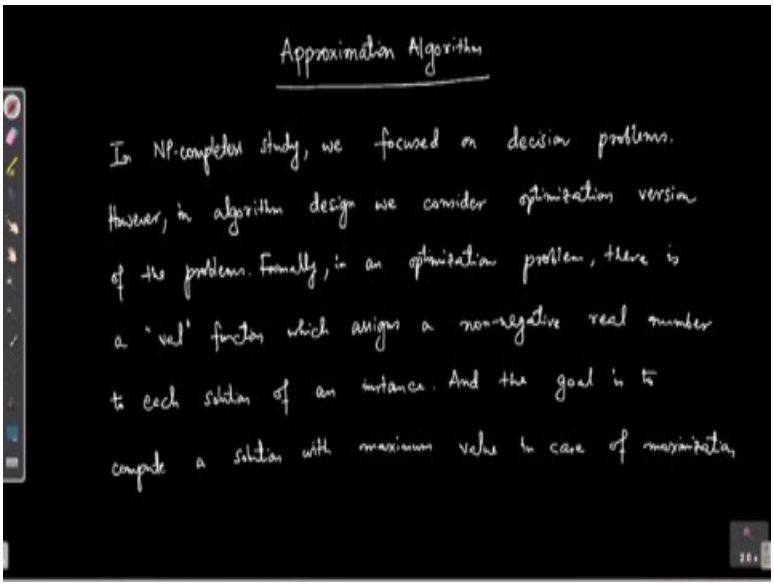
Selected Topics in Algorithm
Prof. Palash Dey
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 39
Self Reduction

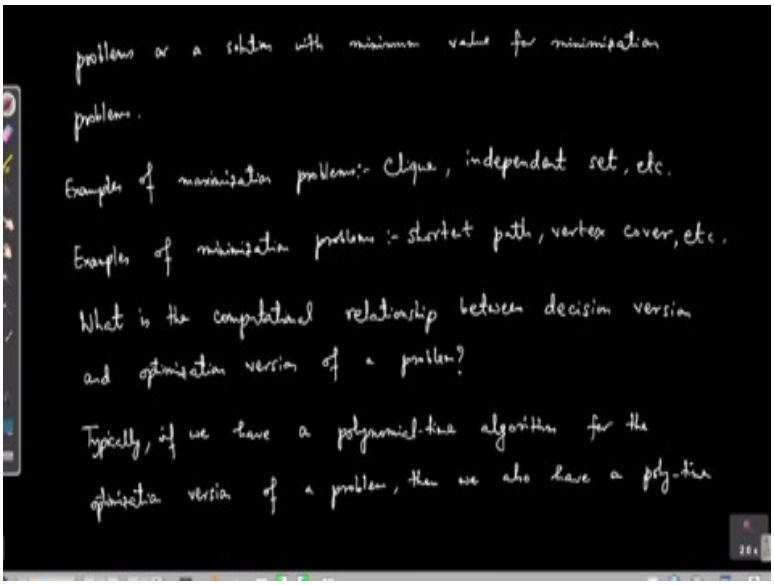
Welcome, so in the last class we have concluded seeing at NP completeness and we have seen various reductions. So, from this class we will start approximation algorithm this is one of the most successful layouts to tackle NP completeness. So, we start a new topic called approximation algorithm.

(Refer Slide Time: 00:49)



So, in NP completeness study we focused on decision problems. That means the answer is yes or no, but typically in algorithm design we look for up we study optimization version. However, in algorithm design we consider optimization version of the problems, what do we mean by that? More formally, in an optimization problem there is a value function called val which assigns are positive or say non-negative real number to each solution of an instance.

(Refer Slide Time: 05:30)



