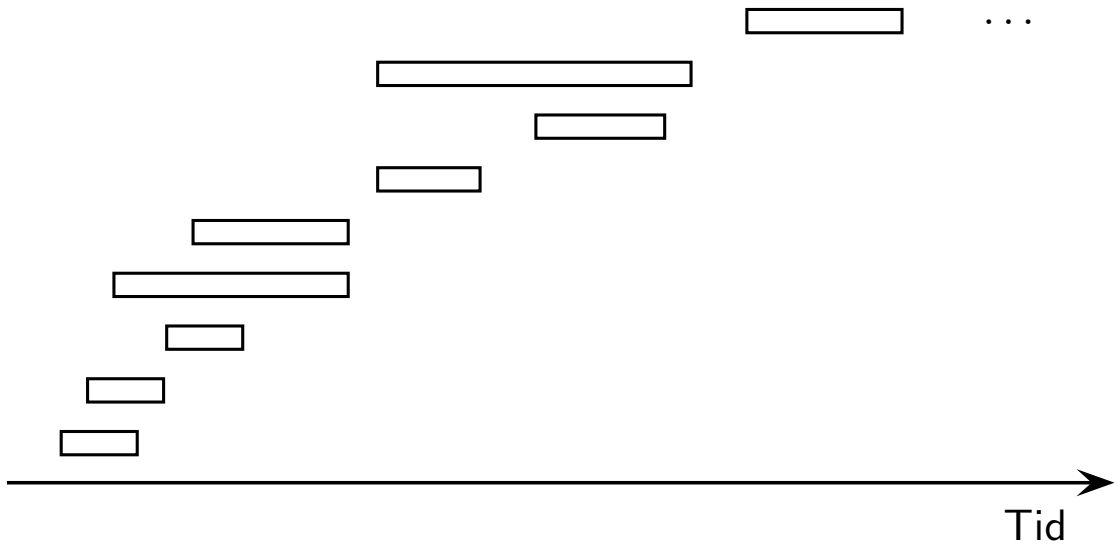
For at metoden virker, skal man kunne finde en "bedste valg"-regel (også kaldet "grådig" regel), som kan vises at give en optimal løsning.

Et simpelt skeduleringsproblem



Find maksimal mængde ikke-overlappende aktivititer.

Et simpelt skeduleringsproblem



Find maksimal mængde ikke-overlappende aktivititer.

"Grådig valg"-regel: blandt de ikke-valgte aktiviteter, som er ikke-overlappende med allerede valgte, vælg den med tidligst sluttid.

Et simpelt skeduleringsproblem

Argument for, at det virker:

- Enhver optimal løsnings første aktivitet kan, uden at inducere overlap, ændres til at have det grådige valgs første aktivitet. Der findes altså en optimal løsning, hvis første aktivitet er lig det grådige valg.
- Resten af denne løsning må selv være en optimal løsning blandt de resterende (ikke-overlappende) aktiviteter. Disse aktiviteter er netop hvad den grådige algoritme fortsætter med.

Et simpelt skeduleringsproblem

Argument for, at det virker:

