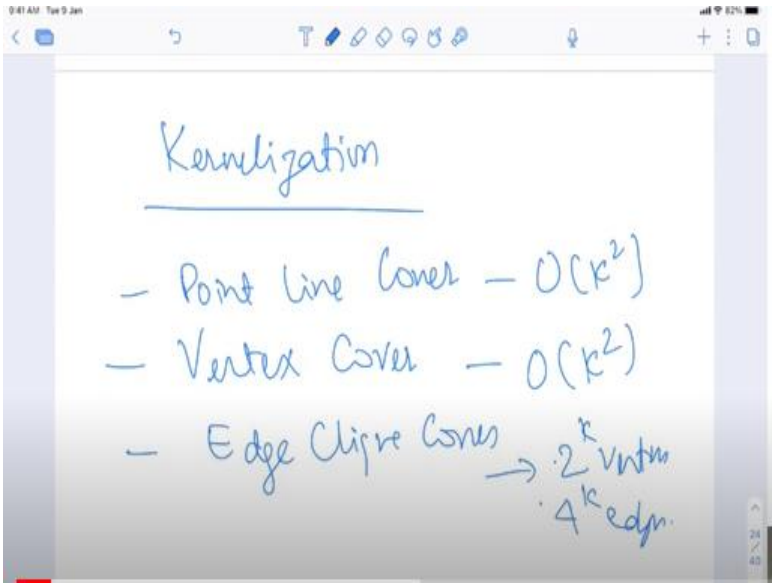
Parameterized Algorithms
Neeldhara Misra – Saket Saurabh
Department of Electrical Engineering
Indian Institute of Technology – Gandhinagar

Lecture - 04
Kernelization: d-Hitting Set

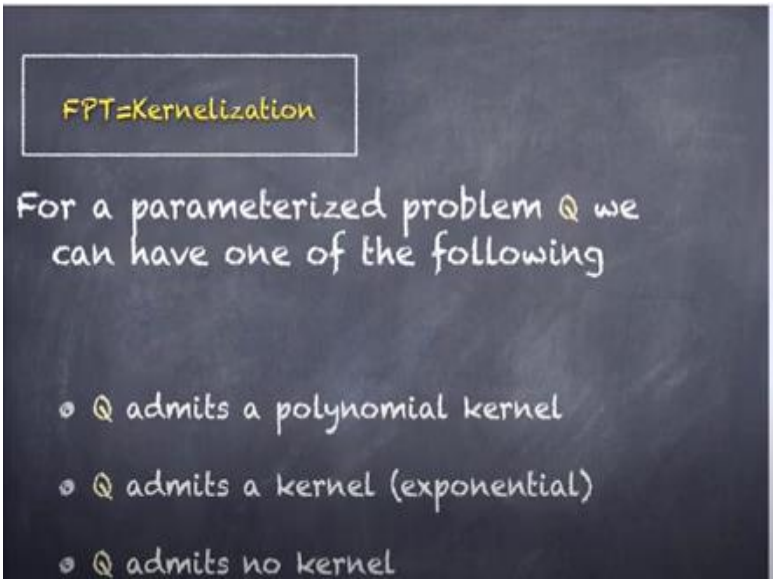
(Refer Slide Time: 00:26)



So, let us start where we left in the previous lecture, where, if you recall, we defined kernelization and gave a kernel for point line cover, vertex cover and we also gave kernel for edge clique cover. This were like order k square points, order k square vertices and edges and this was like 2 power k vertices. And so, since there only n square edges, so, it will have at most 4 power kedges.

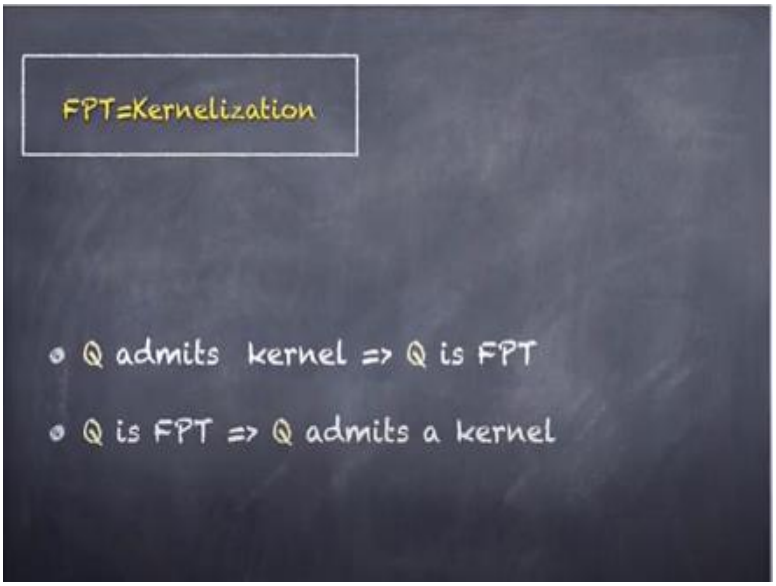
So, notice this first 2 kernels has polynomial bound but the last kernel is exponential. And so, a natural question arises is that; is there a polynomial kernel for this problem? If not, can be proved that no such kernel exists. So, we will see such methods later in the course to show that kernels of certain kind do not exist. But for now, we will continue our foray into discovering techniques to design polynomial time kernels.

(Refer Slide Time: 01:42)



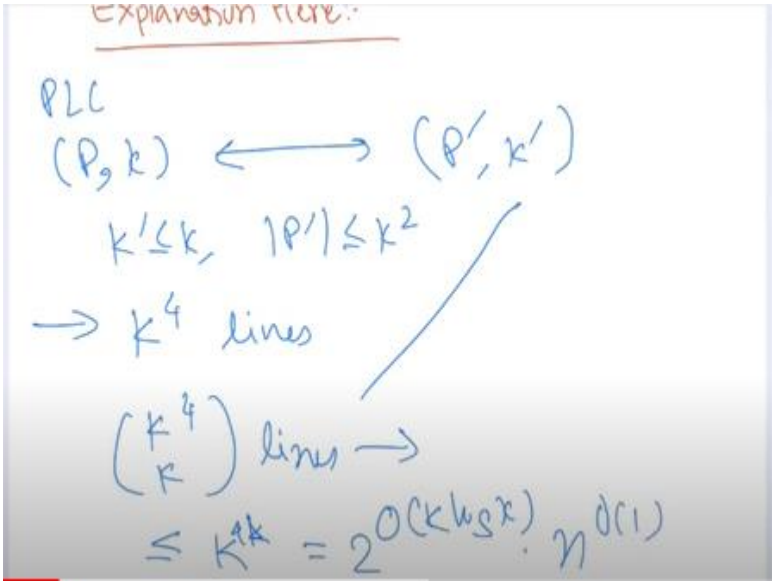
So, let us there is a very well known notion about is FPT same as kernelization. And let us see if we can say any statement like this. So, for a parameterised problem Q, we can have one of the following things. Q admits a polynomial kernel, this could be one statement. Q admits a kernel may be exponential like this is like vertex cover or point line cover, admits a kernel like exponential like edge clique cover or Q admits no kernel.

(Refer Slide Time: 02:25)



Now notice, so, it is very natural right to say that the moment Q admits a kernel Q is fixed parameter tractable right. Because look at why we say this, suppose look at a vertex cover example.

