B I N A R Y T R E E

tekenen van een binaire boom
op de printer.

Rudi van Houten
Academisch Computer Centrum Utrecht
Rijksuniversiteit Utrecht
Budapestlaan 6
Postbus 80.011
3508 TA Utrecht
(tel. 030 531436)

In de ACCUkrant van juni 1974 werd een programmeerwedstrijd aangekondigd. De opgave was een programma te schrijven dat een serie getallen zou sorteren gebruik makend van een binaire boom. Nu is sorteren een veel voorkomende handeling door een computer. Er zijn boeken volgeschreven over algorithmen en strategieen om met zo weinig mogelijk rekenoperaties te sorteren. De in de programmeerwedstrijd beschreven methode is erg bekend en zeer economisch indien de aangeboden rij getallen geen enkele ordening bevat.

De wedstrijdopgave zelf, het sorteren, inspireerde me niet zo. Maar het leek me aardig om de efficientie van de methode zichtbaar te maken door de sorteerboom te laten printen door de computer zelf. Een symmetrische boom betekent immers dat er erg efficient gewerkt is. Ik besloot het programma in SIMULA-67 te schrijven. Deze taal bevat ALGOL-60 als subset waarbij het vage STRING-concept vervangen is door het type TEXT (wat SIMULA-67 zeer geschikt maakt voor het werken met teksten). Verder bevat SIMULA-67 een uitbreiding om lijsten te manipuleren. Daarboven is weer een uitbreiding om processen op een tijdas te plaatsen en ze achtereenvolgens uit te voeren. Deze uitbreidingen zijn gedefinieerd in de systeem classes SIMSET en SIMULATION (*).

SEARCH:

Voor het sorteren zelf schreef ik de recursieve procedure SEARCH (begint op regel 11 in het programma). Deze procedure kan in gewone taal beschreven worden met de volgende stappen:

-Vergelijk het getal (k) met het getal in het element van de boom (number). Deze vergelijking kan een van de volgende uitkomsten geven:

1-k < number. Er zijn nu twee mogelijkheden:

A-er is geen linkertak aan het boom-element, dus we maken er een.

B-er is wel een linkertak, nu passen we de procedure toe op het element van die linkertak.

2-k > number.

Er zijn nu dezelfde twee mogelijkheden a en b voor de rechtertak.

3-k=number.

Nu verhogen we het tellertje in het boom-element.

Laten we het voorbeeld van de ACCU-krant eens verwerken. De te sorteren getallen waren: +16, -1, +100, -50, +6. Met het eerste getal (+16) initieren we de top van de boom (zie

---

(*) Het begrip CLASS dat in ALGOL-60 niet bestaat, is het hart van de programmeertaal SIMULA-76.

programma regel 125). Met de volgende getallen roepen we een voor een de procedure SEARCH aan.

- de vergelijking k < number luidt nu:
  -1 < +16? dit is waar, dus naar stap 1:
- is er een linkertak?
  Nee, dus maken we een linkertak en vullen het element met de waarde -1.

Nu het volgende getal, +100

- de vergelijking k < number luidt nu:
  +100 < +16? dit is niet waar, dus naar stap 2:
  +100 > +16? dit is waar, dus:
- is er een rechtertak?
  Nee, dus we maken een rechtertak en vullen het element met de waarde +100.

Nu het volgende getal, -50:

- de vergelijking k < number luidt nu:
  -50 < +16? dit is waar, dus:
- is er een linkertak?
  Ja, dus we beginnen met de linkertak weer vooraan:
- de vergelijking k < number luidt nu:
  -50 < -1? dit is waar, dus:
- is er een linkertak?
  Nee, dus we maken een linkertak en vullen het element met de waarde -50.

Zo komt het laatste getal (+6) aan een nieuwe rechtertak van het element -1.

UNRAVEL:

Het ontrafelen van die boom, dat wil zeggen vanaf die boom de elementen in de volgorde van grootte afdrukken, gebeurt met de procedure UNRAVEL. Deze procedure is nog eenvoudiger dan SEARCH, en laat zich als volgt omschrijven:

UNRAVEL kijkt naar een element van de boom.

1. als er een linkertak is, ontrafel deze.
2. behandel het element (druk het getal af). Deze stap wordt uitgevoerd als er geen linkertak is, of zodra die tak is afgewerkt.
3. als er een rechtertak is, ontrafel deze.

Toegepast op het voorbeeld levert dit:

1. element met +16 heeft een linkertak (element -1), dus
   1. element met -1 heeft een linkertak (element -50), dus
      1. element met -50 heeft geen linkertak, dus:
      2. druk af: -50
      3. element met -50 heeft geen rechtertak.
      - klaar met element met -50
   2. druk af: -1

3.element met -1 heeft een rechtertak (element +6), dus
1.element met +6 heeft geen linkertak
2.druk af: +6
3.element met +6 heeft geen rechtertak.
- klaar met element met +6.
- klaar met element met -1
2.druk af: +16.
3.element met +16 heeft een rechtertak (element +100), dus
1.element met +100 heeft geen linkertak.
2.druk af: +100.
3.element met +100 heeft geen rechtertak.
- klaar met element +100.
-klaar met element +16, dit was de too dus we zijn helemaal klaar.

De vetgedrukte getallen staan in de gewenste volgorde.

