Les 7: Bomen als abstracte datatypes

SESSIE 1

Les 7: Bomen als abstracte datatypes

In dit hoofdstuk kijken we naar een aantal voorbeelden die laten zien hoe je voor specifieke toepassingen je eigen abstracties kan definiëren. We laten zien hoe de oplossing zich eenvoudig laat formuleren als we de gepaste data- en procedurele abstracties introduceren.

Overzicht

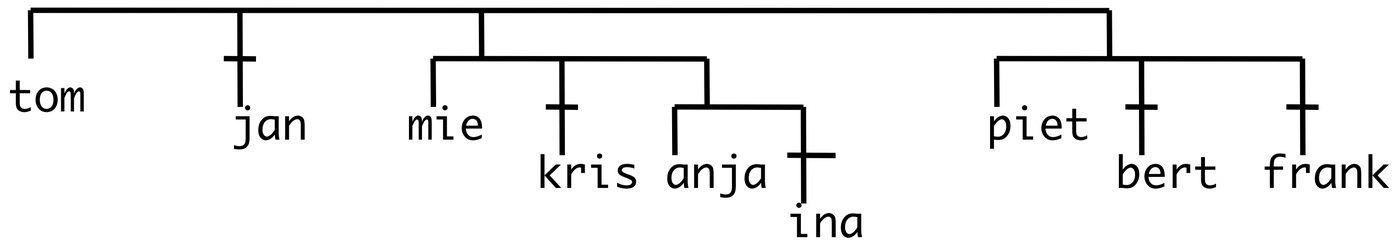
In een eerste voorbeeld kijken we opnieuw naar bomen. We zien bomen nu niet meer als geneste lijsten waar alleen de bladeren van de boom informatie bevatten maar als structuren om allerlei soorten van hiërarchische relaties voor te stellen.

In het tweede voorbeeld schetsen we een probleem nml. het encoderen en decoderen van 'teksten' gebruik makend van een Huffman codering. Daar wordt een specifiek soort boom gebruikt waar bladeren en knopen van de boom verschillend zijn.

Bomen: gezichtspunt 1

```
> family
(tom (jan) (mie (kris) (anja (ina))) (piet (bert) (frank)))
```

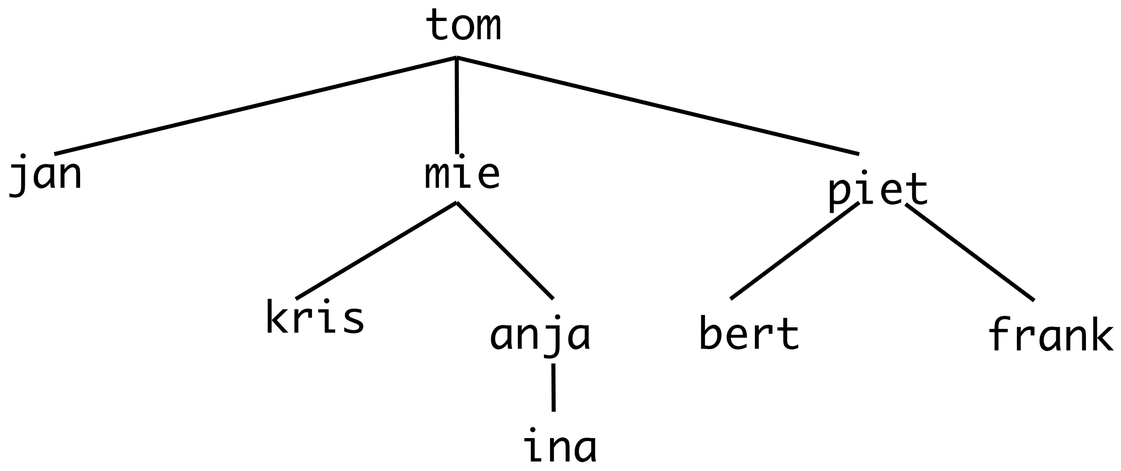
1 2 3 4 elements



Bomen: gezichtspunt 2

```
> family
(tom (jan) (mie (kris) (anja (ina))) (piet (bert) (frank)))
```

parent — subfamilies



Niet alle geneste lijsten zijn familiebomen!

<family> = (<parent> <familie$_1$> <familie$_2$> ... <familie$_n$>)

OK: (tom)
(tom (jan))
(tom (jan) (ann))
(tom (jan (ann (mie))) (frans))
(tom (jan) (ann (mie (ina)) (frans)) (jef))

