Design and Analysis of Algorithms

Prof. Sunder Vishwanathan
Department of Computer Science Engineering
Indian Institute of Technology, Bombay

Lecture - 13
Greedy Algorithms – IV

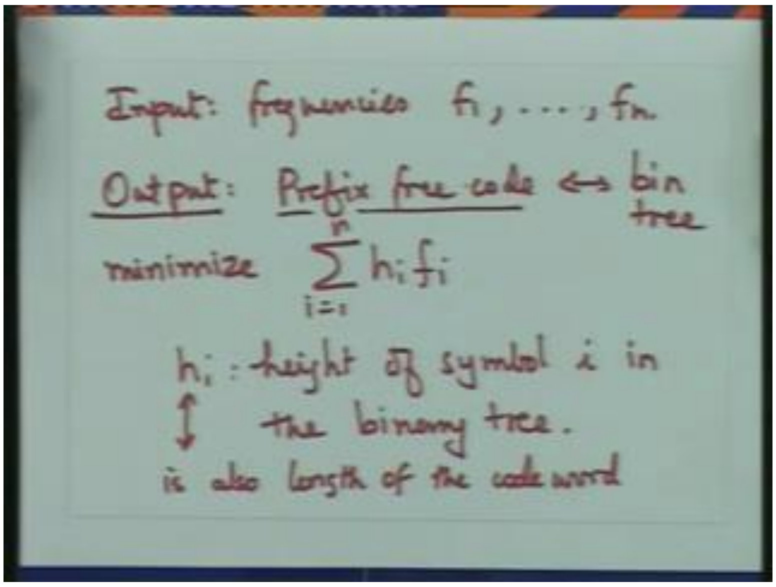
We will look at this problem from information transmission. Let me recall the problem, we had n symbols x 1 through x n. And each at a frequency associated with it, which was the frequency of occurrence. We wanted to construct the code for this set of symbols. So, for each symbol we read the binary string. Now, if the codeword such that ((Refer Time: 01:26)), length of these string, whether same for all of them.

Then, there is really no problem there is we do not have a problem to solve, but if you allow the coding to be such that, these codeword's can have variable length. Then, we do have a problem to solve. Essentially, symbols which appear more frequently, we would like to give a shorter codeword. I just one way to see this is for instance, if you had a large file and you had n words, which these word are repeated.

The all have frequency associated with them, which is nothing but the number of times that the codeword appears in a file, that file. And we would like to encode this file as... So, that in the codeword's are prefix free, which means no codeword is a prefix of another. This is how we want to encode this file and send it across.

Then, the size of the file once encoded the size of the file is nothing but the frequency times for it is some over every word. The frequency of occurrence of the word, times the size of the codeword. And this quantity, which is the size of the file we would like to minimize. So, this is exactly what we had. So, let me put that put it file across.

(Refer Slide Time: 03:08)



Input: frequencies $f_1, \dots, f_n$

Output: Prefix free code $\leftrightarrow$ bin tree

minimize $\sum_{i=1}^{n} h_i f_i$

h_i : height of symbol i in the binary tree.

$\updownarrow$

is also length of the code word

So, the input, the input is a set of frequencies f 1 through f n. And we would like a prefix free code, this is from yesterday. Well, I just get to this binary tree in a minute. Essentially, we want to prefix free code and you want to minimize the following h i times f i, h i here we say it is the height of the symbol i in the binary tree. But, this is the same as a length of the codeword h i is the same is also length of the code word for symbol i.

So, then this tree is nothing but a size of the file, f i is the number of times the word occurs, this is the length of the curve codeword. So, this is the this quantity is nothing but the number of bits used to encode the file. And you would like to minimize this. So, we also saw that prefix free code of prefix free code corresponds to a binary tree. There is a one to one correspondence.

