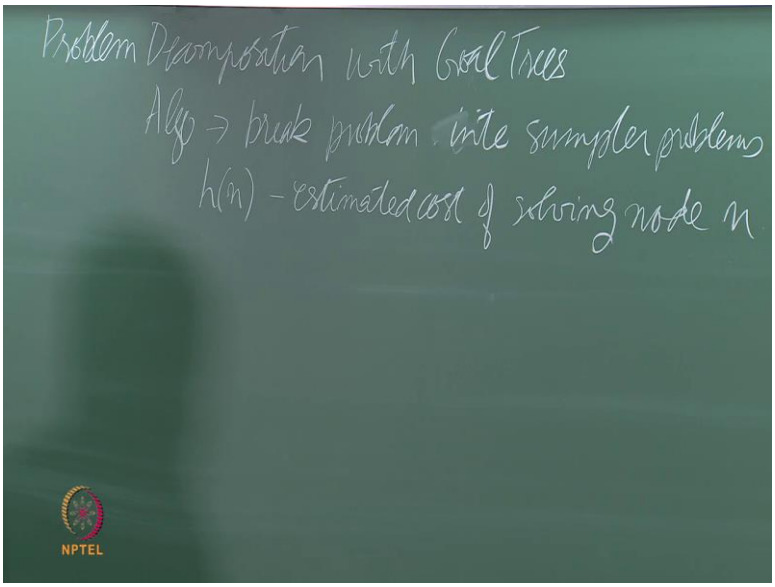
Aritificial Intelligence
Prof. Deepak Khemani
Department of Computer Science and Engineering.
Indian Institute of Technology, Madras

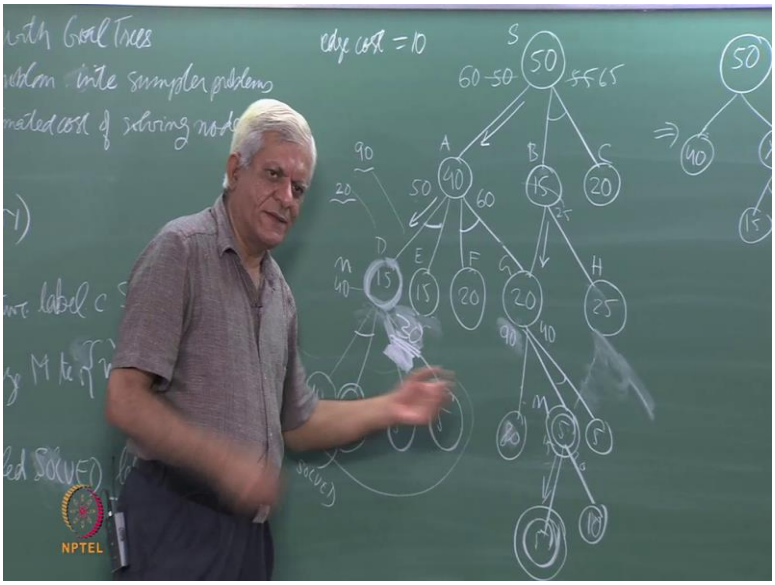
Lecture - 25
AO*Algorithm

(Refer Slide Time: 00:05)



We are looking at problem decomposition with goal trees, and we want to look at an algorithm, which will solve it. So, what should algorithm do? Break it down into simpler problems and we keep doing this, till the problems are severely small or primitives, that you have access to in some systems, especially. So, we will use, of course, one thing is to search the entire tree, but we do not want to do that. We will use the heuristic function, which is estimated cost solving node n. We will use the heuristic function to guide our search.

(Refer Slide Time: 01:43)



Let us say that we have some problem, which we are trying to solve. We compute the heuristic value, and let us say, it is 50, essentially, just to start with. When we start solving the problem, our estimate is that it is going to cost us 50 units of whatever. Let us assume for the sake of illustration here, that every edge has a cost associated with it, which is 10, essentially. Let us say that I get something like this; out of this. I get 2 edges out of it. One of them is, you might say, hyper edge or an edge; and the other one is simple edge. So, this is a tree like this. Let us say the cost associated with this is 15, and this is 20, and let us say this is 40, essentially. Now, in this example, I have mixed up the kind of edges, emulating from a node and if you think about this, it does not really cause a problem to see it like this.

