Universiteit Utrecht

[Faculteit **Bètawetenschappen**
Informatica]

Memofuncties

Stefan Holdermans
Doaitse Swierstra

October 12, 2010

De Fibonaccireeks

Leonardo van Pisa ($\pm$1170 – $\pm$1250):

$$F_n = \begin{cases} n & \text{if } n < 2, \\ F_{n-2} + F_{n-1} & \text{if } n \geqslant 2. \end{cases}$$

$$\begin{array}{lll} fib :: Integer \to Integer & & \\ fib & 0 & = 0 \\ fib & 1 & = 1 \\ fib & n & = fib\ (n-2) + fib\ (n-1) \end{array}$$

Interactieve sessie: tijd- en geheugenverbruik

GHCi met `:set +s`:

```
Main >
```

Interactieve sessie: tijd- en geheugenverbruik

GHCi met `:set +s`:

```
Main> fib 10
```

Interactieve sessie: tijd- en geheugenverbruik

GHCi met `:set +s`:

```
Main> fib 10
55
0.02 secs, 3043752 bytes
Main>
```

Interactieve sessie: tijd- en geheugenverbruik

GHCi met `:set +s`:

```
Main> fib 10
55
0.02 secs, 3043752 bytes
Main> fib 20
```

Interactieve sessie: tijd- en geheugenverbruik

GHCi met `:set +s`:

```
Main> fib 10
55
0.02 secs, 3043752 bytes
Main> fib 20
6765
0.06 secs, 3133924 bytes
Main>
```

Interactieve sessie: tijd- en geheugenverbruik

GHCi met `:set +s`:

```
Main> fib 10
55
0.02 secs, 3043752 bytes
Main> fib 20
6765
0.06 secs, 3133924 bytes
Main> fib 25
```

Interactieve sessie: tijd- en geheugenverbruik

GHCi met `:set +s`:

```
Main> fib 10
55
0.02 secs, 3043752 bytes
Main> fib 20
6765
0.06 secs, 3133924 bytes
Main> fib 25
75025
0.63 secs, 34743476 bytes
Main>
```

Interactieve sessie: tijd- en geheugenverbruik

GHCi met `:set +s`:

```
Main> fib 10
55
0.02 secs, 3043752 bytes
Main> fib 20
6765
0.06 secs, 3133924 bytes
Main> fib 25
75025
0.63 secs, 34743476 bytes
Main> fib 30
```

Interactieve sessie: tijd- en geheugenverbruik

GHCi met `:set +s`:

```
Main> fib 10
55
0.02 secs, 3043752 bytes
Main> fib 20
6765
0.06 secs, 3133924 bytes
Main> fib 25
75025
0.63 secs, 34743476 bytes
Main> fib 30
832040
6.80 secs, 383178156 bytes
```

Interactieve sessie: aantal rekenstappen

Hugs (`http://haskell.org/hugs`):

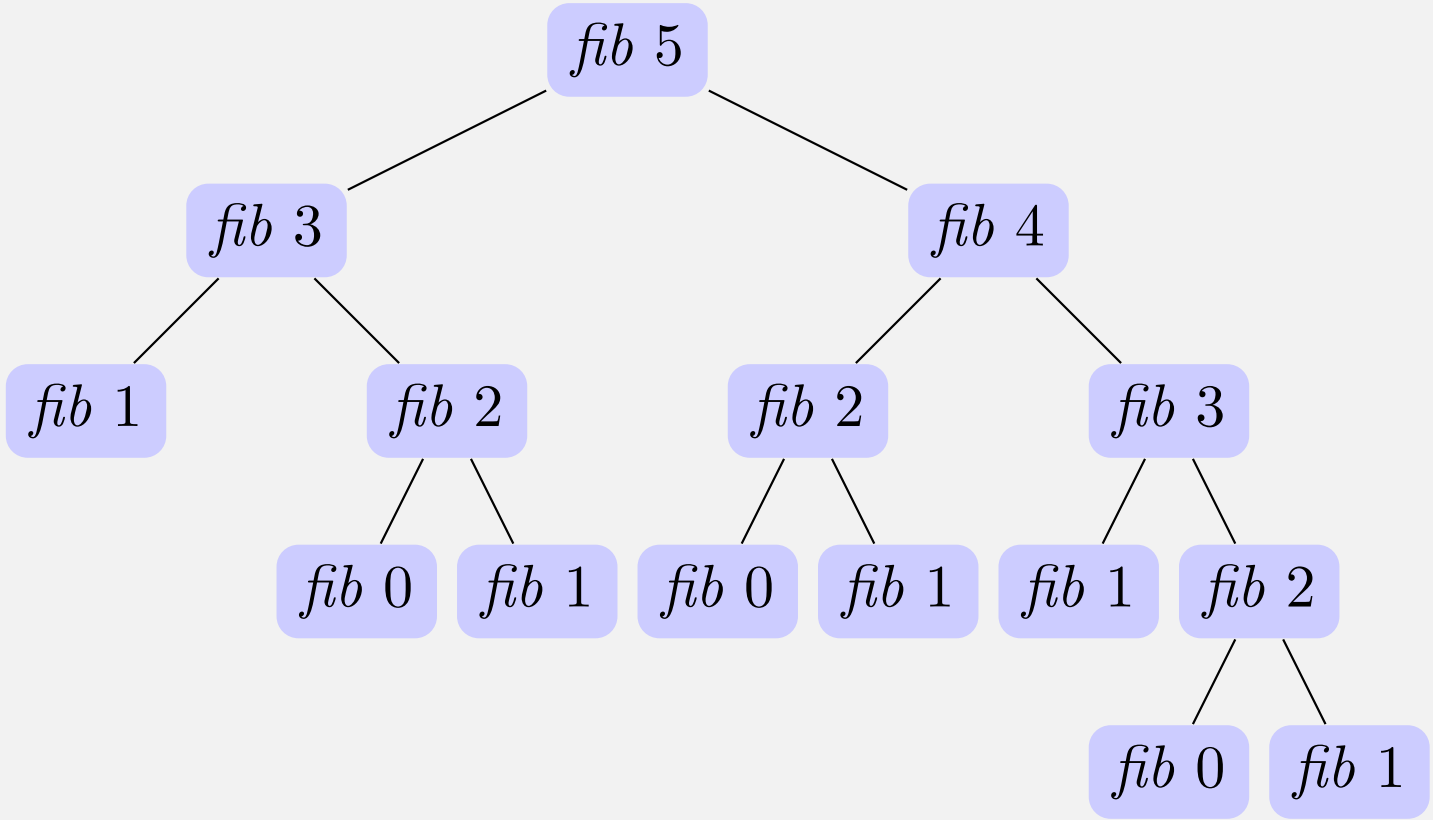
```
Main> fib 10
55

Main> fib 20
6765

Main> fib 25
75025

Main> fib 30
832040
```

Interactieve sessie: aantal rekenstappen

Hugs (`http://haskell.org/hugs`) met `+s`:

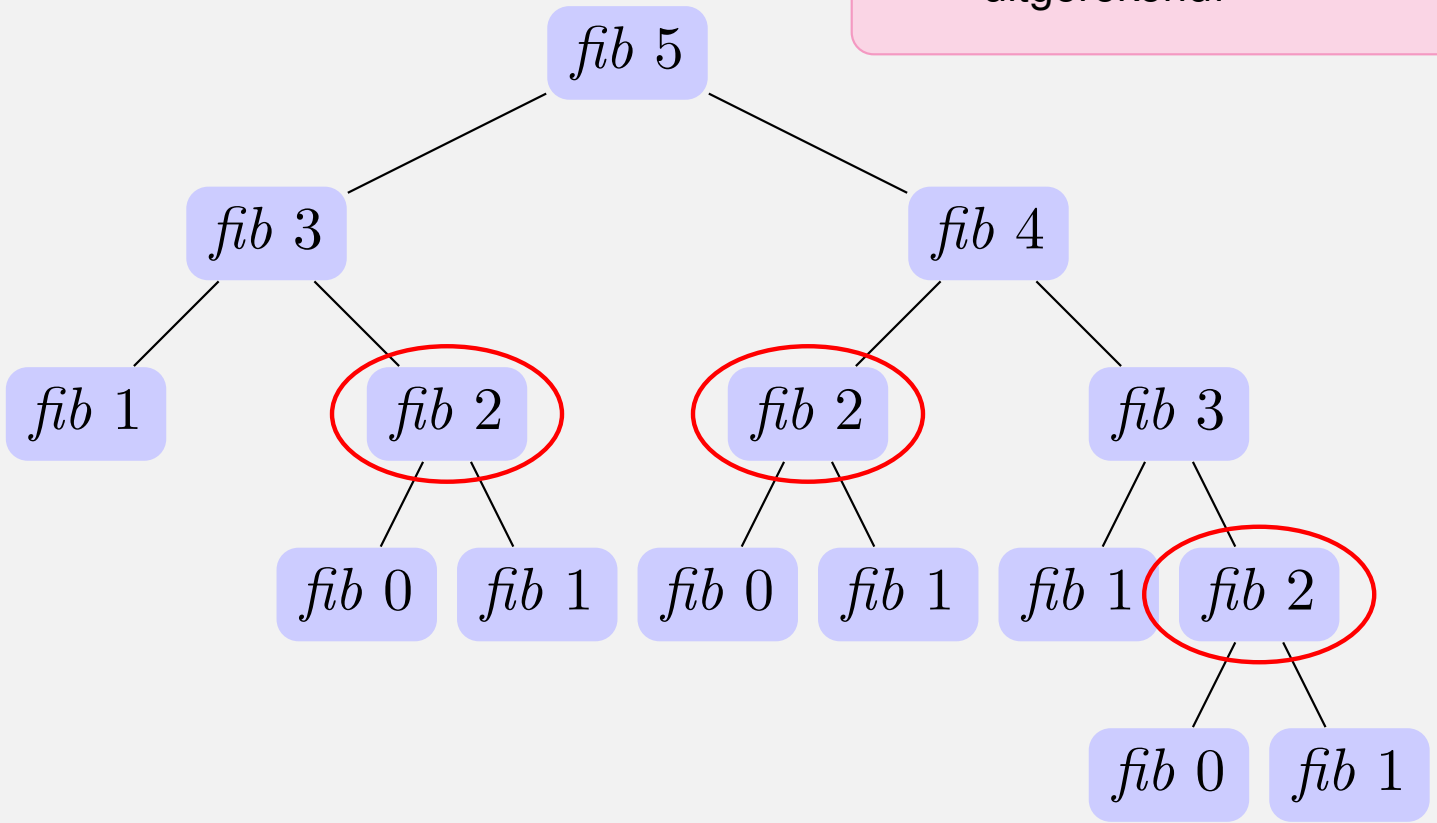
```
Main> fib 10
55
3177 reductions, 5054 cells
Main> fib 20
6765
390861 reductions, 622695 cells
Main> fib 25
75025
4334725 reductions, 6905874 cells, 6 garbage collections
Main> fib 30
832040
48072847 reductions, 76587387 cells, 77 garbage collections
```

Aanroepboom



Aanroepboom

- $fib\ 2$ wordt drie keer uitgerekend!



Aantal recursieve aanroepen

Aantal recursieve aanroepen voor $fib\ n$:

waarde voor n	aantal aanroepen van fib
5	15
10	177
15	1973
20	21891
25	242785
30	2692537

Lokale memoïsatie

Idee: ‘onthoud’ de resultaten van functieaanroepen voor een reeks argumenten.

Lokale memoïsatie

Idee: ‘onthoud’ de resultaten van functieaanroepen voor een reeks argumenten.

$$
\begin{aligned}
&\mathit{fib} :: \mathit{Integer} \to \mathit{Integer} \\
&\mathit{fib} \quad n \qquad\quad = \mathit{fibs}\,!\,n \\
&\quad \textbf{where} \\
&\qquad \mathit{fibs} = \mathit{listArray}\ (0, n)\ \$ \\
&\qquad\quad 0 : 1 : [\mathit{fibs}\,!\,(k-2) + \mathit{fibs}\,!\,(k-1) \mid k \leftarrow [2\,..\,n]]
\end{aligned}
$$

Lokale memoïsatie

Idee: ‘onthoud’ de resultaten van functieaanroepen voor een reeks argumenten.

$$
\begin{array}{l}
\mathit{fib} :: \mathit{Integer} \to \mathit{Integer} \\
\mathit{fib} \quad n \qquad\quad = \mathit{fibs} \,!\, n \\
\quad \textbf{where} \\
\qquad \mathit{fibs} = \mathit{listArray}\ (0, n)\ \$ \\
\qquad\quad 0 : 1 : [\mathit{fibs} \,!\, (k-2) + \mathit{fibs} \,!\, (k-1) \mid k \leftarrow [2\,..\,n]]
\end{array}
$$

☞ Voor opeenvolgende aanroepen van fib wordt telkens een nieuwe array fibs opgebouwd.

Globale memoïsatie

Een globale memofunctie

- onthoudt ook de resultaten van voorafgaande aanroepen,
- onthoudt resultaten voor alle argumenten.

Globale memoïsatie

Een globale memofunctie

- onthoudt ook de resultaten van voorafgaande aanroepen,
- onthoudt resultaten voor alle argumenten.

Doel: een bibliotheekje om eenvoudig memoïserende versies van alle functies van de natuurlijke getallen te maken.

Dekpuntcombinator

Een dekpunt (Eng.: *fixed point*) van een functie f is een waarde x waarvoor $f\ x = x$.

Dekpuntcombinator

Een dekpunt (Eng.: *fixed point*) van een functie f is een waarde x waarvoor $f\ x = x$.

Een dekpuntcombinator is een hogere-ordefunctie die dekpunten van andere functies 'berekent':

$$
\begin{aligned}
&\mathit{fix} :: (\alpha \to \alpha) \to \alpha \\
&\mathit{fix}\quad f \qquad\quad = f\ (\mathit{fix}\ f)
\end{aligned}
$$

Expliciete recursie

Met fix kan het gebruik van recursie expliciet gemaakt worden.

Expliciete recursie

Met fix kan het gebruik van recursie expliciet gemaakt worden.

Bijvoorbeeld:

$$
\begin{array}{lll}
fac :: Integer \to Integer & & \\
fac & 0 & = 1 \\
fac & n & = n * fac\ (n-1)
\end{array}
$$

Expliciete recursie

Met fix kan het gebruik van recursie expliciet gemaakt worden.

Bijvoorbeeld:

$$
\begin{array}{lll}
fac :: Integer \to Integer & & \\
fac & 0 & = 1 \\
fac & n & = n * fac\ (n-1)
\end{array}
$$

kan met fix geschreven worden als

$$
\begin{array}{l}
fac :: Integer \to Integer \\
fac = fix\ fac' \\
\quad \textbf{where} \\
\qquad fac'\ f\ 0 = 1 \\
\qquad fac'\ f\ n = n * f\ (n-1)
\end{array}
$$

Expliciete recursie

Met fix kan het gebruik van recursie expliciet gemaakt worden.

