BESTANDSORGANISATIE (a245)

1987-1988

Prof. ir. D.H. Wolbers

TECHNISCHE UNIVERSITEIT DELFT
Faculteit der Technische Wiskunde
en Informatica
Vakgroep Informatica
Julianalaan 132
2628 BL DELFT

INHOUDSOPGAVE

1. BASISBEGRIPPEN

1.1. Inleiding

Bij gebruik van de computer voor het oplossen van problemen ligt soms de nadruk op het benodigde proces. Dit is bij veel technische processen het geval waarbij dan de benodigde algorithmiek op de voorgrond staat. In andere gevallen en vooral bij het ontwerp en beheer van informatiesystemen is de organisatie van gegevens of data het primaire probleem. In die gevallen zal veelal het ontwerp beheerst worden door de organistaie van de data. Echter ook veel technische problemen worden vaak primair bepaald door de grote hoeveelheid data.

Deze gegevens of data hebben betrekking op of spelen een rol in een organisatie. Voorbeelden van dergelijke organisaties:

school
ziekenhuis
bank
bibliotheek

1.2. Objecten

In een organisatie onderscheidt men voor die organisatie relevante objecten. Deze objecten zijn onder te verdelen in een aantal objecttypen per organisatie.

Voorbeeld

Organisatie	Objecttypen
School	leerlingen
	docenten
	lokalen
	vakken

Organisatie	Objecttypen
Ziekenhuis	patienten
	personeel
	afdelingen
	apparatuur
	medicijnen
Bank	klanten
	personeel
	kantoren
	aandelen
	effecten
Bibliotheek	boeken
	klanten
	bestellingen

In deze voorbeelden zijn per organisatie slechts enkele objecttypen gegeven. Welke typen men wenst te beschouwen hangt af van de toepassing. Let ook op het onderscheid tussen objecttype en concrete objecten of objectexemplaren. Bijvoorbeeld bij een school is leerling een objecttype. Concrete objecten of objectexemplaren kunnen dan zijn de leerlingen Jansen, Pieterse, Klaassen, enz.

1.3. Kenmerken of attributen

Ieder objecttype bezit in de regel vele eigenschappen waarvan slechts een beperkt aantal voor de betrokken organisatie en toepassing van belang zijn.

Deze eigenschappen worden kenmerken of attributen van het objecttype genoemd.

Voorbeelden:

Objecttype	Kenmerken
Leerling	naam
	adres
	woonplaats
	klassenummer
	rapporten
Patient	naam, adres, woonplaats
	verzekering of ziekenfonds
	behandelend arts
Boek	titel
	schrijver
	catalogus nummer
	leennummer

Ook hier weer geen uitputtende lijst en ook in dit geval worden weer alleen die kenmerken beschouwd die van belang zijn voor de toepassing binnen de aangegeven organisatie.

Zoals we bij objecten onderscheid maken tussen typen en exemplaren hebben we hier het kenmerk en in een concreet geval de waarde van dat kenmerk.

Bijvoorbeeld voor een bepaald objectexemplaar van leerling is de waarde van het kenmerk naam "Jansen", het adres "Julianalaan 132", de woonplaats "Delft" en een klassenummer "3A".

1.4. Bestandsopbouw

Overeenkomstig deze opbouw van gegevens worden alle data die betrekking hebben op een organisatie opgeslagen in een of meer bestanden.
Een bestand is opgebouwd uit een of meer recordtypen. Een recordtype komt daarmee overeen met een objecttype, terwijl per objectexemplaar een record aanwezig is.

Een recordtype is weer samengesteld uit velden, waarbij een veld overeenkomt met een kenmerk. Voor een record is de inhoud van een veld de waarde van het bijbehorende kenmerk. In de beschrijving van een recordtype zal als naam van een veld veelal de benaming van het kenmerk worden gebruikt.

Voorbeeld

Een bestand "school" bevat recordtypen "leerling"; "docent"; "lokaal"; "vak".
Een recordtype "leerling" bevat de velden "naam"; "adres"; "woonplaats"; "klassenummer"; enz.
Een veld van een record moet uiteraard alle mogelijke waarden die dat bijbehorende kenmerk kan aannemen kunnen bevatten. Een veld bestaat daarom uit een aantal karakters. Ieder karakter weer uit een aantal bits. Bij moderne machines, zowel micro, mini, als grote computers tegenwoordig meestal 8 bits en dan oktade of ook wel byte genoemd.
Een veld kan dus een aantal alfanumerieke karakters bevatten maar ook zuiver numerieke waarden zijn mogelijk. Deze beslaan dan in de regel een aantal oktaden, b.v. 2,4 of 8 oktaden, overeenkomende met 16, 32 of 64 bits.
In de totale opbouw hebben we dus:

type	betrekking hebbend op
bit	
oktade	symbool of karakter
veld	kenmerk
record	object
bestand	organisatie

1.5. Bestandsorganisatie en databases

In deze geschetste bestandsopbouw ligt nog een aantal vrijheidsgraden. Verder is niet bepaald op welke media en op welke manier alle relevante data kunnen worden opgeborgen. Dit alles is het terrein van de bestandsorganisatie. De technieken die daarbinnen worden toegepast en de keuze uit deze technieken worden vooral bepaald door de eisen die gesteld worden aan het zoeken, wijzigen, weglaten en toevoegen van velden en vooral records.

Tussen records in een bestand en vaak ook tussen records van verschillende bestanden bestaan bepaalde relaties. In een schoolbestand bevat een leerlingrecord bijvoorbeeld gegevens voor klassedocent en verplichte vakken. Daarmee bestaat er verbinding met docentrecords en vakrecords. Soms wordt dit impliciet in een record opgenomen door in een extra veld een verwijzingsadres naar het betrokken andere record op te nemen. Bij het optreden van wijzigingen moeten dan eventueel ook dergelijke verbindingspointers worden meegewijzigd.

Naast dergelijke relaties kan het gemakkelijk voorkomen dat bepaalde gegevens identiek in verschillende recordtypen voorkomen. Omdat bij wijziging van deze gegevens technisch vrijwel nooit echt gelijktijdig de wijziging op alle plaatsen kan worden doorgevoerd bestaat het gevaar dat tijdelijk data in één of meer bestanden niet meer consistent zijn.

Naast relaties en redundantie bestaan er soms ook hiërarchische verbanden tussen bestanden. In een vorig voorbeeld werd school beschouwd als organisatie. Maar binnen een onderwijsbestand van een gemeente kan school weer als object optreden. Voor bepaalde toepassingen kan het daarmee nodig en nuttig zijn meerdere bestanden te koppelen en als een geheel te beschouwen. Toen dergelijke bestandssystemen ontstonden is men gaan spreken over databases of databanken. Om effectief met dergelijke systemen te kunnen werken is er echter meer nodig dan alleen het koppelen van conventionele bestanden. Daarmee heeft het gehele gebied van databeheer zich gesplitst in twee redelijk gescheiden delen. Enerzijds de vooral logische samenhang tussen grote aantallen data- en recordtypen. Dit is dan het gebied van de databases.

Daarnaast de techniek van het opslaan en terugvinden van records in relatief eenvoudige bestanden, de voor- en nadelen van verschillende opslagmethoden alsmede het numeriek schatten van zoek- en verwerkingstijden. Dit geeft men aan met bestandsorgainsatie en soms wel met "klassieke" bestandsorganisatie. Dit diktaat handelt verder alleen over bestandsorganisatie in deze zin.

1.6. Veldstruktuur

Zoals vermeld moet een veld de waarde van een kenmerk kunnen bevatten. Rekening houdend met de waarden die kunnen optreden moet het veld dus voldoende

karakters groot zijn. Het aantal karakters van een veld wordt wel de veldlengte genoemd. Nu kunnen we voor een bepaald veldtype van een recordtype een zodanige veldlengte kiezen dat die lengte ook voldoende is voor de grootste waarde die kan optreden. Bijvoorbeeld voor de veldlengte van het veld "naam" van het recordtype "leerling" nemen we 30 karakters. We gaan er dan vanuit dat geen namen van meer dan 30 karakters optreden. Verder bevat dan ieder record "leerling" een veld "naam" van 30 karakters, ook al worden die niet allemaal gebruikt indien de naam korter is.

Men spreekt in dit geval van een vaste veldlengte. Een andere methode is om binnen ieder record exemplaar "leerling" het "naam" veld een zodanige lengte te geven, dat de betrokken naam er precies in past. In dat geval is de veldlengte voor het "naam" veld binnen het recordtype "leerling" dus verschillend voor verschillende recordexemplaren.

We spreken dan van variabele veldlengte. Het systeem van variabele veldlengte is voordeliger uit een oogpunt van geheugengebruik. Immers er wordt geen geheugenruimte gereserveerd die niet benut wordt. Daar staat tegenover dat bij variabele veldlengte op een of andere wijze extra informatie nodig is die per geval de concrete lengte aangeeft. Deze complicatie en het goedkoper worden van geheugen hebben tot gevolg dat in verreweg de meeste gevallen de voorkeur gegeven wordt aan vaste veldlengte. De wijze waarop bij variabele veldlengte deze lengte wordt aangegeven hangt samen met de recordstruktuur.

1.7. Recordstruktuur

Een record bevat een of meerdere velden. Per recordtype kan ieder recordexemplaar in principe een verschillend aantal velden bevatten. Wel zal vrijwel altijd het maximum aantal velden vastliggen. Immers, het aantal kenmerken van een object dat relevant is voor de toepassing waarvoor het object is opgenomen, is begrensd en daarmee het aantal velden per recordtype. Naast een variabel aantal velden kunnen ook een of meer velden variabel van lengte zijn.

Onder de recordlengte verstaan we het aantal karakters dat nodig is voor alle velden van het record tesamen. Het is duidelijk dat deze recordlengte door een of twee van voornoemde redenen variabel kan zijn.

Variabele recordlengte kan op verschillende manieren gerealiseerd worden. De voornaamste 3 vormen zijn als volgt. Aan het begin van het record wordt een index opgenomen die verwijst naar het begin van de verschillende velden. Het aantal velden is wel vast, alleen de veldlengte per veld kan vrij gekozen worden. Ieder veld kan via de index direct bereikt worden. Een tweede methode is het aanbrengen van een scheidingssymbool tussen de velden. Als scheidingssymbool moet een karakter gekozen worden dat uiteraard binnen een veld niet kan voorkomen. Het aantal velden is vast en ook de volgorde is altijd hetzelfde. Nadeel van deze methode is dat wanneer een verder gelegen veld nodig is men het hele record karakter voor karakter moet aflopen.

Een derde methode is het gebruik van labels. Daarbij wordt ieder veld voorafgegaan door een karakteriserende label. Men is nu vrij in het aantal velden en ook de volgorde is irrelevant. Bij opzoeken van een bepaald veld geldt hetzelfde nadeel als bij de vorige methode in nog iets versterkte mate.

Tegenover de variabele recordlengte staat het record met vaste indeling zowel wat betreft aantal velden, de volgorde van velden als wel een vaste veldlengte per veld. Ieder record van hetzelfde type heeft dan een identieke indeling. Hoewel qua geheugengebruik onvoordeliger is dat het meest gebruikte type. De toegankelijkheid van ieder veld binnen een record is hier optimaal.
Een voorbeeld van de 3 methoden met variabele recordlengte en de vaste recordlengte is hierna gegeven aan de hand van de volgende data:

benaming:	tafel	ligstoel	kast
kleur :	wit	rood	blank
prijs :	335,00	121,50	1450,50

```
                    1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2
```

met indexen

```
3 8 11t a f e l w i t 3 3 5 , 0 0
3 11151 i g s t o e l r o o d 1 2 1 , 5 0
3 7 12k a s t b l a n k 1 4 5 0 , 5 0
```

met scheidingssymbolen

```
                    1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 3 3 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2

T A F E L # W I T # 3 3 5 , 0 0 # ;
L I G S T O E L # R O O D # 1 2 1 , 5 0 # ;
K A S T # B L A N K # 1 4 5 0 , 5 0 # ;
```

met labels

```
                    1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 3 3 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2

N M # T A F E L # K L # W I T # P R # 3 3 5 , 0 0 # ;
K L # R O O D # N M # L I G S T O E L # P R # 1 2 1 , 5 0 # ;
P R # 1 4 5 0 , 5 0 # K L # B L A N K # N M # K A S T # ;
```

vast

```
T A F E L                 W I T           3 3 5 , 0 0
L I G S T O E L           R O O D         1.2 1 , 5 0
K A S T                   B L A N K   1 4 5 0 , 5 0
|------naam-------------|----kleur----|-----prijs-----|
0                      1314          2021            27
```

Bij de methode met indexen is aangenomen dat het record niet meer dan 255 karakters lang is zodat het adres binnen het record met 8 bits binair kan worden aangegeven. Voor grotere records zal men overeenkomstig per indexplaats meer bits nodig hebben. Zoals al uit het voorbeeld is te zien is de eerste index hier altijd 3. Men zou die dus kunnen weglaten. Tegelijk geeft dit getal het aantal velden in het record aan. Daarmee kan men met deze methode ook nog werken met variabel aantal velden mits de weggelaten velden altijd de laatsten zijn.

Bij de methode met scheidingssymbolen en bij die met labels is "#" gebruikt als scheidingssymbool en ";" als eindmarkering van het record. In de praktijk zal men hiervoor een vrije bitcombinatie kiezen die in de normale karakterset niet gebruikt wordt. Bij de methode met labels is ook aangegeven dat de volgorde van de velden binnen het record willekeurig is.

Tenslotte bij de vaste indeling zijn hier 14, 7 en 7 karakters gekozen voor de veldlengten van respectievelijk naam, kleur en prijs. Voor naam en kleur is aangenomen dat de waarde links aangesloten en voor prijs rechts aangesloten is.

1.8. Bestandsstruktuur

Een aantal (in de regel groot aantal) records al of niet van één type kan worden samengevoegd tot een bestand. Een dergelijke verzameling records kan dan in één geheugengebied worden ondergebracht bijvoorbeeld op schijf- of magneetbandgeheugen. De wijze waarop dit gebeurt is de primaire problematiek van de bestandsorganisatie en komt daarmee in de volgende hoofdstukken aan de orde.

1.9. Sleutels

In veel toepassingen is het nodig uit een verzameling records, dus een bestand, één bepaald record op te zoeken. In feite zoekt men dan naar een objectexemplaar waarvan één of meerdere kenmerken een bepaalde waarde hebben.

Het kenmerk of de vereniging van die kenmerken die ieder objectexemplaar binnen de totale verzameling objecten eenduidig bepalen noemt men de sleutel. Bij records over personen kan bijvoorbeeld het veld "naam" gebruikt worden als sleutel mits in het gehele bestand geen personen met dezelfde naam voorkomen. Zou dit laatste soms wel het geval zijn, dan kan mede het veld "geboortedatum" gebruikt worden. De concatenatie van de twee velden "naam" en "geboortedatum" vormt dan de sleutel.

Soms kunnen verschillende velden als sleutel gebruikt worden. Bijvoorbeeld in het geval van persoonsrecords zou een van de velden het "persoonsnummer" kunnen bevatten. In principe kan men een bepaald record dan terugvinden zowel op "naam" als op "persoonsnummer". Beide velden kunnen alternatief als sleutel gebruikt worden. Echter in veel gevallen zal binnen het bestand een bepaalde volgorde voor de records gekozen zijn en wel op basis van een sleutel. Terugzoeken op basis van die sleutel zal dan eenvoudiger en sneller

zijn dan op basis van een andere sleutel. Vergelijk een telefoonboek. Abonnées zijn daarin gerangschikt volgens "plaatsnaam" en "persoonsnaam". Deze vereniging vormt dan de primaire sleutel. Echter ook het telefoonnummer is in feite een sleutel op het abonnébestand en zelfs het precies adres vormt een derde sleutel. In het gebruikelijke telefoonboek als bestand zijn deze sleutels echter zeer inefficiënt. Een telefoondienst zal in de regel wel over bestanden beschikken die op basis van de tweede en derde sleutel zijn georganiseerd, zodat terugzoeken op basis van die sleutel dan wel snel kan plaatsvinden. In feite heeft men dan de beschikking over 3 bestanden die ieder dezelfde records bevatten. Dit is tevens een voorbeeld van eerder genoemde redundantie. Wijzigingen in het abonnementsbestand moeten dan op 3 plaatsen worden doorgevoerd. Een sleutel bepaalt dus eenduidig slechts één record binnen een bestand.

Soms zoekt men in een bestand naar alle recordexemplaren waarvan een of meerdere velden een bepaalde waarde hebben. Bijvoorbeeld in een bestand over personen zoekt men naar alle personen die in een bepaald jaar geboren zijn. Het geboortejaar is dan het zoekargument. Aan die vereiste kunnen uiteraard meerdere records voldoen. Soms wordt in de literatuur ook voor dit geval het woord sleutel gebruikt en spreekt men ook wel over "duplicaatsleutels". In dit diktaat zullen we het woord sleutel uitsluitend gebruiken in de eenduidige betekenis.

1.10. Bestandseigenschappen

Bestanden kunnen op veel manieren worden georganiseerd die ieder voor zich bepaalde voor- en nadelen hebben. In samenhang daarmee worden vaak de volgende begrippen gebruikt.

- Bestandsgroei. Hiermee geeft men aan, meestal in %, hoeveel een bestand in een bepaalde tijdsfase toeneemt. Bijvoorbeeld 10%/jaar.

Bij veel toepassingen komt het voor dat m.b.v. een bepaald programma het gehele bestand wordt gebruikt. Laten we een dergelijk programma eenmaal draaien dan kunnen we de volgende grootheden onderscheiden.

- Bestandsactiviteit. Dit is het percentage records van het totale bestand dat in één run wordt gebruikt.
- Bestandsverandering. Dit is het percentage records dat in één run in één of meer velden wordt gewijzigd. Bestandsverandering is dus kleiner of gelijk aan de bestandsactiviteit.
- Bestandsverloop. Het percentage records dat in één run wordt verwijderd en vervangen door nieuwe records.

De hier gegeven definities worden helaas niet altijd uniform in de literatuur gebruikt. Zo wordt onder bestandsactiviteit soms verstaan datgene wat hier onder bestandsverandering is aangegeven en met bestandsverandering zowel de wijzing in velden van records als de complete vervanging.

Het bestandsverloop wordt in plaats van in percentage records per run ook vaak uitgedrukt in percentage records per tijdseenheid. Bijvoorbeeld 5%/ jaar.

Bij iedere run van een programma ondergaan dus een aantal records een verandering. Men spreekt dan van het aanbrengen van mutaties op records. In veel toepassingen komt het voor dat bepaalde records veel mutaties ondergaan terwijl anderen zeer weinig muteren. Bijvoorbeeld banken kennen klanten met veel bij- en afschrijvingen en anderen met weinig mutaties. Bekend is in dat verband de zogenaamde "80-20 regel". Hiermee wordt bedoeld dat gemiddeld bij iedere run 80% van alle mutaties betrekking hebben op 20% van de records. Zelfs van alle mutaties op de genoemde 20% records heeft weer 80% betrekking op 20% van die 20%, enz. Dit betekent dus dat 64% van alle mutaties betrekking heeft op 4% en 51% van alle mutaties op 0,8% van alle records.

Natuurlijk komen ook wel verhoudingen voor van 70-20 of 80-30 enz. Soms worden dit soort verhoudingen in de literatuur ook wel aangegeven met het begrip bestandsactiviteit. Een dergelijke verhouding is ook van belang voor de keuze van bestandsorganisatie. Immers, het gemakkelijker bereikbaar zijn van frequent gebruikte records heeft dan voordelen.

1.11. Records en blokken

Een record is de logische vereniging van gegevens die betrekking hebben op

een object en wordt als zodanig als een geheel beschouwd. Bij het opbergen op achtergrondgeheugen zoals magneetschijf en magneetband wordt veelal een andere opbergeenheid gebruikt, meestal gebaseerd op de physische eigenschappen van het medium. Een dergelijke eenheid noemen we een blok. Soms ook wel abusievelijk genoemd physiek record (physical records). Een blok bij magneetschijf is vaak een spoor of bij sectorindeling een sector.

Het aantal karakters in een blok noemen we weer de bloklengte. De bloklengte is in de meeste gevallen veel groter dan de recordlengte zodat er dan een aantal records passen in een blok. Bij magneetbanden kunnen we de bloklengte meestal vrij kiezen en maken die dan een geheel veelvoud van de recordlengte. Dit veelvoud wordt genoemd de blokkingsfactor. Bij schijven is zoals gezegd de bloklengte physiek bepaald en dan in de regel niet een geheel veelvoud van de recordlengte. Men zou dan een gebroken getal als blokkingsfactor kunnen gebruiken, maar dat betekent dat bepaalde records over 2 sporen verdeeld worden. In de regel kiest men een geheel getal als blokkingsfactor en laat de overblijvende ruimte per blok onbenut. Men spreekt dan wel over een multi-record blok. Dit laatste in tegenstelling tot het geval van zeer grote records, waarbij de recordlengte veel groter is dan de bloklengte. In dat geval moet een record over meerdere blokken verdeeld worden. Men spreekt dan van een multi-blok record. Men moet ook hier weer kiezen of men een geheel aantal blokken per record kiest of een gebroken getal, hetgeen impliceert dat sommige blokken dan gegevens bevatten van meer dan 1 record en bovendien dat vele records ergens midden in een blok beginnen.

Overigens wordt de grote recordlengte vaak veroorzaakt doordat bijvoorbeeld documentatie in de vorm van veel tekst van variabele lengte deel uitmaakt van het record. In dat soort gevallen is ook de recordlengte variabel. Bij keuze van een geheel aantal blokken/records blijft dan gemiddeld een half blok per record onbenut. In zulke gevallen zal men de voorkeur geven aan gebroken getallen voor de verdeling van de records over de blokken.

2. LIJSTSTRUKTUREN

2.1. Principe van lijststrukturen

De basis van iedere lijststruktuur is de adresverwijzing. Daarbij is de plaats van een record in eerste instantie niet van belang. Voor alle situaties waarbij enige relatie gelegd moet worden tussen records kunnen we één of meer extra velden in een record opnemen. Ieder zo'n dergelijk veld kan dan een adres bevatten dat verwijst naar de plaats van een ander record. Deze verwijzing kan in principe plaatsvinden naar andere records in hetzelfde bestand maar ook naar records in een ander bestand. Naast de velden nodig voor het vastleggen van de waarden van kenmerken van een object kunnen dus speciale "adres velden" voorkomen die vaak ook pointers of wijzers worden genoemd.

Doordat records steeds naar volgende records kunnen verwijzen is het mogelijk gehele ketens of kettingen van records te vormen. Iedere stap in zo'n keten is dan een volgende adresverwijzing. Vooral wanneer de records zich verspreid op een achtergrondgeheugen b.v. schijfgeheugen bevinden kan dit met een grote tijd gepaard gaan. Bestandsorganisaties zullen daarom zelden compleet op dit soort strukturen gebaseerd worden. Voor delen van een bestand, vooral wanneer die in zijn geheel tijdelijk in primair geheugen gebracht kunnen worden, kan deze methode echter zeer flexibel en handig zijn. Bij de implementatie van moderne database management systemen wordt deze methode van adresverwijzing vaak toegepast, juist wegens de flexibiliteit van het aanbrengen van relaties tussen records. Voor de gebruiker van deze systemen blijven dit soort adresverwijzingen dan verborgen en kent hij per recordtype alleen de velden die betrekking hebben op de kenmerken. Vooral bij objecten met relatief weinig kenmerken maar met relatief veel relaties kan een corresponderend record daardoor veel groter zijn dan de gebruiker denkt.

2.2. Soorten lijststrukturen

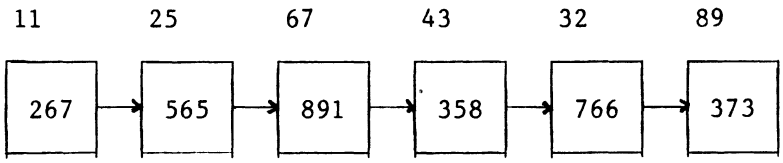
Lijststrukturen kunnen we in klimmende volgorde van gecompliceerdheid in drie hoofklassen verdelen, namelijk:

- lineaire strukturen
- bomen
- netwerken

2.2.1. Lineaire strukturen

Bij lineaire strukturen zijn alle records achter elkaar geschakeld in de vorm van een ketting. Er is daarbij duidelijk sprake van een volgorde, waarbij in het midden wordt gelaten op grond waarvan elke volgorde is gekozen. De eenvoudigste vorm van de lineaire struktuur is daarbij de open ketting met enkelvoudige verwijzing.

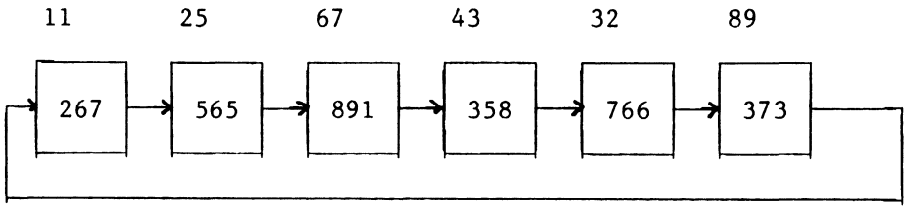
Bijvoorbeeld 6 records met sleutels 267, 565, 891, 358, 766 en 373 die geplaatst zijn op de adressen 11, 25, 67, 43, 32 en 89. We kunnen dit als volgt weergeven:



Ieder record heeft dus één extra adresveld. Zo bevat het adresveld van het record met sleutel 267 de waarde 25 en verwijst daarmee naar het record met sleutel 565 enz.
Bij deze enkelvoudige open ketting zijn twee bijzondere verwijzingen. In de eerste plaats moet men het adres van het eerste record, in dit geval 11, ergens expliciet bewaren. Verder kan het laatste record in de ketting nergens naar verwijzen. Omgekeerd als men de ketting systematisch afloopt en zoekt naar een record dat er niet is moet men aangekomen bij het laatste record kunnen constateren dat er geen verdere records meer zijn. Daarom vult men het adresveld van het laatste record met een waarde die als adres nooit kan voorkomen, bijvoorbeeld -1. Echter in principe is men vrij in de keuze van de waarde. In de literatuur en in sommige programmeertalen wordt deze speciale waarde vaak met NIL aangegeven.

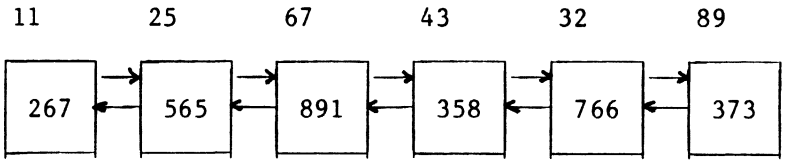
Wanneer er toch geen sprake is van een bepaalde volgorde en de ketting alleen dient om de betrokken records tot een geheel te verenigen heeft men

geen behoefte aan een expliciet eerste en laatste record. We gebruiken dan de enkelvoudige gesloten ketting ook wel genoemd enkelvoudige ring. Met hetzelfde voorbeeld

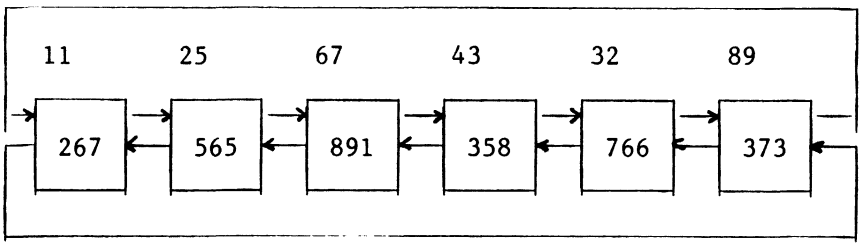


Er is nu geen NIL verwijzing meer. Er is ook geen expliciete eerste meer. Voor verwijzing naar deze serie records is het voldoende één der adressen expliciet elders te onthouden.

Zowel bij de open als gesloten enkelvoudige ketting kan men de records slechts in één richting achtereenvolgens aflopen. Bij sommige toepassingen kan het wenselijk zijn in twee richtingen de ketting te kunnen doorlopen. Daarmee ontstaan de open en gesloten dubbele ketting. Met hetzelfde voorbeeld



en

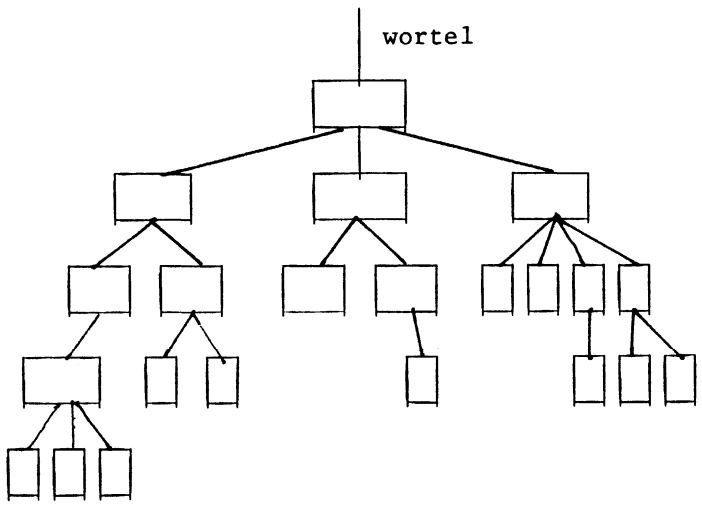


Hoewel dit systeem voor sommige bewerkingen voordeel kan hebben betekent het wel dat ieder record nu twee extra adresvelden nodig heeft. Wanneer een record dus in meerdere kettingen wordt opgenomen betekent het één extra adresveld voor iedere ketting en iedere verwijzingsrichting.

In paragraaf 2.3 en 2.4 worden voorbeelden gegeven van dergelijke lineaire strukturen.

2.2.2. Boomstrukturen

Bij boomstrukturen is er sprake van een hiërarchie van records. Met uitzondering van één record dat de wortel vormt heeft ieder record één voorganger en 0,1 of meer opvolgers. De wortel heeft geen voorganger maar wel 0,1 of meer opvolgers. Bijvoorbeeld



Het is gebruikelijk dergelijke bomen te tekenen met de wortel boven en daaronder de verschillende knopen. Soms noemt men de eindknooppunten, die geen opvolgers meer hebben wel de bladeren. In dit voorbeeld zijn geen sleutels of adressen aangegeven aangezien hier in het midden is gelaten op welke gronden deze struktuur tot stand is gekomen. Voor algemene bomen is niets voorgeschreven omtrent het aantal opvolgers per knoop en ook niets omtrent de hoogte of diepte van de boom via verschillende takken.

Een verwijzing naar een totale boomstruktuur kan dus plaatsvinden door het adres van de wortel. Via de wortel kan men alle verdere records bereiken. De opbouw van de boom zal altijd via een of andere logische struktuur plaatsvinden. Deze struktuur wordt dan gebruikt voor het zoeken naar een bepaald exemplaar.

Er zal in het algemeen naar gestreefd worden een boom zo gebalanceerd

mogelijk op te bouwen. Daarmee wordt bedoeld de afstand van alle "bladeren" tot de "wortel" ongeveer gelijk te krijgen Ieder exemplaar in de boom kan dan met een beperkt aantal stappen bereikt worden. Vooral bij een groter aantal vertakkingen per knoop groeit een boom zeer snel.

Hebben we per knoop k opvolgers dan kunnen we het aantal knopen voor een "volle" boom met hoogte h gemakkelijk berekenen. Met een volle boom wordt dan bedoeld dat alle knopen het maximum aantal opvolgers hebben en alle bladeren zich op dezelfde hoogte h bevinden.

nivo	aantal
1	1
2	k
3	k^2
4	k^3
.	
.	
.	
i	k^{i-1}
.	
.	
.	
h	k^{h-1}

$$n = \sum_{i=1}^{h} k^{i-1} = \sum_{i=0}^{h-1} k^{i} = \frac{k^{h}-1}{k-1}$$

Enkele voorbneelden

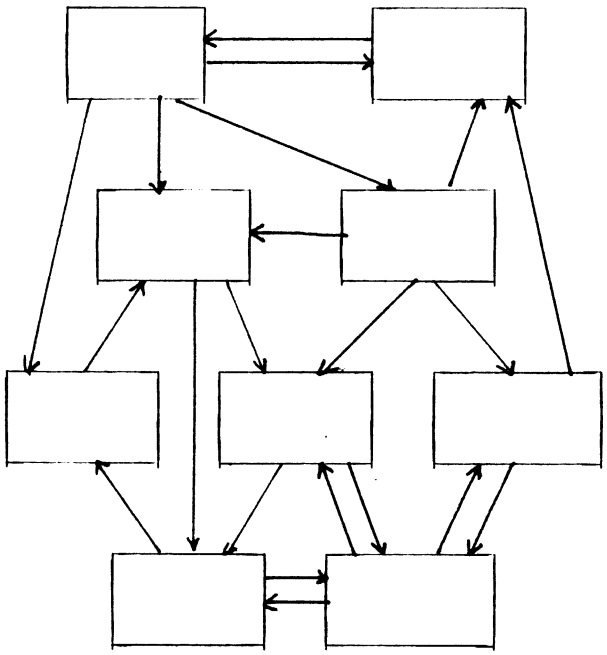
k	h	n
5	10	2 441 406
20	6	3 368 421
50	5	6 377 551
100	5	101 010 101

Vooral bij het laatste voorbeeld zien we dat met 5 stappen men een bereik heeft van meer dan 100M. Op bomen met dergelijke eigenschappen komen we uitvoerig terug in hoofdstuk 7. We vermelden hier nog een bijzonder type boom namelijk die bomen waarbij het maximum aantal opvolgers per knoop 2 is. Dit noemt men binaire bomen en deze worden besproken in hoofdstuk 5.

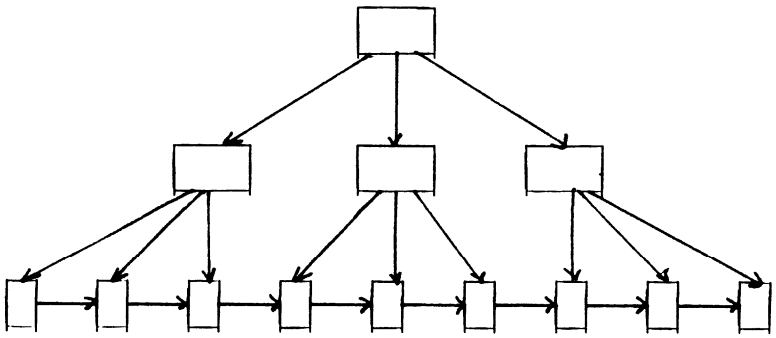
Ten slotte nog een opmerking over de adresvelden in bomen. In het getekende voorbeeld van een boom aan het begin van deze paragraaf zijn geen spijlen aangegeven tussen de knopen van de bomen, maar alleen de logische samenhang. In de regel zijn minstens adresverwijzingen van wortel richting bladeren, dus in iedere knoop adresverwijzingen naar de directe opvolgers. Soms neemt men ook adresverwijzingen op naar de directe voorgangers. Die keuze voor al of niet expliciete terugverwijzingen hangt af van het gebruik dat men van de boom moet maken. Naast terugverwijzingen worden soms ook nog andere verwijzingen opgenomen. Bijvoorbeeld kan het voorkomen dat men alle bladeren nog weer eens in een lineaire struktuur opneemt of dat ieder record een directe terugverwijzing heeft naar de wortel. In het algemeen is daarvoor geen richtlijn te geven. Meer verwijzingen kosten meer geheugenruimte maar versnellen in de regel bepaalde processen. Per toepassing zal men dit tegen elkaar moeten afwegen.

2.2.3. Netwerkstruktuur

Netwerkstrukturen vormen een logisch vervolg op de vorige twee strukturen. Bij lineaire strukturen hadden we één voorganger en één opvolger Bij bomen één voorganger en meerdere opvolgers. Bij netwerken is geen beperking meer en kan ieder record in principe meerdere voorgangers en meerdere opvolgers hebben. Ieder record kan daarbij in principe naar ieder ander record verwijzen en ook ten aanzien van "heen" en "terug" is geen regel te geven. Een willekeurig voorbeeld.



Bij de implementatie van database systemen waarbij vooral veel dwarsrelaties tussen allerlei soorten recordtypen gelegd moeten worden komen dergelijke netwerkstrukturen wel voor. Bij de klassieke bestandsorganisatie is dit veel minder het geval hoewel bepaalde gevallen strikt genomen wel als netwerk zouden moeten aangemerkt.
Bijvoorbeeld het in de vorige paragraaf genoemde geval van een boom waarbij de bladeren in een lineaire struktuur zijn opgenomen.



Men moet bij algemene netwerkstrukturen nog onderscheid maken tussen systemen waarbij de struktuur tussen recordtypen vastligt en daarmee voor alle recordexemplaren gelijk is en systemen waarbij de struktuur per record exemplaar nog kan verschillen. In het eerste geval is per recordtype dan het

aantal adresvelden gefixeerd door een schema zoals bijvoorbeeld aan het begin van deze paragraaf. In het tweede geval betekent het dat het aantal adresvelden per recordtype variabel is, met misschien een maximum. Maakt men in zulke gevallen dan gebruik van records met variabele lengte dan kan het voorkomen dat door een mutatie een record met een extra adresveld moet worden uitgebreid. Daardoor heeft het meer ruimte nodig en kan dan misschien niet meer op de oude plaats in het achtergrondgeheugen worden teruggezet. Door de lijststruktuur is dat op zich geen bezwaar omdat iedere plaats relatief even goed is. Wel moet er aan worden gedacht dat alle adresverwijzingen naar dit record die naar de oude plaats verwijzen moeten worden aangepast aan de nieuwe situatie. Daartoe moet men weer weten welke records naar dit record verwijzen. Dit laatste betekent praktisch dat iedere relatie tussen twee records dan ook dubbel, dus met heen- en terugverwijzing, moet worden uitgevoerd.

Dergelijke netwerkstrukturen worden in dit diktaat verder niet meer expliciet behandeld.

2.3. Ongeordende lineaire lijststukken

In een voorbeeld geven we een bestand gebaseerd op een open enkelvoudige lineaire struktuur. Alle records bevinden zich in een open ketting. Nieuwe records worden aan het begin van de ketting toegevoegd, d.w.z. de volgorde in de ketting is dus een chronologische volgorde met het laatst toegevoegde record voorop. Bij verwijderen van een record wordt eenvoudig de ketting over het betrokken record heen kortgesloten. We nemen aan dat het beschikbare geheugengebied voor dit bestand op achtergrondgeheugen relatief genummerd is van 1 t/m M. Door regelmatig toevoegen en verwijderen van records kan ieder record in principe op ieder adres tussen 1 en M staan. Om bij toevoegen van een record te weten welke plaatsen nog vrij zijn moeten we een registratie van vrije adressen bijhouden. Dit gebeurt door alle vrije adressen te ketenen in een lineaire ketting.

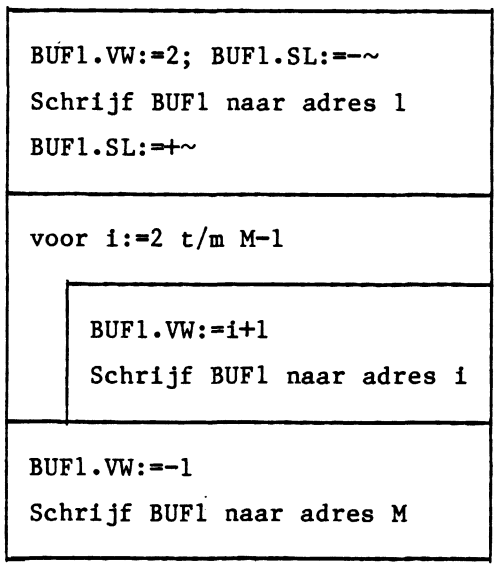
Bij toevoegen van een record in een open ketting moet veelal een afwijkende procedure gevolgd worden indien het record toevallig op de eerste of laatste plaats wordt ingevoegd. Daarnaast moet ook een afwijkende procedure gevolgd worden indien de ketting nog leeg is.

Dit soort moeilijkheden is te vermijden door van de beschikbare M geheugenplaatsen twee af te zonderen en die te gebruiken voor dummy records. Deze twee dummies vormen dan het begin en einde van een in feite lege ketting. Het toevoegen van een record is nu onder alle omstandigheden het toevoegen in een bestaande keten. Voor deze twee dummy records gebruiken we de adressen 1 en 2.

We nemen aan dat ieder record een vaste recordindeling heeft. Twee velden komen in dit voorbeeld expliciet naar voren. Ten eerste het sleutelveld dat we aangeven met SL en het adresverwijzingsveld VW. Bij het lezen en wegschrijven van records maken we gebruik van buffers die we aangeven met BUF1, BUF2 en BUF3.

Voor we de eerste records in het bestand brengen moeten we de geheugenadressen 1 en 2 initialiseren als dummy records en verder alle verdere geheugenplaatsen in een lange vrijketen bundelen.
Dit alles wordt gedaan volgens het volgende struktuurdiagram INITIALISEREN.

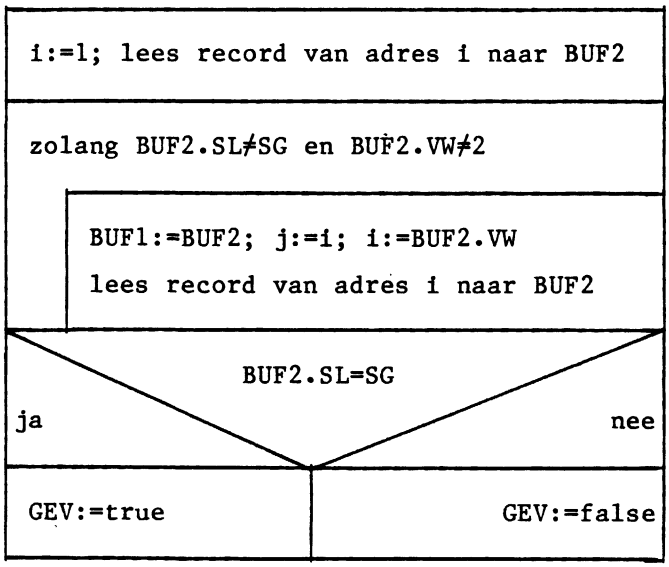
INITIALISEREN



Als NIL indicator gebruiken we hier -1. Het begin van de "vrijketen" wordt vastgelegd in het verwijzingsveld van dummy record 2. Het invullen van de sleutelvelden heeft voor dit voorbeeld geen betekenis. In de volgende paragraaf behandelen we echter hetzelfde probleem met een geordend bestand en kunnen dan ditzelfde programma gebruiken.