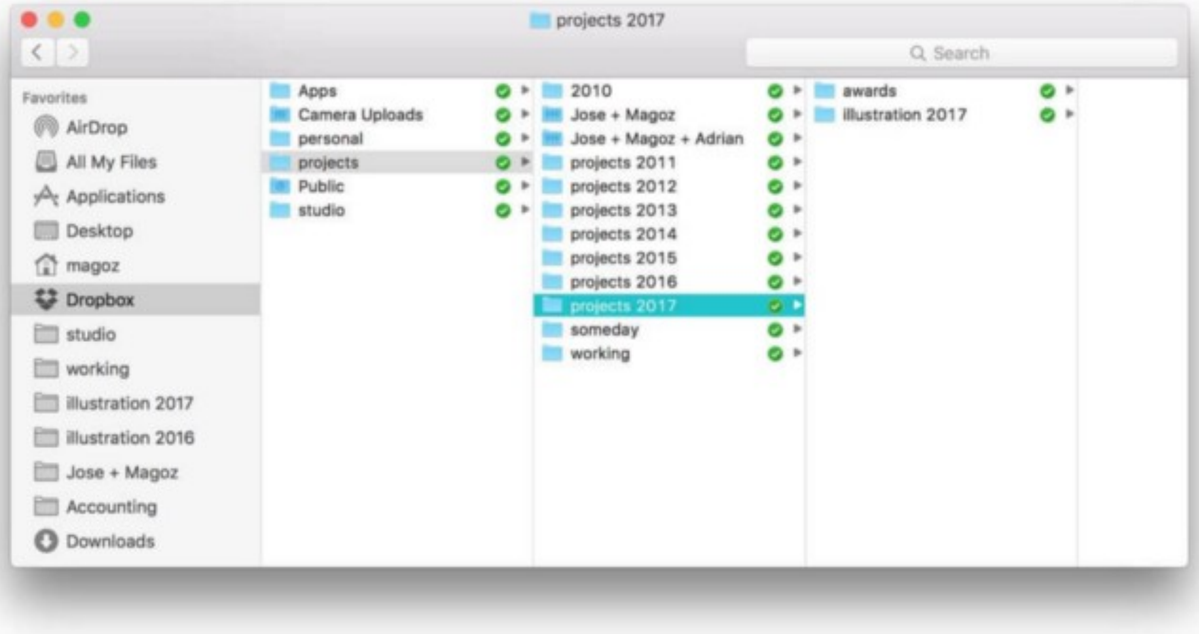
NOT OK: tom
(tom jan)
(tom jan ann)
((tom jan) (ann))
(tom (jan ann))
(tom (jan (ann)) frans)

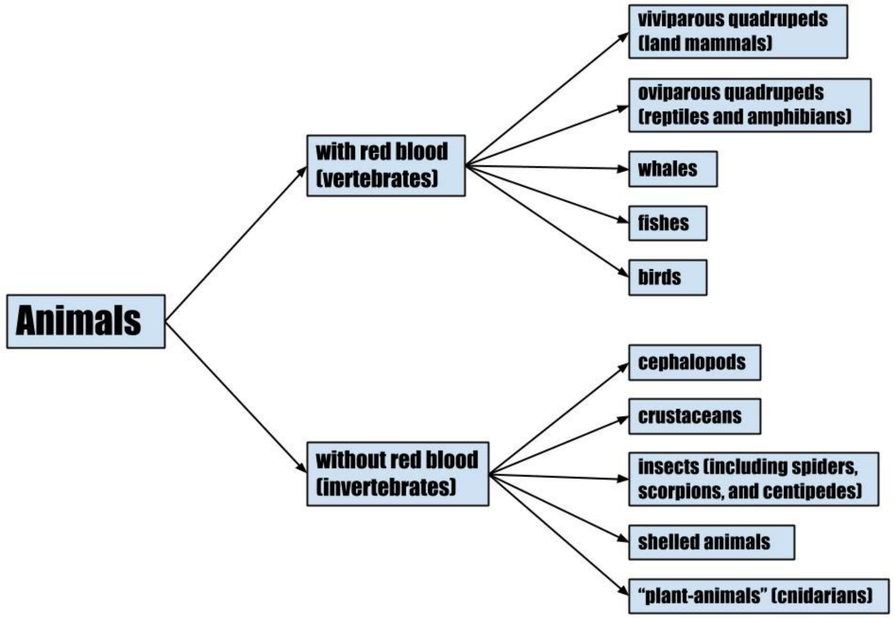
Een abstract data-type familie

```
(define (make-family-tree parent subfams)
   (cons parent subfams))

(define (parent family)
   (car family))

(define (subfams family)
   (cdr family))
```

```
> family
(tom (jan) (mie (kris) (anja (ina))) (piet (bert) (frank)))
> (parent family)
tom
> (subfams family)
((jan) (mie (kris) (anja (ina))) (piet (bert) (frank)))
```

Voorbeelden van hiërarchische structuren





Boom manipuleren als een geneste lijst

```
(define (find? el tree)
  (cond
    ((null? tree) #f)
    ((atom? tree) (eq? tree el))
    (else
      (or (find? el (car tree))
          (find? el (cdr tree))))))
```

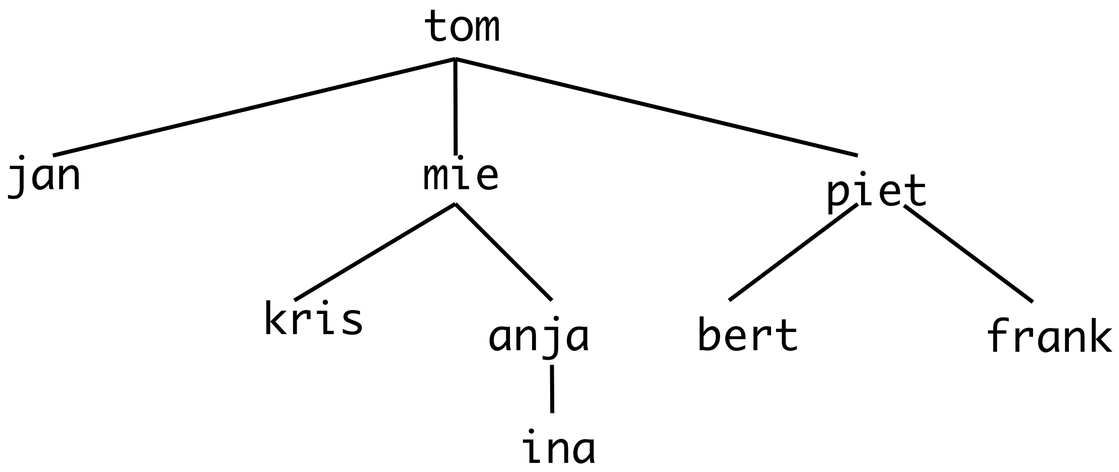
```
> (find? 'jan family)
#t
> (find? 'anja family)
#t
> (find? 'vivi family)
#f
```

```
(define (all tree)
  (cond
    ((null? tree) '())
    ((atom? tree) (list tree))
    (else
      (append (all (car tree))
              (all (cdr tree))))))
```

```
> family
(tom (jan) (mie (kris) (anja (ina))) (piet (bert) (frank)))

> (all family)
(tom jan mie kris anja ina piet bert frank)
```

