Programming and Data Structure

Dr.P.P.Chakraborty
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture # 28

Algorithm Design - III

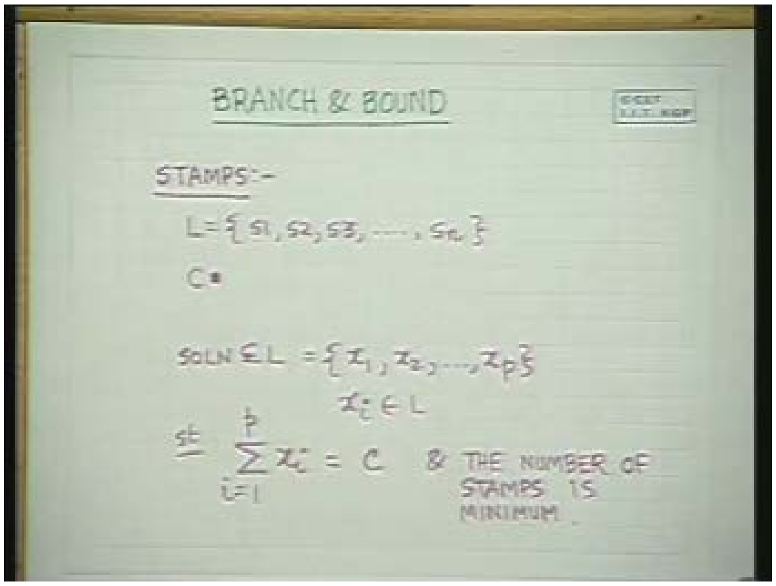
Hello everybody. Today the topic of discussion will be the next design technique in algorithm design. Till now we have seen that the concept of algorithm design is that of getting an, generating an initial solution which is a recursive definition to a problem and then refining it. Among the refinement techniques that we have seen are two very important techniques, first of balancing the split and secondly that of dynamic programming where identical problems are solved only once. Today we will discuss another very important technique is that namely that of pruning. Now, pruning of a sub problem. Now what do we mean by pruning of a sub problem? That is while solving a sub problem; we will try and ensure whether it is at all necessary to solve that sub problem.

Now pruning can come in various ways and one of the most famous techniques of pruning is the technique of branch and bound. It does not take care of all the types of pruning, there is another very important technique of pruning called the greedy approach which essentially is related to the recursive definition itself. Therefore if the recursive definition is correct, you can get a good greedy approach. You can have a look at greedy approaches in standard books on algorithm design but as a process of refinement, branch and bound is one of the most well used and well-studied techniques. It is specially used for solving problems which take enormous amounts of time and are applicable for problems which are optimization problems.

Optimization problems are usually those which require minimization or maximization of a particular value. Sorting is not an optimization problem. Sorting is a problem where you are trying to obtain a sorted version of the solution. Therefore an optimization problem is one in which you are interested in finding out the maximum or minimum value of a certain problem. Now let us see what we mean by such a situation. So, the topic of today's lecture is branch and bound but in order to understand branch and bound, let us pick up a problem and let us see how branch and bound technique comes out in solution to that problem. The problem you will recall we have done earlier that is the stamps problem. In the stamps problem you are given a set of stamps, a list or a set s_1 s_2 s_3 s_n say and you are given a quantity value C.

Now you are to pick out a set of stamps in the solution. You are to pick out a solution which is a subset of L is equal to some x_1, x_2, x_p where each x_i is an element of L and it's a distinct element. Let us for the time being say we are choosing one stamp can be taken only once such that summation x_i i equal to 1 to p is equal to C. And so this is the solution, you are picking up a set of stamps whose value is equal to C. And what do we want? The number of stamps is minimized. That is we want to pick out a set of stamps whose sum is equal to C such that the number of stamps is minimized. So here we have an optimization problem at our disposal.

(Refer Slide Time: 05:19)



