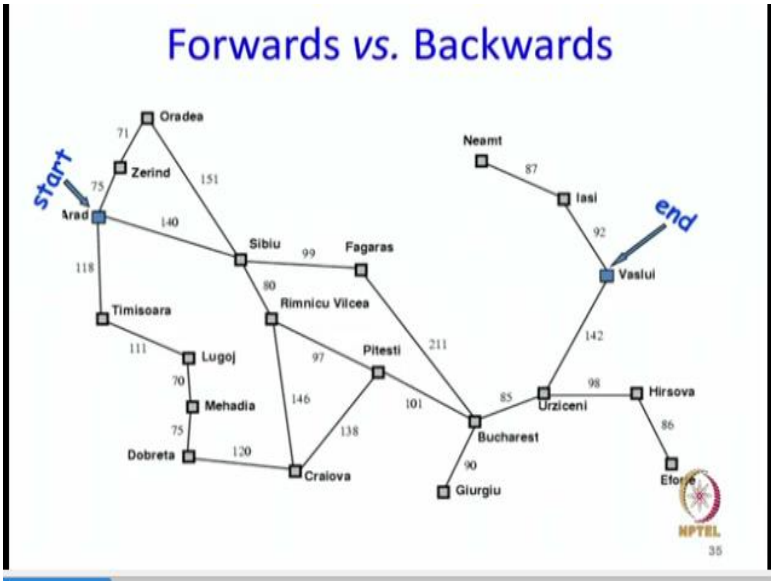
Artificial Intelligence
Prof. Mausam
Department of Computer Science and Engineering
Indian Institute of Technology - Delhi

Lecture – 14
Uninformed Search: Bidirectional Search (Part – 5)

Now, there is one other very interesting algorithm that we should study and the intuition for that algorithm is very simple right now, I am starting from the start state and trying to find a goal. Alternatively, I could start from goal state and try to find the start, right.

(Refer Slide Time: 00:45)



So, one search, you are going to call it a forward search, it is a search in the forward direction whereas the other search, we will call backward search, we will do search in backward direction, right. If my edges are directed we can still do it, we are looking for the predecessor where that action could be taken to reach me, right that is a backward search and this idea is going to stay with you.

So, even when we do logic, we can talk about forward chaining and backward chaining, there are many, many such situations where sometimes you go from start to the gaol and sometimes we go from goal to the start in fact, philosophically also, it is an important thing to consider. So, suppose I have given a problem, some real thing that I need to reach and I have some objectives I want to satisfy etc., etc.

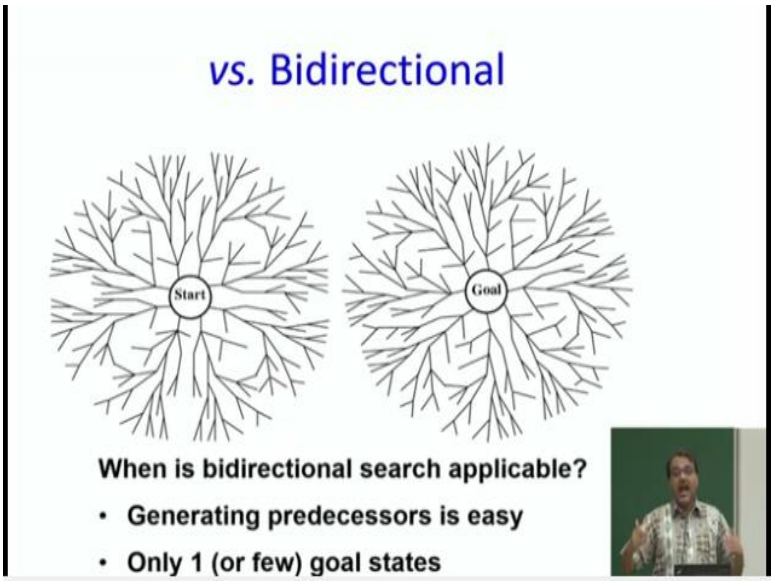
While you can start thing is about where I am right now and what are the possibilities I can do, so I will start searching various kinds of directions trying to get to that point or I can start to ask the question, okay what actions will take me to the goal and I can try to back change these actions in order to see if I can hit the start state and sometimes, you know one works and sometimes another will work, right.

