Föreläsning 1 i ADK

Första timmen: kursöversikt

Andra timmen: algoritmanalys, beräkningsmodeller mm

Viggo Kann och Stefan Nilsson

KTH

ADK-kursinnehåll

Algoritmer – *hur löser man problem effektivt?*
beräkningsmodeller, konstruktionsmetoder,
analys av komplexitet och korrekthet,
exempel på olika slags algoritmer
Datastrukturer – *hur lagrar man data effektivt?*
Teoretiskt sett effektiva datastrukturer
Praktiskt sett effektiva datastrukturer
Komplexitet – *hur svårt är det att lösa problem?*
Lätta, svåra och oavgörbara problem
Reduktioner – omformulering av problem
Komplexitetsklasser: P, NP, PSPACE, RP, APX, NPO

Vad som krävs av dig

- Kursregistrera dig och acceptera Canvaskursrummet
- Följ undervisningen (om du inte vill läsa in själv)
- Plugga under hela kursen
- Gör datorlabb 1–5 med teoriuppgifter (i par)
- Lös mästarprov 1 och 2, skriftligt och muntligt
- Gör teoritentan
- För högre betyg: extralabb (och muntlig tenta för plussning av mästarproven)

Pedagogiskt upplägg

- Studera på det sätt som är effektivast för dig!
- Koncentrerade entimmesföreläsningar med läsanvisningar.
  Kom förberedd och var vaken för bästa resultat!
- Omvänd undervisning används i vissa svårare avsnitt i kursen.
- Övningsuppgifter med lösningar. Ett urval av övningsuppgifterna löses på övningarna. Resten lämnas för egen övning.
- Momenten i kursen tränar verkliga arbetssituationer.
- Aktiverande färgfrågor på föreläsningarna.
- Du förbereds väl för mästarproven med övningsmästarprov, bedömningskriterier och autentiska exempel på tidigare studentinlämningar med kommentarer.
- Undervisning byggd på pedagogisk forskning –
  en hel doktorsavhandling om ADK las fram 2014!
- Målrelaterade betygskriterier; välj själv betyg!

Förändringar i kursen (i urval)

- Ny större textfil att söka i till labb 1.
- Stöd för testning har lagts in i den givna programkoden för labb 2.
- Labbredovisningar och mästarprovsredovisningar erbjuds både fysiskt och digitalt.
- Olika långa mästarprovsredovisningar beroende på hur många uppgifter som ska redovisas.
- Nya teoritentaförberedelsequizfrågor.
- Vi påminner om att ADK-kursens fokus inte är programmering utan teoretisk algoritmdesign.
- Kommunikation sker i första hand genom diskussionsforumet i Canvas.

Ta vara på återkopplingen

- . . . vid färgfrågorna på föreläsningarna
- . . . på dina frågor vid föreläsningar/övningar
- . . . vid kamratredovisningen av labbteoriuppgifterna
- . . . från Kattis
- . . . vid labbredovisningarna
- . . . vid övningsmästarprovet
- . . . vid mästarprovsredovisningarna
- . . . vid teoritentagenomgången

Relevanta förkunskaper i datalogi

- *Algoritmanalys*
  asymptotisk komplexitet angiven med $O(\ldots)$
- *Algoritmbeskrivning*
  pseudokod, motivering
- *Algoritmer*
  sortering (insättnings-, urvals-, quicksort)
  sökning (binär-, träd-, hashning)
- *Datastrukturer*
  listor, köer, stackar, mängder, binärträd,
  prioritetsköer (heapar), hashtabeller

Föreläsning 1, del 2

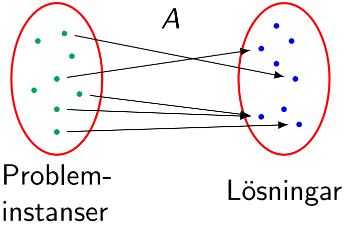
Repetition av algoritmanalys, beräkningsmodeller, bitkostnad, enhetskostnad

Stefan Nilsson

KTH

Algoritm

- Definition:
  - En **algoritm** är en ändlig beskrivning av hur man steg för steg löser ett problem
- En algoritm tar oftast **indata** som beskriver en **probleminstans** och producerar **utdata** som beskriver probleminstansen **lösning**
- En algoritm kan ses som en funktion
  - A : Probleminstanser $\rightarrow$ Lösningar



Analys av algoritmer

Tidskomplexitet

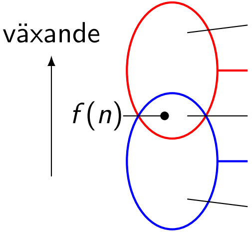
- Hur lång tid tar algoritmen i värsta fallet?
  - Som funktion av vad?
  - Vad är ett tidssteg?

