

Om mulige og tilsynelatende umulige programmeringsoppgaver

Dag Normann*

La oss starte med et eksempel fra filmen *Star Wars, Episode IV*, et eksempel som er relevant for det problemkomplekset vi skal diskutere i denne artikkelen.

I en fjern galakse for lenge, lenge siden hadde noen opprørere fått tak i plan-tegningen til en dødsstjerne, en kampstasjon under det onde *Imperiet*, en kunstig planet med umåtelig slagkraft.

Problemet for opprørerne var å avgjøre om dødsstjernen hadde et svakt punkt, om det var mulig å finne en angrepskjede som ville resultere i stjernens undergang.

Uten at vi skal ta en filmklassiker for alvorlig, kan vi påpeke et problem med realismen her. Antall angrepstrategier mot en så stor dødsstjerne er så overmåte stort at man umulig kunne ha testet dem alle ved hjelp av datamaskiner. Det synes som at problemet om en dødsstjerne er sårbar eller ikke tilhører en av vår tids ivrig studerte kompleksitetsklasser, klassen **NP**. Vi vet ikke hvordan vi kan løse **NP**-problemer i løpet av kort tid, selv ved hjelp av kraftige maskiner basert på digital teknologi.

I denne artikkelen skal vi snakke litt om hva kompleksitetsklasser er for noe, hva som kjennetegner **NP** og litt om konsekvensene av at **NP**-problemer i praksis er umulig å løse fullstendig og i stor skala.

La oss starte med et noe mer nærliggende eksempel, både i sted og tid: En familiefar vil ta med familien på biltur i Skandinavia, og lurer på hvor lang tid det vil ta å besøke alle tettsteder med minst 100 innbyggere. Familiefarens ferieproblem kan sees som et problem i en kjent problemsamling, *den handelsreisendes problem*:

La B være en samling byer hvor avstandene mellom byene er gitt ved en avstandstabell, for eksempel er avstanden angitt i antall kilometer. La n være et tall. Den generelle formen for den handelsreisendes problem som vi vil se på er da:

Er det mulig å besøke alle byene ved samlet å kjøre $\leq n$ antall kilometer?

Hvis vi gir oss selv ubegrenset med tid er det lett å finne en metode for å løse dette problemet, vi lister opp alle reiserutene, måler lengden på hver av dem og ser om en av dem har lengde $\leq n$ km. Problemet med denne metoden er at antall reiseruter er veldig stort, så stort at selv ikke moderne datamaskiner

*Universitetet i Oslo, Matematisk Institutt, Boks 1053 - Blindern, 0316 Oslo

pr. idag vil kunne gjennomføre den i rimelig tid (for eksempel så lang tid det er igjen til ferien eller til neste Big Bang). Hvis vi har k byer, så har vi

$$k! = 1 \cdot 2 \cdot 3 \cdots (k-1) \cdot k$$

antall reiseruter og selv med så lite som 10 byer ender vi opp med 3.628.800 mulige reiseruter. Med 13 byer vokser tallet til over seks milliarder og selv om familiefaren vår begrenser reiselysten til de 25 største byene i Skandinavia, er denne metoden ubrukelig. Selvfølgelig er det mulig å forbedre metoden betraktelig, men hvis man vil være sikker på alltid å få riktig svar er det ingen som har greid å lage en metode for å løse det generelle handelsreisendes problem som gir svar for alle rimelige eksempler fra europeisk eller amerikansk geografi innen overskuelig fremtid. Det er et åpent matematisk problem om det i det hele tatt finnes en praktisk brukbar metode.

En mer vanlig formulering av den handelsreisendes problem er

Hva er den korteste reisestrekningen vi trenger for å besøke alle byene?

Hvis vi effektivt kan løse problemet slik vi formulerte det først, kan vi også løse problemet med den andre formuleringen innen rimelig tid. Vi bestemmer øvre og nedre estimater for den korteste reiselengden ved stadig å teste på gjennomsnittet av de tidligere estimatene.

