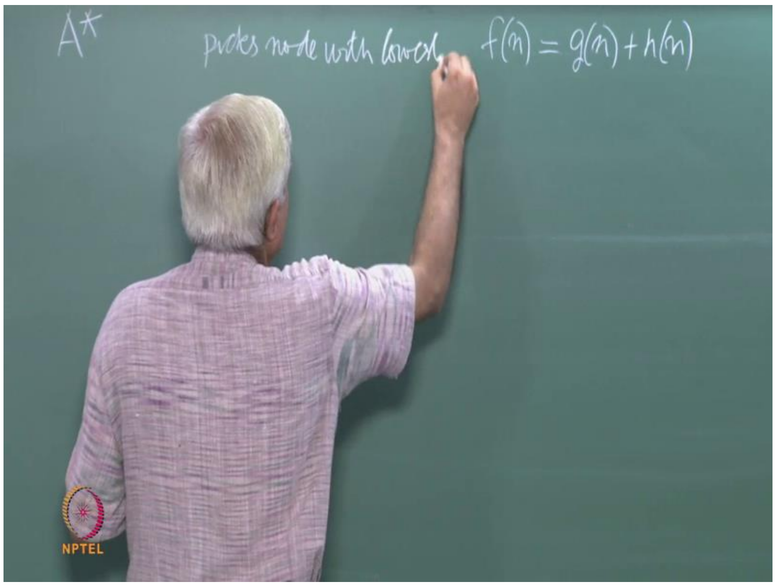
Artificial Intelligence
Prof. Deepak Khemani
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

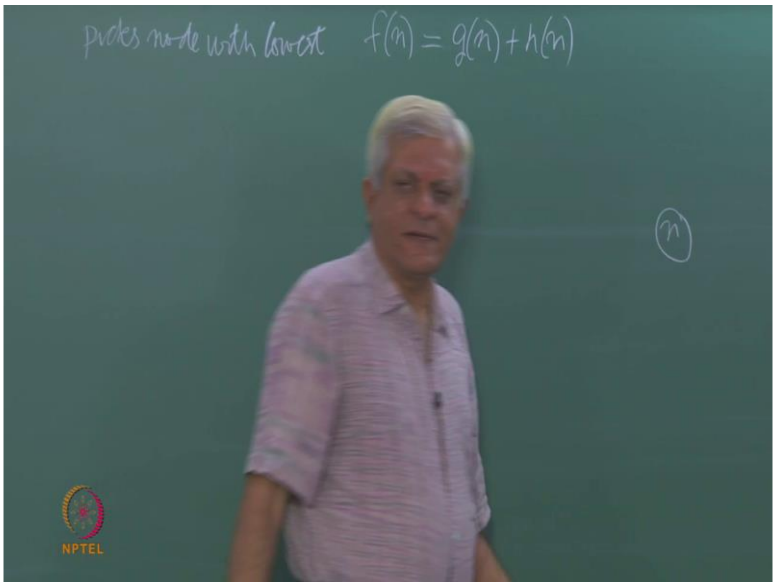
Lecture - 20
Admissibility of A*

(Refer Slide Time: 00:19)



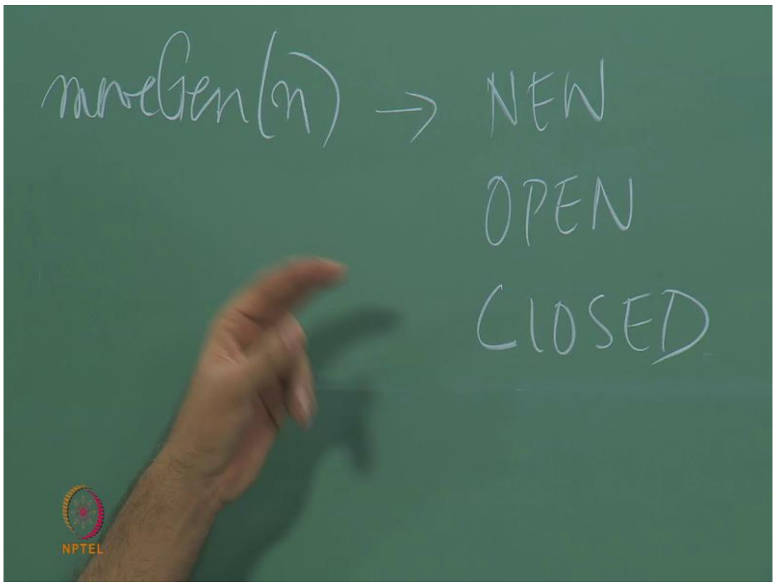
So let us begin, we are looking at the A star algorithm, and if you remember what A star does is that it uses a function f n is equal to g n plus h of n, where g n is the known cost up to node n from the start node, because it has explode part of the space and h n is the estimated cost from node n to the goal node essentially. So, it uses the combination of these two functions; and basically picks node with lowest f of n at every point of time from the open list essentially. So, if you remember g n was the function which we sort of inherited from branch and bound which tries to keep it as close to the goal as to the start at possible, and h n is the function we have inherited from best first search which tries to pull it towards the goal node essentially. So, with the combination of these two functions, the idea is that the algorithm will find optimal paths from start to the goal essentially and today we will show formally that this is how this is done; but before we do that, just a quick recap.

(Refer Slide Time: 01:55)



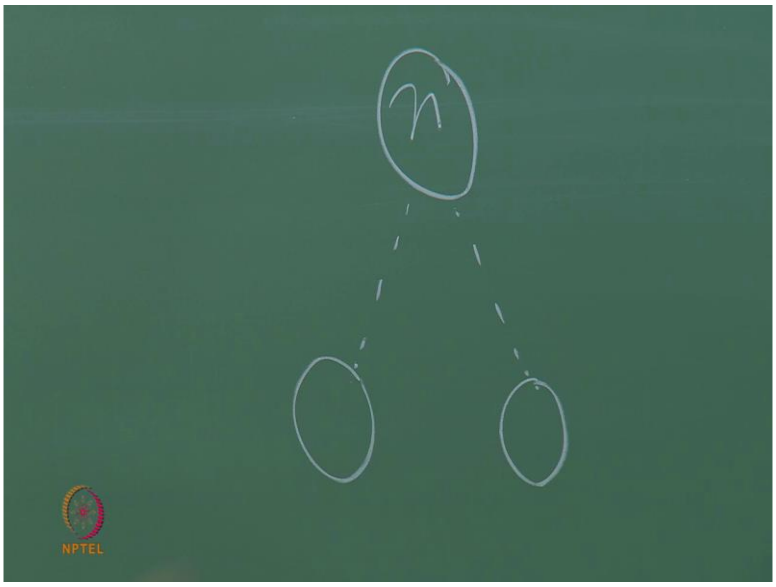
Any time it generates a new node n, it gets different kinds of children - three different kinds of children.

(Refer Slide Time: 02:04)



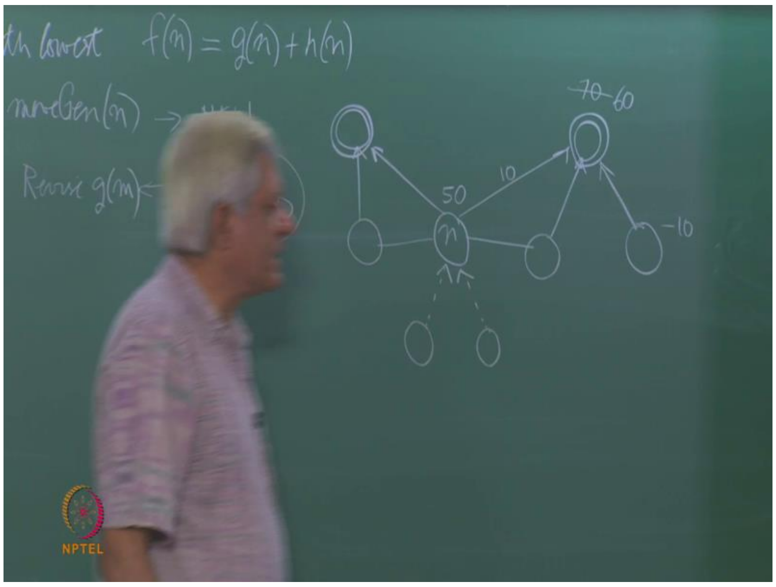
So, we had said, so, when we say moveGen it gives new nodes, it gives nodes on open and it gives nodes which are already on closed. So, when you generate, when you expand the new node which means you call the move gen function with this node n. It can generate three different kinds of nodes either they are completely new or they are already on opened or they are on closed essentially.

(Refer Slide Time: 02:38)



So, completely know new nodes are like this, and what we do is essentially compute the g values for these nodes, which is the g value of this node plus the cost of this arc essentially and put it into open; and we markup parent pointer from these nodes to this node n; so, these steps we do. For nodes which are on open and on closed we need to revise g of m where m is a child, because we make might have found the cheaper path to a node.

