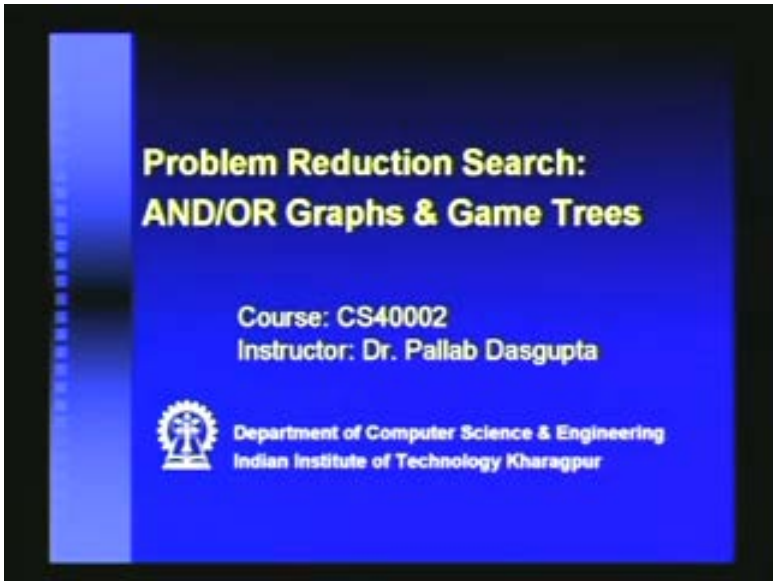
Artificial Intelligence
Prof. P. Dasgupta
Department of Computer Science & Engineering
Indian Institute of Technology, Kharagpur

Problem Reduction Search: And/or Graphs & Game Trees
Lecture - 6

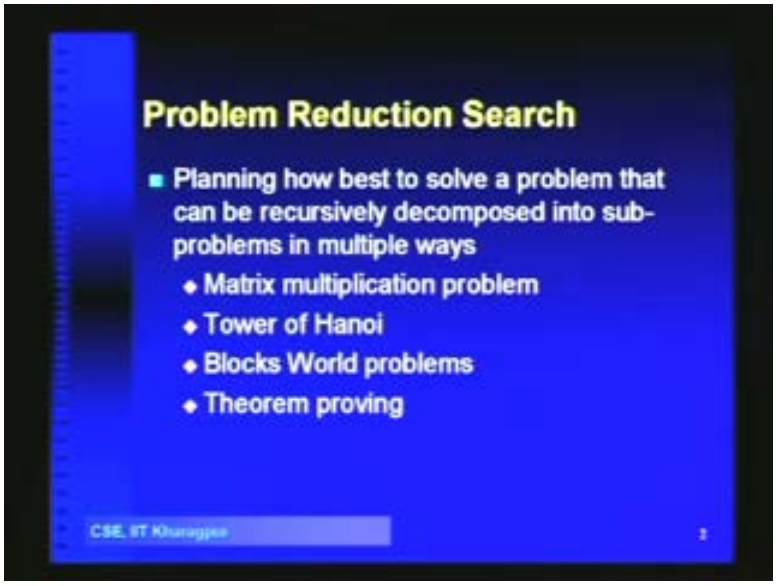
Okay. In the last couple of lectures, we were discussing state space search, and if you remember, in the initial introduction that I gave on search, we said that there are 3 different paradigms for problem solving, using search, broadly.

(Refer Slide Time: 00:01:07)



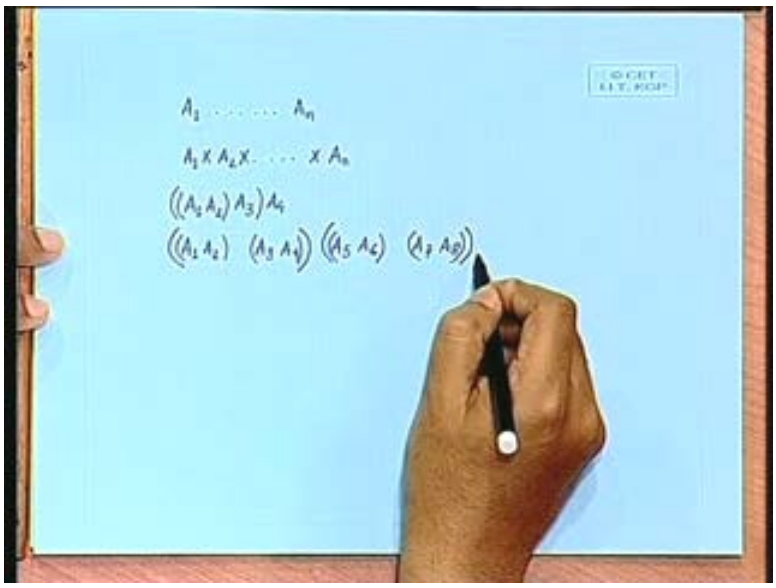
One was state space search, of which there are few topics which are left; we will cover that up later. The second topic is problem reduction search, and under problem reduction search, we will look at 2 kinds of graph search, namely, and/or graphs and game trees. And/or graphs are a kind of structure which we will study, and it has several different applications, though people do not always refer to them as and/or graph search. But the underlying philosophy is the same. And then, we will look at game trees, which is the backbone of game playing programs and for programs which optimize in the presence of an adversary. So, game are situations where you have adversaries, and there is some criterion that you may have to optimize, and in the presence of the adversary, you have to do that optimization.