(Refer Slide Time: 02:35)

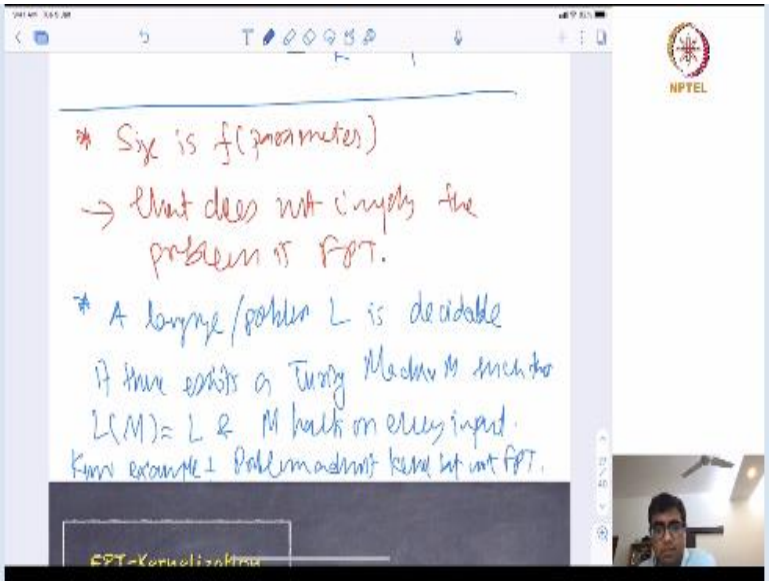


So, what did you do? Given or say point line cover example. So, in point line cover, given a set of points P, k in polynomial time, we got an equivalent instance P prime k prime, where in fact k prime was less than equal to k and P prime was less than equal to k square. Now, to check whether there exists k lines that covers all the points; we can, we know that how many number of lines that we need to cover, need to consider?

The total number of lines that we need to consider are like at most k power 4 lines. And from this k power 4 lines, we need to choose k power 4 choose k lines and check whether they cover all the edges, all the points or not. So, how many such possibilities like this. So, k power 4 choose k is like roughly k to the power 4 k, which is like 2to the power big of O of k log k times n to the power big O of 1 to n to the power big O of 1 to get the kernel and to check all these things. So, we do get a kernel.

So, once we have a kernel, it seems that an obvious way of trying this answers is good, because size of the instance itself is bounded by some f of k.

(Refer Slide Time: 04:12)



So, it seems very obvious that if Q admits a kernel implies Q is FPT. But this may not be true all the time because this is something which if you do not understand, it is perfectly fine. You can, size is some function of parameter that does not imply the problem is FPT. Why? Because I may not have an algorithm to even make a like decide with a yes instance. So, basically what we need at this point of time is about problem to be decidable.

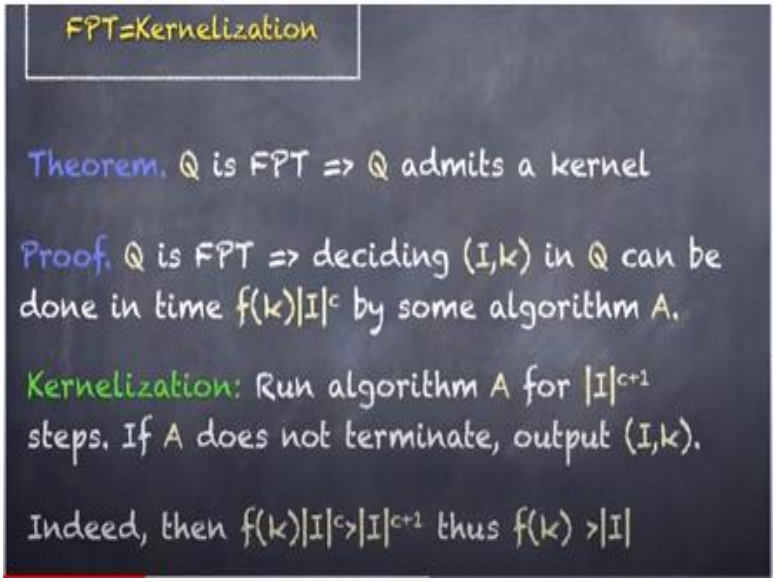
So, we say this is a statement from world of computability that a language problem whatever you want to call it, L is decidable if there exists a turing machine M such that language accepted by turing machine is exactly equal to L and M halts on every input, which could be implies that look a language a problem is decidable, if there is a turing machine like on every input, it will tell you yes or no.

If we have such an algorithm, then basically, Q admits a kernel imply Q is FPT is basically said well if Q admits a kernel. And there is some algorithm for Q, then Q is FPT. So, this is an important point to talk about. And every kernel or every problem that we will consider in this course and in particular, every NP complete problem had such an algorithm. So, if you are dealing with NP complete problem, do not worry at all. They are decidable by definition. But there are problems which are not decidable.

So, I will not go into any detail here. But if you have any, if you like to know example, the object problem admits a kernel but no algorithm but not FPT, ask me by sending an email or some such thing let us not confuse readers with this. But all I want to tell you that do remember that if a problem admits a kernel and it does have an algorithm, then it does imply

the problem is FPT because it just all that it means that you know size is any bounded, you now run the algorithm to make a decision about.

(Refer Slide Time: 07:14)



But what about the other well. So, Q is FPT implies Q admits a kernel, what about this? So, this is nice that even the reverse if Q is FPT, then actually Q admits a kernel. Why? So, suppose Q is FPT implies, you can decide whether I, k belongs to Q in time, some f of k I like instant by say, by some algorithm A. But remember, this was a, this is by FPT is computable, this f by definition is a computable function, f is a computable function in this turing machine which can given k, you can write f of k.

So, what the kernelization algorithm does? If run this algorithm A for run this algorithm A for mod I to the power c + 1 steps. So, either this algorithm just saw like ends by then and tells me the answer. So, I have run the algorithm for mod I to the power c + 1 steps. If it does not terminate, then I will say hey, I, k is your kernel. So, it is a very trivial algorithm. In the following sense what I do?

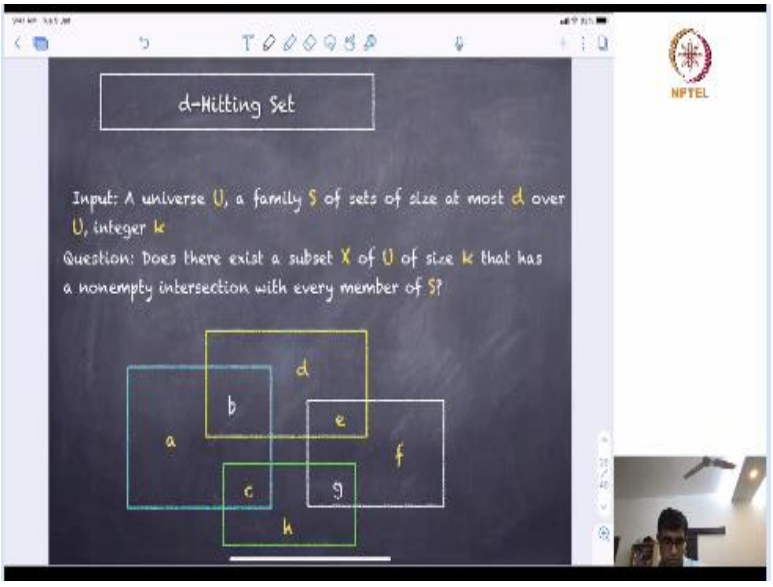
I run my, suppose the instance size I is equal to n, so, I run this algorithm for n to the power c + 1 steps. If it terminates, then we are happy because we have found the solution. I returned to that as a solution. Or, if it does not terminate, it runs beyond this, then I returned the instances. And this is, for this termination point that we needed to know whether f is computable or not, but beside that point, so, what happens? It means, if it ran for more than this step, it means, let us see what happens?

It means f of k mod I to the power c is greater equal to mod I c + 1 which implies that mod I is less than f of k. So, you have indeed got a kernel. But how interesting is that a kernel is a different thing. So, which implies that this is why in this if you talk about decidability decidable problem, then the notion of FPT and kernelization are same. So, we could not defined FPT in terms of kernelization also for decidable.

So, but the kernel which comes from this theorem is basically the function of the parameterised algorithm. And that is not what it is all about. So, it is all about that how smaller kernel can we get, if we just allow ourselves to polynomial time. For example, for vertex cover if we want to solve an FPT algorithm, if you want to design an FPT algorithm, we cannot do better than c power k modulo some conjectures. But we are able to get a kernel with order k square vertices and edges.

So, in polynomial time, we can do much more than just getting a kernel with exponential size. So, the field is interesting, also because that our main goal is to determine which problem admit polynomial time kernel, which problem does not admit polynomial time kernels. And can we show these things? How small kernel we can get and things of this nature? Let us moving ahead.

