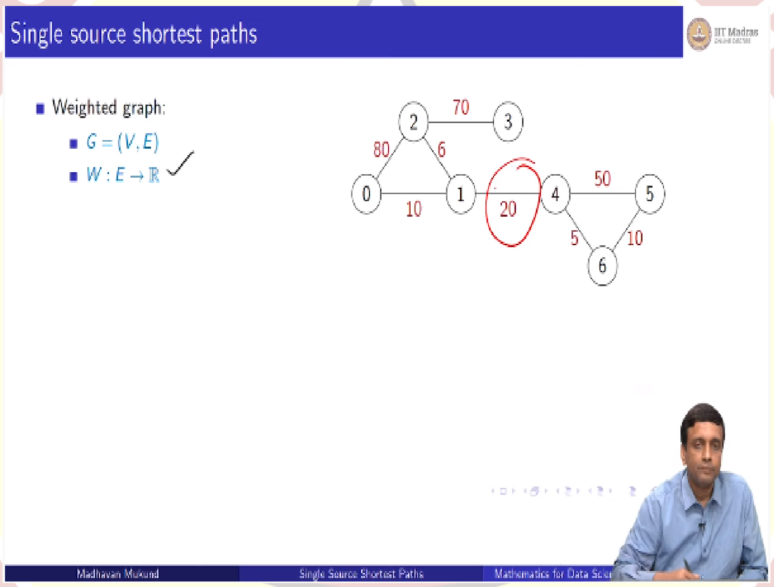
INDIAN INSTITUTE OF TECHNOLOGY MADRAS
सिद्धिर्भवति कर्मजा

IIT Madras
ONLINE DEGREE

Mathematics for Data Science 1
Professor. Madhavan Mukund
Chennai Mathematical Institute
Lecture No. 12.2
Single Source Shortest Paths

So, we are looking at weighted graphs, and we are looking at shortest path problems on weighted graphs and we said that there are two types of shortest path problem that we will focus on, the single source shortest path and the all pair shortest path. So, let us look at how we can compute single source shortest paths in weighted graphs.

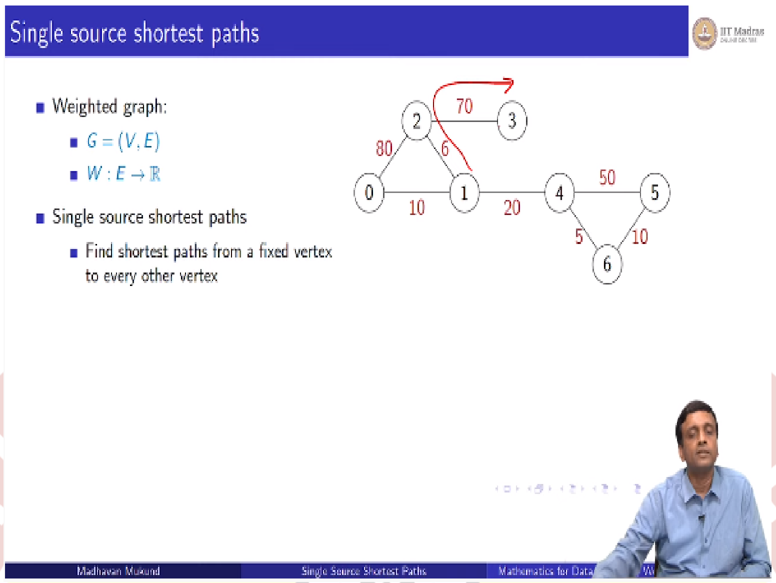
(Refer Slide Time: 0:32)


Single source shortest paths

- Weighted graph:
  - $G = (V, E)$
  - $W : E \to \mathbb{R}$ ✓

So, remember that a weighted graph is just a graph, which has in addition, this weight function, this weight function assigns to every edge some real number. So, here on the right we have a graph where at every edge we have a number written against it, saying, for example that the weight of the edge from 1 to 4 for instance, is 20.

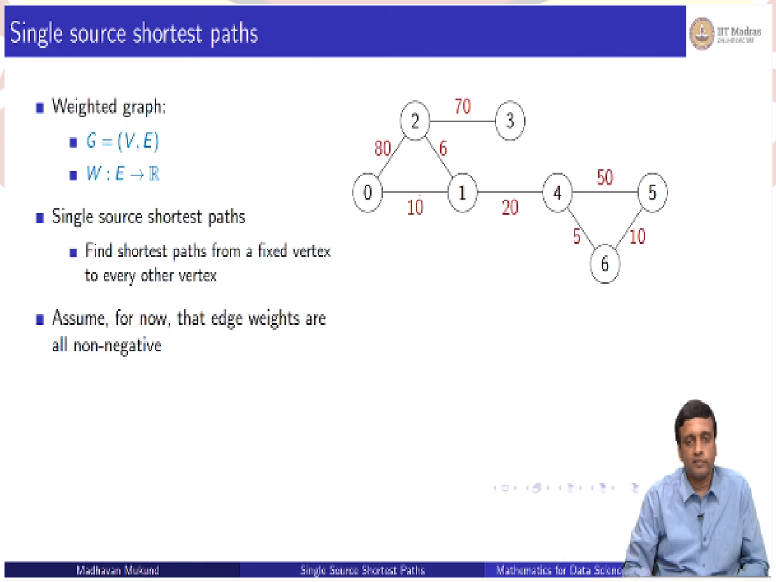
(Refer Slide Time: 0:53)


Single source shortest paths

- Weighted graph:
  - $G = (V, E)$
  - $W : E \to \mathbb{R}$
- Single source shortest paths
  - Find shortest paths from a fixed vertex to every other vertex

So, in the single source shortest path problem, we want to look at these edge weights, so now we are taking paths as representing the edges underneath as representing the weights underneath, so if I take a part, for instance, if I take this path, then the total length of this path the 6 plus 70, it is not two the edges but the weights attached to it that I have to add up. So, we want to find some source vertex, it could be any vertex, I start from there and I want to find the shortest path to every other vertex in the graph.