We can think of it as two choices from the top level, and these choices already being broken down into two parts, essentially. You could have easily transformed this into a node, which says that 50 and then, some node x, and this is 40 and this x has got 2 nodes of cost 15 and 20. These two are equal in nature, and if necessary, we can always transform such a problem into that. Of course, you have to keep in mind, the cost of the edges. So, we have to take care of that somehow. But for the illustration purposes here, we will assume the cost of every edge is 10. So, if you have generated, if we have expanded, we will use the same term expanded a node. Here, expanded the node of this

thing and we have got these three nodes which is the next node that one should pick. So, let me call them A, B and C. This is the start node S. We will use similar term only. Between A, B and C, which is the next one I should expand?

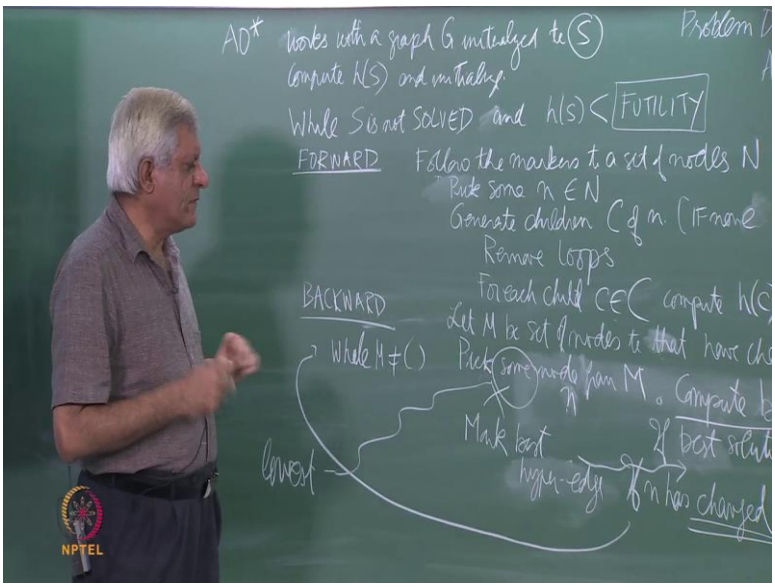
.A; why is it?

Yes. So, the cost of solving the original problem, if I take this option, would be the cost or solving this plus the edge cost which is, I can write this here; 50. And the cost of solving it from the other side is 15 plus 20, 35 plus 20, which is 55. So, obviously, this is better. So, I must choose this side, essentially. Then, let us say I get this, some values, and this is n node; this is an n node. Then, I can compute the cost of solving; revise the cost of solving A or estimated cost of solving A. So, instead of 40, this has become 30 plus 250 here, and this has become 40 plus 260, plus 20; 60. So, 20 plus 20 plus 10 plus10; that 60, and likewise, this is 15 plus 15 plus 10 for each hour; 60. So, I have to revise this cost now. This becomes 60, because this is 50, and this is 60; 50 plus 10; 60. So, at this moment, my algorithm was shifted tension to the other side. You can see, it has a best first nature about it. We are using a heuristic function to guide search. The heuristic function is estimating cost of solving a node. It is just that, because this is an end of graph or end of tree; we cannot make a decision, based simply on value of a node; we have to look at the cost of the full solution of which, this node is a part, essentially. Now, you can see this is consistent with what you were doing earlier.

We said that, for example, when you are looking at the travelling salesmen problem, and the branch and bound algorithm, we are saying the estimate is a estimate of full tour, essentially, and not just one, that edges we have found and so on, essentially. So, now, my attention must shift here, and maybe, I will expand one of these two nodes. Let us say, I expand this. Let us say for argument sake, this can be solved like this, or it can be solved by another node, which is of cost, let us say 25. This is 20, this is 25; this is an odd node. So, this cost is, I have to revise this cost to 25 now, which means I have to revise this cost to 65 and then again, my attention must shift this side, essentially. So, basically, what we need to do is to formulize this process, and sort of make this little bit clearer and put it down as n algorithm. So, let us now write this algorithm. This algorithm is called AO star and it essentially, finds an optimal solution for a problem like

this. When do we terminate this process?

(Refer Slide Time: 08:15)



Let us say, see, what has happened? This has now, gone up to 65. So, we will shift our attention to this 60, and we will come down here. If I note, if I pick up, if I just name them, let us say D, E, F, G and H; what is the next node I should expand? So, I have to pick a leaf. C is a leaf; H is a leaf; G, F, D, E; these I have already expanded. So, D, E, F, G, H and C; which one should be the next node I should expand? Looking at the heuristic values; just think about this, I will come back to this question in a moment. Let us write this. So, A star wolves with a graph, graph G, which is initialized to S, which is a single node S, essentially.

