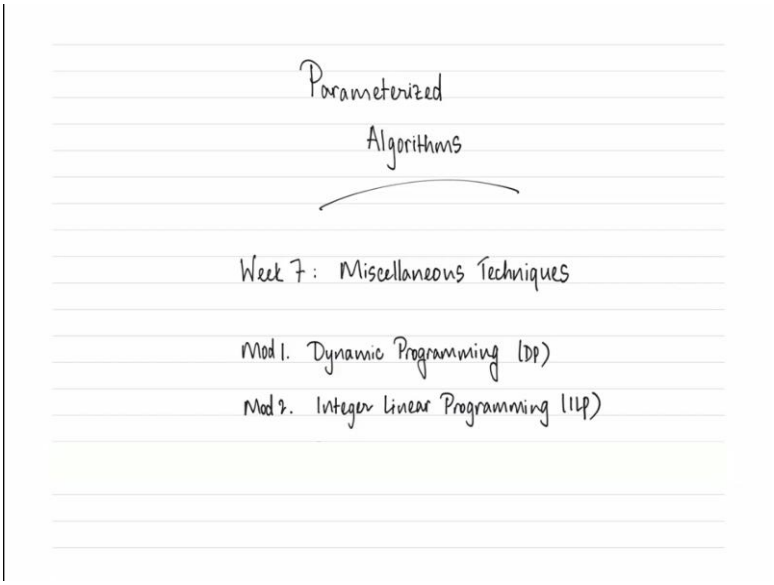
Parameterized Algorithms
Prof. Neeldhara Misra
Prof. Saket Saurabh
Department of Computer Science Engineering
Indian Institute of Technology – Gandhinagar IMSC

Lecture – 31
Dynamic Programming over Subsets for Set Cover

(Refer Slide Time: 00:11)



Welcome back to the 7th week of parameterized algorithms. This week is called miscellaneous techniques and the idea is to introduce you to an assortment of different algorithmic tools all of which are important, but they do not quite fit exactly any of the themes that we have discussed in the weeks so far. So, first I want to talk about dynamic programming.

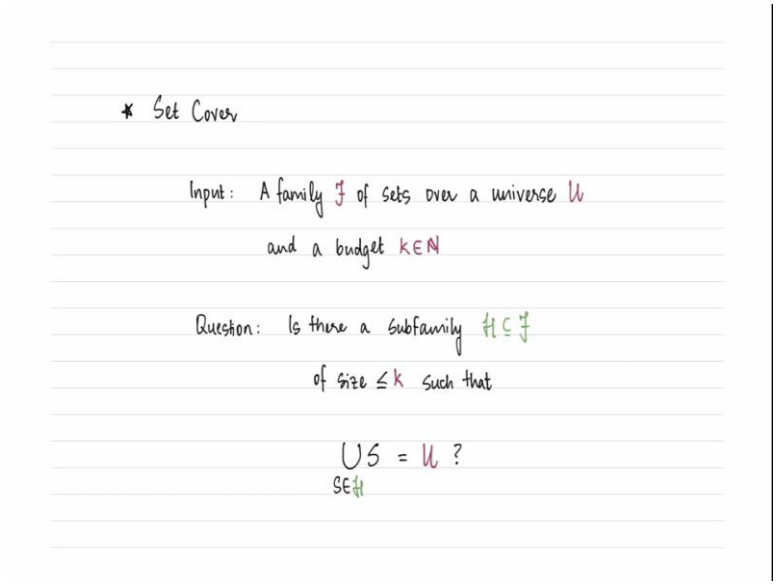
You have already a substantial flavour of dynamic programming in our discussions of 3D compositions and so on, but here we will be talking about a couple of different examples which are diverse from the context of 3D compositions and just emphasize how useful and ubiquitous this is as a technique. Next, we will talk about integer linear programming is a really powerful tool to design FPT algorithms.

