Haskell. Gekwoteerde zitting.

Patrick De Causmaecker

24/5/2023, 9:00 - 12:00

Toegelaten documentatie zijn hoogλe, Haskell wiki en de cursus. Zend op het einde je antwoorden naar `patrick.decausmaecker@kuleuven.be`.

1 De drempel (1 op 3)

Gegeven een stijgende rij gehele getallen, wat is de index van het eerste getal in de rij dat groter is dan een gegeven drempel?

1.1 erover (0.3)

De functie erover neemt een stijgende rij van gehele getallen en geeft de index terug van het eerste getal dat groter is dan de eveneens gegeven grens.

```
> erover [1..] 10000000000
^C^CInterrupted.
> erover [x*x*x|x <- [1..]] 10000000000
2154
```

1.2 erover_f (0.2)

De functie erover_f neemt een functie die een geheel getal afbeeldt op een element van een type dat vergelijking toelaat en geeft de index terug van het eerste getal dat groter is dan de eveneens gegeven grens.

```
> erover_f (\x -> (fromIntegral (x*x))) 0 (10000000000000/111)
300151
> erover_f (\x -> x*x*x*x*x*x) 0 1000000000
32
> erover_f (\x -> x*x) 0 1000000000
31623
> erover_f (\x -> x) 0 1000000000
^CInterrupted.
> erover_f (\x -> ['a'..'z']!!x) 0 'e'
5
```

1.3 fiberover (0.3)

De functie fiberover doet hetzelfde als de functie erover_f, maar ze gebruikt een fibonacci rijen om de grens sneller te overschrijden. (Laat het argument toenemen als een rij van fibonacci tot de grens wordt overschreden. Start dan opnieuw van de voorlaatste index en laat opnieuw het argument toenemen als een rij van fibonacci. Stop als het verschil in index van de waarde juist onder de grens en die juist boven de grens gelijk is aan 1.)

```
> fiberover (\x -> x*x*x*x*x*x) 0 1000000000
32
> fiberover (\x -> x*x) 0 1000000000
31623
```

```
> fiberover (\x -> x) 0 1000000000
1000000001
> fiberover (\x -> (fromIntegral (x*x))) 0 (10000000000000/111)
300151
```

1.4 En rapporteer (0.2)

De functie fiberover_l doet hetzelfde als de functie fiberover, maar ze geeft ook een lijst van gevalueerde argumenten terug.

```
> fiberover_l (\x -> x) 0 10
(11,[1,2,4,7,12,8,8,9,11,10,10,11,11])
> fiberover_l (\x -> x*x*x) 0 1000000000
(1001,[1,2,4,7,12,20,33,54,88,143,232,376,609,986,1596,987,987,988,990,993,998,1006,999,999,
1000,1002,1001])
> fiberover_l (\x -> (fromIntegral (x*x*x))) 0 (10000000000000/111)
(4483,[1,2,4,7,12,20,33,54,88,143,232,376,609,986,1596,2583,4180,6764,4181,4181,4182,4184,4187,4192,
4200,4213,4234,4268,4323,4412,4556,4413,4413,4414,4416,4419,4424,4432,4445,4466,4500,4467,4467,
4468,4470,4473,4478,4486,4479,4479,4480,4482,4485,4483])
```

2 Iets met priemfactoren (1 op 3)

2.1 Geen delers (0.3)

Schrijf een functie nondiv die True terug geeft asa het eerste argument geen deler heeft in de lijst die als tweede argument wordt meegegeven.

```
> nondiv 100 [10,20,30,40,50]
False
> nondiv 100 [11,20,30,40,50]
False
> nondiv 100 [11,22,30,40,50]
False
> nondiv 100 [11,22,30,40,55]
True
```

2.2 Naief, maar goed (0.1)

Schrijf een recht-toe recht-aan functie die True teruggeeft a.s.a. het argument een priemgetal is.

```
> priem_r_t_r_a 10
False
> priem_r_t_r_a 11
True
> priem_r_t_r_a 541
True
> priem_r_t_r_a 1
False
```

2.3 Klassiek (0.3)

Schrijf een functie priem_f die een lijst van een aantal priemgetallen teruggeeft. De functie gebruikt de zeef van Eratosthenes die een getal aanvaardt als geen van de priemgetallen waarvan het kwadraat niet groter is dan dat getal dit getal delen. De vorm waarin je deze functie schrijft is vrij.

```
> priem_f 9
[2,3,5,7,11,13,17,19,23]
> priem_f 19
[2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67]
> priem_f 100
[2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,97,101,103,107,109,113,127,131,
137,139,149,151,157,163,167,173,179,181,191,193,197,199,211,223,227,229,233,239,241,251,257,263,269,
271,277,281,283,293,307,311,313,317,331,337,347,349,353,359,367,373,379,383,389,397,401,409,419,421,
431,433,439,443,449,457,461,463,467,479,487,491,499,503,509,521,523,541]
> priem_f 1
[2]
```

2.4 Als een recursief gedefinieerde lijst (0.3)

Definieer priem als een oneindige lijst van alle priemgetallen in stijgende volgorde. Gebruik een recursieve constructie die de zeef van Eratosthenes implementeert met behulp van takeWhile. [1]

```
> take 9 priem
[2,3,5,7,11,13,17,19,23]
> take 19 priem
[2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67]
> take 100 priem
[2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,97,101,103,107,109,113,127,131,
137,139,149,151,157,163,167,173,179,181,191,193,197,199,211,223,227,229,233,239,241,251,257,263,269,
271,277,281,283,293,307,311,313,317,331,337,347,349,353,359,367,373,379,383,389,397,401,409,419,421,
431,433,439,443,449,457,461,463,467,479,487,491,499,503,509,521,523,541]
> take 1 priem
[2]
```

3 Een beetje IO (1 op 3)

Schrijf een programma dat toelaat een 'tekening' te maken. Het commando teken tekent een bord met een hoogte en een breedte en laat vervolgens toe om puntjes te zetten. (0.7) Je kan het commando fout-tolerant maken op twee manieren:

- controleer op de ingegeven cijfers en vraag opnieuw indien de gebruiker een teken ingeeft dat niet als een getal kan worden gelezen (0.2).
- controleer op de input en vraag opnieuw indien de grenzen niet werden gerespecteerd of iets anders dan 'ja' of 'nee' werd ingegeven (0.1).

Hieronder een voorbeeld met de meest eenvoudige implementatie.

[1] De definitie van priem kan op 1 lijn van minder dan 80 tekens, spaties inbegrepen

```
> teken 3 3
"==="
"   "
"   "
"   "
"==="
"Tekenen (j/n)?"
0
"Bye"
> teken 3 3
"==="
"   "
"   "
"   "
"==="
"Tekenen (j/n)?"
j
"Hoogte?"
0
"Breedte?"
1
"==="
" . "
"   "
"   "
"==="
"Tekenen (j/n)?"
j
"Hoogte?"
1
"Breedte?"
1
"==="
" . "
" . "
"   "
"==="
"Tekenen (j/n)?"
j
"Hoogte?"
2
"Breedte?"
1
"==="
" . "
" . "
" . "
"==="
"Tekenen (j/n)?"
j
"Hoogte?"
1
"Breedte?"
0
"==="
" . "
".. "
" . "
"==="
"Tekenen (j/n)?"
j
"Hoogte?"
1
"Breedte?"
2
"==="
" . "
"..."
" . "
"==="
"Tekenen (j/n)?"
n
"Bye"
```