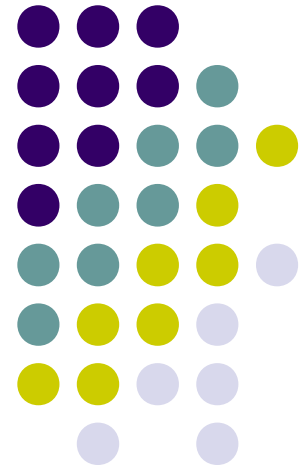


# Øvingsforelesning 12

---

P vs NP

Håkon Jacobsen  
hakoja@stud.ntnu.no

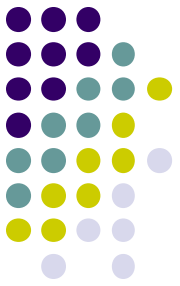


# Dagens tema



- Introduksjon og motivasjon for problemet
- Kompleksitetsklasser
  - P, NP og NPC
- Redusering av problemer
- Eksamenseksemppler

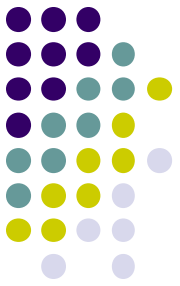
# Introduksjon



Dagens datamaskiner er veldig raske

Men enkelte problemer tar allikevel  
alt for lang tid til at vi kan løse dem

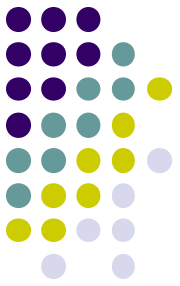
# Eksempel



$$7 \times 13 = ?$$

"Multiplikasjonsproblemet"  
(Løsning: 91)

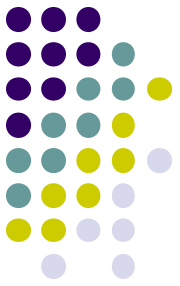
# Eksempel



$$? \times ? = 91$$

"Faktoriseringsproblemet"  
(Løsning: 7 x 13)

# Et litt større multiplikasjonseksempel



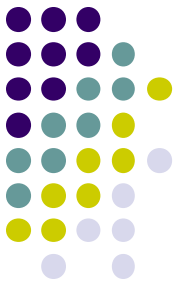
$$\begin{array}{r} 33987174230284385545 \\ 30123627613875835633 \\ 98649596959742349092 \\ 9302771479 \end{array} \times \begin{array}{r} 6264200187401285096 \\ 1516549482644422193 \\ 0203717862350901911 \\ 1660653946049 \end{array} = ?$$

Løsning:

21290246318258757547497882016271517497806703963277  
21627823338321538194998405649591136657385302191831  
6783107387995317230889569230873441936471

Tok under ett sekund å utføre på WolframAlpha

# Et litt større faktoriseringsseksempel

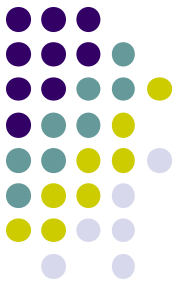


$$\begin{array}{r} ? \times ? = \\ 212902463182587575474978820162715 \\ 174978067039632772162782333832153 \\ 819499840564959113665738530219183 \\ 167831073879953172308895692308734 \\ 41936471 \end{array}$$

$$\begin{array}{r} 339871742302843855 \\ 453012362761387583 \\ 563398649596959742 \\ 3490929302771479 \end{array} \times \begin{array}{r} 626420018740128509 \\ 615165494826444221 \\ 930203717862350901 \\ 9111660653946049 \end{array}$$

Brukte rundt 2000 MIPS-år i 1999

# RSA-212

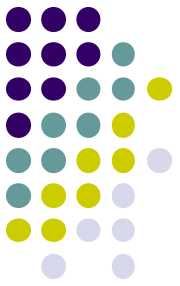


74037563479561712828046796097429573142593188889231289084936  
23263897276503402826627689199641962511784399589433050212758  
53701189680982867331732731089309005525051168770632990723963  
80786710086096962537934650563796359

**\$30 000**

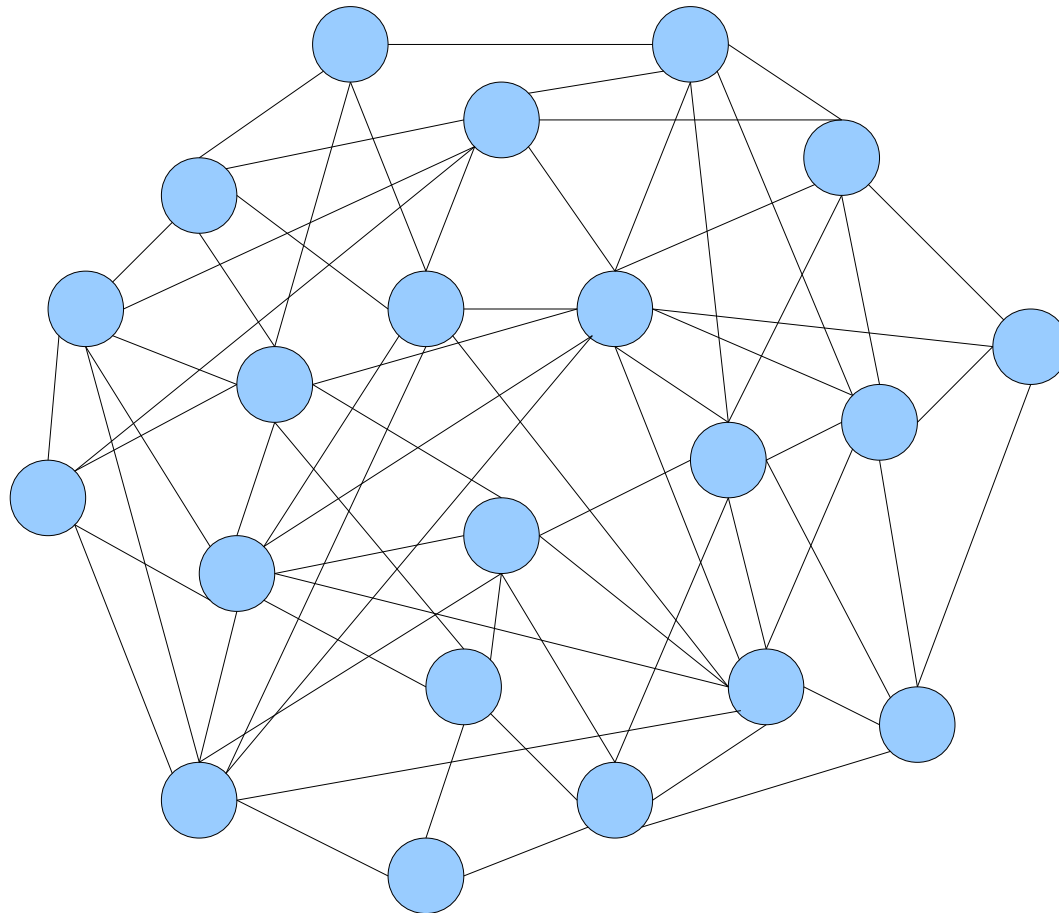
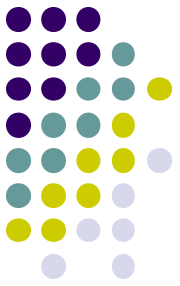
Ingen har foreløpig klart dette og fortsatt  
gjenstår: RSA-220, RSA-230, ..., RSA-617

# Hvorfor er faktorisering så vanskelig?

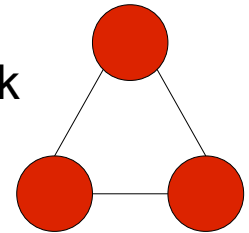


- Del med 2, 3, 5, 7, 11, ... helt til du finner en faktor
- Brute-force-søking er veldig tregt når mulighetsrommet er stort
- Men er søking nødvendig?

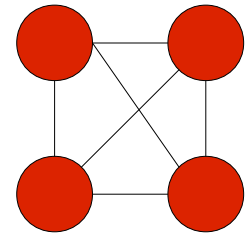
# Clique-problemet



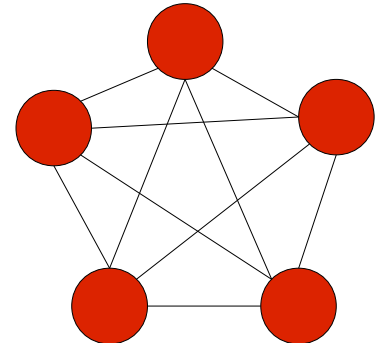
3-klikk



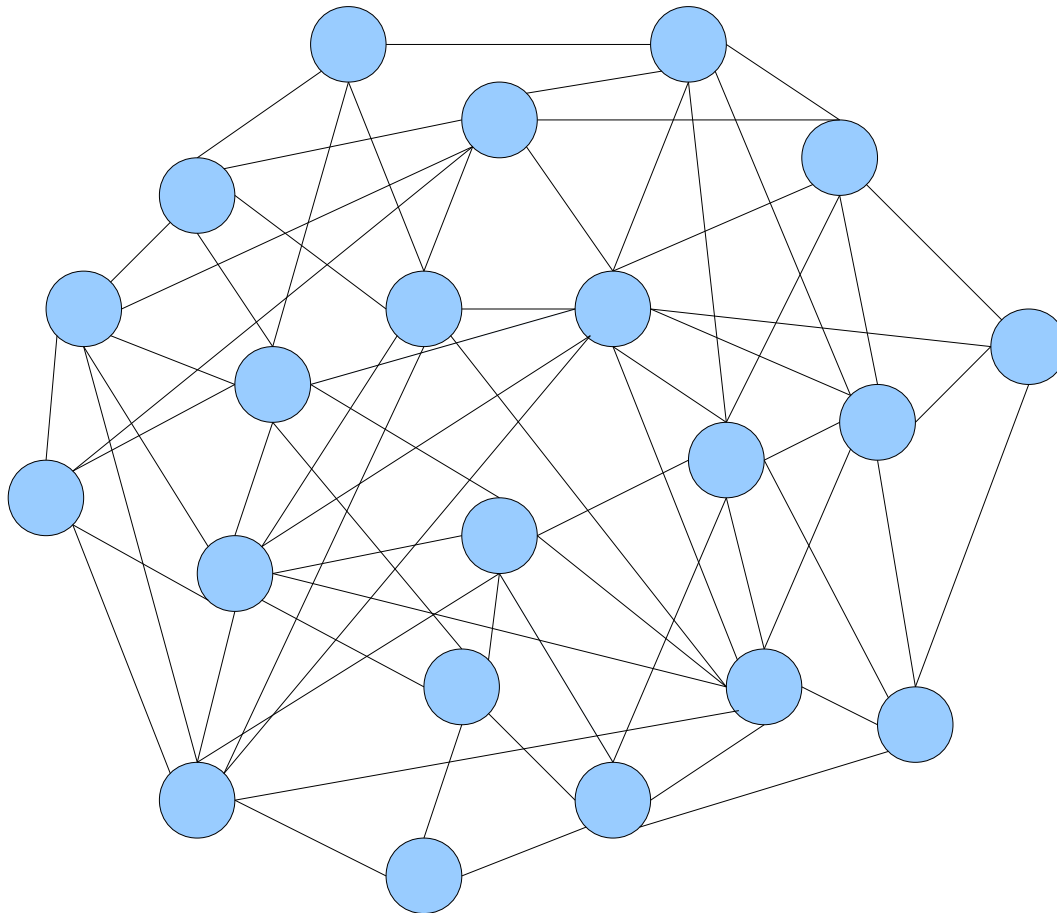
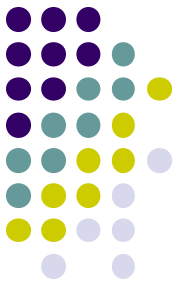
4-klikk



5-klikk



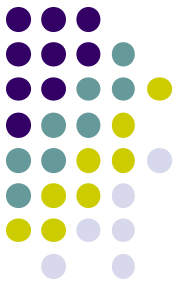
# Clique-problemet



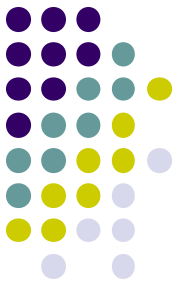
Å finne den største klikken i en graf som inneholder over 100 noder kan ta flere århundrer å finne ved brute-force-søking

Men er det nødvendig?

