Algoritmiek: Tijdscomplexiteit van algoritmen

Opdracht 1: som-algoritme

© Toetsvraag Fundament Informatica

Om de som $n+(n+1)+(n+2)+\ldots 2n$ te berekenen, kun je in Python het volgende programma schrijven:

```
1  som = 0
2  for i in range(n):
3      som = som + n + i
```

Bepaal de orde van dit algoritme. Leg uit hoe je aan je antwoord bent gekomen.

Opdracht 2: frequentie van maximum

© Toetsvraag Fundament Informatica, minimaal aangepast voor de workshop

We willen tellen hoe vaak het grootste getal in een lijst voorkomt. Stel dat de lijst bijvoorbeeld bestaat uit de volgende getallen:

`5 62 62 11 73 29 73 4 11 0 73 55`

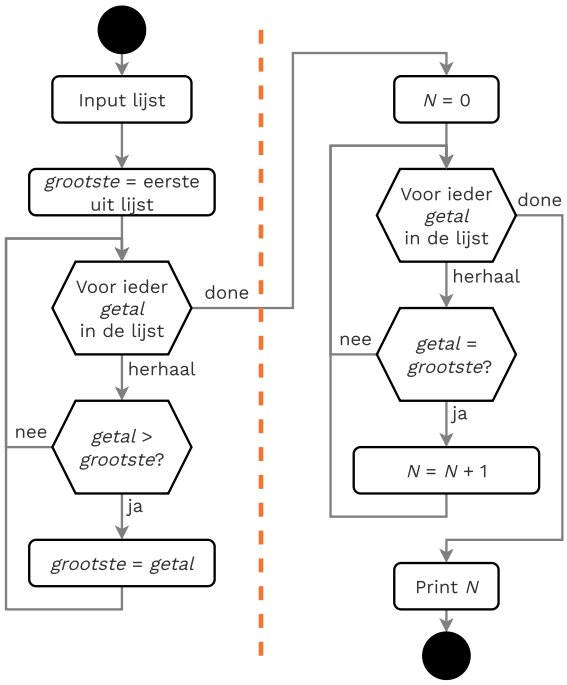
Dan moet het antwoord `3` zijn, omdat `73` het grootste getal is en dit getal drie keer in de lijst voorkomt.

Een algoritme om dit te doen zie je in fig. 1.

[4p, T2] Beredeneer wat de orde van dit algoritme is. Druk je antwoord uit in de grootte van de lijst, n. Leg uit hoe je aan je antwoord bent gekomen.

Tip 1. Het algoritme bestaat uit twee losse algoritmen. In het stroomdiagram zie je dit aan de twee helften. Bedenk eerst wat beide deelalgoritmen doen.

Tip 2. Om de orde van het hele algoritme te vinden, moet je de ordes van de twee losse deelalgoritmen bij elkaar optellen.



Figuur 1: Algoritme voor opdracht 2

Opdracht 3: overeenkomsten vinden

Stel dat we twee lijsten hebben. De ene lijst heeft grootte n en de andere lijst heeft grootte m. n en m kunnen dus verschillend zijn.

Een algoritme heeft als taak om de overeenkomsten tussen de twee lijsten te vinden. Oftewel: welke elementen staan in beide lijsten?

Een manier om dit te doen is door een dubbele herhaling te gebruiken:

```
1 for element1 in lijst1:
2     for element2 in lijst2:
3         if element1 == element2:
4             print(element1)
```

a. **[3p, T1]** Beredeneer wat de orde van dit algoritme is. Druk je antwoord uit in de grootte van de lijsten, n en m. Leg uit hoe je aan je antwoord bent gekomen.

Een andere manier om dit te doen is door beide lijsten eerst te sorteren met het **quicksort**-algoritme. Quicksort heeft de orde $\mathcal{O}(n \lg(n))$. Vervolgens heb je geen dubbele herhaling meer nodig, maar kun je van links naar rechts tegelijkertijd door beide lijsten gaan.

Stel bijvoorbeeld dat je de *gesorteerde* lijsten 1 2 4 5 6 9 en 2 3 5 hebt. Dan zien de iteraties er zo uit:

Iteratie 1: **1** 2 4 5 6 9 **2** 3 5	Iteratie 2: 1 **2** 4 5 6 9 **2** 3 5 **Print** 2	Iteratie 3: 1 2 **4** 5 6 9 2 **3** 5
Iteratie 4: 1 2 **4** 5 6 9 2 3 **5**	Iteratie 5: 1 2 4 **5** 6 9 2 3 **5** **Print** 5	Iteratie 6: 1 2 4 5 **6** 9 2 3 5 **Stop**, want lijst 2 is leeg.

b. **[4p, I]** Beredeneer wat de orde van dit algoritme is. Ga uit van het **worstcasescenario**. Neem in je antwoord mee dat dit algoritme ook **quicksort** twee keer moet uitvoeren. Druk je antwoord uit in de grootte van de lijsten, n en m. Leg uit hoe je aan je antwoord bent gekomen.

