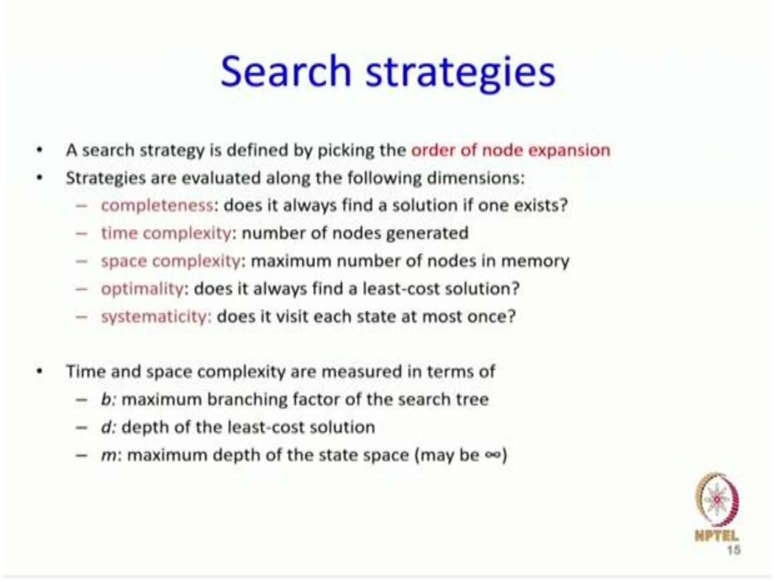
Artificial Intelligence
Prof. Mausam
Department of Computer Science and Engineering
Indian Institute of Technology - Delhi

Lecture – 12
Uninformed Search: Basic Search Strategies (Part – 3)

Now, what we are going to do is what Sukradhi already alluded to which is that we want to solve this problem using some kind of a strategy; a search strategy. Now, the good thing is that there are many possible strategies here and Sukradhi mentioned only one of them which is depth first search and in fact, the first order of business that we have is to go over many search strategies and think about their strengths and weaknesses in the formulation that we have described.

(Refer Slide Time: 00:55)



Search strategies

- A search strategy is defined by picking the order of node expansion
- Strategies are evaluated along the following dimensions:
  - completeness: does it always find a solution if one exists?
  - time complexity: number of nodes generated
  - space complexity: maximum number of nodes in memory
  - optimality: does it always find a least-cost solution?
  - systematicity: does it visit each state at most once?
- Time and space complexity are measured in terms of
  - b: maximum branching factor of the search tree
  - d: depth of the least-cost solution
  - m: maximum depth of the state space (may be ∞)

15

This is the first order of business, this will take our time today and probably part of the next class and how do we evaluate how good a search strategy is, we will evaluate based on these 5 problem characteristics; one is completeness, one is time complexity, space complexity, another is optimality and one is systematicity, so let us quickly go over them. Completeness means that if there is a path to the goal from the start state, will it find it?

There will many paths, we are only interested in one of them, we may find a suboptimal path for completeness, that is okay but if a path exists, our algorithm should finally be able to find, any one of them and that is the notion of completeness, fair enough. Time complexity you understand, space complexity you understand, we can say time complexity is number of nodes expanded or number of nodes generated.

Space complexity is total number of nodes that I am storing in my memory, right notice I am saying nodes are not states because we are in search nodes, we will now we are looking at search nodes as opposed to world states because the world state could come in many, many times. Now, in order to compute time complexity and space complexity, we will use 3 letters, 3 symbols; b, d and m, okay.

