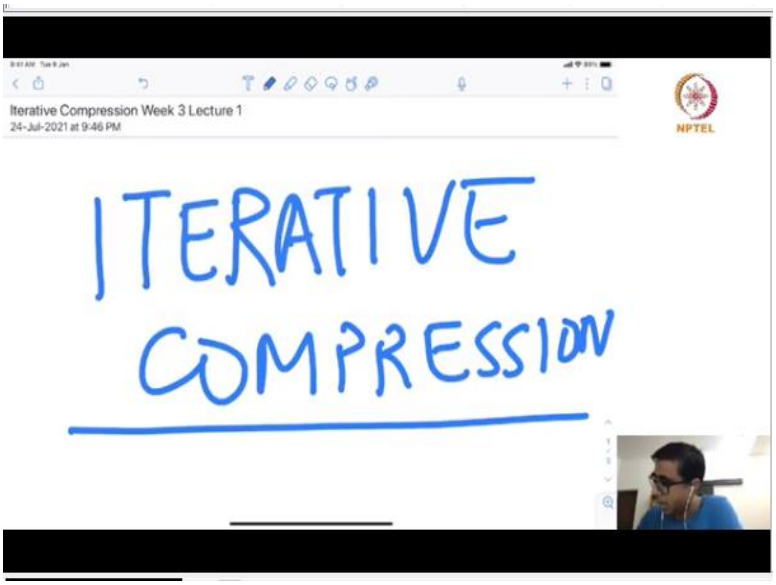
Parameterized Algorithms
Prof. Neeldhara Misra, Saket Saurabh
Department of Computer Science and Engineering
Indian Institute of Technology-Gandhinagar

Lecture-12
Iterative Compression I: Setting Up the Method

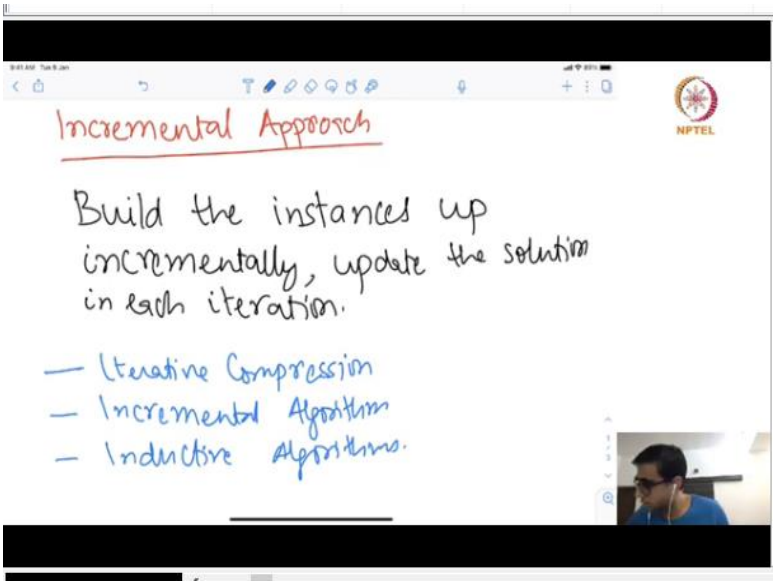
Welcome to all of you to week 3. In this week we will learn another new technique to design parameterized algorithms and technique which we will be learning this week is called iterative compression.

(Refer Slide Time: 00:35)



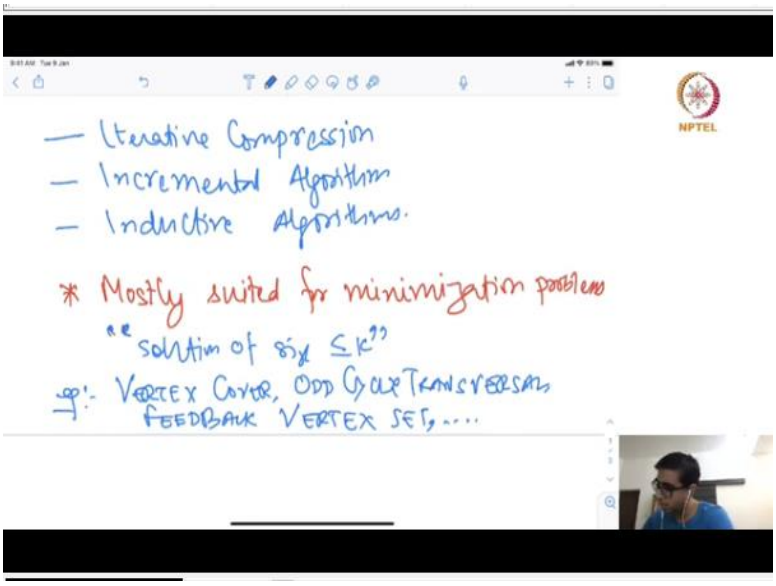
So, the technique which we will learn in this week is called iterative compression and the technique as the name suggests, I think if I had to write a sentence or a line to explain the technique, I would just say that a following.

