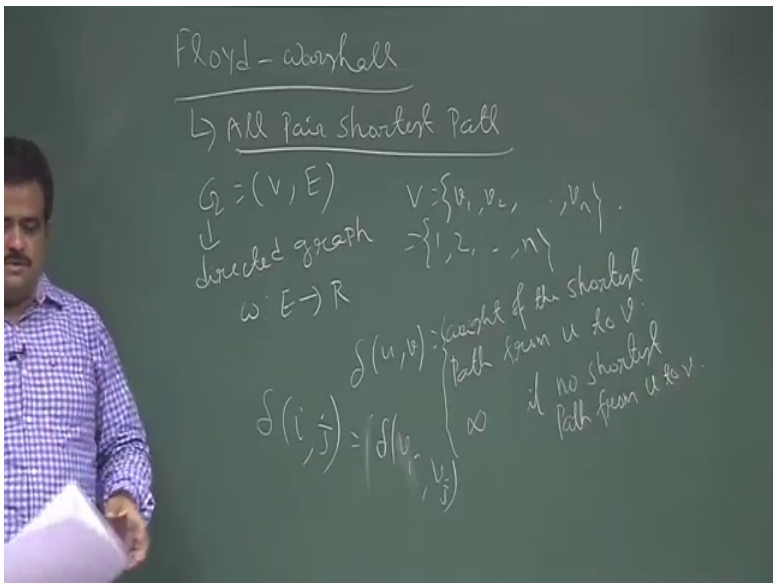
An Introduction to Algorithms
Prof. Sourav Mukhopadhyay
Department of Mathematics
Indian Institute of Technology, Kharagpur

Lecture – 50
Johnson Algorithm

So we are talking about all pair shortest path. We have seen two algorithm dynamic programming based approach like last class we have seen the Floyd Warshall Algorithm. So, today we will talk about some application of the Floyd Warshall as algorithm, that is to find the transitive closure of a graph and then we will moved to the another algorithm for finding the all pair shortest path its called Johnson Algorithm, but this will be using the old algorithm like we know Dijkstras Algorithm.

So, let us first talk about the applications of Floyd Warshall. So, just to recap, so Floyd Warshall algorithm.

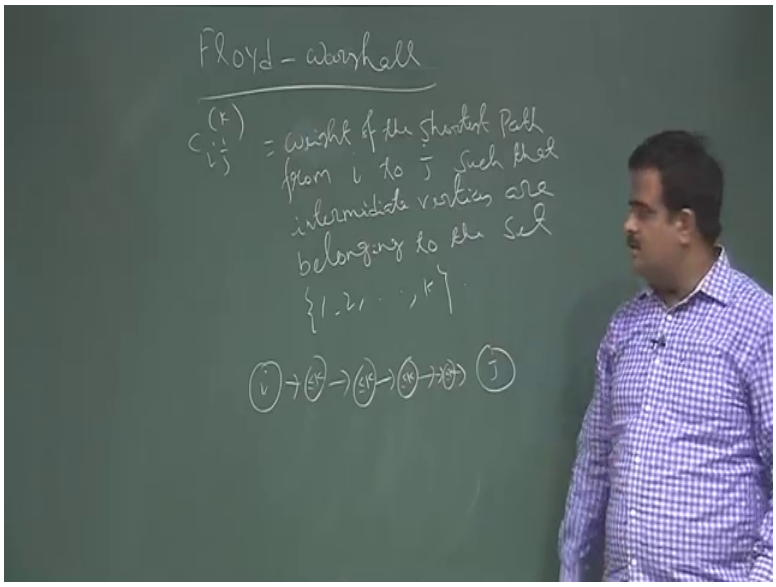
(Refer Slide Time: 01:02)



So, it is basically sorry r is missing a r s. So, this is basically to find the all pair shortest path. So now, we have seen suppose have a given a graph G, V comma E now. So, this is a directed graph or diagraph and we have a edge weight E to R. So, now you, so suppose V is basically V 1, V 2, V and we denote delta u, v is the weight of the shortest path from u to v, if it is exist otherwise it is infinity weight of the shortest path from u to v. If it exists otherwise it is infinity, if there is no shortest path form from u to v.

