INF 4130
15. oktober 2009

Stein Krogdahl

NB: Det som under forelesningen ble kalt et ”vitne” er nå omdøpt til et ”sertifikat”.

- Dagens tema: NP-kompletthet
  - Eller: hvilke problemer er umulig å løse *effektivt?*
- Også her:
  - Dette har blitt framstilt litt annerledes tidligere år
  - Se Dinos forelesninger fra i fjor
  - I år: Vi tenker mer i programmer enn i Turing-maskiner
  - Pensum blir foilene, og en del fra læreboka, kap. 26
- Neste uke: Mer NP-kompletthet
  - Hvordan kan vi generelt vise at problemer er NP-komplette?

Pensum

angående avgjørbarhet og NP-kompletthet

- Angående avgjørbarhet:
  - Foilene fra forelesningen 8/10
  - Støttelitteratur (ikke pensum):
    - Kompendiet til Karebeg og Djurhuus (se fjorårets pensumliste)
    - Oppkopierte sider fra ”Algorithmics” (kap. 8)

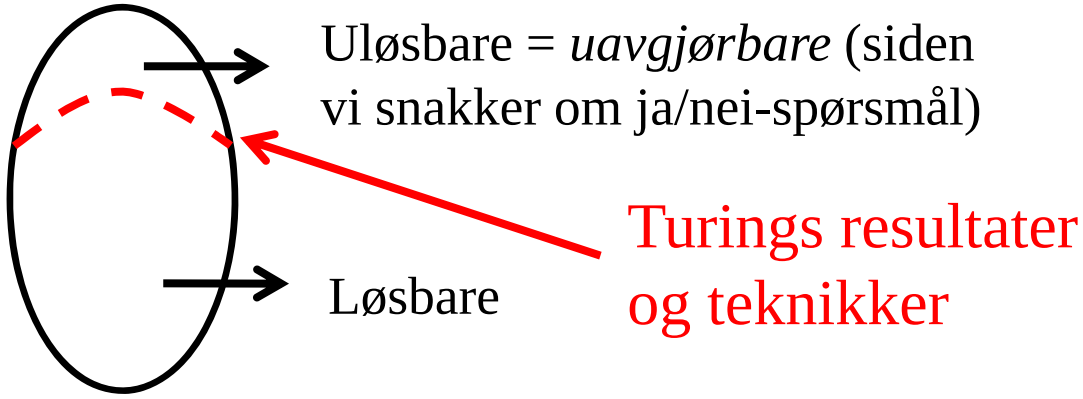
- Angående NP-kompletthet:
  - Foilene fra forelesningene 15/10 og 22/10
  - Boka, kap 26, til og med 26.5 (*kanskje* noe utgår)
  - Støttelitteratur (ikke pensum):
    - Kompendiet til Karebeg og Djurhuus (se fjorårets pensumliste)
    - Boka ”Algorithmics” (kap 7). Må eventuelt skaffes selv, forfatter: David Harel.
    - Boka ”Computers and Intractability”, av Garey og Johnson

Litt historie: Hva kan ikke gjøres

Fra 1900 og utover vokste feltet som kalles metamatematikk (matematiske studier av matematikken selv, dens muligheter og begrensninger).

På 1930-tallet kom en del resultater omkring eksistens/ikke-eksistens av algoritmer for ulike problemer.

- Kurt Gödel (1931): Ufullstendighetsresultater.
- Alan Turing (1936): «*On computable numbers, with an application to the Entscheidungsproblem*».