# Andre søkeproblemer



- Scheduling (planlegging)
- Fargelegging av kart
- Proteinfolding
- Teorem-beviser
- Sudoku
- Stabling av esker
- Design av microchiper
- Traveling salesman
- + velding mange flere...

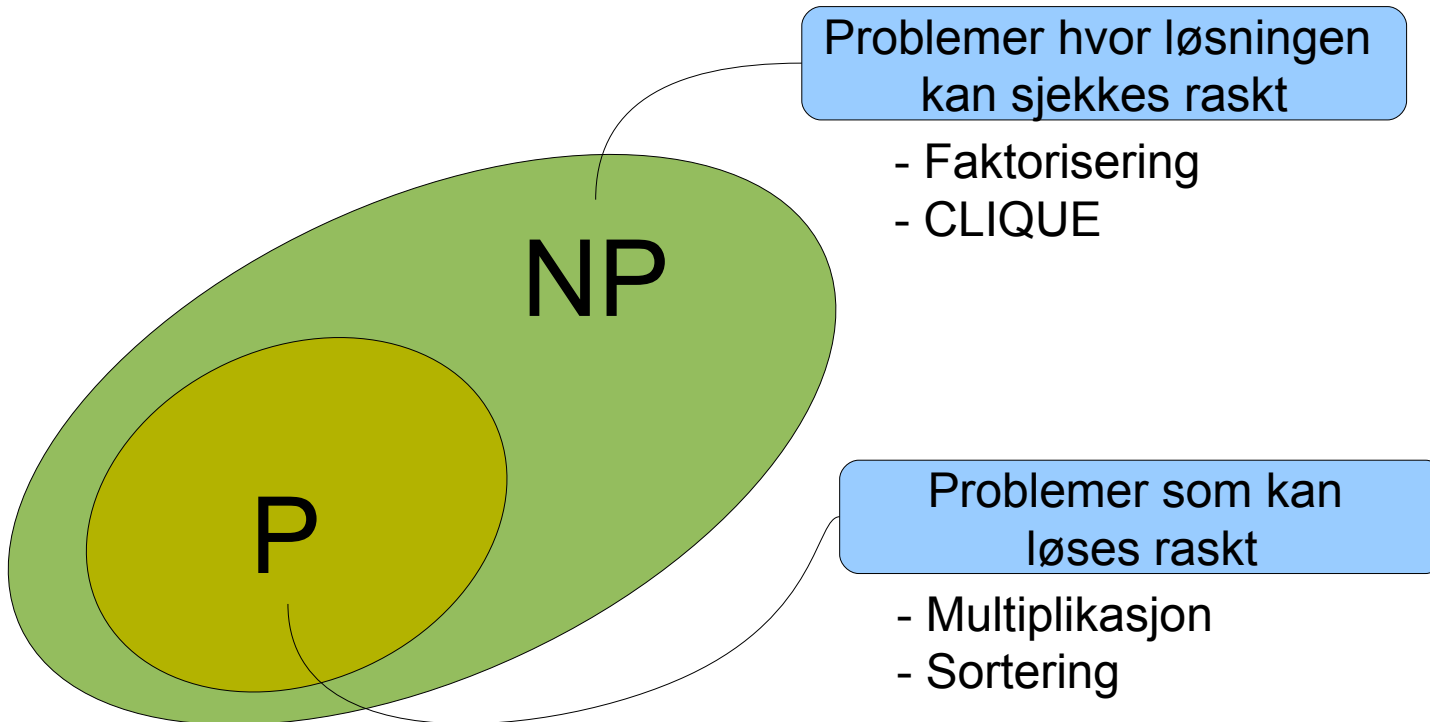
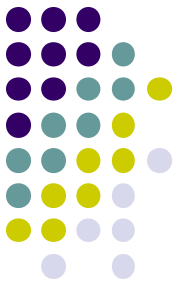


# P og NP

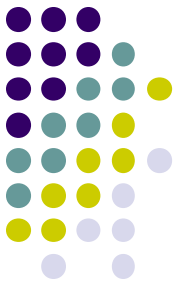
- P – "polynomial time"
  - Problemer som kan *løses* raskt
- NP – "nondeterministic polynomial time"
  - Problemer hvor en løsning kan *verifiseres* raskt
  - Inkluderer søkeproblemene

Merk:  $P \subseteq NP$

# P og NP som problemklasser

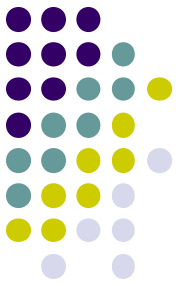


# Det store spørsmålet



Er  $P = NP$   
eller  
 $P \neq NP$ ?

# Litt historie



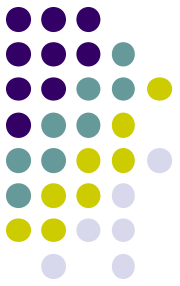
Alan Turing, 1912-1954



Alonzo Church, 1903-1995

- På 30-tallet begynner man å tenke over de fundamentale begrensningene en *regnemaskin* må ha?
- Lager en teoretisk modell for datamaskinen slik vi kjenner den i dag
  - Turingmaskiner
  - $\lambda$ -calculus
- Formaliserer begrepet *algoritme*
- Church-Turing-tesen

# Litt historie - kompleksitetsteori

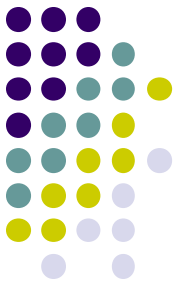


- 1960-tallet: Kompleksitetsteorien dannes, P og NP defineres
  - Hvor effektive er algoritmene våre?
- 1970-tallet: NP-kompletthet defineres
  - Edmonds, Rabin, Karp, Levin, Cook...
- 1956: Uoppdaget brev fra Gödel til von Neumann
  - Gödel kjente allerede til den moderne formuleringen av problemet



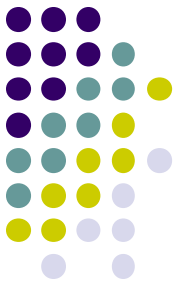
Kurt Gödel, 1906-1978

# Clay Mathematics Institute – Millenium Prize Problems



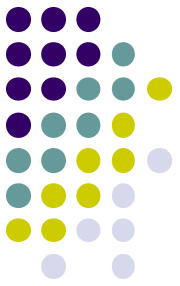
- ✓ Poincaré-formodningen
- P vs NP
- Navier-Stokes' problem
- Riemann hypotesen
- Birch- og Swinnerton-Dyer-formodningen
- Yang-Mills teori
- Hodge-formodningen

\$1 000 000



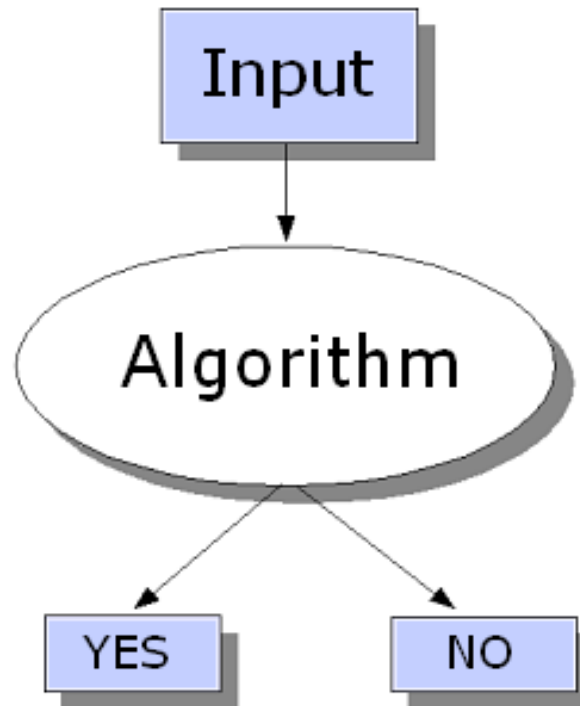
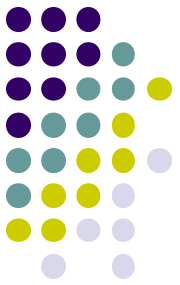
# Kompleksitetsklasser (Problemklasser)

# Avgjørelsesproblemer

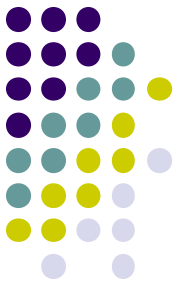


- Vi skal ta for oss avgjørelsesproblemer, som er nært beslektet med optimaliseringsproblemer
- Avgjørelsesproblemer: Problemer der svaret er enten **ja** eller **nei** (evt. 1/0, True/False, Up/Down, etc...)
- Optimaliseringsproblem: Finne det optimale svaret av flere mulige svar.

# Avgjørelsesproblemer

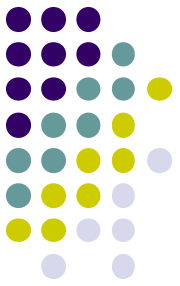


# Avgjørelsesproblemer

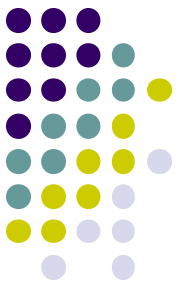


- SHORTEST-PATH (Optimaliseringsproblem)
  - Finne korteste vei fra  $u$  til  $v$  i en urettet, uvektet graf
- PATH (Relatert avgjørelsesproblem)
  - Finne ut om det finnes vei fra  $u$  til  $v$  med max  $k$  kanter

# Avgjørelsesproblemer



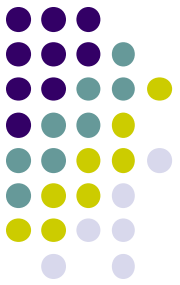
- Optimaliseringsproblemer har ofte et tilhørende avgjørelsesproblem
  - Anta at vi kan løse SHORTEST-PATH kjapt:  
Vi kan da sammenlikne svaret med parameteren  $k$  til PATH, og vi vet da at vi også har løst PATH kjapt
  - Anta motsatt, at vi kan løse PATH kjapt: Spør om det finnes en sti fra  $u$  til  $v$  med  $V-1$  kanter. Hvis ja: spør om det finnes en sti med  $(V-1)/2$  kanter, osv... (mao. Binærsøk). Vi kan dermed løse SHORTEST-PATH via PATH.
- Problemklassene P, NP, og NPC inneholder avgjørelsesproblemer



# Problemlassen P

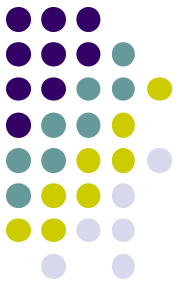
**P:** Mengden av avgjørelsesproblemer som kan løses i polynomisk tid

