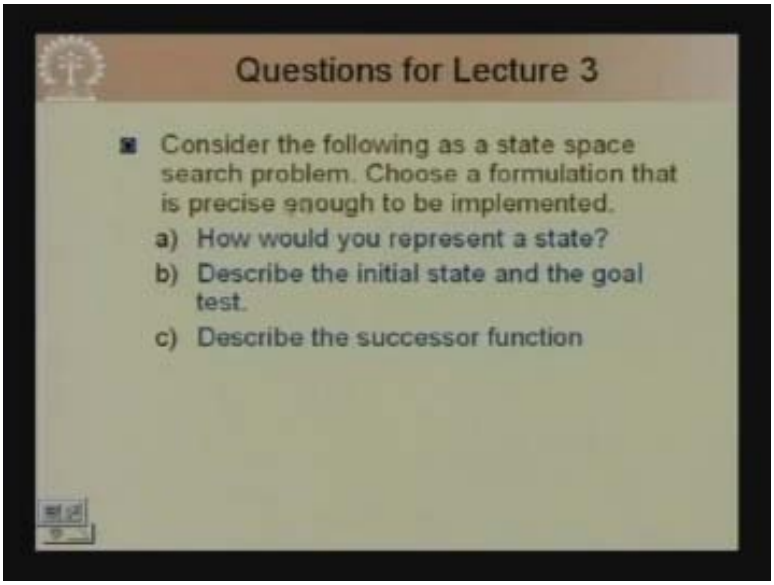
Artificial Intelligence
Prof. Sudeshna Sarkar
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
Lecture - 4
Uninformed Search

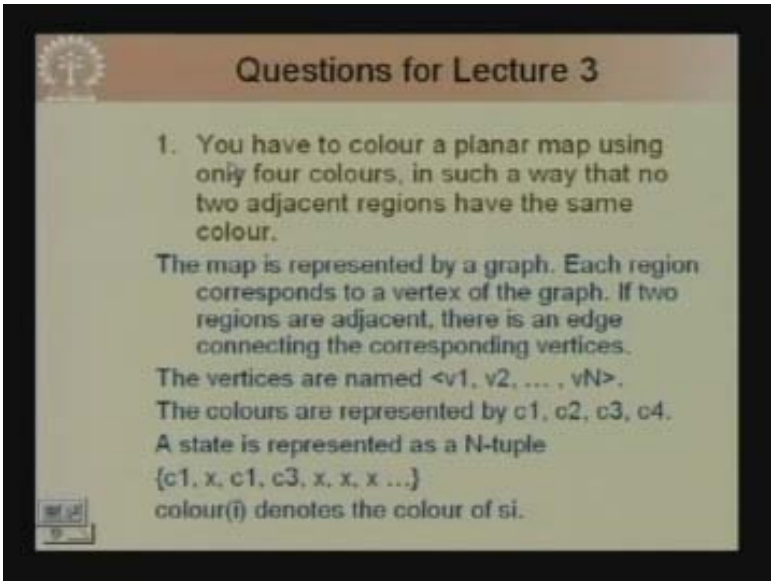
Today we will start the fourth lecture of this course Artificial Intelligence. Today's topic is uninformed search. The last class we saw how the state space can be represented. Today we will look at some of the strategies and algorithms that are used for search. Before we start today's lecture we will look at the questions of the previous class and the answers to those questions.

(Refer Slide Time 01:07)



The first question of last class was this; consider the following as a state space search problem. Choose a formulation that is precise enough to be implemented. For each of the problems that we have described you would have to show the representation of the state, describe the initial state and the goal test and describe the successor function.

(Refer Slide Time 01:30)

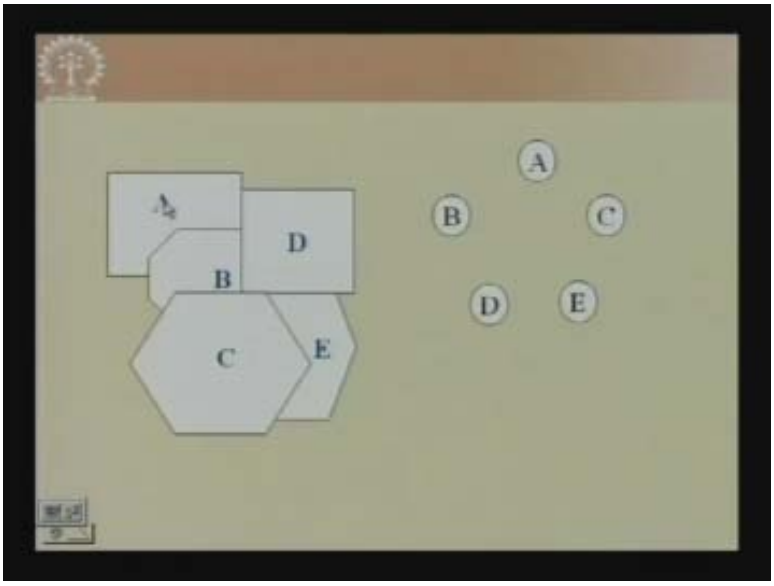


The problems we looked at in the last class were, number one was the graph coloring problem. In the graph coloring problem you have to color a planar map. You have a map which has a number of regions. As you all know, in the map there are several regions. Different regions may denote different countries or different states and you have to use some number of colors to color these regions such that no two adjacent regions have the same color.

Now we will look at our representation of this map coloring problem. Rather than graph coloring this is the map coloring problem. What we will do is we will first represent the map by a graph. For each region we will have the vertex of a map. Suppose this is region A, this is B, this is C this is D, this is E and this is F we will say that the region ‘a’ corresponds to the vertex of a graph. And b and f are adjacent to a. For B, A, F and then D and E these are adjacent to B and so on. We will represent this map by a graph where for every region we will have a vertex of a graph and if two regions are adjacent we will have an edge connecting those two vertices. So we will call these vertices as v1 v2 vn and we will represent the colors by c1 c2 c3 c4. In the map coloring problem we will use four colors to color the map.

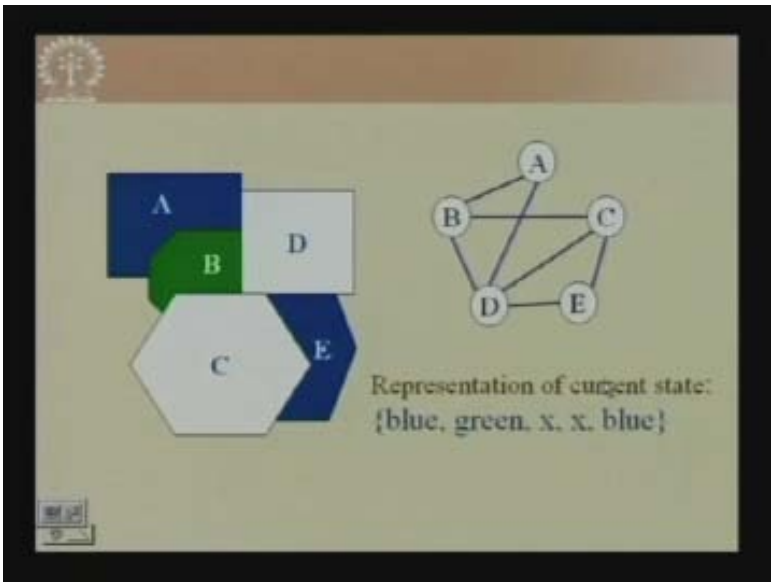
In fact it has been shown that any planar map can be colored by four colors. What we will do is a state will be represented by an n tuple. So, for every region we will have one entry in the state representation. If a region is colored by a particular we will use that color as the attribute for that region. If a particular region is not colored at a particular time we will represent that region by x. So let us look at this example.

(Refer Slide Time 04:33)



This is an example of a map. In this map there are five regions A B C D and E . Corresponding to this we are constructing a graph which has five vertices. Now A and B are connected, A and D are connected, B and C, B and D, C and d, C and e and D and E. So this graph represents this map. Now this is a particular state of the graph where A and E are colored blue, B is colored green, C and D have not been assigned colors as yet.

(Refer Slide Time 05:19)

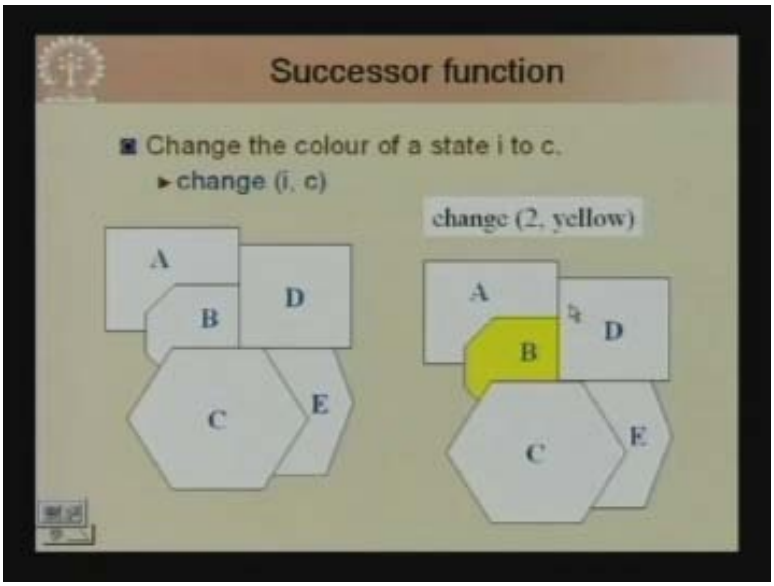


We will represent this state as blue green x x blue. Therefore for this map coloring problem in the initial state we will start with all the states unassigned, all the regions unassigned and none of them has been assigned a color. And we will have reached a goal

test if when all the five regions are colored such that if two vertices si and sj are adjacent their colors are not equal. So goal state is a state where all regions have been assigned colors and the coloring does not violate the constraints.

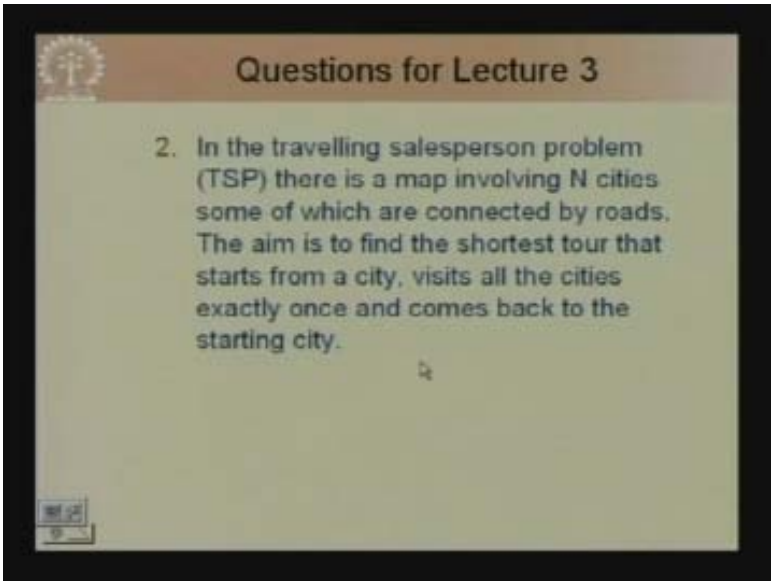
Now, what are the different operations we can apply to the state space?
For example, this is a particular map. We can apply the following operator:
Change to yellow, that is, change region 2 that is b to the color yellow. And as a result we get this state.