De stappen 1b en 2b waarbij de procedure SEARCH zichzelf aanroept, en de stappen 1 en 3 van de procedure UNRAVEL, vormen de recursiviteit. Zo'n recursieve procedure laat zich gemakkelijk opschrijven en lezen, maar is niet de manier om de machine zo efficient mogelijk te laten rekenen. Deze overweging moet een programmeur er echter niet van weerhouden een eenvoudige maar niet optimale techniek te kiezen. Programmeren is altijd het afwegen van de eenmalige (?) inspanning van de mens tegen de herhaalde activiteiten van de machine. Daarbij moet ook bedacht worden dat menselijke denktijd in het algemeen veel duurder is dan de rekentijd van een computer. Bovendien resulteert een bijzonder slimme en snelle oplossing van een probleem vaak in een programma dat slechts ten koste van veel menselijke inspanning is aan te passen aan een iets afwijkend probleem. Dit zou betekenen dat dure mensen een probleem moeten oplossen dat een goedkope machine ook had kunnen doen als die eerste slimme programmeur niet zo (overdreven) zuinig was geweest.
De overweging was in mijn geval dat ik inspanning wou steken in het tekenen van de boom, niet in het sorteren. Het betekende echter wel dat mijn oplossing voor het sorteren geen prijs waard was.

Het tekenen van de boom is echter een heel ander verhaal. Een ieder die een zich steeds vertakkende boom gaat tekenen, zal tot de ontdekking komen dat de vertakkingen elkaar op de tekening in de weg gaan zitten. Hij kan dit oplossen door de vertakkingen steeds kleiner te tekenen. Zoals een echte boom in steeds kleinere takjes en twijgjes vertakt. Deze oplossing is hier uitgesloten want de letters die de schrijfmachine van de computer afdrukt hebben een vaste grootte.
Een andere oplossing is het begin van de boom zo groot te tekenen dat de kleinste takjes keurig naast elkaar getekend

kunnen worden. Deze oplossing is hier uitgesloten vanwege de begrensde afmeting van het papier.
De overblijvende oplossing is, het gedeelte van de boom dat op een pagina past daar te tekenen. Bij een vertakking die niet meer op een pagina past zetten we een merktekentje dat verwijst naar een pagina waarop het resterende gedeelte van de boom getekend wordt, eventueel ook weer met merktekentjes.
Hiervoor gebruikte ik de mogelijkheid van de class SIMULATION om processen te definieren die pas op een later tijdstip uitgevoerd moeten worden. Ik schreef een process PAGE dat een boom tekent voor zover die op een pagina past, uitgaande van de top van die boom (*). De eerste page wordt gestart met de top van de gehele boom op tijdstip nul. Hieronder volgt een beschrijving van het process PAGE:

PAGE:

PAGE bevat een array waarin het beeld van een pagina opgebouwd wordt (TEXT ARRAY LINE, regel 38). Dit array wordt op een pagina afgedrukt als laatste handeling in het process. De pagina kan vier lagen van vertakkingen bevatten. De vijfde laag bestaat uit verwijzingen naar andere paginaas. Dit wordt bijgehouden door de integer variabele LEVEL (regel 38). Verder bevat de class PAGE een recursieve procedure PLACE (regel 39 tot 76) die een element tekent. Deze procedure heeft een parameter LEAF (=het element dat getekend moet worden) en POS (=de regelpositie waar het element getekend moet worden). De vertakkingslaag wordt bijgehouden door LEVEL. De stappen in de class PAGE zijn:

- vul het tekst array met blanke regels, de spaties worden later vervangen door de uiteindelijke symbolen.
- schrijf het merkteken in het midden van de eerste regel (in het array).
- roep procedure PLACE aan.
- print het array LINE (het tekst array).

PLACE:

De procedure PLACE bevat de volgende stappen:

- verhoog LEVEL (vertakkingslaag), nu zijn er twee mogelijkheden:
  - we zouden vertakkingslaag 5 moeten tekenen, maar daar is geen ruimte meer voor op deze pagina. Nu genereren we een nieuw process PAGE die de boom moet tekenen beginnend bij

---

(*) Computermensen tekenen een boom doorgaans van boven naar beneden. Het begin van de boom, de 'root' kan dan ook 'top' genoemd worden.

het onderhavige element. Dit process moet beginnen een tijdseenheid na het laatste process dat op de tijdsas staat.

- LEVEL is kleiner dan 5. Nu wordt het element getekend op positie POS. Als een linkertak aanwezig is, wordt PLACE aangeroepen voor het linkerelement. Hetzelfde gebeurt voor de rechtertak.

De parameter POS voor de procedure PLACE wordt als volgt berekend. De totale regelbreedte is 130 posities. Het topelement moet midden-boven in de pagina getekend worden, dus om positie 65. De twee takken moeten de helft van de beschikbare ruimte naar links en rechts uitsteken. De lengte van die takken is dus 65 gedeeld door 2. Op de volgende lagen is de beschikbare ruimte gelijk aan de lengte van de verwijzende takken. De taklengte is dus alleen afhankelijk van de variabele LEVEL. De algemene formule voor de taklengte is dan

62/2**LEVEL

zie programmatekst regel 69.

De manier waarop het tekenen van zo'n boom hier geprogrammeerd is, is ongeveer gelijk aan het denken van de mens. Het programma begint welgemoed af te werken wat het kan, en stelt de zaken die niet passen uit tot een tijdstip waarop die zaken afgehandeld kunnen worden. Eigenlijk hebben we bij dit probleem helemaal geen tijdas nodig. Het is mogelijk om de processen in een lijst te zetten en die lijst sequentieel af te werken (*). Maar dit vereist extra programmeerwerk om die lijst te behandelen. Terwijl in de class SIMULATION een en ander overeenkomstig het menselijk voorstellingsvermogen al klaar staat.

(*) In feite is dit de wijze waarop de class SIMULATION geprogrammeerd is