- Polynomisk tid:  $O(n^k)$ , der  $k =$  konstant og  $n =$  input-størrelsen
  - Eksempel:  $O(1)$ ,  $O(\log n)$ ,  $O(n)$ ,  $O(n^2)$ ,  $O(n^{100})$
- Problemer i P kalles ofte **tractable** (på godt norsk) og algoritmene som løser dem **effektive**



# Noen problemer i P

- Primtallstesting
  - Først bevist i 2002
- Lineærprogrammering
- Sortering
- Korteste vei
- Maks flyt
- ...omtrent hele pensum

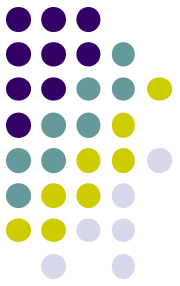


# Problemklassen NP

**NP:** Mengden av avgjørelsesproblemer der en korrekt løsning kan verifiseres i polynomisk tid

- Dvs. at hvis vi blir gitt en løsning på problemet kan vi verifisere at den er korrekt i polynomisk tid
- NP står for *nondeterministic polynomial time*

# COMPOSITES vs PRIMES



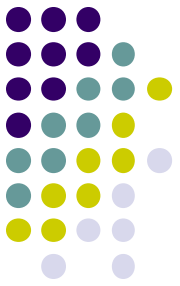
- Dette er et sammensatt tall

15226050279225333605356183781326374297180  
68114961380688657908494580122963258952897  
654000350692006139

- ...fordi den har følgende faktor:

37975227936943673922808872755445627854565  
536638199

# COMPOSITES vs PRIMES

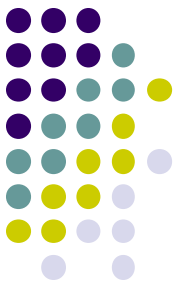


- Dette er et primtall

3347807169895689878604416984821269081770479498371  
3768568912431388982883793878002287614711652531743  
087737814467999489

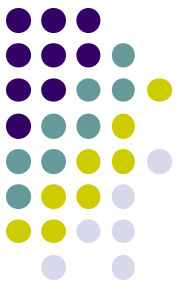
- ...fordi...

???



# Problemklassen NP

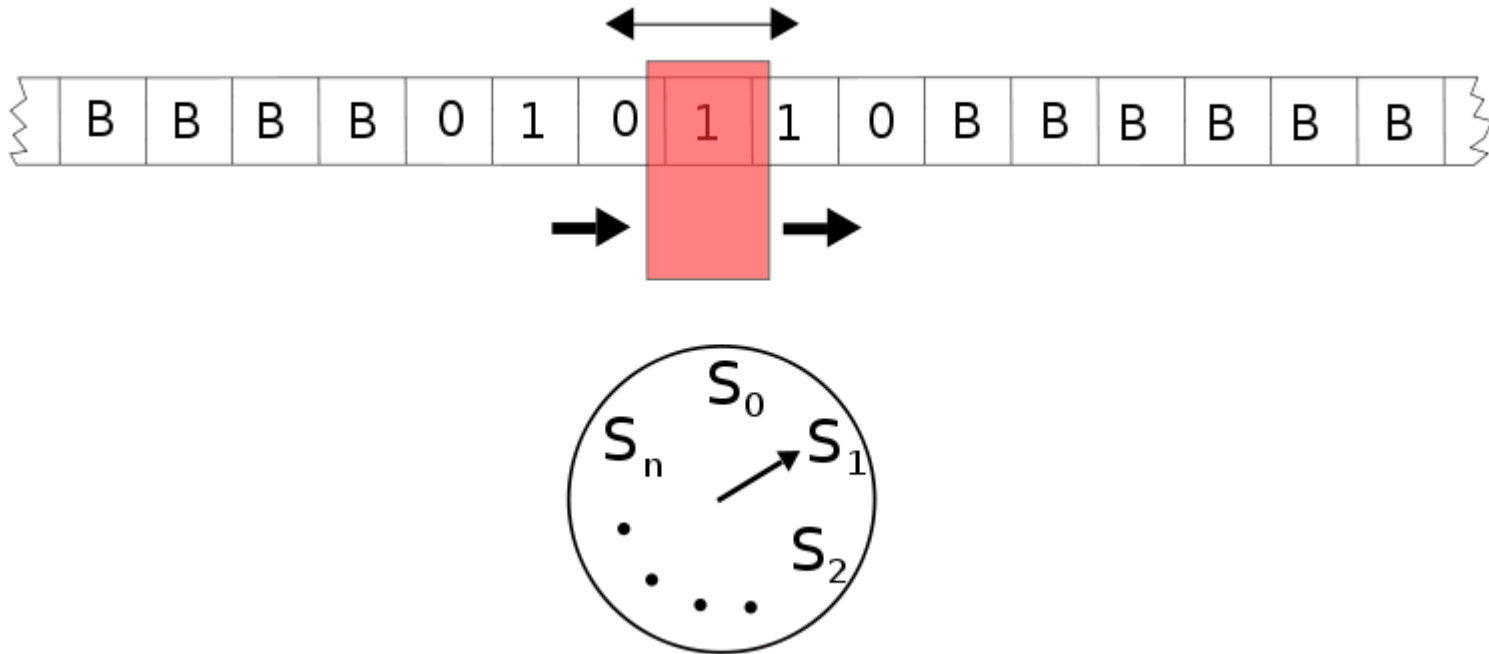
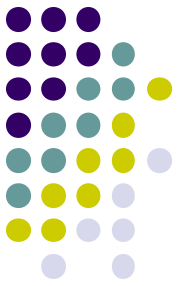
- Åpenbart at P ligger i NP
  - Vi trenger ikke å bli gitt en korrekt løsning, vi kan jo bare beregne den selv (i polynomisk tid)!
- Alle problemer i NP kan løses i eksponensiell tid via brute-force-søk
  - Generer alle mulige løsninger (som det finnes  $2^{O(n^k)}$  av) og verifiser dem (i polynomisk tid)
- Men kan vi klare bedre?
- Det vet vi ikke!



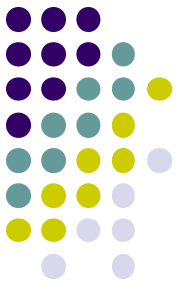
# P vs NP

- Spørsmålet om  $P=NP$  eller  $P \neq NP$  er det viktigste åpne spørsmålet innen Computer Science
- En løsning vil ha dyptgående og omfattende konsekvenser
- Hvis  $P=NP$ :
  - Matematiske beviser kan automatiseres (f.eks alle CMI-problemene)
  - Et hav av problemer lar seg løse effektivt
  - Kreativitet gjøres "overflødig"
  - All kryptografi må revurderes
- Hvis  $P \neq NP$ :
  - Omtrent som før

# Digresjon – Nondeterministic Turingmaskin



# Problemklassen NPC



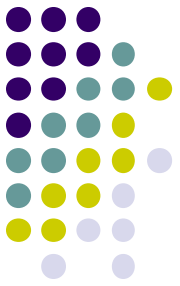
212902463182587575474  
978820162715174978067  
039632772162782333832  
153819499840564959113  
665738530219183167831  
073879953172308895692  
30873441936471

Transformer  
 $\leq_P$

Faktoriserings-  
problemet

CLIQUE-  
problemet

# Problemklassen NPC (NP-complete)

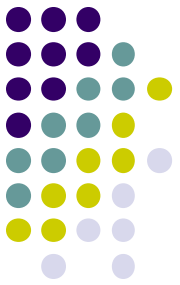


Et problem  $L$  er **NP-complete** hvis

1)  $L$  er i NP

2) Ethvert problem  $L'$  i NP kan transformeres til  $L$  i polynomisk tid (betegnet  $L' \leq_p L$ )

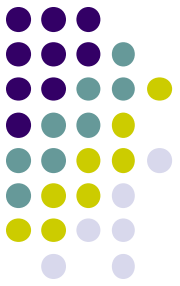
Mengden av alle NP-komplette problemer kalles **NPC**. Et problem kalles **NP-hard** om det oppfyller 2).



# Noen NPC-problemer

- SAT (Satisfiability)
- CIRCUIT-SAT
- 3-CNF-SAT
- SUBSET-SUM
- CLIQUE
- VERTEX-COVER
- HAM-CYCLE
- TSP (Travelling Salesman Problem)
- GRAPH K-COLORABILITY
- KNAPSACK
- 0-1 INTEGER-PROGRAMMING
- SUBGRAPH-ISOMORPHISM

# Slik tror vi det er...

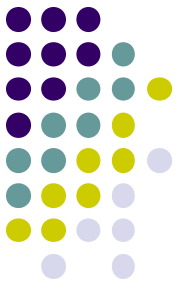


NP

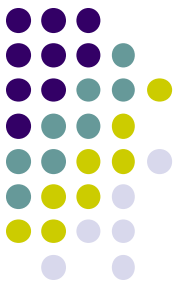
P

NPC

...eller er det slik?