(Refer Slide Time: 00:12:05)



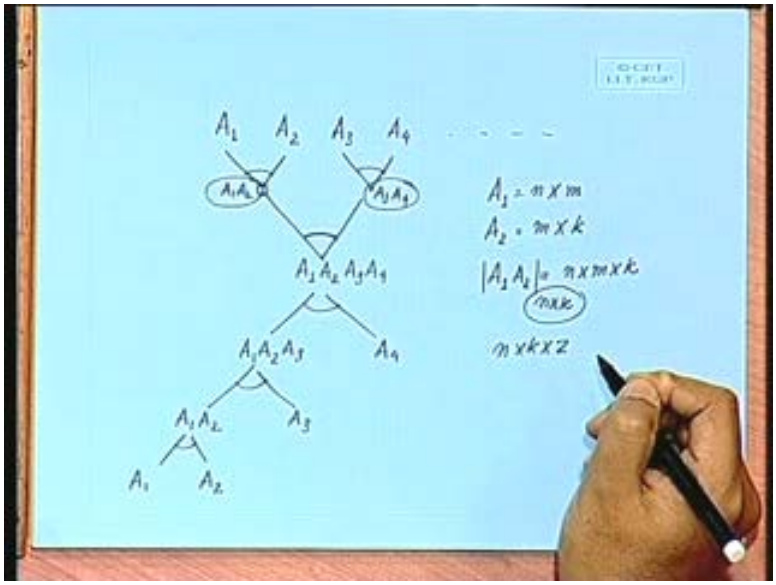
So, problem reduction search can be broadly defined as planning how best to solve a problem that can be recursively decomposed into sub-problems in multiple ways. So, we can solve the same problem by decomposition. There are more than one decompositions of the same problem, and we have to decide which is the best way to decompose the problem, so that the total solution- cost quality of solution or the effort of searching- is minimized. To start with, let us consider the matrix multiplication problem, where you are given a set of matrices A1 till A_n, and we have to find out the product of them. So, we have to find out A1, A2 to A_n, right?

(Refer Slide Time: 00:03:51)



Now, we can do it in several ways. For example, we can first multiply A1, A2; with that product, we can multiply A3, right; with that product, we can multiply A4, this is one way of doing. Another way could be, that we first multiply A1, A2, then we multiply A3, A4, right, then we multiply A5, A6, then A7, A8, in this way. And then, we multiply these 2, multiply the product of A1, A2, with the product of A3, A4, and the product of these 2; and then, finally, in the final step, we multiply the product of this, right? So, it means, that if you look at it bottom up, then here are our matrices A1, and so on. And one way of multiplying is by taking these 2, and these 2, and these 2, right? So, this, if we look at it from the other direction; so, the problem of multiplying it A1, A2, A3, A4, can be thought of, as multiplying A1, A2 and then A3, A4, and then this, right? An alternative way of doing this, would be to have, right. And A1, A2, A3 can be done with the A1, A2, A2. This is another decomposition, right?

(Refer Slide Time: 00:06:41)

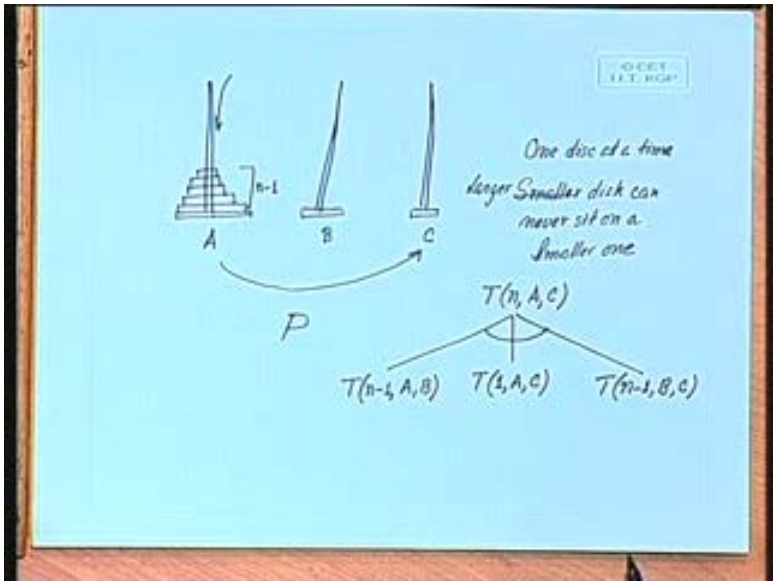


So, this way is one decomposition, this way is another decomposition. These arcs that I draw here, indicate that both of these has to be done, right, so you cannot do one of them. For example, when I have an and here, it means that you have to solve this sub-problem of finding out A1, A2, and this sub-problem of multiplying A3, A4, and then you can solve this. Once you have solved this one, and once we have solved this one, then you can solve this, right? Now, as you can see, that there is cost associated with each of these multiplications. If I have if A1 is n cross m, if A1 is n cross m, and A2 is m cross k, then the complexity of finding the product A1A2 is what? n into m into k, right? And what is the size of this matrix? It becomes n cross k.

So, when I take this n cross k matrix and then I multiply it with another k cross z matrix, then the cost will become n cross k cross z, right? Now, you can easily see that these products which all have, will all have different cause. So, depending on the decomposition, the total number of operations that you have to do for the matrix multiplication will vary. So, one problem is to determine, what is the best way to multiply

a given set of matrices. And there are different kinds of solutions to this problem. For example, you may have studied dynamic programming solutions to this problem. It is not difficult to create a dynamic programming formulation for this problem. We will see how to cost this general problems into an and/or graph search framework, and we will study one algorithm for solving them. Coming to other examples of problem reduction search, there are several planning problems which get decomposed into sub-problems, and there are many ways of decomposition. Are you familiar with the tower of Hanoi problem? Okay.

(Refer Slide Time: 00:11:29)



The problem is like this, that you have 3 pegs, and one of them contains a set of disks. These are disks which have been put in from the top. So, these are round disks, which have been inserted, so they have a hole in between, so you can put them on this disk A. Now, what we have to do is to transfer all of these pegs- all of these disks- from a to c, using b as an intermediate. But, the constraint is that I cannot, at any point of time, put a larger disk on top of a smaller one. So, during the entire transfer, I can move one disk at a time; I am not allowed to move more than one disk at a time.

