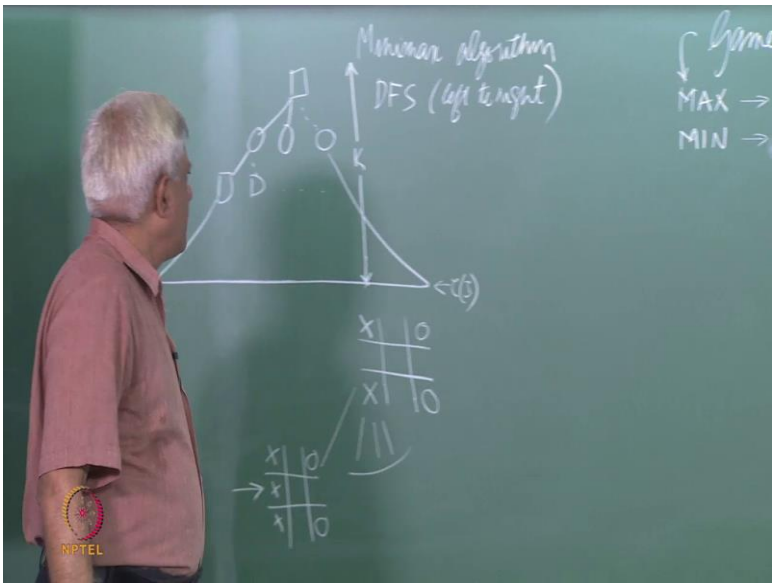
Artificial Intelligence
Prof. Deepak Khemani
Department Of Computer Science And Engineering
Indian Institute Of Technology, Madras

Lecture - 28
Game Playing Alpha Beta

(Refer Slide Time: 00:15)



We are looking at game playing and in the last class, we saw the minimax algorithm. If you remember, what the algorithm basically, does or what the game playing algorithm does is that there are two kinds of players; one is max, and the other is min. Max is trying to maximize the board value and min is trying to minimize the board value. The game tree consist of alternate layers of max and min. To starting with max node, there are some min children, and then, there are max children, and so on. This is the tree which minimax algorithm explores, and we saw that this explore, minimax algorithm does, is depth first search, left to right.

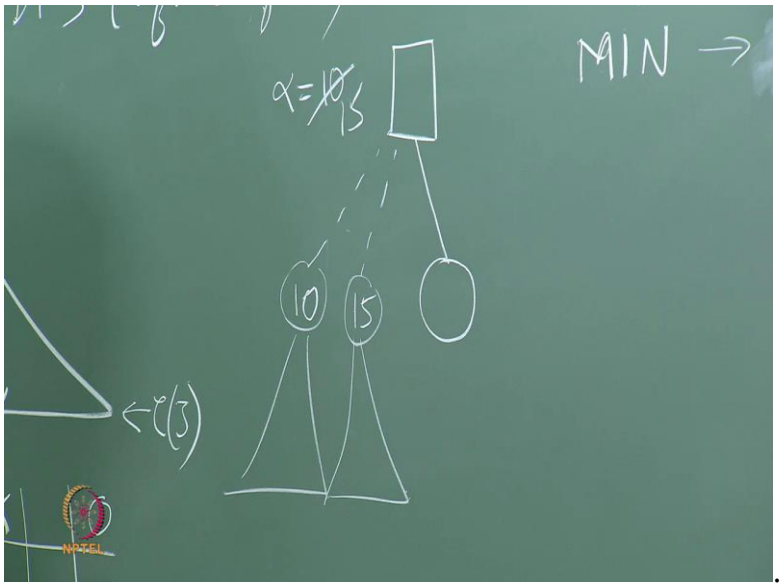
Now, today we want to look at improvement upon minimax, which does not inspect this entire tree, up to this cape lie. So, this is cape lie search that we are doing. Can we do without looking at the entire tree? What minimax is doing is going down all the way, to this level and at this level, the evaluation function e of j is applied, and then, the values of these leaf nodes are backed up using the minimax rule. The minimax rule says that if

you are backing up to a min level, you back up the smallest of the values of all the children. If you are backing up to a max level, you back up the largest of the values of all the children. So, at alternate level, we choose the minimum, the maximum, the minimum, the maximum, and so on. That is why, the algorithm is called minimax algorithm, but the question is that do we have to really inspect the entire tree, essentially.

To consider that, Let us first look at a small example. Let us say you are playing this game of Tick Tack Toe, and for some reason, this is how the game proceeds. Let us say you play this and we are not drawing the game tree; we will just draw the board. The opponent plays this; Let us say. So, the opponent is mirroring your moves. You play this and then, the opponent plays this. Now, Let us say you are doing some search from his point; one ply search. If you want to, now, consider this move. So, you play this move here, or you are looking at this move. Then, you can see at this moment that if you are considering this move, then there is no need to look at all these other children. Why because this is a winning move position, and you have won the game. So, why consider the other moves at all, essentially. So, this is the idea that we will explore, up to a greater depth, essentially, and the algorithm that we want to look at today, is called Alpha Beta .

