Pg 163 CLR

8.3-3 let 0 be in left ½ of array
1 " " right " " "

Then for a sequence of lg n fair coin flips we have our random #.

=

```
Random(a,b)
    if a == b return a.
    if coinflip == 0
        Random(a, ⌊(b+a)/2⌋)
    else
        Random(⌊(b+a)/2⌋+1, b)
```

8.3-4

```
for i = 1, to n do
    destination = Random(1,n)
    Swap A[i] + A[destination]
```

Pg 174 CLR

$$n\lg n - n\lg e = \Omega(n\lg n)$$

? $\exists$ a c $\ni$

$$n\lg n - n\lg e \geq c\, n\lg n \quad ?$$

$$-n\lg e \geq (c-1)\, n\lg n$$

$$-\lg e \geq (c-1)\lg n$$

$$\lg e \leq (1-c)\lg n \qquad \text{yes } \forall c < 1$$

pg 641 CLR

28.2-2

$\langle n, n-1, \ldots, 1 \rangle$ ~~All~~ sorts

$\langle 1, 0, 0, \ldots, 0 \rangle$

$\Rightarrow$

define $f_1 : \langle \rangle \to \langle \rangle$

$f_1(n) = 1$, $f_1(n-1) = f_1(n-2) = \cdots = f_1(1) = 0$ Mono $\nearrow$

$f_2(n) = 1$, $f_2(n-1) = 1$, $f_2(n-2) = 0, \ldots$

$\Leftarrow$?

$i =$ smallest #.

31.1-7

pg 738 CLR

$$A^{-1}A = I$$

trans

$$A^T (A^{-1})^T = I$$

$$A(A^{-1})^T = I \quad + \text{ other ordering} \Rightarrow A^{-1} = (A^{-1})^T.$$

i.e.

$$I = (A^{-1})^T A$$

31.2-6

pg 745 CLR

$$(a+bi)(c+di) = (ac-bd)+i(ad+bc)$$

$$t_1 = ad$$
$$t_2 = bc$$
$$t_3 = (a+b)(c-d) = ac - \underline{ad} + \underline{bc} - bd$$

Then $ad+bc = t_1+t_2$

$ac-bd = t_3+t_1-t_2$

P 923 CLR

36.1-1 longest path length can be solved in poly time iff longest path ~~[illegible]~~. $\in P$.

$\Leftarrow$

longest path $\in P$ $\rightarrow$ $\exists$ an algorithm A that <u>decides</u> L in polynomial time $\Rightarrow$ given an length n (any one) string in $x \in \{0,1\}^*$ x can be accepted or rejected in time $O(n^k)$. Thus given a specific instance of longest path length we can ~~[illegible]~~ compute say the sum of the # of edges $\sum E$. Then deciding longest-path $= \{\langle G,u,v,k\rangle : \ldots\}$ w/ $k = \frac{\sum E}{2}$ in polynomial time will tell one if one can solve this problem w/ $k = \frac{\sum E}{2}$. Bisecting each time, one can solve $\lfloor \lg_2 \sum E \rfloor$ longest path problems, to obtain the solution to longest path length. This would require a time of $O(\lfloor \lg \sum E \rfloor n^k) = O(n^{k+1})$ solves. Thus longest path length can be solved in poly. time.

$\Rightarrow$

If longest path length can be solved in poly time. Then given a n length string $\in \{0,1\}^*$ solve the longest path <u>length</u> problem. If the longest path is $\geq k$ in the longest path problem instance then $A(x) = 1$ & x has been decided in polynomial time, since x was arbitrary the language can be decided in polynomial time. $\therefore$ longest path $\in P$.

36.1-2 (i) Given a undirected graph $G = \cancel{(E, V)}\ (V, E)$. ~~vertex~~ ~~V~~.
find the longest ~~directed cycle~~ simple cycle in that graph.

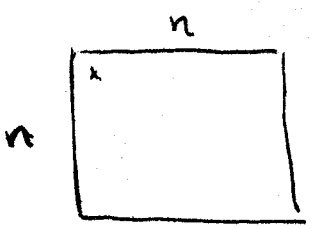
(ii) Given a undirected graph $G = (V, E)$ & an integer k. Is there a simple cycle of length $\geq k$?

(iii) $L = \{ \langle G, k \rangle : G = (V, E)$ is an undirected graph
$k \geq 0$ an integer & $\exists$ a simple cycle of
length $\geq k$ $\}$

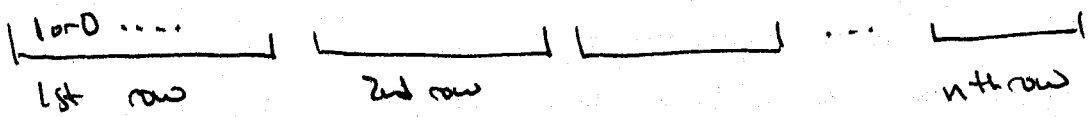
36.1-3 Adjacency-matrix representation: <u>directed graph</u>

Given a directed graph w/ n nodes, a binary string of length n^2

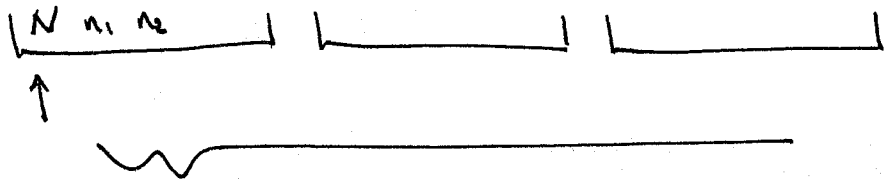
(max # of edges $= n^2$)



would have in a list



Now An Adjacency list representation would be a binary string of length



n blocks

each block has as 1st # the # of elements to follow in binary * & then binary #'s representing the edges attached.