(Refer Slide Time: 01:18)



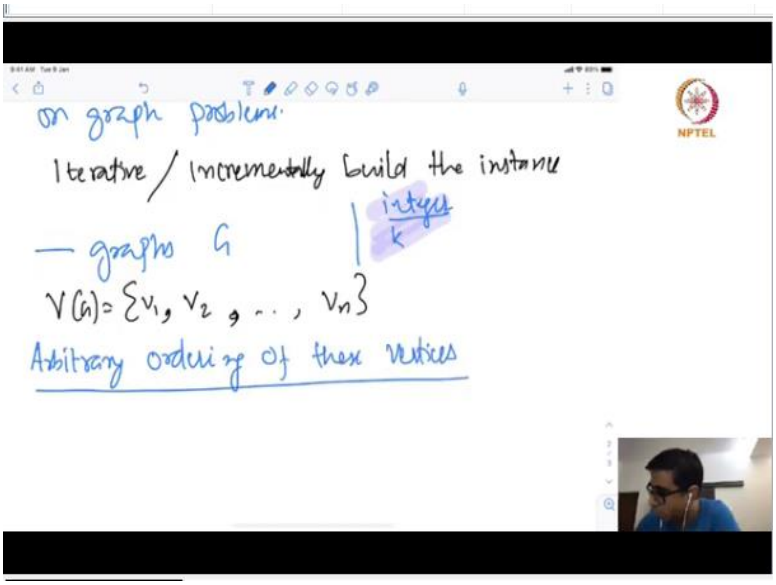
This is basically an incremental approach; all these words will become clear as we go along in the lecture. So, what we do is that it builds the instances up incrementally and updates the solution in each iteration. So, I think name wise as it is called iterative compression, incremental algorithm or we could have told inductive algorithms too, but let us look at this approach what is called build the instances up incrementally?

(Refer Slide Time: 03:00)



And so, this paradigm is mostly suited for minimization problems. So, what are minimization problems? Basically, where you say well gets solution of size at most k. For in our language, this is what minimization problems are. For example, vertex cover is a minimization problem, odd cycle transversal is a minimization problem, feedback vertex set is a minimization problem and so on and so on so forth. So, we will see lots of examples of this algorithm.

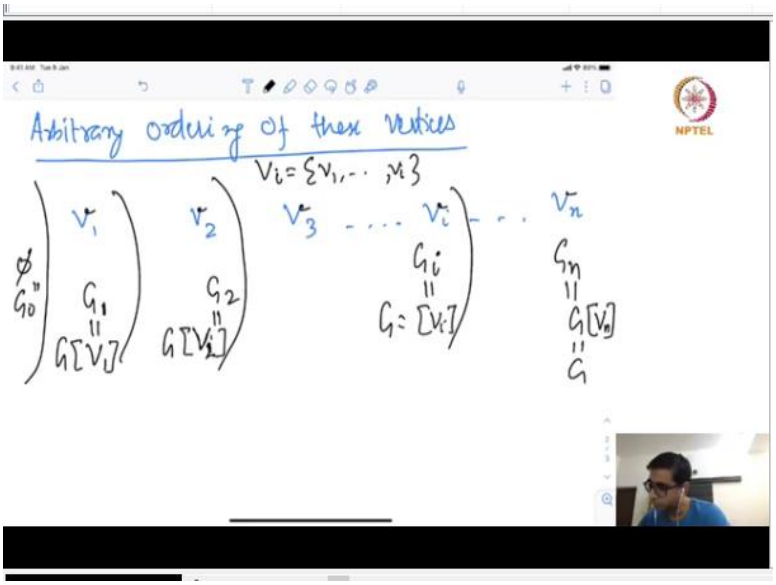
(Refer Slide Time: 04:13)



So first thing first all. So, I am going to illustrate this technique or the some, basic tools of this technique illustrate this technique in this lecture on graph problems. So, the first word we learned iterative or incrementally build the instances. So, instance to a graph problem is what? Instance to a graph problem is going to be, of course, graphs, say G. And of course, if there is a problem, we will have an integer k.

But for now let us not focus on this, but let us see, how could we build the graph G incrementally or iteratively? Very simple; assume that vertex set of G consists of following vertices V 1 to V n, n vertices. So, let us write down an arbitrary ordering of these vertices. What is this arbitrary ordering? I am going to write it for you.

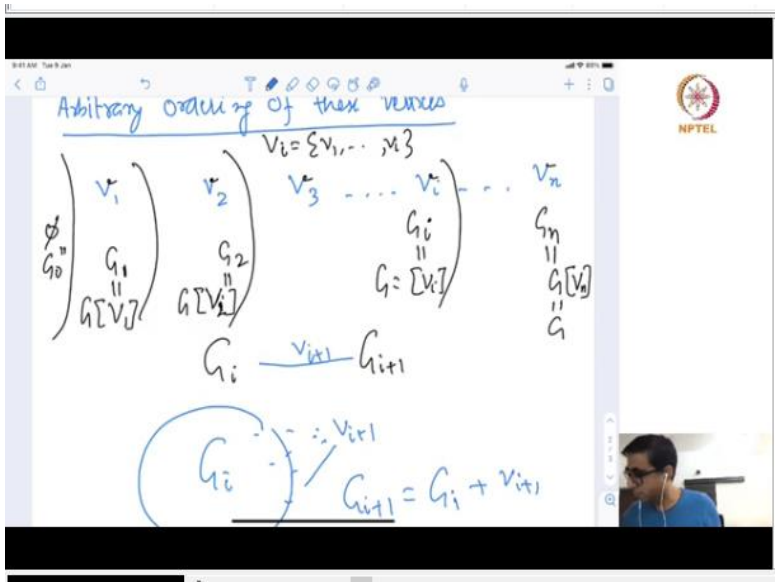
(Refer Slide Time: 06:15)



V 1, V 2, V 3, dot, dot, dot V n V i. And what is my graph? My graph is very simple. I am going to call this graph, G 1. I am going to call this graph G 2. But what is G 2? G 2 is nothing but graph in, so let us call the set V i is like, first i vertices in this ordering. So, what is G i is nothing but graph induced on V 1, V i, V 2, which basically means say V 2. Let us just do small v, small v. They are vertices and v just sets.

And what is G 1 graph induced on V 1, which is just 1 vertex. So, of course, then we also have a notion of empty graph because there is nothing that is like G 0. And what is this? It is G i is what graph induced in V i**.** And what is the last graph? It is G n, which is same as G**,** because this is induced on which includes every vertex, so this is equal to same as G n. Now, notice, what is the difference between, let us try to understand what is G i, and G i + 1, the difference is basically just 1 vertex G i + 1.