Opdracht 4: decimaal naar binair

© Toetsvraag Fundament Informatica, minimaal aangepast voor de workshop

Om een decimaal getal naar een binair getal om te rekenen, moet je kijken welke machten van 2 in het decimale getal passen. Stel dat we het decimale getal 50 willen omrekenen naar binair. De stappen zijn dan:

1. Past 32 in 50? Ja, noteer 1. Rest is $50 - 32 = 18$.
2. Past 16 in 18? Ja, noteer 1. Rest is $18 - 16 = 2$.
3. Past 8 in 2? Nee, noteer 0.
4. Past 4 in 2? Nee, noteer 0.
5. Past 2 in 2? Ja, noteer 1. Rest is $2 - 2 = 0$.
6. Past 1 in 0? Nee, noteer 0.

Het resultaat is dan dat $50_{10} = 110010_2$.

Je gaat nu de orde bepalen van dit algoritme. We noemen N het decimale getal dat we willen omzetten naar een binair getal. Dus in het bovenstaande voorbeeld is $N = 50$.

[3p, T2] Beredeneer wat de orde van dit algoritme is. Druk je antwoord uit in N. Leg uit hoe je aan je antwoord bent gekomen.

Tip. Net zoals bij de zojuist uitgelegde analyse van binair zoeken, zie je ook in dit algoritme dat het probleem iedere stap gehalveerd wordt.

Opdracht 5: decimaal naar hexadecimaal

Om een decimaal getal naar een hexadecimaal getal om te rekenen, moet je kijken welke machten van 16 in het decimale getal passen. Stel dat we het decimale getal 500 willen omrekenen naar binair. De stappen zijn dan:

1. Hoe vaak past $16^2 = 256$ in 500?
   $\lfloor \frac{500}{256} \rfloor = 1$, dus 1 keer. Noteer 1.
   Rest is: 500 mod $256 = 244$.
2. Hoe vaak past $16^1 = 16$ in 244?
   $\lfloor \frac{244}{16} \rfloor = 15$, dus 15 keer. Noteer F.
   Rest is: 244 mod $16 = 15$
3. Hoe vaak past $16^0 = 1$ in 4?
   $\lfloor \frac{4}{1} \rfloor = 4$, dus 4 keer. Noteer 4.
   Klaar.

De notatie $\lfloor \frac{X}{Y} \rfloor$ geeft **integer division** aan. Het equivalent in Python-code is `X // Y`.

Het resultaat is dan $500_{10} = 1F4_{16}$.

We willen nu de orde bepalen van dit algoritme. We noemen N het decimale getal wat we willen omzetten naar een binair getal. Dus in het bovenstaande voorbeeld is $N = 500$.

[4p, T2] Beredeneer wat de orde van dit algoritme is. Druk je antwoord uit in N. Leg uit hoe je aan je antwoord bent gekomen.

Opdracht 1: som-algoritme — uitwerking

Antwoord

We herhalen één keer over een lijst met grootte n. Dat zijn n stappen.

In iedere stap doen we $\mathcal{O}(1)$ werk. Immers, de n die in de berekening staat is een constante.

Misconceptie

Alhoewel hier een n staat, hoef je in deze berekening niet voor ieder element van de lijst iets te doen. Daarom is het $\mathcal{O}(1)$ werk.

In totaal is de orde dan $n \cdot \mathcal{O}(1) = \mathcal{O}(n)$.

Punten

0.5p Inzicht dat we n stappen moeten zetten.

1p Inzicht dat we $\mathcal{O}(1)$ werk per stap moeten doen; expliciet benoemen.

0.5p Consequente conclusie.

Opdracht 2: frequentie van maximum — uitwerking

Antwoord

Stappen tellen

- Linkerhelft: maximum-algoritme.
  We gaan precies één keer door de hele lijst heen. Dat zijn n stappen.
- Onderste helft: aantal-algoritme.
  Hetzelfde. Ook n stappen.

Hoeveel werk?

- Linkerhelft: in iedere iteratie doen we een vergelijking en eventueel een variabele instellen. Dit is onafhankelijk van de grootte van de lijst, dus dit kost $\mathcal{O}(1)$ werk.
- Onderste helft: hetzelfde. Ook $\mathcal{O}(1)$ werk.

Orde

Breng alles samen en laat de constante voorfactor weg:

$$n \cdot \mathcal{O}(1) + n \cdot \mathcal{O}(1) = \mathcal{O}(2n) = \mathcal{O}(n)$$

Misconceptie

In het tweede algoritme komt N voor. Een misconceptie van een leerling kan zijn dat de leerling denkt dat dit algoritme dan $\mathcal{O}(n)$ werk moet doen. Om de leerling niet onnodig te misleiden, gebruiken wij expres een hoofdletter N in het stroomdiagram.

