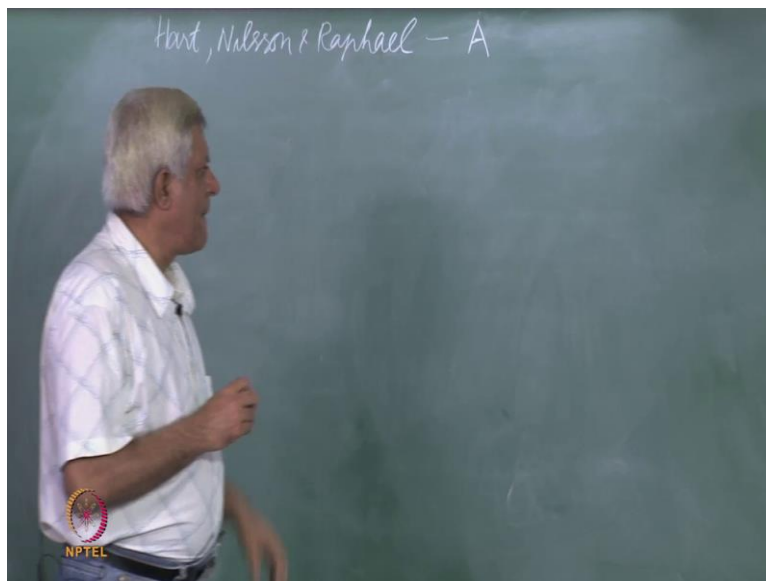


Artificial Intelligence
Prof. Deepak Khemani
Department of Computer Science
Indian Institute of Technology, Madras

Lecture - 19
A * Algorithm

So, today we want to look at this well known A star algorithm and this was devised by Hart, Nelson and Raphael essentially.

(Refer Slide Time: 00:27)



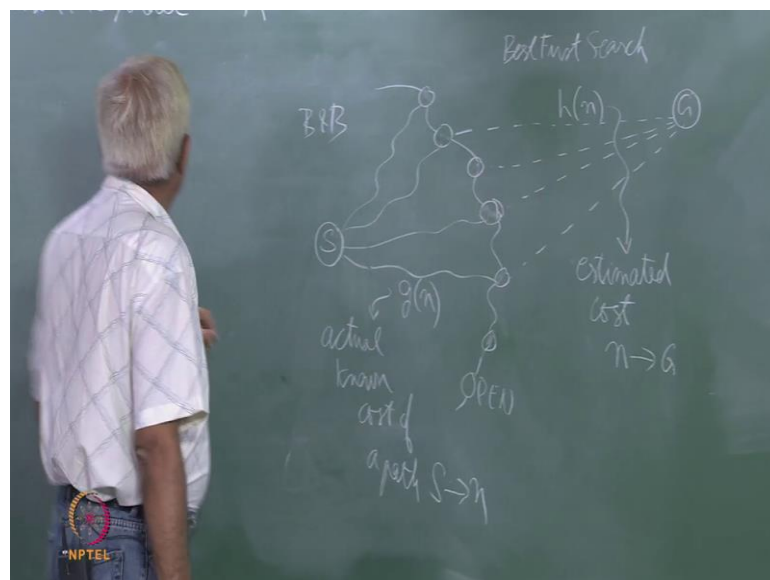
So, out of these Nelson is, we have met these characters very briefly in the introduction. Nelson is one of the founding fathers of a i along with Simon and Noel and Minsky and McCarthy. In fact, the first text book on a i written around nineteen seventy two also was written by Nelson essentially. Problem solving methods in a i and it discusses most of the search algorithm that we have seen accept the randomize algorithm and things like that. The best first search and heuristic functions and effective branching factor and all this kind of stuff was written in that book essentially.

Nelson was teaching at Stanford, Hart some of you may know because Doodah and Hart, they wrote a very well known book on pattern recognition. And Raphael, we had mentioned very briefly in passing, he was one of the first people to write a program to do automated theorem proving essentially. Anyway, today we are interested of course, in

this algorithm A star, but further moment we will just first call it A and then we will see in what condition we can call it A star.

Generally, when we look at some of these algorithms, that star implies that, it is admissible essentially, and by admissible, we mean that it is guaranteed to find optimal solution. So, branch and bound in that sense is admissible essentially. So, before I move on our librarian informs me that, my book which I do not know how many of you have seen is now available in the department library. There are about 5 copies and we are in chapter 5 of this books at this moment, so you should catch up with that essentially.

(Refer Slide Time: 02:44)



So, let us first get the motivation into place which we have just discussed is that you are at some start node S, and you want to go to some goal node g, assuming this is a city map or something like that. And you have some kind of a open list, and there are nodes on this open list which you have to pick a node from for expansion.

This is a old style algorithm that we wrote depth first search, breath first search, heuristic best first search and so on. We dint use this terminology in branch and bound, but you can see that branch and bound also always picks the lowest nodes in lowest leaf in the tree that we were drawing which is like you know open list that you can maintain.

Now, we saw that what best first search use was a function which kind of estimated the distance of every node to the goal node. And we call that function h of n and best first

search basically maintain a priority queue on the heuristic value and always pick the lower which appears to be closest to the goal and in the hope of finding the solution faster essentially.

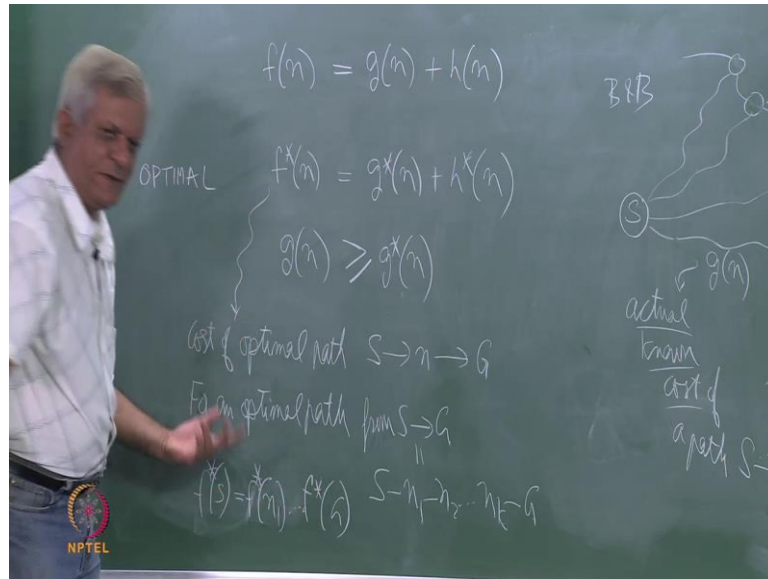
Branch and bound on the other hand keeps track of the parcel cost found so far, which are these cost. So, these are actual cost in the sense these are cost of actual partial solutions found. They may not necessarily be optimal though for diastal algorithm one can argue and prove that every time it picks a node it has found an optimal cost for that, but we will come back to that argument a little bit later. So, this is a actual cost that we have found to a given node so far essentially.