Bijvoorbeeld:

$$
\begin{array}{lll}
fac :: Integer \to Integer & & \\
fac & 0 & = 1 \\
fac & n & = n * fac\ (n-1)
\end{array}
$$

kan met fix geschreven worden als

Idee: introduceer een extra parameter voor de recursieve aanroep.

$$
\begin{array}{l}
fac :: Integer \to Integer \\
fac = fix\ fac' \\
\quad \textbf{where} \\
\qquad fac'\ f\ 0 = 1 \\
\qquad fac'\ f\ n = n * f\ (n-1)
\end{array}
$$

Expliciete recursie

Met fix kan het gebruik van recursie expliciet gemaakt worden.

Bijvoorbeeld:

$$
\begin{array}{lll}
fac :: Integer \to Integer & & \\
fac & 0 & = 1 \\
fac & n & = n * fac\ (n-1)
\end{array}
$$

kan met fix geschreven worden als

Idee: introduceer een extra parameter voor de recursieve aanroep.

$$
\begin{array}{l}
fac :: Integer \to Integer \\
fac = fix\ fac' \\
\quad \textbf{where} \\
\qquad fac'\ f\ 0 = 1 \\
\qquad fac'\ f\ n = n * f\ (n-1)
\end{array}
$$

☞ $fac' :: (Integer \to Integer) \to Integer \to Integer$.

Expliciete recursie: voorbeeld

$$
\begin{aligned}
& fac\ 3 \\
= & \\
& fix\ fac'\ 3 \\
= & \\
& fac'\ (fix\ fac')\ 3 \\
= & \\
& 3 * fix\ fac'\ (3-1) \\
= & \\
& 3 * fix\ fac'\ 2 \\
= & \\
& 3 * fac'\ (fix\ fac')\ 2 \\
= & \\
& 3 * (2 * fix\ fac'\ (2-1)) \\
= & \\
& 3 * (2 * fix\ fac'\ 1)
\end{aligned}
$$

$$
\begin{aligned}
& 6 \\
= & \\
& 3 * 2 \\
= & \\
& 3 * (2 * 1) \\
= & \\
& 3 * (2 * (1 * 1)) \\
= & \\
& 3 * (2 * (1 * fix\ fac'\ 0)) \\
= & \\
& 3 * (2 * (1 * fix\ fac'\ (1-1))) \\
= & \\
& 3 * (2 * fac'\ (fix\ fac')\ 1) \\
= & \\
& 3 * (2 * fix\ fac'\ 1)
\end{aligned}
$$

Nogmaals de Fibonaccireeks

Fibonaccifunctie met expliciete recursie:

$$
\begin{aligned}
&\mathit{fib} :: \mathit{Integer} \to \mathit{Integer} \\
&\mathit{fib} = \mathit{fix}\ \mathit{fib}' \\
&\quad \textbf{where} \\
&\qquad \mathit{fib}'\ f\ 0 = 0 \\
&\qquad \mathit{fib}'\ f\ 1 = 1 \\
&\qquad \mathit{fib}'\ f\ n = f\ (n-2) + f\ (n-1)
\end{aligned}
$$

Nogmaals de Fibonaccireeks

Fibonaccifunctie met expliciete recursie:

$$
\begin{aligned}
&\mathit{fib} :: \mathit{Integer} \to \mathit{Integer} \\
&\mathit{fib} = \mathit{fix}\ \mathit{fib}' \\
&\quad \textbf{where} \\
&\qquad \mathit{fib}'\ f\ 0 = 0 \\
&\qquad \mathit{fib}'\ f\ 1 = 1 \\
&\qquad \mathit{fib}'\ f\ n = f\ (n-2) + f\ (n-1)
\end{aligned}
$$

Idee: vervang fix door een memoïserende dekpuntcombinator.

Bibliotheek voor memofuncties: aanpak

Kies een (geparameteriseerd) datatype $Memo$ voor memotabellen.

Bibliotheek voor memofuncties: aanpak

Kies een (geparameteriseerd) datatype $Memo$ voor memotabellen.

Definieer functies $tabulate$ en $apply$,

$$\begin{aligned} tabulate &:: (Integer \to \alpha) \to Memo\ \alpha \\ apply &:: Memo\ \alpha \to Integer \to \alpha \end{aligned}$$

zodat:

- $tabulate\ f$ een (lazy) memotabel met alle functiewaarden van f oplevert en
- $apply\ mem\ n$ de waarde voor n in mem oplevert.

Bibliotheek voor memofuncties: aanpak

Kies een (geparameteriseerd) datatype $Memo$ voor memotabellen.

Definieer functies $tabulate$ en $apply$,

$$\begin{aligned} tabulate &:: (Integer \to \alpha) \to Memo\ \alpha \\ apply &:: Memo\ \alpha \to Integer \to \alpha \end{aligned}$$

zodat:

- $tabulate\ f$ een (lazy) memotabel met alle functiewaarden van f oplevert en
- $apply\ mem\ n$ de waarde voor n in mem oplevert.

Definieer een dekpuntcombinator $memo$ in termen van $tabulate$ en $apply$.

Memolijsten

In eerste instantie zullen we memotabellen representeren m.b.v. oneindige lijsten:

$$\textbf{type}\ Memo\ \alpha = [\alpha]$$

Memolijsten

In eerste instantie zullen we memotabellen representeren m.b.v. oneindige lijsten:

$$\textbf{type}\ Memo\ \alpha = [\alpha]$$

$$\begin{aligned} &tabulate :: (Integer \to \alpha) \to Memo\ \alpha \\ &tabulate \quad f \qquad\qquad\quad = map\ f\ [0..] \end{aligned}$$

Memolijsten

In eerste instantie zullen we memotabellen representeren m.b.v. oneindige lijsten:

$$\mathbf{type}\ Memo\ \alpha = [\alpha]$$

$$\begin{array}{lll} tabulate :: (Integer \to \alpha) \to Memo\ \alpha & & \\ tabulate & f & = map\ f\ [0\,..] \end{array}$$

$$\begin{array}{llll} apply :: Memo\ \alpha \to Integer \to \alpha & & & \\ apply & (x : \_) & 0 & = x \\ apply & (\_ : xs) & n & = apply\ xs\ (n-1) \end{array}$$

Memocombinator

$$
\begin{aligned}
&memo :: ((Integer \to \alpha) \to Integer \to \alpha) \to Integer \to \alpha \\
&memo \quad f' \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad = f \\
&\quad \textbf{where} \\
&\qquad f = apply\ (tabulate\ (f'\ f))
\end{aligned}
$$

Memocombinator

$$
\begin{aligned}
&memo :: ((Integer \to \alpha) \to Integer \to \alpha) \to Integer \to \alpha \\
&memo \quad f' \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad = f \\
&\quad \textbf{where} \\
&\qquad f = apply\ (tabulate\ (f'\ f))
\end{aligned}
$$

- De combinator levert een dekpunt f van f'.

Memocombinator

$$
\begin{aligned}
&memo :: ((Integer \to \alpha) \to Integer \to \alpha) \to Integer \to \alpha \\
&memo \quad f' \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad = f \\
&\quad \textbf{where} \\
&\qquad f = apply\ (tabulate\ (f'\ f))
\end{aligned}
$$

- De combinator levert een dekpunt f van f'.
- De functie f haalt zijn resultaat uit een memotabel $tabulate\ (f'\ f)$.

Memocombinator

$$
\begin{aligned}
&memo :: ((Integer \to \alpha) \to Integer \to \alpha) \to Integer \to \alpha \\
&memo \quad f' \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad = f \\
&\quad \textbf{where} \\
&\qquad f = apply\ (tabulate\ (f'\ f))
\end{aligned}
$$

- De combinator levert een dekpunt f van f'.
- De functie f haalt zijn resultaat uit een memotabel $tabulate\ (f'\ f)$.
- Elk element in de tabel wordt berekend met f'.

Memocombinator

$$
\begin{aligned}
&memo :: ((Integer \to \alpha) \to Integer \to \alpha) \to Integer \to \alpha \\
&memo \quad f' \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad = f \\
&\quad \textbf{where} \\
&\qquad f = apply\ (tabulate\ (f'\ f))
\end{aligned}
$$

- De combinator levert een dekpunt f van f'.
- De functie f haalt zijn resultaat uit een memotabel $tabulate\ (f'\ f)$.
- Elk element in de tabel wordt berekend met f'.
- Recursieve aanroepen gebruiken de memofunctie f.

Memocombinator

$$
\begin{aligned}
&memo :: ((Integer \to \alpha) \to Integer \to \alpha) \to Integer \to \alpha \\
&memo \quad f' \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad = f \\
&\quad \textbf{where} \\
&\qquad f = apply\ (tabulate\ (f'\ f))
\end{aligned}
$$

- De combinator levert een dekpunt f van f'.
- De functie f haalt zijn resultaat uit een memotabel $tabulate\ (f'\ f)$.
- Elk element in de tabel wordt berekend met f'.
- Recursieve aanroepen gebruiken de memofunctie f.
- Dankzij lazy evaluation worden alleen die elementen in de memolijst uitgerekend die echt nodig.

Memocombinator

$$
\begin{aligned}
&memo :: ((Integer \to \alpha) \to Integer \to \alpha) \to Integer \to \alpha \\
&memo \quad f' \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad = f \\
&\quad \textbf{where} \\
&\qquad f = apply\ (tabulate\ (f'\ f))
\end{aligned}
$$

- De combinator levert een dekpunt f van f'.
- De functie f haalt zijn resultaat uit een memotabel $tabulate\ (f'\ f)$.
- Elk element in de tabel wordt berekend met f'.
- Recursieve aanroepen gebruiken de memofunctie f.
- Dankzij lazy evaluation worden alleen die elementen in de memolijst uitgerekend die echt nodig.
- De tabel is onafhankelijk van de parameter van f; alle aanroepen van f maken dus gebruik van dezelfde tabel.

Fibonaccireeks met memolijsten

Fibonaccifunctie met globale memoïsatie:

$$
\begin{aligned}
&\mathit{fib} :: \mathit{Integer} \to \mathit{Integer} \\
&\mathit{fib} = \mathit{memo}\ \mathit{fib}' \\
&\quad \textbf{where} \\
&\qquad \mathit{fib}'\ f\ 0 = 0 \\
&\qquad \mathit{fib}'\ f\ 1 = 1 \\
&\qquad \mathit{fib}'\ f\ n = f\ (n-2) + f\ (n-1)
\end{aligned}
$$

Memolijsten: aantal reducties

```
Main > fib 10
55
1450 reductions, 2316 cells
Main > fib 20
6765
5060 reductions, 8178 cells
Main > fib 25
75025
7690 reductions, 12463 cells
Main > fib 30
832040
10870 reductions, 17649 cells
```

Memolijsten: opeenvolgende aanroepen

```
Main >
```

Memolijsten: opeenvolgende aanroepen

```
Main > fib 30
```

Memolijsten: opeenvolgende aanroepen

```
Main > fib 30
832040
10870 reductions, 17649 cells

Main >
```

Memolijsten: opeenvolgende aanroepen

```
Main > fib 30
832040
10870 reductions, 17649 cells
Main > fib 30
```

Memolijsten: opeenvolgende aanroepen

```
Main > fib 30
832040
10870 reductions, 17649 cells
Main > fib 30
832040
359 reductions, 583 cells
```

Memolijsten: opeenvolgende aanroepen

```
Main > fib 30
832040
10870 reductions, 17649 cells
Main > fib 30
832040
359 reductions, 583 cells
```

Bij de tweede aanroep hoeft alleen het resultaat in de lijst opgezocht te worden.

Memolijsten: lineaire zoektijd

- Arrays: gefixeerd aantal argumenten, maar constante zoektijd.
- Lijsten: alle argumenten, maar lineaire zoektijd.

Memolijsten: lineaire zoektijd

- Arrays: gefixeerd aantal argumenten, maar constante zoektijd.
- Lijsten: alle argumenten, maar lineaire zoektijd.

```
Main> fib 5000
3878968454388325633701916308325905312082127714...
41.78 secs, 2532516300 bytes
```

Memolijsten: lineaire zoektijd

- Arrays: gefixeerd aantal argumenten, maar constante zoektijd.
- Lijsten: alle argumenten, maar lineaire zoektijd.

```
Main> fib 5000
387896845438832563370191630832590531208212771 4...
41.78 secs, 2532516300 bytes
```

Gulden middenweg: memobomen (alle argumenten, logaritmische zoektijd).

Bibliotheek voor memofuncties: aanpak (onveranderd)

Kies een (geparameteriseerd) datatype $Memo$ voor memotabellen.

Definieer functies $tabulate$ en $apply$,

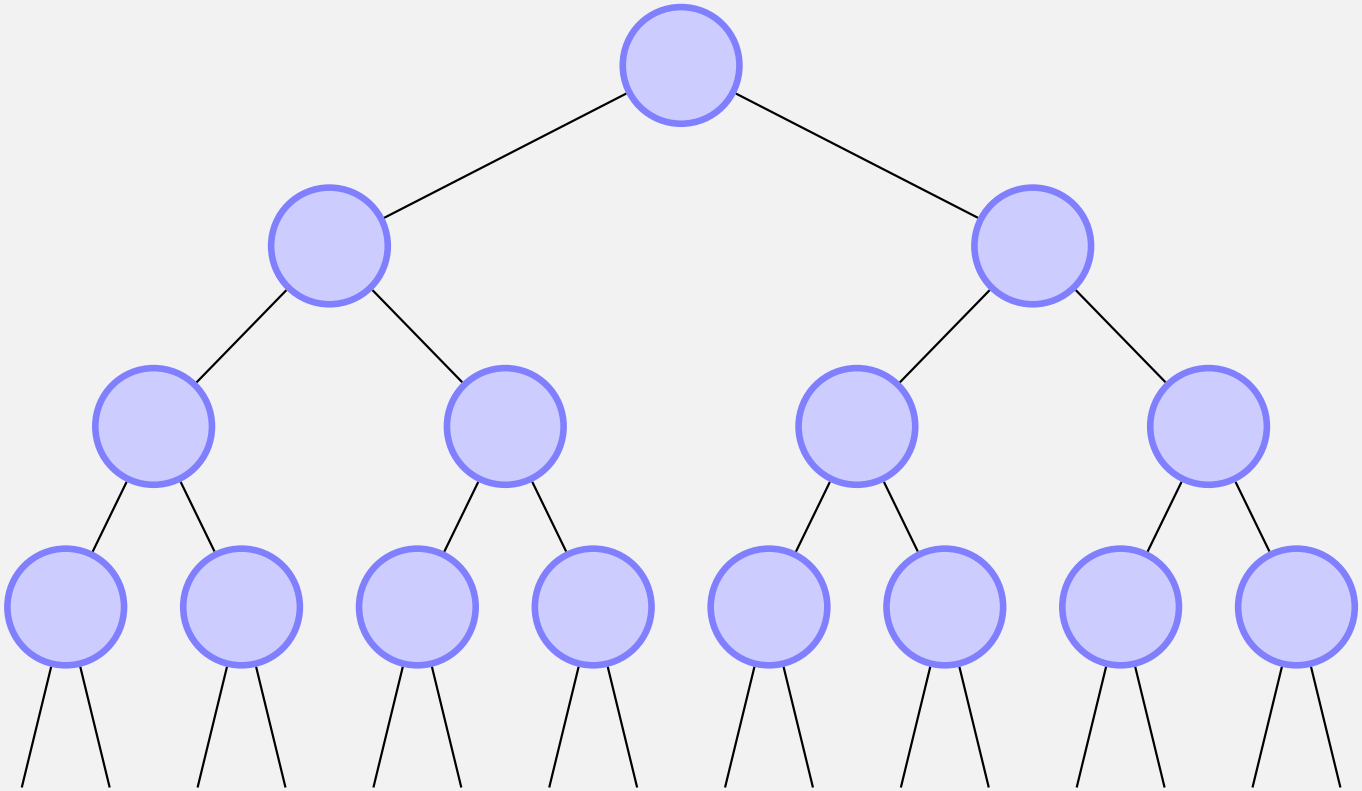
$$\begin{aligned} tabulate &:: (Integer \to \alpha) \to Memo\ \alpha \\ apply &:: Memo\ \alpha \to Integer \to \alpha \end{aligned}$$

zodat:

- $tabulate\ f$ een (lazy) memotabel met alle functiewaarden van f oplevert en
- $apply\ mem\ n$ de waarde voor n in mem oplevert.