It turns out that ILP as a problem it turns out to be FPT when you parameterize it by the number of variables involved. So, it is going to be a fairly general tool and again it may not be the best FPT algorithm that you can come up with for a given problem, but usually given

the versatility of the tool it turns out to be like a quick thing that you might want to do to even just establish that your problem is FPT to begin with.

So, we are going to start off with dynamic programming. We will do a couple of examples here; so this video is split into two segments. The first one which is going to be a relatively short discussion, it is going to be about the problem of set cover and the second one is going to be about Steiner trees. So, let us get started.

(Refer Slide Time: 01:44)



So, in the set cover problem this setting is the following. We are given a family of sets over a universe. So, we will typically call the family f and the universe u and we have a budget k and this might remind you at least in terms of just the setting this might remind you of hitting set and indeed set cover in some sense can be thought of as a dual of the hitting set problem. Yes so what is the goal here?

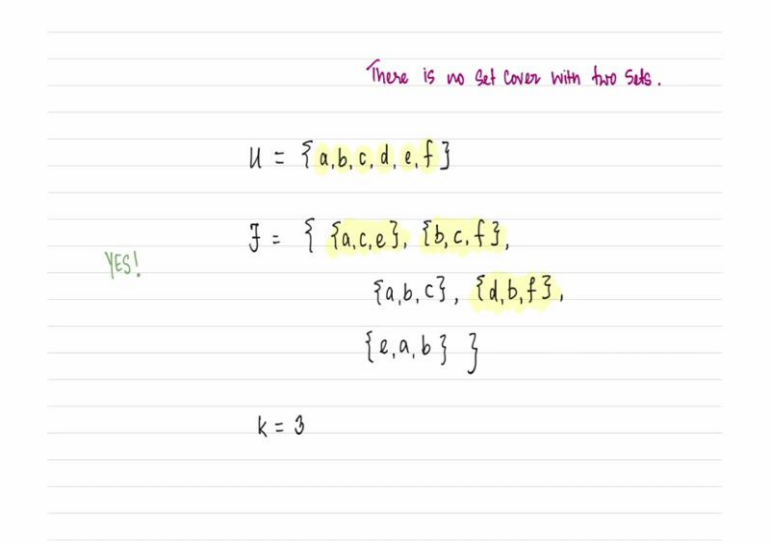
It is kind of the opposite of trying to hit everything it is a goal that involves covering everything. So, what we want is a sub family that is small in terms of the number of sets in the sub family and it should cover the universe and by this I mean that if you look at all the sets in your sub family you want every element in the universe to appear at least once. So, in other words just to make this a little more precise the question we are interested in is if there is a sub family of size at most k.

Such that when you take the union of all the sets in the sub family you get u which is the original universe. Now, there are variations (()) (03:09) this in the sense that you might for

example want every element in the universe to be covered exactly once and this variant is called exact set cover, you could also look at the maximization variant where you have apart from the budget also a target let us call the target t.

And you want to see if you can cover at least t elements with a budget of at most k. This is a specially relevant if the answer to your original question is no. So, it is a good exercise to think about some of these variations after we discuss the algorithm for the basic version that is presented here, but before we get to the algorithm just to make sure that we are on the same page with respect to the definition.

(Refer Slide Time: 03:56)