(Refer Slide Time: 02:57)

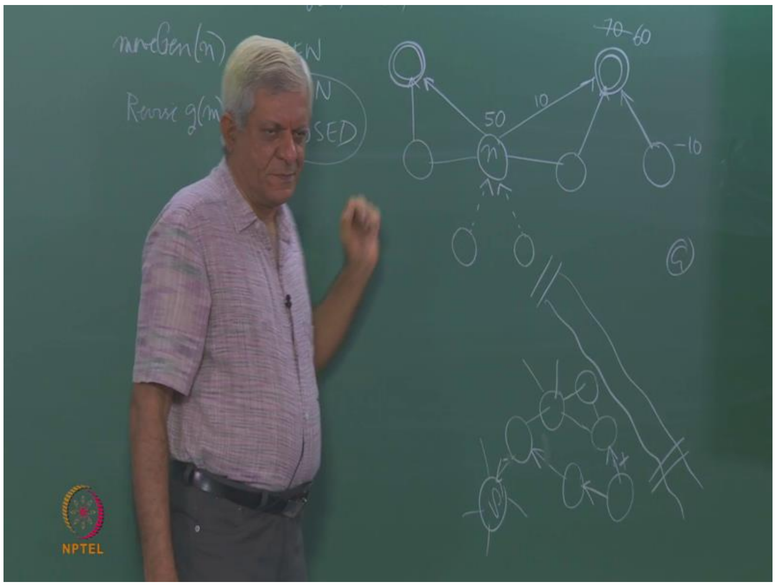


So, for example, if this is a node on open, these two nodes are on open, there is possible. So, since they are on open it means that they already had some parent in the closed of it is, it was a child. So, this could have been this one this could have been this one; remember that this double cycle we will use for closed nodes. So, they already have some parent, but we have found a new path to these nodes on open; it is possible that we might have found the cheaper path and we want to update the cheaper path. So, essentially we revise g values of node essentially and lastly you may get nodes which are on closed. So, for example, these two nodes, in which case not only you have to update the g values of those two nodes which are on closed, but also any nodes which might be pointing to this essentially.

So, for example, if the cost if the g value of this is let us say 50, and the edge this edge is let us say 10 and if this g value was 70, some for some reason then we need to update that g value for that node, because now we have a cost of 50 plus 10 – 60, so we want to make this 60. And we have gained an value amount of 10 which must be passed on to this, so this become must become minus 10 whatever it is value was it must become minus 10 and any children it might have and thinks like that. So, you need to propagate that.

Now why do we need to do this updation? open and closed list why do we bothered to update this g values in finding optimal solutions or the shortest paths essentially.

(Refer Slide Time: 05:21)

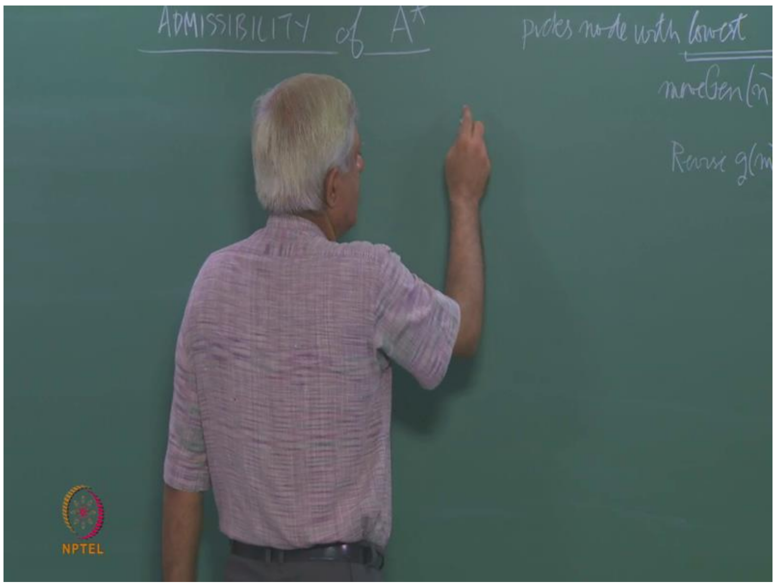


So, by as and when, so let us take this example that we have been looking at if this is a river, and you are at this place, which is the start node and your goal is somewhere here on the other side of the river, we have looked at this example before when you look looking at best first search. So, you may generate some children and you may follow some path because the heuristic function is driving it in this direction; or at that point you can see that there is the, the river on the way and let us assume that the bridges somewhere here or there is another bridge somewhere here, then the search will drive it towards that path essentially. So, it may go from here to here, and then it may go from here to here.

Now best first search what have done the same thing, but it would have remember this as the path which takes you to the goal node, and that is why we said that best first search does not guarantee the optimal paths. Where as so this was the this is the path that you have you have just found for this node, this is the new parent that you have just found it is possible that as search progresses you may generate a node here, and you will find that the shorter paths to this node is via here and not like this. So, this is the kind of situation we are trying to depict here; that you may want to revise you may want to say this is not the parent, but this is the parent and because there is a shorter path here. So, may be you found a path like this.

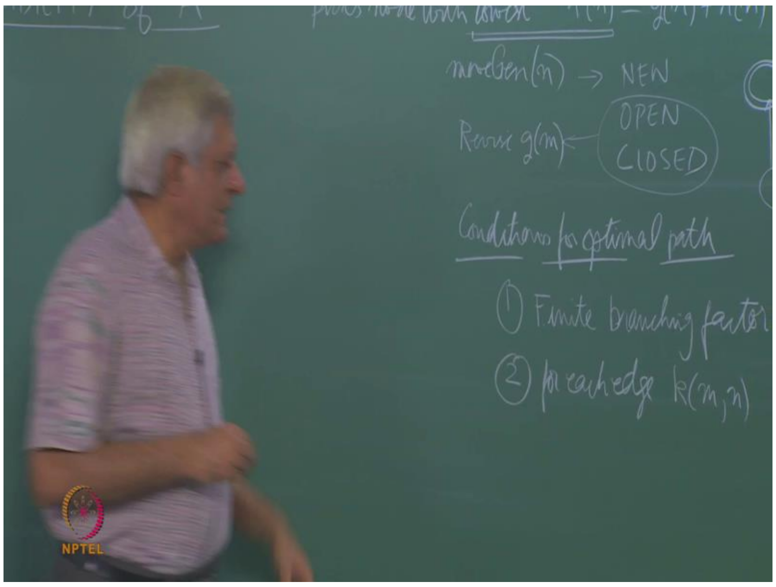
So, the paths that you are looking for may be something like this; go like this; instead of the search would have generated first gone in this direction, because that is what the heuristic function is saying the goal is here go towards the goal, and then it will away from that path essentially. But a star does is that when you does something like this it reallocates parents too shorter paths to any node on the way. So, which is why we say that whenever we expand this node n for nodes on open like these 2 or for children on closed like these 2; if you have found the shorter path, we must update that essentially and that is 1 reason why it ends of finding the shortest paths essentially.

(Refer Slide Time: 07:46)



So, today we want to prove that A star is admissible and by this we mean that it will guarantee an optimal solution or or shortest path if you trying to think of it as a path in the graph, the graph may be an abstraction for some other problem where the cost could be something else, but basically every edge this is the cost associated with it. So, under what conditions will this algorithm...? Now just think of this purely has an algorithm and what is the algorithm the algorithm says that pick the node with the lowest f value from open, expanded, this is what we are doing here, and treat these three different kinds of nodes new, open and closed separately. For new nodes just set up the graph for nodes are opened check if you have found the cheaper path; for nodes on closed also check whether you have find found cheaper paths, but also propagate the cheaper path like that example to it is the children of the closed node, because the node is on closed it will have some children.

(Refer Slide Time: 09:23)



So, under what conditions would a star guarantee an optimal path we have stated one condition earlier. So, let me write this conditions for optimality, the first condition is finite branching factor every node will have a finite number of children essentially. Now obviously you can imagine some continuous problems, where there may be infinite neighbours; now, if there are infinite neighbours then we cannot use this algorithm we will assume that every node has the finite number of children. So, the branching factor is finite essentially that is one condition.

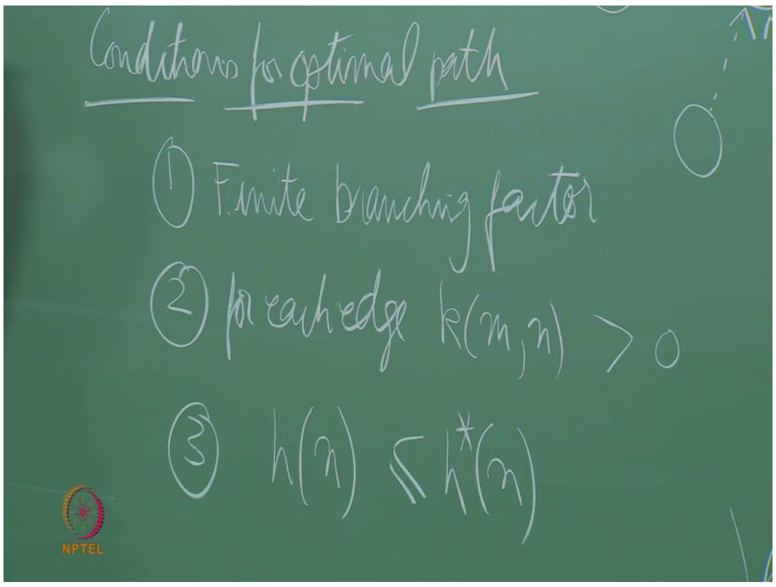
We will allow infinite graph which means that we will allow for infinite number of nodes to be presented in the graph, but branching factor cannot be infinite which mean that you can always move away from some node; if the branching factor was infinite then you know you will spend all your time infinite amount of time generating the children of that node essentially; so obviously, it would not work. There is a condition on the cost; so we will use k for cost function. So, k m n stands for cost of edge going from node n to node n, we will assume this is the symmetric cost, what condition should look cost function satisfy, for the a alba for the algorithm to guarantee optimal solutions.

So, can I allow negative cost? Why?