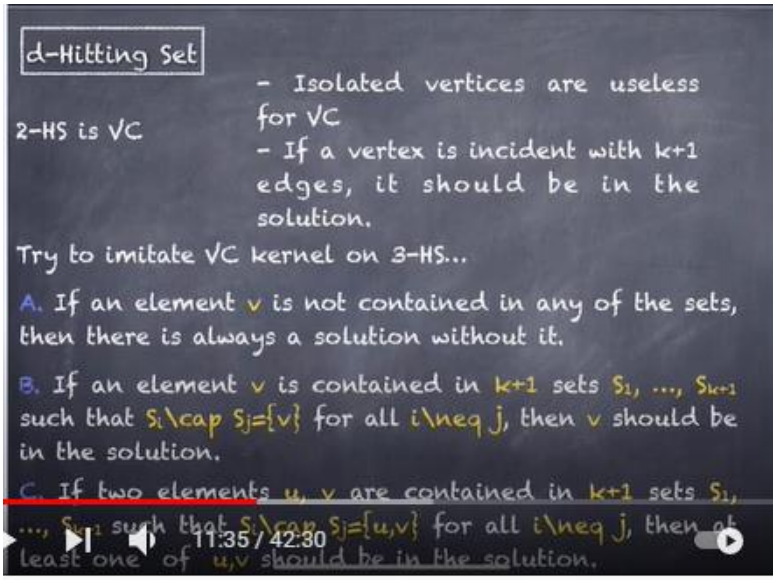
(Refer Slide Time: 10:31)



Let me give you another example of kernel problem. So, my input is an universe U, d-hitting set, a family S of size at most d over U, integer k. And our question is: does there exist a subset X of U of size k that has a non empty intersection with every member of S? So, basically, you are looking for a set a subset X of U whose size is k and it intersects every set.

So, this is a very classical hitting set problem and it is d-hitting set because every set in the family is upper bounded by d. So, for example, this could be your set family, yellow, blue, white, green. And they are intersecting things among themselves. So, we would like to make a kernel for d-hitting set, but I will, okay. But I will instantiate. So, we would like to get a kernel for d-hitting set.

(Refer Slide Time: 11:30)

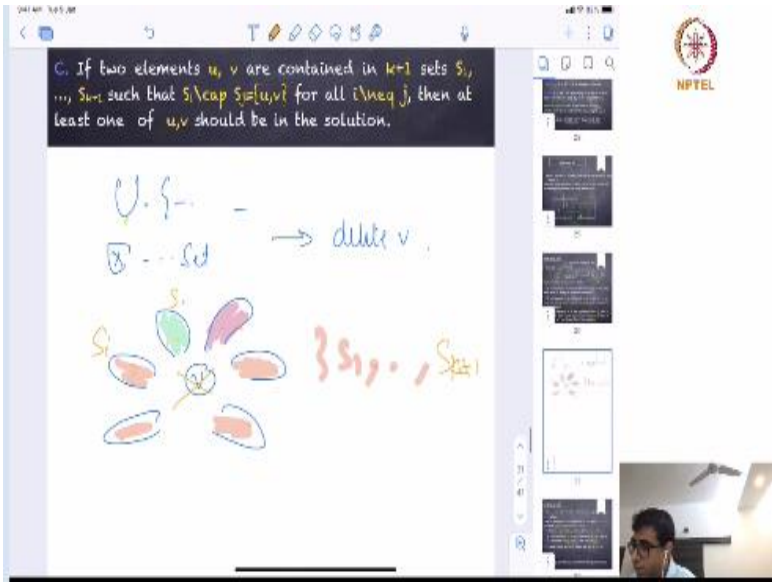


So, before we go to d-hitting set, let us observe that 2 hitting set is nothing but a vertex cover. If every set has size 2, then this is a vertex cover. And what was the kernel for the vertex cover? If you recall there were just 2 reduction rule, isolated vertices were useless for vertex cover, right. Isolated vertex were useless for vertex cover. So, we deleted them or let say, okay.

And if a vertex is incident with k + 1 either if you recall correctly, if the vertex will incident to some k + 1 edges, we included V in the vertex cover and decrease the parameter by 1. Because every vertex of this should be in the solution. Now, what we try to do is like try to mimic this kernel for vertex cover of a 3 hittings. So, now let us assume that d is upper bounded by 3.

So, what is this? If an element V is not contained in any of the sets, then there is always a solution without it. It is same like isolated vertices. What are that? If a vertex v, so let us we are making some observation. So, what is our observation? Let us try to add a page maybe that will be helpful.

(Refer Slide Time: 13:13)



So, what is. If an element V is not contained in any set, so, an element, is present in the universe, but this element does not occur in any set. So, it is same as saying that a vertex is not part of any edge. So, it is an isolated vertex, then we can delete V without changing the answer. Now, notice, if an element V is contained in k + 1 set S 1 to S k, so, basically, suppose you have a sets, here is V and the sets do not intersect, but at v. So, if you look at this part, look at this, they are disjoint, so on and so forth, so these are like disjoint. So, this is what it means?

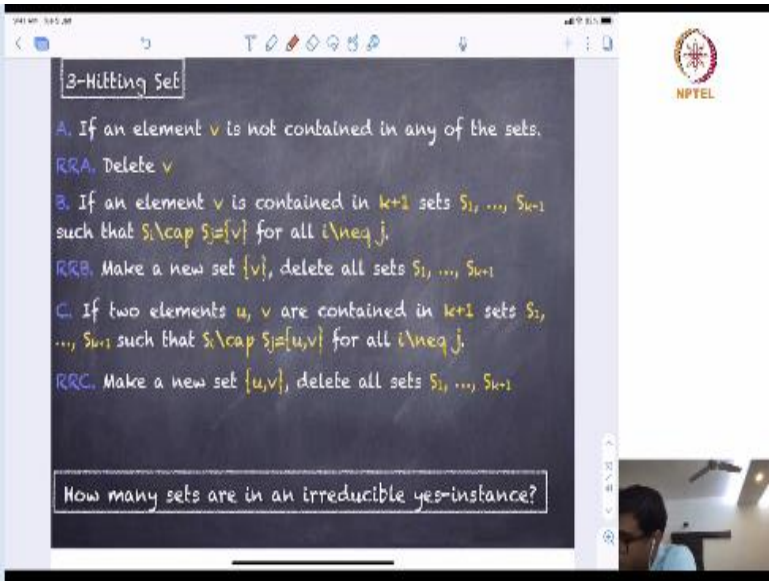
If an element V is contained in k + 1 sets S 1 to S k + 1, so, they are like V is presented S 1 to S k + 1 sets. And S i intersection S j is exactly equal to V for. So, if you look at any set S i and S j, their intersection is exactly V. So, if you look at any pair by sets, they only intersect at V. So, now what happens if you do not pick? In fact, if you do not pick V in the solution, then notice there are like k + 1, then there are like k + 1 pairwise disjoint sets. Who will hit them?

Because to hit this k + 1 pairwise disjoint set, you at least need k + 1 points from U. So, in that implies that we must be included in the solution. So, this is same as a vertex cover analogues. If they just vertex of high degree of degree k + 1, then we must belong to the solution. If you famously, if V is part of k + 1 set, where V is the only set which is common to all this.

Then like for you take any 2 pair by set, V is the only common element between them basically the look like the picture which I have drawn, then V must belong to my solution. Now, what can you say that? Look, I cannot say about V, but I can actually find 2 elements UV and now, I have an extension of this k + 1 sets. So, if I delete UV, then all these sets are pairwise disjoint. Then what happens?

If you do not pick 1 of U and V, then you again need k + 1 elements to cover all these sets. So, it implies that for all i not equal, then at least 1 of UV should be in the solution. So, now let us formalise them this intuition into the reduction rule and it is done as follows. So, what are my reduction rule?