(Refer Slide Time 06:28)



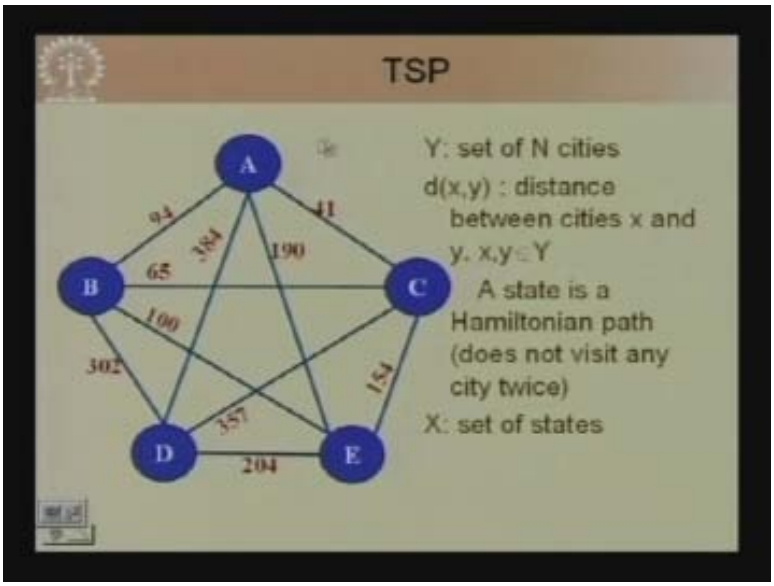
Now on this state if I again apply the operator change three green we get this state. So this is a state space representation of the graph coloring problem. The second problem we assigned in the last class was the traveling salesperson problem. In the traveling salesperson problem we have a map involving n cities some of which are connected by roads. The objective is to find the shortest tour that starts from a city, visits all the cities exactly once and comes back to the starting city.

(Refer Slide Time 07:10)



Now this is how we will represent the traveling salesperson problem.

(Refer Slide Time 07:19)

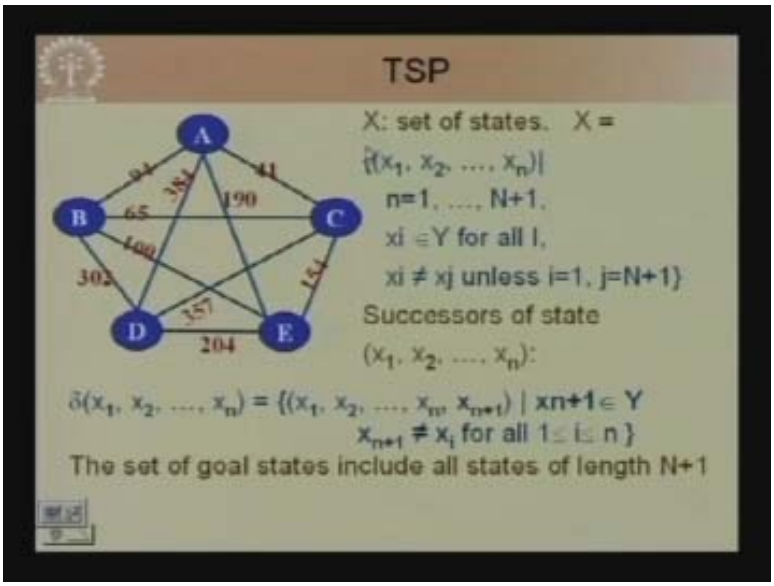


This is the map, this is the graph corresponding to the traveling salesperson problem. And A, B, C, D, E are the different cities and there are roads connecting some of these cities. For example, the distance between city A and B by the direct road is 94 units. Distance between A and D is 384 units and the distance between A and E is 190 units and so on. So we represent by y the set of cities. In this case y is equal to the five cities A, B, C, D and E.

We represent by this distance function d(x, y) the distance between cities x and y. Corresponding to this traveling salesperson problem we will design the state space. In the state space a state would be represented as a path which a path connecting a number of cities such that in this path no city is visited more than once. So such a path is called a Hamiltonian path. In graph theory such a path is called a Hamiltonian path. So a state for our state space representation will be a Hamiltonian path. In this graph a Hamiltonian path is a path which does not visit any city twice.

Now for our representation x will be the set of states in our state space representation. Each state is a Hamiltonian path of this TSP graph.

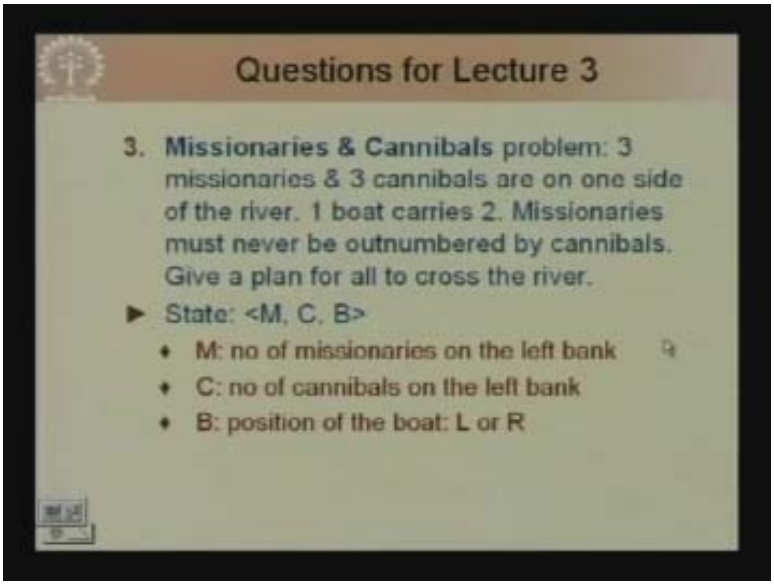
(Refer Slide Time 09:19)



So each state is represented by x_1 x_2 x_n. This is a path such that each of these x_i corresponds to a city and no two cities are same. That is, x_i not equal to x_j unless we get the full traveling salesperson tour that is i is equal to 1 and j is equal to N plus 1 where there are n cities in this TSP problem so we have n cities. A TSP tour is actually a tour of N plus 1 cities where the first city is the same as the N plus 1th city. So that salesperson completes a tour of all the cities and comes back to the starting city and every other city in the path is distinct. So this is the representation of the state. The successor of a state x_1 x_2 x_n is represented like this. So delta x_1 x_2 x_n is the successors of this state or all states which are extensions of this Hamiltonian path. Therefore, at those states which are x_1 x_2 x_n plus 1 where x_n plus 1 is a city which is not equal to any other city which already existed in this path. Now this is the state space representation of the TSP problem.

What is the goal?
The goal state is a state where we have a Hamiltonian tour of length n plus 1. So the set of goal states include all states of length n plus 1.

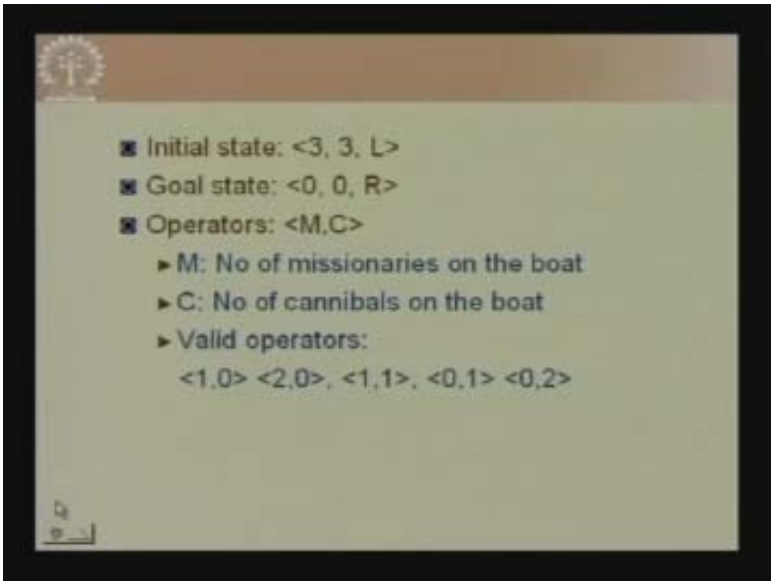
(Refer Slide Time 11:19)



The third problem was the missionaries and cannibals problems. Three missionaries and three cannibals are on one side of the river, there is one boat which can carry only two persons at a time. Missionaries must never be out numbered by cannibals, you need to give a plan to cross the river and for this problem I want you to come up with a state space representation. Now the answer to this is; one way of representing a state is by a three tuple where we represent M C B. M is the number of missionaries on the left bank so this is a river which the missionary is trying to cross.

Initially we have 3 missionaries and 3 cannibals on the left side of the river. So this is the left side and this is the right side and we have one boat and this boat is initially on the left side and it can carry 2 people. Now the state will be represented by three numbers. Here M is the number of missionaries on the left bank so initially M is 3, in the goal state M will be 0 because all the missionaries need to be on the right bank, C is the number of cannibals on the left bank. Initially C is 3 and in the goal state C will be 0 and B is the position of the boat whether the boat is the left bank or the right bank.

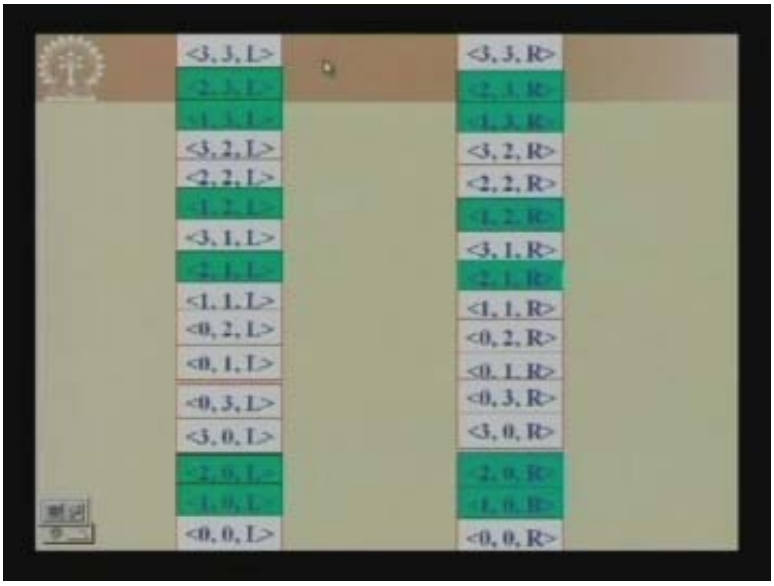
(Refer Slide Time 12:53)



In the beginning the boat will be on the left bank. So the initial state will be presented by 3, 3, L. That is 3 missionaries, 3 cannibals on the left bank and the boat also on the left bank. And the goal state will be represented by 0, 0, R that is no missionary on the left bank, no cannibal on the left bank, 3 missionaries on the right bank, 3 cannibals on the right bank and the boat is on the right bank.

