TECHNISCHE HOGESCHOOL EINDHOVEN
Afdeling Algemene Wetenschappen
Onderafdeling der Wiskunde

INFORMATICA II

Prof. Dr. R.J. Lunbeck

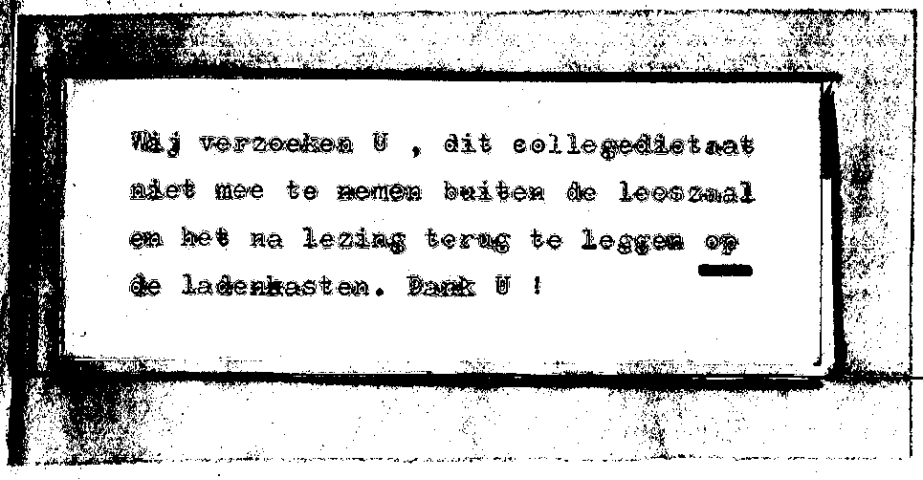
Voorjaarssemester 1974

Technische Hogeschool Eindhoven

Onderafdeling der Wiskunde

Informatica II

Prof. dr. R.J. Lunbeck



Dictaatnr. 2.227 Prijs *f* 4,50

TECHNISCHE HOGESCHOOL EINDHOVEN
Onderafdeling der Wiskunde

INFORMATICA II

Prof.dr. R.J. Lunbeck

Voorjaarssemester 1974

Inhoudsbeschrijving

INFORMATICA II

R.J. Lunbeck

Voorjaarssemester 1974

JdG, 1 Augustus 2005

1.0. INFORMATIESYSTEMEN

1.1. Inleiding

Onder een informatiesysteem zullen wij verstaan het geheel van aktiviteiten, faciliteiten en methoden waarmee een organisatie zijn informatiebehoeften zo goed mogelijk tracht te bevredigen.

Wat bedoelen wij echter precies met deze uitspraak? Volgens een woordenboek is informatie het equivalent van inlichtingen, mededelingen of berichten en als zodanig zullen wij het woord ook blijven gebruiken. Ten onrechte wordt informatie soms als synoniem voor gegevens (data) gebruikt, hetgeen niet juist is omdat (al dan niet numerieke) gegevens pas in een bepaalde context geplaatst, informatie geven. Een temperatuur van 40°C geeft pas in samenhang met patient de informatie dat iemand stevig koorts heeft, doch in samenhang met koffie dat deze drank lauw is. Een gegeven is dus de representatie van een zekere informatie in een aftelbare vorm.

Onder een organisatie zullen wij niet alleen verstaan wat wij menen te begrijpen als we horen van een "organisatie ter bevordering van ...", maar ook een bedrijf of instelling. In het algemeen dus ieder geheel van mensen, tussen wie werkverdelings- en/of hierarchische afspraken bestaan teneinde bepaalde doeleinden, specifiek voor die organisatie, te realiseren.

Aangezien er zoveel soorten organisaties zijn, zullen informatiebehoeften van organisaties sterk uiteenlopen. Zelfs tussen "gelijksoortige" organisaties zullen verschillen optreden in organisatiebehoeftn, aangezien deze behoeften door telkens andere mensen, ieder met eigen opvattingen, worden uitgesproken. Ook binnen een organisatie zullen informatiebehoeften sterk uiteenlopen. De magazijnman zal er in hoofdzaak in geïnteresseerd zijn waar een bepaal artikel ligt en hoeveel exemplaren er nog van zijn. De magazijnmeester wil daarnaast weten wat het verbruik van een bepaald artikel in een bepaalde periode en de besteltijd is, teneinde zijn voorraad op peil te houden. De chef van de magazijnmeester zal in de eerste plaats willen weten hoe vaak een bepaald artikel uitverkocht was en hoeveel werkkapitaal in goederen en opslagruimte vastgelegd is.

Wat faciliteiten zijn, behoeft nauwelijks toelichting; in het algemeen zijn het de financiële, materiële en menselijke hulpmiddelen, nodig om een organisatie draaiend te houden. Hiertoe kunnen rekenautomaten behoren, maar zeker ook andere rekenmiddelen en formulieren waarmee bepaalde gegevens doorgegeven worden.

Bij methoden voor informatievoorziening denken wij bijvoorbeeld aan diverse manuale methoden voor het bewerken van gegevens, methoden van computergebruik, wiskundige methoden in gebruik voor allerlei planningsproblemen maar ook aan de vele naamloze werkmethoden die in de loop der tijd in organisaties gegroeid zijn, enz.

Aktiviteiten zijn werkzaamheden die bij elkaar gegroepeerd kunnen worden omdat ze met elkaar een bepaalde subtaak van een organisatie representeren. Veelal hoort een aktiviteit bij een bepaald organisatie-onderdeel (bijvoorbeeld de produktie-aktiviteit bij de produktie-afdeling) maar niet altijd. Bij oude of grote of door politieke faktoren beheerste organisaties kan het gebeuren dat aktiviteiten en organisatiepatroon niet meer parallel lopen. Dit ontstaat veelal omdat in het verleden een organisatiepatroon wel om bepaalde aktiviteiten en verantwoordelijkheden gebouwd was (zoals het ook hoort en niet aktiviteiten om een organisatiepatroon), doch dat in de loop der tijd aktiviteiten een ander karakter en gewicht hebben gekregen zonder dat het organisatiepatroon (nu veelal om bepaalde mensen) veranderde. Voor de ontwerper van een informatiesysteem is het opsporen van aktiviteiten veel belangrijker dan het volgen van een organisatie-patroon, omdat alleen op die manier werkdoublures en mogelijk inefficient gebruik van mensen en rekenautomaten vermeden kan worden. Dat de producten van een informatiesysteem bepaalde knooppunten in een organisatie-patroon moeten bevredigen is in het geval van een discrepantie tussen aktiviteiten- en organisatiepatroon een komplicerende faktor, waar we echter niet verder op in zullen gaan.

We hebben nu bij de belangrijkste woorden in de eerste zin een kleine toelichting gegeven, doch willen ook kort ingaan op het in velerlei samenhang gebruikte woord "systeem". Meestal wordt een systeem gedefinieerd als "ieder (abstrakt of fysiek) geheel van elkaar afhankelijke delen".
Intuïtief zijn de volgende systemen in overeenstemming met deze definitie:

. een irrigatie systeem,
. een onderwijs systeem,
. een defensie systeem,
. een planeten systeem,
. een regelsysteem,
. een sociaal systeem,
. een biologisch systeem,
. een "club van Rome" systeem,
. enz.

Essentieel voor een systeem is dus dat het, eventueel na enige abstraktie, een geheel moet zijn, dat geïsoleerd van zijn omgeving bestudeerd kan worden èn dat delen ervan met elkaar in wisselwerking treden. In de laatste jaren wordt daarom veel aandacht geschonken aan een vak dat systeemtheorie of systeemwetenschap genoemd wordt. Hierin tracht men de gemeenschappelijke kenmerken van systemen te bestuderen, meestal met het oogmerk om systemen beter te kunnen konstrueren of te beheersen of te begrijpen. De systeemtheorie houdt zich niet bezig met een statische beschrijving van een systeem, doch met het dynamisch gedrag ervan en in het bijzonder met dat van de wisselwerking tussen de delen van een systeem. Vooral het begrip terugkoppeling speelt hierbij een belangrijke rol.

De redenen, dat we hier kort ingegaan zijn op het begrip systeem, zijn om er op te wijzen dat:

- het hier beslist niet gaat over systeemtheorie, doch integendeel de verworvenheden van de systeemtheorie zonder meer geaccepteerd worden. (Wie zich hiervoor interesseert leze boeken en beschriften van o.a. R.L. Ackhoff, J.E. Emery, S. Ramo),
- wanneer we verder over systeem spreken, daar als regel een informatiesysteem mee bedoeld wordt, doch soms ook een hardware systeem of een software systeem. Uit de context moet opgemaakt worden in welke zin we het woord systeem gebruiken,
- gezien het voorgaande een systeemanalist, de man die de specificaties van een informatiesysteem opstelt, beter een informatie-analist genoemd kan worden en een systeemontwerper, de man die deze specificaties nader detailleert naar een operationeel systeem, beter een informatiesysteem-ontwerper.

Informatiebehoeften van een organisatie zullen als regel zonder meer als gegeven beschouwd worden, zodat dit college in hoofdzaak zal gaan over de kennis die men zich eigen moeten maken teneinde, gegeven de informatiebehoeften van een organisatie, een voor automatische gegevensverwerking geschikt systeem te ontwerpen. Bij dit systeemontwerp zullen problemen gesteld door de beschikbare tijd en kosten het accent bepalen, terwijl meer direct met de aard der informatie samenhangende aspecten van levering van slechts relevante, doch alle op een vraag betrekking hebbende informatie nauwelijk aan de orde zullen komen. Deze zouden namelijk leiden in de richting van de in het bibliotheek- en documentatiewezen bestudeerde problemen van classificatie, trefwoorden, thesaurusopbouw, relevantie, documentanalyse, enz.

Het maken van goede informatiesystemen is tijdrovend, niet in de eerste plaats door de "technische" arbeid die men moet verrichten voor de verschillende fasen van het werk (waar we later op in zullen gaan), doch vooral omdat we zowel tijdens ontwerp en ontwikkeling van het systeem als voor de eindprodukten te maken krijgen met een gecompliceerde factor: de mens. Soms is deze weinig bereid om mee te werken, bijv. uit vrees

- overbodig te worden,
- in aanzien te dalen omdat hij minder "onderhorig" personeel zal hebben na een geslaagde automatisering, of
- omdat bepaalde operationele beslissingen automatisch door een machine zullen worden voorbereid (bijv. een magazijnchef die vroeger een eigenhandig gemaakte notitie moest sturen naar de inkoopafdeling met het verzoek om een bepaald artikel te bestellen, zal met een informatiesysteem dat automatisch een bestelbrief produceert, deze brief alleen nog maar hoeven te paraferen),
- het gedetailleerde "direkte kontakt" met gegevens te verliezen omdat ze in onzichtbare (magnetische) vorm in een machine opgeslagen zijn (dit is vooral voor accountants vaak een probleem).

Op de sociale en psychologische problemen van het maken van informatiesystemen zullen we echter niet ingaan.

1.2. Ontwikkeling en voorbeelden van informatiesystemen

Informatiesystemen zijn (uiteraard net als organisaties) al eeuwen oud, doch wij zullen alleen informatiesystemen beschouwen zoals die zich in de laatste 25 jaar na inschakeling van een rekenautomaat hebben ontwikkeld.

Tot ongeveer 1965 waren de meeste systemen niet veel anders dan het geautomatiseerde equivalent van het oude handwerk, zoals dat in hierarchisch gescheiden delen van een organisatie uitgevoerd werd. M.a.w. slechts het schrijfvermogen van een mens werd vervangen door het mechanisme van een regeldrukker en het met de hand sorteren en accumuleren van gegevens werd door de rekenautomaat overgenomen. De snelle regeldrukker werkte aanvankelijk echter in de hand dat men steeds meer detail- en overzichtsstaten ging maken, wellicht zelfs om overbodig geworden personeel met het lezen van staten aan het werk te houden. Daar bovendien een numerieke verschuiving van personeel van een administratieve afdeling naar een rekencentrum optrad en de machinekosten hoog waren, was ondanks de snellere gegevensverwerking het economische nut van automatisering soms uiterst twijfelachtig.

Na 1965 kwam hier in verschillende opzichten verandering in:

. onderdelen van een administratie of gescheiden administraties werden met elkaar "geïntegreerd" in die zin dat eindresultaten van een bepaalde berekening begindata waren voor een volgende berekening (voorheen werden begindata steeds opnieuw manuaal opgesteld door de "verantwoordelijke" sectie en ingevoerd). In bijgaande overdruk, ontleend aan een boek van Eeuwe en Albarda, is zo'n integratie af te lezen,
. men streefde er naar om de hoeveelheid output te beperken, vooral door alleen die gegevens af te drukken die aanleiding zouden kunnn geven tot bepaalde handelingen of beslissingen. Andere gegevens die bijvoorbeeld binnen een vooraf gestelde marge blijven, worden alleen op uitdrukkelijk verzoek afgedrukt (bij technisch-wetenschappelijk werk waar men zelden geïnteresseerd is in tussenresultaten, was altijd gebruikelijk dat alleen eindgegevens afgedrukt werden),
. wiskundige technieken vonden hun weg in allerlei planningsmethoden (statistische en besliskundige optimalisatiemethoden voor lagere transportkosten, lagere voorraadniveau's, betere machinebezetting, betere produktieplanning, enz.),
. men beperkte zich niet tot het verwerken van historische data, maar voerde allerlei berekeningen uit ter ondersteuning van het op de toekomst gerichte beleid.

Deze ontwikkeling gaat nog steeds voort en met name wordt verwacht dat simulatiemethoden en later artificial intelligence hoe langer hoe meer zullen worden toegepast (investeringsselektie, kopersgedrag, enz.), zoals in onderstaand schema aangegeven.

opstellen doelstellingen van organisatie	toekomst
uitwerking van alternatieve ontwikkelingen	
selektie uit deze alternatieven	
operationeel doorrekenen	sinds 1965
uitvoering van operationeel plan	al voor 1965
na-kontrole op uitvoering	

De tendens is hierbij om gegevens centraal op te slaan en te verwerken, doch beslissingsplaatsen te decentraliseren, hetgeen dankzij de ontwikkelingen op het gebied van data-communicatie steeds beter mogelijk wordt.

EEN VOORBEELD VAN TOTALE INTEGRATIE IN EEN FABRIEK

De verschillende procedures, die in dit voorbeeld van een information system zijn samengevoegd, zijn de volgende:

- omzetverwachtingen
- dagelijkse produktieplanning en werkvoorbereiding
- inkoop en ontvangst van ruwe grondstoffen
- crediteurenadministratie
- voorraad- en magazijnbeheer
- loonadministratie
- facturering en debiteuren-administratie

De betekenis van de afkortingen is als volgt:

T = tape
TW = typewriter (Flexowriter)
C = computerrun
C-T = card to tape operator
P = print out

R = een door de computer geproduceerde tape (tijdens een vorige run)

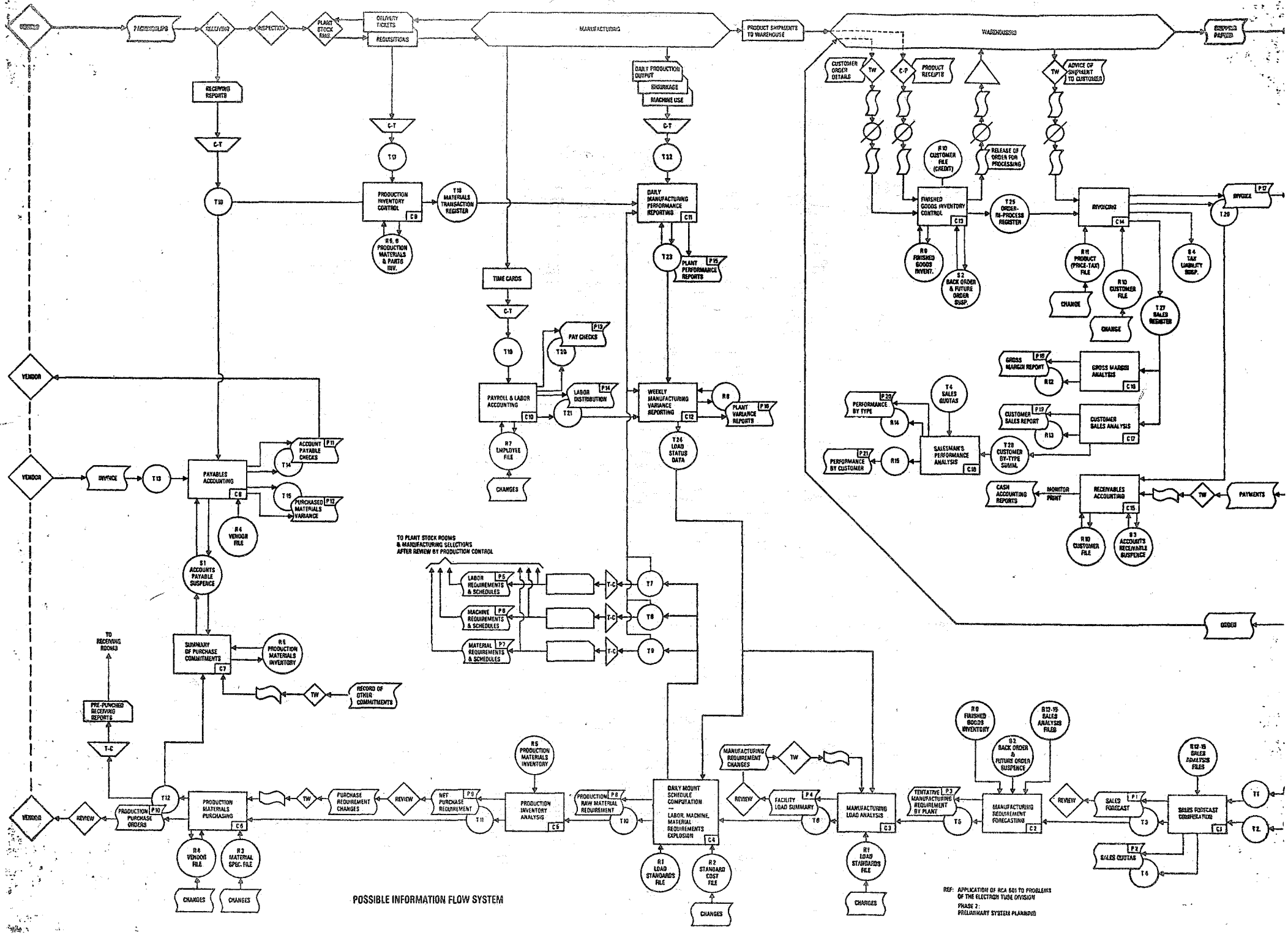
S = tape met nog niet afgewikkelde posten

T1 = marktonderzoek
T2 = marktfactoren
T3 = verwachte omzet
T4 = omzet naar verschillende kenmerken gesplitst
T5 = voorlopig benodigde hoeveelheden eindprodukt
T6 = produktieoverzicht per soort eindprodukt
T7 = uitsplitsing in arbeidsuren
T8 = uitsplitsing in machineuren
T9 = uitsplitsing in materiaal
T10 = benodigde ruwe grondstoffen
T11 = in te kopen ruwe grondstoffen
T12 = bestelde ruwe grondstoffen
T13 = ontvangen facturen
T14 = afgegeven cheques
T15 = afwijkingen bestelde hoeveelheden c.q. prijzen
T16 = ontvangen goederen van leveranciers
T17 = afgiften van magazijn naar fabriek
T18 = grondstoffenverbruik
T19 = aantallen gewerkte uren
T20 = uitbetaalde lonen
T21 = loonkostenverdeling
T22 = dagelijks produktierapport (aantallen gereed produkt, uitval etc.)
T23 = produktieoverzicht t.a.v. grondstoffen, mach.- en manuren)
T24 = nacalculatie; werkelijke bezetting
T25 = verkooporders die in behandeling worden genomen
T26 = debiteurenoverzicht
T27 = verkoopoverzicht
T28 = verkoopoverzicht per klantengroep/ vertegenwoordiger e.d.

R1 = beschikbare capaciteiten (standaard)
R2 = standaardkosten, gespecificeerd naar kostensoort
R3 = standaardgegevens benodigde ruwe grondstof
R4 = crediteuren
R5 = grondstoffenvoorraad
R6 = onderdelenvoorraad
R7 = stamgegevens werknemers
R8 = afwijkingen van voorcalculatie
R9 = voorraad gereed produkt
R10 = stamgegevens debiteuren met kredietgrenzen
R11 = gegevens over prijzen, b.t.w. en omschrijvingen gereed produkt
R12 = brutowinst-analyse
R13 = verkoopstatistiek per afnemer
R14 = omzetanalyse per soort produkt
R15 = omzetanalyse per soort afnemer

S1 = nog te betalen crediteurenposten
S2 = orders in portefeuille
S3 = nog te ontvangen debiteurenposten
S4 = te betalen belasting

Voorbeeld van totale integratie in een fabriek



POSSIBLE INFORMATION FLOW SYSTEM

REF: APPLICATION OF RCA 601 TO PROBLEMS OF THE ELECTRON TUBE DIVISION
PHASE 2:
PRELIMINARY SYSTEM PLANNING

Het spreekt vanzelf dat men tot dusver ontwikkelde systemen op verschillende manieren kan karakteriseren. Een indeling is:

- historische systemen: deze komen neer op het produceren van (meestal zeer uitgebreide) overzichten van werkzaamheden in een bepaalde periode (veelal een maand), met in de meer verfijnde systemen bovendien cumulatieve gegevens vanaf bijvoorbeeld het begin van een boekjaar en een vergelijking met begrotingsbedragen. Historische (jaar)overzichten zijn soms verplicht (belastingsdiensten, bureau voor statistiek, bedrijfsverenigingen, kamers van koophandel) en moeten dan geproduceerd worden al hebben zij voor het toekomstig beleid vaak geen betekenis. Overzichten worden meestal gehanteerd door de (lagere) operationele organisatieleiding en voor eenvoudige accountantscontroles, maar zijn vrij snel voer voor archief of prullemand.
- besturingssystemen: deze zijn in de eerste plaats gericht op het produceren van gegevens ten behoeve van het nemen van tactische beslissingen (met de beschikbare geld-, mankracht- en produktiefaciliteiten direkt te realiseren). Het opzetten van deze systemen vraagt meestal een uitvoerige O.R. studie om de goede beslissingscriteria te vinden.
- information retrieval systemen: in de eerste plaats gericht op het snel geven van antwoorden op vooraf vastgestelde typen vragen. Het invoeren of wijzigen van gegevens kan soms in langzamer tempo gebeuren. Tot deze categorie horen reserveringssystemen (voor vliegtuigen, hotels, schouwburgen, enz.) en signaleringssystemen van bijv. de dienst van kentekenbewijzen.
- management information systemen: het is niet duidelijk of dit een fraaie verkoopskreet is voor historische systemen dan wel of het alsnog te ontwikkelen systemen betreft die gegevens leveren voor het nemen van strategische beslissingen. Zolang we echter niet in staat zijn om algemeen geldige algorithmen op te stellen voor het nemen van strategische beslissingen (ontwikkelings- en investeringsplannen) is er nog niet veel hoop voor echte systemen van dit type.

Informatiesystemen zijn ook op een andere manier te karakteriseren, nl. door de wijze waarop gegevens aan het systeem toegevoerd en rapporten geproduceerd worden. We onderscheiden dan:

- partijgewijze (batchwise) verwerking, waarbij gegevens een bepaalde periode opgespaard worden en dan samen verwerkt met veelal tegelijkertijd het produceren van rapporten. Een voordeel van deze methode is de betrekkelijk grote efficiency en eenvoud, een nadeel dat tussen twee verwerkingstijdstippen de gegevens in de machine niet meer volkomen de realiteit weergeven.

van de afstudeerdocent. Tevens beoogt dit tentamen de uniformiteit te bevorderen in niveau, zwaarte, omvang en beoordeling van de afstudeerprojekten, die binnen de Onderafdeling worden uitgevoerd.
Voor het vaststellen van plaats en tijd van dit tentamen overlegge men met de afstudeerdocent.

2.0. INFORMATIESTRUCTUREN EN INFORMATIEDRAGERS

2.1. Inleiding

In informatiesystemen moeten grote hoeveelheden gegevens opgeslagen, verwerkt en opgezocht worden. We zullen hier onder ogen zien hoe deze opslag van gegevens kan geschieden, in aanmerking nemend welk gebruik men van de gegevens wil maken en welke opslagmogelijkheden er heden ten dage zijn. Onderscheid moet worden gemaakt tussen de logische of gegevensstructuur, die de samenhang of structuur van een gegevensverzameling beschrijft en zodoende weerspiegelt wordt in het, in welke taal dan ook geschreven, programma dat met deze verzameling werkt, en de physieke of opslagstructuur, waarin vastgelegd is hoe een bepaalde logische structuur physiek afgebeeld is op de beschikbare geheugensystemen. De afbeelding van logische op physieke structuur en de toegang tot de laatste is tegenwoordig veelal een onderdeel van een bedrijfssysteem, nl. het z.g. "data-management" systeem. Het nagestreefde ideaal is hierbij dat een programma niet herschreven hoeft te worden wanneer een andere physieke structuur gekozen wordt.

In talen als Fortran en Cobol zijn bijzonderheden over een bepaalde physieke structuur echter terug te vinden in een programma in die taal, al zijn bepaalde aanwijzingen ten aanzien van de physieke structuur vervat in een stuk "karwei-besturing" dat zo'n programma vergezelt. Veranderingen in de logische structuur betekenen wijzigingen in de karwei-besturing. Algol 60 laat zich helemaal niet uit over het gebruik van geheugensystemen, zodat iedere installatie zijn eigen conventies dienaangaande heeft. In enkele recente, deels nog experimentele informatieverwerkingstalen (te noemen zijn o.a. Mark IV, Infol, TDMS) wordt informatie betreffende de physieke structuur van een informatieverzameling slechts bij die verzameling opgeborgen, zodat deze volledig ontbreekt in een programma, waarmee zodoende herprogrammering niet nodig is als men de physieke informatiestructuur wil veranderen.

In het volgende zal eerst ingegaan worden op enkele basisbegrippen en definities, daarna volgt een bespreking van de eigenschappen van de huidige typen informatiedragers. Tot slot volgt dan de behandeling van enkele logische en physieke informatiestructuren, daarbij in het bijzonder aandacht schenkend aan de problemen van toevoegen, verwijderen, zoeken en veranderen van informatie in een bestaande of te maken informatieverzameling.

2.2. Enkele basisbegrippen

2.2.1. Karakters

De kleinste logische bouwstenen voor informatie-vastlegging zullen voor ons karakters zijn, d.w.z. de letters van een alfabet, de cijfersymbolen 0-9 en een hier niet nader bepaald aantal "speciale" karakters als leestekens, rekenkundige symbolen, enz. De afbeelding van deze bouwstenen op de kleinste physieke bouwstenen, t.w. bits zal niet aan de orde komen, behoudens de vaststelling dat voor de afbeelding van ieder karakter een vast aantal bits (meestal 6 of 8 bits, samen byte te noemen) nodig is.

2.2.2. Items

Met deze bouwstenen kunnen we de voor een probleem relevante, logische informatie-eenheden of items opbouwen en wel de naam van een item en de waarde van een item ("value"). Afhankelijk van de aard van het item kan de waarde numeriek (een getal) zijn, of alfabetisch (een naam of woonplaats) of alfanumeriek (een codewoord samengesteld uit letters en cijfers). De naam van een item (attribuut, "property") wordt veelal niet expliciet vastgelegd en, indien dit wel het geval is, meestal met letters alleen. Het vastleggen van itemwaarden alleen (een "homogene" informatie verzameling) impliceert een stilzwijgende afspraak over de volgorde van items en vereist de aanwezigheid van een speciale "nulwaarde" (blank) om aan te kunnen geven dat een bepaald gegeven niet bekend is. Voor de physieke vastlegging van een item in een veld (rubriek, "field") kan enige conversie van de externe naar de interne representatie plaatsvinden; zo worden decimale getallen veelal binair in de machine vastgelegd, de nog gebruikelijke datumschrijfwijze wordt geconverteerd naar de voor vergelijking handiger (en sinds kort genormaliseerde) representatie jaar-maand-dag, een volledige tekst wordt misschien gecomprimeerd door invoering van afkortingen of weglating van klinkers, enz. Deze conversie-problemen zullen hier echter niet besproken worden. De lengte van een item is het aantal karakters, waaruit zijn physieke vastlegging bestaat.

2.2.3. Records

De een bepaald verband hebbende items worden als regel verzameld tot een z.g. (logical) record, bijvoorbeeld omdat het gegevens zijn die betrekking hebben op eenzelfde persoon of eenzelfde apparaat of eenzelfde complex van activiteiten ("entity"). Is de naam van de entity expliciet in het record op-

genomen, dan heet dit centrale item (ook wel object genoemd) meestal sleutelitem; de waarde van het sleutelitem heet de sleutel ("key") omdat dit gegeven als het ware de toegang geeft tot andere bij de entity horende gegevens. Soms zijn meer sleutels nodig om een record éénduidig te bepalen (vgl. het doel van naam, voorletters, geboortedatum).

Afhankelijk van het resultaat van de bestudering van een probleem kan men besluiten tot het ene uiterste, nl. om alle items behorende bij een sleutelitem in één record te plaatsen (dat dan wel erg groot kan worden) of toe het andere uiterste, nl. een groot aantal records slechts bestaande uit het sleutelitem en telkens één ander item, of tot een oplossing tussen deze twee uitersten. Een bepaald gegeven bij een zeker item zal dan meestal via de sleutel in één van de records gezocht moeten worden. Het is echter ook mogelijk dat het gevonden moet worden via een verwijzing (wijzer, "pointer, reference") in één van deze records (vergelijk de situatie in onze telefoongidsen, waarin we Bouw- en Woningtoezicht vinden via Gemeentelijke Instellingen).

De physieke vastlegging van logical records gebeurt door "opblokken" van een aantal logical records tot een blok (physical record). Dit in verband met de efficiëntie van het gebruik van secundaire geheugensystemen, zoals we verderop zullen zien. Blokken worden met één invoer- of uitvoeropdracht gelezen of geschreven (d.i. tussen secundair en primair geheugen gecopieerd); logical records moeten dan verder uit een blok geïsoleerd worden voor gegevensverwerking. In deze zin zijn kaarten uit een kaartsysteem te vergelijken met physical records, daar ze met één beweging gelicht worden.