Sorry

If there are cycle with some of it is say just negative, then the algorithm will just keep repeating the cycle and it is total cost will keep going down essentially, which is also conditions with the algorithm demands algorithm works only for positive cost there are other algorithms which handle negative cost, but we are not looking those kind of thing. So, for the moment, we will assume that this should be greater than 0, where we will not even allow 0 cost essentially. So, that is for the moment know but will come back to that little bit later.

(Refer Slide Time: 12:02)



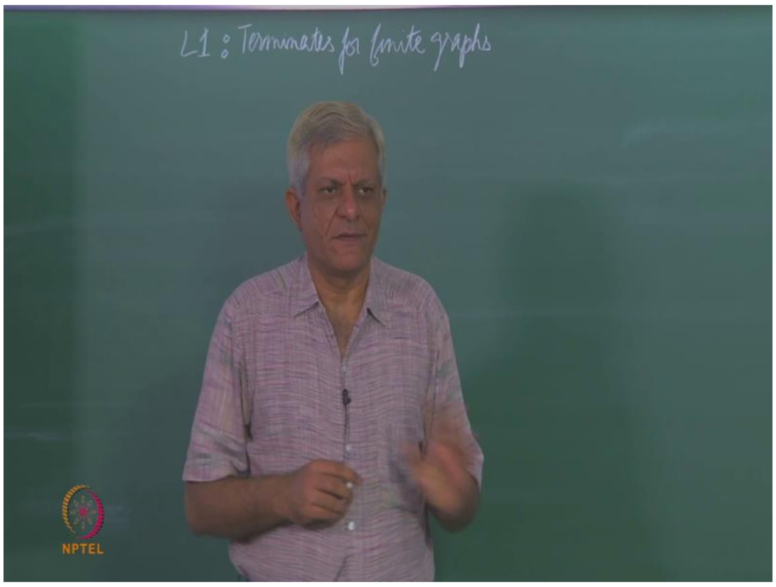
Conditions for optimal path

1. Finite branching factor
2. for each edge $k(m,n) > 0$
3. $h(n) \leq h^*(n)$

A third condition that i have already mentioned and we have seen as example what it which say that h of n should be less than or equal to h star of n, which means that the heuristic function must be an under estimating function, it must only always under estimate the distance to the goal, never over estimate the distance to goal. And the intension behind that is if you over at estimates the distance of some node from the goal then you will never explode that node further, because you will think it is too far from the goal essentially. So, in generally if you think it is closer to the goal where it actually is then you are liked to explore that node and that is a key factor for A star being admissible.

So, we if you remember we saw an example in betweens of 2 heuristic functions h 1 and h 2, one of them underestimating the distance, other was the overestimating the distance and both of them had a wrong notion of which of those two candidates was closer to the goal, but the underestimating function did find that optimal path and the other one did not; today we show this little bit formally. So, if these three conditions are true then a star is admissible and admissible we mean it always finds a optimal path if there is if there is a path from start to the goal, then it will find the optimal path from start to the goal and it will terminate to the goal. So, we will do this through a series of small lamas through a series of statements then eventually conclude with this property of admissibility and the couple of properties we will compare heuristic functions to see if how can we say that one heuristic function is better than another heuristic function.

(Refer Slide Time: 13:57)



So, we will have a series of lemmas, so L 1 simply says that it terminates for finite graphs; the algorithm will terminate for finite graphs; that is the only statement we want to assert at this movement. What is the justification or what is the reasoning to say that it will always terminate for finite graph?

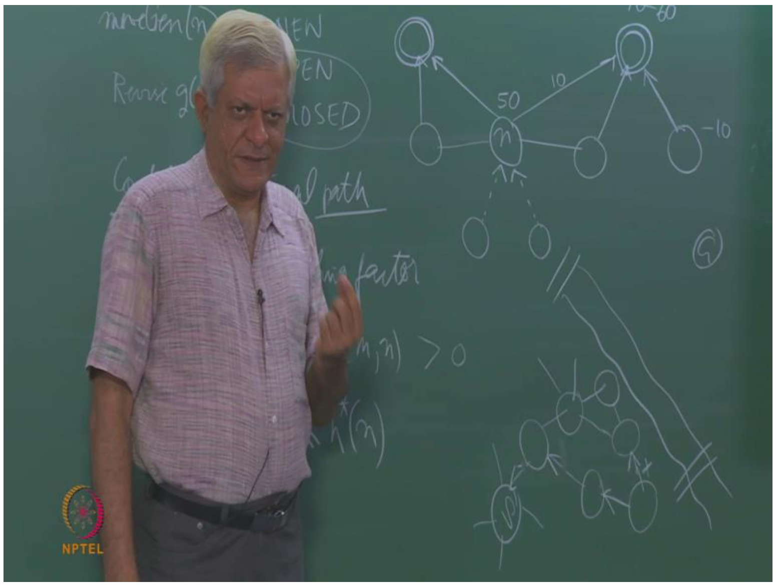
( )

Yes, but that is not to the justification that we are looking for. You should at a algorithm look at a algorithm and comment on what the algorithm is doing.

( )

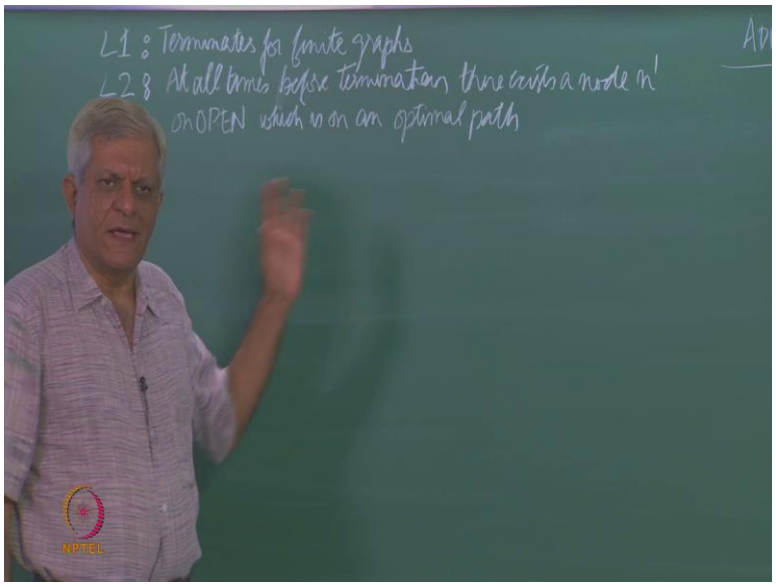
If the graph is finite then the algorithm in every cycle moves one node from open to closed it starts by putting the start node on open list and closes empty and in every cycle it inspects one node from open the one, which has the lowest f value and if it is not the goal it puts that into closed and generates it is children and puts those children into open, which is the priority cubes on f. So, if the number of nodes are finite in the graph, if the graph is finite then this it can do this only a finite number of times. So, it will exhaust the whole graph. Now supposing the supposing in this problem that we have here, which is this problem how this city travel supposing this bridges were not there essentially, and you know there was a small hamlet here, which you could only explore that essentially then the graph then A star would end of exploring all those nodes and say i have nothing else explore because open has become empty.

(Refer Slide Time: 15:54)



So, remember there are two conditions for exit; one is when open is empty exit and report failure, other condition is if the node that you have picked is the goal node, then we construct the path turn it on the path. So, in this case, if these bridges are not there and the goal was there it would never a find the path, but it would terminate that is all we are talking about here essentially; the graph is finite, it will terminate.

(Refer Slide Time: 16:38)

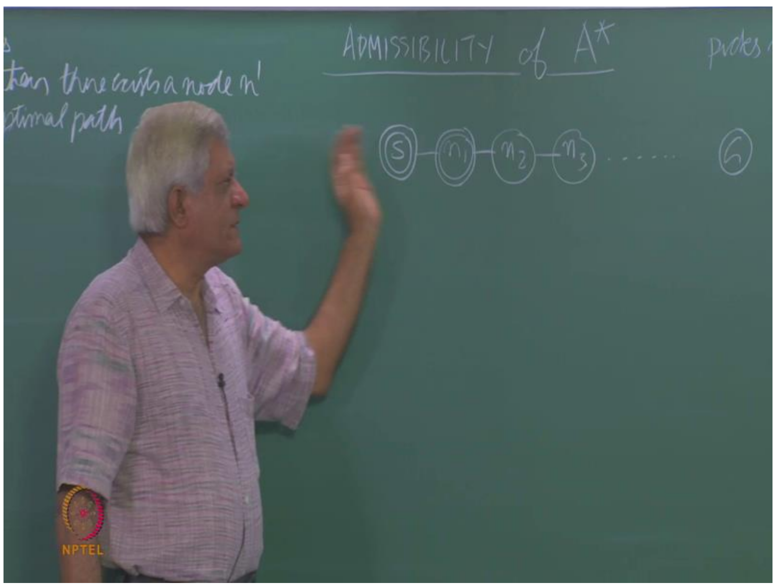


L1 : Terminates for finite graphs

L2 : At all times before termination there exists a node n' on OPEN which is on an optimal path

Then the second lemmas you want to says that at all times before termination there exist a node we call this node n prime; and this node is on open and which is on the optimal path. So, I will just say an optimal path, because a could be more than one optimal paths. So, the next statement that we are making is in that before the algorithm terminates, it will always have in it is open list one node, which we will call n prime which is on the optimal path from start to goal essentially.

