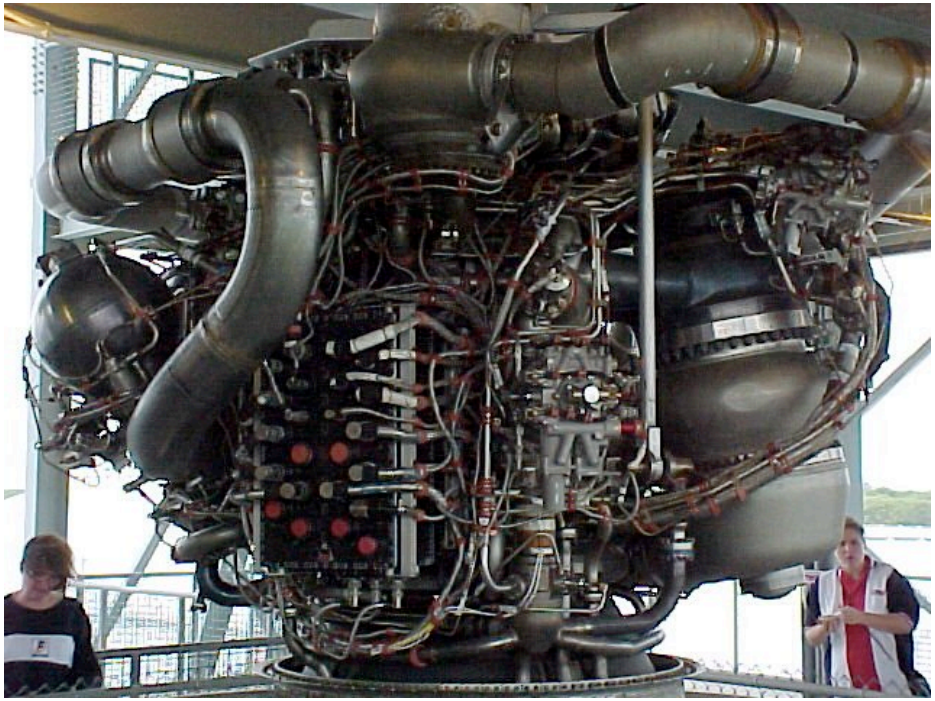
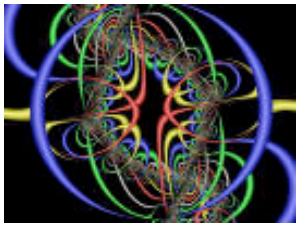
Problemkompleksitet



Problemers kompleksitet



For en stor klasse af vigtige problemer kendes ingen løsningsalgoritmer, der kan garanteres at være hurtige

En **effektiv** algoritme: køretiden er opad begrænset af et polynomium

Et problem, der kan løses med en effektiv algoritme, siges at være **medgørligt** (eller **let)**

En **ineffektiv** algoritme: køretiden vokser eksponentielt med input-størrelsen

Et problem, der ikke kan løses med en effektiv algoritme, siges at være **umedgørligt** (eller **svært)**

Eksempler på svære problemer

- **Den rejsende sælgers problem**
  En sælger skal besøge *N* byer. Find en rejserute, der minimerer rejseomkostningerne

- **Planlægning**
  Et antal opgaver af varierende varighed skal udføres på to identiske maskiner inden en vis tidsfrist. Kan tidsfristen overholdes?

- **Tilfredsstillelighed (satisfiability)**
  Er det muligt at tildele sandhedsværdier til variablerne i et logisk udtryk, som gør det sandt?

  $$(a \vee b) \wedge (\neg a \vee \neg b)$$

Flere eksempler på svære problemer

- **Opdeling**
  Givet en mængde af heltal. Findes der en opdeling i to delmængder, således at deres sum er ens?

- **Knudedækning**
  Givet en graf og et heltal *N*. Findes der en mængde bestående af færre end *N* kanter, der berører alle grafens knuder?

- **3-farvning**
  Kan knuderne i en graf farves med tre farver, således at hvert par af forbundne knuder har forskellig farve?

Mængden $\mathcal{P}$

Mængden af problemer, der kan løses med en **deterministisk** algoritme i **polynomiel** tid

Omfatter næsten alle programmer, der kører på eksisterende computere

Karakteristikken "polynomiel tid" er uafhængig den valgte deterministiske computer og implementering

Mængden $\mathcal{NP}$

Mængden af problemer, der kan løses ved en **ikke-deterministisk** algoritme i **polynomiel** tid

Hvis en maskine kan gætte (og er heldig), så kan den løse et problem i $\mathcal{NP}$ hurtigt (i polynomiel tid)

Konventionelle computere kan simulere heldige gæt i eksponentiel tid ved at afprøve enhver mulighed

Hovedspørgsmålet

Hvilken relation er der imellem $\mathcal{P}$ og $\mathcal{NP}$?

