

COMP90057

Advanced Theoretical Computer Science

Complexity

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat [edu_assist_pro](#)
Drawing on material from
Harald Søndergaard & Elena Kelareva

Part 1 (in 2020)

Second (Spring) Semester, 2020
© University of Melbourne

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 behaviour of algorithms; space bounds; methods to deal with intractable problems

Add WeChat [edu_assist_pro](https://eduassistpro.github.io/)

Preparatory reading/viewing

<http://cacm.acm.org/magazines/2009/9/>

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

Assignment Project Exam Help

Symposium on 50 Years o

of Stephen Cook

<https://eduassistpro.github.io/>

<http://www.fields.utoronto.ca/activ>

the Work

P50

<http://www.fields.utoronto.ca/vide>

nt/2774/2019

Add WeChat edu_assist_pro

Big-Oh Notation

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

$g = O(f)$ iff for all $n > n_0 : g(n) \leq c \cdot f(n)$,

for some integers n_0 ie,

<https://eduassistpro.github.io/>

$$\frac{5n^3 + 2n^2}{1+2+\dots+n} \leq \frac{22n+6}{n}$$

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

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

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.

Polynomially Bounded Functions

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 running time asymptotically, as a function of input size.

Add WeChat `edu_assist_pro`

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

Assignment Project Exam Help

Questions you can try at this point:

Exercise 7.1

<https://eduassistpro.github.io/>

Exercise 7.2

Add WeChat edu_assist_pro

Complexity Classes

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

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

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

This focuses on the worst-case time complexity.

Let $t : \mathbb{N} \rightarrow \mathbb{N}$ be a function. The complexity class of languages decided by machines running in $O(t(n))$ time is called $\text{TIME}(t)$. The complexity class of languages decided by machines running in $O(t(n))$ time is called $\text{TIME}(t)$.

$$\text{TIME}(t(n)) = \left\{ L \mid L \text{ is decided by some } M \text{ with } t_M(n) = O(t(n)) \right\}$$

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

Machine Model Affects Complexity

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

perhaps that it's of the form

<https://eduassistpro.github.io/>

The machine then repeatedly scans across its input,
every second 0 and every second 1, checking after each scan
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).

Model Affects Complexity

0000.01 - 1
↑
Turing

A two-tape deterministic machine has lower time complexity.

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

<https://eduassistpro.github.io/>

The two kinds of Turing machines have the same computational power, but they have different complexity properties.

Add WeChat edu_assist_pro

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

The Class P

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 class of deterministic Turing machines in polynomial time, viz.

Add WeChat edu_assist_pro

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

Robustness of P

$$(x^4 + x^3)(x^{12} + 3x^2) \rightarrow x^{16}$$
$$((x^2 + 3x^2)^4 + (x^4 + 3x^2)^3) \rightarrow x^{48}, \dots$$

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

Assignment Project Exam Help

Class P is robust in the se

models are ~~polynomial~~^{tic computational}
~~deterministic~~
machine model will not change the class

Add WeChat edu_assist_pro

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



Polynomial vs Exponential

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

$2^{2n} = (2^2)^n = 4^n \neq O(2^n)$

Assignment Project Exam Help
Here we take “exponential” to mean $\Theta(2^{cn})$ for some constant $c > 0$.

There are functions, such as <https://eduassistpro.github.io/>, which grow faster than every polynomial function, but more slowly than every exponential function.

Add WeChat edu_assist_pro
Hence it is possible that $P \subsetneq NP$, and at the same time, that NP -complete problems can be solved in sub-exponential time.

We describe NP below.

$$\begin{aligned} 4^{4 \log_2 n} &= (2^2)^{4 \log_2 n} \\ &= 2^{8 \log_2 n} > (2^{\log_2 8})^n \\ &= n^8 \end{aligned}$$

Some Problems in P

6, 35

27, 24

The language

$$\text{size } (\langle x, y \rangle) \approx \log x + \log y.$$

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

Assignment Project Exam Help

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

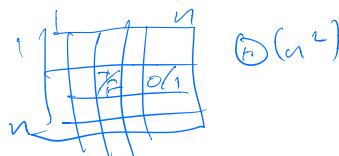
<https://eduassistpro.github.io/>

Likewise, the language

$$\text{size } (\langle n, (u, v), (u, v) \rangle) \approx \log_2 n + m \log n$$
$$\text{PATH} = \left\{ \langle G, s, t \rangle \mid \begin{array}{l} G \text{ is a directed graph} \\ \text{with a path from } s \text{ to } t \end{array} \right\}$$

$$\text{size } (G) = \log_2 n + m(2 \log n) \quad O(m \log n)$$

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



DFS = depth-first search

$\langle G, s, t \rangle \in \text{PATH}$

Textbook Questions

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

DPS running time =
 $O(m+n)$.

($\underline{O(m \log n)}$)

~~Assignment Project Exam Help
 $n=15$
 $\langle n, 2, e^{\pi}, \dots, em \rangle$~~

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

$O(n^2)$

$m - O(\sqrt{n})$

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

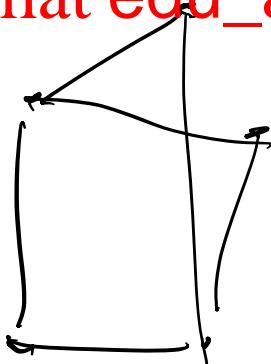
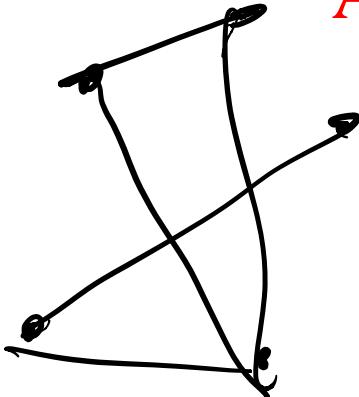
exist? $\exists c$

x y z

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

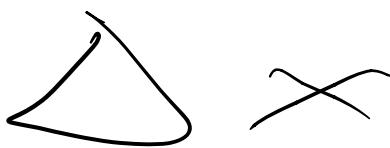
$G = \langle V, E \rangle$ Assignment Project Exam Help
 3, 1, n, $\{u_1, v_1\}, \{u_2, v_2\}, \dots, \{u_m, v_m\}$.
<https://eduassistpro.github.io/>

Add WeChat `edu_assist_pro`



(1, 2, 3)
 1, 2, 4
 1, 2, 5
 ...
 6

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



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

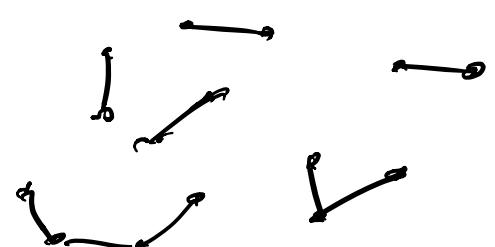
e_1, e_2, e_3



e_1, e_2, e_4



$e_{m-2}, e_{m-1}, e_m \rightarrow \Delta ?$



Nondeterminism Makes a Difference?

The Hamiltonian path problem has underlying language $HAMPATH =$

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

where a *Hamiltonian*

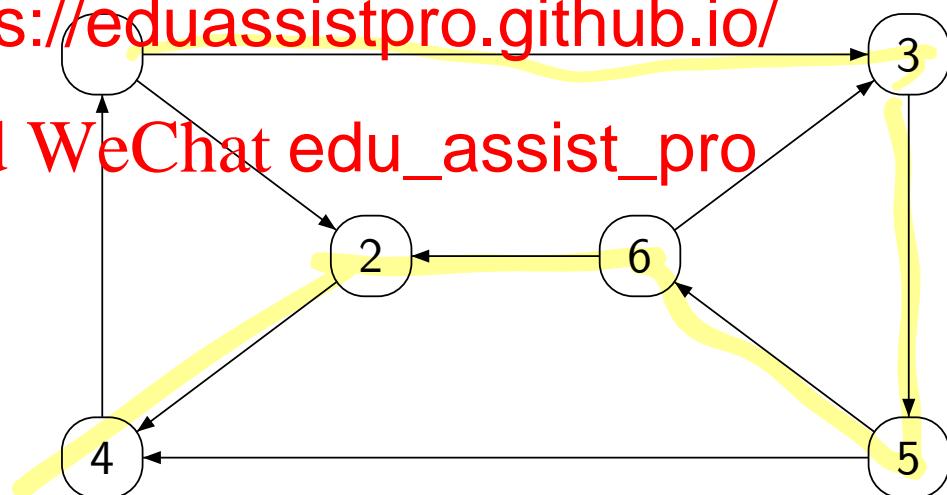
tly once.

<https://eduassistpro.github.io/>

Two instances:

Is there a Hamiltonian
path from 1 to 4?

From 1 to 6?



HAMPATH with Nondeterminism

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.

Assignment Project Exam Help

One tape holds the graph (d)

<https://eduassistpro.github.io/>

Machine N writes a guessed sequence of $i_1, i_2, i_3, \dots, i_m$ between 1 and m , on a different tape, each representing a node.

Add WeChat edu_assist_pro

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

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.

Running-time Analysis

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

<https://eduassistpro.github.io/>

The guess is done in polyno

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

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

Do we need Nondeterminism?

Guessing in linear time is clearly very powerful.

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

A deterministic machine has to consider all $m!$ sequences. <https://eduassistpro.github.io/> (there are $m!$ sequences). Add WeChat edu_assist_pro

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

Polynomial Verifiability

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

Assignment Project Exam Help

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

<https://eduassistpro.github.io/>

On the other hand, if somebody claims that a certain path, [checking](#) their [Add WeChat.edu_assist_pro](#)

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

Polynomial Verifiability

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

$$A = \{ \text{Graph } G, \text{ certificate } c \mid \exists \text{ verifier } V \text{ such that } V(G, c) \text{ accepts} \}$$

Language A is polynomially verifiable if it has a polynomial-time verifier: it runs in time that is polynomial in the length of w .

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

Polynomial Verifiability – complement

Notice that polynomial verifiability need not be closed under complement.

For example, consider HAMPATH.

There is no obvious (polynomial-time) way to verify the non-existence of a path.

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

The Class NP

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 **Assignment Project Exam Help**

<https://eduassistpro.github.io/>

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

Add WeChat edu_assist_pro

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.

P vs NP

In summary:

For P , membership can be **decided** quickly.

For NP , membership can be **verified** quickly

Assignment Project Exam Help

Clearly $P \subseteq NP$. I

<https://eduassistpro.github.io/>

P = N

Add WeChat **edu_assist_pro**

Many computer scientists consider this the

most important unanswered question

in computer science.

The Nature of an *NP* Problem

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?

Assignment Project Exam Help

This problem has the property of being verifiable.

Discovering a Hamiltonian path seems difficult, indeed we do not have a fast algorithm for it. **Add WeChat edu_assist_pro**

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.

Some Problems in NP

A clean NP problem is

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

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

Example: $S = \{2, 2, 2, 2, 2, 2\}$, $t = 17$, $\langle S, t \rangle \in SUBSET-SUM$?

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

2, 2, 2, 2, 2, 2

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.

Textbook Questions

Questions you can attempt at this stage:

Exercise 7.7

Exercise 7.12

Problem 7.16

Problem 7.19

A C N
https://eduassistpro.github.io/
is A G P
Add WeChat edu_assist_pro

Closure under Complement?

Recall that HAMPATH has no obvious certificate of the non-existence of a path.

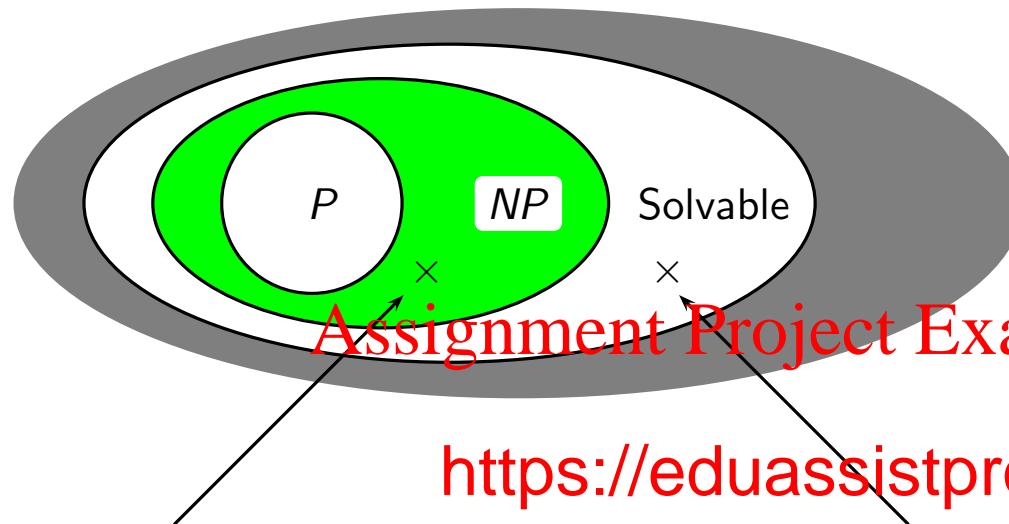
Similarly SUBSET-SUM <https://eduassistpro.github.io/>

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

$$\langle S, t \rangle \dashv A \subseteq S \quad \sum_{s \in S} s = t.$$

$$\text{SUBSET-SUM} \neq \{ \langle S, t \rangle \mid \nexists A \subseteq S. \dashv \{ s \in S \mid s \in A \} \}$$

Venn diagram



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

Does one exist?

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

Add WeChat `edu_assist_pro` at provably
omial time by a NDTM (“nondeterministically intractable”).
EXPSPACE-complete problems,
e.g., . . .

Polynomial-Time Reducibility

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.

<https://eduassistpro.github.io/>

A is polynomial-time (mapping) reducible to B , iff there is some polynomial-time (deterministic) computable function $f: \Sigma^* \rightarrow \Sigma^*$, 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 .

