

IIT Madras
ONLINE DEGREE

Mathematics for Data Sciences 1
Professor. Madhavan Mukund
Chennai Mathematical Institute
Lecture No. 12.4
All-Pairs Shortest Paths

So, we are looking at shortest paths and weighted graphs.

(Refer Slide Time: 0:17)

Shortest paths in weighted graphs

Two types of shortest path problems of interest

Single source shortest paths	All pairs shortest paths
<ul style="list-style-type: none">■ Find shortest paths from a fixed vertex to every other vertex■ Transport finished product from factory (single source) to all retail outlets■ Courier company delivers items from distribution centre (single source) to addressees■ Dijkstra's algorithm (non-negative weights), Bellman-Ford algorithm (allows negative weights)	<ul style="list-style-type: none">■ Find shortest paths between every pair of vertices i and j■ Optimal airline, railway, road routes between cities■ Run Dijkstra or Bellman-Ford from each vertex■ Is there is another way?

Madhavan Mukund All-Pairs Shortest Paths Mathematics for Data Sciences

And we said that there are two types of problems that we might look at in terms of shortest paths, the one that we have looked at already is the one which is called the single-source shortest path problem, which asks you to find the shortest path from this some designated starting vertex, source vertex to every other vertex and we imagined two scenarios where this might be useful.

So, if you have a kind of factory, which is shipping out finished products to retail outlets, or if you have a courier company, which has got a centralized delivery facility from where they have to ship out their things for delivery, both of these would like to know the shortest route from wherever they are shipping out things, either the finished product or the courier deliveries to all the other locations in their transportation graph.

On the other hand, another natural problem is to find the shortest distance between every pair of vertices, and this we said would be very natural if you are running something like a travel site, somebody wants to book a ticket from somewhere to somewhere else, you should have the information about the shortest route from this starting point to the ending point, regardless of which two points they are.

So, you cannot say I am only booking tickets from this city and not from another city. So, that does not make any sense. So, if you have a travel booking site, it should be able to give you optimum routes in terms of cost, or time, or distance, or whatever, from anywhere to anywhere. So, we have seen two algorithms with a single source shortest path problem. Dijkstra's algorithm which works if there are no negative weights, and the Bellman-Ford algorithm, which works even if there are negative weights. But of course, remember that negative cycles are always forbidden because with negative cycles, the whole thing does not make sense at all, the idea of a shortest path is meaningless.

So, how do we find all pair shortest paths? So, we can find the shortest path from a single vertex to every other vertex. So, we can just iterate this by starting it from every vertex. So, I started from 0, I get all paths from 0, I go to 1, and I started from 1 and I get all paths from 1. If I do this $n - 1$ times, I will have the shortest path from every starting vertex to every ending vertex. So, this is one way we could do this is just to repeatedly run Dijkstra's or Bellman-Ford, across all starting points. But what we are trying to see is whether there is another way to do this, which does not involve this kind of going back and restarting the calculation. So, that is what we will look at in this lecture.

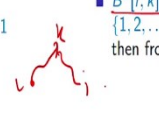
(Refer Slide Time: 02:28)

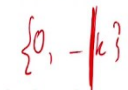
Transitive closure


- Recall transitive closure algorithm
- Adjacency matrix A represents paths of length 1
- Matrix multiplication, $A^2 = A \times A$
 - $A^2[i, j] = 1$ if there is a path of length 2 from i to j
 - For some k , $A[i, k] = A[k, j] = 1$
- In general, $A^{\ell+1} = A^\ell \times A$,
 - $A^{\ell+1}[i, j] = 1$ if there is a path of length $\ell+1$ from i to j
 - For some k , $A^\ell[i, k] = 1$, $A[k, j] = 1$
- $A^+ = A + A^2 + \dots + A^{n-1}$