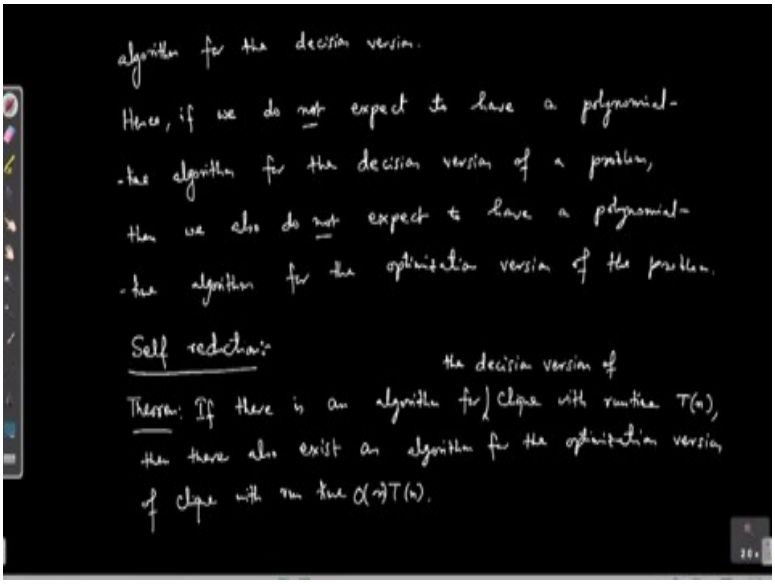
And the goal is to compute solution with maximum value in case of maximization problems or a solution with minimum value for minimization problems. So, this is the typical so what are the examples so example of maximization problems is like you know clique the find the largest subset of vertices which itself is a complete graph. That is a clique or independent set etcetera. Whereas minimization problem is like for example, shortest path given a graph G and two vertices S and T compute the shortest path from S to T or vertex cover etcetera.

Now what is the relationship between decision version and optimization version, because you know in NP framework, we are studying decision versions and that is because it gives a nice formalization of problems as languages and which helps us to study them as language membership problems and compare each problem as a set or subset of set of all strings over some alphabet.

Now what are the relationship between optimization version and decision version of the same problem and in terms of computationally? So, what is the computational relationship between decision version and optimization version of a problem? So, let us understand this with an example so let us take an example of a clique. So, suppose I have a polynomial time algorithm for optimization version.

So, we can see that that implies a probability algorithm for decision version. If I can find the largest clique in polynomial time then given an integer K, I can easily decide whether there exists a clique of size at least K or not. So, typically no, we let us understand this in the typical case. Let us not make it more formal; typically, if we have a polynomial time algorithm for the optimization version of a problem then we also have a poly-time algorithm for the decision version.

(Refer Slide Time: 11:50)

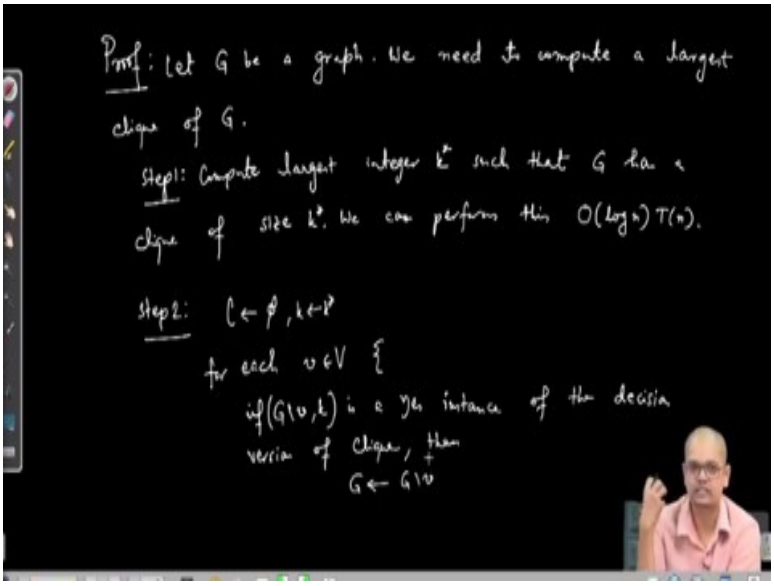


So, this implies that you know if we prove that would not expect to have a polynomial time algorithm for decision version then you do not expect to have a polynomial time algorithm for optimization version. Hence, if we do not expect to have a polynomial time algorithm for the decision version of a problem then we also do not expect to have polynomial time algorithm for the optimization version of the problem.

So, this justifies why it is enough to focus on decision version of the problem when we are proving hardness because, hardness of decision version implies hardness of optimization version. But are the equivalent in some sense, so is the other way can we also says that if we have a polynomial time algorithm for decision version then we have a polynomial time algorithm for optimization version and this is often the case, often you can say and the idea is self-reduction.

So, let me write it let us see self-reduction in sort of one problem a one say clique problem, it the same idea can be used for vertex cover or any typical problems like even SAT or so on. So, let me write theorem, if there is an algorithm for clique with runtime say $T(n)$ for the decision version of clique with runtime $T(n)$ then, they are also exist an algorithm for the optimization version of clique with run time $O(n)T(n)$.

(Refer Slide Time: 17:33)



So, what is the idea? Proof, so I have an algorithm for decision version using that I will have an algorithm for optimization version. What is the optimization version? I need to given a graph G; I need to find the largest clique. So, let G be a graph. We need to compute largest clique of G. First step, step 1, first we find, what is the size of the largest clique and then we will find that clique. So, find compute largest integer k such that G has a clique of size k.

