MAT1030 – Diskret Matematikk

Forelesning 30: Kompleksitetsteori

Dag Normann

Matematisk Institutt, Universitetet i Oslo

12. mai 2010

(Sist oppdatert: 2010-05-12 12:45)

Forelesning 30: Kompleksitetsteori

Oppsummering

- I dag er siste forelesning med nytt stoff!
- Etter det vil vi begynne på en full repetisjon av pensum.
- Nå, først litt repetisjon av kompleksitetsteorien.

Oppsummering

- Kompleksitetsteori: Om algoritmers *tidsforbruk*.
- Fire tilnærminger:
  1. Tell bare de mest tidkrevende operasjonene.
  2. Ta utgangspunkt i de verste tilfellene.
  3. Anta at input er stort.
  4. Ikke skill mellom to tidskompleksiteter hvis vekstraten til den ene er et konstant multiplum av vekstraten til den andre.
- Tidskompleksiteten til en algoritme: En funksjon fra $\mathbb{N}$ til $\mathbb{N}$.
- O-notasjon:
  - f er $O(g)$ hvis det fins en positiv konstant c slik at $f(n) \leqslant c \cdot g(n)$ for alle tilstrekkelig store n.
  - Viktig: Forstå hva det vil si at en funksjon f *ikke* er $O(g)$.
  - Hvis f er en polynomfunksjon med grad $\leqslant k$, så vil f være $O(n^k)$.

Sorteringsalgoritmer

- V i så på to sorteringsalgoritmer.
- I begge tilfellene sorterer vi en liste på n tall ved å plassere ett og ett riktig i forhold til de som allerede er sortert.
- Vi så på to måter å plassere *det neste tallet* på rett plass på:
  1. Sammenlikn med ett og ett tall i den sorterte delen for å finne rett plassering.
  2. Sammenlikn med midten, midten av resten, osv. til vi finner rett plass.
- Metode 1. gir større kompleksitet enn metode 2.

Sorteringsalgoritmer

Definisjon

Hvis n er et tall, lar vi

$$\lg n$$

være tallet m slik at $2^m = n$.

Vi kan kalle dette for binærlogaritmen til n.

- For alle praktiske formål i kompleksitetsteori, kunne vi brukt funksjonen som gir antall sifre i binærrepresentasjonen av n i stedenfor.
- Den mest effektive sorteringsalgoritmen har en tidskompleksitet som er $O(n \cdot \lg n)$. (Se oppgavene i boka.)
- Man bør lese boka og forstå hvorfor følgende er tilfelle.
  - $\lg n$ er $O(n)$
  - n er ikke $O(\lg n)$

Gjennomførbare algoritmer

- Vi har snakket om at vi skal lære å vurdere om en algoritme kan gjennomføres i løpet av realistisk tid.
- Som de gode matematikere vi har blitt skal vi selvfølgelig gi en presis definisjon av hva som menes med en gjennomførbar eller overkommelig algoritme.
- Vi har snakket om algoritmer hvor kompleksiteten er $O(n \cdot \lg(n))$, $O(n^{\frac{3}{2}})$ og $O(n^2)$.
- Alle disse er gjennomførbare.
- Vi skal se på noen algoritmer som ikke er gjennomførbare for store input.

Gjennomførbare algoritmer

- Vi avsluttet gårsdagens forelesning med å se på to eksempler.
- Det å avgjøre om et utsagn er en tautologi eller ikke krever $O(n \cdot 2^n)$ regneskritt om vi bruker sannhetsverdimetoden.
- For å faktorisere et primtall med n siffre, må vi i prinsippet lete gjennom $2^{\sqrt{n}}$ mulige faktorer. Det tar lang tid.
- Vi skal se på et eksempel til.

Gjennomførbare algoritmer

Eksempel

- La G være en sammenhengende graf.
- Hvordan skal vi gå frem for å bestemme om grafen har en Hamiltonsti, det vil si en sti som er innom hver node nøyaktig en gang?
- Hvis n er antall noder i grafen, vil en Hamiltonsti ha $n-1$ kanter
- Det finnes ingen kjent måte å undersøke om G har en Hamiltonsti på som er vesentlig mer effektiv enn den naive; prøv alle stier med $n-1$ kanter og se om en av dem tilfeldigvis skulle være en Hamiltonsti.

Gjennomførbare algoritmer

Eksempel (Fortsatt)

- I det verste tilfellet er antall stier i G med $n-1$ kanter $O(\binom{n^2}{n-1})$, det vil si

$$\frac{(n^2)!}{(n^2-n+1)!(n-1)!}$$

- Dette er et tall som faktisk er større enn 2^{n-1}, så algoritmen er ikke imponerende effektiv.