So, basically our aim is to find this deltas. So, delta of i j. So, these delta of i j is basically delta of u i sorry v i, v j. So, our vertex are these v i, v j. So, these we are denoting by delta of i, j. So, is to simplify you denote the vertex here from 1 to n. So, basically delta i,j is the weight of the shortest path from a i to v j. It could be infinity also if there is no shortest path. So, for Floyd Warshall what we did to find out this delta. So, we have defined. So, this we have discussed in the last class just to recap.

(Refer Slide Time: 03:57)



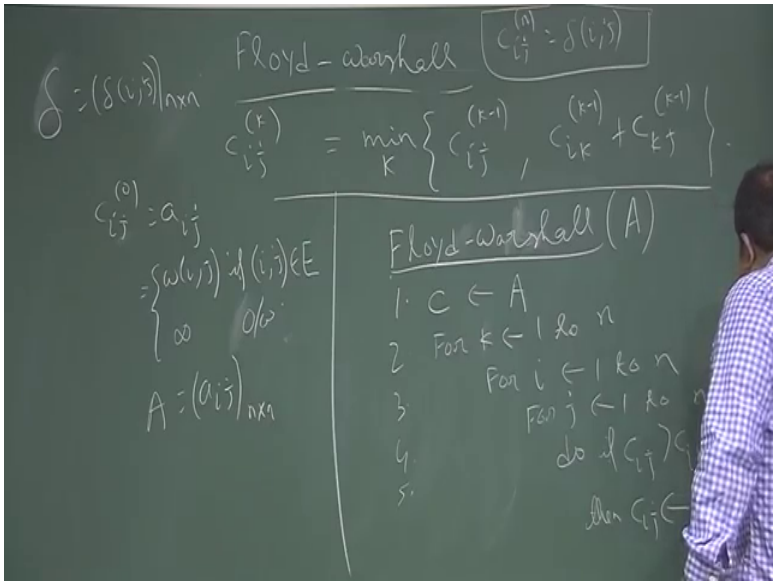
So, we have defined the c i j, the recursive relation c i j k is the minimum of this is the basically weight of the shortest path from path from i to j, v i to v j such that the level the intermediate note, intermediate vertices are belonging to the set 1, 2, k. So that means, we are at i'th node, we want to go to j'th node. So, we will see some vertices in the middle in the path like this dot, dot, dot, these.

So, these are the intermediate node in the path of i to j. So, we want the level of this nodes from 1 to k. So, we. So, this is a i j. So, all the vertex level must be less than k less than equal to k; there is no restriction on i and j, but in the middle the vertexes we have visiting they must be level formed 1 to k. I mean we should not see any vertex whose level is more than k. So, we should not see a vertex whose vertex v k plus 1 say or k plus two. So, that is how we defined a c i j k.

So, basically what we have we have c i j's; what is the recurrence for c i j? So, c i j j 0 is basically a i j, the adjacency matrix. Adjacency matrix means that adjacency; so a i j is w

i j if i j is on edge, it is w i j, if i j is on edge otherwise we have infinity yeah. So, otherwise, so what is a i j? i j is otherwise it is infinity.

(Refer Slide Time: 06:33)



So, basically c i j is what? c i j 0 is basically what? c i j 0 means we are at i'th level. Now in the middle we want a vertex whose size is less than equal to 0, but our vertex starts from 1. So, we dont want any vertex. So, if there is a direct edge from i to j and then that will be the basically w i j. So, it is basicallya a i j if it is, so it is basically the adjacency matrix coming from the adjacency matrix. So, this is the c i j 0 the initial condition, but what is the c i j k?

So, these in the last class we have proved that this is basically minimum among k such that c i j k minus 1 these and c i k k minus 1 plus c k j k minus 1. So, minimum among these. So, from these we have the algorithm divided and conquer approach. So, that is Floyd Warshall Algorithm. So, what is that? So, this is basically taking a graph with the matrix A. A is the adjacency matrix. So, this A is basically consist of a i j n cross n matrix, this is the adjacency matrix.