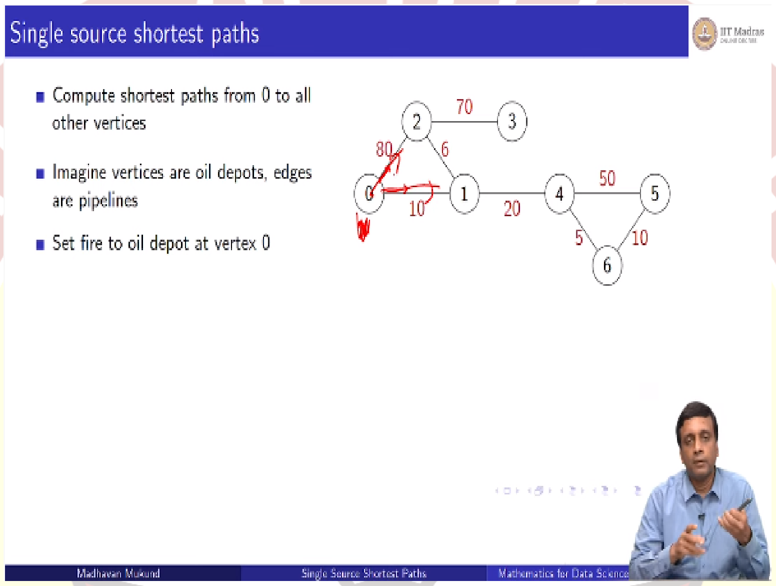
(Refer Slide Time: 1:23)


Single source shortest paths

- Weighted graph:
  - $G = (V, E)$
  - $W : E \to \mathbb{R}$
- Single source shortest paths
  - Find shortest paths from a fixed vertex to every other vertex
- Assume, for now, that edge weights are all non-negative

So, in order to solve this we will first assume that edge weights are all non negative, that is we could have 0 weights, but we certainly do not have negative weights, otherwise algorithms that we are going to look at will not work and we will also explain why it will not work. So, for now we are looking at graphs like the one here, where all the edge weights are either 0 or positive, not negative.

(Refer Slide Time: 1:48)



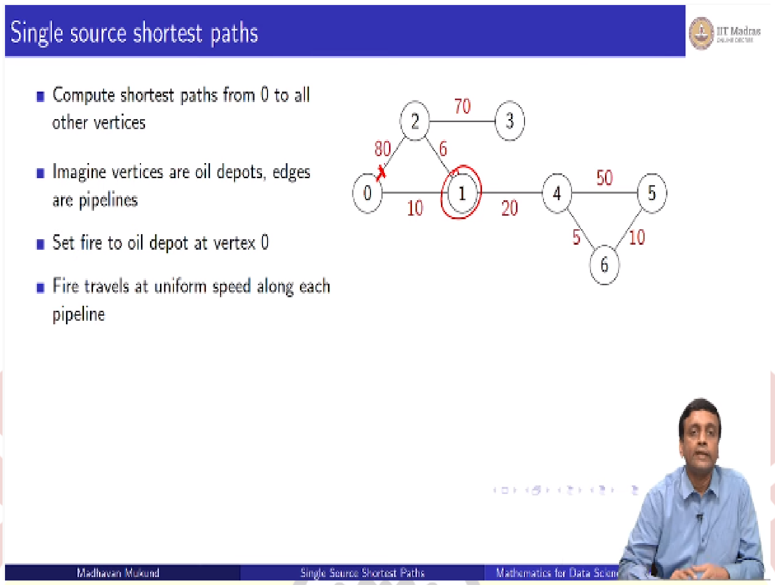
Single source shortest paths

- Compute shortest paths from 0 to all other vertices
- Imagine vertices are oil depots, edges are pipelines
- Set fire to oil depot at vertex 0

So, let us say that our single source in this particular example is 0. So, we want to find the shortest path from 0 to every other vertex in this graph. So, how would we do this? So, one way to imagine this or to visualize how the algorithm works, is to imagine that these are all some kind of pipelines carrying oil and all these nodes are actually oil depots which are storage tanks full of oil.

So, now imagine what happens if we set fire to this thing, so we set fire here. So, what will happen is that the fire will start traveling away from 0 along the pipeline because the pipeline also has oil, so you burn the tank and the pipeline burns. But of course, if you have ever burned firecrackers you know that it takes time for the fire to go along the thread, the wick. So, it takes time for the fire to travel, but it travels in all directions at the same speed. So, the fire, fire is traveling in this direction towards 1, it is also traveling in this direction towards 2, but the length of the edge determines which one will burn next.

(Refer Slide Time: 2:59)

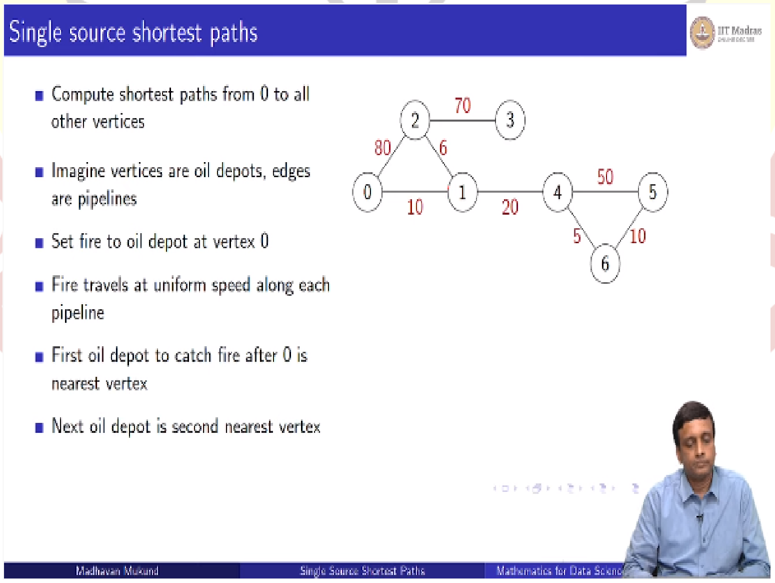


Single source shortest paths

- Compute shortest paths from 0 to all other vertices
- Imagine vertices are oil depots, edges are pipelines
- Set fire to oil depot at vertex 0
- Fire travels at uniform speed along each pipeline

So now, if the fire is traveling at a uniform speed, then after a certain amount of time, right, after some amount of time this one will burn. At that point, because this is 10 and that is 80 the fire would have only reached up to about here. About one eighth of the way.

