# Tentamen
# ORIENTATIE INFORMATICA
## 22 november 2001
## 14.00 – 17.00 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. Deel B dient door iedereen gemaakt te worden.
  Heb je op de toets niet een resultaat 5 of hoger behaald, dan moet je ook deel A maken.
  Heb je op de toets een 5 of hoger gehaald, dan staat het je vrij om nu ook deel A te maken. We nemen in de berekening van het eindresultaat mee het hoogste van wat je op de toets heb gehaald en wat je nu weet te scoren op deel A.

---

# deel A

## Opgave 1

We beginnen dit tentamen met een eenvoudig spelletje voor twee personen. Op tafel ligt een stapel munten (zeg $n$) en om de beurt doen de spelers een zet. Een zet bestaat uit het wegnemen van 1 danwel 6 munten van de stapel. De speler die de laatste munt van tafel neemt is de winnaar.

Bijvoorbeeld:
Het beginaantal is 23 en speler 1 begint. Een mogelijk spelverloop is dan:

$$23 \xrightarrow{1} 22 \xRightarrow{6} 16 \xrightarrow{1} 15 \xRightarrow{1} 14 \xrightarrow{6} 8 \xRightarrow{6} 2 \xrightarrow{1} 1 \xRightarrow{1} 0$$

Speler 2 neemt de laatste munt en is daarmee de winnaar.

[einde voorbeeld]

In verband met wat nu volgt, formuleren we de winst-regel iets anders: winnaar is degene die de tegenstander confronteert met een stapel van 0 munten.

De bedoeling is nu om een Haskell-functie te ontwikkelen die bij een gegeven aantal munten oplevert of de speler die aan de beurt is een winnende stategie heeft. We zoeken dus een functie `win :: Integer -> Bool` met de betekenis:

`win n` = de speler die begint bij `n` munten heeft een winnende stategie

[7 pt] ☐ 1. Geef voor `n` $\in \{$`0, 1, 2, 3, 6, 7`$\}$ de waarde van `(win n)`. Licht je antwoorden toe.

[5 pt] ☐ 2. Leg uit dat voor 'grote' waarden van `n` geldt

```
win n = not (win (n - 1)) || not (win (n - 6))
```

[4 pt] ☐ 3. Hoewel in de context niet erg realistisch, is het voor de implementatie van de functie `win` handig ook de functiewaarde voor negatieve waarden van `n` vast te leggen. Beargumenteer welke waarde `(win n)` heeft voor negatieve waarden van `n`.

[4 pt] ☐ 4. Geef een Haskell-implementatie voor de functie `win`

[4 pt] ☐ 5. Leg uit wat we verstaan onder *Dynamisch Programmeren* en geef aan waarom een implementatie van de functie `win` die gebruik maakt van Dynamisch Programmeren waarschijnlijk een sneller algoritme oplevert.

[5 pt] ☐ 6. Geef, zonder daadwerkelijk gebruik te maken van de functie `win`, een Haskell-functie `hulpWin` met de betekenis

```
      (hulpWin n) =
( win n, win (n - 1), win (n - 2), win (n - 3), win (n - 4), win (n - 5))
```

[3 pt] ☐ 7. Gebruik `hulpWin` om een functie `effWin` te maken die een efficiëntere implementatie is van de functie `win`.

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`
Je mag er vanuit gaan dat alle in de boom opgenomen karakters verschillend zijn.

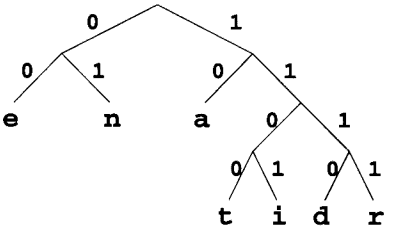
[6 pt] □ 8. Geef een Haskell-functie `inhoud :: Boom -> [Char]` die bij een boom een lijst met alle karakters uit de bladeren van die boom oplevert.