An alternative approach

- $B^k[i, j] = 1$ if there is path from i to j via vertices $\{0, 1, \dots, k-1\}$
 - Constraint applies only to intermediate vertices i to j
- $B^0[i, j] = 1$ if there is a direct edge
- $B^0 = A$
- $B^{k+1}[i, j] = 1$ if
 - $B^k[i, j] = 1$ — can already reach j from i via $\{1, 2, \dots, k-1\}$
 - $B^k[i, k] = 1$ and $B^k[k, j] = 1$ — use $\{1, 2, \dots, k-1\}$ to go from i to k and then from k to j







Madhavan Mukund
All-Pairs Shortest Paths
Mathematics for Data Science

So, we will begin with something that we saw earlier, namely, the transitive closure. So, remember that the transitive closure, we discussed in the context of unweighted graphs and we said the transitive closure, is the reachability calculation. So, we wanted to find out just $l[i, k]$ all-pairs shortest paths, we wanted to find out all pairs reachability given i and j , can I get from i to j in this graph? Is there a path from i to j . So, again, there, we said that we could have run

BFS or DFS from every starting vertex and got it. But instead, we ran this other algorithm, which uses this matrix approach.

So, we started with this adjacency matrix and we said that an adjacency matrix represents implicitly edges, which are paths of length 1. And then we wanted to use this matrix and the operation of matrix multiplication to bootstrap this from length of 1, paths of length 1 to paths of length 2, and so on. So, we said a path of length 2 decomposes are two paths of length 1. So, if I want to find out the path of length 2 from i to j , I just have to check for some intermediate k , with that I can go from i to k in one step, and then k to j and another step.

And we can generalize this and we discussed why this is matrix multiplication. We can generalize this to l steps. If I know how to get an l steps from i to j , then I know how to get an $l + 1$ steps from i to j , because I go to an intermediate k and l steps, and then from k to j in one step. So, by decomposing $l + 1$ step as l steps followed by one step, I can again do a matrix multiplication and do A^l times A is equal to A^{l+1} .

And finally, in the transitive closure, we want to look for a path of any length. But paths remember, are always bounded by length $n - 1$ because there is no need to go through a loop to read something. So, it is sufficient to calculate path length 1 to $n - 1$, and then take the sum, the sum, if you remember, was the OR, we said that this is the OR of so if there is a 1, if $A[i,j]$ is 1 in any of these matrices, then there is a path of length 1,2,3, or up to $n - 1$ and therefore $A^+[i,j]$ the transitive closure reports one.

So, we are going to slightly reformulate this calculation of transitive closure, so we are going to do so. This is a kind of inductive definition, so, we first find transitive closure by first finding paths of length 1, paths of length 2, and so on and then we take the cumulative effect of all this by doing the $+$ at the end. Now, we are going to do another inductive thing but we are going to use a slightly different way of calculating simpler paths and complicated paths not in terms of path links, but which vertices we will go through.

So, we will just to distinguish it from the earlier one, let us call it B . So, earlier we had A sub A to the K . If I said $A^k[i,j]$. So, this was referring to paths of length k , it says that there is a path of length k from i to j . So, here I have something else, $B, k[i,j]$, does not refer to the length, it refers to which vertices I am allowed to visit on the way from i to j , and not obliged to visit all of them. But I cannot visit anything outside the set. So, if I say k as my superscript, it says, I am only allowed to go through vertices, 0 to $k - 1$, I am not allowed to visit any other vertex going from i to j .

So, if I have this constraint, can I reach i to j . So, it could be a direct thing in which I do not go to anything, it could be 3 of these vertices, it could be 2 of these vertices, it could be any number, but it cannot be outside the set. Now, remember, it is going to be a path. So, I never want to visit a vertex in the set more than once. But that does not matter so much, instead of telling me that I can get from i to j without going outside the set 0 to $k - 1$.