BRANCH & BOUND

STAMPS:-

$L = \{s1, s2, s3, \ldots, s_n\}$

C

SOLN $\in L = \{x_1, x_2, \ldots, x_p\}$

$x_i \in L$

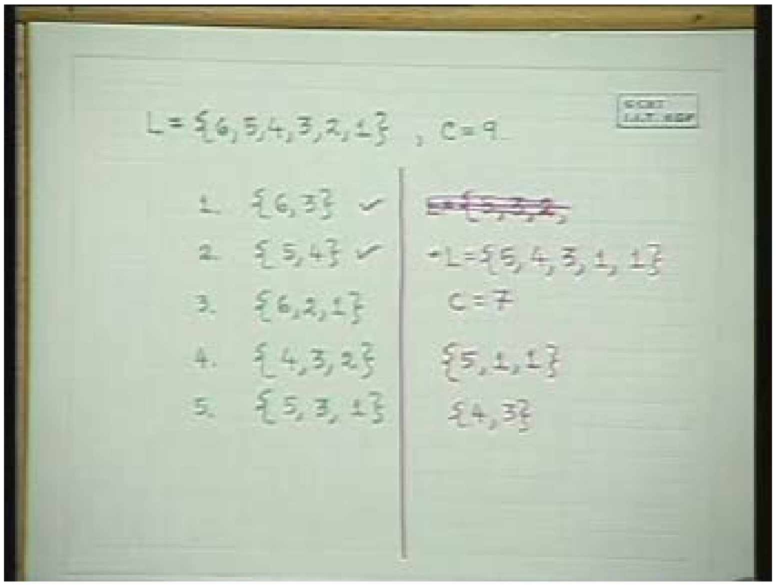
st $\sum_{i=1}^{p} x_i = C$ & THE NUMBER OF STAMPS IS MINIMUM.

The problem stated is as follows. You are given a set of stamps, you have gone to the post office and they said that look here are the only stamps that we have and you have got a certain amount of space and you would like to fill up stamps of exactly a particular value and you ought to choose the minimum number of stamps which meet this value. One version of the problem is choose a set of stamps which meet this value but that is not an optimization problem. The minimum number of stamps which meets this value is an optimization value. Let us take an example that will help to clear the error. Let L be 6 5 4 3 2 1, these are the set of stamps and I can take each stamp, this is a value of each stamp and I can take a stamp only once and the C is equal to 9. Now I have various solutions. The first solution that I can choose is 6 3. Second solution I can choose is 5 4, third solution I can choose is 6 2 1. So these are the solutions that I can choose. There is another fourth solution I can choose 4 3 2. Fifth solution I can choose is 5 3 1. So these are the solutions which I can choose.

Among these five solutions, it is easy to see that this and this both are the minimum quantity because there you are choosing only two stamps. So, how to find out a solution to this problem? It may seem natural that we sort the stamps and choose the maximum and choose the next one. We should be getting the optimal solution but that indeed is not the case because if you are asked to choose a particular value and it does not necessarily mean that if you choose the largest one, for example here you say I will choose 6 and then I am left with 3, so 5 and 4 is not possible then I will choose 3 and here I get very optimal solution sorry that is the case only in the scenario where you are given in a particular scenario. Maybe you will be given a different formulation of the problem where you will get different solution altogether. For example if L was given as 5 3 2 let’s take 5 4 3 2. L is 5 4 3 1 1 and let us say C is 7. Then normally we would say choose 5, now you are left with 2, you cannot take this, you cannot take this. So you will take one. Then again you are left with one. So, you will take one and you would end up with a solution like this whereas the solution 4 3 is optimal.

Therefore a simplistic approach like this where you choose the largest and continue taking the largest element to meet your quantity may not give you a solution. Therefore you must be careful about solving the problem optimally. Therefore a simple first, largest first approach will not give the solution. So how do we go about solving a problem like this?

(Refer Slide Time: 09:37)



$L = \{6,5,4,3,2,1\}$, $C = 9$

1. $\{6,3\}$ ✓
2. $\{5,4\}$ ✓
3. $\{6,2,1\}$
4. $\{4,3,2\}$
5. $\{5,3,1\}$

[illegible]

$L = \{5,4,3,1,1\}$

$C = 7$

$\{5,1,1\}$

$\{4,3\}$

Any problem as we discussed before, in order to solve that problem we need to provide an initial definition to the problem and an initial definition is a recursive one. And we have seen such problems and we have seen the permutation and the combinations problem and wherever where we looked in at the stamps problem and one recursive definition of the stamps problem is to work it out as we do just as follows. To consider all possibilities let us say we have got L C and N is the number of stamps taken till now. This is the number of stamps chosen till now. Therefore initially it will be called with L and C and 0.