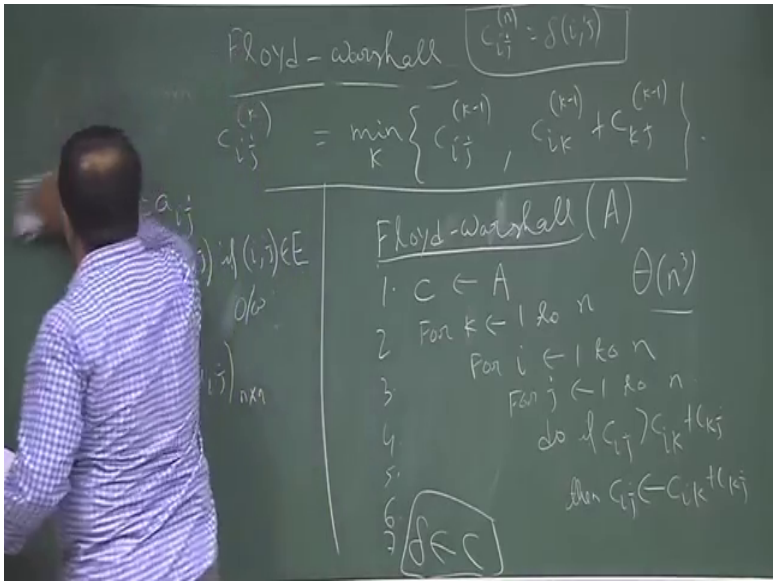
So, basically you initialize by c if you denote this c matrix. So, what is the advantage of getting c if we know c we we are looking for delta. So, delta is basically delta matrix is basically delta i j. So, we are looking for delta. So, what is the relationship between delta and c, now if we can find out all c i j k recursively kind of dynamic programming technique. So, basically c i j n because our vertices start from 1 to n. So, c i j n is

basically giving us delta i j. So, c i j n is basically giving us delta i j. So, that is why if we can some how can find out all the c i j's for all k from 1 to n then you got the deltas.

Anyway, so this is the initialization step. So, we want to find the C. So, this is the C matrix which is initialized by A matrix then, we have this formula. So, for i is equal to 1 to k sorry for k is equal to this is c i j k. So, basically you can writec c k initialize by these, but we are using the same variable. So, you are just doing 0 to n then for i is equal to sorry for i is equal to 1 to n and then again for j is equal to 1 to n. This is 3 for loop, what we are doing? We are testing the value of c i j, what we we have already. So, this is a initializatin c i is a 0 then, you are looking the c i is a 1. So, now, if these value if c i k is plus this 0, if this is less than this c i j 0 then we must have this in our c i j 1. So, that is the formula.

So, do if c i j if this is greater than c i k plus c k j then c i j is replace by c i k plus c k j. So, this is the c i j is replaced by c i k plus c k j. So, this is the minimum one, we are storing this. We can add up a k over here, but they that k is basically in a superscript, but we are getting this here.

(Refer Slide Time: 11:37)

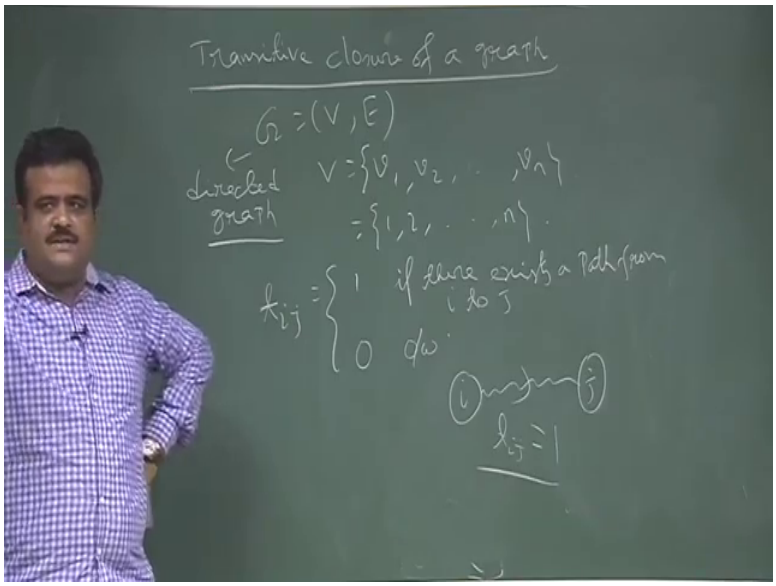


So, this is the formula to finding c i k's and finally, the C matrix will be our delta 1, 2, 3, 4, 5, 6, 7 now that is our delta. So, delta is basically the C matrix. So, this is what we are looking, we are done. So, this is what we are looking for. So, this is what we have this is called the Floyd Warshall Algorithm and the time complexity for this is basically you

