1. (a) In een B-boom van orde m bevat de wortel minimaal 1 sleutel en maximaal $m-1$ sleutels. De andere knopen bevatten minimaal $\lceil \frac{m}{2} \rceil - 1$ sleutels en maximaal $m-1$ sleutels.

   (b) In een B-boom van orde 5 bevat elke knoop minimaal twee sleutels (behalve eventueel de wortel) en maximaal vier sleutels. Derhalve heeft een knoop minimaal drie kinderen (behalve eventueel de wortel, en de bladeren natuurlijk) en maximaal vijf kinderen. Daarom:

op niveau	minimaal # knopen	minimaal # sleutels	maximaal # knopen	maximaal # sleutels
1	1	1	1	4×1
2	2	2×2	5	4×5
3	2×3	$2 \times 2 \times 3$	5×5	$4 \times 5 \times 5$
4	$2 \times 3 \times 3$	$2 \times 2 \times 3 \times 3$	$5 \times 5 \times 5$	$4 \times 5 \times 5 \times 5$
.	...	...	...	...
h	$2 \times 3^{h-2}$	$2 \times 2 \times 3^{h-2}$	5^{h-1}	$4 \times 5^{h-1}$

   Het minimale aantal sleutels op niveau h is derhalve $\begin{cases} 1 & \text{als } h = 1; \\ 4 \times 3^{h-2} & \text{als } h \geq 2. \end{cases}$

   Het maximale aantal sleutels op niveau h is $4 \times 5^{h-1}$ voor iedere $h \geq 1$.

   Er zijn bonuspunten te verdienen voor wie dit netjes met inductie afleidt.

   (c) We pakken de kolommen 'minimaal # sleutels' en 'maximaal # sleutels' uit de tabel van onderdeel (b) en maken die cumulatief:

t/m niveau	minimaal # sleutels	maximaal # sleutels
1	1	4
2	5	24
3	17	124
4	53	624
5	161	3124
6	485	...
7	1457	...

   Met $h \leq 3$ heb je hoogstens 124 sleutels.
   Met $h \geq 7$ heb je minstens 1457 sleutels.
   Met $h = 4$ kun je 624 sleutels precies kwijt.
   Met $h \in \{5, 6\}$ heb je speling.

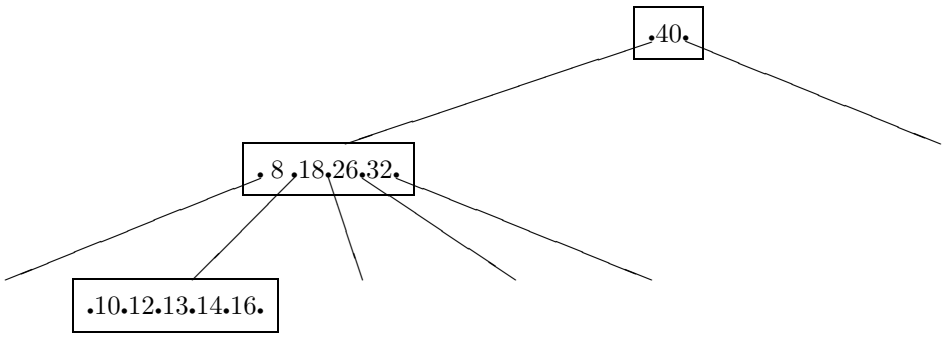
   Met $h \in \{4, 5, 6\}$ kun je dus 624 sleutels in een B-boom van orde 5 hebben.

   Overigens kan men met inductie algemeen afleiden dat een B-boom van orde 5 en hoogte h minstens $2 \times 3^{h-1} - 1$ en hoogstens $5^h - 1$ sleutels bevat. Hiermee zijn echter geen bonuspunten te verdienen.

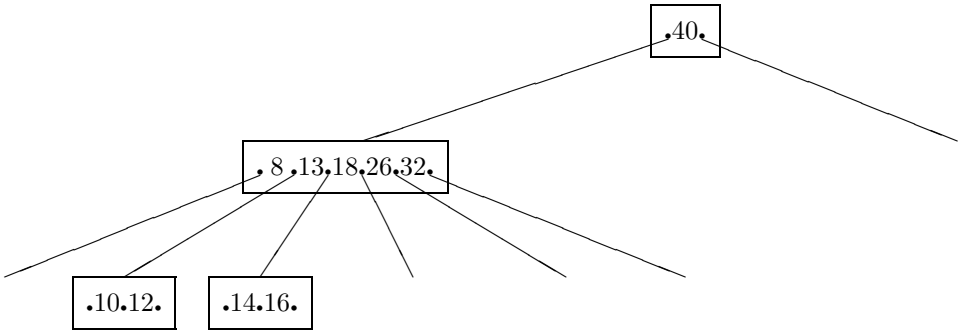
   (d) De plaatjes van de B-bomen in dit onderdeel zijn niet compleet. Subbomen die hetzelfde zijn als in de originele boom B zijn vaak weggelaten.

i.