And this constraint, of course, applies only to the vertices a visit in between, so i and j need not be between 0 and $k - 1$, i could be anything j could be anything. So, I am just saying that when I leave i and before I reach j in between I should not see anything, which is outside the set 0 to $k - 1$. So, in this setting, what would B of 0 mean? So, B of 0 by this definition, would be some 0 up to -1 , because k is 0 . So, this is saying that I am not allowed to visit any vertex in between because the set of vertices I visit in between can have at most number -1 , but I know that my vertices start with 0 , so there is no vertex in the set, which starts between 0 and ends before -1 .

So, therefore, $B^0[i,j]$ says that I cannot have any intermediate vertex because no intermediate vertex can satisfy this constraint, that its index is -1 or smaller. So, therefore, B^0 of $[i,j]$ precisely captures the fact that I can go from i to j without visiting an intermediate vertex means there is an edge from i to j . So, B^0 is just my adjacency matrix A . So, just $l[i,k]$ in the earlier case, we started off with paths of length 1 , and we said paths of length 1 are precisely our adjacency matrix A . Here, we are saying paths which do not pass through any intermediate vertices are exactly captured by an adjacency matrix A . So, the starting point of this induction is A in both cases.

So, how would we proceed? Well, I want $l[i,k]$ before, I want to calculate what would be B^k $+ 1$. So, B^{k+1} says that I am allowed to use 0 to k . So, if I am allowed to use 0 to k from i to j , then there are two possibilities. One is that I do not need this k at all, I could already do it without 0 to k . So, that means that $B^k[i,j]$ is already 1 , it means that without going to k , just staying within 0 to $k - 1$, I have a possibility of going from i to j or I need to use k .

But if I need to use k , then remember now we use this property that we implicit, that we are going to visit k only once there is no point in visiting k multiple times. So, if I need to use k , I need to get from i to k . And I need to get from k to j . But if I am using up k and going from i to k , then I cannot be using k in between, I am using k only once overall. So, going from i to k , I do not see k , going from k to j again, I do not see k . So, I can find out whether I can go from i to k using only the vertices 0 to $k - 1$.

And similarly, with that, I can go from k to j using only the vertices 0 to $k - 1$. And then I can pick these two paths. So, I have a path from i to k and I have another path from k to j . And now together, this gives me a new path from i to j , which visits k , I am forced to go to k . But in between, I do not do anything outside 0 to $k - 1$ except when I hit k . So, this is my condition now. So, I say that $B^{k+1}[i,j]$ is 1 either if it is already 1 in B^k .

So, $B^k[i,j]$ is 1 meaning I do not need k , or I go to k explicitly, in which case I go from i to k in B^k - in $k - 1$, using $k - 1$ vertices, vertices up to $k - 1$. And then I go from k to j using vertices up to $k - 1$. So, this gives me the inductive calculation and this is a different way of calculating the transitive closure.

(Refer Slide Time: 09:23)

Warshall's Algorithm

- The algorithm on the right also computes transitive closure — **Warshall's algorithm**
- $B^n[i,j] = 1$ if there is some path from i to j with intermediate vertices in $\{1, 2, \dots, n-1\}$
- $B^n = A^+$
- We adapt Warshall's algorithm to compute all-pairs shortest paths

Computing transitive closure

- $B^k[i,j] = 1$ if there is path from i to j via vertices $\{0, 1, \dots, k-1\}$
- $B^0[i,j] = A[i,j]$
 - Direct edges, no intermediate vertices
- $B^{k+1}[i,j] = 1$ if
 - $B^k[i,j] = 1$, or
 - $B^k[i,k] = 1$ and $B^k[k,j] = 1$

Madhavan Mukund All-Pairs Shortest Paths Mathematics for Physics and Engineering

So, this particular algorithm is the more standard algorithm that you see in books and this is called Warshall's algorithm. So, this is Warshall's algorithm. So, we use the other algorithm, because it is nicer in terms of describing why it is matrix multiplication. So, this requires a little more work to put it in that framework of matrix multiplication. But essentially, it is capturing this inductive property of finding more and more complicated paths.