So, if you try to visualize this as some sort of a physical system, you can see that branch and bound like breath first search which was a simpler case of branch and bound tries to stick as close to the source as possible always picks a node which is as which has a lowest known cost essentially. So, it tries to stick as close to the source as possible. So, it is like a pull on this thing which is pulling it back essentially. Best first search on the other hand always tries to go as close to the goal as possible essentially.

Because, it is using the heuristic function which is an estimate of given note to the goal essentially. So, in some science that is a pull in that direction essentially. What we want to do in this algorithm A star is to use a combination of these two pulls and that is as it is as simple as that we just need to fill in the details essentially.

So, in A star terminology this value is called g of n and it is a actual cost of actual known cost of a path which is from S to n . So, this actual cost for this path is to n the kind of path that we were drawing in branch and bound we will call g of n the cost that we were using in branch. And bound we will that label we will call g of n , h of n is an estimated cost of the segment from n to g , that we have used in best first search that the h of n tells you how far this goal is from this from the goal node essentially.

(Refer Slide Time: 07:02)



So, in this algorithm A, we use a function called f of n which is basically the sum of g of n plus h of n . And essentially we keep the priority queue sorted on this value of f of n essentially. Now, notice that what is happening in this search algorithm we are starting with the source node of the start state and we are gradually making this implicit graph explicit essentially by a kind of a search tree that we are generating and whatever we have made explicit is divided into two parts. One is called the closed set or closed list which is the internal nodes and the other is the leaves which we call as a open set.

As we build this explicit graph this g value are computed, as we find partial paths we can compute their values and that is the g value. H value is independent of what we are doing, H value is simply a property of a given node. So, it is like saying how far Mailapur from Chennai central. We are just asking that question we has no question of paths here or some measure we said ((Refer Time: 08:12)) distance we can use or Manhattan distance we can use.

A heuristic function is just a property of the node. So, the algorithm that we will dip implement this A star every time we generate a node you can simply compute its h value which is this heuristic value once for all and you are done essentially. Whereas, this g value may change, why because you as we saw in the example first you may find the cost of length 15 to a node and then you may find another paths of cost 14 to that node. So, g of that node may change from 15 to 14.

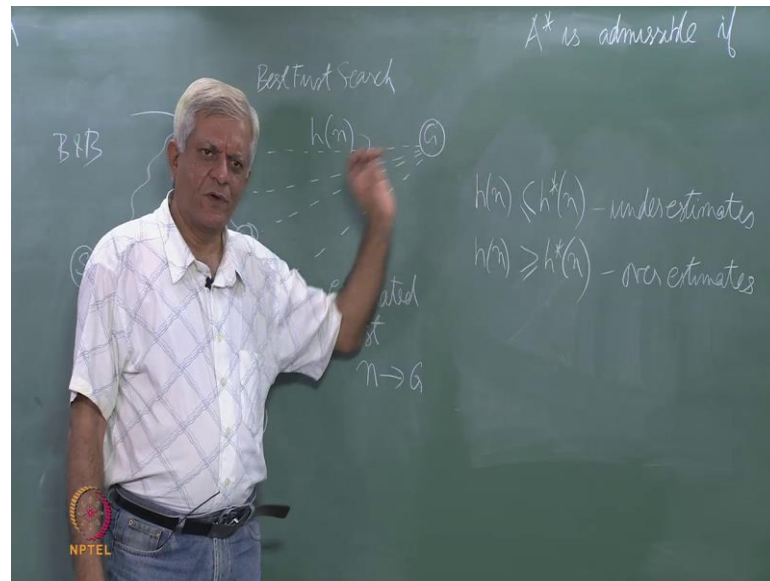
So, it is a quantity which keeps changing. So, it is a actual known cost or actual known cost of a known path I should say because, there is some path that you have not yet explored and which could be cheaper. Now, we use this term $f^*(n)$ is equal to $g^*(n)$ plus $h^*(n)$ and these are optimal cost. So, whenever we use the term star, they just denotes the optimal cost, we may or may not know them essentially. In fact, we do not know them most of the time essentially.

Even if we have found the path from some node to some intermediate node, we do not know what the actual cost, what is a relation between g of n and g^* of n , is it equal lesser than or greater than? Greater than how many people, feels it greater than only one. You said greater than right? Greater than equal to, why greater than equal to? So, g of n is a path that you have found to that node n so far, it may be not the best path. So, it could be cost greater than g^* of n .

So, this f of n is basically or S^* of n is the optimal the cost of optimal path that starts with the start node goes to node n and then goes to the node g . So, for any node n , f^* of n is optimal cost optimal path cost which passes through the node n essentially. And if that path is an optimal path, so if, so there is a distinction between these two. Here we are saying that for any node n what is the optimal cost going through that node n . Here we are talking about an optimal path from S to G . We can assume that, if this is equal to $S \rightarrow n \rightarrow k \rightarrow G$, then we can write f^* of S is equal to f^* of n .

And, so on f^* of g because, it is a same path remember. When we say this is optimal path then whether we say f^* of n any node on that path or f^* of start or f^* of goal they have the same cost because this is only one path that we are talking about. So, for an optimal path these two this sources this we will use as some later point.

(Refer Slide Time: 12:37)



A star is admissible if now, we come to the same question again the relation between h of n and h^* of n what should be the relation. So, I am giving you two options here include some we do not have equality. So, we have included an equality in both the sides which one is better. A first option or the second option? What should my heuristic function do? Should it what is this one doing. This is this underestimates, and this overestimates. And I making a claim and this claim we will show more formally probably in the next class. That if I choose a right condition here, then the algorithm A star becomes admissible and by admissible we mean it will find you the optimal cost path essentially.

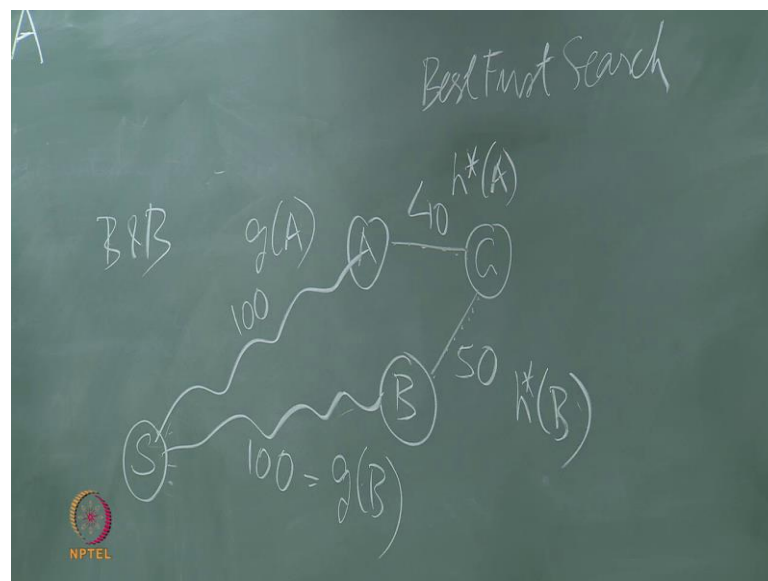
So, should my h of n be an underestimating function which means h of n should be less than h^* of n . Remember h^* of n is the optimal cost of going from n to the goal or should it overestimate the cost of going from n to the goal, which one will make my algorithm admissible. Should I take a vote on this or somebody going to volunteer in answer?