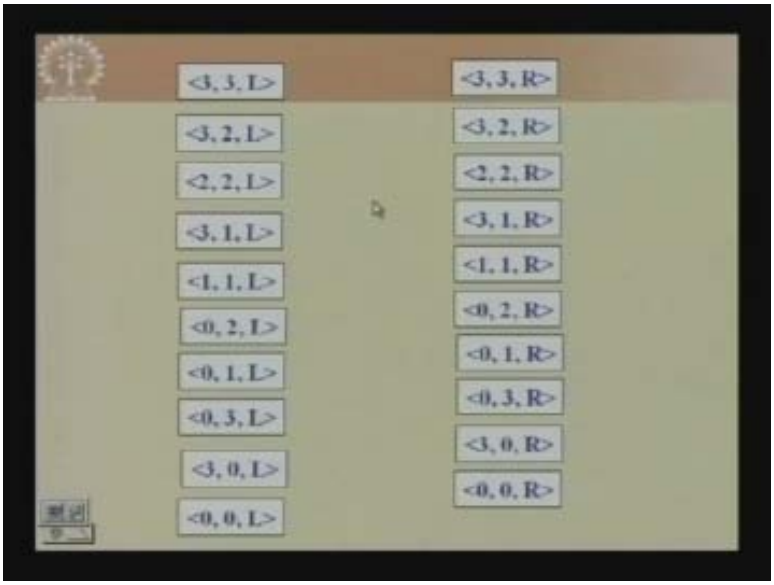
The operators for this problem are those are two tuple M and C. So M is the number of missionaries on the boat and C is the number of cannibals on the boat. Since only at most 2 persons can be on the boat at the same time the possible operators are 1 0 2 0 1 1 0 1 and 0 2 and 0 0 is not a valid operator because the boat needs at least 1 person to steer the boat.

(Refer Slide Time 13:55)



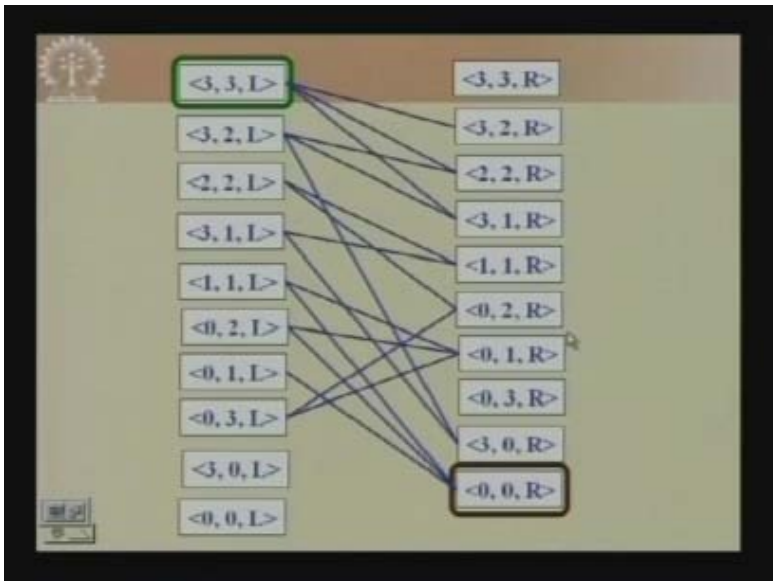
So what we find is that, each state is represented by a three tuple M C B. So m can take four values 0 1 2 3, C can also take four values 0 1 2 3 and b can take two values L or R. Therefore the possible number of states is 4 into 4 into 2 that is 16 into 2 is equal to 32. This 16 contains the boat on the left side and this 16 contains the boat on the right side. But some of these states violate the constraints that missionaries must never be out numbered by cannibals on any side because otherwise the cannibals will devour the missionaries. And if you apply that constraint these green states become invalid states. So finally these are the only valid states.

(Refer Slide Time 15:00)



There are 20 valid states in this state space. And if we apply the operators to each of this state we get the state space graph. This is the initial state:
3, 3, L 0, 0, R is the goal state and these arcs denote the successors of each state. That is, the operators that can be applied on each state to go to the other state. So this is a state space representation of the missionaries and cannibals problem.

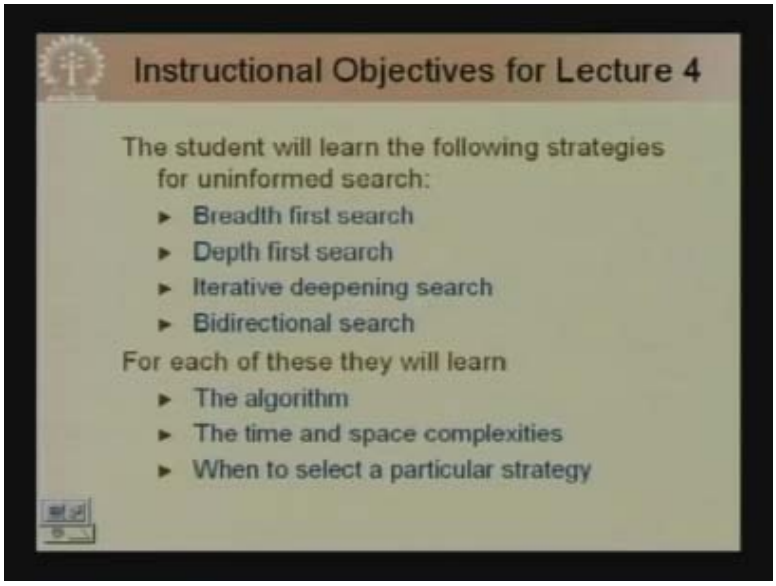
(Refer Slide Time 15:24)



Now we are ready to start today's lecture which is module 2 which I started in the last class, the problem solving using search. In today's lecture we will talk about uninformed search. The instructional objectives of today's lecture: In this lecture the student will learn the following strategies of uninformed search, breadth first search, depth first search, iterative deepening search, bidirectional search.

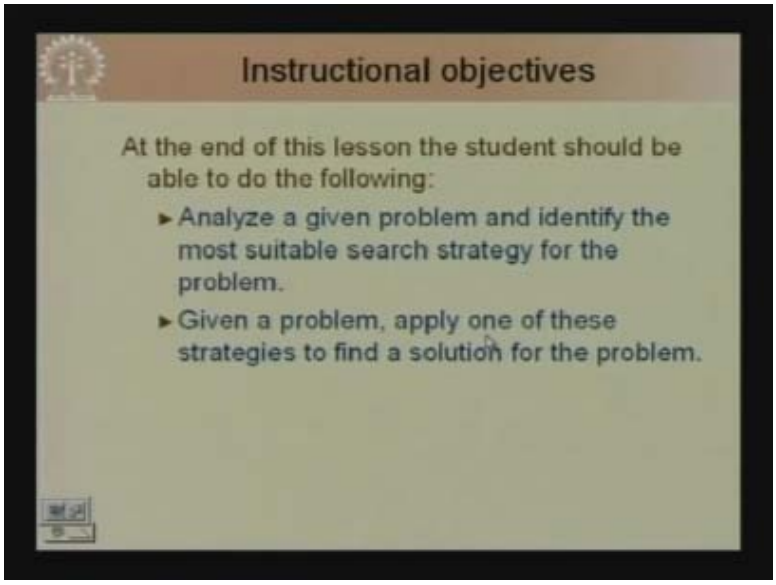
So we will look at these four strategies for uninformed search. We will know what uninformed search is. Uninformed search means searching through the state space without using any extra information, without using any domain specific information. For each of these search strategies the student will learn the algorithm for this strategy and we will analyze each of these algorithms to find their time and space complexities and the student will also learn when to select a particular strategy for a given problem, on what basis given a problem which strategy is the best strategy to select. At the end of this lesson the student should be able to do the following:

(Refer Slide Time 16:50)



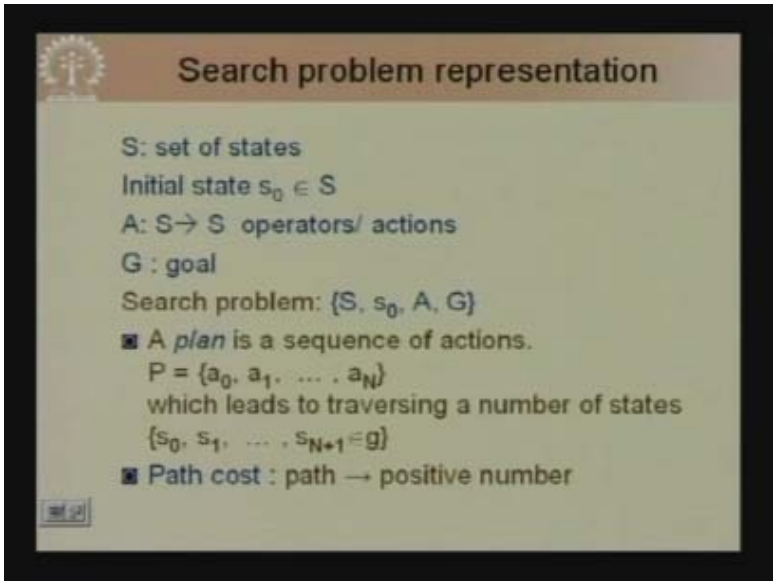
You should be able to analyze a given problem and identify the most suitable search strategy for the problem.

(Refer Slide Time 17:15)



Given a problem she should be able to apply one of these strategies and find the solution to the problem, find the sequence of nodes that will be expanded and what would be the final solution that will be obtained.

(Refer Slide Time 17:26)



Here is a quick recapitulation of the representation of the search problem. We have seen in the search problem we represent s to be the set of states, the initial state is s_0 which is a member of S, we have a set of operators or actions a and we have a set of goals g represented by an explicit set of goal states or represented by a goal test or by explicit goal states.

A solution to a search problem is a plan which is a sequence of actions a_0 a_1 a_n which leads to traversing a number of states starting from state s by applying action a sub 0 the agent goes to state s sub 1 and from s sub 1 by applying action a sub 1 the agent goes to s sub 2 and so on. Finally the agent arrives at state $s_{N \text{ plus } 1}$. And if this plan is a solution is a solution to the search problem then $s_{N \text{ plus } 1}$ must be a goal state. We also mentioned that for every path we will associate a cost. Therefore, path cost is a positive number and usually path cost is the sum of the operator costs. Now, this is the state space of missionaries and cannibals problem which we just discussed.