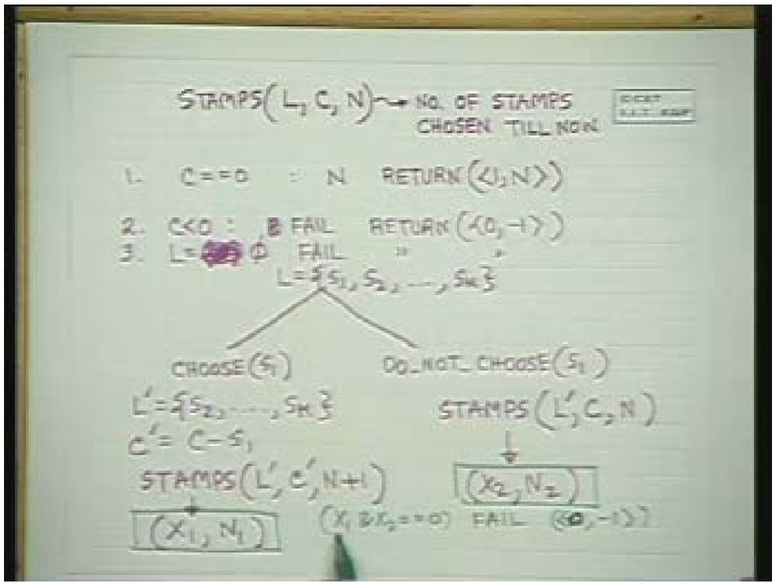
Now what are the cases? If C is equal to 0, obviously I have found a solution and the solution is N because if C is 0, the solution is N, so I return saying I will return with success, so I return with success means 1 and if it is successful, the solution is N. So I return a structure which has got two parts, two integers, one which indicates 1 or 0, 0 means failure and 1 means N. If this case is not true else so each case is else, if C is less than 0 then obviously we fail. That is you have got a negative value to meet and it's not possible, so you return 0 minus 1.

Similarly if these two are not true and N is null that is C is not zero, C is not negative but L is null that is L is null. There is nothing to choose from; therefore C must if it is not zero and it is not negative it must be a positive quantity. If C is positive and L is null, you have no stamps to choose and you will still get a value, again you will fail. These are actually all the base conditions of the recursion.

Now if these are not zero then obviously L is non null and C is greater than 0 and as we discussed before we have got two choices, choice one if L is s_1 s_2 and say s_k you have got two choices, one is choose s_1. That is s_1 will be in your solution and do not choose s_1 that is either I can take s_1 or not s_1. If I take s_1 and I am sure to take s_1 then I will make L dash is equal to s_2 write up to s_k. C dash is equal to C minus s_1 and I will recursively call it, if this was stamps (L, C, N) I will call stamps L dash, C dash and since I have taken one stamp N will be one more, so I will get a solution for this and a solution for this will be say x_1 n_1. And if I do not take it that is I do not take the stamp then obviously since I am not going to take it, L dash is there but C does not change because I have not chosen any other stamp, I have not chosen any stamp here.

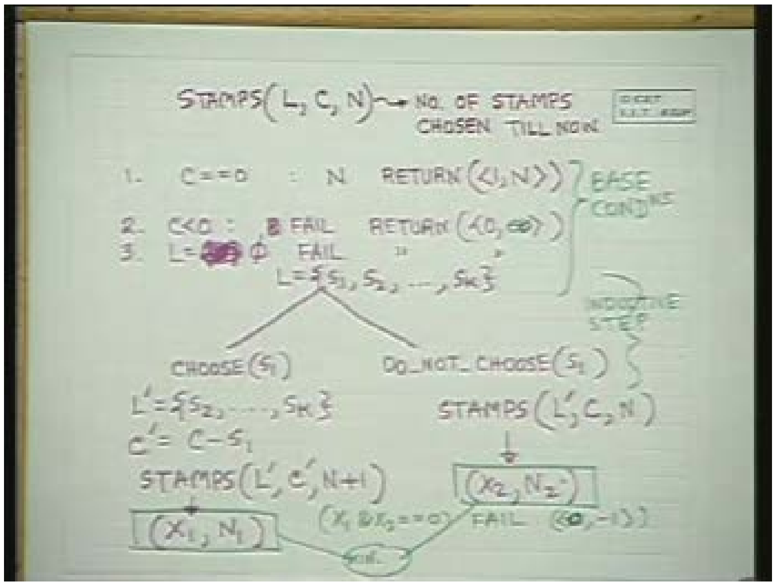
So, I have just decided not to take s_1. So in my new list L dash N remains, how many stamps I have already chosen. And suppose this gives X_2 N_2, so these are the only two options I have at my disposal. I can either take s_1 or not take s_1. I could have done it for any one of these but I can just choose one of them and do it. There are other decomposition but this is the one which we are taking. Now I compare these two, these are the two recursive decomposition, decomposed solutions that I have obtained. Now if both x_1 and x_2 are 0, then I return this. So if x_1 and x_2 are equal to 0 both, then I fail and I return minus 1, so 0 minus 1.