A little bit of nomenclature, before we continue; max nodes are also called alpha nodes, and min nodes are also called beta nodes. Max nodes store alpha values, and min nodes store beta values. What are these alpha and beta values? These are the values of partially computed node, essentially. Let us see how these happen.

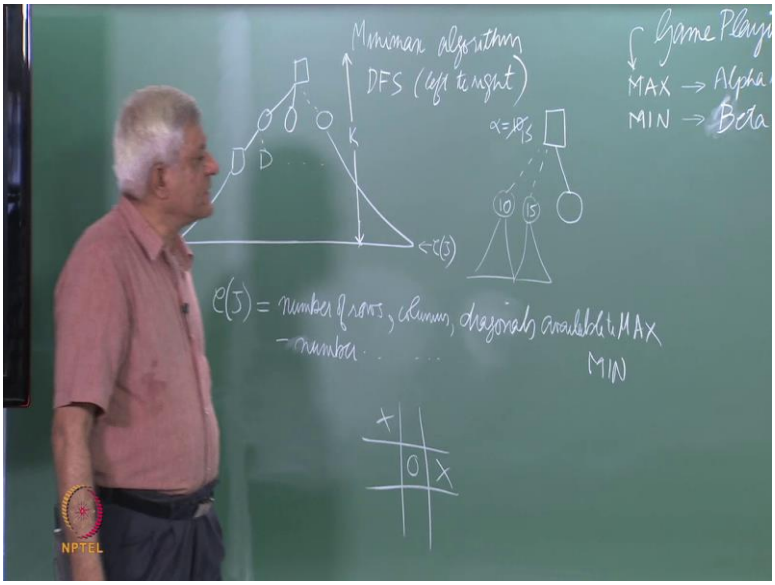
(Refer Slide Time: 04:49)



Let us say this is the route node, which is the max node. We always play the game for max, and at some point of the game, Let us say max is trying to evaluate this particular min child, which is not the first min child, because it has already seen; remember, that we are going from left to right; it has already seen some min children, and the sub trees below that. So, it has the already explored the sub trees this side. It is trying to, now, evaluate this min child. What will it do? If the value supplied by this min child is higher than the values supplied by all these children, then this would be adopted; otherwise, those values would be adopted. Let us say this value is 10, to begin with. So, we say that alpha becomes 10, after this node is completely evaluated; it means that sub tree below that is completely searched; alpha becomes 10, because that is a value, this beta node is giving to this.

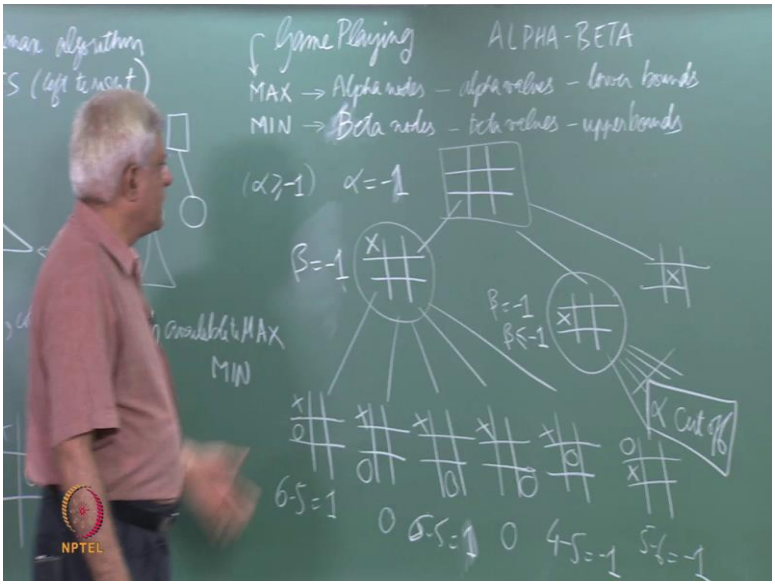
Let us say then, we evaluate the second sub tree below that, and this happens to be 15. Now, we change this alpha to 15. So, this alpha value for this max node is the value it has seen so far, and the value comes from the left hand side of the tree, essentially. Let us look at an example of slightly, deeper example of this Tick Tack Toe game.

(Refer Slide Time: 06:33)



Let us assume that we are using this following evaluation function, that e of j is equal to count of the number of rows, or columns, or diagonals; available to max, minus the number of the same thing, available to min. So, essentially we will evaluate a board position by saying that how many are available to each side. For example, if this is the board position that we are looking at; let us say this is the board position we are looking at. Then, we can see that max has this row, one, two, three and four; two rows and two columns are accessible to max. For min, it is this column, this diagonal, and this row; so, 3 to min, essentially. So, the value of this board position would be 4 minus 3 is min, essentially. So, we will use this evaluation function to illustrate this idea of cut offs, essentially.