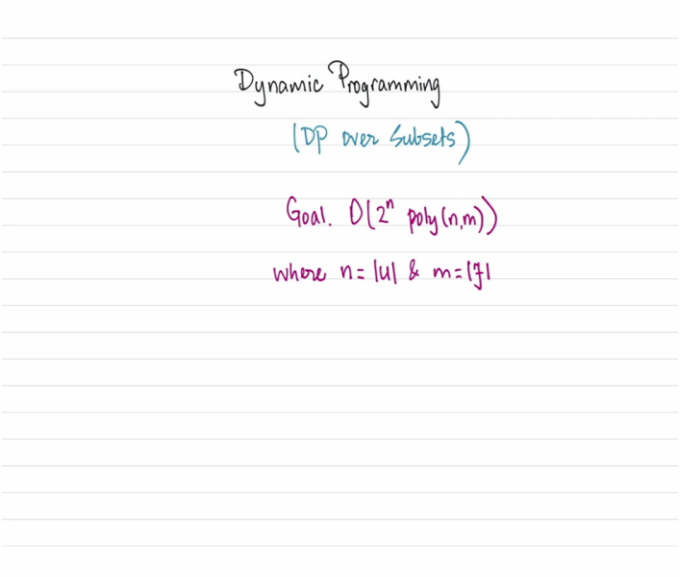
Let us just go over a quick example. So, here we have a universe that consists of 6 elements and we have a family which consist of 5 sets of size 3 each. You have a budget of 3 and I want you to take a moment here to see if you can figure out if this universe has a set cover of size 3 and in fact given the smallness of this example you might even want to see if you can figure out all set covers of size 3 and also answer the question of whether or not this is smaller set cover in particular say with just two sets.

Take a moment here and just see if you could just figure this out. So, you may have noticed that any pair of sets in this family actually intersects at least 1 element which means that there is no set cover with two sets. Let me just write that down. The reason for this is if you pick up any two sets because they overlap at one element and they both have size 3 you are only going to get a total of 5 elements while your universe has 6.

So, that is a quick way of seeing that you cannot have a set cover which is two sets, but what about a budget is 3 which is our original question here. So, maybe we start with the set a, c, e. So, that covers these 3 elements b, c, f additionally gets us b and f. So, we just need a set that contains the element d to complete this to a set cover now. So, for instance, we could pick d, b, f and that would be a set cover of size 3.

So, you could think of this as a yes instance. So, now that we have work for this example let us try and come up with an algorithm for the problem.

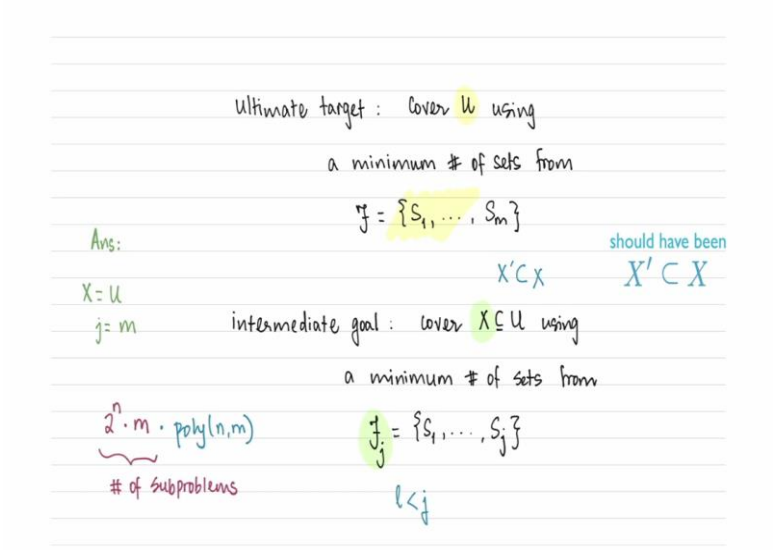
(Refer Slide Time: 05:51)



So, remember we are using dynamic programming for this discussion here and one hint is already in the name that you see here. So, we think of this as dynamic programming over subsets and let me also make the goal explicit in terms of the running time. So, the goal is to come up with algorithm that runs in time 2 to the n with some polynomial overhead in n and m.

So, just to be clear n is the size of the universe and m is the size of the family. So, given this format may be you want to take a moment here to think about what could DP table be and then we could get to trying to figure out how do we want to populate it. So, hopefully you had a moment to reflect on this. Let try and see what would a reasonably natural approach be?

(Refer Slide Time: 06:57)



So, let us recall what are ultimate target is? We want to cover the universe u using a minimum number of sets from the given family f which is sum collection of m sets. So, this point there could be multiple approaches to trying to setup a DP table here. So, I will describe one of them that I think is quite natural. So, in the spirit of looking for an intermediate sub problem instead of trying to cover the whole set.

