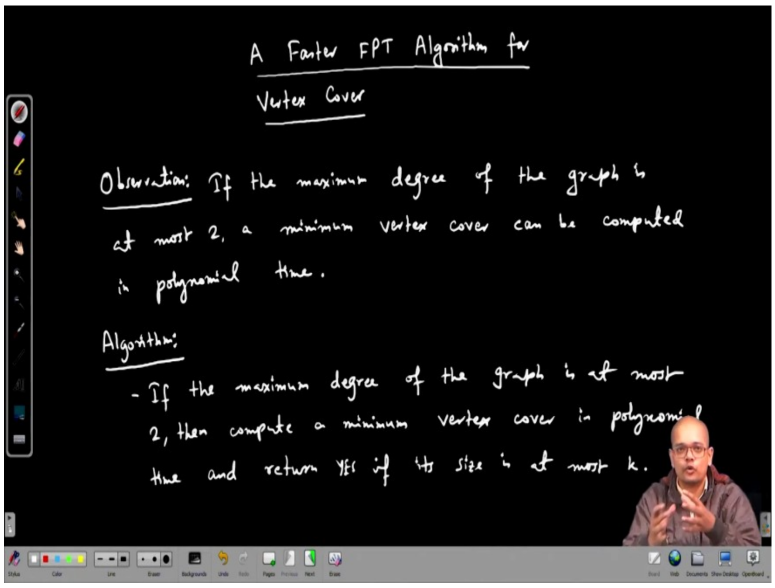
Selected Topics in Algorithm

Prof. Palash Dey
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 58
Faster FPT Algorithm for Vertex Cover

Welcome to the last class we have seen what is parameterised algorithm, what is an FPT algorithm and we have seen easy 2 to the power k time algorithm for finding a minimum vertex cover and now we are designing a faster algorithm for minimum vertex cover. So, let us continue.

(Refer Slide Time: 00:48)



A Faster FPT Algorithm for Vertex Cover

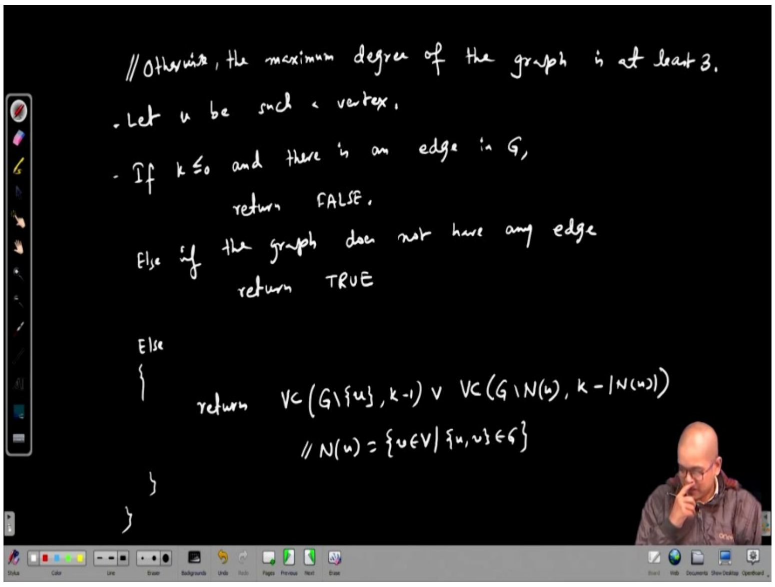
Observation: If the maximum degree of the graph is at most 2, a minimum vertex cover can be computed in polynomial time.

Algorithm:
- If the maximum degree of the graph is at most 2, then compute a minimum vertex cover in polynomial time and return YES if its size is at most k.

So, faster FPT algorithms for vertex cover and for that we will use that observation. What is the observation? That if the maximum degree of the graph is at most 2, minimum vertex cover can be computed in polynomial time. Now what is the algorithm? First of all, again it is a recursive algorithm again, it is a search tree-based algorithm, the base case is if the maximum degree of the graph is at most 2, then return then compute minimum vertex cover in polynomial time and return it return YES.