(Refer Slide Time: 3:16)

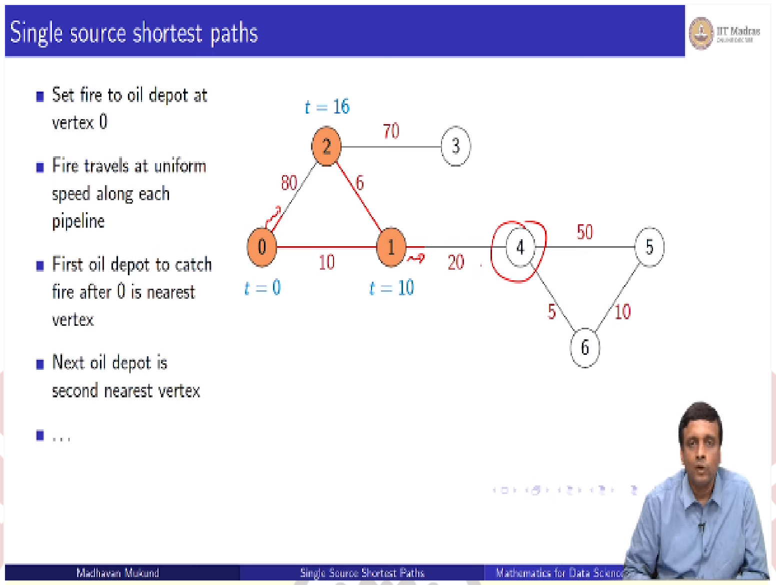


Single source shortest paths

- Compute shortest paths from 0 to all other vertices
- Imagine vertices are oil depots, edges are pipelines
- Set fire to oil depot at vertex 0
- Fire travels at uniform speed along each pipeline
- First oil depot to catch fire after 0 is nearest vertex
- Next oil depot is second nearest vertex

And now, once the next vertex burns, it will start fires in these two directions, so now I have a fire running in that direction, this thing is fully burned and so on. So, this is the intuitive idea that we want to capture. So, let us see how it works.

(Refer Slide Time: 3:34)

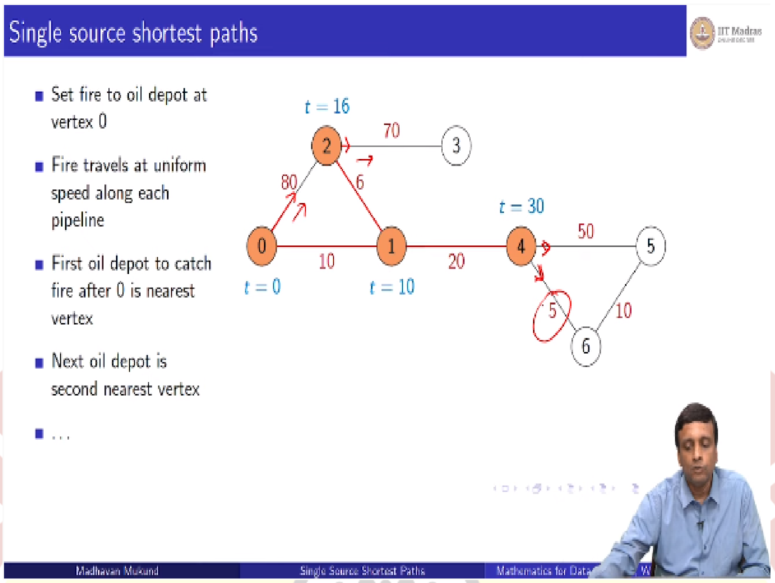

Single source shortest paths

- Set fire to oil depot at vertex 0
- Fire travels at uniform speed along each pipeline
- First oil depot to catch fire after 0 is nearest vertex
- Next oil depot is second nearest vertex
- ...

So, we start with this thing, saying that initially when time is 0, we burn the zeroth vertex and now the fire we said starts moving in these two directions. So, if we imagine the fire burning then after some, let us assume that fire moves at one unit of this, one unit length per one unit time. So, here it has to move 10 units of length, so in 10 units of time it reaches vertex 1. Now in 10 units of time as we have indicated here, it has traveled some short distance along the edge from 0 to 2 as well, but it is not yet anywhere near 2.

Now that this is burned, now something else is going to happen because I am going to end up having fire going in these two directions. So, after 6 units of time from this, vertex 2 is going to catch fire. Notice that this fire which started from 1 is still growing but it is nowhere near 2 and meanwhile, this thing has also started traveling towards 4. So, what is going to happen next is that this fire is going to reach this at time 10 plus 20, remember it is assuming the fire is moving at 1 unit distance per unit time.

(Refer Slide Time: 4:48)



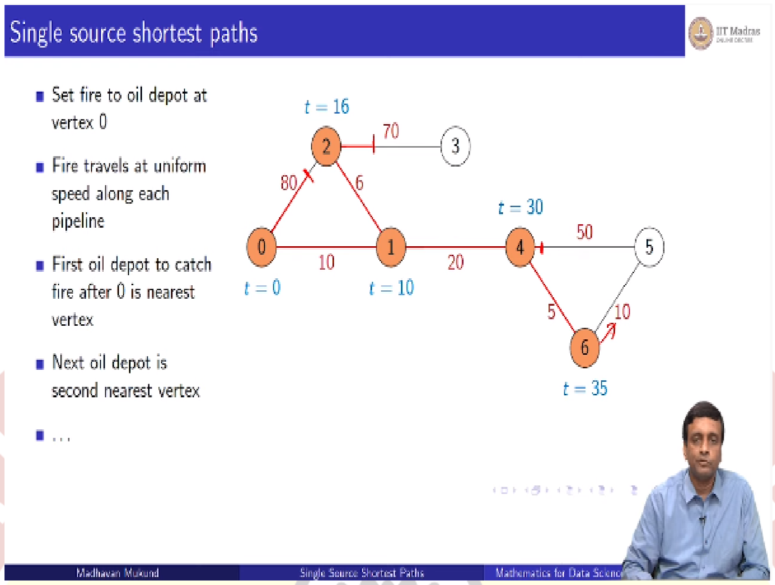
Single source shortest paths

- Set fire to oil depot at vertex 0
- Fire travels at uniform speed along each pipeline
- First oil depot to catch fire after 0 is nearest vertex
- Next oil depot is second nearest vertex
- ...

So, at t equal to 30 vertex 4 catches fire and meanwhile, this fire which is at 2 has started moving there and this fire now is kind of reached a little bit less than halfway from 0 to 2 because it has (())(5:02) units so we have already spent 30 times units so far, so suppose to be about three eighths of the way from 0 to 2.

And now what happens here is that this fire now starts going in these two directions, so this fire is going here, this fire is going here, these 3 edges have all burned and now two new fire start, so which one is going to reach the next one? Well, this is only 5 units of time, so in 5 more units of time the fire from 4 will reach 6.

