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

Lecture – 38
More on the class NP

Welcome to the 38 lecture of this course. Today, we will continue our discussion on complexity theory, and more specifically we will look at the Classes P and NP in more detail and try to understand the connection between them.

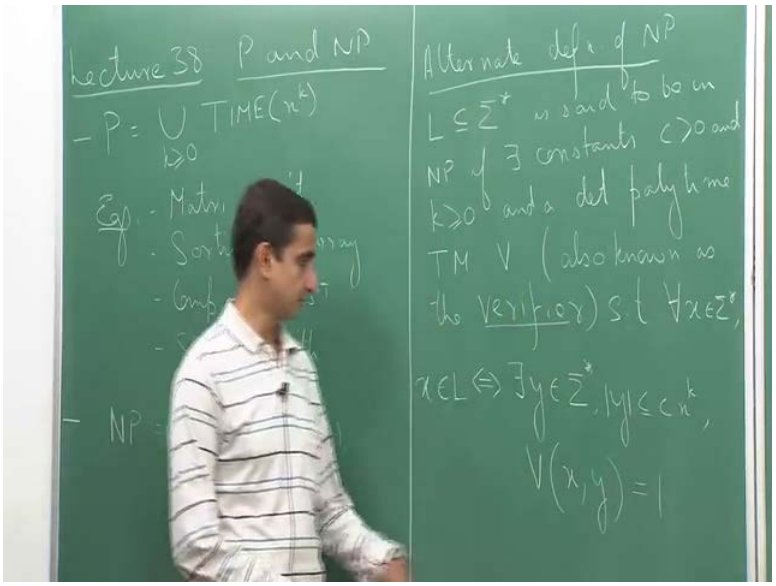
(Refer Slide Time: 00:39)

In our last lecture, we saw the definition of P and NP using Turing machines. So, P was defined as the union over all classes of the form n to the power k. And there are lots of examples or problems in P, for example, matrix multiplication, sorting an array, computing minimum spanning tree of a graph, shortest path and so on. In fact, most algorithms that probably you have seen so far whether in this course or whether in other algorithm courses are polynomial time algorithms or algorithms, which are in P. In other words, we can construct a deterministic Turing machine, which can solve those problems in polynomial number of steps. So, P is this is how it is define.

And we saw the definition of NP as union over n time n raise to the power k, for k greater than or equal to 0. So, what we will see today is we will see first we will look an alternate definition of NP; one that is more in tune with the statement that I made last

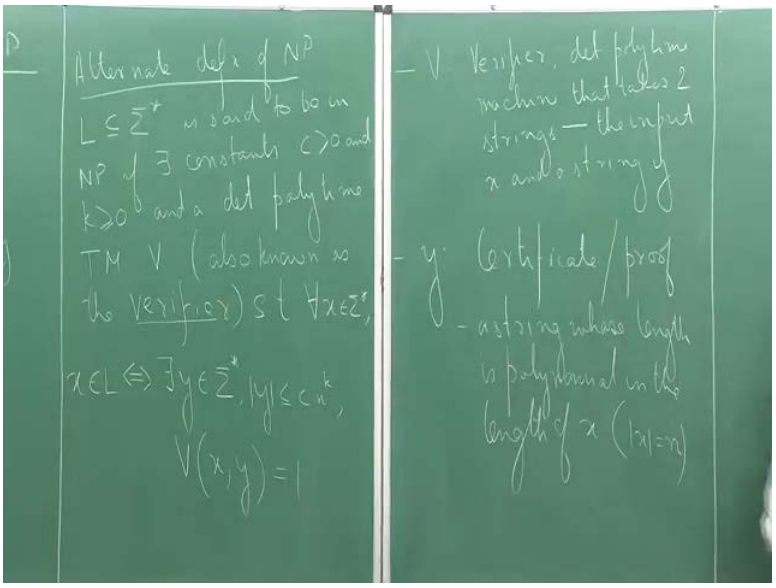
time that NP consist of problems whose solutions are efficiently verifiable. So, P consists of problems whose solutions are efficiently computable. So, you can compute a solution efficiently in the case of in P, it is efficiently verifiable. So, we will give a definition in line with this fact, and then we will look at some examples.

