Achtste college algoritmiek

8 april 2016

Dynamisch Programmeren

Dutch Flag Problem

Gegeven een array gevuld met R, W, en B.
Reorganiseer dit array zo dat van links naar rechts eerst alle 'R', dan de 'W' en dan de 'B' komen te staan. Het algoritme moet lineair zijn en in situ (alleen interne verwisselingen). Het mag maar één doorgang door het array maken.

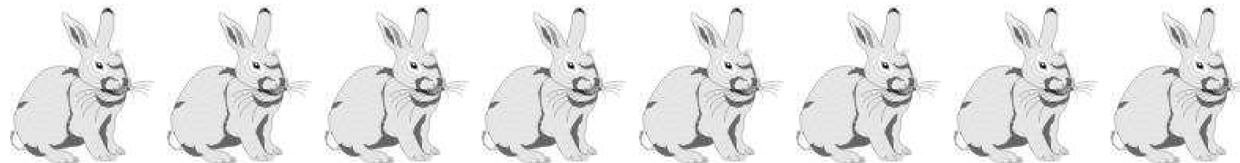
```
// invoer: array A[0...n-1] waarin A[i] = 'R', 'W' of 'B'
// uitvoer: array A[0...n-1] waarin eerst alle 'R' komen,
//          dan alle 'W', en ten slotte alle 'B'
r := 0;
w := 0;
b := n - 1;
while w <= b do
   if A[w] = 'R' then
      wissel( A[r], A[w] );
      r := r + 1;
      w := w + 1;
   else
      if A[w] = 'W' then
         w := w + 1;
      else  // A[w] = 'B'
         wissel( A[w], A[b] );
         b := b-1;
      fi
   fi
od
```

Voorbeeld: RRWWBBRWBRWB

Definitie Fibonacci-getallen:

$$\text{fib}(n) = \begin{cases} 0 & \text{als } n = 0 \\ 1 & \text{als } n = 1 \\ \text{fib}(n-1) + \text{fib}(n-2) & \text{als } n > 1 \end{cases}$$

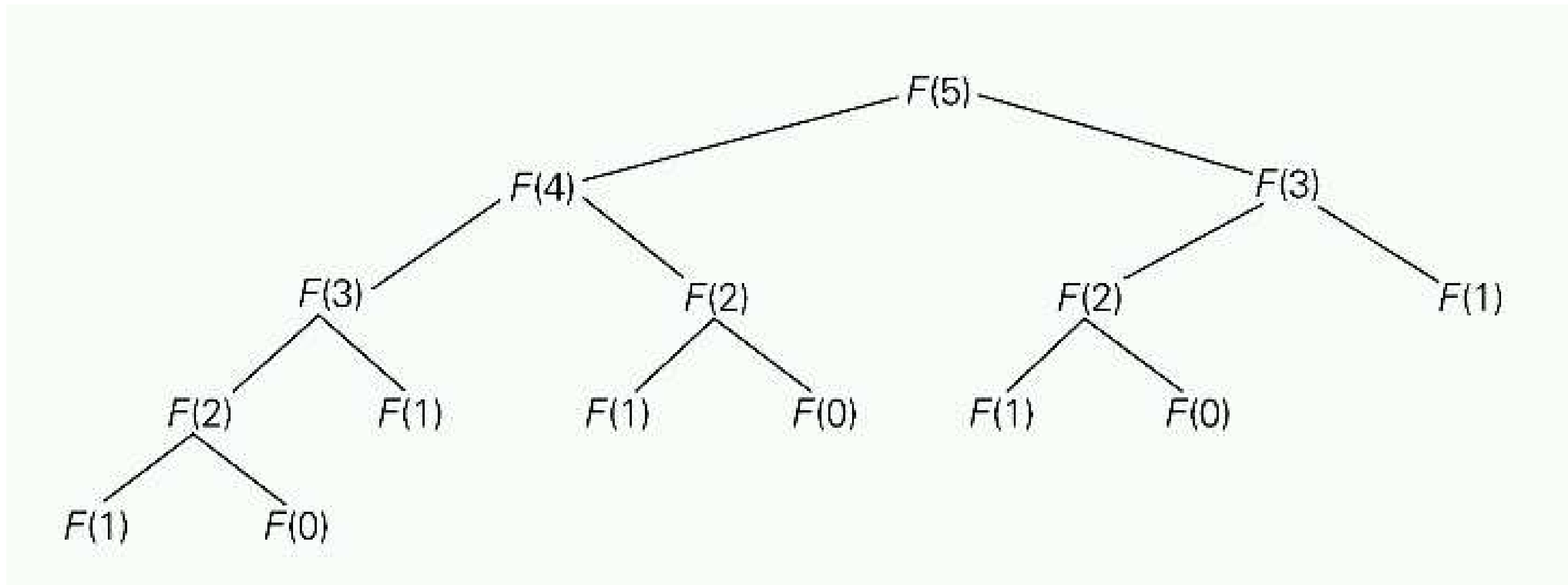
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946, ...



