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

Lecture – 13
Uninformed Search: Iterative Deepening DFS (Part – 4)

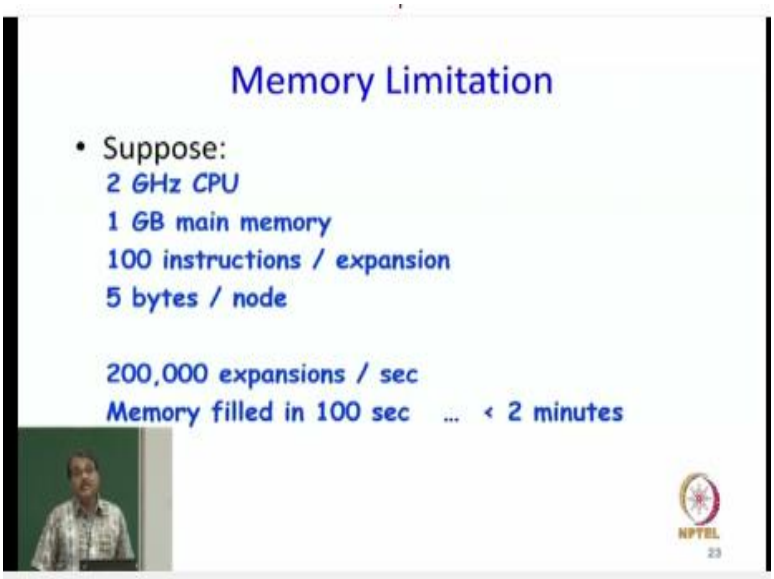
So, this is where we were and now, I ask you the following question; which is better; depth first search or breadth first search and I am going to ask you similar questions in the exams, so you to start critiquing these algorithms, see when have a data structure scores, you often think about okay, what is this algorithm and how does it work and what is the proof of correctness. In AI, you will often ask the question were what is good about this algorithm and what is not good about this algorithm.

So, at prima facie what would you say with when you compare depth first search and breadth first search, which is better in space; depth first search is better in space, which is better in time; breadth first search maybe better in time because d may be less than m and therefore b to the power d may be less than b to the power m, right d is less than equal to m at least. Now, I ask you the question and this is not a fair question to ask.

"Professor –student conversation starts" But still in general, what is more important; time or space, most of us feel the time is more important, why is time more important? Everybody felt time is more important, why is time more important? Yes, yes, what is your name, Vidit; yes Vidit sorry, we are adding storage very fast but we cannot add time very fast, right, although we are increasing the processor speeds also to some extent but maybe not exponentially.

Anymore yes, memory is reusable but time is not very good, what is your name; Akilan; Akilan says memory is reusable but time is not but if I am in one algorithm and my algorithm has not finished, can I reuse the memory; I cannot use the memory, I can reuse the memory if my one algorithm is finished and I want to run the second problem but if I am still in my one problem, it may not be finished, okay. **"Professor –student conversation ends."**

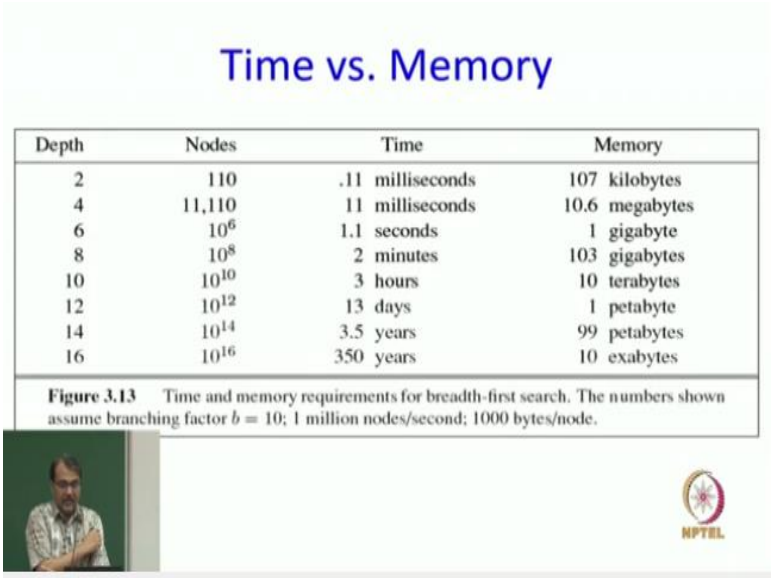
(Refer Slide Time: 02:37)



So, it is okay, right, both are important of course, and it is very hard to say which one is more important but in this context here, some figures, right, so let us say we have an old CPU; 2 gigahertz CPU with 1GB main memory of course, somebody would say, come on who cares for per node.

