

Achtste college algoritmiek

11 april 2013

Restant Decrease and conquer Dynamisch Programmeren

1

```
// 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
```

```

3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

Probleem: Gegeven twee niet-negatieve gehele getallen m en n (niet beide nul). Vraag: wat is de grootste gemeenschappelijke deler, genoteerd als $ggd(m, n)$, van m en n ?

Voorbeeld: $ggd(60,24) = 12$; $ggd(25,0) = 25$;
 $ggd(200, 441) = 1$; $ggd(588,495) = 3$.

Het algoritme van **Euclides** is gebaseerd op het herhaald gebruiken van de gelijkheid

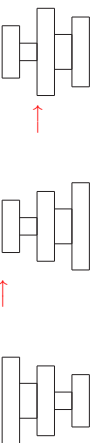
$$ggd(m, n) = ggd(n, m \bmod n),$$

totdat de tweede parameter nul wordt.

Voorbeeld: $ggd(60,24) = ggd(24, 12) = ggd(12, 0) = 12$

5

Probleem: Gegeven een stapel van n pannenkoeken, allemaal verschillend in grootte. Verder is alleen een spateel beschikbaar, die je onder een pannenkoek kan schuiven, waarna je de hele stapel daarbovenop in één keer kan om-draaien. De bedoeling is om uiteindelijk alle pannenkoeken bovenop elkaar te krijgen in volgorde van grootte (de grootste onderop).



Opgave 10: 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 ver-wisselingen). Het mag maar één doorgang door het array maken.

2

Variable-size decrease: de reductie in grootte is variabel, dus kan in elke stap anders zijn

Voorbeelden:

- Algoritme van Euclides
Dit is gebaseerd op: $ggd(m, n) = ggd(n, m \bmod n)$
- Flipping pancakes (Levitin, exercise 4.5.12)



- Binare zoekbomen

4

Probleem: Gegeven een stapel van n pannenkoeken, allemaal verschillend in grootte. Verder is alleen een spateel beschikbaar, die je onder een pannenkoek kan schuiven, waarna je de hele stapel daarbovenop in één keer kan om-draaien. De bedoeling is om uiteindelijk alle pannenkoeken bovenop elkaar te krijgen in volgorde van grootte (de grootste onderop).



6

BOUNDS FOR SORTING BY PEEK REVERSAL.
 William H. GATES
 Microsoft, Albuquerque, New Mexico
 Chaitin H. PAPADIMITROU†
 Department of Electrical Engineering, University of California, Berkeley, CA 94720, U.S.A.
 Received 18 January 1978
 Revised 26 August 1978

For a permutation π of the integers from 1 to n , let $f(\pi)$ be the smallest number of peeks for all π in the permutation group S_n . We show that $f(\pi) \leq 5n/6$, and that $f(\pi) \geq 7n/18$ for some permutation π . We also show that $f(\pi) \leq 5n/6$ for all π in the permutation group S_n if and only if n is a multiple of 6. The corresponding function $g(n)$ is defined to be $f(\pi) + 1$ if n is not a multiple of 6, and $f(\pi)$ otherwise. For example, consider any stack of pancakes. An adversary in advance knows $g(n)$.

1. Introduction
 We introduce our problem by the following quotation from [1].

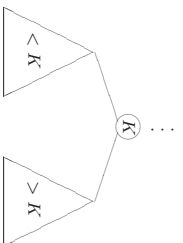
The chef in our dining room, and when he presents a stack of pancakes they come out all different sizes. Therefore, when I deliver them to a customer, or the way to the kitchen, I have to be careful to put them in the right order. I can do this by using a spatula and two hands by grabbing several from the top and flipping them over, repeating this until the stack is in the right order. I can also do this by using a spatula and one hand to flip the top pancake, and then repeat this until the stack is in the right order.

In this paper we derive upper and lower bounds for $f(n)$. Certain bounds were already known. For example, consider any stack of pancakes. An adversary in

7

8

Een **binaire zoekboom** is een binaire boom waarbij voor elke knoop geldt dat de waarde in die knoop groter is dan alle waarden in zijn linkersubboom, en kleiner dan alle waarden in zijn rechtersubboom.



9

Bekijk en vergelijk vier verschillende oplossingsmethoden voor het berekenen van a^n .

- Brute force:** gebaseerd op de definitie, $a^n = \underbrace{a * \dots * a}_n$
- Divide and conquer:** gebaseerd op $a^n = a^{\lfloor \frac{n}{2} \rfloor} * a^{\lceil \frac{n}{2} \rceil}$
- Decrease by one:** gebaseerd op $a^n = a^{n-1} * a$
- Decrease by a constant factor:** gebaseerd op

$$a^n = \begin{cases} (a^{\frac{n}{2}})^2 & \text{als } n \text{ even is} \\ (a^{\frac{n-1}{2}})^2 * a & \text{als } n \text{ oneven is} \end{cases}$$

11

Rekursieve 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



13

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:

- Top down:** memory function
Combineert recursie met het gebruik van een array
- Bottom up:** het klassieke dynamisch programmeren (DP)
Vult het array van klein naar groot (for-loop)

15

Bij het zoeken naar een waarde in een gewone binaire boom (bijv. WL(R) moeten in het slechtste geval alle n knopen bekeken worden. Zoeken in een binaire zoekboom is i.h.a. efficiënter: in het slechtste geval worden $h + 1$ knopen bekeken, met h de hoogte van de boom.

