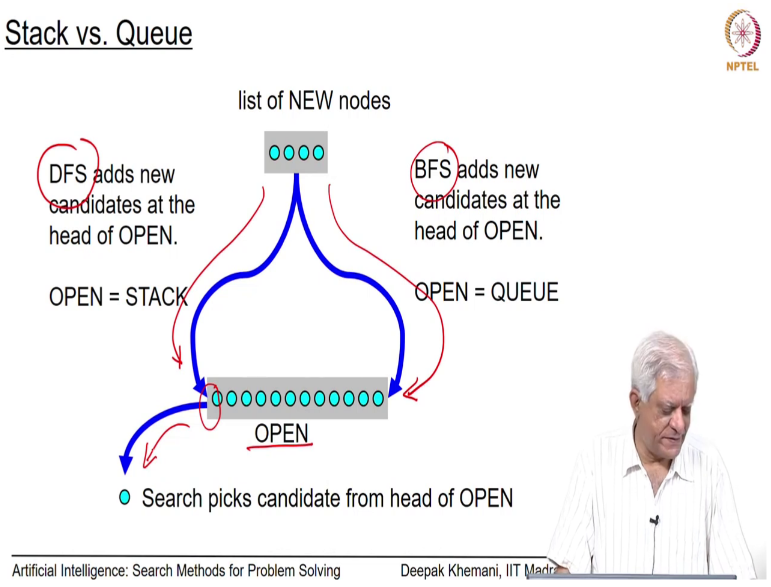
Artificial Intelligence: Search Methods for Problem Solving
Prof. Deepak Khemani
Department of Computer Science & Engineering
Indian Institute of Technology, Madras

Lecture – 21
Comparing DFS and BFS

(Refer Slide Time: 00:14)



Stack vs. Queue

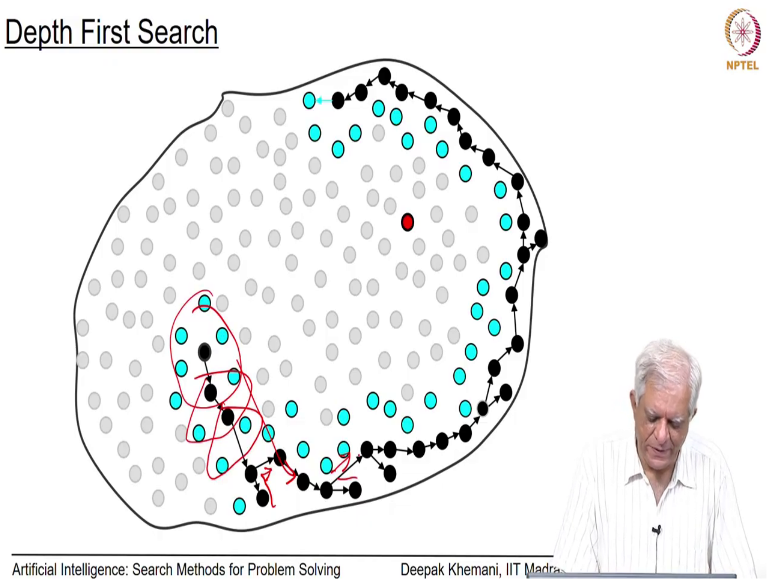
list of NEW nodes
DFS adds new candidates at the head of OPEN.
OPEN = STACK
BFS adds new candidates at the head of OPEN.
OPEN = QUEUE
OPEN
Search picks candidate from head of OPEN
NPTEL

Artificial Intelligence: Search Methods for Problem Solving Deepak Khemani, IIT Madras

So, welcome back. We have just finished looking at the algorithms depth first search and breadth first search and we observed that the basic difference between the 2 algorithms is the way in which they maintain the OPEN list. In the case of depth first search, the NEW nodes that are generated by the movegen function they are added at the head of the list and therefore, we treat open as a STACK. Whereas, in the case of breadth first search the new nodes that are generated are added at the tail of the list thereby treating that list as a queue essentially.

In either case we always pick the node for inspecting from the head of the queue. So, it now we have a uniformity about this and we have a deterministic algorithm. Let us compare the 2 algorithms depth first search and breadth first search in terms of their behavior. And what we mean by behavior is in what manner does the search algorithm explore the state space essentially.

(Refer Slide Time: 01:34)



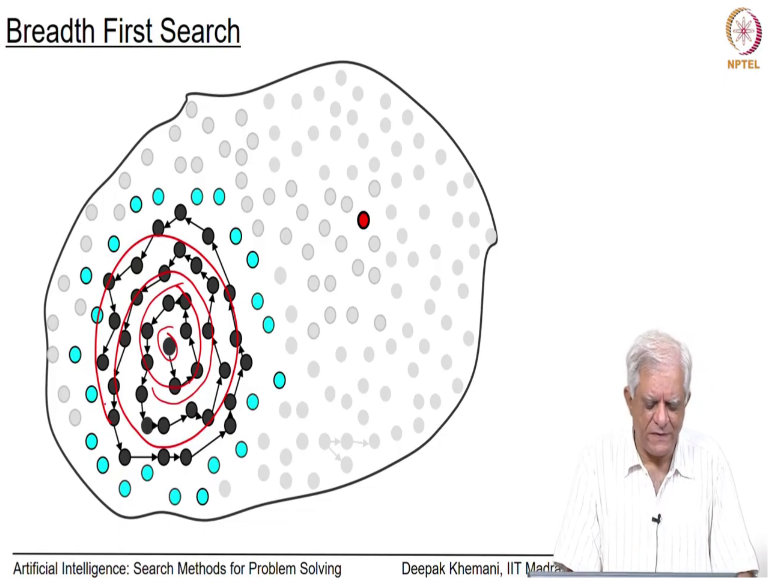
Depth First Search

Artificial Intelligence: Search Methods for Problem Solving   Deepak Khemani, IIT Madras

