Tentamen
ORIENTATIE INFORMATICA
4 februari 2003
09.00 – 12.00 uur

Opmerkingen vooraf:

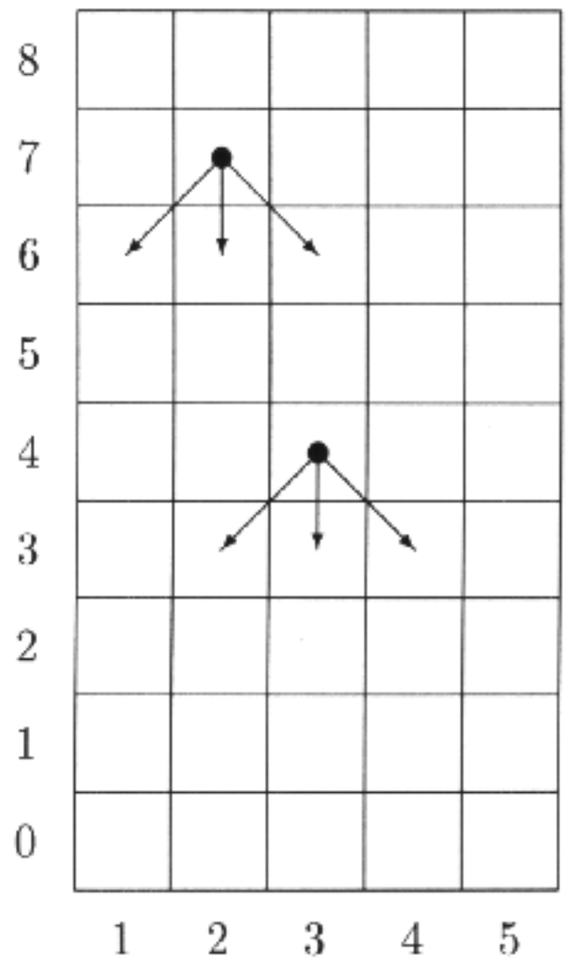
- geef bij elke Haskell-functie ook de typering.
- in elk onderdeel mag je gebruik maken van vorige onderdelen, ook als je niet in staat bent geweest deze te maken.
- dit tentamen bestaat uit de gedeelten A en B. Studenten Natuurkunde die een tentamenbriefje voor een 2-punten willen hebben, maken alleen deel A. De andere kandidaten maken zowel deel A als deel B.
  Reeds behaalde (deel-)resultaten tellen niet meer mee.

---

deel A

Opgave 1

In deze opgave bekijken we het volgende bordspel. Het bord bestaat uit een rechthoekig schema van vierkanten, vijf eenheden breed en een willekeurig aantal (> 0) lang. Het spel begint met het plaatsen van een pion op één van de vierkanten van de hoogst genummerde rij.
Een zet bestaat uit het verplaatsen van de pion op de manier zoals in de tekening hiernasst staat aangegeven: de pion moet een rij vooruit (richting rij 0) en daarbij mag hoogstens 1 kolom worden opgeschoven naar links of naar rechts. De pion mag niet naast het bord worden geplaatst.



In elk vierkant staat een positief geheel getal (niet in de tekening opgenomen). Als de speler zijn pion in een vierkant plaatst, dan wordt het daar genoteerde aantal punten aan zijn score toegevoegd.
Het spel is afgelopen als de pion is aangekomen in een vierkant in rij 0.

lees verder

De bedoeling van deze opgave is een functie te maken die bij een gegeven speelbord bepaalt hoeveel punten er maximaal gescoord kunnen worden. Het speelbord wordt gerepresenteerd door een functie

```
pnt :: Integer -> Integer -> Integer
```

De expressie (`pnt r k`) is het aantal punten dat staat genoteerd in het vierkant met rijnummer `r` en kolomnummer `k`.

[6 pt] ☐ 1. Geef een functie `grootste` die het grootste element uit een lijst met natuurlijke getallen oplevert. Beargumenteer daarbij duidelijk de waarde die je laat opleveren bij een lege lijst.

Gevraagd wordt een functie

```
maxScore :: Integer -> Integer -> Integer
```

met de betekenis: (`maxScore r k`) = het maximaal aantal te scoren punten als wordt begonnen in het vierkant met rijnummer `r` en kolomnummer `k`.

[4 pt] ☐ 2. Wat is een redelijke waarde voor (`maxScore r k`) voor het geval $r < 0$? Beargumenteer je keus.

[4 pt] ☐ 3. Wat is een redelijke waarde voor (`maxScore r k`) voor het geval $k < 1$? Beargumenteer je keus.

[4 pt] ☐ 4. Geef een implementatie voor `maxScore`.

[4 pt] ☐ 5. Leg uit wat we verstaan onder Dynamisch Programmeren.

