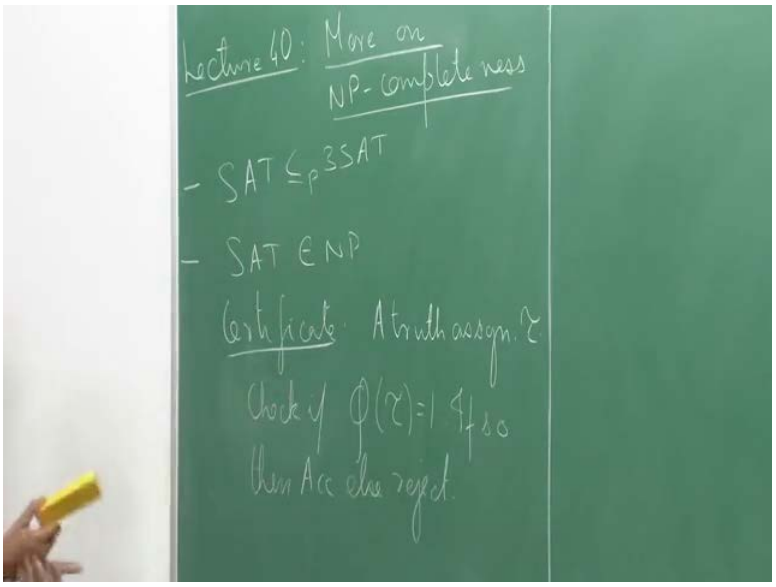
Theory of Computation
Prof. Raghunath Tewari
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

Lecture – 40
More on NP-Completeness

Welcome to the 40th lecture of this course. So, today we are going to discuss more on NP-Completeness. Last time, we defined the problem SAT and the problem 3 SAT. And I mentioned that a historically SAT was the first problem that was proven to be NP complete due to Cook and Levin.

(Refer Slide Time: 00:52)

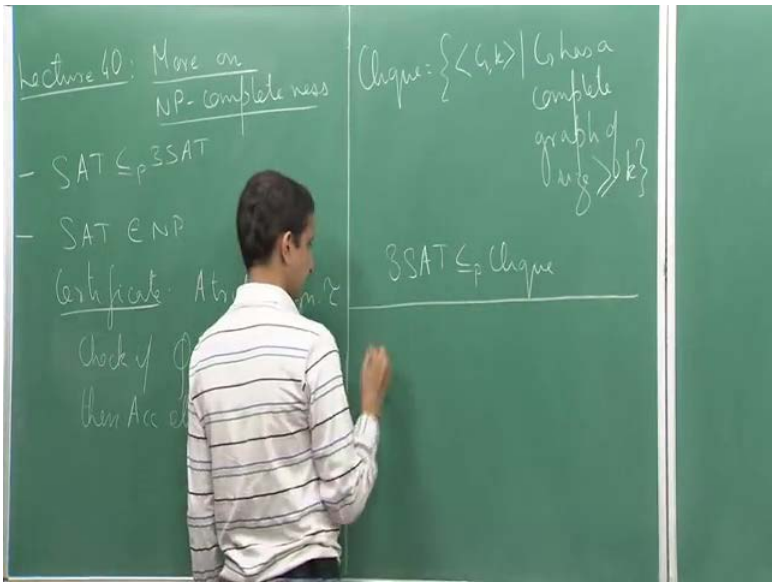


So, this was in the 1970s and thereafter there was a sequence of problems which were shown to be NP complete by the use of reductions. So, three SAT was one of the problems, so it was shown immediately after that SAT reduces to 3 SAT. So, basically you can take a Boolean formula, an arbitrary Boolean formula and convert it to a 3 CNF Boolean formula or a Boolean formula where every clause has exactly 3 literals such that if the first formula is satisfiable, the second is satisfiable. And if the first is not satisfiable then the second is not satisfiable So, this you can try as an exercise; this is not very difficult to show.