We will just try to cover a subset that is a smaller goal to work with and these smaller goals will eventually build up to helping us figure out the solution for the whole problem. So, we want to cover some subset and we are going to just focus on whether some subset can be covered using some partial part as a family that we have been exposed to so far. So here is a natural intermediate goal or if you like you could think of this as a sub problem. You want to cover only some subset of the universe using truncated portion of the original family which we are calling f j.

So, this consists of the first j sets in the family f. You could order the family in any way you like so that this notion of the first j sets makes sense. So, quick question for you here how many sub problems have we generated with this definition? Well, we have a choice for every subset of the universe. So if our universe is an element we already have 2 to the n choices for x and we also have a choice of how much of the original family we are allowed to use up.

So, we have m options for the index j here. So, 2 to the n times m is simply the number of sub problems that we are working with and if you think about whether these sub problems are worth solving where we want to note that these final answer will lie in the sub problem

corresponding to X being equal to the universe and j being equal to n. So, this means that this intermediate goal actually manifest our ultimate target which is to cover all of u using a smallest number of sets from all of f.

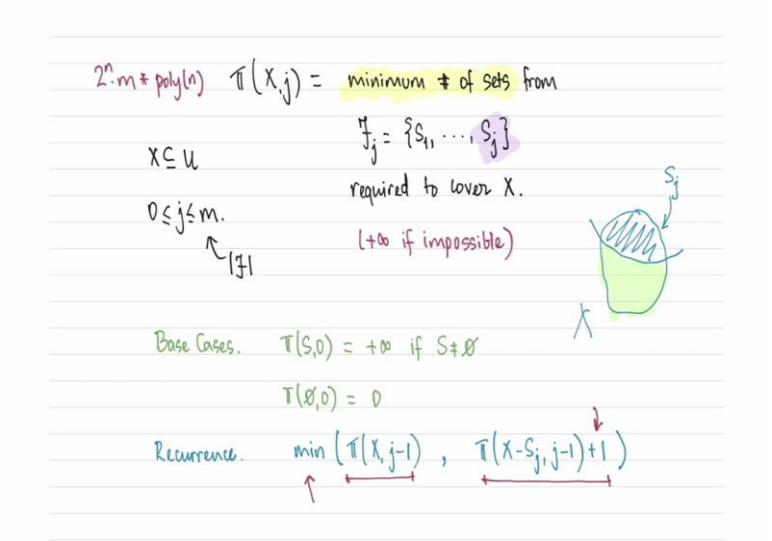
So, if we are able to compute the answers to these intermediate sub problems then we would in fact be done because by the time we get to the end of it we have solved the whole problem. So, just to recall how this kind of work; given that we; have a DP based setup. The idea is that when we are trying to solve the intermediate goal with respect to X and j we are going to assume that we already have access to the answers to the intermediate problems corresponding to X prime and l for all choices of X prime that are subsets of X and for all choices of l that are strictly smaller than j.

So, we assume that we have these answers at hand and they can looked up in constant time and given that we want to figure out what is the answer for these sub problem corresponding to the pair X and j. So, that is essentially the task of coming up with the DP recurrence and to make the description of the program complete of course we want to work out the base case and we know that at the very end the answer that we report is simply the answer to the goal corresponding to X equals u and j equals m like we just discussed.

So, if these intermediate goals can be computed either in constant time or let us say even some polynomial time then the overall running time is going to the amount of time that you need to solve each of these intermediate goals with an overhead of 2 to the n times m because that is the number of sub problem that we have. So, hopefully this overall outline make sense and if it does I would encourage you to really take a pause here and try to figure out what the recurrence should be.

And then we can exchange notes when you are ready to come back. Hopefully, you have had a chance to think through the DP recurrence a little bit.

(Refer Slide Time: 12:00)



Let me begin by rephrasing this business of intermediate goals in the more standard language of DP tables. So, let us say that we have table entry corresponding to pair X, j and what we want this table entry to record is the minimum number of sets from f sub j which if you which if you recall was the first j sets required to cover all of X and let me just remind ourselves that X is a subset of the universe and j is the sum number that ranges from 0 to m. So, m again is the size of the family that you are working with.

