COMP90057
Advanced Theoretical Computer Science

Complexity

Drawing on material pr
Harald Søndergaard & Elena Kelareva

Part 1 (in 2020)

Second (Spring) Semester, 2020

This part of COMP90057: Complexity

Among problems that computers *can* solve, there are many that are intractable: they take an unreasonably long time to solve, or require an unreasonably large amount of space to solve.

Complexity: runtime bo e runtime of algorithms; space bou methods to deal with intractable problems.

Preparatory reading/viewing

http://cacm.acm.org/magazines/2009/9/
38904-the-status-of-the-p-versus-np-problem/fulltext

Symposium on 50 Years o **the Work of Stephen Cook**

http://www.fields.utoronto.ca/acti P50

http://www.fields.utoronto.ca/vide nt/2774/2019

In algorithms classes, you would have seen a definition of $O(\,)$ notation, something like this

$$g = O(f) \quad \text{iff} \quad \text{for all } n > n_0 : g(n) < c \cdot f(n)\,,$$

for some integers n_0 le,

5

$5n^3 + 2n^2 - 22n + 6 \quad =$

$1 + 2 + \cdots + n$

$$3n \log n = O(n \log n)$$
$$3n \log n = O(n^2)$$

You can also think of $O(\cdot)$ as a set of functions, and thus write $g \in O(f)$. However, we'll stick to Sipser's notation convention.

To determine the time complexity of a *problem*, we examine the asymptotic behaviour of algorithms that solve that problem, that is, decide the underlying language. (We assume that the TM halts on all inputs.)

We want to bound the runn asymptotically, as a function of input size.

In particular, function t is polynomiall $= O(n^r)$ for some positive integer r.

Questions you can try at this point:

Exercise 7.1

Exercise 7.2

Let M be a deterministic (single-tape) Turing machine.

The time complexity of M is the function $t_M : \mathbb{N} \to \mathbb{N}$ defined by

$$t_M(n) = \max_{w \in \Sigma^n} \{ m \mid M(w) \text{ halts after exactly } m \text{ steps} \}$$

This focuses on the worst-case for inputs of length n.

Let $t : \mathbb{N} \to \mathbb{N}$ be a function. The complexity class (of languages) for function t is:

$$TIME(t(n)) = \left\{ L \;\middle|\; \begin{array}{l} L \text{ is decided by some } M \\ \text{with } t_M(n) = O(t(n)) \end{array} \right\}$$

This focuses on the *best* of all machines that decide a language.

A one-tape Turing machine can decide the language

$$A = \{0^k 1^k \mid k \geq 0\}$$

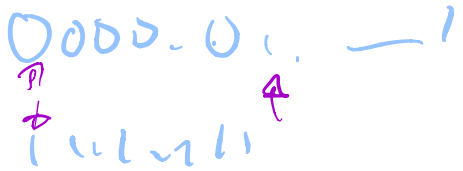
in $O(n \log n)$ time.

The first phase checks wh nd perhaps that it's of the form

The machine then repeatedly scans across its input, rossing off every second 0 and every second 1, checking after eac the number of non-crossed symbols remains even.

Since this halves the number of 0s and 1s (integer division) in each scan, the time taken is $O(n \log n)$ in the worst case.

The machine cannot decide A in linear time: indeed, a linear-time language would have to be regular (Problem 7.49).



A two-tape deterministic machine has lower time complexity.

It can copy all the 1s to its second tape and then match them against the 0s in linear time.

The two kinds of Turing m putational power, but they have different complexity properties.

Thus we lose some of the robustness of the Turing-machine model when we move from studying decidability/computability to (time) complexity.

Fortunately, it is possible to recover much of that robustness.

We need to think more broadly, considering all the polynomial-time deterministic deciders as one class.

Definition: P is the cla deterministic Turing machine in polynomial time, viz.

$$P = \bigcup_{k \in \mathbb{N}} TIME(n^k)$$

$(x^4 + x^3)(x^{12} + 3x^7) \longmapsto x^{16}$

$((x^{12} + 3x^7)^4 + (x^{12} + 3x^7)^3) \to x^{48} + \cdots$

Recall that polynomial functions are closed under addition, multiplication, and composition.

Class P is robust in the se tic computational models are polynomi eterministic machine model will not change the class

Roughly, P corresponds to the class of problems t *reasonable* amount of time.

Recall $t(n)$ is *polynomially bounded* means it is $O(n^r)$ for some positive integer r.

Here we take "exponential" to mean $\Theta(2^{cn})$ for some constant $c > 0$.

There are functions, such ster than every polynomial function, but more slowly than every ex unction.

Hence it is possible that $P \subsetneq NP$, an ame time, that NP-complete problems can be solved in sub-exponential time.

We describe NP below.

6, 35 27, 24

The language size $(\langle x, y \rangle) \simeq \log_2 x + \log y$.

$$RELPRIME = \{\langle x, y \rangle \mid x \text{ and } y \text{ relatively prime}\}$$

is in P. Euclid's algorithm finds the greatest common divisor of x and y: accept iff $\gcd(x, y)$

Likewise, the language

$\log_2 n \sim m \log n$

$$PATH = \left\{ \langle G, s, t \rangle \;\middle|\; \begin{array}{l} G \text{ is a d} \\ \text{with a path from } s \text{ to } t \end{array} \right\}$$

size $(G) = \log_2 n + m(2 \log_2 n) \in \Theta(m \log n)$.

is in P. There are several deterministic algorithms for this problem, each running in polynomial time.

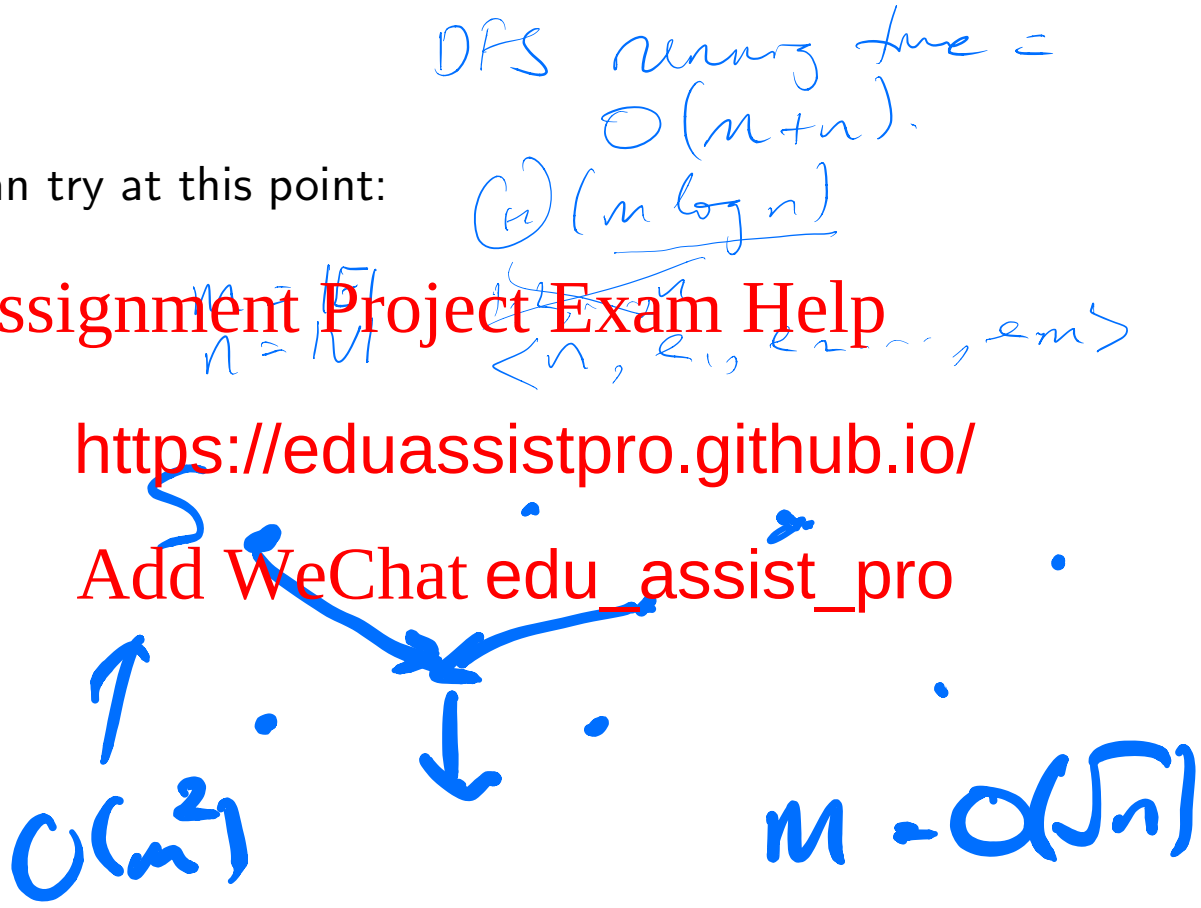
DFS = depth-first search

$\Theta(n^2)$

$\langle G, s, s \rangle \in PATH$.

Questions you can try at this point:

Exercise 7.3
Exercise 7.6
Exercise 7.8
Exercise 7.9
Exercise 7.10
Exercise 7.11
Problem 7.13
Problem 7.14
Problem 7.15



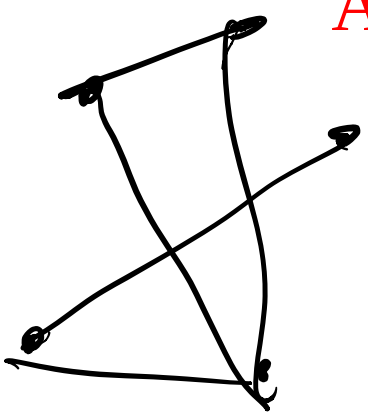
TRIANGLE = $\{\langle G \rangle \mid G$ is undirected graph; G contains a triangle$\}$.

exist? ∃

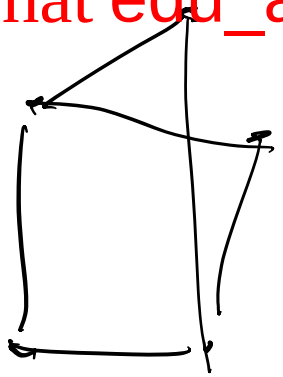
$\{(x,y),(x,z),(y,z)\} \subseteq E$

$G = (V, E)$ $\{(u_1,v_1),(u_2,v_2),\ldots,(u_m,v_m)\}$.

$\{1,\ldots,n\}$



✓



×

1,2,3
1,2,4
1,2,5
⋮
6

$n-2, n-1, n$.

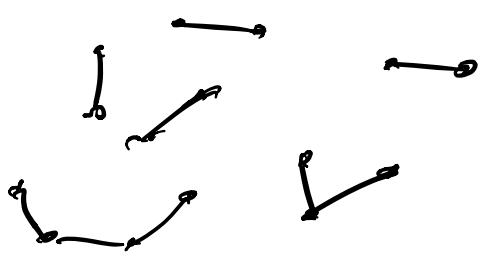
$O(n^3 m)$ running time.

$e_1, e_2, e_3 \to \triangle$?

$e_1, e_2, e_4 \to \triangle$?

$\vdots$

$e_{m-2}, e_{m-1}, e_m \to \triangle$?



The Hamiltonian path problem has underlying language $HAMPATH =$

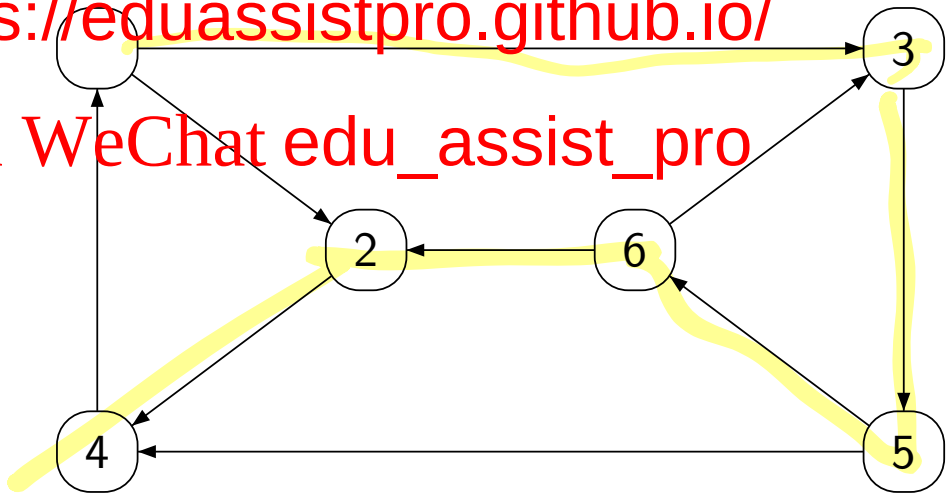
$$\left\{ \langle G, s, t \rangle \;\middle|\; \begin{array}{l} G \text{ is a directed graph with a} \\ \text{Hamiltonian path from } s \text{ to } t \end{array} \right\},$$

where a *Hamiltonian* tly once.

Two instances:

Is there a Hamiltonian path from 1 to 4?

From 1 to 6?



Let G have m nodes. Represent G as, say,

$$x_1 \# y_1 \#\# \cdots \#\# x_n \# y_n \#\#\# m$$

We can build a multi-tape nondeterministic Turing machine N to solve the problem.

One tape holds the graph (d

Machine N writes a guessed sequence of ween 1 and m, on a different tape, each representing a node.

$$\sqcup\ i_1\ \sqcup\ i_2\ \sqcup\ i_3\ \sqcup \cdots \sqcup\ i_m$$

If there is a repeated node on the tape, N rejects. If $i_1 \neq s$ or $i_m \neq t$, N rejects. For each k, if (i_k, i_{k+1}) is not an edge in G, N rejects.

Otherwise, N accepts.

As written, to ease understanding of the process, these verification steps involve more than one tape.

We can simulate the multi-tape machine on a single-tape machine via a polynomial "blowu

The guess is done in polyno

Checking for repetition and verifying the start and en done in polynomial time.

Verifying an edge (by looking up the representation) is also done in polynomial time.

Guessing in linear time is clearly very powerful.

To solve the same problem with a deterministic machine, it seems the best approach is exhaustive se m nodes.

A deterministic machin e (there are $m!$ sequences).

Nobody has come up with a polynomial-time solution – if they did, then we would know that $P = NP$ (see below).

HAMPATH has an important feature shared with many other fundamental computational problems: It is polynomially verifiable.

On the one hand, discovering a Hamiltonian path seems difficult, and certainly we do not have a fas

On the other hand, if somebody claims that a certain p miltonian path, checking their claim is easy: we can do that in p

It is possible – and it seems likely – that verifying a Hamiltonian path is much easier than determining its existence.

A verifier for language A is an algorithm V (that is, a deterministic Turing machine) with

$$A = \{ \quad c \}$$

rtificate

Language A is polynomially verifiable if it h me verifier: it runs in time that is polynomial in the length of w.

$|c| = 2^{|w|}$?

Notice that polynomial verifiability need not be closed under complement.

For example, consider HAMPATH.

There is no obvious (polyn… non-existence of a path.

We don't know how to verify this non-existence with… he same exponential-time method that was needed for determining non-existence (without a certificate) in the first place.

NP is the class of languages that have polynomial-time verifiers.

Theorem: Language A is in NP iff A is decided by some nondeterministic polynomial-time Turing machine.

We can also phrase this as

$$NTIME(t(n)) = \left\{ L \;\middle|\; \begin{array}{l} L \text{ is decided by a nondeterministic} \\ \text{Turing machine in } O(t(n)) \text{ time} \end{array} \right\},$$

where the running time of an NTM is the *maximum* over all computation branches, given a particular input.

This definition is robust, in the sense that NP stays the same even when we change the machine to another nondeterministic model.

In summary:

For P, membership can be decided quickly.

For NP, membership can be verified quickly.

Clearly $P \subseteq NP$. I

$\mathbf{P = N}$

Many computer scientists consider this the

most important unanswered question

in computer science.

Recall the Hamiltonian path problem, *HAMPATH*:

> *Given a directed graph G, and nodes s and t, is there a path, starting in s, ending in t, that visits each node exactly once?*

This problem has the prop erifiable.

Discovering a Ha seems difficult, indeed we do not have a fast algorithm for it.

But if somebody claims to have a valid path, checking that claim is easy (in polynomial time): The path is a certificate.

It is possible that verifying a Hamiltonian path is much easier than determining its existence.

A clean *NP* problem is

$$SUBSET\text{-}SUM = \{\langle S, t\rangle \mid \sum_{x\in A} x = t \text{ for some } A \subseteq S\}$$

(There are several variants; here we consider A and S to be multisets that allow repetition of elements.)

Example: $S = \{2, 2,$ $t = 11$? / $t = 6$?

No polynomial-time decider is known, but the descr A itself provides a succinct certificate.

A nondeterministic TM decides *SUBSET-SUM* in polynomial time:

1. *Guess* a subset $A \subseteq S$, nondeterministically.
2. Accept if $\sum_{x\in A} x = t$, otherwise reject.

