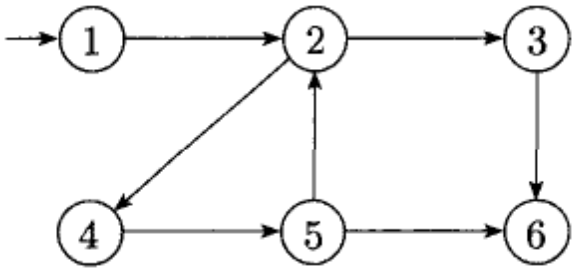
PAL-ADD wala kaafi easy hy woh yaad kareyn, LSPACE wala easy hy woh kareyn, NEAR-TAUT TRUE-SAT, DOBLE-SAT, 3SAT bilkul ready kar lyn in k aaney ki probability ziyaada hy as compared to others

Sehrish Aqeel 15/8/2017 8:00 am

Q:1 Show that, if P = NP, then every language A□ P, except A = Φ and A = Σ*, is NP-complete.

Q:2 Let ALBA = {<M,w> | M is an LBA (Linear Bounded Automaton) that accepts input w}. Show that ALBA is PSPACE-complete.

Q:3 Consider the following generalized geography game wherein the start node is the one with the arrow pointing in form nowhere. Does Player I have a winning strategy? Does Player II? Give reasons for your answers.




.
Q4 A subset of nodes of a graph G is dominating set if every other node of G is adjacent to some node in the subset.Let DOMINATING-SET={<G,k>| G has a dominating set with k nodes}.
Show that it is NP Complete by giving a reduction from VERTEX-COVER.

Shahzaib, [12.08.17 17:43]
A set X of vertices in an undirected graph G is called a dominating set if every vertex of G is either in X or at distance one from a vertex in X. The DOMINATING-SET problem is the decision problem for the language {(G,k): G is an undirected graph that has a dominating set of size k}. Prove that DOMINATING-SET is NP-complete. (Hint: Reduce from VERTEX-COVER.)
Q2 Let CNFk = {<φ>|φ is a satisfiable CNF formula where each variable appears in at most k places }. Show that CNF3 is NP-complete.
Let CNFk = {<$_{\varphi}$> | $_{\varphi}$ is satisfiable cnf-formula where each variable appears in at most k places}. Show that CNF3 is NP-Complete.
Let CNFk = {<ϕ> | ϕ is a satisfiable cnf-formula where each variable appears in at most k places}. Geography game

Another:
Geography game, LPATH, DEAR TAUT, TRUE SAT

Another: 3COLOR, 3CNF, NESTED PARANTHESIS, Geography game
Another: 3CNF, 2CNF, Add in LSpace, LBA
Another, M, x #, t wala, multi sat, 3 color, geography game

SEHRISH AQEEL

Show that A is NP Complete.
2. Show that Hamiltonin is np complete wala tha.
3. Aik Alba is Pspace complete wala.
4. Aik k tape to single tape turing machine wala.
Prove that Cycle-Length is NL-complete.

2. Let CNFk = {<$_{\varphi}$> | $_{\varphi}$ is satisfiablecnf-formula where each variable appears in at most k places}. Show that CNF3 is NP-Complete.
Let CNFk = {<$\phi$> | $\phi$ is a satisfiable cnf-formula where each variable appears in at most k places}.
(a) Show that CNF2 ∈ P.
(b) Show that CNF3 is NP-complete.

3. Let CONNECTED={<G>|G is connected undirected graph} show that it is in P.
CS 701 - Final Term 2015

Let DOUBLE-SAT = { hφi | φ is a Boolean formula with two satisfying assignments }.

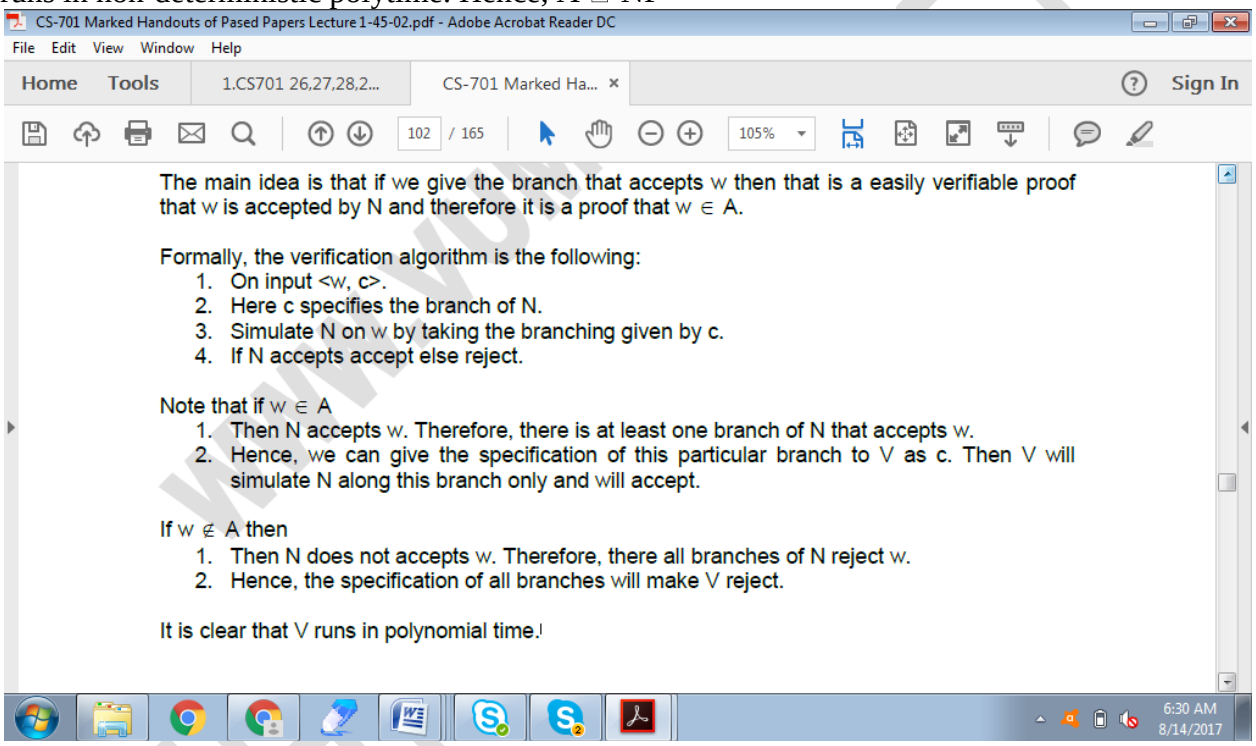
1 Show that DOUBLE-SAT is NP-Complete. (Hint: Reduce 3SAT.)

Answer:
(1) DOUBLE-SAT ∈ NP: Simply guess two different assignments to all variables and verify that each clause is satisfied in both cases.
(2) Reduction of 3SAT to DOUBLE-SAT: Given a 3cnf-function ψ, create a new Boolean function ψ 0 by adding a new clause (x ∪ x) to ψ, where x is a new variable not in ψ. Then check if hψ 0 i ∈ DOUBLE-SAT.
(3) This reduction clearly works in polynomial time.
(4) We now prove that the original 3cnf-function hψi ∈ 3SAT iff the new Boolean function hψ 0 i ∈ DOUBLE-SAT. If the original 3cnf-function ψ is unsatisfiable, then the new function ψ 0 is also unsatisfiable; i.e., hψi 6∈ 3SAT implies hψ 0 i 6∈ DOUBLE-SAT. If hψi ∈ 3SAT, then use the same assignment of variables that are in ψ, and we also have both x = 0 and x = 1 are valid assignments. Thus, there are at least two satisfying assignments of the augmented 3cnf-formula ψ 0 , so hψ 0 i ∈ DOUBLE-SAT.

1. Show that A is NP-Complete.
   Answer: To show that any problem A is NP-Complete, we need to show four things: (1) there is a non-deterministic polynomial-time algorithm that solves A, i.e., A ∈ NP, (2) any NP-Complete problem B can be reduced to A, (3) the reduction of B to A works in polynomial time, (4) the original problem A has a solution if and only if B has a solution. We now show that SET-PARTITION is NP-Complete.

SHIRISH AQEEL

NP is the class of languages that have polynomial time verifiers.
NP is the class of languages that are decided by polynomial time non-deterministic Turing machies.
1. On input w of length n.
2. Non-deterministically guess a string c of length $k n$ .
3. Run V on <w, c> if V accepts accept.
Clearly if w □ A then there is a c such that V accepts <w, c> so at least one branch of N will accept. On the other hand if w □ A then no branch of V will accept. Clearly, this Turing machine runs in non-deterministic polytime. Hence, A □ NP




The machine N will use V as a subroutine. Suppose V runs in time . k n
Here is a description of N:
1. On input w of length n.
2. Guess a certificate c of size at most . k n
3. Run V on <w, c>.
4. If V accepts accept else reject.

Theorem:
If A is NP-complete and A is in NP then P = NP.
So in order to solve P versus NP problems we will have two approaches. Suppose we find a NP-complete problem A. A is the hardest problem in NP. So, either we can
1. Try to prove that A is not in P. In fact, NP-complete problems are the best candidates for this.
2. Try to find a polynomial time algorithm for A. If we succeed then we will show that P = NP.