(Refer Slide Time: 14:55)



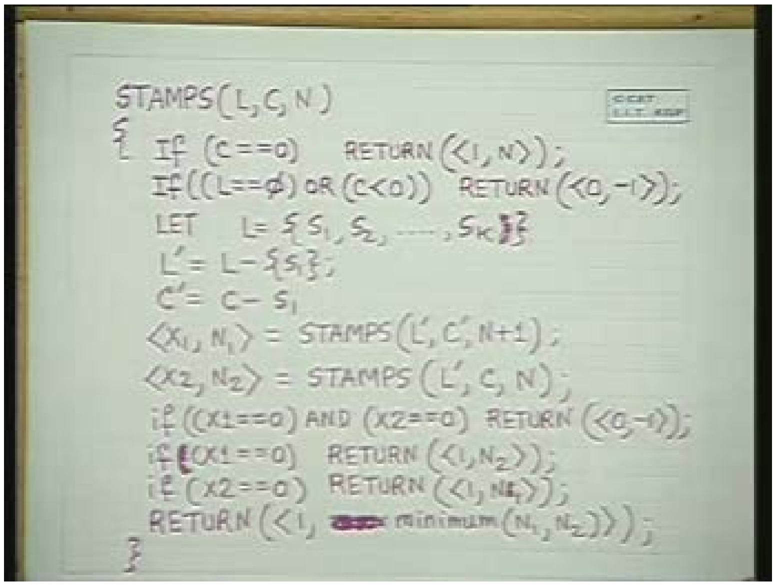
If x_1 is 0, x_2 is not zero I return N_2. If x_2 is 0 and x_1 is not zero, I return N_1, these parts are clear. And if both of them are nonzero, that is this also gives the solution and this also gives the solution then since I am choosing minimum, I will choose the minimum of N_1 and N_2. So to repeat the whole recursive, so this is how we do it. The recomposition step is based on these two, solution of these two. That is either it will be N either it will be failure or it will be N_1 or it will be N_2 or it will be the minimum of N_1 and N_2. Now if under failure if you return infinity then taking a minimum will give you the solution. Well, if it is 0 and you return infinity then obviously taking a minimum of these two will give you the solution because if both are zero both will be infinity, you will return infinity or anyway you can solve the problem. So the steps are, these are the base conditions, this is the inductive step and this is the recomposition.

(Refer Slide Time: 16:19)



So if we write out the algorithm then it will look like this. If C is equal to 0, return 1 N. The failure cases are if L is 0 that is if this does not happen, the else part automatically starts to save space in such a long definition, I have not written out the else part. If L is equal to 0 or C is less than 0 then you fail and if these two are not true, you go into the inductive part of the whole definition. That is these two are the base conditions and from here you decompose.

(Refer Slide Time: 17:01)

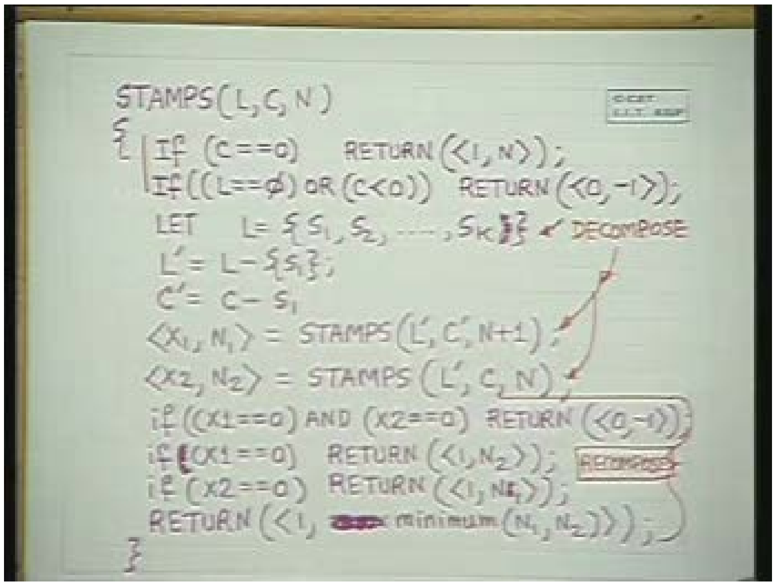


```
STAMPS(L, C, N)
{
    If (C==0)   RETURN(<1, N>);
    If((L==φ) OR (C<0))   RETURN(<0, -1>);
    LET   L = {S1, S2, ...., Sk};
    L' = L - {S1};
    C' = C - S1
    <X1, N1> = STAMPS(L', C', N+1);
    <X2, N2> = STAMPS(L', C, N);
    if((X1==0) AND (X2==0)   RETURN(<0, -1>);
    if(X1==0)   RETURN(<1, N2>);
    if(X2==0)   RETURN(<1, N1>);
    RETURN(<1, minimum(N1, N2)>);
}
```

If L, let L be S_1 to S_k, L dash is L minus S_1, C dash is the value, you subtract the value and here is the two decompositions, you write them in any order, it does not matter. So you will solve them both, stamps on L dash C dash and N plus 1. This is choose S_1 and L dash C and N do not choose S_1 and after this part from here onwards, this part is the recompose. And as we have already discussed, the structure of the initial definition, if x_1 is equal to 0 and x_2 is equal to 0, return 0 minus 1 else that is both are not zero. If x_1 is alone is 0 then return N_2 else that is x_1 neither or both 0, nor is x_1 alone 0. Then x_2 can be 0.