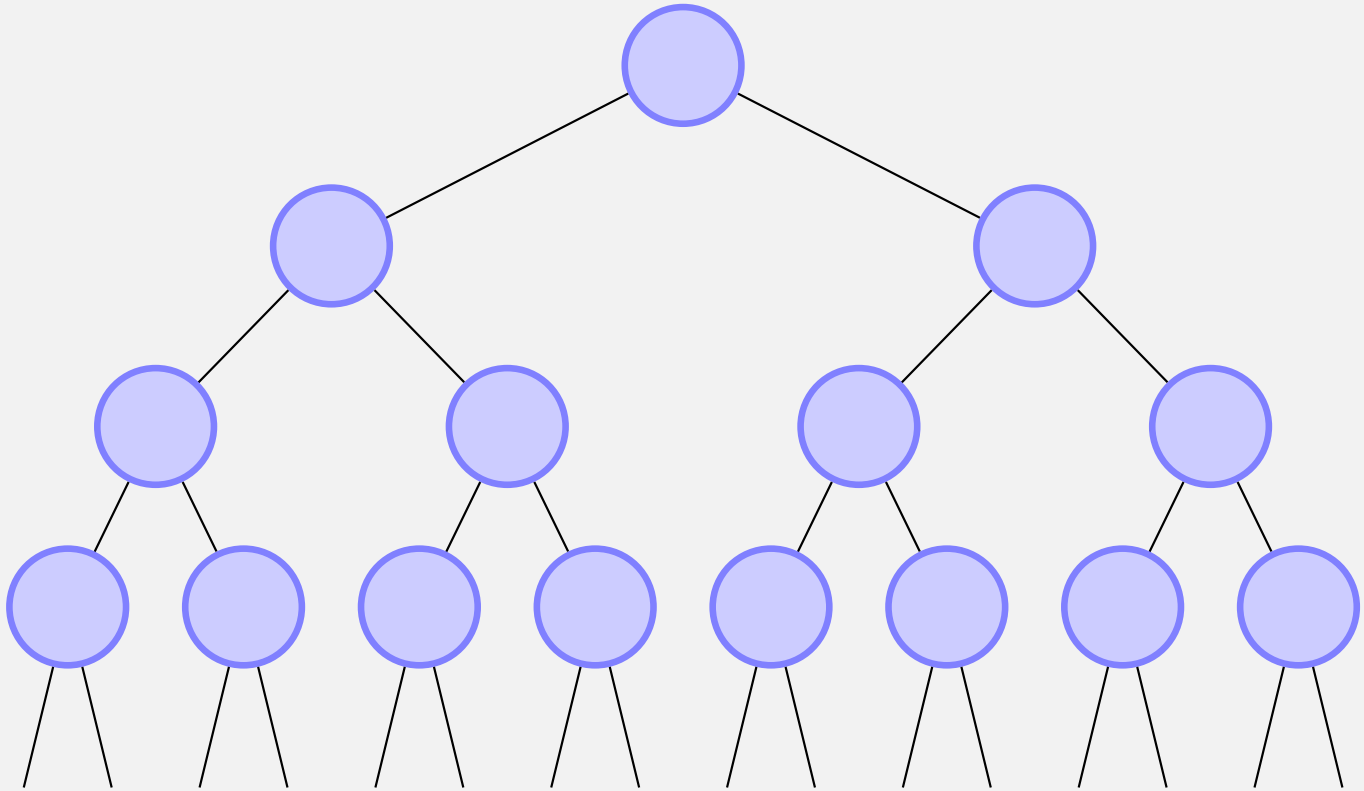
Definieer een dekpuntcombinator $memo$ in termen van $tabulate$ en $apply$.

Memobomen



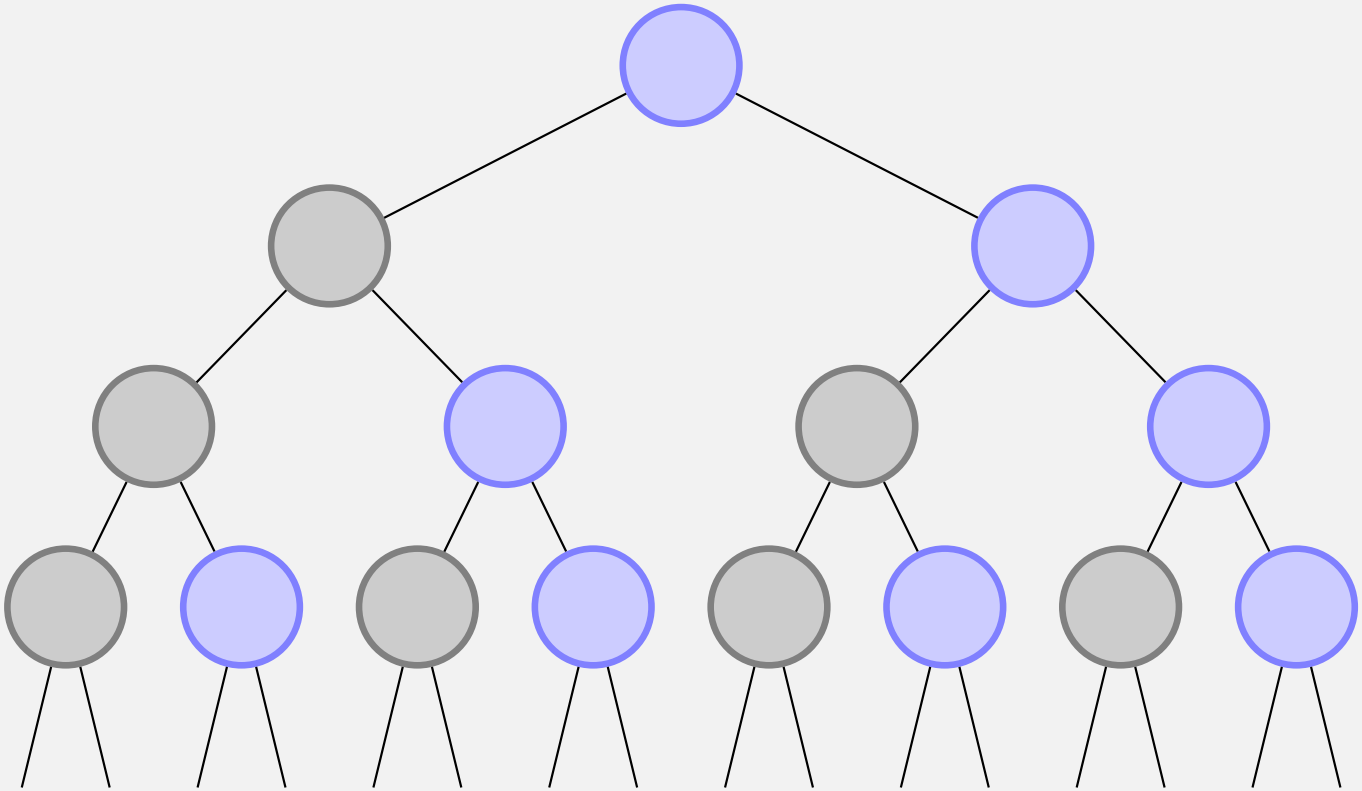
Memobomen

- Oneindige binaire boom met waarden in knopen.
- Geen waarden in linkerkinderen.



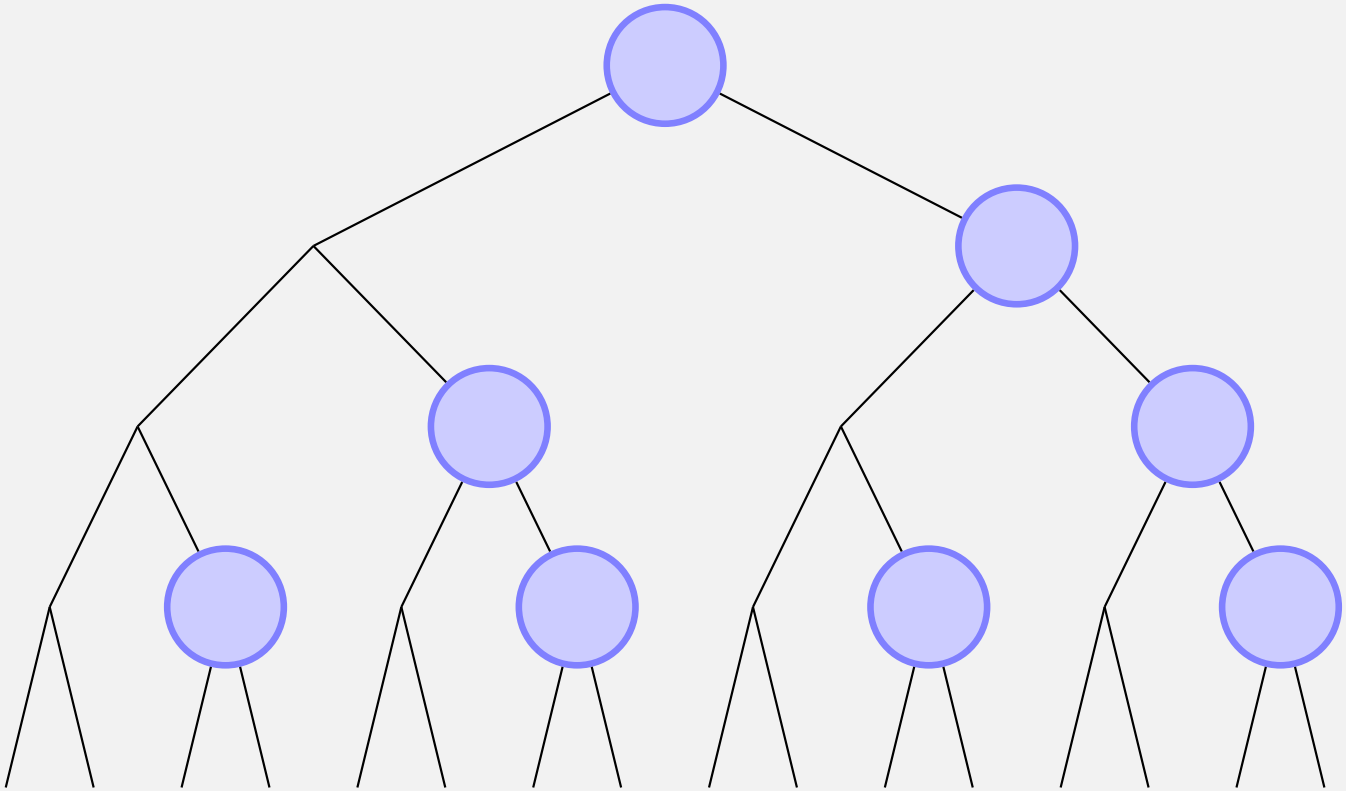
Memobomen

- Oneindige binaire boom met waarden in knopen.
- Geen waarden in linkerkinderen.



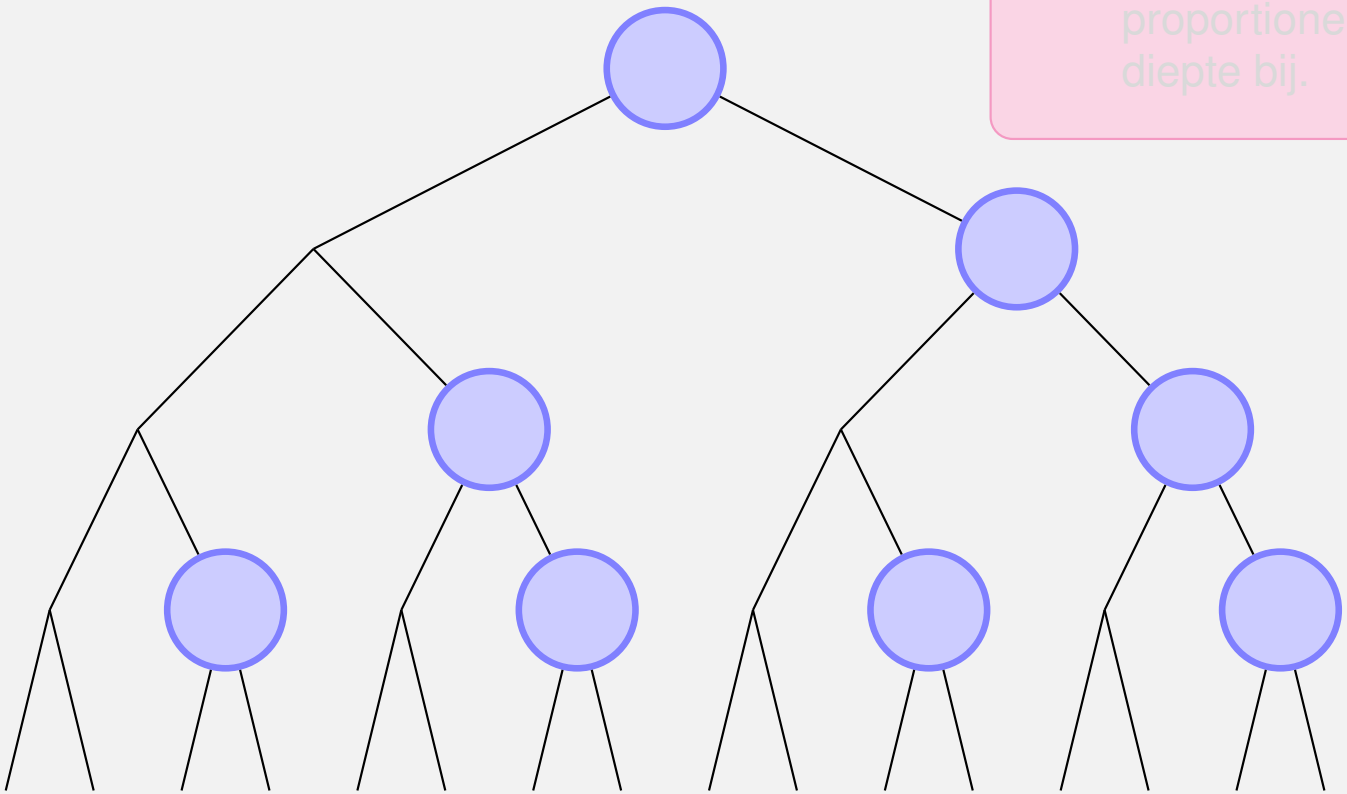
Memobomen

- Oneindige binaire boom met waarden in knopen.
- Geen waarden in linkerkinderen.