No one has been able to do any of the above!
This is a verification algorithm for SAT.
Theorem: (Cook Levin): SAT is NP-complete.
Theorem: (Cook Levin): P = NP if and only if SATP. □
In order to understand the Cook-Levin theorem we have to understand what can be expre

**Solution:**

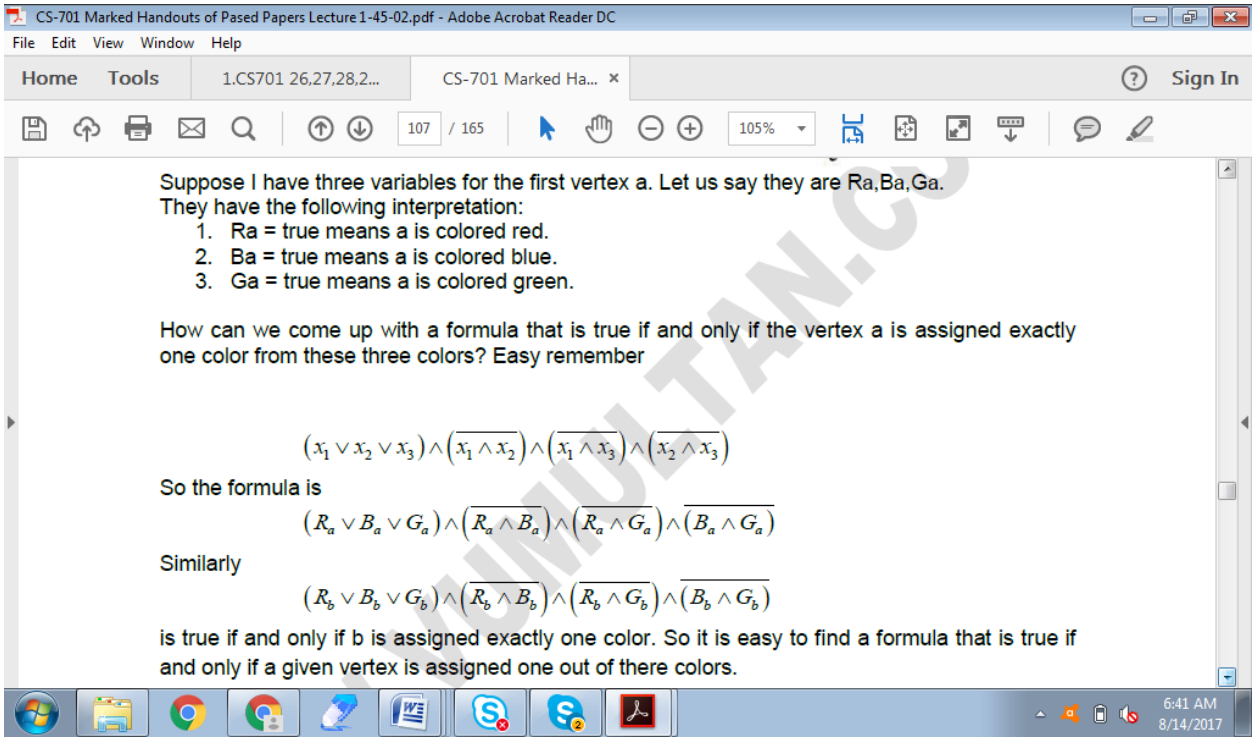
DOUBLE -SAT, like any variant of SAT, is in NP since the truth assignment is the certificate; we can check every clause in polynomial time to see if it is satisfied.

We now show a reduction from 3-SAT. Given a 3-SAT formula $f$, we construct a DOUBLE-SAT formula $f'=f\wedge(x_k \vee \neg x_k)$, where $x_k$ is a variable not used in $f$. Clearly this reduction is poly-time.

We now show that $f$ has a satisfying assignment iff $f'$ has two satisfying assignments.

$\Rightarrow$ If $f$ has a satisfying assignment $\sigma$, then $f'$ has two satisfying assignments, corresponding to $(\sigma,x_k=\text{true})$ and to $(\sigma,x_k=\text{false})$.

$\Leftarrow$ The clause $(x_k \vee \neg x_k)$ can be satisfied with either $x_k=\text{true}$ or $x_k=\text{false}$. In either case, if we have a satisfying assignment $\sigma$ for $f$, this ensures DOUBLE-SAT is validated. In this case, then 3-SAT is also validated.



CS-701 Marked Handouts of Pased Papers Lecture 1-45-02.pdf - Adobe Acrobat Reader DC
File Edit View Window Help
Home Tools 1.CS701 26,27,28,2... CS-701 Marked Ha... Sign In
108 / 165 105%

Let G = (V,E) be a graph and Consider the following formula:

$$\varnothing_G = \left(\bigwedge_{v\in V}\left(R_v \vee B_v \vee G_v\right)\wedge\left(\overline{R_v \wedge B_v}\right)\wedge\left(\overline{R_v \wedge G_v}\right)\wedge\left(\overline{B_v \wedge G_v}\right)\right)$$

$$\wedge$$

$$\left(\bigwedge_{\{a,b\}\in E}\left(\overline{R_a \wedge R_b}\right)\wedge\left(\overline{B_a \wedge B_b}\right)\wedge\left(\overline{G_a \wedge G_b}\right)\right)$$

This formula "says" that G is three colorable. It is satisfiable true if and only if we can find a 3Coloring of G.

Consider the algorithm:

1. On input G.
2. Compute ØG and output it.

This is a polynomial-time reducibility from 3Color to SAT.

6:44 AM 8/14/2017

CS-701 Marked Handouts of Pased Papers Lecture 1-45-02.pdf - Adobe Acrobat Reader DC
File Edit View Window Help
Home Tools 1.CS701 26,27,28,2... CS-701 Marked Ha... Sign In
117 / 165 105%

A formula in conjunctive normal form (CNF) is just an and of clauses. So it looks like this:

$$\left(x_1 \vee x_5 \vee \overline{x_7}\right)\wedge\left(\overline{x_1} \vee x_4 \vee \overline{x_6} \vee x_8\right)\wedge\left(x_3 \vee \overline{x_5} \vee \overline{x_{11}}\right)$$

We will say that a formula is in 3CNF if all the clauses have exactly three literals. Let us define the following language:

3SAT = { Ó : Ó is in 3CNF and Ó is satisfiable}.

We want to show that 3SAT is NP-complete.

1. We have to show that 3SAT is in NP. That is easy and is left as a homework exercise.
2. We will show that SAT $\leq_p$ 3SAT.

Let Ó be a formula. We will show how to construct a formula in 3CNF such that

1. Ψ is in 3CNF.
2. Ó is satisfiable if and only if Ψ is satisfiable.
3. Ψ can be computed from Ó in polynomial time.

Let us illustrate the proof of this theorem by an example. I will leave the details to you. Let us say we have a boolean formula Ó given by

6:52 AM 8/14/2017

CS701_Final_Exam_SolvedKHUSIA.pdf - Adobe Acrobat Reader DC
File Edit View Window Help
Home Tools 1.CS701 26,27,28,2... CS701_Final_Exam_... Sign In
53 / 65 66.7%

**Question: Let TRUE-SAT; given Boolean expression E that is true when all the variables are made true, is there some other truth assignment besides all true that make E true. Show that TURE-SAT is NP complete by reducing SAT to it.**

Answer:

To show TRUE-SAT is NP-complete, we reduce SAT to it. Suppose we are given an expression $E$ with variables $x_1, x_2, ..., x_n$. Convert $E$ to $E'$ as follows:

1. First, test if $E$ is true when all variables are true. If so, we know is $E$ is satisfiable, and so set $E' = E + y$ where $y$ is a variable not already in $E$ so that $E'$ will be accepted by TRUE-SAT.
2. Otherwise, let $E' = E + x_1x_2...x_n$, a polynomial-time reduction. Note that $E'$ will always be true when all variables are true. If $E$ is in SAT then it is satisfied by some truth assignment other than all-true, thus $E'$ is in TRUE-SAT. Conversely, if $E'$ is in TRUE-SAT, then since $x_1x_2...x_n$ is true only for the all-true assignment then $E$ must be satisfiable.

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

3:02 AM 8/14/2017

CS701_Final_Exam_SolvedKHUSIA.pdf - Adobe Acrobat Reader DC
File Edit View Window Help
Home Tools 1.CS701 26,27,28,2... CS701_Final_Exam_... Sign In
53 / 65 66.7%

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 $E'$ as follows:

$$E' = \neg(E \wedge (y \vee \neg y))$$

If $E$ is in SAT, then $E'$ must have at least two non-satisfying assignments, since we can take some satisfying assignment of $E$ and apply it to $E'$, setting $y$ to either true or false and $E'$ will not be satisfied. If $E'$ 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 $E \wedge (y \vee \neg y)$ to ever be true.

3:40 AM 8/14/2017

**Q15. Show that, if every NP-hard language is also PSPACE-hard, then PSPACE = NP.**

**Answer:.**

P $\leq$ NP $\leq$ P-SPACE
NP $\leq$ P-SPACE

Given every NP-Hard language is also PSPACE-Hard want to show that NP = PSPACE. From our assumption we know that if every NP-Hard is also PSPACE-Hard we know that then every NP-Complete language is also PSPACE-Hard since NP-Hard contains all of the NP-complete problems by definition. So we also know that SAT is PSPACE-Hard. And from the assumption that for any A in PSPACE, A reduces to SAT. Claim then we can solve A in NP. Create a TM, N as follows.

on input x, do
Compute f(x), the poly-time reduction between A and SAT.
Decide whether f(x) is satisfiable, if so, accept, otherwise reject.

Claim N decides A since x is in A iff f(x) is in SAT. Also notice that N is an NP machine since computing SAT is in NP.

**Q16.Define UCYCLE = {| G is an undirected graph that contains a simple cycle}. Show that UCYCLE belongs to NL.**

**Answer:.**

*Hint:* We can try to search the tree by always traversing the edges incident on a vertex in lexicographic order i.e. if we come in through the $i$th edge, we go out through the $(i+1)$th edge or the first edge if the degree is $i$. How does this algorithm behave on a tree? How about a graph with a cycle?

SOLUTION: Note that the above process performs a DFS on a tree and we always come back to a vertex through the edge we went out on. However, if the graph contains a cycle, there must exist at least one vertex $u$ and at least one starting edge $(u, v)$ such that if we start the traversal through $(u, v)$, we will come back to $u$ through an edge different that $(u, v)$. Hence, we enumerate all the vertices and all the edges incident on them, and start a traversal through each one of them. If we come back to the starting vertex through an edge different than the one we started on, we declare that the graph contains a cycle. Since we can enumerate all vertices and edges in logspace and also remember the starting vertex and edge using logarithmic space, this algorithm shows $UCYCLE \in \mathbf{L}$.

SEHAR AQEEL

**Q19: let EQnfa={<N,N'>|N,N' are NFA with the same alphabets and L(N)=L(N')} show that EQnfa ε PSPACE**

**Answer:>**

$$EQ_{NFA} = \{\langle N, N' \rangle \mid N, N' \text{ are NFAs with the same alphabet and } L(N) = L(N')\}$$

Show that $EQ_{NFA} \in$ **PSPACE**.
(*Hint:* Can you convert this to an appropriate reachability problem?)