Then, it says compute h of s and initialize. Now, at any stage of the algorithm, I will have a graph which looks like that, essentially. I will go into some sort of a loop while; we have this lotion of the solved node and node, which is not completely solved. For example, in the integration example, something like, integral DW is the solved node, or in the chemistry example, we saw that once you know that complete structure of all the atoms, we are talking about, it is a solved node. We will use a label called solved, while. As long as, and eventually, we want to say that we have solved the original problem, which is a root S, essentially. As and when, we find the solution; we want to percolate

this label of solved, up, essentially.

So, this is a problem decomposition process, and there is going to be a secondary process in which, some labels will travel upwards, and one of the labels will be the solved label. Once a solved label reaches a root node, we can say, we have solved the root node. So, while, S is not solved, means it does not have a label solved. If you look at for example, rich and night, the book which describes this algorithm, that I am describing here; we use that some value call futility, which says that basically, we do not want solution which is more expensive than this value. So, we are going to explore for a solution or look for a solution, whose value is less than this number called futility. That is just an additional thing that we are using. Do the following. We have to pick a node to expand. Of course, in an initial stage, the graph is only one node, but at something like this stage, the graph has evolved. So, let me get back to this question.

I started by expanding this root node s; I got these three nodes then, because this was cheaper than these two combined. I expanded this next; I got this option and this option. So, let us call them DE option and FG option. The DE option costs 50; the FG option costs 60; and add another 10 for this option. So, this becomes 50 plus 10; 60, which was worse than 55. So, I came down to this option. Now, here, I choose between B and C; one 50. This B expanded so that, I got this 20 and 25. When I back up these values, I got 25 here, and that became 65. So, I have to now shift my attention to this one, again. Because going down, that pass is estimated to cost 65, along this, I can cost 60. So, I must come down here. At this point, I must choose this option, because this estimated cost is 50 and this, here, the estimated cost is 60. So, I must choose one of these two nodes for expansion next, essentially.

How do I do that? In the graph, I maintained at every choice point, a marker, which marks the best choice at that point, essentially, or at that stage of the graph. So, for this graph, at this stage, I would have a marker here. Here, I would have marker here. Because, here this is a better choice, here, this is a better choice and here, I would have a marker here. So, this algorithm is going to be a two stage process. In the first stage, I will identify the nodes to be expanded, and that I will do by starting from the root every time. In every cycle, in this while loop, we will start from the root, follow the markers, and a

marker will take us to a set of nodes. We have to pick a set of nodes. So, the forward face is as follows. Then, follow the markers to a set of nodes. Let us call them N and then, pick some n belonging to N. So, we have picked a node. In this example, I have to pick either D or E. Does it matter which one I pick next; whether, I should D or whether, I should pick E? In this example, of course, both the costs are 15 so, obviously, there is no way of telling, which one I should pick.

But let us say, for some other example, this cost was 90 and this cost was 20. Let us say that D had a cost of 20, and E had a cost of 90. Then, of course, they look different. Will it matter if I pick D or if I pick E? You have to think of it from the perspective of reaching a solution, and the perspective of saving on some amount, of course. If in an OR graph, you are searching something, like A star, if you have to choose between two nodes and you chose one and then, you are committed to having a solution to that nodes; I mean, that if that happened to be the cheapest, essentially. In a AO graph; AND OR graph, if you are at an AND choice like this, if you have to, if the solution is along this selection, you will have to eventually solve D, and you will have to solve E also. So, you will have to solve both of them, essentially. From that point of view, it does not matter whether, you pick D or whether, you pick E. Because, you have to solve both if you have to solve, if the solution lies, if the sub tree lies somewhere here, that is one perspective, but supposing, this are the values 20 and 90; is there some other reason for trying to solve one of them? Now, you can think of this as if you are solving C and F kind of a formula, then you are exploring whether, a sub formula.

Let us say f1 and f2; you have to try to solve both. If you have to solve for this formula, which has f1 and f2; you have to solve f1 and you have to solve f2. It is not necessarily C and F; it is some formula and, of course, f 1 and f 2 are themselves compound formulaes. If by looking at the size of those formulaes, does it make sense to choose something, which looks harder or something, which looks cheaper, essentially? So, this is the same problem here. Here, D looks cheap. E looks expensive. Does the estimated cost; should it influence my choice having said that, I have to solve both anyways, essentially. How will it help? No, No, No. You are not listening to what I am saying. If you solve D, you have to also solve E; why because this is an AND node here.

