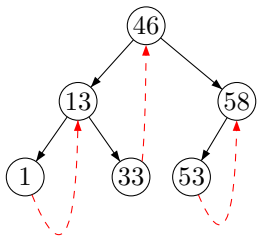
1. a) De bedrade boom:



b) Inorde volgorde: $1, 13, 33, 46, 53, 58$.

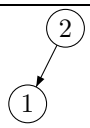
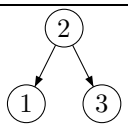
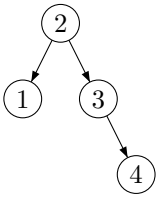
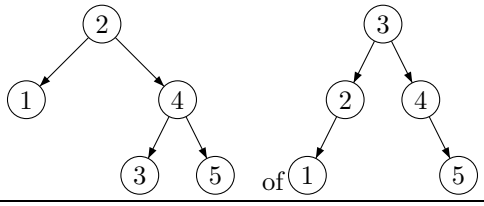
-

c) Algoritme van Morris:

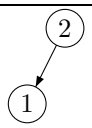
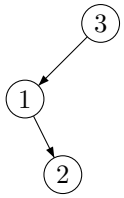
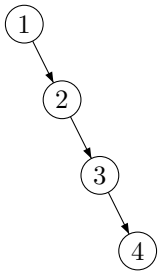
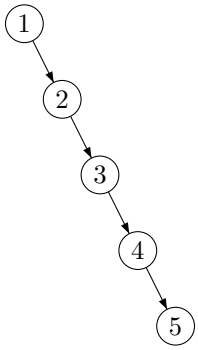
morris-algo

```
1  Curr = Root;
2  while (Curr != nil) do
3      if (Curr.left = nil) then
4          inOrderVisit( Curr )
5          Curr = Curr.right
6      else
7          // find predecessor
8          Pred = Curr.left
9          while (Pred.right != Curr && Pred.right != nil) do
10             Pred = Pred.right
11         od
12         if  (Pred.right=nil) then
13             // no thread: subtree not yet visited
14             Pred.right = Curr
15             Curr = Curr.left
16         else
17              // been there, remove thread
18              Pred.right = nil
19              inOrderVisit( Curr )
20              Curr = Curr.right
21         fi
22     fi
23 od
```

2. **a)** i. Let op! Er zijn meerdere goede antwoorden. Onderstaande tabel geeft een paar mogelijkheden weer, maar niet alle mogelijkheden. In iedere boom mag bij elke knoop het linker- en rechterkind verwisseld zijn! Maar dan wel met een ander label, anders wordt niet aan de BST-eigenschap voldaan dat het linkerkind kleiner is dan de knoop, en het rechterkind groter.

n	ipl	epl	worst-case AVL bomen met n knopen
1	0		
2	1		
3	2		
4	4		
5	6	16	

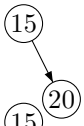
ii. Ook hier zijn meerdere goede antwoorden: bij elke knoop mogen het linker- en rechterkind verwisseld zijn (zie bv $n = 3$)! De labelling moet uiteraard wel voldoen aan de BST eigenschap.

n	ipl	epl	worst-case BST van n knopen
1	0		
2	1		
3	3		
4	6		
5	10	20	

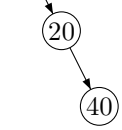
b) Uit het minimale aantal knopen n van een boom met hoogte h kunnen we de worst-case complexiteit afleiden van het zoeken naar een element in de boom. In een AVL-boom van n knopen is h dan namelijk de lengte van het langst mogelijke pad, dus een onsuccesvolle zoekactie kan in het slechtste geval $h + 1$ vergelijkingen kosten, en een succesvolle zoekactie kost maximaal h vergelijkingen.

c) Hieronder wordt achtereenvolgens $15, 20, 40, 84, 91, 80, 2, 77, 34 en 78$ toegevoegd aan een lege AVL-boom.