(Refer Slide Time: 5:30)



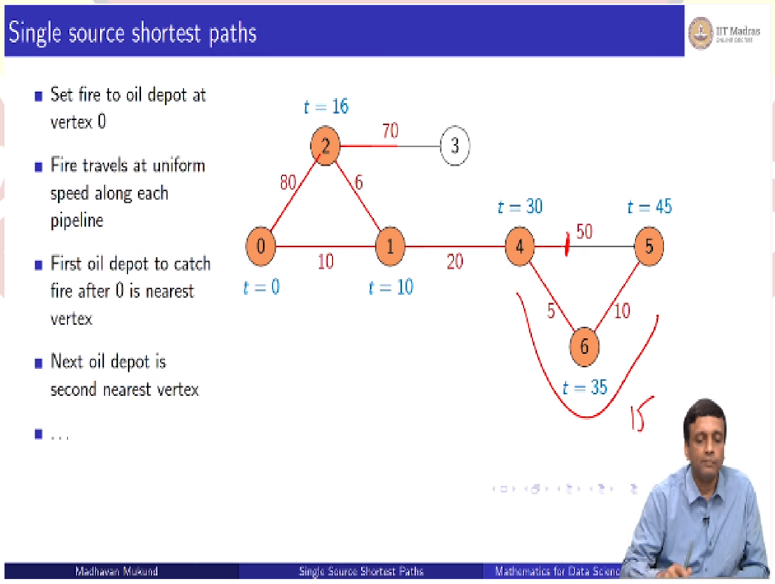
Single source shortest paths

- Set fire to oil depot at vertex 0
- Fire travels at uniform speed along each pipeline
- First oil depot to catch fire after 0 is nearest vertex
- Next oil depot is second nearest vertex
- ...

Madhavan Mukund Single Source Shortest Paths Mathematics for Data Science

So, at t equal to 35 vertex 6 is going to burn. Meanwhile, this thing is started, so in 5 units of time this fire has reached a little distance, this has made more progress, this has made more progress and then from here, a separate fire starts going towards vertex 5 in addition to the old fire which is coming from 4.

(Refer Slide Time: 5:52)



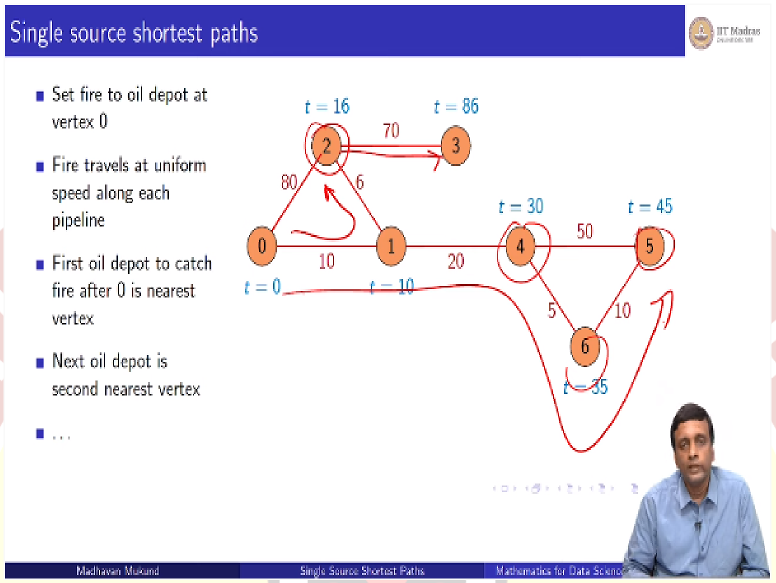
Single source shortest paths

- Set fire to oil depot at vertex 0
- Fire travels at uniform speed along each pipeline
- First oil depot to catch fire after 0 is nearest vertex
- Next oil depot is second nearest vertex
- ...

Madhavan Mukund Single Source Shortest Paths Mathematics for Data Science

So, now the separate fire is going to reach in 10 units of time so at t equal to 45 our vertex 5 is going to burn at this point, we have gone down 15 units of time have passed since 4 was burned, so this fire has only moved 15 by 50 of the way, about one third of the way there and meanwhile, these things are still going.

(Refer Slide Time: 6:10)



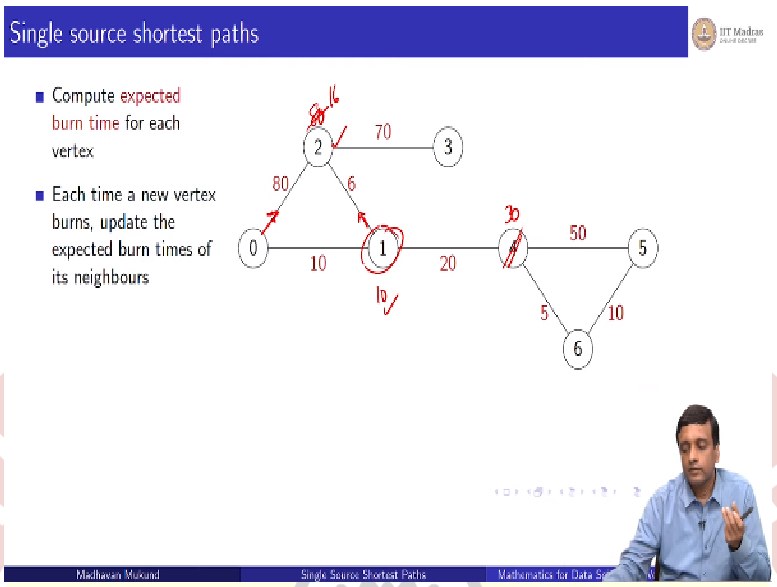
Single source shortest paths

- Set fire to oil depot at vertex 0
- Fire travels at uniform speed along each pipeline
- First oil depot to catch fire after 0 is nearest vertex
- Next oil depot is second nearest vertex
- ...

And then finally, what happens is that at time 86 this fire which started from here reaches here, at this point everything has burnt, so all our vertices have burned but all the edges have also burned it so happens, it does not have to happen always, but now we have discovered the earliest time at which every vertex burns and now we can check that these earliest times are actually the shortest paths.

So, we have discovered some interesting shortest paths for instance, we have discovered this shortest path, we have also discovered this shortest path, so the shortest path to 5 is not 0, 1, 4, 5 but 0, 1, 4, 6, 5. So, this is how this algorithm works. So, let us try to understand how we would actually do this calculation that we said by drawing these burning pipelines, so how do you keep track of when the pipeline is going to burn next.