have 3 for loops. So, time is order of n cube algorithm. So, now, you will talk about an application of this algorithm which is called transitive closer of a graph.

So, suppose you have a graph it is a directive graph and here there is no weight on the edges.

(Refer Slide Time: 12:34)

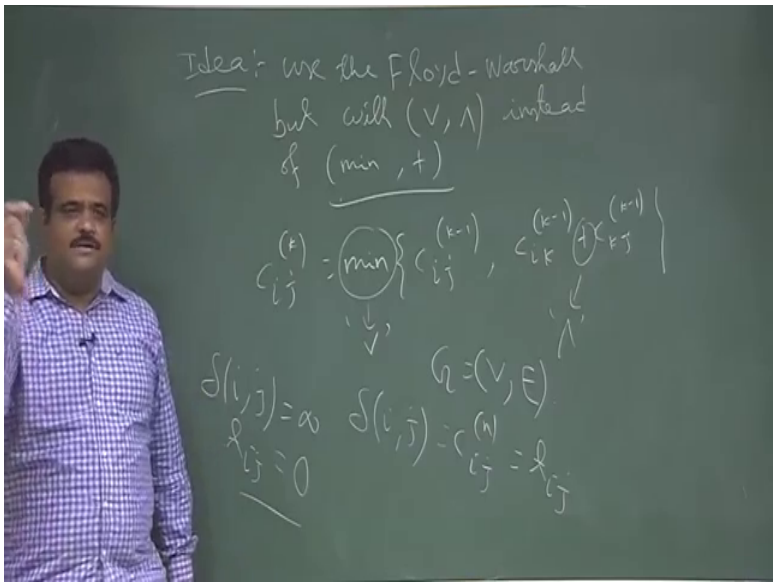


So, this is called transitive closure of a graph. So, suppose we have a graph G. So, say if V say v 1, V 2 there was say n vertices and for simplicity you are writing vertex level inverse 1 to n, so vertex of form 1 to n. Now so we define t i j to be 1 if there is path for i'th vertex j'th vertex. If there is a path 1, if there exists a path from i to j, otherwise it is zero; that means, no path from i to j. So that means, if you have vertex i and if you a have a vertex j then, if we have a direct edge; that means, there is a path. So, that is also a in, but we are not looking for only directives we are looking for if from starting form i'th vertex, if at all you can use reduce to a j'th vertex. So, that is the idea.

So, that is the transitive closer. So, if there is a path from i to j then t i j is 1. So, some path I mean, so this is the matrix we need to find out, the t i j matrix is this problem clear. So, we have given a graph, this bar graph is a directed graph or diagraph we have given a graph and it defined a t i j which is basically 1 or 0, it is 0 if there is no path from i to j. If it is 1 if there is a path which starts from i if there is a physical path to j then, this value is t i j value is 1.

So, basically you want to find this t i j. So, how we can get this t i j? So, we want to basically use the Floyd Warshall Algorithm to have this t i j. So, what is the idea? So, idea is to use the instead of minimum and plus we will just use the or and and operation, so that is the idea.

(Refer Slide Time: 15:30)



Idea is to use the Floyd Warshall Algorithm; Floyd Warshall, but with the operation logical or and and instead of minimum or plus.

So, you remember the algorithm like you have this c i j, k is basically minimum of c i j k minus 1 comma c i k, k minus 1 plus c k j, k minus 1. So, instead of this minimum we will use the or operation and instead of this plus, we will use the and operation. So that means, either present or absent this is the by logic operation. So, either present or absent. So, like as if, if we just give the weight of each because we are not concerned about the weight of the graph. We have given a graph G which is V comma E, v 1 to where, the v is, so our concern is we have to find, so Floyd Warshall will find delta i j.