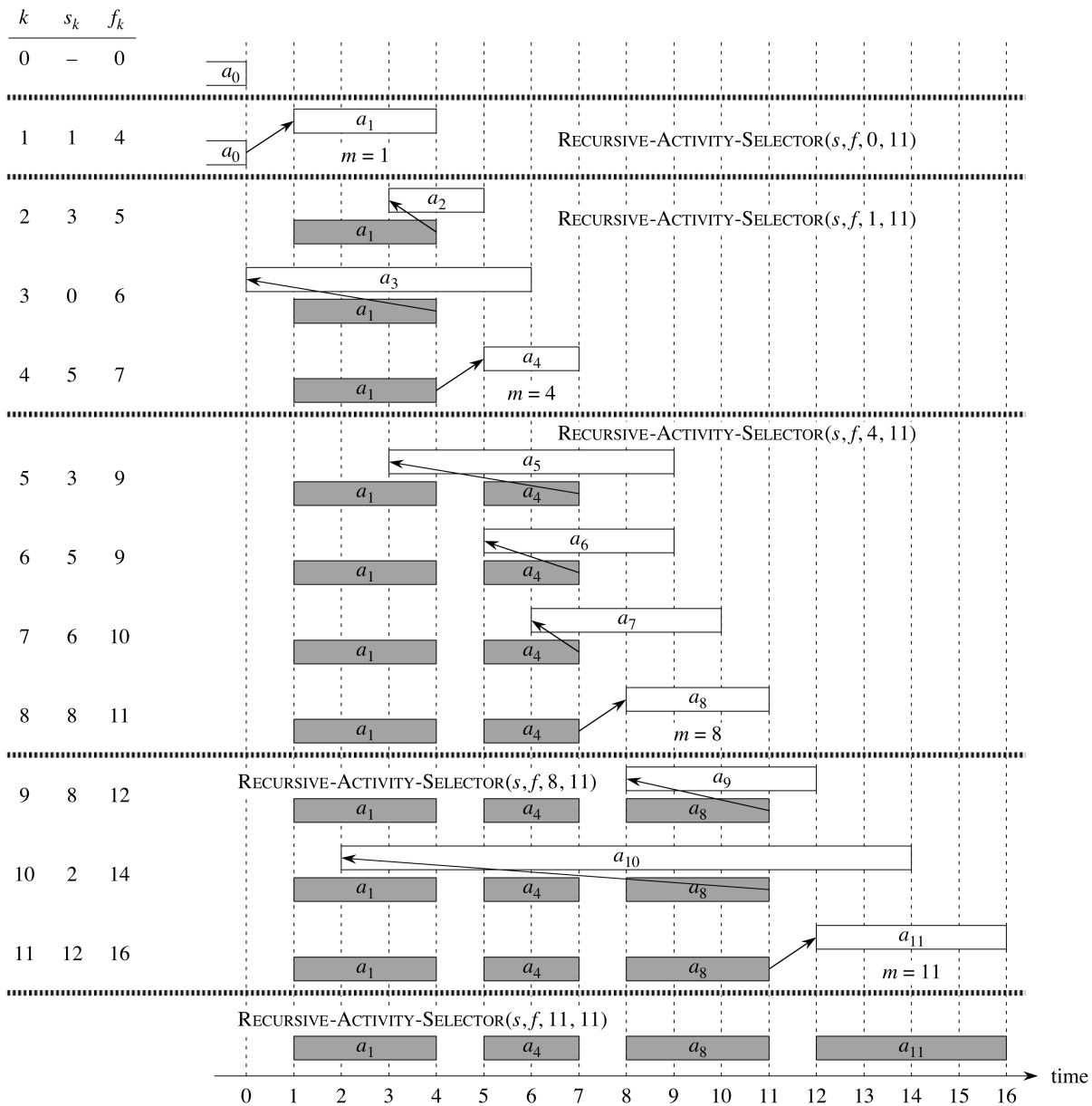
- Enhver optimal løsnings første aktivitet kan, uden at inducere overlap, ændres til at have det grådige valgs første aktivitet. Der findes altså en optimal løsning, hvis første aktivitet er lig det grådige valg.
- Resten af denne løsning må selv være en optimal løsning blandt de resterende (ikke-overlappende) aktiviteter. Disse aktiviteter er netop hvad den grådige algoritme fortsætter med.

Så flg. invariant vedligeholdes:

> Der findes en optimal løsning til det oprindelige problem, som består af de allerede valgte aktiviteter, samt en optimal løsning blandt de resterende (dvs. ikke-overlappende med allerede valgte) aktiviteter.

Algoritmen stopper, når antal resterende aktiviteter er nul. På dette tidspunkt fortæller invarianten, at de valgte aktiviteter er en optimal løsning til det oprindelige problem.

Et simpelt skeduleringsproblem



Rygsæksproblemet

Rygsæk som kan bære W kg.

Ting med værdi og vægt.

Ting nr. i	1	2	3	4	5	6	7
Vægt w_i	4	6	2	15	7	4	5
Værdi v_i	45	32	12	50	23	9	15

Rygsæksproblemet

Rygsæk som kan bære W kg.

Ting med værdi og vægt.

Ting nr. i	1	2	3	4	5	6	7
Vægt w_i	4	6	2	15	7	4	5
Værdi v_i	45	32	12	50	23	9	15

Mål: tag mest mulig værdi med uden at overskride vægtgrænsen.

Rygsæksproblemet

"Fractional" version af problemet (dele af ting kan medtages i rygsækken) kan løses med en grådig algoritme: vælg tingene efter aftagende "nytte" = værdi/vægt.

Rygsæksproblemet

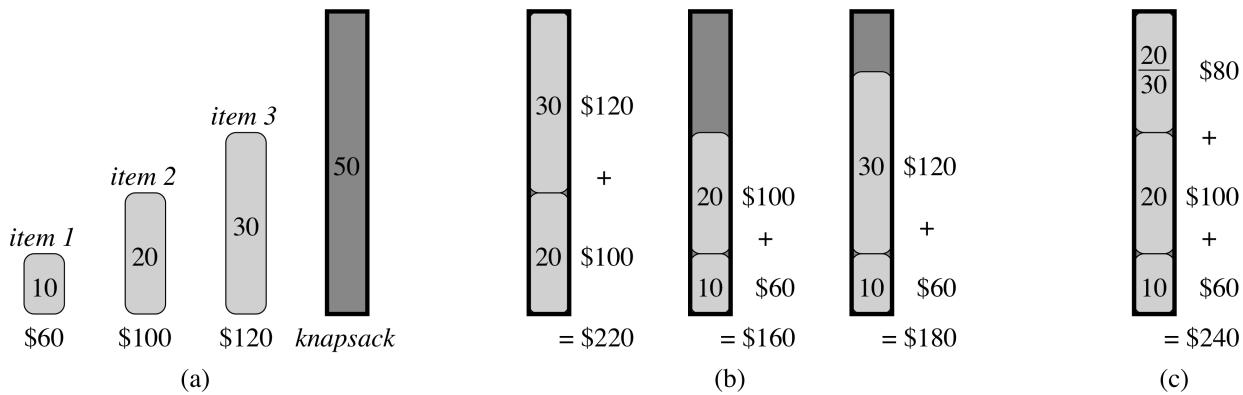
"Fractional" version af problemet (dele af ting kan medtages i rygsækken) kan løses med en grådig algoritme: vælg tingene efter aftagende "nytte" = værdi/vægt.

