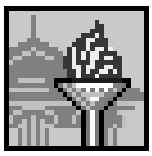

Noter: Om at forstå og bruge algoritmer

- Eller ’algoritmer for ikke-programmører’.

En algoritme er en forskrift! Forskrifter bruger vi mange steder i vores hverdag, ofte uden at tænke over det, når vi udfører dagligdags rutiner som fx at vaske op, børste tænder, brygge kaffe osv. Mere udførligt kan 'algoritme' defineres som i denne tekst fra et amerikansk leksikon:

> ***"An algorithm is a procedure for solving a usually complicated problem by carrying out a precisely determined sequence of simpler, unambiguous steps. Such procedures were originally used in mathematical calculations (the name is a variant of algorism, which originally meant the Arabic numerals and then "arithmetic") but are now widely used in computer programs and in programmed learning."***

Lad os først studere en helt enkel algoritme fra hverdagen, og dernæst diskutere hvorfor forståelsen for algoritmer er så vigtige i forbindelse med computere.

Eksempel for kaffebrygning:

- ***Fyld vand i kaffemaskinen***
- ***Skyl kaffekanden i koldt vand***
- ***Sæt filtertragten på kaffekanden***
- ***Sæt filter i filtertragten***
- ***Hæld kaffe i filteret***
- ***Sæt kaffekanden under vandtuden***
- ***Tænd for strømmen***
- ***Vent til vandet er løbet igennem***

algoritme

SUBST. –*n*, plur. –*r*, -*rne*

/algo’ritme/

(matematik, edb): en samling instruktioner der trin for trin beskriver hvilke operationer der skal udføres for at løse en opgave

Kilde: Politikens NUDANSK ordbog, 1996.

Denne forskrift bryder et større problem op i mindre enheder. Hvert enkelt delproblem er let at udføre, og samtidig sikres det at de enkelte trin forekommer i den helt rigtige rækkefølge (overvej hvad resultatet ville blive hvis punkt 2 eller 5 blev udeladt, og hvis der blev byttet rundt på punkt 4 og 5).

Forskriften, eller algoritmen, skal være utvetydig, effektiv, og praktisk mulig. En algoritme skal desuden altid kunne afsluttes, dvs. den må ikke have åbne, løse ender. Tænk lidt over disse to eksempler fra en kogebogsopskrift:

- ***'smag til med salt og peber'.***
- ***'kog retten til kødet er mørt'.***

Algoritmer kan ofte brydes ned til gradvist mindre dele, med større og større detaljeringsgrad. Dette illustreres af følgende algoritme for 'at udføre et telefonopkald':

Overordnet niveau (for dig der kender en telefon).

1. ***Slå nummeret på den ønskede person op i en telefonbog.***
2. ***Løft telefonrøret.***
3. ***Tast nummeret.***
4. ***Begynd samtalen.***
5. ***Læg telefonrøret på når samtalen er slut.***

Detaljeret niveau (tilføjelser for en person der ikke kender en telefonbog)

1.1. Åben telefonbogen.
1.2. Gå til den side hvor navnene starter med det første bogstav i personens efternavn.
1.3. Find det sted hvor navnene har samme andet bogstav som personens efternavn.
1.4. Fortsæt til du finder navnet på personen, på denne eller følgende sider.
1.5. Aflæs telefonnummeret der står skrevet ved siden af navnet.

Den overordnede rutine kan altså specificeres i mange delrutiner, afhængig af den detaljeringsgrad der er påkrævet. De algoritmer vi har studeret indtil nu, har været helt ***sekventielle***, dvs. ingen punkter overspringes, hvert punkt i algoritmen udføres altid præcis én gang, og i præcis den angivne rækkefølge.

I mere avancerede algoritmer åbnes mulighed for to meget vigtige egenskaber ud over den sekventielle afvikling, nemlig ***valg*** og ***gentagelse***. Der er altså tre væsentlige elementer:

1. ***Sekventielle operationer,*** eller enkelte trin - udfører enkeltoperationer, og når operationen er overstået fortsættes til næste operation.
2. ***Betingede operationer***, eller HVIS… SÅ instruktioner, bruges til at udføre forskellige operationer afhængig af nogle givne forudsætninger, eller betingelser, fx:
   - ***2.1. Hvis opskriften laves til 4 personer, så brug 2 æg.***
   - ***2.2. Hvis opskriften laves til 8 personer, så brug 3 æg… ☺***
3. ***Gentagne operationer,*** eller løkke instruktioner, bruges til at gentage operationer et givet antal gange, eventuelt afhængig af en eller flere betingelser, fx:
   - ***3.1. Gentag følgende to operationer, indtil det totale antal æg svarer til ét for hver person***
   - ***3.2. Tilsæt ét æg***
   - ***3.3. Tilsæt én skefuld flormelis***

? *Hvorfor er algoritmer så vigtige i computerverdenen?*

Algoritmer, computere og programmering…

Alle véd at computere er dumme… Hvorfor er det så så svært at beherske dem? - Det skyldes naturligvis at vi må kompensere for computernes dumhed med vores høje logiske intelligens ☺

Computere skal have instruktion ned til mindste detalje! Vi skal senere tale om, at hver eneste information i en computer rummes i bits, der kan antage værdien sand eller falsk (tændt/slukket). En række af informationer kan udgøre én instruktion, og en lang række instruktioner kan i sidste ende føre til et meningsfuldt og vellykket program. Det kræver altså en overordentlig grundig algoritme at instruere en computer!

Heldigvis behøver den enkelte PC bruger ikke konstruere disse 'endeløse' algoritmer (i så fald havde computeren vistnok ikke den udbredelse vi kender i dag). Det er programmørens opgave!

Værktøjer til overblik

Algoritmerne der er gennemgået ovenfor, kan gøres endnu mere overskuelige ved at sætte de enkelte elementer ind i ***flowdiagrammer***. Det vi du komme til at gøre nogle gange i løbet af året.

Når du har overblik over strukturen for centrale algoritmer i dit program, så kan du vælge at beskrive dem ved hjælp af såkaldt ***pseudokode*** – det vil sige i et sprog der ligger midt imellem 'normaldansk' og 'programmeringssprog'. Ved hjælp af pseudokoden kan du systematisk nærme dig den endelige syntaks for det programmeringssprog du arbejder med.

☞ *Én af de nærmeste gange skal du prøve at arbejde med diagrammer og pseudokode. Det skal bare være små simple eksempler, selv om du kan regne med at få mere brug for værktøjerne des større og komplekst dit program er.*

Husk at algoritmer, flowdiagrammer og pseudokode er uafhængig af det valgte sprog. Det er værktøjer du kan anvende uanset hvilket sprog du vælger at boltre dig i fremover.