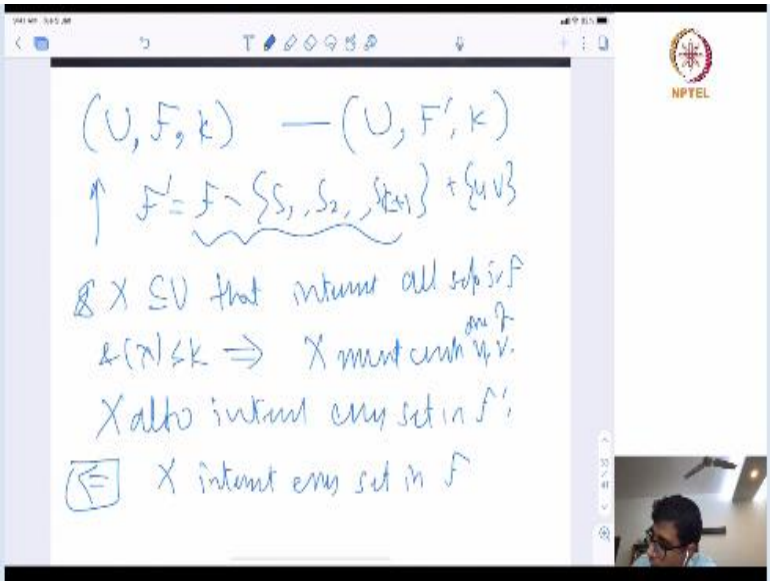
(Refer Slide Time: 13:13)



If an element V is not contained in any of the set so, reduction rule is applies delete V. If an element V is contained in k + 1 set S 1 to S k + 1 says that S i intersection S j is V for all i not equal to G, then what is our reduction rule ; make a new set v delete all set S 1 to S k + 1. Why? Because, look at the forward direction. You know that V go into the solution, then V and every other one else, I will formally prove it in a minute.

Or, you can prove it yourself for this, will contain this and in the backward direction because you have kept the set single set V like single element set V, the reduced instance must pick V the moment to pick V, S 1 to S k + 1 will be hit in the original instance. And if 2 elements UV are contained in the sets k + 1 set such that this, then what you did? So, look at this. What happened?

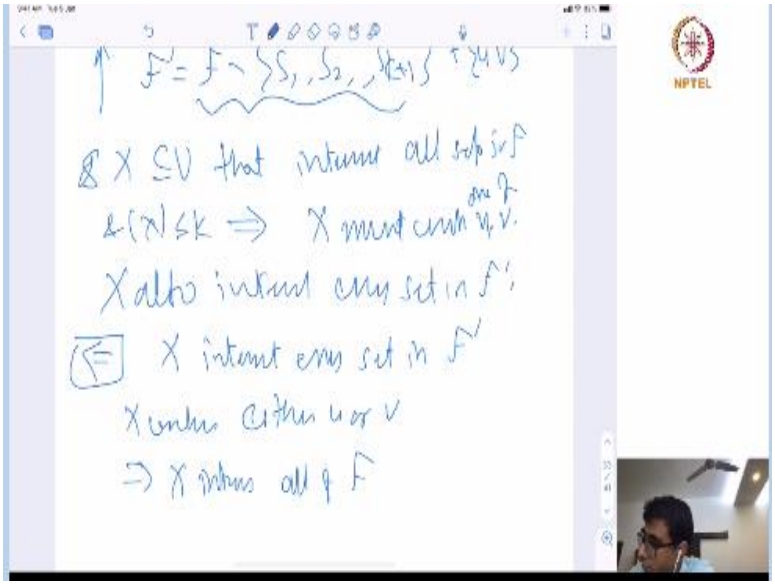
(Refer Slide Time: 17:21)



So, you have U family, what you did here? U in this, you did a U and F prime is F – S 1, S 2, S k + 1 and but you added set + UV and this is what we did. So, this is, I returned F prime, k. Now, let us see, what happens. So, forward direction, you get a forward direction well, in the forward direction, you know that if the register set x subset of U that intersects all sets in F and mod x is less than or equal to k implies x must contain U and V. Then I claim that x also intersects every set in F prime.

Why? Because look, F prime does not, F prime has all the sets like which F contains except UV and x is the same then you have taken care and now since x contains U or one of the V, contains one of U, V, x also intersects every sets.

(Refer Slide Time: 18:56)

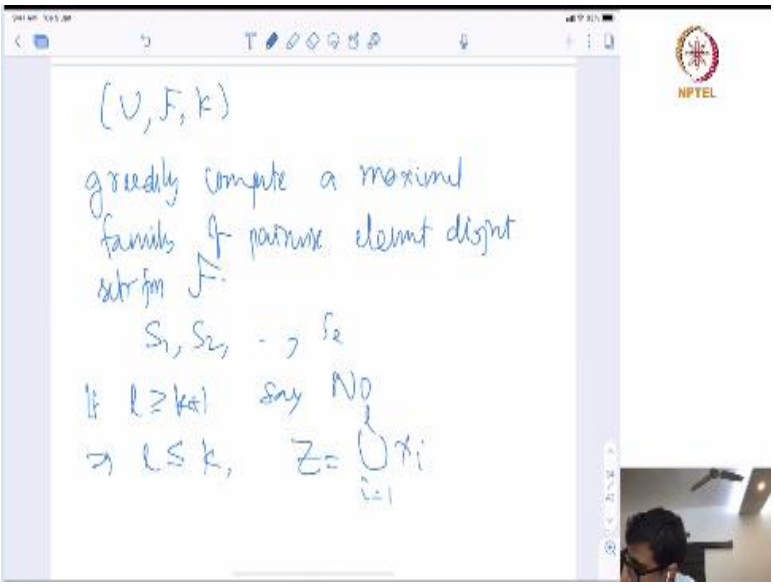


Reverse direction is also obvious, x intercepts every set in F prime implies x contains either U or V implies x intersects all of F. Because, why? Because look here, what is the difference between F prime and F. Well, F prime contains all the sets of F but S 1 to S k + 1 and it contains UV. Now, because F prime contains UV, you know that one of U and V are must be taken in my x but then that U takes care of S 1 to S k + 1.

Or, that we will take care of S 1 to S k + 1 and the remaining element of x will take care of other sets of it. So, this is very simple reduction rule and we can prove its correctness immediately. So, but how many sets are in an irreducible yes instance because how are they obtaining a kernel. We are obtaining a kernel by the following procedure. We apply some reduction rule.

And we say when we are reduction rules are no longer applicable and if this is a yes instance, then the number of sets, number of objects are bounded because if they are not bounded, then I can say no and then we are correct. So, let us try to do the same analysis for the problem okay.

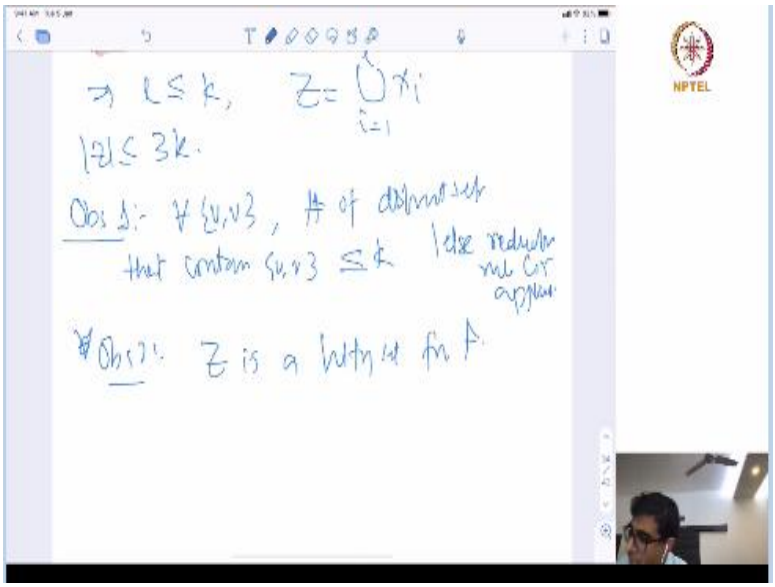
(Refer Slide Time: 20:33)



So, to do the analysis, I am going to do the following. I have U, F, k. Greedily compute a maximal family of pairwise element disjoint sets from F. So, we have greatly computer support this is like S 1, S 2 dot dot dot S l. First of all, if l is greater than equal to k + 1, say no right because now I have k + 1 element disjoint sets and you want to hit all the sets with just k elements, then for this k + 1 set itself you need k + 1 element so, **(()) (21:30)**.

So, we can assume that l is at most k. So, now if I take say, let us not take call it x, let us call it z which is union of x i, i going from 1 to l right.

