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

Lecture -09
Introduction

(Refer Slide Time: 00:13)

So, let me quickly recall what we discussed last time. So, we saw the following, that if we are given the count of the, so we want to decide non reachability. So, just remember that, so if you are given the count of the number of vertices that are reachable from s. We can decide this problem that is path bar using a non-deterministic log space algorithm. And we also saw a very naive approach to coming up with this count which we found out does not work.

Basically if you just circulate through all the vertices and try guessing a path to that vertex and whenever you do find a path if you increment a counter, well along certain computation paths you will get the correct value but there will be computation paths along which you will get a value which is less than the actual value. Precisely for computation paths where you are not able to guess a correct path.

And as a result I mean even if along one computation path if you have an incorrect value of this count. If you look at how this algorithm works you will see that it will give you incorrect

answers. So, basically it will give you false positives. So, instances which are not supposed to be in this language that is there exist an ST path, but it will still output yes whereas it should output No.

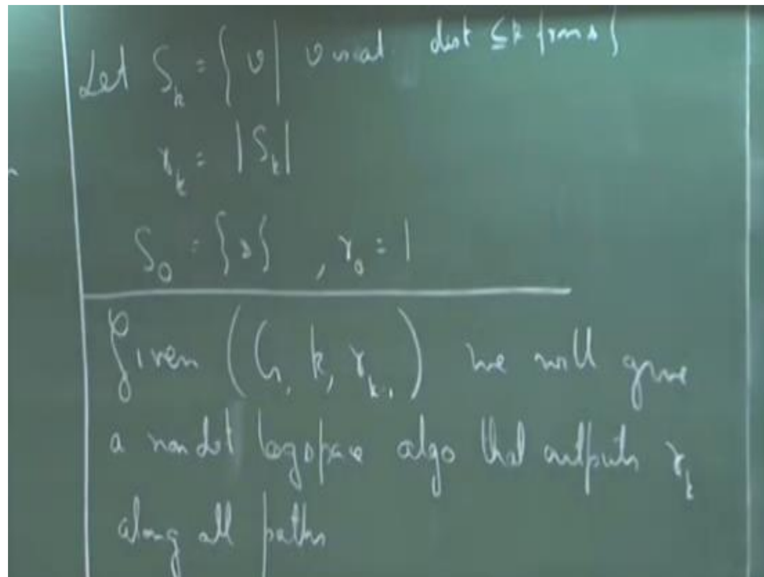
So, what we want, we want basically to get a correct value of this count and we are allowed to use the power of non-determinism. But we have to be careful that the non-deterministic algorithm that we do come up with to get this value count. It should output the same value along all paths. Only then can I club it together with this algorithm and get the correct answer. So, that is very important.

That so we have a some kind of a non-deterministic algorithm, but along all the computation parts it should give me the correct value. So, I think I use the notation r, to denote how many vertices are reachable. So, it should give the same value r, along all paths. So, how do we do that? So, any questions till this point, before we proceed? Please do not hesitate, okay. I mean, if something is not clear, I can go over it again.

So, let us see how we can solve this. So, let us define some notations. So, what we will do is, I will give you the brief strategy what we will do is, we will build an inductive algorithm to come up with this value count. So, in other words we will look at the ball of radius, some k. So, when I say ball of radius k, I look at the set of all vertices that are reachable from s which are at a distance at most k.

And I will count how many such vertices are there and I will use this value to count how many vertices are there which are at a distance at most k + 1, and so on. So, start with 0 and then I will count how many vertices are there at a distance 1, how many vertices are there at a distance 2 and so on. So, it will be an inductive sort of algorithm. So, let me define certain things and it will be more clear.

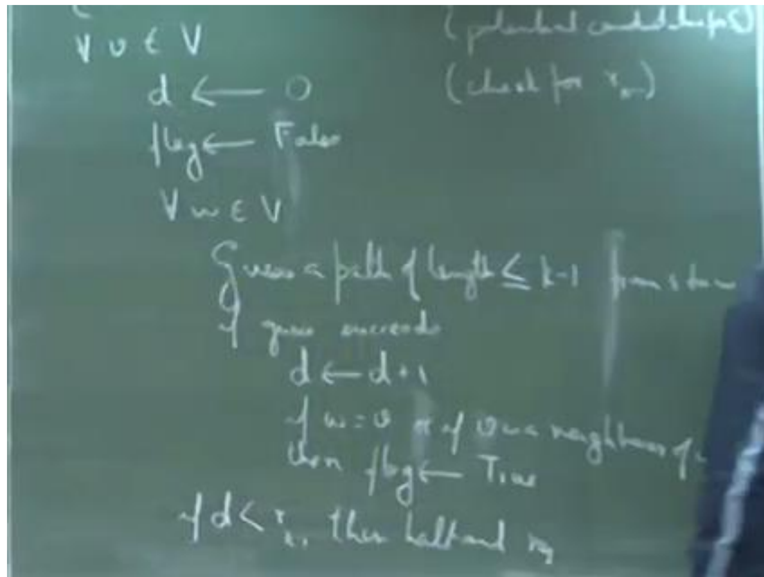
(Refer Slide Time: 05:10)