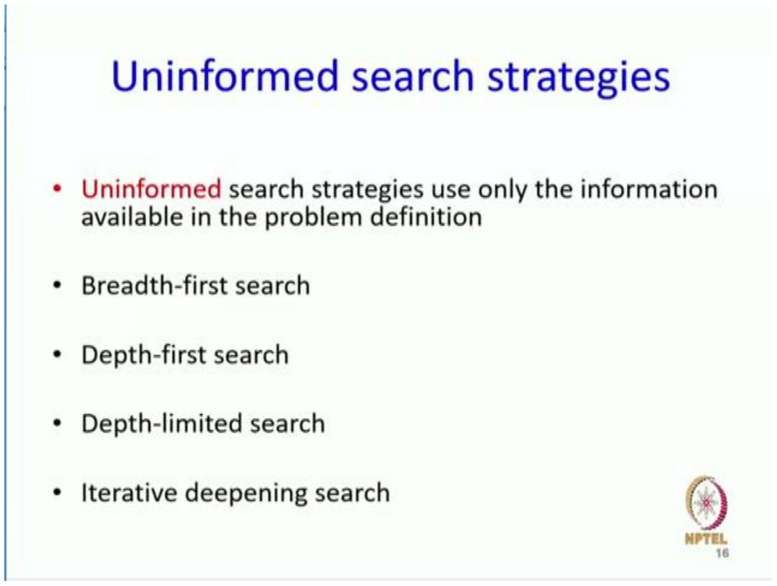
B is the maximum branching factor of the search tree for 4 queens or N queens, it may be n, n for n puzzle it may be 4, okay, so b is the branching factor. How many actions can I take in a given node, in a given state, then m would be the maximum depth of the tree, okay, the root is at depth 0, all the next states at a depth 1, the next nodes are at a depth 2 and so on so forth and we will say that my maximum depth is m.

However, m could be potentially infinite and suppose I am not doing any duplicate deduction in and n puzzle, is it infinite; because I could do gap up, gap down, gap up, gap down, gap up, gap down and I can keep doing this and it will never end, so I can technically have infinite depth in my search tree and last but not the least, my algorithm may not no, d but for the purpose of analysis, we will assume that we are given a d, which is the depth of the least cost solution.

So, it tells us that look your goal exists at depth d and it may exist later also but it at least exists at depth d, so the algorithm does not know it but you know it for the purpose of analysis, so we will compute time complexity and space complexity using b, d and m, optimality is does it always find a lease cost solution and systematicity is one unusual phenomenon but it is an interesting concept, it says do I visit every search node at most once, not state?

Do I visit every search node at most once, okay sometimes I may visited multiple times and then the algorithm would be call non-systematic and we will see some examples, okay.

(Refer Slide Time: 04:21)

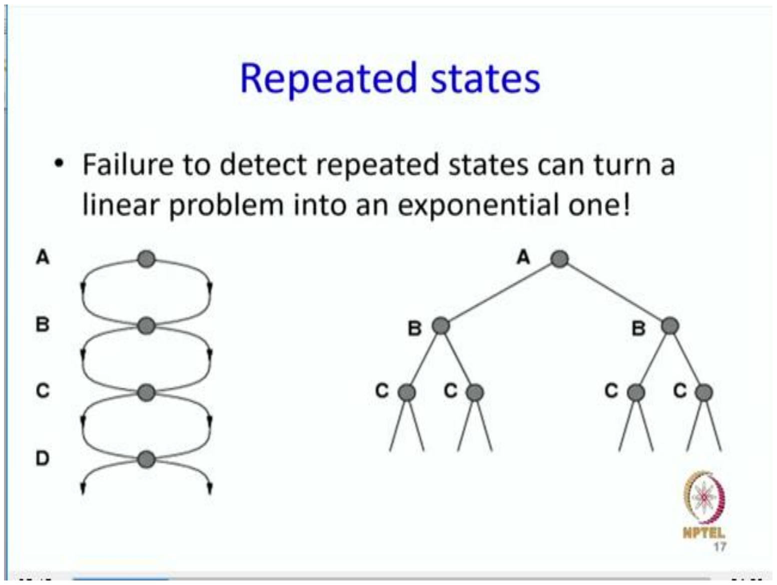


Uninformed search strategies

- Uninformed search strategies use only the information available in the problem definition
- Breadth-first search
- Depth-first search
- Depth-limited search
- Iterative deepening search

And we have to study breadth first search, depth first search, depth limited search, iterative deepening search and beam search has 5 uninformed search strategies, okay.

