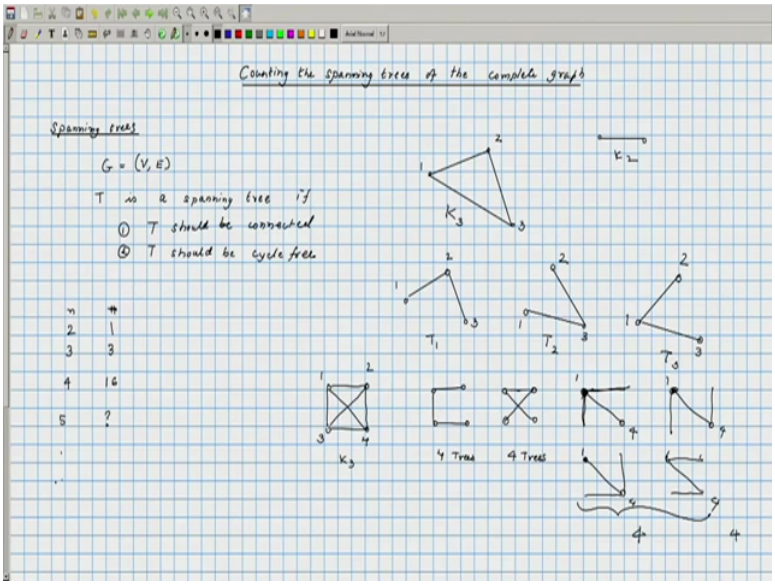
Discrete Mathematics
Professor Sajith Gopalan
Professor Benny George
Department of Computer Science & Engineering
Indian Institute of Technology Guwahati
Lecture 18
Counting Spanning Trees in Complete Graphs

In today's lecture we will learn about counting the spanning trees of the complete graph.

(Refer Slide Time: 0:37)



So let us understand what this means, so we have talked about spanning trees before, so suppose G is a graph, a spanning tree T should satisfy two conditions; the first condition being T should be connected and the second condition is T should be cycle free ok so it is basically a tree such that every vertex there is a path from every vertex to every other vertex, so you can say that it is a minimal such graph, minimal in the sense of number of edges required ok.

So this is the definition of a spanning tree and we are looking at the complete graph on n vertices, so let us say if we have taken 3 vertices this would be the complete graph on 3 vertices which we represent by K 3, if you number the vertices as 1, 2, 3, this graph has many spanning trees in particular if we just try to draw every spanning tree of this graph will have exactly 2 edges so 1-2, 2-3 is one possibility so this is T1.

1-3, 2-3 will be the second spanning tree and there is yet another spanning tree which is 1-2 which consists of the edges 1-2 and 1-3 ok so we can verify that these are the only spanning trees for K3. So when K equals 3, the numbers will be 3, if we had looked at the simpler

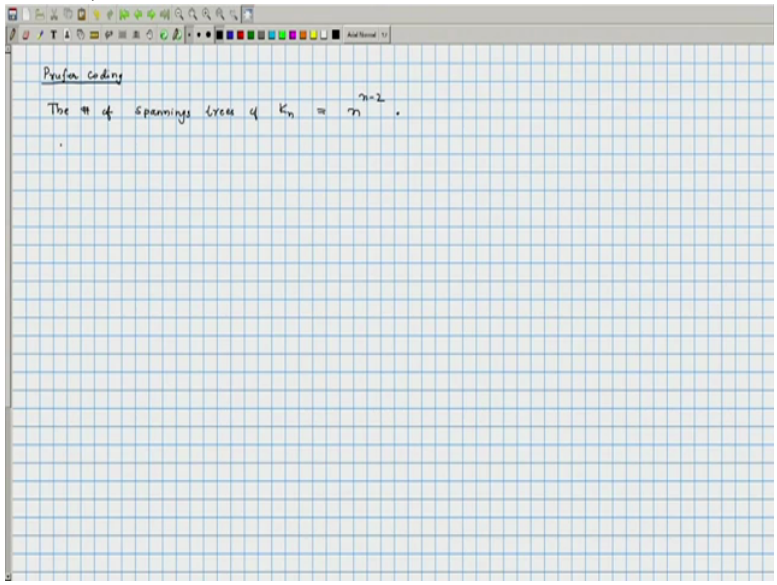
example of K2 it is precisely one spanning tree. So we can say for N equals 2 the value is 1, for N equals 3 let us see what is the case when N is equal to 4. So we are looking at this particular graph, the complete graph on 4 vertices, how many spanning trees does this particular graph has? So we can enumerate them so one would be of this kind so all the spanning trees of K4 will have precisely 3 edges.