So, let us take a vote, how many people feel it should overestimate the cost? Only one, two can I use negation by failure, no. How many people feel it underestimates a cost? A very small number essentially. So, let me we will come back to this. So, let me, let me take an example to illustrate this whole point essentially, and then you will see or let it let me put it this way.

It is an analogy that, I often tend to use that supposing you are buying a new, you want to buy a new, mobile phone the most popular object now a days and you want to get a, you have decided what phone you want to buy and but there are three or four shops which are selling it. So, you go to the first shop and let say I do not know what is the good price for you people. So, let us say it is 10000 rupees, this fellow says I am giving you this phone for 10000 rupees.

Now, there you have some estimate of how much the other people will charge, will they will you go to them if you think they charge less or will you go to them if you think they charge more than 10000. Charge less essentially which means that you think that they are that value of that node is underestimating the cost essentially. So, let us then look at this example.

(Refer Slide Time: 16:57)



Let us say I have only two nodes in open let us call them A and let us call them B and let us say this is the actual path found to them and let us say that they are just one step away from the goal this just to illustrate the idea. So, that is the edge that they are just likely to explore let us say that this cost is 100 just to simplify a matters let say this cost is also 100 and let us say that this cost is 40 and this cost is 50 that is the edge cost A G is the cost of edge A G is 40 and the cost of edge B G is 50.

So, what do we have, we have this 100 is equal to g of B this is also equal to g of A and that is a actual cost we do not that is you might say the h star of A is 40 and h star of B is

50 in this case of course, if there is only one edge it is basically a falls on to the edge cost essentially where it could be more than one I think. So, let us look at examples of both this functions.

(Refer Slide Time: 18:26)

$h(n) \leq h^*(n)$ - underestimates h_1
 $h(n) \geq h^*(n)$ - overestimates h_2
 $h_2(B) = 70$ $f_2(B) = 100 + 70 = 170$
 $h_2(A) = 80$ $f_2(A) = 100 + 80 = 180$
 $f_2(G) = 150$ $g_2(G) = g_2(B) + 50$
 $$ $$
 $$ $ = 150$
 $$ $h_2(G) = 0$

So, let us say h_1 underestimates the cost and h_2 overestimates the cost. And let us pick functions which are not very good in the sense that both of them are giving you wrong information, both of them are telling you is that B is actually closer to the goal than A is closer to the goal, so let us work with this.

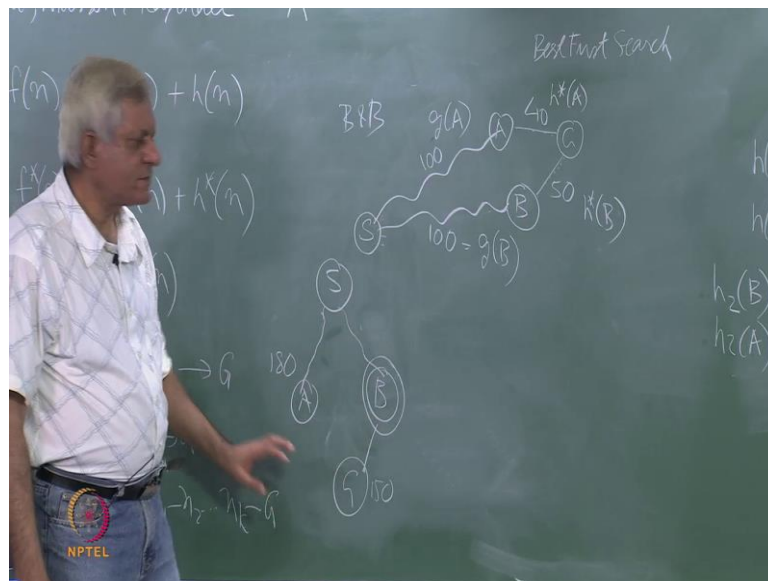
So, h_2 let us say h_2 of B equal to it is actually 50. So, let us say this is 70 and h_2 of A is actually 40, but let us say it is 80. So, what does it mean that this function h_2 which is a heuristic function. So, remember we did this example with this blocks world where we tried to define its functions when we saw that one of them was better than the other. This is looking at another aspect of heuristic functions which is one of them is underestimating and the other one is overestimating.

So, first we are looking at a function which over overestimates the costs which means that this actual cost for this is 50, but this function thing it is 70, actual cost for this is 40, but this function thing it is 80. So, in actual practice A is closer to the goal which means this part should have been better, but this function will think that B is closer to the goal because of the estimate it has h_2 of B is 70 which is less than 80.

So, f_2 of B will be g of A which is g of B which is the same is equal to 100 plus 70 equal to 170 and f_2 of A is equal to 100 plus 80 equal to 180. So, remember that we said that basically what this algorithm A star does is it maintains a priority queue of on f value essentially. So, in this small example there are only two nodes in the open list, one is A one is B and it has to decide which one to pick it uses is f values f_2 is 180 f_2 of A is 180 and f_2 of B is 170.

So, it will pick this node which means it will now what disaster would have said relax the edge here which means it will compute g of g . So, g_2 of g is equal to g_2 of B plus the actual cost which is sorry which one have we picked we have pick B which is 50, correct? The actual cost was or let me write which is 50 which is the actual edge cost which is equal to 100 plus 50 is 150 and h_2 of g is equal to 0. We assume that our heuristic function can at least tell us if we have reached the goal. So, the heuristic value at the goal is 0 which means f_2 of g is equal to 0 plus 150 is 150. So, what is happened in our search space?

