De staartdeling mag dan wellicht verdwijnen uit het basisonderwijs, in het universitair informaticaonderwijs leeft hij voort. **Wim Pijls** gebruikt de staartdeling om het begrip algoritme te introduceren, en om de correctheid van een algoritme te bewijzen. In dit artikel leest u hoe dat in zijn werk gaat. Staartdelingen voor gevorderden...

Rekenen met algoritmen

Inleiding

Afgelopen november kopte *NRC-Handelsblad* 'Requiem voor een staartdeling'. De staartdeling gaat verdwijnen uit de basisschool. Ikzelf gebruik de staartdeling in het universitair informaticaonderwijs ter introductie van het begrip algoritme en ter illustratie van het bewijzen van algoritmen. De rekenkundige methoden voor optellen en aftrekken (met 'onthouden') en voor vermenigvuldigen en delen (staartdelen) zijn algoritmen in de oorspronkelijke betekenis van het woord. De Arabische wiskundige Al-Khwarizmi schreef in 830 na Chr een rekenboek dat pas drie eeuwen later, na een vertaling in het Latijn, in West-Europa bekendheid kreeg [4]. De naam van de auteur werd verlatijnst tot algorismi, waar ons woord algoritme van is afgeleid.
Er is dan ook geen enkele reden om 'algoritme' met een 'h' te spellen. De Griekse letter theta wordt meestal wel in een 'th' omgezet. De woorden afgeleid van $\alpha\rho\iota\tau\mu\sigma$ (= getal) zoals logarithme en arithmetisch hebben daarom wel een 'th'. Dankzij dit boek gebruiken wij tegenwoordig het tientallige positiestelsel in plaats van de tot dan toe gebruikte Romeinse cijfers.
Algoritmen zijn wiskundige concepten waarop wiskundige activiteiten worden toegepast zoals bewijzen, generaliseren, variëren enzovoort. Het zijn weliswaar geen stellingen, maar men kan ze goed vergelijken met constructies in de meetkunde. In die zin zou je de staartdeling en verwante technieken in de VWO-wiskunde weer te voorschijn kunnen halen. Ofschoon algoritmen al bij Euclides (300 voor Chr) optreden, zijn pas in de vorige eeuw formele theorieën dienaangaande ontwikkeld. De correctheid van algoritmen wordt bewezen met behulp van zogeheten invarianten. Staartdelingen evenals de andere rekentrucs zijn geschikte voorbeelden bij het uitleggen van deze in de informatica gebruikte bewijsmethodiek. We zullen in dit artikel enkele bekende en minder bekende rekentechnieken presenteren, in combinatie met de bijbehorende bewijzen.
De lezer ziet hiermee dat algoritmiek, een onderdeel van de informatica, eigenlijk puur wiskundig van aard is. Feitelijk leert de wiskundige lezer hiermee een nieuw deelgebied van de wiskunde kennen. We gaan er wel van uit, dat de lezer over enige elementaire kennis van programmeren beschikt; in het bijzonder moet hij/zij weet hebben van de toekenningsopdracht en van while- en if-constructies.
We gaan in dit artikel de rekenkundige operaties *div* en *mod* gebruiken. Voor gegeven natuurlijke getallen is x *div* y de entierwaarde van x/y, meestal genoteerd als $[x/y]$ en is x *mod* y de rest bij deling van x door y. Merk op dat $x-(x \text{ mod } y)$ altijd een y-voud is.

Vermenigvuldigen

Een methode om getallen te vermenigvuldigen is die van het zogeheten Russisch vermenigvuldigen. Deze methode is onder andere te vinden in [1] en in diverse boeken over recreatieve wiskunde en wordt ook wel Egyptisch vermenigvuldigen genoemd.
Stel we willen twee factoren $f_1 = 357$ en $f_2 = 836$ met elkaar vermenigvuldigen. Het volgende schema wordt dan opgesteld:

k	a	b	$m=2^k$	c	
0	357 → 356	836	1	836	(=836+0)
1	178	1672	2		
2	89 → 88	3344	4	4180	(=3344+836)
3	44	6688	8		
4	22	13376	16		
5	11 → 10	26752	32	30932	(=26752+4180)
6	5 → 4	53504	64	84436	(=53504+30932)
7	2	107008	128		
8	1 → 0	214016	256	298452	(=214016+84436)

