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

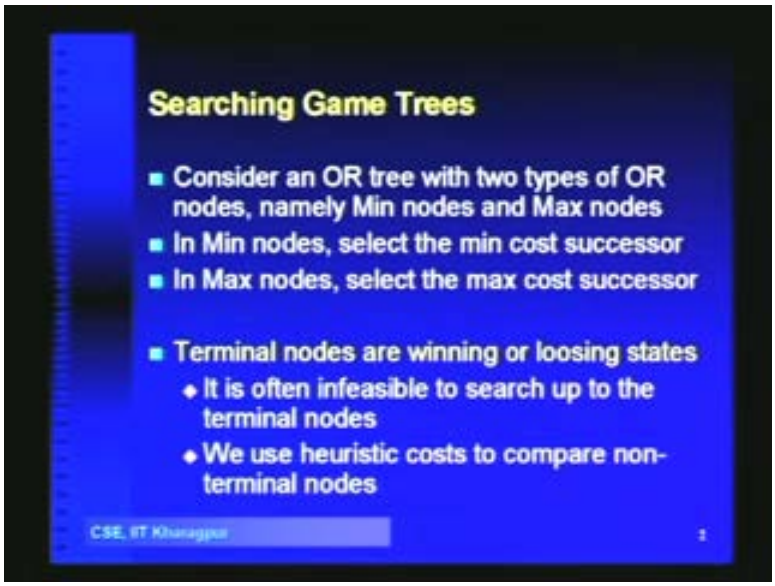
Lecture - 7
Searching Game Trees

In today's class, we will start on game trees, and we will see how we can model different game playing situations in terms of game trees. We will study some very classical algorithms for searching in game trees, and finding out when we are in a better position, and when we are in a worst position.

(Refer Slide Time: 00:01:15)

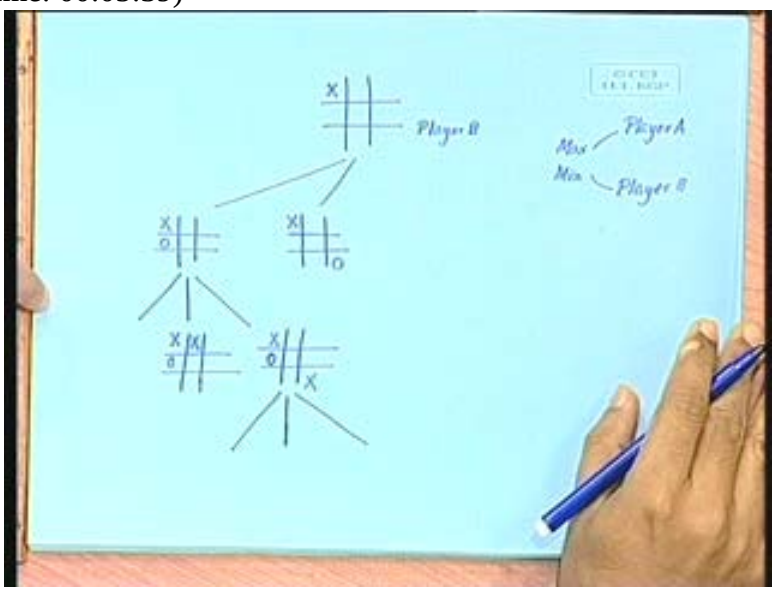
So, in this, on searching game trees, what essentially game trees are? They are OR trees, namely. We have already studied what AND/OR graphs are, and AND/OR trees are. Game trees are a special type of OR tree, but there are 2 types of OR nodes. By OR tree, we mean that at a time, we will be selecting only one of the successors of the OR node, but which one will be selected will depend on whether it is a min node or a max node.

(Refer Slide Time: 00:02:03)



I will shortly explain why we need this kind of a representation. And briefly, we in min nodes, we will select the minimum cost successor, and in max nodes, we will select the maximum cost successor. Terminal nodes can be winning or losing states, but it is often infeasible to search up to the terminal nodes. So, we will use heuristic costs to compare non-terminal nodes. So, let me start by showing a simple formulation of a game tree search problem. Let us say we take that example of tictactoe. In tictactoe, what we have is something like this, right? And the players alternate in putting either a circle or a cross, in any of these positions. Let us say, that we start with a configuration, where we put x here, right? Then, from here, so this is where the one of this is the starting configuration, let us say.

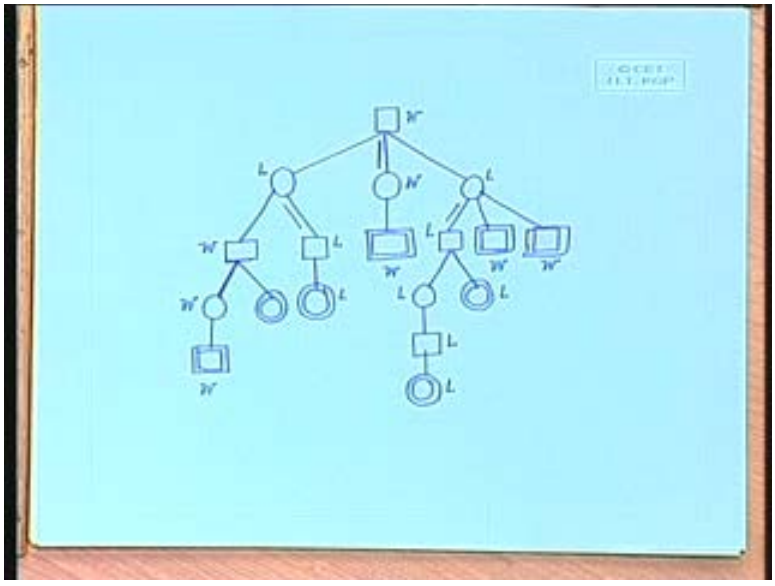
(Refer Slide Time: 00:05:39)