(Refer Slide Time: 22:17)

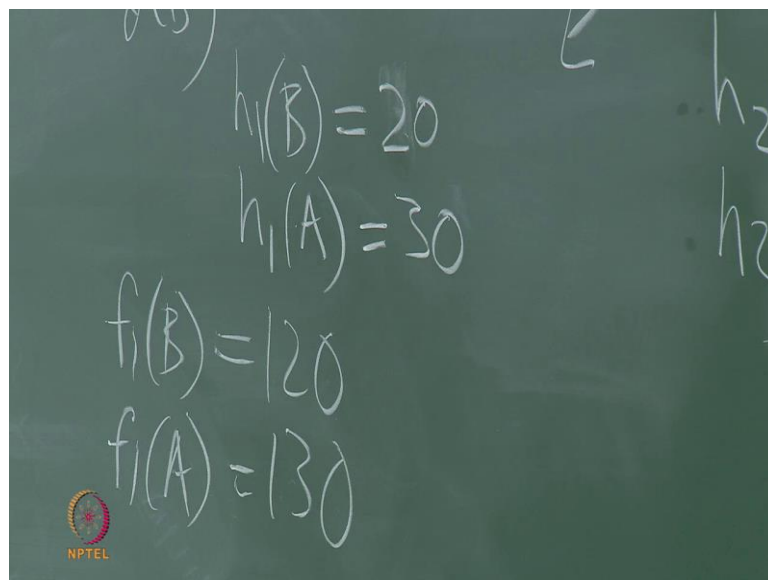


We started with S, then we had this two paths, this is A, this is B, then our algorithm h_2 has pick B and found a path to g , then we have this two node on open A and G. A has estimated cost now, estimated cost we mean the complete cost of going to the goal via this node A is estimated to be 180 because, the heuristic estimate from A is 80 and the

cost up to A is 100. So, 100 plus 80 is estimated to be 180. The estimated cost of g is the cost found, so far which is the g value which is 150.

So, here we have this node sitting with 150 and this node sitting with 180. So, the algorithm will pick g and terminate, the same algorithm that we talked about in the last class when the completely refined solution is becomes a cheapest. It will automatically get picked in this priority queue and then the algorithm will terminate. So, it will, it will find this path essentially.

(Refer Slide Time: 23:33)



The image shows a chalkboard with handwritten mathematical expressions. The expressions are arranged vertically and are as follows:

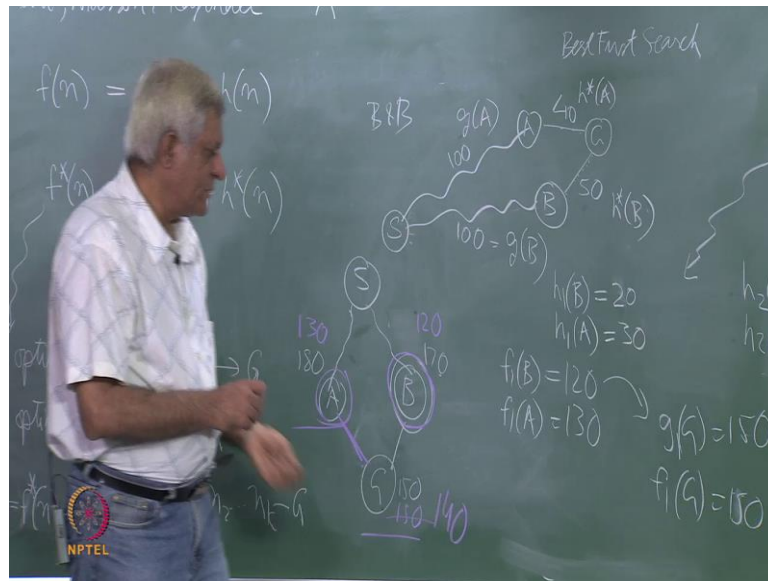
$$h_1(B) = 20$$
$$h_1(A) = 30$$
$$f_1(B) = 120$$
$$f_1(A) = 130$$

In the bottom left corner of the chalkboard, there is a small circular logo with the text "NPTEL" underneath it.

Let us try the other option which is h_1 of A. It underestimate, so let us say h_1 of B equal to it is actually 50, but now we are underestimating. So, let us say it is 30 and h_1 of A is actually 40.

So, let me say this is 20 it is actually 40, but let us say this function thing it is 30. So, in the manner h_2 and h_1 are similar the both of them think that B is closer to the goal, the only difference is h_2 overestimates the cost and h_1 underestimates the cost essentially. So, in which case as you can see f_1 of B equal to 20 plus 100 120 and f_1 of A is equal to 130.

(Refer Slide Time: 24:43)



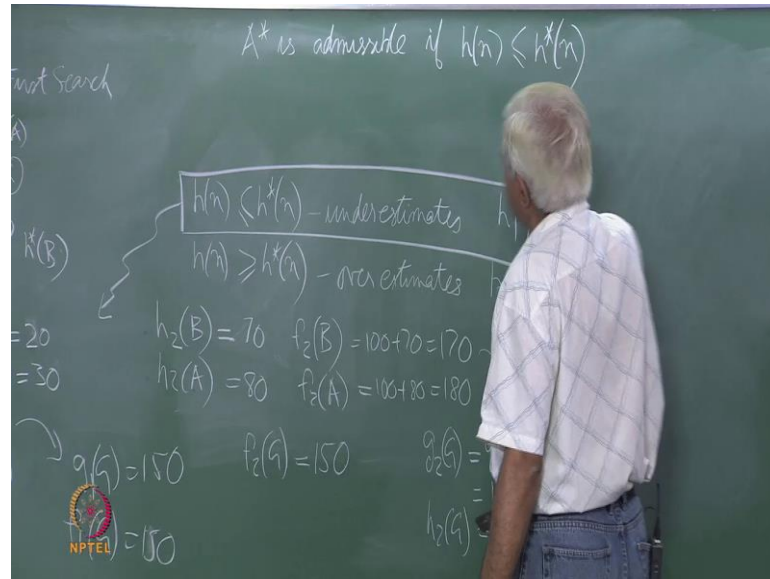
Again this is a very similar situation, this algorithm, so originally this was 180, this was 170 and this became 150 with this new algorithm f_1 this is 130 and this is 120. So, again this algorithm will explore B first, but what will happen now after picking B it will relax this edge from B to G which actual cost is 50. So, $g_1(B)$ is equal to 150 which means $f_1(G)$ is equal to 150.

See after they have found the path going from S to B to G both of them know the actual cost there is no longer this estimation done by h because, h is 0. So, the actual cost, so $f_1(G)$ is 150 as well as $f_2(G)$ was 150, but now you can see that this is with the same cost 150 and this algorithm let us call it A_1 star has to choose between G and A. It turns out that A is lower estimated cost 130. So, this algorithm will pick A, and when it picks A it will find a cheaper cost to G exactly like diastal algorithm would have done, and this cheaper cost is the cost of 100 plus 40 which is 140.

So, this will revise this cost to 140. Originally the estimated cost was 170 and it got revised to 150 here, it was 180 it never got revised to 150. Here it was 120 it got revised to 150, but when this expanded this it got again revised to 140 essentially. And at that time this is the only goal only door left in our open list because we have finished with A, we have finished with B both and only G is left and G will be picked with the cost of 140 which is the optimal solution.

