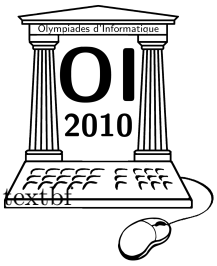


Olympiades in de Informatica

http://uclouvain.acm-sc.be/olympiades

Introductie tot de Algoritmiek

In dit document maken we kennis met algoritmiek en het gestructureerd oplossen van problemen. In het eerste deel beschrijven we stap voor stap de basisideeën van programmeren, en introduceren we pseudo-code: een notatie voor de verschillende programmeerstructuren die we zullen zien. Dit leidt tot een tweede deel, waarin algoritmiek en probleemoplossing aan bod komen. We hanteren een pedagogische stijl, waarbij we de verschillende concepten geleidelijk introduceren en telkens met voorbeelden illustreren. Dit is een voorlopige versie: elke opmerking is welkom en mag doorgegeven worden aan Joachim Ganseman : joachim.ganseman@ua.ac.be. Deze tekst is een Nederlandse vertaling van het Franstalige origineel van de hand van Sébastien Combéfis. [1]

1 Introductie

Laat ons beginnen met een eenvoudig probleem:

> Ik wil een rechthoekige vloer betegelen, die 2,5 m op 4,9 m meet. Wat is de oppervlakte in m^2 die ik moet betegelen?

Allereerst moeten we aandachtig de opgave lezen. Daaruit halen we twee dingen: de *input*, d.w.z. de gegevens die je krijgt, en de *output*, d.w.z. de resultaten die je moet berekenen. In het bovenstaande probleem vinden we als gegevens een lengte van 2,5 m en een breedte van 4,9 m. Het resultaat dat we zoeken is de oppervlakte van deze vloer.

Nu input en output gekend zijn, moeten we beschrijven hoe we de output kunnen berekenen, vertrekkend van de input. We zoeken een *systematische manier om de output te berekenen*. In ons geval, moeten we de oppervlakte berekenen van een rechthoek waarvan we de lengte en breedte kennen. Het is hier dus voldoende om die twee waarden met elkaar te vermenigvuldigen: $2{,}5\,\text{m} \times 4{,}9\,\text{m} = 12{,}25\,\text{m}^2$.

Het voorgaande probleem was wel erg *specifiek*: we hadden slechts één precieze situatie waarbij we een rechthoekige vloer van 2,5 m op 4,9 m moesten betegelen. Die afmetingen komen we maar zelden tegen. Het *algemene* probleem is veel interessanter, want dat kunnen we toepassen op verschillende specifieke situaties.

Hier is een algemenere versie van ons betegelingsprobleem:

> Ik wil een rechthoekige vloer betegelen van een zeker lengte l en een zekere breedte b, beide gegeven in meters. Wat is de oppervlakte in m^2 die ik moet betegelen?

Deze keer zijn er geen precieze waarden voorzien in de gegevens. In de plaats daarvan hebben we *variabelen* die met willekeurige waarden kunnen overeenkomen. Er zijn 2 variabelen: l en b. Het resultaat van het probleem is eenvoudig te berekenen door deze twee variabelen met elkaar te vermenigvuldigen: $l \times b$. We kunnen de oplossing van dit probleem op een systematische manier schrijven als een *algoritme*.

Woord	definieert
Input	de gegevens
Output	de gewenste resultaten
return	het eindresultaat

Algorithm 1: Berekening van de oppervlakte van een rechthoekige vloer.

Input: l en b, lengte en breedte van een vloer in m
Output: Oppervlakte van de rechthoekige vloer

```
1 return l × b ;
```

Er worden 3 woorden in het vet gebruikt in dit algoritme. Elk van deze woorden definieert de verschillende elementen ervan. De onderstaande tabel geeft een overzicht van deze woorden en hun betekenis.

Het algemene probleem kan ook worden gebruikt om het specifieke probleem dat we eerder zagen op te lossen. Het volstaat om de waarde 2,5 toe te kennen aan de variabele l, en de waarde 4,9 aan b. Door op die manier concrete waarden toe te kennen aan de inputvariabelen, lossen we een specifieke *instantie* van het algemene probleem op. Het berekende resultaat is dan ook een concrete waarde: de oplossing van dat specifieke probleem.

1.1 Oefeningen

1. Vertrek van het volgende specifieke probleem, en geef de algemene versie ervan. Schrijf dan het volledige algoritme dat toelaat om het probleem op te lossen.

