© 1999-2003, Flemming Koch Jensen
Alle rettigheder forbeholdt

Algoritmer

Sidst ændret: 07/19/2018 14:11:03
Opgaver

"Parkering foran containeren forbudt"

- Den grå container

Abstract:

Først ser vi på "udsmidnings"-algoritmen. Med udgangspunkt i den, introduceres nedbrydning som den væsentligste konstruktionsteknik, og begrebet tilstande berøres. Dernæst introduceres kontrolstrukturerne til styring af algoritmers forløb. Til slut præciseres begrebet algoritme med en definition.

Forudsætninger:

At man er moden nok til at beskæftige sig med programmering som mere end ad hoc programmering, samt at man er villig til at abstrahere fra et konkret programmeringssprog.

Der var engang en ingeniør og en datalog. De var ude at gå en tur i skoven. På et tidspunkt kom de til en lysning, med en lille hytte. Hytten var i brand og ingeniøren viste at han havde fået en praktisk anvendelig uddannelse ved resolut at gå hen til en lille brønd der var ved huset. Her tog han den tomme spand, sænkede den ned i brønden og med vandet slukkede han heroisk branden, mens datalogen beundrende så på.

En dag traf det sig at datalogen var ude at gå alene, og igen kom han frem til lysningen med den lille hytte. Også denne dag havde ilden besluttet sig for at hjemsøge det lille hjem, og datalogen ihukom ingeniørens fremgangsmåde. Han samlede sit mod og gik hen til brønden. Her fandt han en spand fyldt med vand, men da han skulle bruge en tom spand til at sænke ned i brønden, tømte han vandet ud på jorden. Dernæst sænkede han spanden ned i brønden og med det derved tilvejebragte vand slukkede han ilden.

Formanden for "Anonyme Ingeniører"

Og hvad kan man så lære af det: At have stor respekt for ingeniører!

Hele idéen med at drille EDB-folk (og vi er *ikke* spor provokerede) med ovenstående historie ligger i en generalisering af den tænkemåde man bruger når man programmerer eller konstruerer algoritmer. Når vi taler om **algoritmer** frem for programmer, er det fordi vi vil se på

løsningsmetoder der er uafhængige af programmeringssprog, for først derefter at realisere dem i Java.

"Algoritme" stammer fra arabisk

Ordet algoritme stammer ikke som så mange andre videnskabelige fremmedord fra latin, men fra en arabisk matematiker ved navn **Abu Jafar Muhammad Ibn Musa Al-Khowarizmi**, fra det niende århundrede. Den sidste del af hans navn er kilden til ordet "algoritme".

Det er forøvrigt et billede af denne hr. Algoritme, der pryder oversigten til venstre.

1. Problemløsning

Lad os se et eksempel på en algoritme, der med vilje er valgt så det første man kommer til at tænke på *ikke* er computere. Så læg kodebrillerne for en stund.

Pedel-funktionen varetager udsmidning af studerende

Her på skolen har vi en veludviklet pedel-funktion. En af de vigtige funktioner de varetager i det daglige er udsmidning af uønskede studerende. Når en studerende skal bortvises tager en af pedellerne fat i subjektet, der indledningsvis anbringes med fødderne i et helsende mudderbad, indholdende en relativ høj koncentration af cement. Efter den studerende har nydt gavn af dette bliver han af pedellen ført ud på parkeringspladsen; hvor han anbringes i containeren til gråt affald. Den studerende henstår i containeren indtil en eventuel appel om genoptagelse er behandlet, det kan dog tage nogle dage.

Det var en **løs beskrivelse** af proceduren for bortvisning af studerende. Man vil nu lave en **struktureret beskrivelse**, en algoritme, så denne fagviden ikke vil gå tabt for kommende generationer. Den kunne se ud som følger.