So, I can do forward search and I can do backward search but I can also do yes, yes, I can also do forward, backward search which is also called bidirectional search, so in this case, the running example is that this is the map of Romania, I want to go from Arad to Vaslui and I need to figure out how to reach there and one way to do this is to say okay, can I go from Arad, okay I can go to Zerind, I can go to Sibiu and so on and so forth, Timisoara.

And then I can keep doing this and finally, I will hit Vaslui and I will know that I have found a solution alternatively, I can say, okay, where from can I get to Vaslu, so I can go from (()) (02:58), I can go from other cities, I do not know the pronunciation of all of them and then I can keep going back and then I can see how do I reach to Arad but there is a better solution, so by the way, when would you use forward search against backward search?

Lesser branching factor, so now we have to define 2 different kinds of branching factors, what is the average branching factor going out of a node and what is the average branching factor going into a node, you can call it a fan out and a fan in branching factor and your fan out is lower, you would do forward search and your fan in is lower, you would do backward search, very good and alternatively, you can do bidirectional search.

(Refer Slide Time: 03:49)



What is bidirectional do; it says search, search, search from the forward direction; search, search, search from the backward direction and at someplace, they both magically meet and as soon as they meet, as soon as they reach that one point from where you know the best path from start and you know the best path to goal, you are done, you can stop. So, 2 questions; first question, when is bidirectional search applicable, what do I need in my domain for bidirectional search to work out?

So, in the given problem where I am giving you only a goal test is bidirectional search applicable, I am only giving you the start state and goal test, is bidirectional search going to be applicable, why not? No, no, I am also giving you expansion function in forward direction, good, so one question and I am also giving you expansion in forward direction, why will bidirectional search not work, what are the reasons?

You do not even have the goal to start it, you have only given the start state, in fact there may be many goals because it is a goal test and if there are many goals, then there is no way I can do bidirectional search, because I have to do really backward search from all the possible goals and that will be too much to do. The other thing is I am only given expansion function in the forward direction, I need the expansion function in the backward direction, right.

So, those are the 2 reasons that make bidirectional search inapplicable, so whenever I am given only 1 or maybe 2 or very, very small number of goal states and then generating predecessors are possible, then bidirectional search becomes applicable, okay. The other question is suppose, I am using breadth first search in both directions, in the forward direction and in the backward direction, is it complete?

If there is a path, will it find it; see the breadth first search is complete, so for whatever length has expanded to it definitely knows all the states that can be reached in that particular length, right and in the same way in the backward direction. So, it is going to be complete because whenever it has to meet somewhere, whatever is the length, it has to meet somewhere in fact, not somewhere it will meet at what depth?

Yeah, assuming I do one step expansion of forward and one step expansion of backward, so it will meet at d by 2. So, now this is nice, I have instead of one expansion up to depth d, I have 2 expansions up to depth d by 2, is it going to be better; yes because these are not linear, these are exponential and so we are saving on so much computation by doing this?

