Tentamen Object-oriëntatie (NWI-MOL098)

Donderdag 7 april 2017, 8:30-11:30, LIN 4

Dit is een gesloten boek tentamen. In totaal kun je voor dit tentamen 54 punten halen.

1 Enkele kleine opgaven **totaal 14 punten**

We beginnen met enkele korte vragen. Voor onderdeel a, b en c kun je ieder 3 punten en voor onderdeel d 5 punten krijgen.

1.a) Wat is het verschil tussen een *instantievariabele* en een *lokale variabele*? Geef van beide een voorbeeld.

1.b) Bekijk de volgende declaratie:

```
private static final double ACCELERATION = 9.8;
```

Beschrijf **alle** zes onderdelen (de '=' en de ';' mag je achterwege laten) in deze declaratie kort en krachtig.

1.c) Wat is het doel van een constructor? In wat voor opzicht(en) onderscheidt een constructor zich van andere methodes? Noem 2 verschillen.

1.d) Bekijk de volgende methode:

```
private int calculateSomething ( int n, int m ) {
    int j = m;
    int i = 1;
    while ( i <= n ) {
        j = j - i;
        i++;
    }
    return j;
}
```

Wat is het resultaat van de aanroep `calculateSomething`$(6,42)$? Gebruik eventueel een tracing table. En wat is het resultaat van `calculateSomething` (a,b), uitgedrukt in a en b?

2 Complexiteit: graafalgoritmen **totaal 10 punten**

Stel we hebben een aantal steden die verbonden zijn door wegen. Voor de presentatie hiervan gebruiken we een graaf: De knopen in deze graaf komen overeen met de steden terwijl de wegen als verbindingen tussen knopen worden weergegeven. Iedere verbinding heeft een gewicht waarmee de lengte van de desbetreffende weg wordt aangegeven. Tijdens de colleges zijn verschillende minimalisatieproblemen behandeld die betrekking hadden op dit soort grafen. Een van de problemen was het zogenaamde *handelsreizigerprobleem*: zoek een pad in de graaf dat door alle knopen gaat waarvan de lengte minimaal is. Een ander probleem was het vinden van een *minimale opspannende boom*: geef in de graaf alle verbindingen aan die nodig zijn om vanuit iedere knoop elke andere te kunnen bereiken en wel zodanig dat de som van de aangegeven verbindingen minimaal is.

2.a) Geef voor één van beide problemen een oplossing in de vorm van een algoritme gespecificeerd door middel van een 'hoog niveau' flowchart (dat wil zeggen, een flowchart zonder veel details).

2.b) Wat is het fundamentele verschil tussen de beide problemen wanneer we de complexiteit van mogelijke oplossingen bekijken? Gebruik in je antwoord termen als Niet-Polynomiaal of Exponentieel.

3 Loops: bepaling van het massamiddelpunt **totaal 10 punten**

Gegeven de volgende representatie van 'n *deeltje*:

```
class Deeltje  {
    private Vector positie;
    private double massa;

    public Vector geefPositie () {
        return positie;
    }

    public double geefMassa () {
        return massa;
    }
}
```

3.a) Definieer een methode

```
public Vector massamiddelpunt (List<Deeltje> deeltjes)
```

die voor de lijst van deeltjes het *massamiddelpunt* bepaalt. Het massamiddelpunt $\vec{r}_C$ van een reeks van N deeltjes met massa's $m_0, m_1, \ldots, m_{N-1}$ en plaatsvectoren $\vec{r}_0, \vec{r}_1, \ldots, \vec{r}_{N-1}$ wordt gegeven door de volgende formule:

$$\vec{r}_C = \frac{1}{M} \sum_{i=0}^{N-1} m_i.\vec{r}_i$$

Hierin is M de som van de massa's van alle deeltjes.

Enkele `List`-methodes die wellicht van pas kunnen komen zijn:

```
int size ()
Object get(int index)
void remove (Object o)
boolean isEmtpy ()
```

Voor de klasse `Vector` mag je de volgende methodes veronderstellen:

```
Vector () // een parameterloze constructor die de nulvector als resultaat heeft.
void add( Vector other ) // telt de Vector other op bij deze vector
void sub( Vector other ) // trekt de Vector other af van deze vector
void scale( double v )   // vermenigvuldigt de lengte van deze vector met v
```

Hint: Begin met een nulvector en tel hier in een loop voor ieder deeltje $m_i.\vec{r}_i$ bij op. Gebruik voor het uitrekenen van $m_i.\vec{r}_i$ de `scale`-methode.