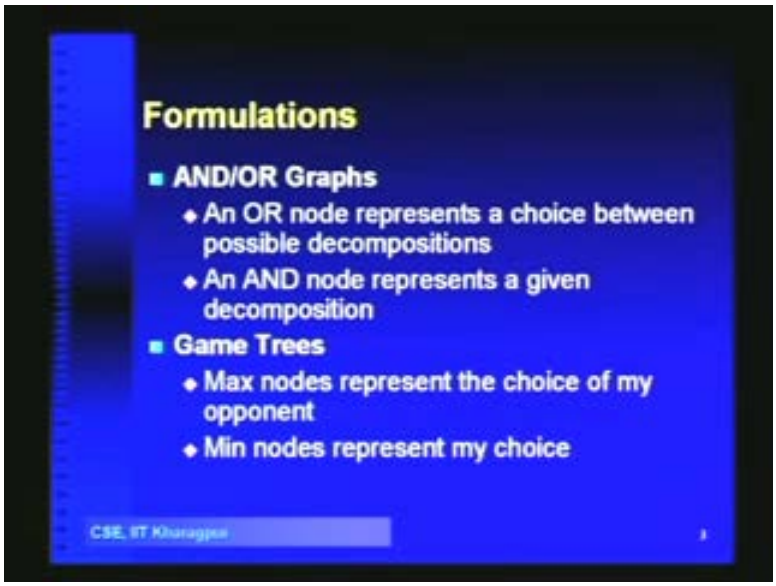
So, the constraints are one disk at a time, right? And the second constraint is that small disk- smaller disk- can never sit on a- rather, it is the other way around: a larger disk cannot sit on a can never sit on a smaller one, right? So, with that restriction- with these 2 restrictions- we have to transfer the set of disks from a to c. There is very neat recursive solution to this, that say, that- use the algorithm; suppose we have an algorithm, say p. Use the algorithm p to transfer the top n minus one disks; the top n minus one disks, from a to b, right?

When you are moving this top n minus one disks, the largest disk is here; it remains here, so it is a sub-problem of shifting the smaller n minus one, to the second disk. Once you have solved this sub-problem, then you transfer the largest disk from a to c; there is no

problem in doing this, because at that time, c is empty. And then, recursively, again, transfer the n minus one disks, that were in b to c, right? So, what we have here is, suppose we say that the initial problem was, to transfer n disks from a to c; then, we are basically decomposing this into 3 sub-problems- that transfer the top n minus one disks from a to b, then, transfer the largest disk from a to c, and then, transfer the n minus one disks that we are moved to b, so they are now moved to b to c.

We have to solve all this problems in order to solve the tower of Hanoi. This is one kind of decomposition of the problem. There can be many other ways of decomposing the problem, right? Once we have chosen a decomposition, the solving procedure is standard, but the objective here, is to determine that which of these methodologies of decomposing the problem works best for a given problem? Then, there are blocks world problems in planning; we will discuss this later, when we talked about planning problems, and also theorem proving, where, in order to prove something, we will see that there are different premises and different __ and the way in which we combine these to arrive at the shortest proof, is also a problem reduction.

(Refer Slide Time: 00:13:39)

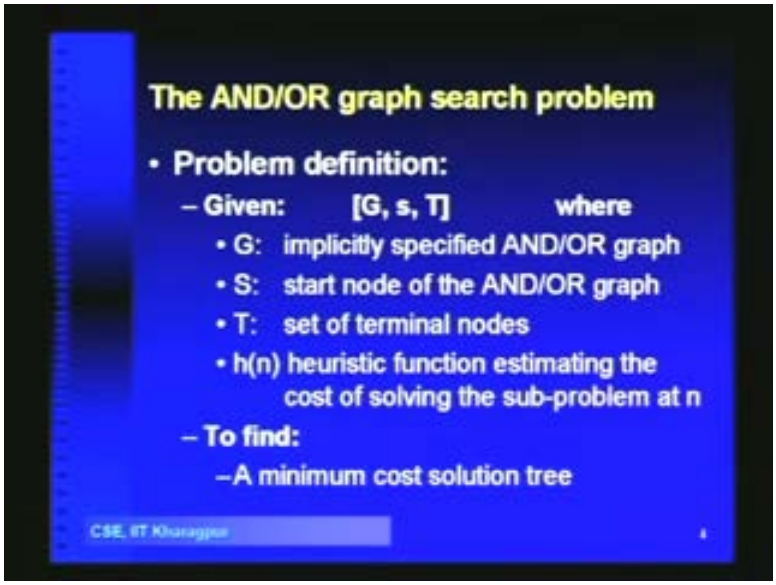


Let us see how we create a formulation of the problem reduction search problem. So, we will have an AND/OR graph, where an OR node represents a choice between possible decompositions. There can be more than one decomposition, and we will use an or node to choose between possible decompositions. I will give you an example- complete example, of what I mean by that. And we will have and nodes, which represents a given decomposition, right? It means that if you have an all node, then you have to solve any one of its successors; if you have an AND node, we have to solve all its successors. So, and nodes are actually the decompositions. And/or nodes will represent the choice between different decomposition. We will come to examples which will clarify this, and later on, we will study game trees, where we will- instead of and and or, we will have

max and min nodes, where max nodes will represent the choice of my opponent, and min nodes will represent my choice.

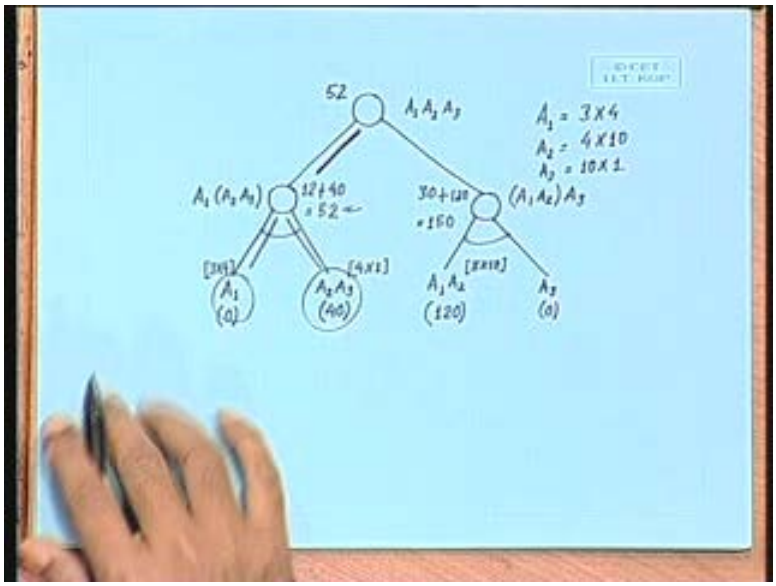
When we come to game trees, I will elaborate up on this, but these 2 formulations have a similarity, in the sense that both have 2 different kinds of nodes; each node has a kind of optimization criterion- like max nodes will maximize, min will minimize, and nodes will take the sum of the cost, or node will take again the minimum of the cost, if you are looking in a minimization problem.

