



IIT Madras

ONLINE DEGREE

Programming, Data Structures and Algorithms using Python
Professor Madhavan Mukund
Representing Graphs

So, we saw that graphs are going to be interesting objects to represent relations, and we have to do some computations on the graph. So, in order to compute on a graph, we need to represent the graph.

(Refer Slide Time: 00:19)

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, \dots, 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?

Airline routes


The diagram shows a directed graph with vertices v_0 through v_9 . The edges represent flight routes: $v_0 \rightarrow v_1$, $v_1 \rightarrow v_2$, $v_2 \rightarrow v_4$, $v_4 \rightarrow v_3$, $v_3 \rightarrow v_6$, $v_6 \rightarrow v_5$, $v_5 \rightarrow v_8$, $v_8 \rightarrow v_7$, $v_7 \rightarrow v_9$, and $v_9 \rightarrow v_0$.

So, to work with the graph, we need to transform this picture into something that we can manipulate in an algorithm. So, we said, for instance, we have these vertices and edges, and then we have paths and we have reachability. Now, if I look at this as a picture as a human being, I can kind of see that this thing is connected. Or maybe if there are edges, I may not be able to see, I may have to calculate that is connected.

Now, if it is a much larger graph, even visually, it may not be possible. And certainly, I cannot use this kind of visual analogy to write an algorithm. So, the algorithm has to represent this graph in a more kind of concrete way, and manipulate this representation in such a way that it can solve these problems without relying on this intuition of what the picture looks like. So, the picture is helpful for us. But it is not helpful for an algorithm.

(Refer Slide Time: 01:12)

Adjacency matrix



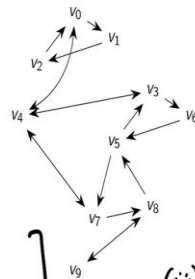
IIT Madras
BSc Degree


- Let $|V| = n$
 - Assume $V = \{0, 1, \dots, n-1\}$
 - 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 $\{0, 1, \dots, n-1\}$
 - $A[i, j] = 1$ if $(i, j) \in E$

Handwritten: $\begin{bmatrix} & i \\ i & \end{bmatrix}$

Handwritten: (i, j)

Airline routes





Madhavan Mukund

Representing Graphs

So, we need to represent this graph in some simple way that we can manipulate. So, the first thing is we need to represent the sets of underlying objects. So, let us start with the vertices. So, we saw that the vertices could be named in many different ways. So, in one of the early examples, we saw names of teachers and courses, so each vertex had a name of either a person or a course, we also saw another graph in which the names corresponded to friends in a group.

So, we could have many different ways of representing the vertices. So, we will choose typically a very simple representation, we will just say that we have n vertices that we will label them $0, 1, 2$ up to n minus 1 . So, we will always think of the vertex names as numbers between 0 and n minus 1 , when there are n vertices. So, essentially, we have to have some way of taking the original graph and mapping them.

So, here, for instance, we have a graph, we will call them V_0, V_1 up to V_9 , so I will replace this by 0 up to 9. So, now, if I look at an edge, an edge now, because vertices are all numbers between 0 and n minus 1, an edge is also a pair of numbers i comma j but, as we said before, we will not allow a reflexive relation in graph, so we will not allow an edge to start at i and end at i . So, when we say that i comma j is an edge, i and j must both be between 0 and n minus 1 that is in the allowed range of vertices, but i should not be equal to j .

So, with this representation, or this way of representing edges, and matrices, and vertices, the most obvious way to represent the graph as a whole is to record which edges are present and which edges are absent in a table, or a matrix called an adjacency matrix. So, we have rows and columns, 0 to $n - 1$, and we put an entry at i comma j . So, I have a matrix that looks

like this, so I put an entry at row i and column j , I put this to 1 if there is an edge from i to j . So, this is now a 0, 1 matrix. And the ones represent those edges which are present in my graph everywhere else, they will be zeros.

