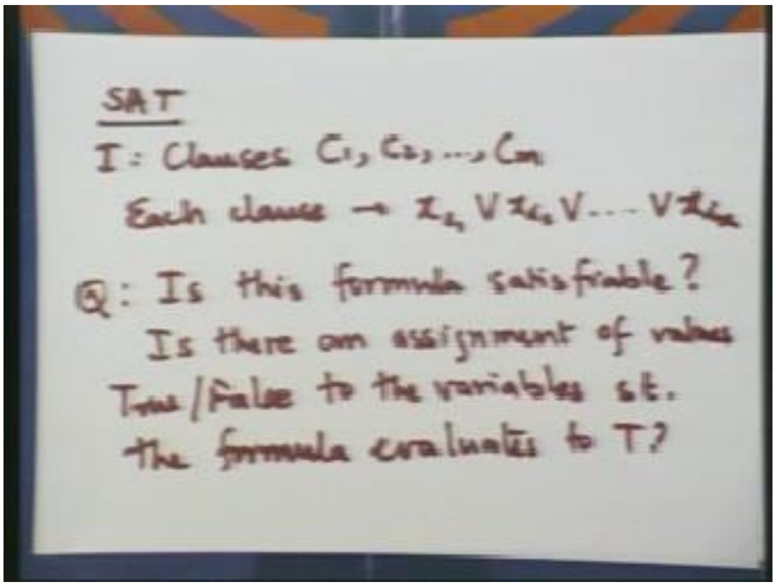
Design and Analysis of Algorithms
Prof. Sunder Vishwanathan
Department of Computer Science Engineering
Indian Institute of Technology, Bombay

Lecture - 29
NP- Completeness – IV

(Refer Slide Time: 00:55)



SAT

I: Clauses $C_1, C_2, \ldots, C_m$

Each clause $\rightarrow x_{i_1} \vee x_{i_2} \vee \ldots \vee x_{i_k}$

Q: Is this formula satisfiable?

Is there an assignment of values True/False to the variables s.t. the formula evaluates to T?

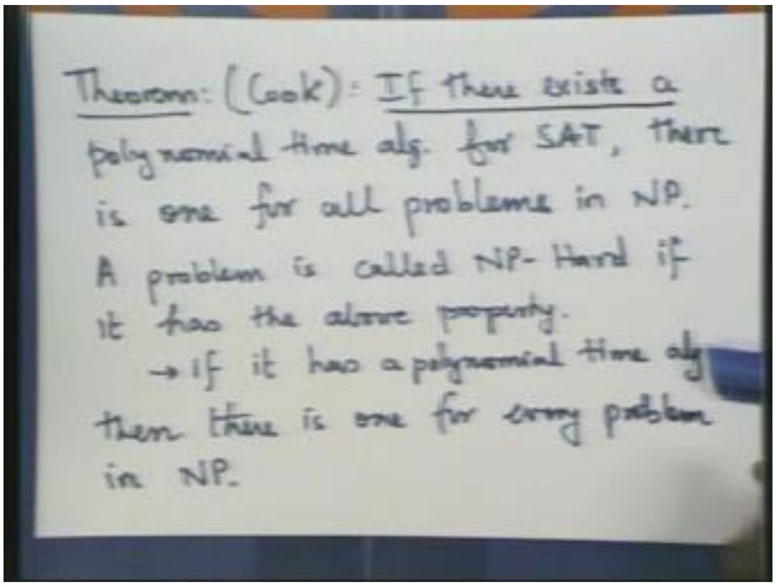
Let us do a quick recap, we have defined problem called SAT. The input was SAT was clauses C 1, C 2 so on up to C m. Each clause was an or of literals. So, it could be something like x i 1 or x i 2 or so on up to i k. So, we had clauses C 1 through C n, each clause was of this form. And the formula we consider is the think of the formula C 1 and C 2 and C 3 so on up to and C m. And the question we ask is, is this formula satisfiable. In other words, is there an assignment of values true or false to the variables such that the formula evaluates to true. So, this is the question.

So, let me just go over this again, you have clauses C 1, C 2 and C m. The formula we are interested in is C 1 and C 2 and C 3 and so on up to C m. Each of these C i's looks like this. These are you have x i 1, x i 2 and so on up to x i k. You could also have negations also. For instance if you put x i 1 or x i 2 bar and so on. So, each clause is an or of literals. And the formula is an and of clauses. And the question you ask is, is this formula satisfiable.

In other words, is there a assignment of values true or false to each variable. Such that, when you evaluate this formula, the formula evaluates to true, which means, in each clause when I look at each clause there must be some literal, which must evaluate to true. Either there should be a x and x should be set to true or there should be a x bar, where x is said to be false. If x is set to false x bar is true. So, in each clause there must be at least one literal, which is, which evaluates to true.

Then we say the formula evaluates to true and such an assignment. An assignment to these variables that gives you the value true is called the satisfying assignment. So, the question we ask given a formula is does this formula have a satisfying assignment, the formula of which are of this kind, where you have and of clauses where each clause is or of literals. These formula's are called formula's in CNF conjunctive normal form. This all this means all are same. So, this is the problem SAT. And we also had a theorem.

(Refer Slide Time: 04:53)

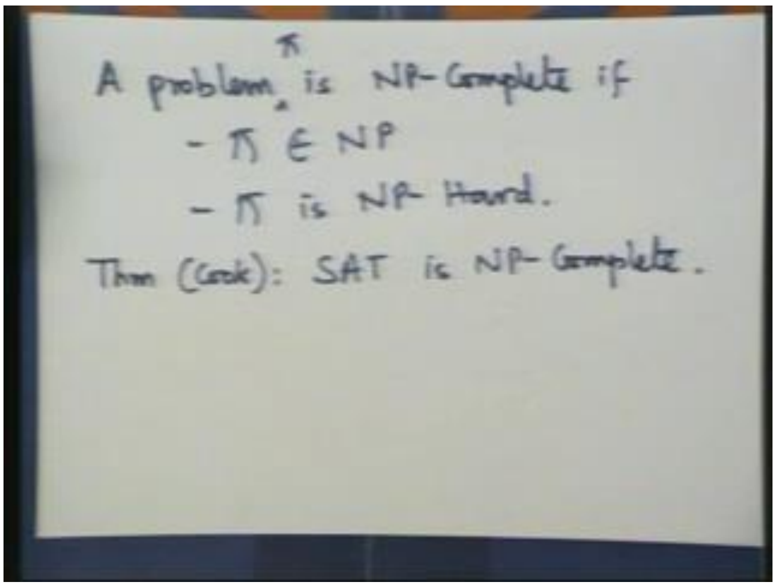


Theorem: (Cook): If there exists a polynomial time alg. for SAT, there is one for all problems in NP.
A problem is called NP-Hard if it has the above property.
→ if it has a polynomial time alg then there is one for every problem in NP.

So, this theorem is due to cook and it is called cook's theorem. It states that, if there is a polynomial time algorithm for SAT. Then there is one for every problem in N P. So, if there exists a polynomial time algorithm for SAT there is one for all problems in N P. So, in other words satisfiability, the problem SAT is among the hardest problems in N P. You can solve this efficiently, which is a efficient algorithm to solve SAT. There is one for every other problem in N P.

So, this somehow identified one problem, which is among the hardest problems in SAT. In fact, there is a name for these problems. So, we call these problems N P hard, these are hard problems in N P. So, let me define this. So, a problem is called N P hard if it has the following property. If it has the above property which means, if what is the property, the property is this. If there exists a polynomial time algorithm for this problem there is one for every problem in it. So, the properties we are looking for is this, if it has a polynomial time algorithm. Then there is one for every problem in N P. To be ((Refer Time: 07:32)) more correct we say that, if there is a polynomial time algorithm that solves this problem. Then there is one for every problem in it. So, these are the hardest among the problems. And hence, they are called N P hard. There is one more definition I need, which is this a problem is called N P complete.

(Refer Slide Time: 07:56)



A problem π is NP-Complete if

- $\pi \in NP$
- π is NP-Hard.