(Refer Slide Time: 00:14:10)



So, this is the AND/OR graph search problem. We are given implicitly specified AND/OR graph, where the start node of the AND/OR graph, s is the start node of the AND/OR graph, t is a set of terminal nodes and h is a heuristic function that estimates the cost of solving the sub-problem at n. And we are to find the minimum cost solution tree. Let me give an example first, and then we will go into the actual algorithm for doing this. So, while I give this example, I will also give you an outline of how we plan to solve such problems. So, first let us understand what is what do we mean by an AND node and an OR node.

(Refer Slide Time: 00:20:03)



So, suppose we start with a problem that we have to solve. Let us say this is something like the matrix multiplication problem. I have to multiply matrices A1, A2, A3, right? Initially, let us say that I have an or node, which will say that I want to solve this as A1, A2, A3, and this one says, I will solve it as A1, A2 and A3. Now, clear, that these are the 2 ways of solving the problem? In order to solve it with this way, I have an AND node, which says that you have to solve A1, and you have to solve A2, A3. So, this is an AND node. And, here also, I will have a decomposition A1, A2 and A3. So this is an AND node, right? Okay. Now, the way I am going to evaluate it, is that I will find out- so, in order to solve this, again, I have to take the product of A2 and A3, right?

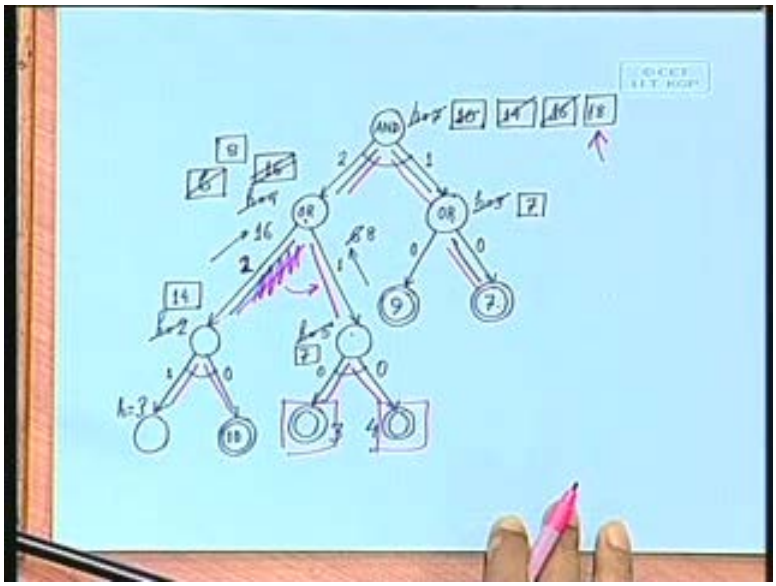
What is the cost of taking this product? Let us put some- put the dimensions of these, so let us say that the dimension of A is 3 cross 4, dimension of A2 is 4 cross 10, and A3 is 10 cross 1, right? If that is the case, what is the cost of this A2, A3? 4 into 10 into 1, so that is 40. So I associate a cost of 40 with this. And what is the cost of this? 0 is- nothing to multiply here, so this is 0, right? What is the dimension of this A2, A3? 4 cross 1, and the dimension of this is 3 cross 4, right? So, since I have to solve both of these, so therefore, what do I have as the cost of this operation?

This particular operation will have a cost of 3 into 4 into 1, so 12, plus the cost of what I have from this, so plus 40, so this comes to 52, right? Likewise, if you look at this- what is the cost of this? 3 into 4 into 10, so 120, right, and this has a cost of this has a dimension of 3 cross 10, right, and what is the cost of this? 0? Okay. What is the cost of taking this product? 3 into 10 into 1, so 30, plus this 120, so this comes to 150. In all nodes, I will be choosing; because this is a minimization problem, I will be choosing the least cost successor, so I will be choosing this one, right? Therefore, finally, what is my solution? It is the sub-tree that is rooted along the solution, if I take the arcs along- the selected arcs along the OR node. So, it means that this is the way to solve the problem.

First, take the product of A2 and A3, and then you take with product, with A1, that is going to give you a total cost of 52 which is the best that you can do for this particular problem. Let us consider, right? If you have many more matrices, then this tree would go deeper, and you will have more or nodes and more and nodes. But in and nodes, the cost of taking this product and this product will add up, plus the cost of taking the product of this and this, will be added to that, right? So, that is the cost of the and nodes. In the or node, I will select only the best among the successors, and the cost of that best successor will be backed up as the cost of the or node, right? Now all this step we have said so far, is in the absence of heuristic functions. in the absence of heuristic function

Just like we had in state space search, we can also have heuristic functions, that given a particular sub –problem, gives us an estimate of the best way to solve that problem. It gives us the least cost by which we are to be we can solve that problem. Again, we can have overestimates; again, we can have underestimates. So, at this point of time, we will assume that we have only underestimates, right? So, if we take heuristics into account, let us say let us see what happens. So, let us say that we start again, with another example, where at the top node, we have h is equal to 7. And let us say that this is an and node. this is an AND node

(Refer Slide Time: 00:33:12)



So, the way in which we will solve this problem is that, we will- just like we were maintaining open for state space search, we will maintain a marked and/or graph, indicating which is the best way of solving the problem up to now. So, if we have a and node, in order to solve it, we will have to solve both of its successors. So, we simply expand the and node and we get its successors. In this case, let us say that we have 2 successors, and then again, we evaluate the heuristic function in the child nodes. So, let us say that this gives us h equal to 4, and this gives us h equal to 3, right, and in general, we can associate some costs with the edges also.