* Note: since the # of vertices is n, the binary #'s should have $\lg n$ binary digits.

The two methods are obviously polynomial related since 1) They are simply related & 2) Reading & writing each takes polynomial time.

36.1-4 To be a polynomial time algorithm the language associated w/ its decision problem must be decided in polynomial time.

Thus from problem 17.2-2 given an instance of the 0-1 knapsack problem v_i values $1 \le i \le n$, w_i weights $1 \le i \le n$ wants to

$$\max \sum_{i=1}^{N\{v_i, w_i\}} v_i$$

can be solved in $O(nW)$

W max weight of items in knapsack. I say no because n & W are not mess related by polynomial fns

$$W \le \sum_{i=1}^{n} w_i \le n w_{max}.$$

Thus the solution can be solved in $O(w_{max} n^2)$ which is polynomial so I now say yes. This is a polynomial time algorithm.

36.1-5 L can accept a string of length n in polynomial time but the algorithm that does this runs in super-poly time when $x \in L$.

Can L be decided in poly time?

My guess would be to stop the algorithm if it is still running after a poly ~~[illegible]~~ length of time & call this instance a NO?

36.1-6 $T(n) = O(n^k)$ $T^*(n) = CT(n)$

$\Rightarrow T^*(n) = C \cdot O(n^k) = O(n^k)$

But if $T(n) = O(n^k)$

$T^*(n) = O(T(n)^p) = O(n^{kp})$ is this not polynomial?

36.1-7 $P = \{L : L$ is accepted by a poly-time algorithm$\}$.

$L_1 \in P$ $L_2 \in P$ $L_1 \cup L_2 \overset{?}{\in} P$

let ~~[illegible]~~ $x \in L_1 \cup L_2$ if $x \in L_1$ ~~[illegible]~~ can be decided by a poly algo.

Not totally sure how to argue.

36.2-1 Graph isomorphism =

$G_1 = (V_1, E_1)$ & $G_2 = (V_2, E_2)$ ∋ ∃ bijection $f: V_1 \to V_2$ ∋

$(u,v) \in E_1$ iff $(f(u), f(v)) \in E_2$

Given a bijection from $V_1 \to V_2$ then ∀ edge in E_1 of which there are $|E_1|$ of these $= O(n^2)$ w/ $n = |V|$ check that $(f(u), f(v)) \in E_2$. This is an $O(n^2)$ or $O(n)$ algorithm depending on counting (with edges or vertices). This would verify the language in poly time

36.2-2 A bipartite graph = is an undirected graph $G = (V, E)$ in which V can be partitioned into two sets V_1 & V_2 ∋ $(u,v) \in E$ ⇒ either $u \in V_1$ & $v \in V_2$ or $u \in V_2$ & $v \in V_1$.

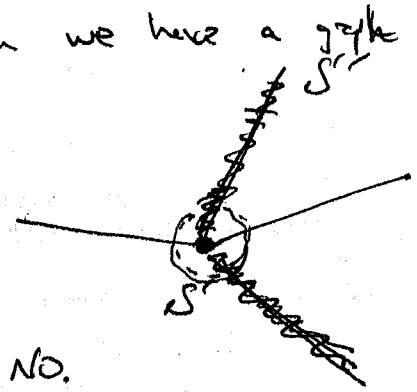
If our bipartite graph has an odd # of vertices either V_1 or V_2 must have an odd #. ~~Assuming A Hamiltonian cycle would not be possible~~ ~~since moving between~~ W.O.L.O.G let V_1 be the odd # vertices ⇒ $|V_1| = |V_2| \pm 1$ so that ~~[illegible]~~ There exists ∃ ~~[illegible]~~ #~~[illegible]~~ ~~[illegible]~~

$$|V_1| + |V_2| = 2|V_2| \pm 1$$

This is not an even # of pairs & this cannot be broken into any # of edges.

36.2-3

Then Given a graph we could remove vertices one at a time until a HC was found. For instance on the 1st call remove every vertice (call this the starting point) Then we have a graph w/



1st Remove edges around a given node one at a time until the answer to HAM-CYCLE is NO.

Pick a node as the starting point S. Remove edges around this node one at a time until HAM-CYCLE is NO. Then we know this edge must be on the hamiltonian cycle connecting vertex S to S''. Move to vertex S' & continue in the same manner After replacing all the removed edges from S'. when HAM-CYCLE becomes NO this edge is another edge in the HAM-CYCLE. moving us to another vertex. Continuing in this way we walk around the graph obtaining a list of vertices of the hamiltonian cycle

36.2-4

NP languages $L = \{x \in \{0,1\}^* : \exists$ a certificate y w/ $|y| = O(|x|^c)$ $\ni A(x,y) = 1\}$.

let L_1 & L_2 be two NP languages $\ni \exists A_1$ & A_2 polynomial time algo $\ni$

$$L_1 = \{x \in \{0,1\}^* : \exists \ldots\}$$
$$L_2 = \{x \in \{0,1\}^* : \exists \ldots\}$$

Pr:

$L_1 \cup L_2$ is a NP language

let $x \in L_1 \cup L_2$ then either $x \in L_1$ or $x \in L_2$ use the corresponding poly algorithm on x?

