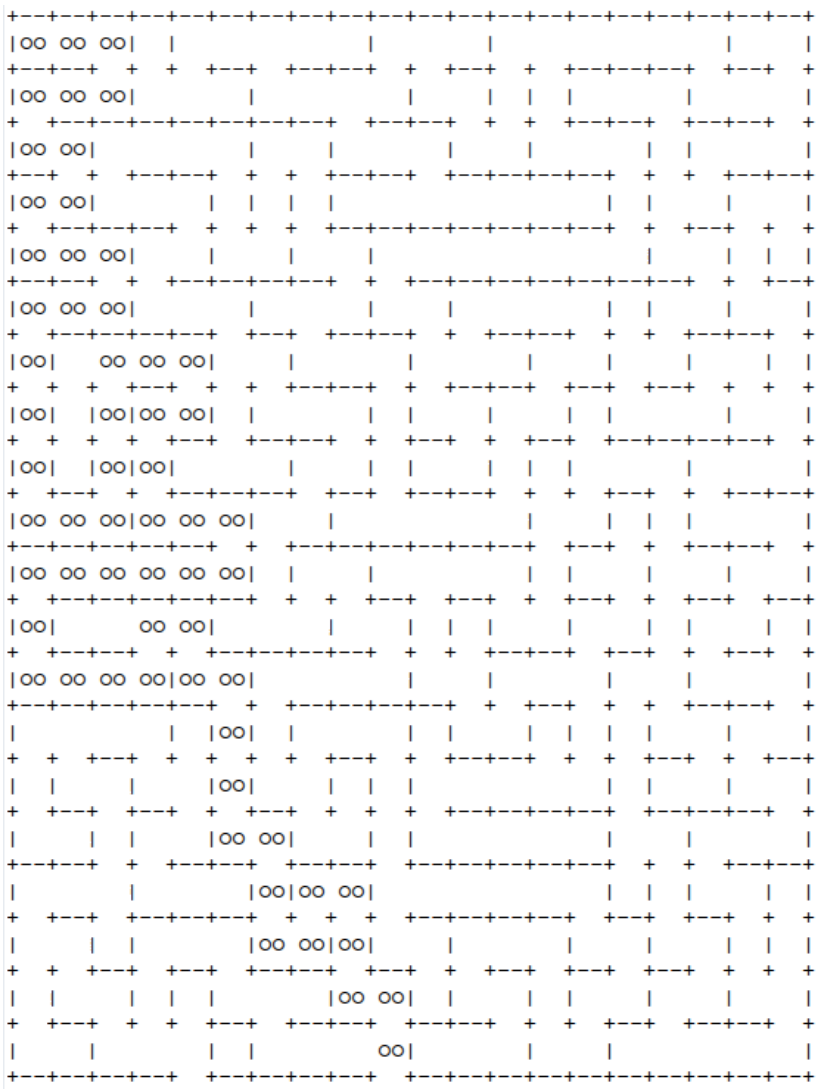
Universiteit van Amsterdam

Programmeertalen

Go

Jouke Witteveen, Robin de Vries

Mazesolver

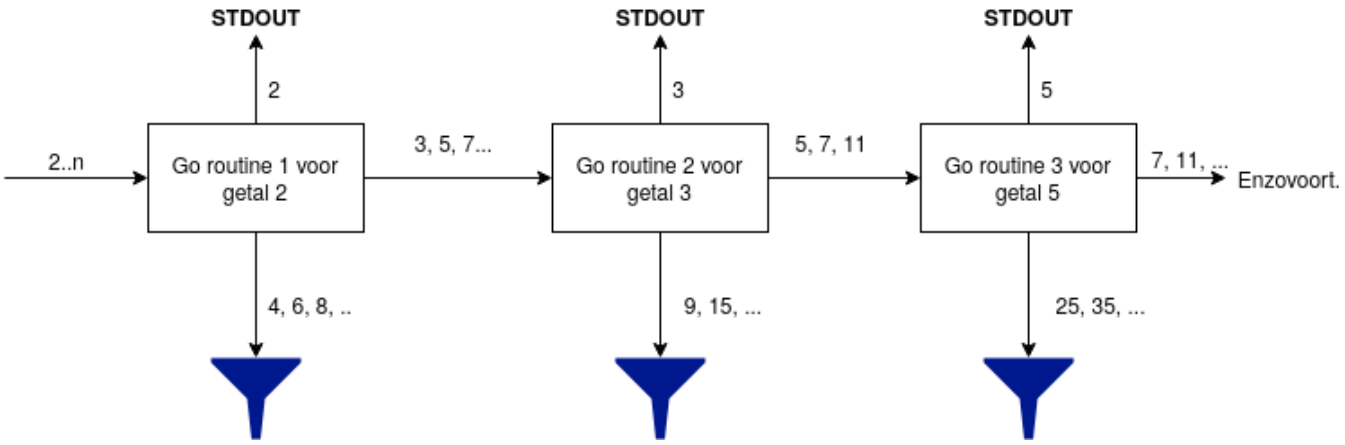


Deadline: zondag 14 maart 2021 om 23:59

1 Puzzels

Als puzzel moeten jullie een Zeef van Eratosthenes implementeren met behulp van channels en go routines. Elke go routine filtert 1 veelvoud eruit en geeft de rest door via een channel aan de volgende goroutine. Je krijgt dus een ketting van goroutines. Als de laatste goroutine een getal tegen komt wat hij niet kan filteren heb je een priem getal gevonden. De laatste goroutine spawned dan een nieuwe goroutine voor dat priem getal. De nieuwe goroutine print het priem getal naar `STDOUT` en zal veelvouden van het priem getal filteren. Uiteindelijk krijg je een goroutine voor elk priem getal.

Als argument op de commandline moet je een getal opgeven tot hoe ver hij moet genereren. Als hij dit getal bereikt heeft moeten alle goroutines netjes afsluiten, je mag dus geen goroutines lekken. Figuur 1 geeft een klein overzicht voor $n \geq 35$.



Figuur 1: Klein overzicht van hoe de goroutines gelinkt moeten worden en hoe getallen afgehandeld moeten worden bij $n \geq 35$. De horizontale pijlen zijn channels en de rechthoeken zijn Go routines. het filteren van getallen gebeurt dus concurrent.

De output wordt getest op Codegrade en die moet precies gevolgd worden. Dus, voor `sieve 50` wordt het volgende verwacht:

Shell snippet

```
 ./sieve 50
2
3
5
7
11
13
17
19
23
29
31
37
41
43
47
```

2 Maze

Omschrijving