La oss se på et annet eksempel. La X være en endelig mengde positive heltall, for eksempel

$$X = \{2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37\}.$$

Er det mulig å dele X opp i to delmengder Y og Z slik at summen av tallene i Y og summen av tallene i Z blir like? I eksemplet over vil det ikke være mulig, fordi summen av alle tallene i X er et oddetall. Hvis vi imidlertid utvider tallmengden vår med tallet 39 er ikke svaret så opplagt. (Forfatteren har valgt ikke å avgjøre dette problemet selv, slik at leseren kan få lov til å bidra med noe).

I dette tilfellet ser vi også at det finnes en generell metode: Vi ser på alle oppdelinger av mengden X i to deler, summerer tallene i hver del og ser om svarene blir like. Problemet er igjen at metoden tar for lang tid, det finnes alt for mange delmengder av X . I vårt eksempel må vi se på $2048 = 2^{11}$ komplementære par av delmengder. Igjen finnes det opplagte måter å forbedre metoden på, men ingen har funnet noen generell metode som virker rimelig raskt også for store tallmengder.

Før vi går inn på en mer generell diskusjon, la oss se på et siste eksempel. Med fare for å drive tekstreklame for Microsofts produkter, la oss se på *Minesveiper*, et spill som følger med på lasset ved kjøp av visse typer programvare. I Minesveiper står man overfor et rektangel med $n \times m$ ruter hvor det er plassert k miner man i utgangspunktet ikke vet hvor ligger. Under spilllets gang kan vi velge mellom å plassere oss i en rute for å *observere* eller å *markere* at en rute har en mine. Når vi plasserer oss i en rute for å observere skal det ideelt sett skje en av to ting:

1. Det er en mine i ruten, vi blåses til himmels og har tapt kampen mot minene.
2. Det er ingen mine der og vi får opp et tall mellom 0 og 8 som forteller oss hvor mange av de åtte naborutene som har en mine.

Spillkonstruktøren har imidlertid forenklet spillet i forhold til det ideelle på ett punkt: Når vi får 0 i en rute, vet vi at ingen av de åtte naborutene inneholder noen mine. Vi kan derfor trygt foreta en videre observasjon fra hver av disse åtte rutene for å kartlegge hvor mange av deres naboruter som har miner. Spillkonstruktøren er tydeligvis redd for at dette blir for kjedelig, så når vi observerer fra en rute, får vi et bilde på skjermen som om alle disse trekkene hadde vært utført. Kjennere av spillet vil vite at tallet 0 ikke en gang skrives, det er markert som en blank rute.

Man trenger ikke å ha spilt dette spillet lenge før man forstår at om vi vet at en rute har én naborute med en mine, og vi allerede har markert en mine i en av naborutene, så er de syv andre rutene trygge. Det å sjekke ut alle disse er også litt kjedelig, så det hadde bare vært en forlengelse av servicen fra spillkonstruktørene om maskinen hadde spilt denne delen av spillet for oss også.

I andre situasjoner kan vi lese ut av brettet at en rute har for eksempel to umarkerte nabominer samtidig som det bare finnes to ledige felt som hverken er markert med miner eller brukt til observasjon. Da vet vi at det må ligge miner i begge disse feltene, og vi kan markere dem med miner. Siden denne delen av spillet også er litt kjedelig, hadde det ikke vært noe i veien for at maskinen hadde gjort det for oss.

Fortsetter vi denne tankegangen, hadde det beste vært om hver gang det er trygt å observere fra en rute, så gjør maskinen det for oss, og hver gang det er sikkert at det ligger en mine i en rute, så markerer maskinen minen for oss.

