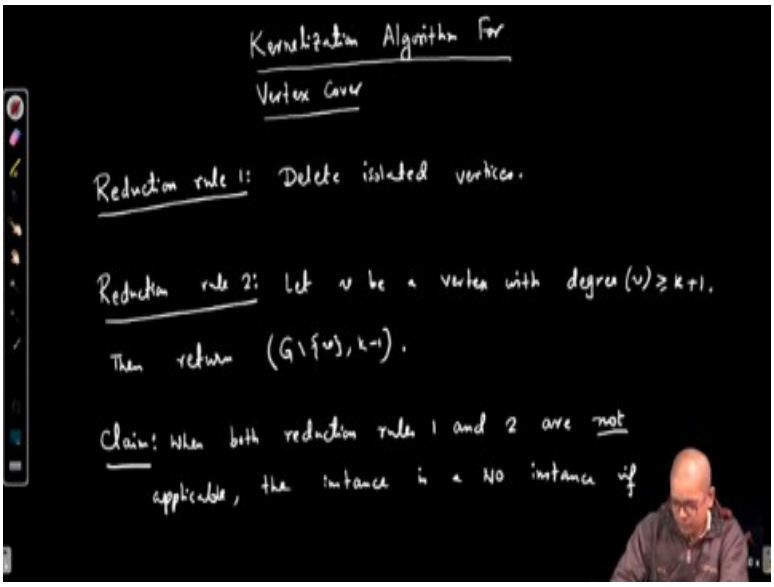
Selected Topics in Algorithm
Prof. Palash Dey
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 59
Introduction to Kernelization

Welcome. So, in the last class we have seen a faster FPT algorithm for vertex cover and you have also seen the definition of kernelization. So, in today's class let us understand what let us see an example of a kernelization algorithm for vertex cover.

(Refer Slide Time: 00:46)



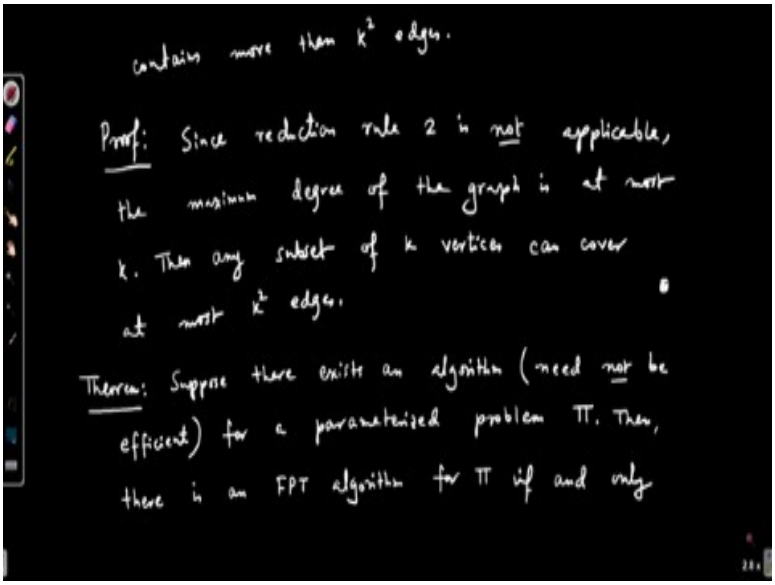
So, let us do some easy pre-processing rules. So, in kernelization they are called reduction rules because you are we are reducing the size of the instance. The first reduction rule one is delete isolated vertices that is vertices with degree zero. So, clearly there is the new instance is equivalent to the old instance this does not affect the size of the minimum vertex covered because isolate vertices does not belong to a minimum vertex cover.

Reduction rule 2 for every vertex or let V be a vertex with degree of v greater than equal to k + 1 at least k + 1 edges incident on v then any case size minimum vertex cover must pick v. Because if it does not peak v it has to pick its neighbourhood whose size is at least k + 1. So, then return G - v pick v and the budget becomes k - 1. Now we claim that the number of vertices and

number of edges must be small. So, now we claim so we keep applying reduction one and two repeatedly.