(Refer Slide Time: 7:00)



Single source shortest paths

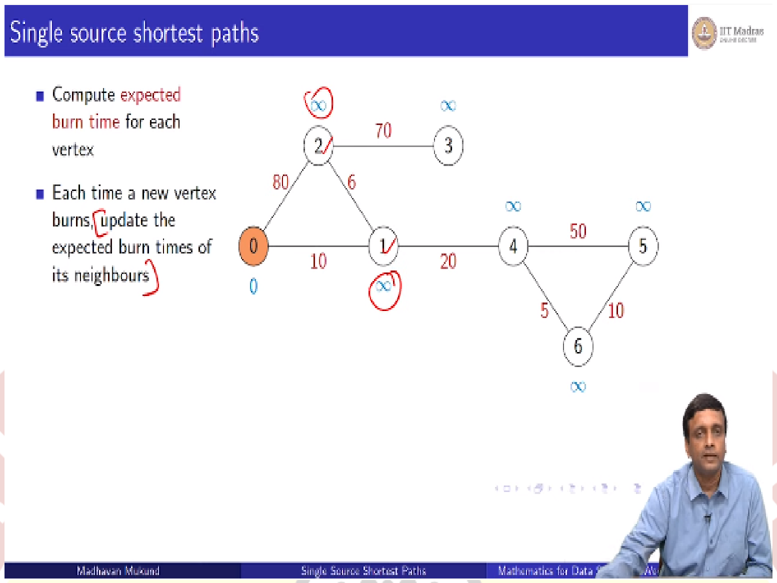
- Compute expected burn time for each vertex
- Each time a new vertex burns, update the expected burn times of its neighbours

So, what we do is we compute at every given point when each vertex is expected to burn. Based on the information that we have so far about which pipelines are burning and which oil depots have burned, we compute the expected time to burn, now among all the vertices, the one which is expected to burn next is going to burn next. So, at that point we burn it, so we update the vertex which is burned. And once a vertex burns, it starts. Remember, new fires along its neighbors towards, so its neighbors.

So, we have to check whether any of its neighbors will burn now faster because of this, so this is what we saw, so we said that when 0 burned, we started a fire here, so we could expect that this will burn at time 80. But after time 10 when this burn, a new fire starts so we said if this burns at 10, then this is actually not going to burn 80, this is going to burn at 16.

So, each time a vertex burns, we figure out something about the neighbors, this is going to burn at 30, at least cannot burn beyond 30 because I already know that a fire is started at time 10 and it is going to reach there in 20 units, similarly a fire has started here at 10 and it is going to reach here in 6 units. So, 2 cannot burn any later than 16, maybe it will burn even earlier but not later than 16. So, each time we burn a vertex, we update the burn time.

(Refer Slide Time: 8:19)


Single source shortest paths

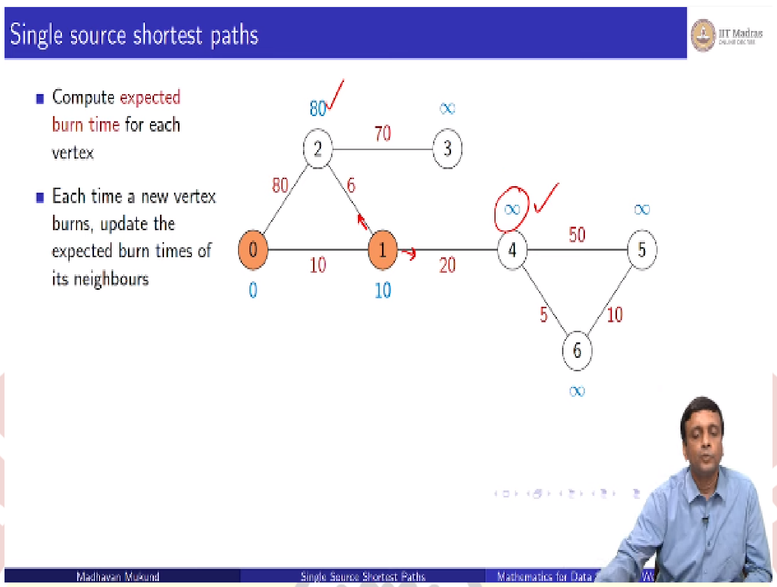
- Compute expected burn time for each vertex
- Each time a new vertex burns, update the expected burn times of its neighbours

So, let us see how this goes. So, initially we know nothing, so initially when we start this whole process every graph, every vertex is going to burn at some indefinite time in the future, so as far as we are concerned it is never going to burn, at time infinity. Maybe the fire is never going to reach there, we have no idea. Now we said, we will start at a single source, so for the single source we know that the burn time is 0, because that is when we start ticking, the clock starts ticking at 0 so, at time 0 we burn our source vertex which in our case is vertex 0.

Now, this is where we start doing this, so we update the burn time of the neighbors. So, once we have burned 0 at times 0, we have to look at its neighbors namely 2 and 1 and say, do I know something better, what do I know about 2? Well, as far as I know 2 is never going to burn but now I know that 2 is going to burn at least by time 80, after time 80 it is there is no chance that 2 is not burnt.

Similarly, 1 will burn at least by time 10, because 0 is burned and the fire has started moving in that direction. So, I update these two entries, these two entries can now be updated with better information I can reduce it to 80 and 10.

(Refer Slide Time: 9:24)

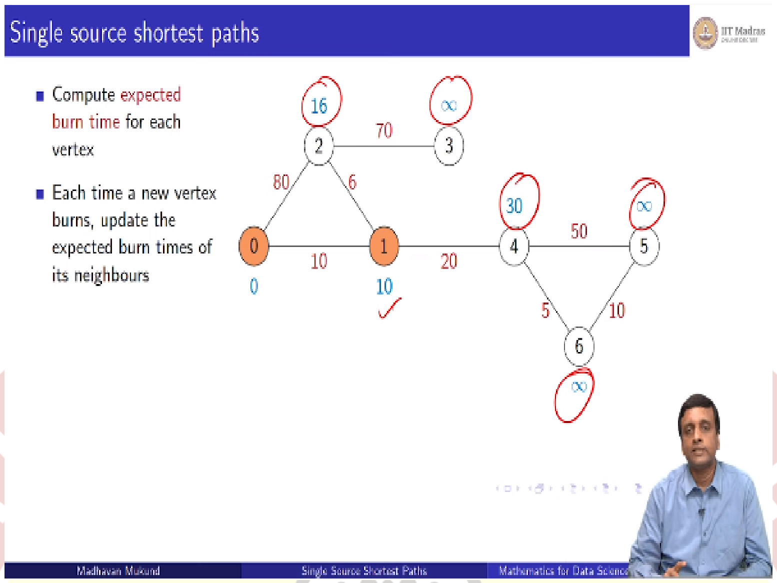

Single source shortest paths

- Compute expected burn time for each vertex
- Each time a new vertex burns, update the expected burn times of its neighbours

Now, I look at all the vertices that I know about and I see which one is going to burn next. So, I look for the smallest unburned vertex, I look for one with the smallest time to burn. So, the smallest time to burn at this point is 10. So, I go to 10 and I say okay you are going burn next. Now, the same thing happens when I burn 10, it starts two new fires if you remember, so it will start a fire back of course there is no fire going back because 0 is already burned but in the directions where there is no, where the vertices are not burnt I get new information.

