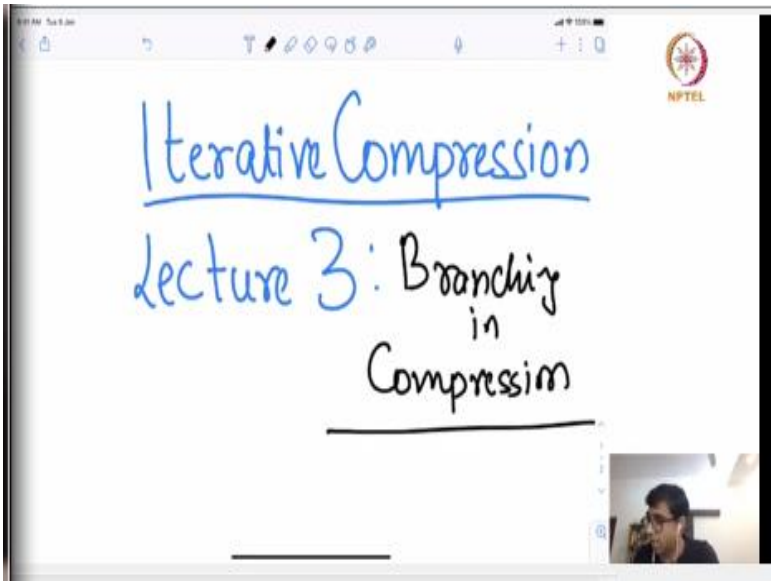
Parameterized Algorithms
Prof. Neeldhara Misra, Saket Saurabh
Department of Computer Science and Engineering
Indian Institute of Technology-Gandhinagar

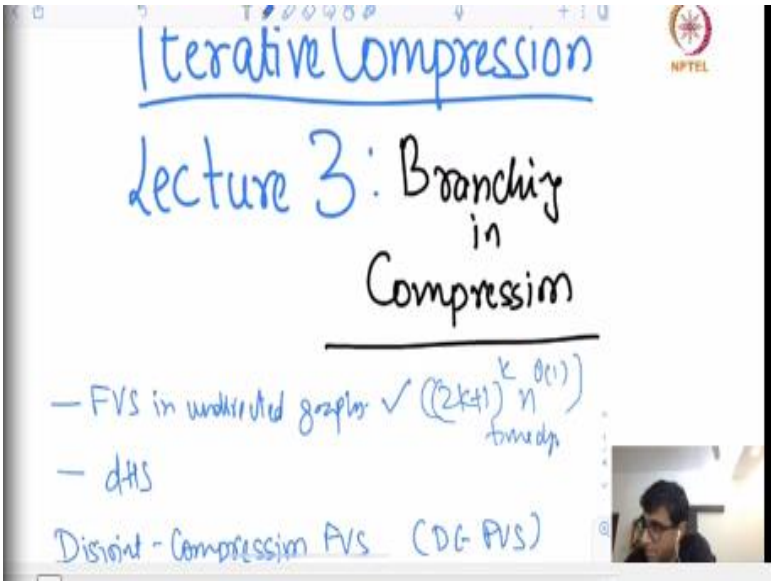
Module-03
Lecture-14
Iterative Compression III: Feedback Vertex Set and 3-Hitting Set

(Refer Slide Time: 00:16)



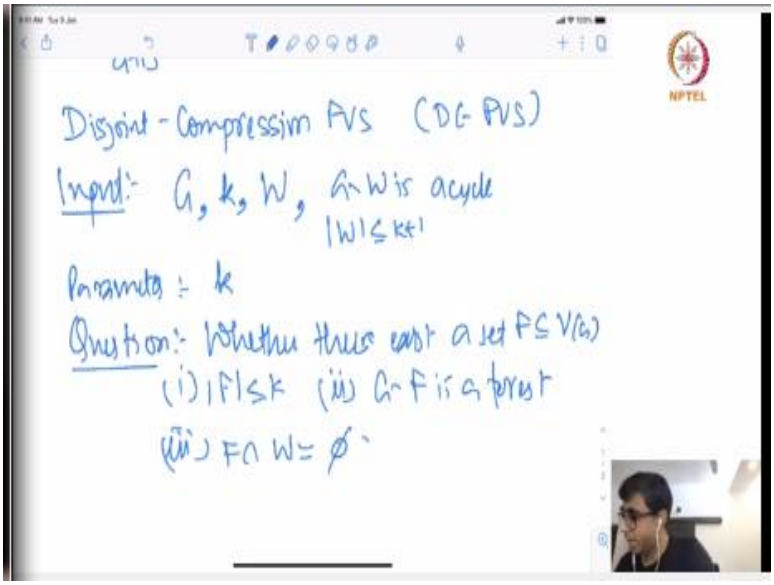
Welcome to the third lecture on iterative compression. And the third lecture will be about branching and composition. So, based on how much time we have we will definitely look at an algorithm for feedback vertex set in undirected graph.

(Refer Slide Time: 00:34)



FVS in undirected graph and if we will have time we will look at the heating set. But let us try to do feedback vertex set in directed graph and this is about how we can do branching in compression. So, recall we are going to solve what problem?

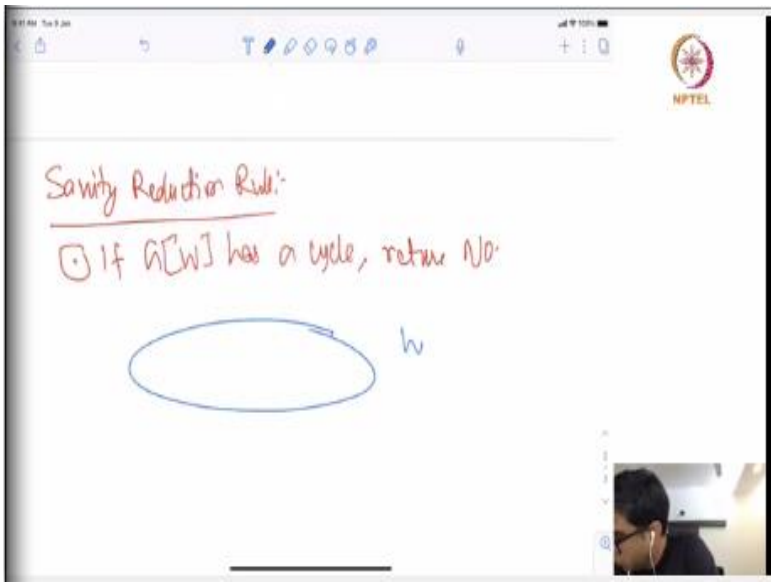
(Refer Slide Time: 01:00)



We are going to solve disjoint compression FVS. So, input is going to be G and integer k say W, what is the property of say W G - W is acyclic and W has size k + 1, my parameter is k. And the question we are going to look is whether there exists a set F subset of V G first F size k G - F is a forest, and third F intersection w is empty.

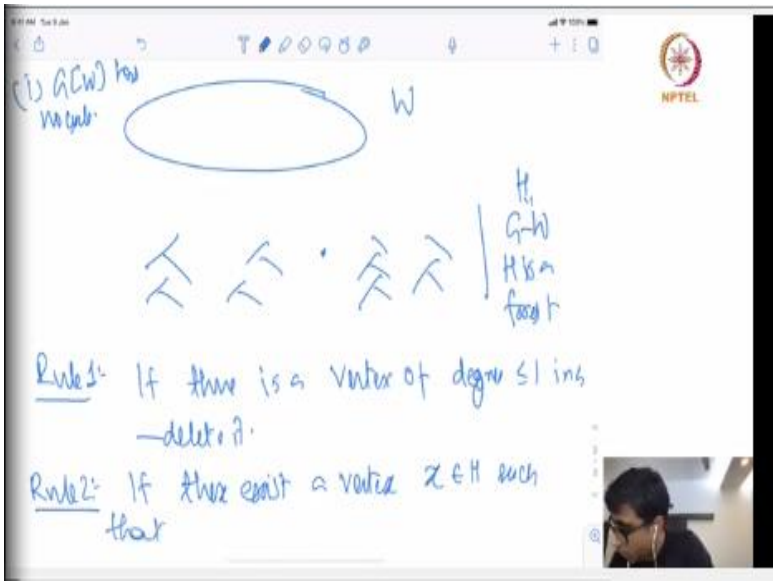
So, we are looking to test whether there exist a feedback vertex set whose intersection with the given feedback vertex set of size k + 1 is disjoint or not. So, as before let us apply our sanity reduction rule. So, what is our sanity reduction rule?