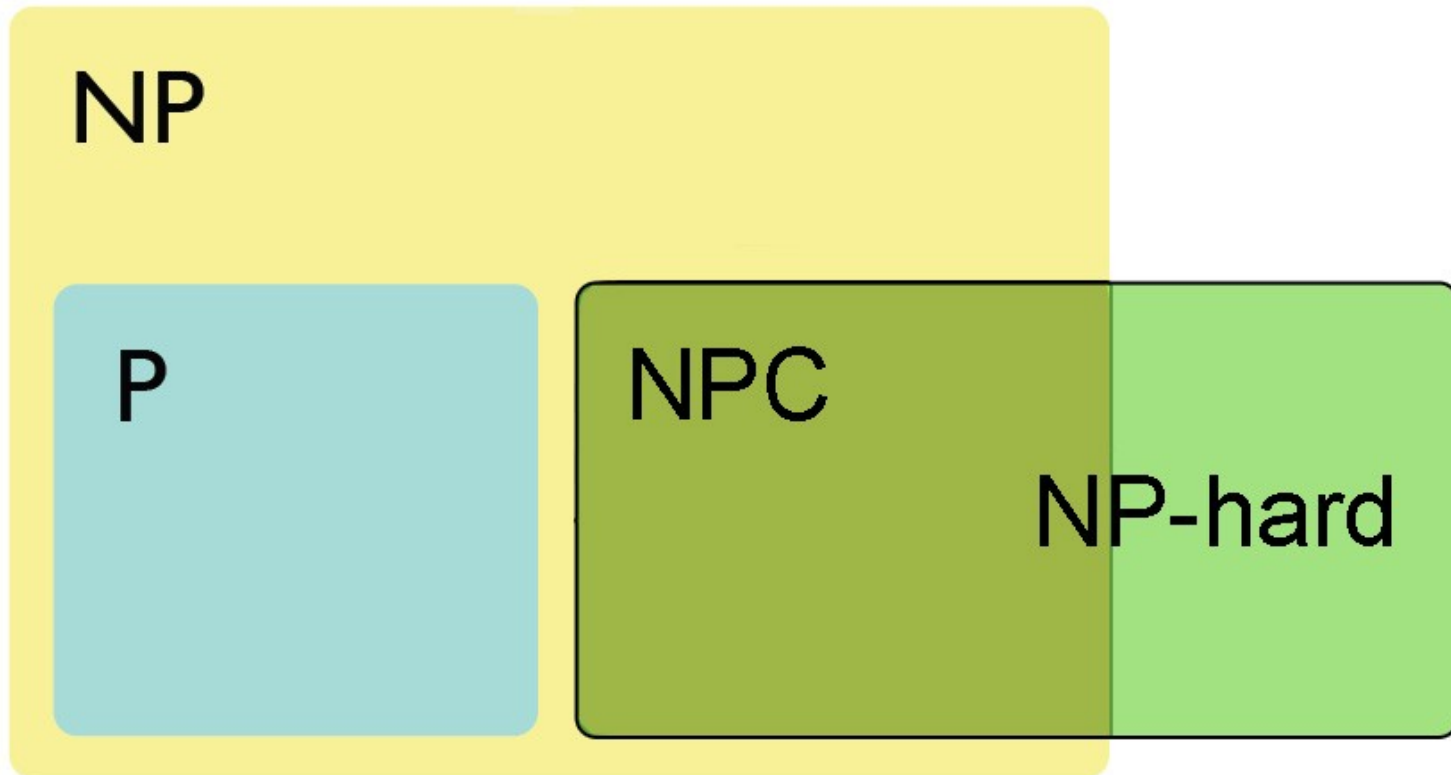


$P = NP = NPC$



# Fullstendig venndiagram

- Slik er det hvis  $P \neq NP$

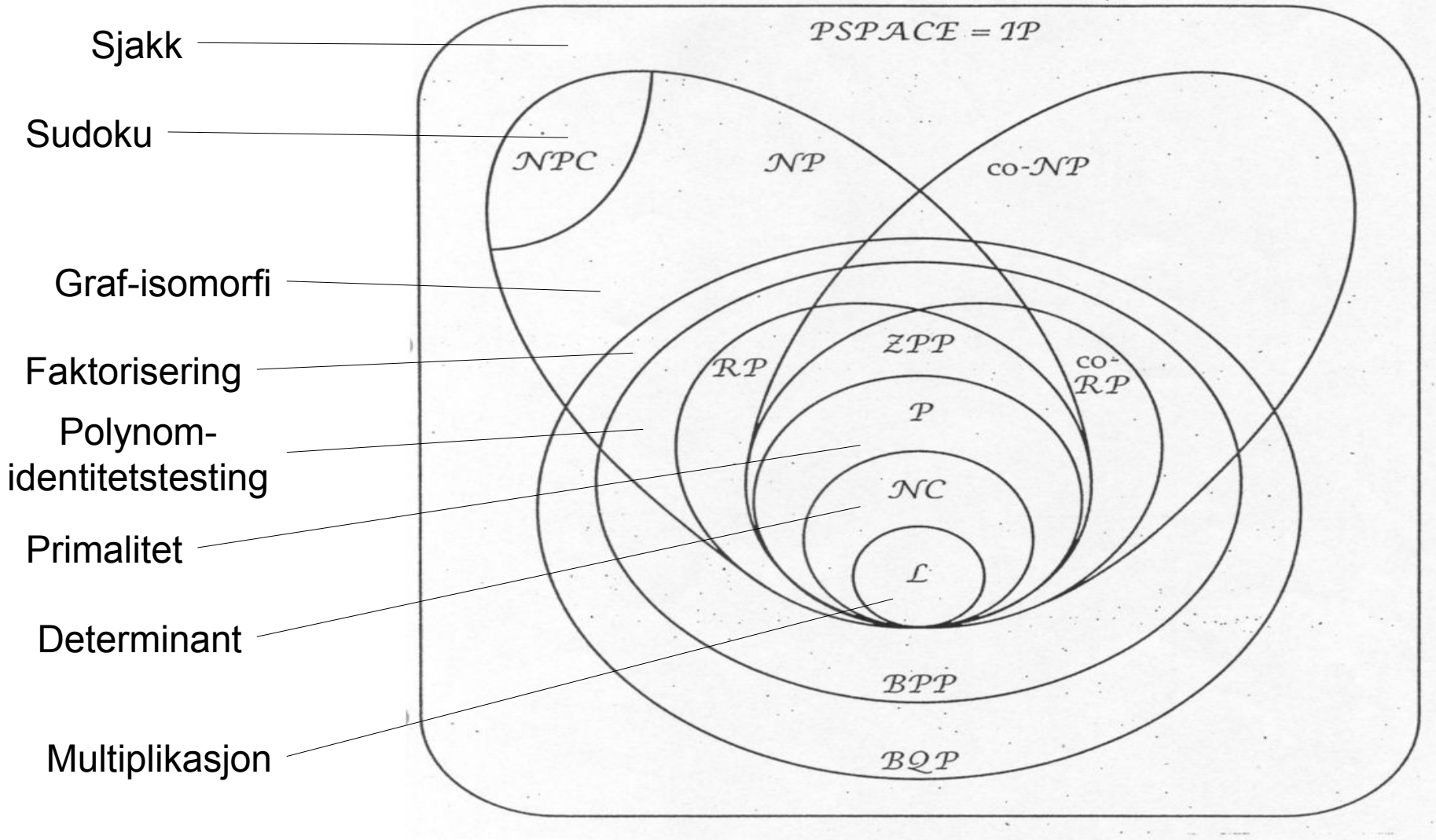
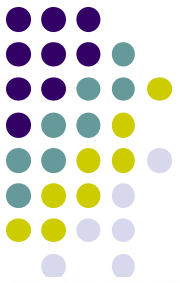


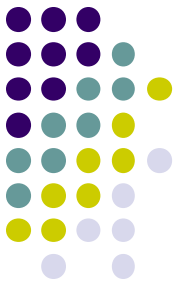
# Noen problemer i NP men IKKE i NPC



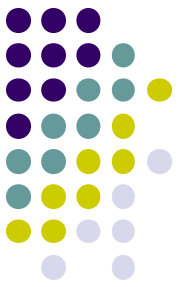
- Vi vet ikke om så mange naturlige problemer...
  - Faktorisering
  - Grafisomorfi
  - Stokastiske spill
  - Diskrete logaritmer
- ...men hvis  $P \neq NP$  sier Ladners teorem (1971) at det finnes uendelig mange av dem

# Flere problemklasser





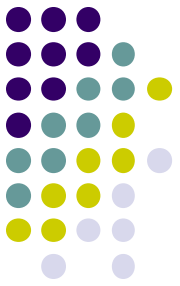
# Problemtransformasjoner



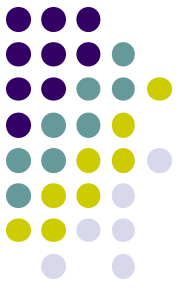
# Nytt ukjent problem

- Du kommer over et nytt problem du ønsker å løse **effektivt**
- Hvordan skal man gå frem for å finne en løsning av problemet?
  1. Løs problemet med en kjent algoritme
  2. Hvis ingen kjent algoritme finnes; se om problemet likner et problem man har algoritme for
  3. Bevis at det ikke finnes noen effektiv løsning av problemet

# Løse problemet med kjent algoritme



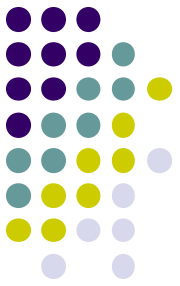
- Hvis det nye problemet kan løses med en kjent algoritme, er problemet enkelt/trivielt
- Men vær oppmerksom på hvilke krav de ulike algoritmene har for input



# Transformere problemet

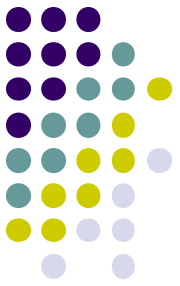
- Vi ønsker å gjøre om det nye problemet vårt til et problem som vi har en kjent algoritme for
- Ønsker å redusere ukjent problem til kjent problem i polynomisk tid
- Løser så det enkle problemet, og vi har et svar på det ukjente problemet

# Bevise at ukjent problem er vanskelig



- Vi klarer ikke å finne noen reduksjon til en kjent algoritme
- Prøv å bevise at det ikke eksisterer en effektiv løsning for vårt nye ukjente problem

# Problemklassen NPC (NP-complete)



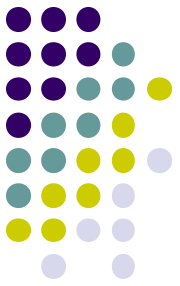
Et problem  $L$  er **NP-complete** hvis

1)  $L$  er i NP

2) Ethvert problem  $L'$  i NP kan transformeres til  $L$  i polynomisk tid (betegnet  $L' \leq_p L$ )

Mengden av alle NP-komplette problemer kalles **NPC**. Et problem kalles **NP-hard** om det oppfyller 2).

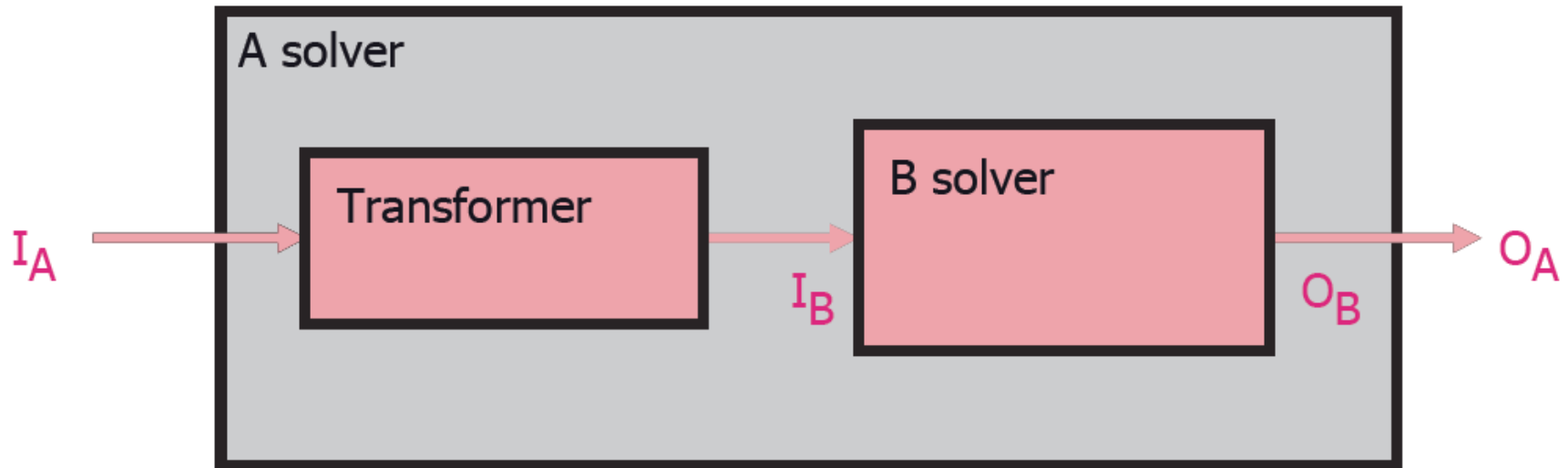
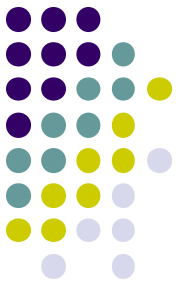
# Cook's teorem (1971)



**Teorem:** SAT er NP-komplett

**Korollar:**  $P = NP \Leftrightarrow SAT \in P$

# Reduksjon med kjent algoritme



Transformerer det nye vanskelige problemet A ( $I_A$ ), ned til en algoritme som løser et kjent problem B ( $I_B$ ), løser problemet B ( $O_B$ ), og vi har fått en løsning til vårt problem A ( $O_A$ )

