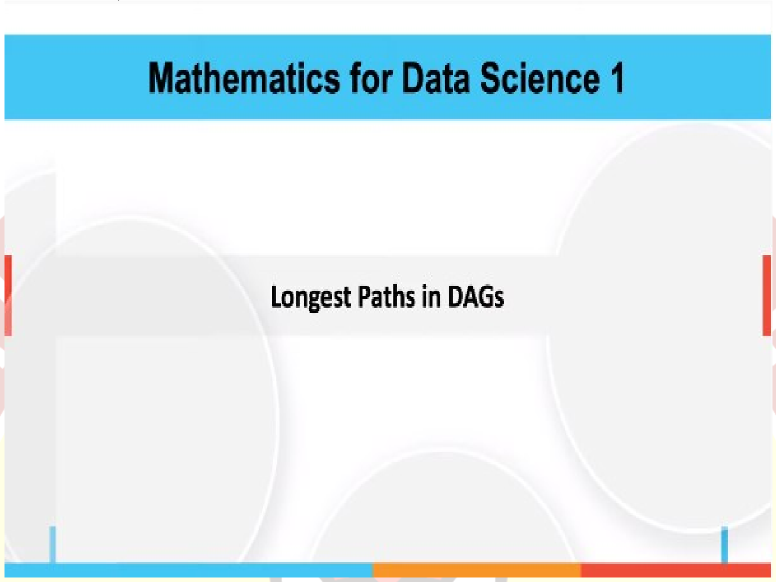
IIT Madras

ONLINE DEGREE

Mathematics for Data Science 1
Professor Madhavan Mukund
Indian Statistical of Technology, Madras
Lecture 69
Longest Paths in DAGs

(Refer Slide Time: 0:09)

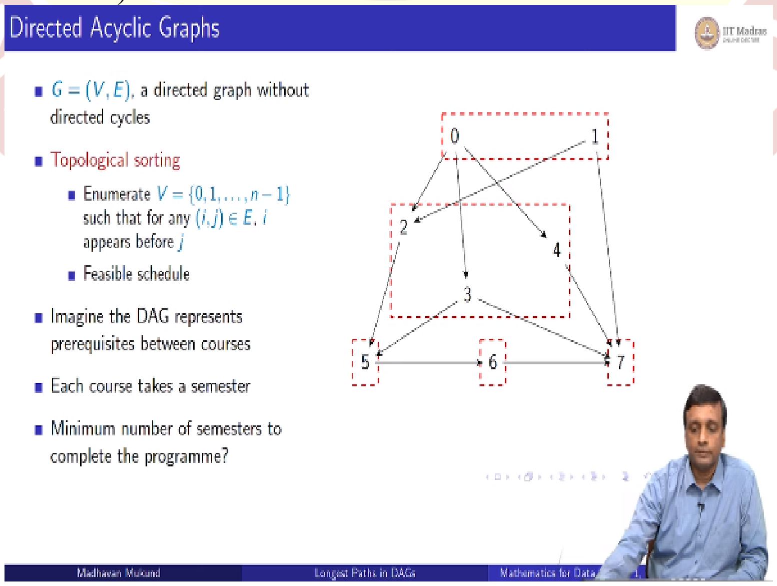


Mathematics for Data Science 1

Longest Paths in DAGs

So, we are looking at directed a cyclic graphs as representations of sets of tasks with dependencies. And we said that there are two natural problems on this. One, is to find a sequence of feasible sequence in which I can do the tasks and this was topological sort. The other one was to try and find out how many steps I need to perform these tasks in an optimal way.

(Refer Slide Time: 0:36)



Directed Acyclic Graphs

- $G = (V, E)$, a directed graph without directed cycles
- Topological sorting
  - Enumerate $V = \{0, 1, \dots, n-1\}$ such that for any $(i, j) \in E$, i appears before j
  - Feasible schedule
- Imagine the DAG represents prerequisites between courses
- Each course takes a semester
- Minimum number of semesters to complete the programme?

0
1
2
4
3
5
6
7
Madhavan Mukund
Longest Paths in DAGs
Mathematics for Data

So, this is my DAG. So, if I do a topological sort and I list out the vertices such that whenever there is an edge from i to j, i appears before j and this gives me a feasible schedule. Now, we may be interested in finding out how fast we can do this if tasks which have no dependencies can be done together.

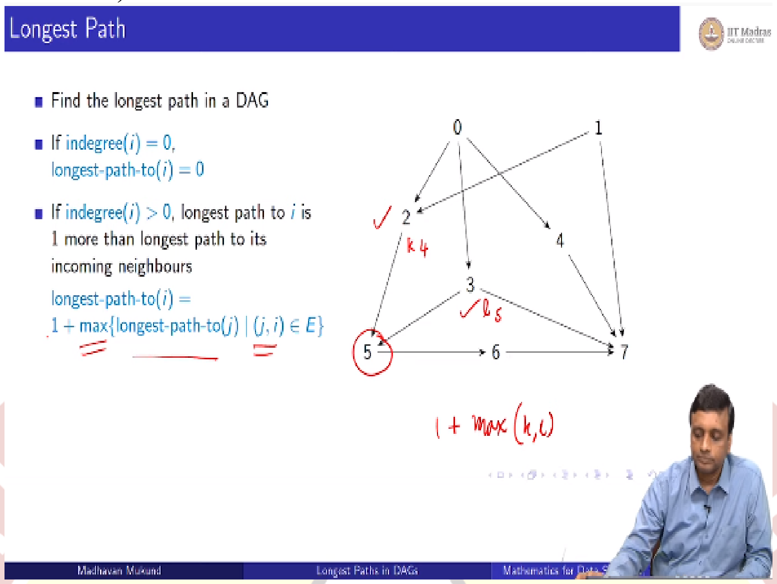
So, for instance, in a topological sort, we would have to put 0 before 1 or 1 before 0 as a sequence. But if we had no dependencies, and it is possible to do 0 and 1 parallelly, then we could do 0 and 1 at the same time, if there are resources to do both. So, a good example is when you are taking courses, so when you are taking courses, you do not do just one course in a semester, you can do many courses. And as long as the number of courses which are available for you to take are reasonable, you can do all of them, say three, or four or five, maybe in a semester.

