NPTEL
NPTEL ONLINE CERTIFICATION COURSE

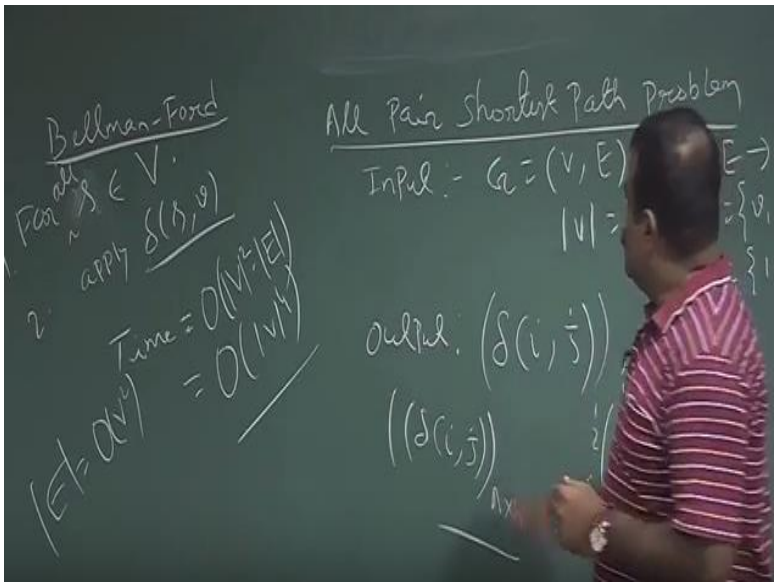
Course Name
Fundamental Algorithms:
Design and Analysis

by
Prof. Sourav Mukhpadhyay
Department of Mathematics
IIT Kharagpur

Lecture 20: Floyd Marshall

Okay, so we will talk about all pair shortest path problem.

(Refer Slide Time: 00:25)



So, so far we have seen the single pair shortest path, like Dijkstra's algorithm and Bellman Ford algorithm, so we have given the source, we have to find out the shortest path from that source to any other source. But here we want to explore all pair shortest path.

So here we have given a graph, so input is a graph (V, E) and we have age weight, so we are not restricting our self on non negative age weight, age weight, age could be negative so there is a

chance of negative cycle. And the output will be, so this V is suppose V has n matrices okay, so we can denote this by V_1, V_2 like V_n for the simplicity we can just write 1 to up to n so these are the vertices.

So and the output will be basically $\delta(i,j)$ so it is a matrix basically. So for i, j both are from 1 to n, so it is basically matrix, so this is the vertices V_1, V_2, V_n so this is the i_{th} this is $j_{th,}$ so this is basically our $\delta(i,j)$, (i,j) is the end of this matrix. So this is, but this matrix will be the output. So this is $\delta(i,j)$ this is a n across matrix, okay. So this is the all pair shortest path.

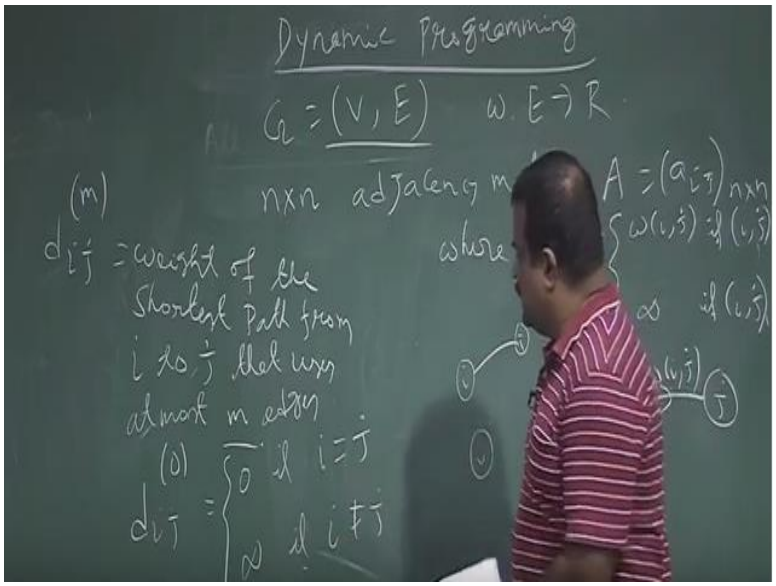
Now what we can do if we just assume that all the weights are non negative when it is possible then we can apply that Dijkstra's algorithm for all the vertices. But here our weight could be negative weight also, so one approach is the first method we can apply Bellman Ford. So we take a vertex from V and we apply Bellman Ford, so this is our S from V so we apply Bellman Ford on this s and this is we do for all the vertices, for all vertices we apply Bellman Ford.

