Tentamen
ORIENTATIE INFORMATICA
3 februari 2006
09.00 – 12.00

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.
- Niet elke opgave telt even zwaar. Bij elk onderdeel staat aangegeven hoeveel punten er maximaal gescoord kunnen worden. Bij 100 of meer punten is het tentamenresultaat een 10. In andere gevallen is het tentamenresultaat het aantal punten gedeeld door 10.
- Studenten die de 3ec-variant van dit vak willen doen, dienen dit expliciet bovenaan het eerste in te leveren blad te vermelden. Een score van 50 punten levert voor hen de beoordeling 10.
  De items 1 tot en met 14 horen bij deze versie van het vak, maar het is toegestaan ook andere items uit te werken om op die manier de score te verhogen.

Opgave 1

De betrouwbaarheid van computersystemen laat helaas nog vaak veel te wensen over. Vooral bij systemen die min of meer autonoom beslissingen nemen (bijsturen van satellieten, afvuren van munitie) kan dat desastreuse gevolgen hebben.

Een manier om de betrouwbaarheid te vergroten is de volgende. In plaats van één computer, laat je een aantal verschillende computers de zelfde berekening uitvoeren, maar dan met verschillende algoritmen. De uitvoer van het systeem wordt nu bij meerderheid van stemmen bepaald (majority vote): de waarde die door een meerderheid van de computers is berekend wordt als het correcte resultaat beschouwd.

Wij vragen van je om schakelingen te bouwen die het meerderheids resultaat bepaalt uit drie respectievelijk vijf bitwaarden. De schakeling heeft dus drie (resp. vijf) ingangen en één uitgang. De waarde (0 of 1) van de uitgang is de waarde die op een meerderheid van de ingangen wordt aangeboden.

[4 pt] □ 1. Maak een tabel voor een schakeling met drie ingangen en één uitgang, waarbij de uitgang de majority vote is van de drie ingangen.

[4 pt] □ 2. Construeer op systematische wijze bij de tabel uit de vorige vraag een digitale schakeling. Als bouwstenen mag je gebruik maken van EN-, OF- en NIET-poorten, waarbij de EN- en OF-poorten willekeurig veel ingangen mogen hebben.

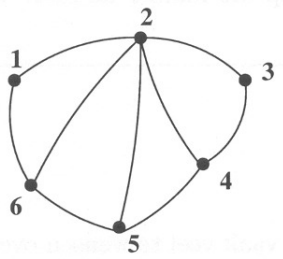
[3 pt] □ 3. Laat zien dat je ook een implementatie kunt maken die geen NIET-poorten gebruikt.

[3 pt] □ 4. Leg uit waarom de (deel-)schakeling waarin wordt gecontroleerd of alle drie de ingangen hoog zijn, wel uit de schakeling weggelaten kan worden.

[5 pt] □ 5. Construeer nu een schakeling die de majority vote voor vijf ingangen implementeert. Licht je werkwijze toe.

Opgave 2

Deze opgave gaat over (ongerichte) grafen. Een graaf $G = (V, E)$ bestaat uit een verzameling V van punten of knopen en een verzameling E van takken of kanten. We nemen aan dat de knopen zijn gelabeld met gehele getallen; een tak wordt gerepresenteerd door een paar van gehele getallen: de labels van de eindpunten van de tak. Daarbij nemen we aan dat in de representatie van een tak de laagst genummerde knoop vooraan staat.
De graaf representeren we door een lijst van knopen en een lijst van takken. In deze lijsten komen geen duplicaten voor. Je mag er vanuit gaan dat deze representatie 'gezond' is in de zin dat de knopen die in de lijst met takken voorkomen ook in de lijst met knopen voorkomen. Verder nemen we aan dat de graaf geen 'oren' heeft, dat wil zeggen dat er geen tak is van een knoop naar zichzelf.
Voorbeeld:



De graaf hiernaast kan worden gerepresenteerd door de lijsten

$V =$ `[1, 2, 3, 4, 5, 6]`

en

$E =$ `[ (1,2), (1,6), (2,3), (2,4), (2,5), (2,6), (3,4), (4,5), (5,6) ]`

[3 pt] □ 6. Eerst een vraag om er in te komen. Onder de graad van een knoop in een graaf verstaan we het aantal takken dat in die knoop samenkomt. Zo heeft in het voorbeeld hierboven de knoop 3 graad 2 en heeft knoop 2 graad 5.
Geef een Haskell functie `graad` die bij een knoop `k` en een takkenverzameling `e` de graad van `k` in `e` oplevert.

Onder een *knopenoverdekking* van een graaf $G = (V, E)$ verstaan we een deelverzameling W van V met de eigenschap dat van elke tak in E minstens één van de eindpunten element is van W. In het voorbeeld hierboven zijn `{1, 2, 3, 5}` en `{2, 4, 6}` knopenoverdekkingen, maar `{1, 3, 5}` niet.