So the ones which consist of the boundaries or the square there will be 3 of them, so there is a missing edge here and there are 4 ways you could choose them so there are 4 trees of this kind, and if you include the diagonal ok so you can include both diagonals, so these 4 are without including diagonal, now the choice becomes do you include 2 diagonals or do you include only one diagonal okay. So let us just look at including 2 diagonals, so these are the 2 edges and the third edge that you want to add could be any of the other 4 okay, because each of them so if you had taken this that will give you a spanning tree and all those 4 choices would give you valid spanning trees so there are 4 trees of that kind.

And if you choose to include just one diagonal okay so let us say if you are using the 1-4 diagonal, the other case would be symmetric ok. Now we can choose either these two, this is yet another case so these are the 4 spanning trees that we will get if we choose to add the 1-4 edge. If we insist on having the 1-4 edge you could add 2, the degree of this particular vertex 1 could be 2 in which case the only possibility is this, the degree could be 0 in which case this is the only possibility, the degree is 1, there are 2 possibilities namely these two. So there are 4 trees when you choose to include 1-4 diagonal and the symmetric case would be when you include the 2-3 diagonals there are again 4 nodes.

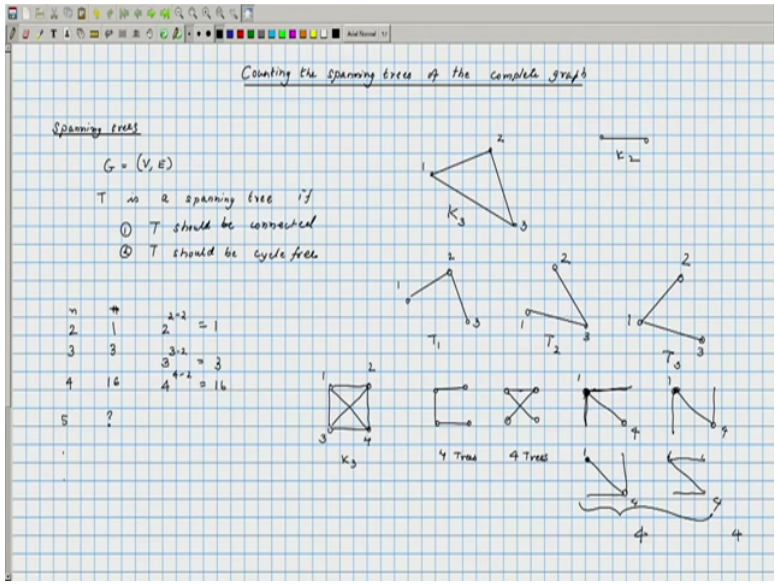
So the case where n is equal to 4, the number of spanning trees is going to be 16. You can imagine that it is going to be extremely time-consuming if we were trying to do the same thing for n is equal to 5, so what will be the number when n is equal to 5 and so on, so this is what we need to understand ok. We need to count the number of spanning trees of the complete graph on n vertices ok.

(Refer Slide Time: 6:21)



And we will do it by a method known as Proofer coding ok. So you can look at these numbers and see if you can detect a pattern there in any case from Proofer coding what we know is the number of spanning trees of K n. So we are looking at labelling K n okay so these are, and after we have obtained the labelling how many distinct spanning trees are there that is what we have count. The number of spanning trees of K n is equal to n raise to n minus 2, this is the term that we have to prove.