(Refer Slide Time: 04:36)



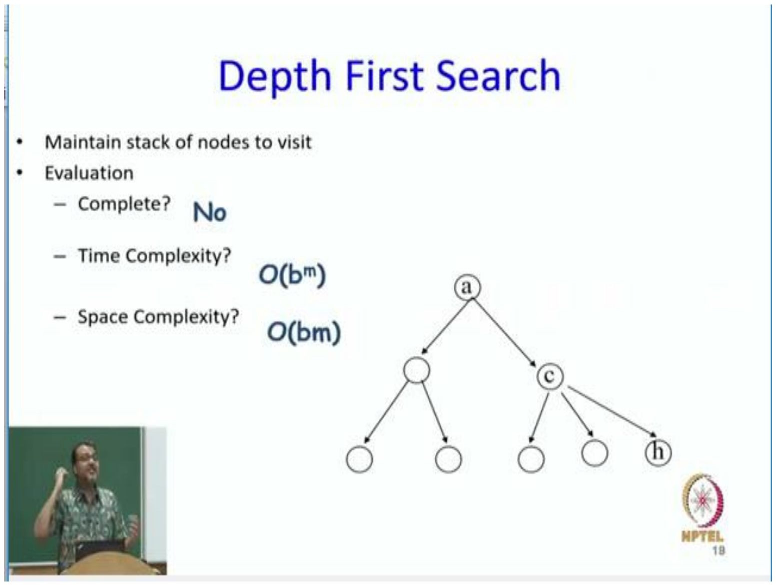
Repeated states

- Failure to detect repeated states can turn a linear problem into an exponential one!

Now we have already discussed this but if our states are repeated, then my tree may become really large, for example, it is a simple example where there are 2 actions in A state, both lead to B, 2 actions in B state both lead to C, 2 actions in C state both lead to D and you can see that when I create a search tree, I will keep replicating a B twice, C 4 times, D 8 times and I will create an exponential tree even though my world state graph was leaning.

So, therefore there are challenges with this and then we can always do the graph search version of this but the other challenges we will discuss later but for now, we keep this caveat in mind. So, let us come back our N puzzle and let us see what we could do.

(Refer Slide Time: 05:24)



Depth First Search

- Maintain stack of nodes to visit
- Evaluation
  - Complete? No
  - Time Complexity? $O(b^m)$
  - Space Complexity? $O(bm)$

a
c
h
NPTEL
18

And Sukradhi suggestion is the first suggestion that we would make which is let us do depth first search and let us say these nodes are a, b, c; a, b, c, d, e, f, g, h, let us quickly see how we would do depth first search, so what is the first node that we will go to? I need everyone to answer collectively what is the first node I get to; a, after a let us say I go to left, okay, so which node I go to; b yeah, we go left, so we go to b.

Now, which node do I go to, do I go to c or do I go to d or e in depth first search? D, right because we are doing depth first search, this is something you know, you know this right, depth first search goes depth twice, it first goes and completes the particular sub tree and then come back and back tracks, this is the definition of a depth first search. So, after b, I go to d, now there is nothing else beyond d, so what do I do I, backtrack.

So, I backtrack back to b and then I go to e, now I have exhausted, there is nothing in e, so I go back to b, I have exhausted b, so I backtrack to a and then I go where; c and then f and then g and then h and at every point I go, I stop, I asked my domain designers black box is this a goal, is this a goal, is this a goal and my domain designer says, no, no, no and eventually when you find the goal, it says yes and then are you done, our job was to output a path.

Suppose, g was the goal, what would you output, a, c, g because we were interested in out putting a path, so far so good, **“Professor – student conversation starts”** yes, very good, very good, you are running ahead of me, what is your name, Vignesh. Vignesh says like no, no, no, this is not good, depth first search may never end (()) (07:44) wait, I am going to ask you exactly that question. Is it complete, think over it, if there is a path to a goal, we did always find it.

How many people say yes, it is okay, you have to say something, it is okay to be wrong, nobody is judging, think about it how many people say that it is always going to find if there is a path to the goal, 1, 2, 3, 4, 5, 6, 7, 8, 9,10, 11; how many people think it will not find a path to the goal even if one exists, in a class of 150, we are at a 9 versus 13 or 14, come on guys, you can do better, okay.

