Een grafe is kei_goed als alle knopen kei_bereikbaar zijn.

Een grafe is kei_nijg als alle knopen TEGELIJK kei_bereikbaar zijn (en je met andere woorden na een eindig aantal verplaatsingen van keien in elke knoop minstens 1 kei hebt).

Opgave is simpel: schrijf prolog-code die deze eigenschappen berekent.

een oplossing

N-queens

Opgave

Schrijf een genereer-en-test programma voor het **N-queens probleem**, ttz zet N koninginnen op een NxN schaakbord, zodat geen enkele koningin door een andere geslagen kan worden. Je mag gelijk welke voorstelling van een bord gebruiken, maar je levert als resultaat een lijst van rijen waarin opeenvolgende koninginnen staan: bv. nqueens(4, L) -> L = [2, 4, 1, 3] Deze oplossing zegt dat de koningin in kolom 1 op rij 2 staat, in kolom 2 op rij 4, ...

Te vinden oplossingen: [2, 4, 1, 3] ** [3, 1, 4, 2]

_____ ** _____

|__|K_|__|__| ** |__|__|K_|__|

|__|__|__|K_| ** |K_|__|__|__|

|K_|__|__|__| ** |__|__|__|K_|

|__|__|K_|__| ** |__|K_|__|__|

een oplossing

Celautomaat (19 juni 2006)

Opgave

Een cellulaire automaat is oneindig lang en bestaat uit een rij zwarte (aangeduid door x) en witte cellen (aangeduid door o). Alle cellen voor en achter degene die we in het geheugen hebben, zijn automatisch wit.

Een cellulaire automaat kan overgaan naar een volgende generatie. Hierbij is de nieuwe toestand van een cel enkel afhankelijk van de toestand in de huidige generatie van de linker-, rechter- en huidige cel. Hierbij wordt gebruik gemaakt van regels: [X,Y,Z,N] betekent dat een cel met waarde Y de waarde N krijgt als de linker - en rechtercel respectievelijk waardes X en Z hadden.

Er bestaat de zogenaamde "javel-regel": [o,o,o,o]. Deze garandeert dat de oneindige stukken witte band voor en achter het stuk band in het geheugen wit blijven.

Voorbeeld: *... o x o ...* wordt *... o x x o o ...* onder de regels *[o, o, x, x], [o, x, o, x], [x, o, o, o]* en de javelregel

1. Schrijf een predicaat *volgendegen(Seq, Regels, Volgende)* dat een volgende generatie berekent van *Seq* met de gegeven regels indien mogelijk en anders faalt. **Hint:** voeg 2 witte cellen toe voor en achter je invoer.