De vastlegging van items in een logical record kan nog op verschillende manieren gebeuren:

- de "fixed" methode:
  - itemwaarden (eventueel spaties, indien niet gedefinieerd of bekend) worden in een vaste volgorde achter elkaar geplaatst,
  - er is een vast aantal items; iedere itemwaarde heeft zijn eigen, constante lengte (eventueel via opvullen met spaties),
  - het begin van iedere itemwaarde is zodoende een vast aantal posities na het begin van een logical record.

Dit is wel de meest gebruikte methode door zijn overzichtelijkheid en eenvoudige hanteerbaarheid. De recordlengte is eveneens fixed en uiteraard gelijk aan de som van de itemlengten.

- de indexed methode:
  - itemwaarden worden in een vaste volgorde achter elkaar geplaatst,
  - er is een vast aantal items, maar itemwaarden hebben een variabele lengte,
  - het begin van iedere itemwaarde wordt vastgelegd in wijzers, die in het begin van een logical record geplaatst worden.

Deze methode is geschikt voor problemen waarbij veel gegevens van wisselende lengte voorkomen, zoals het geval is bij namen en adressen. (De rekenautomaat moet "indirect addressing" of "indexing" faciliteiten bezitten voor een efficiënte programmering van deze methode.)

- de "separator" methode:
  - itemwaarden worden in een vaste volgorde achter elkaar geplaatst,
  - zij worden van elkaar gescheiden door separators.

Deze methode heeft het bezwaar dat het opzoeken van een bepaald gegeven het karakter voor karakter afzoeken van het record vereist, hetgeen een tamelijk complex programma vereist indien er geen speciale hardware instructies zijn.

- de "label" methode:
  - paren itemnaam-itemwaarde worden, eventueel in een willekeurige volgorde, achter elkaar geplaatst,
  - deze paren worden van elkaar gescheiden door separators.

Deze methode is een uitbreiding van de vorige in die zin, dat noch het aantal itemwaarden constant hoeft te zijn, noch de volgorde. Het opzoeken van een bepaald gegeven is echter nog eens zo ingewikkeld.

De twee laatste methoden worden niet veel gebruikt. Mengvormen van deze vier methoden, vooral van de eerste twee, komen ook voor.

Opgave

Vergelijk deze methoden ten aanzien van facetten als geheugengebruik, toegangstijd, programmering, opblokken. Probeer een beeld te vormen van de programmeringsproblemen die op zullen treden bij het gebruik van deze methoden.

2.2.4. Bestanden

Een geordende verzameling logical records, die dezelfde structuur (aard en vastlegging van items) bezitten, heet een bestand (file; tegenwoordig wordt helaas ook het woord dataset gebruikt, dat vroeger alleen gebruikt werd voor

modem, een apparaat voor de connectie van computerapparatuur aan een telefoonlijn). De ordening kan het gevolg zijn van de wijze waarop het bestand oorspronkelijk opgebouwd wordt, maar ook van het sorteren van logical records op grond van de daarin voorkomende sleutel(s). Een bestandsgrootte d.w.z. het aantal logical records, kan als regel slechts bij benadering opgegeven worden. De interne bestandsnaam is de naam waarmee een bestand in een programma benoemd wordt.

De physieke vastlegging geschiedt meestal op een bepaald type informatiedrager (volume), bijvoorbeeld een magneetband (tape-reel) of schijfeenheid (diskpack). Wanneer op zo'n informatiedrager meerdere bestanden worden vastgelegd (uiteraard duidelijk van elkaar gescheiden), dan spreekt men van een multifile volume; vereist omgekeerd een bestand meerdere volumes, dan spreekt men van een multivolume file. Aan een informatiedrager wordt met behulp van een etiket veelal een externe bestandsnaam toegekend voor de menselijke identificatie.

Een verzameling bestanden wordt wel "databank" genoemd, al wordt deze term ook wel gereserveerd voor databanken, die op één volume opgeslagen zijn en waarin ieder item slechts één keer voorkomt.

Een bestandsorganisatie wordt gekenmerkt door de ordening van logical records in een bestand en door de afbeelding van de logical records op de physieke informatiedragers. De keus van een bepaalde bestandsorganisatie wordt bepaald door specifieke probleemeisen t.a.v. toegankelijkheid voor en kosten van het aanbrengen van wijzigingen en aanvullingen, voor het terugvinden van informatie, door beveiligingsaspecten en door de beschikbare apparatuur.

Een permanent bestand (master file) is een bestand waarin wel wijzigingen kunnen optreden, maar dat toch grotendeels blijft bestaan gedurende een tijd die lang is vergeleken met de tijd tussen twee bewerkingen met het bestand.

Een mutatie-bestand (transaction file) leeft als regel slechts totdat de gegevens ervan verwerkt zijn in één of meer permanente bestanden of op papier zijn weergegeven. Soms worden gelijksoortige mutatiebestanden echter verenigd tot een historisch mutatiebestand, dat dan voor "naslag" enige jaren of maanden bewaard wordt.

Soms wordt een permanent bestand opgesplitst in een prime file en enige trailer files die met wijzers vanuit de master file bereikt worden.

Soms wordt tussen een aantal masterfiles een connectie tot stand gebracht met behulp van wijzers die in een reference file zijn opgenomen.

Bestanden worden onderworpen aan bewerkingen, die als volgt te rubriceren zijn:

- toevoegen (inserting) van records aan een bestand (het opbouwen van een nieuw bestand moet bij voorkeur als bijzonder geval hiervan gezien worden),
- veranderen (updating) van (bepaalde delen van) sommige records in een bestand,
- verwijderen (deleting) van bepaalde records in een bestand,
- opzoeken van gegevens in een bestand op grond van hetzij kennis van de sleutel (we zullen dit eenvoudige proces "searching" noemen) of op grond van de waarden van één of meer items ("retrieval" te noemen; searching is dan retrieval op grond van het sleutel-item),
- het sorteren van de records van een bestand.

In samenhand met deze bewerkingen wordt de term file-activity gebruikt, d.i. het percentage logical records in een bestand dat tijdens het gebruik van een bestand betrokken is bij één of meer van deze bewerkingen. De term file-volatility wordt gebruikt voor het percentage logical records in een bestand dat tijdens het gebruik betrokken is bij updating alleen. File-turnover is tenslotte het percentage records dat toegevoegd en/of verwijderd wordt.

De voor ieder van deze bewerkingen benodigde tijd wordt bepaald door de grootte van het bestand en door de bestandsorganisatie. Wat de "beste" bestandsorganisatie is, hangt dan ook af van het soort bewerkingen waaraan een bestand het meest onderworpen wordt en van veiligheidseisen die gesteld kunnen worden.

Opgave

Onderzoek wat gezegd kan worden over activity en volatility van twee bestanden, die ontstaan door splitsing van een gegeven bestand op grond van hetzij sleutels, hetzij logical records.

2.2.5. Structuur van en toegang ("access") tot records in een bestand

We moeten onderscheid maken tussen de structuur van een bestand, d.w.z. de opbouw en plaats van records in een bestand, en de toegang tot een bestand, d.w.z. de wijze waarop we aan een bepaald record komen.

In een ongestructureerd ("sèrial") bestand is er geen enkele relatie tussen de records van een bestand; zij staan bijvoorbeeld in de volgorde waarin de records toevallig aan het bestand toegevoegd zijn. Als regel zijn bestanden echter gestructureerd; men onderscheidt:

- sequentiële structuren, wanneer de volgorde der records op grond van een of ander kenmerk tot stand is gekomen, als regel op grond van de waarde van de sleutel (oplopend of soms afnemend) en soms op grond van de activiteit (bewerkingsfrequentie) van de records. Er is geen relatie tussen recordsleutel en het (absolute) adres van het record op een informatiedrager,
- willekeurig toegankelijke ("random") structuren, wanneer records zodanig op een informatiedrager geplaatst worden dat er wel een verband is tussen recordsleutel en (absoluut) adres van het record op een informatiedrager,
- lijststructuren (sliertstructuur, "list structure"), wanneer records die logisch bij elkaar horen (bijvoorbeeld bij dezelfde sleutel) met elkaar verbonden zijn met behulp van wijzers ("pointers"), die in tegenstelling tot de vorige twee structuren een uitbreiding van ieder record met die wijzer vereisen. Bijzondere gevallen van lijststructuren zijn weer:
  - enkelvoudige kettingstructuren, wanneer in ieder (behalve het laatste) record een wijzer opgenomen is naar het volgende op de een of andere wijze logisch samenhangende record,
  - dubbele kettingstructuren, wanneer in ieder record (behalve laatste en eerste) een wijzer staat naar het volgende en een wijzer naar het vorige record,
  - ringstructuren, wanneer het laatste record in een kettingstructuur een wijzer naar het eerste bevat en het eerste (eventueel) een wijzer naar het laatste record (het "eerste" record moet dan door een bijzonder kenmerk als zodanig herkenbaar zijn),
  - boomstructuren, wanneer naar ieder record, behalve het eerste, door slechts één ander record verwezen wordt,
  - netwerkstructuren, wanneer naar ieder record door één of meer andere records verwezen wordt.

In onderstaande plaatjes zijn deze structuren geschetst (opgemerkt moet worden dat de terminologie in de literatuur uiteenloopt!)
(Gestippelde pijl = toegang, getrokken pijl = wijzer.)

sequentiële structuur : ---→o---→o---→o---→o---→o---→o---→o

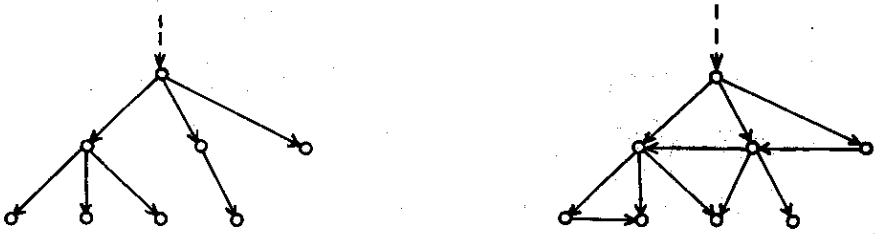
willekeurig toegankelijke structuur : o o o o o o o

lijststructuren :

---→o—→o—→o—→o—→o . . . →• enkelvoudige ketting

---→o←→o←→o←→o←→o . . . →• dubbele ketting

---→•—→o—→o—→o—→o . . . →o ringstructuur



boomstructuur netwerkstructuur

Ten aanzien van de toegang tot een bestand is men tot op zekere hoogte afhankelijk van de structuur van het bestand (en zoals we later zullen zien van de aard van de informatiedrager). Bij ongestructureerde bestanden zit er voor het zoeken van een record niet veel anders op dan alle records achter elkaar te bekijken totdat men de gezochte records gevonden heeft (dit proces heet "lineair" zoeken).

Bij sequentieel gestructureerde bestanden kan men lineair zoeken tot de gewenste records gevonden zijn of de plaats waar ze hadden moeten staan gepasseerd is, hetgeen door vergelijking van sleutels te bewerkstelligen is. In principe zijn ook andere methoden mogelijk (of de informatie-drager daarvoor geschikt is, zullen we later bekijken), bijvoorbeeld "lineair" zoeken, waarbij we ieder interval waarin de gezochte records kunnen liggen (oorspronkelijk het hele bestand) steeds verder in twee stukken (niet noodzakelijk even groot) verdelen totdat de gezochte records al dan niet gevonden zijn. Nog een andere methode heet skip-sequential zoeken, hierbij gaat men steeds sleutels vergelijkend eerst met grote "sprongen" door het

bestand; wordt een passeren van de mogelijke plaats van de gezochte records geconstateerd dan gaat men met wat kleinere sprongen vanaf het beginpunt van de laatste sprong zoeken, enz.

Bij random gestructureerde bestanden heeft men per definitie toegang tot gezochte records zonder andere records te inspecteren. Dit kan gebeuren met een "directe" zoek wanneer het adres van de gezochte records direct volgt uit de sleutel of met een "indirecte" zoek, wanneer uit de sleutel via een of andere algorithme eerst nog het adres berekend moet worden. We komen hier later nog op terug.

Bij lijst-gestructureerde bestanden zal men bij het zoeken van een record voor het volgen van de wijzers weer "voorgaande" records moeten inspecteren (tenzij de relatie tussen sleutel en recordadres bovendien nog via een aparte indexlijst op te sporen is). Boomstructuren worden zo opgezet dat een zo klein mogelijke inspectiereeks nodig is.

Na de bespreking van informatiedragers zullen we aan de hand van enkele in de praktijk veel gebruikte bestandsstructuurvormen zien dat bepaalde combinaties van structuren nuttig zijn.

2.3. Eigenschappen van informatiedragers

2.3.1. Magneetband

Op een magneetband worden de karakters van een blok achter elkaar met ("frames" van) 8 informatie bits (vroeger ook met 6 bits) per karakter geschreven (zg. controlebits om een goede hardware functie te onderzoeken, laten we buiten beschouwing).

De dichtheid van de beschrijving is als regel tegenwoordig 800 of 1600 karakters per inch. Tussen twee blokken bevindt zich een "onbeschreven" gedeelte, de interblok gap (ook wel minder juist interrecord gap genoemd) van ongeveer 1,5 cm, die wel enige "ruis" bevat. De lengte van deze interblok gap is niet exact op te geven, omdat het technisch niet mogelijk is om het schrijven van een blok precies op een bepaalde plaats te doen beginnen. Bovendien is het mogelijk, bijvoorbeeld nadat enige keren vergeefs geprobeerd is een blok correct op de band te schrijven (technisch kan dit onderkend worden), de interblok gap programmatisch te vergroten, zodat dan het schrijven van een blok gebeurt voorbij de plaats die kennelijk vuil of beschadigd was. Het moet daarom als een ernstige fout aangemerkt worden als men bij banden probeert om een gelezen blok na mutatie weer op de oude plaats terug te schrijven. Het bijwerken van een bandbe-

stand impliceert zodoende ook altijd een herschrijven op een nieuwe band! (De oude band kan zo nodig nog voor bestandsreconstructie doeleinden bewaard worden.)
Dit heeft aanleiding gegeven tot het zg. grootvader - vader - zoon systeem, waarbij de "grootvader" band pas na het maken van de "vader" band en "zoon" band weer overgeschreven wordt ten behoeve van de dan volgende "generatie". Op deze manier kan men altijd op twee oudere generaties van een bestand (en de sindsdien aangebrachte mutaties) terugvallen als dat nodig is. Per bestand betekent dit uiteraard een investering van drie banden, hetgeen met het oog op de verhoogde veiligheid echter niet veel is.

Voor files met een fixed record opbouw is de benodigde bandlengte voor een bestand met

L = aantal logical records
B = blocking factor
IBG = interblok gap in inches (veelal 0.6 inch)
D = beschrijvingsdichtheid (characters/inch, bijvoorbeeld 200, 556, 800, 1600 of 3200)
C = aantal characters/logical record

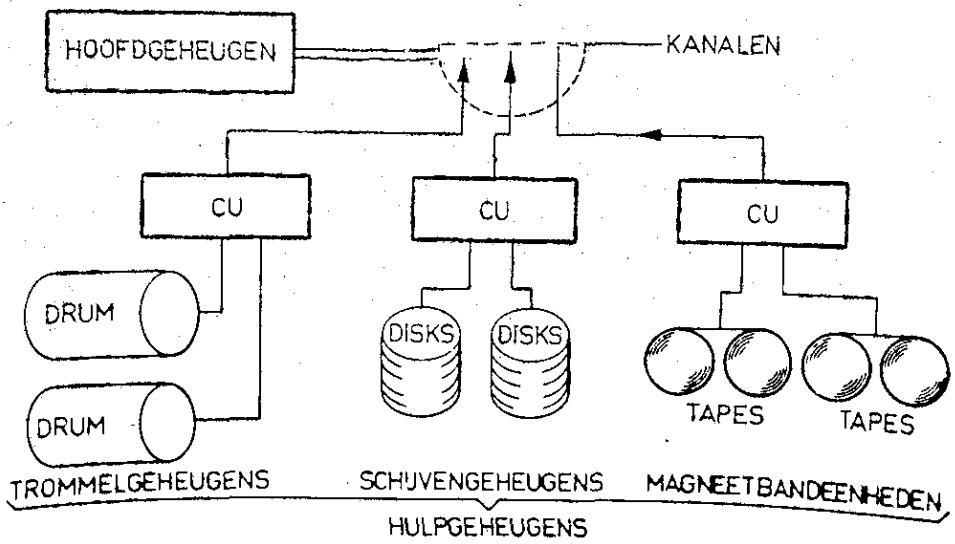
behoudens verlenging van de IBG:

$$\frac{L}{B} * (\frac{C * B}{D} + IBG) = L * (\frac{C}{D} + \frac{IBG}{B}) .$$

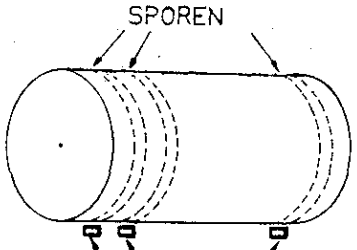
Met de gebruikelijke waarden van D en IBG zien we hieruit meteen dat het weinig zin heeft om het product C * B kleiner dan circa 500 - 1000 karakters te maken, omdat de band dan grotendeels onbeschreven is (in hoofdzaak uit IBG's bestaat).

Factoren die anderzijds pleiten tegen een excessief grote waarde van C * B zijn:

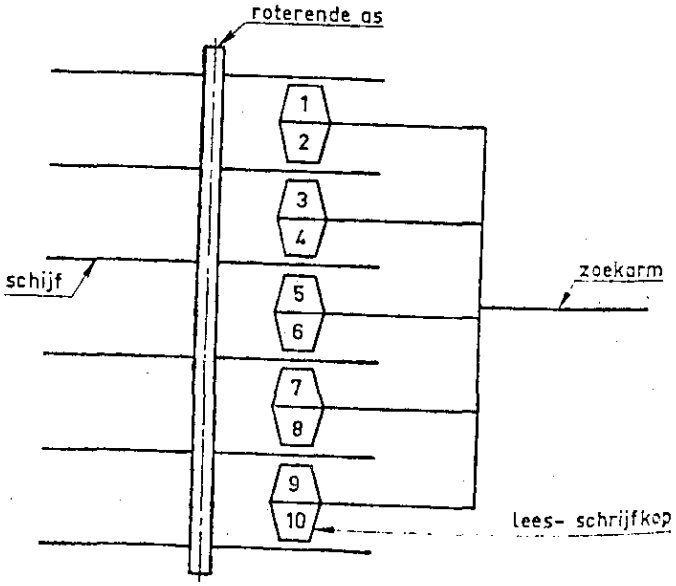
- tenminste dezelfde (buffer) ruimte van C * B karakters moet in het hoofdgeheugen beschikbaar zijn om informatie uit een magneetband te ontvangen of naar een magneetband te verzenden,
- de kans om een slecht stukje magneetband te ontmoeten, neemt bij grote waarden van C * B toe, zodat dan ook de gemiddelde IBG lengte groter zal worden.



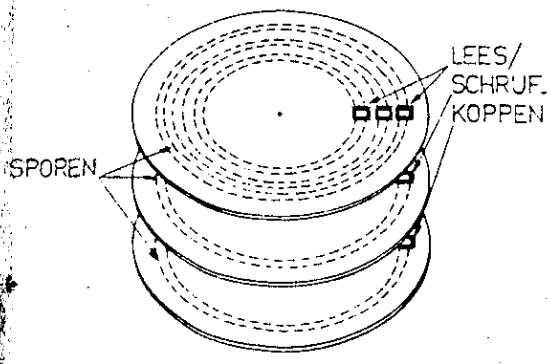
Plaats van de hulpgeheugens in het computersysteem



Trommelgeheugen met individuele koppen per spoor



Schijvenpakket met 'kam' van lees-schrijfkoppen.



Schijvengeheugen met individuele koppen per spoor

Afhankelijk van de omstandigheden zullen blokken zodoende zelden meer dan 5000 - 10.000 karakters bevatten.
Bij betrekkelijk kleine bestanden wordt van een magneetband vaak niet veel meer dan de eerste (van de ca. 700) meter gebruikt. Het "kortwieken" van versleten beginstukken van een band was zodoende gebruikelijk. Het gebruik van trommels of schijven is voor deze kleine bestanden te prefereren, ook al omdat deze niet, zoals bij magneetbanden, door de machine operateur klaargezet hoeven te worden. Door de ontwikkeling van schijven beperkt de rol van magneetbanden zich meer tot de opslag van grote hoeveelheden (historische) gegevens.

Naast de structuur van fixed format records komt men ook de volgende "formats" tegen:
- ongeblokte variabele lengte records, waarbij in het begin van ieder record de recordlengte moet worden vastgelegd,
- geblokte variabele lengte records, waarbij in het begin van ieder blok de bloklengte wordt vastgelegd en in het begin van ieder logical record de logical recordlengte,
- "ongedefinieerde" lengte records, waarbij uiteraard geen blocking mogelijk is en waarbij de controle mogelijkheden gering zijn.

De lees- en schrijftijd van een blok wordt bepaald door een groot aantal factoren, die kenmerkend zijn voor een bepaald type magneetbandeenheid:
- de start- en de stoptijd, ongeveer 5 - 30 ms, nodig om een band uit rust te versnellen tot een constante transportsnelheid en omgekeerd (in rust staat de lees/schrijfkop boven een IBG);
- de tijd nodig om de IBG te doorlopen, ongeveer 8 - 10 ms (worden records op volle snelheid gelezen/geschreven dan hoeft de start- en stoptijd niet in aanmerking te worden genomen);
- de bandtransportsnelheid (van 37.5 tot 250 inch/sec), die samen met de dichtheid de eigenlijke leessnelheid, uiteenlopend van 15.000 tot 400.000 karakters/sec., bepaalt;
- de lengte van het blok.

Daar de som van start- en stoptijd en IBG-tijd tenminste ongeveer 10 ms is, zal bij een bandleessnelheid van 50 karakters/ms pas een bloklengte van 500 karakters zinvol zijn. Bij nog snellere bandeenheden zal men dus flink moeten blokken, alvorens die hogere snelheid tot zijn recht komt.

Opgave

Neem aan dat IBG = 1.5 cm, bandleessnelheid = 50 karakters/ms, (start-stop + IBG)-tijd = 10 ms, banddichtheid = 500 karakters/cm, bandlengte = 700 m; teken dan een grafiek waaruit af te lezen zijn als functie van de bloklengte: de leestijd per blok en het aantal blokken per band.

Andere grootheden, die soms bij tijdschattingen bij gebruik van magneetband in aanmerking moeten worden genomen, zijn de terugspoeltijd van een volle band, die 1-4 m kan zijn, en de opzettijd van een band die, afhankelijk van de werkmethode van een operateur, ½-2 m kan zijn. Banddimensies en magnetische beschrijving zijn in hoge mate gestandaardiseerd.

Banden zijn op grond van de reeds genoemde eigenschappen alleen geschikt voor een sequentiële verwerking van records; beschikt men niet over ("directe toegang") trommel- of schijfeenheden, dan zal men de informatieverwerking volledig aan moeten passen bij deze sequentiële toegankelijkheid.

Opgave

Ga na hoe men $\sum_j a_{ij}\ b_j$ voor alle i moet berekenen wanneer de a_{ij} op één band staan (in record i_o alle $a_{i_o j}$) en de b_j op een andere band (b_{j_o} in record j_o).

2.3.2. Magneettrommel

Magneettrommels waren eerder in gebruik dan magneetband (en zelfs kernengeheugens) maar hebben zich met ups en downs nog altijd gehandhaafd, vooral als buffergeheugen tussen hoofdgeheugen en peripherie apparatuur en/of magneetbanden.
Het principe van een trommelgeheugen is dat informatie op een "spoor" op het magnetiseerbare cylindervlak van een met constante snelheid draaiende trommel geschreven/gelezen wordt met behulp van een vaste schrijf/leeskop per spoor. De constructie is zodanig dat:

- in tegenstelling tot magneetband de positionering van het lees/schrijf-mechanisme zo nauwkeurig is dat informatie (eventueel na verandering) weer precies op de oude plaats teruggeschreven kan worden,
- gemiddeld na een halve omwentelingstijd (d.i. afhankelijk van het type na 5 - 20 ms) een willekeurig blok informatie uitgelezen kan worden (het selecteren van de gewenste leeskop en het overbrengen van informatie naar het kernengeheugen zijn processen, die zich met snelheden van 0.3 tot 1.2 M karakters/sec. afspelen),
- in tegenstelling tot vele nog te bespreken schijfgeheugens de trommels niet verwisselbaar zijn.

In verband met mechanische sterkte- en balanceringsproblemen kunnen trommelgeheugens niet èn snel èn groot gemaakt worden. Langzame grote trommelgeheugens (capaciteit 10 M karakters, halve omwentelingstijd 20 ms) worden verdrongen door schijfgeheugens en snelle kleine trommelgeheugens (capaciteit 1 M karakters, halve omwentelingstijd 5 ms) door kernengeheugens (met lees/schrijftijden van de orde van 0.01 ms!) zodat het de vraag is of trommelgeheugens ondanks hun hoge betrouwbaarheid nog een lange toekomst hebben. De kosten per bit bedragen afhankelijk van de snelheid 0.5 tot 1.5 cent (inclusief de besturingseenheid).

Opgemerkt moet worden dat in sommige trommelgeheugens lees/schrijf-koppen niet vast opgesteld zijn, maar bevestigd op een gemeenschappelijke arm die evenwijdig met het cylinderoppervlak over een aantal spoorposities bewogen kan worden. Bij de halve omwentelingstijd van de trommel komt dan echter ook nog de positioneringstijd van de arm, die afhankelijk van de afstand waarover de arm zich moet verplaatsen van de orde van tientallen ms is. De som van halve omwentelingstijd en positioneringstijd is 40-100 ms, de capaciteit van 20 tot 150 M karakters, de kosten per bit zijn een factor 10 lager dan bij trommels met vaste koppen, de lees/schrijfsnelheden zijn ongeveer 0.1 tot 0.2 M karakters/sec.

Net zoals op magneetbanden (en de nog te bespreken magneetschijf-geheugens) kan de vastlegging van records op een spoor met verschillende "formats" gebeuren. Ook hier kent men de begrippen blokgap (die echter heel klein is omdat die niet meer een functie voor inloop en uitloop van de bandbeweging heeft), bloklengte, enz.; bovendien is op een spoor ook

besturingsinformatie (zoals nummer van een spoor, begin van een spoor, aantal records op een spoor, enz.) vastgelegd. (Deze laatste informatie is echter veelal niet toegankelijk voor de gebruiker, maar wel voor het operating system voor controle doeleinden.)

De bloklengte moet bij voorkeur een geheel aantal keren "passen" op de maximaal beschikbare spoorcapaciteit. Leveranciers verstrekken als regel tabellen waaruit het aantal blokken/spoor en de eigenlijke leestijd afgelezen kunnen worden als functie van de bloklengte.

De belangrijkste functie van snelle trommels is de permanente en tijdelijke (bij gebruik van virtuele geheugens en bij time-sharing) opslag van programma's en harde software die snel beschikbaar moeten zijn. De langzamere trommels worden vooral voor gegevensopslag gebruikt.

2.3.3. Magneetschijven

Magneetschijfgeheugens zijn na ongeveer 1965 in toenemende mate in gebruik genomen om tegemoet te komen aan de wens naar goedkope willekeurig toegankelijke secundaire geheugensystemen. In principe bestaan zij uit een enkele ("cartridge") of een pakket (6-10) op een gemeenschappelijke as draaiende platte schijven waarvan de twee oppervlakken magnetiseerbaar zijn. Informatie wordt op (200-500) concentrische spoten (niet spiraalvormig zoals op een grammofoonplaat!) geschreven met behulp van een lees/schrijfkop voor ieder oppervlak. Deze koppen zijn op een arm ("kam") bevestigd, die evenwijdig met de schijfoppervlakken heen en weer kan bewegen, zodat de koppen boven ieder spoor gebracht kunnen worden. De sporen op de oppervlakken behorende bij eenzelfde kampositie vormen een zg. "cylinder". De capaciteit per spoor is niet afhankelijk van het cylindernummer (zodat de binnenste sporen met een hogere dichtheid geschreven moeten worden). De constructie is zodanig dat:

- zoals bij magneettrommels de positionering van het lees/schrijfmechanisme zo nauwkeurig is, dat informatie (eventueel na verandering) weer precies op de oude plaats teruggeschreven kan worden;
- het uitlezen van een willekeurig blok informatie niet alleen als bij trommels een halve omwentelingstijd (ongeveer 10-20 ms) vraagt, maar ook de tijd nodig om de kam te verplaatsen als de leeskop niet toevallig boven het goede spoor staat. De gemiddelde armpositioneringstijd ligt tussen de 30 en 100 ms en is dus als regel de belangrijkste factor in de totale lees/schrijftijd (net als bij de trommels gaan het electronisch selecteren van de leeskop en het overbrengen van karakters met snelheden van 0.3 tot 0.8 M karakters/sec.);

- de kleinere schijfpakketten (tot een capaciteit van ca. 30 miljoen karakters; kosten per bit ca. 0.01 tot 0.04 cent) veelal uitwisselbaar zijn (en daarvoor in veel opzichten gestandaardiseerd); de grotere tot een capaciteit van een paar honderd miljoen karakters zijn vast opgesteld. De laatsten hebben veelal een vaste kop per spoor, zodat het armmechanisme/tijd vervalt en de karaktertransportsnelheid iets hoger is (0.3 tot 1.5 M kar/sec.); de kosten per bit zijn ca. 0.1 tot 0.6 cent (vergelijk deze gegevens met die van trommels!).

De uitwisselbaarheid van schijfpakketten heeft vele voordelen:

- de secundaire geheugencapaciteit kan (met het bezwaar van manuale handelingen) uitgebreid worden zonder direct nieuwe schijfgeheugenkasten aan te schaffen,
- bij het uitvallen van een schijfgeheugenkast kan het pakket overgezet worden op een andere kast en het werk kan worden voortgezet,
- bij het uitvallen van een hele installatie kan in principe het werk voortgezet worden op een soortgelijke installatie (in de praktijk treden hierbij toch wel vaak andere problemen op!).