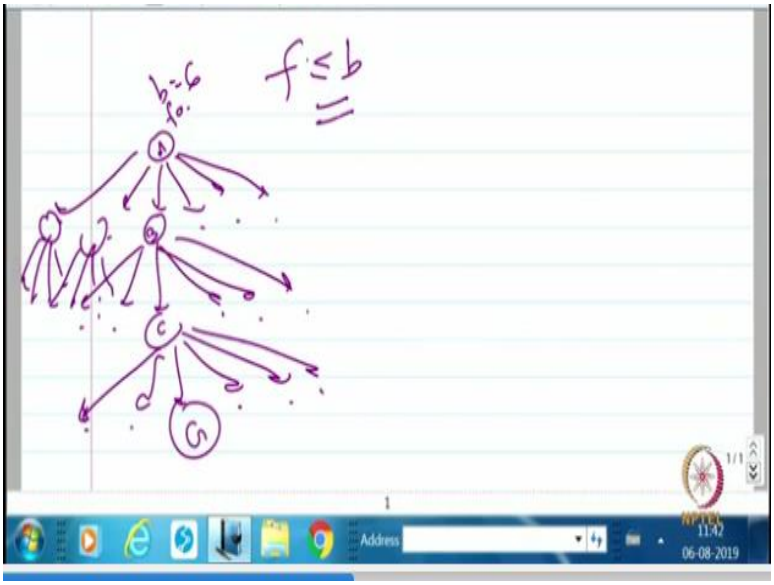
(Refer Slide Time: 07:13)

So, our time complexity will end up becoming b by 2 and space complexity will be equal to 2 times the breadth first search and complexity; a space complexity again with the d by 2 and your optimality criteria would be first step optimality, it would be optimal because you know, you are

saying the same thing the number of steps, you will always be able to reach node in the same number of steps.

Or if you want cost optimality, you can just do uniform cost search in both directions, so if you do uniform cost search in both directions, then you will find that it is optimal, there was a question; **"Professor – student conversation starts"** right, right, right, yeah, so I mean there is a nuanced answer there, the question is how do we define average fan in and average fan out, also you have to think about the situation where one node goes to many, many different nodes, right.

(Refer Slide Time: 08:37)



And eventually, we are only interested in reaching the goal, so we can make a quick diagram hopefully, so one setting would be this, and let us say this is the goal and then these are all the various other nodes that will get explored, if I am doing search in the forward direction. In this case, let us say exactly, this is the search tree then, searching in the forward direction is going to be really difficult, right.

Because my average branching factor or my branching factor here is 6 in the forward direction, so this your fan out, right but if you looking at fan in, it is exactly 1, so in fact even I do not even need to do search, there is only one way to reach the goal which is C. So, if I give you a graph

like this and I am saying there are other nodes here and they all have many branching factor but none of them go to the goal.

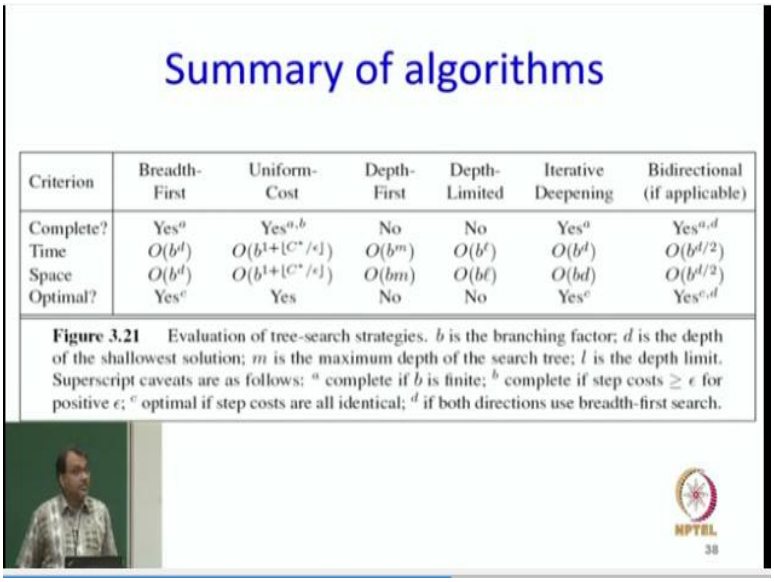
So, in such a situation forward search would be much worse than backward search, if I do a search backwards, then I will be able to reach the start state much faster, make sense, good, alright. Yes, another question; oh, so you are saying that if there is no solution, so the question; what is your name Kalash so, Kalash says what about a problem when there is no solution, what will happen if there is no solution okay, so this is actually good question.