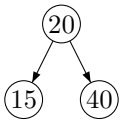
Na insert 15



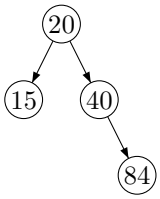
Na insert 20



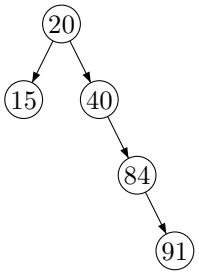
Direct na insert 40 - RR onbalans bij node 15



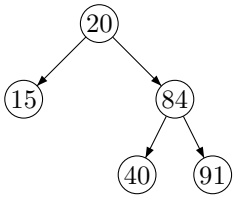
Na insert 40 - balans hersteld mbv rotatie van node 20 om zijn ouder



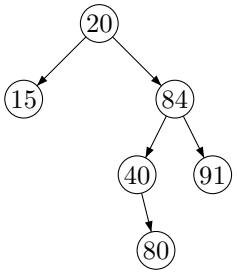
Na insert 84



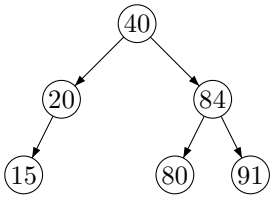
Direct na insert 91 - RR-onbalans bij node 40



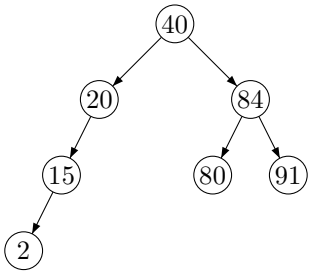
Na insert 91 - balans hersteld mbv rotatie van node 84 om zijn ouder



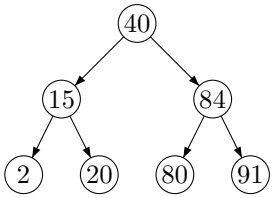
Na insert 80 - RL-onbalans bij node 20



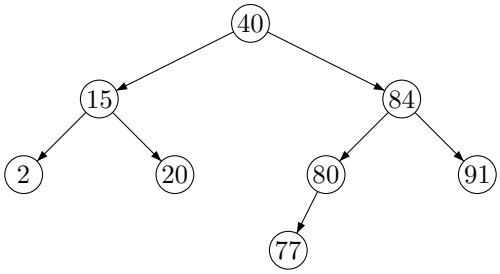
Na insert 80 - balans hersteld met dubbele rotatie van node 40 met (voor)ouders



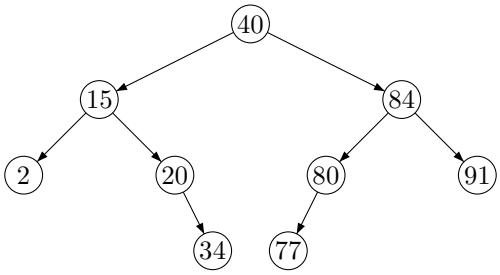
Direct na insert 2 - LL-onbalans bij 20



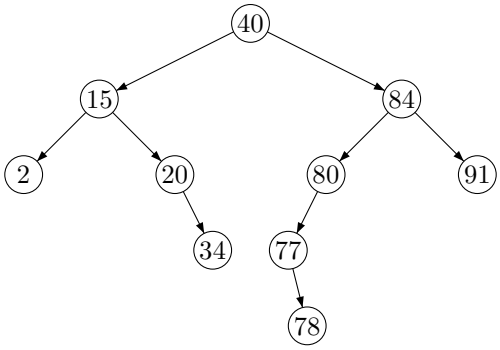
Na insert 2 - balans hersteld met rotatie van node 15 met ouder