These costs will be the costs of merging these sub-problems back into the original problem, like, for example, in that case, this was the cost of the multiplying of the sub-products, right? In general, we can have separate costs for the separate edges, right? So, now, you see- what do we have here? We have 2 successors; this one has an estimate of 4- it means that in order to solve this sub-problem, we are estimating that at least 4 units of costs will have to be incurred to solve this; at least 3 units of cost has to be incurred. And because this is an AND node, we have to mark both the successors, because both have to be solved.

The marked ones are the ones that we will eventually we want to solve. the once we eventually want to solve. So, now, we can revise this cost- we can revise this cost, because now, the cost has grown, because we know to solve this, we require 4 plus 2, 6, and from this side, we have 3 plus one, 4, so, the total of 10 is the new cost that I associate with this AND node, right? This 10 is also an underestimate, because this one will require at least 4, this one will require at least 3, and if I add up this cost, then the total is at least 10. Is it clear how I arrived at this figure-10- 4 plus 2 plus 3 plus 1? Then, let us say I expand. Now, see, when I have this marked sub-tree, it does not matter whether I expand this first or this first, because for the and node, I will have to solve both, right?

So, let us say, I pick up this one, and I expand that, and this, let us say, is an or node. We discover that it is an or node, and now it has 2 successors, right, and as we keep on expanding this, we will go to sub-problems and sub-problems and sub-problems, and finally, we will arrive at an leaf level, where those sub-problems are unit level problems, which we can solve, right? That is the end of the decomposition- the basic units of problems that we will solve. Let us say here, that these 2 that we have arrived at, are the basic sub-problems, right, and let us say, that solving this requires 9, solving this requires 7, and let us say that these 2 are zero, right?

So, I have 9 here and 7 here, okay? Now, because this is an or node, I can solve either this or this, right? I will choose the one which has lesser cost, right? So, I will choose this and mark this successor, right? Now, this cost is going to get updated. What happens is, I am performing a cost revision bottom up, so I update this one to what? 7, right? And then, again, update this one, because this cost has changed, so this will also change. So, now, this is going to be 6 plus 8, so 14. So, this one gets updated to 14. As of now, I can see that this is the best way of solving the problem, and up to so up to this part, that this is the best way of solving it- where I have still not solved this one.

I will expand this I will expand this node and let us say, that this is also a or node. So, I will expand that. And let us say that I get 2 nodes- this one has h equal to 5. This one has h equal to 2, right? h equal to 5, h equal to 2. So, you can use pathmax, if you want, into one where is heuristic less than that of - yes yes, so you can update it, if you want, right, so if- you can update it, if you want, using pathmax, as we have done previously, right? So, let us go on with this example and those optimizations, that you can update this; we can do it later. Let us say that this one has a cost of 1; this also has a cost of 1, right?

Now, see heuristic of this- suppose this has cost of 2, then you have nothing to worry about, because this is 2 plus 2, 4, which agrees with this, right? It is because 2 plus one is 3, and that is less than this, so if you want, we can make this 12, let us make it 2. Now, if we compare this, this has a cost of 4 and this has a cost of 6. Therefore, we will mark this one as the base successor. In an OR node, we will always mark the best successor, the current best successor. Now, if you mark this, then this cost has not changed anymore, this cost has not changed, so therefore, we do not have to do any cost revision beyond this point, because this cost has not changed. So, there is no need to go further up; this cost will still remain as 14, right?

And now, let us see- what is the best solution that we have? We solve this here, and this here, right? Now again, we want to expand another node now, which node we will select? This one. We are not selecting this node for expansion, because currently, this is not good; this is better- the OR node has picked this as a successor. So, we are always going to pick a leaf node of the marked sub-tree. In this case, the marked sub-tree, so far, is this. We will pick up, always, a leaf node of the mark sub-tree and expand that. In the marked sub-tree, we have this one, so we expand this, and this gives us again: let us say we have 2 successor, one has h equal to 3, and the other successor is a solved node, which is 10, this is 0, this is 1, let us say, and this is an AND node.

So, now, what is the cost? This cost is going to get updated; it is going to be how much? 10 plus 0 and 3 plus 1, 4, so 14. This cost has changed, by the way, because this is the and node; both of it successors will be marked. Now, we take this node and again, because this cost has changed, this whole cost revision step will have to be done. Now, this one will get replaced by 16- 14 plus 2, 16. Wait, wait. Now, now now at this point of time, we have to see which is the best for successor of this or node. If you look from this side, what you are getting up, what is being backed up is 16, but from this side, we now have a better one- that 6.

So, we are going to choose the less of the 2, and take 6 as the heuristic cost of this node. Now, does that change this- it does, because it has changed from 4 to 6. This is now going to become 16, is this clear? And now, the marked successor is this- this mark is not there anymore; this mark has shifted in this direction, right? So, the mark shifts when I find the better successor of the OR node, so the mark has shifted here, right? Then, I expand this, and let us say I get the let say that this is an AND node and I have this with a cost of 3, and this has a cost of 4, and they are solved node- terminal nodes. Now, what is going to happen? This is going to get updated to 7. This is going to get updated to 8, but this is still the best successor; this is still the best successor, because we are comparing between 7 plus 1 and 14 plus 2, so between 16 and 8, so, we will select this one, still.