# Karp (1972): Reducibility Among Combinatorial Problems

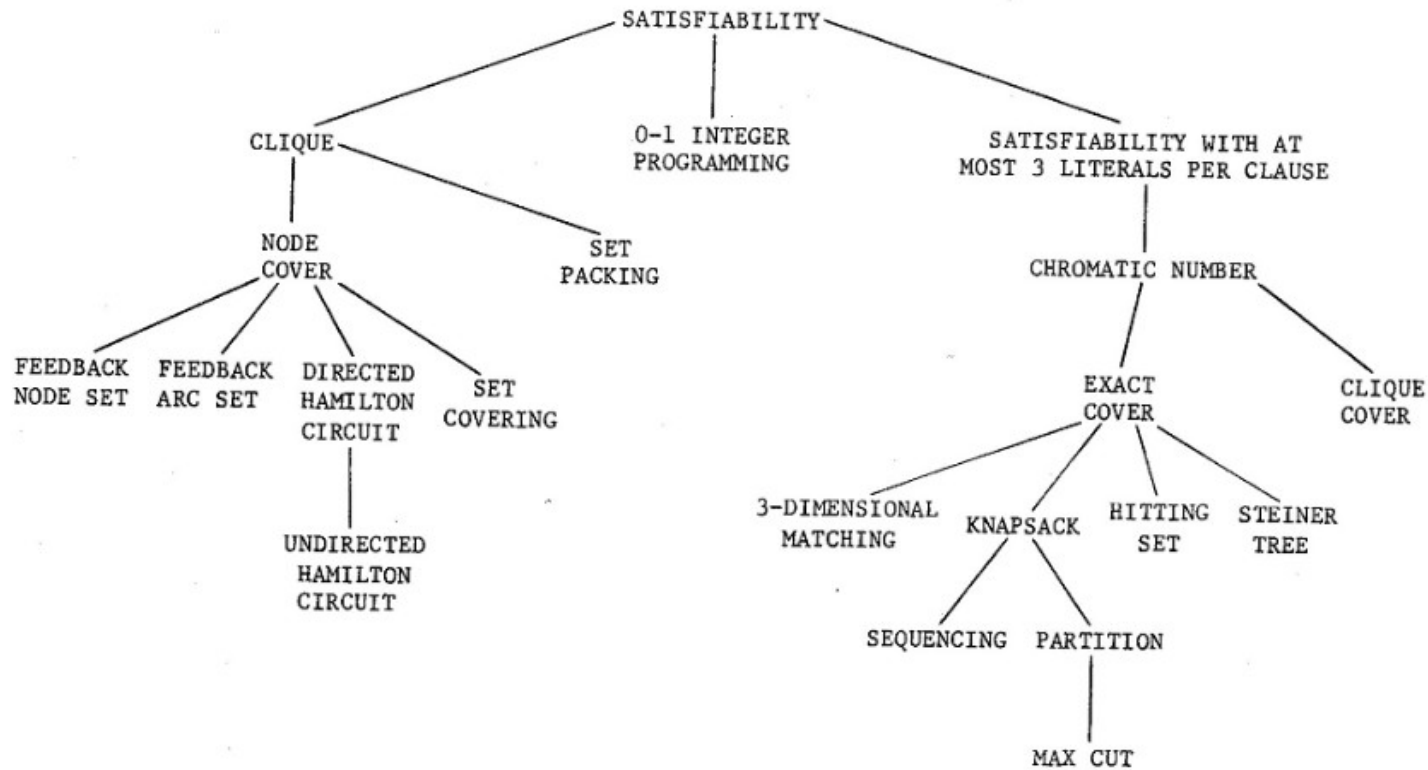
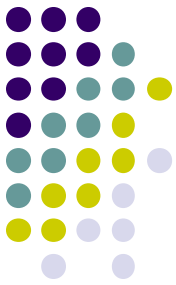
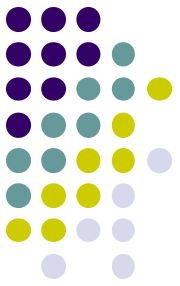


FIGURE 1 - Complete Problems

# Transformasjon



*ukjent problem*

**A**



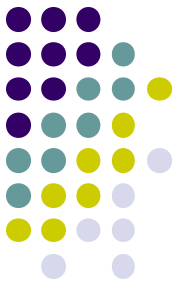
*kjent problem*

**B**

**A**  $\leq_P$  **B**

Hvis ukjent problem A kan reduseres effektivt og enkelt til kjent problem B, kan vi bare løse B, og det ukjente problemet A vil være minst like enkelt som B

# Reduksjon



*kjent problem*

**A**

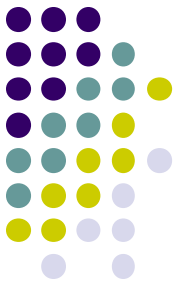


*ukjent problem*

**B**

**A**  $\leq_P$  **B**

Vi vet at A er vanskelig å løse, reduserer så A til vårt ukjente problem B, har da vist at vårt problem B er minst like vanskelig som A



# Vise at ukjent problem er i P

*ukjent problem*

A



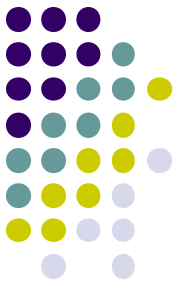
B

*kjent problem i P*

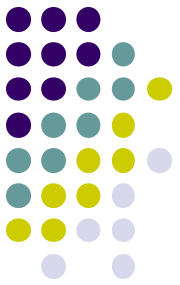
$A \leq_P B$

Hvis ukjent problem A kan reduseres effektivt og enkelt til kjent problem B (B er i mengden P), kan vi bare løse B, og det ukjente problemet A vil være minst like enkelt som B

# Vise at ukjent problem er i NP



- Enten: Finn en algoritme som verifiserer løsningen på problemet i polynomisk tid.
- Eller: Vis at problemet er i P
  - Siden P er en undermengde av NP



# Redusere NP til NPC

Alt i

kan reduseres til..

alt i

NP



NPC

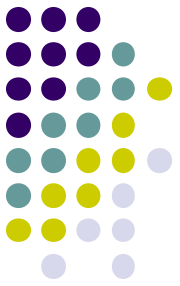
NP

$\leq_P$

NPC

Hvis man har løst ett av NPC-problemene i polynomisk tid, har man løst alle problemer i NP i polynomisk tid (fordi problemene kan reduseres til hverandre)

# Bevise at ukjent problem A er vanskelig



*kjent problem*

**NPC**

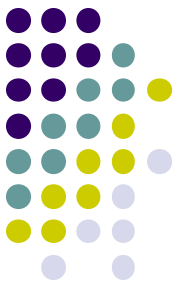


**A**

*ukjent problem*

**NPC  $\leq_P$  A**

Hvis man effektivt kan redusere et NPC-problem til vårt ukjente problem A, har vi vist at A må være minst like vanskelig som NPC-problemet



# Redusere NP til NP-hard

Alt i

kan reduseres til..

alt i

**NPC**



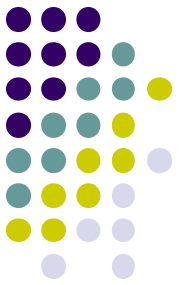
**NP-hard**

**NPC**

$\leq_P$

**NP-hard**

Et NP-hard problem er minst like vanskelig som et NPC-problem, hvis reduseringen er i polynomisk tid



# Bevise at ukjent problem A er NP-hard

*kjent problem*

NP-hard

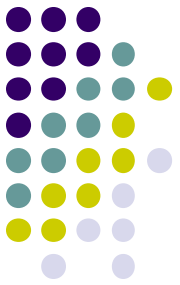


A

*ukjent problem*

$$\text{NP-hard} \leq_P A$$

Hvis man effektivt kan redusere et NP-hard problem til vårt ukjente problem A, har vi vist at A må være minst like vanskelig som NP-hard problemet

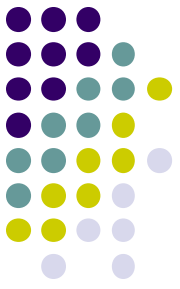


Over til noe mer konkret...

# Bevise at vårt problem er vanskelig



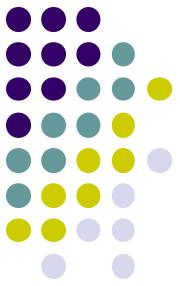
- For å vise at vårt problem er vanskelig, er det lurt å kunne noen NPC-problemer, så vi kan redusere en av dem til vårt problem
- Skal nå gå igjennom en god del problemer og vise hvordan noen av disse kan bevises å ligge i NPC



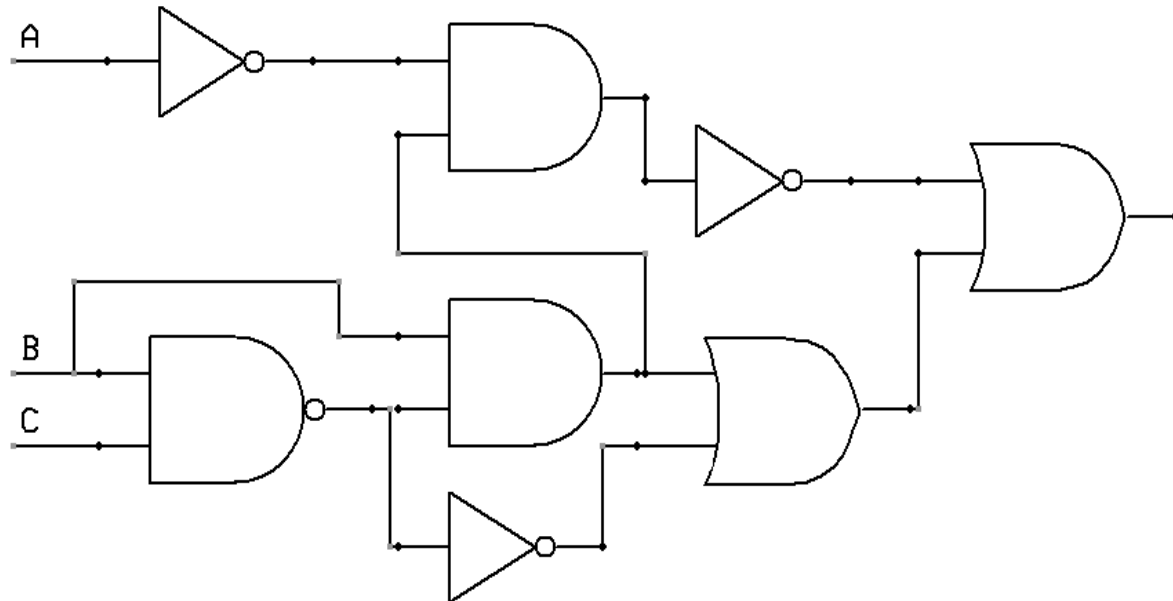
# SATISFIABILITY (SAT)

- Vi har et boolsk uttrykk med  $n$  inputs.
- Kan vi finne en tilordning (0/1) av inputs som gjør at output blir 1 (true)?
- Eks:
  - 1)  $\phi_1 = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge ((x_1 \rightarrow x_2) \vee x_4)$
  - 2)  $\phi_2 = (\neg x_1 \wedge \neg x_2) \wedge (x_1 \vee x_2)$ 
    - 1) Ja, f.eks:  $x_1=1, x_2=1, x_3, x_4$  whatever!
    - 2) Ingen slik ordning finnes.