Boom manipuleren als een familieboom

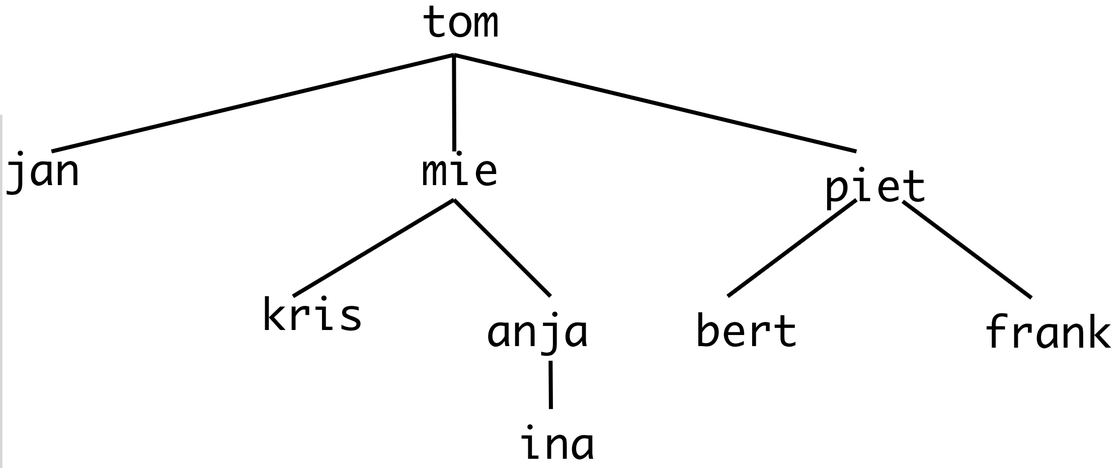
```
(define (find? name family)
  (cond
    ((eq? name (parent family)) #t)
    (else (find-in name (subfams family)))))

(define (find-in? name families)
  (cond
    ((null? families) #f)
    (else (or (find? name (car families))
              (find-in? name (cdr families))))))
```




```
> (find? 'jan family)
#t
> (find? 'anja family)
#t
> (find? 'vivi family)
#f
```

De voorvader-nakomeling relatie controleren

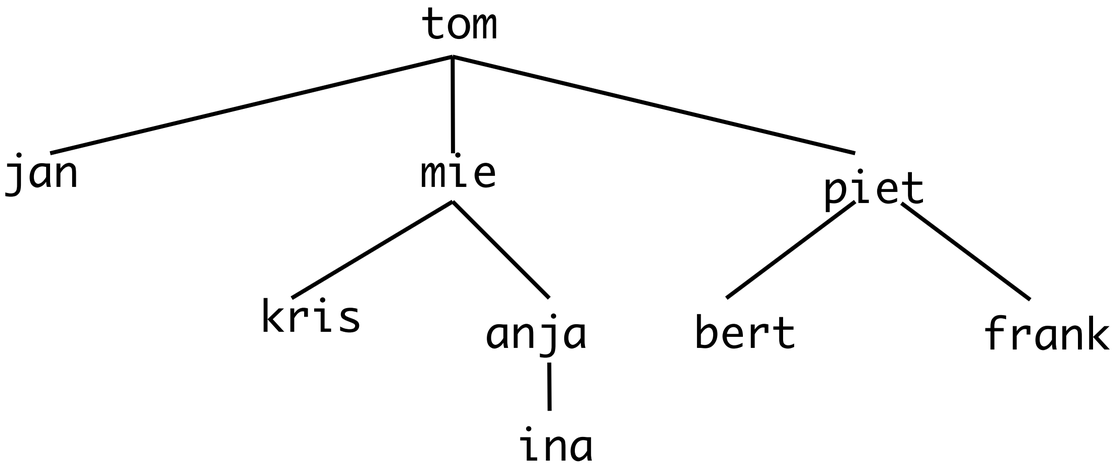
```
(define (offspring? name1 name2 family)
  (cond
    ((eq? name1 (parent family)) (find name2 family))
    (else
      (offspring-in? name1 name2 (subfams family)))))

(define (offspring-in? name1 name2 families)
  (cond
    ((null? families) #f)
    (else
      (or (offspring? name1 name2 (car families))
          (offspring-in? name1 name2 (cdr families))))))
```




```
> (offspring? 'tom 'frank family)
#t
> (offspring? 'piet 'frank family)
#t
> (offspring? 'mie 'frank family)
#f
> (offspring? 'vivi 'frank family)
#f
> (offspring? 'kris 'vivi family)
#f
```

Alle kinderen van een persoon opvragen

```
(define (children name family)
  (cond
    ((eq? name (parent family))
     (map parent (subfams family)))
    (else (children-in name (subfams family)))))

(define (children-in name families)
  (cond
    ((null? families) #f)
    (else
      (or (children name (car families))
          (children-in name (cdr families))))))
```