Pseudo 1:
Udsmidning på to linier

```
Støb den studerende i cement

Placer den studerende i containeren til gråt affald
```

Nedbrydning

Selv om de nuværende pedeller sikkert vil betragte ovenstående algoritme som fyldestgørende, er den det ikke nødvendigvis for kommende pedeller. De kunne måske være i tvivl om hvordan man rent praktisk støber en studerende i cement. Og det at få manøvreret den studerende op i den grå container, inkl. alternativ fodbeklædning, kræver måske også en forklaring. Vi vil bibeholde de to linier i algoritmen som en slags overskrifter, og **nedbryde** dem i mindre dele.

```
Støb den studerende i cement

   Lav cementblanding i baljen

   Anbring den studerendes fødder i baljen
```

Pseudo 2:
Nedbrydning af delagoritmer

```
Placer den studerende i containeren til gråt affald

   Flyt den studerende ud til containeren

   Åben containeren

   Sving den studerende op i containeren

   Luk så vidt muligt containeren
```

Se, nu er det straks mere klart. Spørgsmålet er dog om alle pedel-aspiranter vil være i stand til at udføre denne algoritme. Ved de hvordan man laver en cementblanding? Er de i stand til at svinge en studerende, inklusiv hinkesten, op i containeren? Det er alt sammen spørgsmål vi ikke behøver at bekymre os om, for når vi her på skolen søger nye pedeller er der altid anført under ønskede kvalifikationer.

Ønskede pedel-kvalifikationer

"... Kan du lide at arbejde med unge mennesker ..."

"... Har du sunde fritidsinteresser som f.eks. græsk-romersk brydning ..."

"... Er du bare m/k, der ved hvordan man blander cement, så har vi..."

Det forhindre dog ikke at man pga. en evt. pedelmangel i de kommende år kommer til at lave en mere detaljeret beskrivelse til mindre egnede aspiranter. Lad os derfor foretage en **yderligere nedbrydning**, denne gang af nogle udvalgte linier.

```
Lav cementblanding i baljen

   Tag baljen
   Fyld sand i baljen
   Fyld cementpulver i baljen
   Fyld vand i baljen
   Rør rundt i baljen
```

Pseudo 3:
Yderligere nedbrydning

```
Anbring den studerendes fødder i baljen

   Løft den studerende op
   Før den studerende hen over baljen
   Slip den studerende
   Berolig den studerende med din tilstedeværelse
```

```
Åben containeren
   Tag fat i håndtaget
   Løft låget helt op
   Slip håndtaget
```

På denne måde kan man blive ved med at nedbryde de enkelte skridt i en uendelighed.

Ovenstående konstruktion af bortvisningsalgoritmen dækker i al sin enkelhed en række centrale emner omkring algortimer og deres konstruktion. Dem vil vi se nærmere på i det følgende.

2. Nedbrydning

Nedbryde til forståelig for rette vedkommende

Princippet med at nedbryde en algoritme til den er forståelig for rette vedkommende, er det grundlæggende princip i algortimekonstruktion og enhver anden form for problemknuservirksomhed. Den generelle skabelon er:

Pseudo 4:
Generel

```
Løs problem
   Løs del-problem 1
   Løs del-problem 2
```

nedbrydning af problem

```
    ...
   Løs del-problem n
```

Hvor "Løs problem" deles i n delløsninger, der igen kan nedbrydes i m delløsninger:

Pseudo 5:
Generel videre nedbrydning

```
Løs del-problem 4
   Løs del-problem 4.1
   Løs del-problem 4.2
    ...
   Løs del-problem 4.m
```

På denne måde kan man fortsætte indtil algoritmen kan forstås af den der skal udføre den.

Sekventielt

Man siger at delalgoritmerne til løsning af delproblemerne er ordnet **sekventielt**, når de kommer efter hinanden som **perler på en snor**. Denne måde at opbygge algoritmer på, giver en række begrænsninger. Lad os igen se på en af delalgoritmerne fra bortvisningsalgoritmen.

Pseudo 6:
Algoritme uden angivelse af tilstande

```
Lav cementblanding i baljen
   Tag baljen
   Fyld sand i baljen
   Fyld cementpulver i baljen
   Fyld vand i baljen
   Rør rundt i baljen
```

Tilstanden før og efter

De tre Fyld-linier kommer i logisk orden når man skal blande cement, men hvad nu hvis baljen ikke var tom? De tre Fyld-kommandoer er **afhængige af**, at en bestemt **tilstand** er til stede, for at de kan udføres med det ønskede resultat. Den krævede tilstand er, at der står en tom balje klar. De tre Fyld-kommandoer er også afhængige af hinanden. De tilvejebringer hver for sig den tilstand som den følgende delalgoritme skal bruge for at virke, ud fra den tilstand som delalgoritmen selv arbejder ud fra. Lad os prøve at sætte tilstande ind mellem delalgoritmerne og se hvordan det fungerer.