So, if the DAG represents prerequisites between the courses, and each course takes a semester, so you can finish and go to the next course only in the next semester, then the natural question to ask is how many semesters do I need to complete the remaining requirements? So, I have a set of requirements, and they have some prerequisites between them, how many semesters do I need from now to finish the program satisfying these requirements. So, this is the problem that we want to solve now.

So, in this particular case, for instance, as I said, we can do 0 and 1 together. So this can be done in the first instance, then I can do 2, 3 and 4. In these are courses, then in the first semester, I can do 0 and 1 course, in the second semester, I can do courses 2, 3, and 4. But now I am stuck because I cannot do 7 until I finished 6, I cannot do 6 until I finished 5, 5 I can do because all the prerequisites are 5, namely 2 and 3 are done. So, in the third semester I can do 5, then in the fourth semester I can do 6 and finally in the fifth semester I can do 7.

So, if this was my sequence, I mean, this was a DAG representing my prerequisites, then I cannot do better than a sequence of 5 semesters. So this is the problem that we want to compute.

(Refer Slide Time: 2:41)



Longest Path

- Find the longest path in a DAG
- If indegree(i) = 0, longest-path-to(i) = 0
- If indegree(i) > 0, longest path to i is 1 more than longest path to its incoming neighbours

longest-path-to(i) = 1 + max{longest-path-to(j) | $(j, i) \in E$}

$1 + \max(k, l)$

Madhavan Mukund | Longest Paths in DAGs | Mathematics for Data S

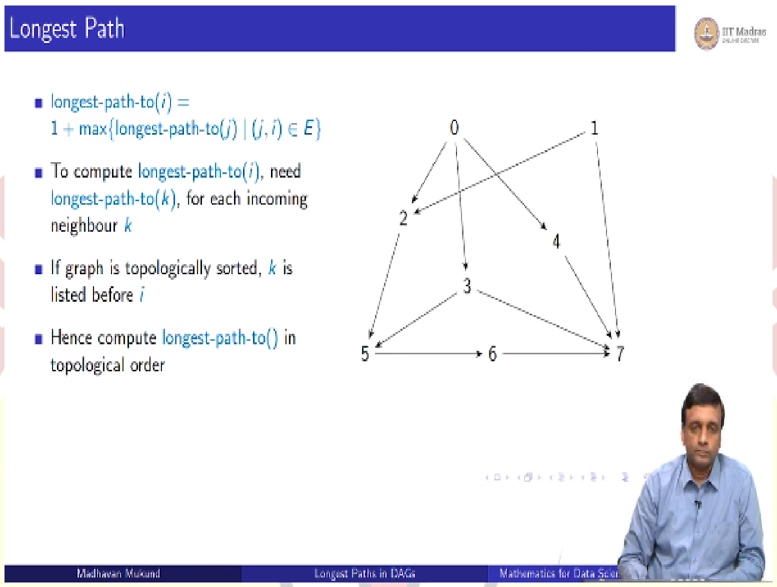
So formally, it consists of finding the longest path in a DAG. So, what we want to do is really compute the longest path to every vertex, and then if we have that, then the longest path among these will be the longest path overall, if I compute the longest path from one of the starting points, so when can I do course 6, when can I do course 3? If I know this, then among all these, if I have the maximum, some course requires me to do it in the fifth semester, then I know overall, I need five semesters. So, one way to solve the longest path problem is to solve this question of computing the longest path to each vertex.

So, with the assumption that the longest path to an initial vertex, so the path is how many semesters before in the course that example, how many semesters do I have to wait? So, with 0 waiting, I can do anything which is indegree 0, so that is a good starting point.

So, in this case, I could do these two things right to begin with, so this is indegree 0. Now what happens next. So, if I have indegree which is not 0, then supposing I look at this vertex, it can happen only after 2 and 3 have happened. So, if I know that the longest path to 2 is some k and the longest path to 3 is some l, then I must wait maximum of k and l, I cannot finish 2 and 3 until max of k and l happen.

So, supposing this is fourth semester, fourth semester, and this is going to happen in the fifth semester then, only in the sixth semester can I get to course number 5, so, this + 1, so, the longest path to i is going to be 1 + the longest path to every incoming neighbor of i. So, remember this set comprehension notation so, this is the set of all the numbers longest path j for j, i in the edge set, all the incoming edges coming into i, I take that, take the maximum of all of those and add 1 because I now have to go to the next semester.

(Refer Slide Time: 4:54)



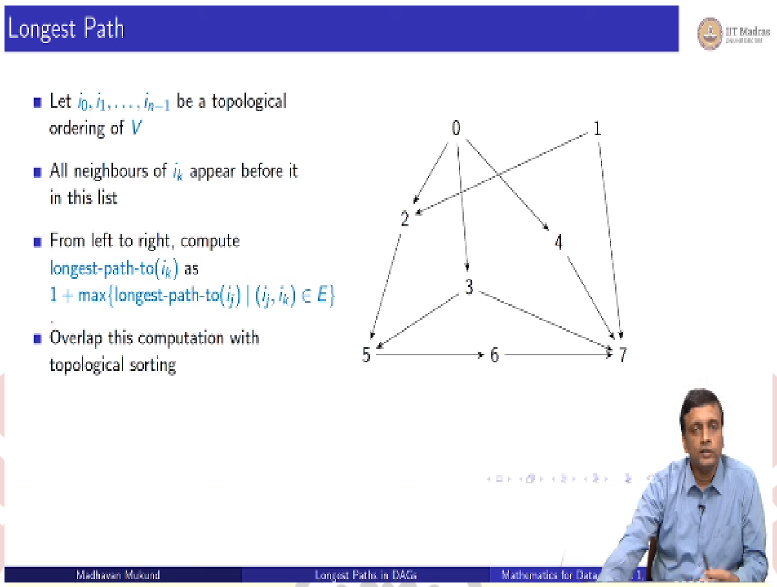
Longest Path

- longest-path-to(i) = $1 + \max\{$longest-path-to(j) $\mid (j,i) \in E\}$
- To compute longest-path-to(i), need longest-path-to(k), for each incoming neighbour k
- If graph is topologically sorted, k is listed before i
- Hence compute longest-path-to() in topological order

So, the longest pathway to i is 1 + the maximum of all the longest paths to the incoming neighbors. So, to compute this, I need to know the neighbors, longest paths for all the incoming neighbors. So, if I know that, then I can take the maximum of those and add 1 to it. But how do I know those? Well, I know those if I have already calculated those before this, and I would have calculated those before this if I calculate them following the topological sort.