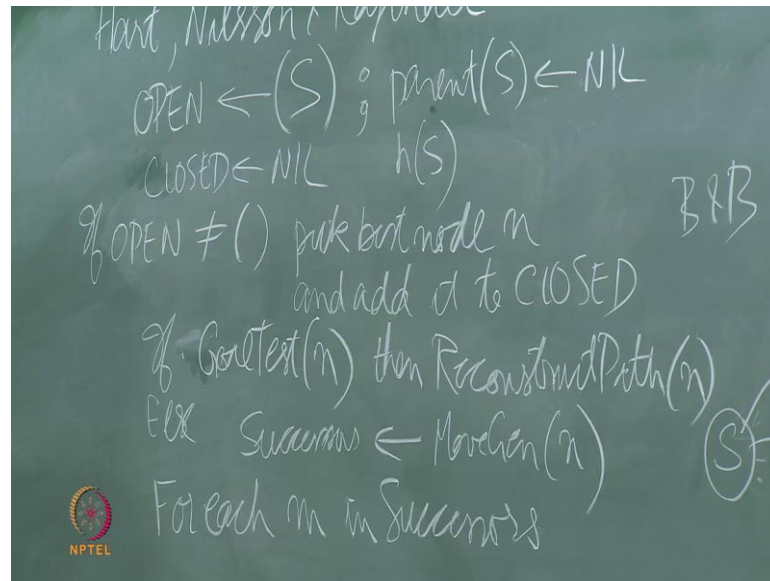
So, you can see that using h_1 of h_1 we could find the optimal solution using h_2 we could not find the optimal solution, even though both of them thought that B was the better choice using h_2 we picked B and then we picked the goal using h_1 we picked B, but then we were forced to pick A and then we were forced to pick the goal essentially. So, we found the optimal cost.

(Refer Slide Time: 27:31)



So, this is the criteria we are looking for. If h of n is less than or equal to h^* of n . So, the algorithm is admissible if the heuristic function underestimates the cost of the goal essentially. So, let us quickly describe the algorithm completely because there are one or two things which were not there in the best first search algorithm. So, I will do a quick description and then we will do it we will take it up again in the next call essentially. So, the algorithm is very similar, except that we keep track of this cost explicitly g of a node n , h of a node n , so on.

(Refer Slide Time: 28:44)

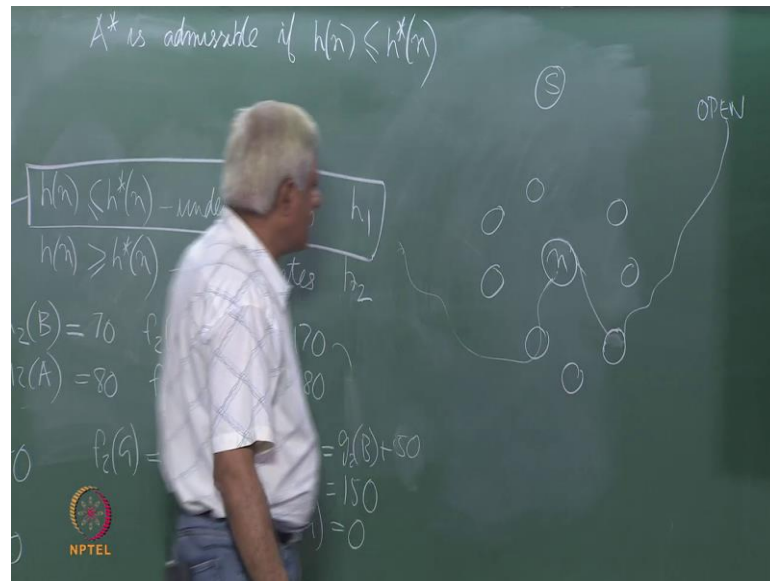


So, the steps are very similar at. So, I will just write the outline here and you can fill in the details. So, I will just very briefly write open gets S which means open gets the start node, but we want to keep track of parents and so on and so for.

So, let us just write here parent of S is nil and h of S is computed we always we can always compute the heuristic value of a node. So, in as a matter of principle we compute the heuristic value and we initialize close to nil as before accept that here we have talked of this parent point of, rather than having this node pair that we had earlier you can use that it does not really matter, this is simply easier to describe. So, as before if open is not equal to nil, pick best node n and add it to closed. I will just write it as an outline I want to focus on the part of what do you do with this node n essentially

So, if n, so if I use the older terminology if gore test n than the reconstruct n. So, we assume that we can trace back the path and reconstruct the path and then. So, all that it is very similar to what we did in earlier search algorithm, else successors is the move gen of n. So, we I am using the same function that we have defined earlier and for each m in successors we do the following essentially. So, let me also draw the situation at the point where this algorithm is doing working.

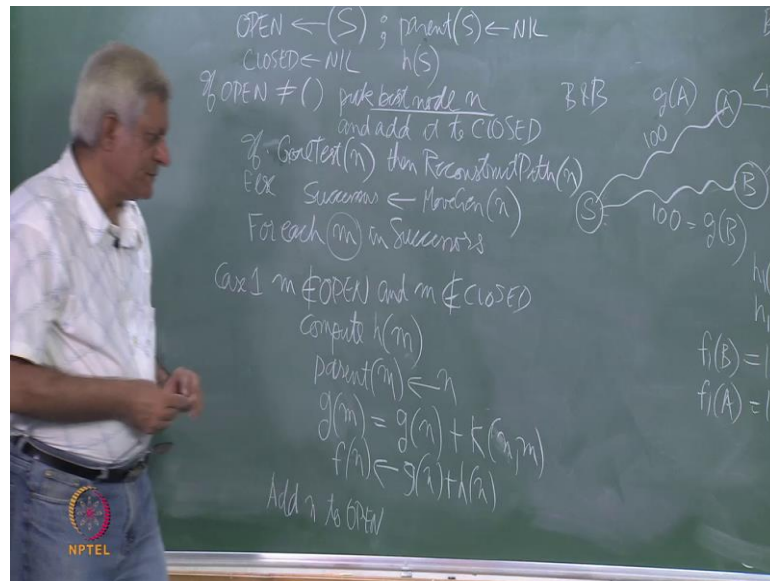
(Refer Slide Time: 31:26)



So, this is S and then somewhere we have this open list, this is the set of open node and let us say this node n is here essentially. So, this line that I have drawn represents open list and we picked some node, nodes with the lowest f value have I said that here, ok I have said best node n and by best I mean the lowest f value and f is g plus h and we it was not the goal node. So, we generate successors.

So, let us say these are the successors of n. All neighbors if you want to say or what are reachable from there. Now you can see that there are three kinds of nodes here one is one kind which is on the open list already. So, these two I mean to draw is that they are already on open there are some which are already in closed and there is some which are new nodes essentially. So, this one represents that new nodes essentially.