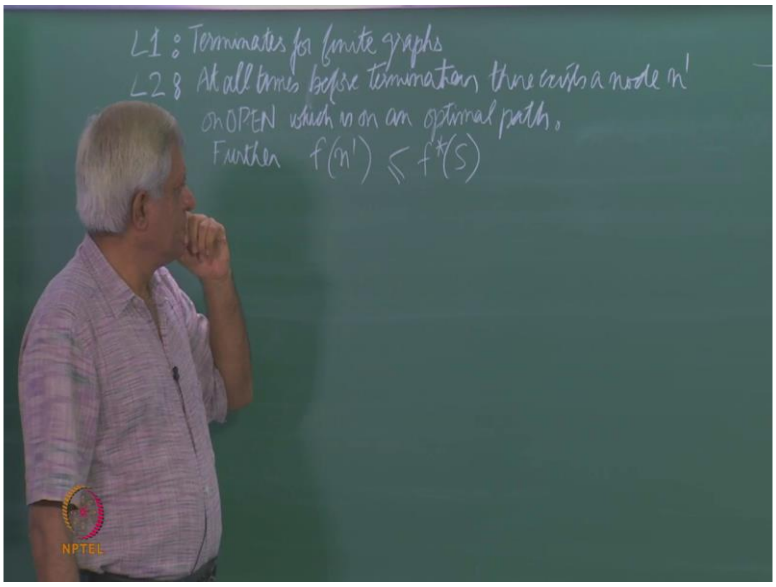
(Refer Slide Time: 18:00)



And the argument for that is the following that initially we put s on to open. So, let us say this is the optimal path, this is the optimal path, let us say there is only One optimal path to start with n 1, n 2, n 3, let us say this is the optimal path. What we are claiming now that from this path before the algorithm terminates, they would always be one node in the open list, which means the optimal path will always be in the sites of this algorithm in some sense. Now when we start the algorithm we put s on to open list. So, at the beginning s is in the open if the algorithm is terminated by finding the path then it is terminated we are not talking of that condition essentially. So, we as talking about it condition before termination which means there is a paths to the goal we assuming there is a path to the goal and it has not yet found a path or the path whatever you want to say we will show.

That when it finds the path it will always finds an optimal path. So, initially we put s on to open. So, the condition is true in the initial situation when we remove s from open we put it into closed and then we end of putting n 1 on to open essentially if we remove n 1 from open then you put this into close and this process will continue essentially if we remove g from open which is the last in this line; that means, algorithm is terminated by founding a finding a path essentially. So, this condition says that at any time 1 of these nodes is always in the open which is available to a star to inspect next.

(Refer Slide Time: 19:57)



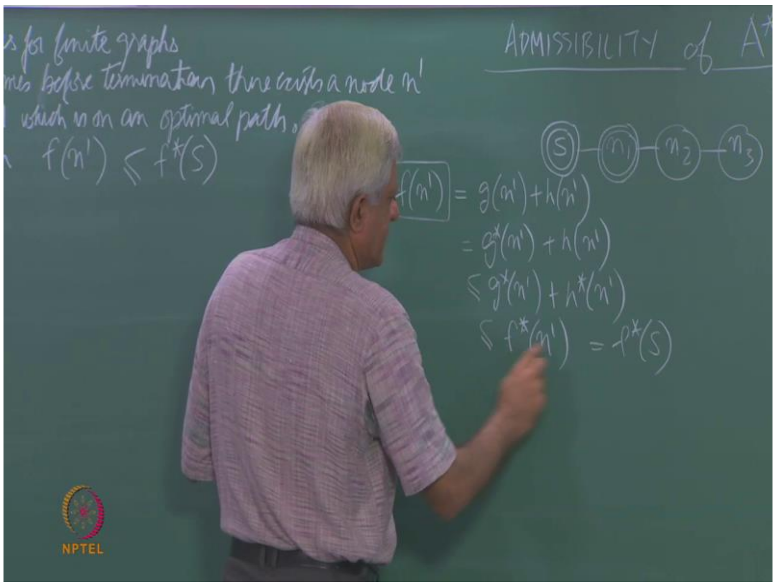
L1 : Terminates for finite graphs

L2 : At all times before termination there exists a node n' on OPEN which is on an optimal path.

Further $f(n') \leq f^*(S)$

Further we say the f value of this node is always less to or equal to the optimal cost from. So, s is the start node, and remembers, s star is a optimal cost of going from start to the goal node essentially. So, this is this is part of the statement, we are saying that not only the such a nodes node exist it will always have f value which is lower than or equal to the optimal cost.

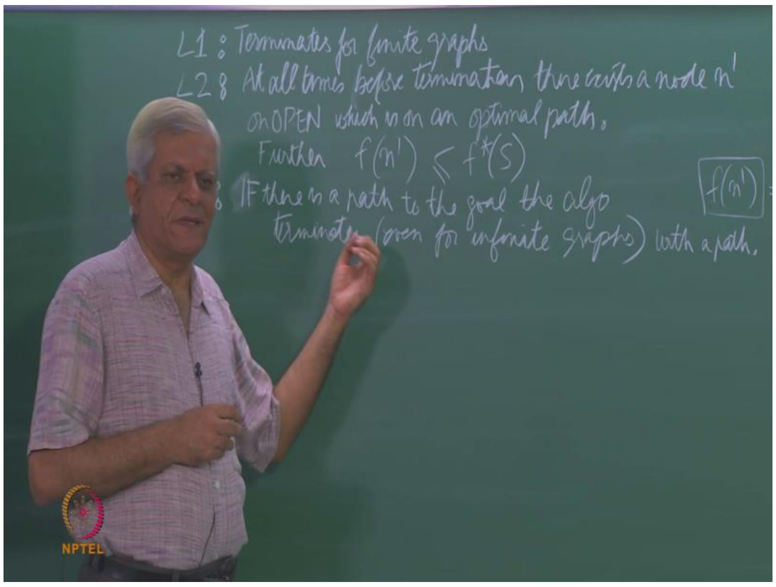
(Refer Slide Time: 20:36)



So, what is the argument behind this you can say that f of n prime is equal to g of n prime plus h of n prime it is by definition of what the f value is now, because it is the known optimal path we know that this is g star of n prime plus h of n prime. Remember that we are talking of optimal path. This will be less than equal to g star of n prime plus h star of n prime, because of the condition, condition number 3 which we are said that the heuristic values always less than or equal to the optimal value, which means this is equal to less than or equal to f star of n prime, where f star of n prime is the cost of the optimal path which passes through n prime; not necessarily the globally optimal path to the optimal path passes through n prime. But these n prime that we have chosen is on the optimal path itself; so this must be equal to f star of s because after all it is it is the same paths that we are talking about.

So, whether we say it is a passing through n prime or a. So, whether we say it is passing through s or n 1 or n 2 or n 3 or g it does not matter the cost is in the same. So, this is the condition that we are looking for; and this is less than or equal to this f star of s that is what we have written.

(Refer Slide Time: 22:20)



L1 : Terminates for finite graphs

L2 : At all times before termination there exists a node n' on OPEN which is on an optimal path.

Further $f(n') \leqslant f^*(S)$

L3 : IF there is a path to the goal the algo terminates (even for infinite graphs) with a path.

$\boxed{f(n')}$

Then the third thing we want to say is that algorithm terminates if there is a path, and it should had even for infinite graphs remember that an infinite graph is only be that which has an infinite number of nodes factor. So, this the third statement we have making is that the algorithm terminates with a path; that if there is a path from the starts to the goal state, the algorithm will always terminate with the path from the start to the goal state we are not yet talked about optimal path, we are saying the it will find the path; and we are saying making this claim that even of the graph is infinite when you say that infinite number of nodes, then algorithm will find a paths to the goal. What would be an argument for this? In other words we are saying that the algorithm will terminate that picking the goal node of the two termination criteria that we have one is the open is empty in this for infinite graphs open can never be empty. So, for infinite graphs it can only terminate by picking a goal nodes essentially.

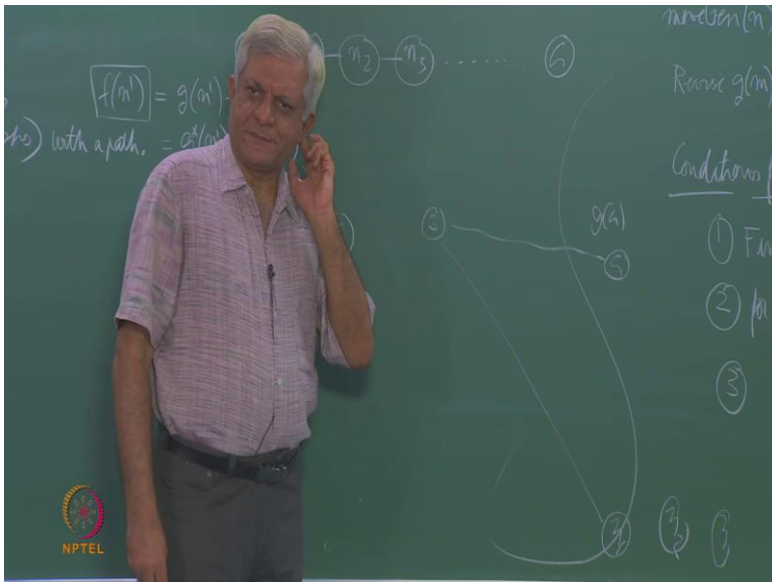
So, we are saying you want to argue that it will necessarily pick the goal node at some point of time just you to imagine this graph just imagine it doing a search in a city or something like that, how the open front yard will move essentially. You remember that it always picks the node with the lowest f value essentially for a goal node the f value is equal to the g value of that node, because h is 0 you already at the goal. So, we want to argue that this node is the goal, which is the goal node will the at some point come to the head of this sorted list or at the at the root of the way of maintaining the priority queue.

In other words we are saying that all other nodes on open will at some point become more expensive then this node g. And why is that?