Reducibility and Language Classification

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

$$A \leq_P A \quad A \leq_P B, B \leq_P C \Rightarrow A \leq_P C.$$

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

<https://eduassistpro.github.io/>

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?

$$A \leq_P B \quad B \in P$$

$$w \xrightarrow{f(w)} j(f(w)) \quad \text{poly}_1(j(w)) \xrightarrow{\tau} \text{poly}_2(\text{poly}_1(f(w)))$$

$$|f(w)| \leq \text{poly}_1(|w|),$$

$$\text{poly}_2(\text{poly}_1(|w|)).$$

NP-Completeness

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.

<https://eduassistpro.github.io/>

We sometimes denote the class of NP-complete languages by NPC .

Add WeChat edu_assist_pro

The NP -complete languages are interesting because

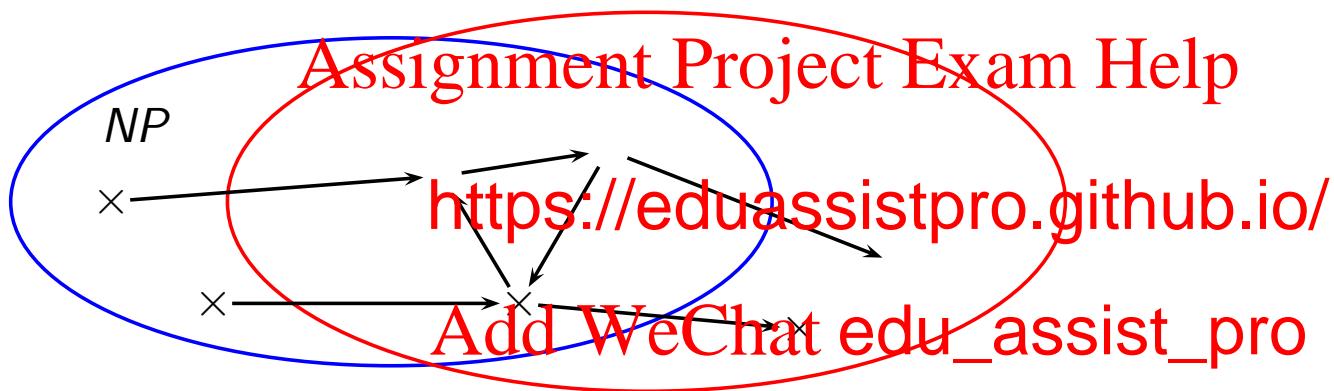
Theorem: If $B \in NPC$ and $B \in P$ then $P = NP$.

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

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

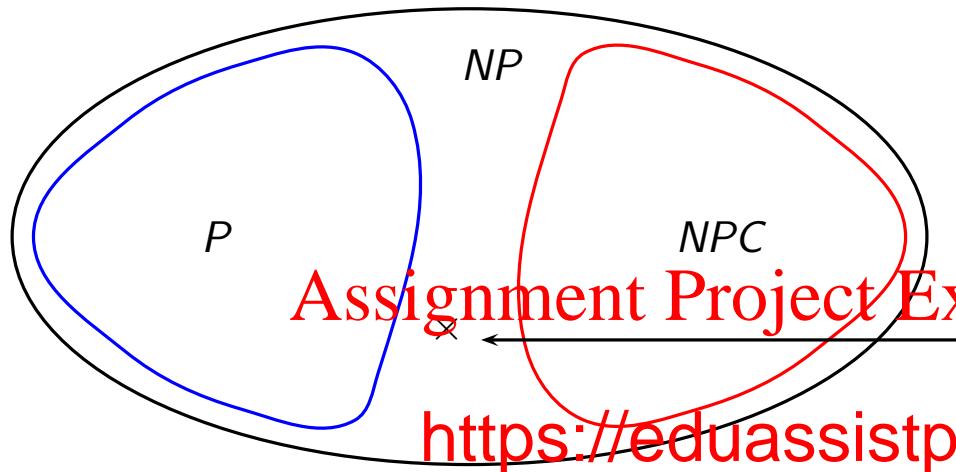
NPC Diagrammatically

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



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

The Fabled “Intermediate” Class NPI



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

Moreover, if $P \neq NP$, this gap is known to be inhabited (this is the [P vs NP Theorem](#)).

Add WeChat `edu_assist_pro`

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/>

Assignment Project Exam Help

Questions to attempt:

Problem 7.18

<https://eduassistpro.github.io/>

Problem 7.20

Add WeChat edu_assist_pro

Proving *NP*-Completeness

The plan for demonstrating *NP*-completeness:

- The *ancestor* of all *NPC* problems: *SAT*.
 - ① Show $\text{SAT} \in \text{NP}$.
 - ② Show that
- For each subsequent problem <https://eduassistpro.github.io/>
 - ① Show $B \in \text{NP}$.
 - ② Show B is *NP-hard* by describing a polynomial reduction from some other known *NP-hard* problem to B .

Cook followed exactly this path in his 1971 paper.

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

Propositional Logic

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

<https://eduassistpro.github.io/>

$\{0, 1\}$.

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

Add WeChat edu_assist_pro

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

Propositional Logic

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

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

x	$\neg x$	$x \wedge y$	$x \vee y$
0	1	0	0
1	0	1	1
0	1	0	1

A clause is a disjunction of literals (using \vee). Assignment Project Exam Help

A formula is in conjunctive normal form if it is a conjunction (\wedge) of clauses. Every formula has an equivalent CNF form. Converting one to the other might require more than just logical steps. This is because some formulas are not in CNF to begin with. For example, consider the formula $(P \rightarrow Q) \wedge R$. It is not in CNF because it contains a implication operator (\rightarrow). To convert it to CNF, we need to use De Morgan's laws and other logical equivalences to eliminate the implication operator. This process can be complex and may result in a formula with many more clauses than the original.

Formula φ is **satisfiable** if there is some assignment of values to the variables such that the truth value of formula φ is 1.

Language $SAT = \{\varphi \mid \varphi \text{ is in CNF and is satisfiable}\}$. $(x \vee y \vee z) \wedge (\bar{x} \vee y)$

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

Example formulas in CNF

\phi

$$x_1 = 1 \quad x_3 = 1$$

\downarrow

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

$$\psi = (x_1 \vee x_2) \wedge (\cancel{x_1} \text{ https://eduassistpro.github.io/}$$

Which of these formulas is satisfiable?

Add WeChat edu_assist_pro

\psi

SAT is in NP

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 $\bar{1}$ denote a string of i 1s and encode

$$x_i \text{ by } \bar{1} \oplus$$

Assignment Project Exam Help

For example, $(x_1 \vee$

<https://eduassistpro.github.io/>
1 11 \oplus
variables Add WeChat edu_assist_pro

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?

SAT is in NP

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

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

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

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

- Scan to the formula $(x_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee x_2 \vee x_3)$
- While there is another clause:
 - For each literal **Add WeChat edu_assist_pro**
 - 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**.

Assignment Project Exam Help

Question to attempt:

Exercise 7.5

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

Modelling NTM Computation

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.)

Assignment Project Exam Help

$$Q = \{q_0, q_1, \dots, q_m\}$$

<https://eduassistpro.github.io/>

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

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

$$u \in \Sigma^n.$$

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

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

$A \subseteq_p B$ $A = \{0, 00, 0000, 010, -\}$

$f(u) f: \Sigma^* \rightarrow \Sigma^*$ $B = \{1, 11, 111, 101, -\}$

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

$$2^n n^2$$

1	2	1
2	4	4
3	8	9
4	16	16
5	32	25

$A \in NP$

$A \subseteq_p SAT$

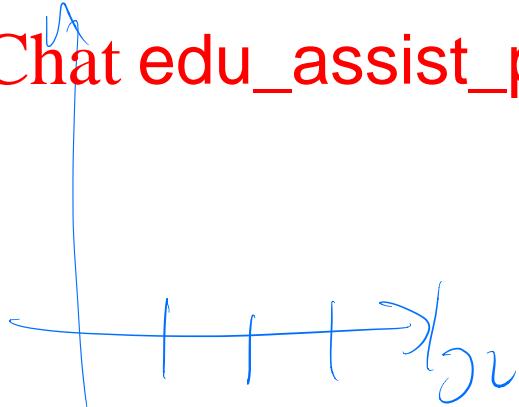
$u \in A \Leftrightarrow \varphi_u \in SAT$.

Assignment Project Exam Help

<https://eduassistpro.github.io/>

6 64 36
7 128 49
10 1024 100
20 10⁶ 400
30 10⁹ 900

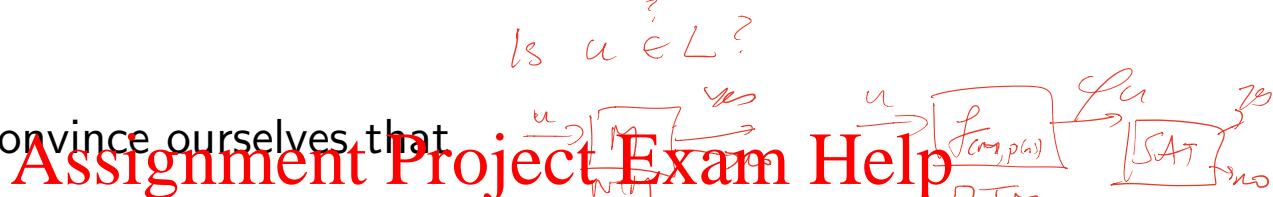
Add WeChat edu_assist_pro



What To Look Out For in the Proof

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 <https://eduassistpro.github.io/> 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 .

Modelling NTM Computation: Variables Used

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??**

Assignment Project Exam Help

The variables have the following interpretations:

<https://eduassistpro.github.io/>

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

Modelling NTM Computation: Starting the NTM

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

Q_{00}
Assignment Project Exam Help
 \wedge S_{000}

\wedge 2
<https://eduassistpro.github.io/>

Add WeChat `edu_assist_pro`
 S_{nr_n0}
 $S_{(n+1)00}$
 \vdots
 \wedge $S_{p(n)00}$

Modelling NTM Computation: Invariants

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

To capture **state invariants**

(Remember, $a \Rightarrow b \equiv \neg a \vee b$)

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

$$a \Rightarrow b$$

$$\neg b \Rightarrow a$$

Assignment Project Exam Help

$$\bigvee_{i=0}^m Q_{ik}$$

At all times, M is in some state,

$$Q_{ik} \Rightarrow$$

$$\neg Q_{ik} \vee \text{other states}$$

<https://eduassistpro.github.io/>

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

Add WeChat edu_assist_pro

$$\bigvee_{j=0}^{p(n)} P_{jk}$$

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

Modelling NTM Computation: Tape Consistency

Symbols **not** under the tape head are **not** changed.

<https://eduassistpro.github.io/>

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

Add WeChat **edu_assist_pro**

Modelling NTM Computation: NTM Transitions

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)

Assignment Project Exam Help

<https://eduassistpro.github.io/>

where

Add WeChat edu_assist_pro

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

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

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

Modelling NTM Computation: Nondeterministic Transitions

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 the possible transitions from a given state q_i is added for each of the possible transitions from a given state q_i . To model this, we add the clauses from the previous slide for each possible 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 .

Modelling NTM Computation: Padding

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

<https://eduassistpro.github.io/>

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

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

Modelling NTM Computation: Acceptance

Assignment Project Exam Help

To capture acceptance:

<https://eduassistpro.github.io/>

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

Add WeChat edu_assist_pro

That's it!

Assignment Project Exam Help

The formula we have constructed is in CNF.

The construction ensures <https://eduassistpro.github.io/>

- 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.

NP-Hard vs Exponential

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 $P = NP$; indeed, it is undecidable.

<https://eduassistpro.github.io/>

It is possible that some problem is *NP*-hard and yet it is not *NP*-hard. Nothing in our definitions rules this out.

Add WeChat edu_assist_pro

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.)

SAT is *NP*-Complete

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

Assignment Project Exam Help

We established that every *NP*-complete problem can be reduced to *SAT* in polynomial time.

<https://eduassistpro.github.io/>

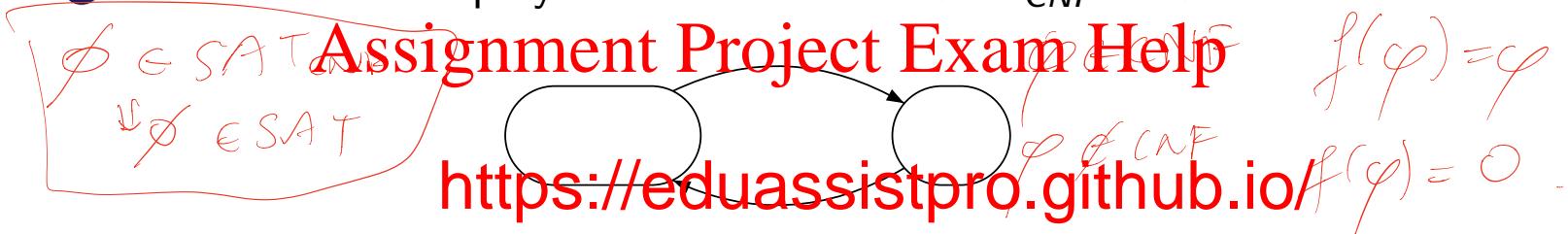
Our construction reduced every *NP*-complete problem to *SAT*, the satisfiability problem for formulas in Conjunctive Normal Form.

Add WeChat edu_assist_pro

SAT is *NP*-Complete

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 generator 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 formula. 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

Assignment Project Exam Help
 $(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_1 \vee x_2 \vee \neg x_4)$

3SAT is the language of s
<https://eduassistpro.github.io/>

Let us show that it is *NP*-complete

~~3SAT~~ ~~CNF~~
Add WeChat edu_assist_pro

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

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

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:

Assignment Project Exam Help

- If C is a single literal esh

variables x and <https://eduassistpro.github.io/>

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

Add WeChat edu_assist_pro

- 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

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

Assignment Project Exam Help
produce $n - 2$ clauses using $n - 3$ fresh variables x_3, \dots, x_{n-1} :

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

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

Why does this preserve satisfiability?

