Vierde college algoritmiek

2 maart 2007

Brute Force en Exhaustive Search

Brute force: a straightforward approach, usually directly based on the problem statement and definitions.

Ofwel: los een probleem op via de meest voor de hand liggende (recht-toe-recht-aan) methode, meestal door eenvoudigweg de definitie van een oplossing te gebruiken. Vaak ook: alle mogelijkheden proberen.

Voorbeeld 1: vind de grootste gemene deler van twee getallen m en n door van alle mogelijke integers ≥ 2 (en $\leq \min(m, n)$) te proberen of ze zowel m als n delen. (Zie college 1.)

Voorbeeld 2: los de DONALD + GERALD = ROBERT puzzel op door alle 9! (er was al gegeven dat D = 5) mogelijke antwoorden te proberen. (Zie college 1.)

Voorbeeld 3: zoek een gegeven X in een array van n stuks door er van links naar rechts doorheen te lopen en X met alle n te vergelijken. (Zie college 3.)

Zoek herhaald de kleinste en zet die op de juiste positie in het array.

```
for i := 0 to n − 2 do
    min := i;
    for j := i + 1 to n − 1 do
        if A[j] < A[min] then
            min := j;
        fi
    od
    wissel(A[i], A[min]);
od
```

Aantal vergelijkingen: $\frac{1}{2}n(n-1)$.

Probleem: bereken de waarde van het polynoom $p(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$ in het punt $x = x_0$ (Levitin, opgave 3.1.4)

Brute force algoritme (uit de definitie):

```
p := 0;
for i := n downto 0 do
    macht := 1;
    for j := 1 to i do
        macht := macht * x; // bereken x^i
    od
    p := p + a[i] * macht;
od
return p;
```

Efficiëntie (aantal $*/+$): $\Theta(n^2)$

Slimmer: we kunnen de efficiëntie eenvoudig flink verbeteren door van rechts naar links te evalueren en de x^i handiger te berekenen:

```
p := a[0];
macht := 1;
for i := 1 to n do
    macht := macht * x;
    p := p + a[i] * macht;
od
return p;
```

Efficiëntie: $\Theta(n)$;
Preciezer: $\#(*) = 2n$; $\#(+) = n$

Dit kan nog beter (methode van Horner), echter niet in orde van grootte.

Gegeven een patroon (= string van m karakters) en een text (= string van $n \geq m$ karakters). **Gevraagd** de index van de beginpositie in de text waar het patroon voorkomt.

Brute force algoritme: patroon v.l.n.r. langs de text schuiven en steeds de overeenkomstige karakters uit text en patroon vergelijken

```
for i := 0 to n − m do
    j := 0;
    while j < m and patroon[j] = text[i + j] do
        j := j + 1;
    od
    if j = m then
        return i;
    fi
od
return −1; // geen match gevonden
```

De werking van het algoritme geïllustreerd aan de hand van het volgende voorbeeld:

```
N  O  B  O  D  Y  -  N  O  T  I  C  E  D  -  H  I  M
N  O  T
   N  O  T
      N  O  T
         N  O  T
            N  O  T
               N  O  T
                  N  O  T
                     N  O  T
                        N  O  T
```

Het aantal vergelijkingen dat dit algoritme doet hangt af van de text en het patroon. In de worst case worden $m * (n - m + 1)$ vergelijkingen gedaan. Dit komt voor wanneer in elke i-stap het patroon helemaal (dus m vergelijkingen) vergeleken wordt met de text. De complexiteit van het algoritme is dus $O(n * m)$.

Opgave: geef een text en een patroon waarvoor het algoritme $m * (n - m + 1)$ vergelijkingen doet.

Opmerking: het kan beter (Boyer-Moore, Knuth-Morris-Prattt), maar voor "gewone-taal" texten is het algoritme zo slecht nog niet.