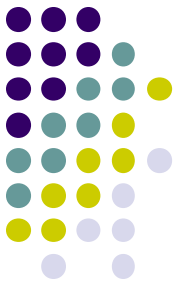
# CIRCUIT-SAT



- Likt som SAT men konkretisert som logiske kretser



- Løses ved å sette  $A=1$



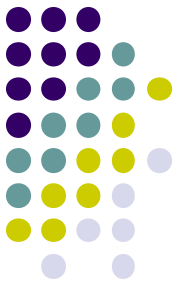
# 3-CNF-SAT

- Likt som SAT, men her må uttrykket være på **konjunktiv normalform**, og alle ledd har 3 variabler

- Eks:

$$(x_1 \vee \neg x_1 \vee \neg x_2) \wedge (x_2 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_3 \vee \neg x_4)$$

- Finn en tilordning av  $x_1$ ,  $x_2$ ,  $x_3$  og  $x_4$  slik at uttrykket blir 1
  - $x_2 = 1$  og  $x_3 = 0$  en mulig løsning



# SUBSET-SUM

- Vi har en sekvens av tall, og vil finne ut om noen av disse tallene kan legges sammen til en viss sum
- Eks:

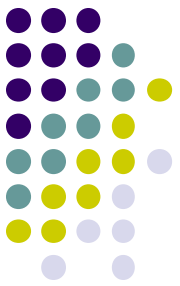
$\{8, 16, 4, 4, 1, 3\}$

1) Finner vi summen 18 her?

2) Finner vi summen 19 her?

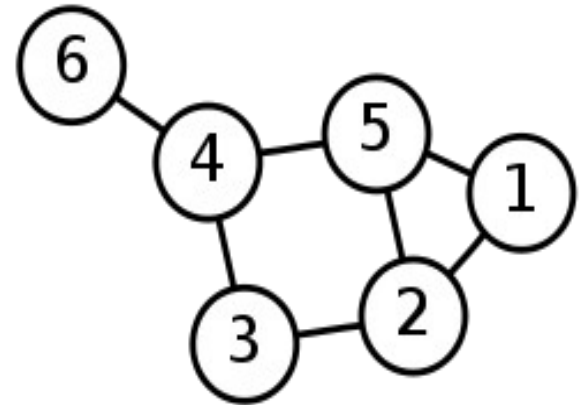
1) Nei

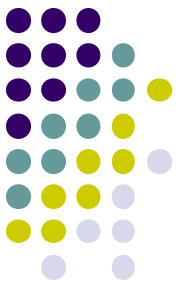
2) Ja, f.eks.  $8+4+4+3$



# CLIQUE

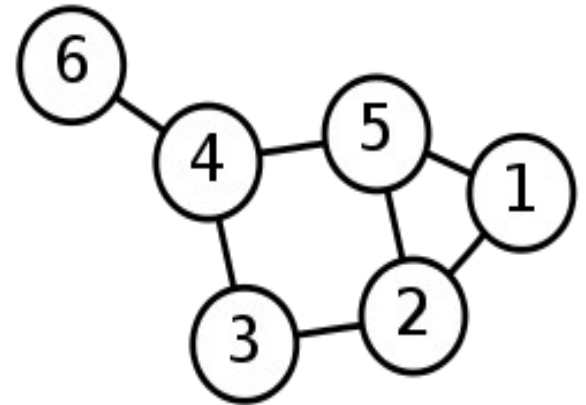
- En klikk er en samling noder i en graf som utgjør en *komplett* subgraf.
- Finnes det en klikk i denne grafen, med minst  $k$  noder?
- Eks:
  - 1) Hvis  $k=3$ ?
  - 2) Hvis  $k = 4$ ?
    - 1) Ja, nodene 1, 2 og 5
    - 2) Nei, det finnes ikke

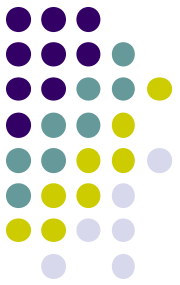




# VERTEX-COVER

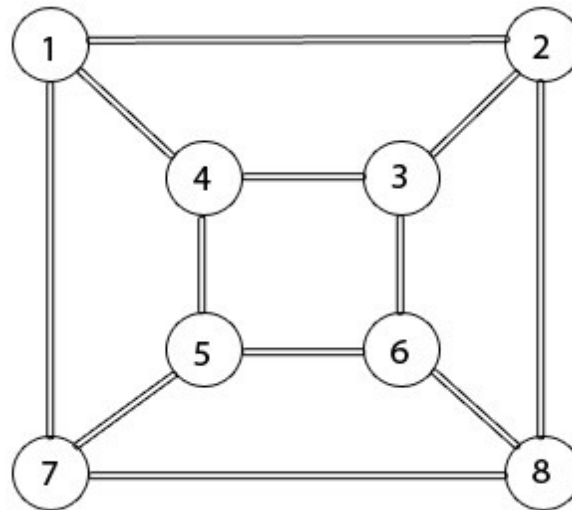
- Kan vi finne en samling av maksimalt  $k$  noder, som gjør at alle kantene er tilknyttet minst én av disse nodene?
- Eks:
  - 1) Hvis  $k=3$
  - 2) Hvis  $k=2$ 
    - 1) Ja, f.eks. Nodene 4, 5 og 2
    - 2) Nei

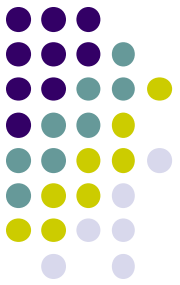




# HAM-CYCLE

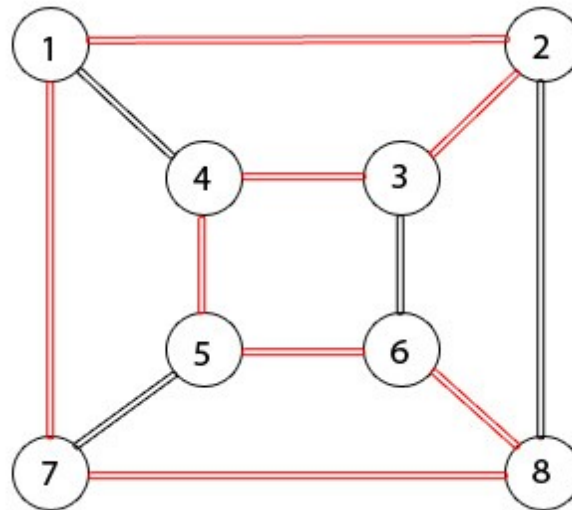
- En **Hamilton sykel** i en urettet graf  $G$ , er en simpel sykel som inneholder alle noder i  $G$
- Finnes det en Hamilton sykel i grafen?

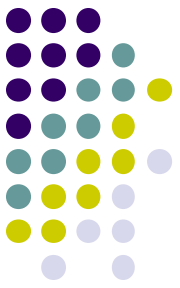




# HAM-CYCLE

- En **Hamilton sykel** i en urettet graf  $G$ , er en simpel sykel som inneholder alle noder i  $G$
- Finnes det en Hamilton sykel i grafen?





# TSP (Travelling Salesman Problem)

- Gitt en urettet graf med noder og vektete kanter, finn billigste sykel som besøker alle nodene én og bare én gang

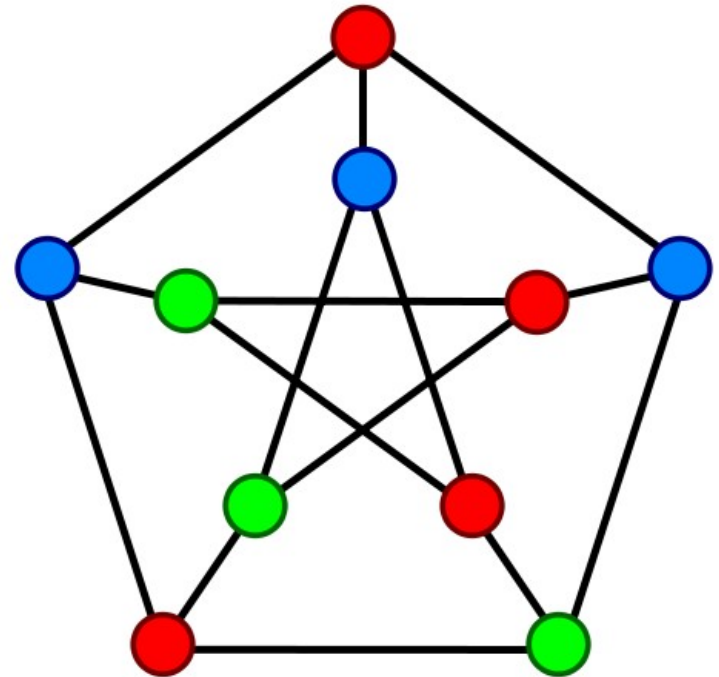
- Eks.
  - den korteste veien gjennom de 15 største byene i Tyskland



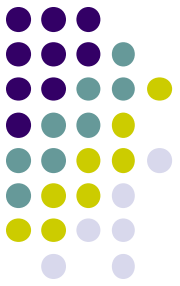
# GRAPH K-COLORABILITY



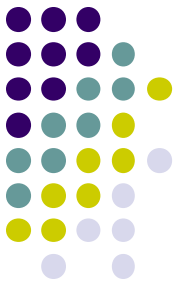
- Gitt  $k$  forskjellige farger, kan vi fargelegge nodene i en graf slik at ingen naboer har samme farge?
- Eks:
  - 1)  $k = 3$
  - 2)  $k = 2$ 
    - 1) Ja, det kan den
    - 2) Nei, det kan den ikke
      - Obs!  $k=2$  kan sjekkes i polynomisk tid!



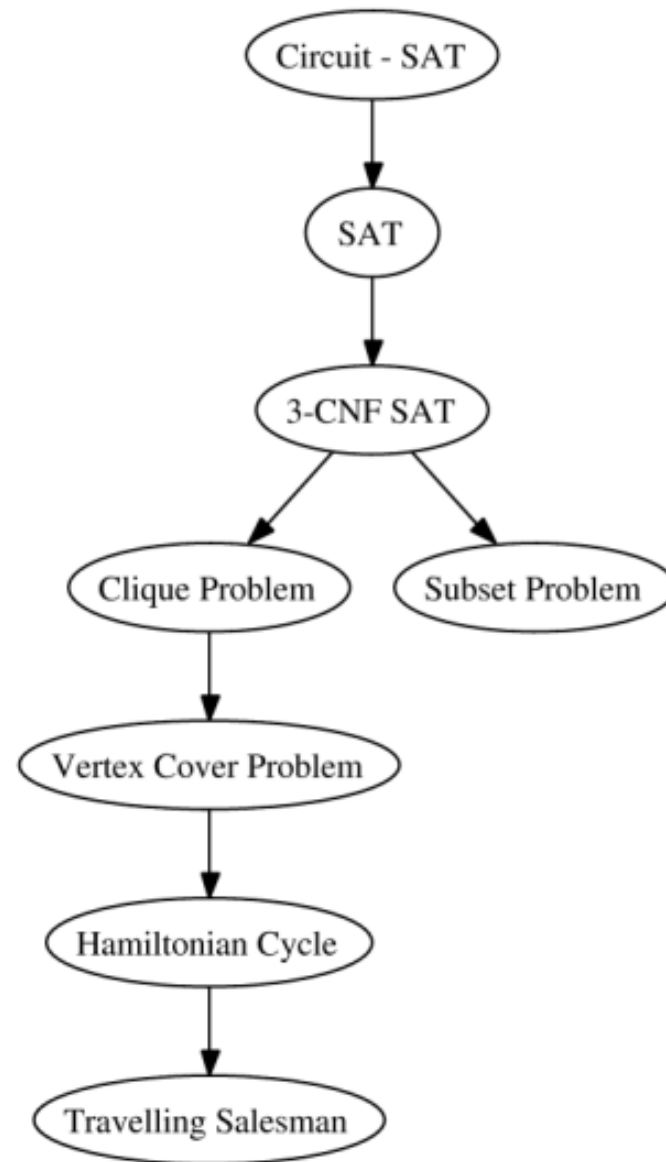




# Noen reduksjoner



Hvordan NPC-problemer kan reduceres til hverandre:



# 3-CNF-SAT $\leq_p$ CLIQUE



- Hva må  $x_1$ ,  $x_2$  og  $x_3$  være for å få output 1?

$$\phi = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$$

$$C_1 = x_1 \vee \neg x_2 \vee \neg x_3$$

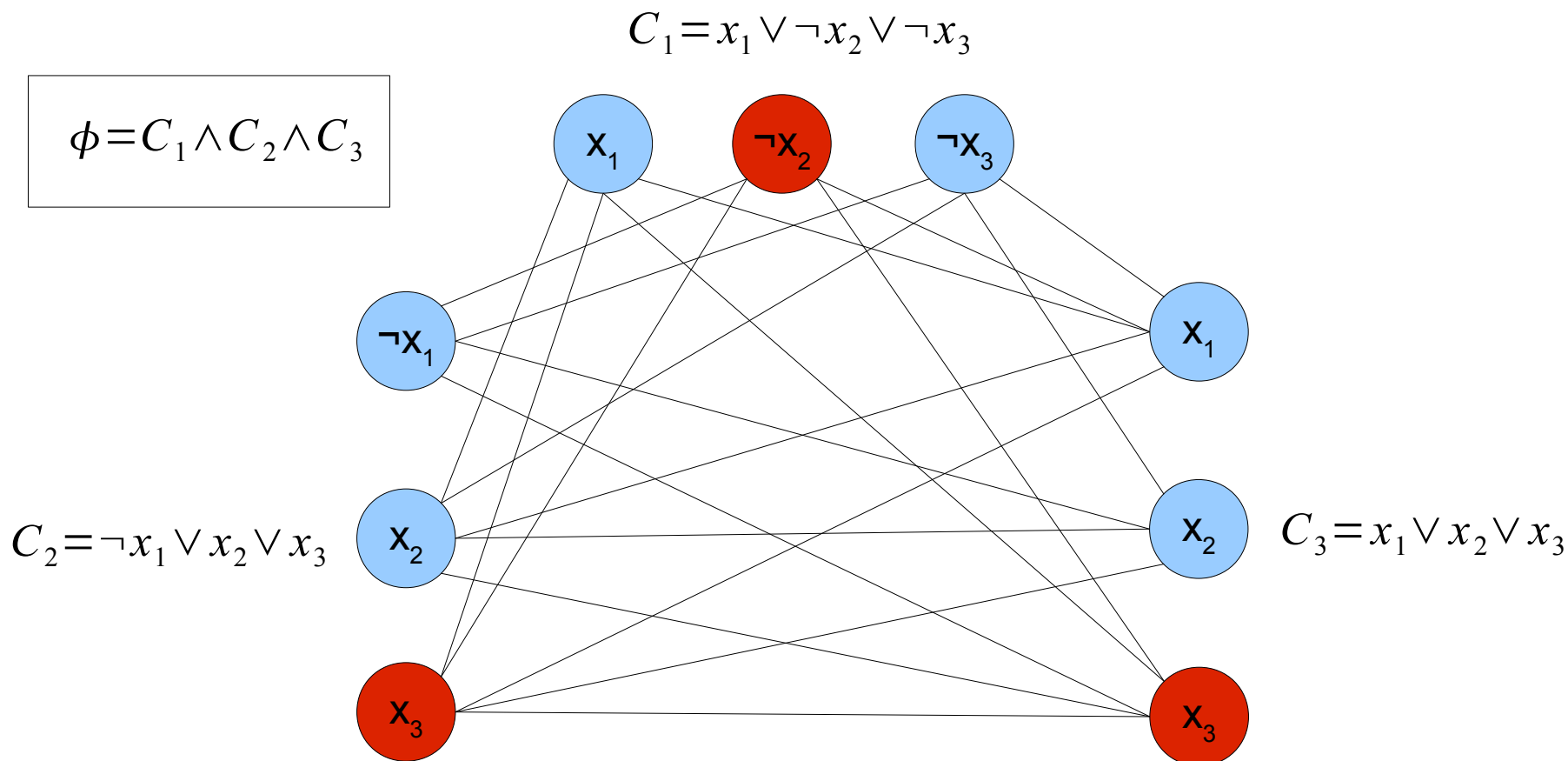
$$C_2 = \neg x_1 \vee x_2 \vee x_3$$

$$C_3 = x_1 \vee x_2 \vee x_3$$

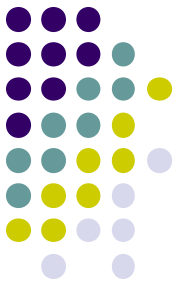
# 3-CNF-SAT $\leq_p$ CLIQUE



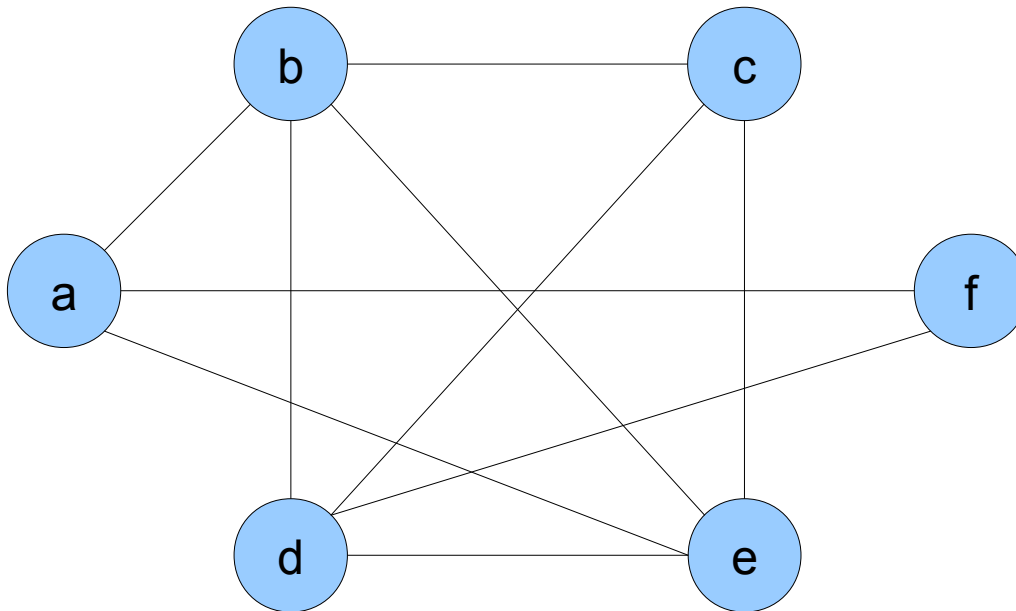
- ...det er jo bare å finne en klikk av størrelse 3 i følgende graf:



# CLIQUE $\leq_p$ VERTEX-COVER



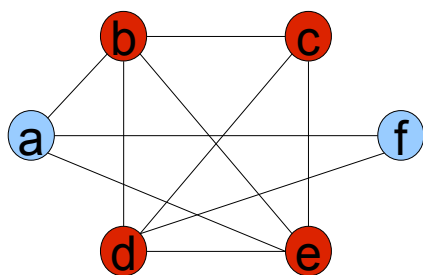
- Klarer du å finne en klikk av størrelse 4 i grafen  $G$ ?



# CLIQUE $\leq_p$ VERTEX-COVER



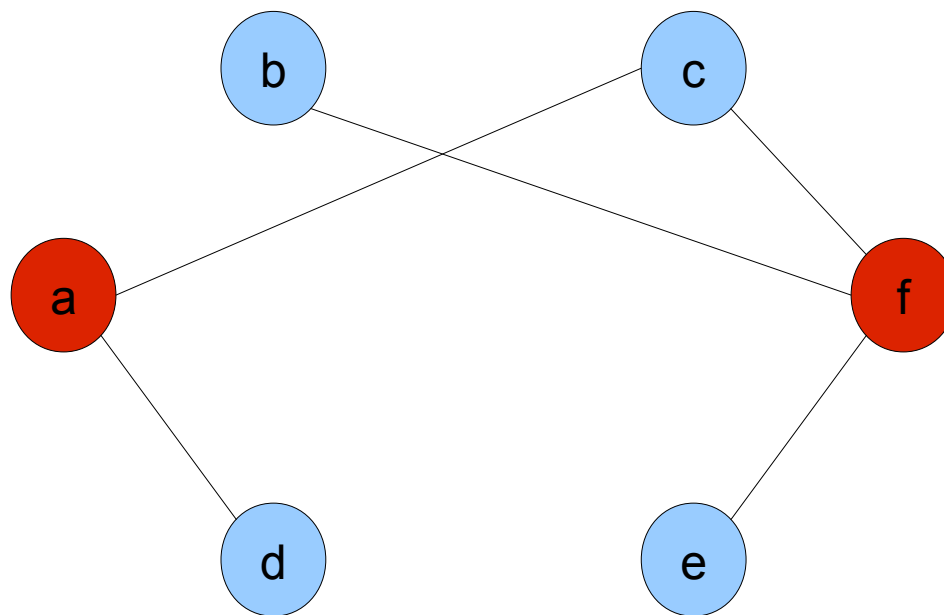
- ...det er jo bare å finne et node-dekke i komplementærgrafen  $\bar{G}$



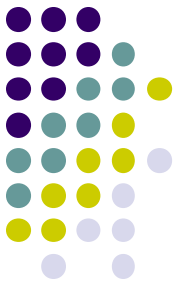
G har en klikk av  
størrelse k



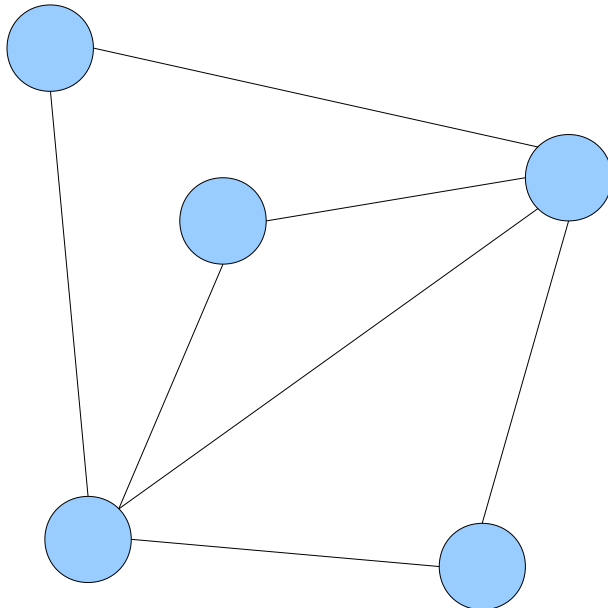
$\bar{G}$  har et nodedekke  
av størrelse  $|V| - k$



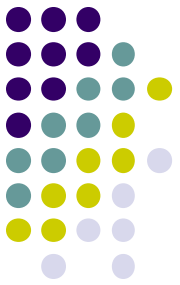
# HAM-CYCLE $\leq_p$ TSP



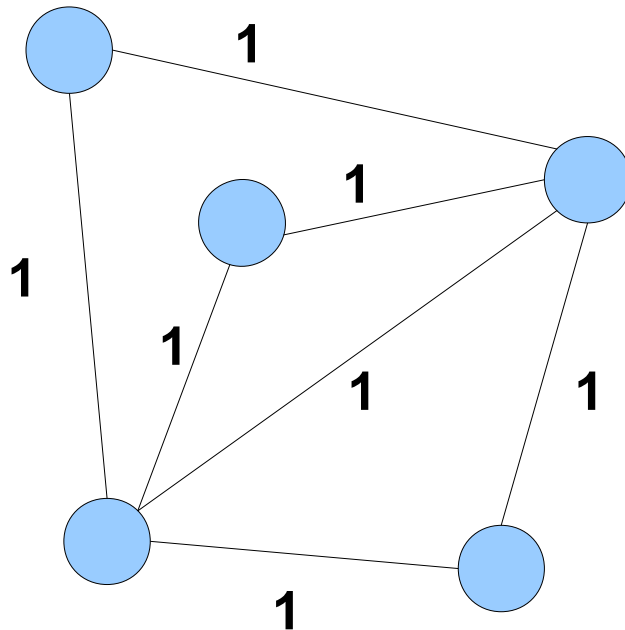
- Klarer du å finne en Hamilton sykel?