Recursieve C++-functie:

```
long fib1 (int n) {
   if ( ( n==0 ) || ( n == 1) )
      return n;  // gaat goed!
   else
      return ( fib1 (n-1) + fib1 (n-2) );
} // fib1
```

Watervaleffect



Voor $n = 5$ worden sommige recursieve aanroepen meerdere malen gedaan. Voor grotere waarden van n wordt dit watervaleffect steeds groter. Dit komt doordat deelproblemen elkaar overlappen.

Oplossing: gebruik een array om tussenresultaten op te slaan, en los op die manier elk deelprobleem precies één keer op.

Dit kan op twee manieren:

1. Top down: memory function
   Combineert recursie met het gebruik van een array

2. Bottom up: het klassieke dynamisch programmeren (DP)
   Vult het array van klein naar groot (for-loop)

```
const int MAX = 45;
long fib2 (int n) {  // recursie met array !
   static long memo[MAX] = {-1};  // eenmalig op -1
   if ( memo[n] > -1 ) // al eerder berekend
      return memo[n];
   else {
      if ( ( n==0 ) || ( n == 1) )
         memo[n] = n;  // gaat goed!
      else
         memo[n] = fib2 (n-1) + fib2 (n-2);
      return memo[n];
   } // else
} // fib2
```

Dynamisch programmeren: gebruikt ook een array voor het opslaan van tussenresultaten, maar werkt bottom up. Gebruikt de recurrente betrekking waaraan de Fibonacci-getallen voldoen.

```
fibo[0] = 0;
fibo[1] = 1;
for (i=2; i<=n; i++) {
    fibo[i] = fibo[i-1] + fibo[i-2];
}
return fibo[n];
```

Je hebt overigens niet het hele array nodig, maar je kunt volstaan met 3 (of zelfs 2) variabelen. Zo krijg je de bekende iteratieve oplossing (zie ook Programmeermethoden).

- nuttig bij problemen met *overlappende deelproblemen*

- druk een oplossing van het probleem uit in oplossingen van deelproblemen (*recursieve formulering*)

- deeloplossingen worden opgeslagen in een *tabel* zodra ze berekend zijn, waardoor elk deelprobleem maar *één keer* hoeft te worden opgelost

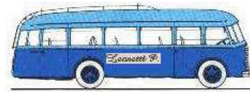
- na afloop bevat (of is) de tabel de oplossing van het oorspronkelijke probleem

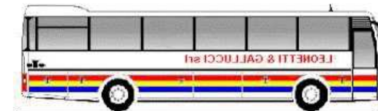
- DP is van oorsprong een *bottom up* methode: start met de kleine gevallen en combineer hun oplossingen tot oplossingen van steeds grotere gevallen

- er is ook een *top down* variant (*memory function*)

- de bottom up methode is *iteratief*, de top down variant is recursief
- bottom up lost *alle* deelproblemen op, top down alleen degene die echt nodig zijn voor het oplossen van het oorspronkelijke probleem
- bij beide varianten wordt eenzelfde soort tabel gebruikt
- bij bottom up wordt de tabel in een *bepaalde volgorde* gevuld, bij top down gebeurt dat meer willekeurig
- bij de bottom up manier is vaak een qua geheugengebruik *efficiënter* algoritme af te leiden

We willen een busreis maken langs steden $0, 1, 2, \ldots, n$, in die volgorde. Aangezien meerdere busmaatschappijen op de verschillende (deel)trajecten rijden, zijn de prijzen voor een rit van plaats i naar plaats j (via alle tussenliggende steden) per bus verschillend. Het kan dus voordeliger zijn om in plaats van rechtstreeks met de goedkoopste bus van plaats 0 naar n te reizen, (een paar keer) over te stappen en met een andere bus verder te gaan.





Laat `prijs`$[i][j]$, de prijs van het goedkoopste buskaartje rechtstreeks van i naar j, gegeven zijn voor alle $i \leq j$. Het probleem is nu om de prijs van de goedkoopste reis van 0 naar n te vinden.