```
knopp* zoeken(knopp* root, int getal) {
    if ( root == null ) // lege boom
        return null;
    else
        if ( root->info == getal ) // gevonden!
            return root;
        else
            if ( getal < root->info )
                return zoeken(root->links, getal);
            else
                return zoeken(root->rechts, getal);
} // zoeken
```

10

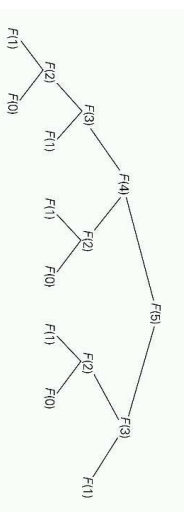
Definitie Fibonacci-getallen:

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

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



12



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.

14

```
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
```

16

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

```

fibonacci[0] = 0;
fibonacci[1] = 1;
for (i=2; i<=n; i++) {
    fibonacci[i] = fibonacci[i-1] + fibonacci[i-2];
}
return fibonacci[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**).

- de bottom up methode is *iteratief*, de top down variant is *recursief*
- bottom up lost *alle* deelproblemen op, top down alleen degenen 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

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

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

Voorbeeld:

$prijs = \begin{pmatrix} 0 & 5 & 10 & 15 \\ - & 0 & 7 & 13 \\ - & - & 0 & 4 \\ - & - & - & 0 \end{pmatrix}$	De prijs van de goedkoopste busreis van 0 naar 3 is hier...
---	---

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 .

```

- 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*)

We willen een busreis maken langs steden 0,1,2,...,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:

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

Voorbeeld:

$prijs = \begin{pmatrix} 0 & 5 & 10 & 15 \\ - & 0 & 7 & 13 \\ - & - & 0 & 4 \\ - & - & - & 0 \end{pmatrix}$	De prijs van de goedkoopste busreis van 0 naar 3 is hier 14 (met tussenstop in plaats 2).
---	---

De recursieve oplossing doet exponentieel veel aanroepen, en er is heel veel overlap tussen de deelproblemen. Opslossing: 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:

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

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.

25

- **Lezen/leren bij dit college:**
Sheets: Inleiding hoofdstuk 8
- **Werkcollege:**
donderdag 11 april 2013, 13:30–15:15, in zaal Noord-einde
- **Opgaven:**
www.llacs.nl/home/rvliet/algoritmiek/
- **Volgend college:** donderdag 18 april 2013
[hoorcollege in zaal Plein](#)

27

```

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.

26