Now let us see what is a plan for this missionary and cannibal problem?
We have already seen the state space for this problem and we have identified that 3, 3, L is the initial state and 0, 0, R is the goal state. So 3, 3, L is the initial state. We will trace a solution so one of the operators brings which operator by taking two cannibals by a boat one comes to the 3, 1, R state and from there the agent can come to the 3, 2, L state and then to the 3, 0, R, 3, 1, L 1, 1, R, 2, 2, L, 0, 2, R, 0, 3, L, 0, 1, R, 0, 2, L and then 0, 0, R. So this is a solution to the missionaries and cannibal problem. By applying a sequence of operators the agent can traverse the state space and reach the solution.

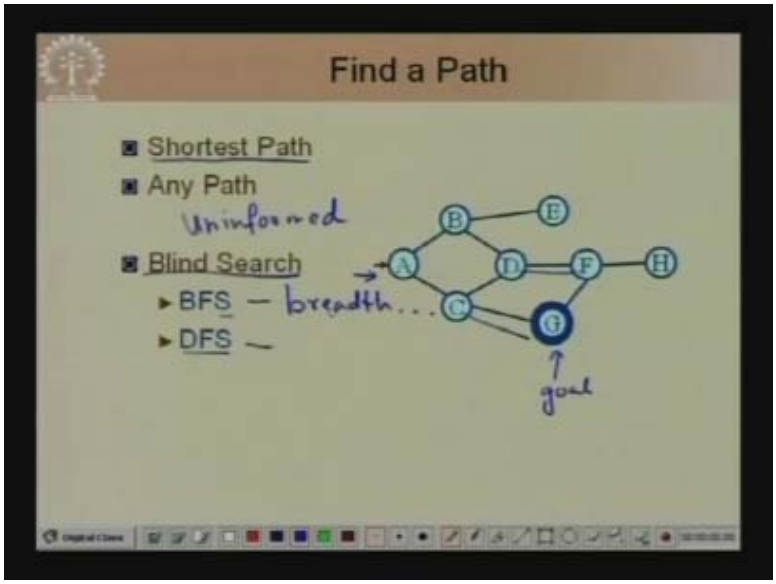
Now, in today's lecture we will look at various strategies to systematically explore the state space by using a search graph or a search tree. Corresponding to this state space we will see how to construct the search tree or a search graph. And we will also see how to

systematically explore this search tree or search graph in order to find a plan or a solution to this problem.

Now this is an abstract example of a very simple state space. We have these eight states A B C D E F G H and these arcs show the connectivity information between the states. Here A is the initial state and G is the goal state. Now, when we look at a search algorithm we would be interested in finding a path from the start state to one of the goal states. Now there are several alternatives for which our algorithm will strive for. The objective of the algorithm may be to find the shortest path from the start to a goal state or it could be to find any path from start to the goal.

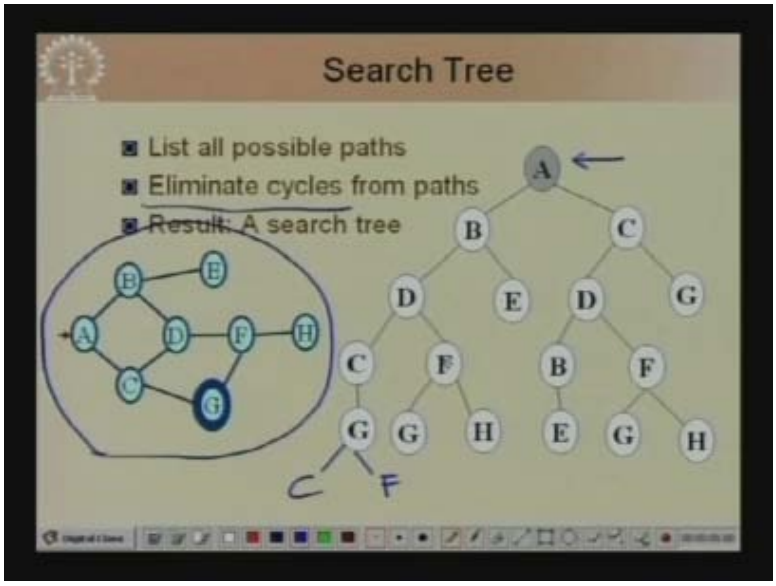
Today we will look at blind search or uninformed search which means that the algorithm does not use any information about the domain. And the most important blind search strategies are Breadth First Search and Depth First Search. In short it is BFS or Breadth First Search and DFS Depth First Search.

(Refer Slide Time 22:34)



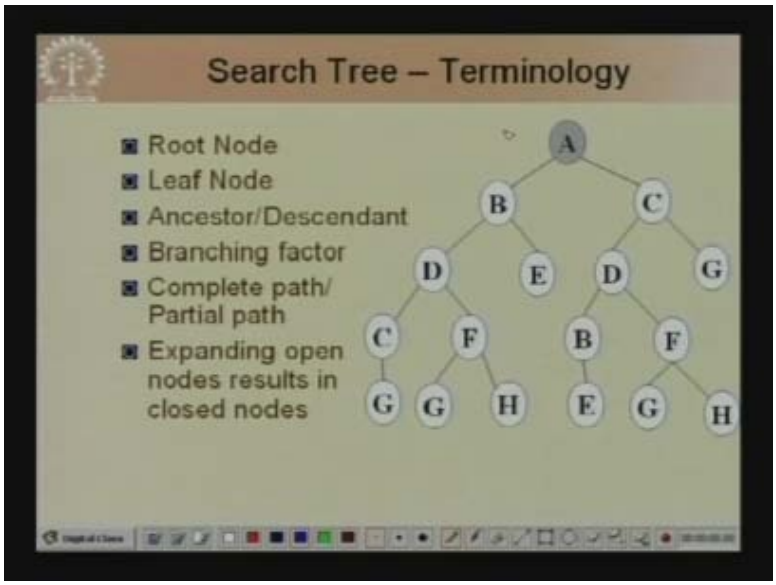
Now we will see what is a search tree corresponding to a problem. How do we get the search tree?
This is the state space graph. From this state space graph we will start from the initial node and we will list all possible paths. So we start from a, from A one can go to B or C, from B one can go to D or E, from D one can go to C or F, from C one can go to G, from F one can go to G or H but again from G one can go to C or F etc. So actually if we take these states, and unfold this graph into a search tree we may get an infinite search tree. In order to make this search tree finite we will need to eliminate cycles from every path. That is, if in this path we have already visited C then from G we will not come back to C. Anyway G is a goal state. So in order to construct a search tree we start from the initial state we unfold the graph and we list all possible paths until one reaches to a goal state or to a dead end and the result is a search tree.

(Refer Slide Time 24:15)



Now let us look at some terminologies corresponding to a search tree.

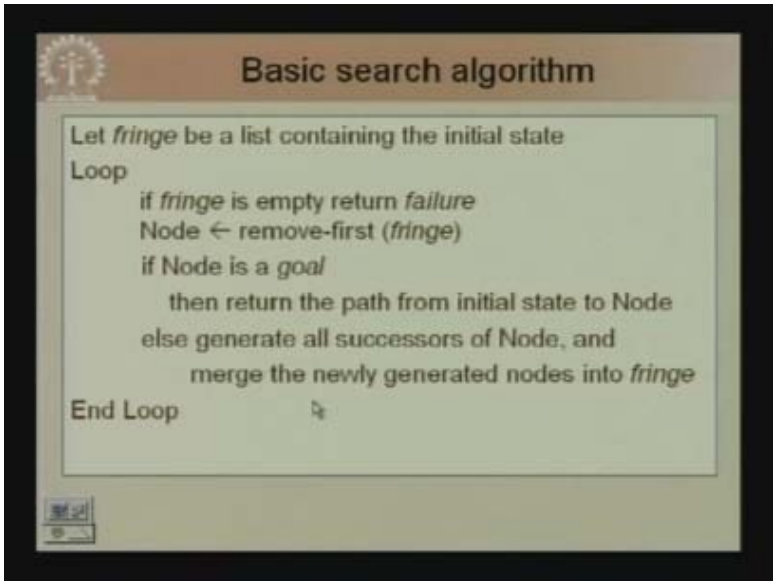
(Refer Slide Time 24:33)



Now the root node is the node from where we initiate the search tree. So in this example A is the root node. The leaf nodes are the leaves of this tree. This particular tree has the following leaves:
G G H E E G H G out of which G is a leaf node because G is a goal node and E and H are leaf nodes because we have reached dead ends.

For the search tree E's parent is B and ancestors are B and A. For this node F the ancestors are D, B and A. The descendents of F are G and H, the descendents of C are D, G, B, F, E, G and H. The branching factor means the maximum number of children of a node. So, in this particular search tree we see that every node has less than equal to two children. Therefore in this case branching factor equal to two. We represent the branching factor usually by this notation b, b is equal to 2. A B D C G is a complete path. A B D is a partial path.

(Refer Slide Time 26:07)



Now let us look at a basic search algorithm, a systematic way of searching the entire search space. We will put certain restrictions on this general search algorithm to come up with special algorithms. So, in the basic search algorithm we maintain a data structure called the fringe. A fringe is referred by many people as open or queue or frontier. So fringe will contain those nodes that our algorithm can expand at a particular time.

Initially fringe will contain only the initial state of the state space. Fringe is a list which contains the initial state in the beginning. The search algorithm will execute this loop. First step is, if fringe is empty, if there is no more state to expand and the algorithm has not found a solution n yet the algorithm returns failure, if fringe is empty return failure which means no goal found and no node remains to be expanded. Otherwise the algorithm removes the first node from the fringe. Let us call this a node. If node is a goal here the goal test is applied. If node is a goal then the algorithm returns the path from initial state to node.

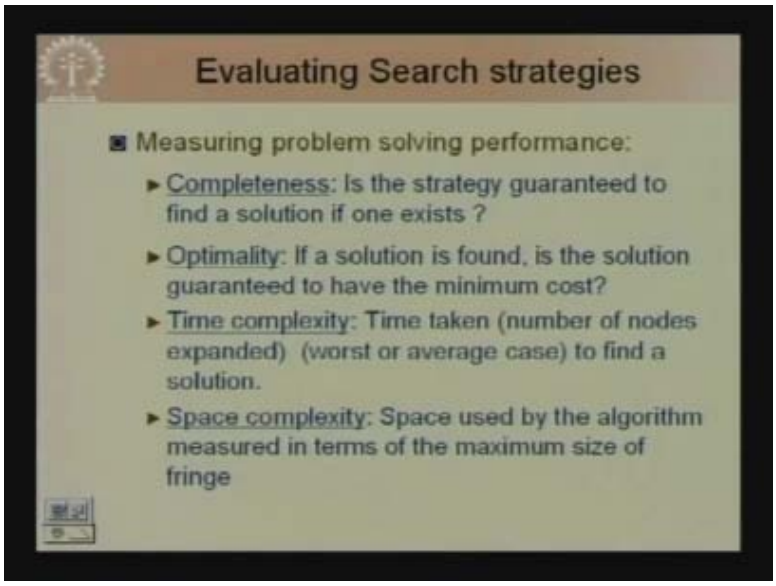
Starting from initial state when the algorithm expands node finds that node is a goal and the algorithm needs to return the path to come from the initial state to the goal. Otherwise it will generate all successors of node and incorporate those successors of node into fringe. The algorithm will put the newly generated successors into fringe. How these new nodes will be placed in fringe will depend on a particular algorithm.