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. Zo'n 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 vanuit gaan dat de ontvangen gecodeerde boodschap correct is, dat wil zeggen dat de string bestaat uit de concatenatie van een aantal codewoorden.*

We nemen nu 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 figuur hiernaast geeft een voorbeeld.




[6 pt] □ 9. Geef een Haskell-functie `decodeOne` zodat `(decodeOne b lst) = (ch, rest)`. Hierin is `b` een boom die een code-tabel representeert en *lst* een lijst van `0`-en en `1`-en die een gecodeerde boodschap bevat. In het resultaat is `ch` het karakter dat in het beginstuk van `lst` is gecodeerd en is `rest` de lijst die ontstaat door de code van `ch` aan het begin van `lst` weg te laten.
Bijvoorbeeld: met `b` de boom uit bovenstaande figuur en `lst = "11010001110010"` geldt `(decodeOne b lst) = ('i', "0001110010")`

[6 pt] □ 10. Geef een Haskell-functie `decode` zodat `(decode 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 `(decode b l) = "ienta"`

Tot hier in totaal: 50 punten.

lees verder

# deel B

## Opgave 3

Deze opgave gaat over eindige verzamelingen van integers en over eindige lijsten 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. Als in onderstaande vragen wordt gesproken over een verzameling, dan bedoelen we dus zo'n lijst zonder duplicaten. Voorlopig veronderstellen we geen ordening in zo'n lijst.

[3 pt] □ 11. Geef een Haskell-functie `addElt` die een element toevoegt aan een verzameling.
Zorg er wel voor dat het resultaat weer een verzameling is en dat er dus geen duplicaten zijn.

[3 pt] □ 12. Onder de vereniging van twee verzamelingen $A$ en $B$ verstaan we de verzameling die bestaat uit elementen die in $A$ en/of in $B$ zitten.
Geef een Haskell-functie `setSum` die de vereniging van twee verzamelingen oplevert.

[5 pt] □ 13. Geef de worst-case tijdcomplexiteit van de functie `setSum`. Geef duidelijk aan wat je telt en druk je antwoord uit in de groottes van de twee verzamelingen. Het is niet voldoende een orde-grootte te geven!

We zoeken nu een Haskell-functie `minSupp` zo, dat voor een lijst `lst` van verzamelingen en twee verzamelingen `x` en `y` geldt:

```
minSupp lst x y = (ok, cnt)
```

Hier is `ok` een boolean die aangeeft of de verzameling `y` de vereniging is van de verzameling `x` en een aantal verzamelingen uit de lijst `lst`. Heeft `ok` de waarde `True`, dan is `cnt` het kleinste aantal verzamelingen uit `lst` dat aan `x` moet worden toegevoegd om `lst` te krijgen; heeft `ok` de waarde `False`, dan geldt `cnt = 0`.

Voorbeelden:
Met

```
lst = [ [1,2,3,4], [3,5,7], [2,4,6,8], [1,3,9], [],  ]
```

geldt

```
minSupp lst [1,5]   [1,3,5,7,9]   = (True,  2)
minSupp lst [5,8,9] [4,5,6,7,8,9] = (False, 0)
minSupp lst [1,3,9] [1,3,9]       = (True,  0)
```

[einde voorbeelden]

[4 pt] □ 14. Geef, met `lst` als in het voorbeeld, het resultaat van

```
(minSupp lst [2,3]   [1,2,3,4,5,6,7,8])
```

Beargumenteer uitvoerig je antwoord.

Gegeven is de volgende Haskell-implementatie van de functie `minSupp`:

```
minSupp :: [[Integer]] -> [Integer] -> [Integer] -> (Bool, Integer)

minSupp [] x y = (x == y, 0)
minSupp (v : lst) x y
  | not a2              = (False, 0)
  | not a1              = (True, 1 + c2)
  | c1 < 1 + c2         = (True, c1)
  | otherwise           = (True, 1 + c2)
      where (a1, c1) = minSupp lst x y
            (a2, c2) = minSupp lst (setSum v x) y
```

[5 pt] □ 15. Bovenstaande geeft, indien een aanvulling mogelijk is, het kleinste aantal elementen dat moet worden toegevoegd. Maar het geeft niet welke elementen dat zijn.
Geef een variant die als resultaat een drietal `(ok, cnt, sublst)` oplevert. Hierin is `sublst` een lijst van `cnt` elementen van `lst` die zo'n aanvulling realiseren.