Laat kosten(n) de prijs van de goedkoopste busreis van 0 naar n voorstellen, langs alle tussenliggende steden (in oplopende volgorde). Dan geldt:

$$\text{kosten}(n) = \begin{cases} 0 & \text{als } n = 0 \\ \min_{0 \leq k < n}(\text{kosten}(k) + \text{prijs}[k][n]) & \text{als } n \geq 1 \end{cases}$$

Voorbeeld:

$$\text{prijs} = \begin{pmatrix} 0 & 5 & 10 & 15 \\ - & 0 & 7 & 13 \\ - & - & 0 & 4 \\ - & - & - & 0 \end{pmatrix}$$

De prijs van de goedkoopste busreis van 0 naar 3 is hier...

Laat kosten(n) de prijs van de goedkoopste busreis van 0 naar n voorstellen, langs alle tussenliggende steden (in oplopende volgorde). Dan geldt:

$$\text{kosten}(n) = \begin{cases} 0 & \text{als } n = 0 \\ \min_{0 \le k < n}(\text{kosten}(k) + \text{prijs}[k][n]) & \text{als } n \ge 1 \end{cases}$$

Voorbeeld:

$$\text{prijs} = \begin{pmatrix} 0 & 5 & 10 & 15 \\ - & 0 & 7 & 13 \\ - & - & 0 & 4 \\ - & - & - & 0 \end{pmatrix}$$

De prijs van de goedkoopste busreis van 0 naar 3 is hier 14 (met tussenstop in plaats 2).

Een recursief algoritme:

```
kosten(n)::
    if n=0 then
       return 0;
    else
       temp := prijs[0][n];   // k = 0
       for k := 1 to n-1 do
           hulp := kosten(k) + prijs[k][n];
           if hulp < temp then
               temp := hulp;
           fi
       od
       return temp;
    fi .
```

De recursieve oplossing doet exponentieel veel aanroepen, en er is heel veel overlap tussen de deelproblemen. Oplossing: deeloplossingen opslaan in een geschikt array.

Laat kosten[i] de prijs van de goedkoopste busreis van 0 naar i voorstellen, langs alle tussenliggende steden (in oplopende volgorde). We zoeken dus kosten[n].
Dan geldt:

$$\text{kosten}[i] = \begin{cases} 0 & \text{als } i = 0 \\ \min_{0 \leq k < i}(\text{kosten}[k] + \text{prijs}[k][i]) & \text{als } i \geq 1 \end{cases}$$

We gaan het array nu bottom up vullen. Merk op dat om kosten[i] te berekenen, *alle* kleinere waarden kosten[k] met $k < i$ nodig zijn. Die moeten dus al eerder berekend zijn. We moeten het array derhalve van links naar rechts vullen.

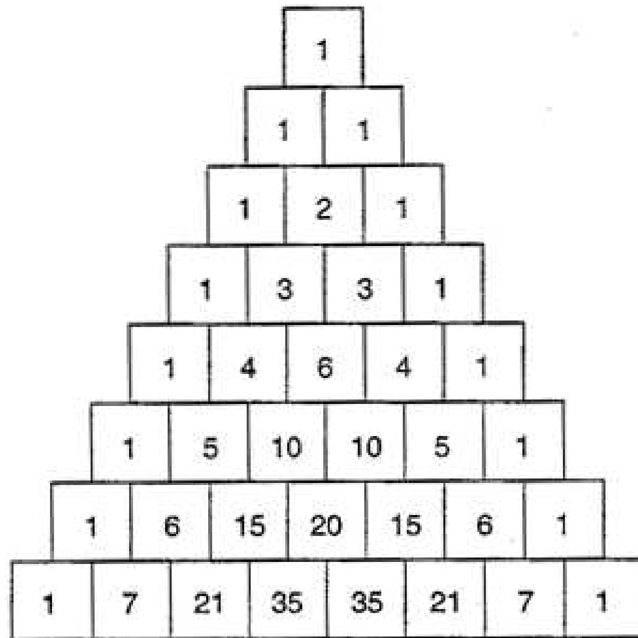
```
kosten[0] := 0;
for i := 1 to n do
    temp := prijs[0][i]; // met 1 bus, zonder overstappen
    for k := 1 to i − 1 do
        hulp := kosten[k]+prijs[k][i];
        if hulp < temp then
            temp := hulp; // goedkoopste tot dusver
        fi
    od
    kosten[i] := temp;
od
return kosten[n];
```

Het algoritme is eenvoudig zo aan te passen dat ook de tussenstops van de goedkoopste reis worden gevonden.

```
kosten[0] := 0; stop[0] := 0;
for i := 1 to n do
      temp := prijs[0][i]; tempstop:= 0; // met 1 bus
      for k := 1 to i − 1 do
            hulp := kosten[k] + prijs[k][i];
            if hulp < temp then
                  temp := hulp; // goedkoopste tot dusver
                  tempstop := k; // bijbehorende tussenstop
            fi
      od
      kosten[i] := temp; stop[i] := tempstop;
od
return kosten[n];
```

Hierin is stop[i] steeds de laatste tussenstop op de goedkoopste reis van 0 naar i.