So, here the complication is a number of which are the different vertices you have to see in between rather than the total number. The total number is what the other one is talking about the length of the path. Here, is restricting you to some subset of vertices and that subset keeps growing. Eventually, you can use all the vertices. So, this is the thing around the formula. Now, so $B^0[i,j]$ is just $A[i,j]$. And $B^{k+1}[i,j]$ is 1 if either $B^k[i,j]$ is 1 , so we already can get from i to j without using the k 'th vertex, or $B^k[i,k]$ is 1 and $B^k[k,j]$ is 1 .

So, this should look very familiar because it looks very similar to the other one. This is the interpretation of this superscript attached to the matrix is different. So, in this now, when do we stop? Well, when the superscript becomes n , it says I am allowed to use any vertex from 0 to $n - 1$, it should be I am allowed to use any vertex from 0 to $n - 1$ that means I am allowed to use any vertex at all. That means there is no constraint so, I have some path of the other.

So, if I do this up to B^n , so in the earlier case also added it up to A^1 to A^{n-1} . So, here if I do it up to B^n , starting from B^0 , then I am done. So, B^n is the same as A^+ . And now what we are going to do is we are going to see that this shortest path algorithm that we have not the shortest path this transitive closure algorithm that we have due to Warshall's is very easy to extend to the shortest path problem for all pairs.

So, remember, the transitive closure is the equivalent of all pairs reachability. Dijkstra's algorithm is the equivalent of BFS, except that we have weights. So, weighted reachability is Dijkstra's. Now, weighted all pairs reachability is all pair shortest path. So, we are going from transitive closure, which is unweighted all pair reachability to weighted all pairs reachability. So, with weights, we are asking what are the shortest paths.

(Refer Slide Time: 11:39)

Floyd-Warshall Algorithm

- Let $SP^k[i, j]$ be the length of the shortest path from i to j via vertices $\{0, 1, \dots, k-1\}$
- $SP^0[i, j] = W[i, j]$
 - No intermediate vertices, shortest path is weight of direct edge
 - Assume $W[i, j] = \infty$ if $(i, j) \notin E$

Handwritten note: $W: E \rightarrow R$

Madhavan Mukund All-Pairs Shortest Paths Mathematics for Data Science

So, we will use a similar notation. So, let us call SP for shortest path. So, shortest path with superscript k between i and j is the length of the shortest path provided I stay within vertices 0 to $k - 1$ when going from i to j . So, that is what this means. So, earlier, it was saying that there is a path. Now, I am saying among the paths that are there, what is the shortest length, I am keeping track of the shortest length, not just the fact that there is a path but there is a path of minimum length and keeping track of that minimum length.

So, what is the base case? Well, if there is no possibility of going through an intermediate vertex, then we saw that $B^0[i,j]$ was just $A[i,j]$. In this case, I do not want the edge or not edge, I want to know the weight of the edge. So, $SP^0[i,j]$ is $W[i,j]$. Remember that W is our weight matrix. So, W tells us for each edge, what is the value? So, $W[i,j]$ is the weight function. So, $W[i,j]$ tells us how much cost is there with this edge.

Now, of course, we have to take care of the fact that there is no cost and no edge carefully. So, what we will do is we will assume that when we have no edge the weight is somehow assigned to a very large number in particular, we can treat it mathematically we can treat it to be infinity. So, if there is no edge, $SP^0[i,j]$ will report infinity, if there is an edge, it will report the weight of the edge.