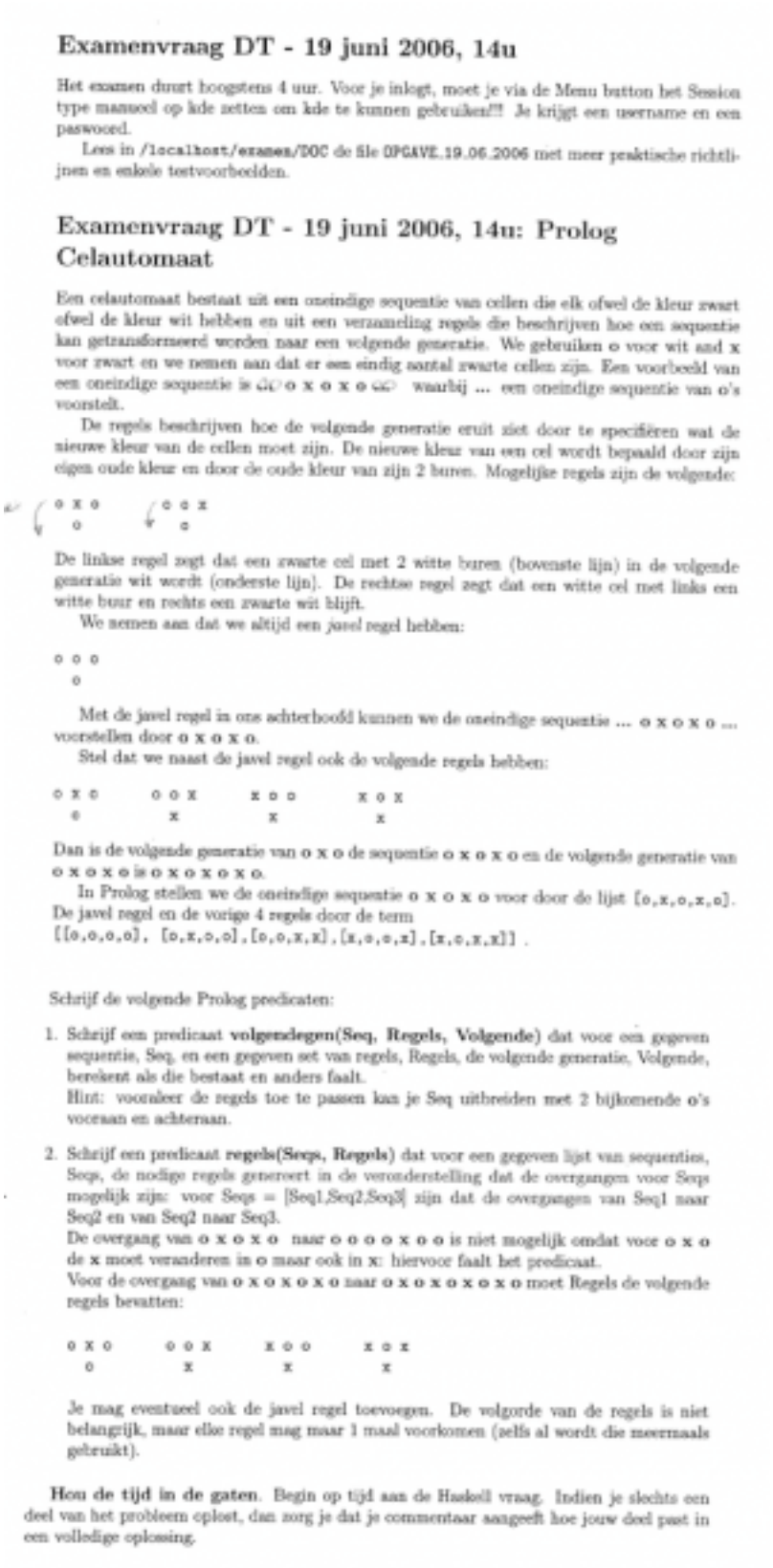
2. Schrijf een predicaat *regels(Sequenties, Regels)* Hierbij is *Sequenties* een lijst van achtereenvolgende generaties van de automaat. **Regels** moet geunificeerd worden met de regels die nodig zijn voor deze transformaties indien dit mogelijk is. De overgang van *o x o x o* naar *o o o o x o o* is niet mogelijk omdat voor *o x o* de x zowel in een o als een x moet veranderen. De regels blijven voor elke generatie-overgang dezelfde. De lijst van regels mag geen dubbels bevatten.

- Voorbeelden:
  - *regels([[o, x, x], [o, o, o, x, o], [o, o, o, o, o, o, o]], X)* geeft

[[x, x, o, x], [o, x, x, o], [o, x, o, o], [o, o, o, o]] (Opmerking: ik denk dat hier enkele regels ontbreken? Volgens mij moeten deze er nog bij: [o, o, x, o], [x, o, o, o], [o, x, o, o])

  - *regels([[o, x, o, x, o], [o, o, o, x, o]], X)* heeft geen geldig resultaat, omdat er geen consistente set regels voor deze overgang bestaat.

een oplossing



Examenvraag DT - 19 juni 2006, 14u

Het examen duurt hoogstens 4 uur. Voor je inlogt, moet je via de Menu button het Session type manueel op kde zetten om kde te kunnen gebruiken!?! Je krijgt een username en een paswoord.

Log in /localhost/examen/DOC de file OPGAVE.19.06.2006 met meer praktische richtlijnen en enkele testvoorbeelden.

Examenvraag DT - 19 juni 2006, 14u: Prolog

Celautomaat

Een celautomaat bestaat uit een oneindige sequentie van cellen die elk ofwel de kleur zwart ofwel de kleur wit hebben en uit een verzameling regels die beschrijven hoe een sequentie kan getransformeerd worden naar een volgende generatie. We gebruiken o voor wit and x voor zwart en we nemen aan dat er een eindig aantal zwarte cellen zijn. Een voorbeeld van een oneindige sequentie is ... o o o x o x o o o ... waarbij ... een oneindige sequentie van o's voorstelt.

De regels beschrijven hoe de volgende generatie eruit ziet door te specifiëren wat de nieuwe kleur van de cellen moet zijn. De nieuwe kleur van een cel wordt bepaald door zijn eigen oude kleur en door de oude kleur van zijn 2 buren. Mogelijke regels zijn de volgende:

```
o x o      o o x
  o          o
```

De linkse regel zegt dat een zwarte cel met 2 witte buren (bovenste lijn) in de volgende generatie wit wordt (onderste lijn). De rechtse regel zegt dat een witte cel met links een witte buur en rechts een zwarte wit blijft.

We nemen aan dat we altijd een *javel* regel hebben:

```
o o o
  o
```

Met de javel regel in ons achterhoofd kunnen we de oneindige sequentie ... o x o x o ... voorstellen door o x o x o.

Stel dat we naast de javel regel ook de volgende regels hebben:

```
o x o      o o x      x o o      x o x
  o          x          x          x
```

Dan is de volgende generatie van o x o de sequentie o x o x o en de volgende generatie van o x o x o is o x o x o x o.