Now, it is the move of player b; so, when it is the move of player b, then there are various different moves that player b can take. Say, out out of which, this is one, or this is one, and so on. Then, for each of these cases, player a has a move. Here, let us say player b has a move. From this point, in each of these states, player a has a move. Suppose player a makes a move here, then player a can, again, take one of many different moves. Suppose, if this is the step, and player b can give something like this, or player b can give something like this, and so on, right? Then, you know that suppose this is the move, that player b has given, then, again, player a will have turns, but if you look further below this, then you will see that all- there is a way of playing, by which this fellow always wins.

Because, if now, the only possible option of this player is to put a dot here, and if that happens, then the other player can put a cross here, and then there is no way of saving the game, right? If you look below this sub-tree, then you will see that that for every move that this player makes, the other player has a winning strategy, right? Now, so, this is the structure of the game tree. The game tree is a tree where every node of the tree, represents a state of the game, and nodes can be either max node or a min node. I have not yet defined what I mean by a max node or a min node, but let us say that one of these type of nodes is for player a, and the other is for player b. So, depending on who has the move, we will distinguish between 2 types of nodes, right? We will go right down up to the winning, or we can go right down up to the winning or losing combination, and then decide what part to take.

(Refer Slide Time: 00:10:32)



For example, that if we start from one player, we will indicate player a by square nodes and player b by round nodes. Initially, player a has a move, and that can possibly lead to- say, 3 moves of 3 different steps are possible, let us say. So, this is move 1, move 2, move 3. Now, player b has a move, and let us say player b has 2 moves here- maybe

player b has just 1 option here, and has say 3 options here, right? Then maybe, from here, player a can have 1 option, 2 options, etc. This could be a winning option for player a, right? This may be a winning option for player b, and then again, suppose this is the scenario:

Let us see: we have win here for player b- for player a; we have win here, for player a, right? We have win here for player a, we have win here for player a. Now, our objective is to determine, that at this point, what move we should take, right? So, we will see, from here, that if this is a win node for player a, and since the opponent does not have any alternative choice here, so, this is a win node for player a as well, right? Then, here, because player a has a choice between these 2, so, this is also win node for player a.
Because, if we arrive at this state, then player a will take this move, and be able to win eventually, right?

Then, this is a losing this is a loss for player a, this is also a loss for player a. Now, here, because it is the choice of the opponent, so, therefore, at this point, it is loss for player a. Because if we arrive at this state, then, the opponent is going to push in this direction, right? Out here, it is win for player a, so out here also, it is win for player a, right? Now, let us look at this- this is loss for player a; this is also loss for player a. This is loss for player a, right? This is also loss for player a, so, this is also loss for player a, right? Now, in this node, because it is opponent's move, so, opponent will always push us in this direction, so, this is also loss for player a. So, that means- but, here, the option is with player a, so player a can always choose here, so, it is win for player a.

That means, overall in this situation, player a has a winning strategy, and what is that winning strategy? It is to take this move, right? If it takes any of the other moves, then player b will have a winning strategy. If it takes this one, then player b will push us in this direction, and it will be win for player b. If it takes this move, then again, player b will have a winning strategy by pushing us in this direction, right? Is it clear? What we have done is, we have seen that we can bottom up propagate this winning or losing, right up to the top, and then decide, that which is the move that is to be taken at this point of time, right?

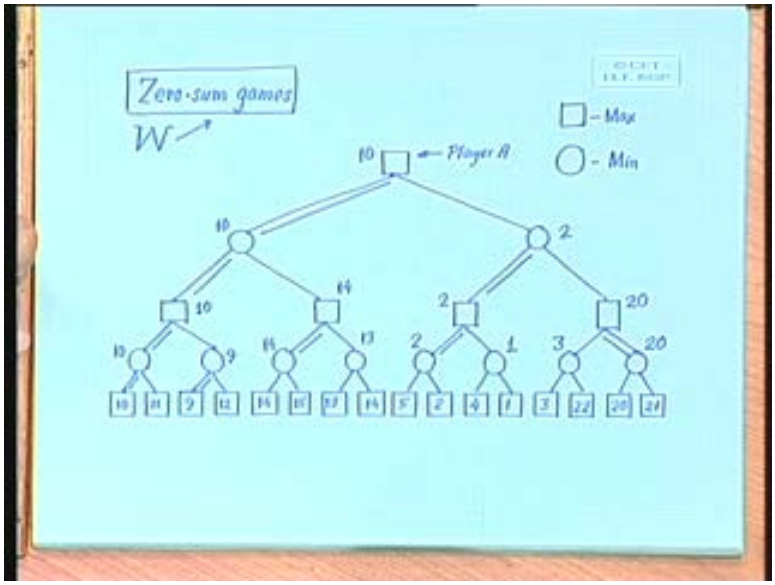
The current state is the root of the tree, and all other states are look-aheads. If I if I give this move, if he give that move- like this, okay? Now, in typically, it is very difficult to do this, when the game state space size is very large, like, you cannot do this kind of thing when we are dealing with something like chess, but tictactoe, yes, because tictactoe state space is not going to grow too big. So, if you write a program just like this, to go right down, up to the winning or losing configurations, you can find out, at each point of time, what is the best move that you have to give, and whether you have a winning combina winning strategy or a losing strategy, right?