SOLUTION OUTLINE: Suppose $N$ and $N'$ both have at most $n$ states. We can then convert them into DFAs $D_N$ and $D_{N'}$ with at most $m = 2^n$ states each using space polynomial in $n$. Finally, we can construct a DFA $S$, which is the product of $D_N$ and $D_{N'}$ (with at most $m^2 = 2^{2n}$ states) and accepts $L(D_N)\Delta L(D_{N'})$ (strings that are in exactly one of the languages). Now, $L(N) = L(N')$ iff $L(S) = \emptyset$ i.e. none of the final states are reachable from the start state in $S$.

Since this is a reachability problem, it can be decided nondeterministically using space logarithmic in the size of the graph (because $PATH \in$ **NL**). Thus, this problem can be decided in $NSPACE(\log(m^2)) = NSPACE(n) \subseteq SPACE(n^2) \subseteq$ **PSPACE**.

**Q25.Let A = {|NTM, M accepts input x within t steps on at least one branch}. Show that A is NP-Complete. Here A= U**

**Answer:.**

SOLUTION: Given any NP language $L$, we have an NDTM $M_L$ such that $\forall x \in L$, $M_L$ accepts $x$ on at least one branch in at most $p_L(|x|)$ steps, where $p_L()$ is a fixed polynomial depending on the machine. Also, $M_L$ does not accept any $x \notin L$. Then, given $x$, we create $y = \langle M_L, x, \#^{p_L(|x|)} \rangle$ in polynomial time. By the previous argument, $x \in L$ iff $y \in U$. Thus, $U$ is NP-hard.

To show that $U$ is also in NP, we can create an NDTM $M_U$, which given an input $u = \langle M, x, \#^t \rangle$, simulates $M$ on $x$ for $t$ steps. $M_U$ nondeterministically guesses all the branches of $M$ and accepts $u$ iff $M$ accepts $u$. Since the input has length at least $t$ and we simulate $M$ for at most $t$ steps, the running time is polynomial in the length of the input (note this is the reason we need $t$ in unary). It is easy to see that $M_U$ accepts exactly the language $U$, thus proving $U \in NP$. Hence, $U$ is NP-complete.

**Question: Define CYCLE = {(G)| G is a directed graph that contains a directed cycle}.**
**OR Show that CYCLE-Length problem is NL-Complete where Cycle Length is a problem where a cycle of nodes in a directed graph has at least 3 nodes.**
**Show that CYCLE is NL-Complete?**
**Answer:**
We reduce PATH to CYCLE. The idea behind the reduction is to modify the PATH problem instance (G, s, t) by adding an edge from *t* to *s* in G. If a path exists from s to t in G, a directed cycle will exist in the modified G.
However, other cycles may exist in the modified G because they may already be present in G. To handle that problem, we first change G so that it contains no cycles. A leveled directed graph is one where the nodes are divided into groups, A1, A2, . . . , Ak, called levels, and only edges from one level to the next higher level are permitted. Observe that a leveled graph is acyclic.

The PATH problem for leveled graphs is still NL-complete, as the following reduction from the unrestricted PATH problem shows. Given a graph G with two nodes s and t, and m nodes in total, produce the leveled graph G´ whose levels are m copies of G's nodes.
Draw an edge from node i at each level to node j in the next level if G contains an edge from i to j. additionally, draw an edge from node i in each level to node i in the next level. Let s´ be the node s in the first level and let t´ be the node t in the last level. Graph G contains a path from s to t iff G´ contains a path from s´ to t´.
If we modify G´ by adding an edge from t´ to s´ we obtain a reduction from PATH to CYCLE. The reduction is computationally simple, and its implementation in logspace is routine. Furthermore, a straightforward procedure shows that CYCLE ∈ NL.
Hence CYCLE is NL-complete.

**Q1. Let CONNECTED={<G>|G is connected undirected graph}. Show that it is in P**

**Class *P*:** $P$ is a class of languages that are decidable in polynomial time on a deterministic single – tape Turing – machine.

Specified language is,
$\text{CONNECTED} = \{\langle G\rangle \mid G \text{ is a connected graph}\}$

Now we have to show that CONNECTED is in $P$.

Given algorithm for determining the language CONNETED is as follows:
Let $M$ be the Turing – machine that determines language CONNECTED $M$ can be described as follows:
$M =$ " On input $\langle G\rangle$ :

1. Select the first node of graph $G$ and mark it.
2. Reject the following stage until no new nodes are marked.
3. For each node in $G$, mark it if it is attached by an edge to a node that is already marked.
4. Scan all the nodes of $G$ to determine whether they all are marked. If they are accept, otherwise, reject"

- To select the start node and to mark the start node the algorithm will take $O(n)$ time.
- So $O(n)$ time is taken by the algorithm to complete stage
- At mast $n+1$ repetitions are caused at stage 2. Because each repetition except the last marks at least one additional node.
- So the algorithm will take $O(n)$ time to complete stage 2.
- In stage 3, the algorithm has to check at most $n$ nodes of $G$ and for each checked node, the algorithm has to examine all adjacent nodes to see whether any have been marked. That will take at most $O(n^2)$ times
- So in total stage 3 takes $o(n)+o(n^2)=o(n^3)$ time.
- Therefore in total stage 2 and stage 3 take $o(n^3)$ time .
- Therefore in total stage 2 and stage 3 take $o(n^4)$ time as stage 3 and stage 2 are combined.
- In sage 4 the algorithm has to scan all the nodes that will take $o(n)$ time.
- Hence the algorithm runs in $o(n^4)$ polynomial time.

**Question: Prove that PAL-ADD is in Space| Show that PAL-ADD Є LSPACE?**
**Let PAL-ADD = {<x, y> | x, y > 0 are binary integers where x + y is an integer whose binary representation is palindrome}. Show that PAL-ADD Є LSPACE.**
**Answer:**
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 the number of 1s in binary on the work tape separately.
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 0(log n) space.
Therefore, A Є L.

Question:
**(a):** Let ADD = {(x,y,z) | x,y,z > 0 are binary integers and x + y = z}. Show ADD Є L

The class $L$: $L$ is the class of languages that are decidable in logarithmic space on a deterministic truing machine.
That is, $L = SPACE(\log n)$

(a) Given Language is
$ADD = \{\langle x, y, z\rangle \mid x, y, z > 0$ are binary integers and $x + y = z\}$

We have to show that $ADD \in L$.
That means, we have to construct a deterministic Turing machine $(DTM)$ that decides $ADD$ in logarithmic space.

Let $M_1$ be the $DTM$ that decides $ADD$ in logarithmic space.
The construction of $M_1$ is as follows:
$M_1$ = "On input $\langle x, y, z\rangle$:
1. If either of the three strings is not a binary number in the sense defined above then reject.
2. Initialize three binary counters $i, j, k$ pointing to the right – most of $x, y$ and respectively.
3. Perform binary addition (bit – wise long addition) using $i, j, k$ and a carry flag.
4. If any discrepancy arises between the calculated next bit of $z$ and the actual next bit of $z$, then reject.
5. If the end of all three number representations is reached and no error detected, then accept. So clearly $M_1$ runs in log space (it uses 3 counter only) and decides $ADD$".
Thus $ADD$ is in $L$. So, $ADD \in L$.

:

**(b):** Let PAL-ADD = {(x,y) | x,y > 0 are binary integers where x+y is an integer whose binary representation is palindrome}. Show that PAL-ADD Є L?

(b) Given language is

$PAL\_ADD = \{\langle x,y\rangle \mid x,y > 0$ are binary integers where $x+y$ is an integer whose binary representation is a palindrome$\}$

We have to show that $PAL\_ADD \in L$.

That means, we have to construct a *DTM* that decides $PAL\_ADD$ in logarithmic space.

Let $M_2$ be the *DTM* that decides $PAL\_ADD$ in logarithmic space.

The construction of $M_2$ is as follows.

$M_2$ = "On input $\langle x,y\rangle$:

1. If either of the two strings is not a binary number in the sence defined above, then reject
2. Compute the length $l$ of $x+y$ in binary.
3. For $i = 1$ to $l/2$.
   (i) Compute the $i^{th}$ bit $a$ of $x+y$
   (ii) Compute the $(l-i+1)^{th}$ bit $b$ of $x+y$
   (iii) if $a \neq b$ then reject
4. otherwise, accept"

Clearly $M_2$ runs in logspace and decides $PAL\_ADD$.

Thus $PAL\_ADD$ is in L. So, $PAL\_ADD \in L$

**Let CONNECTED={|G is connected undirected graph}. Show that it is in P.**

**Answer:.**

**Class *P*:** *P* is a class of languages that are decidable in polynomial time on a deterministic single – tape Turing – machine.

Specified language is,

CONNECTED $= \{\langle G\rangle \mid G$ is a connected graph$\}$

Now we have to show that CONNECTED is in *P*.

Given algorithm for determining the language CONNETED is as follows:
Let *M* be the Turing – machine that determines language CONNECTED *M* can be described as follows:

*M* = " On input $\langle G\rangle$:

1. Select the first node of graph *G* and mark it.
2. Reject the following stage until no new nodes are marked.
3. For each node in *G*, mark it if it is attached by an edge to a node that is already marked.
4. Scan all the nodes of *G* to determine whether they all are marked. If they are accept, otherwise, reject"

- To select the start node and to mark the start node the algorithm will take $O(n)$ time.
- So $O(n)$ time is taken by the algorithm to complete stage
- At mast $n+1$ repetitions are caused at stage 2. Because each repetition except the last marks at least one additional node.
- So the algorithm will take $O(n)$ time to complete stage 2.
- In stage 3, the algorithm has to check at most $n$ nodes of *G* and for each checked node, the algorithm has to examine all adjacent nodes to see whether any have been marked. That will take at most $O(n^2)$ times
- So in total stage 3 takes $o(n)+o(n^2)=o(n^3)$ time.
- Therefore in total stage 2 and stage 3 take $o(n^3)$ time.
- Therefore in total stage 2 and stage 3 take $o(n^4)$ time as stage 3 and stage 2 are combined.
- In sage 4 the algorithm has to scan all the nodes that will take $o(n)$ time.
- Hence the algorithm runs in $o(n^4)$ polynomial time.
- Therefore CONNECTED is in *P*.