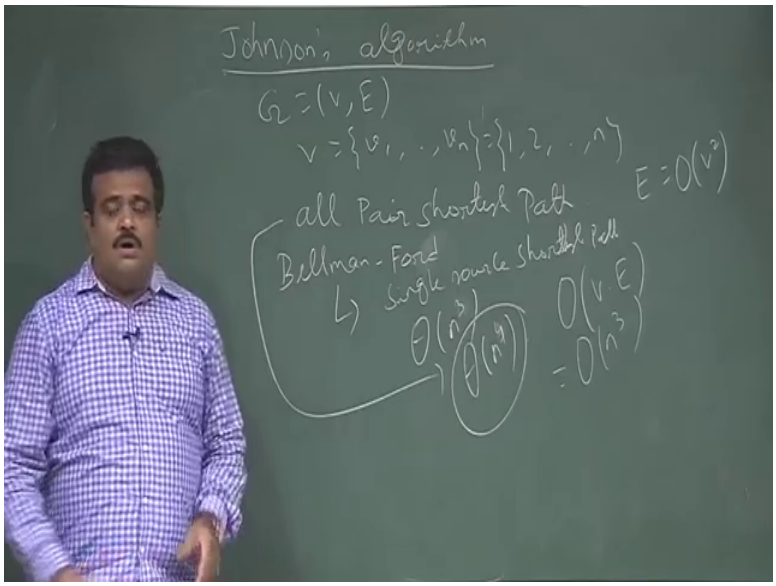
So, this is basically c i j n. So, instead of find delta i j, what we are looking for? We are trying to find t i j. So, basically if we run Floyd Warshall, the algorithm we have discussed with these two operations, this is the logical or and this is the logical and then these will gives us t i j. Why? Because if we just add a weight 1 to each of the edges then there is a path means there is a shortest path. So, we are not bothered about shortest path

here we just bother about whether there is a physical path from i to j. So, if we run these for all the vertices, this is for all the vertices because vertex number is up to n.

Then eventually this will give us t i j, it is exist and if the delta is infinity then the corresponding t i j if there is; that means, the delta infinity means there is no shortest path because you have weight we are assigning 1 to each of the edges. So, there is no question of negative weight. So, if delta is basically infinity then the corresponding t i j is, I mean basically 0; that means, there is no path from i to j. So, that is the idea. So, this is the application of Floyd Warshall on the transitive closure. So, now, we will talk about another applications. So, means another algorithm which is called Johnson Algorithm to finding the all pair shortest path.

So, the let us talk about another approach to finding Johnsons Algorithm.

(Refer Slide Time: 19:00)



So, this is to find the all pair shortest path. So, far we now some dynamic programming technique to find the shortest path. Now if we use the Floyd Warshall we know so, suppose if you have a graph G V comma E and V is say vertex, set of vertexes this is a 1 to n for the simplicity, we are assuming vertex as numbering as 1 to n.

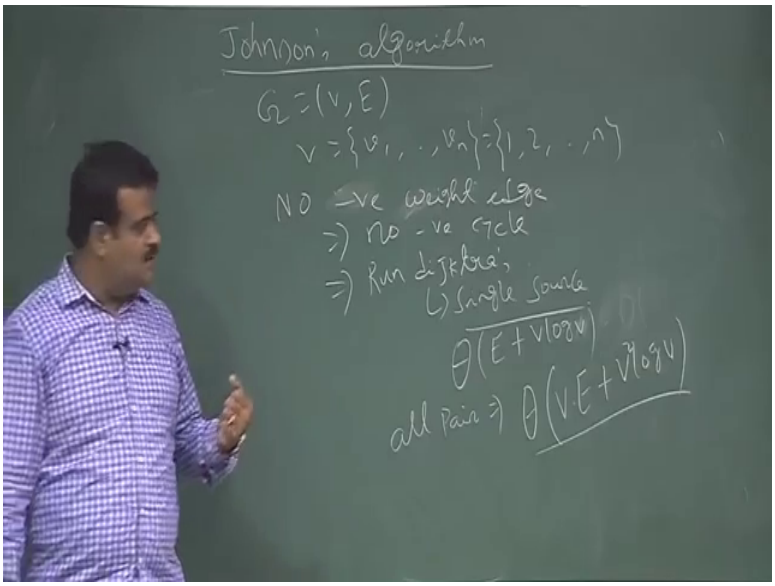
Now, we have seen for Bellman so we want to find out the all pair shortest path. Now we want to see whether we can use the algorithm, we already know like single source shortest path algorithm. We know two single source shortest path algorithm; one is