So, you will be able to do about 200,000 expansions per second and you will be able to fill your memory in 100 seconds, less than 2 minutes, right, you could argue that I can buy more memory but if you buy another 1 GB main memory, you will get another may be 2 minutes, right, okay, that is alright, who knows you know we have 256 gigahertz of memory.

(Refer Slide Time: 03:12)



So, here is some statistics, let us say our depth is 2 and branching factor is 10 where we only need a graph of; we only have a graph of 110 nodes and the time is only 0.11 milli seconds and memory is 107 kilobytes who cares. Let us say something bigger, let us say 10 depth, not 10, depth is not too much, if you think about the problem that I showed you the depth must be in 100's or 1000, right, the simple path finding problem.

So, 10 to the power 10 nodes will lead you to compute for 3 hours, that is substantial but will require 10 terabytes of memory; we do not have 10 terabytes of main memory prevails anywhere currently, so if we have branching factor of 10 and our goal is at depth 10, good luck trying to search it using breadth first search. What is going to happen; you will have segmentation fault or if you have more python weeks, I do not know, what is the right word, I learnt my computer programming in CC++.

But you will basically have a crash you will have memory over flow, right, so 3 hours is not too bad but even technically 13 is not too bad either if really, we had a problem we really wanted to solve and you know it was not imminent but you know our solution would really help, then we would not mind waiting 13 days sometimes in fact, there are lot of algorithms today which run for months at stretch.

Because you know, they are training these really complicated new networks of course, we are not talking about training your networks here but yes the point is we can sometimes wait 13 days or a month but I do not think we can make a computer which has 1 petabyte of RAM at the point. So, in the world where life is exponential of course, time and spaces are both terrible of course, we cannot wait 350 years to solve a problem, right.

So, if that is it you know, there is not much we can do, we can start running the process, you know 7 generations later, somebody will get to know the answer, our child's, child's, child's child, whatever right, sorry, why do we need RAM; we can also work with hard disk very good, so there are algorithms which are called external memory search algorithms and inside in 2010s, 2000, 2007 timeframe, there was a lot of work that was going on in that area.

I have done some work myself in external memory algorithms for an area that I used to work in (()) (06:01) processes yes, that is a possibility but the path that is not very clear is that so suppose I add one more depth, right by adding one more depth, I have basically doubled my memory requirement, so even if I have a disk space how much disk space would I need to solve with the really large problem, it is not clear but it is possible.

You can use external memory, they become very slow because you have to now do lot of access from the disk, you can; you do not want to do random access, so you change your algorithms also you look at those algorithms, which have better cash performance, better IO performance and so on and so forth, so there are; there is a full branch of study here but the point that I wanted to make is that an exponential space algorithm is not an acceptable algorithm in any world.

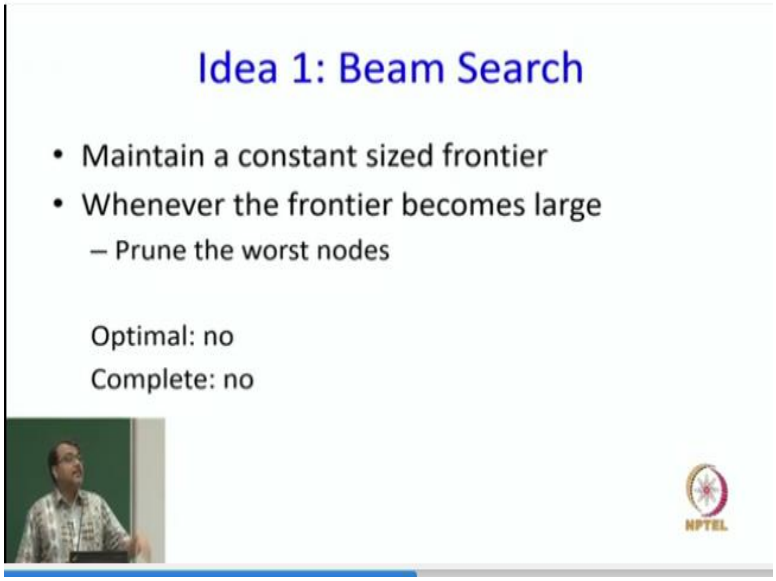
Exponential time algorithm is also not acceptable but exponential space in some ways may be worse because very soon, we will just go out of memory and you will not able to do any computation, right. So, there is a lot of work that has been done in AI to think about how to reduce the space requirements, we know that breadth first search has some very nice properties and uniform first search have some nice properties in that they are somewhat optimal or they are optimal depending upon the situation.