(Refer Slide Time: 13:06)

Floyd-Warshall Algorithm

- Let $SP^k[i,j]$ be the length of the shortest path from i to j via vertices $\{0, 1, \dots, k-1\}$
- $SP^0[i,j] = W[i,j]$
 - No intermediate vertices, shortest path is weight of direct edge
 - Assume $W[i,j] = \infty$ if $(i,j) \notin E$
- $SP^{k+1}[i,j]$ is the minimum of
 - $SP^k[i,j]$
Shortest path using only $\{0, 1, \dots, k-1\}$
 - $SP^k[i,k] + SP^k[k,j]$
Combine shortest path from i to k and k to j
- $SP^n[i,j] = 1$ is the length of the shortest path overall from i to j
 - Intermediate vertices lie in $\{1, 2, \dots, n-1\}$

Madhavan Mukund All-Pairs Shortest Paths Mathematics for Data Science

Now, the update rule is slightly different from the transitive closure rule. So, this is the slide that you need. So, either I can go from i to j without using k . So, $SP^{k+1}[i,j]$ is either $SP^k[i,j]$, or I go using k , and then I have to check the cost. Now earlier, I just want to check whether this was there, or that was there, I just needed to do the logical or either I can go without k or I can go with k and I take the OR.

Here, I have to say how much does it cost me to go without k ? And how much can I gain by going with k ? So, I either take the cost of going with k which is the shortest path to k from i followed by the shortest path from k to j , or I take the shortest path without going through vertex k at all and I will take the minimum of these two. So, this is my update rule for the shortest path matrix.

And as usual, so this again, this same typo, but the shortest path matrix, if I now calculate up to the n th step, then it will tell me the shortest path among all paths, the length of the shortest path among all paths that go through any intermediate vertex from 0 to $n - 1$ and that covers all possible intermediate vertices. So, this will be the overall shortest path from i to j .

(Refer Slide Time: 14:21)

Floyd-Warshall Algorithm

SP^0	0	1	2	3	4	5	6	7
0	∞	10	∞	∞	∞	∞	∞	8
1	∞	∞	∞	∞	∞	2	∞	∞
2	∞	1	∞	1	∞	∞	∞	∞
3	∞	∞	∞	∞	3	∞	∞	∞
4	∞	∞	∞	∞	∞	-1	∞	∞
5	∞	∞	-2	∞	∞	∞	∞	∞
6	∞	-4	∞	∞	∞	-1	∞	∞
7	∞	∞	∞	∞	∞	∞	1	∞

Floyd-Warshall Algorithm

SP^0	0	1	2	3	4	5	6	7
0	∞	10	∞	∞	∞	∞	∞	8
1	∞	∞	∞	∞	∞	2	∞	∞
2	∞	1	∞	1	∞	∞	∞	∞
3	∞	∞	∞	∞	3	∞	∞	∞
4	∞	∞	∞	∞	∞	-1	∞	∞
5	∞	∞	-2	∞	∞	∞	∞	∞
6	∞	-4	∞	∞	∞	-1	∞	∞
7	∞	∞	∞	∞	∞	∞	1	∞