So, maybe at the beginning there is no isolated vertex but after applying reduction rule 2 there are some isolated vertices is created. So, then again, some those isolated vertices we delete and we keep doing this when we keep applying reduction rule one and two whenever any one of them is applicable. When both reduction rules one and two are not applicable the graph has the instance is a NO instance.

(Refer Slide Time: 05:27)



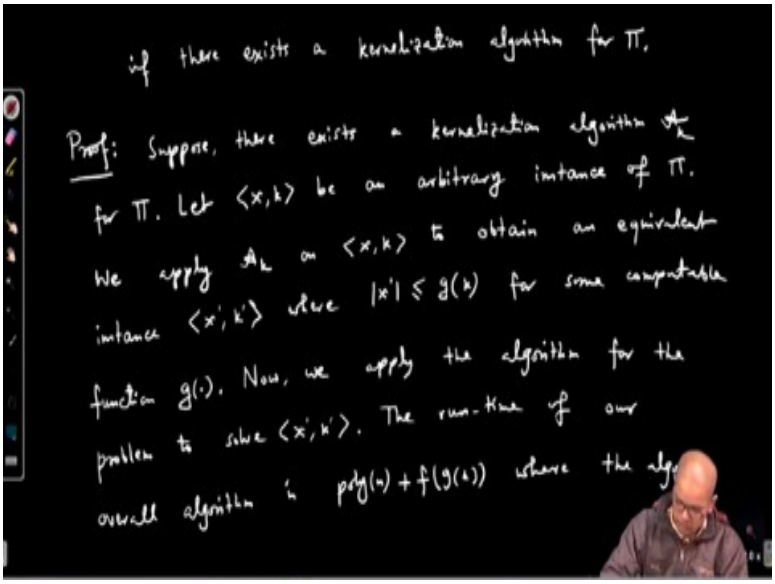
If it contains more than k square edges of course you know if k becomes negative or 0 and there is some isolated convert some each then instance is NO instance. So, we can write it here induction will 0, if k = 0 and the graph contains any edge, return no instance. Recall our kernelization algorithm does not solve the problem it outputs an equivalent instance. So, it in this case if k is 0 and there the there is some H then the graph is then the instance is a NO instance.

And it creates output simply a some trivial no instance maybe a graph with two vertices and the edge and the budget is zero something like that. So, proof, since reduction rule 2 is not applicable the maximum degree of the graph is at most k. Now vertex cover of size at most k then can convert at most k square edges because the degree of every vertex is at most k then any subset of k vertices can cover as at most k square edges.

Hence if the graph has more than k square edges then it must be NO instance which concludes the proof. So, you see that this algorithm this kernelization algorithm runs in polynomial time. What is that algorithm? Keep applying direction to 0, 1 and 2 if any one of them applicable and if they are not applicable at the end if it has more than k square edges then again output dummy NO instance and if it is less than k square edges then output that instance.

So, next we see that no really this kernelization is just the other side of the coin of if bit algorithm. What do we mean by that? So, there is a theorem. Suppose there exists a an algorithm need not be efficient for a parameterized problem Π then there is an FPT algorithm for Π if and only if there exists a kernelization algorithm for Π, proof.

(Refer Slide Time: 11:23)