So, if I sort out the sequences according to the dependencies, then by the time I come to i, and I want to compute the longest path to i, all the incoming neighbors of i should already have been listed before i. And if I am computing longest path as I listed, okay, then that information about all these incoming neighbors will be available to me when I come to i. So, that is the strategy that we are going to do, we are going to compute longest path in topological order. So, we are going to compute the longest path to every vertex as we compute topological sort, in fact.

(Refer Slide Time: 5:59)



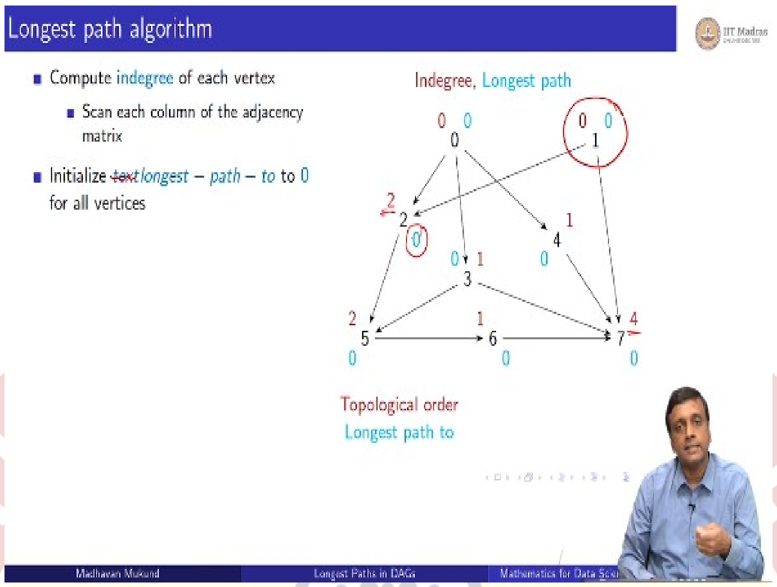
Longest Path

- Let $i_0, i_1, \ldots, i_{n-1}$ be a topological ordering of V
- All neighbours of i_k appear before it in this list
- From left to right, compute longest-path-to(i_k) as $1 + \max\{\text{longest-path-to}(i_j) \mid (i_j, i_k) \in E\}$
- Overlap this computation with topological sorting

So, more formally supposing this is a topological ordering of i, of V, right. All the neighbors of some vertex ik, so i0 to in - 1 is some reordering of the numbers 0 to n - 1. So, the vertices are 0 to n - 1 and I have rearranged them in some sequence, and that sequence is i0 to in - 1. If I look at a particular entry, the kth entry in this was a k + 1 entry i_k, then all its neighbors in the graph must appear before it.

So, if I compute from left to right, then I can compute for each of these ik, the longest path based on the values that I have already computed to the left. And since I am doing this as I am going along, I do not have to actually enumerate all the vertices before I compute, as I come to a vertex I already have the information to compute its longest path information, so I can do the overlapping computation of the topological sort and the longest path, I can do them at the same time. While I am computing topological sort, I can simultaneously compute the longest.

(Refer Slide Time: 6:57)



Longest path algorithm

- Compute indegree of each vertex
  - Scan each column of the adjacency matrix
- Initialize ~~text~~ $longest-path-to$ to 0 for all vertices

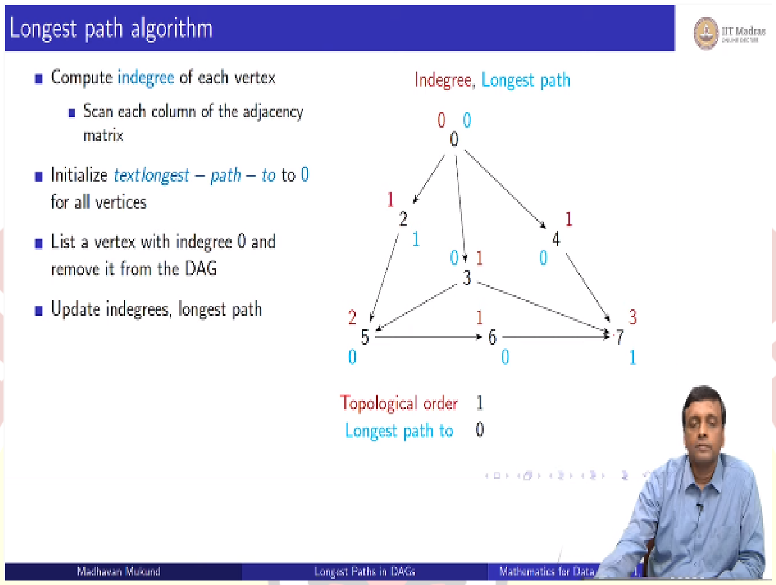
So, how do we do this? As before, we compute the indegree of every vertex, right and we also initialize this longest path, okay to be 0 for all vertices, because initially I do not know anything. So, the blue number indicates my current knowledge about the longest path, right, and the red number is the indegree which we are using for topological sort.

So, now we do this topological sort, as we go along and as we go along we update, so the first vertex, remember we did last time, we will do the same order, we picked 0, I mean, pick 1 which has indegree 0. So, if you have 1 which has indegree 0, then two things happen. One is we are going to update the indegree of everything which is pointing out of it. But we are also going to now update, we have now, we know definitively information about vertex 1. So, this maximum has a kind of monotonic property.