(Refer Slide Time: 32:43)

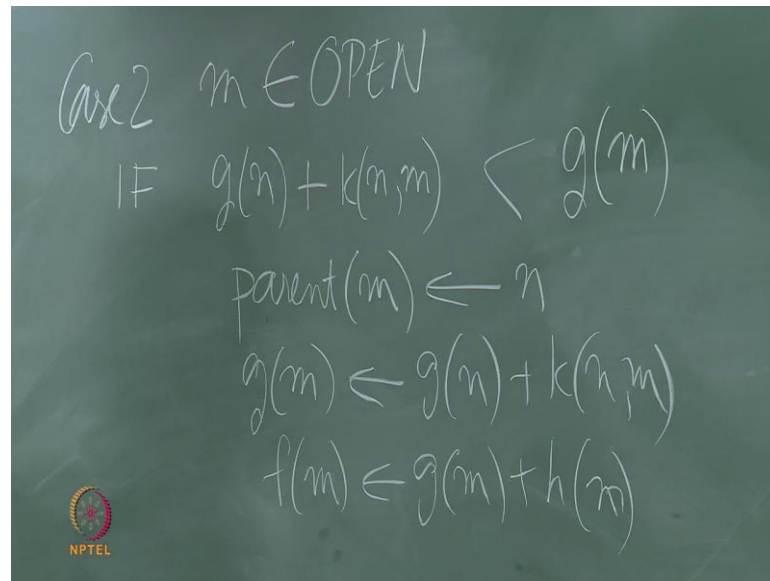


So, for each M, so case one m does not belongs to open and m does not belong to close which means it is this node here that I am talking about, it is a new node that has not been seen earlier essentially. Basically we want to add it to our search space. So, what do we do we say add compute h of m then.

So, we have some heuristic function that we use. Then we say parent m is n and g of m is equal g of n plus I will use k as a cost function. So, that k of n m is the cost of that edge going from n to m. So, we update the value of or the compute value of g m is equal to g of n. The g value of the parent exactly as we were doing in this branch and bound kind of a thing and then we compute f of n and we say add n to open.

So, there are three cases the first case is when m is a new nod. So, we compute its edge value we compute its g value which means we can compute its f value, we mark the pointer of m as its parent that it came from and add it to open essentially. The next case that we want to look at is when it is already on open.

(Refer Slide Time: 35:14)



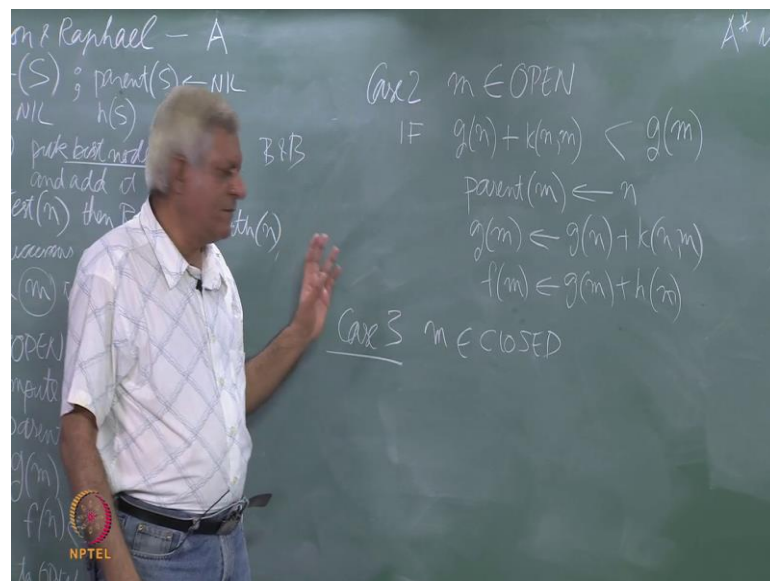
So, case 2, m belongs to open which means what, that we have found this edge here. So, we are dealt with this these two cases. So, now, if this one of these two was m , it means it is on already in open which means it is already has a g value and h value and a parent point. So, this n has just been expanded. So, before that, this was pointing to let us say this one for argument sake, let us say this was pointing to this and this was pointing to this. So, they already had this parent pointer, what do we do for such nodes, or what do we want to do?

We want to see if we have found the cheaper path to that node or not, correct? We already have a path to this node m it is coming through some sequence of nodes n always remember is a parent point. So, it already has a g value and an h value as well. So, if m belongs to open, then we do a check if g of n plus this value A of n m . What is this value?

This value is a new cost that we have found to this node m , this node m was already on open. If this is less than the cost that was already stored for m , remember it is all open. So, it is must be having a g value, but if this new cost that we have found is better than the old cost, then we have to do some readjusting of pointer which is that parent of m becomes n and g of m becomes this value. And f of course, becomes in all these h of m is not changing at all because, it is just a property of that node, it is not the property of that what path we have found to this node.

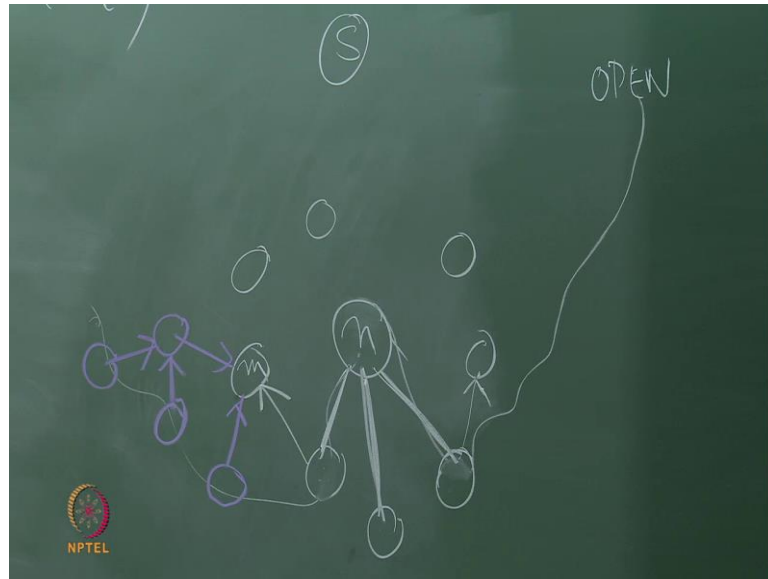
So, these are the two cases. One case for new nodes, we simply add them to the graph, and create the parent pointer, compute the g value, compute the h value, compute the f value and we are done. If we have found a new path to a node in open then we need to check whether this path is better which is what we are doing here, if it is better than we readjust the parent pointer and readjust the g value essentially. So, this is where we are readjusting the g value, this is where we are readjusting the parent pointer and then this of course, taking into account the new g value.