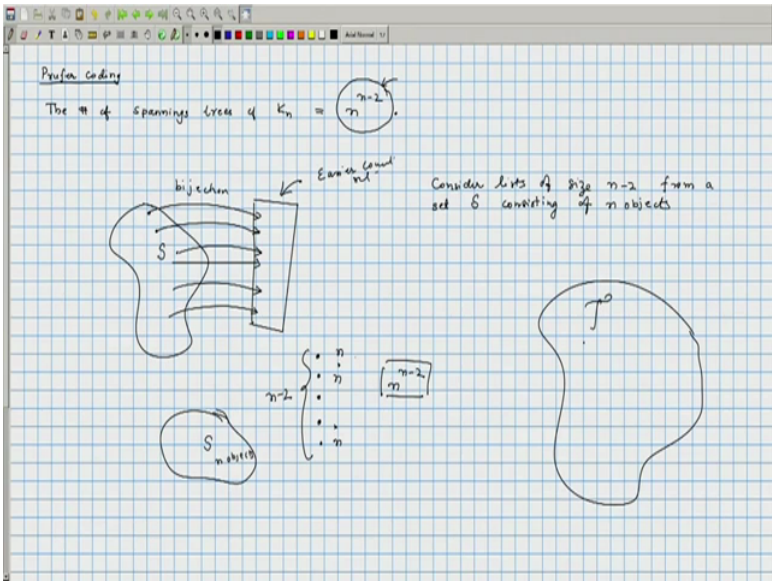
(Refer Slide Time: 7:19)



So the count is n raise to n minus 2, you can verify that when n is 2, 3 and 4 so 2 raise to 2 minus 2 that is going to be 1, 3 raise to 3 minus 2 is going to be 3, 4 raise to 4 minus 2 is

going to be 16 ok, so all those cases we have verified we need to prove this in the general case.

(Refer Slide Time: 7:42)



While proving this we will use an interesting technique we will basically setup an isomorphism between two sets ok. So let us look at this problem, you are given a particular set S and you want to count the number of elements in it. One way to count is to explicitly count the number of enumerate the elements in S and then see what is its count. Another way could be to setup a correspondence ok, so you can set up a correspondence with another set which is easier to count ok, so this we can think of as an easier to count set. If we can find an isomorphism which maps every element to a unique element in the other collection okay, in such a way that every element of the other set is occurring as the image of at least one of the elements.

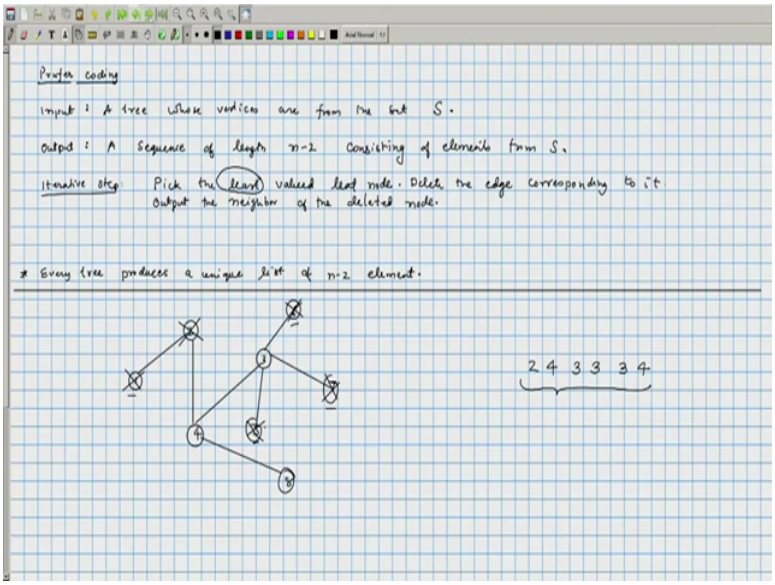
If we can set up a bijection between these 2 sets then we know that the counts are going to agree. So there are many natural objects whose count is n raise to n minus 2, we will show that the spanning trees can be put in bijection to one of those sets. So one natural candidate for set having n raise to n minus 2 elements would be if you consider list, we will think of this as an ordered list of size n minus 2 from a set S consisting of n objects ok.

So we have a set S, let us call this set as S, now this set has n objects, and we are allowed to form list of these and the length of this list is n minus 2, the first element of course can be chosen in n ways and we are allowing repetitions, second element also has n possibilities, n

minus 2 elements can also have n possibilities so multiply and therefore the total number of ways of having this list clearly is n raise to n minus 2 ok.