If I have the maximum of a set, it will only get more if I go along, so if I already know that the incoming vertex 1 has longest path 0, I cannot have of course, 0 is in this case a very simple value because a path cannot have negative values, but I cannot, the maximum cannot go below 0.

So, I already know that if I know that the, if I have frozen the value for the incoming vertex 1 as having longest path 0, then I already know that the value for 2 must be 1 + that and I can keep updating this. So, I can compute the max of those things incrementally and keep moving on.

(Refer Slide Time: 8:30)

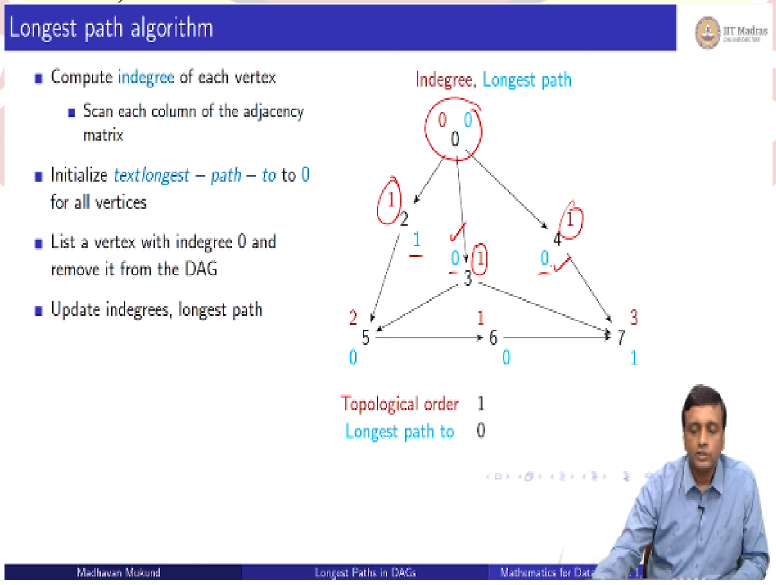


Longest path algorithm

- Compute indegree of each vertex
  - Scan each column of the adjacency matrix
- Initialize *textlongest – path – to* to 0 for all vertices
- List a vertex with indegree 0 and remove it from the DAG
- Update indegrees, longest path

So, what I will do is I will remove that vertex as before and now I will update as I said, these two entries. So, both of these entries, so the 2 will become 1 for the incoming indegree of vertex 2, the 4 will become 3 for indegree of vertex 7. But for both of them, the longest path will now go from 0 to 1 because I know that it is at least 1.

(Refer Slide Time: 8:52)

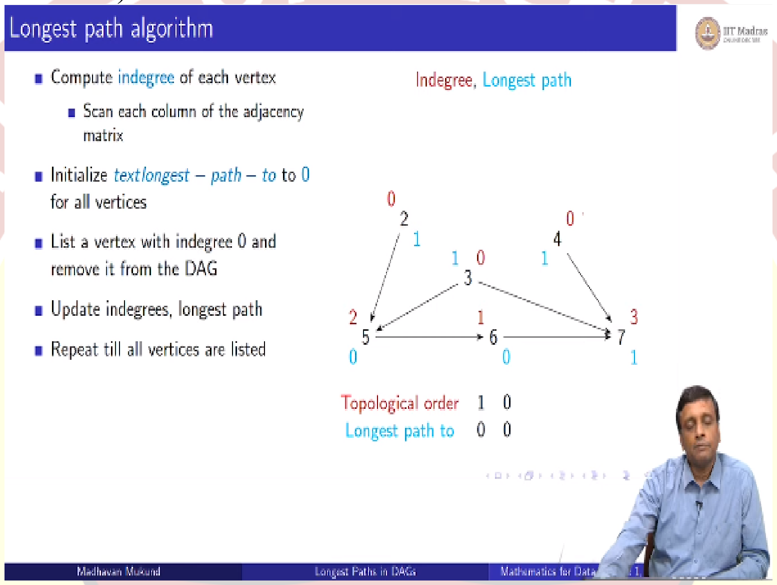


Longest path algorithm

- Compute indegree of each vertex
  - Scan each column of the adjacency matrix
- Initialize *textlongest – path – to* to 0 for all vertices
- List a vertex with indegree 0 and remove it from the DAG
- Update indegrees, longest path

Next, we did this one. So, when we do this one, we will now update as before the indegrees of this, but we will also update these. Now notice that for 2, vertex 2 it already believes its longest path is 1 and if I took the new information from 0, it is 1 + 0, which is 1, so nothing is going to happen, so 1 is going to remain 1, but for 3 and 4, where I had previously believed, I mean this without any justification that the longest path is 0. I am now going to update it to 1 + 0 as 1.

(Refer Slide Time: 9:29)

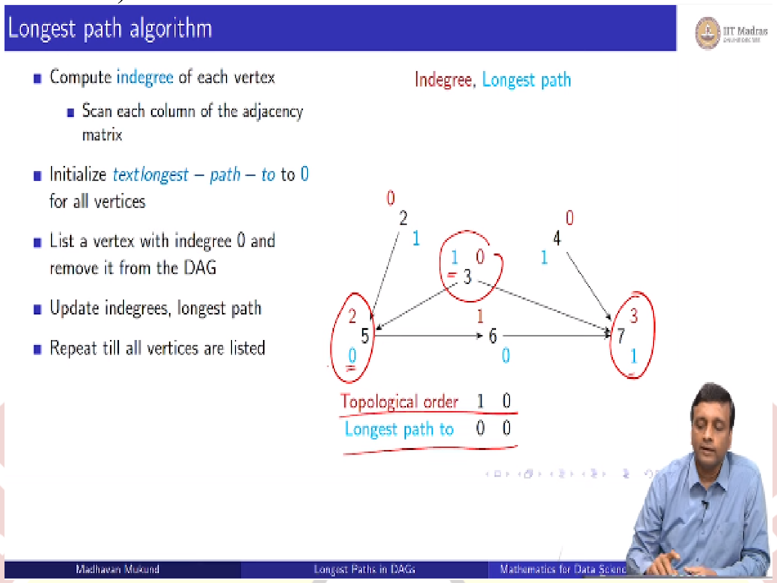


Longest path algorithm

- Compute indegree of each vertex
  - Scan each column of the adjacency matrix
- Initialize *textlongest − path − to* to 0 for all vertices
- List a vertex with indegree 0 and remove it from the DAG
- Update indegrees, longest path
- Repeat till all vertices are listed

So, when I remove the 0, the next step is I make all the indegree 0 because they have all now got no vertices pointing into them, but I also update the longest path for 3 and 4. I have also updated for 2 but no change happened because 2 already knew that it was of longest path 1.