I am going to ask more such silly questions, I hope more hands get raised otherwise; I will pick on someone who is not raising their hand, okay. Why would it not be complete no, well Vignesh would answer, why would it not be complete, if the depth is infinite okay, so let us take an example, we never return back from d let us say, okay the goal is g but we never return from d because that sub tree never ends.

If that sub tree never ends, will we ever find the goal? No, so what if my search tress was finite, then is it complete; then it is complete, very good, so it is not complete because my depth can be infinite. Okay, what is the space complexity; we have to answer it in terms of; no time complexity, we have to answer in terms of b, d and m so, intuitively what is happening, how many nodes might I have under depth 0? 1.

How many nodes might I have in depth 1; b, right because I may have up to b nodes, how many nodes might I have in depth 2; b square, okay so, first of all notice the exponential characteristic of this, I may end up expanding many, many nodes as I go one depth further one. Now, suppose my goal is at a low depth, am I guarantee that I will not be searching beyond the depth, because suppose, my goal is on the right sub tree and I decided to go to the left sub tree.

Then, I need to spend all my time searching over the let sub tree before I finally get to the right sub tree, so I have to basically search the whole of the sub tree in the worst case because in the worst case, it is possible that my goal is the right last node that I encounter in my graph or in my search tree. So, my worst case time complexity would be b to the power m because what is the full size of my search tree is b to the power m.

So, this does not look very good, does it, what will be the space complexity? Now, this is very interesting alright, so let us have people, why do not you raise your hands, give some physical word, yes the person in the blue and red, what is your name? Naman; Naman says order m, okay let us keep thinking about it, you might be right, any other suggestion, anybody else, nobody else, if you do not maintain any stag then it is m, if you maintain a stag then what?

If you maintain visited nodes; so for duplicated deduction or for what; yeah, we are not doing full duplicate deduction, we ignored duplicate deduction, yes, order b to the power d, okay, why? Not known depth, it is a depth at which the goal might be found right, it is a lowest depth, yes but we do not know d, at the for the algorithm, it is only for analysis, algorithm does not know d, yes, anybody else, b to the power m, what is your name?

Vedant; Vedant says b to the power m, any node I visit, I should keep in memory, is that what depth first search does, why would I keep everything in the memory, it does not, what does it need to keep in the memory, so that it can run the depth search algorithm, only the current path but a little bit more, let us think about that little bit more part. So, wait, wait, wait, give me a second, let us say we are here.

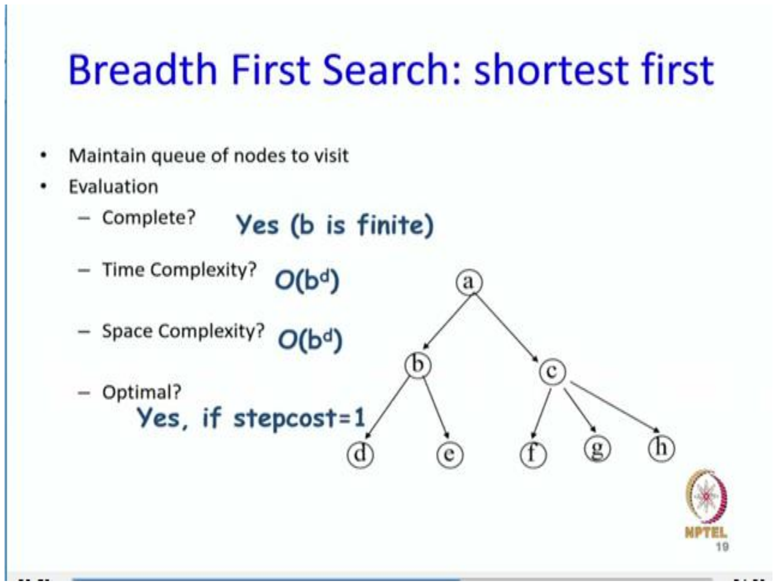
When we realise that we have to backtrack back to b, we need to know that all of b’s children have been explored, fair enough because if it does not know that it may go back to d and then keep oscillating between the d, d, d, right, so it needs to know that oh, b has 2 children and both of them have now been visited, so that it can backtrack back from b, so at every node in the path so far, I need to maintain all the children and whether they have been explored or not explored or visited or not visited, does it make sense.