Het schema is van boven naar beneden opgebouwd. In de eerste regel zijn in de kolommen a, b, en c respectievelijk de waarden 357, 836 en 0 geplaatst. Vervolgens wordt in de a-kolom steeds door 2 gedeeld; eventueel wordt een oneven getal eerst even gemaakt. In de b-kolom wordt steeds met 2 vermenigvuldigd. Indien in de a-kolom een oneven getal in een even verandert, wordt de c-waarde

met de actuele *b*-waarde verhoogd. De *m*-kolom (*m* = macht) bevat de machten van 2 en geeft aan met welke macht de beginwaarde 836 inmiddels is vermenigvuldigd. De *k*- en de *m*-kolom zijn niet wezenlijk in het proces. Deze vermenigvuldigmethode is de pendant in het tweetallig stelsel van een methode die in het tientallig stelsel overbekend is.

k	a	b	$m=10^k$	c	
0	357 → 350	836	1	5852	(= 7 × 836 + 0)
1	35 → 30	8360	10	47652	(= 5 × 8360 + 5852)
2	3 → 0	83600	100	298452	(= 3 × 836000 + 47652)

Beide algoritmen worden door de code in figuur 1 beschreven.
Ingeval de eerste opdracht $r = 2$ (r = radix) is, heeft men het Russisch vermenigvuldigen; als men $r = 10$ neemt, heeft men de tweede bovenstaande methode.
De fragmenten tussen de accolades maken geen deel uit van het programma, maar hebben betrekking op relaties tussen variabelen. Deze relaties tussen variabelen worden in de informatica ook wel asserties genoemd.

```
(1)    r = 2, of r = 10;
(2)    k = 0; a = f1; b = f2; c = 0          {invariant}
(3)    while a > 0 do
(4)        [ if a mod r > 0 then
(5)            [ a = a – (a mod r);
(6)              c = c + b × (a mod r);
(7)        ]                                 {invariant ∧ a mod r = 0}
(8)        k = k + 1;
(9)        b = b × r;
(10)       a = a / r;                        {invariant}
(11)       ]                                 {invariant ∧ a = 0}

(4')       while a mod r > 0 do
(5')         [   a = a + 1;
(6')             c = c + b;
(7')         ]                               {invariant ∧ a mod r = 0}
```

fig. 1 Een algoritme voor vermenigvuldigen

Tijdens de executie van dit algoritme geldt een invariante relatie, kortweg invariant genoemd. Deze heeft de volgende vorm:

$$a \cdot b + c = f_1 \cdot f_2 \wedge \quad b = f_2 \cdot r^k$$

Na de initialisatie van de variabelen in regel (2) geldt deze relatie. Na de statement in regel (5) is de invariant even ongeldig, om in regel (6) weer hersteld te worden. Eenzelfde effect zien we in de regels (8), (9) en (10). Na regel (8) is de gelijkheid $b = f_2 \cdot r^k$ niet meer geldig, maar de geldigheid wordt meteen in (9) hersteld. Wanneer regel (10) begint, is a een r-voud; in de regels (9) en (10) worden b en a met een factor r vergroot respectievelijk verkleind, zodat het product $a \cdot b$ en dus de invariante relatie intact blijft. Voor de regels (4) tot en met (7) bestaat een alternatieve beschrijving. Om a te verlagen tot een r-voud, kan men ook herhaald 1 aftrekken. Dit levert de alternatieve beschrijving waarin de regels (4) tot en met (7) vervangen zijn door (4') tot en met (7'). Het is nu nog duidelijker te zien dat de waarde $a \cdot b + c$ invariant blijft.
Behalve invariante relaties hebben we ook zogeheten postcondities. Als de executie van een while-loop wordt beëindigd is het criterium in de while-statement kennelijk overtreden. Men zegt ook wel: het stop-criterium is opgetreden. Als de while loop van de regels (4') tot en met (7') eindigt, geldt derhalve de postconditie $a \bmod r = 0$. Na afloop van de while-loop in de regels (3) tot en met (11) geldt de postconditie $a = 0$. Met deze postcondities gaat de invariant over in $c = f_1 \cdot f_2$. Na afloop van het algoritme bevat c dus het gewenste product.
We moeten tenslotte ook nog aantonen dat de postconditie echt bereikt wordt en dat de while-loop dus niet eeuwig blijft draaien. Omdat de variabele a geheeltallig is en steeds door 2 wordt gedeeld, zal die eens op 0 uitkomen.
Stel dat bovenstaand algoritme wordt uitgevoerd met $f_2 = 1$. De variabele b is dan steeds een macht van r.
De eindwaarde van c is een lineaire combinatie van machten van r: $c = \Sigma\, d_k r^k$, waarin elke d_k het resultaat is van een bewerking $a \bmod r$ en dus kleiner dan r is. Het algoritme bepaalt derhalve ingeval $f_2 = 1$ de schrijfwijze van f_1 in het r-tallig stelsel.
We hebben in deze paragraaf dus ook een algoritme voor de omvorming van een getal naar het r-tallig stelsel afgeleid en we hebben kennisgemaakt met een methode, waarmee men kan bewijzen dat een algoritme correct is, dat wil zeggen dat het algoritme inderdaad het resultaat geeft, dat beoogd wordt. De begrippen invariant en postconditie zijn in deze methode sleutelbegrippen. Het bewijzen van een invariant lijkt op bewijzen met volledige inductie. Men bewijst dat, indien een relatie op een bepaald moment geldt, deze opnieuw geldt na een volgende stap of na volgende stappen.

