Grådige algoritmer

Grådige algoritmer

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

Grådige algoritmer

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

Ideen er simpel:

- Opbyg løsningen bid for bid ved hele tiden af vælge hvad der lige nu ser ud som "bedste valg" (uden at tænke på resten af løsningen).

Dvs. man håber på at lokal optimering giver global optimering.

Grådige algoritmer

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

Ideen er simpel:

- Opbyg løsningen bid for bid ved hele tiden af vælge hvad der lige nu ser ud som "bedste valg" (uden at tænke på resten af løsningen).

Dvs. man håber på at lokal optimering giver global optimering.

Mere præcist kræver metoden en definition af "bedste valg" (også kaldet "grådig valg"), samt et bevis for gentagen brug af dette ender med en optimal løsning.

Grådige algoritmer

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

Ideen er simpel:

- Opbyg løsningen bid for bid ved hele tiden af vælge hvad der lige nu ser ud som "bedste valg" (uden at tænke på resten af løsningen).

Dvs. man håber på at lokal optimering giver global optimering.

Mere præcist kræver metoden en definition af "bedste valg" (også kaldet "grådig valg"), samt et bevis for gentagen brug af dette ender med en optimal løsning.

Mange grådige algoritmer kan forklares som specialtilfælde af begrebet "Matroide" (afsnit 16.4, ikke pensum), men vi vil nøjes med at tænke på grådige algoritmer som et løst princip, hvor "bedste valg" og korrekthedsbevis skal opfindes individuelt fra problem til problem.

Eksempel: et simpelt skeduleringsproblem

Input: en samling aktiviteter, hver med en starttid og sluttid.

Output: en størst mulig mængde af ikke-overlappende aktivititer.

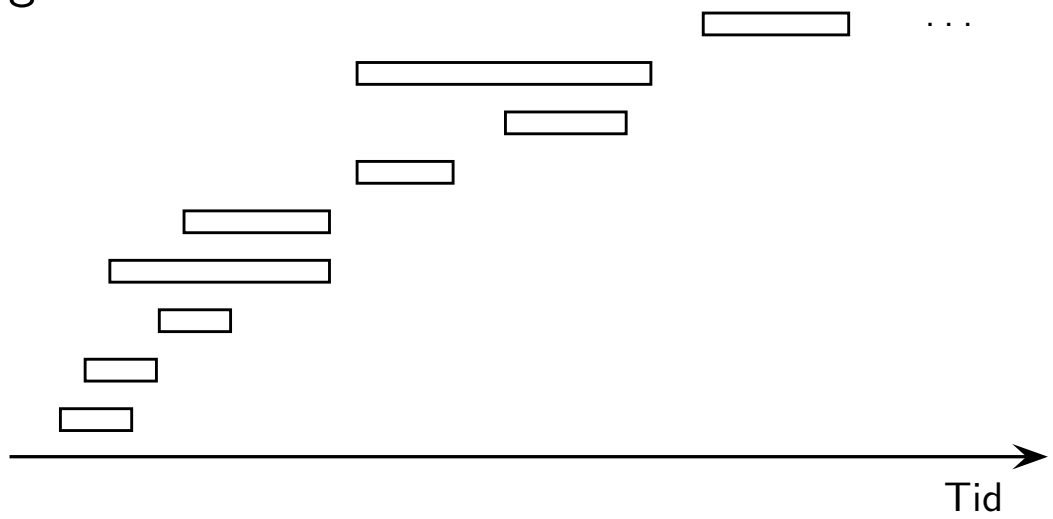
Eksempel: et simpelt skeduleringsproblem

Input: en samling aktiviteter, hver med en starttid og sluttid.

Output: en størst mulig mængde af ikke-overlappende aktivititer.

Grådigt forslag:

Sorter aktiviteter efter stigende sluttid
For hver aktivitet *a* taget i den rækkefølge:
    If *a* overlapper allerede valgte aktiviteter:
        Skip *a*
    Else
        Vælg *a*



Analyse

Bevis for korrekthed:

Flg. invariant vedligeholdes:

> Lad s være sluttiden for aktiviteten testet i sidst udførte iteration af **for**-løkken (og lad $s = -\infty$ før første iteration). Der findes en optimal løsning, hvis aktiviteter med sluttid $\leq s$ netop er de af algoritmen indtil nu valgte aktiviteter.