So, of course it is possible that there is no such collection of sets it so it does not make sense to talk about the minimum number. It is possible that this is somewhat ill defined. So, we are going to notationally use plus infinity to record the fact that it is impossible to cover X using sets from S 1 through S j. So, with all that said we can get started with the base cases. So, the smallest value that j can take on is 0.

So, let us try to figure out what T of S, 0 will be for any subset S. Now, clearly if S is non-empty then we cannot possibly hope to cover it if we are not really given anything to work with. So, f sub 0 is the empty family. So, if S is non-empty then the answer here should be here should be plus infinity if S is not the empty set and of course the empty set is related to (()) (14:16) subset of the universe.

So, if you are working with the empty set then 0 is okay because you have no work to do and nothing to be done. So, in this case the answer is simply 0. So, that is the base case. Now, let us think about the recurrence. Now to figure out what the recurrence should be let us take a

moment to imagine what our solution looks like. There are two possible scenarios that we would encounter especially relative to what is going on with this set S j.

It is possible that your solution does not use the set S j. In this case it is actually a way of covering X using only the sets from 1 through j – 1 and in fact it must be the best possible way to do this because if there was an even better way of doing it than that would have been what would have witnessed your solution as well. So, in case your solution does not pick the set S j if the best way to cover x using sets from S 1 to S j in fact is only using sets from S 1 to S j – 1 then you can actually look this up and something that you have computed already.

So, you could just try to cover X using just the first j – 1 sets. So, that is one possible situation that you could be in if you are in this situation then the answer is what is there in the table entry corresponding to the pair X, j – 1. On the other hand the other situation you could be in is that you do use S j in your solutions. So, there is an optimal solution which involves the set S j then you can think of this solution as being broken up into two parts.

So, you have this set S, j here and you have the rest of the set X being covered by elements by sets from S 1 through S j – 1. So, this rest of X is really just X – S j and we already again know what is the best way of covering X – S, j using sets from S 1 through S j – 1 so that is going to be this table entry corresponding to the pair X - S, j, j – 1 and our solution is going to be basically a combination of this solution along with the set S, j.

So, it is going to be whatever number you get from this table entry plus 1 to account for the fact that you have added the set S, j. So, now you have these two possible scenario you of course do not know which one could response to the truth, but what you can do is certainly compute these two values. And then you know that the truth must correspond to the better of the two values because you know that you are looking for the best possible way to cove X using sets from S 1 through S j.

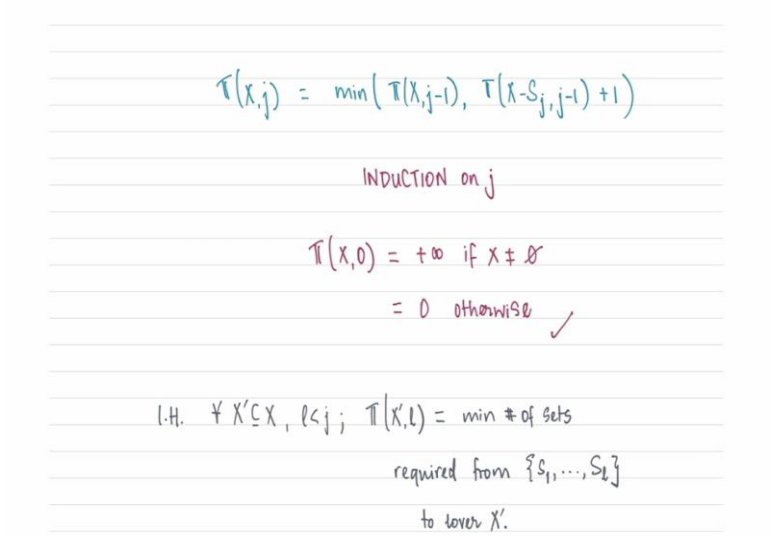
So, the final answer that your report is simply going to be the minimum of these two numbers that you compute here and with that I think we now have a full specification of the algorithm and now that we have completed the recurrence this really is it. And you also know that the running time of this algorithm is going to be the number of table entries which is 2 to the n times m multiplied by the amount of time that you need to compute each table entry.