[4 pt] ☐ 6. Leg uit waarom er mogelijk een efficiëntere implementatie bestaat die gebruik maakt van dynamisch programmeren.

[5 pt] ☐ 7. Geef een functie

```
maxScoreRij ::  Integer -> (Integer, Integer, Integer, Integer, Integer)
```

met de betekenis

```
maxScoreRij r  = ((maxScore r 1), (maxScore r 2), (maxScore r 3),
                                  (maxScore r 4), (maxScore r 5))
```

echter zonder `maxScore` zelf te gebruiken.

[2 pt] ☐ 8. Geef een functie die het maximaal aantal te scoren punten oplevert als functie van het nummer van de rij waarop mag worden begonnen.

lees verder

Opgave 2

In deze opgave bekijken we bomen van de volgende structuur. Elke (interne) knoop heeft twee subbomen en elk blad bevat een karakter. In Haskell modelleren we dit met
`data Boom = Blad Char | Knoop Boom Boom`

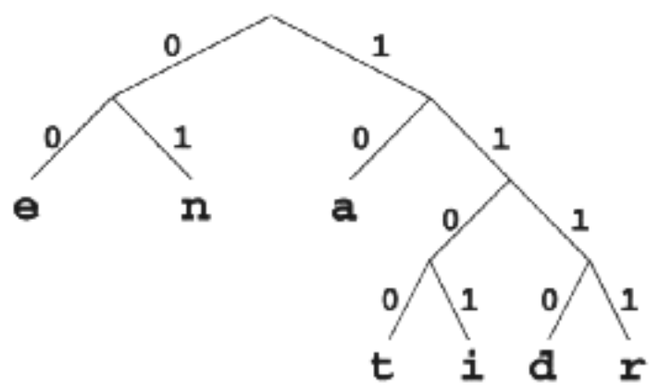
[6 pt] □ 9. Geef een Haskell-functie `inhoud :: Boom -> [Char]` die bij een boom een lijst met alle karakters uit de bladeren van die boom oplevert.

Bij de gebruikelijke ASCII-codering krijgt ieder karakter een (binaire) code van 7 bits. Vaak is het veel efficiënter om een codering te kiezen waarbij karakters die veel voorkomen een kortere code krijgen dan karakters die weinig voorkomen. Om een stuk tekst op die manier te coderen, wordt er eerst een telling uitgevoerd. Aan de hand van een frequentietabel wordt dan bepaald welke codering ieder karakter krijgt. Deze codering wordt opgeslagen in een code-tabel.

Bij ontvangst van een gecodeerde boodschap moet nu aan de hand van de code-tabel de oorspronkelijke inhoud weer worden gereconstrueerd. De bedoeling van deze opgave is om een functie te maken die deze decodeeractie uitvoert. Je mag er daarbij vanuit gaan dat de ontvangen gecodeerde boodschap correct is.

We nemen aan dat de code-tabel wordt gerepresenteerd door een boom, zoals bovenaan deze opgave is beschreven. Het pad van de wortel van de boom naar een karakter representeert de (binaire) code bij dat karakter. Een tak naar links representeert een `0`; een tak naar rechts een `1`. De volgende figuur geeft een voorbeeld. Links staat een code-tabel; rechts de bijbehorende boom.

karakter	code
e	00
n	01
a	10
t	1100
i	1101
d	1110
r	1111



[6 pt] □ 10. Geef een Haskell-functie `eerste :: (Boom, [Char]) -> (Char, [Char])` die bij een boom *b* die een code-tabel representeert en een lijst *lst* van `0`-en en `1`-en die een gecodeerde boodschap bevat, een paar oplevert met als eerste component het karakter dat in het beginstuk van *lst* is gecodeerd en als tweede component de rest van *lst*.
Bijvoorbeeld: met `b` de boom uit bovenstaande figuur en `lst = "11010001110010"` geldt
`eerste (b, lst) = ('i', "0001110010")`

[5 pt] □ 11. Geef een Haskell-functie `decodeer :: (Boom, [Char]) -> [Char]` zo, dat
`decodeer (b, lst)` de lijst `lst` decodeert volgens de code-tabel die door `b` wordt gerepresenteerd.
Bijvoorbeeld: met `b` en `lst` als in het vorige onderdeel geldt
`decodeer (b, l) = "ienta"`

deel B

Opgave 3

Deze opgave gaat over verzamelingen van integers en over verzamelingen van dergelijke verzamelingen. Een verzameling van integers representeren we in Haskell als een lijst. Aangezien er in een verzameling geen duplicaten voorkomen, mag je er vanuit gaan dat lijsten die verzamelingen representeren geen duplicaten bevatten.

Verzamelingen van verzamelingen (van integers) representeren we als lijsten van lijsten (van integers).

