I. Grundläggande metoder för algoritmkonstruktion.

II. NP-fullständighet.

III. Inexakta metoder.

Fö. 6: Komplexitetsteori och NP-fullständighet I.

Fö. 7: Komplexitetsteori och NP-fullständighet II.

Fö. 8: Algoritmer för NP-fullständiga problem.

Komplexitetsteori och NP-fullständighet.

Fö. 6+7

- Grundläggande begrepp.
- Översikt över komplexitetsteori.
- Polynomiska reduktioner.
- NP-fullständighet.

Komplexitetsteori och NP-fullständighet.

Syfte: Demonstrera en klass (intressanta) problem som (troligen) inte kan lösas i polynomisk tid.

Grundläggande begrepp.

Grundläggande begrepp.

Vad är ett (beräknings)problem?

Grundläggande begrepp.

Vad är ett (beräknings)problem?

En beskrivning av relationen mellan in- och utdata.

Grundläggande begrepp.

Vad är ett (beräknings)problem?

En beskrivning av relationen mellan in- och utdata.

Indata. En graf $G = (V, E)$ och två noder $s, t \in V$.
Utdata. Hur långt är det kortaste avståndet mellan s och t i G.

Grundläggande begrepp.

Vi studerar ofta *beslutsproblem* där de enda svaren är "ja" och "nej".

Indata. En graf $G = (V, E)$, två noder $s, t \in V$, heltal $K \geq 0$.
Utdata. Är avståndet mellan s och t i G kortare än K?

Grundläggande begrepp.

Vad är en algoritm?

Grundläggande begrepp.

Vad är en algoritm?

Ändlig uppsättning instruktioner som efter exekvering löser ett problem. Algoritmen måste alltid producera ett korrekt svar inom ett ändligt antal steg.

Grundläggande begrepp.

Vad är en algoritm?

Ändlig uppsättning instruktioner som efter exekvering löser ett problem. Algoritmen måste alltid producera ett korrekt svar inom ett ändligt antal steg.

Instruktionerna måste vara "enkla" $\rightarrow$ Turingmaskiner etc.

Grundläggande begrepp.

Alla problem har inte en algoritm.

Grundläggande begrepp.

Alla problem har inte en algoritm.

Sådana kallas *oavgörbara*.

Grundläggande begrepp.

Alla problem har inte en algoritm.

Sådana kallas *oavgörbara*.

Konkret exempel: Halt-problemet.

Grundläggande begrepp.

Ett beslutsproblem kan ses som en funktion från ändliga teckensträngar till $\{0, 1\}$.

Grundläggande begrepp.

Ett beslutsproblem kan ses som en funktion från ändliga teckensträngar till $\{0, 1\}$.

Det finns ω ändliga teckensträngar.

Grundläggande begrepp.

Ett beslutsproblem kan ses som en funktion från ändliga teckensträngar till $\{0, 1\}$.

Det finns ω ändliga teckensträngar.

Det finns 2^{ω} beslutsproblem.

Grundläggande begrepp.

Ett beslutsproblem kan ses som en funktion från ändliga teckensträngar till $\{0, 1\}$.

Det finns ω ändliga teckensträngar.

Det finns 2^{ω} beslutsproblem.

En algoritm har en ändlig beskrivning. Det finns ω många algoritmer.

Grundläggande begrepp.

Ett beslutsproblem kan ses som en funktion från ändliga teckensträngar till $\{0, 1\}$.

Det finns ω ändliga teckensträngar.

Det finns 2^{ω} beslutsproblem.

En algoritm har en ändlig beskrivning. Det finns ω många algoritmer.

Cantor: $\omega < 2^{\omega}$. Slutsats: Det finns problem som saknar algoritm.

Grundläggande begrepp.

Algoritmkomplexitet vs. problemkomplexitet.

Grundläggande begrepp.

Algoritmkomplexitet vs. problemkomplexitet.

Låt X vara ett problem.

Grundläggande begrepp.

Algoritmkomplexitet vs. problemkomplexitet.

Låt X vara ett problem.

Algoritmkomplexitet. Låt A vara en algoritm för X. Vilken tidskomplexitet har A?

Grundläggande begrepp.

Algoritmkomplexitet vs. problemkomplexitet.