Then, you do not have to solve E. Then, which one is not likely to give a solution? E; the more expensive one, I think, or if you keep in mind, the fact, that we have some bound on the cost of the solution, that we are looking for. So, if you have to do two things any way, and your success depends on doing both the things. If one of them is harder, then you should see whether, you can do that harder thing first, because then, you hope of help solving easier thing later, essentially. The idea being that if you solve for D and then, if you solve for E, then the solution cost shoots up then, we will anyway, have to shift attention to another side, essentially. So, that extra work could be waste, whereas, if you try to solve the more expensive thing first, then you will get to know early whether, it is too expensive or not. So, that is the only idea, but if you have to solve for both you have to solve for both.

We will just assume; pick some nodes n from N; generate children C of n. We have something like the move gen function, which is not like the old move gen function, but something, which defines the problems into different sub problems; gives you the set, essentially. If none, if I do not have; if this node n does not have any children, my system is not giving a child, which means, it means my system is not telling me, that I can decompose it in some way. What should I do? Let us have picked this node n. Let us say this is my node n that I have picked for illustration purposes. Now, I am saying; generate the children of n, which means, it has further solutions expanded, essentially. What if there are no children? What if it cannot be expanded? What if it is a dead end of some sort? What should my search do? If I cannot expand this, I do not find any solution. What should my search do? You have to speak up a bit. I cannot hear. What does back tracking mean here? Should I go and look at E then; no. Then, I should somehow abandon this whole solution of which, this is a part, because if I cannot solve D, then I cannot compose the solution, which has D and E as their parts. So, I should abandon the whole thing; how do I do that?

I simply say that cost of solving n is futility, because my algorithm is looking for something which is less than futility, which means, this will never be considered henceforth, essentially. So, generate children C of n, if none; then you do this. Then, you remove loops. What do I mean by this; remove loops? I am talking in the perspective of AND OR graph. In the OR graph, of course, we have the social of loop, looping that.

You could just go round the same path. Here, what does looping mean? Remember that AND OR problem is, decomposing the problem into simpler problems. If you think of, for example, symbolic integration that we looked at; can we have looping? How can we have? What do you mean by looping? Can we have looping when we are decomposing the problem? You said; you nodded your head.

Student: Integration, it could happen like, when we apply that formula, we will get the same integral part again.

So, it is possible that you to do one transformation; you take sin by 4, by cost by 4; sin rise to 4, divide by cost rise to 4, and you get tan rise to 4 and then, you apply another transformation, which will, let us say, take you back to the same thing. So, it is possible. Looping is possible, because transformations are not one way; transformations can be two way in many situations. So, you want to avoid that extra transformation. Therefore, we move by mean by loops, essentially. Then, otherwise add for each child c belonging to this C; compute h of c. This is a forward phase of the algorithm, that in this situation, I follow the markers. I will come to a set of nodes; n. So, in this example, n is these two nodes; D and E. Then, I say, pick some node n from this set. I have picked D from there. Then, I say, generate the children of this c; this is my set c, and this is the values. So, let us say, I have two choices for solving D. Let us, for argument sake, that each of them is 10 and this is 5 and 5; let us say for argument sake.

So, what I am saying is that for each child, c belonging to this set of children c; this set is c. For each child in this, compute the h values. So, this completes the forward face, essentially. Now, in the backward face, I have to propagate the new cost up, essentially, and I have to readjust the pointers, the markers as I go along. So, that is the second face. Forward face goes from the root to the leaves; the backward face goes from this set c, subset of the leaves, to the root. So, what do I want to do? I want to; I found this heuristic values 10, 10 and 5, 5.

Now, I want to first change the heuristic value of n, essentially. So, this is 10 and 10, 20 plus another 20; this becomes 40, and this is 5 and 5, 10 and plus 20; this becomes 30. I have to now, say that the revised value of this node D is 30, and this is the best part to

follow, if you are solving this node D. So, I have to put this marker here, and I have to revise it to 30. So, this is the new value. These two things I have to do, correct. This was my node n; I expanded this; I got these children; I evaluated the heuristic values, and now, I have to propagate this value up. Till what? Till what stage, this propagation goes? It should go till, as long as this node changes? If this node changes then, I must propagate its value to its parents as well. So, let me take another situation, or this one. Let us say, this is a node that I have. Let us say, this is 10, 5 or let us say, this is 30, 5, 5, and this is 5, and this is 10.