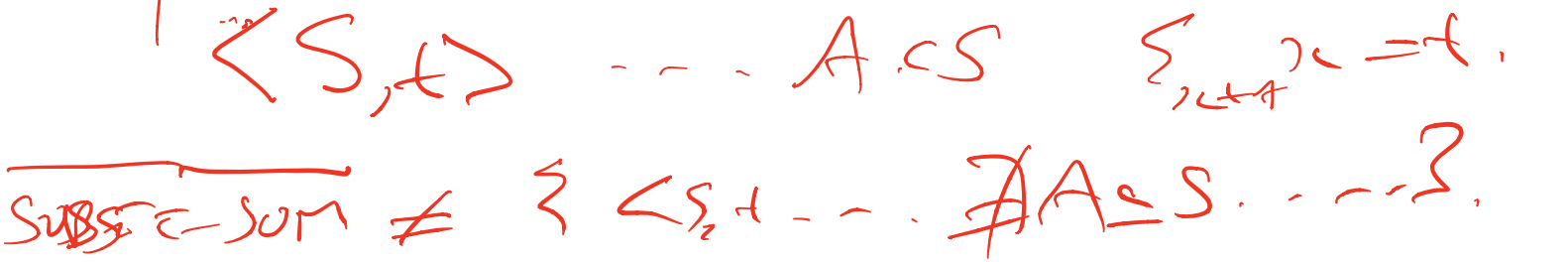
Questions you can attempt at this stage

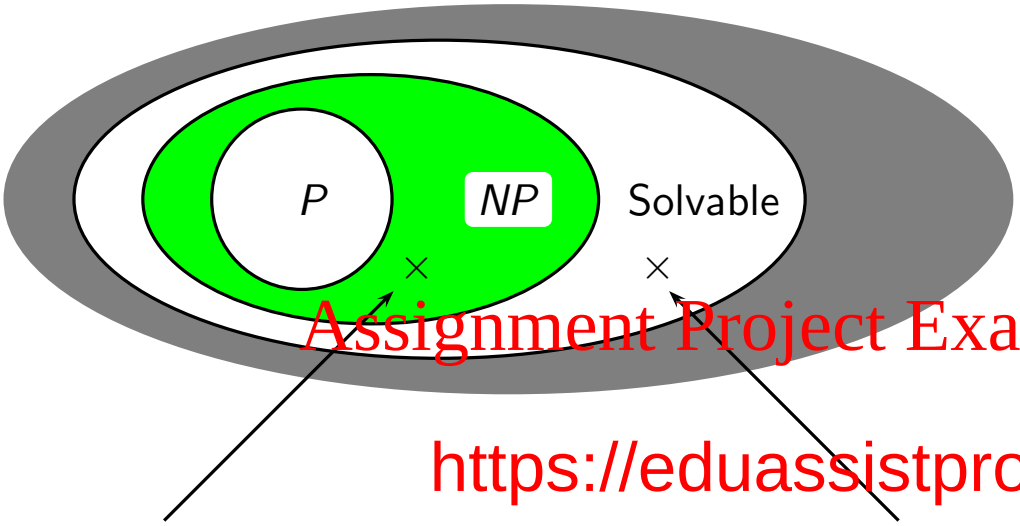
Exercise 7.7
Exercise 7.12
Problem 7.16
Problem 7.19

Recall that $\overline{HAMPATH}$ has no obvious certificate of the non-existence of a path.

Similarly *SUBSET-SUM*

Namely, how could we quickly verify that S sums to t?





Problem which provably can be solved by a NDTM in polynomial time, though not by a deterministic TM.

Does one exist?

at provably omial time by a NDTM ("nondeterministically intractable").

EXPSPACE-complete problems, e.g.,. . .

deciding whether two regexs represent different languages. Four allowed operators: union, concatenation, Kleene star, and squaring (two copies)

Via the concept of reducibility, we can classify problems by *hardness*.

Crucial point: when we talk about reducing problem A to problem B with regard to time complexity, the (mechanistic) reduction must take place in polynomial time.

A is polynomial-time (mapping) reducible to B, iff there is some polynomial-time (deterministic) computable $f : \Sigma^* \to \Sigma^*$, such that for all w,

$$w \in A \text{ iff } f(w) \in B.$$

We say that f is a polynomial-time reduction from A to B.

Since the class of polynomials is closed under composition, we have:

Theorem: The relation $\leq_P$ is a pre-order: reflexive and transitive (and not anti-symmetric: can have $A \leq_P B$ and $B \leq_P A$, with $A \neq B$).

However, we can define an equivalence relation $\equiv_P$ on the set of all languages by saying $A \equiv_P B$ if $A \leq_P B$ and $B \leq_P A$.

The collection of equivalence classes forms a partial order.

Theorem: If $A \leq_P B$ and $B \in P$ then $A \in P$.

Can you prove this?

B is *NP*-hard iff *every* $A \in NP$ is reducible to B in polynomial time.

B is *NP*-complete iff

1. $B \in NP$, and
2. B is *NP*-hard.

We sometimes denote the cla ages by *NPC*.

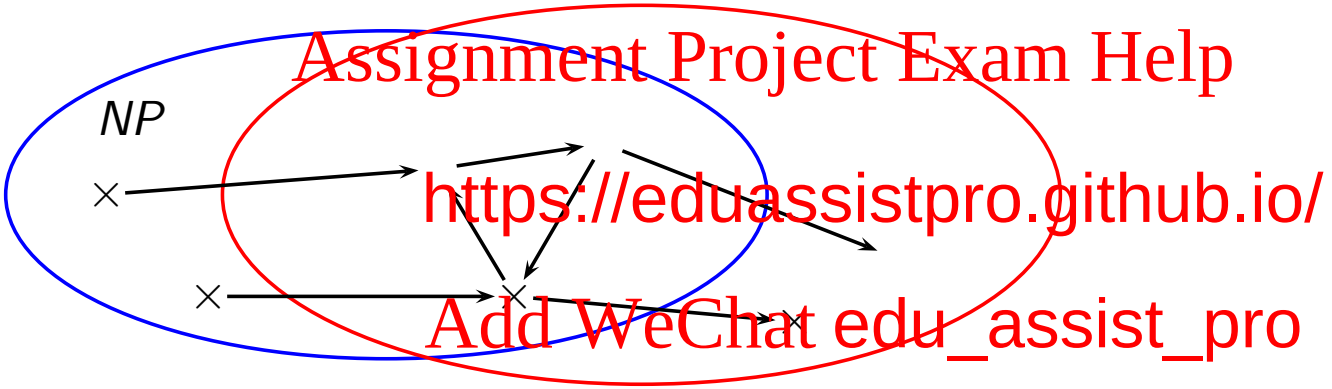
The *NP*-complete languages are interesting be

Theorem: If $B \in NPC$ and $B \in P$ then $P = NP$.

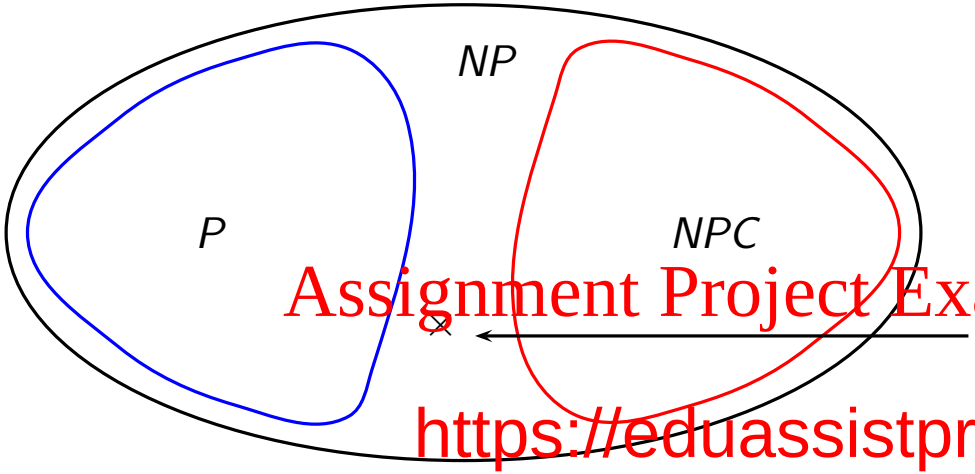
Theorem: If $B \in NP\text{-}hard$ and $B \leq_P C$ then $C \in NP\text{-}hard$.

(If also $C \in NP$, then $C \in NPC$.)

NP-complete problems are the *NP*-hard problems in *NP*.



Arrows indicate polynomial-time reduction (a transitive relation).



If $P \neq NP$ then we have this picture.

Moreover, if $P \neq NP$, this gap is known to be habited (this is Ladner's Theorem).

Let $NPI = NP \setminus (P \cup NPC)$.

Difficult problems that in spite of many attempts have not been proven NP-complete, such as graph isomorphism, are sometimes suggested as candidate members of NPI.

`http://www.quantamagazine.org/20151214-graph-isomorphism-algorithm/`

Questions to attempt:

Problem 7.18
Problem 7.20

The plan for demonstrating *NP*-completeness:

- The *ancestor* of all *NPC* problems: *SAT*.
  1. Show $SAT \in NP$.
  2. Show that
- For each subsequent
  1. Show $B \in NP$.
  2. Show B is *NP*-hard by describing a pol m some other known *NP*-hard problem to

Cook followed exactly this path in his 1971 paper.

However, his proof relied on a weaker concept of (Turing) reducibility.

To describe *SAT*, we introduce some basics of propositional logic.

Let the Boolean values be 0 and 1 and let V be a set of Boolean variables.

A truth assignment $0, 1\}$.

Well-formed propositional formulas include 0, 1, for all $x \in V$.

Moreover, if φ and ψ are well-formed form φ (negation), $\varphi \wedge \psi$ (conjunction), and $\varphi \vee \psi$ (disjunction)—and we can add more connectives if we want.

Truth tables for $\neg$, $\wedge$, and $\vee$ determine truth value of formula φ

For $x \in V$, x and $\neg x$ are literals.

A clause is a disjunction of literals (using $\vee$).

A formula is in conj unction ($\wedge$) of clauses. Every formula ha me.

converting one to the other might require more than p

Formula φ is satisfiable if there is some o the variables such that the truth value of formula φ is 1.

Language $SAT = \{\varphi \mid \varphi \text{ is in CNF and is satisfiable}\}$.

If φ has n (distinct) variables, then there are 2^n relevant truth assignments.

$\varphi = (x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_2 \vee x_3)$

$\psi = (x_1 \vee x_2) \wedge (\overline{x_1}$

Which of these formulas is satisfiable?

To show that *SAT* is in *NP*, we build a two-tape nondeterministic TM over alphabet $\{0, 1, \wedge, \vee, \#, \square, \leftarrow, \oplus, \ominus\}$. Let $\bar{\imath}$ denote a string of i 1s and encode

$$x_i \quad \text{by} \quad \bar{\imath}\oplus$$

$$\neg x_i \quad \text{by} \quad \bar{\imath}\ominus$$

For example, $(x_1 \vee$

$$\underbrace{1}_{\text{variables}} \qquad 11\oplus$$

The NTM will check that this is a valid *SAT* instance.

Then it will guess-and-check as usual, using tape 2 to store the guessed truth assignment.

Why are these labels (allowed to be) in unary?

$x_1 \leftarrow 0 \quad x_2 \leftarrow 0 \quad x_3 \leftarrow 1$

Tape 2: $1 \leftarrow 0 \# 11 \leftarrow 0 \# 111 \leftarrow 1 \sqcup$

Tape 1: $1\#11\#111 \square 1 \oplus \vee 11 \ominus \wedge 1 \ominus \vee 111 \oplus$

$(x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_3) \wedge (x_1 \vee \overline{x_3})$

The latter checking happens in $O(n^2)$ time:

- Scan to the formula's $\ominus$
- While there is another clause:
  - For each literal:
    - If the truth value differs from the tape-1 value (i.e., $0 \neq \oplus$, $1 \neq \ominus$), scan for next $\vee$: if $\wedge$ (or $\sqcup$) is found before $\vee$, reject
    - Else (the variable's value agrees) break out of this for loop
  - Scan just past next $\wedge$ to the start of the next clause
  - If no $\wedge$ is found (we have checked every clause) accept.

Question to attempt:

Exercise 7.5

The important part is to show that SAT is NP-hard.

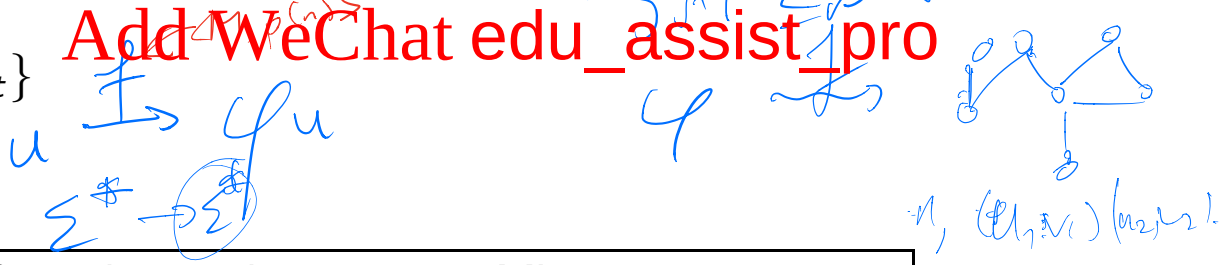
Let M be a nondeterministic TM, whose running time is bounded by polynomial p (that is, given input of size n, on every computation *branch* M halts within $p(n)$ steps).

$Q = \{q_0, q_1, \ldots, q_m$

$\Gamma = \{\sqcup, a_1, \ldots, a_s, a_{s+1}, \ldots, a_t\}$

$\Sigma = \{a_{s+1}, \ldots, a_t\}$

$u \in \Sigma^n$.



We construct a formula φ_u that mimics M's operation on u.

Formula φ_u and its construction depend on $p(n)$.

$A \leq_p B$ $A = \{0, 00, 0000, 010, \ldots\}$

$f(u)$ $f: \Sigma^* \to \Sigma^*$ $B = \{1, 11, 1111, 101, \ldots\}$

$u \in A$ if and only if $f(u) \in B$.

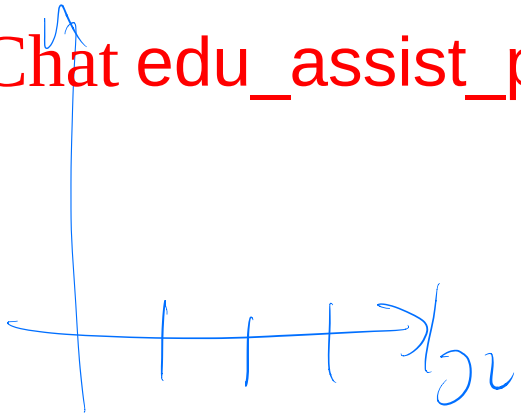
n	2^n	n^2
1	2	1
2	4	4
3	8	9
4	16	16
5	32	25
6	64	36
7	128	49
10	1024	100
20	10^6	400
30	10^9	900

$A \in NP$

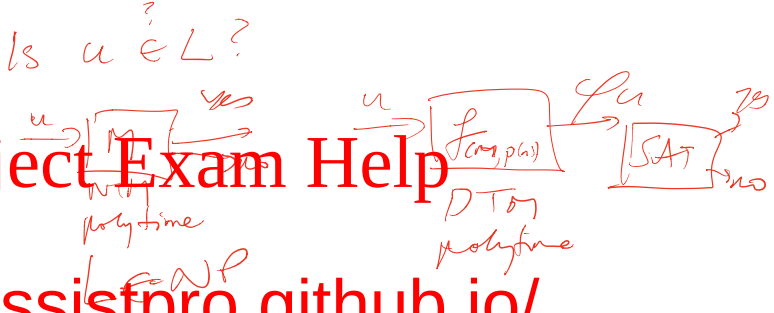
$A \leq_p SAT$

$u \in A \iff \varphi_u \in SAT.$

$f(u) = \varphi_u$



The idea is to create a formula φ_u that it is satisfiable iff the NTM M accepts u.



We need to convince ourselves that

- φ_u emerges as a for
- the size of φ_u i articular, the process to create φ_u runs in polynomial time);
- if φ_u is satisfiable, then M accepts
- if M accepts u, then φ_u is satisfiable.

In short, φ_u will speak the truth, the whole truth, and nothing but the truth, about M's computation history on input u.

The number of propositional variables in the formula is huge, but polynomial in n, the size of u.

Machine M: $m+1$ states, $t+1$ symbols, and at most $p(n)$ time steps. Hence at most $p(n)$ tape cells are touched. Why??

The variables have the foll interpretations:

Q_{ik}	$0 \leq i \leq m$ $0 \leq k \leq p(n)$	M at t
P_{jk}	$0 \leq j \leq p(n)$ $0 \leq k \leq p(n)$	M is scanning position j at time k
S_{jrk}	$0 \leq j \leq p(n)$ $0 \leq r \leq t$ $0 \leq k \leq p(n)$	Tape position j contains symbol a_r at time k

Capturing initial configuration (with input string $u = a_{r_1} \cdots a_{r_n}$): Take the conjunction of

$$
\begin{array}{l}
Q_{00} \\
\wedge\ P_{00} \\
\wedge\ S_{000} \\
\wedge \\
\wedge \qquad\qquad 2 \\
\vdots \\
S_{nr_n0} \\
S_{(n+1)00} \\
\vdots \\
\wedge\ S_{p(n)00}
\end{array}
$$

Generate these formulas for each time-step $k \in [0, p(n)]$:

To capture state invariants (Remember, $a \Rightarrow b \equiv \neg a \vee b$)

$(0 \leq i < i' \leq m)$:

$\bigvee_{i=0}^{m} Q_{ik}$ At all times, M is in some state,

$Q_{ik} \Rightarrow$

To capture tape-head invariants $(0 \leq j$

$\bigvee_{j=0}^{p(n)} P_{jk}$ At all times, the tape he

$P_{jk} \Rightarrow \neg P_{j'k}$ and not in two different positions

To capture symbol invariants, for $j \in [0, p(n)]$ $(0 \leq r < r' \leq t)$:

$\bigvee_{r=0}^{t} S_{jrk}$ At all times, each cell contains some symbol,

$S_{jrk} \Rightarrow \neg S_{jr'k}$ and no cell contains two different symbols

Symbols not under the tape head are not changed.