I already believe that 2 will burn at time 80 but now I can tell that it cannot stay more than time 16. And similarly, 4 which was previously not known to burn at any time at all. Now I know it definitely burned by 30. So, I can update this entry as neighbors of 1, this entry and this entry to be now 16, and 30.

(Refer Slide Time: 10:19)

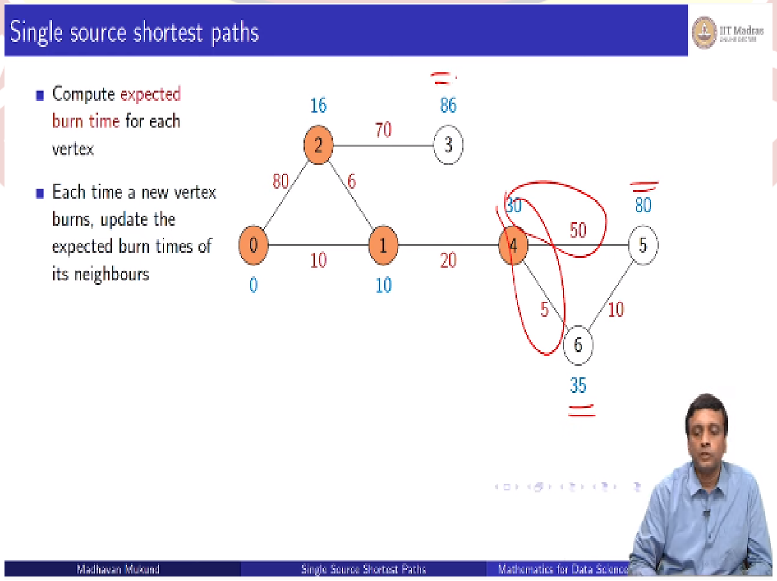


Single source shortest paths

- Compute expected burn time for each vertex
- Each time a new vertex burns, update the expected burn times of its neighbours

Now, again I look among the unburned vertices, so these are my unburned vertices, I look among my unburnt vertices for the one which has the minimum time to burn. Now this is the absolute time from time equal to 0. It is not 16 more units of time but at t equal to 16, which is 6 units of time from now, because right now I am at unit, I am at time 10, 10 is when this happens. So, in 6 more units something is going to happen and that is going to be 2.

(Refer Slide Time: 10:48)



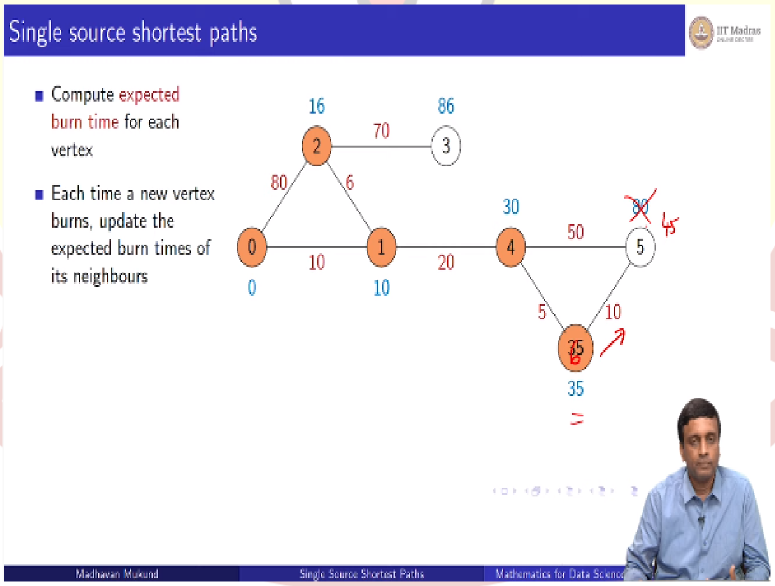
Single source shortest paths

- Compute expected burn time for each vertex
- Each time a new vertex burns, update the expected burn times of its neighbours

So, at times 16 2 burn, when 2 burn, I look at its neighbors, which are not burn, namely 3 and I say 3 which I never knew was going to burn at all, now I know it is going to burn at 86. But I get no new information on this side because 2 is not connected to anybody else, so I have no further information yet about 5 and 6. As far as I know, 5 and 6 are never going to burn.

Now, again I look at those vertices which are not burned and I look for the minimum. And I find that vertex 4 is going to burn at time 30. So, I burn it, so now time is 30. So, now I have information about 5 and 6 because once I have burned 4 I have started these two new fires, which are going here. So, this tells me that vertex 6 is going to burn at time 30 plus 5 and vertex 5 is definitely going to burn by 30 plus 50, so I can update those to 80 and 35. So, this is 30 plus 50 is 80 and this is 30 plus 5 is 35. So, now again I have the same thing which are the ones who did not burn, 35, 80 and 86. So, which is the next one to burn clearly vertex 6.

(Refer Slide Time: 11:52)



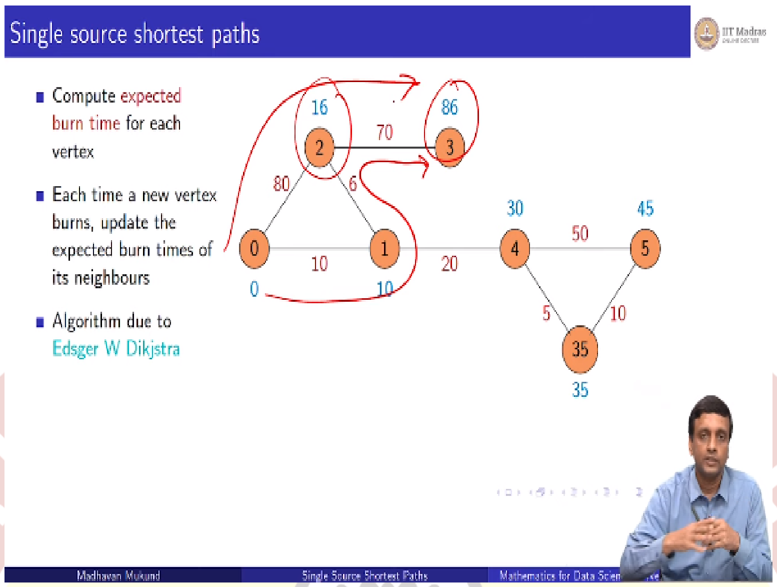
Single source shortest paths

- Compute expected burn time for each vertex
- Each time a new vertex burns, update the expected burn times of its neighbours

Madhavan Mukund | Single Source Shortest Paths | Mathematics for Data Science

So, I burned vertex 6, now having burn vertex 6 again, I have, this should been a 6 here, I do not know how it became 35, so this 6 now passes a new fire here. So, since this was a 35, I know that this is going to burn at 45, my previous information was 80, so I have to remove the 80 and make it 45, with each time I improve my estimate if I can based on new information that I get.