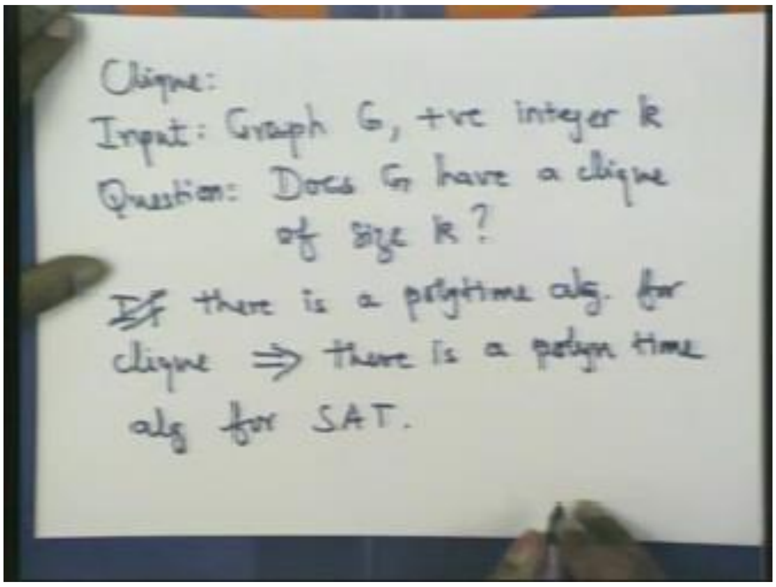
Thm (Cook): SAT is NP-Complete.

A problem is N P complete if it has to satisfy two things. Let us say, this problem is pi. Pi must belong to N P and pi is N P hard. Then it is called N P complete. So, apart from being among the hardest problems of N P. ((Refer Time: 08:28)) It should be see when you say a problem is N P hard. We do not require the problem to in N P. The problem could be outside N P. As long as it has the property that if you can solve this problem. You can solve everything else in N P it is called N P hard.

But, for N P complete we require the problem to be in it. The problem should be in N P and it should be also be N P. That is the only difference. So, now we can state cook’s theorem in a different way. The theorem due to cook, it states that SAT is N P complete. We saw that SAT was in N P. So, SAT belonged to N P and cook's theorem essentially says it is also N P hard. So, it means, that putting these two things together, we know that SAT is N P complete.

Essentially, N P complete problems are just to state it again. They are among the hardest problems in N P. They have to be in N P and they are among the hardest problems. So, our goal is to add many problems to this set. This set of N P complete problems and will begin with clique, which also we defined last time. So, let me just define problem clique.

(Refer Slide Time: 10:11)



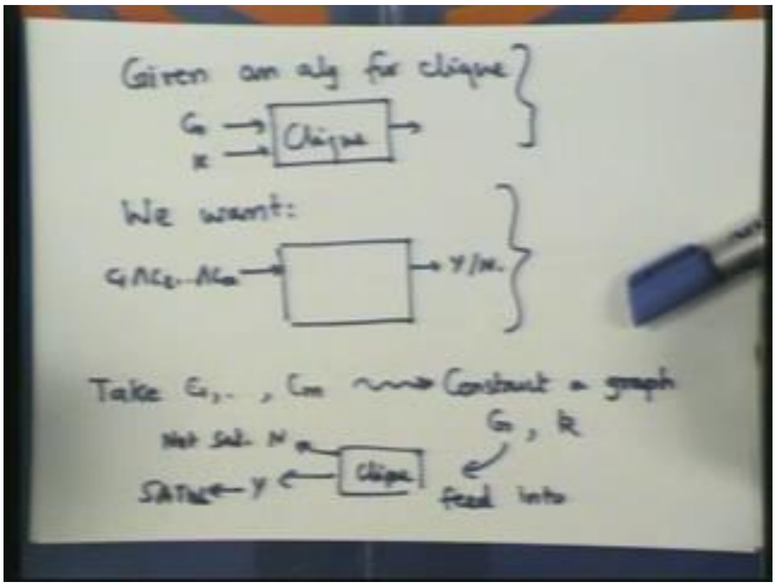
Clique:
Input: Graph G, +ve integer k
Question: Does G have a clique of size k?
~~If~~ there is a polytime alg. for clique ⇒ there is a polyn time alg for SAT.

So, problem clique is as follows, the input is a graph G. And the positive integer k. And the question we ask is this. Does G have a clique of size k. So, this is clique, we saw last time that this problem is in N P. We gave there was a proof that, the Prover could give the verifier. So, that the verifier by looking at the proof and the input would check that a given graph, in fact, did have a clique of size k.

Our objective now is to prove the other part, which is the clique is N P hard, which means, if there is a polynomial time algorithm to solve clique, there is one to solve everything else in N P. Again, we saw I think I sort of entire how will do this and this is how we do it. So, what we do is this. We have to use cooks theorem to do this. So, we show the following, if there is a polynomial time algorithm for clique. We will show that this implies then there is one for let me remove this clique.

So, there is a polynomial time for clique will imply that there is a polynomial time algorithm for SAT. So, this is what we will prove. There is a polynomial given there is a polynomial time algorithm for clique will show that, we will construct a polynomial time algorithm for SAT. Now, cook's theorem says that, if there is a polynomial time algorithm for SAT there is one for everything in N P and we will use this. So, just following these two implications we can see that, we prove what we want. So, all that remains now is I will assume that there is a polynomial time algorithm for clique. And I will construct one for SAT. So, here is how we do it. So, we have a algorithm for clique.

(Refer Slide Time: 13:01)



Given an alg for clique
G
k
Clique
We want:
$C_1 \wedge C_2 \ldots \wedge C_m$
Y/N
Take $C_1, \ldots, C_m$ ⇝ Construct a graph
G, k
feed into
Clique
Not Sat. N
SAT Y

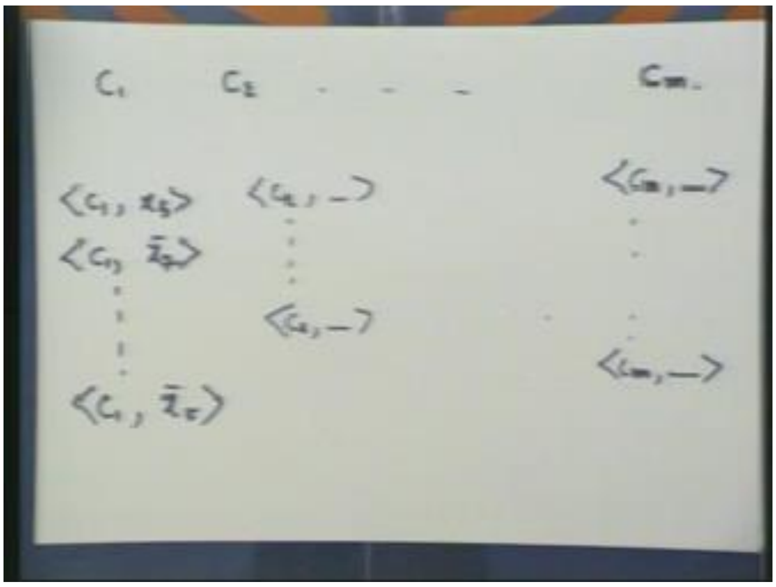
So, this is given, we are given an algorithm for clique. Now, what does this state input. It takes input as graph G, it takes k. And it says yes if the graph is a clique of size k. And it says no otherwise. Now, what we want is this. We want a black box, which will take. So, this is beyond a algorithm for SAT. So, the input to the SAT is a collection of clause, so C 1 and C 2, C m. Now, you want to know whether it is satisfiable or not. Now, what do these two answers, so somehow, we have this sub routine.

We have to use this as a sub routine to solve this problem. So, here is an example, where actually the two things actually look quite different. This is a problem on you know it is something to do with logic. You have all this variables and you have clause this is a problem on graph. So, this is available in a sub routine and we want to construct something. So, how do we sort of use something, sub routine which ((Refer Time: 14:27)) graphs to construct to determine whether a formula is satifiable or not.