Let us say this was my node n. I just want to illustrate this idea. If this was my node n, and these are the two children; I will evaluate this children, and what is the best way of solving this n? I will mark with the pointer and compute the cost, as 5 plus 10; 15, and this is 10 plus 10; 20. So, that is not the best one; this is the best one. Because, this has changed from 5 to 15; I will add its parent here. So, this will become 5 plus 5, sorry, 15 plus 5, 20 plus 20, which is 40. This was originally, also 40, because this 30 plus 10; 40. Let us say, this was 80, to start with; otherwise, I would not have gone there at all. So, if this was 80; this was, this is on 90. Now, after I revise this, I must revise its parent, because its parent is getting affected; it has got from 20 to 40. Because, I am revising this, I must revise both the parents. I must, because I know that the cost of this has changed; this cost must be propagated to this, as well. Whether, I came from this path or whether, I came from this path; it does not matter. I must revise both the parents. So, the process of propagation is basically, to go up to all the parents. If you have got a feel of that, just write it down.

Let M be the set of nodes, that has changed. This is just a comment. So, initialize M. The node that I expanded, I put this into the set M, essentially. Essentially, M is going to be the nodes, whose value has changed, essentially. So, I could have done the next step before this, but it does not matter. Let me write it here.

It says that compute the best cost of n. Pick some; let us say this node p or something like that, or just a way consistent, let me use n; I do not know which is better. Let me use n, because I will just only put n into this m, initially. So, I just use the same name, but of course, as m grows, it may have more elements. How will it grow; because parents will

get added to this set m, essentially. But initially, of course, there is only m inside n, and I am just simply, saying, using the same name, saying, and pick some node m from n. Compute best cost for n. How do I compute best cost for n? For example, if this was n, then I can see that from this side, the cost is 40, and from this side, the cost is 30. So, the best cost for n is 30, essentially. I should have checked that, you know, anyway, you are going to revise the cost, irrespective of whether, it improves or not. So, compute the best cost for n. I must do another thing here, before that. For each child c, compute h of c; I must have another step, which says that if c is primitive, label c solved. If the child, that I have just generated, is the terminal node of the primitive or trivial or whatever, you want to call it, are solved; then put a label solved for this node c. For example, if this was solved, then I could just put a label, saying solved. If I knew this was solved, the others may still have to be expanded and all. So, for every child, you compute the h value, and if it is a solved node or primitive node; you label it as solved. Then, n is the node that we just expanded, and we have added this to the set m.

We are picking some node from m. In this case, it is only n to start with. Compute the best cost for n, which has changed. If a solution below is labeled, solved; label n solved where, I should not say, a solution; is the best solution, say, if the best solution below it is labeled solved, you label it solved. If I take this example again, the other side, supposing, this was n; this is a better node. If this is solved; I will use double circle to denote a solved node. If this is solved, what do I need to do? I need to update n from 5 to 15, that is the new cost; that is the best cost for solving n. Initially, I thought of it is going to cost 5, but after I expanded it, I know that either, I have a choice of 15 or choice of 20. The best cost is 15, but this is labeled solved. So, I must label this also, solved. That is the step here.

If the best solution below is label solved, then label that node; solved. If I have to label this node solved, then the better solution, which is this side; both of them must be labeled solved. If both of them are labeled solved, then I can push the solved label here. In this fashion, the solved label will percolate up, and I want to keep doing this, till solved does not reach the root node, essentially. This is the mechanism for the backward phase. So, this is the best solution below n is solved.

Then, you put label as solved. So, we have done two things; compute best, and possibly labeled it solved. If any of these things happened, we will say that node n has changed, which means, it was 15 originally. It has either changed to 30, in this case. In this example, it has changed from 5 to 15, and also, it has got a label solved, essentially. So, either a change of heuristic value, or the application of solved label means that the node has changed. If a node has changed; I am using this as a kind of technical term, which you have to have a test for doing it. If n has changed, then add all parents of n to m.

So, all parents; that is what I was trying to illustrate here, is that if this has changed, then this will change. If this is changed, not only must you propagate the change of cost, from the direction you came; but also, to any other parents that this may help because in the future date, that parent might become, essentially... So, I must put this, of course, in the loop, while m is not equal to empty; I have this full loop. Pick some node form m. Compute its revised value, which is the best value from all the sub solutions, it has.