(Refer Slide Time: 12:19)



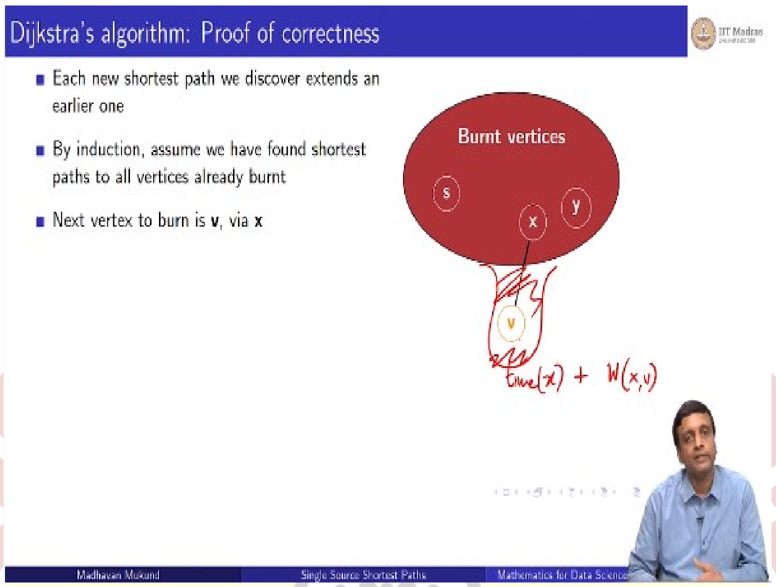
Single source shortest paths

- Compute expected burn time for each vertex
- Each time a new vertex burns, update the expected burn times of its neighbours
- Algorithm due to Edsger W Dikjstra

So, now I improve that 80 to 45 and having improved that 80 to 45. Now I burn it because among the things which are not burned is 60, 86 and 45. So, when I burn 5, I get no new information about 3 or anybody else, because it is not connected. So, finally, 86, 3 burns, so this is my algorithm, this algorithm although we have described it picturesquely in terms of burning vertices and all that, you can keep track of all this information in a matrix and keep track of what is burned we can keep updating this time to burn and so on by looking at the edges in my graph and finding out what are the neighbors, what is the current burning time and what is the expected new burning time and so on.

So, it is a fairly straightforward thing to do if you have the matrix and if you have this information about these burning times, we are not going to describe this algorithm in precise detail now, later on we will study it in the course on algorithms. So, this algorithm is very well known it is due to a very well known computer scientists called Edsger Dikjstra, so this is called Dikjstra's algorithm. This is the Dikjstra's algorithm for the single source shortest path.

(Refer Slide Time: 13:25)



Dijkstra's algorithm: Proof of correctness

- Each new shortest path we discover extends an earlier one
- By induction, assume we have found shortest paths to all vertices already burnt
- Next vertex to burn is **v**, via **x**

So, why does this work, I mean why is this kind of update reasonable? So, the idea is that every time the new shortest path that we find extends an earlier shortest path, right, so if we look back, we go back to this. So, the shortest path to 86 is an extension of the shortest path to 2. So, basically the shortest path to 2 is this way. So, I get the shortest pathway here, I cannot get a shortest path to 86 which goes the other way by a longer path, so every prefix, every beginning segment of a shortest path is also a shortest path.

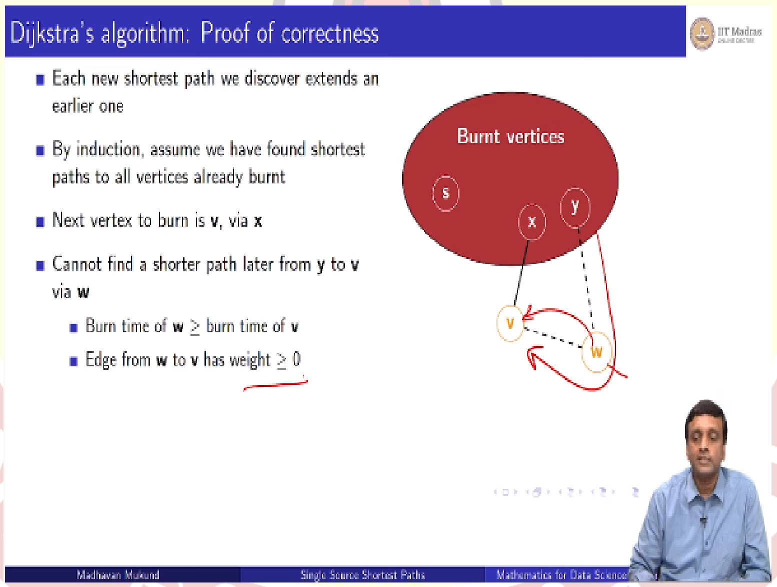
It cannot be that I find the shortest path to a vertex and then to go to its neighbor I find the shortest path with bypasses I mean find the longer path to this vertex and goes. So, every shortest path, actually extends, an earlier shortest path. So, what we are going to show as a kind of correctness proof is to assume that at every point when we have burned some vertices the numbers we have associated, the burning times we have associated are the shortest time, so by induction this is correct, because the starting point is a source vertex which we label with 0.

So, when we label the starting point with time 0, obviously that is the shortest time at which it burns because that is the shortest path from a vertex to itself is length 0. So now, at any given point we have some vertices which are burned and we have some vertices which are yet to burn and with each of them we have an estimate. So, we have some vertices in this case s, x and y, which are burnt and now suppose our next vertex to burn with the shortest burning time is v and it is connected to x. So, what we are assuming therefore is that the shortest path to x now extends by x to v to shortest path to v.

So, basically the time to v is going to be the time to x plus the weight of the edge xv, I mean that is basically what we are saying, there is no faster way to get to v than to first get to x, which is the time it takes for x to burn plus the cost of going from x to v. So, now the question is, is this going to be the final value for v, can we discover, because now we are going to burn it, so this is going to get burned. So, this burn thing is going to include and by our induction assumption once we burn a vertex, its value is correct.

