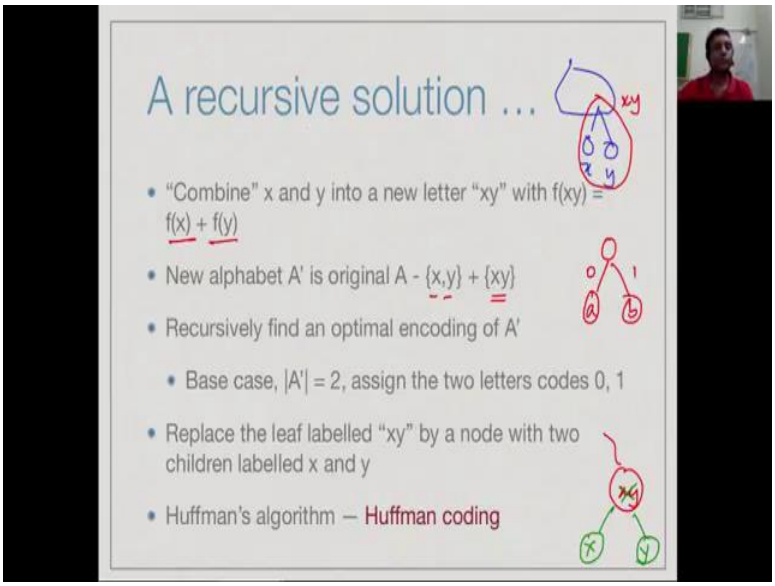
(Refer Slide Time: 17:10)

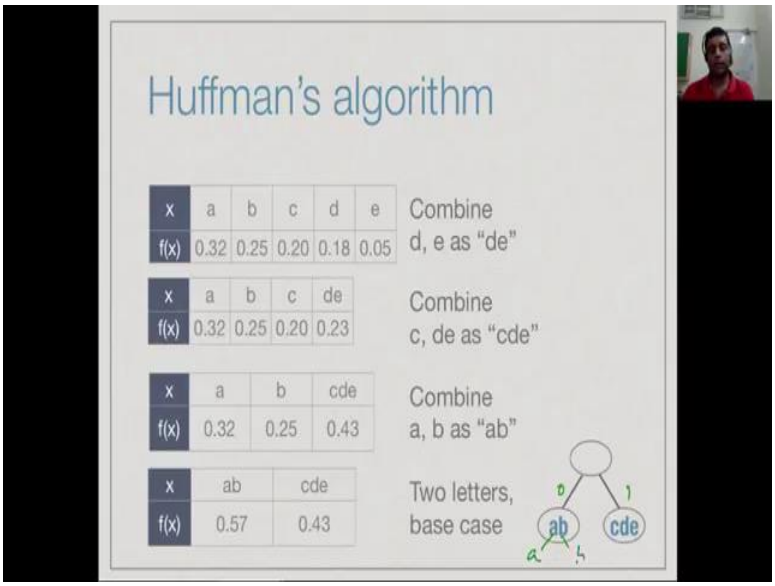


So, now, the recursive a solution will say, that how do a figure of what the rest of the tree looks like, well if I have a situation, where I have decided x and y both here. Then, I will kind of tree, this is a unit and make a new letter call x, y and give it the cumulative frequency effects plus x, y of the old two letter. So, construct the new alphabet and which I drop x and y and I add a new composite of hybrid letter x, y; whose frequencies going to be f x plus f y.

Now, recursion fiction, I have a k minus 1 letter alphabet, so I have recursively find and optimal encoding of that. Now, before coming to how to adapt the solution, the recursive ends when have a only two letters, for two the optimal solution is to build the tree which exactly two leaves, label 0 and 1 at the path. So, this is the basic case, if I have more than two letters I will recursively construct tree to the smaller thing and then I will come back and now, the tree that I constructed I will have some where the leaf label x y.

Now, x y is not a letter, so what I do is, I will replace this, write new two leaves called x and y. So, I will go from the tree over a A prime to a tree over A by doings. So, this is an algorithm called by develop Huffman and this type of coding is call Huffman coding.