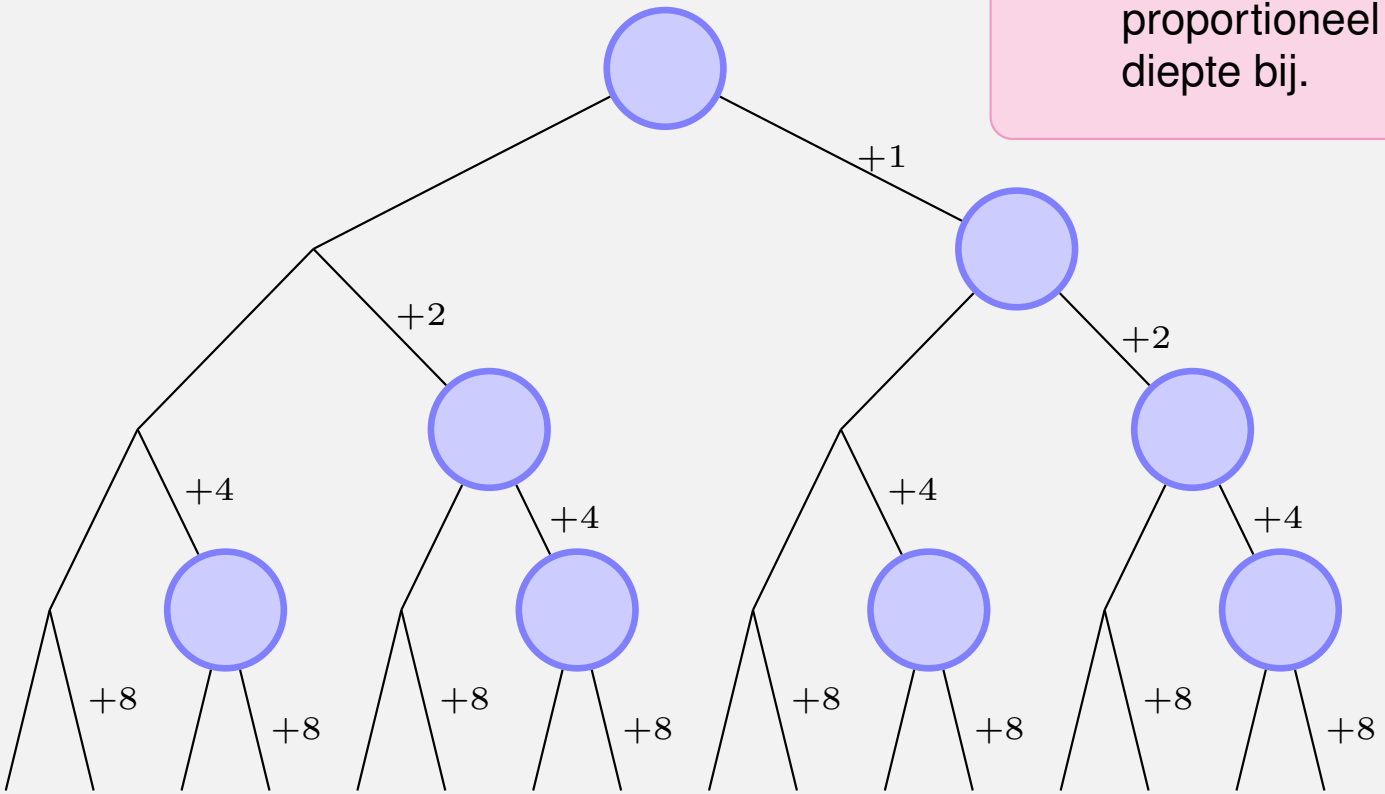
Memobomen

- De zoeksleutel voor een rechterkind wordt bepaald door de rechtertakken op het pad vanaf de wortel.
- Iedere rechtertak draagt proportioneel aan zijn diepte bij.



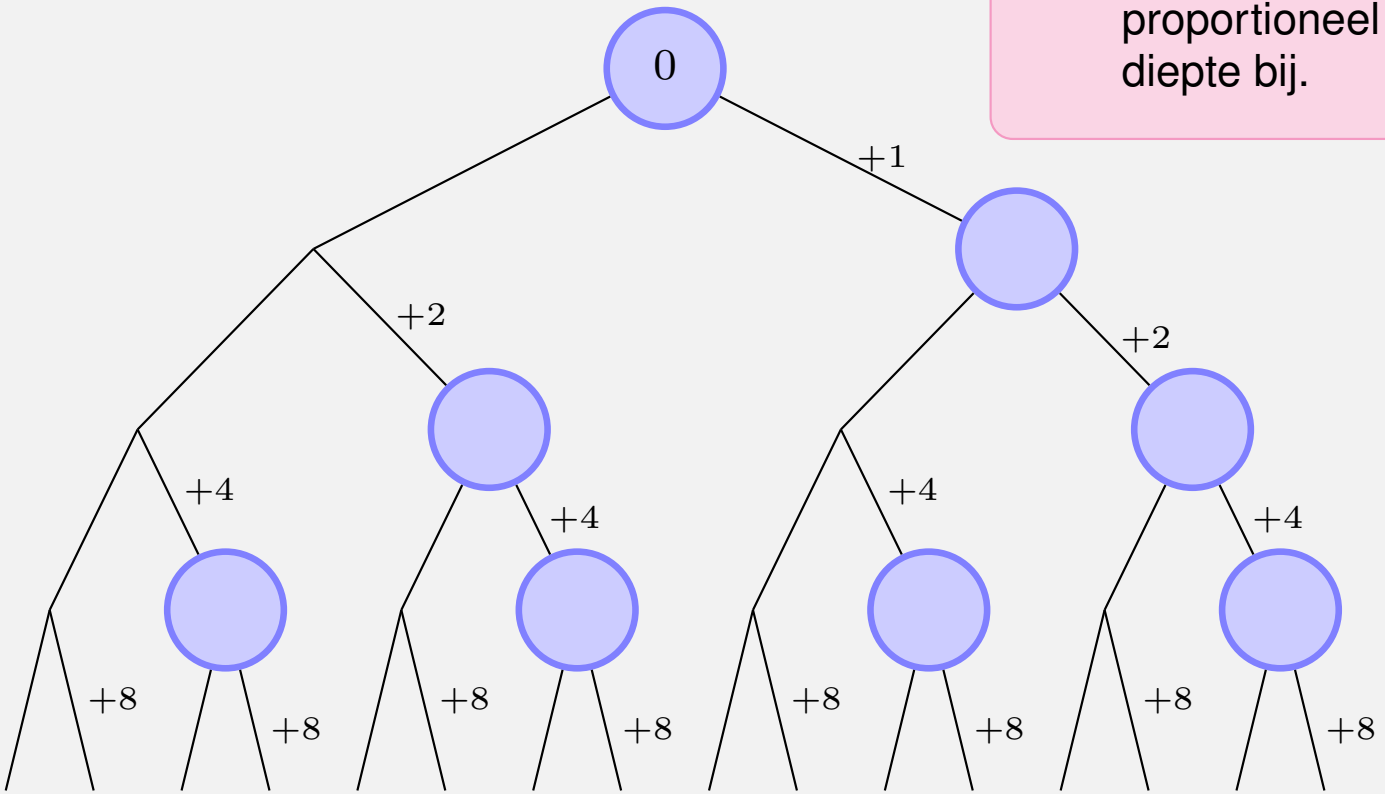
Memobomen

- De zoeksleutel voor een rechterkind wordt bepaald door de rechtertakken op het pad vanaf de wortel.
- Iedere rechtertak draagt proportioneel aan zijn diepte bij.



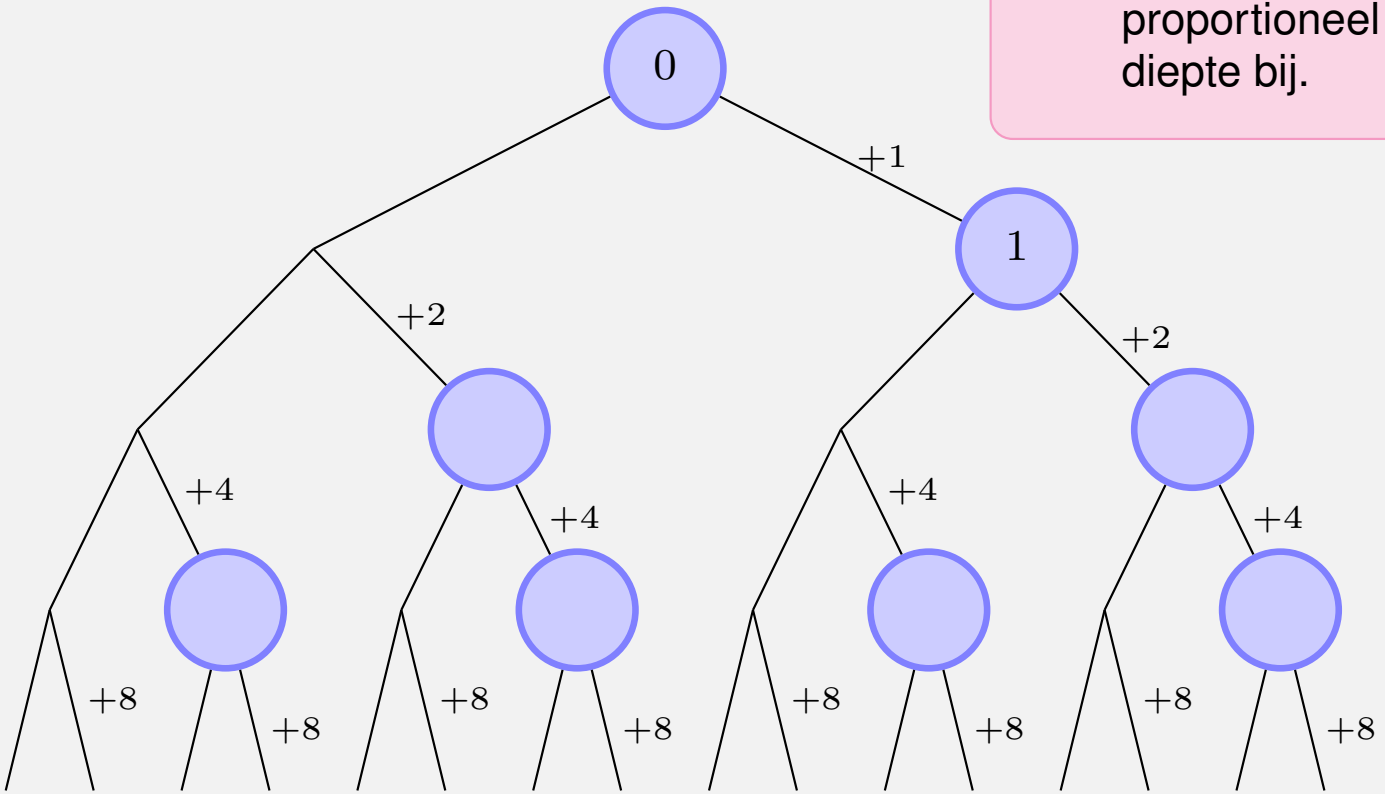
Memobomen

- De zoeksleutel voor een rechterkind wordt bepaald door de rechtertakken op het pad vanaf de wortel.
- Iedere rechtertak draagt proportioneel aan zijn diepte bij.



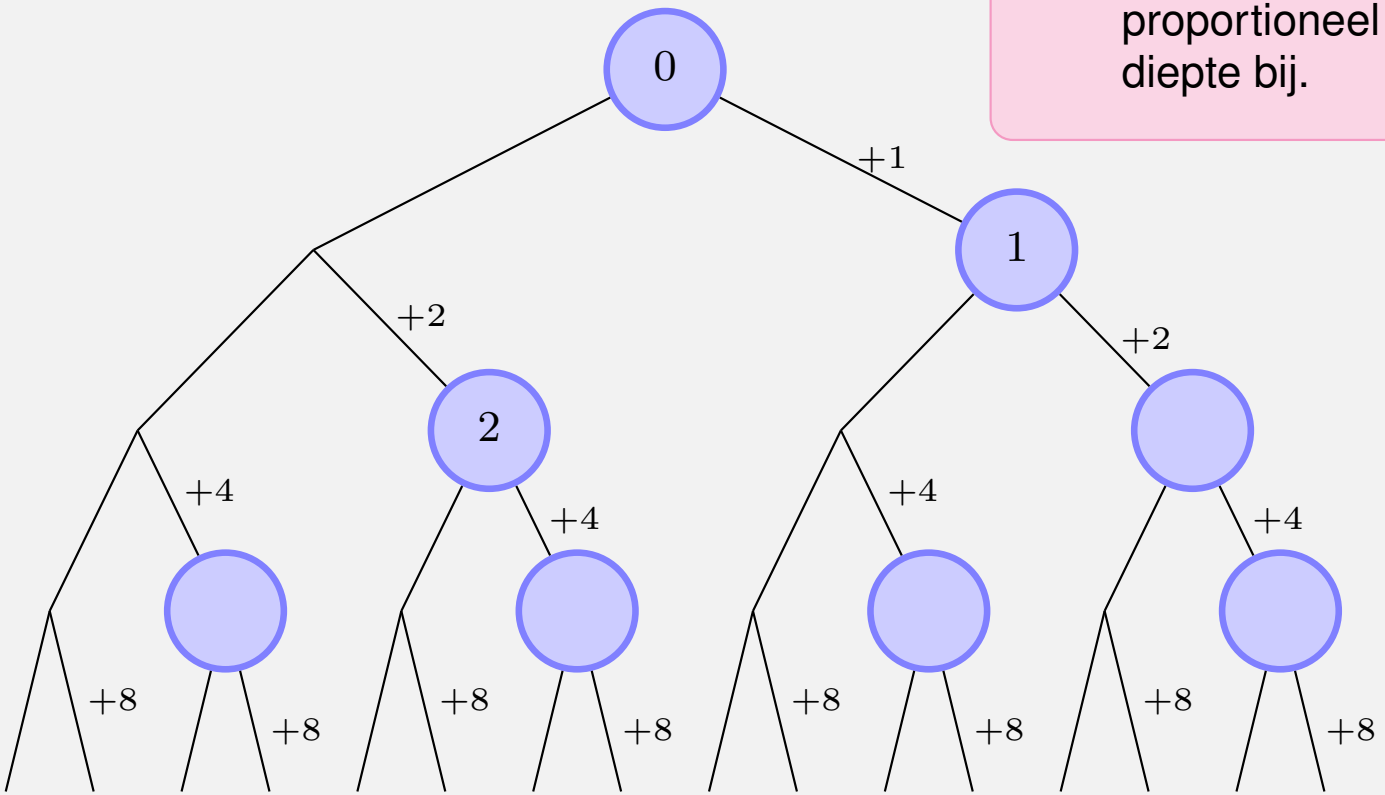
Memobomen

- De zoeksleutel voor een rechterkind wordt bepaald door de rechtertakken op het pad vanaf de wortel.
- Iedere rechtertak draagt proportioneel aan zijn diepte bij.