So, all this argument gives an NP hardness proof of these problems, but what about containment in NP. So, how do we show for example, that is in NP. So, SAT certainly belongs in NP and if you want to give the certificate based definition, so the certificate is going to be a truth assignment tau. Now you just have to so check if so the Boolean formula is phi so check if phi comma tau is equal to 1; if so then accept, else reject. So, observe that if you are given a Boolean formula phi, and we are given a satisfiable a truth assignment tau, we can very easily plug in the values of the assignment onto the Boolean formula and check whether it evaluates to 1 or 0, so that can be checked in polynomial time that is easy to check.

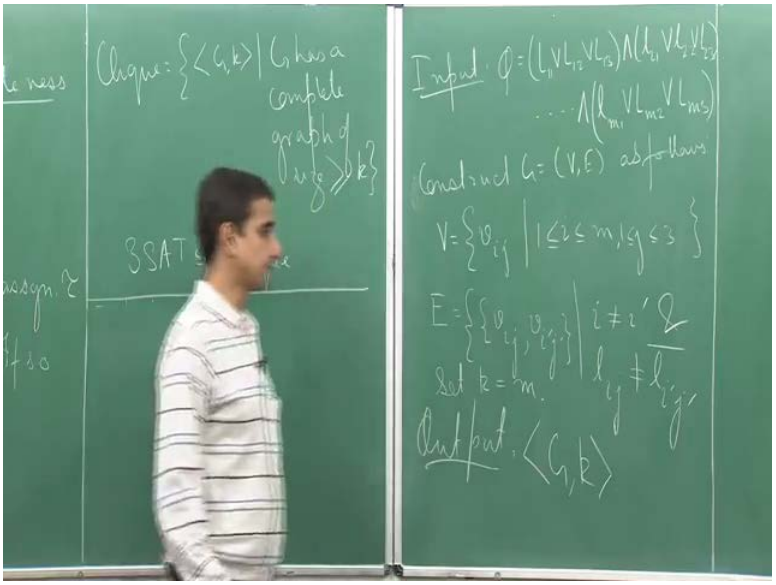
But given just a Boolean formula to check whether it is satisfiable, we do not know whether that can be checked efficiently in polynomial time, whether we can answer given just a Boolean formula whether it is satisfiable or not. So, one approach is to try all possible truth assignments, so if the formula has n variables there will be 2 to the power n many satisfying assignment, but that is not in polynomial time that is exponential time, so that is not a efficient algorithm. So, but this gives an efficient NP algorithm at least, so this approach. So, what we are going to show next is how to use the technique of reduction to show that other problems, other NP hard problems, and subsequently NP complete problems.

(Refer Slide Time: 03:56)



So, recall this problem clique. So, we had seen this problem couple of lectures that. So, clique is the problem of consisting of instances of the form G comma k such that G has a complete graph of size at least k; and we had shown that this problem belongs to NP. So, today, what we will prove is that 3 SAT reduces to clique and because we know that SAT is NP hard this prove that 3 SAT is NP hard and because 3 SAT is NP hard this would prove that clique is NP hard. So, we will look at this complete reduction. So, how does the reduction proceed?

(Refer Slide Time: 05:15)



Suppose as input, we have given a Boolean formula, so let me do it here. So, suppose we have given a Boolean formula phi which is equal to it is a 3 SAT formula, so the formula consist of clauses it is an end of clauses each clause has 3 literals. So, phi is equal to let say L 1 1, or L 1 2 or L 1 3 so these are all literals I am just naming the literals with L and l 2 1 or l 2 2 or L 2 3 and so on we have l m 1 or L m 2 or L m 3. So, the formula has m clauses; each clause has three literals.