(for all $j, k \in [0, p(n)]$, $r \in [0, t]$).

Capturing the (possible) transition $(q_i, a_r) \mapsto (q_{i'}, a_{r'}, d)$: (here i could equal i' and r could equal r')

(this allows for nondeterminism)

ik jk jrk $'$

where

$$j' = \begin{cases} j+1 & \text{if } d = R \\ j \ominus 1 & \text{if } d = L \end{cases}$$

Again, these clauses are added for each time-step k.

Here, $\ominus$ denotes cut-off subtraction ($0 \ominus 1 = 0$).

Since this is a nondeterministic TM, there may be multiple transitions from a given state q_i with a given symbol a_r under the tape head. Only one of the several possible transitions occurs, since the machine cannot be in multiple states at the same time.

The set of clauses specifying [illegible] added for each of the possible transitions from a given st [illegible]. To model this, we add the clauses from the previous slide for [illegible] ossible transition, separated by disjunctions ($\vee$)

To transform this formula into Conjunctive Normal Form, we apply the distributive law $P \vee (Q \wedge R)) \leftrightarrow (P \vee Q) \wedge (P \vee R)$. This increases the size of the formula only locally, overall a constant-factor increase in size. The reduction is still polynomial in the size of the input, u.

To capture inactivity after halting, when the machine is in state q_{accept} or q_{reject}, i.e., $i \in [m+1, m]$, for $j, k \in [0, p(n)]$, $r \in [0, t]$:

$$(Q_{ik} \wedge P_{jk} \wedge S_{jrk})$$

Recall that $p(n)$ is the *longest* nondeterministic computation branch.

To capture acceptance:

(That is, we are in state q_{accept} at time

That's it!

The formula we have constructed is in CNF.

The construction ensures:

- if $u \in L(M)$ then φ_u has a satisfying assignment.
- if $u \notin L(M)$ then φ_u does not have a satisfying assignment.

Many interesting problems have been shown to be NP-hard, without anybody being able to show that they are in NP.

Such a problem may well turn out to require exponential time, even if $P = NP$; inde cidable.

It is possible that some prob nd yet it is not NP-hard. Nothing in our definitions rules this o

Finally, it is possible that some NP-hard problem does not require exponential time. (For example, we may have $P = NP$ and the problem we consider is P-complete.)

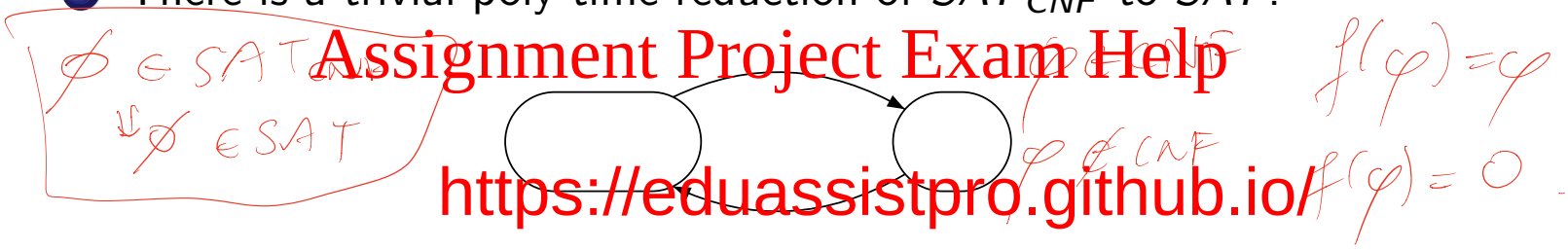
[Read §7.4 and §7.5 of the Sipser text.]

We established that ever ced to *SAT* in polynomial time.

Our construction reduced every *NP* pro , the satisfiability problem for formulas in Conjunctive Normal For

But then SAT (for arbitrary propositional formulas) is also NP-complete:

1. It is in NP (as for the CNF case) Actually, let's check this!, and
2. There is a trivial poly-time reduction of SAT_{CNF} to SAT.



The reduction from right to left is covered by the gener we did in the last lecture.

(A direct reduction is not trivial, by the way: We know how to turn an arbitrary formula into CNF, but that technique can cause an exponential blow-up! However, in the reduction we don't have to preserve equivalence in the forumula. We need only show that the SAT formula is satisfiable if and only if the SAT_{CNF} formula is satisfiable.)

A propositional formula is in 3-CNF if it is in CNF and every clause has exactly three literals (no two of which involve the same variable). An example is

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

3SAT is the language of s

Let us show that it is *NP*-complete.

First, *3SAT* is in *NP*, as the machine that solves SAT_{CNF} in polynomial time also works for *3SAT*.

So it suffices to show $SAT_{CNF} \leq_P 3SAT$.

The reduction is as follows. Given an instance of SAT_{CNF}:

$$C_1 \wedge C_2 \wedge \cdots \wedge C_k$$

we turn each clause C into a conjunction of 3-literal clauses as follows:

- If C is a single litera esh variables x and

$$(\ell \vee x \vee y) \wedge (\ell \vee \neg x \vee y) \wedge (\quad \ell \vee \neg x \vee \neg y)$$

- If C is $(\ell_1 \vee \ell_2)$, produce two clauses using a fresh variable x:

$$(\ell_1 \vee \ell_2 \vee x) \wedge (\ell_1 \vee \ell_2 \vee \neg x)$$

- If C has three literals, leave it as it is.

Clearly satisfiability is preserved for these clauses.

If C has $n > 3$ literals, say

$$(\ell_1 \vee \ell_2 \vee \ell_3 \vee \cdots \vee \ell_n),$$

produce $n-2$ clauses using $n-3$ fresh variables $x_3, \ldots, x_{n-1}$:

$$\begin{aligned} & \wedge \quad (\neg x_3 \vee \ell_3 \vee \\ & \qquad \vdots \\ & \wedge \quad (\neg x_{n-2} \vee \ell_{n-2} \vee x_{n-1}) \\ & \wedge \quad (\neg x_{n-1} \vee \ell_{n-1} \vee \ell_n) \end{aligned}$$

Why does this preserve satisfiability?

$\boxed{\Rightarrow}$ Let t be a truth assignment that satisfies

$$(\ell_1 \vee \ell_2 \vee \ell_3 \vee \cdots \vee \ell_n)$$

and let ℓ_j be the "first" literal that is set to true. We satisfy

$$\begin{array}{rl} & x_3 \quad 3 \\ & \vdots \\ \wedge & (\neg x_{n-2} \vee \ell_{n-2} \vee x_{n-1}) \\ \wedge & (\neg x_{n-1} \vee \ell_{n-1} \vee \ell_n) \end{array}$$

by extending t so that each of $x_3, \ldots, x_j$ is true, while the remaining fresh variables, $x_{j+1}, \ldots, x_{n-1}$ are false.

$\boxed{\Leftarrow}$ Let t' be a truth assignment that satisfies

$$\begin{array}{rl} & (\ell_1 \vee \ell_2 \vee x_3) \\ \wedge & (\neg x_3 \vee \ell_3 \vee x_4) \\ & \vdots \end{array}$$

Then t' also satisfies $C = (\ell_1 \vee \ell_2 \vee \ell_3)$.

Proof by contradiction: assume that it doesn't. Then every ℓ_j is false.

So x_3 is true, hence x_4 is, and so on, including x_{n-1}.

But then $(\neg x_{n-1} \vee \ell_{n-1} \vee \ell_n)$ is false.

We have a contradiction; so C is satisfiable.

Sipser's proof of the *NP*-completeness of 3SAT (Corollary 7.42) is rather different (as is his proof of the *NP*-completeness of SAT).

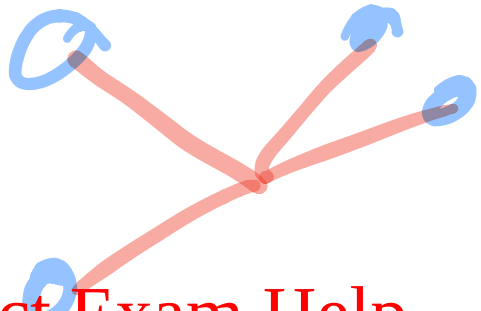
It is instructive to read both.

From Sipser's proof, it is clear that $x \vee x \vee x$ an acceptable 3-CNF formula. We choose to rule tha

Either way can be justified, but our assumption that t iterals in a 3-CNF clause involve different variables makes it more obvious that later proofs of *NP*-hardness, which are based on reductions from *3SAT*, actually work.

Questions you can try at this point:

Problem 7.22
Problem 7.24
Problem 7.25
Problem 7.26

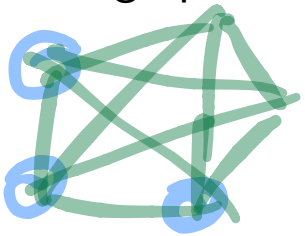


Let $G = (V, E)$ be an undirected graph.

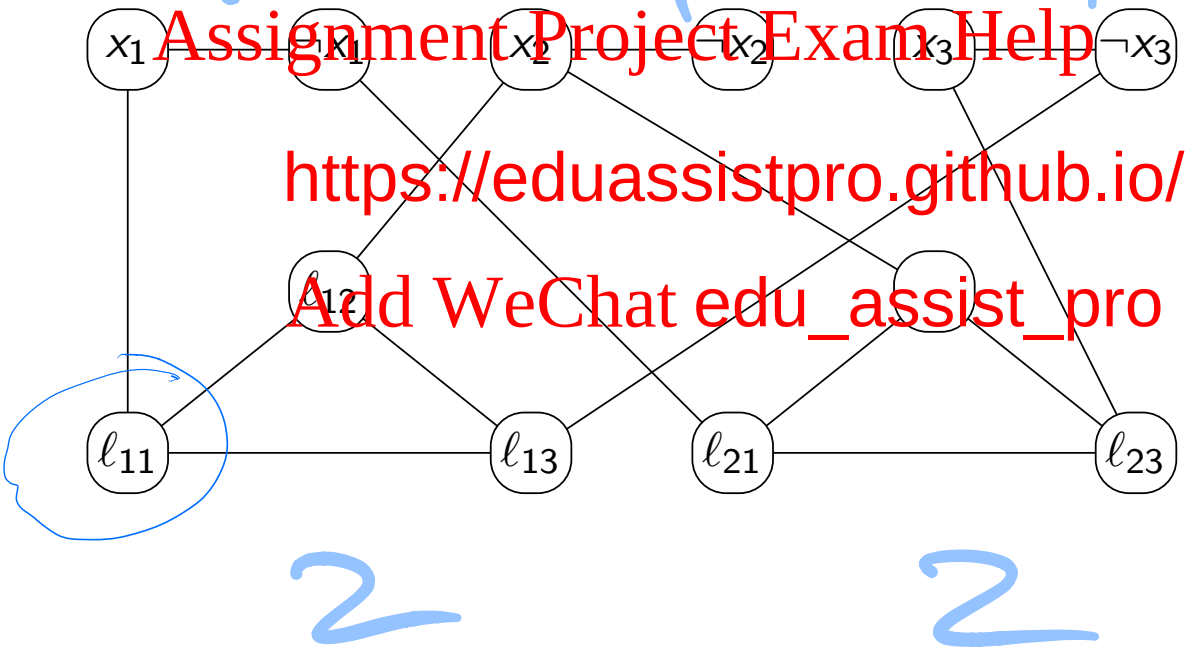
$S \subseteq V$ is a vertex cover of G iff for each edge $(v_1, v_2) \in E$, at least one of v_1, v_2 is in S.

VERTEX-COVER =

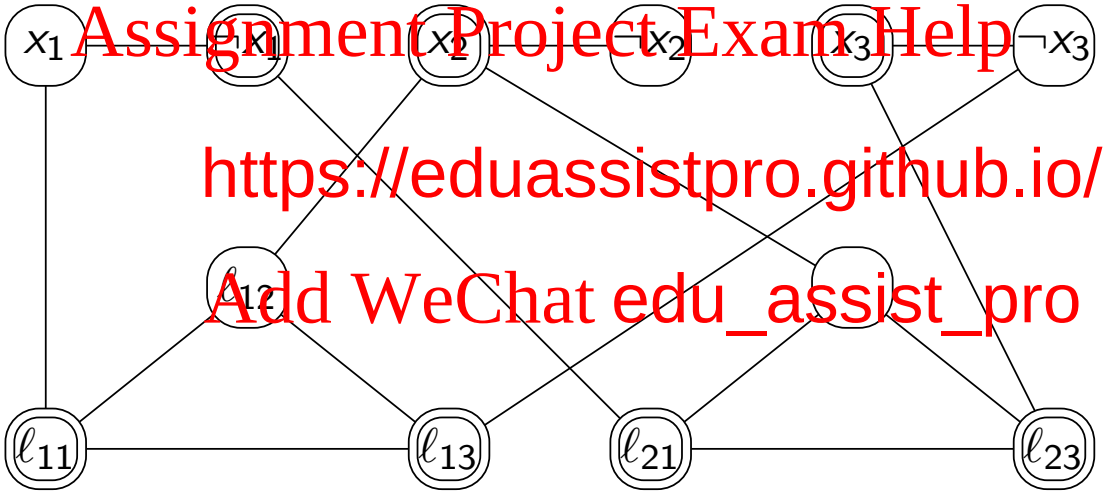
$$\{ \langle G, k \rangle \mid G \text{ is a graph that has a vertex c} \qquad k \}$$



Does this graph have a vertex cover of size 1? Of size 7?



Does this graph have a vertex cover of size 1? Of size 7?



We show that *VERTEX-COVER* =

$$\{\ \langle G, k\rangle \mid G \text{ is a graph that has a vertex cover of size } k\ \}$$

is *NP*-complete.

The problem (language nistic TM can guess a set of k nodes S and check in polyn r they "cover" E.

All it needs is a linear scan through E, and for each edge, a linear scan of the guess S.

$G = \langle n, (u_1, v_1), (u_2, v_2) \ldots, (u_m, v_m) \rangle$

$S = \{ s_1, s_2 \ldots, s_k \}$

For a given 3-CNF formula φ we construct $\langle G, k \rangle$ such that G has a vertex cover of size k iff φ is satisfiable.

Let $\varphi = (\ell_{11} \vee \ell_{12} \vee \ell_{13}) \wedge \cdots \wedge (\ell_{m1} \vee \ell_{m2} \vee \ell_{m3})$ have n variables $x_1, \ldots, x_n$.

We produce $3m + 2$ two per variable:

$\{x_i, \neg x_i \mid 1 \leq i \leq$

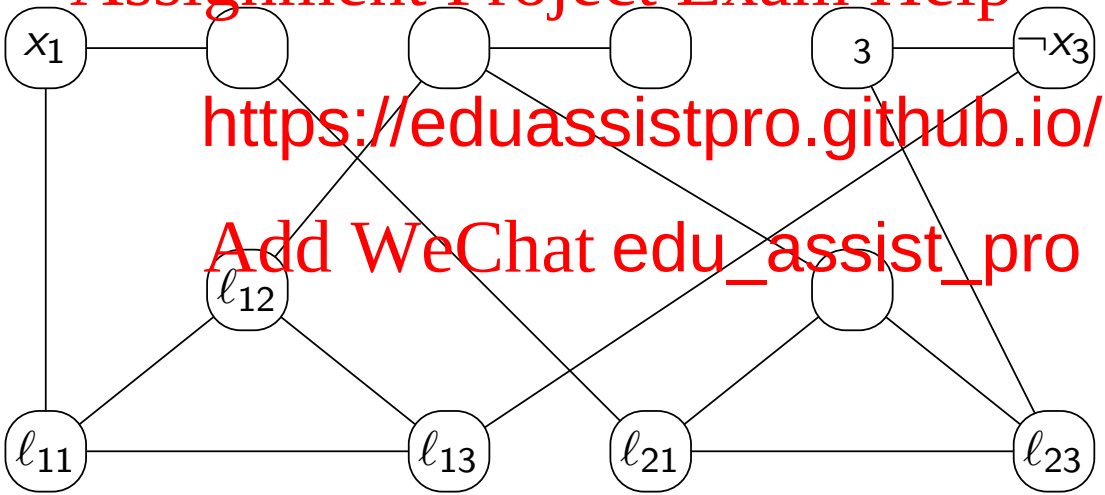
and $6m + n$ edges:

$$\begin{aligned} T_i &= \{(x_i, \neg x_i)\} \\ C_k &= \{(\ell_{k1}, \ell_{k2}), (\ell_{k2}, \ell_{k3}), (\ell_{k3}, \ell_{k1})\} \\ L_k &= \{(\ell_{k1}, v_{k1}), (\ell_{k2}, v_{k2}), (\ell_{k3}, v_{k3})\} \end{aligned}$$

where v_{kj} is x_i or $\neg x_i$, the same as ℓ_{kj}.