We maken nu de stap naar het beslissingsprobleem *Minimale Overdekking.*

**Minimale Overdekking (Minimal Cover) (MINCOV)**
**Parameter:** een eindige lijst $L$ van eindige verzamelingen en een geheel getal $K$.
**Gevraagd:** bestaan er $N$ (met $N \leq K$) elementen van $L$ waarvan de vereniging gelijk is aan de vereniging van alle elementen van $L$?

[3 pt] □ 16. Leg uit wat een beslissingsprobleem is.
[3 pt] □ 17. Wanneer noemen we een beslissingsprobleem beslisbaar?
[3 pt] □ 18. Is (**MINCOV**) beslisbaar? Beargumenteer je antwoord; we vragen dus geen implementatie.
[3 pt] □ 19. Geef de definitie van de klasse **NP** .
[3 pt] □ 20. Beargumenteer dat (**MINCOV**) $\in$ **NP**.

Er valt zelfs te bewijzen dat (**MINCOV**) $\in$ **NPC**, gebruikmakend van het feit dat (**SAT**) $\in$ **NPC**.

[3 pt] □ 21. Leg uit in welke richting de reductie moet gaan en wat daarbij de bewijsverplichtingen zijn.
[3 pt] □ 22. Iemand beweert dat hij een algoritme van $\mathcal{O}(N^5)$ voor (**MINCOV**) heeft gevonden. Geef commentaar op deze claim.

# Opgave 4

In deze opgave bekijken we een fragment van de taal SQL (*Structured Query Language*), een taal om relationele databases mee te bouwen, te modificeren en te bevragen.

```
<query>      ::=  'SELECT' <attrib> 'FROM' <tabellijst> 'WHERE' <cond>
<attrib>     ::=  '*' | <veldlijst>
<veldlijst>  ::=  <naam> | <naam> ',' <veldlijst>
<naam>       ::=  <letter> | <letter><string>
<string>     ::=  <letter> | <letter><string>
                | <cijfer> | <cijfer><string>
<letter>     ::=  'a' ... 'z'
<cijfer>     ::=  '0' ... '9'
<tabellijst> ::=  <naam> | <naam> ',' <tabellijst>
<cond>       ::=  <naam> <comp> <string>
                | 'EXISTS' '(' <query> ')'
                | <cond> <logop> <cond>
                | 'NOT' <cond>
<comp>       ::=  '=' | '<' | '<=' | '>' | '>=' | '#'
<logop>      ::=  'AND' | 'OR'
```

Keywords (de terminale symbolen die met hoofdletters zijn geschreven) moeten van de andere elementen gescheiden worden door een of meer spaties en/of een regelovergang. Vanwege de leesbaarheid hebben we die niet in de productieregels opgenomen. De andere elementen, met uitzondering van de letters en cijfers in een naam of een string, mogen ook door spaties of regelovergangen van elkaar gescheiden worden.

[5 pt] □ 23. Geef een afleidingsboom bij de query

```
SELECT nm, nr
FROM   db
WHERE  ink >= 99999
```

[3 pt] □ 24. Deze grammatica is ambigu. Laat dat zien.

[4 pt] □ 25. Teken een eindige automaat die van de invoer nagaat of hij kan worden afgeleid uit het nonterminale symbool `<veldlijst>`.

[4 pt] □ 26. Is er een eindige automaat die nagaat of de invoer kan worden afgeleid uit het nonterminale symbool `<query>`? Bewijs de correctheid van je antwoord.

[3 pt] □ 27. Is er een Turingmachine die nagaat of de invoer kan worden afgeleid uit het nonterminale symbool `<query>`? Beargumenteer je antwoord.

einde

totaal: 110 punten.