Course: Introduction to Graph Algorithms

Professor: C Pandu Rangan

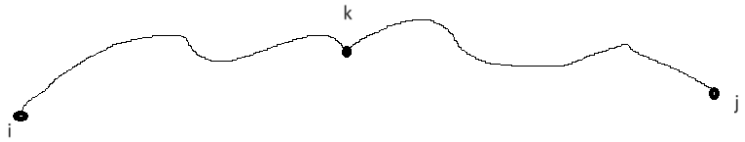
Department: Computer Science and Engineering

Institute: IISc

Week: 05

Lecture 21 All Pair Shortest Path 3&4

Namaskara we are continuing our discussions on all pair shortest path problem. We have seen two algorithms which are based on the Bellman equations for bounded path lengths and when we map the equation to something similar to matrix multiplication we got a more efficient algorithm. We are going to see somewhat similar algorithm but this is going to be based on a slightly different approach. The algorithms that we have described earlier are based on the last edge of the shortest path, okay. So considering the last edge and the remaining edges of the shortest path we derived the recurrence equations. Now we are going to generalize it for an arbitrary intermediate node, so you can say in this picture a path from i to j and k is an intermediate vertex, arbitrary intermediate vertex it could be anywhere in the path it need not be the last edge okay.



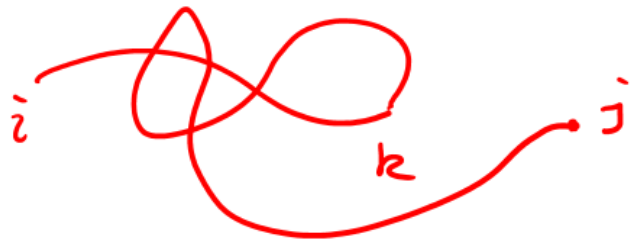
If k is an intermediate vertex in a shortest path from i to j. the portion of the P from i to k as well as the portion of the path from k to j they are all shortest paths. so let us assume that this is the shortest path this part is a shortest path and again this part is also a shortest path okay. For any intermediate vertex both these parts are shortest path.

This part the portion of the path so the whole thing is called P, P from i to j, P from i to j is written as P from i to k plus P from k to j equals P ij,

$$P(i,k) + P(k,j) = P(i,j)$$

With respect to an intermediate vertex we are split into two parts, the claim is this part is a shortest path and this part is also a shortest path. Not only this we have a kind of a converse for this, by combining shortest paths. in some careful manner we will be able to get bigger shortest path, it is a kind of converse for that okay. Suppose Qik is the shortest path from i to k and let us assume that it has got at most l edges and Rkj with the shortest path from k to j with at most l edges okay so you have. Now let us combine for various k right, k is an arbitrary intermediate vertex and k can be any vertex other than i and j. So

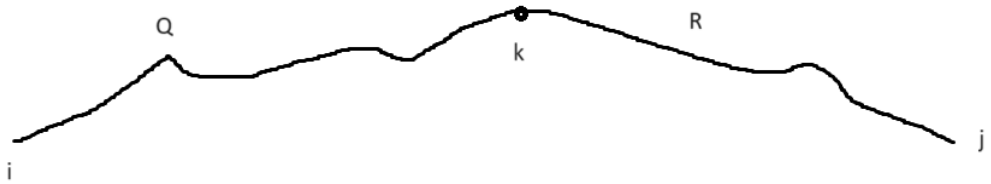
let Wikj be the walk obtained by concatenating Qik and Rkj okay notice that I am using the word walk okay, when two paths are concatenated it could be like this this might be a path from i to k this might be a path from k to j right that means you see that when you combine them there is repetition and other things possible.



Only when they are vertex disjoint they get connected like this and you get a path if the paths are vertex disjoint, and if they are attached at one end the whole thing will be a path. If the paths are not vertex disjoint, when you combine them you get a walk. So let Wijk be the walk obtained by concatenating Qik and Rkj. So let Pij be the shortest walk among all these things, okay the shortest one.

Then Pij is a path, Pij is a shortest path from i to j with less than or equal to 2l edges. Remember G is a graph that has got no negative cycle, no 0 cycle, so if a walk has a cycle, A walk has a cycle this cycle can be simply knocked out and you will get a shorter one. Therefore, the shortest walk will always be a path, if a walk has got a cycle, the edges and vertices in that cycle can be removed and the repetition can be reduced. In this way all repetitions can be removed. That means you are going to get a path, this is true only for the graph that has got no negative cycle or 0 cycle.

Our graph has that property that is the reason why anything that is shortest that will be a path, okay and the total length will be 2l, why it is 2l, you have a path of length, And another path of length l, together there will be 2l edges, so it is a walk of maximum size 2l, it will have maximum 2l edges and this is a shortest among them, that is the reason why it will be a shortest it will be a path and the upper bound for the number of edges will be 2l okay. So this is what the picture like combining Q and R.