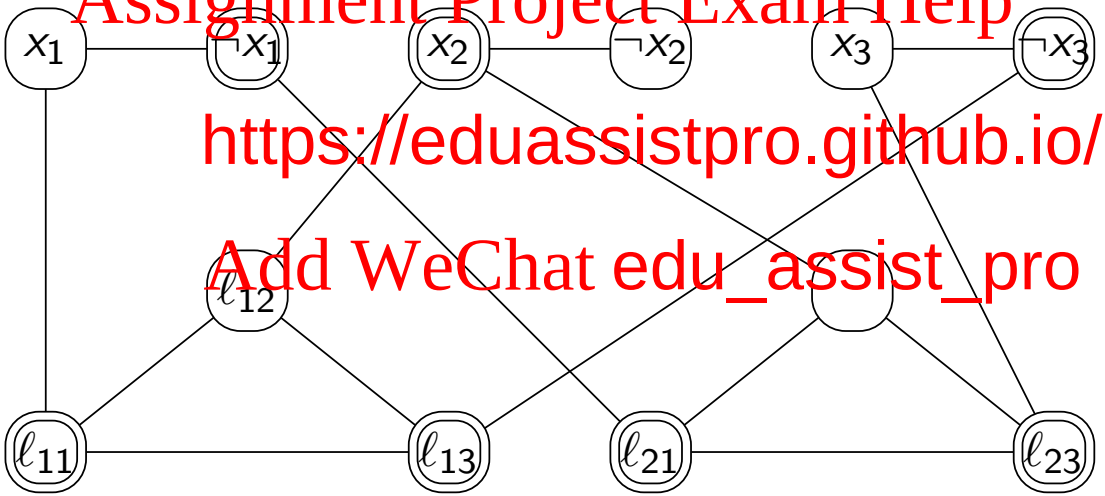
The generated instance is $\langle G, k = 2m + n \rangle$.

For example, $(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3)$ is translated to the problem of whether this graph has a cover of size $2 \cdot 2 + 3 = 7$:



By this construction, every vertex cover contains at least $2m + n$ nodes.

$(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3)$ is translated to the problem of whether this graph has a cover of size $2 \cdot 2 + 3 = 7$:



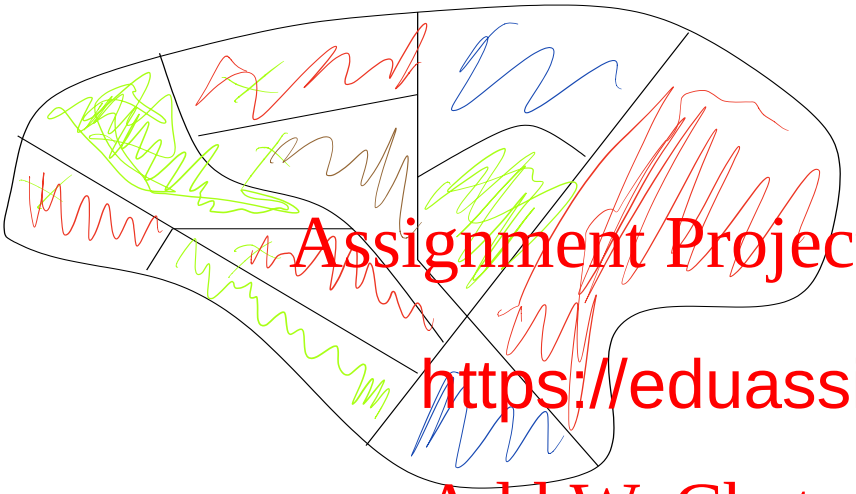
Clearly, the graph is constructed in polynomial time.

We need to show that G has a vertex cover of size $2m + n$ iff φ is satisfiable.

$\boxed{\Rightarrow}$ To be a cover of size $2m + n$, S inc ariable gadget" node and exactly two "clause gadget" node nes a truth assignment t, as it include ly one of $_i$ $_i$ oreover, this t makes each clause true: The ℓ_{kj} that is not in the cover S is set to true.

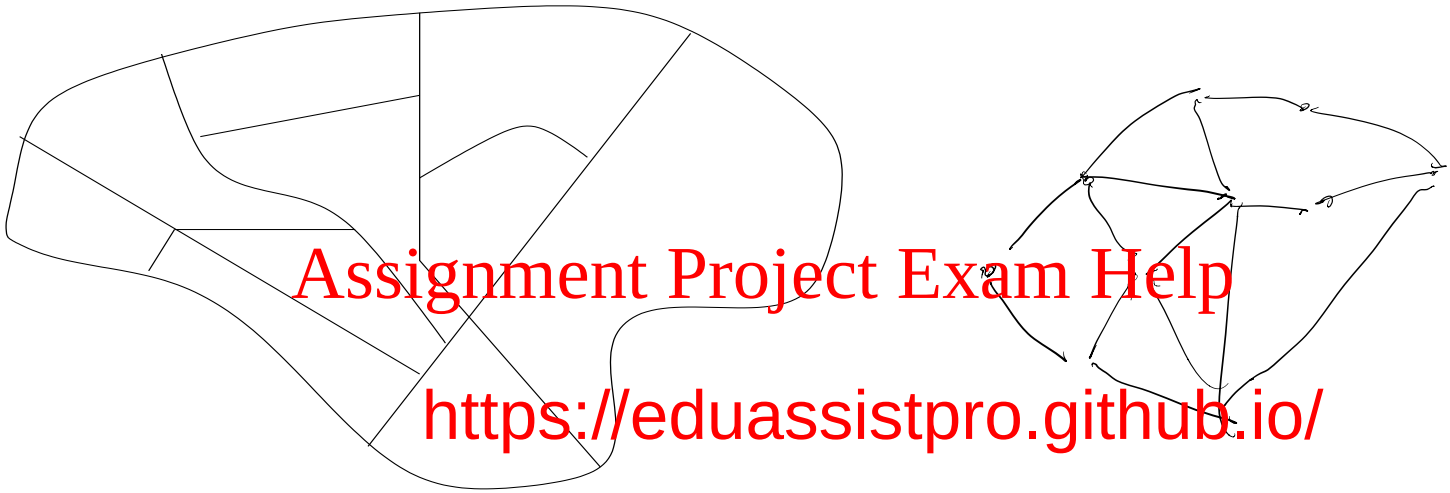
$\boxed{\Leftarrow}$ Let t satisfy φ. That gives us n nodes for each i, either x_i or $\neg x_i$) in the cover.

Now for each clause C_k, t is true with assignment t. Add to the c S the tw ing to the two other literals in C_k. The result is a cover of the requi $n + 2m$.



How few colours suffice to colour this map?

No two countries that share a border can have the same colour.



Four colours suffice in the plane! (Appel and Haken, 1

Tractability of n-colouring:

- 1-colouring is trivial.
- 2-colouring is easy.
- 3-colouring is NP-complete.
- 4-colouring is trivial.

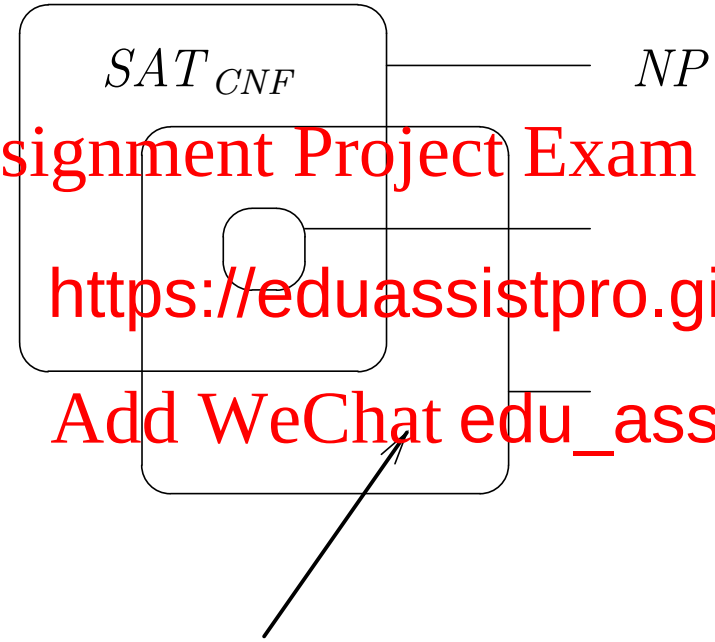
With deterministic machines, reversing the yes/no outcomes decides the complementary problem.

Using a nondeterministic machine to guess-and-check, it is not clear that this property should hold!

For example, how does one prove that a propositional formula is unsatisfiable?

It is not known whether NP is closed under complement. (Of course, if P were equal to NP then it would be.)

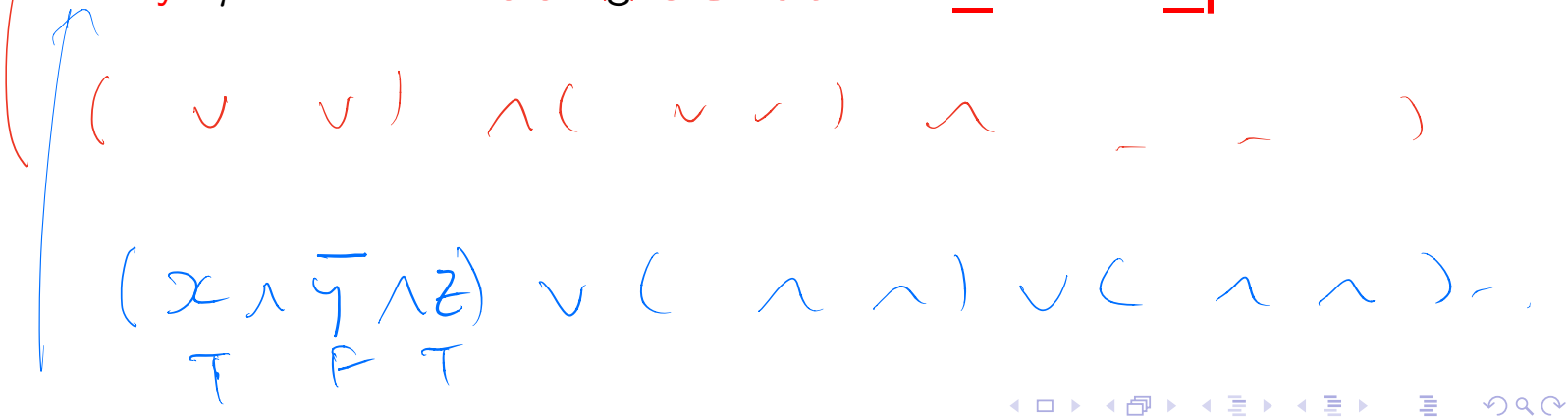
The class of languages *co-NP* is those that are complements of the NP languages.



The dual of *SAT*, i.e. that is, the problem for propositional formulas in DNF, is in

Satisfiability: φ is tru

Validity: φ is true for all assignments of variables.



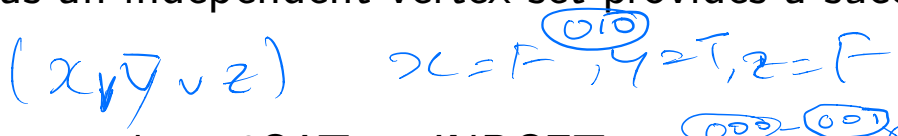
Given an undirected graph $G = (V, E)$, an independent (vertex) set is a set $S \subseteq V$ satisfying $S^2 \cap E = \emptyset$, no edges in induced subgraph of S.

Decision problem: Does a t set of size k?

$$INDSET = \left\{ \langle G, k \rangle \;\middle|\; \begin{array}{l} G \text{ is an un} \\ \text{an indepe} \end{array} \quad k \right\}$$

Let us show that *INDSET* is *NP*-complete.

INDSET is in *NP*, as an independent vertex set provides a succinct certificate.



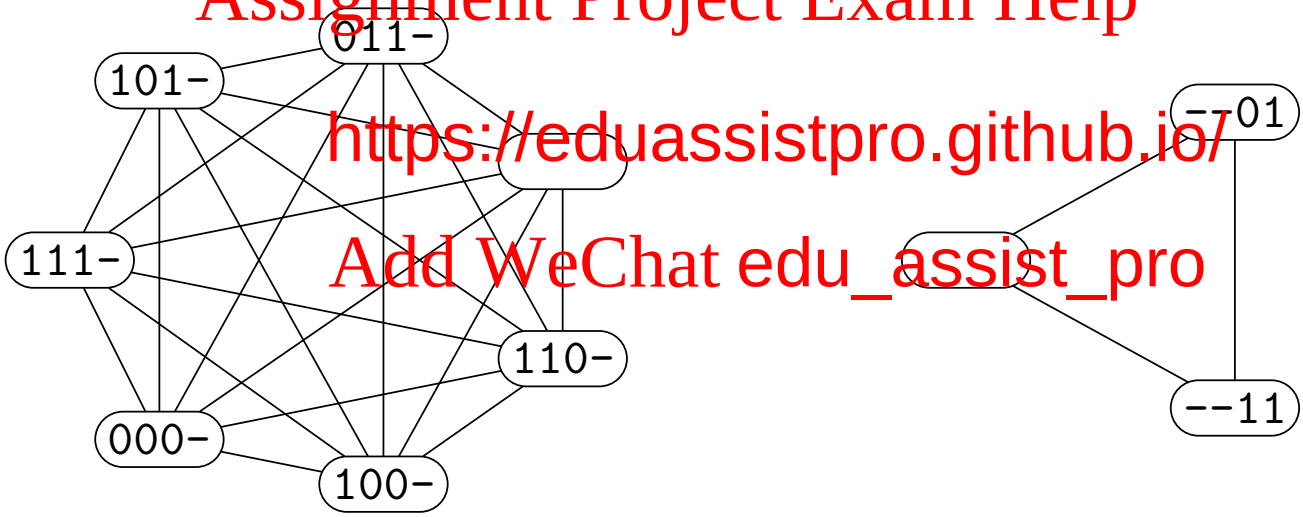
To show *NP*-hardness, reduce *3SAT* to *INDSET*.

Each size-m clause ed to a subgraph of $2^m - 1$ node h assignment that satisfies c.

There is an edge between every pair of nodes in w ated truth assignments are somehow incompatible.

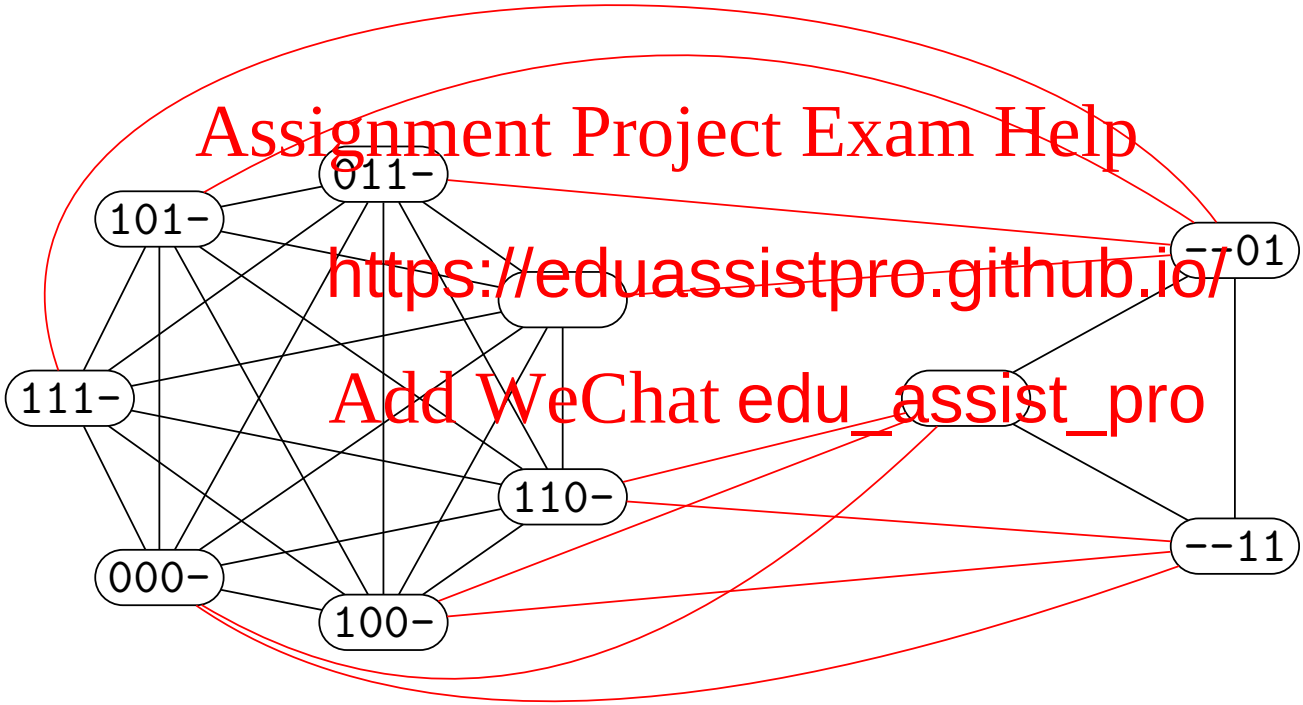
That is, for some variable v in both partial truth assignments, one node assigns v true, while the other node assigns v false.