If x_2 is 0 then return N_1. Now if none of them are 0 then you return 1 that you have got a solution, in both of these also you got the solution and the minimum of N_1 and N_2. So this guarantees because at every step, the only two alternatives are choosing or not choosing. So you have taken care of all possible cases and it is easy to give an inductive proof that this algorithm is definitely correct. So this is the initial recursive definition.

(Refer Slide Time: 18:48)



```
STAMPS(L, C, N)
{
    If (C==0)   RETURN(<1, N>);
    If((L==φ) OR (C<0))   RETURN(<0, -1>);
    LET   L = {S1, S2, ----, SK};  ← DECOMPOSE
    L' = L - {S1};
    C' = C - S1
    <X1, N1> = STAMPS(L', C', N+1);
    <X2, N2> = STAMPS(L', C, N);
    if((X1==0) AND (X2==0)) RETURN(<0, -1>);
    if(X1==0)  RETURN(<1, N2>);     RECOMPOSE
    if(X2==0)  RETURN(<1, N1>);
    RETURN(<1, minimum(N1, N2)>);
}
```

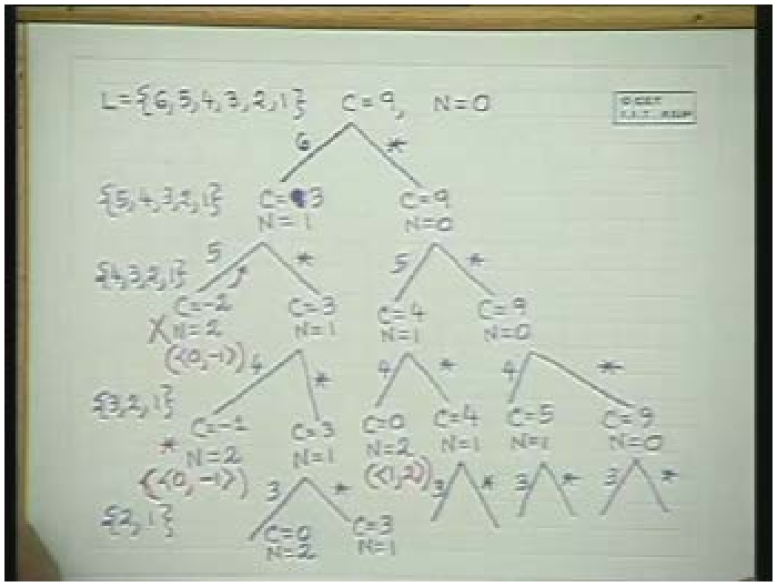
Now let us solve the problem at hand and let us see how the algorithm will proceed to obtain the solution. So let us take the problem that we were looking at that is L and C is equal to 9 and we initialize N to 0. Now C is not zero, L is not null, C is not negative, so obviously the base conditions do not hold and you have got two options choose 6 and do not choose 6. So let this be choose 6, let us work out.

Since the algorithm, let us assume that the algorithm will go in a depth first fashion. So there are two options, 6 is taken and 6 is not taken. If 6 is taken, C becomes 9. If 6, 6 C becomes 3, if 6 is not taken and N becomes 1. If 6 is not taken, C remains 9 and N remains 0. L dash in both cases becomes 5 4 3 2 1. Recall 1 is L dash C dash N plus 1, the other is L dash C N, L dash in both one is C dash N plus 1, the other is C and N. So again you recursively call this one and this one. So this would have given, you can choose 5 and you need not choose 5.

If you choose 5 then C becomes minus 2 and N becomes 2. If you do not choose 5 then C remains 3 and N remains 1. Here let us just work out the recursion tree and then we will see how the algorithm will work and solve. Here again if you choose 5, your C becomes 4 and N becomes 1. If you do not choose 5, C becomes 9 and N becomes, remains 0 and L dash here in this scenario L dash is 4 3 2 1. Now here obviously you stop because C is less than 2 negative, so you return, here you return 0 minus 1 but here neither is C_0 or negative and N is 1, so you have got the option of choosing 4 or not choosing 4. If you choose 4 you get C equal to minus 1 and N is equal to 2, otherwise C remains 3 and N remains 1. Here if you choose 4, C becomes 0 and N becomes 2. If you do not choose 4, C becomes remains 4, N remains 1. Here if you choose 4, C becomes 5 and N becomes 1. Here if you choose 4, do not choose 4, C remains 9, N remains 0.