7.5 Is the following formula satisfiable?

$$(x \vee y) \wedge (x \vee \overline{y}) \wedge (\overline{x} \vee y) \wedge (\overline{x} \vee \overline{y})$$

**Solution**

**The formula is not satisfiable. For any assignment of the boolean values for $x$ and $y$, it always makes one of the four clauses false. Therefore, the formula, which is a conjunction of the four clauses, is always false for any assignment of $x$ and $y$.**

7.7 Show that NP is closed under union and concatenation.

**Solution**

NP is closed under union. For any two NP-languages $L_1$ and $L_2$, let $M_1$ and $M_2$ be the NTMsthat decide them in polynomial time. We construct a NTM $M'$ that decides the union of $L_1$ and $L_2$ in polynomial time:

$M'$ = "On input $\langle w \rangle$:

1. Run $M_1$ on $w$. If it accepts, *accept*.
2. Run $M_2$ on $w$. If it accepts, *accept*. Otherwise, *reject*."

In both stages 1 and 2, $M'$ uses its nondeterminism when the machines being run make nondeterministic steps. $M'$ accepts $w$ if and only if either $M_1$ and $M_2$ accept $w$. Therefore, $M'$ decides the union of $L_1$ and $L_2$. Since both stages take polynomial time, the algorithm runs in polynomial time.

NP is closed under concatenation. For any two NP-languages $L_1$ and $L_2$, let $M_1$ and $M_2$ be the NTMsthat decide them in polynomial time. We construct a NTM $M'$ that decides the concatenation of $L_1$ and $L_2$ in polynomial time:

$M'$ = "On input $\langle w \rangle$:

1. For each way to cut $w$ into two substrings $w = w_1 w_2$:
2. Run $M_1$ on $w_1$.
3. Run $M_2$ on $w_2$. If both accept, *accept*; otherwise continue with the next choice of $w_1$ and $w_2$.
4. If $w$ is not accepted after trying all the possible cuts, *reject*."

**In both stages 2 and 3, $M'$ uses its nondeterminism when the machines being run make nondeterministic steps. $M'$ accepts $w$ if and only if $w$ can be expressed as $w_1 w_2$ such that $M_1$ accepts $w_1$ and $M_2$ accepts $w_2$. Therefore, $M'$ decides the concatenation of $L_1$ and $L_2$. Since stage 2 runs in polynomial time and is repeated for at most $O(n)$ time, the algorithm runs in polynomial time.**

**A7.15** Show that NP is closed under the star operation.

**Solution:**

The Kleene star of a language $L$ is $L^* = \{x_1 \dots x_k \mid k \geq 0 \text{ and } x_1, \dots, x_k \in L\}$. Our goal is to show that **P** and **NP** are closed under the Kleene star. That is, if $L \in \mathbf{P}$, then $L^* \in \mathbf{P}$. Similarly for **NP**.

Suppose that $L \in \mathbf{NP}$, i.e., $L$ is decided by a non-deterministic Turing machine $M$ in polynomial time. Then $L^*$ is decided by a non-deterministic Turing machine $M'$ in polynomial time, as described in the following. Assume an input string $x$. If $x = \varepsilon$, then $M'$ accepts. Otherwise, $M'$ splits $x$ non-deterministically into at most $|x|$ non-empty strings $x'_1, \dots, x'_k$ such that $x'_1 \dots x'_k = x$. Then $M'$ runs $M$ as a subroutine on each of the strings and accepts iff each subroutine run ended in a "**yes**"-state. It is obvious from the construction that $\mathcal{L}(M') = L^*$ and that $M'$ works in non-deterministic polynomial time. Thus **NP** is closed under Kleene star.

**7.21** Let *DOUBLE-SAT* $= \{\langle\phi\rangle \mid \phi \text{ has at least two satisfying assignments}\}$. Show that *DOUBLE-SAT* is NP-complete.

**Solution:**

**On input $\phi$, a nondeterministic polynomial time machine can guess two assignments and accept if both assignments satisfy $\phi$. Thus *DOUBLE-SAT* is in NP. We show *SAT* $\leq_P$ *DOUBLE-SAT*. The following TM $F$ computes the polynomial time reduction $f$.**

**$F$ = "On input $\langle\phi\rangle$, a Boolean formula with variables $x_1, x_2, \dots, x_m$.**

1. **Let $\phi'$ be $\phi \wedge (x \vee \overline{x})$, where $x$ is a new variable.**
2. **Output $\langle\phi'\rangle$"**

**If $\phi \in$ *SAT*, $\phi'$ has at least two satisfying assignments because we can obtain two assignments from the original assignment of $\phi$ by changing the value of $x$. If $\phi' \in$ *DOUBLE-SAT*, $\phi$ is also satisfiable, because $x$ does not appear in $\phi$. Therefore $\phi \in$ *SAT* if and only if $f(\phi) \in$ *DOUBLE-SAT*.**

**7.16** Let *UNARY-SSUM* be the subset sum problem in which all numbers are represented in unary. Why does the NP-completeness proof for *SUBSET-SUM* fail to show *UNARY-SSUM* is NP-complete? Show that *UNARY-SSUM* $\in$ P.

**Solution:**

*Answer.* The crux of the question is that we measure complexity as a function from the size of the input to the number of steps required to run the decided on that input. When the input is represented in unary, the size of the input is equal to the value (that is, it takes $x$ squares to represent the input value $x$). When the input is represented in binary (or any higher base), it takes $\log x$ squares to represent the input value $x$. Hence, the if size of the input in binary is $N$, the size of the same input in unary is $2^N$. This, the size of the input increases exponentially, even though the problem is of the same difficulty as it would be for the corresponding binary input. The reduction proof fails because the reduction from $3SAT$ to *UNARY_SSUM* requires more than polynomial time. The input size increases exponentially, so the transformation to generate the corresponding input for *UNARY_SSUM* requires a number of steps that is exponential in the size of the input.

b. Show that $UNARY\_SSUM \in$ P.

*Answer.* To show a language is in **P**, we need to show there is a polynomial-time algorithm that decides it. Note the we can simulate a nondeterministic TM in exponential time using a deterministic TM. Hence, if *SUBSET_SUM* is in **NP**, we know *UNARY_SSUM* (the same problem, but with exponentially larger input size) is in **P**, since we can solve it by first converting the input to binary representation (which can be done in polynomial time) and shrinks the input size to $\log N$. Then, we can simulate the nondeterministic polynomial-time TM running on this input (which has size $\log N$) in time that is exponential in its input size, which is polynomial in $\log N$.

# Previous Final Term Papers Solved

**Aamir Mustafa**
**MS150400200**

**Q:1 Show that, if P = NP, then every language A ∈ P, except A = Φ and A = ∑*, is NP-complete.**

**Answer:**

Step 1 of 2

***NP* – complete:**

A language $B$ is $NP$ – complete if it satisfies two conditions

1. $B$ is in $NP$ and
2. Every $A$ is $NP$ is polynomial time reducible to $B$.

Step 2 of 2

If 2nd condition is satisfied then $B$ is in $NP$ – hard.

**1.** $A$ is in $NP$: Suppose that $P = NP$ and

Let $A \in P$ be some language that is not in $\phi$ (empty set) or $\Sigma^*$ (infinite set).

Clearly $P = NP$ and $A \in P$ implies that $A \in NP$.

**2.** $A$ is $NP - hard$ :

- Now, we have to show that every $B \in NP$ is polynomial time reducible to $A$.
- Consider arbitrary $B \in NP$, nothing that $B \in P$
- To test if $w \in B$, run the polynomial time algorithm $N$ deciding $B$.
- If $N$ answers yes, out constant $w_e \in A$.
- Otherwise, output constant $w_e \in A$.
- Strings $w_e$ and $w_e$ necessarily exist because both $A$ and $\Sigma^* - A$ are non empty.
- This algorithm runs in polynomial time because $N$ runs in polynomial time and the length of $w_e$ and $w_e$ are constant.
- Therefore, $B$ is polynomial time reducible to $A$.

**Therefore, *A* is in *NP*- complete.**

**Q:2 Let ALBA = {<M,w> | M is an LBA (Linear Bounded Automaton) that accepts input w}. Show that ALBA is PSPACE-complete.**

**Answer:**

Step 1 of 4

$PSPACE$ – complete : A language $B$ is $PSPACE$ – complete if it satisfies two conditions.

1. $B$ is in $PSPACE$, and
2. every $A$ in $PAPACE$ is polynomial time reducible to $B$.

If $B$ satisfies condition 2, we say that $B$ is $PSPACE$- hard

Step 2 of 4

Given language is

$A_{LBA} = \{(M, w)) M \text{ is an } LBA \text{ that accepts input } w\}$

A linear bound automation $(LBA)$ is one – tape, one – head $NTM$.

We need to show that $A_{LBA}$ is $PSPACE$ – complete.

That means $A_{LBA}$ has to satisfy the 2 conditions of $PSPACE$-complete.

**Step 3** of 4

(i) $A_{LBA} \in PSPACE$:

To show $A_{LBA} \in PSPACE$, we need to construct a deterministic Turing machine that decides $A_{LBA}$ in polynomial space.

Let $T$ be the Turing machine $(TM)$ that decides $A_{LBA}$ in polynomial space.

$T$ can be constructed as follows.

$T =$ " on input $\langle M, w \rangle$

Where $M$ is a $TM\left(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject}\right)$ and $w \in \Sigma^*$

1. Take a step count $S$ and initialize $S$ to O.
2. While $S < |Q| \cdot |w| \cdot |\Gamma|^{|w|}$
   (i) Simulate $M$ on $w$
   (ii) If $M$ runs out of $|w|-$bounded tape then reject.
   (iii) accept if $M$ could in this step.
   (iv) increment $S$.
3. Reject."

This machine $T$ runs in polynomial space since the extra counter assumes values at most exponential in the length of the input word.

Thus we constructed a $TM$ $T$ to decide $A_{LBA}$ in polynomial space.

Therefore $A_{LBA} \subset PSPACE$

**Step 4** of 4

(ii) $A_{LBA}$ is $PSPACE-$hard:

Let $L \in PSPACE$

Let $M$ be $TM$ that decides $L$ in space at most $n^k$

clearly $L \leq_P A_{LBA}$ by giving a reduction that maps $w$ to $\left\langle M, wL_1^{|w|^k-1} \right\rangle$.