For $(x_1 \vee \neg x_2 \vee x_3) \wedge (x_3 \vee x_4)$, form two subgraphs. Clearly, each subgraph is (internally) complete.



$x_1 \vee \neg x_2 \vee x_3$ $\quad$ $x_3 \vee x_4$

Now add remaining links, between incompatible assignments:



$x_1 \vee \neg x_2 \vee x_3$ $x_3 \vee x_4$

The reduction takes a k-clause formula φ and produces the *INDSET*-instance $\langle G, k \rangle$.

Since, in particular, each clause has at most 3 literals, the computation of graph G is polynomial.

Since each of the k subgraphs is complete, a G has at most k nodes.

Indeed, an independent set of size k must correspond to a satisfying assignment for φ.

Question you can try at this

Problem 7.23

We already saw that

$$SUBSET\text{-}SUM = \{\langle S, t\rangle \mid \Sigma_{x\in A} x = t \text{ for some } A \subseteq S\}$$

is in NP.

Let us reduce $3SAT$

Say φ has n variable k

We construct a set of $2n + 2k$ numbers a

$$t = \underbrace{1\ 1\ 1 \cdots 1}_{n}\underbrace{3\ 3\ 3 \cdots 3}_{k}$$

The numbers in the set are chosen in a clever way, which we demonstrate by example.

$(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_3)$ yields the numbers shown in the table.

Numbers are written in decimal, or at least base 6, notation.

This set of $2n + 2k$ produced in polynomial ti shown here, it's a multiset, but we can easily add extra columns to make it a set.)

Each "clause column" (upper right part of table) must have one, two, or three 1s (since derived from 3-CNF formula).

		1	2	3	c_1	c_2
x_1		1	0	0	1	0
$\neg x_1$		1	0	0	0	1
x_2		0	1	0	0	1
		0	1	0	1	0
		0	0	1	1	1
			0	1	0	0
			0	0	1	0
		0	0	0	1	0
c_2		0	0	0	0	1
		0	0	0	0	1
t		1	1	1	3	3

Zero 1s in a "clause column" in a subset corresponds to the clause not being satisfied; anything else means the clause is satisfied.

The clause rows (bottom table) are simply fillers tha clause column with one or more 1s to sum to 3.

Clearly, a subset summing to t corresponds to a satisfying truth assignment for the formula.

	1	2	3	c_1	c_2
x_1	1	0	0	1	0
$\neg x_1$	1	0	0	0	1
x_2	0	1	0	0	1
	0	1	0	1	0
	0	0	1	1	1
		0	1	0	0
		0	0	1	0
		0	0	1	0
c_2	0	0	0	0	1
	0	0	0	0	1
t	1	1	1	3	3

It is tempting to suggest that *SUBSET-SUM* can be solved in polynomial time by a table-filling method.

Example: Given $S = \{2, 5, 8, 9\}$ and target sum 13, we fill in a 13

The ith column gives th

constructed using the first i elements of the

A sparse representation, growing a list, and stopping as soon as we find 13, would be more practical, for example: [0], [0,2], [0,2,5,7], [0,2,5,7,8,10,13,15].

Sum	2	5	8	9
0	T	T	T	T
1	F	F		
2	T	T		
3	F	F		
4	F	F		
5	F	T		
6	F	F		
7	F	T		
⋮				
13	F	F		

However, we require that numerical methods involve "reasonable" representation of numbers, such as decimal (or even binary).

The table above is polynomial in the size n of the set. However, it is not polynomial in the si t, which would only be $O(\log t)$. Unless the tion, the table is in fact exponential in the size of in

Nevertheless, intractability of *SUBSET-SUM* depends strongly on the fact that large numbers are allowed. For sets of "small" numbers, the problem is tractable.

Question you can try at this

Problem 7.17

Recall the Hamiltonian path problem $HAMPATH = \{\langle G, s, t\rangle \mid G \text{ is a directed graph with a Hamiltonian path from } s \text{ to } t\}$

where a Hamiltonian path is

HAMPATH is in *NP* since a Hamiltonian path is checked in polynomial time.

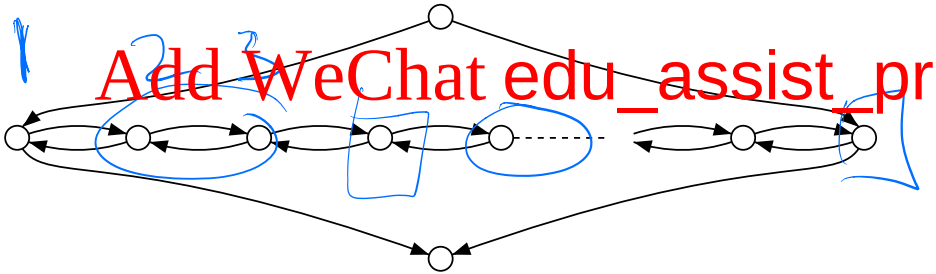
By reduction from *3SAT*, we now establish that *HAMPATH* is *NP-complete*

Assume we have a *3-CNF* formula φ with k clauses.

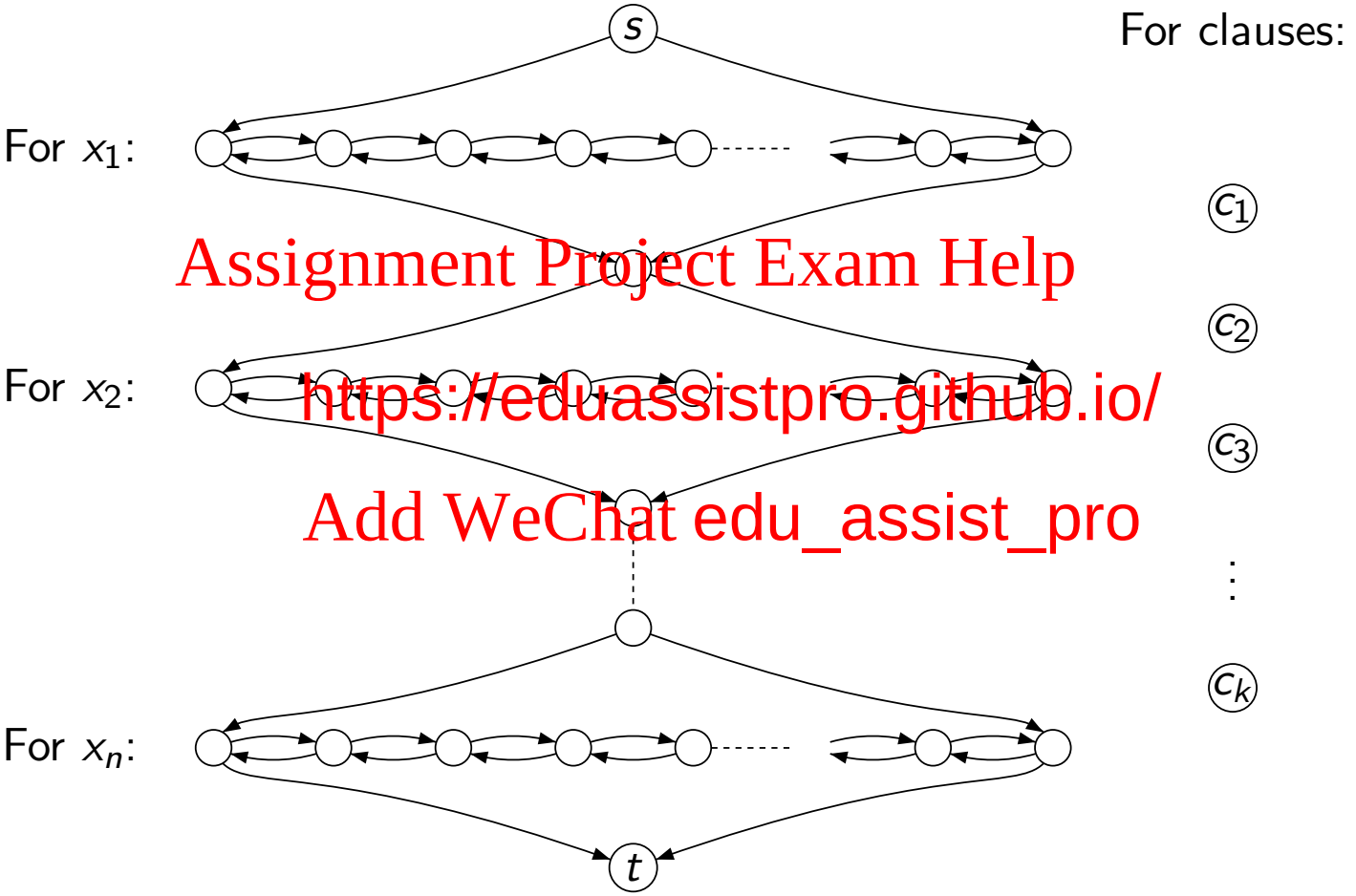
We need to construct (in polynomial time) a graph G in which a Hamiltonian path from s to t exists iff φ is satisfiable.

$$\varphi = (a_1 \vee b_1 \vee c_1) \wedge (a_2 \vee b_2 \vee c_2) \wedge \ldots \wedge (a_k \vee b_k \vee c_k)$$

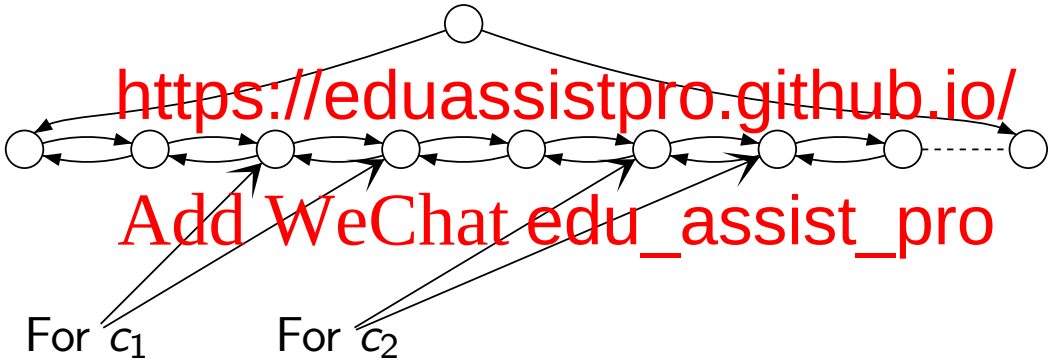
For each variable x_i e this:



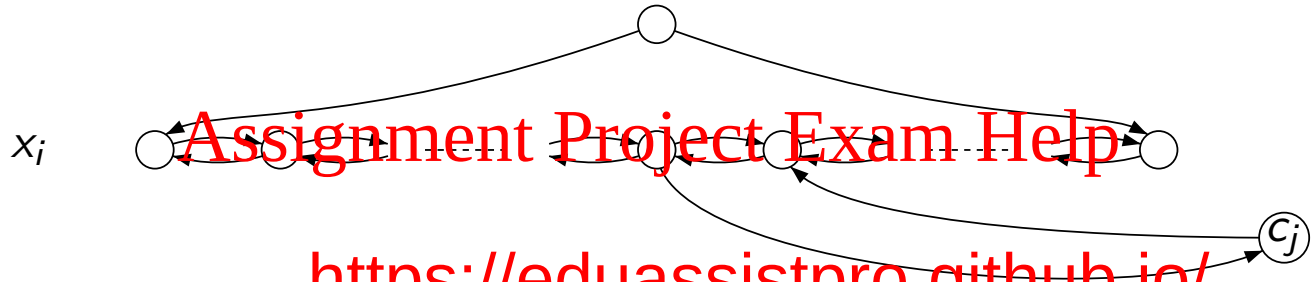
Apart from the nodes on the perimeter, which are shared between gadgets, there are $3k + 1$ nodes, two adjacent nodes per clause, with a separator node either side of the pair. We also construct k separate nodes, one per clause.



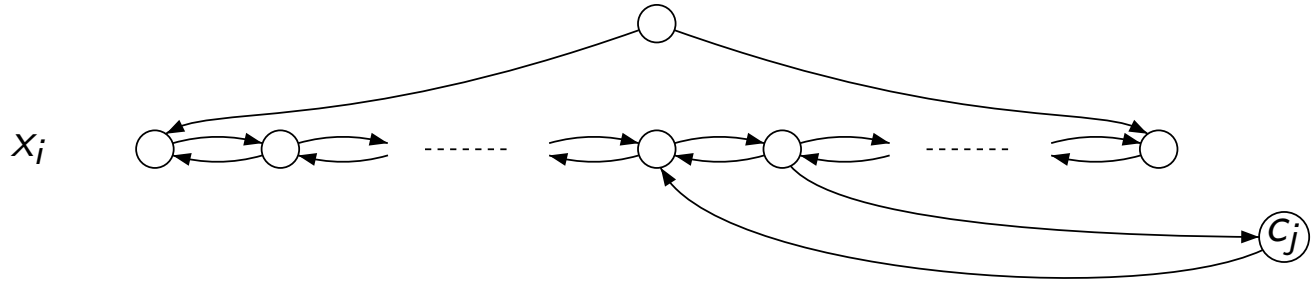
The chain of $3k + 1$ nodes comprises one pair per clause, plus separator nodes on either side:



Consider x_i's chain. If x_i occurs in c_j, we add two edges:

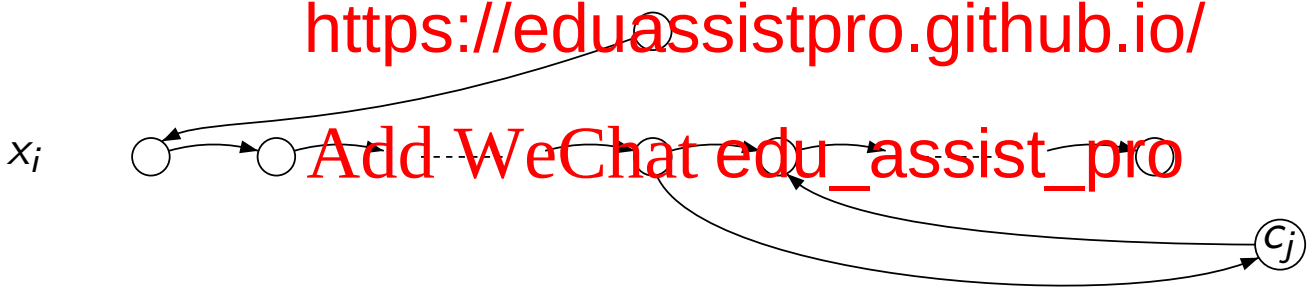


If $\neg x_i$ occurs in c_j, the linking edges are reverse



The idea is that a traversal down through the sequence of diamonds corresponds to a truth assignment to the variables.

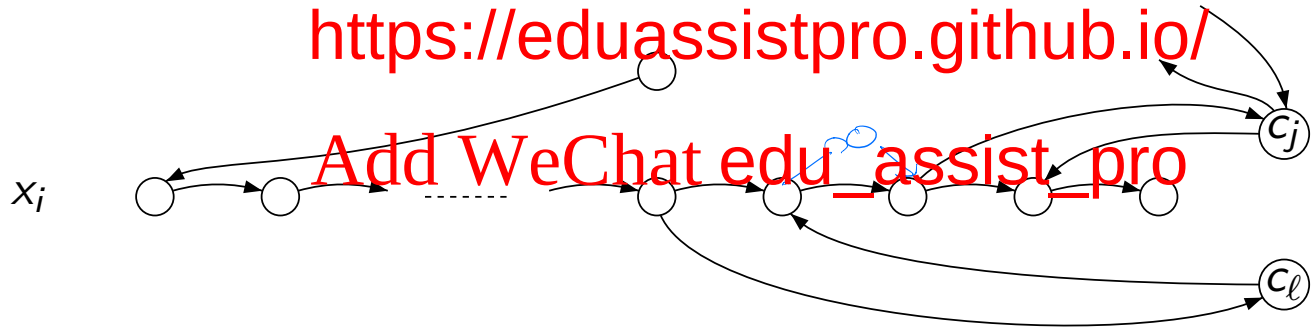
Walking left-to-right through the i^{th} chain corresponds to setting x_i to true.



On the other hand, walking right-to-left through the i^{th} chain corresponds to setting $\neg x_i$ to true.

Given a satisfying truth assignment, there is clearly a Hamiltonian path through the graph.

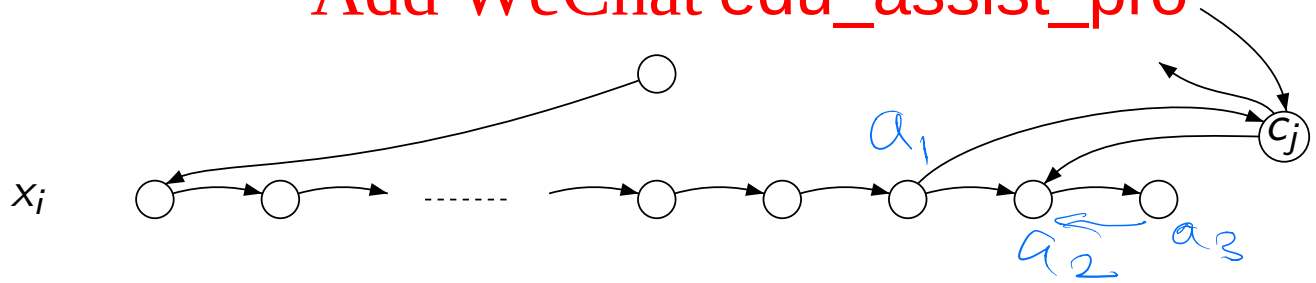
All clause nodes c_j can be reached, but going from a particular chain node to c_j is optional.