And moreover, they do not search for the whole depth of the tree, they only search up to the goal, which is also very nice but they have bad space requirements. So, what can we do and that is a goal for today, okay and the first idea is very simple, let us say I am running breadth first search and my frontiers become too large, what would I do? **"Professor – student conversation starts"** Very good, what is your name; Paul says let us find that part of the frontier which may have a higher chance of taking us to the goal.

"Professor – student conversation ends" And we are not getting there yet, this is what you are going to do in the next segment, next class but for now, we have in a; we are in uninformed setting, so we cannot do too much, we do not have any additional information then what was given in the original problem, so what would you do; if you disk is becoming; I mean if your memory is becoming too large, if your frontier is becoming too large, what would you do?

Remove; remove something, somebody was being nicely said remove duplicates but in order to remove duplicates, we need to store duplicates, in order to store duplicates you know any way taking more memory that is not really the way we want to go right now for sure, right. So, no, not just remove duplicates but remove nodes arbitrarily from the frontier and if I do that, that is called beam search.

(Refer Slide Time: 08:56)



Now, sometimes I am given additional information which allows me to pick which nodes I want in the frontier and which nodes I do not want in my frontier but for now, we will not worry about that right. So, let us say, I just remove some nodes arbitrarily, I prune; if I have a definition of worst nodes, I prune worst nodes or just I prune nodes and I always maintain a constant size frontier based on the amount of space that is given to me.

Now, will this algorithm be optimal; can this algorithm be optimal? No, it is quite possible that by whatever arbitrary decision, we got rid of the node that was going to take us to the goal, so this is the right. Will it be complete; same logic, I may end up removing the node which would have taken me to the goal, if there is another path, we may find it but completeness definition says if there is a goal, will I always, if there is a path to a goal, will I always find that path.

Well there are, we can construct cases where we will not find that path, so therefore it is not complete, right, so this is the easy stuff, right, nobody likes beam search but I can assure you when you add some notion of what is a good node, what is a bad node, in practice believe it or not, many people to be in search. For example, I work in this area called natural language processing, NLP text processing and a lot of problems.

In fact, you know how to make chat bot, you do beam search, believe it or not so, the skills that we learn in this class are going to stay with you if you learn them right, you know ad infinitum, so in a chat bot, we want to come up with the rights utterance, the right sequence of words that gets, this is the good response, so do we do; we do search over the next word so, I start from first possible many words and then I pick the top 5 or top 10 best words.

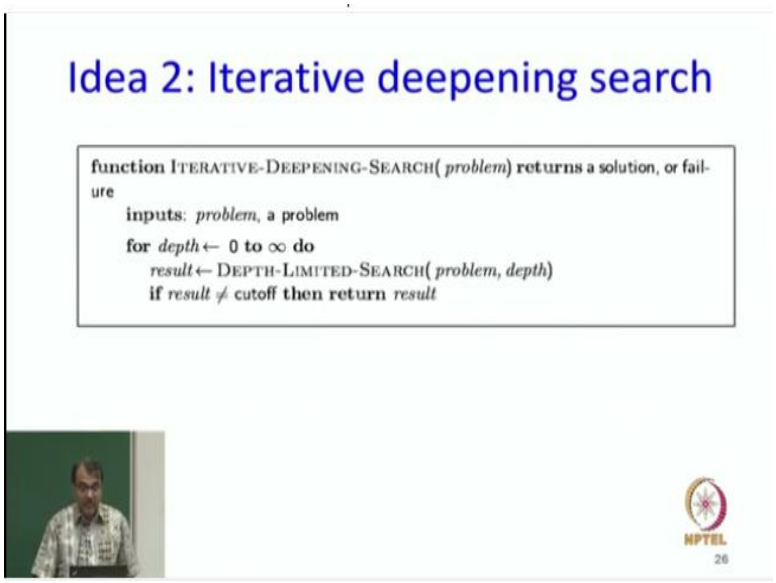
Then, I look at every 2 pair of words with the first word being what we had previously chosen and then reduce it to a beam of again 5 and 10 so now, I have the 2 sequence of words and I keep doing this and finally my sentence ends and then I look at all the possible sentences that ended and then I look at which is the best one and then the outputs, this is how I do beam search but there is a notion of what is a good sequence of words, partial and what is not, right.

