Haskell. Gekwoteerde zitting.

Patrick De Causmaecker

21/5/2025, 9:30 - 12:30

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

1 Catalan getallen (1 op 3)

Catalan getallen komen in veel toepassingen voor. Het n-de getal telt bijvoorbeeld het aantal manieren om een convexe $(n+2)$-hoek te verdelen in niet overlappende driehoeken. Ze zijn genoemd naar de Belgische 19-de eeuwse wiskundige Eugène Catalan.

1.1 Context en formules

Het n-de Catalan getal kan voor $n \geqslant 0$ berekend worden als

$$C(n) = \frac{(2n)!}{(n+1)!n!} \tag{1}$$

De getallen kunnen ook recursief berekend worden als

$$C(0) = 1, C(n+1) = \frac{2(2n+1)}{n+2}C(n) \tag{2}$$

Er bestaat een som-formule voor de getallen van Catalan

$$C(0) = 1, C(n+1) = \sum_{i=0}^{n} C(i)C(n-i) \tag{3}$$

En er is een asymptotische benadering gegeven door

$$C(n) \sim \frac{4^n}{n^{\frac{3}{2}}\sqrt{\pi}} \tag{4}$$

1.2 Opgaven

1. Schrijf een Haskell functie die de formule 1 implementeert. (0.1)
2. Schrijf een Haskell functie die de recursieve formule 2 implementeert.(0.1)
3. Schrijf een Haskell functie die de som formule 3 implementeert.(0.1)
4. Schrijf een functie die de asymptotische benadering implementeert (formule 4). (0.1)
5. Schrijf een tabulatiefunctie die de waarden van een functie van natuurlijke getallen in een lijst zet. Gebruik deze om voor elk van de vorige formules oneindige lijsten te definiëren. (0.2)
6. Schrijf een functie die uit een lijst van Catalan getallen deze selecteert die oneven zijn, samen met hun volgnummer. Valt je iets op? (0.2)

7. Gebruik de lijsten voor de functie *testCatalan start einde* die voor $start \leqslant n \leqslant einde$ tuppels van de vorm $(check, error)$ genereert waarbij $check = True$ als formules 1 en 2 dezelfde waarde genereren (dit moet altijd $True$ zijn), en *error* de absolute waarde van het verschil met de asymptotische benadering gedeeld door de waarde (relatieve fout) voorstelt.(0.2) [1]

1.3 Voorbeelden

```
ghci> catalan 7
429
ghci> catalanRec 7
429
ghci> catalanSum 7
429
ghci> catalanAsy 7
499.111921073996
ghci> take 8 lcatalan
[1,1,2,5,14,42,132,429]
ghci> take 8 lcatalanRec
[1,1,2,5,14,42,132,429]
ghci> take 8 lcatalanSum
[1,1,2,5,14,42,132,429]
ghci> testCatalan 4 7
[(True,0.28957619096630005),(True,0.23032747740411624),(True,0.19119868369915013)]
ghci> odds (take 40 lcatalan)
[(0,1),(1,1),(3,5),(7,429),(15,9694845),(31,14544636039226909)]
```

2 Dyck (1 op 3)

Een Dyck-woord is een opeenvolging van haakjes zodanig dat er tot aan elke positie in het woord steeds meer open haakjes staan dan dichte en dat er evenveel open haakjes als dichte voorkomen, bijvoorbeeld "()", "((()))", "(()())(())". De lege string is ook een Dyck-woord. Als je dat gemakkelijker leesbaar vindt mag je een open haakje bijvoorbeeld vervangen door een letter X en een dicht haakje door een letter Y. De voorbeelden hiervoor worden dan "XY","XXXXYYYY","XXYXYYXXYY".

2.1 Alles van Dyck

De oneindige rij allVanDyck bevat lijsten van strings van toenemende lengte: eerst deze van lengte 0, dan deze van lengte 2, enz. Dus bijvoorbeeld:

$$allVanDyck = [[" "], ["()"], ["(())","()()"], ["((()))","(()())","(())()","()(())","()()()"], ..] \tag{5}$$

1. Schrijf een functie *vanDyck n* die de rij van Dyck-woorden van lengte $2n$ genereert.(0.6)[2]
2. Schrijf de Haskell functie $allVanDyck$ die de rij 5 genereert.(0.2)
3. Schrijf een functie $aantalVanDyck$ *n* die laat weten hoeveel Dyck-woorden van lengte $2n$ er bestaan. Valt je iets op? (0.2) [3]

2.2 Voorbeelden

```
ghci> vanDyck 3
["((()))","(()())","(())()","()(())","()()()"]
```

[1] Moeilijkste deel, stel eventueel uit voor het geval de tijd te beperkt is

[2] Ook ietsje moeilijker, zie vraag 3

[3] Zie de voorbeelden. Het aantal Dyck-woorden van lengte $2n$ is gelijk aan het n-de getal van Catalan. De som-formule 3 kan je misschien inspireren om de Dyck-woorden van een bepaalde lengte te genereren.

```
ghci> take 4 allVanDyck
[[""],["()"],["(())","()()"],["((()))","(()())","(())()","()(())","()()()"]]
ghci> aantalVanDyck 4
14
ghci> aantalVanDyck 7
429
```

3 Een beetje IO (1 op 3)

Een voorbeeld van een zogenaamde *Cantor-verzameling* ontstaat door een interval in drie gelijke deelintervallen te verdelen en kruisjes te zetten aan de grenzen. In het middelste interval worden verder geen punten gekozen, in de twee buitenste herhalen we de procedure recursief tot op willekeurige diepte. Als we de diepte aangeven, dan kunnen we de verzameling als een string voorstellen. Diepte 3 correspondeert dan bijvoorbeeld met de string "x.x...x.x.....x.x...x.x", waarbij het middelste stuk 9 keer '.' bevat, en de twee andere corresponderen met de string "x.x...x.x"voor diepte 2. Bij diepte 1 vinden we "x.x", en bij diepte 0 "x". We gebruiken deze set in de I/O oefening waarbij een diepte wordt gevraagd en de corresponderende string wordt afgedrukt.

3.1 Opgave

1. Schrijf de functie *cantor* :: $Int \rightarrow String$ die de set met een gevraagde diepte voorstelt.(0.3) [4]
2. Schrijf een functie *speelCantor* :: IO () die de gebruiker vraagt om een diepte op te geven en dan de corresponderende set afdrukt. Vraag vervolgens of de gebruiker dat nog eens wil doen.(0.5)
3. Beveilig de vorige functie bij de ingave van de diepte waarbij een foute invoer opnieuw kan worden ingegeven, tot een gespecificeerd aantal keer toe. (*speelCantor* :: $Int \rightarrow IO$ ()) (0.2)

3.2 Voorbeeld

```
ghci> cantor 0
"x"
ghci> cantor 3
"x.x...x.x.....x.x...x.x"
ghci> cantor 4
"x.x...x.x.....x.x...x.x.........................x.x...x.x.....x.x...x.x"
ghci> speelCantor 2
Diepte:
4
x.x...x.x.....x.x...x.x.........................x.x...x.x.....x.x...x.x
Nog eens? (j/n)
j
Diepte:
25^?
Diepte: ??
2
x.x...x.x
Nog eens? (j/n)
j
Diepte:
vijf
Diepte: ??
vijf!!!
Teveel fouten!
Gestopt.
```

[4] Houd dit deel tot het laatste, en los het enkel op als er nog tijd rest. Gebruik voorlopig een dummy implementatie.