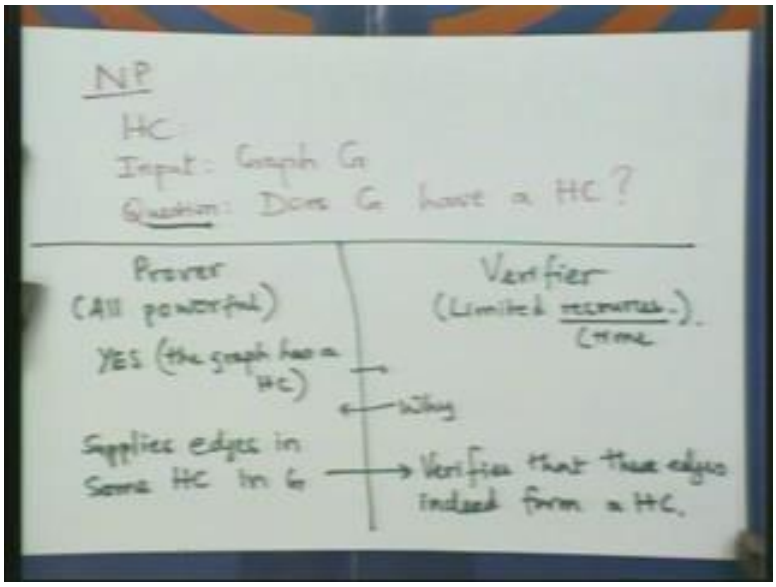
Design and Analysis of Algorithms
Prof. Sunder Vishwanathan
Department of Computer Science Engineering
Indian Institute of Technology, Bombay

Lecture - 28
NP- Completeness – 111

We are ready to define the class NP. We will first do this using an example. So, the example we will pick a problem, the problem we pick is HC.

(Refer Slide Time: 00:57)



The input is a graph G and the question is does G have the Hamiltonian cycle. So, this is the question. Now, forget algorithms for the time being. So, we are not going to discuss algorithms, what we are going to do is discuss a game between two players. There are two players. We will call them prover and the verifier, we will see, why they are called prover and verifier very shortly.

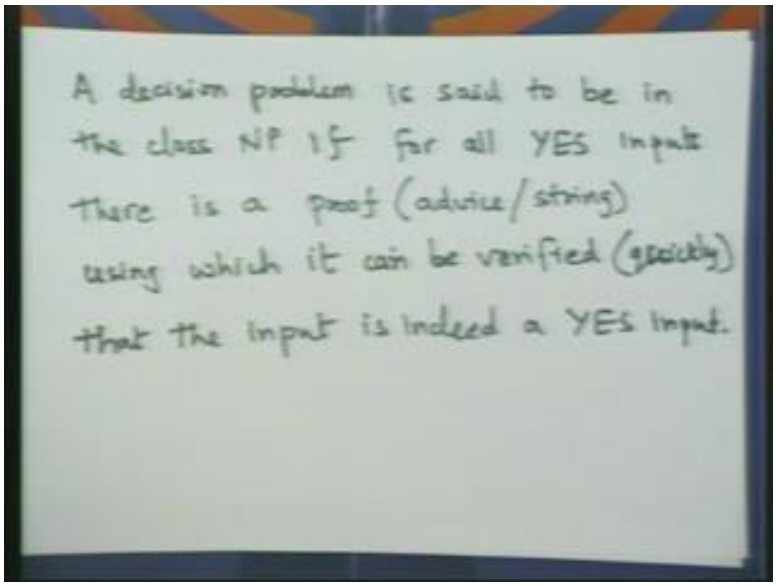
This prover think of them as all powerful, he knows all answers that he supposed to and even answers is not supposed to. And the verifier has limited resources. In particular, he does not have too much time, limited resources or this includes time. Now, the prover and the verifier meet. So, they are sitting in a room and they see a graph in front of it. Now, the prover looks at the graph and says this graph is a Hamiltonian cycle. The verifier, who is bid trustful of the prover, asks him why, why does the graph have this.

Now, the prover can convince the verifier that the graph does have a Hamiltonian cycle very easy, how does he do it. He just gives the edges in the graph, in the form of a Hamiltonian cycle. So, he picks any Hamiltonian cycle in this cycle and he tells he tells the verifier, look at these edges, these edges in the graph form a Hamiltonian cycle. The verifier, even though is sort of limited in resources this much he can do, given a Hamiltonian cycle in a graph you can verify that these edges. In fact, do form a Hamiltonian cycle.

So, let me just write this conversation between the prover and the verifier. So, the prover says yes. So, this yes is an answer to this question above, yes here is an answer to the question. Does G have a Hamiltonian cycle the prover says yes, the graph has Hamiltonian cycle or there is an input graph that both of these are looking at both the prover and the verifier are looking at the prover says yes, the graph has a Hamiltonian cycle. The verifier then asks why. And the prover supplies edges in some HC in G. And the verifier verifies that these edges indeed, form Hamiltonian cycle. So, this is the conversation. Let us go over it again, the prover says yes to this question and to supplement this statement, he supplies the edges in some Hamiltonian cycle in G. And the verifier looks at these edges in the Hamiltonian cycle. And it is easy for him to verify, now that these edges indeed do form a cycle.

This is a game that we will sort of consider and in some ways, we have already proved. I have not defined what a NP is. But, we have actually proved that the Hamiltonian cycle problem is in the class NP. So, let us formally define the class NP. And then we will see, why this constitutes a proof that Hamiltonian cycle is in it. So, Hamiltonian cycle problem is of course, a decision problem. The question we ask is does there exist a Hamiltonian cycle or not it is to be answered yes or no. So, it is a decision problem.