So, let S k, denote the set of all vertices v in my graph such that v is at a distance at most k from s. So, when I talk about distance, I always mean the shortest path from s. Distance between any two vertices always means the shortest path between those two vertices. And let r k be the cardinality of this set. So, once again observe the following thing that if a vertex is not reachable from s then it is at some infinite design.

We can always use the convention that it is at distance infinity from s. It is not possible to reach that vertex within some finite number of steps. So, what is r or what is S 0, according to this definition, it is just the vertex s itself and therefore r 0 is 1. So, this is what we know, so now what we do is so let me write down the algorithm, start here. Given the following, so given my graph G number k the value r k - 1 we will give a non-deterministic log space algorithm that outputs r k along all paths.

(Refer Slide Time: 07:49)



So, this is what we do. So, we will keep a counter c to store our the tentative value of r k, we initialize that to 0. For all vertices v in my graph, I will now check if v lies in this set s k or not. How do I perform that check, so I will initialize a vertex, I mean another counter d. So, this is basically the, so let me write some comments maybe that will help. So, this is the check for the value r k1, v is are the potential candidates for the set S k and c is again for the getting the value r k. And I will keep a flag initially I will set this to false.

So, now for all vertices w again in the graph, so I have a nested loop. I will guess I will try to do the following. I will guess a path of length at most k - 1 from s to w. Basically I am trying to see that if w lies within the ball of radius k - 1 or not. If guess succeeds that if i do land up at w within k - 1 steps. What we will do is, we will increment the counter d and we will do the following check also.

We will check that if w is equal to v or if b is a neighbor of w. So, this information I can get using my using the input, basically the adjacency matrix of the graph. So, if either of these two conditions are true, so note that when is a vertex inside this ball S k when it is either present inside S k - 1 or it is a neighbor of some vertex in S k - 1. So, vertex is at a distance k from s, if it is at a distance at most k - 1 from s.

Or there is some other vertex which is at a distance at most k - 1 and this vertex is a neighbor of that vertex. So, that is what I am doing, so if w is equal to v or if v is a neighbor of w then set the flag to true and then come up. So, then at the end of going through all these w's, so when this for loop exits, i will check that if d is strictly less than r k - 1 or not. If it is, then I will just halt and reject. And else so if it is equal to r k-1 then I know that this particular v is a correct candidate for my set S k. So, I will increment this counter C. So, else if well still not quite.

(Refer Slide Time: 13:10)

Let me write it here. Else if flag is equal to true then I increment the counter C. And that is it. So, now when this outer loop also terminates, I just output the value of c. Because what can happen is I mean, it is not just sufficient to check whether d is equal to r k - 1 or not. Because let us look at couple of scenarios. So, I have s sitting here and let us say that this is the ball of radius r k -1. And maybe this is the ball of radius k. So, it I mean, it is still possible that v lies somewhere even further away from s.

So, in which case even if there is a computation path which correctly guesses all the correct w's. That is a for which d is equal to r k - 1, this flag value would never be set to true. Because my vertex v neither is it contained inside this ball nor is it a neighbor of any vertex which is contained inside this ball. So, that is when I, my flag will still remain false and I do not increment the counter.

So, this counter only gets incremented for all those vertices which is either equal to a vertex, as I said inside this smaller ball or which lies in this larger ball. So, there is some vertex maybe w and v itself neighbor of w. So, now note that along all computation paths this algorithm will give the correct value of r k. I mean computation paths which does output some value c. Because there are certain computation paths which will not output anything. I mean they will just halt and reject.

So, where an incorrect, I mean where d is smaller than r k - 1. But all those paths which do give a correct value c, I mean do we do give an output c it will give you the correct output. In that case, when v is inside this volume, correct I will not set the flag for that particular iteration of this loop. But there will be some other iteration of this loop when my w is actually equal to v and I am guessing a correct part. So, you are right.

