INDIAN INSTITUTE OF TECHNOLOGY MADRAS

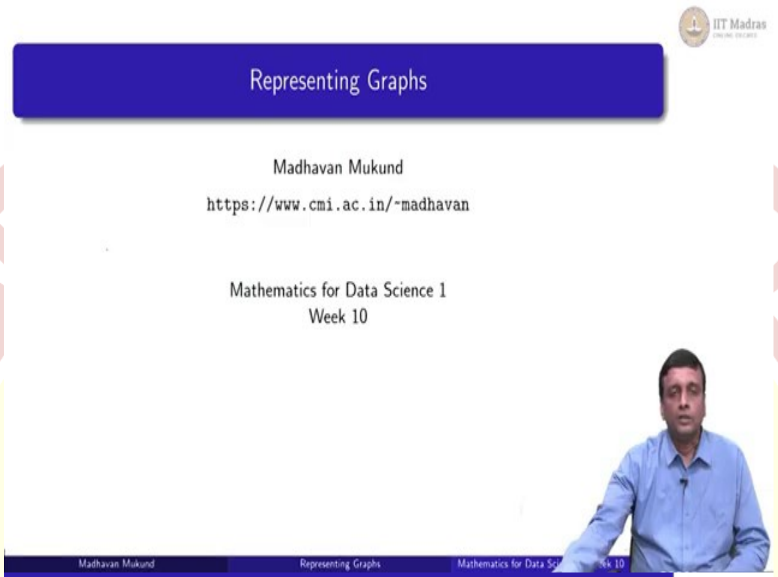
सिद्धिर्भवति कर्मजा

IIT Madras

ONLINE DEGREE

Mathematics for Data Science 1
Professor Madhavan Mukund
Chennai Mathematical Institute
Lecture: 61
Representation of graphs

(Refer Slide Time: 00:14)

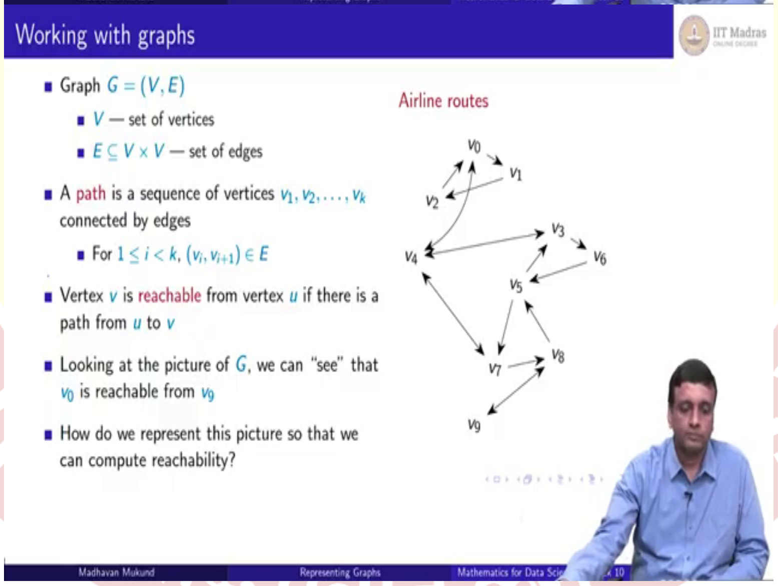


Representing Graphs

Madhavan Mukund

https://www.cmi.ac.in/~madhavan

Mathematics for Data Science 1
Week 10



Working with graphs

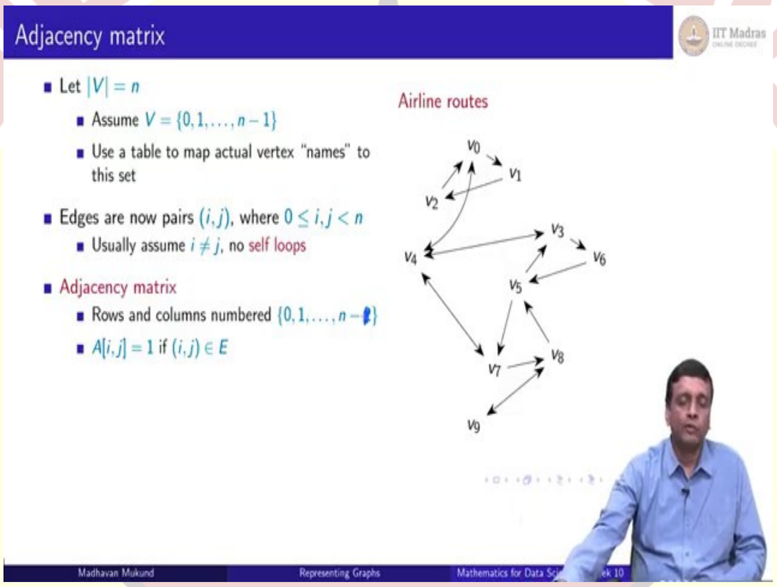
- Graph $G = (V, E)$
  - V — set of vertices
  - $E \subseteq V \times V$ — set of edges
- A path is a sequence of vertices $v_1, v_2, \ldots, v_k$ connected by edges
  - For $1 \leq i < k$, $(v_i, v_{i+1}) \in E$
- Vertex v is reachable from vertex u if there is a path from u to v
- Looking at the picture of G, we can "see" that v_0 is reachable from v_9
- How do we represent this picture so that we can compute reachability?

So, we have been talking about graphs and problems on graphs, we started with reachability. And then we talked about very complicated problems like graph colouring, and vertex cover, and so on. So, now let us get back to reachability, and connectivity and ask a more fundamental question, which is how do we actually work with these graphs in a mathematical setting. So, remember that a graph consists of 2 sets, or either a set and a relation, a set of vertices, and an edge relation. And let us focus on reachability back for now. So, a path in a graph is a sequence of connected edges. And we say that V is reachable from u, there is a path from u to v.

So, as humans, if we see a graph like this, then what we will do is take this picture and stare at it, and extract this graph in some sense visually. So, we will take a graph like this. And so, you can maybe by trial and error, but by exploration, we will do this. So, we can see that there is a path from V_9 to V_0.

The problem is that we want to operate on this mathematically, we do not want to have this picture because there is no way to formally represent this picture and how you operate on this picture, in terms of a procedure that we can execute. So, how do we represent this picture in a way that we can compute reachability, for example.

(Refer Slide Time: 01:30)



Adjacency matrix

- Let $|V| = n$
  - Assume $V = \{0, 1, \ldots, n-1\}$
  - Use a table to map actual vertex "names" to this set
- Edges are now pairs (i, j), where $0 \le i, j < n$
  - Usually assume $i \ne j$, no self loops
- Adjacency matrix
  - Rows and columns numbered $\{0, 1, \ldots, n-1\}$
  - $A[i, j] = 1$ if $(i, j) \in E$

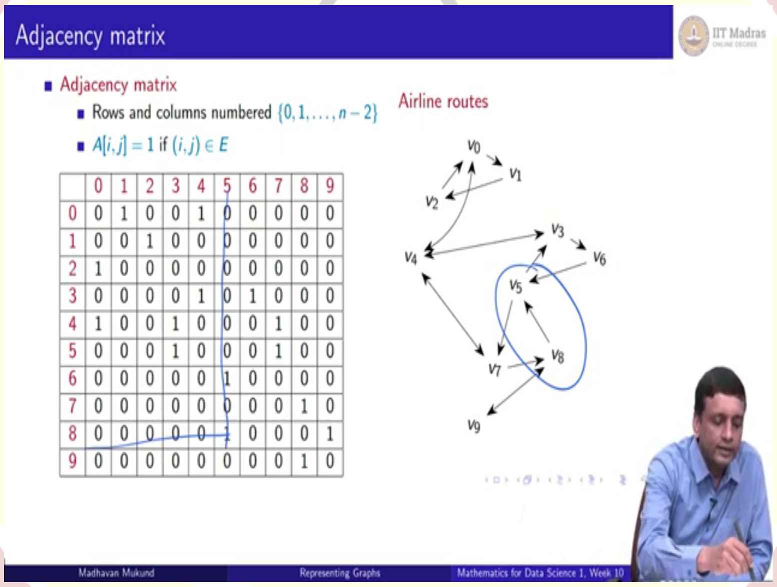
So, first, we need to represent this graph, in a way which is more amenable to computation than a picture like the one on the right. So, one way is to use what is called an Adjacency matrix. So, let us assume that the set of vertices consists of n vertices. So, this is the normal convention and graph theory that we always use small n to represent the number of vertices, and small m very often is used to represent the number of edges.

So, in particular, if we say that the vertices are n in number, then we will usually just for simplicity, number them, and call them 0 to $n - 1$, you can also call them 1 to n. But in computing, it is very common to start numbering at 0, So, we will call it 0 to n minus 1. And of course, when we actually have real vertices, like, names of cities, like in this case, Delhi and Madurai and all that, then we will actually use some kind of a table to map the actual vertex names to the set.

So, for instance, we might have a table which says a Delhi is V 0 and Madurai is V 9 and so on. So, Delhi is 0, Madurai is 9, and So, on. So, in this representation, where vertices are now numbers between 0 and $n - 1$, and edge is a pair of numbers. So, an edge is a pair i,j, where i and j lie in the range 0 to n minus 1. And we usually assume that we do not have edges like this. We do not have the so-called self-loop, So, we do not have an edge from i to i.

So, we usually assume that $i \neq j$. So, when we have an edge i,j, they both are in the range 0 to n minus 1, because that is our set of vertices, but i is ≠j. So, given this, we now have what is called an adjacency matrix. An adjacency matrix, we have rows and columns numbered from 0 to n minus 1. And then you put an entry in this matrix, 1, if there is an edge from i to j, otherwise, it is 0. So, the default is 0, and you put an 1 wherever there is a matrix.