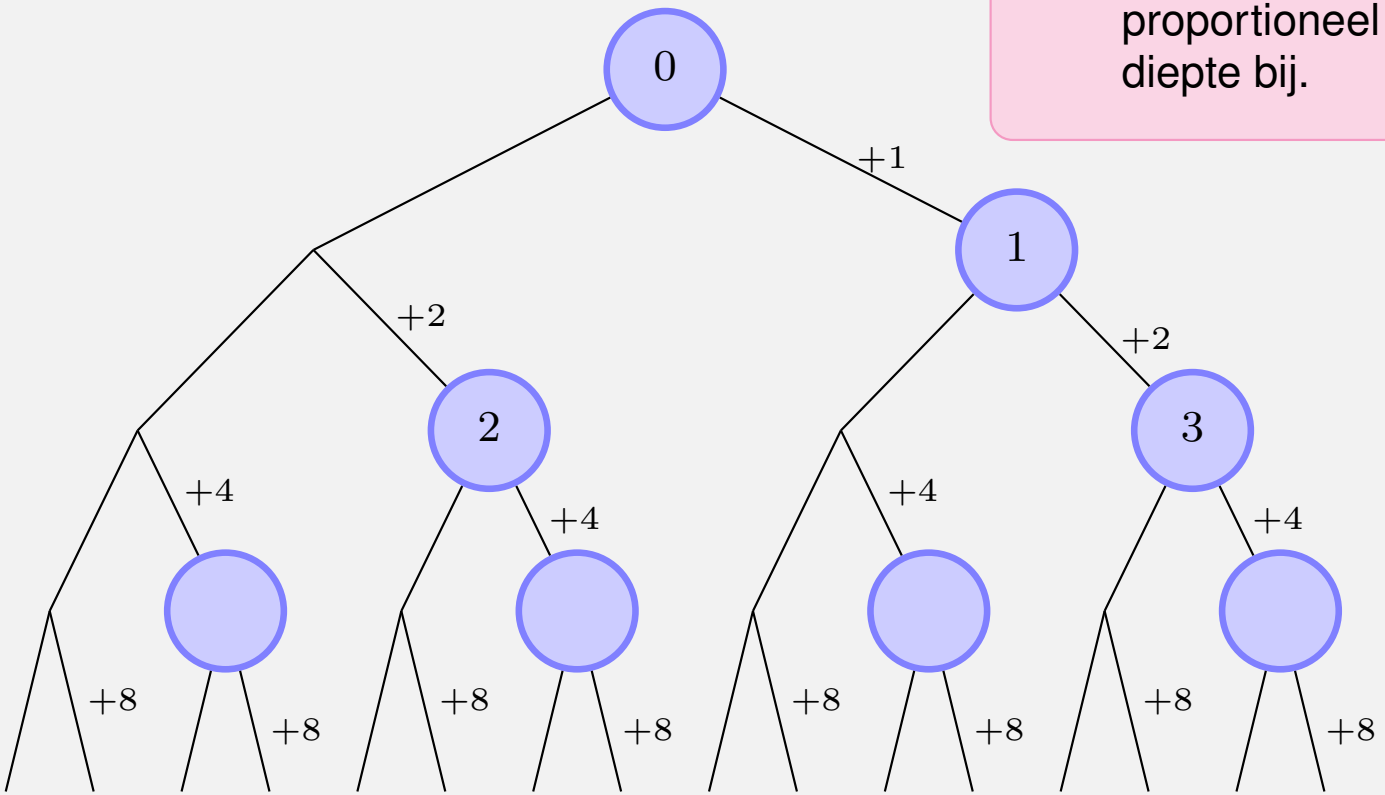
Memobomen

- De zoeksleutel voor een rechterkind wordt bepaald door de rechtertakken op het pad vanaf de wortel.
- Iedere rechtertak draagt proportioneel aan zijn diepte bij.



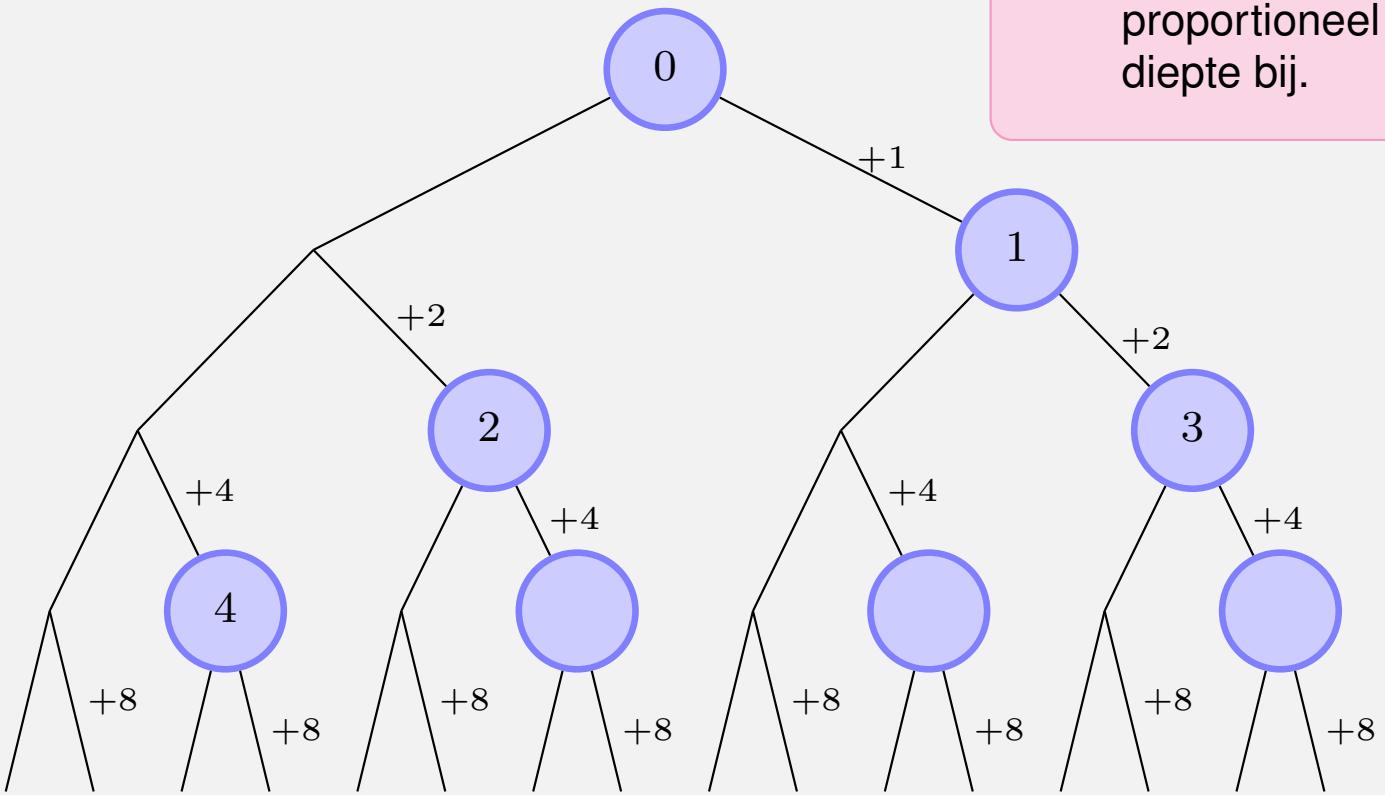
Memobomen

- De zoeksleutel voor een rechterkind wordt bepaald door de rechtertakken op het pad vanaf de wortel.
- Iedere rechtertak draagt proportioneel aan zijn diepte bij.



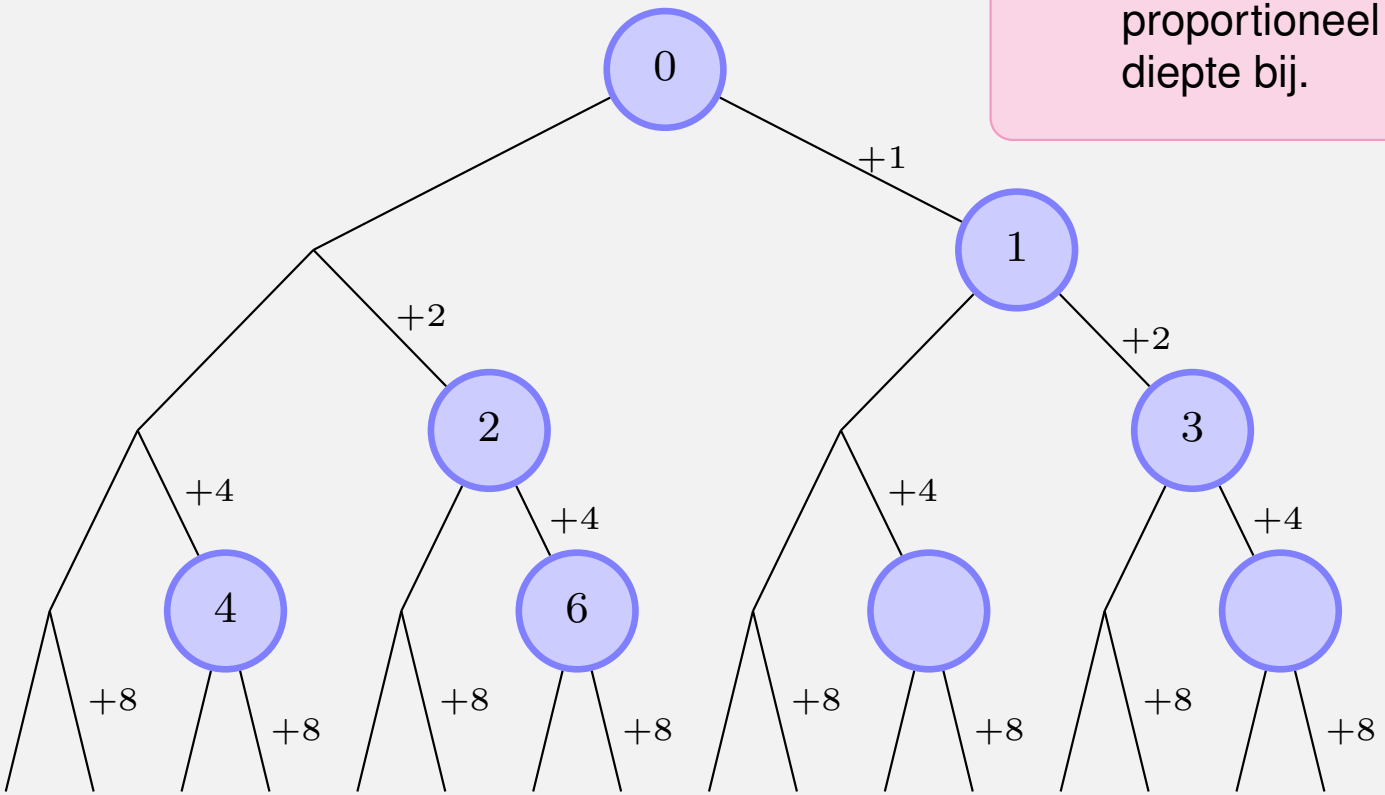
Memobomen

- De zoeksleutel voor een rechterkind wordt bepaald door de rechtertakken op het pad vanaf de wortel.
- Iedere rechtertak draagt proportioneel aan zijn diepte bij.



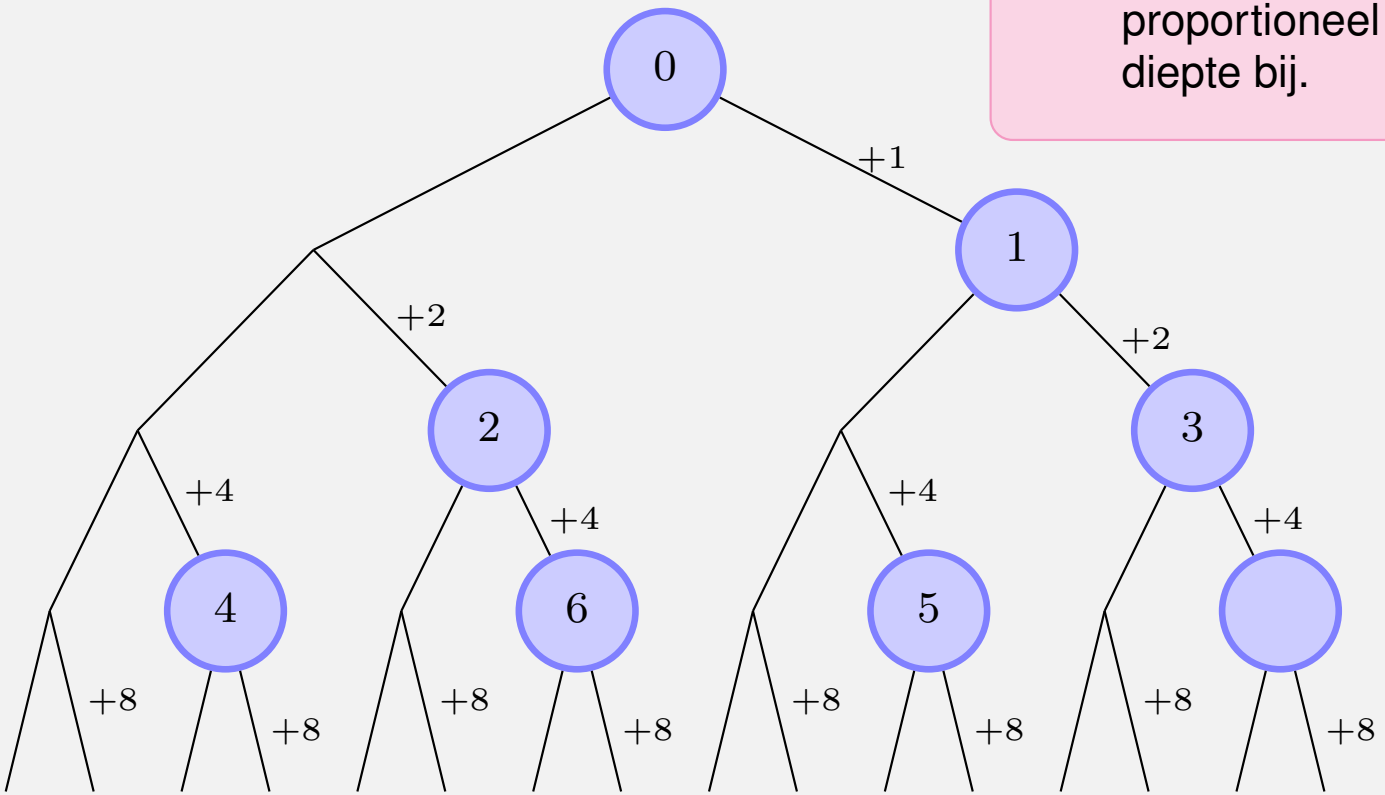
Memobomen

- De zoeksleutel voor een rechterkind wordt bepaald door de rechtertakken op het pad vanaf de wortel.
- Iedere rechtertak draagt proportioneel aan zijn diepte bij.