So, when you look at depth first search for example, we know that it goes to the deepest nodes first. So, once it has generated the first set of children and it is picked one of them and generated its children then it will pick one of the children of the first child and then it will pick one of the children of this second child and so on. So, it basically dives down into the search space till it hits a boundary. When it hits a boundary it backtracks and then try something else.

So, here also we can see it is backtracking and trying something else and so on. So, essentially this is the behavior of depth first search and it is interesting to observe that the behavior is not influenced by the goal node at all essentially it just happens the same every time you even depth first search on this particular domain.

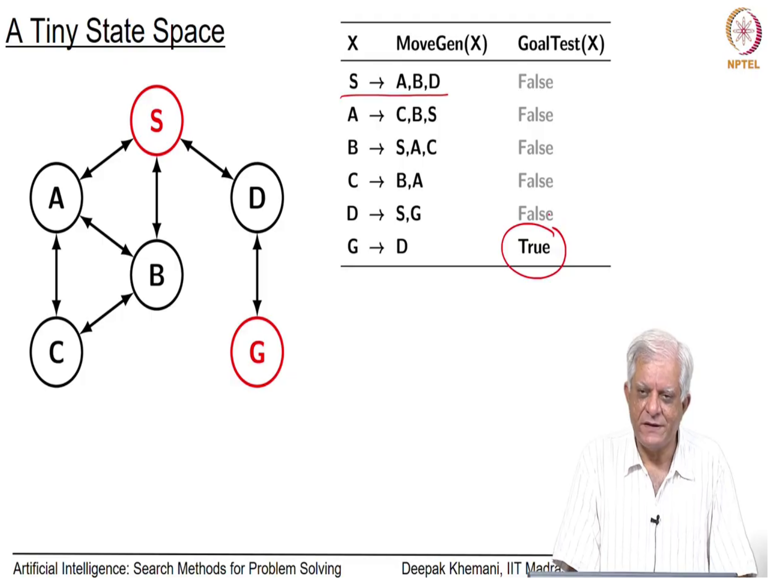
(Refer Slide Time: 02:31)


Breadth First Search
NPTEL
Artificial Intelligence: Search Methods for Problem Solving
Deepak Khemani, IIT Madras

The behavior of breadth first search is also always the same and that is why these 2 algorithms are called blind search algorithms or uninformed search algorithms in the sense that they are not influenced by where the goal is essentially.

Breadth first search as opposed to depth first search is a conservative algorithm. It tries to stay as close to the goal as possible. So, first it investigates a nodes which are closest to the goal, then it investigates a node in the next layer and then the next layer and the next layer and in that manner it only goes away from the goal when it needs to and as that direct consequence of this it always guarantees a shortest path essentially.

(Refer Slide Time: 03:18)



A Tiny State Space

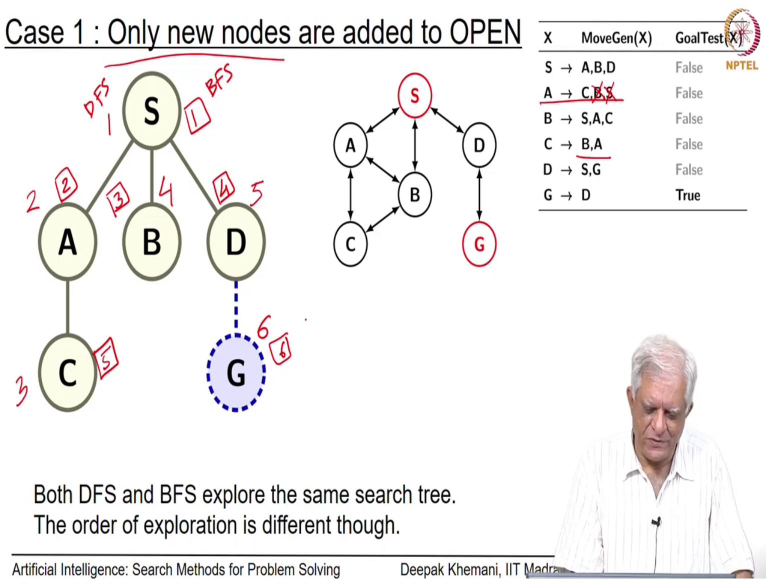
X	MoveGen(X)	GoalTest(X)
S →	A,B,D	False
A →	C,B,S	False
B →	S,A,C	False
C →	B,A	False
D →	S,G	False
G →	D	True

Artificial Intelligence: Search Methods for Problem Solving   Deepak Khemani, IIT Madra

Let us look at an example with a tiny search space here. The MoveGen function is given in the table here. So, for example, S generates A, B, D and A generates C, B, S and so on. So, the importance of giving this MoveGen function is that we know in which order they will be generated and therefore, we can decide which order they will be inspected in.