Similarly, when I go back to a, then I need to know that a should say, look I have 2 children; b and c and b has been visited let us go to c, so therefore how much space do I need to keep, what is my maximum length of the path, m and for each node I have to keep how many children in the worst case; b, so my space complexity becomes order bm. Very good, so what is your name Rashi says, Rashika, I thought AI system become a better at speech recognition, looks like human systems are becoming worse.

Kashika, okay, most speech recognition systems use a vocabulary that is given to them, so that they can take one of those vocabularies, words in this case, your name was not in my vocabulary, so that was the problem, okay, good. So, Kashika says if we knew, if we could order all the nodes in some canonical ordering, then we do not need to maintain all the visited nodes and that is absolutely correct. **“Professor – student conversation ends.”**

So, then you are changing the problem, you are saying not only do I know the children but I also know some order, some canonical ordering of the children, so if you have given that, you can get away the order m, you are correct, good, okay, so then so depth first search is the deepest first but what is an alternative; breadth first search right, you all know this.

(Refer Slide Time: 16:09)



Breadth First Search: shortest first

- Maintain queue of nodes to visit
- Evaluation
  - Complete? Yes (b is finite)
  - Time Complexity? $O(b^d)$
  - Space Complexity? $O(b^d)$
  - Optimal? Yes, if stepcost=1

So, you know in what order am I going to execute, first I will have a, then I will have b, after b what will I have c, right because I am going in breadth order, ones I have exhausted depth one, then I go to depth 2, so then I will have d, e, f, g, h and at every point I will maintain a frontier and this frontier will tell me all the nodes that I have not expanded so far. So, for example after this, I will have in my frontier b and c then I expand b, so I add d and e in my frontier.

And so my frontier will look like d, e, c, then I; and then I expands c and then my frontier will look like d, e, f, g, h, right, so this is how breadth first search works. Now, depth first search at a problem with infinite depth would breadth first search have a problem with infinite depth, why not because we will expand all nodes of a certain depth before we start expending any depth, any node in the further depth, okay, it is an important point.

So, now can we say it is complete; yes, we will say it is complete because we will assume that b will be finite, now in most of AI, we are going to assume b is finite because if b is infinite, we cannot even go beyond first step in a search mechanism, so therefore AI systems were more focused on discrete problems as opposed to continuous variable problems, all the general principles apply but the specifics become very different.

So, therefore we will for most part in the AI course will deal with discrete state spaces and we will assume b is finite, time complexity; okay, so now time complexity earlier was b to the m because we said in the worst case, we would may have to expand up to depth m now, here what is the maximum depth that we may to expand until; d, right, so we will have time complexity b to the d.

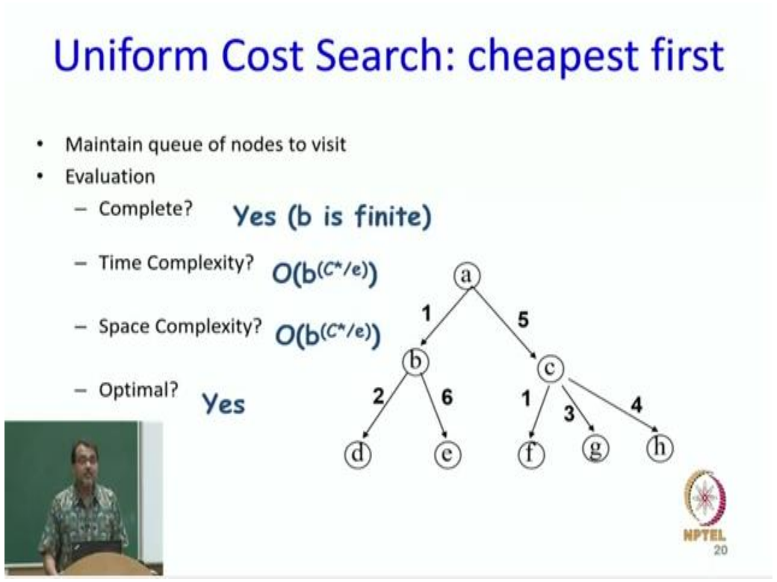
Similarly, space complexity would be now, what about space complexity; in depth first search, we had to only maintain the path but in breadth first search, we have to maintain the full frontier and in this frontier how many nodes in the worst case might I have, again all the nodes are depth d in the worst case right, so b to the d and finally is it optimal; see we did not ask you the optimality question in the depth first search because it is not even complete.