In Prolog stellen we de oneindige sequentie o x o x o voor door de lijst [o,x,o,x,o]. De javel regel en de vorige 4 regels door de term
[[o,o,o,o], [o,x,o,o],[o,o,x,x],[x,o,o,x],[x,o,x,x]] .

Schrijf de volgende Prolog predicaten:

1. Schrijf een predicaat **volgendegen(Seq, Regels, Volgende)** dat voor een gegeven sequentie, Seq, en een gegeven set van regels, Regels, de volgende generatie, Volgende, berekent als die bestaat en anders faalt.
   Hint: vooraleer de regels toe te passen kan je Seq uitbreiden met 2 bijkomende o's vooraan en achteraan.

2. Schrijf een predicaat **regels(Seqs, Regels)** dat voor een gegeven lijst van sequenties, Seqs, de nodige regels genereert in de veronderstelling dat de overgangen voor Seqs mogelijk zijn: voor Seqs = [Seq1,Seq2,Seq3] zijn dat de overgangen van Seq1 naar Seq2 en van Seq2 naar Seq3.
   De overgang van o x o x o naar o o o o x o o is niet mogelijk omdat voor o x o de x moet veranderen in o maar ook in x: hiervoor faalt het predicaat.
   Voor de overgang van o x o x o x o naar o x o x o x o x o moet Regels de volgende regels bevatten:

```
o x o      o o x      x o o      x o x
  o          x          x          x
```

Je mag eventueel ook de javel regel toevoegen. De volgorde van de regels is niet belangrijk, maar elke regel mag maar 1 maal voorkomen (zelfs al wordt die meermaals gebruikt).

Hou de tijd in de gaten. Begin op tijd aan de Haskell vraag. Indien je slechts een deel van het probleem oplost, dan zorg je dat je commentaar aangeeft hoe jouw deel past in een volledige oplossing.

Haskell

E-Mailserver (18 januari 2018)

Noot: Het lijkt erop dat de focus bij Haskell tegenwoordig ligt op ADTs, het schrijven van enkele relatief eenvoudige functies, en een grote vraag in verband met I/O, dus zowat dezelfde structuur (en moeilijkheidsgraad) als bij de gequoteerde oefenzittingen. Er was ook een skelet gegeven met typesignaturen, maar dat stelde niet veel voor.

- Definieer een datatype voor een e-mail waarbij je de afzender, ontvanger, tijd van verzending, status gelezen/ongelezen, onderwerp, en inhoud voorstelt. Maak een instantie voor Show, Eq (mails die op hetzelfde ogenblik zijn verstuurd, zijn gelijk), en Ord, die vergelijkt volgens tijd van verzending.
- Definieer een datatype MailServer dat een lijst van e-mails bevat. Maak een instantiatie voor Show.
  - Implementeer een functie om alle mails van een bepaald tijdstip als gelezen te markeren
  - Implementeer een functie om aan een mailbox een bericht toe te voegen, tenzij er al een bericht inzit dat op hetzelfde tijdstip verstuurd is
- Implementeer voor elk van de volgende zoekoperaties over de server een functie die de zoekopdracht uitvoert:
  - alle berichten van een gegeven afzender, gesorteerd volgens oplopende verzendtijd
  - alle berichten naar een gegeven ontvanger, gesorteerd
  - alle ongelezen berichten naar een gegeven ontvanger, gesorteerd
  - alle berichten die tussen twee gegeven tijdstippen zijn verzonden, gesorteerd
  - alle berichten waarvan het onderwerp een gegeven substring bevat (Hint: gebruik isInfixOf uit Data.List)
  - alle berichten waarvan het lichaam een gegeven substring bevat
  - alle berichten waarvan het onderwerp of het lichaam een gegeven substring bevat (zorg dat er geen dubbels voorkomen)
- Schrijf een functie main_loop :: Time -> MailServer -> IO () die vraagt naar een commando en op basis vaan het antwoord een juiste actie kiest.
  - exit: druk "Exiting normally." af en beeindig de uitvoering van het programma.
  - send: vraag naar een zender, ontvanger, onderwerp, en inhoud, en voeg de mail toe aan de server. Die mail wordt verstuurd op het huidige tijdstip en is uiteraard nog niet gelezen. Herhaal dan de lus, met als tijdstip t+1
  - read: vraag naar een persoon en een tijdstip en print alle mails die op het gegeven tijdstip zijn gestuurd. Markeer deze als gelezen en herhaal de lus op het volgende tijdstip
  - search: vraag naar een zoekterm en druk alle mails af die die zoekterm in het onderwerp of lichaam bevatten. Herhaal de lus op het volgende tijdstip.
  - Voor andere commando's druk je een foutmelding af en herhaal je de lus zonder de tijd te verhogen.