The corresponding search space or state space is drawn on the left hand side and we have marked S as a start node and G as a goal node and this last column of the table also tells you that it is G which is the goal node because for everybody else the goal test function returns false. So, our algorithm will terminate in 2 conditions either the open list will become empty or it will find the goal node and it will know that it is a goal node by applying voltage function to the node that it is looking at essentially.

(Refer Slide Time: 04:22)



Case 1 : Only new nodes are added to OPEN

X	MoveGen(X)	GoalTest(X)
S	→ A,B,D	False
A	→ C,~~B,S~~	False
B	→ S,A,C	False
C	→ B,A	False
D	→ S,G	False
G	→ D	True

Both DFS and BFS explore the same search tree.
The order of exploration is different though.

Artificial Intelligence: Search Methods for Problem Solving Deepak Khemani, IIT Madras

So, let us look at the few cases here. We have seen that we try to remove duplicates from open in the sense that if we have generated some node you do not generate it again and when you say generated you mean it is been put on open and also if you have see a node before we do not add it again to open.

So, in this first case where only new nodes are added to open the behavior on this small problem of both depth first search and breadth first search is same. It will differ on larger problems, but we just want to understand the fact that how the space generated by the algorithm can vary from algorithm to algorithm.

Now, in this particular case it so happens that both generate the same search space which is shown on the left hand side, but they will inspect the nodes in different order. So, if you look at depth first search for example, it will start both will start looking at one and they will generate the children of S which are given by the function A, B and D then depth first search will look at this node 2 and then generate the child of 2 which is C.

Now, we can see that there are 3 children of a C, B and S. It has not added S because it is already on the closed and it is not added B because it is already on the open. So, these 2 nodes are not added by this algorithm and only C is added as the child and because that is a deepest node depth first search goes an inspects this next.

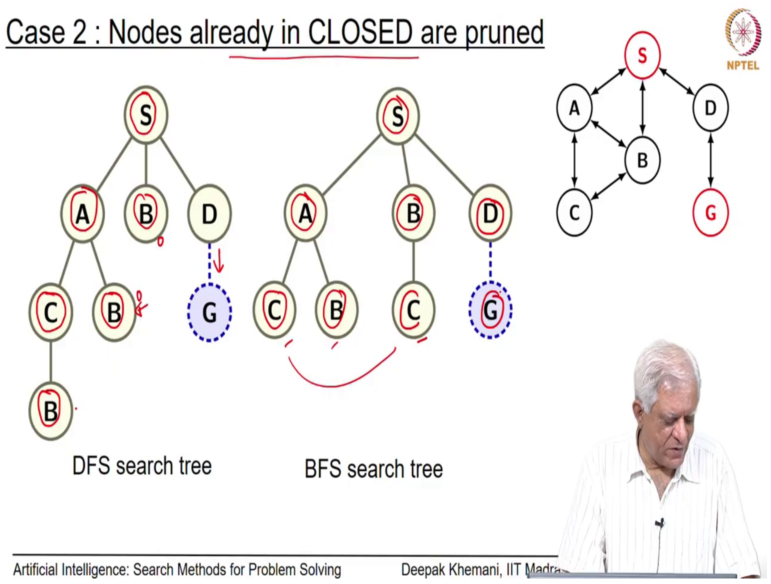
And after that once this is inspected it has no more children to it does not generate any more children because the children of C which are B and A are already been either added to open in the case of B or in close in the case of A. So, it has no further children.

So, in some sense it goes back to open and in open it has got B sitting there. So, it looks at the node B and you can think of this behavior has going down to S and reaching a boundary going to down to C sorry and reaching a boundary and then backtracking and going to B then again reaching a boundary then backtracking and going to D and finally, it will find a paths to G. Breadth first search also starts off by looking at the node S.

So, let us put this in square boxes just to distinguish from depth first search. So, this is for BFS and this one is was for DFS. So, here also we generate the 3 children A, B, D of S and here also breadth first search looks at the first child which is in the open which is A. But, it also generate C as a child of a like depth first search did, but it adds C to the end of the queue. So, as a consequence of this the next node that this will see is B and because B have no more children the next node it will see is D then it will see node C and finally, it will see node 6.

So, you can see that the behavior of the 2 algorithms are different, but they generated in this particular example the same state space. Now, this is a smallest search space you can generate because your pruning everything that you have either seen before or you have added to open before. So, let us look at some variations here.

(Refer Slide Time: 08:14)


Case 2 : Nodes already in CLOSED are pruned
S
A
B
D
C
B
G
B
DFS search tree
S
A
B
D
C
B
C
G
BFS search tree
S
A
D
B
C
G
NPTEL
Artificial Intelligence: Search Methods for Problem Solving
Deepak Khemani, IIT Madras

In case 2, we only look at nodes which are already in closed and we do not had them again. The state space is still the same, but as we can see here the space is the search space is generated by the 2 algorithms are a little bit different.

So, let us just follow the progress of these 2 algorithms and we will put a double circle to stand for the fact that it has been put in closed. So, first of course, we inspect node S and generate its children A, B, D and put them in closed then we look at that node A and this is true in both the cases depth first search and breadth first search, but here onwards the algorithm differs.