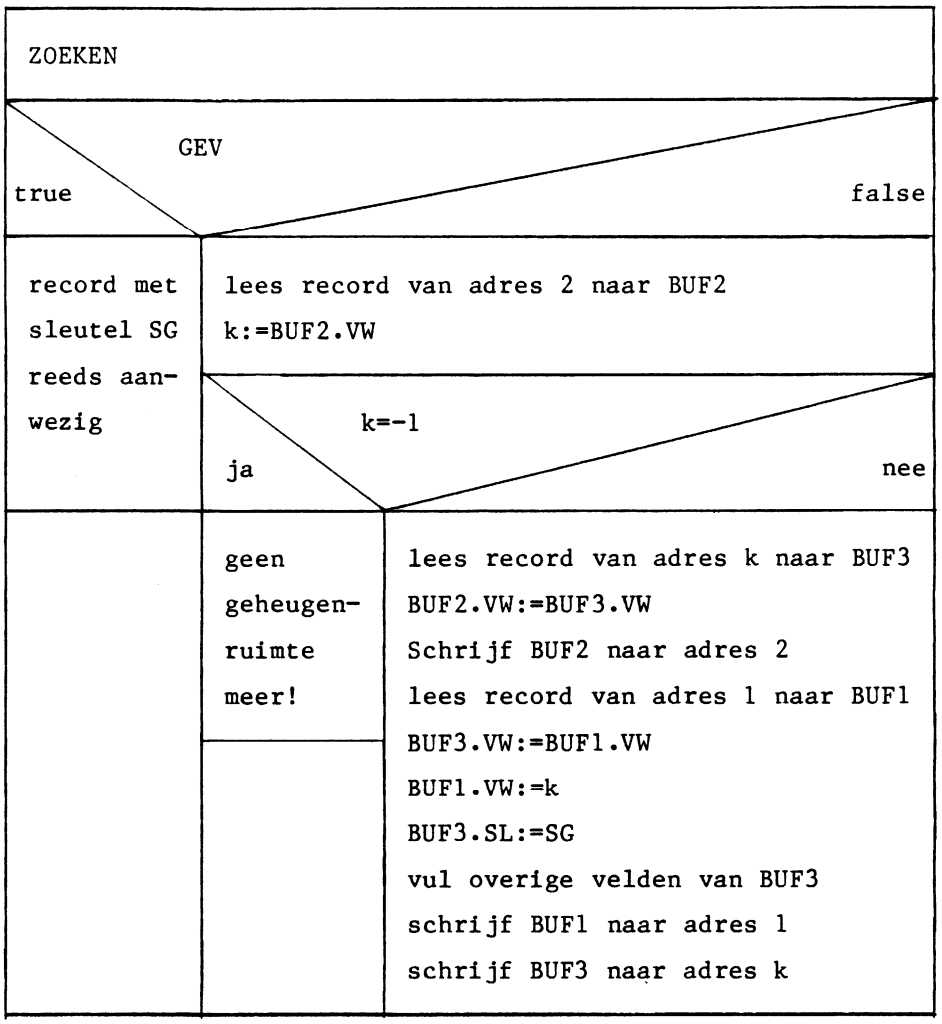
We willen nu het probleem behandelen van toevoegen en verwijderen van records. Bij toevoegen willen we vermijden dat twee records met dezelfde sleutel worden opgenomen en zoeken daarom eerst naar een record met de toe te voegen sleutel, die dan niet aanwezig mag zijn.
Bij verwijderen van een record moeten we eerst de plaats weten waar het record staat. In beide gevallen gebruiken we daarom eerst de routine ZOEKEN. In deze routine wordt gezocht naar het record met sleutel SG. In verband met de enkelvoudige ketting moeten we voor verwijderen van een record ook het adres van het voorgaande record weten. Dit wordt ook door de routine ZOEKEN bepaald.

ZOEKEN



Het toevoegen van een nieuw record gaat daarmee met programma TOEVOEGEN.

TOEVOEGEN



Het nieuwe record wordt dus vooraan in de keten toegevoegd. Verwijderen van een record gaat met het programma VERWIJDEREN.

VERWIJDEREN

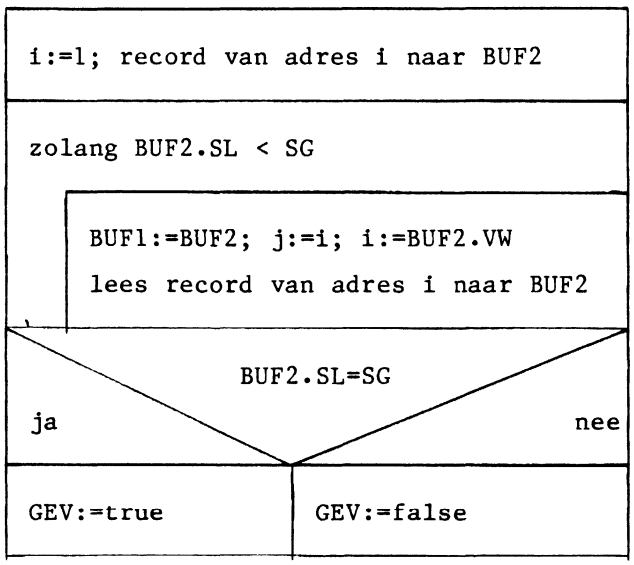
ZOEKEN	
GEV: false	GEV: true
record met sleutel SG niet aanwezig	BUF1.VW:=BUF2.VW schrijf BUF1 naar adres j lees record van adres 2 naar BUF1 BUF2.VW:=BUF1.VW BUF1.VW:=i schrijf BUF1 naar adres 2 schrijf BUF2 naar adres i

De vrijketen wordt dus gebruikt op basis LIFO (last in first out). Zou men uit veiligheidsoverwegingen verwijderde records nog zo lang mogelijk willen bewaren dan moeten verwijderde records achteraan in de vrijketen worden toegevoegd terwijl ruimte voor nieuwe records dan vooraan uit de vrijketen moet worden genomen of eventueel precies omgekeerd. In beide gevallen is dan het gebruik op basis FIFO (first in first out).

2.4. Geordende lineaire lijststruktuur

Dit voorbeeld lijkt veel op het vorige. Het verschil is dat de records in het bestand nu gerangschikt zijn volgens opklimmende waarde van de sleutel. Bij het zoeken naar een record is alleen het criterium voor beëindiging van de zoeklus iets anders.

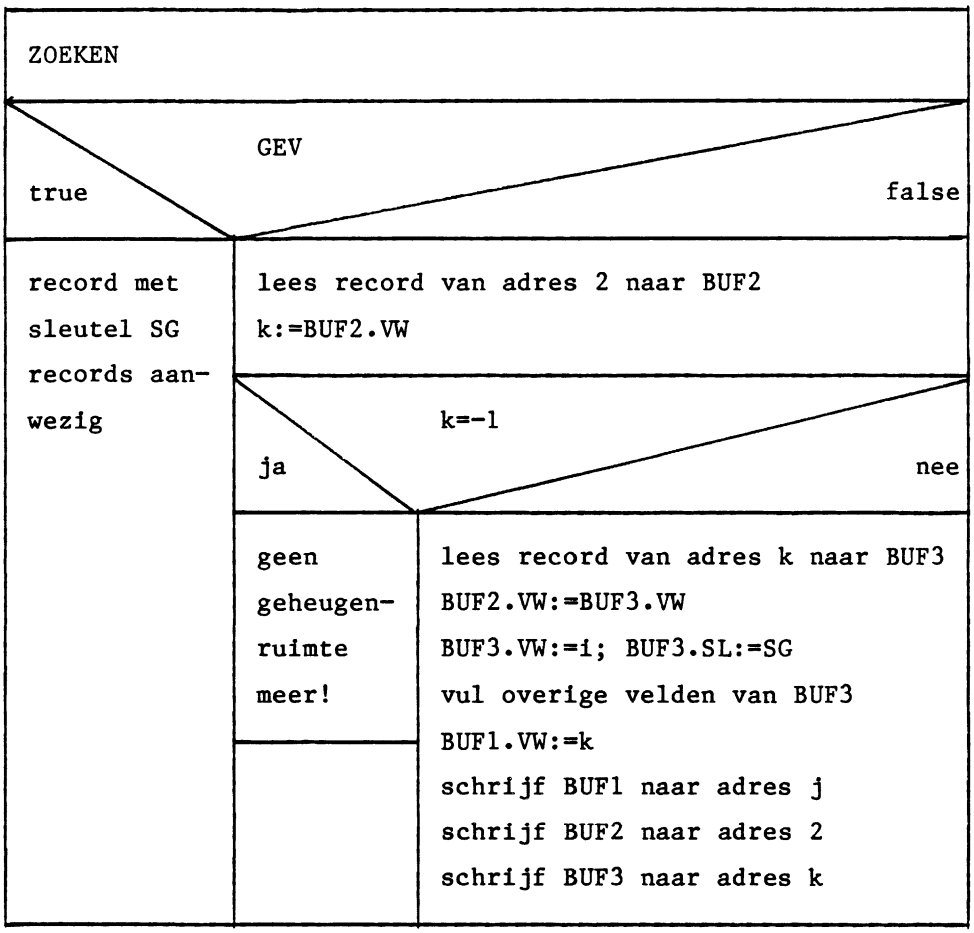
ZOEKEN



We maken hierbij gebruik van het feit dat de sleutelwaarden van dummy records 1 en 2 respectievelijk -~ en +~ zijn. De waarden -~ en +~ staan hier symbolisch voor sleutelwaarden die respectievelijk kleiner en groter zijn dan enige voorkomende sleutel. In COBOL worden daarvoor de waarden LOW-VALUE en HIGH-VALUE gebruikt.
Door deze fictieve sleutelwaarden wordt de zoeklus minstens eenmaal doorlopen en in ieder geval beëindigd.

Bij toevoegen van een nieuw record met sleutel SG moet op de juiste plaats in de keten worden ingevoegd.

TOEVOEGEN



Bij verwijderen van records is voor dit geval geen verschil met het voorbeeld van ongeordende struktuur. Voor verwijderen kan dan ook van dezelfde routine VERWIJDEREN gebruik worden gemaakt met dien verstande dat de verwijzing naar subroutine ZOEKEN dan wel betrekking heeft op de in deze paragraaf besproken routine ZOEKEN.

Om misverstanden te voorkomen, de voorbeelden in par. 2.3 en par. 2.4 lenen zich niet voor grote bestanden omdat dan teveel toegangen tot achtergrondgeheugen benodigd zijn. Op delen van een bestand worden deze methoden echter wel gebruikt. Moet men veel bewerkingen van zoeken, toevoegen en verwijderen na elkaar doen dan zijn deze processen iets te versnellen door de dummyrecords niet ieder naar achtergrondgeheugen terug te schrijven en weer op te halen. Wel moeten aan het einde alle records weer teruggeschreven worden. Het gevaar is dan overigens dat tijdens het proces de records 1 en 2 op achtergrondgeheugen niet meer de juiste informatie bevatten.

3. SEQUENTIELE BESTANDSORGANISATIE

3.1. Organisatievorm

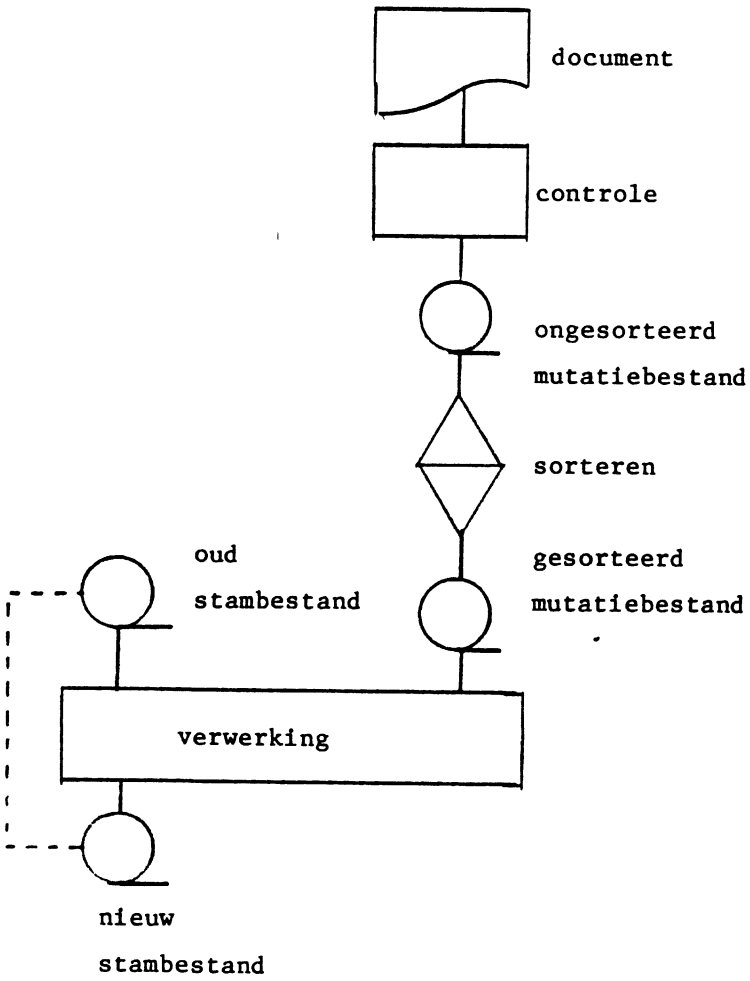
3.1.1. Principe

Bij deze organisatievorm staan alle records direct achter elkaar maar wel op sleutelvolgorde. Vooral magneetbanden lenen zich goed voor deze organisatievorm. Variabele lengte records vormen daarbij geen bezwaar. De records hebben geen vaste plaats, alleen hun relatieve volgorde is bepalend voor de physieke locatie. Bij sequentiële bestanden spelen adressen geen rol. Een willekeurig record is dus niet direct bereikbaar maar moet worden opgezocht door het gehele bestand van begin af te doorlopen tot men het gevonden heeft. De werkwijze met sequentiële bestanden is daarom zo dat bewerkingen op records ook in dezelfde sequentiële volgorde plaatsvinden.

De nodigde bewerkingen of mutaties worden over een zekere tijdsperiode opgespaard. Deze mutaties worden verzameld tot een mutatiebestand dat gesorteerd wordt in dezelfde volgorde van sleutel als het hoofdbestand of stambestand.

Men gaat dan eenmaal in volgorde door het gehele stambestand en brengt in die volgorde waar nodig de veranderingen aan. In de beginperiode van computers toen magneetbanden nog het enige beschikbare medium voor achtergrondgeheugen vormden was deze organisatievorm ook de enig bruikbare bestandsorganisatie. Door de technische uitvoering van magneetbanden kunnen blokken midden in een bestand niet worden overschreven. Daarenboven moeten alle records achter elkaar staan en kunnen nieuwe records niet worden tussengeschoven.

Een verwerkingsgang met sequentiëel bestand gaat daarom gepaard met een volledig copiëren van het stambestand. Bij dit copiëren worden aan de hand van het mutatiebestand de nodige wijzingen aangebracht. Schematisch kunnen we dit als volgt weergeven:



Dit is een zeer eenvoudig voorbeeld van een systeemstroomschema. Het geeft het volgende weer. Alle mutaties komen binnen op documenten. Deze ondergaan een voorbewerking zoals controle op validiteit van grootheden enz. en worden dan achter elkaar op een magneetband geplaatst. Deze band wordt gesorteerd op sleutel en er ontstaat een gesorteerd bestand op magneetband. Of dit physiek een andere band of dezelfde is uit het schema niet af te lezen en hier ook niet ter zake doende.

Dit gesorteerde mutatiebestand wordt in één verwerkingsgang (een run van een computerprogramma) tesamen met het stambestand, zoals dat bestaat, verwerkt. Programmatisch worden deze twee bestanden als een soort ritssluiting met elkaar verwerkt tot een nieuw stambestand. De invoer van het programma bestaat dus uit "oud stambestand" en "gesorteerd mutatiebestand". De uitvoer wordt gevormd door het "nieuw stambestand". Dit nieuwe stambestand geldt de volgende keer als oud stambestand waaruit dan weer een nieuwe ontstaat enz.

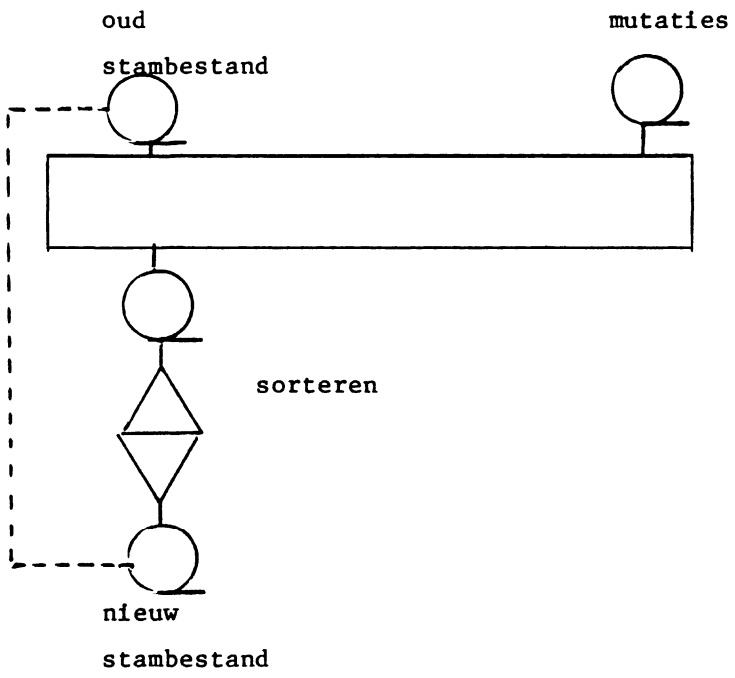
Het laatste wordt in het schema symbolisch weergegeven door de stippellijn.

Ook bij gebruik van schijven in plaats van magneetbanden als achtergrondgeheugen wordt bij sequentiële bestanden dezelfde verwerking toegepast. Weliswaar kunnen op schijven tussengelegen records snel worden gewijzigd mits de lengte maar niet toeneemt, echter het toevoegen of verwijderen van records geeft dezelfde problemen als bij banden.

3.1.2. Mutaties verwerken

Zoals reeds bij het principe aangegeven worden mutaties niet direct verwerkt, maar worden mutaties, ook wel transacties genoemd, over een zekere periode opgespaard. De periode wordt daarbij zo groot gekozen dat het aantal mutaties een behoorlijk deel van het stambestand beslaat, met andere woorden, per verwerkingsgang willen we werken met een zo hoog mogelijke bestandsactiviteit. Per applicatie kan de periode sterk verschillen. Voor het verwerken van saldi van klanten van banken en giro is één dag een gebruikelijke periode. Een personeelsadministratie kan bijvoorbeeld eens per maand verwerkt worden. Telefoonrekeningen worden eens per 2 maanden verwerkt. Maar gegevens over een bouwproject misschien eens per week.

Gedurende de opspaarperiode is het stambestand niet up to date. Behalve de wens van een hoge bestandsactiviteit speelt daarom vaak de actualiteit van de gegevens een rol en neemt men genoegen met een kleinere bestandsactiviteit als anders het stambestand te weinig realistisch zou worden. Mutaties kunnen in het algemeen betrekking hebben op alle velden van een record. Het sleutelveld neemt daarbij een bijzondere postitie in omdat bij wijziging een record dan niet meer op zijn plaats staat. In principe zijn er 3 oplossingen. Ten eerste kan men een sleutelwijziging gelijk behandelen als iedere andere mutatie. Het nieuwe stambestand is dan weliswaar niet meer op sleutelvolgorde maar kan door sorteren daarop gebracht worden. Schematisch:



Het nadeel is dat een volledige sorteergang nodig is voor misschien een gering aantal records dat verkeerd staat. Een tweede methode is het betrokken record laten vervallen en bij de volgende verwerkingsgang weer als nieuw record opnemen. Dit is een veelgebruikte methode, relatief eenvoudig maar tussen de twee verwerkingsgangen bevat het stambestand dan geen enkel gegeven over het betrokken record. Een derde methode is het wegnemen en toevoegen in één verwerkingsgang als twee mutaties uit te voeren. Het nadeel daarvan is dat men alle gegevens van het record eerst voorhanden moet hebben om de mutaties te kunnen aangeven.

Tijdens een verwerkingsgang kunnen per record van het stambestand meerdere mutaties plaatsvinden. In veel gevallen zal men dus deze mutaties in een bepaalde volgorde willen verwerken. Bijvoorbeeld bij een magazijnadministratie per artikel eerst alle aanvullingen en daarna de afboekingen. Soms ook moet de chronologische volgorde binnen de periode worden aangehouden. Men realiseert die volgorde door bij het sorteren van de mutaties als sorteersleutel te gebruiken de echte sleutel gevolgd door een code voor de volgorde. Als voorbeeld bij de magazijnadministratie kunnen we een code 20 voor aanvulling en 40 voor afboeking toevoegen. Bij chronologishe volgorde nemen we in een mutatierecord het tijdstip als extra veld op en denken dat als verlenging van de normale sleutel.

Is het sequentiële bestand op schijven ondergebracht en bestaan de mutaties alleen uit wijzigingen in een aantal velden zonder dat de recordlengte verandert terwijl geen records worden toegevoegd of weggenomen dan kunnen de mutaties ter plaatse worden aangebracht en behoeft niet het gehele bestand te worden gekopieerd. Het in volgorde van sleutel verwerken blijft echter nodig omdat de records achter elkaar staan en de plaats van een record niet rechtstreeks kan worden bepaald. Bovendien wordt door het seqentieel verwerken de instellingstijd van de schrijf-leeskoppen geminimaliseerd.

3.1.3. Beveiliging

Een probleem bij iedere vorm van bestandsorganisatie is de beveiliging van de gegevens in het bestand. Daarbij eerst een onderscheid tussen moedwilligheid en technische storingen. Bij het eerste valt te denken aan het privacy probleem en het onbevoegd en kwaadwillig opvragen en eventueel wijzigen van gegevens. In de bestandsorganisatie zelf is daar niet zo veel aan te doen. Veel meer de overkoepelende operating systeem programma's en organisatorische maatregelen vormen de middelen om daar invloed op uit te oefenen.

Technische storingen zijn een andere zaak. Wanneer tijdens het verwerken van mutaties een magneetband of schijvengeheugen defect raakt kan een deel of het gehele bestand verloren gaan. Men moet dan op een eerdere gecontroleerde goede situatie kunnen terugvallen. Het principe van de sequentiële verwerking zoals beschreven in 3.1.1. biedt daartoe uitstekende mogelijkheden. Zoals aangegeven ontstaat uit een oud stambestand m.b.v. een mutatiebestand een nieuw stambestand. Dit is als het ware een nieuwe generatie. Men bewaart een aantal opeenvolgende generaties stambestanden met bijbehorende mutatiebestanden liefst in duplo en gebruikt magneetbanden of schijvenpakketten pas weer opnieuw nadat ze minstens 3 à 4 generaties oud zijn. Men geeft dit type beveiliging veelal aan met de naam grootvader-vader-zoon systeem.

Uit dit beveiligingsoogpunt wordt daarom veelal ook bij simpele mutaties op schijven toch iedere keer een volledig nieuw bestand gevormd. Brengt men namelijk de mutaties ter plaatse aan dan moeten door andere maatregelen, bijvoorbeeld regelmatige dumps van het gehele bestand de nodige veiligheid geschapen worden.

Valt men na een fout volledig op de vorige generatie terug, dan is het herstellen, de "recovery", in principe eenvoudig, namelijk het herhalen van een vorige verwerkingsgang. Voor de herstelprocedures moet men wel speciale voorschriften opstellen om te vermijden dat bijvoorbeeld door een technisch mankement aan een schijf-leeskop ook vorige generaties bestanden worden opgeblazen.

Bij directe verwerking in het bestand op schijfgeheugen kan men voor herstel vaak de laatste mutatie herhalen. Daarbij moet wel bedacht worden dat niet abusievelijk bijvoorbeeld een optelling of aftrekking opnieuw plaatsvindt. Wanneer het om grote bestanden gaat met veel mutaties kan een verwerkingsgang aanzienlijk tijd kosten. Wanneer een fout optreedt tegen het einde gaat zeer veel tijd verloren. In zulke gevallen en ook bij zeer grote bestanden, verdeelt men het stam- en mutatiebestand over een aantal physische eenheden, dus banden of schijven. Men spreekt dan over "multi volume files". Dit laatste niet te verwarren met "multi file volumes", waarmee men bedoelt het bewaren van een aantal relatief kleine bestanden op één physieke eenheid. DIt heeft niets te maken met beveiliging maar met efficiënter gebruik van geheugenmedia.

3.2. Voorbeeld sequentiële verwerking in schemavorm

In deze paragraaf geven we een iets uitgebreider voorbeeld gebaseerd op het principe van een girodienst zoals die dagelijks allerlei mutaties moet verwerken. Overigens is dit voorbeeld een zeer gesimplificeerd beeld van een werkelijk verwerkingsprogramma voor een giro of bankdienst.

De totale dagelijkse verwerking voor dit voorbeeld is weergegeven in het systeem stroomschema in fig. 3.2.1., waarbij de volgende toelichting. De gegevens van iedere klant worden bijgehouden in een stambestand. Het oude stambestand is aangegeven met STAMIN, het nieuw te vormen bestand met STAMUIT.

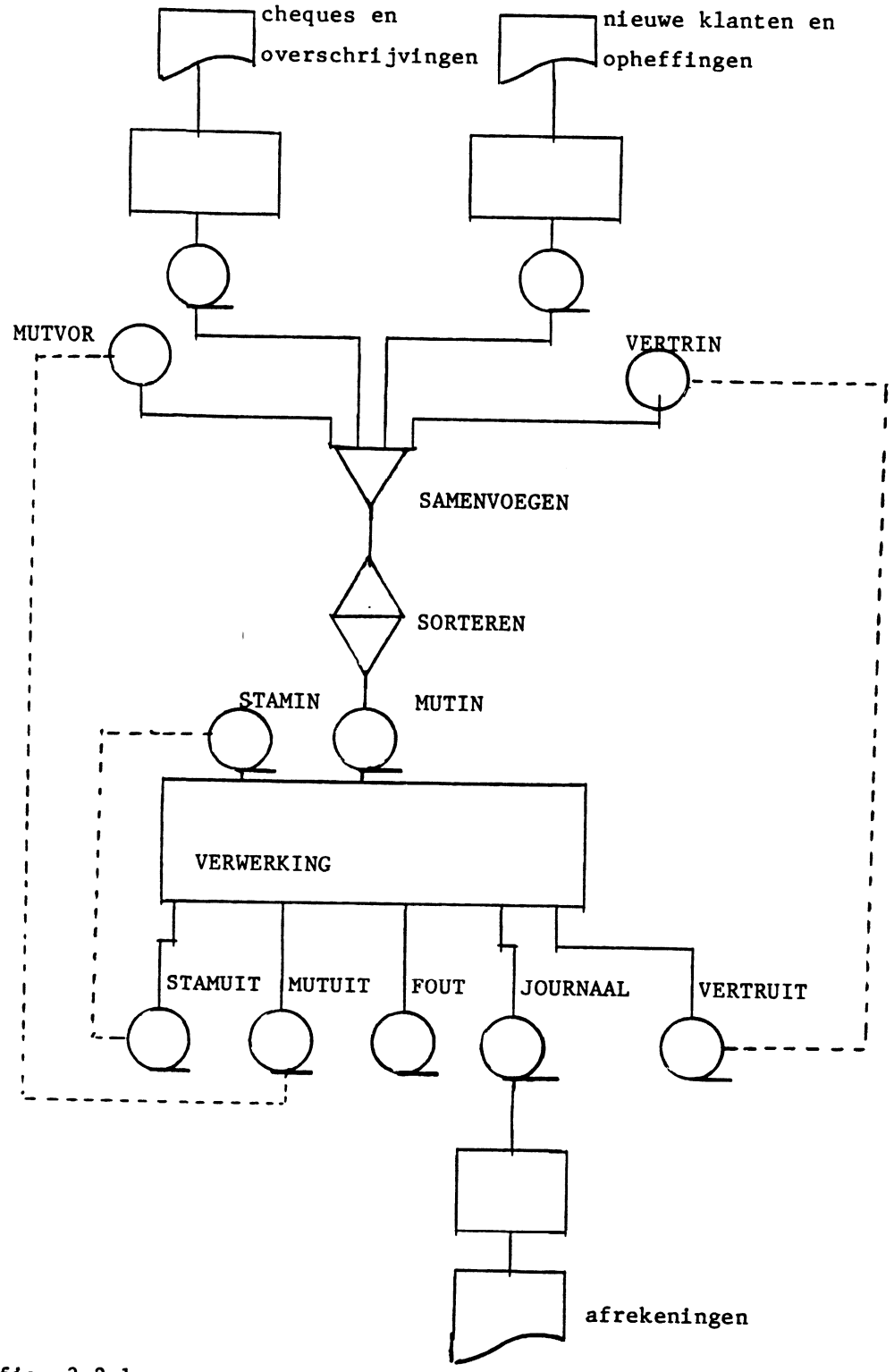

cheques en
overschrijvingen
nieuwe klanten en
opheffingen
MUTVOR
VERTRIN
SAMENVOEGEN
SORTEREN
STAMIN
MUTIN
VERWERKING
STAMUIT
MUTUIT
FOUT
JOURNAAL
VERTRUIT
afrekeningen

fig. 3.2.1.

Aangenomen is dat alle bestanden op banden zijn ondergebracht. Dagelijks kunnen de volgende mutaties optreden. Er kunnen nieuwe klanten worden ingeschreven en bestaande rekeningnummers worden opgeheven. De gegevens hiervoor worden op machine leebare documenten aangeleverd. Na een controle worden deze gegevens verzameld op een magneetband. Hierbij wordt in ieder record een veld CODE toegevoegd en wel voor nieuwe rekeninghouders een waarde 10 en voor opheffen een waarde 40.

Rekeninghouders kunnen betaalcheques gebruiken en overschrijvingen doen. Betaalcheques worden zonder meer van het rekeningnummer afgeschreven, ook al wordt het saldo negatief. Een overschrijving wordt niet uitgevoerd als het saldo niet toereikend is. Een dergelijke overschrijving wordt overgebracht naar het bestand VERTRUIT (vertraagd uit) en enige dagen later opnieuw via VERTRIN aangeboden voor afschrijving. Alle geaccepteerde afschrijvingen worden opgenomen in het bestand MUTUIT waarbij bij- en afschrijvingsnummer worden verwisseld, terwijl bovendien de mutatiecode wordt gewijzigd. Bij een volgende verwerkingsgang de volgende dag wordt dit bestand dan via MUTVOR weer gebruikt voor invoer en daarmee worden de bijschrijvingen geregeld. Bijschrijving vindt dus een dag later plaats dan de afschrijving.

Voor de mutatie gelden vier invoerstromen, namelijk goedgekeurde afschrijvingen die bijgeschreven moeten worden, aangeboden afschrijvingen, nieuwe en vervallen rekeningen en vertraagde afschrijvingen als gevolg van ontoereikend saldo. Na sortering vormen zij tesamen MUTIN. Tenslotte worden alle geslaagde mutaties vastgelegd in een uitvoerbestand JOURNAAL waaruit de afrekeningen naar de klanten kunnen worden samengesteld. Wanneer tijdens de verwerking verkeerde situaties worden geconstateerd dan worden die mutaties vastgelegd in een foutbestand FOUT.
Om verschillende mutaties in de juiste volgorde per rekeningnummer te laten verlopen worden alle mutaties van de volgende mutatiecode voorzien.

CODE	Omschrijving
10	inschrijving
20	afschrijven cheque
25	afschrijven overschrijving
30	bijschrijven cheque
35	bijschrijven overschrijving
40	opheffing

Een record in een stambestand heeft de volgende velden

SLEUTEL
SALDO
REST

Met REST is bedoeld de samenvatting van alle velden die voor dit voorbeeld geen rol spelen zoals adres, naam, enz.
Een mutatierecord heeft de velden

SLEUTEL1
SLEUTEL2
CODE
BEDRAG
REST

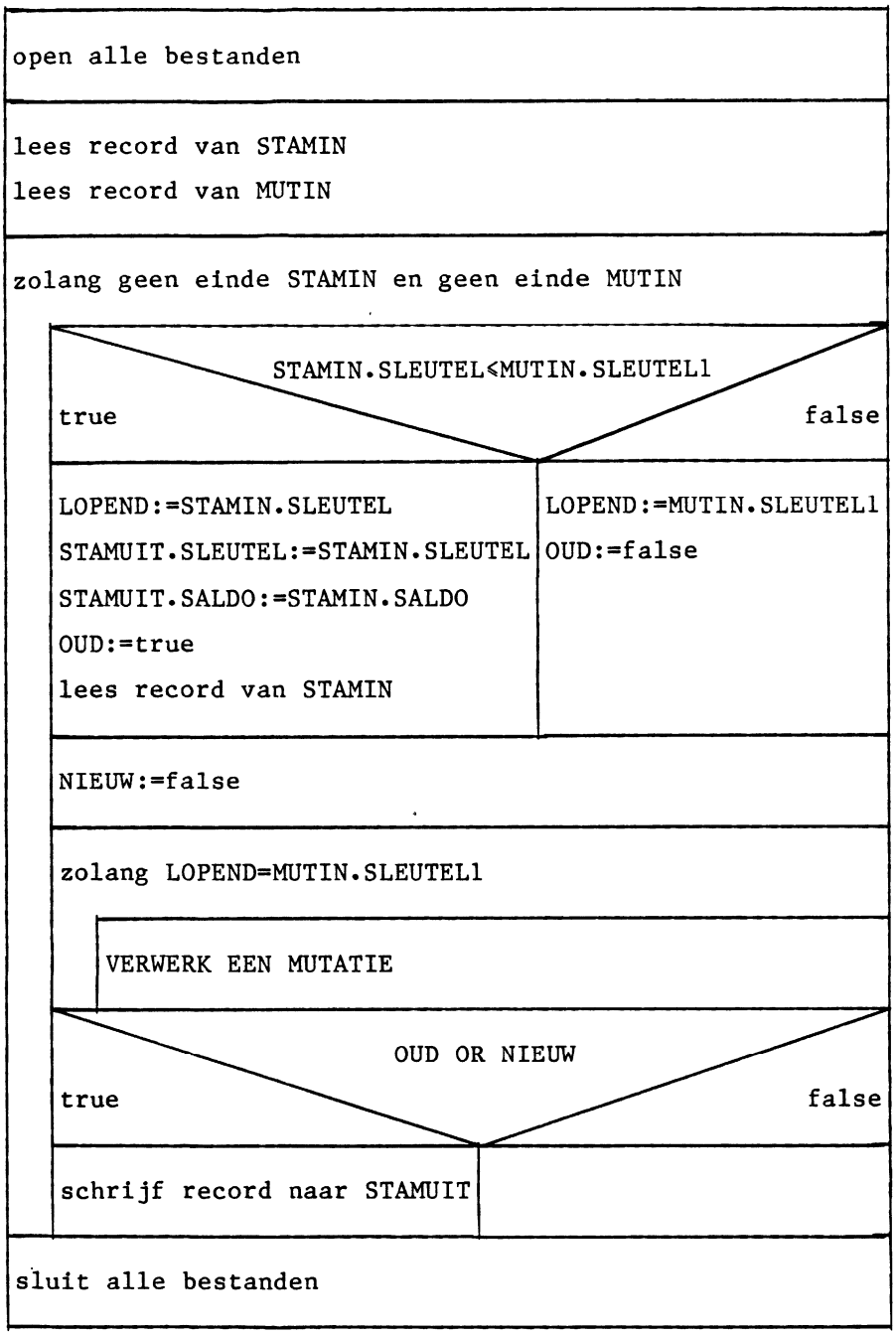
In de volgende programmastructuurdiagrammen is het verwerkingsprogramma nader uitgewerkt. Daarbij is het volgende principe aangehouden. Zodra een invoerrecord verwerkt en niet meer nodig is wordt een leesopdracht gegeven voor het volgende record van dat bestand. Zodra een uitvoerrecord is samengesteld wordt de uitvoeropdracht naar dat uitvoerbestand gegeven. Daarmee wordt in- en uitvoer zoveel mogelijk gelijktijdig met verwerking uitgevoerd.

Aangenomen wordt dat nadat het laatste record van een invoerbestand is gelezen, dus STAMIN of MUTIN, het sleutelveld op de hoogst mogelijke waarde wordt gesteld die voor normale sleutels niet bereikbaar is (In COBOL de waarde HIGH-VALUE). Daarmee geeft de vergelijking

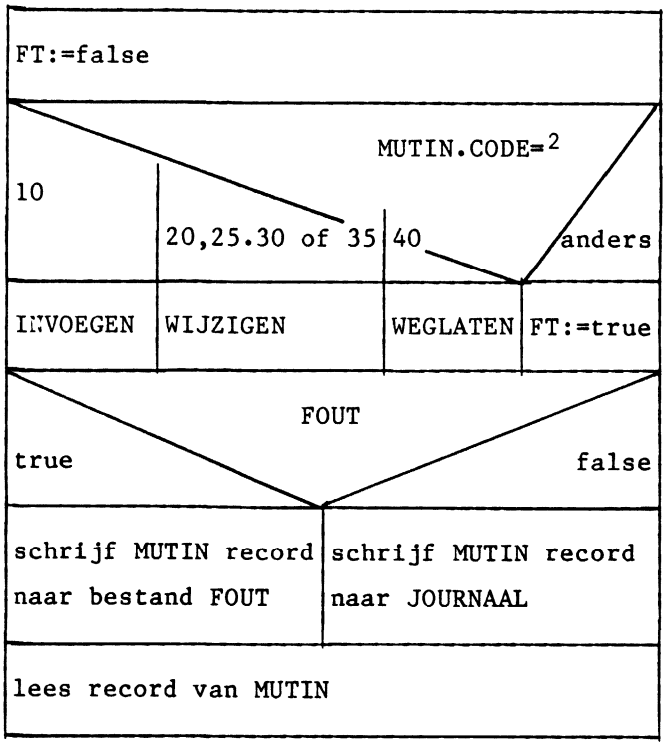
STAMIN.SLEUTEL < MUTIN.SLEUTEL1

altijd de juiste waarde true of false.
OUD en NIEUW zijn logische variabelen om controle uit te oefenen dat geen nieuw rekeningnummer wordt ingevoerd voor een bestaand of mutaties voor een niet bestaand nummer, enz.
Verschillende situaties die foutieve mutaties zouden veroorzaken worden als zodanig herkend en geven aanleiding tot een foutmelding. In de routine M-IN-UIT wordt uit een geaccepteerde afschrijving een nieuw mutatierecord van bijschrijving gevormd.

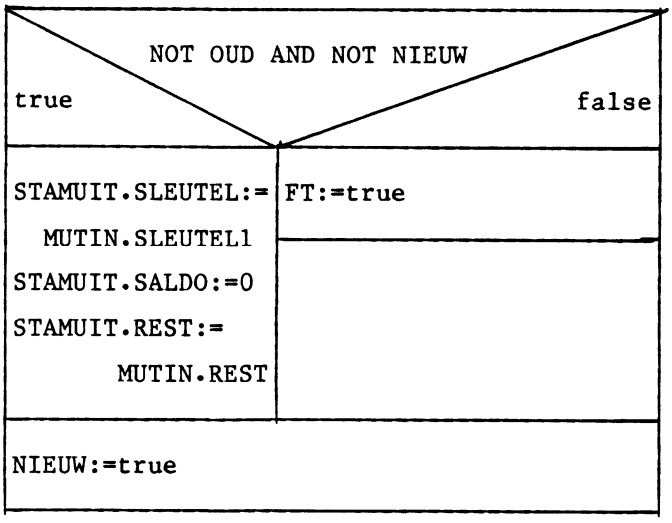
VERWERKING


open alle bestanden
lees record van STAMIN
lees record van MUTIN
zolang geen einde STAMIN en geen einde MUTIN
STAMIN.SLEUTEL<MUTIN.SLEUTEL1
true
false
LOPEND:=STAMIN.SLEUTEL
STAMUIT.SLEUTEL:=STAMIN.SLEUTEL
STAMUIT.SALDO:=STAMIN.SALDO
OUD:=true
lees record van STAMIN
LOPEND:=MUTIN.SLEUTEL1
OUD:=false
NIEUW:=false
zolang LOPEND=MUTIN.SLEUTEL1
VERWERK EEN MUTATIE
OUD OR NIEUW
true
false
schrijf record naar STAMUIT
sluit alle bestanden

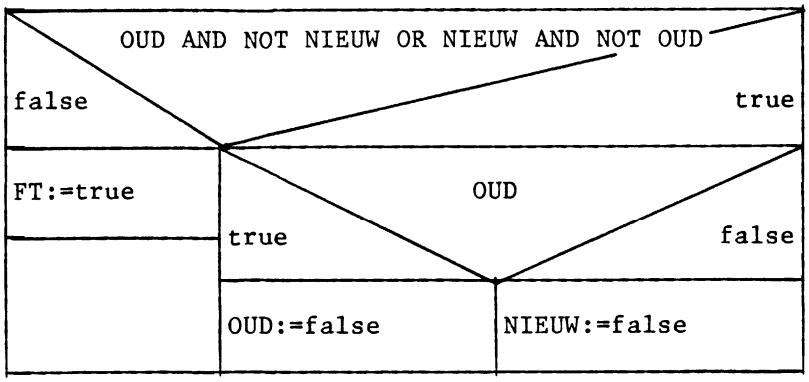
VERWERK EEN MUTATIE



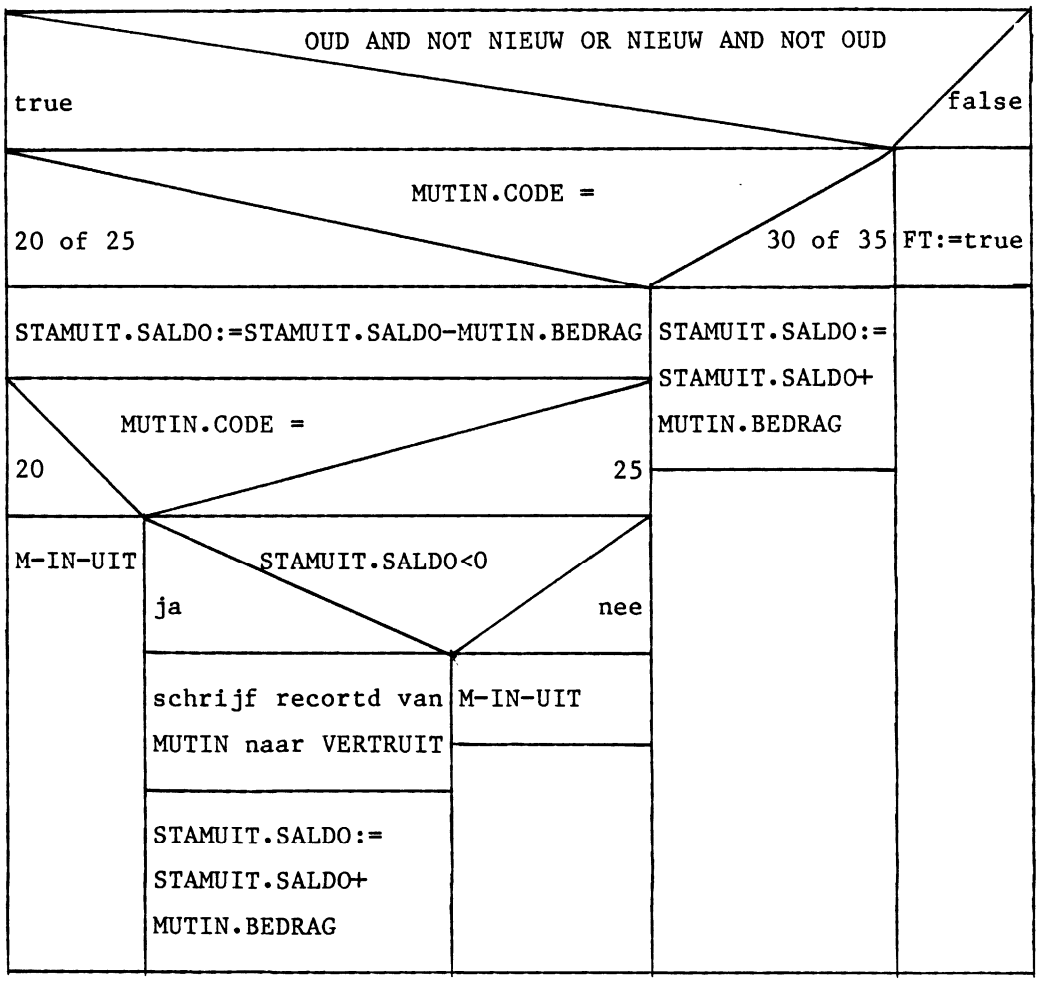
INVOEGEN



WEGLATEN



WIJZIGEN



M-IN-UIT

```
MUTUIT.SLEUTEL1:=MUTIN.SLEUTEL2
MUTUIT.SLEUTEL2:=MUTIN.SLEUTEL1
MUTUIT.CODE:=MUTIN.CODE+10
MUTUIT.BEDRAG:=MUTIN.BEDRAG
MUTUIT.REST:=MUTIN.REST
```

```
schrijf record naar MUTUIT
```

3.3. Voorbeeld sequentiële verwerking in COBOL

Het voorbeeld zoals gegeven in 3.2. wordt hier nog een keer herhaald als COBOL programma. Daarbij wordt vooral de aandacht gevestigd op de wijze waarop een goed gestruktureerd programma ook in COBOL op juiste en overzichtelijke wijze kan worden gecodeerd. Voor namen van subroutines worden namen gebruikt overeenkomstig de schema's van par. 3.2. en aangevuld met letters en cijfers, overeenkomstig de logische nesting van de programma's.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. VOORBEELD.
AUTHOR. WOLBERS.

ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
   SELECT STAMIN ASSIGN TO DA-S-STIN
   SELECT STAMUIT ASSIGN TO DA-S-STUIT
   SELECT MUTIN ASSIGN TO DA-S-MIN
   SELECT MUTUIT ASSIGN TO DA-S-MUIT
   SELECT FOUT ASSIGN TO DA-S-FT
   SELECT JOURNAAL ASSIGN TO DA-S-JN
   SELECT VERTRUIT ASSIGN TO DA-S-VTR
```

```
DATA DIVISION.
FILE SECTION.
FD STAMIN.
OI STAMIN-REC.
   02 SLEUTEL PIC 9(8).
   02 SALDO PIC S9(9)V99.
   02 REST

FD STAMUIT.
01 STAMUIT-REC.
   02 SLEUTEL PIC 9(8).
   02 SALDO PIC S 9(9)V99.
   02 REST

FD MUTIN.
01 MUTIN-REC.
   02 SLEUTEL1 PIC 9(8).
   02 SLEUTEL2 PIC 9(8).
   02 CODE PIC 99.
   02 BEDRAG PIC 9(7)V99.
   02 REST

FD MUTUIT.
01 MUTUIT-REC.
   02 SLEUTEL1 PIC 9(8).
   02 SLEUTEL2 PIC 9(8).
   02 CODE PIC 99.
   02 BEDRAG PIC 9(7)V99.
   02 REST

FD FOUT.
01 FOUT-REC.
   02 SLEUTEL1 PIC 9(8).
   02 SLEUTEL2 PIC 9(8).
   02 CODE PIC 99.
   02 BEDRAG PIC 9(7)V99.
   02 REST
```

```
FD JOURNAAL.
01 JOURNAAL-REC.
   02 SLEUTEL1 PIC 9(8).
   02 SLEUTEL2 PIC 9(8).
   02 CODE PIC 99.
   02 BEDRAG PIC 9(7)V99.
   02 REST

FD VERTRUIT.
01 VERTRUIT-REC.
   02 SLEUTEL1 PIC 9(8).
   02 SLEUTEL2 PIC 9(8).
   02 CODE PIC 99.
   02 BEDRAG PIC 9(7)V99.
   02 REST

WORKING-STORAGE SECTION.

77 LOPEND PIC 9(8).

01 VARIABELEN USAGE IS COMP.
   02 OUD PIC 9.
   02 NIEUW PIC9.
   02 FT PIC 9.
   02 TEST PIC 99.
   02 STAM PIC 99.
   02 MUT PIC 9.

PROCEDURE DIVISION.
HOOFD SECTION.
   PERFORM OPENEN-1.
   PERFORM STARTLEZEN-2.
   PERFORM MUTEREN-3 UNTIL STAM=1 AND MUT=1.
   PERFORM SLUITEN-4.
   STOP RUN.
```

```
OPENEN-1 SECTION.
   OPEN INPUT STAMIN MUTIN
        OUTPUT STAMUIT MUTUIT FOUT
            JOURNAAL VERTRUIT.
   MOVE ZERO TO STAM MUT.

STARTLEZEN-2 SECTION.
   PERFORM LEES-STAM-2A
   PERFORM LEES-MUT-2B.

*2A OOK UIT 3.
LEES-STAM-2A SECTION.
   READ STAMIN
        AT END MOVE 1 TO STAM
            MOVE HIGH-VALUE TO SLEUTEL IN STAMIN-REC.

* 2B OOK UIT 3A.
LEES-MUT-2B SECTION.
   READ MUTIN
        AT END MOVE 1 TO MUT
            MOVE HIGH-VALUE TO SLEUTEL1 IN MUTIN-REC.

MUTEREN-3 SECTION.
   IF SLEUTEL IN STAMIN-REC NOT GREATER THAN
   SLEUTEL IN MUTIN-REC
   MOVE SLEUTEL IN STAMIN-REC TO LOPEND
   MOVE STAMIN-REC TO STAMUIT-REC
   MOVE ZERO TO OUD
   PERFORM LEES-STAM-2A
   ELSE
   MOVE SLEUTEL1 IN MUTIN-REC TO LOPEND
   MOVE 1 TO OUD.
   MOVE 1 TO NIEUW.
   PERFORM MUTATIE-3A UNTIL
        SLEUTEL1 IN MUTIN-REC NOT EQUAL TO LOPEND.
   IF OUD IS ZERO OR NIEUW IS ZERO
   PERFORM SCHRIJF-STAM-3B.
```

```
MUTATIE-3A SECTION.
   MOVE 1 TO FT.
   MOVE CODE IN MUTIN-REC TO TEST.
   IF TEST=10 PERFORM INVOEGEN-3A1
   ELSE IF TEST=20 OR TEST=25 OR TEST=30
       OR TEST=35 PERFORM WIJZIGEN-3A2
   ELSE IF TEST=40 PERFORM WEGLATEN-3A3
   ELSE MOVE ZERO TO FT.
   IF FT IS ZERO PERFORM SCHRIJF-FOUT-3A4
   ELSE SCHRIJF-JOURNAAL-3A5.
   PERFORM LEES-MUT-2B.

INVOEGEN-3A1 SECTION.
   IF OUD=1 AND NIEUW=1 MOVE SLEUTEL1 IN
   MUTIN-REC TO SLEUTEL IN STAMUIT-REC
   MOVE ZERO TO SALDO IN STAMUIT-REC
   MOVE REST IN MUTIN-REC TO REST IN
   STAMUIT-REC
   ELSE MOVE ZERO TO FT.
   MOVE ZERO TO NIEUW.

WIJZIGEN-3A2 SECTION.
   IF OUD=ZERO AND NIEUW =1 OR
       NIEUW=ZERO AND OUD=1 PERFORM GOED-3A2 A
   ELSE MOVE ZERO TO FT.

GOED-3A2A SECTION.
   IF CODE IN MUTIN-REC=20 OR 25
   PERFORM MINDER-3A2A1
   ELSE PERFORM MEER-3A2A2.
```

```
MINDER-3A2A1 SECTION.
   SUBSTRACT BEDRAG IN MUTIN-REC
       FROM SALDO IN STAMUIT-REC.
   IF CODE IN MUTIN-REC=20 PERFORM M-IN-UIT-3A2A1A
   ELSE IF SALDO IN STAMUIT-REC<ZERO
   PERFORM SCHRIJF-VTR-3A2A1B
   ADD BERAG IN MUTIN-REC TO
       SALDO IN STAMUIT-REC
   ELSE PERFORM M-IN-UIT-3A2A1A.

M-IN-UIT-3A2A1A SECTION.
   MOVE SLEUTEL2 IN MUTIN-REC TO
       SLEUTEL1 IN MUTUIT-REC
   MOVE SLEUTEL1 IN MUTIN-REC TO
       SELUTEL2 IN MUTIN-REC
   ADD CODE IN MUTIN-REC 10 GIVING CODE IN MUTUIT-REC
   MOVE BEDRAG IN MUTIN-REC TO
       BEDRAG IN MUTUIT-REC
   MOVE REST IN MUTIN-REC TO
       REST IN MUTUIT-REC.
   WRITE MUTUIT-REC.

SCHRIJF-VTR-3A2A1B SECTION.
   WRITE VERTRUIT-REC FROM MUTIN-REC.

MEER-3A2A2 SECTION.
   ADD BEDRAG IN MUTIN-REC TO
       SALDO IN STAMUIT-REC

WEGLATEN-3A3 SECTION.
   IF OUD=0 AND NIEUW=1 OR
       NIEUW=0 AND OUD=1
   IF OUD=0 MOVE 1 TO OUD
   ELSE MOVE 1 TO NIEUW
   ELSE MOVE ZERO TO FT.

SCHRIJF-FOUT-3A4 SECTION.
   WRITE FOUT-REC FROM MUTIN-REC.
```

```
SCHRIJF-JOURNAAL-3A5 SECTION.
   WRITE JOURNAAL-REC FROM MUTIN-REC.

SCHRIJF-STAM-3B SECTION.
   WRITE STAMUIT-REC.

SLUITEN-4 SECTION.
   CLOSE STAMIN STAMUIT MUTIN MUTUIT
       FOUT JOURNAAL VERTRUIT.
```

4. SORTEREN

4.1. Intern sorteren

Er bestaat een groot aantal sorteeralgoritmen. Deze zijn ruwweg in 2 klassen te verdelen. Ten eerste die algoritmen die als basis gebruiken dat alle records op ieder moment tijdens het sorteerproces even snel bereikbaar zijn. Ten tweede die processen die tenminste deelseries sequentieel benaderen. Beide klassen hangen ten nauwste samen met het geheugen waarop het te sorteren bestand is opgeslagen. Wanneer alle te sorteren records tegelijk in het primaire geheugen kunnen worden opgeslagen hebben we duidelijk te maken met de eerste klasse. In dat geval zijn vele bekende methoden beschikbaar, zoals "bubble sort", "quicksort", "heapsort", e.a. Wanneer niet meer het gehele bestand tegelijk in het primaire geheugen kan worden opgeslagen gaat de efficiency van dergelijke methoden opeens sterk achteruit. Immers, het benaderen van individuele records gaat met grote vertragingen gepaard. Bij gebruik van magneetbanden is dit zonder meer prohibitief. Maar ook bij schijvengeheugens is de totale tijd om een willekeurige record op te halen of terug te brengen relatief groot tengevolge van de insteltijd van de arm en de wachttijd tot het begin van het record is bereikt (seektime en latency).

In het algemeen worden sorteerprocessen daarom verdeeld in 2 fasen. In de eerste fase wordt het gehele te sorteren bestand gesplitst in series zodanig dat iedere serie nog in het primaire geheugen kan worden opgenomen. Iedere serie wordt in het primaire gesorteerd volgens een optimale sorteermethode. Er ontstaan series records die per serie dan gesorteerd zijn.

Deze fase van het sorteerproces noemt men de interne fase ofwel het intern sorteren De volgende fase is dat men steeds 2 of meer van deze reeds gesorteerde series sequentieel inleest en hieruit langere gesorteerde series formeert, die weer op achtergrondgeheugen worden weggeschreven, enigszins vergelijkbaar met het verwerken van mutaties bij een sequentieel bestand. Er ontstaan dan gesorteerde series van grotere lengte. Met dit proces gaat men door tot er tenslotte één serie ontstaat en die serie is dan het gesorteerde bestand. Deze fase van het sorteerproces wordt genoemd het extern sorteren.

Er bestaat een aantal algoritmen om series te combineren. Hiervan zullen de 2 meest gebruikte methoden worden behandeld in paragraaf 4.2, namelijk

P-weg menging en polyphase menging.

Het totale proces verloopt dus in 2 stappen. Eerst de interne sorteerfase waarbij zo groot mogelijke gesorteerde series worden gevormd. Daarna de externe sorteerfase waarbij door menging de series worden gecombineerd tot een uiteindelijk volledig gesorteerd bestand. Wanneer de records relatief klein zijn kan men met de interne sortering reeds grote series verkrijgen, waardoor het aantal mengslagen in de externe sorteerfase beperkt blijft. Zijn de records zeer groot dan is de interne fase veelal nauwelijks lonend. Men moet daarbij bedenken dat in een volledig chaotisch bestand de gemiddelde serielengte toch al iets boven de 2 ligt. In veel sorteerprocessen laat men daarom de interne fase achterwege en start meteen met de externe sorteerfase. Zelfs beschouwt men in dat geval het te sorteren bestand van n records als bestaande uit n series ieder ter lengte van één record. Een toevallige juiste volgorde wordt dan niet eens expliciet gebruikt. De externe sorteerfase kan plaatsvinden zowel bij gebruik van schijvengeheugen als magneetbandgeheugens. De meeste externe sorteermethoden zijn trouwens ontwikkeld in de tijd dat alleen nog maar magneetbanden beschikbaar waren. In het algemeen zal een mengmethode sneller zijn naarmate men meer ingangsstromen van series tegelijk kan mengen tot één uitvoerstroom. Bij gebruik van magneetbanden is dan het nadeel dat men ook meer magneetbandeenheden nodig heeft. Bij gebruik van schijvengeheugens is dit minder een probleem. Echter, iedere invoerstroom heeft zijn eigen buffer, eventueel dubbele buffer, nodig in het primaire geheugen. Verder moet iedere stroom wel gereserveerd worden op achtergrondgeheugen waardoor vaak extra reserveringen nodig zijn. Tenslotte zullen door het groter worden van de schijfeenheden veelal alle in en uitvoerstromen van series records op één schijfeenheid thuishoren en dus toch alleen maar consecutief gelezen en geschreven kunnen worden. Het voordeel van vele stromen gaat dan grotendeels verloren.

4.2. Extern sorteren

4.2.1. P-weg menging

Bij P-weg menging worden steeds P series uit P invoerstromen samengevoegd tot één serie die P maal zo lang is. De ontstane series worden verdeeld over P uitvoerseries. Nadat alle series verwerkt zijn verwisselen de uitvoerstro-

men en invoerstromen van functie. Eén dergelijke verwerking noemt men één sorteergang of sorteerslag. Per sorteergang neemt de serielengte toe met een factor P. Na x sorteergangen is de serielengte dan P^x. Als n het aantal records is dan is het sorteerproces beëindigd zodra $P^z>n$ dus $x>P\log n$. Een voorbeeld van P-weg menging geven we in het volgende schema. Daarbij is P=2 dus steeds 2 invoerstromen en 2 uitvoerstromen. In dit schema en ook in de schema's van de volgende paragraaf wordt de volgende notatie gebruikt.

P^q betekent P series ieder met een lengte die q maal zo groot is als de serielengte waarmee het externe sorteerproces is gestart. Dus bijvoorbeeld 63^4 betekent 63 series van 4 maal oorspronkelijke lengte.

500^1	0	0	0
0	0	250^1	250^1
125^2	125^2	0	0
0	0	63^4	62^4
31^8 1^4	31^8	0	0
0	0	16^{16}	15^{16} 1^4
8^{23}	7^{32} 1^{20}	0	0
0	0	4^{64}	3^{64} 1^{52}
2^{128}	1^{128} 1^{116}	0	0
0	0	1^{256}	1^{244}
1^{500}	0	0	0

Voor deze sortering zijn dus 9 sorteergangen nodig waarbij in iedere sorteergang alle 500 records worden getransporteerd plus een eerste gang waarbij de eerste verdeling van 500 series naar de eerste 2 maal 250 series plaatsvindt. Ieder record wordt 10 maal verplaatst.

In het algemeen gelddt bij P weg menging dat het aantal verplaatsingen per record is

$$1 + {}^{P}\log n \ .$$

In dit geval $1 + {}^{2}\log 500 = 1 + 9 = 10$ verplaatsingen per record.

4.2.2. Polyphase menging

4.2.2.1. Principe van polyphase menging

Bij P-weg menging zijn bij iedere sorteergang steeds P invoerstromen maar slechts één uitvoerstroom betrokken, terwijl het totaal aantal stromen is 2P. Bij polyphase menging worden van de totaal P+1 stromen steeds P als invoerstromen en één als uitvoerstroom gebruikt.
We geven eerst een voorbeeld van het sorteren van 653 oorspronkelijke series waarbij totaal 4 stromen worden gebruikt dus steeds 3 invoer- en 1 uitvoerstroom. Het schema is dan als volgt.

653^{1}	0	0	0				
				653	x	1 =	653
0	274^{1}	230^{1}	149^{1}				
				149	x	3 =	447
149^{3}	125^{1}	81^{1}	0				
				81	x	5 =	405
68^{3}	44^{1}	0	81^{5}				
				44	x	9 =	396
24^{3}	0	44^{9}	37^{5}				
				24	x	17 =	408
0	24^{17}	20^{9}	13^{5}				
				13	x	31 =	403
13^{31}	11^{17}	7^{9}	0				
				7	x	57 =	399
6^{31}	4^{17}	0	7^{57}				
				4	x	105 =	420
2^{31}	0	4^{105}	3^{57}				
				2	x	193 =	386
0	2^{193}	2^{105}	1^{57}				
				1	x	355 =	355
1^{355}	1^{193}	1^{105}	0				
				1	x	653 =	653
0	0	0	1^{653}				
							4925

Het gemiddeld aantal transporten per serie is hier nu 4925/653 = 7,542. In de hier gegeven vorm is polyphase menging alleen mogelijk met zeer bepaalde aantallen startseries. Deze mogelijke aantallen worden bepaald door het bovengenoemde schema in omgekeerde volgorde op te stellen. Daarbij vermelden we per regel alleen het aantal series per stroom of deelbestand en de som van alle stromen.

n					totaal t_n
0	1	0	0	0	1
1	0	1	1	1	3
2	1	0	2	2	5
3	3	2	0	4	9
4	7	6	4	0	17
5	0	13	11	7	31
6	13	0	24	20	57
7	37	24	0	44	105
8	81	68	44	0	193
9	0	149	125	81	355
10	149	0	274	230	653
11	423	274	0	504	1201
12	927	778	504	0	2209

Beschouwen we de systematiek in dit schema dan merken we het volgende op. Het grootste aantal series op de n^e regel is het kleinste aantal series op de $(n+1)^e$ regel. Het deelbestand van die grootste is op de $(n+1)^e$ regel 0. De andere deelbestanden zijn op de $(n+1)^e$ regel precies de waarde van de n^e regel plus die grootste van de n^e regel.

In de laatste kolom is steeds vermeld de som van alle series per regel. Het is duidelijk dat op de gegeven manier alleen aantallen records die toevallig gelijk zijn aan getallen in die kolom op deze wijze gesorteerd kunnen worden. In het voorbeeld was daarom niet toevallig 653 gekozen. Hoe men voor willekeurige aantallen dit proces gebruikt wordt aangegeven in paragraaf 4.2.2.5. Deze speciale waarden zullen we aangeven met t_n.

4.2.2.2. Berekening voor polyphase menging

Bij polyphase menging is het aantal sorteerslagen en het gemiddeld aantal transporten per record voor het algemene geval van P invoerstromen en 1 uitvoerstroom iets moeilijker te bepalen dan bij pweg menging (zie voor een afleiding Knuth, sortering en harching).

We stellen het aantal te sorteren oorspronkelijke series op t_n. Het aantal sorteerslagen inclusief de eerste verdeelslag noemen we s en het gemiddeld aantal transporten per record w. Men kan aantonen dat dan met zeer goede benadering geldt:

$s = a \ln t_n + b$ en

$w = c \ln t_n + d$

Hierin zijn a, b, c en d constanten die nog wel van de waarde van P afhangen en verder gelden deze formules strikt genomen alleen voor waarden van t_n die een karakteristiek aantal zijn zoals in de vorige paragraaf aangegeven. Voor a, b, c en d gelden

P	a	b	c	d
2	2.07809	0.67228	1.50372	0.99204
3	1.64102	0.36367	1.01484	0.96452
4	1.52380	0.07772	0.86299	0.92061
5	1.47935	-0.18482	0.79578	0.86352
6	1.46006	-0.42424	0.76182	0.79650
7	1.45112	-0.64163	0.74363	0.72288
8	1.44684	-0.83862	0.73365	0.64587
9	1.44475	-1.01715	0.72813	0.56824
10	1.44372	-1.17928	0.72506	0.49209
11	1.44320	-1.32699	0.72337	0.41888
12	1.44295	-1.46216	0.72245	0.34940
13	1.44282	-1.58642	0.72194	0.28403
14	1.44276	-1.70122	0.72167	0.22280
15	1.44273	-1.80777	0.72152	0.16553
16	1.44271	-1.90711	0.72144	0.11197
17	1.44270	-2.00012	0.72140	0.06180
18	1.44270	-2.08752	0.72137	0.01470
19	1.44270	-2.16996	0.72136	-0.02962
20	1.44270	-2.24794	0.72135	-0.07145

Vergelijk het gegeven voorbeeld met 53 records en p = 3 dan

s = 1,64102 en 653 + 0,36367 = 11,0000677
w = 1,01484 en 653 + 0,96452 = 7,5423

4.2.2.3. Polyphase menging voor willekeurig aantal record

Het laatste probleem is wat te doen bij een aantal gegeven records dat niet overeenkomt met een waarde van t_n. We nemen de eerste t_n die groter is dan het gegeven aantal records. en beschouwen de record die "te veel" zijn als dummy records. Men kan die dummy records dan nog op verschillende wijzen verdelen. Een niet geheel optimale maar wel praktische oplossing is een gelijke verdeling over alle stromen bij de eerste distributie.

Voor het eerder gegeven voorbeeld van 500 series met 4 stromen kiezen we n=10 waarmee t_n=653. Het aantal dummy series is dan 153 en verdeeld over 3 stromen krijgt ieder 51. Deze dummy series worden natuurlijk niet echt getransporteerd maar dienen alleen voor berekening van de juiste verdelingen. Het sorteren van die 500 series gaat als volgt:

0	500^1	0	0	
				500 x 1 = 500
51^D	0	51^D	51^D	
98^1		223^1	179^1	
				98 x 3 = 294
0	51D	125^1	81^1	
	98^3			
				51 x 2 +
51^2	68^3	44^1	0	30 x 5 = 252
30^5				
				44 x 6 = 264
7^2	24^3	0	44^6	
30^5				
				7 x 11 +
13^5	0	7^{11}	20^6	17 x 14 = 315
		17^{14}		
				7 x 22 +

0	7^{22} 6^{25}	11^{14}	7^{6}	6 x 25 = 304
7^{42}	6^{25}	4^{14}	0	7 x 42 = 294
3^{42}	2^{25}	0	4^{81}	4 x 81 = 324
1^{42}	0	2^{148}	2^{81}	2 x 148 = 296
0	1^{271}	1^{148}	1^{81}	1 x 271 = 271
1^{500}	0	0	0	1 x 500 = 500
				3614

Het gemiddelde aantal transporten per record is 3614/500 = 7,288 en vergelijk dit met 10 bij 2 weg menging. De gegeven tabel voor berekening van s en w geldt in feite alleen voor exacte t_n waarden. Een redelijke benadering is het overigens wel voor tussengelegen waarden. In dit geval:

$$s = 1{,}64102 \ln 500 + 0{,}36367 = 10{,}56$$

$$w = 1{,}01484 \ln 500 + 0{,}96452 = 7{,}271$$

4.3. Sorteren van grote records

Zowel bij de interne fase als de externe fase vinden tijdens het proces veel verplaatsingen van records plaats. Wanneer de records groot zijn gaat dit gepaard met veel tijd. Men kan hier wat op besparen door een tabel te maken van sleutels en recordadressen. Deze lijst is te sorteren waarna men in prinicpe weet waar ieder record thuis hoort zodat ieder record in zijn geheel maar eenmaal verplaatst hoeft te worden. Deze methode is bruikbaar zolang maar alle records in het primaire geheugen staan opgeslagen. Is dat niet meer het geval dan is deze methode problematischer. Dit proces is als volgt nader te specificeren.

We gaan uit van n records die ieder bestaan uit sleutel S en verdere infor-

matie I. We nemen aan dat I groot is en daarmee ieder record. We hebben dus een lijst

S_1, I_1
S_2, I_2
S_k, I_k
S_n, I_n

Door alle records 1 maal in te lezen en alleen de sleutels te bewaren met hun relatieve adressering in deze lijst ontstaat:

S_1, 1
S_2, 2
S_k, k
S_n, n

We hebben nu in feite een lijst van kleine records. Neem aan dat deze lijst zelfs past in het primaire geheugen zodat we hier een snelle interne sorteermethode op kunnen loslaten. Daaruit ontstaat de volgende lijst die gesorteerd is op de sleutel $Sa_1 < Sa_2 < Sa_3$ enz.

Aan deze lijst voegen we nu weer volgorde nummers toe waardoor de volgende lijst ontstaat.

1, Sa_1, a_1
2, Sa_2, a_2
k, Sa_k, a_k
n, Sa_n, a_n

Tenslotte sorteren we deze lijst weer op de oorspronkelijke volgnummers a waardoor ontstaat

b_1, S_1, 1
b_2, S_2, 2
b_k, S_k, k
b_n, S_n, n

De b-kolom geeft voor ieder record weer op welke plaats ieder record behoort

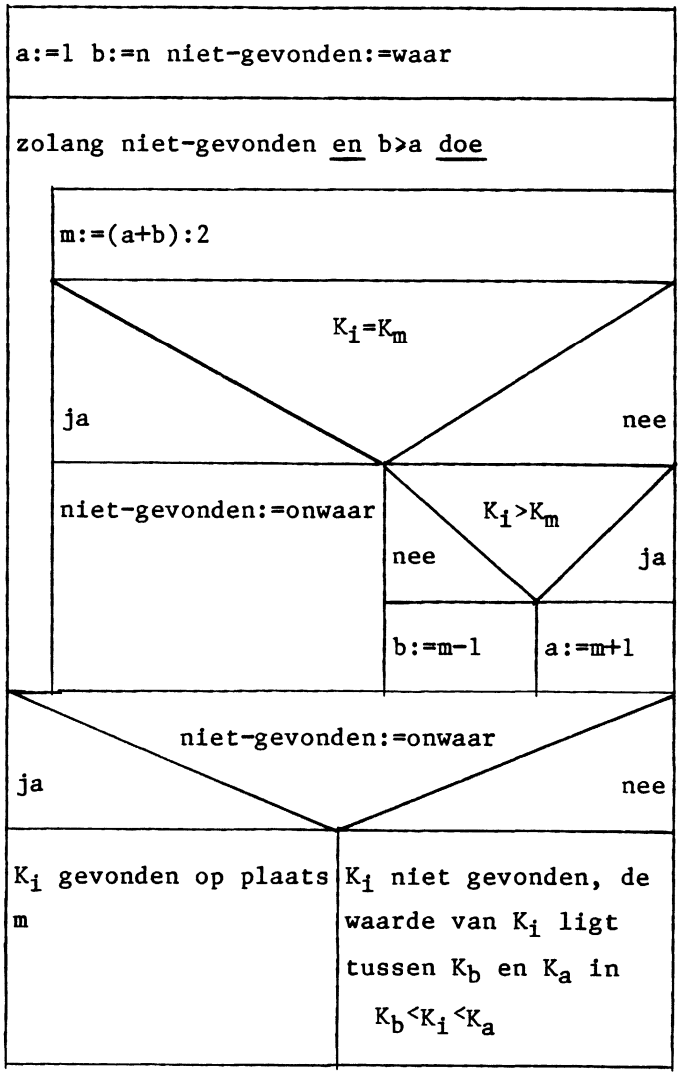
te staan. Het noodzakelijk verplaatsen van praktisch alle records aan de hand van deze tabel is inderdaad mogelijk maar het betekent wel dat we in het achtergrondgeheugen kris kras heen en weer gaan. Met de gebruikelijke insteltijden van schijvengeheugens komen we dan vaak nauwelijks of niet gunstiger uit dan bij andere sorteermethoden.

5. BINAIRE BOMEN

5.1. Binair zoeken in een geordende tabel

Een tabel bevat de records R_1, R_2, ..., R_n met sleutels K_1, K_2, ..., K_n. De records staan gerangschikt naar oplopende volgorde van de sleutels. Het opzoeken van een record met sleutel K_i kan worden gedaan met de binaire zoekmethode.

Het algoritme hiervoor is:



Voorbeeld

Een tabel bevat 11 records met de sleutels

37, 48, 59, 63, 70, 80, 85, 93, 95, 109, 125.

a) Het zoeken van K_i=85.

doorgang	a	b	m	K_m
	1	11		
1	7		6	80
2		8	9	95
3			7	85

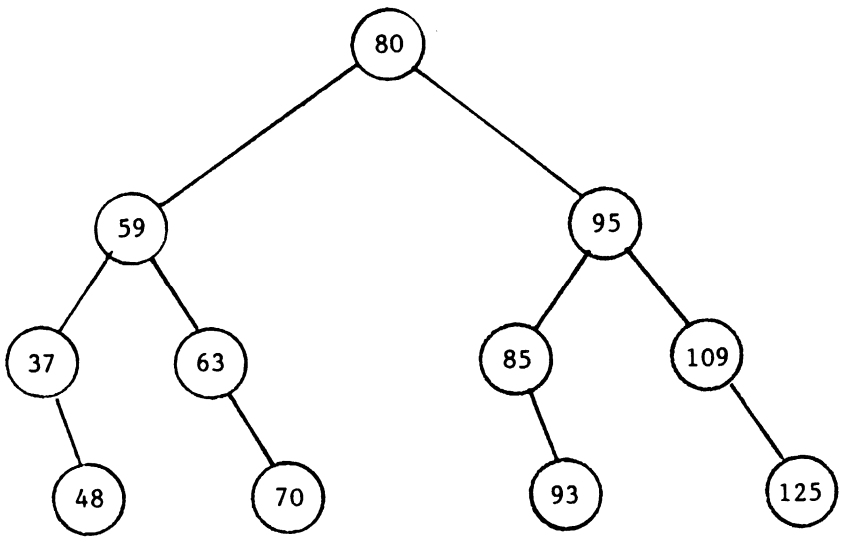
stop, gevonden

b) Het zoeken van K_i=90

doorgang	a	b	m	K_m
	1	11		
1	7		6	80
2		8	9	95
3	8		7	85
4		7	8	93

stop, niet gevonden.

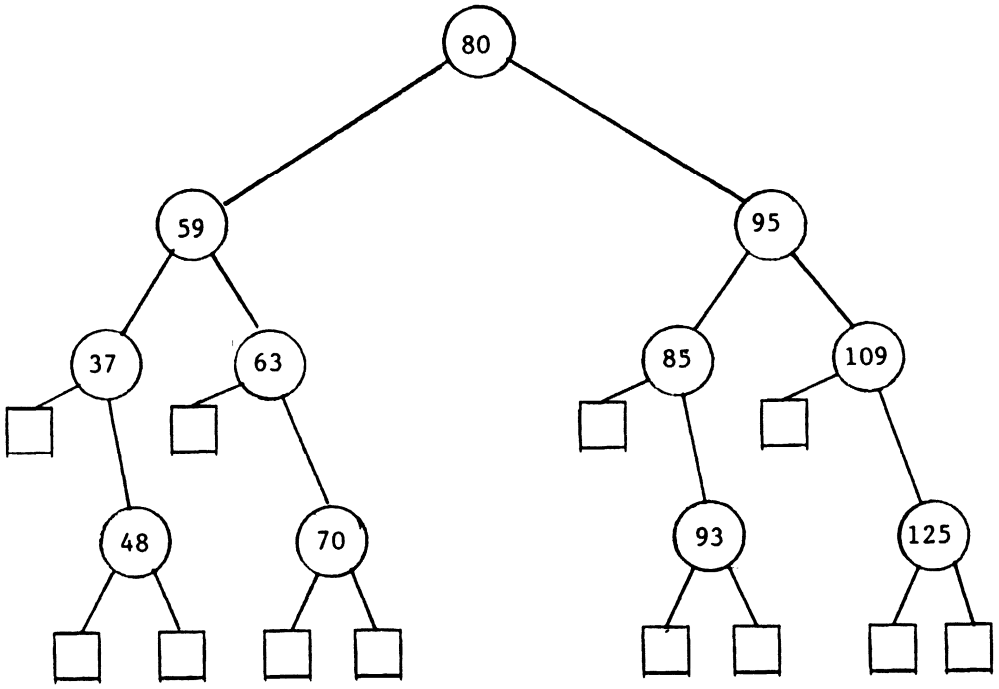
Het binair zoeken in een geordende tabel kan worden vergeleken met het zoeken in een binaire boom. De boom behorend bij bovenstaand algoritme ziet er voor dit voorbeeld als volgt uit:



5.2. Enige definities en eigenschappen m.b.t. binaire bomen

5.2.1. Interne- en externe knopen

Om een meer geschikte structuur te krijgen, breiden we de binaire boom zodanig uit, dat elk knooppunt twee uitgangen heeft. Hiertoe voegen we zgn. externe knopen toe. De boom uit het voorbeeld in 5.1. gaat er dan als volgt uit zien:



In tegenstelling tot de externe knopen, noemen we de andere knopen interne knopen. Een interne knoop heeft dus nu twee uitgangen en een externe geen uitgangen.
Vergelijken we nog eens het binaire zoeken in de tabel met het zoeken in de binaire boom, dan zien we dat het zoeken eindigt bij een interne knoop als het record in de tabel voorkomt, terwijl in het andere geval het zoeken eindigt bij een externe knoop. Het verband tussen het aantal interne knopen en het aantal externe knopen is eenvoudig af te leiden.
Is n het aantal interne knopen en m het aantal externe knopen, dan geldt:

aantal uitgangen is 2n en
aantal ingangen is n+m-1

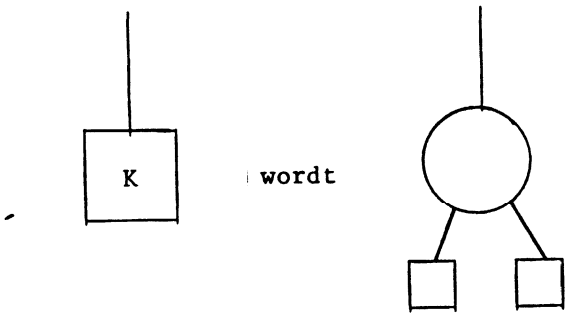
Daar het aantal ingangen gelijk is aan het aantal uitgangen (!), volgt hieruit m=n+1.

5.2.2. Interne- en externe taklengte

Onder de afstand van een knoop tot de wortel verstaaan we het aantal takken dat moet worden doorlopen om van de wortel naar de desbetreffende knoop te komen. Zo heeft de knoop 93 in de boom uit het voorafgaande voorbeeld een afstand 3 tot de wortel.

Onder de interne taklengte I_n verstaan we de som van de afstanden tot de wortel van alle interne knopen.
De externe taklengte E_n is de som van de afstanden tot de wortel van alle externe knopen.
In beide definities is n het aantal interne knopen.

Het verband tussen I_n en E_n is als volgt af te leiden.
Door toevoeging van een record gaat een externe knoop over in een interne:



De nieuwe boom bevat n+1 interne knopen en m+1 externe knopen. Nemen we aan dat K een afstand t tot de wortel had, dan geldt:

$$I_{n+1} = I_n + t$$
$$\text{en } E_{n+1} = E_n - t + 2(t+1) = E_n + t + 2$$

Hieruit volgt:

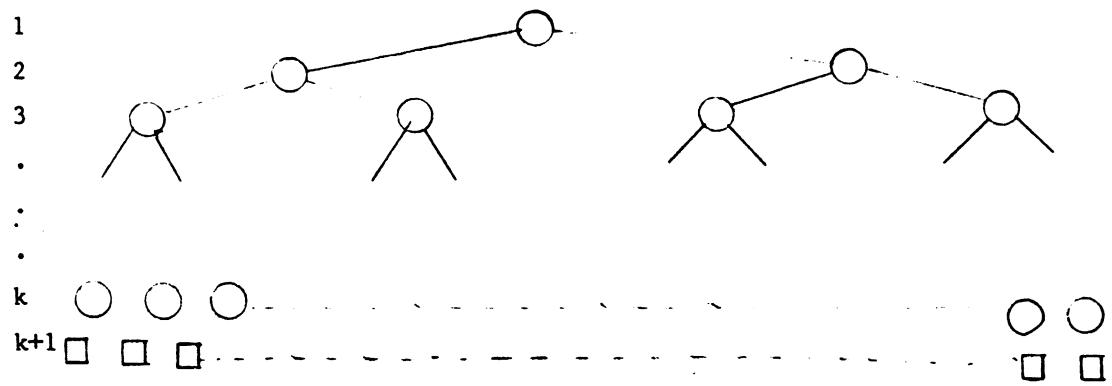
$$E_{n+1} - I_{n+1} = E_n - I_n + 2$$

In verband met $E_0=I_0=0$ is de oplossing van deze recurrente betrekking

$$E_n-I_n = 2n.$$

5.2.3. Volle binaire boom

Voor zowel interne als externe knopen spreken we over niveaus waarmee de relatieve afstand tot de wortel wordt aangegeven. Knopen met gelijke afstand tot de wortel liggen dan op hetzelfde niveau. De niveaus worden genummerd 1, 2, 3 enz. waarbij 1 het niveau van de wortel is.
Een volle binaire boom is een binaire boom waarbij alle externe knopen op één niveau (het hoogste) voorkomen.



We beschouwen een volle boom met k niveaus interne knopen en één niveau (k+1) met alleen externe knopen.
Het aantal interne knopen n is

$$n = \sum_{i=1}^{k} 2^{i-1} = 2^k-1$$

Het aantal externe knopen is $n+1 = 2^k$ en alle externe knopen hebben een afstand k tot de wortel. Daarmee

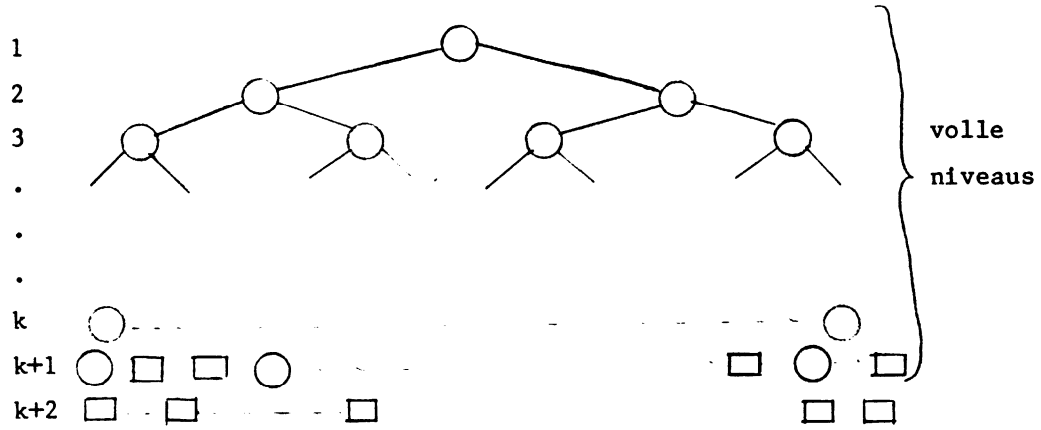
$$E_n = k.2^k$$

en $I_n = E_n-2n = k.2^k-2^{k+1}+2$

Het is gemakkelijk in te zien dat van alle binaire bomen met een aantal interne knopen $n=2^k-1$ de volle boom de boom is waarvoor de interne en daarmee ook de externe taklengte minimaal is.

5.2.4. Complete binaire boom

Bij een willekeurig gegeven aantal interne knopen zal dit aantal in de regel niet gelijk zijn aan een 2-macht -1. Willen we toch een boom met minimale interne en externe taklengte dan komen we vanzelf op een boomfiguratie waarbij de externe knopen zich op twee opeenvolgende niveaus bevinden. Het hoogste niveau is niet vol en bevat alleen externe knopen. Het op een na hoogste is wel vol, maar bevat gedeeltelijk interne en gedeeltelijk externe knopen. Alle andere niveaus bevatten alleen interne knopen. Neem aan dat dit laatste aantal niveau k is.



Een dergelijke boom noemen we een complete binaire boom.

Als n het aantal interne knopen is dan geldt dus:

$$2^k \leq n < 2^{k+1}$$

De (2^k-1) interne knopen die zich in de eerste k niveaus bevinden hebben samen een interne taklengte van $k.2^k-2^{k+1}+2$.
De overige $n-(2^k-1)$ interne knopen op niveau k+1 hebben een afstand k tot

de wortel. Daarmee geldt voor de totale boom:

$$I_n = k.2^k - 2^{k+1} + 2 + k(n-2^k+1) = kn - 2^{k+1} + k + 2$$

en

$$E_n = I_n + 2n = (k+2)(n+1) - 2^{k+1}$$

5.3. C_n en C_n' voor complete binaire bomen

We hebben eerder opgemerkt dat binair zoeken in een geordende tabel vergeleken kan worden met het zoeken in een binaire boom. Voor het volgende noemen we de verwachting voor het aantal zoekstappen bij succesvol zoeken C_n en bij niet succesvol zoeken C_n'.
We veronderstellen dat alle records, dus interne knopen, een gelijke opvragingskans hebben en dat bij niet succesvol zoeken alle n+1 intervallen tussen de sleutels - externe knopen - eveneens even waarschijnlijk zijn.
Er geldt dan voor elke binaire boom:

$$C_n = 1 + \frac{I_n}{n} \quad \text{en}$$

$$C_n' = \frac{E_n}{n+1}$$

Door gebruik te maken van de betrekking $E_n - I_n = 2n$ kan worden afgeleid

$$C_n' = \frac{n}{n+1}(C_n+1).$$

Hieruit blijkt dat voor grote n geldt: $C_n' \sim C_n + 1$.

In verband met de in 6.2.4. gevonden uitdrukkingen voor I_n en E_n geldt voor complete binaire bomen:

$$C_n = 1+\frac{1}{n}(kn-2^{k+1}+k+2) = k+1-\frac{2^{k+1}-k-2}{n} \text{ en}$$
$$C'_n = \frac{1}{n+1}((k+2)(n+1)-2^{k+1}) = k+2-\frac{2^{k+1}}{n+1}$$

Voor k geldt $2^k \leq n < 2^{k+1}$, dus $k = \lfloor {}^2\log\ n \rfloor$.
Stellen we $n = 2^{k+x} (0 \leq x < 1)$, dan is

$$C_n = {}^2\log\ n - 1 + \frac{k+2}{n} + \varepsilon \text{ waarin } \varepsilon = 2-x-2^{1-x}$$

Eenvoudig aan te tonen is dat geldt:

$$0 \leq \varepsilon < (1-\frac{1+\ln\ln 2}{\ln 2}) \sim 0{,}0861$$

Voor C'_n kan worden geschreven:

$$C'_n = {}^2\log\ n + \frac{2^{1-x}}{n+1} + \varepsilon$$

Voor niet te kleine n is dus het resultaat:

$$C_n \sim {}^2\log\ n-1 \text{ en } C'_n \sim {}^2\log\ n.$$

5.4. Binaire bomen als lijststruktuur

5.4.1. Representatie van binaire bomen

Uit het voorafgaande blijkt dat binair zoeken in een geordende tabel een zeer goede nmethode is. Immers bij een gegeven aantal elementen is de binaire boom die impliciet met het zoekalgoritme correspondeert de binaire boom met de kleinste waarde voor C_n. Heeft de tabel echter geen vaste lengte

maar worden geregeld elementen toegevoegd en verwijderd, dan kan het op volgorde houden van de tabel meer kosten dan de binaire zoekmethode aan winst oplevert.

Door in de tabel gebruik te maken van een expliciete binaire structuur kan dit bezwaar worden ondervangen. Hiertoe wordt elk tabelelement uitgebreid met twee verwijzingen, een "linker" en een "rechter" pointer. Deze pointers verwijzen naar de linker en rechter subboom van het betreffende element.

Voor de volgende algoritmen zullen de verwijzingen genoemd worden LADR en RADR en het veld voor de sleutel SLEUTEL.

5.4.2. Initialisatie van de tabel

Het eerste element van de tabel wordt gebruikt om naar het begin van de boom te verwijzen. De vrije elementen - oorspronkelijk dus alle - worden in een keten opgenomen. Voor dit doel wordt gebruik gemaakt van LADR.

De algoritme voor het initialiseren van de tabel ziet er als volgt uit. Het aantal elementen staat in AANTAL.