(Refer Slide Time: 08:07)



Let us say we are starting the game from the beginning, and then, first we explore max, playing at this corner, here. Now, remember, the game playing algorithm are used as combination of search or look ahead and evaluation function. So, they do not evaluate this board position at all. They would look ahead a little bit for evaluating the board position at this, and then, back up the value essentially. So, let us say about, we are doing two searches that is easy to defect here. So, we look at one more level. So, at this level, let us assume that we are looking at this move for, when we are going look at all the moves for min. Let us say we look at this move for min.

We evaluate this board position. Now, if you look at this board position and count them carefully, you will see that there are six rows or columns or diagonals available for max. So, 1,2,3,4,5 and 6. So, 1,2,3 sorry, 1,2,3,4,5,6, and if we look at min then, they are 5; they happen to be 6 minus 5. So, the board position of this is 1, essentially. Now, this is a max node. So, we are going to compute alpha values for this. So, let me draw this as the max node, and this is the min node, because it is min play here, and these on max nodes, but it does not matter. At this point, you can see that beta will become 1.

The moment we evaluate this position and this node knows that this beta is 1. Now, what are beta values? Beta values are values of min nodes, and they are upper bounds on the

values that they can take. So, beta values are upper bounds, and likewise, alpha values are lower bounds. What do we mean by this? We mean that once, this node has seen one value from a left child, which is 1, it is not going to accept any value, which higher than this value. From the remaining children, it is only looking for lower value. So, this beta value, which is the partial value, it has got from here, is an upper bound on the value of this node. It can only be 1 or less, essentially, less, if one of the children evaluate to less, essentially. So far, we do not have an alpha value, because none of its children is completely evaluated. So, we look at the second child. Let us say this is the second option, we are looking at. This, we can see, is symmetric in nature, that both are at two corners. So, the number must be equal to 0. So, I will leave that for you to verify, and let us see this one and this one. Now, if we count this, you will see that max has 1, 2, 3, 4, 5, and min has 1, 2, 3, 4, 5. So, let me know if it is wrong; 5 minus 5 is equal to 0.

One thing, we should have done. The moment we saw this 0, we should have changed this value to 0, because beta has gone down to 0. Now, 0 is the upper bound on this value. Then, we look at this or may be this one is 1. 6 minus 5 is equal to 1. This one is again, symmetric. So, it must be 0, whatever, the count is. Finally, we look at one more move for win, which is this. Now, this turns out is max as 1, 2, 3 and 4; only 4 available, and min has 1, 2, 3, 4 and 5, minus 5 is equal to minus 1. Now, we have a new value for beta. So, beta becomes minus 1 and now, this node is, of course, completely evaluated, which means, it can say we can think of these as suppliers.

So, beta nodes are suppliers to alpha nodes and below them, alpha nodes are suppliers to beta nodes and so on and so forth. So, beta nodes always choose the smallest value, and alpha node will always choose the highest value from what its suppliers have given. So, here we can see that alpha is equal to minus 1; that is the value this first beta node is supplying to it, which we can read as saying that alpha is going to be greater than equal to minus 1, which is the characteristics of an alpha node, essentially. So, from the other children that we going to look at for max, it is only going to be interested in a node, if its value is going to be greater than minus 1.

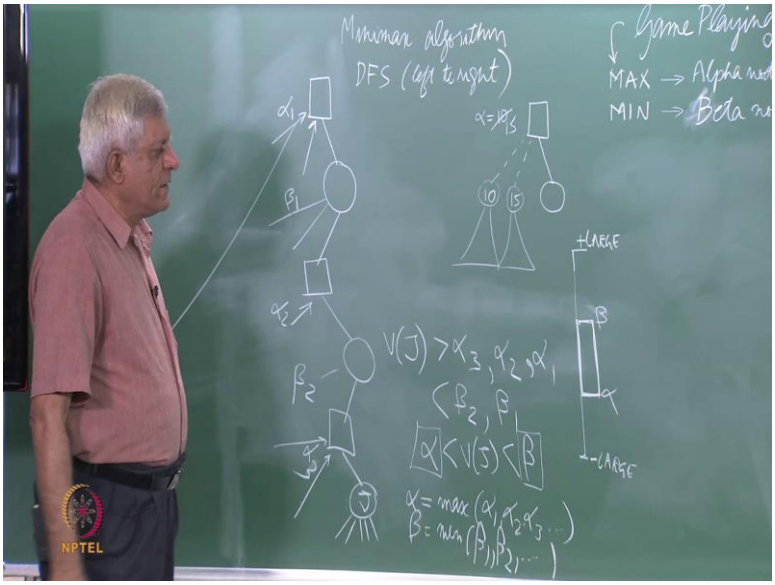
Let us try the second option. Let us say max try this option, and we try the first option for min, and see this is the first option, which I have for min. We always begin from the top

