Artificial Intelligence

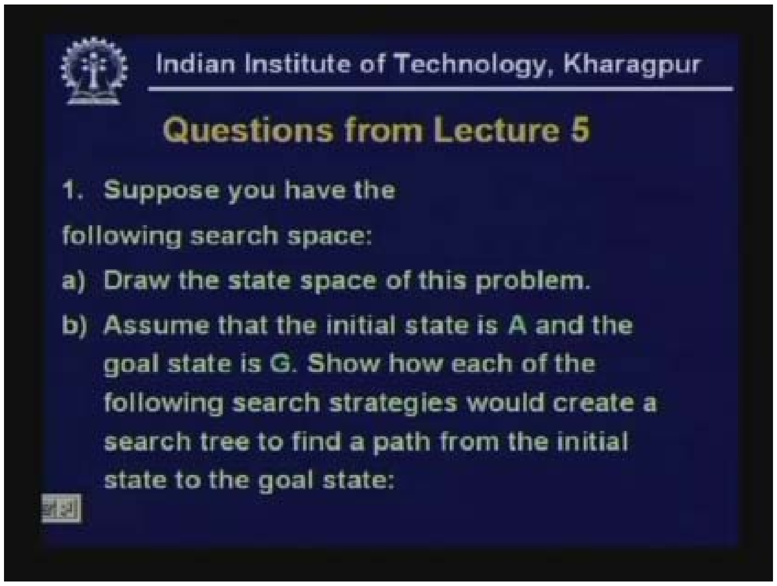
Prof. Sudeshna Sarkar
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 7
Two Player Games - 1

Today we have the seventh lecture of the course Artificial Intelligence. In today's lecture we will talk about two player search. In the last few lectures we have looked at different state space formulations of some types of problems which were formulated as search problems. But in all these cases we assumed that there is one agent or one machine who is trying to reach the goal from the start position.

In today's class we will consider situations where there is more than one agent. In particular we will talk about two agent situations. In fact a specialized two agent situations where there are two competing players so that the gain of one player is the loss of the other player. So we will see that such problems can be formulated as different kinds of search problems which have different characteristics than the single agent search problems we considered earlier. But before we go on to today's lecture I would like to discuss the questions of the previous two lectures.

(Refer Slide Time 02:29)


Indian Institute of Technology, Kharagpur

Questions from Lecture 5

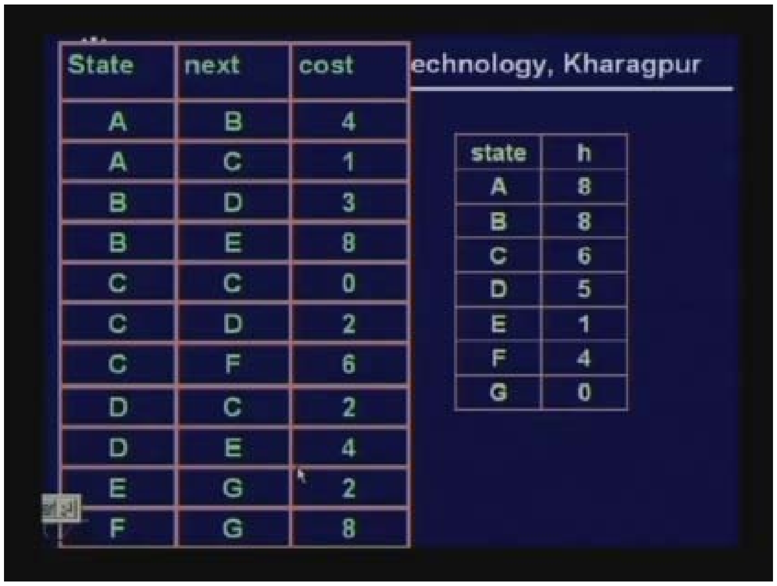
1. Suppose you have the following search space:

a) Draw the state space of this problem.

b) Assume that the initial state is A and the goal state is G. Show how each of the following search strategies would create a search tree to find a path from the initial state to the goal state:

First let me come to the questions from lecture 5.
Question number one:
Suppose you have the following search space this search space will be given to you in the form of a table you were supposed to draw the state space of this problem, assume that the initial state is A and the goal state is G and you have to work out a few search strategies and show the search tree and what sort of path the search generates from the initial path to the goal.

(Refer Slide Time 03:08)



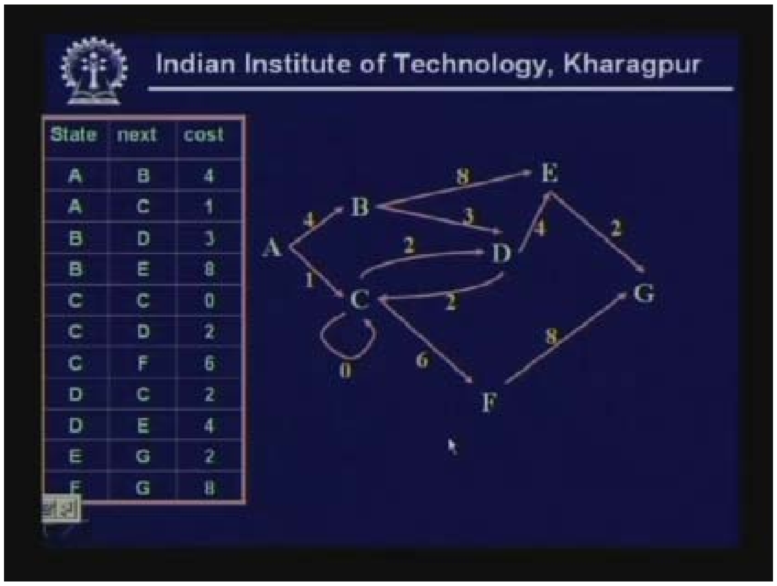
echnology, Kharagpur

State	next	cost
A	B	4
A	C	1
B	D	3
B	E	8
C	C	0
C	D	2
C	F	6
D	C	2
D	E	4
E	G	2
F	G	8

state	h
A	8
B	8
C	6
D	5
E	1
F	4
G	0

This table gives you a representation of the state transition of the system. There are states A B C D E F G and these are the edges AB, AC, BD, BE, CC, CD, CF, DC, DE, EG and FG and in this column we have the costs of the different edges. In addition, for heuristic search we will need to have an estimate of the cost at a particular state. So the estimate at A is equal to 8 at B is equal to 8 at C is equal to 6, at D is equal to 5 at E is equal to 1 at F is equal to 4 and at G which is goal state the estimate is 0.

(Refer Slide Time 04:07)



Indian Institute of Technology, Kharagpur

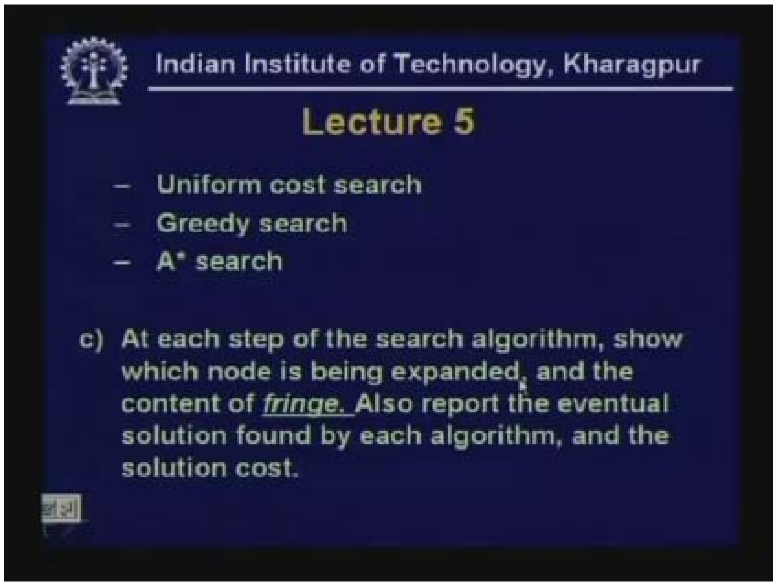
State	next	cost
A	B	4
A	C	1
B	D	3
B	E	8
C	C	0
C	D	2
C	F	6
D	C	2
D	E	4
E	G	2
F	G	8

$$\begin{tikzcd}
 & B \arrow[rr, "8"] \arrow[r, "3"] & D \arrow[r, "4"] \arrow[dl, bend left, "2"] & E \arrow[dr, "2"] & \\
A \arrow[ur, "4"] \arrow[r, "1"] & C \arrow[ur, bend left, "2"] \arrow[loop below, "0"] \arrow[rr, "6"] & & F \arrow[r, "8"] & G
\end{tikzcd}$$

Now given this state transition diagram we have to work out three search strategies:
1) Uniform cost search

2) Greedy search and
3) A star search.

(Refer Slide Time 04:28)



Indian Institute of Technology, Kharagpur

Lecture 5

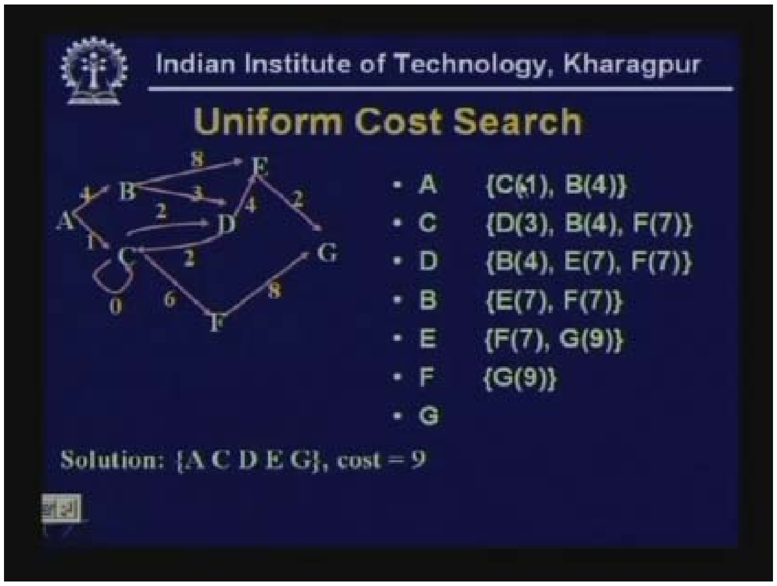
- Uniform cost search
- Greedy search
- A* search

c) At each step of the search algorithm, show which node is being expanded, and the content of *fringe.* Also report the eventual solution found by each algorithm, and the solution cost.

And you have to show at each step of the search algorithm the node which is being expanded and the content of fringe. Also, you have to find out the eventual solution discovered by the specific search algorithm and the solution cost that is achieved. Now let us go back to this table representing the transition. So A to B we have an edge with a cost of 4. This is a graphical representation of the same table. A to B there is an edge with cost 4; A to C there is an edge with cost 1, B to D has cost 3, B to E has cost 8, C to C has cost 0, C to D has cost 2, then C to F has cost 6, D to C has cost 2, D to E has cost 4, E to G has 2 and F to G has cost 8. This is a representation of the state space for this problem. Now let us quickly look at how uniform cost search works on this problem. In uniform cost search we start from the initial state and we expand nodes in the order of their distance from the initial state. So, from the state A we can reach the nodes C and B.

Initially we will put the node A in the fringe or in the open list. This will be the initial content of fringe. At the next instance we will remove this node from the fringe and expand it. So A will be the first node that gets expanded and as a result we will get its two children B and C which we will put in fringe. B has a cost of 4 and c has a cost of 1 so C and B will be put in fringe, C with a cost of 1 and B with a cost of 4.

(Refer Slide Time 06:09)



Uniform Cost Search

• A	{C(1), B(4)}
• C	{D(3), B(4), F(7)}
• D	{B(4), E(7), F(7)}
• B	{E(7), F(7)}
• E	{F(7), G(9)}
• F	{G(9)}
• G	

Solution: {A C D E G}, cost = 9

Now at the next step we remove the lowest cost node from fringe which happens to be C. So we remove C from fringe expand it and then find the children of C which happened to be D, F and C itself. Now C has already been expanded so if we keep track of it then we will not put C in the list instead we will generate D and F. D will have a cost of 1 plus 2 is equal to 3 and F will have a cost of 7. So we will have D with a cost of 3, B with a cost of 4 and F with a cost of 7. Next we will again remove the lowest cost node and expand it which happens to be D. After that we will find the children of D which happen to be C and E out of which C is in the closed list if we do maintain a closed list but if we do not we will put it in there.

