Problemlösning och algoritmer

Human Centered Systems
Inst. för datavetenskap
Linköpings universitet

Attribution-NonCommercial

Översikt

- Stegvis förfining
  - Pseudokod
  - Flödesdiagram
- Dekomposition
  - KISS-regeln
  - Procedurell dekomposition
  - DRY-regeln
- Algoritmer
  - Sortering och sökning

Lars Degerstedt
Attribution-NonCommercial-ShareAlike2.5 License
SOME RIGHTS RESERVED

Att konstruera ett program...

"Vi skulle vilja ha..."

"Iterera"

1. Förstå och beskriv problemet/kraven/behoven
2. Gör en principiell lösning för problemet
3. Skriv källkod som implementerar lösningen
4. Testa och utvärdera/verifiera din lösning

Uppgift → **Lösning**

Ok – jag fixar!

Lars Degerstedt
Attribution-NonCommercial-ShareAlike2.5 License
SOME RIGHTS RESERVED

Stegvis förfining

- Metod för att skapa ett program från ett analyserat problem
  - ingen garanti – det är ett *kreativ* process
- Formulera programmet på en enkel informell nivå
  - Börja i vanlig svenska
- Förfina olika steg tills en "mekanisk" nivå är nådd
  - Formalisera din "svenska" gradvis så att den blir alltmer källkodslik
- Kallas även *top-down-design*
- **Pseudokod**: mix av svenska och programmeringstermer
- **Flödesdiagram**: schematisk bild av programmet

Lars Degerstedt
Attribution-NonCommercial-ShareAlike2.5 License
SOME RIGHTS RESERVED

Exempel: skriv ut x^2 i tabell

Uppgift: skapa en tabell med uträkning av x^2 för varje x mellan 1 och 20.

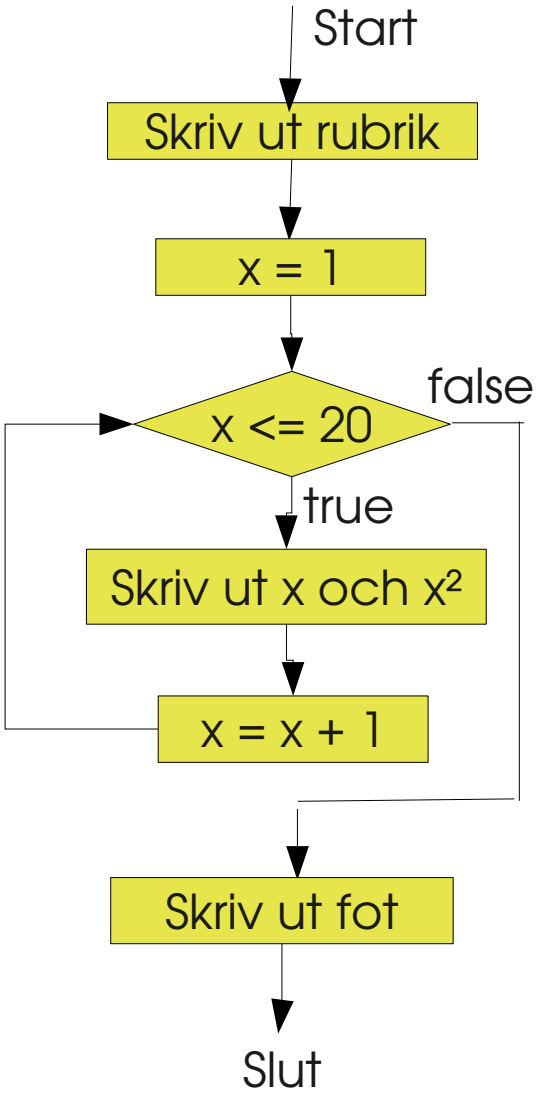
1 -> 1
2 -> 4
3 -> 9
etc

Informell lösning för lösning i konsol:

1. Skriv ut tabellrubrik för tvåkolumntabell
2. För varje tal x mellan 1 och 20:
   skriv ut värdet på x i kolumn ett och värdet på x^2 i kolumn 2
3. Skriv ut tabellfot