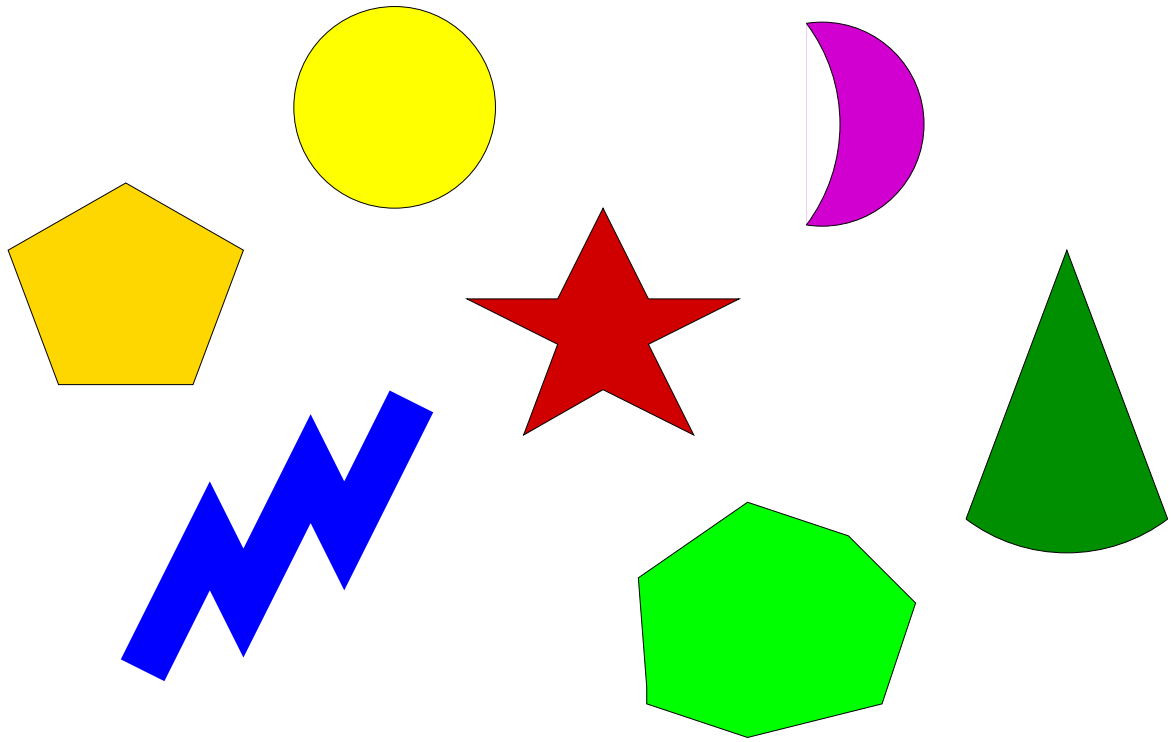
Gegeven n punten $p_1 = (x_1, y_1), \ldots, p_n = (x_n, y_n)$.
Gevraagd het/een tweetal punten dat het dichtst bij elkaar ligt. Afstandsmaat: $\mathrm{d}(p_i, p_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$.

Brute force algoritme: alle paren (p_i, p_j) (met $i < j$) aflopen en hun onderlinge afstanden $\mathrm{d}(p_i, p_j)$ vergelijken.

```
dmin := ∞;
for i := 1 to n − 1 do
    for j := i + 1 to n do
        d := (x_i − x_j)^2 + (y_i − y_j)^2;
        if d < dmin
            dmin := d; k := i; l := j;
        fi // (p_k, p_l) voorlopig closest pair
    od
od
```