Analyse

Bevis for korrekthed:

Flg. invariant vedligeholdes:

> Lad s være sluttiden for aktiviteten testet i sidst udførte iteration af **for**-løkken (og lad $s = -\infty$ før første iteration). Der findes en optimal løsning, hvis aktiviteter med sluttid $\leq s$ netop er de af algoritmen indtil nu valgte aktiviteter.

Bevis (induktion):

Basis: Klart før første iteration af **for**-løkken (for enhver optimal løsning er aktiviteterne med sluttid $\leq -\infty$ de samme som de af algoritmen indtil nu valgte aktiviteter, nemlig ingen).

Analyse

Bevis (induktion):

Skridt: Hvis udsagnet gælder før en iteration, gælder det også efter: Lad X være den optimale løsning fra induktionsudsagnet før iterationen.

If-case: Her overlapper den næste aktivitet a de allerede passerede aktiviteter fra X, så dennne aktivitet kan ikke være en del af X. Da algoritmen ikke vælger a, kan X (gen)bruges i induktionsudsagnet efter iterationen.

Else-case: Lad a' være den aktivitet i resten af X, som har mindst sluttid. Det er nemt at se, at man kan udskifte a' med den af algoritmen valgte a uden at få overlap med andre i X. Da dette ikke ændrer antal aktiviteter i X, er den stadig optimal, og kan bruges i induktionsudsagnet efter iterationen.

Analyse

Bevis (induktion):

Skridt: Hvis udsagnet gælder før en iteration, gælder det også efter: Lad X være den optimale løsning fra induktionsudsagnet før iterationen.

If-case: Her overlapper den næste aktivitet a de allerede passerede aktiviteter fra X, så dennne aktivitet kan ikke være en del af X. Da algoritmen ikke vælger a, kan X (gen)bruges i induktionsudsagnet efter iterationen.

Else-case: Lad a' være den aktivitet i resten af X, som har mindst sluttid. Det er nemt at se, at man kan udskifte a' med den af algoritmen valgte a uden at få overlap med andre i X. Da dette ikke ændrer antal aktiviteter i X, er den stadig optimal, og kan bruges i induktionsudsagnet efter iterationen.

Køretid:

Analyse

Bevis (induktion):

Skridt: Hvis udsagnet gælder før en iteration, gælder det også efter: Lad X være den optimale løsning fra induktionsudsagnet før iterationen.

If-case: Her overlapper den næste aktivitet a de allerede passerede aktiviteter fra X, så dennne aktivitet kan ikke være en del af X. Da algoritmen ikke vælger a, kan X (gen)bruges i induktionsudsagnet efter iterationen.

Else-case: Lad a' være den aktivitet i resten af X, som har mindst sluttid. Det er nemt at se, at man kan udskifte a' med den af algoritmen valgte a uden at få overlap med andre i X. Da dette ikke ændrer antal aktiviteter i X, er den stadig optimal, og kan bruges i induktionsudsagnet efter iterationen.

Køretid: Sortering + $O(n)$.

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. Et simplet udskiftningsargument viser, at den optimale løsning kun kan være som den af algoritmen valgte.

Rygsæksproblemet

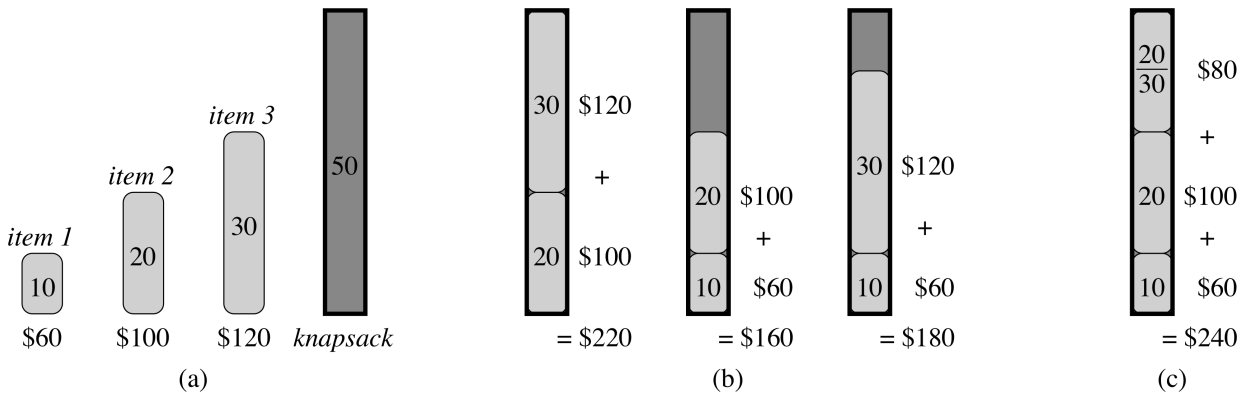
"Fractional" version af problemet (dele af ting kan medtages i rygsækken) kan løses med en grådig algoritme: vælg tingene efter aftagende "nytte" = værdi/vægt. Et simplet udskiftningsargument viser, at den optimale løsning kun kan være som den af algoritmen valgte.