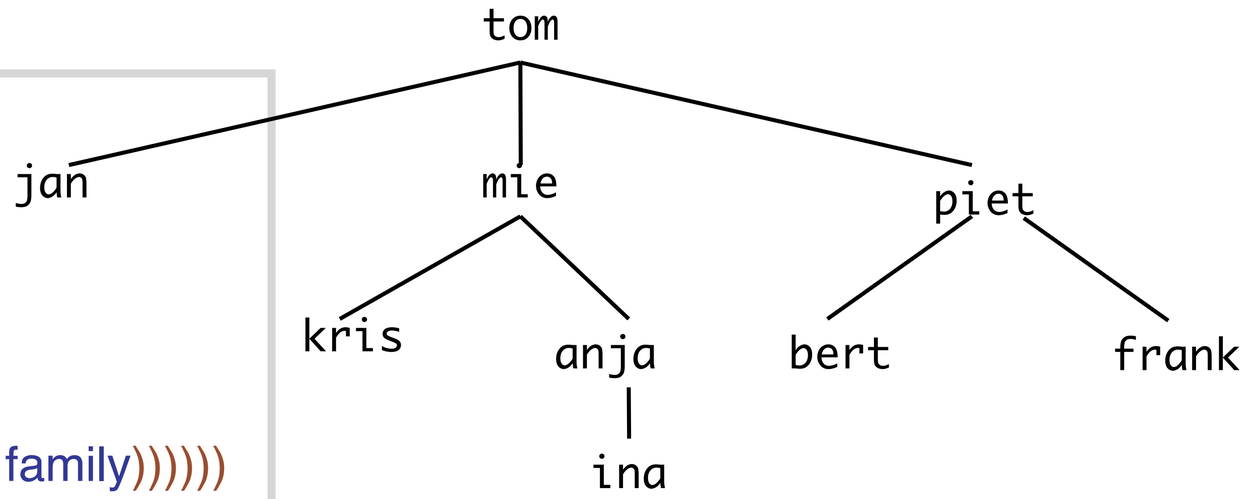



```
> (children 'tom family)
(jan mie piet)
> (children 'mie family)
(kris anja)
> (children 'anja family)
(ina)
> (children 'frank family)
()
> (children 'vivi family)
#f
```

De plaats (diepte) van een persoon in de boom berekenen

je kan niet gewoon +1 doen omdat er ook #f kan terugkomen

```
(define (incr n) (if n (+ 1 n) #f))

(define (generation name family)
  (cond
    ((eq? name (parent family)) 1)
    (else
      (incr (generation-in name (subfams family))))))

(define (generation-in name families)
  (cond
    ((null? families) #f)
    (else
      (or (generation name (car families))
          (generation-in name (cdr families))))))
```

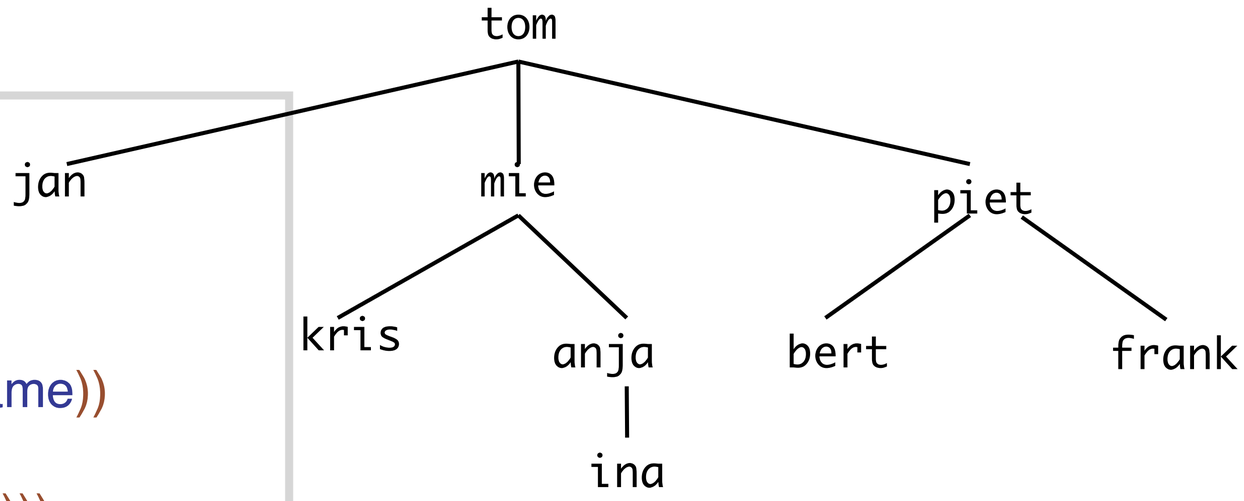



```
> (generation 'tom family)
1
> (generation 'anja family)
3
> (generation 'frank family)
3
> (generation 'vivi family)
#f
```

Het pad van de top naar een persoon opvragen

je kan niet gewoon cons doen omdat er ook #f kan terugkomen

```
(define (add e p) (if p (cons e p) #f))

(define (path name family)
  (cond
    ((eq? name (parent family)) (list name))
    (else (add (parent family)
      (path-in name (subfams family))))))

(define (path-in name families)
  (cond
    ((null? families) #f)
    (else (or (path name (car families))
              (path-in name (cdr families))))))
```




```
> (path 'tom family)
(tom)
> (path 'anja family)
(tom mie anja)
> (path 'frank family)
(tom piet frank)
> (path 'vivi family)
#f
```