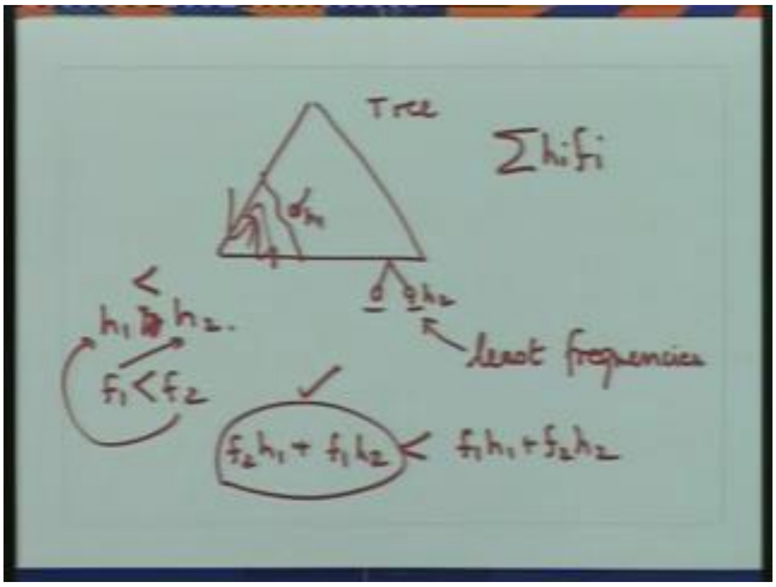
So, given a binary tree and symbols at each leafs, you can construct the prefix free code. Now, this gives you a prefix free code for the symbols. Essentially, when you traverse right, you think of it as one, if you traverse left on the tree you think of it as a zero. You traverse a path from root to this leaf and that gives you a codeword. Each time you going to right you right a one, each time you going to left to right a zero. And when you end up with leaf this code word associated with it.

Similarly, given prefix free code for n symbols, you can construct the binary tree with n leafs for which this is n or more leafs, which as which the same property. So, essentially if your code codeword is say 1 1 1 1 0. Then, you get to this leaf by traversing right, right, right and then left. That gives you the position, similarly for any binary string, you sort of get two a leaf.

So, you traverse right if it is a one traverse left if it is a zero. And then, gives you part of the binary tree, you can fill up the rest of the binary tree if you want. Since, it is prefix free each time you will end up with leaf. So, all these symbols will sit on leafs. So, you can check this using small example. In fact, I encouraged do it, that will help you understand this better.

So, this is what we want to do. So, we want to output prefix free code or a binary tree, which corresponds to this which minimize this something like this. We saw that once this structure of the binary trees fixed. Then, we know how to associate these words to the binary tree, with two with the smallest frequencies, will see that leaves that two with smallest frequencies will sit at leafs, which are at the bottom

(Refer Slide Time: 06:43)



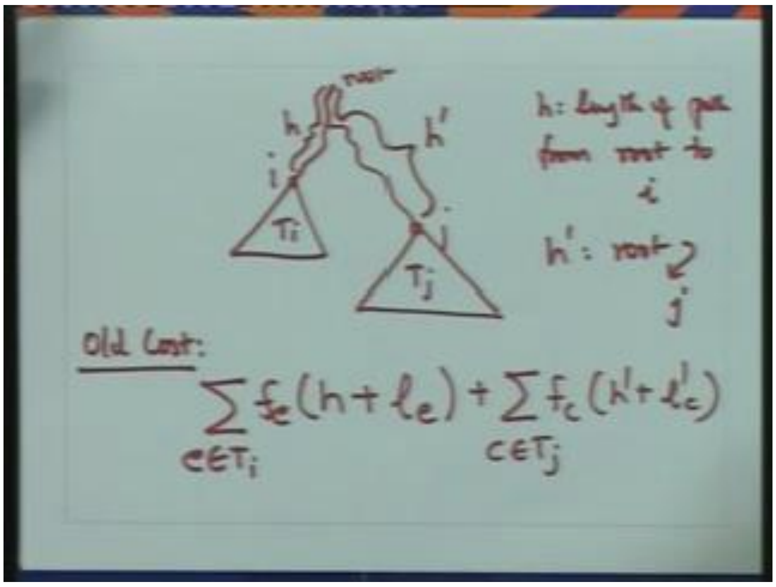
So, here is my tree, so it is a binary tree. And let us say somewhere here at see it is important, it need not be sort of balance you can even be skew like this. Let us say these two are the bottom most leafs, then I know the these will have the least frequencies. These two the frequencies will be the smallest. Because, take supposing it was not the case and you know somewhere here, you had you had a leaf. Now, the quantity of minimizing is sigma h i f i.