Binnen de context van deze opgave is een doolhof (of *maze*) een rechthoekig veld van vierkante hokjes. Deze hokjes hebben vier zijden welke of *open* zijn, of geblokkeerd zijn door een *muur*. Een oplossing voor het doolhof is een reeks van verbonden hokjes zodanig dat er een *pad* loopt van de ingang van het doolhof naar de uitgang, zonder dat het pad door een muur belemmerd wordt. De ingang bevindt zich altijd op de noordwester rand van het doolhof, en de uitgang op de zuidooster rand van het doolhof.

Tijdens deze opgave ga je een gedistribueerde maze solver schrijven: `maze.go`. Dit programma moet als enige argument de bestandsnaam van een doolhof meekrijgen. Een template voor deze opgave om mee te beginnen is meegeleverd welke een doolhofbestand kan inlezen.

Bestandsformaat (Ontwikkelend)

De `readMaze` functie leest een doolhof naar een twee-dimensionale array van bytes. Het bestandsformaat dat is gebruikt bestaat uit een reeks van 8-bits getallen, waarvan we 2 bits gebruiken: Dit geeft ons de mogelijke waarden van 0 t/m 3. De eerste bit geeft aan of de zuidkant van het hokje een muur bevat en de tweede bit of de oostkant van het hokje een muur bevat. Omdat aangrenzende hokjes binnen het doolhof muren delen, kun je met alleen het vastleggen van deze twee muren per hokje het hele doolhof definiëren, behalve de buitenmuren. Voor de volledige noord- en westzijden van het doolhof leggen we vast dat deze met een muur omsloten zijn.

Een Python script is meegeleverd dat de meegeleverde bestanden kan visualiseren. Hiermee kan je ook testen of je formatting correct is. De tests lezen jullie output in en kijken dan of de maze correct is, hiervoor moet het formaat gevolgd worden, dus er mogen geen spaties tussen de getallen staan. Test ook weer vaak op Codegrade om te kijken of alles klopt met je output.

Bedenk je dat error handling in Go door de programmeur moet gebeuren, het is dus ook belangrijk dat je dit doet aangezien er anders vreemde dingen kunnen gebeuren.