Låt X vara ett problem.

Algoritmkomplexitet. Låt A vara en algoritm för X. Vilken tidskomplexitet har A?

Problemkomplexitet. Vilken tidskomplexitet har snabbast möjliga algoritm för X?

Grundläggande begrepp.

Algoritmkomplexitet vs. problemkomplexitet.

Låt X vara ett problem.

Algoritmkomplexitet. Låt A vara en algoritm för X. Vilken tidskomplexitet har A?

Problemkomplexitet. Vilken tidskomplexitet har snabbast möjliga algoritm för X?

I detta sammanhang mäts tidskomplexitet alltid i termer av indatas storlek.

Grundläggande begrepp.

Beräkna fakultetsfunktionen.

Grundläggande begrepp.

Beräkna fakultetsfunktionen.

$N! = 1 \cdot 2 \cdot \cdots \cdot (N-1) \cdot N$

Grundläggande begrepp.

Beräkna fakultetsfunktionen.

$N! = 1 \cdot 2 \cdot \cdots \cdot (N-1) \cdot N$

Tid: $O(N)$

Grundläggande begrepp.

Beräkna fakultetsfunktionen.

$N! = 1 \cdot 2 \cdot \cdots \cdot (N-1) \cdot N$

Tid: $O(N)$

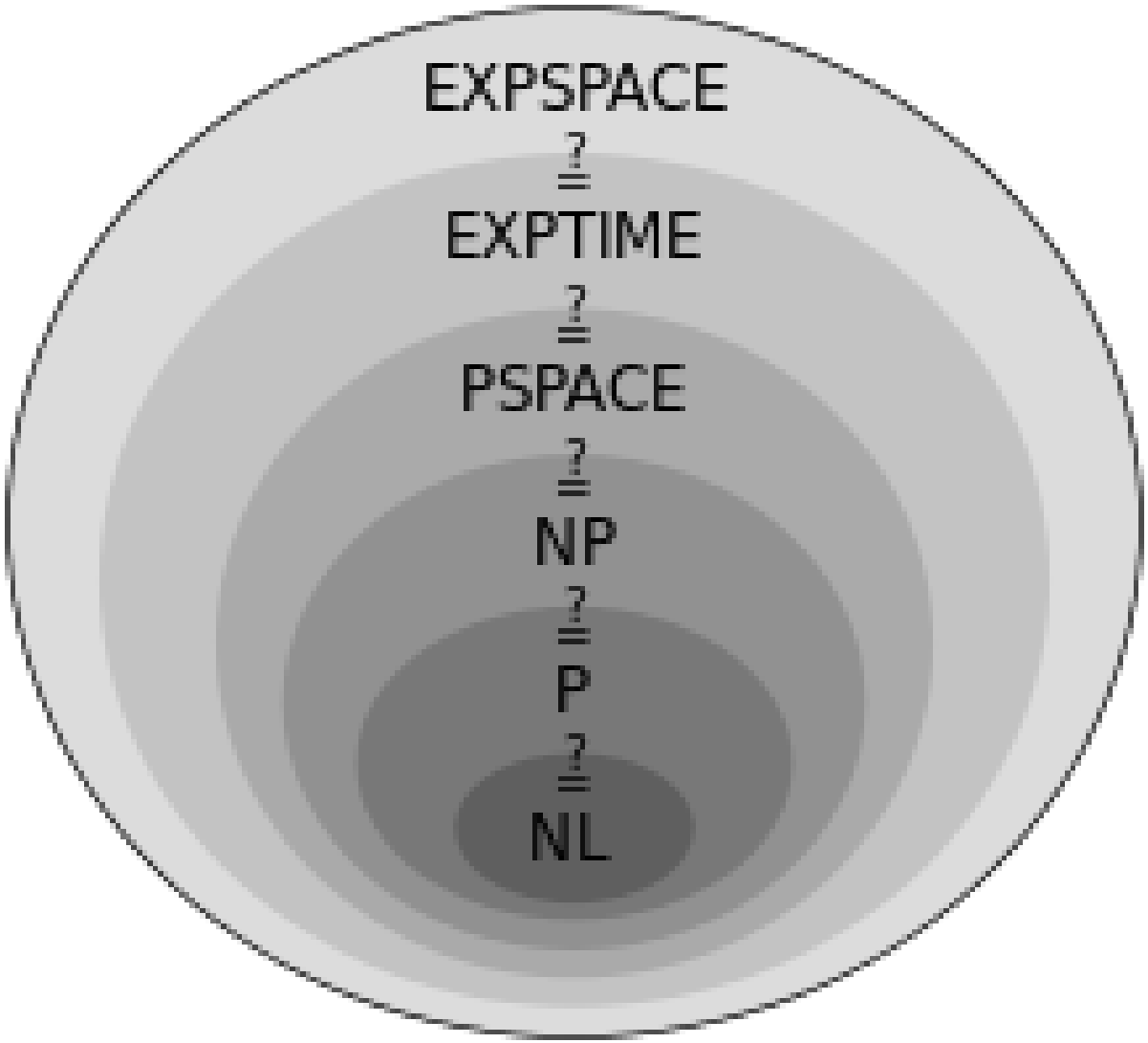
...men indata har bara längd $n = \log_2(N)$. Detta ger $O(N) = O(2^n)$ och algoritmen är exponentiell.