(Refer Slide Time: 08:10)

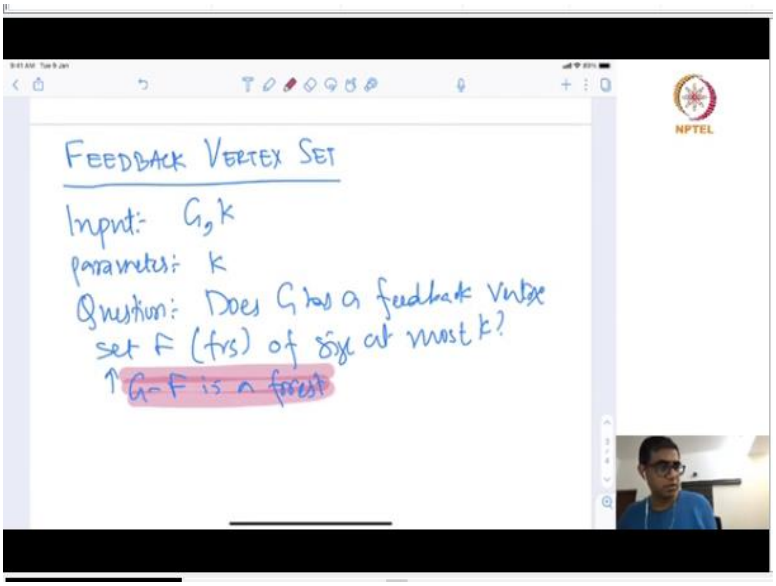


So, basically, what happens is that, here is your G i and you pick up another vertex V i + 1 and you just want to include V i + 1 into G i. So, you just include, so, G i + 1 is nothing but G i + Vi + 1. So, this is how we can incrementally build our graph, there was nothing specific about, vertices, we could have also taken edges, E 1 to E m, m edges and build our graph by adding 1 edge at a time.

So, at G i step you would have a graph on i edges. If i + 1 step, it up taken added a i + 1edge. So, this is how you can build your graph incrementally. So and there is nothing specific about which order you know just write down any particular ordering and this. So, now let us try to see what statements can be made. So, this is how, this is why this? So, what we are going to

do to solve our favourite problem, say for example, let us say, we have been talking about a vertex cover all the time.

(Refer Slide Time: 09:28)

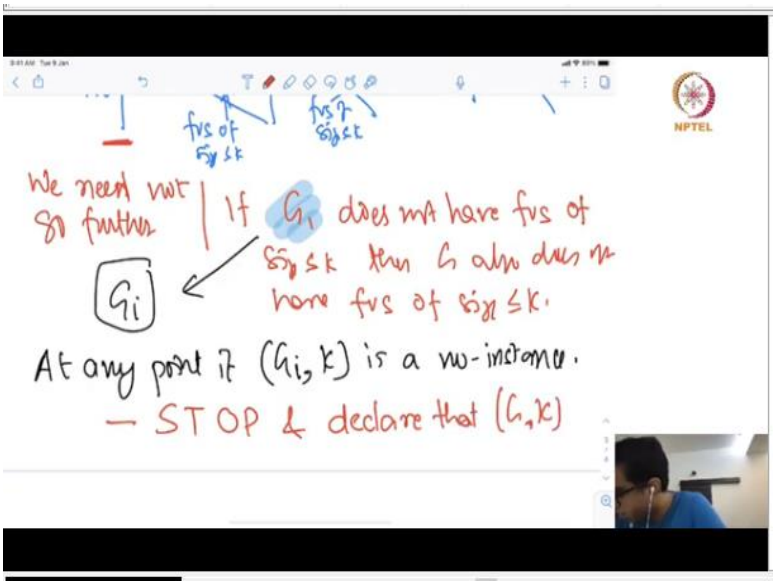


And we will again talk about vertex cover in a while. But let us for just to illustrate this, let us take an example of feedback vertex problem, which you will have seen an algorithm in the last lecture or in the last week when we were doing branching. So, what do you feedback vertex set? Input G, k. My parameter is k. And my question is the G has a feedback vertex set F I will also denote F of size at most k and what are your feedback vertex set?

Basically, G - F is a forest. So, basically I want to delete minimum number of vertices says that what remains is a forest. And in the vertex cover if you recall what was the promise that you want to delete minimum number of vertices such that what is left is an independent set. So, these are the graph classes, like these are the problems where we want to delete some k vertices.

So, that graph belongs to some family F**,** but besides that, let us just focus on one particular problem which is feedback vertex set that if we want to delete k, do you want to check whether there exists k vertices whose recent gives me a forest. So, now I want to understand, so let us see.

(Refer Slide Time: 11:36)



Now so I am going to ask going to solve my problem as follows. Solve the problem as follows or other let us say, we are going to solve the problem incrementally or iteratively whichever word you like or inductively, whichever word you like. So, remember, what is G 1, k is like. So, we have just to repeat this given a graph G we have fixed an arbitrary ordering V 1 to V n and G i is nothing but graph induced on V 1, V 2, some first i vertices, which is nothing but V.

That is it just to once repeating is not bad. So, now, I am just going to ask G 1 k or ask myself, I want to solve G 1, k. So, like I have given this small little graph and k and I asked does there exist a feedback vertex set. So, this is we are going to solve our main feedback vertex set problem we are going to solve different set of problems like first you start one small problem, then another small problem and finally, by taking help from each of this, we will be able to solve the problem on the whole instance.

So, now I asked some G 1, k. There are 2 things. There are there could be 2 answers. One could say no or it says hey, look, here is your feedback vertex set of size at most k. So, I know that this small g 1, k is a good for me, I can solve this then I said fine. Then let me try G 2, k. Again, it can say no, or it can produce me a set S 2, which is FVS of size at most.