(Refer Slide Time: 02:33)



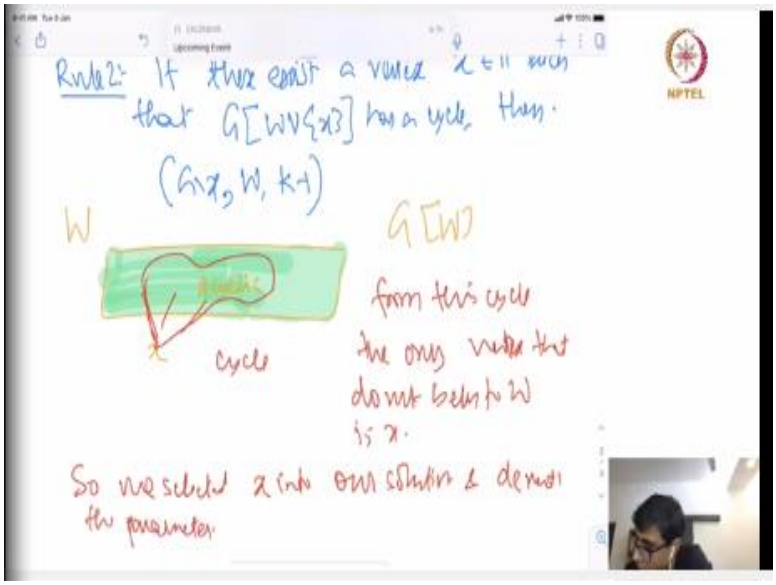
So, our sanity reduction rule is if graph induced and W has a cycle return no. So, if we are trying to find a feedback vertex set or a set which intersects all the cycles and graph induced and W has a cycle it means there is absolutely no way we can hit this cycle by a vertex which is not a W vertex. So, in that case we can immediately written no, so the world of our world looks like W.

(Refer Slide Time: 03:17)



And this is my forest, this is G - W which is a let us call it H is a forest, and I have drawn forest field. So, there are some reduction rules that we could apply immediately, so we will first apply some reduction rule. So, from now onwards first of all assume that graph induced on W has no cycle. Now, so first reduction rule that we will apply, rule 1 if there is a vertex of degree at most 1 in G, delete it. Now rule 2, if there exist a vertex x in H.

(Refer Slide Time: 04:40)



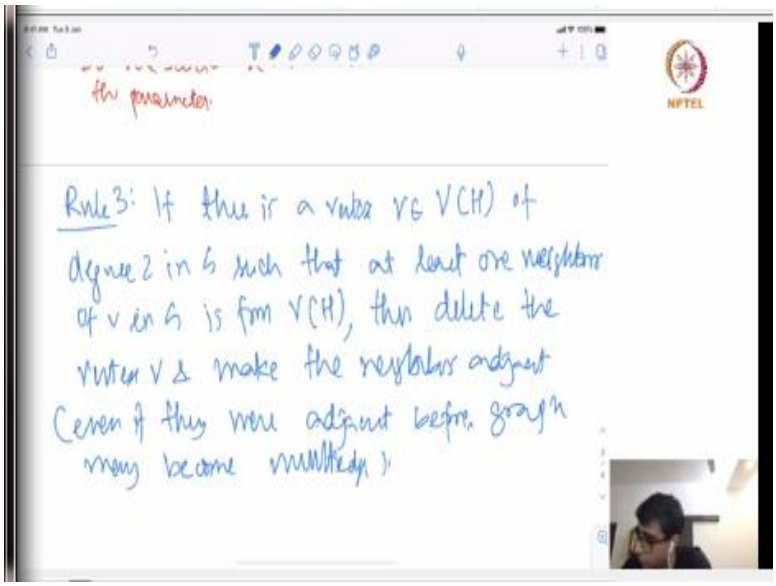
Such that graph induced on W union x has a cycle, then what will we do? G - x, W K – 1. Then we delete x remain W, why? Because what is this lemma tells? What is this reduction rule tells us that? Look, here is a cycle we know that W has no cycle, graph induced on W, so what is the reduction rule 2 it is exactly the way we have been doing for this triangle thing that look if there is a cycle that contains exactly one vertex which is not in W.

Then we must pick that vertex set. So, what we know? That look at this G W, this is W which is coloured and I have a vertex x, this is acyclic. But the moment I add x it has a cycle, what is the meaning of this? There exist a cycle which is of the following nature. So, from this cycle the only vertex that do not belong to W is x, so we selected x into our solution and decrease the parameter.

So, if you notice we have been applying these kind of reduction rules in for feedback vertex set of tournament also, where we had that look if there is a triangle where 2 vertices of that triangle

belongs to the forbidden set, then the third vertex must belong to the set we are going to construct. So, final we have a reduction rule 3, so what is the rule 3?

(Refer Slide Time: 07:03)

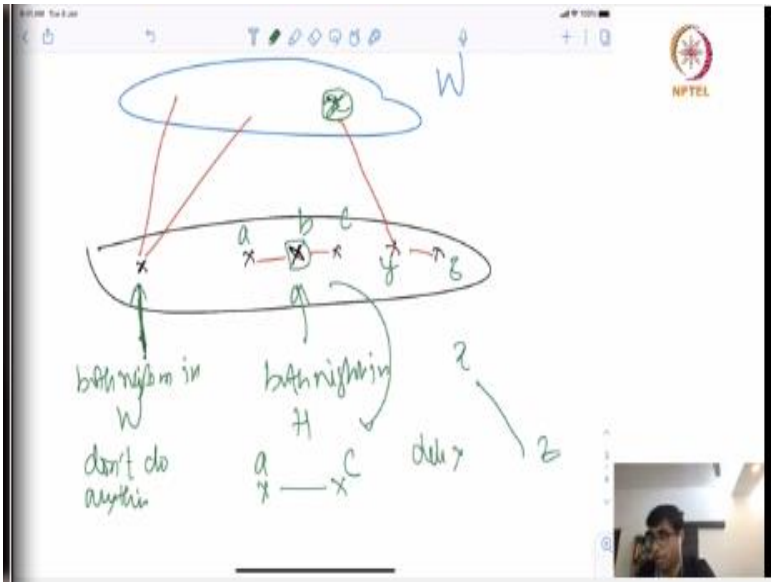


If there is a vertex V in vertex set of H of degree 2 in G, if there is a vertex V which belongs to vertex set of H but it has degree 2 in G. So, you must have seen degree 0, 1, 2 reduction rules to make a graph degree at least 3 in your lectures on kernelization or not kernelization but in the branching algorithm where you would have seen for feedback vertex set. You have previously seen this previously seen this 2 k + 1 to the power k n to the power o of 1 time algorithm.

So, in that you must have seen this reduction rules. So, we will try to make a similar degree 2 reduction rule. If there is a vertex V in V of H of degree 2 in G such that at least one neighbour of V is in V in G is from vertex set of H. So, it is not that you know I have a degree like I have a vertex of degree 2 where both his neighbours are like in W but there is at least one neighbour which is in V of H.

Then delete the vertex V and make the neighbours adjacent, even if they were adjacent before. So, this might lead to multi edges, graph may became multi edge, so what is this reduction rule?

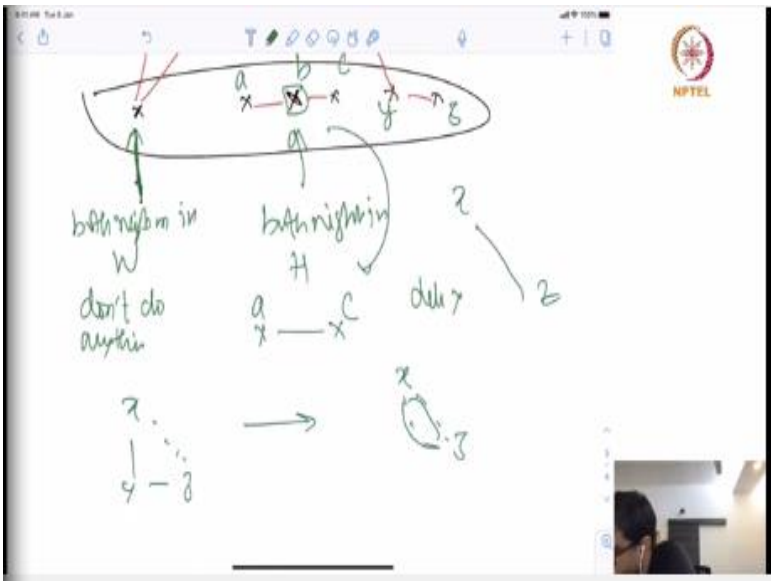
(Refer Slide Time: 09:33)



So, basically what it tells us that look, first of all I am only going to look at those vertices of degree 2, so look at a degree 2 vertex here. How can his neighbours are? His neighbours could be where are his degree 2 neighbours? His degree 2 neighbours could be this or this vertex could be both of his degree neighbours are here, or this vertex could have a degree here, here. So, what I wanted to draw is it could be like this.

So, what is our reduction rule tells us that? Look, if you have something of this nature where both of it is neighbours in W do not do anything, both neighbours in H then what will we do? What will the reduction we will do here? Well, here the vertex, so we will just make this guy this. So, suppose this is a, b, c, here is an a and c delete b. Now suppose let us call it you give me a name say x, y and j, if you have y, so that there is one vertex in y other in z. So, what will you do? You will make x and z neighbour and by deleting y.