Now the computation of each table entry can be derived by just starring at this recurrence for a bit. So, essentially you have a couple of table lookups and then you have a couple of operations here and you also would need some time to compute X – S j. So, it is a simple table lookup, but you do need to go through X and it is all the elements that belong to S, j so that is going to be some work.

But all of that is really just polynomial overhead so I am going to leave out with that. Now, finally notice that the correctness of this algorithm really hinges on the accuracy of the recurrence that we established here. And hopefully the correctness of this recurrence was reasonably clear at least intuitively based on the discussion that we have when we came up with it.

But nonetheless just to keep this a self contained argument let us go ahead and actually prove this a little more explicitly.

(Refer Slide Time: 19:00)

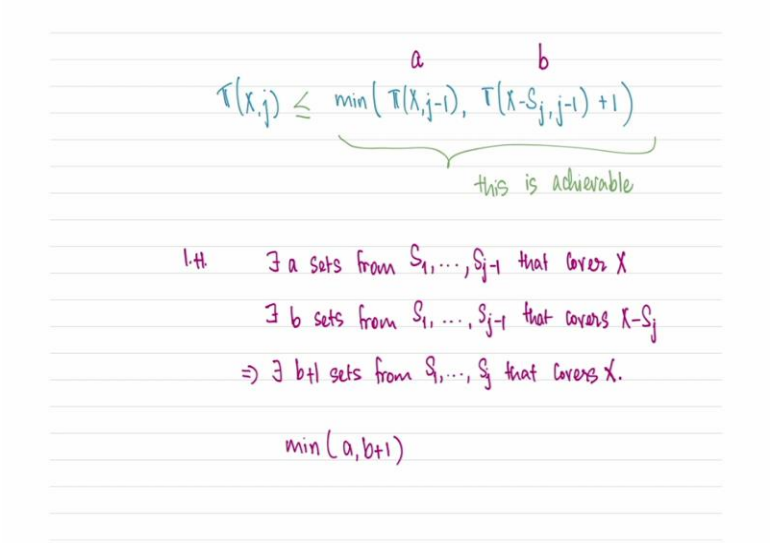


So, here we have the recurrence that we used in the algorithm and the formal framework that we want to use to establish the correctness of the recurrence is going to be induction and in particular we want to perform induction on j. As always with induction you want to start off with the base case. So, the base case in this algorithm was really quite straightforward. We said that X, 0 basically evaluates to impossible if X is non-empty and 0 if X is an empty set.

And this practically nothing to prove here so this is fine. Now to be able to actually prove the main claim we need some sort of a meaningful induction hypothesis. So, what we are going to say is that the claim is true meaning that T of X, j basically does carry the value claimed by its semantics for all j that are smaller than the current value of j. So, in particular T of X prime, l is indeed the best possible way of covering X prime using the first l sets for all X prime subset of X and for all l strictly smaller than j.

So that is the induction hypothesis which again is natural given that we are doing induction in j. Now to prove the equality we are basically going to break this up into two inequalities.

(Refer Slide Time: 20:25)