For, when you go for games like chess, then you cannot do this, because the state space is just too big. So, what we will do is, we will expand these moves up to a certain depth, and we will have some heuristic functions to evaluate the position of the game after that many look-aheads. So, it means that- in chess, let us say, I decide to do 6 move look

ahead. So, I look ahead up to 6 moves, and at that level, I analyze all the board positions, depending what kind of positional and tactical advantage I have. So, that is where a lot of knowledge will come in about chess, and then I will evaluate those board positions and associate a heuristic cost function with them, which will indicate the probability of my winning from that position, right? Probability, or the amount of cost that I have to incur.

And then, we will imply what is called a minmax search, to determine what is the best move. For example, let us look at this game tree. In this game tree, we have looked ahead, up to this many number of moves, and then we have found out some cost criterion here, right? Now, what does this cost criterion means? It, these values tell me, what is the cost that I have to incur if I have to win? So, in other words, the lesser the cost, the more are my chances of winning; the more the cost, the lesser is my chance of winning, right?

(Refer Slide Time: 00:20:22)



So, we compute the heuristic function in that way. We associate a cost, which tells me the badness of the state in which I end up. For example, if I am here, then the badness is 9, right? Badness or goodness, depending on how I model, right? So, we will have 2 types of node here- one are the square nodes, which we will call max nodes. At these nodes, we will try to maximize the cost, and the square nodes are min nodes. In these nodes, we will try to minimize the cost. Now, let us understand- why do I want to maximize or minimize?

The kind of games that we are considering here, are called zero sum games. The zero sum games means that both players are playing for a share of some amount of cost. So, let us say that w is the amount of cost for which they are playing, right? So, this cost is going to get distributed between player a and player b. It means that if player a gets more, then player b gets less, and vice versa, right? So, if w if the the zero sum games idea came from a from the notion that w is zero; if somebody gets a positive cost, the other fellow will get a negative cost. If somebody has a profit, the other person has a loss, right? And,

to indicate this scenario, that one the the advantage of one is the disadvantage of the other, is what we want to represent by zero sum games.

Now, this is not always true in practice; it is not always true in practice, because there are cases where the 2 competitors are adversaries of each other, but the profit of one, does not necessarily mean the loss of the other, right? But, at the same time, there is a large class of game playing situations, where actually you are competing for the same resource. So, the win of one is the loss of the other, right? For those kinds of scenarios, we will have 2 types of nodes- max nodes and min nodes, right? Because the loss of one is the gain of the other, it means that if I am trying to maximize my cost, my opponent is trying to minimize my cost.

If I am trying to maximize my profit, then the opponent strategy is to minimize my profit. And this will happen in zero sum games, understood? Because my objective and that person's objectives, are exactly opposite of each other, right? In that case, if I have my heuristic function, to compute my goodness- if I evaluate the board positions, and I find that this is this has goodness of hundred for me, right, then if the total, w, is 100, then it goodness of zero for the other. So, if it is goodness of 50 for me, then it is goodness of 50 for the other, right?

So, as I am trying to maximize my profit, my opponent will try to minimize that. If you look at this game playing situation here, these represent my profit. I am the player a I am the player a, who has to make a move here. This game tree has been formulated, based on the fact that I am now I do have the move. So, I am player a; I have the move, right? When I have the move, I will expand the game tree up to this; these steps, and I will evaluate each of these steps, depending on how much profit I get. So, these represent the amount of profit that I am getting, right? Okay. Now, let us see what happens.

Here, my opponent has a move, so square nodes means my move, round nodes means my opponent's move. So, here, the opponents has has a move, right? So, what is the opponent going do here? Will minimize; so, he will pick up 10 here. So, it means that if I reach this state of the game, then I cannot expect more than 10, because the opponent will push me in this direction, right? Likewise, here also, this is a min node, so I will get 9, right? This is also a min node, so I will have 14. This is a min node, so I will have 13. Here, I will have 2; here, I will have 1; here, 3, and 20, right?

Now, if you look at this layer, this is again a max node, which means that, here, I have my move. In that case, if I were in this situation, I would know that going this way will give me 10; going this way will give me 9, so, I will always select this direction, right? So, I will go for 10. Now, see, we are also maintaining a mark here, we are maintaining a mark just like we were doing in your start. Again, out of these 2, because this is a max node, this is my move, so I will select this one, right? And get 14. Here, I will be selecting this one and get 2, here I will be selecting this move and I will get 20.