NB: Denne grådige algoritme virker IKKE for 0-1 versionen af problemet (hvor 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. Et simplet udskiftningsargument viser, at den optimale løsning kun kan være som den af algoritmen valgte.

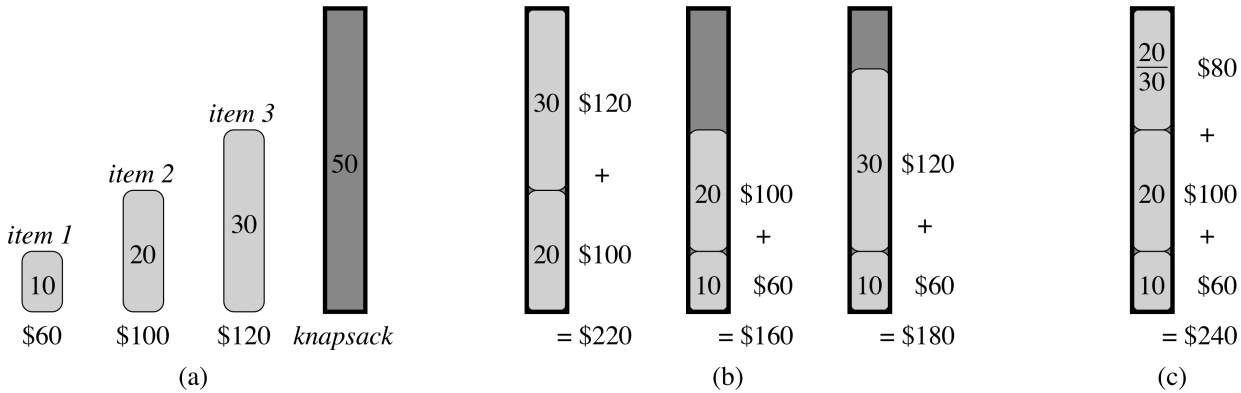
NB: Denne grådige algoritme virker IKKE for 0-1 versionen af problemet (hvor kun hele ting kan medtages):



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. Et simplet udskiftningsargument viser, at den optimale løsning kun kan være som den af algoritmen valgte.

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



Eksempel på at "grådige valg" ikke bare kan antages at virke for alle problemer (lokal optimering giver ikke altid global optimering).

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:

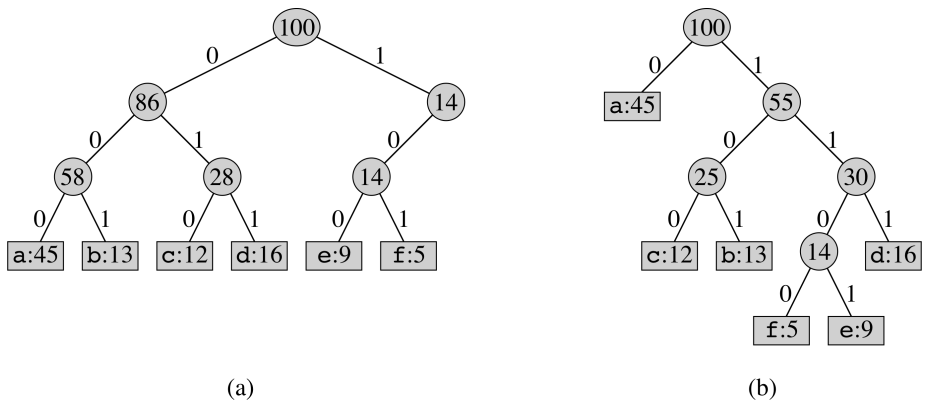
$$1 \cdot 45.000 + 3 \cdot 13.000 + \cdots + 4 \cdot 5.000 = 224.000 \text{ bits}$$

