Varför är beräkningsproblem Svåra?

Johan Håstad

Nada
Kungl Tekniska Högskolan

Pessimistiskt svar

VET INTE

Mer optimistikt svar

Ibland vet vi.

Oftast har vi en grov gissning.

Vetenskapligt område

Komplexitetsteori.

Studie av effektiva algoritmer med matematiska metoder.

Grundläggande definitioner

Dator	En "vanlig" dator som opererar på tal av begränsad storlek.
Tid	Antal elementära operationer.
Problem komplexitet	Hur tiden växer som funktion av indatas storlek.

Exempel:Multipliktion

Multiplicera två tal med vardera n siffror, tex

$$a = 3136674400728904326556039381962950876341797834016454346243440772665024014059218945 6$$

och

$$b = 6103447631703121904666471413035369758980595076314114905306172676730771567868069446 1$$

men tänk på riktigt stora tal med $n \approx 1000000$.

Standardalgoritm: Multiplicera varje siffra i a med varje siffra i b, $\approx n^2$ operationer.

Bäst möjligt?

Optimalitetsbevis?!

Varje siffra i a måste "kombineras" med varje siffra i b, alltså krävs n^2 operationer.

Fullständigt struntsnack som inte
borde övertyga någon.

Effektiv multiplikation.

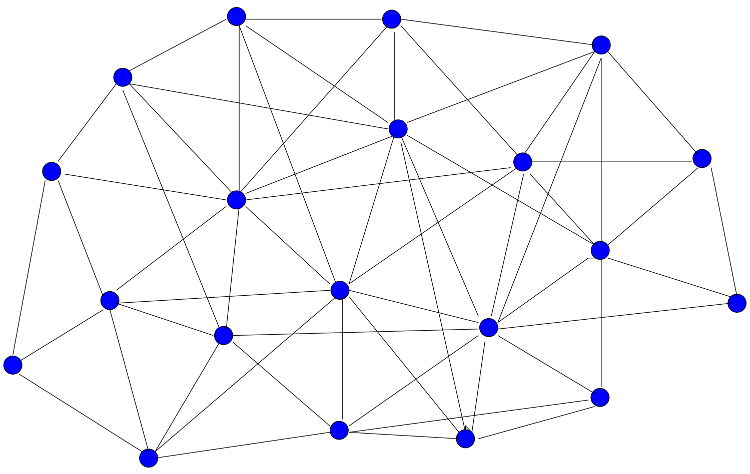
Den mest effektiva algoritmen för att multiplicera två heltal har följande delsteg.

- Betrakta talen a och b som vektorer med n element.
- Beräkna diskreta fouriertransformen av dessa vektorer.
- Multiplicera fouriertransformerna punktvis.
- Gör en invers fouriertransform.
- Konvertera svaret till ett heltal.

Räcker med $\approx n \log n \log \log n$ operationer.

Klick problemet

Givet en graf med n hörn finns det k hörn som alla är parvis förbundna?



Uppenbara sätt att leta går igenom väldigt många delmändger av hörn. Kom ihåg att det finns 2^n delmängder av n hörn.

Kan detta göras i tid n^3?

Allmänt mål

Givet ett beräkningsproblem. Visa

- Att det finns en algoritm som löser instanser av storlek n inom tid $T_1(n)$.

- Att varje algoritm som löser problem kräver tid $T_2(n)$ för någon instans av storlek n.

Hur uppför sig $T_1(n)$ och $T_2(n)$ för stora n?

Växer de som n^c (polynomiellt) eller som 2^n?

Hur inses att ett problem är effektivt lösbart?

Man utvecklar och analyserar en effektiv algoritm. Antingen genom

- Kända algoritmiska metoder. Dynamisk programmering, dela och erövra, reduktion till linjär programmering.

- Specialdesignade algoritmer.

Fortlöpande förbättringar.

Bevismetoder olösbarhet

Två olika bevisprinciper.

Inre Titta i detalj på inre mekanismen av beräkningarna. Funktionen är alltför komplicerad att beräknas så snabbt.

Yttre Fanns det en algoritm som kan beräkna X i tid n^3 så kan vi beräkna problem Y i tid n^4 vilket vi vet inte går.

Inre resultat

Framgångar mest i begränsade beräkningsmodeller.

- Jämförelsebaserade algoritmer. Vi vet (nästan) precis hur många operationer som krävs för att sortera n poster om vi bara får använda indata för jämförelser.

- Monotona algoritmer. Inget negativt tänkade.

Nedslående faktum

För de flesta, tämligen enkla beräkningsproblem, vet vi inte att det krävs mer tid än att läsa indata och skriva utdata.

Kan multiplikation vara lika lätt som addition dvs bara kräva tid $\approx n$ på n siffrors tal?

