TECHNISCHE HOGESCHOOL EINDHOVEN
Afdeling Algemene Wetenschappen
Onderafdeling der Wiskunde

41 Oefeningen bij het college

Inleiding in de

KUNST van PROGRAMMEREN

samengesteld door

Ir. W.H.J. Feijen

Najaarssemester 1971

Oefeningen bij het college Inleiding in de Kunst van Programmeren.

Opgave 0.

Gevraagd wordt een algorithme te ontwerpen die van een gegeven integer array A[1:N], N≥1 bepaalt of de rij (A[1],...,A[N]) monotoon stijgend, dan wel monotoon dalend, dan wel geen van beide is.

Laten we beginnen met te formuleren wanneer een rij monotoon stijgt, en wanneer een rij monotoon daalt.
Een rij $(a_1,\ldots,a_n)$ van gehele getallen heet monotoon stijgend indien voor elke i met $1\leq i<n$ geldt dat $a_i<a_{i+1}$.
Een rij $(a_1,\ldots,a_n)$ van gehele getallen heet monotoon dalend indien voor elke i met $1\leq i<n$ geldt dat $a_i>a_{i+1}$.
We merken op dat een rij bestaande uit slechts één element (n=1) zowel monotoon stijgend als monotoon dalend is.

(eigenschap 1): de rij (a_1) is monotoon stijgend èn monotoon dalend.

Bovendien is heel eenvoudig te verifieren dat een rij $(a_1,\ldots,a_k)$ voor $k\geq 2$ alleen dan monotoon stijgend is indien de beginrij $(a_1,\ldots,a_{k-1})$ monotoon stijgt en bovendien voldaan is aan $a_{k-1}<a_k$. Een soortgelijke bewering geldt voor een monotoon dalende rij.
Deze uitspraken leiden derhalve tot de volgende twee eigenschappen.

(eigenschap 2a): voor $k\geq 2$ geldt dat $(a_1,\ldots,a_k)$ d.e.s.d. monotoon stijgend is indien $(a_1,\ldots,a_{k-1})$ monotoon stijgend is èn $a_{k-1}<a_k$.

(eigenschap 2b): voor $k\geq 2$ geldt dat $(a_1,\ldots,a_k)$ d.e.s.d. monotoon dalend is indien $(a_1,\ldots,a_{k-1})$ monotoon dalend is èn $a_{k-1}>a_k$.

Met behulp van deze drie eigenschappen is het nu al niet meer zo moeilijk een algorithme te poneren. En deze volgt nu.

```
begin integer i; boolean monincr,mondecr;
      i:=1; monincr:=true; mondecr:=true;
      while i<N do
        begin i:=i+1; monincr:=monincr and A[i-1]<A[i];
                      mondecr:=mondecr and A[i-1]>A[i]
        end;
      if monincr do printtext(M);
      if mondecr do printtext(D);
      if non (monincr or mondecr) do printtext(NONE)
end
```

Dit programma bedient zich van een integer gedeclareerde variabele i, die achtereenvolgens de waarden 1,...,N zal aannemen, zijnde de indices van de opeenvolgende elementen in het gegeven integer array A[1:N]. Initieel krijgt i de waarde 1. Bovendien zijn er in het programma twee boolean gedeclareerde variabelen, te weten monincr en mondecr, geïntroduceerd, en verder zien we een repetitiecycle. Na uitvoering van iedere slag, ook na uitvoering van de nulde, heeft monincr de waarde van het predicaat "de rij (A[1],...,A[i]) is monotoon stijgend". Op grond van eigenschap 1 moet monincr na uitvoering van de nulde slag dan de waarde true hebben. In de te repeteren statement zelf wordt de nieuwe waarde van monincr verkregen uit de oude, en uit de relatie A[i-1]<A[i]. De wijze waarop dit geschiedt is duidelijk geïnspireerd door eigenschap 2a. Een analoge beschouwing geldt voor de variabele mondecr. De cycle eindigt zodra de waarde van i≥N geworden is. Dat beeindiging inderdaad optreedt volgt uit het feit dat i in het begin de waarde 1 krijgt en daarna per cyclus telkens met 1 verhoogd wordt. Op grond van de betekenis van monincr en mondecr kunnen we dan na beeindiging van de repetitieve statement de passende boodschap, bijvoorbeeld uitprinten.

We zullen over deze algorithme nu niet meer verder uitweiden, en in het bijzonder geen correctheidsbewijs geven, aangezien er bezwarende kanttekeningen bij te maken zijn. Uit de te repeteren statement zien we dat, indien de variabele monincr één keer de waarde false heeft aangenomen, dat hij deze vanaf dat ogenblik zal blijven behouden. En evenzo voor de variabele mondecr.
Deze constateringen zijn consistent met wat uit eigenschap 2a (resp. eigenschap 2b) is af te leiden, nl.
als de rij $(a_1,\ldots,a_k)$ niet monotoon stijgend is, dan is ook de uitgebreide rij $(a_1,\ldots,a_{k+1})$ niet monotoon stijgend (en analoog voor monotoon dalend).
M.a.w., zodra we een beginrij hebben waarvan vaststaat dat deze niet monotoon stijgt, dan kan het in beschouwing nemen van een nieuw element de monotone stijging voor de uitgebreide rij niet alsnog bewerkstelligen.
Deze overwegingen geven aanleiding tot de volgende eigenschappen:

(eigenschap 3a): als de rij $(a_1,\ldots,a_k)$ niet monotoon stijgend is voor enige k met $2\leq k\leq n$, dan is ook de rij $(a_1,\ldots,a_n)$ niet monotoon stijgend.

(eigenschap 3b): als de rij $(a_1,\ldots,a_k)$ niet monotoon dalend is voor enige k met $2\leq k\leq n$, dan is ook de rij $(a_1,\ldots,a_n)$niet monotoon dalend.

Aangezien de gewenste algorithme moet vaststellen of de gegeven rij (A[1],...,A[N]) monotoon stijgend, monotoon dalend dan wel geen van beide is, heeft het geen zin meer nog een nieuw element A[i+1]in de beschouwing te betrekken, zodra van een subrij (A[1],...,A[i]) met i<N vaststaat dat deze niet meer monotoon stijgt of daalt.
Na deze opmerkingen is het volgende programma dan ook niet meer erg verrassend.

```
begin integer i; boolean monincr,mondecr;
      i:=1; monincr:=true; mondecr:=true;
      while i<N and (monincr or mondecr) do
        begin i:=i+1;
              if monincr do monincr :=(A[i-1]<A[i]);
              if mondecr do mondecr :=(A[i-1]>A[i])
        end;
      if non (monincr or mondecr) then printtext(NONE)
                                  else
        begin if monincr do printtext(M); if mondecr do printtext(D) end
end
```

Ook in dit programma heeft de boolean variabele monincr na elke uitvoering van de te herhalen statement de betekenis "de rij (A[1],...,A[i]) is monotoon stijgend". En analoog heeft mondecr de betekenis "de rij (A[1],...,A[i]) is monotoon dalend". De eigenschappen 3a en 3b leren dan dat het alleen nog maar zinvol is (wat het resultaat betreft) het volgende element A[i+1] in de bepaling van het gewenste resultaat te betrekken indien monincr or mondecr de waarde true heeft.

Vervolgens zullen we een min of meer formeel bewijs trachten te geven van de correctheid van de bovenstaande algorithme. Hierbij zal blijken dat het verloop van dit bewijs belangrijk bepaald wordt door de analyse die we vooraf aan het probleem verricht hebben, en dus door de manier waarop wij de algorithme hebben ontworpen en begrepen.

Bij het opzetten van de algorithme hebben we erop gemikt dat na afloop van elke te repeteren statement zal gelden dat de variabele monincr d.e.s.d de waarde true zal hebben als de subrij (A[1],...,A[i]) monotoon stijgend is. Evenzo zal op die momenten de variabele mondecr d.e.s.d. de waarde true hebben als de rij (A[1],...,A[i]) monotoon dalend is. Dit zijn derhalve twee beweringen over de programmavariabelen monincr en mondecr die ongevoelig zullen zijn voor het aantal malen dat de te repeteren statement is uitgevoerd. De standaard techniek die gebruikt wordt om dit aan te geven is het in de programmatekst

inpluggen van een invariante bewering of relatie aan het begin van de cycle. Hier wordt de invariante relatie beschreven door het predicaat Q(i),

Q(i): $1 \le i \le N$ and $P_{incr}(i)$ and $P_{decr}(i)$

waarin $P_{incr}(i)$ resp. $P_{decr}(i)$ staan voor de predicaten

$P_{incr}(i)$: monincr eq (A[1],...,A[i]) monotoon stijgend,

$P_{decr}(i)$: mondecr eq (A[1],...,A[i]) monotoon dalend.

Het bewijs dat deze relatie steeds geldt na uitvoering van de te repeteren statement geven we met inductie naar het aantal malen k dat de te herhalen statement is uitgevoerd.

k=0: Het netto-effect van de initialiseringsstatements
" i:=1; monincr:=true; mondecr:=true "
is dat vlak na afloop ervan geldt:
i=1 and monincr and mondecr.
Met behulp van eigenschap 1 is het dan triviaal dat de invariante relatie Q(i) geldig is vóór de eerste uitvoering van de te repeteren statement, dus ná de nulde uitvoering ervan.

k>0: De veronderstelling is nu dat de invariante relatie al geldt na de $k-1^{ste}$ uitvoering van de te herhalen statement.
Vlak voor de uitvoering van de eerste statement in de k^{de} slag geldt dus
Q(i) and (i<N and (monincr or mondecr)),
hetgeen te reduceren is tot
$1 \le i < N$ and $P_{incr}(i)$ and $P_{decr}(i)$ and (monincr or mondecr).
Vlak na uitvoering van de statement "i:=i+1" geldt:
$1 < i \le N$ and $P_{incr}(i-1)$ and $P_{decr}(i-1)$ and (monincr or mondecr).
Het netto-effect van de conditionele statement
"if monincr do monincr:=A[i-1]<A[i]"
is onder deze omstandigheden zo dat na afloop ervan geldt:
$1 < i \le N$ and $P_{incr}(i)$ and $P_{decr}(i-1)$.
Dit is als volgt in te zien.