At this point this is failing, so it returns 0 minus 1. This one continues and L here is 3 2 1. Here you may choose 3, if you choose 3, C becomes 0 and N becomes 2. If you do not choose 3 then C remains 3, N remains 1. Here you return with success because C has become 0, you return with 1 2 and here again you will continue by choosing 3 or not choosing 3. Here you will continue by choosing 3 or not choosing 3. Here you will continue by choosing 3 or not choosing 3 and there are 2 more steps. Here you end with 2 1 and you will have 2 more levels, one with only one and one with null and this is how the recursion tree continues and each of them at the leaf node.

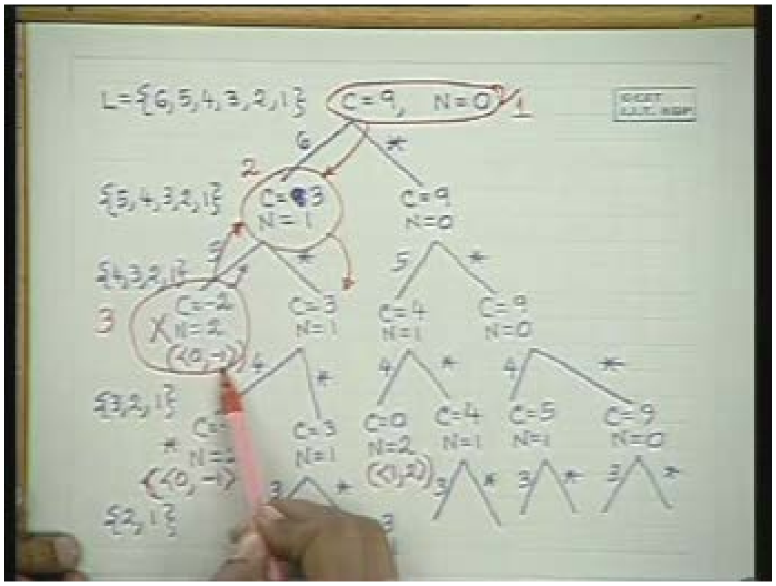
(Refer Slide Time: 24:25)



So this is the structure of the whole recursion tree and based on this structure, let us see how our algorithm will proceed. In order to understand how our algorithm will proceed let us see let us put in this scenario. Here we will choose possibly first. Let us say first we will do this one and then this one. We could have done any one of them earlier that would have meant that here we will do, if it is this side first and choose first and not choose second in the recursion then you would have done it in this way. If you reversed it then we would have done this side first and this side second.

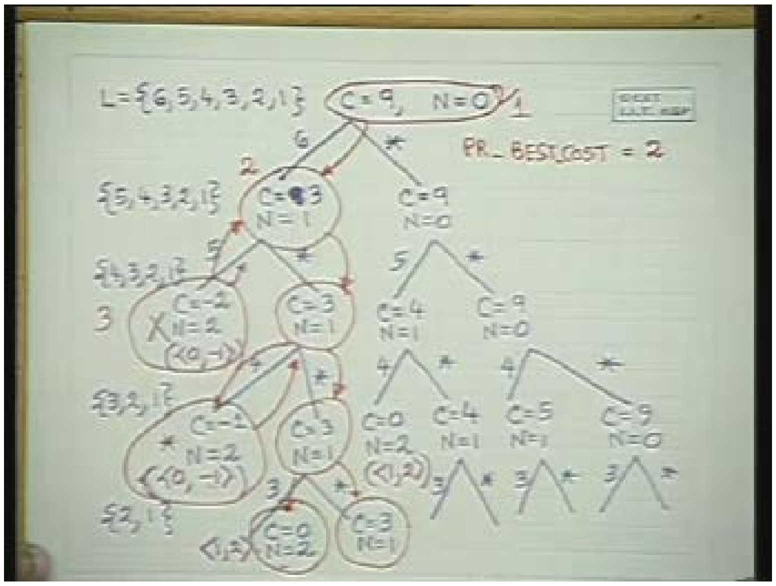
Now let us assume and you will see this practical to do this side first and this side second. Then in that scenario how will the algorithm proceed? The algorithm will come here at this node then it will come here. So first it will do this then it will come here. At this point, it will break up into these two and due to the depth first nature it will come here. Once it comes here, it will back track back here because it ends here then it will come here, from here it came here, when it back tracks it has returned this solution. So the value of X_1 N_1 is this value 0 and minus 1 then it will come here, then it will come here. Here again it will back track because it has failed, so it will come back here. Here then it will come here, after it comes here it will come here. Now at this point it has found a solution, so it will return, it will return 0 and 2.