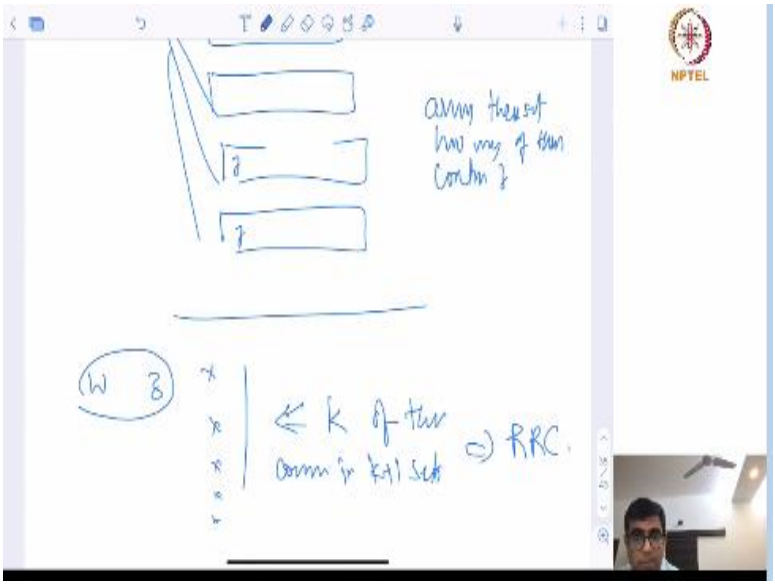
(Refer Slide Time: 21:47)



Then size of z is at most 3 times k. Now, observation 1 and suppose this is a yes instance. So, if it is yes instance, l is always less than equal to k. Now, look at observation 1. What is this? For every element U, V, number of distinct sets that contain UV is upper bounded by k. Otherwise, we would have applied the reduction rule C. This is. Else reduction rule C is applicable. I am going to claim for; first of all, what is a observation 2? z is a hitting set for F.

Why? Because a maximal element disjoint sets from F. So, if you cannot add any sets, why? Because this set intersects 1 of the sets previously selected set. This is so generating set of F of size at most 3k.

(Refer Slide Time: 23:44)

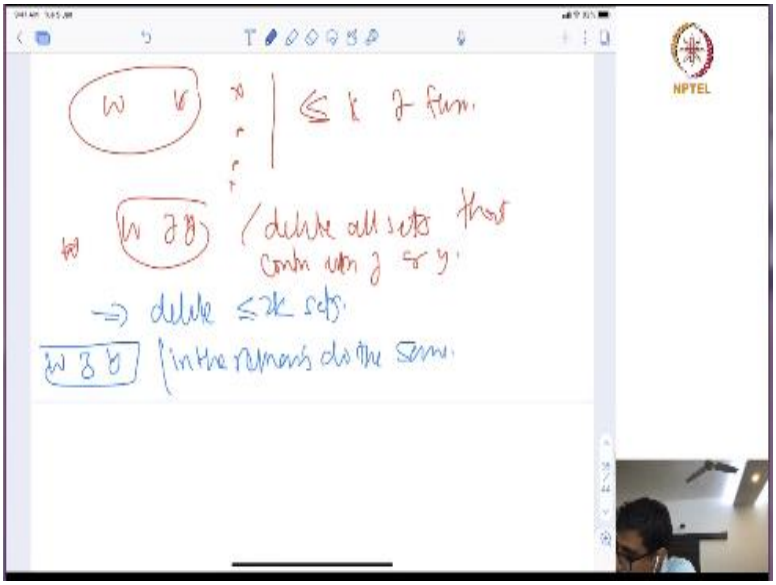


Now, this is our main claim. Every element injured intersects at most some, we will fix some k square, some question mark, question number Mark k sets. So, basically some order k square sets. Let us see why. Because once we are done then we will be happy. So, why? Let us fix element w. This is injured and suppose, it participates into lots of sets. So, suppose it participates into more than some k square plus some set.

We will see what happens. So, what is a common about these things? w is inside each of this set. Now, let us pick up the first set here. I pick up an element here z. I pick up an element here z. And I asked myself among these sets. Among the sets, among these sets, how how many of them contains z? Let us ask ourselves suppose z was here, z was here, z was here and z was here. So, let us pick up.

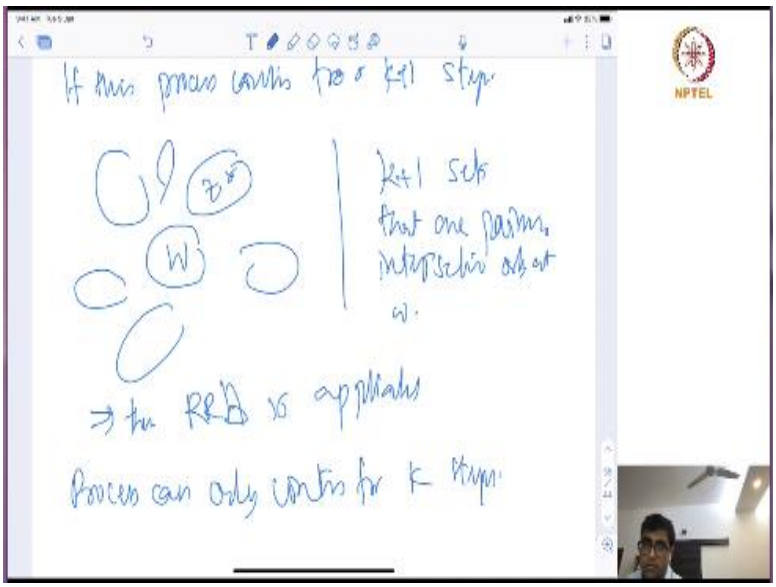
So, first of all we picked up w. Now, we picked up z and be picked up all the sets with z contains. Well, this can be at most k of them. Otherwise, w z is common with k + 1 sets which implies reduction rule C is applicable. Okay? Now, so z appears. Now, let us ask ourselves how many sets suppose another element y, How many sets contains y?

(Refer Slide Time: 26:24)



Same way we can say hey, look at w, y and look at all the sets it contains. It is also at most k of them. Okay, great. So, now, what I am going to do, I am going to pick up z, y. I am going to pick up this set and delete all the sets. So, what I do? I pick up the set, I pick up the set w z y. And now, delete all sets that contain either z or y, how many sets will be deleted such? By this process, we will delete at most 2 sets. So, but now, we have got a set w z y. In the remaining set, do the same. What will we get?

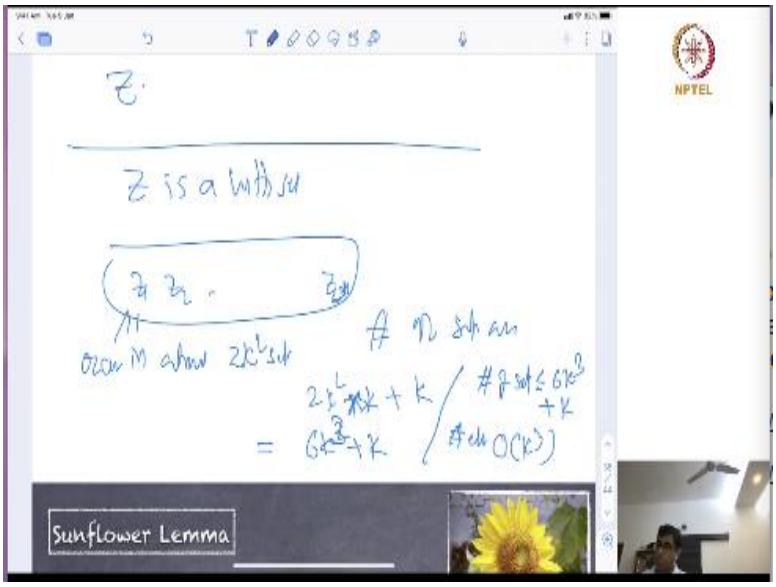
(Refer Slide Time: 27:32)



So, if this process continues for k + 1 steps, what do we get? We get w z y, some other set, some other set, some other set, some other set, we get k + 1 sets that are pairwise intersecting only at w. But then reduction rule B is applicable. W, you should have a set containing just w which implies that this process can only continue for k steps which implies that each step you knock out 2k sets.