Bellman ford another one is Dijkstras, but Bellman Ford you have seen Bellman Ford will give us if we run the Bellman Ford yeah; Bellman Ford. So, if we use Bellman Ford, so this is basically the single source shortest path then, Bellman Ford itself we will take order of n square for a single source and we have to run for all the source all the vertices. So, this will give us for all pair this will give us the Bellman Ford order of n cube.

So, this is not good as the we already have order of n cube algorithm, n cube or n square. This is you have to, this is may be yeah, so you know this is n square, n cube because Bellman Ford is order of V into E right. So, V is n and E is basically order of V, order of V square basically, so this is n cube. So, this is n to the power 4, this is not as good as. So, Bellman Ford if you run say n to the power 4, but if you run Dijkstras, for Dijkstras there is a restriction; Bellman Ford can handle the negative weight edges because it can handle the negative cycle, but dijkstras cannot handle negative weight cycle.

So, somehow if we know that our graph is having no negative weightage then we one can try for Dijkstras.

(Refer Slide Time: 21:51)



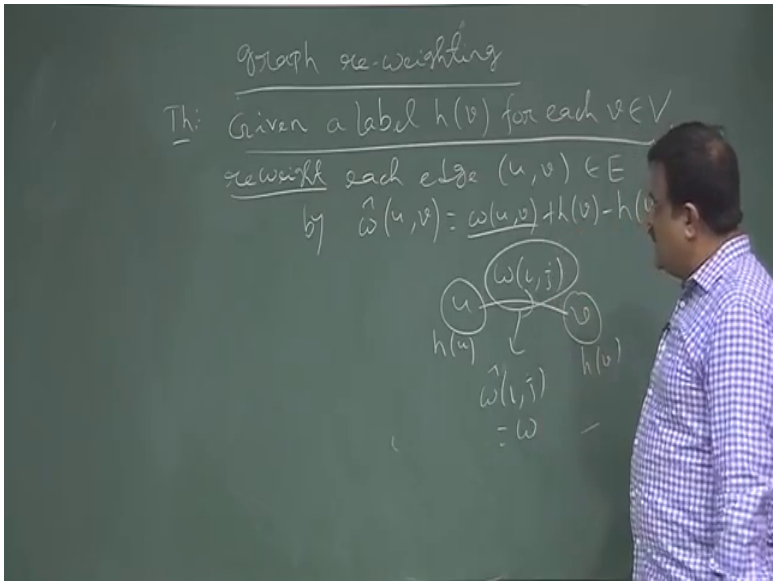
So, if no negative edge, negative weight edge. So, this imply no negative cycle, this imply we can run Dijkstras. Now if you run Dijkstras for all pair shortest path. So, Dijkstras will take how much time? The Dijkstras is also single source; single source shortest path, single source . So, now, one run of Dijkstras will tell you order of, it

depends on which data structure you are using. So, if we use the fibonacci if and the worst case amortize analysis we have seen which is order and V log n, log V sorry.

Now, this is basically order of E is order of V. So, this is basically order of any way let it be. So, now, if we run it for all pair shortest path then it will be for all pair, it will be order of V into E plus V log V, sorry V square log V. Now this is same as this is as good as the Floyd Warshall because this is at most order of n cube, but sometimes if E is not order of n cube then it is less. So, depending on E it will be either order of n cube or order n square log n.

So, this is good, now the problem is here we have this assumption. So, there should be no negative weight edge.