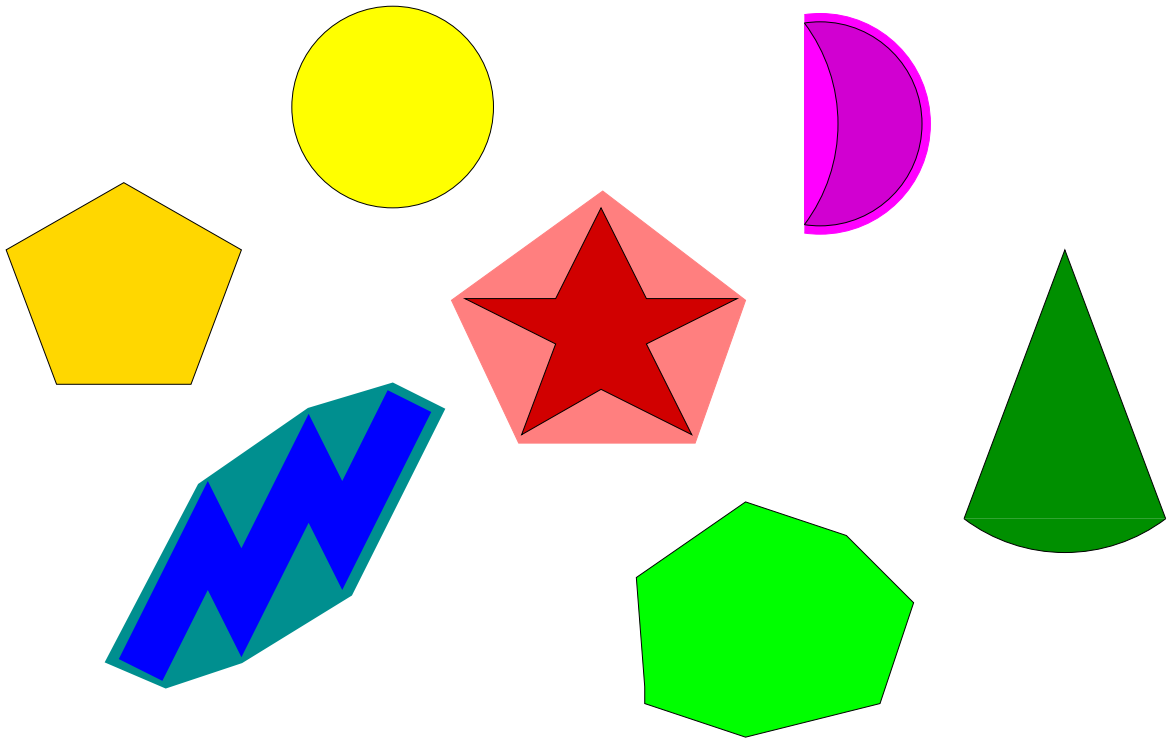
Complexiteit: $\frac{1}{2}n(n-1) = \Theta(n^2)$

Een verzameling punten in het platte vlak heet convex als voor elk tweetal punten uit die verzameling geldt dat het verbindend lijnstuk ook weer in die verzameling ligt.



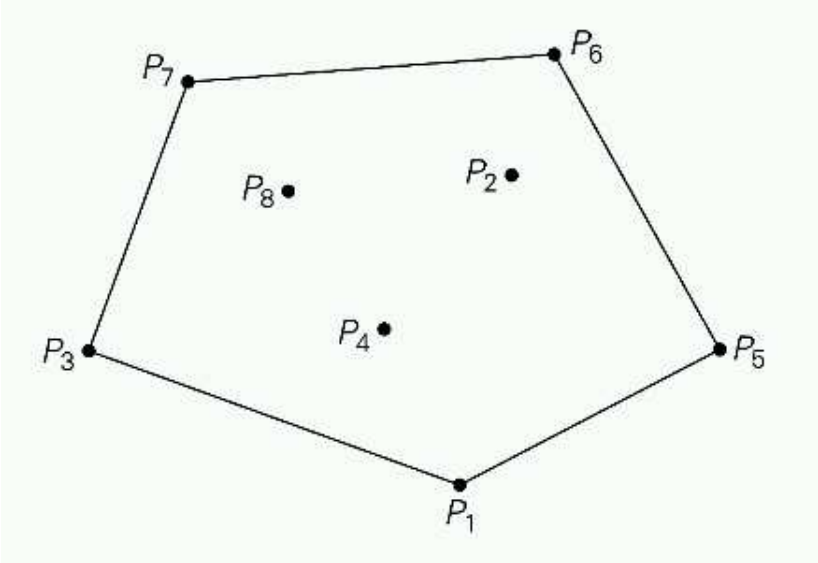
Convexe en niet-convexe vormen

De convex hull (convex omhulsel) van een verzameling S van punten in $\mathbf{R}^2$ is de kleinste convexe verzameling die S bevat.



Convexe omhulsels

Stelling: de convex hull van een verzameling S van $n > 2$ punten in $\mathbf{R}^2$ (niet alle op één lijn) is een convexe veelhoek (polygoon) met als hoekpunten enkele punten uit S.