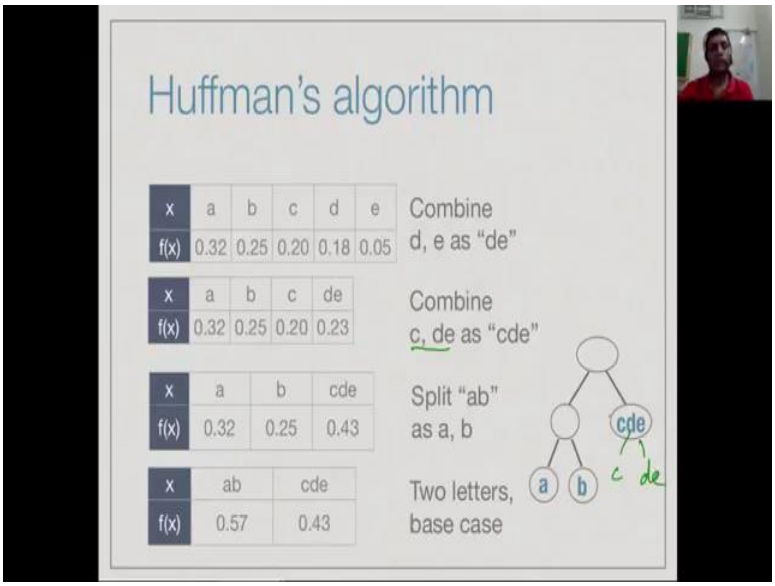
(Refer Slide Time: 18:36)



So, let us look at this example that we had earlier, so here the two lowest frequency letters d and e. So, we merge them into the new letter d, e and this is a frequency 0.23, because it is 0.18 plus 0.05. Now, these two are a two lowest letters, so we merge them and we get a new letter c, d, e of cumulative frequency 0.43, which is some of all the frequencies are that two values.

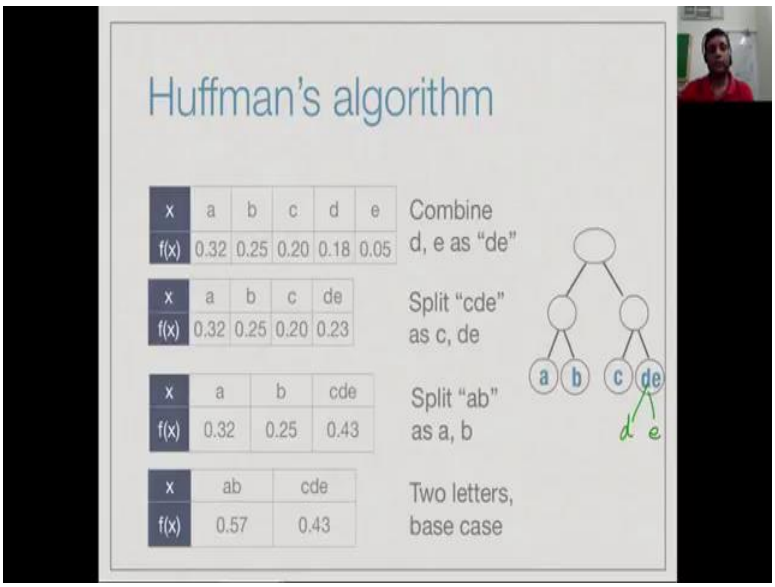
Now, terms out that, these two are the smaller two. So, I is them the letter a, b and now, I breast my base case where have exactly two letters. So, I can set of the trivial tree this two letters, label 0 and 1. And now I work backwards, so the last thing that I did was to merge a and b, now I will take this a and b thing and split it has a and b.

(Refer Slide Time: 19:30)



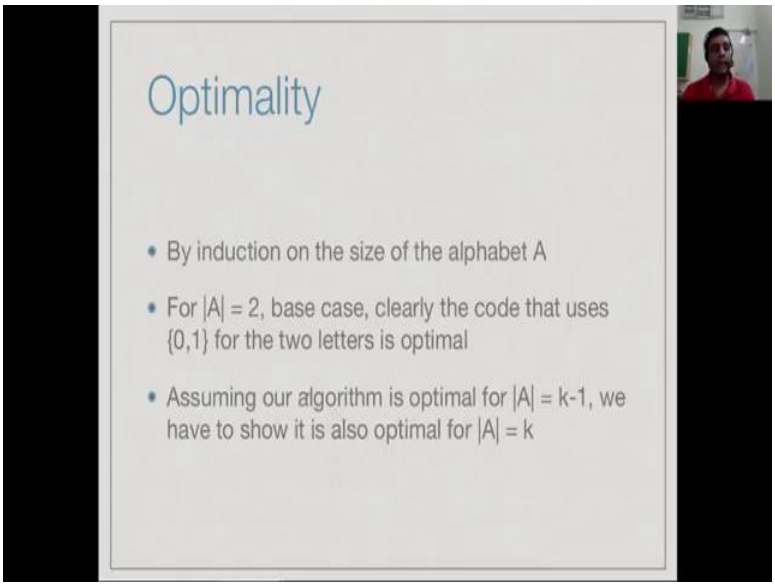
I will split a, b as a and b, I will get this print, then the previous step was to combine c, d e into c and d, e. So, I am going to the split this c and d, e.