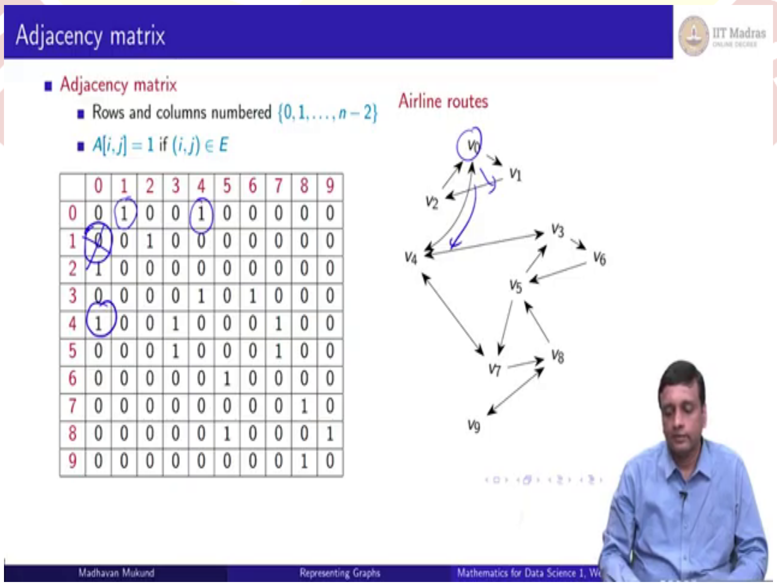
(Refer Slide Time: 03:31)



Adjacency matrix

- Adjacency matrix
  - Rows and columns numbered $\{0, 1, \ldots, n-2\}$
  - $A[i,j] = 1$ if $(i,j) \in E$

	0	1	2	3	4	5	6	7	8	9
0	0	1	0	0	1	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0
3	0	0	0	0	1	0	1	0	0	0
4	1	0	0	1	0	0	0	1	0	0
5	0	0	0	1	0	0	0	1	0	0
6	0	0	0	0	0	1	0	0	0	0
7	0	0	0	0	0	0	0	0	1	0
8	0	0	0	0	0	1	0	0	0	1
9	0	0	0	0	0	0	0	0	1	0



Adjacency matrix

- Adjacency matrix
  - Rows and columns numbered $\{0, 1, \ldots, n-2\}$
  - $A[i,j] = 1$ if $(i,j) \in E$

	0	1	2	3	4	5	6	7	8	9
0	0	1	0	0	1	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0
3	0	0	0	0	1	0	1	0	0	0
4	1	0	0	1	0	0	0	1	0	0
5	0	0	0	1	0	0	0	1	0	0
6	0	0	0	0	0	1	0	0	0	0
7	0	0	0	0	0	0	0	0	1	0
8	0	0	0	0	0	1	0	0	0	1
9	0	0	0	0	0	0	0	0	1	0

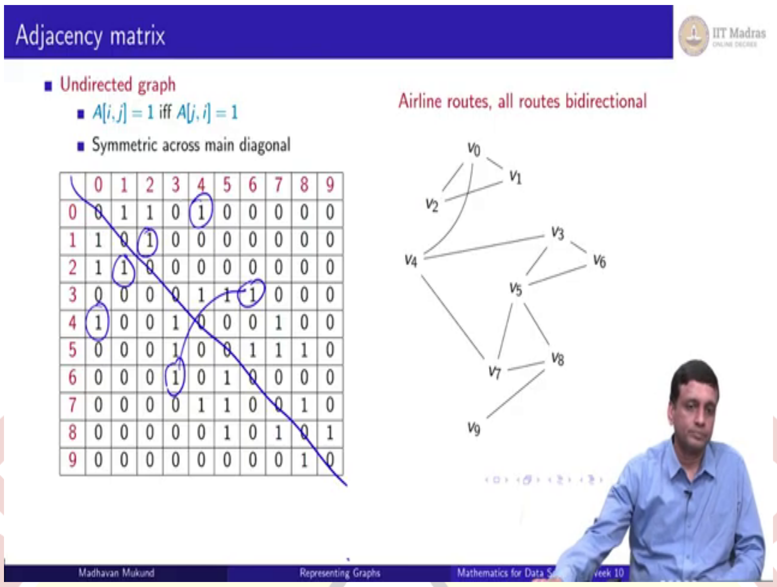
So, if we look at this graph on the right. Here is the corresponding adjacency matrix. So, first of all, because we have 10 vertices numbered 0 to 9, we have 10 rows and 10 columns, and the headers in red and the column headers and the row names on the left are in red, indicating that this is row I in column j. So, now we take an edge, say, for example, we take an edge from 8 to 5.

So, 8 to 5 says that the row 8, and the row, column 5 must be a 1. So, if I look at row 8, and I look at column 5, then I get a 1 at this position. So, that is how I fill entries in this matrix. So, you can check that this matrix represents all the edges in the graph. So, if you take any edge in this graph.

So, if you go here, for instance, and you look at this graph, and you say, 0 is connected to 1 and 4, will indeed 0 is connected to 1 and to 4. And because the edge, this bi-directional edge between 0 and 4 is actually 2 edges as we said, So, this is not an undirected graph. It is a directed graph. So, in this directed graph, we are just using a shortcut to represent it 2 edges by putting an arrow at both ends, but actually, there is a separate edge from 4 to 0.

So, there is an edge from 0 to 4, and there is an edge from 4 to 0, whereas, there is an edge from 0 to 1, but there is no corresponding edge here from 1 to 0 because there is no backward edge on that group. So, this is the adjacency matrix for this directed graph. Now, we can take the same airline route matrix.

(Refer Slide Time: 05:07)



Adjacency matrix

- Undirected graph
  - $A[i,j] = 1$ iff $A[j,i] = 1$
  - Symmetric across main diagonal

	0	1	2	3	4	5	6	7	8	9
0	0	1	1	0	1	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0
2	1	1	0	0	0	0	0	0	0	0
3	0	0	0	0	1	1	1	0	0	0
4	1	0	0	1	0	0	0	1	0	0
5	0	0	0	1	0	0	1	1	1	0
6	0	0	0	1	0	1	0	0	0	0
7	0	0	0	0	1	1	0	0	1	0
8	0	0	0	0	0	1	0	1	0	1
9	0	0	0	0	0	0	0	0	1	0

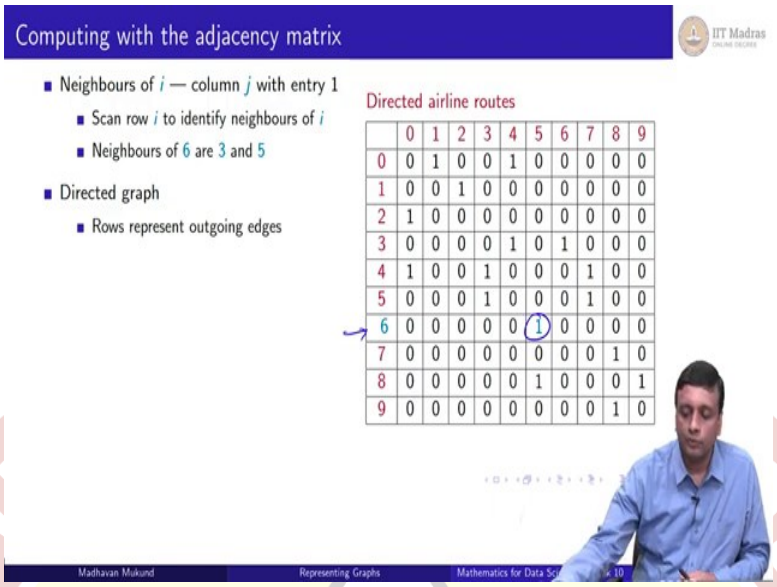
Airline routes, all routes bidirectional

And supposing we assume that all the routes are actually bi-directional. That is, whenever the airline flies from one city to another, it also flies back. So, then we do not need to record these edges anymore. And then it is better, as we said, to take such a graph where the edges are all symmetric, and explicitly call it an undirected graph, rather than recording arrows in both directions.

So, we will draw it in this fashion where we just draw an edge as a line connecting these 2 vertices with no arrows. And now if we look at this graph, every time there is an edge from i to j, there must necessarily be an edge from j to i because it is asymmetric edge relation. And if you look at the matrix, then if you go across down this main diagonal, and then if you look at any entry like this entry here, then if we look at the symmetric entry on the other side must be the same.

If there is an edge from 6 to 3, there must be an edge from 3 to 6. Similarly, there is an edge from 2 to 1, there must be an edge from 1 to 2. So, this is now our thing, this is an edge from 0 to 4, there must be an edge 4 to 0, there must be an edge 0 to 4. So, an adjacency matrix is very simple. So, we just create a row and a column for every node in your graph. And then at the intersection of the corresponding row and a column, if there is no edge, you put a 0, if there is an edge, you put a 1, that is all there is to it.