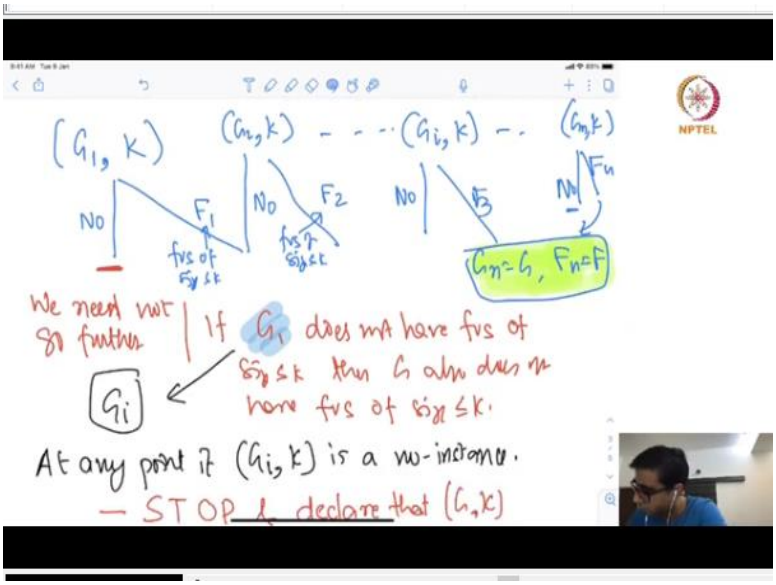
Let us use the word letter F, because this is what we want to say. So, we keep on going asking this. But let us just look at first itself. What can I say if it says that G 1, k is a new instance? The moment I get G 1, k as a no instance they did not further why? If G 1 does not

have feedback vertex set of size at most k then G also does not have a feedback vertex set of size k does not have maybe I should write cleanly does not have FVS of size at most k.

Then G also does not have FVS of size at most k. So, even little graph I cannot delete k vertices because what is feedback vertex set? Feedback vertex set is about deleting vertices So, that you can hit all cycles, you can obstruct all cycle, with k vertices if I cannot even obstruct cycles of an induced graph, because look at any solution, if you look at the intersection to that induced graph, that also forms a solution of size at most k.

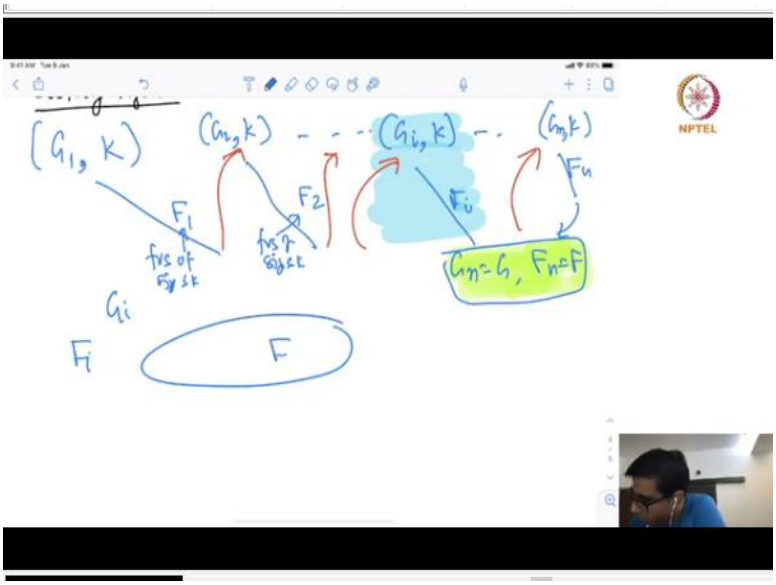
So, if I cannot even intersect all the cycles of a small graph, I cannot hope to obstruct cycles of the whole graph. So, notice and there is nothing about G 1 at any stage, if**,** I could replace this with G i. So, at any stage we get a no instance, if we get that, so, at any point if G I, k is a no instance what will we do? Stop and declare that G, k does not have G, k is also no instance. So, this is what we can. So, this is one assertion**.** This is one assertion we can make.

(Refer Slide Time: 17:37)



So, at any stage, so let us just for a moment that we succeed in each step. So, let us write, so no, then we get F 3 dot, dot, dot. So, at the last, either we get F n or we get no and what is this? So, G n = G and F n = F is your desired solution. This is your desired solution. So, this is perfectly fine. So, now let us try to understand, let us cut this, let us say looking again.

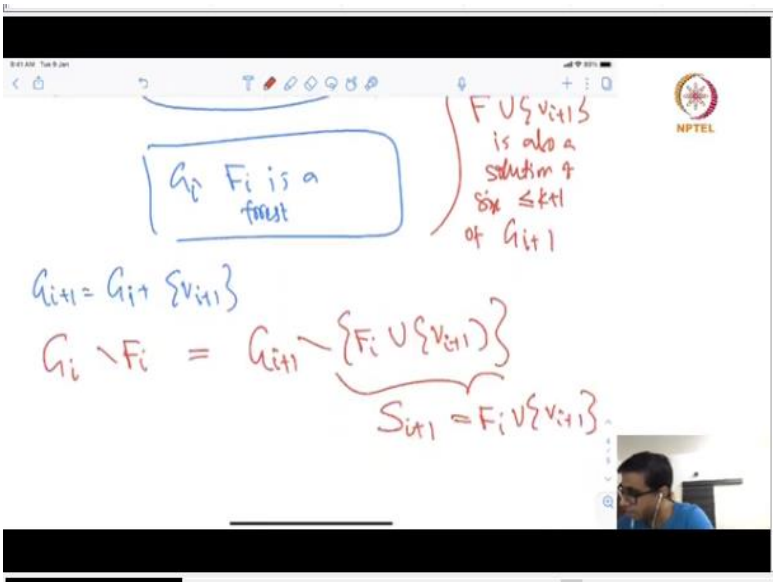
(Refer Slide Time: 18:58)



You may ask what gain we get fine. I mean, you are trying to solve your problem incrementally, like**,** fine. So, rather than solving one problem we have made any smaller problem and you are trying to solve, but ultimately that I mean, what extra information did you get if you succeeded in each of this step. Well, let us see what happens. So, we succeeded.