Denne grådige algoritme virker IKKE for 0-1 versionen af problemet (kun hele ting kan medtages):

Rygsæksproblemet

"Fractional" version af problemet (dele af ting kan medtages i rygsækken) kan løses med en grådig algoritme: vælg tingene efter aftagende "nytte" = værdi/vægt.

Denne grådige algoritme virker IKKE for 0-1 versionen af problemet (kun hele ting kan medtages):



Rygsæksproblemet

0-1 version: Kan løses med dynamisk programmering (når W og w_i'er er heltal):

Rygsæksproblemet

0-1 version: Kan løses med dynamisk programmering (når W og w_i'er er heltal):

Om en optimal løsning for en rygsæk med kapacitet w, hvis indhold skal vælges blandt ting nr. 1, 2,. . . ,i, gælder at den:

- enten medtager ting nr. n og resten af løsningen er optimal for ting nr. 1 til $(n-1)$ med rygsæk kapacitet $W - w_n$
- eller medtager ikke ting nr. n, og er så selv optimal for ting nr. 1 til $(n-1)$ med rygsæk kapacitet W

Rygsæksproblemet

0-1 version: Kan løses med dynamisk programmering (når W og w_i'er er heltal):

Om en optimal løsning for en rygsæk med kapacitet w, hvis indhold skal vælges blandt ting nr. 1, 2,. . . ,i, gælder at den:

- enten medtager ting nr. n og resten af løsningen er optimal for ting nr. 1 til $(n-1)$ med rygsæk kapacitet $W - w_n$
- eller medtager ikke ting nr. n, og er så selv optimal for ting nr. 1 til $(n-1)$ med rygsæk kapacitet W

Lad $\mathrm{k}(i, w)$ betegne værdien af en optimal løsning for en rygsæk med kapacitet w hvis indhold skal vælges blandt ting nr. 1, 2,. . . ,i.

$$\mathrm{k}(i,w) = \begin{cases} 0 & \text{if } i = 0 \text{ or } w \leq 0 \\ \max(\mathrm{k}(i-1, w-w_i) + v_i, \mathrm{k}(i-1, w)) & \text{if } i, w > 0 \end{cases}$$

Rygsæksproblemet

0-1 version: Kan løses med dynamisk programmering (når W og w_i'er er heltal):

Om en optimal løsning for en rygsæk med kapacitet w, hvis indhold skal vælges blandt ting nr. 1, 2,. . . ,i, gælder at den:

- enten medtager ting nr. n og resten af løsningen er optimal for ting nr. 1 til $(n-1)$ med rygsæk kapacitet $W - w_n$
- eller medtager ikke ting nr. n, og er så selv optimal for ting nr. 1 til $(n-1)$ med rygsæk kapacitet W

Lad $\mathrm{k}(i, w)$ betegne værdien af en optimal løsning for en rygsæk med kapacitet w hvis indhold skal vælges blandt ting nr. 1, 2,. . . ,i.

$$\mathrm{k}(i,w) = \begin{cases} 0 & \text{if } i = 0 \text{ or } w \leq 0 \\ \max(\mathrm{k}(i-1, w-w_i) + v_i, \mathrm{k}(i-1, w)) & \text{if } i, w > 0 \end{cases}$$

Fyld $n \times W$-tabel for $\mathrm{k}(i, w)$ $(0 \leq i \leq n,\ 0 \leq w \leq W)$ på struktureret måde (målet er $\mathrm{k}(n, W)$). Find derefter en løsning top-down ved at følge de optimale valg (lavet i anden linien af rekursionsformlen ovenfor).

Huffman-koder

Kodning af sekvens af tegn (fil) via binære koder (for at gemme på disk, sende over netværk,...).

Ønsker kortest mulig fil (kompression).

Huffman-koder

Kodning af sekvens af tegn (fil) via binære koder (for at gemme på disk, sende over netværk,...).

Ønsker kortest mulig fil (kompression).

Eksempel:

	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

Kodning af sekvens af tegn (fil) via binære koder (for at gemme på disk, sende over netværk,...).

Ønsker kortest mulig fil (kompression).