Översikt över komplexitetsteori.

Översikt/komplexitetsteori.

Komplexitetsteori handlar om att klassificera beräkningsproblem efter deras resursanvändning och att relatera sådana klasser med varandra.

Översikt/komplexitetsteori.



Översikt/komplexitetsteori.

P är klassen av problem som är lösbara i polynomisk tid.

Översikt/komplexitetsteori.

P är klassen av problem som är lösbara i polynomisk tid.

NP återkommer vi till.

Översikt/komplexitetsteori.

P är klassen av problem som är lösbara i polynomisk tid.

NP återkommer vi till.

PSPACE är klassen av problem som är lösbara med polynomiskt mycket minne.

Översikt/komplexitetsteori.

P är klassen av problem som är lösbara i polynomisk tid.

NP återkommer vi till.

PSPACE är klassen av problem som är lösbara med polynomiskt mycket minne.

EXPTIME är klassen av problem som är lösbara i $2^{poly(n)}$ tid.

Översikt/komplexitetsteori.

P är klassen av problem som är lösbara i polynomisk tid.

NP återkommer vi till.

PSPACE är klassen av problem som är lösbara med polynomiskt mycket minne.

EXPTIME är klassen av problem som är lösbara i $2^{poly(n)}$ tid.

EXPSPACE är klassen av problem som är lösbara med $2^{poly(n)}$ minne.

Översikt/komplexitetsteori.

Varför är $\mathbf{P} \subseteq \mathbf{EXPTIME}$?

Översikt/komplexitetsteori.

Varför är $\mathbf{P} \subseteq \mathbf{EXPTIME}$?

Varför är $\mathbf{P} \subseteq \mathbf{PSPACE}$?

Översikt/komplexitetsteori.

P är klassen av problem som är lösbara i polynomisk tid.

Översikt/komplexitetsteori.

P är klassen av problem som är lösbara i polynomisk tid.

NP är klassen av problem vars lösningar kan verifieras i polynomisk tid.

Översikt/komplexitetsteori.

Problem. SAT.
Indata. En propositionslogisk formel F på CNF.
Fråga. Är F satisfierbar?

Översikt/komplexitetsteori.

Problem. SAT.
Indata. En propositionslogisk formel F på CNF.
Fråga. Är F satisfierbar?

Lösning: tilldelning $f : \{x_1, \ldots, x_k\} \to \{0, 1\}$.

Översikt/komplexitetsteori.

Problem. SAT.
Indata. En propositionslogisk formel F på CNF.
Fråga. Är F satisfierbar?

Lösning: tilldelning $f : \{x_1, \ldots, x_k\} \to \{0, 1\}$.

Enkelt att i polynomisk tid kontrollera om f är satisfierande.

Översikt/komplexitetsteori.

Problem. SAT.
Indata. En propositionslogisk formel F på CNF.
Fråga. Är F satisfierbar?

Lösning: tilldelning $f : \{x_1, \ldots, x_k\} \to \{0, 1\}$.