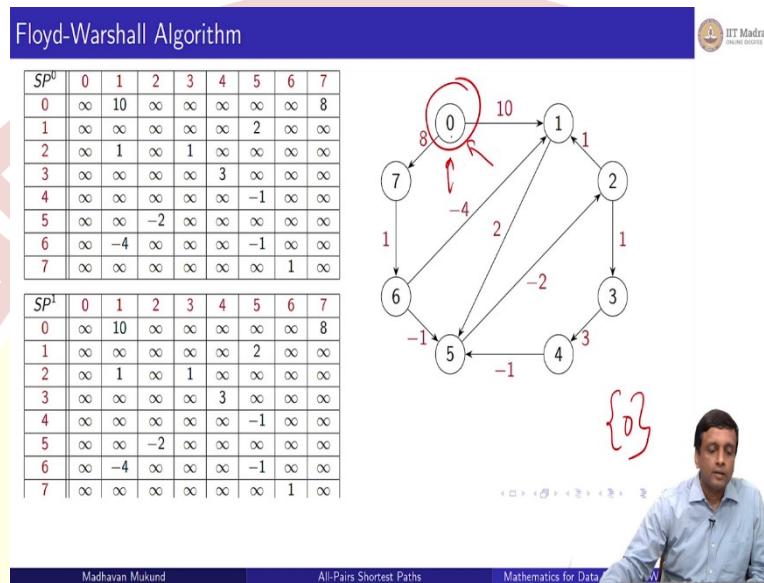
Floyd-Warshall Algorithm

SP^0	0	1	2	3	4	5	6	7
0	∞	10	∞	∞	∞	∞	∞	8
1	∞	∞	∞	∞	∞	2	∞	∞
2	∞	1	∞	1	∞	∞	∞	∞
3	∞	∞	∞	∞	3	∞	∞	∞
4	∞	∞	∞	∞	∞	-1	∞	∞
5	∞	∞	-2	∞	∞	∞	∞	∞
6	∞	-4	∞	∞	∞	-1	∞	∞
7	∞	∞	∞	∞	∞	∞	1	∞

So, this particular algorithm is called the Floyd-Warshall algorithm. So, the original algorithm for transitive closure is due to Warshall and Floyd is the person who adopted this algorithm for shortest paths, all pair shortest paths, so jointly it is called the Floyd-Warshall algorithm. So, let us go back to this graph, which we have seen before. So, this is a graph with directions and with negative vertices. So, I have these negative edges. So, I have some negative edges here and there.

So, now, SP^0 is the adjacency matrix of this graph, is $A[i,j]$ where I replace every entry the weight. So, if I look at 0 to 10, 0 to 1, it has weighed 10. For instance, if I look at 4 to 5 for instance, it has weight -1, and so on. So, this is just the adjacency matrix for this graph, where every entry $[i,j]$ represents the weight and if there is no edge, then it is infinity. So, we have a lot of infinity entries, we have very few entries, which are not infinity.

(Refer Slide Time: 15:20)



So, now what is SP^1 , so SP^1 represents all things that you can reach going through the set 0. But look at 0, see 0 has no incoming edges, I cannot go to 0 and then go from 0 anywhere. Because 0 is such that you can only go out from 0, so I cannot go from i to 0 and then 0 to anywhere. So, intermediate, 0 as an intermediate vertex is not useful in this graph, there is nothing I can go to via 0. So, therefore, SP^1 is the same as SP^0 , that is if I am allowed to go through 0, again, nothing. If I am allowed to go through 0, again, nothing.

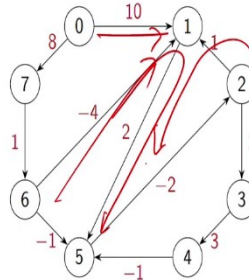
So, this is a certain difference between this and the transitive closure thing. So, there are paths of length 1, and then there are paths of length 2, but the paths of length 2 cannot go through 0, because if I go through paths of length 2 through 0, I have to enter 0, and I cannot enter 0, because the directions are all pointing out of 0. So, in this case, the first iteration produces no change.

(Refer Slide Time: 16:19)

Floyd-Warshall Algorithm



SP^1	0	1	2	3	4	5	6	7
0	∞	10	∞	∞	∞	∞	∞	8
1	∞	∞	∞	∞	∞	2	∞	∞
2	∞	1	∞	1	∞	∞	∞	∞
3	∞	∞	∞	∞	3	∞	∞	∞
4	∞	∞	∞	∞	∞	-1	∞	∞
5	∞	∞	-2	∞	∞	∞	∞	∞
6	∞	-4	∞	∞	∞	-1	∞	∞
7	∞	∞	∞	∞	∞	∞	1	∞



$\{0,1\}$

Navigation icons

Madhavan Mukund

All-Pairs Shortest Paths

Mathematics for Data Science

W

Now, this was my first iteration. So, this is SP^1 , which I just calculated is the same as SP^0 . Now I want to calculate SP^2 , SP^2 tells me I can go through 0 and 1. So, now I can go through 1. So, there are edges which come into 1, so where can I go by going into 1 and then out of 1 because I cannot go back to 0 anyway. So, I can go, for instance, from 6 to 5 via 1, or I can go from say 2 to 5 via 1. So, these are two of the things that I should get new.