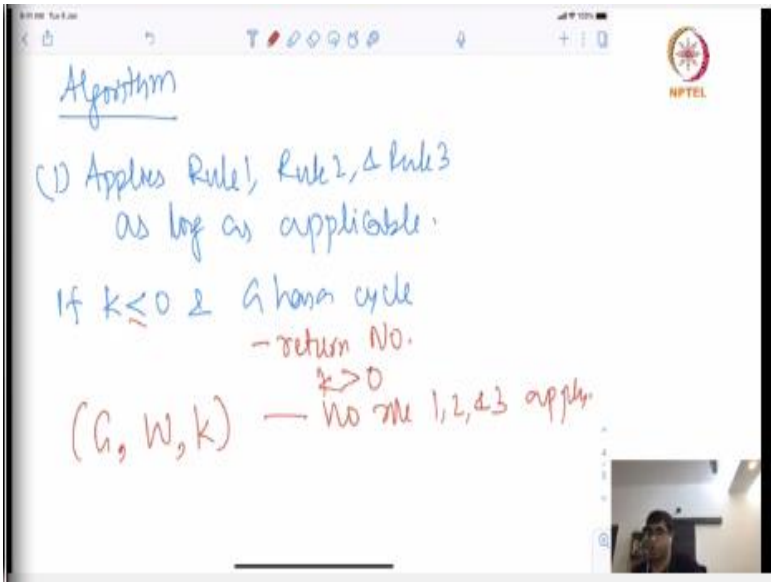
(Refer Slide Time: 11:44)



It could have been, so for example our situation could have been that x is there, y is there and z is there and there is an H here. But then the reduction rule will do, x and z designation so this is why it say. That if you have a degree 2 vertex in graph G, so these are vertices of degree 2 in graph G, such that at least one neighbour is in H then we will apply this following short circuit operation, delete this degree 2 vertex and add an H between both of it is neighbours.

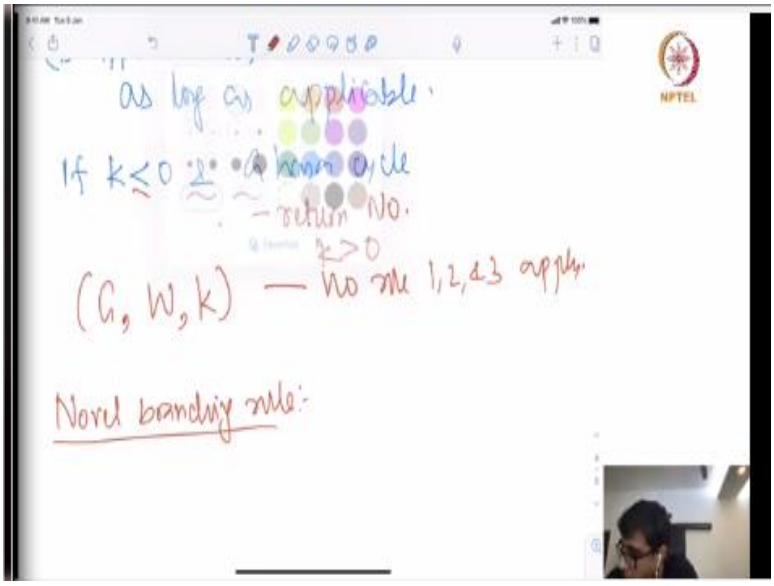
Even though even if there was an edge before you make it a multi edge, so this is what you do degree 1, degree 2. So, what is our algorithm does?

(Refer Slide Time: 12:27)



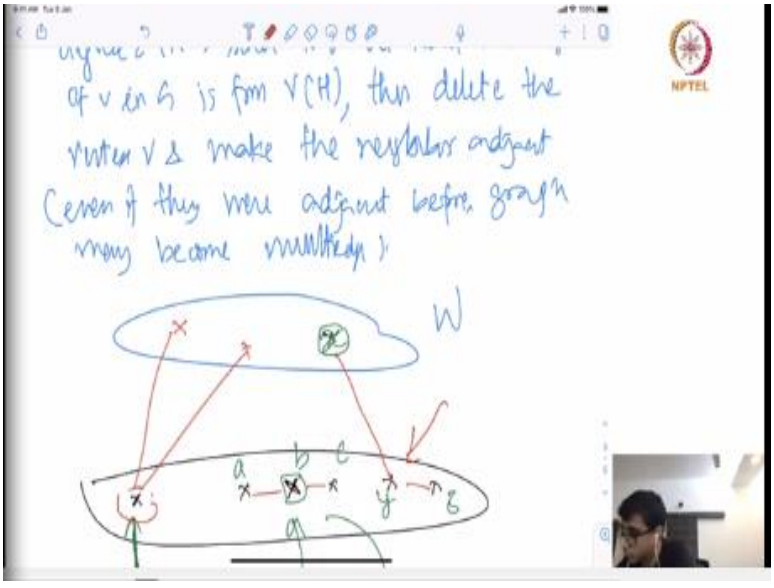
Our algorithm is very simple, applies rule 1, rule 2 and rule 3 as long as applicable. So, applies rule 1, 2 and 3 as long as applicable. Now at any stage if k is less than 0 and G has a cycle what will be do? We will say return no. So, for now let us assume that we have a instance G, W, k, k is greater than 0, strictly greater than 0 and no rule 1, 2 and 3 apply. Now we will apply a very novel branching rule.

(Refer Slide Time: 13:57)



So, our idea is very simple, novel branching rule. So, let us look at our world again.

(Refer Slide Time: 14:12)



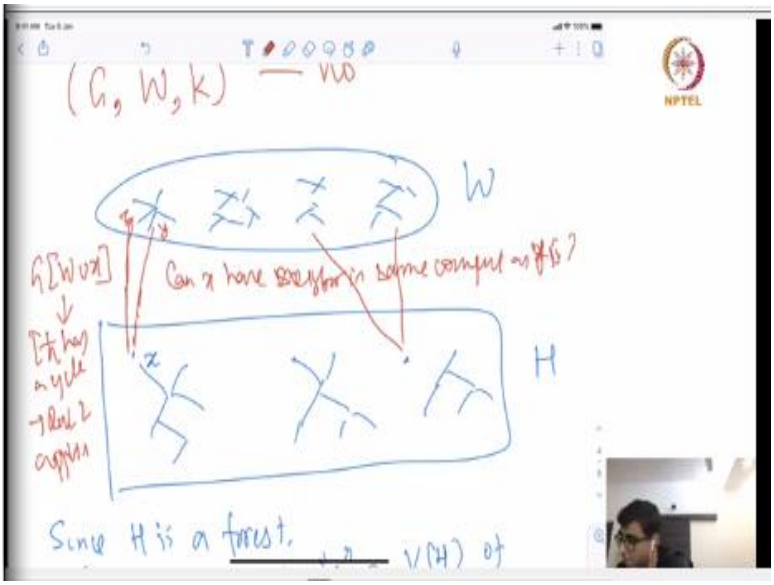
Look I have not proved the soundness of reduction rule 1, 2 and 3 but they are just obvious. So, like similarly like for example why they are obvious? Look why do you need to pick y? Because

every cycle that do pass through y here every cycle that do pass through y, every cycle that do pass through y also passes through x and z. And although x is not allowed to be picked, you can pick z into your solution.

Similarly you can delete b and every cycle that passes through b also passes through a and c. So, in the replacement solution and the reason why I cannot apply reduction rule here. Because look it is both neighbours are in W and both are forbidden. So, if I say that ok, let us put an edge and look at a solution where we would like to replace wherever x appears, we cannot replace with any of these vertices because they are forbidden.

That is why in this disjoint feedback vertex set, we are only able to apply reduction rules to eliminate degree 2 vertices for certain kind of degree 2 vertices not all kind of degree 2 vertices. And we will see that whatever elimination rule or whatever degree 2 reduction rules we have been able to apply that is sufficient for us to make progress in the branching algorithm that I am going to explain next.

(Refer Slide Time: 15:40)

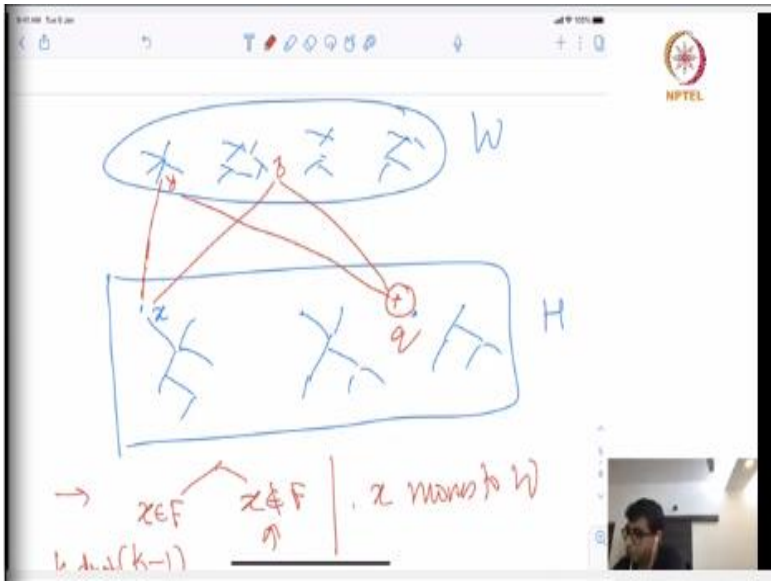


So, let us look at how the world looks like. So, the world looks likes very interesting, here is my W and this is also some forest, here is my H. Now since H is a forest, what can we say? There exist a vertex x in vertex set of H of degree at most 1. Now let us say this is the vertex x of degree at most 1. Now ask ourselves, where are it is neighbours?