(( ))

No no no . So, what I am saying I am saying that let us say this is my start node, and this is my paths to the goal it has some cost essentially. What I am saying is that this search cannot ignore this whole node that eventually it may have it may end of looking at a node here.

(Refer Slide Time: 25:41)



It may end of looking at a node there in this direction or that direction, but it cannot go indefinitely in any direction. And the reason for that is that whichever direction it goes, the g values, think of the g values supposing there is a node here let us call it n 1 or something if that is on the open list. So, let us say the open is looking something like this, you know refusing to go towards the goal node, but there is a node n 1 g value? What is going to be this g value of this node is going to be the length of the path from here to here, let us say it picks this node n 1 it will have some other node let us collate n

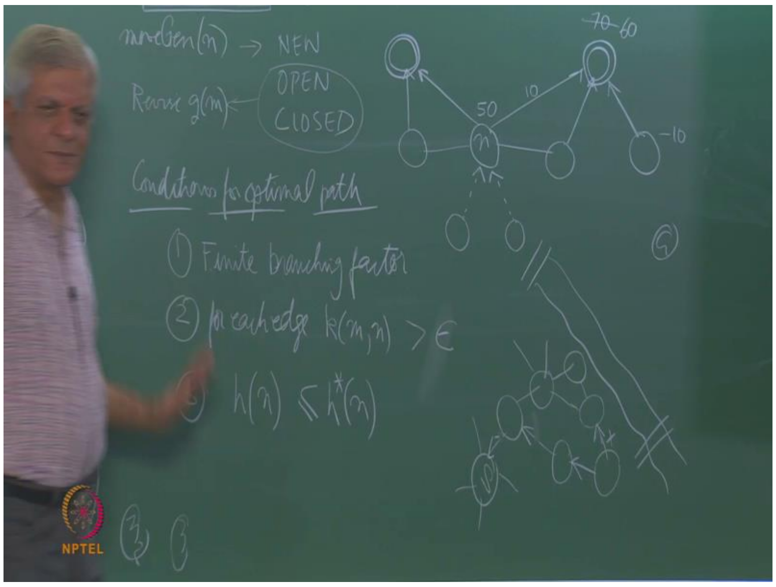
2, and then it let us say picks n 3, can it keep doing in can it keep going in indefinitely in along some other direction, because the g, because the paths are becoming longer and longer in those other direction; g value remember is an is an length of the path found up to that point. And at some point this f value will become the smallest that is say condition that we have that the paths are edges are positive in cost essentially. So, if you go in some direction and some point your cost will becomes. So, what is the value of this sum g of g the some value at some point all other nodes that you have in open will have paths which are longer than this; or to think of it in another manner, there are only a finite number of paths which are shorter than this lengths g of g; and even if the algorithm will explore all those paths and then it will take of g of g. Is this they seems to be a gently silence about this essentially.

All we are saying is that if you go of in some other direction your path cost will become more than the cost of this g value and remember that the heuristic function underestimate. So, you can never over estimate the cost to in any case g will come at some point and the some point this will become the smaller. So, it will have to pick the node. So, as it turns out this was many years ago, when I was teaching the same very point in the class 1 of the students in the class, he is now professor in Princeton raises hand, and said this is not correct; and it something you do with this condition that is why i said for the moment let us assume this this condition is the paths the edge of e, the cost of each edge is greater than 0.

So, it is possible to assign edges, edge cost for example, I will assign 1 to the first edge, half to the second edge, one-fourth to the third edge 1 8 to the fourth edge and keep dividing by 2. What will happen, I will have an infinite path whose total path length will be finite essentially, and that finite could be less than this g value, which means that it could get actually trapped in that infinite path essentially, which has the finite total cost. So, this is like you know that paradox is that (( )) in the hair paradox that epidemics paradox. That you say that the diabet, the tortoise tells the rabbit that i if you give me a lead of let us 100 meters then you can never beat me in the race and this argument is the following.

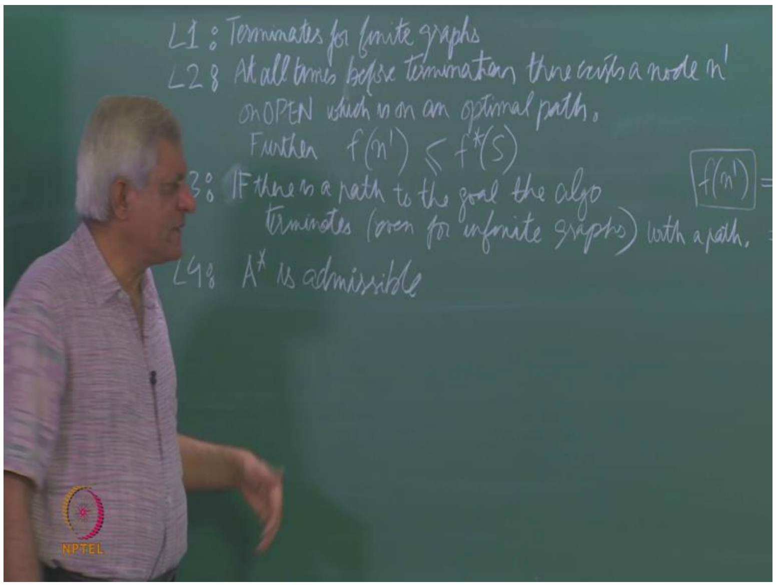
So, let us say rabbit is at starting point and the tortoise is 100 meters away, the tortoise say that by the time you come to this 100 meters point, I would have to move to some point let us call it x 1 essentially and by the time you come to x 1, I would have to move to x 2 and by the time you come to x 2, I would have to move to x x 3 and this will ofcourse keep happening infinitely and so you can never over take me essentially. So, is 1 of the paradoxes which comes into picture and something similar would happen if you go to allow arbitral small edge cost essentially. So, if you cannot allow arbitral small edge cost then this will not happen.

(Refer Slide Time: 31:04)


moveGen(n) → NEW
OPEN
CLOSED
Revise g(m) ←
Conditions for optimal path
① Finite branching factor
② for each edge k(m,n) > ε
③ h(n) ≤ h*(n)
~~70~~ 60
50
10
-10
n
G
NPTEL

So, which is where the correct condition is not greater than 0, but greater than some value some positive value and as long as that is the case, it cannot be infinite especially small then every path fill at some point become longer than this path to g, and then g will come into. So, i will take it that this point has been accepted.

(Refer Slide Time: 31:30)



L1 : Terminates for finite graphs

L2 : At all times before termination there exists a node n' on OPEN which is on an optimal path.
Further $f(n') \leq f^*(S)$

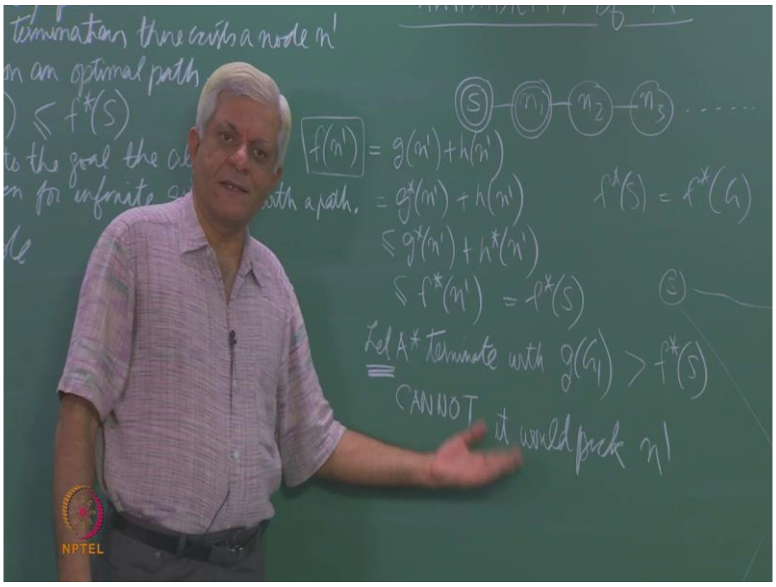
L3 : IF there is a path to the goal the algo terminates (even for infinite graphs) with a path.

L4 : A^* is admissible

$\boxed{f(n')} =$

So what have we done so far? So far we have said between statements 1 and 3 1 means said that the algorithm always terminate; in 3 means said if there is a path to the goal it will always find the path to the goal. Now we are saying that it will always find the optimal paths to the goal essentially. And it has very simple argument, the argument is as follows that so the proof for this I will write here.