3SAT

⇒ Let t be a truth assignment that satisfies

well

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

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

$$\text{https://eduassistpro.github.io/} (\ell_1 \vee \ell_2 \vee \ell_3 \vee \cdots \vee \ell_n)$$

$$\begin{aligned} & x_3 \quad 3 \\ & \vdots \\ & \text{Add WeChat } \text{edu_assist_pro} \\ & \wedge \quad (\neg x_{n-2} \vee \ell_{n-2} \vee x_{n-1}) \\ & \wedge \quad (\neg x_{n-1} \vee \ell_{n-1} \vee \ell_n) \end{aligned}$$

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

Let t' be a truth assignment that satisfies

$$\begin{aligned} & (\ell_1 \vee \ell_2 \vee x_3) \\ \wedge \quad & (\neg x_3 \vee \ell_3 \vee x_4) \end{aligned}$$

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Then t' also satisfies $C = (\ell_1 \vee \ell_2 \vee \ell_3 \wedge \neg x_3 \vee \ell_3 \vee x_4)$

Add WeChat edu_assist_pro

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).

Assignment Project Exam Help

It is instructive to read both.

From Sipser's proof, it is clear that <https://eduassistpro.github.io/> is an acceptable 3-CNF formula. We choose to rule that

Add WeChat edu_assist_pro
Either way can be justified, but our assumption that the literals 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.

Textbook Questions

Questions you can try at this point:

Assignment Project Exam Help
Problem 7.22

Problem 7.24

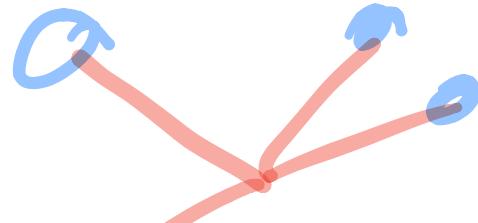
Problem 7.25

Problem 7.26

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

The Vertex Cover Problem



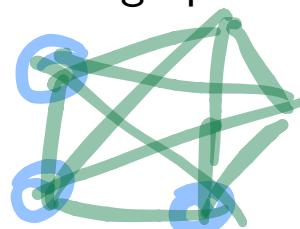
Let $G = (V, E)$ be an undirected graph.

Assignment Project Exam Help
 $S \subseteq V$ 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 .

<https://eduassistpro.github.io/>

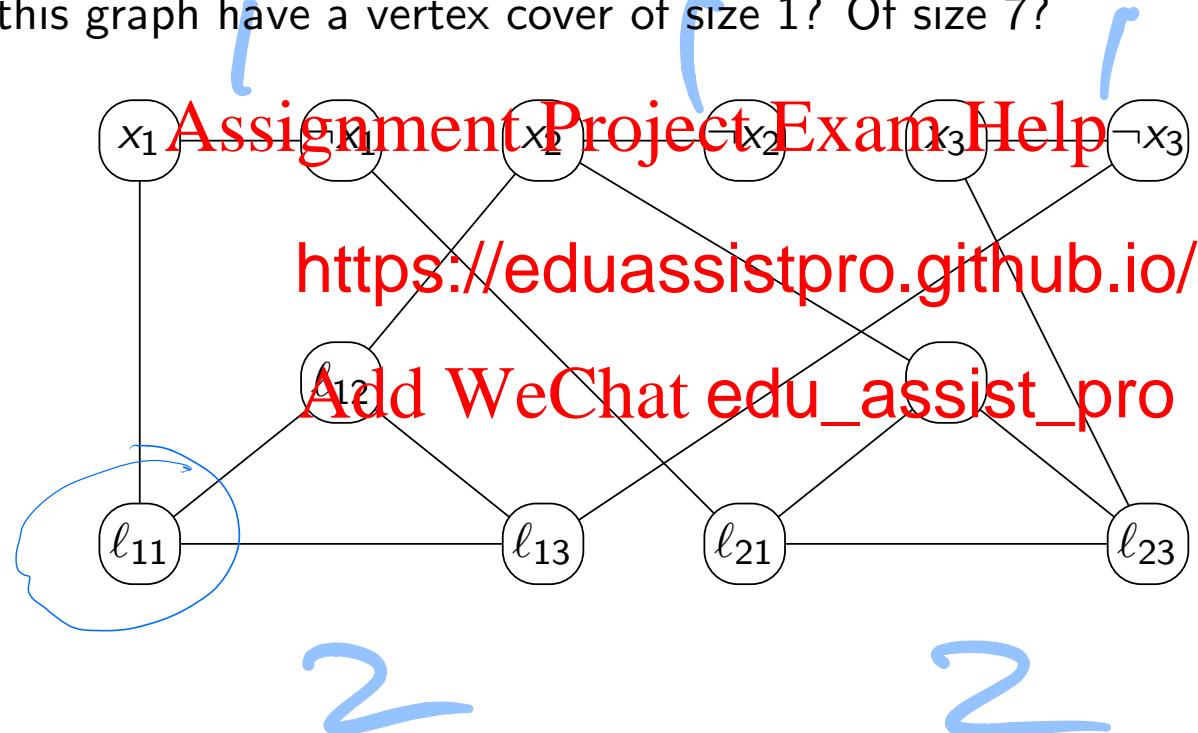
VERTEX-COVER =

$\{ \langle G, k \rangle \mid G \text{ is a graph that has a vertex cover } S \text{ such that } |S| \leq k \}$



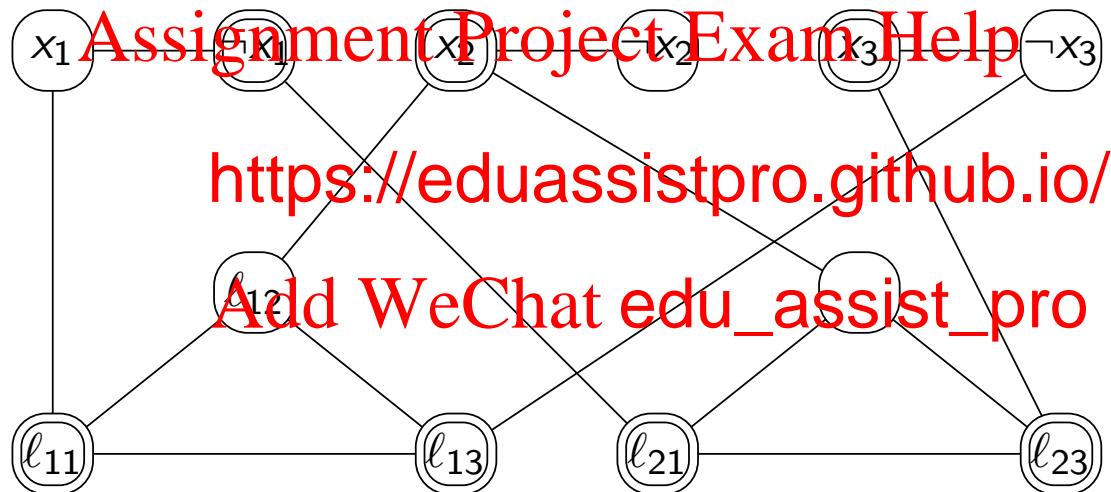
Vertex Cover Example

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



Vertex Cover Example

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



Vertex Cover is in NP

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.

<https://eduassistpro.github.io/>

The problem (language) nistic TM can
guess a set of k nodes S and check in polynomial time if they
“cover” E .

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

$$G = \langle n, (u_1, v_1), (u_2, v_2), \dots, (u_m, v_m) \rangle$$
$$S = \{ s_1, s_2, \dots, s_k \}$$

Reducing 3SAT to Vertex Cover

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

Assignment Project Exam Help

We produce $3m + 2$

two per variable:

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

and $6m + n$ edges:

$$T_i = \{(x_i, \neg x_i) \mid 1 \leq i \leq n\}$$

$$C_k = \{(\ell_{k1}, \ell_{k2}), (\ell_{k2}, \ell_{k3}), (\ell_{k3}, \ell_{k1})\}$$

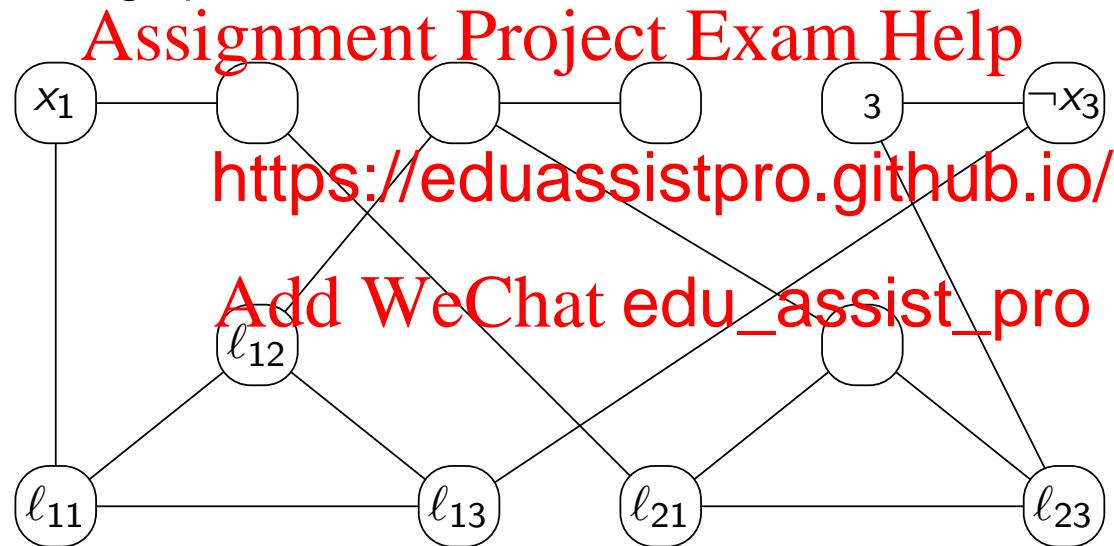
$$L_k = \{(\ell_{k1}, v_{k1}), (\ell_{k2}, v_{k2}), (\ell_{k3}, v_{k3})\}$$

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

3SAT to Vertex Cover

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

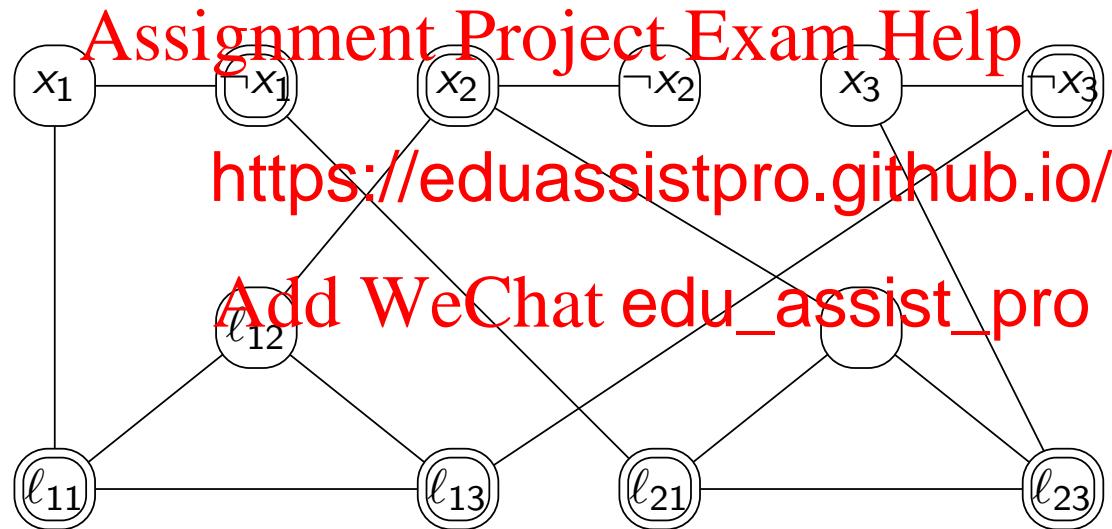
For example, $(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3)$ is translated to the problem of whether this graph has a cover of size $2 \cdot 2 + 3 = 7$:



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

3SAT to Vertex Cover 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 $2 \cdot 2 + 3 = 7$:



3SAT to Vertex Cover

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.

<https://eduassistpro.github.io/>

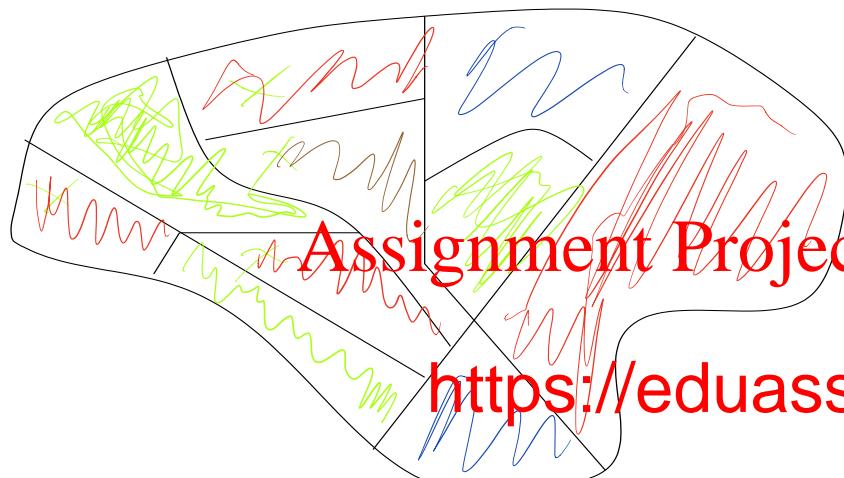
⇒ To be a cover of size $2m + n$, S includes exactly one variable gadget node and exactly two “clause gadget” nodes. This defines a truth assignment t , as it includes exactly one of x_i or \bar{x}_i for every variable x_i . Moreover, this t makes each clause true: The ℓ_{kj} that is not in the cover S is set to true.

3SAT to Vertex Cover

Let t satisfy ψ . That gives us m nodes. For each i , either x_i or $\neg x_i$) in the cover.

Now for each clause <https://eduassistpro.github.io/> it is true with assignment t . Add to the clause C_k the two other literals in C_k . The result is a cover of the requirement $n + 2m$.