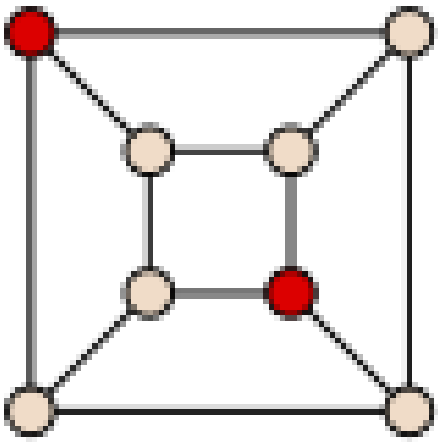
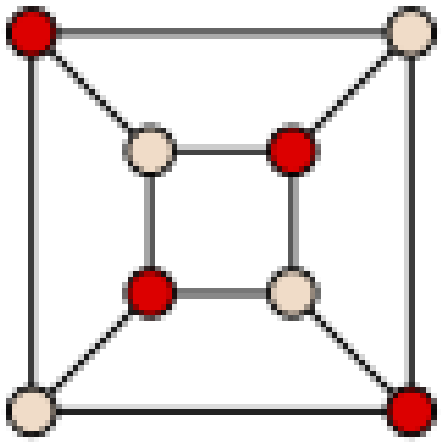
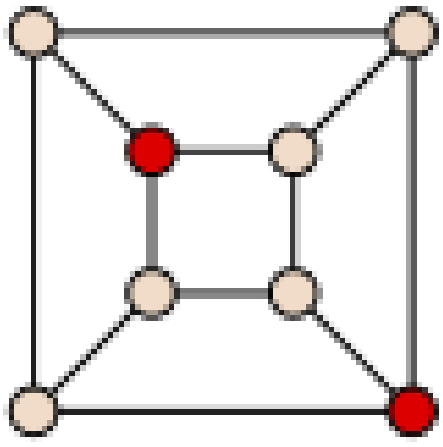
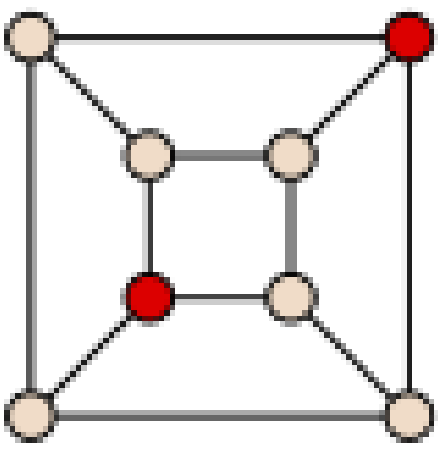
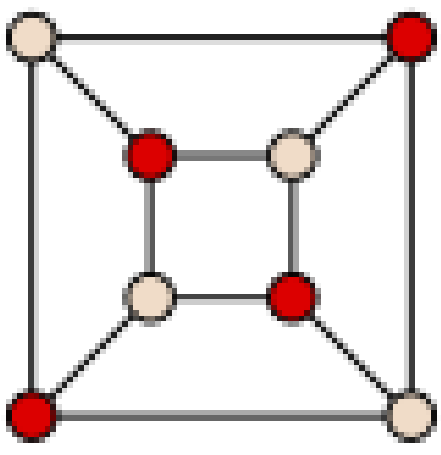
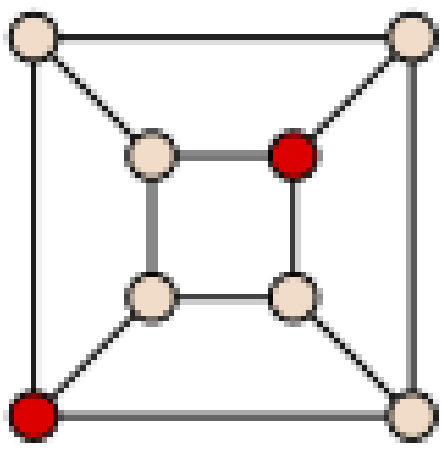
Enkelt att i polynomisk tid kontrollera om f är satisfierande.

SAT är i **NP**.

Översikt/komplexitetsteori.

Problem. Oberoende Mängd.
Indata. En graf $G = (V, E)$ och ett heltal K.
Fråga. Innehåller G en oberoende mängd med K eller fler noder?

Översikt/komplexitetsteori.