(Refer Slide Time: 06:23)



Computing with the adjacency matrix

- Neighbours of i — column j with entry 1
  - Scan row i to identify neighbours of i
  - Neighbours of 6 are 3 and 5
- Directed graph
  - Rows represent outgoing edges

Directed airline routes

	0	1	2	3	4	5	6	7	8	9
0	0	1	0	0	1	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0
3	0	0	0	0	1	0	1	0	0	0
4	1	0	0	1	0	0	0	1	0	0
5	0	0	0	1	0	0	0	1	0	0
6	0	0	0	0	0	1	0	0	0	0
7	0	0	0	0	0	0	0	0	1	0
8	0	0	0	0	0	1	0	0	0	1
9	0	0	0	0	0	0	0	0	1	0

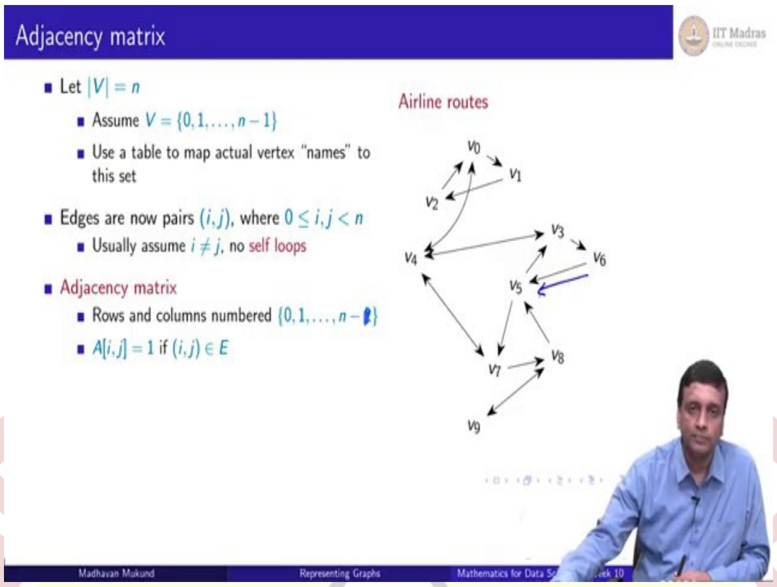
Madhavan Mukund   Representing Graphs   Mathematics for Data Sci

So, now we want to compute with this matrix. So, the whole purpose of doing this is now that this matrix on the right, for instance, suppose to represent the same picture as this undirected graph, So, we have thrown away the picture. And now we are looking at only the matrix. So, in this, if we look at the undirected graph, for instance, then if I want to know the neighbours of I, that is, which are the vertices which is I is connected by an edge, then I go to the row, for example, I am looking for neighbours of 6, I go to the row, where 6 is the starting point.

And then I find that there is a 1 entry in column 3 and column 5. So, this says that the neighbours of 6 are 3 and 5, which if you go up is indeed the case, the neighbours of 6 are 3 and 5. So, without looking at the picture, I can just, in some sense, mechanically analyze this table, or this matrix, and get the same information that I would get by looking at the picture. And this is what we need because there is no way that we can actually design a computational procedure, which looks at the picture and then makes decisions based on the picture.

So, if you have a directed graph is slightly more complicated. Because in a directed graph, remember that we have outgoing edges and incoming edges. So, the notion of a neighbour is slightly more complex. So, we have rows which represent outgoing edges. So, if I take the row for 6, So, the row for 6 has an entry pointing to 5.

(Refer Slide Time: 07:58)



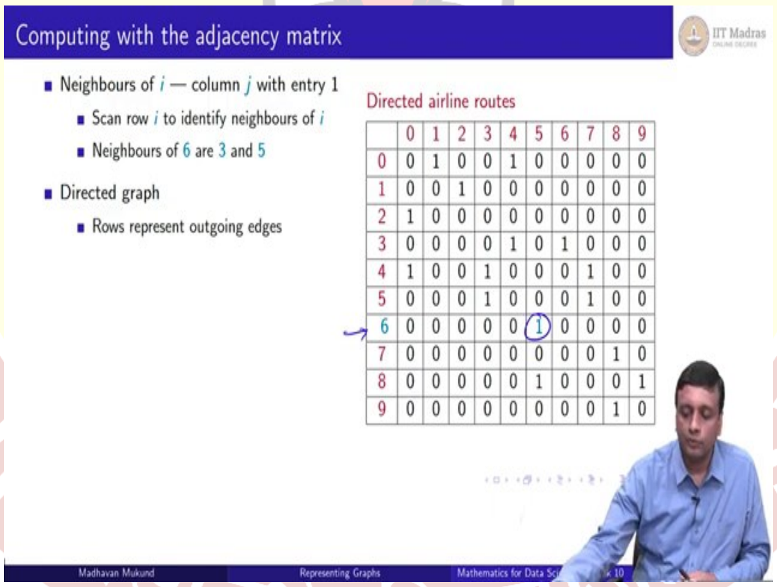
Adjacency matrix

- Let $|V| = n$
  - Assume $V = \{0, 1, \ldots, n-1\}$
  - Use a table to map actual vertex "names" to this set
- Edges are now pairs (i, j), where $0 \leq i, j < n$
  - Usually assume $i \neq j$, no self loops
- Adjacency matrix
  - Rows and columns numbered $\{0, 1, \ldots, n-1\}$
  - $A[i, j] = 1$ if $(i, j) \in E$

Madhavan Mukund Representing Graphs Mathematics for Data Sc Week 10

So, now we are looking at this picture, So, we have an entry point into 5 because there is an edge from 6 to 5, but the edge from 3 is coming in. It is not an outgoing edge.

(Refer Slide Time: 08:08)



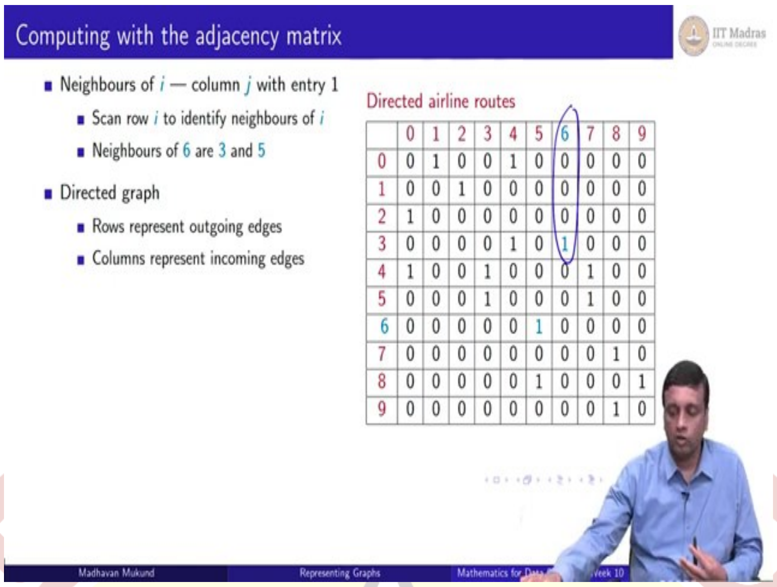
Computing with the adjacency matrix

- Neighbours of i — column j with entry 1
  - Scan row i to identify neighbours of i
  - Neighbours of 6 are 3 and 5
- Directed graph
  - Rows represent outgoing edges

Directed airline routes

	0	1	2	3	4	5	6	7	8	9
0	0	1	0	0	1	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0
3	0	0	0	0	1	0	1	0	0	0
4	1	0	0	1	0	0	0	1	0	0
5	0	0	0	1	0	0	0	1	0	0
6	0	0	0	0	0	1	0	0	0	0
7	0	0	0	0	0	0	0	0	1	0
8	0	0	0	0	0	1	0	0	0	1
9	0	0	0	0	0	0	0	0	1	0

Madhavan Mukund Representing Graphs Mathematics for Data Sc 10



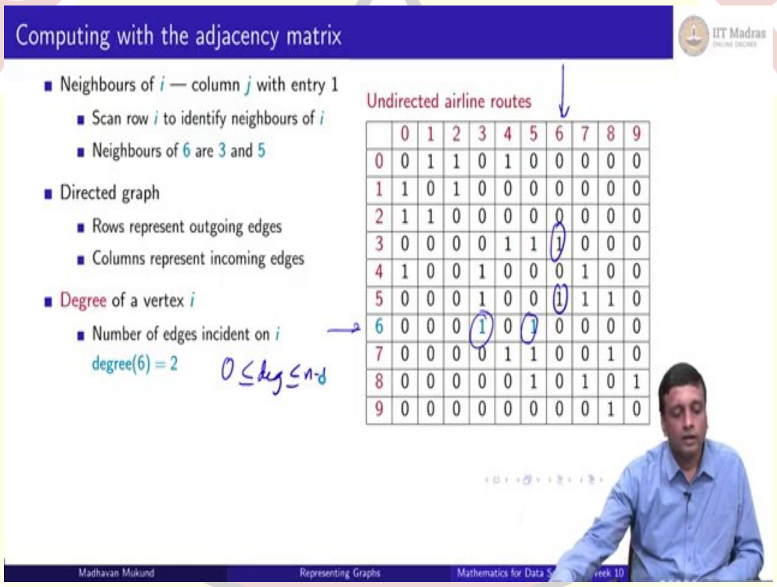
Computing with the adjacency matrix

- Neighbours of i — column j with entry 1
  - Scan row i to identify neighbours of i
  - Neighbours of 6 are 3 and 5
- Directed graph
  - Rows represent outgoing edges
  - Columns represent incoming edges

Directed airline routes

	0	1	2	3	4	5	6	7	8	9
0	0	1	0	0	1	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0
3	0	0	0	0	1	0	1	0	0	0
4	1	0	0	1	0	0	0	1	0	0
5	0	0	0	1	0	0	0	1	0	0
6	0	0	0	0	0	1	0	0	0	0
7	0	0	0	0	0	0	0	0	1	0
8	0	0	0	0	0	1	0	0	0	1
9	0	0	0	0	0	0	0	0	1	0

Madhavan Mukund | Representing Graphs | Mathematics for Data S | Week 10



Computing with the adjacency matrix

- Neighbours of i — column j with entry 1
  - Scan row i to identify neighbours of i
  - Neighbours of 6 are 3 and 5
- Directed graph
  - Rows represent outgoing edges
  - Columns represent incoming edges
- Degree of a vertex i
  - Number of edges incident on i

    degree(6) = 2 $0 \leq deg \leq n-1$

Undirected airline routes

	0	1	2	3	4	5	6	7	8	9
0	0	1	1	0	1	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0
2	1	1	0	0	0	0	0	0	0	0
3	0	0	0	0	1	1	1	0	0	0
4	1	0	0	1	0	0	0	1	0	0
5	0	0	0	1	0	0	1	1	1	0
6	0	0	0	1	0	1	0	0	0	0
7	0	0	0	0	1	1	0	0	1	0
8	0	0	0	0	0	1	0	1	0	1
9	0	0	0	0	0	0	0	0	1	0

Madhavan Mukund | Representing Graphs | Mathematics for Data S | Week 10

So, if we look at this graph, for the directed case, then we have to look at the column for 6, we have to see where which things end in 6, So, the column for 6 has an entry in row 3. So, there is an edge from 3 to 6. So, the rows represent outgoing edges. And the columns represent incoming edges. Now, in an undirected graph, these are symmetric, if I have an outgoing edge to j, then i must have an incoming edge from j. So, both of these will have the same information.