Map and Graph Colouring



Assignment Project Exam Help

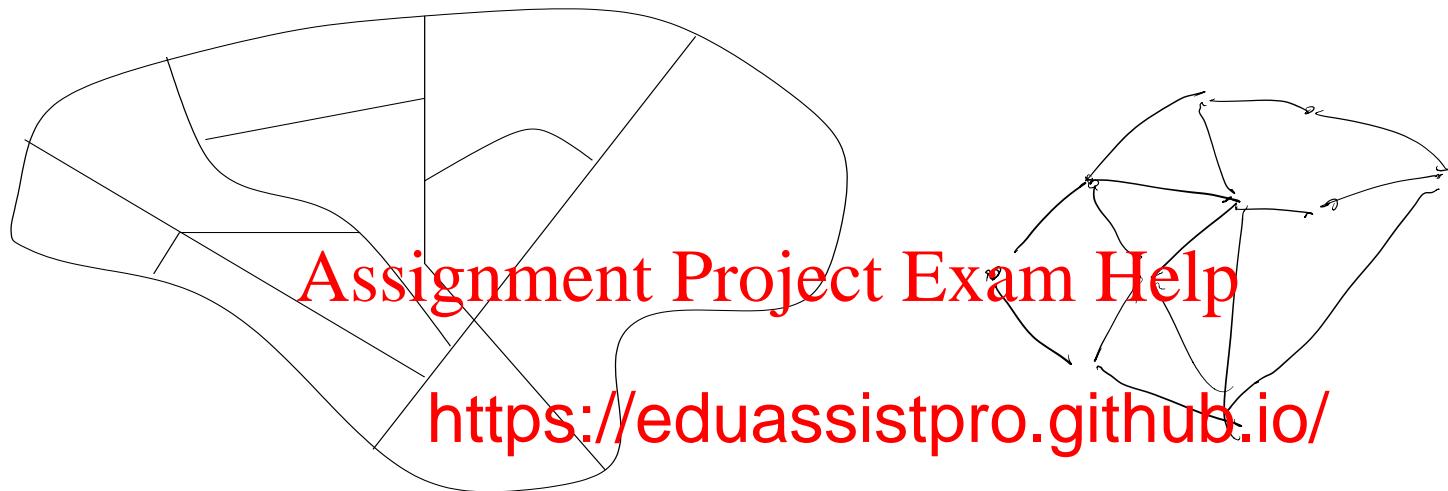
<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

How few colours suffice to colour this map?

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

Map and Graph Colouring



Add WeChat `edu_assist_pro`
Four colours suffice in the plane! (Appel and Haken, 1976)

Tractability of n -colouring:

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

NP and *co-NP*

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

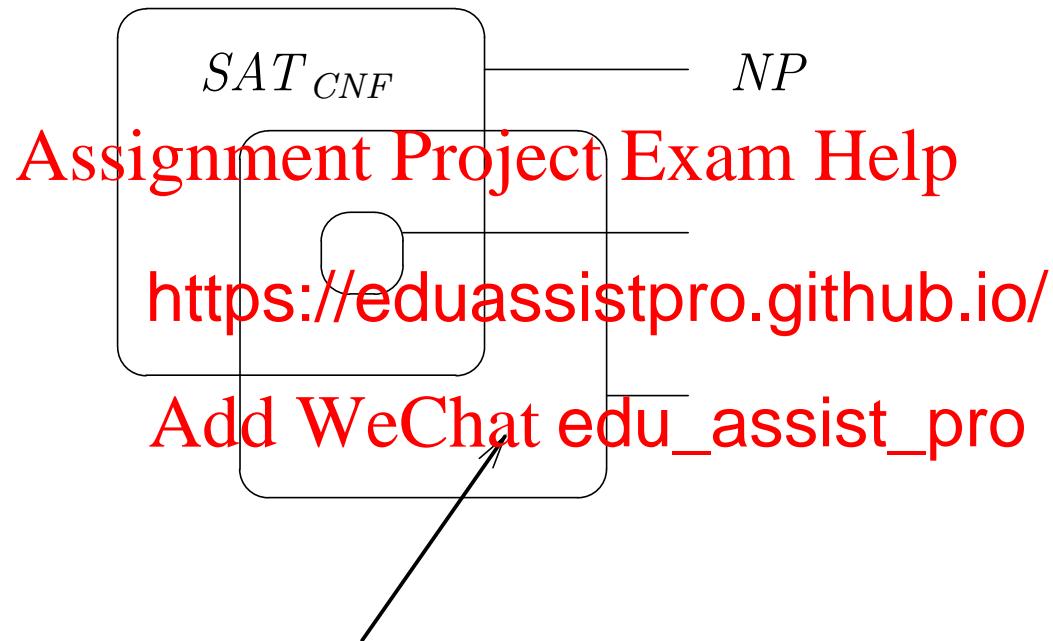
Using a nondeterministic machine to guess-and-check it is not clear that this property should hold!

For example, how does an NFA formula is unsatisfiable?

Add WeChat `edu_assist_pro`

It is not known whether *NP* is closed under complementation. (In fact, 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.



NP and *co-NP* Example

The dual of SAT_{DNF} , that is, the DUAL problem for propositional formulas in DNF, is in

Satisfiability: φ is true <https://eduassistpro.github.io/>

Validity: φ is true for all assignments of variables

$$\begin{array}{c} ((\vee \vee) \wedge (\sim \vee)) \wedge \dots \\ ((x_1 \bar{y} \bar{z}) \vee (\sim \sim) \vee (\sim \sim)) \\ T F T \end{array}$$

INDSET is NP-Complete

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 **INDSET** have an independent set of size k ?

<https://eduassistpro.github.io/>

$$\text{INDSET} = \left\{ \langle G, k \rangle \mid \begin{array}{l} G \text{ is an un} \\ \text{Add WeChat } \text{edu_assist_pro} \\ \text{an indep} \end{array} \right\}$$

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

INDSET is NP-Complete

$v_1, v_2, v_3, \dots, v_m$ S_1, S_2, \dots, S_n .

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

$$(x_1 \vee y \vee z) \quad x_1 = F, y = T, z = F$$

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



Each size- m clause of $2^m - 1$ node that satisfies c .

<https://eduassistpro.github.io/>

ed to a subgraph
assignment

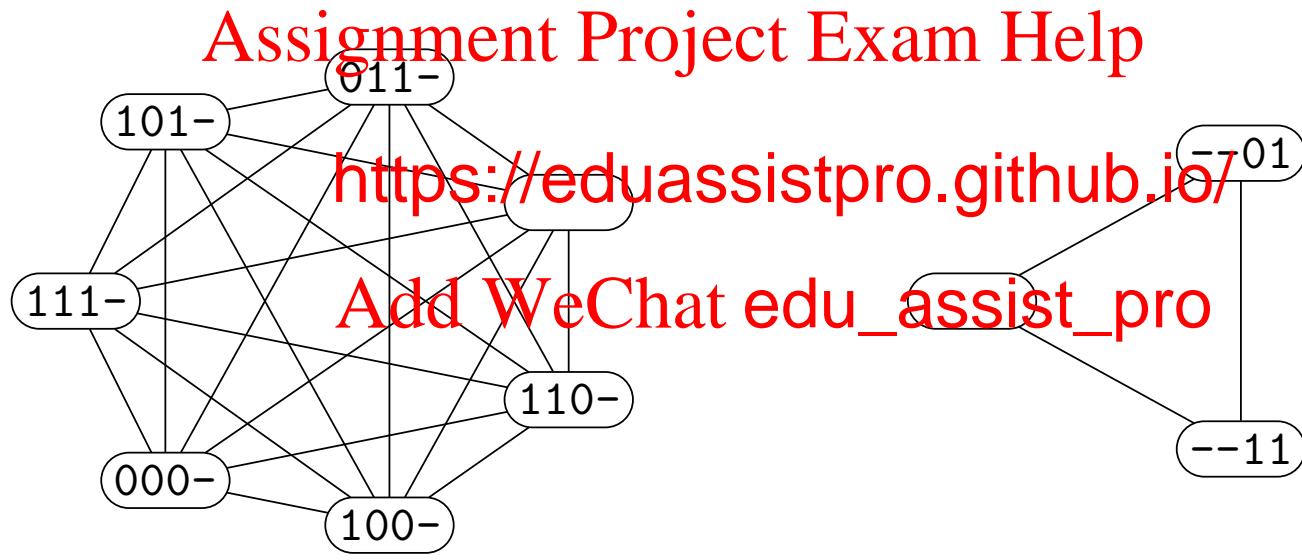
Add WeChat edu_assist_pro

There is an **edge** between every pair of nodes in w if their assigned 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.

3SAT to *INDSET* Reduction: Example

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.

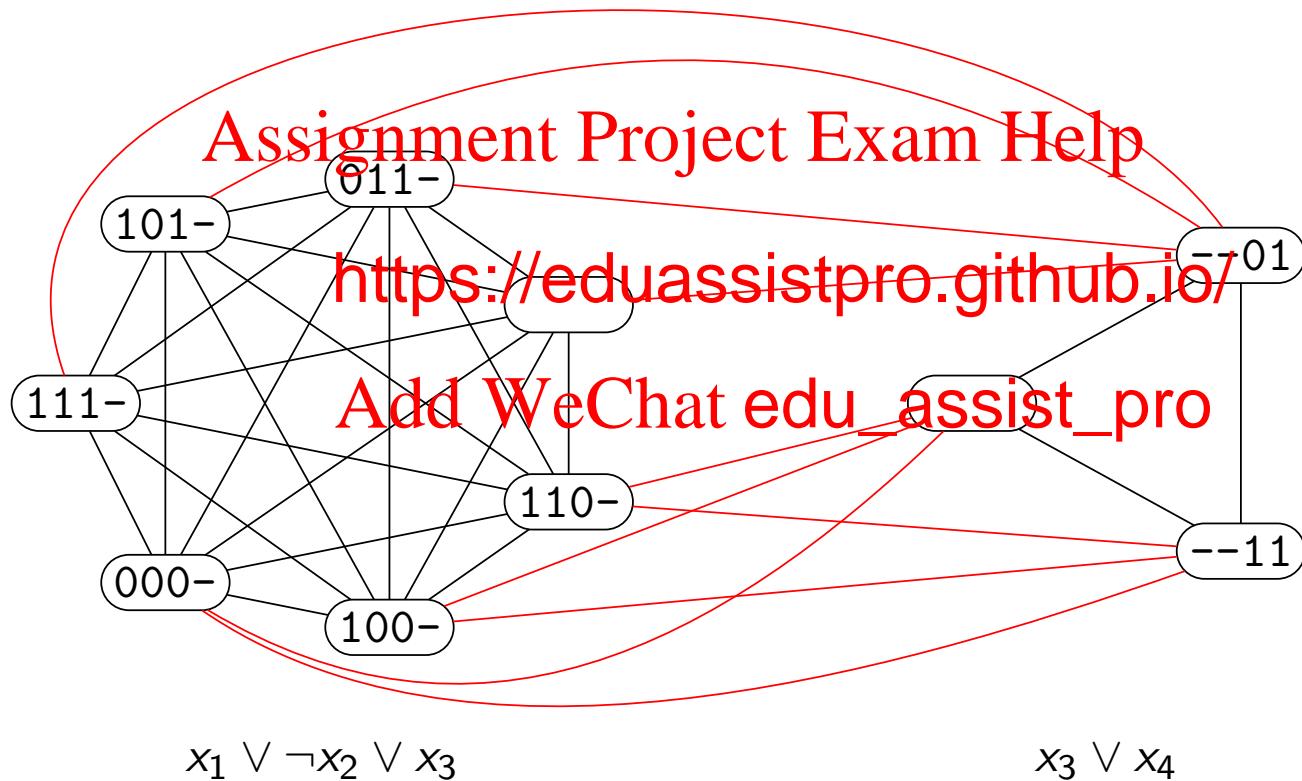


$$x_1 \vee \neg x_2 \vee x_3$$

$$x_3 \vee x_4$$

3SAT to *INDSET* Reduction: Example

Now add remaining links, between incompatible assignments:



3SAT to *INDSET* Reduction

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

Assignment Project Exam Help

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

<https://eduassistpro.github.io/>

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

[Add WeChat edu_assist_pro](#)

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

Textbook Question

Assignment Project Exam Help

Question you can try at this

Problem 7.23

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

Reducing 3SAT to SUBSET-SUM

We already saw that

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

is in NP .

Assignment Project Exam Help

Let us reduce 3SAT

<https://eduassistpro.github.io/>

Say φ has n variables 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.

$3SAT$ to $SUBSET-SUM$

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

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

Assignment Project Exam Help

This set of $2n + 2k$
produced in polynomial time
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	0	1	1	0
c_2	0	0	0	0	1
	0	0	0	0	1
t	1	1	1	3	3

$3SAT$ to $SUBSET-SUM$

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 that 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	0	1	1	0
	0	0	1	0	0
c_2	0	0	0	0	1
	0	0	0	0	1
t	1	1	1	3	3

Pseudo-Polynomial Algorithms

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

<https://eduassistpro.github.io/>

The i th column gives the constructed using the first i elements of the

Add WeChat edu_assist_pro

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].

$\log_2 k$

$\langle t, a_1, a_2, \dots, a_n \rangle$

Sum	2	5	8	9
0	T	T	T	T
1	F	F	F	F
2	T	T	T	T
3	F	F	F	F
4	F	F	F	F
5	F	T	T	T
6	F	F	T	T
7	F	T	T	T
8
9
10
11
12
13	F	F	.	.

Pseudo-Polynomial Algorithms

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

Assignment Project Exam Help

The table above is polynomial in the size n of the set. However, it is not polynomial in the ~~size~~ t , which would only be $O(\log t)$. Unless the ~~size~~ <https://eduassistpro.github.io/> ~~size~~ t , the table is in fact ~~exponential~~ in the size of ~~in~~

Add WeChat edu_assist_pro

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

Textbook Question

Assignment Project Exam Help

Question you can try at this

Problem 7.17

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

Hamiltonian Path is *NP*-Complete

Recall the Hamiltonian path problem $HAMPATH =$

Assignment Project Exam Help
 $\{(S, g, t) \mid G \text{ is a directed graph with a Hamiltonian path from } s \text{ to } t\}$

where a Hamiltonian path is a simple cycle that visits every vertex exactly once.