Översikt/komplexitetsteori.

Problem. Oberoende Mängd.
Indata. En graf $G = (V, E)$ och ett heltal K.
Fråga. Innehåller G en oberoende mängd med K eller fler noder?

Översikt/komplexitetsteori.

Problem. Oberoende Mängd.
Indata. En graf $G = (V, E)$ och ett heltal K.
Fråga. Innehåller G en oberoende mängd med K eller fler noder?

Lösning: En delmängd $V' \subseteq V$.

Översikt/komplexitetsteori.

Problem. Oberoende Mängd.
Indata. En graf $G = (V, E)$ och ett heltal K.
Fråga. Innehåller G en oberoende mängd med K eller fler noder?

Lösning: En delmängd $V' \subseteq V$.

1. Verifiera att $|V'| \geq K$.

Översikt/komplexitetsteori.

Problem. Oberoende Mängd.
Indata. En graf $G = (V, E)$ och ett heltal K.
Fråga. Innehåller G en oberoende mängd med K eller fler noder?

Lösning: En delmängd $V' \subseteq V$.

1. Verifiera att $|V'| \geq K$.
2. Verifiera att det inte finns någon båge mellan $v, w \in V'$.

Översikt/komplexitetsteori.

Problem. Oberoende Mängd.
Indata. En graf $G = (V, E)$ och ett heltal K.
Fråga. Innehåller G en oberoende mängd med K eller fler noder?

Lösning: En delmängd $V' \subseteq V$.

1. Verifiera att $|V'| \geq K$.
2. Verifiera att det inte finns någon båge mellan $v, w \in V'$.

Båda testerna kan göras i polynomisk tid.

Översikt/komplexitetsteori.

Problem. Oberoende Mängd.
Indata. En graf $G = (V, E)$ och ett heltal K.
Fråga. Innehåller G en oberoende mängd med K eller fler noder?

Lösning: En delmängd $V' \subseteq V$.

1. Verifiera att $|V'| \geq K$.
2. Verifiera att det inte finns någon båge mellan $v, w \in V'$.

Båda testerna kan göras i polynomisk tid.

Oberoende Mängd är i **NP**.

Översikt/komplexitetsteori.

Varför är **NP** en intressant klass av problem?

Översikt/komplexitetsteori.

Varför är $\mathbf{P} \subseteq \mathbf{NP}$?

Översikt/komplexitetsteori.

Varför är $\mathbf{P} \subseteq \mathbf{NP}$?

Låt X vara ett godtyckligt problem i $\mathbf{P}$ med algoritm A. Algoritmen A går i polynomisk tid per definition.

Översikt/komplexitetsteori.

Varför är $\mathbf{P} \subseteq \mathbf{NP}$?

Låt X vara ett godtyckligt problem i $\mathbf{P}$ med algoritm A. Algoritmen A går i polynomisk tid per definition.

Välj godtyckligt indata I.

Översikt/komplexitetsteori.

Varför är $\mathbf{P} \subseteq \mathbf{NP}$?

Låt X vara ett godtyckligt problem i $\mathbf{P}$ med algoritm A. Algoritmen A går i polynomisk tid per definition.

Välj godtyckligt indata I.

Vi kan verifiera om en påstådd lösning s (som är antingen "ja" eller "nej") till I är korrekt genom att applicera algoritmen A på instansen I och jämföra svaret med s. Detta tar endast polynomisk tid.

Översikt/komplexitetsteori.

Varför är $\mathbf{P} \subseteq \mathbf{NP}$?

Låt X vara ett godtyckligt problem i $\mathbf{P}$ med algoritm A. Algoritmen A går i polynomisk tid per definition.

Välj godtyckligt indata I.

Vi kan verifiera om en påstådd lösning s (som är antingen "ja" eller "nej") till I är korrekt genom att applicera algoritmen A på instansen I och jämföra svaret med s. Detta tar endast polynomisk tid.

X är därmed ett problem i $\mathbf{NP}$. Eftersom X är godtyckligt vald i $\mathbf{P}$ så gäller $\mathbf{P} \subseteq \mathbf{NP}$.

Polynomiska reduktioner.

Polynomiska reduktioner.

Polynomiska reduktioner används exempelvis för att identifiera de "svåraste" problemen i en komplexitetsklass.