Now, if this ((Refer Time: 07:37)) like closure in the h of this was smaller, h of this was larger. So, let us say this is the h 1 and this is the h 2, h 1 smaller than h 2. H 1 is smaller than h 2, this is lower down and that is higher up. Now, you like to put smaller frequency here. So, that if I had f 1, f 2 supposing, let us say this way f 1 was smaller. Then, I could like f 1 this to sit there and this to sit there. Because, then it will be f 2 h 1 plus f 1 h 2 which is smaller. Then, f 1 h 1 plus f 2 h 2, you can check that this is smaller and that.

So, this is what we want, which means things which are smaller in frequency, must be lower down in the tree. Things, which are larger in frequency should be higher of in that tree, which stands to reason. Because, the larger the frequency, this smaller should be the codeword, which means should be higher up in the tree. So, this much we know... So, given the shape of the tree, we can certainly fill it up with these frequencies, that is easily done.

But, what is the shape of the tree, that is the question that we would like to answer yes. So, that is what we would like to answer and we would like to use this exchange trick. So, what you like to do this? Supposing you are given some tree and you filled it up somehow.

(Refer Slide Time: 09:29)



h: length of path from root to i

h': root to j

Old Cost:

$$\sum_{c \in T_i} f_c (h + l_c) + \sum_{c \in T_j} f_c (h' + l'_c)$$

Now, suppose there are these two parts from the root. This is let us say node i, next node j. There is a tree sitting here call it T i and there is a tree sitting here call it T j, here is the root. Now, so what you like to do is exchange these two sub trees. So, remember the or exchange trick was the somehow exchange some part of the output. Supposing, we had some constructed solution, we would like to change twist the solutions slightly and see what happens.

And earlier it was exchanging part to the solution with something, which is outside. Well, here since we are constructing trees, what you like to see is suppose your exchange to sub trees. And what is the result of this action? What happens with this action? What happens to the you know the function we are trying to minimize, which is product to the height times the frequency. So, we would like to exchange these two sub trees and see what really happens.

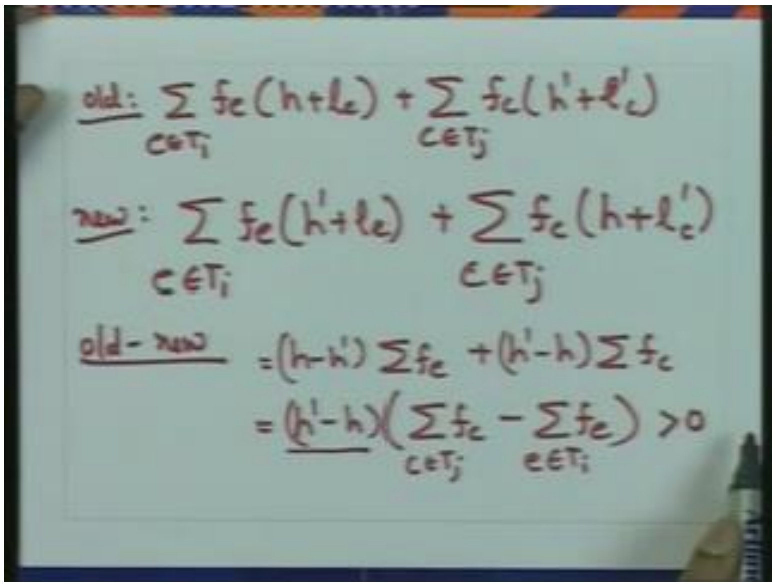
So, let us do this calculation. So, let us say this is at height, so this let us say is height h, let us say this portion is h prime, this is all the way up to the root. So, the root two node I is height h. So, h is length of path from root to i. H prime is from root 2 j, that is the length of the path from root 2 j at h prime. And we would like to now compute this function. So, see when you exchange these two sub trees, the rest of the elements remain that they are.

So, there contribution to this cost remains the same, we are not changing that at all. So, the anything we need to worry about is the contribution change in the cost, because of the exchange. So, let us calculate the old cost, because of these two sub trees. The new cost because of these two sub trees and we will see what the differences. So, here the old cost.

So, let me do it for T i so for every element e in T i. So, I have the frequency times the length of the part from root 2 that element, which is h length root to i plus the length of the path inside this sub tree. Let me call this l e. L e is the length of the path from i to the element e in T i. So, this is the cost old cost for T i and the old cost for T j is sigma.