Punten

1p Inzicht dat we n stappen moeten zetten.

1p Inzicht dat we $\mathcal{O}(1)$ werk per stap moeten doen.

1p Inzicht dat bovenste en onderste helft apart moeten worden bepaald, en daarna bij elkaar opgeteld.

1p Consequente conclusie.

Opdracht 3: overeenkomsten vinden — uitwerking

Antwoord (a)

Stappen tellen

We hebben een herhaling in een herhaling. In totaal zijn er n stappen in de eerste herhaling, en voor iedere iteratie zijn er weer m stappen van de tweede herhaling. Dit zijn in totaal $n \cdot m$ stappen.

Hoeveel werk?

We moeten twee elementen met elkaar vergelijken, en eventueel dit printen als ze hetzelfde zijn. Dit is onafhankelijk van de groottes van beide lijsten, dus $\mathcal{O}(1)$ werk.

Orde

Breng alles samen en laat de constante voorfactor weg:

$$n \cdot m \cdot \mathcal{O}(1) = \mathcal{O}(n \cdot m)$$

Punten (a)

1p Inzicht dat we $n \cdot m$ stappen moeten zetten.

1p Inzicht dat we $\mathcal{O}(1)$ werk per stap moeten zetten.

1p Consequente conclusie.

Antwoord (b)

Stappen tellen

In het **worstcasescenario** moeten we alle elementen van beide lijsten één keer bekijken. Dit zijn $n + m$ elementen.

Hoeveel werk?

In iedere iteratie vergelijken we twee getallen en printen eventueel het resultaat. Dit is onafhankelijk van de groottes van beide lijsten, dus $\mathcal{O}(1)$ werk.

Orde

Breng alles samen en laat de constante voorfactor weg:

$$(n + m) \cdot \mathcal{O}(1) = \mathcal{O}(n + m)$$

Maar het algoritme moet eerst ook **quicksort** op beide lijsten toepassen. Dit is $\mathcal{O}(n \cdot \lg n) + \mathcal{O}(m \cdot \lg m)$ werk. Breng dit samen:

$$\begin{aligned} &\mathcal{O}(n \cdot \lg n) + \mathcal{O}(m \cdot \lg m) + \mathcal{O}(n + m) \\ =&\mathcal{O}(n \cdot \lg n + m \cdot \lg m + n + m) \end{aligned}$$

Behoud vervolgens alleen de snelst stijgende termen in n en m:

$$\mathcal{O}(n \cdot \lg n + m \cdot \lg m)$$

Punten (b)

1p Inzicht dat we $n + m$ stappen moeten zetten.

1p Inzicht dat we $\mathcal{O}(1)$ werk per stap moeten zetten.

1p Inzicht dat de totale orde quicksort(n) + quicksort(m) + zoeken wordt.

1p Consequente conclusie.

Opdracht 4: decimaal naar binair — uitwerking

Antwoord

Stappen tellen

In het voorbeeld moesten we voor het decimale getal 50 in totaal 6 stappen zetten. Dit is altijd precies gelijk aan het aantal bits van het resulterende binaire getal.

Wiskundige uitleg

Stel dat we b bits hebben. We hebben b stappen nodig om deze b bits te berekenen. Daarmee kunnen we hoogstens het decimale getal $N = 2^{b+1} - 1$ maken. Deze uitdrukking kunnen we omschrijven:

$$\begin{aligned} N - 1 &= 2^{b+1} \\ \lg(N-1) &= b + 1 \\ b &= \lg(N-1) - 1 \end{aligned}$$

Als we dit vereenvoudigen voor grote N, vinden we dat het aantal stappen $\mathcal{O}(\lg N)$ is.

Uitleg met een argument

Het aantal bits is gelijk aan het aantal keer dat we het getal N door 2 kunnen delen. Bijvoorbeeld:

$$50 \underbrace{/2/2/2/2/2/2}_{6 \text{ keer}} \approx 1$$

Het aantal keer dat je door 2 kunt delen is per definitie de logaritme met grondtal 2. Het aantal stappen is dus $\mathcal{O}(\lg N)$.

Hoeveel werk?

Dit algoritme deelt in iedere stap door 2. Delen is een $\mathcal{O}(m \log m)$-operatie. Deze getallen bestaan uit $m = \log_{10}(N)$ cijfers. Dus het zou dan gaan om $\mathcal{O}(\log N \cdot \log(\log N))$ werk per stap.

Echter, door 2 delen is voor de computer het opschuiven van maar één bit. Dat kost maar $\mathcal{O}(1)$ werk. Bovendien is de rekentijd hiervan totaal verwaarloosbaar voor alle realistische waarden van N, waardoor dit algoritme altijd effectief $\mathcal{O}(1)$ werk zal doen.

