Föreläsning 8: Intro till Komplexitetsteori

Formalisering av "rimlig tid"

- En algoritm som har körtid $O(n^k)$ för någon konstant k är rimligt snabb.
- En algoritm som har körtid $\Omega(c^n)$ för någon konstant $c > 1$ är för långsam.

Begreppet "effektiv algoritm" är alltså synonymt med "går i polynomisk tid" i den här kursen.
Är detta en rimlig uppdelning?

Formell definition av polynomiella problem

Ett *formellt språk* L är en mängd av strängar.
Exempel:
{"abc", "qwerty", "xyzzy"}
{binära strängar av udda längd}
{binära strängar som beskriver ett primtal}
{syntaktiskt korrekta C-program}
Ett språk kan beskrivas på olika sätt:

- Uppräkning av strängarna i språket.
- Uppsättning regler som definierar strängarna i språket (d.v.s. en *grammatik*).
- Algoritm som känner igen strängarna i språket.

Varje beslutsproblem motsvarar ett språk:
Språket som består av alla ja-instanser.
Omvänt säger vi att algoritmen A *känner igen* språket L om

$$A(x) = \text{Ja om } x \in L,$$
$$A(x) = \text{Nej om } x \notin L.$$

A tar *polynomisk tid* om det finns en konstant k så att $A(x)$ tar tid $O(|x|^k)$.
$\mathbf{P} = \{L : \exists A \text{ som löser } L \text{ i polynomisk tid}\}$
Dessa problem (språk) är de vi i kursen
betraktar som lätta.
Finns det svåra problem?: Två svar kan ges.
1. Det finns *artificiella* problem som man kan bevisa att de inte ligger i $\mathbf{P}$.
2. Det finns *naturliga* problem som man antar inte ligger i $\mathbf{P}$.

Exempel på (troligen) svåra problem

Schemaläggning:
KTH har, som vanligt, fått fler elever utan att ha tillgång till mer salar. Schemaläggarna måste ta hänsyn till ett stort antal villkor när de lägger schemat:

- Ingen sal får dubbelbokas
- Ingen lärare får dubbelbokas
- Ingen årskurs får dubbelbokas
- Övningar måste ligga efter föreläsningar etc
- 8-10 i Kista och 10-12 på campus går inte
- Jämn fördelning mellan olika läsveckor
- Lärare X arbetar bara onsdagar–fredagar

Går det att konstruera ett schema som uppfyller alla villkoren?

Satisfierbarhet:
I t.ex. AI och programverifikation kan man ofta beskriva ett problem som en logisk formel i n variabler där en variabeltilldelning som gör formeln sann t.ex. kan svara mot att ett program är korrekt.
Ex. Betrakta formeln

$$(x \vee y \vee \neg w) \wedge (\neg x \vee z) \wedge$$
$$(\neg y \vee w) \wedge (x \vee \neg w \vee \neg z)$$

Finns det en satisfierande variabeltilldelning?
Formeln ovan satisfieras om x och z är sanna medan y och w är falska.

Handelsresandeproblemet:
En handelsresande vill besöka alla större städer i Sverige för att sedan återkomma till sin hemstad. För att spara tid och pengar vill han inte åka längre än han måste.
Travelling Salesman Problem (TSP):
Givet en graf $G = (V, E)$ med vikter på kanterna, finns det någon tur av längd högst L som passerar alla hörnen och återkommer till starthörnet?
Detta visar sig vara relaterat till problemet
Givet en graf, innehåller den någon hamiltoncykel?
(En hamiltoncykel är en cyklisk delgraf som passerar alla hörn exakt en gång.)

Kappsäcksproblemet:
En fjällvandrare ska packa en ryggsäck och orkar inte bära mer än W kg. Det finns många saker han vill ha med, och varje sak har känd vikt och användbarhet:

Sak	Vikt	Nytta
Tält	10	100
Sovsäck	7	80
Kudde	0.5	10
Extra tröja	1	25
Tandborste	0.01	5
Rakhyvel	0.1	2
etc		

(Det går inte att dela på någon sak.)
Går det att välja ut saker av total vikt högst W kg så att nyttan blir minst U?

Graffärgning:
I vissa sammanhang är det naturligt att betrakta komponenterna i ett system som hörnen i en graf där varje komponent är i ett visst tillstånd. En konfliktsituation kan då uppstå om angränsande komponenter är i samma tillstånd.
Som grafproblem:
Går det att färga hörnen i grafen G med högst K färger så att ingen kant förbinder två hörn med samma färg?
Också om G är planär förblir problemet svårt.