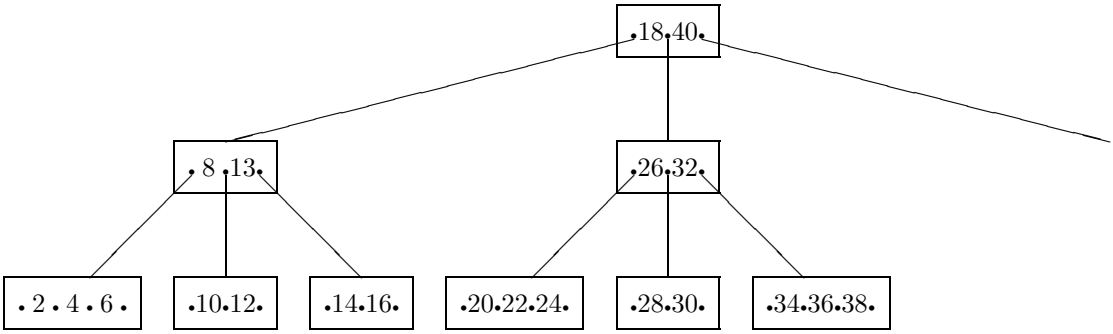
De sleutel 13 hoort thuis in het tweede blad vanaf links:



Na toevoeging van 13 aan dit blad, bevat het blad 5 sleutels, en dat is te veel. Het blad splitst op, en de middelste sleutel (toevallig 13 zelf) gaat omhoog naar de ouderknoop:

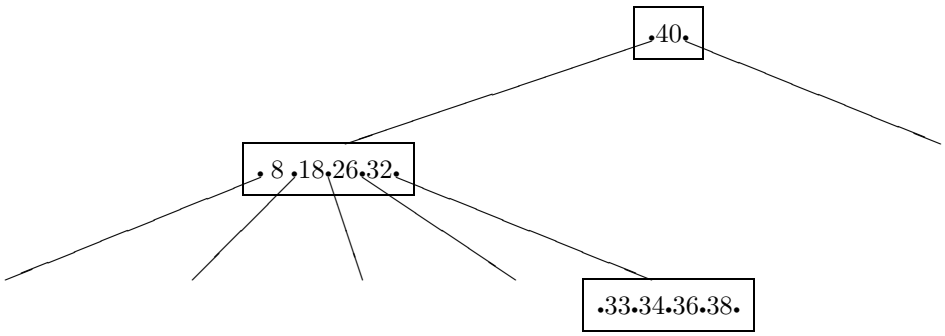


Nu is de ouderknoop op zijn beurt te vol. Ook deze splitst op, en de middelste sleutel (18) gaat omhoog naar de wortel:



ii.

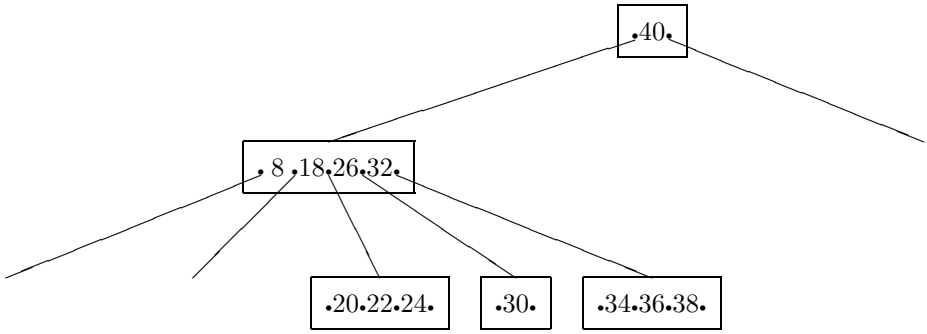
De sleutel 33 hoort thuis in het vijfde blad vanaf links. Wanneer we 33 daar aan toevoegen, bevat die knoop 4 sleutels, en dat mag gewoon. We hoeven geen knopen op te splitsen:



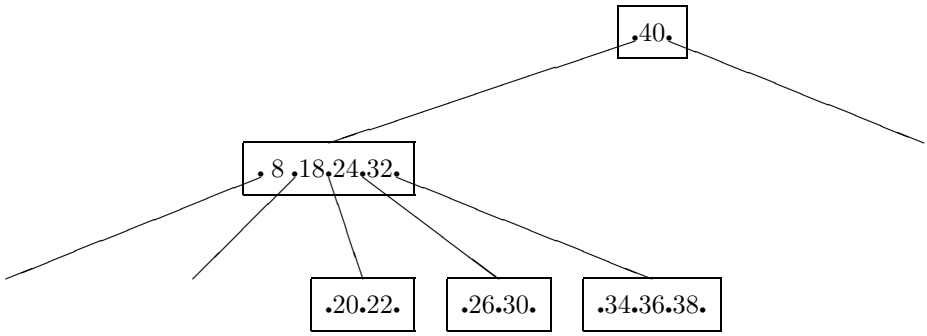
(e) De plaatjes van de B-bomen in dit onderdeel zijn niet compleet. Subbomen die hetzelfde zijn als in de originele boom B zijn vaak weggelaten.

i.

De sleutel 28 bevindt zich in een blad, en kan hier eenvoudig uit verwijderd worden:

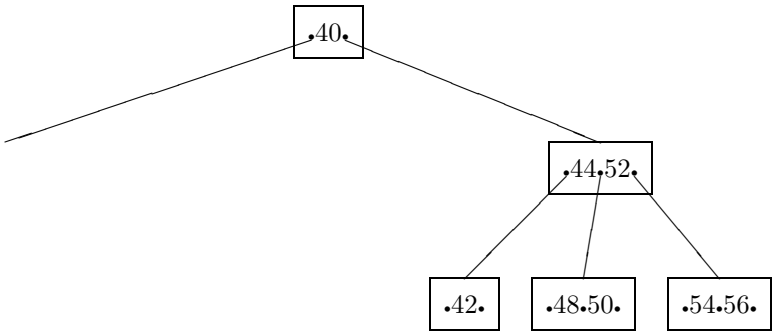


De knoop waar 28 uit verwijderd is, bevat nu nog maar één sleutel, en is dus te leeg geworden. Gelukkig heeft zijn linkerbroer (en ook zijn rechterbroer trouwens) een sleutel meer dan het minimale aantal twee. We lenen daarom een sleutel van deze linkerbroer ('via' de ouder):

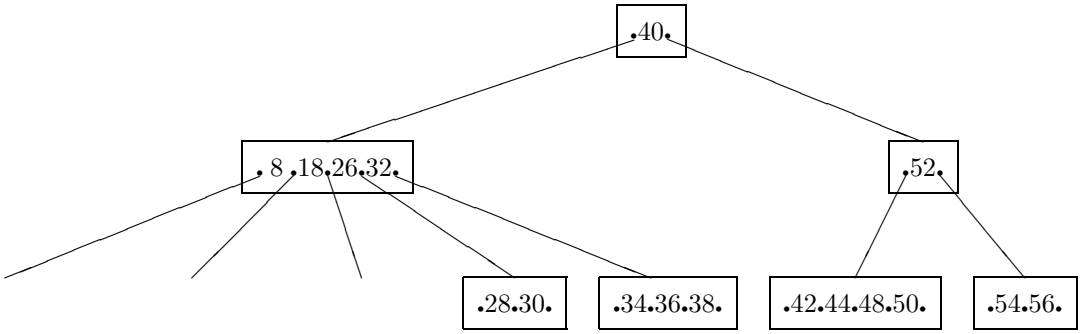


ii.