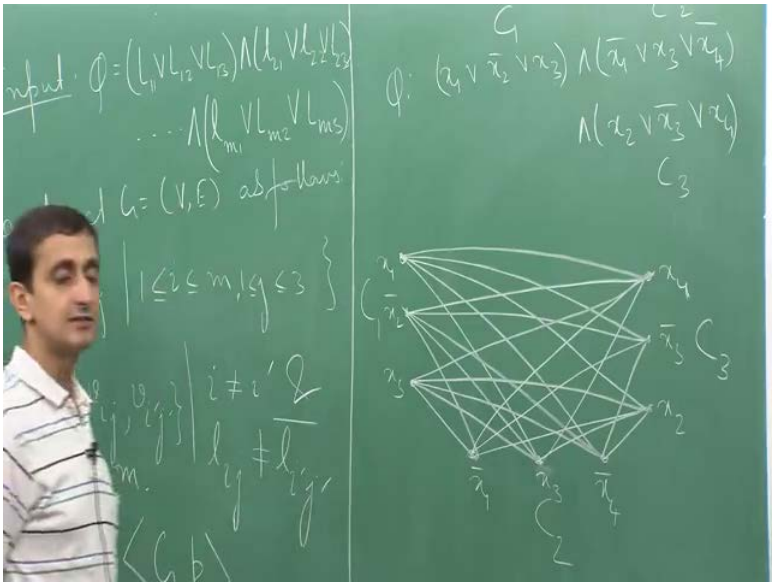
And of course, some of these literals can be the same, for example, I can have a variable x 1 here, and may be again x 1 here or maybe I have for L 1 3 I have x 3 complement. And again in l m 1, I have may be x 3 So, there can be any collection of variables. So, given a formula like this, I have to construct an instance of clique. So, what is an

instance of clique, an instance of clique are a graph and a number k. So, how do I construct the graph? So, I define so construct G equal to V comma E as follows. So, V is defined as so for every literal, I will have a vertex. So, I have a vertex v i j such that l i j is a literal or I mean I can write this as i goes from 1 up to m, and j goes from 1 to 3.

And I define edges as follows. So, I will have undirected edges. So, let say I have an edge between a vertex v i j and a vertex v i prime j prime, so I have an edge between these two vertices, if i is not equal to i prime, so that is these two vertices come from correspond to literals that come from different clauses. And the corresponding l, so l i j is not equal to l i prime j prime complement. So, first of all, they must come from different clauses and one literal must not be the complement of the other literal. For example, if l i j is x 1, and l i prime j prime is x 1 complement, then I will not have an edge. I can of course, have an edge between x 1 and x 1. I can have an edge between x 1 and x 2 complement, but not between x 1 and x 1 complement.

So, to motivate and then I just output, so that is my input, so I will just output this graph G. So, actually I need the output more, so I need to output a graph, and also a number k, So, I need to specify that also. So, I will set k to be equal to this number m that is the number of clauses, and then I output G comma k. So, this is the entire construction. Observe that this construction is clearly in polynomial time, because given a graph sorry given a Boolean formula, construction of this graph is very easy, because I just have to scan through all the literals, every pair of literal, and depending whether it satisfies this these two conditions or not, I will just keep on adding edges that is all.

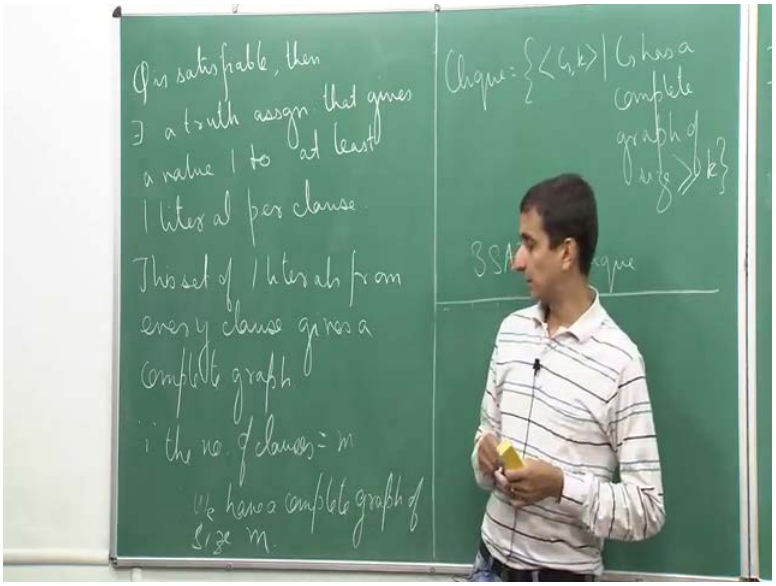
(Refer Slide Time: 10:20)