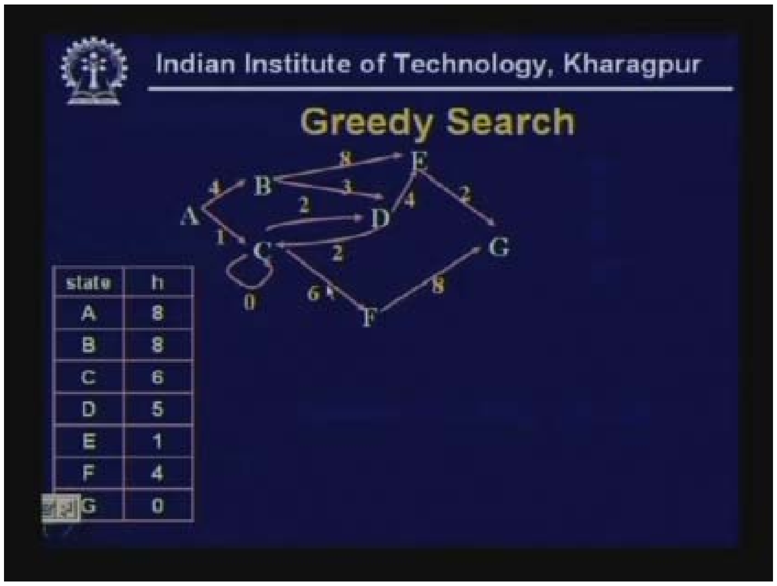
Let me not put in C right now. Let me put in E so what we get is B, E and F.
D has a cost of 3 plus 4 is equal to 7, so E will have a cost of 7. Then B is already there with a cost of 4 and F is there with a cost of 7. Now out of these three nodes which are in the fringe at this instant we will remove B which is the smallest cost. And then we will find the children which happen to be E and D out of which E is already there in the fringe and the current cost of E through B would be 4 plus 8 is equal to 12 and E is already there in fringe with a cost of 7. So E will not go into fringe but D will and D will have a cost of 4 plus 3 is equal to 7. So, in the fringe we will have D with a cost of 7, E with a cost of 7 and F with a cost of 7. Now we have to remove the lowest cost node in fact all these three nodes have the same cost and we can remove any one of them.

Let us say that we remove D but D has already been expanded so we will not put D and then we will remove E. If we remove E we will have F left in fringe and E has the child G. Now G will have a cost of 9 and F is there in fringe with a cost of 7. Next we will expand F and F has no new child. F has another child G which will have a cost of 15 so we will not put it in here instead we will keep G with original cost of nine and the next step G is the only node in fringe so we will expand it and G happens to be the goal so we are done.

Therefore in this search we have expanded the following nodes A C D B E F G. Then if we broke this tie in other way rather if we had expanded F first then we might have got some other sequence of nodes. But once we get the goal we can retrace and find the path. So it happens that in this case G will have the parent as E, E will have its parent as D, D will have its parent as C and C will have its parent as A. So the path found would be A C D E and this path has a cost of 9 so uniform cost search has found an optimum solution and this optimum solution has a cost of 9.

Let us now go and see what happens if we use greedy search for this problem. In greedy search we start from A and find the node which is closest to A or rather we find the node which is reachable from A and whose H value is at minimum.

(Refer Slide Time 12:06)


Indian Institute of Technology, Kharagpur

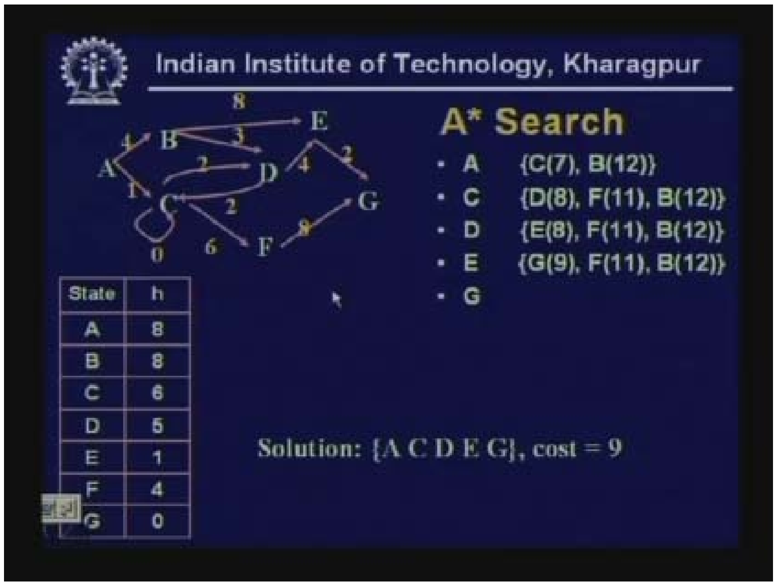
Greedy Search

state	h
A	8
B	8
C	6
D	5
E	1
F	4
G	0

Now from A there are two nodes which are reachable B and C. B has A edge value of 8 and C has a edge value of 6 so what we will do is that we will take C because C has a lower H value. Now from C the two successors are D and F. D has an H value of 5 and F has a H value of 4 so we expand F so we go to F. Now F has one successor which is G which has H value of 0 so G is expanding so we get a path A C F G which is the solution found by greedy search in this case.

What is the cost of the solution?
It is 1 plus 6 is equal to 7 plus 8 is equal to 15 so cost is 15. Obviously in this case the greedy algorithm does not give you the optimum cost path. Next we will look at A star search as to how A star works on the same graph. Now in A star first the fringe contains the single node A then A is expanded and its children C and D are generated.

(Refer Slide Time 13:41)



A* Search

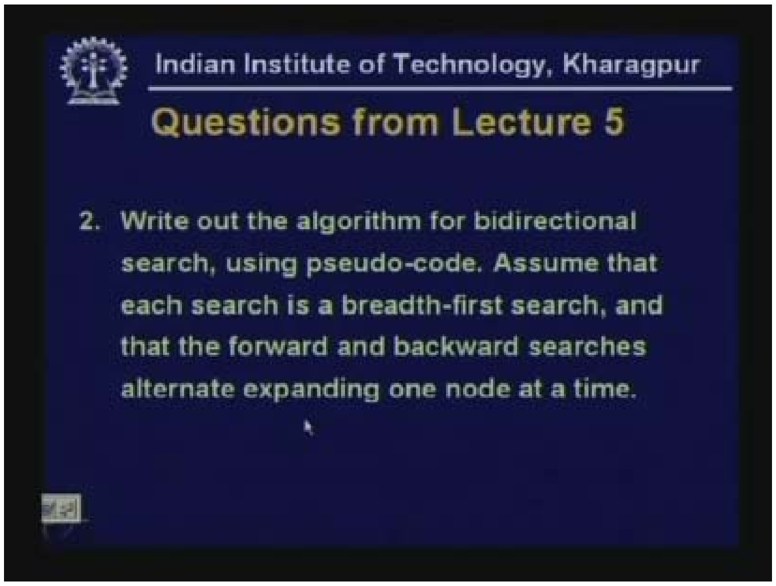
- A {C(7), B(12)}
- C {D(8), F(11), B(12)}
- D {E(8), F(11), B(12)}
- E {G(9), F(11), B(12)}
- G

State	h
A	8
B	8
C	6
D	5
E	1
F	4
G	0

Solution: {A C D E G}, cost = 9

If you remember in A star the F value of a node N is equal to its G value that is its cost from the initial state plus the estimate of its cost to the goal. So, for the node C the G value is 1 here and h value is 6 so C has a cost of 7, B has a G value of 4 and H value of 8 so B's cost is 12. Now C is expanded removed from open and its children are C, D and F and C has already been closed so D and F are put in open whose cost F value happens to be 8 and 11. Next D is removed from open and its children are inserted. Then E is removed then finally we get G and as a result we get the solution path A C D E G whose cost is 9 and the nodes expanded are only A C D E and G which happens to be lesser than the number of nodes we expanded in the case of uniform cost search on the same graph. Now let us look at the second question.

(Refer Slide Time 15:10)



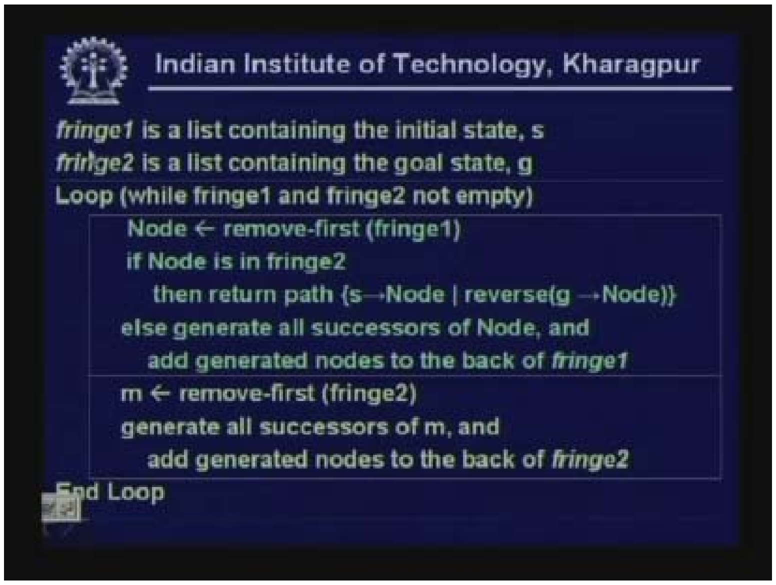
Indian Institute of Technology, Kharagpur

Questions from Lecture 5

2. Write out the algorithm for bidirectional search, using pseudo-code. Assume that each search is a breadth-first search, and that the forward and backward searches alternate expanding one node at a time.

In the second question you were asked to write the algorithm for bidirectional search using pseudo code. Assume that each search is a breadth first search and that the forward and backward searches alternate expanding one node at a time.

(Refer Slide Time 15:29)

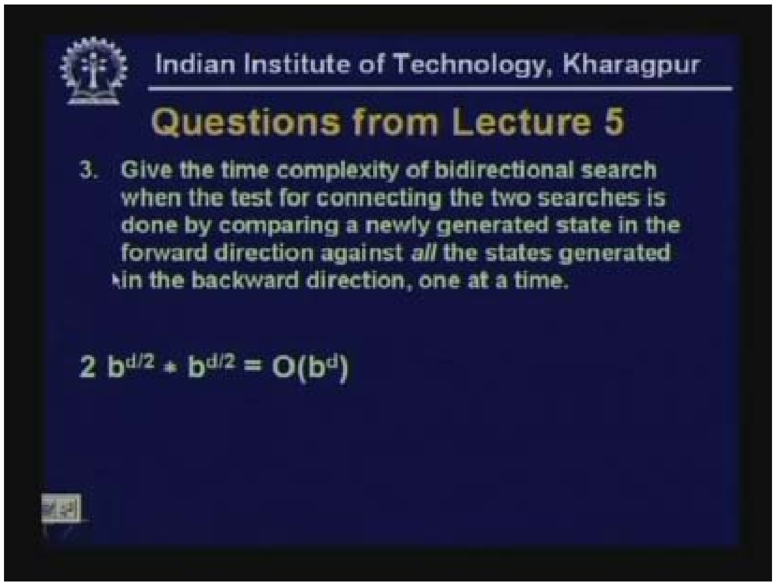


Indian Institute of Technology, Kharagpur

```
fringe1 is a list containing the initial state, s
fringe2 is a list containing the goal state, g
Loop (while fringe1 and fringe2 not empty)
    Node ← remove-first (fringe1)
    if Node is in fringe2
        then return path {s→Node | reverse(g →Node)}
    else generate all successors of Node, and
        add generated nodes to the back of fringe1
    m ← remove-first (fringe2)
    generate all successors of m, and
        add generated nodes to the back of fringe2
End Loop
```

This is a pseudo code. Fringe: We maintain two open lists fringe1 and fringe2. Fringe1 contains initially the start state. It represents the search tree as we move forward from the start state and fringe2 is the open list of the search tree which we get while going backward from the goal. So while fringe1 and fringe2 are not empty both are non empty we first remove the first node from fringe1 let it be node. So if node is in fringe2 then we return the path that we get from start to node concatenated with the path we get from goal to node reversed. Otherwise if node is not in fringe2 we generate all successors of node and add the generated nodes to the back of fringe1. If we are done we go to fringe2, we remove the first node of fringe2 generate all successors of m and add the generated nodes to the back of fringe2. So this is a pseudo code for bidirectional search.

(Refer Slide Time 16:49)



Indian Institute of Technology, Kharagpur

Questions from Lecture 5

3. Give the time complexity of bidirectional search when the test for connecting the two searches is done by comparing a newly generated state in the forward direction against *all* the states generated in the backward direction, one at a time.