Les 7: Bomen als abstracte datatypes

SESSIE 2

Huffman coding (1)

Fixed length representation of character set

Char	Code
a	000
b	001
c	010
d	011
e	100
f	101
g	110
h	111

3 bits per char

text n chars -> 3 n bits

AHABADBAACBEDAAAB

000111000001000011001000000010001100011000000000001

Huffman coding (2)

Variable length representation of character set

Char	Freq	Code
a	40	1
b	20	001
c	10	011
d	8	0001
e	8	0101
f	6	0100
g	4	00001
h	4	00000

1-5 bits per char

text n chars < 3n bits

AHABADBAACBEDAAAB

100000100110001001110110010101000111001

Letterfrekwentie voor Engels

Letter	Relative frequency in the English language	
	Texts	Dictionaries
a	8.2%	7.8%
b	1.5%	2%
c	2.8%	4%
d	4.3%	3.8%
e	13%	11%
f	2.2%	1.4%
g	2%	3%
h	6.1%	2.3%
i	7%	8.6%
j	0.15%	0.21%
k	0.77%	0.97%
l	4%	5.3%
m	2.4%	2.7%
n	6.7%	7.2%
o	7.5%	6.1%
p	1.9%	2.8%
q	0.095%	0.19%
r	6%	7.3%
s	6.3%	8.7%
t	9.1%	6.7%
u	2.8%	3.3%
v	0.98%	1%
w	2.4%	0.91%
x	0.15%	0.27%
y	2%	1.6%
z	0.074%	0.44%

13% van de gebruikte letters in een corpus teksten is e
11% van de gebruikte letters in een woordenboek is e

q is niet populair

verschilt van taal tot taal

Nederlands **e** 18.91%

Portugees **a** 14.63%

https://en.wikipedia.org/wiki/Letter_frequency#Relative_frequencies_of_letters_in_other_languages

Huffman coding:
de prefix eigenschap

No character has a representation that is the prefix of another character's representation

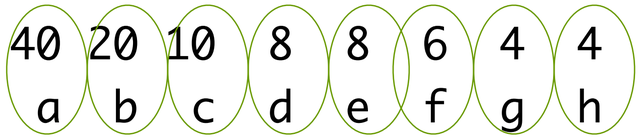
!!! without this property decoding is impossible

Char	Freq	Code
a	40	1
b	20	001
c	10	011
d	8	0001
e	8	0101
f	6	0100
g	4	00001
h	4	00000

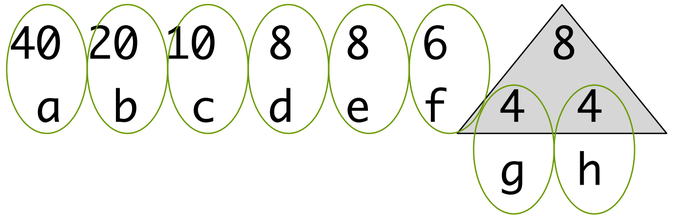
100000100110001001110110010101000111100 1

Huffman coding: het algoritme (1)