(Refer Slide Time: 19:40)



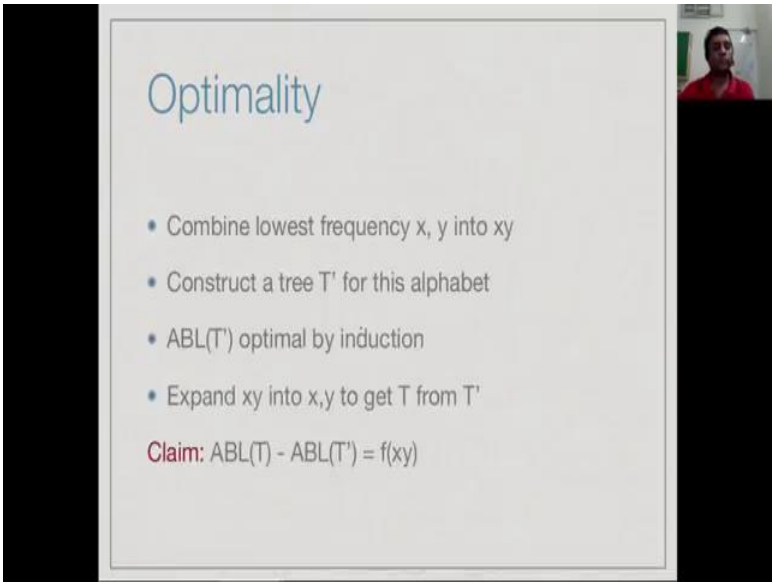
And finally, I am going to split this up is d and e. So, this is Huffman's algorithm and by recursively combining the two lowest frequency nodes, and then taking the composite node and splitting them back up to it is.

(Refer Slide Time: 19:56)



So, to show that this algorithm is the optimal, we go by the end of the size in the algorithm. Now, clearly when I have only two letters, I cannot do any better than assign the one of them 0 and one of them 1, so the base case is optimal. So, we will assume that this optimal for k minus 1 letters and now show that also optimal for k letters.

(Refer Slide Time: 20:17)

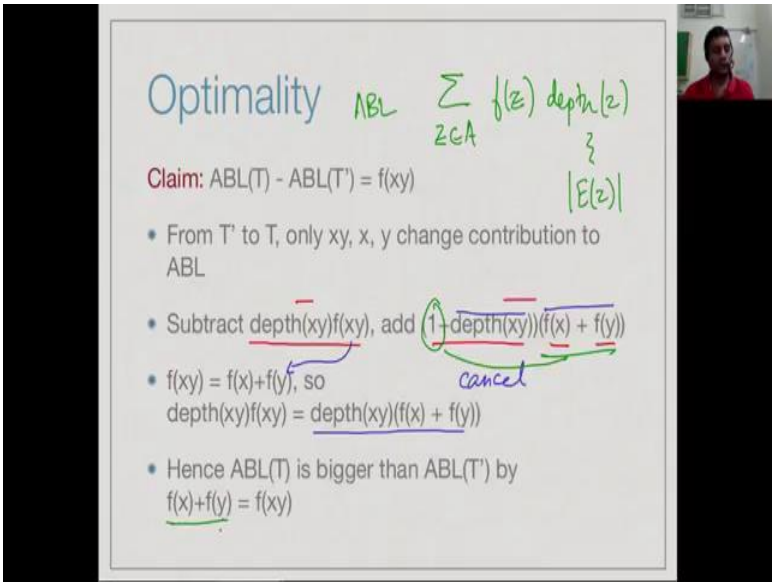


So, recall that, we went to k minus 1 by combining the two lowest frequency letters as 1, constructing an optimal tree for these smaller alphabet and then expending the x, y get a new. So, the claim was when I go from the tree over k minus letters to the tree over k

letters, the cost is going to increase, but this cost is going to be fix by whichever letters I choose to contract.