36.2-5 Given a problem instance x of size n one can consider the exhaustive set of certificates which would be

$$O(n!) \leq O(2^n)$$

Since $n! = \omega(2^n)$

$$\sqrt{2\pi n}\left(\frac{n}{e}\right)^n \leq n! \leq \sqrt{\pi n}\left(\frac{n}{e}\right)^{n+\frac{1}{12n}}$$

Each of these instances would require $O(n^k)$ amount of time to verify & Thus the total verification time would be $O(n^k \cdot 2^n)$

36.2-7

H. Path can be solved in poly. tim on directed acyclic

acyclic would mean that no ~~poly~~ cycles exist & thus

the answer would always be <u>no</u>. I'll assume this is a typo

& try to solve the problem w/ directed cyclic grphs.

36.2-8

36.2-8 TAUTOLOGY = $\{L^T : L^T = \{\text{A tautology}\}\}$

co-NP =

If I complement this language I get the set of Boolean functs that return 1 for some inputs + 0 for some inputs. Since this language is in NP (this is shown in a latter section), in fact it is circuit satisfiability. + is NP

$\Rightarrow$ TAUTOLOGY is in co-NP.

36.2-9 Pr $P \subseteq$ co-NP

$P = \{L : L$ is accepted by a polynomial-time Algorith$\}$

$\overline{P} = \{L : L$ <u>cannot</u> be accepted by a polynomial-time Algorithm$\}$

How can I show that a given language in $\overline{P}$ can be "checked" i.e. that there is a certificate for the problem instance?

36.2-10 Pr if $NP \neq$ co-NP
then $P \neq NP$

~~NP = co-NP~~ Assume $P = NP$ then co-P = co-NP
Since co-P = P (P is closed under complement)

$\neg$ $P = co\text{-}NP$? Where is the contradiction?

36.2-11 G^3 = Graph obtained from G by connecting all vertices that are connected by at least

36.3-1 Pr:

$L_1 \leq_p L_2$ & $L_2 \leq_p L_3$ then $L_1 \leq_p L_3$

Now $L_1 \leq_p L_2 \Rightarrow \exists$ a polynomial time computable function

$f_{12} : \{0,1\}^* \to \{0,1\}^*$ $\ni$ $\forall$ $x \in \{0,1\}^*$

$x \in L_1$ iff $f_{12}(x) \in L_2$

Since $L_2 \leq_p L_3$ $\exists$ a polynomial time computable fn f_{23} $\ni$

$f_{23} : \{0,1\}^* \to \{0,1\}^*$ $\ni$ $\forall$ $x \in \{0,1\}^*$

$x \in L_2$ iff $f_{23}(x) \in L_3$

Then consider $f_{13} = f_{23}(f_{12}(x))$ Then f_{13} is poly time computable since polynomials are closed under composition.

If $x \in L_1$ then $f_{12}(x) \in L_2$ & $\therefore$ $f_{23}(f_{12}(x)) \in L_3$ ✓

If $f_{23}(f_{12}(x)) \in L_3 \Rightarrow f_{12}(x) \in L_2 \Rightarrow x \in L_1$

$\therefore$ $L_1 \leq_p L_3$

36.3-2 Pr: $L \leq_p \bar{L}$ iff $\bar{L} \leq_p L$

Assume $L \leq_p \bar{L}$

$\Rightarrow \exists$ polynomial-time ~~reduction~~ ~~from~~ ~~E~~ computable fn

$f: \{0,1\}^* \to \{0,1\}^*$ $\ni$ $\forall x \in \{0,1\}^*$

$x \in L$ iff $f(x) \in \bar{L}$

The complementing gives $\overline{f(x)} \in L$ iff $\bar{x} \in \bar{L}$

$\bar{x} \in \bar{L}$ iff $\overline{f(x)} \in L$

let ~~x~~ $v = \bar{x}$ $\Rightarrow$ $v \in \bar{L}$ iff $\overline{f(\bar{v})} \in L$

Now ~~R~~ $g(v) = \overline{f(\bar{v})}$ is polynomial time computable

& thus $\bar{L} \leq_p L$. since these two expressions are symmetric

the other direction is like wise proven

36.3-3 Lemma 36.5 : The circuit-satisfiability problem belongs to the class of NP. ⇒

CIR-SAT ∈ NP ⟺ ∃ two-input polynomial time algo. A

I don't understand ~~why~~ what the difference is between the question that is asked & the method of proof taken in the text.

36.3-4 I don't understand this problem.

36.3-5 ϕ cannot be polynomial reducible to P

Since there are no instances of ϕ

to map into instances of P.

$\{0,1\}^* \overset{?}{\leq}_p P$ cannot be complete since

∞ long instances (a member of $\{0,1\}^*$) are yes instances

& No polynomial ~~Pa~~ computable fn could even write/print these instances let alone reduce them to a yes instance in P.

PS 939 CLR 04-16-02

36.3-6 Pr: L is complete for NP iff $\bar{L}$ is complete for co-NP

Assume L is complete for NP

$\Rightarrow$ L is in NP & $L' \leq_p L$ $\forall$ $L' \in$ NP

$\Leftrightarrow$ $\bar{L}$ is in co-NP & $\bar{L'} \leq_p \bar{L}$ $\forall$ $L' \in$ NP

$\Leftrightarrow$ $\bar{L}$ is ~~in co-NP~~ ~~& $\bar{L'} \leq_p \bar{L}$~~

let $\tilde{L} = \bar{L}$ & $\tilde{L'} = \bar{L'}$

$\Rightarrow$ $\tilde{L}$ is in co-NP & $\tilde{L'} \leq_p \tilde{L}$ $\forall$ $\tilde{L'} \in$ co-NP.