(Refer Slide Time: 06:15)



So, a decision problem here is the definition, a decision problem is said to be in the class NP, if for all YES inputs. Let me write this down then I will explain what this means. If for all YES inputs, there is a proof, you can think of this as an advice or a string. So, this is, think of this as a string the prover gives to the verifier. So, this advice is what the prover will give to the verifier. So, there is a proof using which, it can be verified quickly where, the input is indeed a YES input. So, let us now, look at this.

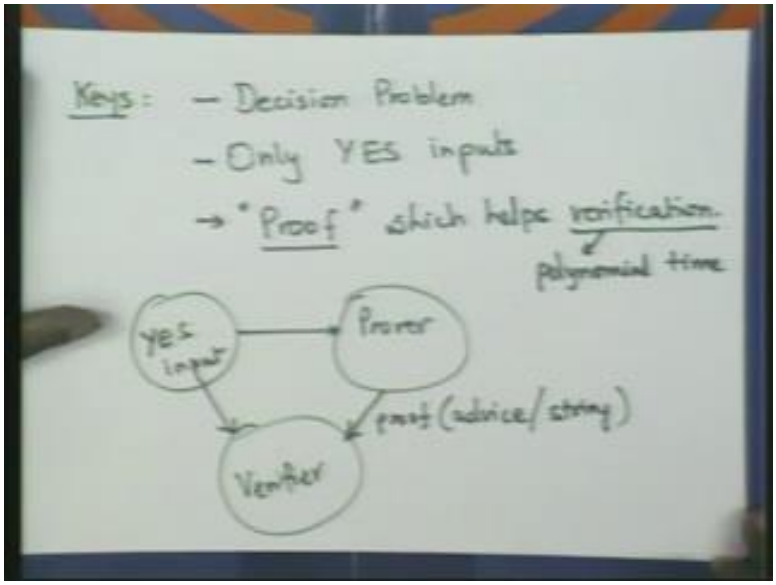
So, there are some terms that I need to describe. So, first thing is what is this YES input, what is quickly and then what does the statement mean, let us focus on the YES input. See this problem is a decision problem, since it is a decision problem for each input the answer is either yes or no right. So, YES input are those inputs, for which the answer is yes. So, we look at only those inputs, for which the answer is yes and we say that for these inputs, I do not worry about what happened to the no inputs.

For the YES inputs there is a proof, this is a proof that the prover gives the verifier, so, this is proof. So, YES input, YES inputs are inputs for which, the answer is YES, this is 1. The proof is what the prover sends to the verifier and quickly in time polynomial in the input size. So, I could have said efficiently, I guess that is what I have been using. But, quickly efficiently they all mean the same thing, which is that, it can be verified in time polynomial in the input side.

So, let me just, let go again this definition again, since it is sort of important. So, let us move over definition again. A decision problem is said to be in NP. If for all YES inputs, there is a proof, there is a proof using, which it can be verified that the input is indeed a yes. So, let us look at our previous example for Hamiltonian circuit. (Refer Time: 10:57) If for all YES inputs for a YES input, this means, if the graph does have a Hamiltonian cycle.

There is a proof that the prover can give the verifier using, which this proof consists of edges in some Hamiltonian cycle. And using this verifier can easily verify, in polynomial time whether the graph has Hamiltonian circuit or not. So, this actually proves that Hamiltonian circuit the problem Hamiltonian circuit is in the class NP.

(Refer Slide Time: 11:41)

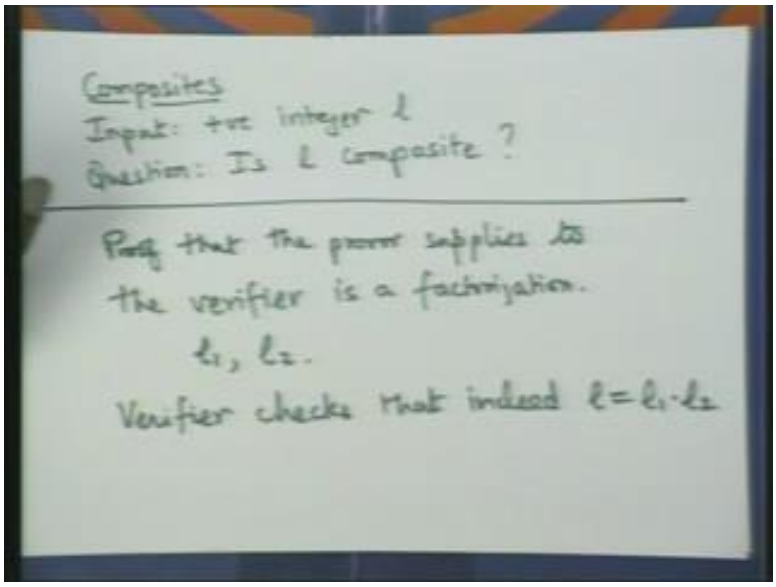


So, let me highlight the main sort of points in the definition. Then we will go over this again. So, first it has to be a decision problem. So, answer has to be yes or no. We, look at only YES inputs and on YES inputs supply a proof. This proof will help the verifier verify that it is a YES input, which helps verification. And this verification has to be in polynomial time. So, these are the initial things, it has to be a decision problem, we only look at YES inputs.

It is only for YES inputs that the prover has to convince the verifier and to convince he can give a proof, which can help the verification. So, here is let me sort of draw this diagrammatically. So, here is the YES input. So, you pick any one of the YES inputs.