So this is one of the quantities that we have, now we have this collection of the set, our main set that we are interested in let us call it as T, T consist of all the spanning trees of K n ok. So if we are able to find the bijection such that each such list corresponds to one unique element of the set of spanning trees then what we know is that the counts match ok, and this bijection is basically known as the proofer coding ok, so let us see what is this bijection.

(Refer Slide Time: 10:58)



The bijection will be given via means of an algorithm. Ok so we will assume that we are given a tree, so input is a tree whose vertices are from the set S ok, and we are going to assume that the set S is an ordered set in the sense given 2 elements in the set we can say which is the larger element, which is the smaller elements and so on ok, so we will assume that there is a total order on the set S.

And the output will be a sequence of length n minus 2 or a list of n minus 2 elements, so proofer coding basically when given an input tree it will produce an output sequence. What we will need to show is that every sequence will be generated by precisely one tree. When we look at this encoding, when we look at the algorithm for encoding we can see that if you are given a tree there is only a unique sequence that can be produced.

Sequence produced will be unique, but what we will argue is that for every sequence there is a tree which generates that as well, and there is precisely one tree which will generate that. So

let us see what encoding is, so the proofer code is generated in an algorithmic fashion, so this is the iterative step okay so we will repeatedly do this iterative step for n minus 2 steps.

What we do is, pick the least valued leaf node ok so you are given a tree so the tree must surely have some leaf nodes, amongst the leaf nodes find the smallest valued leaf node, delete that node along with the edge that it is along with, the so since it is a leaf node, it is connected to one node, delete that particular edge and output the neighbour of the deleted node ok.

Since it is a leaf node, its neighbour is unique, it is precisely one neighbour so that neighbour we will output, and we repeatedly do this for n minus 2 steps, we get a sequence of n minus 2 elements. So the easy factors, every tree produces a unique list of n minus 2 elements okay. So if you give a tree the answer is unique, but it could happen that two trees could give rise to the same list, it could also happen that certain list cannot be produced by any tree, we will rule out both these cases.

This fact is obvious, the fact that every tree produces a unique list is obvious because the algorithm has no randomness and at each step since the least valued leaf node is picked that is unique and its neighbour is also unique and therefore, if you give a particular tree the output list is automatically fixed, let us work this out for an example.

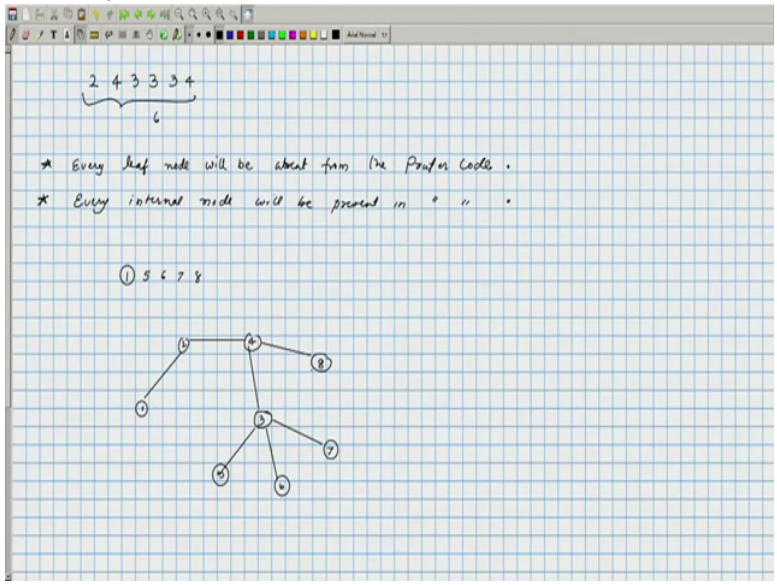
Ok, so suppose we look at this particular tree, the leaf nodes are nodes 1, 7, 5 and 6. And the least valued leaf node is 1 so we will delete that and its neighbour is 2 so that is what we will first note ok. And now if we will look at the remaining tree, the least valued leaf node is the node number 2 or vertex number 2 and its neighbour is 4 so we will remove this and we will write out the number 4.