Lars Degerstedt
Attribution-NonCommercial-ShareAlike2.5 License
SOME RIGHTS RESERVED

Exempel 2: flödesschema



Lars Degerstedt
Attribution-NonCommercial-ShareAlike2.5 License
SOME RIGHTS RESERVED

Exempel 2: implementation

```
table_size = 10;
print 'x\tx^2\n'

for x in range(1,table_size):
    print x + '\t' + x*x
print '----------'
```

```
x    x^2

1    1
2    4
3    9
4    16
5    25
6    36
7    49
8    64
9    81
---------
```

Lars Degerstedt
Attribution-NonCommercial-ShareAlike2.5 License
SOME RIGHTS RESERVED

Programmeringens hantverk

*"**Controlling complexity** is the essence of computer programming"*

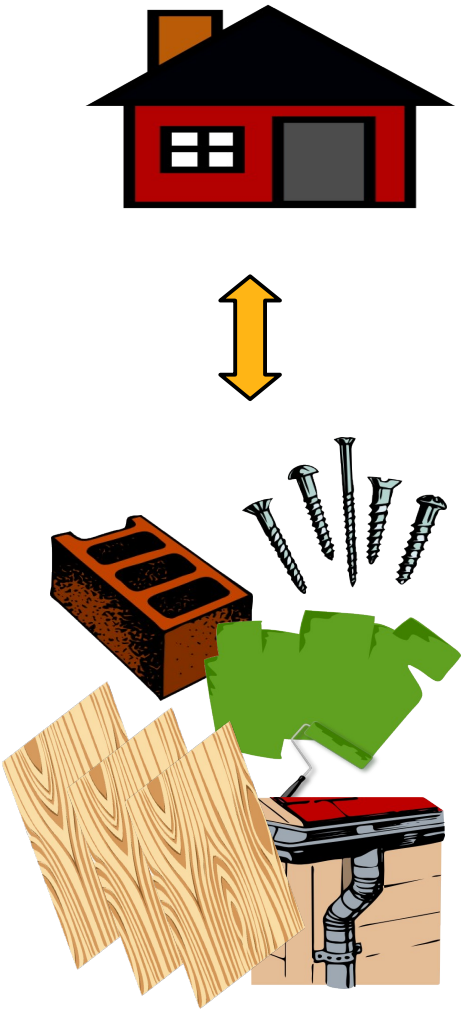
--Brian Kernighan

Lars Degerstedt
Attribution-NonCommercial-ShareAlike2.5 License

SOME RIGHTS RESERVED

Dekomposition

- Uppdelning av ett uppgift/program i delar
- Uppdelningen gör varje del "enklare"
- Varje del bör ha ett tydligt och "enkelt" syfte
- Komposition av delarna ger helheten
- Exempel på dekomposition:
  - Variabler lagrar delresultat
  - Varje sats löser en liten avgränsad uppgift
  - Funktioner separerar ut delar av programmet under nya namn



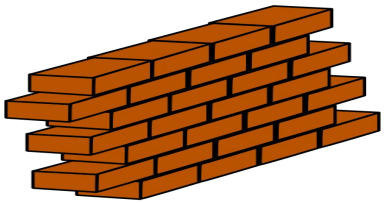
Lars Degerstedt
Attribution-NonCommercial-ShareAlike2.5 License
SOME RIGHTS RESERVED

KISS-regeln: Keep It Simple, Stupid!

Komplex uppgift = kombination av flera enkla deluppgifter

*Genom att **dela upp** programmets totala uppgift i delar håller vi varje deluppgift avgränsat och enkelt. Vi löser större uppgifter genom att kombinera lösningar på flera enklare uppgifter.*

Skriv aldrig komplex källkod – fel sätt att lösa komplexa uppgifter!