(Refer Slide Time: 02:57)



A definition of NP, so L subset of 0 1 star or you can have any alphabet, I mean instead of 0 1 star may be I will just use sigma here is said to be in NP, if there exists constants c greater than 0 and k greater than or equal to 0. And a deterministic poly time Turing machine V, so this is also known as the verifier. Such that for all x in sigma star, we say that x belongs to L, if and only if there exist y in let say sigma to the power that there exist y in sigma star and length of y is at most c times n to the power k, and the verifier V given x comma y accepts. So, outputs 1 means that it accept.

(Refer Slide Time: 05:40)



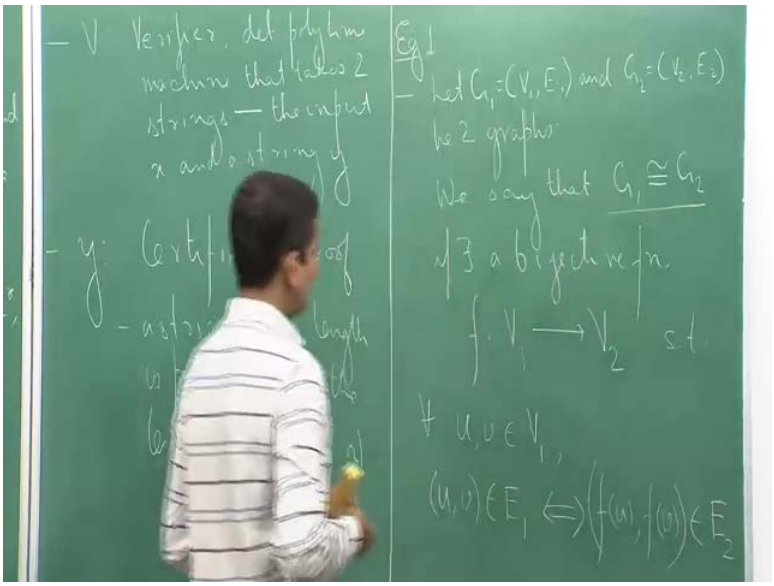
Let us try to understand this definition. So, in this definition, so we have V, which is our which we call as a verifier. So, as I said this is a deterministic poly time machine that takes two strings the input x and a string y. And what is this string y, so y is what we call as the certificate or proof. So, y is what is called a certificate or in some cases we call it as the proof. So, y is a string whose length is polynomial in the length of x that is so we just denote mod x equal to n.

So, what essentially we are saying that we say that x belongs to the language, if there exist some y. So, there is some certificate y, whose length is at most polynomial in the length of x. So, n is nothing but the length of x that is what I have written here. Such that if the verifier is given these two strings x and y, it will accept. On the other hand if x is not in L, then there is no such y; in other words for every y no matter what y you give whose length is at most this one the verifier always rejects that is the verifier will output 0. So, this is the important fact. So, this is what is called the certificate based definition of NP.

So, and the important point is that the machine V is a deterministic polynomial time machine. So, once somebody gives a certificate or once somebody gives a potential solution then the verifier is able to efficiently verify whether the solution is correct or not. If we do not have the solutions, then it might be hard to come up with the solution to

come up with y, but it is easy to verify a given solution, so that is essentially what the definition means.