This is going to get updated to 8, and this is going to get updated to 18, right? And now, we use- we are in a point where all the leaf nodes of the marked tree are solved. The best way to solve this problem, is by following the marked tree, and solving those problem. What we are going to do is, we are going to solve this problem- we are going to solve this problem, then we are going to solve this, then we are going to solve this, then we will solve this, that will solve this, and then we will solve this one, and the best cost of solving

this, is going to be 18. Is it clear? Is that alright? Yes. Coming back to here, you are saying that we have not yet solved this one. Our heuristic are underestimates, right?

This one is going to cost at least 16, if not more. If you go further down, your cost can only increase, because the heuristic cost are underestimates, whereas on this side, we have actually got a solution of cost 8, right? A complete solution of cost 8for this one, and in this side it says it is at least 16. Between at least 16, and exactly 8, we know that exactly 8 is always going to be better, right? So, we do not actually go to solve these things any further. The full and/or graph is not going to be explicitly generated; this is progressively expanding its best first, only up to the point where we have got the minimum cost solution, right? Now, if you think that we could have solved A* also in the same way, we could have taken the initial problem, except that in A*, we do not have and nodes; we only have or nodes.

To solve a start problem, there are several different choices in a star- the set of successors and then take the successors of those successors, and in this way, you could expand it out into an or graph, right? A graph which has only or nodes, right, and we could use the same marking strategy- always maintain the best cost successor at every or node, right? Now, if we did that, then we will still arrive at the same solution as A* does. At every point of time, the node that you are selecting for expansion will be the node that is along the best cost path from the start state, that is, the minimum cost node that we are always expanding. That will be the only node which is along the marked path, because we do not have and nodes, there will be only one marked path from the start state. So, try to think of a star in the context of and/or graph search.

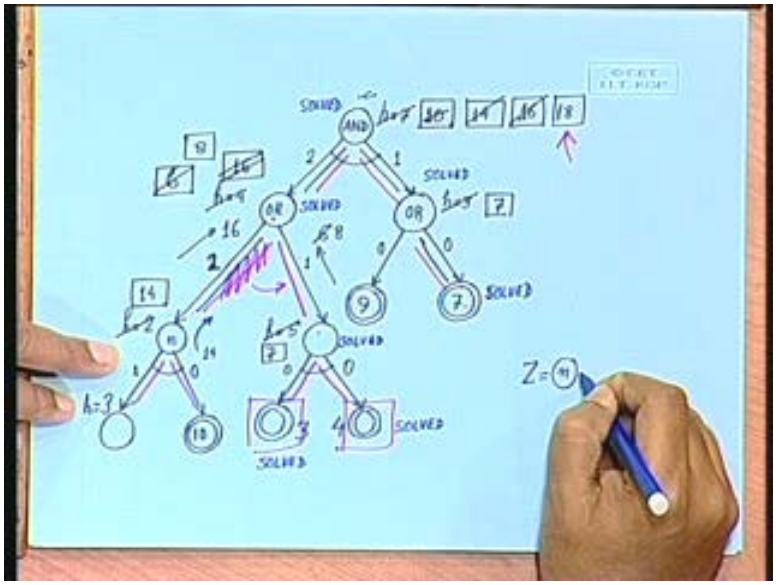
It actually is doing the same thing as well, but there we were maintaining open as a separate list, but if you think of open and close together, it is nothing but this tree. Let us look at the algorithm that we just now worked out. Initially, we will define a thing called g*. g* here, is the marked sub-tree, rooted at the start state. At every point of time, it will denote the sub-graph, which is which has so far being generated out and rooted at the state space. Initially, the estimate of the cost at the start state is equal to hs, because we do not have any other cost at that point of time. Now, if we find that the start state itself is a state belonging to t; t is the set of terminal states- terminal states represents those sub-problems which are unit level sub-problems, which we do not further decompose, but solve directly. So, t is the set of sub-problems, which are directly solvable.

If we find that s belongs to t, then we will label s as solved; whichever is a terminal node can be labeled as solved. Then, if a if at any point of time, the start state is labeled as solved, then we terminate. Let, now, when we are in or node, right, and the best cost successor is marked solved; then we labeled the or node as solved. In a and node, when all its successors are labeled solved, then we label it as solved; so we are coming to that. The third step is the select step- so, what we do is, we select a non-terminal leaf node from the marked sub-tree, as we were doing previously. At every point of time, we just look at marked sub-tree and select the non-terminal leaf node, then expand that node to generate the successors of this state n.

For each new successor, we set fm equal to hm; for each new successor mind you, we set fm equal to hm. If m is terminal-if any of the successors is a terminal node- then we label it as solved. And then, after we have done this, we will call cost revise, which is the bottom of cost revision step, and finally, we will return to step 2, right? Now, let us recall the example that we had done just now, where, well, it is all cluttered up here.

But if you remember, that we had initially started with the and node, generated it successors, right? Expanded the node, generate it successors, then marked for an n node. We marked both of its successors, right? And then, for an or node, we mark only the best cost successor, and we label a node as solved only when the best cost successor is labeled solved. When this is solved, and at this node, we find that the best successor is this- and the best successor is solved, so we can label this as solved. Then, we expanded this node, and initially, first time, we went this way- when we backed up this cost; when this was the best cost successor, it was not solved, that is why we never labeled as solved, at that point of time.