"Kumulativ lösning: systematiskt nerifrån och upp"

Lars Degerstedt
Attribution-NonCommercial-ShareAlike2.5 License
SOME RIGHTS RESERVED

Dekomposition på satsnivå – exempel 1

Komplex lösning:

```
print 'Uträknat svar är ' + str((x + y) / 4.3)
```

Uppdelad lösning:

```
result = (x + y) / 4.3
response =  'Uträknat svar är ' + str(result)
print response
```

Lars Degerstedt
Attribution-NonCommercial-ShareAlike2.5 License
SOME RIGHTS RESERVED

Dekomposition på satsnivå – exempel 2

***Uppgift**: vi vill göra en uträkning för varje tal 0..9, samt en annan uträkning för varje jämnt tal.*

Komplex "smart" lösning:

```
results_1 = []
result_2 = []
for x in range(10):
        results_1.append(x * 3 / 5.0)
        if (x % 2 == 0):
                results_2.append(x * 4 / 3.0)
```

Lars Degerstedt
Attribution-NonCommercial-ShareAlike2.5 License
SOME RIGHTS RESERVED

Dekomposition på satsnivå – ex 2 (forts)

Uppdelad lösning:

```
results_1 = []
for x in range(10):
        result = x * 3 / 5.0
        results_1.append(result)
results_2 = []
for x in range(0,10, 2):
        result = x * 4 / 3.0
        results_2.append(result)
```

- Delproblemen tydligare
- Ev lite mindre effektiv
- Mer läsbara segment
- Vi skulle kunna införa två funktioner...
- OBS: vi "slipper" en if-sats

Lars Degerstedt
Attribution-NonCommercial-ShareAlike2.5 License
SOME RIGHTS RESERVED

Procedurell dekomposition

```
def calculate_results_1():
    results_1 = []
    for x in range(10):
        result = x * 3 / 5.0
        results_1.append(result)
    return results_1

def calculate_results_2():
    results_2 = []
    for x in range(0,10, 2):
        result = x * 4 / 3.0
        results_2.append(result)
    return results_2

results_1 = calculate_results_1()
results_2 = calculate_results_2()
present_results(results_1)
present_results(results_2)
```

- Ortogonalitet: en funktion - ett syfte
- Håll isär **presentation** och **innehåll**
  - T ex I/O och uträkningar
- "Göm information" i olika funktioner

Kort och abstrakt/läsbar källkod. Liknar pseudokoden!

Lars Degerstedt
Attribution-NonCommercial-ShareAlike2.5 License
SOME RIGHTS RESERVED

DRY-regeln: Don't Repeat Yourself

DRY = Varje bit av information ska finnas representerad endast en gång i ett program.

***Exempel på information**: namn på användare, färger i gränssnittet, storlek på fonter, uträkningar, format, etc.*

***Motivering**: Duplicering av information gör det svårt att underhålla ett program. Vid ändring av ett program vill man ändra enbart på ett ställe.*

Lars Degerstedt
Attribution-NonCommercial-ShareAlike2.5 License
SOME RIGHTS RESERVED

DRY genom att införa en funktion

```
Hej värld!
++++++
Hej värld!
```

En lösning med två förekomster:

```
print 'Hej Värld!'
print '++++++'
print 'Hej Värld!'
```

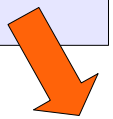
- Vi skriver ut Hej värld två gånger
- Däremellan skriver i ++ +++
- Hur få uskriften att bara finnas på **ett** ställe i källkoden??

Lars Degerstedt
Attribution-NonCommercial-ShareAlike2.5 License
SOME RIGHTS RESERVED

DRY - genom att införa en funktion (forts)

```
def hello_world()
    print 'Hej värld!'

helloWorld()
print '++++++'
helloWorld()
```



```
Hej värld!
++++++
Hej värld!
```

- **två** anrop görs till **samma** funktion
- 'Hej värld' står bara på ett ställe i programmet

Lars Degerstedt
Attribution-NonCommercial-ShareAlike2.5 License
SOME RIGHTS RESERVED