   > Ik leerde een nieuw kaartspel waarbij elke speler eerst een hand van 4 kaarten krijgt. We gebruiken een klassiek kaartspel van 52 kaarten. Hoeveel verschillende handen kan ik toebedeeld krijgen?

2. Het volgende algoritme berekent het volume van een cylinder. Vervolledig het.

Algorithm 2: Berekening van het volume van een cylinder.

Input: R en H, straal en hoogte van de cylinder in m
Output: Volume van de cylinder

```
1 // Te vervolledigen ... ;
```

2 Variabelen

Nu we de basisbegrippen van het oplossen van problemen door een algoritme hebben gezien, leren we beetje bij beetje de verschillende constructies aan die ons zullen toelaten om interessantere problemen op te lossen. Als we lange berekeningen maken, maken we vaak gebruik van tussenresultaten onderweg. Bijvoorbeeld, als we de wortels van een tweedegraadsfunctie $P \equiv ax^2 + bx + c$ berekenen (in het

1. Opmerkingen bij de originele, Franstalige versie, mag u in het Frans of in het Engels ook steeds richten tot de auteur Sébastien Combéfis : sebastien.combefis@uclouvain.acm-sc.be

geval dat er twee verschillende reële wortels zijn), berekenen we dit in twee stappen. We beginnen met de determinant

$$\Delta = b^2 - 4ac$$

en berekenen daarmee de twee wortels

$$x_1 = \frac{-b + \sqrt{\Delta}}{2a} \qquad \text{et} \qquad x_2 = \frac{-b - \sqrt{\Delta}}{2a}.$$

Met het volgende algoritme lossen we dit probleem op. Merk op dat het algoritme twee resultaten berekent, wat we hier noteren door die waarden na elkaar te schrijven, gescheiden door een komma, en het geheel tussen vierkante haakjes te zetten. Hier komen we later nog op terug.

Algorithm 3: Berekening van de wortels van een tweedegraadsfunctie.

Input: a, b en c, parameters van de veelterm $P \equiv ax^2 + bx + c$
Output: De twee reële wortels van de veelterm

1 $\Delta \leftarrow b^2 - 4ac$;
2 $x_1 \leftarrow \dfrac{-b + \sqrt{\Delta}}{2a}$;
3 $x_2 \leftarrow \dfrac{-b - \sqrt{\Delta}}{2a}$;
4 **return** $[x_1, x_2]$;

In dit algoritme hebben we drie inputgegevens: de variabelen a, b en c. Zoals je kan lezen in het algoritme, noemen we ze *parameters*, om ze te onderscheiden van andere variabelen die we kunnen gebruiken in het algoritme zelf. Van die andere variabelen zijn er 3: Δ, x_1 en x_2. *Variabelen* hebben steeds een *naam* en een *waarde*, en we gebruiken ze vooral om tussentijdse resultaten in op te slaan. We kunnen ze gebruiken in een verdere berekening, en we kunnen hun waarde verderop steeds aanpassen. We gebruiken de notatie $x \leftarrow v$ om aan te duiden dat de waarde van variabele x verandert in v. Dat doen we ook in het bovenstaande algoritme. Daarin berekenen we de determinant $(b^2 - 4ac)$ en we slaan het resultaat op in de variabele Δ. Daarna gebruiken we dat tussenresultaat om de twee wortels te berekenen, die we opslaan in de variabelen x_1 en x_2. Tenslotte geven we het resultaat van het algoritme terug, dat bestaat uit de waarden van de variabelen $([x_1, x_2])$.

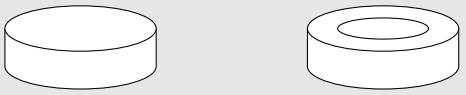
2.1 Expressies

Als we een berekening opschrijven, schrijven we een *expressie*. Dit is een berekening die gebruikmaakt van getallen, variabelen en operatoren, en die leidt tot een zekere waarde, het resultaat van de berekening. Tot nu toe zagen we *wiskundige expressies*, waarvan het resultaat steeds een getal is. Zo hebben we bijvoorbeeld: $l \times b$, b^2, $\sqrt{x+1}$, ... We bouwen een expressie op door getallen, variabelen en operatoren te combineren. We kunnen dus ook schrijven: $x + y$, $x - y$, $x \times y$, x/y, $\sqrt{x}$, x^n, ...