The first claim is that if k is an intermediate vertex then the portion from i to k will be the shortest one. Suppose this is not the shortest the part of Q, the part of the P name the Q from i to k that is not the shortest, consider the shortest path same kind of an argument

okay Q dash I do not know again it is causing some problem this is Q dash, Q dash is the shortest path, therefore weight of Q dash will be less than weight of Q.

$$w(Q') < w(Q)$$

Now Q dash plus R maybe a walk, every walk.

$$Q' + R = \text{walk}$$

Has a sub walk which is a path and its weight will be less than or equal to weight of the walk this standard property in a graph that has got no negative cycle. If a walk has a cycle I can knock the cycle and get a shorter walk and in this way I can keep knocking all the cycles. I can get a sub walk which is in fact a path and weight of that path will be less than or equal to weight of the walk. The reason is everything that has knocked out is positive all cycles are positive no negative or 0 cycles.

So when you knock a positive cycle out, the weight reduces so Q plus maybe a walk and it has a path P dash such that weight of P dash is less than weight of Q dash plus R.

$$w(P') \leq w(Q' + R)$$

It has a path which has this property less than or equal to okay. If the walk itself is a path it will be equal if the walk has cycles you are going to remove the cycles and get a path with a smaller weight, therefore smaller than or equal to this. Now you can see that weight of P dash which is less than or equal to weight of Q dash plus R which is less than or equal to weight of Q dash plus weight of R, which is less than weight of Q plus weight of R weight of Q plus weight of R is equal to weight of P, that means weight of P dash is less than weight of P

$$w(P') \leq w(Q' + R)$$

$$\leq w(Q') + w(R)$$

$$< w(Q) + w(R)$$

$$= w(P)$$

$$w(P') < w(P)$$

This contradicts the minimality of P, this contradicts, P is the shortest path. Now I have found out another path from i to j which has smaller weight that is not possible you cannot have something smaller than the shortest. So this contradicts the minimality, therefore such a Q dash cannot exist hence such a Q cannot exist, which implies Q is a shortest path, similarly R is a shortest path. Therefore, if you take a path P from i to j and split the shortest path at any point any intermediate point these portions will be shortest

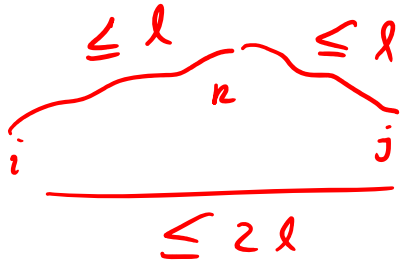
paths. Therefore the shortest path from i to j is a combination of shortest path from i to k and k to j.

You do not know for which k this is true right. We do not know the path yet. I know that the path has that property. It is a combination of two shortest paths from i to k and k to j but I do not know k. So what I do is I do the combination for every k and take the minimum right.

The natural way to when you do not know exhaustively search, I do not know the k so for every k, I compute this. I can compute, note that these are the shortest paths with smaller bound, if P has length l, all these things will have length less than l. In fact what we have seen is by the way of combining the upper bound is 2l, right. So here is the Dij2l, Dij2l is minimum of Dlik plus Dlkj.

$$D_{ij}^{(2l)} = Min\left\{D_{ik}^{(l)} + D_{kj}^{(l)}\right\}$$

There is a shortest path with upper bound on edges with l. So this is less than or equal to l, i to k this is less than or equal to l, k to j this is less than or equal to l, this is less than or equal to l, the whole will be less than or equal to 2l.



So something on the notation I would like to discuss now. The notation with parenthesis in the top defines a sequence of matrices. So D1, D2 that defines a sequence of matrices. The notation of a matrix without the parenthesis is the power of the matrix. For example, w times w times w is w cube.

This is a matrix multiplication, done twice in other words w cube, and if you take w square dot w square it will be w4. So when I write a number without parenthesis it represents the exponent or the power of the matrix but when I put a parenthesis it defines a sequence of matrices. So I want to have a sequence of matrices and the sequence of matrices is denoted BY D1, D2 and Dn and so on. Here D2l is given as a combination of

these two and this combination shows that it is like a matrix multiplication we have already seen min and plus. Min would be converted to sigma plus will be converted to dot it will look identical to a matrix multiplication operation. Therefore D of 2l can be written as D of l square okay so we must add a parenthesis D of 2l.

$$D^{(2l)} = D^l \cdot D^l = \left(D^{(l)}\right)^2$$

Therefore just by keep squaring you are going to get higher and higher powered values in the sequence. So you do not have to compute D1, D2, D3 and so on, from D1 you can go to D2 from D2 by squaring you can go to D4, from D4 by squaring you can go to D8 and so on. I can always go higher value because I know that the shortest path not only that any path cannot have more than n minus 1 edges. Therefore D of n minus 1 is equal to D of l for all l greater than or equal to l minus 1 the same thing that we did earlier. All higher powers are same so I can go to the nearest higher power which is a power because in the doubling process I get a power of 2.