Delen

Naast Russisch vermenigvuldigen bestaat ook Russisch delen. Het schema uit de vorige paragraaf wordt dan in een andere volgorde opgebouwd. Stel men wil 298452 delen door 836. Het volgende schema is dan van toepassing en als volgt tot stand gekomen:

– Allereerst is kolom b ingevuld; deler 836 wordt bovenin geplaatst; de overige getallen in deze kolom worden verkregen door herhaald met 2 te vermenigvuldigen; dit vermenigvuldigingsproces wordt voortgezet tot een waarde groter dan deeltal 298452 is verkregen; in bovenstaand voorbeeld is deze waarde gelijk aan 428032. Parallel met de b-kolom worden ook de k- en de m-waarden gegenereerd; de m-waarde zal pas in paragraaf 4 een rol gaan spelen.
– Ter hoogte van het grootste getal uit b wordt in de c-kolom het deeltal 298452 en in de a-kolom het getal 0 geplaatst.

Daarna worden de kolommen a en c rijgewijs van onderen naar boven opgebouwd; in eerste instantie geldt dat in a

steeds met 2 wordt vermenigvuldigd en dat in *c* elke nieuwe waarde een kopie is van de vorige; in elke rij wordt nagegaan of de *b*-waarde afgetrokken kan worden van de *c*-waarde; is dat het geval, dan wordt de *c*-waarde verlaagd en wordt de *a*-waarde met 1 opgehoogd.

k	a	b	$m=2^k$	c	
0	357 (=178×2+1)	836	1	0	(=836-836)
1	178	1672	2		
2	89 (=44×2+1)?	3344	4	836	(=4180-3344)
3	44	6688	8		
4	22	13376	16		
5	11 (=5×2+1)	26752	32	4180	(=30932-26752)
6	5 (=2×2+1)	53504	64	30932	(=84436-53504)
7	2	107008	128		
8	1 (=0×2+1)	214016	256	84436	(=298452-214016)
9	0	428032	512	298452	

De code in figuur 2 beschrijft dit algoritme:

```
(1)    r = 2, of r = 10;
(2)    k = 0; a = 0; b = deler; c = deeltal          {invariant ∧ k ≥ 0}
(3)    while c ≥ b do
(4)        [ k = k + 1
(5)          b = b × r;
(6)        ]                                        {invariant ∧ k ≥ 0 ∧ c < b}
(7)    while k > 0 do
(8)        [ k = k – 1;
(9)          b = b / r;
(10)         a = a × r;                             {invariant ∧ k ≥ 0}
(11)         while c ≥ b do
(12)             [ a = a + 1;
(13)               c = c – b;
(14)             ]                                  {invariant ∧ k ≥ 0 ∧ c < b}
(15)       ]                                        {invariant ∧ k = 0 ∧ c < b}

(11')        if c ≥ b do
(12')            [ a = a + (c div b);
(13')              c = c mod b;
(14')            ]
```

fig. 2 Een algoritme voor delen

We hebben hier ook nog een preconditie, te weten: *deler* > 0 en *deeltal* > 0. De versie met $r = 2$ komt overeen met het Russisch delen. De versie met $r = 10$ komt overeen met het bekende staartdelen, hieronder in een iets andere configuratie weergegeven.