Schijfgeheugens zullen misschien op den duur verdrongen worden door "solid-state" of door optische (laserstraal) geheugensystemen, wanneer deze door het experimentele stadium heen zijn. Tot ca. 1975 wordt echter verwacht dat door toenemende capaciteit per pakket (30 M kar. in 1967, 100 M kar. nu, 300 M kar. in 1975?) de opslagkosten per bit verder zullen dalen, zodat schijven in toenemende mate trommels en banden zullen verdringen. Dat dit niet gedachtenloos moet gebeuren, is als volgt te illustreren: bij een physieke recordlengte van 3600 karakters is de record leestijd zowel voor magneetschijf als magneetband veelal ongeveer 50 ms (bij een trommel ongeveer 10 ms). Voor de sequentiële verwerking van een high-activity bestand opgebouwd uit zulke records biedt een schijfgeheugen dan geen voordelen boven een magneetband, die bovendien een veel goedkopere informatiedrager is. Pas wanneer het bestand een low activity heeft, of de verwerking niet-sequentieel geschiedt, biedt een schijfgeheugen voordelen.

Magneetstripgeheugens vormen wat gebruikskarakteristiek betreft enige gelijkenis met schijfgeheugens, al zijn ze nog een factor tien langzamer. Informatie wordt daarbij vastgelegd op magneetbandstrips, waarvan een aantal op een plastische drager is geplakt. Deze dragers

zitten in een bakje. Hieruit wordt langs elektromechanische weg de gewenste drager geselecteerd, de drager wordt op een draaiend trommeltje opgespannen, waarna een beweegbare of vaste lees/schrijfkop als bij een trommel de gewenste informatie afleest. De random toegangstijd van ongeveer ½ s maakt dat deze stripgeheugens alleen voor sequentiële informatieverwerking geschikt zijn. Bovendien zijn ze tot dusver meestal niet erg betrouwbaar gebleken, zodat de toekomst van stripgeheugens ondanks lage kosten/bit erg onzeker is.

2.3.4. Informatietransport tussen secundaire informatiedragers en hoofdgeheugen

De schrijver van een proceduretaalprogramma verwacht met voor een bepaalde taal karakteristieke opdrachten - die we voortaan READ en WRITE zullen noemen - een logisch record te lezen of weg te schrijven. Anderzijds bewerkstelligen hardware opdrachten - die wij GET en PUT zullen noemen - de uitwisseling van informatie tussen hoofdgeheugen. Wat is nu de samenhang tussen deze opdrachten?

Deze samenhang wordt tot stand gebracht door het operating system dat bijv. met een GET opdracht physiek informatietransport van een secundair geheugen naar een "bufferruimte" in het hoofdgeheugen veroorzaakt. Het eerste logische record vervat in het blok in de bufferruimte wordt dan met de eerste READ opdracht "toegankelijk" gemaakt voor verdere behandeling (of dit nu gebeurt ter plaatse of dat het logical record weer verhuisd wordt naar een andere plaats is voor de gebruiker verder niet van belang). Met iedere volgende READ opdracht wordt telkens het volgende logische record toegankelijk gemaakt tot het laatste in de buffer toe. Alvorens de volgende READ opdracht te gehoorzamen moet het operating system eerst een GET opdracht uitvoeren, waarmee de buffer met het volgende blok gevuld wordt, enz.

Om te voorkomen dat het verwerkingsprogramma moet wachten op de uitvoering van de GET opdracht, werkt men soms met twee of drie buffers, die door het operating system alvast "vooruit" gevuld worden zodra een bufferruimte vrijkomt. Dit heeft uiteraard alleen maar zin wanneer de bewerkingstijd van de logical records in een buffer gemiddeld niet korter is dan de vultijd van een buffer omdat de informatiebewerking anders "inloopt" op het informatietransport. Is omgekeerd de bewerkingstijd veel groter dan de buffervultijd dan is een tweede buffer ook niet zinvol, omdat dan een stuk kernengeheugen lange tijd "ongebruikt" blijft.

OVERZICHT VAN ENKELE INFORMATIEDRAGERS

	ponsband	ponskaart	magneetband	magneet-trommel	schijfpakket	magneet-strip
betrouwbaarheid	goed	goed	zeer goed	zeer goed	goed	gering
kosten inf.drager	~ƒ 3,- /rol	~½ct/kaart	~ƒ 100,-/band			
capac. inf.drager (in karakters)	0.1 M	80 kar.	4-20 M	0.1-10 M	10-100 M	50-1000 M
opbergmogelijkheid	gering	goed	zeer goed		goed	goed
leessnelheid (in duizendtallen)	0.5-2 kar/s	0.5-2 cards/m	15-400 kar/s	300-1200 kar/s	300-800 kar/s	~50 kar/s
random toegangstijd (ms)				50-20	10-100	200-700
kosten per kar(ct)		0.005	~0.001	4-12	1-5	~0.5
max.physical record	~ ∞	80 kar	~ ∞	spoor-lengte	spoor-lengte	spoor-lengte

Het zal duidelijk zijn dat het vaststellen van de optimale bloklengte en het aantal buffers afhankelijk is van factoren als:

- hardware eigenschappen (toegangs- en informatietransporttijden),
- software eigenschappen (hoelang duurt de verwerking van READ en GET opdrachten, gezien alle controles op de goede uitvoering die onzichtbaar voor de gebruiker daarbij uitgevoerd worden door het operating system),
- probleemeigenschappen die niet alleen bepalen hoeveel ruimte in het hoofdgeheugen nog beschikbaar is voor buffers, maar ook hoe lang de verwerking van een logical record duurt.

Een voor alle toepassingen geldig wiskundig model is hiervoor nauwelijks op te zetten, zodat van geval tot geval dit probleem bestudeerd moet worden. Dat het niet triviaal is, kan hieruit blijken dat de verwerkingstijd van een slecht opgezet systeem factoren drie tot vijf langer kan zijn dan bij een goede opzet. Bij een eerste verkenning van een probleem hanteert men meestal vuistregels van het type

- zorg er voor dat een instructie-reeks niet meer dan 5000 tot 10.000 woorden in het geheugen beslaat (dit correspondeert met ongeveer 500 tot 1000 statements in een proceduretaal, die voor een programmeur nog goed te overzien zijn);
- zorg er voor dat bloklengten tussen 500 en 5000 karakters zijn, daar dit een redelijk compromis is t.a.v. leestijd en gebruik van band en schijf;
- voorkom, indien enigszins mogelijk, het werken met multivolume files en multifile magneetband-bestanden om verwerkingstijd te sparen.

In de grotere moderne machines gebeurt het informatie-transport van secundaire naar hoofdgeheugen via de zg. kanalen, die ieder voor zich een eigen toegang tot het kernengeheugen hebben. Daar ook de centrale verwerkingseenheid een eigen toegang tot het gheugen heeft wordt het met deze configuratie mogelijk om tijdens het "rekenen" gelijktijdig informatietransporten naar en van secundaire geheugens en invoer- en uitvoerapparaten te bewerkstelligen. Het geheel moet uiteraard wel gecoördineerd worden door het operating system (bedrijfssysteem). Welke problemen hierbij optreden en hoe die opgelost worden is het onderwerp van een ander college, zodat we hier niet verder op in zullen gaan.

Ten onrechte is in de begintijd van de met moderne configuraties mogelijke multiprogrammering wel eens gedacht dat de problemen van bestandsopbouw en recordgrootte niet meer zo belangrijk waren omdat "de centrale verwerkingseenheid wel altijd iets nuttigs kon doen". Men ziet met deze argumentatie dan de volgende factoren over het hoofd:

- secundaire geheugeneenheden moeten zo gebruikt worden dat niet alleen hun capaciteit zo goed mogelijk benut wordt, maar ook zo dat ze zo min mogelijk beslag leggen op de kanalen omdat anders wachttijdproblemen kunnen optreden bij het benutten van kanalen;
- het aanroepen van het operating system voor informatietransporten of voor overgang naar een ander programma betekent iedere keer een "omsteltijd" die als verloren tijd moet worden beschouwd. Het aantal aanroepen van het operating system moet dan ook niet groter worden dan beslist nodig is;
- de informatie-transporttijd per blok kan sprongsgewijs veranderen i.v.m. bijvoorbeeld de rotatietijd en armtijd bij schijfsystemen ("mist" men bij het lezen van een record net het laatste moment, dan is nog een omwenteling van de schijf nodig hetgeen de toegangstijd van het record vrijwel verdubbelt. Moet men door een slechte bestandsorganisatie steeds nieuwe cylinders opzoeken in plaats van binnen dezelfde cylinder te blijven, dan wordt de gemiddelde toegangstijd tot een record ongeveer viermaal zo lang).

De systeemopzet moet er dan ook net als in het verleden op gericht zijn om de rekentijd in een programma niet veel korter te maken dan de informatietransporttijd, maar beslist ook niet veel langer omdat anders grote stukken hoofdgeheugen lange tijd ongebruikt zouden kunnen blijven ("balancing"). Tactische middelen om de rekentijd nuttig te verlengen bij een gegeven informatietransporttijd zijn:

- met de, na het informatietransport, beschikbare gegevens meer bewerkingen uit te voeren door meer taken in één programma te verenigen (nadelen: meer geheugenruimte voor het programma, ingewikkelder probleemanalyse en moeilijker te onderhouden programma's),
- ingewikkelder recordstructuren in te voeren als dit tenminste tot geheugenruimtebesparing voert; het isoleren van data uit deze ingewikkelde structuren kost immers meer tijd en bovendien zal de informatietransporttijd door de compactheid van ingewikkelder recordstructuren afnemen,

- records die om de een of andere reden, volgend uit de probleemstelling, niet betrokken worden bij het rekengedeelte af te splitsen (file splitting) in een "niet-actief" deelbestand (nadeel: ingewikkelder bestandsadministratie, het wel of niet-actief zijn is vaak een tijdelijk iets),
- data in een record vast te leggen in een vorm die optimaal is voor input/output (bijv. in BCD vorm) maar conversie naar binaire vorm vereist voor het rekengedeelte.

Opgave

Onderzoek waarom in het algemeen in een low activity file de physical records lang moeten zijn en omgekeerd. Waarom moeten transaction files en rapportage bestanden dus bij voorkeur uit korte physical records opgebouw worden?

Opgave

Beredeneer de voordelen van file consolidatie, het tegenovergestelde van het hierbovengenoemde file splitting, met betrekking tot bestandsadministratie, operators werk, programmeerwerk en geheugengebruik.

In samenhang met het voorgaande zij er ook nadrukkelijk op gewezen dat bij druk bezette kanalen de toegangstijdgegevens van een secundair geheugensysteem niet bepalen hoeven te zijn voor de wachttijd, die verloopt tussen het aanvragen van informatie-transport en het arriveren van de gevraagde informatie in het kernengeheugen.
Onder bepaalde veronderstellingen (o.a. het random binnenkomen van vragen om informatie-transport, een exponentiële verdeling van transportafhandelingstijden) geeft de wachttijdtheorie het resultaat dat de gemiddelde wachttijd evenredig is met $1/(1-r)$ waarin r de verhouding is van de gemiddelde afhandelingstijd van informatie-overdracht en de tijd die gemiddeld verloopt tussen twee aanvragen om informatietransport. Voor kleine r is de gemiddelde wachttijd zodoende vrij constant, doch als r tot 1 nadert gaat de gemiddelde wachttijd drastisch omhoog. Al gelden de genoemde veronderstellingen niet precies, toch wordt zo'n gedrag van de gemiddelde wachttijd waargenomen bij druk bezette kanalen. De conclusie die hieruit getrokken moet worden is zodoende dat wanneer naar schatting $r > 0.5$ is, dan een gedetailleerde analyse (eventueel door simulatie) gemaakt moet worden van het hele informatietransport, wil men niet later voor teleurstellingen komen te staan.

2.4. Sequentiële strukturen

Sequentiële strukturen zijn blijkens het voorgaande gekenmerkt door het feit dat records fysiek in een bepaalde volgorde in een bestand geplaatst zijn. Meestal is het een volgorde op grond van een bepaald sorteerkenmerk (of een combinatie van kenmerken), zoals een bepaald sleutelitem uit een record. Het kan echter ook gebeuren op grond van een niet in een record opgenomen kenmerk, zoals bijvoorbeeld een sortering op grond van de frequentie van raadpleging van records. Eenvoudigheidshalve herleidt men zo'n kenmerk echter meestal tot het eerste door geschikte keuze van de sleutel. Daar er geen relatie is tussen recordsleutel en recordadres, is toegang ("access") tot de records alleen maar mogelijk met de sequentiële access methode, d.w.z. het na elkaar lezen van records in de aanwezige volgorde.

Op de problemen van toevoegen, verwijderen of veranderen van records bij gebruik van een sequentiële struktuur zullen we dieper ingaan, na eerst het zoeken van een bepaald record besproken te hebben.

2.4.1. Zoekmethoden

Voor alle hierna te behandelen zoekmethoden geldt dat een veel voorkomende programmeringsfout is dat geen rekening wordt gehouden met een negatief zoekresultaat of met de mogelijkheid dat (terecht of ten onrechte) meer dan één record een bepaalde zoeksleutel heeft.

a) ongesorteerd bestand

Wanneer we in een ongesorteerd bestand naar een record met een bepaalde sleutel zoeken, zit er niet veel anders op dan alle records één voor één, beginnend met de eerste, te vergelijken met de zoeksleutel. Als bekend is dat de betreffende sleutel maar één keer voorkomt, kan zodra overeenstemming gevonden is, het vergelijkingsproces gestaakt worden. Anders moet het bestand tot het einde doorzocht worden. Rekening houdend met het twee of meer keren voorkomen van records met dezelfde sleutel is de zoektijd uiteraard evenredig met n, het aantal aanwezige records.

b) gesorteerd bestand

Dezelfde "lineaire" zoekmethode als net genoemd kan ook in een gesorteerd bestand toegepast worden, met de verijfing dat, zodra een recordsleutel groter is dan de zoeksleutel, niet meer verder gezocht hoeft te worden. Omdat de zoektijd nu evenredig is met de helft van het aantal records worden bij grote sequentiële bestanden zoekvragen opgespaard, op sleutel gesorteerd en dan in één gang (batch) verwerkt.

Een iets gekompliceerder, maar veel efficiënter zoekproces is de "binaire" zoekmethode. Hierbij begint men de zoek in het "midden" van het bestand en herhaalt het proces op de volgens de sleutels relevante "helft" van het oorspronkelijke bestand. Tegen het einde van dit proces is een lineaire zoekmethode in het overgebleven deel soms efficiënter door zijn programma-eenvoud. Deze binaire zoekmethode is uiteraard direct toepasbaar bij een "immediate-access" informatiedrager (zoals een kernengeheugen) en gezien de gemiddelde toegangstijden ook bij "random-access" informatiedragers (zoals trommels en schijven), maar niet bij sequentiële informatiedragers (zoals magneetbanden) met hun problemen van passeren van alle records bij het lezen en terugspoelen.

In de literatuur treft men ook nog de z.g. "skip-sequential" zoekmethode aan, die als het ware tussen de lineaire en binaire in staat. Met vrij grote sprongen gaat men dan door het bestand totdat men ôf konstateert dat het gewenste record gevonden is ôf sequentieel teruggaat tot hooguit het vorige "sprongpunt". In theorie vraagt deze methode meer inspecties dan de binaire methode, doch het program is iets eenvoudiger.

Het maximum aantal inspecties bij een binaire zoekmethode wordt bepaald door de kleinste z die voldoet aan $2^z \geq n+1$, als n het aantal records is. (Immers door volledige inductie bewijzen we dat tot 2 records toe 2 inspecties voldoende zijn, dan tot 4 records 3 inspecties,..., en tot 2^{z-1} records z inspecties.) Bij benadering is de zoektijd dus evenredig met $^2\log n$, hetgeen voor grote n aanmerkelijk beter is dan bij een lineaire zoek. Het programma is, zoals gezegd, echter iets ingewikkelder.

Opgave

Ontwerp Algolprogramma's voor de bovengenoemde zoekmethoden.

2.4.2. Bestandsmutatie

De gegevens van een bestand vormen zelden een statisch geheel; vrijwel altijd zullen gegevens worden veranderd, toegevoegd of verwijderd omdat de situatie waarop deze gegevens betrekking hebben ook aan verandering onderhevig is. De snelheid van verandering en het belang dat men hecht aan het "bij" zijn van een bestand bepalen de frequentie waarmee het bestand bijgewerkt moet worden.

Is deze frequentie zeer hoog dan zal men over willen gaan op post-voor-post verwerking van mutaties. Zuiver sequentiële informatiedragers komen dan niet meer in aanmerking (tenzij het bestand zo klein is, dat het in korte tijd in zijn geheel in het kernengeheugen kan worden opgeslagen en daar ge-

muteerd). Willekeurig toegankelijke geheugens openen dan ook de weg voor het on-line muteren van een bestand, d.i. het vanuit terminals direct invoeren van mutatiegegevens voor verwerking door de centrale machine. Het probleem van het vinden van het betreffende record in het bestand is eenvoudig vergeleken met problemen van koördinatie van bestandsbijwerking en bestandsbewaking. Immers bij het bijwerken moet men bedacht zijn op het "bijna tegelijkertijd" muteren van eenzelfde record vanuit twee verschillende terminals, hetgeen gauw tot foute behandeling aanleiding kan geven. De op te lossen problemen zijn soortgelijk aan die bestudeerd voor het ontwerpen van een bedrijfssysteem (operating system), zodat hier niet verder op ingegaan zal worden. (Wanneer dit gezien de omstandigheden mogelijk is, is te overwegen om de on-line mutaties op te sparen en deze dan in batchvorm toe te voeren aan het bestand. Ook de problemen van bestandsbewaking worden dan veel eenvoudiger).

In het volgende zullen we eerst ingaan op in batch verwerken van mutaties op een sequentieel georganiseerd bestand. De vier typen mutaties die we zullen beschouwen, zijn het invoegen van nieuwe records, het veranderen van gegevens in een record, het verwijderen van records uit het bestand en tenslotte het veranderen van recordsleutels. Om verder de gedachten te bepalen, nemen we aan dat

- het bestand gesorteerd is op band (waarom noodzakelijk?),
- ook de mutaties gesorteerd zijn op band (waarom noodzakelijk?),
- bij iedere sleutel maximaal één record in het bestand aanwezig is,
- doch bij iedere sleutel meer mutaties kunnen voorkomen,
- dat het mutatieproces van eventueel voorkomende fouten een verslag moet vastleggen,
- dat van uitgevoerde mutaties een (kort) verslag ("audit trail" of "journal") moet worden vastgelegd.

Hoe lang bestandsrecords en mutatierecords zijn en hoe de verslagleggingen precies moeten zijn zal hier in het midden worden gelaten, aangezien dit toch alleen maar programmeerdetailleringen met zich meebrengt. Evenzo laten we in het midden hoe precies het einde van het bestand of van de mutaties wordt gekonstateerd en of bijvoorbeeld ten behoeve van controle het laatste bestandsrecord een telling van het aantal voorafgaande records bevat en eventueel een totalisatie van bepaalde items uit die voorafgaande records. (Dit betekent niet dat zulke controles onbelangrijk zijn, doch zij betekenen wederom een programmeringsdetail.) Het einde van het mutatiebestand wordt vaak aangegeven met een

"sluitrecord", waarin de hoogst mogelijke sleutel staat (en mogelijk weer controle totalen). Dit geeft echter vaak vergissingen omdat men vergeet om dit sluitrecord toe te voegen, zodat men eerder moet vertrouwen op de hardware signalering van het einde van een bestand.

Een voor de hand liggende en in de praktijk nog vaak voorkomende methodiek is het in vier verschillende gangen uitvoeren van de vier genoemde typen mutaties. Deze methodiek moet om verschillende redenen verworpen worden ten gunste van een methodiek waarbij alle mutaties in één gang verwerkt worden:

- het bestand wordt vier keer gelezen en overgeschreven, waarmee veelal de totale verwerkingstijd onnodig vervierdubbeld wordt,
- het vereist grote oplettendheid van de operateurs omdat de mutaties beslist in de goede volgorde (toevoegen, veranderen, sleutelveranderingen, verwijderen) moeten worden uitgevoerd (ga dit na),
- foutmeldingen en audit trails betrekking hebbende op eenzelfde record staan over meer staten verspreid, hetgeen de manuale inspectie van rapportages omslachtig maakt.

Dat veelal toch de omslachtige methodiek wordt gebruikt, is vermoedelijk te wijten aan de overweging dat het in één keer aanbrengen van mutaties moeilijk is en de angst (niet ten onrechte) fouten te maken in de programmalogica. In het volgende zullen we hier een blauwdruk voor aangeven, welke altijd toegepast kan worden.

Het zal duidelijk zijn dat bij verwerking in één gang in de mutatierecords (verder transacties te noemen) aangegeven moet worden om welke mutatiesoort het gaat. Een eenvoudige en effektieve methode is om de transactiesleutel met een extra digit achteraan te "verlengen", bijv. als de records in opklimmende volgorde in het bestand staan, met een

0 als het om een toevoeging gaat,
2 als het om een verandering gaat,
4 als het om een sleutelverandering gaat,
6 als het om een verwijdering gaat.

Aangezien transactierecords toch gesorteerd moeten worden, geeft sortering met de verlengde sleutel meteen de goede volgorde van mutaties bij eenzelfde recordsleutel. (Deze gedachte van sleutelverlenging kan ook toegepast worden als

we de veranderingen in een bepaalde, bijvoorbeeld chronologische, volgorde willen aanbrengen. Hoewel dit in situaties, waarbij veranderingen zowel een positief als een negatief teken kunnen hebben, soms zeer belangrijk kan zijn, zullen we deze gedachte niet verder vervolgen.)

Bij een eerste overpeinzing zal het mutatieprobleem eenvoudig lijken. Op grond van de transactiesleutel zoekt men in het bestand het bijbehorende record op (of het daarop volgende in het geval van toevoeging), brengt de mutatie aan en schrijft het record weer weg. De complicaties ontstaan echter doordat:

- bij dezelfde recordsleutel weleens meer mutaties kunnen voorkomen, zodat wegschrijven pas mag gebeuren nadat een transactie met een hogere recordsleutel is geconstateerd,
- bij toevoeging van een record het wegschrijven van een (eventueel nog gemuteerd) transactierecord moet plaats vinden, bij verandering het wegschrijven van een gemuteerd bestandsrecord, terwijl bij verwijdering helemaal geen record wordt weggeschreven.
  Zoals we later in detail zullen zien, worden deze problemen opgelost door naast de buffers voor bestandsrecord en transactierecord een derde buffer in te voeren, van waaruit naar het nieuwe bestand geschreven wordt,
- het mutatieproces ook zoveel mogelijk menselijke fouten moet signaleren.

Het mutatieproces zal worden beheerst door de transactierecords. Is een transactierecord een toevoeging, dan zal het volgende moeten gebeuren:

- alle bestandsrecords met een sleutel kleiner dan dat van het transactierecord kunnen overgeschreven worden naar het nieuwe bestand,
- als een bestandsrecord met dezelfde sleutel als het transactierecord wordt aangetroffen, dan is het transactierecord door een menselijke fout toegevoegd, waarvan dan ten behoeve van de controle een boodschap wordt gegenereerd,
- in andere gevallen wordt het transactierecord met mogelijk nog volgende mutatierecords bij dezelfde recordsleutel aangevuld en dan pas op het nieuwe bestand geschreven.

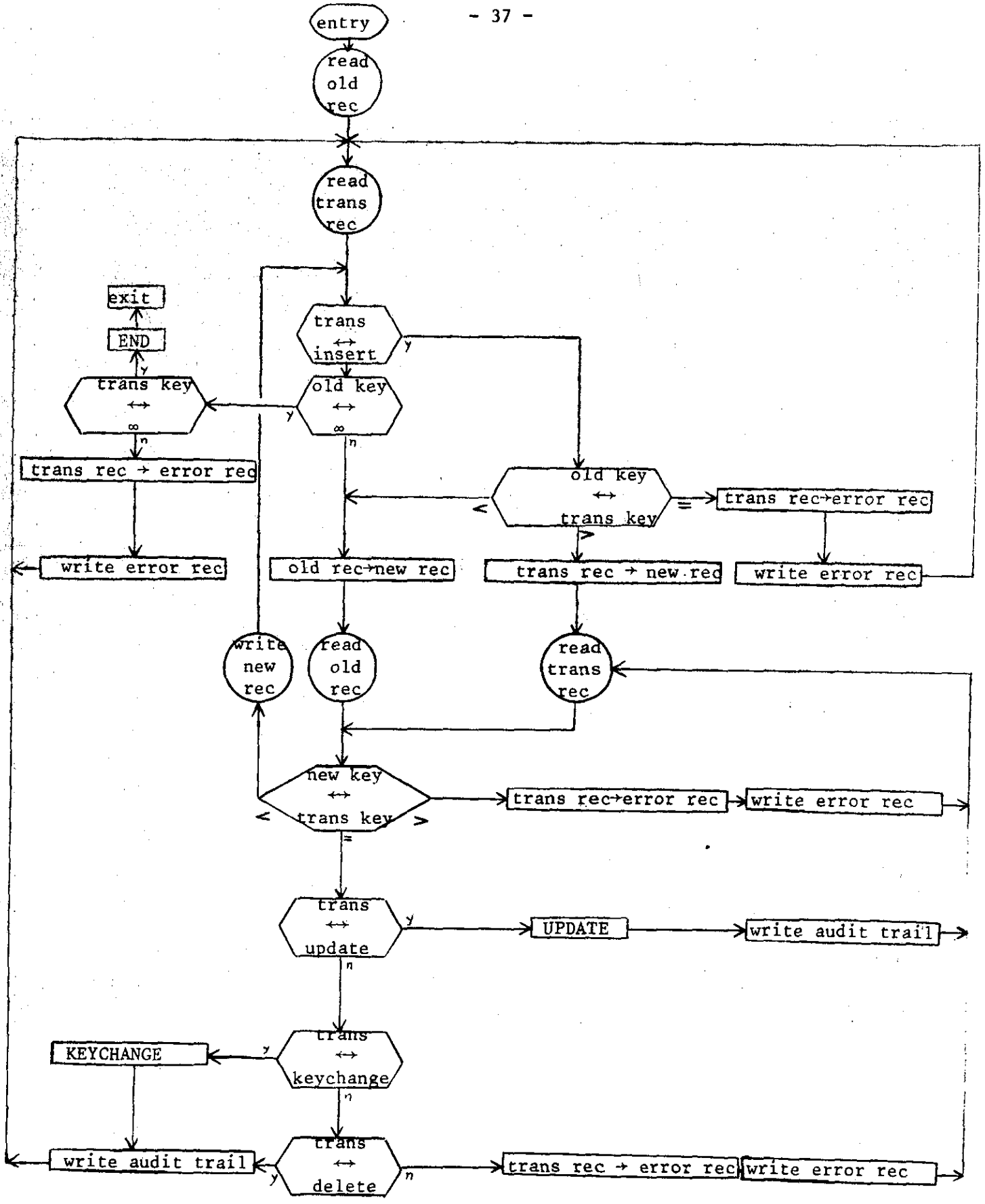
Is het transactierecord een verandering van inhoud of sleutel, of een verwijdering, dan moet het volgende gebeuren:

- alle bestandsrecords met een sleutel kleiner dan dat van het transactierecord kunnen overgeschreven worden naar het nieuwe bestand,

. ontbreekt een bestandsrecord met dezelfde sleutel als het transactierecord, dan is het transactierecord door een menselijke fout toegevoegd, waarvan wederom ten behoeve van de controle een boodschap wordt gegenereerd,
. in andere gevallen zorgen we er voor dat het bestandsrecord aangevuld, resp. weggelaten wordt in het nieuwe bestand.

In de voorgaande gevallen is het mogelijk dat wederom door menselijke fouten de sorteervolgorde van transactiebestand of oude bestand verstoord is, hetgeen voor de sequentiële verwerking van beide bestanden uiteraard funest is en aanleiding kan geven tot reeksen foutmeldingen! Het is daarom verstandig om in de record leesprocedure van beide bestanden de controle op te nemen dat de sleutel van ieder gelezen record niet kleiner is dan die van het daarvoor gelezen record. Is dit het geval dan kan het hele mutatieproces beter afgebroken worden om één of beide bestanden eerst opnieuw te sorteren.

Ten aanzien van het afbreken en later weer herstarten van een mutatieproces moet men ook zeer voorzichtig zijn wat de veranderingen betreft. Is de verandering een vervanging van een of meer oude items door nieuwe items, dan kan er weinig kwaads gebeuren bij een tussentijds afbreken van een mutatieproces. Is de verandering daarentegen een vermeerdering of vermindering van één of meer recorditems, dan kan het bij een tussentijds afbreken en later weer herstarten van een mutatieproces gebeuren dat de mutatie ten onrechte twee keer wordt uitgevoerd! Men mag er immers nooit op vertrouwen dat de operateur weet tot welk punt de mutaties met succes zijn aangebracht (wanneer de mutaties trouwens al magnetisch vastgelegd zijn, kan hij het niet weten). Men dient er daarom bij een verandering voor te zorgen:
. òf alleen maar vervanging toe te staan (denk aan adresveranderingen),
. òf als dit door de aard van een probleem niet mogelijk is (denk aan magazijnmutaties), de uitvoering van een verandering afhankelijk te maken van een tweede (uniek) kenmerk (naast de transactiesleutel) in het transactierecord en dit over te nemen in het nieuwe bestand. De verandering mag alleen dan aangebracht worden, wanneer dit kenmerk in transactierecord en bestandsrecord niet overeenstemmen! Voor dit tweede kenmerk kan afhankelijk van de aard van het probleem bijvoorbeeld een datum of een bonnummer, enz. gebruikt worden,
. òf het oude sequentiële bestand op een magneetband te bewaren, zodat altijd nog de uitgangssituatie voor een mutatiebewerking volledig beschikbaar is.


entry
read old rec
read trans rec
exit
END
trans ↔ insert
trans key ↔ ∞
old key ↔ ∞
trans rec → error rec
old key ↔ trans key
trans rec→error rec
write error rec
old rec→new rec
trans rec → new rec
write error rec
write new rec
read old rec
read trans rec
new key ↔ trans key
trans rec→error rec
write error rec
trans ↔ update
UPDATE
write audit trail
trans ↔ keychange
KEYCHANGE
write audit trail
trans ↔ delete
trans rec → error rec
write error rec

```
begin   read old record
    RT: read (next) transaction record T;
    IN: if T = insert then begin if old key = transaction key
                        then begin errorrec:=  T; write errorrec; go to RT end
                        else if old key < transaction key
                              then begin newrec:= old rec; read (next) old rec;
                                         go to UD
                                   end
                              else begin new rec:= transaction rec;
                                         read (next) transaction rec; go to UD
                                   end
                    end
               else if old key = ∞
                    then if trans key = ∞ then END
                                   else begin errorrec := T;
                                              write error rec; go to RT
                    else begin new rec := old rec; read (next) old rec;
    UD:                          if new key < trans key
                                 then begin write new rec; go to IN end
                                 else if new key > transaction key
                                      then begin error rec := T; write error rec;
                                                 read (next) trans rec; go to UD
                                           end
                                 else if T = update
                                      then begin update new record;
                                                 write audit trail;
                                                 read (next) trans rec;
                                                 go to UD
                                           end
                                      else if T = keychange
                                           then begin keychange;
                                                      write audit trail;
                                                      go to RT
                                                end
                                           else if T = delete
                                                then begin write audit trail;
                                                           go to RT
                                                     end
                                                else begin error rec := T;
                                                           write error rec;
                                                           read (next) trans rec;
                                                           go to UD
                                                     end
                         end
end
```

Pseudo Algol schema voor updating

Na bestudering van het voorgaande, zal men de programmalogica voor een bestandsmutatie, die op bijgaande pagina's is gegeven in pseudo-Algol en stroomdiagram vorm, kunnen volgen. Hierbij kunnen nog de volgende opmerkingen gemaakt worden:

- de procedure update new record kan aanleiding geven tot een lang programma als de updating een groot record betreft (transactierecords zijn veelal niet groter dan 60-80 karakters, korresponderend met een ponskaart).
- de procedure keychange is niet verder uitgewerkt. Bij een sleutelverandering van slechts enkele records is de beste methode als regel te bewerkstelligen dat het oude bestandsrecord met nieuwe sleutel als transactierecord uitgeponst wordt en in de volgende mutatiegang weer wordt ingevoerd. Als van vrijwel het hele bestand de sleutels moeten veranderen kunnen bestandsrecords direkt met de nieuwe sleutel weggeschreven worden; na afloop van de bestandsmutatie moet dan het bestand opnieuw gesorteerd worden.
- de procedure read record moet inhouden dat na het lezen van het laatste record de sleutel op ∞ (in werkelijkheid natuurlijk de grootst mogelijke sleutel met het betreffende aantal sleutelposities) wordt gezet, teneinde afloop van het mutatieproces te bewerkstelligen. In veel gevallen is het verstandig om, zoals gezegd, de procedure read record uit te breiden met de controle dat de sleutel van het laatst gelezen record niet kleiner is dan dat van het voorlaatst gelezen record (aannemend dat het bestand in opklimmende sleutelvolgorde gesorteerd is). Een misplaatst record kan immers een groot aantal foutsignaleringen geven (ga dit na), zodat het meestal niet zinvol is om het mutatieproces voort te zetten.
- afhankelijk van de omstandigheden kan het nuttig zijn om de laatste records van transactiebestand en bestand te gebruiken voor controletotalen. De bewerking van de controlerecords moet dan gebeuren in de niet uitgewerkte procedure END, terwijl tijdens de mutatiebewerking de corresponderende controletotalen moeten worden opgebouwd.