So, now we in the succeed branch, no no's, so now I am let us see what we can feed. No no's, no no's, no no's. Now notice what happens, what is the property? So, let us look at F i so this is what is the property. So, let us just focus on this part of the instance, what is the property of F i?

(Refer Slide Time: 20:40)

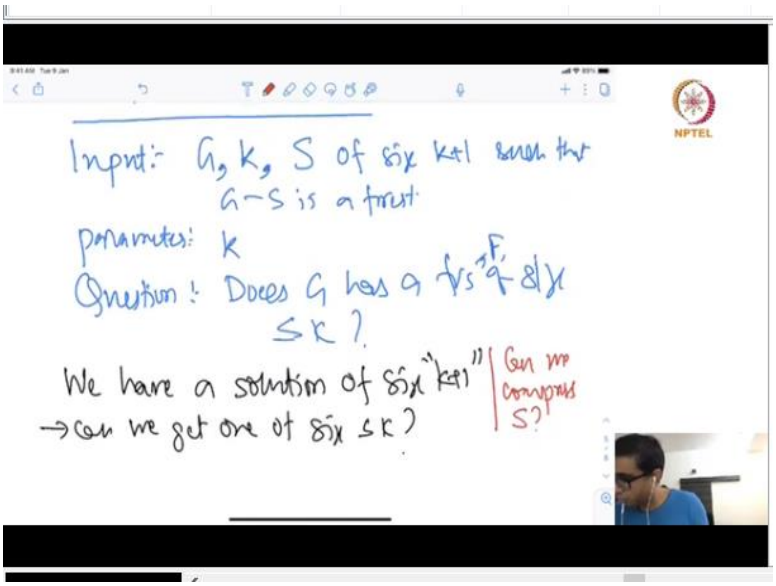


So, property of F i is here is my G i, suppose G i, the property is F i and G - F i is a forest, great.. Now, what is G i + 1? G i + 1 G i + vertex V i + 1. So, notice if I push V i + 1 here, then F union V i + 1 is also a solution of size maybe at most k + 1 of G i + 1. Because look after delete F i G i – F i is a forest. So, if I look at G i + 1 a bit, I not only delete F i I also delete V i + 1 then what is remaining G i – F i is nothing but G i + 1.

So, in this graph, what is this? So, let us call it G i - F i, it is same as G i + 1 - F i union V i + 1. That is great. Let us call this set. I am going to call it set F i + 1 hat or nothing hat might be a bit too much of a notation. Let us call this S i + 1. So, what is S i + 1? F i union V i + 1 or maybe, let us just leave that way. So, we are getting an additional information.

So, what is it new instance? Look at what is a new information we are getting, where the new information we are getting is the following problem. So, now if we are just a slide our self to a feedback vertex set then let us ask ourselves, what problem am I getting actually?

(Refer Slide Time: 23:17)



So, I am getting the following problem. And as the name suggests**,** we are going to call it compression FVS or feedback vertex set, what is the compression feedback vertex set? My input is G, k and some S of size k + 1 such that G - S is a forest. So, you are also getting an additional information is that now, I not only give you input I also give you a solution, could you just one more, my parameter is k and my question is does G had FVS of size at most k? Say F.

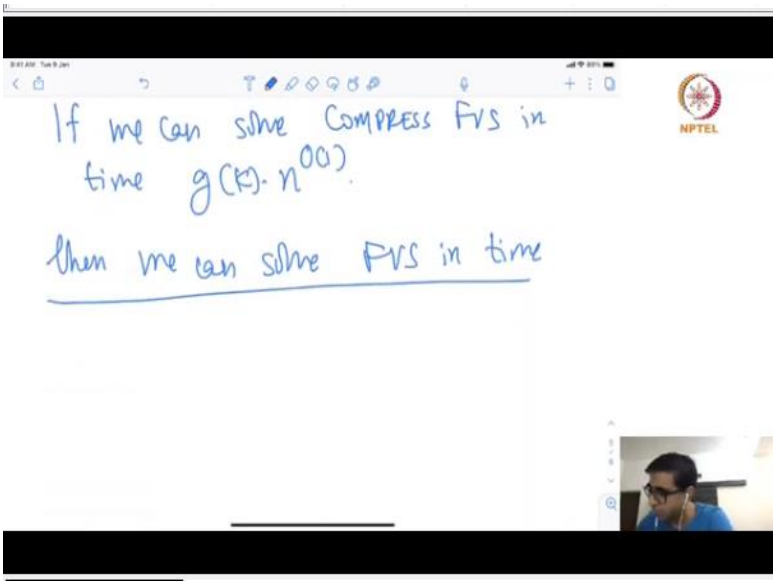
Now, it will make sense to you why are we using the word compression? Because what are we compressing? So, we have a solution of size k + 1 and what is the question we are asking? Can we get one of size at most k. So, which basically entails to us to say can we compress this? That is it. So, this is why the word iterative compression. We are iteratively solving this problem.

So, first of all, we are solving iteratively how that building our instance or building the graph incrementally one vertex each or you could do one edge or there are several other operations to build instances, you apply that and in each iteration because a minimization problem, we will either be able to say, look, if I am not able to find a solution to this problem, for the smaller instances, then I will not be able to succeed in solving this problem for the general instances.