Een tweede type expressies zijn *booleaanse expressies*. Daarvan is het resultaat één van twee waarden: waar (genoteerd als **true**) of onwaar (genoteerd als **false**). We komen tot deze waarden als we *relationele operatoren* gebruiken, zoals $=$, $\neq$, $<$, $>$, $\leq$, $\geq$, waarmee we twee waarden kunnen vergelijken met elkaar. We kunnen dus schrijven: $x \neq 0$, $b^2 - 4ac > 0$, $x \neq y$, ...

Deze nieuwe concepten kunnen we gebruiken in het volgende probleem:

Ik heb twee verschillende types cylinders, met dezelfde hoogte H. De cylinders van type I zijn vol, terwijl die van type II hol zijn, zoals op de tekening hieronder. We willen weten: heeft een cylinder van type I met straal R meer volume dan een cylinder van type II met uitwendige straal R_O en inwendige straal R_I?



We lossen het probleem als volgt op. Er zijn 4 gegevens: de hoogte van de cylinders H, de straal R van de cylinder van type I, en de uitwendige en inwendige straal R_O en R_I van de cylinder van type II. Het resultaat is een booleaanse waarde: ja (**true**) of nee **false**). De formule voor het volume van een cylinder moest je vinden in oefening 1.2 . Ze is $V = \pi R^2 H$. Het algoritme is vrij eenvoudig: bereken het volume van beide cylinders en vergelijk ze. Voor de tweede cylinder berekenen we het uitwendige volume en we trekken het inwendige volume van de holte ervan af. Het volgend algoritme doet dit alles:

Algorithm 4: Vergelijking van het volume van cylinders van verschillend type

Input: H: hoogte van de cylinders; R: straal van cylinder type I; R_O, R_I: uitwendige en inwendige straal van cylinder type II

Output: **true** als de cylinder van type I een groter volume heeft dan die van type II, en anders **false**

```
1 // Volume van de eerste cylinder ;
2 V_1 ← πR²H ;
3 // Volume van de tweede cylinder ;
4 V_O ← πR_O²H ;
5 V_I ← πR_I²H ;
6 V_2 ← V_O − V_I ;
7 return V_1 ≥ V_2 ;
```

De berekende volumes van de twee cylinders werden opgeslagen in de variabelen V_1 en V_2. Voor de tweede cylinder hebben we die berekening in meerdere stappen gedaan. Tussentijdse resultaten werden opgeslagen in variabelen V_O en V_I. Het eindresultaat is een booleaanse waarde, $V_1 \geq V_2$ (d.w.z.: is het volume van de eerste cylinder groter of gelijk aan dat van de tweede?)

2.2 Gehele deling en rest

Als we alleen met gehele getallen werken, gedraagt de deling zich anders dan je meestal gewend bent. We spreken van de *gehele deling* (ook bekend als staartdeling): het quotiënt van $a$div$b$ is ook een geheel getal q. Zo is het quotiënt van 12 gedeeld door 3 gelijk aan 4. We noteren de gehele deling met : "div". Die deling is niet altijd exact. Als b geen deler is van a (m.a.w. a is niet deelbaar door b), dan is er een rest r. Ook deze rest kunnen we berekenen en we noteren ze met : "mod". Voorbeeld: de rest van de gehele deling van 12 door 7 is 5. Inderdaad kunnen we 12 schrijven als $1 \times 7 + 5$. In het algemeen kunnen we schrijven:

$$a = q \times b + r$$

waarbij q het quotiënt is van de gehele deling van a door b, en r de rest is van dezelfde deling. Laat ons een concreet voorbeeld nemen waarbij we die notatie zullen gebruiken. We schrijven een algoritme dat test of een geheel getal even is. Het algoritme moet een booleaanse waarde berekenen: **true** als het getal even is, anders **false**. Om dat te bekomen is het voldoende om te kijken wat de rest is na deling door 2. Die rest zal 0 zijn als het getal even is, en anders niet.

Algorithm 5: Test of een natuurlijk getal even is.

Input: n, een positief geheel getal
Output: **true** als n even is, en anders **false**

```
1 return n mod 2 = 0 ;
```

2.3 Oefeningen

1. Schrijf een algoritme dat twee natuurlijke getallen a en b neemt, en test of a een veelvoud is van b. Het resultaat van dit algoritme moet dus **true** zijn als a een veelvoud is van b, en anders **false**.

2. Schrijf een algoritme dat berekent wat het volume is dat zit tussen een cylinder met straal R en hoogte H, en de omvattende minimale rechthoekige balk (dat is een balk met lengte en breedte $2R$ en hoogte H).