Opgave

Onderzoek wat men met controletotalen moet doen bij sleutelveranderingen!

Opmerking

In het voorgaande is verschillende keren gesproken over controle's en controletotalen. Voor ieder probleem moet onderzocht worden welke controle's en controletotalen zinvol en op welke plaatsen ingevoerd kunnen worden teneinde te voorkomen dat een bestand "verontreinigd" raakt. Voor 100% is dit zelden te garanderen, zodat steeds onderzocht moet worden welke graad van controle te verantwoorden is, gezien de kosten van zo'n controle en de kosten van "zuivering"van een verontreinigd geraakt bestand. Controle's en controletotalen zijn meestal met weinig moeite en geringe kosten aan te brengen op punten waar gegevens een informatiesysteem binnenkomen. Controletotalen geven dan een redelijke garantie dat een partij mutatie's goed verwerkt is (invoering van duplo's, verloren transacties en optreden van hardware fouten worden hiermee meestal wel ontdekt), maar geven geen garantie t.a.v. fouten die ook al in de controletotalen verwerkt zijn. Ten aanzien van de laatste categorie fouten kunnen plausibiliteitscontroles soms uitkomst bieden, of "dubbele" controle's (bijv. invoeren van naam èn registratienummer in plaats van één van deze gegevens alleen).

Opgave

De hiervoor behandelde programmalogica is ontwikkeld voor magneetbanden als informatiedragers. Onderzoek welke veranderingen mogelijk zijn bij gebruik van schijven of trommels als informatiedragers en onder welke omstandigheden.

Opgave

Hoe zou het mutatieproces voor een over twee informatiedragers verdeeld bestand georganiseerd moeten worden, wanneer t.a.v. de records niet bekend is of zij op het eerste of op een vervolg bestand staan. Wanneer het eerste bestand high-activity records bevat, hoe is dan te bewerkstelligen dat vrijwel automatisch records op de goede wijze over de informatiedragers verdeeld blijven?

2.5. Willekeurig toegankelijke structuren

Bij gebruik van deze structuren is de opzet niet om records in een bepaalde volgorde te plaatsen (dit kan toevallig weleens zo zijn), maar om bij een bepaalde sleutel "direct" het betreffende record te vinden en om aanwezige records enigszins gelijkmatig over de beschikbare geheugenruimte te verdelen. Er moet bij willekeurig toegankelijke opslagstructuren dus een relatie zijn tussen recordsleutel en recordadres (de fysieke plaats van het record). Deze relatie kan verschillende vormen hebben:

directe relatie tussen recordsleutel en recordadres in die zin dat ze hetzij identiek zijn, hetzij een constant bedrag verschillen. Om deze methode met vrucht te gebruiken is nodig dat

- de omvang van het bestand de capaciteit van de informatiedrager niet overschrijdt,
- de recordsleutel eenvoudigheidshalve numeriek is (een alfanumerieke sleutel kan echter als een getal in een 36-tallig stelsel opgevat worden),
- de in de praktijk voorkomende sleutelwaarden gelijkmatig en dicht verdeeld zijn over het traject bepaald door het verschil tussen de grootst mogelijke en kleinst mogelijke sleutel. Wordt aan deze voorwaarde niet voldaan, dan is de bezettingsgraad van de informatiedrager laag, hetgeen de economische verantwoording moeilijk maakt.

tabel relatie (ook wel relatieve organisatie genoemd) tussen recordsleutel en recordadres, bijvoorbeeld doordat in de linkerkolom van een tabel de aanwezige recordsleutels (voor het zoeken meestal in volgorde) staan en in de rechterkolom de bijbehorende recordadressen (telefoongids gedachte). Eventueel wordt deze relatie tot stand gebracht door van meer tabellen gebruik te maken: in een hoofdtabel staat vermeld in welke subtabel men moet zoeken voor een bepaalde groep recordsleutels (vergelijk het systeem van alle Nederlandse telefoongidsen samen). Het voordeel van dit systeem, boven dat met een directe relatie, is naast een gunstige bezettingsgraad dat de laatste daar genoemde beperking vervalt; het nadeel uiteraard dat een tabel geraadpleegd en bijgehouden moet worden. Het bestand zelf hoeft echter vrijwel nooit gereorganiseerd te worden (tenzij een zeer groot aantal records moet vervallen).

functionele relatie tussen recordsleutel en recordadres, d.w.z. het recordadres wordt op de een of andere wijze uit de recordsleutel berekend (de elementaire rekenbewerkingen vervat in de directe en tabelrelatie zijn hiervan dus eigenlijk bijzondere gevallen). De functionele relatie moet zo gekozen worden dat bij een gegeven verzameling oorspronkelijke sleutels de bezettingsgraad van de informatiedrager economisch aanvaardbaar is en dat ook niet teveel last wordt ondervonden van "synoniemen". Daar de ruimte van potentieel voorkomende sleutels afgebeeld wordt op de vrijwel altijd kleinere ruimte van beschikbare recordadressen, moet immers een voorziening worden getroffen ("overloop") voor de mogelijkheid dat uit twee verschillende sleutels (de synoniemen) hetzelfde recordadres volgt. Als regel berust deze voorziening op een van de volgende methoden:

- buiten het in eerste instantie ter beschikking gestelde gebied wordt een "overloop" gebied gereserveerd, waarin de records die geen plaats meer kunnen vinden in het "primaire" gebied, (meestal) sequentieel worden ondergebracht. Desgewenst koppelt men synoniemen in primair en overloopgebied met een kettingstructuur,
- een overlooprecord wordt ondergebracht in de eerstvolgende vrije locatie, volgende op het recordadres dat in eerste instantie berekend was. Ook hier kan men (ter wille van een snel terugzoeken) bovendien nog een kettingstructuur gebruiken,
- men reserveert voor iedere getransformeerde sleutelwaarde niet één maar een vast, klein aantal consecutieve recordadressen ("bin" of "bucket"), waarin eventuele synoniemen opgeslagen worden. Voor noodgevallen is ook hier nog een overloopgebied nodig.

Bij gebruik van een functionele relatie, aanleiding gevend tot wat veelal een random georganiseerd bestand genoemd wordt, is het in tegenstelling tot de eerste twee willekeurig toegankelijke bestandsvormen, niet mogelijk om het bestand sequentieel te verwerken (tenminste niet zonder het hele bestand eerst op een sequentiële informatiedrager te copiëren en vervolgens te sorteren).

2.5.1. Sleutelconversie

Vat men het fysieke recordadres op als "nieuwe" sleutel, dan kan de functionele relatie tussen recordadres en recordsleutel beschouwd worden als een sleutelconversie.

De oorspronkelijke recordsleutels zijn alle verschillend, maar de verdeling van sleutels over het beschikbare gebied is als regel alles behalve uniform. Gaten en ophopingen ontstaan immers door de manier waarop de sleutels opgebouwd zijn, vaak al lang voordat over het efficiënt gebruik van informatiedragers gedacht werd. Meestal volgt een sleutelwaarde uit classificatie-overwegingen, waarbij aan bepaalde digits een bepaalde betekenis werd toegekend. Door de aard van de te classificeren grootheden kunnen dan bepaalde digitcombinaties niet voorkomen ("gaten"), terwijl andere digitcombinaties wellicht vaak voorkomen ("ophopingen"). De te kiezen sleutelconversiemethode moet zo zijn dat ter wille van een minimum overflow en een optimale bezettingsgraad de geconverteerde sleutels zo homogeen mogelijk verdeeld zijn.

Men kan zich indenken dat zo'n sleutelconversiemethode niet eenvoudig te vinden is voor een éénmaal historisch gegroeid sleutelsysteem, waarvan de statistische eigenschappen vaak niet eens direcht bekend zijn. Evenzo is te verwachten dat sommige sleutelverdelingen bijzonder gevoelig zijn voor een bepaalde conversiemethode, zeker gevoeliger dan een random sleutelverdeling. De laatste komt echter omdat er geen classificerend element in zit in de praktijk zelden voor (toevallig wel bij de in ontwikkeling zijnde landelijke bevolkingsadministratie).

Vele sleutelconversiemethoden zijn ontwikkeld in de loop der tijd:

- "truncation", d.w.z. het weglaten van (meestal de voorste) digits van een sleutel en het gebruik van de overblijvende digits als nieuwe sleutel. Door de classificerende eigenschappen van een sleutel is deze methode als regel, ondanks zijn eenvoud, niet aan te raden zonder de oorspronkelijke sleutelverzameling zorgvuldig te onderzoeken; de straf kan immers een sterke ophoping om bepaalde door conversie ontstane sleutelwaarden zijn.
- "extraction", d.w.z. bepaalde digitposities van de oorspronkelijke sleutel worden achter elkaar geplaatst als nieuwe sleutel opgevat (als regel zal men digitposities waarin sterke ophopingen van bepaalde digits voorkomen hierbij uiteraard niet meenemen). Ook deze methode mag pas na een zorgvuldig onderzoek van de oorspronkelijke sleutelverdeling eventueel gehanteerd worden.

- "folding" is meestal een beter proces dan truncation omdat hierbij in ieder geval de volledige oorspronkelijke sleutel gebruikt wordt. Deze wordt nl. gesplitst in 2 of meer stukken die dan bij elkaar opgeteld worden, bijv. voor folding van 6 naar 2 digits:

  12 34 56 → 02
  63 72 88 → 23 .

  Opm. Soms wordt onder folding ook verstaan het berekenen van de nieuwe sleutel door middel van een arithmetische relatie tussen sleuteldelen. Folding kan goed werken, maar heeft enigszins een hocus-pocus-karakter en vereist eveneens een statistisch onderzoek van de oorspronkelijke distributie van sleutels.

- "mid-squaring" is een met voorzichtigheid te hanteren methode om random getallen te berekenen: een sleutel van n digits wordt met zichzelf vermenigvuldigd waarna bijv. de middelste n digits (eventueel gevolgd door truncation) als nieuwe sleutel gehanteerd wordt. Mid-squaring produceert echter vaak veel nullen en is in het algemeen daarom niet aan te raden.
- "radix-conversion" werd vroeger, toen deelbewerkingen tamelijk langzaam waren, op een computer nog wel eens gebruikt, bijv. voor radix 10 → radix 11 conversie gevolgd door truncatie

  $$12\ \ 34 \rightarrow 1 \times 11^3 + 2 \times 11^2 + 3 \times 11^1 + 4 = 1610 \rightarrow 10 .$$

- "prime division" wordt als niets bekend is van de oorspronkelijke sleuteldistributie als de veiligste methode beschouwd. Wanneer n adressen beschikbaar zijn (dus n verschillende nieuwe sleutels mogelijk zijn), dan kan de rest van de deling van de oorspronkelijke sleutel door n als nieuwe sleutel gebruikt worden. Een eigenschap van deling is dat n opeenvolgende originele sleutels ook n verschillende opeenvolgende getransformeerde sleutels geven. Dit is ook nog waar als de oorspronkelijke sleutels telkens een bedrag d van elkaar verschillen, mits d en n geen gemeenschappelijke delers bezitten. Bijvoorbeeld voor d = 3 en n = 7 :

  oude sleutels : 20, 23, 26, 29, 32, 35, 38, 41, enz.
  nieuwe sleutels : 6, 2, 5, 1, 4, 0, 3, 6, enz.

  (Algemeen: twee oude sleutels $a+k_1d$ en $a+k_2d$ geven dezelfde nieuwe sleutel als $|k_1-k_2|$ deelbaar is door n.)

In de praktijk zullen (oude) sleutels wel niet equidistant verdeeld zijn, maar toch ligt het voor de hand om voor de deler een priemgetal te gebruiken en wel (teneinde de beschikbare nieuwe ruimte zo goed mogelijk te gebruiken) het grootste priemgetal dat net nog kleiner is dan het aantal beschikbare nieuwe sleutelwaarden.
Priemgetaldeling is wel de veiligste sleutelconversiemethode, wanneer men geen tijd heeft voor een analyse van oorspronkelijke sleutels, maar natuurlijk niet noodzakelijk de beste methode voor iedere sleutelverdeling. Evenmin zijn alle priemgetallen even goed te gebruiken; i.h.b. kunnen priemgetallen van de vorm $a \cdot 10^b \pm 1$ voor kleine a waarden beter vermeden worden, dus priemgetallen als 11, 19, 29, 31, 41, 101, 199, 401, 499, 599, 601, 701, 1999, 2999, 3001, 4001, 4999, 7001, 8999, 9001, 49999, 59999, 70001, 79999, 90001, enz. Uit de ontwikkeling $(p \pm 1)^{-1} = 1/p \pm 1/p^2 \pm 1/p^3 \ldots$ zien we met $p = 10^b$ dat gebruik van deze priemgetallen neerkomt op optelling van p-digit groepjes van de oorspronkelijke sleutel, d.w.z. op folding. Uit ervaring is bekend dat folding echter meestal minder goed is dan prime-division. Ook als $a \neq 1$ is komt gebruik van deze priemgetallen vrijwel op folding neer.

2.5.2. Gebruik van sleutelconversie bij schijfsystemen

Varianten van de priemgetalmethode worden vaak toegepast bij gebruik van schijfsystemen. We zullen twee varianten toelichten aan een gesimplificeerd schijfsysteem met slechts 1 cylinder, verder 4 primaire sporen en 1 overloopspoor op deze cylinder en ruimte voor 4 records per spoor. Laten er verder gemiddeld niet meer dan 12 records aanwezig zijn in het bestand. Het te kiezen priemgetal is, daar er in de primaire sporen ruimte is voor 16 records, natuurlijk 13. Laten we verder aannemen dat records worden aangeboden met de volgende sleutels: 17, 16, 14, 9, 19, 7, 6, 4, 12, 25, 11, 26.

a) methode met alleen spoorberekening:
   - bepaal eerst de rest van de deling van de sleutel door 13
   - deel dan deze rest door 4 (d.i. het aantal records per spoor), gebruik het quotiënt als spoornummer
   - probeer dan zo veel mogelijk de primaire sporen op te vullen en gebruik het laatste spoor voor de overflow.

Door narekenen is dan is verifiëren dat we de volgende vulling krijgen voor de aangeboden sleutels:

spoor 0 : 16, 14, 26
spoor 1 : 17, 19, 7, 6
spoor 2 : 9, 11
spoor 3 : 12, 25
(overloop-) spoor 4 : 4

b) methode met spoor + adresberekening:
deze is gelijk aan de vorige methode, doch bovendien wordt de rest na deling door 4 (het aantal records per spoor) gebruikt als adres (begin bij 0 te tellen) in het reeds bepaalde spoor.
Wederom door narekenen is de volgende vulling te verifiëren voor de aangeboden sleutels:

spoor 0 : 26, 14, , 16
spoor 1 : 17, , 19, 7
spoor 2 : , 9, , 11
spoor 3 : 12, , ,
(overloop-) spoor 4 : 6, 4, 25, (ongestructureerd gevuld) .

Opgave

Verifieer de volgende vullingen als we in plaats van 13 het getal 15 gekozen hadden als eerste deler:

	a)	b)
spoor 0 :	17, 16	, 16, 17
spoor 1 :	19, 7, 6, 4	19, , 6, 7
spoor 2 :	9, 25, 11, 26	9, , 25
spoor 3 :	14, 12	14,
spoor 4 :		4, 12, 11, 26 (!) .

Methode a) heeft vergeleken met b) het nadeel dat een record door sequentiële inspectie in een spoor gezocht moet worden (en eventueel ook nog in het overloopspoor, bijv. record 4), maar het voordeel dat het overloopspoor minder gauw vol zal raken. Om zoektijd in het overloopspoor te verminderen, zal men in variant b soms een kettingstructuur in het overloopspoor gebruiken en in variant a soms in primaire en overloopspoor.

2.5.3. Overflowproblematiek

Bij iedere sleutelconversiemethode zal overflow in het algemeen onvermijdelijk zijn, zodat het er om gaat om de nadelen van overflow zo klein mogelijk te maken. Nu kan overflow zich voordoen bij het formeren of aanvullen van een bestand, maar ook bij het zoeken van een record in een bestand. Is het aantal raadplegingen van een bestand veel groter dan het aantal veranderingen, dan zal men een methode zoeken waarbij het zoeken zo snel mogelijk is, maar het formeren of aanvullen van een bestand desnoods een flinke tijd kan vergen (zoals bijvoorbeeld met een kettingstructuur het geval is. De variant b uit het voorgaande is voor het zoeken bijvoorbeeld ook gunstiger dan variant a).

Helaas zijn in de praktijk de aantallen raadplegingen en veranderingen vaak nog niet bekend op het moment dat een bepaalde bestandsorganisatie gekozen moet worden, zodat over verfijningen als wel dan niet kettingstructuren of wel dan niet sequentieel opvullen van sporen moeilijk te beslissen is.

In het volgende zullen we bekijken hoe in het bijzonder voor schijfsystemen ook zonder de net genoemde verfijningen het nadeel van overflow verminderd kan worden. We kunnen hierbij onderscheid maken tussen methoden die de kans op een overflow zo klein mogelijk maken en de eerst te bespreken strategie om het nadelig effect van een onvermijdelijke overflow zo klein mogelijk te maken.

Bij schijven dient men om te beginnen een armbeweging te vermijden, zodat men overflowsporen in ieder geval in dezelfde cylinder moet plaatsen als de primaire sporen. Een overflow vereist dan nog aan extra tijd een halve omwentelingstijd (en de electronische schakeltijd van een andere leeskop, welke echter meestal te verwaarlozen is). Wil men ook nog van de halve omwentelingstijd afkomen dan is, aannemend dat apparatuur en programmatuur dat zinvol maken, te overwegen om de overflow in de eerste plaats onder te brengen aan het "eind" van ieder primair spoor. Een extra overloopspoor zal men toch moeten handhaven omdat de ruimte aan het eind van ieder spoor, uiteraard zo klein mogelijk gekozen, wel eens vol kan raken. Waar men, wederom aannemend dat apparatuur en programmatuur dat zinvol maken, ook nog over kan denken is om een overflow niet aan het eind van een spoor te plaatsen, maar in de eerste de beste vrije locatie op een spoor die beschikbaar komt. Tenzij de bezettingsgraad van een spoor betrekkelijk laag is, geeft deze laatste methode wel een zeer rommelige bestandsorganisatie en gemiddeld een langere zoektijd van een record bij een redelijke bezettingsgraad.

De kans op overflow kan men tenslotte verminderen door bij iedere waarde van de geconverteerde sleutel niet, zoals in het voorgaande verondersteld, één maar twee (of zelfs meer, meest opeenvolgende) recordruimten te reserveren. Pas wanneer zo'n "multiple-record bin" of "bucket" helemaal met synoniemen gevuld is, moet naar een apart overflowgebied uitgeweken worden (of in de volgende bucket). Naast het kleine bezwaar dat een bepaald record toch nog even sequentieel in een bucket gezocht moet worden, geldt dat wanneer de getransformeerde sleutels niet gelijkmatig verdeeld zijn, de meeste bins slecht gevuld zijn en enkele overbezet rakende bins toch een overflow behandeling vergen.

Dat bij gelijkmatig verdeelde geconverteerde sleutels bins gunstiger zijn dan enkele recordruimten is wel aan te voelen wanneer tenminste de zoektijd in een bin veel korter is dan de toegangstijd tot een bin. Bij veel schijfsystemen en trommels is dit het geval omdat de toegangstijd tot een bin dan ongeveer een halve omwentelingstijd is (dus van de orde van millisec.), terwijl de zoektijd in een bin (in het kerngeheugen) een kwestie van microsec is.

Het is van belang te schatten wat de te verwachten overflow is.
We nemen ter berekening van het aantal records dat in het primaire gebied in de bins geen plaats kan vinden, het volgende aan:

	geheugencapaciteit	: g recordplaatsen
	bincapaciteit	: b recordplaatsen/bin
	vullingsgraad geheugen	: f .
Dan is:	aantal records	: fg = n
	aantal bins	: g/b
	vullingsgraad per bin	: fg/(g/b) = fb = μ (zonder overflow)

en, wanneer de geconverteerde sleutels homogeen verdeeld zijn, de kans dat een record in een bepaalde bin terecht komt 1/(aantal bins) = b/g. De kans dat er precies n records in een bepaalde bin terecht komen wordt dan gegeven door de binomiaalverdeling

$$p_n = \binom{m}{n}(b/g)^n\ (1 - b/g)^{m-n} = \binom{m}{n}(\mu/fg)^n\ (1 - \mu/fg)^{m-n}\ .$$

Als we de geheugencapaciteit g naar ∞ laten gaan, gaat ook m = fg naar ∞, evenals het aantal bins, maar nemen we daarbij aan dat de vullingsgraad per bin μ constant blijft, dan krijgen we met

$$p_n = \frac{(fg)!}{n!(fg-n)!} \frac{\mu^n}{(fg)^n} \left(1 - \frac{\mu}{fg}\right)^{fg} \left(1 - \frac{\mu}{fg}\right)^{-n}$$

$$= \frac{\mu^n}{n!} \left(1 - \frac{\mu}{fg}\right)^{fg} \left(\frac{fg}{fg} \cdot \frac{fg-1}{fg} \cdots \frac{fg-n+1}{fg}\right) \left(1 - \frac{\mu}{fg}\right)^{-n} \simeq \frac{\mu^n}{n!} e^{-\mu}$$

de Poissonverdeling als benadering voor de binomiaalverdeling, welke zoals bekend zelfs voor kleine m en n waarden nog heel goed is.
Voor deze Poissonverdeling geldt

$$\sum_0^\infty np_n = \mu \quad \text{en} \quad \sum_0^\infty p_n = 1 .$$

Gezien de bincapaciteit wordt de fractie van de records (met dezelfde geconverteerde sleutel), die geen plaats kunnen vinden in een bepaalde bin gegeven door

$$\alpha_b = \frac{1}{\mu} \sum_{n=b+1}^{\infty} (n-b)\, p_n = \frac{1}{\mu} \left(\mu - \sum_0^b np_n\right) - \frac{b}{\mu} \left(1 - \sum_0^b p_n\right) .$$

Dan is

$$\alpha_{b-1} = \frac{1}{\mu} \left(\mu - \sum_0^{b-1} np_n\right) - \frac{b-1}{\mu} \left(1 - \sum_0^{b-1} p_n\right)$$

en dus

$$\alpha_b - \alpha_{b-1} = -\frac{bp_b}{\mu} + \frac{b}{\mu} p_b - \frac{1}{\mu} \left(1 - \sum_0^{b-1} p_n\right) = -\frac{1}{\mu} \left(1 - \sum_0^{b-1} p_n\right) \qquad b = 1,2,\ldots$$

en

$$\alpha_0 = \frac{1}{\mu} \sum_1^\infty np_n = \frac{1}{\mu} (\mu) = 1 \quad \text{(zoals te verwachten).}$$

Hiermee is het overflowpercentage α_b te berekenen als functie van b en f, hetgeen dan de volgende tabel geeft (ontleend aan W. Buchholz, IBM Systems J. (1963) 99).

bin grootte	vullingsgraad 0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	1.1	1.2
1	4.84	9.37	13.61	17.58	21.31	24.80	28.08	31.17	34.06	36.79	39.35	41.77
2	0.60	2.19	4.49	7.27	10.36	13.65	17.03	20.43	23.79	27.07	30.24	33.30
3	0.09	0.63	1.80	3.61	5.99	8.82	11.99	15.37	18.87	22.40	25.91	29.33
4	0.02	0.20	0.79	1.96	3.76	6.15	9.05	12.32	15.86	19.54	23.25	26.93
5		0.07	0.37	1.12	2.48	4.49	7.11	10.26	13.78	17.55	21.42	25.30
6		0.02	0.18	0.67	1.69	3.38	5.75	8.75	12.24	16.06	20.06	24.11
7		0.01	0.09	0.41	1.18	2.60	4.74	7.60	11.04	14.90	19.00	23.19
8			0.05	0.25	0.84	2.03	3.97	6.68	10.07	13.96	18.15	22.46
9			0.02	0.16	0.61	1.61	3.36	5.94	9.27	13.18	17.44	21.86
10			0.01	0.10	0.44	1.29	2.88	5.32	8.59	12.51	16.85	21.36
11			0.01	0.07	0.33	1.04	2.48	4.80	8.01	11.94	16.34	20.94
12				0.04	0.24	0.85	2.15	4.36	7.51	11.44	15.89	20.58
14				0.02	0.14	0.57	1.65	3.64	6.67	10.60	15.15	19.99
16				0.01	0.08	0.39	1.28	3.09	6.00	9.92	14.56	19.53
18					0.05	0.28	1.01	2.65	5.45	9.36	14.07	19.16
20					0.03	0.20	0.81	2.30	4.99	8.88	13.66	18.86
25					0.01	0.09	0.48	1.65	4.10	7.95	12.87	18.31
30						0.04	0.29	1.23	3.47	7.26	12.31	17.93
35						0.02	0.18	0.94	2.98	6.73	11.87	17.66
40						0.01	0.12	0.73	2.60	6.29	11.53	17.47
50							0.05	0.45	2.04	5.63	11.03	17.20
60							0.02	0.30	1.65	5.14	10.68	17.03
70							0.01	0.20	1.37	4.76	10.41	16.93
80							0.01	0.13	1.14	4.46	10.21	16.86
90								0.09	0.97	4.20	10.05	16.80
100								0.06	0.83	3.99	9.92	16.77

Een vullingsgraad groter dan 1 betekent dat de overflow in een apart gebied moet worden opgeborgen omdat het primaire gebied dan te klein is om de records op te nemen. Uit de tabel is ook af te lezen dat een flinke bingrootte voordeliger is dan een groot aantal kleine bins (vergelijk bij vrijwel hetzelfde overflowpercentage van ongeveer 17.5 % een bingrootte van 5 records met een vullingsgraad van 1.0 en een bingrootte van 1 record met een vullingsgraad van 0.4).

Een ander voorbeeld van gebruik van deze tabel is het volgende:
laat een bestand van 4000 records opgeborgen moeten worden op een schijfsysteem met 10 sporen per cylinder en 10 records per spoor. Hoeveel cylinders zijn dan nodig als de gewenste vullingsgraad 80 % is?
Reserveren we als eerste poging per cylinder 9 primaire sporen en 1 overloopspoor dan kunnen in de primaire sporen 0.8 × 9 × 10 = 72 records een plaats vinden. Het aantal overflow records (als functie van de bingrootte) bij deze 72 records berekenen we als volgt:

bingrootte	2	3	4	5	6	7	8
overflow %	20.4	15.4	12.3	10.3	8.8	7.6	6.7
overflow aantal	~ 15	~ 11	~ 9	~ 7	~ 7	~ 6	~ 5

We zien hieruit dat vanaf bingrootte 4 dus 9 of minder records in de overflow terecht komen, wat nog in het 10^{e} spoor kan. Bij bingrootte 4 hebben we dan 4000/72+1 extra overflow cylinder = 57 cylinders nodig.

2.5.4. Het zoeken en muteren in een willekeurig toegankelijke structuur

Bij een directe relatie tussen recordsleutel en recordadres is de techniek voor het vinden van records triviaal; het record is in principe in één accesstijd beschikbaar.

Bij een tabelrelatie is de zoektechniek uiteraard dat in de tabel (en eventueel de subtabellen) naar de recordsleutel gezocht wordt, waarna het recordadres bekend is. De zoektijd in de tabellen is gemiddeld van de orde n/2 microseconden als de tabel voor n records al in het kernengeheugen aanwezig is, maar van de orde van de accesstijd (milliseconden) als de tabel nog op een secundair geheugensysteem staat. Om het record zelf in het kernengeheugen te krijgen moet één accesstijd opgeteld worden. De zoektijd in de tabel, hoewel reeds kort, kan nog wat verbeterd worden door de recordsleutels in de tabel te sorteren en dan de binaire zoekmethode te gebruiken. (Tegenover een korte zoektijd staat bij de gesorteerde tabel uiteraard een langere opbouwtijd van

de tabel omdat daarin de sleutels gesorteerd moeten worden. Het aantal keren dat de tabel opgebouwd moet worden versus het aantal keren dat de tabel voor een zoekproces gebruikt moet worden, bepaalt dan de keus tussen gesorteerde en ongesorteerde tabellen.)