Prefix-kode = træer

Kodeord = sti i binært træ: 0 $\sim$ gå til venstre, 1 $\sim$ gå til højre

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



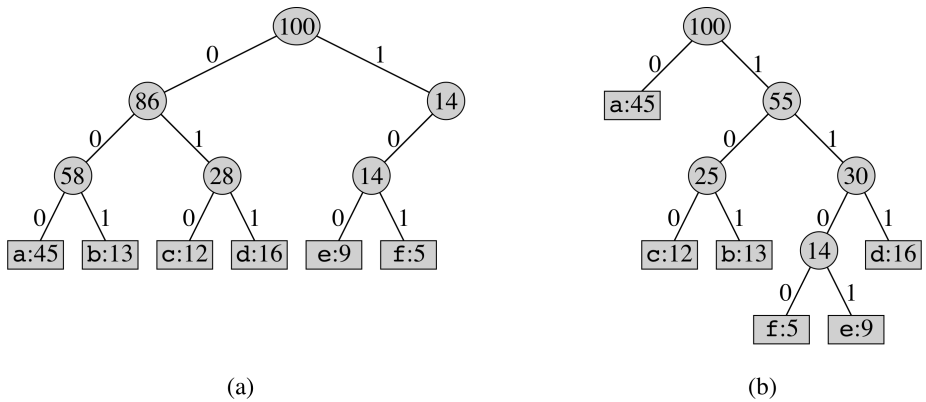
(a) (b)

Prefix-kode = træer

Kodeord = sti i binært træ: 0 $\sim$ gå til venstre, 1 $\sim$ gå til højre

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)

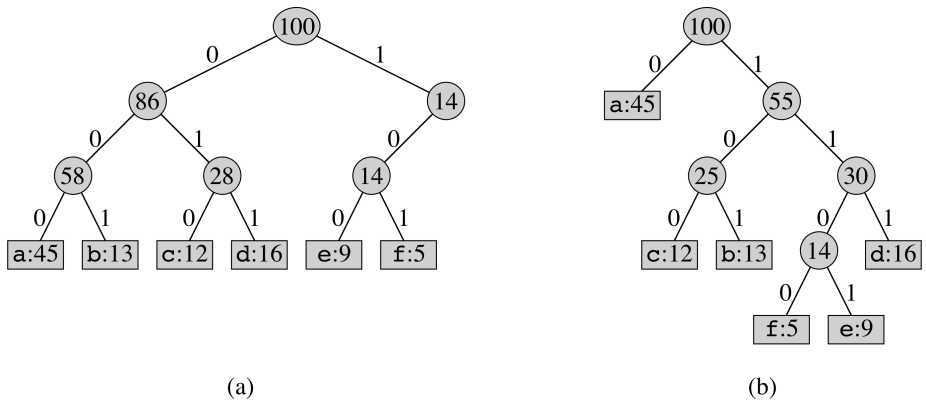
For en givet fil (tegn og deres frekvenser), find bedste variable-length prefix-kode. Dvs. for Cost(tree) = |kodet fil|, find træ med lavest cost.

Prefix-kode = træer

Kodeord = sti i binært træ: 0 ∼ gå til venstre, 1 ∼ gå til højre

Prefix(-fri) kode: ingen kode for et tegn er starten (prefix) af koden for et andet tegn (⇒ 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