Look, since we have applied there is no vertex of degree 1 in this graph, it has at least 1 neighbour and since it is degree is 1 in H it is other neighbour must belong to some vertex in W. So, suppose it has neighboured some y, great. Now let us ask ourselves a following question. But why it has only one neighbour? It has degree say exactly equal to 1.

So, now where like but if this is how it looks like then why did we not apply the reduction rule 2? Reduction rule 3 which takes care of this kind of degree 2 vertices, that is because x has at least one more neighbour somewhere. So, now there are 2 questions, can x have neighbour in same component as x is, as y is? Well, no, because suppose I have this z then what is the property? Then look at the graph induced on W union x, it has a cycle, rule 2 applies. So, the world is slightly different.

(Refer Slide Time: 18:42)

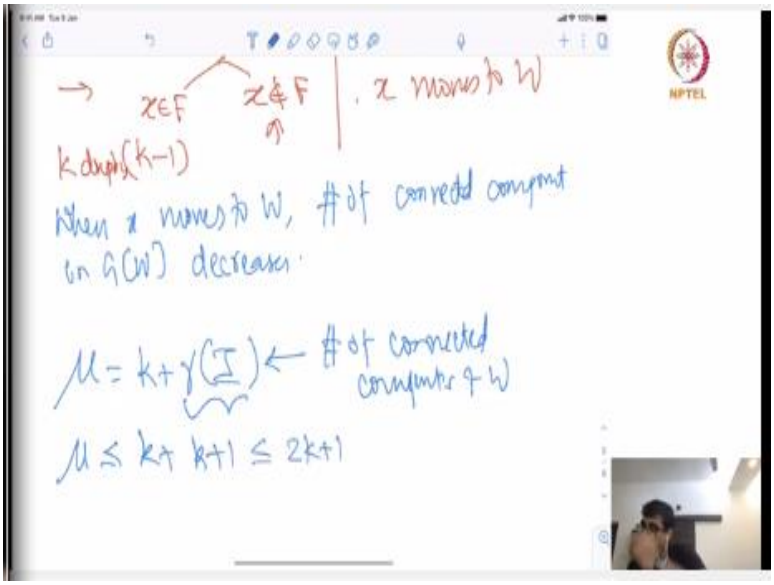


So, this situation cannot arise, so where is his other neighbour, it must be somewhere in other component. So, now on this kind of vertices I would like to branch, so how will I branch? Correct, let us do branching. So, what is my branching? x belongs to my F, x does not belong to F. So, if x belongs to F, k drops by k - 1 but what happens in this case?

In this case all I know that x moves to W, but if x moves to W what happens? Nothing much can I say that all I know is that you have increase the number of vertices that what is that called

which are forbidden. But there is something interesting happens actually, something interesting that happens is and this is the something which people have exploited is that.

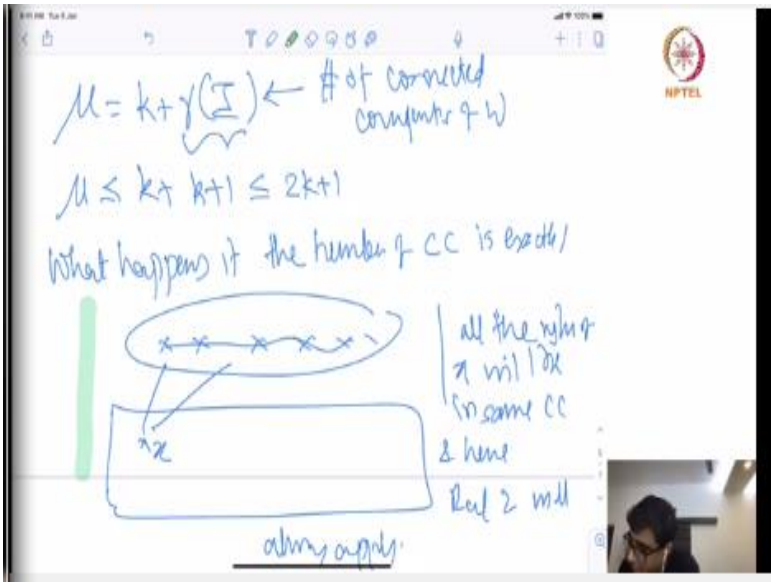
(Refer Slide Time: 20:17)



When x moves to W, number of connected component in graph induced in W decreases. So, in this case also something does decrease, and what is decreasing number of connected components of W. So, the measure which I am going to use is k plus let us say let us say gamma of W or let us say gamma of, what is this number of connected components of W.

In the beginning what is the measure? It is at most upper bounded by k, how many vertices are there in W k + 1. So, number of connected component is upper bounded by k + 1, so this is great. But intuitively you must ask yourself I mean yes; we are making a progress but then what happens if there is only one connected component? What happens if the number of connected component is exactly equal to 1?

(Refer Slide Time: 21:49)

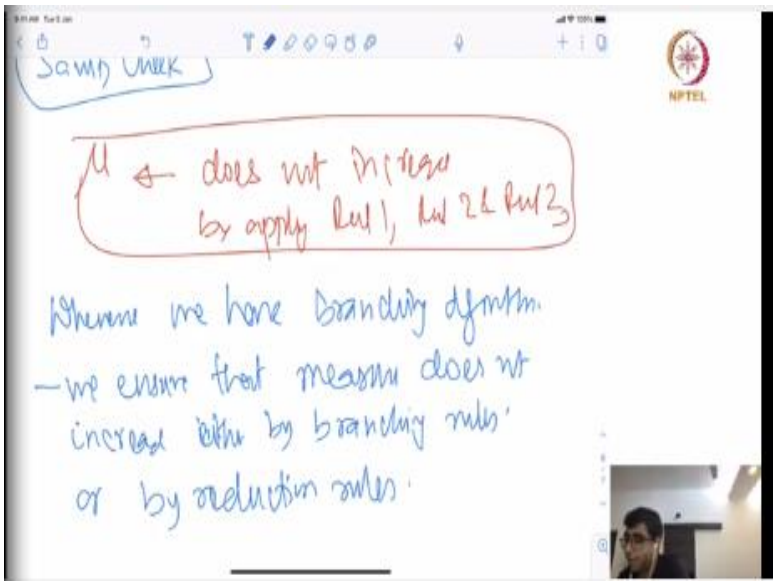


Then the number of connected component is not going to decrease, because number of component then if you notice what happens? It is that, this is just one connected component and look at a vertex which is still you will, because you cannot apply reduction rule will still find a vertex x who will have at least 2 neighbours there. But now these neighbours are in the same connected component and hence all the neighbours of x will be in same connected component and hence rule 2 will always apply.

So, this is not required for the proof but this is just for your sanity check that a natural question that do arise is that fine, oh! it is great that we are making a progress when the number of connected components are decreasing. But what happens when the number of connected components do not decrease. So, if the moment you reach that state of the algorithm then you can solve the problem in polynomial time just by applying reduction rule 1, 2 and 3.

So, either what can happen that reduction rule 1 applies because you can remove degree 1 vertices and W finishes then you have an empty solution or either or every time because there is only one connected component. And notice that because this major.

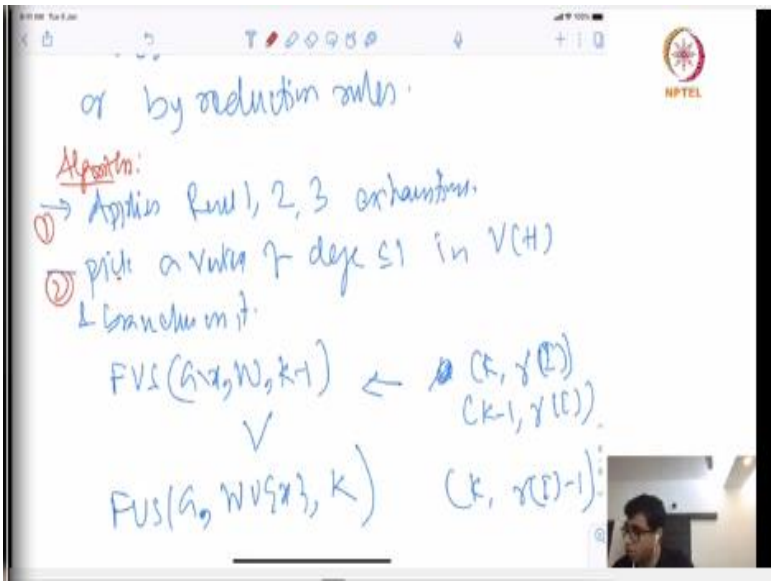
(Refer Slide Time: 23:49)



So, now that we have introduced a measure important point to notice is that measure mu does not increase by applying rule 1, rule 2 and rule 3. Because what is rule 1? Remove degree 1 vertex, when if you remove degree 1 vertex the number of connected component cannot decrease, like it cannot increase number of connected component cannot increase by deleting a degree 1 vertex. By deleting a vertex in x the number of connected component in W will not decrease.