Again, in the min level, if I if the opponent is in this state, opponent will know that this way, I will get 14; this way, I will get 10, and remember that the opponent's objective is

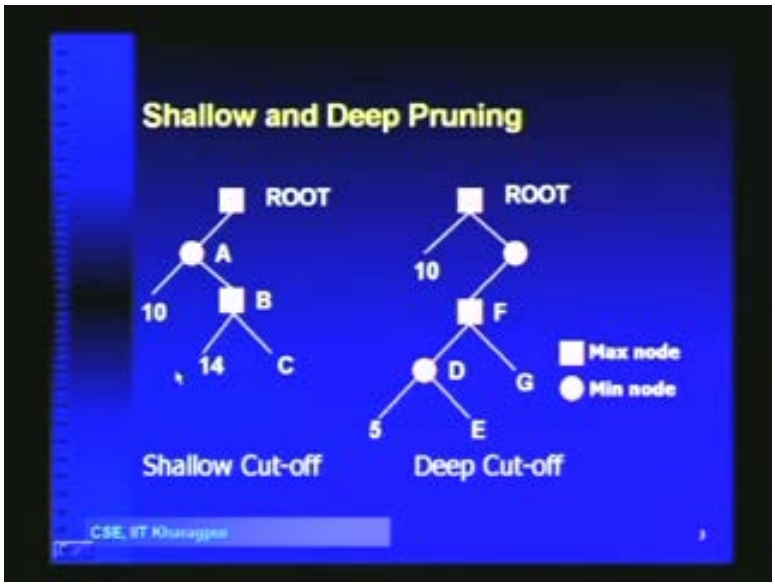
exactly the opposite of mine, because it is a zero sum game. So, therefore, the opponent will choose this and give me 10. Here also, the opponent will choose this and give me 2, right? And here, this is a max node, so it is my choice, so, I will select this one and I will get 10. That means that from this configuration, based on the heuristics that I have computed at this level, this is the best move that I have. So, it is if I take this move, the opponent is going to give me this; then, I will take this, then the opponent will give me this, so, that is what I will get.

Assuming that the opponent does not make a mistake- if the opponent makes a mistake, then, actually, I gain more. Then, I will have more profit. Because, just see- if the opponent, instead of going towards this direction, if you are push me towards this direction, then I have a strategy of getting 14, right? Because I will push it in this direction, and then, the opponent will have to give me either this or this, so I will have at least 14, right? So, at any point of time, what I am trying to find out is, I am trying to maximize my gains. Assuming that the opponent is an adversary, is an adversary who will never make mistakes.

And if the opponent makes mistakes, I will get more. If he does not make a mistake, then at least this much I am guaranteed, right? And this is a fair assumption, because we are talking about zero sum games, where my gain is the opponent's loss. Now, there are some optimizations that we can do. We need not actually develop the tree, right up to this point. We can actually do some kinds of pruning, by going depth first in the tree. And this brings us to some pruning criteria; this pruning criteria, and the algorithm called alpha beta pruning, was proposed by Donald Knuth many years back.

We will study that algorithm and see how we can avoid visiting the entire state space of the game tree. So, again, here, we will assume that the square nodes are max nodes here, and the round nodes are min nodes. So, let us look at this scenario. Here, we are going depth first, so, we find 10 here, right? Then, we backtrack up to a, then we are visiting b, and let us say, that somewhere in this tree, we get 14.

(Refer Slide Time: 00:24:38)



So, if we get 14 backed up from here, then the claim is that, I need not traverse down to c. And the reason is that: see, out here, the min node already has a 10. So, if you get anything more than 10 on this arc, then the min node is always going to go that way, because the min node will always minimize. So, if ever the cost of b exceeds 14, if the cost of b exceeds 14, exceeds10, rather; if the cost of b exceeds 10, then a will always select the arc towards this 10, right? Now, let us see what was happened here in b- this is a this is a max node, we already have 14, and the cost of the maximum node can only grow further, because it is max of all its successors. So, if it if you already have 14, and then there is no point visiting c, because whatever you get from there, b will be at least 14, and a is never going to give this move towards b- it is going to select the move in the other direction, right?

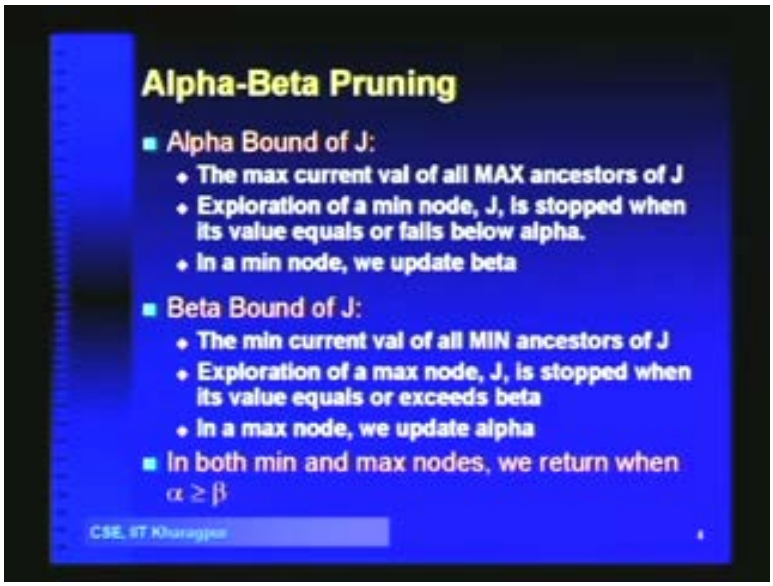
So, we can prune now, the sub-tree rooted at c, and not search that part of the sub-tree at-all right, is this clear? So, this is a case of shallow pruning, where the pruning happens with respect to some parent, which is close by, but we can also have deep pruning, like we have here. Here, my claim is that the sub-tree routed at e, can be pruned. Now, let us understand why. Here, at this max node, we already have 10, right? We have 10. Now, out here, in this min node, we have already got 5, right?

