Dt examens 2024

24 januari 2024

1 Prolog: Nonogram

`puzzle(Breedte, Hoogte, RijConstraints, KolomConstraints)`
Bijvoorbeeld: `puzzle(3, 3, [[1, 1], [1, 1], [1, 1]], [[3], [], [3]])`
stelt de puzzel voor:

$$\begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix}$$

1.1 Hulpfuncties

Gegeven een ingevulde puzzel, vind de n-de rij/kolom. Bijvoorbeeld:
`select_row(Puzzel, 1, Rij)` $\rightarrow$ `Rij = [1, 1]`
(Hetzelfde voor kolom)
Gegeven een ingevulde rij/kolom, krijg de constraints van die rij/kolom. Bijvoorbeeld:
`get_constraints([0, 1, 1, 1, 0, 0, 1, 0, 1, 0], C)` $\rightarrow$ `C = [3, 1, 1]`
Gegeven een breedte en hoogte, maak een leeg bord met niet-geïnstantieerde variabelen. Bijvoorbeeld:
`make_empty_board(2, 3, Bord)` $\rightarrow$ `Bord = [[_, _], [...], [...]]`
`solve(Puzzel, Oplossing)` (duidelijk)

1.2 Haskell: Fibonacci en Bomen

`fib0 :: Int → Int`
Implementeer Fibonacci.
`fib1 :: Int → [(Int, Int)]`
Implementeer Fibonacci, maar houd bij elke recursieve oproep en het resultaat daarvan bij. (Int, Int) hier vertegenwoordigt (n, fib0 n). Bijvoorbeeld:
`fib1 4 → [(4, 3), (3, 2), (2, 1), (1, 1), (0, 0), (1, 1), (2, 1), (1, 1), (0, 0)]`
omdat fib 4 recursief fib 3 oproept, wat op zijn beurt fib 2 oproept, enzovoort.

1.2.1 Tweede Vraag over Bomen

Maak een datatype `RoseTree` dat een waarde van het type `a` heeft en een (mogelijk lege) lijst van kinderen. Bijvoorbeeld:
`RoseTree 4 [RoseTree 2 [], RoseTree 1 [RoseTree 8 []], RoseTree 5 []`
Maak een mapfunctie die twee functies krijgt en `f1` toepast op alle elementen behalve het laatste, waar `f2` wordt gebruikt.

1.2.2 Fibonacci en Rose Bomen

`fib2 :: Int → RoseTree Int`
Vergelijkbaar met `fib1`, maar in plaats van een platte lijst maakt het een `RoseTree`.

1.3 Interessante Opmerking

Het Haskell-examen op Esystant was aanvankelijk iets compleet anders dan de Fibonacci-opdracht die we op papier hadden, maar ze hebben het snel opgelost. Het zou heel goed het examen voor morgen kunnen zijn. Ik heb er niet echt naar gekeken, maar vraag 1 ging over het maken van een datatype voor een binaire boom, dus het zal waarschijnlijk zoiets zijn als het mooi afdrukken van de boom en het berekenen van de diepte, wat aan bod kwam in een oefening op Esystant dit semester (Week 4 → Tree folds, niet erg moeilijk, maar het geeft je wat oefening met binaire bomen).

2 examen 16/01/2024

2.1 Prolog: Domino

geg:

- Afmetingen spelbord: `board(#rijen, #kolommen)`
- Cell: `cell(X, Y)` met X = cord-rij, Y = cord-kolom
- Stone: `stone(V1, V2)` met $V1/V2$ het aantal ogen op de domino steen van `cell_1/cell_2`
- Placement: `placement(stone(V1, V2), cell_1, cell_2)`
- Placements = [Placement]

2.1.1 find_value(Cell, Placements, Value)

```
1 Function find_value(Cell, Placements, Value):
2     for each Placement in Placements do
3         if Cell is part of Placement then
4             Value ← calculate value from Placement;
5             return Value;
6         end
7     end
8     Value ← default value;
9     return Value;
```

2.1.2 checkCell(Cell, Board, Placements)

```
1  Function checkCell(Cell, Board, Placements):
2  |  if Cell is not within the dimensions of Board then
3  |  |  return false;
4  |  end
5  |  if Cell is not part of placements then
6  |  |  return true;
7  |  end
8  |  Value ← get value of Cell from Board;
9  |  Neighbours ← get neighboring cells from Board;
10 |  for each Neighbour in Neighbours do
11 |  |  if Neighbour has a different value than Cell then
12 |  |  |  return false;
13 |  |  end
14 |  end
15 |  return true;
```

2.1.3 checkAllCells(Board, Placements)

```
1  Function checkAllCells(Board, Placements):
2  |  for each Cell in Placements do
3  |  |  if not checkCell(Cell, Board, Placements) then
4  |  |  |  return false;
5  |  |  end
6  |  end
7  |  return true;
```

2.1.4 solve functie

hier ben ik niet meer geraakt maar is gelijk aan de solve functies van oudere examenvragen

2.2 functie 5

hier ben ik niet meer geraakt

2.3 Haskell

2.3.1 vraag 1:

oefening gelijk aardig aan opgave 2 van spawnanalyser (individuele sessie).
opl:

```
1  groupSpawnsBy :: Eq k => (Spawn -> k) -> [Spawn] -> [(k, [Spawn])]
2  groupSpawnsBy f = foldr (\x -> insert (f x) x) []
3    where
4      insert k s [] = [(k, [s])]
5      insert k s ((k', ss) : kss)
6          | k == k' = (k, s : ss) : kss
7          | otherwise = (k', ss) : insert k s kss
```

2.4 vraag 2: vervolg op vraag 1

aanmaken van een datatype dat als input een key value list neemt gemaakt met opgave 1

2.5 vraag 3-4-5:

schrijf een algoritme greedy (3) en een efficienter algoritme (4): zie coin exchange, week 2 opgave 7 en vergelijk de snelheid (5) (? geen idee hou dit moest).

2.6 vraag 6:

IO met pretty print gelijkaardig aan die van personal manager:

```
"enter amount: (in cents)"
65
"enter amount received : (in cents)"
100
```

pretty print:

```
"50 cents amount: 1"
"10 cents amount: 1"
"5 cents amount: 1"
```