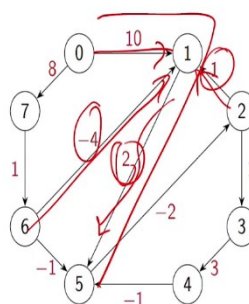
(Refer Slide Time: 16:51)

Floyd-Warshall Algorithm



SP^1	0	1	2	3	4	5	6	7
0	∞	10	∞	∞	∞	∞	∞	8
1	∞	∞	∞	∞	∞	2	∞	∞
2	∞	1	∞	1	∞	∞	∞	∞
3	∞	∞	∞	∞	3	∞	∞	∞
4	∞	∞	∞	∞	∞	-1	∞	∞
5	∞	∞	-2	∞	∞	∞	∞	∞
6	∞	-4	∞	∞	∞	-1	∞	∞
7	∞	∞	∞	∞	∞	∞	1	∞

SP^2	0	1	2	3	4	5	6	7
0	∞	10	∞	∞	∞	12	∞	8
1	∞	∞	∞	∞	∞	2	∞	∞
2	∞	1	∞	1	∞	3	∞	∞
3	∞	∞	∞	∞	3	∞	∞	∞
4	∞	∞	∞	∞	∞	-1	∞	∞
5	∞	∞	-2	∞	∞	∞	∞	∞
6	∞	-4	∞	∞	∞	-2	∞	∞
7	∞	∞	∞	∞	∞	∞	1	∞



Navigation icons

Madhavan Mukund

All-Pairs Shortest Paths

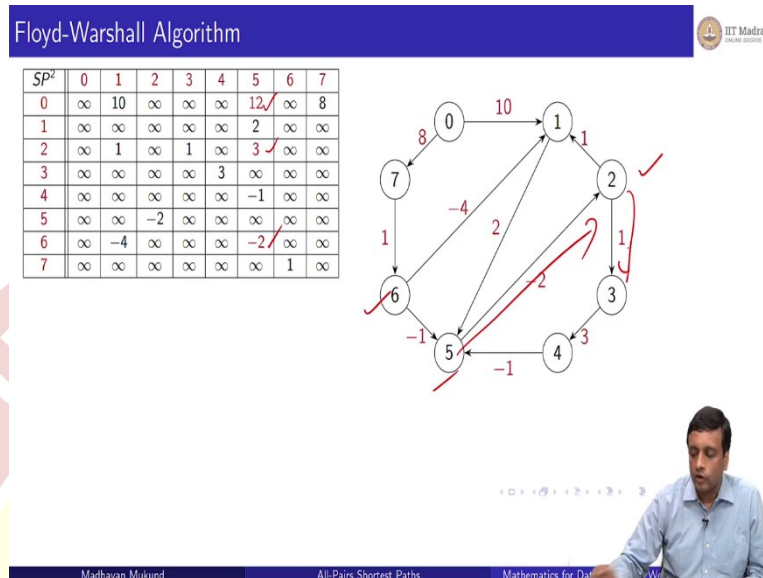
Mathematics for Data Science

W

And indeed, when you see that, you will see that I get from, so the 3 incoming edges are 0, 6, and 2, these are the three incoming edges to 1. So, these give me new possibilities of going from 0 to somewhere from 2 somewhere and 6 to somewhere, but the only thing I can reach from 1 is 5. So, the all the updates happen in column 5 because that is the target. So, from 0 to 5, I now find that there is a shortest path, which is $10 + 2$, so I go 10 and then 2. Similarly, from

2 to 5, I find the shortest path, which is $1 + 2$ is 3. And similarly, from 6 to 5, I find the shortest path, which is $-4 + 2$ is -2. So, this is my first update.