Now, by that, can we say, okay, out here, we have already got 5, right? Now, if you go here- further down here- then, what are you going to get? You will get something, and this cost is going to be less or equal to 5, right? But, if it if this is less or equal to 5, then it is of no interest for f, because unless f is able to produce a cost larger than 10, the root is always going to divert it to that side. Suppose, f is able to manage a cost which is less than 10, right? Then, when we are in this min mode, this min node will have a cost less than 10, because if f is not able to get few more than 10, then the min node will always minimize. And so, it will always give you less than or equal to 10, right?

If f is less than 10, then this is also less than 10, right? But if f is not able to master more than 10, then there is no point, because the root is going to select this way. Now, please try to understand this, but the root is always going to select this arc, unless it finds more attractive options, this way, and it can only find the more attractive option in this way, provided that the successors of this min node are all able to give you more than 10. But, if now, f is unable to give you more than 10, then there is no point in pursuing of any further. And now, what we have found here is, the d already has 5, and it will only go only further down, right?

So, therefore, there is no point in checking this, because this this branch, at least, is not going to give f more than 10. If f takes this branch, it is not going to take more than 10, right? So, we can prune it off at e. Is this clear? (Student speaking). What is the difference between- okay, when we are talking about deep cutoff, we will now introduce a couple of bounds. We have to formalize this, right? We will have to formalize, that when do we do this cutoff? So, deep cutoff actually gives us an idea- see here, this 10 is just not the immediate parents of d; it is the parents' parents' parent, and this can happen even if you drag d right down up to several more levels of min and max. So, let us formalize that, then you will understand why this deep cutoff is useful.

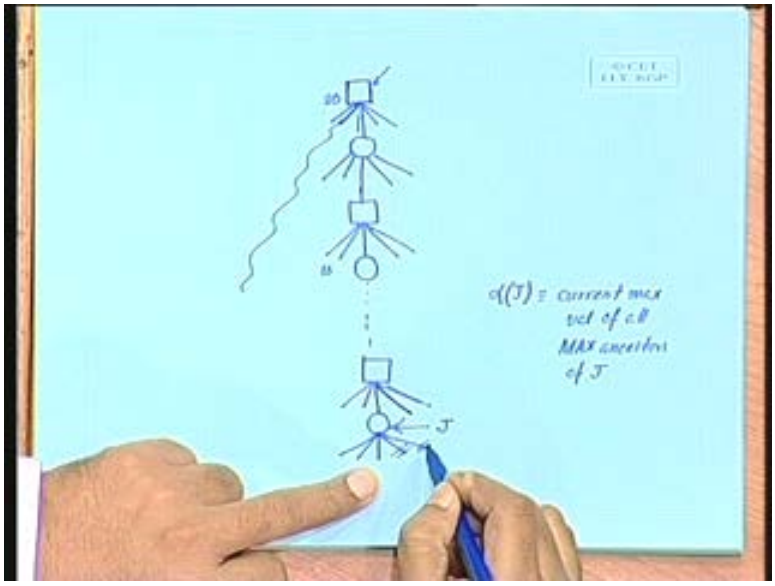
(Refer Slide Time: 00:35:00)



So, we will consider 2 bounds on the states of the game: one is the alpha bound. The alpha bound of j, is the maximum current value of all max ancestors of j, and we will stop the exploration of a min node, when its value falls, equals, or falls below alpha. So, let us look at a scenario here. I have this: so, there are other states, I am just looking at one path, okay? These are the other paths, which I have already visited, and these are the paths which I have not yet visited, right? So, this is just one path of the game tree, right? So, in this way, let us say that this is the min node j, right? Now, the alpha bound of j, or let us say alpha of j, is defined as the current maximum value of all max ancestors of j. So, it is the values that are backed up here. Suppose this has backed up something like 20,

and this has backed up something like 5; let us say, this has backed up 10, right? And so on.

(Refer Slide Time: 00:33:14)



So, it is the current maximum among all these. Now, whenever the value of this j- the current value of j- falls below this alpha, we will not explore the remaining successors of j anymore. Now, let us understand why. Consider that max ancestor, which has this value alpha j, right? So, in this case, let us say this is the value. Now, the moment this fellow's value falls below that value, we know that there is no point exploring this any further, because the game is never going to come to this state. Because the max node here, has a strategy of taking you to a node, which has at least 20- what do I mean by having a value 20, here?