(Refer Slide Time: 32:13)



termination there exists a node n' on an optimal path

$) \leq f^*(S)$

to the goal the al

en for infinite with a path.

$$\begin{tikzcd} S \arrow[r, dash] & n_1 \arrow[r, dash] & n_2 \arrow[r, dash] & n_3 \arrow[r, dash, dotted] & {} \end{tikzcd}$$

$$\begin{aligned} \boxed{f(n')} &= g(n') + h(n') \\ &= g^*(n') + h(n') \\ &\leq g^*(n') + h^*(n') \\ &\leq f^*(n') = f^*(S) \end{aligned}$$

$f^*(S) = f^*(G)$

Let A^* Terminate with $g(G) > f^*(S)$

CANNOT it would pick n'

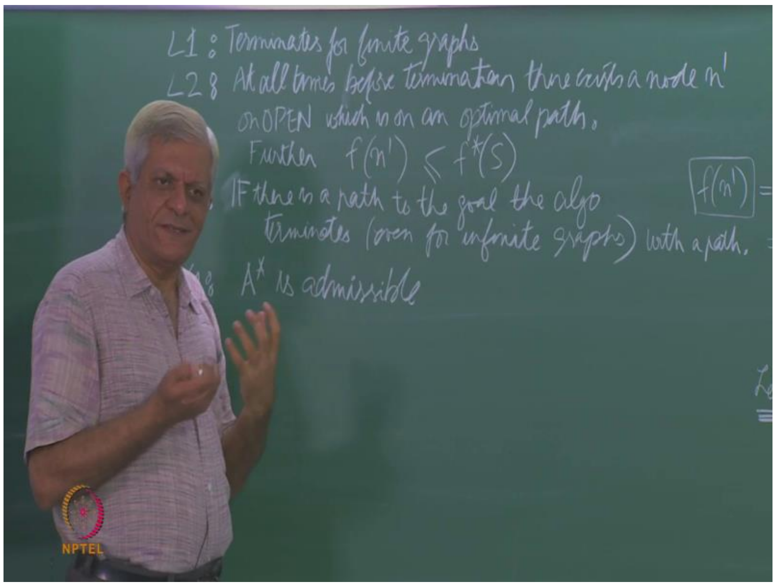
Let A star terminate with a non optimal path, which means it is found some paths to the goal where the g value of that path. So, let me just call a g 1 is greater than f star of s, remember f star of s is the optimal path; whenever you use the star it is an optimal path, whether we know in the value or not essentially. So, we are saying that let proof a contradiction that a star terminate with a by picking a goal node g 1, whose g value is more than the optimal cost, and now I want to make a statement that this cannot happen let me ask you, why can this happen, why can it not terminate by a finding a non optimal path.

So, if it is picked goal node it will terminate that is the algorithm; that you must remember like the algorithm. If it picks the goal node for open; that means, it will terminate; it will say I found the goal and so on. Now we are saying that let it terminate by picking the goal node g 1 such that it is cost is more than the optimal cost; remember f star of f star of s is equal to f star of g, and in fact, it f star of any value on this node essentially, all of them the values are the same, it is cost of the optimal it is a cost of this optimal path. So, we are saying that assume that a star terminates by picking a value which is more than the optimal.

Now what I am trying to say that we cannot make this assumption, I mean it is not the long proof or something it just 1 statement it say that it can never pick this such a node and terminate. So, if you look at the statement number 2 that we have written here; there is reason for every statement. That statement says that at all point before termination there exist a node n prime, which is on the optimal path, and whose f value is less than f star of s. So, so simple answer is it cannot. Why because, it would pick n prime as simple as that the algorithm simply says pick the one of the lowest f value.

So, just imagine there it is about to terminate by picking this goal g 1, and this goal g 1 has it is cost more than the f f f. So, it will never be at the head of the priority queue essentially, because we have said that at all points before termination, there is always a node n prime which is on the optimal path and which is on open. So, f this n prime is on open and it is value is less than f star s.

(Refer Slide Time: 35:21)



L1 : Terminates for finite graphs

L2 : At all times before termination there exists a node n' on OPEN which is on an optimal path.

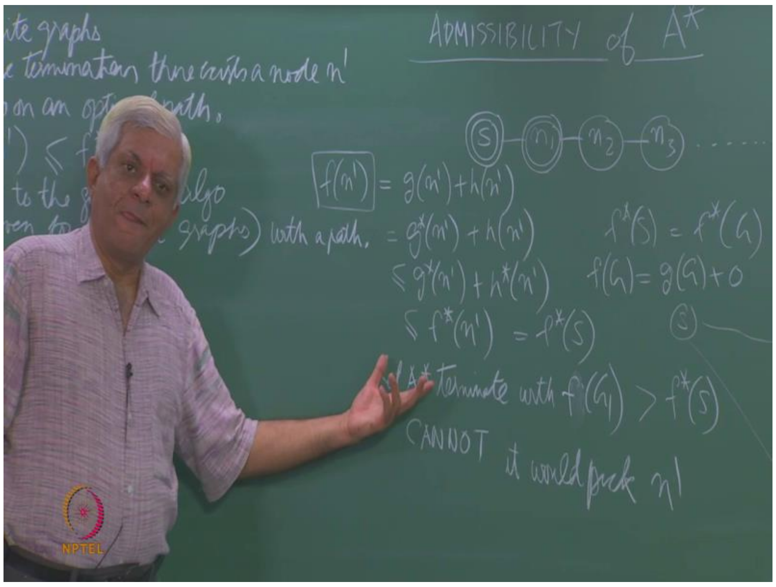
Further $f(n') \leq f^*(S)$

IF there is a path to the goal the algo terminates (even for infinite graphs) with a path.

A* is admissible

So, we it has if a star also choose between these 2 nodes 1 whose value is less than or equal to f star s and one whose value is definitely greater than f star s. Remember I have written g of g i could have written f of g is a same thing, because f of g is equal to g of g plus 0, because h is 0. So, you can write f of g or g of g does not matter, it is a same thing.

(Refer Slide Time: 35:40)



ADMISSIBILITY of A*

$$\begin{tikzcd} S \arrow[r, dash] & n_1 \arrow[r, dash] & n_2 \arrow[r, dash] & n_3 \cdots \end{tikzcd}$$

$$\boxed{f(n')} = g(n') + h(n')$$
$$= g^*(n') + h(n')$$
$$\leq g^*(n') + h^*(n')$$
$$\leq f^*(n') = f^*(S)$$

$f^*(S) = f^*(G)$

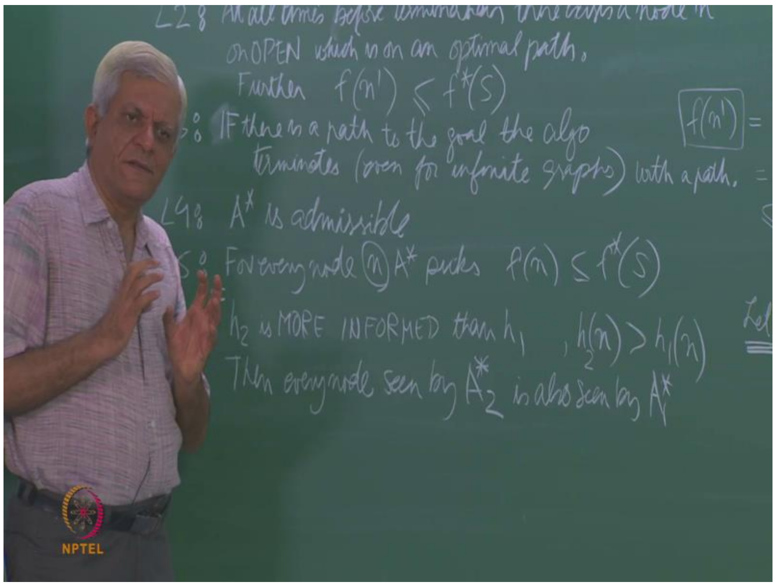
$f(G) = g(G) + 0$

A* terminate with $f(G) > f^*(S)$

CANNOT it would pick n'

So, if a star at to pick between these 2 nodes n prime and g 1, it would pick n prime which mean it cannot terminate by picking g 1 like this essentially. So, the only time it can pick a goal node is that if it is at least as good as the optimal cost node essentially. So, it will always terminate only with an optimal path; is it okay? So, now we want to look at 1 more property and this is as for.

(Refer Slide Time: 36:31)



L2: At all times before termination there exists a node n' on OPEN which is on an optimal path.
Further $f(n') \leq f^*(S)$

L3: IF there is a path to the goal the algo terminates (even for infinite graphs) with a path.

L4: A^* is admissible

L5: For every node (n) A^* picks $f(n) \leq f^*(S)$

h_2 is MORE INFORMED than h_1 , $h_2(n) > h_1(n)$
Then every node seen by A^*_2 is also seen by A^*_1

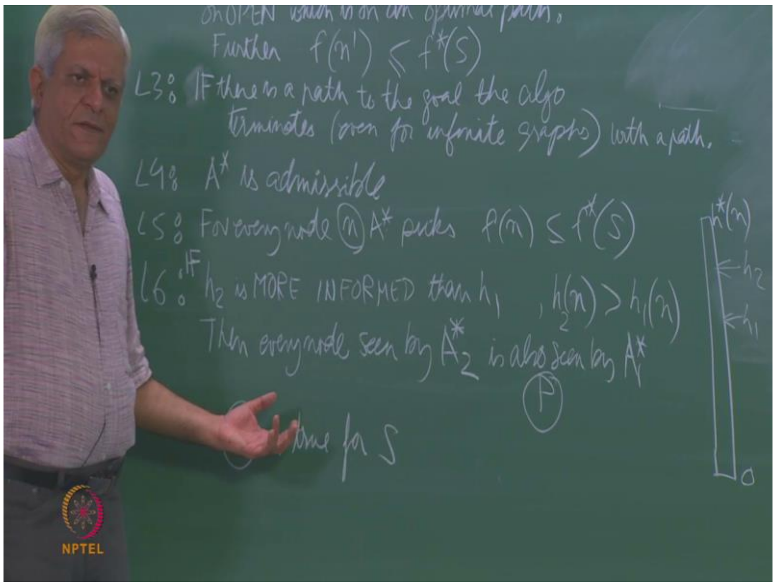
So, one more statement before that property that I am talking about is that for every node N that a star picks f of n is always less than equal to the optimal cost. So, we are saying that a star will only pick nodes whose f value is less than the optimal, all this is coming from fact that the heuristic function is under estimating, because the heuristic function under estimates we are shown that you know nodes like this will always be less an optimal cost or they will always be such a node less an optimal cost.

