PROGRAMMEERTALEN (Deel II)

door W. F. H. Alleijn

9 Andere vormen van programmeren

In de operationele research worden methodieken toegepast, die worden aangeduid als een vorm van programmeren. Voorbeelden hiervan zijn onder meer:
- lineair programmeren,
- dynamisch programmeren.

In principe gaat het hier echter om het oplossen van bepaalde problemen volgens een vaste methode: een algoritme of rekenregel. Dat oplossen kan met de hand geschieden, maar die methode kost in het algemeen teveel tijd. Daarom wordt dit soort problemen thans veelal met behulp van een computer-systeem opgelost.

Hierbij moet worden aangetekend dat de term programmeren in de operationele research verwarrend werkt. Beter zou gesproken kunnen worden van bijvoorbeeld lineair optimaliseren. Dit heeft echter maar weinig ingang gevonden.

Het lineair en dynamisch programmeren vindt in drie fasen plaats:
1 Het definiëren van het probleem en dit onderbrengen in een stelsel van vergelijkingen.
2 Het opstellen van het - voor dit probleem geschikte - algoritme.
3 Het programmeren van het algoritme.
Als programmeertalen worden daartoe Fortran of Algol gebruikt.

In sommige gevallen kan gebruik worden gemaakt van standaard-software-pakketten, waardoor het opstellen van het algoritme, alsmede het programmeren, kan vervallen. Wel moeten dan de in te voeren gegevens voldoen aan de eisen die het software-pakket stelt.

Een andere vorm van programmeren - die in de operational research voorkomt - is simulatie.

Simulatie wordt toegepast om een al dan niet bestaande situatie na te bootsen, met name om invloeden van wijzigingen op die situatie na te gaan. Gebruik wordt gemaakt van simulatie, indien het niet mogelijk is deze wijzigingen in de werkelijkheid uit te proberen. Simulatie kan ook weer met de hand worden uitgevoerd of met behulp van een computer-systeem. In het algemeen zijn de te simuleren situaties zo gecompliceerd en is het nodig om een dusdanig lange tijd te simuleren, dat het met de hand uitwerken een ondoenlijke zaak is.

De eerste fase in simulatie is het bouwen van een model (een wiskundig model), dat analoog is aan de te simuleren werkelijkheid.

De tweede fase is dit model onder te brengen in een computer-programma. Dit kan worden gedaan door een simulatie-programma te schrijven in Fortran of Algol, maar ook kan gebruik worden gemaakt van speciale simulatie-talen, zoals General Purpose Simulation System (GPSS), Simula en Simscript.

Tenslotte nog een globale aanduiding van de inhoud van lineair en dynamisch programmeren.

Gesproken wordt van *lineair programmeren* indien een (groot) aantal variabelen voldoet aan een stelsel van (vele) lineaire vergelijkingen. Onder deze voorwaarden moet een lineaire vergelijking van die variabelen geminimaliseerd of gemaximaliseerd worden. Het probleem moet dus in een stelsel van lineaire vergelijkingen worden ondergebracht.

Zowel bij het met de hand oplossen van het probleem als bij het oplossen met behulp van de computer, wordt gebruik gemaakt van de simplex-methode. Hiermede wordt, volgens een bepaalde systematiek, het optimaliseringspunt binnen een raam van mogelijkheden vastgesteld. Momenteel bestaan er een aantal standaard-software-pakketten die volgens deze methode werken.

Een voorbeeld van een probleem, dat met behulp van lineaire programmering kan worden opgelost, is het dieet-probleem.

Stel uit een aantal voedingsmiddelen, met bepaalde waarden aan calorieën, vet, eiwit e.d., en bepaalde kosten per eenheid, een dieet samen, dat aan bepaalde voorwaarden met betrekking tot calorieën, vet, eiwit e.d. voldoet en dat zo goedkoop mogelijk is.

Dynamisch programmeren berust op het kiezen van een bepaalde basisoplossing, waarna door stapsgewijs veranderen getracht wordt een betere oplossing te verkrijgen. Men krijgt als het ware families van oplossingen, waaruit steeds de beste worden gekozen.