So, let us talk about decision version that is if it is size is at most k it means we are given a graph G and an integer k and the question is what is the size of the minimum vertex cover at most k or not.

(Refer Slide Time: 04:44)



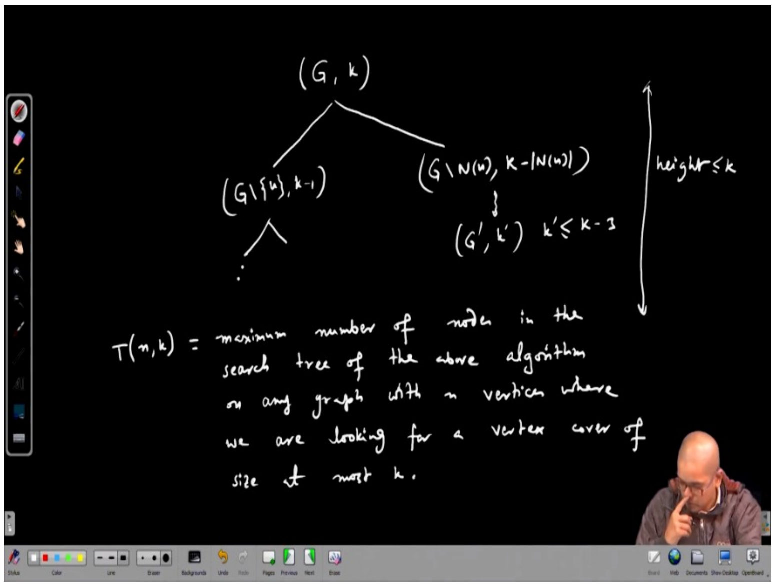
```
// Otherwise, the maximum degree of the graph is at least 3.
- Let u be such a vertex.
- If k ≤ 0 and there is an edge in G,
        return FALSE.
  Else if the graph does not have any edge
        return TRUE

  Else
  {
        return  $VC(G \setminus \{u\}, k-1) \vee VC(G \setminus N(u), k-|N(u)|)$
              // $N(u) = \{v \in V \mid \{u,v\} \in G\}$
   }
}
```

So, next otherwise the minimum degree, so you will comment otherwise the max degree maximum degree of the graph is at least 3. So, pick such a vertex, so let u be such a vertex of course we will have those base cases. If k is 0 and there is an uncovered there is an edge in G then return FALSE, else if k 0 and else if not k is 0 if the graph does not have any edge, then return TRUE.

Else what you do is that let u be a such vertex either you pick u and recurs but if you do not pick u you have to pick its neighbourhood of u. So, return in this case return call the vertex cover problem this function recursively and you remove the u and drop the budget by 1 to make it k is less than equal to 0 or either after picking u and you deleting you in the remaining vertex there is a vertex cover of size k - 1 or look for is there a vertex cover of size.

If you do not pick you need to pick the neighbour of u, $N(u)$ is the set of vertices which has direct edge with you in this case drop the budget by $N(u)$. So, this you return, so what is $N(u)$? $N(u)$ is set of vertices such that there is an edge from B, so this is the algorithm. So, let us analyse it.

(Refer Slide Time: 08:34)