(Refer Slide Time: 17:37)



Now, let us do one more update. So, I have now shortest path. So, these were the updates, which I got when I did the first update, and now I can now I have discovered that I can reach 2, I can reach 6, and I can reach 5, no, the new thing I have discovered is I can reach 5. So, now I obviously want all I can reach from 5. So, from 5, for example, I can go back to 2, but from 2, I can go to 3 and so on.

(Refer Slide Time: 18:04)

Floyd-Warshall Algorithm

SP^2	0	1	2	3	4	5	6	7
0	∞	10	∞	∞	∞	12	∞	8
1	∞	∞	∞	∞	∞	2	∞	∞
2	∞	1	∞	1	∞	3	∞	∞
3	∞	∞	∞	∞	3	∞	∞	∞
4	∞	∞	∞	∞	∞	-1	∞	∞
5	∞	∞	-2	∞	∞	∞	∞	∞
6	∞	-4	∞	∞	∞	-2	∞	∞
7	∞	∞	∞	∞	∞	∞	1	∞

SP^3	0	1	2	3	4	5	6	7
0	∞	10	∞	∞	∞	12	∞	8
1	∞	∞	∞	∞	∞	2	∞	∞
2	∞	1	∞	1	∞	3	∞	∞
3	∞	∞	∞	∞	3	∞	∞	∞
4	∞	∞	∞	∞	∞	-1	∞	∞
5	∞	-1	-2	-1	∞	1	∞	∞
6	∞	-4	∞	∞	∞	-2	∞	∞
7	∞	∞	∞	∞	∞	∞	1	∞

B^8 $\{0, -7\}$

So, in the next round, I get a lot of updates, so I get updates from 5 to 1 because now I can come to 5. And then I can go to 1 using -1. And so why is that the case because I can go around this way. So, I can go up and go this way. And this gives me - two + 1, which is not the one I want, I think, 5 to 1, - 2 + 1 is - 1, that is the 1.

So, in this way, I can now update everything one more time. And I can keep doing this. And if I do this, now there are 8 vertices in this. So, if I do this up to B to the power 8, then I will have all paths which go to 0 to 7. And you can work it out, I am not going to work it out, you will get a matrix which gives you all the shortest paths from everywhere to everywhere. So, this is the Floyd-Warshall algorithm.

(Refer Slide Time: 18:59)

Summary

- Warshall's algorithm is an alternative way to compute transitive closure
 - $B^k[i, j] = 1$ if we can reach j from i using vertices in $\{1, 2, \dots, k-1\}$
- Adapt Warshall's algorithm to compute all pairs shortest paths
 - $SP^k[i, j]$ is the length of the shortest path from i to j using vertices in $\{1, 2, \dots, k-1\}$
 - $SP^n[i, j]$ is the length of the overall shortest path
 - Floyd-Warshall algorithm
- Works with negative edge weights, assuming no negative cycles

So, to summarize, we started with Warshall's algorithm. So, Warshall's algorithm is an alternative way to compute transitive closure. So, it is an iterative transitive closure algorithm. Earlier, we did it in terms of lengths of paths and now we have done it in terms of which intermediate vertices are I am allowed to visit while traveling from i to j . So, that is the difference between Warshall's algorithm and the way we had formulated in terms of paths of length 1, 2, 3, and so on.

So, $B^k[i,j]$ is 1, if I can only reach i to j using only vertices 0 to $k - 1$. So, when we adapt Warshall's algorithm for shortest paths, what we do is we replace that calculation of OR, I can either reach it without using k or I can use it reach it with using k , I replace that OR by minimum, what is the minimum I can do without using k , what is the minimum I can do with using k and I use that to update this thing from i to j . So, this is called the Floyd-Warshall algorithm and it should work as we saw with negative weights provided there are no negative cycles.