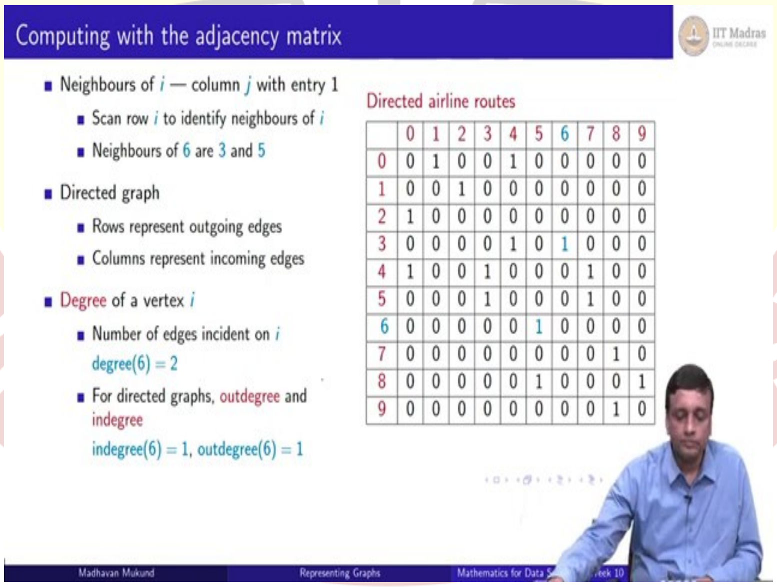
So, actually, I can also, find the neighbours of a vertex in an undirected graph by looking at the column o, it is conventional look at the row I. But the column has the same information. But in a directed graph, this is different. The outgoing edges are in the row, the incoming edges are on the column. So, the number of neighbours has a name and graph there, it is called a degree. So, the degree of a vertex is the number of edges, which are incident on that vertex that is number of edges, which start from that vertex in an undirected sense.

So, here, for instance, we saw that the degree of 6 is 2, because if I look at the row for 6, or if I look at the column for 6. So, if I look at the column for 6, also, I find that there are 2 incoming edges. So, to speak from 3 to 5, 3, and 5, and there are 2 outgoing edges from 6 to 3 and 5, this undirected. So, there is a uniform notion of degree, whether you are counting edges is coming in or going out, it does not matter. So, the degree of a vertex is the number of edges, and notice that each edge must go to a different vertex.

So, if you count the vertex you are starting at there are n minus 1 other vertices. So, the degree could be 0, in which case this vertex is not connected to anybody. I am a friendless soul, for example. Or I am in a city which is not served by this airline. So, I do not have any edges. So, I could have degree 0, or I could at most a degree n minus 1, everybody in the class is my friend.

So, I am 1 person, and all the n minus 1 other people are my friend. So, I have degree n minus 1, or I have a direct flight to every other city on the network. So, this is the case. So, the degree is between 0 and, so, 0 less than equal to degree is less than equal to $n - 1$. So, this is something that you should remember.

(Refer Slide Time: 10:13)



Computing with the adjacency matrix

- Neighbours of i — column j with entry 1
  - Scan row i to identify neighbours of i
  - Neighbours of 6 are 3 and 5
- Directed graph
  - Rows represent outgoing edges
  - Columns represent incoming edges
- Degree of a vertex i
  - Number of edges incident on i
    degree(6) = 2
  - For directed graphs, outdegree and indegree
    indegree(6) = 1, outdegree(6) = 1

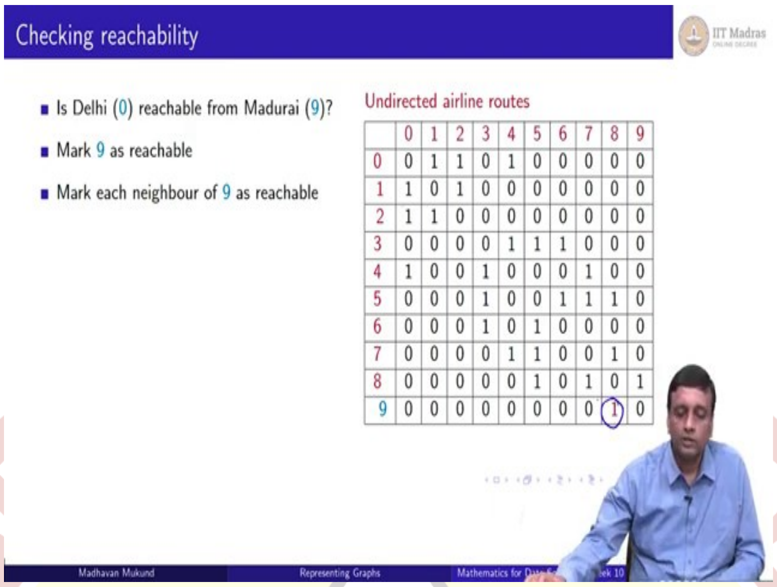
Directed airline routes

	0	1	2	3	4	5	6	7	8	9
0	0	1	0	0	1	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0
3	0	0	0	0	1	0	1	0	0	0
4	1	0	0	1	0	0	0	1	0	0
5	0	0	0	1	0	0	0	1	0	0
6	0	0	0	0	0	1	0	0	0	0
7	0	0	0	0	0	0	0	0	1	0
8	0	0	0	0	0	1	0	0	0	1
9	0	0	0	0	0	0	0	0	1	0

Madhavan Mukund | Representing Graphs | Mathematics for Data Science | Week 10

Now, if I have a directed graph, this notion of degree now gets split because I have incoming edges and outgoing edges. So, typically we will talk about the in-degree and out-degree. So, the degree of 6 in the undirected setting was 2, because there were 2 edges, but actually, 1 was pointing out to 5, and 1 was pointing in from 3. So, we can say that the end degree is 1 and the out-degree is 1.

(Refer Slide Time: 10:33)

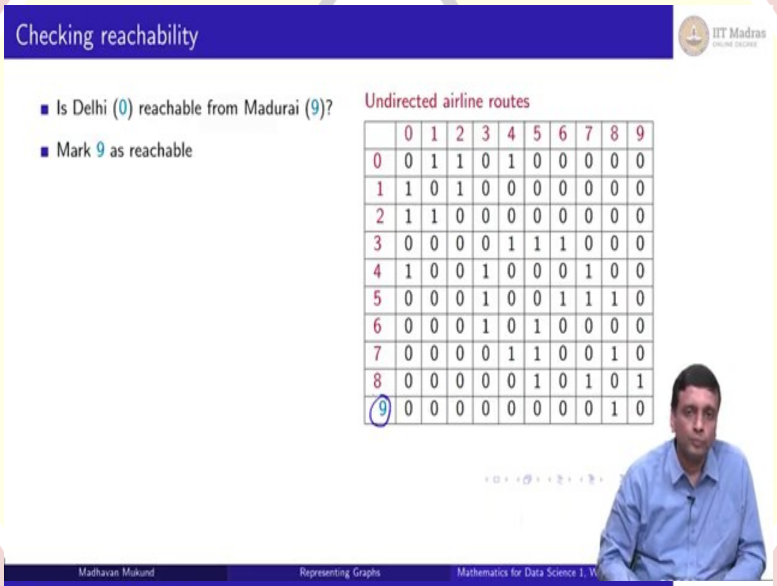


Checking reachability

- Is Delhi (0) reachable from Madurai (9)?
- Mark 9 as reachable
- Mark each neighbour of 9 as reachable

Undirected airline routes

	0	1	2	3	4	5	6	7	8	9
0	0	1	1	0	1	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0
2	1	1	0	0	0	0	0	0	0	0
3	0	0	0	0	1	1	1	0	0	0
4	1	0	0	1	0	0	0	1	0	0
5	0	0	0	1	0	0	1	1	1	0
6	0	0	0	1	0	1	0	0	0	0
7	0	0	0	0	1	1	0	0	1	0
8	0	0	0	0	0	1	0	1	0	1
9	0	0	0	0	0	0	0	0	1	0