$\Rightarrow$ $\bar{L}$ is complete for co-NP.

36.3-7 To show something is NP-hard means that

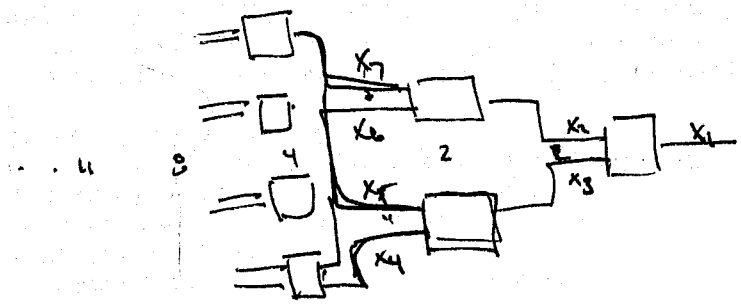
$L' \leq_p L$ $\forall$ $L' \in C_1$ w/ C_1 a language class.

Now this intuitively means that L is "harder" than all other languages L' in C_1.

$L' \leq_p L \Rightarrow \exists$ a polynomial time computable fn taking ~~YES/NO~~ ~~instances~~ ~~in~~ ~~f~~ from $\{0,1\}^*$ yes instance in L_1 into yes instance of L_2. We don't have to specify what this fn is just that an exists. We are not constructing its value just stating its existence

pg 745 CLR

36.4-1



SAT = satisfiability (formula).

$L' \leq_p L$ for some $L' \in NPC$, L is NP-hard

n = # of input arguments

CIRCUIT-SAT $\leq_p$ SAT

$$\phi = x_1 = x_2 \bigcirc x_3 = (x_7 \bigcirc x_6) \bigcirc (x_4 \bigcirc x_5)$$

The Idea is to make the length of the formula 2^n, but I don't understand the following:

1) n for CIRCUIT-SAT must be the # of input x's.

How do I determine the # of circuit elements?

n as # of inputs. This would then means that an input of size n would have $\lg n$ tiers. which would not be exponential.

36.4-2 Formula 36.3

$$\phi = ((x_1 \to x_2) \vee \neg((\neg x_1 \leftrightarrow x_3) \vee x_4)) \wedge \neg x_2$$

$\cdots$

$\phi' =$

$$\phi' = y_1 \wedge (y_1 \leftrightarrow (y_2 \wedge \neg x_2))$$
$$\wedge (y_2 \leftrightarrow (y_3 \vee y_4))$$
$$\wedge (y_3 \leftrightarrow (x_1 \to x_2))$$

$\vdots$

After applying a binary parsing tree.

Then for each clause we convert it into CNF.

Since the 1st one $y_1 \leftrightarrow (y_2 \wedge \neg x_2)$ has already been done in the book I'll do the second

$$y_2 \leftrightarrow (y_3 \vee y_4)$$

y_2	y_3	y_4	$y_2 \leftrightarrow (y_3 \vee y_4)$
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0

y_2	y_3	y_4	$y_2 \leftrightarrow (y_3 \vee y_4)$
1	0	1	1
1	1	0	1
1	1	1	1

Thus $\neg(y_2 \leftrightarrow (y_3 \vee y_4)) = (\neg y_2 \wedge \neg y_3 \wedge y_4) \vee$

$(\neg y_2 \wedge y_3 \wedge y_4) \vee$

$(\neg y_2 \wedge y_3 \wedge y_4) \vee$

$(y_2 \wedge \neg y_3 \wedge \neg y_4)$

Thus

$$(y_2 \leftrightarrow (y_3 \vee y_4)) = (y_2 \vee y_3 \vee \neg y_4) \wedge$$
$$(y_2 \vee \neg y_3 \vee \neg y_4) \wedge$$
$$(y_2 \vee \neg y_3 \vee \neg y_4) \wedge$$
$$(\neg y_2 \vee y_3 \vee y_4)$$

This is in 3-CNF.

36.4-3 Just forming the truth table takes 2^n amount of time.

Because he ~~test~~ has to list every boolean choice, before he can find a ~~DN~~ DNF. This is not a polynomial reduction.

36.4-4 Ex 36.3-6 is

L is complete for NP iff $\bar{L}$ is complete for co-NP

~~since let L = language about which a boolean formula is either all true or all false. Then we know L ∈ NP~~

let $L = \{$ satisfying assignments of a ~~circuit~~ boolean formula $\}$, then

we know $L \in NPC \Rightarrow \forall\ L' \in NP \quad L' \leq_p L$.

Thus L is complete for the class NP, by the definition of complete.

By Ex 36.3-6 $\bar{L}$ is complete for co-NP

since $\bar{L} = \{$ set of <u>non</u> satisfying assignments of a boolean formula $\}$.

... E ~~doesn't seem to be the~~ s

I don't see how to relate this to a class of tautologies?