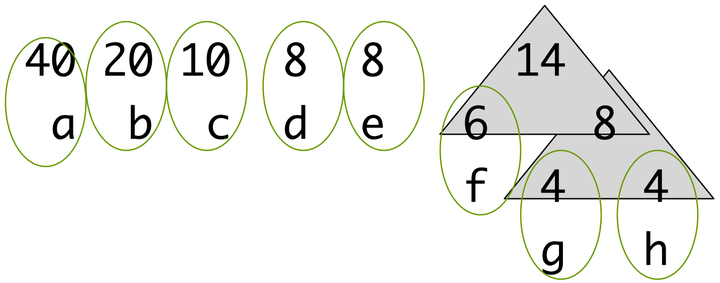
1



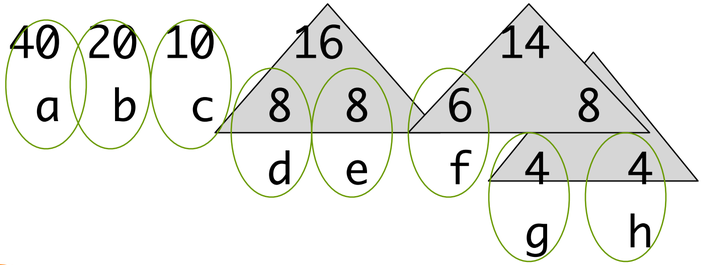
2



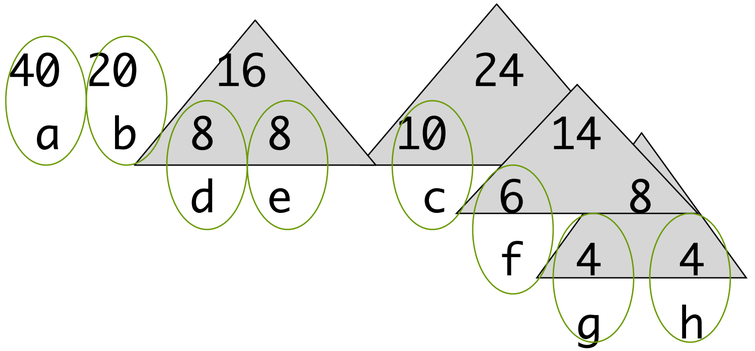
3



4

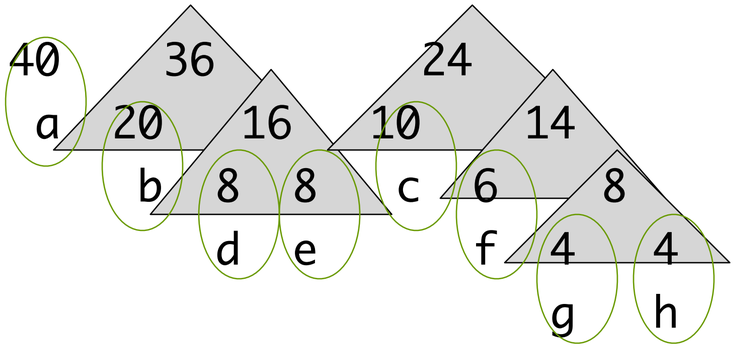


5

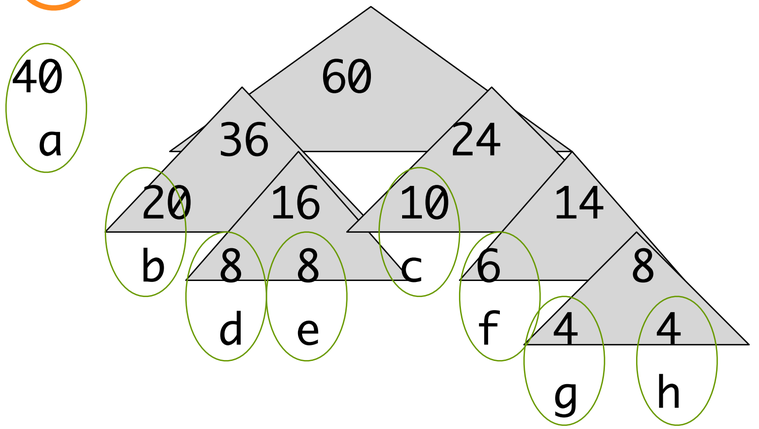


Huffman coding: het algoritme (2)

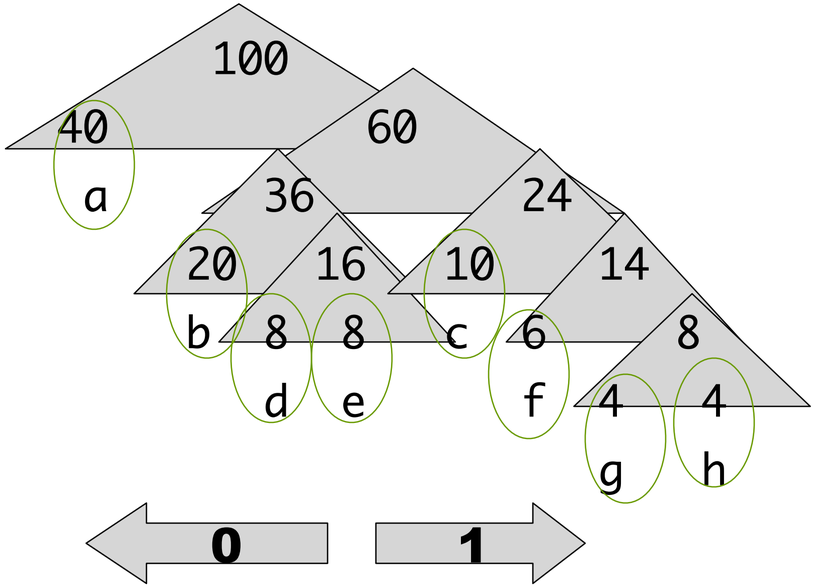
6



7



8

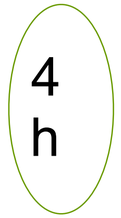


a	**0**
b	**100**
c	**110**
d	**1010**
e	**1011**
f	**1110**
g	**11110**
h	**11111**

Huffman coding data-abstractie: bladeren

```
(define (make-leaf symbol weight)
   (list 'leaf symbol weight))

(define (leaf? object)
   (eq? (car object) 'leaf))

(define (leaf-symbol leaf)
   (cadr leaf))

(define (leaf-weight leaf)
   (caddr leaf))
```

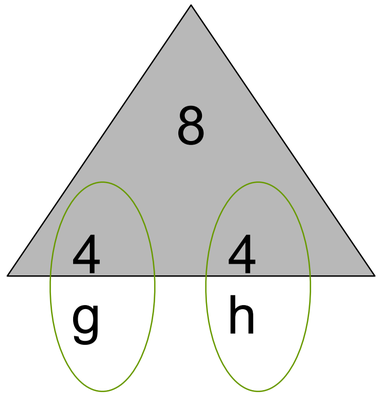
om een blad te herkennen wordt het symbool 'leaf' mee opgeslagen



(leaf h 4)

Huffman coding data-abstractie: knopen

```
(define (make-node left right)
   (list left
         (+ (weight left) (weight right))
         right))

(define (node-weight node)
   (cadr node))

(define (left node)
   (car node))

(define (right node)
   (caddr node))
```



((leaf g 4) 8 (leaf h 4))

kan zowel op knopen als op bladeren werken

```
(define (weight tree)
   (if (leaf? tree)
       (leaf-weight tree)
       (node-weight tree)))
```

Huffman coding data-abstractie: een geordende lijst van bomen

```
(define (insert el seq)
  (cond
    ((null? seq) (list el))
    ((< (weight el) (weight (car seq)) (cons el seq))
    (else (cons (car seq) (insert el (cdr seq))))))
```

als je de lijst met deze insert opbouwt zullen de 2 'lichtste' bomen vooraan staan