Gjennomførbare algoritmer

Definisjon

Vi sier at en algoritme er gjennomførbar (tractable på engelsk) hvis tidskompleksiteten er $O(n^k)$ for en k.

Vi merker oss følgende sammenhenger.

- 2^n er ikke $O(n^k)$ for noen verdi av k
- 2^n er $O(n!)$

Gjennomførbare algoritmer

- Det er flere grunner til at man har falt ned på dette som en fornuftig definisjon.
- Tidligere erfaringer tilsa at hvis en algoritme er gjennomførbar i henhold til denne definisjonen, kan den brukes i praksis.
- Det er ofte slik at k ligger rundt tre eller lavere.
- Ganske overraskende viste en gruppe indere for noen år siden at det finnes en algoritme som avgjør om et tall er et primtall eller ikke som faller inn under denne definisjonen, men der var k (og konstanten c) så stor at algoritmen hadde mer teoretisk enn praktisk verdi.
- Definisjonen er også ganske robust, selv om forskjellige matematiske modeller for hva en beregning består i kan gi forskjellige verdier på graden.

Gjennomførbare algoritmer

- Vi skal avslutte disse forelesningene med å snakke bittelitegrann om **P** og **NP**.
- **P** er klassen av problemer som kan løses i polynomisk tid, det vil si de som kan løses av en gjennomførbar algoritme slik vi har definert det.
- Eksempler på problemer som ligger i **P** er om en graf er sammenhengende og om den har en Eulerkrets, om to termer lar seg unifisere, om et uttrykk svarer til en term på polsk form og etterhvert om et tall er et primtall eller ikke (det kom som en overraskelse).
- **NP** er grovt sagt klassen av problemer hvor vi med flaks bare trenger å bruke polynomisk tid for å løse det den ene veien, mens vi tilsynelatende bruker eksponensiell tid om løsningen går den andre veien.

Gjennomførbare algoritmer

- Hvis G er en graf, og noen streker opp en Hamiltonsti, er det raskt å få bekreftet at det er en Hamiltonsti det er, mens hvis det ikke finnes noen Hamiltonsti trenger vi lang tid.
- Hvis A er et uttrykk som ikke er en tautologi, kan vi få vite det veldig fort hvis vi tilfeldigvis prøver den fordelingen av sannhetsverdier som gjør utsagnet usant, mens vi fortsatt må skrive ut hele sannhetsverditabellen hvis utsagnet er en tautologi.
- Gitt en eske med mange puslespill-brikker, er det lett å vise at brikkene kan settes sammen til et bilde hvis vi greier å legge brikkene riktig én for én, mens det tar lang tid å bli sikker på at det er feil ved noen av brikkene, om det er tilfelle.

Gjennomførbare algoritmer

- Det store åpne problemet er om disse mengdene av problemer er de samme, eller om det finnes problemer som er i **NP** men ikke i **P**.
- Litteratur rundt $\mathbf{P} = \mathbf{NP}$-problemet finner dere på nettet og i en rekke lærebøker om automatateori eller kompleksitetsteori, eller i en NORMAT-artikkel fra 2005 (Årgang 53, bind 1).
- Dette er et av de syv milleniumsproblemene i matematikk, og det er en dusør på $\$10^6$ for hvert av de seks som står fortsatt uløst.
- Den som løste det eneste milleniumsproblemet som er løst, sa nei takk til pengene.

Slutt

Repetisjon

Repetisjon av hele pensum

- Vi går systematisk gjennom alle delene av pensum og trekker frem de ferdighetene man må beherske for å være sikker på å greie 90% av eksamensettet.
- For å være sikker på greie 100% må man kunne alt som er oppgitt som pensum.
- Oppgaver kan hentes fra hele pensum, men hovedvekten blir på stoff som ikke er testet gjennom de to obligatoriske oppgavene.
- Hvis man kan løse alle oppgaver av den typen som er gitt som treningsoppgaver gjennom semesteret, ligger man meget godt an.
- Vi skal se på noen eksempler, men det er ikke slik at alle eksemplene dekker oppgaver som blir gitt til eksamen, og heller ikke slik at vi illustrerer alle eksamensoppgavene med eksempler her.

Kapittel 1

- I Kapittel 1 så vi på algoritmebegrepet generelt og kontrollstrukturer spesielt.
- Vi har brukt kontrollstrukturer når det har passet seg slik gjennom hele semesteret.
- Viktige deler av læringsmålet i dette emnet er at studentene skal
  - Kunne utarbeide en kontrollstruktur som en nærmere presisering av en algoritme.
  - Kunne lese en kontrollstruktur og forstå sammenhengen mellom inputverdiene og outputverdiene.
  - I tilknytning til Kapittel 13, kunne vurdere tidskompleksiteten.