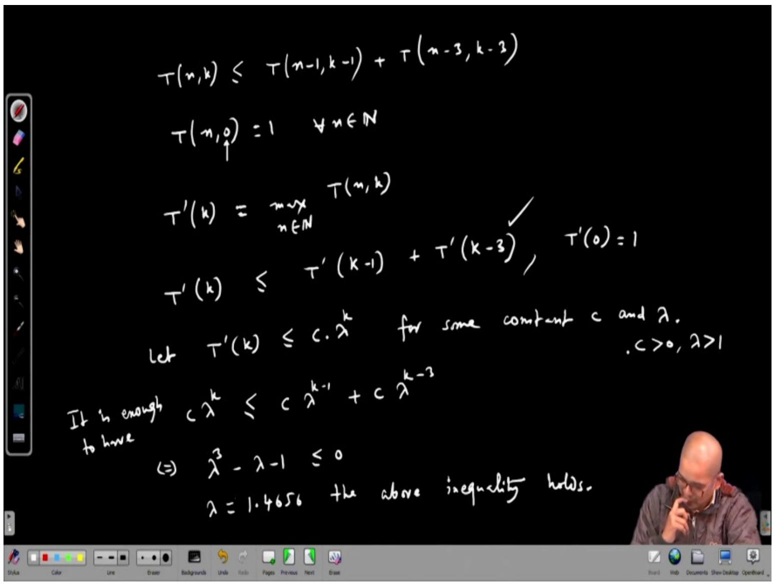


$T(n,k)$ = maximum number of nodes in the search tree of the above algorithm on any graph with n vertices where we are looking for a vertex cover of size at most k.

So, pictorially let us see G and k, now in one branch it picks suppose u is that vertex, so G, k - 1 and in the other branch it is G and you pick the neighbourhood in u and k - cardinality $N(u)$ which is like you know the sum graph G' and k' and k' is $N(u)$ is at least 3 because the degree of u is at least 3. So, k' is less than equal to k - 3 and so on. Now here also the height is k but one branch is much more shallower.

So, how many nodes are there? So, let us see so let $T(n,k)$ be the maximum number of nodes in the search tree of the above algorithm on any graph with in vertices where we are looking for vertex covered of size at most k.

(Refer Slide Time: 11:32)



$T(n,k) \leq T(n-1,k-1) + T(n-3,k-3)$

$T(n,0) = 1 \quad \forall n \in \mathbb{N}$

$T'(k) = \max_{n \in \mathbb{N}} T(n,k)$

$T'(k) \leq T'(k-1) + T'(k-3), \quad T'(0) = 1$

let $T'(k) \leq c.\lambda^k$ for some constant c and λ. $c > 0, \lambda > 1$

It is enough to have $c\lambda^k \leq c\lambda^{k-1} + c\lambda^{k-3}$

$\Leftrightarrow \lambda^3 - \lambda - 1 \leq 0$

$\lambda = 1.4656$ the above inequality holds.

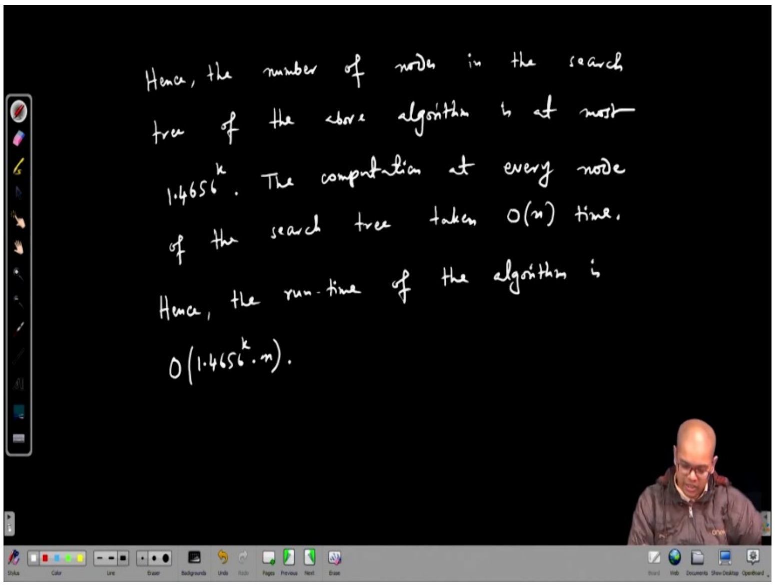
So, $T(n,k) \leq T(n-1,k-1) + T(n-3,k-3)$ and of course $T(n,0) = 1$ this is for all n. Now because you know n and k are dropping by the same quantity, so this recursion tree will at the leaf k will be heated 0. So, it is enough. So, define T prime of k so focus on k because it will be a function of k only $T(n,k)$ the max of in $T(n,k)$. Now then what is $T'(k)$? $T'(k)\leq T'(k-1)+T'(k-3)$ and the base case is $T'(0)=1$.