(Refer Slide Time: 38:27)



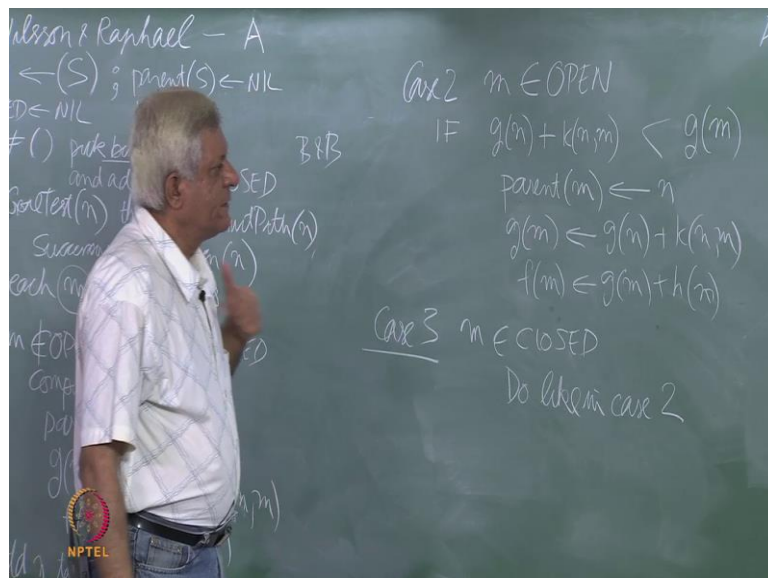
Case 3, m belongs to closed. Which means we had already visited m and expanded m and generated children of m n, so on and so for, so let us take this as a case.

(Refer Slide Time: 38:47)



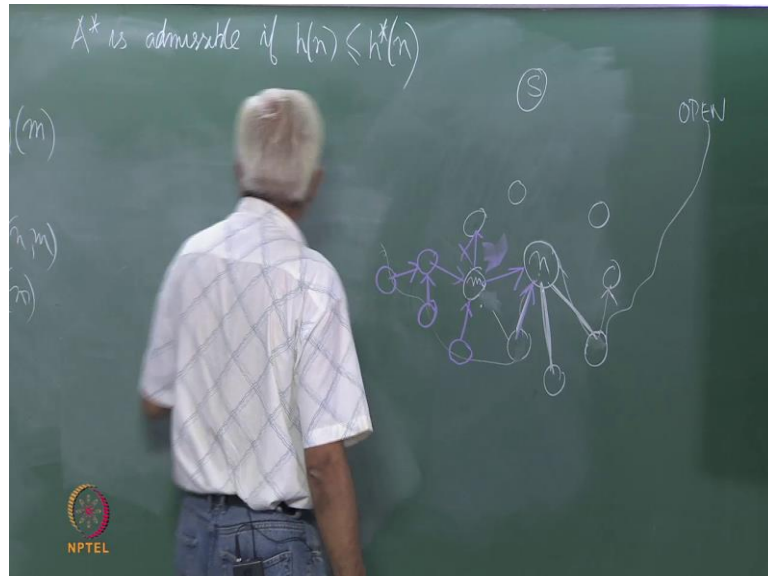
So, let us say now this is m , which means it was already generated before which means it could have other children. So, let me use a different color here. So, this node could have been a child of m and maybe this node could have been a child of m which could have its own children child here anything is possible essentially. The difference between the node on open and a node on close is that a node on close may have children of its own essentially. So, these purple nodes represent children of m essentially. So, if m is on close first of course, we have to do what we did for case 2.

(Refer Slide Time: 39:41)



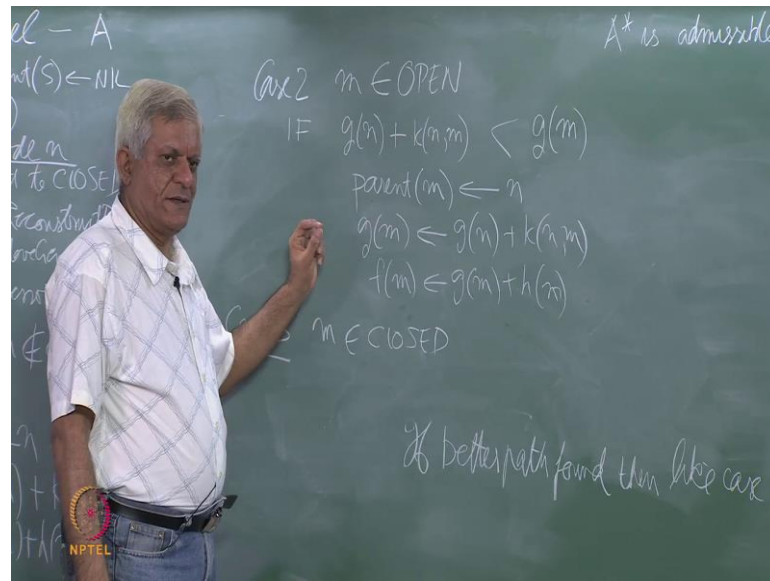
So, first I will say do like in case 2, which means that this node m we have found a better path to this node m.

(Refer Slide Time: 39:56)



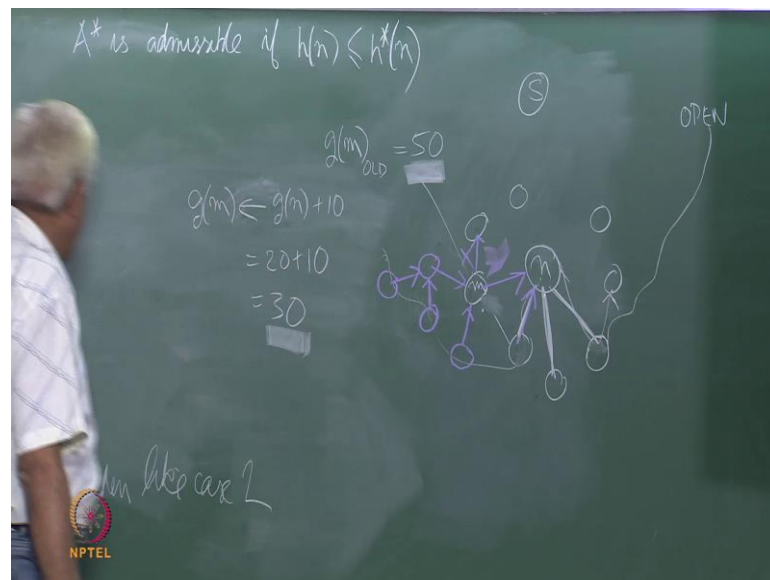
Originally this m had some other pointer let us say this pointer. So, remember this are pointed to parent nodes. So, m was pointing to its parent here and if we find a better path to m, we must shift this pointer here and say that this is the new parent, exactly like what we did for n. Originally this node first pointing to this, but now we have removed this and we have said this is a new pointer if we have found a better path likewise for close nodes essentially. So, do like in case 2.

(Refer Slide Time: 40:39)



So, if let me use this if better path found which basically means this condition g of n plus k of n m is better is less than g of the old value of m then. So, actually I should do like this then like case 2 which means readjust the pointer, parent pointer readjust g value, but now the g value has changed essentially.