k	a	b	$m=10^k$	c	
0	357 (=7+10×35)	836	1	0	(=5852-7×836)
1	35 (=5+10×3)	8360	10	5852	(=47652-5×8360)
2	3 (=3+10×0)	83600	100	47652	(=298452-3×83600)
3	0	836000	1000	298452	

De invariant in figuur 2 is:

$$a \cdot b + c = deeltal \wedge b = deler \cdot r^k \wedge c \geq 0$$

Merk op dat de ongelijkheid $c \geq 0$ in (13) gehandhaafd wordt dankzij de conditie in regel (11). De postconditie van de binnenlus in (11) tot en met (14) (de 'innerloop') luidt:
$c < b$ en die van de grote lus in (7) tot en met (15) (de 'outer loop') luidt: $k = 0$. Omdat de relatie $b = deler \cdot r^k$ geldt, had het criterium in (7) ook kunnen luiden: $b > deler$. Aan de postconditie van de outer loop kan derhalve toegevoegd worden: $b = deler$.
Uit de relaties op regel (6) of (14) leiden we af:
$a \cdot b \leq a \cdot b + c = deeltal$ en $(a + 1) \cdot b = a \cdot b + b > a \cdot b + c = deeltal$.
Na afloop van het algoritme geldt $b = deler$ en we hebben dus $a \cdot deler \leq deeltal$ en $(a + 1) \cdot deler >$ deeltal, hetgeen equivalent is met $a = deeltal\ div\ deler$ en $c = deeltal\ mod\ deler$.
Omdat k geheeltallig en $k \geq 0$, zal dankzij de herhaalde bewerking $k - 1$ de postconditie ook daadwerkelijk worden bereikt.
Omdat na regel (6) of (14) geldt $0 \leq c < b$, hebben we na regel (9): $0 \leq c < r \cdot b$. Hieruit volgt dat de innerloop in (11) tot en met (14) minder dan r keer wordt doorlopen. Dit houdt in dat de while-constructie in een if-constructie met *div* en *mod* veranderd kan worden, zoals aangegeven in de regels (11') tot en met (14').

Een andere beschrijving van delen

Tot dusver speelde zich alles binnen de verzameling van de natuurlijke getallen af. De getoonde programma's gaan we nu uitbreiden naar de rationale getallen. De staartdeling van paragraaf 3 is ook met figuur 3 te beschrijven.

```
(1)    r = 2, of r = 10;
(2)    k = 0; a = 0; m = 1;
(3)    c = deeltal;                                 {invariant ∧ k ≥ 0}
(4)    while c ≥ deler do
(5)        [ k = k + 1;
(6)          m = m × r;
(7)          c = c / r;
(8)        ]                                        {invariant ∧ k ≥ 0 ∧ c < deler}
(9)    while k > 0 do
(10)       [ k = k – 1;
(11)         m = m / r;
(12)         c = c × r;                             {invariant ∧ k ≥ 0}
(13)         while c ≥ deler do
(14)           [ a = a + m;
(15)             c = c – deler;
(16)           ]                                    {invariant ∧ k ≥ 0 ∧ c < deler}
(17)       ]                                        {invariant ∧ k ≥ 0 ∧ c < deler}
```

fig. 3 Een alternatieve beschrijving voor delen

De eerste gelijkheid in de invariant van figuur 2 kunnen we schrijven als $deler \cdot a \cdot r^k + c = deeltal$.
In figuur 3 luidt de invariant:

$$deler \cdot a + c \cdot m = deeltal \wedge m = r^k \wedge c \geq 0$$

Dit houdt in dat we in de regels (9) tot en met (17) met een grotere *a*- en een kleinere *c*-waarde werken, vergeleken met figuur 2. Als we 298452 / 836 uitrekenen met

$r = 10$, dan neemt c de waarden 2,98452, 29,8452, 298,452 enzovoort aan; de variabele a is achtereenvolgens gelijk aan 300, 350 en 357. Figuur 3 is alleen handig als in het r-tallig stelsel gewerkt wordt. Ingeval $r = 2$ ontstaat een ander schema dan we voor het Russisch delen in paragraaf 3 gezien hebben. Het grote voordeel van de nieuwe beschrijving is dat we nu achter de komma kunnen werken. Als een quotiënt met 6 cijfers achter de komma gewenst is, vervangen we het criterium in regel (9) door $k > -6$. Uit de relaties op regel (16) en (17) volgt dat $deler \cdot a \leq deeltal$ en $deler \cdot (a + m) > deeltal$. Op regel (17) krijgen we natuurlijk $k = -6$ (in plaats van = 0) en dit impliceert dat de uitkomst op een waarde $m = 10^{-6}$ nauwkeurig is.