Een voorbeeld van een probleem, dat o.a. met dynamisch programmeren kan worden opgelost, is:

Een bedrijf heeft n afdelingen en een totaal-bedrag om te investeren van x. Investering van een bepaald bedrag in een bepaalde afdeling geeft een bepaalde opbrengst.

Gevraagd wordt nu het beschikbare bedrag zodanig over de afdelingen te verdelen, dat het totale resultaat zo groot mogelijk is.

Verder is te noemen *heuristisch programmeren,* waarbij wordt gezocht naar een oplossing volgens een bepaalde methode, waarbij een aanvaardbare oplossing wordt verkregen, zonder zekerheid dat dit de beste oplossing is.

Bij heuristisch programmeren kan *niet* worden bewezen dat:
- een oplossing bestaat;
- indien deze methode niet tot een oplossing komt, geen oplossing bestaat;
- de verkregen oplossing optimaal is;
- de verkregen oplossing *niet* optimaal is.

Hierbij is aan te tekenen - alhoewel het misschien een wat filosofisch probleem is - dat in vele gevallen wel kan worden bewezen, dat een heuristisch tot stand gekomen oplossing niet-optimaal is; namelijk in alle gevallen waarin een andere oplossing bekend raakt, die dichter bij het optimum ligt. In feite is dit ook wezenlijk voor de heuristische werkwijze, daar anders de eerste de

beste (of slechtste!) oplossing meteen als dé oplossing moet worden gezien, omdat van volgende oplossingen niet zou kunnen worden bewezen dat zij „beter" zijn.

Een voorbeeld van een - met heuristische methode op te lossen - probleem is een job/shop probleem: een serie karweien, bestaande uit een aantal bewerkingen, uit te voeren op een aantal machines met verschillende bewerkingstijden, een voor elk karwei gegeven afleveringstijdstip en voor elke machine de vroegste beschikbaarheid.

Gevraagd wordt een volgorde-plan op te stellen met een zo kort mogelijke doorlooptijd per karwei.

Van de hier behandelde methodieken is, naast de simulatie, de lineaire programmering het verst ontwikkeld; hiervoor zijn een groot aantal standaardpakketten beschikbaar. De andere methodieken verkeren nog niet in zo'n afgebakend stadium. Bovendien zijn er in de operationele research nog vele methodieken en algoritmen, die niet onder bovengenoemde programmeringswijzen vallen.

Het is tenslotte goed om er de aandacht op te vestigen, dat in de operationele research de nadruk ligt op het bouwen van de modellen en het vinden van de methodieken. Als dit gebeurd is, vormt het „op de computer zetten" van het probleem een relatief eenvoudige zaak.

In kort bestek volgt nu een voorbeeld van programmeren in verschillende talen.

10 Programma-voorbeeld

Probleemstelling
Druk een lijst af, aan de hand van een stapel ponskaarten, van de kaarten met het codegetal 07.

Maak een telling van de bedragen uit de kaarten met het codegetal 07, en een telling van de bedragen uit de overige kaarten met andere codegetallen.

Druk deze totalen onderaan de lijst af.

Gegevens:
De stapel bestaat uit ponskaarten met verschillende codegetallen en is afgesloten door een sluitkaart.
code: 2 alfanumerieke posities in de kolommen 1 en 2.
bedrag: 4 numerieke posities in de kolommen 3 t/m 6.
code laatste kaart: ZZ in de kolommen 1 en 2.

Wijze van afdrukken:

Code	*Bedrag in hele guldens*	De totalen van de bedragen worden niet groter dan *f* 100.000.–
XX	XXXX	
..	....	
..	....	
..	....	
TOTAAL	XXXXXX	
REST	XXXXXX	

Toekennen van gebieden in het werkgeheugen:

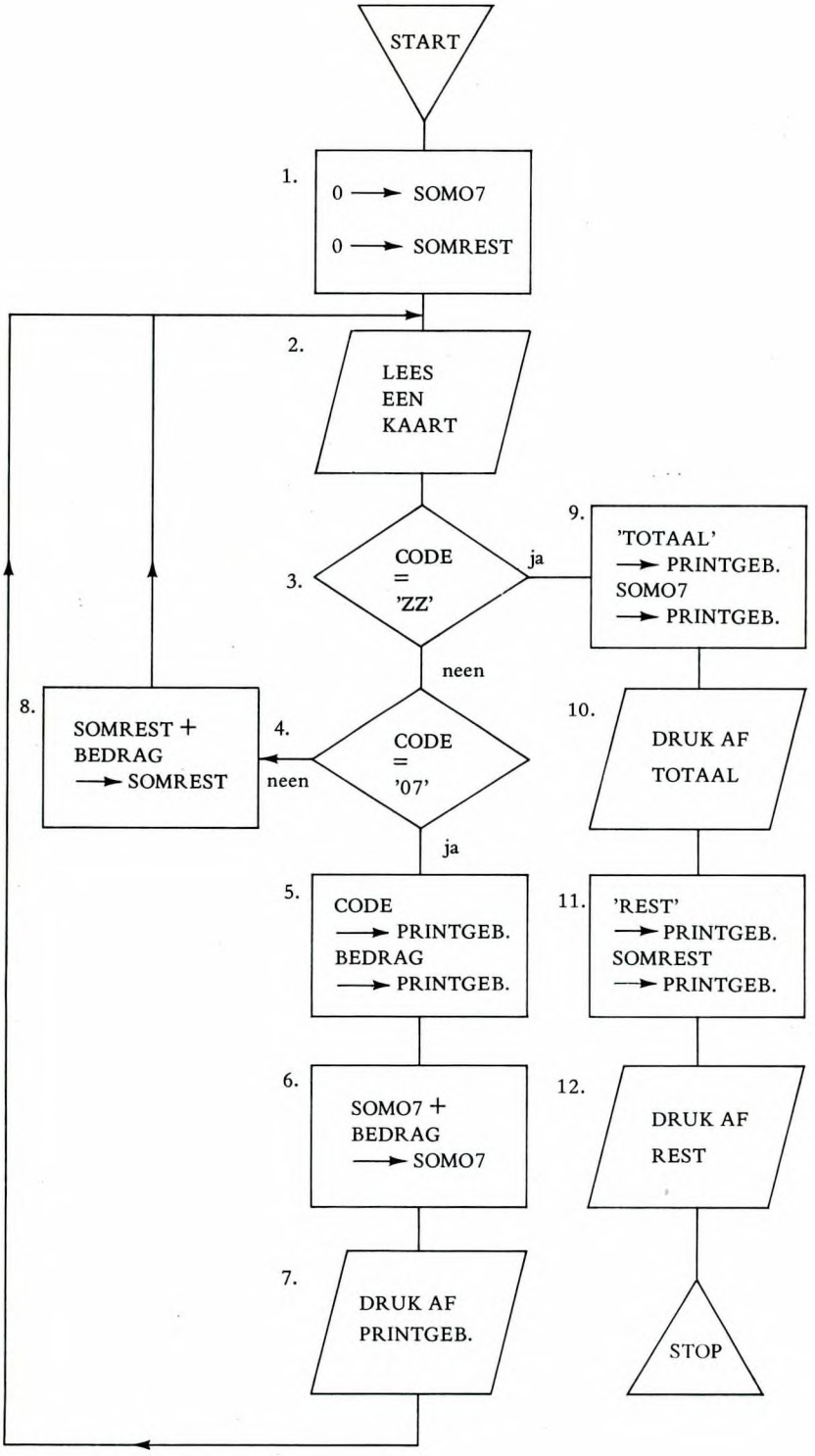
Leesgebied
0100 code 2 cijfers
0101 bedrag 4 cijfers

Werkgebied
0500 som van de bedragen van de code = 07 (SOMO7)
0501 som van de bedragen met een code ≠ 07 (SOMREST)
0502 constante 'ZZ' (COZZ)
0503 constante '07' (CO7)
0504 constante 'TOTAAL'
0505 constante 'REST'
0506 constante nul 0. (NUL)

Printgebied
0900 code
0901 bedrag

Eén woord kan bevatten: 12 cijfers inclusief + of – teken, of 8 alfanumerieke tekens.