(Refer Slide Time: 9:49)



Longest path algorithm

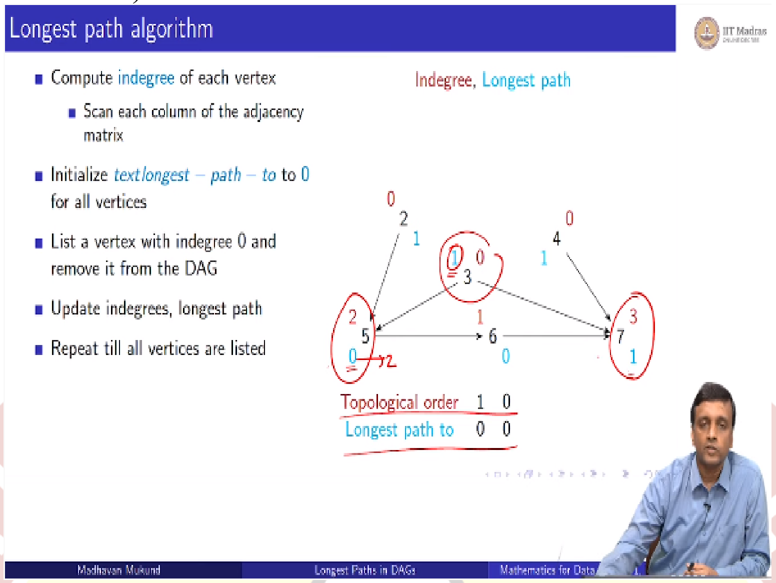
- Compute indegree of each vertex
  - Scan each column of the adjacency matrix
- Initialize *textlongest − path − to* to 0 for all vertices
- List a vertex with indegree 0 and remove it from the DAG
- Update indegrees, longest path
- Repeat till all vertices are listed

Topological order	1	0
Longest path to	0	0

Next, I will pick 2 in this, next I pick 3, sorry, this is what we did last time. So, notice that on the bottom we are keeping track of this information, right, so we are keeping track of both the topological order, the order this, so the top row indicates the vertex number as I enumerated, so I first enumerated vertex 1, then I enumerated vertex 0 and in the lower row we are keeping track of the longest path, which incrementally we are getting a final value for every vertex as we enumerate.

So, now output vertex 3 and this is going to update these values, so 3 has longest path 1, 7 believes his longest its path is 1, but now it must be at least 1 + 1, 2. So, 7 is going to change similarly, 5 is going to now move from 0 to 1 and of course, the indegree are going to reduce.

(Refer Slide Time: 10:35)



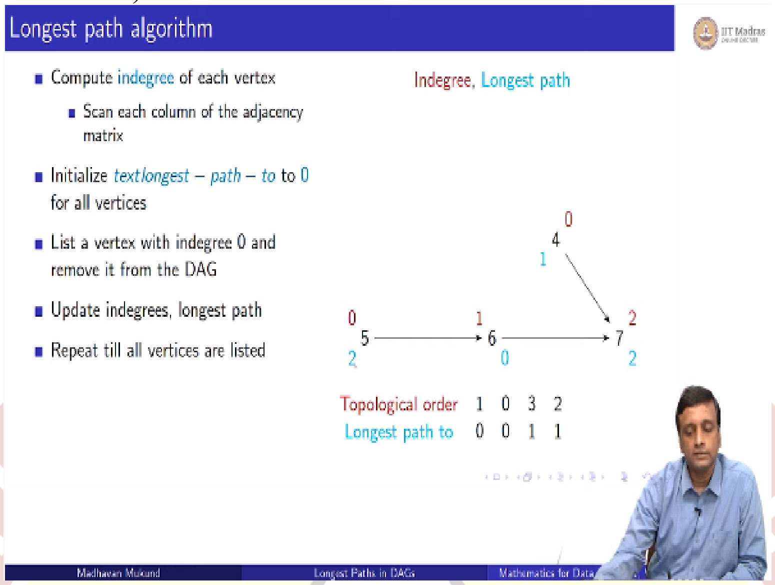
Longest path algorithm

- Compute indegree of each vertex
  - Scan each column of the adjacency matrix
- Initialize *textlongest – path – to* to 0 for all vertices
- List a vertex with indegree 0 and remove it from the DAG
- Update indegrees, longest path
- Repeat till all vertices are listed

So, I remove the 3 and then I increment the longest path here, this should be, so this is also notice that see, the path here was 1, so this goes directly from 0 to 2, it goes to something which is 1 + the maximum known incoming thing. So, the maximum known incoming thing is now 1, so it goes to 2 and this was already 1 and it goes to 2 because, so it is not that it is incrementing, it is computing 1 + max, so this goes to 2, this also goes to 2 and the degree has reduced.

So, we are overlapping this topological sort, with this longest path computation. So, in the topological sort, we are decrementing the indegree each time we remove an edge, in the max, in the longest path computation we are doing a max. So, sometimes it changes, sometimes it does not change, when it does change, it might change by more than 1 and so on, you have to keep track of what was the incoming max that you saw so far and add 1 to it.