This is the basic search algorithm. You start from the initial state and then you expand its successors so that you can get new nodes. And when you select a node for expansion you must check whether it is a goal. If it is not a goal you expand it and get its successors. The successors are put into fringe which is potential nodes. Now we will discuss several search algorithms today. For each of these algorithms we will look at the following:

We like to measure how effective these algorithms are in terms of, number one, completeness, is this algorithm guaranteed to find a solution if a solution exists, given a problem given the state space representation of the problem if a solution exists to this problem can the algorithm find the solution or is the algorithm guaranteed to find the solution.

Secondly, we will evaluate whether the algorithm always results in an optimum solution. If a solution is found by this algorithm is the solution guaranteed to be the one that has minimum cost?
Thirdly we will look at the efficiency of the algorithm. Namely we will look at the time complexity of the algorithm which the time taken is measured by the number of nodes expanded either in the worst case or average case to find the solution. So we will look at the time complexity of the algorithm. We will also look at the space complexity of the algorithm. That is the memory required by the algorithm which we will measure in terms of the maximum size of fringe.

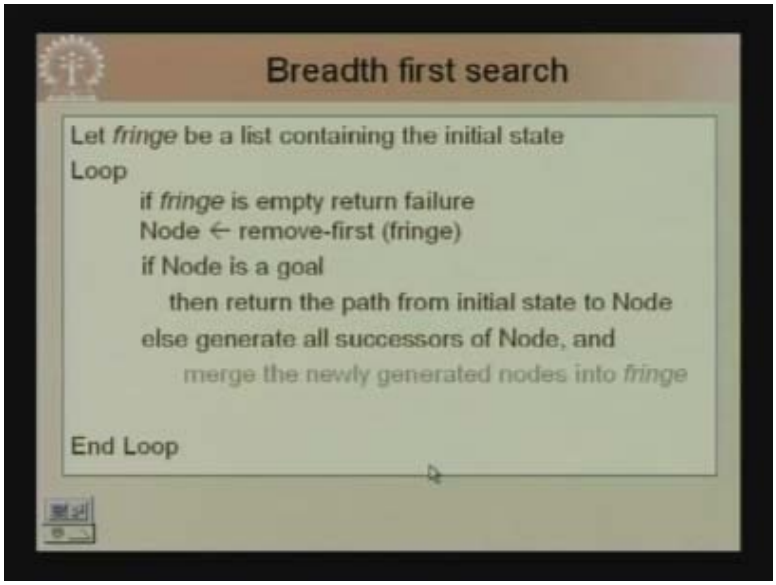
(Refer Slide Time 31:04)



Fringe is the list the algorithm must keep track of. How big can be the size of fringe in the worst case or average case, could be a measure of the efficiency of this algorithm or whether the algorithm is practical to all. Today we will look at several blind search strategies. Depth first search, breadth first search, iterative deepening search and bidirectional search.

In the subsequent classes we will look at informed search: Search using heuristic function and we will also look at constraint satisfaction problems which can be modeled as search problems and two person games or adversarial search. Now the first algorithm we will discuss is breadth first search. We will see how the basic search algorithm can be modified to give rise to the breadth first search algorithm. This is the search algorithm we discussed. And we now mentioned that, in order to have different strategies we need to change the way this step works. Merge the newly generated nodes into fringe.

(Refer Slide Time 32:27)



How to merge the newly generated successors into fringe will determine what sort of search strategies we get. So the strategy in breadth first search is to expand the shallowest node first. In order to achieve this, the merging of successors into fringe will take place like this. We will add the generated nodes to the back of fringe. Fringe is a list of nodes which are waiting to be expanded.

The new nodes which are generated will be put at the back of fringe. And we will see that this strategy results in the expansion of the shallowest nodes earlier than the deeper nodes. Suppose this is a very simple search space we will see how we will apply breadth first search on this search space. Earlier we saw that this search space gives rise to this search tree and breadth first search will actually search on this tree in order to find a goal.

Now this is the potential search tree that can be generated by search. Initially we have in fringe a single node A. This node is the only node in fringe. Now we will execute the loop of our algorithm which says that remove the first node from fringe. The first node is A so we will remove this node from fringe and get its successors. The successors happen to be B and C. So A will be removed from fringe and B and C will be incorporated at the back of fringe. There is nothing in fringe so B and C will be put at the fringe.

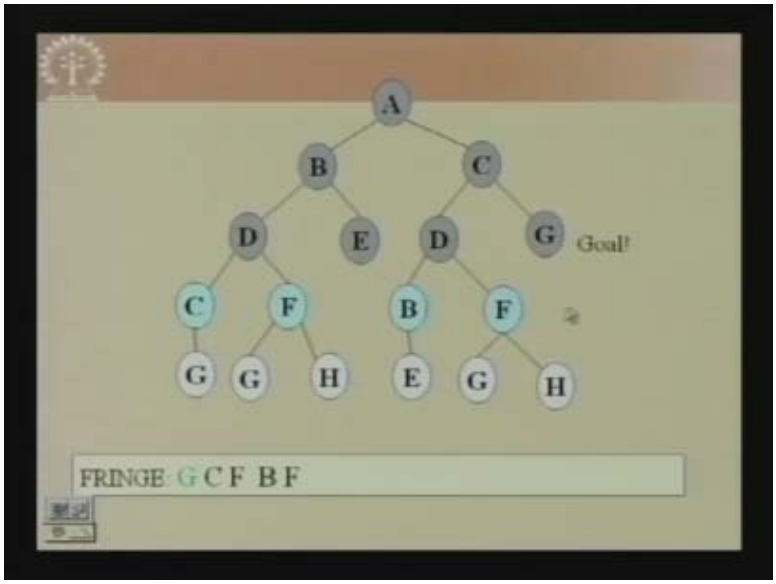
Therefore what should be the next step?

B has to be removed from fringe as B is the first node in fringe and its successors D and E will be generated. So B has to be removed from fringe and its successors D and E are generated. At the next state C is removed from fringe and its successors D and G are generated and put at the back of fringe.

In next step D will be selected for expansion so D will be removed from fringe and its successor C and F will be generated and put at the back of fringe. The next step would be, E has to be removed from fringe and its successors have to be put at the back of fringe, E has no successors so E is simply removed from fringe. So you notice here that this is the order in which nodes are expanded. First A is expanded, then B then C then D then D then E then F then G etc.

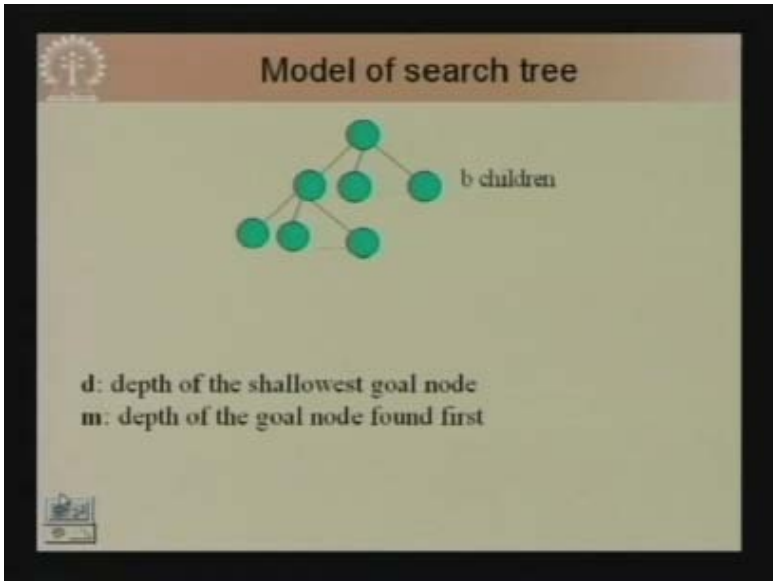
In breadth first search nodes are expanded in this order according to their level. First it will be A then B then C then D E D G. So nodes will be expanded in the order of their depths. E has been removed from fringe, then D will be removed from fringe and its successors B F will be added to the back of fringe and then G is the next node to be expanded and G happens to be a goal node so in this case the search will stop. We notice that first all nodes at level zero from depth zero from the start state get expanded that is A itself then all nodes which are at distance one from the start state then all nodes at distance two then all nodes at distance three and so on.

(Refer Slide Time 37:03)



Now we will analyze this algorithm and look at its properties whether a breadth first search is optimal whether it is complete and will also look at the time and space complexities so that we can discuss the time and space complexities of this algorithm. We must assume a particular structure of the state space.

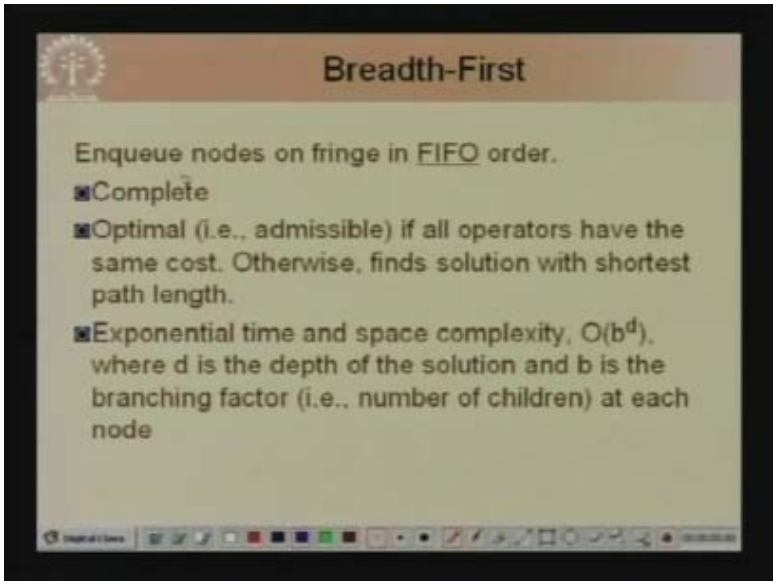
(Refer Slide Time 37:34)



Therefore what we will do is we will assume that in our state space we have a very uniform type of search tree. Having a uniform search tree will enable us to analyze the complexity. So let us assume that every node has utmost b children or exactly b children and let us assume that D is the depth of the shallowest goal node.

There can be many goal nodes but let us say that the shallowest goal node is at depth D and let m be the depth of this entire search tree. So m is the maximum depth of the search tree, D is the depth of the shallowest goal node and we have a uniform search tree where each node has exactly b children except the leaf nodes. So under this assumption let us look at the properties of breadth first node.