Het algoritme (Competent)

De zoektocht naar een oplossing moet op een gedistribueerde manier worden geïmplementeerd in je programma. Ofwel, je moet tegelijkertijd meerdere paden onderzoeken met behulp van goroutines. Zodra een oplossing voor het doolhof is gevonden (in de vorm van een slice van `Positions`) moet de zoektocht worden gestaakt en moet het programma de oplossing teruggeven. Het programma mag niet termineren terwijl er nog goroutines lopen, zorg er dus voor dat je deze allemaal netjes afsluit. Mits er nog goroutines lopen krijg je 1 punt aftrek.

Je programma moet `channels` gebruiken voor de zoektocht. De werking van je programma mag niet afhankelijk zijn van de grootte van de buffer of überhaupt de aanwezigheid van je buffer in de channels. Daarnaast mag je programma ieder hokje in het doolhof slechts één keer bezoeken. En als laatste mag je programma niet schrijven naar gedeeld geheugen tussen de goroutines. Bij `race conditions` in je code krijg je 1 punt aftrek.

De uitvoer van het programma moet weer een vergelijkbare reeks getallen zijn, maar als een hokje ligt op het gevonden pad moet ook de 3e bit worden aangezet. Als het doolhof geen oplossing heeft moet het programma, zonder te crashen of vreemde uitvoer te geven, het oorspronkelijke doolhof teruggeven.

Mits je ook onoplosbare doolhoven kan verwerken kom je in het niveau `gevorderd`.

Korste Pad (expert)

Het is de vraag of het gevonden pad ook het kortste pad is om uit het doolhof te komen. Schrijf een kleine uitleg waarom het gevonden pad wel of niet het kortste pad is, dit moet geschreven worden in de comments van je programma. Als het gevonden pad volgens jou niet het kortste pad is, moet je ook beschrijven hoe je het programma zou moeten aanpassen om te zorgen dat wel het kortste pad wordt gevonden. Deze aanpassing mag niet zorgen dat je programma niet meer concurrent is. Dit niveau kan alleen worden afgemaakt als je het niveau `gevorderd` hebt gehaald.

3 Vragen en Antwoorden

Hoe gebruik ik het print programma mazeprint.py?

Het print programma leest een doolhof representatie van de invoer en genereert een meer aantrekkelijke weergave. Om de bovenste rand van het doolhof te tekenen moet het programma weten hoe breed het doolhof is. Daarom verwacht het programma de breedte van het doolhof als argument.

> Om een doolhof uit een tekstbestand weer te geven, gebruik dan, bijvoorbeeld
> `$ ./mazeprint.py 40 < maze1.txt`
> waarbij `40` de breedte van het doolhof is en `maze1.txt` het tekstbestand waarin het doolhof staat.

Het print programma werkt ook als uiteinde van een "pipe", maar controleert de invoer die het krijgt niet. Foutmeldingen uit je programma kunnen dus het print programma verstoren. Zoals eerder beschreven is error handling in go de taak van de programmeur, het is dus de bedoeling dat je deze errors netjes oplost.

> Om de uitvoer van je uiteindelijke implementatie van de opdracht weer te geven kun je het volgende gebruiken
> `$ go run maze.go maze1.txt | ./mazeprint.py 40`

Waarom en hoe werkt de detectie van muren?

De bytes die worden ingelezen in `readMaze` zijn de Ascii bytes van de ingelezen karakters. In de Ascii tabel zijn de cijfers "0", "1", "2", … uitgelijnd op een voldoende hoge macht van twee. Meer specifiek is de binaire representatie van het karakter "0" in de Ascii tabel 00110000. Merk op dat de laatste vier bits allemaal 0 zijn. Hierdoor kunnen de cijfer karakters gebruikt worden om de laatste vier bits in te stellen conform de binaire representatie van het betreffende cijfer.

De constructie `(1 << i)` die in de code gebruikt wordt resulteert in een binaire één gevolgd door `i` nullen. Het is een 1 die `i` posities naar links is geduwd en representeert daardoor de `i + 1e` "flag". De `&` operator berekent de "bitwise and" van zijn operanden, waardoor `someByte & someFlag` precies 0 is wanneer `someFlag` niet aanwezig is in `someByte`. De `|` operator, onderdeel van de `|=` operator, berekent de "bitwise or" van zijn operanden, waardoor `someByte | someFlag` verkregen wordt door `someFlag` op 1 te zetten in `someByte`.

Wat moet ik met al die informatie over zoekbomen?

Iedere informaticus hoort te weten wat een zoekboom is, hoe ze doorlopen kunnen worden, en wat het effect van het snoeien ervan is.