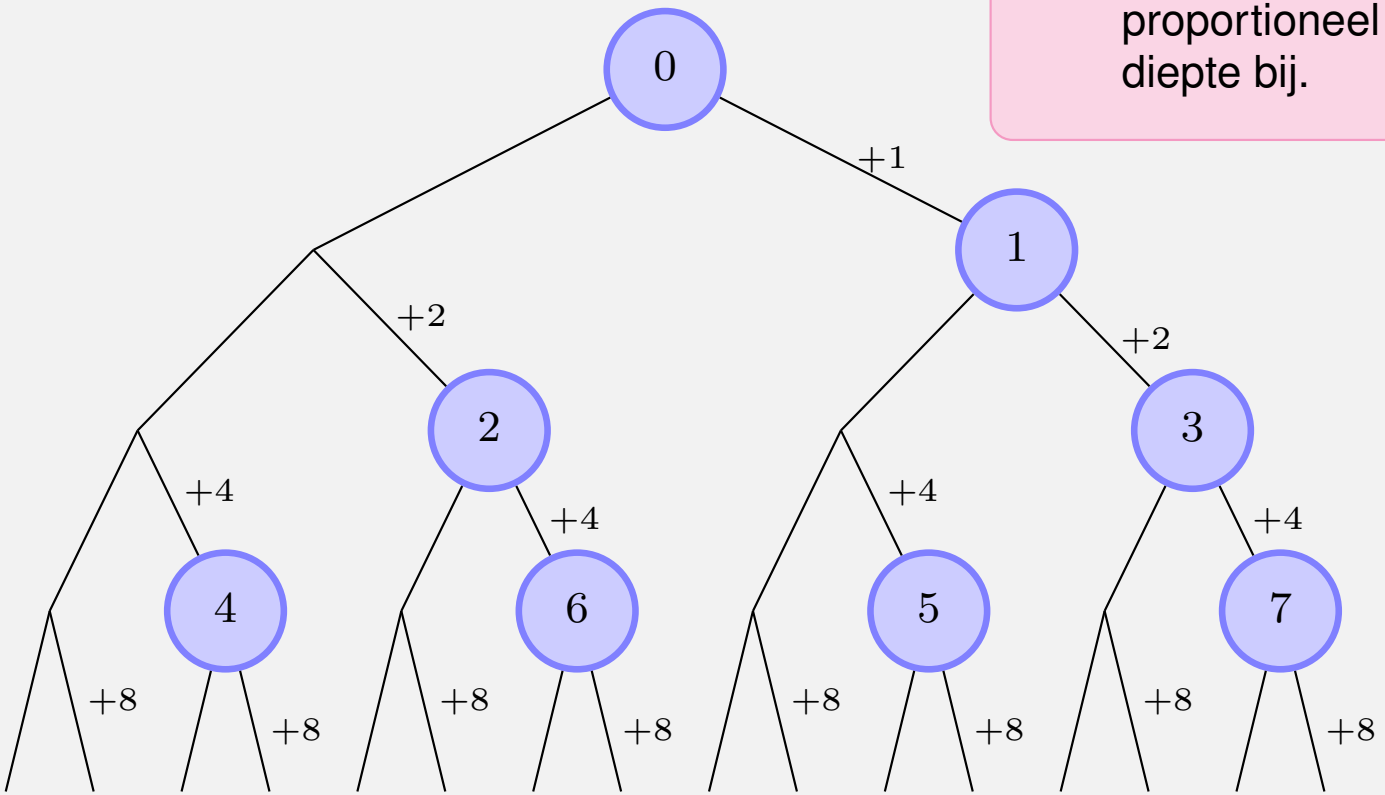
Memobomen

- De zoeksleutel voor een rechterkind wordt bepaald door de rechtertakken op het pad vanaf de wortel.
- Iedere rechtertak draagt proportioneel aan zijn diepte bij.



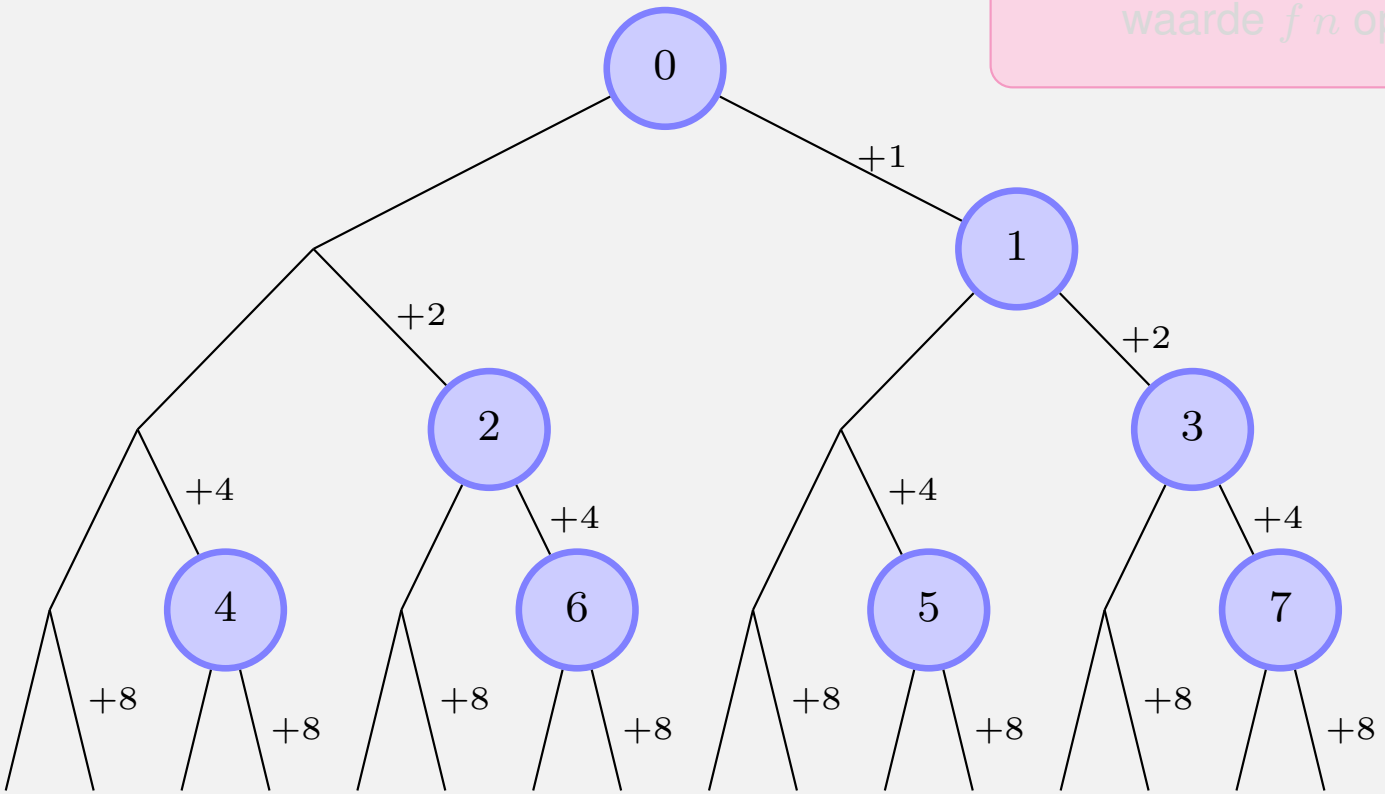
Memobomen

- De zoeksleutel voor een rechterkind wordt bepaald door de rechtertakken op het pad vanaf de wortel.
- Iedere rechtertak draagt proportioneel aan zijn diepte bij.



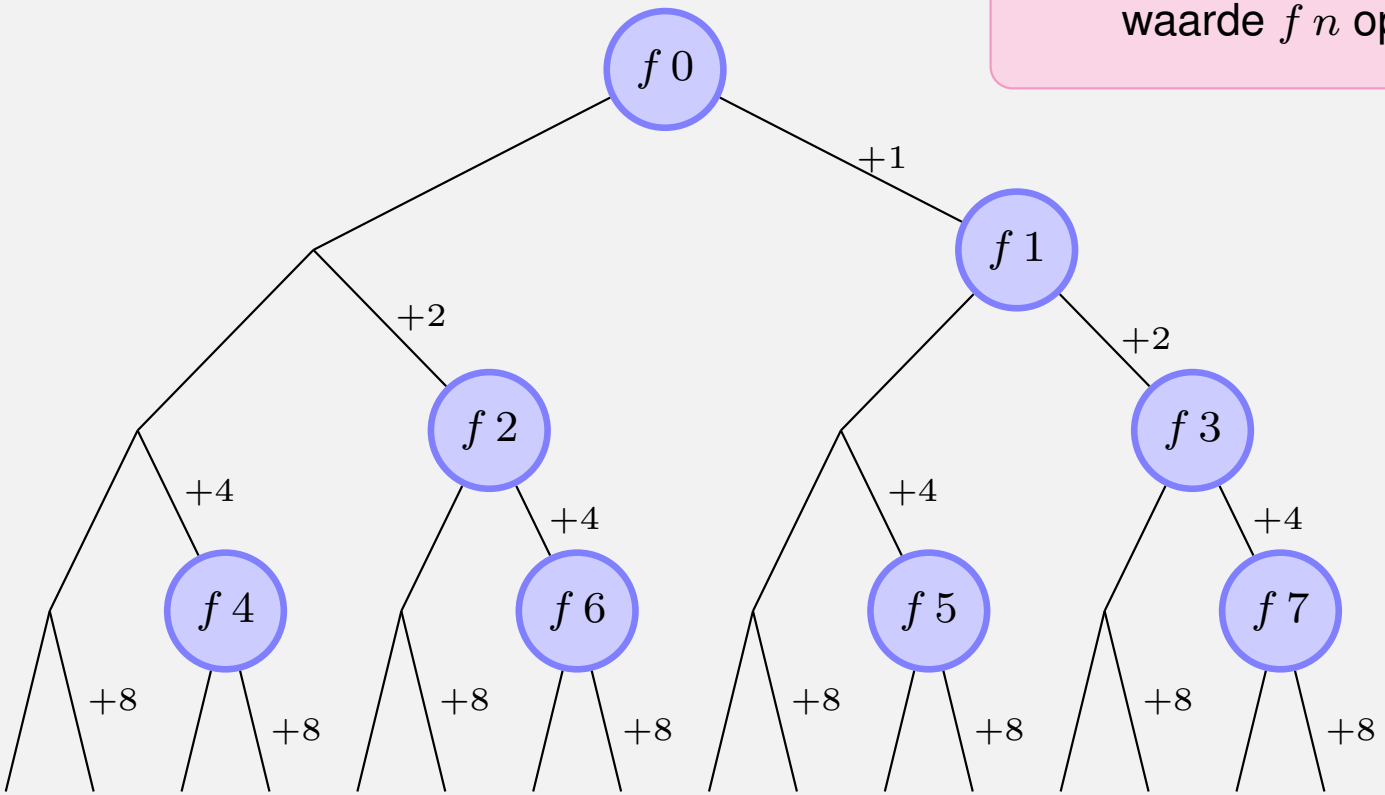
Memobomen

- In de knopen slaan we waarden voor een te memoïseren functie f op.
- In een rechterkind met zoeksleutel n wordt de waarde $f\ n$ opgeslagen.

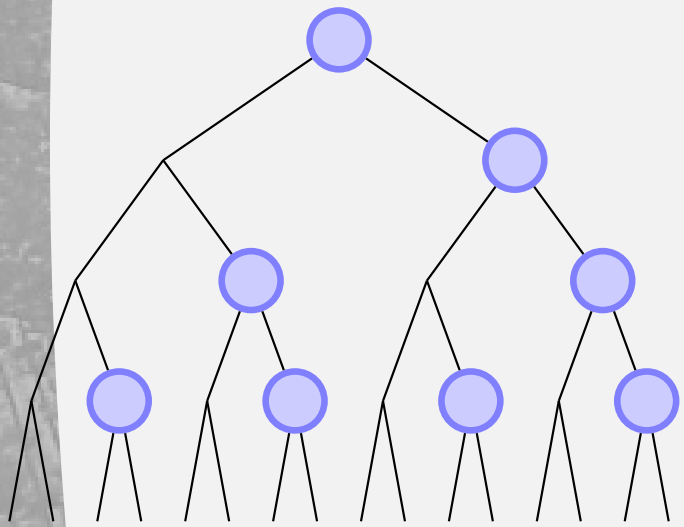


Memobomen

- In de knopen slaan we waarden voor een te memoïseren functie f op.
- In een rechterkind met zoeksleutel n wordt de waarde $f\ n$ opgeslagen.

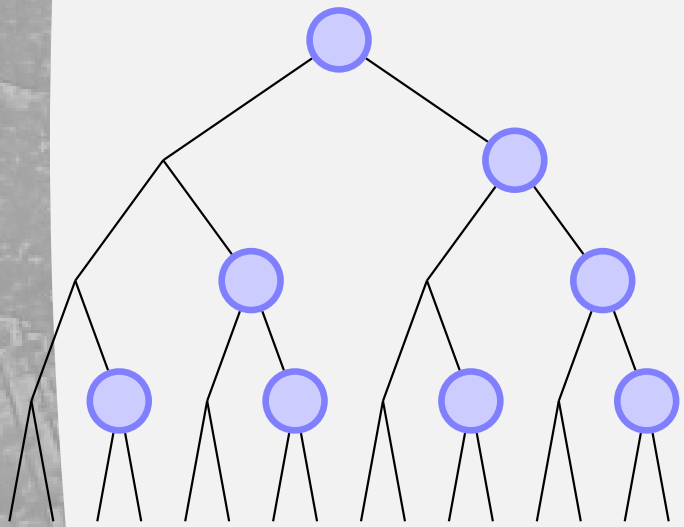


Datatype voor memobomen



Type van oneindige binaire bomen met waarden in de wortel en in elk rechterkind.

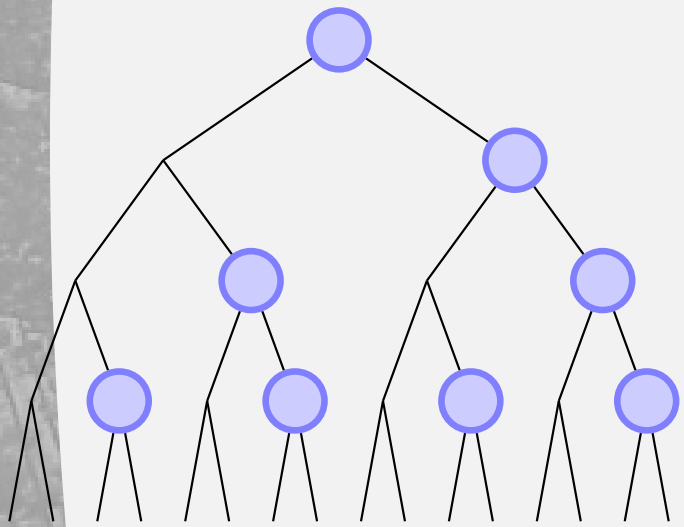
Datatype voor memobomen



Type van oneindige binaire bomen met waarden in de wortel en in elk rechterkind.

$$\begin{aligned}&\textbf{data}\ Memo\ \ \alpha = Memo\ \ (Memo'\ \alpha)\ \alpha\ (Memo\ \alpha)\\&\textbf{data}\ Memo'\ \alpha = Memo'\ (Memo'\ \alpha)\quad (Memo\ \alpha)\end{aligned}$$

Datatype voor memobomen



Type van oneindige binaire bomen met waarden in de wortel en in elk rechterkind.

$$\begin{array}{l}\mathbf{data}\ Memo\ \ \alpha = Memo\ \ (Memo'\ \alpha)\ \alpha\ (Memo\ \alpha) \\ \mathbf{data}\ Memo'\ \alpha = Memo'\ (Memo'\ \alpha)\quad (Memo\ \alpha)\end{array}$$

☞ $Memo$ en $Memo'$ zijn wederzijds recursief definieerd.

