Algoritmer, datastrukturer och komplexitet, föreläsning 1

Första timmen: kursöversikt

Andra timmen: algoritmanalys, beräkningsmodellen RAM

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 dej

- Gör kursregistrering i personliga menyn på KTH-webben
- Följ undervisningen (om du inte vill läsa in själv)
- Plugga under hela kursen
- Gör datorlabb 1-4 med teoriuppgifter
- Lös mästarprov 1 och 2, skriftligt och muntligt
- Gör teoritentan
- För högre betyg: muntlig tenta eller extralabb

Kursens pedagogik

- Studera på det sätt som är effektivast för dej!
- Koncentrerade entimmesföreläsningar med läsanvisningar.
  Kom förberedd och var vaken för bästa resultat!
- Övningsuppgiftshäfte med lösningar
- Momenten i kursen tränar verkliga arbetssituationer
- Aktiverande färgfrågor på föreläsningarna
- Undervisning byggd på pedagogisk forskning - ett pedagogiskt forskningsprojekt pågår under ADK
- Målrelaterade betygskriterier; välj själv betyg!

Förändringar från adk12

- Nytt upplägg 2012 av momentet dynamisk programmering finslipas och utvärderas
- Nytt forskningsprojekt om pseudokod och korrekthetsmotiveringar
- Läsläxor inför föreläsningarna ges så att föreläsningarna kan ägnas åt det som känns oklart och svårt

Relevanta förkunskaper i datalogi

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

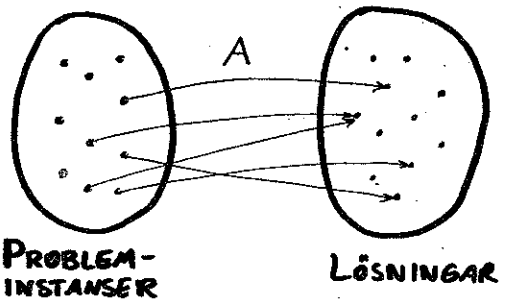
DEFINITION:

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 PROBLEMINSTANSENS 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?

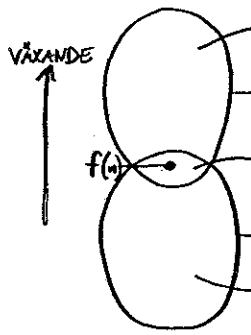
TÄNK PÅ ATT FUNKTIONS- OCH PROCEDURANROP OCKSÅ TAR MINNE.

Hur komplexitet kan anges

Hur ändras komplexiteten för växande storlek 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 $f(n)$

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

Lösning till vanliga rekursionsekvationer

Sats: ("Master theorem")

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

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

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

Exempel: Binärsökning

```
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'
```

Analys:

Låt $T(n)=$ (i värsta fall) tiden att söka bland n tal med BINSEARCH.

$$T(n)=\begin{cases} \Theta(1) & \text{om } n=1 \\ T\left(\lceil\frac{n}{2}\rceil\right)+\Theta(1) & \text{om } n>1 \end{cases}$$

Om $n=2^m$ får vi $T(n)=\begin{cases} \Theta(1) & \text{om } n=1 \\ T\left(\frac{n}{2}\right)+\Theta(1) & \text{om } n>1 \end{cases}$

Mästarsatsen (Master theorem) säger då

$n^{\log_2 1}=n^0\in\Theta(1)$

Då är $T(n)=\underline{\Theta(\log n)}$

Analys av problem

Ringa in ett problems komplexitet!

Övre gräns:

Ge en algoritm som löser problemet.

Algoritmens komplexitet är en <u>övre gräns</u> för problemets komplexitet.

Undre gräns:

Ofta svårt att ange.

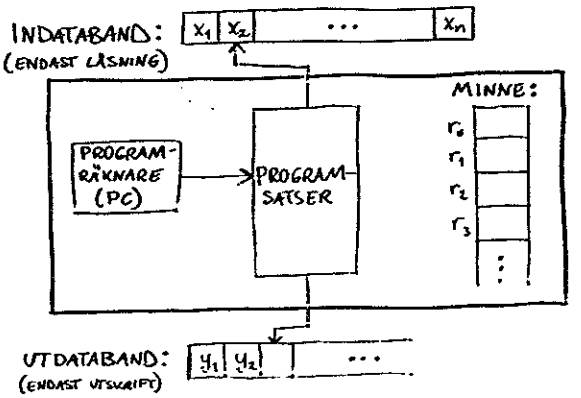
Egenskaper hos problemet måste användas.

Exempel:

- Måste titta på alla indata $\Rightarrow \Omega(n)$
- Måste producera hela utdata
- Beslutsträd – ett visst antal olika fall måste särskiljas

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 $O(\,)$-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 $O(1)$ med enhetskostnad

tid $O(n)$ med bitkostnad