```
VRIJ:=AANTAL
LADR(VRIJ):=0
zolang VRIJ>2
    LADR(VRIJ-1):=VRIJ
    VRIJ:=VRIJ-1
SLEUTEL(1):=HIGH-VALUE
LADR(1):=0
```

Een tabel bestaande uit 14 elementen ziet er na initialisatie als volgt uit:

ADRES	SLEUTEL	LADR	RADR
1	∞	0	
2		3	
3		4	
4		5	
5		6	
6		7	
7		8	
8		9	
9		10	
10		11	
11		12	
12		13	
13		14	
14		0	

VRIJ wijst naar het begin van de keten van vrije elementen d.w.z. VRIJ = 2.

5.5.3. Opzoeken van een element

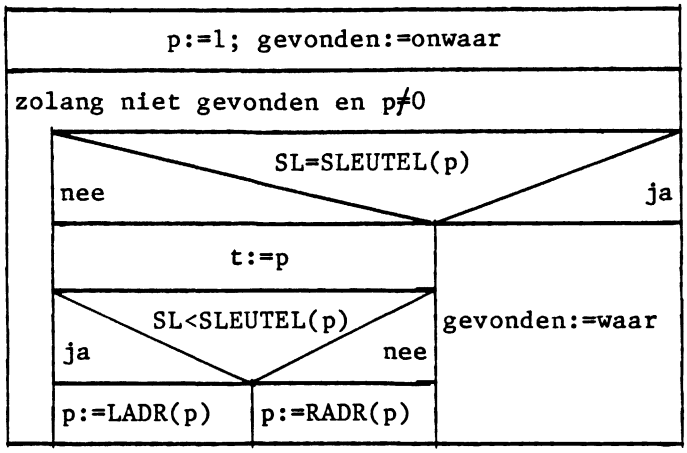
ZOEK(SL, p, t) zoekt het record met sleutel SL en heeft als uitvoer

p: adres van het gezochte record

t: adres "voorganger" van het gezochte record.

Indien het gevraagde record niet aanwezig is, is p=0.

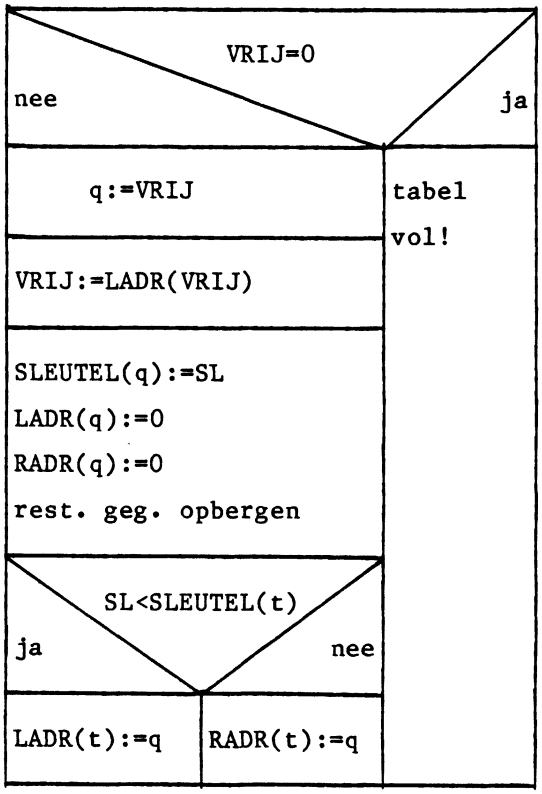
Het structuurdiagram van ZOEK:



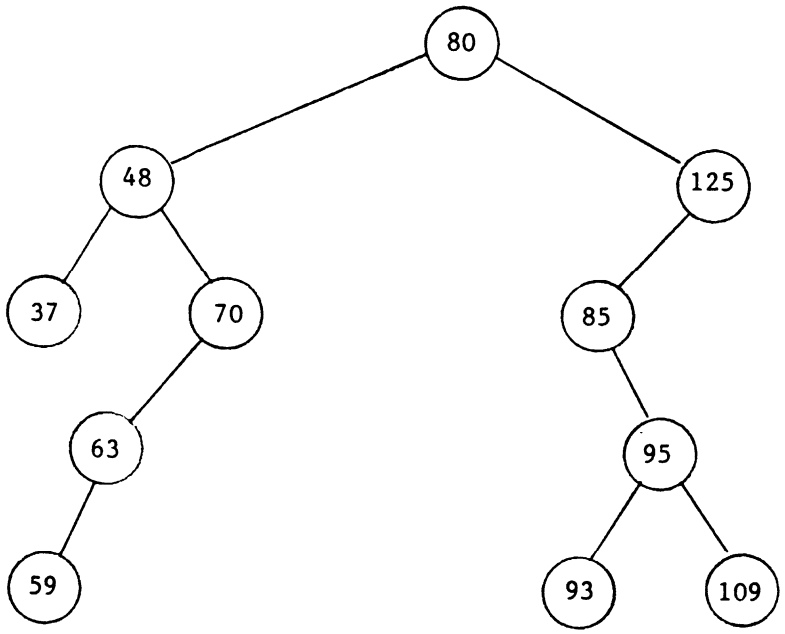
5.4.4. Toevoegen van een element

Indien een record ter toevoeging wordt aangeboden, wordt ZOEK (6.5.3.) gebruikt om de plaats in de tabel te bepalen waar de verwijzingen moeten worden aangepast. Uiteraard wordt het record alleen toegevoegd als het nog niet aanwezig blijkt (p=0).

Het struktuurdiagram voor TOEVOEGEN is dan:



Zouden we b.v. de records met sleutels 80, 125, 48, 85, 70, 37, 95, 109, 63, 59, 93 in deze volgorde aanbieden, dan wordt de boom:



en de tabel:

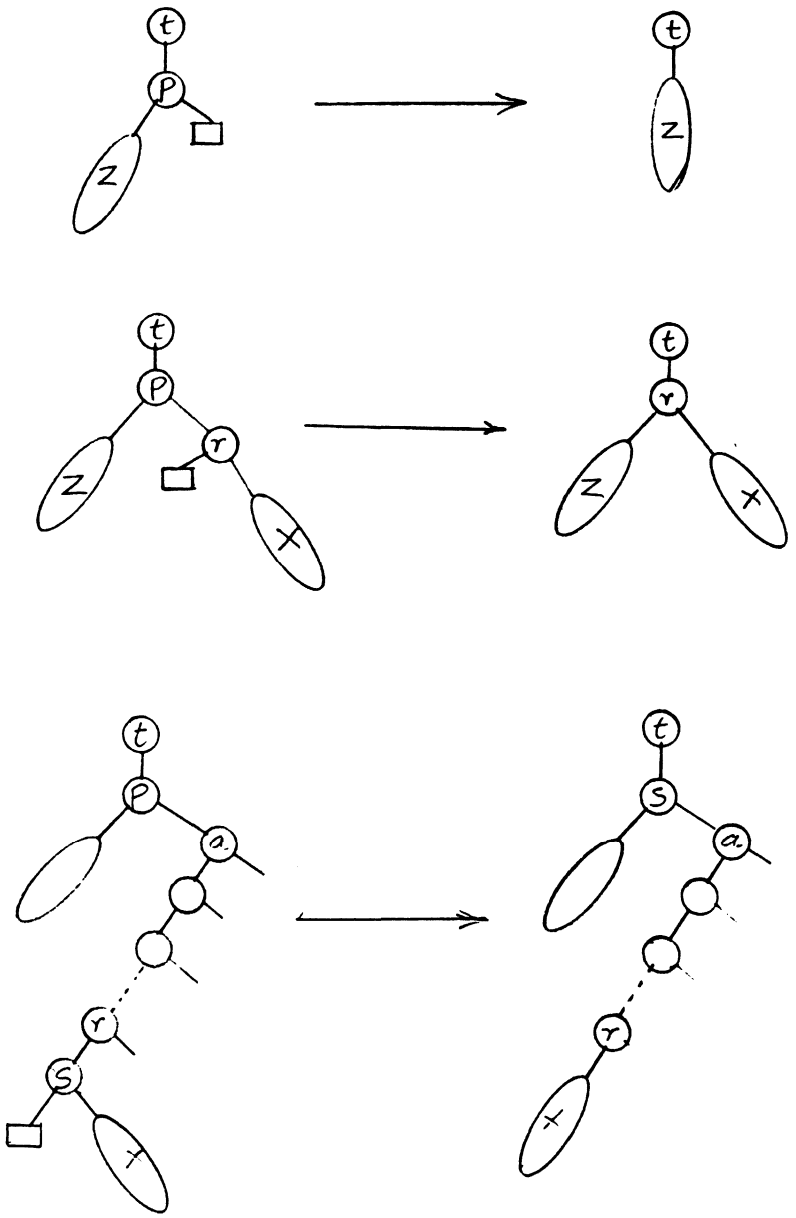
ADRES	SLEUTEL	LADR	RADR
1	∞	2	
2	80	4	3
3	125	5	0
4	48	7	6
5	85	0	8
6	70	10	0
7	37	0	0
8	95	12	9
9	109	0	0
10	63	11	0
11	59	0	0
12	93	0	0
13		14	
14		0	

VRIJ heeft nu de waarde 13.

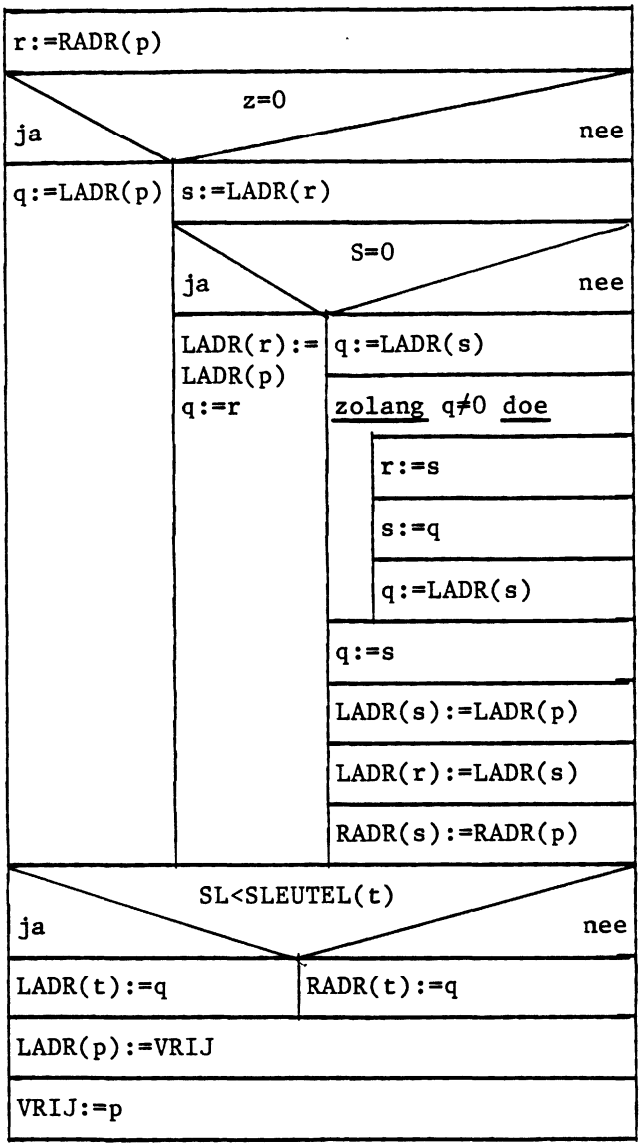
5.4.5. Verwijderen van een element

Indien een element moet worden verwijderd, kan dit in de boom worden vervangen door het kleinste element in de rechtersubboom van het te verwijderen element of door het grootste element in de linkersubboom. Dit geldt uiteraard indien er gekozen kan worden. Verder dienen de verwijzingen te worden aangepast.

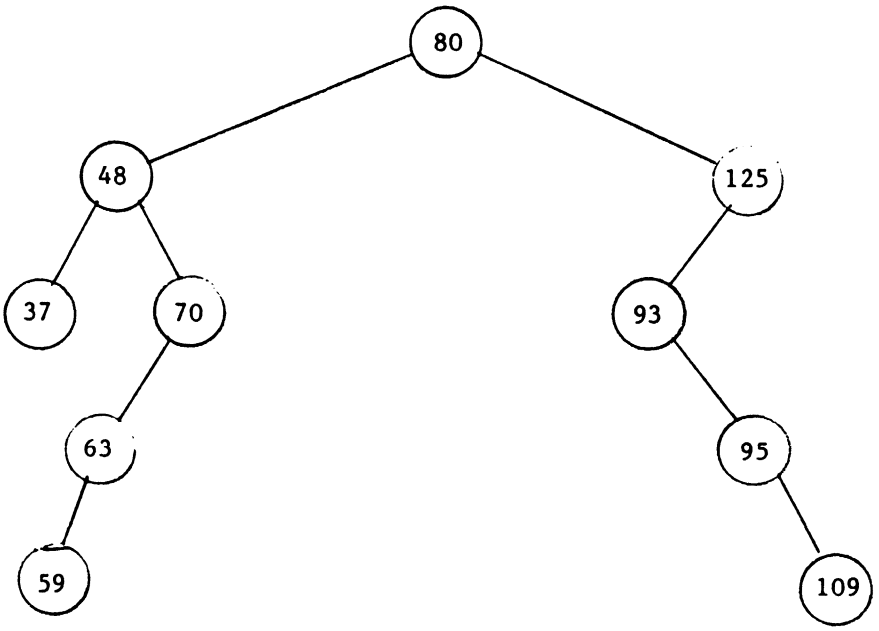
In onderstaande figuren wordt de transformatie schematisch weergegeven. Er is gekozen voor het kleinste element in de rechtersubboom. p stelt het te verwijderen element voor.



Onderstaand diagram geeft het verwijderen van een record weer. De adressen p en t worden vooraf bepaald met ZOEK (5.4.3.).



Zouden we in de boom uit 5.4.3. b.v. het record met sleutel 85 verwijderen, dan wordt de boom:



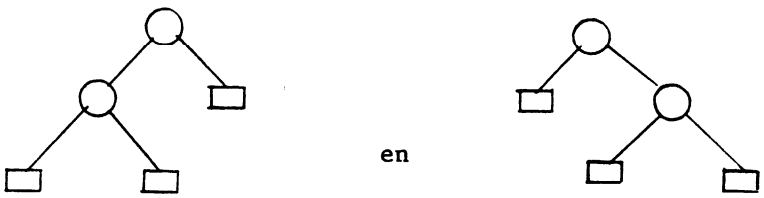
en de tabel:

ADRES	SLEUTEL	LADR	RADR
1	∞	2	
2	80	4	3
3	125	ß 12	0
4	48	7	6
5	85	Ø 13	8
6	70	10	0
7	37	0	0
8	95	12 0	9
9	109	0	0
10	63	11	0
11	59	0	0
12	93	Ø 0	Ø 8
13		14	
14		0	

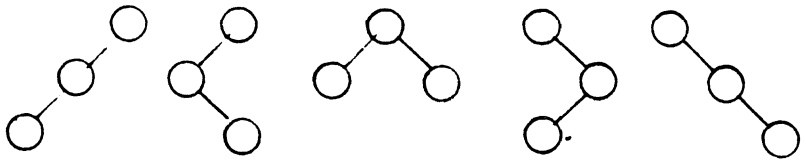
VRIJ = 13 5

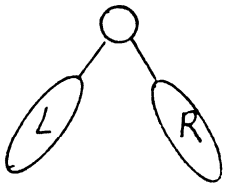
5.5. Aantal binaire boomconfiguraties

Voor een boom met 2 interne knooppunten bestaan 2 mogelijkheden van samenstelling.



Voor een boom met 3 interne knopen zijn dit reeds 5 mogelijkheden.





Dit aantal neemt zeer snel toe voor groter aantal knopen. Noem b_n het aantal boomconfiguraties voor bomen met n interne knopen.

Voor b_n is af te leiden dat

$$b_n = \frac{1}{n+1} \binom{2n}{n}$$

Voor een aantal waarden van n geeft dit

n	b_n
0	1
1	1
2	2
3	5
4	14
5	42
6	132
7	429
8	1430
9	4862
10	16796

Voor grotere n gebruiken we de asymptotische reeksontwikkeling van n

$$n! = \sqrt{2\pi n}\ (\frac{n}{e})^n\ \{1+\frac{1}{12n}+\frac{1}{288n^2}-\frac{139}{51840n^3}\ ....\}$$

Volstaan we met de eerste term dan wordt

$$b_n \sim \frac{4^n}{(n+1)\sqrt{\pi n}}$$

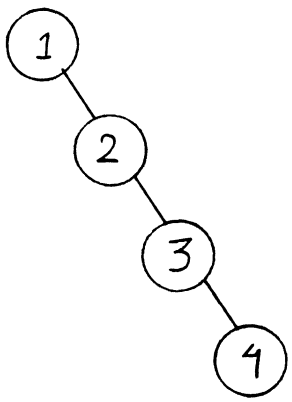
b_n neemt dus exponentieel toe met n

n	b_n
100	$8.98_{10}56$
1000	$2,05_{10}597$
10000	$2,25_{10}6014$

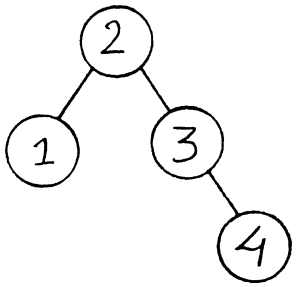
De zoektijd in een binaire boom bij gegeven n kan sterk variëren. Optimaal is de complete boom en het slechtste de volledig uitgestrekte boom. Daarmee varieert C_n van $^2\log n-1$ tot $\frac{1}{2}(n+1)$ en C'_n van $\sim^2\log n$ tot $\sim\frac{1}{2}n$.

Gezien het grote aantal boomconfiguraties bij gegeven n zou men na kunnen gaan hoe de "gemiddelde" configuratie zich gedraagt. Zouden we aannemen dat alle configuraties gelijke kansen hebben dan is aan te tonen dat C_n evenredig wordt met $\sqrt{n}$.

Echter, die aanname is niet zo reëel. Bijvoorbeeld sleutels 1, 2, 3, 4 in die volgorde vormen de boom



De boom



wordt gevormd door de volgorden 2, 1, 3, 4 alsmede 2, 3, 1, 4 alsmede 2, 3, 4, 1.
In vele gevallen is juist de aanname dat alle volgorden gelijke kansen hebben veel aannemelijker, maar dat voert tot een geheel andere kansverdeling voor de boomconfiguraties. Op deze aanname komen we in de volgende paragraaf terug.

5.6. C_n en C'_n voor binaire bomen

5.6.1. Alle externe knopen gelijke kans bij toevoegen knopen

Het toevoegen van een nieuwe interne knoop op basis van gelijke kansen voor iedere externe knoop, dus voor ieder interval, komt overeen met de aanname dat bij opbouw van de boom iedere volgorde van recordaanbieding even waarschijnlijk is.
Dit is als volgt in te zien.
We beschouwen een boom met n knopen en daarbij de sleutels genummerd 1 t/m n. Bij iedere volgorde van de getallen 1 t/m n behoort nu een boom. Stel a_1, a_2, ..., a_n is een permutatie van de getallen 1 t/m n. Aan deze reeks willen we toevoegen een getal m, zo dat $1 \leq m \leq n+1$. Voor dit toevoegen veranderen we de getallen a_1 t/m a_n in a'_1 t/m a'_n zo dat

$a'_i = a_i$ als $a_i < m$

en $a'_i = a_i + 1$ als $a_i \geq m$

Er wordt in de getallenrij dus "ruimte" gemaakt voor m. Door de overgang van a_1 t/m a_n naar a'_1 t/m a'_n is de bijbehorende boom niet veranderd. Beschouwen we de rij

$a'_1, a'_2, \ldots, a'_n, m$

en laten we m achtereenvolgens de range van 1 t/m (n+1)doorlopen dan komt m

dus achtereenvolgens op alle intervallen (externe knopen) van de boom met n knopen terecht.

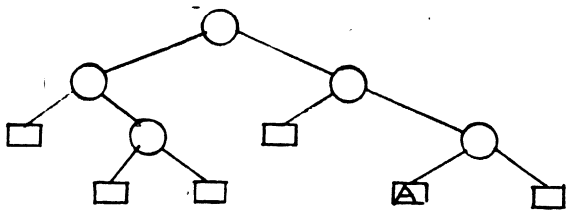
Laten we daarna de getallen a_1 t/m a_n alle permutaties doorlopen, dan krijgen we gecombineerd met alle m waarden alle permutaties van de getallen 1 t/m (n+1) met hun bibehorende bomen.

De (n+1)ste toevoeging heeft daarbij plaats gevonden op basis alle intervallen gelijke kans en het resultaat is hetzelfde als de basis alle (n+1)! permutaties van de (n+1) sleutels gelijke kans.

We nemen nu verder aan dat bij succesvol zoeken alle interne knopen, dus aanwezige records, gelijke opvragingskansen hebben.

Ook bij niet succesvol zoeken is de aanname dat alle externe knopen, dus intervallen in sleutelwaarden, gelijke opvragingskans hebben.

We bekijken een willekeurige boom



Om in deze boom te constateren dat A een extern knooppunt is, zijn 3 stappen nodig, namelijk de 3 voorafgaande interne knopen naar en inclusief de wortel. Voegen we op de plaats A een interne knoop toe en zouden we die later terugzoeken dan zijn daar 4 stappen voor nodig. In het algemeen is voor het terugzoeken van een interne knoop één stap meer nodig dan bij het plaatsen nodig was om te constateren dat het record er nog niet was. Bij onafhankelijke processen en iedere toevoeging is onafhankelijk van de vorige, is de verwachting van de som gelijk aan de som van de verwachtingen en daarmee

$$C_n = 1 + \frac{c'_0 + c'_1 + \ldots c'_{n-1}}{n} \qquad (1)$$

In paragraaf 5.3. is afgeleid dat onder de gegeven condities geldt:

$$C'_n = \frac{n}{n+1}(C_n+1) \qquad (2)$$

Uit (2) volgt

$$C_n = \frac{n+1}{n} C'_n - 1 \qquad (3)$$

(3) in (1) geeft

$$(n+1)C'_n = 2n + \sum_{i=0}^{n-1} C'_i$$

daarmee

$$nC'_{n-1} = 2n-2+ \sum_{i=0}^{n-2} C'_i$$

dus

$$(n+1)C'_n - nC'_{n-1} = 2+C'_{n-1}$$

$$C'_n = C'_{n-1} + \frac{2}{n+1}$$

Als randvoorwaarde geldt $C'_0=0$
dus $C'_1 = \frac{2}{2}$

$$C'_2 = \frac{2}{2} + \frac{2}{3}$$

$$C'_3 = \frac{2}{2} + \frac{2}{3} + \frac{2}{4}$$

$$C'_4 = \frac{2}{2} + \frac{2}{3} + \frac{2}{4} + \frac{2}{5}$$

$$C'_n = \frac{2}{2} + \frac{2}{3} + \frac{2}{4} + \ldots + \frac{2}{n+1}$$

$$c'_n = 2(H_{n+1}-1) = 2H_{n+1}-2$$

M.b.v. (3) is dan

$$C_n = 2(1 + \frac{1}{n})H_n - 3$$

Voor H_n geldt

$$H_n = \ln n + \gamma + \frac{1}{2n} - \frac{1}{12n^2} \ldots \text{ met } \gamma = 0{,}57721566490$$

daarmee

$C_n \sim \ln n + \gamma - 3$

$C_n \sim 2 \ln n - 1{,}8456 = 1{,}3863\ {}^2\log n - 1{,}8456$

Vergelijk dit met de complete boom waarvoor geldt

$C_n \sim {}^2\log n - 1$

Enkele waarden

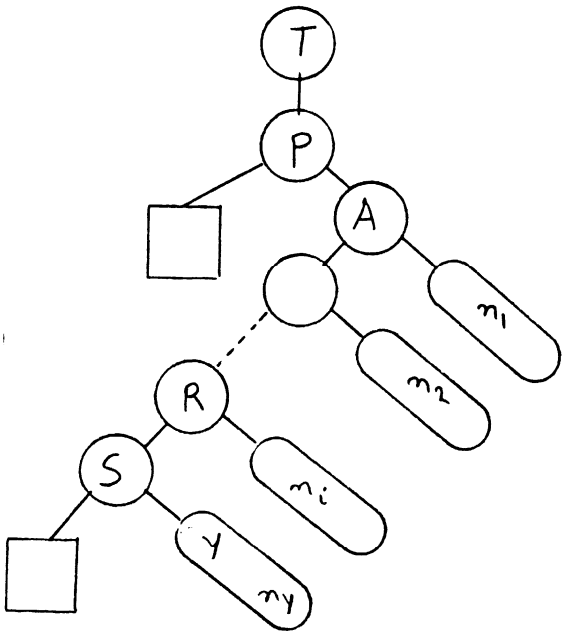
n	C_n	C_n (complete boom)
100	7,36	6,64
10^3	11,97	9,97
10^4	16,58	13,29
10^5	21,18	16,61

5.7. Verbeterd verwijder algoritme

In de vorige paragraaf is nagegaan wat de verwachting is voor succesvol en niet succesvol zoeken als opbouw van de binaire boom plaatsvindt op basis van alle volgorden bij aanbieden van records gelijke kans. Hiermee kwam overeen het toevoegen op basis van alle intervallen dus externe knopen gelijke kans.

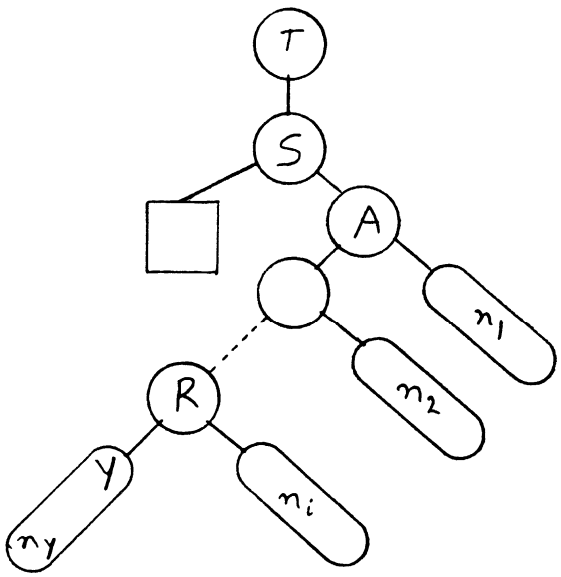
De vraag is wat gebeurt er bij weglaten. Hierbij nemen we aan dat alle aanwezige records gelijke kans hebben om verwijderd te worden. Dit laatste

betekent dus alle interne knopen gelijke kans bij verwijdering.
Reeds in 1962 is door T.N. Hibbard aangetoond dat onder deze aanname en bij gebruik van het weglaatalgoritme beschreven in paragraaf 5.4.5 ook dan de formules voor C_n en C_n' blijven gelden.
Dit laatste impliceert dat zelfs een geringe verbetering bij weglaten mogelijk is. Beschouw het geval dat de linkersubboom van het te verwijderen element leeg is.



n_1, n_2, ..., n_i en n_y zijn het aantal interne knopen in de subbomen.

Verwijdering van P met het oorspronkelijke algoritme geeft:

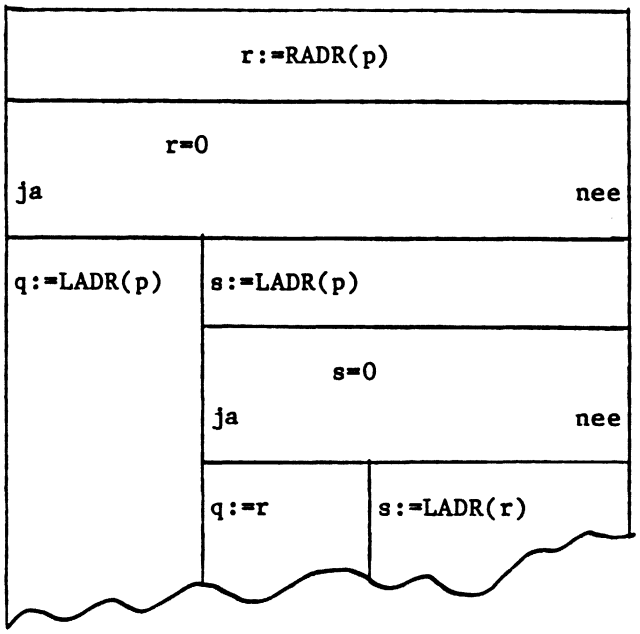


De afstand van S tot de wortel is met i+1 verminderd en de afstand van de

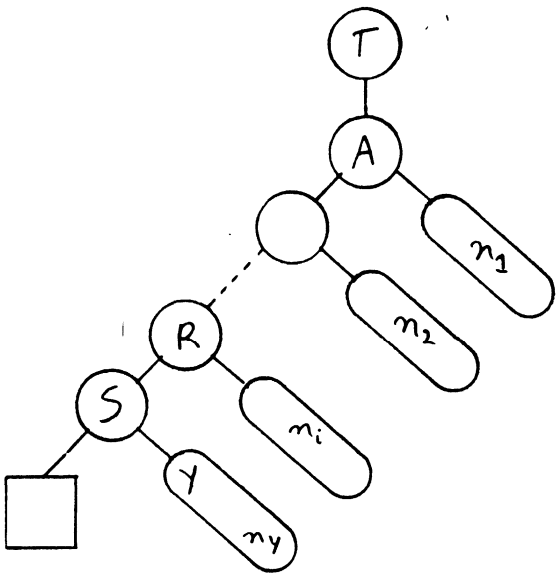
knopen in subboom Y met 1. De interne taklengte is dus met i+1+n afgenomen. Uiteraard moet ook de bijdrage van P aan de interne taklengte in mindering worden gebracht.

Het algoritme wordt als volgt gewijzigd:

Als de rechtersubboom van het te verwijderen element niet leeg is en de linkersubboom wel, dan wordt de rechtersubboom in zijn geheel gekoppeld aan het voorafgaande element. In schema is de wijziging:



Verwijdering van P met het gewijzigde algoritme geeft:



De afname van de interne taklengte is (exclusief afname door verwijdering van P):

$$n_1+1+n_2+1+\ldots+n_i+1+n_y+1 = i+1+n_1+n_2+\ldots+n_y.$$

Vergeleken met het oorspronkelijke algoritme is er een extra afname van de interne taklengte van $n_1+n_2+\ldots n_i$. Dit betekent dat ook C_n is afgenomen. Wanneer we aan een binaire boom regelmatig elementen toevoegen en weglaten volgens het aangepaste algoritme dan ontstaan binaire bomen die iets meer in evenwicht zijn en daardoor gemiddeld iets betere zoektijden hebben. De verbetering bedraagt overigens maar enkele grootten.

5.8. AVL-bomen

5.8.1. Inleiding

Door het toevoegen en verwijderen van elementen in een binaire boom kan deze scheef groeien. De gemiddelde zoektijd wordt hierdoor nadelig beïnvloed. Bij gelijke opvragingskansen van de sleutels geeft een complete boom een minimale C_n. Men zou bij elke toevoeging of verwijdering de boom zo kunnen reorganiseren, dat ze compleet blijft. Deze reorganisaties kosten i.h.a. veel werk.
Een soort bomen die dit bezwaar wat ondervangt en waarbij toch de hoogte beperkt blijft zijn de gebalanceerde bomen. Een gebalanceerde boom houdt het midden tussen een optimale boom en een binaire boom zonder beperkingen.

Een bijzonder geval van de gebalanceerde boom is de AVL-boom, genoemd naar de russische wiskundigen G.M. Adel'sen-Vel'skii en E.M. Landis (1962). De definitie van een AVL-boom is:
Een binaire boom is een AVL-boom als voor ieder knooppunt geldt dat de absolute waarde van het verschil in hoogte tussen de linker- en de rechtersubboom kleiner dan of gelijk is aan 1.

Het zal duidelijk zijn dat ook in een AVL-boom een element i.h.a. niet zonder meer kan worden toegevoegd of worden verwijderd, omdat dan niet altijd meer aan genoemde voorwaarde is voldaan. In sommige situaties moet een beperkt deel van de boom worden gereorganiseerd. Om deze situaties te kunnen

onderkennen, wordt ieder knooppunt uitgebreid met een grootheid die we balans zullen noemen.

Balans = 0 : linker- en rechtersubboom hebben gelijke hoogte.
Balans = -1: linkersubboom 1 hoger dan rechtersubboom.
Balans = +1: rechtersubboom 1 hoger dan linkersubboom.

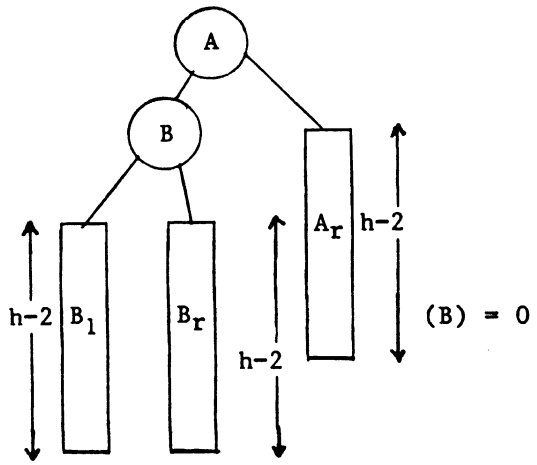
5.8.2. Het toevoegen van element

Indien een element aan de boom wordt toegevoegd, zal er in bepaalde omstandigheden vanaf een zeker knooppunt een reorganisatie van de boom moeten plaatsvinden.
Er moet een reorganisatie plaatsvinden indien er een knooppunt is, waarvan de balans -1 is en de hoogte van de linkersubboom 1 groter wordt, of de balans +1 is en de hoogte van de rechtersubboom 1 groter wordt.
Gaan we vanuit het toegevoegde knooppunt naar de wortel dan is het eerste knooppunt waarvan de balans niet nul is het kritische knooppunt. Op dit niveau wordt beslist of een reorganisatie nodig is. Heeft het kritische knooppunt een balans +1 en wordt de hoogte van de linkersubboom één groter of is de balans -1 en de hoogte van de rechtersubboom wrodt één groter, dan wordt de balans 0 en is er geen reorganisatie nodig, want de hoogte van de boom waarvan het kritisch knooppunt de wortel is, is niet toegenomen.

Heeft het kritische knooppunt een balans +1 en wordt de hoogte van de rechtersubboom één groter of is de balans -1 en wordt de hoogte van de linkersubboom één groter dan is een reorganisatie noodzakelijk.
Van een knooppunt gelegen op het kritische pad (tussen het toegevoegde en kritische knooppunt) is de balans nul. De balans van zo'n knooppunt wordt +1 of -1 afhankelijk van de subboom, waarin het nieuwe knooppunt is opgenomen. De hoogte van de boom waarvan het beschouwde knooppunt de wortel is neemt dus altijd met één toe. Dit betekent dat het naast hogere knooppunt uit balans zou kunnen raken. Dit verschijnsel treedt net zolang op totdat we het kritische knooppunt hebben bereikt.

We onderscheiden twee soorten reorganisaties, te weten:

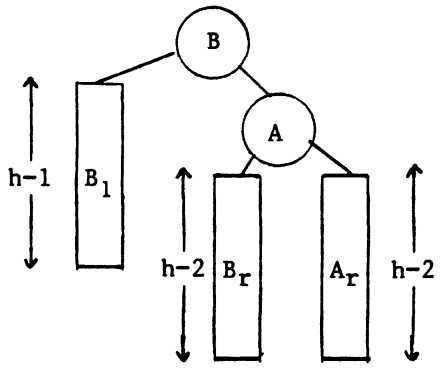
1.



hoogte(A) =h
hoogte (A_r) = h-2
hoogte (A_l) = h-1
balans (A) = -1
hoogte (B_l) = h-2
hoogte (B_r) = h-2

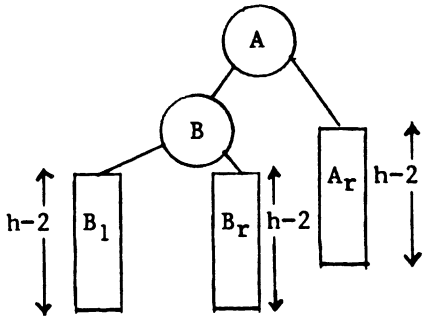
Het nieuwe knooppunt komt in de linkersubboom van B(B_l) en de hoogte van B_l neemt met 1 toe. De hoogte van de boom met B als wortel is h-1 en wordt dus h, waardoor A niet meer aan de eisen voldoet. De reorganisatie die plaatsvindt is een enkelvoudige rotatie van A en B.

De nieuwe boom wordt:

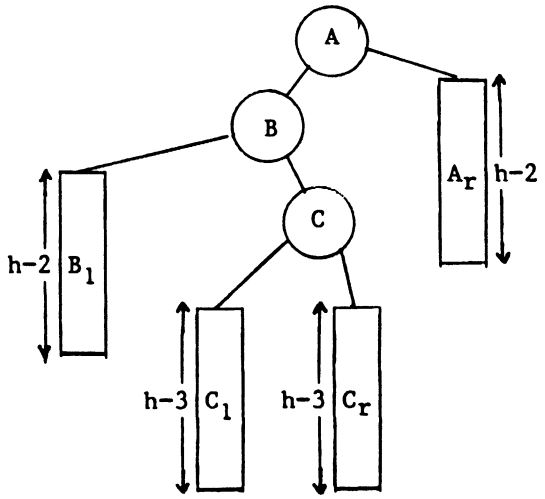


hoogte (A) = h-1
balans (A) = 0
hoogte (B_l) = h-1
hoogte (B) = h
balans (B) = 0

Voor de knooppunten oorspronkelijk gelegen boven A verandert er niets, want in de oude boom was de hoogte (A) = h en in de nieuwe boom is hoogte (B) = h.

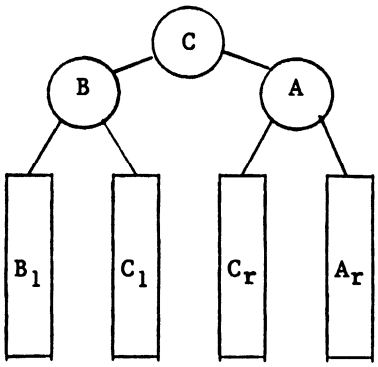


Het nieuwe knooppunt komt in de rechtersubboom van B(B_r) en de hoogte van B_r neemt met 1 toe. De balans van B wordt hierdoor +1 en A voldoet niet meer aan de voorwaarde.
Bij de reorganisatie wordt nu ook de wortel van B_r betrokken. We beschouwen daarom de boom:



Het toe te voegen knooppunt komt dus of in C_l of in C_r.

Na een dubbele rotatie wordt de nieuwe boom:



hoogte (C) = h
balans (C) = 0
hoogte (B) = h-1
hoogte (A) = h-1

Als het toegevoegde knooppunt in C_l is opgenomen dan geldt:

hoogte (C_l) = h-2
balans (B) = 0
hoogte (C_r) = h-3
balans (A) = +1

Als het toegevoegde knooppunt in C_r is opgenomen dan geldt:

hoogte (Cl) = h-3
balans (B) = -1
hoogte (C_r) = h-2
balans (A) = 0

Voor de knooppunten oorspronkelijk gelegen boven A verandert er niets, want in de oude boom is de hoogte (A) = h en in de nieuwe boom is hoogte (C) = h.

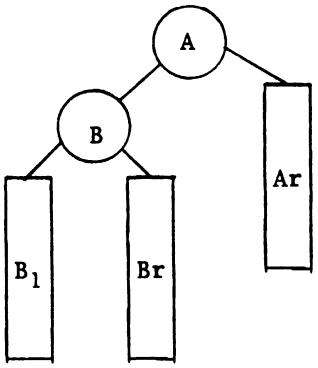
Een belangrijke conclusie is:

Bij toevoeging van een knooppunt is hoogstens één reorganisatie noodzakelijk.
Uiteraard geldt dezelfde redenatie ook bij toevoeging in de rechtertak van A.

5.8.3. Het verwijderen van een element

Het verwijderen van een knooppunt uit een AVL-boom verloopt in eerste instantie hetzelfde als het verwijderen uit een binaire boom. Het te verwijderen knooppunt wordt vervangen door hetzij het kleinste knooppunt in de rechter-subboom, hetzij het grootste knooppunt in de linkersubboom. Dit knooppunt wordt toegevoegd op de plaats van het te verwijderen knooppunt en verwijderd uit zijn oorspronkelijke plaats. Deze handeling kan tot gevolg hebben dat de hoogte van sommige subbomen afneemt. Hierdoor kunnen knooppunten ontstaan die niet meer aan de voorwaarden voldoen.

Beschouwen we de subboom:



balans (A) = -1
hoogte (A_r) = h-2
hoogte (A_l) = h-1

We veronderstellen dat uit A_r een knooppunt wordt verwijderd en dat de hoogte van A_r hierdoor afneemt. De hoogte van A_r wordt h-3. Het knooppunt A voldoet dan niet meer aan de voorwaarde. Een reorganisatie is dus noodzakelijk.
Er zijn nu drie mogelijkheden:

1. De balans van B is -1 dus hoogte (B_l) =h-2 en hoogte (B_r) = h-3. In dit geval vindt een enkele rotatie plaats. Hierna is hoogte (A_r)= h-3, hoogte (A) = h-2 en hoogte (B) = h-1. De oorspronkelijke hoogte van de subboom was h, m.a.w. de hoogte van de subboom is met 1 afgenomen. Hierdoor kan er hoger in de boom weer een reorganisatie nodig zijn.

2. De balans van B is 0 dus hoogte (B_l) = h-2 en hoogte (B_r) = h-2. Ook in dit geval vindt een enkele rotatie plaats. Hierna is hoogte (A_r) = h-3, hoogte (A) = h-1 en hoogte (B) = h. De hoogte van de nieuwe subboom is dus gelijk aan de hoogte van de oorspronkelijke subboom. Er is dus geen verdere reorganisatie nodig.

3. De balans van B is +1, dus hoogte (B_l) = h-3 en hoogte (B_r) = h-2. Er vindt nu een dubbele rotatie plaats. Hierna is de hoogte (A_r) = h-3, hoogte (B) = h-2, hoogte (A) = h-2 en hoogte (C) = h-1.
   Ook in dit geval is de hoogte van de nieuwe subboom 1 kleiner dan die van de oorspronkelijke boom. Ook hier kan dit reorganisaties hoger in de boom tot gevolg hebben.

Conclusie:

Bij verwijdering van een knooppunt kunnen meer reorganisaties nodig zijn, hoogstens net zo veel als de boom hoog is.

Later zullen we zien dat de hoogte van de boom van de orde $^2\log n$ is.

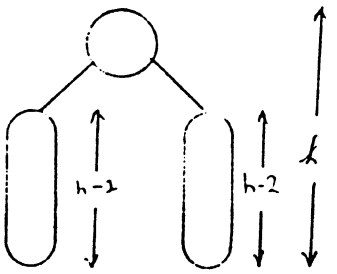
5.8.4. Kwantitatief gedrag van AVL bomen

De vraag is bij een gegeven aantal records georganiseerd volgens een AVL boom wat de minimale en maximale hoogte van een dergelijke boom kan zijn en zo mogelijk een verwachting voor succesvol en niet succesvol zoeken.
Wat minimale en maximale hoogte betreft is deze vraag niet direct te beantwoorden. Echter wanneer we de vraag omdraaien dan wordt een antwoord iets eenvoudiger. Namelijk gegeven een hoogte h wat is dan het maximale en minimale aantal records dat in een AVL boom kan worden ondergebracht.
Het maximale aantal is eenvoudig. Immers, een AVL boom die maximaal gevuld is is een volle boom en dan geldt:

$$n = 2^h-1$$

dus het maximale aantal interne knopen is 2^h-1. Dat wil ook zeggen dat bij gegeven aantal records de hoogte $h \geq {}^2\log (n+1) > {}^2\log n$.
Voor een minimaal gevulde AVL boom geldt dat deze boom zelf en ook alle subbomen minimaal gevuld zijn.

Noem bij hoogte h het minimaal aantal knopen N_h, dan geldt:



$N_h = N_{h-1} + N_{h-2} + 1$ met $N_0 = 0$ en $N_1 = 1$

Enkele waarden voor N_h:

N	N_h
0	0
1	1
2	2
3	4
4	7
5	12
6	20
.	.
.	.
.	.

De algemene oplossing voor N_h levert

$$N_h = 1{,}1708(1{,}6180)^h - 0{,}1708(-0{,}6180)^h - 1$$

Voor grotere waarden is N_h praktisch gelijk de eerste term waarmee

$$n > 1{,}1708(1{,}6180)^h$$

Gecombineerd met $h > {}^2\log n$ geldt

$${}^2\log n < h < 1{,}44\ {}^2\log n - 0{,}328$$

Voor C_n en C_n' is af te leiden dat

$${}^2\log n < C_n' < 1{,}0423\ {}^2\log n - 0{,}007964$$

en

$${}^2\log n - 1 < C_n < 1{,}0423\ {}^2\log n - 1{,}007964.$$

Het verschil tussen maximum en minimum is dus gering.

Uit simulatieproeven blijkt dat C_n en C_n' ook in dit bereik liggen en een zeer kleine spreiding vertonen.

Enkele waarden

	min	max	sim	min	max	sim
n	C_n	C_n	C_n	C'_n	C'_n	C'_n
500	7,97	9,34	8,19	8,97	10,34	9,17
1000	8,97	10,38	9,20	9,97	11,38	10,19
2000	9,97	11,42	10,21	10,97	12,42	11,21

Bij deze simulaties is ook het aantal noodzakelijke rotaties bij toevoegen en verwijderen gemeten nog gesplitst naar enkele en dubbele rotaties. Dit aantal blijkt vrijwel onafhankelijk van n.

rot/mut	enkel	dubbel	totaal
toevoegen	0,230	0,228	0,458
verwijderen	0,138	0,083	0,221
of omgekeerd			
mut/rot	enkel	dubbel	totaal
toevoegen	4,35	4,39	2,18
verwijderen	7,25	12,0	4,52

Merkwaardig is dus dat hoewel bij verwijderen meer dan een reorganisatie noodzakelijk kan zijn, gemiddeld bij verwijderen minder reorganisaties nodig zijn dan bij toevoegen.

5.8.5. Hoogte gebalanceerde bomen

De AVL boom is te generaliseren door het maximum verschil in hoogte van linker en rechter subbomen niet te beperken tot 1 maar tot k.
Het is duidelijk dat een dergelijke boom meer scheef kan groeien dan een AVL boom, maar vanzelfsprekend zijn bij mutaties minder reorganisaties nodig. De maximum hoogte is weer te vinden door te bepalen het minimum aantal knooppunten bij gegeven hoogte.
Noemen we dit N_h bij hoogte h dan geldt nu

$$N_h = N_{h-1}+N_{h-k-1}+1 \text{ voor } h>k \text{ en } N_h=h \text{ voor } 0\leq h\leq k$$

Hiermee is ook de maximale hoogte h te berekenen.

In volgende tabel is voor enkele waarden van k die maximale hoogte gegeven.

k	maximale h
1	$1{,}4404\ {}^{2}\log n - 0{,}3277$
2	$1{,}8134\ {}^{2}\log n - 0{,}7133$
3	$2{,}1507\ {}^{2}\log n - 1{,}1308$
5	$2{,}7625\ {}^{2}\log n - 2{,}0250$
10	$4{,}0983\ {}^{2}\log n - 4{,}4514$
20	$6{,}3959\ {}^{2}\log n - 9{,}6977$

6. DIRECTE ORGANISATIE

6.1. Inleiding

Voor vele toepassingen is het wenselijk en veelal nodig dat records in willekeurige volgorde snel kunnen worden opgevraagd, zonodig gewijzigd en weer worden teruggeplaatst. Hiervoor is nodig dat het achtergrondgeheugen direct toegankelijk en adresseerbaar is. Pas met de komst van de schijvengeheugens werd deze werkwijze mogelijk.
In principe zijn er twee mogelijke werkwijzen. Ten eerste via een systeem van indextabellen of boomstrukturen, waardoor in een klein aantal stappen het gewenste record bereikt kan worden. Deze methoden worden behandeld in hoofdstuk 8.
De tweede methode is een rechtstreekse of directe relatie te brengen tussen sleutel en adres van de plaats waar het record is opgeborgen. Dit noemt men de directe organisatie.

6.2. Algemene begrippen bij directe organisatie

6.2.1. Absolute en relatieve adressen

Bij de relatie sleutel-adres maken we voor het adres eerst nog onderscheid in absoluut en relatief adres. Met absoluut adres wordt bij schijvengeheugens dan bedoeld volumenummer, cilindernummer, spoornummer en eventueel sectienummer. De preciese bepaling daarvan vindt in de praktijk altijd plaats door het bedrijfssysteem dat op de betrokken computer gebruikt wordt. Voor de gebruiker wordt in de regel via stuurinformatie een aansluitend geheugengebied ter beschikking gesteld waarbij een relatieve nummering in sporen of eventueel blokken of sectoren wordt gebruikt. Dit noemen we de relatieve adressen. De omzetting van relatief adres naar absoluut adres gebeurt dan door het bedrijfssysteem. De transformatie van sleutel naar relatief adres behoort tot de problematiek van de directe bestandsorganisaties.
Waar in volgende paragrafen met relatieve adressen wordt gerekend denken we ons deze relatieve adressen genummerd van 1 t/m M en spreken daarbij dan alleen nog kortweg over adressen.

6.2.2. Relatie sleutel-adres

De eenvoudigste en indien mogelijk ook aantrekkelijkste vorm is een lineaire relatie tussen adres A en sleutel S dus

$$A = p*S+q$$

waarin p en q constanten.

Aangezien we een aaneengesloten adresgebied van 1 t/m M hebben legt dit dus speciale eisen op aan de sleutelwaarden. Deze methode is alleen bruikbaar als we nog vrij zijn in de sleutelbepaling zelf, m.a.w. dat we de sleutels in alle records nog naar eigen inzicht kunnen vaststellen. Dit is bijna nooit het geval zodat deze methode slechts zelden bruikbaar is. Wanneer hij echter gebruikt kan worden dan is deze relatie zeer aantrekkelijk omdat verschillende problemen die hierna nog behandeld worden dan totaal vermeden worden.

In de meeste toepassingsgevallen is de sleutelsamenstelling gegeven. Dat kan een rekeningnummer of artikelnummer zijn. Veelal is een sleutel ook van alphanumerieke aard, bijvoorbeeld persoonsnamen, adressen, namen van artikelen, voorwerpen enz. De relatie tussen sleutel en adres moet dan op andere wijze worden vastgelegd. Een mogelijkheid is soms het bijhouden van een tabel die alle combinaties sleutel-adres bevat. De vraag is hoe men deze tabel dan weer organiseert. Als een geordende tabel, een binaire boom of ook zelf weer direct op de nog te bespreken wijze. In feite heeft men met een tabel het bestandsorganisatieprobleem teruggebracht tot een bestand met kleine omvang records. Overigens, de in paragraaf 6.7. nog te bespreken index-directe organisatie is hiervan een voorbeeld.

Tenslotte de algemeen gebruikelijke methode, die vaak ook als enige "echte" directe organisatie wordt aangegeven.

Men spreekt dan van functionele relatie. De sleutel wordt als numeriek gegeven opgevat. Ook alphanumerieke sleutels worden via interne binaire representaties als getallen gehanteerd. Op de sleutel wordt een numerieke functie losgelaten waaruit een getal onstaat binnen het gegeven adresbereik dus in ons geval in het bereik 1 t/m M.

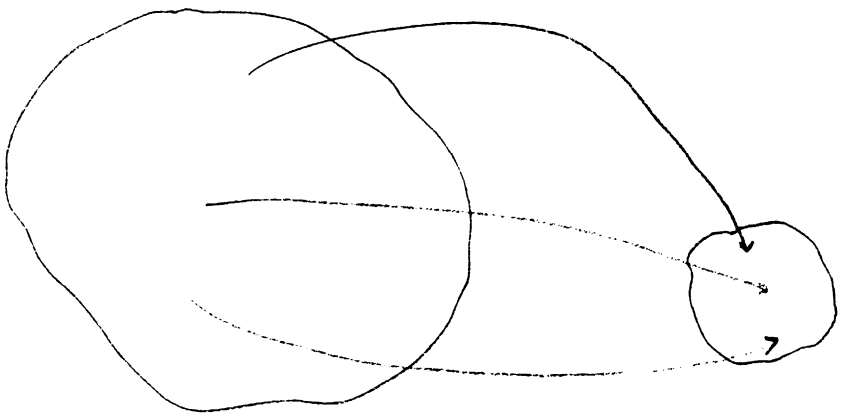
Is A weer het adres en S de sleutel dan is dus

$$A = h(S)$$

De functie h noemt men wel de hashfunctie en het proces van omzetting hashing. Methoden voor hashing worden behandeld in paragraaf 6.3. waarbij verschillende functies voor h worden aangegeven.

6.2.3. Synoniemen en overloop

Bij de functionele relatie wordt een hashfunctie h zo gekozen dat iedere mogelijke sleutel eenduidig wordt omgezet naar een adres in het gewenste bereik. De functie h is dus een afbeeldingsfunctie van ieder punt uit het sleutelgebied naar een punt in het adresgebied.



sleutelgebied adresgebied.

Daarbij is het sleutelgebied vele malen groter dan het adresgebied. Dat groter zijn is in regel ook orden groter. Bijvoorbeeld een bestand van 100.000 records moet worden ondergebracht.In verband met mutaties en groei kiest men voor het adresgebied een ruimte voor 150.000 records. De sleutel bestaat uit namen met alleen letters waarvan het aantal 30. De omvang van het sleutelgebied is dan $26^{30} = 2{,}8_{10}42$.
Aan dit eenvoudige maar zeker karakteristieke voorbeeld onderkent men een aantal problemen. In de eerste plaats zullen vele punten uit het sleutelgebied worden afgebeeld op hetzelfde punt in het adresgebied. Sleutels die na hashing verwijzen naar hetzelfde adres noemt men synoniemen. Ondanks het feit dat we er van uitgaan dat alle records een verschillende sleutel hebben (zie ook paragraaf 1.9.) kan het toch voorkomen dat sommige records op

hetzelfde adres thuishoren omdat de sleutels door de gebruikte hashfunctie synoniemen zijn. Een van die synoniemen, in de regel het eerst aangeboden record, kan op het juiste adres staan en dit adres wordt daarom vaak aangegeven met de naam huisadres. De overige synoniemen moeten ergens anders geplaatst worden. Men spreekt van overloop. Soms wordt daarvoor een afzonderlijk deel van het geheugen aangewezen, dat met overloopgebied wordt aangeduid. Het gebied met alle huisadressen heet primair gebied. Soms heeft men geen apart overloopgebied maar gebruikt men eenvoudig andere lege plaatsen in het primaire gebied.
Een tweede probleem bij de afbeelding is de gelijkmatigheid. Wanneer we als primair gebied een bepaalde ruimte ter beschikking stellen willen we dat de mogelijke sleutels zo gelijkmatig mogelijk over het primaire gebied worden verdeeld. Dit laatste ondanks het feit dat sleutels in de regel zeer inhomogeen over het sleutelgebied zijn verdeeld en vaak in clusters voorkomen. Ook al construeert men een ideale hashfunctie die het gehele sleutelgebied met inachtname van de ongelijke verdeling netjes homogeen afbeeldt op het adresgebied dan nog treden synoniemen op. Immers, de werkelijk optredende sleutels zijn als het ware een steekproef uit het gehele sleutelgebied. Daarbij treden de normale statistische afwijkingen op ten opzichte van de verwachting. Men moet dus altijd rekening houden met overloop. Deze overloop kan op verschillende manieren worden gerealiseerd.

Ten eerste door herhaald hashing. Men heeft een aantal hashfuncties $h_1(s)$. $h_2(s)$; $h_3(s)$ enz. Daarmee worden achtereenvolgens de adressen A_1, A_2, A_3 enz. bepaald. Zodra een vrij adres gevonden is stopt het proces en wordt het record daar geplaatst. Deze methode wordt nader uitgewerkt in paragraaf 6.4.

Een tweede methode is met kettingadressen. Wanneer het huisadres is bezet wordt een vrij adres in het overloopgebied gekozen. Op het huisadres staat reeds een record en hieraan wordt een adresveld toegevoegd dat verwijst naar het record in overloop. Komen er meer synoniemen voor hetzelfde huisadres dan wordt de ketting verlengd. Deze methode wordt nader aangegeven in paragraaf 6.5.

Een derde methode wordt genoemd open adressering. Als een huisadres reeds bezet is wordt eenvoudig het volgende adres genomen of als dat bezet is het volgende, enz. tot een vrije plaats is gevonden. Bereikt men de laatste

plaats in het geheugen dan gaat men weer aan het begin verder. In dit geval is er dus geen apart overloopgebied. Deze methode wordt nader aangegeven in paragraaf 6.6.

Bij al deze methoden past men bij terugzoeken van een record hetzelfde algorithme toe als bij het plaatsen. Komt men op een vrije plaats of aan het einde van de ketting, dan is het record niet aanwezig. Wanneer geen overloop zou optreden heeft men in principe maar één stap nodig om een willekeurig record terug te vinden. Door de overloop is gemiddeld meer dan een stap nodig zowel bij succesvol als niet-succesvol zoeken.

6.2.4. Bucket en vullingsgraad

De toename in zoektijd door het optreden van overloop wil men zoveel mogelijk beperken. Het is duidelijk dat door toename van de overloop de zoektijd toeneemt, al is het verband niet lineair. Met een aantal maatregelen kan men de toename in zoektijd beperken. Ten eerste moet men natuurlijk een zo goed mogelijke hashfunctie kiezen en daarmee bij voorbaat het aantal synoniemen en de overloop beperken.

Een tweede maatregel is voor het primaire gebied meer geheugenruimte te reserveren dan nodig voor het aantal records. Als het aantal records n is en het primaire gebied ruimte bevat voor M records dan noemen we

$$\alpha = \frac{n}{M}$$

de vullingsgraad waarbij dan $\alpha \leq 1$. Gebruikelijke waarden zijn 0,7 à 0,8. Door een vullingsgraad kleiner dan 1 zal de kans op synoniemen afnemen en daarmee de overloop.

Een derde maatregel is het gebruik van "buckets" of "bins". Tot nu toe hebben we aangenomen dat per huisadres ruimte is voor één record plus eventueel een kettingadres. We kunnen de totale beschikbare primaire geheugenruimte echter ook zo verdelen dat per huisadres ruimte is voor meer dan één record. Zo'n ruimte wordt bucket of bin genoemd. Het aantal mogelijke records per bucket zullen we aangeven met b. Het aantal huisadressen blijven we aangeven

met M zodat de totale primaire ruimte is M.b en daarmee de vullingsgraad

$$\alpha = \frac{n}{Mb}$$

De situatie waarbij per huisadres maar ruimte is voor één record is dus alleen het bijzondere geval dat de bucketgrootte b = 1.
Grotere waarden van b hebben een afvlakkend effect op de stochastische afwijkingen t.o.v. de verwachting bij de afbeelding van sleutel- naar adresgebied.
Vaak wordt de waarde van b mede bepaald door de transporteereenheid van en naar achtergrondgeheugen. Dus bij voorkeur een spoor op een schijvengeheugen.

6.2.5. Verdeling over geheugen als Poisson proces

In voorgaande paragrafen hebben we gezien hoe de hashfunctie bepalend is voor de afbeelding van sleutelgebied naar adresgebied. Het ideaal daarbij is dat ongeacht de verdeling van sleutelwaarden over het sleutelgebied de verdeling over het adresgebied volledig homogeen is. Doordat altijd afwijkingen optreden ten opzichte van het theoretische ideaal ontstaan verschillende vormen van directe bestandsorganisaties. Deze organisaties verschillen vooral t.a.v. de behandeling van overloop.

In de verdere beschouwingen willen we een splitsing brengen tussen enerzijds het niet "ideaal" zijn van de hoofdfunctie en anderzijds de aannamen over bezetting van geheugenruimte.

Wat dat betreft gaan we nu uit van de volgende premissen. Wanneer een nieuw record wordt aangeboden dan wordt aangenomen dat na toepassing van de hashfunctie op de sleutelwaarde iedere geheugenpositie van 1 t/m M een even grote trefkans heeft.

Verder nemen we aan dat voor ieder huisadres een buckgrootte b geldt. Met n records is dan de vullingsgraad

$$\alpha = \frac{n}{M.b}$$

Door de statistische verdeling van de records over de huisadressen zal niet iedere bucket exact evenveel records bevatten. We noemen de kans op k records per huisadres P_k. Aangezien n en M in de praktijk grote getallen zijn geldt met zeer goede benadering voor P_k de Poisson verdeling, zodat

$$P_k = \frac{(\alpha b)^k e^{-\alpha b}}{k!}$$

Als we aannemen dat bij verwijdering van een record de kans op verwijdering van ieder record even groot is dan blijft ook dan de Poisson verdeling geldig.

6.3. Hashing

6.3.1. Hashing algemeen

Bij het ontwerpen van een hashfunctie is het belangrijkste dat de kans op synoniemen zo klein mogelijk wordt. Daarnaast spelen ook andere factoren een rol bij de keuze. Natuurlijk is de verdeling van te verwachten sleutels zeer belangrijk en er bestaat ook geen algemeen geldende hashfunctie. Wel zijn enkele methoden tamelijk favoriet en worden het meeste gebruikt. Hashfuncties worden algemeen opgezet als een eenvoudig numeriek algorithme.

Is de sleutel reeds numeriek dan gaat het numerieke proces rechtstreeks op de sleutel. Is de sleutel alphanumeriek dan wordt deze eerst numeriek gemaakt, meestal door uit te gaan van de interne binaire karaktervoorstelling. Daarbij kan men niet altijd zonder meer de binaire voorstelling hanteren. Wordt b.v. ieder karakter voorgesteld door 8 bits in ASCII code en bestaat de sleutel uit letters van 1 type, dan zijn 3 bits in ieder karakter van de sleutel gelijk en spelen dus in feite geen rol. Op welke manier dan ook, iedere sleutel wordt eventueel na aanpassing als numeriek opgevat.
In sommige gevallen hebben sleutels een systematische samenstelling. Bijvoorbeeld artikelnummers van een groot bedrijf. In de sleutel kan dan bijvoorbeeld een branchenummer, materiaalnummer, e.d. verwerkt zijn zodat ze voor insiders herkenbaar zijn. In veel gevallen zijn enkele cijferplaatsen

voorbehouden aan niet systematische volgordenummering. Door deze cijfers uit het geheel te lichten en samen te vormen tot een nieuw getal ontstaat reeds een eerste vervorming.
Ook getallen die te groot zijn kunnen door "opvouwen" gereduceerd worden. Bijvoorbeeld het getal 37265438941 is te reduceren tot een getal van 4 cijfers als volgt:

```
    8941
    6543
     372
1   5856
  ↳    1
    5857
```

Veel hashmethoden hebben tot effect het creëren van schijnbaar "willekeurige" cijfercombinaties. Sommige methoden zijn daarmee gelijk aan procedures voor randomgeneratoren.
Bijvoorbeeld de methode van "mid-squaring". Daarmee wordt de sleutel gekwadrateerd en van het resultaat worden de middelste cijfers genomen.
Bijvoorbeeld hetzelfde voorbeeld

$$37265438941^2 = 1388712939465399201481$$

De "middelste" 4 cijfers vormen dan 9465. Met deze methode moet men oppassen bij sleutels die op veel nullen kunnen eindigen of beginnen. Zo zijn nog wel enkele methoden aan te geven maar in verreweg de meeste gevallen valt men terug op een van de volgende twee methoden, namelijk de priemgetaldeling en de vermenigvuldiging.

6.3.2. Priemgetaldeling

Bij de priemgetaldeling kiest men de hashfunctie

$$A = S \bmod P$$

Hierin is P een priemgetal, ongeveer gelijk aan het adresbereik M. Het adres is dan de rest bij deling van sleutel S door priemgetal P. Dat voor P een priemgetal wordt gekozen heeft te maken met mogelijke regelmaat die in sleutels kan voorkomen. Bijvoorbeeld als P een even getal zou zijn dan leveren alle even sleutels ook even adressen. Natuurlijk zijn alle sleutels die een veelvoud van P plus een constante zijn synoniemen omdat dan A = constante. Men moet altijd het sleutelgedrag onderzoeken in relatie tot de gekozen methode. Behalve dat P priem moet zijn en natuurlijk ongeveer gelijk moet zijn aan het gewenste adresbereik M moet men ook oppassen dat niet afzonderlijke cijfers of deelgetallen van de sleutel een rol blijven spelen.

6.3.3. Hashing door vermenigvuldiging

Neem aan dat we rekenen met een woordlengte W. We gaan er vanuit dat de sleutel past binnen deze woordlengte. Is dat niet het geval dan kunnen we eerst door b.v. opvouwen die lengte daartoe reduceren. Stel als voorbeeld $W=10^6$ en de sleutel weer 37265438941.

Dit reduceren we

$$\begin{array}{r} 438941 \\ 37265 \\ \hline 476206 \end{array}$$

en beschouwen 476206 als de sleutel. Een sleutel S wordt vermenigvuldigd met een constante K. Dit geeft een dubbele lengteproduct, waarvan we de kop weglaten. We bepalen dus een grootheid, zo dat

$$Q = (K*S) \bmod W$$

Het adres A wordt dan bepaald door

$$A = \left\lfloor M*\frac{Q}{W} \right\rfloor$$

Er zijn twee voordelen aan deze methode. Tot aan de laatste vermenigvuldiging blijven alle karakteristieken van de sleutel S nog aanwezig, mits de constante K zo gekozen wordt dat K en W onderling ondeelbaar zijn dus ggd (K,W) = 1.

Een tweede voordeel bij vermenigvuldigen is dat opeenvolgende sleutelwaarden sterk gespreid worden over het adresgebied terwijl bij de methode van priemgetaldeling opeenvolgende sleutels ook opeenvolgende adressen geven. Aansluitende sleutelwaarden komen in de praktijk nogal eens voor, doordat het laatste karakter vaak een soort volgnummer is, bijvoorbeeld

STOEL-A; STOEL-B; STOEL-C
of GROEP1; GROEP2; GROEP3; GROEP4

6.4. Overloop door herhaalde hashing

Met opeenvolgende hashfuncties $h_1(s)$; $h_2(s)$; ... worden achtereenvolgens adressen A1; A2; bepaald. Bij het plaatsen wordt het record geplaatst op het eerste adres uit deze reeks dat vrij is. Bij terugzoeken wordt ook deze procedure gevolgd waardoor men dezelfde adressen in dezelfde volgorde afloopt. Een mogelijkheid voor hashfunctie h_i is bijvoorbeeld als volgt:

$$Q_i = (aQ_{i-1}+b) \bmod W \qquad (1)$$

$$A_i = \left\lfloor M * \frac{Q_i}{W} \right\rfloor$$

waarbij $Q_0 = S$

Betrekking (1) is een veelgebruikte methode voor randomgeneratoren. Bij geschikt gekozen waarden voor a en b doorlopen de Q_i's alle waarden tussen 0 en W in een zodanige volgorde dat ze als "random"getallen in het bereik 0-W kunnen worden opgevat (zie bijvoorbeeld Knuth, Art of Computer programming, Chapter 3, Random Numbers). Voor gegeven S ligt de rij Q_i en daarmee de rij adressen A_i dus vast. Echter voor Q_i geldt wel dat zowel voorganger als opvolger eenduidig zijn bepaald maar voor de reeks A_i geldt dit niet. Iedere sleutel S heeft dus zijn eigen reeks adressen. Zou men de herhaalde hashfunctie zo kiezen dat er sprake is van altijd dezelfde reeks adressen en de sleutel alleen het beginpunt in de reeks bepaalt dan kan men eenvoudiger opeenvolgende adressen kiezen. Dat laatste is echter de methode van open adressering zoals te bespreken in paragraaf 6.6.

Doordat de reeks adressen verschillend is voor verschillende sleutels is deze methode minder geschikt voor bestanden waaruit records moeten worden weggelaten. Bijvoorbeeld, neem aan dat record met sleutel S_1 niet kan worden geplaatst op adressen a en b maar wel op c, dan behoort bij S_1 de reeks:

$S_1 \rightarrow$ a, b, c

Neem aan op gelijke manier

S2 $\rightarrow$ p, q, c, r, s en

$S_3 \rightarrow$ u, v, w, c, z

Wanneer we nu het record met sleutel S_1 zouden verwijderen dan komt adres c vrij. Daardoor worden de ketens voor S_2 en S_3 verbroken en die zijn niet meer terug te vinden. En aan de hand van alleen adres c zijn ook de adressen r en z niet te bepalen. Komen weglatingen niet te veel voor en komt ook niet-succesvol zoeken nauwelijks voor dan bestaat nog de volgende mogelijkheid. Bij vervallen van sleutel S_1 wordt adres c niet volledig vrijgegeven maar voorzien van een merkteken dat het in gebruik is geweest. Men kan het voor een nieuw record dan wel weer gebruiken. Verder bij het terugzoeken van sleutels zoals S_2 en S_3 kan men over dit adres heengaan naar de opvolgers r, s en z. Het gevaar is dat na verloop van tijd praktisch geen adres meer echt vrij is en alleen een echt ongebruikt adres is een eindcriterium voor niet-succesvol zoeken.

In plaats van C_n en C_n' zullen we nu hanteren de overeenkomstige grootheden C_α en C_α'. Hiervoor geldt bij deze methode

$$C_\alpha = \frac{1}{\alpha} \ln \frac{1}{1-\alpha} \quad \text{en} \quad C'_\alpha = \frac{1}{1-\alpha}$$

Enkele waarden

α	C_α	C'_α
0,5	1,39	2
0,6	1,53	2,5
0,7	1,72	3,33
0,8	2,01	5
0,9	2,56	10
1,0	∞	∞

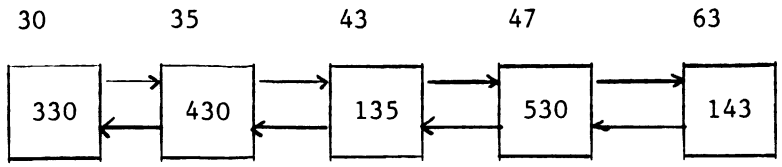
Bij niet te grote α is C_α dus nog vrij redelijk, echter C'_α is aan de hoge kant. Deze methode is daarom beperkt tot gevallen met geen weglatingen en waar succesvol zoeken overwegend is.

6.5. Overloop met kettingadressen

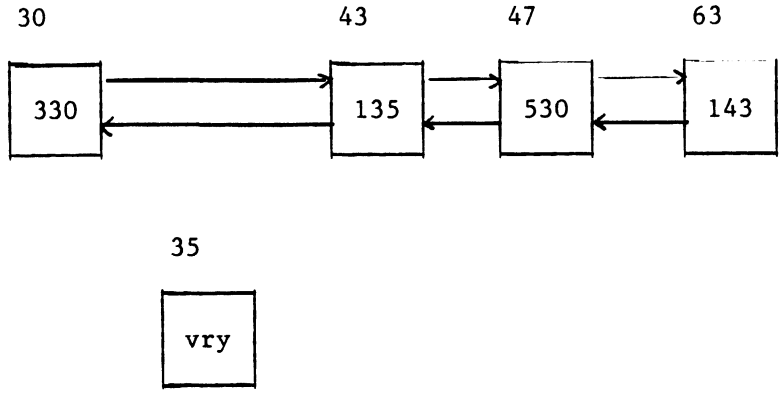
6.5.1. Algemeen

In feite maken we bij deze methode voor de overloop gebruik van de lineaire lijststruktuur zoals aangegeven in hoofdstuk 2. Nu zijn nog een aantal uitvoeringen mogelijk. Ten eerste kan men de overloop onderbrengen op nog vrije plaatsen in het primaire gebied of de overloop wordt geplaatst in een gescheiden overloopgebied.

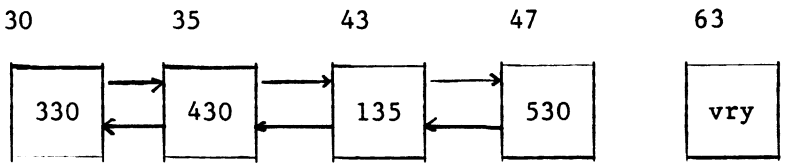
Brengt men de overloop in het primaire gebied dan zijn nog twee uitvoeringen mogelijk. Als een nieuw record moet worden ingebracht dan kan het voorkomen dat zijn huisadres reeds bezet is. Als dat laatste record daar niet thuis hoort en dus in feite een overloop is van een ander adres zijn er twee mogelijkheden. Men verplaatst het verkeerd staande record, past de ketenadressering aan en plaatst het nieuwe record op zijn eigen huisadres. De kettingen die op een huisadres starten blijven nu gescheiden. Een andere mogelijkheid is het nieuwe record eenvoudig op een vrije plaats te zetten en aan te hangen aan de keten die over zijn huisadres loopt. Het woord "over" is hier gebruikt omdat de keten die over zijn huisadres loopt niet op dit adres hoeft te eindigen. In dit systeem zullen kettingen die betrekking hebben op verschillende huisadressen samenvallen en één ketting vormen. Bij dit systeem ontstaan complicaties bij het verwijderen van records, enigszins vergelijkbaar met die bij herhaald hashing. Echter in dit geval is er een ketenstruktuur die het mogelijk maakt weer de nodige correcties aan te brengen door verder in de keten geplaatste records terug te plaatsen en als het nodig is de ketting ook weer te splitsen in deelkettingen. Een eenvoudig voorbeeld van deze laatste situatie is als volgt. Als hashfunctie nemen we voor het gemak even de laatste 2 cijfers van de sleutel. Er zou de volgende keten kunnen zijn opgebouwd.



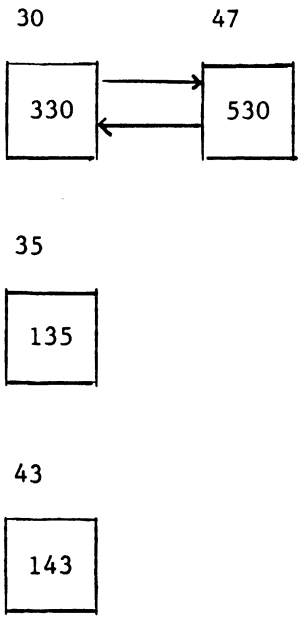
Nu willen we 430 verwijderen. Zou men adres 35 eenvoudig kortsluiten dan ontstaat



Echter dan is 135 niet meer terug te vinden. Daarom corrigeren we als volgt:



Ieder record dat terug kan naar zijn huisadres wordt teruggeschoven. Zouden we echter de records met sleutels 330, 135, 143 en 530 in die volgorde hebben aangeboden dan waren 3 kettingen als volgt ontstaan.



Men moet daarom het weglaatalgorithme zo invoeren dat niet alleen records naar hun eigen huisadres worden teruggeschoven, maar dat de ketting ook weer gesplitst wordt als dit kan. Doet men dit niet dan neemt de gemiddelde kettinglengte steeds maar toe tot uiteindelijk bijna alle records in één ketting zitten.
Wegens de ingewikkeldheid van het algorithme past men deze methode weinig toe hoewel de zoektijden bij deze methode vergelijkbaar zijn met die van de gescheiden kettingen.

Houden we de kettingen wel gescheiden dan moeten we dus wel extra moeite doen om een nieuw record te plaatsen als het huisadres reeds bezet is door een record dat er niet thuis hoort. Meestal vermijdt men ook die complicatie en gebruikt eenvoudig een afzonderlijk overloopgebied. Wel is het wenselijk bij het ontwerp een schatting te hebben van de benodigde overloopruimte.

6.5.2. Verwachtingen bij kettingadressering

Ook hier definiëren we weer C_α als de verwachting van het aantal stappen bij succesvol zoeken en onder aanname dat $n,M \to \infty$ zodat de Poisson verdeling geldt. Op overeenkomstige manier C_α' bij niet succesvol zoeken.
Verder hanteren we de grootheden ε_b en μ_b. Hierbij is ε_b de relatieve overloop, d.w.z. de verwachting van het aantal over te lopen records gedeeld

door het totale aantal records.
Met μ_b wordt aangegeven de overschrijdingskans, d.w.z. de kans dat een bucketgrootte b niet toereikend is.
Voor het geval dat b = 1 is dan af te leiden dat

$$C_\alpha = 1 + \frac{1}{2}\alpha \quad C'_\alpha = e^{-\alpha} + \alpha$$

$$\varepsilon_1 = 1 - \frac{1}{\alpha} + \frac{1}{\alpha} e^{-\alpha} \quad \mu_1 = 1 - e^{-\alpha}(1+\alpha)$$

Voor waarden van b met b > 1 zijn geen gesloten formules af te leiden. Wel zijn via reeksontwikkeling numeriek deze grootheden te bepalen. Voor een aantal waarden van b en α zijn deze grootheden in volgende tabellen aangegeven.
Bij de berekening van deze waarden is er van uitgegaan dat in het verloopgebied geen buckets meer worden toegepast. Dit betekent dat in het overloopgebeid de records op enkelvoudige adressen via kettingen aan elkaar hangen.

Dit heeft onder andere tot gevolg dat voor grotere α waarden bij toenemende b er aanvankelijk een stijging is voor C_α'.

De resultaten van C_α en C'_α zijn ook nog eens in grafische vorm weergegeven. Men ziet daar ook duidelijk het laatst genoemde effect. Voor C_α speelt dit effect pas als α>1.

Overloop d.m.v. kettingen.
Aantal stappen bij succesvol zoeken.

et	vullingsgraad									
tte	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
1	1.0500	1.1000	1.1500	1.2000	1.2500	1.3000	1.3500	1.4000	1.4500	1.5000
2	1.0063	1.0242	1.0520	1.0883	1.1321	1.1823	1.2381	1.2988	1.3637	1.4323
3	1.0010	1.0071	1.0215	1.0458	1.0806	1.1259	1.1812	1.2459	1.3192	1.4004
4	1.0002	1.0023	1.0097	1.0257	1.0527	1.0922	1.1450	1.2111	1.2900	1.3809
5	1.0000	1.0008	1.0046	1.0151	1.0358	1.0699	1.1195	1.1856	1.2684	1.3675
6	1.0000	1.0003	1.0023	1.0091	1.0251	1.0543	1.1004	1.1657	1.2514	1.3575
7	1.0000	1.0001	1.0012	1.0057	1.0179	1.0429	1.0856	1.1497	1.2374	1.3496
8	1.0000	1.0000	1.0006	1.0036	1.0130	1.0343	1.0737	1.1363	1.2256	1.3433
9	1.0000	1.0000	1.0003	1.0023	1.0095	1.0278	1.0640	1.1249	1.2153	1.3381
0	1.0000	1.0000	1.0002	1.0015	1.0070	1.0226	1.0560	1.1151	1.2063	1.3336
5	1.0000	1.0000	1.0000	1.0002	1.0017	1.0089	1.0307	1.0805	1.1726	1.3184
0	1.0000	1.0000	1.0000	1.0000	1.0005	1.0038	1.0181	1.0594	1.1496	1.3093
5	1.0000	1.0000	1.0000	1.0000	1.0001	1.0017	1.0111	1.0453	1.1323	1.3031
0	1.0000	1.0000	1.0000	1.0000	1.0000	1.0008	1.0071	1.0353	1.1186	1.2985
5	1.0000	1.0000	1.0000	1.0000	1.0000	1.0004	1.0046	1.0279	1.1073	1.2949
0	1.0000	1.0000	1.0000	1.0000	1.0000	1.0002	1.0030	1.0223	1.0978	1.2920
5	1.0000	1.0000	1.0000	1.0000	1.0000	1.0001	1.0020	1.0180	1.0896	1.2896
0	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0013	1.0147	1.0825	1.2876
5	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0009	1.0120	1.0762	1.2858
0	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0006	1.0099	1.0707	1.2843
5	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0004	1.0082	1.0657	1.2830
0	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0003	1.0068	1.0612	1.2818
5	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0002	1.0057	1.0572	1.2807
0	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0001	1.0048	1.0535	1.2797
5	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0001	1.0040	1.0501	1.2788
0	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0001	1.0034	1.0471	1.2780
5	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0001	1.0028	1.0442	1.2773
0	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0024	1.0416	1.2766
0	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0017	1.0370	1.2753
0	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0013	1.0330	1.2743
0	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0009	1.0295	1.2733
0	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0007	1.0265	1.2725
0	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0005	1.0239	1.2717
0	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0004	1.0215	1.2710
0	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0003	1.0195	1.2704
0	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0002	1.0177	1.2698
0	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0001	1.0160	1.2693
0	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0001	1.0146	1.2688
0	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0092	1.2668
0	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0060	1.2654
0	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0040	1.2642
0	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0027	1.2633
0	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0018	1.2625
0	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0012	1.2619

Overloop d.m.v. kettingen.
Aantal stappen bij niet-succesvol zoeken.

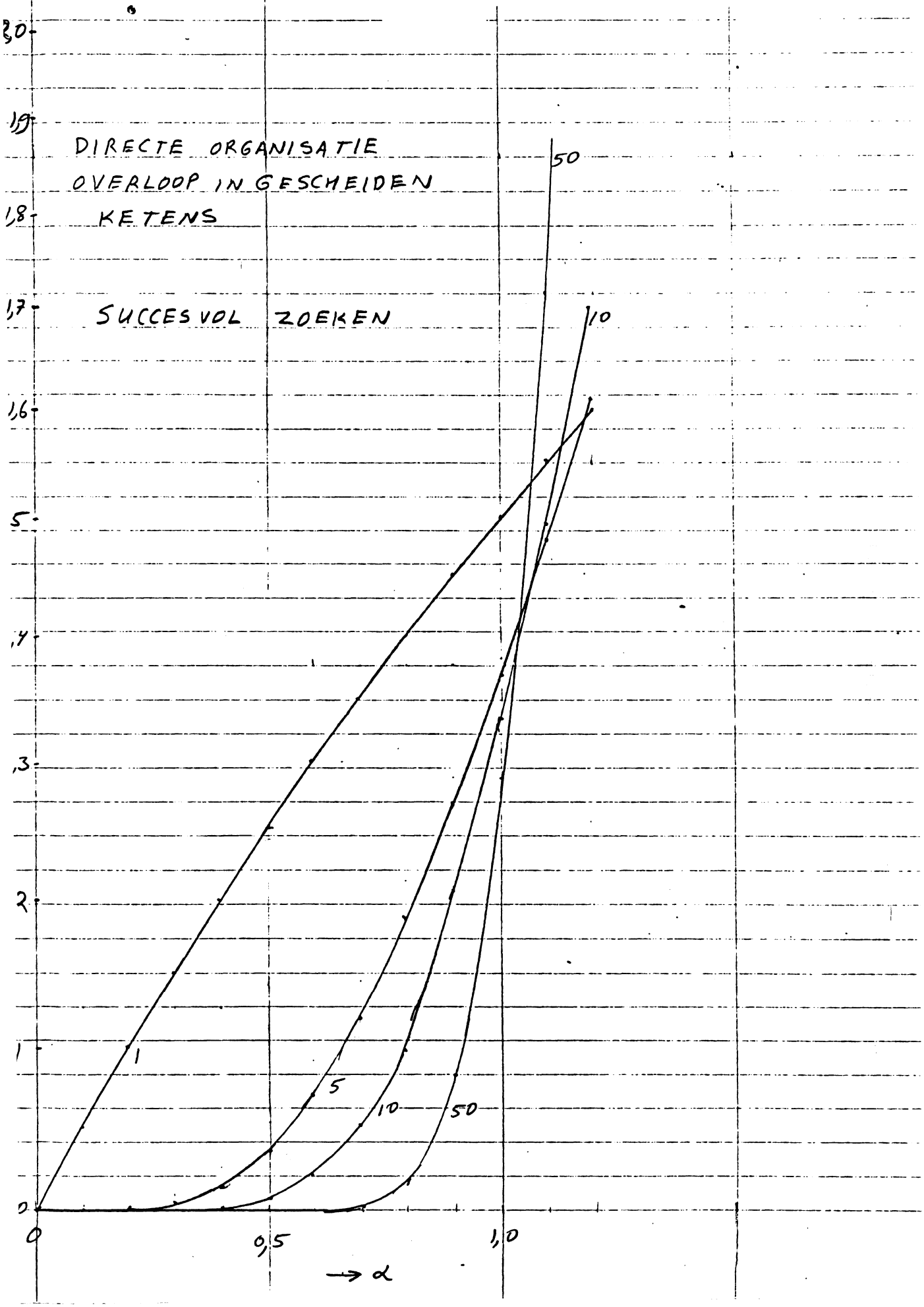
bucket	vullingsgraad									
grootte	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
1	1.0048	1.0187	1.0408	1.0703	1.1065	1.1488	1.1966	1.2493	1.3066	1.3679
2	1.0012	1.0088	1.0269	1.0581	1.1036	1.1638	1.2384	1.3268	1.4281	1.5413
3	1.0003	1.0038	1.0162	1.0433	1.0898	1.1588	1.2517	1.3689	1.5095	1.6721
4	1.0001	1.0016	1.0095	1.0314	1.0751	1.1476	1.2533	1.3944	1.5710	1.7815
5	1.0000	1.0007	1.0056	1.0225	1.0619	1.1346	1.2490	1.4103	1.6202	1.8773
6	1.0000	1.0003	1.0033	1.0160	1.0507	1.1215	1.2416	1.4200	1.6609	1.9637
7	1.0000	1.0001	1.0019	1.0114	1.0413	1.1090	1.2324	1.4254	1.6954	2.0430
8	1.0000	1.0001	1.0011	1.0081	1.0336	1.0975	1.2224	1.4277	1.7250	2.1167
9	1.0000	1.0000	1.0007	1.0058	1.0273	1.0869	1.2119	1.4276	1.7507	2.1858
10	1.0000	1.0000	1.0004	1.0041	1.0222	1.0773	1.2013	1.4259	1.7732	2.2511
15	1.0000	1.0000	1.0000	1.0008	1.0078	1.0427	1.1522	1.4019	1.8527	2.5365
20	1.0000	1.0000	1.0000	1.0001	1.0028	1.0234	1.1129	1.3674	1.8979	2.7767
25	1.0000	1.0000	1.0000	1.0000	1.0010	1.0128	1.0832	1.3308	1.9232	2.9881
30	1.0000	1.0000	1.0000	1.0000	1.0004	1.0071	1.0612	1.2955	1.9357	3.1790
35	1.0000	1.0000	1.0000	1.0000	1.0001	1.0039	1.0450	1.2627	1.9395	3.3546
40	1.0000	1.0000	1.0000	1.0000	1.0000	1.0022	1.0331	1.2329	1.9372	3.5179
45	1.0000	1.0000	1.0000	1.0000	1.0000	1.0012	1.0243	1.2060	1.9303	3.6712
50	1.0000	1.0000	1.0000	1.0000	1.0000	1.0007	1.0179	1.1820	1.9201	3.8163
55	1.0000	1.0000	1.0000	1.0000	1.0000	1.0004	1.0132	1.1607	1.9074	3.9542
60	1.0000	1.0000	1.0000	1.0000	1.0000	1.0002	1.0097	1.1418	1.8929	4.0859
65	1.0000	1.0000	1.0000	1.0000	1.0000	1.0001	1.0072	1.1250	1.8770	4.2123
70	1.0000	1.0000	1.0000	1.0000	1.0000	1.0001	1.0053	1.1102	1.8601	4.3338
75	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0039	1.0972	1.8425	4.4511
80	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0029	1.0857	1.8244	4.5645
85	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0021	1.0755	1.8059	4.6745
90	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0016	1.0666	1.7873	4.7812
95	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0012	1.0587	1.7686	4.8850
100	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0009	1.0518	1.7499	4.9861
110	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0005	1.0403	1.7128	5.1810
120	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0003	1.0313	1.6765	5.3672
130	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0001	1.0244	1.6414	5.5457
140	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0001	1.0190	1.6074	5.7175
150	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0148	1.5748	5.8833
160	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0116	1.5437	6.0436
170	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0090	1.5139	6.1990
180	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0070	1.4856	6.3499
190	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0055	1.4587	6.4966
200	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0043	1.4332	6.6395
250	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0013	1.3244	7.3057
300	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0004	1.2424	7.9080
350	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0001	1.1810	8.4617
400	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.1352	8.9772
450	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.1010	9.4613
500	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0755	9.9191

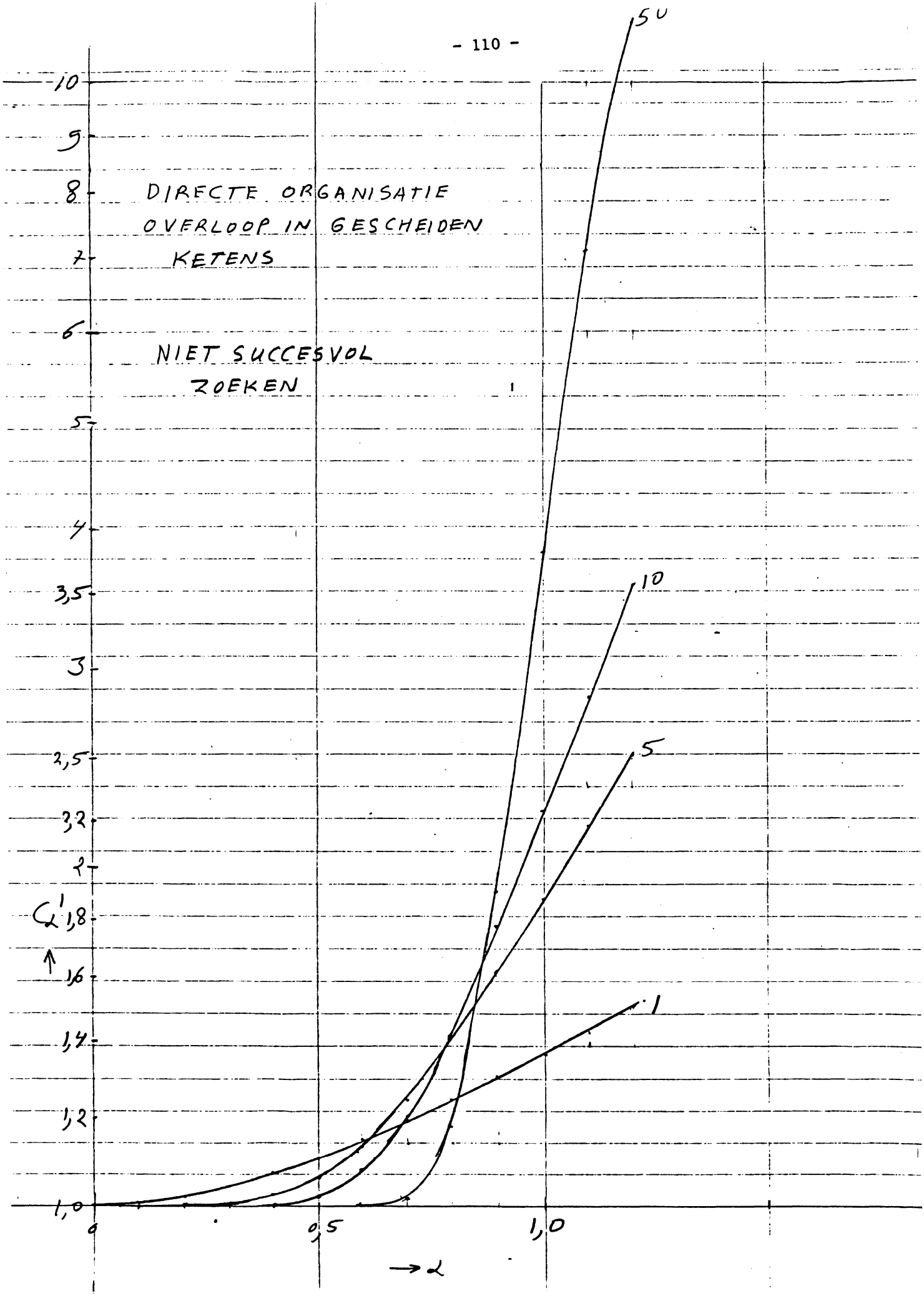
Overloop d.m.v. kettingen.
relatieve overloop.

et	vullingsgraad									
tte	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
1	48E-03	94E-03	14E-02	18E-02	21E-02	25E-02	28E-02	31E-02	34E-02	37E-02
2	60E-04	22E-03	45E-03	73E-03	10E-02	14E-02	17E-02	20E-02	24E-02	27E-02
3	94E-05	63E-04	18E-03	36E-03	60E-03	88E-03	12E-02	15E-02	19E-02	22E-02
4	16E-05	20E-04	79E-04	20E-03	38E-03	61E-03	90E-03	12E-02	16E-02	20E-02
5	30E-06	69E-05	37E-04	11E-03	25E-03	45E-03	71E-03	10E-02	14E-02	18E-02
6	59E-07	24E-05	18E-04	67E-04	17E-03	34E-03	58E-03	87E-03	12E-02	16E-02
7	12E-07	90E-06	91E-05	41E-04	12E-03	26E-03	47E-03	76E-03	11E-02	15E-02
8	24E-08	34E-06	47E-05	25E-04	84E-04	20E-03	40E-03	67E-03	10E-02	14E-02
9	51E-09	13E-06	24E-05	16E-04	61E-04	16E-03	34E-03	59E-03	93E-03	13E-02
0	11E-09	50E-07	13E-05	10E-04	44E-04	13E-03	29E-03	53E-03	86E-03	13E-02
5	56E-13	50E-09	61E-07	13E-05	10E-04	47E-04	14E-03	33E-03	63E-03	10E-02
0	34E-16	59E-11	33E-08	18E-06	28E-05	20E-04	81E-04	23E-03	50E-03	89E-03
5	22E-19	75E-13	19E-09	27E-07	79E-06	86E-05	48E-04	17E-03	41E-03	80E-03
0	15E-22	10E-14	12E-10	44E-08	24E-06	39E-05	29E-04	12E-03	35E-03	73E-03
5	11E-25	14E-16	79E-12	73E-09	74E-07	19E-05	18E-04	94E-04	30E-03	67E-03
0	81E-29	20E-18	52E-13	13E-09	23E-07	90E-06	12E-04	73E-04	26E-03	63E-03
5	61E-32	30E-20	36E-14	22E-10	76E-08	44E-06	77E-05	57E-04	23E-03	59E-03
0	47E-35	45E-22	25E-15	39E-11	25E-08	22E-06	51E-05	46E-04	20E-03	56E-03
5	37E-38	68E-24	17E-16	70E-12	84E-09	11E-06	34E-05	37E-04	18E-03	54E-03
0	0	10E-25	12E-17	13E-12	28E-09	58E-07	23E-05	30E-04	17E-03	51E-03
5	0	16E-27	88E-19	23E-13	96E-10	30E-07	16E-05	24E-04	15E-03	49E-03
0	0	25E-29	63E-20	43E-14	33E-10	15E-07	11E-05	20E-04	14E-03	48E-03
5	0	40E-31	46E-21	80E-15	11E-10	81E-08	74E-06	16E-04	12E-03	46E-03
0	0	64E-33	34E-22	15E-15	40E-11	42E-08	52E-06	13E-04	11E-03	45E-03
5	0	10E-34	25E-23	28E-16	14E-11	22E-08	36E-06	11E-04	11E-03	43E-03
0	0	16E-36	18E-24	53E-17	49E-12	12E-08	25E-06	92E-05	97E-04	42E-03
5	0	26E-38	14E-25	10E-17	17E-12	63E-09	18E-06	77E-05	90E-04	41E-03
0	0	43E-40	10E-26	19E-18	61E-13	34E-09	12E-06	65E-05	83E-04	40E-03
0	0	0	58E-29	71E-20	77E-14	98E-10	62E-07	46E-05	72E-04	38E-03
0	0	0	33E-31	27E-21	98E-15	29E-10	31E-07	33E-05	63E-04	36E-03
0	0	0	19E-33	10E-22	13E-15	85E-11	16E-07	23E-05	55E-04	35E-03
0	0	0	11E-35	38E-24	16E-16	25E-11	82E-08	17E-05	48E-04	34E-03
0	0	0	64E-38	15E-25	22E-17	75E-12	42E-08	12E-05	43E-04	33E-03
0	0	0	38E-40	56E-27	28E-18	23E-12	22E-08	90E-06	38E-04	32E-03
0	0	0	0	22E-28	38E-19	69E-13	11E-08	66E-06	34E-04	31E-03
0	0	0	0	84E-30	50E-20	21E-13	60E-09	49E-06	30E-04	30E-03
0	0	0	0	33E-31	68E-21	64E-14	32E-09	36E-06	27E-04	29E-03
0	0	0	0	13E-32	91E-22	20E-14	17E-09	27E-06	24E-04	28E-03
0	0	0	0	13E-39	42E-26	56E-17	72E-11	63E-07	14E-04	25E-03
0	0	0	0	0	20E-30	17E-19	32E-12	15E-07	90E-05	23E-03
0	0	0	0	0	10E-34	52E-22	15E-13	40E-08	57E-05	21E-03
0	0	0	0	0	55E-39	17E-24	74E-15	10E-08	38E-05	20E-03
0	0	0	0	0	0	56E-27	37E-16	28E-09	25E-05	19E-03
0	0	0	0	0	0	19E-29	19E-17	75E-10	17E-05	18E-03

Overloop d.m.v. kettingen.
overschrijdingskans.

bucket	vullingsgraad									
grootte	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
1	47E-04	18E-03	37E-03	62E-03	90E-03	12E-02	16E-02	19E-02	23E-02	26E-02
2	11E-04	79E-04	23E-03	47E-03	80E-03	12E-02	17E-02	22E-02	27E-02	32E-02
3	27E-05	34E-04	13E-03	34E-03	66E-03	11E-02	16E-02	22E-02	29E-02	35E-02
4	61E-06	14E-04	77E-04	24E-03	53E-03	96E-03	15E-02	22E-02	29E-02	37E-02
5	14E-06	59E-05	45E-04	17E-03	42E-03	84E-03	14E-02	21E-02	30E-02	38E-02
6	33E-07	25E-05	26E-04	12E-03	34E-03	73E-03	13E-02	21E-02	30E-02	39E-02
7	77E-08	11E-05	15E-04	81E-04	27E-03	64E-03	12E-02	20E-02	30E-02	40E-02
8	18E-08	45E-06	86E-05	57E-04	21E-03	56E-03	11E-02	20E-02	30E-02	41E-02
9	43E-09	19E-06	50E-05	40E-04	17E-03	49E-03	11E-02	19E-02	30E-02	41E-02
10	10E-09	83E-07	29E-05	28E-04	14E-03	43E-03	99E-03	18E-02	29E-02	42E-02
15	77E-13	12E-08	20E-06	51E-05	46E-04	22E-03	68E-03	16E-02	28E-02	43E-02
20	61E-16	19E-10	15E-07	94E-06	16E-04	12E-03	48E-03	13E-02	27E-02	44E-02
25	50E-19	30E-12	11E-08	18E-06	56E-05	62E-04	34E-03	11E-02	26E-02	45E-02
30	41E-22	49E-14	79E-10	34E-07	20E-05	33E-04	24E-03	96E-03	24E-02	45E-02
35	35E-25	80E-16	60E-11	65E-08	71E-06	18E-04	17E-03	82E-03	23E-02	46E-02
40	29E-28	13E-17	45E-12	13E-08	25E-06	99E-05	12E-03	71E-03	22E-02	46E-02
45	25E-31	22E-19	34E-13	25E-09	92E-07	54E-05	90E-04	61E-03	21E-02	46E-02
50	21E-34	36E-21	26E-14	48E-10	34E-07	30E-05	65E-04	53E-03	20E-02	46E-02
55	18E-37	60E-23	20E-15	95E-11	12E-07	16E-05	48E-04	46E-03	19E-02	46E-02
60	0	10E-24	16E-16	19E-11	45E-08	91E-06	35E-04	40E-03	19E-02	47E-02
65	0	17E-26	12E-17	37E-12	16E-08	51E-06	25E-04	34E-03	18E-02	47E-02
70	0	29E-28	95E-19	74E-13	61E-09	28E-06	19E-04	30E-03	17E-02	47E-02
75	0	49E-30	74E-20	15E-13	22E-09	16E-06	14E-04	26E-03	16E-02	47E-02
80	0	82E-32	58E-21	29E-14	83E-10	88E-07	10E-04	23E-03	16E-02	47E-02
85	0	14E-33	45E-22	59E-15	31E-10	49E-07	73E-05	20E-03	15E-02	47E-02
90	0	24E-35	35E-23	12E-15	11E-10	28E-07	54E-05	17E-03	15E-02	47E-02
95	0	40E-37	28E-24	24E-16	42E-11	15E-07	40E-05	15E-03	14E-02	47E-02
100	0	69E-39	22E-25	47E-17	16E-11	87E-08	29E-05	13E-03	13E-02	47E-02
110	0	0	13E-27	19E-18	22E-12	27E-08	16E-05	10E-03	12E-02	47E-02
120	0	0	83E-30	78E-20	30E-13	87E-09	87E-06	78E-04	12E-02	48E-02
130	0	0	52E-32	32E-21	42E-14	28E-09	48E-06	60E-04	11E-02	48E-02
140	0	0	32E-34	13E-22	59E-15	89E-10	26E-06	46E-04	10E-02	48E-02
150	0	0	20E-36	53E-24	83E-16	28E-10	14E-06	36E-04	93E-03	48E-02
160	0	0	13E-38	22E-25	12E-16	91E-11	80E-07	27E-04	86E-03	48E-02
170	0	0	0	89E-27	16E-17	29E-11	44E-07	21E-04	80E-03	48E-02
180	0	0	0	37E-28	23E-18	94E-12	24E-07	16E-04	75E-03	48E-02
190	0	0	0	15E-29	33E-19	30E-12	13E-07	13E-04	70E-03	48E-02
200	0	0	0	62E-31	46E-20	97E-13	75E-08	99E-05	65E-03	48E-02
250	0	0	0	76E-38	27E-24	34E-15	40E-09	28E-05	46E-03	48E-02
300	0	0	0	0	16E-28	12E-17	21E-10	83E-06	33E-03	48E-02
350	0	0	0	0	92E-33	45E-20	12E-11	24E-06	24E-03	49E-02
400	0	0	0	0	55E-37	16E-22	65E-13	72E-07	18E-03	49E-02
450	0	0	0	0	0	61E-25	36E-14	21E-07	13E-03	49E-02
500	0	0	0	0	0	23E-27	20E-15	64E-08	95E-04	49E-02


DIRECTE ORGANISATIE
OVERLOOP IN GESCHEIDEN
KETENS
SUCCESVOL ZOEKEN
50
10
1
5
10
50
0
0,5
1,0
→ α


DIRECTE ORGANISATIE
OVERLOOP IN GESCHEIDEN
KETENS
NIET SUCCESVOL
ZOEKEN
50
10
5
1
10
9
8
7
6
5
4
3,5
3
2,5
2
1,8
1,6
1,4
1,2
1,0
Q'
0
0,5
1,0
→ α

6.6. Overloop met open adressering

6.6.1. Inleiding