And now we are saying the same argument as we as we did for this case 4 that if f star is going to pick this node n, it must be better than n prime, because it has picked n and if it is better than n prime it must be less than or equal to f star of s essentially. So, it only pick nodes which the better than equal to optimal cost node. L 6 is the question; so, let there be 2 functions. So, we say h 2 is more informed than h 1 if for all nodes h 2 n is greater than h 1 n. So, if the heuristic value h 2 is greater than h 1 n, then we say that h 2 is more informed. Just a definition, we say then if this is the case then every node seen by a 2 and a 2 is 1 version of a star, which is using this h 2 function is also seen by A 1 star.

So, let us say we have this two versions of a star algorithm 1 uses this h 2 function heuristic function the other uses the h 1 heuristic function and h 2 is always greater than h 1 of n we say that the h 2 is more inform than. So, take the extreme case when h 1 is equal to 0 h 1 is equal to 0 it always thinks that the cost goal is 0 essentially which is what does where as the h 2 thinks that it is at least some value. So, such a function is called and we are making a claim that if h 2 is more inform than h 1 so I should write if here if h 2 is more informed than h 1, then every node seen by a 2 star is also seen by a 1 star, which means a 2 star will see in general a smaller number of nodes, of which means it will explore a smaller part of the search space essentially, it will be more focused towards the search.

So, we will do this proof by induction, and you can see that for the start node does properties true. So, what is the property we are trying to prove that if even if A 2 see some node then A 1 also sees that node we want to prove that let us call this property p that if A 2 sees the node A 2 star why star, because both are admissible. So, what is the situation? Situation is like this for any given node n, there is a band starting from 0 to h star of n.

(Refer Slide Time: 40:46)



on OPEN which is on an optimal path.
Further $f(n') \leq f^*(S)$

L3: IF there is a path to the goal the algo terminates (even for infinite graphs) with a path.

L4: A^* is admissible

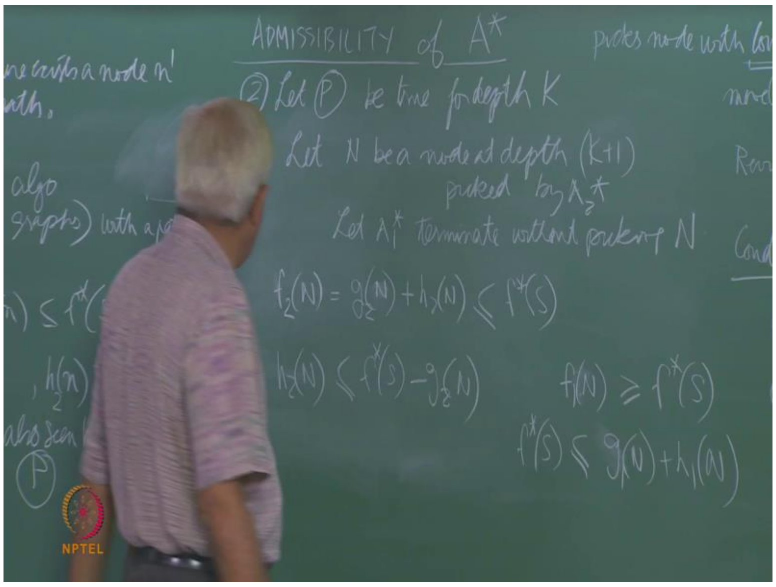
L5: For every node n A^* picks $f(n) \leq f^*(S)$

L6: IF h_2 is MORE INFORMED than h_1, $h_2(n) > h_1(n)$
Then every node seen by A^*_2 is also seen by A^*_1 P

true for S

What we are saying is that h 2 is somewhere here and h 1 is somewhere here this is what we mean by more informed closer to h the optimal value. And we want to make a claim that if this is the situation, then the search space explore by the algorithm using h 2 will be smaller than the search space by algorithm using h 1. And we will this proof by induction. So, you can see that for s. So, we call the statement p; so p is true for s which is a start node. So, we are just saying that if if a 1 sees if a 2 sees the start node and even will also see start which is really true, because both the algorithms are the A star always starts were picking the star node essentially. So, the second part is the hypothesis, which says that let so I will just use this for this whole statement.

(Refer Slide Time: 41:51)

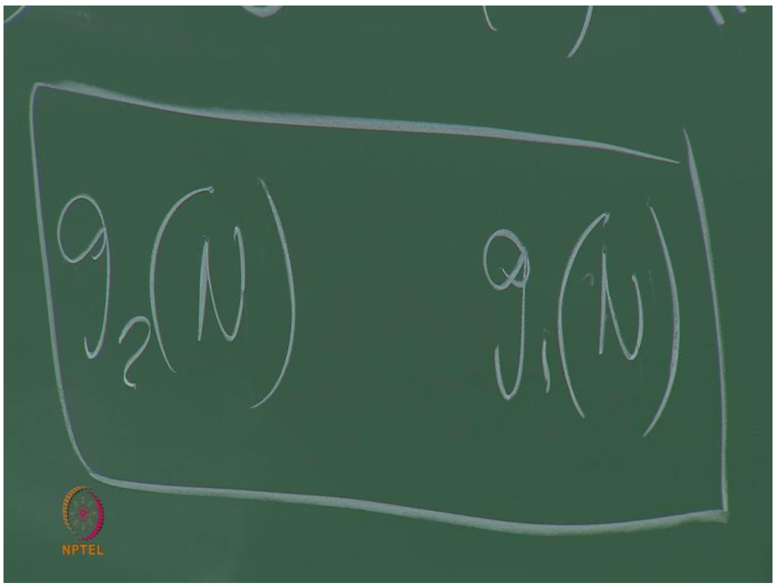

ADMISSIBILITY of A*
② Let Ⓟ be true for depth K
Let N be a node at depth (K+1) picked by A_2^*
Let A_1^* terminate without picking N
$f_2(N) = g_2(N) + h_2(N) \leq f^*(S)$
$h_2(N) \leq f^*(S) - g_2(N)$
$f_1(N) \geq f^*(S)$
$f^*(S) \leq g_1(N) + h_1(N)$
picks node with
NPTEL

So, instead of every let me say any node which is the same thing essentially. So, let p be true for depth k 1 or depth k. Now we want show that this is also true for depth k plus how much is the induction step, and this you will do by contradiction. So, let n be a node at depth k plus 1, which is picked by A 2 star and let A 1 star terminate without picking n. So, the induction step, you will do by contradiction. So, we are making an assumption that there is some node we have call this capital N at depth k plus 1 and we are saying a 2 star has pick that node, but a 1 star terminated without ever picking that node essentially, which means a 1 star found a paths to the goal. Now if you look at this value h f 2 of n, so f 2 is further second algorithm which is using h 2 is g 2 of n plus h 2 of n.

So, we can rewrite this as H 2 of n and this is less than f star of s; remember that any node that you take the statement 5 that view it that will have this value less than f star of s. So, we are just writing f star it has does not matter f 1 or f 2 it is a optimal cost f star is starts for the optimal cost. So, we can write this as h 2 of n is less than equal to less than equal to f star of s minus g 2 of n.

Now even star has terminated without picking this node n, which means that f 1 of n is greater than or equal to f star of s. It cannot be less than f star of s, because is a a 1 star has terminated and it will terminate by picking a node with the optimal cost value, and it has not picked n. So, if this equal inequality must hold; that it must be at best equal to that, but it cannot be less in that essentially. So, this we can write rewrite as following f star of s is less than equal to g 1 of n plus h 1 of n, I am just rewriting this I am replacing f 1 of n by g 1 of n plus h 1 of n, and I just inverted the sign in equality sign essentially

(Refer Slide Time: 46:39)

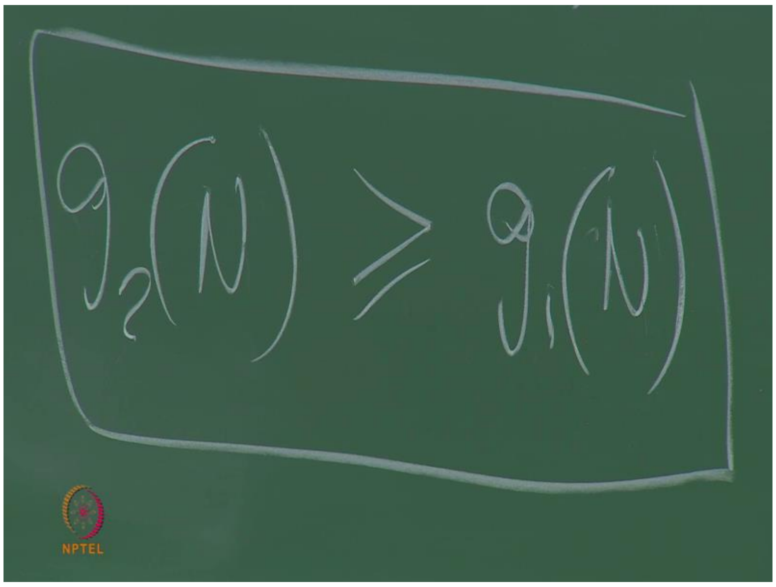


Now the question I want to ask is what is the relation between g 1 of n and g 2 of n this g 1 this g 2 of n and g 1 of n; if I have if I have to put a in equality what would I put or equal to what is relation between this. What should I put, what symbol should I put in between here g 2 is greater than equal to, you are the only one who are saying that.

why is other case?

(( ))Need be optimal there only the path found so far path found so far by A 2 need not be the optimal you are not getting the exactly, but you must remember this statement we made let this condition be true up to depth k the induction hypothesis, which means that every node that is been seen by A 2 up to depth k has also been seen by a 1. So, whichever path A 2 found a 1 would have also seen that path, but may it had seen some other path as well be equal, but we are saying it does not have to be equal.