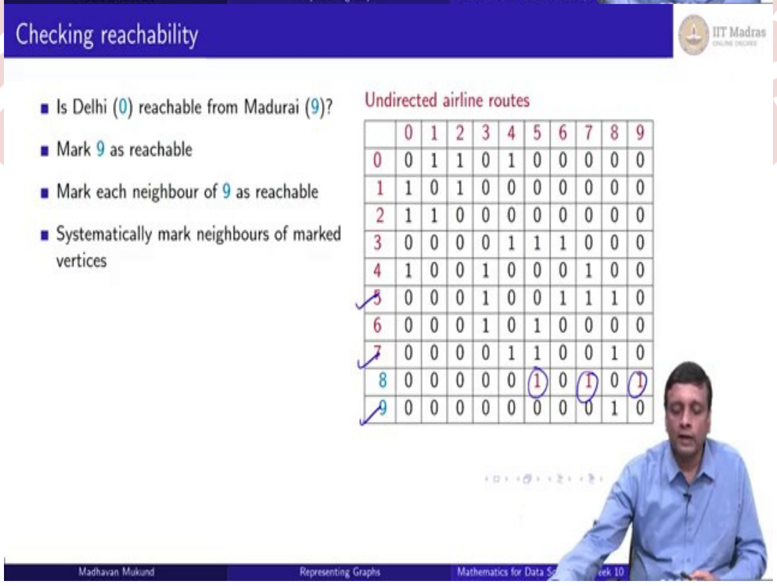


Checking reachability

- Is Delhi (0) reachable from Madurai (9)?
- Mark 9 as reachable

Undirected airline routes

	0	1	2	3	4	5	6	7	8	9
0	0	1	1	0	1	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0
2	1	1	0	0	0	0	0	0	0	0
3	0	0	0	0	1	1	1	0	0	0
4	1	0	0	1	0	0	0	1	0	0
5	0	0	0	1	0	0	1	1	1	0
6	0	0	0	1	0	1	0	0	0	0
7	0	0	0	0	1	1	0	0	1	0
8	0	0	0	0	0	1	0	1	0	1
9	0	0	0	0	0	0	0	0	1	0



Checking reachability

- Is Delhi (0) reachable from Madurai (9)?
- Mark 9 as reachable
- Mark each neighbour of 9 as reachable
- Systematically mark neighbours of marked vertices

Undirected airline routes

	0	1	2	3	4	5	6	7	8	9
0	0	1	1	0	1	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0
2	1	1	0	0	0	0	0	0	0	0
3	0	0	0	0	1	1	1	0	0	0
4	1	0	0	1	0	0	0	1	0	0
5	0	0	0	1	0	0	1	1	1	0
6	0	0	0	1	0	1	0	0	0	0
7	0	0	0	0	1	1	0	0	1	0
8	0	0	0	0	0	1	0	1	0	1
9	0	0	0	0	0	0	0	0	1	0

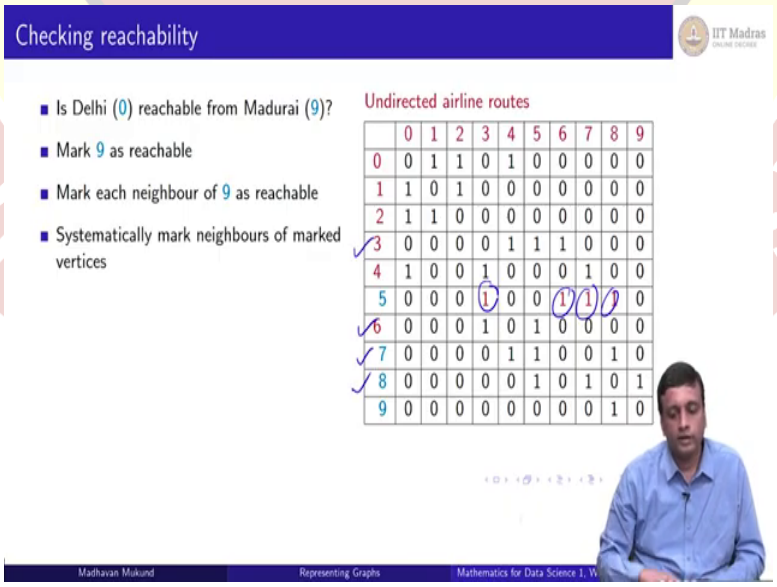
So, our goal, as we said, was to compute. So, we want to do something with this graph. So, in this particular case, how can I use this Adjacency matrix to check whether Delhi which is the vertex 0 is reachable from Madurai, which is a vertex 9. So, we do the natural thing, which is we start at 9, and then we see where all we can go.

So, we first mark that from 9, we can go to 9. So, by marking what I mean is, I will now take the row entry, and I will change the colour from red to blue. So, 9 is now reachable. So, now I can look at the neighbours of 9, in this particular case, there is only 1 neighbour 8. And if 9 is reachable, and I can fly from 9 to 8, then 8 is also reachable. So, I would mark 8 as reachable.

Now, what do I do, I have to start from 8 and see where all I can go. So, I Just have to systematically repeat this procedure. So, I have to systematically mark all the neighbours of marked vertices. So, 8 was marked. So, now 8 has 3 neighbours 5, 7, and 9. Now, notice that I do not have to refer to the picture, I am only referring to a table, I just have to look at the row for 8. And if I look at the row for 8, I know what the neighbours are at 8.

So, I do not have to go back to the picture. And imagine anything, this is just a mechanical analysis of this table. So, I look at this, and this tells me that I must now mark 5, I must mark 7. And I must mark 9, but 9 is already marked, so, it would not have any effect. So, the next step is to mark these new guys as also being visited or reachable.

(Refer Slide Time: 12:14)



Checking reachability

- Is Delhi (0) reachable from Madurai (9)?
- Mark 9 as reachable
- Mark each neighbour of 9 as reachable
- Systematically mark neighbours of marked vertices

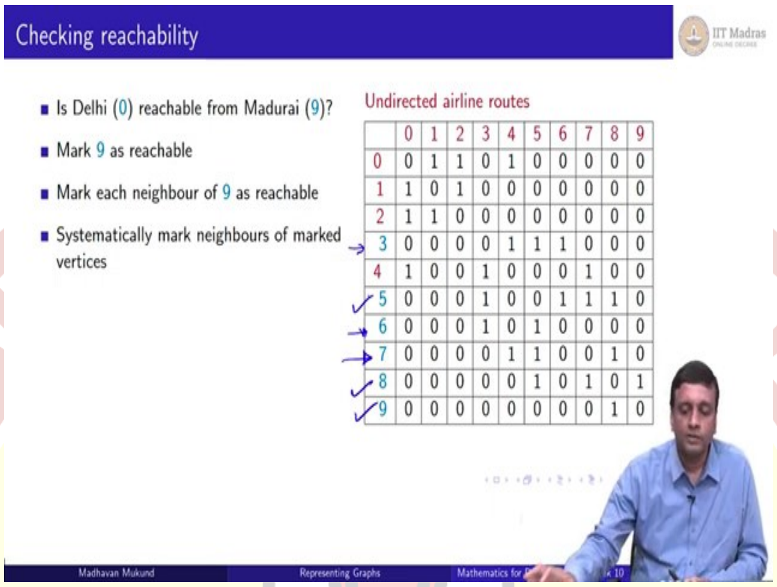
Undirected airline routes

	0	1	2	3	4	5	6	7	8	9
0	0	1	1	0	1	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0
2	1	1	0	0	0	0	0	0	0	0
✓3	0	0	0	0	1	1	1	0	0	0
4	1	0	0	1	0	0	0	1	0	0
5	0	0	0	1	0	0	1	1	1	0
✓6	0	0	0	1	0	1	0	0	0	0
✓7	0	0	0	0	1	1	0	0	1	0
✓8	0	0	0	0	0	1	0	1	0	1
9	0	0	0	0	0	0	0	0	1	0

Madhavan Mukund Representing Graphs Mathematics for Data Science 1, W

So, I mark 5, 7 and 9, as also reachable. So, now I have from 9, I can reach 5, 7, and 8, other than 9 itself. So, now I have not been very careful to keep track of it. But I know that I have explored the neighbours of 8. But I have now discovered 2 new neighbours, which I could reach 5 and 7. So, I pick 1 of them say pick 5. So, I look at the neighbours of 5. So, 5 has his neighbours 3, 6, 7 and 8. So, I have to know Mark 3, 6, 7 and 8 for which 7 and 8 are already known. So, I would mark 3 and 6.

(Refer Slide Time: 12:47)



Checking reachability

- Is Delhi (0) reachable from Madurai (9)?
- Mark 9 as reachable
- Mark each neighbour of 9 as reachable
- Systematically mark neighbours of marked vertices

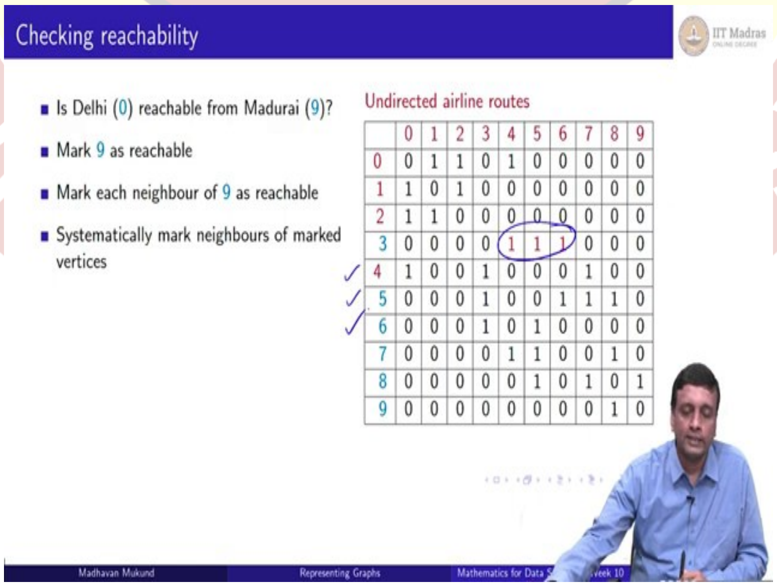
Undirected airline routes

		0	1	2	3	4	5	6	7	8	9
	0	0	1	1	0	1	0	0	0	0	0
	1	1	0	1	0	0	0	0	0	0	0
	2	1	1	0	0	0	0	0	0	0	0
→	3	0	0	0	0	1	1	1	0	0	0
	4	1	0	0	1	0	0	0	1	0	0
✓	5	0	0	0	1	0	0	1	1	1	0
→	6	0	0	0	1	0	1	0	0	0	0
→	7	0	0	0	0	1	1	0	0	1	0
✓	8	0	0	0	0	0	1	0	1	0	1
✓	9	0	0	0	0	0	0	0	0	1	0

Madhavan Mukund | Representing Graphs | Mathematics for | 10

Now, once again, I have now as unexplored things I had marked 7, but I have not looked at the numbers of 7, I have marked now 3 and not looked at it and. So, 5, 8, and 9 have been explored that is I marked them, and then I also mark their neighbours. So, let me again go to the smallest unmark things with 3.