(Refer Slide Time 38:56)



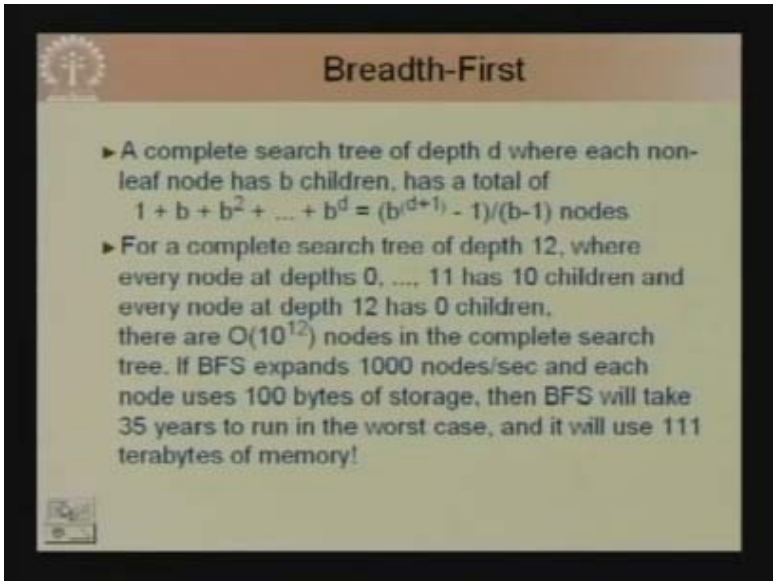
We have seen that in the breadth first node the generated nodes are put at the back of fringe. That is, new nodes are put on the fringe in first in first out order or FIFO order like a queue. So fringe is actually implemented as a queue. New nodes are put at the back of the queue and then node to be expanded is the first node in front of the queue. Firstly, breadth first search is complete. That is, if a solution exists breadth first search systemically expands zero level nodes first, then all one level node, then all two level nodes, all three level nodes and will eventually expand all D level nodes and will come to the goal node which is at level d. So breadth first search is a complete search algorithm.

Secondly, breadth first search is optimum. If all operators have same cost, you notice that breadth first search expands node according to their level. So a node at level D will be expanded before any node at a larger level is expanded. So breadth first search will expand that goal node first which is at the lowest level. So if our search cost is equal to the number of steps to the solution then the breadth first search is optimum. However, if operators have different costs then a goal node at depth 5 may have a cost of 10 whereas a goal node at depth 3 may have a cost of 23. So, in this case breadth first case may not be optimum. Therefore breadth first search is optimum if all operators have same cost. Otherwise breadth first search finds that solution whose path length is shortest but path cost is not necessarily the cheapest. And finally it is the time and space complexity.

If we go back to the previous slide, suppose this is the search tree and this is depth D and there is a goal here at depth D and G1 is at depth D then breadth first search will expand all these nodes up to depth D minus 1 and some of the nodes at depth D in order to get this solution. Therefore the number of nodes expanded by breadth first search is equal to order of b power d. And the number of nodes which are at fringe is also of the order of maximum number of nodes at a depth. We can see that the time and space complexities of breadth first search are both of the order of b power d where D is the depth of the solution and b is the branching factor at each node.

In breadth first search if we have a complete search tree of depth D where each non leaf node has b children then breadth first search will have a total of 1 plus b plus b square plus b power d.

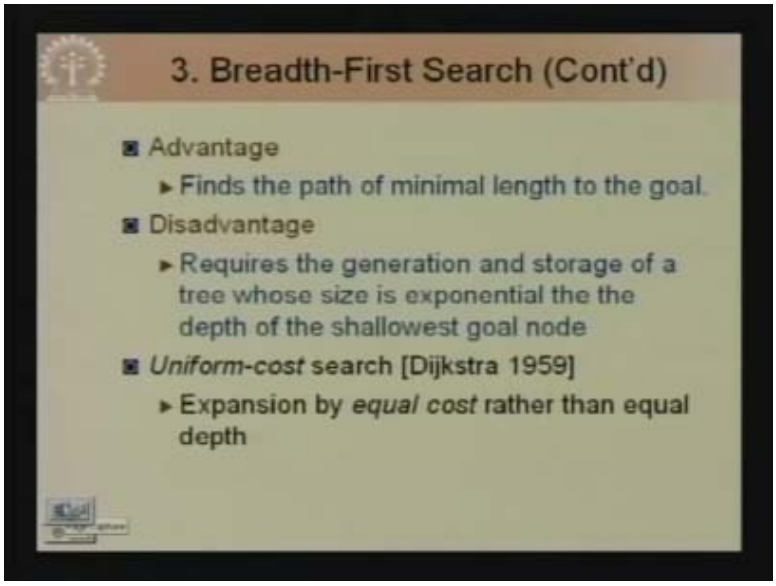
(Refer Slide Time 43:28)



So this is b power d plus 1 minus 1 by b minus 1 which is the order where b power d nodes will be expanded by breadth first search. For example, if we have a complete search tree whose depth is 12 and every node up to depth 11 has 10 children then the number of nodes expanded by breadth first search will be order of 10 power 12 that is 10 followed by twelve zeros.

To give you an idea of how big this number is, if we have a computer where 1000 nodes can be expanded per second and each node uses 100 bytes of storage then breadth first search will take 35 years to run in the worst case and it will use 111 terabytes of memory so this is a truly frightening figure. So the time and space complexities of breadth first search are exponential in terms of D which is the depth of the solution.

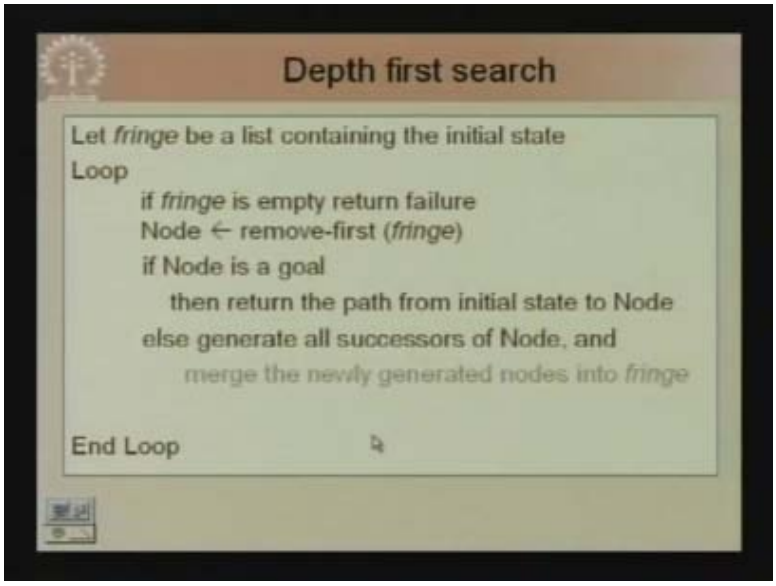
(Refer Slide Time 44:49)



The advantage of breadth first search is that it finds a path of minimal length to the goal. But one of the major disadvantages of breadth first search when we compare it to the next strategy depth first search is that, breadth first search requires the generation and storage of a tree whose size is exponential. So it requires exponential space to store the fringe and it also requires exponential time.

Later we will look at uniform cost search which is optimal in terms of giving the lowest cost solution. But today let us look at the other search strategies. Now we come to the next search strategy which is depth first search. This algorithm will be again a variation of the basic search strategy we outlined in the beginning where instead of merging the newly generated nodes into fringe we will give a particular strategy for merging. In breadth first search we put the generated nodes at the back of fringe.

(Refer Slide Time 46:04)

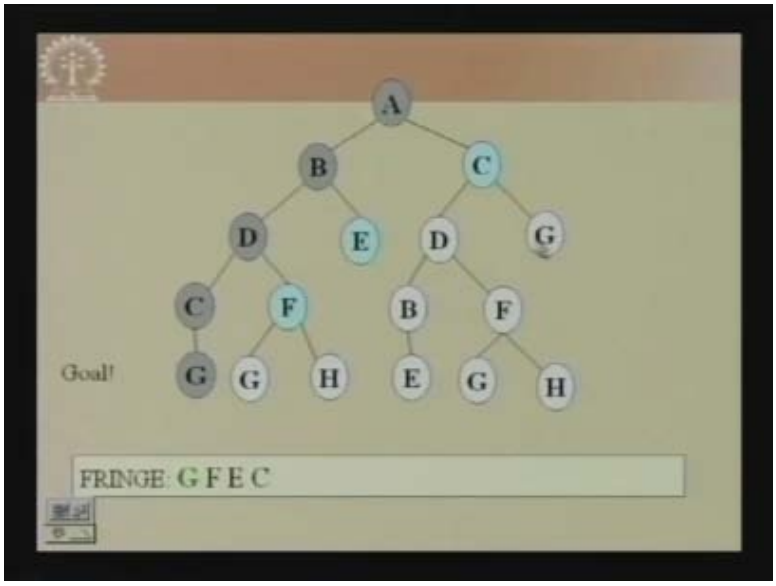


In depth first search our objective is to expand the deepest node first and to achieve this we add the generated nodes to the front of fringe. That is, fringe is maintained as a last in first out structure or a stack structure. The newly generated nodes are put at the front of fringe.

Now let us use the same search tree we analyzed earlier and trace depth first search through this search tree. Initially the fringe will contain the node A. In the next step A will be removed from fringe and its successors B and C will be generated and put in the beginning of fringe. Now B will be removed from fringe and its successors D and E will be put in the beginning of fringe.

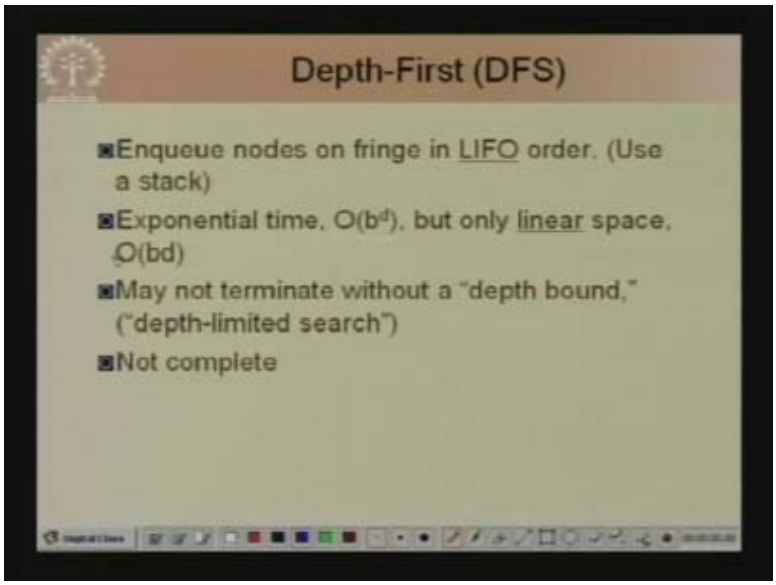
Next D will be removed from fringe and its successors C and F are generated and put in the front of fringe. Next C is removed from fringe and its successor G is put at the front of fringe. At the next level G will be expanded found to be a goal node so we terminate our search after having found this goal. So we notice that A B D C and G are the nodes expanded. A goal node at depth four has been found by depth first search. On the same search tree we saw that this goal was found as a result of breadth first search. This goal is at depth two and in that case we had expanded A B C D E and D.