Overloop vindt nu plaats als het huisadres bezet is eenvoudig naar de eerstvolgende vrije plaats. Dit betekent als we buckets >1 toepassen het eerstvolgende bucket dat nog niet vol is. Terugzoeken van records gebeurt volgens hetzelfde principe. Niet-succesvol zoeken eindigt bij een bucket dat niet vol is. Er is geen overloopgebied en men zal daarom in de regel een iets kleinere α toepassen dan in het geval van kettingen. Verwijderen van records vereist enige zorgvuldigheid. Immers wanneer een record verwijderd wordt uit een bucket dat vol is dan kan het voorkomen dat een record uit een der volgende buckets teruggeplaatst moet worden. Bij redelijke bucketgrootte en niet te grote α is de overloop zo gering dadt dit niet te vaak hoeft te gebeuren. Door dit terugplaatsen bereiken we dezelfde situatie als na het aanvankelijk laden. Voor de situatie direct na het aanvankelijk laden is aan te tonen dat de volgorde van lading niet relevant is voor de waarde van C_n. Dit is aangetoond door W.W. Peterson in 1957 en wel als volgt.
Stel a en b zijn twee records waarbij b direct na a wordt geladen. Beschouw daarna de situatie dat a direct na b wordt geladen en de rest ongewijzigd. Indien de records a en b in beide gevallen op dezelfde plaatsen terecht komen zijn de eindsituaties identiek.
Komt in tweede situatie b op de plaats waar eerst a kwam, dan komt automatisch a nu op de plaats waar b kwam. De zoeklengte voor a neemt dan toe maar die voor b neemt met hetzelfde bedrag af. Het gemiddelde blijft dus gelijk. Aangezien we iedere ordening van records kunnen bereiken door steeds 2 opeenvolgende te wisselen is het gestelde bewezen.

Ook bij deze methode zijn onder aanname van Poisson proces en via reeksontwikkelingen numerieke waarden voor C_α en C'_α te bepalen. Hiervan zijn de resultaten in volgende tabellen gegeven.

Gebaseerd op de gebruikte formules is ook de kans op een overloop q_0 en de kans op overloop $(1-q_0)$ in tabellen weergegeven. Omdat voor grotere waarden van α en b de waarde q_0 ongeveer 1 wordt is overloop $(1-q_0)$ dan slechts met beperkte nauwkeurigheid vast te stellen. De hier gegeven tabellen zijn uitgerekend met ongeveer 11 decimale cijfers nauwkeurigheid. Het gevolg daarvan is dat waarden voor $(1-q_0)$ kleiner dan 10^{-8} niet meer konden worden bepaald en als 0 staan opgegeven.

Open adressering.

Aantal stappen bij niet-succesvol zoeken.

bucket	vullingsgraad									
	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	0.95
1	1.0556	1.1250	1.2143	1.3333	1.5000	1.7500	2.1667	3.0000	5.5000	10.500
2	1.0062	1.0242	1.0553	1.1033	1.1767	1.2930	1.4936	1.9033	3.1469	5.643
3	1.0009	1.0066	1.0201	1.0450	1.0872	1.1584	1.2862	1.5539	2.3779	4.040
4	1.0002	1.0021	1.0085	1.0227	1.0497	1.0984	1.1897	1.3861	2.0000	3.244
5	1.0000	1.0007	1.0039	1.0124	1.0307	1.0661	1.1355	1.2891	1.7767	2.770
6	1.0000	1.0002	1.0019	1.0072	1.0199	1.0467	1.1016	1.2266	1.6300	2.456
7	1.0000	1.0001	1.0009	1.0043	1.0135	1.0342	1.0789	1.1834	1.5266	2.234
8	1.0000	1.0000	1.0005	1.0026	1.0093	1.0257	1.0627	1.1520	1.4500	2.067
9	1.0000	1.0000	1.0002	1.0017	1.0066	1.0197	1.0508	1.1282	1.3911	1.939
10	1.0000	1.0000	1.0001	1.0011	1.0047	1.0154	1.0419	1.1098	1.3446	1.837
15	1.0000	1.0000	1.0000	1.0001	1.0011	1.0052	1.0185	1.0584	1.2089	1.534
20	1.0000	1.0000	1.0000	1.0000	1.0003	1.0020	1.0095	1.0359	1.1443	1.385
25	1.0000	1.0000	1.0000	1.0000	1.0001	1.0009	1.0054	1.0239	1.1072	1.298
30	1.0000	1.0000	1.0000	1.0000	1.0000	1.0004	1.0032	1.0167	1.0834	1.241
35	1.0000	1.0000	1.0000	1.0000	1.0000	1.0002	1.0020	1.0122	1.0671	1.201
40	1.0000	1.0000	1.0000	1.0000	1.0000	1.0001	1.0012	1.0091	1.0553	1.171
45	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0008	1.0069	1.0464	1.148
50	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0005	1.0054	1.0395	1.130
55	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0003	1.0042	1.0341	1.115
60	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0002	1.0033	1.0297	1.103
65	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0002	1.0027	1.0261	1.093
70	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0001	1.0022	1.0231	1.085
75	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0001	1.0018	1.0205	1.078
80	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0001	1.0014	1.0184	1.071
85	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0012	1.0165	1.066
90	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0010	1.0149	1.061
95	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0008	1.0135	1.057
100	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0007	1.0123	1.053
110	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0005	1.0103	1.047
120	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0003	1.0087	1.041
130	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0002	1.0074	1.037
140	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0002	1.0064	1.033
150	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0001	1.0055	1.030
160	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0001	1.0048	1.027
170	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0001	1.0042	1.025
180	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0037	1.023
190	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0033	1.021
200	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0029	1.019
250	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0016	1.014
300	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0010	1.010
350	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0006	1.007
400	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0004	1.006
450	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0003	1.005
500	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0002	1.004
600	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0001	1.002
700	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.002
800	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.001
900	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.001
1000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.000

Open adressering.

Aantal stappen bij succesvol zoeken.

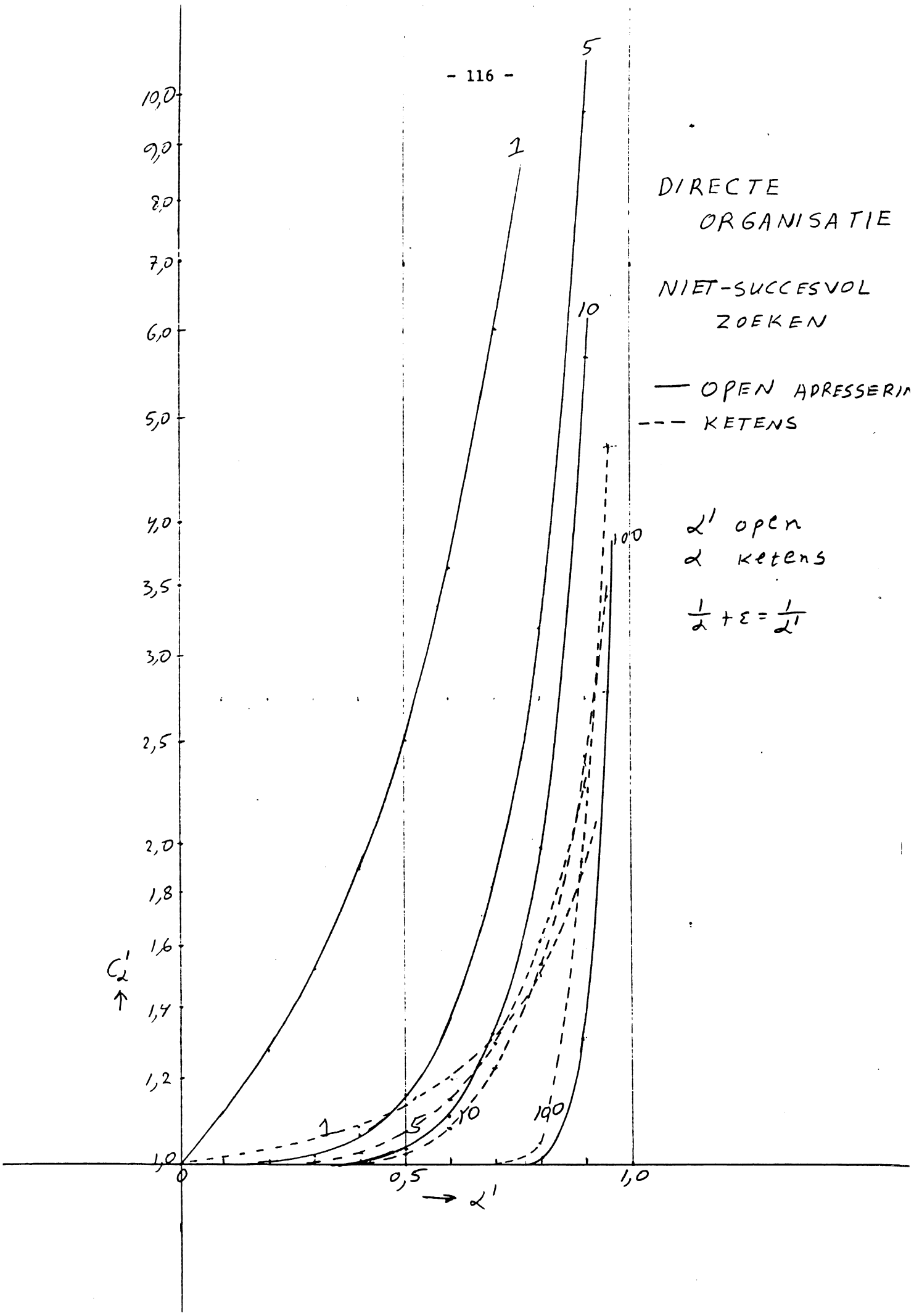
at	vullingsgraad									
	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	0.95
	1.1173	1.2812	1.5204	1.8889	2.5000	3.6250	6.0556	13.0000	50.5000	200.5000
	1.0183	1.0724	1.1711	1.3386	1.6305	2.1814	3.3866	6.8500	25.5922	100.5886
	1.0036	1.0247	1.0756	1.1740	1.3570	1.7140	2.5086	4.8096	17.2973	67.2916
	1.0008	1.0093	1.0375	1.1010	1.2284	1.4873	2.0753	3.7939	13.1533	50.6460
	1.0002	1.0037	1.0198	1.0625	1.1562	1.3556	1.8187	3.1871	10.6687	40.6602
	1.0000	1.0015	1.0108	1.0402	1.1114	1.2706	1.6501	2.7844	9.0136	34.0040
	1.0000	1.0006	1.0060	1.0266	1.0817	1.2120	1.5313	2.4980	7.8323	29.2503
	1.0000	1.0003	1.0034	1.0179	1.0612	1.1697	1.4435	2.2842	6.9469	25.6854
	1.0000	1.0001	1.0019	1.0122	1.0465	1.1380	1.3763	2.1187	6.2580	22.9131
	1.0000	1.0000	1.0011	1.0084	1.0358	1.1136	1.3234	1.9870	5.7087	20.6956
	1.0000	1.0000	1.0001	1.0014	1.0107	1.0485	1.1717	1.5969	4.0617	14.0452
	1.0000	1.0000	1.0000	1.0003	1.0035	1.0232	1.1028	1.4072	3.2414	10.7222
	1.0000	1.0000	1.0000	1.0000	1.0012	1.0117	1.0656	1.2968	2.7513	8.7296
	1.0000	1.0000	1.0000	1.0000	1.0004	1.0061	1.0435	1.2255	2.4259	7.4021
	1.0000	1.0000	1.0000	1.0000	1.0001	1.0032	1.0297	1.1763	2.1945	6.4545
	1.0000	1.0000	1.0000	1.0000	1.0001	1.0017	1.0206	1.1408	2.0218	5.7443
	1.0000	1.0000	1.0000	1.0000	1.0000	1.0010	1.0145	1.1142	1.8881	5.1923
	1.0000	1.0000	1.0000	1.0000	1.0000	1.0005	1.0103	1.0938	1.7817	4.7510
	1.0000	1.0000	1.0000	1.0000	1.0000	1.0003	1.0074	1.0778	1.6952	4.3903
	1.0000	1.0000	1.0000	1.0000	1.0000	1.0002	1.0053	1.0651	1.6234	4.0899
	1.0000	1.0000	1.0000	1.0000	1.0000	1.0001	1.0038	1.0548	1.5630	3.8358
	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0028	1.0464	1.5116	3.6183
	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0020	1.0394	1.4673	3.4299
	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0015	1.0337	1.4287	3.2652
	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0011	1.0289	1.3949	3.1200
	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0008	1.0248	1.3651	2.9910
	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0006	1.0214	1.3386	2.8757
	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0004	1.0185	1.3148	2.7721
	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0002	1.0139	1.2743	2.5933
	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0001	1.0105	1.2410	2.4445
	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0001	1.0080	1.2133	2.3189
	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0061	1.1899	2.2115
	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0047	1.1699	2.1186
	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0036	1.1527	2.0374
	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0028	1.1378	1.9660
	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0021	1.1248	1.9027
	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0017	1.1134	1.8461
	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0013	1.1032	1.7953
	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0004	1.0669	1.6037
	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0001	1.0451	1.4776
	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0312	1.3888
	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0219	1.3232
	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0157	1.2730
	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0113	1.2335
	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0060	1.1759
	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0033	1.1362
	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0018	1.1077
	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0010	1.0865
	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0006	1.0704

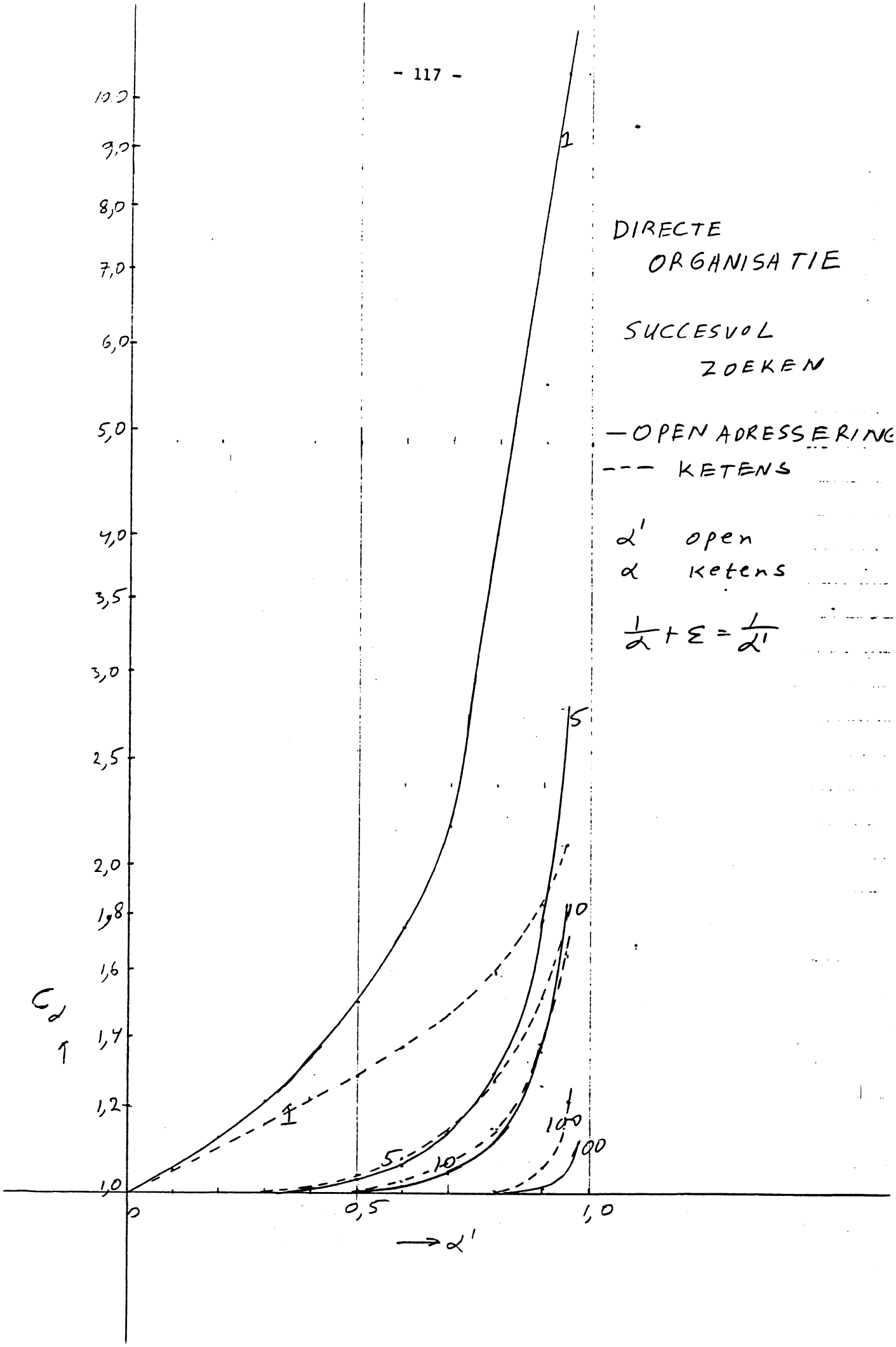
Open adressering.
kans op geen overloop.

bucket	vullingsgraad									
	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	0.95
1	0.9947	0.9771	0.9449	0.8951	0.8244	0.7288	0.6041	0.4451	0.2460	0.1293
2	0.9988	0.9913	0.9722	0.9366	0.8787	0.7925	0.6706	0.5047	0.2850	0.1514
3	0.9997	0.9965	0.9852	0.9594	0.9121	0.8349	0.7179	0.5494	0.3156	0.1692
4	0.9999	0.9986	0.9918	0.9732	0.9345	0.8656	0.7542	0.5854	0.3413	0.1843
5	1.0000	0.9994	0.9954	0.9820	0.9503	0.8889	0.7833	0.6156	0.3635	0.1976
6	1.0000	0.9997	0.9974	0.9877	0.9618	0.9071	0.8073	0.6415	0.3833	0.2096
7	1.0000	0.9999	0.9985	0.9915	0.9703	0.9217	0.8276	0.6642	0.4012	0.2206
8	1.0000	1.0000	0.9991	0.9941	0.9768	0.9335	0.8448	0.6843	0.4175	0.2308
9	1.0000	1.0000	0.9995	0.9959	0.9817	0.9433	0.8598	0.7024	0.4325	0.2402
10	1.0000	1.0000	0.9997	0.9971	0.9855	0.9514	0.8728	0.7187	0.4465	0.2492
15	1.0000	1.0000	1.0000	0.9995	0.9953	0.9764	0.9188	0.7820	0.5047	0.2874
20	1.0000	1.0000	1.0000	0.9999	0.9984	0.9879	0.9459	0.8259	0.5500	0.3185
25	1.0000	1.0000	1.0000	1.0000	0.9994	0.9937	0.9630	0.8582	0.5871	0.3450
30	1.0000	1.0000	1.0000	1.0000	0.9998	0.9966	0.9742	0.8830	0.6185	0.3684
35	1.0000	1.0000	1.0000	1.0000	0.9999	0.9982	0.9819	0.9024	0.6456	0.3892
40	1.0000	1.0000	1.0000	1.0000	1.0000	0.9990	0.9871	0.9180	0.6693	0.4081
45	1.0000	1.0000	1.0000	1.0000	1.0000	0.9995	0.9908	0.9307	0.6905	0.4255
50	1.0000	1.0000	1.0000	1.0000	1.0000	0.9997	0.9933	0.9411	0.7094	0.4415
55	1.0000	1.0000	1.0000	1.0000	1.0000	0.9998	0.9952	0.9497	0.7265	0.4564
60	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999	0.9965	0.9569	0.7420	0.4703
65	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999	0.9974	0.9630	0.7562	0.4833
70	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9981	0.9681	0.7693	0.4957
75	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9986	0.9724	0.7813	0.5073
80	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9990	0.9762	0.7925	0.5184
85	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9993	0.9793	0.8028	0.5290
90	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9995	0.9820	0.8124	0.5390
95	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9996	0.9844	0.8214	0.5486
100	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9997	0.9864	0.8298	0.5578
110	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9998	0.9896	0.8450	0.5751
120	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999	0.9921	0.8584	0.5910
130	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9939	0.8704	0.6059
140	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9953	0.8811	0.6197
150	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9964	0.8907	0.6327
160	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9972	0.8993	0.6449
170	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9979	0.9072	0.6564
180	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9983	0.9142	0.6673
190	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9987	0.9207	0.6776
200	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9990	0.9266	0.6874
250	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9997	0.9493	0.7297
300	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999	0.9643	0.7637
350	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9746	0.7917
400	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9817	0.8152
450	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9867	0.8352
500	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9903	0.8524
600	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9948	0.8803
700	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9971	0.9019
800	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9984	0.9189
900	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9991	0.9325
1000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9995	0.9435

Open adressering.
Kans op overloop.

:ket	vullingsgraad									
	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	0.95
1	53E-4	22E-3	55E-3	10E-2	17E-2	27E-2	39E-2	55E-2	75E-2	87E-2
2	11E-4	86E-4	27E-3	63E-3	12E-2	20E-2	32E-2	49E-2	71E-2	84E-2
3	26E-5	34E-4	14E-3	40E-3	87E-3	16E-2	28E-2	45E-2	68E-2	83E-2
4	61E-6	14E-4	81E-4	26E-3	65E-3	13E-2	24E-2	41E-2	65E-2	81E-2
5	14E-6	59E-5	45E-4	18E-3	49E-3	11E-2	21E-2	38E-2	63E-2	80E-2
6	32E-7	25E-5	26E-4	12E-3	38E-3	92E-3	19E-2	35E-2	61E-2	79E-2
7	76E-8	10E-5	15E-4	84E-4	29E-3	78E-3	17E-2	33E-2	59E-2	77E-2
8	18E-8	45E-6	86E-5	58E-4	23E-3	66E-3	15E-2	31E-2	58E-2	76E-2
9	42E-9	19E-6	50E-5	41E-4	18E-3	56E-3	14E-2	29E-2	56E-2	75E-2
.0	10E-9	83E-7	29E-5	28E-4	14E-3	48E-3	12E-2	28E-2	55E-2	75E-2
:5	0	12E-8	20E-6	51E-5	46E-4	23E-3	81E-3	21E-2	49E-2	71E-2
20	0	0	14E-7	94E-6	15E-4	12E-3	54E-3	17E-2	45E-2	68E-2
25	0	0	10E-8	17E-6	55E-5	63E-4	37E-3	14E-2	41E-2	65E-2
30	0	0	0	33E-7	19E-5	33E-4	25E-3	11E-2	38E-2	63E-2
35	0	0	0	64E-8	70E-6	18E-4	18E-3	97E-3	35E-2	61E-2
40	0	0	0	12E-8	25E-6	99E-5	12E-3	81E-3	33E-2	59E-2
45	0	0	0	24E-9	92E-7	54E-5	92E-4	69E-3	30E-2	57E-2
50	0	0	0	0	33E-7	29E-5	66E-4	58E-3	29E-2	55E-2
55	0	0	0	0	12E-7	16E-5	48E-4	50E-3	27E-2	54E-2
60	0	0	0	0	44E-8	91E-6	35E-4	43E-3	25E-2	52E-2
65	0	0	0	0	16E-8	50E-6	25E-4	37E-3	24E-2	51E-2
70	0	0	0	0	61E-9	28E-6	18E-4	31E-3	23E-2	50E-2
75	0	0	0	0	23E-9	15E-6	13E-4	27E-3	21E-2	49E-2
80	0	0	0	0	0	87E-7	99E-5	23E-3	20E-2	48E-2
85	0	0	0	0	0	49E-7	73E-5	20E-3	19E-2	47E-2
90	0	0	0	0	0	27E-7	53E-5	17E-3	18E-2	46E-2
95	0	0	0	0	0	15E-7	39E-5	15E-3	17E-2	45E-2
00	0	0	0	0	0	86E-8	29E-5	13E-3	17E-2	44E-2
10	0	0	0	0	0	27E-8	15E-5	10E-3	15E-2	42E-2
20	0	0	0	0	0	87E-9	87E-6	79E-4	14E-2	40E-2
30	0	0	0	0	0	27E-9	47E-6	60E-4	12E-2	39E-2
40	0	0	0	0	0	0	26E-6	46E-4	11E-2	38E-2
50	0	0	0	0	0	0	14E-6	35E-4	10E-2	36E-2
60	0	0	0	0	0	0	79E-7	27E-4	10E-2	35E-2
70	0	0	0	0	0	0	44E-7	21E-4	92E-3	34E-2
80	0	0	0	0	0	0	24E-7	16E-4	85E-3	33E-2
90	0	0	0	0	0	0	13E-7	12E-4	79E-3	32E-2
00	0	0	0	0	0	0	74E-8	99E-5	73E-3	31E-2
50	0	0	0	0	0	0	38E-9	28E-5	50E-3	27E-2
00	0	0	0	0	0	0	0	82E-6	35E-3	23E-2
50	0	0	0	0	0	0	0	24E-6	25E-3	20E-2
00	0	0	0	0	0	0	0	71E-7	18E-3	18E-2
50	0	0	0	0	0	0	0	21E-7	13E-3	16E-2
00	0	0	0	0	0	0	0	62E-8	97E-4	14E-2
00	0	0	0	0	0	0	0	32E-9	52E-4	11E-2
00	0	0	0	0	0	0	0	0	28E-4	98E-3
00	0	0	0	0	0	0	0	0	15E-4	81E-3
00	0	0	0	0	0	0	0	0	87E-5	67E-3
00	0	0	0	0	0	0	0	0	49E-5	56E-3


DIRECTE
ORGANISATIE
NIET-SUCCESVOL
ZOEKEN
— OPEN ADRESSERIN
--- KETENS
α' open
α ketens
1/α + ε = 1/α'
C'α
α'
10,0
9,0
8,0
7,0
6,0
5,0
4,0
3,5
3,0
2,5
2,0
1,8
1,6
1,4
1,2
1,0
0
0,5
1,0
1
5
10
100


10,0
9,0
8,0
7,0
6,0
5,0
4,0
3,5
3,0
2,5
2,0
1,8
1,6
1,4
1,2
1,0
C_d ↑
0
0,5
1,0
→ α'
1
5
10
100
I
DIRECTE ORGANISATIE
SUCCESVOL ZOEKEN
— OPEN ADRESSERING
--- KETENS
α' open
α ketens
$\frac{1}{\alpha} + \varepsilon = \frac{1}{\alpha'}$

6.7. Index-directe organisatie

In paragraaf 6.2.4. is reeds genoemd dat men als bucket bij voorkeur een ruimte neemt die aangepast is aan de eenheid van achtergrondgeheugen bijvoorbeeld een spoor van een schijvengeheugen. Een probleem bij directe organisatie kan ontstaan wanneer we een bestand hebben van grote records. De bucketgrootte b wordt dan relatief klein. De vullingsgraad α kan men niet te klein kiezen, want dan kost het gehele bestand weer te veel ruimte.

Een oplossing die men soms kiest is de volgende. Er wordt een tabel samengesteld waarin ieder element bestaat uit een sleutel en een adres. adres. Het adres verwijst naar de plaats waar het record staat en de sleutel is de sleutel van dat record. Zo'n element sleutel+adres wordt opgevat als een klein record en de gehele tabel als een bestand van kleine records. Dit bestand wordt nu direct georganiseerd met een zeer kleine α en een zeer grote b. Daardoor wordt de overloop en de kans op overloop verwaarloosbaar klein. Succesvol zoeken vergt nu altijd 2 stappen en niet-succesvol zoeken 1 stap. Dit systeem noemt men index-direct. De tabel is als het ware een index op het bestand en deze index is direct georganiseerd.

Succesvol zoeken zal in de regel daarmee iets ongunstiger uitvallen. Niet-succesvol zoeken zeker sneller. Omdat voor het bestand nu niet meer ruimte gereserveerd behoeft te worden dan strikt noodzakelijk en de index practisch verwaarloosbaar is, is het geheel qua geheugenruimte zeer voordelig.

We kunnen dit nog toelichten aan de hand van een voorbeeld. Het bestand bestaat uit 40.000 records met 3300 karakters per record. De grootte van het record is dus ongeveer een tekst van 42 regels van 80 symbolen, een volgetypt A4-formaat. In het record bevindt zich een sleutel van 20 karakters. Beschikbaar is een schijvengeheugen met de volgende gegevens.

823 cilinders/volume
10 sporen/cilinder
20160 kar's/spoor
seektime 22 msec
1 omw 16,4 msec.

We bepalen eerst het aantal records per spoor.

$$\frac{20160}{3300} = 6,11$$

dus 6 records per spoor. Daarmee kiezen we b=6. We kiezen eerst een gewone directe organisatie. Bij een gegeven α is het aantal benodigde sporen dan

$$\left\lceil \frac{40000}{6*\alpha} \right\rceil$$

En dit weer gedeeld door 10 en naar boven afgerond het aantal benodigde cilinders voor het primaire gebied. We nemen aan dat we de overloop zullen onderbrengen op afzonderlijke cilinders en met kettingadressen zoals besproken in paragraaf 6.5. Met de formules van paragraaf 6.5. berekenen we C_α, C'_α, ε_b en μ_{b+1}. De overloop is dan $40.000*\varepsilon_b$

Voor de overloop zijn dan nodig:

$\left\lceil \frac{40000*\varepsilon_b}{6} \right\rceil$ sporen en dat zijn

$\left\lceil \frac{\left\lceil \frac{40000*\varepsilon_b}{6} \right\rceil}{10} \right\rceil$ cilinders

Een toegang tot een bucket vergt gemiddeld

$$22 + 8,2 + 16,4 = 46,6 \text{ msec.}$$

Dit bedrag vermenigvuldigd met C_α en C'_α geeft dan de tijden voor succesvol en niet-succesvol zoeken. Voor een aantal waarden van α zijn deze grootheden nu zo berekend.

α		1	0,95	0,9	0,85	0,8
C_α		1,3575	1,3019	1,2514	1,2060	1,1657
C_α'		1,9637	1,8048	1,6609	1,5326	1,4200
E_b		0,1606	0,1412	0,1224	0,1044	0,0875
μ_{b+1}		0,3937	0,3456	0,2983	0,2526	0,2092
primair	sporen	6667	7018	7408	7844	8334
	cilinders	667	702	741	785	834
over-	records	6424	5648	4896	4176	3500
loop	sporen	1071	942	816	696	584
	cilinders	108	95	82	70	59
totaal cilinders		775	797	823	855	893
zoeken in msec	succes	63	61	58	56	54
	niet-succes	92	84	77	71	66

We zien dat voor $\alpha<0,9$ een volume zelfs te klein is voor dit bestand. Voor $\alpha=0,9$ is het precies passend, maar de overloop is alleen een verwachting en er is dan geen enkele speling. Gezien deze getallen zal men dan kiezen voor $\alpha=0,95$ waarbij reeds $C_\alpha=1,3$ en $C_\alpha' = 1,8$ met tijden van 61 msec. en 84 msec.

Nemen we nu een oplossing met index-direct. Een sleutel is 20 karkaters. Voor het adres gebruiken we 4 karakters, dus totaal 24 karakters. Op één spoor passen nu

$$\left\lfloor \frac{20160}{24} \right\rfloor = 80 \text{ elementen van de indextabel.}$$

We brengen niet meer dan 400 elementen per spoor. Het aantal benodigde sporen voor de index is

$$\frac{40000}{400} = 100 \text{ sporen} = 10 \text{ cilinders}$$

Voor het bestand zelf is nodig

$$\left\lceil \frac{40000}{6} \right\rceil = 6667 \text{ sporen, is } \left\lceil \frac{6667}{10} \right\rceil = 667 \text{ cilinders.}$$

Totaal dus 667 + 10 = 677 cilinders. Dit is dus duidelijk minder dan bij de vorige oplossingen. De indextabel wordt nu direct georganiseerd met α = 400/840 en b = 840. Dit geeft C_α = 1; C'_α = 1; E = $1{,}6_{10}$-84; μ = $3{,}3_{10}$-82. De overloop en kans op overloop zijn dus volledig verwaarloosbaar. Echter, succesvol zoeken wordt nu 2 x 46,6 = 93,2 msec. en niet succesvol zoeken 1 x 46,6 = 46,6 msec. Afhankelijk of men via het operating system daar nog invloed op uit kan oefenen kan men bij succesvol zoeken nog iets verdienen door de sporen met de indextabel in het midden van het bestand te zetten. Voor de eerste toegang naar de index is dan de seektime inderdaad 22 msec. maar voor de armverplaatsing van de indextabel naar record positie is dan minder nodig.

7. BESTANSORGANISATIES GEBASEERD OP INDEXEN

7.1. Inleiding

Voor directe toegang tot willekeurige records is in hoofdstuk 6 het principe van de directe organisatie gegeven. De rangschikking van de records in het geheugen is dan redelijk chaotisch (vandaar vaak de benaming random organisatie) en dit geeft problemen wanneer alle of een groot deel van de records een mutatie moeten ondergaan in één run.

Een methode is dan een of andere systematische rangschikking gekoppeld met indextabellen op verschillende niveaus. Deze indextabellen vormen op zich in feite een boomstructuur.

De benamingen voor dergelijke organisaties hangen in feite af of men primair uitgaat van de boomstructuur die men systematisch doorloopt of de primair systematische rangschikking waarop men een boomstructuur aanbrengt. Zo kunnen ook de binaire bomen besproken in hoofdstuk 6 hieronder worden gerangschikt omdat men via het binaire zoekproces willekeurige records kan vinden en door het systematisch aflopen van de boom ook alle records in een bepaalde volgorde kan bereiken.

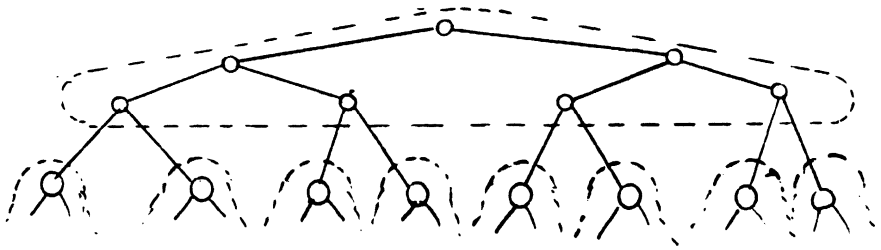
In paragraaf 7.2. wordt nog wat nader ingegaan hoe men binaire bomen ook op achtergrondgeheugen kan gebruiken. Legt men de nadruk op de systematische rangschikking en dan in het bijzonder de sequentiële en brengt men hierop een boom van indices aan, dan ontstaat de index-sequentiële bestandsorganisatie zoals besproken in paragraaf 7.3. Gaat men primair uit van de boomstruktuur dan gaat de index-sequentiële organisatie over in B-bomen zoals besproken in 7.4.

Voor al deze organisaties geldt in het algemeen nog het volgende. Al deze vormen van index-organisaties vereisen een adresseerbaar achtergrondgeheugen. Bij deze vormen is de gemiddelde zoektijd zowel succesvol als niet-succesvol praktisch altijd groter dan bij een directe organisatie. Door de boomstruktuur zal de zoektijd ook toenemen bij grotere bestanden in tegenstelling tot de directe organisatie waar het gemiddelde juist onafhankelijk is van de bestandsgrootte. De indexorganisaties garanderen wel bovengrenzen van zoektijden ook voor de slechtste gevallen. Bij directe organisatie kan

een slechtste geval ook zeer slecht worden.

7.2. Binaire bomen op achtergrondgeheugen

Het zoeken van een element in een tabel die een binaire boom representeert is efficiënt als de tabel in zijn geheel in het interne geheugen staat. Bevindt de boom zich op achtergrondgeheugen en zoeken we op de gewone manier, dan wordt het aantal toegangen tot het achtergrondgeheugen ongeveer $^2\log n$ als n het aantal knopen is. Voor een boom met 10^6 knopen betekent dit gemiddeld 20 toegangen.
Het aantal toegangen kan worden verminderd door de boom te verdelen in pagina's. Zo'n pagina of blok wordt dan als eenheid tussen inter- en achtergrondgeheugen getransporteerd, waardoor met één toegang dan de inhoud van een aantal knopen ter beschikking staat.



In bovenstaande boom bevat elk blok 7 sleutels. Ieder blok verwijst zelf naar 8 andere blokken. Door deze verdeling is de boom van tweetallig eigenlijk achttallig geworden. Wanneer de ruimte het toelaat kunnen we ook nog meer niveaus in één blok onderbrengen. We kunnen ook hier de hoogte weer verder beperken door de binaire boomstruktuur bijvoorbeeld uit te voeren in de vorm van een AVL-boom. Bij een dergelijke organisatie is het aantal knopen per blok gelijk. We kunnen daarom dan een vaste struktuur aannemen waardoor we de "linker" en "rechter" adresverwijzingen impliciet maken. Volstaan we in de boom met alleen de sleutels en de adresverwijzing naar de plaats waar het record zelf staat, dan kunnen we in een blok (meestal een spoor op het schijvengeheugen) vrij veel knopen opnemen. Het succesvol zoeken eindigt dan bij het ontmoeten van een externe knoop terwijl bij succes-