Now, amongst the remaining nodes the leaves are 5, 6, 7 and 8, the least being 5 and its neighbour is 3 so we will basically remove 5 and after that we will remove 6, 7. And after we remove 6 and 7 and both their neighbours with 3 so when 6 is removed we get a 3, when 7 is removed we again get a 3 and at this point the tree consists of 3, 4 and 8. And if we look at the leaves, there are precisely two leaves 3 and 8, the least valued one is 3 and its neighbour is 4 ok, so this is the sequence that we get ok.

Now what we will claim is, if we know the underlying vertex set, from that information and from these numbers 2, 3, 4, 3, 3, 4, we can recover back the tree. So I hope this process is clear, we can just the iterative step is at each stage that the least value of the leaf node deletes

the edge corresponding to it and the node the neighbour of the deleted node. Okay so let us just look at the sequence, 2, 3, 4, 3, 3, 4 and try and generate the tree back.

(Refer Slide Time: 17:45)



Ok so suppose we know just this information, and we do not have access to what was the original tree, from this can we recover back the tree. If we look at the sequence, one thing that we know is there are 2 facts that we can know, every leaf node will be absent from the proofer code and the second aspect is every internal node will be present in the proofer code ok. Why is this so? The leaf node the only numbers which appear in this are internal node because leaf node is the node that you are going to remove and when it is removed the node itself is never returned, but its neighbour is returned.

So anything which appears in the proofer code at some time it will have to be an internal node, so we can clearly see that the leaf nodes will be surely absent from the proofer code. If we look at the process of generating the proofer code, we are starting at a tree and we are systematically removing the edges, so there will be a point at which any node, its degree will keep on decreasing and there will be a point when it becomes either 0 or 1.

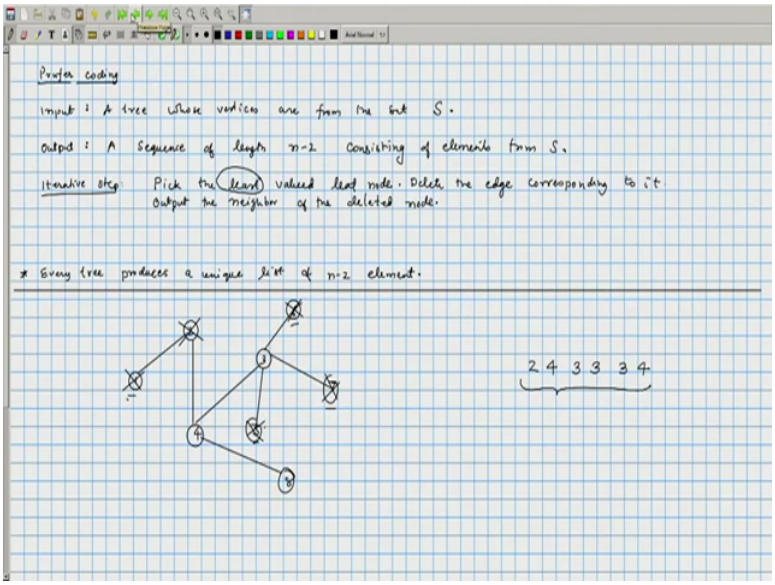
The process of generating the proofer code we took at tree and we were removing the edges one after the other, if you look at the initial tree, the process of generating the proofer code basically removes an edge from the tree one edge at a time. So every internal node is going to lose some of the edges corresponding to it, and when this happens that node is going to appear in the proofer code and therefore, we can argue that every internal node will be present in the proofer code ok.

If you note these things, we know that by looking at the proofer code 2, 4, 3, 3, 3, 4 that the leaf nodes essentially have to be the missing numbers. So this is the sequence of length, the length is 6, and if the vertices were numbered 1 to 8, the leaf nodes would have been 1, 5, 6, 7, 8 ok, we can look at the diagram and see that this is indeed the case, 1, 5, 6, 7 and 8. And amongst these, the first one to be removed would have been 1 and that should have been connected to 2, so 1-2 would have been an edge.

And if you look at the remaining sequence now 4, 3, 3, 3, 4. 2 is missing and therefore, 2 must have been a leaf node for that particular tree so we can say that 2 must have been connected to 4 ok, 4 is still present and amongst the numbers that are not present the least one is 5 and 5 is to be connected to 3.