Bij een functionele relatie tussen recordsleutel en recordadres moet voor het zoeken van een record uiteraard eerst een sleutelconversie worden uitgevoerd met de sleutel als gegeven. Bij een geschikte "randomizing" methode zal slechts nu en dan een synoniem optreden, zodat aangezien de berekeningstijd meestal te verwaarlozen is, gemiddeld in (iets meer dan) één accesstijd een record beschikbaar zal zijn. Is het aantal synoniemen daarentegen niet te verwaarlozen, dan kan voor het zoeken eventueel een groot aantal accessen nodig zijn.

Samenvattende kan gezegd worden dat het probleem meestal zit in het vinden van een goede "randomizer", doch dat daarna niet alleen het zoeken, maar ook het toevoegen, veranderen of verwijderen van records weinig problemen geeft en snel is. Omdat iedere zoek ook onafhankelijk van de vorige is, hoeven ook de transacties niet gesorteerd aangebracht te worden. Deze bestandsstructuur is dan ook zeer geschikt voor "on-line" toepassingen, al zullen zoals reeds eerder gezegd door het systeem voorzieningen getroffen moeten worden om "bijna gelijktijdige" mutaties op een bepaald record goed te verwerken.

2.6. Index-sequentiële structuren

Een bekende tussenvorm van een sequentiële en een willekeurig toegankelijke structuur is de zg. index-sequentiële organisatie, waarbij met behulp van een "indextabel" de relatie tussen recordsleutel en recordadres is vastgelegd. Bij veel toepassingen heeft men namelijk behoefte aan een structuur waarbij:

- records willekeurig toegankelijk zijn voor postgewijze verwerking (magneetbanden zijn niet bruikbaar voor deze structuur);
- anderzijds vaak behoefte is om alle records in sleutelvolgorde te verwerken.

Aangezien zodoende alleen magneetschijven (of schijfachtige systemen) in aanmerking komen, is voor dat soort toepassingen een organisatie ontwikkeld die aangepast is aan de eigenschappen van een schijf. Met name aan het feit dat bij een bepaalde stand van de schijfarm vele sporen op de schijven snel uitgelezen kunnen worden (de "cylinder"), maar dat een verandering van de stand van de schijfarm relatief lang duurt.

2.6.1. Principe van de index-sequentiële organisatie

De index-sequentiële organisatie komt hierop neer dat in het kernengeheugen in een zg. "cylindertabel" afgelezen kan worden wat voor iedere cylinder de hoogst voorkomende sleutel is. De cylindertabel is op sleutelvolgorde gesorteerd zodat (bijv. met een binaire zoekmethode) snel gevonden kan worden in welke cylinder een record met een bepaalde sleutel zich kan bevinden. (Hoewel niet strikt noodzakelijk zal men grote hoeveelheden mutaties op zo'n bestand liefst gesorteerd verwerken, omdat dan het aantal armbewegingen minimaal is!)

Van een vast spoor op iedere cylinder wordt dan de zg. sporentabel ("tracktable") in het kernengeheugen gelezen. Hierin staat, weer in sleutelvolgorde, wat in ieder spoor de hoogst voorkomende sleutel is. Na met een zoekmethode zo voor een bepaalde recordsleutel vastgelegd te hebben in welk spoor zich het betreffende record kan bevinden, moet de inhoud van dit spoor naar het kernengeheugen gebracht worden. (We nemen hierbij dus stilzwijgend aan dat er in het kernengeheugen voldoende ruimte is om een kompleet spoor in te lezen; is dit niet het geval dan moeten in de sporentabel ook de nummers van de spoordelen opgenomen worden.) Met een (korte) sequentiële zoek kan dan in het spoor het goede record gevonden worden.

Samenhangend met het feit dat in een gegeven bestand records toegevoegd of verwijderd worden, bestaan voor de implementatie van deze bestandsvorm (die in ieder geval langzamer is dan de random structuur) verschillende varianten:

- de simpelste vorm, alleen te gebruiken als (nauwelijks) records worden verwijderd of toegevoegd in bestand, is met de gesorteerde records een bestand en bijbehorende tabellen op te bouwen op de beschikbare sporen in een cylinder en zo nodig in de volgende cylinder(s);
- een variant is dat men bij de eerste opbouw van het bestand de sporen van een cylinder volledig met records vult, behalve één of enkele overloopsporen van die cylinder. Moet later bij een mutatie een record tussengevoegd worden, dan gebeurt dit na op het goede spoor de aanwezige records een plaats opgeschoven te hebben. Het record dat zodoende van het "primaire spoor afvalt", wordt dan op de eerstvolgende vrije plaats van een overloopspoor geplaatst. Een nadeel van deze methode is dat in het kernengeheugen geschoven en in het overloopgebied sequentieel gezocht moet worden, en dat de sporentabel moet worden bijgewerkt na iedere toevoeging;

. een andere variant beoogt het beschrijven van een overloopspoor zolang mogelijk uit te stellen door in eerste instantie de primaire sporen niet volledig te vullen. Pas wanneer de vrije ruimte op een spoor opgebruikt is, moeten overloopsporen gebruikt worden. Aangezien dit gebruik van vrije ruimte als regel niet gelijkmatig is voor alle sporen, is deze variant niet gunstig t.a.v. het geheugengebruik. Als bij de vorige variant moet mogelijk de sporentabel bijgewerkt worden;
. in nog weer een andere variant worden de sporen weer volledig gevuld, maar bovendien bevat ieder record in een primair spoor zo nodig een wijzer naar het begin van een gesorteerde ketting in een overloopspoor. Hiermee is het opschuiven van records in een primair spoor overbodig en is de zoek in een overloopspoor efficiënter dan zonder ketting. Het bijwerken van de sporentabel is meestal niet nodig. (Soms wordt het spoor waarop de sporentabel staat nog als extra overflowruimte gebruikt; soms wordt naast de overflowsporen nog een onafhankelijke overflowcylinder gebruikt voor noodgevallen. Het laatste vergroot natuurlijk wel de gemiddelde accesstijd van een record!)

Vergelijking van de gewone sequentiële en index-sequentiële methoden geeft het volgende overzicht:

	sequentieel	index-sequentieel
record toevoegen	hele file bijwerken	slechts deel file bijwerken
record verwijderen	hele file doorwerken	record kenmerken
record veranderen	hele file doorwerken	betreffende record bijwerken
zoeken van een record	halve file doorwerken	vrijwel random
sorteren transacties nodig	ja	neen (mag wel)
bij lage file activity	neen	ja
bij hoge file activity	ja	eventueel
bij lage file volatility	neen	ja
bij hoge file volatility	eventueel	eventueel
bij lage file turnover	neen	ja
bij hoge file turnover	eventueel	eventueel.

2.6.2. Mutatie verwerking bij een index-sequentiële organisatie

Het veranderen van records in een index-sequentieel georganiseerd bestand is op het eerste gezicht een eenvoudige bewerking: een record kan direct opgespoord worden met de sleutel van het transactierecord, veranderd en dan weer teruggeschreven. Omdat mogelijk meerdere mutaties op een oud record moeten worden aangebracht is het bij een hoge file volatility efficiënter om de transactierecords toch te sorteren op sleutelvolgorde. Zoals reeds besproken bij sequentieel georganiseerde bestanden is oppassen geboden ten aanzien van vermeerdering of vermindering van items, wanneer er een kans is op tussentijds afbreken van het mutatieproces. Hetzelfde geldt voor eventuele controle-totalen.

Het toevoegen van records in een index-sequentieel bestand is daarentegen niet een simpele bewerking omdat veelal de indextabel(len) bijgewerkt moet(en) worden. Vooral wanneer de toe te voegen records in het overloopgebied terecht komen, moet enige administratie in de tabellen en/of in kettingwijzers in de records in het overloopgebied uitgevoerd worden. Als het werkgeheugen te klein is om complete sporen van een willekeurig toegankelijk geheugensysteem op te nemen, kunnen ingewikkelde programma's hiervan het gevolg zijn.

Ook het verwijderen van records kan aanleiding geven tot het bijhouden van indextabel(len), maar vaak kiest men de oplossing dat een overbodig geworden record als zodanig "gemerkt", doch niet verwijderd wordt.

Wanneer na enige tijd grote aantallen records in overloopgebieden terecht zijn gekomen, en/of overbodig geworden records flink in aantal zijn toegenomen, dan wel een overloopgebied vol is geraakt, moet men met het oog op toegenomen verwerkingstijden overgaan tot een bestandsreorganisatie. Dit gebeurt door het totale bestand in sleutelvolgorde te copiëren op een nieuwe schijveneenheid, en daarbij overbodige records te verwijderen en overloopgebieden leeg te maken. Bij deze bewerking zal veelal tevens een copie van het bestand op magneetband vastgelegd worden ten behoeve van bestandsbeveiliging.

Het voorgaande zullen we toelichten aan de hand van een index-sequentieel systeem met de volgende kenmerken:

- in een cylindertabel worden voor iedere cylinder de laagste en hoogste in de cylinder voorkomende recordsleutel bijgehouden;
- in een sporentabel in iedere cylinder worden laagste en hoogste recordsleutel voor ieder spoor bijgehouden;

. iedere cylinder heeft zijn eigen overloopspoor; de primaire sporen zijn volledig gevuld;
. voor overlooprecords wordt met een kettingorganisatie gewerkt;
. grote aantallen transacties moeten in batch verwerkt worden.

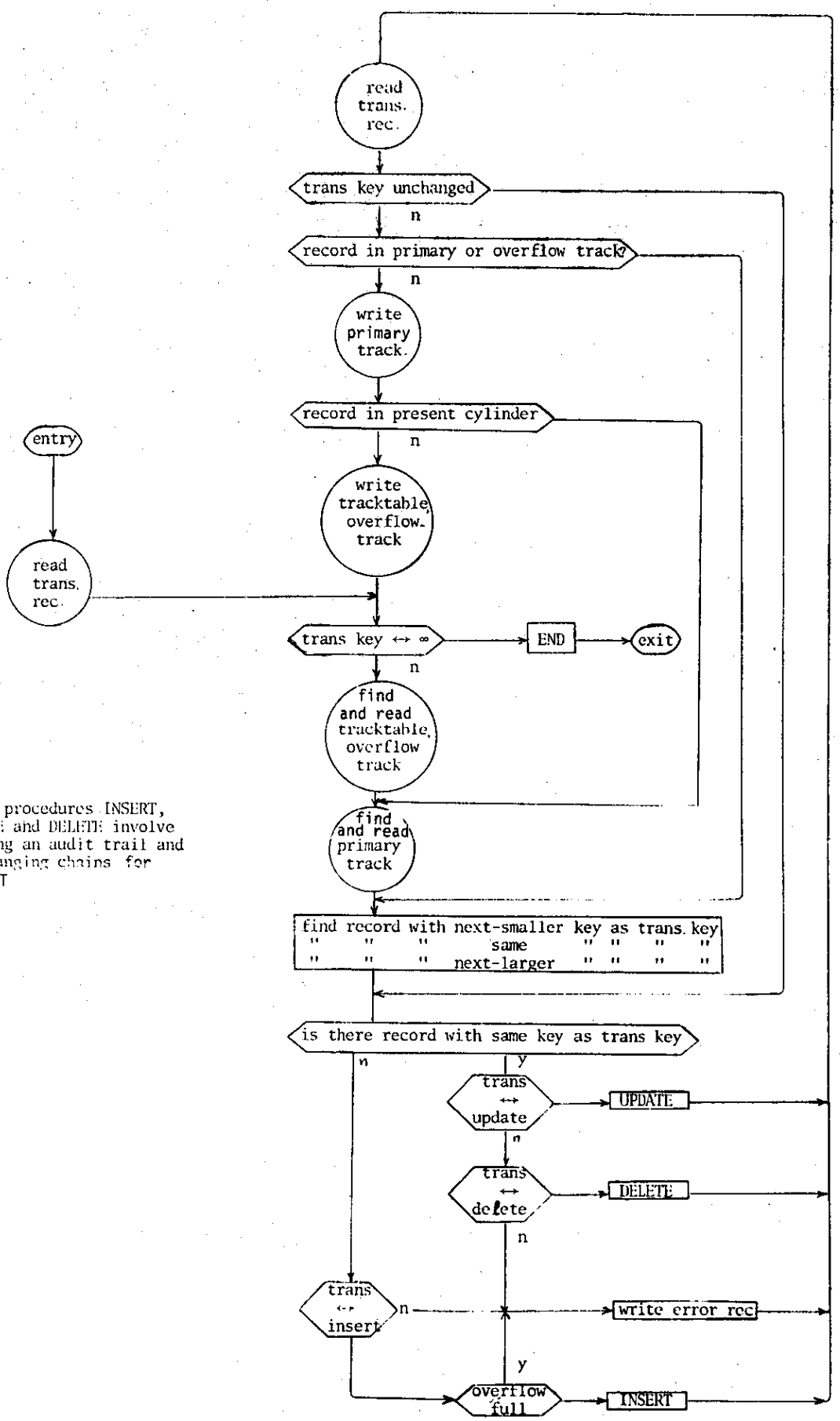
Op grond van het laatste gegeven besluiten we om de transacties in sorteervolgorde te verwerken. Om armbewegingen zo veel mogelijk te vermijden is het voorts belangrijk om voor de verwerking van een transactie eerst te onderzoeken of de transactie betrekking heeft op een record dat in dezelfde cylinder of spoor staat of (bij een insert) komt te staan als het bij de vorige transactie betrokken record. De besparing in tijd is dan vaak een armverplaatsingstijd (50-100 ms) of een halve omwentelingstijd (ca 10-20 ms) van het schijfsysteem.

In bijgaand schema is aangegeven dat wanneer het bij de transactie betrokken record en (terwille van de ketting) voorgaand en volgend record gevonden zijn, het UPDATE, INSERT of DELETE proces gebeuren. Bij een UPDATE bewerking zijn verder geen problemen, ook aan de kettingorganisatie hoeft niets te gebeuren. Voor een INSERT moet
. onderzocht worden of er nog ruimte is in het overloopspoor;
. zo neen, dan kan de INSERT niet uitgevoerd worden;
. zo ja, dan moeten in ieder geval de kettingen veranderd worden;
. in de sporen- en cylindertabel zo nodig de sleutel van het nieuwe record toegevoegd worden.

Bij de DELETE moet het corresponderende gebeuren, als het record inderdaad verwijderd moet worden. (Bij andere varianten van de index-sequentiële organisatie vindt in grote trekken hetzelfde plaats, behalve dat de kettingbehandeling en alles wat daarmee samenhangt door bijvoorbeeld sequentiële zoekmethoden voor het overloopspoor vervangen moet worden.)

2.7. Index-random structuren

Een minder vaak gebruikte opslagstructuur is de index-random structuur, waarbij een indextabel in het secundaire geheugen gevonden wordt door randomizing van de recordsleutel. In de zo gevonden indextabel staan de volledige oorspronkelijke recordsleutels en de corresponderende schijfadressen. Het toevoegen van records gebeurt door een nieuw record bij te schrijven op het eerstvolgende vrije schijfadres en dit schijfadres op te nemen in de na randomizing



Note: procedures INSERT, UPDATE and DELETE involve writing an audit trail and rearranging chains for INSERT

File maintenance for index-sequential system with chained record organization

gevonden indextabel. Het verwijderen van een record vereist het verwijderen van zijn sleutel uit de betreffende indexpagina en het terugplaatsen van de recordruimte bij de verzameling vrije recordruimten (zodat voor het laatste een kettingstructuur gunstig is).

Deze opslagstructuur is zodoende gunstig voor een bestand met een hoge turnover, dat gemiddeld echter niet veel van totale omvang verandert. De vulling van het secundaire geheugen is immers zeer compact, hetgeen bij een gewone random organisatie slechts na het vinden van een goede randomizer redelijk te garanderen is. Om overflow te vermijden moeten de indextabellen niet te ver gevuld zijn (dit vereist echter niet veel schijfruimte) en evenzo moeten overflow indexpagina's ruim bemeten zijn (ook dat eist weer niet veel schijfruimte), omdat het vollopen hiervan een bewerkelijke reorganisatie van het hele bestand vergt.

Tenzij het aantal indextabellen zo klein is dat men deze in het kernengeheugen kan houden, vergt deze methode per zoek echter twee leesoperaties tegenover één leesoperatie bij random structuren. Net zo min als bij random structuren is echter sequentiële verwerking van het bestand mogelijk, zoals wel kan bij een index-sequentiële organisatie. De laatste vergt echter twee à drie leesoperaties per zoek en iets meer geheugenruimte.

Opgave

Onderzoek zelf details van deze opslagstructuur, o.a. de vraag of men de indextabellen zal sorteren of hier een kettingorganisatie zal gebruiken.

2.8. Inverted file structuren

De tot dusver besproken structuren zijn min of meer geschikt voor "zoekprocessen", dus voor het vinden van gegevens op grond van de sleutel van een record. Wanneer het echter om een "retrieval" probleem gaat, d.w.z. het vinden van records waarvan bepaalde items aan zekere voorwaarden (meestal gelijkheid of ongelijkheid) moeten voldoen, zit er met de behandelde structuren vaak niet veel anders op dan alle records van een bestand te lezen, de items waar het om gaat te onderzoeken en de records die voldoen aan de voorwaarden weg te schrijven op een andere informatiedrager. Het volledige bestand moet zo doorgewerkt worden, hetgeen voor zeer grote bestanden kostbaar kan zijn.

Wanneer het aantal voor een retrieval relevante items, vermenigvuldigd met het aantal mogelijke waarden dat ieder van die items kan aannemen, niet al te groot en het oorspronkelijke grote bestand willekeurig toegankelijk is, kan een inverted file goede uitkomst bieden. In feite is deze te vergelijken met een trefwoordensysteem van de systematische katalogus van een grote bibliotheek. Men bouwt namelijk bij het oorspronkelijke bestand een geordend hulpbestand - de "inverted file" - op, waarvan de recordsleutels bepaald worden door de namen en mogelijke waarden van voor retrieval relevante items. De items van de inverted file records zijn dan de (gesorteerde) sleutels (of fysieke adressen) van alle records van de oorspronkelijke file, waarvan de corresponderende items aan dezelfde bepaalde voorwaarde voldoen. Een retrieval vraag laat zich met behulp van logische and, or en not operatoren formuleren, bijvoorbeeld een retrieval van die records in een bevolkingsadministratie behorende bij bewoners die èn getrouwd zijn èn drie minderjarige òf geen kinderen hebben, maar niet een vrij beroep hebben (is deze zin eigenlijk zonder gebruik van haakjes wel éénduidig te interpreteren?). Met behulp van de inverted file vindt men dan direct de recordsleutels (of recordadressen) van de gewenste records in het oorspronkelijke bestand, immers in het hulpbestandstuk "gehuwd" vindt men de recordsleutels van alle gehuwde bewoners, in het hulpbestandstuk "drie minderjarige kinderen" de recordsleutels van alle bewoners met drie minderjarige kinderen, in het hulpbestandstuk "nul kinderen" die van alle bewoners met nul kinderen en in het hulpbestandstuk "vrij beroep" tenslotte de recordsleutels van alle bewoners met een vrij beroep. Door dan overeenkomstig de retrievalvraag logische bewerkingen op de groepen recordsleutels uit te voeren isoleert men tenslotte de recordsleutels van de gewenste records, die dan zo mogelijk direct opgezocht kunnen worden. De groepen recordsleutels moeten voor een efficiënte logische bewerking uiteraard gesorteerd zijn.

Hoe de record- en bestandsopbouw is van de inverted file moet door de ontwerper van deze structuur bepaald worden op grond van de specifieke toepassing, overeenkomstig hetgeen bij voorgaande structuren opgemerkt is.

Opmerking

Bij retrieval problemen betrekking hebbende op literatuurverwijzingen treden vaak problemen op bij de keus van sleutels of zoals ze in dit terrein meestal genoemd worden: descriptoren. Descriptoren zijn namelijk vaak synoniemen in die zin dat zij voor een retrieval een identieke functie hebben. Zo kunnen bijvoorbeeld bepaalde wiskundige artikelen geclassificeerd worden met de descriptoren: discrete wiskunde of combinatorische wiskunde of misschien ook coderingstheorie. Wanneer men nu bij een retrieval vraag niet alle synoniemen voor een bepaalde descriptor opgeeft, dan is het duidelijk dat bij retrieval slechts een deel van de gewenste literatuur gevonden wordt.

Bij automatische retrieval systemen wordt het synoniemenprobleem als regel opgelost met een zg. thesaurus, die hierop berust dat alle synoniemen verwijzen naar een standaardterm die op zijn beurt verwijst naar alle records waarop de standaardterm betrekking heeft. De gebruiker van een thesaurussysteem hoeft deze standaardterm dus niet te kennen of op te zoeken (vergelijk het bij bibliotheekkaartjes bekende systeem van: "zie ook ...") om toch de goede antwoorden op zijn retrieval vraag te krijgen (uiteraard verder aannemend dat de descriptoren goed gebruikt zijn en geen spelfouten worden gemaakt). Voorbeeld: de synoniemen A'dam, Amstelredam, Mokum, enz. verwijzen allemaal eerst naar Amsterdam.

Een ander probleem dat zich bij automatische retrieval problemen kan voordoen, is dat een bepaalde descriptor met verschillende interpretaties gebruikt wordt. Soms is dit een blunder te noemen bij de keus van toegelaten descriptoren, maar in andere gevallen is het misschien niet te vermijden. Voorbeeld: woorden als systeem, data set. Bij gebruik van zo'n descriptor zonder meer krijgt men uiteraard te veel niet-relevante literatuurverwijzingen. Wordt zo'n descriptor in een and-relatie met een andere descriptor gebruikt dan elimineert deze fout zich veelal zelf (bijvoorbeeld data set en frequency geeft vrijwel zeker alleen literatuur over modems). Bij geraffineerde retrieval systemen wordt de vrager aangespoord om eerst nog een synoniem voor een door hem gebruikte descriptor te geven of uit een lijst van synoniemen een keus te doen (bijvoorbeeld bij dataset moet hij kiezen uit modem of file).

Opgave

Schets wat er bij een inverted file systeem moet gebeuren bij toevoeging, verwijdering of verandering van een record in het oorspronkelijke bestand.

2.9. Gebruik van bestandsmutaties in een informatiesysteem

Nadat in het voorgaande enkele bestandsstructuren zijn behandeld, zullen we nu aan een enkel voorbeeld bekijken hoe een bestandsmutatie toegepast wordt in een informatiesysteem. Dit zal gebeuren aan de hand van het volgende (uiterst gesimplificeerde) informatiesysteem, waaraan slechts de volgende eisen worden gesteld:

. bedrijfsafdelingen, die op willekeurige dagen stukgoederen uit een centraal magazijn betrekken, willen maandelijks een overzicht hebben van betrokken artikelen (hoeveelheden en kosten) en jaarlijks een overzicht van de kosten van groepen van betrokken artikelen samen met de begrotingsbedragen van het afgelopen jaar;

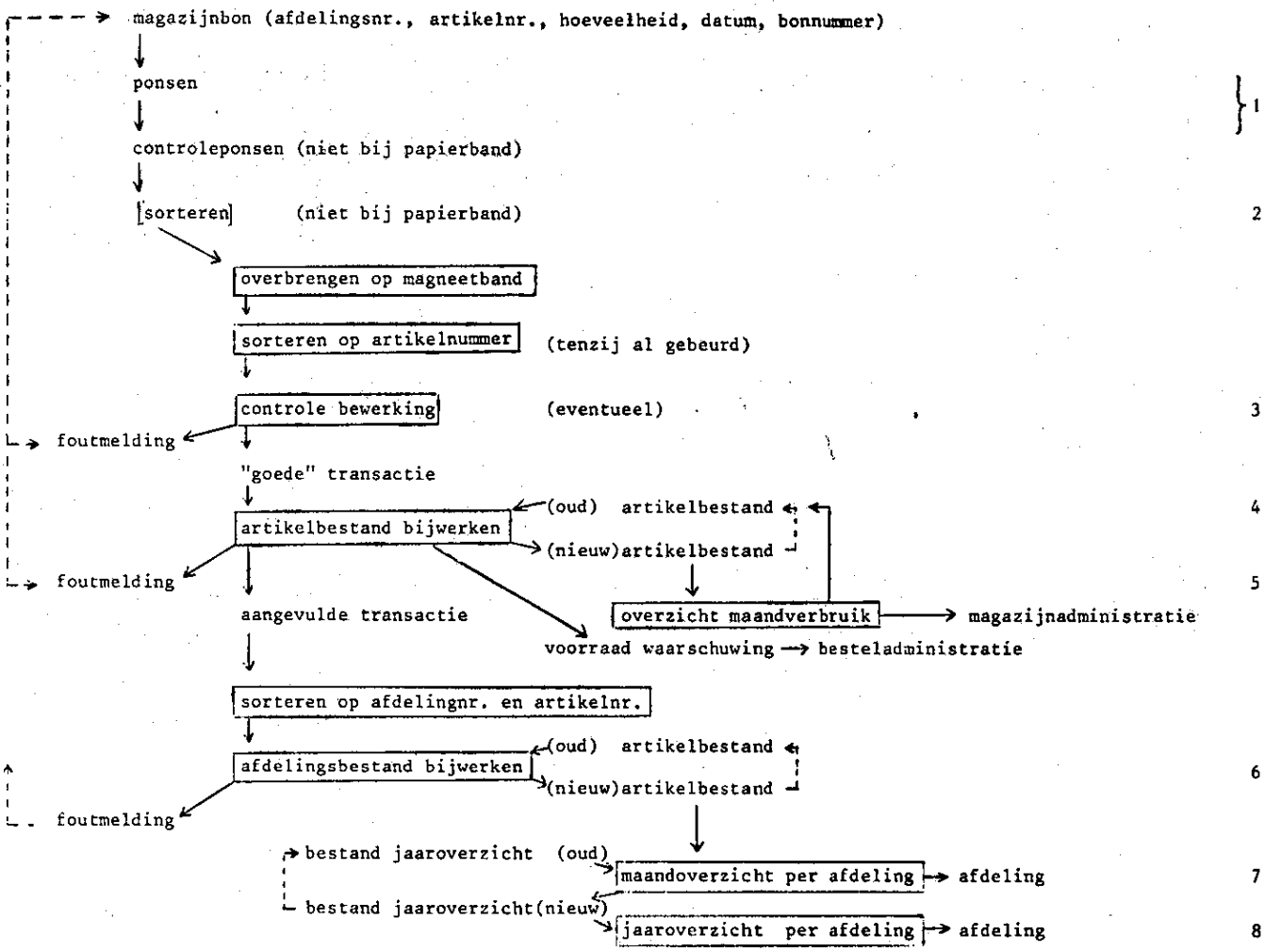
. het centrale magazijn wil maandelijks een overzicht hebben van afgegeven artikelen en dagelijks, wanneer nodig, een waarschuwing dat van een bepaald artikel de voorraad beneden een zeker minimum gedaald is, opdat de bestel-administratie voor aanvulling kan zorgen.

Op grond van deze gegevens kan tot in bijgaan (onvolledig) geschetst schema voor een informatiesysteem besloten worden, waarin slechts drie bestanden een rol spelen.

In het afdeelingsbestand, dat begin van iedere maand opnieuw opgebouwd wordt, staat gesorteerd op afdelingsnummer en daarbinnen op artikelnummer, de informatie op de oorspronkelijke magazijnbon en de naam en kosten van de per bon betrokken artikelen. In het bestand jaaroverzicht, dat begin van ieder jaar opnieuw opgebouwd wordt, staan gesorteerd op afdelingsnummer en daarbinnen op groep van artikelnummers slechts de cumulatieve kosten voor iedere groep van gedurende een jaar betrokken artikelen en de corresponderende begrotingsbedragen. Bij de eerste opbouw van dit bestand worden de cumulatieve kosten op nul gezet en de begrotingsbedragen ingevuld, hetgeen een aparte bestandscreatie vergt, die niet in het schema is aangegeven.

Voorts liggen de volgende procedures voor de hand:

. een magazijnbon (mogelijk een deels voorgeponste kaart), waarvan indeling en afmeting handig zijn voor zowel afdeling als ponstypiste, bevat tenminste de in het schema genoemde gegevens (en verder vakjes voor een handtekening, omschrijving van artikel, enz.). De afgegeven hoeveelheid artikelen en eventueel het artikelnummer worden pas ingevuld door de magazijnmeester, die voorts de bonnen dagelijks doorgeeft aan het rekencentrum; de andere gegevens worden door de bestellende afdeling ingevuld en geverifieerd door de magazijnmeester;


magazijnbon (afdelingsnr., artikelnr., hoeveelheid, datum, bonnummer)
ponsen
controleponsen (niet bij papierband)
sorteren (niet bij papierband)
1
2
overbrengen op magneetband
sorteren op artikelnummer (tenzij al gebeurd)
controle bewerking (eventueel)
3
foutmelding
"goede" transactie
(oud) artikelbestand
artikelbestand bijwerken
(nieuw)artikelbestand
4
foutmelding
5
aangevulde transactie
overzicht maandverbruik
magazijnadministratie
voorraad waarschuwing
besteladministratie
sorteren op afdelingnr. en artikelnr.
(oud) artikelbestand
afdelingsbestand bijwerken
(nieuw)artikelbestand
6
foutmelding
bestand jaaroverzicht (oud)
maandoverzicht per afdeling
afdeling
7
bestand jaaroverzicht(nieuw)
jaaroverzicht per afdeling
afdeling
8

. aan het eind van iedere werkdag worden de binnengekomen magazijnbonnen verponst, geverifieerd en 's avonds door de computer verwerkt; foutmeldingen en voorraadwaarschuwingen liggen de volgende ochtend bij de betreffende afdelingen. Verponste magazijnbonnen worden een jaar lang bewaard voor eventuele reclame's;
. de besteladministratie die binnengekomen artikelen aan de voorraden toevoegt, heeft een uit slechts nullen bestaand afdelingsnummer;
. een slechts uit negens bestaand artikelnummer correspondeert niet met een aanwezig artikel, maar wordt voor controledoeleinden gebruikt. De (laatste) bon met dit artikelnummer bevat namelijk bijv. op de plaats van het afdelingsnummer het totaal aantal ingeleverde magazijnbonnen van een dag en op de plaats van de hoeveelheid het totaal aantal op die dag afgegeven artikelen;
. gesignaleerde fouten mogen de verwerking niet ophouden, maar moeten de volgende werkdag(en) door boekingen met negatief teken gecorrigeerd worden;
. artikelen zijn zodanig gecodeerd dat de in het jaaroverzicht gewenste groepen van artikelen corresponderen met de voorste digits van het artikelnummer.