In each steps you can, so, basically you will get, if you get like, you cannot have more than 2k square because, in each step you knock out 2k. So, you cannot have more than 2k square set that contains an element of z. So let us go back. So, 2k square, is the right **(()) (29:19)**. We did not know what it is. Now, we have got it 2k square okay, set at most 2k square.

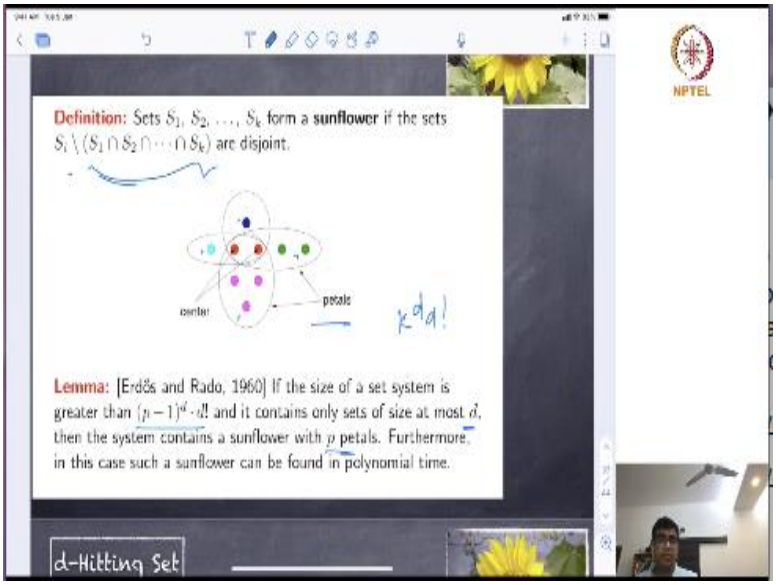
(Refer Slide Time: 29:38)



Now, what we know about z? z is a hitting set. And so, like z 1, z 2, z at most 3k. Each element occurs in at most 2k square sets. So, the total number of sets are 2k square plus into 3k plus the sets from z 1 to z t k which was like at most k, so, which is like into 3k. So, this is like 6k square + k. So, total number of sets are at most 6k square 6k cube, there is k cube. Number of sets is at most 6k cube plus some roughly k.

So, the number of elements will be like you multiply with some 3 (()) (30:57). So, this is how you can get a kernel for 3 hitting set, just generalising the idea of vertex cover. And in fact, this follows from which I did not teach you, the notion of sunflower.

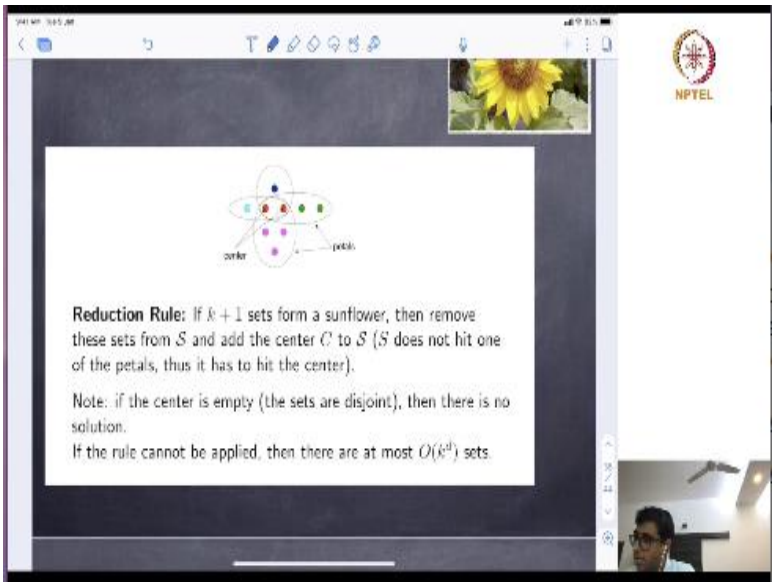
(Refer Slide Time: 31:08)



So, what is sunflower? Your sets S 1 to S k form of sunflower. if the sets S i – S y look at, so, they just have one common intersection otherwise pairwise disjoint. So, S i – their common intersection are disjoint. So, this is called centred and actually these things if you delete the centre or delete the common intersection, whatever these are called petals.

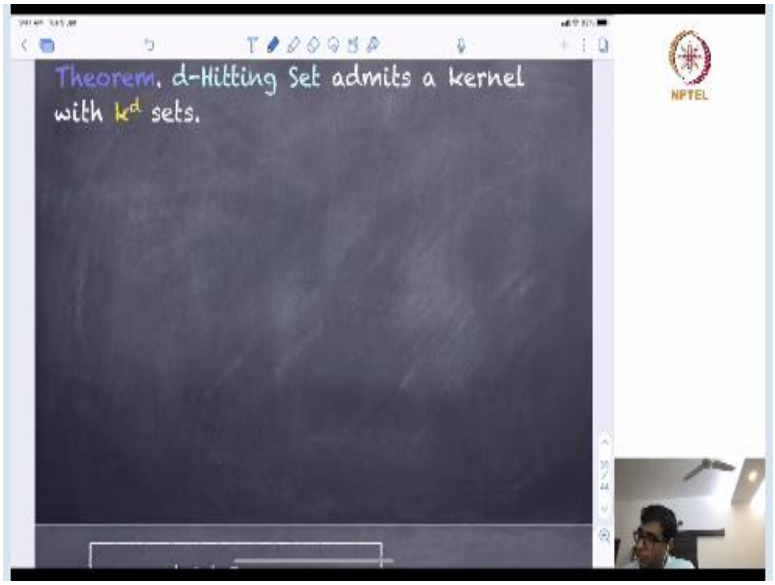
So, in as early as 1960 Erdos and Rado prove the following lemma. If the size of system is greater than p – 1 to the power d times d factorial and it only contains set of size at most d, then the system contains a sunflower with p petals. So, in our case, we applied this with p equal to k + 1. So, if you have more than k power d, d factorial sets then you will contain a sunflower with k + 1 petals. And furthermore, in this case, you can find such a sunflower in polynomial time.

(Refer Slide Time: 32:15)

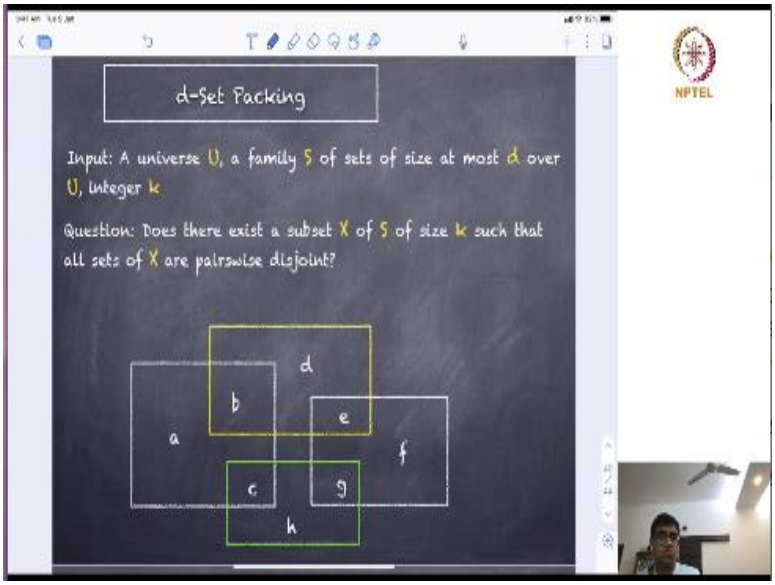


So, how do we get a d-set? If k + 1 sets form the sunflower, then remove the sets from S and add the centred C to S, does not hit one of the petals, thus it has to hit the centre. Note if the centre is empty, the sets are disjoint, then there is no solution. So, if the rule cannot be applied, then we know that number of sets is upper bounded by k power d and you can get a kernel for d-hitting set applying this classical sunflower. And the proof is of this lemma is not very hard and you can find in the textbook of the course.