De convex hull voor de verzameling $\{P_1, P_2, \ldots, P_8\}$ is de convexe veelhoek met hoekpunten P_1, P_5, P_6, P_7 en P_3

We baseren ons brute force algoritme op de volgende observatie: een lijnstuk P_iP_j maakt deel uit van de rand van de convex hull d.e.s.d.a. alle andere punten van de verzameling aan een en dezelfde kant van de lijn door P_i en P_j liggen.

Brute force: Ga *voor elke tweetal* punten $P_i = (x_i, y_i)$ en $P_j = (x_j, y_j)$ na of *alle andere punten* aan dezelfde kant van de lijn $(y_j - y_i)x + (x_i - x_j)y = x_iy_j - y_ix_j$ liggen. Zo ja, dan is P_iP_j dus deel van de convex hull.

Complexiteit: $O(n^3)$

Opmerking: het kan beter, namelijk $O(n^2)$

Brute force:

- **Voordelen**:
  - algemeen toepasbaar
  - eenvoudig
  - levert voor een aantal belangrijke problemen (zoeken, patroonherkenning) een zeer behoorlijk algoritme op
- **Nadelen**:
  - levert meestal geen efficiënt algoritme op
  - soms onacceptabel langzaam

Exhaustive search: een brute force benadering voor problemen die te maken hebben met het vinden van een element met een speciale eigenschap binnen een verzameling van bijvoorbeeld permutaties of deelverzamelingen of ...

Methode:

. construeer op een systematische manier alle kandidaat-oplossingen

. evalueer elk van deze mogelijke oplossingen

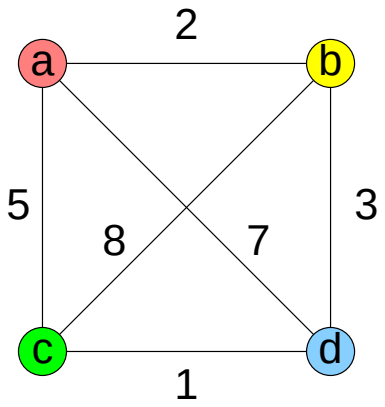
. retourneer de kandidaatoplossing met de gevraagde eigenschap (als die bestaat) (*)

(*) soms, zoals bij optimalisatieproblemen, *moet* je daartoe alle kandidaatoplossingen gezien hebben

Traveling Salesman Problem (handelsreizigersprobleem)

Gegeven n steden waarvan alle onderlinge afstanden bekend zijn. **Gevraagd**: de/een kortste route die elke stad precies één keer aandoet, en weer terugkeert in het vertrekpunt.
Ofwel: vind de/een kortste Hamiltonkring in een samenhangende gewogen (complete) graaf.

Voorbeeld:



minimale route
a b d c a
(of a c d b a)

Route	**Lengte**
a $\rightarrow$ b $\rightarrow$ c $\rightarrow$ d $\rightarrow$ a	$2+3+7+5=17$
a $\rightarrow$ b $\rightarrow$ d $\rightarrow$ c $\rightarrow$ a	$2+4+7+8=21$
a $\rightarrow$ c $\rightarrow$ b $\rightarrow$ d $\rightarrow$ a	$8+3+4+5=20$
a $\rightarrow$ c $\rightarrow$ d $\rightarrow$ b $\rightarrow$ a	$8+7+4+2=21$
a $\rightarrow$ d $\rightarrow$ b $\rightarrow$ c $\rightarrow$ a	$5+4+3+8=20$
a $\rightarrow$ d $\rightarrow$ c $\rightarrow$ b $\rightarrow$ a	$5+7+3+2=17$

Complexiteit: minstens $\Theta((n-1)!)$, immers alle $(n-1)!$ mogelijke Hamiltonkringen worden bekeken.

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 $\leq W$).