De functie main was al gedefinieerd en riep main_loop op met als argumenten 0 (tijdstip) en een lege e-mailserver.

Foto's van ballen (16 januari 2013)

Gegeven een 1-dimensionale-ruimte met daarin een aantal ballen. Op tijdstip 0 nemen we er een foto van. Omdat we in een heel eenvoudige omgeving werken kunnen we de foto als volgt weergeven: [10,12] waarbij 10 en 12 de posities van bal 1 en bal 2 zijn. Wanneer de tijd 1 eenheid vooruitgaat, verplaatsen alle ballen precies 1 positie naar links of naar rechts. Ballen bewegen altijd in dezelfde richting. Ballen mogen nooit tegen elkaar botsen. Ook, elke keer de tijd vooruit gaat, wordt er 1 extra bal in de bak geplaatst. Vb: Foto 1 = [10,12] Foto 2 (na 1 stap) = [9,10,13]

Vraag 1: Schrijf een functie berekenMappings die als invoer 2 foto's krijgt en een int die het aantal tijdseenheden tussen het nemen van foto 1 en foto 2 voorstelt. Voor ons voorbeeld zou een mapping dus kunnen zijn [(10,9),(12,13)] omdat de bal die op positie 10 lag nu op positie 9 zou kunnen liggen, en de bal die op positie 12 lag nu op positie 13 zou kunnen liggen.

```
Main> berekenMappings [10,12] [9,10,13] 1
[[(10,9), (12,13)] ]

Main> berekenMappings [10,12] [9,11,13] 1
[[(10,9), (12,11)], [(10,9), (12,13)], [(10,11), (12,13)]]

Main> berekenMappings [10,11,12] [12,9,14,10,8] 2
[[(10,12), (11,9), (12,14)], [(10,12), (11,9), (12,10)], [(10,8), (11,9), (12,14)], [(10,8), (11,9), (12,10)]]
```

Vraag 2: Schrijf een functie die mappings tussen 2 foto's berekent voor alle tijdstippen. (ofzo)

Oplossing(en)

Winkelen (01.09.2010)

Jan en zijn vrouw Ann gaan winkelen, maar Jan doet dat niet graag. Jan mag daarom het pad kiezen. Hij zoekt het snelste pad. vb.

```
  _____10__________20__________30__________10_____
             |           |           |
             3           2           3
  ______5_____|____30_____|____15_____|_____8_____
```

Maak een functie helpJan dat als argument de informatie krijgt van de straat. In het geval van het voorbeeld is die informatie:

```
[Section 10 5 3, Section 20 30 2, Section 30 15 3, Section 10 8 0]
```

Het antwoord hiervan moet zijn:

```
[(Z,5),(B,3),(N,20),(B,2),(Z,15),(Z,8),(B,0)]
```

waar N, Z en B respectievelijk staan voor Noordkant, Zuidkant en Brug. Merk op dat het niet uitmaakt of je aan de zuidkant of de noordkant begint of eindigt. De laatste term (B,0) mag eventueel. worden weggelaten. Schrijf de nodige types.

Deze opgave is letterlijk overgenomen van **Learn You A Haskell**.

Oplossing(en)

Alternatieve ordes (15 januari 2009)

De Nederlande taalunie wil onze taal grondig hervormen, zelfs de klassieke alfabetische volgorde wordt overboord gegooid.

Je krijgt een lijst van woorden geordend volgens de nieuwe 'alfabetische' orde. Hieruit bepaal je het alfabet en de orde voor dat alfabet.

Schrijf hiervoor een functie berekenorde woordenlijst. Bijvoorbeeld voor de woordenlijst ab abd abc ba bd cc bereken je als mogelijke ordes a b d c en a d b c

Voor de woordenlijst 132 123 zijn de mogelijkheden 1 3 2 3 1 2 en 3 2 1.

```
Main> berekenorde ["ab", "abd", "abc", "ba", "bd", "cc"]
["abdc", "adbc"]

Main> berekenorde ["aa", "ba", "dc"]
["abdc","abcd","acbd","cabd"]

Main> berekenorde ["123", "132"]
["123","213","231"]
```

Zet in commentaar hoe je dit probleem aanpakt. Geef duidelijk aan welke types je gebruikt. Geef ook de typering van je functies.