The prover is looking at this, shows the verifier. And the prover sends the proof or an advice, sometimes called an advice (Refer Time: 13:28) some string think of zeroes and ones, whichever way the verifier looks at the string and sort of figures out, what the prover means. So on YES inputs, the prover can send some proof to the verifier using, which the verifier in polynomial time can verify that. In fact, this is a YES input. So, let us do one more example.

(Refer Slide Time: 14:06)



This example is a composites, the input is a positive integer l, the question is l composite, it is the question. So, we want to know whether composites this problem composites is NP or not. First thing, it is a decision problem. So, we can it is a problem that we can look at, which could be. In the first thing it has to satisfy that the first property it has to satisfy is that it must be a decision problem which it is. The question asked is l is composite and the answer is either yes or no.

So, what is the next thing for all YES inputs, there must be some way for the prover to convince the verifier that. In fact, it is composite. So, let us just do this. So, we are looking at YES inputs which means, l is composite. So, what is the proof that the prover can give verifier. So, well I guess, it must be sort of obvious by now, that if you, now if you know that something is composite. And you want to convince somebody that it is composite, you can just give the two factors.

So, you give. So, the proof that the prover supplies to the verifier is a factorization. So, he gives let us say l 1 and l 2. So, this is the factorization of l. So, the verifier checks that indeed l equals l 1 times l 2 and neither of these is 1. Okay l 1 and l 2 cannot be 1. So, this proves that this is the proof that the prover can give to the verifier, to convince him at some number is composite.

So, one question I guess which most people sort of encounter when this definition is made. You know, you focus only on the YES inputs what about the no input. So, take composites for the time being. Input is a positive integer l, question is l composite let us say the answer is no, the prover says no it is not composite. Now, how can he convince the verifier that it is not composite, which means you have to somehow convince the verifier that it is prime, you know the only sort of numbers between 1 and the number that divides it are 1 and the number itself, nothing else in between.

Well, to think about it, this is not this does not seem to be as easy. Actually, in this case there is a way to do it. But, ways to do it is certainly not easy, there is a recent research, which says even without an advice, one can figure out whether a number is prime. But, that is way beyond the syllabus, for this course. So, let us again do this for the Hamiltonian cycle problem also, just to get the hang of this here, now (Refer Time: 18:26) when the answer was YES.

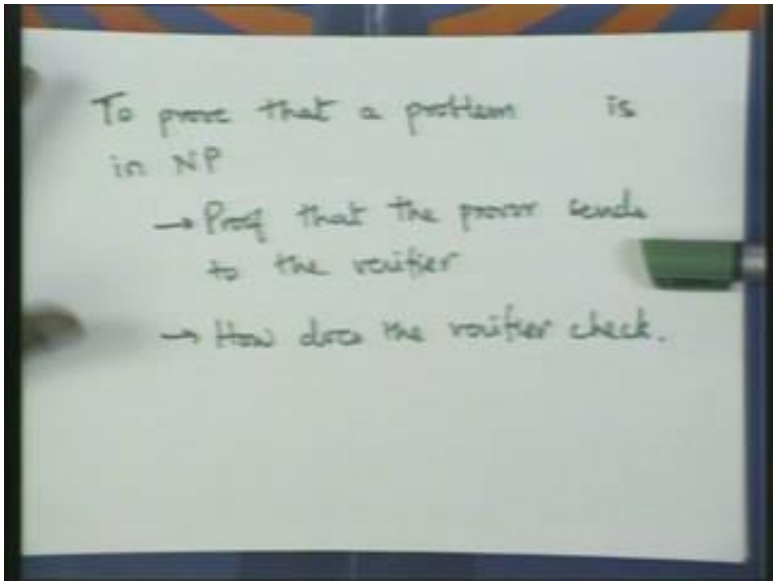
Yes the graph has a Hamiltonian circuit; this proof can be supplied right edges having Hamiltonian circuit in this. Supposing the prover had said no, if we look at the no inputs. How can a prover, how can you convince somebody that graph does not have a Hamiltonian circuit. So, if you think about it for a while, you see that this is actually a very difficult thing. Supposing, you say that this graph does not have a Hamiltonian circuit by some way or method whatever, you know this.

How, do you convince somebody else that it does not have a Hamiltonian circuit, seems to be extremely difficult thing. So, only way it looks like is the other person with or without your help, looks like all possible collections of edges. And sees whether these selection let us say, all edges of it take n edges where, n is the number of vertices. The graph takes all possible subsets of n edges and check whether they form a Hamiltonian circuit, now this could take a long time.

The time taken to go through all this exhaustive technique is certainly not may, certainly not be polynomial in the input side. So, and up to date nobody knows anything of better, so for at least HC and many of the problems that we look at. It looks like the YES input behaves differently from the no input. When the answer is yes, there is an easy proof by which you can convince the other person that it is the YES input.

The no input seems to behave very differently. For the class NP, all that we require is that when the answer is yes, there should be an easy way to convince somebody else that. In fact, it is a YES input. So, how do you prove that something is in NP, we have seen two examples. But, let us just sort of note this down to prove something is in NP, you need to do two things.

(Refer Slide Time: 20:59)



You have to say the proof to prove that a problem is in NP, you have to do the following things. You have to say, the proof that the prover sends to the verifier and then how does the verifier check. So, these are the two things that you need to send, this proves by the way for only YES inputs. For YES inputs, you must say what is the proof given, any given YES input, what is the proof that the prover sends to the verifier.