vol zoeken nog één extra toegang nodig is voor het record zelf.

Ook al organiseert men de totale boom als AVL-boom waardoor de hoogte beperkt blijft is het wel zo dat de hoger genummerde lagen dus richting externe knopen steeds dunner bezet worden. Door de geblokte uitvoering betekent dit dat de vullingsgraad voor de blokken op hoger niveau erg ongunstig kan worden, waardoor het totaal aantal benodigde blokken excessief toeneemt.
Nu kan men natuurlijk altijd door verdere reorganisatie ervoor zorgen dat de blokken weer beter gevuld worden en daarmee het aantal blokken verminderd. Als n het aantal records en v het maximum aantal records/blok dan is een situatie mogelijk waarbij het aantal blokken is $\lceil \frac{n}{v} . \rceil$ Dit aantal is uiteraard het minimum. Naarmate het aantal beschikbare blokken een groter overschot vertoont t.o.v. dit minimum zal men minder vaak compleet moeten reorganiseren. Overigens worden de zoektijden weer extra gunstig beïnvloed door een dergelijke extra reorganisatie omdat niet alleen het totaal aantal blokken afneemt, maar daarmee ook de gemiddelde hoogte.

Ondanks deze gunstige factor wordt deze methode minder gebruikt wegens de genoemde extra reorganisatie die soms nodig is. De frequentie van deze reorganisaties hangt overigens wel af van het gemiddelde "gedrag" van de sleutels bij mutaties. Dit probleem treedt overigens ook op bij index-sequentiële organisatie en komt daar nog nader ter sprake.

7.3. Index-sequentiële organisatie

7.3.1. Principe index-sequentieel

Deze vorm van organisatie is ontstaan met de komst van de schijvengeheugens. De indeling van de boomstructuur was daarbij vooral gericht op de physieke structuur en hiërarchie van deze geheugens en wel volgens

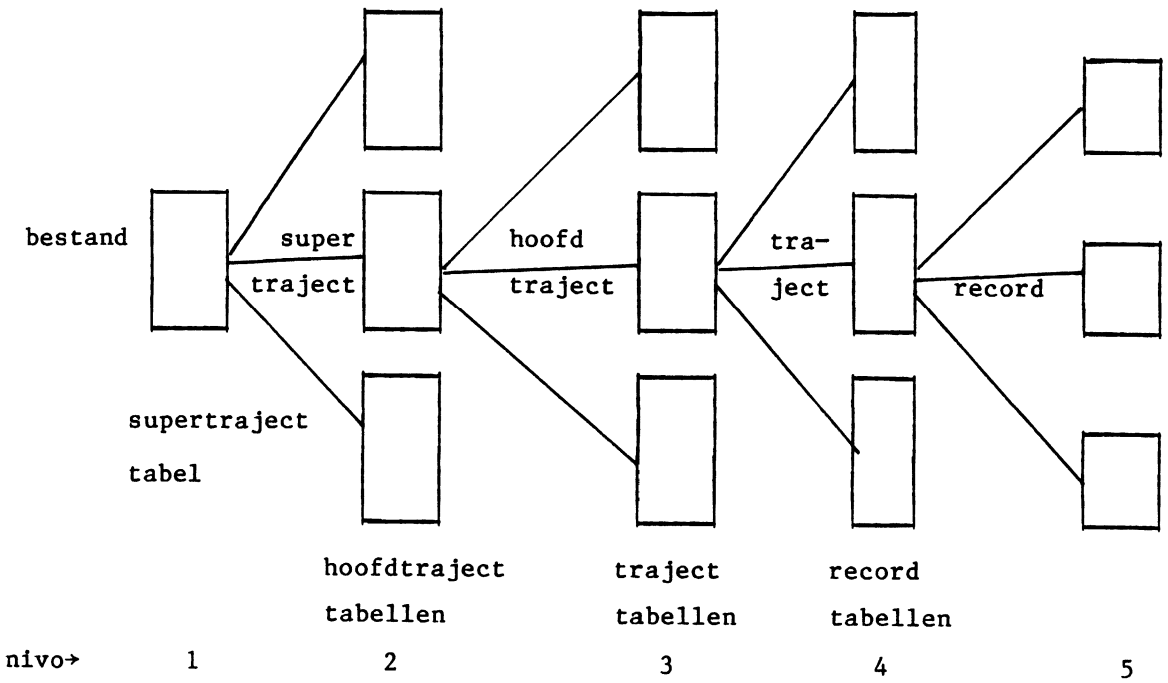
volume → cylinder → spoor → record.

In het streven naar scheiding tussen logische en physische opslagstructuur heeft men deze genoemde indeling verlaten en geeft voor de verschillende niveaus in de boom dan andere benamingen. We zullen dit verder trajecten noemen.

Omdat bij deze boomstructuur het aantal vertakkingen relatief groot is blijft het aantal niveaus beperkt. Er ontstaat daardoor het volgende beeld. Een bestand is verdeeld in supertrajecten. De adresverwijzingen naar deze supertrajecten zijn verenigd in de supertrajecttabel die daarmee tevens de wortel van de boom vormt.

Ieder supertraject is verdeeld in hoofdtrajecten. De adresverwijzingen naar hoofdtrajecten binnen een supertraject zijn gerangschikt in een hoofdtrajecttabel. Deze hoofdtrajecttabellen vormen de knopen in de tweede laag van de boom. Daarop volgt de verdeling van hoofdtrajecten in trajecten waarbij een trajecttabel een knoop vormt op de derde laag.

Tenslotten bevinden zich op de vierde laag de trajecten die ieder bestaan uit een aantal records, zodat een knoop op deze laag dan identiek is met een aantal records. In uitvoeringen met zeer grote records kan het overigens voorkomen dat deze records als knopen op een vijfde laag worden gerangschikt en een traject dan in feite bestaat uit een recordtabel die naar een aantal records verwijst.



Wanneer alle tabellen zich op achtergrondgeheugen bevinden betekent ieder niveau een onafhankelijke toegang. In het geschetste geval dus 5 maal. In de meeste gevallen is dit aantal kleiner. Ten eerste ontbreekt in vele gevallen niveau 5 en zijn de records zelf binnen de trajecten opgenomen. Ten tweede zijn vele bestanden toch wel zo beperkt van omvang dat men in de geschetste hiërarchische opbouw niet verder komt dan een aantal hoofdtrajecten die slechts één supertraject vormen. Dit supertraject is dan vanzelfsprekend tevens het bestand en in de schets vervalt dan eigenlijk niveau 1.

Het eerste en tweede geval gecombineerd beperkt het aantal niveaus al tot 3. Ten derde wordt de wortel van de boom altijd gevormd door 1 tabel hetzij de supertrajecttabel of de hoofdtrajecttabel. Moet men het bestand meerdere malen na elkaar raadplegen dan loont het deze worteltabel permanent in het primaire geheugen te houden. Het aantal toegangen tot achtergrondgeheugen wordt dan weer met 1 stap beperkt.

Een verdere maatregel om de totale toegangstijd te beperken is een vorm van "platdrukken" van deze boom. In het maximale geval met 5 niveaus gaat men als volgt te werk. Alle records die samen een traject vormen worden achter elkaar in het geheugen geplaatst met in het midden ruimte voor de recordtabel. Op deze wijze worden trajecten achter elkaar geplaatst met steeds een hoofdtrajecttabel in het midden van de serie trajecten waarop deze betrekking heeft. Op dezelfde manier wordt iedere supertrajecttabel in het midden van zijn hoofdtrajecten geplaatst. En tenslotten de supertrajecttabel geheel in het midden. Mits het achtergrondgeheugen niet gelijktijdig door andere programma's wordt gebruikt worden door deze organisatie de armbewegingen geminimaliseerd.

Met de geschetste vormgeving van index-sequentiële organisatie is aan twee voorwaarden voldaan. Ten eerste is een sequentiële organisatie gehandhaafd en kunnen alle bewerkingen die gebruikelijk zijn voor die organisatie ook hier worden uitgevoerd. Ten tweede is een directe toegang mogelijk gemaakt door de introductie van de indextabellen. Met deze vorm is nog geen oplossing voor het derde probleem, namelijk de mogelijkheid tot weglaten en toevoegen van records zonder het hele bestand te moeten copiëren. Alle oplossingen voor dit probleem zijn gebaseerd op het volgende principe.

Bij het vaststellen van de beschikbare ruimte voor het bestand reserveert

men een grotere ruimte dan strikt noodzakelijk. De extra beschikbare ruimte dient als overloopruimte voor die records die niet op hun plaats kunnen staan, enigszins vergelijkbaar met de situatie bij directe organisaties. De ruimte waarin de records aanvankelijk of na een reorganisatie worden geladen noemt men de primaire ruimte. De vele verschillende oplossingen die men voor dit probleem kiest hangen samen met de wijze waarop men die indeling in primaire en overloopruimte tot stand brengt.

Omdat overloop bij deze systemen veelal wordt gerealiseerd door kettingadressering kan daardoor het gemiddeld aantal stappen voor toegang iets hoger uitkomen dan hiervoor aangegeven. De verschillende varianten worden voornamelijk bepaald door de wijze waarop de overloop over trajecten en hoofdtrajecten wordt verdeeld. Men kiest daarbij de overloopruimte zo groot dat overloop boven hoofdtrajectniveau niet meer voorkomt. Mocht dit gevaar na veel mutaties toch dreigen, dan gaat men over tot zogenaamde reorganisatie. Dit houdt in dat het gehele bestand wordt gekopieerd, waarbij de vrije ruimte opnieuw gelijkmatig wordt verdeeld. Tegelijkertijd worden alle indextabellen weer opnieuw opgebouwd.

Zoals reeds opgemerkt was index sequentiële organistatie aanvankelijk vooral bepaald door de physieke structuur van schijvengeheugens. Een traject was een spoor; een hoofdtraject een cylinder en een supertraject een volume. Men sprak dan over spoortabel, cylindertabel en volumetabel.

In het streven applicatieprogramma's onafhankelijk te maken van de gebruikte apparatuur heeft men dit systeem verlaten en is weer overgegaan op indeling in meer logische eenheden die hier trajecten worden genoemd. Dit laatste sluit niet uit dat men zich in sommige gevallen bij de keuze van de trajectgrootte toch nog wel laat leiden door te gebruiken hardware eigenschappen. Zou men bijvoorbeeld een trajectgrootte kiezen die juist iets meer is dan een spoor dan is duidelijk dat dit onevenredig slecht werkt voor toegangstijden.

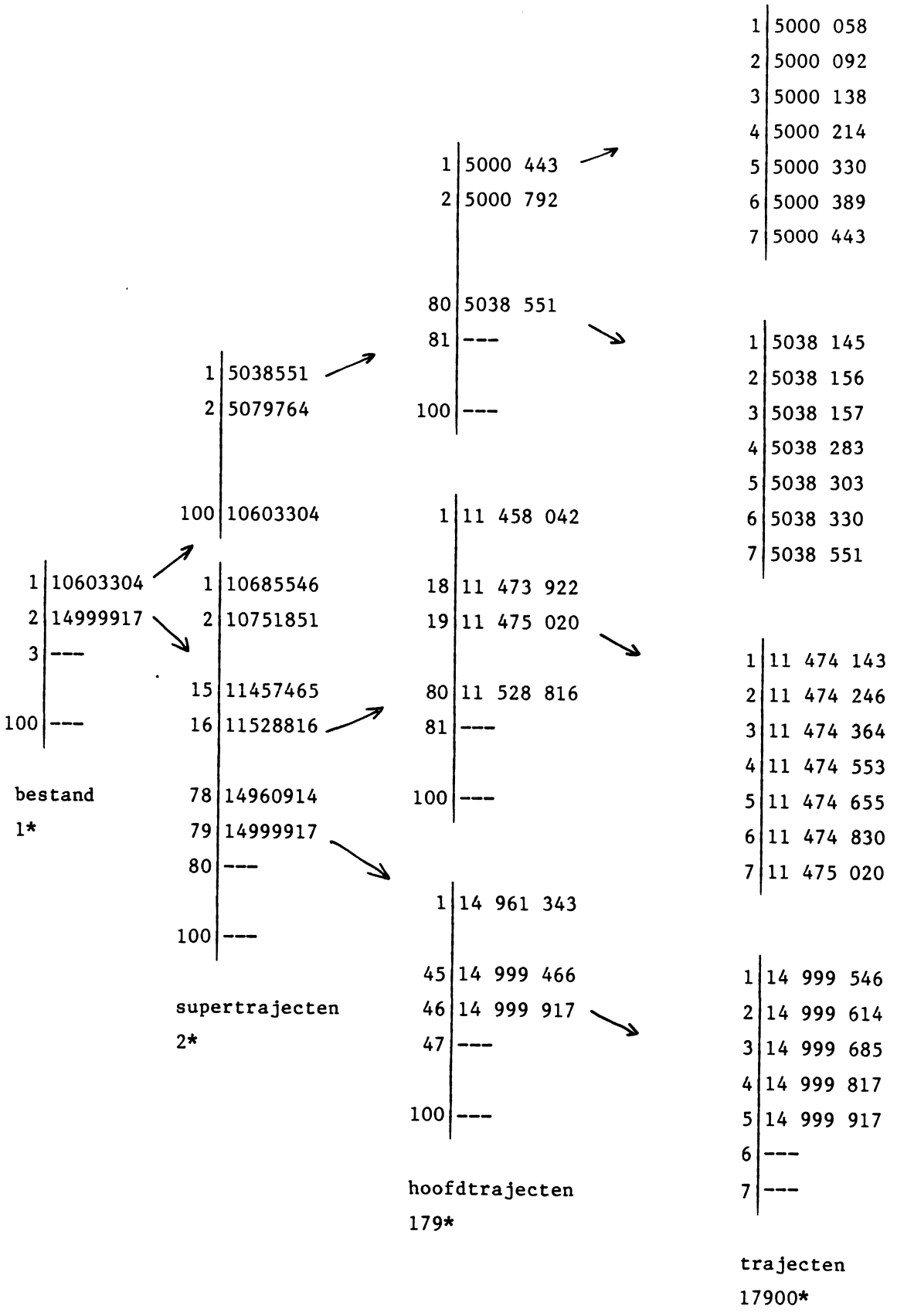
Liever bouwt men het systeem dan zo dat bij reorganisatie de trajectgrootte nog als parameter kan worden opgegeven. Gaat men dan over tot vervanging van b.v. schijvengeheugens door nieuwere typen dan moet toch gekopieerd worden en men doet dit dan door een totale reorganisatieslag.

7.3.2. Voorbeelden van index-sequentiëel

Als voorbeelden beschouwen we 2 varianten in uitvoering van hetzelfde bestand. Dit bestand bestaat uit 100.000 records van ieder 500 karakters. Als traject kiezen we een grootte van 7 records d.w.z. 3500 karakters. In verband met het eerder genoemde "platdrukken" van de boom en dus het plaatsen van indextabellen tussen recordblokken verdient het de voorkeur ook alle indextabellen de grootte van een blok te geven. We nemen aan dat de sleutel in ieder record een grootte heeft van 35 karakters. Dat betekent dat we iedere indextabel dan 100 ingangen geven.

Een eerste variant is een initiële lading van het bestand met een bezettingsgraad van 100% per traject en 80% per hoofdtraject. Dit betekent dus 7 records per traject. Per hoofdtraject 80 trajecten vol en 20 trajecten reserve. Een hoofdtraject bevat dan 80*7 = 560 records. Nu is 100.000 = 14.285*7+5 dus nodig 14.286 trajecten. 14.286 = 178*80+46 dus 179 hoofdtrajecten 179 = 1*100+79 dus 2 supertrajecten.

Om een voorbeeld te geven hoe zo'n indeling er uit zou kunnen zien vereenvoudigen we de sleutel voor het moment tot een decimaal getal van 8 cijfers in het bereik van 5_{10}+6 tot 15_{10}+6 en nemen aan dat de sleutels gelijkmatig verdeeld zijn. Ten behoeve van het voorbeeld zijn de sleutels eenvoudig met een randomgenerator bepaald.


1 10603304
2 14999917
3 ---
100 ---
bestand
1*
1 5038551
2 5079764
100 10603304
1 10685546
2 10751851
15 11457465
16 11528816
78 14960914
79 14999917
80 ---
100 ---
supertrajecten
2*
1 5000 443
2 5000 792
80 5038 551
81 ---
100 ---
1 11 458 042
18 11 473 922
19 11 475 020
80 11 528 816
81 ---
100 ---
1 14 961 343
45 14 999 466
46 14 999 917
47 ---
100 ---
hoofdtrajecten
179*
1 5000 058
2 5000 092
3 5000 138
4 5000 214
5 5000 330
6 5000 389
7 5000 443
1 5038 145
2 5038 156
3 5038 157
4 5038 283
5 5038 303
6 5038 330
7 5038 551
1 11 474 143
2 11 474 246
3 11 474 364
4 11 474 553
5 11 474 655
6 11 474 830
7 11 475 020
1 14 999 546
2 14 999 614
3 14 999 685
4 14 999 817
5 14 999 917
6 ---
7 ---
trajecten
17900*

In dit voorbeeld hebben we symbolisch in de trajecten alleen de sleutels vermeld, terwijl in werkelijkheid trajecten gehele records van 500 karakters bevatten. Echter, in de indextabellen komen alleen maar sleutels voor en geen adresverwijzingen! Immers, het indexnummer in een trajecttabel is evenredig met het relatieve adres van het bijbehorende traject. Zo ook het indexnummer in een hoofdtrajecttabel is evenredig met het adres van het bijbehorende hoofdtraject enz.

Wanneer records gesorteerd zijn naar opklimmende sleutelwaarde, dan staan in de indextabellen steeds de hoogste bijbehorende sleutelwaarden. In dit voorbeeld bestaat ieder hoofdtraject uit een trajecttabel met 100 ingangen. Na de eerste lading en voordat enige mutatie is uitgevoerd, bevatten de eerste 80 ingangen sleutels die betrekking hebben op 80 achtereenvolgende trajecten en van ieder traject is steeds de hoogste sleutel in de trajecttabel opgenomen. Na 80 vol bezette trajecten volgen 20 trajecten die leeg zijn en de bijbehorende ingangen in de trajecttabel zijn opengelaten.

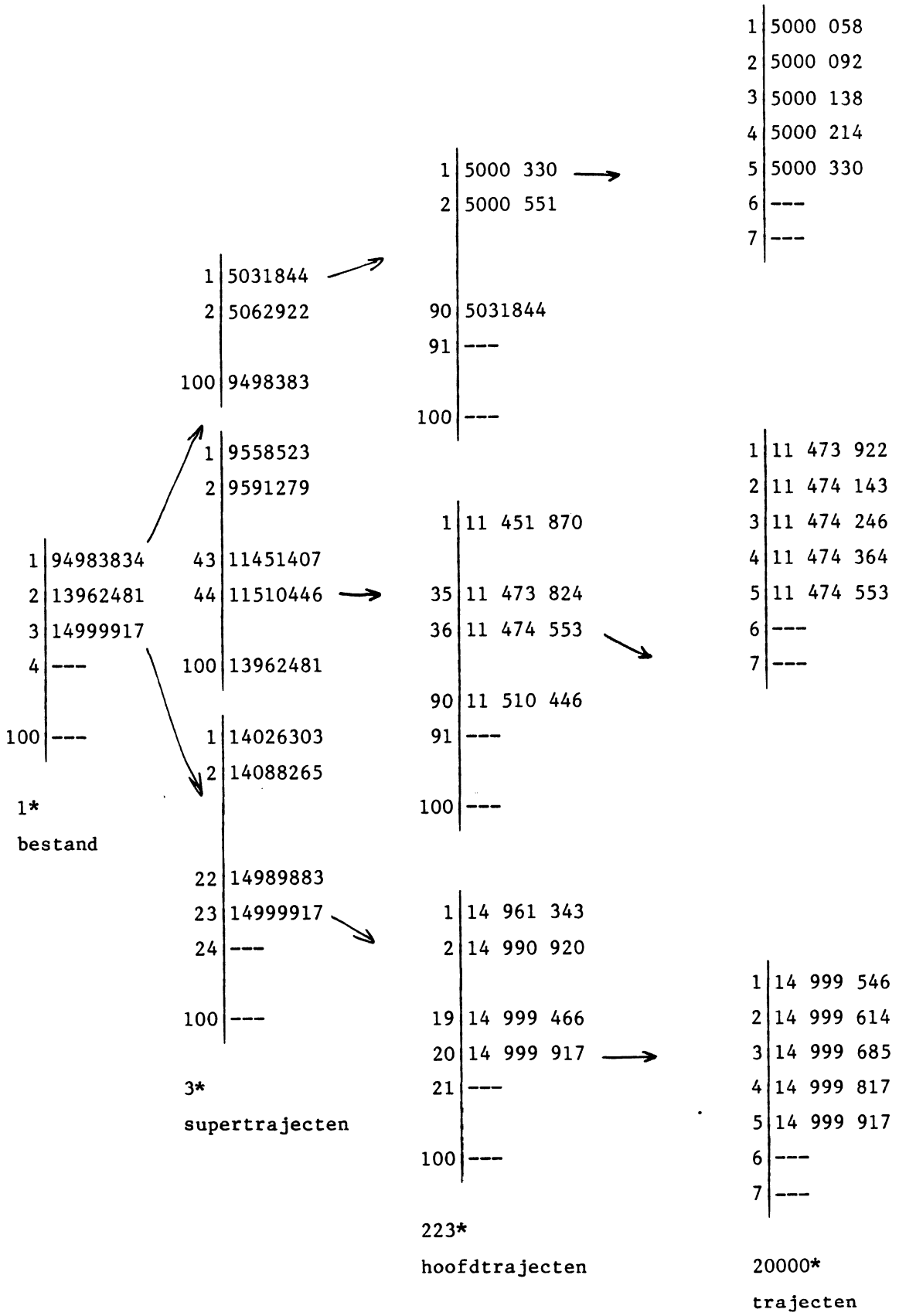
Op dezelfde manier bestaat in dit voorbeeld ieder supertraject uit een hoofdtrajecttabel met 100 ingangen. Er zijn er steeds 2 waarvan de eerste geheel gevuld is en de tweede alleen de eerste 79 ingangen. Tenslotte: het bestand wordt gevormd door één supertrajecttabel met maar 2 ingangen. Het zoeken naar een record gebeurt door de gewenste sleutel sequentiëel in iedere tabel op te zoeken. Bij het sequentiëel zoeken stopt men wanneer in de betrokken tabel een sleutel wordt gevonden die groter of gelijk is aan de gewenste.

Bijvoorbeeld zoek record met sleutel 11 474 553. In de supertrajecttabel geeft dit de tweede ingang, want 14 999 917 > 11 474 553. In de bijbehorende hoofdtrajecttabel vinden we ingang 16 want 11 528 816 is de eerste sleutel waarvoor geldt > 11 474 553. Vervolgens vinden we ingang 19 in de betrokken trajecttabel en tenslotte het 4e record in het traject. Zouden we zoeken naar 11 474 600 dan volgen we dezelfde weg met alleen het verschil dat in de laatste stap blijkt dat dit record niet in het traject en dus in het totale bestand niet voorkomt.

Gaan we na de eerste lading mutaties aanbrengen dan kan het zijn dat in een traject een record zou moeten worden toegevoegd. Echter, de trajecten zijn vol. Daarom wordt het overlooprecord dan geplaatst in de vrije ruimte van

het bijbehorende hoofdtraject, terwijl in het traject een verwijzing wordt opgenomen naar die plaats. Voor een dergelijke adresverwijzing moet in een traject dus nog wel plaats zijn.
Voor dit voorbeeld nemen we daarom aan dat een blok dat als transporteenheid dienst doet tussen primair en secundair geheugen iets meer dan 3500 karakters groot is.

Hoe die verwijzing gebeurt laten we zien aan da hand van een 2e variant voor ditzelfde bestand. Bij deze variant gaan we uit van een verdeling van de reserveruimte over traject- en hoofdtrajectniveau, en wel als volgt. Per traject is de eerste lading 5 records dus een vullingsgraad van 5/7. Per hoofdtraject is de eerste lading 90 trajecten dus een vullingsgraad van 9/10. Nu geldt 100.000 = 20.000 x 5 dus nodig 20.000 trajecten 20.000 = 222*90+20 dus nodig 223 hoofdtrajecten.
223 = 2*100+23 dus nodig 3 supertrajecten. Hetzelfde voorbeeldbestand ziet er in deze uitvoering uit als volgt.


1 94983834
2 13962481
3 14999917
4 ---
100 ---
1*
bestand
1 5031844
2 5062922
100 9498383
1 9558523
2 9591279
43 11451407
44 11510446
100 13962481
1 14026303
2 14088265
22 14989883
23 14999917
24 ---
100 ---
3*
supertrajecten
1 5000 330
2 5000 551
90 5031844
91 ---
100 ---
1 11 451 870
35 11 473 824
36 11 474 553
90 11 510 446
91 ---
100 ---
1 14 961 343
2 14 990 920
19 14 999 466
20 14 999 917
21 ---
100 ---
223*
hoofdtrajecten
1 5000 058
2 5000 092
3 5000 138
4 5000 214
5 5000 330
6 ---
7 ---
1 11 473 922
2 11 474 143
3 11 474 246
4 11 474 364
5 11 474 553
6 ---
7 ---
1 14 999 546
2 14 999 614
3 14 999 685
4 14 999 817
5 14 999 917
6 ---
7 ---
20000*
trajecten

Wanneer we op dit bestand mutaties gaan uitvoeren en in het bijzonder wanneer records moeten worden toegevoegd dan ligt het voor de hand dat eerst de betrokken trajecten worden opgevuld. Bijvoorbeeld toevoegen van records met sleutels 11 474 189 en 11 474 293 dan ontstaat.

		1	9 558 523	1	11 451 870	36→	1	11 473 922
							2	11 474 143
1	9498 383	43	11 451 407	36	11 474 553 →		3	11 474 189
2	13 962 481 →	44	11 510 446 →				4	11 474 246
3	14 999 917			90	11 510 446		5	11 474 293
		100	13 962 481				6	11 474 364
				100	---		7	11 474 553

Binnen het betrokken traject zijn sommige records opgeschoven om ruimte te maken voor de nieuwe. De index tabellen blijven ongewijzigd en door deze toevoegingen is ook de gemiddelde zoektijd nog niet veranderd.

Nu voegen we toe record met sleutel 11 473 873. Qua sleutel hoort deze dus thuis in het 36e traject van het 44e hoofdtraject van het 2e supertraject. Omdat het traject zelf vol is brengen we de overloop onder in het overloopgebied van het bijbehorende hoofdtraject. Omdat er misschien al meer overloop geweest is voor dit hoofdtraject wordt daar voor aangewezen de 5e plaats in het 93e traject. Gebruikelijk is de index tabellen ongewijzigd te laten. Dus in dit geval 11 474 553 blijft de grootste sleutel die in het 36e traject thuishoort. Men kan nu of zonder meer het nieuwe record in de overloop plaatsen of het nieuwe record inschuiven op zijn plaats in het traject en daarna het record met de grootste sleutel verplaatsen naar het overloopgebied. Voor terugzoeken is deze methode iets aantrekkelijker omdat dan alleen voor dit record één extra toegang naar het overloopgebied nodig is terwijl bij de eerste methode bij niet succesvol zoeken voor ieder interval in dit traject toch het overlooprecord bekeken moet worden. Voeren we de aanvulling uit op de tweede manier dan ontstaat

1	9 498 383
2	13 962 481 43
3	14 999 971 44

1	9 558 523
100	13 962 481

1	11 451 870
36	11 474 553
90	11 510 446
100	---

1	11 451 407
90	11 510 446
100	---

1	11 451 870	
36	11 474 553	
90	11 510 446	
100	---	

36→	1	11 473 873
	2	11 473 922
	3	11 474 143
	4	11 474 189
	5	11 474 246
	6	11 474 293
	7	11 474 364
		93.5

93→	1	---
	2	---
	3	---
	4	---
	5	11 474 553 | 0.0
	6	---
	7	---

In het 36e traject is een adresverwijzing 93.5 opgenomen. In het overloopgebied is achter record 11 474 553 opgenomen. 0.0 ten teken dat de ketting is geëindigd. Bij het ontwerp moeten we dus rekening houden dat in een primair traject nog ruimte is voor één adresverwijzing. In een overlooptraject moet bij ieder record nog ruimte zijn voor een adresverwijzing. Aangezien de blokgrootte voor transport meestal een gegeven is kan dit betekenen dat in een overlooptraject minder records geplaatst kunnen worden dan in een primair traject. Zouden we in dit voorbeeld hierna nog toevoegen 11 474 445 dan wordt ook deze eerst weer ingepast in het betrokken traject. Daarna valt de grootste uit het primaire project weer af en dat is toevallig dit nieuwe record zelf. Deze gaat nu naar een overloopplaats, zeg 91.4 en wordt wel op de juiste plaats in de ketting ingehangen. De relevante trajecten komen er nu zo uit te zien.

36→	1	11 473 873		91→	1				93→	1	
	2	11 473 922			2					2	
	3	11 474 143			3					3	
	4	11 474 189			4	11 474 445	93.5			4	
	5	11 474 246			5					5	11 474 553	0.0
	6	11 474 293			6					6	
	7	11 474 364			7					7	
		91.4									

Het is duidelijk dat door het toevoegen van records in overlooptrajecten de gemiddelde zoektijd zal toenemen. Bij het verwijderen zijn ook nog verschillende strategieën mogelijk. Een eenvoudige methode is de betrokken record positie aleen van een merkteken te voorzien dat dit record niet meer bestaat. De mutatie zelf wordt op die manier het snelste verwerkt. Echter, wanneer er voor het betrokken traject al overloop bestaat is de gemiddelde kettinglengte dan groter dan noodzakelijk. Wanneer men op die wijze doorgaat wordt een totale reorganisatie na verloop van tijd noodzakelijk. Ten koste van iets meer werk bij de mutatie kunnen we dan beter "terugschuiven" en de ketting die aan het betrokken traject hangt met 1 inkorten.

Op deze wijze zijn in combinatie nogal een aantal variaties in toevoeg- en weglaatalgorithme mogelijk. Algemeen kan het volgende worden opgemerkt. Als door mutaties weliswaar records verdwijnen en bijkomen maar de verdeling van sleutels over mogelijke sleutelwaarden blijft gelijk dan verdient het aanbeveling kettingen zo kort mogelijk te houden. Reorganisatie zal dan maar sporadisch nodig zijn. Hebben we een verlopend sleutelbereik dan moeten we toch van tijd tot tijd reorganiseren. In dat geval heeft het weinig zin bij mutaties veel extra werk te doen. Als voorbeeld van dit laatste een studentenbestand met als sleutel een inschrijvingsnummer. Als het inschrijvingsnummer bijvoorbeeld gebaseerd wordt op datum van eerste inschrijving, dan schuift in de tijd gezien het gehele sleutelbereik steeds op. Bij indexsequentiële organisatie moet dan worden gereorganiseerd als de opschuiving van sleutels te groot is geworden. Immers, aan het begin van het bestand vallen dan gaten terwijl aan het eind alle nieuwe records in overloop terecht komen.
Dit in tegenstelling tot b.v. de organisatie met geblokte AVL-bomen en de nog te bespreken B-bomen die in feite een automatisch reorganisatie mechanisme hebben.

7.3.3. Berekeningen met index-sequentiëel

Zoals reeds vermeld is index-sequentiëel meer geschikt als het sleutelgedrag niet te sterk wijzigt. In dat geval kunnen we een deel van de theorie van directe organisatie hier ook toepassen. Een traject kunnen we vergelijken met een bucket. Voor gemiddelde zoektijd, overloop en overloopkans kunnen we zonder meer de formules in paragraaf 7.5.3. toepassen voor de overloop op trajectniveau. Iets anders is de situatie ten aanzien van overloop op het hoofdtrajectniveau. In de regel is echter het aantal primaire trajecten per hoofdtraject relatief groot waardoor we met redelijke benadering ook op hoofdtrajectniveau een Poisson verdeling mogen aannemen. Daarom passen we ook daar de formules van paragraaf 7.5.3. toe. We geven als voorbeeld de twee varianten uit de vorige paragraaf. De eerste variant

1 traject = 1 bucket = 7 records

vullingsgraad 100%. Dus α = 1,0 en b= 7 geeft C_α = 1,3496 C'_α = 2,0430 ε_b = 0,15 μ_b = 0,40

Per hoofdtraject hebben we 80 primaire trajecten. Dit geeft een overloop van 80*7*0,15 = 84. Hiervoor is beschikbaar 20*7 = 140 overloop posities dus op dit niveau α = 84/140 en b = 140. Dit levert $\varepsilon_b = 2{,}5_{10}\text{-}10$ en $\mu_b = 8{,}9_{10}\text{-}9$. Alleen deze grootheden zijn hier van belang. Immers vooral de kans dat op dit niveau nog overloop zal optreden moet verwaarloosbaar klein zijn. Dit is met $8{,}9_{10}$-9 wel het geval. Er is een totale geheugenreservering nodig als volgt. VOor primair gebied zijn nodig

$\left\lceil \frac{100\ 000}{7} \right\rceil$ = 14.286 trajecten.

Hiervoor $\left\lceil \frac{14.286}{80} \right\rceil$ = 179 hoofdtrajecten, waarbij ieder hoofdtraject wel 100 trajecten omvat waarvan 80 primair en 20 overloop. Er zijn $\left\lceil \frac{179}{100} \right\rceil$ = 2 super-trajecten. Totaal aan trajecten van 3500 karakters is dus

1 + 2 + 179 + 179*100 = 18082

en dit voor 100.000 records, dus een gemiddelde bezettingsgraad voor het geheel van

$$\frac{100.000}{18082*7} = 0,790$$

Tenslotte de zoektijden. Neem aan dat we de suptertrajecttabel en de twee hoofdtrajecttabellen permanent in het primaire gebied houden. Voor een record dat in zijn primaire gebied staat zijn dan 2 toegangen nodig. Door de overloop wordt dit meer, en wel

$C_\alpha = 2,3496$ en $C'_\alpha = 3,0430$.

We beschouwen nu de 2e variant. Per traject geldt:

$\alpha = 5/7$ en $b = 7$, dus

$C_\alpha = 1,0934$ $C'_a = 1,2555$ $\varepsilon_b = 0,0511$ $\mu_b = 0,133$.

Overloop op hoofdtraject niveau
90*5*0,0511 = 22,995 records.
Hiervoor beschikbaar 10*7 = 70 plaatsen.
Dus $\alpha = 22,995/70$ $b = 70$ geeft $\varepsilon = 5,4_{10}-17$ en $\mu = 8,4_{10}-16$
Deze kans is klein genoeg.

Totaal nodig

$$\left\lceil \frac{100.000}{5} \right\rceil = 20.000 \text{ primaire trajecten}$$

$$\left\lceil \frac{20.000}{90} \right\rceil = 223 \text{ hoofdtrajecten}$$

$$\left\lceil \frac{223}{100} \right\rceil = 3 \text{ suptertrajecten}$$

Totaal 1 + 3 + 223 + 223*100 = 22527 trajecten.

Totale bezettingsgraad:

$$\frac{100000}{7 \times 22527} = 0,634$$

Totaal wordt nu C_α = 2,0934 en C'_α = 2,2555. Ten koste van meer geheugen beperkt men dus zoektijden waarvan vooral het niet-succesvol zoeken, namelijk van 3,04 naar 2,26 terwijl verder bij de 2e variant de overloopkans op hoofdtraject nog een orde kleiner is geworden, namelijk van $_{10}{}^{-9}$ naar 10^{-6}.

7.4. B-bomen

7.4.1. Principe en typen B-bomen

Bij de index sequentiële organisatie wordt het probleem van later invoegen van records opgevangen door reeds bij de initiële lading of bij reorganisatie op verschillende niveaus overloopgebieden te reserveren. Tot aan een volgende reorganisatie is deze verdeling dan gefixeerd.

Een ander idee is om bij lading nog geen ruimte te reserveren. Komt een deelgebied door toevoeging van records op een gegeven moment ruimte tekort, dan gebeurt het volgende. Aan dit volle deelgebied wordt een leeg deelgebied toegevoegd. De inhoud wordt gelijk verdeeld, zodat ieder voor de helft wordt gevuld en uit één vol deelgebied zijn twee halfvolle deelgebieden gecreëerd. Dit idee is door R. Baywer en C. McCreight in 1972 uitgewerkt in de door hen geïntroduceerde B-boom.

Een B-boom van de orde K heeft de volgende eigenschappen:

- elk pad van de wortel naar een blad heeft dezelfde lengte;
- behalve de wortel bevat ieder knooppunt minstens K records en maximaal 2K records;
- de wortel bevat minimaal 1 record en maximaal 2K records;
- de bladeren hebben geen opvolgers of zonen;
- als een knooppunt p records bevat en geen blad is bevat het p+1 verwijzingen naar p+1 zonen.

De bladeren zijn de knooppunten op het hoogste niveau. De "zonen" van de bladeren zijn de externe knopen die zelf geen informatie bevatten en dus ook niet echt in de boom aanwezig hoeven te zijn. Voor de opslag op achtergrondgeheugen wordt uitgegaan van pagina's (blokken), waarbij elke pagina de inhoud van één knooppunt bevat. Binnen de pagina staan de sleutels gerangschikt volgens toenemende grootte.