Thus every $L$ in $PSPACE$ is polynomial time reducible to $A_{LBA}$.

Hence $A_{LBA}$ is $PSPACE$-hard.

From (i) and (ii) $A_{LBA}$ is $PSPACE-$complete

---

**Q:3 A directed graph is strongly connected if every two nodes are connected by a directed path in each direction. Let**
**STRONGLY-CONNECTED = { <G> | G is a strongly connected graph}.**
**Show that STRONGLY-CONNECTED is NL-complete.**

**Answer:**

**Step 1** of 4

Specified that
A directed graph is strongly connected if every two nodes are connected by a directed path in each direction.

Let $STRONGLY\_CONNETED = \{\langle G \rangle \mid G$ is a strongly connected graph$\}$

We have to show that $STRONGLY\_CONNETED$ is $NL\_$ complete

NL-completeness: A language '$B$' is $NL\_$ complete if

1. $B \in NL$, and
2. Every $A$ in $NL$ is log space reducible to $B$.

So, to show that $STRONGLY\_CONNECTED$ is $NL\_$ complete

We need to prove the 2 conditions of $NL\_$ completeness.

**Step 2** of 4

(1) $\overline{STRONGLY\_CONNECTED \in NL}$:

We know that

"$NL$ is the class of languages that are decidable in logarithmic space on non-deterministic Turing machine $(NTM)$"

So to prove $STRONGLY\_CONNECTED \in NL$, we need to construct a $NTM$ $N$ that decides $\overline{STRONGLY\_CONNECTED}$ in logarithmic space.

The construction of $N$ is as follows:

$N_1$ = " On input $\langle G \rangle$:

1. Select two nodes $a$ and $b$ non-deterministically.
2. Run $PATH\ (a,b)$.
   - If it rejects, then the graph is not strongly connected, so accept.
   - Otherwise, reject.

Since storing the node numbers $a$ and $b$ only takes log space, and $PATH$ uses only log space, so $\overline{STRONGLY\_CONNECTED} \in NL$.

We know that $NL = CONL$.

Thereof $STRONGLY\_CONNECTED \in NL$

**Step 3** of 4

(2) Next we must show that every language in $L$ is log space reducible to $STRONGLY\_CONNECTED$.

We do this by reducing $PATH$ to $STRONGLY\_CONNECTED$

The $NTM$ $N_2$ will do this procedure.

$N_2$ = "On input $\langle G,s,t \rangle$, where $G$ is a graph and $s$, $t$ are vertices in $G$

1. Copy all of $G$ onto the output tape
2. Additionally for each node $i$ in $G$.
3. Output on edge from $i$ to $s$.
4. Output an edge from $t$ to $i$. "

**Step 4** of 4

This algorithm only needs log space to store the counter for $i$

- If there is a path from $s$ to $t$, then the constructed graph is strongly connected because every node can now get to every other node by going through the path $s-t$.
- If there is no path from $s$ to $t$, then the graph is not strongly connected. Because the only additional edges in the constructed graph go into $s$ and out of $t$, so there can be no new ways of getting from $s$ to $t$.

So, from (1) and (2)

$STRONGLY\_CONNECTED$ is $NL$ complete

**Q4:Graphs G and H are called isomorphic if the nodes of G may be reordered so that it is identical to H. Let ISO = {<G, H> | G and H are isomorphic graphs}. Show that ISO ∈ NP.**

**Answer:**

Step 1 of 3

**Class *NP* :**

*NP* is a class of languages that are nondeterministic polynomial time on a non – deterministic single – tape Turing Machine.

From the definition 7.19 *NP* is the class of languages that have polynomial time verifies

Given that

$\text{ISO} = \{\langle G, H\rangle \mid G \text{ and } H \text{ are isomorphic graphs}\}$

• If the nodes of *G* may be reordered so that it is identical to *H* then Graphs *G* and *H* are said to be isomorphic.

• Now it has to be proved that $ISO \in NP$

• Let $G = (V_G, E_G)$ and $H = (V_H, E_H)$ be the two graphs

• Let $V_G = \{u_1, u_2, ...u_m\}, V_H = \{v_1, v_2, ...v_n\}$ be the sets of vertices of *G* and *H*.

Step 2 of 3

**Isomorphism:**

An isomorphism is defined by a mapping $f : V_G \to V_H$ with the property that it is a one – to –one correspondence. That means it is both one – to – one and onto.

• This one – to – one correspondence is possible only if $m = n$ and for all $u, v \in V_G$ we have $(v, v) \in E_G$ if and only if $(f(u), f(v)) \in E_H$ .

• Thus, the correspondence takes edges into edges and non – edges into non – edges.

• A mapping *f* can be represented. By a sequence $S = (S_1, S_2, ...S_m)$ of indices with the property that $f(u_i) = v_{s_i}$, that is $i^{\text{th}}$ point of *G* is mapped into the $S_i^{th}$ point of *H*.

• This sequence *S* can be taken as certificate.

Step 3 of 3

Now *N* is the non – deterministic Turing machine $(NTM)$ that decides *ISO* in polynomial time.

$N =$ "On input $(\langle G, H\rangle, S)$:

Where *G* and *H* are graph as defined above *S* is the certificate.

1. Check whether *G* and *H* have same number of points.

2. If *G* and *H* have same number of points then checks that for each pair $i, j$

i. have $S_i \neq S_j$ and that $(u_i, u_j) \in E_U$ if and only if $(v_{S_i}, v_{S_j}) \in E_U$

ii. If so then accept.

3. Otherwise reject".

All these checking can be done in time $O(m^2)$, so in time polynomial in the description of $(G, H)$.

Therefore $ISO \in NP$.

**Q5: The game of Nim is played with a collection of piles of sticks. In one move a player may remove any nonzero number of sticks from a single pile. The players alternately take turns making moves. The player who removes the very last stick loses. Say that we have a game position in Nim with k piles containing s1, ..., sk sticks. Call the position balanced if, when each of the numbers si is written in binary and the binary numbers are written as rows of a matrix aligned at the low order bits, each column of bits contains an even number of 1s. Prove the following two facts.Starting in an unbalanced position, a single move exists that changes the position into a balanced one.**
**Starting in a balanced position, every single move changes the position into an unbalanced one.**

## Answer:

Step 1 of 5

In a game of **NIM** that consists two players, there are $k$ heaps containing $s_1, s_2, \ldots, s_k$ numbers of stacks. Now, the binary equivalent of each of the numbers $(s_i)$ is taken and arrange them row wise and aligned them at the low order bits.

• The arrangement is said to be "balanced" if there are even no of one's in each column. An XOR operation is performed on the stick numbers in each heaps by the column values. **This is called "NIM-sum" of the numbers**.

• Hence, from the definition of balanced position, it can be said that if **zero value is acquired by the NIM-sum, then the arrangement is balanced otherwise not**.

Step 2 of 5

Now to prove the given facts, consider that $s_1, s_2, \ldots, s_k$ be the heaps size before any move is made and $s_1', s_2', \ldots, s_k'$ be the size of heaps after a move is made.

• Assume $NIM_s$ be the NIM-sum before any move (that is, $NIM_s = s_1 \oplus s_2 \oplus \cdots \oplus s_k$) and $NIM_s'$ be the NIM-sum after any move (that is, $NIM_s' = s_1' \oplus s_2' \oplus \cdots \oplus s_k'$).

• If the move is in the heap $l$ then it has $s_i = s_i'$ for all $i \neq l$, and $s_l > s_l'$. It is considered that, the XOR function or more precisely in this case.

• The NIM-sum function $(\oplus)$ follow the simple associative and communicative laws and also follow one more property, that is, $s \oplus s = 0$ .

Therefore, by using these properties, which are discussed above:

$$\begin{aligned} NIM_s' &= 0 \oplus NIM_s' \\ &= NIM_s \oplus NIM_s \oplus NIM_s' \\ &= NIM_s \oplus (s_1 \oplus s_2 \oplus \cdots \oplus s_k) \oplus (s_1' \oplus s_2' \oplus \cdots \oplus s_k') \end{aligned}$$

$$\begin{aligned} &= NIM_s \oplus (s_1 \oplus s_1') \oplus \cdots \oplus (s_k \oplus s_k') \\ &= NIM_s \oplus 0 \oplus 0 \oplus \cdots \oplus (s_l \oplus s_l') \oplus \cdots \oplus 0 \\ &= NIM_s \oplus s_l \oplus s_l' \end{aligned}$$

**Step 3** of 5

Consider an unbalanced position of piles (that is when $NIM_s \neq 0$), Suppose $d$ be the most significant position of non-zero bit in the binary format of $NIM_s$.

• Now, select a $l$ in such a way that a non-zero value is acquired by $d$th bit of $s_l$. The null or zero value acquired by the $d$th bit of $NIM_s$, when no such $l$ exists.

• Now consider, $s_l' = NIM_s \oplus s_l$ and $s_l' < s_l$ because all bits to the left of the $d$th bit is are same in $s_l$ and $s_l'$ and bit $d$ decreases from 1 to 0 or the value decreased by $2^d$.

• So the first player by the winning strategy makes a move by choosing $s_l - s_l'$ objects from heap $l$. Then

$$\begin{aligned} NIM_s' &= NIM_s \oplus s_l \oplus s_l' \\ &= NIM_s \oplus s_l \oplus \left(NIM_s \oplus s_l\right) \\ &= 0 \end{aligned}$$

**Therefore, there exists a single move that changes the position from unbalanced to a balanced one.**

**Step 4** of 5

Now, initialize from a balanced position, that is $NIM_s = 0$, then $NIM_s' = s_l \oplus s_l'$. Suppose, the first move is made on heap $l$ making $s_l \neq s_l'$ then $NIM_s'$ acquires a non-zero value**. Hence, every single move changes the position balanced to an unbalanced one.**

**Step 5** of 5

In a normal play condition, after considering the fact (that is, none of the players makes any mistakes) it is easy to find a winning strategy. It is clear from the above mathematical deduction, that the strategy is to make the NIM-sum zero after each move.

• Consider the winning strategy, if the game is in balanced condition (that is NIM-sum is already zero). User has to move second letting Player II to move first and conversely when the game condition is unbalanced one (that is, the NIM-sum is non-zero). Then player I's has to move first.