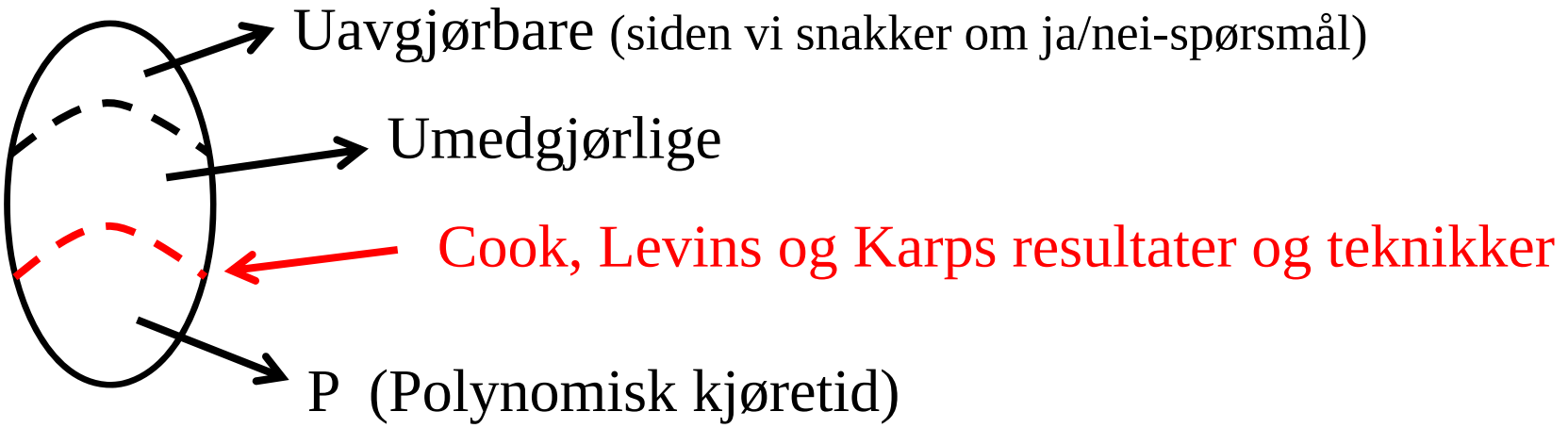


Historie: Hva kan gjøres, men ikke raskt

Edmonds, 1965: «*min algoritme har polynomisk kjøretid, den trivielle bruker eksponensiell tid!*»

Vi har altså nå resultater omkring problemer som har algoritmer med polynomisk kjøretid.

Cook / Levin (1971): NP-completeness. Også en Karp var sentral



Også her: Hva er et ”problem”?

Men her må vi også tenke på: Hva er en instans *størrelse*?

- Et problem P består av en mengde *instanser* {P_1, P_2, ...} som er de egentlige konkrete problemene, og som hver har et svar.
  - Vi skal her først og fremst snakke om ”desisjons-problemer”, altså problemer der hver enkelt instans har et svar *Ja* eller *Nei.*
- En *koding* av et problem P, K(P), er en tekstlig sekvensiell representasjon av hver av instansene til problemet.
- Men det blir nå også viktig hvor lang (regnet i antall tegn) denne kodingen blir for en gitt instans. Vi kaller denne lengden $n_i = |K(P_i)|$.
  - Ofte blir den regnet i antall bit, men forskjellig alfabet-størresle gir bare lineær forskjell i lengden
  - F.eks. trengs fire ganger så mange tegn om du skal kode noe i et 2-tegns-alfabet, enn om du skal kode det i et alfabet med 16 tegn.
- En algoritme A *som løser* (eller *avgjør*) P *i polynomisk tid* er en algoritme som ved input $K(P_i)$ leverer riktig svar på P_i etter $t(n_i)$ *skritt,* der $t(x) = O(x^k)$ for en gitt konstant k.

Hvordan måle lengden av instanser

Spørsmålet om et problem er løsbart i polynomisk tid er derved avhengig av hva slags koding K man bruker for problemet!

- Kodingen skal ikke være ”unødvendig dum” mht. plassbehov

  I de fleste tilfelle vil lengden av forskjellige kodinger bare forholde seg lineært (eller hvertfall polynomisk) til hverandre, og da spiller kodingen ingen rolle for hva som er en polynomisk algoritme.

- Men, spesielt for tallverdier må man passe seg. Det de bidrar med i lengden av instansen bør i en rimelig koding være antall siffer (= log (tallverdien))

- Og IKKE tallverdien selv, som altså er eksponensiell i forhold til antall siffer.

Problemklassen *P*

- *P* er klassen av de (desisjons-problemer) som er løsbare i polynomisk tid
- Der ligger jo svært mange av de problemer vi studerer i våre kurs (under satt opp som *optimerings-problemer*):
  - Finn korteste vei gjennom en graf med positive kantlengder
  - Finn letteste spenntre i en graf med kant-vekter
  - Har denne grafen en løkke? (Er allerede et desisjons-problem)
  - ...
- Vi tenker altså ofte på disse som optimerings-problemer
  - men i sammenhengen her bør vi tenke på desisjons-varianten av dem, f.eks.: ”Finnes det et spenntre som er lettere enn en oppgitt verdi”
  - Man kunne tenke seg at optimerings-varianten er vesentlig vanskeligere enn desisjons-varianten, men det viser seg som regel å ikke være tilfelle (Vi kan bruke binærsøking etter optimal-verdien med desisjons-algoritmen som test under søket).