- Stel dat monincr geldt. Dit betekent op grond van de geldigheid van $P_{incr}(i-1)$ dat (A[1],....A[i-1]) monotoon stijgend is. Nu wordt de statement "monincr:=A[i-1]<A[i]" uitgevoerd, hetgeen inhoudt dat daarna geldt:
(A[1],...,A[i-1]) monotoon stijgend and(monincr eq A[i-1]<A[i]).

Uit eigenschap 2a volgt dan dat
monincr eq (A[1],...,A[i]) monotoon stijgend, ofwel dat $P_{incr}(i)$ weer geldt.

- Stel dat monincr niet geldt. Dit betekent op grond van de geldigheid van $P_{incr}(i-1)$ dat (A[1],...,A[i-1]) niet monotoon stijgend is. En met eigenschap 3a is hieruit af te leiden dat (A[1],...,A[i]) evenmin monotoon stijgt. Anderzijds wordt de statement "monincr:=A[i-1]<A[i] " niet uitgevoerd wegens de ongeldigheid van monincr. Derhalve blijft monincr ongeldig, en dus de relatie monincr eq (A[1],..., A[i]) monotoon stijgend, ofwel het predicaat $P_{incr}(i)$, geldig.
- Omdat de variabelen i en mondecr in deze conditionele statement niet gewijzigd worden geldt na afloop ervan onverminderd $1<i\leq N$ and $P_{decr}(i-1)$.

Het netto-effect van de conditionele statement
"if mondecr do mondecr:=A[i-1]>A[i]"
is nu zodanig dat onmiddellijk na afloop ervan geldt
$1<i\leq N$ and $P_{incr}(i)$ and $P_{decr}(i)$,
hetgeen op analoge wijze is aan te tonen.
Dit was de laatste statement van de k^{de} slag, en inderdaad geldt weer:
Q(i): $1\leq i\leq N$ and $P_{incr}(i)$ and $P_{decr}(i)$, q.e.d..

M.a.w., de uitspraak Q(i) geldt na elke uitvoering van de te herhalen statement, en wel in het bijzonder na de laatste (als die er is).
Echter na uitvoering van de laatste cyclus geldt eveneens
non(i<N and (monincr or mondecr)).
Ofwel in toto moet dan voldaan zijn aan

$(1\leq i\leq N$ and $P_{incr}(i)$ and $P_{decr}(i))$ and $(i\geq N$ or non(monincr or mondecr))

Stel nu dat monincr geldt na afloop van de repetitiestatement. Dan volgt daaruit dat i=N moet gelden , en dus $P_{incr}(N)$ eveneens, hetgeen betekent dat (A[1],...,A[N]) monotoon stijgend is.
Evenzo volgt uit de geldigheid van mondecr dat (A[1],...,A[N]) monotoon dalend is.
Indien noch monincr noch mondecr geldt, dus indien non(monincr or mondecr) waar is, dan volgt uit de juistheid van $P_{incr}(i)$ en die van $P_{decr}(i)$, en uit de eigenschappen 3a en 3b dat de rij (A[1],...,A[N]) noch monotoon stijgt, noch monotoon daalt.

Als we nu nog aantonen dat de in de algorithme voorkomende repetitiestatement eindigt, dan is daarmee de correctheid van de algorithme bewezen.
Geef het eindigheidsbewijs nu zelf!

Opmerking: Bovenstaand bewijs is niet erg formeel als we bedenken dat we de primitiva uit de taal waarvan we ons bedienen, zoals de assignment statement, de conditionele statement, de alternatieve statement, de repetitiestatement en het begrip variabele, niet geformaliseerd hebben. We hebben de werking van deze concepten intuitief begrepen en op dit begrip is het bewijs gebaseerd. Prof. C.A.R. Hoare heeft deze fundamentele concepten wel geaxiomatiseerd, en hij heeft bovendien aangegeven hoe met behulp van deze axioma's een correctheidsbewijs moet worden gevoerd. In een later stadium komen we hier nog op terug.
Gelukkig is ons intuitieve begrip omtrent de primitiva uit de programmeertaal niet in strijd met de axioma's, en is onze bewijsvoering consistent met de formeel voorgeschrevene, en in het licht daarvan gezien mogen we het gegeven bewijs wel aanvaardbaar achten.

Opgave 1.
Een rij $(a_1,\ldots,a_n)$ van gehele getallen heeft de eigenschap K (Knik) indien er een natuurlijk getal m bestaat waarvoor geldt dat $1<m<n$, én dat hetzij de rij $(a_1,\ldots,a_m)$ monotoon dalend en $(a_m,\ldots,a_n)$ monotoon stijgend is, hetzij de rij $(a_1,\ldots,a_m)$ monotoon stijgend en $(a_m,\ldots,a_n)$ monotoon dalend is.
Als nu gegeven is een integer array A[1:N], $N\geq 1$, maak dan een boolean operator die de waarde true aflevert indien de rij (A[1],...,A[N]) de eigenschap K bezit, en de waarde false in alle andere gevallen.

Opgave 2.
Op een band staat een rij $(a_1,\ldots,a_n)$ van gehele getallen, met n bekend en gegeven. Voor $i=1,\ldots,n$ geldt dat $0\leq a_i\leq 9$. Laat $k_1,\ldots,k_m$ $(k_1<\ldots<k_m)$ de indices voorstellen van de m elementen in de rij die gelijk zijn aan 9 (negen), $m\geq 0$.
Met behulp van deze m indices definieren we de volgende m disjuncte subrijtjes:
$r_1=(a_1,\ldots,a_{k_1})$, $r_i=(a_{k_{i-1}+1},\ldots,a_{k_i})$ voor $i=2,\ldots,m$.
Geef nu een algorithme die de rijtjes $r_1,\ldots,r_m$ in deze volgorde afdrukt, en wel rijtje r_i van 'links naar rechts' indien i oneven is, en van 'rechts naar

links' indien i even is.
(N.B. Het netto-effect van de opdracht "x:=read" is dat de waarde van het huidige getal op de band wordt toegekend aan de variabele x, en dat het volgende getal op de band het huidige wordt. Vóórdat de eerste leesactie is geïnitialiseerd is het eerste getal op de band het huidige.)

Opgave 3.

Laat gegeven zijn een integer array p[1:N], $N \geq 1$, $p[i] \geq 1$ voor $1 \leq i \leq N$. Op een band staat een rij natuurlijke getallen $(a_1,\ldots,a_k)$ die afgesloten is door een 0 (nul). Gevraagd wordt een algorithme te ontwerpen die het aantal consecutieve subrijen $(a_{i+1},\ldots,a_{i+N})$ bepaalt waarvoor $a_{i+j}=p[j]$, $1 \leq j \leq N$.

Opgave 4.

Laat gegeven zijn een integer array A[1:N], $N \geq 1$, en een natuurlijk getal p, $1 \leq p \leq N$. Zij $(a_1,\ldots,a_N)$ de rij gehele getallen die bepaald is door $a_i=A[i]$ voor $i=1,\ldots,N$. Maak nu een algorithme die het array A zodanig transformeert dat na afloop geldt:
$A[i]=a_{i+p}$ voor $i=1,\ldots,N-p$, en $A[i]=a_{i+p-N}$ voor $i=N-p+1,\ldots,N$.
De restrictie is dat het aantal grootheden (variabelen) waarvan de algorithme zich mag bedienen onafhankelijk van N en van p moet zijn.

Opgave 5.

Gegeven is een natuurlijk getal N. Maak een algorithme die alle rijen opeenvolgende natuurlijke getallen bepaalt waarvan de som gelijk is aan N.

Opgave 6.

Gegeven is een natuurlijk getal a, in decimale voorstelling te interpreteren (eenduidigheidsafspraak: meest significante cijfer ongelijk aan nul). Construeer een operator die aan de integer gedeclareerde variabele b de waarde toekent van het grootste natuurlijke getal dat met de cijfers van a kan worden opgebouwd.

Opgave 7.
Gegeven zijn twee natuurlijke getallen a en b, $a \leq b$. Maak een algorithme die alle natuurlijke paren (n,n^3) genereert waarvoor geldt $a \leq n^3 \leq b$. De beperking daarbij is dat de optelling en aftrekking de enige toegelaten arithmetische operaties zijn.

Opgave 8.
Gegeven zijn twee natuurlijke getallen A en k, $k<A$. Gevraagd wordt een algorithme te schrijven die alle tripels (a,b,c) natuurlijke getallen genereert die voldoen aan $a \leq A$, $a-b=k$, $a^2-b^2=c^2$. De restrictie is dat de enige toegestane rekenkundige bewerkingen de optelling en de aftrekking zijn.