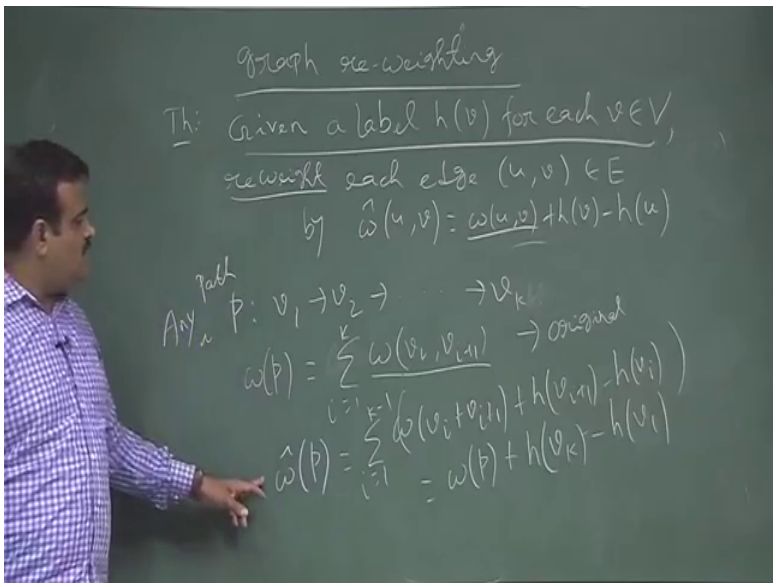
(Refer Slide Time: 23:49)



So, how we can do that? So, to ensure that we have to do something called graph relabeling, re weighting; graph re weighting. So, this is just to ensure there will be no negative weight edge. So, we are going to level the vertices, we are going to this is the theorem we are going to suppose there is a level given a level h v for each vertex V such that we reweight the graph, reweight the edge reweight each edge u v belongs to E by. So, w hat u v is equal to, so it was having a original weight w u v then, plus h of v minus h of u, w u v plus h of v minus h of u.

So, basically what we are doing? We have a two vertex these, this is w i j. Now we are leveling the vertices by h, h is a function you have to get this function. You will come to know how will get this function, h is a function such that, so we are leveling this by h of u, we are leveling this by h of v. Now this we are rewriting as w hat i v is basically w, the original weight then plus the difference between their weight. So, these we are rewriting. So, if we reweight this then, so we reweight all the vertice all the path. all the edges like this.

(Refer Slide Time: 26:10)



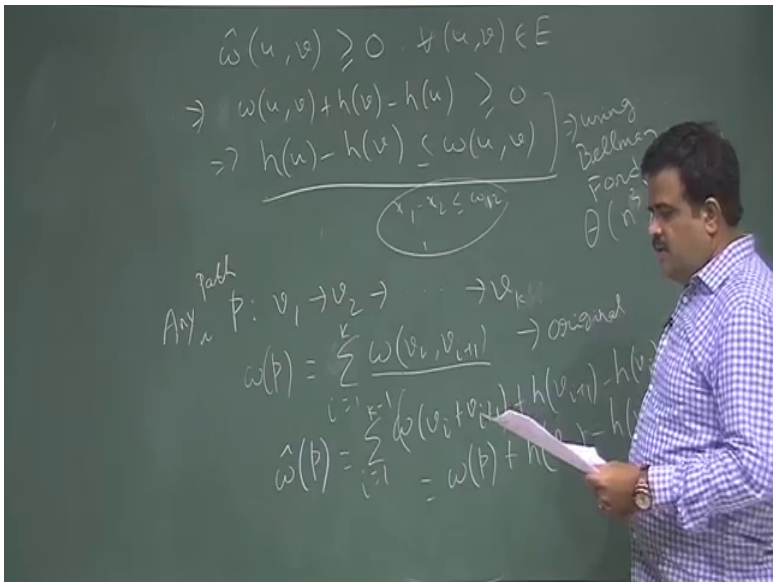
If we do that then suppose you have a path p, p from v 1 to v k.

Suppose you have a path, any path let us take a path p from this. Now what is the weight of this path? Weight of this path is basically w of summation of W of weight of these weight of these like this. So, w of v i to v i plus 1 or v i minus 1 to v i to v i plus 1 say i plus 1 and i starting from say, if it is a k plus 1, 1 to k say. So, if you do like this then what is the this is the original weight, original weight of the path. Now what is the new weight of the path after the relevelling of the weight? So, this will you basically, so we relevel each of these. So, this is basically summation of this is relevel as W of, so we relevel each of these. So, W of v i plus v of i plus 1 plus h of v of i plus 1 minus h of v i. So, if we i square 1 to k.