So, maybe that v lies somewhere here, and my w is let us say some other vertex here and I guess a path like this. So, in this case neither is w equal to this v nor is w in neighbor of v. But along in another iteration of this loop, so in that iteration the flag does not get set to true. So, therefore the counter is not incremented well according to this particular w. But there will be some other iteration of this loop when w is equal to v and I am a guessing path from s to v.

For that iteration the flag would correctly get set. So, that is a good point. And the thing to observe is that we have two loops, one sitting inside the other. So, I have been using this terminology but let me explain. So, when I say guess a path of length at most k - 1. So, how do I translate this to a non-deterministic algorithm. So, there are several smaller steps which I kind of overlooked and I gave a very high level algorithm.

So, when I say guess a path of length what it means is that the non-deterministic machine when it is at a vertex, it tries to decide which of its neighbors to pick. So, maybe it has more than one neighbor. Let us assume without loss of generality that it has two neighbors. So, it will decide without I mean, it will try to decide at that vertex which of its two neighbors to pick. And it depending on which neighbor it did pick, it jumps to that neighbor and it carries on this thing.

So, when I say a guess succeeds it means that in that many steps, so suppose I had guessed a length some l which is less than k - 1 and I am running basically another small loop over here of l steps. And at the end of that l steps I am at the vertex w. So, that is what it means here. So, this part is basically some loop of size at most k - 1. That is also another way of writing it. I mean, it does not matter how you write it.

I mean, I just do not want to go into the low level details of the algorithm. So, then it becomes more clumsy in some sense. So, that is why I am using this these kind of statements. But intuitively I mean, if you want to look at the lower level details, the machine level details the way to think of it is that so how do you get this number L also. So, when I am saying that you have a number l how do you get that. So, then again first you have to guess that value L.

So, the only thing that you know is if you look at the input is G, k and r k - 1. So, first what you have to do at this point is you have to guess a number L that is smaller than k minus 1. So, guess along let us say, k - 1 different computation paths. You guess a number 1, 2, 3, so on up to l minus 1 and for each of those guesses you try to guess a path which is of length l which starts at s and terminates at w.

So, that is the complete meaning of this one sentence. Yes, along one computation path you mean. No, so, this is a non-deterministic algorithm so you should not think it so. Basically one way to think of this is that, so what is a guess. So, a guess is basically a pointer. So, one way to think of this is a guess is a coin toss depending on whether you are getting a head or a tail you go into different directions.

Now when do you accept a particular input. So, this is my input that is given to me. So, I will accept a particular input. Well this is not a decision problem, here what I am trying to output a value. So, I will output, so basically as I said I am doing certain coin tosses along my computation paths. So, I start at my initial configuration C start. And depending on what the coin tosses are I have a list of final configurations that I can reach.

So, maybe some of these are deterministic where I am not performing any coin toss. Now what are the values that I am getting at the end of these computation paths, if you look at this algorithm either I am getting a reject value. That is some computation paths get terminated when I am just making this statement or some computation paths get terminated when I am outputting an actual number.

So, let us say that this is a computation path where I am rejecting, maybe this is also rejecting here, I am outputting some value c 1 some value c 2 maybe here I am rejecting some c 3 maybe another c 4 here. So, all I am saying is that all these c i's so whenever I am outputting a value ci all these ci's will be equal to my number r k that I want. And whenever I am doing a reject it just means that it is not outputting any value.

So, now what I can do is that along all these computation paths where I do get the correct value, I rerun this algorithm to get the next value of r that is to get r k + 1. So, what do I have along these paths. I have some value r k, so now I want to get what is my value r k + 1. So, I have my graph G, I can increment this value by 1 and now I want to compute r k + 1. So, now this computation again proceeds from these configurations, these four configurations and it proceeds to compute r k + 1. So, there also certain paths get terminated because of a reject.

But certain paths do correctly compute the value r k + 1 and I proceed in this manner. So, you do not basically rerun this algorithm and you should not think of this algorithm as just working along one computation path. It is some sense parallelly working along all these paths. So, it is kind of a parallel thing happening over here. When I am saying, I reject it means that basically just halts that computation path.