Laat $V = \{x_1, \ldots, x_n\}$ nu een verzameling zijn en $C = \{D_1, \ldots, D_m\}$ een verzameling van deelverzamelingen van V, dus voor elke $i \in \{1, \ldots, m\}$ geldt $D_i \subseteq V$. We noemen C een *exacte overdekking* van V als elk element van V in precies één element van C voorkomt.

Voorbeeld: met de volgende verzamelingen, waarbij de verzamelingen als Haskell-lijsten zijn gerepresenteerd,

```
V   =   [2, 5, 4, 7, 9, 1, 3]
C1  =   [[4, 1, 2], [7], [3, 9, 5]]
C2  =   [[4, 3], [7, 9, 2], [5, 3], [1]]
C3  =   [[9, 2, 4], [7, 3, 1]]
```

is $C1$ een exacte overdekking van V en zijn $C2$ en $C3$ geen exacte overdekkingen van V.

In een aantal stappen ontwikkelen we een functie die bij een verzameling V en een verzameling C van deelverzamelingen van V bepaalt of C een exacte overdekking is van V.

[3 pt] ☐ 12. Maak een functie `freq` die bij een integer `x` en een lijst `lst` van integers oplevert hoe vaak `x` voorkomt in `lst`.

[3 pt] ☐ 13. Geef een functie `maakLijst :: [[Integer]] -> [Integer]` die bij een lijst `lst` van lijsten een lijst oplevert die ontstaat door de elementen van `lst` achterelkaar te zetten. Bijvoorbeeld: met `C2` uit het vorige voorbeeld geldt

```
maakLijst C2 = [4, 3, 7, 9, 2, 5, 3, 1]
```

[3 pt] ☐ 14. Maak een functie `uniek` die bij twee lijsten `lst1` en `lst2` oplevert of elk element van `lst1` precies één keer voorkomt in `lst2`.

[3 pt] ☐ 15. Maak een functie `exOverd` die bij een verzameling `V` en een verzameling `C` van deelverzamelingen van `V` oplevert of `C` een exacte overdekking is van `V`.

[5 pt] ☐ 16. Laat n het aantal elementen van `V` zijn en m het aantal elementen van `C`. Beargumenteer dat de worst case tijdcomplexiteit van `exOverd` van $\mathcal{O}(n^2 \cdot m)$ is. Tel daarvoor het aantal vergelijkingen dat wordt uitgevoerd.

lees verder

Bekijk nu het volgende beslissingsprobleem

Exacte Overdekking / Exact Cover (XC)
Parameter: een eindige verzameling V en een verzameling M van deelverzamelingen van V
Gevraagd: is er een deelverzameling C van M die een exacte overdekking is van V?

[4 pt] ☐ 17. Laat weer `V = [2, 5, 4, 7, 9, 1, 3]` en laat

```
M = [[2, 5], [2, 7, 1], [4, 2, 3], [5, 4], [1, 9, 5], [9, 4, 3], [3, 9]]
```

Laat zien dat `(V, M)` een ja-instantie is van (**XC**).

[4 pt] ☐ 18. Bewijs dat (**XC**) $\in$ **NP**

[3 pt] ☐ 19. Er valt zelfs te bewijzen dat (**XC**) NP-volledig is.
Geef de definitie van NP-volledig.

[4 pt] ☐ 20. In het bewijs dat (**XC**) $\in$ **NPC** maakt iemand gebruik van het feit dat (**SAT**) $\in$ **NPC**.
Leg uit in welke richting de reductie moet gaan en wat de bewijsverplichtingen zijn.

Opgave 4

Gegeven is de volgende grammatica:

```
<S>  ::=   <A><S><B> | 'c'
<A>  ::=   'b'<A> | <A>'b' | 'a'
<B>  ::=   <A><A>
```

Het symbool `<S>` is het startsymbool.

[4 pt] ☐ 21. Beschrijf de strings die kunnen worden afgeleid uit het hulpsymbool `<A>`.

[4 pt] ☐ 22. Teken een eindige automaat die van de invoer bepaalt of ze kan worden afgeleid uit het hulpsymbool `<B>`.

[4 pt] ☐ 23. Geef een afleidingsboom (parse tree) bij de string `bbabacbaaabbab`.

[4 pt] ☐ 24. Is de grammatica ambigu? Bewijs je bewering.

[4 pt] ☐ 25. Beschrijf nauwkeurig de strings die in deze grammatica afgeleid kunnen worden.

[5 pt] ☐ 26. Is er een eindige automaat die van de invoer nagaat of ze kan worden afgeleid uit het startsymbool `<S>`? Bewijs je bewering.

[3 pt] ☐ 27. Is er een Turingmachine die van de invoer nagaat of ze kan worden afgeleid uit het startsymbool `<S>`? Beargumenteer je bewering.

einde

totaal: 110 punten.