Nå skal vi ikke tro at spill-leverandøren har latt være å gjøre dette for oss bare for at vi skal ødsle mer tid på spillet, det er faktisk utenkelig at de har greid å lage et program som spiller Minesveiper for oss ved å utføre alle de trekkene vi kunne resonert oss til selv. Den engelske matematikeren Richard Kaye [1] har vist at det i prinsippet er like vanskelig å finne ut av om vi har sikker informasjon om en rute i Minesveiper som det er å avgjøre partisjonsproblemet eller å løse den handelsreisendes problem. En metode for å avgjøre om vi har sikker informasjon om en rute er:

Se på alle mulige fordelinger av de resterende minene på de gjenværende rutene som er forenlig med opplysningene i spillet så langt.

Hvis det finnes to slike fordelinger, en som plasserer en mine i den ruten vi ser på og en annen som ikke gjør det har vi ikke sikker informasjon, ellers har vi det.

Problemet med denne metoden er at det fort blir veldig mange måter å fordele minene på. Har vi for eksempel syv miner som skal fordeles på trettifire ledige ruter, har vi like mange muligheter som i et norsk lottospill. Det interessante er at det er liten grunn til å tro at det finnes en vesentlig mer effektiv metode.

Det finnes mange måter å formulere et problem på, men hvis vi skal kunne skrive et program som løser problemet må det presenteres på en slik måte at det kan bearbeides av en datamaskin. Hvis p er et problem, lar vi $|p|$ betegne antall bits vi trenger for å formulere problemet som data lesbart av en datamaskin.

En *problemfamilie* vil være en samling av problemer av samme art, hvor problemene kan presenteres som inngangsdata til en datamaskin. Den handelsreisendes problem er en slik problemfamilie. Det er også partisjonsproblemet og *minesveiperproblemet*:

Har vi sikker informasjon om innholdet av en rute på et gitt minesveiperbrett?

Vi skal nå presisere hva vi mener med rimelig raskt. Vi antar at leseren vet hva et *polynom* med én variabel og positive heltallskoeffisienter er for noe, et illustrerende eksempel er

$$f(x) = 3x^{17} + 13x^{15} + 2x^7 + 1024x^3 + 11.$$

Vi sier at en problemfamilie A kan løses i *polynomisk tid* hvis det finnes et program Q og et polynom f slik at om vi forer Q med beskrivelsen av et problem p i A , så gir Q oss svaret på problemet etter $\leq f(|p|)$ antall regneskritt.

Antall regneskritt er mer presist enn tiden som er brukt, derfor bruker vi det når vi definerer 'løsbarehet i polynomisk tid'. Det er imidlertid en tett sammenheng mellom antall regneskritt og tiden som vil bli brukt, så vi kunne like gjerne snakke om 'antall sekunder brukt' som 'antall regneskritt'.

Vi ser at det er noen upresise elementer i denne definisjonen. Vi har ikke relatert definisjonen til hvordan vi presenterer problemet p for en maskin, vi har ikke tatt hensyn til hva slags maskin vi bruker og vi har ikke tatt hensyn til hvilket programmeringsspråk vi skal benytte. Det viser seg at begrepet 'løsbare i polynomisk tid' er ganske robust i forhold til slike variasjoner.

Den presise matematiske definisjonen baserer seg på matematiske modeller for regnemaskiner, kjent som *Turingmaskiner*, modeller som ble utviklet av den engelske matematikeren Alan Turing for å kunne studere beregnbarhet med matematiske metoder. Klassen av problemfamilier som kan løses i polynomisk tid kalles \mathbf{P} . En slik klasse av familier kalles en *kompleksitetsklasse*.

Vi skal se på to eksempler.