(Refer Slide Time: 11:39)



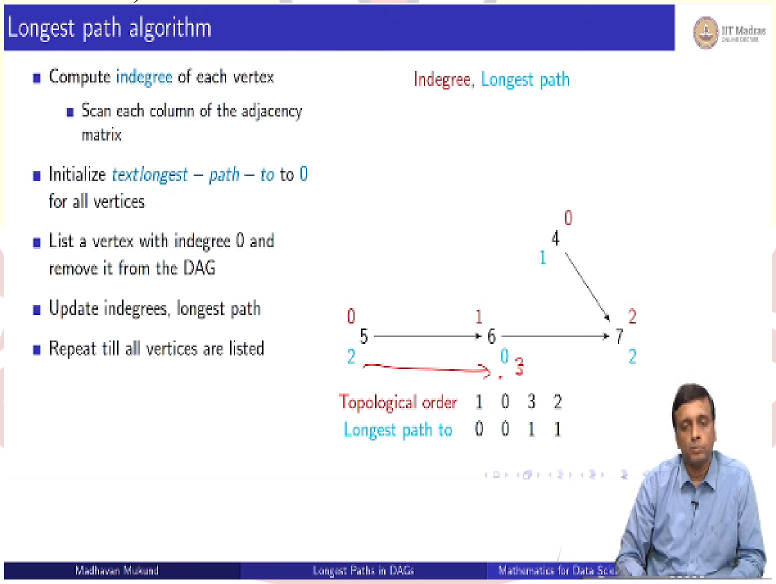
Longest path algorithm

- Compute indegree of each vertex
  - Scan each column of the adjacency matrix
- Initialize *textlongest* − *path* − *to* to 0 for all vertices
- List a vertex with indegree 0 and remove it from the DAG
- Update indegrees, longest path
- Repeat till all vertices are listed

Topological order	1	0	3	2
Longest path to	0	0	1	1

So, next we did 2, so if we do 2, then because 1 + 1 is 2, nothing is going to happen here but the indegree of 5 will reduce, so I remove 2 and now the indegree of 5 reduces but its longest path does not change.

(Refer Slide Time: 11:43)



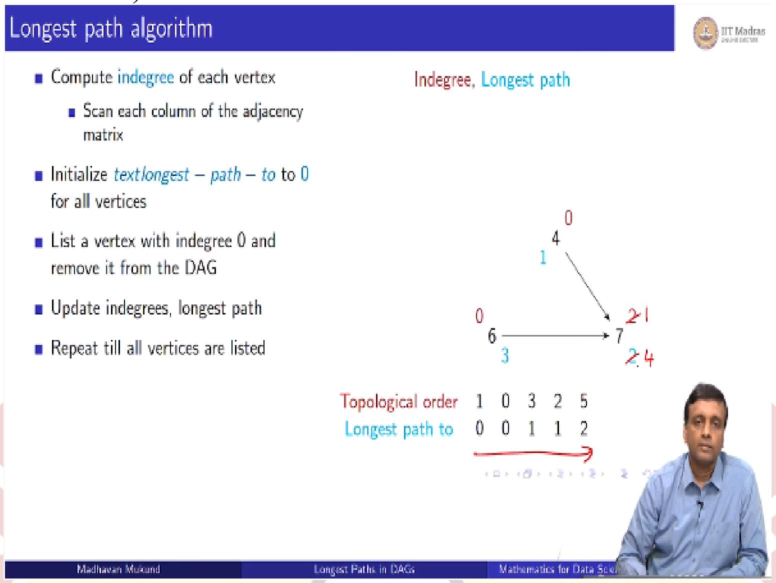
Longest path algorithm

- Compute indegree of each vertex
  - Scan each column of the adjacency matrix
- Initialize *textlongest* − *path* − *to* to 0 for all vertices
- List a vertex with indegree 0 and remove it from the DAG
- Update indegrees, longest path
- Repeat till all vertices are listed

Topological order	1	0	3	2
Longest path to	0	0	1	1

Next, I do 5 and when I do 5, this now influences this, and this says this must become 3, right, because the incoming edge to 6 has longest path 3 at the other end, 2 at the other end. So, the incoming path to 3 must be at least 2 + 1.

(Refer Slide Time: 12:00)



Longest path algorithm

- Compute indegree of each vertex
  - Scan each column of the adjacency matrix
- Initialize textlongest − path − to to 0 for all vertices
- List a vertex with indegree 0 and remove it from the DAG
- Update indegrees, longest path
- Repeat till all vertices are listed

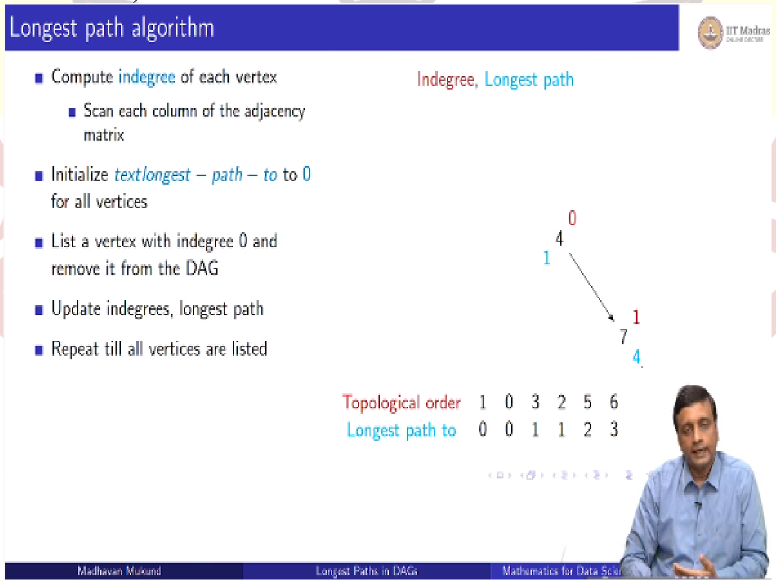
Indegree, Longest path

Topological order	1	0	3	2	5
Longest path to	0	0	1	1	2

Madhavan Mukund Longest Paths in DAGs Mathematics for Data Sci

So, now the degree of 6 will reduce from 1 to 0, but the length of the longest path will go from 0 to 3 and notice as before that we are, as we are enumerating the vertices, we are also enumerating the longest path, because now I have enumerated it, its longest path is known. Next, we did 6, so when we remove 6, then this will go to 1, but this will go up to 4, so I remove 6.