Machtsverheffen en logarithme berekenen

De code in figuur 1 is gemakkelijk te wijzigen in een algoritme voor machtsverheffen. Als we g^x willen uitrekenen met g (grondtal) en x (exponent) natuurlijke getallen, dan wordt $b^a \cdot c = g^x$ de nieuwe invariant. In de code moeten dan alleen de operaties optellen en vermenigvuldigen met operanden c en b in de regels (6) en (9) vervangen worden door respectievelijk vermenigvuldigen en machtsverheffen. Voor het geval $r = 2$ krijgen we een heel praktisch algoritme. Het machtsverheffen is dan slechts kwadrateren, dus feitelijk ook vermenigvuldigen.

Een probleem uit de praktijk is: hoeveel is g^x *mod* z met z een derde geheel getal. Als $z = 1000$, is het probleem equivalent aan: wat zijn de laatste drie cijfers van g^x? De rekenkundige operaties op b en c hoeven dan ook alleen maar *mod* 1000 uitgevoerd te worden, dat wil zeggen na elke vermenigvuldiging worden alleen de laatste drie cijfers genomen en de overige cijfers kunnen worden weggegooid. De berekening van g^x *mod* z is van grote betekenis voor de cryptografie, de geheimschriftkunde, in het bijzondervoor de RSA-methode [6, 7]. Tegenwoordig wordt deze methode veel gebruikt voor het crypteren van gegevens, onder andere bij Internettechnologieën als PGP en SSL, de laatste wordt onder andere bij internetbankieren gebruikt. De getallen g, x en z zijn dan grote getallen van 100 tot 200 cijfers.

Het algoritme in figuur 3 is gemakkelijk te veranderen in een algoritme voor logarithme-berekening. Stel je wilt ${}^g\log y$ berekenen. Vervang *deler* en *deeltal* door respectievelijk g en y in figuur 3. Verder moeten alleen de bewerkingen met c in de regels (7), (12) en (15) gewijzigd worden. Delen door r wordt de r-de machtswortel trekken, vermenigvuldigen wordt machtsverheffen en aftrekken wordt delen.
De gelijkheid $deler \cdot a + c \cdot m = deeltal$ in de invariant van figuur 3 gaat over in: $g^a \cdot c^m = y$. Deze manier van logarithmeberekening is eerder behandeld in [3] met $r = 10$. Voor $r = 2$ met $y < g$ (het gedeelte (4) tot en met (8) kan dan overgeslagen worden) is dit algoritme behandeld in [5].