De manier waarop we een vraagstuk als dit plegen op te lossen beantwoordt aan een min of meer standaardpatroon, en we zullen nu proberen dit patroon door middel van dit vraagstuk te illustreren.
Vooraleer we ons verdiepen in enige algorithme kunnen we toch al een opmerking maken die van bijzondere betekenis blijkt te zijn voor de te ontwerpen algorithme. Aangezien we zoeken naar een sequentiële algorithme is het onvermijdelijk om in een of ander stadium van het ontwerp vast te leggen in welke volgorde de algorithme de gezochte tripels zal moeten voortbrengen. Anderzijds zal blijken dat het hebben van zo'n ordening in de verzameling der gezochte tripels ons bijzonder behulpzaam kan zijn bij het vinden van een geschikte algorithme, mits zo'n ordening geschikt gekozen is. Veelal geldt zelfs dat de ordening die we introduceren bepalend is voor het gemak waarmee we de uiteindelijke algorithme vinden. Immers als we een tripel (a,b,c) hebben dat een oplossing van het probleem is, dan hangt het van de geïntroduceerde ordening af hoe eenvoudig we uit deze oplossing de volgende kunnen destilleren.
In algemenere zin geformuleerd kunnen we stellen, dat in gevallen waarin gevraagd wordt een collectie objecten te genereren, het vaak zinvol is een ordening in de gezochte collectie te definieren, en in het licht van deze ordening trachten te begrijpen welke relatie er tussen twee opeenvolgende objecten bestaat. Deze relatie leidt in vele gevallen alras naar een algorithme. Hoe aanvaardbaar en begrijpbaar deze algorithme dan is hangt nauw samen met de geïntroduceerde ordening.
In dit vraagstuk hebben we niet zo'n grote keuze tussen alle mogelijke ordeningen die vanuit algorithmisch standpunt redelijk hanteerbaar en attractief

zijn. We constateren dat voor ieder tripel (a,b,c) dat een oplossing is geldt dat $k+1 \leq a \leq A$, en dat voor elke a met $k+1 \leq a \leq A$ er hoogstens één tripel (a,b,c) bestaat dat oplossing is. Dit betekent dat we twee oplossingen (a,b,c) en (a',b',c') kunnen onderscheiden via de waarden van a en a'. Een voor de hand liggende en ogenschijnlijk hanteerbare ordening van de oplossingsverzameling is dan ook:

$(a',b',c') \geq_t (a,b,c)$ d.e.s.d. als $a'>a$.

Laten we nu eens uitgaan van een oplossing (a,b,c). D.w.z. dat $a \leq A$, $a-b=k$, $a^2-b^2=c^2$. Laat verder (a',b',c') een op (a,b,c) volgende oplossing zijn. Dan staat in ieder geval vast dat $a' \leq A$, $a'-b'=k$, $(a')^2-(b')^2=(c')^2$, $a'>a$.
Uit $a'>a$ volgt de existentie van een natuurlijk getal i zodat $a'=a+i$.
Uit $a' \leq A$ volgt dan dat $i \leq A-a$ is.
Wegens $a'-b'=k$ en $a-b=k$ moet verder $b'=b+i$ gelden.
En tenslotte volgt uit $(a')^2-(b')^2=(c')^2$ en $a^2-b^2=c^2$ dat $(c')^2=c^2+2ik$.
Deze korte analyse voert nu onmiddellijk tot

stelling 1: Als (a,b,c) een oplossing van het gestelde probleem is, en I is de verzameling van alle natuurlijke getallen $i \leq A-a$ waarvoor c^2+2ik een kwadraat is, dan is de verzameling van alle oplossingen $(a',b',c') \geq_t (a,b,c)$ gelijk aan de verzameling van alle tripels $(a+i,b+i,(c^2+2ik)^{\frac{1}{2}})$ met $i \in I$.

We zien hieruit dat we uitgaande van de existentie van een oplossing op vrij eenvoudige manier een verband hebben gelegd met alle volgende oplossingen. Dit verband, dat beschreven wordt door het kwadraat-zijn van de uitdrukking c^2+2ik, vormt de grondslag voor de te ontwerpen algorithme. Voordat we kunnen overgaan tot het beschrijven van deze algorithme moeten we ons evenwel nog bekommeren om één oplossing, waarvan de existentie in deze hele beschouwing vooronderstelt is. Vooralsnog is er echter geen enkele aanwijzing die erop duidt dat er een oplossing bestaat, en bij bepaalde keuze van A en k zal er zelfs geen oplossing zijn. Een veel gebruikte methode in deze gevallen is de introductie van een kunstmatige, virtuele oplossing, die er toe dient de algorithme van de grond te helpen. Gezien de volgorde waarin we van plan zijn de gevraagde tripels te genereren, moet de virtuele oplossing uiteraard wel zó zijn dat deze in dit volgorde-schema past, m.a.w. onmiddellijk vooraf moet gaan aan de eventuele eerste echte oplossing. Hier is die virtuele oplossing erg duidelijk, nl.

Het tripel (k,0,k) vormt een virtuele eerste oplossing.

Dit gecombineerd met stelling 1 kenmerkt nu precies alle oplossingen die we zoeken:

stelling 2: Als I de verzameling van alle natuurlijke getallen $i \leq A-k$ is waarvoor geldt dat k^2+2ik een kwadraat is, dan is de verzameling der gevraagde tripels gelijk aan de verzameling van alle tripels $(k+i,i,(k^2+2ik)^{\frac{1}{2}})$ waarvoor $i \in I$.

Het vinden van een algorithme voor het oorspronkelijk probleem is getransformeerd in het vinden van een algorithme die kwadraten van de vorm k^2+2ik, $1 \leq i \leq A-k$ zoekt.

Beschouw daarom de volgende drie rijen.

$\{c_i\}$ met $c_i=k+i$ voor $i \geq 0$,
$\{u_j\}$ met $u_j=k^2+2jk$ voor $j \geq 0$,
$\{v_m\}$ met $v_m=(k+m)^2$ voor $m \geq 0$.

Het verband tussen de rijen $\{c_i\}$ en $\{v_m\}$ is zodanig dat voor elke $m \geq 0$ geldt dat $v_m=c_m^2$.
Laat nu v_m een gegeven kwadraat uit de rij $\{v_m\}$ zijn, en u_j het kleinste element uit de rij $\{u_j\}$ dat niet meer kleiner is dan v_m, dan kunnen we op grond van stelling 2 een oplossing hieruit afleiden indien $v_m=u_j$ en indien tevens $j \leq A-k$ is. Dit geeft aanleiding tot de volgende eigenschap die bruikbaar zal zijn voor de opbouw van de algorithme.

(eigenschap 1): Als $m \geq 1$ is en $u_{j-1} < v_m \leq u_j$, dan is het tripel $(k+j,j,c_m)$ een oplossing van het oorspronkelijke probleem indien $u_j=v_m$ en $j \leq A-k$.

Voor de elementen uit de drie rijen kunnen we nu de volgende recurrentieschema's opstellen.

$c_0=k$, $c_i=c_{i-1}+1$ voor $i \geq 1$,
$u_0=k^2$, $u_j=u_{j-1}+2k$ voor $j \geq 1$,
$v_0=k^2$, $v_m=v_{m-1}+2(k+m-1)+1=v_{m-1}+2c_{m-1}+1$ voor $m \geq 1$.

Op grond van eigenschap 1 moeten we voor elk kwadraat v_m de kleinste u_j uit de rij $\{u_j\}$ bepalen waarvoor $v_m \leq u_j$. Met behulp van de recurrente betrekkingen is nu eenvoudig in te zien dat voor deze u_j bovendien geldt dat $u_j < v_{m+1}$. Met andere woorden, we moeten vanaf deze u_j tenminste één stap in de rij $\{u_j\}$ doen voordat we het kleinste element dat niet meer kleiner dan v_{m+1} is gevonden hebben.

(eigenschap 2): Als $m \geq 1$ is en $u_{j-1} < v_m \leq u_j$, dan is $u_j < v_{m+1}$.

Nu kunnen we de algorithme eenvoudigweg opschrijven.

```
begin integer j,c,u,v;
      j:=0; c:=k; ASSIGN TO BOTH u AND v THE VALUE k^2;
      while j<A-k do
        begin v:=v+c+c+1; c:=c+1;
              repeat j:=j+1; u:=u+k+k until u≥v or j=A-k;
              if u=v do printtriple(k+j,j,c)
        end
end
```

Gezien de voorgaande analyse is de structuur van de algorithme en de erin voorkomende representatie van de in het spel zijnde grootheden niet erg verwonderend. De variabele j doorloopt de waarden 0,1,...,A-k en fungeert als index in de rij $\{u_j\}$. De variabelen c,u en v corresponderen met de grootheden c_i, u_j en v_m uit de mathematische formulering.
In de initialisering van de grootheden j,c,u en v komt de operator "ASSIGN TO BOTH u AND v THE VALUE k^2" voor. Aangezien de algorithme bedoeld is voor een machine die als arithmetische operaties uitsluitend optelling en aftrekking toelaat, en aangezien we er ons op dit niveau van beschrijving nog niet over wensen uit te laten hoe in termen van optelling en aftrekking de waarde van k^2 wordt berekend en toegekend aan de variabelen u en v, drukken we deze activiteit tijdelijk uit in termen van een abstractere operator, die niet behoort tot het primitieve opdrachtenrepertoire van de betrokken machine.
Wel moeten we - willen we een algorithme uitdrukken in termen van abstractere operatoren - formuleren wat het netto-effect is dat zo'n operator heeft op de variabelen van de algorithme.
Voor de operator "ASSIGN TO BOTH u AND v THE VALUE OF k^2" postuleren we hier het netto-effect:

. Uitvoering van de operator geeft aanleiding tot een eindig proces.
. Na uitvoering van de operator geldt $u=k^2$ en $v=k^2$.
. Alle andere variabelen behouden hun aanvankelijke waarde en identiteit.

