7.17 Show that, if P = NP, then every language $A \in \mathrm{P}$, except $A = \emptyset$ and $A = \Sigma^*$, is NP-complete.

Let A be any language in P except $A = \emptyset$ and $A = \Sigma^*$. To show that A is NP-complete, we show that $A \in$ NP and that every $B \in$ NP is polynomial time reducible to A. The first condition holds because $A \in$ P so $A \in$ NP. To demonstrate that the second condition is true, let $x_{\text{in}} \in A$ and $x_{\text{out}} \notin A$ be two strings that are guaranteed to exist by virtue of our assumptions about A. The assumption that P = NP implies that B is recognized by a polynomial time TM M. A polynomial time reduction from B to A simulates M to determine whether its input w is a member of B, then outputs x_{in} or x_{out} accordingly.

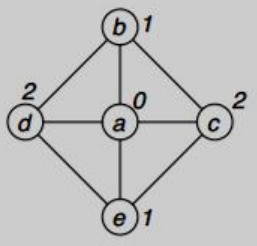
7.29 Consider the following scheduling problem. You are given a list of final exams $F_1, \ldots, F_k$ to be scheduled, and a list of students $S_1, \ldots, S_l$. Each student is taking some specified subset of these exams. You must schedule these exams into slots so that no student is required to take two exams in the same slot. The problem is to determine if such a schedule exists that uses only h slots. Formulate this problem as a language and show that this language is NP-complete.

Proof. SCHEDULE is in NP because we can guess a schedule with h slots and verify in polynomial time that no student is taking two exams in the same slot. We prove that SCHEDULE is NP-hard by showing that 3COLOR $\leq_P$ SCHEDULE. Given a graph $G = (V, E)$ we simply create an instance $\langle F, S, h \rangle$ where $F = V$, $S = E$, and $h = 3$ (which can obviously be done in polynomial time).

Given a 3-coloring for G, we can get a schedule for $\langle V, E, 3 \rangle$ by assigning finals that correspond to nodes colored with color i to slot i for $i = 0, 1, 2$. Adjacent nodes are colored by different colors, so corresponding exams taken by the same student are assigned to different slots.

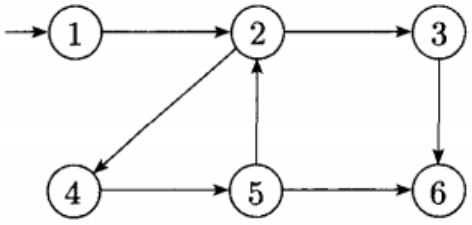
Given a schedule for $\langle V, E, 3 \rangle$, we can get a 3-coloring for the original graph by assigning colors according to the slots of corresponding nodes. Since exams taken by the same student are scheduled in different slots, adjacent nodes are assigned different colors. □

Below is an example 3-colored graph which divide the 5 finals into 3 slots for 8 students. Final a is scheduled into slot 0, b and e into slot 1, and c and d into slot 2.



F = {a, b, c, d, e}
S = {{a,b}, {a,c}, ... {e,c}}
h = 3

8.3 Consider the following generalized geography game wherein the start node is the one with the arrow pointing in from nowhere. Does Player I have a winning strategy? Does Player II? Give reasons for your answers.



Player 1 start the game from node 1 and then pass to player 2 this game win the player 2 because he have more nodes than player 1 coz player 1 stuck on 2 but 2 have nodes 3 to 6 and 4 to 5,6.

Isko thora r explain kr dena ap.

8.13 Define $A_{\text{LBA}} = \{\langle M, w\rangle | \ M \text{ is an LBA that accepts input } w\}$. Show that A_{LBA} is PSPACE-complete.