Pseudo 7:
Algoritme med angivelse af tilstande

```
Lav cementblanding i baljen
    Der er en tom balje
   Tag baljen
    Der står en tom balje på gulvet
   Fyld sand i baljen
    Der står en balje med sand på gulvet
   Fyld cementpulver i baljen
     Der står en balje med sand og cementpulver på gulvet
   Fyld vand i baljen
     Der står en balje med sand, cementpulver og vand på gulvet
   Rør rundt i baljen
    Der står en balje med våd cement på gulvet
```

Delalgoritmer ændrer

Tilstandene kunne godt være gjort mere detaljerede. F.eks. er det for "Fyld sand i baljen" også nødvendigt, at der er noget sand. Det man ser er at delalgortimer stiller visse krav til den tilstand de skal arbejde ud fra, men at de så til gengæld **ændrer tilstanden** på en måde som vi så kan bruge

tilstanden

i den videre algoritme. På samme måde er det for hele den samlede algoritme. Den kræver også visse ting for at kunne fungere. I vores udsmidningsalgoritme ville dette være.

Pseudo 8:
Hele algoritmens start- og slut-tilstand

Der er en studerende, der skal smides ud. Der er sand, cementpulver, vand og en tom balje

```
Bortvis en studerende
```

Den studerende står i den grå container med fødderne støbt i cement

3. Kontrolstrukturer

Først tilvejebringe den ønskede tilstand

Historien om ingeniøren og datalogen har som tidligere nævnt sin rod i en sekventiel algoritme-idé. Når de enkelte delalgoritmer er afhængige af én og netop én tilstand, og dermed er meget lidt fleksible, kan anvendelse af dem kræve at man først bringer "virkeligheden" i den tilstand. Det er netop det datalogen gør ved at tømme spanden. Han skal bruge en tom spand, og det får han. Nu siger denne opfattelse, af algoritmer og deres klodsethed, mere om hvor meget historiens forfatter har lært om programmering, end om de muligheder vi har. Hvis vi kun kunne bruge disse **sekventielle** forløb af delalgoritmer, var vi også dårligt stillet.

Valg mellem flere muligheder

Der er brug for en **kontrol** til at styre anvendelsen af delalgoritmer, til f.eks. ud fra den konkrete situation at vælge mellem flere forskellige delalgoritmer ud fra de krav de stiller til starttilstanden. I historien med spanden kunne man have én delagoritme der skaffede en spand fuld af vand på grundlag af en spand fuld af vand, en rimelig simpel delalgoritme. Man kunne kalde den I-algoritmen. Og en anden algoritme der krævede den starttilstand, at spanden var tom. Vi kunne kalde den D-algoritmen. Eftersom spanden enten er tom eller fuld af vand, er vores styring af de to algoritmer et valg. Vi kunne strukturere det ved flg. formulering af delalgoritmen.

```
Fyld spanden med vand
   hvis spanden er tom
     udfør D-algoritmen
   ellers
     udfør I-algoritmen
```

Pseudo 9:
Valg mellem to del-algoritmer

```
D-algoritmen
   Sæt spanden på krogen
   Sænk spanden ned i brønden
   Træk spanden op af brønden
   Tag spanden af krogen
```

```
I-algoritmen
   Gør ingen ting
```

Konstruktionen med:

Pseudo 10:
"hvis"-strukturen

```
hvis betingelse
   gør noget
ellers
   gør noget andet
```

kaldes en kontrolstruktur. Det den kontrollerer, eller styrer, er andre delalgoritmer og den kommer

derfor selv til at udgøre en algoritme.