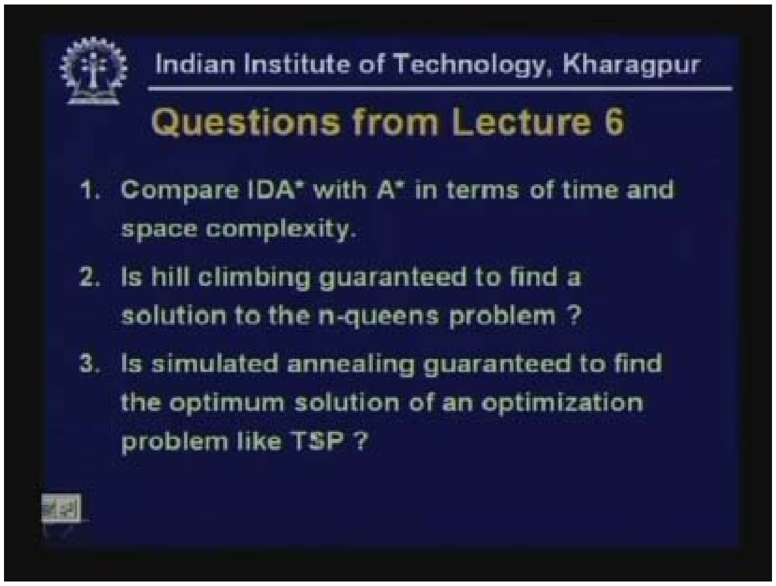
$2\ b^{d/2} * b^{d/2} = O(b^d)$

Third question, give the time complexity of bidirectional search when the test for connecting the two searches is done by comparing a newly generated state in the forward direction against all the states generated in the backward direction one at a time. So, in bidirectional search we have the start state here, we have the goal state here, we have a forward search from the start state, we have a backward search from the goal state until they meet. And when they meet we get a path from start to this node N and from goal to this node N and we concatenate them to get the path.

Now if the tree has a branching factor of b and if these two search tree generations alternate these trees will meet about half way. So the depth of each of these trees would be d by 2. So the number of nodes in this tree is b power d by 2 and the number of nodes in this tree is b power d by 2 so total two times b power d by 2 is expanded. But every time we expand a node in the forward tree we check it against all the nodes in the backward tree and the number of nodes in the fringe of the backward tree is of the order of b power d by 2.

Therefore, for every node expanded in the forward direction we spend order of b power d by 2 time checking it against the fringe of the backward tree. So the total time taken is order of b power d. But bidirectional search can be made faster if we do not check every node in this fringe but use some hashing scheme so that this test can be performed quickly. In that case we may be able to get a performance which is nearer to order b power d by 2 which is square root of order b power d by 2 for plain breadth first search. Now we move onto questions from lecture 6.

(Refer Slide Time 19:12)



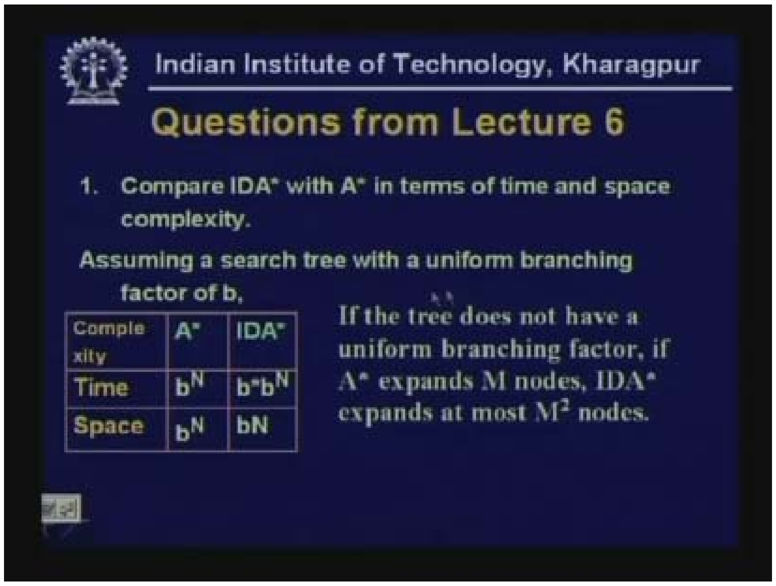
Indian Institute of Technology, Kharagpur

Questions from Lecture 6

1. Compare IDA* with A* in terms of time and space complexity.
2. Is hill climbing guaranteed to find a solution to the n-queens problem ?
3. Is simulated annealing guaranteed to find the optimum solution of an optimization problem like TSP ?

1) Compare IDA star with A star in terms of time and space complexity.

(Refer Slide Time 19:20)



Indian Institute of Technology, Kharagpur

Questions from Lecture 6

1. Compare IDA* with A* in terms of time and space complexity.

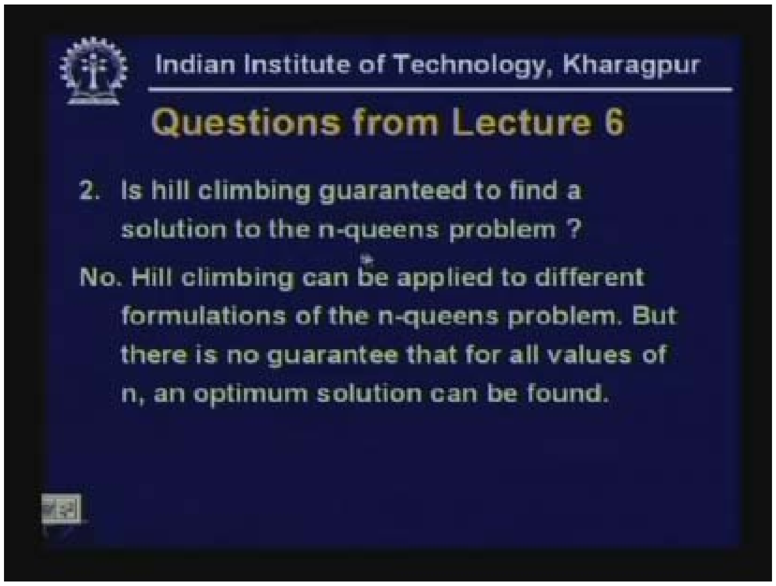
Assuming a search tree with a uniform branching factor of b,

Complexity	A*	IDA*
Time	b^N	$b*b^N$
Space	b^N	bN

If the tree does not have a uniform branching factor, if A* expands M nodes, IDA* expands at most M^2 nodes.

So let us assume a search tree with a uniform branching factor of b. In this case the time complexity of A star is b power n of IDA star is b into b power n. The space complexity of A star is b power n of IDA star is only b into n. This is linear. However, if the tree does not have a uniform branching factor this may not be true but we can see that if A star expands m nodes IDA star expands utmost m^2 nodes. This happens if at every new iteration exactly one new node is expanded.

(Refer Slide Time 20:03)



Indian Institute of Technology, Kharagpur

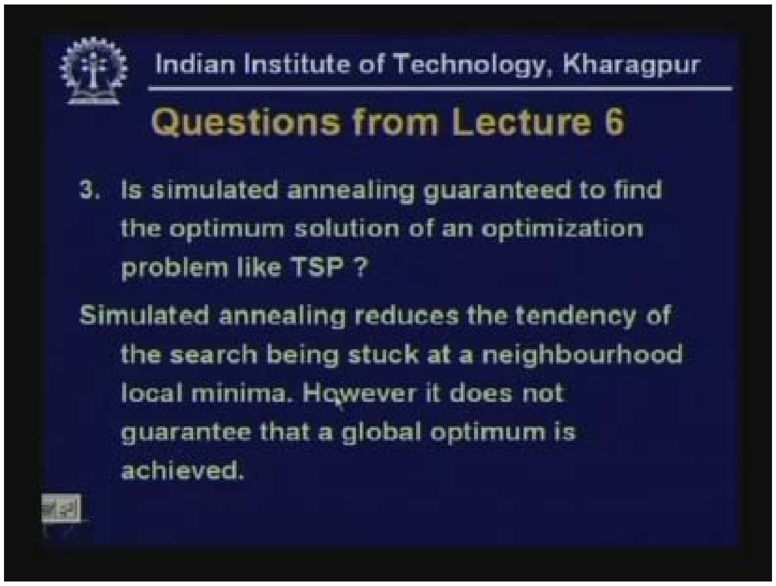
Questions from Lecture 6

2. Is hill climbing guaranteed to find a solution to the n-queens problem ?

No. Hill climbing can be applied to different formulations of the n-queens problem. But there is no guarantee that for all values of n, an optimum solution can be found.

Question number 2; Is hill climbing guaranteed to find a solution to the n queens problem?
No, hill climbing can be applied to different formulations of the n queens problem but there is no guarantee that for all value of n an optimum solution can be found.

(Refer Slide Time 20:19)



Indian Institute of Technology, Kharagpur

Questions from Lecture 6

3. Is simulated annealing guaranteed to find the optimum solution of an optimization problem like TSP ?

Simulated annealing reduces the tendency of the search being stuck at a neighbourhood local minima. However it does not guarantee that a global optimum is achieved.

3) Is simulated annealing guaranteed to find the optimum solution of an optimization problem like traveling salesperson problem?
No, simulated annealing reduces the tendency of the search being stuck at a neighborhood local minima. However it does not guarantee that a global optimum is achieved. Now we will move onto today's lecture and today's lecture as I have said earlier is on two player search.

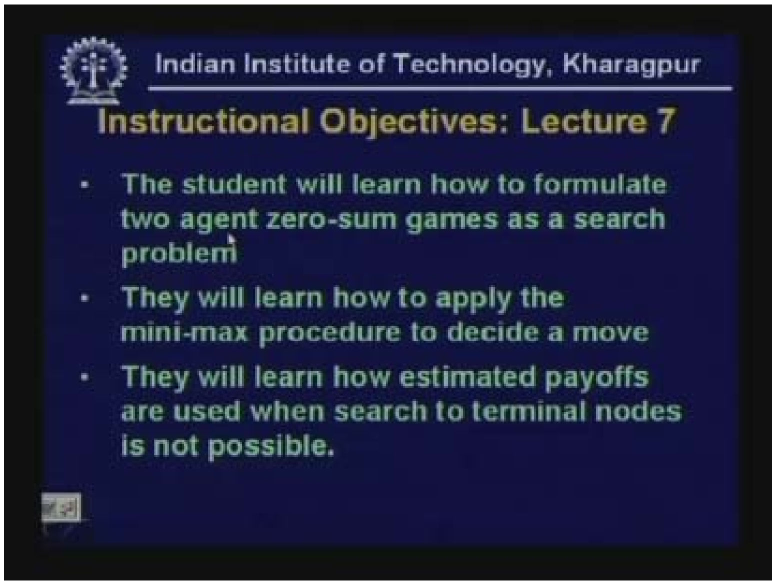
(Refer Slide Time 20:46)

Indian Institute of Technology, Kharagpur

Artificial Intelligence

This is module 2 problem solving using search and lecture 7 is two player search part 1.

(Refer Slide Time 21:00)

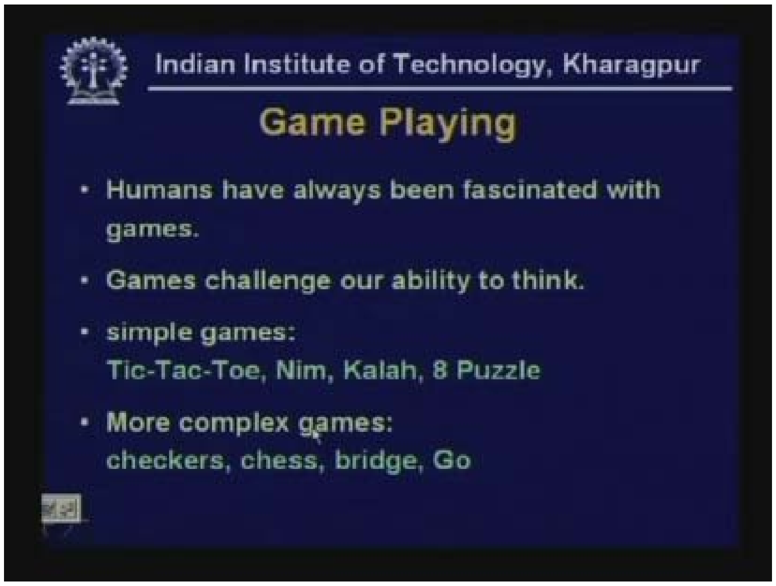

Indian Institute of Technology, Kharagpur

Instructional Objectives: Lecture 7

- The student will learn how to formulate two agent zero-sum games as a search problem
- They will learn how to apply the mini-max procedure to decide a move
- They will learn how estimated payoffs are used when search to terminal nodes is not possible.

The objectives of today's lecture:
The student will learn how to formulate two agents zero sum games as a search problem. They will learn how to apply the mini max procedure to decide a move. They will learn how estimated pay offs are used when search to terminal nodes is not possible.