Let us say f c h prime plus l prime c and c is in T j the similar cost for T j h prime is this l prime c is the length of the path from j to the element c f c is a frequency. This is the old cost of elements in T i and elements in T j. Similarly, we can write the new cost, let me actually I guess I will have to write this again.

(Refer Slide Time: 13:50)



$\underline{\text{old}}$: $\sum_{c\in T_i} f_e(h+l_e) + \sum_{c\in T_j} f_c(h'+l'_c)$

$\underline{\text{new}}$: $\sum_{c\in T_i} f_e(h'+l_e) + \sum_{c\in T_j} f_c(h+l'_c)$

$\underline{\text{old - new}}$ $= (h-h')\sum f_e + (h'-h)\sum f_c$

$= \underline{(h'-h)}\left(\sum_{c\in T_j} f_c - \sum_{e\in T_i} f_e\right) > 0$

So, let me try and write all this in one. So, here the old cost sigma e belong to T i f e times h plus l e plus sigma c belongs to T j f c times h prime plus l prime this is old, a new cost, well here and just exchanging. Let us go back here ((Refer Time: 14:10)) I am exchanging T i and T j. So, this h will remain the same, but here I will have T j instead of T i sigma e belongs to T i. Now, T i shifted from h to h prime. So, it will be f e times h prime plus l e plus sigma c belongs to T j f c remains the same h plus l prime.

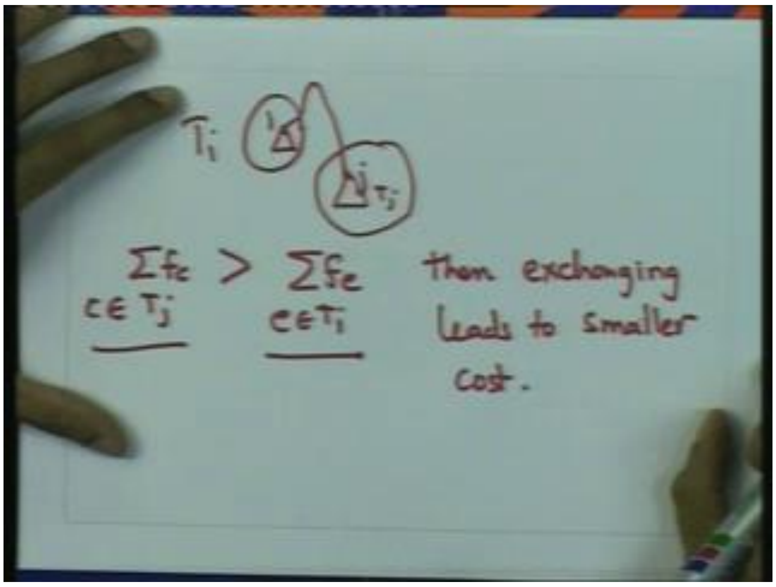
So, this cost this length inside the tree, inside the sub tree remains fixed. But, earlier it was attach to i that is why I add h prime here. Now, it is attach to j, so earlier it was attach to j, that is why it is h prime, now it is attach to i, so it is h. So, this is new cost, so what is difference in cost. So, I want to compute old minus new. So, when I do old minus new, this f e l e cancels in both.

Similarly, f c l primes c cancels, so with f c when I subtract h minus h prime is what remains. So, this is nothing but h minus h prime time's sigma f e plus h prime minus h times sigma f c. So, this is nothing but and this rewriting this h prime minus h times sigma f c minus sigma f e. This is c belong into T j and this is E belong into T i.

So, let us go back from slide h prime I will assume is greater than h ((Refer Time: 16:19)) I have done a shift of this if h prime is greater than h what is the result? So, h prime is greater than h, then this quantity is positive. So, if this quantity is negative, then we have achieve something which is good, the cost as decreased, I mean in the cost as increased. So, this is old minus new. If old minus new is greater than 0 at means the old cost is greater than the new cost, we have decreased the cost.

So, which is what we want, so when is this, this is greater than 0. So, this is greater than 0, then we are in good shape. This fellow is greater than 0, then we have achieve something. So, which means, so let me just write this term, so just conclude.

(Refer Slide Time: 17:33)


T_i i j T_j

$$\underline{\sum_{c \in T_j} f_c} > \underline{\sum_{e \in T_i} f_e}$$ then exchanging leads to smaller cost.