Pg 946 CLR 04-21-021

36.4-5 If ~~the~~ a boolean formula is given in disjunctive Normal form (or of Ands)

$$(\cdots \vee \cdots \vee \cdots) \wedge (\cdots \vee \cdots \vee \cdots) \wedge (\cdots \vee \cdots \vee \cdots) \cdots$$

to make satisfiable one simply goes to each clause (a group of ors) & picks a variable (litteral) not already picked & set it equal to 1. (This will make the entire clause one) Then n (at most) applications of this rule will yield an assignment that will satisfy ϕ.

~~to make 1st clause true require one of the following to be true:~~

~~x_1, $\neg x_2$, x_4, ...~~

~~2nd " " " " " " " " " " " " " :~~

What about deciding the language. It might be harder to decide all of the assignments that give one a one as the output of the formula. Does this matter?

36.4-6 ~~For~~ ~~Take~~ Take a sub formula the formula in question w/ values hard wired into its variables. Say expression of interest is $\phi(x_1, x_2, \ldots, x_{10})$. Then ask our subroutine that decides formula satisfiability does the formula $\phi(0, x_2, \ldots, x_{10})$ have a satisfying assignment? (It ask the question for $\phi(x_1, \ldots, x_{10})$ or else stop) If the answer is ~~no~~ no. Then since you asked about $\phi(x_1, \ldots, x_{10})$, $\phi(1, x_2, \ldots, x_{10})$ must ~~be~~ ~~a~~ ~~sat~~ have a satisfying assignment. Thus at most ~~n~~ ~~n+1~~ n+1 calls to our satisfiability routine would be required to determine a satisfying assignment

Pg 946 CLR 04-21-02

36.4-7

2-CNF-SAT CNF: "AND of ORs"

Given ϕ in 2-CNF show 2-CNF-SAT is $\in P$ i.e. $\exists$ a polynomial algorithm that decides 2-CNF-SAT

$$\phi = (x_1 \vee x_2) \wedge (x_3 \vee x_4) \wedge \cdots \wedge (x_{n-1} \vee x_n)$$

$$x \vee y \leftrightarrow \neg x \rightarrow y$$

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

~~the~~ seems to be an XOR.

34.5-1 To show something is ~~A~~ NP complete, say L, we have to show

2 things

1) it is in NP

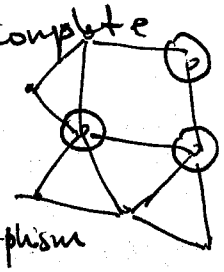
2) it is NP-hard. $\Rightarrow \exists$ a NPComplete problem L' $\ni$

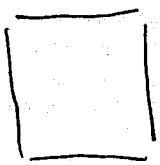
$L' \leq_p L$

1) Given an instance of subgraph-isomorphism we take as our certificate the ~~[illegible]~~ bijection ~~isomorphism~~ fn f. we can certainly check that ~~given~~ $(u,v) \in G_2$ iff $(f(u), f(v)) \in G_1$.

This involves only a polynomial # of applications of the function f to verify.

2) I must find a known NPComplete problem & show that a given instance of this problem can be decomposed into an instance of Graph isomorphism





two graphs G_1 & G_2 $\ni$ if G_1 is a subgraph of G_2 the original instance is a yes instance & if G_1 is not a subgraph of G_2 then the original instance is a no instance

I don't see easily which NP problem to try & reduce from?

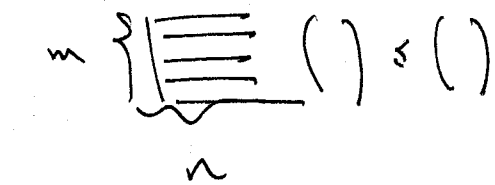
pg 960 CLR

04-21-02

36.5-2

$A \in \mathbb{Z}^{m \times n}$ $b \in \mathbb{Z}^{m \times 1}$

$x \in \{0,1\}^{n \times 1}$



$Ax \leq b$.

To prove something is NP complete we must show 2 things

1) it is in NP
2) it is NP-hard

to show #1 consider a proposed solution. This certificate x can easily be checked in polynomial time thus integer-programing $\in$ NP

to show #2, lets find a known NP complete problem say 3-CNF-SAT

& show 3-CNF-SAT $\leq_p$ integer-programing.

Assume we have an instance of 3-CNF-SAT

$\neg\phi(x_1, \ldots, x_n) = C_1 \wedge C_2 \cdots \wedge C_m$ w/ $C_i(x_1, \ldots, x_n)$ implicitly w/ only 3 literals.

If I could produce from each C_j A matrix A

x_1 x_2 x_3

pick b to be a vector of length m w/ 3 in every location

~~pick~~ Eg:

$C_{10} = x_1 \vee \neg x_5 \vee x_7$ $x_1 = 1$ or $x_5 = 0$ or $x_7 = 1$

$$\begin{array}{l} \text{10th row} \downarrow \quad 1 \quad 1 \quad 1 \\ \quad\ \uparrow \times \times \times \uparrow \ \times \uparrow \\ \quad \text{1st col.} \quad \text{5th col} \quad \text{7th col.} \end{array} \begin{pmatrix} 1 \\ x \\ x \\ x \\ 0 \\ x \\ 1 \\ \vdots \end{pmatrix} \leq 2 = \text{\# of } \cancel{B} \text{ non negated litterals}$$