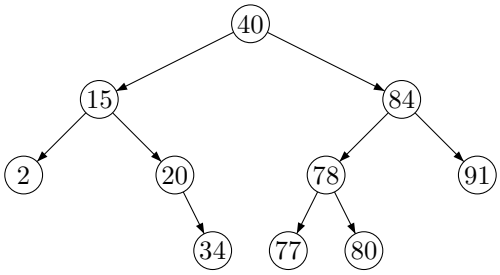
Na insert 77



Na insert 34

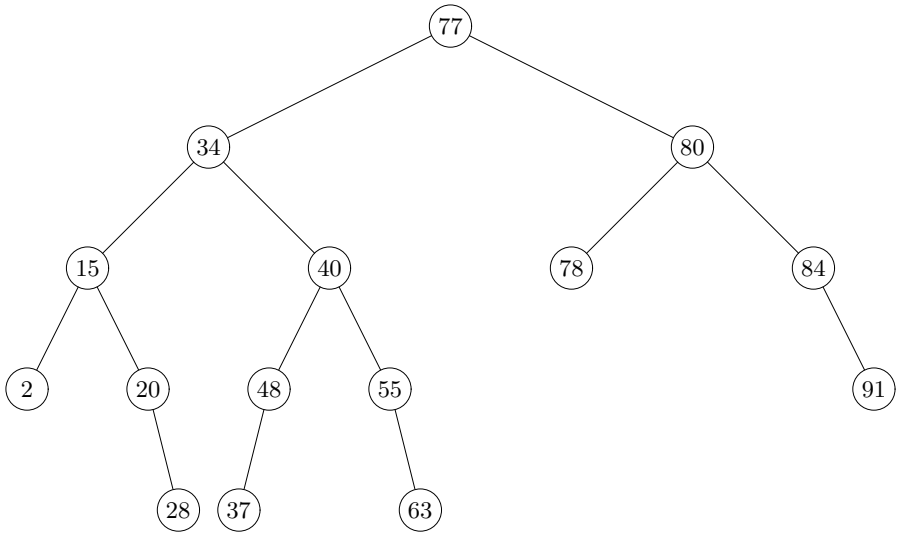


Direct na insert 78 - LR-onbalans bij 80

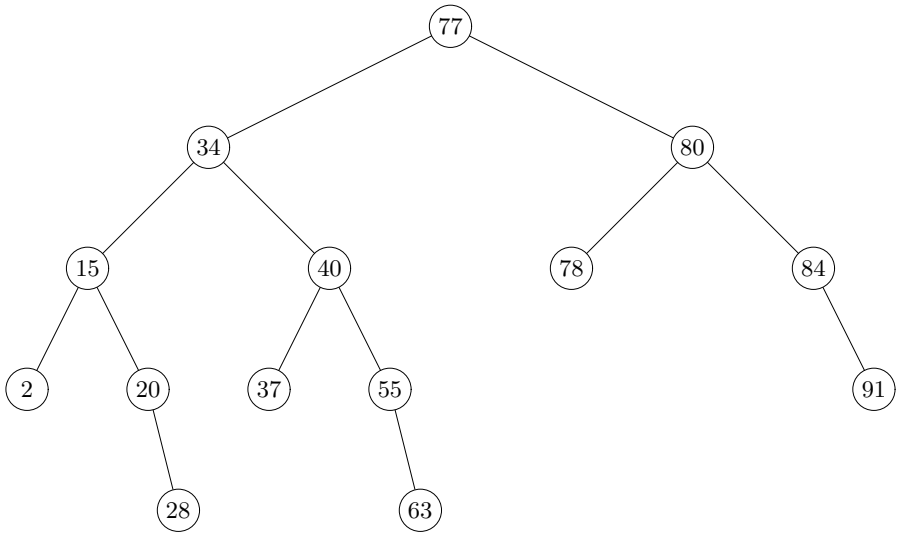


Na insert 78 - balans hersteld met dubbele rotatie van node 40 met (voor)ouders

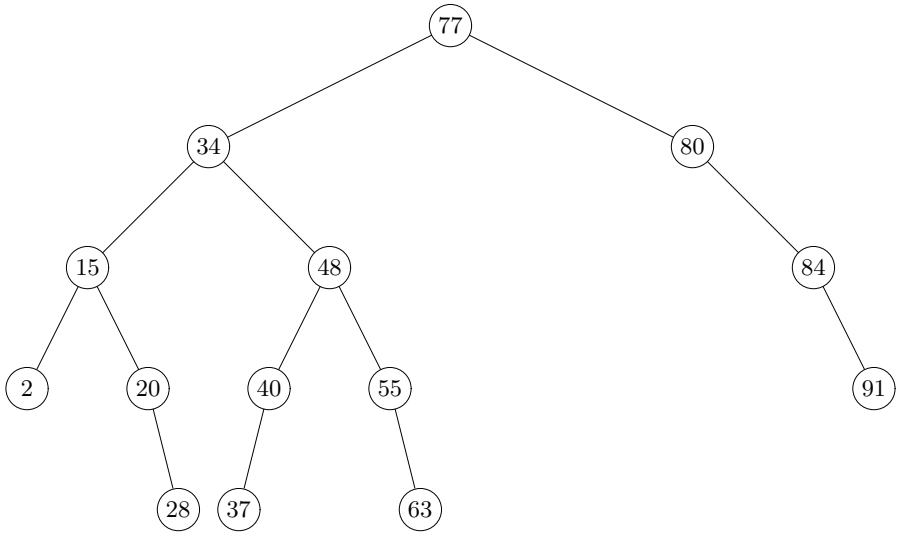
d) i. Verwissel eerst 48 met zijn inorde voorganger (40) of opvolger (55). Op de nieuwe plek van 48, heeft 48 nog maar 1 kind en kunnen we hem direct verwijderen. Hieronder verwisselen we met inorde voorganger, dat resulteert in:



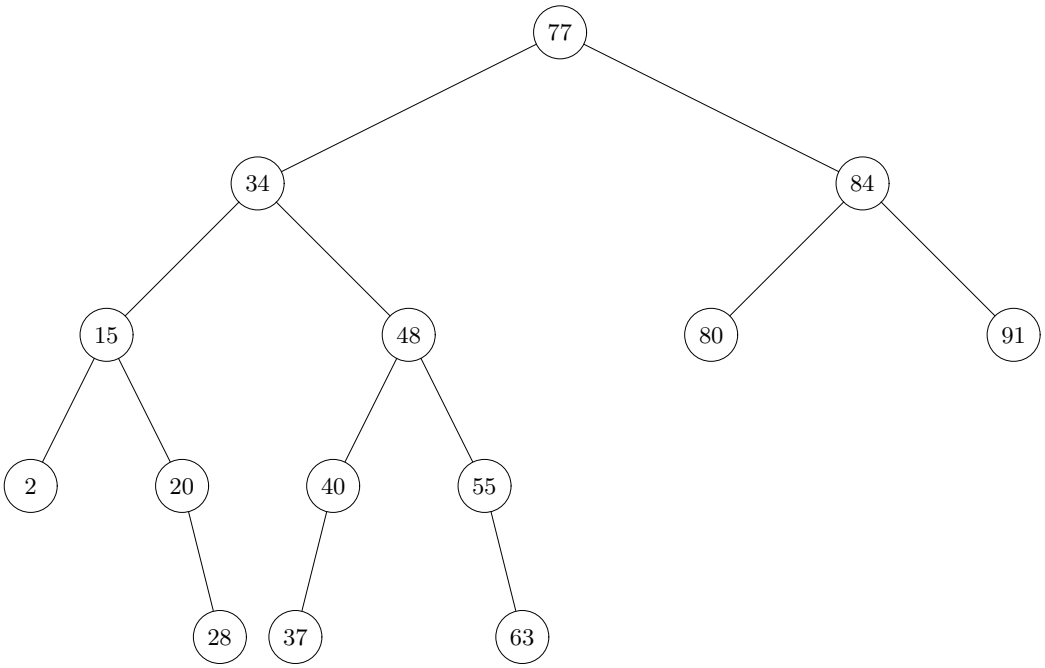
Verwijderen van 48 geeft nu:



De boom is voldoende in balans (voldoet aan AVL eigenschap), dus we zijn klaar.
De node met 78 heeft geen kinderen dus kunnen we direct verwijderen, met als resultaat:

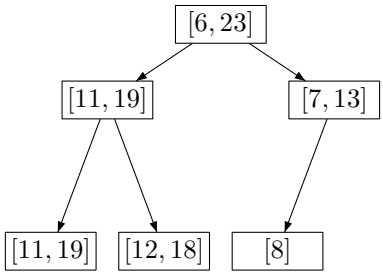


Als we nu naar boven lopen, is de eerste node met onbalans node 80. Er is RR-onbalans, dus roteer 84 om 80 heen:

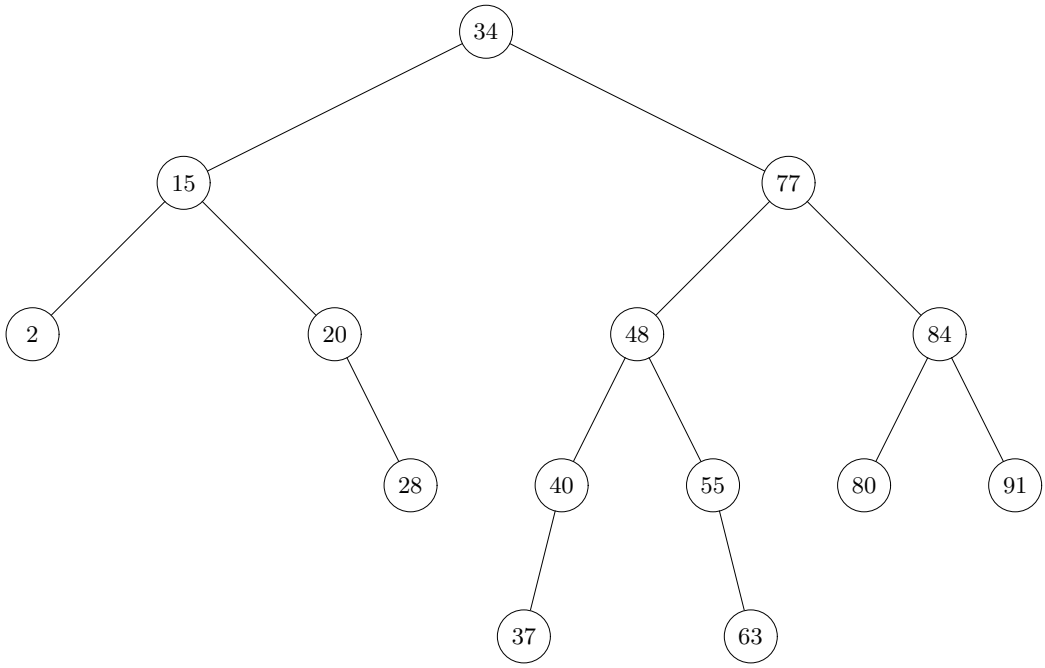


Verder naar boven lopen, zien we dat LL-onbalans bij 77 is (of: LR-onbalans, dan is dubbele rotatie op 48 met (voor)ouders nodig), dus roteer 34 om 77 heen:

Klopt niet
met algoritme.
maar is wel een
interval heap



Figuur 1: Interval heap met elementen $11, 8, 11, 12, 6, 19, 13, 7, 18, 19, 23$



Bovenstaande boom is voldoende in balans (voldoet aan AVL-eigenschap) dus we zijn klaar.

3. a) Een heap is een *complete* binaire boom waarbij de waarde in iedere knoop kleiner of gelijk is aan die in zijn kinderen.

b) Een binaire heap waarbij ieder knoop twee getallen bevat (behalve de laatste knoop, als de heap een oneven aantal getallen bevat). De twee getallen, bijvoorbeeld x, y, worden als intervals $[x, y]$ beschouwd en geordend als inclusie van intervals. Dat wil zeggen: als een knoop het paar x, y bevat, en zijn kind bevat x', y', dan moet gelden: $x \leq x' \leq y' \leq y$, ofwel $[x', y'] \subseteq [x, y]$.

Zowel het minimale als maximale element kan snel (in constante tijd) gevonden worden: deze getallen staan in de root van de interval heap.

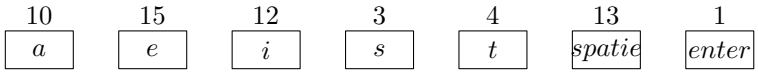
c) Fig. 1 toont een interval heap met de gevraagde elementen.

4. a) Lossless compresseren van gegevens (data kan worden gedecompresseerd naar zijn oorspronkelijke inhoud).

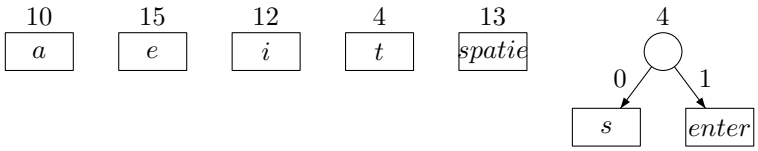
b) Het algoritme van Huffman compresseert het bericht M in 7 stappen, weergegeven in de figuren: Fig. 2, Fig. 3, Fig. 4, Fig. 5, Fig. 6, Fig. 7, Fig. 8. De volgende tabel geeft de uiteindelijke codering (af te lezen in Fig. 8).

letter	*a*	*e*	*i*	*s*	*t*	spatie	enter
freq.	10	15	12	3	4	13	1
code	110	10	00	11110	1110	01	11111

Let op! Meedere antwoorden mogelijk. *Welke* twee bomen gemerged worden in iedere stap is belangrijk (de twee met de laagste frequentie), maar de studenten mogen zelf kiezen of de bomen die gemerged worden als linker- of rechterkind worden gebruikt van de nieuwe boom.