PROGRAMMA SCHEMA



m a b blz. 251

PROGRAMMA-SCHEMA

Toelichting

1 Stel telvelden op 0.
2 Lees een kaart.
3 Vraag: code = 'ZZ'?
neen: dan geen sluitkaart en ga door met stap 4.
ja: dan sluitkaart en ga naar stap 9.
4 Vraag: code = '07'?
ja: ga door met stap 5.
neen: ga naar stap 8.
5 Breng code en bedrag naar het printgebied.
6 Tel bij de reeds verkregen telling van de bedragen uit de voorgaande 07 kaarten het bedrag uit de laatst gelezen 07 kaart.
7 Druk inhoud van printgebied af en keer terug naar „lees".
8 Tel bij de reeds verkregen telling van de bedragen uit de voorgaande *niet* 07 kaarten het bedrag uit de laatst gelezen *niet* 07 kaart, en keer terug naar „lees".
9 Breng het woord 'TOTAAL' en het bedrag van SOMO7 naar het printgebied.
10 Druk inhoud van printgebied af.
11 Breng het woord 'REST' en de som van de rest naar het printgebied.
12 Druk inhoud van printgebied af.

Machine-code			*PROGRAMMA-VOORBEELDEN*	*Assembleer-taal*		
Opdracht		*Adres ope-randum*	*Toelichting*	*Opdracht*		*Adres ope-randum*
Adres	*Code*			*Adres*	*Code*	
1000	10	0100	Begin programma in te zetten vanaf adres 100		BGN	100
1001	21	0506	Haal positief in accu A, constante nul		HPA	NUL
1002	31	0500	Breng schoon uit accu A (nul) naar adres 0500 resp. SOM07		BSA	SOM07
1003	31	0501	Breng schoon uit accu A (nul) naar adres 0501 resp. SOMREST		BSA	SOMREST
1004	11	1	Lees kaart in buffer 1 : 80 (vanuit lezer 1)	LEES	LSK	
1005	12	1 : 2	Haal in accu A en accu B positie 1 : 2 (code) van buffer		HAB	1 : 2
1006	32	0502	Haal positief uit accu A naar adres 0502 resp. COZZ		HPA	COZZ
1007	40	1023	Spring indien inhoud accu A = accu B naar adres 1023 resp. EINDE		SAB	EINDE
1008	32	0503	Haal positief in accu A (constante 07) van adres 0503 resp. CO7		HPA	CO7
1009	40	1015	Spring indien inhoud van accu A = accu B naar adres 1015 resp. TEL07		SAB	TEL07
1010	12	3 : 6	Haal in accu A en accu B (bedrag alfa-numeriek)		HAB	3 : 6

Machine-code			*PROGRAMMA-VOORBEELDEN*	*Assembleer-taal*		
Opdracht		*Adres ope-randum*	*Toelichting*	*Opdracht*		*Adres ope-randum*
Adres	*Code*			*Adres*	*Code*	
1011	09		Konverteer alfa-numeriek naar getal		KAG	
1012	45	0501	Tel op in accu B inhoud adres 0501 resp. SOMREST		OPB	SOMREST
1013	34	0501	Breng schoon uit accu B de inhoud naar adres 0501 resp. SOMREST		BSB	SOMREST
1014	41	1004	Spring altijd naar adres 1004, resp. LEES		SAL	LEES
1015	34	0900	Breng schoon uit accu B de inhoud naar adres 0900 resp. PR1	TEL07	BSB	PR1
1016	12	3 : 6	Haal in accu A en accu B positie 3 : 6 (bedrag)		HAB	3 : 6
1017	30	0901	Breng positief uit accu B naar adres 0901 resp. PR2 (printgebied)		BPB	PR2
1018	09		Konverteer van alfa-numeriek naar getal		KAG	
1019	45	0500	Tel op in accu B inhoud adres 0500		OPB	SOM07
1020	34	0500	Breng schoon uit accu B inhoud naar adres 0500 resp. SOM07		BSB	SOM07
1021	22		Druk af (inhoud printgebied)		DRU	
1022	41	1004	Spring altijd naar adres 1004 resp. LEES		SAL	LEES
1023	32	0504	Haal positief in accu A inhoud van adres 0504 resp. TOTAAL	EINDE	HPA	TOTAAL
1024	31	0900	Breng schoon uit accu A inhoud naar adres 0900 resp. PR1 (printgebied)		BSA	PR1
1025	29	0500	Haal positief in accu B inhoud van adres 0500 resp. SOM07		HPB	SOM07
1026	08		Konverteer getal in alfanumeriek		KGA	
1027	30	0901	Breng positief uit accu B naar adres 0901 resp. PR2 (printgebied)		BPB	PR2
1028	22		Druk af (inhoud printgebied)		DRU	
1029	32	0505	Haal positief in accu A inhoud adres 0505 resp. REST		HPA	REST
1030	33	0900	Breng schoon uit accu A naar adres 0900 resp. PR1 (woord REST naar printgebied)		BSA	PR1
1031	29	0501	Haal positief in accu B inhoud adres 0501 resp. SOMREST		HPB	SOMREST
1032	08		Konverteer getal in alfa-numeriek		KGA	

