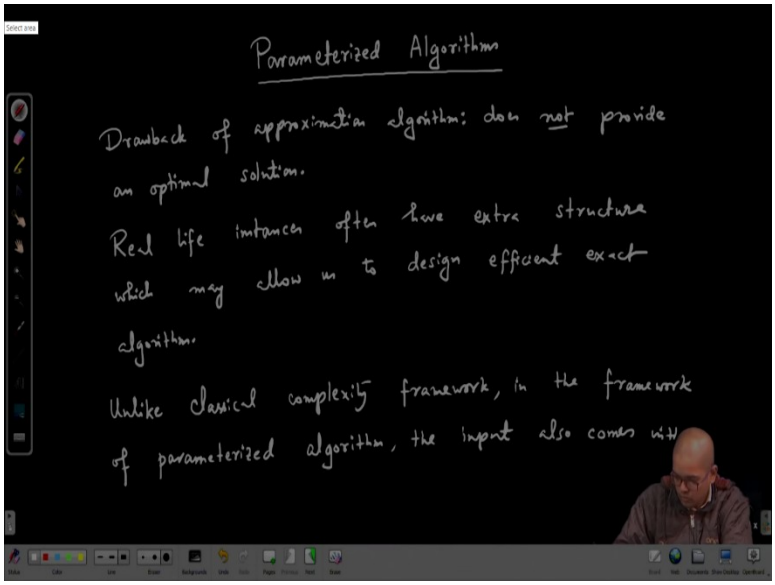
Selected Topics in Algorithm
Prof. Palash Dey
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 57
Introduction to Parameterized Algorithm

In the last class we have seen a primal dual schema-based approximation algorithm design and that concludes our second part which is on approximation algorithm. Now in the next couple of lectures we will see another way to tackle with the hardness of NP completeness and that is parameterized algorithm.

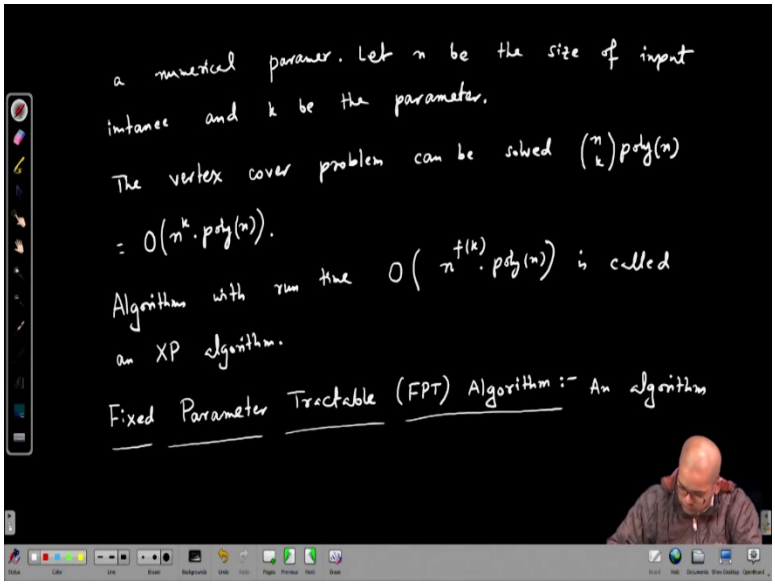
(Refer Slide Time: 00:45)



So, parameterized algorithm, so what is the drawback of approximation algorithm? Although the runtime of the algorithms were polynomial time the algorithms were efficient you know it failed to give an exact solution does not provide and optimal solution. Now in many instances often in often real-life instances have extra structure which may allow us to design and exact algorithm even fast.

Real life instances often have extra structure which may allow us to design efficient exact algorithm. Now with this goal in mind what parameterise algorithm does this framework of parameters algorithm sort of to model this extra structure extra information. The input consists of two parts one is the n which is the length of the input and another is parameter. So, unlike classical complexity framework in the framework of parameterized algorithm the input also comes with a numerical parameter.

(Refer Slide Time: 05:11)



So, let n be the size of input instance and k be the parameter. This parameter could be anything for example for a vertex cover problem the parameter could be the size of the vertex cover that you are looking for max cut problem the parameter could be the size of the max cut that we are looking for or it could be neither structural parameter. For example, for the vertex cover problem the parameter could be the highest degree of the graph or the size of the minimum cut in the graph.