En observasjon *nesten* som for uavgjørbarhet:

For et problem som har et *endelig antall instanse*r er det ikke meningsfylt å snakke om at en algoritme er polynomisk eller ikke:

- Vi antar at instansene for P er $P_1, P_2, \ldots, P_N$
- Følgende algoritme vil da løse problemet:

```
if  <input = P1> then <levér svaret på P1> else
if  <input = P2> then <levér svaret på P2> else
…
if  <input = PN> then <levér svaret på PN> ;
```

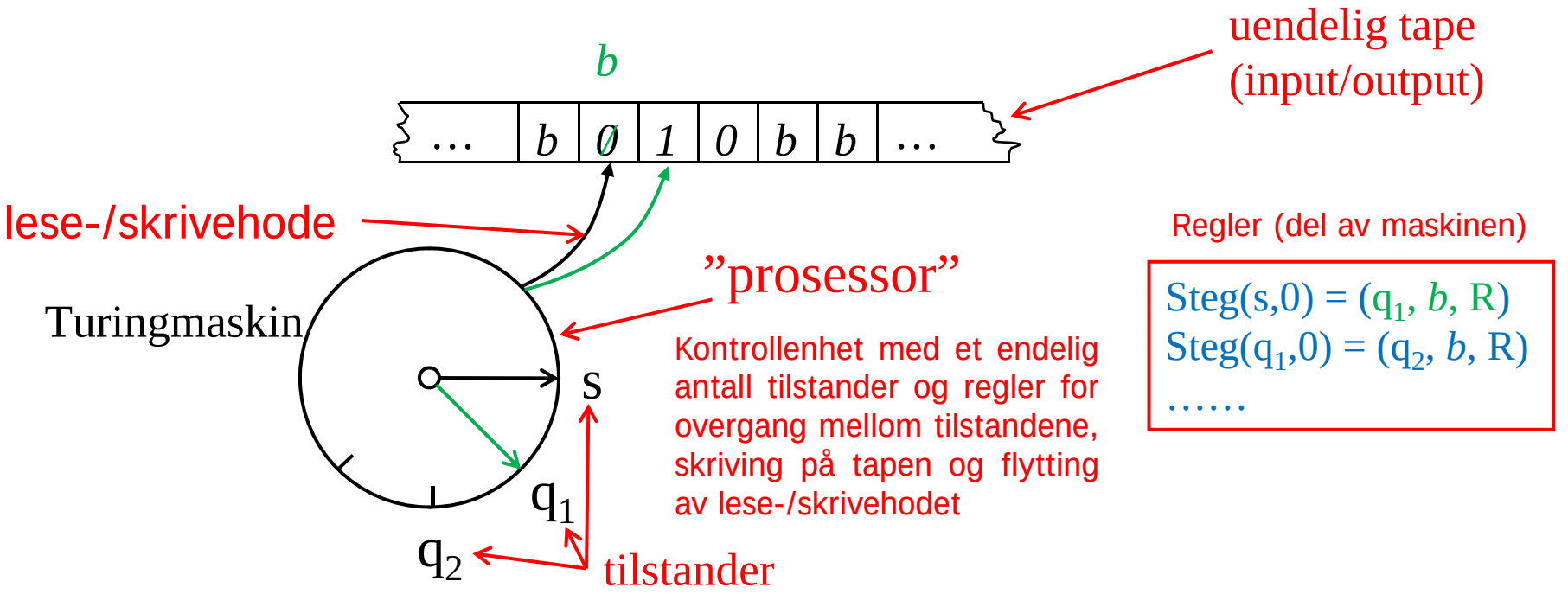
- Det finnes dermed her en maksimal tid algoritmen vil bruke på noen instans.
- Vi kunne da helst si at algoritmen går i konstant tid, altså er *O(1)*. Men dette sier jo egentlig lite.

Vi må nå være nøye med hva som er et elementært skritt

- Vi må nå også være nøye med at det vi definerer som elementære skritt i utførelsen av en algoritme (eller et program) faktisk kan utføres på en tid vi har kontroll over:
  - Helst ville vi at hvert skritt kunne utføres innen en fast tid.
  - Dette er imidlertid upraktisk, siden vi da ikke kunne regne f.eks. en addisjon som elementær operasjon.
  - Vi tillater derfor at et elementært skritt kan ta varierende tid, men det får bare lov å ta *polynomisk tid* i forhold til størrelsen av instansen.
  - Dette vil ikke forandre hva som er polynomiske algoritmer.