Huffman coding: de boom opbouwen

```
(define (make-huffman-tree pairs)
  (succ-merge (make-leafs pairs)))

(define (make-leafs pairs)
  (cond
    ((null? pairs) '())
    (else
      (let ((pair (car pairs)))
        (insert (make-leaf (car pair) (cadr pair))
                (make-leafs (cdr pairs)))))))

(define (succ-merge trees)
  (cond
    ((null? trees) 'error)
    ((null? (cdr trees)) (car trees))
    (else (succ-merge
            (insert (make-node (car trees) (cadr trees))
                    (cddr trees))))))
```

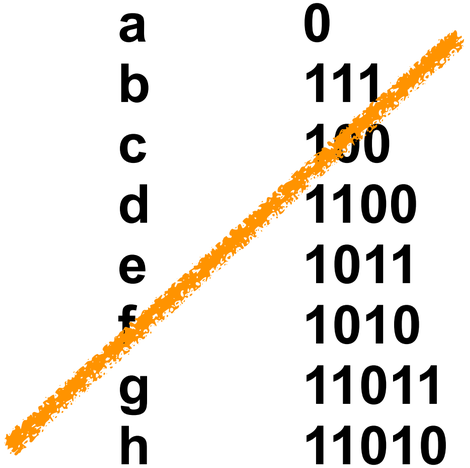
opstarten met lijst van (symbool frequentie) combinaties

van alle (symbool frequentie) combinaties een initiële sequentie van bladeren

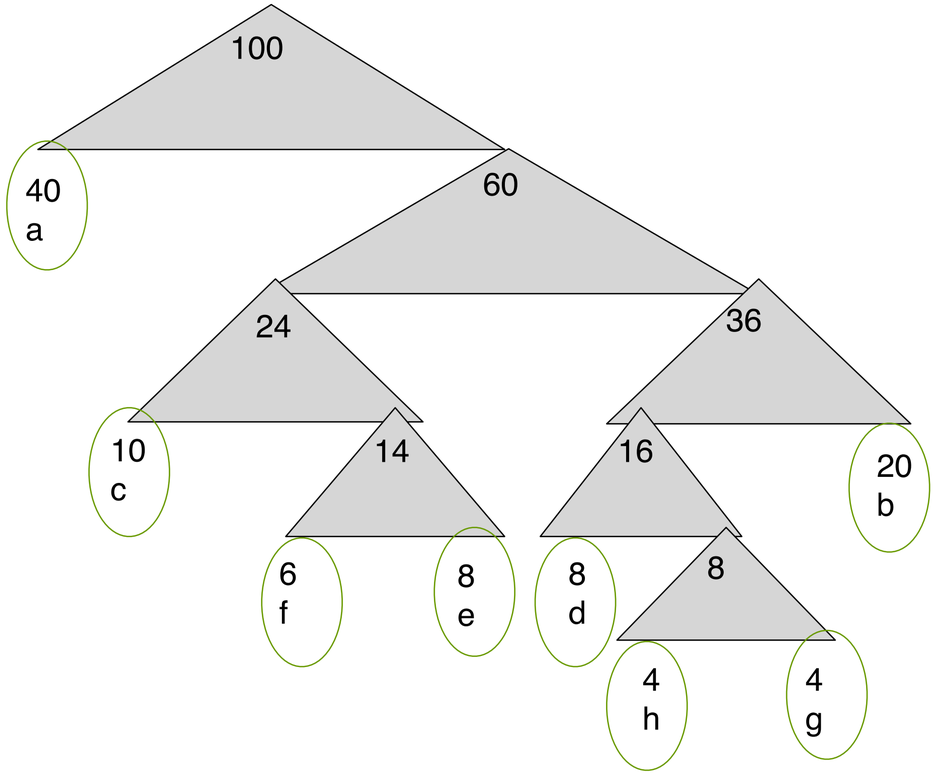
de twee lichtste bomen combineren tot er maar 1 boom meer overblijft

Huffman coding: de boom omvormen naar een tabel is overbodig

```
> (define test (make-huffman-tree '((a 40) (b 20) (c 10) (d 8) (e 8) (f 6) (g 4) (h 4))))

> test
((leaf a 40)
 100
 (((leaf c 10) 24 ((leaf f 6) 14 (leaf e 8)))
  60
  (((leaf d 8) 16 ((leaf h 4) 8 (leaf g 4)))
   36
   (leaf b 20))))
```