$\mathcal{P} = \mathcal{NP}$: vi behøver ikke at bruge ikke-determinisme for at løse problemet i polynomiel tid

$\mathcal{P} \neq \mathcal{NP}$: umedgørlige (hårde) problemer kan ikke løses i polynomiel tid, undtagen ved brug af ikke-determinisme

For mange problemer er der ikke andre løsningsmetoder end at prøve enhver mulighed, **men** ingen har endnu været i stand til at bevise, at der findes noget problem i $\mathcal{NP} \setminus \mathcal{P}$

$\mathcal{NP}$-komplette problemer

Et problem i $\mathcal{NP}$ siges at være **$\mathcal{NP}$-komplet**, hvis eksistensen af en polynomiel algoritme for problemet ville medføre, at $\mathcal{P} = \mathcal{NP}$

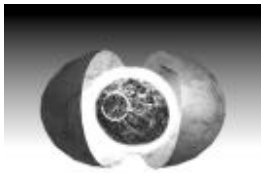
Hvis et $\mathcal{NP}$-komplet problem kan løses i polynomiel tid, så kan **alle** problemer i $\mathcal{NP}$ løses i polynomiel tid

Reduktion af problem *A* til problem *B*:

* konverter en instans af *A* til en instans af *B*
* løs *B*-instansen
* konverter løsning af *B* til løsning af *A*

Sætning: Hvis *B* er et $\mathcal{NP}$-komplet problem, og *A* kan reduceres til *B* i polynomiel tid, så er *A* også $\mathcal{NP}$-komplet.

Betydningen af $\mathcal{NP}$-komplethed



Tusindvis af problemer er blevet bevist at være $\mathcal{NP}$-komplette
Hvis blot ét af disse kan løses effektivt, så kan de alle

Et bevis for, at der i det mindste findes et $\mathcal{NP}$-komplet problem blev givet af Cook (1971). Han beviste, at logisk tilfredsstillelighed (satisfiability) er $\mathcal{NP}$-komplet. Beviset herfor er kompliceret

$\mathcal{P} = \mathcal{NP}$?

Enten ($\mathcal{P} = \mathcal{NP}$)

kan virkelige (deterministiske) computere være lige så effektive som computere, der er udstyret med ikke-determinisme. Men vi ved endnu ikke hvordan

Eller ($\mathcal{P} \neq \mathcal{NP}$)

ikke-determinisme hjælper. Men det er en fiktion, og intet $\mathcal{NP}$-komplet problem kan løses i polynomiel tid

Praktisk talt ingen tror, at $\mathcal{P} = \mathcal{NP}$ ... men det er muligt

At et problem er $\mathcal{NP}$-komplet tages sædvanligvis som udtryk for, at der ikke findes nogen effektiv løsning

Besvar spørgsmålet og tjen 1.000.000$ (http://www.claymath.org/millennium/)

Afgørlighed

Uafgørlige problemer er problemer, som ikke kan løses algoritmisk

Eksempler:

- Afgør, om en sætning i prædikatlogikken er gyldig eller ej
- Afgør, om to syntaksbeskrivelser definerer det samme sprog
- Bevis, at en algoritme altid terminerer (Stopproblemet)

Terminering?

(a)

```
while (x != 1)
    x = x - 2;
```

(b)

```
while (x > 1)
    if (x % 2 == 0)
        x = x / 2;
    else
        x = 3 * x + 1;
```

Collatz sekvenser: (12, 9)

12, 6, 3, 10, 5, 16, 8, 4, 2, 1

9, 28, 14, 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1

Terminering?

(fortsat)

(c)

```
for (int x = 3; ; x++)
for (int a = 1; a <= x; a++)
for (int b = 1; b <= x; b++)
for (int c = 1; c <= x; c++)
for (int n = 3; n <= x; n++)
    if (Math.pow(a,n) + Math.pow(b,n) == Math.pow(c,n))
        System.exit(0);
```

Programmets stopper, hvis og kun hvis Fermats sidste teorem er falsk

For $n \geq 3$ findes ikke tre heltal a, b, og c, så $a^n + b^n = c^n$ P. de Fermat (1601-65)

Et bevis for teoremet blev fundet i 1994 (beviset er meget komplekst og fylder 150 sider)

Stopproblemet

Det er umuligt at konstruere en algoritme, der for *enhver* algoritme *A* kan afgøre, om denne terminerer

Bevis (indirekte):

Antag eksistensen af **`terminates(A)`**, som for enhver metode **`A`** returnerer **`true`**, hvis **`A`** terminerer; ellers **`false`**

Definer:

```
void A() {
    while (terminates(A)) /* do nothing */;
}
```

Stopper et kald af **`A`**?