$\overset{y_1}{\boxed{\neg x_1}} \vee \neg x_2 \vee \neg x_3$ $x_1 = 0 = x_2 = x_3$

$$\begin{matrix} 0 & 0 & 0 & \\ 1 & 0 & 1 & \cdots \\ 1 & 0 & 0 & \end{matrix}$$

$$\begin{pmatrix} 1 & 1 & 1 \\ & & \end{pmatrix}\begin{pmatrix} x\ 0 \\ x\ 0 \\ x\ 0 \\ \vdots \end{pmatrix} \leq 2$$

$x_1 \vee x_2 \vee x_3$ $x_1 = 1$ or $x_2 = 1$ or $x_3 = 1$

$$\begin{pmatrix} 0^c & 0^c & 0^c & \cdots \\ \vdots & & & \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \end{pmatrix} \begin{matrix} 1\,0\,0\,1 \\ 0\,0\,1\,1 \\ 0\,1\,0\,0 \end{matrix} \leq \cancel{3}\ 0$$

For "trues" get anything from

$1c, 2c, 3c$

For falses get anything from

0

$\phi = C_1 \wedge C_2 \cdots \wedge C_m$ removing all not letters of y's

$\cancel{\phi(x_1, \ldots, x_{n+p})}$

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

$y_1 \vee y_2 \vee y_3$

My Idea is to transform $\phi(x_1 \cdots, x_n) = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ into a system of $m \times n$ equations w/ constant weights such that if the three variables for each literal were correct the g would be 1 + the sum would less than a computed by value. But I couldn't seem to find what #'s would make this work.

$\neg x_1 \vee x_2 \vee x_3$

$c x_1 + d x_2 + d x_3 < f$ $\qquad x_1, x_2, x_3 \in \{0, 1\}$

$c x_1 + \cancel{c x}. \; d(x_2 + x_3) < f$

then if $x_1 = 0$ or $x_2 = 1$ or $x_3 = 1$

$x_1 = 0 \quad x_2 = 0, x_3 = 0 \Rightarrow 0 < f$

$x_1 \neq 0 \quad c + d(x_2 + x_3) < f$

$1 + d(x_2 + x_3) < f$

Pg 960 CLR

36.5-3 t expressed in unary a string of k ones.

$$\text{"}t\text{"} = (\underbrace{1, 1, 1, \ldots, 1}_{t \text{ ones.}})$$

I don't see how this affects things at all.

36.5-4

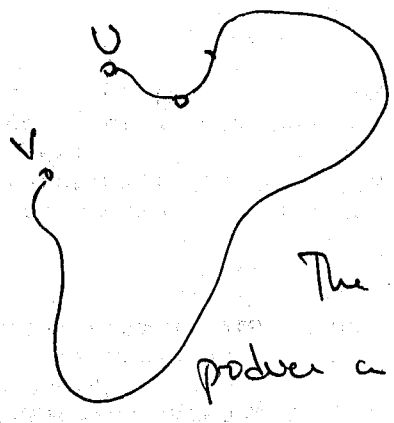
1) Given a certificate of the set A one can easy compute in polynomial time whether the two sums are equal.

2) We show subset sum $\leq_p$ set-partition

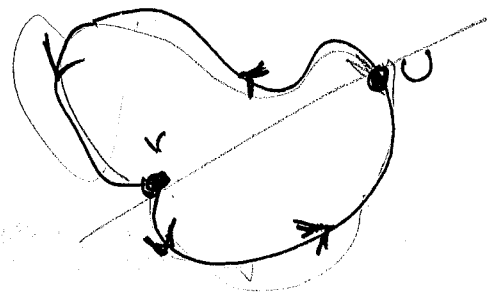
~~Et~~ ~~S~~ Given an instance of set-partition create an instance of subset sum in the following way. Add up all the #'s & set ones target equal to $\frac{\sum_{a \in S}}{2}$. Then if this instance of subset sum is a yes or no answer so must the instance of subset-partition. This transformation is obviously polynomial time & Thus Subset ~~sum~~ partition $\leq_p$ Subset sum.

36.5-5 Hamiltonian Path = simple path that visits every vertex exactly once, starting at U & ending at V.

We showed in problem 36.2-6 that this problem is in NP.



Given An instance of Hamiltonian Path, consider an instance of Hamiltonian cycle w/ starting vertex V.



The problem is that this might produce a HC that has U in the middle of the tour rather than the end like is required.

36.5-6 1st construct a decision problem.

~~1) Verification~~

2nd show verification ~~of~~ of a certificate is easy.

3rd reduce

$G=(E,V)$, $w(E)$, k + $\exists$ a cycle of length $\geq k$.

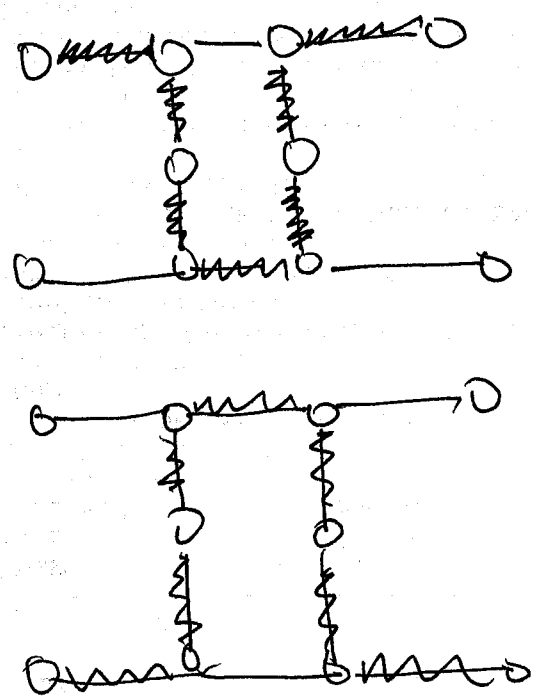
Why would this # k be not a bounded choice by the hamiltonian cost?