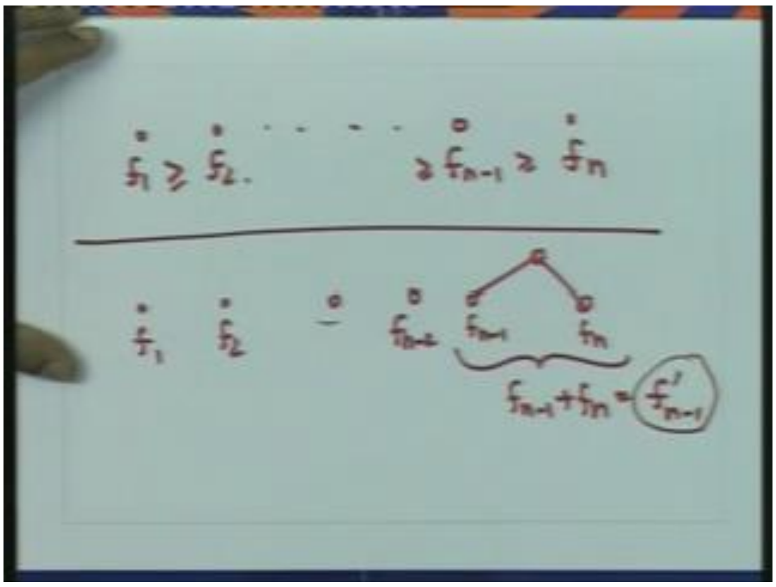
So, here is i T i and somewhere here is j and T j. If sigma f c c belongs to T j is greater and sigma f e e belong to T i c belongs to T j e belongs to T i this is true is then, exchanging leads to smaller cost. So, we have two sub trees, one is this T i that is T i and that is fellow T j. If the some of the frequencies of the elements in T j is greater than the sum of the frequencies of elements in T I, then if I exchange I get a smaller cost.

Remember, it is just this when I look at a sub tree I need to look at only if some of the frequencies inside, that is what these two things it. Only the some of the frequencies nothing to do the length inside the tree, etcetera. I do not care what this tree looks like. As long as the some of the frequencies of elements inside this sub tree is greater than this I should move it up.

That is what the exchange trick tells us and our algorithm. In fact, we will use this fact. So, when I look at a sub tree the quantity that I am going to keep looking at is the some of the frequencies of elements in this sub tree. Not the average length or anything of that kind. So, this will be the mantra that we will follow during the course of this algorithm.

So, now what so we would like to follow greedy approach, which is we would like to build this sub trees one by one slowly. Initially each element will be a sub tree of it is own. So, I am going to build this sub trees bottom up, slow increase the size of the sub trees that I build slowly. Initially, each element will be a sub tree of it is own this is the bottom thing and I know the first step.

(Refer Slide Time: 20:25)

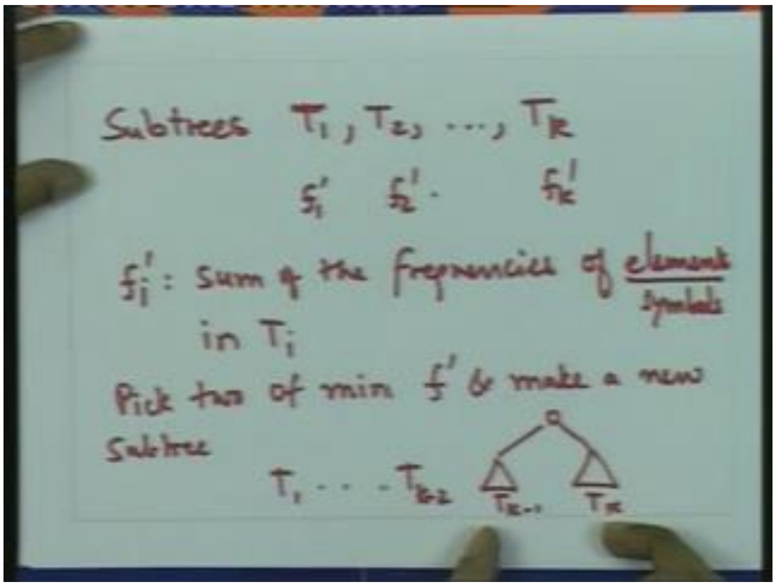


So, if I have let us say I have these elements, let us say this as frequency f 1, f 2, f n minus one and f n. Suppose these are the frequencies and let us say, they are in decreasing order. So, f 1 is greater than equal to f 2 and so on two of the smallest once are f n minus 1 and f n. Then, I know part of the sub tree, I know that my next step I can have f 1, f 2 and so on. I can join the last, this will be a sub tree I know. So, this is f 1 minus 1 and this is f n.