left hand corner, let us say. Now, we can see that, we have already seen this position. This is the opposite of this position, and the value of this is 5 minus 6 equal to minus 1. The moment we see this, we know that this is minus 1, and by this, you remember that beta is less than equal to minus 1. Now, here we have alpha is already equal to minus 1 and this beta says, that I am going to be minus 1 or less. Though this alpha, in some sense, will tell this beta that we do not, I would not be interested in you any more; that you do not need to value it yourself any further; which means all these other children that beta was considering, like this five children we had here; there also, we would have five children. It would not be evaluated.

So, this is the cut off which takes place, and this is called an alpha cut off. So, an alpha cut off appears at a beta node, which is a descendant or in this case, a child of alpha node. It happens when the beta node promises to be worst than or lower than or not higher than, rather, than what alpha already seen, essentially. So, after alpha has seen one side, it will now do this in a very control fashion, which is only, as long as we have better than minus 1, I am going to be interested in you; otherwise, do not explore the sub tree below that, essentially. So, this will get cut off and then, alpha will try; this is a third option. So, if we ignore symmetries, I mean, if we take into account symmetries, then you can see that max has only three moves to start with, either corner, or on a side, or on this. This small move, I will leave as an exercise for you to explore. The idea is basically, that a cut off takes place, when there is enough information, essentially. If we were doing 10 ply search, for example, then the entire trees; 8 ply trees below this, would be cut off. So, the saving would be considerable in amount, essentially.

The algorithm that we are looking at today; this alpha beta algorithm, essentially, is like minimax, in the sense that its searches from left to right, but it does cut offs along the way. We have illustrated one cut off, which is alpha cut off, which you can also think of as alpha induced cut off, and it happens at the beta node. So, beta node stops evaluating itself, if the parent alpha node tells it to stop evaluating itself. Then, it is an alpha cut off. In a similar fashion, beta cut off will take place at an alpha node, or it could be thought of a beta induced cut off, essentially. Now, these cut offs do not necessarily, have to be at the immediate level, essentially. They can happen at a much deeper level. So, let me illustrate that with a diagram. Let us say that you are evaluating this deep game tree.

(Refer Slide Time: 18:02)



This is the route node, alpha node and we are, this diagram, I am basically, repeating it here. We are evaluating this beta node and you have evaluated some part of the tree; so, always the left side of the tree, you have finished evaluating. So, we have got some value from here; alpha 1; from this side of the tree and essentially, what this node is trying to do is to see, if it can get a better value, better than this alpha 1 value. What is this alpha 1? Alpha 1 is the best amongst these here, and it is time to see, if this node will supply it a better value, essentially. Likewise, this node may be looking at a alpha child, and it may have got some value, which we will call beta 1, from the left tree, that it has explored on the left side, essentially. So, this process continues. This has got some value alpha 2 from here.

Then, this has got some value beta 2 from here. Then, let us say this is a node. Let us say that this is a beta node that we are about, we are trying to evaluate, essentially. Now, if this node we will call j, which means, we are evaluating this slowly, by looking at its children. The question we want; so, this has alpha 2 here. So, I hope this diagram is clear. Just imagine, this that depth first search, sweeping from left to right, and all these values are coming from the left side of the tree; alpha 1 coming from the left children of this, when it is trying to evaluate this. This, in turn, is trying to evaluate this. It has got some partial values beta 1. This one is trying to evaluate this. This has got some partial

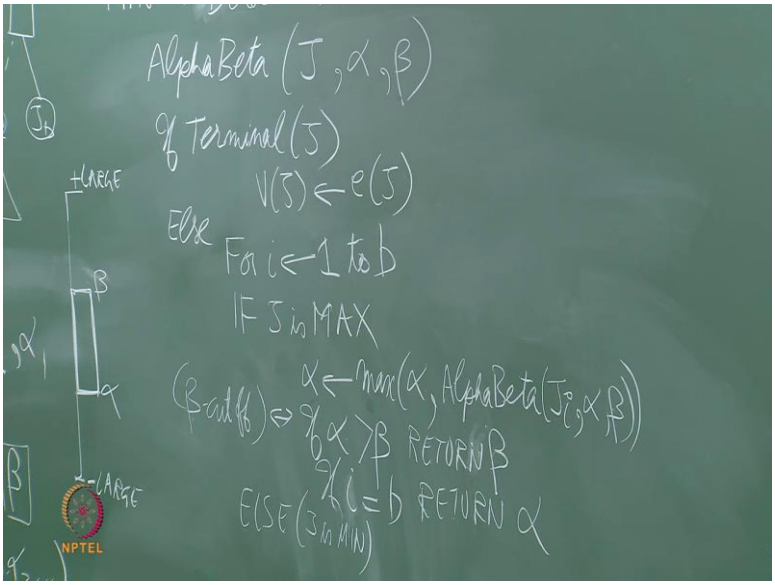
value alpha 2 and this has got some partial values beta 2. This has got some alpha 3 and this j is what we are trying to evaluate. The question we want to ask is; when will this j value reach the route, or in other words, when will this j value influence the game, essentially? Is it the node worth exploring? You can see that this j value will influence this node, only if it is better than alpha 3. So, that means, let us call it v j, that is the value of this node j, because we may be searching deeper, must be greater than alpha 3; otherwise, this node will take alpha 3 from here. Likewise, it must be better than alpha 2 as well, because otherwise, this will take alpha 2 and also, alpha 1.