Memoboomconstructie

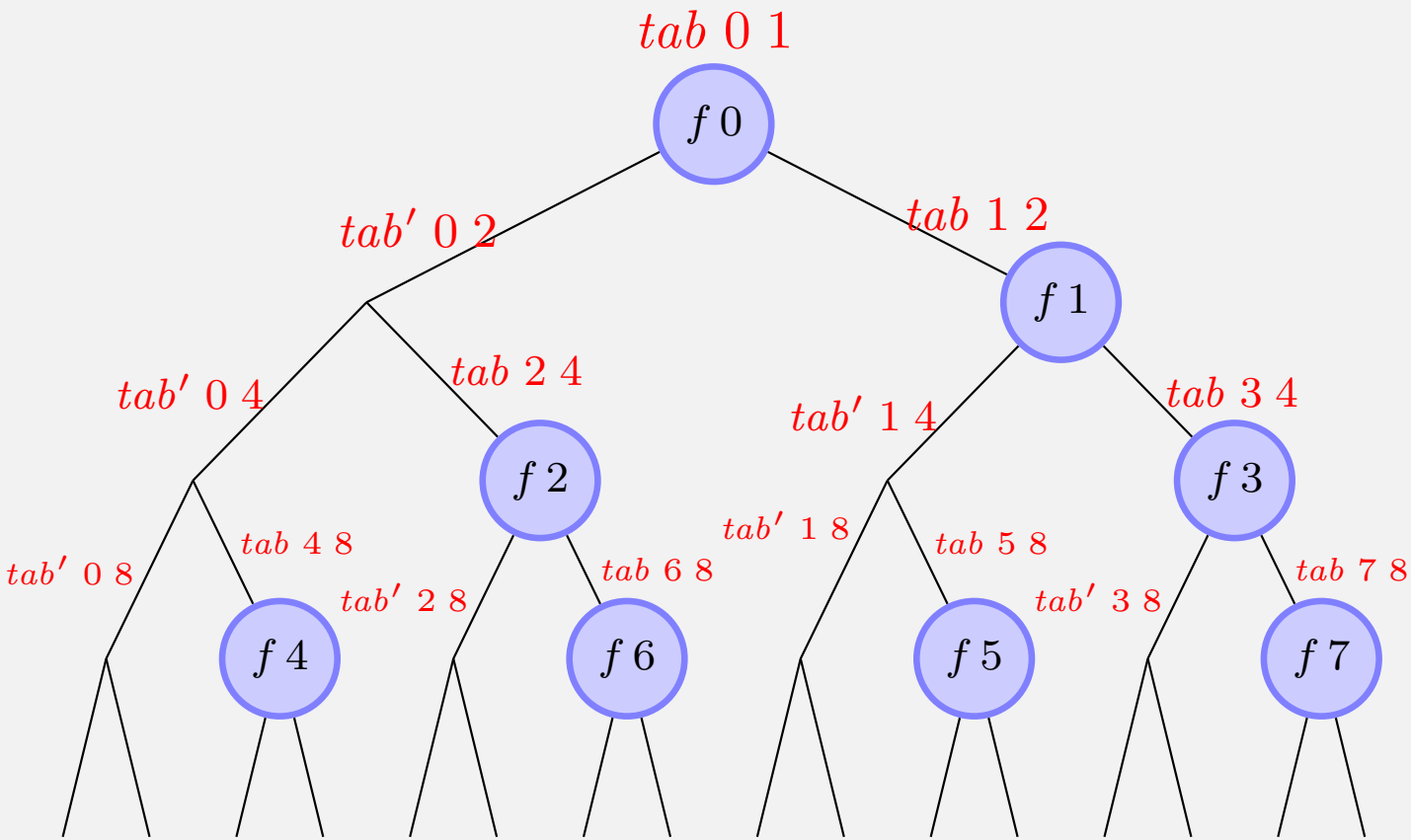
$$
\begin{array}{l}
tabulate :: (Integer \to \alpha) \to Memo\ \alpha \\
tabulate\quad f \qquad\qquad\qquad\qquad = tab\ 0\ 1 \\
\quad \textbf{where} \\
\qquad tab\ k\ i = \\
\qquad\quad \textbf{let}\ j = 2 * i\ \textbf{in}\ Memo\ (tab'\ k\ j)\ (f\ k)\ (tab\ (k+i)\ j) \\
\qquad tab'\ k\ i = \\
\qquad\quad \textbf{let}\ j = 2 * i\ \textbf{in}\ Memo'\ (tab'\ k\ j)\ (tab\ (k+i)\ j)
\end{array}
$$

Hulpfunctie argumenten:

- Voor tab: eerstvolgende zoeksleutel en volgende toename van de zoeksleutel.
- Voor tab': vorige zoeksleutel en volgende toename van de zoeksleutel.

Memoboomconstructie: voorbeeld

tabulate f



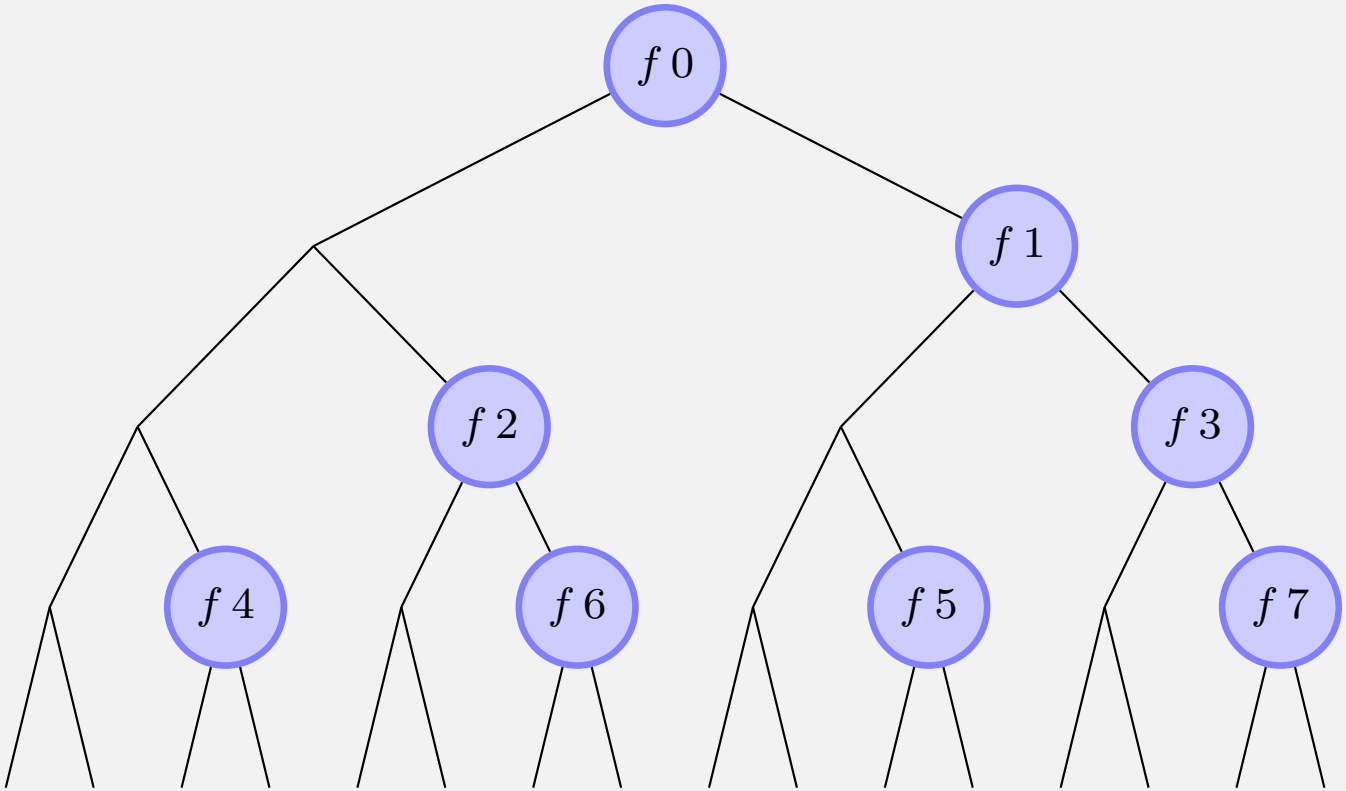
Zoeken in een memoboom

```
apply :: Memo α → Integer → α
apply = app
  where
    app (Memo l x r) n | n ≡ 0       = x
                       | even n      = app' l (n `div` 2)
                       | otherwise   = app r (n `div` 2)
    app' (Memo' l r) n | even n      = app' l (n `div` 2)
                       | otherwise   = app r (n `div` 2)
```

In iedere recursieve stap wordt de zoeksleutel gehalveerd en dalen we één niveau af in de memoboom.

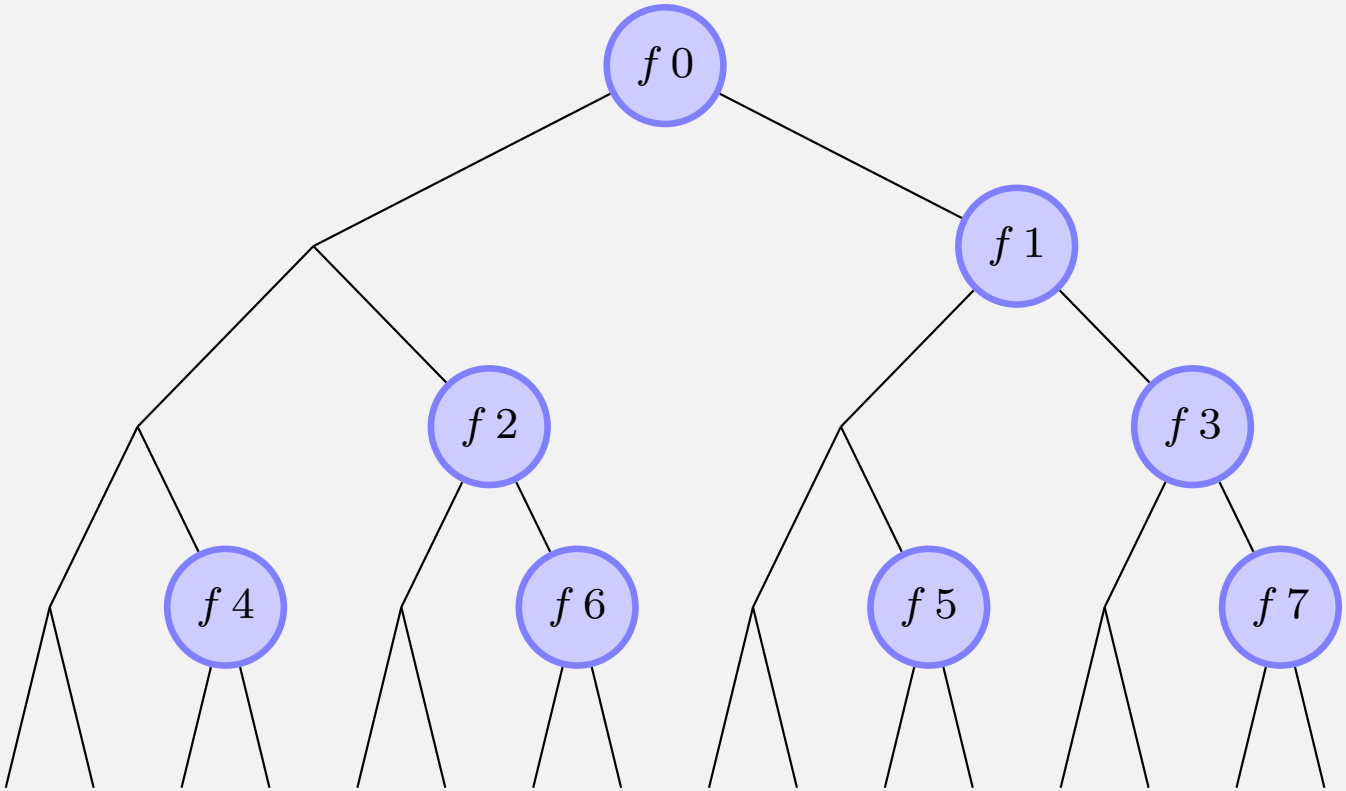
Als de zoeksleutel 0 is, leveren we de waarde in de huidige knoop op.