8.22

a. Let $ADD = \{\langle x, y, z\rangle | \ x, y, z > 0 \text{ are binary integers and } x + y = z\}$. Show that $ADD \in$ L.

b. Let $PAL\text{-}ADD = \{\langle x, y\rangle | \ x, y > 0 \text{ are binary integers where } x + y \text{ is an integer whose binary representation is a palindrome}\}$. (Note that the binary representation of the sum is assumed not to have leading zeros. A palindrome is a string that equals its reverse). Show that $PAL\text{-}ADD \in$ L.

Add wala question mje aya tha solve ni hy r Alba wale ka b ni hy

PLANROME SOL:

Palindrome

M1 = "On input string w:
1. Scan across the tape and look for 0s and 1s
2. Count number of 0s separately

3. Count number of 1s separately
4. compare both the counters if both are equal accept, otherwise reject

the machine counts the number of 0s and, separately, the number of 1s in binary on the work tape. The only space required is that used to record the two counters. In binary, each counter uses only logarithmic space and hence the algorithm runs in O(log n) space. Therefore, A ϵ L.

Q: STRONGLY-CONNECTED

Proof:
A language B is NL-complete if
1. B ϵ NL, and
2. every A in NL is log space reducible to B.
We know that PATH is NL-Complete. So we will construct a log space reduction from Strongly-connected to PATH. i.e
STRONGLY-CONNECTED <L PATH
Let say NTM M decides STRONGLY-CONNECTED in O(logn) space.
Functionality of M is as under
The nodes of G are the configurations of M on w.
For configuration c1 and c2 of M on w , (c1,c2) are the edges if c2 is the next possible configuration of M starting from c1.
Node s is the start configuration of M on w.
If machine reaches configuration t then accept.
Now to show that this operation has been performed in logarithmic time, we introduce a log space transducer from <G,s,t> on input w. We describe G by listing its nodes and edges. Listing of nodes is easy because each node is configuration of M on w and can be represented in clogn space for some constant c. The transducer possibally goes through all possible strings of length clogn.
Thus the STRONGLY-CONNECTED <L PATH.

Q: Pallindrome

M1 = "On input string w:
1. Scan across the tape and look for 0s and 1s
2. Count number of 0s separately
3. Count number of 1s separately
4. compare both the counters if both are equal accept, otherwise reject

the machine counts the number of 0s and, separately, the number of 1s in binary on the work tape. The only space required is that used to record the two counters. In binary, each counter uses only logarithmic space and hence the algorithm runs in O(log n) space. Therefore, A ϵ L.

Q: Mult-SAT

On input ϕ a non-deterministic TM can guess two assignments and accept both if both satisfy ϕ. Thus Double-SAT is NP. DOUBLE-SAT is mapping reducible to MULT_SAT. MULT-SAT ≤double_SAT. The following Turing machine R computes the assignments in polynomial time reduction.

R= "On input <ϕ>, a Boolean formula with variables x1,x2,. . ., xm

Let ϕ' be the ϕ □ (x v $\overline{x}$), where x is a new variable.

Output <ϕ'>"

If ϕ ϵ Double-SAT, ϕ' has at least two satisfying assignments because two satisfying conditions from the original documents of ϕ by changing the values of x.

Q: Clique reducibility to Half-clique

We give a polynomial time mapping reduction from CLIQUE to HALF-CLIQUE. The input to the reduction is a pair (G, k) and the reduction produces the graph (H) as output where H is as follows. If G has m nodes and k = m/2, then H =G. If k < m/2, then H is the graph obtained from G by adding j nodes, each connected to every one of the original nodes and to each other, where j = m − 2k.
Thus, H has m + j = 2m − 2k nodes. Observe that G has a k-clique iff H has a clique of size k + j = m − k, and so hG, ki 2CLIQUE iff hHi2HALF-CLIQUE.

If k > m/2, then H is the graph obtained by adding j nodes to G without any additional edges, where j = 2k − m. Thus, H has m + j = 2k nodes, and so G has a k-clique iff H has a clique of size k. Therefore, hG, ki 2 CLIQUE iff hHi2HALF-CLIQUE. We also need to show HALF-CLIQUE 2NP. The certificate is simply the clique.