But in our world, we will just do this arbitrarily, again this is not satisfying because now, we have lost optimality, we have lost completeness, these were nice properties of breadth first search, we do not want to lose that, really we are in the market for an algorithm which has excellent time, excellent space is optimal and it is complete, we are looking for that magical algorithm at this point.

Ideally, we want space to not be exponential ideally, we want time well it is okay, the time is explanation but not in m, it is okay if it is exponential in d and definitely, we want optimality and completeness to the extent possible. What might that algorithm look like; it is not very clear, right, where the hint has to come from depth first search because let us start thinking about depth first search as because it has the right space properties.

And then think about, how can I add some kind of optimality or completeness criteria, so the algorithm that we are going to study today, the main algorithm for today is called iterative deepening depth first search, okay. I hope for most of you this is a new algorithm although, is there anybody was learnt it before; good, so you can sleep, it is okay, alright.

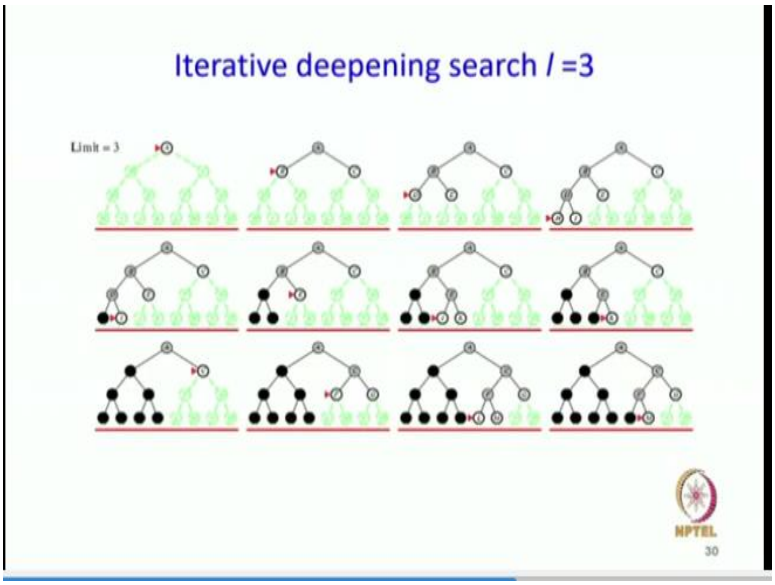
(Refer Slide Time: 13:06)



So, the idea of iterative deepening depth first search is do depth first search but stop in the middle and repeat and when we stop in the middle, we call this algorithm depth limited search, okay. So let us first look at the pseudo code; the pseudo code says, iterative deepening search returns a solution or a failure and for depth is equal to 0 to infinity, so it starts with 0 and goes for as long as it wants, you do depths limited search up to that depth.

If you find the solution, great return it, if you do not find the solution, increase depth so, let us see what is iterative deepening search going to do, when your depth cut off is 0.

(Refer Slide Time: 14:00)



That means, I am not allowed to search beyond depth 0, what will it do; it will just touch the root node and it will try to expand but it will not expand because after expanding, the depth will become 1 and we do not want depth to be greater than 0, so we will stop, that is it, I just touch the root node, come back home and say I did not find the goal but that is okay.