(Refer Slide Time: 03:25)

Adjacency matrix

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

edges = [(0,1), (0,4), (1,2), (2,0), (3,4), (3,6), (4,0), (4,3), (4,7), (5,3), (5,7), (6,5), (7,4), (7,8), (8,5), (8,9), (9,8)]

```
import numpy as np
A = np.zeros(shape=(10,10))
for (i,j) in edges:
    A[i,j] = 1
```

Airline routes

Diagram showing a directed graph with nodes v_0 through v_9 and edges connecting them. The nodes are arranged in a circular pattern with arrows indicating the direction of the edges.

Madhavan Mukund Representing Graphs

So, here, for instance, is how you would possibly do it in Python, if you use a NumPy array. So, first of all, the actual graph could be represented explicitly by a set of edges. So, here, for instance, if I as I said if we replace this by 0, and this by 1, and this by 2 and so on, we have an edge from 0 to 1, we have an edge from 0 to 4, we also have an edge from 4 to 0, which is a separate edge and so on.

So, the edge list here represents, in a literal way, the arrows which are drawn in the graph there. But we want to translate this into the 0, 1 matrix. So, what we do is we know that we have 0 to 9 in this case, so we create a 10 by 10, matrix 0 to 9, 0 to 9, initialize it to zeros. So, this is a NumPy initialisation, which has give me a matrix of all zeros of size 10 by 10. And now, we scan through this list. So, for every i comma j pair, which we see in this list of edges, we set the ij 'th entry of the matrix to 1, so we already initialized to 0, so wherever we did not find an edge, it is implicitly 0.

(Refer Slide Time: 04:36)

Adjacency matrix

■ Adjacency matrix

- Rows and columns numbered $\{0, 1, \dots, n-1\}$
- $A[i, j] = 1$ 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	0	1	0
5	0	0	0	1	0	0	0	1	0	0
6	0	0	0	0	0	0	1	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

Airline routes

So, if we do this, then we get a matrix which we can visualize like this. So, for instance, here it says that 3 comma 4 is 1, which means that in my graph, I should have an edge from 3 to 4, or it says that 7 comma 6 is 0. So, if I look at 7 and 6, I should not find an edge between them. So, the zeros represent edges which are not there. The ones represent edges which are there.

(Refer Slide Time: 05:01)

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


Airline routes, all routes bidirectional

So, if this is a undirected graph then remember that if i, j is an edge j, i will also be an edge. So, if I take the same graph that I had before. And I removed the directions, I get this undirected graph. Now, the matrix will be symmetric. So, if I see 0 comma 1, I must see 1 comma 0. So, if I take this diagonal here. If I take any entry, which is above the diagonal,

then if I go the same distance below the diagonal, I will find, so if I have 4 comma 7, I will have 7 comma 4. So, there is a kind of symmetry over this main diagonal. So, everything above and everything below the main diagonal, there will be a symmetric either it is both one or both zeros.

(Refer Slide Time: 05:43)

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, 5]

```
def neighbours(AMat, i):
    nbrs = []
    (rows, cols) = AMat.shape
    for j in range(cols):
        if AMat[i][j] == 1:
            nbrs.append(j)
    return(nbrs)
neighbours(A, 7)
[4, 5, 8]
```

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


So now, we need to manipulate in terms of this matrix. So, earlier, when we had this picture, if we wanted to go to the neighbour of a vertex, if we want to extend a path, for instance, then we just look at all the outgoing arrows and see which are all the neighbour, which are connected to outgoing arrows. So, in an adjacency matrix representation, this corresponds to looking along a row.

So, basically what we have is, say the neighbour of 6 are 3 and 5. This means that if I look at the rows 6, in my matrix, if I look at all the entries here, these are all entries of the form 6 comma j . So, some of these 6 comma j pairs are in my edge relation, some are not. So, it turns out that 6 comma 3 is there at 6 comma 5 is there. So, here is a simple Python function, which computes the neighbour of a given vertex and returns it as a list.

So, it takes an adjacency matrix, which is given and it takes i as this vertex for whom you want to find the neighbours. So, what you do is you initialize the list of neighbours to be empty. Now, you need to find out the size of this matrix. So, NumPy has this shape attribute, which tells you the number of rows and columns.