In het geval dat de algorithme bedoeld is voor een machine die wel kan vermenigvuldigen, kan de detaillering van deze operator luiden:

"u:=k×k; v:=u".

In ons geval waarin uitsluitend optelling en aftrekking zijn toegestaan, is een mogelijke beschrijving:

```
"begin integer i,ksq;
      i:=0; ksq:=0;
      while i<k do begin i:=i+1; ksq:=ksq+k end;
      u:=ksq; v:=ksq
 end"
```

Het bewijs dat de gegeven algorithme correct is, d.w.z. het bewijs dat alle oplossingen van het oorspronkelijke probleem geprint worden, zullen we hier niet tot in detail weergeven. Doch wel vermelden we dat de manier waarop het bewijs geleverd moet worden sterk gebaseerd is op de structuur van de algorithme enerzijds en derhalve op de voorafgaande analyse anderzijds, en wat betreft deze analyse in het bijzonder op stelling 2, die álle oplossingen karakteriseert.

Zo zal een gedeelte van de invariante relatie die moet gelden na elke uitvoering van de buitenste repeatable statement zijn:

voor alle i met $1 \leq i \leq j$ geldt dat het tripel $(k+i, i, (k^2+2ik)^{\frac{1}{2}})$ geprint is d.e.s.d. indien k^2+2ik een kwadraat is,

of iets formeler:

$$\forall_{1 \leq i \leq j}(\text{ tripel } (k+i,i,(k^2+2ik)^{\frac{1}{2}}) \text{ geprint } \underline{\text{eq}}\ \exists_{c \in Nt}(k^2+2ik=c^2)\ )$$

zodat na afloop van de laatste uitvoering van de repeatable statement, wanneer j=A-k, precies de voorwaarde geldt waarmee stelling 2 alle oplossingen garandeert.

De invariantie van deze relatie wordt nu bewezen m.b.v. invariante relaties die voor de variabelen j,c,u en v gelden, eveneens weer na uitvoering van de repeatable statement, zoals

$0 \leq j \leq A-k$ and $u=u_j$ and $v=c^2$ and $(0<j<A-k$ impl $u_{j-1}<v \leq u_j)$

die op hun beurt weer bewezen kunnen worden door gebruik te maken van de eigenschappen 1 en 2.

We zullen na deze summiere opmerkingen nu niet meer verder ingaan op het bewijs.

Wel past het nog enkele kanttekeningen te plaatsen bij de kwaliteit van de gegeven algorithme.

Er zijn algorithmen die het probleem veel inefficienter oplossen doch daarentegen veel overzichtelijker zijn, makkelijker uit te vinden, en waarvan het onmiddellijk vaststaat dat ze correct zijn, en dit voornamelijk wegens het feit dat de analyse die eraan voorafgaat bijna nihil is.

Zo kunnen we ons bijvoorbeeld voorstellen dat we constateren dat elke oplossing (a,b,c) moet voldoen aan $k+1 \leq a \leq A$, $1 \leq b \leq A-k$ en $k+1 \leq c \leq A$, en dat we een algorithme construeren die alle tripels (a,b,c) genereert die in dit gebied liggen, en dat deze algorithme voor elk zulk tripel nagaat of de relaties $a-b=k$ en $a^2-b^2=c^2$ gelden. Van een algorithme die zó geconstrueerd is, staat de correctheid spoedig vast, en de analyse die eraan voorafgaat is verwaarloosbaar.
Wellicht zijn er ook algorithmen die het gegeven probleem veel sneller oplossen. Stel bijvoorbeeld dat na een uitgebreid getaltheoretisch onderzoek een expliciet verband komt vast te staan tussen twee opeenvolgende oplossingen, dan kunnen we na zo'n onderzoek een algorithme maken die de ene oplossing na de andere genereert, zonder probeeroplossingen, en misschien is zo'n algorithme een orde efficienter dan de onze.
Hier zien we een verschijnsel dat veelvuldig voorkomt bij het ontwerpen van algorithmen: een lage investering in analyse vooraf en een vaak inefficiente, duidelijke en makkelijk bewijsbare algorithme, tegen een heel hoge investering in vooraf-analyse en een efficiente, misschien onoverzichtelijke en moeilijker begrijpbare algorithme als resultaat. Daartussen bevindt zich een optimale versie. Een compromis dat telkens weer gesloten moet worden, en sterk afhankelijk is van het probleem in kwestie, van het medium waarin de algorithme gebruikt moet worden, en van kwaliteiten van degene die de algorithme moet ontwerpen.

Opgave 9.
Gegeven is een natuurlijk getal M. Geef een algorithme die alle natuurlijke paren (n,m) genereert die voldoen aan $n^3-m^2=0$, $m \leq M$. De beperking is dat als arithmetische operaties uitsluitend optelling en aftrekking zijn toegestaan.

Opgave 10.
Gegeven is een natuurlijk getal R. Genereer alle paren niet-negatieve gehele getallen (a,b) die voldoen aan $a \leq b$, $a^2+b^2=R$.

Opgave 11.
Gegeven is een natuurlijk getal M. Genereer alle delers van M.

Opgave 12.

Als gegeven is dat een integer gedeclareerde variabele uitsluitend geheeltallige waarden kan aannemen die niet groter zijn dan 10^7 en niet kleiner dan -10^7, maak dan een algorithme die alle decimale cijfers van 734! berekent en uitprint.

(N.B. Het netto-effect van de operator "printsymbol(x)" is dat het 'symbool' x wordt afgedrukt op de regeldrukker indien x geheel is en $0 \leq x \leq 9$, en het is ongedefinieerd wanneer x andere geheeltallige waarden heeft.)

Opgave 13.

Gegeven zijn twee eindige verzamelingen A en B, bestaande uit m resp. n gehele getallen, en gerepresenteerd als

integer array A[1:m], integer m, $m \geq 0$, $A[i-1] < A[i]$ voor $1 < i \leq m$

integer array B[1:n], integer n, $n \geq 0$, $B[j-1] < B[j]$ voor $1 < j \leq n$.

Maak een algorithme die zowel $A \cap B$, als $A \cup B$, als $A \setminus B$ bepaalt.

Schrijf bovendien een boolean operator die bepaalt of $A \subset B$.

Opgave 14.

Gegeven is integer m, integer array A[1:m], $m \geq 0$, $A[i-1] \leq A[i]$ voor $1 < i \leq m$,

en integer n, integer array B[1:n], $n \geq 0$, $B[j-1] \leq B[j]$ voor $1 < j \leq n$.

Maak een algorithme die alle elementen van A en van B onderbrengt in een integer array C[1:(m+n)] en wel op een zodanige manier dat geldt $C[k-1] \leq C[k]$ voor $1 < k \leq m+n$.

Opgave 15.

Gegeven is een eindige verzameling A bestaande uit gehele getallen en gerepresenteerd als integer m, integer array A[1:m], $m \geq 0$, $\forall_{1<i\leq m}(A[i-1] < A[i])$.

Maak een niet-recursief werkende algorithme die alle deelverzamelingen van A genereert.

Opgave 16.

Maak een niet-recursieve algorithme die van een gegeven natuurlijk getal n alle partities in de natuurlijke getallen bepaalt.

Laten we beginnen met te formuleren wat een partitie van n in de natuurlijke getallen voorstelt.

Een partitie van n in de natuurlijke getallen -kortweg een partitie van n te noemen- is een rijtje $(p_1,\ldots,p_1)$ bestaande uit natuurlijke getallen en met som gelijk aan n.

Alle partities van 4 zijn

(4), (3,1), (2,2), (1,3), (2,1,1), (1,2,1), (1,1,2) en (1,1,1,1).

Het zijn er acht in aantal.

Algemeen geldt dat het aantal partities a_n van n gelijk is aan 2^{n-1}.

Dit is eenvoudig aan te tonen via een recurrentierelatie die voor a_n geldt, nl.

$a_n = \Sigma_{i=1}^{n} a_{n-i}$ voor $n\geq 1$, $a_0=1$,

i.e. een partitie voor n kan gevormd worden door $p_1=i$ te nemen met $1\leq i\leq n$, en een partitie van n-i erachter te zetten.

Nu is

$$a_n = \Sigma_{i=1}^{n} a_{n-i} = \Sigma_{i=2}^{n} a_{n-i} + a_{n-1} = \Sigma_{i=1}^{n-1} a_{n-1-i} + a_{n-1} = a_{n-1}+a_{n-1}$$

$$= 2a_{n-1} \quad \text{voor } n>1, \quad \text{en } a_1=a_0=1.$$

Dus is $a_n = 2^{n-1}$ voor $n\geq 1$.

Wat de algorithme betreft moeten we nu overwegen in welke volgorde we de partities van n wensen te genereren. In de bovenstaande opsomming van de partities van 4 hebben we de partities gerangschikt in volgorde van niet-afnemende lengte. Echter zoals te constateren valt biedt dit criterium alléén nog geen uitsluitsel, daar er in het algemeen verscheidene partities zijn van gelijke lengte. In dit geval hebben we de partities gerangschikt in alfabetisch omgekeerde volgorde. Deze twee ordeningscriteria tesamen definieren dan een volledige ordening in de verzameling van alle partities van n.

Er bestaat echter een eenvoudiger ordeningscriterium dan dit samengestelde, nl. gewoon de alfabetische ordening.

In alfabetische volgorde zijn de partities van 4

(1,1,1,1), (1,1,2), (1,2,1), (1,3), (2,1,1), (2,2), (3,1) en (4).