• Now considering an algorithm to design a TM for Player I's winning strategy, where each input is already in its binary equivalent. These easy steps can be followed:

Suppose, M = Step

Cross off the 1's on finding of two 1s.

a. If no 1 left uncrossed after the entire scan of input, Player-I has to move second.

b. Else, Player-I has to move first.

**So, the only space tradeoff is to store the binary numbers, which requires $O\left(\log\left(k\right)\right)$ space.**
**Therefore, $NIM \in L$.** □

**Q6 Let HALF-CLIQUE = {<G> | G is an undirected graph having a complete subgraph with at least m/2 nodes, where m is the number of nodes in G}. Show that HALF-CLIQUE is NP-complete.**

Step 1 of 2

Clique is an undirected graph where every two nodes connected by an edge.

***NP*- complete:**

A language $B$ is $NP-$ complete if by an edge it satisfies two conditions.

1. $B$ is in $NP$
2. Every $A$ in $NP$ is polynomial time reducible to $B$.

Step 2 of 2

1. $HALF-CLIQUE \in NP$**:**

Let $N$ be the nondeterministic polynomial time *(NTM)* that decides $HALF-CLIQUE$ in polynomial time.

$N$ can be described as follows:

$N =$ "on input graph $\langle G \rangle$:

1. Non-deterministically choose at least $n/2$ nodes
2. Verify whether $n/2$ nodes form a clique
3. If they form a clique then accept.
4. Otherwise, reject".

**Therefore,** $HALF-CLIQUE \in NP$

2. $CLIQUE \leq_p HALF-CLIQUE$:

A reduction from $CLIQUE$ to $HALF-CLIQUE$ as follows:

On input $\langle G,k \rangle$, where $G$ is a graph on $n$ verifies and $k$ is an integer:

1. If $k = n/2$ then output $\langle G \rangle$.
2. If $k < n/2$, then construct a new graph $G'$ by adding a complete graph with $n-2k$ vertices and connecting them to all vertices in $G$, and output $\langle G' \rangle$.
3. If $k > n/2$, then construct a new graph $G''$ by adding $2k-n$ isolated vertices to $G$, and output $\langle G'' \rangle$.

*When* $k = n/2$ :It is clear that $\langle G, n/2 \rangle \in CLIQUE$ if and only if $\langle G \rangle \in HALF-CLIQUE$.

When $k < n/2$ :If $G$ has a $k$-clique, then $G'$ has a clique of size

$$k+(n-2k)=(2n-2k)/2.$$

Therefore, $\langle G' \rangle \in HALF-CLIQUE$ as $G'$ is a graph with $2n-2k$ vertices.

Conversely,if $\langle G' \rangle \in HALF-CLIQUE$, that is $G'$ has a clique of size $n-k=k+(n-2k)$, then at most $n-2k$ of the clique come from the $n-2k$ new vertices. Therefore the reaming at least $k$ vertices form a clique in $G$.

**Hence,** $\langle G,k \rangle \in CLIQUE$

When $k > \frac{n}{k}$: if $G$ has a $k-$clique, then $G''$ has a clique size $k = \frac{2k}{2}$, and

Therefore, $\langle G' \rangle \in HALF-CLIQUE$ as $G''$ is a graph with $n+2k-n=2k$ vertices.

Conversely, if $\langle G'' \rangle \in HALF-CLIQUE$, that is if $G''$ a clique of size has $k$, then the clique does not contain any of the new vertices as they are isolated.

**Thus, the clique is a $k-$ clique of $G$, and hence $\langle G,k \rangle \in CLIQUE$.**

**Therefore,** the $HALF-CLIQUE$ **is *NP*- complete.**

**Q7 Let SET-SPLITTING = {<S,C> | S is a finite set and C = {C1, ... ,Ck} is a collection of subsets of S, for some k > 0, such that elements of S can be colored red or blue so that no Ci has all its elements colored with the same color.}. Show that SET-SPLITTING is NP-complete.colored with the same color.}**

Step 1 of 3

**NP –Complete:**

A language $B$ is $NP$-complete if it satisfies 2 conditions

1. $B$ is in $NP$
2. Every A in $NP$ is polynomial time reducible to B.

Step 2 of 3

1. **SET – SPLITING is in NP :**

   SET – SPLITING is in NP because we can verify in polynomial time that no subset $C_i$ is monochromatic.

2. $3\,SAT \leq_p SET\ -\ SPLITING$ :

   To prove that the problem is NP complete, we give a polynomial time reduction from 3SAT to SET-SPLITING.

   Given an instance of 3SAT $\phi$ , set $S = \left\{ x_1, \bar{x_1}, \ldots . x_n, \bar{x_n}, y \right\}$, where $x_i s$ are the variables and y is a special color variable.

Step 3 of 3

**The splitting is done as follows:**

For every clause $C_i$ in $\phi$, Let $C_i$ be a subset of S containing the elements corresponding to the literally, in $C_i$ and the special elements $y \in s$, Then $C = C_1, \ldots, C_k$

If $\phi$ is satisfiable, consider a satisfying assignment.

If we color all the true literals red, all the false ones are blue, and y blue, then every subset $C_i$ of S has at least one red element (because it is satisfiable and it also contain one blue element y.

In addition, for a given splitting $\langle S, C \rangle$, we can able to set the literals that are colored differently from y to true.

In the same way, we can able to set the literals that have the same color as y to false.

This concludes that satisfying assignment for $\phi$.

**Thus, SET – SPLITTING is NP-Complete.**

## Q8

**Question:**

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

b. Let PAL-ADD = { $\langle x, y \rangle$ | x, y > 0 are binary integers where x + y 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-ADD ? L.

---

The class $L$: $L$ is the class of languages that are decidable in logarithmic space on a deterministic truing machine.
That is, $L = SPACE(\log n)$

(a) Given Language is
$ADD = \{\langle x, y, z\rangle \mid x, y, z > 0$ are binary integers and $x + y = z\}$

We have to show that $ADD \in L$.
That means, we have to construct a deterministic Turing machine $(DTM)$ that decides $ADD$ in logarithmic space.

Let $M_1$ be the $DTM$ that decides $ADD$ in logarithmic space.
The construction of $M_1$ is as follows:

$M_1$ = "On input $\langle x, y, z\rangle$:

1. If either of the three strings is not a binary number in the sense defined above then reject.
2. Initialize three binary counters $i, j, k$ pointing to the right – most of $x, y$ and respectively.
3. Perform binary addition (bit – wise long addition) using $i, j, k$ and $a$ carry flag.
4. If any discrepancy arises between the calculated next bit of $z$ and the actual next bit of $z$, then reject.
5. If the end of all three number representations is reached and no error detected, then accept. So clearly $M_1$ runs in log space (it uses 3 counter only) and decides $ADD$".

Thus $ADD$ is in $L$. So, $ADD \in L$.

**Step 2** of 2

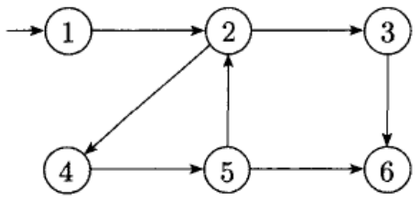
(b) Given language is
$PAL\_ADD = \{\langle x, y\rangle \mid x, y > 0$ are binary integers where $x + y$ is an integer whose binary representation is a palindrome$\}$
We have to show that $PAL\_ADD \in L$.
That means, we have to construct a $DTM$ that decides $PAL\_ADD$ in logarithmic space.
Let $M_2$ be the $DTM$ that decides $PAL\_ADD$ in logarithmic space.
The construction of $M_2$ is as follows.

$M_2$ = "On input $\langle x, y\rangle$:

1. If either of the two strings is not a binary number in the sence defined above, then reject.
2. Compute the length $l$ of $x + y$ in binary.
3. For $i = 1$ to $l/2$.
   (i) Compute the $i^{th}$ bit a of $x + y$
   (ii) Compute the $(l - i + 1)^{st}$ bit $b$ of $x + y$
   (iii) if $a \neq b$ then reject.
4. otherwise, accept"

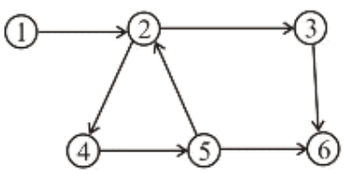
Clearly $M_2$ runs in logspace and decides $PAL\_ADD$.
Thus $PAL\_ADD$ is in L. So, $PAL\_ADD \in L$

**Q9 Consider the following generalized geography game wherein the start node is the one with the arrow pointing in form nowhere. Does Player I have a winning strategy? Does Player II? Give reasons for your answers.**




**Step 1** of 3

Given generalized geography game is




.

**Step 2** of 3

Geography is a game contains nodes connecting. When one player begins at one node, another player continues with connecting the same nodes. If who will get stuck first i.e., there is no connection of node that player will lost the game.

For given problem, say that player I is the one who moves first and player II second.

**Step 3** of 3

**Player I has a wining strategy as follows:**

- Player I starts at node I, the designated start node.
- Node 1 points only at node 2, so player I must select node 2.
- Now Node 2 points at nodes 3 and 4.
- So player II must choose one of these two choices. He chooses node 3.
- Then player I must select 6, which doesn't point to any node.
- So player II is stuck, and thus player I win.

**Player II has a wining strategy as follows:**

- Player II starts at node 1, the designated start node.
- Node 1 points only at node 2, so player I must select node 2.
- Node 2 point at nodes 3 and 4.
- So player II must choose one of these two choices. He chooses node 4.
- Then player I must select node 5.
- Node 5 points at nodes 2 and 6.
- As node 2 already chosen, so player II must select node 6, which doesn't point to any node.
- So player I get stuck and thus player II wins.

**Q10 Let CONNECTED={<G>|G is connected undirected graph}. Show that it is in P**

**Class *P*:** $P$ is a class of languages that are decidable in polynomial time on a deterministic single – tape Turing – machine.

Specified language is,

$\text{CONNECTED} = \{\langle G \rangle \mid G \text{ is a connected graph}\}$

Now we have to show that CONNECTED is in $P$.