And once the verifier receives this and he looks at the input, the verifier should you should say how the verifier checks, how the verifier is convinced by looking at this proof and the input that the input is, in fact YES inputs. And for the Hamiltonian circuit case and for composites, for the problem composites we specified both. In the composites

case for instance, the prover sends the factorization and the verifier just multiplies two of them and checks whether equal to.

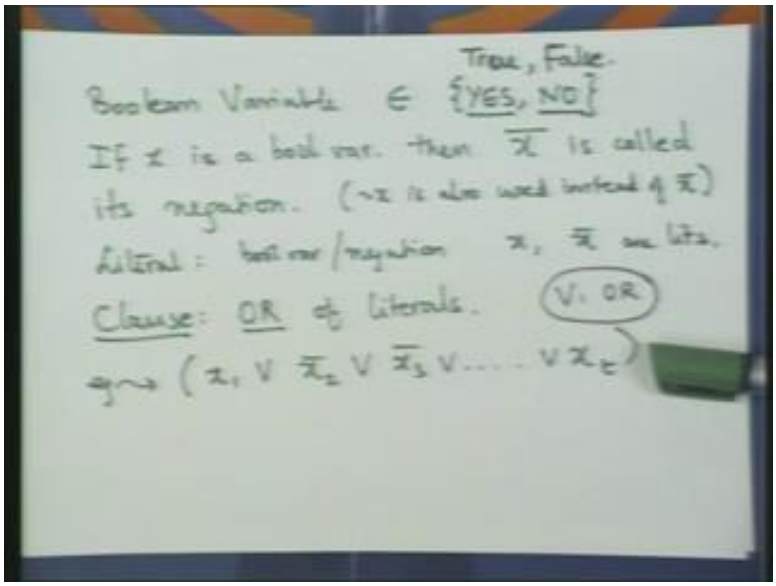
So, we have, in fact now defined a class NP we will see many other examples. In fact, I would encourage you to look at problems you have previously come across, at least the decision problems and try and prove whether, some of these fit into the class NP. So, one importance of this class NP is why I define this class NP at all. Well a large number of problems, which occur in real life fall into this class NP in the following sense.

I mean, if you look at problems in real life they, most of them you do not want problems with a yes no answer, the answer is usually many bits. But, these problems can be transformed into a problem, which is not much easier. In fact, as hard as the original problem where, the answer is one bit. And when one looks at these versions of real life, very often they cannot be that is why this class is very important because, a large number of problems that we will encounter.

And that people have been encountering in real life, they all fall into this class defined yet another problem. But, this is sort of central problem that will work around with. So, this will be a problem in NP and it will be the central problem, you would have seen something like, this in many of your previous courses in computer science. If you have done logic, you would have seen something like this; you would have seen this Boolean circuit etc, etc.

So, many of these concepts that we will deal with and will define should not be new to you anyway, I will define the problem. The problem is this I need to make few definitions, before I get on to the problem.

(Refer Slide Time: 24:40)

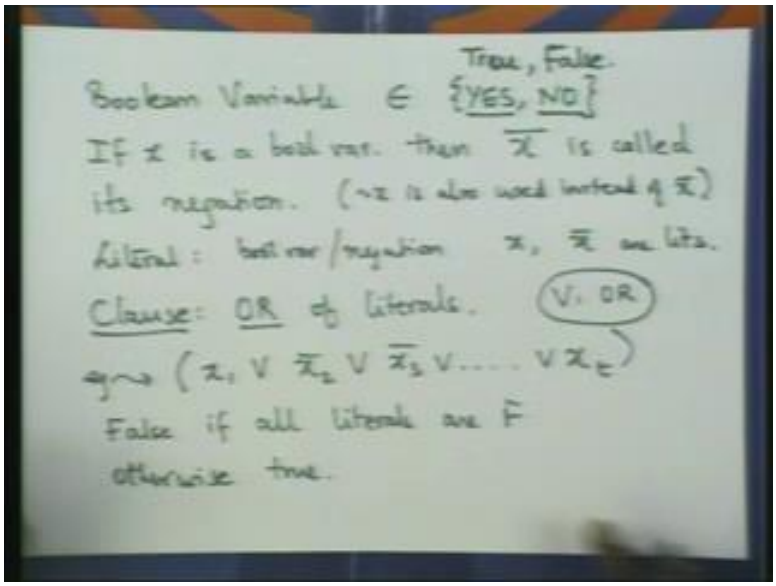


So, a Boolean variable is a variable, which takes value, which takes values in the set YES, NO. It takes two values, YES or NO. Now, if x is a Boolean variable Then x bar is a negation, is called the negation, is called its negation and if x is we would. In fact, may be YES or NO will also sort of alternate between true or false, instead of YES and NO. Essentially there are two values YES NO, true false or 1, 0. And if let us say x is true then x bar is false and if x is false then x bar is true and x bar is called the negation.

Some people also use, x is also used instead of x bar. A literal is either a Boolean variable or negation. So, both x and bar are literals. A clause is an OR that will define what an OR is of literals. So, OR is actually the usual logical OR for example, here is an example a clause x 1 OR x 2 bar OR x 3 bar OR x t. So, this symbol OR. So, clause is just an OR of literals and this clause evaluates to either true or false, depending on values that these variables here.