# HAM-CYCLE $\leq_p$ TSP



- ...det er jo bare å løse følgende TSP:

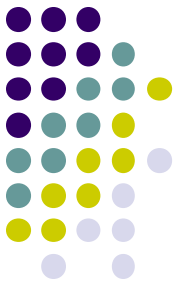


Budsjett = 5

Grafen har en Hamilton sykel

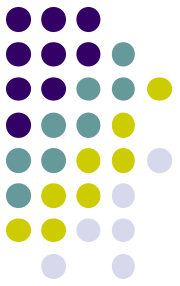
↔

det finnes en TS-sykel av  
kostnad nøyaktig 5



Noen tidligere eksamensoppgaver

# Eksamen desember 2009



## Oppgave 1

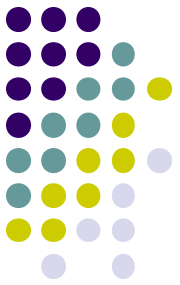
e) Om du står overfor et ukjent problem  $A$  i NP, hvordan vil du vise at det er NP-komplett?

Reduser kjent problem i NPC til  $A$  i polynomisk tid

f) Er det mulig å bevise at noen NP-komplette problemer kun kan løses av algoritmer med eksponentiell kjøretid uten at man avgjør hvorvidt de andre kan løses i polynomisk tid? Begrunn svaret kort.

Nei, for hvis et annet problem i NPC kunne løses i polynomisk tid kunne vi bare redusert eksponentiell-problemet til dette og løst det i polynomisk tid i stedet. Selvmotsigelse.

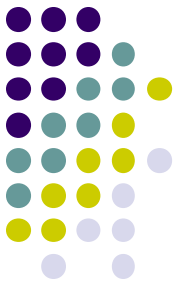
# Eksamen desember 2006



## Oppgave 5

b) Du har observert at dataspillet Slurm Invaders ligner svært på det NP-komplette problemet VERTEX-COVER, og du akter å bruke denne likheten til å vise at Slurm Invaders er NP-komplett. Hvordan vil du gå frem? Du kan anta at du alt har vist at Slurm Invaders er i mengden NP. (Skriv kort.)

Reduser VERTEX-COVER til Slurm Invaders i polynomisk tid. Dvs transformer enhver probleminstans i VERTEX-COVER til en probleminstans i Slurm Invaders.



## Oppgave 1

Du står overfor de tre problemene A, B og C. Alle tre befinner seg i mengden NP. Du vet at A er i mengden P og at B er i mengden NPC. Anta at du skal bruke polynomiske reduksjoner mellom disse problemene til å vise ulike egenskaper.

Merk: Enkelte av egenskapene kan selvfølgelig vises på andre måter. Du kan se bort fra det i denne oppgaven.

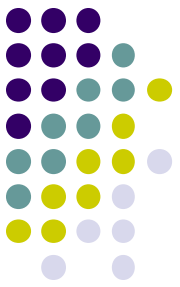
f) Fullfør følgende utsagn

For å bevise at C er i P må C reduseres til A i polynomisk tid.

For å bevise at C er i NPC må B reduseres til C i polynomisk tid.

Hvis B kan reduseres til A i polynomisk tid, så er  $P = NP$ .

# Eksamen august 2005



## Oppgave 2

Det er NP-komplett å finne den lengste stien i en graf. Anta at nodene i grafen  $G = (E, V)$  er sanger og at kanten  $(u, v)$  finnes i  $E$  dersom sang  $v$  er en nyinnspilling (“cover”) av sang  $u$ . Anta at nodene i grafen  $G' = (E', V')$  er artister og at kanten  $(u', v')$  finnes i  $E'$  dersom artist  $v'$  har gjort en nyinnspilling av en av artist  $u'$  sine sanger. Anta at du kan ha vilkårlig mange sanger og artister, og at man kan ha nyinnspillinger av en nyinnspillinger.

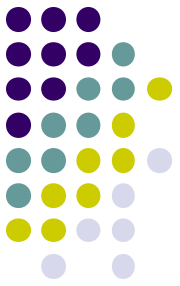
**a)** Enten (1) argumenter for at det er NP-komplett å finne den lengste stien i  $G$ , eller (2) beskriv kort en effektiv algoritme som finner den lengste stien i  $G$  (angi kjøretid).

**Dette blir en DAG hvor hver node bare har én forgjenger. Bruk BFS eller DFS på skogen. Kjøretid:  $\theta(V+E)$**

**b)** Enten (1) argumenter for at det er NP-komplett å finne den lengste stien i  $G'$ , eller (2) beskriv kort en effektiv algoritme som finner den lengste stien i  $G'$  (angi kjøretid).

**Her kan sykler oppstå så problemet er ekvivalent med å finne den lengste stien i en vilkårlig graf. Problemet er derfor NP-komplett.**

# Eksamen mai 2006



## Oppgave 1

d) Anta at du har to NP-komplette problemer A og B. Anta også at du har et problem C fra mengden P.

1. Hvis A befinner seg i P vil B også gjøre det.

Ja Nei

2. Hvis du kan redusere problemet C til problemet A i polynomisk tid så følger det at C er NP-komplett.

3. Hvis  $P = NP$  vil det likevel finnes problemer i NP som ikke er NP-komplette.

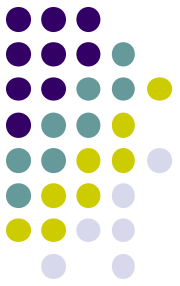
4. Hvis  $P \neq NP$  så er NP-komplette problemer ikke beregnbare.

Ja Nei

5. 1-0-ryggsekkproblemet (kjøretid  $O(nW)$ ) er NP-komplett.

Ja Nei

# Eksamen mai 2006



## Oppgave 1

d) Anta at du har to NP-komplette problemer A og B. Anta også at du har et problem C fra mengden P.

1. Hvis A befinner seg i P vil B også gjøre det.

Ja Nei

2. Hvis du kan redusere problemet C til problemet A i polynomisk tid så følger det at C er NP-komplett.

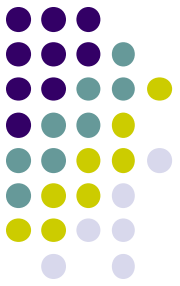
3. Hvis  $P = NP$  vil det likevel finnes problemer i NP som ikke er NP-komplette.

4. Hvis  $P \neq NP$  så er NP-komplette problemer ikke beregnbare.

5. 1-0-ryggsekkproblemet (kjøretid  $O(nW)$ ) er NP-komplett.

Ja Nei

# Eksamen mai 2006



## Oppgave 1

d) Anta at du har to NP-komplette problemer A og B. Anta også at du har et problem C fra mengden P.

[ ] 1. Hvis A befinner seg i P vil B også gjøre det.

Ja Nei

[ ]  2. Hvis du kan redusere problemet C til problemet A i polynomisk tid så følger det at C er NP-komplett.

[ ] [ ] 3. Hvis  $P = NP$  vil det likevel finnes problemer i NP som ikke er NP-komplette.

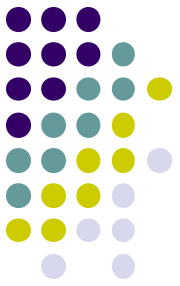
[ ] [ ] 4. Hvis  $P \neq NP$  så er NP-komplette problemer ikke beregnbare.

Ja Nei

[ ] [ ] 5. 1-0-ryggsekkproblemet (kjøretid  $O(nW)$ ) er NP-komplett.

Ja Nei

# Eksamen mai 2006



## Oppgave 1

d) Anta at du har to NP-komplette problemer A og B. Anta også at du har et problem C fra mengden P.

[ ] 1. Hvis A befinner seg i P vil B også gjøre det.

Ja Nei

[ ]  2. Hvis du kan redusere problemet C til problemet A i polynomisk tid så følger det at C er NP-komplett.

[ ]  3. Hvis  $P = NP$  vil det likevel finnes problemer i NP som ikke er NP-komplette.

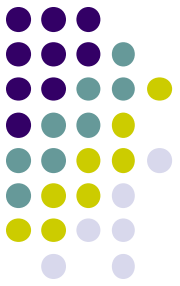
[ ] [ ] 4. Hvis  $P \neq NP$  så er NP-komplette problemer ikke beregnbare.

Ja Nei

[ ] [ ] 5. 1-0-ryggsekkproblemet (kjøretid  $O(nW)$ ) er NP-komplett.

Ja Nei

# Eksamen mai 2006



## Oppgave 1

d) Anta at du har to NP-komplette problemer A og B. Anta også at du har et problem C fra mengden P.

[ ] 1. Hvis A befinner seg i P vil B også gjøre det.

Ja Nei

[ ]  2. Hvis du kan redusere problemet C til problemet A i polynomisk tid så følger det at C er NP-komplett.

[ ]  3. Hvis  $P = NP$  vil det likevel finnes problemer i NP som ikke er NP-komplette.

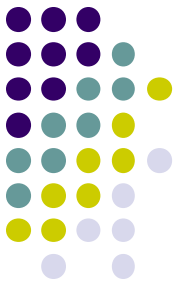
[ ]  4. Hvis  $P \neq NP$  så er NP-komplette problemer ikke beregnbare.

Ja Nei

[ ] [ ] 5. 1-0-ryggsekkproblemet (kjøretid  $O(nW)$ ) er NP-komplett.

Ja Nei

# Eksamen mai 2006



## Oppgave 1

d) Anta at du har to NP-komplette problemer A og B. Anta også at du har et problem C fra mengden P.

[ ] 1. Hvis A befinner seg i P vil B også gjøre det.

Ja Nei

[ ]  2. Hvis du kan redusere problemet C til problemet A i polynomisk tid så følger det at C er NP-komplett.

[ ]  3. Hvis  $P = NP$  vil det likevel finnes problemer i NP som ikke er NP-komplette.

[ ]  4. Hvis  $P \neq NP$  så er NP-komplette problemer ikke beregnbare.

Ja Nei

[ ] 5. 1-0-ryggsekkproblemet (kjøretid  $O(nW)$ ) er NP-komplett.

Ja Nei