It means that from this node, I have already found out some strategy of reaching cost of at least 20; if the opponent makes a mistake, probably more; if the opponent does not make a mistake, then at least 20. So, I have a strategy of going to a node which gives me at least 20. Now, here, I am finding that when I am visiting j, the cost has already dipped below 20, and if I explore further, it may dip further, or be at least, at most, 20. It is not going to exceed 20, because this is a min node. So, the node here is never going to try the strategy of coming to this node, so the strategies for visiting this node are of no interest from this layer, because there are better strategies already.

So, this move itself will not be selected, if it is the case that we have to end up here. If it is the case, that we have to end up here, then out here, it will not select this node, right? Okay. So, we do not explore this, anything this any further, but does it mean that we have already chosen this one? No, because along the other path from these max ancestors, we might still get some cost, which is larger than 20, right? So, we are just pruning off this min node; we are just pruning of the min node here, we are not doing anything with respect to the other max ancestors of this node. Is it clear? Right.

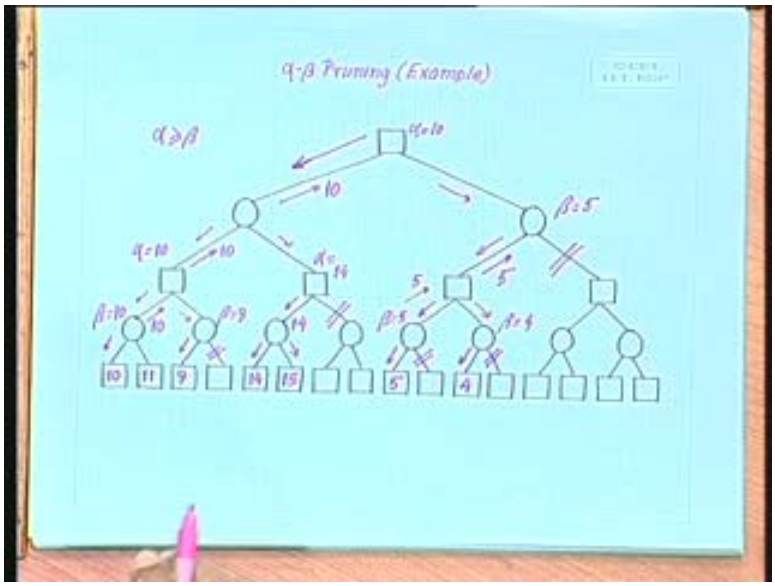
So, in the min nodes, what we are checking is, whether its current value has fallen below the value backed up in the max ancestor of the node. If it has, then we do not further explore that min node, we just backtrack to its previous max parent. And in a min node, we will update beta, because beta the beta bound of j, is the minimum current value of all min ancestors of j. And in a similar by a similar argument, we can say that exploration of a max node j is stopped, when its values equals or exceeds beta. In a max node, we update alpha, because if you see, that the value of beta is the minimum current value of all min ancestors of j. So, we will continuously keep on updating beta, when we are in the min nodes, and we will keep on updating alpha, as we are exploring a max node. Is this all right?

Now, before we go into the algorithm, okay, right? Now, when do we when do we turn when do we return when do we backtrack? In both min and max nodes, we will return when alpha is greater than or equal to beta. Now, see, this criterion is actually encompassing these criteria, that exploration of a min node is stopped when its value equals or fall bel falls below alpha. An exploration of a max is stopped when its value equals or exceeds beta, because you see the the beta value that we are maintaining here, is the minimum of the of the min node that that we are currently exploring, and all its min ancestors. Now, the very fact that we are still working with this node, means that the previous beta value is not greater than or equal to alpha.

So, previous beta value is still less than or equal to alpha; that is why we are still visiting this, right? And when this value the value of this min nodes falls below alpha, that is where the beta value will also fall below alpha, so that is where we will stop exploration. So, in this so this is the criterion, which covers both of this conditions, where in min or max nodes, we return when alpha is greater than or equal to beta, right? (Student speaking).Yes. No no no. See, we have gone a long way from that. We were talking about terminal nodes; when we were talking about small games, right, that is where we can go right down, up to the visiting. Now, we are talking about heuristic functions is the in the level where we are cutting off. Are you talking about them as terminal node? Are you referring to the ground level nodes, as the terminal node? Right. Yes.

So, when you are in the terminal nodes, that is where we will- that is where the induction basis will come. So, in the as you go down recursively, that is where you will reach the basis of the induction. You will start assigning your initial values, at that point. We will come to the algorithm, then, I will explain. Let us do one thing- let us work out the alpha beta pruning on one of the on a on an example. So, let us say that this is the- initially, we do not have this graph; initially, we do not have this tree. We are just developing it as we are going depth first. In alpha beta pruning, we are going depth first, and we are we will do the pruning as and when we require. so we are We will start from here, okay? Then, we will take we are going depth first, so we will go this way, generate this state in this way, and this way, and this way.