Worteltrekken

Voor het trekken van de wortel uit een natuurlijk getal bestaat ook een soort 'staartdelings'algoritme. Dit algoritme zullen we hier presenteren aan de hand van een voorbeeld. Dit algoritme ben ik tegengekomen in een oude handleiding bij een mechanische rekenmachine (een 'koffiemolen'), zoals die in de zestiger jaren bij banken en wisselkantoren werd gebruikt. Het algoritme werd in iets andere vorm ook behandeld, meestal in kleine lettertjes, in de algebraboeken van vóór de Mammoetwet.
Veronderstel dat we de wortel uit het getal 522737 willen bepalen. We splitsen het getal in mootjes van twee cijfers: 52|27|37. Het algoritme bestaat uit drie fases, corresponderend met de drie mootjes, respectievelijk 52, 27 en 37. In de eerste fase wordt van 52 cumulatief 1, 3, 5, … afgetrokken. Dit gaat 7 keer (en we noteren dan ook het getal 7) dat wil zeggen we halen $1 + 3 + 5 \ldots + 13$ van 52 af. Het resultaat is gelijk aan 3. De waarde 15 lukt niet meer, want dat zou een negatief resultaat geven. In de oude algebra-boeken wordt deze fase als volgt beschreven: zoek het hoogste cijfer (digit) d in het bereik 1 tot en met 9 zodat $d^2 \leq 52$. Beide beschrijvingen zijn gelijkwaardig vanwege het feit dat de som van de eerste k oneven getallen gelijk is aan k^2.
Nu komt de tweede fase. De restwaarde in de eerste fase was 3. We gaan het volgende mootje, te weten 27, bijhalen en krijgen dan 327. De laatste kandidaat uit de rij oneven getallen die we probeerden was 15. We verlagen dit getal 15 met 1 en krijgen dan 14. (We zullen zo dadelijk zien dat het geen toeval is dat 14 ook gelijk is aan 2×7 met 7 het genoteerde cijfer). Vervolgens plaatsen we achter de cijfers 14 het cijfer 1, zodat je 141 krijgt. We gaan nu cumulatief van 327 de getallen 141, 143, 145, …, aftrekken. Dit gaat 2 keer met als restwaarde $327 - 141 - 143 = 43$.
We noteren het cijfer 2. Deze tweede fase wordt in de oude algebraboeken nogal afwijkend beschreven. Daar staat te lezen: zoek het hoogste cijfer d (digit) zodat $(70 + d)^2 \leq 5227$ ofwel $(4900 + 140d + d^2) \leq 5227$. (De waarde 70 is 10×7 met 7 het eerst genoteerde cijfer.) Omdat al 4900 van 5200 is afgetrokken, zoeken we naar de hoogste d met $(140 + d) \times d \leq 327$, ofwel de grootste waarde uit de rij 141×1, 142×2, 143×3, 144×4, … die ≤ 327 is. Deze beschrijving is gelijkwaardig aan onze eerdere omschrijving: tel hoeveel termen van de sommatie $141 + 143 + 145 + 147 + \ldots$ binnen de 327 blijven. Door toepassing van de somformule voor rekenkundige rijen is dit na te gaan.
In de derde fase wordt het volgende mootje (37) bijgehaald; we plaatsen dit bij het laatste resultaat van de aftrekkingen (43) en krijgen dan 4337. De laatste kandidaat bij de aftrekkingen (145) wordt met 1 verlaagd. Het is weer geen toeval dat $144 = 2 \times 72$ met 7 en 2 de genoteerde cijfers. We voorzien 144 van een extra cijfer 1 en krijgen dan 1441. We trekken 3 oneven getallen 1441, 1443 en 1445 van 4337 af en noteren een 3. De laatste uitkomst van de aftrekkingen is 8.

De conclusie is: $522737 = 723^2 + 8$.

```
(1)   r = 2, of r = 10;
(2)   c = getal;
(3)   k = 0; a = 0; af = 1; m = 1;                    {invariant}
(4)   while c > m do
(5)       [ k = k + 1;
(6)         m = m × r^2;
(7)       ]                                  {invariant ∧ k ≥ 0 ∧ c < m·af}
(8)   while k > 0 do
(9)       [ k = k – 1;
(10)        m = m / r^2;
(11)        a = a × r;
(12)        af = r × (af – 1) + 1                {invariant ∧ k ≥ 0}
(13)        while c ≥ m × af do
(14)          [ c = c – m × af;
(15)            a = a + 1;
(16)            af = af + 2
(17)          ]                                  {invariant ∧ k ≥ 0 ∧ c < m·af}
(18)      ]                                      {invariant ∧ k = 0 ∧ c < m·af}
```

fig. 4 Een algoritme voor worteltrekken

Het gehele proces wordt in figuur 4 beschreven. De beschrijving blijft geheel binnen de natuurlijke getallen (geen cijfers achter de komma) en past in dat opzicht bij figuur 2.
Als je achter de komma wilt werken, moet je een transformatie richting figuur 3 toepassen. Merk op dat we in de inner loop van figuur 2 steeds de bewerking $c - b = c - m \cdot deler$ deden; hier doen we $c - m \cdot af$. De invariant is:

$$m \cdot a^2 + c = getal \wedge af = 2a + 1 \wedge m = r^{2k} \wedge c \geq 0$$

In de beschrijving van het algoritme is ons al opgevallen dat aan het begin van elke nieuwe fase $af = 2a + 1$ geldt. Om te bewijzen dat de invariant door de regels (14) en (15) intact blijft, hebben we de volgende herleiding:
$m \cdot a^2 + c = m \cdot a^2 + m \cdot af + c - m \cdot af =$
$m \cdot a^2 + m \cdot (2a + 1) + c - m \cdot af =$
$m \cdot (a + 1)^2 + (c - m \cdot af)$
Dit laat zien dat $m \cdot a^2 + c$ ongewijzigd blijft, als a met 1 verhoogd en c met $m \cdot af$ verlaagd wordt.
Aan het einde van de innerloop hebben we $0 \leq c < m \cdot af$. De volgende relaties gelden dan:
$m \cdot a^2 \leq getal$ en $m \cdot (a + 1)^2 = m \cdot a^2 + m \cdot (2a + 1) =$
$m \cdot a^2 + m \cdot af > m \cdot a^2 + c = getal$.
Omdat aan het einde $m = r^k = r^0 = 1$ geldt, heeft de geheeltallige variabele a na afloop een waarde met $a^2 \leq getal$ en $(a + 1)^2 > getal$.