We have sort of been sidestepping this particular issue in all of these exercises, what happens to depth first search, if there is no solution and let us say b and m are finite. If we can search the whole tree, and we find no goal, we say no goal found but in practice mostly, what will happen; it will just keep going and you know, it will never finished because m would be large enough, it will just keep going, it will never finish.

Similarly, for breadth first search, if you can finish expanding the whole tree and find no goal and we can say goal not found but it will take a huge amount of time and space, right, same for bidirectional search, you are going in bidirectional; in 2 directions eventually, you will end up doing 2 full expansions and after doing 2 full expansions, if you are not able to find any match at any time point in the fringe, then you have not found a path to the goal and that is where you finish.

In practice, however if you are really in a problem where no goal is found, you would really end up searching the whole thing, you would usually crash before that either your space will be exhausted or your time will become too large that you know, you will say okay, I could not find anything but all of them will have more or less the same behaviour, when it comes to a situation where there is no path to the goal, okay, good question**. "Professor – student conversation ends."**

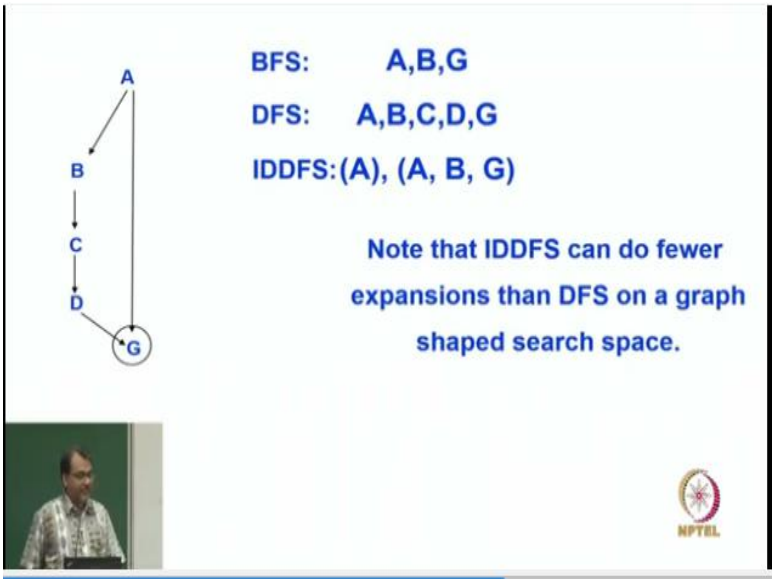
(Refer Slide Time: 11:42)



So, this is our summary so far that breadth first search is nice but it is a bad space, uniform cost search is also nice, it also has backspace, its complexity depends on the total cost to the optimal as well as the minimum cost edge, depth; iterative deepening search is much better because it has time complexity similar to breadth first search and space complexity better than depth first search.

And then bidirectional search is also pretty nice, it requires space and time complexity which is far less exponential in so, in b and d, alright.

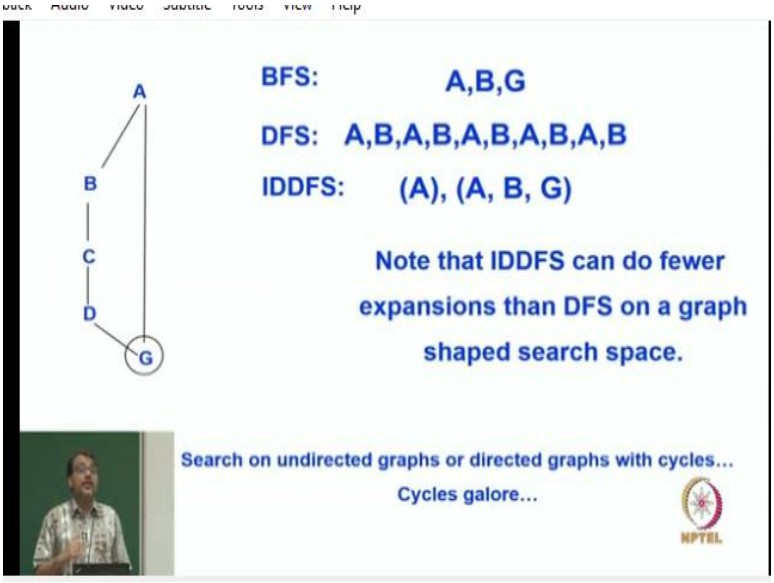
(Refer Slide Time: 12:30)