So, here is a trick. So, what we do is this. So, we take these clause. So, we take C 1 through C n. Now, somehow I will tell you how, to construct a graph G and some k. Feed this to feed into, so it is called this algorithm clique into clique. And if this says yes, we will say that the formula is satisfiable. If this says no, we will say not satisfiable. So, let us look at this. We take these formulas's C 1 through C m. We construct some graph G and some k. And we feed this into sub routine feed.

Now, this will say yes or no, if it says yes means we will we want to say that, the formula is satisfiable. It says no then we will say no. Now, how do we do this transformation. From these clauses we have to construct a graph. And from a clique, what we get as output is graph has a clique of size k. Somehow clique of size k should indicate that the formula is satisfiable. So, let us see how this is done. So, you take the formula C 1 through C m. And here is how you construct the graph.

(Refer Slide Time 16:32)



So, here are the clauses C 1, C 2 so on up to C m. Now, for each clause I will have a number of vertices in the graph. So, let us say, C 1 x 5. I will tell you what, this x 5 are x 7 bar, C 1, x t bar. If the clause C 1 were x 5, x 7 bar or, or, or up to or x t bar. Then I have a vertex for which each of them. In other words, I have a vertex for each literal clause literal pair. So, a literal appears in a clause I have a vertex for that. Similarly, I have listed them for C 2 so on. So, C m will have it is own set of vertices.

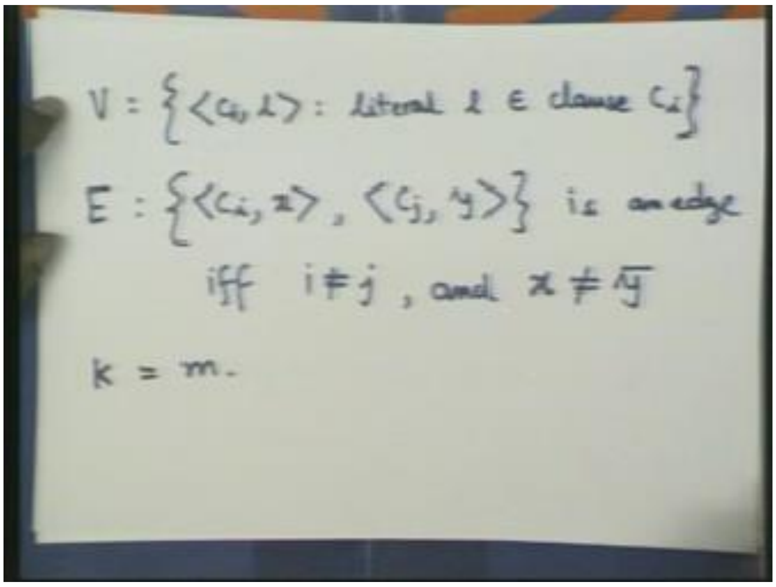
So, for each clause literal pair I have a vertex I have divided the vertex set into these end parts one for clause. Now, what do we want is that, from a clique of size k. We still have not specified what k is from a clique of size k I should be able to figure out that there is a satisfying assignment. So, what does the satisfying assignment look like, the satisfying assignment essentially says, for each clause which of them I can set to true, which literal I could set to true. May be I can set x 7 bar to true, which means x 7 to false.

So, may be here I have C 2 and x 30. May be x 30 I can set to true. Somewhere, down the line let us say, C 45 I have x 30 bar. I could have x 30 here and x 30 bar elsewhere and so on. Now, somehow each of these clauses I want to set one of them to be true. I want to make that one of them is true. And this I should be able to pick out with a clique. So, it stands to reason that I would have a clique of size m. And this clique I should pick by picking one vertex from each of these sets.

I should be able to pick one vertex from each of these sets, if I am picking only one vertex from each of these sets. Then each of these sets may be an independent set. I do not want edges between any of these. All the edges in the graph ((Refer Time: 19:21)) may only go between vertices ((Refer Time: 19:23)). Now, let us look at these two C 2 30 and C 45 x 30 bar. Now, clearly I cannot if I say this is true, I cannot set this to true, this will immediately become false, which means, I should not be able to pick these two in my clique.

Remember, the way we are going to get from a clique to a satisfying assignment is, when i pick the vertices in the clique they will decide, which literal in a clause is satisfied. This is roughly the intuition that I want. So, from each vertex the clique I will pick one vertex from a clause. And you can tell me which i can, which is the literal which is satisfied. So, if i pick x 30 in clique 2, I cannot pick x 30 bar in clique 30 or 37. So, these two I should not be able to pick, which means there should not be any edge between them. Otherwise, I can add all the vertices. So, this roughly actually is the concept. So, let me formally tell you what the construction is construction of the graph is.

(Refer Slide Time: 20:38)



$V : \{ \langle c_i, l \rangle : \text{literal } l \in \text{clause } c_i \}$

$E : \{ \langle c_i, x \rangle, \langle c_j, y \rangle \}$ is an edge

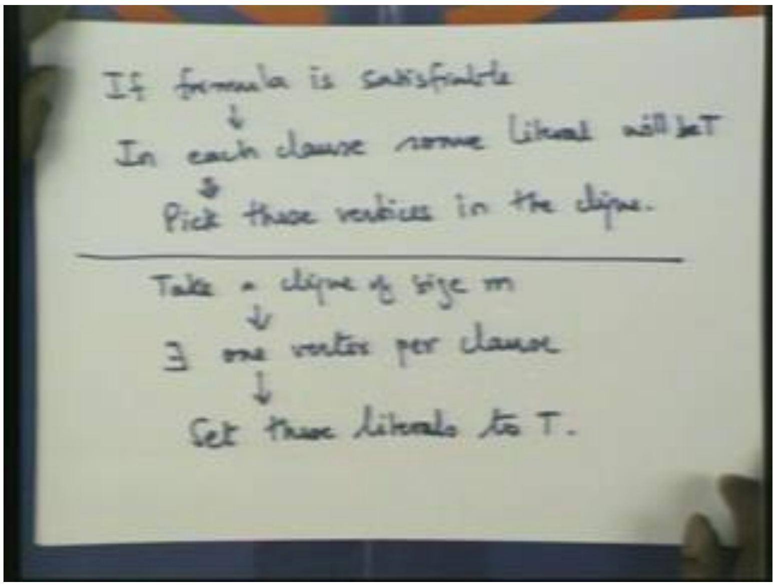
iff $i \neq j$, and $x \neq \bar{y}$

$k = m.$

So, the vertex set. So, vertex set V is consists of c and l, where literal. Let me, make this c i and l, literal l belongs to clause c i. So, there is one for each clause literal pair, if the literal belongs to that clause. So, what are the edges, this is the vertices edge set is c i, x and c j. y is an edge. If and only if first of all i is not equal to j, because edges flow only between vertices corresponding to different clauses and x is not y bar. Because, if x is equal to y bar. Then I do not I cannot pick both of them in my clique.

So, I can pick both of them in the clique, if they have different variables all together. One is x 15 another is x 30, it does not matter what value I set to it. But, I cannot pick x and x bar together. So, this is the graph and k is m. So, given my clauses, set of clauses I construct this instance of clique. I feed it into the sub routine for clique. If it says yes, it has a clique of size m. When I say yes, the formula is satisfiable, if say no I say the formula is not satisfiable. So, let us actually, formally prove this. So, if we actually, informally argued this, but let us quickly sort of do a recap. So, supposing the formula is satisfiable.

(Refer Slide Time: 23:00)



If formula is satisfiable
↓
In each clause some literal will be T
⇓
Pick those vertices in the clique.

Take a clique of size m
↓
∃ one vertex per clause
↓
Set those literals to T.