In feite willen we een unieke orde hebben. Schrijf een functie maakuniek orde1 orde2 waarbij uitgaande van 2 mogelijke ordes we gaan detecteren wat de vrijheidsgraden nog zijn.

```
Main> maakuniek "abdc" "adbc"
[NogVrij 'b' 'd']

Main> maakuniek "abcd" "adbc"
Nogvrij b d, NogVrij c d

Main> maakuniek "abcd" "cabd"
Nogvrij a c, NogVrij b c
```

Oplossing(en)

Medicijnen (12 januari 2009)

Opgave

In deze tijd van het jaar zijn er veel zieken. De apothekers doen dus goed zaken. Nu moet er echter geleverd worden bij al die apothekers. Ook vervallen medicijnen moeten opgehaald worden. Een busje met een bepaalde capaciteit rijdt hiervoor rond. Schrijf een functie ronde die

- Als eerste argument de capaciteit van het busje krijgt
- Als tweede argument een lijst koppels (naam,hoeveelheid) die voorstelt hoeveel er geleverd moet worden in die bepaalde apotheek
- Als derde argument een lijst koppels (naam,hoeveelheid) die voorstelt hoeveel er afgehaald moet worden in die bepaalde apotheek
- Een route teruggeeft zodat:
  - Het busje in elke stad slechts 1 keer komt (je mag veronderstellen dat alle hoeveelheden kleiner zijn dan de capaciteit)
  - Het busje zo weinig mogelijk terug naar het depot moet gaan om bij te vullen (dus eigenlijk steeds zo vol mogelijk geladen is).

Je mag er hierbij steeds van uitgaan dat er eerst medicijnen in een apotheek afgeleverd worden en pas daarna de oude medicijnen ingeladen worden. Enkele voorbeelden ter illustratieac

```
ronde 5 [("a",3)] [("a",3)] = ["a","depot"]
```

```
ronde 5 [("a",3),("b",4),("c",2)] [("a",2),("b",4)] = ["a","c","depot","b","depot"]
```

```
ronde 10 [("a",9),("b",5)] [("a",3),("b",5),("c",7)] = ["a","c","depot","b","depot"]
```

(merk bij de voorgaande op dat ["c","a","depot","b","depot"] geen goede oplossing is)

```
ronde 5 [("a",3),("b",2)] [("a",2),("b",2),("c",1)] = ["a","b","c","depot"]
```

Oplossing(en)

Stern-Brocot boom van rationale getallen (25 januari 2008)

We kunnen de rationale getallen opsommen in een Stern-Brocot boom. Dit is een boom die we als volgt opstellen. Begin met de twee "getallen" 0/1 en 1/0 en neem 1/1 als wortel van de boom met "virtuele voorouders" 0/1 en 1/0. Dan kun je aan een knoop 2 kinderen toevoegen door het tussengetal te berekenen van deze knoop en zijn meest rechtse linkervoorouder en meest linkse rechtervoorouder voor linker- en rechterkind respectievelijk. Het tussengetal van twee breuken t1/n1 en t2/n2 is gedefinieerd als (t1+t2)/(n1+n2). Meer uitleg hierover vind je op **http://en.wikipedia.org/wiki/Stern-brocot_tree**

Schrijf een functie die deze boom gebruikt om een rationaal getal in een interval te vinden

```
>vindGetal 0.0 7.0
1/1
>vindGetal 0.45 0.55
1/2
```

Schrijf een functie die voor een gegeven rationaal getal een encodering opstelt via de Stern-Brocot boom door het pad dat we van de wortel tot dat getal moeten volgen.

```
>enc (3,5)
"LRL"
```

een oplossing

Algoritme van Dijkstra

Definieer eerst een data type Afstand dat de gehele getallen "bevat" en ook "oneindig". Voeg dat data type toe aan de type class Ord. (zoek de class declaratie van Ord op in de Prelude, dan weet je wat je moet definieren).

I.p.v. een rij Prologfeiten voor de bogen, krijg je twee functies:

boog :: String -> String -> Afstand

die voor twee knopen (die van het datatype String zijn) de lengte van de boog geeft tussen de twee knopen. Die kan "oneindig" zijn. Je mag veronderstellen dat (boog b1 b2) altijd gelijk is aan (boog b2 b1).

De tweede functie is

knopen :: [String]

die als resultaat een lijst met knopen geeft.

Definieer de functie dijkstra::String -> String -> (Afstand,Pad) die tussen twee gegeven knopen een tupel aflevert met als tweede component een kortste Pad en als eerste component de lengte van dat Pad.

Pad is gedefinieerd als type Pad = [String]

Je mag veronderstellen dat er een pad bestaat.

Het kortstepadalgoritme van Dijkstra

Gegeven twee verschillende knopen a en z in een samenhangende gewogen graaf G(V,E) met gewichten w voor bogen (x, y); de lengte van een kortste pad van a naar z wordt gevonden door algoritme:

1. Initialisatie: zet T gelijk aan V en definieer de afbeelding L op V door L(a) = 0 en L(x) = oneindig, voor alle x van V die verschillend zijn van a
2. Kies volgende: kies een v element van T met kleinste L(v) en haal v uit T
3. Herbereken L: voor alle knopen y in T die verbonden zijn met knoop v, zet L(y) = min(L(y),L(v) + w(v, y))
4. Test einde: indien z geen element van T stop anders ga naar Kies volgende.

een oplossing

een alternatieve oplossing

Graafisomorfie

Een graaf G(V,E) wordt bepaald door zijn vertices V en bogen (edges) E. Daarbij is E een deel van VxV, de gerichte bogen van G zijn van de vorm (v,w) met v en w in V.