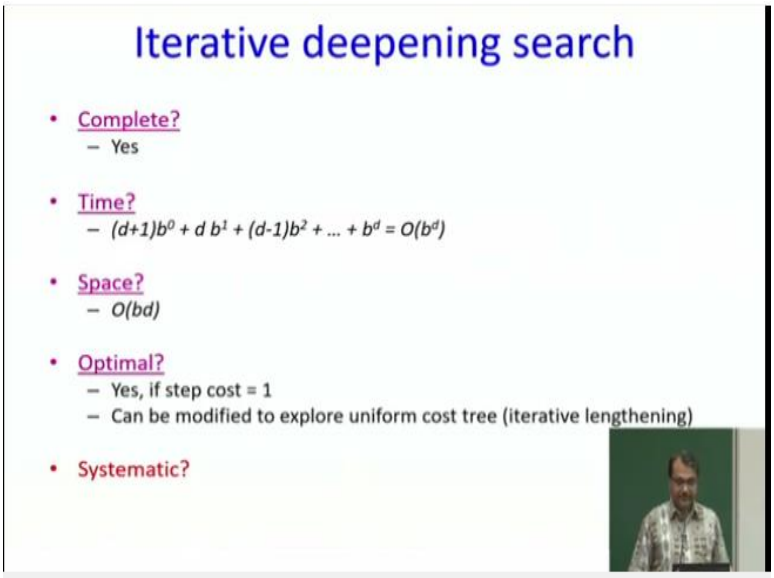
Because now, I am going to increase my limit to 1 and what would I do when I increase my limit to 1, how many nodes will I touch; you should be able to answer this question. My branching factor is b, my depth limit is 1, how many nodes am I going to touch; b + 1; 1 node at the top and d nodes at the next level. If I find the goal grid, if I do not find the goal, I increase depth by limit by 1.

So, now how many nodes am I going to touch; b square + b + 1 and that I am always doing depth for search now, notice, I go into one path whenever my depth starts to increase, I backtrack and I go into a different path whenever my depth starts to increase, I will backtrack and then and so on and so forth. So, this is what happens at depth limit 2, I start, I go left, I explore the full tree left, I go right, I explore all the sub trees in right.

And then I finally, I am done expanding, I say I could not find the goal and I keep doing this for L equal to 3, this is sort of what the story looks like. Now, why is this algorithm, a good

algorithm? Let us think about these properties, so first question's first; would it be complete, let us think about this.

(Refer Slide Time: 16:03)



My goal is at depth d, am I ever going to search beyond depth limit d; no, because when I am at depth limit d, I will keep searching at some point, I am going to find the goal and once I find the goal, I am done, I am not going to search anymore, I am not going to increase my depth, I am not going to do any deepening; iterative deepening any more, right okay that sounds like a nice property, it is complete.

What about time now, time must be terrible here, I mean time has to be terrible because I am wasting my time doing the same computation again and again and again in fact, how many times will I touch the top node, if my goal is at depth d; well, depth limit 0, depth limit 1, depth limit 2 up to depth limit d, how many times am I going to touch the top node; d +1, so I am wasting so much time, what about the next set of nodes.

How many times am I going to touch them; d times because I will just not touch them at L equal to 0 but then I will touch them every other time, next I will touch them d – 1 times and so on and so forth and last set of nodes, I am going to touch them ones. So, my time is going to be something like d +1 into 1 + d into b, why b; because that the next level that the first level I have b nodes, at level 2, at depth 2, I have b square nodes and they are going to be touch d - 1 times.

So, I create this series called arithmetico geometric progression, it is both arithmetic paths; d + 1, d, d - 1 and a geometric path; b, b square, b cube and so on, believe it or not, if you solve this, and you know you are all engineering students, so you have done this in your 11th and 12th time, this will end up being order b to the d, you can solve it, right, you know how to solve it, so leave it as homework.

But what is the intuition, why does it feel (()) (18:29) b to the d, I mean hopefully they should be many more, right it should be much more, b to the d is only the last set of nodes in my at that depth d but what is going to happen is yes, you have an answer; some of all the nodes at the last level is sort of equal to all the nodes that I have touched in the trees so far, see these are exponentially increasing.