Q: ISOMORPHIC…..

A nondeterministic polynomial time algorithm for *ISO* operates as follows:

"On input $\langle G, H \rangle$ where G and H are undirected graphs:

1. Let m be the number of nodes of G and H. If they don't have the same number of nodes, *reject*.
2. Nondeterministically select a permutation π of m elements.
3. For each pair of nodes x and y of G check that (x, y) is an edge of G iff $(\pi(x), \pi(y))$ is an edge of H. If all agree, *accept*. If any differ, *reject*."

Stage 2 can be implemented in polynomial time nondeterministically. Stage 3 takes polynomial time. Therefore $ISO \in$ NP.

In this question replace 1 with #....

U is in NP because on input $\langle d, x, 1^t \rangle$ a nondeterministic algorithm can simulate M_d on x (making nondeterministic branches when M_d does) for t steps and accept if M_d accepts. Doing so takes time polynomial in the length of the input because t appears in unary.

To show $3SAT \leq_P U$, let M be a NTM that decides $3SAT$ in time cn^k for some constants c and k. Given a formula ϕ, transform it into $w = \langle \langle M \rangle, \phi, 1^{c|\phi|^k} \rangle$. By the definition of M, $w \in U$ if and only if $\phi \in 3SAT$.

LPATH and UHMPATH question.

First, $LPATH \in$ NP because a nondeterministic machine can guess and verify a simple path of length at least k from a to b. Next, $UHAMPATH \leq_P LPATH$, because the following TM F computes the reduction f.

F = "On input $\langle G, a, b \rangle$, where G is an undirected graph, and a and b are nodes of G.

1. Let k be the number of nodes of G.
2. Output $\langle G, a, b, k \rangle$."

If $\langle G, a, b \rangle \in UHAMPATH$, G contains a Hamiltonian path of length k from a to b, thus $\langle G, a, b, k \rangle \in LPATH$. If $\langle G, a, b, k \rangle \in LPATH$, G contains a simple path of length k from a to b. Since G has k nodes, the path is Hamiltonian. Thus $\langle G, a, b \rangle \in UHAMPATH$.

The following is a nondeterministic Turing machine (NTM) that decides the *HAMPATH* problem in nondeterministic polynomial time. Recall that in Definition 7.9, we defined the time of a nondeterministic machine to be the time used by the longest computation branch.

N_1 = "On input $\langle G, s, t \rangle$, where G is a directed graph with nodes s and t:

1. Write a list of m numbers, $p_1, \ldots, p_m$, where m is the number of nodes in G. Each number in the list is nondeterministically selected to be between 1 and m.
2. Check for repetitions in the list. If any are found, *reject*.
3. Check whether $s = p_1$ and $t = p_m$. If either fail, *reject*.
4. For each i between 1 and $m - 1$, check whether (p_i, p_{i+1}) is an edge of G. If any are not, *reject*. Otherwise, all tests have been passed, so *accept*."

To analyze this algorithm and verify that it runs in nondeterministic polynomial time, we examine each of its stages. In stage 1, the nondeterministic selection clearly runs in polynomial time. In stages 2 and 3, each part is a simple check, so together they run in polynomial time. Finally, stage 4 also clearly runs in polynomial time. Thus, this algorithm runs in nondeterministic polynomial time.