Gentagelse

Når man slukker en brand, ved at blive ved med at smide vand på den, så længe det stadig brænder, får man brug for en anden slags styring. Kontrollen skal ligge i, at man gør noget *så længe* en betingelse bliver ved med at være opfyldt. Det bliver altså en algoritmisk konstruktion som:

Pseudo 11: *"så længe"-strukturen*

```
så længe betingelse
   gør noget
```

I vores eksempel kunne det være:

Pseudo 12: *sluk ilden*

```
så længe det brænder
   smid vand på ilden
```

Denne konstruktion vil i sig selv også udgøre en algoritme.

4. Algoritmebegrebet

Betegnelsen algoritme har nu været anvendt flere gange uden at vi har set på en egentlig definition. Vi vil anvende flg.:

Definition: Algoritme

En algoritme er en beskrivelse af en fremgangsmåde for løsningen af et problem, der:

1. Er opdelt i en række veldefinerede skridt, som selv er algoritmer
2. Terminerer efter et endeligt antal skridt

Som det ses af formuleringen er det svært kort og præcist at beskrive hvad en algortime er. Ovenstående er da også mest tænkt som et oplæg til et videre studie af begrebet.

Nedbrydning

Beskrivelsen af et skridt i en algortime som værende endnu en algoritme lyder umiddelbart som et forsøg på at skubbe problemet foran sig, men hvis man husker princippet om konstruktion af algoritmer ved nedbrydning ses der en sammenhæng. Man starter med et skridt "Løs problemet", der kommer til at bestå af en række del-algoritmer. Hver af disse udgør i sig selv en algoritme.

Terminere, er at stoppe

Det andet punkt er, at en algoritme skal terminere efter et endeligt antal skridt. **Terminerer** vil sige at algoritmen stopper. "Efter et endelig antal skridt" vil sige at den stoppe efter at have udført en række skridt der kan tælles. Det lyder måske lidt mærkeligt, men har man først prøvet at lave et program, der ved en fejl går i **uendelig løkke** ved man præcist hvad problematikken er.

Men hvor stopper det hele, man kan ikke fortsætte i det uendelige. I vores udsmidningsalgoritme var målet for nedbrydningen, at pedellerne skulle være i stand til at udføre hvert skridt uden yderligere instruktion, men nu er det ikke pedeller vi skal lave algortimer til men computere, så hvor stopper vi så? Det vil vi se nærmere på i næste kapitel.

Repetitionsspørgsmål

1 Hvorfra stammer ordet "algoritme"?

2 Hvorfor taler vi om algoritmer frem for programmer?

3 Hvad er nedbrydning?

4 Hvad betyder "sekventielt"?

5 Hvad er sammenhængen mellem tilstande og delalgoritmer?

6 Hvad er kontrol-strukturer?

7 Hvordan kan en kontrol-struktur for valg mellem delalgoritmer opbygges?

8 Hvordan kan en kontrol-struktur for gentagelse af en delalgoritme opbygges?

9 Hvad er en algoritme?

10 Hvad betyder at "terminere"?

11 Hvornår stopper man nedbrydning?

Svar på repetitionsspørgsmål

1 Fra en arabisk matematiker.

2 Fordi vi ønsker at se på løsninger der ikke afhænger af programmeringssprog.

3 At man deler et problem i mindre delproblemer; hvis løsning til sammen løser det oprindelige problem.

4 At noget er sekventielt vil sige, at det følger efter hinanden.

5 En delalgoritme stiller krav til tilstanden for at den kan fungere. En algoritme bringer os fra en tilstand til en anden.

6 En algoritmisk struktur der styrer afviklingen af delalgoritmer.

7 Man kan gøre det ved at betinge udførelsen af en delagoritme af at tilstanden opfylder visse krav som man specificerer.

8 Man kan gøre det ved at betinge gentagelsen af en delagoritme af at tilstanden opfylder visse krav som man specificerer (bemærk den store lighed med svaret på spørgsmål 7, på dette abstraktions-niveau).

9 En beskrivelse af løsningen på et problem, der er formuleret i en række skridt og som terminerer.

10 At stoppe.

11 Når man er nede på et detaljerings-niveau; hvor den der skal udføre algoritmen udmiddelbart kan gøre det.