Now we need to solve this recursion, now this is a standard there is a standard technique to solve this recursion so for that I will solve this we will give the recursive solution. So, let $T'(k)$ be less than equal to the there is a form of a form of $T'(k)$ with this sort of recursions it is $c\lambda^k$ for some constant c and λ. Now let us put it here, this is $c\lambda^k$ we need this to be less than equal to.

For these to hold which this enough to have $c\lambda^k \leq c\lambda^{k-1}+c\lambda^{k-3}$. If this is true then this inequality holds because then $T'(k-1)\leq c\lambda^{k-1}$ and $T'(k-3)\leq c\lambda^{k-3}$ and if this sum is more than $c\lambda^k$.

Now this is if and only if $\lambda^3-\lambda-1\leq 0$, so solving this what we get is λ is and of course c and λ must be positive and c must be positive and λ should be greater than 1. So, with all these constraints what we have is λ is we need to choose λ to be equal to for λ = 1.4656 above inequality holds.

(Refer Slide Time: 16:52)



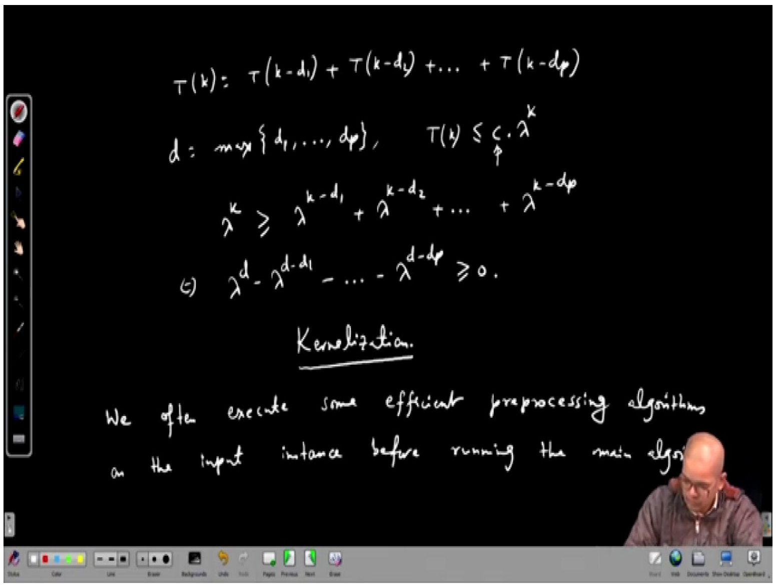
Hence, the number of nodes in the search tree of the above algorithm is at most 1.4656^k. The computation at every node of the search tree takes $O(n)$ time. Hence, the run-time of the algorithm is $O(1.4656^k \cdot n)$.

Hence the number of nodes in the search tree of the above algorithm is at most 1.4656^k. This will be greater this should be greater than enough to have this. Now the computation at every node of search tree, now what is the computation? So, let us go to the algorithm, so at the base case maybe we need to find a minimum vertex cover for a collection of cycles and paths a disjoint collection of cycles and paths.

And the rest of the computation is takes order in time. So, the computation at every node of the search tree takes order in time even for the base case when the max degree is at most to finding the minimum vertex cover takes order of n + m time you need simply needs to iterate over all the edges and but the number of edges is at most two times the number of vertices because the degree is at most 2.

It is basically like the same number of vertices because you know sum of degree is twice the number of edges. So, the number of edges is at most the number of vertices in that case, so in order of n + m is order of m order of n only. So, computation at every node circuit is ordering time. Hence the runtime of the algorithm is $O(1.4656^k n)$.