(Refer Slide Time: 32:47)



(Refer Slide Time: 32:53)

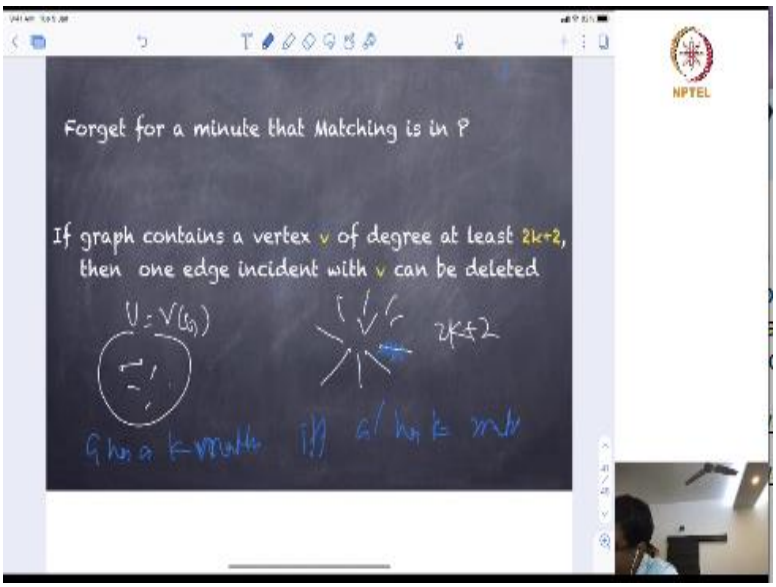


So, d-hitting set admits a kernel with k power d-sets. So, before I end my lecture today, for, let me give you another problem which is very classical and which has, it is called d-set packing. So, input is universe, a family S of sets of size at most d over U and integer k. And

the question is: does there is a subset x of S of size k so, that all sets of x appeared while disjoint.

So, here for example, abc and efg are pairwise disjoint. Similarly cgh and bde are pairwise disjoint and there are only 2 pairwise disjoint sets. So, in the d-set packing, our goal is that can I find cases that are pairwise disjoint?

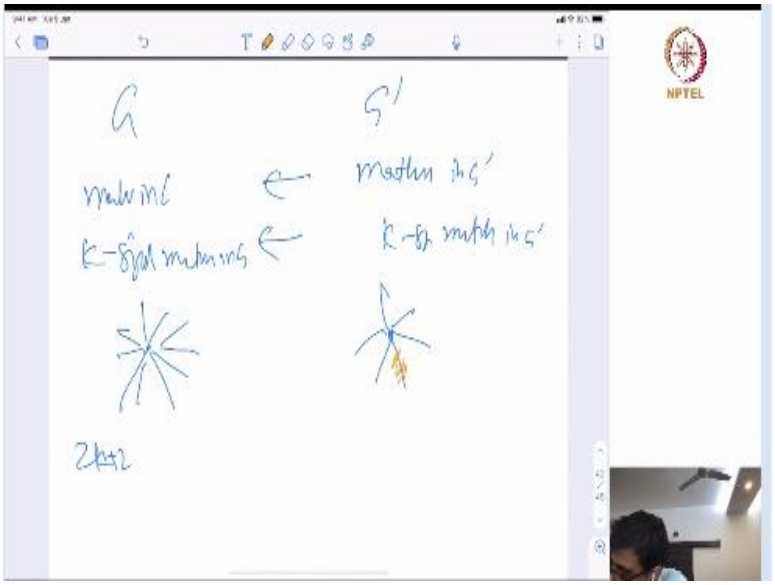
(Refer Slide Time: 33:27)



So, I will just give you up. Let us look at 2 set pack. A 2 set packing is nothing but you want to find 2 sets which are pairwise disjoint. And basically if you think of 2 sets, then they are like a graph. Each you think of universe as vertex set, think of universe as vertex set and universe and for each 2 sizes, you just put an edge. So, this is basically a graph and 2 set packing is nothing but finding a maximum matching in a graph which you can do in polynomial but forget for now.

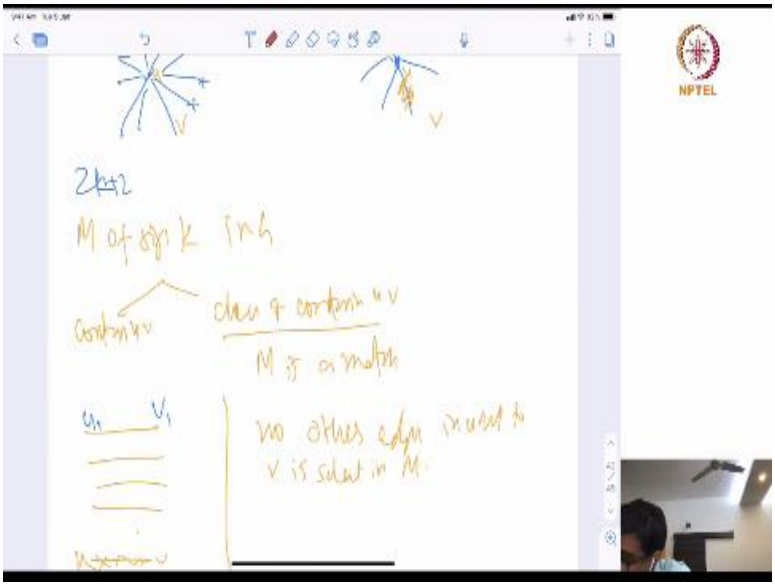
Notice that if I have a vertex V whose degree is say 2k + 2, then I claim to you that you know, you can forget one edges, you can delete one edges insert into V. And now, we can show that G has a k matching if and only if G prime has k matching. So, look at the forward direction, backward direction is very easy. So, the backward direction is very easy. Why?

(Refer Slide Time: 34:49)



Because backward direction is very easy because here is a G, here is G prime. So, any matching in G prime is a matching in G. So, if you gave me a k sized matching in G prime, I can get k sized matching in G, but whatever the forward direction that is a interesting point. The point is look at forward direction. We have 2k + 2 edges. And what happens is that in G prime, some edges missing, this is important.

(Refer Slide Time: 35:35)

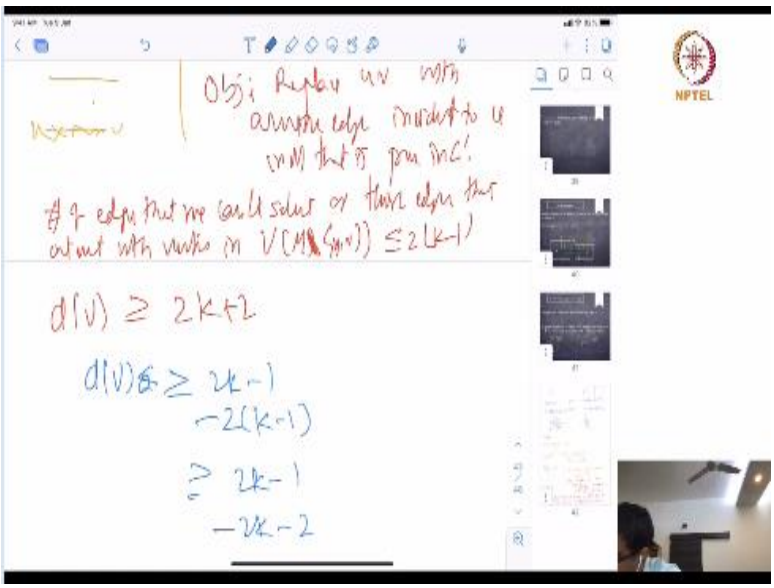


So, you let us look at a matching M of size k in G. Let us suppose, this is edge UV; this is edge UV. Here 2 things; contains UV; does not contain UV. This is easy, does not contain a VM and then M is a matching, fine. But now it contains UV. If it contains UV, then what happens? If M contains UV, look at this UV. Let us delete this UV. Now, because you delete UV, what happens?

Now, I need to find because it contained edge UV, it contained an edge UV because it contained an edge UV, you know that no other edge, no other edge incident to V is selected in M, fine. Now, let us look at this U 1, V 1. Okay, maybe this edge and this is edge, I cannot. This edge and this is, I cannot, I would like to what I do, I would like to replace in the M UV edge by some other edge which is present in G prime that is my goal.

But unfortunately, this edge is, so let us look at the C 1, V 1. So, that is our objective is to replace UV with some other edge.