Given algorithm for determining the language CONNETED is as follows:
Let $M$ be the Turing – machine that determines language CONNECTED $M$ can be described as follows:

$M =$ " On input $\langle G \rangle$ :

1. Select the first node of graph $G$ and mark it.
2. Reject the following stage until no new nodes are marked.
3. For each node in $G$, mark it if it is attached by an edge to a node that is already marked.
4. Scan all the nodes of $G$ to determine whether they all are marked. If they are accept, otherwise, reject"

- To select the start node and to mark the start node the algorithm will take $O(n)$ time.
- So $O(n)$ time is taken by the algorithm to complete stage
- At mast $n+1$ repetitions are caused at stage 2. Because each repetition except the last marks at least one additional node.
- So the algorithm will take $O(n)$ time to complete stage 2.
- In stage 3, the algorithm has to check at most $n$ nodes of $G$ and for each checked node, the algorithm has to examine all adjacent nodes to see whether any have been marked. That will take at most $O(n^2)$ times
- So in total stage 3 takes $o(n)+o(n^2)=o(n^3)$ time.
- Therefore in total stage 2 and stage 3 take $o(n^3)$ time .
- Therefore in total stage 2 and stage 3 take $o(n^4)$ time as stage 3 and stage 2 are combined.
- In sage 4 the algorithm has to scan all the nodes that will take $o(n)$ time.
- Hence the algorithm runs in $o(n^4)$ polynomial time.
- Therefore CONNECTED is in $P$.

**Q11 A triangle is in an undirected graph is a 3-clique. Show that Triangle in P where Triangle={<G>|G Contains a Triangle}**

Class *P*: *P* is a class of languages that are decidable in polynomial time on a deterministic single – tape Turing – machine.

Specified language,

- A triangle is an undirected graph is a 3 – clique.
- TRIANGLE $=\{\langle G\rangle \mid G \text{ contains a triangle}\}$.
- Now we have to show that TRIANGLE $\in p$
- Let $A$ be the Turing machine that decides TRIANGLE is polynomial time
- $A$ can be described as follows:

  $A$="on input $G\langle V,E\rangle$:

  $V$ denotes set of vertices of the graph $G$.
  $E$ denotes set of edges of the graph $G$.

  1. For $u,v,w\in V$ and $u<v<w$, we enumerate all triples $\langle u,v,w\rangle$.
  2. Check whether all three edges $(u,v),(v,w)$ and $(w,u)$ exist in $E$ or not. If exist then accept.
  3. Otherwise reject."
- Enumeration of all triple require $o\left(|v|^3\right)$time
- Checking whether all three edges belong to $E$ take $O(|E|)$time.
- Overall time is $O\left(|V|^3|E|\right)$ which is polynomial in the length of the inplet
- Therefore TRIANGLE$\in P$

**Q12 Let DOUBLE-SAT={ <φ>| φ has atleast two satisfying assignments }. Show that DOUBLE-SAT is NP-Complete**

Step 1 of 3

***NP* –complete:** A language $B$ is $NP$ – complete if it satisfies two conditions.

1.$B$ is in $NP$ and
2. Every $A$ in $NP$ is polynomial time reducible to $B$.

Step 2 of 3

**1.** $DOUBLE-SAT\in NP$

NTM $N$ can decide double – $SAT$ as follows:

$N$ = " on input a Boolean formula $\phi$:

1. Non-deterministically guess two Boolean assignments $t_1$ and $t_2$ which are different from each other.
2. If both $t_1$ and $t_2$ satisfies $\phi$ then accept
3. Otherwise, reject.

Thus, we construct a NTM $N$ to decide Double – $SAT$.

Hence, $DOUBLE-SAT\in NP$.

Step 3 of 3

2. $SAT\leq_p DOUBLE-SAT$

- The function $f$ which maps an instance $\phi$ of $SAT$ to an instance $\phi'$ of $DOUBLE-SAT$ work as follows:

  $\phi'=\phi\wedge(x_1\vee x_2)$
- Where $x_1$ and $x_2$ are new variables. They do not occur in $\phi$.
- This reduction is certainly polynomial time.
- If $\phi$ is unsatifiable, certainly $\phi'$ is also unsatisfiable. Because we have only conducted an additional term.
- But if $\phi$ has some satisfying assignments $t$, then $\phi'$ has at least three satisfying assignments, corresponding to the 3 different ways of extending $t$ to the new variables $x_1$ and $x_2$.

**Thus, the** $DUBLE-SAT$ **is *NP*- complete.**

**Q13 Let G represents an undirected graph. Also let**
**SPATH={<G,a,b,k>|G contains a simple path of length at most k from a to b}**
**LPATH={<G,a,b,k>|G contains a simple path of length at most k from a to b}**
**Show that SPATH in P**
**Show that LPATH is NP-Complete. You may assume that NP-Completeness of UHAMPATH, the Hamiltonian path problem for undirected graph.**

Step 1 of 3

a)

**Class-*P*:** $P$ is class of languages that are decidable in polynomial time on a deterministic single − tape Turing machine. We have to construct an deterministic Turing machine $(DTM)$ to decide $SPATH$ in polynomial time.

Let $M$ be the $DTM$ to decide $SPATH$ in polynomial time.

The algorithm of $M$ is as follows:

$M =$"on input $\langle G,a,b,k\rangle$ where $m$-node graph $G$ has nodes $a$ and b:

1. Place a mark "o" on node $a$.
2. for each $i$ from 0 to $m$:
3. If an edge $(s,t)$ is found connecting $s$ marked as "$i$" to an unmarked node $t$, mark node $t$ with "$i+1$".
4. If $b$ is marked with a value at most $k$, accept. Otherwise reject.

This algorithm is similar to PATH algorithm. Here we additionally need to keep the track of length of the shortest paths discovered. That will be done in polynomial time $O(|V|+|E|)$.

Hence, we constructed a $DTM$ $M$ to decide $SPATH$ in polynomial time.
Therefore, $SPATH+1 \in P$.

Step 2 of 3

(b)

***NP*- complete:** A language $B$ is $NP$ − complete if it satisfies two conditions.

1. $B$ is in $NP$ and
2. Every $A$ in $NP$ is polynomial time reducible to $B$.

To show $LPATH$ is $NP$ − complete, we need show

$LPATH \in NP$ and $UHAMPATH \leq_p LPATH$

**1.** $LPATH \in NP$**:**

We know that "$NP$ is the class of languages that have polynomial time verifies.
We construct a verifies $V$ for $LPATH$ as follows:

$V =$"on input $\langle G,a,b,k,c\rangle$, where $c$ is a path:

1. Check $c$ is a non − repeated sequence of nodes in $G$.
2. Check the first term of $c$ is a and last is $b$.
3. Check the length of $c$ is larger than or equal to $k$.
4. If $c$ satisfies the conditions 1 to 3, accept.
5. Otherwise, reject

This verifier $V$ can finish in $O(ICI)$ where $|c|$ is the length of $c$.

**So,** $LPATH \in NP$**.**

Step 3 of 3

2. $UHAMPATH \leq_p LPATH$ :

Consider an instant $\langle G,a,b\rangle$ of $UHAMPATH$ problem where $G=\langle V,E\rangle$ is a graph with assigned starting node $a$ and ending node $b$.

- The mapping copy $\langle G,a,b\rangle$ and set $k=|V|-1$, then $\langle G,a,b,k\rangle$ is an instance of $LPATH$.
- It can be finished in polynomial time $\left(O\left(|V|+|E|\right)\right)$
- We need to prove $\langle G,a,b\rangle \in UHAMPATH \Leftrightarrow \langle G,a,b,k\rangle \in LPATH$

If $\langle G,a,b\rangle \in UHAMPATH$, then $G$ has a Hamiltonian path from $a$ to $b$.

- It must be a simple path that goes through every node exactly once,
- Which implies that the length is $|v|-1=k$
- So $\langle G,a,b,k\rangle \in LPATH$

If $\langle G,a,b,k\rangle \in LPATH$, there exists a simple path from $a$ and $b$ with length $k=|v|-1$.

- Because the graph $G$ only has $k+1$ nodes.
- So this simple path must pass through all of needs in graph $G$ exactly once.
- So this simple path must be a Hamiltonian path.
- It implies that $\langle G,a,b\rangle \in UHAMPATH$

**Therefore, the $LPATH$ is $NP$ – complete.**

**Q14 Let CNFk={< φ >| φ is satisfiable cnf formula where each variable appears in at most k places}.**

a. **Show that CNF2 in P**
b. **Show that CNF3 is NP-Complete.**

Step 1 of 3

$$CNF_K = \left\{ \langle \phi \rangle \mid \phi \text{ is a satisfiable } cnf - \text{formula where each variable appears in at most } k \text{ places} \right\}$$

**CNF is Conjunctive normal form, it contains few rules.**

- A literal is Boolean variable or negated Boolean variable in the form
- Clause contains several literals connected with $\vee$s and $\wedge$s.

Step 2 of 3

(a) We have to show that $CNF_2 \in P$.

Class-$P$: $P$ is a class of languages that are decidable in polynomial time on a deterministic single – tape Turing – machine.

Let $T_p$ be the polynomial time decider for $CNF_2$.

$T_p$ can be described as follows:

$T_p =$ "on input $\langle \phi \rangle$":

According to CNF rules, we will choose the clauses:

1. CNF is the form $c$ in first clause of $\phi$, if at all any negation of $c$ then we will reject.
2. CNF is the form $c \vee A$, where $A$ is CNF. Search with $\phi$, if at all any $c$ or negation of $c$ in more than two places then we will reject.
3. Solve CNF where $c$ occurs in every clause, where negation of $c$ is not appears.
4. When searching with $\phi$, if clauses found in the form $c \vee A$ and $\neg c \vee B$ then remove. Add $A \vee B$ in $\phi$
5. Go to step 1.

Every time $T_p$ processes each variable and reaches either accept or reject. Because of this the number of clauses in $\phi$ might decrease by 1 or 2. Hence running time of $T_p$ becomes polynomial time in terms of the number of variables. So, $\boxed{CNF_2 \in P}$.

Step 3 of 3

(b) We have to show that $CNF_3$ is $NP$- complete.

$NP-$complete : A language $B$ is $NP-$ complete if is satisfies two conditions:

1. $B$ is in $NP$
2. Every $A$ in $NP$ is polynomial time reducible to $B$.