(Refer Slide Time 47:55)



So let us analyze the characteristics of depth first search. In depth first search we en queue the nodes on fringe in a last in first out order by using a stack. Depth first search also requires exponential time ordered b power d to explore the entire search space. However, if you notice the size of the fringe is utmost the total and the maximum depth of the tree.

(Refer Slide Time 48:40)

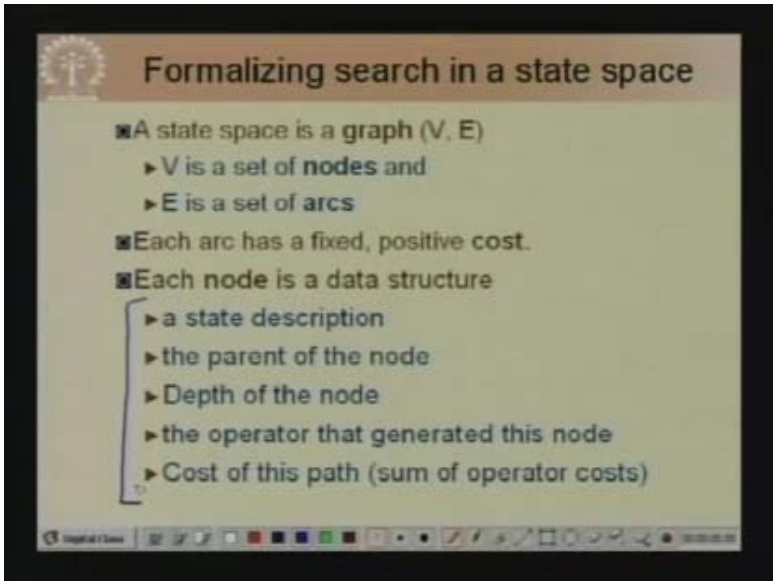


So the space required is actually of the order bm where m is the depth of the entire search tree. And the time required is order b power d. Depth first search may not terminate if the search tree is infinite. If the search tree is infinite depth first search may not terminate.

However, if it does terminate the time taken is equal to the size of the search tree if the search tree is finite at the worst case and the space taken is order of depth of the search tree. One advantage of depth first search is that it only takes linear space. However, the other characteristics are not that good in the sense that it is not complete. It may not be able to find the solution if the search tree is infinite and when it does find a solution the solution may not be an optimum solution.

Now to recapitulate let us look at the State Space Search problems. A state space is a graph b power d where V is a set of nodes E is the set of arcs. Each arc has a cost. And in order to run these algorithms we will use the data structure for every node. This is very important.

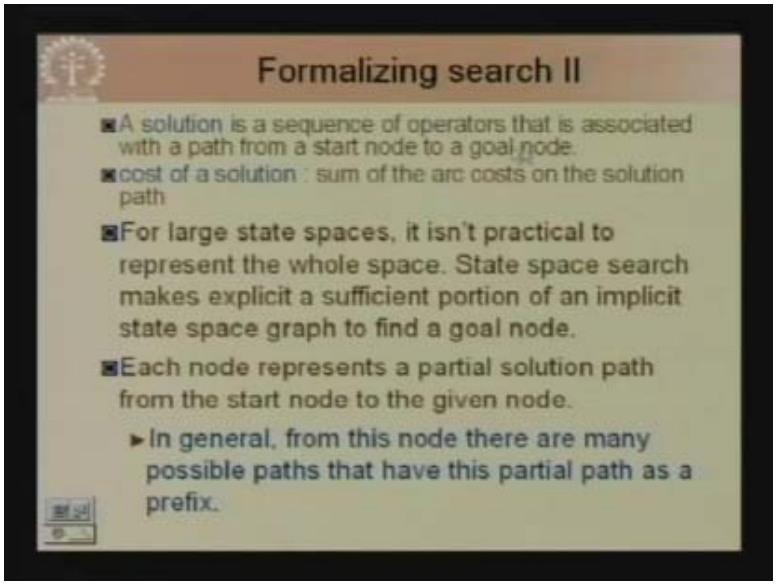
(Refer Slide Time 50:29)



Every node data structure will contain a description of the state the parent of the node. Keeping information about the parent is very important if we have to retrace the solution. So when we get a goal node in many cases we need to output the solution by which this state was obtained. So, to find the solution we need to keep track of the parent of each node.

For some algorithms we may need to keep track of the depth of the node. We also need to keep track of the operator that generated this node. If you want to find the final plan we need to find the operator that generated this node. And we need in some case to keep track of the cost of the path. Especially when we look at uniform cost search and best first search in subsequent lectures cost is important. But at least we need to keep track of the state description, the parent and the operator. And in some algorithms we need to keep track of the depth and sometimes the cost.

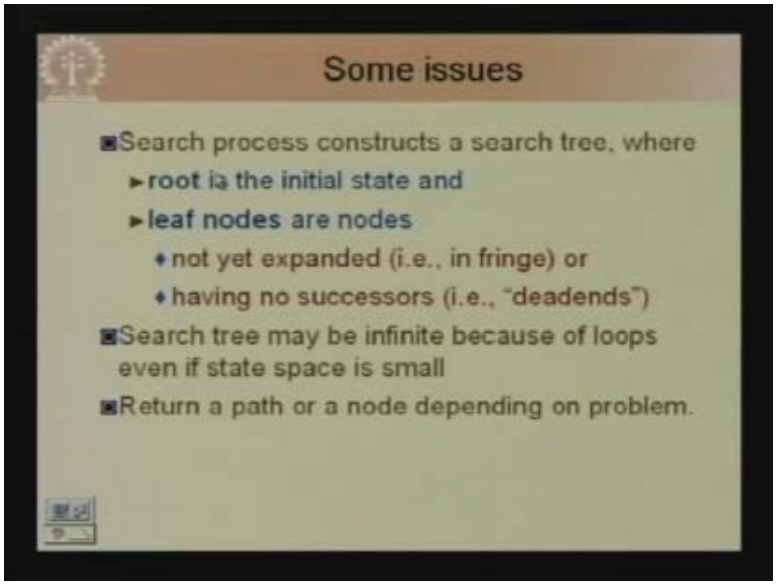
(Refer Slide Time 51:55)



A solution we have seen is a sequence of operators associated with the path from a start node to a goal node. For large state space it is not practical to represent the entire space. So the state space like the 15 puzzle problem we may not generate the entire state space and represent it as an explicit graph. Rather as we expand and generate nodes we will make explicit only a portion of the implicit state space graph. That is required for us to obtain the solution. We need not unfold the state space all at a time. So we can work with implicit state spaces and only make those portions explicit by our successor generator operations.

Now each node, because at every node we keep a link to the parent of the node and the operator that generated the node every node represents a partial solution path from the start to the given node. And from this node we can expand this partial solution to get may be one or more actual solutions.

(Refer Slide Time 53:22)

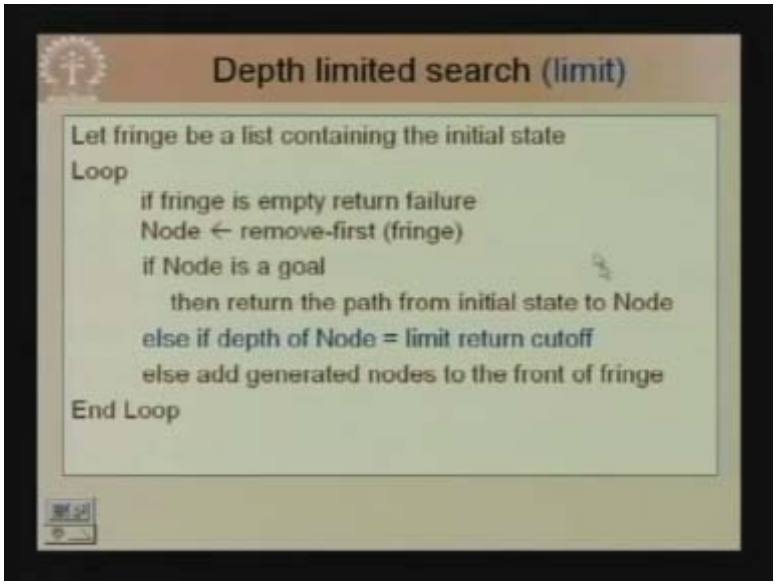


Our search process will construct a search tree where the root is the initial state and the leaf nodes are those nodes for this search tree that we currently have. The leaf nodes are the nodes which we keep at fringe. They are those nodes which are not yet expanded. That is which have been generated and are yet to be expanded or those nodes that have no successors that is they are dead ends. The search tree may become infinite even when the search space is finite because there could be loops in the state space. And in some cases our search algorithm needs to return the entire path and in some cases only the final state.

For example, in the 8 queens problem the final state is just the placement of the queens on the board. We need not give the solution path because given the state the solution path is evident. For the 8 puzzle problem the final state description does not carry any information as to how that state was reached. So we have to output the entire solution plan.

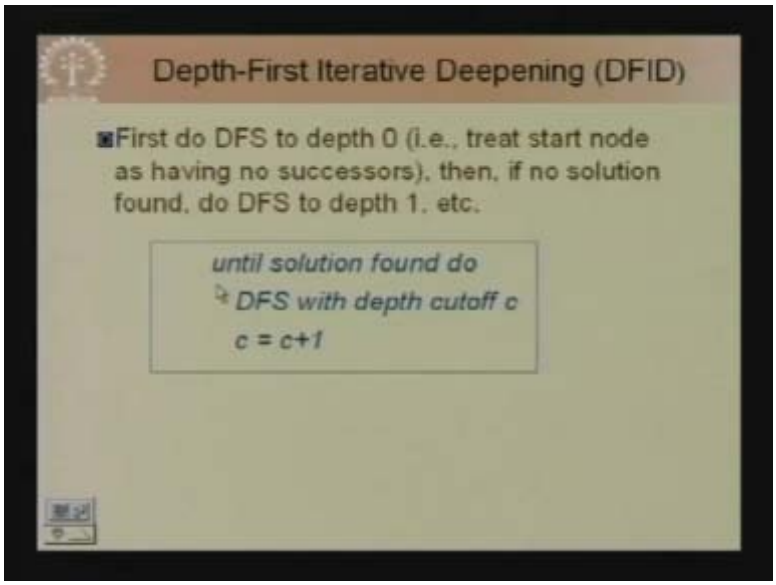
We mentioned that depth first search may not work in infinite state spaces where it may not reach a solution and go into an infinite loop. So we have a variation of depth first search which we call depth limited search where we cut off search at a particular depth. So depth limited search works like this:
At every node we keep track of the depth or level of that node and we modify depth first search so that if depth of node which we try to expand is equal to limit then we terminate the search.

(Refer Slide Time 55:24)



So if we modify depth first search by cutting off search at a limit we get depth limited search which takes as parameter a depth limit and does depth first search up to that limit. Now, if we choose a limit before hand a solution may not be found at that depth. Therefore we have a variation of depth first search which is called depth first iterative deepening search.