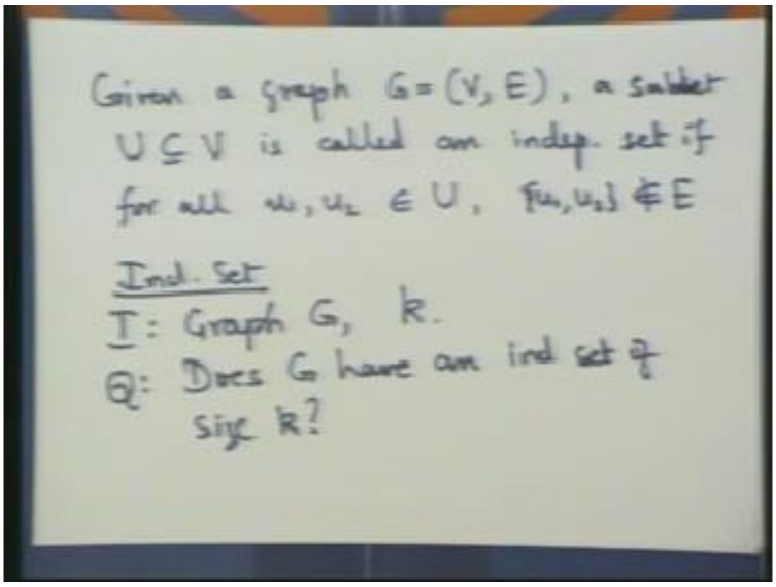
So, if formula is satisfiable, what does this mean in each clause. So, look at any satisfying assignments. If there is a satisfying assignment look at any satisfying assignment, in each clause some literal will be true. So, pick all this, so pick these vertices in the clique. So, we have picked m vertices. So, we have picked one vertex per clause though we have m vertices. And clearly, there will be a edge between any two of them. Because, Firstly they are in different clauses.

And if they are the same sort of they used the same variable. Then they cannot be negations of each other. So, these in fact, form a clique in the graph. And in fact, the other sort of other direction is also similar. So, take a clique of size m, which means there exists one vertex per clause. So, I am going to set these literals to true. So, these literals which corresponds to these vertices set them to true.

And you can check that this can be a satisfying assignment, in this clique with a variable x appears x bar will not appear. So, this will not be a satisfying assignment. ((Refer Time: 25:04)) We will just prove that clique is N P hard we saw last time that clique is empty. So, put these two things together, we have just shown that clique is N P complete. So, the thing to notice is that once we have cook's theorem in hand. It is much easier to prove some other problem is N P complete.

All you need to do is prove that, if there is a polynomial time algorithm for this new problem. There is one for any of the old problem, which you have already proved N P complete for instance SAT. But, now you can use clique. So, our next problem is called independent set. So, let us define what a independent set is.

(Refer Slide Time 25:51)



Given a graph $G = (V, E)$, a subset $U \subsetneq V$ is called an indep. set if for all $u_1, u_2 \in U$, $\{u_1, u_2\} \notin E$

Ind. Set

I: Graph G, k.

Q: Does G have an ind set of size k?

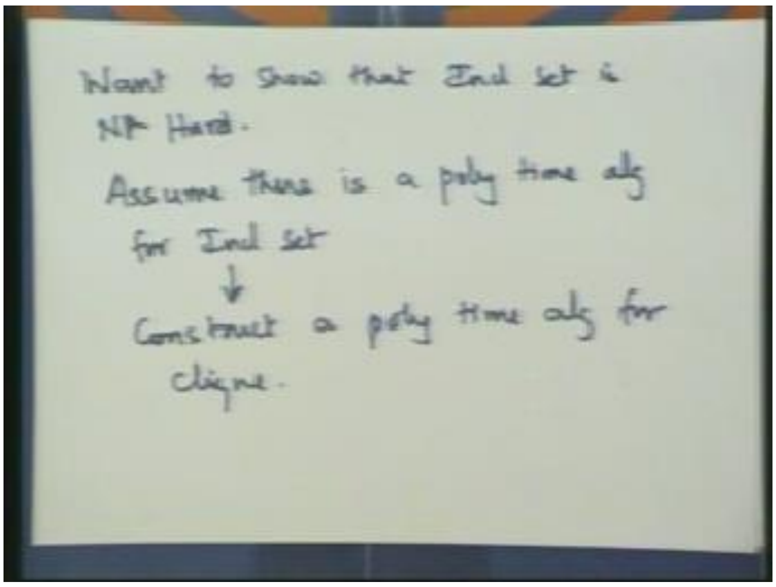
So, given a graph G, this is the vertex set this is the edge set a subset U of V is called an independent set. If there are no edges between any pair of vertices in U. If all u 1, u 2 belonging to U. u 1, u 2 is not an edge. There are several subsets, so that if I take any two of them, there should not be an edge between them. This I mean for clique he wanted an edge to be in every pair. Here, you do not want a edge between any pair. So, our next objective is to prove the version of the independent set. Decision version of independent set problem to be N P complete, but the search was an independent set essentially states that given a graph find an independent set of maximum size. The decision so we convert this to a decision version in a fashion, which is very similar to clique. So, it looks like this. So, independent set problem independent set has input graph G, the positive integer k. The question you ask is does G, have an independent set of size k. This is the question we ask.

So, the first thing to see is this problem in N P. The well I hope most of you can now show that this problem is in N P. Anyway, let us do it. So, if the answer to the question is yes, we are looking at only yes inputs, inputs for which the answer is yes. And for these inputs, the prover should be able to convince the verifier that in fact, it is a yes input. So, we need to figure out, what is the hint or proof or advice that the prover gives the verifier.

In this case, again it is just the independent set. Then he just picks out k vertices from the graph. And tells the verifier here, these vertices form independent set. The verifier looks at these verifiers. He checks that there is no edge between any of them. So, that is what he verifies and that is very easy to verify. And we convince that, in fact, it is a yes input. The advice that the prover gives the verifier in all most all cases is supposing the question is yes, why is this question yes. You just answer the question.

So, here for instance if it is yes if G has an independent set of size k. So, focus on the independent set of size. If this is given to the verifier and the verifier can check that it is in fact, good. So, this problem is in n P, our job is my job for the time being is to prove that it is N P complete. So, let us do it. So, what we do is this.

(Refer Slide Time 30:01)



Want to show that Ind set is NP Hard.

Assume there is a poly time alg for Ind set

↓

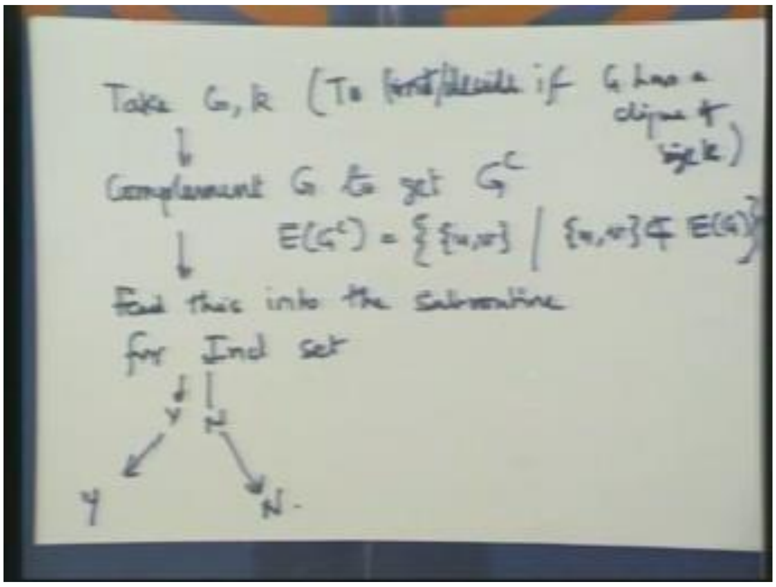
Construct a poly time alg for clique.

So, we will the strategies is we want to show, where independent set is N P hard. How do we do this we will this. So, assume there is a polynomial time algorithm for independent set. So, assuming this will construct a polynomial time algorithm or clique. What does this give us? We know that clique is N P complete, which means if there is a polynomial time algorithm for clique there is one for everything in use good. So, that is what we will use. And you can see that we have done, if we can do this step.

Assuming that, there is a polynomial time algorithm for independent set. We construct polynomial time algorithm for clique. And because, there is one for clique, there is one for everything in n p. So, to prove essentially that, if there is a polynomial time algorithm for dependent set, there is one for everything. This is not as difficult as the previous one. And independent set and clique they are both sort of problems in graphs. And we also look feel similar in one case we have an edge, in the other you have subset where there are no edges.