Een pagina ziet er als volgt uit:

p_0	s_1	p_1	s_2	p_2	s_3	p_3	..	s_1	p_1	vrije ruimte

De bij de sleutels behorende gegevens - rest van de records - zijn in de figuur weggelaten. S_i stelt een sleutel voor en P_i een verwijzing naar een andere pagina (zoon).

Is $P(p_i)$ de pagina waarheen p_i wijst en $S(p_i)$ de verzameling sleutels van de pagina's in de subboom waarvan $P(p_i)$ de wortel is, dan geldt:

$0 \leq i \leq q$

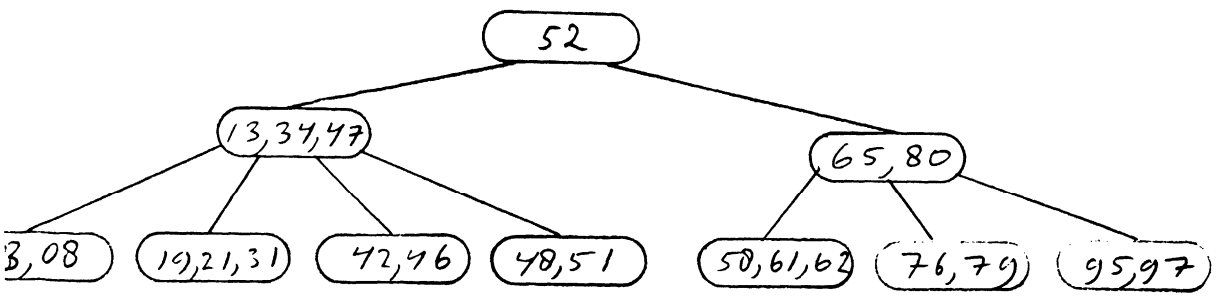
$\forall y \in S(P_0) : y < s_1$

$\forall y \in S(p_i) : S_i < y < s_{i+1}$ (i=1, 2, ... q-1)

$\forall y \in S(p_1) : y > s_1$

Voorbeeld

k=2



Bij dit type B-bomen bevinden de records zich dus in de knopen. Het is direct duidelijk dat voor grotere waarden van K het aantal niveaus beperkt blijft. Dat betekent dat men ieder record met enkele toegangen kan bereiken. Voor grotere records wordt een knoop dan wel erg groot in omvang zodat een deel van het voordeel weer teniet wordt gedaan. Een oplossing kan dan zijn in de knopen alleen de sleutels op te nemen plus een adresverwijzing naar de plaats van het record. Ondanks een grotere waarde van K kan dan toch de omvang van een knoop beperkt blijven. Dit systeem is vergelijkbaar met de index-directe organisatie, alleen de index is hier niet georganiseerd volgens de directe methode maar volgens een B-boom. Dit type B-boom wordt wel aangegeven met B+ boom. Gewone B-bomen worden verder behandeld in paragraaf 7.4.2. en B+ bomen in paragraaf 7.4.3.

7.4.2. Gewone B-bomen

7.4.2.1. Het toevoegen van een element

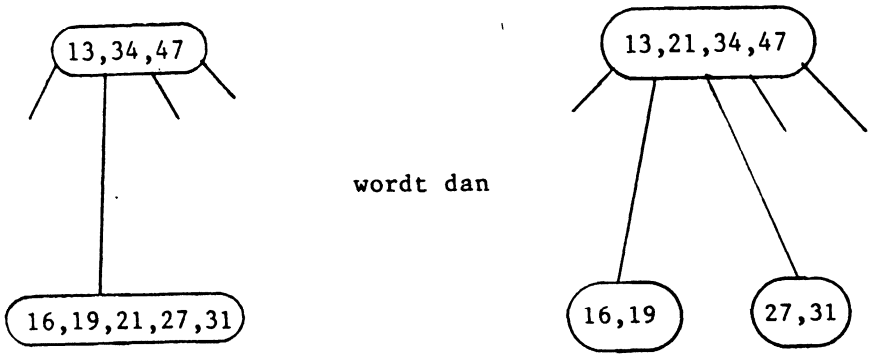
Door het systeem van verwijzingen is het zoeken van een element in een B-boom erg eenvoudig. Ook het toevoegen van een element blijkt betrekkelijk eenvoudig te zijn. In eerste instantie komt een toe te voegen sleutel in een blad terecht. Stel dat aan de B-boom uit het voorbeeld in paragraaf 8.4.1. de sleutel 27 moet worden toegevoegd. Er hoeft dan niets ander te worden gedaan dan

19,21,31 te wijzigen in 19,21,27,31

Moet daarna b.v. sleutel 16 worden toegevoegd, dan zou

16,19,21,27,31

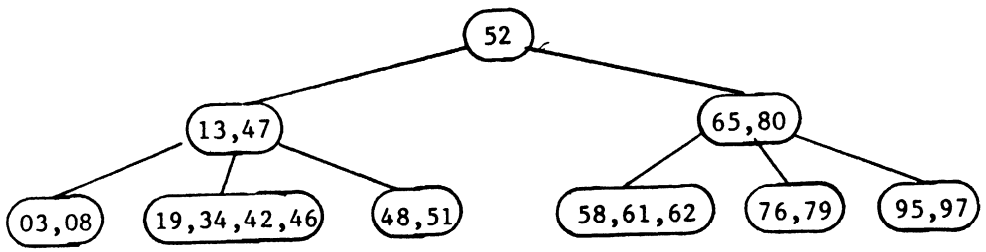
ontstaan. Dit is niet toegestaan omdat het aantal sleutels op deze pagina dan groter dan 2k zou worden. Daarom wordt de middelste sleutel "omhoog" gebracht en de volle pagina gesplitst in twee pagina's.



Is de pagina waarop de middelste sleutel moet worden geplaatst vol, dan wordt hetzelfde proces op deze pagina toegepast. Dit proces kan doorwerken tot de wortel. In dat geval wordt op dezelfde wijze de wortel gesplitst en een nieuwe wortel gecreëerd. De boom groeit dus naar boven.

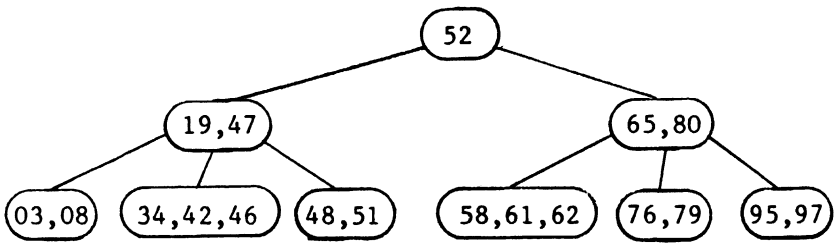
7.4.2.2. Het verwijderen van een element

Bij het verwijderen moet onderscheid worden gemaakt tussen het verwijderen van een element in een blad en in een ander knooppunt. Indien het te verwijderen element zich in een blad bevindt, kan het meestal zonder meer worden weggelaten. Verwijdering van b.v. sleutel 21 uit de boom in 8.4.1. doet (19,21,31) overgaan in (19,31) . Het blad bevat nog 2 sleutels, hetgeen i.v.m. k=2 is toegestaan. Een verwijdering kan echter ook tengevolge hebben dat het aantal sleutels kleiner dan k wordt. Verwijderen we ook sleutel 31 dan zou (19) ontstaan.
In het algemeen kan een dergelijke "underflow" worden weggewerkt door de sleutel(s) van het betreffende knooppunt en van zijn rechter (of linker) broeder te kombineren en deze ongeveer gelijkelijk over beide knooppunten te verdelen. Deze strategie faalt als de broeder al een minimale bezetting heeft. In zo'n geval kunnen de beide knooppunten worden samengevoegd - met een sleutel van de vader - tot één knooppunt. Na verwijdering van 31 wordt de boom dan:



Door het verplaatsen van een sleutel van de vader kan ook daar het aantal sleutels kleiner dan het toegestane aantal worden. Het proces wordt dan op dezelfde wijze voortgezet.

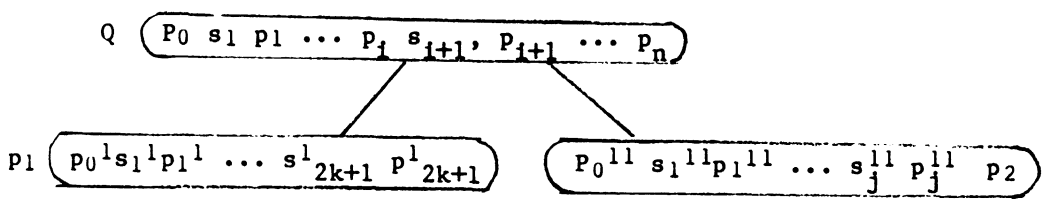
Bekijken we nu het verwijderen van een element uit een knoooppunt dat geen blad is. De te verwijderen sleutel kan dan worden vervangen door de sleutel in de zoon die er in grootte op volgt, terwijl deze laatste sleutel dan kan worden weggelaten.
Wordt b.v. in bovenstaande boom sleutel 13 verwijder, dan ontstaat:



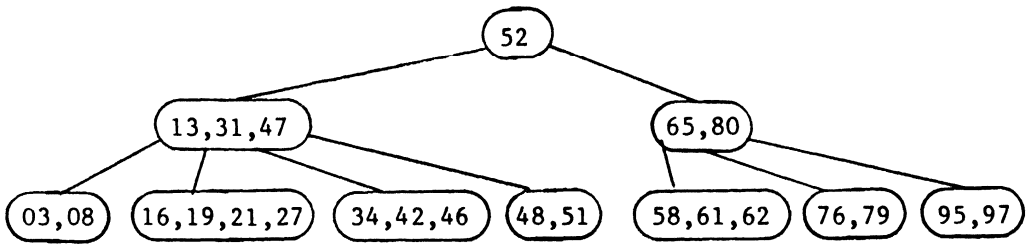
Ook hier kan door het verdwijnen van een sleutel in de zoon underflow ontstaan, waardoor toepassing van het eerder beschreven proces noodzakelijk is.

7.4.3. Varianten

Door het toevoegen en verwijderen van sleutels kan het voorkomen dat een toevoeging een splitsing van een pagina veroorzaakt, terwijl daarna een verwijdering weer een samenvoeging van de pagina's ten gevolge heeft. Om het aantal splitsingen te beperken, kan het algoritme voor het toevoegen van een sleutel als volgt worden gemodificeerd. Indien door het toevoegen in een pagina "overflow" zou ontstaan dan wordt de pagina niet gesplitst, maar wordt eerst gekeken naar de rechter broeder. Is het aantal sleutels van de broeder kleiner dan 2k, dan maakt een eenvoudige herverdeling van de sleutels een splitsing overbodig.
Veronderstel de volgende situatie:



Van de rij sleutels $s_1^1\ s_2^1 \ldots s_{2k+1}^1\ s_{i+1}\ s_1^{11}\ s_2^{11} \ldots s_j^{11}$ wordt dan de "middelste" in Q op de plaats van s_{i+1} gezet, het linkerdeel van de rij in P_1 en het rechterdeel in P_2. Nemen we als voorbeeld de boom uit 3.2.1. en voegen vwe weer sleutel 27 toe en daarna sleutel 16, dan ontstaat de boom



Is het aantal sleutels in de rechterbroeder niet kleiner dan 2k, dus 2k, dan kunnen de twee pagina's samen tot drie pagina's worden gemaakt die elk voor ongeveer 2/3 zijn gevuld.

Indien de volle pagina geen rechter broeder heeft kan natuurlijk ook de linker broeder op analoge wijze in het proces worden betrokken. Het is zelfs mogelijk een volle pagina pas te splitsen als en de rechter en de linkerbroeder vol zijn. Heeft het te splitsen knooppunt helemaal geen broeders, dan moet het de wortel zijn. Zouden we de definitie van de B-boom zo wijzigen dat de wortel maximaal 2k + 2k/3 sleutels mag bevatten, dan ontstaan na splitsing daarvan 2 zonen met elk k + 1/3 k sleutels, dus ieder voor 2/3 gevuld.

Het bovenstaande geeft aanleiding tot een variant op de B-boom. De gemodificeerde B-boom heeft dezelfde structuur als de B-boom, maar heeft andere grenzen voor het aantal sleutels per knooppunt.

- de wortel heeft minimaal 1 en maximaal $2k + \frac{2}{3}k$ sleutels
- de overige knooppunten hebben minimaal $k + \frac{1}{3}k$ en maximaal 2k sleutels.

Het voordeel van dez boom is een betere geheugenbezetting, daar elke pagina voor tenminste 2/3 deel is gevuld. Bovendien verloopt het zoekproces efficiënter, daar de hoogte i.h.a. kleiner zal zijn dan die van de gewone B-boom.

8.4.2.4. Capaciteit van B-bomen

Wanneer een B-boom minimaal gevuld is noemen we het aantal knopen P_{min}. Op gelijke manier is P_{max} het aantal knopen bij maximale vulling van de knopen. Voor P_{min}, P_{max} geldt het volgende overzicht:

nivo	P_{min}	P_{max}
1	1	1
2	2	2k+1
3	2(k+1)	$(2k+1)^2$
4	$2(k+1)^2$	$(2k+1)^3$
.		
.		
h	$2(k+1)^{h-2}$	$(2k+1)^{h-1}$

Voor grotere waarden van k bevinden zich bijna alle records op het nivo van de bladeren. Bijvoorbeeld bij k = 100 en minimale vulling is 99% van alle records toch in de bladeren.
Dit betekent voor k>>1

$$P_{min} \simeq 2k^{h-2} \qquad P_{max} \simeq (2k)^{h-1}$$

Bij minimale vulling bevatten alle bladeren k records en bij maximale vulling 2k records.
Dus $N_{min} \simeq 2k^{h-1}$ $\quad N_{max} \simeq (2k)^h$.

Voor het aantal records n geldt

$$N_{min} < n < N_{max}$$

$$2k^{h-1} < n < (2k)^h.$$

Hieruit volgt $h_{min} < h < h_{max}$

$$\text{met } h_{min} = \left\lceil \frac{\ln n/n}{\ln 2k} \right\rceil \qquad h_{max} = \left\lfloor 1 + \frac{\ln n/2}{\ln k} \right\rfloor$$

In het volgende zijn een aantal waarden van k en n gegeven h_{min} en h_{max}.

Minimum en maximum hoogten

n \ k	100	250	500	1000	2500	5000
10^5	3 3	2 2	2 2	2 2	2 2	2 2
10^6	3 3	3 3	2 3	2 2	2 2	2 2
10^7	4 4	3 3	3 3	3 3	2 2	2 2
10^8	4 4	3 4	3 3	3 3	3 3	2 3
10^9	4 5	4 4	3 4	3 3	3 3	3 3
10^{10}	5 5	4 5	4 4	4 4	3 3	3 3
10^{11}	5 6	5 5	4 4	4 4	3 4	4 3
10^{12}	6 6	5 5	4 5	4 4	4 4	3 4
10^{13}	6 7	5 6	5 5	4 5	4 4	4 4

Uit deze tabel blijkt dat voor grotere waarden van k zelfs bij zeer grote bestanden toch het aantal nivo's en daarmee het aantal toegangen tot achtergrondgeheugen zeer beperkt blijft.

7.4.3. B+ bomen

7.4.3.1. B+ bomen algemeen

Zoals reeds vermeld bevinden zich in de knopen nu alleen de sleutels, adresverwijzingen naar de recordplaatsen en als het geen bladeren zijn ook adresverwijzingen naar zonen. Er zijn nog een aantal variaties met B+ bomen ook mogelijk.

De sleutels in de knopen die geen bladeren zijn hebben primair een functie voor doorverwijzing in de boomstruktuur dus in feite een indexfunctie. We kunnen de sleutels in die knopen ook beperken tot louter die indexfunctie. Dit is het geval indien we alle recordsleutels in de bladeren opnemen. Zouden we alle records zelf in de bladeren opnemen dan vormen die tezamen zelfs een sequentieel bestand. Bevatten de bladeren alleen de sleutels van alle records met adresverwijzingen dan is het daarmee toch eenvoudig het bestand

sequentieel te doorlopen als dit nodig mocht zijn. Zoals gezegd hebben de sleutels in de andere knopen dan alleen een indexfunctie en behoeven zelfs niet echt te bestaan. Voldoende is wanneer de sleutelwaarden in de "index-knopen" (de niet-bladeren) maar scheidend kunnen werken voor de ondergelegen knopen. Een mogelijkheid is ook een index-sleutel gelijk te maken aan de grootste sleutel van de daarbij behorende zoon. Met dat systeem zijn we in feite terug bij de index-sequentiële organisatie. Echter, er is één groot verschil. Bij index-sequentiële organisatie bevatten de indextabellen alleen sleutels en moet de plaats van de tabel op volgend niveau berekend worden uit de indexwaarde. Bij een B+ boom is bij iedere index-sleutel een adres-verwijzing naar een volgende knoop verbonden. Dit betekent dat de knopen op volgend niveau niet meer sequentieel in het geheugen hoeven te staan. En dit laatste maakt het splitsen en recombineren van knopen mogelijk. We lichten dit nog toe aan de hand van een voorbeeld. We nemen daartoe het voorbeeld uit paragraaf 8.3.2. en daarvan de 2e variant na aanvulling met de sleutels 11 474 189 en 11 474 293.

1	9 498 383
2	13 962 481→
3	14 999 917

1	9 558 523
43	11 451 407
44	11 510 446
100	13 962 481

1	11 451 870
35	11 473 824
36	11 474 553→
90	11 510 446
100	---

36→	1	11 473 922
	2	11 474 143
	3	11 474 189
	4	11 474 246
	5	11 474 293
	6	11 474 364
	7	11 474 553

Wanneer nu een record met sleutel 11 473 873 wordt toegevoegd wordt het traject "36" in 2 delen met 4 records gesplitst en in tegenstelling met de indexsequentiële organisatie wordt ook de indextabel op het eerste hogere niveau bijgesteld. Dit levert dan

1	9 498 383
2	13 962 481→
3	14 999 917

1	9 558 523
43	11 451 407
44	11 450 446→
100	13 962 481

1	11 451 870
35	11 473 824
36	11 474 189
37	11 474 553
91	11 510 446
100	

36→	1	11 473 873
	2	11 473 922
	3	11 474 143
	4	11 474 189
	5	
	6	
	7	

37→	1	11 474 246
	2	11 474 293
	3	11 474 364
	4	11 474 553
	5	
	6	
	7	

Hoewel dit voorbeeld strikt genomen geen B+ boom is geeft het wel op juiste wijze het mechanisme weer waarop B+ bomen worden gesplitst en gecombineerd. Door het voorbeeld via sequentiële organisatie uit paragraaf 8.3.2. hier voor te stellen als een B+ boom blijkt duidelijk de nauwe verwantschap tussen boomorganisaties en indexorganisaties.
Voor B+ bomen gelden dezelfde grenzen voor hoogten als gewone B-bomen zoals vermeld in par. 7.4.2.

7.5. Geïnverteerde bestanden

In het voorgaande hebben we een aantal methoden gezien waardoor we individueel exemplaren in een systematisch georganiseerd bestand sneller kunnen benaderen. Al deze methoden hadden een ding gemeen, namelijk dat ieder record eenduidig wordt bepaald door één sleutel. In veel gevallen kan een record echter ook eenduidig worden bepaald door andere velden die tezamen een alternatieve of "duplicaat" sleutel vormen zoals ook reeds aangegeven in paragraaf 1.9.

Nu wordt een bestand georganiseerd op basis van zijn primaire sleutel of dit nu direct of sequentieel of index-sequentieel of welke methode dan ook is. Ten aanzien van iedere andere sleutel bevindt het bestand zich in ieder

geval in een chaotische volgorde. Wil men toch ook op basis van een duplicaat sleutel records benaderen dan zijn er 2 mogelijkheden. De eerste is het vormen van 2 gescheiden bestanden, ieder gebaseerd op één van de 2 sleutels. Dat betekent wel grote redundantie want ieder record komt in feite dan 2 maal voor. De andere methode is één bestand gebaseerd op de ene sleutel, liefst de meest gebruikte. Ten aanzien van de andere sleutel volstaat men met een tabel waarvan ieder element een 2e sleutel+adresverwijzing bevat. Deze tabel zelf kan men weer op iedere passende wijze organiseren dus direct, index-sequentieel, via binaire boom, via B-bomen enz.

Deze methode kan worden toegepast ten aanzien van iedere alternatieve sleutel. Wel dient men er op bedacht te zijn dat mutaties nu ook veranderingen ten gevolge kunnen hebben ten aanzien van deze andere ingangssleutels. Dit alles geldt dus wat betreft meer dan één echte sleutel.

Vaak heeft men ook andere wensen wat betreft navraag in een bestand. Sommige vragen ten aanzien van de inhoud van een bestand hebben niet betrekking op één record maar op een serie records waarvan de inhoud van bepaalde velden aan gegeven criteria voldoet.

Bijvoorbeeld bij een bestand over personen wenst men alle personen die in een bepaalde plaats wonen of in een bepaald jaar geboren zijn of een bepaald beroep hebben. In dat geval wenst men een deelverzameling van het totale bestand. Wanneer een dergelijke vraag eenmalig is bestaat er maar een oplossing namelijk alle records stuk voor stuk ophalen en beoordelen op het gewenste criterium. Komen dergelijke vragen regelmatig voor dan kan het lonen een zogenaamd geïnverteerd bestand aan te leggen. Dit bestand is dan georganiseerd op basis van zo'n criterium, bijvoorbeeld de plaatsnaam. Bij iedere plaatsnaam wordt dan vermeld welke records uit het bestand die naam als plaatsnaam hebben. Een geïnverteerd bestand bestaat dus uit records met een variabel aantal velden. De sleutel is het gewenste criterium en de verdere velden bevatten verwijzingen naar de records in het hoofdbestand die aan het gewenste criterium voldoen. Naast een hoofdbestand kan men dus een geïnverteerd bestand aanleggen ten aanzien van ieder criterium dat men wenst. Het bepalen van gewenste deelverzamelingen van het bestand wordt daarmee uiteraard vereenvoudigd. Daar staat tegenover dat iedere mutatie nu in de regel veel meer werk vereist.

Daarbij moet men bovendien bedacht zijn op het consistent blijven van het gehele systeem. Immers, wanneer een storing optreedt halverwege het aanbrengen in de mutaties dan kan dat betekenen dat in sommige gemuteerde bestanden de verandering al is aangebracht en in anderen niet. In feite vormen geïnverteerde bestanden ook een vorm van redundantie en daarmee een extra probleem voor consistentie van het geheel.

8. VOORBEELD VAN BESTANDSORGANISATIE

8.1. Gegevens bestand en geheugen

We nemen als voorbeeld een bestand van 100.000 records. Ieder record is 500 karakters. In deze 500 karakters bevindt zich een sleutelveld van 30 karakters. Wanneer het nodig is verwijzingsadressen te gebruiken zijn hiervoor 5 karakters vereist.

Het bestand wordt ondergebracht op een schijvengeheugen met de volgende gegevens

404 cilinders per volume
19 sporen per cilinder
13030 karakters per spoor
seektime 75 msec
1 omw 10 msec

We beschouwen en vergelijken een aantal organisatievormen, namelijk directe, index-directe, index-sequentiële in enkele varianten, B-boom, B+ boom en geblokte AVL-boom organisatie.

8.2. Voorbeeld

8.2.1. Directe organisatie

Voor een redelijke toegangstijd kiezen we $\alpha = 0{,}8$. Als bucket ligt voor de hand de grootte van een spoor.

$$\left\lfloor \frac{13030}{500} \right\rfloor = 26 \text{ dus } b=26$$

We kiezen voor overloop via kettingadressen in een apart overloopgebied. Dan wordt

$$C_\alpha = 1{,}0430 \quad C'_\alpha = 1{,}3236 \quad E_b = 0{,}0156 \quad \mu_b = 0{,}1087$$

De verwachte overloop is 100.000*0,0156 = 1560 records. Op een overloopspoor hebben we per record nodig 500 + 5 = 505 karakters.

$$\left\lfloor \frac{13030}{505} \right\rfloor = 25$$

Dus per overloopspoor slechts 25 records. Voor primaire ruimte zijn nodig

$$\left\lceil \frac{100000}{0,8*26} \right\rceil = 4808 \text{ sporen}$$

Voor overloop zijn nodig

$$\left\lceil \frac{1560}{25} \right\rceil = 63 \text{ sporen.}$$

Totaal 4808 + 63 = 4871 sporen. Dit geeft

$$\left\lceil \frac{4871}{19} \right\rceil = 257 \text{ cilinders.}$$

257 cilinders = 257 x 19 = 4883 sporen. Voor overloop dan beschikbaar 4883 - 4808 = 75 sporen. 75 overloopsporen = 75 x 25 = 1875 recordposities. Verwachting is 1560. α=1560/1875 b=1875 geeft $\mu=4,8_{10}-15$ dus verwaarloosbare overschrijdingskans.

Het lezen van 1 spoor kost 75+5+10 = 90 msec. Succesvol zoeken wordt daarmee 94 msec en niet-succesvol zoeken 119 msec.

8.2.2. Index directe organisatie

Sleutel + verwijzingsadres = 30+5 = 35 karakters. Maximaal

$$\left\lfloor \frac{13030}{35} \right\rfloor = 372 \text{ regels per spoor}$$

We nemen 200 regels per spoor waarmee

α = 200/372 en b=372

geeft $\mu_b = 1{,}5_{10}\text{-}28$ dus verwaarloosbaar.

Benodigd

$\left\lceil \frac{100000}{26} \right\rceil$ = 3847 sporen voor de records en

$\left\lceil \frac{100000}{200} \right\rceil$ = 500 sporen voor de index. Totaal 3847+500 = 4347 sporen.

Dit is $\left\lceil \frac{4347}{19} \right\rceil$ = 229 cilinders. Nu altijd C_α=2 en C'_α=1 dus succesvol zoeken 2x90 = 180 msec en niet-succesvol zoeken 1*90 = 90 msec

8.2.3. Index-sequentiële organisatie

Het ligt voor de hand als traject de grootte van een spoor te nemen. Met 26 records/spoor blijven nog 30 karakters over. Deze zijn voldoende voor adresverwijzing in geval van overloop. Voor een indextabel hebben we alleen te maken met een sleutel van 30 karakters.

$$\left\lfloor \frac{13030}{30} \right\rfloor = 434$$

We nemen nu 400 ingangen per tabel. Voor overlooptrajecten geldt

$$\left\lfloor \frac{13030}{(505+5)} \right\rfloor = 25 \text{ overlooprecords/spoor.}$$

We beschouwen alleen overloopruimte op traject en hoofdtrajectniveau. Daarbij moet de ruimte op hoofdtraject niveau zo bemeten zijn dat de kans op overloop ook op dat niveau verwaarloosbaar is, zeg kleiner dan 1 op 10^6 à 10^7.

Om armbeweging te minimaliseren brengen we de trajecttabel binnen de ruimte van het hoofdtraject. Dat wil zeggen de trajecttabel is zelf ook een traject. Per hoofdtraject is een aantal primaire trajecten p en een aantal overlooptrajecten q. Dan moet in dit geval gelden

$$1+p+q = 400$$

De verdeling p en q is niet analytisch te berekenen maar moet eenvoudig door proberen bepaald worden totdat men een verhouding vindt die qua zoektijden en overschrijdingskans voor overloop aanvaardbaar is. Een keuze is verder nog verdeling van overloopruimte over trajecten en hoofdtrajecten.

We nemen 3 varianten, namelijk 0%, 10% en 20% op traject niveau en daarna zoveel overloopruimte op hoofdtraject niveau dat deze aanvaardbaar is.

Eerst 0% trajectniveau. Dit betekent een lading van 26 records per primair traject. Een primair traject als bucket gezien geeft α=1; b=26 dus C_α = 1,3020; C'_α = 3,0277; ε_b = 0,0779 en μ_b = 0,448. We gokken op ongeveer 10% overloopruimte per hoofdtraject dan q=40 en p=359 dus 1+359+40 = 400. De 359 primaire trajecten geven een overloop van 359x26x0,07799 = 728 records. Hiervoor beschikbaar 40x25=1000 dus α=728/1000; b=1000. Dit geeft een $\mu_b = 6{,}1_{10}-22$. Proberen we q=30 dan p=369. Overloop 369x26x0,07799=749 waarvoor beschikbaar 30x25=750. Nu α=749/750 en b=750, geeft een μ_{b+}=0,476. Deze kans is uiteraard veel te groot. We proberen nog p=35 en p=36 en vinden dan het volgende overzicht.

q	μ_b
40	$6{,}1_{10}-22$
36	$2{,}9_{10}-9$
35	$5{,}2_{10}-7$
30	0,5

De waarde van p is dus redelijk kritisch en gezien het feit dat de veronderstelling van Poisson proces op hoofdtrajectniveau niet geheel geldig is kiezen we de veilige kant en nemen q=36. Daarmee 1 hoofdtraject verdeeld in

1 trajecttabel
363 primaire trajecten
36 overlooptrajecten.

Benodigde geheugenruimte berekenen we als volgt. Lading per hoofdtraject 363 x 26=9438 records. Nodig:

$\left\lceil \frac{100000}{9438} \right\rceil$ = 11 hoofdtrajecten. Dit betekent 1 hoofdtrajecttabel ≡1super-traject ≡bestand

11 hoofdtrajecten → 11 x 400 = 4400 sporen →

$\left\lceil \frac{4400}{19} \right\rceil$ = 232 cilinders.

Houden we de hoofdtrajecttabel in het primaire geheugen dan wordt een record dat in zijn traject staat benaderd met 2 toegangen. Door de overloop wordt C_α = 2,3020 en C'_α = 4,0277. In tijden 207 msec en 362 msec.

2e variant 10% overloopruimte op trajectniveau. Nu is 0,9x26 = 23,4. We kiezen voor een lading van 23 records per traject van 26. Daarmee α=23/26 en b=26 geeft C_α = 1,1112; C'_α=1,8037; ε_b=0,0349; μ_b=0,228. Voor een aantal waarden van p en q berekenen we weer de overschrijdingskans op hoofd-trajectniveau. Dat levert:

q	p	μ_b
359	40	$< 10^{-100}$
379	20	$6{,}2\,10^{-25}$
380	19	$1{,}6\,10^{-19}$
381	18	$5{,}8\,10^{-15}$
382	17	$8{,}0\,10^{-11}$
383	16	$2{,}3\,10^{-7}$
384	15	$1{,}2\,10^{-4}$
389	10	0,9999

We kiezen p=17 en q=382.

Nu $\left\lceil \frac{100000}{382*23} \right\rceil$ = 12 hoofdtrajecten

12 x 400 = 4800 sporen → $\left\lceil \frac{4800}{19} \right\rceil$ = 253 cilinders.

Zoektijden C_α = 2,1112 C'_α = 2,8037 en in msec 190 en 252.

3e variant 20% overloopgruimte op trajectniveau. 0,8x26=20,8. We nemen 21 records/traject. α=21/26; b=26; C_α=1,0473; C'_α=1,3550 ε=0,0169 μ=0,117. We proberen weer een aantal p en q waarden.

q	p	μ_b
359	40	$<_{10}$-100
379	20	$<_{10}$-100
389	10	$9{,}3_{10}$-18
390	9	8_{10}-12
391	8	$4{,}7_{10}$-7

We nemen p=9 en q=390. Dan

$\left\lceil \frac{100000}{390*21} \right\rceil$ = 13 hoofdtrajecten.

13 x 400 = 5200 sporen → $\left\lceil \frac{5200}{19} \right\rceil$ = 274 cilinders

Verder C_α=2,0473; C'_α=2,3550 met tijden van 184 msec en 212 msec.

8.2.4. B-boom

We beschouwen eerst de gewone B-boom dus met de records in de knopen. De bepaling van K volgt uit

$2h*500+(2h+1)*5 \leqslant 13030$

$k \leqslant 12,896$

Dus k=12. Dit geeft een $h_{min}=4$ en $h_{max}=5$. Als p het aantal knopen in een B-boom dan geldt voor een minimaal gevulde B-boom

$$n = 1 + (p-1) * k$$

waaruit

$$p = 1 + \frac{n-1}{k}$$

Er zijn dus nodig $1 + \left\lfloor \frac{100000-1}{12} \right\rfloor$ sporen = 8335 sporen.

8335 sporen $\rightarrow \left\lceil \frac{8335}{19} \right\rceil$ = 439 cilinders.

Dit aantal is groter dan het aantal cilinders/volume. Dat zou betekenen dat het bestand over meer dan 1 volume verdeeld zou moeten worden. Deze oplossing laten we buiten beschouwing. We nemen daarom het geval van de B+ boom. Per knoop geldt nu

$2k*30+2k*5 \leqslant 13030$

dus $k \leqslant 186,143$

We kiezen k = 180 daarmee $h_{min}=2$ en $h_{max}=3$. Voor de knopen zijn nodig

$1 + \left\lfloor \frac{100000-1}{180} \right\rfloor$ = 557 sporen. Voor het bestand zelf

$\left\lceil \frac{100000}{26} \right\rceil$ = 3847 sporen.

Samen 3847+557 = 4404 sporen $\rightarrow \left\lceil \frac{4404}{19} \right\rceil$ = 232 cilinders.

Nu geldt C_α = 3 of 4 en C'_α = 2 of 3. In tijden succesvol zoeken 270 of 360 msec, niet-succesvol zoeken 180 of 270 msec.

8.2.5. Vergelijking methoden

De belangrijkste resultaten van de verschillende methoden geven we weer in de volgende tabel

Organisatie	Cα	Cα'	geheugenruimte in cilinders
direct	1,043	1,324	257
index direct	2	1	229
index sequentieel			
1e variant	2,302	4,208	232
2e variant	2,111	2,804	253
3e variant	2,047	2,355	274
B+boom	3 à 4	2 à 3	232
AVL geblokt	3,3	2,3	285

Het is vrijwel niet mogelijk op grond van deze getallen alleen een keuze te maken. Voor die keuze spelen ook andere factoren een rol. Bijvoorbeeld als ook sequentieel verwerkt moet worden verdienen index-sequentieel en B+ boom de voorkeur. Bij sterk verloop van sleutels komt vooral B+ boom en in mindere mate AVL naar boven. Deze twee methoden hebben namelijk een soort automatisch aanpassingsmechanisme.

Als snelheid van toegang primair van belang is dan verdient direct de voorkeur. Verder kan een keuze ook nog sterk beïnvloed worden door de begrenzingen van de beschikbare apparatuur en daarnaast van de ondersteuning van reeds aanwezige softwarepakketten.