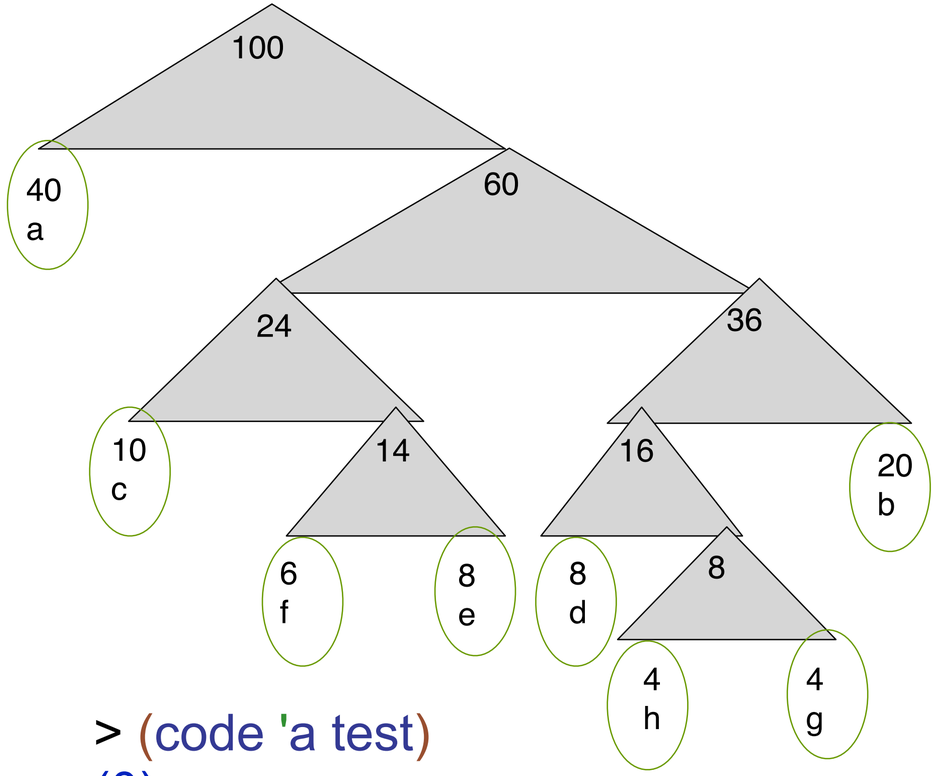
a	**0**
b	**111**
c	**100**
d	**1100**
e	**1011**
f	**1010**
g	**11011**
h	**11010**



Huffman coding: een bericht coderen

zoek een pad van de top van de boom naar de letter

```
(define (code symbol tree)
  (cond ((null? tree) #f)
        ((leaf? tree)
         (if (eq? symbol (leaf-symbol tree))
             '()
             #f))
        (else (or (add 0 (code symbol (left tree)))
                  (add 1 (code symbol (right tree)))))))

(define (add number lst) (if lst (cons number lst) #f))

(define (encode message tree)
  (cond
    ((null? message) '())
    (else (append (code (car message) tree)
                  (encode (cdr message) tree)))))
```



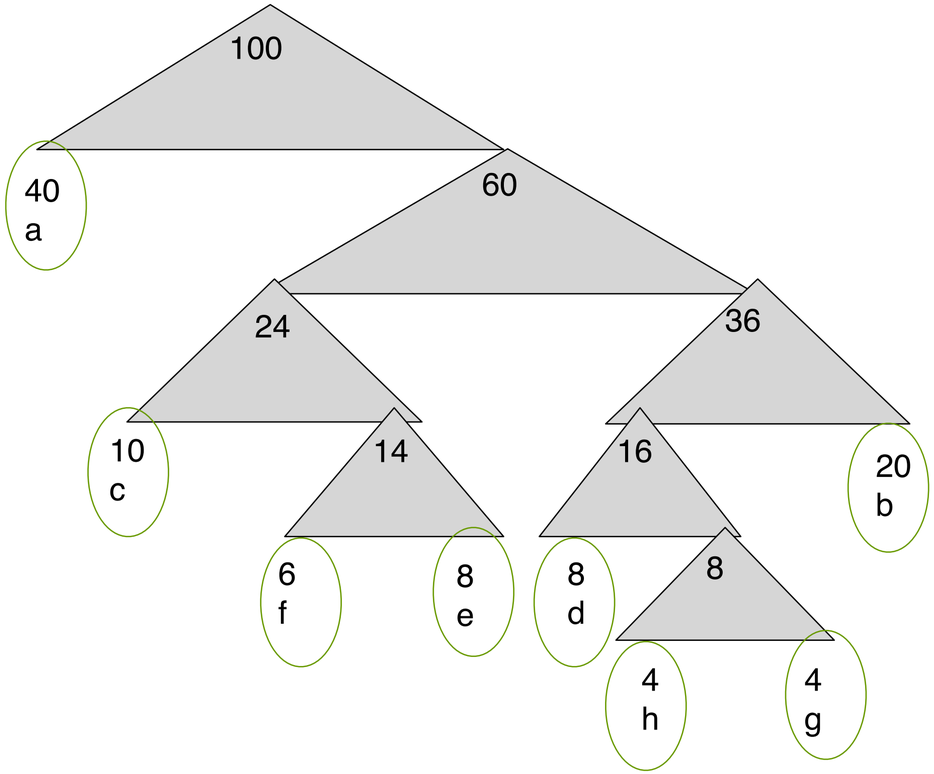

```
> (code 'a test)
(0)
> (code 'b test)
(1 1 1)
> (code 'd test)
(1 1 0 0)
>(encode '(b a d) test)
(1 1 1 0 1 1 0 0)
```

Huffman coding: een bitstring decoderen

loop links/ rechts tot blad bereikt wordt; 'herneem' bovenaan voor volgende letter

top van boom meenemen om te kunnen 'hernemen'

```
(define (decode bits tree)
  (decode-aux bits tree tree))

(define (decode-aux bits cur tree)
  (cond
    ((null? bits) '())
    ((= 0 (car bits))
     (cond
       ((leaf? (left cur))
        (cons (leaf-symbol (left cur))
              (decode-aux (cdr bits) tree tree)))
       (else (decode-aux (cdr bits) (left cur) tree))))
    ((= 1 (car bits))
     (cond
       ((leaf? (right cur))
        (cons (leaf-symbol (right cur))
              (decode-aux (cdr bits) tree tree)))
       (else (decode-aux (cdr bits) (right cur) tree))))))
```




```
> (define w1 '(1 1 1 0 1 1 0 0))
> (define w2 '(1 1 1 1 0 1 1 1 1 0 0))
> (decode w1 test)
(b a d)
> (decode w2 test)
(b e d)
```

Les 8: Generische operatoren