Now, why is this correct? So, before I talk about the correctness of this approach, let us look at a small example. So, I have example here. So, suppose we have phi which is equal to x 1 or x 2 bar or x 3, and then we have x 1 bar or x 3 or x 4 bar and x 2 or x 3 bar or x 4. What would be the corresponding graph? So, first I have vertex vertices for all these vertices. So, this is for x 1 x 2 and x 3 let say this is my clause c 1, this is my clause c 2, and this is my clause c 3. So, this corresponds to c 1 then I have three vertices which correspond to c 2 and then I have three vertices which correspond to c 3. So, this is let say corresponding to x 1 here, x 2 bar, x 3, x 1 bar, x 3, x 4 bar. And I have x 2, x 3 bar and x 4.

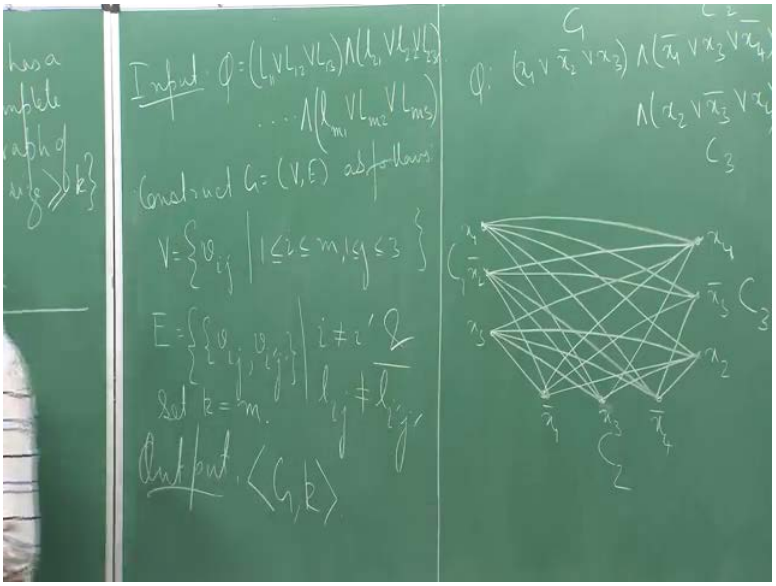
Now how do I add edges of course, I will not have an edge from x 1 to x 1 bar, but to every other vertex I will have an edge. So I have an edge from here to here, here to here, and from x 1 to x 4. From x 2 bar, I will have an edge to x 1 bar, x 3, x 4 bar; I will not have an edge to x 2, I will have to x 3 bar and x 4. Similarly, from x 3, I will have to x 1 bar, I will have an edge to x 3, and x 4 bar, I have an edge to x 2 not to x 3 bar, and to x 4. From this x 1 bar, I have edge to x 2, x 3 bar and x 4. And from x 3, I have an edge to x 2 and x 4; and from x 4 bar, I will have an edge to x 2 and x 3 bar only ok. So, this is the construction of the graph corresponding to the Boolean formula phi.

(Refer Slide Time: 13:42)



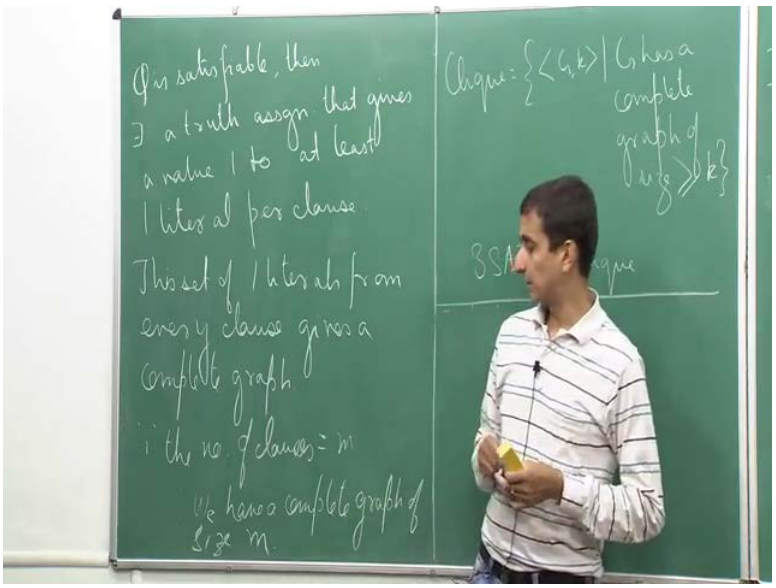
Now, why is this construction correct? So, suppose phi is fiable. So, what happens if phi is satisfiable, let us look at this graph. So, if phi is satisfiable by our construction there exist one literal for every clause that gets a value one. So there is truth assignment there is some truth assignment, which gives the value one to at least one literal in every clause. So, then there exist a truth assignment that gives a value one to at least one literal per clause. So, the claim is that if I pick these one literal from every clause, so this set of one literal from every clause gives a complete graph.