Eksempel:

	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

Fixed-length version:

$$3 \cdot (45.000 + 13.000 + \cdots + 5.000) = 300.000 \text{ bits}$$

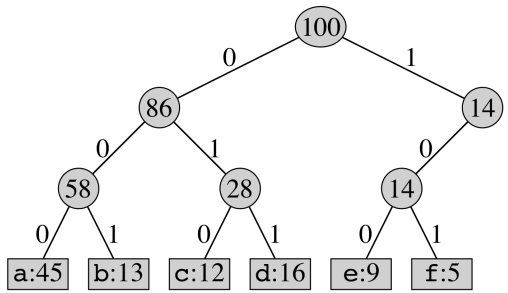
Variable-length version:

$$1 \cdot 45.000 + 3 \cdot 13.000 + \cdots + 4 \cdot 5.000 = 224.000 \text{ bits}$$

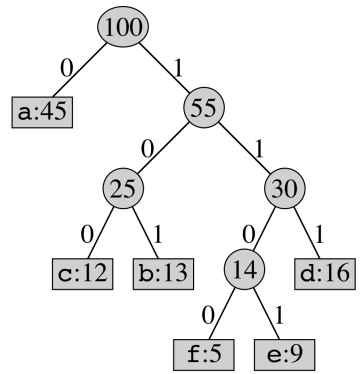
Prefix-kode = træer

Prefix-fri kode: ingen kode for et tegn er starten (prefix) af koden for et andet tegn ($\Rightarrow$ dekodning utvetydig). Så tegn svarer til knuder med nul børn (blade).

	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



(a)

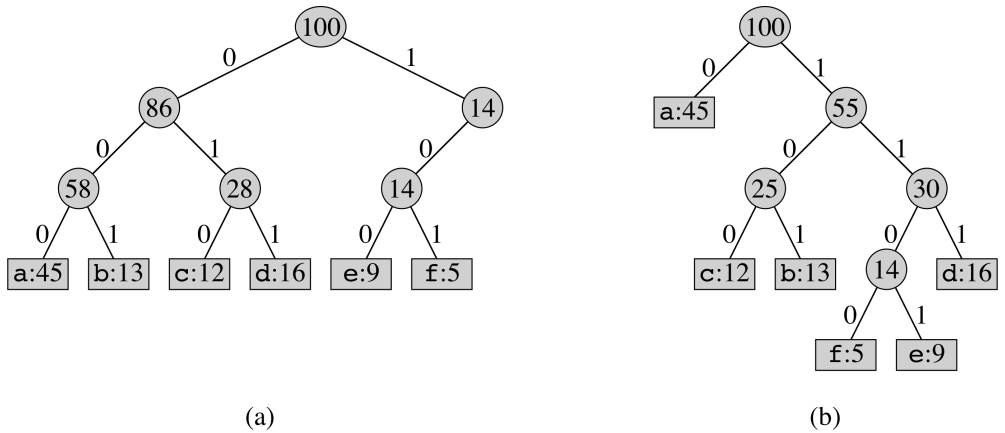


(b)

Prefix-kode = træer

Prefix-fri kode: ingen kode for et tegn er starten (prefix) af koden for et andet tegn ($\Rightarrow$ dekodning utvetydig). Så tegn svarer til knuder med nul børn (blade).

	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



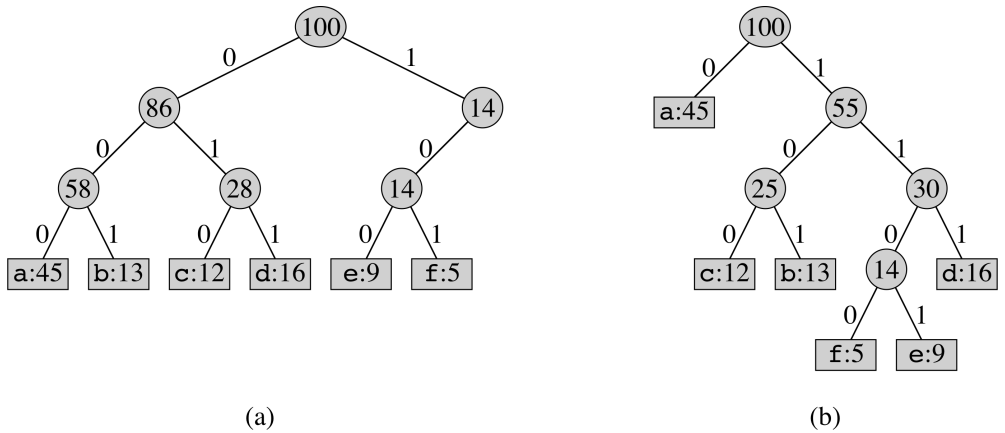
(a) (b)

For en givet fil (tegn og deres frekvenser), find bedste variable-length prefix-kode. Dvs. for Cost(tree) = |kodet fil|, find træ med lavest cost.

Prefix-kode = træer

Prefix-fri kode: ingen kode for et tegn er starten (prefix) af koden for et andet tegn ($\Rightarrow$ dekodning utvetydig). Så tegn svarer til knuder med nul børn (blade).