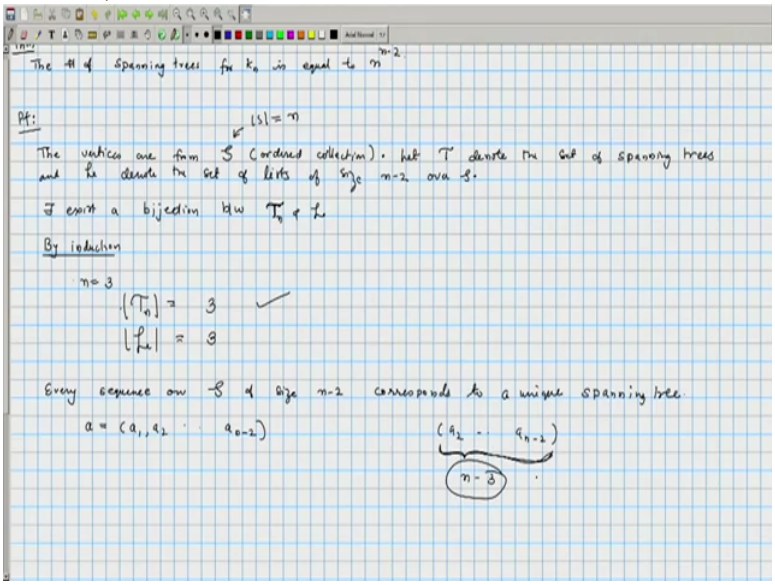
The next missing number is 6 so 6 is also connected to 3 and then you have 7, 7 is also connected to 3. And after this point 3 is no longer going to be present, and therefore 3 must have been connected to 4 and then at this point there is only one node that is remaining that is 8 and that must have been connected to 4 ok.

(Refer Slide Time: 21:39)



So from this number we could in some sense generate the tree, we have been given a formal proof we will do that shortly. Vertex 3 is connected to 5, 6, 7 and vertex 4 is connected to 8 and 2 and vertex 1 is connected to 2 that was all the. So at least in this case we have verified that things work nicely, so let us formally prove the theorem.

(Refer Slide Time: 22:05)



The number of spanning trees for K n is equal to n raise to n minus 2, so let us assume that the vertices are from a set S which we will assume to be an ordered set. And let T denote the set of spanning trees and L denote the set of lists of size n minus 2 where we will assume that the size of S is equal to n, so the list of size n minus 2 over S ok.

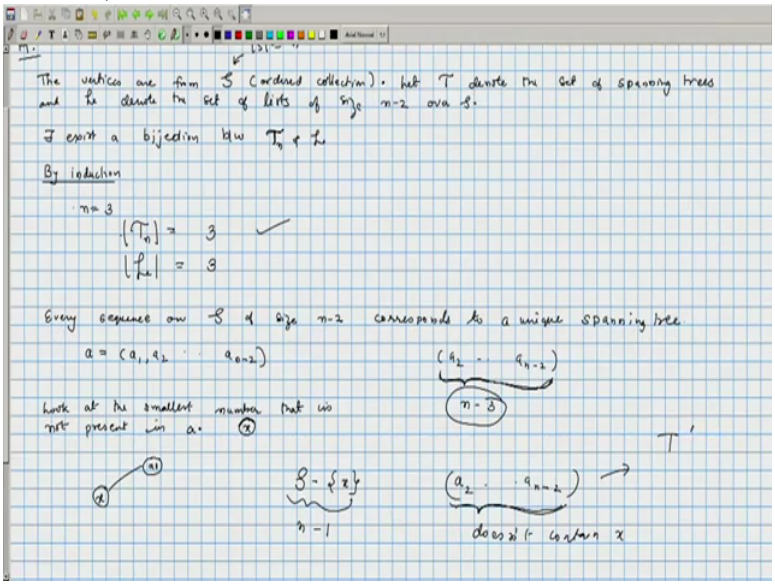
What we will prove is there exist a bijection between T and L, so we will prove this by induction, base cases we have already checked and n is equal to let say 3, we had verified that T contains, if we call this as T n, T n contains 3 elements and the list also contains 3 elements. So because what you are looking at is list of size 3 minus 2 that is list of size 1 over S there are precisely 3 elements in S so the list of size 1 are precisely 3 so the base case can be easily checked.