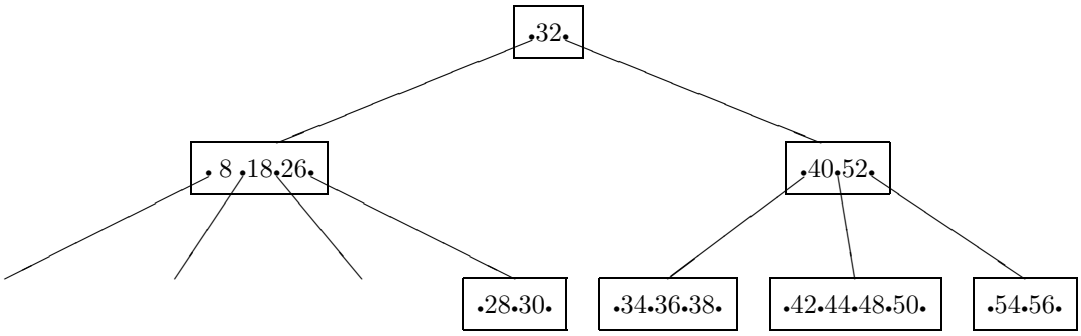
De sleutel 46 bevindt zich niet in een blad. We verwisselen hem met zijn voorganger in de B-boom (de grootste kleinere sleutel), sleutel 44, en verwijderen vervolgens 46:



De knoop met sleutel 42 bevat nu nog maar één sleutel, en is dus te leeg. Een linkerbroer heeft deze knoop niet, een rechterbroer wel, maar die zit al op het minimum van twee sleutels. Een sleutel lenen is dus niet mogelijk. De te lege knoop wordt daarom samengevoegd met zijn rechterbroer:



Nu bevat de knoop met sleutel 52 nog maar één sleutel, en is dus te leeg. Zijn linkerbroer bevat vier sleutels en daar lenen we er een van. Merk op hoe de tussenliggende subboom mee verhuist:



2. (a) Bij het coderen starten we in de wortel van de initiële codeerboom.

Iedere keer dat we een letter lezen, kijken we of de huidige knoop een kind(tak) met die letter heeft. Zo ja, dan stappen we gewoon naar dat kind toe. Zo nee, dan voeren we de code van de huidige knoop uit, we geven de huidige knoop een kind(tak) met de zojuist gelezen letter, kennen aan dit kind de eerstvolgende nieuwe code toe, springen terug naar de wortel van de codeerboom en stappen van daar naar het kind met de zojuist gelezen letter.

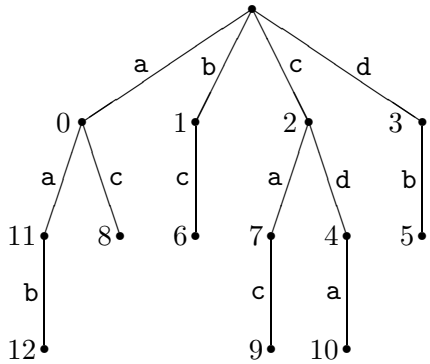
Wanneer we alle letters van de tekst gelezen hebben, voeren we de code van de huidige knoop uit.

Voorbeeld: we beginnen in de wortel van de codeerboom. We lezen de letter `c` en stappen naar het kind `c` van de wortel. Vervolgens lezen we de letter `d`. De huidige knoop heeft geen kind(tak) `d` (hij heeft überhaupt nog geen kinderen), dus we voeren de code 2 van de huidige knoop uit, geven de huidige knoop een kind(tak) `d`, kennen aan dit kind de eerstvolgende nieuwe code 4 (= `cd`) toe, springen terug naar de wortel van de codeerboom en stappen van daar naar het kind `d`.

Gaan we zo verder met coderen, dan vinden we:

Voer codes uit:	2	3	1	2	0	7		4		0	11		6	
Lees tekst in:	c	d	b	c	a	c	a	c	d	a	a	a	b	c
Maak nieuwe codes:	4	5	6	7	8	9		10		11	12			

De codeerboom die we opgebouwd hebben, ziet er als volgt uit:



(b) Iedere keer dat er bij het coderen een code werd uitgevoerd, werd er onderaan die code een nieuwe tak aangemaakt voor de eerstvolgende nieuwe code. De betreffende tak was de eerste tak van de code die als volgende zou worden uitgevoerd.

Voorbeeld: de eerste twee codes die werden uitgevoerd zijn code 1 (= `b`) en code 3 (= `d`). De eerste (tevens enige) letter van code 3 is `d`. Die letter werd dus als tak onderaan code 1 gehangen. Zo ontstond code 4 (= `bd`).