Om te kunnen beslissen of deze volgorde algorithmisch hanteerbaar is moeten we onderzoeken welke relatie er bestaat tussen een partitie van n en de in alfabetische zin onmiddellijk volgende partitie van n. Daartoe definieren we de alfabetische volgorde iets nauwkeuriger.
Laat $\underline{p}=(p_1,\ldots,p_1)$ en $\underline{q}=(q_1,\ldots,q_k)$ twee verschillende partities van n voorstellen, en m het kleinste natuurlijke getal waarvoor $p_m \neq q_m$, dan heet $\underline{q}$ alfabetisch groter dan $\underline{p}$ -notatie $\underline{q} \gtrdot_a \underline{p}$- d.e.s.d.indien $q_m > p_m$.
Met deze definitie van alfabetische volgorde is het niet moeilijk om in te zien dat voor de alfabetisch onmiddellijke successor $S_a(\underline{p})$ van een partitie $\underline{p}=(p_1,\ldots,p_1)$ van n geldt: $S_a(\underline{p}) = (p_1,\ldots,p_{1-1}+1,1,\ldots,1)$,
waarbij het staartstuk bestaat uit p_1-1 enen.

Op grond van deze eenvoudige relatie tussen twee opeenvolgende partities kunnen we veilig de maakbaarheid van de operator veronderstellen, die een partitie $\underline{p}$ transformeert in $S_a(\underline{p})$. Zonder dat we ons nu bekommeren om de gedetailleerde layout van deze operator kunnen we nu toch al de algorithme opschrijven.

```
begin partition p of n;
      INITIALIZE p WITH THE ALPHABETICALLY FIRST PARTITION OF n;
      while p IS NOT THE ALPHABETICALLY LAST PARTITION OF n do
        begin PRINT PARTITION p;
              TRANSFORM p INTO THE ALPHABETICALLY NEXT PARTITION OF n
        end;
      PRINT LAST PARTITION p OF n
end;
```

Deze algorithme is opgesteld in termen van objecten en operatoren die niet toebehoren aan het primitieve typen- en opdrachtenrepertoire van de taal waarin we de algorithme moeten uitdrukken. We hebben de algorithme geformuleerd in een terminologie die een directe weerspiegeling is van de wijze waarop we over de algorithme hebben gedacht, en we hebben van deze terminologie verondersteld dat deze uitdrukbaar is in de primitieve typen en opdrachten van de machine waarvoor de algorithme bedoeld is.
Het voordeel van het beschrijven van een algorithme in termen van abstractere -nog niet primitieve- objecten en operatoren is enerzijds gelegen in het feit dat we een duidelijker getuigenis afleggen van het soort abstractie dat we tijdens het bedenken van de algorithme gepleegd hebben en van het soort structuur dat we hebben aangebracht in het probleem, en anderzijds dat we allerlei gedetailleerde beslissingen over hoe de diverse operatoren en objecten samengesteld zijn explicieter uitstellen tot een nader tijdstip.

De met deze beschrijving van de algorithme corresponderende abstractie betreft de gedetailleerde layout van het object partition p of n en die van de op dit object werkende operatoren en inspectiehandelingen. Zelfs kunnen we stellen dat deze beschrijving -als zodanig- abstraheert van het feit dat partition p of n een partitie van n voorstelt, immers dit wordt pas expliciet tot uitdrukking gebracht zodra we precies formuleren wat het netto-effect van de diverse operatoren is. En dan is het ook duidelijk dat de met deze beschrijving van de algorithme corresponderende beslissingen zijn geweest, enerzijds dat we de partities van n alfabetisch geordend hebben en anderzijds dat we besloten hebben een algorithme te maken die de partities van n genereert in deze alfabetische volgorde.
Verder zien we dat, benevens de constante n, de variabele partition p of n het enige object is dat op dit niveau van beschrijving een rol speelt in de algorithme. Dat wil zeggen dat alle operatoren en inspectiehandelingen samenwerken en communiceren - en begrepen kunnen worden - via de waarden die de grootheid p aanneemt. De variabele p en de gegeven constante n tesamen vormen de zogenaamde 'interface', en het is uitsluitend in termen van deze interface-grootheden dat we het netto-effect van de diverse operatoren en inspectiehandelingen kunnen beschrijven. We kunnen welhaast stellen dat een van de grootste opgaven in de activiteit programmeren is, het zodanig structureren van het gegeven probleem dat de eruit voortvloeiende interface begrijpbaar, manipuleerbaar, maar bovenal op eenvoudige wijze beschrijfbaar is.

Tenslotte zullen we in dit voorbeeld niet tot in het kleinste detail gaan formuleren wat precies het gewenste netto-effect van de gebruikte operatoren is, doch maar meteen overgaan tot de verfijningen van de in de algorithme voorkomende niet-primitieve concepten.

```
partition p of n:   {integer length; integer array p[1:n]}

INITIALIZE p WITH THE ALPHABETICALLY FIRST PARTITION OF n:
      begin integer i;
            i:=0; while i<n do begin i:=i+1; p[i]:=1 end;
            length:=n
      end

p IS NOT THE ALPHABETICALLY LAST PARTITION OF n:
      begin length≠1 end
```

```
PRINT PARTITION p:
      begin integer i;
            i:=0; while i< length do begin i:=i+1; print(p[i]) end;
            new line
      end

TRANSFORM p INTO ALPHABETICALLY NEXT PARTITION OF n:
      begin integer i,r;
            i:=length; r:=p[i]-1; i:=i-1; p[i]:=p[i]+1;
            while r>0 do begin i:=i+1; p[i]:=1; r:=r-1 end;
            length:=i
      end

PRINT LAST PARTITION p OF n:
      begin print(p[1]) end.
```

Bewijs zelf dat vlak na elke uitvoering van de repeatable statement in de detaillering van TRANSFORM p INTO THE ALPHABETICALLY NEXT PARTITION OF n geldt dat

$$\sum_{j=1}^{i} p[j] + r = n.$$

Toon bovendien aan dat, indien vlak voor uitvoering van de operator geldt dat $p=\underline{p}$, dat dan onmiddellijk na uitvoering ervan $p=S_a(\underline{p})$.

Opgave 17.

Maak een algorithme die met gebruikmaking van een recursieve operator alle partities van een gegeven natuurlijk getal n bepaalt.

Bij de behandeling van de niet-recursieve algorithme voor het genereren van alle partities hebben we terloops een formule voor het aantal afgeleid. Laten we deze afleiding nu eens wat nader beschouwen.
Gevraagd wordt het aantal verschillende partities, a_n, van een natuurlijk getal n te bepalen. Een partitie van n is een rijtje $(p_1,\ldots,p_l)$ van natuurlijke getallen met som n. Dit betekent dat $1\leq l\leq n$ is, en dat voor elk element p_i, $1\leq i\leq l$, geldt dat $1\leq p_i\leq n$ is. In het bijzonder kan p_1 alle waarden in het bereik $1,\ldots,n$ aannemen, en bestaat er voor ieder van deze door p_1 aangenomen waarden minstens één partitie van n. We kunnen nu het aantal

partities van n bepalen door voor elke toegestane waarde van p_1 het aantal partities van $n-p_1$ te bepalen. Hiermee is dan een recurrentierelatie gegeven voor het aantal:

$$a_n = \Sigma_{j=1}^{n} a_{n-j} , \quad n \geq 1$$
$$a_0 = 1.$$

Uitgaande van de veronderstelling dat we het aantal partities van m kennen, $m < n$, kunnen we nu ook het aantal partities van n bepalen. Van dit idee maken we gebruik voor de voortbrenging, zeg het afdrukken op de regeldrukker, van alle gevraagde partities. Het genereren van alle partities van n komt neer op het genereren van alle rijtjes $(p_1,\ldots,p_1)$ met som n. Als we nu veronderstellen dat we alle partities van m, met $m < n$, kunnen genereren, dan vinden we de partities van n door p_1 achtereenvolgens de waarden $1,\ldots,n$ te laten aannemen en voor iedere waarde van p_1 de partities van $n-p_1$ te genereren. Iedere partitie van $n-p_1$ moet dan tijdens dit proces aanleiding geven tot een partitie van n. Daartoe moet het afgesplitste element p_1 tijdens de bepaling van de partities van $n-p_1$ beschikbaar blijven, opdat op elk moment dat een partitie $(q_1,\ldots,q_k)$ van $n-p_1$ gevonden is, het rijtje $(p_1,q_1,\ldots,q_k)$ kan worden uitgeprint. M.a.w. het genereren van alle partities $(p_1,\ldots,p_1)$ van n komt - onder de gemaakte inductiehypothese - neer op het achtereenvolgens toekennen van de waarden $1,\ldots,n$ aan p_1, en voor iedere waarde van p_1 het staartstuk $(p_2,\ldots,p_1)$ alle partities van $n-p_1$ te laten doorlopen, en tenslotte voor iedere gevonden partitie van $n-p_1$ te zorgen dat het totale rijtje $(p_1,\ldots,p_1)$ wordt afgedrukt.