A single x_i may make several different clauses true.

Conversely, every Hamiltonian path through the graph from s to t determines a satisfying truth assignment.

If a path takes us from a chain node to a clause node, we must return to the neighbouring (pair) node in the chain. Suppose we don't: we exit from a_1 to clause node c_j. Then the next node in the chain from a_1, a_2, is cut off. Depending on its sta ints are a_1, c_j or the next node in the chain, t available as entry points, as they lead elsewhere; node a_3 i int.



So the graph traversal must correspond to a satisfying truth assignment.

The graph is produced ('computed') in polynomial time.

Questions you can try at this point:

Problem 7.21
Problem 7.27
Problem 7.28
Problem 7.29
Problem 7.30
Problem 7.31
Problem 7.37
Problem 7.43

Our definitions of (mapping) reduction and of NP apply to decision problems—they are phrased in terms of language membership.

But often we are interested in more than a yes/no answer—we want a witness.

Is that substantially harder than getting a yes/no?

In the context of optimization problems, ute of length no more than k?", but "what is the shortest route?"

Is this optimization problem substantially harder than getting a yes/no?

It turns out that, if $P = NP$ then we can produce witnesses for $L \in NP$ with only a polynomial-ti

The next slide exemplifie rhead indeed is polynomial.

If M were a polytime decider for SAT, formula φ has n variables, and if $M(\varphi) =$ 'yes', then a satisfying assignment is found thus:

$t := []$

For $i \in [1, n]$:

If $M(t_i^0(\varphi)) =$ 'yes':

$\varphi := t_i^0(\varphi)$

$t := t_i^0 \circ t$

else

$\varphi := t_i^1(\varphi)$

$t := t_i^1 \circ t$

Output t

Again, if $P = NP$ then the typical "optimization" problem is not substantially harder than its decision version.

For example, given a graph G with n nodes, we can find the size of its smallest vertex cover by invoking the decider for *VERTEX-COVER* up to $\log_2 n$ times.

Similarly the decision pro

$$MINVC = \left\{ \langle G, k \rangle \;\middle|\; \begin{array}{l} G \text{ is a gra} \\ \text{vertex cover has size } k \end{array} \text{ minim} \right\}$$

can be decided by running the deterministic decider for *VERTEX-COVER* on the two instances $\langle G, k-1 \rangle$ and $\langle G, k \rangle$.

Unclear whether *MINVC* is in *NP*, in *coNP*, (or neither).

Question you can try at this

Problem 7.46

Review of this block

We saw how to reduce *SAT* to several different *NP*-hard problems, both directly and indirectly (via other *NP*-hard problems).

We explored a little of the difference between a decision problem and an "answer" problem.

What's next

Randomised algorithms: probabilistic TMs.

Space complexity: look at the beginning of Chapte pser text.

Consider a Turing machine that has the ability to make random choices as it computes.

A mental model: The mac dom bits, initialised freshly for

Each nondeterministic step is a coin toss legal next moves, of equal probability. (There is no loss of generality in co ring just two moves.)

A deterministic Turing machine either accepts or rejects its input, with the same outcome every time the machine is run on the same input.

A nondeterministic machine shares this property, though only one accepting branch is need

In contrast, a probabilisti ith some probability: unless this probability is 0 or 1, the result ca runs.

For example, it may take strings in $\{0,1\}^*$ as input, and accept strings beginning with 0 with probability 2^{-n}, where n is the length of the string.

Let M be a probabilistic Turing machine (PTM) and w be an input string.

When running M on w, each computation branch b has probability $Pr[b] = 2^{-k}$, where k is the number of coin-tosses on branch b.

The probability that

$$Pr[M \text{ accepts } w] =$$

$$Pr[M \text{ rejects } w] = 1 - Pr[M \text{ accepts } w].$$

Let $0 \leq \varepsilon < \frac{1}{2}$.

M recognises L with error probability ε iff

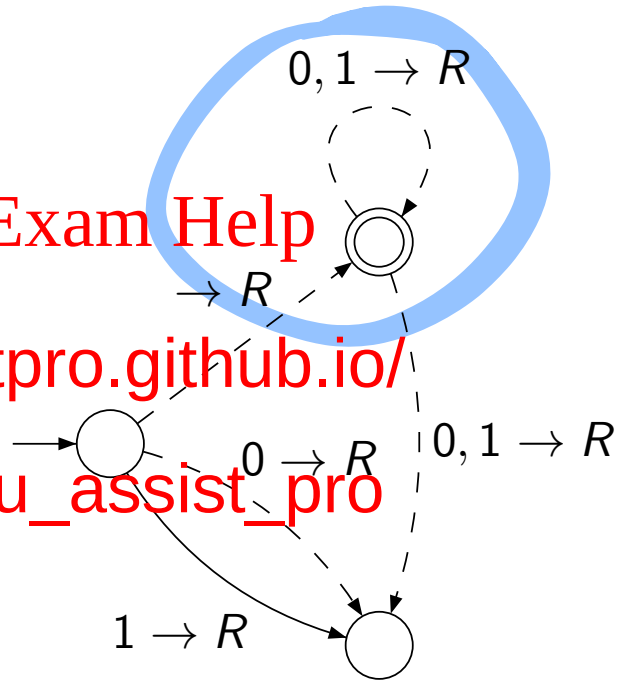
1. If $w \notin L$ then $Pr[M \text{ accepts } w] \leq \varepsilon$
2. If $w \in L$ then $Pr[M \text{ accepts } w] \geq 1 - \varepsilon$

The error probability may depend on the length n of input, as happened with the example PTM above.

Here is a PTM for the language mentioned before.

The machine accepts each binary string starting with 0 with pr
where n is the length of the
other string is rejected.

A dashed edge indicates a transition step that has probability $1/2$.



Language L is in BPP iff L is decided in polynomial time by some PTM M, with an error probability of $1/3$.

Note that M must ruome of coin-tosses along the way

That is, the probabilistic computation "tree" musnt (across all branches) which is polynomially bounded.

The choice of $1/3$ in the definition of BPP is rather arbitrary. It's simply a value strictly between 0 and $1/2$, indeed the "simplest" fraction in that range.

This follows from the mial-time effort, an error probability can be made exponentially sma

Sipser gives a proof, and we shall see an example shortly.

Language L is in RP iff L is recognised in polynomial time by some PTM M, with a one-sided error probability, as follows:

1. If $w \notin L$ then P
2. If $w \in L$ then P

RP stands for "randomised polynomial".

For such machines, a 'yes'/'accept' answer can be trusted.

Again, for $L \in RP$, when deciding membership of L, repeated runs of M on the same input will give independent, equally reliable, results.

With an error probability [illegible] g a wrong 'no' answer from 100 consecu

Success is so close to certainty that, for our purposes, it i [illegible] ardware failure has a much higher likelihood.

$\{\langle G \rangle \mid G$ undirected & contains a triangle $\}$.

Input: An undirected graph $G = (V, E)$.

Output: Yes if G contains a triangle.

Method:

Repeat the following

Select an edge $(x$

Select a node $z \in V$ uniformly at rando $V \setminus \{x, y\}$.

Output 'yes' if $(x, z) \in E$ and (y, z) xit.

Output 'no'.

Clearly the first of the two conditions for RP membership is satisfied: If G contains no triangle then we can only get a 'no'.

As for the second condition, let $n = |V|$ and $m = |E|$. If G has a triangle, the probability that it is selected in a particular iteration is $3/(m(n-2))$.

For k iterations, the probability that the triangle will be found is

For small x, $(1-x)^k \approx e^{-kx}$.

If we pick $k = 1/x$, $1 - e^{-kx} \approx 0.63$.

Thus with $k = m(n-2)/3$, the probability that a graph with a triangle is accepted is well over $1/2$.

The complexity of the algorithm is polynomial.

In particular, $k = O$ ratic in the input size (assuming

The running time for each round is linear in the input siz

Hence the problem is in RP.

The randomized method we described has one-sided error: it never gives a false positive result.

However, it could reject in

But the probability if this happening can be made as sm want.

Running the triangle algorithm 30 times, the likelih lse negative is less than one in a billion.

Theorem: $P \subseteq RP \subseteq NP$.

The first inclusion follows immediately from the definitions.

For the second, a nondeterministic TM N can simulate a probabilistic TM M (with the same ti

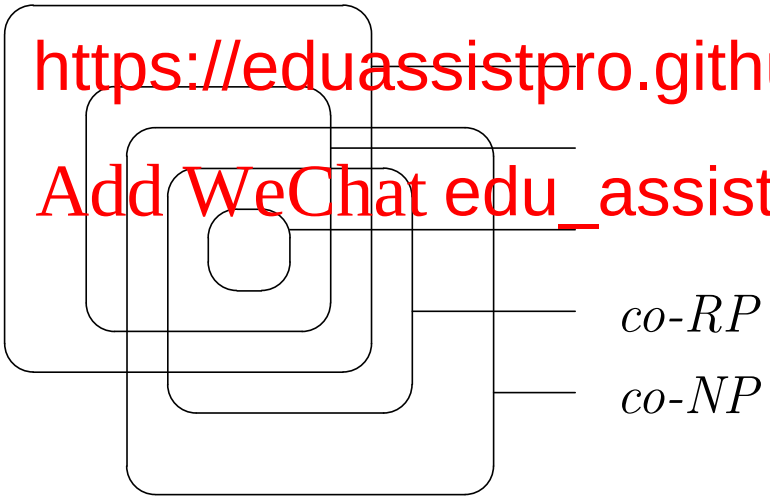
Whenever M uses a random bit, N choo istically between the two values.

If w is in L, then M has a probability of at least $1/2$ of accepting w, so there must be some sequence of choices for N that leads to acceptance.

If w is not in L, no such sequence exists, so N will reject.

RP is one-sided, so it has a natural dual class.

A co-*RP* language has a PTM *M* that accepts members with probability 1, and rejects non-members with probability $\geq 1/2$.



An algebraic circuit is like a Boolean circuit, except inputs are integers and the "gates" perform addition, subtraction, and multiplication.

The polynomial identity testing problem asks whether two algebraic circuits compute the sam

This is an example of a problem for which a fast randomi thm exists.

Unlike the case of the triangle problem discussed above, no efficient deterministic algorithm is currently known for polynomial identity testing.

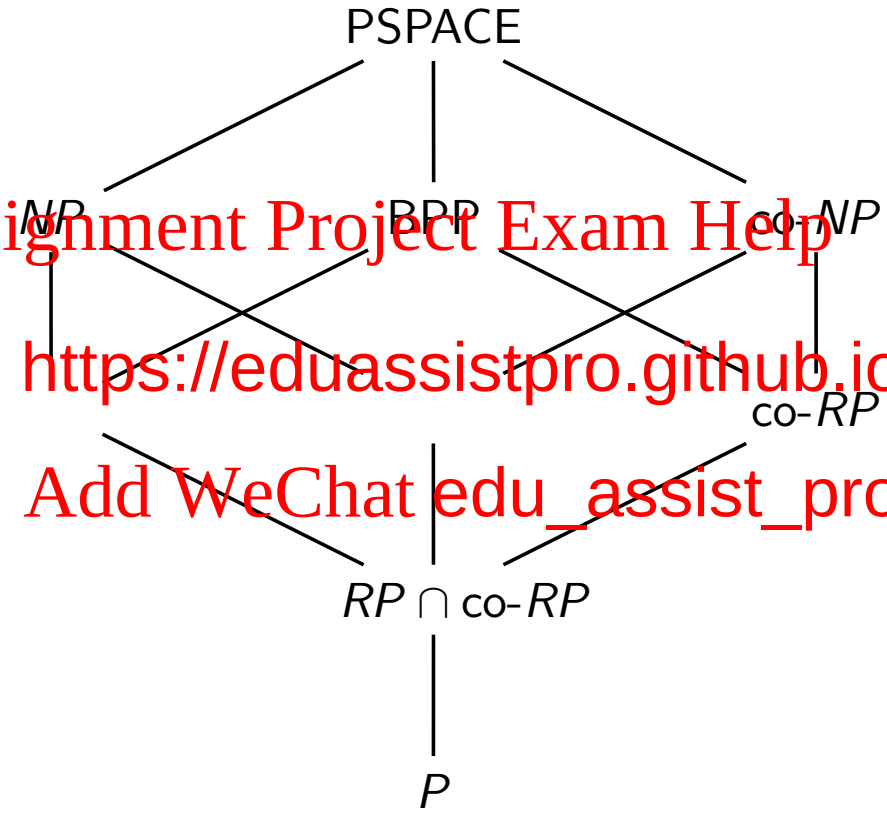
Many interesting questions remain open.

For example, is $RP = \text{co-}RP$?

Is $RP \subseteq NP \cap \text{co-}N$ s")

Is $BPP \subseteq NP$?

Is $BPP = P$? (Believed to be yes)



$L \in RP \cap \text{co-}RP$ means:

(RP) There is a PTM M running in time $p(n)$ such that

- $w \notin L \Rightarrow Pr[M \text{ rejects } w] = 1$
- $w \in L \Rightarrow Pr[M \text{ rejects } w] \geq \frac{1}{2}$

(co-RP) There is a PTM M' running in time $p'(n)$ such that

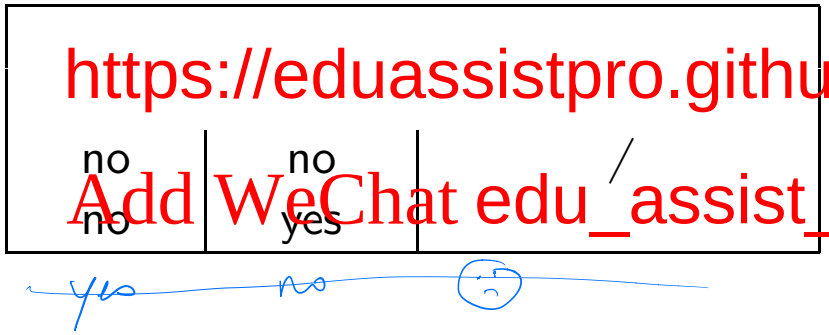
- $w \in L \Rightarrow Pr[M' \text{ acc}] \geq \frac{1}{2}$
- $w \notin L \Rightarrow Pr[M' \text{ rej}] = 1$

In the first case, a 'yes' can be trusted.

In the second case, a 'no' can be trusted.

We can run M and M' on input w in time $p(n) + p'(n)$.

There are three possible outcomes:



The combined PTM is sound (we can trust its answer, if it gives one), and it will answer '?' with probability less than $1/2$.

The randomized methods discussed so far are Monte Carlo algorithms.

These run in (guaranteed) polynomial time and err with some probability less than $1/2$.

RP and co-RP contain pro Monte Carlo algorithms.

A Las Vegas algorithm also uses randomizati

However, it does not have the polynomial-time guarantee for every computation branch. Instead, it runs in expected polynomial time.

The Z in ZPP stands for "zero error".

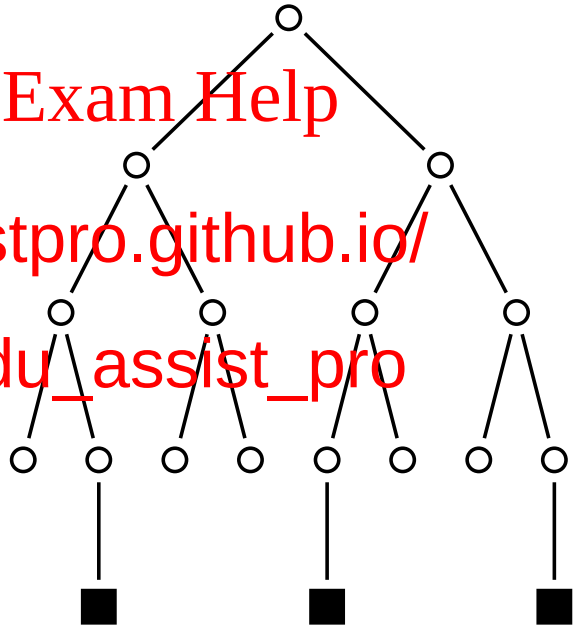
ZPP corresponds to the L give the correct answer, but ha input), the expectati

In the following several slides, we prove the interesti that ZPP $= RP \cap$ co-RP.

Let M be a PTM for a problem in ZPP and consider M's computation tree for input w, of size n. Let $q(n)$ be the expected (polynomial) running time.

At a given polynomial depth, $p(n)$, we may have subtrees of super polynomial depth (sh

Let M' be the PTM which works like M, except (if it has not already) it stops after $p(n)$ steps and answers '?'.