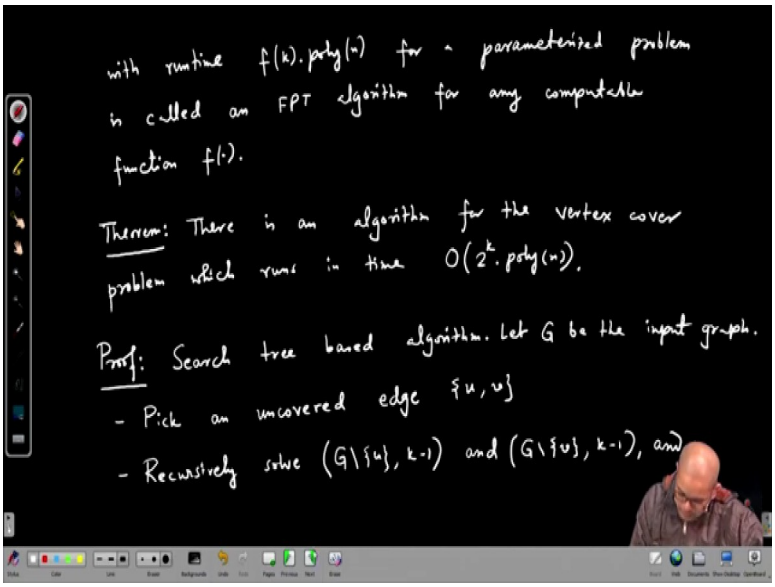
Or any other structural parameter or the chromatic number of the graph and so on. So, anything and so each of the same problem with different parameters are different problems in the parameterized world. So, and what we are looking for? For example, the vertex cover problem can be solved you know, if you are looking for a key size vertex cover and then we can simply iterate over all into scale subsets of vertices.

And check whether it is a vertex cover or not, so in this much time which is like big of n to the power k times poly n and this is typical. For example, y vertical only for example click independent set or typical problems can be solved in n to the power k time but this sort of time where the parameter sits in the exponent of n or polynomial in n these are called XP algorithms.

So, algorithms with runtime big O of some into some $f(k)$ times poly n algorithms with runtime this is called and XP algorithm. But this is not what we are looking for in

parameterized algorithms. So, what in parameters algorithm what we are looking for is called fixed parameter tractable algorithm, in short FPT algorithms. So, what is an FPT algorithm?

(Refer Slide Time: 10:13)

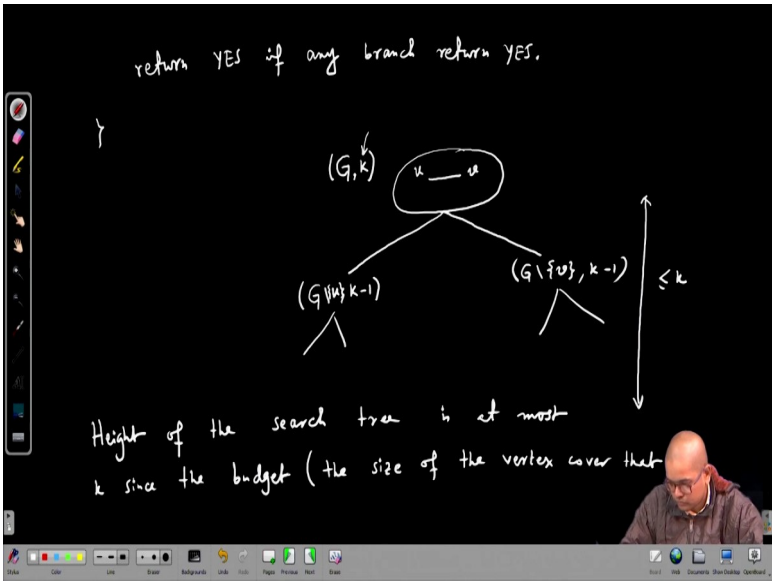


An algorithm which runtime f of k times quality of n for parameterized problem is called and FPT algorithm for any computable function f. So, example, so there is a theorem there is an algorithm for the vertex cover problem which runs in time big O of 2^k times poly n. So, what is the algorithm proof? It is a search tree-based algorithm, what it does is that it picks an edge because that edge needs to be covered.