Minneskomplexitet

- Hur stort minne behöver algoritmen i värsta fallet?
  - Som funktion av vad?
  - Mätt i vad?
  - Tänk på att funktions- och proceduranrop också tar minne

Hur komplexitet kan anges

- Hur ändras komplexiteten för växande storleken n på indata?
- Asymptotisk komplexitet
  - Vad händer när n växer mot oändligheten?
- Mycket enklare om vi bortser från konstanta faktorer



$\omega(f(n))$ växer snabbare än $f(n)$

$\Omega(f(n))$ växer minst lika snabbt som $f(n)$

$\Theta(f(n))$ växer lika snabbt som $f(n)$

$\mathcal{O}(f(n))$ växer högst lika snabbt som $f(n)$

$o(f(n))$ växer långsammare än $f(n)$

$$\lim_{n\to\infty}\frac{g(n)}{f(n)} = \begin{cases} 0 & \text{om } g(n)\in o(f(n)) \\ c>0 & \text{om } g(n)\in \Theta(f(n)) \\ \infty & \text{om } g(n)\in \omega(f(n)) \end{cases}$$

$\mathcal{O}(f(n))$

$\Omega(f(n))$

Exempel: Binärsökning

```
function BinSearch(v[a..b],x)
    if a < b then
        m ← ⌊(a+b)/2⌋
        if v[m].key < x then
            return BinSearch(v[m+1..b], x)
        else
            return BinSearch(v[a..m], x)
    if v[a].key = x then
        return a
    else
        return 'Not Found'
```

Exempel: Binärsökning

Analys:

- Låt $T(n) =$ Tiden att i värsta fall söka bland n tal med binsearch
- $T(n) = \begin{cases} \Theta(1) & \text{om } n = 1 \\ T(\lceil \frac{n}{2} \rceil) + \Theta(1) & \text{om } n > 1 \end{cases}$
- Om $n = 2^m$ får vi $T(n) = \begin{cases} \Theta(1) & \text{om } n = 1 \\ T(\frac{n}{2}) + \Theta(1) & \text{om } n > 1 \end{cases}$
- Mästarsatsen (Master Theorem) säger då:
  - $n^{\log_2 1} = n^0 \in \Theta(1)$
  - Då är $T(n) = \underline{\Theta(\log n)}$

Lösning till vanliga rekursionsekvationer

Sats ("Master Theorem"):

Om $a \geq 1, b > 1$ samt $d > 0$ så har rekursionsekvationen

$$\begin{cases} T(n) = aT(\frac{n}{b}) + f(n) \\ T(1) = d \end{cases}$$

den asymptotiska lösningen:

- $T(n) = \Theta(n^{\log_b a})$ om $f(n) = \mathcal{O}(n^{\log_b a - \varepsilon})$ för något $\varepsilon > 0$
- $T(n) = \Theta(n^{\log_b a} \log n)$ om $f(n) = \mathcal{O}(n^{\log_b a})$
- $T(n) = \Theta(f(n))$ om $f(n) = \mathcal{O}(n^{\log_b a + \varepsilon})$ för något $\varepsilon > 0$
  och $a \cdot f(\frac{n}{b}) \leq c \cdot f(n)$ för någon konstant $c < 1$
  för alla tillräckligt stora n

Analys av problem

Ringa in ett problems komplexitet!

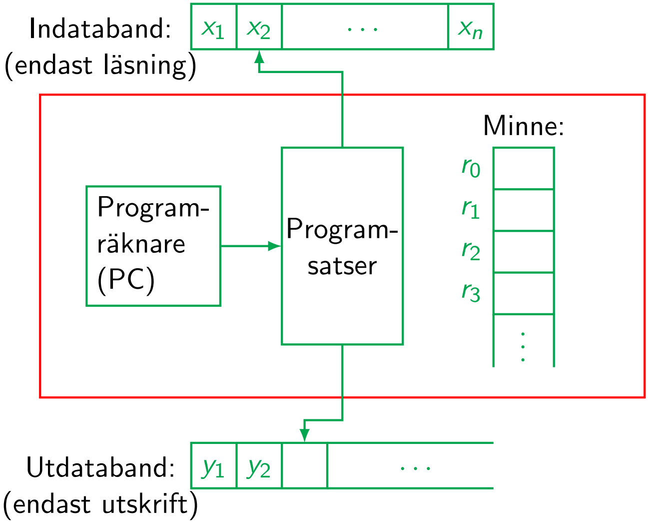
Övre gräns:

- Ge en algoritm som löser problemet
- Algoritmens komplexitet är en **övre gräns** för problemets komplexitet

Undre gräns:

- Ofta svårt att ange
- Egenskaper hos problemet måste användas
- Exempel:
  - Måste titta på all indata $\Rightarrow \Omega(n)$
  - Måste producera hela utdata
  - Beslutsträd - ett visst antal olika fall måste särskiljas

Beräkningsmodell 1: RAM (Random Access Machine)



Beräkningsmodell 1: RAM (Random Access Machine)

- Programmet består av vanliga satser som utförs sekvensiellt (inte parallellt)
- Varje sats kan bara läsa och påverka ett konstant antal minnesplatser
- Bara en symbol kan läsas/skrivas i taget
- Varje sats tar konstant tid

På grund av $\mathcal{O}()$-notationens robusthet kan vi strunta i vilka värden konstanterna har

Kostnadsmått

Enhetskostnad:

- Varje operation tar en tidsenhet
- Varje variabel tar en minnesenhet
- Beräkningsmodell: RAM

Bitkostnad:

- Varje bitoperation tar en tidsenhet
- Varje bit tar upp en minnesenhet
- Beräkningsmodeller
  - RAM med begränsad ordlängd
  - Turingmaskin
- Används när algoritmen räknar med tal av godtycklig storlek

Exempel: Addition av två n-bitsheltal

- Tid $\mathcal{O}(1)$ med enhetskostnad
- Tid $\mathcal{O}(n)$ med bitkostnad