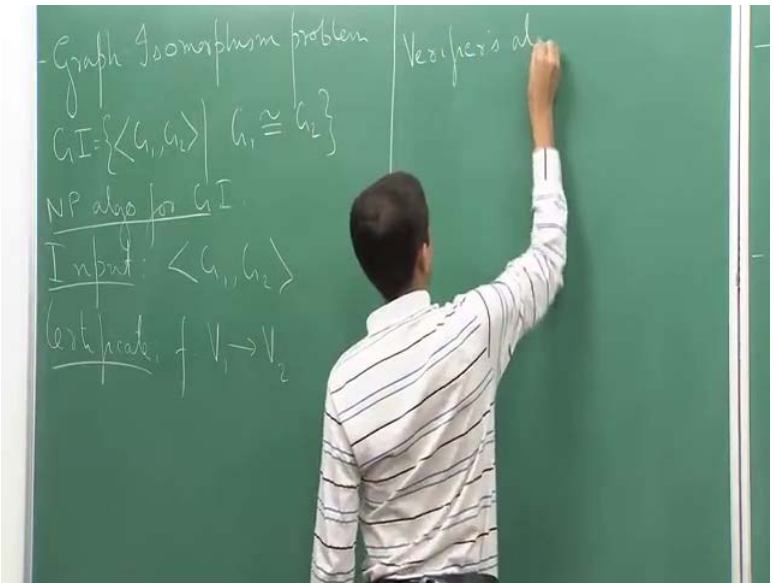
(Refer Slide Time: 08:57)



Let us look at a few examples to understand this better. So, we look at two examples. The first example will be the graph isomorphism problem. Let me define what graph isomorphism is. Let G 1 equal to V 1 comma E 1, and G 2 equal to V 2 comma E 2 be two graphs. Then we say that G 1 is or I will just use a notation, we say that G 1 is isomorphic to G 2 if there exists a bijective function f from the vertex set of G 1 to the vertex set of G 2, such that for all u comma v belonging to G 1 or let me call it V 1. So, for all vertices u comma V belonging to V 1 u, v is an edge in G 1 if and only if f of u comma f of v is an edge on G 2.

So, basically what this definition says is that there is some bijective mapping of the vertex set of the graph G 1 to the vertex set of the graph G 2, there is some way to map this thing. And importantly it is bijective function; that means, it is 1 to 1 and on 2 such that edges are mapped two edges according to this function and non-edges are map to non edges. So, if a pair u v forms an edge in G 1, then the mapped pair f of u comma f of v is an edging G 2. and if u, v does not form an edge in G 1, let us say if u comma v is not an edge then f of u comma f of v also should not be an edge, so this is what it means to say that two graphs are isomorphic.

(Refer Slide Time: 12:20)



Now using this definition, we can define the following problem the graph isomorphism problem. The graph isomorphism problem is defined as follows. So, I call it G I in short. So, it takes as input two graphs in coding of two graphs G 1 and G 2. So, basically the language is defined as the S instances of the language are pairs G 1 comma G 2, such that G 1 is isomorphic to G 2. So, all pairs of graphs such that the first graph is isomorphic to the second graph, so these are the instances; so how can we solve graph isomorphism, so what is the idea? Of course, if I ask you to give a deterministic algorithm, so one deterministic algorithm might be to do the following.

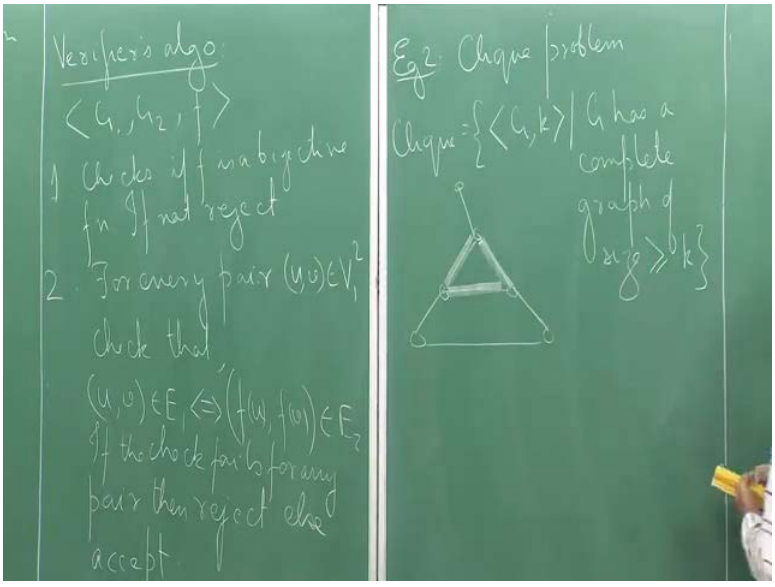
So, you look at the number of vertices in G 1. So, of course, the number of vertices in G 1 and G 2 must be same if they are not you can immediately reject because that means, that there is no isomorphism. But assuming that the number of vertices is the same which is let say n, you can try out all possible functions from a set of n elements to another set of n elements, try out all possible bijective functions.