So we get δ,u so the times end times we are applying Bellman Ford algorithm, for each time we are getting the shortest path from the particular vertex to any other vertex. So what is the time complexity for this? Bellman Ford time complexity is basically we know one time Bellman Ford timing is order of V x E, here we are running this for V times.

So basically order of (V^2 x E) so now E, if the graph is dense graph then E is basically this is the size basically, E is basically $O(V^2)$ so then it will become order of V^4. I just go to which is the exponential, but which is polynomial here so that sense it is good. But we want to see whether we can do it better than this or not.

So in particular we want to look at this problem in terms of whether we could use the dynamic programming problem, because we have seen there are indication of, there are hallmark satisfying for this dynamic programming problem for finding the shortest path.

(Refer Slide Time: 04:26)



So we want to explore that dynamic programming. We want to see over there, because we have seen that optimal sub structure is used, if you take a path, if you take a sub part of that, then that will be the shortest path from these two node, so and the multiple sub problems are also there. So dynamic programming could be applied here, so now we are going to explore that area of applying the dynamic programming problem.

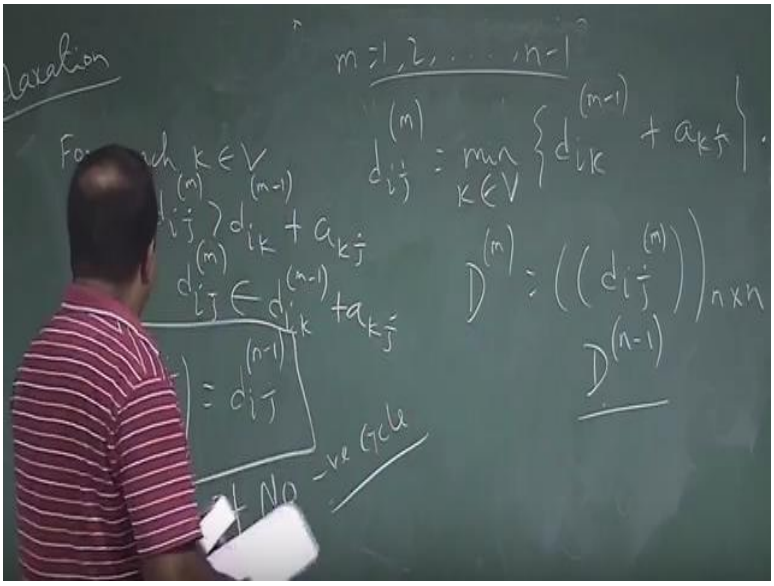
So now we consider, so we have a graph G and we have a weight function, so we consider the adjacency matrix A which is basically (a, i, j) then across A matrix where (a, i, j) is basically, so we have a vertex i, we have a vertex J if there is a direct age (a, i, j) is basically weight of that age otherwise it is infinity. So weight of, if there is age from i to j otherwise it is defined as infinity, we cannot make it 0, because 0 could be the weight of age.

So to be safe side we will make it infinity if i,j is not an edge this is what is the adjacent symmetric, so we define, so to apply a dynamic programming timing we should have a recursive formula so we will define that, we will define $d_{i,j}^{(m)}$ as weight of the shortest path from i to j that is a at most m number of edges so that means if we take i and j so we consider all the path where number of edges is maximum n.

And this is the shortest path among them so this is the weight of the shortest path from i to j where in the middle we are considering the number of edges to the maximum m it would be less than m also, but maximum at most m edge we are seeing in the in this path so that is the $d_{i,j}^{(m)}$ so what is $d_{i,j}^{(m)}$ so we want to recur recessive formula for this in order to have this dynamic programming.

So what is $d_{i,j}^{(m)}$ 0 so $d_{i,j}^{(m)}$ 0 means we want to go from i to j without seeing any edge so this is 0 if i=j otherwise it is infinity if $i \neq j$ if $i \neq j$ there is no way to because if we if there is a there direct path also if there is a edge then this is one edge so this is not 0 so if we are at i now we will remain that i that is basically 0 or 8. So this is basically the initial condition of this now what is the recessive formula okay.