Gegeven een pot met n verschillende objecten.
Doe in één keer een greep van k objecten uit de pot.
Hoeveel mogelijkheden?



Driehoek van Pascal

$$C(n,k) = \binom{n}{k} = \begin{cases} \binom{n-1}{k-1} + \binom{n-1}{k} & 0 < k < n \\ 1 & k = 0, n \end{cases}$$

Een recursief algoritme ligt voor de hand:

```
int bin1(int n,int k) {
    if ( ( k == 0 ) || ( k == n ) )
        return 1;
    else
        return ( bin1(n-1,k-1) + bin1(n-1,k) );
}
```

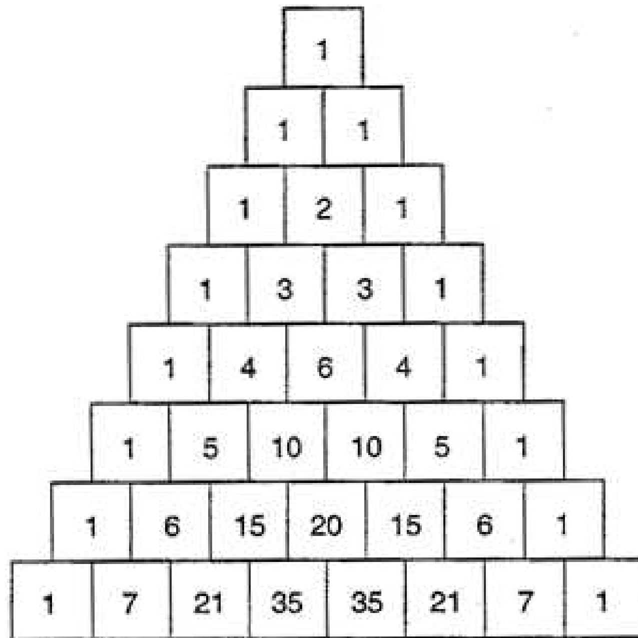
Veel van de `bin1(i,j)`'s worden echter herhaald berekend.

Aanroep: `bin1(n,k)`.
Complexiteit: $O(\binom{n}{k})$

Recursief algoritme met een array `C` voor het opslaan van tussenresultaten (dus `C[i][j]` $= \binom{i}{j}$):

```
int bin2(int n,int k) {
// C is globaal (foei) en op nul geinitialiseerd
    if ( C[n][k] != 0 )  // reeds eerder berekend
        return C[n][k];
    else {
        if ( ( k == 0 ) || ( k == n ) ) {
            C[n][k] = 1;
        else
            C[n][k] =  bin2(n-1,k-1) + bin2(n-1,k);
        return C[n][k];
    }
}
```

Complexiteit: $O(n * k)$; extra geheugen: $\Theta(n * k)$



Driehoek van Pascal

$$C(n,k) = \binom{n}{k} = \begin{cases} \binom{n-1}{k-1} + \binom{n-1}{k} & 0 < k < n \\ 1 & k = 0, n \end{cases}$$

We gebruiken een globaal (op nul geïnitialiseerd) array `C` voor het opslaan van tussenresultaten, dus `C[i][j]` $= \binom{i}{j}$.

	0	1	2	...	$k-1$	k
0	1					
1	1	1				
2	1	2	1			
⋮						
k	1					1
⋮						
$n-1$	1				$C(n-1, k-1)$	$C(n-1, k)$
n	1					$C(n, k)$

Bottom up:

Het array wordt rij voor rij gevuld, te beginnen bij rij 0, en per rij van links naar rechts, gebruikmakend van de recurrente betrekking (recursieve formulering).

```
int bin3(int n,int k) {
    for ( i = 0; i <= n; i++ )
        for ( j = 0; j <= min(i,k); j++ )
            if ( ( j == 0 ) || ( j == i ) )
                C[i][j] = 1;
            else
                C[i][j] = C[i-1][j-1] + C[i-1][j];
    return C[n][k];
}
```