Now, I have to do 2 quick things, so let us finish this chapter, so let us say I am giving you this graph; ABCDG, okay and I need quickly for you to respond, if I did breadth first search and let us say I want to go from A to G, if I did breadth first search, what will be the order of expansion of nodes; A, B, G let us say, I am always going lexicographic in my children, if I am doing depth first search, my order of expansion would be A, B, C, D, G, goal found, okay, we have found a path not optimal path but we have found a path.

If I did iterative deepening depth first search, what will happen; A, B, G, okay so, this is a cute example in which iterative deepening search ends up doing fewer expansion than depth first search, okay not that bigger deal, the big deal is coming now.

(Refer Slide Time: 13:26)



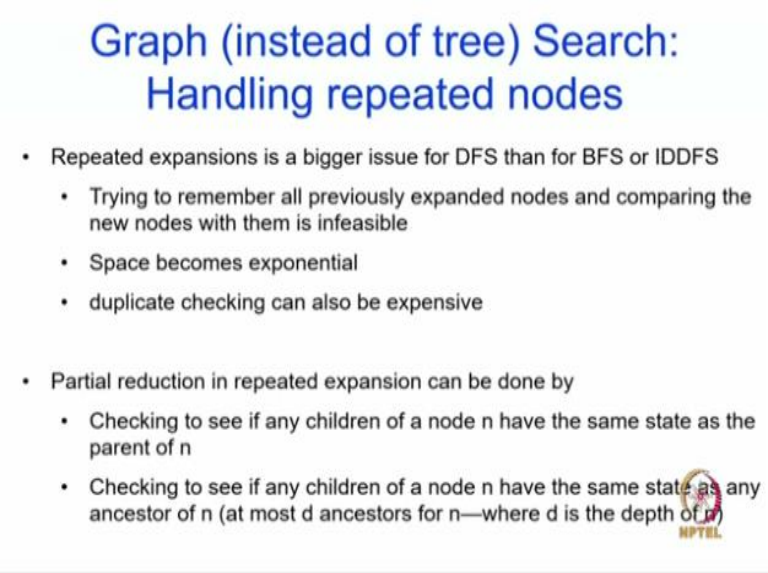
Let us take a view this graph, now the difference between this graph and this graph is that my edges are; yeah, bidirectional, I can go from A to B, I can go from B to A and let us say, I do lexicographic ordering again so, for breadth first search I will do A, common guys, this is not very hard, breadth first search, I will do A, B, G very good okay, now comes the good question. What will happen in depth first search?

Let us go slow; A, B, okay what is next, what are the children of B; A and C and let us say I am using lexicographic ordering what comes first; A, so I will do A, B, A, B, A, B, does this make sense, so this is where problems get really complicated if I have symmetric state space; search

spaces, symmetric as in bidirectional search spaces, cyclic search spaces because I can keep going in the cycle and unless I store additional information, there is no way to come out of it.

In order to go from A, B and not go back to A, what do I have to remember, that I already visited A but I may have visited in some other sub tree and then it will become very difficult because that will require a full blown duplicate detection.