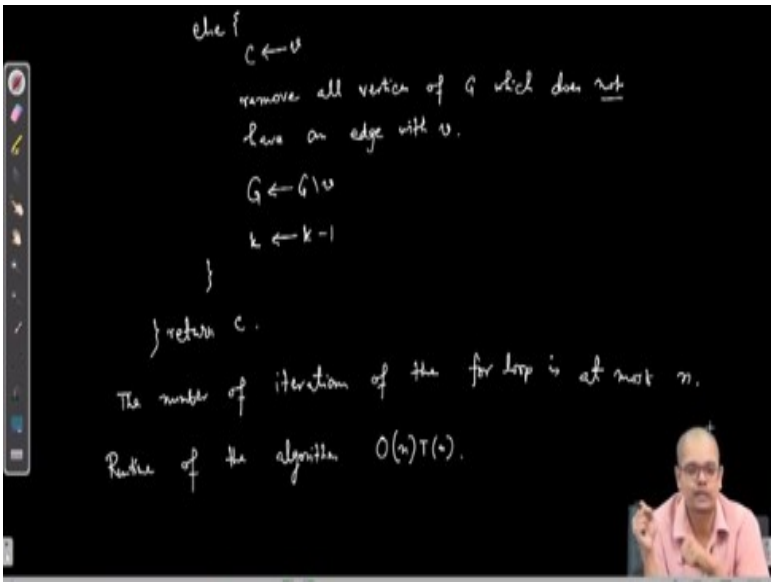
So, here we use the decision version of the problem, one nice approach is to start from one and go up to n and for each such k, $k=1,\ldots,n$ run the decision version to see whether there exists a clique of size k or not. The decision version for each k will say either yes or no. So, as these are linear search work you can do a binary search and we can perform this in how much time $O(\log n)T(n)$.

We need to call that algorithm for the decision version of the clique in big O of log times to this much time. So, now we know largest integers will call it k^*. Now we know the size of the largest

clique, now we need to compute the largest clique. So, what we do is that the second step, what we do is that for each so we will build the clique iteratively. So, let us call the clique C which is initialized to empty set.

And for each vertex $v \in V$ what we do is that we try to pick v we remove V from G and if the size of the largest clique does not get affected, it remains k^* then with throw V because there exists a clique of size k star without V. So, if G - v, and call it say $k = k^*$ is instance of the decision version of clique. That means what? That means after removing V we still have a clique of size k, then we simply drop G delete G, G there exists a large largest clique without C.

(Refer Slide Time: 23:13)



Otherwise, else what we do is that, we pick that vertex v in our clique is a clique we are building and we include v there. And we remove all vertices of G which does not have an edge with C. Because the clique that we are building those vertices which does not have an edge cannot be part of the trick, so we will move those vertices also from the graph. And we then remove v also from the graph G is G - v and v reduces k by1, it is the following.

So, what is the idea we pick an algorithm, we pick a vertex from G we delete it and after deleting does the size of the largest clique drop or not, if the size of the largest clique remains same, we simply delete v on the other hand if the size of the largest clique drops by 1. Then that means all

largest clique in that graph includes v that is what we mean we can infer from it. And then we picked v in my solution that we are building.

And then all the other vertices which are not have an edge with v we remove that those vertices also because they cannot be part of the clique that I am looking for a maximum clique. And once that is done, I remove v also because I know that in the clique that I am building v is there. And all vertices are connected to v. So, after that I make a decrement k also k is k - 1. So, the number of iterations of for loop is at most n, because in every iteration we are deleting at least one vertex.

If this if condition is true, then we delete one vertex otherwise we of course delete v and all other vertices not connected to v. So, at least one vertex is deleted and hence the number of iterations is at most n and then at the end we return C. So, the runtime of the algorithm is $O(n)$ how many calls we make to the decision version in each iteration of the for loop we make at most one call exactly one call and in step one we have made at most log n calls.

So, $O(n)+O(\log n)$ which is $O(n)T(n)$ that is the runtime of the algorithm. Hence what we have observed due to self-reduction is that both ways it is it is true. That means if we have a polynomial time algorithm for a decision version of a problem of a typical problem say clique or vertex cover or say set or knapsack whatever this using this self-reduction framework, we can have a polynomial time algorithm for the optimization version as well.

Of course, if we have a polynomial time algorithm for the optimization version, we have a polynomial time algorithm for decision version. So, this is just for the convenience mostly that we have focused on decision version of problems.