Only if it is better than this alpha 3 value, because these are max nodes, remember, and they are looking higher values. Only if this node supplies the value, which is higher than alpha 3 and higher than alpha 2 and higher than alpha 1; will it influence the route node, essentially. Likewise, it must be less than this beta 2, because this is the beta node, and it is going to only take lower values and also, beta 1. So, we can generalize this and say that we need to evaluate this node j, only if v j is less than beta and is greater than alpha where, alpha is equal to max of alpha 1, alpha 2, alpha 3 or in general, all the ancestors of this node; all the alpha ancestors of this node. So, alpha must be higher than all these ones, and this value must be higher than that max of that. Beta is min of beta 1, beta 2 and all the beta ancestors of the node, essentially. I just repeat, this node is worth evaluating only, if it is higher than this alpha 3, and this alpha 2, and this alpha 1, and at the same time, lower than this beta 2, and this beta 1, and all the beta ancestors, essentially. So, this alpha and beta can be seen as the bounce, within which, we want the algorithm to search; otherwise, it should abort the search or prune that below that, essentially.

Let me write the algorithm first, and then, we will look at a slightly, more detailed example, essentially. So, I hope this is clear. So, we can think of this alpha beta as the window. If these are the values of the game three, then this alpha beta is a window. Beta is an upper limit and alpha is a lower limit. The absolute possible is plus large and absolute minimum is minus large. So, remember, we had said that the evaluation function can be something, like in a range of minus 1000 to plus 1000 or something like that; instead of 1000, I am writing plus large, some suitably large number.

What the alpha beta algorithm does is that for evaluating any node, it passes a window and says, only if you can get me a value inside this window, I am going to be interested in that. So, again, look at this node. The window is defined by this alpha and this beta where, alpha is a maximum of all the ancestors, and beta is the minimum of all the ancestors. What will the node try to do, because this example here, it is a beta node; it is going to try to pull down this beta and say, I want a value slightly, lower than this beta. If it, alpha node try to raise the value. So, as we sweep this from left to right, this window gets smaller and smaller, and search progresses only inside this window; otherwise, the tree is pruned off. Let us write this algorithm, or at least, I will write a part of it, and you can write the rest.

(Refer Slide Time: 25:01)



Alpha, beta; it takes an argument j, which is the node, and a value; alpha, and a value; beta, which are parameters, essentially. These parameters are basically, the window sizes that are passed on to the node, essentially. Let me first ask you what should be the value alpha and beta, when we call the game playing algorithm at the root, essentially? So, alpha should be minus large and beta should be plus large, when the first time we call it; that means, windows completely opened. Then, the algorithm is simpler. It is basically, a small variation of the minimax algorithm that we have seen. So, if we assumed that we have a function, which tells us, weather j is on the horizon or not. You can do this by