(Refer Slide Time: 08:57)



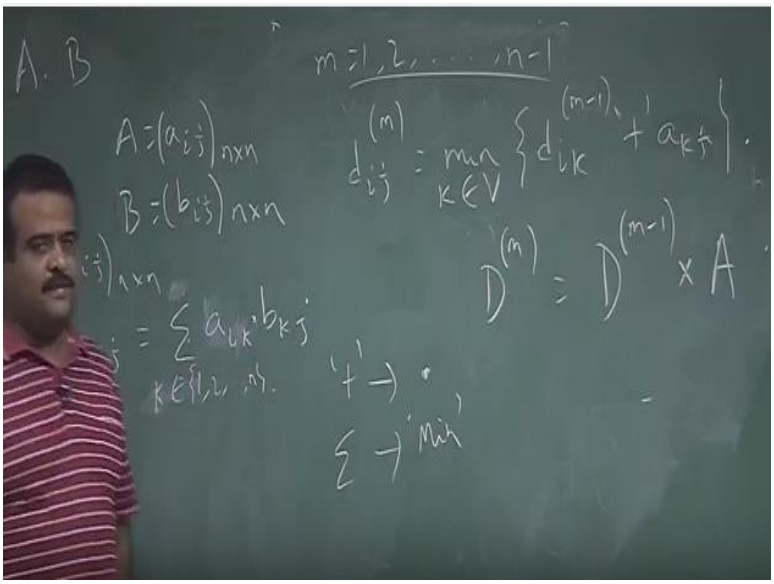
So now this is for m=0 now for m=1, 2 or 2 n-2 how we define the idea $d_{i,j}$ so basically $d_{i,j}^{(m)}$ is the minimum on the k base coming from the vertex $d_{i,j}k^{m-1}$ + ak okay so the idea we have j, i vertex we have j vertex so we have to go from i to j we have to take a path from i to j now there could be some direct edge to j okay so there could be some direct edge to j and these are the vertices is represented by this k I mean this may be v_1, v_2 something like that in general v_k

Okay so these are the direct edge going to j mean in these are all income edge to j okay so now so we want to go from i to j with seeing at most m vertices we have already explore on one vertex there you one edge you want to see at most m edge so we already explore one edge so from this to this we can go like so we want to explore maximum m-1 vertices so this is defined by d_{ik}^{m-1} then + this vertex through these vertex is basically a a_j so this is the minimum amount all this will give us so this is path from if you take any path if this is basically d_{i1} if k is1 is m-1.

So from this to this we want to explore maximum m-1 edges so from here this, so this formulas okay so this is the recursive formula we are going to use in our dynamic programming and we base case over here so now if we want to write this formula in a pseudo code how to write this so for each k belongs to V what we are doing if $d_{ij}^{(m)}$ is greater than $d_{ik}^{m-1} + a_{kj}$ then is sort of relaxation if this is better than we relax a replace by d_{kj} so this is sort of relaxation state, relaxation okay so now so this way we can find the so this from bottom to up we can find the D_{ij} n for we start with D_{i0} so if we get all the D_{ij} then what is our benefit? Our benefit is actually we are looking for δij, so what is the relationship between D_{ij} and δij? If there is no negative cycle then we should δij is the shortest path form i to j so it should give us the, this is basically $D_{ij}^{(n-1)}$if there is no negative cycle.

If no negative cycle then $D_{ij}^{(n-1)}$ should be as the this δij, so how to find out that? If you take this Di matrix we define $D^{(n)}$ the matrix basically $D_{ij}^{(n)}$ okay. So now so we want to find out $D_{ij}^{(n)}$ so basically we want to find out $D^{(n-1)}$ this matrix, so we want to so now how we can get this matrix so we want to bring some matrix multiplication technique over here.

(Refer Slide Time: 13:47)



So just to recap suppose we have two matrix A and B. And we are multiplying and we are getting a C matrix suppose A is say (a_{ij}) which is nxn and B is also b_{ij} which is nxn now what is C? C will be then C_{ij} which is nxn and C $_{ij}$ is basically Σ (V_{ik} sorry $b_{ik\ x}$ b_{ko}, so this k is running from 1 to n, okay. So this is the sum this is the okay now can you relate this with this operation? Now instead of multiplication this if we denote this addition by the multiplication over here and we denote this mean so this is also k is from all of vertices.