In exponential series, the last layer is sort of dominates everything and so our goal is that the last layer should happen as less as possible as many as fewer times as possible, at the top I have only one node, even if I you know touch it 100 more times or 200 or 1000 more times or d more times or b square more times, it was not change anything, everything is dominated by the last layer; b to the d.

And that I am touching only once and so on and so forth, so that is the intuition why it is okay to repeat touching all the nodes up to this point because I know now, I have to increase my depth because I have to increase my depth all the nodes up to this point will become insignificant and the last; not insignificant but sort of as big as everything as that I am going to see afterwards and so it is not going to hurt me in time complexity, very good.

So, therefore my time complexity is actually pretty amazing, it is b to the d, it is sort of same as time complexity of breadth first search of course, constants are different. What about space complexity; same as actually not exactly same as DFS, so DFS time complexity was b times m, is there any relevance of m in iterative deepening search, what is the maximum depth I am going to search to; d, space complexity is bd, very nice.

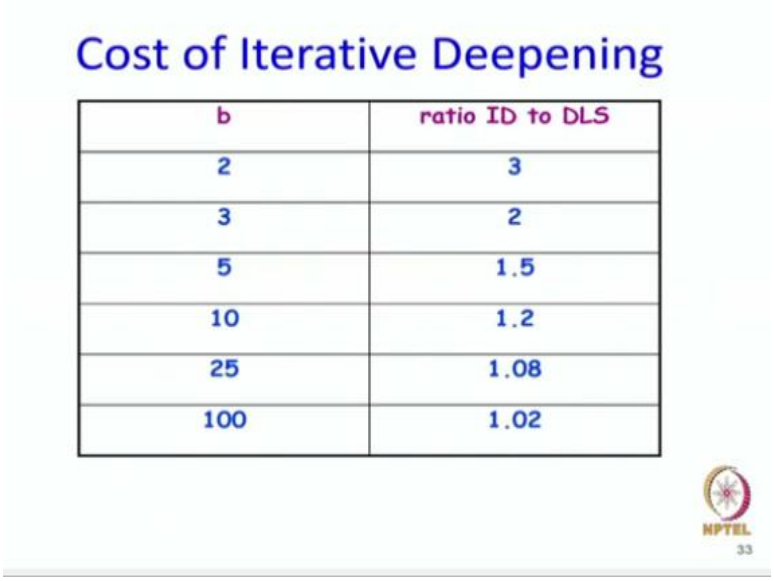
And what about optimality; yes, if when is breadth first search optimal, all costs are the same, all cost are 1 whatever, right. So, only in the case where all the step cost are equal only then will I say that the optimality is there, otherwise optimality is not there. Now, it is for you to think about but can I change this algorithm, so that it is cost optimal also, cost optimal means if the cost are varying, so there we need to do iterative deepening over uniform cost search.

And in uniform cost search, if there is this notion that I am exploring all nodes up to a certain cost before I explore another node of a different cost, so like I had depth bound here, I can have a cost bound there and that algorithm is called iterative lengthening. It can be modified, iterative deepening depth first search can be modified to explore uniform cost tree and that is called iterative lengthening, okay.

And last but not the least is the systematic algorithm that means, do I touch every node only once; of course, not, so this is an algorithm which is not systematic but who cares; it is better systematic is just a property, it is not necessary a strength or a weakness basically, what we are saying is that space is good, time is alright, it is complete and it is decently optimal as at least step optimal and so it is a good algorithm.

Now, you may wonder what is the cost of iterative deepening right, what about comparing it to depth limited search right, how bad is it?

(Refer Slide Time: 22:40)



And notice that this is the constant factor here, ratio of iterative deepening search to depth limited search up to that depth, now you may not know the depth that is okay, suppose somebody told you that your goal is at depth d, I told you the depth, then you can just to depth limited search for that particular depth and you will find the goal versus if I did iterative deepening, I did started with 0, then 1, then 2 and so on.

How much will the constant be it and the constant will be pretty bad, if b is small that 3 times, iterative deepening search may be about 3 times depth limited search but it is not terrible, 3 time is okay but as b increases, as your branching factor increases the constant reduces drastically and in fact, if your b is terrible like 100, then the ratio between the 2 would end up being rather small, okay.

So, therefore iterative deepening search is an algorithm that we would implement if I was ever in a situation, where I had to solve this problem.