Then, obviously it is not optimal but now we can ask the question, how many people think it is optimal, okay, good, more people are raising hands. **"Professor – student conversation starts"** How many people think it is not optimal; 1, 3, 5, 7 why not; person in the blue shirt, what is your name; Harish yes, why not; no, optimal not in terms of time complexity, the question about optimality is if it finds a path, would it be the least cost path that is a question, that is the meaning of optimality is, that whatever path it returns, is it the least cost path to the goal.

So, anybody else who still think it is not optimal, yes, what is your name, Saigon yes, very good, least cost in shortest path may not be the same, cheapest is what we are looking for not shortest, so when would it be optimal; if there are no cost, so equivalently, all costs are the same, very good. The slide says yes, id step cost is 1, but basically we can say all the step cost are the same, very good. **"Professor – student conversation ends."**

So, now we have understood breadth first search, I will take 3, 4 more minutes, I want to show you the next important algorithm and then we will come back to it later. So, the next algorithm is uniform cost search or the cheapest first, so we want to solve this problem that it was not optimal because if I have multiple step cost, then breadth first search may not always be the optimal solution.

(Refer Slide Time: 21:16)



Uniform Cost Search: cheapest first

- Maintain queue of nodes to visit
- Evaluation
  - Complete? Yes (b is finite)
  - Time Complexity? $O(b^{(C^*/e)})$
  - Space Complexity? $O(b^{(C^*/e)})$
  - Optimal? Yes

$$\begin{tikzcd}[column sep=small] & & & a \arrow[dl, "1"'] \arrow[dr, "5"] & & & \\ & & b \arrow[dl, "2"'] \arrow[d, "6"] & & c \arrow[d, "1"'] \arrow[dr, "3"] \arrow[drr, "4"] & & \\ & d & e & & f & g & h \end{tikzcd}$$

And so we have a different algorithm called uniform cost search, which basically says expand the closest node first, cheapest node, close; not closest but cheapest right, whichever has the cheapest path to that node expanded first so, I will start with a but after a, what will I expand; b, what is my decision, b or c and which is cheaper; b, right so, I expand b. Now, as soon as I expand b, I can know expand d, e and c but d is at cost 3, e is at cost 7 and c is at cost 5.

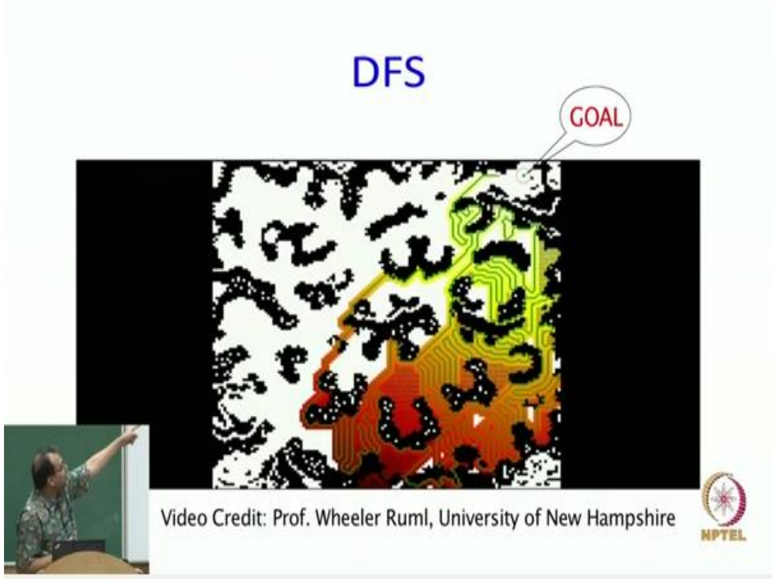
So, what do I expand; d because that is the cheaper exactly, it is at cost 3, so what will be next; c, right because it is 4, what will be next; come on guys, everybody, if you do understand this means; e, f, g, h these are the 4 options we have; f, because its cost is 6 whereas the next better cost is 7, so we expand f next, then e, then g and then h, okay, is this complete; this is a tricky question; the costs are integers okay.