Machine-code			*PROGRAMMA-VOORBEELDEN*	*Assembleer-taal*		
Opdracht		*Adres ope-randum*	*Toelichting*	*Opdracht*		*Adres ope-randum*
Adres	*Code*			*Adres*	*Code*	
1033	30	0901	Breng positief uit accu B naar adres 0901 resp. PR2 (printgebied)		BPB	PR2
1034	22		Druk af (inhoud printgebied)		DRU	
1035	00		Stop programma		STP	
			Symbolische adressen voor totalen	SOM07		
				SOMREST		
			Symbolische adressen voor constanten	COZZ	'ZZ'	
				CO7	'07'	
				TOTAAL	'TOTAAL'	
				REST	'REST'	
				NUL	0	

Na de vertaling (assemblage) van de assembleer-taal wordt het programma in de machinetaal verkregen.

Accu - accumulator (rekenorgaan).

Haal positief in accu - het getal wordt positief ingebracht in tegenstelling tot negatief.

Breng schoon uit accu - de inhoud van de accu wordt overgebracht naar het vermelde adres, waarna de accu op nul wordt gesteld.

Constante - een gegeven op adres dat kan worden opgeroepen voor vergelijken of afdrukken.

Buffer - een hulpgeheugen of geheugenplaats waarin gegevens tijdelijk worden opgeslagen.

Konverteren - omzetten.

Programma-voorbeeld in COBOL

```
IDENTIFICATION DIVISION.  beperkt aangegeven.
PROGRAM-ID. 'VOORB1'.
ENVIRONMENT DIVISION.  niet nader uitgewerkt.
DATA DIVISION.
WORKING-STORAGE SECTION. (het gedefinieerde deel van het werkgeheugen)
01 KAARTIN.
   02 CODE      PICTURE XX.        (picture = verb)
   02 BEDRAG    PICTURE S9 (4).
   02 FILLER    PICTURE X(74).     (filler = blanco)
01 PRINT.
   02 CODEP     PICTURE X(6).
   02 BEDRAGP   PICTURE ZZZ,ZZ9.
   02 FILLER    PICTURE X(120)     VALUE SPACES.
77 SOM07        PICTURE S9(6)      VALUE 0.
77 SOMREST      PICTURE S9(6)      VALUE 0.
PROCEDURE DIVISION.
   OPEN INPUT KAART, OUTPUT PRINTER.
LEES.
   READ KAART INTO KAARTIN AT END GO TO EINDE.
   IF CODE = 'ZZ' GO TO EINDE.
   IF CODE = '07' ADD bedrag TO SOM07.
   MOVE CODE TO CODEP MOVE BEDRAG TO BEDRAGP,
   WRITE REGEL FROM PRINT AFTER 1 LINE,
   ELSE ADD BEDRAG TO SOMREST.
   GO TO LEES.
EINDE.
   MOVE 'TOTAAL' TO CODEP.
   MOVE SOM07 TO BEDRAGP.
   WRITE REGEL FROM PRINT AFTER 1 LINE.
   MOVE 'REST' TO CODEP.
   MOVE SOMREST TO BEDRAGP.
   WRITE REGEL FROM PRINT AFTER 1 LINE.
   CLOSE KAART, PRINTER.
   STOP RUN.
```

Na de vertaling (compilering) ontstaat het programma in de machinetaal.