When we generate the children of A this time we have said that only for node is on CLOSED we will not add it again. So, in this case S is on CLOSED. So, of the 3 children of A which are S, B and C we do not add S, but we add C and B. So, you will notice here that they are duplicate copies of B sitting here.

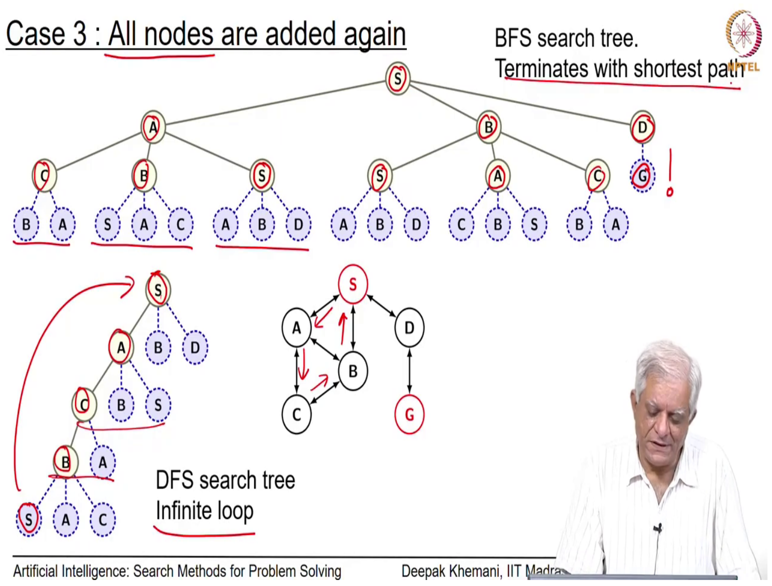
We added this B because it is on open, but we have said that we will allow that to be added again and even though those both these instances of B are on open we still add the third instance of B as open and then when we put C on CLOSED and then we go to B and then we inspect B.

Now, as you can see you will end up inspecting B 3 times in this algorithm before you come to D and finally, path of find the paths to the goal node essentially. Let us look at breadth first search as usual it generates S as starts by looking at S generates the 3 children A, B, D looks at a generates the children C and B again it is generated B again because even though B is present in open because our criteria is it only if it is on close we will not add it again.

But after looking at A it looks at B and again add C and you can see that C is here and C is here as well. So, we have 2 copies of C then it goes and looks at D and then it looks at this C then B then C and finally, at G essentially.

But, as you can see from this diagram the search space is generated by the 2 algorithms are different. So, let us look at the third case in the third case we do not we just indiscriminately add everything that comes over B. So, whatever the MoveGen function returns to us we add that as a children of the given node and consequently as we can imagine the search space is generated a pretty large essentially.

(Refer Slide Time: 11:02)



Case 3 : All nodes are added again

BFS search tree.
Terminates with shortest path

DFS search tree
Infinite loop

Artificial Intelligence: Search Methods for Problem Solving   Deepak Khemani, IIT Madras

So, here we are saying that we will add all the nodes again and again. So, as you can see for in the case of depth first search we at every stage we add the same nodes. For S we add A, B and D because 3 children of S for a we add C, B and S for C we add B and A because these are the 2 children of C and again for B we add C, S, S, A, C again because these are the 3 children of B. But, now you will notice that after having seen S and A and C and B the algorithm is going to look at S again.

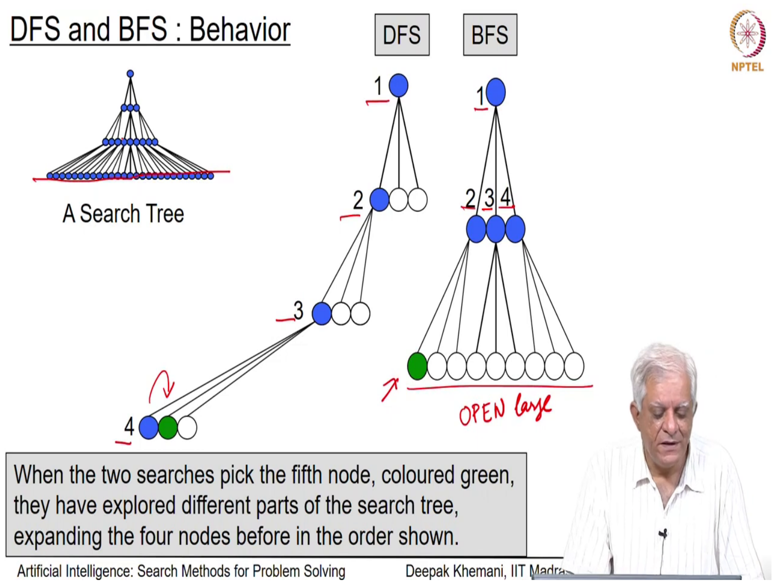
Now, clearly you can see that it is in the same situation that it started. It is gone back to square one. So, to speak and in some sense it is gone back there. So, everything that it did from the start node S it will repeat here and consequently this algorithm goes into an infinite loop.