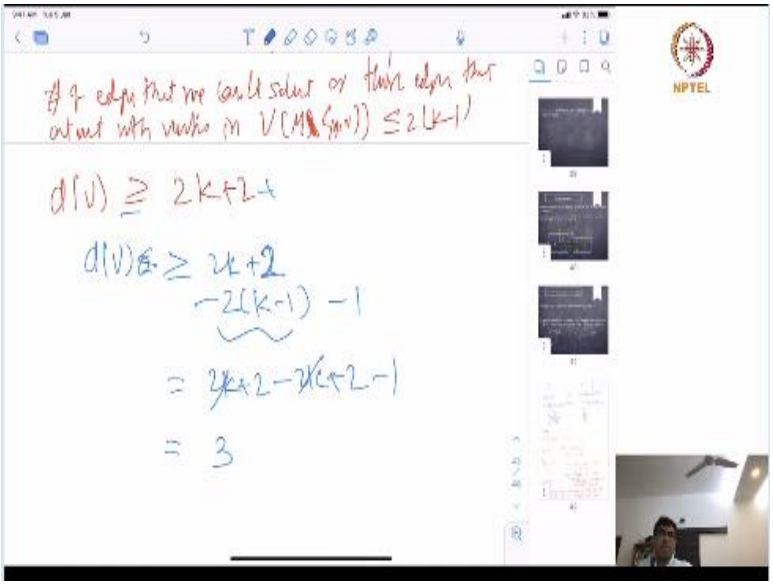
(Refer Slide Time: 37:57)



Objective: replace UV with another edge incident to U in M that is present in G prime. Okay, so, let us ask ourselves, what about this edge? Oh, I cannot pick this edge because oh, because there is some vertex, some edge which is incident to this vertex, fine. So, an edge can only kill 2 edges, some other present edges can only kill 2 edges incident to U. So, by doing this Okay, this cannot be done, this cannot be done, this cannot be done but how many number of edges that we cannot select.

Or, those edges that intersect with vertices in V of M – UV is at most 2 times k – 1. But what was a degree of A? Degree of V is greater than equal to 2k + 2. So, after I deleted 1 edge, a degree of V is at least 2k – 1. Now, I deleted 2 into k – 1. So, degree of V is at least say, let us say, 2k – 1 – 2k. No, it was 2k + 2. I deleted this. So, let us see how many edges are.

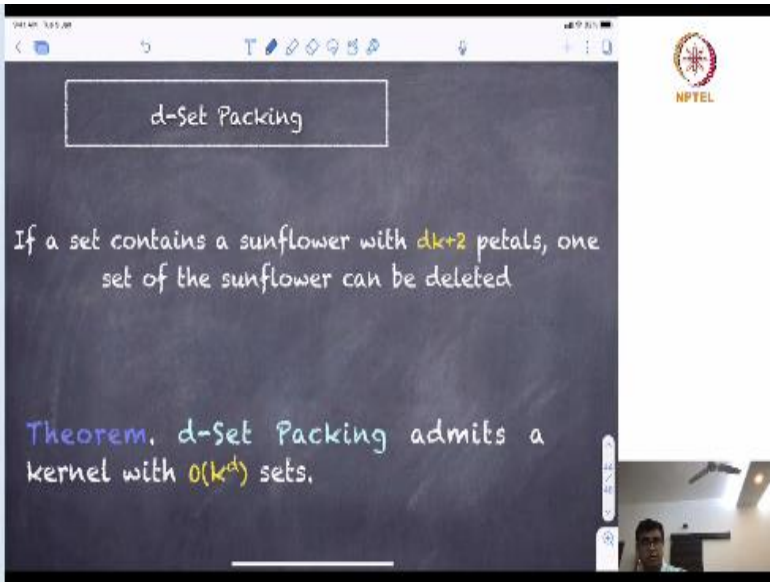
(Refer Slide Time: 39:52)



So, maybe, we need at least 2k + 2, so say this my bad, say degree of V at least 2k + 2 + 1. So, even after I deleted, so maybe we should do that is my bad, this is something. So, we started with 2k + 2 edges, 1 edge, we are not allowed to pick because of UV. And these many edges are deducted because of the, so, let us see, 2k + 2 – 2k minus, minus plus 2, – 1, which is so, 3.

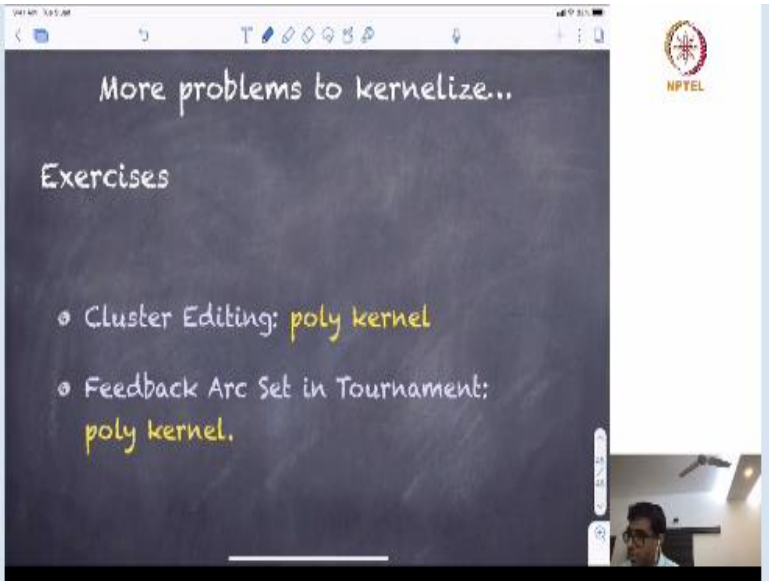
So, we still have a lot many more edges left at V that we could replace that edge of M with this particular edge and we can get. So, basically what it tells us that if you are looking for to preserve matching of size k and if a vertex has very high degree like to 2 k order k plus order k, like 2k plus 2, then you can do something interesting. Like you can delete any.

(Refer Slide Time: 41:36)



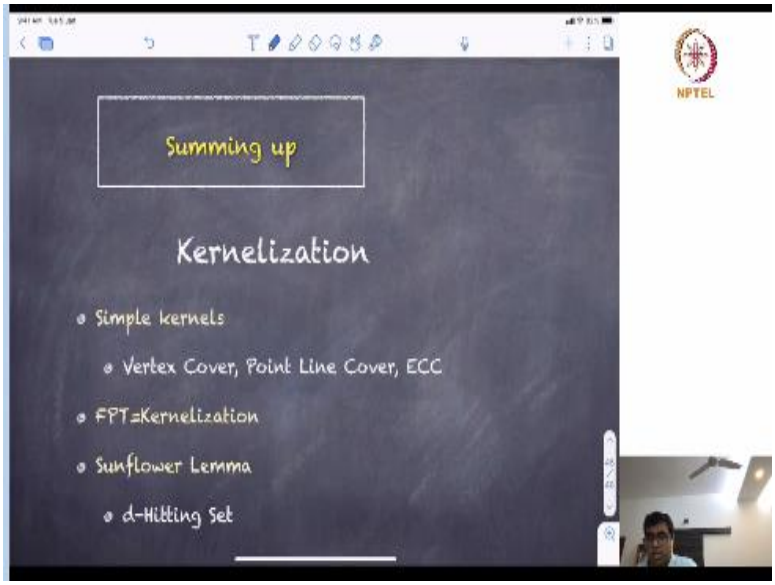
We can apply the same kind of idea but using, so, you can prove that if a set contains a sunflower with dk + 2 petals, 1 set of the sunflower can be deleted. So, you can apply this reduction rule based on sunflower and get the d-set packing admits a kernel with order k to the power d-sets. So that is the last kernel that I want to touch.

(Refer Slide Time: 41:56)



So, there are more problems, one can kernalize like cluster editing or feedback Arc set in tournament. Look at the exercises that will come follow.

(Refer Slide Time: 42:04)



So, let us just summing up these 2 lectures on kernel. So, we saw some simple kernel for vertex cover, point line cover, edge clique cover. We saw when FPT is equal to kernelization under what condition. We saw this nice sunflower lemma and saw an application to d-hitting

set and d-set packing. We will have couple of more lectures on kernelization to follow in the due course. Okay, so, thanks for now.