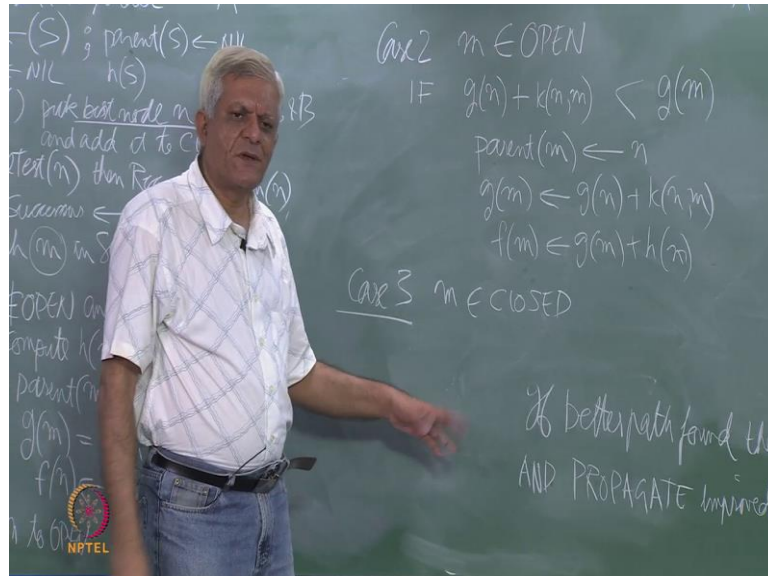
(Refer Slide Time: 41:32)



So, we need to, so just to take an example, if the old g of m let me say old is equal to 50. And if g of m gets a new value which is let us say g of n plus 10, and let us say g of n is 20 equal to 20 plus 10 equal to 30.

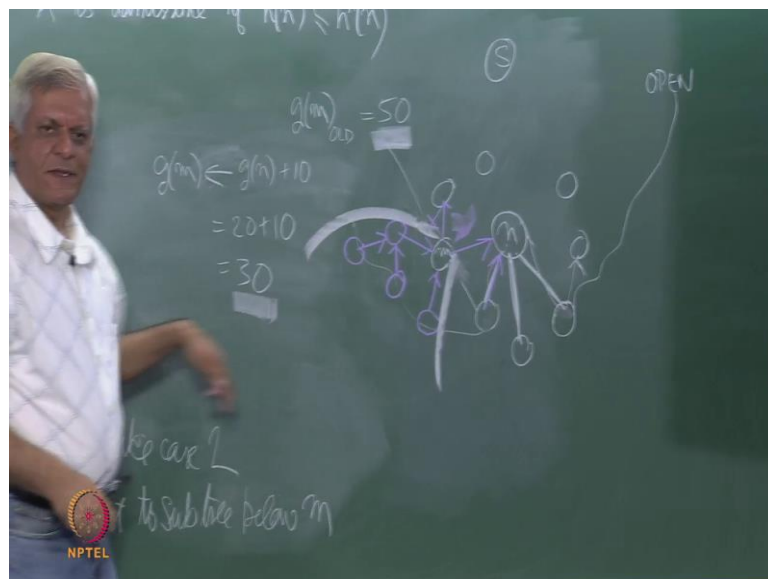
So, if the old value was 50 and the new value was 30, then we have found a new path to this m which is 20 unit cheaper than the old path which means that all path going from m to other nodes we must pass on this improvement to them essentially. So, you must reduce the cost of this by 20 reduce the cost of this by 20 and then reduce the cost of this by 20 and so on and so for.

(Refer Slide Time: 42:34)



So, I will just write this and propagate improve cost to sub tree below m . How can we do this propagation? You can just do a depth first traversal or something like...

(Refer Slide Time: 43:13)



that of this purple tree which is rooted at this m , which means this part of the tree to this node to this node to this node that improve cost must be preoperative to them.

So, that is a third case, if you have found a new path to a node in the closed. A better path to a node in the closed, then that improvement has to be passed on to children of that node essentially, which was not the case in disaster. I mentioned, when we were talking about disaster algorithm, that if you have found the path to a node in disasters algorithm you have already found optimal path to there. But in A star that is not the case essentially, like we saw in the example that we saw, if you have found a path to g which was through the node B which was a more expensive path you can find a new path from that node essentially.

And it is also possible for nodes and close you can easily construct an example. So, we need this extra step that if we have found better path, so node that we have already expanded earlier we need to propagate this improvement to their children essentially. So, which is a little bit more work than case 2. Where? This one? And here, no that is all and add m to correct. So, under the conditions when heuristic function underestimates a cost to the goal this algorithm is called A star and it is admissible essentially.

In the next class, when we meet we will prove that it is admissible. So, this is one of the few cases where we will give a formal proof of admissibility because, A star is a very well known and often used algorithm, but this is not the only condition for admissibility I mean this was the condition only with respect to the heuristic function.

So, what we have shown with an example today, that if a heuristic function underestimates the cost of the goal then the algorithm is admissible. And you can see branch and bound is a special case of A star where h of n is always 0. That is why we said that you can think of this as an estimated cost, if you just assume h of n is 0 for every node then this algorithm becomes branch and bound essentially, but this is not the only condition.

So, what I want you to do is, to think just imagine that. So, A star is actually some people just call it graph search. It is a weighted graph and you have to find the least cost path from one node to another node. We make some simplifying assumptions, but it is different from disaster algorithm in one respect and disaster algorithm assumes that the

complete graph is available to you and it is a kind of an old in the sense that, it finds the shortest path to every other node from a single source essentially.

We are not interested in to every other nod, we are only interested in finding a path to a given goal node and we do not have the graph available to us, the graph is generated on the fly as we go along essentially for some were we have this function, move gen function. The graph is generated as we go along using the move gen function. So, we start with some node we keep adding more nodes to the graph and then we work with that under these conditions A star is a little bit different from disasters algorithm.

It also uses this idea of a heuristic function diastral algorithm did not need to do this because it was anyway going to find shorter path to all nodes in the graph essentially. It did not have a goal in mind, A star has a goal in mind. If you want to say it has a goal in mind and therefore, it benefits from the use of a heuristic function, and if the heuristic function is underestimating function then that is one of the conditions for admissibility.

So, I wanted to think of what other conditions you can might require for admissibility, just imagine that this is some obituary graphs such problem some graph is given to you and under what conditions will you, will this algorithm find an optimal path. And we will take this up in the next class we will do a formal proof of this essentially after we do that we will compare heuristic functions again formally and show that heuristic functions are which are more informed they do lesser search.

So, remember that when we were talking about branch and bound we say that this estimate must be as high as possible, it must be a lower bound even this is a lower bound, but it must be as high as possible, and we will formally show that the higher the estimate the lesser the number of nodes that this algorithm will search before termination essentially, we will do that these two formal things in the next class.

So, we will stop here.