Of course, there will be exponentially many such functions, and for each function you check whether first of all you verify whether it is bijective or not which, you can easily do and then you verify that whether it satisfies the second property or not. So, whether it is satisfy the property that u v in E 1 implies f of u comma f of v is in E 2 and vice versa, you check this for every pair again that you can do.

But the point is that you have to go through all functions and that is certainly more than polynomial it is exponential. So, deterministically we cannot I mean in fact, so one might ask that other any better algorithm, so as it turns out till date, there is no better algorithm no better algorithm in the sense that there is no polynomial time algorithm there are some better algorithms, but there is no polynomial time algorithm which can solve graph isomorphism deterministically.

Now, let us try to see what we can say non-deterministically. So, what we will do is that we will give a NP algorithm for graph isomorphism and the NP algorithm is not very difficult. So, what we have as input is of course, two graphs G 1 and G 2, and now we have to prove it is in NP. So, to prove it is in NP, let us use the certificate based definition of NP that we saw. So, as our certificate, so we will set our certificate to be a function f from the vertex set of G 1 to the vertex set of G 2. So, it is I mean a function is basically why is the length polynomial you have to ensure that the length of the certificate is polynomial in the length of the input. So, a function is nothing but a collection of tuples, collection of pairs, where the first element maps to the second element and you have n such pairs for where the first element is each element of V 1.

(Refer Slide Time: 16:50)



Now we have to verify that this is indeed a correct thing. So, verifiers' algorithm, observe that what does the verifier take as input. The verifier will take as input the original input and the certificate. The verifier has input G 1, G 2 and the function f. So, check if f is a

bijective function, if not then reject; if f is not bijective, you immediately reject. Now suppose it passes the first check that f is indeed a bijective function what we do is that. For every pair u comma v belonging to V 1 square that is both the vertices are coming from V 1; check if check that u comma v is in E 1, if and only if f of u comma f of v is in E 2.

So, you are on a nested loop basically through overall vertices of V 1. And for every pair u comma v in this loop, you basically check whether if u, v is an edge then the map f of u comma f of v this pair also should be an edging G 2. And how can you compute f of u and f of v, so that you can easily compute from the input that is given because f is given as an input. So, once you know u, v, you can you know what f of u and f of v are. So, you check. So, if this is an edge this should be an edge if this is not an edge then this should not be an edge. So, if the check fails for any pair then reject, else accept. So, if it is passes for all the pairs only then you accept, otherwise you reject.

Now, let us look at the correctness. Let me just briefly say how to argue correctness. So, suppose G 1 and G 2 is isomorphic. So, if G 1 and G 2 are isomorphic then there exist some functions some bijective function. So, there will be some function f which is bijective and which will satisfy this property, and hence the verifier will end of accepting. On the other hand, if G 1 and G 2 are not isomorphic no matter what function you choose, either it will not be a bijective function, or even if it is a bijective function, it will not satisfy the second step, somewhere, in the second step, it would end up with rejecting no matter what function is.