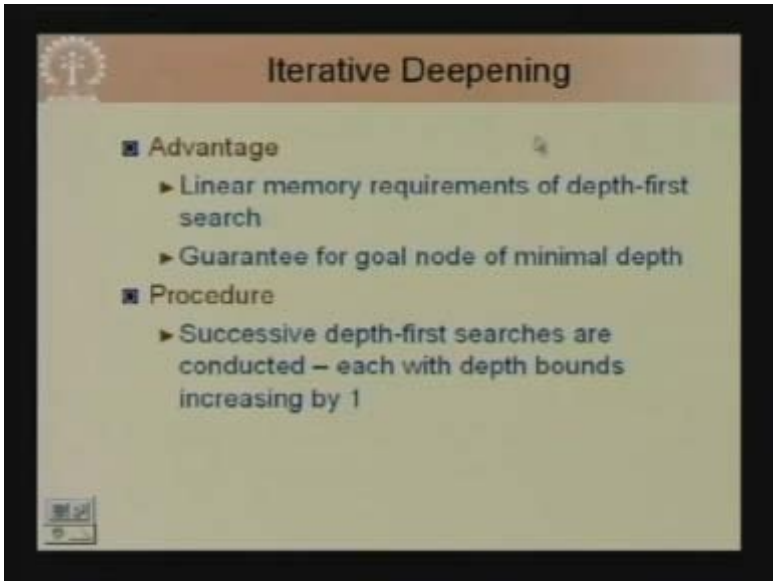
(Refer Slide Time 55:58)



The idea is that we do DFS up to a limit and if we do not find a solution we increase the limit by 1 and continue. So until solution found do DFS with depth cutoff C then put C is

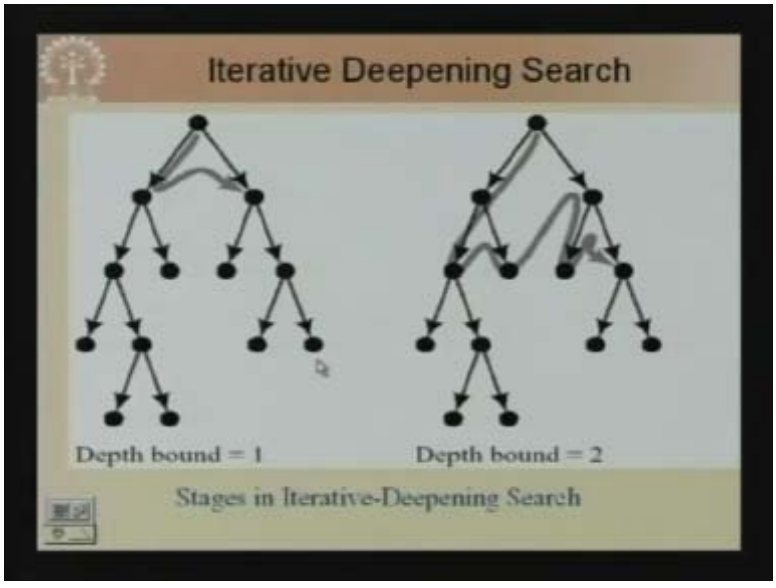
equal to C plus 1 and so on. So depth first iterative deepening is a complete search strategy where initially we set limit to 1 then to 2 then to 3 then to 4 and so on.

(Refer Slide Time 56:26)



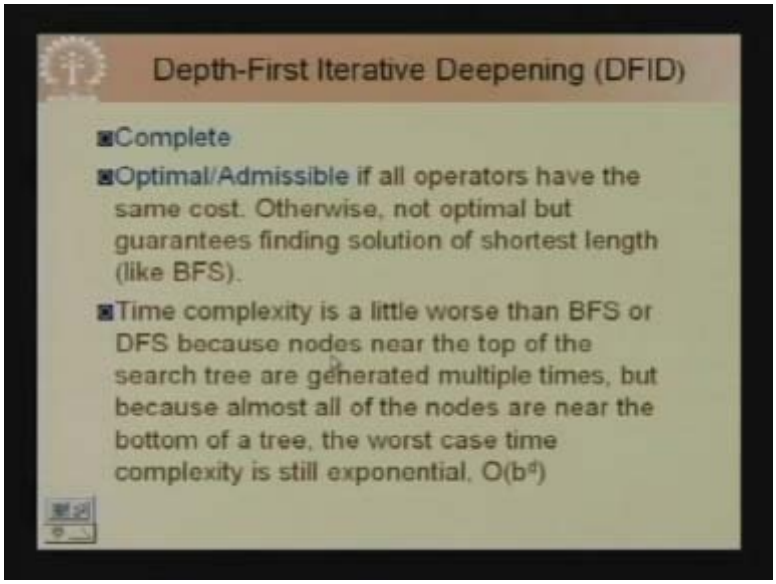
The advantage of iterative deepening search is that it has linear memory requirement because at every step we do a depth first search. And we start with limit equal to 1 then limit equal to 2 then limit equal to 3. Therefore it is guaranteed that when the algorithm finds a goal node it is a node of minimum depth.

(Refer Slide Time 56:52)



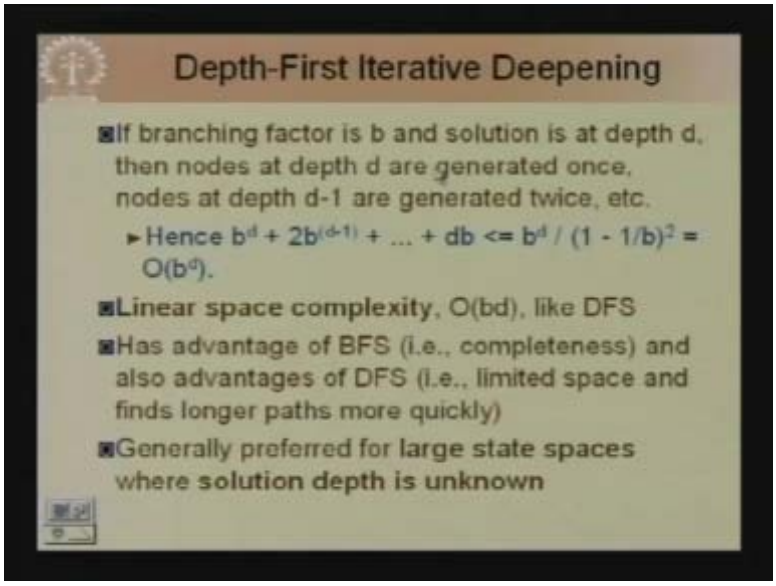
So let us illustrate this. This is a search tree. If we do iterative deepening search this is the first node to be expanded. Initially if limit is equal to 1 so we get this node then this node then this node. For the second depth limit with limit equal to 2 this node is expanded then this then this then this then this then this then this. Then we set limit equal to 3 for which we get this this this this this and so on. And then we set limit equal to 4 and we have a depth first search of the search tree with depth bound is equal to 4.

(Refer Slide Time 57:44)



So depth first iterative deepening is complete. It is optimum if all operators are of the same cost and its time complexity is a little worse than breadth first search because you notice that some nodes are expanded more than once they are repeated. However, the order of node expansion is almost similar to breadth first search.

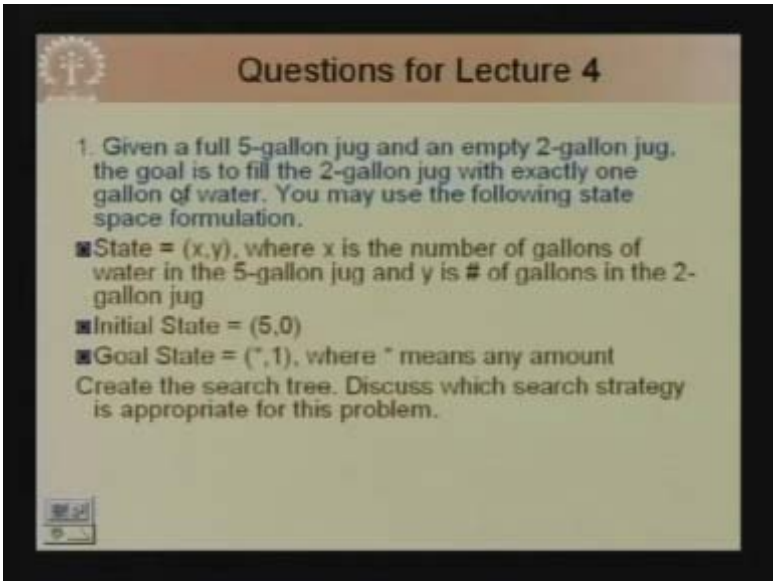
(Refer Slide Time 58:08)



So, if the branching factor is b and the solution is at depth D the nodes at depth D are generated once, nodes at depth D minus 1 are generated twice and so on. Therefore if this is the number of total nodes expanded by depth first iterative deepening then it is actually of the order of b power d. However, depth first iterative deepening search has linear space complexity unlike BFS and it is complete and it is usually the preferred algorithm for large state spaces. I come to the questions for today's lecture.

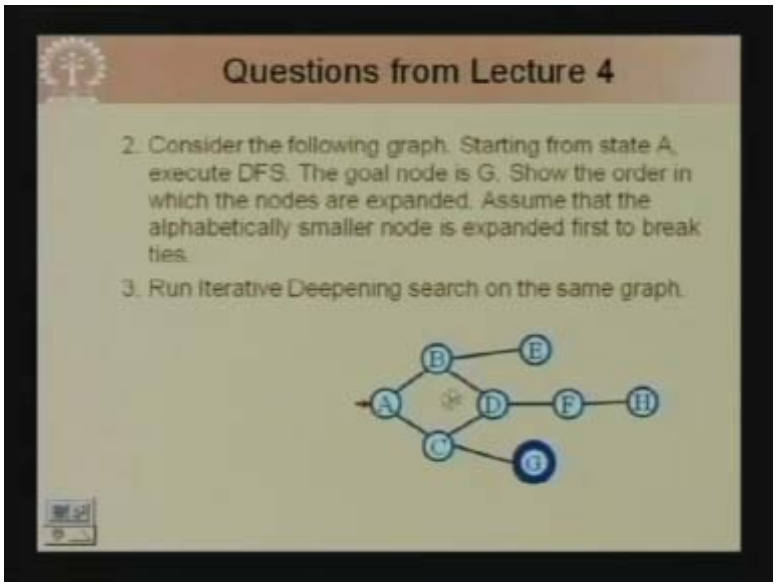
1) You are given a 5 gallon jug and a 2 gallon jug. Initially the 5 gallon jug is full and the 2 gallon jug is empty. Your goal is to fill the 2 gallon jug with exactly 1 gallon of water.

(Refer Slide Time 59:12)



Given this problem description your problem is to create the search tree and discuss which search strategy is appropriate for this problem.

(Refer Slide Time 59:26)



2) Consider the following graph.
Starting from state A you execute GFS. G is the goal node, you have to show the order in which the nodes are expanded assuming that the alphabetically smaller nodes are expanded earlier in case of ties. On the same graph you also run iterative deepening search. That is your problem 3.