$$D^{(n-1)} = D^{(l)} \text{ for all } l \geq (n-1)$$

The same illustration you can recall, the way in which I would compute D13 is, I know that D13 is D16, D16 can be easily computed as a D1 to D2 to D4 to D8 to D16. It is easy to compute so I will compute D13 by computing D16 easily, so maximum I am going to have log n steps. Because 2 power log n will be greater than n minus 1 so by stepping, by squaring log n times I get the answer.

$$2^{\log n} \geq n > n-1$$

So I start from D equal to W, notice that I am not starting at D equal to D0 as I did in the previous algorithm. I am starting at D equals D1, D1 is nothing but W, so D is W then I keep squaring it and I return D after log n iterations that is it and a problem is solved. It is solved in a different way, we are now considering the intermediate vertex. And based on that approach we have got another n cube log n algorithm the complexity is again n cube log n, the reason is I am doing log n times the squaring, each squaring is a matrix multiplication. A matrix multiplication cost n cube steps log n matrix multiplications they would cost log n n cube step this algorithm is due to Fisher and Mayer.

For the all pair shortest path problem we have seen an n power 4 algorithm and 2 n cube log n algorithm.We are going to discuss about another algorithm whose complexity is order n cube and it is based on an ingenious and an out of the box thinking. We will introduce the notion of k paths and then discuss the algorithm by Floyd and Warshall whose complexity is order n cube, then we conclude the discussions on all pair shortest path problem by discussing Johnson's algorithm which is just a very clever combination of Dijkstra's algorithm and Bellman and Ford algorithm to achieve further efficiency on sparse graphs, okay. So this algorithm is going to be based on a different formulation on

the intermediate vertex. There are several researchers who have worked on this idea and more or less at the same window of time several algorithms based on this idea have been published okay. Call a path from i to j is a k path from i to j if all intermediate nodes or less than or equal to k that is the path from i to j pass through the set of vertices 1 through k, only these vertices are there. So every k path is automatically an l path for all l greater than k if I have a 20 path, it is automatically a 21 path, 27 path, 43 path and so on, because all intermediate vertices are less than or equal to 20 means they are less than or equal to 25, they are less than or equal to 43 and so on. So any k path is automatically an l path for all l greater than k. So what is a 0 path? 0 path is just an edge i to j if it exists no intermediate vertex. So i to j,no intermediate vertex means it is a path with no intermediate vertex, it is an edge therefore 0 path is nothing but an edge.

Note that the k is independent of i and j, nodes i and j are source and destination vertices. The upper bound on k is applicable only for the intermediate nodes not for the end nodes, end nodes and k are independent okay. For example I can have 15 it can go via 7, 12, 8 this is a 12 path connecting 15 and 20, Suppose instead of 8 it is 18, it is an 18 path because all intermediate nodes are less than or equal to 18 you can see that the k path, the k value can be smaller than both and it can be larger than both it can be an in between value, k is an independent value. So let delta kij be the weight of the shortest k path from i to j, since n is a largest vertex label, you have vertex v is defined from 1 to n, so all the intermediate vertices are obviously less than or equal to n, the largest value an intermediate vertex can have is n. Therefore delta n ij equal to delta ij for all ij okay because n is the last this is n let me write down that delta n ij is equal to delta n ij is equal to delta ij. because delta nij is the shortest among all paths.

$$\delta_n(i,j) = \delta(i,j) \; \forall \; i,j \; \in V$$

All paths are included in this set therefore delta nij is, therefore delta nij is the value that we are looking for our approach is same I want delta nij if I know delta k minus 1 ij for all i and j

$$\delta_{k-1}(i,j) \; \forall (i,j)$$

$$\delta_k(i,j) \; \forall (i,j)$$

Can I use this to compute delta k ij for all ij is it possible for me to do this okay. If it is possible to do that, then I will start from delta 0 ij keep applying that process again and again and reach delta n, right from delta 0, I go to delta 1 for all ij and from delta 1 I go to delta 2 and I stop it when I get delta n, the standard approach okay. I know delta 0 ij, delta 0 ij is w ij because if an edge exists , that is the path and its weight is the weight of the shortest path if it does not exist it is infinity and that is there in w ij, therefore delta 0ij is w ij

$$\delta_o(i,j) = w(i,j)$$

And I am going to define A power k is given by delta kij alright.

$$A^{(k)} = \left[a_{ij}^{(k)}\right]_{n\times n} \text{ by } a_{ij}^{(k)} = \delta_k(i,j)$$