De gerichte graaf G1(V1,E1) is isomorf met de gerichte graaf G2(V2,E2) indien er een bijectie b bestaat van V1 naar V2, die een bijectie f induceert van E1 naar E2 als volgt:

```
f((v,w)) = (b(v), b(w))
```

Schrijf een functie iso die twee isomorfe grafen als argumenten neemt en de knopenbijectie tussen de twee (de functie b van hiervoor) aflevert. Die knopenbijectie beschrijf je door een lijst van koppels waarbij elk koppel van de vorm (v,b(v)) is, dus de eerste component zit in V1 en de tweede in V2. Je moet expliciet nagaan dat de geinduceerde f een bijectie is natuurlijk.

De grafen worden gegeven als een tuple met als eerste component een lijst van de knopen, en als tweede component een lijst van de bogen. Een boog is een tuple met de twee eindpunten van de boog als component. De grafen zijn niet gericht.

Je functie moet polymorf zijn in het type van de knopen. Dus volgende 2

```
iso (["a","b"],[("a","b")]) (["c","d"],[("c","d")])
iso (["a","b"],[("a","b")]) ([1,2],[(1,2)])
```

moet kunnen - de eerste geeft bijvoorbeeld [("a","d"), ("b","c")] als antwoord en de tweede bijvoorbeeld [("a",1), ("b",2)]

Als er verschillende isomorfismen bestaan, dan maakt het niet uit welk er als antwoord gegeven wordt.

Als je daarmee klaar bent gebruik je de essentie van wat je al schreef voor het vervolg van de opgave: je gaat een functie schrijven die het graaf-subisomorfisme probleem oplost. De functie heet subiso en ze krijgt twee grafen als invoer. subiso bepaalt of de eerste isomorf is met een subgraaf van de tweede. Een graaf is een subgraaf van een andere als ze een deel van de bogen en knopen ervan heeft. subiso heeft als resultaat True of False al naar gelang ... Twee voorbeelden:

```
?- subiso (["a","b"],[("a","b")]) ([1,2],[(1,2)])
True
?- subiso (["a","b"],[("a","b")]) ([1,2,3],[(1,2),(2,3)])
True
```

want ([1,2,3],[(1,2),(2,3)]) bevat ([1,2],[(1,2)]) als deelgraaf en die is isomorf met (["a","b"],[("a","b")])

```
?- subiso (["a","b"],[("a","b")]) ([1,2,3],[])
False
```

want geen enkele deelgraaf van ([1,2,3],[]) is isomorf met (["a","b"],[("a","b")])

een oplossing

Boom betegelen (augustus 2005)

Opgave

We krijgen een boom en een hoop tegels waarmee de boom moet bedekt worden. De boom is nogal heterogeen (niet elke knoop heeft hetzelfde aantal kinderen) en daarom hebben we een declaratie:

```
data Boom t = Knoop1 t (Boom t) | Knoop2 t (Boom t) (Boom t) | Blad t
data Tegel t = Tknoop1 t (Tegel t) | Tknoop2 t (Tegel t) (Tegel t) | Tblad t | Leeg
```

Om een tegel op de top van een boom te kunnen leggen:

- moeten de overeenkomstige waarden gelijk zijn
- moet een Knoop1 met een Tknoop1, Tblad of Leeg overeenkomen
- moet een Knoop2 met een Tknoop2, Tblad of Leeg overeenkomen
- moet een Blad met een Tblad of Leeg overeenkomen

Op die manier is misschien niet alles van de boom bedekt door die tegel in de top en blijven er nog stukken boom over: elk van die stukken is zelf een boom die dan misschien weer getegeld worden in zijn top. Als er geen stukken meer overblijven hebben we een tegeling van de hele oorspronkelijke boom.

Een boom heeft dus een tegeling indien de top kan bedekt worden en indien elke deelboom die dan overblijft ook bedekt kan worden met die verzameling tegels. Je schrijft een functie tegel die als argumenten een boom krijgt en een lijst met tegels. Als resultaat geeft de functie True indien er een tegeling bestaat en False indien niet. Elke tegel mag 0, 1 of meerdere keren gebruikt worden.