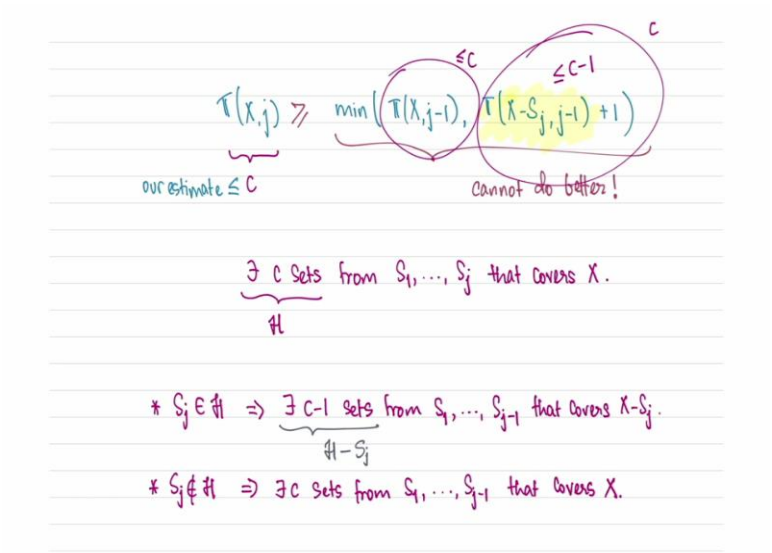
The first one is that the left hand side is less than or equal to the right hand side which is to say that the right hand side is a good upper bound for the left hand side. So, I am going to keep calling the right hand side our estimate for T of X, j and this inequality is basically saying that this is a realistic estimate you can actually come up with a set cover that has the value whose size is the value given by our estimate.

On the other hand the other inequality is some sort of a lower bound saying that this estimate is actually tight you cannot do any better than what has been claimed here. So, these two inequalities combined will finally give you what you want. Now first let us show the upper bound. So, let us just give these values some names. So, let us say T of X, j – 1 is a and T, X – S j, j – 1 is b.

So, based on the induction hypothesis what we know is that there is a collection of a sets from X 1 sorry this should have been I think S 1 so S 1 to S j – 1 that cover X and similarly also by the induction hypothesis we know that there is a collection of b sets from S 1 to S j – 1 that covers not X, but at least it covers X – S j and the second statement implies that there is a collection of b + 1 sets from again S 1 through S j that covers X because we could simply take this connection of b sets here and append the set S j to it to get a cover for all of X.

So, that is what we have from the induction hypothesis. So, we have solutions at our disposals that have sizes a and b + 1. So, in particular we do have a solution whose size is the smaller of the two because we have both of them. So, clearly this is an achievable estimate for sure.

(Refer Slide Time: 22:53)



Now the next thing we want to say is that this estimate is tight in particular we cannot do better than what we have on the right hand side. To establish this side of the inequality let us begin by talking about what is on the left hand side. So, let us call this number c T, X, j. So, just by definition or semantics what this means is that there is a collection of some c sets from among S 1 through S j that covers all of X.

Now let us call this collection of c sets H and let us ask ourselves like we did when we were coming up with the recurrence where S j belongs to H or not. Now, if S j does belong to H then what that implies is that there is a collection of c – 1 sets from S 1 through S j – 1 that covers X – S j and this is simply witnessed by H – S j. So, H was a collection of c sets and we knock out one from it and we are left with c – 1.

And those c – 1 sets of course come from S 1 to S j – 1 and they cover X – S, j. So, this implies that in this case at least what we know is that T of X - S j, j – 1 is going to be at most c – 1 because you can definitely do this with c – 1 maybe better, but certainly c – 1 is a valid upper bound. So, in this case this quantity here is going to amount to c. On the other hand it is possible that S i does not belong to H.

But this is even more direct because this implies that there is some collection of c sets from S 1 through S j – 1 that in fact covers all of X and in this case we know that this quantity is going to be at most c. So, with this what we have is that no matter what the scenario is the expression on the right hand side is going to be at most c could be something that even better if the quantity that is not under the consideration actually overrides the value that is under consideration.

Since we are taking the minimum of both the values involved, but hopefully you can see that no matter what happens the expression on the right hand side evaluates to being at most c. So, in particular we could say that our estimate is going to be at most c, but that is exactly what we wanted to show. So, that completes this direction of the inequality and in fact with that we actually complete the entire argument.

So, this wraps up our discussion on set cover and we are going to look at dynamic programming and action in the context of another problem called Steiner tree. So, that is coming up in the next segment of this video. So, I will see you there.