(Refer Slide Time: 00:47:20)

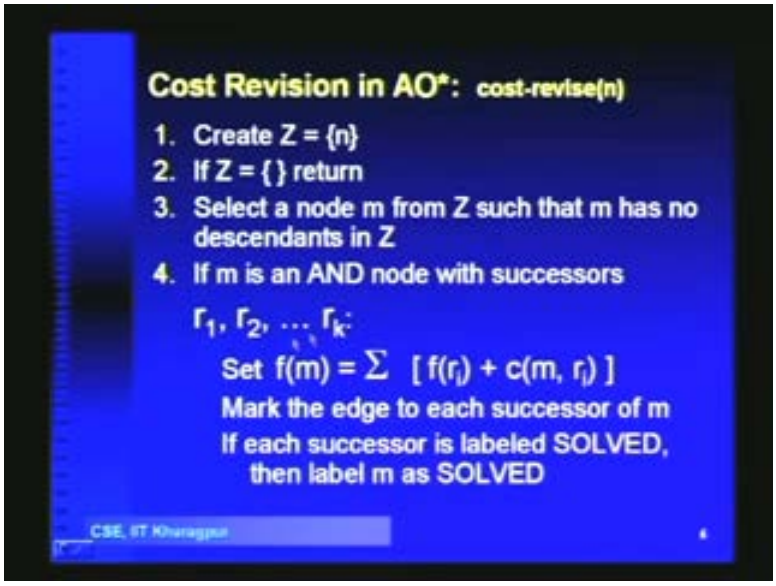


But when we went this way, and found that this was solved, this was solved and then because this is an and node, both of its successors are solved, so, we labeled this as solved. Then, when we went back to this or node and found that the best successor is solved, that is when we label this as solved, right? When the best successor is labeled solved, we label it as solved. Why? Yes, because the other along the other directions, we will only have more costs, because they are all underestimates. And if this is the better, then those underestimates, and this is the best that we can get, so we label it as solved. And then, when this AND node has both of its successor were solved, then we label this as solved.

And, if you look at the algorithm, whenever s is solved, that is when we terminate. Let us look at in details at this step, what do we do in cost revise? what do we do in cost revise.

That is where we will compute the costs of the parents from the successors, and in and node, we will decide to shift the marking if necessary.

(Refer Slide Time: 00:43:23)

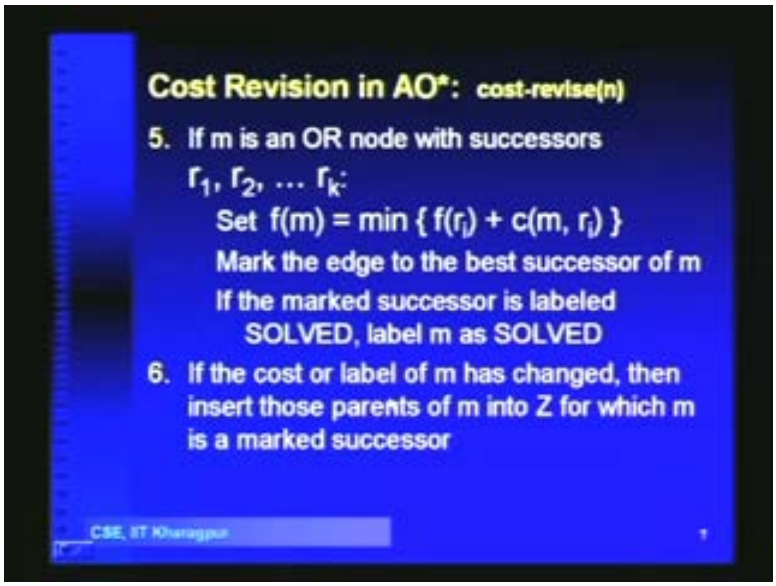


What we are going to do is, we are going to start with the node from where we want to do the cost revision. Let us say that in this, when we came up here and updated this cost, right, then, from here, we will do the cost revision backward. Initially, which- the node that we generate at the bottom, is the one from where we will start the cost revision. So, we found out these 3, plus one- 4, and 10 plus zero- 10, so 14. So, we start cost revision from here, right? Now, what we do here is, we create z equal to n, so we start with z equal to this state n, right? Then, coming to the slides, slides slides- when z is empty, that is, where we have done. Otherwise, select a node m from z, such that m has no descendants in z, right? Why do we do that? Because we want to go bottom up; we want to go bottom up, so we will always select first, the state which has no descendants in z.

Then, if m is a and node with successors r1 to rk, then we set fm to the sum of the child nodes, plus the cost on the edges, right? So, this is fri plus cmri, which is the cost of the successor ri plus the cost of the transition from m to ri, cost of the edge from m to ri. Then, we submit over all the edges, that that is because this is AND node, so we have to submit over all the edges, and then mark the edge to each successor of m. Because it is an and node, we mark the edge to every successor. And if each successor is labeled solved, then label m as solved, right? Okay.

After this, what we need to do is, we have to check whether, after this exercise, the cost of m has changed. If cost of m has changed, we will put it back into z- we will put it into z, so that in the next iteration or subsequent iterations, we also examine whether the parent of m is also going to change, right? What happens if m is a OR node?

(Refer Slide Time: 00:45:27)



If m is a OR node with successors r1 to rk, then similarly, we compute the cost of solving every child as fri plus cmri, but because this is a or node, we will find out the minimum among this. We will find out the minimum among this, and that is going to be the new cost of m. So, cost of the or node is the minimum of the cost of solving its successors. And then, we mark the edge to the best successor of m, and if the marked successor is label solved, then we label the or node as solved, okay?

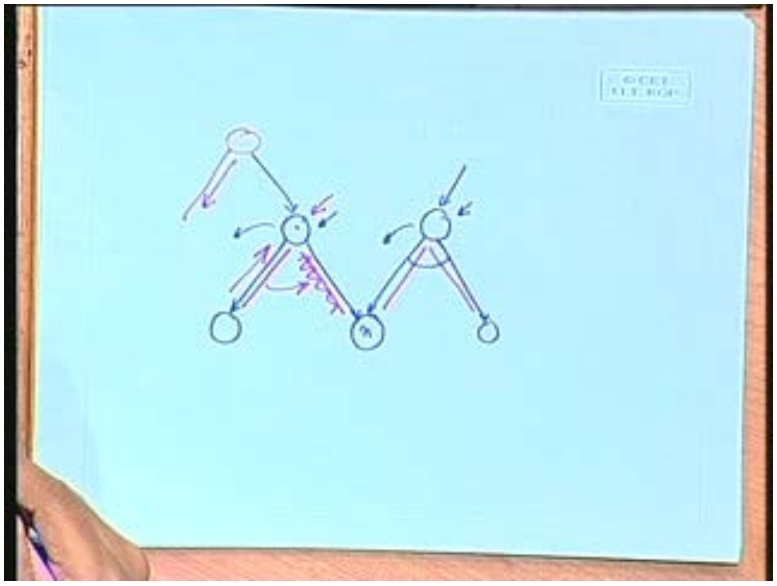
And then, finally, if the cost of m has changed- regardless of whether m was a or node or and node, if the cost or label of m has changed- then, insert the parents of m into z for which m is a marked successor. Now, you might find something interesting here. We are talking about- insert those parents of m into z. Now, when we are talking about and/or trees, as we have done here so far, there can be only one parent, right? But, in case you have a graph- and/or graph- then, you can have more than one___ . At this point of time, we are not talking about graph, right? If you are further interested in this area, you can read up what happens in the case of and/or graphs, okay?