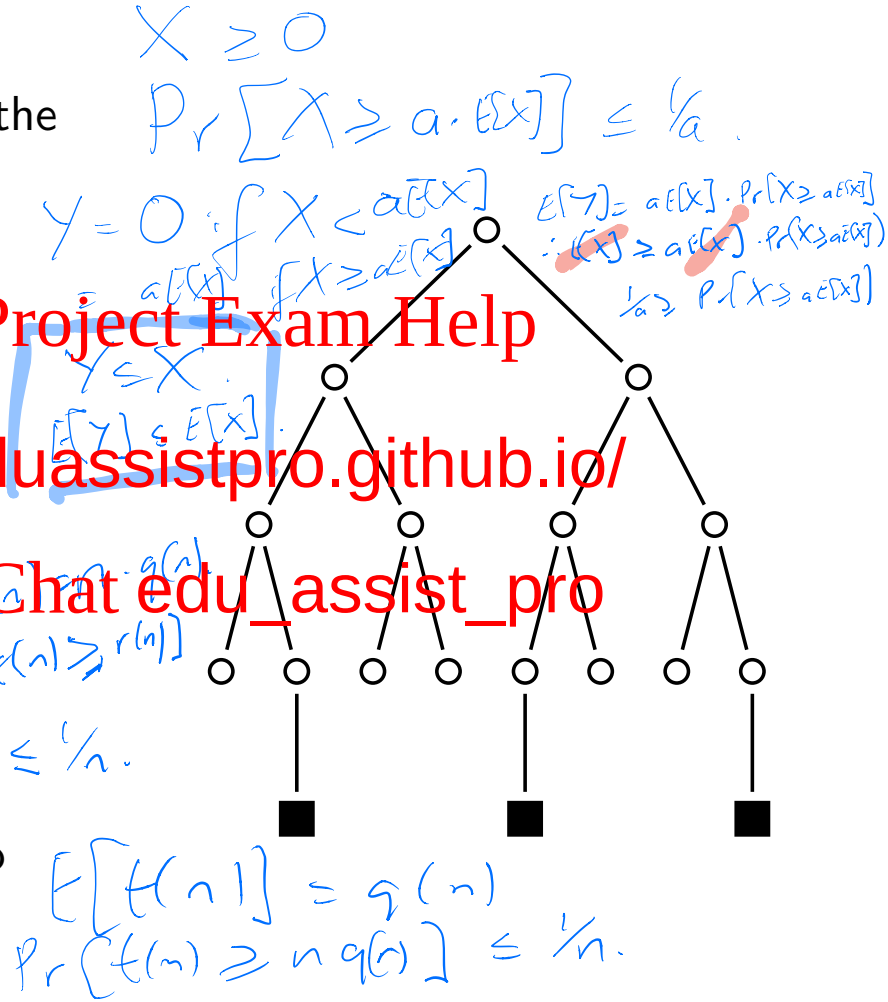
We show that p can be chosen so that the probability π of M' answering '?' can be brought as close to 0 as we want.

Let $s(n)$ be the average length of the super-polynomial paths (so $s(n)$ is super-polynomial). Let π be the probability that M' says '?'.

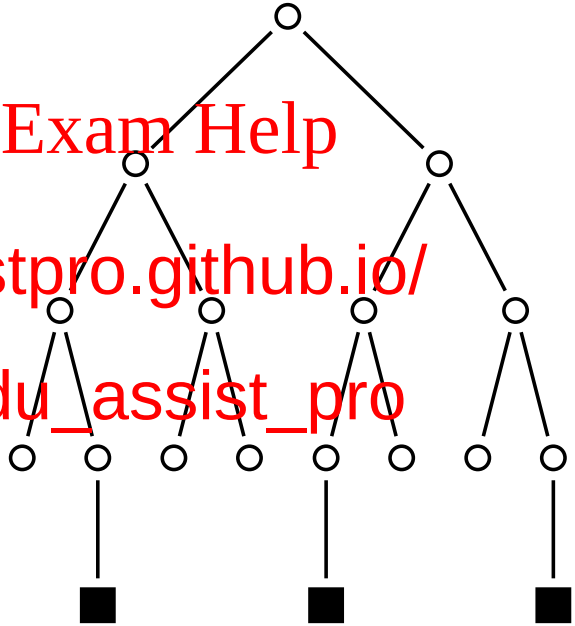
The expected running tim
$(1-\pi)\cdot p(n) + \pi \cdot s$
know is $q(n)$. Isolating π, we have

$$\pi = \frac{q(n) - p(n)}{s(n) - p(n)}$$

Since the denominator is super-polynomial, we can choose $p(n)$ so that π is arbitrarily small.



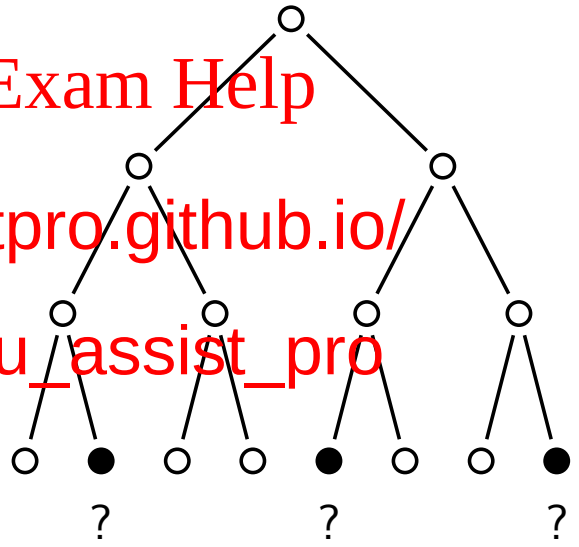
But this means that conditions for membersh $RP \cap \text{co-}RP$.



Now let M be a machine for a problem in $RP \cap \text{co-}RP$. Let $1/k$ be an upper bound on the probability that M says '?' ($k > 1$).

Machine M has polynomial time $r(n)$.

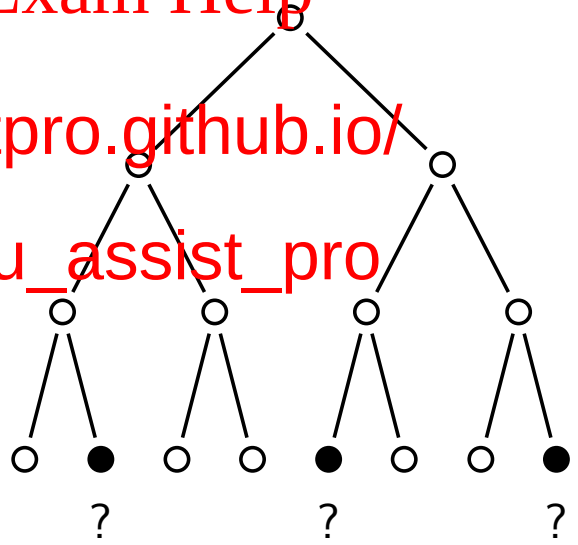
Since $RP \subseteq NP$, we can alwa e the problem brute-force style, without randomization, in time at most $k^{q(n)}$, for some polynomial $q(n)$.



To build a corresponding machine M' that satisfies the conditions for membership of ZPP, we just run M up to $q(n)$ times.

As soon as M provid answer, machine M' and halts.

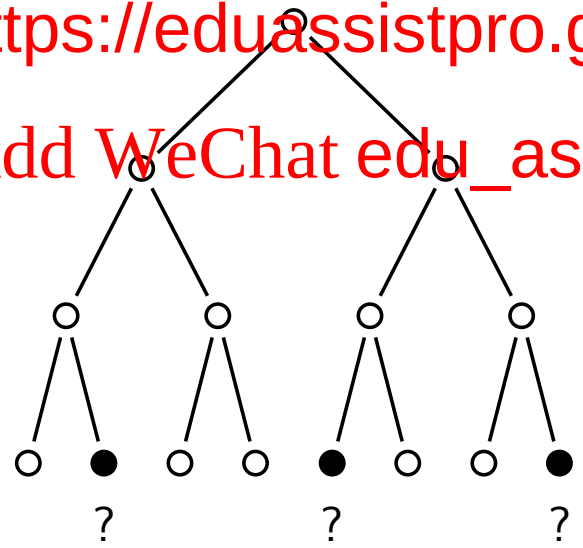
If none of the $q(n)$ runs of M provides an answer, then M' solves the instance the slow way: deterministically brute force.



The expected running time of M' is at most

$$r(n) \cdot q(n) \; + \; \underbrace{k^{-q(n)}}_{Pr[\text{`?'}]} \; \cdot \; k^{q(n)} = r(n)q(n) + 1\,,$$

which is polynomial in n.



Questions you can try at this point:

Problem 10.11
Problem 10.15
Problem 10.19
Problem 10.22

Just like time, space is a relevant resource to study.

The space complexity nct tape positions it visits, as a fu

This leads to a complexity measure that is robust.

It is common to prefix the terms “TIME” and “SPACE” with “N” for “nondeterministic”, and “LOG”, “P”, “EXP”, etc., for complexity:

	N
PSPACE	NPSPACE
EXPTIME	NEXPTIME
EXPSPACE	NEXPSPACE
⋮	

We can define space complexity classes analogous to *TIME* and *NTIME*:

$$SPACE(f(n)) = \left\{ L \;\middle|\; \begin{array}{l} L \text{ is decided by some deterministic} \\ \text{Turing machine in } O(f(n)) \text{ space} \end{array} \right\}$$

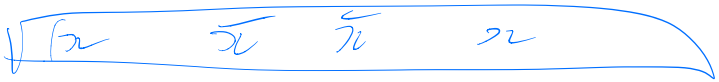
and

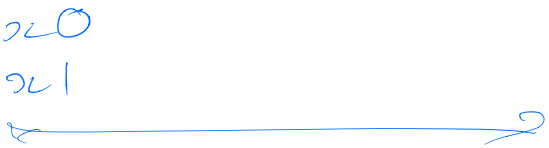
$$NSPACE(f(n)) = \left\{ L \;\middle|\; \text{Turing mach} \quad \text{rministic} \right\}$$

$$\text{) space}$$

In analogy with P and NP,

$$\begin{array}{lcl} \text{PSPACE} & = & \bigcup_k SPACE(n^k) \\ \text{NPSPACE} & = & \bigcup_k NSPACE(n^k) \end{array}$$





SAT can be solved in linear space thus:

We solve *SAT* in exponential time by generating all possible truth assignments, and evalu

Apart from space for the for one truth assignment at a time, the one currently being evaluat

Overwriting each truth assignment with the next, in the same space, means that space consumption is linear in the size of input.

Clearly $P \subseteq PSPACE$: If a Turing machine makes only a polynomial number of moves, then it cannot visit a super-polynomial number of cells.

On the other hand, a polynomial-space-bounded TM can only make an *exponential* number o n.

For a machine with s states, γ symbols in t t most $f(n)$ tape cells visited, the number of configurations is $s\gamma^{f(n)}$. A machine that halts cannot repeat a configuration!

From this, it follows that PSPACE $\subseteq$ EXPTIME, which is $\bigcup_k TIME(2^{n^k})$.

Question you can try at this

Exercise 8.4

Perhaps surprisingly, PSPACE = NPSPACE.

To demonstrate this, we show that a deterministic TM can simulate an NTM in a fairly small amou

Theorem: For a function $f : \mathbb{N} \to \mathbb{N}$ wi

$NSPACE(f(n)) \subseteq S$

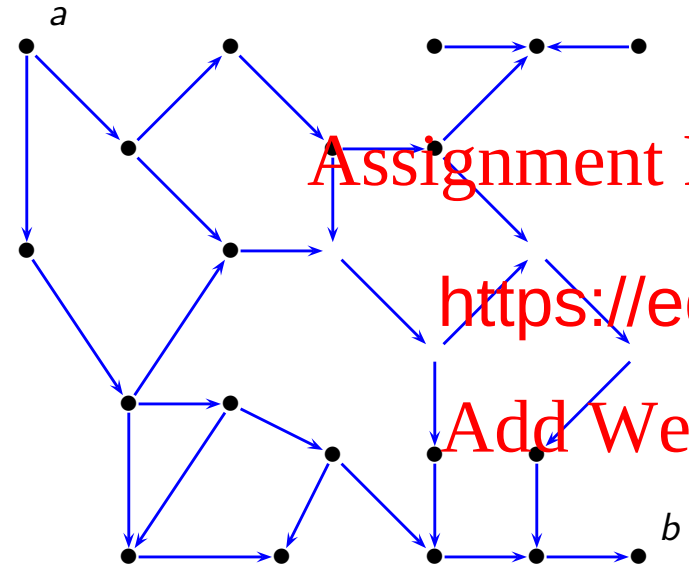
A central tool for proving space-complexity results.

Path Existence

Given a graph G, n ber e can ask "is node b reachable from node teps long?"

We can decide this in $O(\log^2 n)$ spa exponential time.

We apply this to configurations of a Turing machine: the combination of tape contents, head location, and state.



In this directed graph, we can go from a to b in 13 steps if and only if:

There exists some node x for which we can:

1. Go from a to x in 7 steps
2. Go from x to b in 6 steps.

So we t x.

The ru er be deeper than $\log n$ (and each frame holds little more than the numerical path-length bound).

Remember, we can reuse the space when we 'pop' the stack frame.

This insight leads us to the following

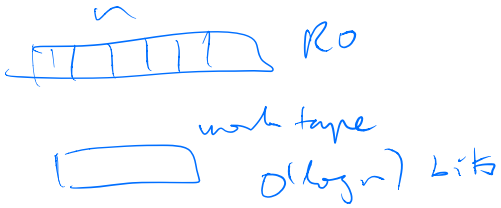
Theorem: For a function $f : \mathbb{N} \to \mathbb{N}$ with $f(n) \geq \log n$,

$$NSPACE(f(n)) \subseteq SPACE(f^2(n))$$

Suppose N is an NTM w

We construct a deterministic TM M to si

Machine M uses a recursive procedure canyield, which takes configurations c_1 and c_2 together with a number t and outputs *accept* iff c_2 can be obtained from c_1 in t steps or fewer, according to the rules of N.



Assuming

s	states in Q
γ	symbols in Γ
$f(n)$	cells touched on tape

there at most $s \cdot f(n$

That is, for suitable constant d, no more tha $d^{f(n)}$ onfigurations. (We can achieve this by treating the presence of the tape he xtra symbol.)

(To achieve the $f(n) \geq \log n$ condition, we treat the input as read only, so that the number of configurations is $n2^{O(f(n))}$: provided $f(n) \geq logn$, this is $2^{O(f(n))}$. Without this, we require $f(n) \geq n$, just to account for the input.)

canyield(c_1, c_2, t) procedure

1. If $t = 1$, accept if $c_1 = c_2$ or c_1 yields c_2 in one step, otherwise reject.
2. For each configuration c,
3. Run canyield(
4. Run canyield(
5. If both accept then accept, otherwise reject.

Now we can define M, which simulates N.

To have a single accept configuration, M can alter N slightly so that N clears its tape and rewinds upon accepting.

Let c_s be the start configuration of N on w, and let c_a be the accept configuration.

M simply runs canyield$(c_s, c_a, 2^{df(n)})$ for some suitable constant d.

Each level of recursion uses $O(f(n))$ additional space, and the maximum depth of recursion is)), for $f(n) \geq \log n$.

Therefore the overall machine M simula ^{2} n)) space.

Therefore for any function $f : \mathbb{N} \to \mathbb{N}$ wi ,

$$NSPACE(f(n)) \subseteq SPACE(f^2(n))$$

(We do not know $f(n)$, but can run canyield with increasing values to determine this.)

Since $NP \subseteq \text{NPSPACE} = \text{PSPACE}$, we have

$P \subseteq N$ XPTIME

We don't know which of th

But the Time Hierarchy Theorem says that IME, so at least one of the inclusions must be strict.

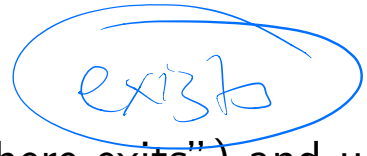
Language B is PSPACE-hard if $A \leq_P B$ for all $A \in$ PSPACE.

This is a polynomial-time reduction; to be meaningful, we want the reduction to perform a fairly small amount of computation.

B is PSPACE-complete if

1. $B \in$ PSPACE, and
2. B is PSPACE-hard.

A fundamental PSPACE-complete problem is concerned with the truth of Quantified Boolean Formulas (QBFs).



We can add existential ($\exists$, "there exits") and universal ($\forall$, "for all') quantification to a propositional formula. For example,

If each variable is bound by a quantifier, the formula is closed, or fully quantified.

We can expand these quantifiers thus:

$$\begin{aligned} \exists x[E] &= E_0 \vee E_1 \\ \forall x[E] &= E_0 \wedge E_1 \end{aligned}$$

where E_0 is E with x replaced by *false* and E_1 is E with x replaced by *true*.

A fully quantified formula

$$TQBF = \{\langle \varphi \rangle \mid \varphi \text{ is a true } \}$$

Order matters:

- $\forall x \exists y \left[(x \vee y) \wedge (\neg x \vee \neg y) \right] = \text{true}$
- $\exists y \forall x \left[(x \vee y) \wedge (\neg x \vee \neg y) \right] = \text{false}$

Here is a linear-space decision procedure for $TQBF$:

1. If φ has no quantifiers, evaluate it, and accept iff φ is true.
2. If $\varphi = \exists x[E]$, evaluate E_0 recursively, then E_1 recursively. Reject iff both lead to rejectio
3. If $\varphi = \forall x[E]$, ev ecursively. Accept iff both lead to acceptance (true).

If there are n variables, the depth of recursion is n, and each activation record only needs to hold a single variable's value.

Hence $TQBF \in$ PSPACE.

We need to show that *TQBF* is PSPACE-hard.