And that means that one of its two endpoints must be picked in the solution. So, it picks both the one of its endpoints and tries, so pick and uncovered age u, v and recursively solved suppose G is the input graph. So, let G be the input graph circuits will be solved you pick u from G, G - u and this deletes all the edges which are incident on you input that means it deletes all the edges remove.

It removes all the edges from G which are covered by u and in the remaining graph I am looking for the a vertex curve of size k - 1. Now you see that input consists of two things one is the instance and under these parameters the parameter value also drops. So, you can see we solve this and to try both the possibilities and this.

(Refer Slide Time: 15:18)

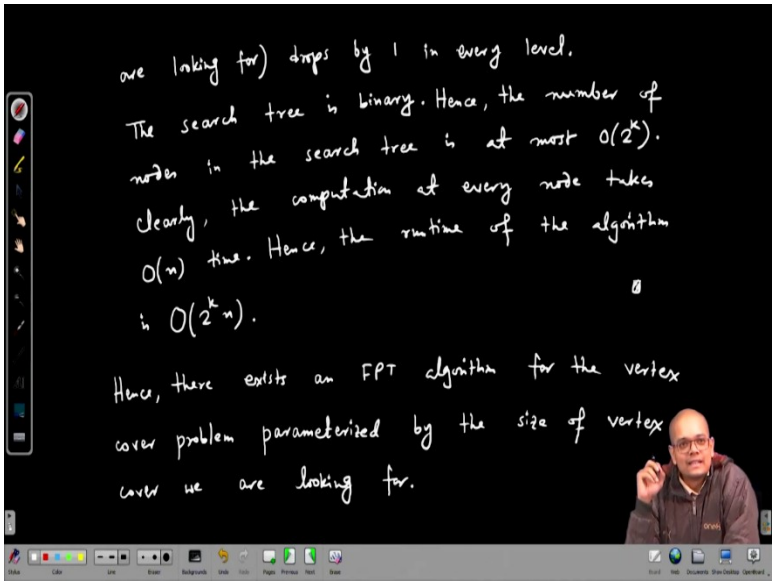


And return is YES means is there indeed exists of vertex cover of size k in the graph G, so return is if any branch return YES and so what will be the base case base case is if k = 0 or if k is negative, then we cannot have or if k = 0 then the dash must not be any edge. If k = 0 and there is an uncovered u, v in return NO. So, pictorially I have a graph G and, in this graph, I am looking for a vertex cover of size k I am asking is there a vertex cover of size k.

So, it is a graph and there is an edge which is uncovered between u and v what I do is that I try picking both one at a time, so I go to G - u and in that graph after deleting u I am looking for a vertex cover of size k - 1 and G - v in this graph, we are looking for a vertex cover of size k - 1 and so on. Now what is the depth of the tree? The depth of the tree is at most k if we if at some point the budget key is non-negative and there is no edge in the graph then then it is YES.

So, here also let me write if no edges then also return YES otherwise if there is uncovered edge and if k = 0 return NO, so this height is at most k, so height of the search tree is at most k. Why? Since the size of the vertices see the budget which is the size of the vertex cover, we are looking for since the budget drops by one in every level.

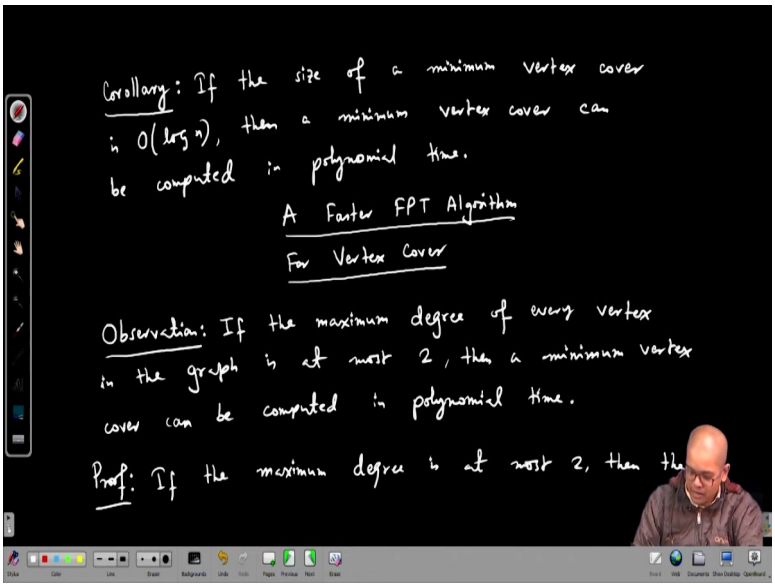
(Refer Slide Time: 19:19)