Zoeken in een memoboom: voorbeeld

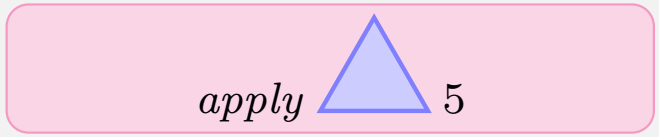


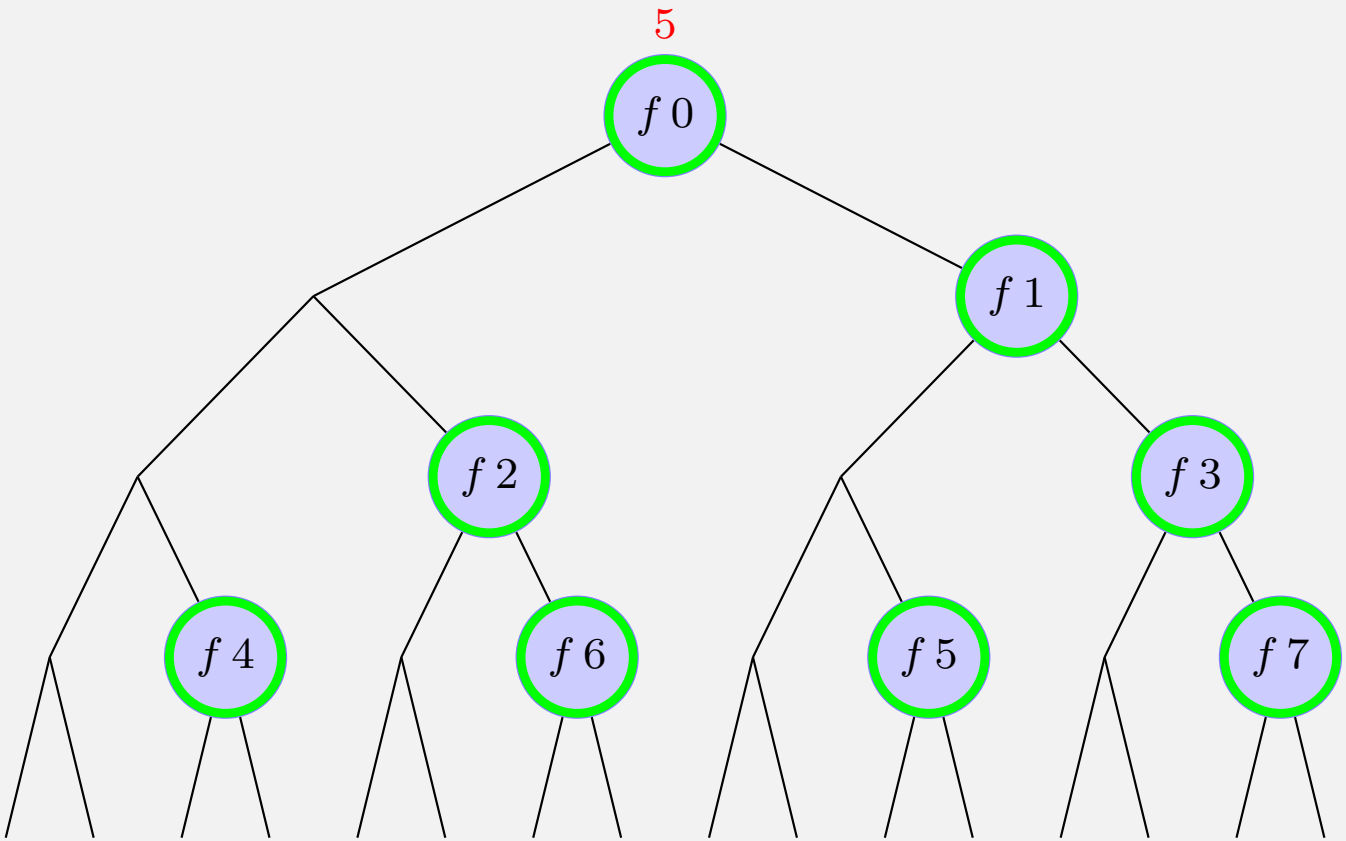
Zoeken in een memoboom: voorbeeld

$apply$  5



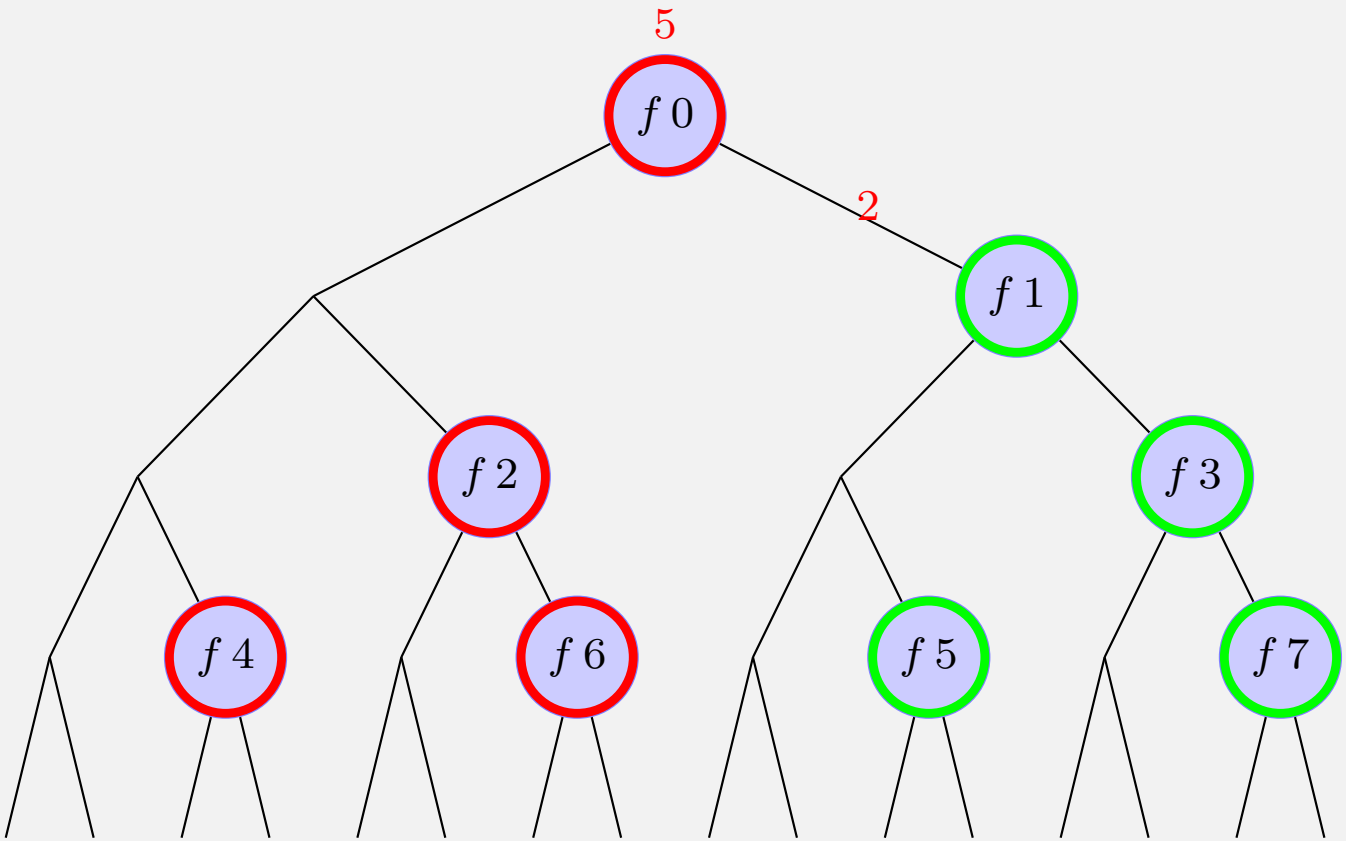
Zoeken in een memoboom: voorbeeld





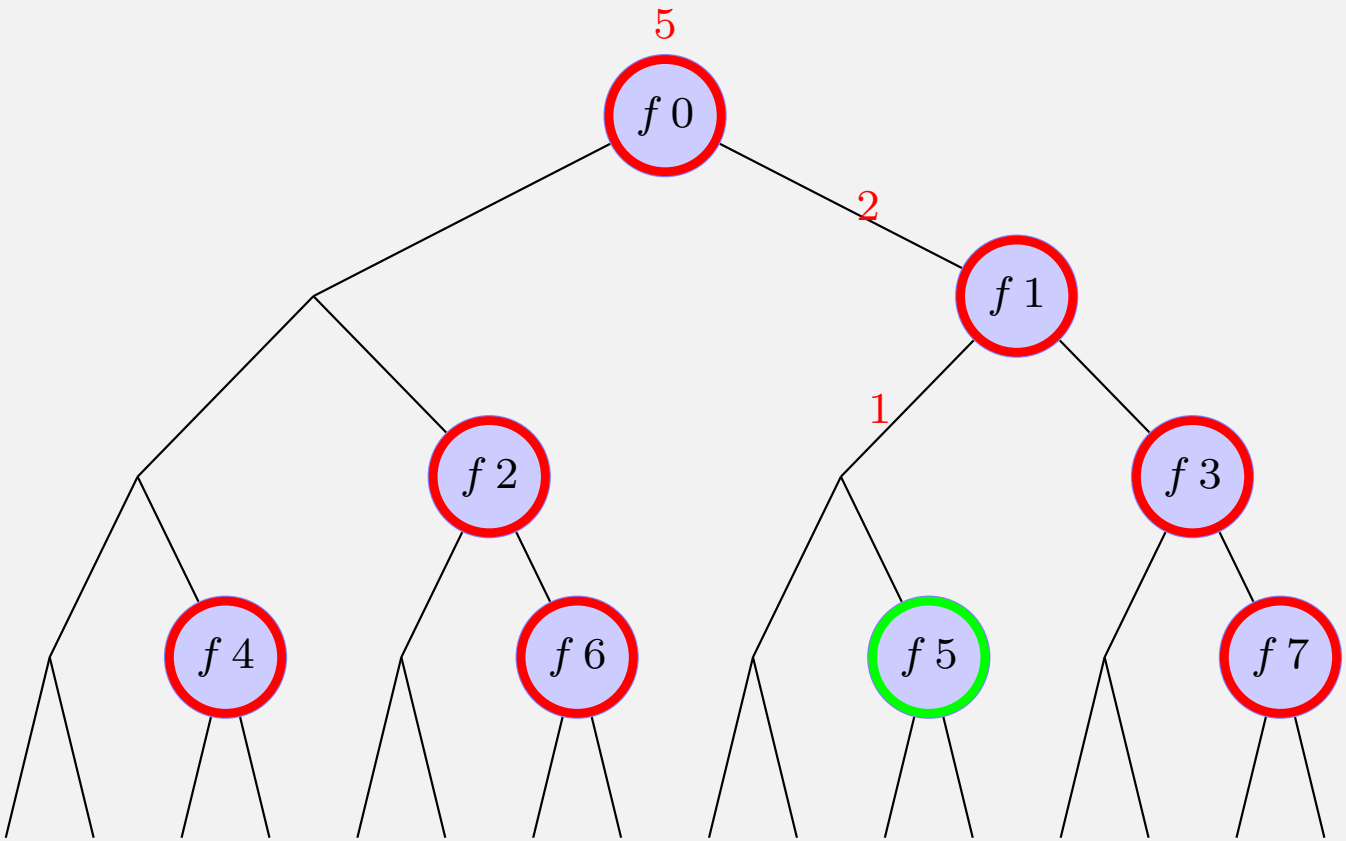
Zoeken in een memoboom: voorbeeld

$apply$  5



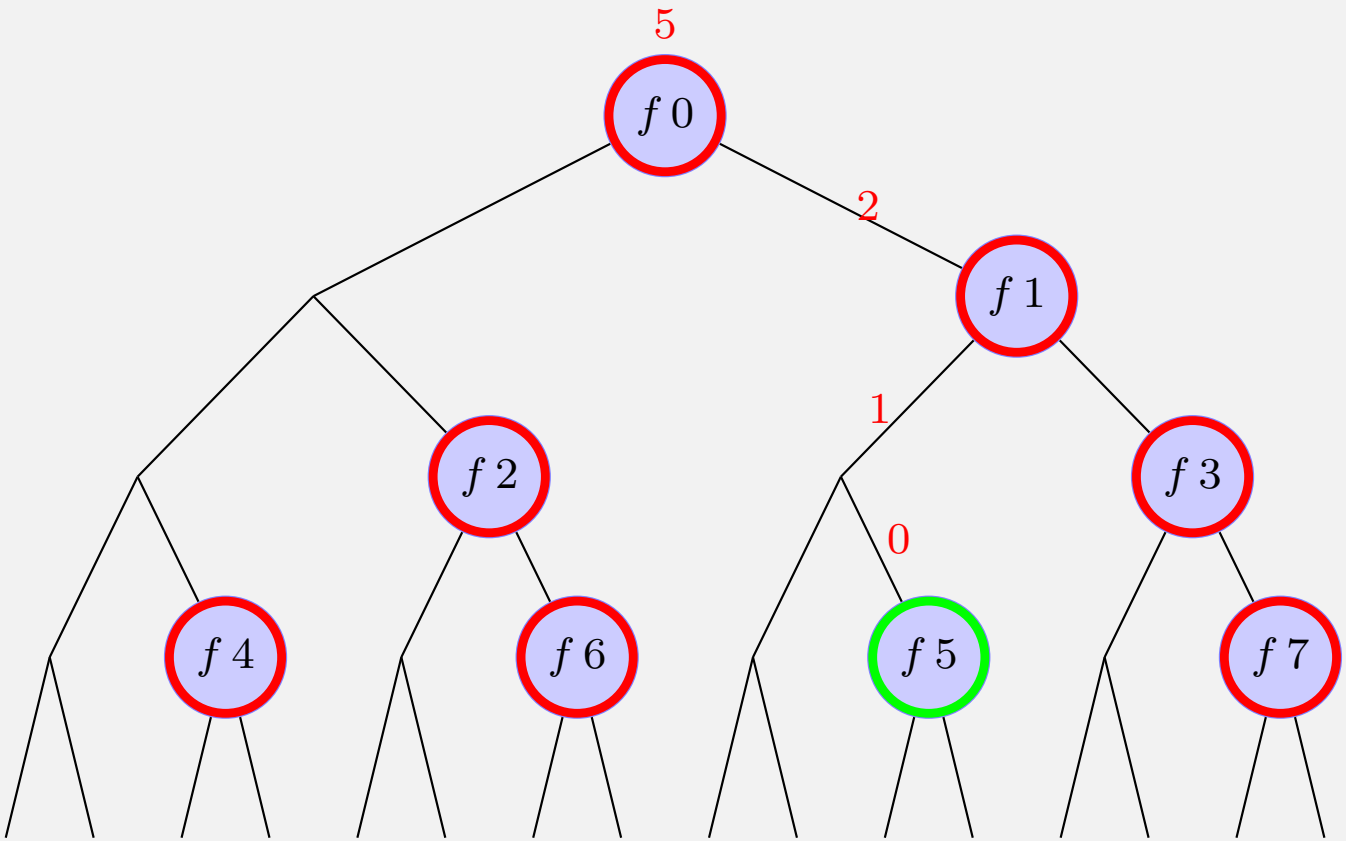
Zoeken in een memoboom: voorbeeld

$apply$  5



Zoeken in een memoboom: voorbeeld

$apply$  5



Memocombinator (onveranderd)

De definitie van $memo$ is onafhankelijk van de tabelrepresentatie:

$$
\begin{aligned}
&memo :: ((Integer \to \alpha) \to Integer \to \alpha) \to Integer \to \alpha \\
&memo \quad f' \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad = f \\
&\quad \mathbf{where} \\
&\qquad f = apply\ (tabulate\ (f'\ f))
\end{aligned}
$$

Fibonaccireeks met memobomen

$$
\begin{aligned}
&\mathit{fib} :: \mathit{Integer} \to \mathit{Integer} \\
&\mathit{fib} = \mathit{memo}\ \mathit{fib}' \\
&\quad \textbf{where} \\
&\qquad \mathit{fib}'\ f\ 0 = 0 \\
&\qquad \mathit{fib}'\ f\ 1 = 1 \\
&\qquad \mathit{fib}'\ f\ n = f\ (n-2) + f\ (n-1)
\end{aligned}
$$

Memobomen: tijd- en geheugenverbruik

```
Main >
```

Memobomen: tijd- en geheugenverbruik

```
Main >  fib 5000
```

Memobomen: tijd- en geheugenverbruik

```
Main > fib 5000
3878968454388325633701916308325905312082127714...
0.37 secs, 26809216 bytes

Main >
```

Memobomen: tijd- en geheugenverbruik

```
Main > fib 5000
387896845438832563370191630832590531208212771 4...
0.37 secs, 26809216 bytes
Main > fib 5000
```

Memobomen: tijd- en geheugenverbruik

```
Main > fib 5000
3878968454388325633701916308325905312082127714...
0.37 secs, 26809216 bytes
Main > fib 5000
3878968454388325633701916308325905312082127714...
0.02 secs, 532752 bytes
```

Conclusies

- Efficiëntere tabelstructuur vergt wat meer programmeerwerk, maar is een ‘one-time investment’.
- Keuze voor datastructuur niet zichtbaar voor gebruiker bibliotheek.
- Enige aanpassing aan gebruikerskant bibliotheek: expliciet maken van het recursiepatroon.