So, no implies no for the whole problem. But if I have not, yes, then I do have an extra information. And what is that extra information? The extra information is that I have a solution to of size at most k to my current instance, with the help of the additional new vertex, which is going to be added in the next instance, gives me a solution for the new instance of just one size larger.

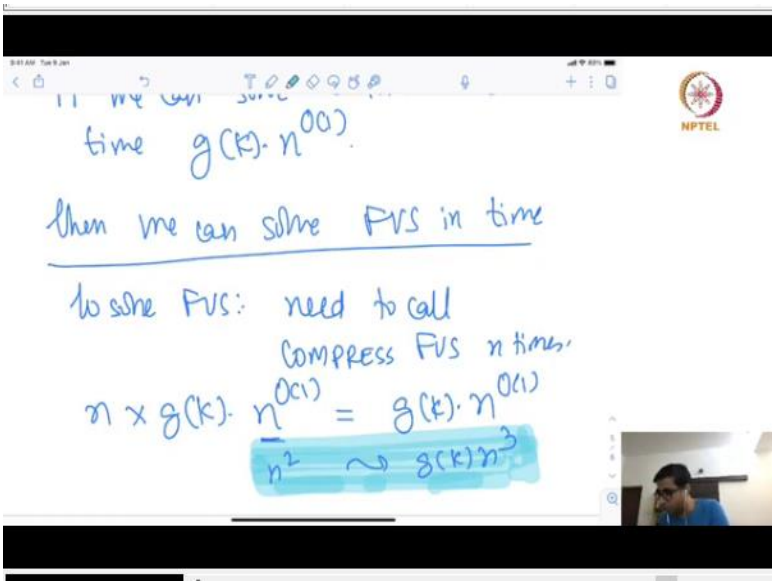
Notice that if I would have just directly returned O input G, S of size k + 1, you will ask me from where are you getting this solution, but the way this process is set up, this comes naturally. So, it does not look like magic, that I mean, you are looking for a solution of size k, but you are given an input a solution of size k + 1 from where are we getting, but this is the beauty or this is a strong point of this method that this solution of size k + 1 comes because of the inbuilt process which is involved in this whole method. So, now to solve FVS what do we need to solve?

(Refer Slide Time: 26:53)



Suppose if we can solve compress FVS in time, say g of k n to the power, O of 1 then we can solve feedback vertex set in time, how much? Look, we are going to call compress FVS how many times? N times at most.

(Refer Slide Time: 27:32)

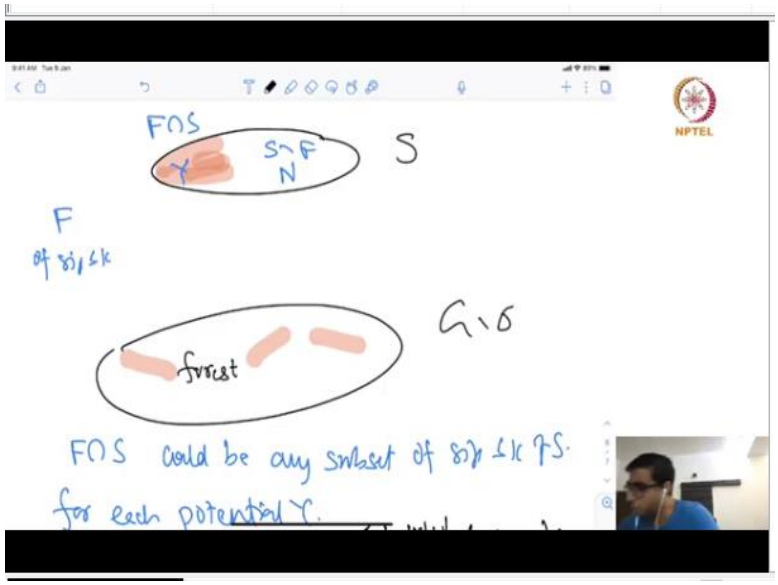


So, to solve FVS need to call compress FVS n times, so the running time is going to be n times g of k n to the power, O of 1. So, for example, which is still g of k, n to the power O of 1. But this is cheating in some sense, because if this would have been n square, then this will become g k of n cube. But for the purposes of this discussion, let us just ignore it for now.

It just an additional factor in the polynomial but you will see that sometimes it is important to get as good running time dependence even on the input size as possible, but let us not get into

that discussion at this point of time. Now let us look at compressed FVS slightly more. So, let us look at compressive FVS.

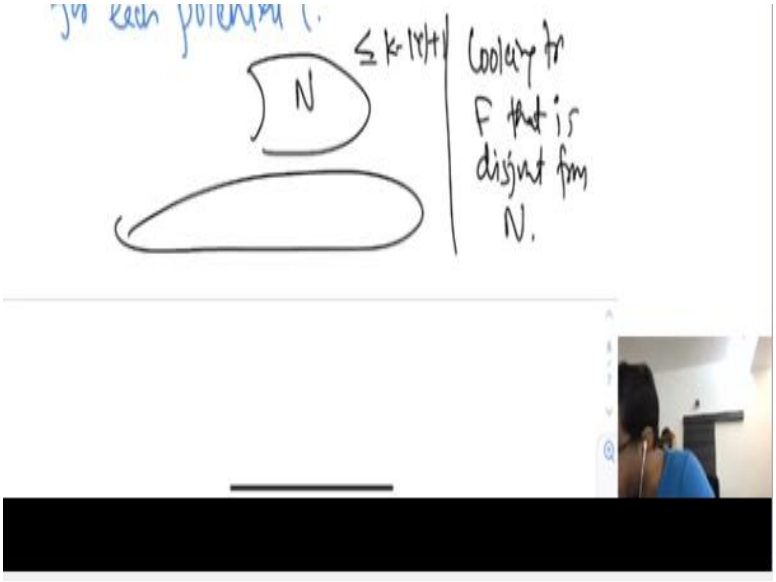
(Refer Slide Time: 29:00)



Now, we have seen this kind of approach. Just even in our first lecture, I showed you how to do some of these things. So now, let us look at how the world looks like, world looks like that I have S, I have G - S, which is a forest and what is my goal? Goal is to find F of size at most k.. So, now how could F look like? F could look like picking some vertex from here and maybe picking some vertex from here, this is how could F looked like.