(Refer Slide Time: 15:35)



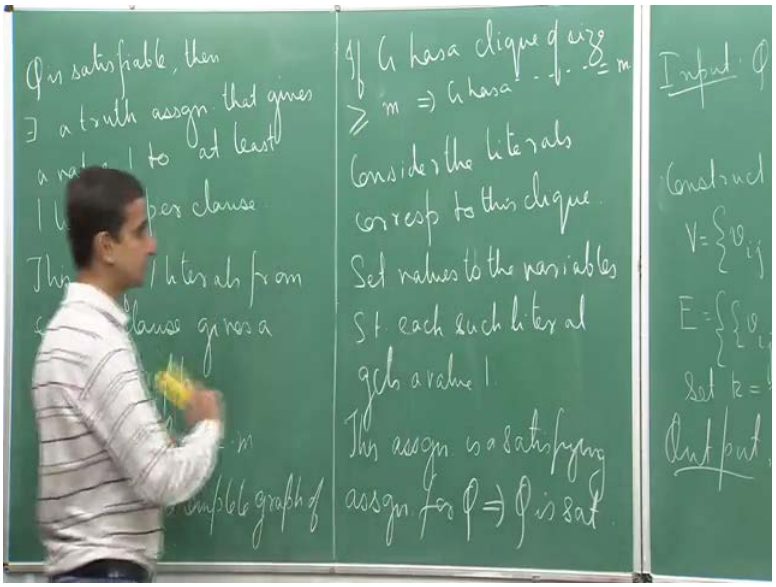
Because observe that if I have tool let say I have a literal L 1 1 here and I have a literal l 2 2 here, which gets a value 1, then clearly it cannot be that this variable is a not of this variable, because then both cannot get the value 1. So, if both gets the value 1, it should not be the case that one is a negation of the other which means that there is an edge between l 1 1 and l 2 2.

(Refer Slide Time: 16:09)



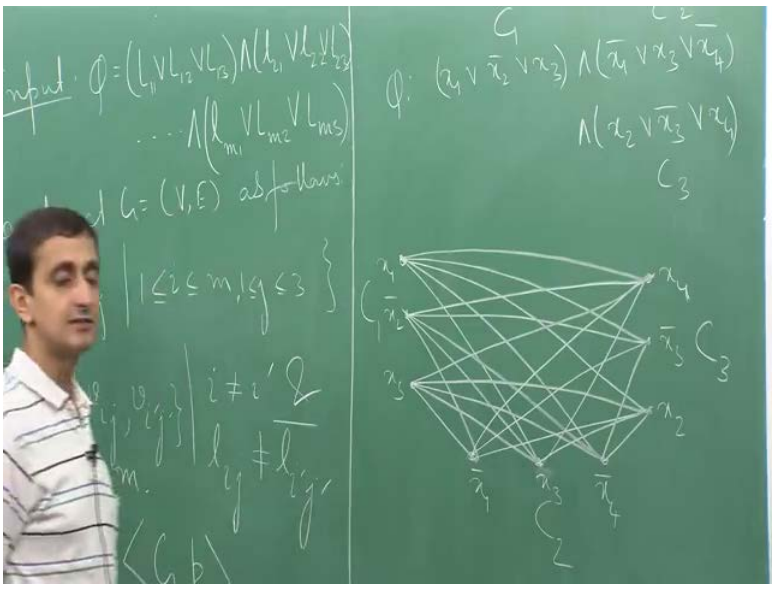
So, how many clauses do we have, so since the number of clauses is to equal to m therefore, we have a complete graph of size m. So, I take this set of m literals one per clause, and if I look at these the vertices corresponding to these literals in my graph, they will constitute a complete graph. Hence, I have a complete graph of size m. So, if phi is satisfiable, I get a complete graph of size m.