In this case, remember, it is going to be a square matrix, or rows and columns are going to be the same, we need either of them, we will use columns. So, because we are going to scan

through the columns, so for every j in the range columns, which is for every j from 0 to n minus 1. If I see that the vertex i , that I am looking at, has a neighbour j , then I append j to the list of neighbours, and I return it. And so, if I run this function on this particular example, if I run it on 7, for instance, if I start in row 7, then I will pick up 4, I will pick up 5, and I will pick up 8, so this should be 4, 5, and 8. So, this is what the function should return on this graph.

(Refer Slide Time: 07:41)

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, 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	0	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

Madhavan Mukund

Representing Graphs

So, if I have a directed graph, the rows represent outgoing edges. And these are not the same as the incoming edges, because not every outgoing edge has a matching incoming edge. So, the columns represent the incoming edges. So, if I look at row 6, then this entry represents an edge from 6 to 5. If I look at column 6, this represent an entry from 3 to 6, so from 3 there is an edge in to 6, and from 6, there is an edge out 5, and there is no correspondence between these two in general, they could be but they need not be.

(Refer Slide Time: 08:15)

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, 5]
- Directed graph
 - Rows represent outgoing edges
 - Columns represent incoming edges
- Degree of a vertex i
 - Number of edges incident on i
 $\text{degree}(6) = 2$

Undirected airline routes

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

So, the degree of a vertex, as you know is a number of edges which are incident on it. So, if I am just looking at an undirected graph, then I know that if you look at this thing, for instance, so the column 6. So, this has two ones, and this has two ones because if I have an edge from 6 to 3 and 6 to 5, I also have incoming edges from 3 to 6 and 3 to 5 because there are no directions. So, I just need to see how many edges are connected to six and I get the degree which is 2.

(Refer Slide Time: 08:42)

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, 5]
- Directed graph
 - Rows represent outgoing edges
 - Columns represent incoming edges
- Degree of a vertex i
 - Number of edges incident on i
 $\text{degree}(6) = 2$
 - For directed graphs, **outdegree** and **indegree**
 $\text{indegree}(6) = 1, \text{outdegree}(6) = 1$

Directed airline routes

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

Madhavan Mukund

Representing Graphs

But if I have not directed graph, this is no longer the same. So, I look at the outgoing degree. So, it says that the out-degree of 6 is 1 and it says the in degree of 6 is 1. So, there is one edge

coming in and one is going out, in general, these is not be the same. So, if you look at, for instance, let us see an example.

Supposing you look at 1 for example, overtake 0, it has two edges going out, it has 0 to 1 and 0 to 4. So, it has an out-degree of two, but if I look at the in-degree of here, oh it has 2 here also, it has 4 to 0 and 2 to 0. So, this is not anyway so you can have situations where you have unequal numbers of edges coming in and out. So, let me see if I can spot one here. So, I think in this particular graph, they are more or less all equal, but they need not be equal.

(Refer Slide Time: 09:41)

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

So now, one of the questions that we asked was reachability. So, we said given these edges, we want to find out if there is a path from one vertex to another vertex. So, concretely, if we interpret these as airline routes, we said that Delhi is city 0, Madurai is city 9. Is there a route from 0 to 0 from 9? So, the way we do this is a straightforward exploration, we start at 9 and we say, Okay, if we start at 9, certainly we can reach 9 from there because we have to do nothing.

Then we look at all the places we can reach from 9. So, where we can go from 9? Well, if we look at the row of 9, it says that I can reach 8. So, if I start from 9, then I can reach all its neighbours. In this case, it is 8. So, now I say that 8 is also reachable from 9. Now, I see where all I can go from 8. So, 8 can go to 5, 8 can go to 7, of course, 8 can come back to 9.

But that is where I started. So, that does not add any new information. So, I can now look at 5 and 7 and add them to my list, 9 is already there. But I have already, in some sense, I know that 9 is reachable, so there is no point in marking it again. So, then I pick these up in turn, so

I pick 5 and 7, for instance. And then I can extend my thing, saying that from 5, I can reach, for instance, 3 and 6.