2 oplossingen

Min-max

Opgave

maak een algoritme minimax die een spelboom doorloopt en aanpast:

```
spelboom = doosboom (id, [bolbomen], winst) | bolboom (id, [doosbomen], winst)
```

- id is voor iedere knoop verschillend (en integer)
- winst is in het begin een integer voor de bladeren (die zonder deelbomen) en "nog niet beslist" voor de andere knopen

bedoeling is om winst in te vullen bij het doorlopen van de boom, zodat in een doosboom het maxx van de sub-bol-bomen komt en in een bolboom het min van de sub-doos-bomen.

```
                    ["nog niet beslist"]
("nog niet beslist") ("nog niet beslist") ("nog niet beslist")
       [1][2]               [3][4]               [5][6]
```

--->

```
        [5]
 (1)    (3)   (5)
[1][2] [3][4] [5][6]
```

bijvraag: maak een algoritme dat het pad teruggeeft dat moet gevolgd worden bij optimale keuze:

```
                                [5]
                                ___
                     (1)        (3)         (5)
                                            ___
                   [1] [2]     [3][4]      [5][6]
                                             ___
```

een oplossing

Oneindige rij

Opgave

maak een oneindige rij van een datatype t.

geg: een variabele van dat type een functie die een rij van t's omzet naar een t waarbij iedere keer de functie wordt toegepast op de hele vorige rij.

bvb. infinite 1 sum (sum is een functie uit prelude, die de som neemt van alle elementen uit de rij die die meekrijgt)

```
[1,1,2,4,8,16,32,..]
```

ieder getal is de som van alle voorgaande

een oplossing

Formuleboom (juni 2005)

Opgave

(Ik probeer hier de vraag te reconstrueren ...)

Maak een datatype dat een logische formule voorstelt (En, Of, Wel, Niet)

Maak vervolgens ook een datatype dat een formule voorstelt als een boom:

1. Vertakking uit knoop 'a' naar links betekent "Wel a", vertakking naar rechts betekent "Niet a"

--edit: Ik denk dat je hier het net andersom bedoelt: links is "Niet a", rechts is "Wel a"

1. Als bladeren moet je "T" of "F" geven. T = True = "dit pad is geldig", terwijl F = False = "dit pad geldt niet"

Om dit wat visueler te maken:

```
          ___a___
         /       \
       _b_       _c_
      /   \     /   \
     T     F   T     F
```

betekent zoveel als: "[(Wel a) En (Wel b)] Of [(Niet a) En (Wel c)] --edit: Hier wordt het dan: "[(Niet a) En (Niet b)] Of [(Wel a) En (Niet c)]

Een boom kun je dus telkens tot een logische formule omvormen. Maak nu dus een procedure die voor een gegeven boom en een gegeven logische zin nagaat of de boom de zin al dan niet beslist.

Test met volgende regels:

```
Main> boom_is_formule tree1 form1
True
Main> boom_is_formule tree2 form1
False
```

een oplossing

Buurgraden (19 juni 2006)

Opgave

Een *Graaf t* wordt voorgesteld in dit soort notatie: *Graaf [Knoop naam1, Knoop naam2, ...] [Boog naam1 naam2, Boog naam3 naam4, ...]* In de graaf kunnen geen lussen voorkomen. Tussen elke 2 knopen kan hoogstens 1 boog zijn. Bogen zijn ongericht.

- De graad van een knoop is het aantal bogen dat uit die knoop vertrekt.
- De buurgraad van een knoop is een lijst met daarin de graden van de buren. Bijvoorbeeld *[4,2,2]*. De buurgraad is gesorteerd van groot naar klein.
- De buurgraad van een graaf is een lijst koppels van de buurgraad van een knoop, en de lijst van alle knopen met die buurgraad. Het ziet er uit als dit: *[([4,2,2],['a','b']), ([3,2],['c']), ([2,1],['d','e'])]*
- De buurgraad is gesorteerd, waarbij buurgraden met meer elementen eerst komen en bij even lange buurgraden gewoon de grootste buurgraad wordt genomen (dus *[x,y,z]* komt voor *[u,v]]', en* [4,1,1] *komt voor* [3,2,2]*)*

1. Schrijf een functie om de buurgraad van een graaf terug te geven.
2. Indien 2 grafen isomorf zijn, dan zullen er voor elke knoop-buurgraad evenveel knopen in beide grafen zijn. Schrijf een functie *nooit_isomorf :: (Graaf t) -> (Graaf t) -> Bool* die False teruggeeft indien dit het geval is.

Tip van de persoon die dit examen gehad heeft: zoek eens in de prelude naar *sortBy*

> Je mag ook de module List importeren die een functie sort bevat.
> sort::(Ord a) => [a] -> [a]


Examenvraag DT - 19 juni 2
Mogelijks Isomorfe Grafen

Voor een graaf $G(V, E)$ is V de verzameling kn
van een knoop v in $G(V, E)$ is het aantal bogen
dat de graaf geen lussen bevat. De buurgraa
de graden van de buren van een knoop.