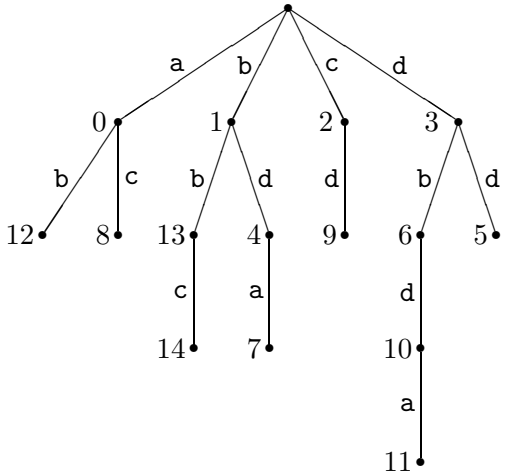
Gaan we zo verder met decoderen, dan vinden we:

Lees codes in:	1	3	3	4	0	2	6	10		0	1	13		9
Voer tekst uit:	b	d	d	b	d	a	c	d	b	d	b	d	a	b	b	b	c	d
Maak nieuwe codes:	4	5	6	7	8	9	10	11		12	13	14		

Twee gevallen verdienen een aparte vermelding, omdat de ingelezen code op het moment van decoderen nog niet in de boom hangt:

- De nieuwe code 10 bestaat uit code 6 (= `db`) gevolgd door de eerste letter van code 10. De eerste letter van code 10 is dus de eerste letter van code 6, de letter `d`. De totale code 10 wordt dan code 6 gevolgd door deze letter: `dbd`.
- De nieuwe code 13 bestaat uit code 1 (= `b`) gevolgd door de eerste letter van code 13. De eerste letter van code 13 is dus de eerste letter van code 1, de letter `b`. De totale code 13 wordt dan code 1 gevolgd door deze letter: `bb`.

De codeerboom die we opgebouwd hebben, ziet er als volgt uit:



(c) Iedere keer dat er bij het coderen een code werd uitgevoerd, werd er onderaan die code een nieuwe tak aangemaakt voor de eerstvolgende nieuwe code. De locaties van de toegevoegde codes (aan de initiële boom) 4, 5, 6, 7, 8, 9 verraden dus de substring die op dat moment gecodeerd werd.

Voorbeeld: voor de nieuwe code 4 (= `ab`) werd onderaan code 0 (= substring `a`) een tak `b` aangemaakt. De tekst begint dus met `ab` en de laatste letter hiervan (= `b`) moet de eerste letter van code 5 zijn (dat klopt).

Gaan we zo verder, dan vinden we:

Voer tekst uit:	a	b	a	a	b	c	a	b	a
Lees nieuwe codes:		4	5	6	7	8	9		

De string `abaabcaba` vormt derhalve het begin van de tekst. Na het aanmaken van de laatste code (code 9, eindigend met een tak `a`) zijn we teruggesprongen naar de wortel en zijn daar naar kind `a` gestapt.

Nadien is geen nieuwe code meer aangemaakt, dus de resterende letters van de tekst waren allemaal in de boom te vinden. Vanuit kind `a` van de wortel kunnen we de volgende paden in de boom volgen: – (het lege pad), `a`, `b`, `ba`, `bc`. De totale tekst kan derhalve geweest zijn:
`abaabcaba`
`abaabcabaa`
`abaabcabab`
`abaabcababa`
`abaabcababc`
(vijf mogelijkheden).

(d) We kunnen het coderen van de tekst vervolgen door

i. gewoon verder te gaan met de huidige codeerboom, maar geen nieuwe codes meer toe te voegen (zoals in de programmeeropdracht);

ii. helemaal opnieuw te beginnen met een nieuwe codeerboom voor de rest van de tekst (dit gebeurt uiteindelijk bij gif-compressie);

iii. de bladeren uit de codeerboom te verwijderen; deze corresponderen namelijk met codes (substrings) die niet opnieuw zijn voorgekomen na het aanmaken; de vrijkomende code(nummer)s kun je hergebruiken voor nieuwe substrings;

iv. de codelengte met een bit uit te breiden (dit gebeurt in eerste instantie bij gif-compressie); dan ontstaan er weer nieuwe codes die je kunt gebruiken; dit antwoord is maar half goed, omdat hierbij kennelijk toch nog niet alle beschikbare codes zijn toegekend.

3. (a) We construeren achtereenvolgens de matrices $A^0, A^1, \ldots, A^4$. Hierbij is A^0 de directe afstandenmatrix. Voor $k = 1, \ldots, 4$ bevat $A^k[i, j]$ het totale gewicht van het kortste pad van knoop i naar knoop j waarbij onderweg knopen uit de verzameling $\{1, \ldots, k\}$ gepasseerd mogen worden (voor $k = 0$ klopt dit overigens ook!). In deze laatste vier matrices is een waarde $A^k[i, j]$ onderstreept als ze beter is dan de corresponderende waarde in de vorige matrix ($A^{k-1}[i, j]$ dus).