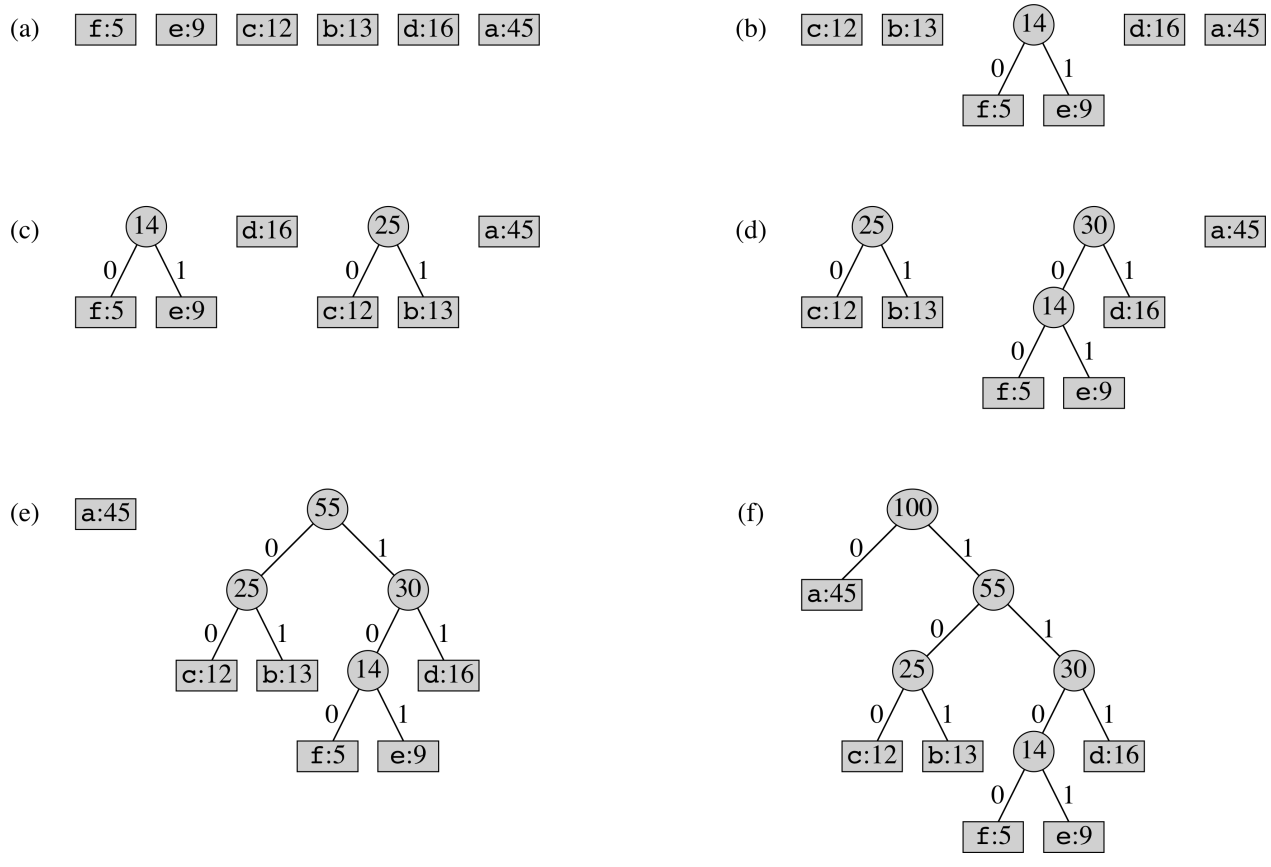
(a) (b)

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 ét barn (alle tegn i undertræet for en sådan knude kan forkortes med en 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, ét træ til slut, og hver iteration mindsker antallet med præcis én.)

Køretid

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

- $n - 1$ iterationer.

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

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

- to ExtractMin-operationer
- én 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, ét træ til slut, og hver iteration mindsker antallet med præcis én.)

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

- to ExtractMin-operationer
- én 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, ét træ til slut, og hver iteration mindsker antallet med præcis én.)

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

- to ExtractMin-operationer
- én 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

Eksempel: Hvis input er

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

er følgende rækkefølger (samt en del yderligere) alle mulige:

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

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

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

Huffman vælger én af dem, og slår de to første tegn sammen.

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 c_1 og 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 c_1 og 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 ét undertræ, se tidligere). Dens andet undertræ må være blot 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 c_1 og 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 ét undertræ, se tidligere). Dens andet undertræ må være blot 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 c_i og c_j, med $i < j$.