And then from maybe 7, I can reach 4, then from 4, I can reach 0. And then I can stop in this case, because my target was to meet 0. So, I may be able to reach more things from 9, but I am not interested in knowing whether I can reach 1 or 2. So, this is the type of calculation that we want to do with the graph, checking reachability.

(Refer Slide Time: 11:24)

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	0	1	1	0	1	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

So, abstractly, we mark the sources reachable. And then we systematically mark neighbours, of marked nodes. So, we say that 9 is reachable. So, what are the neighbours of 9, 8, mark 8, what are the neighbours of 8, 5, and 7, mark them, and so on. And we stop when the target is marked. But the main thing that we need to do is recognize that we do not go back and get into a loop.

So, we do not want to go back and say that from 8, we know that 9 is a neighbour. So, let us go back and explore what we can reach from 9 because we already started with 9. So, if we go back from 9, then we will come back to 8, and then 8 will take us back to 9. And then we go round and round in circles without making progress. So, we need a systematic way of making sure that we do not get into this loop.

And as you should remember, there are two ways to do this, one is called breadth-first search, which does it layer by layer, how many things can reach in one step two steps, three steps, and so on. And then there is depth-first search, which keeps following a path until you get

stuck, then it comes back and takes an alternative path, and so on. So, we will look at these in detail.

(Refer Slide Time: 12:25)

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	[4,8]
8	[5,9]
9	[8]

Airline routes

Madhavan Mukund
Representing Graphs

So, what we are talking about now, right now is representing the matrix, representing the graph. So, we want to make sure that the graph that we are dealing with is not a picture but something that our algorithm can manipulate. So, we came up with this straightforward representation, which was this adjacency matrix, which had zeros everywhere, except where there are edges and these are ones.


Now, as you can imagine, very often, they will be mostly zeros and very few ones. So, we could look for a more compact representation. And this is what is called an adjacency list. So, notice that if I have in an undirected graph, if I have n vertices, then each of these can be connected to n minus 1 others and then they are symmetric, so we do not count them. So, n into n minus 1 by 2 is the number of edges that I could have. So, if I have n vertices, I have order n squared edges possible, but very often I do not.

Similarly, in a directed graph, that factor of two goes away, we have at most n into n minus 1 different edge. But in many realistic situations, the number of edges is not n squared is not order of n squared, but it is closer to order of n . So, in such situations, most of my matrix will be zeros. So, here, a better representation is what is called an adjacency list. That is, for every vertex, I explicitly record instead of that whole sequence of zeros and ones in the matrix, I only record the ones, I only record those vertices that are connected.

So, here I say that 0, vertex 0 is connected to 1 and 4, I do not have to put the other zeros we say is not connected to 2, is not connected to 3, is not connected to 5, and so on. So, this is the adjacency list representation. So, for every vertex, it just gives you the list. So, it says 8 is connected to 5, and to 9, we do not have to say it is not connected to 7, is not connected to 0, is not connected to 6, and so on.

(Refer Slide Time: 14:15)

Adjacency lists



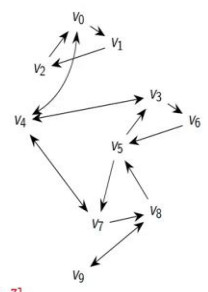
- Adjacency matrix has many 0's
 - Size is n^2 , regardless of number of edges
 - Undirected graph: $|E| \leq n(n-1)/2$
 - Directed graph: $|E| \leq n(n-1)$
 - Typically $|E|$ much less than n^2
- Adjacency list


```

AList = {}
for i in range(10):
    AList[i] = []
for (i,j) in edges:
    AList[i].append(j)
print(AList)


{0: [1, 4], 1: [2], 2: [0], 3: [4, 6], 4: [0, 3, 7],
5: [3, 7], 6: [5], 7: [4, 8], 8: [5, 9], 9: [8]}
```

Airline routes



Madhavan Mukund

Representing Graphs



So, the simplest way to build up such a list in Python is to use a dictionary. So, we use the names of these vertices, in this case, 0 to n minus 1, 0 to 9 as the keys. And then for every edge that is there in my edge list, I just append it to this. So, if I assume that the edges are given an ascending order, then I will get these adjacency lists also in ascending order. So, very often convenient to assume that the lists are enumerated the same way that the columns are enumerated.