So, if I choose x and y to merge to go from T to T prime, then the amount by the which the average bits per letter is going to change is exactly the frequency of this combine letter f x y. So, the deterministically fix by which one I choose, then even though I do know the cost of the trees directly, I can tell you that the cost is going to be different by this some after.

(Refer Slide Time: 21:08)

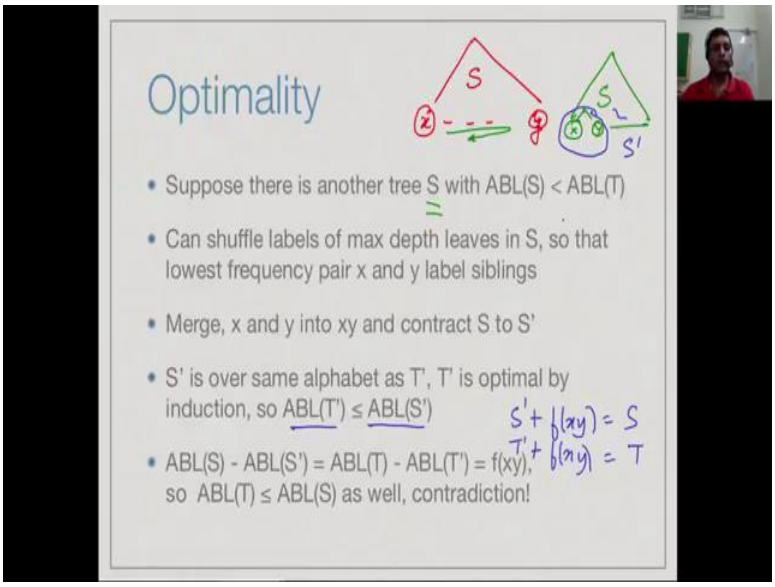


This is not very difficult to prove, so in that summation that we had, so in the tree notation remember this depth of z is the same as the length of the encoding of set, because the depth exactly a reflex the length of the string use to encoded. So, this is our A B L calculation. Now, for every node other than the x y and x y, there are exactly at the same position in T and T primes.

So, this these sum summation with terms do not change, so the only changes are these three nodes. So, what I will do and going to T prime to T is I will remove this contribution of the composite letter x, y. And then at a next level which is 1 plus the depth of the x y, I will add the node f x and I add the node f x and I will get the components x y, f x times that plus x y times, I am subtracting this amount in the left, then I am adding this amount to the right.

So, now, actually f of x y is nothing but f x plus f y. So, this left hand side term is actually this right hand side component depth of x y, times of x plus y. So, I am subtracting this and adding it back, so the cancel each other, so therefore, all am left with is one times that the x plus y which is f x by f y or f x y. So, therefore I am going from T prime to T, the crucial thing is only depend for which letters I have contract, that fix is the difference unique.

(Refer Slide Time: 21:51)



Now, let us assume that, we know how to solve all k minus 1's say alphabets efficiently and we have done is recursive thing to construct the tree for size k. Suppose, this is another strategy would produce the better tree for size k, so this another tree candidate tree S produce by some different strategy, who is average bits for letter is strictly better than the one that we construct recursive.

Now, in S, we know for sure that these two letters that we use the in recursive construction x and y. So, they occur somewhere the bottom of this tree. So, this is my tree S, these must be leave nodes, because they have the lowest frequency is over all the letters, so must be a maximum depth, that they may not be next to each other. But, it does not matter, because since they are both and maximum depth, I can move them around, this step, I can move letters around, I can reassign the two other leaf to a sender.

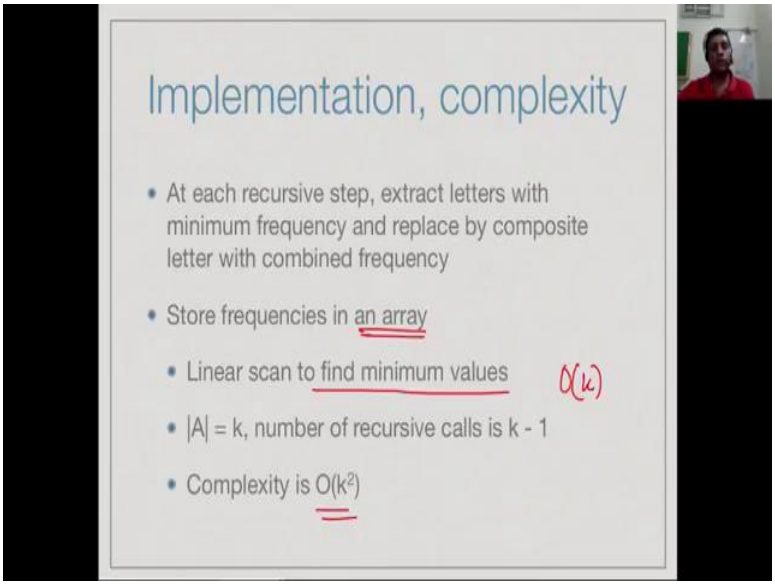
Such that, I come with the configuration I come to new S, I call it S again, where actually have x and y together. I can assume that the S, that has this optimal property, which is