Korrekthed

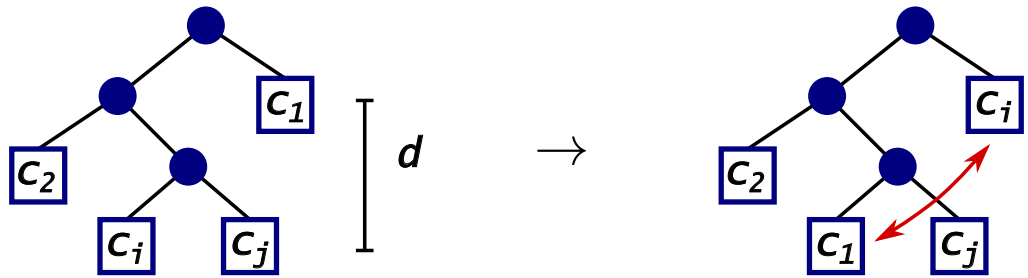
Mulige situationer:

c_1	c_2	$c_3 \ldots$	Handling:
c_i	c_j		Ingen
c_i		c_j	Byt c_2 og c_j
	c_i	c_j	Byt c_1 og c_j
		$c_i \quad c_j$	Byt c_1 og c_i, samt c_2 og c_j

Korrekthed

Mulige situationer:

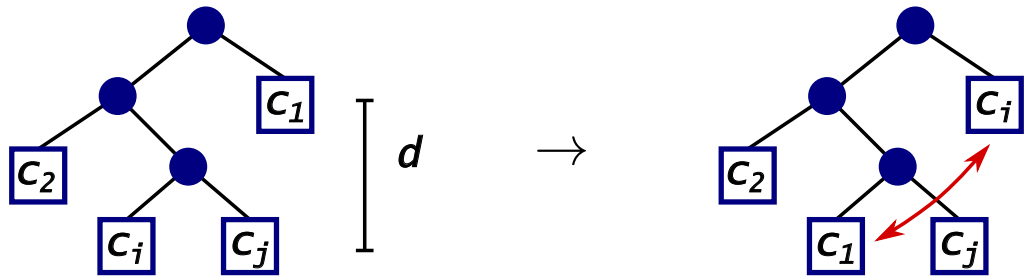
c_1	c_2	$c_3 \ldots$	Handling:
c_i	c_j		Ingen
c_i		c_j	Byt c_2 og c_j
	c_i	c_j	Byt c_1 og c_j
		$c_i \quad c_j$	Byt c_1 og c_i, samt c_2 og c_j



Korrekthed

Mulige situationer:

c_1	c_2	$c_3 \ldots$	Handling:
c_i	c_j		Ingen
c_i		c_j	Byt c_2 og c_j
	c_i	c_j	Byt c_1 og c_j
		$c_i \quad c_j$	Byt c_1 og c_i, samt c_2 og c_j

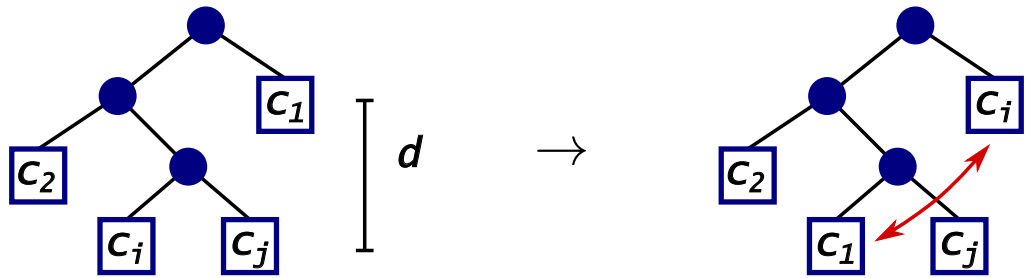


Lad d være c_i's blads dybde minus c_1's blads dybde. Da er ændringen af filens længde ved byt af c_1 og c_i lig $d \cdot f_1 - d \cdot f_i = d \cdot (f_1 - f_i)$.

Korrekthed

Mulige situationer:

c_1	c_2	$c_3 \ldots$	Handling:
c_i	c_j		Ingen
c_i		c_j	Byt c_2 og c_j
	c_i	c_j	Byt c_1 og c_j
		$c_i \quad c_j$	Byt c_1 og c_i, samt c_2 og c_j