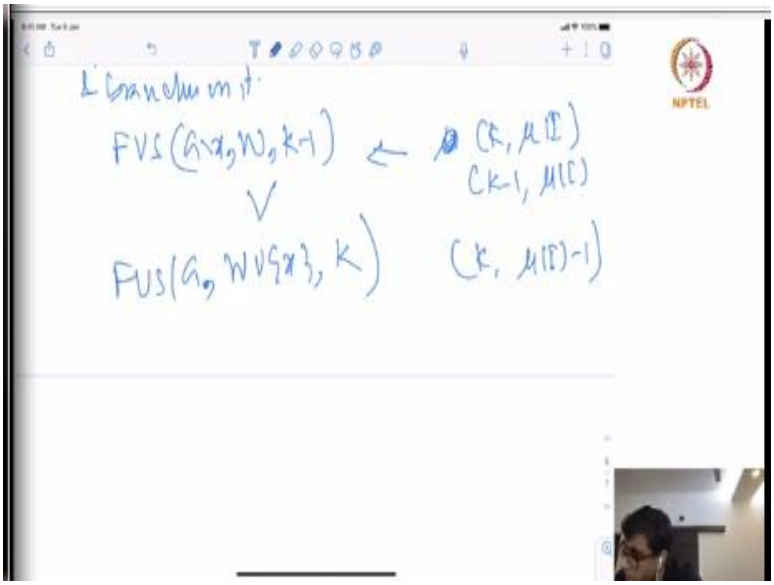
Similarly if I look at the degree 2 vertex the way we were making this edges those also will not decrease the number of connected components in W. Which implies, that the mu does not increases what you call? Mu does not increase by applying reduction rule 1, 2 and 3. So, this is very important that whenever we have branching algorithm we ensure that measure does not increase either by branching rules or by reduction rules. So, now what is happening? So, now algorithm is very simple.

(Refer Slide Time: 25:53)



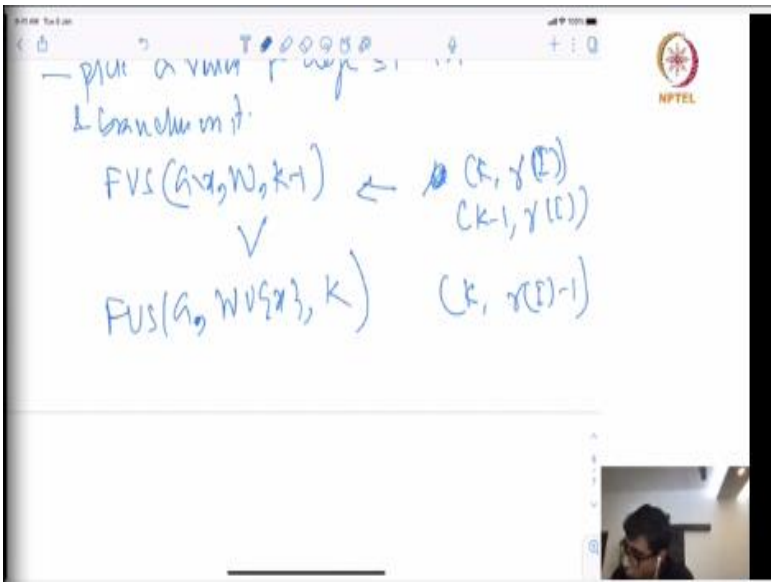
Applies rule 1, 2, 3 exhaustively picks a vertex of degree at most 1 in vertex set of H and branches on it. How? So, suppose I have so variety branches of this, FVS of G – x, W, k - 1 or FVS of G, W union x, k.

(Refer Slide Time: 26:41)



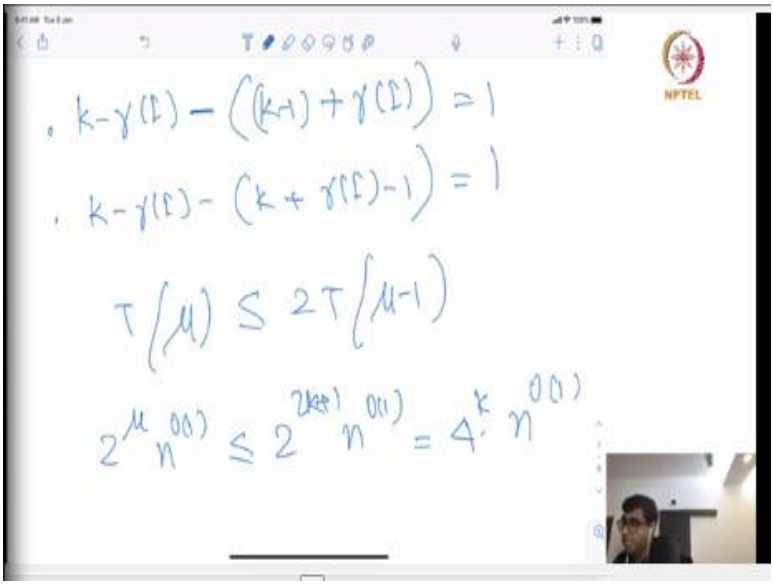
And in both cases what is this? So, in both cases here k drops, so I can think k and mu of I, so in this k drops by 1 mu of I remains same, here k remains same mu of I becomes less than 1. So, now what did I use? Mu of I, no, I use gamma sorry.

(Refer Slide Time: 27:12)



Number of component and here.

(Refer Slide Time: 27:27)



So, in the beginning we have this, so what is the change in measure? k - 1 - + gamma I, so this is what the major changes, so this is the drop is equal to 1 in this case. And in the second case what happens is that we have k - gamma I - k but - 1, so this is it. So, which implies the running time of this algorithm is given by T of mu is 2 times T of mu - 1, so the algorithm runs in times 2 to the power mu n to the power O of 1. And mu is like how much?

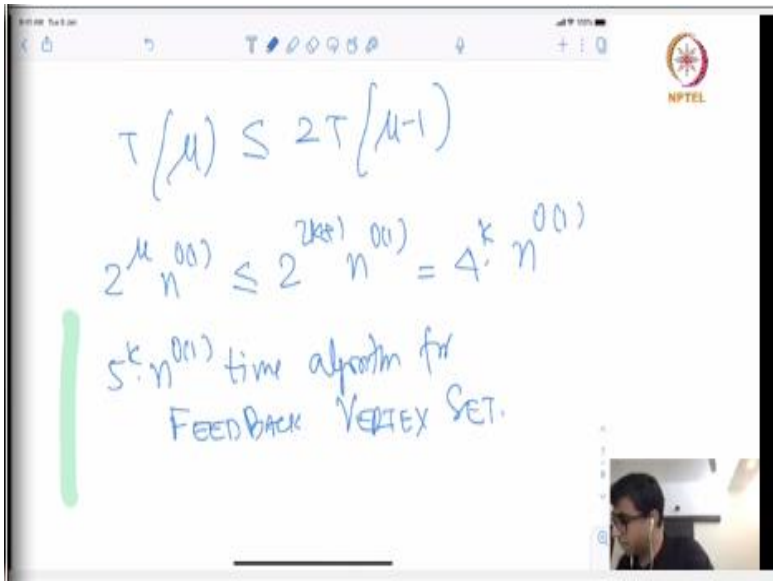
We just got an upper boundary mu is like 2k + 1 here, mu is upper bounded by 2k + 1. So, what do we get? Which is same as 4 to the power k into n. One thing I did not tell you is that here we

said look at a vertex of, but we are only looking for a vertex of degree at most 1. So, you could also have a vertex of degree 0, if you will have a vertex of degree 0, then where are it is neighbours? Then it is easy to see that it is neighbours are, it has at least 2 neighbours for sure we know and they must belong to 2 different components.

Because otherwise if they belong to the same component reduction rule 2 would have applied again or here, otherwise you can just branch on this vertex call it q same way as you have branched on x, because either q goes into the solution or q moves to W and reduces the number of connected component by 1. So, that kinds of completes my algorithm, algorithm was fairly simple, algorithm just applied some 3 reduction rules, algorithm just applied reduction rule.

Step 1 and step 2 and after applying the reduction rule it picked a vertex of low degree like degree at most and branched either by putting this in the solution or not putting the solution. But we saw a very interesting measure or interesting way to analyze the running time of this algorithm which decreases the measure in both the branches leading to an algorithm with running time 4 to the power k. Now by using general methodology what do we get?

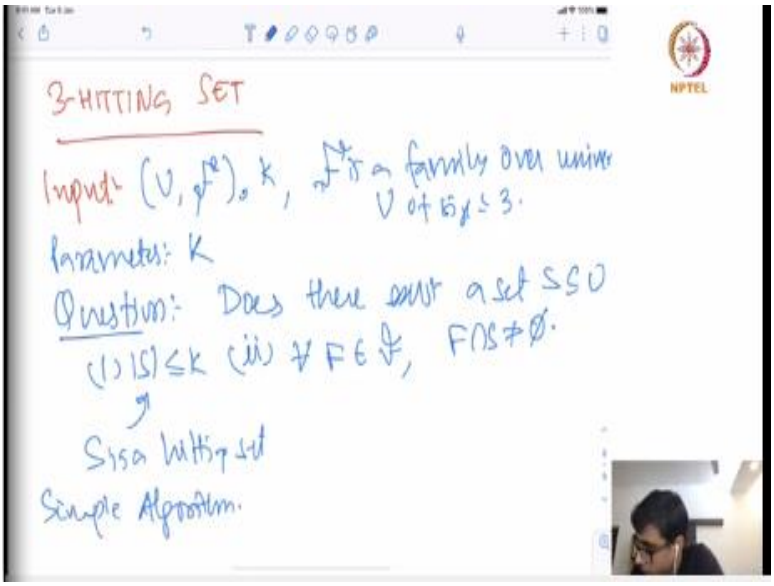
(Refer Slide Time: 30:45)



We get 5 power k n to the power O of 1 time algorithm for feedback vertex. So, what we have been able to get is a 5 power k n to the power O of 1 time algorithm for feedback vertex set using this. Because remember we said that if we can get an alpha to the power k algorithm for disjoint

version then 1 + alpha to the power k time we can get an algorithm for a feedback vertex set. I still have some time and I wanted to give you some algorithm other than this. So, I want to give you an algorithm for a different problem.