better in the tree as constructed, actually as x and y a sibling leaves other maximum depth. Now, what I am going to do is this S, I am going to concretely fuse this and get an S prime.

So, this explain will be our k minus 1 letters except instead of doing this by a recursive call, I am actually taking a call concrete tree over k letter and I am actually compressing to two nodes into 1 and call in it to tree over k minus 1 data set. But, because this over k minus 1 letters and these are represent the encoding, it cannot be any better than the encoding that I recursively computed for k minus 1 letters, because I am assumed by induction by that algorithm those efficiently for k minus 1 letters.

So, so S prime is no better than T time, but as prime plus f of x y is S, T prime plus f of x y is T. So, the different T prime and T and S prime S and exactly the same. So, the S prime is a then T prime, then S cannot any better than T. So, it was a contradiction to assume that as I strictly better than T is this 2's that strategy of recursively computing T is optimal for all k.

(Refer Slide Time: 25:21)

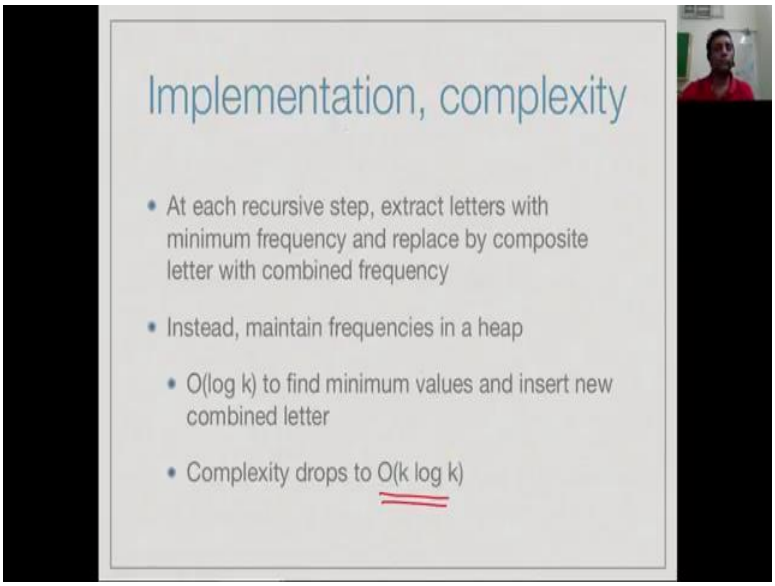


The word about the implementation, so what we have to do is k minus 1 time, we have to merge this two minimum values and compute the recursive solution, bottle neck, what will make is finding the minimum values. If you use an array, then as we know scan the array instant to find the minimum values, remember that the minimum of values keep changing, I cannot short it one send for all. Because, each time I combine two letters, I

can use a new letter into my array with a new minimum value which was not there before and not a new value, which may not there before it is may or may not be the minimum.

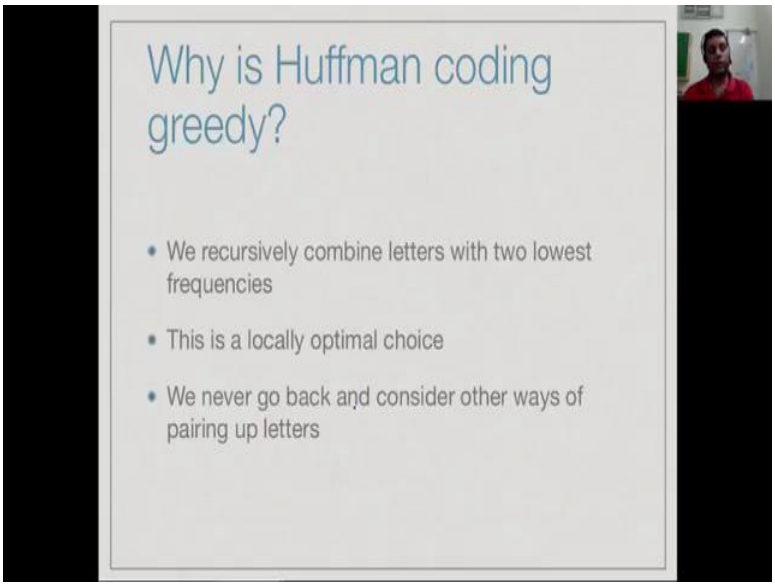
So, each state I have to find the minimum, so it is an order case can each time, so linear scan and I do this appropriate k these times. So, I get order case two, but it should be fairly cleared into see that this bottle neck can be got around by using a heap, where there is precise what heaps are good at finding the minimum.