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

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

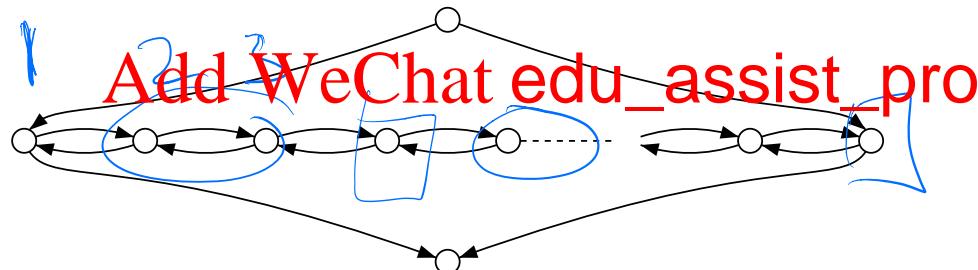
Reducing 2SAT to HAMPATH

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.

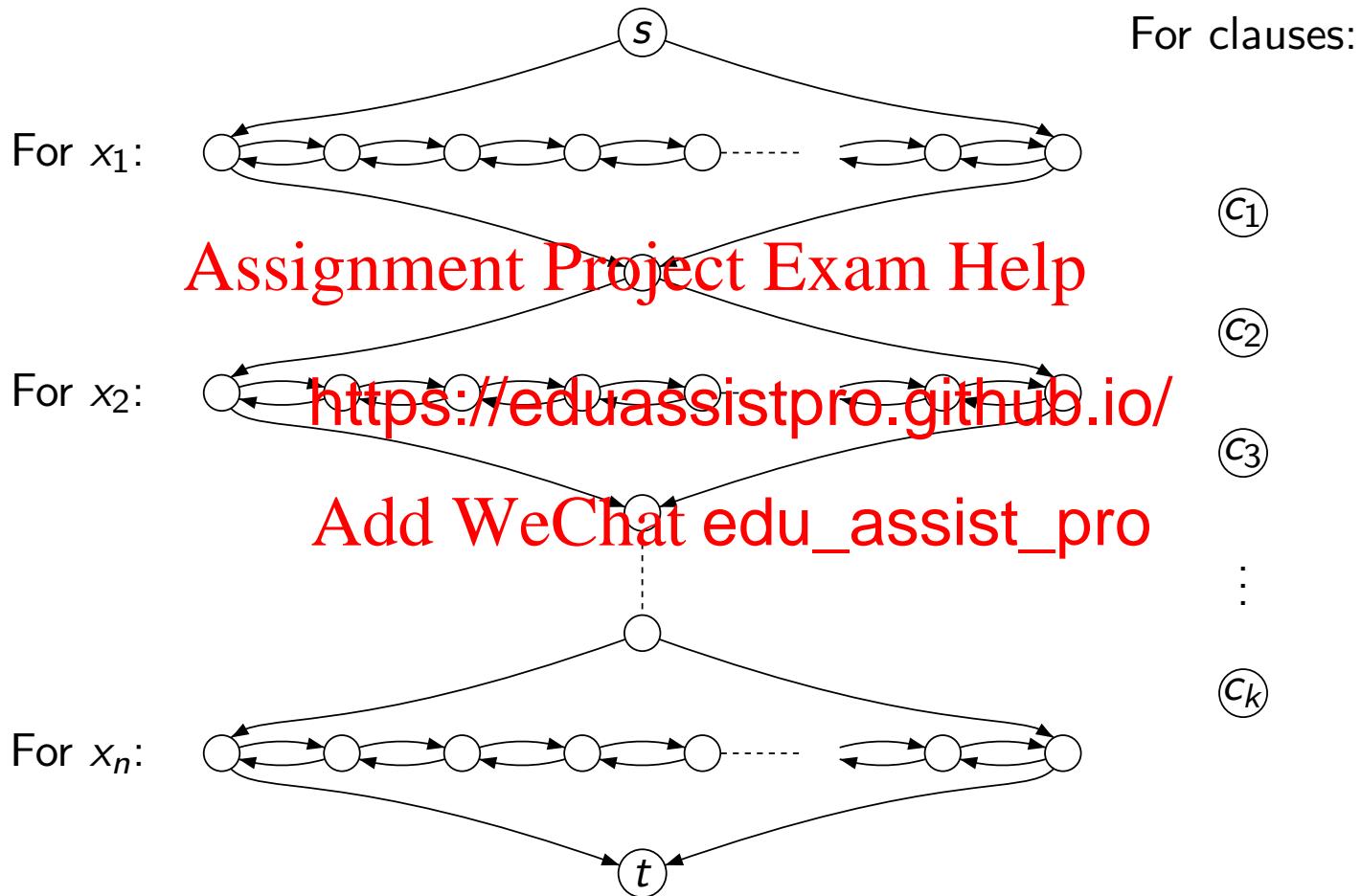
$$\varphi = (a_1 \vee b_1 \vee c_1) \wedge (a_2 \vee b_2 \vee c_2) \wedge \dots \wedge (a_k \vee b_k \vee c_k)$$

For each variable x_i <https://eduassistpro.github.io/> do 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.

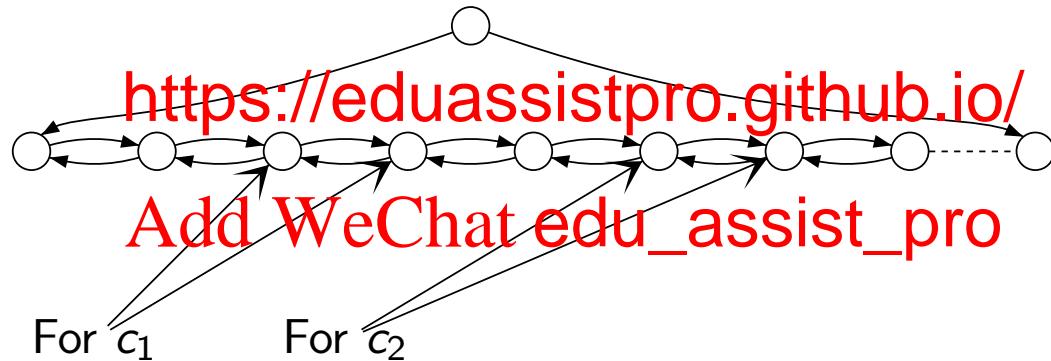
Reducing SAT to HAMPATH



Reducing *SAT* to *HAMPATH*

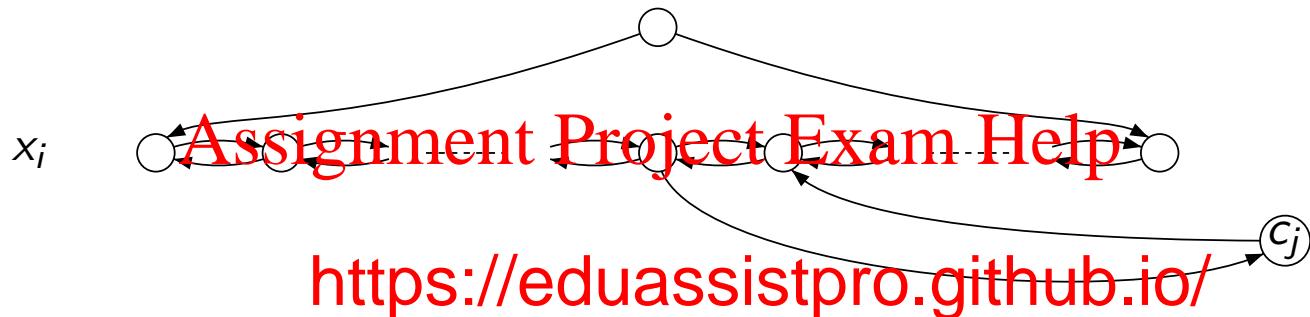
The chain of $3k + 1$ nodes comprises one **pair** per clause, plus separator nodes on either side:

Assignment Project Exam Help

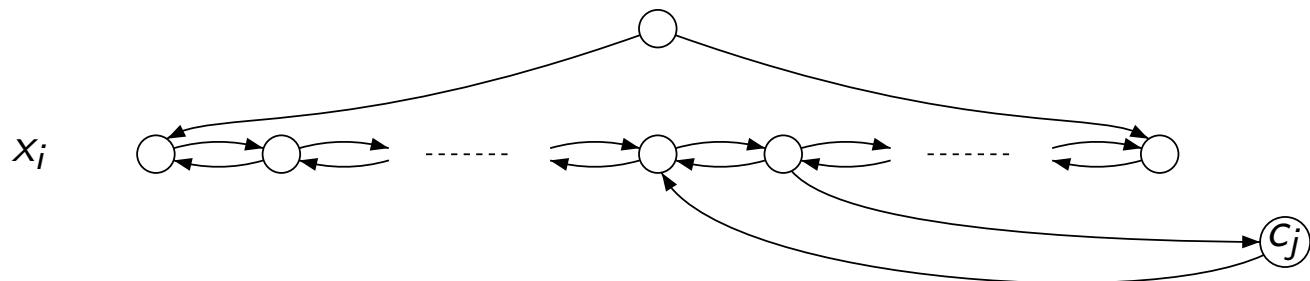


Reducing SAT to HAMPATH

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
Add WeChat edu_assist_pro



Reducing SAT to $HAMPATH$

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.

<https://eduassistpro.github.io/>



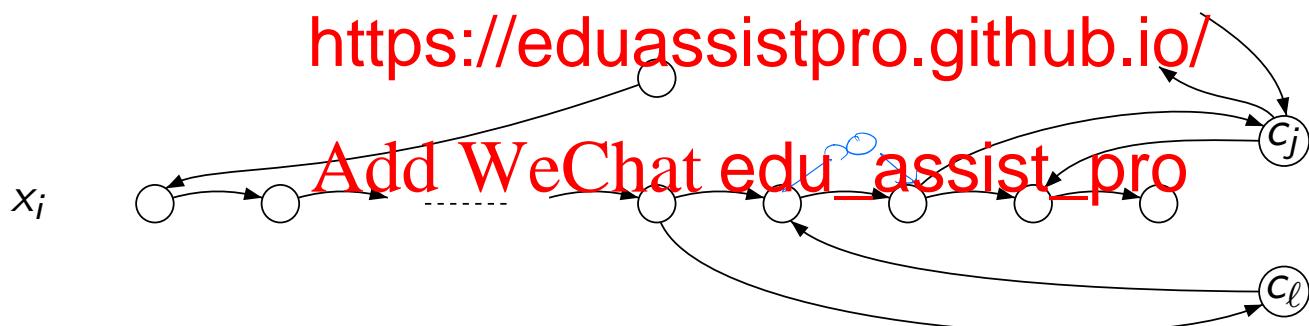
On the other hand, walking right-to-left through the i^{th} chain corresponds to setting $\neg x_i$ to true.

Reducing SAT to $HAMPATH$

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.

<https://eduassistpro.github.io/>



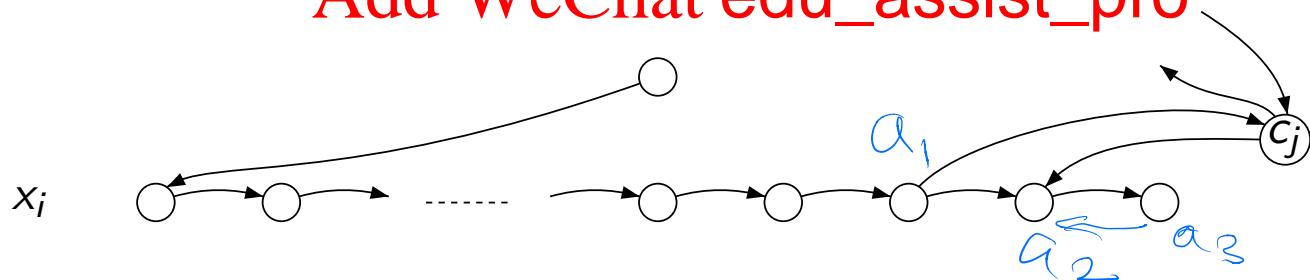
A single x_i may make several different clauses true.

Reducing SAT to HAMPATH

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 state, the next node in the chain, a_3 , is an entry point, as they lead elsewhere; node a_3 is an exit point. Available as entry points are a_1, c_j .

<https://eduassistpro.github.io/>



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

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

Textbook Questions

Questions you can try at this point:

Problem 7.21

Assignment Project Exam Help

Problem 7.27

Problem 7.28

Problem 7.29

Problem 7.30

Problem 7.31

Problem 7.37

Problem 7.43

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

Decision vs Witness Production or Search

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

Assignment Project Exam Help
But often we are interested in more than a yes/no answer—we want a **witness**.

<https://eduassistpro.github.io/>

Is that substantially harder than getting a yes/no?

In the context of **optimization problems**,
Add WeChat edu_assist_pro route of
length no more than k ?", but "what is the shortest route?"

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

Decision vs Witness Production

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

<https://eduassistpro.github.io/>

The next slide exemplifie rhead indeed is polynomial.

Add WeChat edu_assist_pro

Decision vs Search

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

Assignment Project Exam Help
For $i \in [1, n]$:

<https://eduassistpro.github.io/>

If $M(t_i^0 \circ \varphi) = \text{'yes'}$
Add WeChat: edu_assist_pro

$t := t_i^0 \circ t$

else

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

$t := t_i^1 \circ t$

Output t

Decision vs Search

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 <https://eduassistpro.github.io/>

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

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).

Textbook Question

Assignment Project Exam Help

Question you can try at this

Problem 7.46

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

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.

<https://eduassistpro.github.io/>

What's next

Randomised algorithms: probabilistic TMs

Add WeChat edu_assist_pro

Space complexity: look at the beginning of Chapter

arser text.