(Refer Slide Time: 25:52)



Now let us come here. Here we have come at a node. Till now, this is the first solution that I have got and the solution that I have got here that is this will return 1 2. Now let us analyze, at this point till we did not get a solution, there was nothing to be done but as we come here we have got one solution and the cost of that solution that is the present best cost is 2. It may decrease because we don't know but in this solution… Now when we come here, we have come to a point where we have chosen one stamp and this is not a terminal solution.

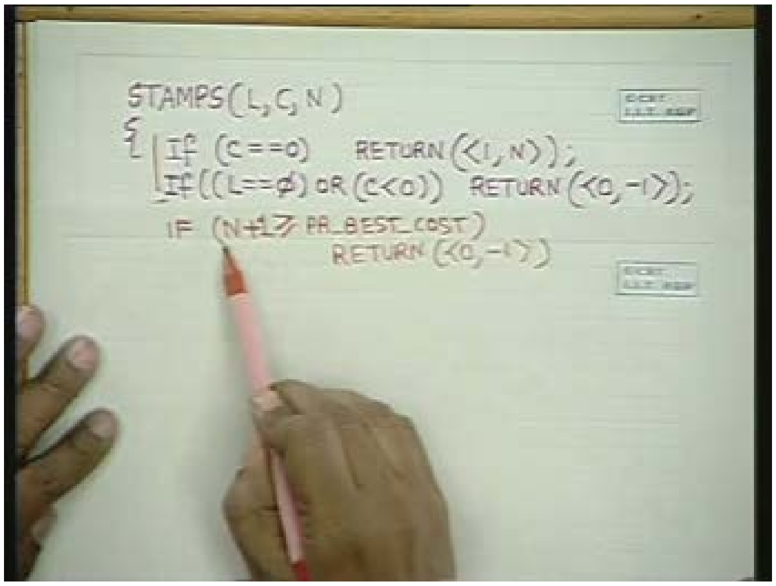
(Refer Slide Time: 27:38)



Therefore we will choose at least one more stamp. Now if we choose one more stamp, still we are not guaranteed whether we will get a solution or not. At this point you can argue in the following manner. If I continue below this, I am not going to get a solution which is better than 2 because I have already got a solution of cost 2 and there is no chance of getting a solution better than 2 here. Therefore what I can do is I can simply say it's not necessary to go beyond this point. Therefore other than these two back tracking conditions or the base conditions that we have, I can do what is called a prune here because what is meant by prune. I will say that if N in the algorithm here, this is one base condition, this is another base condition.

At this point I can say the following. What can I say? I can say at this point that if N let us assume that the present best cost is a global variable. If N is N plus 1 is greater than or equal to the present best cost simply return. I will return with failure, there is no point, I will not get anything.

(Refer Slide Time: 29:31)



```
STAMPS(L, C, N)
{
  IF (C == 0)   RETURN(<1, N>);
  IF ((L == ∅) OR (C < 0))   RETURN(<0, -1>);
  IF (N+1 ≥ PR_BEST_COST)
              RETURN (<0, -1>)
```

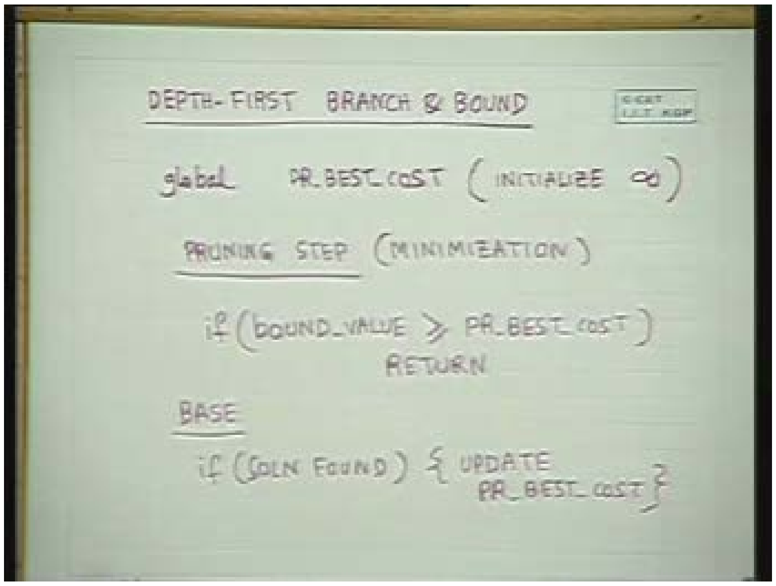
The value of N plus 1 is greater than equal to the present best cost. Look at the situation here. The value of N plus 1 is greater than or is equal to 2 which is the present best cost, there is no point continuing here. So what I actually do is here I just simply return. This is the pruning step and when I do get a solution here I update present best cost. When I get another solution I will update the present best cost because if the new solution is better than the present best cost, obviously available otherwise I will prune. So this is a pruning which will help to reduce the search which would have unnecessarily examined the whole portion below this. So here I prune, I come back here, I come back here, I come back here, I come back here, I come here.