(Refer Slide Time: 17:05)



Now the other direction let us look at the graph now. If G has a clique of size greater than or equal to let me call it m, m and k are the same basically because that is how I define this. So, if my graph has a clique of size greater than m first of all observe that it cannot have a clique of size strictly greater than m for example, m plus 1.

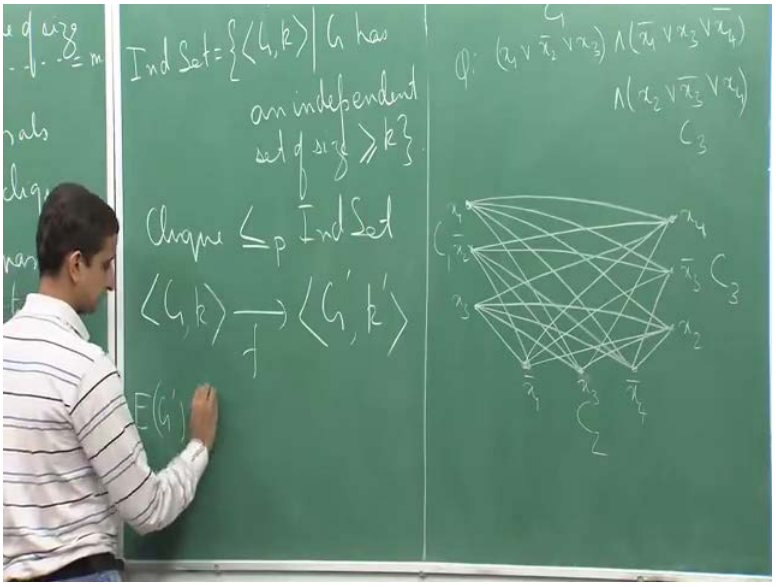
(Refer Slide Time: 17:39)



Because, if you look at the vertices corresponding to only to one clause, there is no edge between them, and the number of clauses is equal to m. So, the maximum size of a clique and only be m if I have m plus one vertex then there are at least two vertices from the same clause by pigeonhole principle which cannot be possible. So, if G has a clique of size greater than m then it implies that G has a clique of size equal to m. Now consider the clique, so consider this clique. So, consider the literals corresponding to this clique So each literal must come from a different clause first of all that is by definition and it also must be the case that because this is a clique no literal is in negation of another literal in this set. So, set values to the variables such that each such literal gets a value 1.

For example, I have literal x 1, and then I will just set x 1 equals to 1. If I have a value let say a x 2 bar then I will set x 2 to be equal to 0, which would mean that x 2 bar is equal to 1, so that is what I mean that set values to variable such that each literal gets a value 1. And this assignment or truth assignment is a satisfying assignment for phi which implies that phi is satisfiable. So, if I start with the graph which has the clique of size at least m then I am able to show that phi is satisfiable. So, this shows both directions. So, therefore, we have proven that if we take the problem 3 SAT then we can reduce 3 SAT to clique which shows that clique is NP hard.

(Refer Slide Time: 20:42)