Eksempel 1

Elever i videregående skole lærer å løse systemer av likninger med flere ukjente, og fortsetter man med matematikk på universitet eller høyskole lærer man effektive metoder for å løse forholdsvis store systemer av førstegradlikninger med mange ukjente. En metode er å løse første likning med hensyn på en av variablene, sette inn løsningen i de andre likningene og fortsette på den måten. Til sist finner man ut av om systemet har uendelig mange løsninger, nøyaktig en løsning eller ingen løsninger, og i de tilfellene det finnes én løsning finner man den. Hvis man tenker igjennom hvor lang tid man trenger for å løse et slikt sett ser vi at kvadratet av antall likninger kommer inn som en proporsjonalitetsfaktor, og hvis vi regner med brøkkuttrykk og ikke lange desimaltall, vil lengden av likningene, hvor både antall variable og størrelsen på koeffisientene spiller inn, være

en proporsjonalitetsfaktor. Det er selvfølgelig kjedelig å løse ti likninger med ti ukjente for hånd, men det er overkommelig. Antall symboler vi må skrive eller tiden vi trenger for å finne løsningen av et vilkårlig likningsett er begrenset av et polynom av grad høyst fire, trolig lavere. Familien av problemer knyttet til systemer av førstegradslikninger med flere variable og rasjonale koeffisienter ligger altså i P .

Det finnes gode programmer som løser enorme systemer av slike likningsett. De brukes som en del av teknologiske beregninger, for eksempel ved belastningsberegninger av broer eller oljeplattformer eller ved avlesning av bildet av et skadd kne eller en skadd hjerne i en "scanner" på et sykehus.

Eksempel 2

Endel påskenøttoppgaver går ut på å tegne en gitt figur uten å løfte pennen eller tegne en strek to ganger, å gå igjennom alle dører i et hus nøyaktig én gang eller å spasere i en by med flere broer (Königsberg, Stockholm, Trondheim og liknende steder) slik at hver bro benyttes nøyaktig én gang. Slike problemer er ganske like, vi har punkter som forenes med kanter, og hver kant skal brukes nøyaktig en gang. For byene regner vi hvert landområde (øyer, elvebredder og liknende) som et punkt og broene som kanter. For husene vil rommene være punkter og dørene være kanter.

Den sveitsiske matematikeren Euler, som lenge bodde i Russland, viste at en slik oppgave kan løses med en rundtur hvis og bare hvis det alltid er mulig å spasere fra et vilkårlig punkt til et annet og antall kanter ut fra et punkt alltid er et partall. Det betyr at hvert landområde må være forbundet med andre landområder med et partalls antall broer for at det skal være mulig å gå en slik rundtur. En slik figur med punkter og kanter kaller vi en *graf*, og Euler viste at spørsmålet om det finnes en rundtur i en graf som går via alle kantene, men aldri to ganger via samme kant, (en typisk problemfamilie) er i kompleksitetsklassen P . En slik rundtur kalles forøvrig en *Euler-sykel*.

Vi har sett på to eksempler på problemfamilier i kompleksitetsklassen P . Felles for disse to er at graden på polynomet vi trenger er lav og at det finnes metoder for å løse disse problemene i praksis. Dette er en erfaring som gjelder flere problemfamilier. Hvis en familie av problemer er naturlig og ligger i klassen P , finnes det som oftest en metode for å løse problemet som er gjennomførbart i praksis. Denne erfaringen bidrar også til at klassen P er en naturlig kompleksitetsklasse å studere.

Det store matematiske problemet er om de tre andre problemfamiliene vi har sett på, den handelsreisendes problem, partisjonsproblemet og minesveiperproblemet, er i P . Dette er formulert som det matematiske problemet

$$P = NP$$

og vi skal nå se på hva som kjennetegner et NP -problem.

Det vil føre for langt å gi en presis definisjon av kompleksitetsklassen NP , men vi skal finne noen trekk som er felles for våre tre eksempler.

1. For å bevise at det finnes en reiserute begrenset av en viss lengde er det

tilstrekkelig å gjette på den rette reiseruten og så verifisere at den er kort nok.

2. For å bevise at det er mulig å dele en gitt tallmengde i to delmengder med samme sum, er det tilstrekkelig å gjette på en god oppdeling og deretter verifisere at summene blir de samme.
3. For å bevise at vi ikke har sikker informasjon om en gitt rute i en gitt minesveiperstilling er det tilstrekkelig å gjette på to fordelinger av miner som er forskjellige for den aktuelle ruten og så bekrefte at de er i overensstemmelse med opplysningene så langt.