Met deze gegevens is het bijgaande schema eenvoudig te volgen; enkele opmerkingen zijn de volgende:

1. De moeilijkheid om ponswerk in de vroege avonduren te doen verrichten is er vaak de oorzaak van dat de aflevering van lijsten een halve tot hele dag vertraagd wordt of dat men al midden op een werkdag moet "afsluiten". In beide gevallen dus een administratie die zeker een halve werkdag achterloopt. Vandaar het toenemend gebruik van automatisch verwerkbare documenten of on-line apparaten (als de kosten daarvan gerechtvaardigd zijn).
2. Mechanische sortering van ponskaarten is te overwegen (indien een sorteermachine en bediener aanwezig zijn), omdat dit kan gebeuren terwijl de rekenautomaat nog met iets anders bezig is.
3. Zoals geschetst is de controlebewerking slechts een gedeeltelijke:
   . de controletotalen uit de laatste bon kunnen worden vergeleken met de door de machine opgebouwde totalen van de gegevens in de bonnen. Dit is slechts een administratieve controle op wegraken van magazijnbonnen en/of ponskaarten en/of ponsfouten!
   . de in de bonnen vermelde gegevens kunnen aan plausibiliteitscontroles onderworpen worden (bestaan van afdelings- en artikelnummers, redelijk zijn van hoeveelheden in combinatie met die nummers, enz.)
   . in combinatie met een (oud) artikelbestand is een diepergaande plausibiliteitscontrole uit te voeren.

Daar tijdens het bijwerken van het artikelbestand verschillende controles herhaald kunnen worden, kan de controlebewerking achterwege worden gelaten, wanneer

. de accuratesse van het invullen van bonnen hoog is (meestal te bereiken door "zondaars" door de machine te laten signaleren!);
. het doorslippen van een fout geen ernstige consequenties heeft voor de rest van het informatiesysteem;
. eventueel later geconstateerde fouten eenvoudig gecorrigeerd kunnen worden (door tegenboekingen).

Bedenk wel dat een aperte "verschrijving" (verkeerd afdelings-of artikelnummer) vrijwel niet door de machinale bewerking opgevangen kan worden, maar pas achteraf door een attente persoon, die maandoverzichten serieus doorneemt.

4. In het artikelrecord staan naast items als momentane hoeveelheid, hoeveelheid aan het begin van de maand, artikelnummer, datum van de laatste mutatie ook gegevens als artikelomschrijving, eenheidsprijs, minimumvoorraad (op grond van OR-studie te bepalen), leverancier, enz. die met een niet-geschetste bestandsmutatie ingevoerd worden.
5. Enkele andere opmerkingen:
   . in de aangevulde transactie staan naast de oorspronkelijke transactiegegevens nu ook de artikelnaam en het product van hoeveelheid × prijs;
   . het verbruik van ieder artikel per maand wordt berekend uit:
   (begin hoeveelheid + toevoegingen) - hoeveelheid na de laatste mutatie;
   . de voorraadwaarschuwing (eventueel met een bestelbrief als de te bestellen hoeveelheid ook in het artikelrecord opgenomen is) wordt alleen geproduceerd als de voorraad onder het minimum zakt (bij meer geraffineerde systemen moet rekening worden gehouden met bestelde, maar nog niet ontvangen, hoeveelheden, levertermijn, enz.).
6. In het afdelingsbestand staan gesorteerd op afdelings- en artikelnummer de aangevulde magazijnbonnen, zodat
7. maandelijks deze informatie getotaliseerd kan worden en (eventueel met de uit het bestand jaaroverzicht afkomstige groeptotalen) afgedrukt.
8. Uit het bestand jaaroverzicht kan dan eind van het jaar op analoge wijze een jaaroverzicht worden opgebouwd. Of en hoe het bestand jaaroverzicht samengevoegd kan worden met het afdelingsbestand willen we in het midden laten (werk dat zelf uit!).

Opgave

Tracht stukken van het in de inleiding gegeven geïntegreerde informatiesysteem te detailleren, zoals hiervoor een magazijnprobleem is geschetst. Werk ook dit magazijnprobleem in meer details uit.

2.10. Bestandsbescherming

Dit woord omvat eigenlijk verschillende aspecten die zich als volgt laten groeperen:

file protection - hieronder verstaat men als regel bescherming tegen calamiteiten, die tot de vernietiging van de physieke informatiedrager of de informatie daarop kunnen leiden. Naast het zo goed mogelijk voorkómen van zulke calamiteiten zal men ook maatregelen (als regel het bewaren van bestandcopiëen op veilige plaatsen) moeten treffen om in het geval van een calamiteit binnen korte tijd toch de operaties te kunnen hervatten. Enkele gevarenbronnen en te treffen maatregelen zijn bijvoorbeeld:

- brand: minimum aan brandbare materialen om en in de computerruimte (vooral onder de vloer en boven de zoldering), rook- en brandmelders, rookverbod, brandbestrijdingsmiddelen (geen water), brandbestrijdingsoefeningen, permanente bewaking;
- wegraken van een informatiedrager: op te vangen met bestandscopiëen en/of een grootvader/vader/zoon systeem, en een goede administratie van informatiedragers;
- sabotage: beperkte toegang tot computerruimte en/of rekencentrum, voorkomen van waterschade, geen bezoekers bij magneetband- en schijfopslag, antecedentenonderzoek van personeel;
- no-break stroomvoorzieningen wanneer apparatuuronderdelen stuk kunnen gaan bij onverwacht uitvallen van de stroomvoorziening.

Aangezien het treffen van sommige maatregelen hoge kosten met zich meebrengt, zal van geval tot geval onderzocht moeten worden of ze gezien de te verwachten of te aanvaarden risico's verantwoord zijn.

file integrity - hieronder verstaat men als regel bescherming van bestandsgegevens tegen (foute) overschrijving:

- hardware fouten door verkeerde afstelling van bandlezers of kaartlezers, trommel- en schijfeenheden, door overmatige stofafzetting op deze eenheden. Het optreden van zulke fouten moet, voor zover de hardware ze niet zelf al signaleert, gedetecteerd worden met programmatische middelen als controletotalen, enz.

- software fouten van de programmeur(s). De kans op het optreden van zulke fouten moet door een goede training, programmeringssystematiek, enz. zo veel mogelijk verkleind worden. Waar mogelijk moet een correctheidsbewijs van een programma worden gegeven. (Het "testen" van een programma is geen bewijs voor de correctheid van een programma, maar alleen voor het niet-fout zijn bij de gebruikte testdata!);
- operateursfouten, zoals het opzetten van verkeerde magneetbanden, het vergeten van bandprotectie-ringen, of inlezen van verkeerde data.
  Naast een goede training van operateurs moet er in de programma's voor gezorgd worden dat eventueel gemaakte fouten niet ongedetecteerd blijven. Bijvoorbeeld door het gebruik van interne filelabels, die eerst door het programma gecontroleerd worden, het signaleren van twee keer met dezelfde data muteren van een bestand, het zodanig opbouwen van programma's dat in geval van storing een operateur het werk op een eenvoudige wijze kan hervatten.

file privacy - hieronder verstaat men een eigenlijk wijder probleem, nl. hoe te voorkomen dat gegevens van vertrouwelijke (persoonlijke) aard terecht komen bij niet daartoe gerechtigden en omgekeerd hoe te bewerkstelligen dat degene op wie de gegevens betrekking hebben, weet wat er over hem vastgelegd is, dat hij het recht heeft om wijziging van onjuiste gegevens via bepaalde instanties te eisen, enz. Deze problematiek heeft als zodanig niets met een computer te maken, maar is toch wel door die computer acuut geworden omdat dit apparaat niet alleen de mogelijkheid verschaft om inderdaad snel gegevens te produceren, maar door zijn karakteristieken ook vele instanties er toe zal brengen om gecentraliseerd grote hoeveelheden gegevens op te slaan. Het recht op privacy, d.w.z. het recht om persoonlijke gegevens niet aan iedereen te hoeven vertellen kan door gecentraliseerde opslag van gegevens een farce worden.

Het grootste deel van deze problematiek, nl. wie gerechtigd is om informatie te ontvangen, moet een oplossing vinden in de ethisch-juridische sfeer (had in principe al 25 jaar geleden een oplossing moeten hebben). In de technische uitvoeringssfeer moet gegevenstransport naar niet daartoe gerechtigden automatisch voorkomen worden, ook als deze zich via terminals of door omkoping toegang trachten te verschaffen tot die gegevens. Van geval tot geval zullen hiertoe de geeigende oplossingen moeten worden gezocht door:

- organisatorische maatregelen, zoals tweesleutel kluizen voor zeer vertrouwelijke gegevensbestanden, toezicht bij het verwerken van deze bestanden door terzake deskundigen, persoonlijke overdracht van resultaten van computerbewerkingen, enz.
- software maatregelen, zoals het gebruik van bestand- en recordlabels, het inbouwen van geprogrammeerde controles op de aanvrager van bewerkingen met het bestand, het afdrukken van naam en adres van de aanvrager op de eerste pagina van computeroutput, het na afloop van de bewerking uitwissen van primaire en secundaire geheugens (behalve uiteraard het oorspronkelijke bestand), enz.
- hardware maatregelen, zoals het gebruik van zichzelf identificerende en afsluitbare terminals, enz.

Vooral door een combinatie van deze maatregelen zal een bijna volledige file privacy bereikt kunnen worden tegen relatief geringe meerkosten. Dit tenminste tot het moment dat de gegevens bij de "gerechtigde" ontvanger arriveren.

3. SORTEREN

Onder sorteren wordt verstaan het plaatsen van de (logical) records in een voorgeschreven volgorde. Deze volgorde is meestal naar stijgende of dalende sleutelwaarde, waarbij de sleutel een bepaald (vast) item uit het record is. Het kan voorkomen dat een bepaald bestand voor de ene toepassing volgens een andere item gesorteerd wordt dan voor een andere. Bijv. volgens salarisnummer of achternaam. Bij alpha-numerieke sleutels moet de volgorde van letters en cijfers afgesproken worden!

Bij de bestandsmutaties hebben we gezien dat sorteren een essentiële rol speelt. We gingen er toen van uit dat om een zo efficiënt mogelijk mutatieproces te verkrijgen (zeker bij een sequentieel hoofdbestand) het mutatiebestand gesorteerd was. Bijgevolg worden ook aan het sorteerproces efficientieeisen gesteld. Wanneer het sorteren met de hand of d.m.v. ponskaartmachines gebeurt worden de informatiedragers (kaarten) van de oorspronkelijke plaatsen in het ongesorteerde bestand naar de uiteindelijke plaats in het gesorteerde bestand verplaatst (eventueel via een aantal "tussenplaatsen"). Gebeurt dit d.m.v. een rekenmachine dan wordt de informatie van de ene naar de andere plaats gecopieerd. Om in dit laatste geval een machine-onafhankelijke vergelijking van sorteerprocessen mogelijk te maken, wordt het gemiddeld aantal bewerkingen geteld die nodig zijn om uitgaande van een random rij de gesorteerde rij te verkrijgen.

We zullen een aantal sorteerprocessen bekijken die m.b.v. een rekenautomaat uitgevoerd kunnen worden. We splitsen deze in twee klassen, de zg. interne sorteermethoden, hierbij staan de gegevens in het kernengeheugen, en de externe sorteermethoden, waarbij de gegevens op een achtergrond geheugen staan.
We zullen ons voornamelijk beperken tot de interne methoden en onderschijden daarbij

a) invoegtechnieken
b) selectietechnieken
c) samenvoegtechnieken
d) radixtechnieken
e) splitsingstechnieken.

In de te behandelen voorbeelden nemen we eenvoudigheidshalve aan dat de records slechts bestaan uit een integer, de sleutel, en opgeslagen zijn in integer array A[1:n] ($n \geq 1$). Gesorteerd wordt naar opklimmende sleutelwaarden.

Invoegtechnieken

De rij te sorteren records is d.m.v. een wijzer gesplitst in een reeds gesorteerd stuk, en een nog te sorteren stuk. Het element waar de wijzer naar wijst, wordt op de goede plaats in het gesorteerde stuk tussengevoegd, waarbij dus in het algemeen een gedeelte van deze rij één plaats moet opschuiven. Begonnen wordt met een ongesorteerde rij, m.a.w. de wijzer wordt geïnitialiseerd op het eerste element. Wij zijn klaar wanneer de hele rij geordend is, dus als de wijzer wijst voorbij het laatste element.

```
begin integer array A[1:n]; integer w,h,i,j;
        w := 1;
    while      w ≤ n do
        begin   h := A[w]; i := 1;
                while h > A[i] do  i := i+1;
                for j := w step -1 until i+1 do A[j] := A[j-1];
                A[i] := h;
                w := w+1
        end
end
```

Voor n elementen hebben we n geheugen plaatsen nodig, plus een aantal extra plaatsen voor de wijzer w en andere (hulp) variabelen. De geheugenbezetting is dus evenredig met n, of anders gezegd van de orde n, weergegeven als $\mathcal{O}(n)$.

Wanneer m elementen reeds gesorteerd zijn zal men, wanneer uitgegaan van een random ongesorteerde rij, gemiddeld ½ m vergelijkingen maken om de plaats te vinden waar het "aangewezen" element komt te staan, vervolgens zijn er gemiddeld ½ m opschuivingen nodig om deze plaats vrij te maken. Het aantal bewerkingen voor een element is dus evenredig met ½ m. Het totaal aantal bewerkingen dus evenredig met $\sum_{m=1}^{n} \frac{1}{2} m = \frac{1}{4} n^2$. In de praktijk zal echter de tijd die nodig is voor een vergelijking niet dezelfde zijn als die voor een opschuiving, met het gevolg dat de gemiddelde tijd nodig om een element op zijn plaats te zetten evenredig is met $\frac{1}{2} m\alpha + \frac{1}{2} m\beta = \frac{1}{2} m(\alpha+\beta)$.

Versie met minder array "kontakten":

```
begin integer array A[1:n]; integer w,h,i;
              w := 1;
              while w ≤ n do
                    begin
                     i := w; h := A[w];
                     while A[i]≥h and i>1 do begin A[i]:=A[i-1]; i:= i-1 end;
                     A[i] := h
                    end
end
```

Selectiemethoden

In de ongeordende rij wordt het kleinste element gezocht. Dit element wordt op de eerste plaats gezet in een nieuw te vormen geordende rij, op de plaats van dit element in de ongeordende rij wordt een merkteken geplaatst. Van de aldus overgebleven ongeordende rij wordt het kleinste element bepaald en dit wordt geplaatst op de tweede plaats in de geordende rij, enz. Deze rij is kompleet wanneer n maal een kleinste element uit de (oorspronkelijk) ongeordende rij bepaald is.

```
begin integer array A[1:n], AS[1:n]; integer i,j,k,kleinste;
      for i := 1 step 1 until n do
          begin kleinste := A[1]; k := 1;
                for j := 2 step 1 until n do if A[j] < kleinste then
                                             begin kleinste := A[j];
                                                   k := j
                                             end bepaling kleinste;
                A[k] := merkteken;
                AS[i] := kleinste
          end
end
```

Dit algorithme werkt alleen maar goed wanneer merkteken groter is dan de grootst mogelijk voorkomende sleutel, of wanneer deze niet bekend is merkteken gelijk gemaakt wordt aan de grootst mogelijke integer die de machine kan bevatten.

Om het kleinste element te bepalen zijn n vergelijkingen nodig, dit gebeurt n maal; bijgevolg zal de sorteertijd $\theta(n^2)$ bedragen. De geheugenbezetting is $\theta(2n)$.

De geheugenbezetting is eenvoudig met een faktor 2 te reduceren door de gesorteerde rij "in" het array A op te bouwen; m.a.w. we splitsen A wederom in een gesorteerd en een ongesorteerd stuk. Als volgt: we bepalen van het array A[1:n] het kleinste element. Dit element plaatsen we op de eerste plaats van A, en het element dat oorspronkelijk op deze plaats stond, verhuist naar de plaats van het kleinste. Nu is het eerste element van het gesorteerde array A bepaald, en moeten we nog slechts het array A[2:n] sorteren. Dit gebeurt weer op dezelfde manier. Op deze wijze bepalen we de eerste n-1 elementen van A. Na afloop hiervan is dus A[1:n] gesorteerd.

```
begin integer array A[1:n]; integer w,i,k,kleinste;
      w := 1;
      while w < n do
            begin kleinste := A[w]; k := w;
                  for i := w+1 step 1 until n do if A[i] < kleinste then
                                                 begin kleinste := A[i];
                                                       k := i
                                                 end bepaling kleinste;
                  A[k] := A[w]; A[w] := kleinste; comment omwisseling;
                  w := w+1; comment gesorteerd stuk met een element uitgebreid;
            end
end
```

Het aantal vergelijkingen dat nu nodig is om het kleinste element te bepalen bedraagt de eerste keer n-1, de tweede keer n-2, etc. Het totaal aantal vergelijkingen wordt dus $\sum_{n-1}^{1} k = n(n-1)/2$. Hiermede is ook de orde van de sorteertijd bepaald, en we zien dat ook nu voor grote n deze tijd $O(n^2)$ bedraagt.

Een kenmerk van de tot nu toe behandelde algorithmen is dat het aantal bewerkingen nodig om het array A te sorteren onafhankelijk van de oorspronkelijke volgorde van de elementen van A is. Dit is zeker wanneer men a priori reeds weet dat het array gedeeltelijk gesorteerd is een bezwaar. Hieraan komt het volgende algorithme tegemoet: Twee naast elkaar gelegen elementen van A worden met elkaar vergeleken, is de volgorde verkeerd dan worden deze van plaats verwisseld. Deze verwisseling wordt genoteerd (in het Algolprogramma wordt een boolean variable w true gemaakt). Op deze manier wordt het hele array doorlopen.

Konstateren we nu dat er tenminste één verwisseling heeft plaatsgehad, dan herhalen we vorenstaand proces weer voor het gehele array. Hiermee gaan we zolang door totdat we na een "array doorgang" ontdekken dat er geen verwisseling meer heeft plaatsgevonden, en kennelijk het array gesorteerd is. Hiervan bespreken we twee versies:

bubble-sort: de kleinste ("lichtste") elementen "borrelen" naar hun plaats

```
begin integer array A[1:n]; integer i; boolean w;
      w := true;
      while w do begin w := false;
                       for i := n step -1 until 2 do
                            if A[i] < A[i-1] then begin wissel (A[i],A[i-1]);
                                                        w := true
                                                 end
                 end
end
```

Om te ontdekken of het array A reeds bij aanvang gesorteerd is, zullen we dat één keer moeten doorlopen, vandaar dat w op true geïnitialiseerd wordt. Bij iedere "array-doorgang" hopen we dat dit de laatste is, vandaar dat we hier w false maken. Is dit niet de laatste, dan zullen (tenminste) twee buurelementen van plaats verwisselen, en wordt gezien onze afspraak w true gemaakt.

brick-sort: de grootste ("zwaarste") elementen "zakken" naar hun plaats

```
begin integer array A[1:n]; integer i; boolean w;
      w := true;
      while w do begin w := false;
                       for i := 1 step 1 until n-1 do
                       if A[i] > A[i+1] then begin wissel (A[i], A[i+1]);
                                                   w := true
                                            end
                 end
end
```

Merk op dat bij de bubble-sort na een array-doorgang het kleinste element op de eerste plaats staat, dit zal dus in het verdere verloop niet meer van plaats veranderen.

Voor de brick-sort geldt dit voor het grootste element en de laatste plaats. Hiervan kunnen we gebruik maken door in het eerste geval de benedengrens uit de forstatement per slag met één te verhogen, en in het laatste de bovengrens met één te verlagen.
Beide algorithmen kunnen gecombineerd worden tot:
bubble-brick sort

```
begin integer array A[1:n]; integer ℓ,r,i,m;
      ℓ := 1; r := n;
            while r > ℓ do
                  begin comment brick; i := ℓ; m := i;
                        while i < r do begin if A[i] > A[i+1] then begin wissel (A[i], A[i+1]);
                                                                         m := i
                                                                   end;
                                                i := i+1
                                          end;
                        r := m; i := r;
                        bubble:
                        while i > ℓ do begin if A[i] < A[i-1] then begin wissel (A[i], A[i-1]);
                                                                         m := i
                                                                   end;
                                                i := i-1
                                          end;
                        ℓ := m
                  end
end
```

De variabelen ℓ en r zijn twee wijzers met de volgende betekenis: het array A[ℓ:r] dient nog gesorteerd te worden, A[1:(ℓ-1)] is gesorteerd en bevat de "kleinste" elementen A[(r+1):n] is gesorteerd en bevat de "grootste" elementen.
In het "brick gedeelte" wordt in de variabele m bijgehouden waar de laatste verwisseling heeft plaats gevonden, m.a.w. A[(m+1):n] is na afloop van brick gesorteerd, dus (zie boven) kan nu r := m uitgevoerd worden. Ga zelf na dat een soortgelijke redenering voor m in het "bubble gedeelte" opgesteld kan worden.
Zolang er nog een te sorteren stuk overblijft voeren we dit proces uit, m.a.w. zolang er geldt dan r > ℓ.
De behandelde "bubble" algorithmen bezetten een stuk geheugen van $\theta(n)$.
De berekening van de sorteertijd is, omdat deze afhankelijk wordt van de oorspronkelijke volgorde en grootte van de elementen, statistisch van aard en daardoor moeilijker dan in de voorgaande algorithmen. De kleinste sorteertijd krijgen we wanneer het array gesorteerd aangeboden wordt, dit wordt vastgesteld

d.m.v. n-1 vergelijkingen, dus $\Theta(n)$. De grootste sorteertijd treedt op in het ongunstige geval wanneer het array "omgekeerd" gesorteerd aangeboden wordt, het grootste element staat dan op de eerste plaats, het kleinste op de laatste plaats enz. Per "slag" (n-1 vergelijkingen) schuift dit kleinste element een plaats naar links, na n-1 slagen staat dit dus op zijn plaats. Gedurende de n^e slag wordt gekonstateerd dat het array gesorteerd is, en stopt het proces. In totaal zijn er nu n(n-1) vergelijkingen gemaakt, en zal deze sorteertijd $O(n^2-n)$ zijn.
Voor een afleiding van de gemiddelde sorteertijd zie (Stone); hierbij wordt uitgegaan van een volledige random verdeling van de elementen in het array. Het blijkt dat deze tijd evenredig is met $N^2 - 1.25\ N^{3/2} - N + 1.25\sqrt{N}$.

Er is in het bovenstaande een aantal keren gebruik gemaakt van de procedure wissel. Deze kan als volgt geprogrammeerd worden:

```
procedure wissel (k,ℓ); integer k,ℓ;
                begin integer s;
                      s := k; k := ℓ; ℓ := s
                end
```

Samenvoegtechnieken

Kenmerkend voor de tot nu toe behandelde sorteer algorithmen is dat de sorteertijd een kwadratische funktie van het aantal te sorteren objecten is. Het zal duidelijk zijn dat, zeker voor zeer grote bestanden, men gezocht heeft naar algortihmen waarbij de sorteertijd minder snel met het aantal objecten toeneemt. Hiervan zullen we een aantal technieken bespreken.
Uitgaande van twee (of meer) gesorteerde deelrijen wordt één gesorteerde rij gemaakt.

```
begin integer array A[1:n],B[1:n], C[1:(n+m)];
      integer i,j,k;
      i := j := k := 1;
      while i ≤ n and j ≤ m do
            begin if A[i] < B[j] then begin C[k] := A[i]; i := i+1 end
                                 else begin C[k] := B[j]; j := j+1 end;
                  k := k+1
            end;
      while i < n then begin C[k] := A[i]; i := i+1; k := k+1 end;
      while j < m then begin C[k] := B[j]; j := j+1; k := k+1 end;
end
```

Bij deze samenvoeging gaan we er van uit dat de sorteervolgorde van de deelrijen (A en B) dezelfde is als die van de resultaatrij (C). We beginnen met het eerste element van A en dat van B met elkaar te vergelijken. Het kleinste van deze twee wordt op de eerste plaats van C gecopieerd. In de (deel)rij waaruit dit kleinste element afkomstig is, wordt het hierop volgende element geselecteerd. Van het nu geselecteerde element uit A en uit B wordt weer het kleinste bepaald, enz. We zien dit in vorenstaand algorithme weergegeven. De wijzers i en j wijzen naar het geselecteerde element in A resp. B, k wijst naar de eerstvolgende te vullen plaats in de resultaatrij C.
We moeten wel bedenken dat de deelrij A en B niet even lang behoeven te zijn; dat we een aantal malen "achter elkaar" uit dezelfde deelrij kunnen selecteren, met het gevolg dat A en B niet gelijktijdig "uitgeput" behoeven te zijn.
In de eerste while clause wordt zólang een kleinste element uit A en B naar C overgeheveld totdat één van beide deelrijen uitgeput is.
In de tweede en derde while clause wordt nagegaan welke deelrij nog niet leeg is, en zijn resterend stuk wordt in C gecopieerd.
Deze methode wordt in de literatuur de two-way merge genoemd, bij gebruik van p deelrijen spreekt men van een p-way merge (schrijf hiervoor zelf een Algol-programma).
Wanneer we deze techniek willen gebruiken voor het sorteren van een willekeurige rij getallen, moeten we deze rij dus splitsen in een aantal gesorteerde deelrijen. Uitgaande van een volledige random rij van n getallen, kunnen we (ten hoogste) n gesorteerde deelrijen verwachten. Deze deelrijen kunnen we met een two-way merge in een aantal slagen tot een gesorteerde rij van n getallen samenvoegen. Per slag wordt de lengte van een deelrij verdubbeld, het aantal deelrijen gehalveerd (als n een tweevoud is). Hoeveel slagen zijn hiervoor nodig? Stel x is het aantal slagen; dan is x het kleinste gehele getal dat voldoet aan $2^x \geq n$, dus $x \geq {}^2\log n$. Hadden we een p-way merge gebruikt dan werd het aantal slagen bepaald door ${}^p\log n$. Geef voor beide processen de formule voor de sorteertijd.
Literatuurstudies hebben aangetoond dat een merge proces het snelst verloopt wanneer bij iedere slag de te mengen deelrijen even lang zijn.
Wanneer we terugdenken aan de vorige technieken (vooral de bubble-sort) dan zien we dat daar eenvoudiger programma's bij gebruikt worden. Dit heeft tot gevolg dat voor kleine rijen getallen de bubble-sort toch sneller zal zijn dan een merge sort. (Let wel: wij hebben ons bezig gehouden met grote orde beschouwingen en ons niet bekommerd over de functie die het verband aangeeft tussen de

sorteertijd en het aantal records. Deze functie is implementatie- en systeemafhankelijk. Zodoende kunnen voor kleine waarde van n bijv. evenredigheidsfactoren in de lineaire term een grote rol gaan spelen.)
In de praktijk zien we dan ook dat de rij van n getallen gesplitst wordt in rijtjes ter lengte s (waarbij s maximaal 15 is). Deze rijtjes worden gesorteerd met bijv. de bubble-sort en de gesorteerde deelrijen worden met bijv. de two-way merge tot een gesorteerde rij samengevoegd. Voor een p-way merge bestaat dit proces uit sorteerslagen, die gevolgd worden door $^{p}\log \frac{n}{s}$ merge slagen (ga dit na!). De tijd nodig om een rijtje te sorteren bedraagt s^2, de totale tijd om alle deelrijen te sorteren $\frac{n}{s} * s^2 = ns$. Per merge slag zijn p vergelijkingen nodig om het "volgende kleinste" getal te bepalen. Dit moet per slag in totaal n keer gebeuren, dus de merge tijd zal evenredig zijn met $np\ ^{p}\log \frac{n}{s}$. De totale tijd voor dit sorteerproces is evenredig met $np(\frac{s}{p} + {}^{p}\log \frac{n}{s})$.
Ook zien we vaak dat de random rij van n getallen gesplitst wordt in $\sqrt{n}$ deelrijen van $\sqrt{n}$ getallen (mits deze wortel getrokken kan worden, anders het kleinste gehele getal dat groter is dan de wortel). Deze deelrijen worden na sortering in één merge slag tot een gesorteerde rij samengevoegd, en de sorteertijd zal evenredig zijn met $n^{3/2}$ (ga na).
De geheugenbezetting is van $\mathcal{O}(2n)$.

Radixtechnieken

Deze techniek is al zeer oud. De sleutel van een record wordt gerepresenteerd door een getal uit een getalstelsel. Dit stelsel wordt gekarakteriseerd door het grondtal (radix) (bijv. 2 of 10), en de sleutel is bepaald door een aantal cijfers uit dit getalstelsel. Bij het sorteren volgens deze techniek kan men beginnen met de records bijv. op het meest significante cijfer te sorteren. Voor het 10-tallige stelsel maakt men hierbij 10 verschillende stapeltjes, zodanig dat in het 0^e stapeltje alle records komen waarvan de sleutel begin met een 0, 1^e stapeltje alle sleutels beginnend met 1, etc. (als sleutels 413 en 11 voorkomen schrijven we deze laatste als 011). Vervolgens sorteert men binnen ieder stapeltje op het volgend cijfer. Binnen groepjes records waarvoor dit cijfer hetzelfde is, wordt op het volgende cijfer gesorteerd. Hiermee gaat men door totdat alle cijfers van de sleutel aan de beurt geweest zijn. Als laatste stap worden de stapeltjes op elkaar gelegd, en de oorspronkelijke stapel is gesorteerd. Met dit algorithme zou men ook bijv. geadresseerde enveloppen alphabetisch kunnen ordenen.