The question is what next? Now, the crucial trick is rather than treat them as leafs, I treat all of them as sub trees. This is one sub tree, this is another sub tree well a leaf is a sub tree. This is sub tree, that is a sub tree, this is sub tree. Now, which are the two sub trees that I want to be at bottom. Remember, the previous exchange thing as the bottom I want the sub tree with whose some of the frequencies of the nodes inside it must be the smallest.

So, I now look at these frequencies I have f 1 f 2 and so on. So, this is f n minus 2. Now, this I treat us one sub tree and it is f n minus 1 plus f n. So, this is my new f prime let say n minus 1. These frequencies remain the same, now this I look upon as a sub tree with this frequency. Among these I chose two of this smallest and I put a new node and join them together and this is my algorithm. So, I just keep doing this as I go up. So, my generic step of the algorithm is this, I have sub trees. So, initially treat each element is treated as sub tree and as I go long I will have may sub trees.

(Refer Slide Time: 22:59)



Subtrees $T_1, T_2, \ldots, T_k$

f_1' f_2' · f_k'

f_i': sum of the frequencies of $\frac{\text{element}}{\text{symbols}}$ in T_i

Pick two of min f' & make a new subtree

$T_1 \ldots T_{k-2}$ T_{k-1} T_k

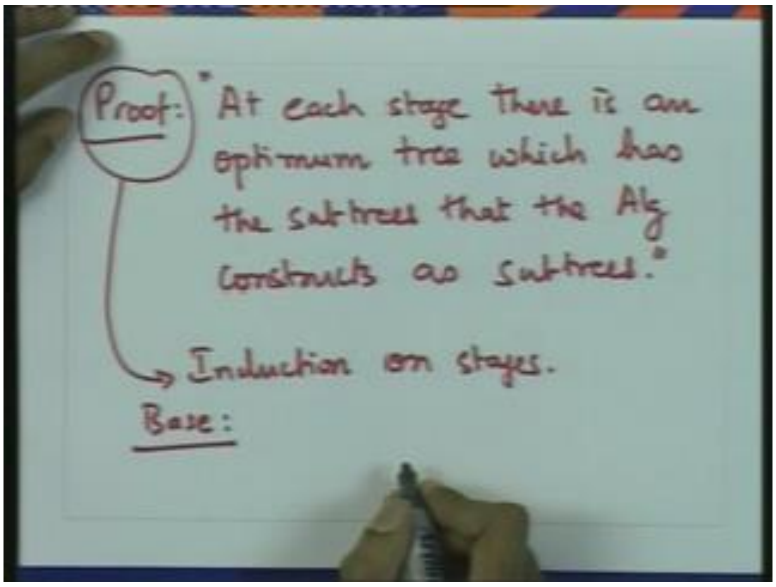
So, the intermediate stage is this I have sub trees, let us say t 1 t 2 so on up to T k. Now, I associate weight to each of them, which is just the some of the frequencies of the elements in side each. So, let us say f 1 prime f 2 prime so on f k prime. So, f i prime is the sum of frequencies of elements in T i or symbols or words. So, just sum of all the frequencies and that is my f i prime.

Now, I pick two of minimum f prime and join them together and make a new sub tree. For instance, if f k prime and f k minus 1 prime where the smallest once. Then, I would take T k minus 1 and T k and join them together. The others remain as they are T k minus 2 and so on. This is if T k and f k prime was the smallest, let us say and f k minus 1 prime was the next smallest.

When, in that iteration I take this as input and I create this. So, number of tree is as decreased by one I have just merge these two sub trees together. The new frequency of this will be just the sum of the frequencies of these two things. This will remain f 1 prime and so on. Because, here you have just the elements of these two just get union.