(Refer Slide Time: 15:19)



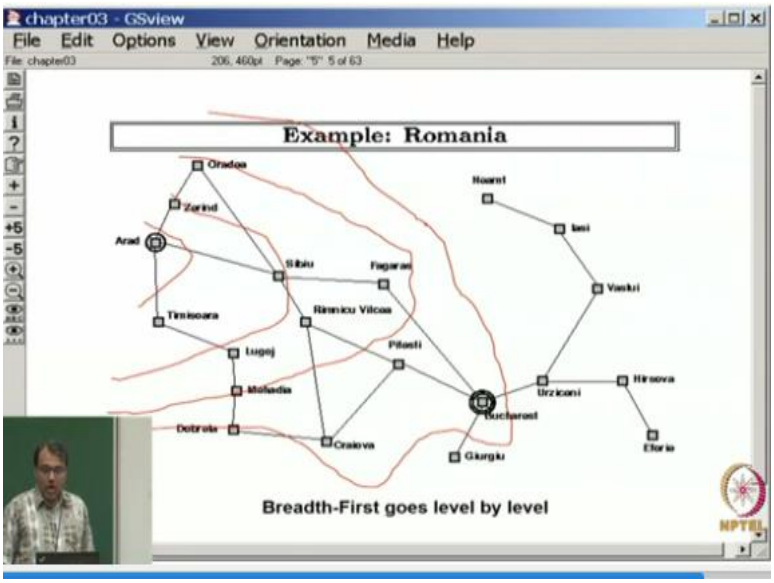
So, in practice handling repeated nodes becomes a challenge, it is a bigger issue for depth first search then for breadth first search and iterative deepening depth first search, although it is an issue for all 3 of them, trying to remember all previously expanded nodes and comparing new nodes with them is just not going to happen, it requires a huge amount of space, we do not have that space.

All our exercise was to reduce space, space becomes exponential duplicate checking also becomes expensive, right you have to do some hash tables and you need to come right. So, therefore in practice when you implement these things, you do partial reduction in repeated expansion basically, you check some repeated nodes. So, for AB, AB, AB, the simplest thing you need to check is; do not go in to your parent.

See, if every edges bidirectional, then if A is a child of B, then B is a child of A, so this is going to just you know cycle down, so just say okay, do not go to the same state if you are the parent, you can do something better by saying do not go to the same state, if it is your ancestor in the current path and this is the extend to repeated node handling that you would typically do, so checking to see if any child of node n is the same state as any ancestor of n, right that in the current path.

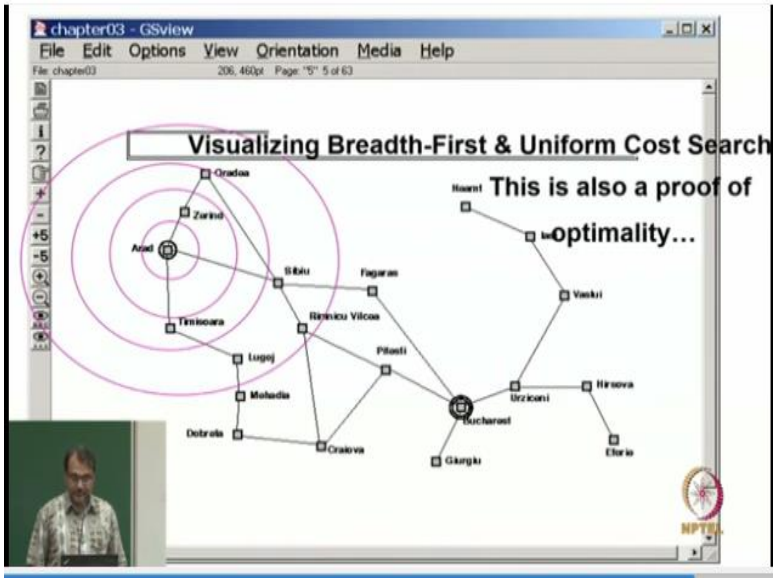
That would be sufficient; now, while sufficient in the sense that is a good balance of keeping some memory to reduce significant amount of duplicate computation, okay.

(Refer Slide Time: 17:03)



So, just some visualisations, this is how breadth first search is working out; first expand the node at depth 0, then expand nodes at depth 1, then expand nodes at depth 2 and so on and so forth, it goes level by level, connectivity by connectivity.