Illustration av bevisproblem

Vi vill lösa klickproblemet med $k = n/200$.
Betraka följande algoritmer.

- Beräkna näst största egenvärdet för grann-matrisen. Är det större än $n/100$ svara ”ja” annars ”nej”.

- Beräkna antalet trianglar i grafen, om antalet är udda svara ”ja” annars ”nej”.

- Betrakta bitrepresentation av grafen som ett heltal svara ”ja” om det är primtal.

Ta majoritetsbeslut av dessa. Korrekt?

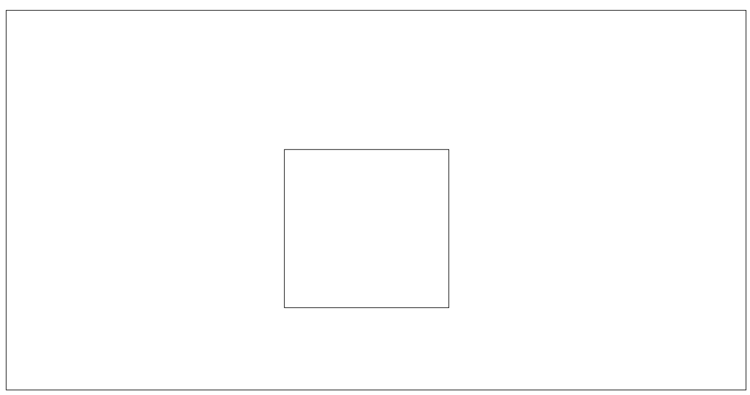
Önskedröm

Kan det finnas ett rimligt mått på komplexitet hos en funktion?

För väldigt enkla mått av komplexitet så är detta möjligt, men inget känt om komplexitet är "Tid på riktig dator".

Yttre argument; principer

Svart låda. Givet en algoritm för problem X så kan vi lösa ett annat problem Y genom att använda given algoritm som subrutin.



Ger inga absoluta resultat. Kan vi inte lösa Y så kan vi inte lösa X, men hur kan vi möjligtvis bevisa att vi inte kan lösa Y? Vi behöver en grundbult.

Riktiga Grundbultar

Olösliga problem	Givet ett datorprogram och ett indata, kommer programmet någonsin att stanna?
Problem som kräver lång tid	Givet ett datorprogram och ett indata, kommer programmet stanna inom tid T?

Det senare kräver nästan tid T.

Moral: Det finns ingen generellt sätt att förutsäga utfallet av beräkningar som är effektivare än att utföra beräkningen.

Troliga Grunbultar

Det finns skäl att tro att vissa beräkningsproblem är svåra. Vi skiljer på

Mänsklig Självsäkerhet	Många smarta människor har tänkt på detta problem och ingen har lyckats lösa det effektivt. Exempel: Faktorisering av stora heltal.
Strukturella Skäl	Om detta problem är enkelt så hjälper aldrig gudabenådade gissningar beräkningar. Exempel: Klick.

Gudabenådade gissningar!?

Ta klickproblemet. Vi vill hitta ett antal parvis förbunda hörn i en graf. Jämför

- Algoritmen måste lösa det själv.
- Genom en bra gissning så vet vi var klicken är och det är bara att verifiera att den är OK.

Om klick är lätt att beräkna så är denna typ av gissningar aldrig av avgörande betydelse.

NP-fullständiga problem.

En samling med tusentals problem som alla
är grovt sett ekvivalenta med klick.
Troligen svåra men okänt.

- Satisfierbarhet av logiska formler.
- Handelsresandeproblemet.
- Färgning av grafer.
- Heltalsprogrammering.
- Felrättning av allmänna koder.

Slutsatser

Det finns åtminstone två skäl att beräknings problem är svåra. De kan vara

Komplicerade	Problemet innehåller så mycket komplikationer som måste redas ut.

När detta är fallet lyckas vi i dagsläget ofta inte visa att de är svåra.

Användbara	Om vi kunde lösa detta problem så kan vi lösa en mängd andra problem.

Vi lyckas i detta fall bevisa eller göra mycket troligt att problemet är svårt.

En egendomlig värld

Nästan alla naturliga beräkningsproblem är antingen lätta eller användbara. De flesta problem är därmed, om än i en väldigt grov skala, klassificerade.

Det finns bara en handfull problem som inte är åtminstone lika svåra som de NP-fullständiga eller känt lätta.

Knivig fråga

RSA-systemet för kryptering och digitala signaturer är säkert bara om det är svårt att faktorisera stora heltal. Ska samhället investera ett antal miljarder för att bygga upp en infrastruktur baserat på digitala signaturer?

Ja	Redan Gauss tänkte mycket på faktoriseringsproblemet och ingen har kommit på något riktigt bra.
Nej	Väsentliga algoritmframsteg kommer med jämna mellanrum och ett stort steg kan komma när som helst.