CIS 511 Spring 2007: Homework 5 Solutions
Problem 1
We construct a formula ' that selects the k edges which are in the cut in such a way that ' is satis_able if and only if G has a cut of size k. The size of ' will be polynomial in jGj and k.
The variables pv correspond to vertices in V and pv set to 1 corresponds to v 2 U. Furthermore we use variables pie, for i 2 1; 2; : : :k and e 2 E. pie set to 1 indicates that e is the i-th edge in the cut.
Now we will construct formulas that say that there are exactly k edges in the cut.
'1 =^
i
(_
e
pie)
'2 =^
i
(^
e
(pie ! ^
e0 6=e
:pie0 ^ ^

j6=i
:pje))
'1 implies there are at least k edges in the cut and '2 implies that there are at most k edges in the cut.
Now we must ensure that the edges selected by '1 and '2 are in the cut, i.e. they have one endpoint in U and the other not in U.
'3 = ^
i
^
e=(u;v)
(pie ! (pu _ pv))
'4 = ^
e=(u;v)
((pu _ pv) ! _
i
pie)
'3 means that if pie is set then one of its endpoints is in U while the other is not. '4 means that, for each edge, if one of its endpoints is in U while the other is not, it must be the i-th edge in our cut for some i.
The conjunction '1 ^ '2 ^ '3 ^ '4 is thus satis_able if and only if there is a cut of size k.
The size of the formula is in O(k _ jEj2), since the size of '2 is in O(k _ jEj2) and the other formulas are smaller in size.

Problem 2

It can be easily seen that DOM-SET is in NP. The nondeterministic algorithm can guess which vertices are in the dominating set and then verify its guess was correct.
To show that DOM-SET is NP-hard, we reduce Node Cover to this problem. Let (G; k) be an instance Node Cover. We suppose without loss of generality that there are no isolated vertices in G, since these vertices do not inuence whether (G; k) is a \yes" instance. We show a polynomial algorithm that produces a graph G0 such that G has a node cover of size k i_ G0 has a dominating
1
set of size k. If G = (V;E), we de_ne G0 as follows:
G0 = (V 0 = V [fvab j (a; b) 2 Eg; E0 = E [f(a; vab); (b; vab) j (a; b) 2 Eg)
i.e. we add to G a vertex for each edge (a; b) and connect it to the vertices a and b.
First, consider the case when G has a node cover U of size k. We show that it is also a dominating set for G0. Consider a node in V . It is either in U or adjacent to a node in U, since U is a node cover in G. Consider a node in vab 2 V 0 n V . Either a or b must be in U, thus vab is adjacent to a node in U.
Second, consider the case when G0 has a dominating set W of size k. We start by constructing a dominating set W0 for G0 of size k such that it contains only nodes from V . If W contains a node vab, we can replace it by a (or b) and the set of vertices will remain a dominating set, since all the vertices adjacent to vab (i.e. a and b) will either be in W0 or adjacent to a vertex in W0. Now

we can prove that W0 is a vertex cover for G. Consider an edge (a; b). Either a or b are in W0 (since vab is adjacent to a node in W0), thus the edge (a; b) is covered.

Problem 3

First, we prove that the problem of equivalence of two formulas is in co-NP. We show an NP algorithm for the complement of the problem. It checks whether the input represents two well-formed formulas ' and . If not, the algorithm accepts immediately. If it is the case, the algorithm guesses an assignment and veri_es that ' is satis_ed while is not (or vice-versa). If P=NP, then co-NP=co-P=P. Thus P=NP implies that the problem of equivalence of two formulas is in P.

If equivalence is in P, we prove that MIN-FORM is in co-NP. Consider the following NP algorithm for the complement of MIN-FORM. It checks whether the input is a well-formed formula '. If it is not, the algorithm accepts immediately. If it is, the algorithm non-deterministically chooses a smaller formula and veri_es the equivalence of _ and and accepts if the two formulas are equivalent. The veri_cation of equivalence can be done in P (under the assumption P=NP). Thus if P=NP, MIN-FORM is in co-NP and we can conclude that P=NP implies **MIN-FORM** is in P.

Problem 4a

TRUE-SAT is in NP. We need only to test whether the expression is true when all variables are true (a polynomial-time, deterministic step) and then guess and check some other assignment. The complement of TRUE-SAT consists of all inputs that are not well-formed expressions, inputs that are well-formed expressions but that are false when all variables are true, and well-formed expressions that are only true when all variables are true. We will show TRUE-SAT is NP-complete, so it is unlikely that the complement is in NP.