3. Ik heb een fles met een capaciteit van 1,5 l en een glas met een capaciteit van 210 ml. Schrijf een algoritme dat berekent hoeveel volledig gevulde glazen ik nodig heb om de fles helemaal te vullen.

4. Schrijf een algoritme dat de conversie doet van graden Fahrenheit (t_F) naar graden Celsius (t_C). Ter herinnering, de formule is: $t_C = 5/9 \times (t_F - 32)$.

3 Condities

In een algoritme is het soms nodig om keuzes te maken. We zien zo dadelijk een structuur die dit toelaat, gebaseerd op een *conditie*. Deze conditie is een booleaanse waarde. Laat ons beginnen met een voorbeeld: een algoritme dat de absolute waarde van een reëel getal berekent.

Algorithm 6: Berekening van de absolute waarde van een getal

Input: n, een getal
Output: De absolute waarde van dit getal

```
1 if n < 0 then
2 |  n ← (−1) × n ;
3 return n ;
```

Om de absolute waarde van een getal n te berekenen moeten we dat getal vermenigvuldigen met -1 als het negatief is. Dat is exact wat er gebeurt op lijnen 1 en 2. In het geval dat het gegeven getal positief is, moet er niets gebeuren. Om een conditie uit te drukken, gebruiken we het woord **if**, gevolgd door de conditie, en daarna het woord **then**. Daarop volgt alles wat er moet gebeuren als aan die conditie voldaan is.

Soms willen we niet alleen iets doen als aan een conditie voldaan is, maar ook iets anders doen

als er niet aan de conditie voldaan is. We schrijven bijvoorbeeld een algoritme dat twee getallen vergelijkt, en het grootste selecteert.

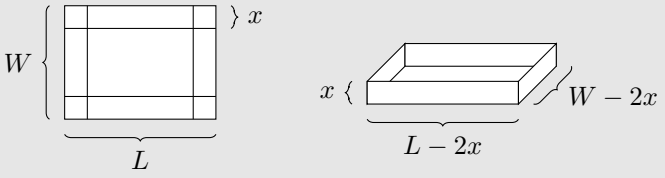
Algorithm 7: Berekening van het maximum van 2 getallen

Input: a, b, twee getallen
Output: Het grootste van de getallen a en b

```
1 if a > b then
2 |  max ← a ;
3 else
4 |  max ← b ;
5 return max ;
```

De conditie in **if** test of getal a groter is dan getal b. Als dit het geval is, slaan we de waarde a op in de tussentijdse variabele max. Indien niet, duiden we aan wat er moet gebeuren met het woord **else**. Hier willen we, als b groter of gelijk is aan a, het getal b opslaan in max. Op die manier bevat de variabele max aan het eind de grootste van de twee getallen a en b, wat exact is wat we wilden berekenen.

Hier is een nieuw, ingewikkelder probleem waarbij we een **if-else** structuur goed kunnen gebruiken:

> Ik heb een blad karton met lengte L en breedte W. Hiermee kunnen we een open doos maken met een zekere hoogte x. Voor welke hoogte x krijgen we een doos met maximale inhoud?
>
> 
>

Het volume van de doos, een balk, is de vermenigvuldiging van de drie zijden:

$$V = x(L-2x)(W-2x) = 4x^3 - 2(L+W)x^2 + LWx$$

Om het maximum hiervan te vinden, zoeken we de eerste afgeleide hiervan:

$$V' = 12x^2 - 4(L+W)x + LW$$

Daar waar dit gelijk is aan 0, is het maximum. We zoeken dus de wortels van de vergelijking. We gaan verder zoals in deel 2 van deze tekst, maar we moeten een extra voorwaarde inbouwen. Als de determinant van deze tweedegraadsvergelijking 0 is, is er slechts 1 reële wortel, en als ze positief is zijn er twee. In dat laatste geval nemen we de wortel die het hoogste volume teruggeeft. De determinant is:

$$\Delta = \big(-4(L+W)\big)^2 - 4 \cdot 12 \cdot LW = 16(L^2+W^2+2LW) - 48LW = 16(L^2+W^2-LW)$$

De wortels van de vergelijking $V' = 0$ worden dus gegeven door (in het geval dat $\Delta > 0$):

$$x_1, x_2 = \frac{4(L+W) \pm 4\sqrt{L^2+W^2-LW}}{24} = \frac{(L+W) \pm \sqrt{L^2+W^2-LW}}{6}$$

Als $\Delta = 0$, is de enige bestaande wortel:

$$x = \frac{-\big(-4(L+W)\big)}{24} = \frac{L+W}{6}$$

Tenslotte, als $\Delta < 0$, is er geen reële wortel.