	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



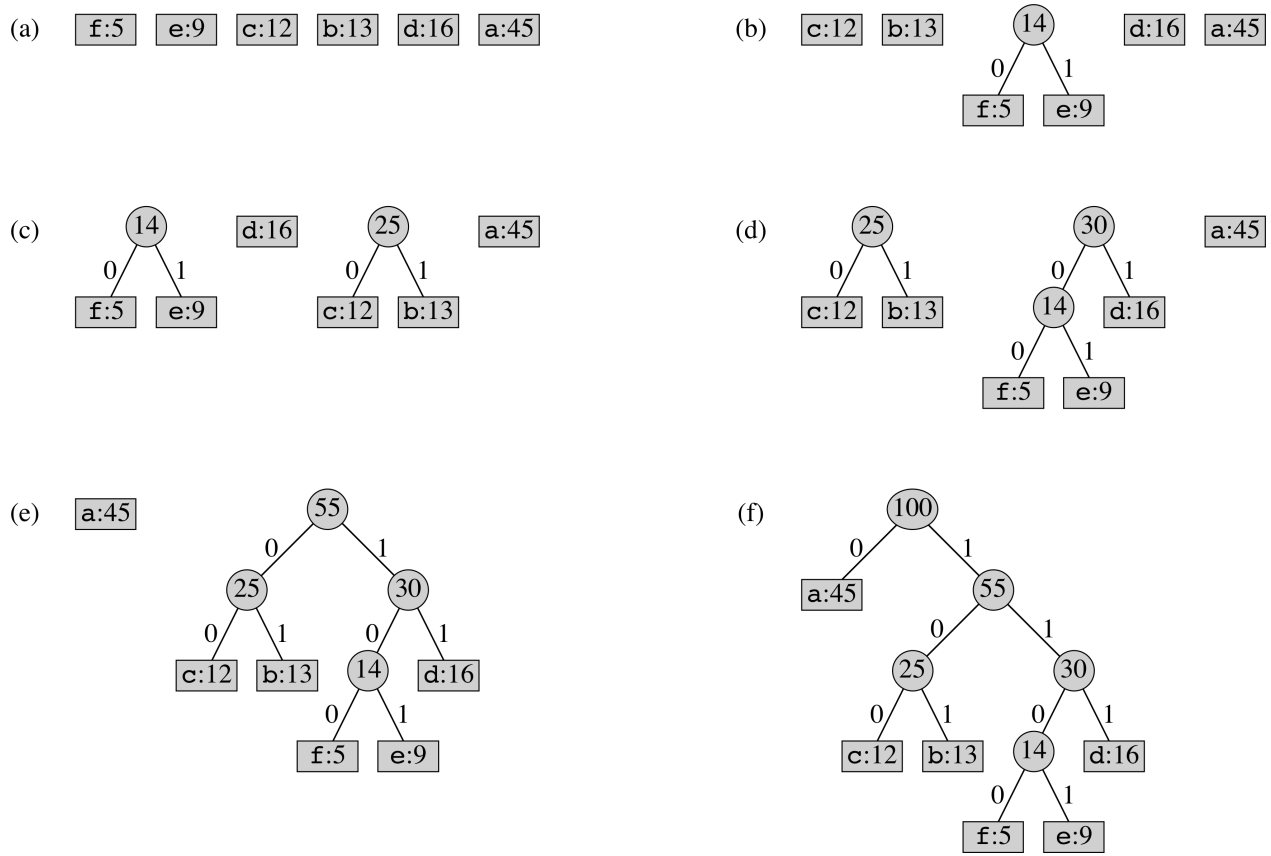
(a) (b)

For en givet fil (tegn og deres frekvenser), find bedste variable-length prefix-kode. Dvs. for Cost(tree) = |kodet fil|, find træ med lavest cost.

Optimale træer kan ikke have knuder med kun eet barn (alle tegn i undertræet for en sådan knude kan forkortes med een bit, jvf. (a) ovenfor). Så kun knuder med to eller nul børn findes.

Huffmans algoritme

Byg op nedefra (fra mindste til største frekvenser) ved hele tiden at lave flg. "grådige valg": slå de to deltræer med de to mindste samlede frekvenser sammen:



Køretid

Givet en en tabel med n tegn og deres frekvenser laver Huffmans algoritme

- $n - 1$ iterationer.

(Der er n træer til start, eet træ til slut, og hver iteration mindsker antallet med præcis een.)

Køretid

Givet en en tabel med n tegn og deres frekvenser laver Huffmans algoritme

- $n - 1$ iterationer.

(Der er n træer til start, eet træ til slut, og hver iteration mindsker antallet med præcis een.)

Ved at bruge en (min-)prioritetskø, f.eks. implementeret ved en heap, kan hver iteration udføres med:

- to ExtractMin-operationer
- een Insert-operation
- $O(1)$ andet arbejde.

Køretid

Givet en en tabel med n tegn og deres frekvenser laver Huffmans algoritme

- $n - 1$ iterationer.

(Der er n træer til start, eet træ til slut, og hver iteration mindsker antallet med præcis een.)