$\leq_p$ TSP w/ identity transformation.

Assuming all weights on edges are positive

PS 961 CLR

3b. 5-7



It seems to be a simpler widget?

pg 961 CLR

36-1 Independent Set Decision problem:

(a) Given a graph $G=(V,E)$ does there exist a independent set of size greater than or equal to k.

To pr. that this problem is NP complete we must show 2 things

1) That it is in NP
2) That it is NP-Hard

For the 1st Assume we have a certificate for Independent set Decision We can loop over all the edges of the graph checking that each edge is incident on at least one vertex in the certificate. This can be done in polynomial time.

We can also check that there are $\geq k$ elements in the independent set. Since all of this can be done in polynomial time we can verify a certificate for this problem in polynomial time & Thus it is an element of NP.

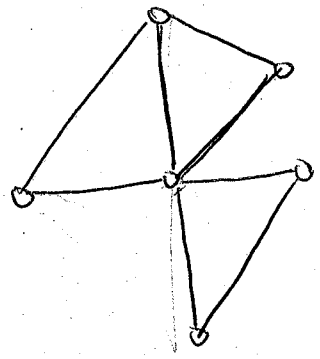
To show it is NP-hard it suffices to polynomially reduce some problem (known to be NPC) into a given instance of Independent-Set

To this end we will ~~consider~~ reduce from the Clique problem

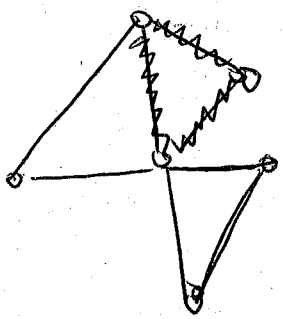
$$CLIQUE = \{\langle G,k\rangle : G \text{ is a graph w/ a clique of size } k\}$$

Clique is maximally connected subgraph

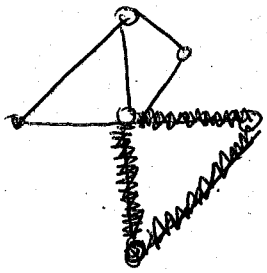
consider :



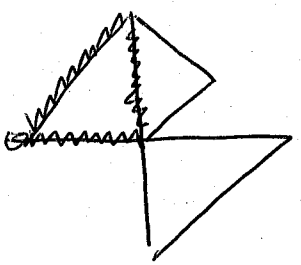
There are two cliques of size 3



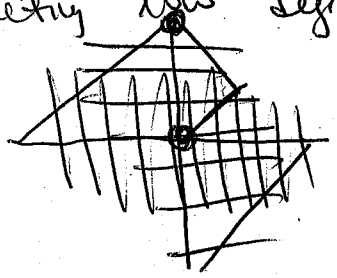
and

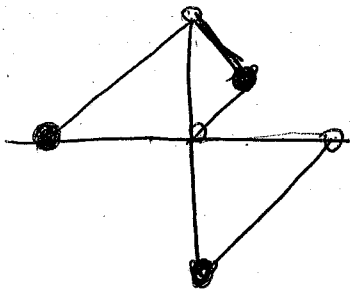


and

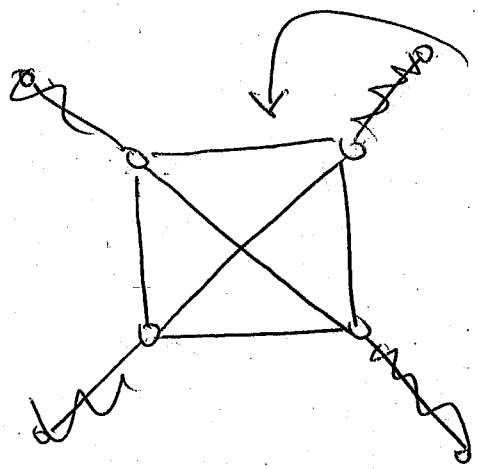


And the maximally independent set would ~~[illegible]~~ be heuristically taken from selecting low degree nodes for example





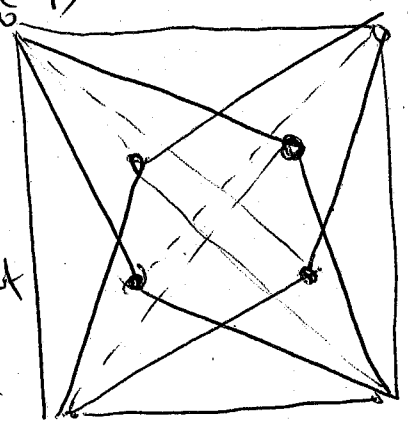
04-21-02



Clique (size 4)