(Refer Slide Time: 13:11)



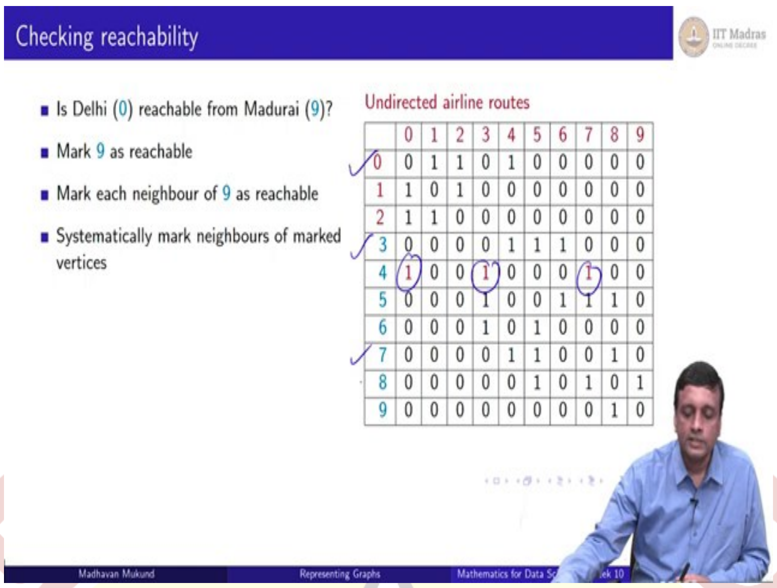
Checking reachability

- Is Delhi (0) reachable from Madurai (9)?
- Mark 9 as reachable
- Mark each neighbour of 9 as reachable
- Systematically mark neighbours of marked vertices

Undirected airline routes

		0	1	2	3	4	5	6	7	8	9
	0	0	1	1	0	1	0	0	0	0	0
	1	1	0	1	0	0	0	0	0	0	0
	2	1	1	0	0	0	0	0	0	0	0
	3	0	0	0	0	1	1	1	0	0	0
✓	4	1	0	0	1	0	0	0	1	0	0
✓	5	0	0	0	1	0	0	1	1	1	0
✓	6	0	0	0	1	0	1	0	0	0	0
	7	0	0	0	0	1	1	0	0	1	0
	8	0	0	0	0	0	1	0	1	0	1
	9	0	0	0	0	0	0	0	0	1	0

Madhavan Mukund | Representing Graphs | Mathematics for Data S | Week 10



Checking reachability

- Is Delhi (0) reachable from Madurai (9)?
- Mark 9 as reachable
- Mark each neighbour of 9 as reachable
- Systematically mark neighbours of marked vertices

Undirected airline routes

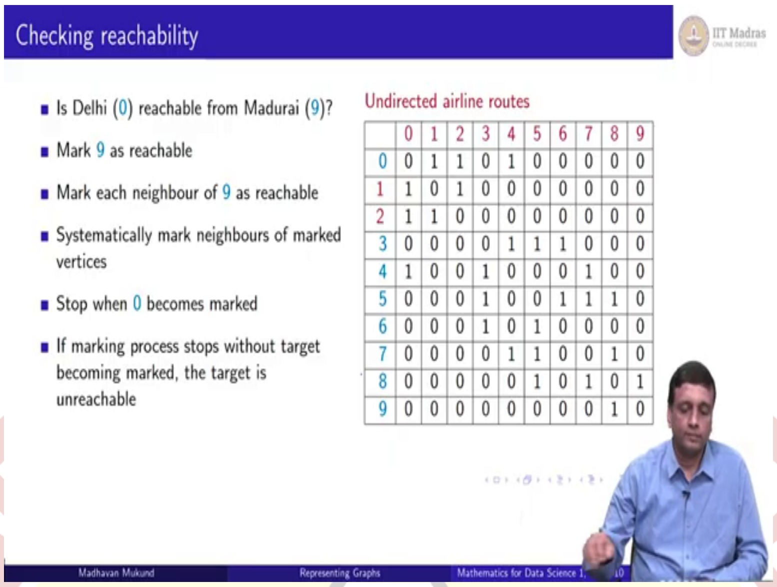
	0	1	2	3	4	5	6	7	8	9
✓0	0	1	1	0	1	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0
2	1	1	0	0	0	0	0	0	0	0
✓3	0	0	0	0	1	1	1	0	0	0
4	1	0	0	1	0	0	0	1	0	0
5	0	0	0	1	0	0	1	1	1	0
6	0	0	0	1	0	1	0	0	0	0
✓7	0	0	0	0	1	1	0	0	1	0
8	0	0	0	0	0	1	0	1	0	1
9	0	0	0	0	0	0	0	0	1	0

Madhavan Mukund | Representing Graphs | Mathematics for Data Sc | ek 10

So, I look at 3 and I look at its neighbours. So, the neighbours of 3 are 4, 5, and 6. So, this means I was marked 4, 5, and 6, of which 5 and 6 were already marked before. So, nothing happens, but 4 gets marked. And now the smallest unexplored vertex is 4. So, I look at the outgoing neighbours of 4, and I get 0, 3, and 7, and therefore I must Mark 0, 3, and 7. And, So, once I have marked 0, 3, and 7, I can stop because this was my target, my target was to find out whether 0 is reachable from 9.

What I have seen is by systematically marking everything which is reachable in 1 statement, 2 steps and so, on, I have found out a way of reaching 0 from 9. So, I may or may not be able to reach 1 and 2, I do not need to do it for this particular calculation, if I wanted to find out what all is reachable from 9, I would continue and see if 1 and 2 get marked. But in this particular case, my only goal was to find out whether 0 is reachable from 9, I can stop.

(Refer Slide Time: 14:08)



Checking reachability

- Is Delhi (0) reachable from Madurai (9)?
- Mark 9 as reachable
- Mark each neighbour of 9 as reachable
- Systematically mark neighbours of marked vertices
- Stop when 0 becomes marked
- If marking process stops without target becoming marked, the target is unreachable

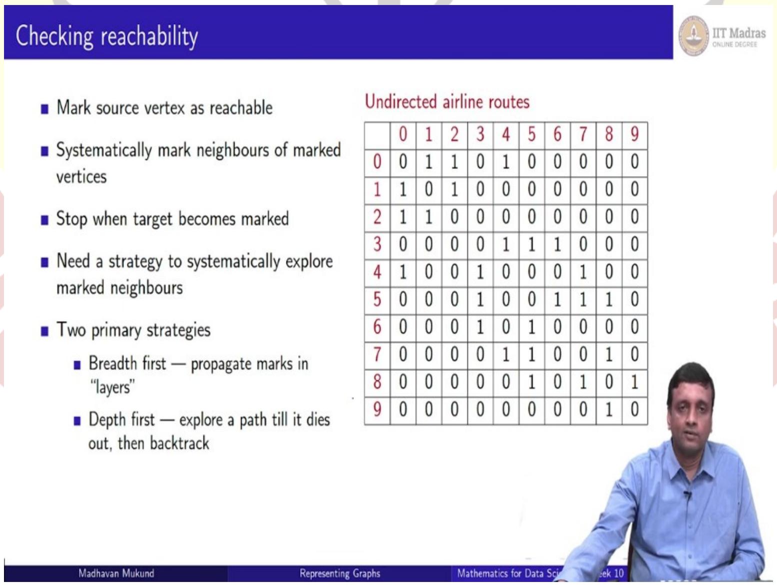
Undirected airline routes

	0	1	2	3	4	5	6	7	8	9
0	0	1	1	0	1	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0
2	1	1	0	0	0	0	0	0	0	0
3	0	0	0	0	1	1	1	0	0	0
4	1	0	0	1	0	0	0	1	0	0
5	0	0	0	1	0	0	1	1	1	0
6	0	0	0	1	0	1	0	0	0	0
7	0	0	0	0	1	1	0	0	1	0
8	0	0	0	0	0	1	0	1	0	1
9	0	0	0	0	0	0	0	0	1	0

Madhavan Mukund Representing Graphs Mathematics for Data Science 1, 10

So, the other situation is that it is perhaps not possible to reach 0 from 9 and what will happen there is that after I mark everything that can be marked, I will find that 0 is still not marked. So, if at the end of this process, where I have marked all the vertices and all the neighbours of all the vertices and I cannot mark anything more, and I find that some vertex remains unmarked, then that vertex is not reachable from where I started.

(Refer Slide Time: 14:34)



Checking reachability

- Mark source vertex as reachable
- Systematically mark neighbours of marked vertices
- Stop when target becomes marked
- Need a strategy to systematically explore marked neighbours
- Two primary strategies
  - Breadth first — propagate marks in "layers"
  - Depth first — explore a path till it dies out, then backtrack

Undirected airline routes

	0	1	2	3	4	5	6	7	8	9
0	0	1	1	0	1	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0
2	1	1	0	0	0	0	0	0	0	0
3	0	0	0	0	1	1	1	0	0	0
4	1	0	0	1	0	0	0	1	0	0
5	0	0	0	1	0	0	1	1	1	0
6	0	0	0	1	0	1	0	0	0	0
7	0	0	0	0	1	1	0	0	1	0
8	0	0	0	0	0	1	0	1	0	1
9	0	0	0	0	0	0	0	0	1	0

Madhavan Mukund Representing Graphs Mathematics for Data Sc eek 10

So, abstractly, what we said is we mark the starting vertex of the source vertex is reachable, and we systematically mark neighbours of mark vertices and we stop when the target becomes marked. So, this is what we computed. And we did this using the matrix that was the important thing that we did not go back to the picture and try to follow edges on the graph as a picture, but rather we scan the rows and we did some recolouring of the row headers. And in this process, we were able to explicitly compute whether we could reach 0 from 9 or not.