So, this is the generic step and I put this into a loop. And this is my algorithm, you can see that the algorithm terminates ease to see. Because, initially start with each leaf, each element as a sub tree. So, these will be the leafs and there are n of them, each time the number of sub trees decreases by one. So, finally, I will just have one sub tree, this will be my binary tree, this will be a the binary tree that I want. And this will give me the prefix free code that I am looking for. So, why does this algorithm work. So, let us write a proof of correctness and again it will just invoke the same exchange principle, that done to write a proof. So, here is a proof.

(Refer Slide Time: 26:16)



Proof: "At each stage there is an optimum tree which has the subtrees that the Alg constructs as subtrees."

→ Induction on stages.

Base:

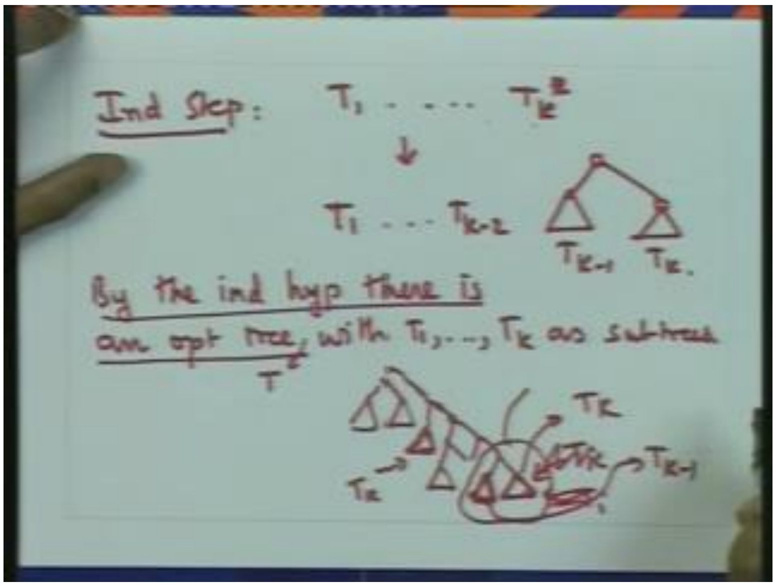
So, what we would like to argue is that, at each stage see at each stage you have ((Refer Time: 26:26)) these sub trees T 1, T 2 up to T k will the proof statement, that we would like is that. At each stage, there is an optimum tree which has the sub trees, that the algorithm constructs as sub trees. So, what does it mean. So, if the algorithm for instance here if the algorithm had sub trees ((Refer Time: 27:19)) T 1 T 2 up to T k, there must be an optimum tree which has these as sub trees.

You do not know how these are connected, but the optimum must have these as sub trees, there must be some optimum tree with these a sub trees. Now, this statement is if you prove this statement, then we are done. Because, once algorithm terminates we just have one tree. And we just said that an optimum must have this as a sub tree, which means optimum must be this tree.

So, if this statement is true for all stages of the algorithm, it should be true when the algorithm terminates. In which case we have what we want, so why is this true. So, again the proof you can write this proof by induction on the stages. So, we would like to prove this maybe I should shift this here. So, proof by induction on stages, the first stage at the beginning.

So, the base case well it is true. Because, what you have as sub trees I just the leaves and these should be leaves even in an optimum tree. The base case you have n elements which are not connected to anything. And this is true even in the optimum case. So, the base case is... So, what is what about the inductive step, well what happened in the inductive step. So, here inductive step.

(Refer Slide Time: 29:13)


Ind Step: $T_1 \ldots T_k$

$\downarrow$

$T_1 \ldots T_{k-2}$ T_{k-1} T_k.

By the ind hyp there is an opt tree, with $T_1, \ldots, T_k$ as subtrees T^*

Well, you started out with trees let us say T 1 up to T k and you ended up with T 1 up to T k minus 2. Two of them let us say the last two you join, this is T k minus 1 and T k this is what the algorithm ((Refer Time: 29:37)). You know by induction, that there is an optimum which at these as sub trees. There is an optimum tree, which at these as sub tree. So, let us look at this optimum. So, by the inductive hypothesis, there is an optimum tree with T 1 up to T k as sub trees.

Now, we would like to argue that there is an optimum, which as all these as sub trees, voice are true well the reason is this. Let us look at this optimum tree, there is an optimum tree let us say call it T let us look at T. Now, among these… So, how this look like. So, it is something like this, then there is a sub tree maybe goes as a sub tree somewhere around this is a sub tree, then the branches of line this maybe there is a sub tree here and so on.