And in fact, this is exactly what we do. So, supposing I have a polynomial time algorithm for independent set. Now, how do I find for a graph as a clique of size k. So, I take this graph, where ever there is an edge, if there is an edge between two vertices I remove it. If there is no edge between two vertices I add an edge. So, I sort of complement the edge set. And in this new graph I feed into the independent sub routine. If it says there is a independent set of size k. Clearly, these k vertices must form a clique in the original value. Because, I complemented the graph, well this is all there is. So, let us just write this down.

(Refer Slide Time 32:38)



Take G, k (To ~~find~~/decide if G has a clique of size k)

$\downarrow$

Complement G to get G^c

$E(G^c) = \{ \{u,v\} \mid \{u,v\} \notin E(G) \}$

$\downarrow$

Feed this into the subroutine for Ind set

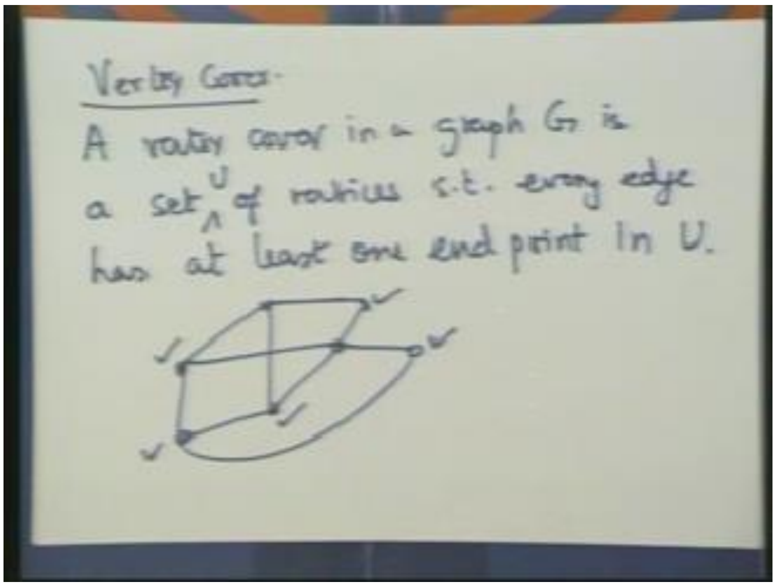
$$\begin{tikzcd} & Y \arrow[dl] & N \arrow[dr] & \\ Y & & & N. \end{tikzcd}$$

So, here is the transformation. So, take G and k. So, complement G to get G c. So, what is the complement essentially, the edge set is the same. I have complemented, the vertex set is the same the edge set I have complemented. So, the edge set of G c is, so u, v such that, u v is not E of G. If it is not there I put it in. If it is there I do not put it in. So, this is the complement. So, essentially the complement of E G. Now, keep this into so feed this into the sub routine or the independent set.

And if, this says yes then you output yes. If this says no then you output no. So, we recall that what we want is to find or decide to decide if G has a clique of size k. This is what we are doing, we are going to decide whether G has a clique of size k. We complement the graph feed this into the sub routine for independent set. And if, this independent set sub routine says the yes then we say yes, when we say no, the answer is no. And you can check that this box.

I mean in the sense that, if G has a clique of size k. G complement also will have a size of k. If G does not have a clique of size k, I am sorry if G has a clique of size k, then G complement will have a independent set of size k. If G does not have a clique of size k. And G complement will not have a independent set of size k. So, the answer in both case is correct. Our next problem is called vertex cover.

(Refer Slide Time 35:20)



Vertex Cover.

A vertex cover in a graph G is a set $\overset{U}{\wedge}$ of vertices s.t. every edge has at least one end point in U.

Vertex cover, so what is a vertex cover. So, a vertex cover in a graph G is a set of vertices. Such that, every edge has at least one end point in the set. Is a set let us say, U of vertices, so that every edge has at least one end point in U. Now, let us take an example. This graph let us see, suppose I pick this, then I take this. This let us say that, now, I need to cover this edge. So, hopefully I have covered all edges. So, this edge, this end point is in, for this edge this end point is in, this edge this is in, this edge this is in.

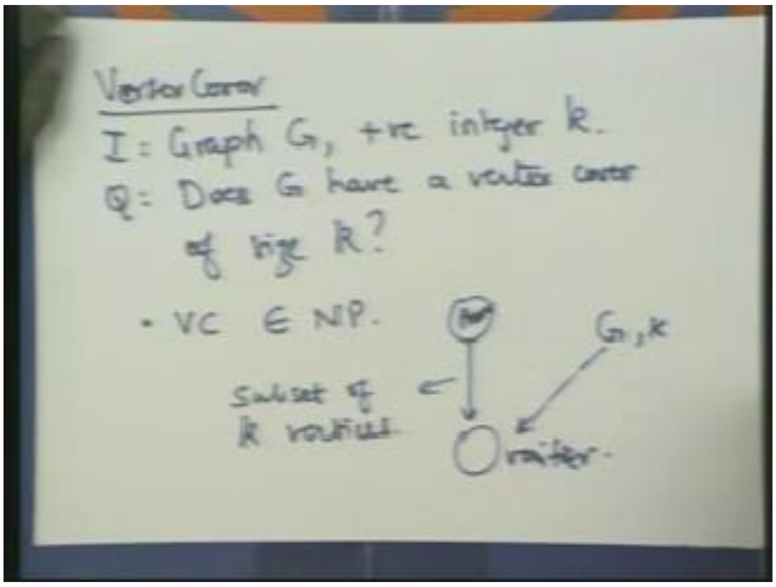
For these both are in, but that is fine. Similarly, for this edge both are in and that is fine this is in the vertex ((Refer Time: 36:57)). So, if I pick these vertices, which are tick marked, that forms a vertex cover in the graph. It is a subset of the vertex set. So, that every edge has at least one end point. For instance, supposing you have a computer network. You have a network and you want to monitor each link. You want to monitor each link.

What do you want to do is pick a set of these nodes. Pick a set of these computers and to each computer you can assign a link. To each computer you can assign a link. So, that it monitors traffic on the links both ways from the computer and from going into the computer. And you want a subset of the computers ((Refer Time: 37:45)) every link is covered, which means for every link at least one of the end points must be chosen to monitor it.

So, this is the vertex cover ideally, we would like to pick very small number of computers to do this job. And in fact, we will look for vertex covers of minimum size. Given a graph find the vertex cover of minimum size is the optimization or the search problem that one looks for. And our objective is to show that this problem is hard, in the sense that you can solve this problem. Then you can solve every problem in N P in polynomial time. So, this is among the hardest problems in N P.

If you want to pick up minimum number of computers to, so that traffic on every link can be monitored this problem is N P hard. Unlike that you will come up with a algorithm. So, that is what we are going to do good. So, I just mentioned the search version of vertex cover, what is the decision version. Remember, if you want to prove something is N P complete, we look at the decision version they are easier to handle. So, here is how we do it the trip is the same. We introduce this extra positive integer k. And we ask if the graph has a vertex cover of size k. So, if it has a vertex cover of smaller size, clearly it has one of size k. So, here is the problem.

(Refer Slide Time 39:34)



Vertex Cover

I: Graph G, +ve integer k.

Q: Does G have a vertex cover of size k?

• VC $\in$ NP.

G, k
Subset of k vertices
verifier

So, vertex cover input is a graph G, the positive integer k, the question is does G have a vertex cover of size k. So, I will leave it as an exercise for you to do the following as a exercise for you to do, which is supposing there is a algorithm for this problem. And argue that there is an algorithm for the problem of finding minimum vertex cover. It is very similar to the things that we have done in the past.