(Refer Slide Time 21:25)



Indian Institute of Technology, Kharagpur

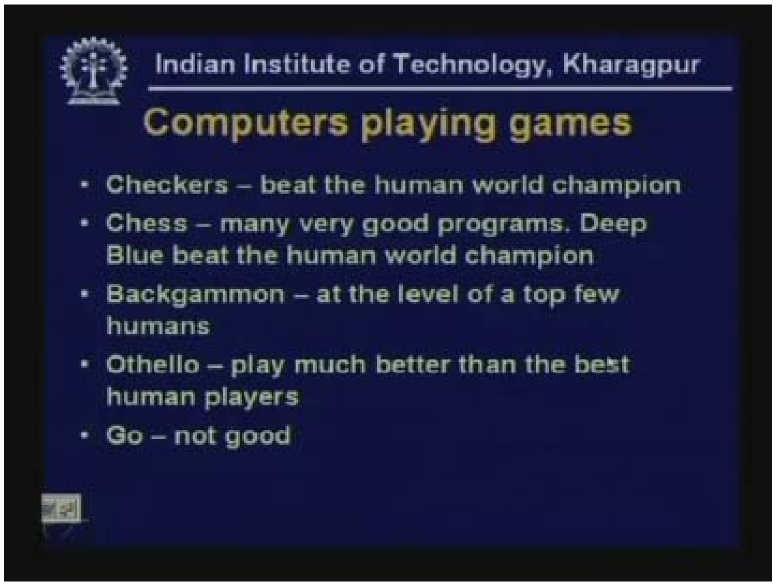
Game Playing

- Humans have always been fascinated with games.
- Games challenge our ability to think.
- simple games:
  Tic-Tac-Toe, Nim, Kalah, 8 Puzzle
- More complex games:
  checkers, chess, bridge, Go

Humans have always been fascinated with games. Games challenge our ability to think. So people have been engaged in many different types of games. There are simple games that we play like Tic Tac Toe and Knots and Crosses. Then there are other simple games like Nim, Kalah and 8 puzzle and there are more difficult or more complex games like Chess, Othello, Bridge, Go, Checkers etc but Othello is actually not that complex.

Now computers play games. Since the beginning of the history of computers or Artificial Intelligence people have been fascinated by the idea of making computers play games. So if you recollect we discussed that Alan Turing conceived that computers would play chess.

(Refer Slide Time 22:00)



Indian Institute of Technology, Kharagpur

Computers playing games

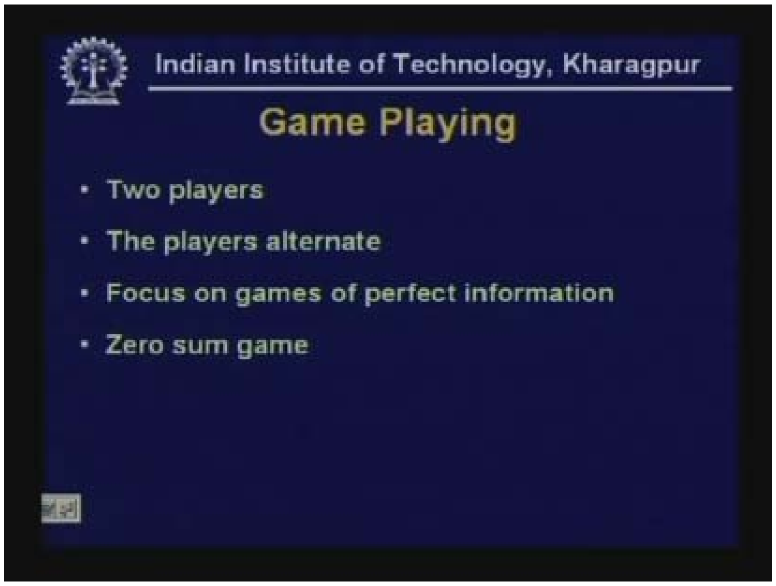
- Checkers – beat the human world champion
- Chess – many very good programs. Deep Blue beat the human world champion
- Backgammon – at the level of a top few humans
- Othello – play much better than the best human players
- Go – not good

In fact the first machine to play chess or first machine that claims to play chess going back many years. The machine was called the Turk. Actually it was a contraption which claimed to play chess. Even though actually there was no machine like intelligence involved in it because this contraption had a human hidden inside the interior who would play the moves. Since then we have come a long way and today the chess program in 1997 the deep blue chess program developed by IBM succeeded in defeating the world champion Gary Kasparov in a 6 game match. There are other games for which computer programs have been written like checkers. Checkers programs have been able to beat the human world champion.

Backgammon: Backgammon like TD Gammon by Tesauro has been written. And backgammon is actually a game of chance involves the throw of dice but the TD Gammon competes with the top few human players. It is very good but there are human players who are also as good as it.

In Othello the computer program plays much better than the best human players. Then there is a game of GO where unfortunately computer programs do not do so well so GO has a very high branching factor and computer programs have not been successful in GO so human players are much better. So we will consider a particular type of game involving two players where the two players alternate they compete and both the players have perfect information of each other, there is no dice throw or any chance factor involved. And these are fully competitive games called zero sum games where the pay off of one player is exactly the negative of the pay off of the other player. They have exactly opposing interests and that is why they are called zero sum games.

(Refer Slide Time 24:22)



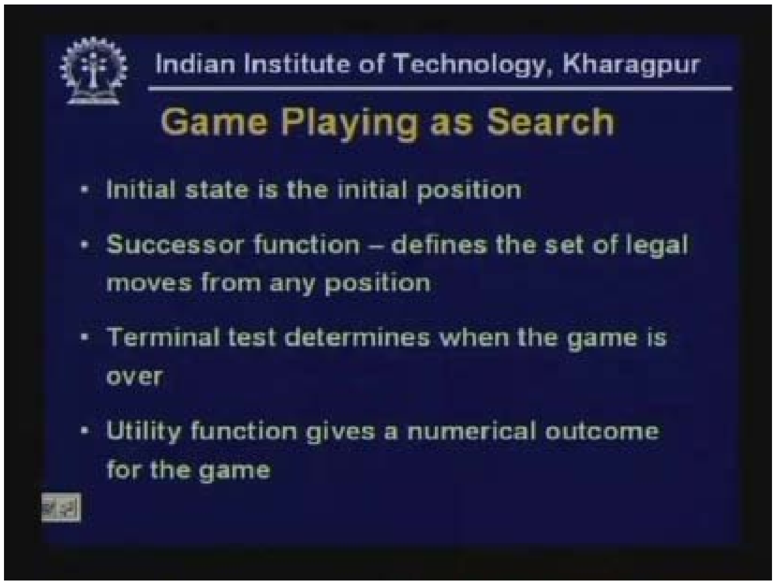
Indian Institute of Technology, Kharagpur

Game Playing

- Two players
- The players alternate
- Focus on games of perfect information
- Zero sum game

For example, this is like the game of chess where if there are two players a and b; if a wins b loses and if b wins a loses or a and b both have a draw. Now let us see that game playing can be formulated as a search problem.

(Refer Slide Time 25:11)



Indian Institute of Technology, Kharagpur

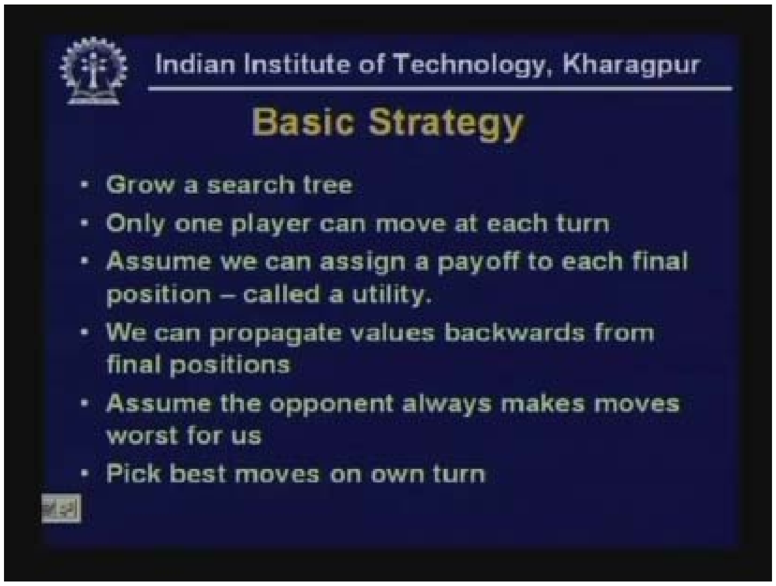
Game Playing as Search

- Initial state is the initial position
- Successor function – defines the set of legal moves from any position
- Terminal test determines when the game is over
- Utility function gives a numerical outcome for the game

We can take the initial position of the board. For example, in chess the initial board can be taken as the start position and the goal position is the position where one of the kings has been captured. If you take the game of Tic Tac Toe the initial position is a 3 by 3 board with no cross or not in it. And the final position is a position where either there are three crosses or three Os in a row in a column or the diagonal or all the positions are occupied. The successor function in a game tree defines the set of legal moves from any position. Like in chess board if white is playing white has a number of possible moves. So these are the possible successor positions for white. The terminal test determines when the game is over. In chess the terminal test means the king of one of the sides is captured. When a game is over the utility function gives an outcome of the game.

For example, in chess if white and black are playing if white wins we give white a utility of plus 1 and if white loses we give white a utility of minus 1 and if white draws with black we give white a utility of 0. So the basic strategy in game playing is to grow a search tree.

(Refer Slide Time 26:48)


Indian Institute of Technology, Kharagpur

Basic Strategy

- Grow a search tree
- Only one player can move at each turn
- Assume we can assign a payoff to each final position – called a utility.
- We can propagate values backwards from final positions
- Assume the opponent always makes moves worst for us
- Pick best moves on own turn

In search trees for games only one player can move at each door. We have a tree starting from the initial position to the successor positions and so on until we reach the goal position. At this level it is player one‘s turn to move. At the next instance it will be player b’s turn to move then again player one’s turn to move and so on. At the leaf position when the game is finished we can assign utility to the player.

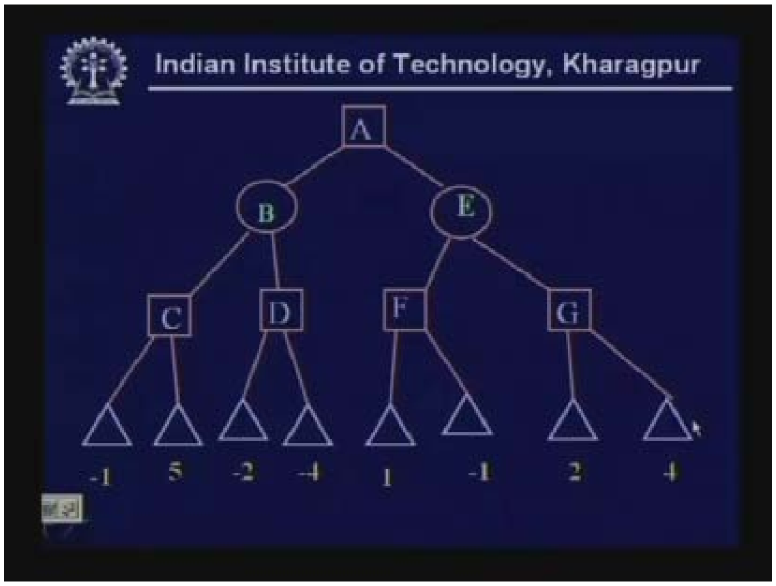
For example, suppose it is a simple game involving just win, draw and loss we can give a utility of 1 to the winner, minus 1 to the loser and 0 to both the players if there is a draw. In games like backgammon there is concept of gammon and the backgammon. So a normal win gets a value of plus 1, a gammon may get a value of plus 2 and the backgammon may get a value of plus 4. So there are games where the utilities are more.

Therefore, in general we can have a numerical value of the utility. Once we have the search tree we know the utility of the leaf positions and we can assign utility to the leaf positions of the game tree. And we can propagate these values backwards to compute the utility of other nodes which are in the intermediate nodes of the tree or at the root of the tree. The difference between single agent search and two agent search is that in single agent search there is only one player who moves at every step. So we can plan the entire path for the player, we can find the entire path to the goal state.

However, in two agent games if I am one player I can only make one move and then I have to wait for the move of the other player. Only after the other player moves then only I can plan my next move. So I cannot plan the entire sequence of moves at a time. I can only plan one move at a time. And then depending on the move the opponent makes we make our next move. Now we do not have any control over the move the opponent makes. But for the sake of analyzing the worst case performance of these algorithms we will assume that the opponent makes the move which is the best for the opponent. And because it is a zero sum game it is that move which puts us in the worst position. We will pick the best moves on our own turn and assume that in the opponent turn the opponent picks the worst move for us. This is a very simple example of a game tree.

A is the initial position, it has two possible moves to B and E, from B the opponent can make a move to C or D, from E the opponent can make a move from F to G. And from C we can make a move to this position or this position, from D we can move here or here, F we can move from here or here, G we can go from here or here. These are the terminal positions. In the terminal positions we have the following utility. Here we have a utility of minus 1, here we have a utility of 5, minus 2, minus 4, 1 minus 1, 2, 4.