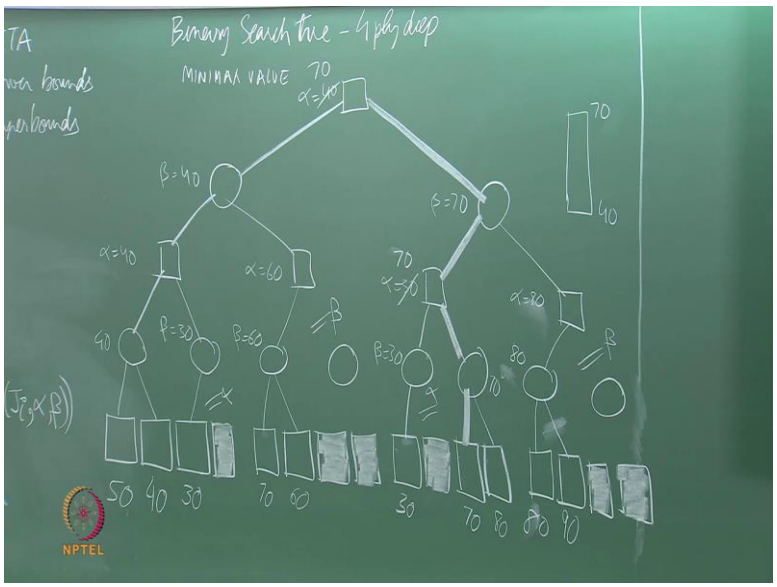
keeping some kind of a counter as you search deeper, so that, every time you make a recursive call, you also decrement the counter by 1. Let us say you are doing eight ply search. When you make this call, they count as 8; when you make this call, they count as 7 and so on. When the counter becomes 0 that means, it is a terminal node on the horizon. The implication of being a terminal is that you have to apply the evaluation function stop searching further, essentially. So, if it is terminal, then we can say that v j is e j where, e j is the evaluation function that we are using in the game. Else, we have to evaluate still then, which means you looking at a node; it is not a terminal node.

So, you have to evaluate all the children, essentially, one by one. So, for j, let us say i going from one to b. So, Let us assume that b is a bouncing factor, or every node has b children. We are going to evaluate them from left to right, essentially. If j is a max node, what do we want to do here? If it is a max node, like this one, for example, or this max node; it has got some alpha and beta bounce, given to it, and we want to evaluate this from left to right. So, this is j 1; this is j 2 and so on, up to j t; we want to evaluate from left to right, and we want to see, if we can get a higher value than what alpha has. Remember, it has to operate within this window, and seek a higher value, essentially. I will write the part for the max, and you write the part for min. So, if j is a max node, then what you do is; alpha gets maximum of alpha, and a recursive call for j i, the highest child, with the bounce alpha and beta; the bounce keep getting propagated. So, what does this step say? We are doing from; i going from 1 to b. So, for every i, we will evaluate the highest child, and if its providing the better value, which means a higher value, because we are using the max function here, then alpha. Then, we will update alpha to that, essentially.

If alpha becomes greater than beta, then we say, return beta. What does this mean? Look at this alpha value here. It has got this beta bound coming from the top, which is beta 1 and beta 2. If the value that we are trying to compute for this alpha becomes higher than this beta or this beta, it does not matter. Then, they are going to say that no need to value it anything here. So, we are making a return statement there, and say, return whatever beta bound is and that is the best you can do with this. So, this amounts to a beta cut off, because it is happening in an alpha node; it is dictated by the beta value; it is a beta cut off; otherwise, if j equal to b, sorry, i equal to b, which means you who have looked at

the last child and evaluated it, and you already done this, choosing the best of this values. You can return alpha. So, this takes care of the alpha side of thing. The other option is l s, which means j is min; I will not write this completely, but you can fill it up yourself; this would be similar if i equal to b, return beta. This would be analogous, which says beta is min of beta and the recursive call. This test will remain the same; alpha greater than beta, because this test signifies that this window has closed in some sense. This alpha value has gone above the beta value. So, there is nothing left to explore, but the return value would be alpha, essentially. So, you can fill up those details, essentially. Let us now look at a slightly more detailed example of this algorithm.

(Refer Slide Time: 31:43)



Let us assume that we have binary search tree, and it is 4 ply deep. Let us draw the tree first. We will have 16 leaf nodes and since, a binary tree; there would be beta nodes sitting here, and then, alpha nodes sitting here, and beta nodes here. This is the root. This is the alpha node. So, I have just drawn the space that is going to be searched; the space of nodes; we have not drawn the tree. What we want to see is how will alpha beta algorithm explore this tree, essentially. Now, observe that the minimax algorithm will explore this entire tree from left to right, and back up the value. So, these beta nodes will take the minimum of what they get from here. Then, alpha nodes will take the maximum of beta children. This beta will take the minimum of these alpha children, and so on. So,

you want to see what alpha beta does; we will fill in some random values, just to explore the algorithm here. So, it is during depth first search; left to right. So, it first goes and evaluates the first left most node in the tree. Let us, for argument sake, say that this value is 50, some random value, essentially. Then, it evaluates its next child. Let us say that is 40. So, at this point, this beta is completely evaluated. Its value is 40 and since, this is completely evaluated, this alpha is equal to 40, which, remember, you must read as saying that alpha is greater than equal to 40, essentially. Then, it starts the next round. It looks at this, and looks at this. Let us say this happens to be 30, which means this beta is equal to 30. Now, you can see this relation between this beta and this alpha node. This alpha is saying, I am at least, 40. This beta is saying, I am at most, 30. So, this alpha node is not going to be in this beta node. So, we have an alpha cut off. I will just write alpha here, to signify, it is an alpha cut off.