Kapittel 2 og 3

- Kapittel 2 og 3 omhandler tallsystemer og prinsipper for digital representasjon av hele tall og reelle tall.
- Mye av dette stoffet er behandlet mer grundig i andre emner.
- Forståelsen av digital representasjon ble forsiktigvis testet i det første obligatoriske oppgavesettet.
- Vi har etterhvert hatt bruk for å vise hvordan relasjoner, grafer, trær og andre dataobjekter kan representeres i en datamaskin.
- Det er lite aktuelt med en oppgave som behandler stoff fra disse kapitlene direkte, men man kan ha fordel av å forstå hvordan grafer, trær og andre objekter representeres, især i forbindelse med en eventuell oppgave i kompleksitetsteori.

Kapittel 4

- I dette kapitlet om logikk har vi konsentrert oss om
  - utsagnslogikk
  - kvantorer
  - føring av argumenter
- I tillegg er det en del snakk om betydningen av logikk i programmering.
- Dette siste er mest motiverende, men en motivasjon man bør ta med seg til senere emner.

Kapittel 4

- I utsagnslogikk har vi lagt mest vekt på det formelle språket, hvor vi tar utgangspunkt i noen utsagnsvariable, og bygger opp utsagn fra utsagnsvariablene ved å bruke bindeordene (eller konnektivene) $\neg$, $\vee$, $\wedge$, $\rightarrow$ og $\leftrightarrow$.
- Et typisk eksempel kan være

$$A = \neg(p \rightarrow q) \rightarrow \neg q \vee p.$$

- Vi lærte hvordan vi kunne spare på parenteser, slik at vi vet hvordan parentesene egentlig skal settes i eksemplet over.
- Vi skal kunne avgjøre om et utsagn som det over er en tautologi, en kontradiksjon eller ingen av delene.
- Den sikreste måten å gjøre det på er å sette opp en sannhetsverditabell.
- Tabellen til utsagnet over finner vi på neste side.

Kapittel 4

p	q	$p \rightarrow q$	$\neg(p \rightarrow q)$	$\neg q$	$\neg q \vee p$	A
T	**T**	**T**	**F**	**F**	**T**	**T**
T	**F**	**F**	**T**	**T**	**T**	**T**
F	**T**	**T**	**F**	**F**	**F**	**T**
F	**F**	**T**	**F**	**T**	**T**	**T**

Vi ser at utsagnet er en tautologi.

Kapittel 4

- Vi lærte også om bruk av kvantorer.
- Kvantorene $\exists$ - "det eksisterer", og $\forall$ - "for alle", brukes for å gi matematiske definisjoner og setninger en presis formulering.
- Kvantorer ble ikke noe sentralt tema dette året, men vi fikk bruk for dem da vi skulle undersøke om f er $O(g)$ for bestemte funksjoner f og g.
- Det er ofte viktig å ha preise definisjoner når man skal vise at en egenskap ikke holder.

Kapittel 4

- Det siste temaet vi tok opp under dette kapitlet var bevisformer hvor vi skiller mellom direkte bevis og indirekte eller kontrapositive bevis.
- I et kontrapositivt bevis antar vi at den påstanden vi vil vise er usann, og så beviser vi at noen av forutsetningene heller ikke kan være sanne.
- Vi har sett på noen eksempler på bevis, men har ikke laget noe stort nummer av det.
- Det er meningen at dere skal kunne gjennomføre et enkelt resonement om dere blir bedt om det.

Kapittel 5

- Kapittel 5 tar opp et bredt tema over få sider.
- Først har vi en innføring i Boolsk mengdelære med snitt, union, komplement og mengdedifferens.
- Det er viktig at dere kjenner definisjonene av $A \cap B$, $A \cup B$, $\overline{A}$ og $A - B$.
- To mengder er like hvis de har de samme elementene.
- $A \subseteq B$, A er inneholdt i B, hvis alle elementene i A også er elementer i B.
- Når vi snakker om komplementet $\overline{A}$, har vi alltid antatt at vi har en universell mengde $\mathcal{E}$.

Kapittel 5

- Hvis vi har et Boolsk uttrykk hvor det inngår to eller tre vilkårlige mengder, kan vi undersøke egenskapene ved dette uttrykket ved å bruke et Venndiagram
- Et eksempel på en mulig oppgave kan være å bruke et Venndiagram til å avgjøre om følgende inklusjon alltid vil holde:

$$(A \cap \overline{B}) - C \subseteq \overline{C - (B \cup A)}.$$

- Vi tar den på tavla.

Kapittel 5

- Kardinaliteten til en mengde er et finere ord for hvor mange elementer mengden har.
- Vi har lært om ordnede par, kartesisk produkt og om potensmengder.
- Potensmengden til A er mengden av alle delmengder av A.
- Dere bør vite at hvis A har n elementer, så har A 2^n delmengder, og A^2 har n^2 elementer.