Orde

Breng alles samen:

$$\mathcal{O}(\lg N) \cdot \mathcal{O}(1) = \mathcal{O}(\lg N)$$

Punten

- 1p Inzicht dat het aantal stappen gelijk is aan het aantal bits.
- 1p Inzicht dat dit tot een $\lg N$-verband leidt.
- 0.5p Inzicht dat we $\mathcal{O}(1)$ werk per stap moeten doen *of* inzicht dat dit schaalt met het aantal cijfers. Leerling kan tijdens de toets niet meten of een term voor realistische N verwaarloosbaar is, dus we rekenen beide goed.
- 0.5p Consequente conclusie.

Opdracht 5: decimaal naar hexadecimaal — uitwerking

Antwoord

Stappen tellen

In het voorbeeld moesten we voor het decimale getal 500 in totaal 3 stappen zetten. Dit is altijd precies gelijk aan het aantal hexadecimale cijfers in het resulterende hexadecimale getal.

Wiskundige uitleg

Stel dat we b hexadecimale cijfers hebben. We hebben b stappen nodig om deze b hexadecimale cijfers te berekenen. Daarmee kunnen we hoogstens het decimale getal $N = 16^{b+1} - 1$ maken. Deze uitdrukking kunnen we omschrijven:

$$\begin{aligned} N - 1 = 16^{b+1} = \left(2^4\right)^{b+1} &= 2^{4b+4} \\ \lg\left(N-1\right) &= 4b + 4 \\ b &= \frac{1}{4}\lg\left(N-1\right) - 1 \end{aligned}$$

Als we dit vereenvoudigen voor grote N, vinden we dat het aantal stappen $\mathcal{O}\left(\lg N\right)$ is.

Uitleg met een argument

Het aantal bits is gelijk aan het aantal keer dat we het getal N door 16 kunnen delen. Bijvoorbeeld:

$$500 \underbrace{/16/16/16}_{\text{3 keer}} < 1$$

Het aantal keer dat je door 16 kunt delen is per definitie de logaritme met grondtal 16. Maar om aan te geven dat dit algoritme in de logaritmische efficiëntieklasse valt, maakt het niet uit welk grondtal we noteren. Dus we schrijven dat het aantal stappen $\mathcal{O}\left(\lg N\right)$ is.

Hoeveel werk?

Dit algoritme deelt in iedere stap door 16. Delen is een $\mathcal{O}\left(m \log\ m\right)$-operatie. Deze getallen bestaan uit $m = \log_{10}\ N$ cijfers. Dus het zou dan gaan om $\mathcal{O}\left(\log N \cdot \log\ \left(\log\ N\right)\right)$ werk per stap.

Echter, door 16 delen is voor de computer het opschuiven van maar vier bits. Dat kost maar $\mathcal{O}\left(1\right)$ werk. Bovendien is de rekentijd hiervan totaal verwaarloosbaar voor alle realistische waarden van N, waardoor dit algoritme altijd effectief $\mathcal{O}\left(1\right)$ werk zal doen.

Orde

Breng alles samen:

$$\mathcal{O}\left(\lg N\right) \cdot \mathcal{O}\left(1\right) = \mathcal{O}\left(\lg N\right)$$

Punten

- 1p Inzicht dat het aantal stappen gelijk is aan het aantal 'bits'.
- 1p Inzicht dat dit tot een lg N-verband leidt.
- 1 Inzicht dat we $\mathcal{O}\left(1\right)$ werk per stap moeten doen *of* inzicht dat dit schaalt met het aantal cijfers. Leerling kan tijdens de toets niet meten of een term voor realistische N verwaarloosbaar is, dus we rekenen beide goed.
- 0.5p Consequente conclusie.
- 0.5p Inzicht dat het grondtal niet uitmaakt: noteer log of lg, en *niet* het logaritme met grondtal 16.

Locatie van het volledige lesmateriaal

Lesmateriaal over keuzethema G. *algoritmiek, berekenbaarheid en logica*, zoals ontwikkeld in opdracht van SLO, is beschikbaar via keuzethemas.nl. Instruct heeft dit materiaal in Fundament opgenomen en verbeterd.

Het is te vinden onder 'Keuzethema's → G. Algoritmiek, berekenbaarheid en logica'. Aan het einde van onderdeel 2 in Fundament vind je ook heel veel nieuwe casussen om leerlingen te helpen oefenen met orde-analyses in nieuwe contexten. Dit is vooral belangrijk voor de vwo-eindtermen; voor havo-leerlingen is het dus niet nodig.

Ben je enthousiast geworden van de workshop, maar ben je nog niet in het bezit van een licentie voor de keuzethema's? Onze adviseurs helpen je graag verder. Neem contact op via fundament@instruct.nl!