So, if I have an edge from say, 8 to 5, and 8 to 9, then the adjacency list of 8 will be 5 comma 9, it is not 9 comma 5. So, if I construct the adjacency list for this graph, for instance, it shows this dictionary. So, 0 is connected to 1 and 4, 1 is connected to 2, and so on. So, for example, 8 is connected to 5 and 9. So, this is a more compact representation in general, especially if the number of edges is small with respect to n square.

(Refer Slide Time: 15:13)

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

	0	1	2	3	4	5	6	7	8	9
0	0	0	1	0	0	1	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]
1	[2]
2	[0]
3	[4,6]
4	[0,3,7]

5	[3,7]
6	[5]
7	[4,8]
8	[5,9]
9	[8]



Madhavan Mukund

Representing Graphs

So, one of the advantages that an adjacency list requires less space. Now, there are some disadvantages. So, if I want to check whether i is connected to j . Now, I want to check whether i is connected to j in an adjacency matrix, I just have to look at the ij 'th row. So, supposing I want to know whether 5 is connected to 7, then I just have to look at the entry 5 comma 7 and say, yes, 5 is connected to 7.

On the other hand, if I want to check in the adjacency list with a 5 is connected to 7, I have to take this list and walk down the entire list. So, this, in this case, it is a small graph. So, it is a small list. But in general, you can imagine that takes more time because you have to look at that entire list and go from one end to the other and find it.

(Refer Slide Time: 15:58)

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

	0	1	2	3	4	5	6	7	8	9
0	0	0	1	0	0	1	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]
1	[2]
2	[0]
3	[4,6]
4	[0,3,7]

5	[3,7]
6	[5]
7	[4,8]
8	[5,9]
9	[8]



Madhavan Mukund

Representing Graphs

On the other hand, if I want to add together all the neighbours, then it is more efficient use the list because the non-neighbours are absent. So, if I have a small number of neighbours, I will pick them up from the list, but I have a large number of neighbours also I will need to anyway scan them. But in an adjacency matrix, there is no way to find the neighbours except to scan all these n minus 1 entries in the columns.

So, even if I have only one neighbour, I will not know it without looking at all the entries. So, in an adjacency matrix, finding all the neighbours always takes time proportional to the number of vertices, there is an adjacency list, it takes time proportional to the degree or more precisely if you are doing a directed graph the out-degree. So, depending on what we are planning to do one representation or the other can be more efficient. Very often, we will see that adjacency list is better than adjacency matrix for the kinds of problems that we are looking at.

(Refer Slide Time: 16:52)

Summary

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

Madhavan Mukund Representing Graphs

So, to summarize, what we said is that we cannot just think of graphs as pictures, and then try to compute on them because this picture cannot be read by our algorithm. So, we need to represent this graph in some concrete form. So, we have two representations that we propose one is 0, 1 matrix, this adjacency matrix, where i, j represents A_{ij} represents the edge from i to j , so it is 1 if there is an edge, 0 if there is no edge, so I canonically change my vertex names from whatever they are to 0 to n minus 1.

And then an adjacency list, think of it as a dictionary, where the keys are 0 to n minus 1. And each key is attached to a list the list of vertices which are connected to vertex. So, if I look up A , A square bracket i in an adjacency list, it gives me a list of values, precisely those

neighbours. So, if I want to scan this list, it is only proportional to the degree of that vertex. So, with both of these, we can systematically explore the graph. So, we saw that we want to do the reachability kind of analysis and we have to do it, start into a graph and mark all the vertices are reachable. So, we will now describe how to do these computations using these representations.