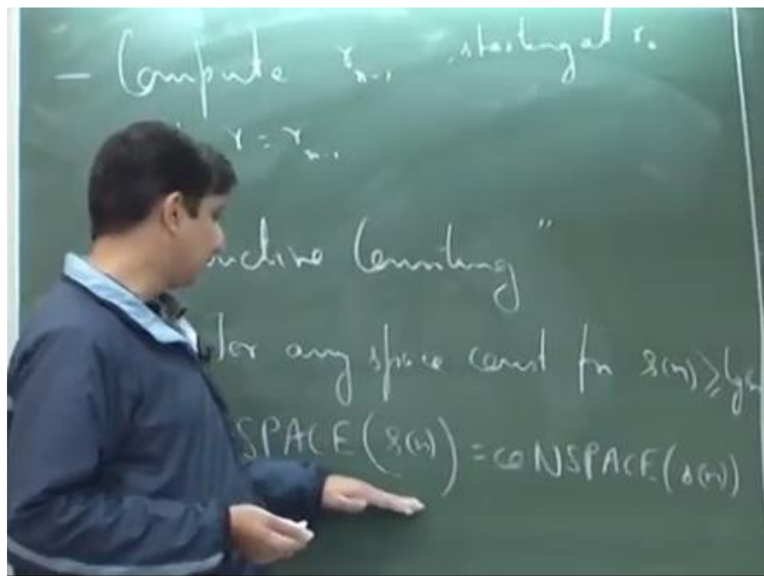
It will not so, when I am saying I am rejecting it means that I am not proceeding any further. So, I will only proceed along which I had a different color chalk, I will only proceed with these computation paths where I am actually getting some value. So, what is the value that I am getting so if you analyze this algorithm, you will find that there is at least one computation path which will give me a value and all paths that do give me a value will give me the correct value of r k.

So, when I am getting a value of r k, then I will again run this entire algorithm for the next value of k that is for k + 1. And now by our assumption, I already have r k, so therefore I can correctly compute r k + 1. So, let us say from here, again I proceed to compute r k + 1 same here and here. And again certain paths reject but those paths which do come up with a number, they give the correct number r k + 1 and so on.

So, how do I use this information now, I mean what we actually want is the number of vertices that are reachable from s. What we are getting here is these values a number of vertices which are at a distance at most k. So, what is the I mean, how do we get this information from these values. Or in simpler terms how far do we need to carry on. What is the maximum value of k that we need to go to? n - 1, n or n - 1, I mean it will be the same I mean there will come a time when it will be the same.

I mean when our k will be the same as r k - 1. That is basically when I stop encountering any new vertices. So, at that point I can just stop and that will be the correct value r and then what I do is that again from those configurations which do give me the correct value are I rerun this algorithm which is a non-deterministic algorithm. And we have already seen its correctness earlier and therefore it will give me the correct answer.

(Refer Slide Time: 27:44)

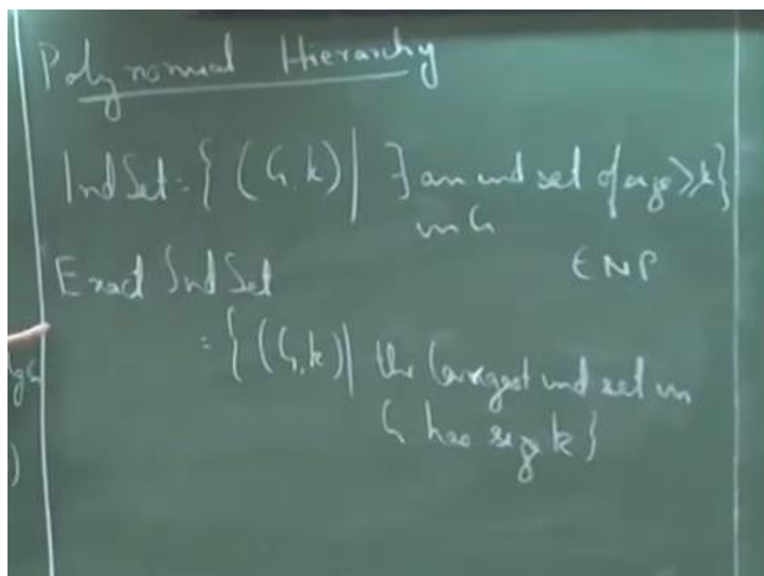


So, what we do is we compute, let us say r n - 1, starting at r 0 and r is nothing but this r n - 1. So, this is a correct non-deterministic algorithm. So, this method is what is known as the technique of inductive counting. So, we are keeping some kind of a count and we are building that count inductively and it was very surprising when this result first came out in 88. Because it was not believed that complement of a non-deterministic class is actually equal to that class itself.