(Refer Slide Time: 17:17)



On the other hand, uniform cost search was like this, it starts from start state, explores everything in the circle, then explores everything in the circle, then explores everything in the circle and so on and so forth and basically, explores all the nodes in all the directions that are within a particular cost problem and you can actually extrapolate into proof of optimality, you can prove that all the states that are closer than to the goal will be visited before you visit the goal and so on and so forth.

And therefore, you can say that as long as your cost are positive, this is going to get you an optimal solution. Now, there is a fundamental issue with all these algorithms, these are all dissatisfying, think about it. Suppose, that is the chair I want to go to, I am starting in the start state, my thinking says let us go there, okay, this is uniform cost search, let us go there, okay, let us go there and so on.

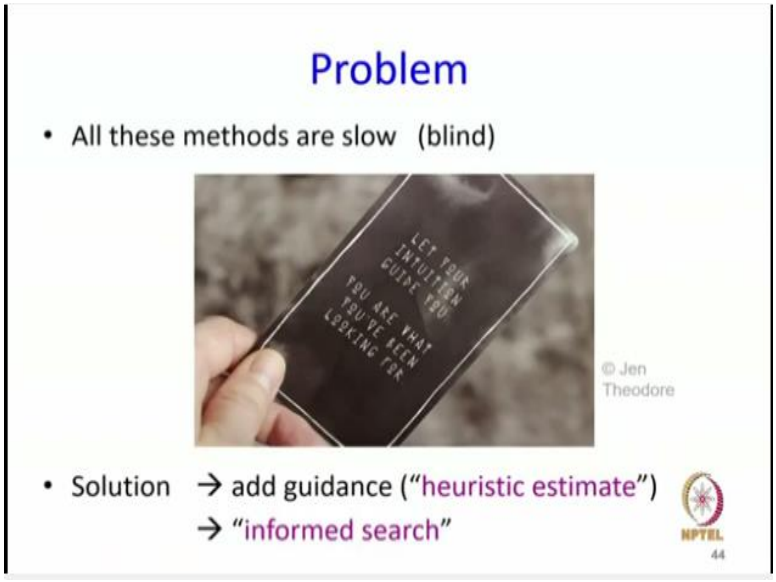
I am going in all directions before I start to go into a little farther and then I again, go in all the directions, you do not want like this right, I hope, if you have to reach that goal, you start walking right away, why do you do that? If the lights in the room are switched off, he really walks like this he says, yeah, correct, if we do not have any information about which state is going to be closer to the goal, there is just no way what you can do what you can, I mean, this is the best you can do, right.

By the way even then, just out of curiosity, US people will do depth first search or breadth first search, you do not say I go there then, I come back and try this and also we do not have that much memory to keep the whole frontier actually, computer has a lot more memory, we cannot remember all such possibilities, in life we mostly, if we have to do anything, backtracking, we do depth first search backtracking.

We do this in conversations, you start talking to me, I start talking to you, I go off for the tangent. If I go off for the tangent, at some point, I come back from the tangent and follow the main part of the conversation that is some kind of a depth first search that I am doing but I will not do breadth first search, I will say, okay, we have to do 3 conversations, let us do one step of this conversation, one step of this conversation, one step of this conversation, then the second step of this conversation, first conversation, we do not do like this.

We cannot keep all these things in our head, we have terrible space, we may okay time but we have terrible space, so anyway we do depth first search but then we also do not do depth first search because we have some intuition that this state is closer to that state that I want to reach to than that state behind.

(Refer Slide Time: 20:15)



Now, my intuition could be wrong, I am not saying my intuition is always right but we live a world where we have lot of intuition about the problem we are trying to solve. So, therefore all

these methods are slow, they are blind, they are uninformed, we need to have some intuition put in into these methods which is what we use all the time in doing all the problems and so the solution is going to be add a heuristic estimate about how far the status from the goal and that is called inform search and we are going to talk about it in the next segment that is tomorrow.