So, is this value correct, that is what we want to know. So, if we choose to burn it now, are we being hasty, are we going to discover a better path to V later on, can it happen?

(Refer Slide Time: 16:03)



Dijkstra's algorithm: Proof of correctness

- Each new shortest path we discover extends an earlier one
- By induction, assume we have found shortest paths to all vertices already burnt
- Next vertex to burn is **v**, via **x**
- Cannot find a shorter path later from **y** to **v** via **w**
  - Burn time of **w** $\geq$ burn time of **v**
  - Edge from **w** to **v** has weight ≥ 0

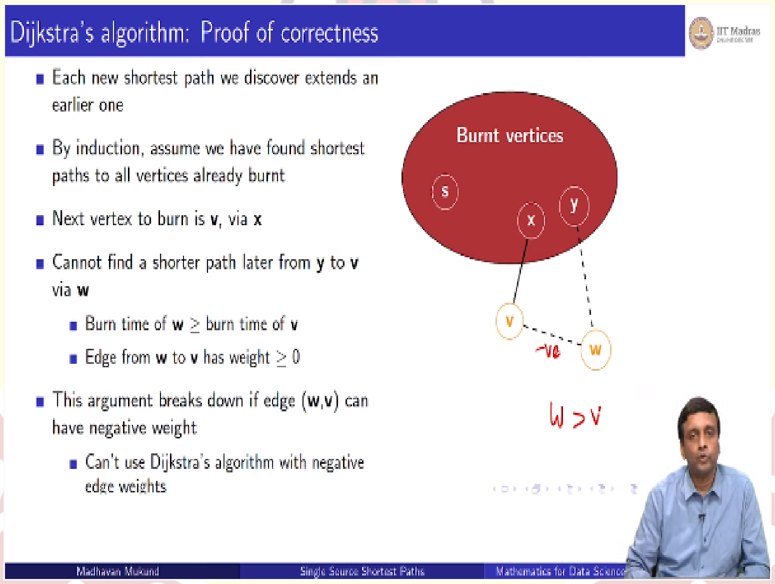
Madhavan Mukund Single Source Shortest Paths Mathematics for Data Science

So, supposing there is a better path later on so where will that better pat come? It will come from some other vertex which is not yet burnt because among the burnt vertices we already found the best path. So, at a later stage supposing we find a path from y to v via some w, we burn a new vertex w and through that new burning vertex we somehow magically discovered that there is a shorter path to v.

Now, can this happen? So, first of all, why did we burn w after v? We burned w after v because w has a burning time which is not smaller than v, it could be exactly equal but it is greater than or equal to it. So, the time at this point is the time is already equal to or greater than v and then I have to come from here. So, I have to spend some time, because there is some cost involve with that edge, it could be 0. So, I certainly I could get the same value, maybe by some magic, the burning time of w is exactly equal to the burning time of v, it cannot be smaller because otherwise we burned w before v. So, it is at least as big, maybe it is not bigger but maybe equal to, but even if it is equal to, at best, then I can come in 0 cost from w to v because edges are not negative.

So, the edge from w to v has a weight which is bigger than equal to 0. So, this is crucial, so if we did not have this, then the decision to burn v based on its distance from x could have been premature.

(Refer Slide Time: 17:21)



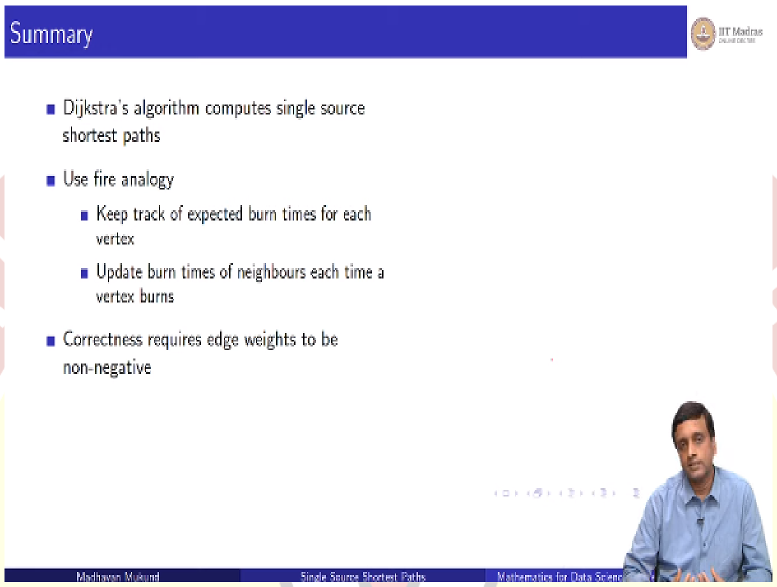
Dijkstra's algorithm: Proof of correctness

- Each new shortest path we discover extends an earlier one
- By induction, assume we have found shortest paths to all vertices already burnt
- Next vertex to burn is **v**, via **x**
- Cannot find a shorter path later from **y** to **v** via **w**
  - Burn time of **w** $\geq$ burn time of **v**
  - Edge from **w** to **v** has weight ≥ 0
- This argument breaks down if edge (**w**,**v**) can have negative weight
  - Can't use Dijkstra's algorithm with negative edge weights

So, this tells us that we cannot use Dikjstra's algorithm, if we have negative edge weights because this could happen, I might discover a long path after I burnt a vertex which becomes shorter, because there are some negative edge weights which cancel out the initial thing, so it could be that the, this is, so w is bigger than v, but because this is negative, the cost of coming back from w to v actually drops the cost of v below the cost that I had got when I burnt it.

So, for this reason we have to assume, when we run Dikjstra algorithm that the edge weights are bigger than equal to 0 otherwise this strategy for updating is not going to always work.

(Refer Slide Time: 18:00)



Summary

- Dijkstra's algorithm computes single source shortest paths
- Use fire analogy
  - Keep track of expected burn times for each vertex
  - Update burn times of neighbours each time a vertex burns
- Correctness requires edge weights to be non-negative

Madhavan Mukund | Single Source Shortest Paths | Mathematics for Data Science

So, to summarize, we have found an algorithm to compute single source shortest paths provided the edges have non negative weights. And the way to think about this algorithm is to use this fire analogy, so we set fire to every, we set fire to the initial vertex, think of it as an oil tank, and then think of the edges as pipelines. So, this oil, fire that starts at source vertex spreads at a uniform rate through these pipelines, and then we can calculate through this pipeline uniform burning the rate at which, the time at which every vertex burns, and when we can systematically keep track of this, you will get the shortest path to every vertex. So, this is Dikjstra's algorithm, which works single source shortest paths for weighted graphs with non negative edge weights.