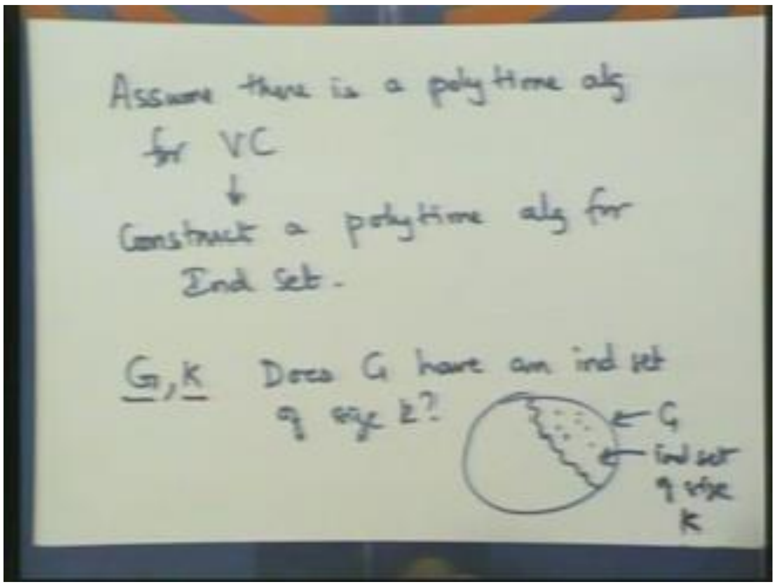
The other direction is simple. If you can find vertex cover of small size I mean smaller size and clearly given this question you can determine whether, there is a vertex cover of size k. If that size is smaller than or equal to k the vertex cover you found. And there is a vertex cover of size k. So, adding vertices to a vertex cover, it still remains a vertex cover. So, you can form as large vertex cover that is not a problem. You put all the vertices in the vertex cover.

If you take all the vertices in the graph it will give you a vertex cover. So, here is the problem and we would like to prove that this is N P complete. The first thing we need to do is to prove that it is in N P. So, the first step is V C belongs to N P. Why is this true? So, what we do is, so what is we go to our usual game. So, here is the prover, here is the verifier and here is the graph G and K. Now, what does the prover give the verifier? The verifier is looking at this graph G and K.

What does the prover give the verifier? So, that the verifier can verify that there is a vertex cover of size k. Well, the prover just gives the subset. Subset of k vertices, so this is the prover gives the verifier. Now, the verifier looks at the subset. And when he looks at every edge. And checks that, this for every edge one of the end points is in the subset. So, that is what the verifier checks. He takes the subset of size k from the prover.

Now, he goes through the edges one by one and for each edge checks that at least one of these edge these two end points is in the subset and the prover is correct. So, this proves that vertex cover is in N P. Now we need to show that vertex cover is N P hard. So, given we want to show that, if there is a polynomial time algorithm for vertex cover, there is one for everything in N P.

(Refer Slide Time 43:04)



Assume there is a poly time alg for VC

$\downarrow$

Construct a poly time alg for Ind Set.

$\underline{G}, \underline{K}$ Does G have an ind set of size k?

G
ind set of size k

So, we will show we will assume that, there is a polynomial time algorithm for vertex cover. So, what we will do is we will construct a polynomial time algorithm for dependent set. Instead of independent set I could have chosen clique or I could have chosen satisfiability. Either of these one of these three problems I could have chosen. In fact, as we go along, we could choose any of these already proved N P complete problems here.

But, trick is to choose the right one. So that, we choose the right sort of problem, then this proving becomes easier. So, we assume that there is a polynomial time algorithm for vertex cover. We will construct the polynomial time algorithm for independent set. Now, how does one do this, what is the relationship between vertex cover and independent set.

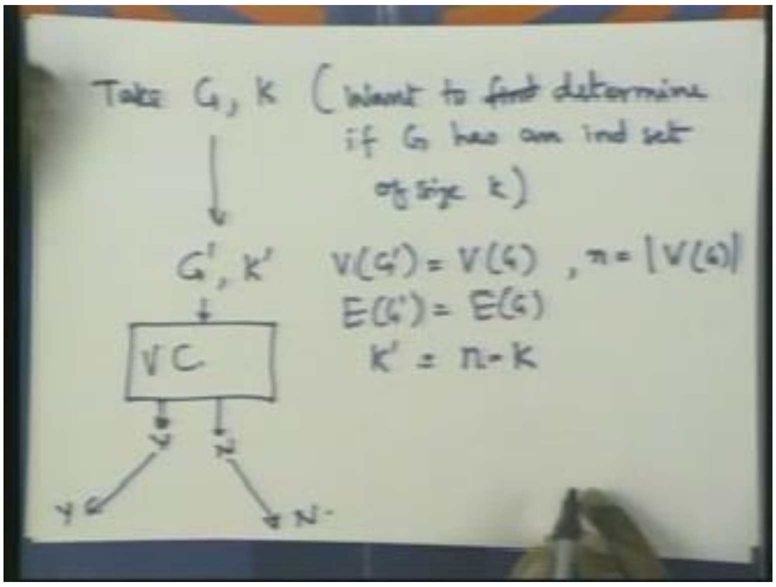
So, here is my graph G and K. I have a graph G and K and I want to find out if this graph G.

Does G have an independent set of size k? This is what we want to sort to prove that, we want to construct this is the algorithm we want to construct one for independent set. So, supposing this is my graph G. Now, supposing there is a independent set of size k. Let us say, this is the independent set. Supposing this portion of the graph is independent set. Let us, look at what can we say, we know that there are no edges here. There are no edges between any pair of vertices here, where are the edges in the graph.

Well, the edge can be on this side. Or the edges can be of this form. Edges are either completely in this side or one end point is here and one end point is here, which means that this portion is a vertex cover. If this portion is a independent set, this portion is a vertex cover. And in fact, this goes this implication goes the other way. If this portion is a vertex cover and the rest of the graph better be an independent set. Because, there cannot be a edge here. Every edge must have one end point in the vertex cover.

So, if a subset of the vertex set is a independent set. The complement of this set is subset will be a vertex cover. And if it is a vertex cover the other will be a independent set. So, we will use this to prove. So, essentially if you want to find independent set of size k. We will look for a vertex cover of size n minus k, where n is the number of vertices. So, if there is a vertex cover of size n minus k, then clearly there is a independent set of size k. And if there is no vertex cover of size n minus k, there is no independent set of size either. So, that is all there is. So, what we do is we just take the same graph, instead of k we take n minus k. And then we call the algorithm for vertex cover. So, given a algorithm for vertex cover, we are considering one for independent set.

(Refer Slide Time 47:08)



Take G, K (want to ~~find~~ determine if G has an ind set of size k)

G', K' V(G') = V(G) , n = |V(G)|

E(G') = E(G)

K' = n - K

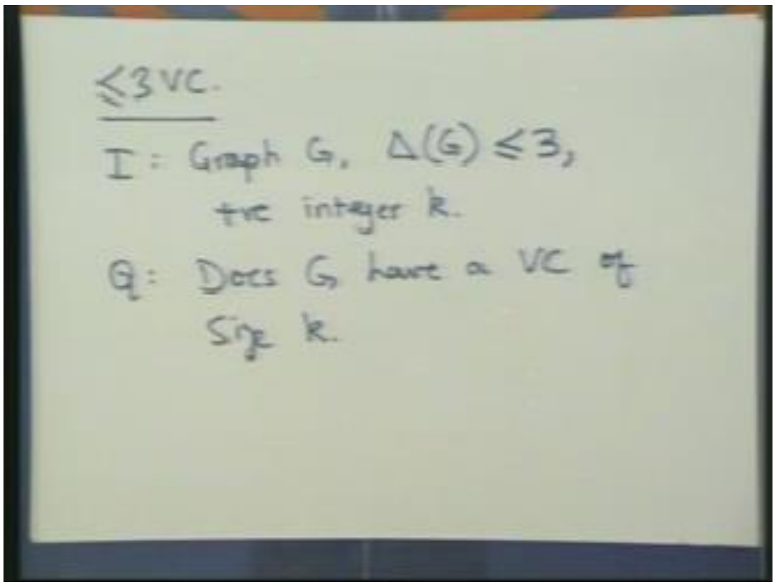
So, we take G and K. So, we want to find well we want to determine, if G has an independent set of size k. What we do is we take G prime and K, where the vertex set of G prime is the same as the vertex set of G. The edge set of G prime is also the same as edge set of G. This would be K prime, K prime is nothing but, n minus K, where n is the size of the vertex set. So, this is the same graph as the original. So, we just copy the graph down. And K prime is nothing but, n minus K.