(Refer Slide Time: 26:13)



So, if I maintain the frequencies is not at as a heap, then the order log k time, I can find the minimum values and then I can insert back the new composite letter also into heap in to log k time. So, each iteration takes some log k, and so I am improving from k square to k log k.

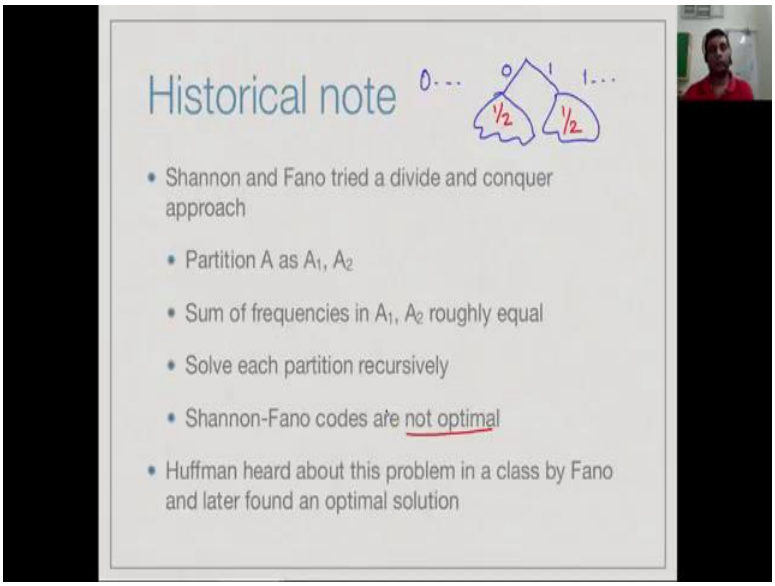
(Refer Slide Time: 26:33)



So, recall that, we mention that the beginning that this is going to be a last greedy algorithm. So, y is greedy, well because every time, we make a recursive call, we are deciding to combine two nodes into one and the two letters we are choose always once with the lowest frequencies. Now, what is to say, that we could not to better by choosing the lowest then the third rows per, but we now a try, we only try to lowest to the second rows.

So, we make a locally optimal choice and we keep going with choice, never going back to the visited, and finally we get a global solution. Now, we are prove that this global solution is actually optimal and we have to do that, because there is no other reason is expect that I making a short sited choice, at the current time take the two worst frequencies and combine them, that you are always going to get the best solution. So, this is very much the greedy is letter.

(Refer Slide Time: 27:25)



So, finally a brief historical note, so Clot Shannon is the father of information theory and when, we are looking at these problems around 1950 are, so they where face to this problem are finding and efficient. So, Shannon and Fano, proposed the divide and conquer thing, so what us it was let us look at the encoding of the alphabet. So, some of them are going to start with 0, some other going to start with 1.

Everything which is in the left sub tree of this coding tree that we construct is going to have a code of the found 0 are something, something, everything on the right is going to have something at the found once. So, it seem intuitive to them, then divide and conquer strategy is good, what you can put letters on this side, such that the occupied roughly of the frequency weight of the total alphabet. That is a frequencies of all the letters, who's encoding start with 0, their frequencies added to roughly hard and the other one also to hard.

So, split the alphabet in the two of equal weight assign some of them to start with 0's, the other to start with 1, then I recursive it is solve this t. So, a partition is A 1, A 2, sums of the frequencies in each other sets are roughly equal, solve them recursively. Unfortunately, this is not guaranty to generate an optimal encoding, you can come up to the example, where you can do this and then end of the something which you can improve by doing some other.

So, it turned out the Huffman was a graduates student in a course of Fano, he heard about this problem and we thought about it, and after a few years he came up with this clever algorithm which we are done in it.