In any case, any graph can be unfolded into a tree, so if you do not want to remember whether you are visiting the same node from 2 different paths, you can keep that as a and/or tree, is that alright? Okay. So, the question is, why do not we just climb up the tree and update all of them along that path, right? Now, that is essentially what we are doing; also what we are doing is, in the case of a tree- because there is only one parent- we can just what will happen is, it is you will have always z will n will have a single parent.

So, if the cost of n changes, then the cost- the parent of z will be put in z, and then in the next iteration, the parent will be picked up, right? And if the parent of that parent gets changed, then that is one which is going to get inserted. So, if it is the same thing that is happening; but in the case of a graph, suppose you have the state n, and this state n could

have been the or successor of some node, and could have been the and successor of some node, right? Or could have been the or successors of 2 nodes also.

(Refer Slide Time: 00:50:15)



So, in that case, you see, when the cost of n changes, the cost of this node can also change, the cost of this node can also change, right? Okay. So, as a result, we will put this also into z; we will put this into z, and we will also put this into z, right, so that both of them are again updated, with respect to their respective parents, right? So, because this algorithm that I have talked about, is for and/or graphs in general, those are examples that I have given this for and/or trees. But this algorithm itself is applicable to and/or graphs as well.

So, therefore, the cost revision steps that we have mentioned here, are actually made for graphs. (Student speaking).Yes. Yes. No no no, it can be, see- look at this examples in this or node. The best way to solve this or node could be this, and in this and node, both of these are marked. So, it is also the marked successor of this, and as well marked successor of this. (Student speaking). No, see, if it is not marked, suppose this or node is actually pointing in this direction. So, it is not- it is not selecting this. So, this option for this or node- this option is still better, right? Therefore, this cost is going to come up from this side; it is not going to come up from this side, right?

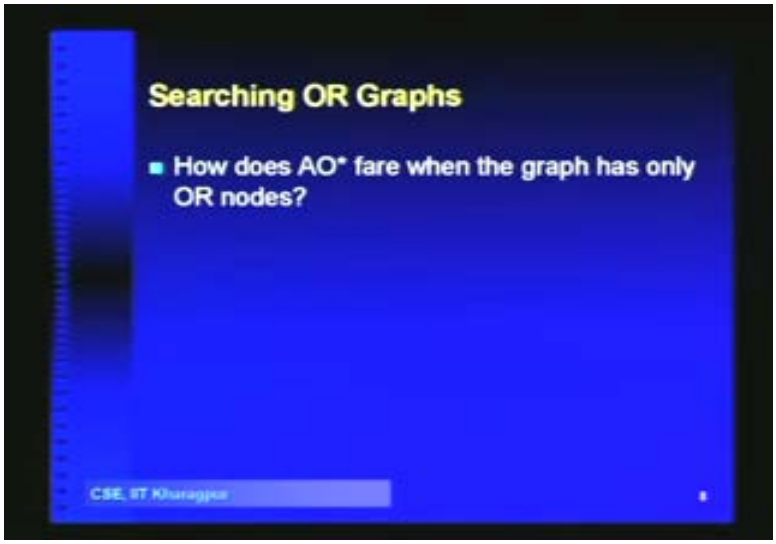
So, if there is if the the the parent node is still marked the other way, then you need not do cost revision beyond that point, right? If this fellow's cost changes, and this the mark shifts from here to here, then we need to put this back into z, so that this fellow's parent also gets a chance to see, whether, now, this becomes the better path, understood? Is that clear? Any other any questions regarding this algorithm? No, right? What I would suggest is, you take a few more examples by AND by gr graph and/or graph, right, and try to see how best to how you can apply this algorithm by and, to solve it. (Student speaking). Yes. If a node is marked as solved, it means that you have already found out

the best way to solve that sub-problem. So, for example, if an and node recursively- if an and node- all of you are successor solved, then you have solved that and node, because you know exactly how to solve everything from below that and node.

In an or node, if the currently selected successor is labeled as solved, then you know that you need not worry about the other successor, because those other successor are going to cost you more. So, the marked successor- if that is labeled as solved, then you know that if I follow along this mark, then I have already solved everything below that. So, I can label that node as solved. What are the initial set of solved nodes? The terminal nodes. The basic sub-problems, which cannot be further decomposed, are the initial set of solved node. There is a fixed cost associated with each of those terminal nodes.

So, that is the cost that you have to incur, that unit sub-problem which cannot be further decomposed, okay? So, a class- so, how does AO star fare? So, the this algorithm that we had so far, is called AO star; it is a and/or graph search algorithm, so we call it AO star and/or star. How does it fare when the graph has only OR nodes? Well, it fares exactly like we having a*, right?

(Refer Slide Time: 00:52:23)



So, the next topic that we will look at is searching of game tree. I will not start this topic today, because we are nearly towards the end of this lecture. We are not starting this topic; what we are going to see in game tree is, we will have 2 types of nodes- just like we have and nodes and or nodes, we will have max nodes, min nodes. And we will see how we can model a game playing problem, as a problem of solving game trees, with max nodes and min nodes, right?

So, the next lecture- we will talk about game trees and how we solve game trees, right? And, I also intend to wrap up a little of the previous things that we left on search, so that from the next to next lecture, we can get started with knowledge and deduction, okay? Thank you.