So, we are not going to look at this node, essentially. Then, we continue the depth first fashion; go here, go here, go here, and let us say that this is 70. So, this beta becomes 70 at this moment. Then, explore this, and this is 60. So, this beta is 60, now, and this alpha is also 60. This beta is 40, because it is getting this 40 value from here, and it is essentially, asserting that I am 40 or less. This one is getting 60 from here, and we say, it is 60 or more, essentially. So, which means, this beta will induce a cutoff here. So, this will be cutoff, and this is the beta cut off, or beta induced cut off, essentially. So, we are not looked at those two nodes, those two leaf nodes. In fact, we have not looked at the entrie sub tree there, essentially. At this point, this alpha is equal to 40 and again, we do depth first search, and let us say then, this value is 30.

Now, you remember this; a node is influenced or the bound that node gets, is influenced by all the ancestors. In this case, then, only one ancestor, which has a value, which is this root for this value is 40. Look at this beta node. This beta node is saying it is 30. Of course, it could go lower than 30, but at the moment it is 30. So, this now induced the cut off, what we call as a deep cut off. That, because this is upper bounded by 30, this node is never going to be interested in what is going to happen here; so, might as well, cut it off, and this is an alpha cut off. So, the alpha cut off does not have to be induced by a parent node. It can be induced by some ancestor, of course, which is what, we have said here, explicitly. When you have said that this alpha bound is actually, the maximum of

all the alpha bounds, essentially.

So, this alpha is 30. It is saying, I am going to be 30 or more. Let us say we come here; we come here. Let us say this happens to be 70, and this happens to be 80. Now, this becomes 70, and this value of also becomes 70, and this beta is 70. So, what is happening? Now, when we explore this child, it is getting the bound of, beta is 70 and alpha is 40. It still has the window opened, so, we must explore this tree below here. So, we do that in the depth first fashion. Let us say that this is also 30, then again, we introduce a cutoff here, very similar to these same values. So, this is now, 30 or Let us say now, this is 80. So, we do not get this cut off. We do investigate this node. Let us say this happens to be 90, and this happens to be now, 80. This says alpha is equal to 80. Now, look at this beta and this alpha. This beta says, I am at most, 70. This beta is saying, I am at least, 80. So, we have a cut off here.

We can see that we did not look at this node; we do not look at this node, and we did a fair amount of cut off. So, out of the 16 nodes, this alpha bit algorithm has not seen 6 nodes; it has seen only 10 nodes, essentially. As an exercise, I will ask you to fill in values, so that, the number of cut offs are maximum. At the same time, as a different exercise, fill in the values, so that, there are no cut offs at all. Now, it is possible that the values are such, that there are no cut offs, essentially. That happens, because the algorithm is searching from left to right. What does left and right mean? It basically, means in what order you are generating the moves, essentially. This is some game in which, you have two moves; let us call them, a and b. You are generating a first, and then, b, essentially.

Now, what this exercise will reveal to you is now, what is a mini max value of this game tree? This is 70, this is 40, and this is an alpha node. So, this value is actually, 70 where is it coming from? It is coming from this node here. This 70 is coming here. This 70 is going here. This 70 is coming here; which means that the game that will be played, if only on this analysis; would be that max would make this move; min would make this move, because that is what min can do best; it is getting 70 here, and 80 there; max would make this move, and min would make this move.

If you go and flip this tree left to right, which means, this game value would come on the left part of the tree. Then, you would notice that the number of cut offs are moved, essentially. In other words, if somehow, the best moves are made earlier, if they are made in the left part of the three, then the number of cut offs will be more. This, you can, sort of, understand by constructing an example in which, the best moves are on the left hand side. In fact, if you try to construct a tree in which, you fill in values, so that, the number of cut offs are maximum, you will see that the game value will come from this left side of the tree, essentially. Now, that has an implication for game playing programs. If you are writing a game playing program, you would like to generate your moves. For example, you are doing the Othello program, and you have some set of moves to generate. You would like to generate the moves as far as possible, in such a manner that the best moves are considered first, and then, the worst moves, essentially. So, the question is; how can you order moves, essentially? Remember, we are discussing this in domain independent fashion. Of course, you can apply some domain knowledge to saym these moves are to be considered, and so on, but in a domain independent passion. Any suggestions?