Aanroep: `bin3(n,k)`.
Complexiteit: $\Theta(n * k)$; extra geheugen: $\Theta(n * k)$.
We kunnen hier echter volstaan met een een-dimensionaal array ter lengte k. Er is dus maar $O(k)$ extra geheugen nodig. (Zie ook exercise 8.1.9)

Knapzakprobleem

Gegeven n objecten, met gewicht $w_1, \ldots, w_n$ en waarde $v_1, \ldots, v_n$, en een knapzak met capaciteit W. **Gevraagd**: de meest waardevolle deelverzameling der objecten die in de knapzak past (dus met totaalgewicht $\leq W$). **Aanname**: gewichten zijn integers > 0.

Voorbeeld:

object	gewicht	waarde
1	8	42
2	3	14
3	4	40
4	5	27

knapzakcapaciteit 12

Laat $F[i][j]$ de waarde zijn van de meest waardevolle deelverzameling van de eerste i ($1 \leq i \leq n$) objecten, die in een knapzak met capaciteit j ($1 \leq j \leq W$) past. We zoeken dus $F[n][W]$. We nemen hier impliciet aan dat W een positief geheel getal is.

Laat $F[i][j]$ de waarde zijn van de meest waardevolle deelverzameling van de eerste i $(1 \leq i \leq n)$ objecten, die in een knapzak met capaciteit j $(1 \leq j \leq W)$ past. We zoeken dus $F[n][W]$. We nemen hier impliciet aan dat W een positief geheel getal is.

Dan geldt (want object i zit er wel of niet in):

$$F[i][j] = \begin{cases} \max\{F[i-1][j], v_i + F[i-1][j-w_i]\} & \text{als } j \geq w_i \\ F[i-1][j] & \text{als } j < w_i \end{cases}$$

En we definiëren:

$$F[0][j] = 0 \text{ voor } j \geq 0 \text{ en } F[i][0] = 0 \text{ voor } i \geq 0$$

We kunnen het array bijvoorbeeld rij voor rij (en per rij v.l.n.r.) vullen.

for $i := 0$ **to** n **do**
  for $j := 0$ **to** W **do**
    if $i = 0$ **or** $j = 0$ **then**
      $F[i][j] := 0;$
    else
      if $j < w_i$ **then**
        $F[i][j] := F[i-1][j];$
      else
        $F[i][j] := \max\ (F[i-1][j], v_i + F[i-1][j-w_i]);$
fi fi od od

		0	$j - w_i$	j	W
	0	0	0	0	0
	$i-1$	0	$F[i-1][j-w_i]$	$F[i-1][j]$	
w_i, v_i	i	0		$F[i][j]$	
	n	0			goal

Complexiteit: $\Theta(n * W)$; extra geheugen: $\Theta(n * W)$

Voor het voorbeeld wordt de tabel als volgt gevuld:

		j	→											
		0	1	2	3	4	5	6	7	8	9	10	11	12
i	0	0	0	0	0	0	0	0	0	0	0	0	0	0
↓	1	0	0	0	0	0	0	0	0	42	42	42	42	42
	2	0	0	0	14	14	14	14	14	42	42	42	56	56
	3	0	0	0	14	40	40	40	54	54	54	54	?	
	4													

Voor het voorbeeld wordt de tabel als volgt gevuld:

		j →												
		0	1	2	3	4	5	6	7	8	9	10	11	12
i	0	0	0	0	0	0	0	0	0	0	0	0	0	0
↓	1	0	0	0	0	0	0	0	0	42	42	42	42	42
	2	0	0	0	14	14	14	14	14	42	42	42	56	56
	3	0	0	0	14	40	40	40	54	54	54	54	?	
	4													

		j →												
		0	1	2	3	4	5	6	7	8	9	10	11	12
i	0	0	0	0	0	0	0	0	0	0	0	0	0	0
↓	1	0	0	0	0	0	0	0	0	42	42	42	42	42
	2	0	0	0	14	14	14	14	14	42	42	42	56	56
	3	0	0	0	14	40	40	40	54	54	54	54	56	82
	4	0	0	0	14	40	40	40	54	54	?			

- **Lezen/leren bij dit college:**
  slides, 8.inl., 8.2

- **Werkcollege:**

  donderdag 14 april 2016, 13:45–15:30, in B01/B02

- **Opgaven:**

  zie `http://www.liacs.leidenuniv.nl/~graafjmde/ALGO/`

- **Volgend college:**

  vrijdag 15 april 2016