And now, you ask is this pair. For this pair I feed this into sub routine for vertex cover. So, you ask does this is spread into sub routine V C. If it says yes, then you output yes. If it says no, then you output no, that is it. So, G and K are input. So, this is an algorithm for independent set. G and K are input, I take the same graph and I take N minus K as K prime. G prime is the same as G and K prime is N minus K. This I feed into sub routine for vertex cover, which we have assumed.

Remember, we have assumed that there is a polynomial time algorithm for vertex cover. And if this says yes, then our output is yes, if this says no our output is no. And it is easy to see we have argued that, if G has a vertex cover of size N minus K. It has a independent set of size K. And if it does not have a vertex cover of N minus K, it does not have a independent set of size K. So, we have just added new problem to our list of N P complete problems. We have just proved that vertex cover is N P complete.

So, we started out cook's theorem gave us SAT. SAT was N P complete. And we then showed that clique is N P complete then we showed that independent set is N P complete and now you have showed that vertex cover is N P. So, our next problem is very similar to vertex cover. But, it is a restricted form of vertex cover.

(Refer Slide Time 50:28)



$\leq$3 VC.

I: Graph G, $\Delta(G) \leq 3$,
+ve integer k.

Q: Does G have a VC of Size k.

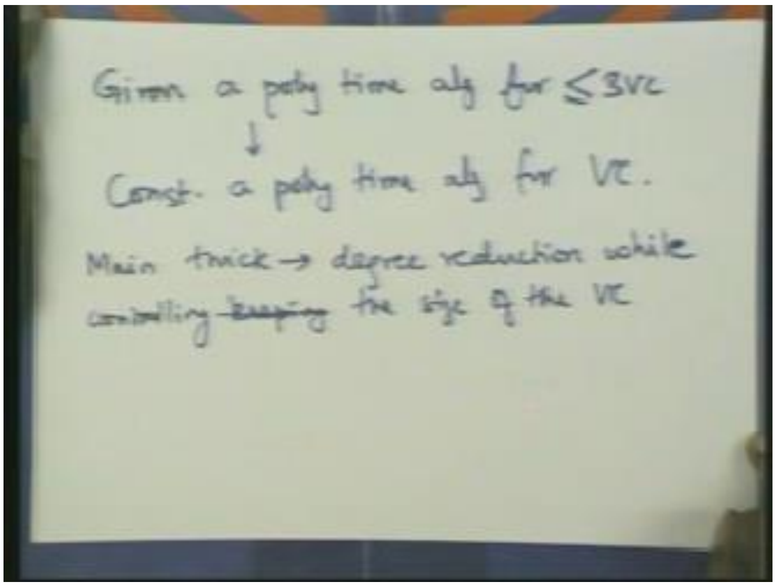
So, this is called less than equal to 3 V C. So, the input is a graph G, such that the maximum vertex degree is at most 3 and the positive integer k. The question is same does G have a vertex cover of size k. Without this, if I omit this just graph G and integer positive k, this is just vertex cover. I add this additional constraint to the input. The only inputs I can look for is graphs where the maximum vertex degree is 3. Delta of G is the maximum vertex degree in G. Delta G is max degree of a vertex in G.

So, look at the degrees of vertices in the graph. And delta is the maximum. So, the maximum degree I want is 3. So, this problem is like vertex cover, the only thing is that we focus on graphs on only some graphs, not all graphs. Only graphs which have maximum vertex degree at most 3. Now, is this as hard we know that vertex cover in general is the hard problem.

Is this problem as hard, is it as hard is it much easier to find vertex covers in graphs, where the maximum degree is just 3 and not more than 3. The answer is no. In fact, even in this case the problem turns out to be hard. So, even for graphs where the degree of a vertex does not exceed 3 vertex cover problem vertex cover is hard. We will show that this problem is N P hard or N P complete. To prove that, this is in N P is very similar to the vertex cover thing and I will not do it. So, just do it. I hope most of you can do it.

All of you can do it good. We will just prove that, less than equal to 3 V C is N P hard. Now, which of the problems, which have already been proved N P complete should be used for this, I guess vertex cover is a closest. So, once you sort of once you go for vertex cover I mean it is correct. So, vertex cover we will do. So, what we will do is given an efficient algorithm polynomial time algorithm or less than equal to 3 V C. We would like to construct efficient algorithm polynomial time algorithm for V C general V C. So, let me write this statement now.

(Refer Slide Time 53:59)



Given a poly time alg for ≤ 3VC

$\downarrow$

Const. a poly time alg for VC.

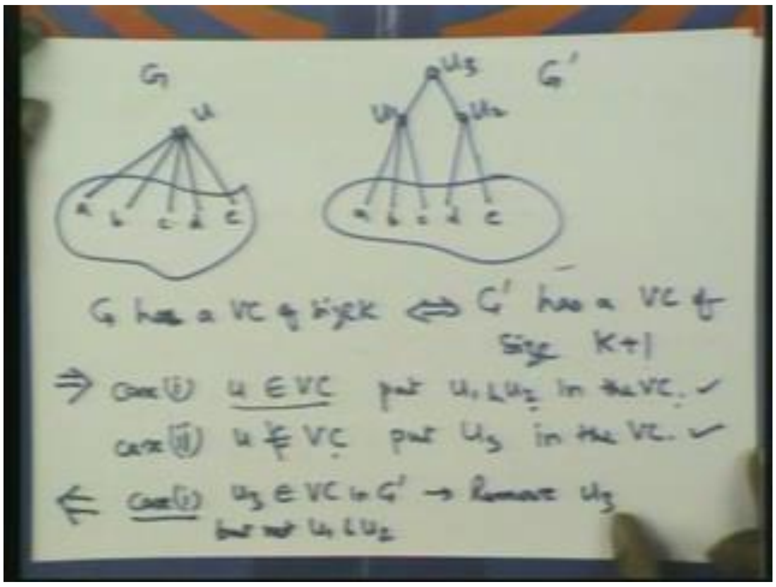
Main trick $\rightarrow$ degree reduction while controlling ~~keeping~~ the size of the VC

So, given polynomial time algorithm for less than equal to 3 V C. We want to prove that there is we want to construct a polynomial time algorithm for V C. This is what we want to do. And once we do this, we know that from the previous construction that there is one for everything in N P. So, let me just sort of say what this means, see going the other direction is easy. Supposing, you have polynomial time algorithm for V C.

Clearly, if you restrict the input in any way, it does not sort of I mean you can still feed the same input to the algorithm. But, here you cannot just do that. Your input is a graph and k. Now, this graph could have vertex degrees, which are much higher than 3. What do we do with these? That is the ((Refer Time: 55:05)). If all vertex degrees were 3 then there is no problem, I just feed this into algorithm for less than equal to 3 V C ((Refer Time: 55:13)).

So, what happens when that of vertices with degree greater than 3, 4, 5, 6 whatever up to N minus K largest N minus 1, when the trick is to somehow decrease that decreasing the vertex degrees by creating new. So, given this original graph i construct u graph, where the vertex degrees are smaller. And somehow, you know you must have some control over the vertex cover. The vertex cover in the new graph must be related very closely related to the vertex cover in the older. So, here is the trick. So, I will tell you what the trick is to reduce the degree of the vertex. So, the trick, main trick is degree reduction while keeping the size of the V C, while controlling. The size of V C if there is some controlled over how it changes, when we are here. So, let us take.

(Refer Slide Time: 56:46)



G has a VC of size K $\Leftrightarrow$ G' has a VC of size K+1

$\Rightarrow$ Case (i) $\underline{u \in VC}$ put u_1 & u_2 in the VC. ✓

case (ii) $u \notin VC$ put u_3 in the VC. ✓

$\Leftarrow$ $\underline{\text{Case (i)}}$ $u_3 \in VC$ in G' $\rightarrow$ Remove u_3 but not u_1 & u_2

So, here is the vertex of degree 5. Somehow I need to that is here is rest of the graph. So, what do we do is this, if you change the degree I break it up into three parts, I break this vertex into three parts. So, here is the rest of the graph. ((Refer Time: 57:08)) three of them here. So, let us see, what I have done. See there are 5 of them I have just broken it up into two parts. Let us say, 1, 2, 3 this is a, b, c, d and e. a b c. So, this one vertex I have broken up into three vertices.