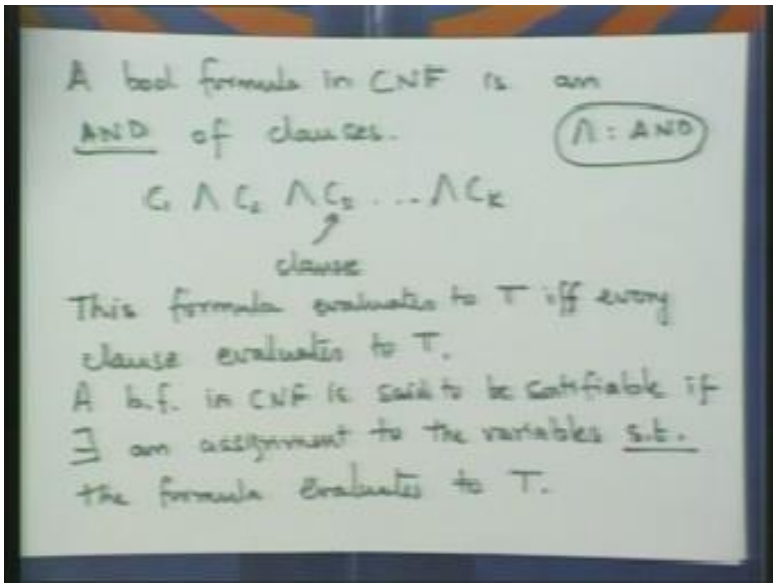
If any one of them evaluates to true, then the clause evaluates to true otherwise, it evaluates to false. For instance if x 1 is true the clause is true. If x 2 bar is true then it is true and. So, on any one of them, evaluates to true it is true. It is false if all of them are false, which means x 1 should be false x 2 bar should be false, which means x 2 should be true x 3 bar should be false x 3 should be true and x 3 should be false. All of them are false then it is false otherwise it is true.

(Refer Slide Time: 28:39)



Let us let me write this. So, it is false if all literals are false otherwise true you could have said it is false if and only, if all literals are false. This is the usual OR, AND is coming up right now.

(Refer Slide Time: 29:19)



So, a Boolean formula in CNF is an AND of clauses. So, I have c 1 AND c 2 AND c 3 AND c k and each of these is a clause, this symbol is AND this is again the usual symbol that we use in logic. So, this is another definition. So, Boolean formula in CNF is an AND of clauses, each of them is a clause c 1 c 2 and c k and c 1 AND c 2 AND c 3, so

on up to c k, this I will call a Boolean formula in CNF, CNF stands for conjunctive normal form.

So, each of these clause remember is an OR of literals. So, this is an AND of OR's and it is called the conjunctive normal form. So, I just have to tell you the logical behavior of AND, which is again the usual thing. This formula is true evaluates to true if, every clause evaluates to true otherwise it evaluates to false. So, maybe I can just write this formula, evaluates to true if and only if every clause evaluates to true.

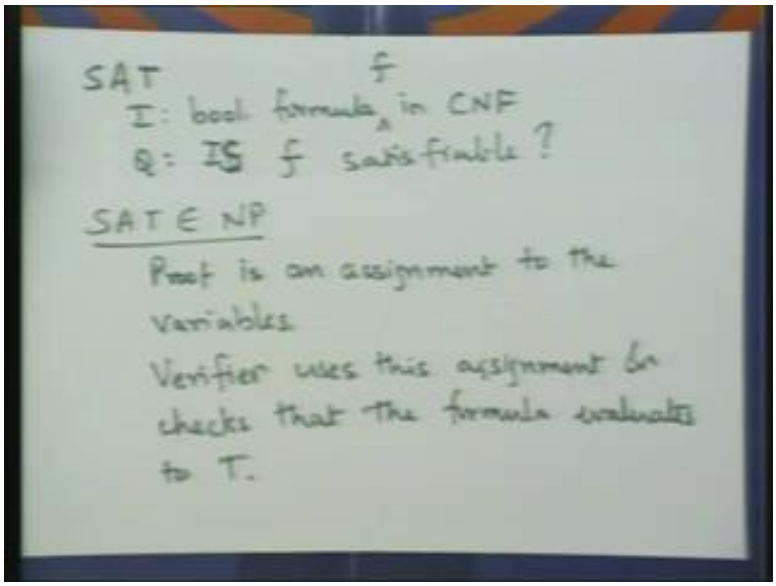
If any one of those clauses is false and this formula evaluates to false. So, what do I mean by evaluates to false. It means, when you give values to these variables then if it is a variable then it evaluates to true. If it is the negation evaluates to false then you check each clause whether, it is value is true or not that you check. A clause evaluates to true if, any one of those literals evaluates to true. And then you look at the formula, you look at the collection of these clauses and this evaluates to true, if all clauses evaluate to true.

Of course these are special Boolean formulae, which we call CNF, which is you have AND's outside and OR's inside. Of course you can form general Boolean formula, which AND's OR's and NOT's any kind of way you like. But, these are the only Boolean formula that we are interested in and one can show, but we will not do it. One can show that every Boolean formula, take any Boolean formula that can be written equivalently in CNF. I need one more definition then we are ready to roll.

So, Boolean formula, the Boolean formula in CNF is said to be satisfiable, if there exists an assignment to the variables, an assignment assigns a value either true or false to each variable. There is an assignment to the variables such that the formula evaluates to true. There is some way of assigning values to these variables. Each of these variables there is a way to assign values either true or false that when you evaluate the Boolean formula, it evaluates to true.

In this case, you call the Boolean formula satisfied, if not you say that it is not satisfied. So, clearly there are Boolean formulae, which are in CNF, which are satisfiable and which are not satisfiable. For instance x and x bar if I take one clause to be just x. And another clause to be x bar and there is no way that you can satisfy this formula whatever, value you get to x the formula will always evaluates to false.