Alla problemen kan alla lösas med *uttömmande prövning*:
Prova alla möjliga kombinationer av de ingående variablerna och kolla om någon löser problemet.
Nackdel:
För alla dessa problem ger det väldigt
dåliga tidskomplexiteter — typiskt $\Omega(n!)$ eller $\Omega(2^n)$.
Gemensamma egenskaper:
Alla problem delar följande egenskap:
Om svaret är "ja" går det givet en lösning lätt att verifiera att den är korrekt.
D.v.s. någon som har löst problemet kan snabbt övertyga mig om det.
Obs att omvändningen inte behöver gälla: Om det t.ex. inte finns något tillåtet schema behöver det inte finnas något enkelt sätt att bevisa detta.

Klassen NP

Komplexitetsklassen **NP** innehåller de ja/nej-problem där det till varje ja-lösning finns ett "bevis" som kan kontrolleras effektivt (d.v.s. polynomisk tid).
Några problem i **NP**:

- Är vektorn $a[1..N]$ sorterad?
- Är den logiska formeln Φ satisfierbar?
- Är talet N sammansatt?

Annan definition av NP

Man kan se uttömmande prövning som ett stort träd där varje lösning svarar mot ett löv
Höjden är polynomisk men antalet löv exponentiellt.
Om man i varje nod valde rätt stig skulle man kunna lösa problemet på polynomisk tid (förutsatt att det finns någon lösning).
Detta kan användas som definition av **NP**.

Olika formella definitioner av NP

Första definitionen:
A *verifierar* instansen x av problemet L om det finns ett certifikat y så att

$$A(x, y) = \text{Ja} \quad \Leftrightarrow \quad x \in L$$

D.v.s A avgör språket
$L = \{x \in \{0,1\}^* : \exists y \in \{0,1\}^* : A(x, y) = \text{Ja}\}$
NP $= \{L : \exists A$ som verifierar L i polynomisk tid$\}$
P $\subseteq$ **NP** eftersom alla problem som kan lösas snabbt också kan verifieras snabbt.
Andra definitionen:
En ickedeterministisk algoritm är en algoritm som gör slumpmässiga val i olika steg. Utfallet av en körning är slumpmässigt. A accepterar ett språk L om:

$$x \in L \Rightarrow A(x) = \text{Ja} \quad \text{med sannolikhet} > 0$$
$$x \notin L \Rightarrow A(x) = \text{Nej} \quad \text{med sannolikhet } 1$$

NP $= \{L : \exists$polynomisk icke-deterministisk alg. som avgör $L\}$

P kontra NP

Om något **NP**-fullständigt problem kan lösas på polynomisk tid så kan alla lösas på polynomisk tid.
P $\subseteq$ **NP** eftersom alla problem som kan lösas på polynomisk tid också kan verifieras snabbt.
Är P $\neq$ NP?
Nästan alla forskare tror det, men sedan frågan formulerades 1971 har ytterst små framsteg mot ett bevis gjorts.
Den bästa undre gränsen för **NP**-fullständiga problem är $\Omega(n)$ d.v.s. den triviala gränsen.

Polynomiska reduktioner (Karp-reduktioner)

Problemet Q kan *reduceras* till Q' om varje instans x av Q kan omvandlas till en instans x' av Q' så att $x \in Q$ om och endast om $x' \in Q'$.
Den här typen av reduktion kallas för
karpreduktion.

Om reduktionen tar polynomisk tid skriver vi

$$Q \leq_P Q'$$

eftersom Q inte kan vara svårare att lösa än Q'.
Viktiga konsekvenser av detta:

- Om $Q \leq_P Q'$ och $Q' \in \mathbf{P}$ så $Q \in \mathbf{P}$.
- Om $Q \leq_P Q'$ och $Q \notin \mathbf{P}$ så $Q' \notin \mathbf{P}$.

NP-fullständiga problem

Problemet Q är **NP**-fullständigt om

1. $Q \in \mathbf{NP}$
2. Q är **NP**-svårt: $Q' \leq_P Q$ för alla $Q' \in \mathbf{NP}$

Att ett problem är **NP**-fullständigt betyder i praktiken att det inte finns någon effektiv algoritm för det; nästan alla tror att $\mathbf{P} \neq \mathbf{NP}$.