I have added two new vertices. And you can see that, vertex degrees are decreasing. Initially I had one of degree 5. And now, if I look at these vertices, there is one of degree

2, one of 3 and one of degree 4. So, these vertex degrees I have decreased, if this were 4 I would not have one of these edges. Let us say, that a was not there. Then I am done. So, I have three vertices one of degree 2 and two of degree 3. I am done. I have decreased vertex degree of degree 4 vertex.

If it is 5 I will have one with 4. If I have larger again I just split it into two parts. Of course, I have to do this repeatedly. I take this graph I split this vertex into two. So, I decrease the degree. And I keep doing this, till every vertex has degree 3 or less. That is the rough idea. Let us see, what happens to the vertex cover, when I do this split. So, just to recap. I look at this graph look at if there is a vertex of high degree I split it into I split this degree up. I split this degree into two parts. And add a new add another vertex.

So, instead of vertex and I have a three vertices. And the degree has been reduced. Initially, they were 4 ((Refer Time: 59:03)). They have two of them are degree 3 and 1 is degree 2 and so on. So, I keep doing this. Now, I have to somehow say, that the vertex cover does not change much. Even if it changes there must be I must say that it changes clearly by this much. Only, then can I make the connection between the vertex cover in the final graph. And the vertex cover in the original graph.

Final graph has degree of vertex at most three, that is fine. I should also be able to say that, if there is a vertex cover of this size in the final graph. If and only if there is a vertex cover of size k the original graph, and that we will see right now. So, supposing the claim is this. So, here is the graph G and here is G prime, which I get this way. Now, the claim is that G has vertex cover of size K, if and only if G prime has a vertex cover of size K plus 1, which means the size of the minimum vertex cover goes up by 1.

So, this is my each time I use this split the vertex cover size goes up by 1 ((Refer Time: 1:00:28)). So, why is this true? So, let us prove both ways of this inequality. So, let us prove this. Supposing G has a V C of size K. Why should G prime should have a vertex V C of K plus 1. So, take the V C, which G i. There are two cases. So, case 1 is let us call this vertex u. And here let me call these u 1, u 2, u 3. Case 1 is u belongs to the V C. Now, u is in the V C then put u 1 and u 2.

The rest of the vertices are in the vertex cover remain the same. There are some these vertices in the vertex cover in this portion of the graph. And I have used. I put u 1, u 2 and the same set of vertices in the vertex cover. And now, let us observe that this is also a vertex cover. So, the vertex cover size has gone up by 1. If there were l plus 1, l vertices here and 1 vertex here, then I have l vertices here and 2 vertices. So, it is l plus 1. Why is this the vertex cover. Now, if the edges completely in this portion it is covered.

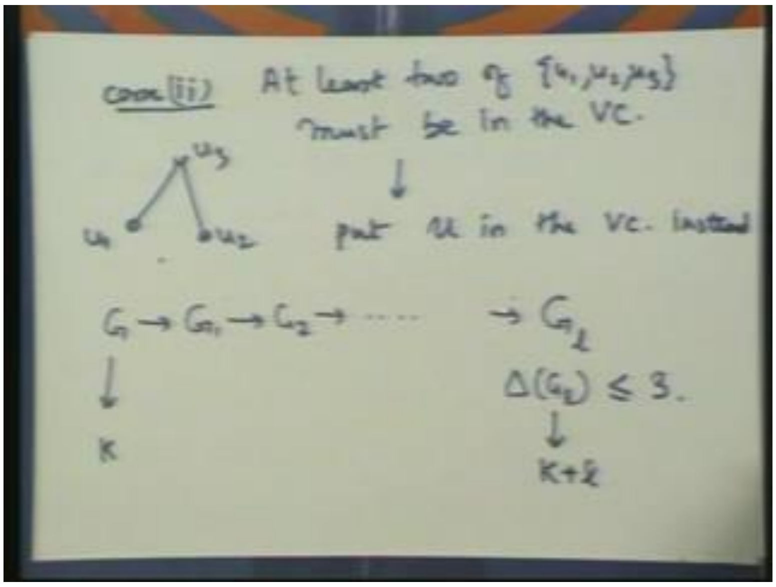
Now, since both u 1 and u 2 are in these edges are covered. And these two edges are also covered. So, every edge is covered. So, this is a vertex cover. Now, case 2, u is not in V C. Then put u 3 in the V C. Now again let us, check this is the vertex cover. Now, all of these vertices all of this edges a u, b u and so on. Have to be covered, which means a b c d e must be in the vertex cover, u is not in the vertex cover. So, all these edges are covered by the vertex cover in this portion. Since, a b c d e is in the vertex cover.

So, the only edges we need to worry about is u 1, u 3 and u 2, u 3 and because we put u 3 in the vertex cover these four also covers. So, this is also ((Refer Time: 1:02:54)). Just to recap, if you is in the vertex cover remove u from the vertex cover and put u 1 and u 2 to get the new vertex cover in G prime. This is the vertex cover. If u is not in the vertex cover, take the old vertex cover and put u 3 additionally in this vertex cover. And this will be a vertex cover in the graph. This proves one direction.

So, we need to prove, the other direction in which that is also very, very simple. Now, just to observe let us look at u 1, u 2 and u 3. Now, there are two cases again. So, only look at the vertex cover here. I will show that the vertex cover of 1 size 1 less on this side. Whatever, vertex cover is there on this side I will show there is 1 of size 1 less. So, the case 1 is u 3 belongs to the vertex cover. We remember this vertex cover is in G prime. Then I just remove u 3.

And you can check the rest of the vertices form a vertex cover for this input. I must say that, only u 3 is in V C, but not u 1 and u 2. So, only u 3 among u 1, u 2, u 3; so that is case 1. u 3 is in the vertex cover, but u 1 and u 2 are not in the vertex cover. Now; that means, that a b c d e all of these edges are present in the vertex cover. So, these edges are taken care of and clearly if there is a edge only in this portion it is taken care of here also. Because, this portion of the vertex cover remains the same. I have just removed them. So, if u 3 is in the vertex cover, u 1 and u 2 are not in the vertex cover, then you are done you just remove u 3.

(Refer Slide Time: 1:05:27)


case (ii) At least two of {u1, u2, u3}
must be in the VC.
put u in the VC instead
u3
u1
u2
G → G1 → G2 → ⋯ → Gl
k
Δ(Gl) ≤ 3.
k+l

So, the next case case 2, if it is not this case so let us, look at u 3, u 1, u 2. Then I claim that at least two of u 1, u 2, u 3 must be… Two of these, why 2 of these well, I cannot have only u 1 in the vertex cover. I cannot have only u 1 in the vertex cover, because if both u 2 and u 3 are not there, then this edge is not covered. The case only u 3 is taken care of. So, we must have at least two of these. In this case, I remove this and put remove and put u in the V C instead. Then, I get vertex cover of 1 less.

Then, well what if all three are present. Well this cannot be a minimum vertex. If there is a vertex cover of size there is one of smaller size on the right hand side. So, that does not impede our progress. So, when I do this split, the vertex cover may rise by at most one. The vertex cover will rise by at most one by so then the procedure is clear. I start with G, then I do this sequence of these transformations G 1, G 2. Each time I split a vertex into 3. So, I keep doing this as long as there is a vertex of degree 4 or greater.

So, finally, I end up with G l. I have done this l times and I get a graph. So, delta of G l is less than or equal to 3. So, I want to find out, if G has a vertex cover of size k. What I do is I ask if this has a vertex cover of size k plus l. If it says yes, I say yes G has a vertex cover of size k, which says no I say G does not have a vertex cover of size k. And the proof that it works essentially is we have actually done it. Reason is each time we do a split vertex cover rises by 1.

So, this actually, proves that less than equal to 3 V C is N P complete. We just one small thing we need to worry about. Well, you start with a initial graph start splitting, what if there are too many splits. We have got the polynomial time algorithm for this. We started out with the graph. And supposing you construct a new graph you take lot of time. Then this will not work. We will show next time that this is not true and l is bounded. So, l will not be too large. If you show that l is not too large, then the final graph is not so big and. so our entire procedure will run in polynomial time.