Algorithm 8: Optimale hoogte voor een doos gemaakt van een blad karton.

Input: W, L, twee strikt positieve gehele getallen, breedte en lengte van een stuk karton
Output: x, hoogte van de open doos met maximale inhoud die met dat karton gevormd kan worden

```
1  // Berekening van de determinant ;
2  Δ = 16(L² + W² − LW) ;
3  // We beschouwen verschillende gevallen afhankelijk van de waarde van Δ ;
4  // Δ < 0, geen wortels, het algoritme geeft 0 terug ;
5  if Δ < 0 then
6  |  x ← 0 ;
7  // Δ = 0, één enkele wortel gegeven door −b/2a ;
8  else if Δ = 0 then
9  |  x ← (L + W)/6 ;
10 // Δ > 0, twee oplossingen (−b ± √Δ)/2a ;
11 // Hiervan kiezen we diegene die het grootste volume teruggeeft ;
12 else
13 |  temp ← √(L² + W² − LW) ;
14 |  x1 ← (L + W + temp)/6 ;
15 |  x2 ← (L + W − temp)/6 ;
16 |  // We nemen de grootste van x1 en x2 ;
17 |  if x1 > x2 then
18 |  |  x ← x1 ;
19 |  else
20 |  |  x ← x2 ;
21 return x ;
```

Hier kunnen we verschillende opmerkingen bij maken. Eerst en vooral dat we 3 verschillende gevallen onderscheiden, afhankelijk van de waarde van Δ. Om dat te doen, kunnen we verschillende **if-else** structuren in ketting zetten, en we komen dan tot de volgende structuur: **if** (voor $\Delta < 0$), **else if** (voor $\Delta = 0$) en **else** (voor $\Delta < 0$). Wanneer we veel mogelijkheden moeten onderscheiden kunnen we dit soort structuur gebruiken. Als er slechts 1 mogelijkheid is, is een eenvoudige **if** voldoende, en bij 2 mogelijkheden kunnen we **if-else** gebruiken.

De tweede opmerking gaat over het geval dat $\Delta > 0$. We berekenen twee wortels, en moeten daarna die met het grootste resultaat teruggeven. Je ziet gemakkelijk dat dit hetzelfde is als wat we al gedaan hebben in het algoritme 7 op pagina 6. Later zullen we zien dat we dit kunnen hergebruiken, om te vermijden dat we hetzelfde werk 2 keer doen.

Tenslotte nog dit over de berekening van x_1 en x_2. Om het algoritme beter leesbaar te maken, hebben we eerst de waarde $\sqrt{L^2 + W^2 - LW}$ berekend en opgeslagen in een variabele *temp*. Zo moeten we hetzelfde niet twee keer schrijven, en het maakt de code zelf ook beter leesbaar. Dit is een goede gewoonte.

3.1 Booleaanse Algebra

De condities die we tot nu toe hebben gebruikt waren allemaal *enkelvoudig*. Ze bestonden uit één operator en twee waarden of operanden. De operatoren die we kunnen gebruiken zijn: $=$, $\neq$, $<$, $>$, $\leq$ en $\geq$. Nu gaan we meerdere condities met elkaar combineren door middel van *logische operatoren*. Beeld je in dat je wil testen of een waarde voor x gelegen is tussen 12 en 20. Dan kunnen we schrijven: ($x \geq 12$ **and** $x \leq 20$). De logische operator **and** laat toe om twee condities te combineren. De waarde van de conditie A **and** B is **true** als A en B allebei **true** zijn; in alle andere gevallen is het resultaat **false**. Met de operator **and** kunnen we dus testen of aan twee voorwaarden tegelijk is voldaan.

Er is ook een operator **or** die toelaat om te testen of één, dan wel een andere conditie waar is. De waarde van de conditie A **or** B is **true** als A, of B, of allebei **true** zijn. Het resultaat is **false** als A en B allebei **false** zijn.

Tenslotte is er de operator **not** die toegepast kan worden op een conditie. De waarde van de conditie **not** A is **true** als A zelf gelijk is aan **false**, en vice versa.

In de tabel hieronder staan de drie operatoren **and**, **or** en **not** opgesomd, samen met de waarden waartoe ze leiden als ze toegepast worden op A en B.