(Refer Slide Time: 12:25)



Longest path algorithm

- Compute indegree of each vertex
  - Scan each column of the adjacency matrix
- Initialize textlongest − path − to to 0 for all vertices
- List a vertex with indegree 0 and remove it from the DAG
- Update indegrees, longest path
- Repeat till all vertices are listed

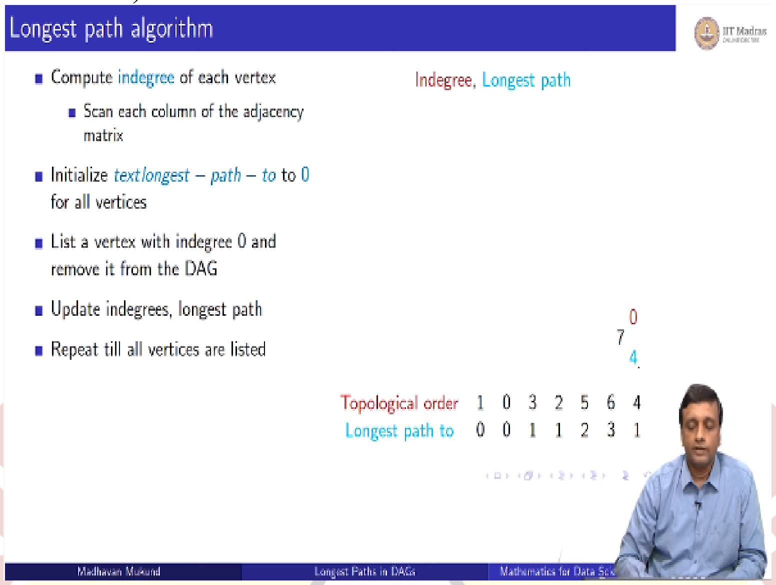
Indegree, Longest path

Topological order	1	0	3	2	5	6
Longest path to	0	0	1	1	2	3

Madhavan Mukund Longest Paths in DAGs Mathematics for Data Sci

And now 7 jumps from longest path 2 to longest path 4 because I have discovered this path of length 3 coming up to 6, which is one of the incoming neighbors. But I cannot enumerate 7 yet, because its indegree is not yet 0.

(Refer Slide Time: 12:42)



Longest path algorithm

- Compute indegree of each vertex
  - Scan each column of the adjacency matrix
- Initialize *textlongest − path − to* to 0 for all vertices
- List a vertex with indegree 0 and remove it from the DAG
- Update indegrees, longest path
- Repeat till all vertices are listed

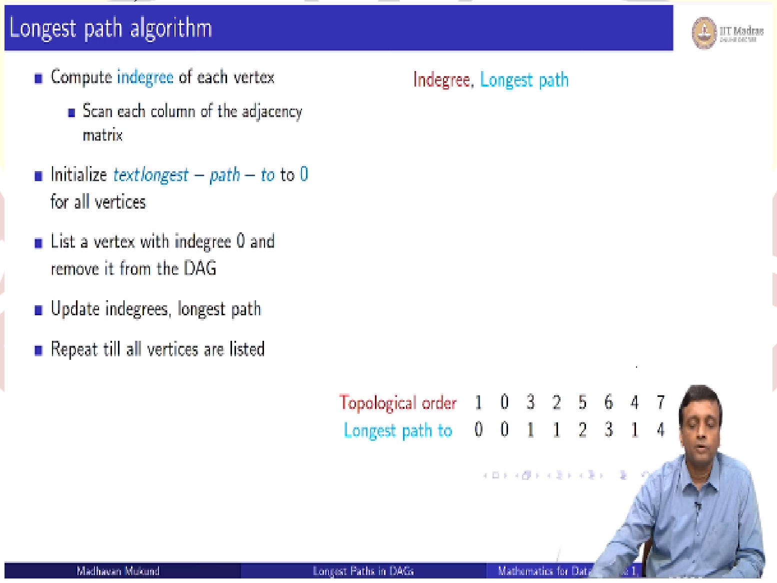
Indegree, Longest path

Topological order	1	0	3	2	5	6	4
Longest path to	0	0	1	1	2	3	1

Madhavan Mukund Longest Paths in DAGs Mathematics for Data Sci

So finally, when I output 4, at this point, the indegree of 7 becomes 0 but there is no change to the length of the longest path, because the longest path does not come through 4 it has already come through 6, and 7 has already discovered that. It is just that I cannot finalize it until I actually reach the enumeration of 7.

(Refer Slide Time: 13:02)



Longest path algorithm

- Compute indegree of each vertex
  - Scan each column of the adjacency matrix
- Initialize *textlongest − path − to* to 0 for all vertices
- List a vertex with indegree 0 and remove it from the DAG
- Update indegrees, longest path
- Repeat till all vertices are listed

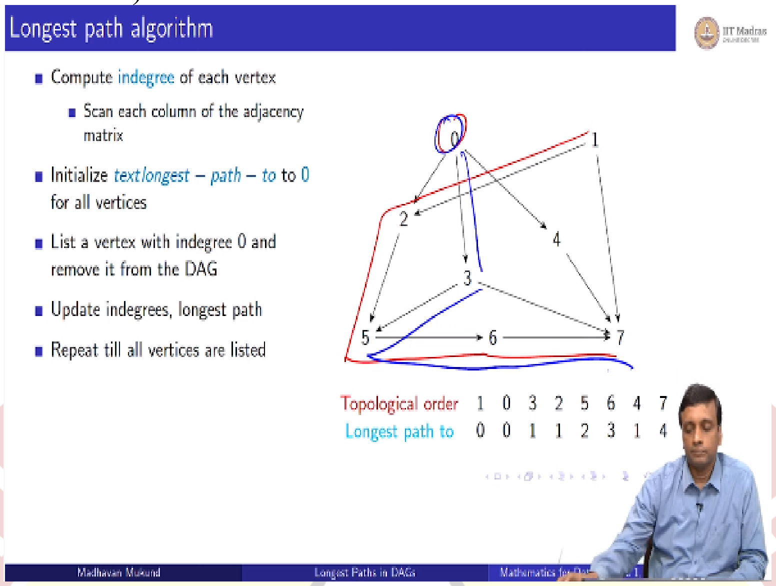
Indegree, Longest path

Topological order	1	0	3	2	5	6	4	7
Longest path to	0	0	1	1	2	3	1	4

Madhavan Mukund Longest Paths in DAGs Mathematics for Dat

So finally, I list out 7 and now I have the longest path with every vertex listed below.