2

To show TRUE-SAT is NP-complete, we reduce SAT to it. Suppose we are given an expression E with variables x1, x2, ..., xn. Convert E to E0 as follows:

1. First, test if E is true when all variables are true. If so, we know is E is satis_able, and so set E0 = E + y where y is a variable not already in E so that E0 will be accepted by TRUE-SAT.
2. Otherwise, let E0 = E + x1x2:::xn, a polynomial-time reduction. Note that E0 will always be true when all variables are true. If E is in SAT then it is satis_ed by some truth assignment other than all-true, thus E0 is in TRUE-SAT. Conversely, if E0 is in TRUE-SAT, then since x1x2:::xn is true only for the all-true assignment then E must be satis_able.

Thus we have a polynomial-time reduction from SAT to TRUE-SAT, therefore TRUE-SAT is NP-complete.

Problem 4b

FALSE-SAT is in NP. We can test whether the expression is false when all variables are false and then guess and check some other assignment. The complement of FALSE-SAT consists of all inputs that are not well-formed expressions, inputs that are well-formed expressions that are true when all variables are false, and well-formed expressions that are only false when all variables are false. We

will show FALSE-SAT is NP complete, so it is unlikely that the complement is in NP.

To show FALSE-SAT is NP-complete, we reduce TRUE-SAT to it. Suppose we are given an expression E with variables x1, x2, ..., xn. Convert E to E0 as follows:

1. Replace every instance of xi in E with :xi to obtain E00.
2. Set E0 = :E00. If E is in TRUE-SAT then E00 is true for the all-false assignment and some other assignment, thus E0 is false when all variables are false and for some other assignment. Conversely, if E0 is in FALSE-SAT then E00 is false for the all-true assignment and some other assignment, thus E is true for the all-true assignment and some other assignment.

Thus we have a polynomial-time reduction from FALSE-SAT to TRUE-SAT (which we proved above is NP-complete), therefore FALSE-SAT is NP-complete.

Problem 4c

DOUBLE-SAT is NP. We need only guess and check two assignments that satisfy the given expression. The complement of DOUBLE-SAT consists of all inputs that are not well-formed expressions and inputs that are well-formed expressions but have no more than one satisfying assignment. We will show that DOUBLE-SAT is NP complete, so it is unlikely that the complement is in NP.

To show DOUBLE-SAT is NP-complete, we reduce SAT to it. Given an expression E, we convert it to E0 as follows:

E0 = E ^ (y _ :y)

If E is in SAT then E0 must have at least two satisfying assignments, since we can take some satisfying assignment of E and apply it to E0, setting y to either true or false and E0 will be satis_ed. If E0 is in DOUBLE-SAT then E must have at least one satisfying assignment to allow E0 to be satis_ed. Since we have a polynomial-time reduction from SAT to DOUBLE-SAT then DOUBLE-SAT must be NP-complete.

Problem 4d

NEAR-TAUT is in co-NP. The complement of NEAR-TAUT consists of all inputs that are not well-formed expressions and inputs that are well-formed expressions such that there exist at least two non-satisfying assignments. To decide the complement in polynomial time we need only guess and check two assignments that do not satisfy the given expression. We will show that the complement is NP-complete, so it is unlikely that NEAR-TAUT is in NP.

To show the complement of NEAR-TAUT is NP-complete, we reduce SAT to it. Given an expression E, we convert it to E0 as follows:

E0 = :(E ^ (y _ :y))

If E is in SAT, then E0 must have at least two non-satisfying assignments, since we can take some satisfying assignment of E and apply it to E0, setting y to either true or false and E0 will not be satis_ed. If E0 is in the complement of NEAR-TAUT then it must have at least two non-satisfying assignments, thus E must have at least one satisfying assignment in order for the expression E ^ (y _ :y) to ever be true.