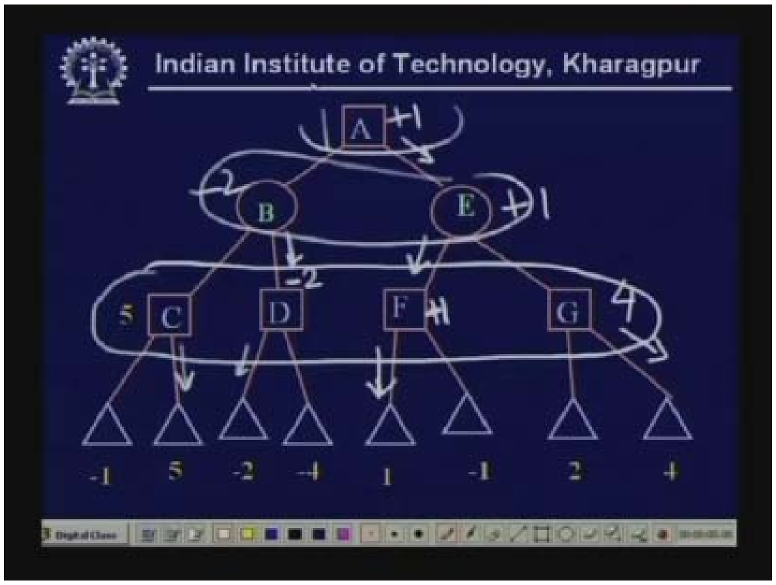
(Refer Slide Time 31:16)



Since this is our turn to move and at C we have two choices minus 1 or 5 we seek to maximize our utility. So we will choose the move to this node whose utility is 5. So the backed up utility value at C is 5. Similarly, at d we have two choices to move to minus 2 and minus 4 of which minus 2 is better for us so the backed up utility of t will be taken to be minus 2. At F we have two choices and one is better a choice than minus 1 so the backed up utility at F will be plus 1 and we will move in this direction. At G we have a choice of 2 and 4, 4 is better for us so G will have a backed up utility of plus 4. Now here at B and E it is the opponents chance to move. So the opponent has a choice of going to 5 or minus 2. The opponent will choose something which is worst for us. If the utility is 5 for us it means that from the opponents' point of view it is minus 5. And at D from opponents' point of view it is plus 2 so opponent will choose this and the backed up utility at this node will be minus 2. At E the opponent has a choice of going to plus 1 or 4. It will choose this which is the worst for us and the backed up utility is plus 1. At A which is our turn to move we have a choice of getting utility of minus 2 or plus 1 and we will choose this because we get a better value up here. Therefore the backed up utility will be plus 1.

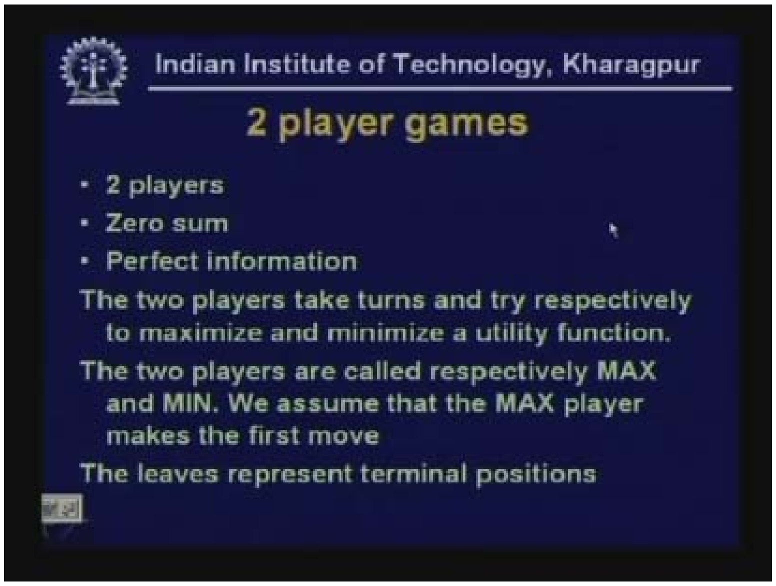
Thus, we are assuming while backing up the utility values that at every step we make the best moves and the opponent makes the worst move for us. And we guarantee that if we go along this path the minimum pay off t we can get is plus 1. To compute these utility values we are alternating minimization and maximization. So here we are doing minimization at this layer where we play and we are doing minimization at this layer where the opponent plays again maximization at this layer where we play.

(Refer Slide Time 34:07)



This sort of backing up is called a mini max algorithm. This is the algorithm in action. These are the backed up values and A has a backed up value of plus 1.

(Refer Slide Time 34:25)



Indian Institute of Technology, Kharagpur

2 player games

- 2 players
- Zero sum
- Perfect information

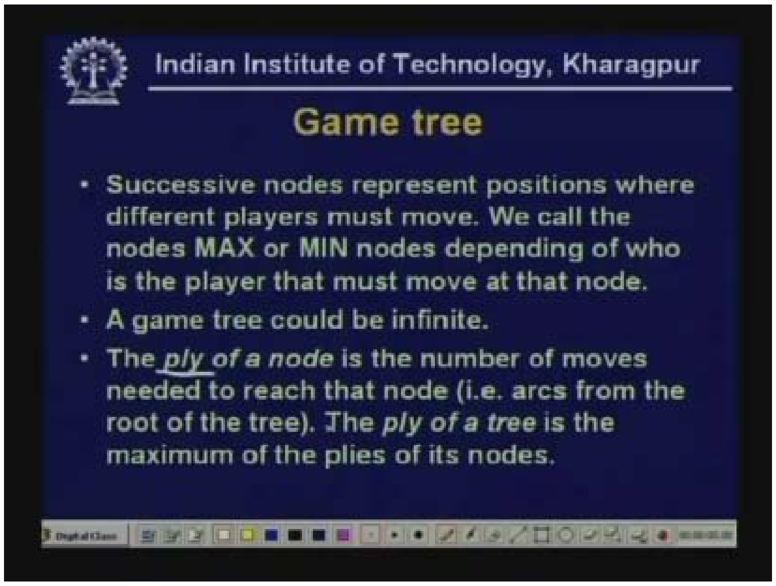
The two players take turns and try respectively to maximize and minimize a utility function.

The two players are called respectively MAX and MIN. We assume that the MAX player makes the first move

The leaves represent terminal positions

In two player games we have two players alternating, this is a zero sum game. It has perfect information where we know exactly that all the moves are deterministic and there is no chance involved. The two players take turns and try. The max players are the players who try to maximize utility and the other player the min player tries to minimize the utility function. We assume that the max player makes the first move. The leaves represent the terminal positions. Now, given the description of the game we will draw a game tree. In the game tree the successive nodes represent the positions where the different players make moves. A game tree could be infinite in size. So there could be a path which never leads to a goal state.

(Refer Slide Time 35:35)



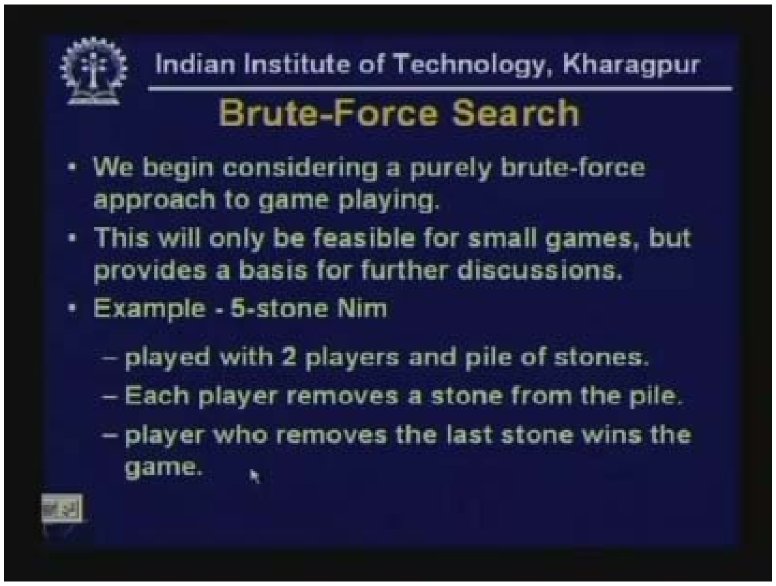
Indian Institute of Technology, Kharagpur

Game tree

- Successive nodes represent positions where different players must move. We call the nodes MAX or MIN nodes depending of who is the player that must move at that node.
- A game tree could be infinite.
- The *ply of a node* is the number of moves needed to reach that node (i.e. arcs from the root of the tree). The *ply of a tree* is the maximum of the plies of its nodes.

Here are some quick to finish definitions: The ply of the node is the number of moves needed to reach that node. Suppose this is the root of the game tree and then to reach a particular state we go through different levels. To reach this node from the root we need to move along three arcs. And given a complete tree the ply of a tree is defined as the maximum of the ply of its node. That is, the depth of the deepest leaf is called the ply of the game tree. First we are going to look at a brute force search algorithm to back off the utility function.

(Refer Slide Time 36:46)



Indian Institute of Technology, Kharagpur

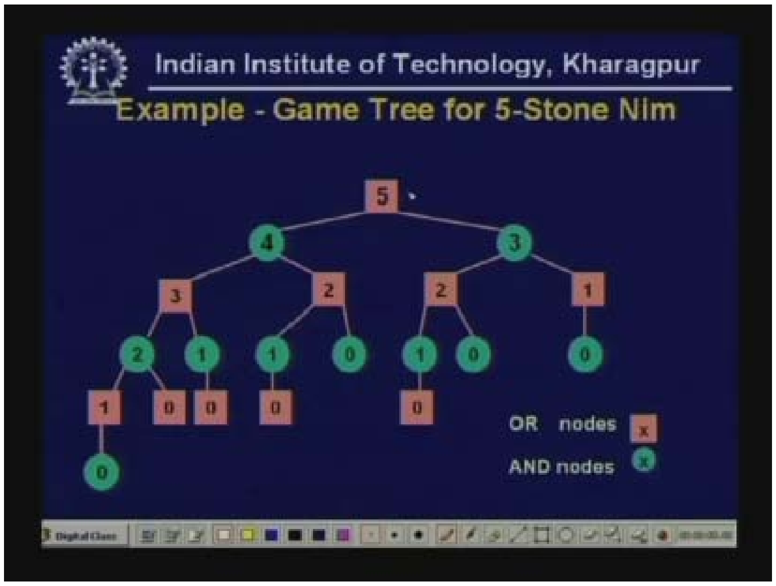
Brute-Force Search

- We begin considering a purely brute-force approach to game playing.
- This will only be feasible for small games, but provides a basis for further discussions.
- Example - 5-stone Nim
  - played with 2 players and pile of stones.
  - Each player removes a stone from the pile.
  - player who removes the last stone wins the game.

And this brute force approach wants us to start from the initial node and generate the entire search tree up to the leaf positions assuming that the game tree is finite. If the game tree is finite we can reach the leaf positions and infinite time if we have sufficient space. And we will see how to back up the utility values there by the mini max method.

However, for all but the most trivial games the game tree can be extremely large. And it will not be feasible for us to generate the entire game tree. Then we will see how to decide the best move using some heuristic. Nevertheless we will initially look at the brute force search and then will improve upon that. Let us take an example of a simple game 5-stone Nim. So this game is played with two players and a pile of stones. Suppose we have 15 stones to start with. Each player removes either one or two stones from the pile and the player who removes the last stone wins the game. This is the description of the game of the 5-stone Nim where each player can remove one or two stones at a time. Let us draw the game tree of this game.