If the best solution below it, is labeled solved; label that node solved. If the node is changed, add its parent to m. What will happen? Initially, I will add only this node n to m, because it has changed; I will add to this parent, and because this has changed; I will add this parent. In this way, the revised value will propagate up. In this example, if this has changed, I will get a new label solved. So, I will add this to m, but this cannot be labeled solved, because you know, that branch still is unsolved. So, I will not label this solved, but I will change its value. Because now, it was originally, 20 and this has become 15 plus 5; 20 plus 20; 40. So, from 20, it has got to 40. So, it is changed. So, I must add both its parents to solve; they will change to m, sorry, they will change and I will add root to m.

After I change m, I will add nothing to m and eventually, all these ms will get revised. All these loops will terminate, which means the backward phase will terminate. So, the algorithm works in two phases; in the forward phase, you move forward following the markers. I have missed out one step here, somewhere here; I must have this thing marked as best sub solution, or I should say, hyper edge, because each of these are the two hyper edges; just mark the best ones. So, whenever you are looking at the node m, node in this m, you must recompute its value, which will get from its children. You must mark the

best path, because next time, you are going in the forward phase; you must know which direction to go, and you must also percolate this solved label back, if possible, essentially. It is possible, if all the sub solution; if this entire hyper edge has nodes, which has label solved, essentially. Just to repeat, if this was to be labeled solved, and this was to be labeled solved, then this will get labeled solved. If this was to be labeled solved, and this was to be labeled solved, then this will get labeled solved. The moment it gets labeled solved, and if this is the best one, if the arrow still pointing to this; that will get labeled solved and then, you will stop. In this example, this has got labeled solved; this will get labeled solved, but this will not get labeled solved, unless that gets labeled solved, and this is the better choice. So, I must have a marker here, and a marker here.

Once this gets slipped, if this has to get labeled solved, and the marker was still here, then I can label this solved and so on, essentially. So, in the forward phase, you follow the markers, end up with some unsolved set of nodes, which we have called as this n. You pick some node from n, may be, the most expensive as we discussed, a little bit earlier. So, that is not really so important. We pick some node from n, generate its children. If none, we call it futility. We assign the cost futility of that. Then, we remove loops and for each child, we compute the h values, that are this set of nodes here, or it could be this set of nodes here, depending on what is n. There, the forward phase ends. Then, you add this node n, that you just expanded to this set m; initialize m to this set n.

Then, one by one, you pick. So, this is not correct; this sum is not correct. I should pick the lowest. So, you can construct examples to show that you have to make this right. If m is the collection of nodes, pick the one which is lowest in terms of the graph. Pick the lowest node; compute its best cost below that. If the best cost leads to label to solved nodes, then label these nodes solved, and if either, it is not labeled solved or if its cost has changed, you must add its parents to m.

This loop here, this whole loop here, works under the condition as long as, m is not equal to empty. You pick some node and do this process, essentially. This process is essentially, backing up the values from the leads, towards the root node, essentially. Once this ends, the algorithm will go back to the forward node; again, follow the mark

path; again, expand some node. Then it will go into the backward phase, backing up the values from there. These forward and backward phases will keep happening, till either, one of the two things happens; that either, the value solved is percolated to the root, or the value futility is percolated to the root. So, in which case, either, it will give you a solution; solution would be a sub graph or a sub tree, or it will say that we cannot find the solution of this cost.

So, there is only one more point left. For this algorithm to be called AO star, for it to deserve the suffix of star, we need a condition on the heuristic function. Can you guess what the condition is? So, let me, Sorry?

Student: (( ))

What does it mean? This is the optimal cost solution and algorithm is designed to find the optimal cost solution and what here; saying here is that as long as the heuristic function under estimate the optimal cost, then it is going to be, give you an optimal solution. So, as an exercise, I will ask you to construct a small problem like this, with some leaf nodes, solved nodes and so on. You can basically, manipulate the edge cost to make it of under estimating or over estimating, and first work with a over estimating function, and you will see, that it gives you some solution, which may not necessarily be the optimal solution. Then, try out with an under estimating function, and all you could go up and look up the past test papers, you will see one such, one of such problems there, where an AND OR graph is given, and you have been asked to find it solution. So, do try it out at home.

In the next class, we will take a diversion from here. We will move to games primarily, because I want to give the games assignment in early October, so that, you can finish it by the end of October, essentially. Then, we will come back to this idea of expert systems, tool based systems. I mentioned things like, R1 is called an expert system or prospective. How are they implemented, and what is a rule based system, and how does it work? So, we will come back to that after, we finish with games.

So, we will stop here.