(Refer Slide Time: 47:05)



So, let us say that we are working with a depth look ahead of so many moves, right? When we have made so many moves, at this point, we will evaluate the board type, boards' position. At this position, let us say, we find that the board position is 10, right? Then, we will backtrack. Now, this is this answers your question. Then, we have reached the level of look aheads; we will backtrack, we will backtrack with the value of 10, right? Now, the beta value of this node is 10, right? The beta value here is 10. Then, I look at this state, and let us say I evaluate this, and find the level. Then, what am I going to back up? I am going- because this is a min node- it is going to back up 10, right?

the alpha value here So, here, the beta was 10; now, the alpha here becomes 10, right? Because alpha is the values that the max ancestors- that we are making. Then, we will go this way, right? Again, depth first, then we go this way. Depth first, and let us say, at this point, we find 9. Now, note that here, the beta value is now 9, and the beta value has fallen below the alpha value. So, we will prune here; we will not visit this side anymore, and why? Let us see why we will not visit here. This max node has already got 10, and in this min node, you have 9. And it may only dip further, right? So, we will not- there is no point exploring this anymore, because the max node is always going to go this way, because this is this is going to give you 9 or less, and this is already 10, so we will always take this this link, clear? So, therefore, this is pruned.

Then, let us continue. We go back here, what is this going to back up? It is going to backup 10, right? Now, our alpha is 10, beta is 10, right? Then, we go this way. Again, we will go depth first, so this way, this way, this way, and we find 14 here, right? So, now, the beta value here is 14, alpha is 10, so, we are still in business, right? We are still in business, because if we can the moment we will the when will we when will we be out of business? If we find that this fellow's cost also exceeds 10, because then, this node

here is never going to push us in this direction. It knows- if it pushes us in this direction, it will be trouble, right?

So, let us see, so, we will still traverse, we will take this one here, we find 15. Now, the moment we find 15 here, what do we have? We will have 14 here, and that means that out here, the alpha value is 14, right? Alpha is 14, beta is 10, so we have again reached the pruning criteria, and therefore, there is no point in checking this. And why? Because if you see, that this max node has already got 14, and this can only grow, right? And so, in this min node, we will never take this this way, because if we go this way, it is going to be at least 14. But if it goes this way, then we know that it is at most 10.

So, this fellow is trying to minimize, so he will always it will always push push us in this direction, right? Therefore, what what do we back up here? 10, right? Again, we have alpha equal to 10, beta equal to 10, right? Then, we go in this direction again- depth first, depth first, depth first, depth first, right? Here, we find 5, right. Now, the beta value here is going to be 5. Now, again, alpha is 10, beta is 5, so we have a pruning criterion, right? What is the pruning criterion? Whenever we find alpha is greater than or equal to beta- this is the pruning criteria.

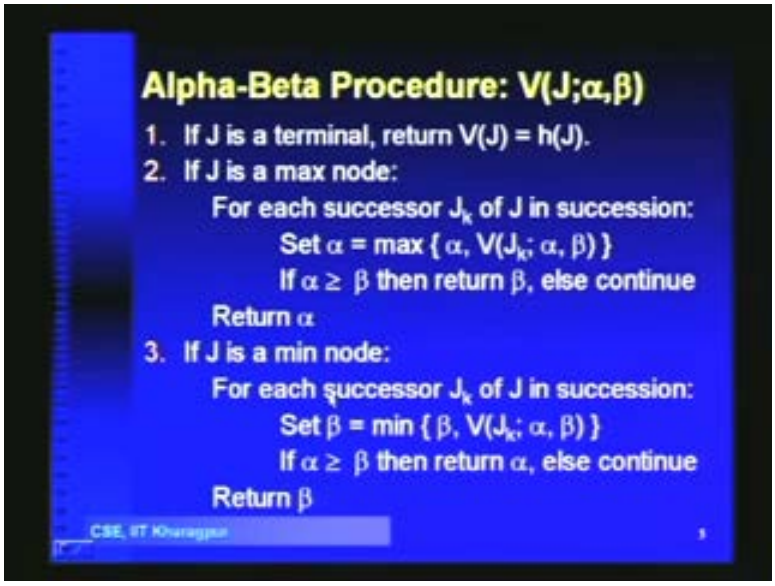
So, that has happened again. Alpha is 10, beta is 5, so, we prune this. And what is the reason of that? Because we know that if you come in this direction, it is not going to be for this node, because if you come here, you are going to get 5 or less, and this fellow already has a strategy of getting 10. So, if even if he chooses this link, it is not going to be because of this node. Therefore, there is no point visiting this node further, right? Whether this is 5 or 3 or one does not matter; all of them are equally bad, because we already have 10, okay? Then, what is this backing up from here? 5, right? Okay.

Alpha here is still 10, because a max ancestor here has 10, so alpha is still 10. And again, what do we have as the beta out here? 5, right? We still do not have 5 here; 5 was here, but we have already backtracked from there, so, it is only on the min ancestors, right? Okay. So, we will go in this direction, and we will come here. See? There is still a possibility that you get something more than 10 from this side; if you get something more than 10, then this still remains an attractive option. So, if you can get more than 10 here, more than 10 here then this link will become better. So, we come here and we get 4, right?