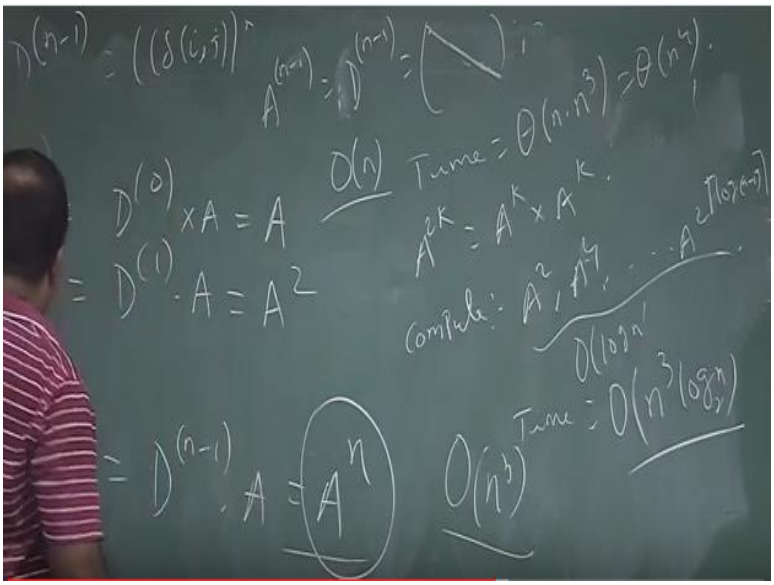
And if we denote this by this minimum if we change this operation then this is nothing but an matrix multiplication, okay. We are changing just the operation this operation is multiplication operation dot this we are taking as a plus and this operation is sum operation and this we are taking the minimum so sum over all the K and here we are finding the minimum over all the K so if we think of these two operation then this is nothing but a matrix multiplication.

So then we can just write $D^{(m)}$ is basically $D^{(m-1)}$ x A this A matrix this matrix multiplication is defined over this operation, okay. And this could be well defined if we have some algebraic fraction and this is a satisfying the semi reign property so those details we are not going here but

this satisfying the those properties to have the matrix multiplication go through. So this is nothing but a matrix multiplication.

So if we have this then what is and the D0 basically D0 basically $D_{ij\ 0}$ is basically this one so all the diagonal element are zero and all diagonal are infinity, okay all are infinity this is solve as a identity matrix, okay. So now how we can use this?

(Refer Slide Time: 16:37)



So basically we need to find $D^{(n-1)}$ so what is D1? D1 is basically D0 x a so this is basically A because this is just a identity matrix so what is D2? D2 is basically D1 x A which is basically A^2 like this.

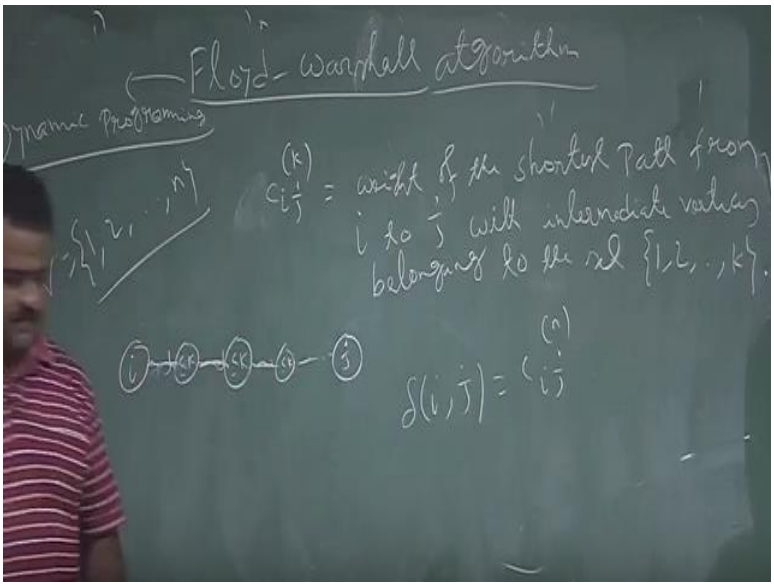
So $D^{(n)}$ is basically $D^{(n-1)}$ A which is basically A^n so now the problem is to find the A^n, I of we get A^n tat will give us basically delta i mean $A^{(n-1)}$ because we are looking for $D^{(n-1)}$ this is basically our $\delta(i, j)$ if there is no cycle so basically we need to find out A^n or (n-1) so how we can do that? So station apply cannot be applied here because these has different operation so stations may not be straight forward to apply here.