(Refer Slide Time: 13:07)



Longest path algorithm

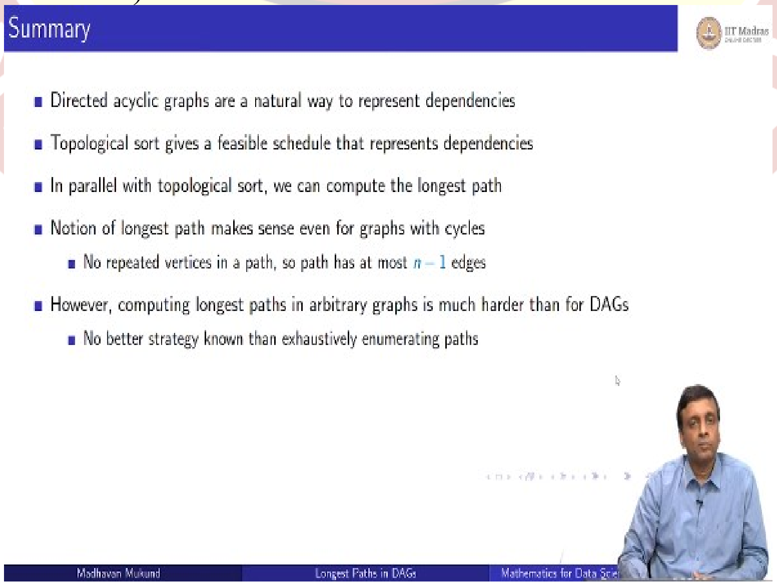
- Compute indegree of each vertex
  - Scan each column of the adjacency matrix
- Initialize *textlongest − path − to* to 0 for all vertices
- List a vertex with indegree 0 and remove it from the DAG
- Update indegrees, longest path
- Repeat till all vertices are listed

Topological order	1	0	3	2	5	6	4	7
Longest path to	0	0	1	1	2	3	1	4

Madhavan Mukund · Longest Paths in DAGs · Mathematics for Data Science

So, if you go back to the graph, you can verify for instance, that the longest path to 7 goes through say 1, 2, 3, so this is one longest path so 1, 2, 3, 4. So, basically the longest path is in terms of number of edges, at how many things I have to do before I come here, how many semesters I have to work before I do course number 7, there are multiple longest paths in general, so in this particular case, there is only one longest path to 7, I guess you can find but no, you can also find the longest path which goes this way. So, you can take a path, which goes from 0 to 3 to 5 to 6 to 7, so this is also a longest path of length. So, just the fact that you have a longest path, does not mean that is a unique one.

(Refer Slide Time: 13:51)



Summary

- Directed acyclic graphs are a natural way to represent dependencies
- Topological sort gives a feasible schedule that represents dependencies
- In parallel with topological sort, we can compute the longest path
- Notion of longest path makes sense even for graphs with cycles
  - No repeated vertices in a path, so path has at most $n-1$ edges
- However, computing longest paths in arbitrary graphs is much harder than for DAGs
  - No better strategy known than exhaustively enumerating paths

Madhavan Mukund · Longest Paths in DAGs · Mathematics for Data Science

So, just to reiterate that directed acyclic graphs are a natural way to represent dependencies. So, we saw before the topological sort is how to get a feasible schedule, right, how to extract a sequence in which I can do these tasks, such that all dependencies are satisfied before I come to a task.

But now, we also saw this problem of how to compute the shortest duration that I need if I am allowed to do tasks in parallel. So, that is this longest path problem and we said that the longest path can be computed in an overlapping way with the topological sort. Because once we process of all the dependent vertices of a vertex, then we have enough information to process that vertex itself.

So, in the same sequence that we enumerated, we can also associate with each vertex, its longest path and incrementally build this up in parallel, so we do not have to take any extra work, while we are doing topological sort we can also compute longest path. Now, you might ask whether longest path makes sense in a graph with cycles because when I have cycles, I can go round and round.

I can go to a vertex and then come back to this place and go ahead but if you remember we made a distinction between a path and a walk. So, we said a path cannot repeat a vertex, a walk can repeat a vertex. So, going around the cycle and then proceeding technically is a walk not a path. So, if you use this literal interpretation of path, so we can have longest paths in cyclic graphs also, it makes sense, what is the longest sequence of edges I can travel before I repeat a vertex? This could be at most n - 1 but is it n - 1? So, that is a question. So, you could ask the same question about longest paths in a graph which has cycles.

So, in a directed a cyclic graph, we came up with a reasonably efficient algorithm which processed each vertex only once, and then depending on how we represented it as an adjacency list or a adjacency matrix, we had to do some scanning of that to find out the incoming and outgoing edges. But beyond that, it is a very reasonable algorithm, it takes time proportional to n or n + m, I mean n squared or n + m depending on how you are doing it.

What happens in this general case, well, in this general case, actually turns out to be surprisingly hard. So, if you have cycles in your graph, you can definitely define the notion of a longest path by looking at only paths in which vertices do not repeat, right. So, you know, that is going to be n - 1, but to know whether it is n - 1 or not is surprisingly hard.

So, actually there is no known efficient way to do this. Of course, you can do it by brute force, by examining every possible path in the graph and counting its length and then taking the maximum and essentially what we do not know right now is there any better way than that to do it, right. So, there is a huge gap between the longest path problem and DAGs, in the longest path problem and directed graphs which are not DAGs.

So, in directed graphs which are not DAGs, the longest path problem is very difficult computationally compared to the relatively simple problem that we have when we have DAGs.