We ontwikkelen nu een aantal functies, om uiteindelijk te komen tot een functie die bij een graaf een zo klein mogelijke knopenoverdekking oplevert.

[4 pt] □ 7. Geef een Haskell-functie

```
deleteNode :: Integer -> [(Integer, Integer)] -> [(Integer, Integer)]
```

die bij een knoop `k` en een lijst van takken `lst` de lijst van takken oplevert die ontstaat door uit `lst` alle takken te verwijderen die `k` als eindpunt hebben.

[4 pt] □ 8. Geef een Haskell-functie `deleteSet` die bij een lijst van knopen `w` en een lijst van takken `lst` de lijst van takken oplevert die ontstaat door uit `lst` alle takken te verwijderen die minstens één eindpunt in `w` hebben.

[4 pt] □ 9. Beredeneer wat de worst-case tijdcomplexiteit is van de functie `deleteSet`. Formuleer je antwoord in termen van de lengtes van de twee lijsten.

[3 pt] □ 10. Geef een functie `isCover` die bij een lijst van knopen `w` en een lijst van takken `lst` oplevert of `w` een knopenoverdekking is van `lst`.

De oplossing van de vorige vraag geeft bij een graaf $G = (V, E)$ de mogelijkheid om van een deelverzameling W van V na te gaan of W een knopenoverdekking is. We ontwerpen nu een functie die bij een graaf een zo klein mogelijke knopenoverdekking oplevert.

[3 pt] □ 11. Leg uit dat er bij elke graaf een knopenoverdekking bestaat.

[5 pt] □ 12. Bekijk de gedeeltelijke implementatie van `smallestCoverH`:

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

smallestCoverH ks []        = (True, [], 0)
smallestCoverH [] grf       = (False, [], 0)
smallestCoverH (k:ks)  grf
  | ....
          . . . . . .

    where (metB, metC, metI)          = smallestCoverH ks (deleteNode k grf)
          (zonderB, zonderC, zonderI) = smallestCoverH ks grf
```

De eerste parameter is een lijst van knopen; de tweede parameter is een lijst van takken. Het resultaat `(smallestCoverH w lst)` bestaat uit drie componenten. De eerste component is een boolean die aangeeft of `w` een deelverzameling `u` bezit die een knopenoverdekking van `lst` is. Zo ja, dat is de tweede component deze deelverzameling `u` en is de derde component het aantal elementen van `u`. Is de eerste component van het resultaat `False`, dan zijn de twee andere componenten irrelevant.

Vul de implementatie aan.
NB: in deze implementatie mogen geen aanroepen van `isCover` voorkomen!

[2 pt] □ 13. Geef een implementatie van de functie `smallestCover` die bij een knopenverzameling `v` en een takkenverzameling `lst` een kleinste knopenoverdekking oplevert.

[4 pt] □ 14. Wat is de worst-case tijdcomplexiteit van `smallestCover`? Beargumenter je antwoord.

Tot hier 51 punten

De functie `smallestCover` is een algoritme bij een optimaliseringsprobleem. We gaan nu over tot een verwant beslissingsprobleem:

Vertex Cover (VC)
Parameter: een graaf $G = (V, E)$ en een natuurlijk getal K
Gevraagd: heeft G een knopenoverdekking W met hoogstens K elementen?

[3 pt] ☐ 15. Leg uit wat we verstaan onder een beslissingsprobleem.
[3 pt] ☐ 16. Geef een `ja`-instantie van (**VC**)
[3 pt] ☐ 17. Geef een `nee`-instantie van (**VC**)
[5 pt] ☐ 18. Beargumenteer dat (**VC**) $\in$ **NP**.

Er valt zelfs te bewijzen dat (**VC**) $\in$ **NPC** door gebruik te maken van het feit dat (**SAT**) $\in$ **NPC**.

[3 pt] ☐ 19. Leg uit wat we verstaan onder de klasse **NPC** .
[4 pt] ☐ 20. In welke richting moet de reductie gaan? Beargumenteer je antwoord.
[5 pt] ☐ 21. Geef aan wat je bewijsverplichtingen zijn als je de NP-volledigheid van (**VC**) wilt bewijzen op grond van het feit dat (**SAT**) NP-volledig is.

Opgave 3

Gegeven is de volgende grammatica:

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

Het symbool `<S>` is het startsymbool.

[4 pt] ☐ 22. Beschrijf nauwkeurig de strings die kunnen worden afgeleid uit het hulpsymbool `<C>`.
[4 pt] ☐ 23. Teken een eindige automaat die van de invoer bepaalt of ze kan worden afgeleid uit het hulpsymbool `<B>`.
[5 pt] ☐ 24. Geef een afleidingsboom (parse tree) bij de string `aaaacabaaabbaba`.
[4 pt] ☐ 25. Is de grammatica ambigu? Bewijs je bewering.
[5 pt] ☐ 26. Beschrijf nauwkeurig de strings die in deze grammatica afgeleid kunnen worden.
[4 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: 103 punten.