A	B	**not** A	A **and** B	A **or** B
false	**false**	**true**	**false**	**false**
false	**true**	**true**	**false**	**true**
true	**false**	**false**	**false**	**true**
true	**true**	**false**	**true**	**true**

3.2 Oefeningen

1. Schrijf een algoritme dat de waarde berekent van de functie $f(x) = |x^2 - 2x - 10|$ voor een gegeven x.

2. Schrijf een functie dat het resultaat (geheel getal) van een toets op 20 punten kan converteren naar de volgende notatie: strikt minder dan 8 : F, tussen 8 en 10 : D, tussen 11 en 13 : C, tussen 14 en 16 : B, tussen 17 en 20 : A.
   (Om een algoritme tekst te laten teruggeven in plaats van een getalwaarde of variabele, kan je schrijven : **print** "tekst om terug te geven")

4 Lussen

Daarnet zagen we hoe we ervoor konden zorgen slechts een deel van het algoritme uit te voeren, door te testen of aan een voorwaarde was voldaan. Nu zien we een andere belangrijke structuur, de *lussen*. Die laatsten staan toe om een deel van een algoritme meerdere keren te herhalen. We vertrekken van een eenvoudig voorbeeld dat het principe moet uitleggen: laat ons de som berekenen van de reeks $1 + 2 + \cdots + n$, zonder daarvoor de formule $n(n+1)/2$ te gebruiken.

Algorithm 9: Berekening van de som van de eerste n positieve gehele getallen.

Input: n, een natuurlijk getal
Output: De som $1 + 2 + \cdots + n$

```
1 i ← 1 ;
2 sum ← 0 ;
3 while i ≤ n do
4 |   sum ← sum + i ;
5 |   i ← i + 1 ;
6 return sum ;
```

Dit algoritme werkt als volgt. Eerst definiëren we twee variabelen i en sum. De variabele sum laat ons toe om de som te berekenen, en de variabele i gaat de opeenvolgende termen opslaan die we willen toevoegen aan de som, te weten $1, 2, \cdots, n$.

Daarna schrijven we **while**, gevolgd door een conditie, gevolgd door **do**. Al wat daarna volgt wordt herhaald zolang er aan de conditie in **while** is voldaan, d.w.z. zolang de waarde van die conditie **true** is. En dus, zolang de waarde van i kleiner of gelijk is aan n, voegen we de waarde van i toe aan de variabele sum en verhogen we de de waarde van i met 1. Stel bijvoorbeeld dat n gelijk is aan 3. De variabele i heeft als initiële waarde 1 gekregen en de lus zal zich herhalen zolang $i \leq 3$. Als we dan weten dat de waarde van i telkens verhoogd wordt met 1 aan het eind van deze lus, kunnen we afleiden dat deze lus zich 3 keer zal herhalen: voor i met de waarden 1, 2 en 3. Als we dit zonder lus helemaal zouden uitschrijven, zouden we het volgende krijgen:

```
i ← 1
sum ← 0
// Nu is i gelijk aan 1 en dus, i ≤ 3.
sum ← sum + i
i ← i + 1
// Nu is i gelijk aan 2 en dus, i ≤ 3.
sum ← sum + i
i ← i + 1
// Nu is i gelijk aan 3 en dus, i ≤ 3.
sum ← sum + i
i ← i + 1
// Nu is i gelijk aan 4 en dus stopt de lus.
// De variabele sum bevat nu 1 + 2 + 3, dus 6.
```

Het volgende probleem zullen we met behulp van een lus kunnen oplossen.

> Een positief geheel getal n is een priemgetal als het exact twee delers heeft (1 en n). Schrijf een algoritme dat test of een gegeven getal n een priemgetal is of niet.

Een eenvoudig algoritme om dat te doen bestaat er in om alle getallen te testen tussen 1 en n. We onthouden hoeveel ervan delers zijn van n (dit kunnen we bepalen door te kijken naar de rest van de gehele deling). Het getal n is een priemgetal als het exact 2 delers heeft.

Algorithm 10: Test of een getal een priemgetal is

Input: $n > 0$, een natuurlijk getal
Output: **true** als n priemegetal is, anders **false**

```
1  i ← 1 ;
2  cnt ← 0 ;
3  // We testen alle i die liggen tussen 1 en n ;
4  while i ≤ n do
5      // Als i een deler is van n, verhogen we de waarde van cnt met 1 ;
6      if n mod i = 0 then
7          cnt ← cnt + 1 ;
8      // We gaan verder met de volgende i ;
9      i ← i + 1 ;
10 // n is priem als het exact 2 delers heeft ;
11 return cnt = 2 ;
```

Inhoudsopgave