So, it was quite surprising. But it was quite nice for complexity theory. This has lot of applications as well I mean since 1988 this result has found various applications in other complexity results as well. So, let us move on. So, any questions? So, actually I mean if you look at the way we prove this. So, all that we assumed was that we have a graph and we want to decide non reachability in this graph.

So, if you just think this through you will realize that this not only holds for the class NL, it actually holds for any non-deterministic space bounded class which has space greater than NL. So, for any space constructible function S of n, NSPACE S of n is equal to Co NSPACE s of n. Because given a language here or I can always look at its configuration graph and do this entire thing and it will still work. So, let us look at a some different class of problems.

(Refer Slide Time: 30:57)



So, we will try and define today what we mean by the polynomial hierarchy and I will try to motivate why we study these classes and what are their importance. So, let us look at some, in fact let us look at one problem and I will try to motivate why these classes are studied. So, we saw this problem earlier, independent set. So, what was the definition of independent set. So, what is the computational problem G, k such that independent set.

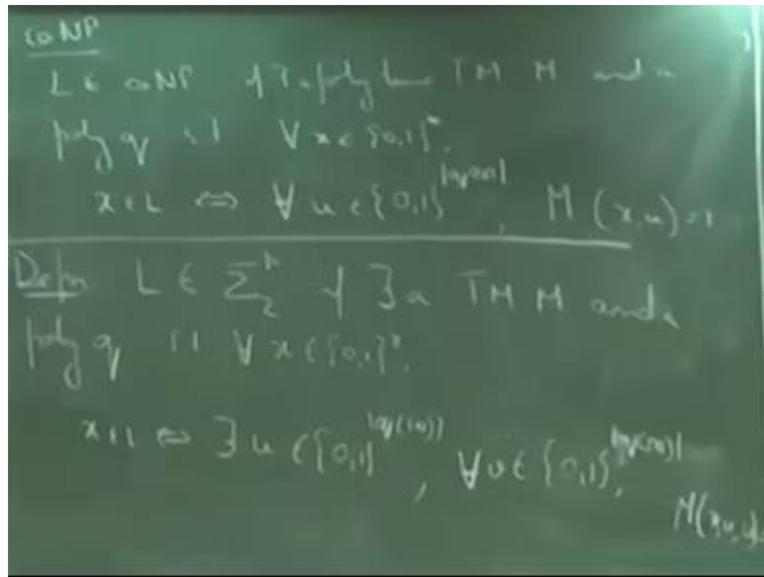
Now what if I look at the following problem. Let us say I want to exactly get what is the largest sized independent set in a graph. So, exact independent set, I will again define it to be set of all tuples G, k such that G has an independent set, the largest independent set in G has size k. So, let us try and see what is the complexity of this problem. So, the first thing is that so we know that this is in NP, in fact we know that this is a complete for the class NP.

What about exact independent set. So, firstly is the definition clear to everybody. So, we are looking at tuples G k such that the largest independent set in G has size exactly k. So, is this class in NP. So, basically you want to check. So, it is easy to get a certificate which gives an independent set of size k, just some k vertices. But now you have to check for size k + 1 that does not exist any independent set of size k + 1.

If the graph has n vertices, you are trying to look at some n choose k + 1 different subsets which can be huge. So, it is well we do not know whether this is in NP or not, but it certainly looks a class which is more difficult than being in NP. I mean a problem which is more difficult than being in NP. What about Co NP, I mean Co NP we saw earlier are again languages which are not contained in NP.

I mean I am so sorry Co NP is basically all languages whose complements are there in NP. And we never looked at a certificate definition of NP. But actually I can give a certificate definition also.

(Refer Slide Time: 34:49)



So, one way to define Co NP is that we say that a language L is in Co NP, if there exists a polynomial time Turing machine M and some polynomial q such that for all x, x belongs to L if and only if for all strings U of length q mod x, M of x comma u is 1. So, this is just a certificate based definition of Co NP and actually you can show equivalence very easily, I mean if a language so the definition we saw earlier was it is the complement of the, I give this definition earlier.

Anyway so you can prove the equivalence very easily I do not think I gave the proof earlier. but so now can we argue about or can we try to see if this set is contained in Co NP or not. So, and Co NP will tell you that for all sets of size k + 1, whether they do not form an independent set or not. But then a Co NP based definition will not tell you whether there exist independent set of size k or not.

So, if you look at this language in some sense this is encompassing the power of both the sets NP and Co NP. So, I need some structure from NP, to guess a independent set of size k and I need some structure from NP to argue that that does not exist any independent set of size k + 1. So, that is the reason which motivates us to look at classes which are in some sense generalization of both these Co NP and NP.