Deze aftastende analyse geeft aanleiding tot de volgende - meer algemene - opzet. Laat $(p_1,\ldots,p_k)$ een rijtje van natuurlijke getallen voorstellen met $\Sigma_{i=1}^{k} p_i \leq n$. Zij $m := n - \Sigma_{i=1}^{k} p_i$, dus $m \geq 0$.
Hoe vinden we nu alle partities $(p_1,\ldots,p_k,\ldots,p_1)$ van n die alle het gemeenschappelijke beginstuk $(p_1,\ldots,p_k)$ hebben? Als $m=0$, dan stelt $(p_1,\ldots,p_k)$ reeds een partitie van n voor. Als $m>0$, dan vinden we alle partities van n met dit beginstuk door het rijtje $(p_1,\ldots,p_k)$ met één element p_{k+1} uit te breiden, dit element achtereenvolgens de waarden $1,\ldots,m$ te laten aannemen, en tenslotte voor iedere waarde van p_{k+1} alle partities van n te genereren die als beginstuk de rij $(p_1,\ldots,p_{k+1})$ hebben. Op deze wijze vinden we dan alle partities van n door met een lege beginrij, $k=0$, te starten.
Dit resulteert in de volgende stelling:

stelling: Zij $(p_1,\ldots,p_k)$ een eventueel lege rij van natuurlijke getallen, en $m:=n-\Sigma_{i=1}^{k} p_i \geq 0$. Dan geldt voor de verzameling $V_n(p_1,\ldots,p_k)$ van alle partities van n die als beginstuk $(p_1,\ldots,p_k)$ hebben dat $V_n(p_1,\ldots,p_k) = \bigcup_{p_{k+1}=1}^{m} V_n(p_1,\ldots,p_k,p_{k+1})$.

En de algorithme die alle partities van een gegeven n genereert volgt hieruit onmiddellijk.

```
begin sequence p of n;
      procedure genpart(integer value m);
         if m=0 then PRINT SEQUENCE p
                else
            begin integer q;
                  EXTEND SEQUENCE p;
                  q:=0;
                  while q<m do
                    begin q:=q+1; PLACE q IN EXTENDED SEQUENCE p;
                          genpart(m-q)
                    end;
                  SHRINK SEQUENCE p
            end;
      INITIALIZE SEQUENCE p;  genpart(n)
end
```

met als detailleringen

```
sequence p of n:   {integer k; integer array p[1:n]}

PRINT SEQUENCE p:
       begin integer i;
             i:=0; while i<k do begin i:=i+1; print(p[i]) end;
             new line
       end

EXTEND SEQUENCE p:
       begin k:=k+1 end

PLACE q IN EXTENDED SEQUENCE p:
       begin p[k]:=q end
```

SHRINK SEQUENCE p:

begin k:=k-1 end

en tenslotte

INITIALIZE SEQUENCE p:

begin k:=0 end.

We zullen nu aantonen dat deze algorithme inderdaad alle partities van het gegeven natuurlijk getal n genereert.
Daartoe bewijzen we de volgende eigenschap van de recursieve procedure genpart(m):

als vlak voor uitvoering van genpart(m) tussen de parameter m en de variabelen n,k en (p[1],...,p[k]) de relatie $m=n-\Sigma_{j=1}^{k} p[j]$ and $m\geq 0$ geldt, dan is het netto-effect van de uitvoering van genpart(m), dat alle partities van n, die als beginstuk de rij (p[1],...,p[k]) hebben, op de regeldrukker worden afgedrukt.

Ofwel in termen van de notatie die door Hoare en door de voorgaande stelling gehanteerd wordt:

$m=n-\Sigma_{j=1}^{k} p[j]$ and $m\geq 0$ {genpart(m)} $V_n(p[1],...,p[k])$ afgedrukt.

Deze eigenschap van de recursieve operator genpart(m) bewijzen we met inductie naar de parameter m.

m=0: De premisse reduceert in dit geval tot $n=\Sigma_{j=1}^{k} p[j]$, m.a.w. de rij (p[1],...,p[k]) stelt een partitie van n voor. In de algorithme zien we dat voor m=0 de handeling PRINT SEQUENCE p wordt uitgevoerd, dus dat de rij (p[1],...,p[k]) wordt afgedrukt, zijnde de enige partitie van n met dit beginstuk.

m>0: We nemen nu als inductiehypothese aan dat de eigenschap geldt voor alle parameters m' waarvoor $0\leq m'<m$ is.
De eerste handeling die wordt uitgevoerd -wegens m>0- is EXTEND SEQUENCE p, waardoor de premisse overgaat in $m=n-\Sigma_{j=1}^{k-1} p[j]$.
Vervolgens wordt de locale variabele q geinitialiseerd, en dan volgt een repetitiecycle. De invariante relatie die na elke uitvoering van de te herhalen statement geldt is, dat de verzameling V van alle (tot dan toe) in genpart(m) gegenereerde partities van n te schrijven is als de vereniging van alle $V_n(p[1],...,p[k-1],l)$ met l=1,...,q.
Ofwel iets volledieger:
na elke uitvoering van de te herhalen statement geldt invariant

$0 \le q \le m$ and $V = \bigcup_{l=1}^{q} V_n(p[1],\ldots,p[k-1],l)$ and $m = n - \Sigma_{j=1}^{k-1} p[j]$.

Het gevolg is dat na de laatste uitvoering van de te herhalen statement geldt dat

$V = \bigcup_{l=1}^{m} V_n(p[1],\ldots,p[k-1],1)$, omdat dan $q=m$.

En door de handeling SHRINK SEQUENCE p gaat dit over in

$V = \bigcup_{l=1}^{m} V_n(p[1],\ldots,p[k],1)$.

Op grond van de stelling is nu anderzijds

$\bigcup_{l=1}^{m} V_n(p[1],\ldots,p[k],1) = V_n(p[1],\ldots,p[k])$,

zodat de verzameling V van alle gedurende deze handeling genpart(m) afgedrukte partities van n precies gelijk is aan $V_n(p[1],\ldots,p[k])$, waarmee de eigenschap bewezen is.

We moeten wel nog aantonen dat de relatie

$V = \bigcup_{l=1}^{q} V_n(p[1],\ldots,p[k-1],1)$

invariant geldt na iedere uitvoering van de te herhalen statement.

Hiertoe maken we gebruik van de inductiehypothese.

Voor $q=0$ geldt de invariante relatie, omdat enerzijds nog geen enkele partitie van n gedurende deze call van genpart(m) gegenereerd is, i.e. $V=\emptyset$, en omdat anderzijds $\bigcup_{l=1}^{0} V_n(p[1],\ldots,p[k-1],1) = \emptyset$.

Indien $q<m$ is dan transformeert de invariante relatie ten gevolge van de twee acties ' q:=q+1; PLACE q IN EXTENDED SEQUENCE p' in

$V = \bigcup_{l=1}^{q-1} V_n(p[1],\ldots,p[k-1],1)$ and $p[k]=q$.

Dan volgt een aanroep van de recursieve operator, maar nu met argument m-q. Noem m'=m-q. Omdat $1 \le q \le m$ is, geldt $0 \le m' < m$. Verder is, wegens $m = n - \Sigma_{j=1}^{k-1} p[j]$ en wegens $p[k]=q$, $m' = n - \Sigma_{j=1}^{k} p[j]$ and $m' \ge 0$ hetgeen precies de premisse is voor de aanroep van genpart(m').

Op grond van de inductiehypothese volgt nu dat onmiddellijk na uitvoering van genpart(m-q) alle partities uit $V_n(p[1],\ldots,p[k])$ afgedrukt zijn. En omdat nog steeds $q=p[k]$ geldt, is de conclusie dat

$V = \bigcup_{l=1}^{q} V_n(p[1],\ldots,p[k-1],1)$, waarmee de invariantie van deze relatie aangetoond is.

Hiermee is bovendien de geponeerde eigenschap van de recursieve operator bewezen, en om nu te verifieren dat de algorithme inderdaad alle partities van n genereert hoeven we alleen nog maar op te merken dat in het'hoofdprogramma' - waar de eerste aanroep van de recursieve operator plaats vindt -

tengevolge van INITIALIZE SEQUENCE p inderdaad aan de premisse voldaan wordt (k=0) en dat $V_n(\emptyset)$ inderdaad de verzameling van alle partities van n voorstelt.

Wederom kan in de zin van Hoare's axiomatische aanpak van problemen als deze bovenstaand bewijs geen bewijs genoemd worden. Zo hebben we bijvoorbeeld het begrip'procedure' als elementair gereedschap uit de door ons gebezigde programmeertaal niet geformaliseerd, en is het bewijs gebaseerd op een erg intuitief idee van het wezen van een locale variabele en van een parameter. Verder hebben we voetstoots aangenomen dat volledige inductie hét mathematisch apparaat bij uitstek is waarmee we bewijstechnieken omtrent recursieve procedures associeren.
Wat dit laatste betreft zegt Hoare in zijn artikel 'Procedures and Parameters, an Axiomatic Approach' in Lecture Notes in Mathematics, Symposium on Semantics of Algorithmic Languages, Springer-Verlag,pp 102-116 :
"This assumption of what we want to prove before embarking on the proof explains well the aura of magic which attends a programmer's first introduction to recursive programming."
De aanwezigheid van het gereedschap 'recursieve procedure' in een programmeertaal is meer dan alleen maar wat franje. Talloze malen worden we geconfronteerd met problemen waarop we mentaal alleen maar enige greep kunnen krijgen door een inductief, recursief begrip ervan. Dit begrip wordt in een algorithme dan weerspiegeld door gebruik van het concept recursie. Is dit fundamentele gereedschap ons echter onthouden - en er zijn nog steeds programmeertalen waarin op grote schaal geprogrammeerd wordt en die het concept recursie missen - dan zijn we gedwongen, teneinde toch een algorithme voor het probleem te ontwerpen, het onderhavige probleem op een volslagen andere, wellicht onnatuurlijke, en veelal een orde van grootte moeilijker wijze te begrijpen. Welhaast kunnen we stellen dat recursie, bijna net als het begrip variabele, behalve een programmeergereedschap vooral ook een denkgereedschap voorstelt. En worden ons zulke concepten onthouden dan kan dit -begrijpelijk- in gedachtenkronkels en in niet-overtuigende, onbegrijpbare algorithmen resulteren.
Voor verdere overdenkingen en beschouwingen omtrent recursie wordt verwezen naar het artikel van drs. C. Bron: 'Over het nut van recursieve programmeertechnieken' in Informatie nr. 12, december 1970.