Polynomiska reduktioner.

Polynomiska reduktioner används exempelvis för att identifiera de "svåraste" problemen i en komplexitetsklass.

Antag att vi har en algoritm A för ett problem X och att A går i $O(f(n))$ tid. Polynomisk reduktioner ger oss (under vissa förutsättningar) en möjlighet att lösa problemet B i $O(f(n^c))$ tid genom att återanvända algoritmen A.

Polynomiska reduktioner.

Låt X och Y vara beslutsproblem med instansmängderna I_X och I_Y.

Polynomiska reduktioner.

Låt X och Y vara beslutsproblem med instansmängderna I_X och I_Y.

Det finns en *polynomisk reduktion* från X till Y om följande gäller. Det existerar en algoritm A som avbildar I_X på I_Y med följande egenskaper:

1. A går i polynomisk tid,
2. om $I \in I_X$ är en "ja"-instans så är $A(I)$ en "ja"-instans till problemet Y och
3. om $I \in I_X$ är en "nej"-instans så är $A(I)$ en "nej"-instans till problemet Y.

Polynomiska reduktioner.

Problem. SAT.
Indata. En propositionslogisk formel F på CNF.
Fråga. Är F satisfierbar?

Problem. Oberoende Mängd.
Indata. En graf $G = (V, E)$ och ett heltal K.
Fråga. Innehåller G en oberoende mängd med K eller fler noder?

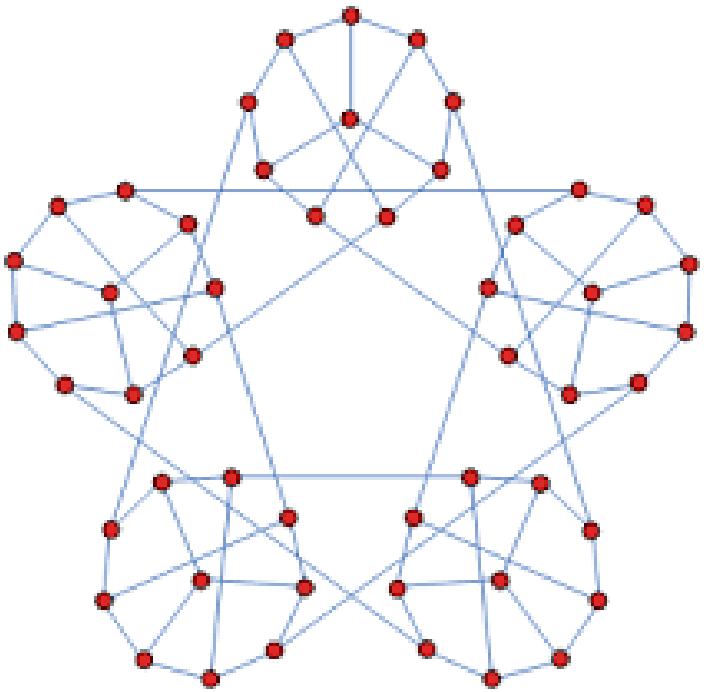
Polynomiska reduktioner.

$$(p \vee q \vee \neg r) \wedge (q \vee s \vee t) \wedge (\neg p \vee \neg t) \wedge (p \vee q \vee r \vee s \vee t)$$

Polynomiska reduktioner.

Applicera algoritm A.

Polynomiska reduktioner.



$K = 12$

Polynomiska reduktioner.

Antag att B är en algoritm för problemet Y som går i $O(f(n))$ tid.

Polynomiska reduktioner.

Antag att B är en algoritm för problemet Y som går i $O(f(n))$ tid.

Antag att det finns en polynomisk reduktion från X till Y (med algoritm A).

Polynomiska reduktioner.

Antag att B är en algoritm för problemet Y som går i $O(f(n))$ tid.

Antag att det finns en polynomisk reduktion från X till Y (med algoritm A).

Då finns en algoritm som löser X i $O(f(n^c))$ tid för något c.

Polynomiska reduktioner.

Antag att B är en algoritm för problemet Y som går i $O(f(n))$ tid.

Antag att det finns en polynomisk reduktion från X till Y (med algoritm A).

Då finns en algoritm som löser X i $O(f(n^c))$ tid för något c.