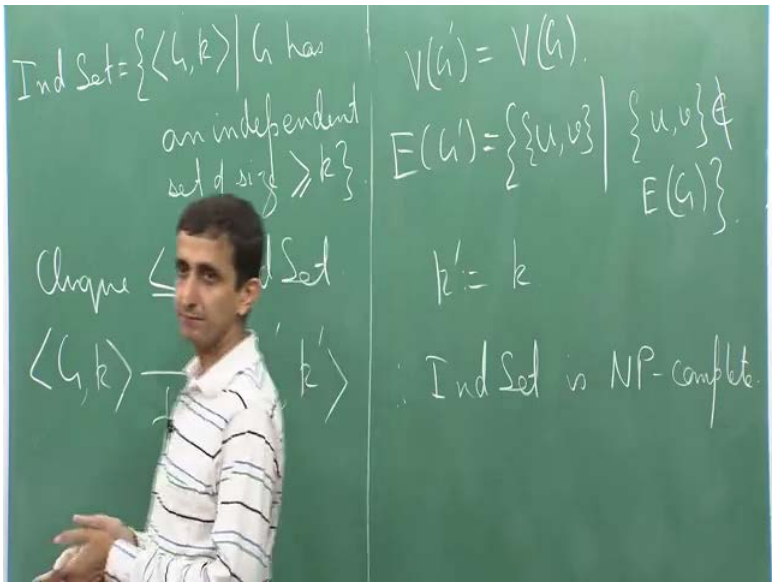
Let us look at another example. So, this is a simpler reduction. So, this is called the independent set problem. So, the independent set problem is defined as follows So it defined as set of pairs of the form G comma k. Such that G is a graph G has an independent set of size at least k. So, what is an independent set, so independent set is basically just the opposite of clique? So, if you have independent set inside a graph, it just means that there is there are no edges between any pair of vertices in that set. So, s is an independent set if for every pair of vertices u v in s, no edge exists, so that is the definition.

For example, if I look at in this graph example, if I look at x 1, x 2 bar and x 3, so if I look at these three vertices they form an independent set of size three because there are no edges between these three vertices. Similarly, if I look at these three vertices they also form an independent set and so on.

So, what we will show is that clique reduces to independent set. So, we take an instance of clique let say G comma k, and we give a function that outputs an instance of independent set, I need to give a different name, so I will call it G prime comma k prime. So, this is very simple this reduction. So, what essentially the reduction does is it computes the complement graph. G prime is the complement graph of G. So, wherever G

has an edge in G prime, you will not have an edge; and if G does not have an edge then in G prime you have an edge. So, let say that I define G prime as follows; so the edge set of G prime as is defined as follows.

(Refer Slide Time: 23:18)



Let me do it here. So first of all the vertex set of G prime is equal to the vertex set of G, so they have the same set of vertices. Now the edge set of G prime is defined as follows. So, u comma v is an edge in G prime, if u v is not an edge in E of G. And this is equality which means that this exactly consist of those pairs. So, every pair which is not an edge in E of G is an edge in E of G prime; and every pair which is an edge in G will not be an edge in G prime, so that is the definition. And set k prime to be just equal to k. Now, the proof of correctness is also this first of all the construction is simple; the construction is clearly in polynomial time.

And the proof of correctness also is not very difficult to argue because look at first of all look at an independent set of size k in G. So, if G has an independent set of size k by definition the same set, sorry if G has a complete graph of size k, by definition the same set will be an independent set in G prime, because there will be no edges between any pair of vertices in that set. And if I take an independent set here, by definition the same set will be a clique here, so that is it follows very simple. And one can also show that

independent set is in NP, because again I will guess a certificate which will be a set of vertices, and I just need to verify whether for every pair of vertices in this set no edge exists. So, therefore, this shows that independent set is NP hard and subsequently NP complete.

So, I will end my discussion here. So, let me give a brief review of what we have seen in this course. So in this course we saw models of computations. So, our goal was to understand computation via different models like what is the power of different computational models, what are the kinds of problems that they can solve, what are the limitations of these computational models in the sense that what are the things that they cannot solve. And by understanding limitations, which we constructed models which had more power which somehow were able to tackle this limitations.

For example, initially we looked at finite automata; and we saw that there are languages which are not computable by finite automata, so we defined context free grammars and pushdown automata. So, they were able to recognize some non regular languages, but then they were again languages which were beyond which were not decidable by a pushdown automata or a context free grammar, so then we moved onto Turing machines and we talked about decidable and undecidable problems.

And in the region of decidable problems, we have this gradation of problems that are defined based on the amount of time that that it takes to solve a problem, and amount of space also. So, we look at various resources; and based on the amount of resources that a class of language uses, we classify these languages. And then we try to study properties between these languages. So, this is basically what has seen in this course and so that is all ok.

Thank you.