- Derfor er det OK å definere følgende som elementære skritt:
  - Addisjon: Lineær i antall siffer
  - Multiplikasjon: kvadratisk i antall siffer.

En turing-maskin går sakte, men …

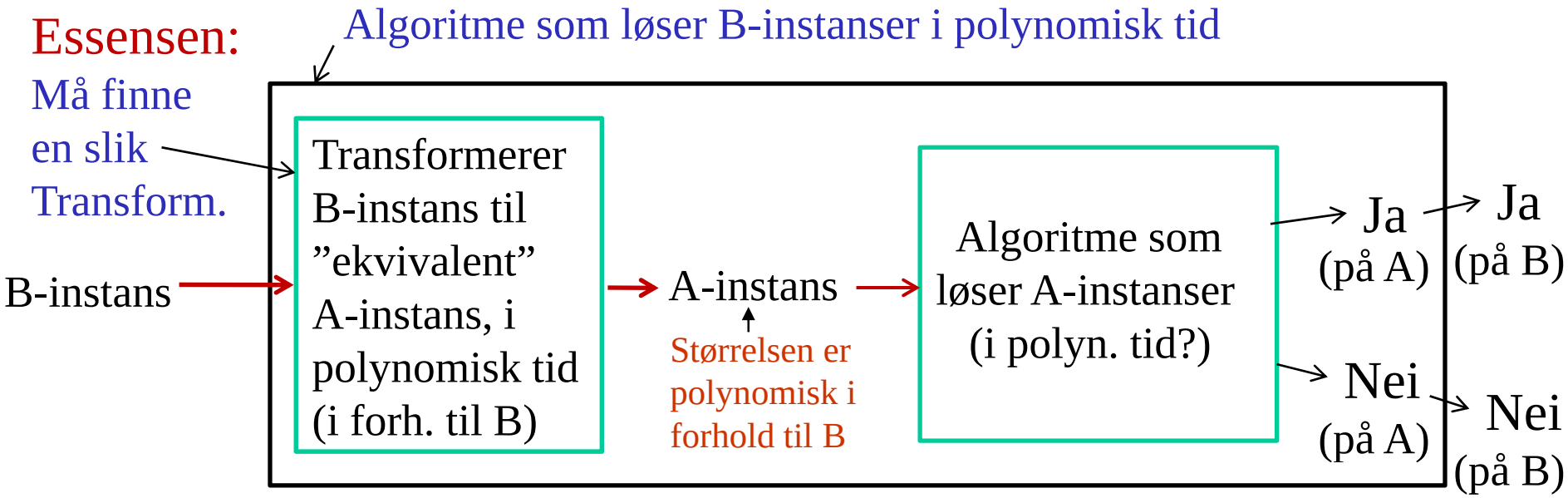


Er maskin-modellen viktig for hva som kan gjøres i polynomisk tid?

- Det er klart at forskjellige maskin-modeller vil bruke forskjellig tid på en gitt algoritme.
  - Tenk på en Turing-maskin som må flytte seg ett og ett hakk langs tapen for å komme dit en bestemt verdi er lagret
  - Mans andre modeller har direkte-aksess til lageret
- Det ”fantastiske” er at alt en moderne kraftig datamaskin kan gjøre i polynomisk tid:
  - Det kan også en Turing-maskin gjøre i polynomsik tid!!
- Vi skal ikke forsøke oss på noe bevis av dette, men klassen *P* er derfor veldig robust:
  - Om vi definerer *P* i forhold til en Turing-maskin eller en moderne RAM-maskin så får vi samme mengde av ”polynomisk algoritmer”.
  - (Dog vil det ikke være samme grad på polynomet for alle modeller)
- Altså: Vi slipper å tenke på maskinmodell i denne teorien.

Vi vil vise følgende ved hjelp av ”polynomisk reduksjon”:

Om A er løsbart i polynomisk tid, så er også B det



- ”Ekvivalent” betyr her at den A-instansen som produseres er en ja-instans hvis og bare hvis B-instansen er en ja-instans.
- Om vi klarer å finne en slik transformasjon, så har vi vist setningen over.
- Vi sier da at B er *polynomisk reduserbar* til A, som kan skrives $B \propto A$
- Vi vet da også at om B *ikke* er løsbar i polynomisk tid, så er *heller ikke A det!*

Problemer man ikke kjenner noen polynomisk algoritme for

Det viser seg å være en rekke problemer man ikke har funnet noen polynomisk algoritme for. Et (veldig) lite utvalg:

Hamiltonsk løkke (HAM)
Instans: En urettet graf G
Spørsmål: Er det en enkel løkke i G som er innom alle nodene?