Indata: Instans I av X.
Låt $J = A(I)$.
Returnera $B(J)$.

Polynomiska reduktioner.

Antag att X är ett problem som har "hög" tidskomplexitet.

Polynomiska reduktioner.

Antag att X är ett problem som har "hög" tidskomplexitet.

Antag att det finns en polynomisk reduktion från X till Y.

Polynomiska reduktioner.

Antag att X är ett problem som har "hög" tidskomplexitet.

Antag att det finns en polynomisk reduktion från X till Y.

Då har även Y "hög" tidskomplexitet.

Låt $\mathcal{C}$ vara en komplexitetsklass.

Låt $\mathcal{C}$ vara en komplexitetsklass.

Låt X vara ett problem.

Låt $\mathcal{C}$ vara en komplexitetsklass.

Låt X vara ett problem.

Om alla problem i $\mathcal{C}$ kan polynomiskt reduceras till X så säger man att X är $\mathcal{C}$-*svårt*.

Låt $\mathcal{C}$ vara en komplexitetsklass.

Låt X vara ett problem.

Om alla problem i $\mathcal{C}$ kan polynomiskt reduceras till X så säger man att X är $\mathcal{C}$-*svårt*.

Om X dessutom är med i $\mathcal{C}$ så säger man att X är ett $\mathcal{C}$-*fullständigt* problem.

Låt $\mathcal{C}$ vara en komplexitetsklass.

Låt X vara ett problem.

Om alla problem i $\mathcal{C}$ kan polynomiskt reduceras till X så säger man att X är $\mathcal{C}$-*svårt*.

Om X dessutom är med i $\mathcal{C}$ så säger man att X är ett $\mathcal{C}$-*fullständigt* problem.

Notera: Om X är $\mathcal{C}$-svårt och kan lösas i polynomisk tid så kan alla problem i $\mathcal{C}$ lösas i polynomisk tid.

NP-fullständighet.

Betrakta komplexitetsklassen **NP**.

Betrakta komplexitetsklassen **NP**.

Låt X vara ett problem.

Betrakta komplexitetsklassen **NP**.

Låt X vara ett problem.

Om alla problem i **NP** kan polynomiskt reduceras till X så säger man att X är **NP**-svårt.

Betrakta komplexitetsklassen **NP**.

Låt X vara ett problem.

Om alla problem i **NP** kan polynomiskt reduceras till X så säger man att X är **NP**-svårt.

Om X dessutom är med i **NP** så är X är ett **NP**-fullständigt problem.

NP-fullständighet.

Finns det något **NP**-fullständigt problem?

NP-fullständighet.

Finns det något **NP**-fullständigt problem?

Ja! SAT var det första exemplet.

NP-fullständighet.

Finns det något **NP**-fullständigt problem?

Ja! SAT var det första exemplet.

Det finns nu tusentals **NP**-fullständiga problem beskrivna i litteraturen.

Om ett av dessa kan lösas i polynomisk tid så kan alla det.

Ingen polynomisk algoritm har hittats trots 50+ år av intensivt arbete.

Viktig öppen fråga: Är **P** lika med **NP**?

NP-fullständighet.

Konkret reduktion.

NP-fullständighet.

Konkret reduktion.

Problem. SAT.
Indata. En propositionslogisk formel F på CNF.
Fråga. Är F satisfierbar?

NP-fullständighet.

Konkret reduktion.

Problem. SAT.
Indata. En propositionslogisk formel F på CNF.
Fråga. Är F satisfierbar?

Problem. 3SAT.
Indata. En propositionslogisk formel F på 3-CNF.
Fråga. Är F satisfierbar?

NP-fullständighet.

Betrakta en "för lång" klausul.

$(p \vee q \vee r \vee \neg s)$

NP-fullständighet.

Betrakta en "för lång" klausul.

$(p \vee q \vee r \vee \neg s)$

Introducera en ny variabel t.

NP-fullständighet.

Betrakta en "för lång" klausul.

$(p \vee q \vee r \vee \neg s)$

Introducera en ny variabel t.

Dela upp klausulen i två delar.

$(p \vee q \vee t) \wedge (\neg t \vee r \vee \neg s)$

NP-fullständighet.