So, there are these sub trees T 1 T 2 up to T k sitting here. So, look at the lowest sub trees. I clean that I can put T k here, the lowest sub tree must contain T k. Now, why is that is so supposing it is not supposing something else it is here. What I do is some T i? What I do is exchange, supposing let us say T k for here and you know T i for here.

Then, I exchange T i and T k I do this sub tree exchange. And by the previous argument that we did, we saw that the cost decreased. When we did the exchange trick and we did the calculation, we saw that the cost actually decreased. So, this should not be possible, in which means in the optimum I should always have T k sitting at the bottom.

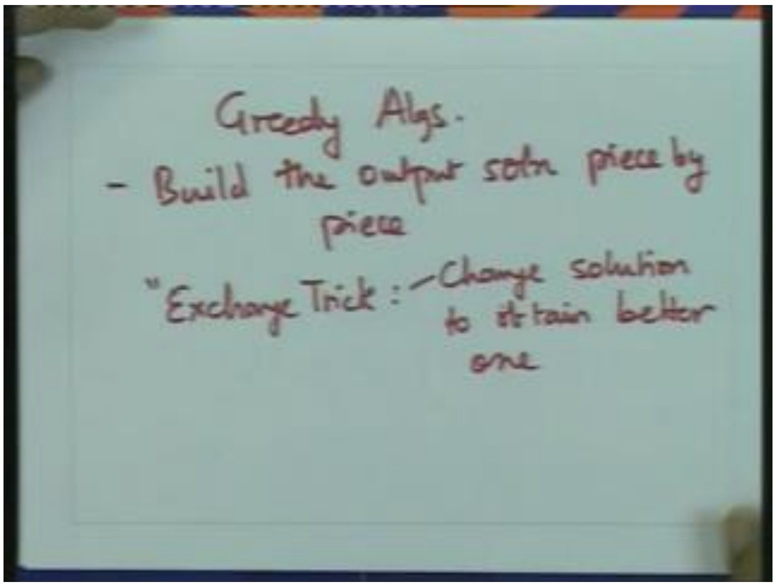
Similarly, T k minus 1 must also these sitting at the bottom, which means the height must be again maximum. So, what I can do is take the bottom to nodes, take the bottom to nodes. And I can always exchange whatever sits here, I can exchange it with this should be T k and this as T k minus 1. Remember, that these frequencies are the smallest T k was the smallest and this was the second smallest.

So, I can always exchange any of those trees with these two, these are the bottom most. And I will always get cost which is less. So, I can assume that there is an optimum solution with T k and T k minus 1 and the bottom. Well, now I am done because this structure I just pull out from here. This is T k that T k minus 1 and here is the other nodes. So, this structure I just pull out from here good.

So, there is an optimum tree which as T 1 T k minus 1 and this as sub trees. And we are we are actually done. So, finish the proof that this algorithm is optimum. So, each time you look at you merge to trees, which I have the least weight, the minimum weights of tree and the second minimum. And you create the new tree by merging these two into a binary tree.

So, that is the algorithm and networks, this was done by Huffman and it is called this coding is called Huffman coding. So, let us summarize our discussion of greedy algorithms.

(Refer Slide Time: 33:42)



Greedy Algs.

- Build the output soln piece by piece

"Exchange Trick : - Change solution to obtain better one

Well, the main thing was we build the solution output, solution piece by piece each time we somehow get the right piece two imagine it is a kickshaw. And you know pulling out pieces to fit. And each time put the right piece and place, that is the crucial thing to help us we used what is called exchange trick.

Essentially start with any solution and change solution to obtain better one. This is just something that you do to figure out, what your algorithm should be doing. This is not what exactly the algorithm does, this is just you help you figure out what the algorithm does, which is you start with any solution. Now, we start of ((Refer Time: 34:50)) the solution a bit and see what happens.

And typically it is exchanging part of the solution for something else. You takes something out of the solution puts something in, you do a better exchange. Then, you see what happens to the profit or the objective function. If it is increases, if it is gets better than some tells you gives your ((Refer Time: 35:14)) how to proceed. So, this is about greedy algorithms I would recommend, that you study metroids, theory of methorids and linear programming, especially the so called primary dual method, which is available in most text books and linear programming. And these two may give you a better sort of feeling for greedy algorithms, in how you know the strategies work.

Good luck.