(Refer Slide Time: 35:30)



We are ready to define our next problem that we will study, which SAT not to confuse with exams, SAT stands for satisfiability. So, the input is a Boolean formula in CNF, the question is let me call this Boolean formula something f, is f satisfiable, this is the question. So, now let us observe that SAT is now empty. So, SAT is empty or is this, well it is a decision problem. And suppose the answer is yes, we have to only focus on YES input, so the prover says yes it is satisfied.

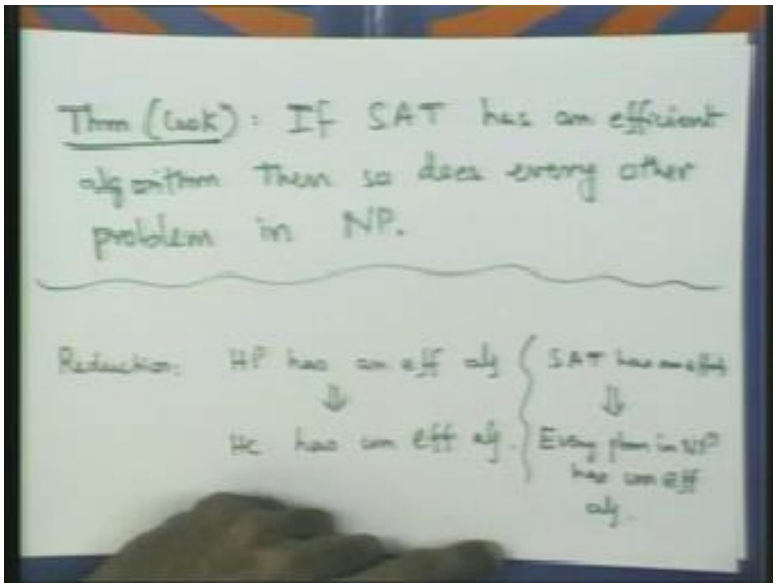
Then, how can the verifier be convinced that a given formula is indeed satisfied. So, what is the prover give the verifier, so that the verifier. So, the verifier can look at this advice that the prover has given him, look at the input formula and you know we completely convince that yes indeed the formula is yet satisfied. Well I hope most of you have answered this question, by now the proof that the prover gives is just the satisfying assignment. So, the proof is an assignment to the variables.

The verifier uses this assignment and checks that the formula evaluates to true. The verifier uses this assignment and he checks the formula evaluates to true. So, given an assignment you should be able to convince yourself that it is easy to check, what the value of your formula is. So, you just plug in the values into for each variables, now check whether each of these clauses are true.

All clauses are true then you are true the formula is satisfiable. So, if it is a YES input that is the proof that the prover can give the verifier using which the verifier can easily

check that the formula is. In fact, true. So, what is the big deal yet another problem in NP that is yet another problem in NP, but this as we shall see in a minute is a very special problem in NP. So, thing that makes SAT very special is the following theorem of cook.

(Refer Slide Time: 39:01)



Cook is Steven Cook, it is a name of a person Steven Cook and there is no relationship to (Refer Time: 39:22). So, this theorem says if SAT has an efficient algorithm then so does every other problem in NP. Let us tire at this time for a minute. So, this is among the most important theorems in computer science. Let us make sure that we understand this. By efficient I mean polynomial time, efficient means polynomial time.

This says that if, there is a polynomial time algorithm for SAT then there is a polynomial time algorithm for every other problem in NP. So, to solve any other problem in NP, it suffices to solve just SAT I mean, remember that NP includes factor, factoring it also includes Hamiltonian cycle, matching etc, etc, etc. Large number of problems that we have seen, so far, in fact is in this class empty and this theorem says that if SAT has an efficient algorithm. So, does every other problem in NP.

We have proved statements of which are sort of similar. We have proved I mean this is the notion of reduction. So, we showed that if HP has an efficient algorithm. We showed this means that HC has an efficient algorithm, we have done this. Now, and this was not easy we were, we took a little bit of time and effort to do something, which is as sort of

well specified at this. Both these problems are very well specified, we knew where we were going.

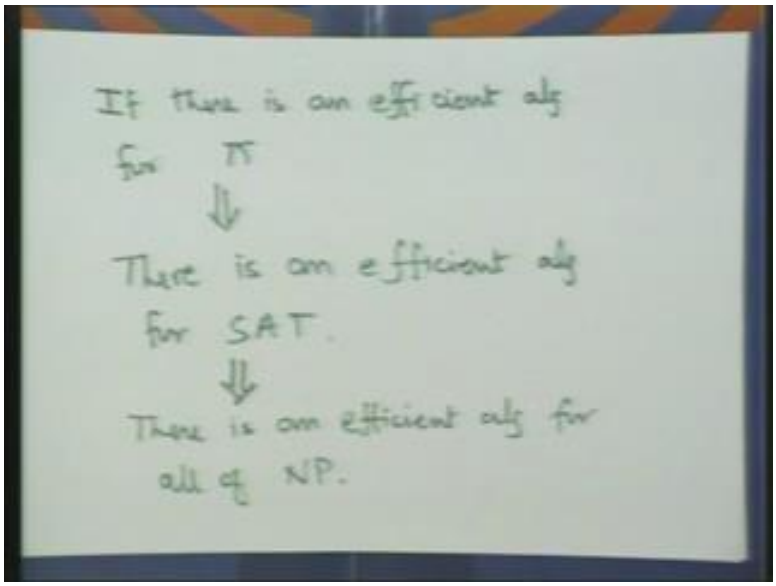
Look at this statement here this says that SAT as an efficient algorithm implies every problem in NP has an efficient algorithm. This is not just one problem somewhere, it is not Hamiltonian cycle, Hamiltonian path what have you which means every problem in NP has efficient algorithm. So, I hope you appreciate the power of the statement and this theorem has changed the shape of computer science. So, there are two ways of reading this statement, one is to say that if you can solve SAT, then you can solve everything else.