Betrakta en "för lång" klausul.

$(p \vee q \vee r \vee \neg s)$

Introducera en ny variabel t.

Dela upp klausulen i två delar.

$(p \vee q \vee t) \wedge (\neg t \vee r \vee \neg s)$

Notera att $(p \vee q \vee r \vee \neg s)$ är satisfierbar om och endast om $(p \vee q \vee t) \wedge (\neg t \vee r \vee \neg s)$ är satisfierbar.

NP-fullständighet.

Låt F vara en propositionslogisk formel på CNF.

NP-fullständighet.

Låt F vara en propositionslogisk formel på CNF.

Applicera transformationen tills alla klausuler har längd ≤ 3.

Observera att den resulterande formeln F' är satisfierbar om och endast om F är satisfierbar.

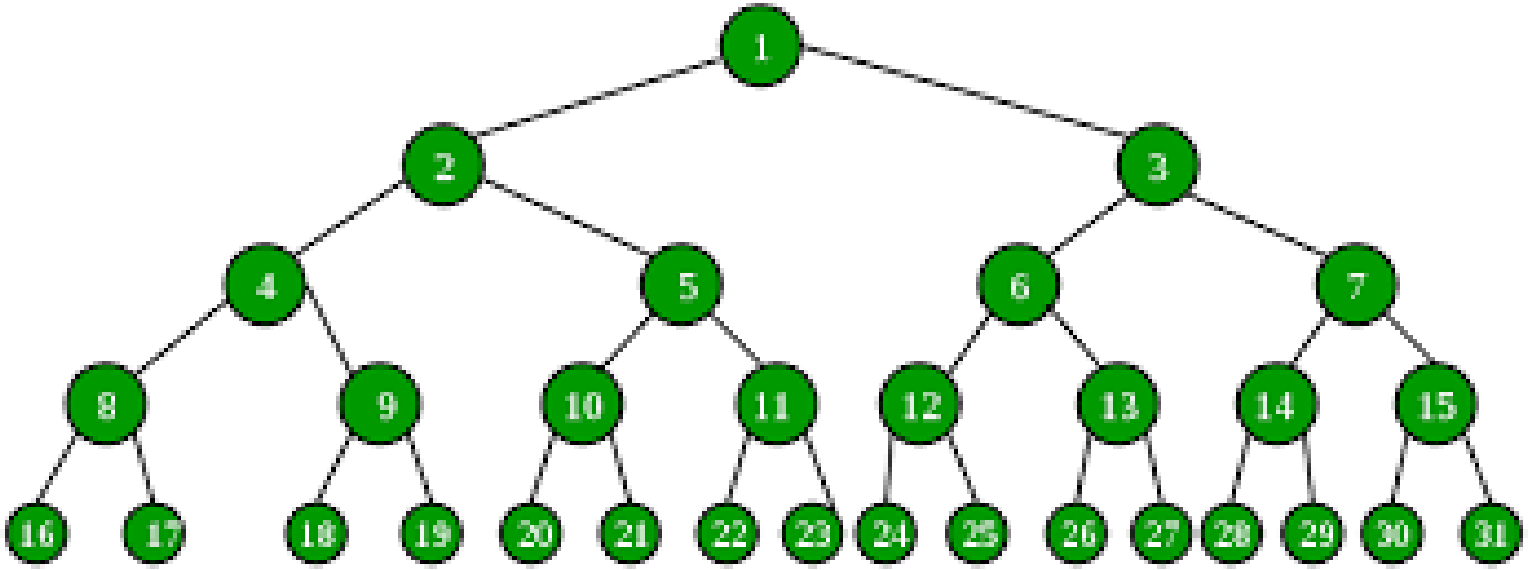
NP-fullständighet.

Tidskomplexitet.

Antag att formeln innehåller n variabler och m klausuler.

NP-fullständighet.

Varje klausul delas succesivt upp i två delar.



Trädets djup är maximalt $\log_2 n$ och delningsfaktorn är 2. Vi får $\leq 2^{\log_2 n} = n$ noder. Total tid: $O(m \cdot n)$.

NP-fullständighet.

Denna metod är en polynomisk reduktion från SAT till 3SAT.
Slutsats: 3SAT är ett **NP**-fullständigt problem.