Figuur 2: Huffman stap 1 - Beginsituatie



Figuur 3: Huffman stap 2 - na mergen bomen met frequenties 3 en 1

c) De codering van iedere letter kan worden afgelezen uit de laatste Huffman boom uit 4b. Onderstaande tabel geeft een samenvatting van de frequentie en bitlengte van de gecompresseerde versie van iedere letter:

letter	a	e	i	s	t	spatie	enter
freq.	10	15	12	3	4	13	1
bitlengte	3	2	2	5	4	2	5

Om de gecompresseerde lengte van M te bepalen: tel op van alle letters op: freq*bitlengte. Dit geeft $\#code(M) = 10*3+15*2+12*2+3*5+4*4+13*2+1*5 = 146$ bits.

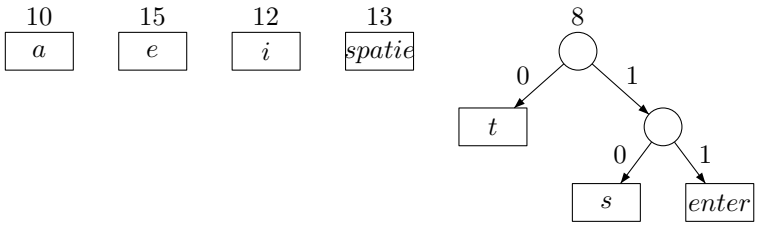
5. De inhoud van de arrays *dist* en *prev* (voor source code, zie de Listing op pg 87 van het dictaat) na iedere waarde van k zijn de tussenstappen. Vier steden, dus er zijn tussenresultaten voor $k = 1, 2, 3, 4$. Als een waarde in *dist* (en *prev*) wijzigt, tonen we dat in vetgedrukt, rood lettertype. Er zijn dus in totaal tien wijzigingen.

De afstanden tussen ieder paar knopen (dwz, de inhoud van *dist*) zijn na iedere waarde van k:

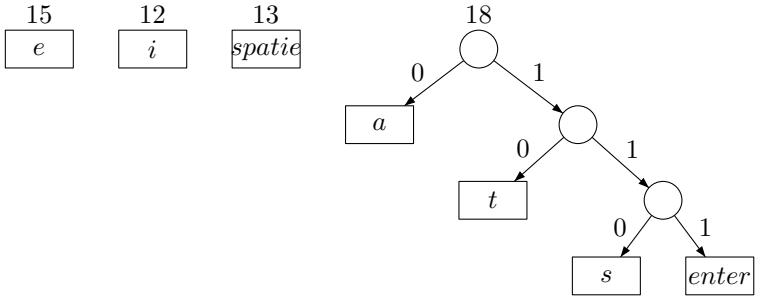
$k=1$
$$\begin{pmatrix} 0 & 3 & 1 & \infty \\ 3 & 0 & 1 & 4 \\ \infty & 1 & 0 & 6 \\ -2 & \mathbf{1} & \mathbf{-1} & 0 \end{pmatrix}$$
$k=2$
$$\begin{pmatrix} 0 & 3 & 1 & \mathbf{7} \\ 3 & 0 & 1 & 4 \\ \mathbf{4} & 1 & 0 & \mathbf{5} \\ -2 & 1 & -1 & 0 \end{pmatrix}$$
$k=3$
$$\begin{pmatrix} 0 & \mathbf{2} & 1 & \mathbf{6} \\ 3 & 0 & 1 & 4 \\ 4 & 1 & 0 & 5 \\ -2 & \mathbf{0} & -1 & 0 \end{pmatrix}$$
$k=4$
$$\begin{pmatrix} 0 & 2 & 1 & 6 \\ \mathbf{2} & 0 & 1 & 4 \\ \mathbf{3} & 1 & 0 & 5 \\ -2 & 0 & -1 & 0 \end{pmatrix}$$

De inhoud van *prev*, waaruit we de route van de kortste paden kunnen afleiden, is:

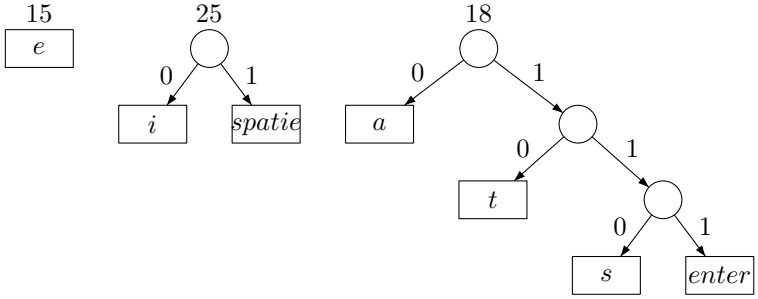
$k=1$
$$\begin{pmatrix} 1 & 1 & 1 & \infty \\ 2 & 2 & 2 & 2 \\ \infty & 3 & 3 & 3 \\ 4 & \mathbf{1} & \mathbf{1} & 4 \end{pmatrix}$$
$k=2$
$$\begin{pmatrix} 1 & 1 & 1 & \mathbf{2} \\ 2 & 2 & 2 & 2 \\ \mathbf{2} & 3 & 3 & \mathbf{2} \\ 4 & 1 & 1 & 4 \end{pmatrix}$$
$k=3$
$$\begin{pmatrix} 1 & \mathbf{3} & 1 & \mathbf{2} \\ 2 & 2 & 2 & 2 \\ 2 & 3 & 3 & 2 \\ 4 & \mathbf{3} & 1 & 4 \end{pmatrix}$$
$k=4$
$$\begin{pmatrix} 1 & 3 & 1 & 2 \\ \mathbf{4} & 2 & 2 & 2 \\ \mathbf{4} & 3 & 3 & 2 \\ 4 & 3 & 1 & 4 \end{pmatrix}$$



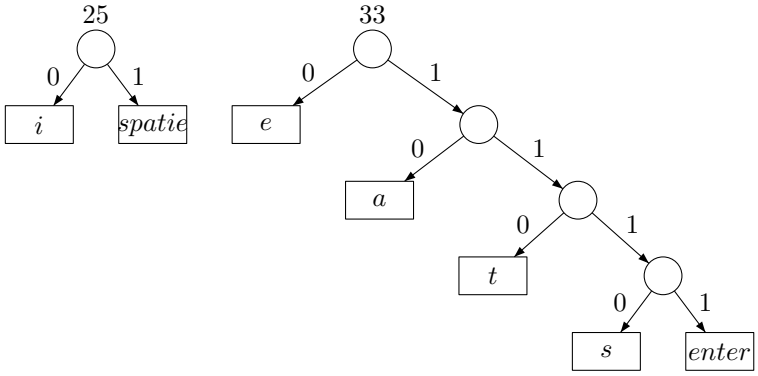
Figuur 4: Huffman stap 3 - na mergen van de twee bomen met frequenties 4



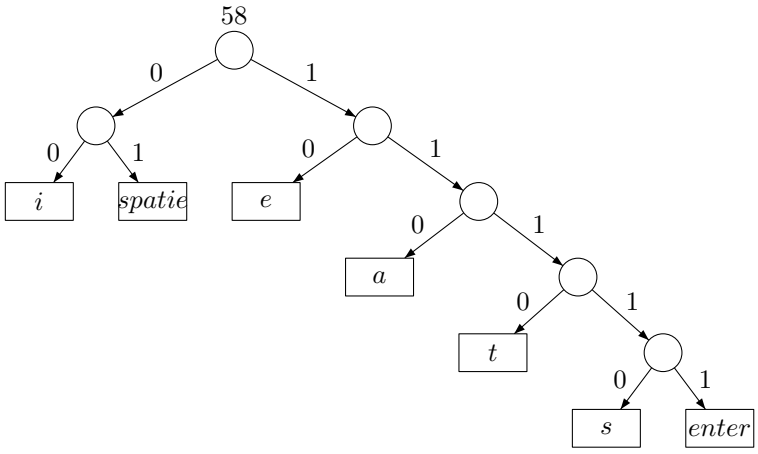
Figuur 5: Huffman stap 4 - na mergen van bomen met frequenties 10 en 8



Figuur 6: Huffman stap 5 - na mergen van bomen met frequenties 12 en 13



Figuur 7: Huffman stap 6 - na mergen van bomen met frequenties 15 en 18



Figuur 8: Huffman stap 7 - eindsituatie, na mergen van bomen met frequenties 25 en 33