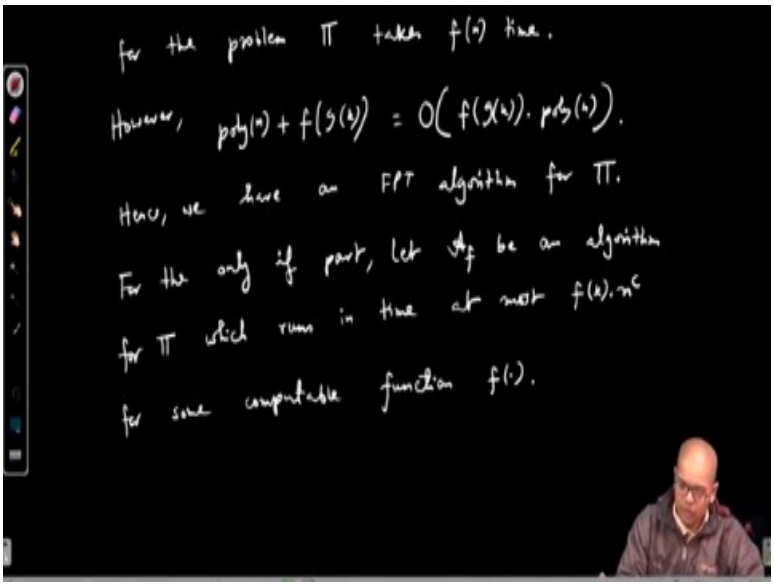
So, if direction so suppose there exists a kernelization algorithm A k for Π. So, using these we will design an FPT algorithm for Π. So, let x, k be an arbitrary instance of Π. What we do is that we first apply the kernelization algorithm on A k to create an equivalent instance. We apply A k on x, k to obtain and equivalent instance $x^{'}$, $k^{'}$ where size of $x^{'}$ is less than equal to $g(k)$ for some computable function g.

Now what we do is it now we use the assumption that there exists an algorithm will not be efficient to not run in polynomial time but there existing algorithm for the problem. So, we apply

that algorithm. Now we solve the algorithm, we solve we apply the algorithm for the problem to solve x', k'. The runtime of our overall algorithm is you know first we applied the conversation algorithm which took poly n time which is a polynomial time algorithm.

Plus, then we apply may be brute-force algorithm which run in say $f(n)$ time but the size of the instrument is $g(k)$.

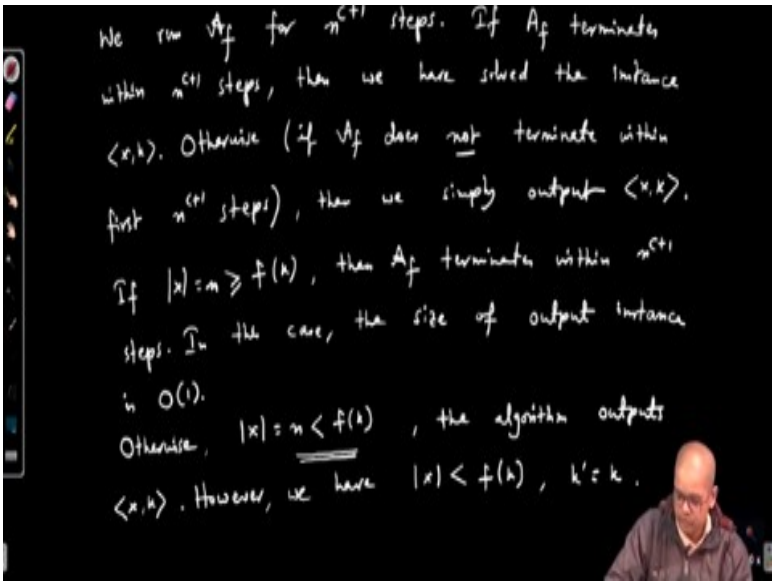
(Refer Slide Time: 16:25)



So, $f(g(k))$ where the algorithm for the problem; Π takes $f(n)$ time. However, the runtime is not exactly in FPT form it may appear but f of poly n + $f(g(k))$ is big O of you know $f(g(k))$ times poly n. Hence, we have an FPT algorithm for Π so if part is done. Now for the only if part means suppose that we have an FPT algorithm then we will show that there also exist a kernelization algorithm.

For the only if part let A f be an algorithm for Π which runs in time big O of $f(k)$ times poly n time at most $f(k)$ times n to the c of time at most this for some computable function f. Now what we do is we compare n with f.

(Refer Slide Time: 19:43)



So, what we do is you know we run A f for n to this c + 1 steps if A f terminates within n to this c + 1 steps then we have solved the instance x, k. And once solve we can always output accordingly if the instance is found to be YES instance, we can output a dummy YES instance and if the our instance is found to be a NO instance then we will output a dummy NO instance. Otherwise, it means if A f does not terminate within first n to this c + 1 steps then we simply output x, k.