A^0	1	2	3	4
1	0	3	1	∞
2	3	0	1	4
3	∞	1	0	6
4	-2	4	6	0

A^1	1	2	3	4
1	0	3	1	∞
2	3	0	1	4
3	∞	1	0	6
4	-2	$\underline{1}$	$\underline{-1}$	0

A^2	1	2	3	4
1	0	3	1	$\underline{7}$
2	3	0	1	4
3	$\underline{4}$	1	0	$\underline{5}$
4	-2	1	-1	0

A^3	1	2	3	4
1	0	$\underline{2}$	1	$\underline{6}$
2	3	0	1	4
3	4	1	0	5
4	-2	$\underline{0}$	-1	0

A^4	1	2	3	4
1	0	2	1	6
2	$\underline{2}$	0	1	4
3	$\underline{3}$	1	0	5
4	-2	0	-1	0

(b) In A^4 zien we dat het kortste pad van knoop 1 naar knoop 4 totaal gewicht 6 heeft. Dit gewicht is berekend in A^3. Dit betekent dat het kortste pad van knoop 1 naar knoop 4 via knoop 3 loopt ($1 \sim 3 \sim 4$).

In A^4 zien we dat het kortste pad van knoop 1 naar knoop 3 totaal gewicht 1 heeft. Dit gewicht kwam al voor in A^0. Dit betekent dat het kortste pad van knoop 1 naar knoop 3 de directe tak van knoop 1 naar knoop 3 is (1 – 3).

In A^4 zien we dat het kortste pad van knoop 3 naar knoop 4 totaal gewicht 5 heeft. Dit gewicht is berekend in A^2. Dit betekent dat het kortste pad van knoop 3 naar knoop 4 via knoop 2 loopt ($3 \sim 2 \sim 4$).

In A^4 zien we dat het kortste pad van knoop 3 naar knoop 2 totaal gewicht 1 heeft en dat het kortste pad van knoop 2 naar knoop 4 totaal gewicht 4 heeft. Beide gewichten kwamen al voor in A^0. Dit betekent dat het kortste pad van knoop 3 naar knoop 2 de directe tak van knoop 3 naar knoop 2 is (3 – 2) en dat het korste pad van knoop 2 naar knoop 4 de directe tak van knoop 2 naar knoop 4 is (2 – 4).

Het kortste pad van knoop 1 naar knoop 4 is derhalve 1 – 3 – 2 – 4.

(c) Kies beginknoop 3. De werking van het algoritme van Dijkstra kan dan als volgt in een tabel worden weergegeven:

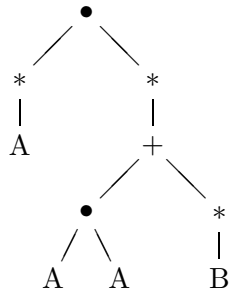
Kandidaat-waardes	Knoop 1	2	3	4	Actie
Stap 1	∞	1	-	6	Kies 2
2	3	-	-	4	Kies 1
3	-	-	-	4	Kies 4

Volgens het algoritme van Dijkstra zou het kortste pad van knoop 3 naar knoop 1 dus 3 – 2 – 1 zijn, met totaal gewicht 4. Het is echter 3 – 2 – 4 – 1, met totaal gewicht 3. Dit laatste pad wordt ook door het algoritme van Floyd gevonden; kijk maar na in de matrices van onderdeel (a)!

(d) Als een graaf een kring met een negatief totaal gewicht bevat, dan is niet voor alle paren knopen (i, j) het kortste pad van i naar j gedefinieerd. Neem immers twee knopen i en j op deze kring. Dan kun je het totale gewicht van het pad van i naar j altijd weer kleiner maken door de kring een keer (extra) te doorlopen.

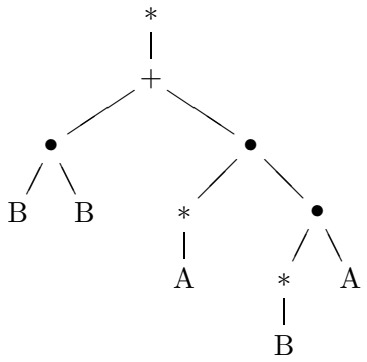
Als het kortste pad van knoop i naar knoop j niet gedefinieerd is, kun je moeilijk van het algoritme van Floyd verwachten dat het wel het kortste pad oplevert.