So, height is at most k and the search tree is binary, hence the number of nodes in the tree is at most big O of 2^k. Now what is the computation in each node? That computation is this that checking if there is NO edge and if budget is 0 then return NO if there is an uncovered rate this is the computation at every node. Clearly, the computation at every node takes order n time hence the runtime of the algorithm is $O(2^k n)$.

So, this is an FPT algorithm hence there exists are FPT algorithm for the vertex cover problem parameterized by the size of vertex covered we are looking for. Also, a quality of this algorithm is that if the size of the minimum vertex cover is big O of $\log(n)$ then such a vertex cover can be found in polynomial time.

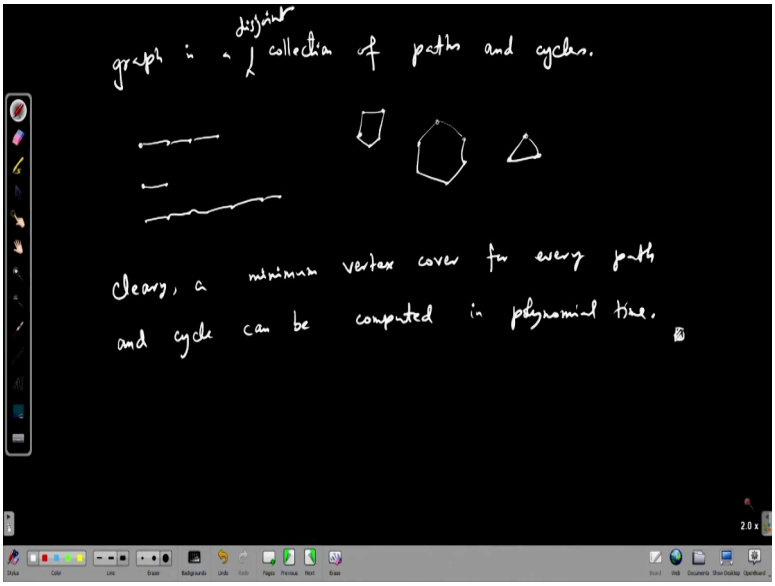
(Refer Slide Time: 23:35)



If the size of minimum vertex cover is bigger of log n, then minimum vertex covered can be computed in polynomial time. Next what we see is a faster algorithm faster FPT algorithm for vertex cover, so a faster FPT algorithm for vertex covered. So, for that we observed that if the maximum degree of a word of any vertex in a graph is 2 or at most 2, then the vertex cover can be computed in polynomial times.

So, there is an observation, if the maximum degree of any vertex in the graph is at most 2, then now let me write of every vertex, then minimum vertex covered can be computed in polynomial time. How? First observe that under this assumption if the maximum degree is 2 or at most 2, if the maximum degree is at most 2.

(Refer Slide Time: 28:04)



Then the graph is a collection of paths and collection of either disjoint collection of paths and cycles that means this looks like the graph, there are some paths and there are some cycles. Now for paths and cycles the vertex cover clearly can be computed the minimum vertex cover can be clearly compared in polynomial time. Now clearly the vertex cover the minimum vertex cover for every path and cycle can be computed in polynomial time.

And what is the minimum vertex cover of this graph? It is a union of minimum vertex cover of each connected components that means for each path and cycles you find the minimum vertex cover and take this joint union that is the minimum vertex cover of the whole graph. So, let us stop here and we will see in the next class how using this we will be able to design a much faster algorithm for finding a minimum vertex cover.