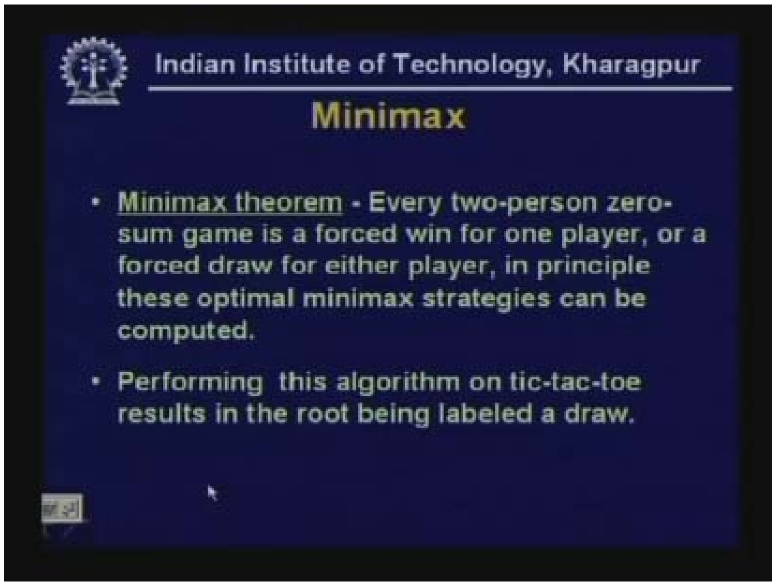
(Refer Slide Time 38:35)


Indian Institute of Technology, Kharagpur
Example - Game Tree for 5-Stone Nim
5
4
3
3
2
2
1
2
1
1
0
1
0
0
1
0
0
0
0
0
0
OR nodes
AND nodes

Initially let us assume that we start with 5 stones. Now, at this position the max player can move either one stone or two stones giving rise to either 4 or 3 stones left over. The min player can remove either 1 or 2 stones from 4 or from 3. Again the max player can remove 1 or 2 stones etc and we stop expanding the tree until all the leaves have 0 that is no stone left. The nodes are colored alternately by red and green. Red or pink represents the max player and green represents the min player. This is the max player and this is the min player.

Let us look at the leaves.
The green leaves correspond to win for min so the pay off for max here is minus 1. The pink 0s represent pink for max and the pay off is plus 1. Now let us apply mini max to this tree. Let us look at this node. It has only one child so its pay off will be same as this child which is minus 1. This node has two children whose utilities are minus 1 and 1. This is a minimizing node so the backed up utility value will be minus 1. At this node the backed up utility value is 1, this is a max node, the backed up utility value is the max of 1 and minus 1 which is 1. At this node the utility is 1, at this max node the utility is max of 1 or minus 1 which is 1 at this node the utility is min of 1 and 1 which is 1, at this node the utility is 1 and this node the utility is again 1. Here the utility is minus 1, here this is a min node the utility is minus 1 and max node utility is 1. So, for the max player the backed up utility at this node is plus 1 and the best move for the max player is along this direction that is removed once. At the max nodes the max player needs to consider only the best move. At the min node it must consider all possibilities and it must make sure that for all ways in which min could move max still has the best value. So these min nodes can also be looked upon as AND nodes and the max nodes as OR nodes.

(Refer Slide Time 41:56)


Indian Institute of Technology, Kharagpur

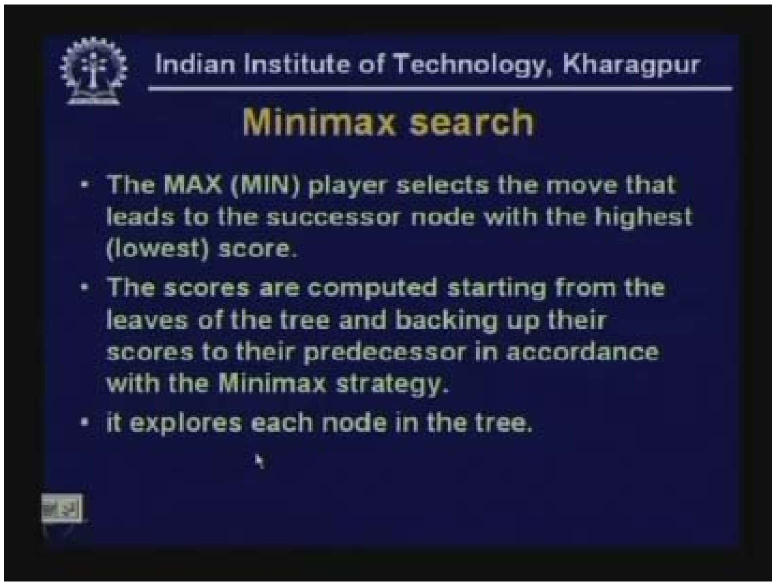
Minimax

- Minimax theorem - Every two-person zero-sum game is a forced win for one player, or a forced draw for either player, in principle these optimal minimax strategies can be computed.
- Performing this algorithm on tic-tac-toe results in the root being labeled a draw.

Minimax theorem:
The Minimax theorem states that every two person zero sum game is a forced win for one player or a forced draw for either player, in principle these optimal MINIMAX strategies can be computed. So, if we have the pay offs at the leaves and we back this up finally at the root we will get a value. That value can be either 0 or positive or negative. So if the value is positive, if max plays judiciously max can force a win from this position. If the backed up pay off value at the root is negative no matter how well max plays if min plays well and extremely intelligently then min can force Max's loss. If the pay off is zero if both the players play at their best this game would end in a draw

In the previous slide we looked at the game tree of 5-stone Nim and we saw that it was a win for max. Similarly, if we draw the game tree of Tic Tac Toe, Tic Tac Toe is a game which all of you are familiar with you have nine squares and you play cross the other player plays zero, you play cross, the other player plays zero, you play cross the other player plays zero and this is a win for min or win for O. Tic Tac Toe is actually a very simple game and you can see that the pay off of the root is 0 that is you can force a draw. If both players are intelligent you can force a draw.

(Refer Slide Time 43:58)



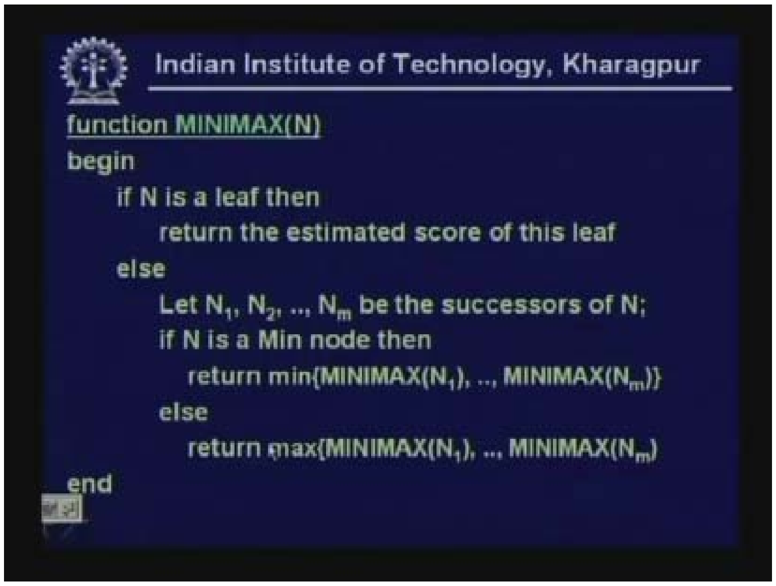
Indian Institute of Technology, Kharagpur

Minimax search

- The MAX (MIN) player selects the move that leads to the successor node with the highest (lowest) score.
- The scores are computed starting from the leaves of the tree and backing up their scores to their predecessor in accordance with the Minimax strategy.
- it explores each node in the tree.

In MINIMAX search the max player selects the max player selects the move that leads to the successor node with the highest score. Conversely the min player selects the move that leads to the successor node with the lowest score. The scores are computed starting from the leaves of the tree and backing up these scores to their predecessors in accordance with the MINIMAX strategy. In the MINIMAX strategy every node in the tree is exhibited. So you start from the root you back off at every intermediate node until you get the backed up value at the root.

(Refer Slide Time 44:52)

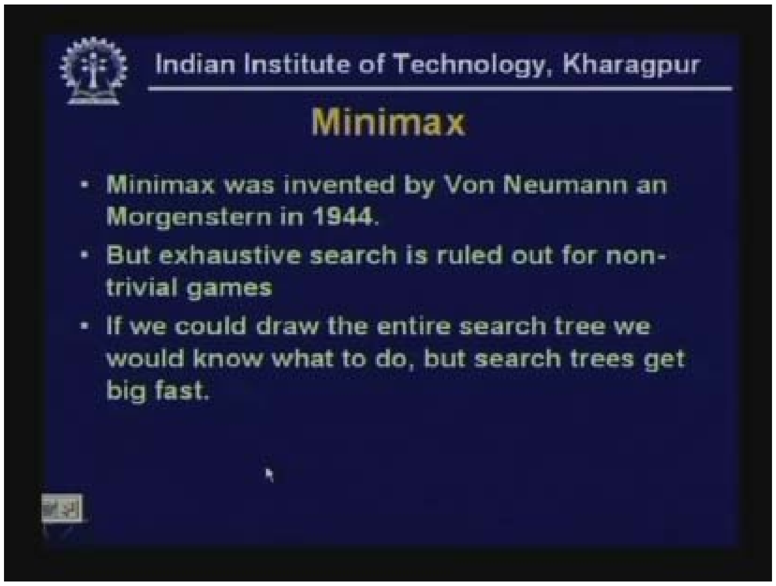


Indian Institute of Technology, Kharagpur

```
function MINIMAX(N)
begin
    if N is a leaf then
        return the estimated score of this leaf
    else
        Let $N_1$, $N_2$, ..., $N_m$ be the successors of N;
        if N is a Min node then
            return min{MINIMAX($N_1$), .., MINIMAX($N_m$)}
        else
            return max{MINIMAX($N_1$), .., MINIMAX($N_m$)
end
```

This is the pseudo code for the function MINIMAX. So we are starting at the node N in the function MINIMAX. If N is a leaf node then we return the pay off value of this leaf, the estimated score of this leaf. Otherwise if N is not a leaf suppose N has M successors N sub 1, N sub 2, N sub 3 up to N sub m if n happens to be a min node then we will return min of MINIMAX of N1 MINIMAX of N2 … MINIMAX of N sub m. Otherwise if n is a max node then we will return max of MINIMAX of N1 MINIMAX of N2 up to MINIMAX of Nm. So this is the MINIMAX algorithm.

(Refer Slide Time 45:58)



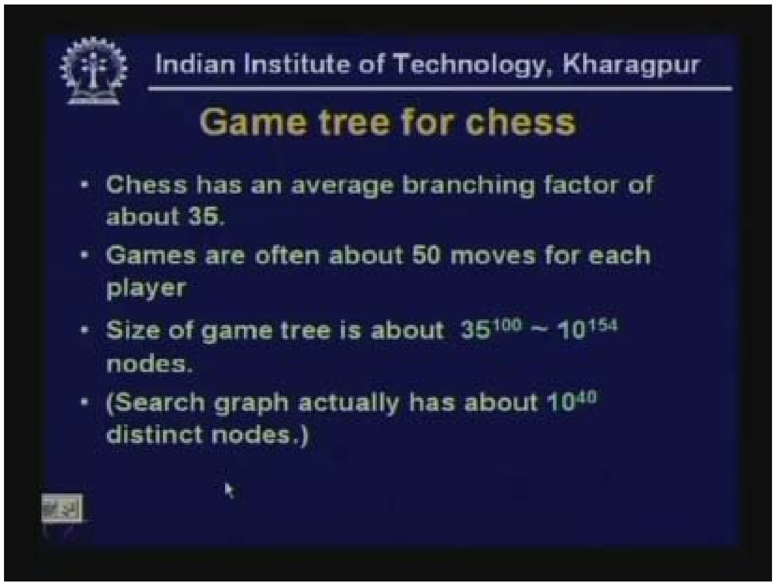
Indian Institute of Technology, Kharagpur

Minimax

- Minimax was invented by Von Neumann an Morgenstern in 1944.
- But exhaustive search is ruled out for non-trivial games
- If we could draw the entire search tree we would know what to do, but search trees get big fast.

This algorithm was invented by Von Neumann and Morgenstern on 1944. However, the game tree for all but the most trivial games become extremely large so we cannot do exhaustive search, we cannot generate the entire game tree. The MINIMAX strategy as we described earlier works when we can draw the entire search tree. But such trees can get big very fast, can get big exponentially so you know that if the tree has a branching factor of B then if we look at a search tree of depth D the number of nodes in the search tree is of order B power d. Thus, in an exponential number of nodes the search trees grow very fast. So, if D is not extremely small but D is reasonably large then if B is more than 1 this becomes large very fast.

(Refer Slide Time 47:08)

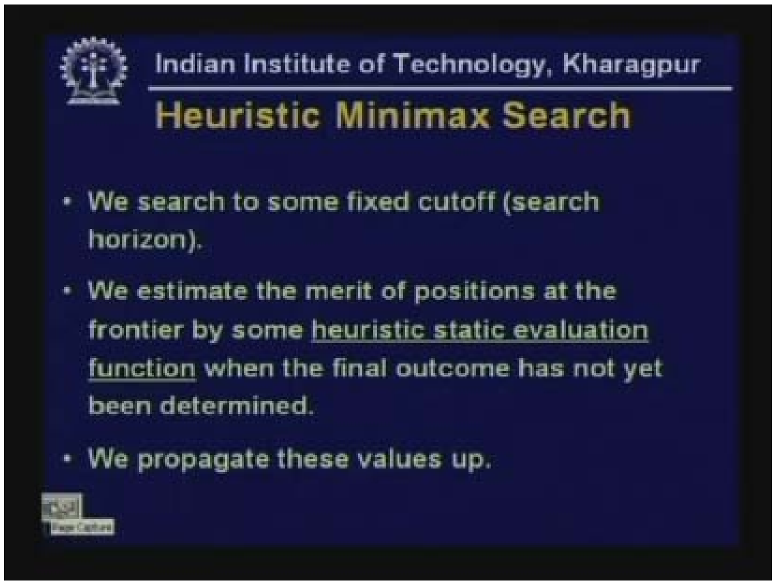

Indian Institute of Technology, Kharagpur

Game tree for chess

- Chess has an average branching factor of about 35.
- Games are often about 50 moves for each player
- Size of game tree is about $35^{100} \sim 10^{154}$ nodes.
- (Search graph actually has about 10^{40} distinct nodes.)