Veel handiger is het echter om de cijfers uit de sleutel van "achteren naar voren" af te werken. Eerst wordt de stapel volgens het minst significante cijfer gesorteerd. Nadat dit gebeurd is wordt deze stapel volgens het hierop volgende cijfer gesorteerd. We merken op dat voor die records waarvoor dit cijfer gelijk is, de volgorde door het voorafgaande cijfer vastgelegd is. De stapel is gesorteerd wanneer we zo het laatste cijfer behandeld hebben. We hebben nu dus geen (deel)stapeltjes nodig.
Volgens dit laatste principe werkt de mechanische kaartensorteermachine. Deze beschikt o.a. over 12 uitgangen corresponderend met de 12 rijen die men op een ponskaart kan onderscheiden. In de eerste sorteergang sorteert men dus volgens het laatste cijfer en verkrijgt men (voor een sleutel in het 10-tallig stelsel) 10 stapeltjes corresponderend met de cijferwaarde 0, 1 enz. Deze stapels worden weer tot een stapel verenigd door het "0-pak" boven op te leggen, hieronder het "1-pak" enz.
Het aantal keren dat dit pak door de machine gehaald moet worden is gelijk aan het aantal cijfers van de grootst voorkomende sleutel.
Met behulp van het voorafgaande kunnen we opmerken dat de sorteertijd evenredig zal zijn met het produkt van het aantal records en het aantal cijfers uit de grootste sleutel. Tot op zekere hoogte is dit laatst aantal (K) onafhankelijk van het aantal records (N). Wordt de sleutel weergegeven in een getalstelsel met grondtal R, dan bepaalt deze R niet alleen K, maar geldt ook $N \leq R^K$.
Radix sort is erg inefficient, behalve wanneer de verhouding tussen het aantal voorkomende sleutels en de bij de K behorende grootste sleutel dicht bij 1 ligt. Als computeralgorithme is dit dan ook niet erg populair, voor 10-tallige sleutels is bovendien de geheugenbezetting $\Theta(10*n)$.
Aangezien de meeste rekenmachines intern werken met een binair talstelsel, heeft men voor de radix sort met radix 2 altijd de meeste belangstelling gehad. In het volgende zullen we een voorbeeld van deze radix sort behandelen die een efficiënter geheugengebruik garandeert, en bovendien eleganter te programmeren is dan de bovenstaande methode.

Splitsingstechnieken

De ongesorteerde rij wordt gepslitst in twee stukken zodanig dat voor de getallen uit de eerste helft geldt dat deze kleiner zijn dan een bepaald getal (het splitsingscriterium). Voor die uit de tweede helft geldt dat ze alle groter zijn dan dit criterium.

De beide helften worden nu volgens (uiteraard) verschillende splitsingscriteria wederom in twee stukken gesplitst, enz. Hiermee gaat men door totdat de overblijvende stukken alle uit één getal bestaan, en de rij dus gesorteerd is.
De radix-exchange techniek is een voorbeeld van deze methode, toegepast voor binaire sleutels.
Wanneer we aannemen dat alle sleutels even lang zijn, K bits, dan is het splitsingscriterium voor de eerste splitsing 2^{K-1}. Dit kan gemakkelijk gerealiseerd worden door nu "lopend" langs alle sleutels slechts te letten op het meest significante bit (K^e bit). De splitsing gebeurt nu als volgt: het begin van de rij wordt zover doorlopen totdat we een sleutel tegenkomen waarvan de K^e digit een 1 is. Nu wordt vanaf het einde de rij zover naar voren doorlopen totdat een sleutel met K^e digit gelijk aan 0 gevonden wordt. Deze beide sleutels (getallen) worden van plaats verwisseld, en het eerste zoekproces wordt weer voortgezet vanaf de "halte plaats", enz. Twee adreswijzers (pointers) i,j worden gebruikt om aan te geven waar de voorwaartse resp. achterwaartse "zoek" gestopt zijn. Wanneer de twee wijzers gelijk aan elkaar geworden zijn, dan is de rij verdeeld in twee stukken met K^e bit 0 resp. 1. Deze twee deelrijen worden nu gesplitst (afzonderlijk!) met behulp van $(K-1)^e$ bit, (splitsingscriterium 2^{K-2}). Na K sorteer slagen is de rij gesorteerd.
Uit bovenstaande beschrijving volgt, dat deze sorteermethode zeer geschikt moet zijn om met behulp van een recursief werkend programma uitgevoerd te worden. We nemen aan dat we beschikken over een (machine afhankelijk) integer procedure bit (g,k) die van een variabele g het k^e bit bepaalt.

```
begin integer array A[1:n]; integer K;
      procedure split (A,m,n,ℓ,h); value m,n,ℓ; array A; integer m,n,ℓ,h;
                comment deze procedure splitst A[m:n] in twee stukken met
                        bit (A[R],ℓ) = 0 voor m ≤ R < h
                        bit (A[R],ℓ) = 1 voor h ≤ R ≤ n;
                begin integer i,j;
                      i := m-1; j := n+1;
                      repeat begin repeat i := i+1 until bit(A[i],ℓ) = 1 or i = n;
                                   repeat j := j-1 until bit(A[j],ℓ) = 0 or j = m;
                                   if i < j then begin wissel (A[i],A[j]);
                                                       i := i+1; j := j-1
                                                 end
                            end
                      until i ≥ j;
                      h := i
                end;
      procedure radixexchange(A,k,m,n); value k,m,n; array A, integer k,m,n;
                comment deze procedure sorteert A[m:n] bestaande uit integer
                        getallen ter lengte van k bits;
                begin if m < n and k > 0 then begin split (A,m,n,k,h);
                                                   radixexchange(A,k-1,m,h-1);
                                                   radixexchange(A,k-1,h,n)
                                             end
                end;
      radixexchange(A,K,1,n)
end
```

Aangezien het aantal "sorteer-slagen" evenredig is met het aantal digits, wordt deze methode tijdrovend wanneer de sleutels groot zijn. Bedenk hierbij dat een binair getal ongeveer driemaal zo lang is als het corresponderende decimale getal, zodat voor sleutels bestaande uit 5 cijfers 15 sorteer-slagen vereist zijn.
Kunt U een situatie bedenken waar niettemin de radixexchange techniek goed zal voldoen?
Aan dit bezwaar komt het sorteeralgorithme van Hoare, de zg. Quicksort, tegemoet. Met behulp van een referentie element wordt de te sorteren rij als volgt in drie stukken verdeeld: Vanaf het begin wordt de rij zover doorlopen totdat we een element tegenkomen dat groter is dan het referentie element. Vanaf het einde van de rij wordt de rij zover naar voren doorlopen totdat we een element tegenkomen dat kleiner is dan het referentie element. De aldus geselecteerde elementen worden van plaats gewisseld. Hierbij wordt wederom gebruik gemaakt van twee adreswijzers i en j. Gestopt wordt wanneer de wijzers "over elkaar heen" gelopen zijn. De rij is dus nu in drie stukken gedeeld met de volgende eigenschap:

beginstuk bevat elementen die kleiner dan of gelijk zijn aan het referentie element,
middenstuk bevat elementen gelijk aan het referentie element,
eindstuk bevat elementen die groter dan of gelijk zijn aan het referentie element.

Het begin- en eindstuk kunnen weer met behulp van een nieuw referentie element in drie stukken gesplitst worden. De getallen uit het middenstuk staan op hun uiteindelijke plaats in de gesorteerde rij! Gestopt wordt wanneer we begin- en eindstukken bestaande uit één getal overhouden.
Men kan aantonen dat dit sorteerproces het snelst verloopt wanneer telkens na een splitsing de begin- en eindstukken (ongeveer) even groot zijn. Hiervoor is nodig dat het referentie element gelijk is aan het gemiddelde van de te splitsen rij. In de praktijk ziet men dit echter nooit toegepast, het bepalen van het gemiddelde kost immers ook rekentijd! Meestal wordt als referentie element, een willekeurig element uit de rij genomen.

```
begin integer array A[i:n];
      procedure partition(A,m,n,i,j); value m,n; array A; integer m,n,i,j;
                comment deze procedure splitst met behulp van een random
                        element x uit A[m:n](m < n), de rij A in drie stukken
                        m ≤ j < i ≤ n,
                        A[R] ≤ x voor m ≤ R ≤ j
                        A[R] = x voor j < R < i
                        A[R] ≥ x voor i ≤ R ≤ n
                        Met behulp van een procedure random (m,n) wordt random
                        een integer F bepaalt met m ≤ F en F ≤ n.
                begin integer x,F;
                        F := random (m,n); x := A[F];
                        i := m-1; j := n+1;
                      repeat begin repeat i := i+1 until A[i] > x or i = n;
                                   repeat j := j-1 until A[j] < x or j = m;
                                   if i < j then begin wissel (A[i], A[j]);
                                                       i := i+1;
                                                       j := j-1
                                                 end
                             end
                      until i ≥ j;
                      if i < F then begin wissel (A[i],A[F]); i := i+1 end
                               else if F < j then begin wissel (A[F], A[j];
                                                        j := j-1
                                                  end
                comment wanneer de wijzers over elkaar heen geschoven zijn
                        wordt gekeken hoe het referentie element ligt t.o.v.
                        A[j:i]
                        Ligt A[F] hier buiten dan kan het middengebied nog met
                        een element (A[F]) uitgebreid worden met aanpassing van
                        de wijzer;
                end;
```

```
procedure quicksort (A,m,n); value m,n; array A; integer m,n;
          comment deze procedure sorteert A[m:n];
          begin integer i,j;
               if m < n then begin partition (A,m,n,i,j);
                                   quicksort (A,m,j);
                                   quicksort (A,i,n)
                             end
          end;
     quicksort (A,1,n)
end
```

Voor de quicksort zullen we aantonen dat de sorteertijd $\Theta(n\ell n(n))$ bedraagt. In eerste instantie zouden we geneigd zijn te denken dat de geheugenbezetting $\Theta(n)$ zal bedragen. Bedenk hierbij echter dat quicksort een recursief proces is, iedere (recursieve) aanroep van quicksort vereist voor opslag van variabelen en link-informatie een aantal extra geheugenplaatsen. Dit aantal wordt bepaald door machinestructuur en algolvertaler. Het totaal aantal extra geheugenplaatsen zal dus evenredig zijn aan de recursie diepte (deze is evenredig aan log n, zoals aangetoond kan worden).

(Hoe zit dit bij de radix-exchange methode?)

In plaats van de random keuze van het referentie element hadden we ook steeds het eerste element van de te sorteren rij kunnen nemen. Aanroep van quicksort voor een reeds gesorteerde rij zou echter nu aanleiding geven tot n^2 vergelijkingen. De te sorteren rij wordt steeds gesplitst in het eerste element en de rest, bijgevolg dat de recursiediepte n bedraagt. Dit kan aanleiding geven tot zoveel geheugenruimte, dat de machinecapaciteit overschreden wordt. Ook bij een andere rij kan dit gebeuren, wanneer steeds bij splitsing het eerste en laatste stuk zéér ongelijk van lengte zijn. Soms lost men dit op door het langste stuk met een andere techniek (bijv. bubble sort) verder te sorteren. Ook bij iedere andere keuze van het referentie element kan men een rij bedenken waarvoor het aantal vergelijkingen $\Theta(n^2)$ zal bedragen. Kiest men het referentie element steeds random, dan is de kans op voorkomen van deze ongunstigste rij te verwaarlozen.

De afleiding van het aantal vergelijkingen nodig voor het sorteren van n getallen, met een random keuze van het referentie element verloopt als volgt:

a(n) = verwachtingswaarde van het aantal vergelijkingen

K = random getal uit {1,...,n}.

Alle getallen worden vergeleken met het referentie element, na n-1 vergelijkingen houden we twee deelrijen over, een met K-1 getallen, de ander met n-K getallen, K is de plaats waar het referentie element na splitsing terecht is gekomen. Omdat dit referentie element random uit de rij gekozen wordt, is K dus een random getal uit {1,...,n} en de kans op het voorkomen van één bepaalde splitsing is dus $\frac{1}{n}$.

$$a(n) = n - 1 + \frac{1}{n} \sum_{K=1}^{n} \{a(K-1) + a(n-K)\} \text{ met } a(0) = a(1) = 0,\ a(2) = 1$$

$$\sum_{K=1}^{n} a(K-1) = \sum_{K=1}^{n} a(n-K)$$

$$a(n) = n - 1 + \frac{2}{n} \sum_{K=1}^{n} a(K-1)$$

$$na(n) = n(n-1) + 2 \sum_{K=1}^{n} a(K-1); \text{ dus geldt voor een rij ter lengte } n-1$$

$$(n-1)a(n-1) = (n-1)(n-2) + 2 \sum_{K=1}^{n-1} a(K-1) .$$

Trekt men de twee laatste vergelijkingen van elkaar af:

$$na(n) - (n-1)a(n-1) = 2(n-1) + 2a(n-1)$$

$$\frac{a(n)}{n+1} - \frac{a(n-1)}{n} = \frac{2n-2}{n(n+1)} = \frac{4}{n+1} - \frac{2}{n} .$$

Deze recurrente betrekking kunnen we uitwerken en komen dan tot:

$$\frac{a(n)}{n+1} - \frac{a(1)}{2} = \frac{4}{n+1} + \frac{4}{n} + \ldots \frac{4}{3} - \{\frac{2}{n} + \frac{2}{n-1} + \ldots \frac{2}{2}\}$$

$$\frac{a(n)}{n+1} - \frac{a(1)}{2} = \frac{4}{n+1} + 2\{\frac{1}{n} + \frac{1}{n-1} + \ldots \frac{1}{3} + \frac{1}{2} + \frac{1}{1}\} - \frac{2}{2} - 2\{\frac{1}{2} + \frac{1}{1}\}$$

$$a(n) = 4 - 4(n+1) + 2(n+1) \sum_{t=1}^{n} \frac{1}{t}$$

$$a(n) = -4n + 2(n+1) \sum_{t=1}^{n} \frac{1}{t} .$$

Er is bekend dat

$$\sum_{t=1}^{n} \frac{1}{t} = \ell n(n) + J + \mathcal{O}(\frac{1}{n}),$$

waarin J de constante van Euler is

$$a(n) = 2\{(n+1)\ell n(n) + (n+1)J - 2n + (n+1)\mathcal{O}(\frac{1}{n})\}$$

of

$$a(n) \approx 2n\ell n(n).$$

Niet alleen het aantal vergelijkingen, maar ook het aantal verplaatsingen van de te sorteren getallen bepaalt de totale sorteertijd. Het aantal verplaatsingen kan echter nooit groter zijn dan het aantal vergelijkingen! Het exacte aantal is moeilijker te berekenen, uit literatuurstudie blijkt dat dit gemiddeld $\frac{1}{6}$ van het aantal vergelijkingen is.
We concluderen: de sorteertijd $\mathcal{O}(n\ell n(n))$.

3.7. Sorteren van grote records

Tot nu toe hebben we het sorteren van een rij getallen bekeken, de records bestonden slechts uit één item, de sleutel. In het algemeen zullen bij het sorteren van een bestand de records uit vele items bestaan. We zullen even stilstaan bij de gevolgen die dit kan hebben voor een sorteeralgorithme in het geval dat het bestand is opgebouwd uit vaste lengte records bestaande uit een vast aantal items van gelijke lengte. Aan problemen die voortvloeien uit een "flexibele" data organisatie zullen we voorbijgaan. (Kunt U toch een aantal problemen onderkennen?)
Zouden we bijvoorbeeld de behandelde bubble-sort willen toepassen op het bestand dat opgeslagen is een array A[1:n, 1:10] (dus een bestand van n records ter lengte van 10 items, waarvan het eerste item de sleutel is) dan kan dit met het volgende Algol programma:

```
begin integer array A[1:n, 1:10]; integer j,ℓ;
      procedure wisselrecords (b,i,j); integer i,j; array b
                begin integer k,h;
                      for k:= 1 step 1 until 10 do begin h := b[j,k];
                                                         b[j,k] := b[i,k];
                                                         b[i,k] := h
                                                   end
                end procedure verwisselt de records i en j die "rij-gewijs"
                    in een array b opgeborgen zijn, en uit 10 items bestaan;
      j := n; ℓ := 1
repeat begin while j > ℓ do begin if A[j,1] < A[j-1,1] then
                                       begin wisselrecord (A,j,j-1);
                                             m := j
                                       end;
                                 j := j-1
                           end;
                     ℓ := m; j := n
      end   until j ≤ ℓ
```

Het programma werkt met behulp van twee wijzers (pointers) j en ℓ, de eerste wijst naar de plaats waar het volgend te vergelijken record paar zich bevindt. Het array wordt van rechts naar links doorlopen. De wijzer ℓ verdeelt het array in een reeds gesorteerd stuk en een nog te sorteren stuk, A[1:ℓ-1,1:10] is gesorteerd en A[ℓ:n,1:10] is ongesorteerd. Hieruit volgt dat bij de aanvang ℓ op 1 en j op n geinitialiseerd moeten worden.
Na afloop van een array doorgang (while clause) wijst m naar de plaats waar de laatste verwisseling heeft plaats gevonden, dit wil zeggen dat A[1,1] < A[2,1] < ... < A[m-1,1]. Deze elementen staan op hun uiteindelijke plaats in het gesorteerde array, te sorteren blijft dus nog A[m:n,1:10]. Gezien onze afspraak wordt voor de volgende sorteerslag ℓ := m en j := n. Deze volgende sorteerslag is aanwezig wanneer er een ongesorteerd rechterstuk aanwezig is, m.a.w. zolang er geldt j > ℓ. Het array is gesorteerd wanneer er geldt j ≤ ℓ.
We merken hierbij op dat gedurende dit sorteerproces een record zich op verschillende tussenplaatsen kan bevinden, voordat de eindplaats bereikt is.

Dit van plaats tot plaats copiëren kost machinetijd, die evenredig is met de lengte van een record. Ten opzichte van de sorteertijd voor een rij bestaande uit integer getallen, veroorzaakt dit een verlenging van de sorteertijd. Het zal duidelijk zijn, dat we zoeken naar methoden om deze verlenging zo klein mogelijk te houden. We kunnen dit bereiken door de sleutels inclusief een wijzer naar de plaats van het betreffende record in het ongesorteerde bestand, in een array slw[1:n,1:2] op te slaan. We spreken af dat slw[i,1] de sleutel van een record i is, waarvan de plaats in het ongesorteerde bestand A[1:n,1:10] gegeven wordt door slw[i,2]. Dit array slw[1:n,1:2] wordt gesorteerd volgens opklimmende sleutelwaarden.
Met behulp van dit array kunnen we het bestand A sequentieel volgens opklimmende sleutelwaarde doorlopen. We hebben hiermee een (gesorteerde) kettingstructuur geintroduceerd. Het begin van de ketting wordt gegeven door slw[1,2]. De plaats van het i^e lid wordt bepaald door slw[i,2], de wijzer naar het opvolgende record bevindt zich op slw[i+1,2], die naar het voorafgaande record op slw[i-1,2]. Wanneer er geëist wordt dat de records gesorteerd in een array opgeslagen worden, dan wordt het array AS[1:n,1:10] gevuld met behulp van slw[1:n,1:2].

```
begin integer array A[1:n,1:10], AS[1:n,1:10], slw[1:n,1:2];
      integer j,ℓ,m;
      for j := 1 step 1 until n do begin slw[j,1] := A[j,1];
                                         slw[j,2] := j
                                   end vullen van slw[1:n,1:2];


      j := n; ℓ := 1;
      repeat begin while j > ℓ do begin if slw[j,1] < slw[j-1,1] then
                                        begin wissel(slw[j,1], slw[j-1,1]);
                                              wissel(slw[j,2], slw[j-1,2]);
                                              m := j
                                        end;
                                        j := j-1
                                 end;
             ℓ := m; j := n
             end until j ≤ ℓ; comment sorteren van slw[1:n,1:2];
```

```
        for j := 1 step 1 until n do
            begin ℓ := slw[j,2];
                  for k := 1 step 1 until 10 do AS[j,k] := A[ℓ,k]
            end vullen van AS;
end
```

We merken op dat bij dit sorteerproces de records uit A slechts één keer gecopieerd worden. Helaas gaat dit gepaard met het vullen en sorteren van een hulparray slw[1:n,1:2], dat gelukkig steeds (onafhankelijk van de record-opbouw van A) twee-dimensionaal zal zijn. Zeker wanneer de records van A zeer groot zijn (wij werkten met records van 10 integers) zal dit proces sneller zijn dan het voorafgaande.
Indien het gebruik van het array AS[1:n, 1:10] niet mogelijk is door ruimtegebrek in het kernengeheugen, dan kan het array A met het volgende programma gesorteerd worden. (Er wordt nu aangenomen dat slw[1:n, 1:2] gesorteerd is.) De grove structuur van dit programma moet de volgende zijn:
for j := 1 step 1 until n do "vul j^e rij van A"

Vul j^e rij van A: vul j^e rij van A met rij uit A gegeven door slw[j,2]. Omdat er geen informatie uit A verloren mag gaan, wordt dit uitgevoerd als een omwisseling van de rijen A[j,1:10] en A[slw[j,2], 1:10]. Nu moeten we echter wel opppassen, de rij (A[slw[j,2], 1:10]) die op de plaats van A[j,1:10] moet komen te staan, is van zijn oorspronkelijke plaats verdwenen wanneer geldt slw[j,2] < j. Deze verdwijning is het gevolg van een omwisseling, en de rij zal zich bevinden op A[slw[slw[j,2],2],1:10], mits het nu gebruikte rijnummer van A groter is dan j, zoniet dan moeten we nogmaals de tabel raadplegen, enz.

```
begin integer array A[1:n,1:10], slw[1:n,1:2];
      integer j,ℓ;
      for j := 1 step 1 until n do
          begin ℓ := slw[j,2];
                while ℓ < j do ℓ := slw[ℓ,2];
                if ℓ > j then wisselrecord (A,ℓ,j)
          end
end
```

Het vullen van de volgende rij in het reeds gedeeltelijk gesorteerde array A, vereist een verwisseling van twee records. (Afgezien van het zoeken in een tabel). Dit maakt dit proces langzamer dan het voorafgaande. Hoeveel? Wat kunt U zeggen over dit proces, in vergelijking met het eerst behandelde?

3.8. Externe sorteermethoden

Wanneer een bestand een zodanige omvang heeft, dat dit niet meer in het kernengeheugen past, dan heeft dit consequenties voor het sorteerproces. Aangezien nu dit proces zeer sterk beïnvloed wordt door de beschikbare ruimte in het kernengeheugen en de karakteristieke gegevens van het achtergrondgeheugen (back-up storage), zullen we deze methoden slechts zeer summier aanduiden. Beschikken we over 4 magneetbandeenheden dan kunnen we gebruik maken van de balanced sort. De gegevens die op de oorspronkelijke band staan, worden als volgt over twee banden verdeeld: De records worden van de (invoer) band gelezen; zolang de sleutelvolgorde klopt met de sorteervolgorde worden de records weggeschreven naar de eerste (uitvoer) band. Klopt de volgorde niet meer, dan wordt overgeschakeld naar de tweede (uitvoer) band. Het laatst gelezen record wordt hierop weggeschreven, dit wordt gevolgd door (volgende) records van de invoerband zolang nu weer de volgorde klopt. Bij onjuist volgorde wordt weer overgeschakeld naar de eerste uitvoerband enz. Dit proces wordt voortgezet totdat de invoerband in zijn geheel gelezen is. Nu wordt de invoerband meestal afgenomen en op de twee vrije magneetbandeenheden worden nieuwe banden geplaatst. De twee uitvoerbanden worden nu invoerband, en door middel van een two-way merge proces worden een (gesorteerde) deelrij van invoerband 1 en invoerband 2 samengevoegd tot een nieuwe (gesorteerde) deelrij die op de (nu) eerste uitvoerband komt te staan. De volgende twee deelrijen worden op de (nu) tweede uitvoerband geplaatst, de hierna volgende weer op de eerste, enz. Zijn zo alle deelrijen samengevoegd, dan wisselen uit- en invoerband weer van functie, enz. Dit proces wordt voortgezet totdat de gehele rij gesorteerd is. Beschikken we over magneetbandeenheden die ook "achteruit" lezen kunnen (backward read), dan kunnen we bij bovengenoemde functiewisseling de terugspoeltijd (rewind time) besparen. Welke gevolgen heeft dit voor het merge proces?

Wanneer we oorspronkelijk uitgegaan waren van een volkomen ongeordend bestand, dan zal na splitsing de gemiddelde lengte van de gesorteerde deelrijen op de twee magneetbanden twee records bedragen (waarom?).
Het aantal merge slagen kan verminderd worden door bij de vorengenoemde splitsing, eerst zoveel mogelijk records in het kernengeheugen in te lezen, deze te sorteren met behulp van een interne sorteertechniek, en de aldus gevormde deelrijen afwisselend hetzij op de eerste of de tweede uitvoerband weg te schrijven. Het aantal merge slagen dat nu nodig is, kan gemakkelijk berekend worden met behulp van een soortgelijke redenering als bij de behandelde interne merge sort. Dit idee kunnen we generaliseren voor 2p magneetbandeenheden. De uitsplitsing van het oorspronkelijke bestand gebeurt nu over p magneetbanden. De deelrijen op deze p magneetbanden worden met een p-way merge samengevoegd. Iedere nieuwe samengevoegde rij wordt op de volgende magneetband weggeschreven (geteld wordt modulo p). Het gehele sorteerproces vereist nu minder merge slagen. De relatieve tijdwinst voor dit proces is toch echter minder dan de relatieve vermindering van het aantal merge slagen in vergelijking met het proces dat gebruik maakt van de two-way merge. Dit komt omdat nu p input blocks in het kernengeheugen aanwezig moeten zijn in plaats van 2, met het gevolg dat de blokkingsfactor kleiner zal moeten zijn, en dit maakt de totale lees-schrijftijd per merge slag groter.
De behandelde methode heet balanced sort omdat steeds geldt dat de te mengen magneetbanden een (ongeveer) gelijk aantal deelrijen bevatten.
Zullen we nog bijzondere complicaties moeten verwachten wanneer het oorspronkelijke bestand op meer dan één magneetband opgeslagen is?
Een nadeel van de balanced sort is dat slechts de helft van het aantal bandeenheden deelneemt aan het merge proces, van de andere helft wordt een eenheid gebruikt als uitvoermedium, de rest is inactief.
Bij de polyphase sorting worden alle bandeenheden op één na gebruikt als invoerstroom bij het merge proces. De ene overblijvende dient als uitvoerband. Dit proces vereist echter een bepaalde initiële verdeling van het aantal (gesorteerde) deelrijen over de banden. Voor drie bandeenheden wordt dit aantal bepaald door twee opeenvolgende getallen uit de rij van Fibonacci (2,3,5,8,13,21,etc.). Begonnen wordt bijv. met 8 deelrijen op band 2 en 13 deelrijen op band 3. Deze worden gemengd, en het resultaat wordt geschreven op band 1, dit mengproces wordt echter gestopt wanneer de 8 deelrijen op

band 2 gebruikt zijn. Nu wordt een nieuw mengproces gestart, en wel met band 1 en band 3, en band 2 dient nu als uitvoerband. Gestopt wordt wanneer de deelrijen op band 3 opgebruikt zijn, nu gaat band 3 als uitvoerband dienen, enz. Dit proces stopt wanneer op band 1 en band 2 één deelrij staat, die vervolgens gemengd worden tot een gesorteerde rij op band 3. (ga na).

Voor meer dan 3 bandstations is de bovengenoemde verdeling niet eenvoudig aan te geven.

Aan bovengeschetst proces zien we dat hier meer merge slagen nodig zijn dan voor een proces dat met 4 bandeenheden gebruikmaakt van een balanced sort. We moeten echter wel bedenken dat per merge slag niet alle records meedoen, hierdoor kan toch een verkorting van de sorteertijd optreden, in de literatuur spreekt men wel van 15%.

Wanneer het achtergrond geheugen bestaat uit trommels of schijven wordt ook meestal een merge sort toegepast. De relatie tussen de sorteertijd en het aantal records is nu niet meer eenvoudig aan te geven. Bedenk hierbij dat bijv. voor een schijvensysteem de toegangstijd voor een record in de "heersende" cylinder veel kleiner is dan die voor een record uit een andere cylinder.

4. OPZET VAN INFORMATIESYSTEMEN

Een informatiesysteem is in veel opzichten op te vatten als ieder productiesysteem, zij het dat het product iets immaterieels is, nl. informatie. Als bij productiesystemen kunnen we onderscheiden: grondstoffen - de primaire, zo dicht mogelijk bij de winplaats te verzamelen, gegevens -; halffabrikaten - bijv. de controlelijsten - en eindproducten - bijv. jaarverslagen, eindafrekeningen.

Het is daarom eigenlijk merkwaardig dat, vooral in de begintijd, over het hoofd is gezien dat de ervaring en methodiek opgedaan bij de opzet van productiesystemen overgenomen kan worden bij de opzet van informatiesystemen. Met name de ervaring dat de opzet van een systeem niet alleen een technisch probleem is (en dus technische kennis vereist), maar daarnaast ook een bestuurlijk - organisatorisch probleem. Belangrijke facetten zijn in dit laatste verband

- de ontwikkelingsfasen van een systeem,
- de algemene beheersaspecten van een systeem,
- de samenstelling van een groep die een systeem moet opzetten.

4.1.1. Ontwikkelingsfasen

Tegenwoordig wordt algemeen aanvaard dat een systeem in het algemeen de volgende vier fasen moet doorlopen:

- vooronderzoekfase ("toepasbaarheidonderzoek" of "feasibility study"). Het hoofddoel van deze fase is uit een aantal mogelijke projecten een eerste keus te doen, nadat op grond van globale overwegingen en verwachtingen geconcludeerd is dat deze projecten in principe in aanmerking komen voor verdere bestudering. Bepalend voor deze keus zijn meestal de verwachtingen t.a.v. het rendement van de benodigde investeringen, de levensduur van een project, externe factoren die slagen of mislukken van het project kunnen beïnvloeden (beschikbaarheid van kapitaal en expertise, industrialisatiegraad, politieke factoren), interne factoren (scholingsgraad, ervaring met automatisering), enz.
- onderzoekfase ("systeemanalyse", een beter woord zou informatie-analyse zijn). Het hoofddoel van deze fase is om de in de vorige fase doorgelaten projecten in detail verder te onderzoeken teneinde

- de (werkelijke) informatiebehoeften precies vast te stellen,
- te schetsen hoe deze informatiebehoeften in principe bevredigd kunnen worden met een (nieuw of gewijzigd) informatiesysteem,
- een verantwoorde schatting te geven van investerings- en operationele kosten van het te maken informatiesysteem en een kosten-baten analyse te maken met bestaande of andere alternatieven.

• implementatiefase voor projecten die met een positief resultaat door de vorige fase gekomen zijn. Deze fase kan weer onderverdeeld worden in
- systeemontwerpfase, waarin de in de vorige fase gegeven schets voor een informatiesysteem nader gedetailleerd wordt tot concrete voorschriften voor handbewerkingen ("procedures") en/of rekenautomaatbewerkingen (met alles wat daaraan vastzit als detaillering van de bestandsorganisatie, programma-onderdelen, omscholings- en invoeringsvoorstellen, enz.)
- programmerings- en bestandsopzetfase,
- systeeminvoering en -overdracht aan de gebruiker(s).