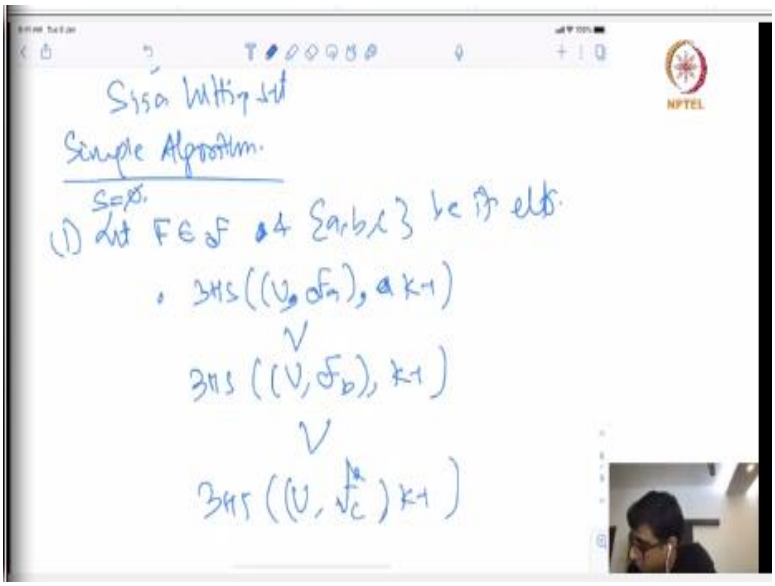
(Refer Slide Time: 31:47)



So, the different problem that I would like to teach now is say 3 hitting set, so what is the 3 hitting set? Input is universe, a family, an integer k, parameter k, and what is the, so I will write it down. And F is a family of U; F is a family over universe U of size at most k. And the question is does that exist a set S subset of U first, it is size is at most k.

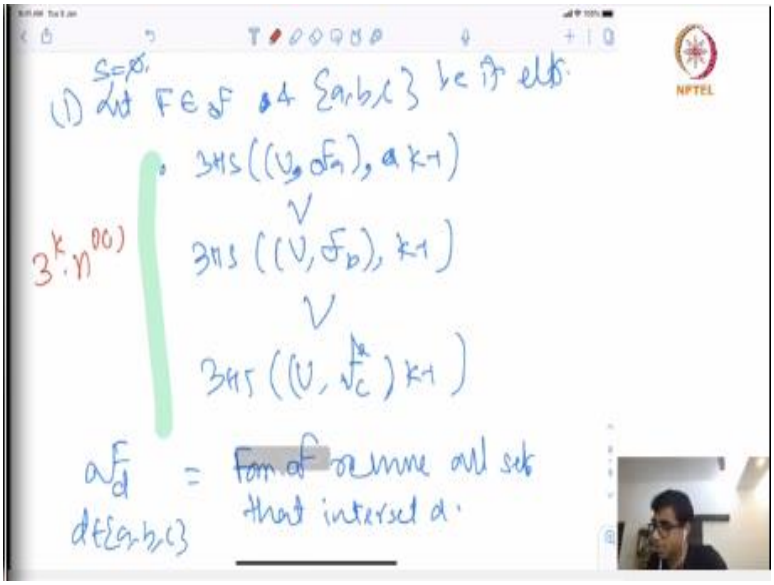
And secondly for every F in curly F let us say curly F, F intersection S is not equal to empty set, that is S is a hitting set. So, you are given a universe where each and universe and a family, in a family that a each set is of size 3 or less no more than that. And so clearly 3 hitting set has an algorithm, what is a simple algorithm?

(Refer Slide Time: 33:42)



Here simple algorithm branching algorithm first. Let F in, so in the beginning suppose S is empty set, let F in F and a, b, c be it is element, then what I am going to do? Three hitting set U, I will write it down F a, k - 1 or 3 hitting set U, F b, k - 1 or 3 hitting set U, F c, k - 1.

(Refer Slide Time: 34:53)

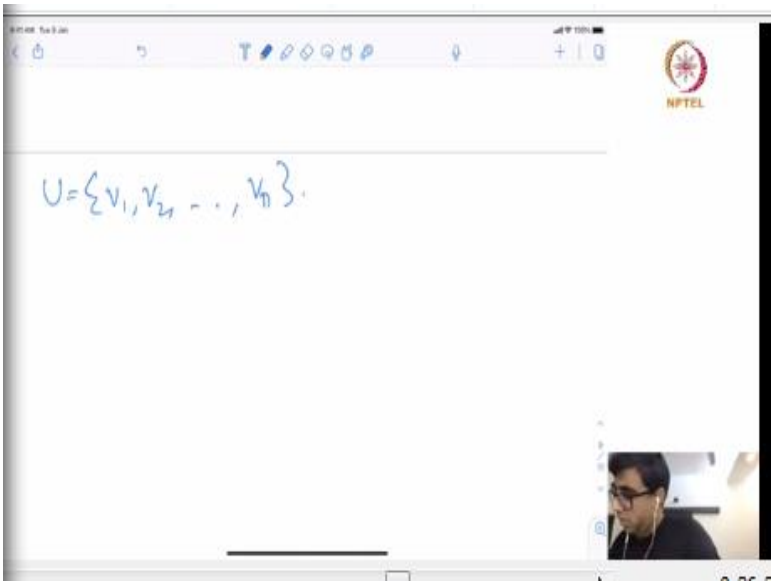


And what is F d? d belongs to a, b and c is that from the family remove all sets that intersect d. So, basically I have looked at F b, now that I have decided that I am going to select a into my solution, what I do? I look at the family, now that I have decide that a must belong to my solution, then I remove all the sets that contains a because those are already hit by a, so you do not need to hit any more.

So, that it reduces to problem of hitting the new family F a. Now so what this algorithm will run time is? So, this algorithm is going to run in time 3 power k n to the power of O of 1. So, each of time you call this and like in the beginning it was, so this is a if you are looking for a decision version you do not need any S or something, you can just run this algorithm and you are very happy.

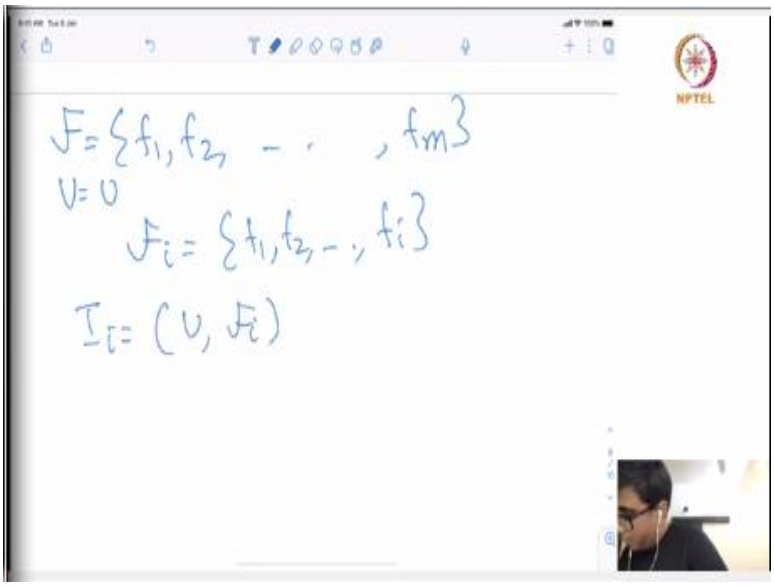
Now what I am going to do? I am going to apply a compression version of this, so look it is not a graph but the compression is, so how will you apply compression version for this? It is not a graph, so we still will do the same thing.

(Refer Slide Time: 36:16)



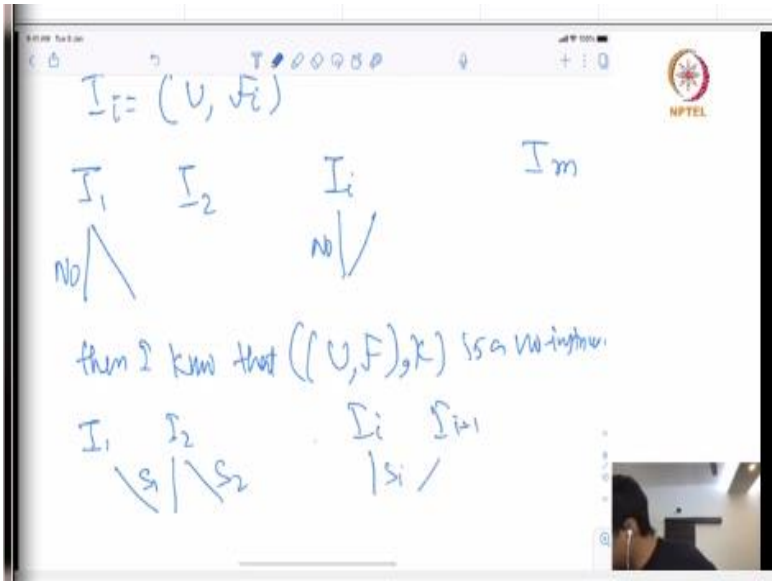
Suppose you have universe is like V 1, V 2, V n or rather you will see a different strategy here.