So we will try to do the matrix multiplication separately so how many matrix multiplication? We have n many so in each matrix multiplication we get order of n^3 type so the total time for A this is so O?(n^4) which is same as the Bellman Ford running for all the vertices that is also order of n^4 n is the number of vertices size of the vertex, okay so now we want to do better so how we can do better we can have a recursive fall on this dividend concurrent afterwards. So we know the A^{2k} is basically, so what we can compute we can just define concurrent forming a matrix, so we can compute a square A^4 like this $2^{n \log(n-1)}$ so then we can get the, so this is basically $\log n^2$ you can so then the time will be basically $n^3 \log n$.

But even though no this is greater than the earlier one, but we are looking for to get it by n^3 algorithm and that is the Floyd Marshall algorithm we will talk about that. Now how to detect there is any negative cycle or not, so to detect the negative cycle we will look at the diagonal element, so we are computing A^n or n-1 which is basically B^{n-1} so we will look at the diagonal element, okay. So if there is any negative quantity in the diagonal element that means there is a negative cycle, so that checking can be done in order of n times.

Because there are n elements in the diagonal, okay so now we will talk about what is call Floyd Marshall algorithm to get it in order of n^3 time.

(Refer Slide Time: 20:18)



So this is also a dynamic programming approach, Floyd Marshall, Floyd Marshall algorithm to timing the all we are satisfied, so this is also a dynamic programming approach, okay. so here what we are doing we are just, we are changing the recursive definition like so we define this $C_{ij}^{(k)}$ so this we define as weight of the shortest path, shortest path form i to j, so we have i and we have j, so we are considering all the shortest path and we are encountering the vertices over here.

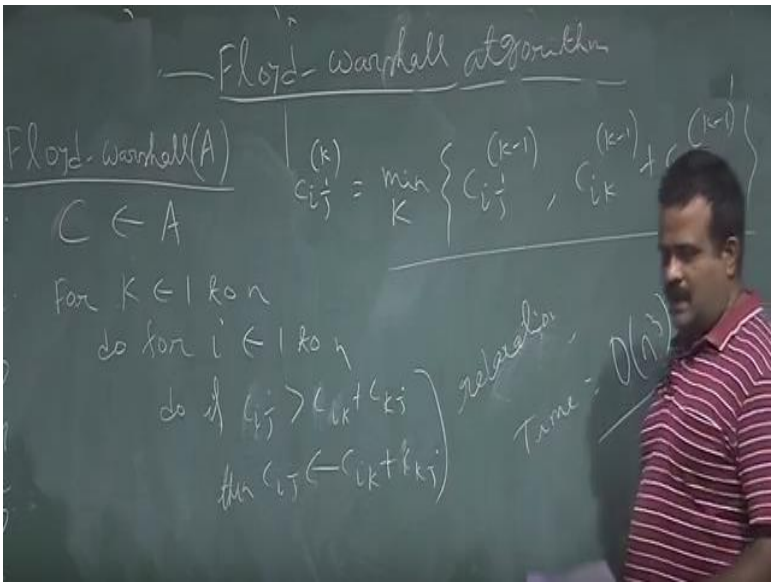
Now all the vertices we are looking are level by less than k that is the $C_{ij}^{(k)}$ so our vertices set is basically v1, v2, vn simplify as 1 to n, so how to write this from i to j with intermediate vertices, intermediate vertices belongs to, belonging to, to the set 1 to k. So there is no restriction on i and j, i and j could be more than k, but only thing now about the vertex we see in the, in the intermediate node that vertex they were cannot be more than k, it is at most k, it could be less than k.

But i and j could be anything it could be more than k also, okay so that is how we define this $C_{ij}^{(k)}$ now what is the benefit if we have $C_{ij}^{(k)}$ then how we get δ, basically δ is basically C_{ij}^{n} because if there is no negative cycle, if we consider a simple path then any, any path from i to j

the vertex we covered is maximum n-1. So if we just calculate the C_{ij}^{n} we are done, if there is no negative cycle and what is the base case over here, $C_{ij}^{(0)}$ so that means we want to go from i to j without seeing any vertex in the middle, so that means if there is a direct edge and that weight of that edge will be, so this is basically aij and the adjacent symmetric which we have defined in the beginning of this lecture.

Okay, so now how to get the recurrence for this C_{ij}, $C_{ij}^{(k),}$ we need to have a recurrence in order to apply the dynamic programming technique. Okay, so we want to go from i to j and in the middle we want to see all the vertices whose level are less than k, so it may happen that there is no vertex whose level is k so that means this is basically, this is basically cij a – 1 this is on possibilities so there is no vertex on level k. Now suppose there is vertex of level k and we go from here to here and then from here to here, so this is basically cik. Now we do not see any vertex of level k here and this is basically ckj of k – 1 so the basically cijk will be the minimum number of these two.