Satisfiability (SAT)
Instans: Et logisk uttrykk på formen: (b1 $\vee$!b2) $\wedge$ (b3 $\vee$!b1 $\vee$ b2) $\wedge$ (...) $\wedge$ ...
Spørsmål: Finnes det en verdisetting av b1, b2, ... slik at uttrykket er **true** ?

Subset sum (SUBSET SUM)
Instans: Gitt en sekvens av positive heltall, og et positivt heltall K
Spørsmål: Finnes det et utplukk av tallene i sekvensen som har sum K?

- Merk: Dette siste problemet har vi løst med dynamisk programmering i Oblig1, under navnet ”sum av utplukk” (SAU).
- Den algoritmen vi brukte brukte tid O(K*Sekv-lengde), og siden K er eksponensiell i forhold til antall siffer i K er denne eksponensiell

Med polynomisk reduksjon kan vi vise at to problemer er ”ekvivalente” mht. polynomisk eller ikke-polynomisk løsbarhet

- Vi vil vise for problemene A og B at enten er begge løsbare i polynomsk tid eller så er ingen av dem det.
- Greit nok, vis følgende:
  - Om problemet A er løsbart i polynomisk tid, så er også B det
  - Om problemet B er løsbart i polynomisk tid, så er også A det
- Dette vises ved å vise h.h.v. $B \propto A$ og $A \propto B$
  - Se bevis-metode på tidligere foil
- Da vet vi også:
  - Om B *ikke* er løsbart i polynomisk tid, så er heller ikke A det
  - Om A *ikke* er løsbart i polynomisk tid, så er heller ikke B det
- Dermed vet vi at A og B er ekvivalente mht. polynomisk eller ikke-polynomisk løsbarhet

Med dette kan man ”lett” vise at:

- De problemene vi så på tidligere (HAM, SAT osv.)
  - Samt en rekke andre ja/nei-problemer som vi heller ikke har funnet nonen polynomisk algoritme for
- Er ekvivalente på følgende måte:
  - Enten er *alle* løsbare i polynomisk tid, ellers er *ingen av dem* det.
  - Dette kan lettest vises ved en ”ring” av reduksjoner (slår flere sammen i en ”smekk”, og viser ikke bare at $A \propto B \propto A$, men at):

$$A \propto B \propto C \propto D \propto \ldots \propto A$$

- Vi kan kalle disse problemene for ”vriene problemer”
  - Vi har altså *ikke* vist at de ikke er polynomisk løsbare
  - Men det er enten alle eller ingen!
  - NB: Betegnelsen ”vriene problemer” skal straks erstattes av noe annet!

De ”vriene” problemene:

- Vi vet da om disse problemene:
  - Kanskje er *ingen* av dem løsbare i polynomisk tid (ingen har funnet noen slik algoritme, men hvem vet?!).
  - Men *om* vi finner en polynomisk algoritme *for bare én av dem* så er *alle* løsbare i polynomisk tid.
  - Ingen har klart å vise at de *ikke kan* løses i polynomisk tid
  - Men her ligger en Nobel-pris, en Turing Award eller en Abel-pris om man klarer å vise det!

- Men vi kan vise mer enn vi har gjort til nå!

... Vi ser på NP-teorien: ...

Hva er *NP* og *NP-kompletthet*?

- NP er en klasse av problemer
  - som også omfatter alle problemer i *P*

- NP står *ikke* for "Not solvable in Polynomial time" e.l.
- Snarere er det en *oppad avgrensing* av problemenes vanskelighet.
  - NP står for "*Non-deterministic Polynomial*"
  - Men denne problem-klassen skulle (som vi skal se) heller hett *"Polynomisk Sertifiserbar" e.l.*

- Problemene i NP er de som *ikke er verre enn* følgende:
  - For hver ja-instans skal det finnes et såkalt "sertifikat" av polynomisk størrelse (altså med et fast polynom for alle instanser!)
  - Og om noen for en instans A oppgir et slikt sertifikat så skal det kunne sjekkes i polynomisk tid om dette faktisk er et sertifikat som viser at A er en ja-instans
  - Størrelsen på et slikt sertifikat må derfor også være polynomisk

Hamiltonsk Løkke er i NP

- Vi ser på problemet Hamiltonsk Løkke (HAM):

  Instans: En urettet graf

  Spørsmål: Finnes det en enkel løkke i denne grafen som er innom alle nodene?