(Refer Slide Time: 36:36)



So, here is my family F suppose it contains a sets f 1 to f m over U. So, now what I am going to construct? And U is U I am going to construct family F i is some first f i sets. Now my instance I i is going to be U, F i.

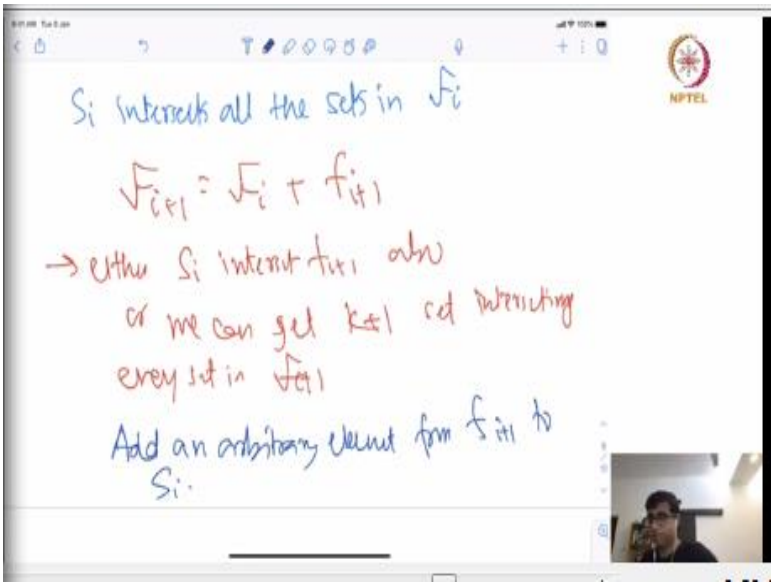
(Refer Slide Time: 37:07)



So, now notice, so I will run I 1, I 2, I m, this is how I am going to run my compression algorithm. Notice that if it says me no at any point of time if it says mean no, if it saves me no what do I know? If it says me no, then I know that U, F, k is a no instance, because if you cannot even hit small subset of family with k elements, what can you hit with? How can you hit all the sets?

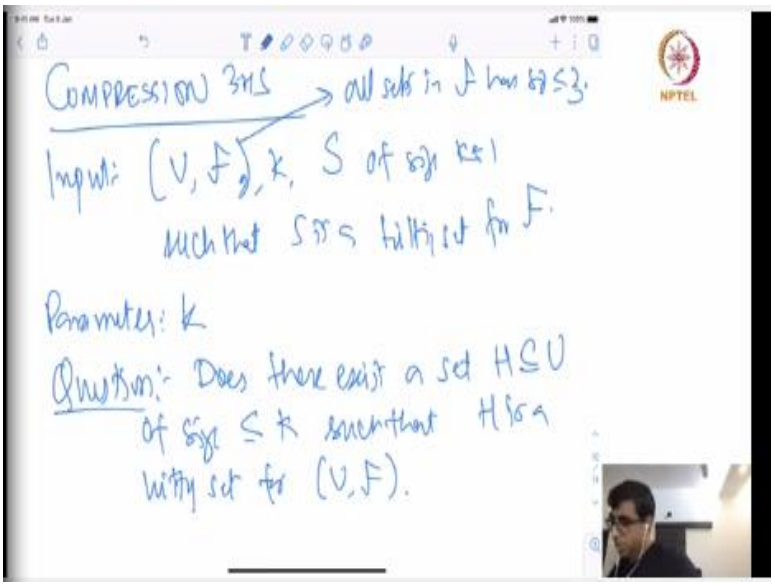
So, this is exactly like the way we have been doing for vertex cover or feedback vertex, so this is perfectly valid reason. Now what happens? So, suppose we do not get into know, so what is happens? Then we are going to get I 1 and sum set S 1, I 2 but what do I know about? Then get to S 2 so on and so on look at I i, S i and look at I i + 1. Now what I know?

(Refer Slide Time: 38:36)



S i intersects all the sets in I i or all the sets in F i, it intersects all the set. Now but what is F i + 1? If I + 1 is actually nothing but is F i + f i + 1 set, so what could happen? Either S i intersects f i + 1 also or we can get k + 1 set intersecting every set in F i + 1, how? Add an arbitrary element from F i + 1 to S i, that is it. So, this is how we can get to a compression problem. Now let us say how we can solve the compression problem.

(Refer Slide Time: 40:08)



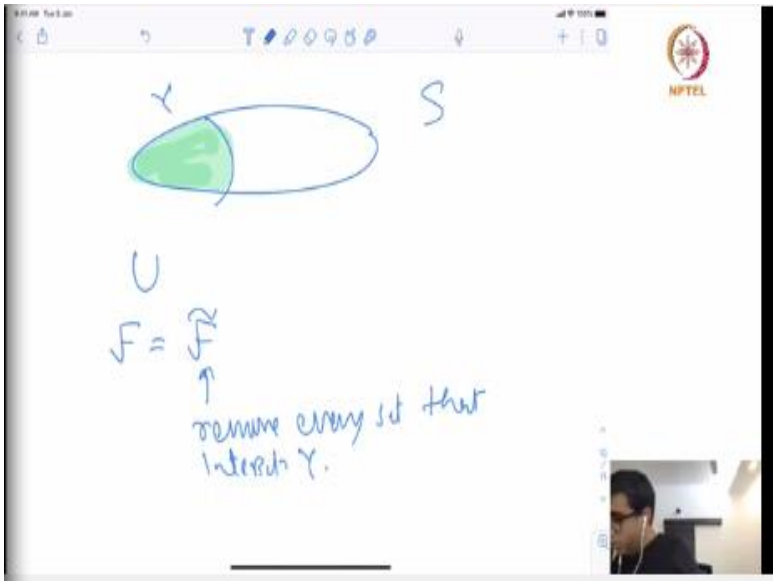
So, what is our compression problem? Compression problem is exactly the same way, compression 3 hitting set; what will the compression 3 hitting set problem? Input is going to be U, F an integer k, a set S of size k + 1 such that S is a hitting set for F. And I am assuming from now onwards that all sets in F has size at most 3 do not like I am not going to repeat it again, parameter is k.

And the question is does there exist a set H subset of U of size at most k of size does there is a set H subset of E of size at most k. Such that H is a hitting set for U, F, this is a compression 3 hitting set problem.

(Refer Slide Time: 41:53)

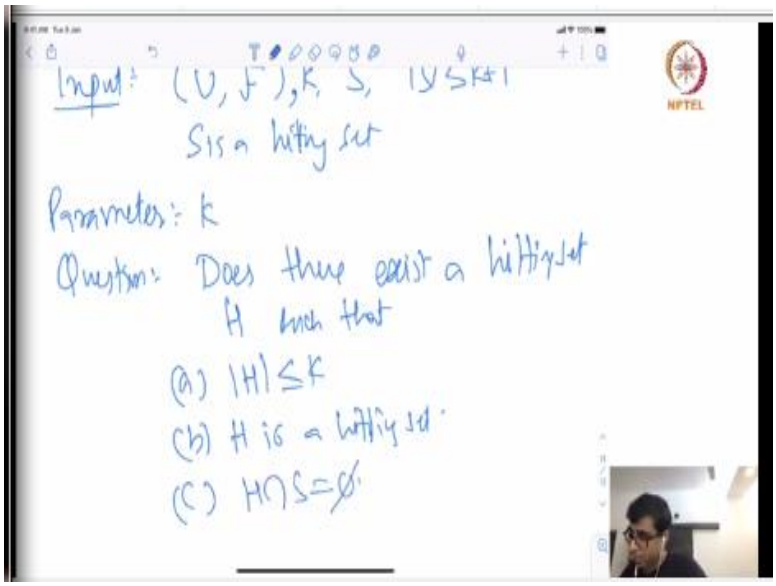
Now how do we get to a, let us say disjoint compression 3 hitting set problem. How? So, let us we will get to that problem in a minute. So, to get there what we used to do? We used to we said ok, let us from here we said look you gave me an S I guess a set, I guess a Y I guess a set Y which is going into my solution. And then what should I do? So, for graph problem we just deleted Y and then like solve the remaining problem on the remaining graph. But here what will I do? I have decided to put Y, so what I will do?