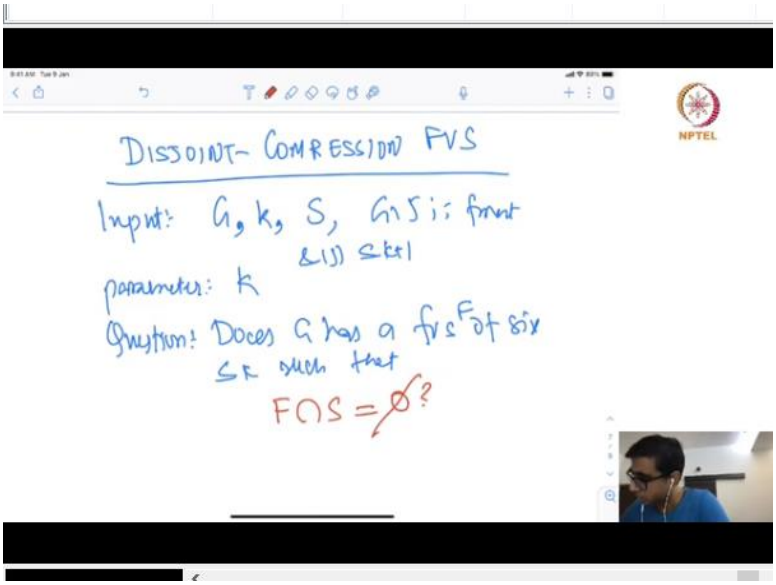
So, let us call F intersection S this part y, which means S, I will take you into my solution. That is why writing y. And this is S - F, I am going to write N. So, now ask yourself, how many potential choices of F intersection S are? So, F intersection S we do not know, could be any subset of size at most k of S. So, what I am going to do for each potential, y for each potential i, what problem are we solving for each potential. So, now, if you notice, then the problem becomes as follows. What is left?

(Refer Slide Time: 31:07)



This is what is left, which is n. And what are you looking for? You are looking for this is still found a solution, but maybe of size k - mod y + 1. And now, we are looking for F that is disjoint from no or n, because we already have guessed y. So, you delete y; like for most of the problems, we really we know that I am going to construct F which contains these y vertices then I guess y vertices and I delete. So, what problem am I getting? So, in general the problem which you will get to solve is the following.

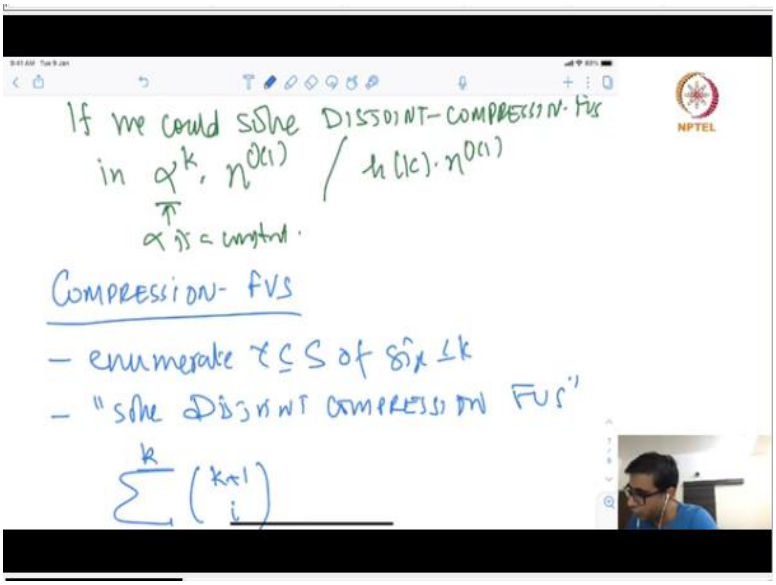
(Refer Slide Time: 32:08)



This is what is called disjoint compression problem FVS. We have generally except the feedback vertex set is just one example to illustrate but you could try to do this for odd cycle transversal, vertex cover any of the problems you have (()) (32:28) minimization problem, input is graph G, k, S such that G - S is forest, mod S is at most k + 1 or say k + 1.

Parameter is k and the question is does G has a feedback vertex set of size F of size at most k such that what is an important point F intersection S is empty. So, we are looking for a feedback vertex set which is disjoint from the given feedback vertex set and trust me this additional property gives so much structured to exploit and we will see several examples in coming lecture once I have set up the whole tool for you.

(Refer Slide Time: 33:48)

