Haskell. Gekwoteerde zitting.

Patrick De Causmaecker

20/5/2019, 14:00 - 17:00

Zend voor je begint, de url van de websites die je denkt te gebruiken naar `patrick.decausmaecker@kuleuven.be` Zend op het einde je antwoorden naar hetzelfde adres.

1 Benaderingsformules revisited $(\frac{0.7}{3})$

De Taylor benadering van een functie $f(x)$ van één reële veranderlijke x in het punt 0 is gegeven door

$$f(x) = f(0) + \frac{f'(0)}{1!}(x) + \frac{f''(0)}{2!}(x)^2 + \frac{f^{(3)}(0)}{3!}(x)^3 + \dots$$

waarbij $f^{(n)}$ de n^{de} afgeleide voorstelt. We gaan uit van

1. Een functie $f(n)$ die de n^{de} afgeleide van een functie f voorstelt in een punt 0
2. Een functie $\theta(n)$ die een bovengrens voor de fout na n termen in de Taylor benadering van de functie f rond 0 in een punt van het interval $(-1, 1)$ geeft.

Schrijf een Haskell functie die voor de functie f in een punt x in $(-1, 1)$ en een fout ϵ een benadering van f genereert waarbij de fout niet groter is dan ϵ. Demonstreer deze functie door een benadering van het getal e te berekenen tot op 5 decimalen nauwkeurig. ($f(x) = e^x \Rightarrow f^{(n)}(0) = 1$, de fout na n termen in de Taylor reeks is kleiner dan $\frac{3}{(n+1)!}$).

2 Priemgetallen revisited $(\frac{0.8}{3})$

Toen Erathostenes rond 200 v.Chr. zijn zeef bedacht, was er nog geen geheugenprobleem. Hij kon nadenken over willekeurig lange lijsten ($n \lessapprox 1000$). Toen de computer werd uitgevonden en men rond 1960 n.Chr de zeef wou gebruiken om grote priemgetallen te berekenen had men wel geheugenproblemen. Een manier om dit aan te pakken was de lijst van n ($n \lessapprox 10^7$) getallen op te splitsen in $\sqrt{n}$ keer $\sqrt{n}$ getallen. Het 'nieuwe' algoritme verwijdert de veelvouden van priemgetallen kleiner dan of gelijk aan $\sqrt{n}$ uit achtereenvolgens $]\sqrt{n}, 2 \times \sqrt{n}]$, $]2 \times \sqrt{n}, 3 \times \sqrt{n}]$, ... Implementeer dit algoritme als *batchedSieve n* in Haskell.

3 Paarden bewegen $(\frac{0.8}{3})$

Deze oefening bekijkt de mogelijkheden van een paard in een schaakspel op een bord van R rijen en K kolommen. Een veld wordt voorgesteld door een koppel (r, k), waarbij $r, 1 \leqslant r \leqslant R$, het rijnummer en $k, 1 \leqslant k \leqslant K$, het kolomnummer zijn. Een paard is een stuk in het schaakspel. Als het zich bevindt op een veld (r, k), dan kan het in één zet de velden $(r+1, k+2), (r+1, k-2), (r-1, k+2), (r-1, k-2), (r+2, k+1), (r+2, k-1), (r-2, k+1)$ en $(r-2, k-1)$ bereiken, voor zover deze zich binnen de grenzen van het bord bevinden. Een paard op veld $(4, 3)$ kan dus 8 velden bereiken als $R \geqslant 6$ en $K \geqslant 5$. Een paard op veld $(1, 1)$ kan de velden $(2, 3)$ en $(3, 2)$ bereiken als $K \geqslant 3$ en $R \geqslant 3$.

Gevraagd zijn drie functies met signatuur gegeven door

$$data\ PaardeBoom = PaardeBoom\ (Int, Int)\ [PaardeBoom]$$
$$paardePaden :: (Int, Int) -> Int -> Int -> PaardeBoom$$
$$paardeVelden :: PaardeBoom -> Int -> [(Int, Int)]$$
$$paardePad :: PaardeBoom -> (Int, Int) -> Maybe\ [(Int, Int)]$$

De functie *paardePaden* (r, k) *rijen kolommen* stelt in de vorm van een oneindige boom de velden voor die een paard kan bereiken op een schaakbord met R rijen en K kolommen, vanaf het veld (r, k) (de wortel van de boom). (Een knoop op diepte d van die boom is een veld dat na d zetten van het paard kan bereikt worden.).

De functie *paardeVelden paden d* geeft de verzameling van velden terug die in d of minder zetten kunnen bereikt worden vanuit de wortel van *paden*. *paardeVelden p* 0 is het singleton met als enig element de wortel van de boom p.

De functie *paardePad paden* (r, k) neemt een parameter paden, een oneindige boom gegenereerd door *paardePaden*, en geeft - indien mogelijk (*Maybe*) - een pad terug dat het paard naar het veld (r, k) brengt.

4 Toon dat bord $(\frac{0.7}{3})$

De bedoeling is de bewegingen van vorige oefening weer te geven als opeenvolgende posities op een bord. Maak daartoe een type aan voor de output van *paardePad* samen met de dimensies van het bord. Zorg ervoor dat de volgende output kan geproduceerd worden (voorbeeld voor een bord van 4x4 met drie zetten beginnende van positie $(1, 1)$):

```
 === === === ===
| 1 |   |   |   |
 === === === ===
|   |   | 4 |   |
 === === === ===
|   | 2 |   |   |
 === === === ===
|   |   |   | 3 |
 === === === ===
```

Als tweemaal dezelfde positie wordt bereikt, wordt enkel het laatste nummer getoond:

```
 === === === ===
| 1 |   |   |   |
 === === === ===
|   |   |   |   |
 === === === ===
|   | 4 |   |   |
 === === === ===
|   |   |   | 3 |
 === === === ===
```

Illustreer je oplossing in een functie *kijkEnToon* die de gebruiker toelaat een bord te specifiëren, de beginpositie en de zetten één voor één in te geven, terwijl telkens opnieuw het resultaat wordt getoond (leeg bord, bord met positie 1, bord met positie 1 en 2, ...).

Extraatje: *Zorg ervoor dat, voor het geval de gebruiker zich bij een zet vergist, men door* -1 *in te geven de laatst ingegeven zet kan verwijderen.*