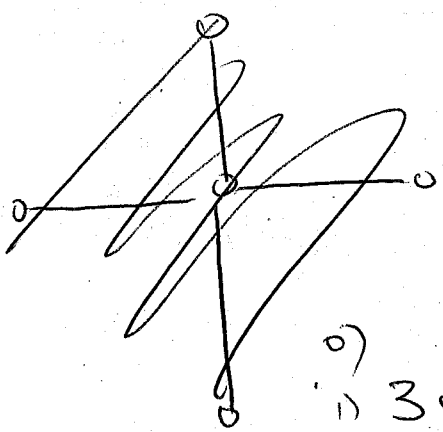
$\Longrightarrow$

graph complement



is ~~a~~ an ~~clique of~~

No cliques of ~~[illegible]~~ ~~[illegible]~~ ~~a~~ "empty set". No

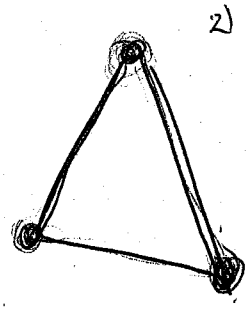


$\Longrightarrow$

0)

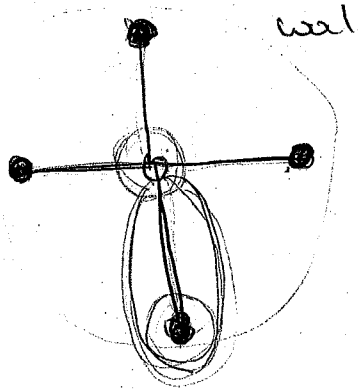
1) 3 cliques of size 2 of size 2

2) 1 clique of size 3. of size 3



$\Longrightarrow$

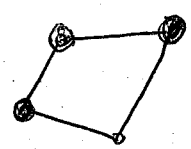
(b) If I can ~~solve~~ solve the decision problem. I start w/ The graph would have all other ~~vert~~ vertices as its maximal independent set.

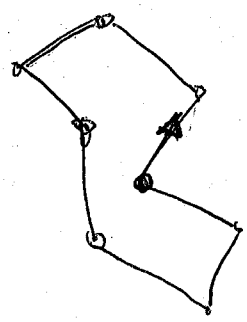


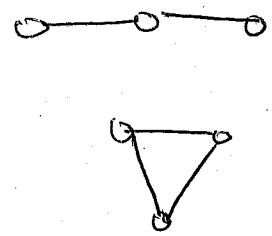
1) 1st find the size of the maximal independent set, ~~by~~ by using a bisection method starting w/ 1 & |V|.

Starting w/ All degree one nodes

(C) If each vertex has degree 2. Then my guess is that any ~~two~~







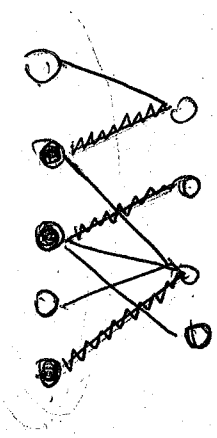
A graph ~~that~~ ~~only~~ where each vertex only has degree 2 must form a closed loop like the examples shown. Pf. Starting at a given node pick one of its two edges to leave on. When this edge enters a node then can be only one edge to travel out of this node on. We travel from node to node never visiting the same node twice

If the graph is finite

Thus the maximal ~~[illegible]~~ independent set would take every other node but not picking the last node so that it violates the condition of being incident on at most one node

$T(n) = \Theta(n)$

(d) From 27.3 using the Ford-Fulkerson method we can solve the maximum-bipartite matching problem in $O(EV)$ time



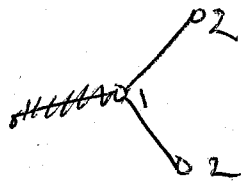
{ This gives the maximum # of edges one can have between the two sets & not have 2 edges incident into a node.

I think one would simply partition the vertex set into $V = L \cup R$ & select all of the Left or Right vertexes as the maximal independent set depending on which set is larger.

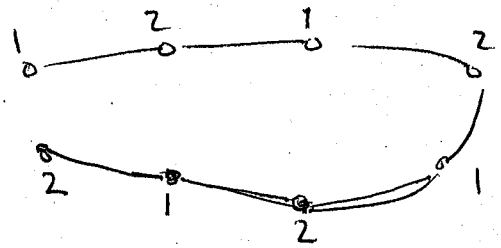
PS 962 CLR 04-28-021

36-2

(a) Assuming a 2 coloring exists ⇒ No vertex can have degree greater than two, for if it did then it would require more than 2 colors



An efficient algorithm would simply be to number the vertices one after another in alternating colors.



(b) The decision problem would be: given a graph and an integer k does there exist a graph coloring w/ k or fewer colors.

~~I can solve~~

I can solve the decision problem in polynomial time ⇒ I can solve the graph coloring problem in poly time.

Pf: Simply performing |V| calls to the decision problem will result in Decision(1), Decision(2), ..., Decision(|V|) must result in a yes returned.

I can solve the ~~[illegible]~~ graph coloring problem in poly. time ⇒ I can solve the decision problem ~~to~~ in polynomial time.

~~[illegible]~~ Just solve the full problem & compare inputs ~~of~~ to the

decision problem w/ the solution to the full problem.

(c) Our decision problem must be NP-complete, for if it was NOT

~~$L_{3\text{-color-Decision}} \leq L_{3\text{-color}}$~~

since $L_{3\text{-color}} \leq L_{3\text{-color-Decision}}$

This could make $L_{3\text{-color}}$ ... ?

(d) $\phi = \underbrace{(\quad) \wedge (\quad) \wedge (\quad) \cdots}_{\substack{m \text{ clauses} \\ \text{w/ } n \text{ variables}}}$

want to transform $\phi \Longrightarrow G = (V, E)$

$x_1 \quad \neg x_1 \quad \cdots \quad x_n \quad \neg x_n$