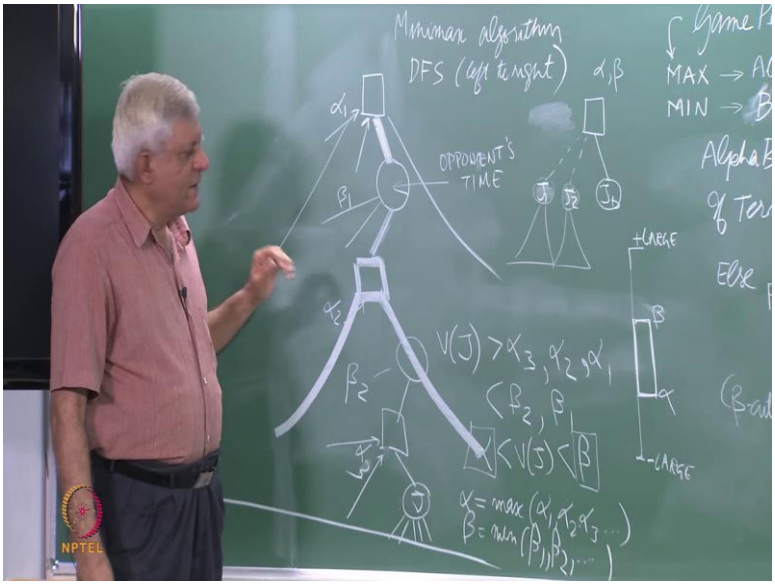
Student: Heuristically

Heuristically, but how do you choose that heuristically?

Student: Evaluation.

So, here is what many people do. Remember, that when we are playing a game playing program, you are doing some search up to some Cape lie, deep search, here.

(Refer Slide Time: 43:59)



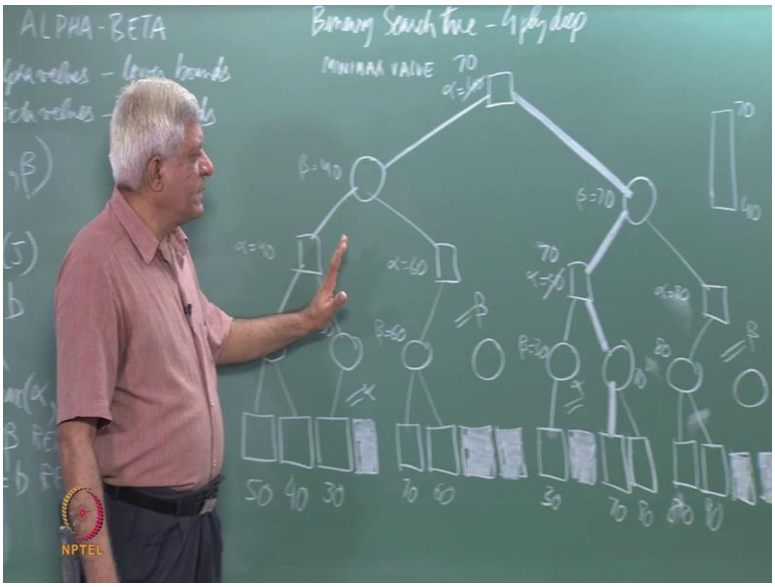
Let us say up to this level or whatever, and you decide to make. Let us say this happens to be the move that you are making, essentially. This is a move that you made, after you have done this whole game play search, essentially. Then, what happens? Then, its opponent turns to make the move, because opponent is going to play the next move. You have made a move, and opponent will play a move, but what you have access to, is your search tree, and has to below this max node. Now, you will have to your next move at this, assuming that this happens to be the best move for min, which will be the case, because your analysis search that you are considering the best moves for min. Min is taking the minimum value from all these. Let us assume that this is the minimum value that min can get. Next time, when your turn comes, you will have to play from here, which means, you will be searching the tree, starting from here, and going down, essentially.

Now, you can exploit the search that you did in the previous round, below this node, because you are getting values from the all the children. You order the children, so that, the higher order min children come first, and below that, the lower order max children come first. You can order the sub tree, so that, the best nodes are coming to the left side. If we do that you are likely to get more cut off, which in the real tournament environment, will mean that your move would be faster, and you would have more time

for the subsequent moves, essentially. You can also do some amount of analysis in opponent's time. This is opponent's time, here. What do the mean by opponent's time? It is when opponent is thinking. So, you have made a move, and opponent is thinking; what to move? That time, in the real world is available to you. You can also think of what opponent might move, and what you would respond; you could do this analysis in opponent's time.

The only thing is, of course, in the game playing assignment that we are going to give you; you will not get this opponent's time, because you will have a separate thread running. We should be invoked, only after opponent has made their moves, but in the real world game playing situation where, you take your own computer to play. Again, you do have this time to do some further analysis. One analysis that we can do is to try to order the moves in such a way, that the best moves comes first, essentially

(Refer Slide Time: 46:36)



Now, this algorithm suffers from this common trade that we have been observing throughout this course, which is that it is a blind or uninformed algorithm. It searches from left to right. Of course, this desire to order the moves is, in some sense, the desire the give it a direction, but given a fixed order of moves; can we have an algorithm, which will have a sense of direction, which is, which will be like a best first search algorithm?

Indeed, there is such an algorithm called sss star algorithm, but we will take that up in the next class.

We will stop here now, with alpha beta.