Probabilistic Turing Machines



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

Assignment Project Exam Help

A mental model: The mac

bits, initialised freshly for <https://eduassistpro.github.io/>

dom

Each nondeterministic step is a choice of legal next moves, of equal probability. (There is no loss of generality in considering just two moves.)

Add WeChat edu_assist_pro

What Does Acceptance Mean?

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 needed.

<https://eduassistpro.github.io/>

In contrast, a probabilistic machine may accept or reject with some probability: unless this probability is 0 or 1, the result can depend on which of the possible 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.

Probability of Acceptance

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 <https://eduassistpro.github.io/>

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

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

Recognition with Error Probability

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

M recognises L with error probability ε iff

- ① If $w \notin L$ then $P[\text{https://eduassistpro.github.io/} \text{ accepts } w] \leq \varepsilon$
- ② If $w \in L$ then $Pr[M \text{ accepts } w] \geq 1 - \varepsilon$

Add WeChat edu_assist_pro

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

An Example of a PTM

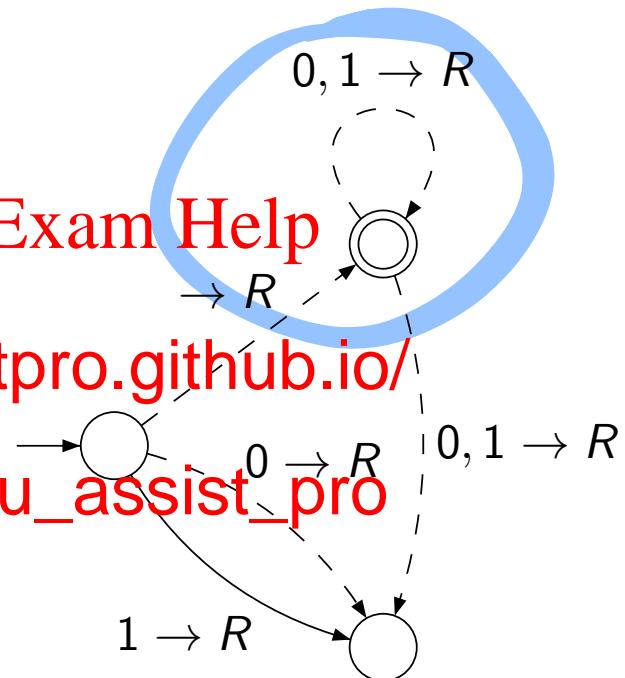
Here is a PTM for the language mentioned before.

The machine accepts each binary string starting with 0 with probability n .

where n is the length of the other string is rejected.

Add WeChat [edu_assist_pro](https://eduassistpro.github.io/)

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



The Class BPP

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 run some of <https://eduassistpro.github.io/> coin-tosses along the way

That is, the probabilistic computation “tree” must (across all branches) which is polynomially bounded.

The Amplification Lemma

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 fact that with polynomial-time effort, an error probability can be made exponentially small.

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

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:

Assignment Project Exam Help

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

<https://eduassistpro.github.io/>

2

Add WeChat edu_assist_pro

RP stands for “randomised polynomial”.

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

“Almost Certainly” vs “Certainly”

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
<https://eduassistpro.github.io/>
answer from 100 consecutive runs will give a wrong ‘no’

Success is so close to certainty that, for our purposes, it is hardware failure has a much higher likelihood.

An Example RP Problem

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

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

Output: Yes if G contains a triangle.

Method:

Repeat the following

Select an edge (x, y)

Select a node $z \in V$ uniformly at random

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

An Example RP Problem

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

Assignment Project Exam Help

<https://eduassistpro.github.io/>

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

Add WeChat edu_assist_pro

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 $1/2$.

An Example RP Problem

The complexity of the algorithm is polynomial.

Assignment Project Exam Help

In particular, $k = O$

the input size (assuming

<https://eduassistpro.github.io/>

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

Add WeChat edu_assist_pro

Hence the problem is in *RP*.

And with Amplification . . .

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

Assignment Project Exam Help

However, it could reject in

<https://eduassistpro.github.io/>

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

Add WeChat edu_assist_pro

Running the triangle algorithm 30 times, the likelihood of a false negative is less than one in a billion.

Where RP Lives

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

The first inclusion follows immediately from the definitions.

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

<https://eduassistpro.github.io/>

Whenever M uses a random bit, N chooses probabilistically between the two values.

Add WeChat edu_assist_pro

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.

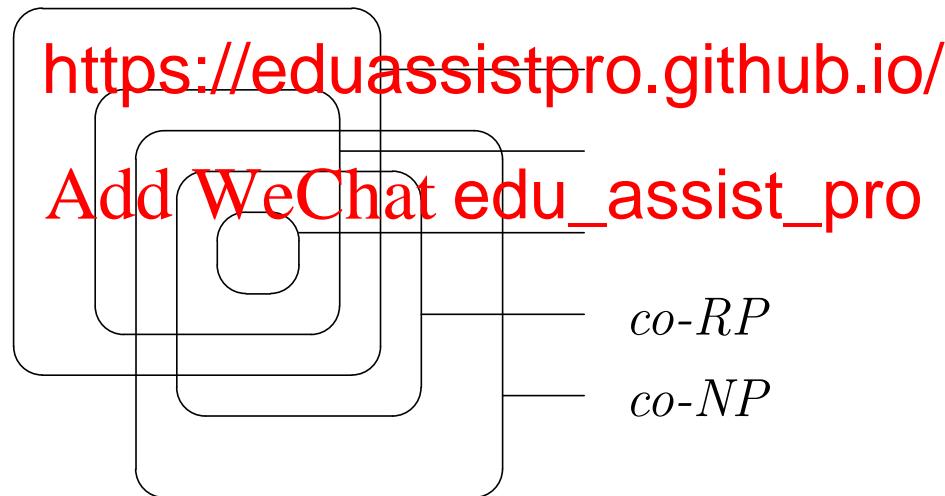
If w is not in L , no such sequence exists, so N will reject.

The Class co-RP

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 $\geq 1/2$.

Assignment Project Exam Help



An Example co-RP Problem

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

Assignment Project Exam Help

The polynomial identity testing problem asks whether two algebraic circuits compute the same function.

<https://eduassistpro.github.io/>

This is an example of a problem for which a fast randomized algorithm exists.

Add WeChat edu_assist_pro

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

Open Problems

Many interesting questions remain open.

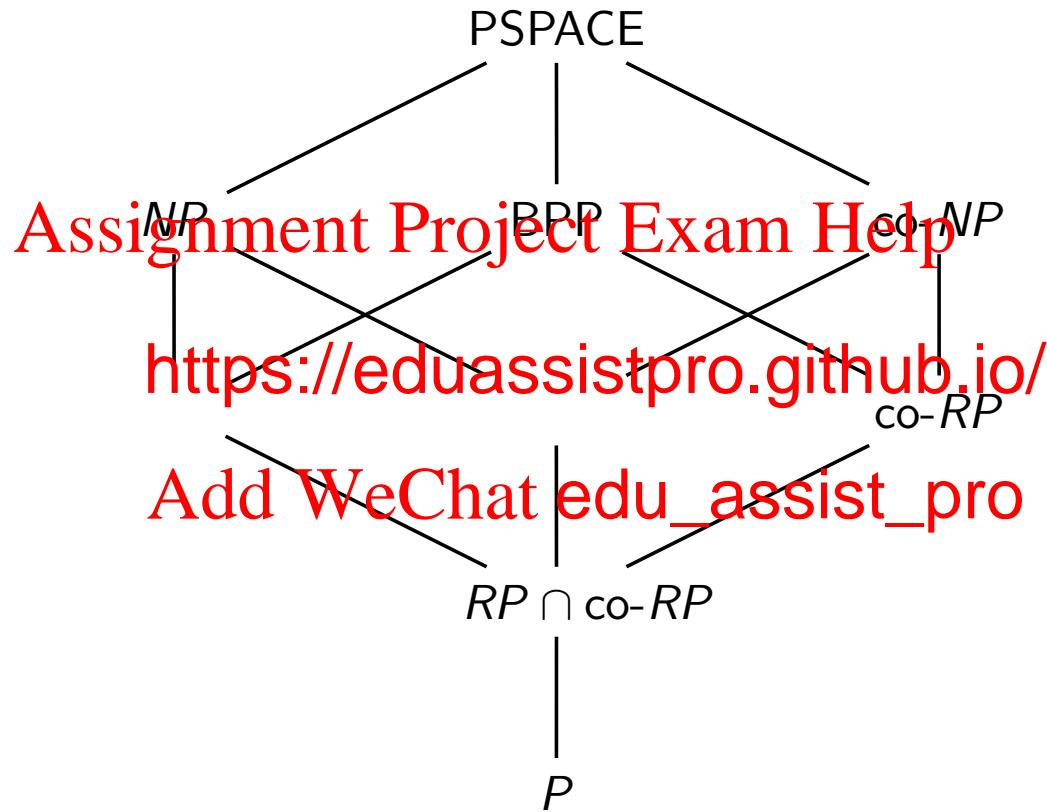
Assignment Project Exam Help
For example, is $RP = \text{co-}RP$?

Is $RP \subseteq NP \cap \text{co-}N$ <https://eduassistpro.github.io/>

Is $BPP \subseteq NP$? Add WeChat [edu_assist_pro](#)

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

Where Some Randomized Classes Fit



Summarising RP and co-RP

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

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

• $w \in L \Rightarrow \Pr[M \text{ rejects } w] = 0$

2

<https://eduassistpro.github.io/>

(co-RP) There is a P

$p'(n)$ such that

- $w \notin L \Rightarrow \Pr[M' \text{ acc}] = 1$
- $w \in L \Rightarrow \Pr[M' \text{ rej}] = 0$

In the first case, a ‘yes’ can be trusted.

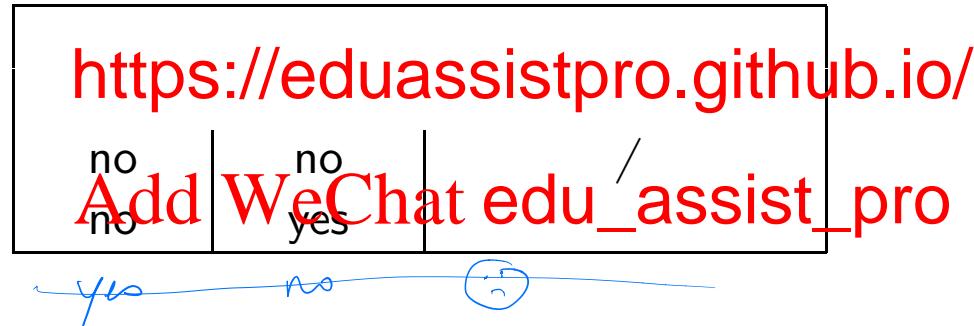
In the second case, a ‘no’ can be trusted.

Understanding RP \cap co-RP

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

There are three possible outcomes:

Assignment Project Exam Help



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

Monte Carlo and Las Vegas Methods

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 Monte Carlo algorithms.
<https://eduassistpro.github.io/>

A Las Vegas algorithm also uses randomization.

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

The Class ZPP

The Z in ZPP stands for “zero error”.

Assignment Project Exam Help

ZPP corresponds to the L

the correct answer, but ha
input), the expectati

Add WeChat edu_assist_pro

In the following several slides, we prove the interesting fact that $ZPP = RP \cap \text{co-}RP$.

$$\text{ZPP} \subseteq RP \cap \text{co-}RP$$

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.

Assignment Project Exam Help

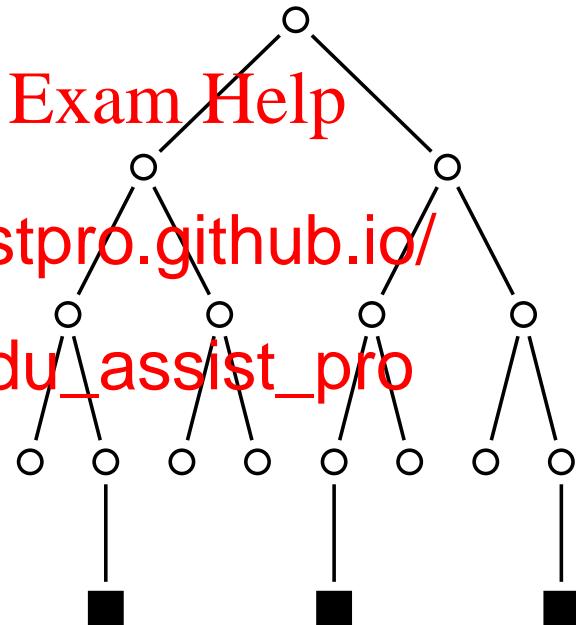
At a given polynomial depth, $p(n)$, we

may have subtrees of super

polynomial depth ([shhttps://eduassistpro.github.io/](https://eduassistpro.github.io/))

Let M' be the PTM which works like M , except (if it has 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 '?'.