It never stops coming out of this loop and the loop that it is going through is from S to A to C to B from S to A and A to C, C to B and B to S. So, this is a loop that it is constantly finding itself in. So, that is a danger with depth first search algorithm. If the space search space is infinite it can get caught into a loop.

The search space for breadth first search also is infinite here because we have put no constraints we had all the children, but the difference is that in the tree that we see here we start by looking at S then because breadth first search goes on level by level we look at A, we look at B, we look at D, then we go down to the next level. By the time we have finishing looking at A, B and D we have generated its children C, B, S, S, A, C S, A, C and G and then when we look at C we add these 2 children A, B when you look at B we add this 3 children A, B.

So, we are doing the same thing we are adding the nodes again and again and again without any constraint. So, we, but the difference is that by the time we have finished with S, A and C we can reach the goal and we can terminate here essentially. So, as you can see even though the search space is infinite breadth first search terminates and not only terminates it terminates with the shortest path. So, that is one plus point as we had observed about the breadth first search algorithm.

(Refer Slide Time: 13:41)

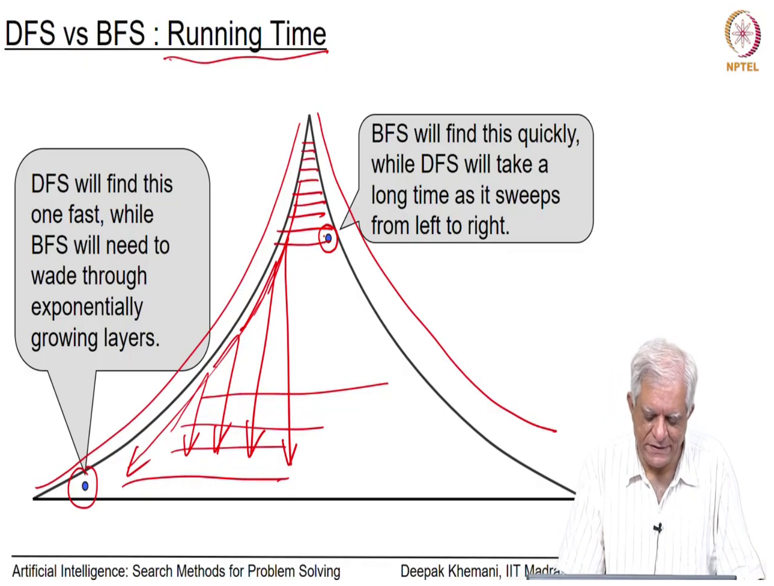

DFS and BFS : Behavior

When the two searches pick the fifth node, coloured green, they have explored different parts of the search tree, expanding the four nodes before in the order shown.

So, in general if you look at the behavior of depth first search and breadth first search if you have a small search tree of depth 3 and branching factor 3 as well then by the time you are looking at the fifth node. So, the order is given here depth first search looks at the root first then the left child then the left child of the left child then the left child of the left child of the left child and then it reach reaches the dead end which is at level 3.

So, in some sense it backtracks and goes and looks at this node next. So, the fifth node that it visits is this green node here and the nodes that you see which are blank or uncolored these are the nodes which are kept in the open. Breadth first search as we have already said goes on level by level it looks at this node 1 then 2 then 3 then 4 by the time it is finished generating 2 3 and 4 it is generated all the children of these 3 nodes and as you can see the size of open is large. In fact, we have observed that it is going to be exponential in nature.

(Refer Slide Time: 15:09)



DFS vs BFS : Running Time

DFS will find this one fast, while BFS will need to wade through exponentially growing layers.
BFS will find this quickly, while DFS will take a long time as it sweeps from left to right.
NPTEL
Artificial Intelligence: Search Methods for Problem Solving
Deepak Khemani, IIT Madra

So, this again depicts the same thing that we were talking about the different behavior of the 2 search algorithms as they go into a search space. The search space in this case is a small tree with branching factor 3 and depth 3. What about the running time of these 2 algorithms? Well, that really depends upon the location of the goal node.

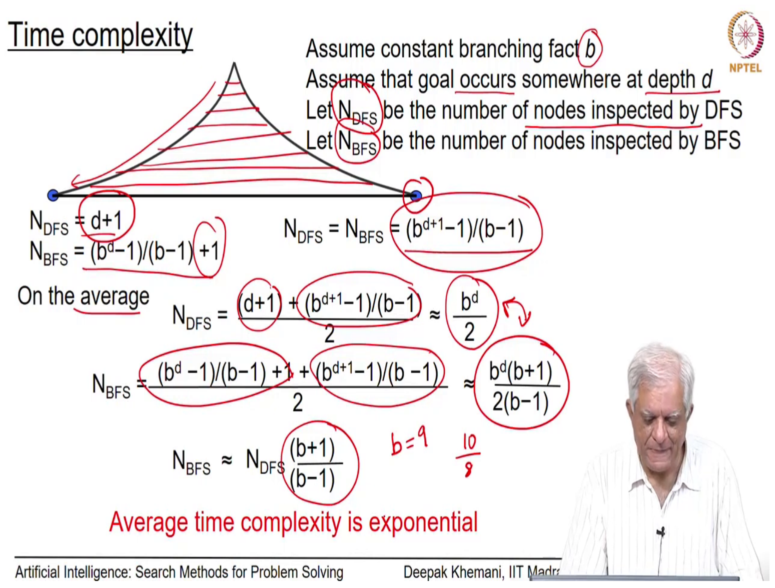
Now, if this if diagram if this diagram represents the search tree and I try to draw search tree is like this trying to kind of hint towards the fact that they are going exponentially the of course, this is not really an exponential function, but anyway I hope it gives an idea.