  – Et *sertifikat* på at en gitt instans her er en ja-instans er rett og slett å angi en slik løkke (f.eks. som en sekvens av noder)
  – Det er da lett å sjekke (i pol. tid) at dette faktisk er en Hamiltonsk Løkke for grafen.

En ja- og en nei-instans av problemet Hamiltonsk Løkke, med et sertifikat angitt for ja-instansen.

Alle problemer i NP er avgjørbare

Gitt et problem i NP: For å løse en gitt instans av det kan man ”bare”:

- Gå gjennom alle mulige strenger som kan tenkes å utgjøre et sertifikat:
  - Man har et polynom som avgrenser hvor lange sertifikater maksimalt kan være for dette problemet.
  - Dermed er det (for hver instans) bare et endelig antall sertifikat-kandidater Vi kan rett og slett se på alle strenger som ikke er lenger enn dette (selv om de fleste da vil være bare tull).
  - Men det kan være et eksponensielt antall av dem (men det gjør jo ingen forskjell for avgjørbarheten)
- For hver av disse sertifikat-kandidatene tester vi om de utgjør et ja-sertifikat for den aktuelle instansen
  - Hver slik test kan pr. def. gjøres i polynomisk tid, og dermed har vi en algoritme for å sjekke om dette er en ja-instans.
  - For HAM kan vi konkret gjøre dette ved å se på alle rekkefølger av nodene, og sjekke om denne utgjør en løkke i grafen

Hvorfor kalles denne problemklassen ”Non-deterministic Polynomial”

- Det er fordi teoretikere innen feltet liker å forestille seg en såkalt ”ikke-deterministisk” maskin:
  - Dette er en maskin som når den stilles overfor et valg ”ikke-deterministisk” kan velge hvilket valg den vil forfølge
  - Og man sier at en slik maskinen *løser et problem* hvis det finnes en *mulig eksekvering (= sekvens av valg)* som svarer ja dersom dette er en ja-instans.
- Man sier at en slik maskinen går i polynomisk tid om:

  det finnes et polynom som begrenser hvor lang tid det kan ta før et eventuelt ja-svar blir avgitt.

”Non-deterministic Polynomial” II

- En slik ”ikke-deterministisk” maskin kan løse f.eks. HAM i polynomisk tid som følger:
  - Start fra en tilfeldig node, og første valget blir å velge en av kantene som går ut fra denne, og så gå til den andre endenoden
  - Hver gang man kommer til en ny node får man så et nytt valg, mellom alle de kantene som går til noder man ikke har vært ved tidligere. Velg en av dem, og gå til den andre endenoden.
  - Om man så kommer tilbake til startnoden kan man svare ja.
  - Og begrensningen på antall steg til et evenuelt ja-svar er når vi har gjort like mange valg som det er noder i grafen.
- I stedet for å si at man må gjøre valg under veis kan man heller (slik foreleseren liker) tenke seg at:
  - man har mange maskiner til disposisjon, og at man i stedet for å velge setter i gang én maskin for hvert av alternativene.
  - Da får man masse maskiner som går i parallell, og om en svarer ja er svaret ja, ellers kan vi vente den riktige tid og angi svar nei.

Hvordan kan Cooks teorem bevises?

Enkel skisse!

- Det han gjorde var å se på et helt generelt problem i NP
  - Og tenke seg en gitt instans A av dette
  - Og så se på hva betingelsene er for at en generell streng (som altså er polynomisk begrenset i lengde) er et ja-sertifikat for A.
  - Sertifikatet representerte han som et antall boolske variable
- Han klarte så å sette opp en betingelse (på ”SAT-form”, i de boolske variable) som inneholdt alle kravene for at dette skal være et ja-vitne for denne instansen.
  - Det betydde at dette logiske uttrykket var tilfredstillbart hvis og bare hvis det fantes et ja-sertifikat for instansen A.
  - Og det logiske uttrykket var polynomisk i størrelse i forhold til A (noe som først og fremst kommer av at sertifikat-lengden er polynomisk i forhold til A).
- Dermed hadde han altså vist at et vilkårlig problem A i NP kan reduseres til SAT, altså at $A \propto SAT$