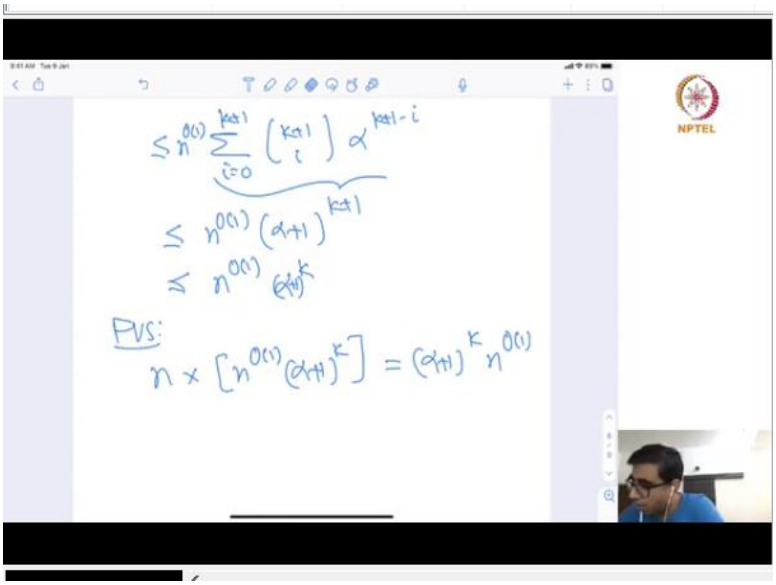


But imagine now that if we could solve disjoint FVS or for that matter disjoint compression FVS. Our people also omit the word compression if it is clear from the contracts. But let us for our initial lecture, let us see, if we could solve disjoint compression FVS in, say alpha to the power k n the power O of 1 or say some h of k n to the power O of 1. Like where alpha is a constant here. Let us say alpha is a constant. What can we solve our general problem?

So, let us look at to disjoint compression FVS. So, now I will tell you how to solve compression FVS. So, what we do? Enumerate y subset of S of size at most k and solve disjoint compression FVS. So, what is your running time this algorithm? Running time of this algorithm is very simple. So, I want to sum this up, i going from 0 to k from k + 1 choose i. So, this is a size of S and this is a selection of y and then I am going to solve what problem; I am going to solve disjoint compression.

So, for example, this runs in time alpha. So, this is going to be alpha to the power k - 1, k - i n to the power O of 1, that this is my running time to solve disjoint compression.

(Refer Slide Time: 36:00)

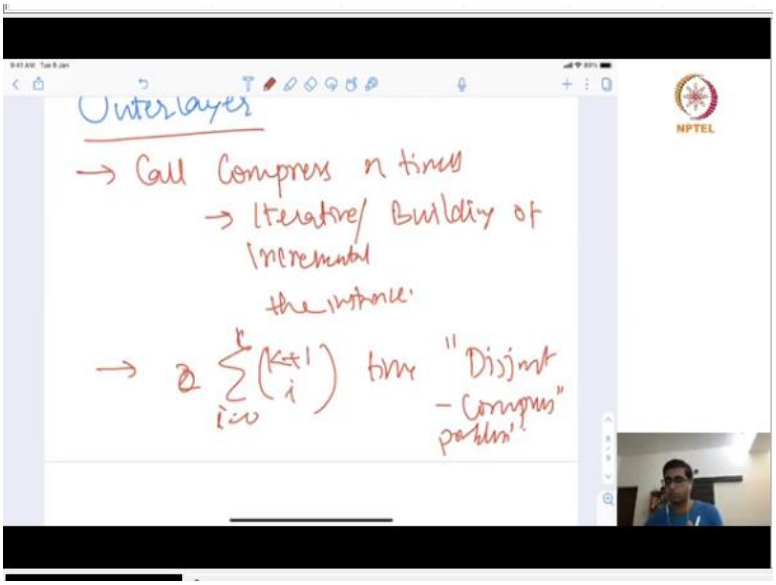


But what is this let us see, this is upper bounded by summation let us get this n to the power O of 1 out i = 0 to k + 1, k + 1 choose i alpha k + 1 - i. But then if you apply binomial coefficient what is this? This is n to the power O of 1 alpha + 1 to the power k + 1 which is like if alpha is a fixed constant, we can say this is nothing but n to the power of O of 1 alpha + 1 to the power k.

So, then to solve FVS, what is this? N times n to the power of O of 1 alpha + 1 to the power k, which is another alpha + 1 to the power k. So, this is why, so, we will just concentrate on how to solve disjoint compression of a problem, because the moment we know how to solve disjoint compression, if we put up everything here, then you can get it and similarly, I mean, if this not have been fixed constant alpha, but like some function, g of k, I mean, nothing will change, you can get some g of k + 1 to the power k into power n.

Maybe, let us not try let me I have not done the computation let us just check for yourself. So, basically, what I am saying that, some h of k, you can also do this, but like, that is beside the point. So, what I am trying to tell you that like, if we can solve disjoint compression problem, then we can solve the main problem. So, what we learnt.

(Refer Slide Time: 37:56)



So, there is an outer layer, what was an outer layer call compress n times. And that was because iterative building or incremental building of the instance. Now to solve compress call roughly summation k + 1 choose i, i gone from k times disjoint compress problem. So, this is how we got it. So like just let us just go back and let us see what we build.

So, now, I think I have justified you the term or the approach, builders instances up incrementally update the solution in each iteration and why I use the word inductive, because you can think of these instances or build up inductively we have built something of some size then one extra and so on, so forth. And so, first, I told you how to do incrementally build the instances example for the graph, which you will the most best as case and then you will understand how to do other things.

And I told you how to build for by arbitrary ordering of the vertices, but the same approach could also be done 2 edges. And then we looked at the feedback vertex set problem, we said what happens if we incrementally solve the problem if at some point if we get a no instance we can say no immediately, but what was interesting that when we are able to say yes. For the next instance, we are able to build a solution of one extra size.

So, now we have an help is that or we have a help that we do have a solution with size k + 1 and your question becomes can you find me one of size at most k with the help of the size k + 1 size solution. And that led to from compress FVS what we let to disjoint compress FVS. Now look at what is additional property. Because now we are not looking for a solution but

we are looking you have to reduce the problem to instances where you are looking for a solution which is disjoint from one solution.

So, when you had this property then what happens is that like you suddenly throw up so much more structure because you are looking for 2 disjoint solutions and 1 disjoint solution is given to you and you are looking. So, these distances are way mode structure and one can exploit them to design algorithm and that is what I will be doing in the next lectures, I will solve disjoint compress for several other problems and with that I will end this lecture today.