So, yes and no so, the only thing you need to ensure that cost are not 0 because if the costs are 0, then I may keep adding many, many infinite 0 cost edges and then I may get to non-completeness, so that is a tricky saddle point but as long as costs are positive; by the way cost are positive you know all these discussion and in fact negative costs create even more trouble, we will come back to that or you will get that question in the exam.

So, negative cost are actually more complicated you think about why, we will discuss later. Time complexity; now, this will be tricky for you so, I will not ask you to come up with it but let us say my lowest cost; the lowest cost is epsilon and my costs; the optimal cost is c star, then you can prove that your time complexity would be order b to the power c star by epsilon similarly, for space complexity.

And this one is optimal even for non-equal costs, so before I stop I want to show you 2 videos, these videos will give you a quick understanding about how depth first search and breadth first search work.

(Refer Slide Time: 24:06)


DFS
GOAL
Video Credit: Prof. Wheeler Ruml, University of New Hampshire
NPTEL

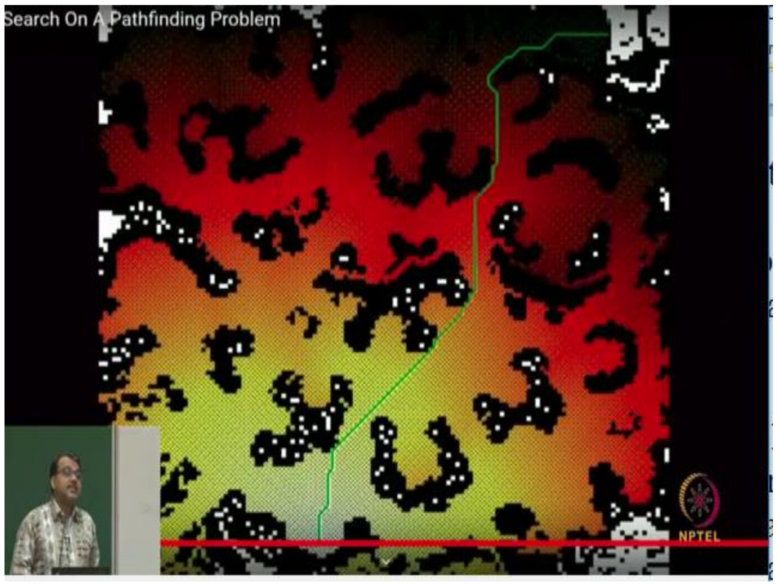
So, this is how depth first search works, it starts from the start state and let us say, this is the goal it has to go and it is taking some random action, it has no information about where the goal is, so it has started from here, it needs to go here, so it keep searching and it keep searching and when it cannot find anything, it backtracks as you can see some where it backpacks like for example, it reaches this particular hole, it does not know what to do so, it starts to backtrack.

And it keeps going and going and going and now notice that even for the simple problem, m is pretty high, the maximum depth is pretty high of the search tree, so therefore if it has made a wrong decisions for then it will find very hard to come back and find a path that can reach that particular goal, right, this is how depth first search is working, it just make some random call and says, I am going to there and now, I can go somewhere else, I am going to somewhere else.

And it keeps doing it, then it cannot go anywhere else, it keeps backtracking and then it goes somewhere else and it keeps doing that. So, can you guess what is going to happen here, is it going to find the path or not; yes because my search tree is finite, my m is finite, right we are doing full duplicate detection here but oh, it found a way to go out of this hole, oh not yet. Now, also notice what this green shows?

This green shows the path found so far, so now it is certainly goes around it and it might find an alternative way to reach this place and even if it finds it, what path would it output, this complicated path to reach the goal and which is why we said if it find the answers, it is not going to be optimal and if you had an AI system, returning that particular path to the goal state, you will say that the AI system is quite doubt.