4. (a) Voor de tweede reguliere expressie zijn twee mogelijke semantische bomen. De keuze voor de een of de ander hangt af van de interpretatie van de term `A*B*A`, als '`(A*B*)A`' of als '`A*(B*A)`'. We geven de boom die hoort bij de tweede interpretatie. Merk op: bij de andere boom hoort ook een andere postfix notatie, hoewel hij natuurlijk wel dezelfde verzameling strings representeert.



Postfix notatie:
`A*AA•B*+*•`

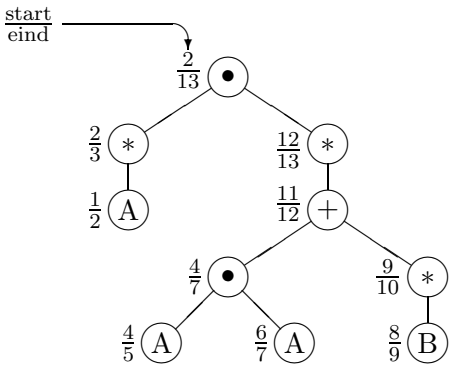
`i.`



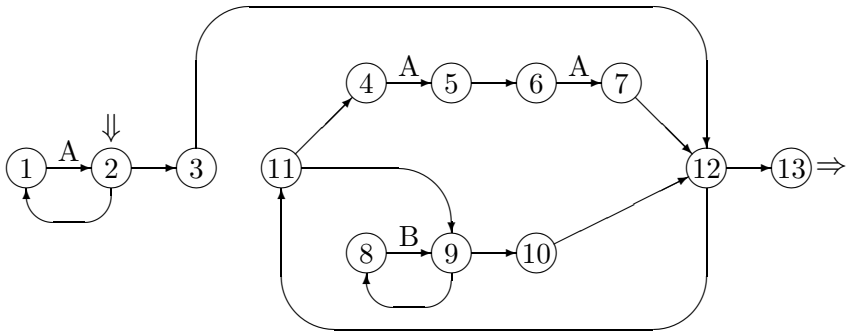
Postfix notatie:
`BB•A*B*A••+*`

`ii.`

(b) Als we bij iedere knoop in de semantische boom de corresponderende begin- en eindknoop van de eindige automaat zetten, ziet de boom er als volgt uit:

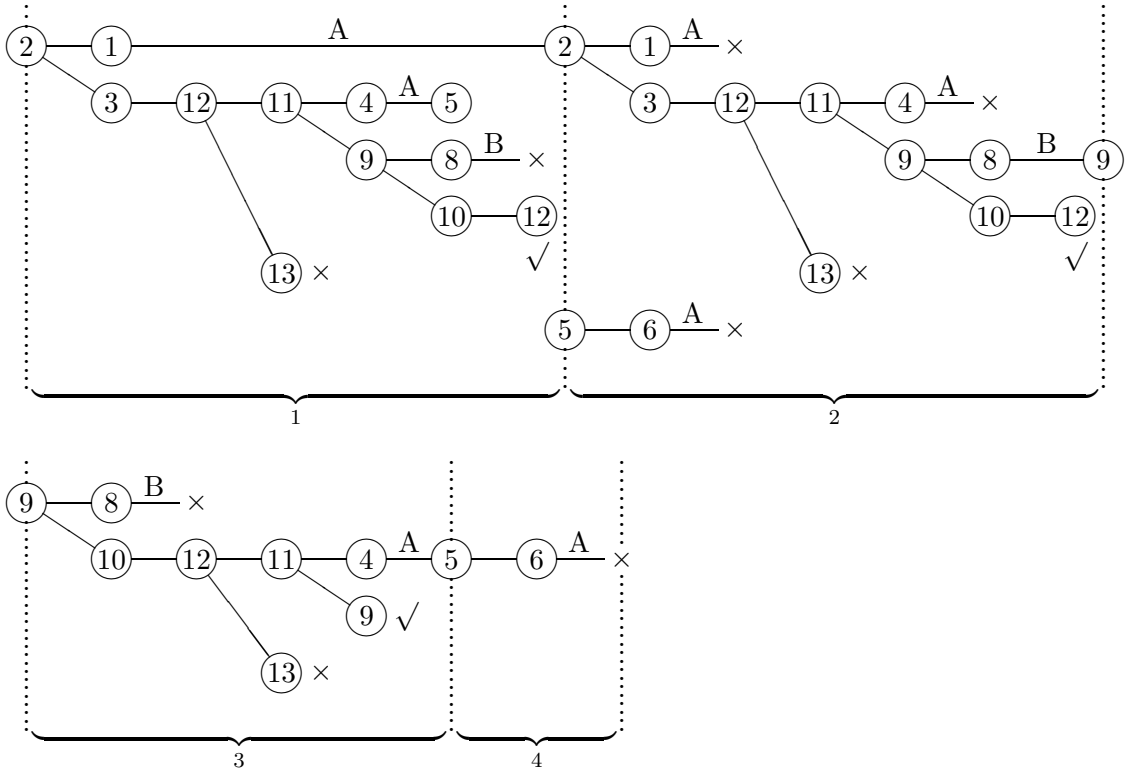


De bijbehorende eindige automaat is



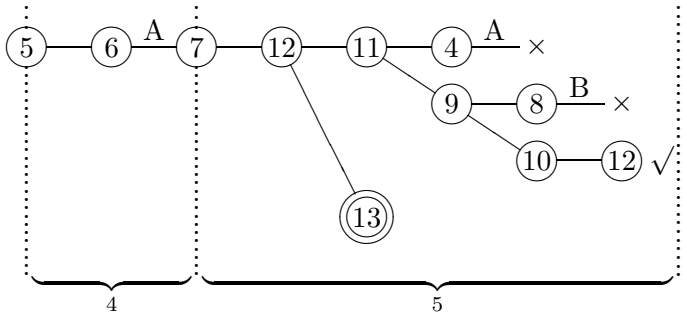
(c) Voor beide gezochte strings moeten we *letter-voor-letter* een depth-first-search uitvoeren. In de bijbehorende 'boom'structuren betekent:
×: tak met verkeerde letter of te vroeg in de eindtoestand;
√: deze knoop hebben we in de huidige stap al bezocht;
(13) : eindtoestand bereikt toen we er naar zochten.

Voor de eerste string ('ABA') voeren we vier depth-first-searches uit: eerst drie voor de drie letters van de string, en vervolgens nog een om te proberen de eindknoop van de automaat te bereiken.



Helaas, het lukt ons in de vierde zoektocht niet om de eindknoop van de automaat te bereiken. De string 'ABA' voldoet dus niet aan de gegeven expressie.

Voor de tweede string ('ABAA') voeren we vijf depth-first-searches uit: eerst vier voor de vier letters van de string, en vervolgens nog een om de eindknoop van de automaat te bereiken. De eerste drie zoektochten zijn uiteraard gelijk aan die bij de string 'ABA'. We vervolgen met de vierde zoektocht.



Nu bereiken we wel de eindknoop. De string '`ABAA`' voldoet dus wél aan de gegeven expressie.

(d) Een methode om erachter te komen of er een substring van een string S is die aan een reguliere expressie X voldoet, is:

- construeer op de gebruikelijke manier de eindige automaat voor de expressie X;
- voer de zoektocht naar de string S op de gebruikelijke manier uit, met twee aanpassingen:
  - begin iedere volgende depth-first-search niet alleen in de knopen die bereikt zijn in de vorige depth-first-search, maar ook in de beginknoop;
  - breek de zoektocht af zodra de eindknoop bereikt is; als we dit stadium bereiken, dan voldoet een substring van S aan X (en omdat we weten bij welke letter van S we waren gebleven, weten we ook waar die substring eindigt); als we dit stadium helemaal niet bereiken, dan voldoet geen enkele substring van S aan X.

Een alternatieve methode is:

- construeer op de gebruikelijke manier de eindige automaat voor de expressie X;
- voeg voor iedere letter die voor kan komen in string S een tak toe van de beginknoop naar zichzelf, met die letter als label;
- voeg voor iedere letter die voor kan komen in string S een tak toe van de eindknoop naar zichzelf, met die letter als label;
- creëer indien gewenst een nieuwe eindknoop, en een tak zonder letter van de oude eindknoop naar deze nieuwe eindknoop;
- voer de zoektocht naar de string S op de gebruikelijke manier uit.

Beide methodes zijn aangepaste varianten voor het testen van S aan de expressie `(A+B+···+Z)*` X `(A+B+···+Z)*`, waarbij `A`, `B`, ..., `Z` de letters in het alfabet zijn.

(Rudy van Vliet, juli 2002)