Voorbeeld:

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

knapzakcapaciteit 12

deelverzameling	gewicht	waarde
$\emptyset$	0	0
$\{1\}$	8	42
$\{2\}$	3	14
$\{3\}$	4	40
$\{4\}$	5	27
$\{1,2\}$	11	56
$\{1,3\}$	12	82
$\{1,4\}$	13	te zwaar
$\{2,3\}$	7	54
$\{2,4\}$	8	41
$\{3,4\}$	9	67
$\{1,2,3\}$	15	te zwaar
$\{1,2,4\}$	16	te zwaar
$\{1,3,4\}$	17	te zwaar
$\{2,3,4\}$	12	81
$\{1,2,3,4\}$	20	te zwaar

Complexiteit: minstens $\Theta(2^n)$, immers alle 2^n deelverzameling van n objecten worden bekeken.

Assignmentprobleem (toewijzingsprobleem)

Gegeven n personen en n taken (jobs). Persoon i kan taak j doen voor `kosten`$[i][j]$ euro. **Gevraagd**: de/een toewijzing van de personen aan de jobs (één persoon per job en één job per persoon) met minimale kosten.

Voorbeeld:

	job 1	job 2	job 3	job 4
Anna	9	2	7	8
Bob	6	4	3	7
Carla	5	8	1	8
David	7	6	9	4

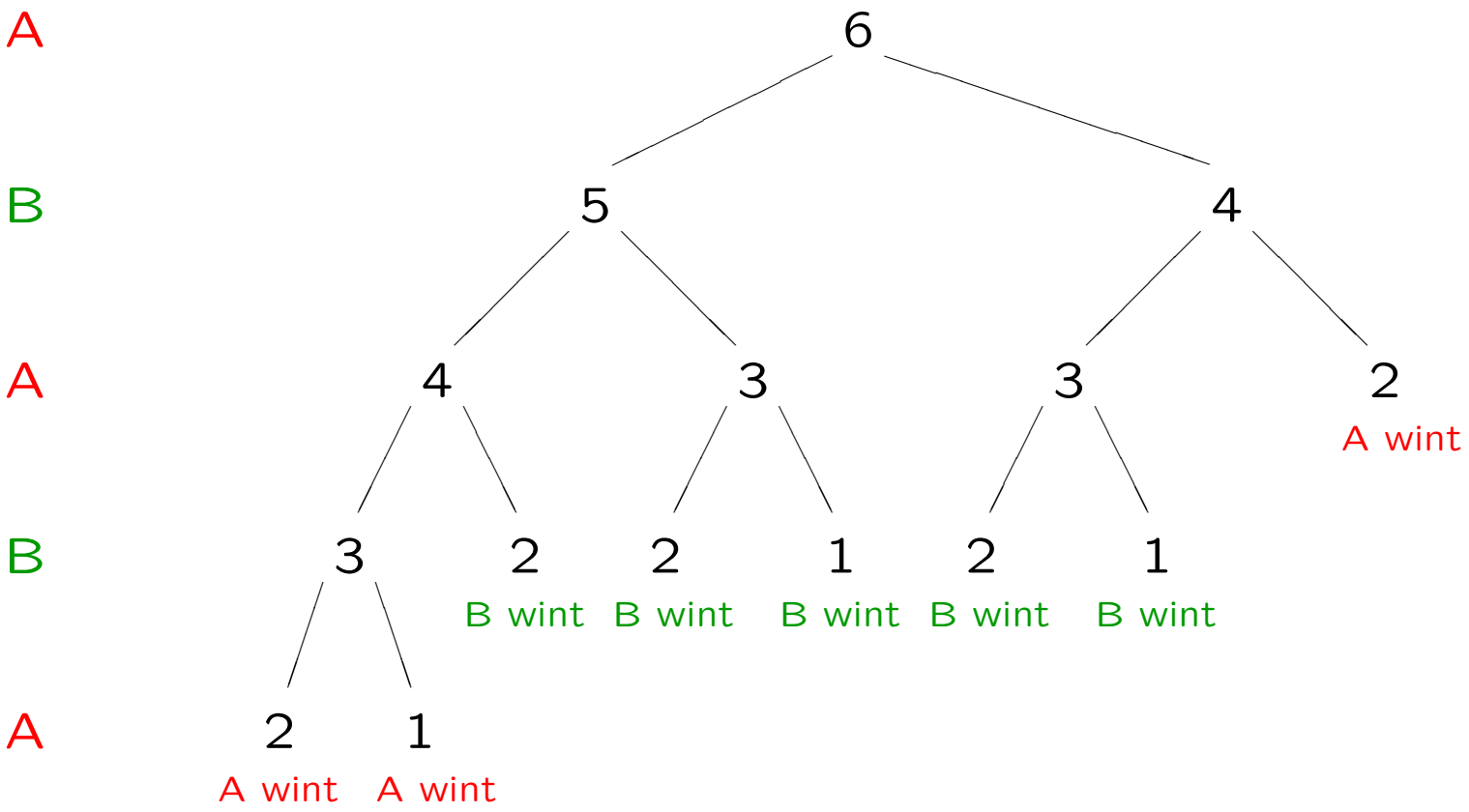
$n = 4$

	job 1	job 2	job 3	job 4
Anna	9	2	7	8
Bob	6	4	3	7
Carla	5	8	1	8
David	7	6	9	4

```
1,2,3,4 -> 9+4+1+4 = 18        2,3,1,4 -> ..        3,4,1,2 -> ..
1,2,4,3 -> 9+4+8+9 = 30        2,3,4,1 -> ..        3,4,2,1 -> ..
1,3,2,4 -> 9+3+8+4 = 24        2,4,1,3 -> ..        4,1,2,3 -> ..
1,3,4,2 -> 9+3+8+6 = 26        2,4,3,1 -> ..        4,1,3,2 -> ..
1,4,2,3 -> 9+7+8+9 = 33        3,1,2,4 -> ..        4,2,1,3 -> ..
1,4,3,2 -> 9+7+1+6 = 23        3,1,4,2 -> ..        4,2,3,1 -> ..
2,1,3,4 -> 2+6+1+4 = 13        3,2,1,4 -> ..        4,3,1,2 -> ..
2,1,4,3 -> 2+6+8+9 = 25        3,2,4,1 -> ..        4,3,2,1 -> ..
```

De goedkoopste toewijzing is hier `2,1,3,4`, met kosten 13.

Complexiteit: minstens $\Theta(n!)$, immers alle $n!$ mogelijke toewijzingen worden bekeken.



Exhaustive search: doorloop (als het ware) de hele spelboom om te bepalen of een stand winnend is. Alle toestanden worden zo bekeken.

Belangrijke observatie:

> een stand is winnend voor degene die aan de beurt is, dan en slechts dan als ten minste één van zijn directe vervolgstanden niet winnend is voor de tegenstander

$\Longrightarrow$ RECURSIE

Met terugzetten:

```
bool nimwinst (int stand) {
    int lucifer;
    if ( stand == 0 )
        return false;
        // de tegenstander heeft zojuist de laatste lucifers gepakt
    else {
        // directe vervolgstanden aflopen
        for ( lucifer = 1; lucifer <= 2; lucifer++ ) {
             stand -= lucifer; // doe een zet
             if ( !nimwinst (stand) ) {
                 stand += lucifer; // terugzetten
                 return true;
             }
             stand += lucifer; // terugzetten
        }
        return false;
    }  // else
}
```

Gebruik een kopie en doe daarin de zetten:

```
bool nimwinst (int stand) {
    int lucifer, kopie;
    if ( stand == 0 )
        return false;
        // de tegenstander heeft zojuist de laatste lucifers gepakt
    else {
        // directe vervolgstanden aflopen
        for ( lucifer = 1; lucifer <= 2; lucifer++ ) {
             kopie = stand;  // maak een kopie
             kopie -= lucifer; // doe een zet in de kopie
             if ( !nimwinst (kopie) ) {
                 return true;
             }
        }
        return false;
    }  // else
}
```

* Exhaustive search algoritmen werken **alleen voor kleine probleeminstanties** in acceptabele tijd

* Voor veel problemen zijn er veel efficiëntere algoritmen bekend (Eulerkring, kortste paden, toewijzingsprobleem)

* Voor andere problemen is exhaustive search (of varianten daarop) in essentie de enig bekende oplossing (handelsreizigersprobleem, knapzakprobleem)

- **Werkcollege** programmeeropdracht 1:

  donderdag 8 maart 2007 in computerzaal 306/308

- **Opgaven:**

  zie http://www.liacs.nl/home/graaf/ALGO/algo2007.html

- **Volgend college:**

  vrijdag 9 maart 2007