Now, if the goal node happens to be here because depth first search dives into the search tree at some early point it will find this goal and it will terminate much faster whereas, breadth first search would have gone down slowly level by level eventually coming here and here and here and here and then finding this goal node.

So, if the goal node happens to be along the path that depth first dives into it will terminate faster, but if the goal node is for example, on the right hand side as shown here then depth first search will go down this and then backtrack and go down here then backtrack and go down here and backtrack and go down here backtrack and go down here.

So, it will take a considerable amount of time by the time it finishes sweeping the search tree whereas, breadth first search would have gone down layer by layer and found the goal node faster essentially. It is really depends upon where the goal node is in the search space.

(Refer Slide Time: 16:38)



Time complexity

Assume constant branching fact b

Assume that goal occurs somewhere at depth d

Let N_{DFS} be the number of nodes inspected by DFS

Let N_{BFS} be the number of nodes inspected by BFS

$N_{DFS} = d+1$

$N_{BFS} = (b^d-1)/(b-1) + 1$

$N_{DFS} = N_{BFS} = (b^{d+1}-1)/(b-1)$

On the average

$$N_{DFS} = \frac{(d+1) + (b^{d+1}-1)/(b-1)}{2} \approx \frac{b^d}{2}$$

$$N_{BFS} = \frac{(b^d-1)/(b-1) + 1 + (b^{d+1}-1)/(b-1)}{2} \approx \frac{b^d(b+1)}{2(b-1)}$$

$$N_{BFS} \approx N_{DFS}\frac{(b+1)}{(b-1)}$$

$b=9$ $\frac{10}{8}$

Average time complexity is exponential

Artificial Intelligence: Search Methods for Problem Solving Deepak Khemani, IIT Madr

Now, let us talk about complexity of the 2 algorithms. We assume a constant branching factor B and we assume for simplicity that the goal node occurs at depths D. So, this figure on the left shows that this is the search space generated and the goal node occurs that the last layer and let us for the sake of calculation assume that it either occurs at the left hand of this tree or at the right hand of this tree.

So, the time complexity is measured by the number of nodes inspected and so for depth first search we will call this number N DFS for breadth first search we will call this number N BFS and let us see what these numbers look like. If node if the goal happens to be in the extreme left, then depth first search will not need to backtrack at all and in its first attempt itself as it goes down this branch it will find the thing after looking at d plus 1 node. So, d plus 1 because the root is the first node it inspects.

Breadth first search on the other hand would have to completely search the tree up to the previous layer. So, it would have started like this. So, it has seen all these nodes and you can verify that the number of nodes which is going exponentially is given by this expression b raised to d minus 1 upon b minus 1 plus 1 when doing complexity analysis we may tend to ignore this small constants which occur on the way.

If the goal happens to be here then both of them will end ups inspecting the complete search space search tree and inspecting these many number of nodes which is b raised to d plus 1 minus 1 divided by b minus 1.

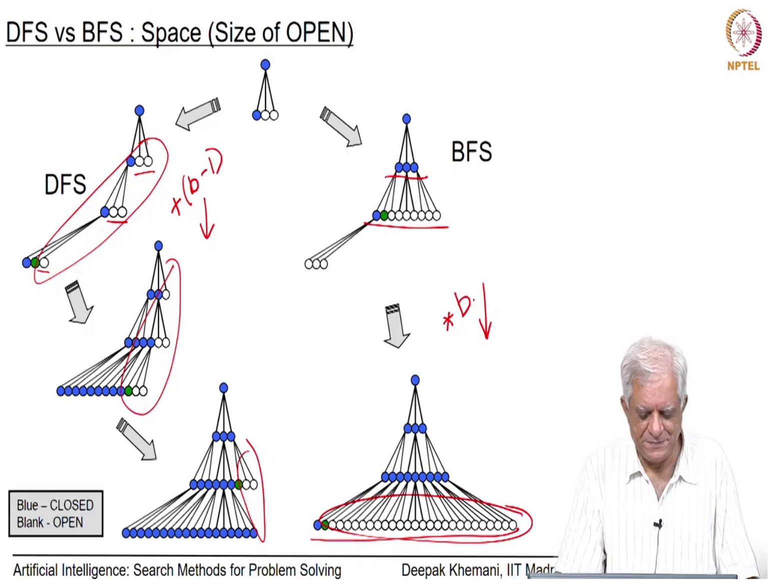
So, you want to see what is this average that they will see. So, we will assume that the average is the average of these 2 values. So, this value for depth first search and these 2 values which is which are given here and here and you can see that after you ignore small constants these comes to around b raised to d divided by 2. Likewise for breadth first search it is the average of these 2 values the one on the left which these heads written there and the one on the right which is here and this once you ignore the small constants it comes to about this much.

So, if you look at the ratio of these 2 values that tells you what is the difference between the number of nodes seen by depth first search and breadth first search and this ratio as we can see from this here turns out to be b plus 1 divided by b minus 1.