• evaluatiefase, die als regel pas ½ tot 2 jaar na invoering van het informatiesysteem uitgevoerd wordt met als hoofddoelen
- te verifiëren of de bij de vorige fasen gemaakte veronderstellingen juist zijn geweest (in de eerste plaats om er zelf van te leren voor volgende projecten),
- te onderzoeken of het informatiesysteem werkelijk aan de behoeften voldoet, dan wel gestopt of verbeterd (en hoe) moet worden.

Een vaak gemaakte fout is dat van deze vier fasen slechts de derde wordt uitgevoerd, hetgeen soms resulteerde in technisch volmaakte maar praktisch niet-bruikbare of -aanvaarde informatiesystemen, investeringen die oorspronkelijke schattingen met factoren te boven gaan, vertragingen van maanden tot jaren, inflexibiliteit, enz.
De oorzaak is waarschijnlijk dat in de allereerste begintijd informatiesystemen slechts een ondergeschikt aanhangsel waren bij allerlei productiesystemen; ze waren eenvoudig te overzien en te definiëren. Met andere woorden: de eerste twee fasen werden niet als zodanig onderkend en aanvankelijk bleef men dit doen al werden de deelsystemen steeds groter en ging men hoe langer hoe meer tot "horizontale en verticale" integratie van informatiesystemen over.

Men spreekt van horizontale integratie wanneer bepaalde basisgegevens gemeenschappelijk zijn voor verschillende naast elkaar staande informatiesystemen of wanneer het product van een informatiesysteem dienstig gemaakt kan worden voor het werk van twee of meer naast elkaar staande functies in een organisatie. Van verticale integratie wordt gesproken wanneer de eindproducten van het ene informatiesysteem tevens grondstoffen zijn voor het volgende informatiesysteem. Dit laatste correspondeert vaak met de lagen in een "organisatiepyramide", waarin de onderste laag gevormd wordt door uitvoerende en controlerende functies: het werken volgens een planning en het controleren dat men zich aan die planning houdt. De volgende laag wordt gevormd door functies verantwoordelijk voor de tactische bedrijfsvoering - met de beschikbare faciliteiten zo goed mogelijk plannen - en dan volgt de laag verantwoordelijk voor de strategische bedrijfsvoering - te bepalen welke faciliteiten nodig zijn, gezien de doelstellingen van de organisatie, en zo nodig de doelstellingen bij te stellen. De grenzen tussen deze lagen zijn natuurlijk niet zo scherp als hier gesuggereerd, maar zij corresponderen toch met bepaalde tijdschalen en met verschillende informatiebehoeften.

Met het groter en geintegreerd worden van informatiesystemen ging een verzelfstandiging gepaard (de "automatiseringsfunctie" werd losgemaakt van de financiële administratie, waarin deze functie oorspronkelijk veelal ondergebracht was), terwijl ook de aanpak veranderde van een input-output oriëntering (wat wordt gevraagd en wat is beschikbaar) naar een "database" oriëntering (welke gegevens moeten worden verzameld om reeds uitgesproken en ook in de toekomst te verwachten vragen te beantwoorden).

Een andere vaak gemaakte fout is dat men deze fasen continu in elkaar laat overgaan zonder eerst

- iedere fase (en zelfs subfase) af te sluiten met een duidelijk rapport over het verrichte werk en een taxatie van (de kosten van) het in de volgende fasen uit te voeren werk,
- expliciet acceptatie van het rapport en instemming met de verdere voortgang met het werk gevraagd te hebben aan de verantwoordelijke organisatieleiding en aan de organisatie-onderdelen die later met het informatiesysteem moeten werken.

4.1.2. Beheersaspecten

Vooral voor wat grotere informatiesystemen is het projectbeheer een belangrijke factor naast het eigenlijke technische werk. Dit betekent dat na de goedkeuring om met een bepaalde fase door te gaan door de projectbeheerder het volgende moet worden gedaan:

- een zo nauwkeurig mogelijke schatting (in de tijd en in kosten) moet worden gemaakt van onderdelen van een fase en bepaalde mijlpalen moeten worden gefixeerd op de kalender,
- bij het bereiken van die mijlpalen moet gecontroleerd worden of inderdaad de beoogde werkzaamheden zijn uitgevoerd tegen de geschatte kosten, dan wel vertraging/versnelling en/of budgetoverschrijding heeft plaats gevonden,
- vastgesteld moet worden of correctie van een eventuele afwijking van de oorspronkelijke planning met de beschikbare middelen kan plaats vinden in de volgende periode dan wel of een nieuwe planning moet worden opgesteld (verzetten van mijlpalen en/of grotere investering in mankracht/faciliteiten),
- een korte rapportage van het voorgaande aan de verantwoordelijke organisatieleiding is noodzakelijk (een afwijking van een planning is meestal te vergeven, maar een achterwege laten van een rapportage niet!).

4.1.3. Samenstelling van een groep die een systeem moet opzetten

Slechts voor heel kleine of eenvoudige informatiesystemen kan een groep die verantwoordelijk is voor een (nieuw) informatiesysteem bestaan uit alleen automatiseringsdeskundigen. Alleen dan kan immers een volledige probleemdefinitie opgeschreven worden. Meestal is het opschrijven van een volledige probleemstelling te tijdrovend en zal men overgaan tot het instellen van een projectgroep.

Een projectgroep kan het best bemand worden met een zo klein mogelijk aantal mensen, die dan wel hun volle tijd aan een project kunnen geven. Naast automatiseringsdeskundigen (eerst vooral informatie-analisten en systeembouwers, later ook programmeurs) moeten in een projectgroep geplaatst worden de deskundigen uit het organisatie-onderdeel dat later gebruiker van het informatiesysteem zal worden. De inbreng van de laatste deskundigen bestaat uit:

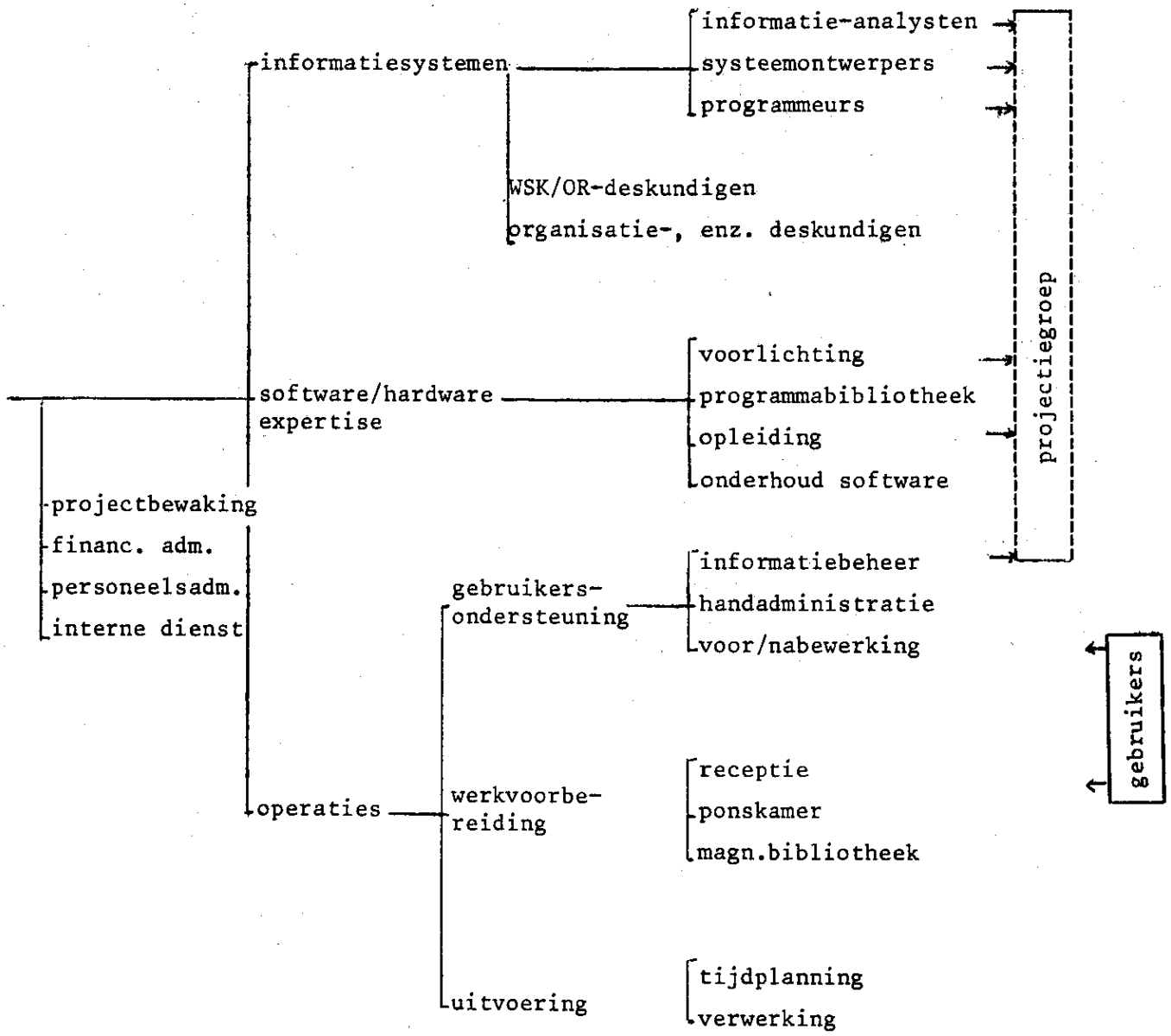
- de specifieke en gedetailleerde kundigheid op hun vakgebied
- de voorbereiding van bestandsopzet of bestandsomvorming
- de voorbereiding van de invoering van het informatiesysteem.

Bij grotere projecten kunnen nog anderen bij een projectgroep betrokken worden op grond van voor het project mogelijk vereiste specifieke kennis (hardware/software deskundigen, wiskundigen, organisatie-deskundigen, deskundigen op het terrein van de γ-wetenschappen, enz.). Wanneer echter niet hun volle werktijd vereist is bij de projectgroep kunnen deze deskundigen beter zitting nemen in een naast de projectgroep in te stellen begeleidingscommissie wier voornaamste functie is om wanneer nodig als klankbord voor de projectgroep te fungeren. Een andere functie van zo'n commissie is vaak om in overleg een keus te maken uit door de projectgroep voorgestelde alternatieve oplossingen of tot een uniforme oplossing te komen voor projecten die meer dan één organisatie-onderdeel betreffen.

Wanneer een project belangrijke consequenties heeft (door de vereiste investeringen of omdat het organisatieveranderingen vereist) kan naast (of soms in plaats van) de begeleidingscommissie beter een stuurgroep ingesteld worden door de top van de organisatie met als belangrijkste taak het verzekeren van medewerking aan het project door de betreffende organisatie-onderdelen.

Zonder een goede begeleidingscommissie of stuurgroep lopen zelfs technisch uitmuntende informatiesystemen het gevaar niet geaccepteerd of gesaboteerd te worden op vaak zeer subjectieve gronden!

In samenhang met het voorgaande wordt hierbij in grote lijnen geschetst hoe tegenwoordig veelal de organisatiestructuur van de informatievoorziening in een grote organisatie is.



4.2. Vooronderzoekfase

Zowel wanneer de opgave is om een organisatie (-onderdeel) "te gaan automatiseren" als wanneer de opgave een detailprobleem betreft, is het voor de voor-onderzoeker nuttig om het volgende vast te leggen teneinde zijn werk een achtergrond te geven:

- de doelstellingen van de organisatie en vooral de te verwachten veranderingen in de doelstellingen (betere of grotere productie, kostenverlaging, nieuwe situatie's enz.),
- het bestaande organisatiepatroon en de uitgaven,
- het "bedrijfsmodel", d.w.z. wat zijn de "grondstoffen" van de organisatie, hoe komen deze in de organisatie, welke operaties worden hiermee uitgevoerd (wat zijn dus de activiteiten), welke hulpmiddelen staan daarbij ter beschikking (mensen, financiën, productiemiddelen, inventaris, leveranciers, enz.), wat zijn de "eindproducten" en hoe verlaten deze de organisatie; wat zijn de beslissingsknooppunten.

Deze vastlegging is vooral van belang voor de voor-onderzoeker zelf en in mindere mate voor de opdrachtgever (in Amerikaanse rapporten verschijnen ze vaak in de vorm van appendices bij een rapport). Van primair belang voor de opzet van een informatiesysteem is nu om op grond van deze gegevens de "ware" belangrijke activiteiten van de organisatie te onderkennen. Het organisatiepatroon alleen kan daarvoor onvoldoende zijn omdat dit vaak óf om historische redenen niet het activiteitenpatroon weergeeft óf bewust volgens een bepaalde "functionele" verdeling (onderzoek, inkoop, productie, transport, verkoop, enz.) of productverdeling of geografische verdeling opgebouwd is en daardoor ook geen uitsluitsel geeft over de belangrijke activiteiten van de organisatie. Een nuttig middel om de activiteiten op te sporen is de uitsplitsing van de totale uitgaven over de knooppunten van het organisatiepatroon; het hierdoor gevormde beeld geeft als regel gauw de gewenste informatie over de activiteiten.

Het onderkennen van het activiteitenpatroon is (afgezien van het mogelijk tot een aangepast organisatiepatroon leiden) belangrijk omdat:

- het inzicht geeft in het nut van een bestaand of voorgenomen informatiesysteem,
- het inzicht geeft op welke gegevens en informatiestromen men zich in de eerste plaats moet concentreren en hoe duplicering van werk vermeden kan worden,

- het inzicht geeft over mogelijke horizontale en/of verticale integratie van informatiedeelsystemen.

Het opstellen van enkele alternatieven voor een informatiesysteem vergt dan het goed nadenken over het bekende wie, wat, waar, wanneer, waarom en over de globale consequenties van ieder van de alternatieven. Al te nauwkeurig moet men dit in eerste instantie niet doen omdat het er nu nog maar om gaat:

- uit de alternatieven de meest aantrekkelijke te kiezen,
- na te gaan of die een antwoord geeft op de oorspronkelijke doelstelling van het onderzoek, dan wel deze doelstelling te wijzigen (of mogelijk zinloos te vinden),
- na beoordelen van de personeels-, financiële en mogelijk onweegbare factoren (bijv. duidelijker taak- of verantwoordelijkheidsvaststelling) de goedkeuring te krijgen voor (eventueel een deel van) de volgende fase.

In een rapport over het voor-onderzoek moeten naast de reeds genoemde punten ook de verdere werkorganisatie en een eerste schatting van tijd en kosten van de volgende fasen opgenomen worden.

4.3. *Onderzoekfase*

Het belangrijkste onderdeel van deze fase is om eerst volkomen los van iedere gedachte aan rekenautomaten tot een nauwkeurige vastlegging te komen van een "logisch" ontwerp, inhoudende een beschrijving van

- data-items (definitie, eenheid, ontstaan, bestaande en/of gewenste plaats van vastlegging, wijze van vastlegging, samenhang met andere data-items, wijze en plaats van gebruik, aantal en frequentie van ontstaan/gebruik, toelaatbare vertragingstijd in productie, aard van vertrouwelijkheid)
- de logische groepering van data-items tot records en bestanden (volgt meestal uit plaats en frequentie van ontstaan/gebruik, dan wel uit de bewerkingen waaraan bepaalde data-items worden onderworpen om tot nieuwe data-items te komen,
- de informatiestromen (ook de mondelinge!) waarvan deze data-items een deel uitmaken volgens mededeling van organisatie knooppunten (meestal onvolledig beeld!), eigen gedachten (snel, maar mogelijk fout!) of post/archief kamer (langzaam en mondelinge overdracht missend!),

- het (nieuwe of gewijzigde) verwerkingssysteem, daarin aangevend de inhoud (niet de opmaak!) van input- en outputdocumenten, de bewerkingen die ze ondergaan van ontstaan tot gebruik (zo nodig gesplitst in handbewerkingen of procedures en rekenautomaatbewerkingen), bij voorkeur in de vorm van beslissingstabellen of stroomschema's (daarbij voor bewerkingsgangen zo mogelijk nog geen volgorde of lussen of input/outputoperaties aangevend).

Al deze zaken moeten in een rapport vastgelegd worden, waarin verder moet worden opgenomen:

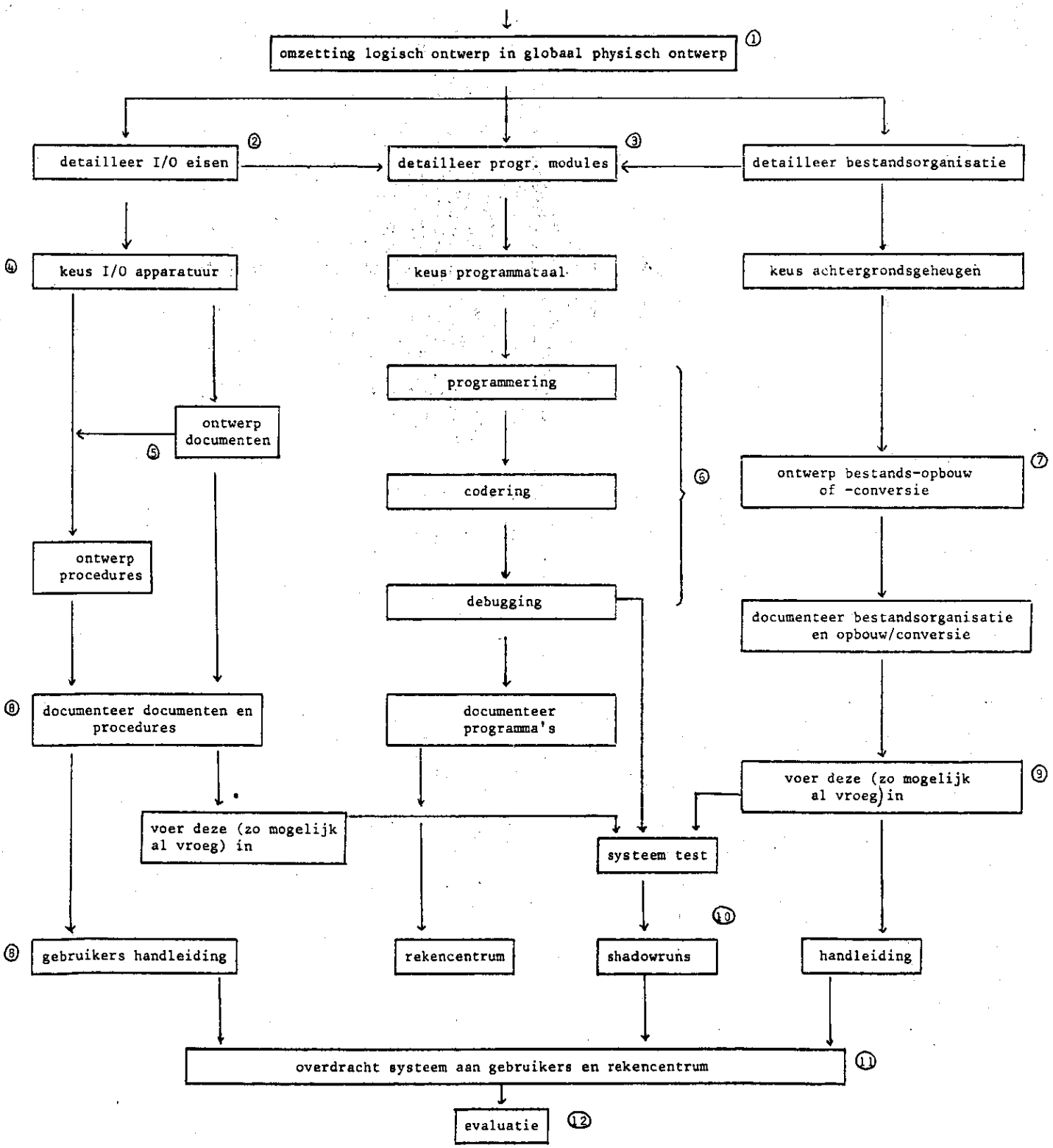
- in hoeverre het voorgestelde verwerkingssysteem voldoet aan de doelstellingen van het onderzoek en flexibel is t.a.v. mogelijke veranderingen,
- vereiste policyveranderingen (organisatie, werkverdeling, betaalwijze, enz.),
- een zo nauwkeurig mogelijke schatting van investerings- en operationele kosten voor de volgende fasen en het tijdschema van realisering (incl. gebruik van rekenautomaten en andere apparatuur) en benodigde mankracht (incl. problemen van aantrekken, training, omscholing, afvloeiing, enz.),
- een kosten/baten analyse van het voorgestelde systeem en indien mogelijk een vergelijking met het oude systeem (kwaliteit en frequentie van rapportage, verminderen van dupliceringen, overwerk, en administratief werk, enz.),
- een verzoek tot expliciete instemming met het voortgaan met de volgende fasen.

Voor-onderzoek en onderzoekfase kunnen het best door een zeer klein (2-5) aantal mensen worden uitgevoerd; ze vergen door meestal moeilijk te organiseren (zinvolle) gesprekken vrij veel tijd (weken tot maanden) doch de kosten zijn in de regel nog niet zo hoog (0.1 tot 2 manjaar voor ongeveer *f* 100.000,- per manjaar). Een voorwaarde is wel dat men in deze twee fasen weerstand heeft kunnen bieden aan de verleidingen van "physiek" ontwerp, programmeren en proefdraaien, waarbij als regel veel meer mensen betrokken zijn en die daarom veel hogere kosten vergen.

Een poging tot bezuiniging op de eerste twee fasen wordt als regel in latere fasen (vaak onzichtbaar) afgestraft in de vorm van vertragingen (door moeilijker coördinatie van nog niet in detail vastgelegd werk en door misverstanden), inflexibele of niet te onderhouden informatiesystemen, onnodig groot computergebruik, slechte documentatie, enz.

4.4. Implementatiefase en evaluatiefase

Al is het voor de vorige twee fasen tot op zekere hoogte denkbaar dat het werk uitgevoerd wordt zonder grondige kennis van rekenautomaten (raadzaam is het niet), dan geldt dit beslist niet voor de implementatiefase. Deze valt uiteen in een aantal subfasen, die gedeeltelijk in serie en gedeeltelijk parallel uitgevoerd kunnen worden en waarbij als regel vrij veel mensen betrokken zijn. De samenhang tussen deze subfasen wordt door het volgende schema gegeven:



Opmerkingen

1 - in de eerste plaats moeten de logische bestanden vertaald worden in physische bestanden en informatiedragers, waarbij de volgende punten in aanmerking moeten worden genomen:
  . benodigde omvang en toegankelijkheid der bestanden (incl. de mogelijkheid van splitsing van bestanden) en wijze van verwerking,
  . de verdeling van data-items (bekend is bij welke activiteiten ze behoren) over physieke bestanden moet zodanig zijn dat bij iedere activiteit niet te veel bestanden behoren (anders corresponderen activiteiten niet met bepaalde programmamodules of zijn te veel informatiedragers nodig bij een bepaalde programmamoduul),
  . bestandsbeveiligingsaspecten;
  - in de tweede plaats moeten input/output operaties en transporten tussen primair en secundair geheugen toegevoegd worden aan het logische ontwerp, vervolgens moeten programmamodules, hun koppeling en de volgorde van verwerking gefixeerd worden om enerzijds aan de systeemeisen, anderzijds aan efficient machinegebruik tegemoet te komen,

2 deze volgen uit aantal en frequentie van data-items en plaats van ontstaan en gebruik en tenslotte uit de voorlopige physieke bestandsopbouw,

3 onderzoek hierbij vooral ook of soortgelijke modules niet reeds beschikbaar zijn in een programmabibliotheek dan wel of ze gekocht kunnen worden,

4 voor zover niet een rekenautomaat beschikbaar is, vormen deze keuzen minimumeisen voor een aan te schaffen rekenautomaat. Veel I/O apparatuur is nog maar pas experimenteel in gebruik!

5 het is uiterst belangrijk om dit zeer snel ter hand te nemen omdat goede documenten en procedures essentieel zijn voor de wisselwerking tussen mens en machine (en soms zelfs een machine overbodig maken). De indeling van documenten hoort vanzelfsprekend te zijn voor mensen die deze documenten invullen of lezen, maar bovendien moet rekening gehouden worden met de registratiefunctie (voor latere naslag), de bewijsfunctie (paraaf) en de boodschapfunctie (circulatie, transport, copieën) en tenslotte bepaalde normen (afmetingen, vensterenveloppen, opbergmiddelen, karakter- en regelafstand voor schrijfmachines, optische lezers, handinvulling, internationale manifesten, uitvoering in kleuren, doorslagen, enz.). Vooral voor het ontwerpen van (gedeeltelijk) voorbedrukte documenten kan daarom beter een vast man aangewezen worden, die ook de zorg heeft voor het nabestellen ervan.

Van documenten die later door een rekenautomaat geleverd zullen worden, moet zo spoedig mogelijk met behulp van een rekenautomaat een dummy afdruk vervaardigd worden om de reactie van toekomstige gebruikers te toetsen. Gedeeltelijk voorbedrukte outputdocumenten heffen soms limiteringen t.g.v. een regeldrukker (max. aantal karakters per regel, beperkte karakterset) op, maar zijn voor de operaties in een rekencentrum lastig (papierverwisseling en regeldrukker instellen) en daarom zelden aan te bevelen.

In een vroeg stadium met input- en outputdocumenten experimenteren in overleg met de gebruiker(s) kan nog zonder overmatig tijdverlies of kosten leiden tot een andere detaillering van programmamodules en daarmee beslissend zijn voor succes of mislukking van een overigens technisch goed ontwerp!

6 zie hiervoor het college "Kunst van programmeren". Met debugging kan slechts aangetoond worden dat een programmamoduul voor de gebruikte gegevens doet wat verwacht wordt!

7 dit onderdeel wordt vaak over het hoofd gezien totdat de systeemtest nadert! De tijd hiervoor benodigd wordt meestal onderschat omdat men de ponstijd vergeet of de tijd nodig om een oud en meestal verwaarloosd bestand bij te werken,

8 de documentatie en later de handleidingen plegen ten onrechte het kind van de rekening te zijn als omvang en kosten van een project zijn tegengevallen. Achterwege laten van documentatie en handleidingen kan misschien wel een bezuiniging (van 15-20%) op de investeringskosten van een project betekenen, maar de operationele kosten stijgen zeker in dezelfde mate (vooral als er later wat veranderd moet worden aan het systeem),

9 als men al in een vroeg stadium kan experimenteren met de nieuwe bestandsorganisatie, desnoods door met noodprogramma's verbindingen tot stand te brengen met andere delen van een oud systeem, dan moet dit zeker niet nagelaten worden. Vooral als men het nieuwe bestand kan updaten is een geleidelijke overgang van oud op nieuw systeem mogelijk, hetgeen zowel voor de gebruikers als het rekencentrum grote voordelen heeft. Een overgang van oud naar nieuw systeem van de ene dag op de andere is in de practijk niet uitvoerbaar,

10 als de debugging goed verlopen is maar de systeemtest niet, dan is dit helaas vaak een aanwijzing dat de onderzoekfase niet goed of niet volledig is uitgevoerd!

Met shadowruns wordt bedoeld dat oude en nieuwe systeem enige tijd naast elkaar gebruikt worden, hetgeen uiteraard extra moeite en kosten met zich brengt. Het is noodzakelijk wanneer een geleidelijke overgang van oude naar nieuwe systemen niet mogelijk is,

11 de overdracht kan inhouden dat al maanden voor de overdracht gebruikers en rekencentrum geoefend hebben met het nieuwe systeem, als is het alleen maar op papier. Een geleidelijke overdracht is, indien mogelijk, wenselijk,

12 de evaluatie van het nieuwe systeem moet niet te vroeg (groeistuipen), maar ook niet te laat plaatsvinden (omdat er dan mogelijk al systeemveranderingen zijn aangebracht). Het is daarom niet gewenst om na een bepaald stadium in de implementatiefase, bijv. na punt (5), nog veranderingsvoorstellen te accepteren. Deze moeten, indien enigszins mogelijk, blijven liggen totdat (12) uitgevoerd is.

Literatuur

Bijgaande literatuur moet gezien worden als (niet verplichte) aanvulling op de stof die op college behandeld wordt. De met een sterretje aangeduide boeken komen op grond van hun kwaliteiten en/of lage prijs voor eigen aanschaf in aanmerking.

Bij hoofdstuk I

* S. Blumenthal - Management Information Systems (Prentice Hall)
Th.B. Glans, B. Grad en D. Holstein - Management Systems (Holt, Rinehart)
M Euwe en J.D. Albarda - Bedrijfsvoering met de computer I, II (Samson)
L.E. Groosman - Moderne computers en hun toepassingen I, II (Kluwer)
H. Reinoud - Automatisering en overheid (Spectrum)
* F.W. Purchall en R.S. Walker - Case studies in business data processing (McMillan)

Bij hoofdstuk II

A.T. Berztiss - Data structures (Ac. Press)
H.S. Stone - Intr. to Computer Organization and data structures (McGraw-Hill)
I. Flores - Data structures and management (Prentice Hall)
* F. Remmen - Inleiding in de informatica (Stenfert Kroese)
O. Dopping - Computers and data-processing (Oxford U.P.)
A. Kent - Information analysis and retrieval (Becker & Hayes)

Bij hoofdstuk III

W.A. Martin - Sorting (Computing Surveys 3 (1971) 147)
I. Flores - Computer sorting (Prentice Hall)
C.W. Gear - Computer organization and programming (McGraw-Hill)

Bij hoofdstuk IV

* H.N. Laden & T.R. Gildersleeve - System design for computer applications (Wiley)
W. Hartman, H. Matthes, A. Proeme - Information systems Handbook (Kluwer)
A. Daniels & D. Yeats - Basic training in systems analysis (Pitman)
B. Drenth & G.W. Eveleens - Systematische aanpak van systeemanalyse en -ontwerp (Samson)

H.D. Clifton - Data processing system design (cases) (Business books Ltd.)
" - Systems analysis for business data processing (Business books Ltd.)
J.E. Bingham & G.W.P. Davies - Handbook of systems analysis (McMillan)
T.R. Gildersleeve - Design of sequential file systems (Wiley)

Algemeen

J. Martin - Design of real-time computer systems (Prentice Hall)
" - Programming real-time computer systems (")
B. Langefors - Theoretical analysis of information systems (Akademisk Vorlag)
E.D.P. Analyzer (tijdschrift).
D. Knuth - Art of Computer programming (Addison-Wesley)