Assignment Project Exam Help

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

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

Add WeChat [edu_assist_pro](https://eduassistpro.github.io/)

$$\Pr[t(n) \geq r(n)] \leq \frac{1}{n}.$$

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

$$X \geq 0$$

$$\Pr[X \geq a \cdot E[X]] \leq \frac{1}{a}.$$

$$Y = 0 \cdot \Pr[X < a \cdot E[X]] + 1 \cdot \Pr[X \geq a \cdot E[X]]$$

$$\Pr[Y] = a \cdot E[X] \cdot \Pr[X \geq a \cdot E[X]]$$

$$\Pr[Y] \geq a \cdot E[X] \cdot \Pr[X \geq a \cdot E[X]]$$

$$\frac{1}{a} \geq \Pr[X \geq a \cdot E[X]]$$

$E[Y] \leq E[X]$

$$E[Y] \leq E[X]$$

$$\Pr[t(n) \geq r(n)]$$

$$\leq \frac{1}{n}.$$

$$E[t(n)] = q(n)$$

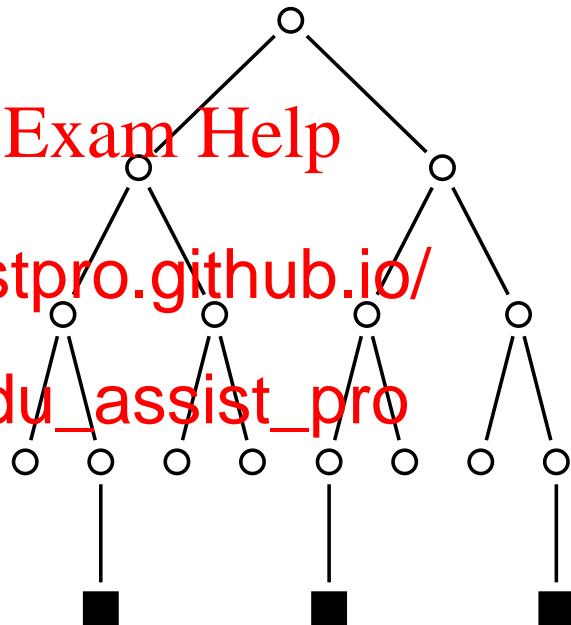
$$\Pr[t(n) \geq n \cdot q(n)] \leq \frac{1}{n}.$$

$$\text{ZPP} \subseteq RP \cap \text{co-}RP$$

Assignment Project Exam Help

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

Add WeChat [edu_assist_pro](https://eduassistpro.github.io/)



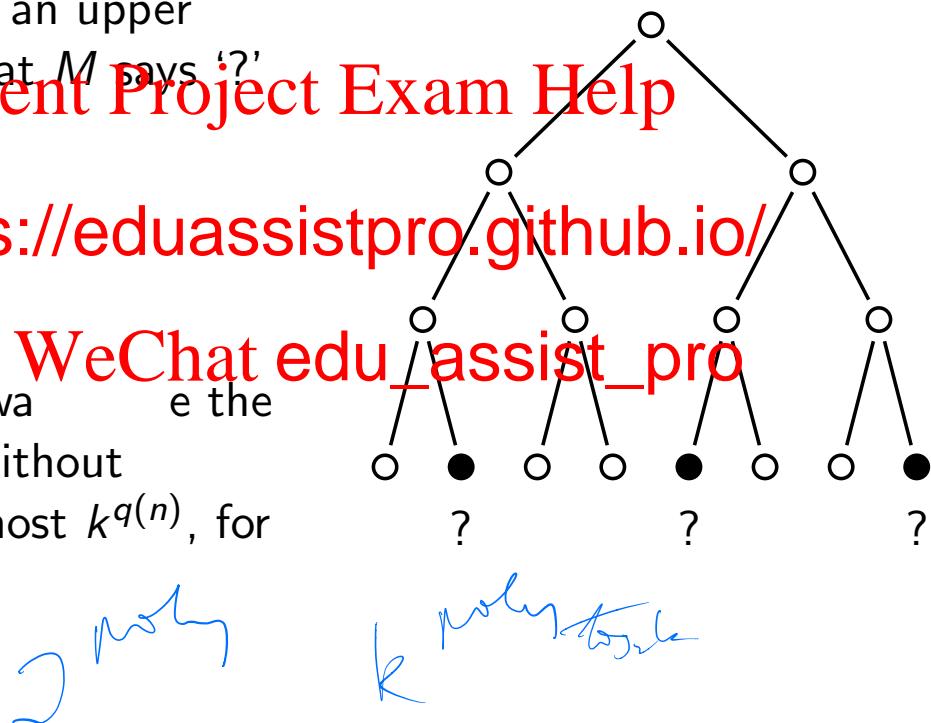
$$RP \cap \text{co-}RP \subseteq \text{ZPP}$$

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 polytime <https://eduassistpro.github.io/> time $r(n)$.

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

Assignment Project Exam Help



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

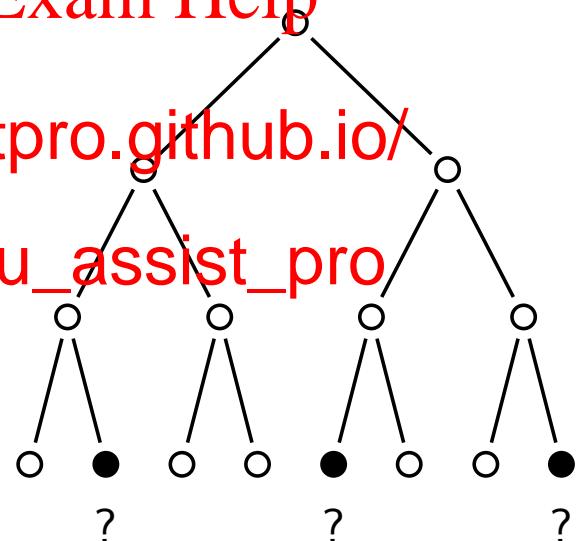
Assignment Project Exam Help

As soon as M provid

answer, machine M' <https://eduassistpro.github.io/> and halts.

Add WeChat `edu_assist_pro`

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



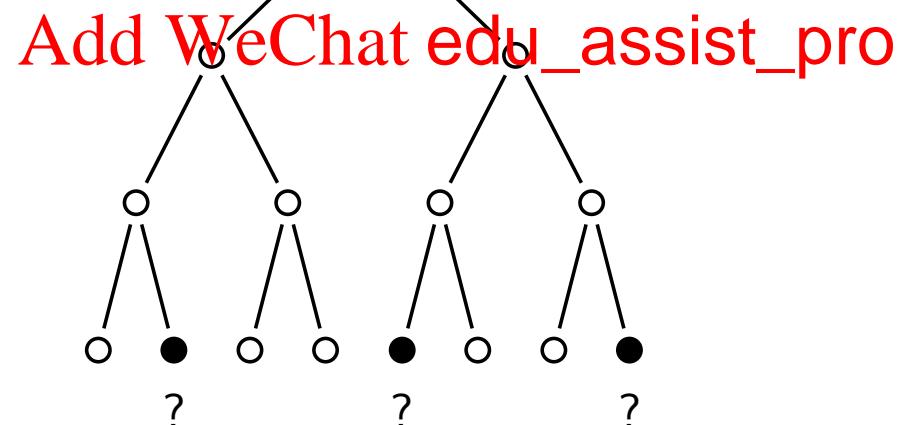
$$RP \cap \text{co-}RP \subseteq \text{ZPP}$$

The expected running time of M' is at most

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

which is polynomial in n .

<https://eduassistpro.github.io/>



Textbook Questions

Questions you can try at this point:

Assignment Project Exam Help
Problem 10.11

Problem 10.15

Problem 10.19

Problem 10.22

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

Space Complexity

Just like time, Assignment Project Exam Help

The space complexity positions it visits, as a function tape
<https://eduassistpro.github.io/>

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:

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro
N
PSPACE NPSPACE
EXPTIME NEXPTIME
EXPSPACE NEXPSPACE

:

SPACE and *NSPACE*

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

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

Assignment Project Exam Help

and

$$NSPACE(f(n)) = \left\{ L \mid \begin{array}{l} L \text{ is decided by some non-deterministic} \\ \text{Turing machine in } O(f(n)) \text{ space} \end{array} \right\}$$

Add WeChat edu_assist_pro

In analogy with *P* and *NP*,

$$\text{PSPACE} = \bigcup_k SPACE(n^k)$$

$$\text{NPSPACE} = \bigcup_k NSPACE(n^k)$$

SAT is in PSPACE

$\exists x_1 \exists x_2 \exists x_3 \exists x_4$

x_0
 x_1

SAT can be solved in linear space thus:

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

<https://eduassistpro.github.io/>

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

Add WeChat edu_assist_pro

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

Polynomial Space

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.

Assignment Project Exam Help

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

<https://eduassistpro.github.io/>

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

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

Assignment Project Exam Help

Question you can try at this

<https://eduassistpro.github.io/>

Exercise 8.4

Add WeChat edu_assist_pro

Savitch's Theorem

Perhaps surprisingly, $\text{PSPACE} = \text{NPSPACE}$.

To demonstrate this, we show that a deterministic TM can simulate an NTM in a fairly small amount of space.

<https://eduassistpro.github.io/>

Theorem: For a function $f : \mathbb{N} \rightarrow \mathbb{N}$ with

Add WeChat edu_assist_pro
 $\text{NSPACE}(f(n)) \subseteq S$

Node Reachability in Sub-Linear Space

A central tool for proving space-complexity results.

Path Existence

Assignment Project Exam Help

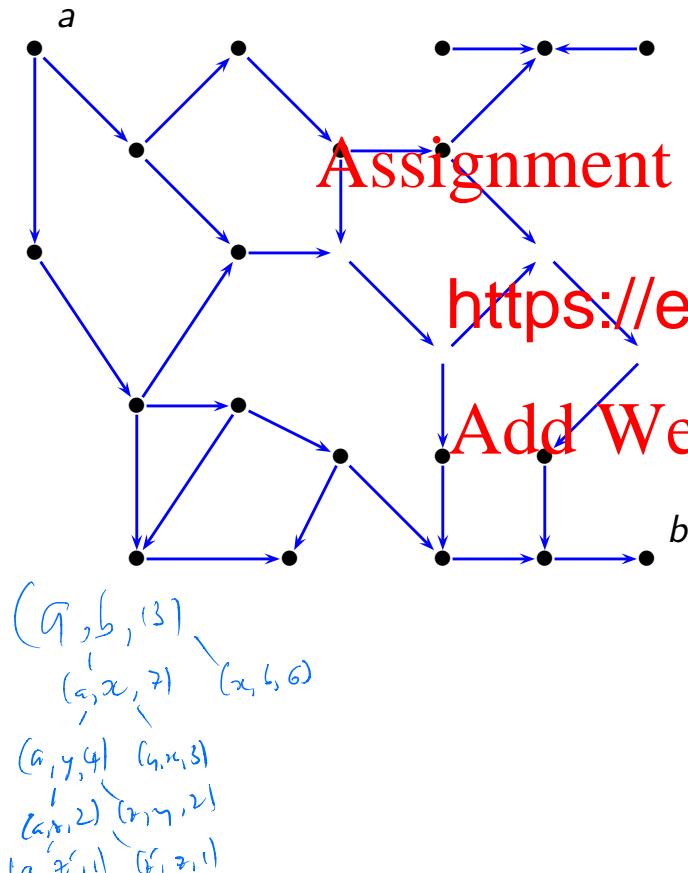
Given a graph G , n
reachable from node a in b steps

https://eduassistpro.github.io/

We can decide this in $O(n \log n)$ space exponential time

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

Node Reachability Example



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:

Assignment Project Exam Help

1 Go from a to x in 7 steps

o 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.

Savitch's Theorem

This insight leads us to the following

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

$$\text{NSPACE}(f(n)) \subseteq \text{SPACE}(f^2(n))$$

Suppose N is an NTM with

<https://eduassistpro.github.io/>

We construct a deterministic TM M to simulate

Add WeChat edu_assist_pro

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 .

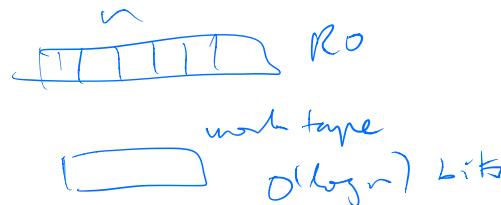
How Many Configurations?

Assuming

s states in Q

γ symbols in Γ

$f(n)$ cells touched on tape



Assignment Project Exam Help

there at most $s \cdot f(n)$

<https://eduassistpro.github.io/>

That is, for suitable constant d , no more than $s^{d f(n)}$ configurations. (We can achieve this by treating the presence of the tape as an extra symbol.)

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

Savitch's Theorem

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(c, c_1)
- ④ Run canyield(c, c_2)
- 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.

Savitch's Theorem

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

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

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

<https://eduassistpro.github.io/>

Therefore the overall machine M simulates $\text{SPACE}(f^2(n))$ space.

Add WeChat edu_assist_pro

Therefore for any function $f : \mathbb{N} \rightarrow \mathbb{N}$ we have,

$$\text{NSPACE}(f(n)) \subseteq \text{SPACE}(f^2(n))$$

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

Complexity Class Relations

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

Assignment Project Exam Help

$P \subseteq N$

XPTIME

<https://eduassistpro.github.io/>

We don't know which of th

Add WeChat edu_assist_pro

But the Time Hierarchy Theorem says that $\text{TIME} \subsetneq \text{DTime}$, so at least one of the inclusions must be strict.

PSPACE-Completeness

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

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

<https://eduassistpro.github.io/>

- ① $B \in \text{PSPACE}$, and [Add WeChat edu_assist_pro](#)
- ② B is PSPACE-hard.

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

Boolean Quantification

exists

We can add existential (\exists , “there exists”) and universal (\forall , “for all”) quantification to a propositional formula. For example,

<https://eduassistpro.github.io/>

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

Quantifier evaluation

We can expand these quantifiers thus:

$$\exists x [E] = \overbrace{E_{\text{false}} \vee E_{\text{true}}}^{\exists x [E]} \quad \forall x [E] = \overbrace{E_{\text{false}} \wedge E_{\text{true}}}^{\forall x [E]}$$

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

A fully quantified formula <https://eduassistpro.github.io/>

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

Order matters:

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

TQBF is PSPACE-Complete

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 rejection.
- ③ If $\varphi = \forall x[E]$, evaluate E_0 recursively. Accept iff both lead to acceptance (true).

Add WeChat edu_assist_pro

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 $TQBF \in \text{PSPACE}$.

TQBF is PSPACE-Complete

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 formula <https://eduassistpro.github.io/> accepts w .

Importantly, the formula, as well as the time to produce it, must 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.

TQBF is PSPACE-Complete

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

Assignment Project Exam Help

Configuration representation: assert that position y_{jA} , $0 \leq j \leq n^k$, that overloading of notation). Each configuration is polynomial size.

Add WeChat `edu_assist_pro`

Notation convenience: If $I = \{x_1, \dots, x_m\}$ ds for $\exists x_1 \dots \exists x_m$.

Similarly for $\forall I$.