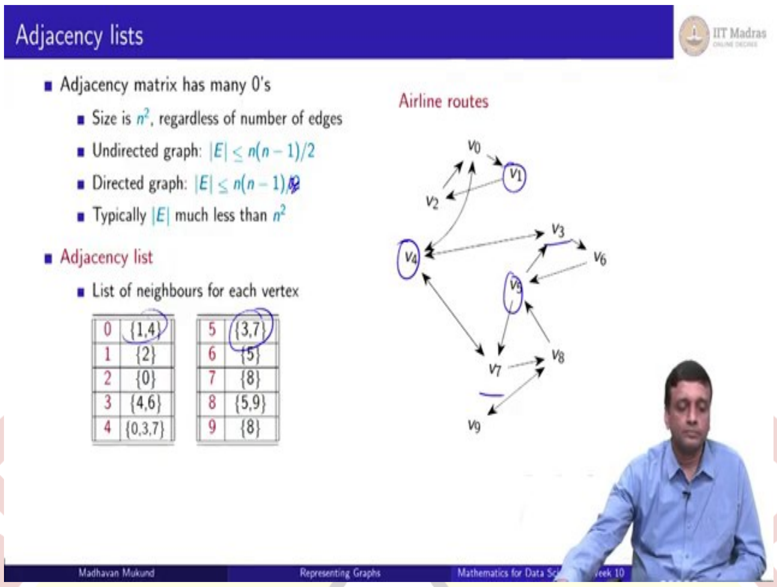
So, we had a kind of ad hoc strategy, which said that we will mark some things, and then we will pick up the smallest thing we have not explored and all that, but we did not systematically tell, how to do that. So, we actually need to elaborate that strategy more carefully, in order to get an actual computational procedure out of this.

So, how do you systematically explore the mark neighbours? So, it turns out that there are 2 broad strategies for this. So, one is called breadth-first, which is what we were doing in a sense, but not really, which is that you propagate the marks and layers. So, we look at things that are reachable in 1 step. And then from 1 step, we look at things reachable in 2 steps, and so, on.

And the other strategy is called depth-first, which is I find one place, I can go to, there may be 3 places I can go to, but instead of going to the second place, I go to the place I could go to first and from there, I start exploring further where I can go. So, then I keep going down that path. And then eventually, I hit a dead-end, saying that there is no more places I can reach. And then I go back, and I say, Okay, now let me pick up the second place I could have started with and see where all I can go from there.

So, this is called depth-first search. So, you go as far as you can go in 1 direction, and your backup. And then you take the second direction back up until you run out of directions. And you keep doing this at every point. And this is called depth-first search. So, we will study breadth-first and depth-first search in more detail as we go along.

(Refer Slide Time: 16:28)



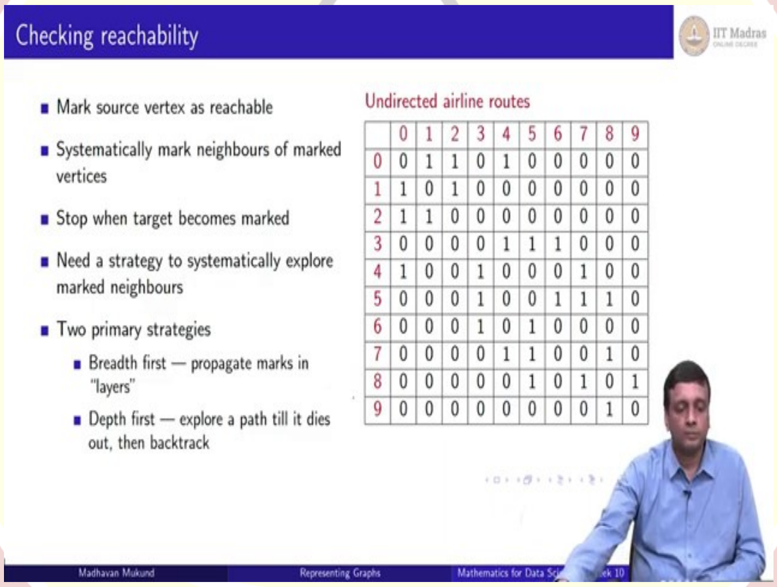
Adjacency lists

- Adjacency matrix has many 0's
  - Size is n^2, regardless of number of edges
  - Undirected graph: $|E| \leq n(n-1)/2$
  - Directed graph: $|E| \leq n(n-1)$
  - Typically $|E|$ much less than n^2
- Adjacency list
  - List of neighbours for each vertex

0	{1,4}
1	{2}
2	{0}
3	{4,6}
4	{0,3,7}

5	{3,7}
6	{5}
7	{8}
8	{5,9}
9	{8}

Airline routes



Checking reachability

- Mark source vertex as reachable
- Systematically mark neighbours of marked vertices
- Stop when target becomes marked
- Need a strategy to systematically explore marked neighbours
- Two primary strategies
  - Breadth first — propagate marks in "layers"
  - Depth first — explore a path till it dies out, then backtrack

Undirected airline routes

	0	1	2	3	4	5	6	7	8	9
0	0	1	1	0	1	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0
2	1	1	0	0	0	0	0	0	0	0
3	0	0	0	0	1	1	1	0	0	0
4	1	0	0	1	0	0	0	1	0	0
5	0	0	0	1	0	0	1	1	1	0
6	0	0	0	1	0	1	0	0	0	0
7	0	0	0	0	1	1	0	0	1	0
8	0	0	0	0	0	1	0	1	0	1
9	0	0	0	0	0	0	0	0	1	0

But before we do that, let us get back to this notion of how to represent a graph. So, the strategy that we have seen so, far is to use this adjacency matrix. So, the problem with the adjacency matrix is that as we have seen before, if you look at the adjacency matrix, here, you will see that there is a large number of 0s and 0s convey no information to us, it is only the once which are useful to us. So, the number of once is relatively small compared to the number of 0s. So, 0s are non-edges, and once are edges. And we are only interested only in the edges.

So, the size of this adjacency matrix, if I have n vertices is going to be n squared, regardless of how many edges I actually have. And now this is not a real problem in the long run, or in the extreme case, because you could have about n squared edges. So, if you have an undirected graph, then every pair of vertices will actually be an edge. So, you will have in terms of undirected edges, you will have n into n minus 1 by 2 because every edge is counted twice. But if you look at the matrix, it will actually have n into n minus 1 entry, because i j will be in the entry, and j i will also be in their entry. So, i j and j i represent the same edge. So, the number of actual edges is half that, but the number of entries is going to be n into $n - 1$.

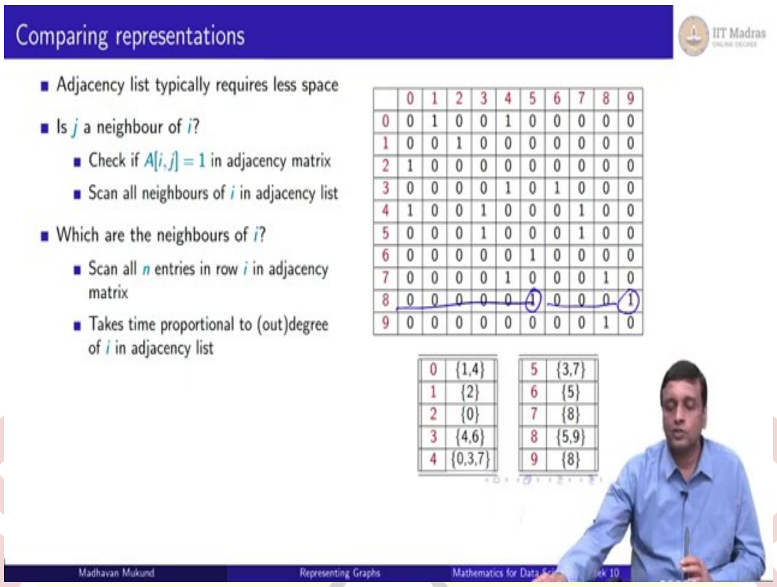
And of course, it was a directed graph. Also, you have n into n minus 1 should not be by 2. So, in both cases, you could have about n squared edges, but in typically you will not have n squared. So, typically a graph will have much fewer than n^2 entries. And if you have much fewer than n^2 entries, then it is not clear that having this large matrix is the best way to represent a graph.

So, the other option is to just directly represent the neighbours of each matrix of each vertex in a list. So, this is what is called an adjacency list. So, in an adjacency list, what you do is you write down for each vertex, which are the neighbours of that vertex. So, it is most sensible in a directed graph.

So, let us look at this directed graph here. So, it says that 0 is connected to 1 and 4. So, again, 0, you put this list 1, 4, similarly, 5 is connected to 3, and 7. So, against 5, you put 3 and 7. So, for each vertex in your graph, you just record what would have been in the adjacency matrix, what will be in the 1s in the row for that vertex.

So, if you look at row 5 in the vertex in the adjacency matrix, for this directed graph, it will have 2 once at 3 and 7, So, instead of writing all the 0s on the other 8 positions, we Just write the 1 position as 3 and 7. So, this is an adjacency list. So, this is an alternative way of presenting a graph, and we can work with this representation as well.