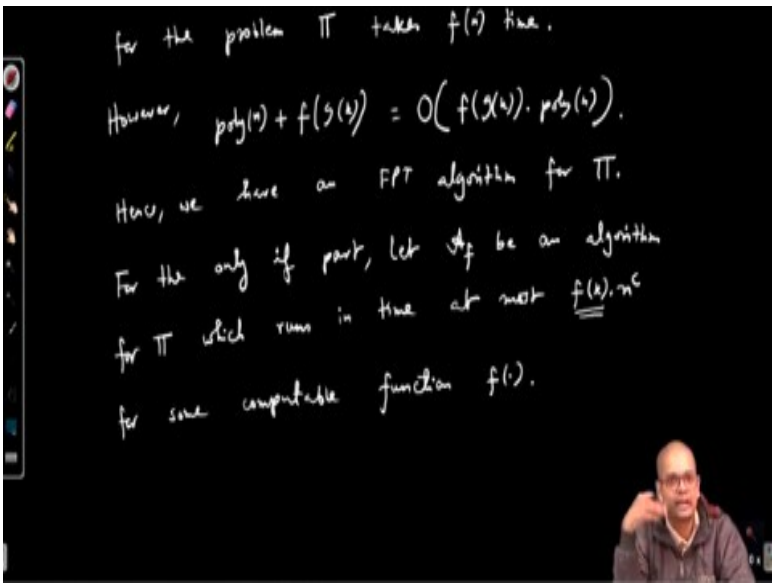
Now we claim that this algorithm is a kernelization algorithm. So, two cases if size of x which is n size of input instance if this is more than $f(k)$ or if this is less than $f(k)$ then an FPT algorithm. How much time it takes? It takes at most $f(k)$ times n to the c and $f(k)$ is at most n then A_f terminates within n to the c + 1 steps. Why? Because it takes at most $f(k)$ times n to the c steps and $f(k)$ that moves to n.

So, $f(k)$ times n to this c is at most n to the c + 1 and in this case hence the output instance is small. Because we will be outputting, we are solving the infinite we have solved the instance and we will simply output a dummy instance of constant size. So, in this case the size of output instance is big O of 1. Otherwise, we have card in size of x which is n which is greater than f of k. In this case algorithm may not terminate but if it does not terminate then what will the output?

What is the size of the output instance? The size of the output instinct is x only the same instance we output and x is size f of x is n and which is greater than $f(k)$. In this case the algorithm outputs x, k however we have size of x is if $f(k)$ is less than equal to n then only $f(k)$ times n to the c is less than equal to n to the c + 1 and the other case is n is less than $f(k)$. If n is less than $f(k)$ then $f(k)$ times n to the c is need not be less than equal to n to the c + 1.

So, hence but in this case, we have a size of x is less than $f(k)$. Of course, k' is a new parameter which is same as k.

(Refer Slide Time: 26:23)



So, hence we have kernelization algorithm for Π which outputs kernel of instance size at most f of k and parameter is bounded above by k. Moreover, f is computable function by the assumption of FPT algorithm A_f for Π. So, this concludes the proof. Hence, you know to design an FPT algorithm there is also an alternative technique to design a kernelization algorithm. And you know for some problems the best FPT algorithm known is via this suit that means we apply we design a kernelization algorithm.

And if we design a kernelization algorithm here whose output kernel is $f(k)$ and typically if the problem is in x p that means there exist an algorithm of $A(x)$ into the k time then we can fit that and using that we can design an FPT algorithm. So, this shows that if we want to design a kernelization algorithm we can as we will design a FPT algorithm and vice versa. To design a

FPT algorithm it is equivalent to design a kernelization algorithm but the runtime may be different.

As we have seen you know for vertex cover the same search based technique but by applying more clever technique we are able to reduce the run time of the algorithm for vertex cover from 2^k to 1.46^k. So, let us stop here today. So, the next class will see some more examples of FPT algorithms and kernelization.