There are 2 observations to be made here that on the average time complexity is going to be exponential for either of these 2 algorithms and the number of nodes inspected by breadth first search in only slightly more on the average than the number of nodes inspected by depth first search and it is just b plus 1 divided by b minus 1.

So, for example, if b were to be 9 then it would be 10 divided by 8 essentially which is about 25 percent node nodes that breadth first search has seen and if branching factor is larger this fraction will become closer to 1 essentially.

(Refer Slide Time: 20:14)



DFS vs BFS : Space (Size of OPEN)
NPTEL
DFS
BFS
x(b-1)
*b
Blue – CLOSED
Blank - OPEN
Artificial Intelligence: Search Methods for Problem Solving
Deepak Khemani, IIT Madr

What about the size of the open list because that is one criteria that we used for space complexity. And we will look at this algorithm and as you can see the open nodes are depicted by the blank nodes or the uncolored nodes and the blue nodes are the closed nodes.

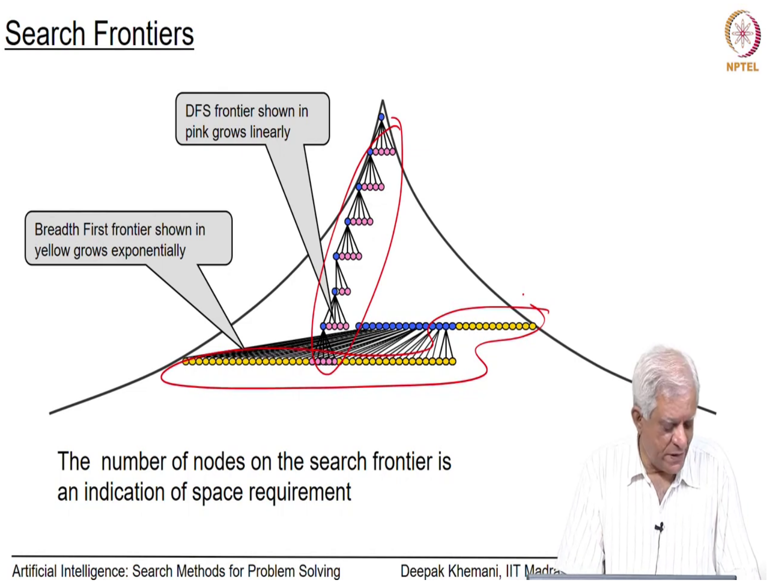
And when depth first search proceeds it is generates as it sweeps from left to right you can see that the number of open nodes that it maintains is kind of not growing very rapidly. In fact, it is always a linear function with respect to depth because in this small example tiny example we are adding 2 nodes at every layer essentially and after we have finish adding some of seen some of them it becomes less than 2.

So, at every layer we are adding b minus 1 nodes where b is the branching factor and therefore, it as you can imagine if the search space grows linearly. Breadth first search on the other hand finishes looking at a layer generating all it is children and then when it goes to the next layer it is generated all it is children.

So, you can see that the size of OPEN is growing quite rapidly and in fact you are multiplying if you use this as a symbol for multiplying then you are multiplying by b as you go down essentially.

In the case of depth first search, you were adding a constant amount as we went down and therefore, the search space was growing linearly. In the case of depth first search, you are multiplying by a constant amount and therefore, it is growing exponentially ok.

(Refer Slide Time: 21:56)



Search Frontiers

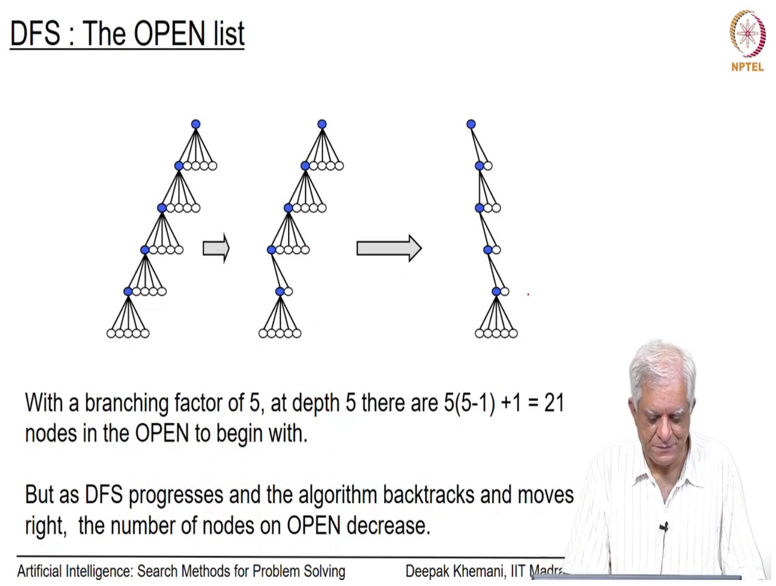
DFS frontier shown in pink grows linearly
Breadth First frontier shown in yellow grows exponentially

The number of nodes on the search frontier is an indication of space requirement

Artificial Intelligence: Search Methods for Problem Solving Deepak Khemani, IIT Madra

So, this is how the search frontiers of the tool look. The pink nodes that you see here is the search frontier for the breadth first depth first search and this yellow nodes that you see here is the search frontier for the breadth first search. And as you can see the pink nodes are going linearly you are adding a constant number at every level and the yellow nodes are growing exponentially.