So, I will very soon come to the definition as to what we mean by polynomial hierarchy. But let me make the following definition for the time b. So, we say that a language L is contained in this class sigma 2 p if there exist a turing machine M and a some polynomial q such that for all x we say, x is in L if and only if there exists some certificate of length q x such that for all certificates v again of length q x, M x, u, v is equal to 1.

So, what I am trying to do in this definition is create a class which kind of takes into account both the structure of NP as well as Co NP sets. So, I say that a string x will belong to this language if there exist some string of length q x such that for all strings v of again length q x this deterministic machine m on the input xu and v will output 1. So, the obvious question now is exact independent set in this class or not. What do you think?

So, why is that so what will be my certificate u. So, what would be the certificate, so u would be an set of vertices of size k which forms an independent set and what would be v. So, again v can be anything. So, what we would check is so if v does not form a set of size k + 1, we will just ignore that v. So, it is base so this statement should be true for all v. So, in particular I look at all those strings which do form a set of vertices of size k + 1 and then my polynomial time machine would just check the following.

So, given this input x which is my graph G and number k, it will first check if this first k vertices forms an independent set and these k - 1 vertices do not form an independent set. So, there is some pair there is some edge between some two vertices in this thing. So, these are very easy polynomial time computations. I can just look at the adjacency matrix of the graph and that will tell me.

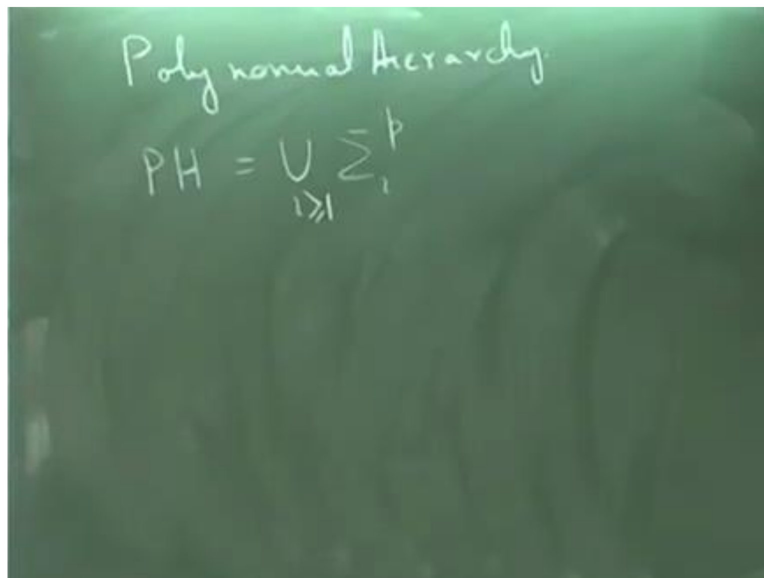
So, again you should not confuse this algorithm to with the following notion that you have to go through all sets of I mean, all strings of length q x. So, basically for all strings which have this length you just assume that M is given these strings x, u and v and then it just performs it com its computation based on what these strings are. And for a string x to be in the language it should have this particular nature.

Sets which are not independent of size k. Some polynomial is sufficient to describe a set of k or k + 1 vertices, let us say k + 1 vertices. So, given my string u, I just first check whether u encodes k vertices or not. I just go through the encoding. If it does then generalize this definition. So, we say that a language L belongs to the class sigma I p, if there exist a Turing machine M and a polynomial q such that x belongs to L if and only if there exist a certificate u1 of length some q x such that for all strings of length all strings u 2 of length again q x and so on.

We alternate between the quantifiers there exist and for all until we reach some quantifier qi, u i is has size again q x such that together with all these strings and x the machine accepts where qi is the quantifier the existential quantifier or the universal quantifier depending on whether I is odd or even respectively. So, basically this should be for all strings, for all x. So, I am just generalizing this definition to instead of looking at just two quantifiers, I am just generalizing it to an arbitrary many I quantifiers.

And we just saw the motivation at least why we need two such quantifiers. So, now it is time to define what the polynomial hierarchy is;

(Refer Slide Time: 46:33)



So, the polynomial hierarchy is just the union of all these classes its denoted by PH so it is the union over all I of sigma i p. So, let us stop here we will carry on from here on Wednesday. It does not matter so what so what happens if we set I to be 1 in this definition. So, sigma 1 p is

nothing but the class NP. So, it is basically the union of all these classes together within. So, just hold on to that question we will look at them next class.