Algoritmer

Sortering, sökning och att göra egna

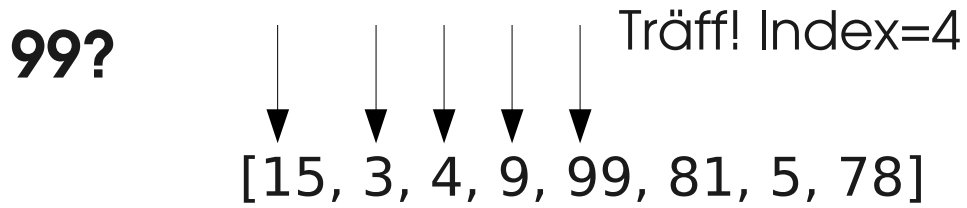
Lars Degerstedt
Attribution-NonCommercial-ShareAlike2.5 License

Att skapa och studera algoritmer

- Hitta rätt uppgift - "systemdesign"
  - Ett bra användargränssnitt
  - "rätt program" - det som ofta behövs...
- Hitta rätt lösning – skapa en algoritm
  - Rätt och bra lösningar för kända uppgifter/problem.
  - Det finns normalt flera bra lösningar för en viss uppgift
  - Algoritmer är (i princip) oberoende av programmeringsspråk
  - Vi lär oss skapa algoritmer bl a genom att studera kända sådana , t ex sökning och sortering
- I Laboration 5-6 finns uppgifter att arbeta med liststrukturer som är "algoritmiskt svåra"

Lars Degerstedt
Attribution-NonCommercial-ShareAlike2.5 License
SOME RIGHTS RESERVED

Linjär sökning – i osorterad vektor



5 jämförelser:
värsta fallet 8
*dvs **linjärt** i termer av antal element*

```
def linear_search(x, seq):
    for index in range(len(seq)):
        if x == seq[index]:
            return index
    return -1
```

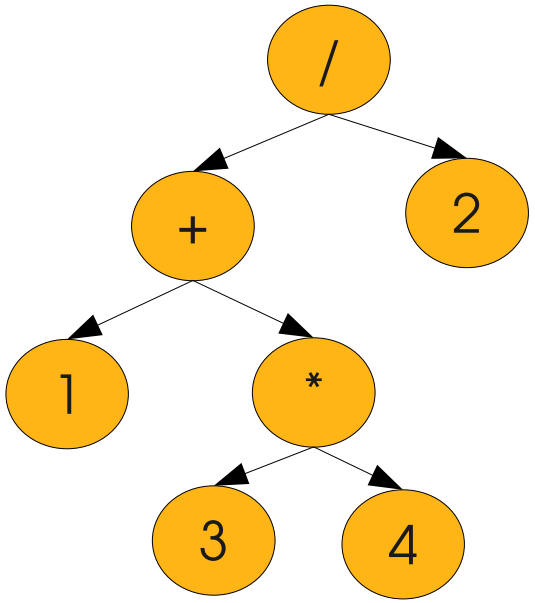
Lars Degerstedt
Attribution-NonCommercial-ShareAlike2.5 License
SOME RIGHTS RESERVED

Trädstrukturer – vanligt begrepp inom programmering

Uttryck/term:

(1 + (3 * 4)) / 2

Trädstruktur:



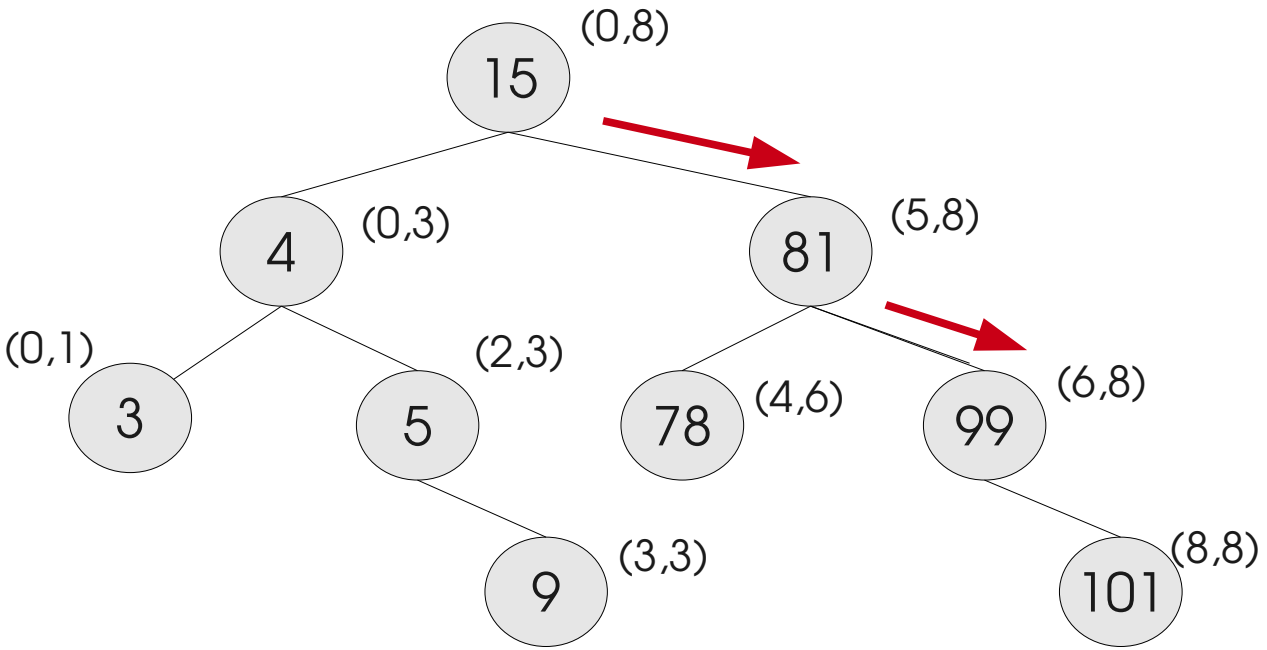
- Trädstrukter används
  - t ex för att beskriva uttryck/termer
  - Hur program arbetar
- Grafer består av **noder** och **bågar**
- Träd är en graf där vi har
  - En nod utan ingående bågar, **roten**
  - Riktade pilar utan loopar och flätor

T ex som lista av listor: [[1, '+', [3, '*', 4]], '/', 2]

Lars Degerstedt
Attribution-NonCommercial-ShareAlike2.5 License
SOME RIGHTS RESERVED

Binär sökning – i sorterad vektor

Hitta 99? [3, 4, 5, 9, 15, 78, 81, 99, 101]



- Intervallhalvering
- testa mot elementet i mitten
- Skär bort den halva där elementet inte kan finnas
- Sluta när elementet är nått eller arrayen tom

2 jämförelser
värsta fallet log_2(9)
*dvs **logaritmiskt** i termer av antal element*

Lars Degerstedt
Attribution-NonCommercial-ShareAlike2.5 License
SOME RIGHTS RESERVED

Implementation - bin_search

```
def bin_search(x, seq):
    low = 0
    high = len(seq) - 1
    while low <= high :
        mid = (low + high) / 2
        if seq[mid] < x:
            low = mid + 1
        elif seq[mid] > x:
            high = mid - 1
        else:
            return mid;
    return -1;
```

Lars Degerstedt
Attribution-NonCommercial-ShareAlike2.5 License
SOME RIGHTS RESERVED

Sortering

- Givet en osorterad vektor sortera dess element
  - Ex: "oordnat" datamaterial – sortera upp det tidsmässigt + områdesmässigt
  - Sorterar ofta på "nyckel" dvs en specifik kolumn
- "in place": utan att införa hjälpvektorer
- swap: byta plats på två element i en array
- I praktiken: ofta bra att hålla data sorterat
  - enklare att hitta både för algoritmer och människor
  - specialfall: ett nytt element i sorterad vektor

Lars Degerstedt
Attribution-NonCommercial-ShareAlike2.5 License
SOME RIGHTS RESERVED