(Refer Slide Time: 22:19)



DFS : The OPEN list

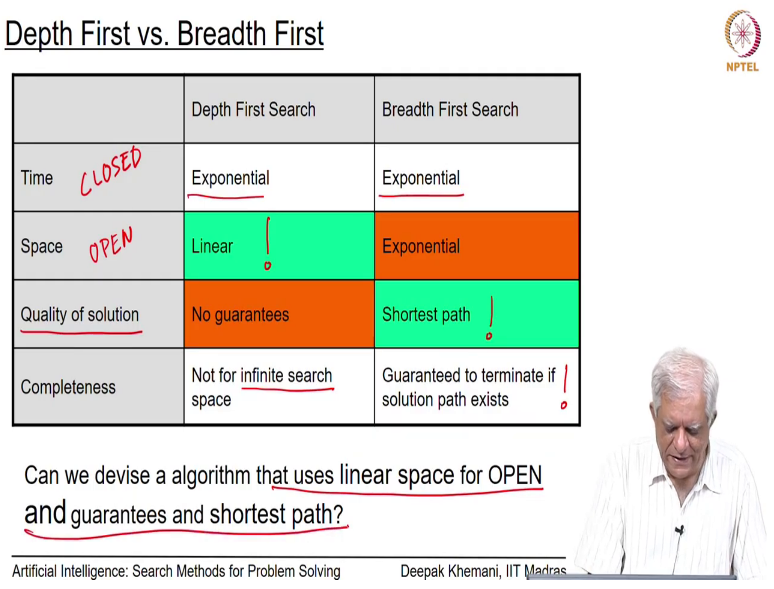
With a branching factor of 5, at depth 5 there are 5(5-1) +1 = 21 nodes in the OPEN to begin with.

But as DFS progresses and the algorithm backtracks and moves right, the number of nodes on OPEN decrease.

Artificial Intelligence: Search Methods for Problem Solving Deepak Khemani, IIT Madr

As thus depth first search algorithm sweeps from left to right the number of open nodes keep decreasing because it has seen all of them essentially.

(Refer Slide Time: 22:31)



Depth First vs. Breadth First

	Depth First Search	Breadth First Search
Time CLOSED	Exponential	Exponential
Space OPEN	Linear !	Exponential
Quality of solution	No guarantees	Shortest path !
Completeness	Not for infinite search space	Guaranteed to terminate if solution path exists !

Can we devise a algorithm that uses linear space for OPEN and guarantees and shortest path?

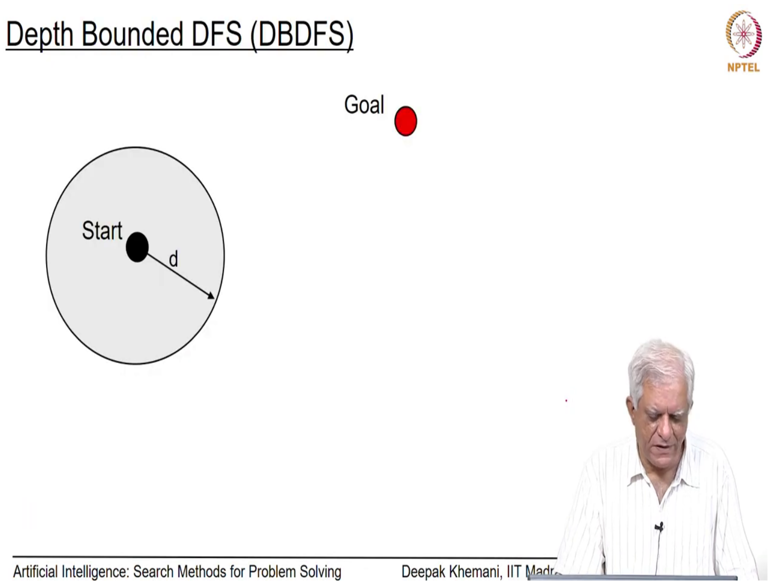
Artificial Intelligence: Search Methods for Problem Solving Deepak Khemani, IIT Madras

So, if you look at these 2 properties of the 2 algorithms then you can see that for both of them time complexity is exponential in nature. If you look at space complexity which is the size of open time complexity can be seen think thought of as the size of closed.

Because those are the nodes that you have seen and space complexity is a set of candidates that you are keeping which is the size of open and you can see here that depth first search wins on space complexity. But, breadth first search wins on the quality of the solution because we have already observed that it will give you the shortest path.

In terms of completeness for finite search spaces both are complete in the sense both will end up exploring the entire space, but if you have infinite search space we have seen that depth first search can go into an infinite loop essentially whereas, again this is a plus point for depth first search then it is guaranteed to terminate if there is a paths to the goal node because the path will always be of a finite length and depth first search breadth first search goes down level by layer exploring parts of greater length.

(Refer Slide Time: 23:49)



Depth Bounded DFS (DBDFS)

Artificial Intelligence: Search Methods for Problem Solving Deepak Khemani, IIT Madr

So, can we have an algorithm that uses linear space for open and which guarantees the shortest path.