So, concentrate all your attention on SAT that is one way to do it. Lot of people have tried and you know, they have not been able to find an efficient solution for SAT. So, the other way of to do this look at the statement, SAT is among the hardest problems. Here is a class NP and SAT is amongst the hardest problems in the sense that you can solve SAT everything else follows. So, it is among the hardest problems.

So, SAT is one of the hardest problems are there other problems in NP, which are as hard of which I can make a statement, which is similar that. If this problem is as hard is the you know, I can solve this problem in NP and I can solve every other problem in NP and I make this statement of other problem. And how easy and difficult is it to prove these statements.

Fortunately with cook's theorem in hand there are. In fact, let me further say there are such problems in NP. And with cook's theorem in hand proving that these problems are among the hardest problems in NP becomes a bit simpler I mean. How does it go about doing, so here is one. So, I take a problem say some problem pi. So, what I do is this.

(Refer Slide Time: 44:23)



If there is an efficient algorithm for pi then there is an efficient algorithm for SAT. Supposing, I can prove something like, now this is something we have been doing and it looks like we are capable of doing such things. We have done it for instead of pi and SAT, we have done it for we have taken HP HC and well matching. We have seen some matching where, we have taken two algorithms, two problems like this. And we have shown if, there is one algorithm for a problem and there is another algorithm.

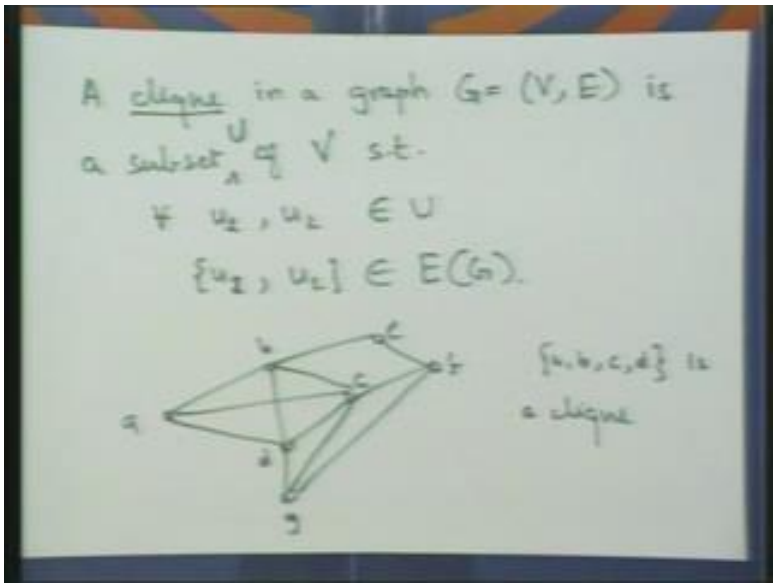
Essentially we use the sub routine for pi and we construct an algorithm for SAT something like this, we can do. But, now let us see what this gives us, now here is the other sort of implication that I would like to use. Cook's theorem says that if there is an efficient algorithm for SAT. There is an efficient algorithm for every problem in it, there is an efficient for all of NP which means, all problems in NP.

So, let us just put these two things together, if there is an efficient algorithm for pi there is one for SAT and there is one for SAT then there is a efficient algorithm for all of NP. This follows from cook and this result we will have to hook up. So, this is up to us to prove. So, to prove that pi is amongst the hardest problems in NP, all I need to do is prove that if there is a efficient algorithm for pi, there is an efficient algorithm for ((Refer Time: 46:36)) and once I do this.

Using this implication, I am down to prove that there is an efficient algorithm for all of NP. So, this would sort of identify pi among the problems in NP. So, let us identify our

goal for the next few lectures, few hours will be to identify more and more problems in NP, which are among the hardest, in the sense that if you solve this then you can solve every other problem in it. So, here is the first one, I need to make a definition.

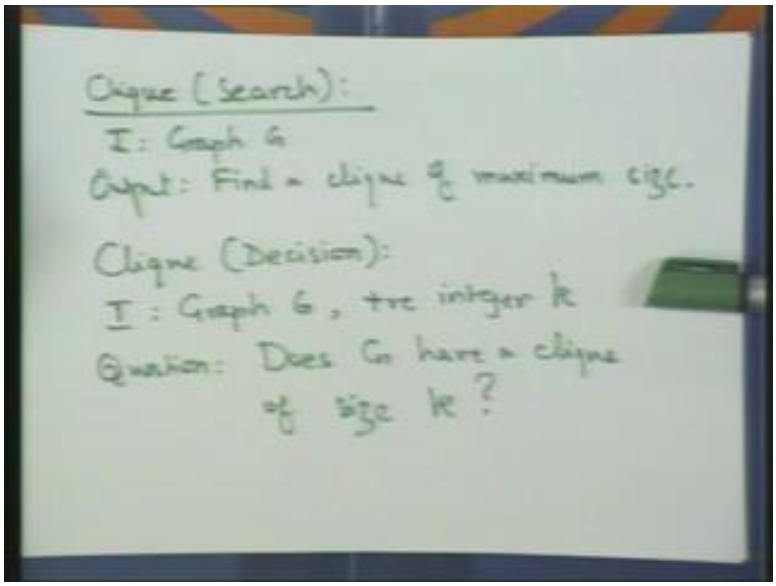
(Refer Slide Time: 47:36)



So, a clique in a graph G is a subset of V such that lets say U of V such that, when I look at any two vertices in U, there must be an edge between them. So, you look at a graph you look at a small subset of the vertex set and in this subset when, I look at the vertices. If between any pair of them there is an edge, in the graph then this I will call a clique. The subset U such that for every u 1, u 2, till U u 1, u 2 must be an edge in G, must be an edge in G which means, if I restrict myself to U, I must get the complete graph.