For å avkrefte hvert av tilfellene må vi vise at det er umulig å gjette rett.

NP står for *Nondeterministic Polynomial* og henspiller på at vi kan bevise en påstand ved hjelp av heldig gjetning. En *deterministisk* prosess er en prosess som følger forhåndsbestemte regler, mens en *ikke-deterministisk* prosess kan ha skritt som ikke er forhåndsbestemt. De kan eksempelvis være bestemt av terningkast eller gjetning. **NP** vil bestå av familier av problemer hvor vi kan svare positivt i de tilfellene hvor svaret er JA ved hjelp av gjetning og verifisering som samlet bruker et antall regneskritt begrenset av et polynom, men hvor antall mulige gjetninger vokser eksponensielt med størrelsen på det aktuelle problemet, slik at vi i det minste tilsynelatende ikke kan vise i rimelig tid at svaret er NEI, selv om vi er heldige. Det å skulle bevise at et **NP**-problem har positiv løsning kan sammenliknes med å spille lotto. Vi gjetter på en rekke og må bevise at dette er vinnerrekken. Sammenlikningen halter litt fordi det ikke alltid blir trukket ut en vinnerrekke i **NP**-spillene.

Problemet om $P = NP$ eller ikke er et av de syv *milleniumsproblemene*, problemer som ble betraktet som en spesiell utfordring ved starten av det nye årtuset, og som pengesterke folk satte en dusør på en million dollar stykket på. Hvis **NP**-problemene hadde begrenset seg til slike sære problemer som vi har sett på, hadde det vært uforståelig at $P = NP$ -problemet hadde blitt betraktet som viktig. I et litt større perspektiv kan imidlertid den handelsreisendes problem sees på som et problem om effektiv utnyttelse av ressurser, noe som selvfølgelig har stor praktisk og økonomisk betydning. Et annet område hvor **NP**-problemer oppstår er ved verifikasjon av logiske kretser eller integrerte kretser. En moderne datamaskin er bygget opp av kretser, hvor signalene som skal komme ut er bestemt på et komplisert vis av signalene som sendes inn. Problemet om det er en feil ved en slik krets, om det finnes måter å sende signalene inn på slik at vi ikke får det ønskede signalet ut, er et typisk **NP**-problem. Den amerikanske matematikeren/informatikeren Stephen Cook brukte nettopp logiske kretser, eller mer presist spørsmålet om en gitt formel i utsagnslogikken kan gjøres sann, som et eksempel på et **NP-komplett** problem. Kan vi løse et **NP-komplett** problem i polynomisk tid kan vi løse alle **NP**-problemer i polynomisk tid. De tre problemfamiliene vi har sett på er **NP**-komplette. Richard Kaye viste at en logisk krets kan omstruktureres til et minesveiperbrett slik at om vi avgjør problemer om minesveiperbrettet i polynomisk tid avgjør vi samtidig relevante problemer om den logiske kretsen. Detaljene er tilgjengelige i [1].

Nå skal vi ikke overdrive den praktiske betydningen av å få løst $\mathbf{P} = \mathbf{NP}$ -problemet. For mange viktige \mathbf{NP} -problemer finnes det gode og raske programmer som gir oss et godt svar som ikke nødvendigvis er det teoretisk sett beste (når det gjelder ressursutnyttelse) eller et svar som er rett med stor grad av sannsynlighet om vi ber om et JA/NEI-svar. Lærdommen er likevel at \mathbf{NP} -problemer forekommer ofte, og det kreves matematisk innsats for å håndtere hvert enkelt av dem tilfredsstillende. Det faller utenfor rammen for denne artikkelen å gå videre inn på det sporet.

References

- [1] R. Kaye, *Minesweeper is NP-complete*, The Mathematical Intelligenser, Vol. 22, No 2, pp. 9 - 15 (2000).