Tenslotte nog enkele opmerkingen naar aanleiding van de wijze waarop dit probleem opgelost is.

Bij het bewijs van de correctheid van de algorithme hebben we verzuimd een belangrijk aspect van het gebruik van recursieve procedures naar voren te brengen, en dat is het eindigheidsbewijs. We moeten ons ervan overtuigen dat de procedure die zichzelf aanroept dit niet ad infinitum blijft doen. Hier is de toepasselijke redenering dat we naar een willekeurige aanroep 'genpart(m)' kijken, vervolgens enerzijds constateren dat $m \geq 0$ is, en anderzijds dat voor elke inwendige call van deze procedure het argument m' voldoet aan $m' < m$, tengevolge waarvan aan het aantal elkaar omvattende aanroepen van de procedure een expliciete bovengrens is gesteld.
Men voere dit eindigheidsbewijs zelf iets netter.

Een tweede overweging, die wellicht valide is voor het hele terrein van de activiteit "programma's ontwerpen" en zelfs voor de activiteit "ontwerpen" in het algemeen, maar die in het kader van deze opgave ook al op zijn plaats is, betreft de heuristiek. Veelal treft men bij de behandeling van een vraagstuk, en vooral in de wiskunde, een gepolijst en gladlopend verhaal aan, meestal gepresenteerd als een opeenvolging van veronderstellingen, definities, lemma's en theorema's. De heuristische aspecten van het ontwerp van zo'n oplossing of theorie zijn op deze wijze geheel of nagenoeg geheel onder de tafel geraakt. Natuurlijk zijn ze wel aanwezig geweest, doch vooral in een harde wetenschap als de wiskunde, dienen ze ten tijde van de uiteindelijke presentatie van de oplossing bijna volledig naar de achtergrond te zijn verdwenen. Dit, althans, leert de dagelijkse wiskundige praktijk. Deze stijl ziet men wel eens overgenomen in andere gebieden waar men zich bezig houdt met 'ontwerpen', en ook wel in de algorithmiek. Het lijkt nu de moeite waard om te overwegen of deze stijl,vooral in de programmering, gerechtvaardigd is. Laten we daartoe veronderstellen dat de oplossing van dit vraagstuk op de strenge wiskundige werkwijze was geserveerd. Dan was het onvermijdelijk geweest met een heel scala van definities, stellingen en eigenschappen omtrent partities voor de dag te komen. Vervolgens zou dan de algorithme zijn geformuleerd en tenslotte zou het formele correctheidsbewijs, in alle glorie, ten tonele zijn gevoerd. Het staat buiten kijf dat een dergelijke verhandeling op zijn allerminst tien keer zoveel ruimte op papier in beslag zou nemen dan de gevraagde algorithme. Een ietwat ervaren programmeur daarentegen, zou met de algorithme in de hand en met een hoeveelheid additionele uitleg al spoedig ervan overtuigd zijn dat de gemaakte algorithme correct is. Het formele correctheidsbewijs en de formalistisch opgestelde vooraf-analyse echter, zouden hem waarschijnlijk niet sterker overtuigen, en misschien zouden zij hem zelfs doen twijfelen.

Waar gaat het nu om? Indien het erom gaat een algorithme te ontwerpen, dan is het uitermate gewenst de algorithme te omlijsten met commentaar, met documentatie. Deze documentatie moet erop gericht zijn een beschrijving van de oplosmethode te geven, en wel zodanig dat de eruit voortvloeiende algorithme 'triviaal' is, en duidelijk correct. Vooral dít is een eigenschap die we van documentatie moeten eisen, en hieruit volgt -en het lijkt een open deur- dat documentatie leesbaarder moet zijn dan de bijbehorende algorithme. Verder kan de in de documentatie aanwezige analyse aan het probleem zich in het heuristische vlak bewegen, indien het maar mogelijk blijft de op deze wijze gepleegde analyse te formaliseren zodra dit gewenst is. Of een analyse correct en toepasbaar is hangt niet af van de manier waarop ze gepleegd en beschreven is -heuristisch of formeel- maar hangt af van hoe degene die de analyse uitgevoerd heeft ertoe in staat is geweest het probleem te concipieren, te structureren en onder zijn geestelijke controle te krijgen. Een goed idee wordt niet beter naarmate het formeler gepresenteerd wordt!
Een voordeel dat tenslotte lijkt toe te behoren aan een ietwat heuristische behandeling van de analyse van een probleem is dat duidelijker wordt benadrukt via welk soort overwegingen de gevonden oplossing tot stand is gekomen, en misschien dat deze stijl - meer dan de formele - inzicht kan verschaffen in de wijze waarop oplossingen tot staan komen, en dat is toch waar het in het vak programmeren bijna altijd op aan komt.

Opgave 18.
Gegeven is een natuurlijk getal n. maak een niet-recursieve algorithme die alle partities $(p_1,\ldots,p_k)$ van n genereert die voldoen aan $\forall_{1\leq i<k}(p_i\leq p_{i+1})$. Maak vervolgens ook een algorithme die deze partities genereert met gebruikmaking van een recursieve operator.

Opgave 19.
Gegeven zijn twee natuurlijke getallen 1 en n. Maak een algorithme die alle partities $(p_1,\ldots,p_k)$ van n genereert die voldoen aan $k\leq 1$ en aan $\forall_{1\leq i<k}(p_i\leq p_{i+1})$.

Opgave 20.

Gegeven zijn vier natuurlijke getallen k,l,m en n, en een integer array u[1:l]. Gegeven is bovendien dat k≤l en dat m≤n. Genereer alle l-tupels $(x_1,\ldots,x_l)$ die voldoen aan

- $\forall_{1\le i\le l}(x_i \in Gh,\ 0\le x_i\le u[i])$,
- $\forall_{1\le i<l}(x_i\le x_{i+1})$,
- $\sum_{j=1}^{k} x_j = m, \quad \sum_{j=1}^{l} x_j = n.$

Opgave 21.

Gegeven zijn twee natuurlijke getallen a en b. Construeer een algorithme die het aantal verschillende manieren bepaalt waarop een bedrag van b cents betaald kan worden met behulp van centen, stuivers, dubbeltjes, kwartjes en guldens, indien bovendien vereist wordt dat het totaal aantal munten niet meer dan a mag bedragen.

Opgave 22.

Gegeven is een integer array z[1:m], m≥1, $\forall_{1\le i\le m}(z[i]\ge 1)$, $\forall_{1\le i<m}(z[i]<z[i+1])$. Gegeven zijn bovendien twee natuurlijke getallen l en n. Gevraagd wordt een algorithme te schrijven die nagaat of er een k-tupel $(p_1,\ldots,p_k)$ bestaat zodanig dat

- $1\le k\le l$,
- $\forall_{1\le i\le k}\exists_{1\le j\le m}(p_i=z[j])$,
- $\sum_{i=1}^{k} p_i = n.$

(Of iets anders geformuleerd: ga na of het bedrag n plakbaar is met maximaal k zegels uit de serie z[1],...,z[m].)

Opgave 23.

Gegeven is een natuurlijk getal n. Bepaal, met gebruikmaking van een recursieve operator, alle permutaties van {1,...,n}.

(Een permutatie van {1,...,n} is een eeneenduidige afbeelding van {1,...,n} op {1,...,n} , en kan worden gerepresenteerd als een rijtje $(p_1,\ldots,p_n)$ waarbij p_i het beeld van i voorstelt (1≤i≤n).)

Opgave 24.

Gegeven is een natuurlijk getal n. Bepaal alle permutaties $(p_1,\ldots,p_n)$ van $\{1,\ldots,n\}$ waarvoor geldt dat

$\exists_{1<m<n}$($(p_1,\ldots,p_m)$ monotoon stijgend, $(p_m,\ldots,p_n)$ monotoon dalend) .

Opgave 25.

Gegeven is een natuurlijk getal n. Bepaal alle permutaties $(p_1,\ldots,p_n)$ van $\{1,\ldots,n\}$ waarvoor geldt dat $\forall_{1\le i\le n}(p_i\neq i)$. Als a_n het aantal van dit soort permutaties voorstelt, toon dan aan dat $a_n = n!\Sigma_{i=1}^{n}(-1)^i/i!$.

Opgave 26.

Gegeven zijn twee natuurlijke getallen m en n, $m\ge n$. Bepaal alle afbeeldingen van $\{1,\ldots,m\}$ op $\{1,\ldots,n\}$. (Een afbeelding op is een afbeelding die zodanig is dat elk element van de verzameling waarin wordt afgebeeld minstens één keer als beeld fungeert.)

Opgave 27.

Gegeven is een natuurlijk getal a. Beschouw de verzameling der cijfers in de decimale representatie van a (eenduidigheidsafspraak: meest significante cijfer ongelijk aan 0). Maak een algorithme die alle natuurlijke getallen genereert waarvan de decimale voorstelling dezelfde cijfers als a bevat.

Opgave 28.

Gegeven is een natuurlijk getal n. Maak een algorithme die alle rijtjes $(a_1,\ldots,a_{2n})$ genereert die voldoen aan

- $\forall_{1\le i\le 2n}(a_i\in\{1,-1\})$,
- $\forall_{1\le k\le 2n}\left(\sum_{i=1}^{k} a_i \ge 0\right)$,
- $\sum_{i=1}^{2n} a_i = 0$.