(Refer Slide Time: 24:56)



So yeah ij this will be the minimum among this k, minimum among this to this path or this path so this is basically ci(k – 1) or Cik (k – 1 + Ckj k – 1) okay, there is another there is another

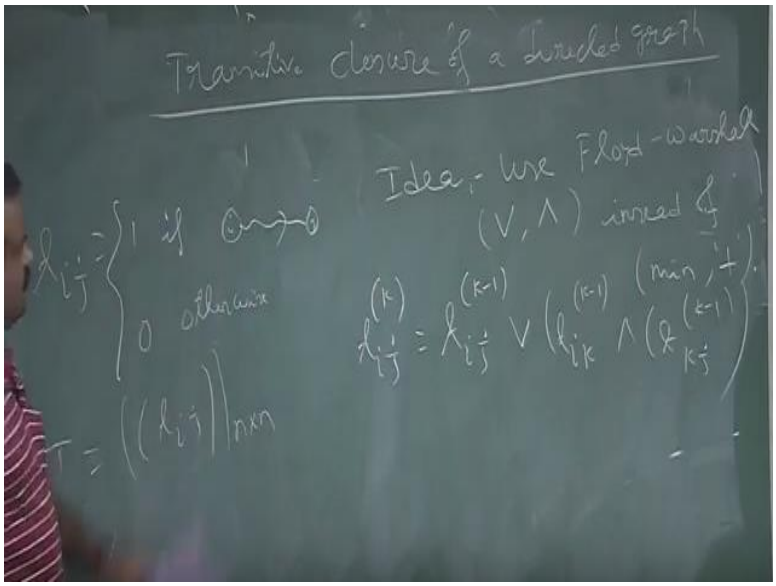
possibilities these that we may have more than one vertex vj level by k but that will come under a loop and we can ignore that loop because we are looking for the shortest path okay, so this is the recursive formula we will use for our dynamic programming.

Problem, so now this will give us a pseudo code for Bellman code. So let us write the code for Bellman code, sorry Floyd Marshall, so the input is a graph or the adjacency matrix, okay so we initialize c by these adjacency matrix, C vector by C matrix, basically C is C_i we want to compute C^n basically so these while you compute are initialize by this and then for k, for 1 to n do for i = 1 to n, this we are just writing in the C code, do if this is also sort of relaxation, C do if, if C_{ij} the current value of C_{ij}.

Is greater than Ck sorry $C_{ik} + C_{kj}$ these are small c these are the element in the matrix if this is the case then this is sort of relaxation then we replace the value of C_{ij} by the new $C_{ik} + C_{kj,}$ so this is sort of, this is also this is we know this by relaxation, that is it, so this is the code for Floyd Marshall, so now what is the runtime for this? Basically you have three loops so this is basically order of and each loop is having size n^3 order of n^3 so which is better than the earlier one.

Now we will see some application of this Floyd Marshall Algorithm in a graph, so one application is called transitive closure of a graph okay.

(Refer Slide Time: 28:04)



So transitive closure of a graph of a directed graph, so how we define transitive closure, so we know if there is a path from say u u and v, if there is a, if there is h from u to v and if there is h from v to w then by transitivity log if A is less than v and v is less than C then transitivity we telling is a less than c.

So then we can have a then we can say that there is a path from u to v, this is a path so this we define by this T_{ij}, T_{ij} is one if there is a path from i to j, so this is i vertex, this is j vertex if there is a path from i to j then we define this as one otherwise if there is no path from i to j we define this as 0. So this T matrix is basically transitive, transitivity closure matrix so we want to get this T matrix using the Floyd Marshall algorithm, so the idea is very simple we will just change the operation, so here we do not need to calculate the shortest path, we just need to calculate.

Whether there is a path or not so just we will do the logical operation like or and ending, so we just, idea is to use the Floyd version with the operation or and ending instead, of instead of mean or class, because here we are just looking for whether there is a path, we do not care about the shortest path weight so if there is a path then it will give us a one otherwise it will give us a 0, so T_{ij} here we define as T_{ij} k – 1 or so just we are.

Changing the operation and remaining, basically we are running the Floyd watts algorithm okay, that is it so this will give us this transitivity closure matrix and the time to calculate this is order of $n^{3,}$ okay, and thank you.