(Refer Slide Time: 20:49)



$$T(k) = T(k-d_1) + T(k-d_2) + \dots + T(k-d_p)$$

$$d = \max\{d_1, \dots, d_p\}, \quad T(k) \leq c \cdot \lambda^k$$

$$\lambda^k \geq \lambda^{k-d_1} + \lambda^{k-d_2} + \dots + \lambda^{k-d_p}$$

$$\Leftrightarrow \lambda^d - \lambda^{d-d_1} - \dots - \lambda^{d-d_p} \geq 0.$$

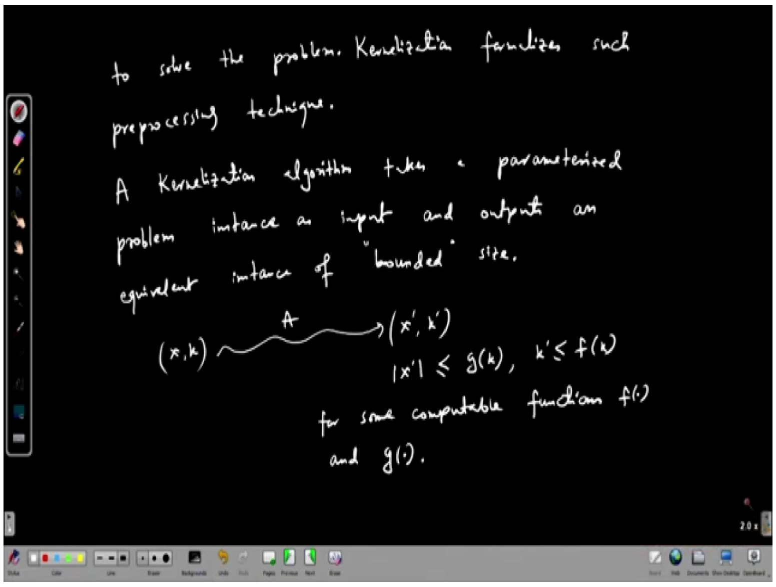
Kernelization.

We often execute some efficient preprocessing algorithms on the input instance before running the main algo

So, in general how to solve recurrent solutions of the form say $T(k)=T(k-d_1)+T(k-d_2)+\dots T(k-d_p)$. It is like you guess let d be max of $\{d_1,\dots,d_p\}$ and you guess $T(k)\leq c\lambda^k$, the value of c will get from the base case of this recursion. So, this boils down to it λ^k is greater than equal to you know $\lambda^{k-d_1}+\dots+\lambda^{k-d_p}$.

Which is if and only if $\lambda^d-\lambda^{k-d_1}-\dots-\lambda^{k-d_p}\geq 0$. So, next what we will see is what is called a kernelization. So, often you know in real world instances or in real life we do some pre-processing before applying the algorithm to solve the problem. So, we often execute sum efficient pre-processing techniques or pre processing using algorithms on the input instance before running the main algorithm to solve the problem.

(Refer Slide Time: 24:11)



to solve the problem. Kernelization formalizes such preprocessing technique.

A Kernelization algorithm takes a parameterized problem instance as input and outputs an equivalent instance of "bounded" size.

$$(x,k) \xrightarrow{A} (x',k')$$

$$|x'| \leq g(k), \quad k' \leq f(k)$$

for some computable function $f(\cdot)$ and $g(\cdot)$.

Kernelization formalizes such pre-processing techniques. For example, for vertex cover we can simply delete all the isolated vertices. The input instance can contain some isolated vertices and you can simply delete it without affecting the optimal solution or you know there are some more of this sort of preparation techniques which can sort of clean the input instance and focus on the core problem.

For example, for vertex cover all the degree one vertices we can delete and take the take its neighbour as in the part of optimal vertex commercial and this sort of simple technique. So, you can use to clean up the data to do the pre-processing. So, what is a kernelization task? So, the kernelization algorithm takes a parameterized problem instance as input and outputs and equivalent instance of bounded size.

What do I mean by that? So, suppose there is an instance x, k is the parameter and the parameter range algorithm take x, k, this algorithm A and outputs an equivalent instance that means this new instance the output instance is a YES instance. If and only if the input instance is a YES instance and the size is bounded by some parameters by some function of parameter alone.

So, size of $x^{'}$ is less than equal to G of k and $k^{'}$ is also less than equal to $f(k)$ for sum computable functions if and G. So, the next class we will see some examples of parameterized algorithms for vertex cover and some other problems and then we will see that you know this condensation algorithm is really, the other side of the parameters algorithm and its connection to FPT algorithms. So, let us stop here for today.