Ga na dat elke ongeorienteerde wortelboom met n takken eenduidig beschreven kan worden door precies één zo'n rijtje $(a_1,\ldots,a_{2n})$.
(De nu volgende opmerking staat los van de algorithme: Als α_n het aantal van dit soort rijtjes $(a_1,\ldots,a_{2n})$ voorstelt dan voldoet α_n aan het volgende recurrentieschema:

$\alpha_n = \sum_{i=0}^{n-1} \alpha_i \alpha_{n-i-1}$ voor $n \geq 1$, $\alpha_0 = 1$.

Toon bovendien aan dat $\alpha_n = \frac{(2n)!}{(n+1)!n!}$ voor $n \geq 0$.)

Opgave 29.

Gegeven is een ongerichte graaf in matrixrepresentatie. Maak een algorithme die bepaalt of de graaf samenhangend is.

(Een graaf met n punten kan gerepresenteerd worden door een boolean array G[1:n,1:n], waarbij G[i,j] de waarde true heeft indien i=j of indien i en j door een lijn van de graaf met elkaar verbonden zijn.)

Opgave 30.

Gegeven is een ongerichte graaf in matrixrepresentatie. Bepaal alle cycli in de graaf.

Opgave 31.

Gegeven is een ongeorienteerde boom bestaande uit n takken, n≥0. De knooppunten zijn genummerd van 1 t/m n+1 en de takken zijn voorgesteld door integer array from[1:n], to[1:n], length[1:n]. Hierbij zijn from[i] en to[i] de nummers van de beide knooppunten aan de i^{de} tak, en is length[i] de lengte van de i^{de} tak (1≤i≤n).

Maak nu een algorithme die het langste pad in de gegeven boom bepaalt.

Opgave 32.

Gegeven is een ongerichte graaf bestaande uit m punten, genummerd van 1 t/m m, en uit n lijnen, genummerd van 1 t/m n (m≥0, n≥0). De graaf is gegeven door de collectie lijnen, en wel door integer array from[1:n], to[1:n]. Hierbij zijn from[i] en to[i] de nummers van de beide eindpunten van de i^{de} lijn in de graaf (1≤i≤n). De arrays zijn zodanig gerangschikt dat from[i]<to[i] voor 1≤i≤n, en dat from[i]≤from[i+1] voor 1≤i<n. Maak een algorithme die nagaat of de gegeven graaf een boom is.

Opgave 33.

Een n-kubus (n-cube) is een ongerichte graaf bestaande uit 2^n punten, genummerd met de gehele getallen $0,1,2,\ldots,2^n-1$, waarvoor geldt dat er een lijn tussen i en j bestaat d.e.s.d. indien de binaire schrijfwijzen van i en j (n bits) in precies één positie verschillen (i.e. Hamming-afstand 1 hebben).
Ga na dat de n-kubus samenhangend is.
Een Hamiltoniaanse cyclus in een graaf is een cyclus in deze graaf die alle punten precies één keer bevat. In de 3-kubus is (0,4,6,2,3,7,5,1) een Hamiltoniaanse cyclus.
Maak een algorithme die bij gegeven n alle Hamiltoniaanse cycli in een n-kubus bepaalt (n natuurlijk getal).

Opgave 34.

Laat $\underline{h}^{(i)}=(h_0^{(i)},h_1^{(i)},\ldots,h_i^{(i)},\ldots)$ een oneindig voortlopende, monotoon stijgende rij van natuurlijke getallen zijn ($i\geq 0$). Zij $v=h_i^{(i)}$ de bij deze rij $\underline{h}^{(i)}$ behorende 'veger'. Nu wordt de rij $\underline{h}^{(i+1)}$ uit de rij $\underline{h}^{(i)}$ verkregen door uit $\underline{h}^{(i)}$ de elementen $h_{i+v}^{(i)}$, $h_{i+2v}^{(i)}$, $h_{i+3v}^{(i)}$,... te verwijderen.
Als de eerste rij de rij $\underline{h}^{(o)}=(2,3,4,5,\ldots)$ is, i.e. de rij der natuurlijke getallen groter dan 1, dan definieert de limietrij $\underline{h}=\lim_{i\to\infty}\underline{h}^{(i)}$ de verzameling der 'gelukkige getallen' (gelukkig omdat ze niet weggeveegd zijn).
Ter illustratie de eerste paar rijen:

$\underline{h}^{(0)}$ = (2,3,4,5,6,7,8,9,10,...)
v=2 ↓
$\underline{h}^{(1)}$ = (2,3,5,7,9,11,13,15,17,19,21,23,25,27,29,31,...)
v=3 ↓
$\underline{h}^{(2)}$ = (2,3,5,7,11,13,17,19,23,25,29,31,...)
v=5 ↓
$\underline{h}^{(3)}$ = (2,3,5,7,11,13,17,23,25,29,31,...)
v=7 ↓
⋮

$\underline{h}$ = rij der gelukkige getallen.

Construeer een algorithme die de eerste duizend gelukkige getallen bepaalt.

Opgave 35.

Gegeven is een integer array A[1:n] , $n\geq 0$. Van de rij (A[1],...,A[n]) is gegeven dat geen drie opeenvolgende elementen gelijk zijn. Maak een algorithme die het array A zodanig transformeert dat na afloop alle twee opeenvolgende gelijke elementen uit de oorspronkelijke rij verwijderd zijn, en de overblijvende elementen aaneengesloten en in de oorspronkelijke volgorde in het array

A voorkomen.

Opgave 36.

Gegeven zijn n punten in R_2, gerepresenteerd door real array x[1:n], y[1:n], n≥2. Hierbij stelt (x[i],y[i]) het coördinatenpaar voor het i^{de} punt voor. Maak een algorithme die middelpunt en straal van de kleinste cirkel bepaalt waarvoor geldt dat alle punten in of op deze cirkel liggen.

Opgave 37.

Maak een niet-recursieve algorithme die alle configuraties van acht dames op een 8x8-schaakbord genereert zodanig dat de dames elkaar niet aanvallen.

Opgave 38.

Gegeven is een willekeurige configuratie van vier dames op een 8x8-schaakbord. Schrijf een algorithme die nagaat of de verzameling van de nog niet door deze vier dames aangevallen velden door plaatsing van een vijfde dame wel gedekt kan worden. (Bedenk vooral een geschikte representatie van de gegeven configuratie van vier dames.)

Opgave 39.

In dit vraagstuk gaan we uit van de volgende definitie van een arithmetische expressie:

<arithmetic expression> ::= (<arithmetic sequence>)

<arithmetic sequence> ::= <operand>/<operand><operator><arithmetic sequence>

<operand> ::= <unsigned integer>/ <arithmetic expression>

<operator> ::= +/-/x/:/↑

<unsigned integer> ::= <digit>/<unsigned integer><digit>

<digit> ::= 0/1/2/3/4/5/6/7/8/9

Op een band staat een correcte arithmetische expressie. Maak een algorithme die de waarde van deze arithmetische expressie bepaalt.

(De gebruikelijke prioriteitsniveau's gelden voor de gegeven operatoren, en met behulp van haakjes kan deze doorbroken worden. Indien ook dit geen uitsluitsel geeft, dan geldt de regel dat de expressie van'links naar rechts' wordt uitgewerkt.

Bijvoorbeeld: (7+3-3:2)=8.5, (7+(3-3):2)=7, (8x3:4x6)=36.

Het netto-effect van de opdracht c:=RNC (Read Next Character) is dat c de waarde krijgt van het huidige symbool op de band, en dat het eventueel volgende symbool het huidige wordt. Voordat de eerste leesactie is geïnitialiseerd is het eerste symbool op de band het huidige.)

Opgave 40.

In dit vraagstuk hanteren we de volgende definitie van een arithmetische expressie:

<arithmetic expression> ::= (<arithmetic sequence>)

<arithmetic sequence> ::= <operand>/<operand><operator><arithmetic sequence>

<operator>::= +/-/x/:/↑

<operand> ::= <unsigned integer>/<arithmetic expression>/<identifier>

<unsigned integer> ::= <digit>/<unsigned integer><digit>

<identifier> ::= <letter>/<identifier><letter>/<identifier><digit>

<digit> ::= 0/1/2/3/4/5/6/7/8/9

<letter> ::= a/b/c/d/e/f/g/h/i/j/k/l/m/n/o/p/q/r/s/t/u/v/w/x/y/z

Op een band staat een rij symbolen bestaande uit letters,digits,operatoren en ronde haakjes. Schrijf een algorithme die nagaat of deze rij symbolen, in de volgorde waarin ze op de band staan, een arithmetische expressie voorstelt. De rij symbolen is afgesloten door een puntkomma (;).
De symbolen kunnen met de opdracht RNC van de band gelezen worden.

Opgave 41. (Een postzegelprobleem)

Op een brief mogen maximaal drie zegels geplakt worden. Gevraagd wordt een serie van M zegels $z_1,\ldots,z_M$ te ontwerpen zodanig dat het kleinste bedrag dat niet meer op een brief geplakt kan worden maximaal is.
Opmerking: De zegel met waarde 1 zal steeds in de serie moeten voorkomen, anders kan het bedrag 1 zelfs niet geplakt worden.
Voorbeeld: Als M=2 dan is de maximale serie de serie 1,3. Het eerste bedrag dat niet meer geplakt kan worden is 8, en er bestaat geen serie van twee waarvoor dit bedrag groter is.