Again ,we have beta equal to 4, alpha is 10; pruning criterion applies- we prune here, clear? And then, we go back, so this is going to back up 5, right? If this backs up 5, then herein, we will have beta equal to 5. Again, alpha is 10, beta is 5, so we can prune out this whole part, because this gives us, that if you take this route, you cannot get more than 5, and if that happens, then this at in this max node, we will never consider this move, because if we consider this move, the opponent has a strategy of giving us 5 or less. But we already have a strategy of getting 10, so we will never choose this move. So, no point visiting this part at all.

So, this whole sub-tree gets pruned, and we finally come here with alpha equal to 10. And we will choose this one. The best move at this point is this, which is what we also found out by doing that whole bottom up stuff, but this alpha beta pruning helps us in getting rid of significant portions of the game tree. Right. Any questions? Okay. Now, let us have a look at the algorithm.

(Refer Slide Time: 00:49:44)



In this algorithm, if j is a terminal, then we return vj equal to hj. If we are at the at the leaf level, after the number of look aheads we have done, we written vj is equal to hj. vj is the cost that we will eventually return at the root. So, at a at a state j, vj is the best cost we can get, from the point of view of player a. So, all costs are with respect to player a. If j is a max node, then for each successor jk of j, in succession, we set alpha as the maximum of the existing alpha, which it has inherited from its parent, say this is a recursive procedure. So, the alpha that you are getting, is coming from your parent, right? And what is the definition of the alpha value? It is the maximum value that is among all the max nodes, including this current max node and its max ancestors, right?

So, it is the maximum of the existing alpha, and the value that I obtain by recursively calling v jk with alpha and beta bound, right? This alpha and beta bounds, because they are coming from the ancestors, so we can recursively call this procedures with the alpha and beta bound, being passed down, right? We are passing the alpha and beta bounds down into the recursion. And then, we apply the pruning criterion, that if alpha is greater than or equal to beta, then, we return beta, otherwise, we continue. And then, if we have finished with all the successors, then we have not yet pruned anything, so we have visited all the successors.

Then the best cost that I have in the max node is the alpha cost, so we return alpha, right, and we do exactly complementary stuff for the min node. So, for each successor jk of j, in succession, set beta equal to minimum of beta and v jk alpha beta. In the min nodes, we

are maintaining the value of beta, and again, we check if alpha is greater than or equal to beta, then beta and alpha, otherwise, continuing this loop, and if you are finished this loop with all successors, then return beta, right? So, what I would ask you to do is, you can either write a small piece of code to check this out on some game like tictactoe or any other game that you can think of, and also, to hand over this algorithm on one game tree, right, so that you have confidence in what is meant by this alpha beta pruning procedure, okay? Okay.

Now, with that, we come to the end of the lecture on game trees. You have any questions? (Student speaking). If the opponents make a mistake, you may go to those parts which are pruned, but in those cases, you will definitely get a profit more than what you normally get. No, so you will again- see, that is the next move. See, what are we doing this analysis for? To determine what move we will make now, right? No, no, no, we will not use those paths, because we will assume that the opponent is intelligent; the opponent will not make mistakes, right?

We are taking into consideration the worst case scenario with respect to the opponent. We will choose our move based on that, right? Now, if the opponent makes the wrong move, or the move which I have not expected, then what we can do is, we can we will again be in a max node after 2 levels of move. From there, we will again expand out the game tree and do the analysis again, right? And then, determine which would be our best move there, but what we are assured is that, the best move that we get from there, is going to be at least as good as what I originally had planned for. That is because the opponent has made a mistake, so my min max value can only be more than what I have obtained here, isn't it? (Student speaking).

No, in the previous case means, when we were not pruning, yes. So, in that case, what we were doing is, we were looking ahead right up to the leaf level, which is the number of lookaheads that we are doing. (Student speaking).Yes. No, no, no. I think I must clarify that- this is the current decision that I am taking. If the opponent puts me here, I will again do that many more look aheads, right? See, why I am restricting the number of look aheads- because of the computational complexity involved, because the state space grows very fast. Particularly in games like chess, the branching is very large, so the state space will really blow up. If you look, in- 15, 16 moves; I mean, beyond that it is- even 15, 16 moves is phenomenal, right?

So, you cannot really afford to go much further below; that is why we are looking look ahead that many number of times. But when I have already played a couple of moves- I have taken my move, the opponent has taken the move- I will again do an analysis of them. Now, if you look at the expert level at which you are playing, with the chess playing program, that actually, determines the number of moves, lookaheads, that it is trying. So, if you look at a greater expert level, then it is using more knowledge, and it is using more look aheads, right? It will take more time also. (Student speaking). So, in tictactoe, we do not normally do that, because it is a small game, so you can actually expand it right up to the winning or losing configurations, right?

But in chess, that is a million dollar question- that is, determining the heuristic function is the is the key factor here. So, they do a lot of things- they keep history of different games and find out, that from this position, what is the how many winning scenarios were there? They try to follow also, some of those existing patterns, or they can actually evaluate. Then, say, a simple thing could be, the number of pieces that you have, right? The number of pieces that you have in your board, is one of the issues, right? Actually, what happens is, there this is not a single objective scenario. in a In a chess playing scenario, there can be multiple criterion, which determine the tactical and positional advantage that you have, right? So, there is some work on that; actually, some of my own work is on that- on how to extend multi-criteria searching algorithms to game playing situations. So, I can give you the citation, if you want to take a look later on.