1. $CNF_3 \in NP$ : If $CNF_3$ is in $NP$
   - $V_p$ is a verified Polynomial time and described as follows:
     $V_p =$ "on input $\langle \langle \phi \rangle, c \rangle$"

According to CNF rules, we will verify the clauses:

- Verify each variable in $\phi$ occurs in at most 3 places.
- Verify whether $c$ is a satisfying assignment in $\phi$.
- If both allow then accept.
- Otherwise, reject.

2. $3SAT \leq_P CNF_3$ : It is best example for $CNF_3$ satisfying assignment.

- Let $r_p$ be the polynomial time reduction from $3SAT$ to $CNF_3$.
- When given an input instance $\phi$ of $3SAT, r_p(\langle \phi \rangle)$ constructs an instance of $CNF_3$ from following:
  - First read from left to right, select the best example variable that access more than three times in the formula. Example variable as $S$ occurs in $m$ multiple places. $(c_1 \vee A_1), \ldots (c_m \vee A_m)$ where $c_i$ is $S$ or negation $S$.
  - If nothing results as more than three times then output is $\phi$
  - Select variables $S_1, \ldots, S_m$, If any $(c_i \vee A_i)$ remove from the formula $(S_1 \vee A_1) \wedge (\neg S_1 \vee S_2) \wedge (S_2 \vee A_2) \wedge (\neg S_2 \vee S_3) \ldots (S_m \vee A_m) \wedge (\neg S_m \vee S_m)$
  - Go to step 1

Obviously reduced polynomial time $r_p(\langle \phi \rangle)$ is a formula identified that every variable occurs at most three times. It is also clear that $\phi$ is satisfiable if and only if $r_p(\langle \phi \rangle)$ is satisfiable.

$r_p$ is a reduced polynomial time in terms of the number of variable in $\phi$ from (1) and (2) $\boxed{CNF_3 \text{ is } NP\text{-complete}}$.

**Q15**

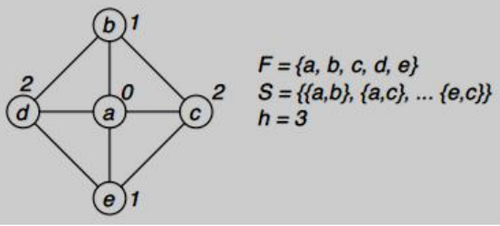
Consider the following scheduling problem. You are given a list of final exams $F_1, \ldots, F_k$ to be scheduled, and a list of students $S_1, \ldots, S_l$. Each student is taking some specified subset of these exams. You must schedule these exams into slots so that no student is required to take two 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.



**Q16 A subset of nodes of a graph G is dominating set if every other node of G is adjacent to some node in the subset.Let DOMINATING-SET={<G,k>| G has a dominating set with k nodes}.**
**Show that it is NP Complete by giving a reduction from VERTEX-COVER.**

Step 1 of 2

**NP-Complete:**

A language B is NP-Complete if satisfies 2 conditions.

1. B is in NP
2. Every A is NP is polynomial time reducible to B.

**1. DOMINATING – SET is in NP:**

- Given an instance $\langle G,k\rangle$ of the DOMINATING-SET and a covering D; check that each node of G is adjacent to some node in D.
- This is done in polynomial time.
- Therefore DOMINATING-SET is in NP.

Step 2 of 2

2. VERTEX-COVER $\leq_p$ DOMINATING -SET

- Now we show that VERTEX-COVER reduces to DOMINATING-SET.
- Given an instance $\langle (V,E),k\rangle$ of VERTEX-COVER., construct instance $\langle ((V-S)UV^1, EUE^1),k\rangle$ of DOMINATING-SET where $S \subseteq V$ are nodes of degree O.
- For each edge $(u,v) \in E$ there are edges $(u,w)$ and $(w,v)$ in $E^1$, where $w \in v^1$ in new vertex corresponding to $(u,v)$.
- Let $G=\langle V,E\rangle$ and $G^1=\langle (V-S)UV^1, EUE^1\rangle$

$(\Rightarrow)$ Suppose $(\langle V,E\rangle,k)$ is in VERTEX-COVER.

- There exits $C \subseteq V$ of size k where each edge $(u,v)\in E$ has either $u \in c$ or $u \in c$.
- Using the above construction, we claim that graph namely c.

- If $u \in (V-S)$, the degree of $v$ is one or more.
- Then, there exists node $u$ such that $(u,v)\in E$, which implies that at least one of $u$ *or* $v$ is in C, so is $v$ is covered.

- If $w \in V^1$, then $w$ is adjacent to both $u$ and $v$ where $(u,v)\in E$, which implies that at

least one of $u$ or $v$ is in $c$, so $w$ is covered.

In other direction, suppose that $\langle((V-S)UV^1, EUE^1), k\rangle$ is in DOMINATING-SET.

- Then there exists $C \subseteq ((V-S)UV^1)$ of size $k$.
- In cases where multiple such $C$ exist, note that at least one includes no vertices in $V^1$.
- This always exists, since $w \in (C \cap V^1)$ that corresponds to edge $(u,v)$ covers only nodes $u, v, w$, but using $u$ instead of $w$ covers $u, v, w$, and possibly more.
- Therefore, $C \subseteq (V-S)$, and $C$ is a vertex cover for $G$.
- This is because $C$ is a DOMINATING-SET for $G^{11}$, implying that all nodes of $V^1$ are covered, so every edge $(u,v) \in E$ has at least one of $u, v$ in $c$.

Therefore, VERTEX-COVER $\leq_p$ DOMINATING-SET.

Thus, from (1) and (2)

**DOMINATING-SET is NP-Complete.**

**Q:17 Show that for any function f:N→R+, where f(n)≥n, the space complexity class SPACE(f(n)) is the same whether you define the class by using the single tape TM Model or the two tape read only input model.**

**Step 1** of 3

SPACE$(f(n))$: Let $f: N \to R^+$ be a function. The space completely class SPACE$(f(n))$ is defined as follows:

$$\text{SPACE}(f(n)) = \left\{ \begin{array}{l} L \mid L \text{ is a language decided by an } O(f(n)) \text{ space} \\ \text{deterministic Turning machine} \end{array} \right\}$$

Now we have to prove that, class SPACE$(f(n))$ is same whether we define $z$ by using the single tape $TM$ or two tape read-only input $TM$ model.

**Step 2** of 3

To prove this, we have to do the following two things

(i) We have to simulate single – tape $TM$ on two tape read – only $TM$ and

(ii) We have to simulate two tape read only $TM$ on single – tape $TM$.

(i) **Simulation of single – tape *TM* on two tape read – only *TM:***

1. First we have to scan the read only tape.
2. Contents of the read only tape are copied to the work tape.
3. Remainder of the contents is simulated by treating the read | write work – tape as the input tape of a single – tape $TM$.

- Clearly only $\log n$ space is used to copy the contents.
- Since $f(n) \geq n$, we can write all of the input on are work – tape

In this way we simulated single-tape $TM$ on two tape read only $TM$.

**Step 3** of 3

(ii) **Simulation two tape read only *TM* on single – tape *TM:***

- There is no participation from single tape TM; we are working over the first input symbols, using remainder of the tape as our work area.
- For n symbols, add $n$ amount of space. By increasing space we add constant to $n$.

So from (i) and (ii) simulations we proved that class $SPACE\left(f\left(n\right)\right)$ is same whether we defined it by using the single tape *TM* or two – tape read _ only input *TM*.

---

## Q:18

**Question:**

Read the definition of MIN-FORMULA in Problem 7.46.

**a.** Show that $MIN\text{-}FORMULA \in$ PSPACE.

**b.** Explain why this argument fails to show that $MIN\text{-}FORMULA \in$ coNP: If $\phi \notin MIN\text{-}FORMULA$, then $\phi$ has a smaller equivalent formula. An NTM can verify that $\phi \in \overline{MIN\text{-}FORMULA}$ by guessing that formula.

Problem 7.46

Say that two Boolean formulas are equivalent if they have the same set of variables and are true on the same set of assignments to those variables (i.e., they describe the same Boolean function). A Boolean formula is minimal if no shorter Boolean formula is equivalent to it. Let MIN-FORMULA be the collection of minimal Boolean formulas. Show that if P = NP, then MIN-FORMULA ? P.

**Step 1** of 1

a. Consider following algorithm

On Input $F$:

1. For each string $s$ such that $|s| < |F|$, if $s$ is a valid representation of a formula (this can be easily checked) which is equivalent to $F$ (this can be checked in polynomial space by evaluating both $F$ and $s$ over all possible truth assignments and comparing the results) then reject.

2. If all string has been tried without rejecting, accept.

Correctness of the algorithm should be evident. Only space used by the algorithm is for storing formula $F$, string $s$ and current assignment of literals which amounts to polynomial space. Hence, $MIN-FORMULA \in PSPACE$.

b. It fails to show $MIN-FORMULA \in coNP$ because it is not known if one can verify equivalence of two Boolean formulae is polynomial time.

## Q:19

**Question:**

Let A be the language of properly nested parentheses. For example, (()) and (()(()))() are in A, but )( is not. Show that A is in L.

**Step 1** of 3

The class L: L is the class of languages that are decidable in logarithmic space on a deterministic truing machine.

That is $L = SPACE(\log n)$

Given that

'$A$' be the language of property nested parenthesis.

For example, $(( \ ))$ and $(((\ )))$ etc are in $A$. But not ) (.

**Step 2** of 3

We have to show that $A$ is in $L$.

That means, we have to construct deterministic Turing machine $(DTM)$ that decides $A$ in logarithmic space.

Let $M$ be the $DTM$ that decides $A$ in logarithmic space.

The construction of $M$ is as follows:

$M$ = "On input $w$:

Where $w$ is a sequence of parenthesis.

1. Starting at the first character of $w$, move right across $w$.
2. when left parenthesis '(' is encountered, add 1 to the work – tape and move right.
3. when right parenthesis ')' is encounter and the work tape is blank, then reject.
Otherwise subtract 1 from the work – tape and move right.
4. When the end is reached, accept if the work tape is blank, reject if the work tape in not blank."

**Step 3** of 3

- Clearly, the only space used by this algorithm is for the counter on the work tape.
- If this counter is in binary, then the most space used by the algorithm is $O(\ln k)$
  Where $k$ is the number of $(s$.
- Since the number of $(s$ is less than or equal to $n$ (the size of tape), this places the language $A$ in $L$.
  Thus we proved that $A \in L$