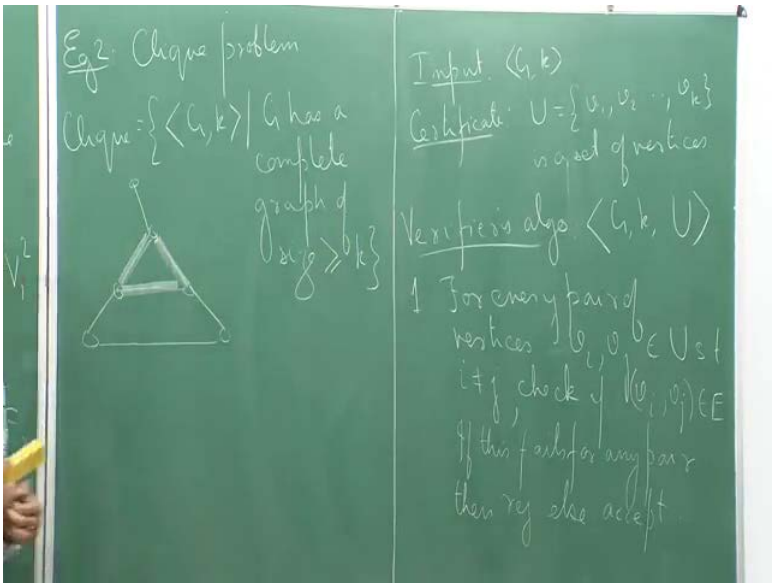
So, for every function that would happen because if yeah. so G 1 and G 2 are not isomorphic. So, therefore, we end up so therefore, we are able to exhibit a proof that graph isomorphism is an NP; otherwise although we do not know whether there is deterministic polynomial time algorithm, but at least there is a nondeterministic polynomial time algorithm for graph isomorphism.

Let us look at another example this is known as the clique problem ok. So, a clique is a complete graph on some vertices. So, that is definition of a clique and we say that a graph has a clique of size k if there are some k vertices in the graph which forms a clique. So, we define clique as the language of pairs of the form G comma k such that G

has a complete graph of size at least k it can be more than k also. So, it is greater than or equal to k.

For example, if I look at a graph which looks like this. So, these are the vertices may be that is something here also. So, this graph has a clique of size 3, for example, if I take this vertex this and so if I take these three vertices, then there is complete graph of size 3 comprising of these three vertices, which is a triangle, but you can easily see there is no clique of size 4. So, clique of size 4 would be a complete graph of size four which is not present in this graph. Now, I want to argue that clique is in NP. So, what we have is our input is of course, say pair of the form G comma k and what we will take as certificate.

(Refer Slide Time: 23:17)



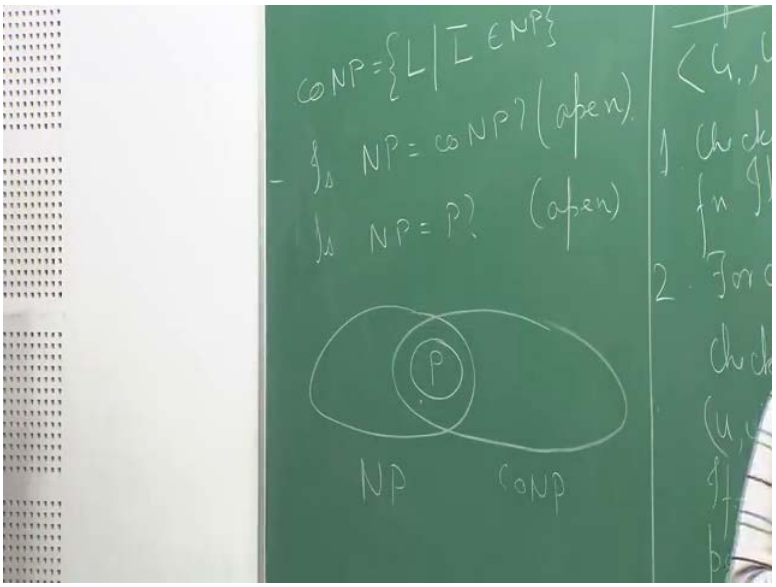
So, we must be careful in choosing the certificate. So, what should this property of the certificate b that if G has a clique of size at least k then there will be some certificate that would allow a verifier to accept. And if G does not have this clique of size at least k then no matter what certificate you choose, the verifier will always reject. So, our certificate is going to be a set of k vertices. So, a set U such that it has vertices let us call them v 1, v 2 up to v k. And what we will basically be checking, so our verifiers' algorithm will be as follows. So, it takes as input the original input which is G comma k, and it also take the set U.