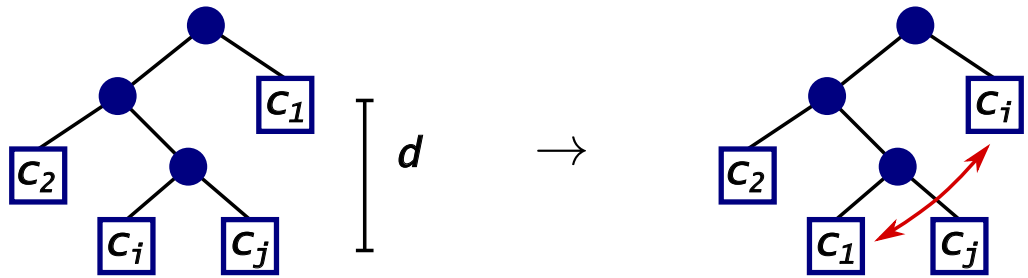
Lad d være c_i's blads dybde minus c_1's blads dybde. Da er ændringen af filens længde ved byt af c_1 og c_i lig $d \cdot f_1 - d \cdot f_i = d \cdot (f_1 - f_i)$.

Eftersom $d \geq 0$ (da c_i'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 c_1 og c_i.

Korrekthed

Mulige situationer:

c_1	c_2	$c_3 \ldots$	Handling:
c_i	c_j		Ingen
c_i		c_j	Byt c_2 og c_j
	c_i	c_j	Byt c_1 og c_j
		$c_i \quad c_j$	Byt c_1 og c_i, samt c_2 og c_j



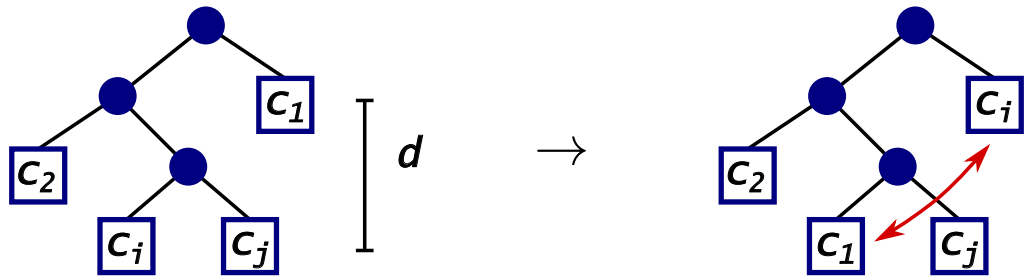
Lad d være c_i's blads dybde minus c_1's blads dybde. Da er ændringen af filens længde ved byt af c_1 og c_i lig $d \cdot f_1 - d \cdot f_i = d \cdot (f_1 - f_i)$.

Eftersom $d \geq 0$ (da c_i'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 c_1 og c_i. Tilsvarende kan vises for et byt af c_1 og c_j, og for et byt af c_2 og c_j.

Korrekthed

Mulige situationer:

c_1	c_2	$c_3 \ldots$	Handling:
c_i	c_j		Ingen
c_i		c_j	Byt c_2 og c_j
	c_i	c_j	Byt c_1 og c_j
		$c_i \quad c_j$	Byt c_1 og c_i, samt c_2 og c_j



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

Eftersom $d \geq 0$ (da c_i'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 c_1 og c_i. Tilsvarende kan vises for et byt af c_1 og c_j, og for et byt af c_2 og c_j.

Da træet var optimalt før byt, er det også efter. Og c_1 og c_2 er nu søskende. □

Korrekthed

Sætning: Huffmans algoritme returnerer et optimalt træ.

Korrekthed

Sætning: Huffmans algoritme returnerer et optimalt træ.

Bevis: Vi vil vise via induktion at det gælder for input med n tegn, for alle $n \geq 2$.

Korrekthed

Sætning: Huffmans algoritme returnerer et optimalt træ.

Bevis: Vi vil vise via induktion at det gælder for input med n tegn, for alle $n \geq 2$.

Basis $n = 2$: Da knuder i optimale træer enten har to eller nul børn, er der præcis ét træ for $n = 2$ (en rod og to blade), hvilket dermed er det optimale træ. Det er netop træet returneret af Huffmans algoritme når $n = 2$.

Korrekthed

Sætning: Huffmans algoritme returnerer et optimalt træ.

Bevis: Vi vil vise via induktion at det gælder for input med n tegn, for alle $n \geq 2$.

Basis $n = 2$: Da knuder i optimale træer enten har to eller nul børn, er der præcis ét træ for $n = 2$ (en rod og to blade), hvilket dermed er det optimale træ. Det er netop træet returneret af Huffmans algoritme når $n = 2$.