(Refer Slide Time: 26:16)



On the other hand, let us also run the uniform cost search algorithm goes in all directions, it maintains this yellow thing which is the frontier and it keeps expanding in all directions and when it finally converges, it will reach the optimal path see, here uniform cost search and breadth first search are the same because all costs are the same, right. Now, I can assure you that you are going to even get bored watching this.

This is a full 2 minute video but it gives you an intuition about what breadth first search does and what depth first search does, so what have you been trying to do; our goal has been to solve pretty much every problem in the world and of course not every problem as possible, so we are interested in solving every deterministic single agent problem in the world for now, okay and in that context, we try to define a general definition of a search problem, right.

What is the general definition; I will be given a set of states, I will be given a set of operators are set of actions and I will also be given a tests which tells me whether it is a goal or not and start state but I will not enumerate all the state space for you instead, I will give you the start state and I will give a black box function, call the evaluation function or the expansion function or evaluation; expansion function and given a state, if you give it this expansion function, it will tell me the next states you can reach from that particular state with the action that are given in my system, okay.

And we discuss that we are given such a problem, half would be solved such a problem and we realise that this is nothing but a draft search kind of an algorithm in that is coming in, like depth first search or the breadth first search and of course you have all studied depth first search and breadth first search, so I went very too fast but the main point is that in depth first search I take an action and I explore that sub tree completely before I backtrack and get to the next action, right.

So, if I made a bad move early on, then I will end up exploring a lot and possibly not getting right solution, the optimal solution so, for example are we looked at this particular video where our goal is to go from the start state to this goal state and the system starts by you know, taking some actions like it starts to go in this direction straight and so on and so forth and then it backtracks from some place.

So, these shaded regions are the regions where it tried to find a path to the goal from and it could not, so it sort of backtrack and it keeps doing this and it keeps doing this and finally, it finds a path that can go around the start state towards the left and then it finds a solution, I have stop the video just before it finds a solution and notice that the path that it returns is this complicated path, this is sort of the final path that it is able to return, this green line.

And of course, that is highly dissatisfying but there is one advantage of this and the advantage of this is that it takes less space although, it may take a lot of time, okay on the other hand, we have breadth first search, okay and in breadth first search or uniform cost search right, so we also studied uniform cost search, this as an example of an uniform cost search which will end up being exactly equal to breadth first search.

Because in all actions are equal right, so they are the same thing, so you start with the start state and then you keep going and at every point, you have a fringe, okay, the colours here; the red and the green and the yellow is sort of tell you how far from the goal you are and so notice that all states which are one step away will have the same colour, all states which are you know 10 steps away which will have the same colour.

All states which have you know 20 steps away will have the same colour and so on and so forth and so notice that it is going in all directions from the start state and always looking at all the states that are possible in a certain fixed depth because of which, it keeps going from left to right to straight to you know whatever all possibilities and finally in the very end, it finds a path. So, when you look at this, you realise at least it is giving you the optimal solution, right.

So, let us look what is the final solution it finds, it will basically find the optimal solution, right and that is great at least if you find the optimal solution but what will it have to do at any point in time it will have to at least maintain the whole fringe and that fringe will be exponential sized, b to the d sized, right. So, we also did some time complexity and space complexity computations and we realised that for depth first search, the time complexity came out to be exponential in the maximum depth of the tree.

But the space complexity came out to be only linear or sort of b into m, right b times m, where b is a branching factor and m is the maximum depth of the tree, on the other hand for breadth first search, we found time complexity and space complexity both to be b to the power d, where d is the maximum is the depth at which the first goal is going to be found, right and we also studied uniform cost search and there was a slight confusion last time.

We discussed that for 0 costs edges, uniform cost search will not terminate but that is not full story in fact, a person came back at the end of the class and discuss what about I have geometrically decreasing edge costs, like you have 1 and then you have 1/2 and then you have 1/4th and then you have 1/8 and so on and so forth and if I will have to keep taking those edges and it will take me infinite time to get to 2, whereas that side the optimal path is 2.25, I never reach the goal on that, again he is right.

So, therefore uniform cost search we typically say that they should be a minimum cost in the system and the minimum cost should be positive, so in this case, in this slide, the minimum cost is denoted by e or sometimes it is denoted with epsilon and then we are guaranteed for it to be complete.