For example, if you consider the game tree for chess, chess has an average branching factor which is estimated to be about 35. Chess games are often about fifty moves for each player which means that the size of the game tree is about 35 to the power 100 which happens to be approximately 10 to the power 154 which is an extremely large number, inconceivably large number. However, if you consider not a search tree of chess but the search graph of chess in fact in chess there are about 10 to the power 40 distinct positions but even 10 to the power 40 is an extremely large number. So we cannot enumerate the entire search tree for chess in anyway.

(Refer Slide Time 48:03)


Indian Institute of Technology, Kharagpur

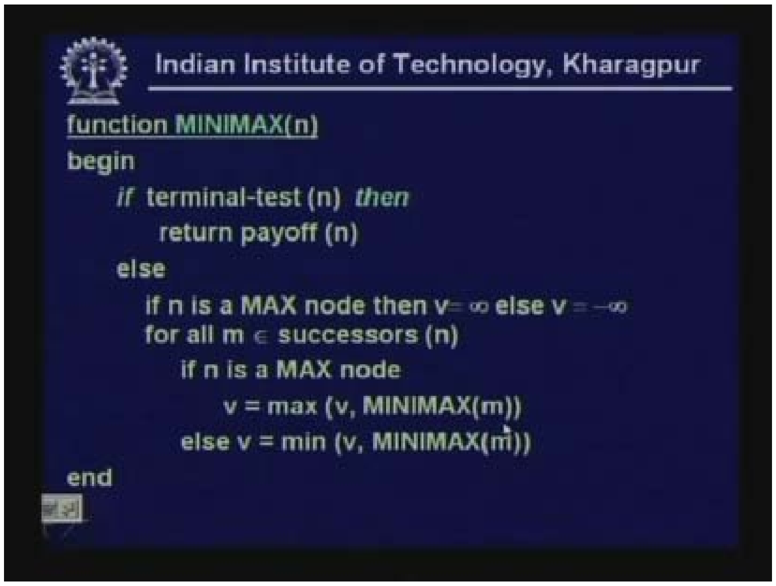
Heuristic Minimax Search

- We search to some fixed cutoff (search horizon).
- We estimate the merit of positions at the frontier by some heuristic static evaluation function when the final outcome has not yet been determined.
- We propagate these values up.

So what we do instead is to use heuristic MINIMAX search where we search to some fixed depth fixed cutoff which is called the search horizon. And if you only expand the tree up to a limited depth what will happen is that, we will not reach the leaf nodes. And if we do not reach the leaf node, that is, such an intermediate position of chess we will not know whether it is a win for white or win for black or draw for both of them. So we have to estimate at the intermediate position whether it is good for white or good for black. So we estimate the merit of positions at the frontier of the search tree by some heuristic static evaluation function because we do not yet know the final outcome.

Now, after we make some moves we have to know whether that move is a good move and we can even apply the static evaluation function at this level. But if you assume that when we grow deeper in the tree we can have a better evaluation function. We can expand the tree up to a cutoff, we can estimate the value of the boards at this step and on the basis of this estimate we can apply MINIMAX to find the backed off utility value at the root and also which is the best move for the root to take?

(Refer Slide Time 49:52)

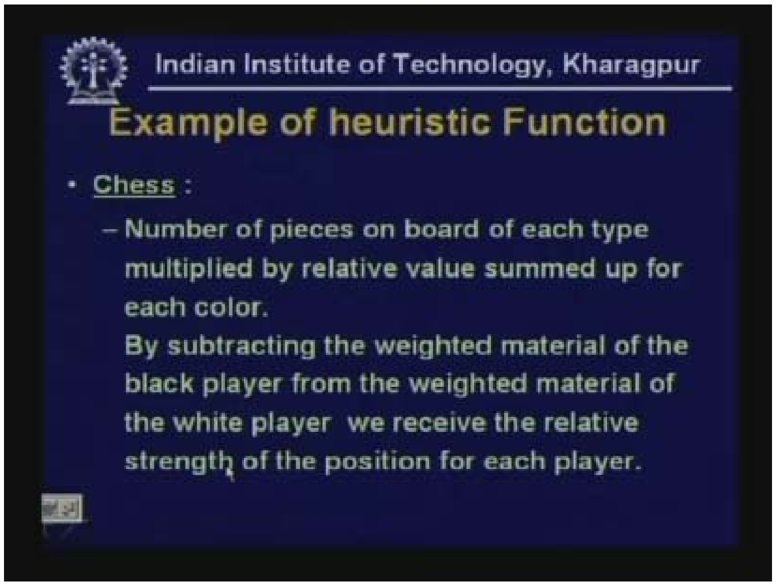


Indian Institute of Technology, Kharagpur

```
function MINIMAX(n)
begin
    if terminal-test (n) then
        return payoff (n)
    else
        if n is a MAX node then v= ∞ else v = −∞
        for all m ∈ successors (n)
            if n is a MAX node
                v = max (v, MINIMAX(m))
            else v = min (v, MINIMAX(m))
end
```

So the modified heuristic MINIMAX function will look like this MINIMAX at the node n. If node n has reached the cutoff has reached our terminal test of where we want to terminate search then we return the estimate of n we call this the pay off. So this will be an estimated pay off of n. Else if n is a max node then we set v equal to plus infinity. If n is a min node we set v equal to minus infinity. Then we find out all successors of n. For each successor m(n), if n is max node v equal to max of earlier value of v and MINIMAX of n. Otherwise if n is a min node then v is min of earlier value of v and MINIMAX of m. So, basically this is MINIMAX of the partial search tree where the search tree is only evaluated up to a certain cutoff. And the estimated value at this frontier is computed by some heuristics and then MINIMAX is applied on those estimated values.

(Refer Slide Time 51:22)



Indian Institute of Technology, Kharagpur

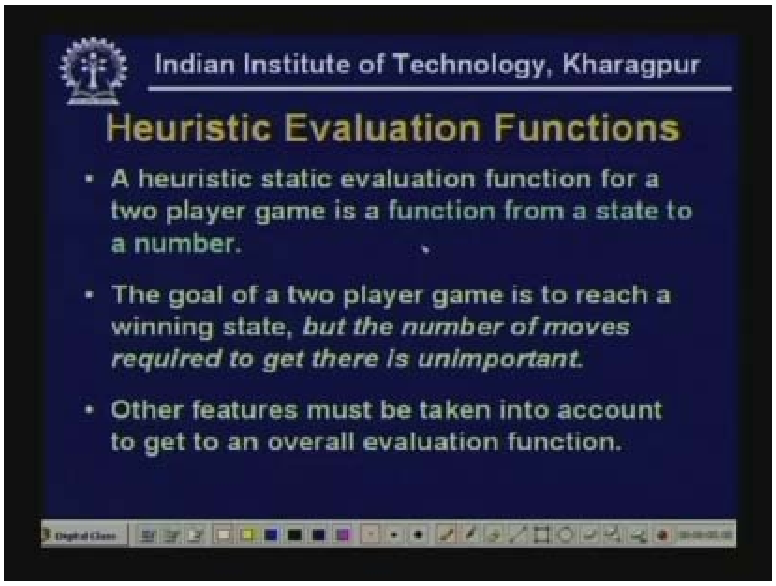
Example of heuristic Function

- Chess :
  - Number of pieces on board of each type multiplied by relative value summed up for each color.
    By subtracting the weighted material of the black player from the weighted material of the white player we receive the relative strength of the position for each player.

Here are some examples of heuristic function:
In chess a lot of people have written lots of programs on chess. We will just mention one very simple heuristic that is used for chess. It is, how good a board is for a player depends on how many pieces the player has. But just the number of pieces is not enough. The types of pieces which are there are also important. So, in chess the utility function could be the number of pieces on board of each type multiplied by relative value summed up for each color. We call this heuristic the material gain.

We find out the material gain for white minus the material gain of black to compute the utility of white at a node. And the material gain of white is the weighted sum of the pieces that white has. And each piece has a weight associated with it. Like a pawn can have some small weight may be the knight has a larger weight and so on. So this gives us an estimate of the board. Therefore heuristic static evaluation function for a two player game is a function from a state to a number, to a number which gives its estimated pay off.

(Refer Slide Time 53:03)



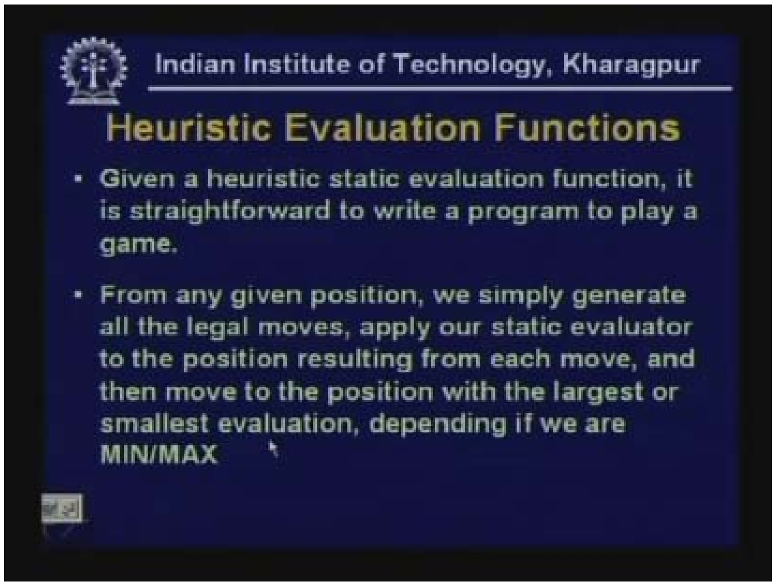
Indian Institute of Technology, Kharagpur

Heuristic Evaluation Functions

- A heuristic static evaluation function for a two player game is a function from a state to a number.
- The goal of a two player game is to reach a winning state, *but the number of moves required to get there is unimportant.*
- Other features must be taken into account to get to an overall evaluation function.

The goal of a two player game is to reach a winning state but the number of moves to reach that winning state is unimportant. So the length of the path is not important but what is important is whether we can get to a winning state or the state with a higher pay off or not. This is one of the differences with one agent search.

(Refer Slide Time 53:32)



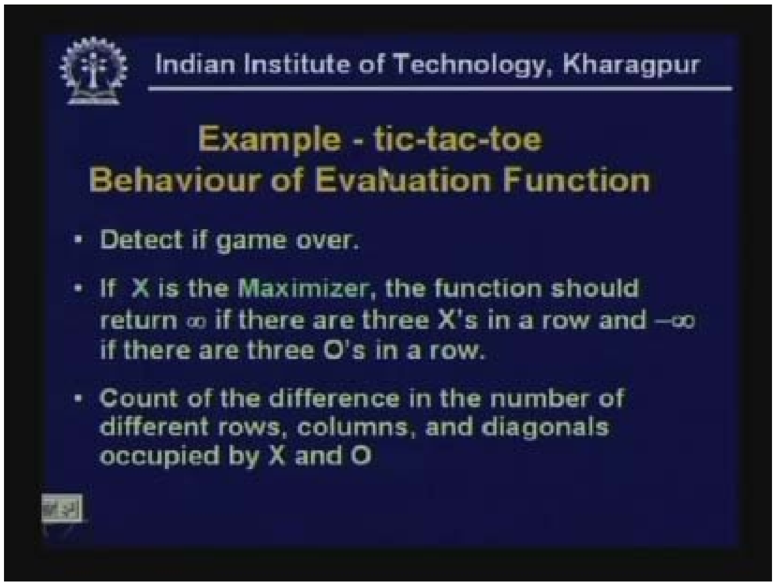
Indian Institute of Technology, Kharagpur

Heuristic Evaluation Functions

- Given a heuristic static evaluation function, it is straightforward to write a program to play a game.
- From any given position, we simply generate all the legal moves, apply our static evaluator to the position resulting from each move, and then move to the position with the largest or smallest evaluation, depending if we are MIN/MAX

So, if you design a heuristic static evaluation function for a two player zero sum perfect information game then we can use this MINIMAX strategy to play the game. From any given position we simply generate all the legal moves, apply our static evaluator to the position and then move to the position with the largest or smallest evaluation. Suppose we want to find out the best move to take from this position we find out all the successors, find out the evaluation of all the successors and take the max of them and move towards that. However, static evaluation functions are not very accurate. So instead of going on one level we might go for several levels and then back up the values there.