(Refer Slide Time: 42:58)



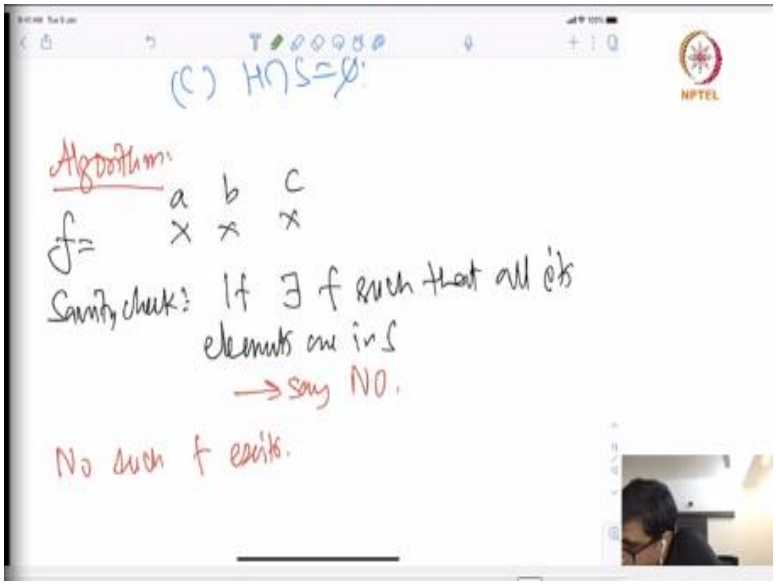
I mean I do not care about Y, but now given a family F I will make new family F tilde, remove every set that intersects Y. So, every set that any element of Y is being able to take care of you delete it. So, you only keep those sets in F tilde who needs to be hit, that is it, so this is how you can get a compression disjoint version of the problem for 3 hitting set. So, it is not like graph but it is more or less like a graph, so what is the disjoint version of the problem we will get?

(Refer Slide Time: 43:44)



We will get input U, F, k, S, mod S is at most sum k + 1 and S is a hitting set, my parameter is k. Question; does there exist a hitting set of size at most k? Rather it is a hitting set H such that a k, b H is a hitting set, c H intersection S is empty set. So, now let us look at what happens to this problem, here is my algorithm, very simple algorithm.

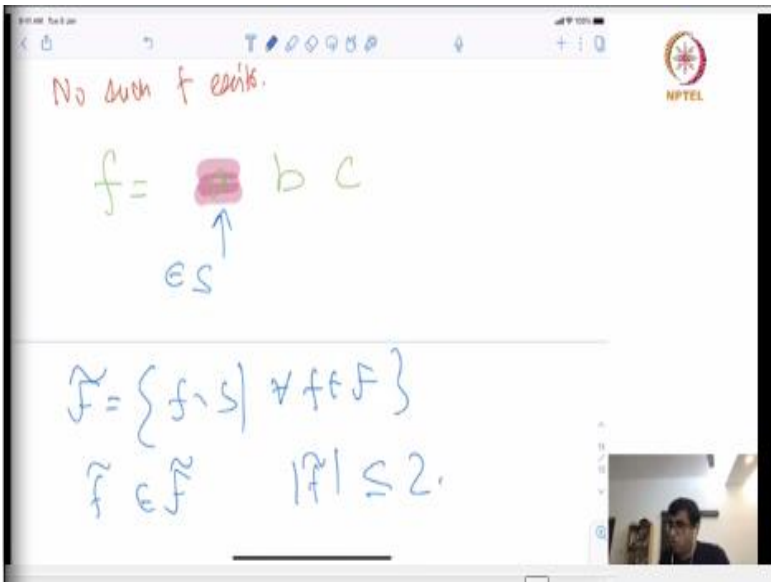
(Refer Slide Time: 44:52)



I said fine, so look at a set, this is my some family F, there is an a, there is an b and there is a c, what is our sanity check algorithm? What is our sanity check? Sanity check is if that exists f such that all it is elements are in S say no. Because no way we can hit this f, no way we can hit this family because there gives an f where all the elements are forbidden to be part of my solution.

So, no such F exists, but then let us looks at what is the property of such? So, here is my property.

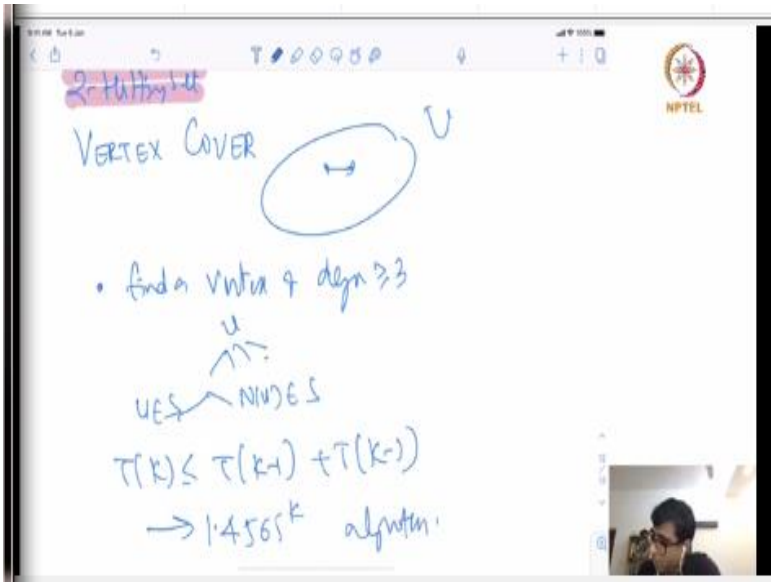
(Refer Slide Time: 46:07)



So, now look at this is my f there is some a, and b, and c what I know? At least one element of this belongs to say S, where does it belongs to? Because my S is my hitting set, this must belong to S, at least one maybe more but at least one element definitely belongs to S. So, now I am going to create a family F tilde, what is my new family F tilde? It is f - S for all f in F and what is the property of this?

So, now each f i tilde in F or each F tilde each, what is the property? The size of f tilde is at most 2, so what has happened? So, we were looking for a 3 hitting set problem but in the compression step we have got down to an algorithm, got down to a problem where each set has size at most 2. Because look whatever hitting set you are going to construct it must hit the elements which are not allowed to be hit, like which are allowed to be hit, not, so what you do? You construct a different family where the arity or the size of each set decreases by 1, now what is this problem?

(Refer Slide Time: 47:35)



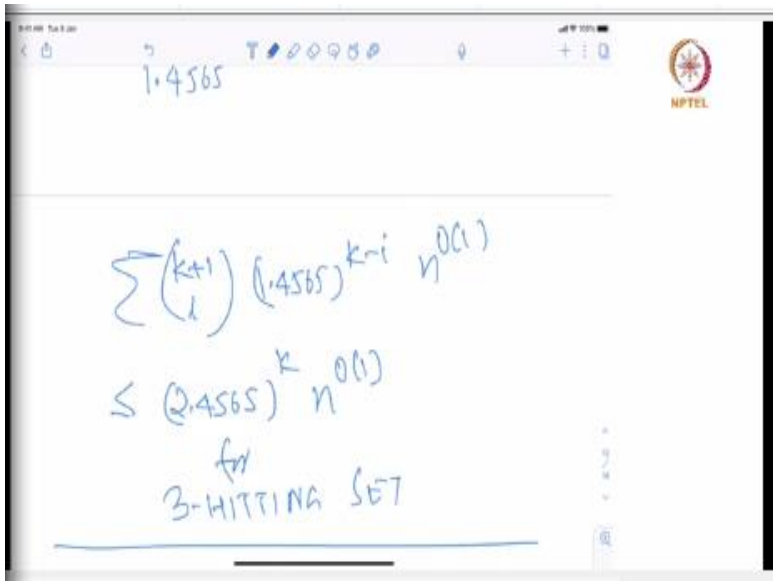
This is nothing but a 2 hitting set problem and for 2 hitting set problem what do we know? This is same as vertex cover in fact, so of course if there is a set of size 1, you definitely have to put them into solution move them off, so each set has size exactly equal to 2. Then you can think of a graph where a graph on U and each set will corresponds to an edge, so this is exactly vertex cover problem.

And for vertex cover you must have seen a very good branching algorithm in your lecture on branching and what are that algorithm? At least one algorithm I can tell you, find a vertex or of degree greater equal to 3, so if you have a vertex of degree greater than equal to 3, U. In one branch you include U in the solution in the other branch you include N U in the solution leading to the following recurrence T of k is less than equal to T of k - 1 + T of k - 3 leading to 1.456 to the power 5 k algorithm.

(Refer Slide Time: 48:57)

And when there is no vertex of degree 3, so or a maximum degree of the graph is at most 2, solve the problem in polynomial time. So, now we know such a good algorithm for vertex cover, so the vertex cover algorithm we know is 1.4565 to the power k. So, now if I do my computation, so what is the disjoint algorithm will take? It is going to be summation k + 1 choose i, 1.456 to the power 5 k – i.

(Refer Slide Time: 49:40)



And if you do the computation as we have done before, we have got an algorithm with running time roughly n to the power O of 1 for 3 hitting set. So, we have got such an algorithm, so we started with a 3 to the power k algorithm but in the compression step we reduced this problem to a 2 hitting set problem. And we employed the fast algorithm for 2 hitting set to get a fast

algorithm for 3 hitting set. So, I think that all that I would like to talk about in this lecture and we might have one more lecture on iterative compression.