Induktionsskridtet $n > 2$: 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, da antal tegn i dette input er $n-1$).
- Lad T'' være et optimalt træ på input I hvor bladene for tegnene c_1 og c_2 er søskende (et sådant optimalt træ eksisterer iflg. lemma).
- Lad T''' være T'' hvor bladene for tegnene c_1 og c_2 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, da antal tegn i dette input er $n-1$).
- Lad T'' være et optimalt træ på input I hvor bladene for tegnene c_1 og c_2 er søskende (et sådant optimalt træ eksisterer iflg. lemma).
- Lad T''' være T'' hvor bladene for tegnene c_1 og c_2 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 c_1 og c_2.

Korrekthed

- Lad T være Huffmans output på input I.
- Lad T' være Huffmans output på input I' (optimalt træ pr. induktionsantagelse, da antal tegn i dette input er $n-1$).
- Lad T'' være et optimalt træ på input I hvor bladene for tegnene c_1 og c_2 er søskende (et sådant optimalt træ eksisterer iflg. lemma).
- Lad T''' være T'' hvor bladene for tegnene c_1 og c_2 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 c_1 og c_2.

- $T'' = \text{Expand}(T''')$ (pga. definition af T''').
- $\text{Cost}(\text{Expand}(\tau)) = \text{Cost}(\tau) + f_1 + f_2$ (koder for c_1 og c_2 forlænges med én).
- $T = \text{Expand}(T')$ (pga. Huffman-algoritmens virkemåde).

Korrekthed

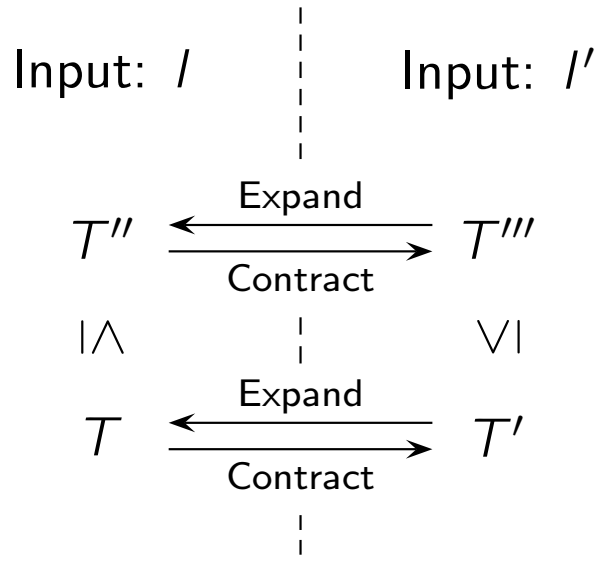
- Lad T være Huffmans output på input I.
- Lad T' være Huffmans output på input I' (optimalt træ pr. induktionsantagelse, da antal tegn i dette input er $n-1$).
- Lad T'' være et optimalt træ på input I hvor bladene for tegnene c_1 og c_2 er søskende (et sådant optimalt træ eksisterer iflg. lemma).
- Lad T''' være T'' hvor bladene for tegnene c_1 og c_2 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 c_1 og c_2.

- $T'' = \text{Expand}(T''')$ (pga. definition af T''').
- $\text{Cost}(\text{Expand}(\tau)) = \text{Cost}(\tau) + f_1 + f_2$ (koder for c_1 og c_2 forlænges med én).
- $T = \text{Expand}(T')$ (pga. Huffman-algoritmens virkemåde).

$\text{Cost}(T'') \leq \text{Cost}(T) = \text{Cost}(\text{Expand}(T')) = \text{Cost}(T') + f_1 + f_2 \leq \text{Cost}(T''') + f_1 + f_2 = \text{Cost}(\text{Expand}(T''')) = \text{Cost}(T'')$.

Illustration



Contract(τ) er her det omvendte af Expand() [kun mulig når c_1 og c_2 er søskende i τ].

Argument igen:

$\text{Cost}(T'') \leq \text{Cost}(T) = \text{Cost}(\text{Expand}(T')) = \text{Cost}(T') + f_1 + f_2 \leq \text{Cost}(T''') + f_1 + f_2 = \text{Cost}(\text{Expand}(T''')) = \text{Cost}(T'')$.

Så $\text{Cost}(T'') = \text{Cost}(T)$, hvilket beviser at Huffman leverer et optimalt træ for input I. □