Every edge must be present or let us take an example. Let us take this example a b c d e f let me add this in this example, a, b, c, d is a clique. So, I look at these four vertices. Every possible edge is present I cannot replace c with e. a, b, d and e is not a clique because e is not adjacent to a and d, if I just take a, b, d that is a clique. If, I take just d, c and g that is also a clique. So, this is a clique of size 3, this is a clique of size 4. It is a subset of the vertex such that if I take any two vertices, it is in edge between any pair of them, problem is this.

(Refer Slide Time: 50:10)



So, this is a search version. So, the input is a graph G, output find a clique of maximum size, find a subset find a largest subset you can. So, that when I look at these set of vertices it forms a clique. So, in this graph for instance, the largest clique that you can find is of size 4, which is a, b, c that is the largest. So, the search version is this remembers we are trying to identify problems in NP this is not even a decision problem. So, you want the vertices in a largest clique.

So, we need to look at, what we will do is we will define a problem, which is very similar to this, which will be a decision problem and I will focus on that. So, the decision version is this clique, input is a graph G and a positive integer k, you see how k creeps in or y creeps in a minute. The question is does G have a clique of size k. So, this is the question that we look at. So, let us focus on these two problems.

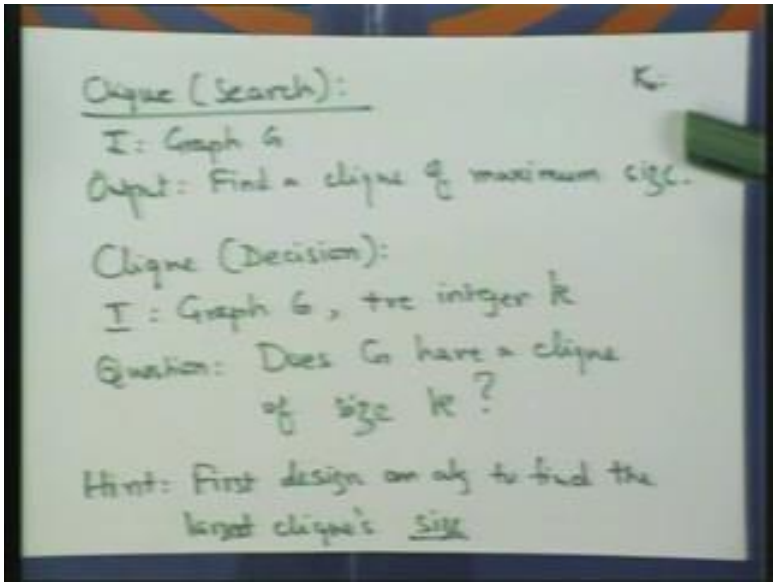
Now, the second problem which is a decision problem, in the sense the question the answer is either yes or no. So, that is fine, now if you can solve the search problem. If there is an algorithm to solve the search problem, clearly there is an algorithm to solve the decision problem. Just find the clique and find the size. If the size is k or larger then you are done the graph does have a clique of size k. On the other hand, suppose you have an algorithm for this.

How do you find, can you solve the search problem well the answer is yes and the construction is very similar to what we have done earlier. So, what you do is first find the

size of the largest clique. You can do a binary search on k, feeding the graph with various values of k. And essentially want to find, the largest value I mean if the graph has an edge when you place two, the answer is yes.

So, find the largest k for which, this is true, this we can do by binary search. You can even go sequentially and do this k equals 1 2 3 4 5 6 stop as soon as the answer is no. The previous one was the largest clique size of the largest clique in the graph, once you found this size.

(Refer Slide Time: 53:57)



Let me write this, first the hint I am going to give you more hint, the hint is first design algorithm to find largest cliques size or size of the largest clique. Once you do this, now you use this to solve this. What you do is look at the vertices one by one, throw away vertex and ask does this graph, what is the size of the largest clique in this graph. If it is k, I mean if it is you know same as that of G. So, let me just backtrack a little bit.

So, first you feed this graph G and find the size of the largest clique, let us say k 0. So, k 0 is size of the largest clique in G. Now, I remove a vertex and ask whether, there is a clique of size k 0 in this graph. If yes, I throw the vertex away, if no I retain the vertex. I go through all vertices and at the end, I would have thrown away every vertex except, k 0 vertices and these will form a clique.

This is very similar to finding the edges in the Hamiltonian cycle, if you did earlier. Only here we throw away vertices, instead of edges. So, I would encourage you to, I strongly encourage you to follow through the steps, which I have said and make sure that you can do this. Given an algorithm that solves the decision version of the clique problem, describe an algorithm that solves the search version of the clique.

So, now, the decision version seems to be as hard as the search version of the clique. And from now on, we will focus on the decision version. Can we solve the decision version or not. In fact, our goal is to show that the decision version is among the hardest problems in NP, which means if there is an efficient algorithm to solve the decision version of this problem, there is an efficient algorithm to solve all problems in it.