TQBF is PSPACE-Complete

The formula φ will have the form

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

Assignment Project Exam Help

where

- I_s represents the initial configuration
- I_f represents accepting (final) configuration
- S expresses that the simulation starts correctly
- 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.

TQBF is PSPACE-Complete

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

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

It has $\neg y_{jA}$ otherwise.

<https://eduassistpro.github.io/>

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

Add WeChat [edu_assist_pro](#)

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.

TQBF is PSPACE-Complete

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 we let $I = J$ stand for

Add WeChat $\bigwedge_{j,A} \bigvee_{y_A} z_{jA}$ as variables z_{jA} , then

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

TQBF is PSPACE-Complete

$N_1(I, J)$ 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

Assignment Project Exam Help

we add the conjunct <https://eduassistpro.github.io/>

$$\bigwedge_{i=0}^{n^k-1} ((y_{iq} \wedge y_{(i+1)u}) \rightarrow (\text{Add WeChat edu_assist_pro}))$$

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.)

TQBF is PSPACE-Complete

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

Assignment Project Exam Help

$$N \quad J)] .$$

<https://eduassistpro.github.io/>

Unfortunately, this would not work well. Every time we halve i . Since M might run for an exponential number of steps , this can result in an exponential size formula, which is inefficient.

Add WeChat edu_assist_pro

TQBF is PSPACE-Complete

The trick is to apply quantifiers cleverly:

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

$$N_{2i}(I, J) = \exists K \left[I \wedge J \left[\begin{array}{l} \text{Assignment} \\ \text{Project} \\ \text{Exam} \\ \text{Help} \end{array} \right] \vee N_i(I', J') \right]$$

<https://eduassistpro.github.io/>

Recalling that $\neg a \vee b$ is equivalent to $a \rightarrow b$, the formula says **Add WeChat edu_assist_pro** quantified

- 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.

TQBF is PSPACE-Complete

Max number of time steps is 2^{dn^k} where d is some constant. Writing N_{2i} takes the time to write M , 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 is evaluated in polynomial time.

Add WeChat [edu_assist_pro](https://eduassistpro.github.io/)

The formula is true iff M accepts w .

Textbook Questions

Questions you can try at this point:

Assignment Project Exam Help

Exercise 8.6

Problem 8.8

Problem 8.9

Problem 8.11

Problem 8.12

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

The Complexity of Games

Interestingly, ~~TQBF~~ Assignment Project Exam Help gives us a handle on the complexity of various games (or their winning strategies) such as noughts-and-crosses (<https://eduassistpro.github.io/>)

Such games often have a polynomial-time reduction to ~~TQBF~~.
Add WeChat edu_assist_pro

The Formula Game

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

$\exists x \forall y \exists z [(x \vee y) \wedge (y \vee z) \wedge (\neg y \vee \neg z)]$
Assignment Project Exam Help

Players E and A take

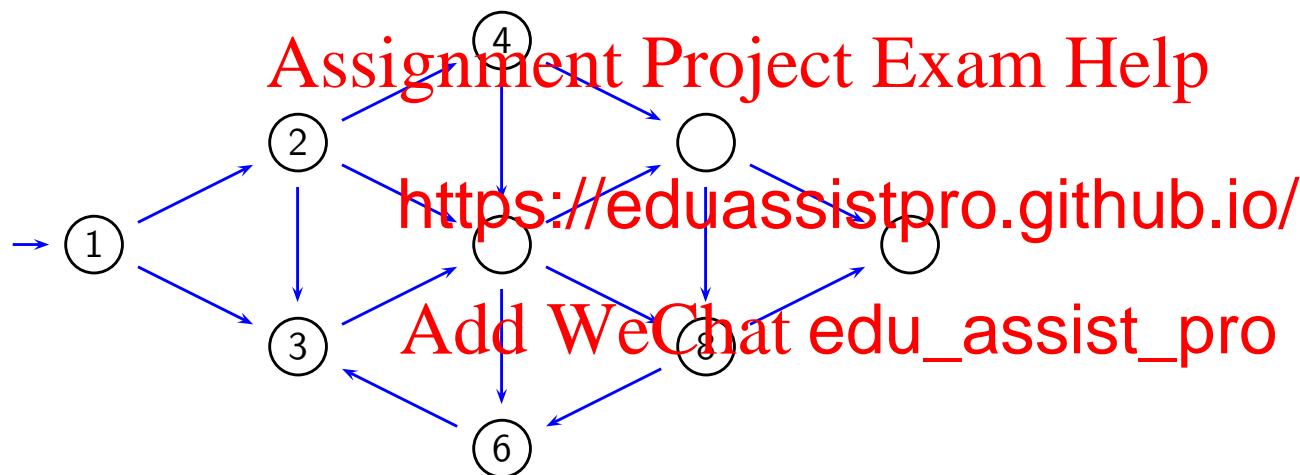
E chooses values for the existentially quantified variables and wins if the formula (without quantifiers) evaluates to **true**.
 A chooses values for the universally quantified variables and wins if the formula (without quantifiers) evaluates to **false**.

Add WeChat [edu_assist_pro](https://eduassistpro.github.io/)

This game is just $TQBF$ in disguise.

Other Games

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

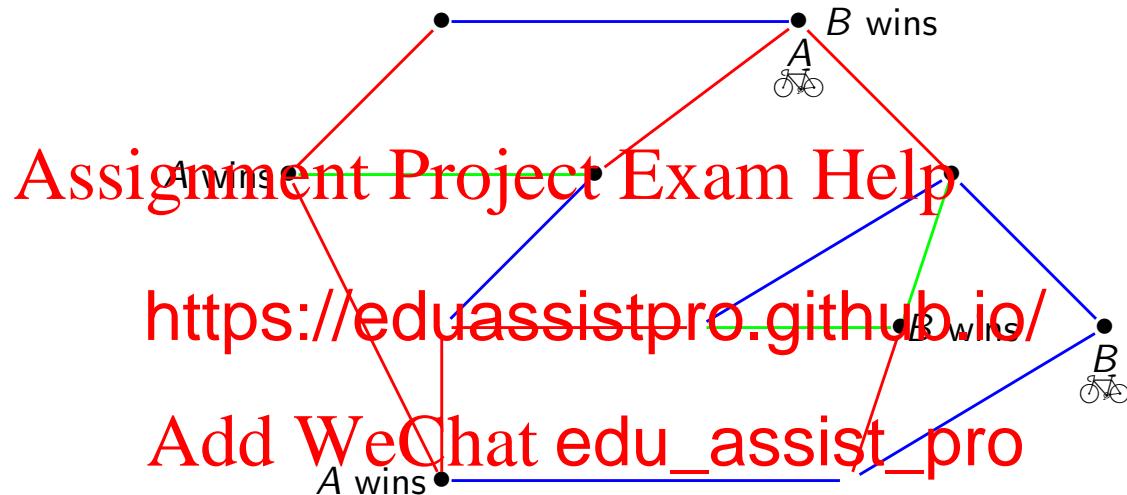
Problem 8.16

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

Provable Intractability: “Roadblock”

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.

Even Greater Intractability

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

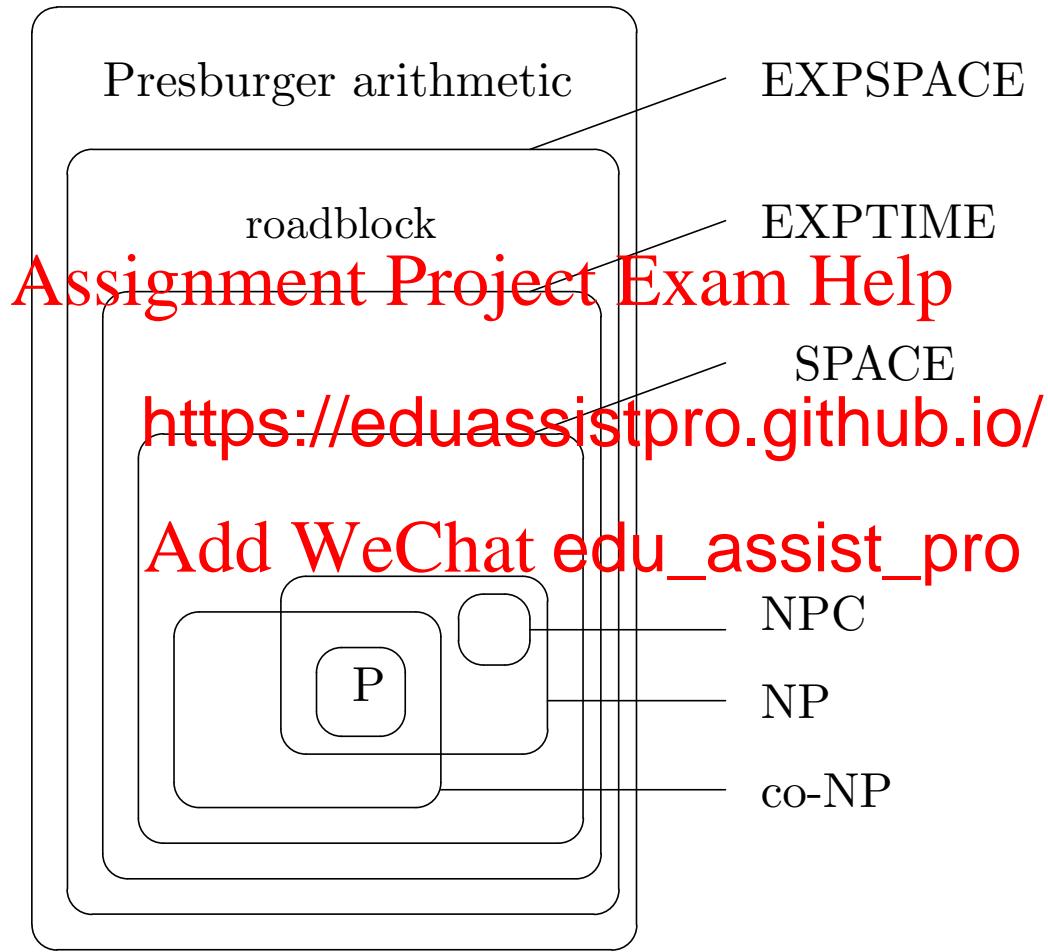
This generalises to k-EXPTIME.

The theory of natural numbers (arithmetic) is decidable

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

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

The Bigger Picture



Recap

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

Assignment Project Exam Help

Preparation

<https://eduassistpro.github.io/>

Read through the proof that GENERALI Y is
PSPACE-complete. Add WeChat edu_assist_pro

Sub-Linear Space

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. <https://eduassistpro.github.io/> now what that is any more?

Add WeChat `edu_assist_pro`

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

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

The Classes L and NL

Reading “decidable in logarithmic space” to mean “in logarithmic **working** space”, we can define

Assignment Project Exam Help

<https://eduassistpro.github.io/>

The focus on logarithmic behaviour is again justified by results: Logarithmic behaviour is preserved amongst undecidable variants of the Turing machine.

NL Example: PATH

We previously looked at the language

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

Assignment Project Exam Help
and noticed that $PATH$ is in P .

A nondeterministic TM can do this in 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.

Textbook Questions

Assignment Project Exam Help

Exercise 8.1

<https://eduassistpro.github.io/>

Exercise 8.5

Exercise 8.7

Problems 8.

Problems 8.17 – 8.25 Add weechat.org/~assist/pro

Problems 8.17 – 8.25 [Add WeChat add_assist_pro](#)

Log-Space Reduction

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

Assignment Project Exam Help

We need a concept of reduction. ^{time} reductions will not be very helpful.

<https://eduassistpro.github.io/>

Suppose we reduce A to B and we have a log-space reduction for B . Unless the reduction itself is log-space, that does not give us a log-space decision procedure for A .

Log-Space Reduction

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

<https://eduassistpro.github.io/>

The function f should be realised by a transducer (writing to a write-only tape) and we say that f is a log-space reduction from A to B .

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

Log-Space Reduction

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.

<https://eduassistpro.github.io/>

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

NL-Completeness

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

B is **NL-complete** iff

Assignment Project Exam Help

- ① $B \in NL$, an
- ② B is NL-hard.

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

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.

NL-Completeness

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

Assignment Project Exam Help

Given NTM M that decides A_{NTM} , we can construct $\langle G, s, t \rangle$ in $O(n)$ time. This graph contains a path from s to t if and only if M accepts w . The graph has n nodes and $O(n)$ edges. It is a directed graph where each node is a configuration of M and each edge is a transition between configurations. The graph is constructed in $O(n)$ space.

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

NL and other complexity classes

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 .

Assignment Project Exam Help
Hence we can extend our chain of inclusions to

$$L \subseteq NL \subseteq P$$

https://eduassistpro.github.io/

It is known that $NL \subsetneq PSPACE$, so in the chain

$$NL \subseteq P \subseteq NP \subseteq PSPACE$$

at least one of the inclusions is proper.

Regular Grammar Non-Emptiness Is NL -Complete

The language

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

is in NL . Namely, for each $w \in \Sigma^*$, we can build a directed graph with <https://eduassistpro.github.io/> node B .

The language generated by the grammar is non-empty if there is a path from S to some node C , where $C \rightarrow w$ is a path having 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.

Regular Grammar Non-Emptiness Is NL -Complete

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

To show NZ hardness, we reduce $\lambda A\lambda\forall\forall$ to $\lambda E\lambda G$:

Assignment Project Exam Help

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

This grammar generates $\{\epsilon\}$, the empty string, if there's a path from S to T , and \emptyset (that is, no strings) if no such path exists.

Assignment Project Exam Help

Questions you can try at thi

<https://eduassistpro.github.io/>

Problems 8.26 – 8.34

Add WeChat edu_assist_pro

NL and coNL

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-Szelepcsenyi Theorem, came as a surprise in 1988:

<https://eduassistpro.github.io/>

$NL = co$

Add WeChat edu_assist_pro

(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.

Assignment Project Exam Help

Again, we can model computationability, so

NL-completeness of <https://eduassistpro.github.io/>

Further reading

Add WeChat edu_assist_pro

The fact that $NL = co-NL$ is unexpected: r

tbook.