Ved at bruge en (min-)prioritetskø, f.eks. implementeret ved en heap, kan hver iteration udføres med:

- to ExtractMin-operationer
- een Insert-operation
- $O(1)$ andet arbejde.

Hver prioritetskø-operation tager hver $O(\log n)$ tid.

Køretid

Givet en en tabel med n tegn og deres frekvenser laver Huffmans algoritme

- $n - 1$ iterationer.

(Der er n træer til start, eet træ til slut, og hver iteration mindsker antallet med præcis een.)

Ved at bruge en (min-)prioritetskø, f.eks. implementeret ved en heap, kan hver iteration udføres med:

- to ExtractMin-operationer
- een Insert-operation
- $O(1)$ andet arbejde.

Hver prioritetskø-operation tager hver $O(\log n)$ tid.

Så samlet køretid for de n iterationer er $O(n \log n)$.

Korrekthed

Huffman algoritme slår i første skridt de "to mindste" tegn sammen.

Mere præcist: For en eller anden nummerering

$$c_1, c_2, c_3, \ldots, c_n$$

af tegnene i inputalfabetet, for hvilken der gælder at de tilsvarende frekvenser er ikke-faldende

$$f_1 \leq f_2 \leq f_3 \leq \cdots \leq f_n,$$

slår Huffmans algoritme tegn c_1 og c_2 sammen.

Denne formulering dækker alle mulige valg af de "to mindste" tegn, også for input såsom disse tre:

Tegn	a	b	c	d	e	f	g
Frekvens	7	3	6	3	3	7	5
Frekvens	7	3	6	5	7	5	5
Frekvens	3	3	3	3	3	3	3

Korrekthed

Lemma: For ethvert input ($n \geq 2$)

Tegn	c_1	c_2	c_3	$\ldots$	c_n
Frekvens	f_1	f_2	f_3	$\ldots$	f_n

med

$$f_1 \leq f_2 \leq f_3 \leq \cdots \leq f_n,$$

findes et optimalt træ hvor bladene for tegnene $x = c_1$ og $y = c_2$ er søskende.

Korrekthed

Lemma: For ethvert input ($n \geq 2$)

Tegn	c_1	c_2	c_3	$\ldots$	c_n
Frekvens	f_1	f_2	f_3	$\ldots$	f_n

med

$$f_1 \leq f_2 \leq f_3 \leq \cdots \leq f_n,$$

findes et optimalt træ hvor bladene for tegnene $x = c_1$ og $y = c_2$ er søskende.

Bevis: Lad T være et optimalt træ. Se på et blad af størst dybde i T. Da $n \geq 2$, har bladet en forælder. Denne forælder har mere end nul undertræer, altså har den to (i optimale træer har ingen knuder eet undertræ, se tidligere). Dens andet undetræ må også være et blad, ellers er det første blad ikke et dybeste blad.

Korrekthed

Lemma: For ethvert input ($n \geq 2$)

Tegn	c_1	c_2	c_3	$\ldots$	c_n
Frekvens	f_1	f_2	f_3	$\ldots$	f_n

med

$$f_1 \leq f_2 \leq f_3 \leq \cdots \leq f_n,$$

findes et optimalt træ hvor bladene for tegnene $x = c_1$ og $y = c_2$ er søskende.

Bevis: Lad T være et optimalt træ. Se på et blad af størst dybde i T. Da $n \geq 2$, har bladet en forælder. Denne forælder har mere end nul undertræer, altså har den to (i optimale træer har ingen knuder eet undertræ, se tidligere). Dens andet undetræ må også være et blad, ellers er det første blad ikke et dybeste blad.

Altså findes to blade som er søskende og begge er af størst dybde. Lad deres tegn være $a = c_i$ og $b = c_j$, for $i < j$.

Korrekthed

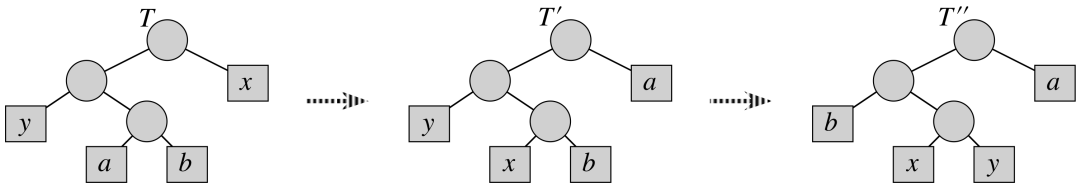
Mulige situationer:

c_1	c_2	...	Handling
x	y		
a	b		Ingen
a		b	Byt y og b
	a	b	Byt x og b
		a b	Byt x og a, samt y og b

Korrekthed

Mulige situationer:

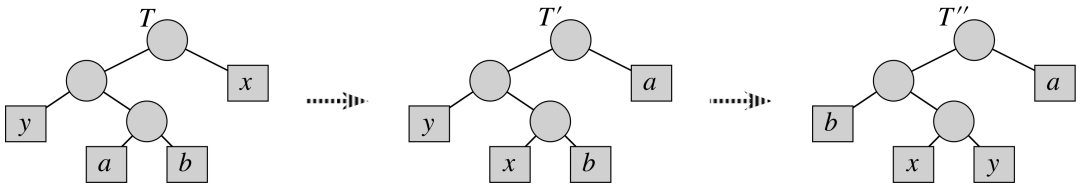
c_1	c_2	...	Handling
x	y		
a	b		Ingen
a		b	Byt y og b
	a	b	Byt x og b
		a b	Byt x og a, samt y og b



Korrekthed

Mulige situationer:

c_1	c_2	...	Handling
x	y		
a	b		Ingen
a		b	Byt y og b
	a	b	Byt x og b
		a b	Byt x og a, samt y og b

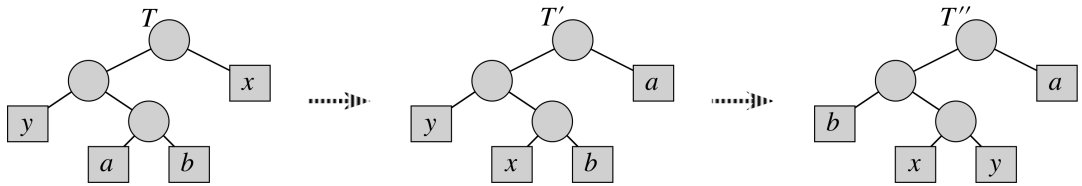


Lad d være a's blads dybde minus x's blads dybde. Da er forandringen i filens længde ved byt af $x = c_1$ og $a = c_i$ lig $d \cdot f_1 - d \cdot f_i = d \cdot (f_1 - f_i)$.

Korrekthed

Mulige situationer:

c_1	c_2	...	Handling
x	y		
a	b		Ingen
a		b	Byt y og b
	a	b	Byt x og b
		a b	Byt x og a, samt y og b



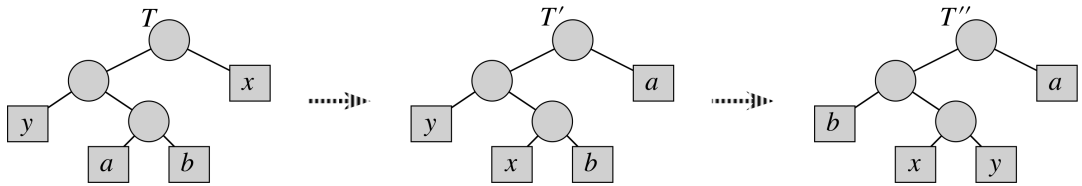
Lad d være a's blads dybde minus x's blads dybde. Da er forandringen i filens længde ved byt af $x = c_1$ og $a = c_i$ lig $d \cdot f_1 - d \cdot f_i = d \cdot (f_1 - f_i)$.

Eftersom $d \geq 0$ (da a's blad var af størst dybde) og $(f_1 - f_i) \leq 0$ (da $i \geq 1$, se tabel ovenfor), kan filens længde ikke blive større. Dvs. træet kan ikke blive dårlige ved byt af x og a.

Korrekthed

Mulige situationer:

c_1	c_2	...	Handling
x	y		
a	b		Ingen
a		b	Byt y og b
	a	b	Byt x og b
		a b	Byt x og a, samt y og b



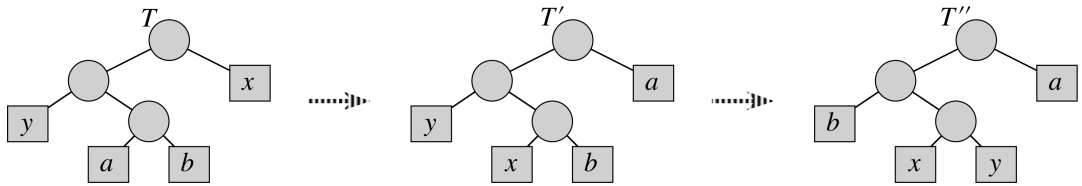
Lad d være a's blads dybde minus x's blads dybde. Da er forandringen i filens længde ved byt af $x = c_1$ og $a = c_i$ lig $d \cdot f_1 - d \cdot f_i = d \cdot (f_1 - f_i)$.

Eftersom $d \geq 0$ (da a's blad var af størst dybde) og $(f_1 - f_i) \leq 0$ (da $i \geq 1$, se tabel ovenfor), kan filens længde ikke blive større. Dvs. træet kan ikke blive dårlige ved byt af x og a. Tilsvarende kan vises for et byt af x og b, og for et byt af y og b.

Korrekthed

Mulige situationer:

c_1	c_2	...	Handling
x	y		
a	b		Ingen
a		b	Byt y og b
	a	b	Byt x og b
		a b	Byt x og a, samt y og b



Lad d være a's blads dybde minus x's blads dybde. Da er forandringen i filens længde ved byt af $x = c_1$ og $a = c_i$ lig $d \cdot f_1 - d \cdot f_i = d \cdot (f_1 - f_i)$.