Let A be a language decided in space n^k for some k, by TM M. We describe a polynomial-time reduction to *TQBF*.

We want to produce a form accepts w.

Importantly, the formula, as well as the time to produ be polynomial in the input size.

The ideas from the proof that *SAT* is *NP*-complete help us capture invariants about tape and so on, but we cannot afford to use a family of variables per time-step, as that may be an exponential number.

Instead, reachability of configurations guides us, similarly to the proof of Savitch's Theorem.

Configuration represe y_{jA}, $0 \leq j \leq n^k$, that assert that position i.e. a clever overloading of notation). Each configuration is pol size.

Notation convenience: If $I = \{x_1, \ldots, x$ ds for $\exists x_1 \cdots \exists x_m$.

Similarly for $\forall I$.

The formula φ will have the form

$$\exists I_s \; \exists I_f \; [S \wedge N \wedge F]$$

where

- I_s represents the in
- I_f represents accepting (final) configurat

- S expresses that the simulation starts corre
- N expresses that M can take I_s to I_f in less than i steps.
- F expresses that the simulation finishes correctly, that is, I_f is accepting.

S is a conjunction ($\wedge$) of literals in I_s.

S has y_{jA} if the jth position of the initial configuration (on input w) is A.

It has $\neg y_{jA}$ otherwis

Since the machine runs in polynomial space, this is a pol al-time reduction.

F can be written as a disjunction of the variables y_{jA}, in I_f, for which A represents not a symbol, but an accepting state.

N is constructed recursively.

This recursion doubles the number of "steps" considered, each time only adding only $O(n^k)$ symbols to the formula per level of recursion.

Pairs of configurations: If I has variables y_{jA} as variables z_{jA}, then we let $I = J$ stand for

$$\bigwedge_{j,A} (y_{jA} \Leftrightarrow z_{jA})$$

We will construct formulas $N_i(I, J)$, for $i = 1, 2, 4, 8, \ldots$ to express that J can be reached from I in i or fewer steps (of machine M).

$N_1(I, J)$ asserts that either $I = J$, or M can take I to J in one step.

This can be expressed by a polynomial-sized formula, generated (automatically) from M's transition function δ. For example, if

$$\delta(q, u) = (p, v, R)$$

we add the conjunct

$$\bigwedge_{i=0}^{n^k-1} \left((y_{iq} \wedge y_{(i+1)u}) \rightarrow \right)$$

That is, for all tape prefixes w and suffixes w', configuration $w\ q\ u\ w'$ goes to $w\ v\ p\ w'$.

(But the formula contains only a constant number of literals per position-transition combination.)

How to construct N_{2i} from N_i? The obvious formula is

$$N \qquad J)] .$$

Unfortunately, this wo ime we halve i. Since M might run for an exponential n , this can result in an exponential size formula, which is infe .

The trick is to apply quantifiers cleverly:

This is one of the most subtle slides in this subject!

$$N_{2i}(I,J) = \exists K \quad I \quad J \left[\left[P(I',J') = I \wedge J' = K \right) \right] \vee N_i(I',J') \Bigg]$$

Recalling that $\neg a \vee b$ is equivalent to a uantified formula says

- If $(I',J') = (I,K)$ then $N_i(I',J')$, and
- If $(I',J') = (K,J)$ then $N_i(I',J')$.

Writing N_{2i} takes only the time to write N_i, plus $O(n^k)$ additional work.

Max number of time steps is 2^{dn^k} where d is some constant. Writing N_{2i} takes the time to write N_i, plus $O(n^k)$ additional work, so the time taken to write $N_{2^{dn^k}}(I_s, I_f)$

All in all, the formula $_{s}$ $_{f}$ ted in polynomial time.

The formula is true iff M accepts w.

Questions you can try at this point:

Exercise 8.6
Problem 8.8
Problem 8.9
Problem 8.11
Problem 8.12

Interestingly, *TQBF* gives us a handle on the complexity of various games (or their winning strategi y noughts-and-crosses t

Such games often have a polynomial-time reducti *TQBF*.

In fact, we can associate a simple game with QBFs:

$$\exists x \forall y \exists z [(x \vee y) \wedge (y \vee z) \wedge (\neg y \vee \neg z)]$$

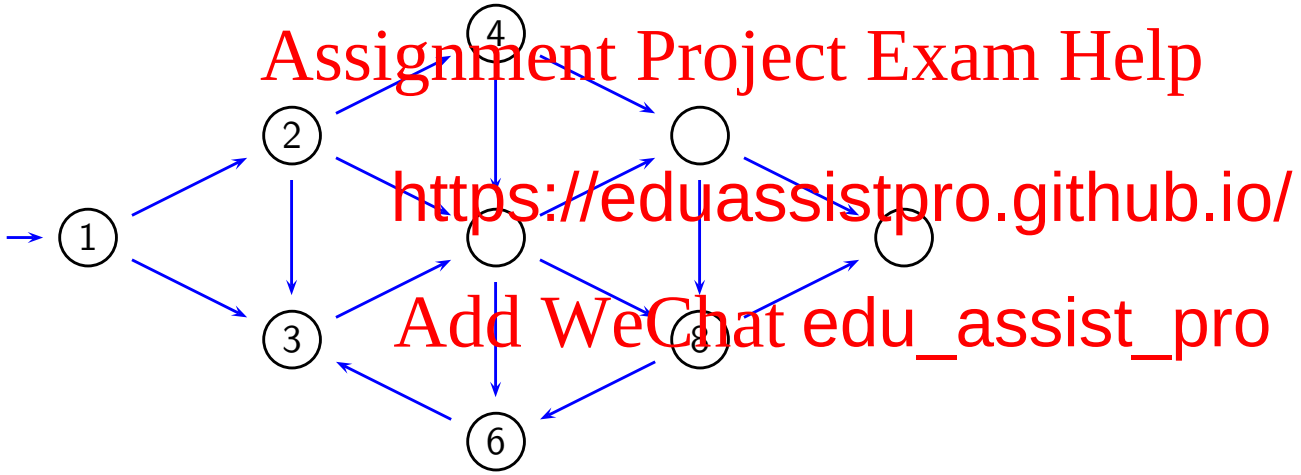
Players E and A tak

E chooses values for the e d wins if the formula (without quantifiers) evaluates to

A chooses values for the universally quantified variables and wins if the formula (without quantifiers) evaluates to *false*.

This game is just *TQBF* in disguise.

The formula game can be reduced to a number of more realistic two-player games in PSPACE, such as Generalised Geography:

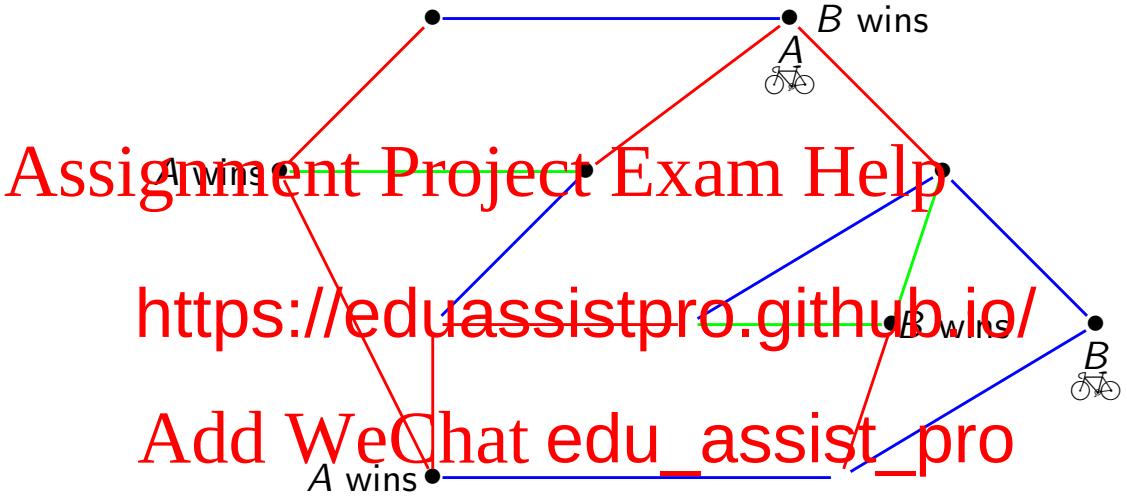


showing that deciding winning strategies for such games is PSPACE-complete.

Questions you can try at this point:

Problem 8.10
Problem 8.14
Problem 8.15
Problem 8.16

Is there a winning strategy for A? Not even in EXPTIME!



The rules: There is a board with towns and roads connecting towns. Each player has a fleet of bicycles, initially placed in towns, and aims to get one bicycle to some "home" town for the player. Roads have colours, and a bicycle can, in one move, travel some number of segments of the same colour, unless blocked by some other bicycle.

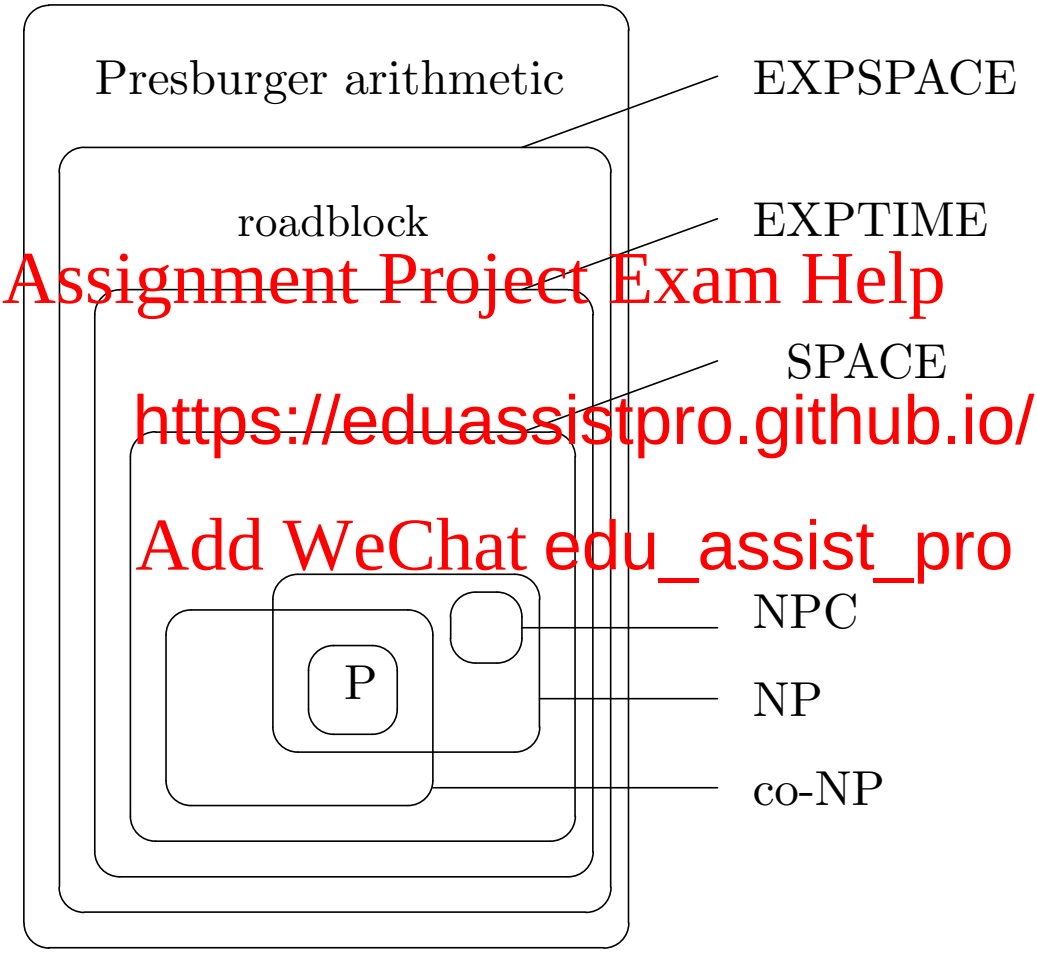
The class 2-EXPTIME is the class of problems solvable in $2^{2^{p(n)}}$ time, deterministically, for some polynomial p.

This generalises to k-EXPTIME.

The theory of natural num urger arithmetic) is decidable

Somewhat easier is the problem of real tion, which is still NEXPTIME-hard.

(Contrary to what we were told in school, integers are more difficult than reals!)



Recap

Reachability of configurations helps us prove that there is only a quadratic space blowup in simulating nondeterminism, and that TQBF is PSPACE-complete.

Preparation

Read through the proof that GENERALIZED GEOGRAPHY is PSPACE-complete.

At first glance, it makes no sense to talk about a Turing machine using space that is less than linear in its input. How do we even store the input!

However, we can refine the Turing machine model by considering that the TM has an input tape that is separate from its working tape.

The input tape is read-only.

Anyone know what that is any more?

If the Turing machine also has a separate write-only o , we call it a transducer.

This separation allows us to consider space complexity in terms of working space only.

Reading "decidable in logarithmic space" to mean "in logarithmic working space", we can define

The focus on logarithmic behaviour is again justified by robustness results: Logarithmic behaviour is preserved amongst under variant forms of the Turing machine.

We previously looked at the language

$$PATH = \left\{ \langle G, s, t \rangle \;\middle|\; \begin{array}{l} G \text{ is a directed graph} \\ \text{with a path from } s \text{ to } t \end{array} \right\}$$

and noticed that $PATH$ is in P.

A nondeterministic TM can decide $PATH$ in $\log$ space.

(Think of it as "guessing" a path from s to t by storing the "current" node and "next" nodes on its working tape after n guesses, where n is the number of nodes in G.)

So $PATH$ is in NL, but we don't know whether it is in L.

In fact, $UPATH$ is in L: For this result, Reingold won best paper in STOC 2005.

Questions you can try at this point:

Exercise 8.1
Exercise 8.5
Exercise 8.7
Problems 8.17 – 8.25

In analogy with NP-completeness, we can try to isolate the core of "most difficult" problems in NL.

We need a concept of reduc time reductions will not be very helpful.

Suppose we reduce A to B and we have a log- cedure for B. Unless the reduction itself is log-space, that do us a log-space decision procedure for A.

A is log-space (mapping) reducible to B, $\boxed{A \leq_L B}$, iff there is some log-space computable function $f : \Sigma^* \to \Sigma^*$ such that for all w,

The function f should be realised by a transduc
write-only tape) and we say that f is a reduction of A to B.

Theorem: If $A \leq_L B$ and $B \in L$ then $A \in L$.

Many of the polynomial-time reductions in previous examples are in fact log-space reductions, or it is possible to find log-space replacements for them.

In complexity theory papers, you will often see that au er to give (the stronger) log-space reductions if they can, eve trying to establish, say, NP-completeness.

B is NL-hard iff every $A \in NL$ is reducible to B in logarithmic space.

B is NL-complete iff

1. $B \in NL$, an
2. B is NL-hard.

Theorem: If $B \in L$ is NL-complete then

Theorem: If B is NL-complete and $B \leq_L C$ for $C \in NL$ then C is NL-complete.

It can be shown that *PATH* is NL-complete.

Given NTM M that d A O n e and input w, we can construct $\langle G, s, t \rangle$ i l irected graph that contains a path from

Nodes = configurations, edges = transitions. Each ated in $O(\log n)$ space.

Since a log-space reducer is a polynomial-time reducer, every language in NL can be reduced, in polynomial time, to $PATH$, which is in P.

Hence we can extend our chain of inclusions to

$$L \subseteq NL \subseteq P \subseteq \text{EXPTIME}$$

It is known that $NL \subsetneq \text{PSPACE}$, so in the cha

$$NL \subseteq P \subseteq NP \subseteq \text{PSPACE}$$

at least one of the inclusions is proper.

The language

$$NE_{RG} = \left\{ \langle V, \Sigma, R, S \rangle \;\middle|\; \begin{array}{l} (V, \Sigma, R, S) \text{ is a regular} \\ \text{grammar with } L(G) \neq \emptyset \end{array} \right\}$$

is in *NL*. Namely, for eac $w \in \Sigma^*$), we can build a directed graph with ode B.

The language generated by the grammar is non-em ath from S to some node C, where $C \to w$ i aving only terminals (alphabet symbols) in its right-hand side.

This is essentially the *PATH* problem, which, as we have seen, can be solved in nondeterministic logarithmic space.

To show NL-hardness, we reduce $PATH$ to NE_{RG}.

Given a $PATH$ instance G S T e regular grammar that has S B for each edge $(A, B) \in G$, plus the rule

This grammar generates $\{\epsilon\}$, the empty strin s a path from S to T, and $\emptyset$ (that is, no strings) if no such path exists.

Questions you can try at thi

Problems 8.26 – 8.34

It follows from Savitch's theorem that *NPSPACE* is closed under complement, as are all the "higher" nondeterministic space classes. But we do not know whether $L = NL$.

The following result, known as the Immerman-Szelepcsényi Theorem, came as a surprise in 1988:

$NL = co$

(For those interested, Sipser ends his Chapter 8 with the proof.) This result won the authors the Gödel prize in 1995.

We can conclude that the emptiness problem for regular grammars also is *NL*-complete, as is the problem of whether, given a graph G and nodes s and t, no path exists in G from s to t.

Recap

The intuition for space is different from time.

Yet we can instead of polytime reductions focus on log-space reductions.

Again, we can model com bility, so NL-completeness of

Further reading

The fact that $NL = co\text{-}NL$ is unexpected: r tbook.