Now a k path from i to j may contain k or may not contain k, if it does not contain k, all its intermediate vertices are less than or equal to k minus 1 and hence it is in fact a k minus 1 path right. So consider this path i to j, k is not there, all of them are less than or equal to k and k is not there therefore all of them are less than or equal to k minus 1 since all of them are less than or equal to k minus 1, it is a k minus 1 path. Therefore delta k ij could be the minimum in this which is the delta k minus 1,

$$\delta_k(i,j) = \delta_{k-1}(i,j)$$

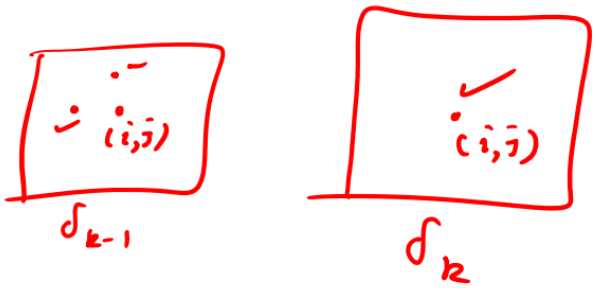
Therefore this is one possibility if the k path contains k. That is the second case if it contains k here is the picture for them the k path contains k therefore this part does not have k it does not have k it is all less than or equal to k therefore it is less than or equal to k minus 1, same is the argument for that. We know that if k is an intermediate vertex, this condition is true and they are the shortest path. Therefore, delta k ij equal to delta k minus 1 ik this is delta k minus 1 ik plus delta k minus 1 kj.

$$\delta_k(i,j) = \delta_{k-1}(i,k) + \delta_{k-1}(k,j)$$

This is the length of the shortest path and this is the length of the second shortest path they are combined and then you so this is the case if k is there we do not know whether it is there or not so what we do is that we find the minimum of these two.

$$\delta_k(i,j) = \text{Min } \{\delta_{k-1}(i,j), \delta_{k-1}(i,k) + \delta_{k-1}(k,j)\}$$

This is the case when k is not there, this is the case when k is there. The minimum of these two take care of all possibilities, therefore delta k ij is computed from delta k minus 1 matrix. So in the delta k minus 1 matrix what are the entities you look at, delta k minus 1 ij okay, so here. delta k minus 1 matrix delta k matrix in delta k matrix I want to compute this if I want to compute that what is that I have to do I have to look at this value the same ij value ik and ik and kj depending on where k is let us say ik, k is less than j let us say then ik will be somewhere here, kj will be somewhere here, ik and kj.



So just look at these 3 values, that is all we are not going to look at anything else. So to compute delta one addition and one comparison that is it. So you can see how one matrix is generated from the previous matrix, exactly n square steps, n square addition, n square comparisons because per element one addition and one comparison, one addition and one comparison to evaluate one element, there are n square element. Therefore n square additions and n square comparisons, you can see, therefore this process for each hop n square and n square and n square each hop is n square there are n such hops.

$$A^{(0)} \rightarrow A^{(1)} \rightarrow A^{(2)} \rightarrow \cdots \rightarrow A^{(n-1)} \rightarrow A^{(n)}$$

Therefore the total complexity is n cube, so with n cube steps we were able to solve the problem, it is an amazingly clever solution right. Our previous solutions where all either n power 4 or n cube log n, here by looking at the collection of shortest path, from the maximum value of the intermediate one and not the lengths we got a formulation and for this formulation the time it takes to go from one matrix to the next matrix is only n square, therefore I can pass through all the steps in n cube time amazingly clever and simple algorithm.

So here is the algorithm Floyd Warshall for k equal to 1 to n, I have to compute Ak using Ak minus 1, that is I compute A1 using A0, I compute A2 using a1, I compute A3 using A2, I similarly compute An. How do I do that, how do I go from one to the next if I want to compute kth element, I have to consult k minus 1 element and then get the job done okay. So each is taking one comparison and one addition. Therefore the total complexity is n cube very simple algorithm stunningly simple algorithm and this is called the Warshall's algorithm and the complexity is n cube. The problem with Warshall's algorithm is whether the graph is a dense or sparse the complexity is always n cube okay suppose the graph is a sparse graph like planar graphs or any sparse Is there a way I can reduce the complexity further because that graph the vertices are same but the number of edges are small.

The number of edges can vary, for example if you have 1000 vertices the number of edges could be as high as half a million right and it could be even 5000, 6000 the same order as the number of vertices. if this is sparse is there a way I can take the advantage of the smallness of the edge set and arrive at a more efficient solution. In the worst case it

will be the same for the dense graph the complexity will be same but for sparse graph we may take an advantage of the sparsity and may arrive at a clever and that is what is done by Johnson's algorithm. This completes the discussions on Floyd Warshall's algorithms, thank you.