(Refer Slide Time: 19:06)

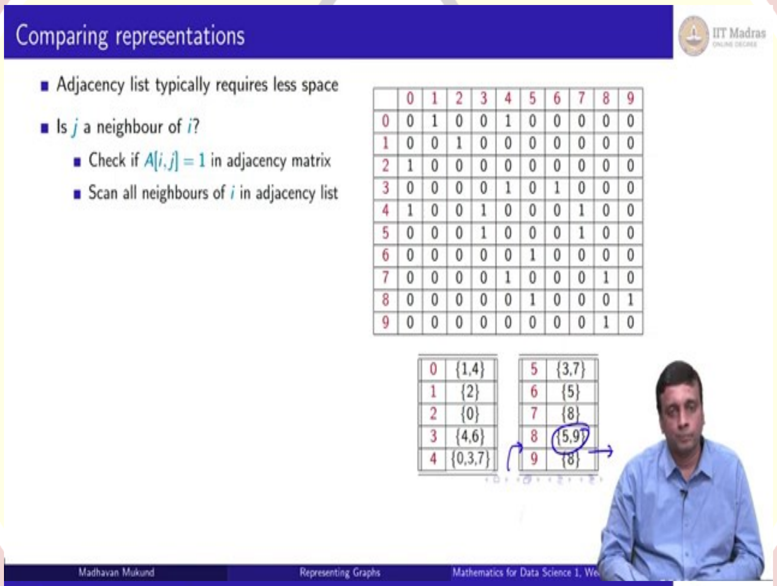


Comparing representations

- Adjacency list typically requires less space
- Is j a neighbour of i?
  - Check if $A[i,j] = 1$ in adjacency matrix
  - Scan all neighbours of i in adjacency list
- Which are the neighbours of i?
  - Scan all n entries in row i in adjacency matrix
  - Takes time proportional to (out)degree of i in adjacency list

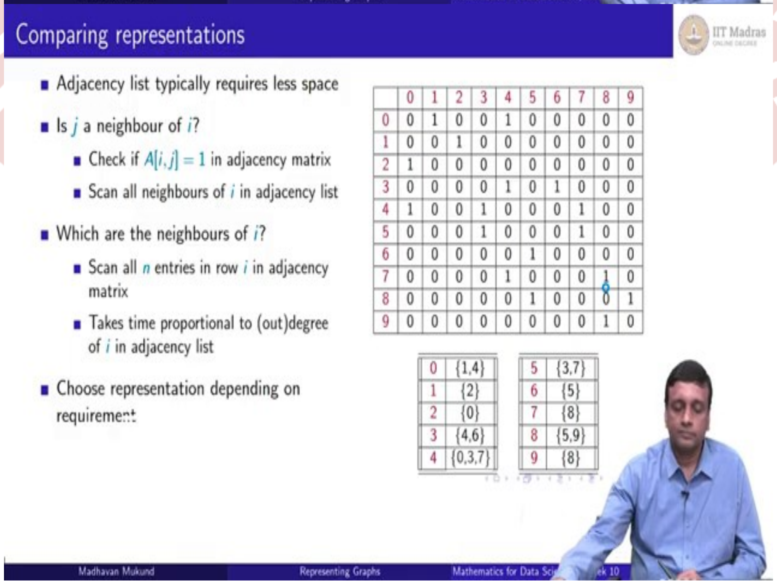
	0	1	2	3	4	5	6	7	8	9
0	0	1	0	0	1	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0
3	0	0	0	0	1	0	1	0	0	0
4	1	0	0	1	0	0	0	1	0	0
5	0	0	0	1	0	0	0	1	0	0
6	0	0	0	0	0	1	0	0	0	0
7	0	0	0	0	1	0	0	0	1	0
8	0	0	0	0	0	1	0	0	0	1
9	0	0	0	0	0	0	0	0	1	0

0	{1,4}	5	{3,7}
1	{2}	6	{5}
2	{0}	7	{8}
3	{4,6}	8	{5,9}
4	{0,3,7}	9	{8}



Comparing representations

- Adjacency list typically requires less space
- Is j a neighbour of i?
  - Check if $A[i,j] = 1$ in adjacency matrix
  - Scan all neighbours of i in adjacency list



Comparing representations

- Adjacency list typically requires less space
- Is j a neighbour of i?
  - Check if $A[i,j] = 1$ in adjacency matrix
  - Scan all neighbours of i in adjacency list
- Which are the neighbours of i?
  - Scan all n entries in row i in adjacency matrix
  - Takes time proportional to (out)degree of i in adjacency list

- Choose representation depending on requirement

So, on the right-hand side, you see now, the 2 representations for that particular graph we have been looking at the top is the adjacency matrix for that directed airline graph. And the bottom is the adjacency list for the directed airline graph. And it is obvious from this picture, that the adjacency list is typically much more compact in terms of the amount of space that it occupies.

But of course, you have to do different things when you compute with these 2 things. So, for instance, if I want to check whether a vertex j is a neighbour of vertex I, is there an edge from I to j, in the adjacency matrix, I just have to look at the cell i,j, and check whether it is 1 or not.

So, assuming that I can look up every entry in the matrix in constant time, in the same amount of time, then checking with as an edge between i and j is taking the same amount of time for every i and j. On the other hand, if I want to check in the adjacency lists matrix representation. Whether, say, for example, this is an edge from 8 to 9, then I have to go to 8. And then I have to walk down this list. In this case, I have to first check that the first entry is 5, and then I have to look at 9 and so, on. So, I have to go through the entire list for a given vertex to determine whether or not there is a neighbour. So, it is a little bit more expensive.

On the other hand, if I want to know all the neighbours of i, then the adjacency list directly gives it to us. So, if there are 5 neighbours, there are 5 entries anyway, I look at 5 entries, I will find them if there are 2 end neighbours, I will get 2 entries. On the other hand, in an adjacency matrix, regardless of how many real neighbours there are, you have to scan the entire row, because you do not know whether there is a 1 coming up or not.

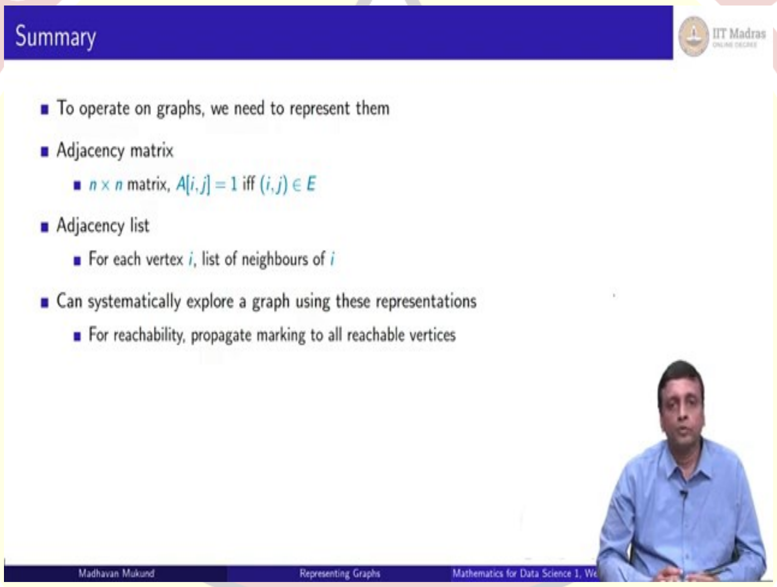
So, you cannot stop and say okay, after 7, I do not believe there are any more neighbours. So, you have to go. For example, if you started 8, I cannot go up to this point and say, Okay, I found 1 neighbour, and there are no more, you have to keep going because you might find a neighbour at last position.

So, regardless of what is the actual degree or out-degree, in this case, because it is a directed graph, regardless of the actual degree of the vertex, you have to spend order n time in order to collect all the neighbours of a given node, in an adjacency list, the time you take at each node is actually proportional to the degree. Now, in practice, many graphs will have a small degree, a given node will not be connected to very many other nodes.

So, therefore, if you have a procedure, which is proportional to degree rather than the number of nodes, it often works much faster. That is why it is useful to have this representation. So, there are trade-offs. So, it is not always the case that 1 is better than the other. So, typically, an adjacency list will save space. But there are some situations where it will incur some additional computation and vice versa. So, if you can make do with an adjacency matrix, and it is very simple to work with it, but then you have to do the scanning of rows and columns.

On the other hand, with a decency list, some things are not so convenient. For example, imagine how you would find out where there is an incoming edge in a directed graph, because an incoming edge will be recorded in the list for the other one, so, you have to go there and look at that list. So, there are other things that you have to do indirectly, with an adjacency list representation. But it is important to recognize that there are these 2 different ways of representing graphs and both of them are used in algorithms.

(Refer Slide Time: 22:15)



Summary

- To operate on graphs, we need to represent them
- Adjacency matrix
  - $n \times n$ matrix, $A[i,j] = 1$ iff $(i,j) \in E$
- Adjacency list
  - For each vertex i, list of neighbours of i
- Can systematically explore a graph using these representations
  - For reachability, propagate marking to all reachable vertices

So, to summarize, what we have seen is that, we cannot just think of a graph as a picture to describe a procedure on it, we need a representation. And we need a way of writing it down in a way that we can manipulate mathematically. And we came up with 2 different representations.

So, 1 is the adjacency matrix, which is a matrix of n cross n for n vertices, which says that A i j is 1 if there is an edge from i to j, otherwise, it is 0. And the other 1 which is more compact, in general, if we have not very many edges is what is called adjacency list, where for each vertex, we list out the neighbours of that vertex in the list.

And with either 1 of them, we did an example using the adjacency matrix, you can do now systematic computation. So, it is a systematic exploration of whether or not a vertex V is reachable from a vertex U. So, we will look at more such things, in particular, we will look at these 2 strategies which we described mentioned but did not do in detail which is breadth-first search and depth-first search and then we will also, look at other properties of graphs that you can compute using these 2 representations.