And now what it will check is that do these set of k vertices form a clique or not. The algorithm is going to be for every pair of vertices v i comma v j in U such that i is not

equal to j, they are not the same vertex; check if v i comma v j is an edge or not. So, if this fails for any pair then reject, else accept. So, suppose G has a clique of size k, then there will be some set of k vertices, such that for every pair of vertices in that set there will be an edge between them, which means that it forms a clique.

And if G does not have a clique of size k, then no matter what set of k vertices you choose, I mean it does not matter, when you do this check there will be some pair where this check will break down and we end up rejecting. So, every competition part is essentially rejects. So, there is no such certificate. So, this will prove that the problem clique is in NP as well. So, these are two examples of problems that are in NP.

(Refer Slide Time: 27:18)



Now, we will look at the class of problems whose complement problems are in NP. So, this is what is known as the class co NP. So, co NP is the class of problems L such that L complement belongs to NP. So, what is the relation between NP and co NP? So, is for example, NP equal co NP. So, what can we say about these two. So, observe that, so when we are talking about Turing machines we argued that well if I have a nondeterministic Turing machine, I can convert it into a deterministic Turing machine and such that it accepts the same language.

But if you recall that algorithm to convert a nondeterministic Turing machine into a deterministic Turing machine, essentially involved going over all configurations of the nondeterministic Turing machine. So, we have doing a BFS over the configuration graph.

And the number of configurations can very well be exponential. So, therefore if I want to take configuration graph of an NP machine, and try to go over it in a deterministic manner that is clearly will not give us a polynomial time algorithm, a polynomial time a deterministic algorithm because the number of configuration is exponential, so that is approach cannot be used to get a deterministic polynomial time solution or even to show that it is closed under complement. Because if I have a language which is in NP, all I know is that for s instances there is at least one accepting path and for no instances there are no accepting path.

So, therefore, if I have a language which is in co NP, it only means that if I have a S instance then every path accepts; and if I have a no instance then at least one path rejects that is all. So, there is no way in which I can convert the first type into the second type or vice versa. So, as it turns out this problem is an open question. And of course, so is the question NP verses P. So, this is also an open problem in fact this is one of the famous open problems in computer science and mathematics.

The relation between these 3 classes P, NP and co NP that is known to us as of now is something like this. So, we have the class NP and we have the class co NP. So, of course, they can have some common problems in between. And we have the class P with sets in the intersection of these two classes. So, if I have a problem which is both which is in P then of course, it is both in NP and co NP, but if I have a problem which is in NP and in co NP it we do not know for sure whether it is in P. I mean it can very well lie outside also, I mean we do not know. So, what the precise relations between these three classes are not known, but as of now this is the picture that we have.

So, I will stop here today. So, in our next lecture, we will look at what is known as NP completeness. So, NP completeness is in some sense looking at the set of hardest problems in NP. So, if you look at this picture, we can have many problems that are there in NP. So, some problems are of course in P as well, there are some problems in NP that are in co NP as well possibly, but are they problems which are even harder than them, somewhere some problems which sit outside. So, we have to define this notion of hardness first what does it means to say that a problem is hard, and then we will talk about existence of such problems. So, I will stop here.

Thank you.