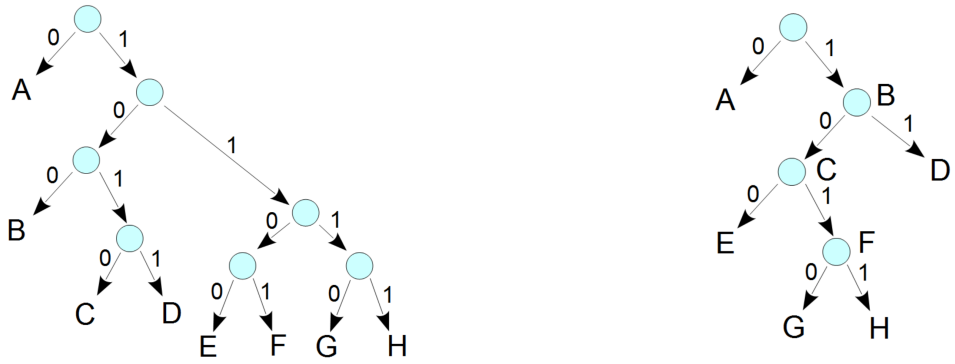
4 Compressie: Huffman-bomen **totaal 10 punten**

Stel we willen boodschappen versturen waarbij iedere boodschap is opgebouwd uit de acht symbolen A, B, C, D, E, F, G en H.

Als we verder geen informatie over de inhoud van de boodschap hebben en dus moeten aannemen dat ieder symbool even waarschijnlijk is, dan kunnen we het beste een *fixed-length codering* gebruiken: voor ieder symbool wordt hetzelfde aantal bits gebruikt:

4.a) Hoeveel bits per symbool hebben we dan nodig?

Als nu blijkt dat sommige symbolen vaker worden gebruikt dan andere dan kan het voordelig zijn om zogenaamde *variable-length codes* te introduceren, waarbij verschillende symbolen met een verschillend aantal bits kunnen worden aangegeven. In onderstaande plaatjes vind je zulke alternatieve representaties terug in de vorm van twee code-bomen.



(a) Huffman-boom

(b) Code-boom

Figuur 1: Compressie-bomen

In beide bomen kun je voor elk symbool een binaire code aflezen door de takken van de boom te volgen tot het betreffende symbool. De letter C, bijvoorbeeld, heeft in de linkerboom code `1010` en in de rechter code `10`. Het linkerplaatje is van een een zogenaamde *Huffman-boom*. Over deze boom gaan de volgende drie vragen.

4.b) Codeer/comprimeer de tekst: `BACADAEEFGAH`.

4.c) Hoeveel bits hebben we uitgespaard met deze alternatieve representatie t.o.v. de codering met fixed-length codes?

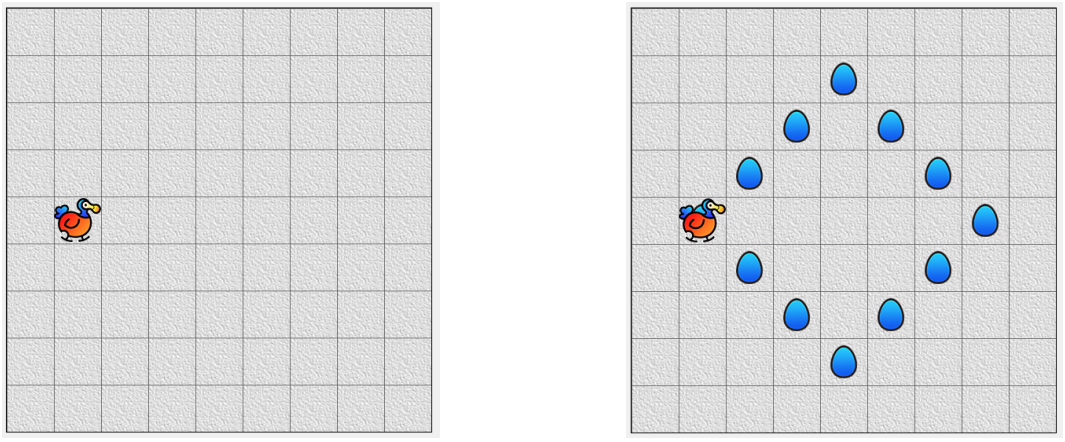
4.d) Decodeer/decomprimeer `101100100111001010010110`.

De laatste vraag heeft betrekking op de boom zoals weergegeven in het rechterplaatje.

4.e) Leg uit wat het probleem is met deze code-boom als we hem zouden gebruiken voor het gecomprimeerd versturen van boodschappen.

5 Algoritmen: Dodo tekent een diamant **totaal 10 punten**

Dodo staat in wereld (zie onderstaand linkerplaatje). Ze kijkt naar het oosten. Haar doel is om het eierenpatroon zoals getoond wordt in het rechterplaatje, te produceren. Opm: In het rechterplaatje ligt ook een ei in cel waar Dodo is gestopt.



(a) Beginsituatie (b) Eindsituatie

Figuur 2: Dodo maakt een diamant

5.a) Geef, in Java, een algoritme dat het rechterplaatje oplevert. Dit algoritme dient generiek te zijn, dat wil zeggen, de lengte van de zijde van de te produceren ruit dient als parameter te worden meegegeven. Hint: definieer eerst een hulpmethode `void diagonal( int length )` die één diagonaal met de opgegeven lengte produceert.

– EINDE TENTAMEN –