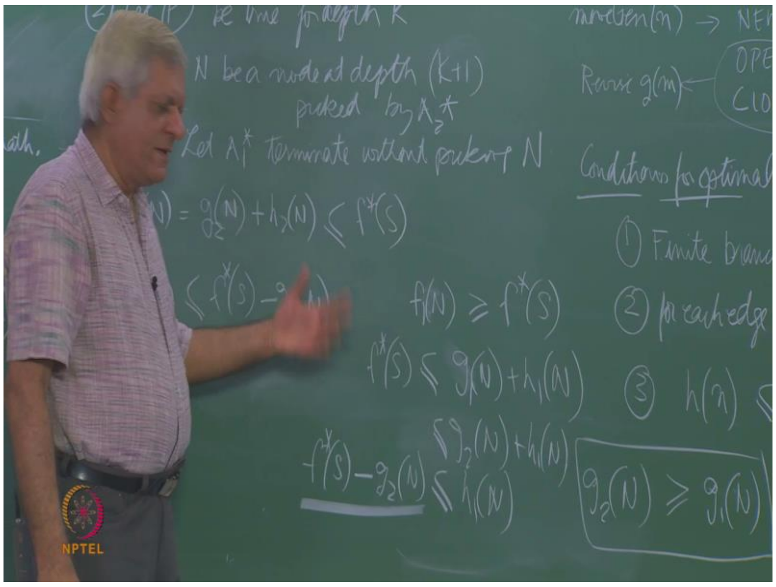
(Refer Slide Time: 47:47)



$$\boxed{g_2(N) \geq g_1(N)}$$

So, if this will be greater than so, is possible that A 1 might find a shorter paths to n because it is explore more nodes on the way to n essentially.

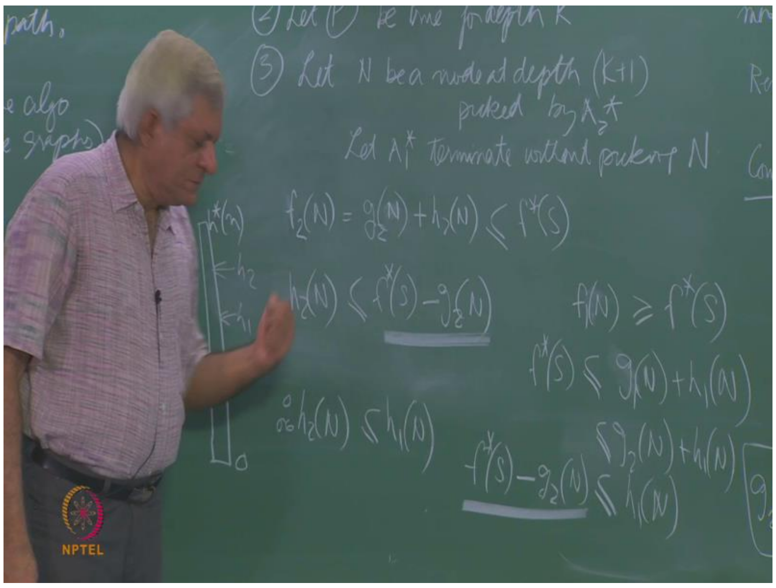
(Refer Slide Time: 48:09)



N be a node at depth (K+1) picked by A_2^*
Let A_1^* terminate without picking N
$f_2(N) = g_2(N) + h_2(N) \leqslant f^*(S)$
$\leqslant f^*(S) - g_2(N)$
$f_1(N) \geq f^*(S)$
$f^*(S) \leqslant g_1(N) + h_1(N)$
$\leqslant g_2(N) + h_1(N)$
$f^*(S) - g_2(N) \leqslant h_1(N)$
$g_2(N) \geq g_1(N)$
Revise g(m)
Conditions for optimal
① Finite branc
② for each edge
③ h(n)

It could always so the point of this whole exercise is that we can replace this by g 2 of n, we can rewrite this as we can replace it by g 2 of n plus h 1 of n, which means we can write this as f star of s minus g 2 of n less than equal to h 1 of n. So, I just wrote g 2 on to the left hand side and rewritten the statement here. So, I have this f star of s minus g 2 of n is less than equal to h 1 of n, n is that node, which even did not pick and that is why we could write this in equality.

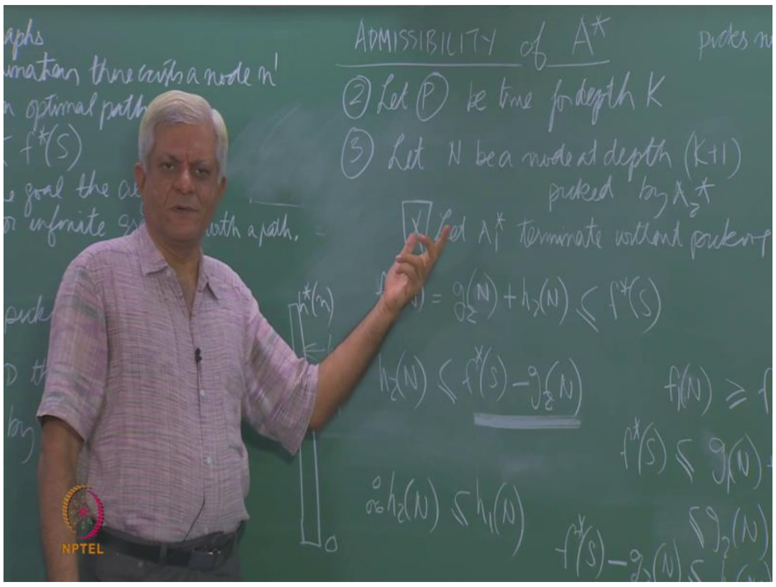
(Refer Slide Time: 49:05)



(3) Let N be a node at depth (K+1) picked by A_2^*
Let A_1^* terminate without picking N
$f_2(N) = g_2(N) + h_2(N) \leq f^*(S)$
$h_2(N) \leq f^*(S) - g_2(N)$
$f_1(N) \geq f^*(S)$
$f^*(S) \leq g_1(N) + h_1(N)$
$\leq g_2(N) + h_1(N)$
$\therefore h_2(N) \leq h_1(N)$
$f^*(S) - g_2(N) \leq h_1(N)$
NPTEL

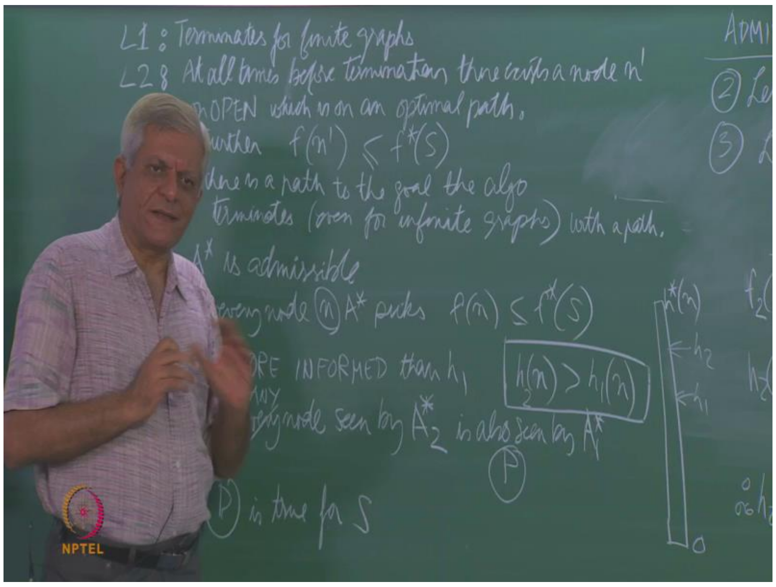
Here we have said the same thing that this is less than this and therefore simply by a transitivity here h 2 of n is less than equal to h 1 of n which is the contradiction, because we assumed this that is h 2 of n is greater is always greater than h 1 of n and here we have said that if this and this is only assumption we have made that let a 1 star terminate without picking n and that was select to this statement with contradicts over original statement essentially. So, this cannot be the case.

(Refer Slide Time: 49:35)


ADMISSIBILITY of A*
② Let P be true for depth K
③ Let N be a node at depth (K+1) picked by A_2^*
$f_2(N) = g_2(N) + h_2(N) \leqslant f^*(S)$
$h_2(N) \leqslant f^*(S) - g_2(N)$
$\therefore h_2(N) \leqslant h_1(N)$
NPTEL

And therefore, it cannot terminate without picking n, which means that if a 2 has pick this node n, A 1 also must pick this node n and which is the statement we are making that every node seen by A 2 will also be seen by A 1.

(Refer Slide Time: 50:00)


L1 : Terminates for finite graphs
L2 : At all times before termination there exists a node n' on OPEN which is on an optimal path
further $f(n') \leqslant f^*(S)$
there is a path to the goal the algo terminates (even for infinite graphs) with a path.
A* is admissible
every node ⓝ A* picks $f(n) \leq f^*(S)$
MORE INFORMED than h_1
$h_2(n) > h_1(n)$
every node seen by A_2^* is also seen by A_1^*
NPTEL

So, the effect of this is that the space explored by a 2 is contained not necessarily strictly contained, but contained in the space explored by A 1 essentially. So, it always space to have a better heuristic function; the higher the h 1 h 2, the better (( )). Any questions here? So, there is one more property which needs to be looked at, but we will do that in next class. So, I will stop here, and will sort of come back to this A star and it is behavior will do a quick recap in the next class, and then will start again. So, I will stop here.