Figure 1: Gra

In graaf G heeft de knoop a 3 buren, namel
De buurgraad van knoop a is [3,2].
Schrijf een functie buurgraad die als argu
urgraad van die graaf teruggeeft. De buurgra
knopen-met-die-buurgraad) gesorteerd volgens
Haskellvorm [([3,3,2,2],['d']), ([4,2,2
([3,3],['h']), ([3,2],['b','f'])].
De Haskellvorm van graaf G is '
'a' 'b', Boog 'b' 'c', Boog
Boog 'g' 'h', Boog 'h' 'a', E
Definieer in Haskell het type voor een graaf
alleen Char als representatie van knopen moge
die je schrijft.
Hint: Je mag de module List importeren die
Daarvoor zet je in het begin van je programma:

```
import List
-- sort :: (Ord a) => [a] -> [a]
-- zie ook de manual sectie over de  mo
```

De grafen $G(V_1, E_1)$ en $H(V_2, E_2)$ zijn isom
zodanig dat $e : E_1 \rightarrow E_2$ waarbij $e((v, w)) = ($
$e((v, w)) \in E_2$ en e een bijectie is. Je kan de n
de knopen uit V_1 en dan testen of de grafen g
$n!$ hernoemingen uit proberen. Je kan efficiën
equivalentie klassen op te delen en daarmee reh
Schrijf een functie `niet_iso_graaf`, die 2
als de 2 grafen onmogelijk isomorf zijn. Deze fu
de grafen in partities op te delen en geeft Tru

**grafen uitsluiten: om eventueel isomorf te zij
zijn met dezelfde buurgraden. Voor de grafen
False terug. Indien uit graaf H bijvoorbeeld d
het resultaat.**

**Indien je slechts een deel van het probleem
hoe jouw deel past in een volledige oplossing.**

een oplossing

N-queens

Opgave

N-queensin haskell. Geef 1 of alle oplossingen.

Een oplossing

Paginering (ex-examenvraag van Toledo)

Opgave

Paginering wordt gebruikt om de virtuele adresruimte van een programma x te mappen naar het fysisch werkgeheugen van onze computer. De fysische adresruimte wordt opgedeeld in n paginaâ€™s, 0, ... n âˆ’ 1. Wanneer de uitvoering van een programma toegang nodig heeft tot een fysisch adres, wordt de overeenkomstige pagina uit het virtuele werkgeheugen gehaald. Het virtuele werkgeheugen kan maximaal m paginaâ€™s bevatten, waarbij m < n. De pagineringstabel houdt bij of een fysische pagina in het werkgeheugen zit en zo ja in welke virtuele pagina van het werkgeheugen, 0,..., m âˆ’ 1.

Initieel is het virtuele werkgeheugen leeg. Wanneer het virtuele werkgeheugen vol is, moet een oude pagina plaats maken voor een nieuwe. Om te bepalen welke pagina eruit moet, bestaan er verschillende aanpakken. Wij gebruiken â€œleast recently usedâ€ (lru): de pagina die het langst niet gebruikt is, wordt verwijderd wanneer het virtuele werkgeheugen vol is. Je kan dit implementeren door de tijd van gebruik te registeren.

Schrijf een functie lru die als argumenten m, n, en een lijst van fysische paginanummers krijgt. De lijst van paginaâ€™s is de sequentie van paginaâ€™s die tijdens de uitvoering van het programma gebruikt worden. Voor een werkgeheugen van grootte 3 (m gelijk aan 3) en de lijst [2,3,2,1,5,2,4,5,3,2,5,2], evolueert het virtuele werkgeheugen onder lru als volgt:

Tijd	**Lees**	**Virt. Pagina**		
		0	1	2
0				
1	2	2		
2	3	2	3	
3	2	2	3	
4	1	2	3	1
5	5	2	5	1
6	2	2	5	1
7	4	2	5	4

8	5	2	5	4
9	3	3	5	4
10	2	3	5	2
11	5	3	5	2
12	2	3	5	2

Op tijdstip 5 treedt de eerste pagineringsfout op: we hebben pagina 5 nodig en het werkgeheugen bevat paginaâ€™s 2, 3 en 1 die respectievelijk nog gebruikt werden op tijdstippen 3, 2 en 4. Pagina 3 werd het langst niet gebruikt en zal dus vervangen worden door 5. De functie lru geeft als resultaat de lijst van pagineringsfouten. Voor ons voorbeeld is dat de lijst [..., (5,1,3,5),(7,2,1,4), (9,0,2,3),(10,2,4,2)] waarbij ... eventueel rapporteert over de eerste paginaâ€™s die in het lege werkgeheugen worden geladen en (5,1,3,5) de eerste echte pagineringsfout kenmerkt aan de hand van het tijdstip, de positie in het werkgeheugen, de oude en de nieuwe pagina. Definieer in Haskell de nodige types. Geef ook het type van al de functies die je schrijft.

Indien je slechts een deel van het probleem oplost, dan zorg je dat je commentaar aangeeft hoe jouw deel past in een volledige oplossing.

een oplossing

Categorieën: 3bi | Aoti | 3bw