Now what is the value of N plus 1 is 1, so I cannot prune because N plus 1, 1 is not greater than. Let us come here. At this point N plus 1 is 2, I have already got a solution of cost 2. There is no point continuing here and look even if there is a solution below this I still prune and return 0 minus 1. Why because I know that whatever the solution, even if there is a solution below this point that solution is not going to be better than the best solution that I have already got. Therefore I will simply get out and come. This is the pruning step of branch and bound. Now I will come here, once I come here, here again I cannot prune because n plus 1 is 1 but once I come here, I will prune, get back and come here again continue but notice that those portions like this below this whatever I could have seen is a big tree, two more levels was there, I did not see. All these portions are pruned which normally an algorithm would have done.

Below this I do not see and I continue below this and you will, you can drop the whole such tree to see that lot of things will actually not be examined. So this is the idea of pruning. Now this pruning depends on the direction. This sort of pruning in a minimization problem is called the depth first branch and bound approach. This idea since we are doing depth first search, going down like this, this is called depth first branch and bound.

What is meant by bound? Branch is this branching, bound is I have got a value and I know that I can bound my cause by the value. For example here I can bound it by N plus 1, I know that the solution is bounded by N plus 1, it must be more than N plus 1, at any point which is not a gold node which is not, either C is not equal to 0 or C is not negative or L is not null. I am definite at this point that the solution cost is greater than equal to N plus 1. Therefore that is the bound which I used with my present best cost. Therefore what I do in depth first branch and bound for the stamps problem? I keep a global present best cost, initialize to infinity or to the total number of stamps whatever any upper bound and then in the same thing I use the pruning step. What is the pruning step? If bound in, this is in a minimization problem. I will leave it to you to understand what would be in a maximization problem. If bound value is greater than equal to present best cost, return. So, based on this concept you need not go into the decomposition step. This will be just after your base conditions prior to decomposition and in the base condition if solution found with previously you are returning the value, you will update the present best cost.

(Refer Slide Time: 35:18)



DEPTH-FIRST BRANCH & BOUND

```
global PR_BEST_COST (INITIALIZE ∞)
PRUNING STEP (MINIMIZATION)
    if (BOUND_VALUE ≥ PR_BEST_COST)
        RETURN
BASE
    if (SOLN FOUND) { UPDATE
                      PR_BEST_COST }
```

So, these are the two additional things that you do in initial recursive definition and once you do these two additional things, you are sure to reduce your search to a large extent. So this is the essential philosophy or idea of branching. The idea of branch and bound is in the initial definition you try and examine. This is specially applicable for optimization problem. In a minimization problem you will define a bound, a bound which guarantees that the solution will be greater than equal to the cost at this point. In the stamps problem you have seen that if the base conditions are not true N plus 1 is a bound. That is whatever number of stamps you have chosen till now, one more stamp at least you are going to require because you have not yet used the solution. Now if that bound is greater than equal to the present best cost, there is no point decomposing generating some problems and trying to solve it from them.

On the other hand if it is not so, you have to do it but if it is so you will reduce the search and prune to a very very large extent. And you can work out the complete stamps problem; you can start the depth first from any other directions and see. In a maximization problem and you can look at problems of maximization, there since you are maximizing a value you need a bound which is an upper bound. That is it should be saying that the solutions should be no less than this and you will prune if the upper bound is less than the or equal to the current best because the current best will continue increasing. In a minimization problems the current best will continue decreasing; in a maximization problem the current best will continue increasing. This is the concept of depth first branch and bound. There are other branch and bound techniques like the best first branch and bound and other variations.

Therefore like dynamic programming where identical problems are done like the balancing where the split is to be done, branch and bound is one of the foremost techniques of algorithm design. Therefore it is important, we are not going to detail out the properties, the formal properties of branch and bound you would do it in an algorithm design course but here where we are doing programming methodology, our idea is from the initial solution there is a concept of refinement and this concept of refinement has got various aspects. Among the three well designed aspects in algorithm design, one of them is definitely branch and bound. So we have just done a flavor of branch and bound to indicate how to obtain solutions and you will use them in various problems, especially graph algorithms and other problems where optimization that is minimization or maximization is effectively required. Thank you.