Eftersom $d \geq 0$ (da a's blad var af størst dybde) og $(f_1 - f_i) \leq 0$ (da $i \geq 1$, se tabel ovenfor), kan filens længde ikke blive større. Dvs. træet kan ikke blive dårlige ved byt af x og a. Tilsvarende kan vises for et byt af x og b, og for et byt af y og b.

Da træet var optimalt før byt, er det også efter. □

Korrekthed

Lemma: Huffmans algoritme returnerer et optimalt træ.

Korrekthed

Lemma: Huffmans algoritme returnerer et optimalt træ.

Bevis: Vi vil vise via induktion at det gælder for input med n tegn, for alle n.

Korrekthed

Lemma: Huffmans algoritme returnerer et optimalt træ.

Bevis: Vi vil vise via induktion at det gælder for input med n tegn, for alle n.

Basis $n = 1$: Her returnerer Huffmans et træ som er et blad. Dette er eneste mulige træ (da knuder har enten to eller nul børn), og derfor optimalt.

Korrekthed

Lemma: Huffmans algoritme returnerer et optimalt træ.

Bevis: Vi vil vise via induktion at det gælder for input med n tegn, for alle n.

Basis $n = 1$: Her returnerer Huffmans et træ som er et blad. Dette er eneste mulige træ (da knuder har enten to eller nul børn), og derfor optimalt.

Induktionsskridtet $n \geq 1$: Lad input I være

Tegn	c_1	c_2	c_3	$\ldots$	c_n
Frekvens	f_1	f_2	f_3	$\ldots$	f_n

med $f_1 \leq f_2 \leq f_3 \leq \cdots \leq f_n$,

og se på input I'

Tegn	z	c_3	$\ldots$	c_n
Frekvens	$f_1 + f_2$	f_3	$\ldots$	f_n

Korrekthed

- Lad T være Huffmans output på input I.
- Lad T' være Huffmans output på input I' (optimalt træ pr. induktionsantagelse).
- Lad T'' være et optimalt træ på input I hvor bladene for tegnene $x = c_1$ og $y = c_2$ er søskende (eksisterer iflg. lemma).
- Lad T''' være T'' hvor bladene for tegnene x og y samt deres forælder er erstattet af bladet z (med frekvens $f_1 + f_2$).

Korrekthed

- Lad T være Huffmans output på input I.
- Lad T' være Huffmans output på input I' (optimalt træ pr. induktionsantagelse).
- Lad T'' være et optimalt træ på input I hvor bladene for tegnene $x = c_1$ og $y = c_2$ er søskende (eksisterer iflg. lemma).
- Lad T''' være T'' hvor bladene for tegnene x og y samt deres forælder er erstattet af bladet z (med frekvens $f_1 + f_2$).

For et træ τ for input I', lad Expand(τ) være τ med bladet med tegnet z erstattet af et træ med en rod og to blade med tegnene $x = c_1$ og $y = c_2$.

Korrekthed

- Lad T være Huffmans output på input I.
- Lad T' være Huffmans output på input I' (optimalt træ pr. induktionsantagelse).
- Lad T'' være et optimalt træ på input I hvor bladene for tegnene $x = c_1$ og $y = c_2$ er søskende (eksisterer iflg. lemma).
- Lad T''' være T'' hvor bladene for tegnene x og y samt deres forælder er erstattet af bladet z (med frekvens $f_1 + f_2$).

For et træ τ for input I', lad Expand(τ) være τ med bladet med tegnet z erstattet af et træ med en rod og to blade med tegnene $x = c_1$ og $y = c_2$.

- Cost(Expand(τ)) = Cost(τ) + f_1 + f_2.
- Expand(T') = T (pga. Huffman-algoritmens virkemåde).
- Expand(T''') = T'' (pga. definition af T''').

Korrekthed

- Lad T være Huffmans output på input I.
- Lad T' være Huffmans output på input I' (optimalt træ pr. induktionsantagelse).
- Lad T'' være et optimalt træ på input I hvor bladene for tegnene $x = c_1$ og $y = c_2$ er søskende (eksisterer iflg. lemma).
- Lad T''' være T'' hvor bladene for tegnene x og y samt deres forælder er erstattet af bladet z (med frekvens $f_1 + f_2$).

For et træ τ for input I', lad Expand(τ) være τ med bladet med tegnet z erstattet af et træ med en rod og to blade med tegnene $x = c_1$ og $y = c_2$.

- Cost(Expand(τ)) = Cost(τ) + f_1 + f_2.
- Expand(T') = T (pga. Huffman-algoritmens virkemåde).
- Expand(T''') = T'' (pga. definition af T''').

Cost(T'') $\leq$ Cost(T) = Cost(Expand(T')) = Cost(T') + f_1 + $f_2 \leq$
Cost(T''') + f_1 + f_2 = Cost(Expand(T''')) = Cost(T''). $\square$