So, if we do this or if you put this k then you have to take it as k minus 1 any way this is just a notation. So, if we just take this, this will be basically W of p plus h of. So, this

will give us a v k minus h of v 1, v k minus h of v 1. Now we want weight of each path there should not be any negative cycle. So, we want weight your weight of each path should be positive. So that means, we want to choose the leveling, so this is the new path. So, new path will depend on this.

(Refer Slide Time: 28:50)



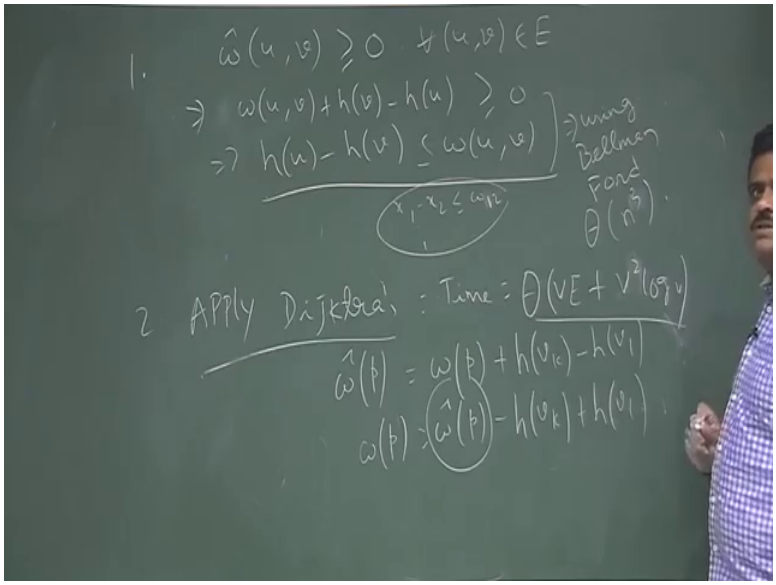
So, we want to choose the leveling in such a way that for all the edges w v must be greater than 0 for all u v belongs to E.

So, for that what we are doing? So, we want, so this is nothing, but what this is W of u v plus h of v minus h of u. So, these we want greater than 0 so; that means, what? That means, we want W of we want basically. So, these we take this side, so W of u minus w of v if less than equal to W of u v. So, these we want for all of the vertices, we want to have a h functions. So, this h is basically how come we got this h? So, we can get this h from the, this is the solving the different constant.

So, like we have seen x 1 minus x 2 is less then equal to w i 1 2 like this. So, solving that this, so again we can apply the Bellman Ford algorithm to solve this difference constant. Using the Bellman Ford algorithm we can get this h using Bellman Ford. So, using bellman ford algorithm we can get these h. So, this will take Bellman Ford into order of n cube say V comma E. And then once we relevel it then we know that there will be no negative weight h in the graph then we can run Dijkstras. So, after releveling once we

got the solution of these then we relevel these edges, we go the edge then we relevel the edges. Once we relevel the edges then we applied this is the first step.

(Refer Slide Time: 30:55)



So, once we relevel the edges then the next step is we apply the Dijkstras for all pair shortest path because there is anegative cycle. And this will take time all pair order of V into E plus V square log v if we use the data structure fibonacci if and that amortized analysis. So now, we got a path, but this path is basically shortest path is w p. Now from w p we need to get back the w, like w hat p we need to get back the w p. So, that we that we have the formula w hat p is basically w p plus h of v k minus h of v 1 and this will take. So, we we know this, so basically w p is basically if w p hat minus h of v k plus h of v. So, we know this function h and we know this by step two then we get the, so the run time is basically this one. So, this is the way because we want to use the Dijkstras Algorithm because Dijkstras Algorithm is time complexity is good compared to all other algorithm like Bellman Fords.

So, we want to use Dijkstras for further you need to have this no negative weight edge, further we need to reweighting the graph. So, these algorithm this idea is by Johnson's idea.

Thank you.