Bubble Sort

Exempel:

:: 8 6 11 9 3 6 2 7 4
2 :: 8 6 11 9 3 6 4 7
2 3 :: 8 6 11 9 4 6 7
2 3 4 :: 8 6 11 9 6 7
2 3 4 6 :: 8 6 11 9 7
2 3 4 6 6 :: 8 7 11 9
2 3 4 6 6 7 :: 8 9 11
2 3 4 6 6 7 8 :: 9 11
2 3 4 6 6 7 8 9 11 ::

- Iterera vä-höger
- För varje loop låt det parvis minsta elementet "bubbla" åt vänster
- Upprepa tills alla element bubblat
- Nackdel: många swaps

Lars Degerstedt
Attribution-NonCommercial-ShareAlike2.5 License
SOME RIGHTS RESERVED

Selection Sort

Exempel:

:: 8 6 11 9 3 6 2 7 4
2 :: 6 11 9 3 6 8 7 4
2 3 :: 11 9 6 6 8 7 4
2 3 4 :: 9 6 6 8 7 11
2 3 4 6 :: 9 6 8 7 11
2 3 4 6 6 :: 9 8 7 11
2 3 4 6 6 7 :: 8 9 11
2 3 4 6 6 7 8 :: 9 11
2 3 4 6 6 7 8 9 11 ::

- Input: array av längd n
- Hitta minsta elementen och lägg det först
- Efter i steg är de i första sorterade. Upprepa för arrayen i+1 till n

Lars Degerstedt
Attribution-NonCommercial-ShareAlike2.5 License
SOME RIGHTS RESERVED

Implementation av selection sort

```
def selection_sort(seq):
    for i in range(0, len (seq)):
        low = i
        for j in range(i + 1, len(seq)):
            if seq[j] < seq[low]:
                low = j
        seq[i], seq[low] = seq[low], seq[i]
```

Lars Degerstedt
Attribution-NonCommercial-ShareAlike2.5 License
SOME RIGHTS RESERVED

Quicksort

Pivotvärdet

[8, 6, 11, 9, 3, 6, 2, 7, 4]

[4, 7, 2, 6, 3, 6]::[8]::[9, 11]

[3, 2]::[4]::[6, 6, 7]

[2]::[3]::[]

[]::[2]::[]

[]::[6, 6]::[7]

[]::[7]::[]

[]::[9]::[11]

[]::[11]::[]

[2, 3, 4, 6, 6, 7, 8, 9, 11]

mindre-än::lika-med::större-än

- Hitta ett pivot-värde och sortera runt det
- Upprepa för varje dellista
- Snabbare i genomsnitt än Bubble och Selection
- Naturligt en rekursiv lösning
  - Funktionen anropar sig själv för varje dellista

Lars Degerstedt
Attribution-NonCommercial-ShareAlike2.5 License
SOME RIGHTS RESERVED

Beräkningskomplexitet

- Ordo-funktionen: antal exekveringssteg i termer av indatas storlek
  - 1) Vad beskriver *variabel* storlek på indata?
  - 2) Räkna steg givet en viss obekant storlek "n"
  - summa-index/Arraylängd - möjliga variable indata-storheter
- Exempel på Ordo:
  - Linjär sökning: O(n) Binärsökning: O(log n)
  - Bubblesort, Selectionsort och Quicksort är i värsta fallet $O(n^2)$
  - Quicksort: O(n log(n)) i medelfallet (vilket är bäst)
  - "Delmängd-problem" är O(2^n)

Lars Degerstedt
Attribution-NonCommercial-ShareAlike2.5 License
SOME RIGHTS RESERVED

Summering

- Stegvis förfining: från uppgift till lösning med pseudokod och flödesdiagram
- Dekomposition: att konstruera program på rätt sätt
- KISS och DRY-regeln: håll det enkelt och duplicera inte
- Lär dig algoritmer genom att studera sökning och sortering - komplexitet

Lars Degerstedt
Attribution-NonCommercial-ShareAlike2.5 License
SOME RIGHTS RESERVED