(Refer Slide Time 54:27)



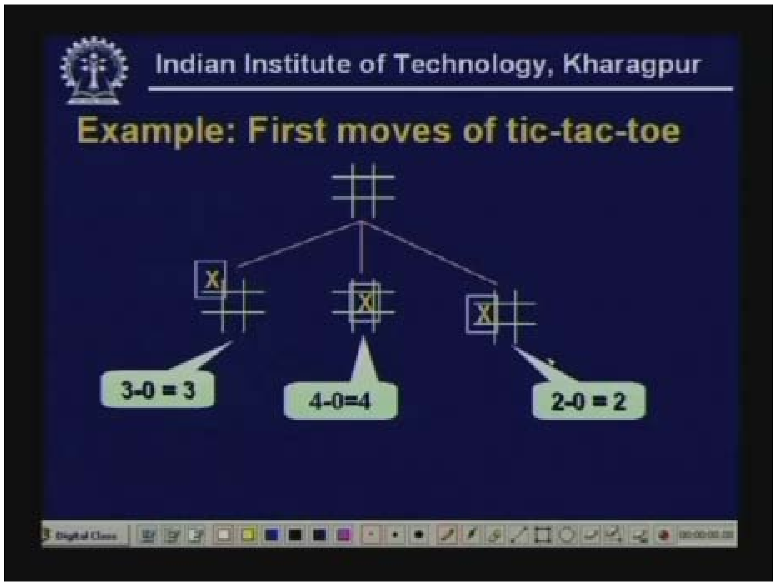
Indian Institute of Technology, Kharagpur

Example - tic-tac-toe
Behaviour of Evaluation Function

- Detect if game over.
- If X is the Maximizer, the function should return ∞ if there are three X's in a row and –∞ if there are three O's in a row.
- Count of the difference in the number of different rows, columns, and diagonals occupied by X and O

Let us look at Tic Tac Toe. In Tic Tac Toe we must have a function to detect if the game is over. If X is maximizer then the function returns infinity if there are three X's in a row which is win for X or plus 1 minus 1 if there are three O's in a row. And our static evaluation function is the count of the difference in the number of different rows columns and diagonals occupied by X and O. So we count the number of rows columns and diagonal occupied by X minus the number of rows columns diagonals occupied by O. Hence, this is a Tic Tac Toe search tree. This is the initial position, this is Max's turn to move. There are 9 possible successors but since this is symmetric there are three different actual successors.

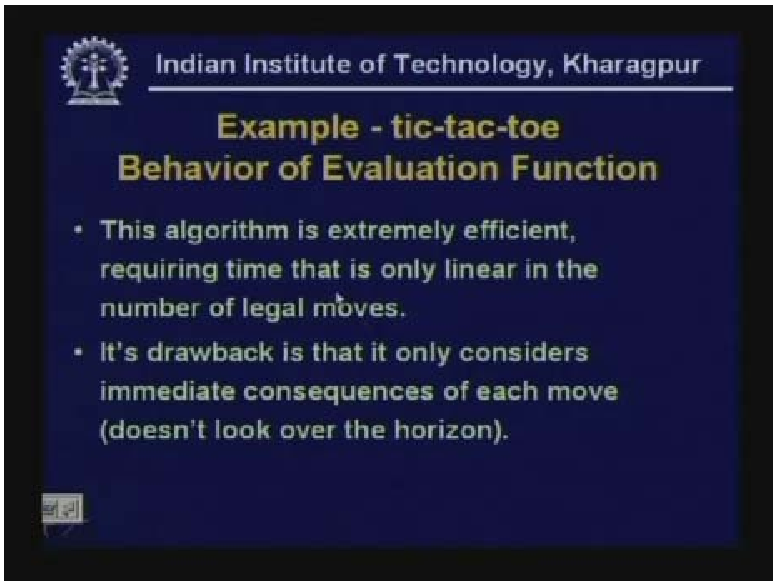
(Refer Slide Time 55:22)


Indian Institute of Technology, Kharagpur

Example: First moves of tic-tac-toe

For this one the evaluation is 3 minus 0 is equal to 3 so X occupies one row, one column, one diagonal and O occupies 0. Here X occupies one column one row and two diagonals, here X occupies one row and one column. So the evaluation is 3 here 4 here 2 here so this is the best move to take for X based on this one level of look ahead.

(Refer Slide Time 55:59)

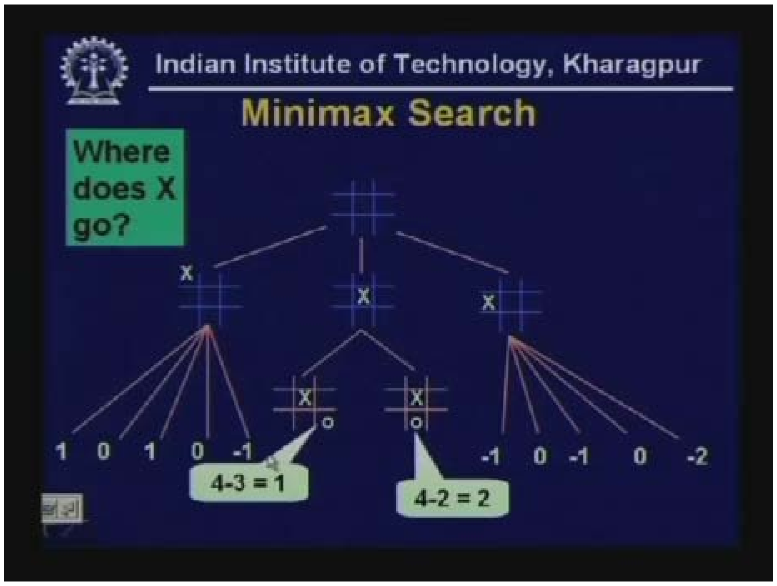

Indian Institute of Technology, Kharagpur

Example - tic-tac-toe Behavior of Evaluation Function

- This algorithm is extremely efficient, requiring time that is only linear in the number of legal moves.
- It's drawback is that it only considers immediate consequences of each move (doesn't look over the horizon).

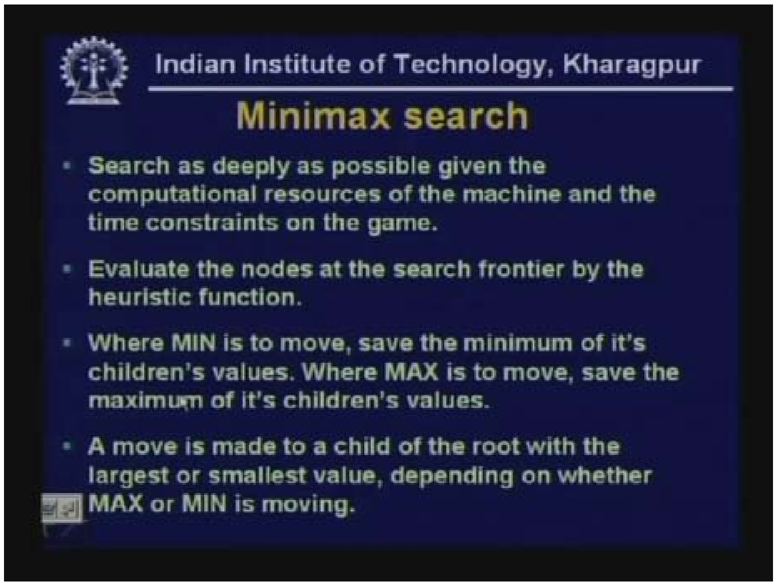
This algorithm is very efficient but it only considers immediate consequences of move. That is, it does not look over the horizon.

(Refer Slide Time 56:06)



Therefore, in MINIMAX search we expand the search tree further, look at further depth and then find out, suppose this looks at one more depth and finds out the evaluation at this depth and uses MINIMAX to backup these values. For example, this board has a pay off of 1, this has a pay off of 2 so this pay off will be min of 1 or 2 that is 1. Similarly, we will just find out, here the min is minus 2, here the min is 1 and here the min is minus 1 so max will still move here because it has a pay off of 1. So we search as deeply as possible and then use MINIMAX.

(Refer Slide Time 56:59)

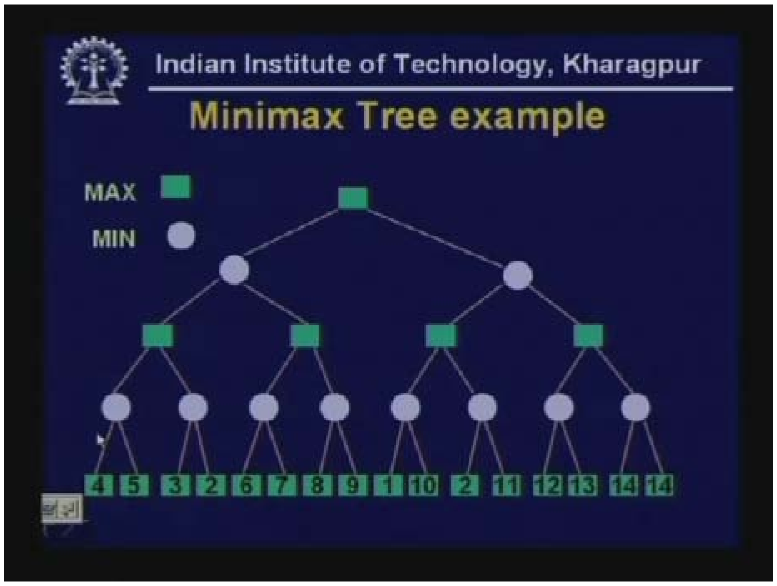


Indian Institute of Technology, Kharagpur

Minimax search

- Search as deeply as possible given the computational resources of the machine and the time constraints on the game.
- Evaluate the nodes at the search frontier by the heuristic function.
- Where MIN is to move, save the minimum of it's children's values. Where MAX is to move, save the maximum of it's children's values.
- A move is made to a child of the root with the largest or smallest value, depending on whether MAX or MIN is moving.

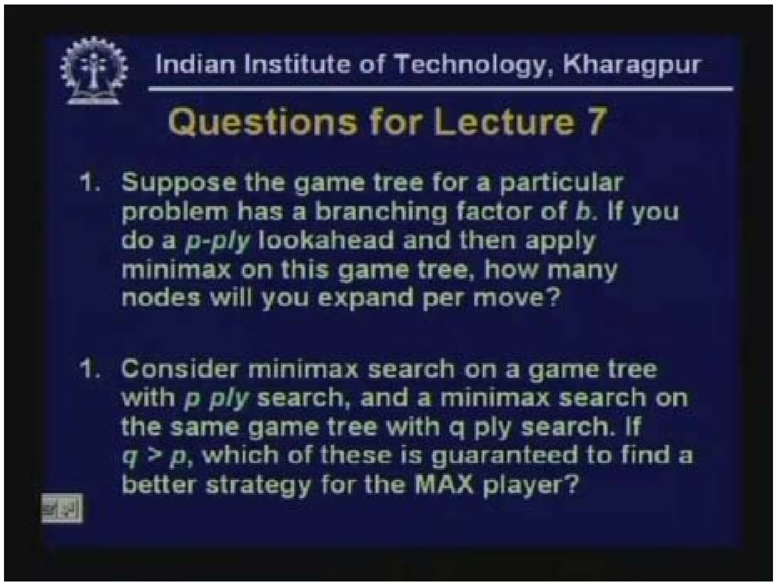
(Refer Slide Time 57:04)


Indian Institute of Technology, Kharagpur

Minimax Tree example

We will just stop with this last example of the MINIMAX tree. This is a MINIMAX tree with showing us the terminal evaluation functions. In MINIMAX we will evaluate this node which is 4, this is 2, this is minimizer nodes. The blue ones are minimizing nodes, the green ones are maximizing nodes and then the final evaluation function is 4.

(Refer Slide Time 57:49)


Indian Institute of Technology, Kharagpur

Questions for Lecture 7

1. Suppose the game tree for a particular problem has a branching factor of *b*. If you do a *p-ply* lookahead and then apply minimax on this game tree, how many nodes will you expand per move?

1. Consider minimax search on a game tree with *p ply* search, and a minimax search on the same game tree with q ply search. If *q > p*, which of these is guaranteed to find a better strategy for the MAX player?

Questions for lecture 7:

Suppose the game tree for particular problem has a branching factor of b.

If you do a p-ply look ahead and then apply MINIMAX on this game tree how many nodes will you expand per move?

Question number 2:
Consider MINIMAX search on a game tree with p-ply search and another MNIMAX search with a q-ply search. If q is greater than p which one is better?