Zoals we dat ook bij het delingsalgoritme deden, kunnen we ook nu weer aantonen dat de innerloop in (13) tot en met (17) minder dan r keer doorlopen wordt. Na regel (7) of (17) hebben we $0 \leq c < m \cdot af$. Na regel (10) hebben we dan $0 \leq c < r^2 \cdot m \cdot af = m \cdot r^2 \cdot (af - 1) + m \cdot r^2$ en na regel (12) wordt dit: $0 \leq c < m \cdot r \cdot (af - 1) + m \cdot r^2$. Dit laatste rechterlid is gelijk aan de som van r termen uit de reeks $m \times af$, $m \times (af + 2)$, m x$(af + 4)$, …, zo is met de formule voor rekenkundige rijen na te gaan. De loop wordt derhalve minder dan r keer uitgevoerd.

Slotopmerkingen

De hier getoonde algoritmen hebben praktische waarde als men multiprecisie bewerkingen met de computer wil uitvoeren. Multiprecisie bewerkingen zijn bewerkingen met getallen die niet meer in het integer-bereik van de computer passen, zodat men de cijfers afzonderlijk of per twee of drie in een array-cel moet plaatsen. Om de hier behandelde algoritmen te kunnen uitvoeren, moet men eerst beschikken over programma's voor multiprecisie-optellen en aftrekken, programma's die zonder veel moeite te schrijven zijn. Zijn deze programma's eenmaal beschikbaar, dan zijn de andere algoritmen gemakkelijk te implementeren. Merk op dat het hele spectrum van 'Meneer van Dalen wacht op antwoord' is gepasseerd. Zoals eerder opgemerkt is de formele theorie rondom algoritmen pas vorige eeuw, met de opkomst van de informatica, ontwikkeld. De hier gebruikte bewijsmethodiek is eind jaren zestig en begin jaren zeventig ontwikkeld door onder andere de Nederlandse informaticus Edsger Dijkstra. Een uitvoeriger behandeling is te vinden in [2]. Wanneer in de toekomst de staartdeling niet meer bekend zal zijn bij de studenten zal ik hem weer introduceren, omdat het bewijzen van algoritmen er goed mee te illustreren is. Al hebben de rekenkundige technieken zoals staartdelen geen functie meer in de dagelijkse rekenpraktijk en kunnen ze in die zin gerust geschrapt worden uit het basisonderwijs, het zijn fraaie wiskundige concepten, waarin wiskundige activiteiten zoals bewijzen, generaliseren, specialiseren enzovoort een duidelijke rol spelen. Zoals eerder opgemerkt zijn ze te vergelijken met meetkundige constructies. Deze waren onderdeel van het lesprogramma vanwege hun wiskundige karakter, niet vanwege te verwachten practische toepassingen. Het feit dat de grafische rekenmachine kan differentiëren of vergelijkingen kan oplossen, is ook geen reden de wiskundige theorie achter deze onderwerpen te schrappen uit het lesprogramma. 'Requiem voor de staartdeling', leve de algoritmen!

Wim Pijls, Erasmus Universiteit, Rotterdam

Noot

[1] Bunt, L.N.H. (1968). *Van Ahmes tot Euclides.* Wolters Noordhoff.
[2] Dijkstra, E.W. & W. Feijen (1984). *Een methode van programmeren.* Academic Service.
[3] Pot, H. Logaritmen en het rekendoosje. *Euclides 55*(3), p. 99-102.
[4] Hogendijk, J.P. (1997). Algorismi's rekenboekje compleet. *Nieuwe Wiskrant 17*(1), p. 30-31, Zevenbergen
[5] Sluis, A. van der (1969). *Computers en algoritmen.* Tousreeks nr. 5. Wolters Noordhoff.
[6] Singh, S. (1999). *Code* (vertaling van The Code Book). De Arbeiderspers.
[7] Tel, G. (2002). *Beveiliging van de digitale maatschappij.* Addison Wesley.