Now, let us assume that the induction hypertrophy holds for all values up to m minus 1 ok. Now we want to show that for m also this statement is true ok. So let us look at particular sequence, so every sequence on S of size n minus 2 corresponds to a unique spanning tree ok. So let us look at a particular sequence, so if you call the sequence by a, that is equal to a 1, a 2, a n minus 2, we will exhibit a tree corresponding to this and we will argue that that tree is a unique tree.

Now, if you look at let say a 2 to a n minus 2, that again as a sequence of length n minus 3, it is a subsequence. And if we knew which are the vertices that are under consideration here, we will inductively argue that a unique spanning tree can be constructed corresponding to that.

To that spanning tree we add a particular edge and we will get the tree that we are interested in, so that is our line of attack, so let us look at it more carefully.

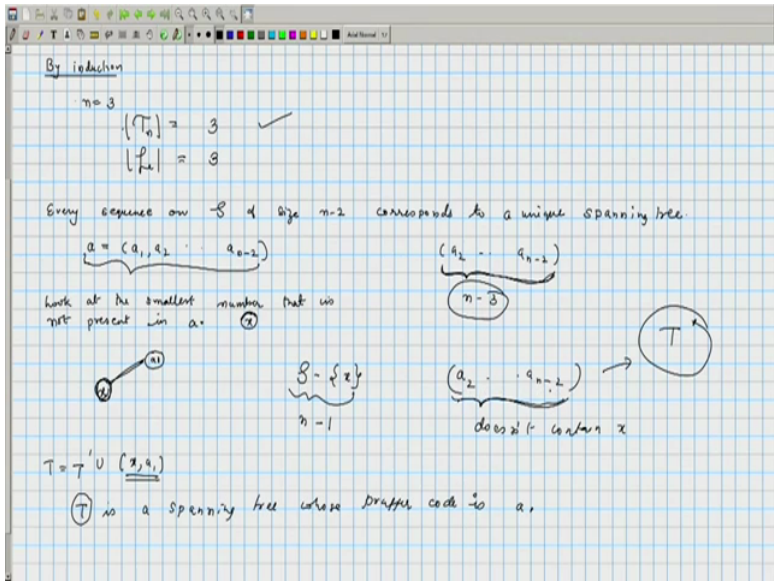
(Refer Slide Time: 25:58)



So look at the smallest number that is not present in a ok, let us call it as x. We had argued earlier that the numbers that are missing in a 1, a 2 up to a n minus 2 has to be clearly leafs, and the smallest of them would have been the first node that would have been removed when we were constructing the spanning tree. So if you look at that node and call it as x, clearly x is the node x is the leaf node that was removed initially and x would have been connected to a 1.

Now, if we look at the set S minus X, this is the set of size n minus 1 and if you look at the subsequence a 2 to a n minus 2, this sequence cannot contain x in it. This does not contains x in it because the original sequence itself did not contain x so subsequence surely cannot contain x.

(Refer Slide Time: 27:41)



And by our induction hypothesis, this part a 2 to a n minus 2 is going to correspond to unique tree T prime ok. Now that tree with the edge x a 1 will correspond to this particular sequence, so T union the edge x a 1, T prime union x a 1 if you call this as T, T is a spanning tree whose proofer code is a, so why is this so? Well, clearly if you add x a 1 to T prime, it does not create any cycle because we assumed that x is not present here so that would make x a leaf, and if you attach a leaf to some other node that is now going to create a cycle.

So the newer so the graph that you get is indeed going to be a tree, and if you look at the proofer code of that that is precisely going to be a 1, a 2, a n minus 2 ok. So this is basically the proof which says that corresponding to every sequence you can get a unique spanning tree, the uniqueness comes out because a 2 to a n minus 2 by induction hypothesis has a unique T prime, and the only tree that is going to be obtained is by adding x a 1 to T.

So the only tree that could possibly correspond to this sequence, the only graph that could possibly corresponding to correspond to this particular proofer sequence or this particular code is going to be T that is obtained by adding the x a 1 edge to a 2 a n minus 2 ok, so that concludes the proof and the lecture.