

**Design and Analysis of Algorithms, Chennai Mathematical Institute**  
**Prof. Madhavan Mukund**  
**Department of Computer Science and Engineering,**

**Module – 02**  
**Lecture - 19**  
**Representing Graphs**

So, we have seen that graphs are very useful mathematical structures for modeling problems. But, when we write an algorithm to solve a graph theoretic problem, we need

a way to represent and manipulate the graph in our algorithm. So, we will look at that in this lecture.

(Refer Slide T

**Graphs, formally**

$G = (V, E)$

- Set of vertices  $V$
- Set of edges  $E$ 
  - $E$  is a subset of pairs  $(v, v')$ :  $E \subseteq V \times V$
  - Undirected graph:  $(v, v')$  and  $(v', v)$  are the same edge
  - Directed graph:
    - $(v, v')$  is an edge from  $v$  to  $v'$
    - Does not guarantee that  $(v', v)$  is also an edge

The slide includes two hand-drawn diagrams. The first diagram shows two vertices,  $v$  and  $v'$ , connected by a simple horizontal line, representing an undirected edge. The second diagram shows the same two vertices, but with a curved red arrow pointing from  $v$  to  $v'$  and a straight black arrow pointing from  $v'$  to  $v$ , representing a directed edge. The pair  $(v', v)$  is written in red next to the curved arrow, and the pair  $(v, v')$  is written in black next to the straight arrow.

So, recall that a graph is a set of vertices or nodes  $V$  connected by a set of edges  $E$ . We can have two types of edges, undirected edges and directed edges. An undirected edge is drawn as just a line between two vertices and it represents the fact that  $v$  and  $v$  prime are connected. It does not matter whether we call this edge  $v$  comma  $v$  prime or  $v$  prime comma  $v$ , there is only one edge.

On the other hand in a directed graph, we actually associated direction with an edge. So, we could draw an edge from  $v$  to  $v$  prime and this we would write in our edge set as a pair  $v$  comma  $v$  primes saying that the start vertex is  $v$  and the end of the edge is  $v$  prime. And this is not the same as having an edge from  $v$  prime to  $v$  which will be written as  $v$  prime comma  $v$ .

So, directed graphs the order of the vertices when you mention the edge matters, then the

undirected graph is just the pair of vertices, it does not matter whether we think of it, it is  $v$  to  $v$  prime or  $v$  prime to  $v$ , it is just a connection between these two pairs of vertices.

(Refer Slide Time: 01:22)

### Finding a route

- Directed graph
- Find a sequence of vertices  $v_0, v_1, \dots, v_k$  such that
  - $v_0$  is New Delhi
  - Each  $(v_i, v_{i+1})$  is an edge in  $E$
  - $v_k$  is Trivandrum

So, we saw two typical problems involving finding a route, which can be represented for both undirected and directed graphs. So, in a directed graph, we would looking for path from  $v_0$  to  $v_5$ , such that each pair  $v_0, v_1$ ;  $v_1, v_2$  etcetera, these are directed edges in our graph.

(Refer Slide Time: 10:40)

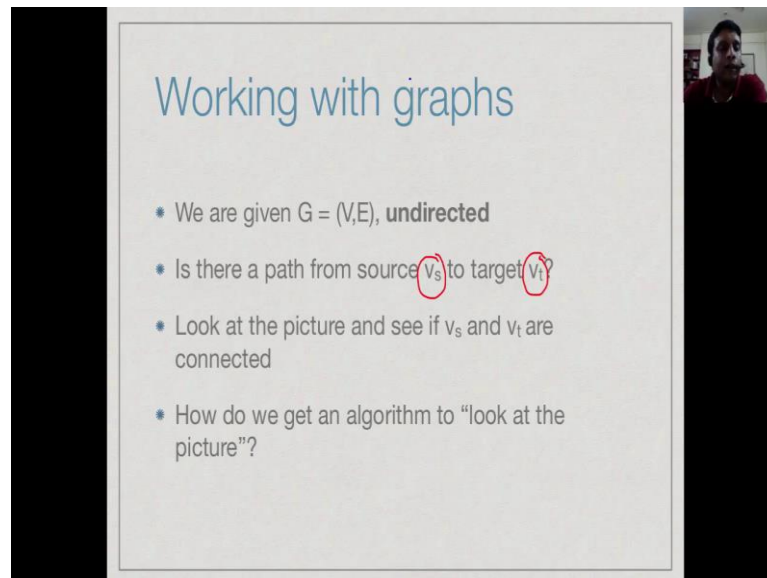
### Finding a route

- Also makes sense for undirected graphs
- Find a sequence of vertices  $v_0, v_1, \dots, v_k$  such that
  - $v_0$  is New Delhi
  - Each  $(v_i, v_{i+1})$  is an edge in  $E$
  - $v_k$  is Trivandrum

At the same time, we can make the same graph undirected and again we are looking for a path, where we start at  $v_0$  and every adjacent pair  $v_0$  to  $v_1$ ,  $v_1$  to  $v_2$  is an edge, such

that we finally end up in the target vertex  $v_5$ .

(Refer Slide Time: 01:53)



Working with graphs

- We are given  $G = (V, E)$ , **undirected**
- Is there a path from source  $v_s$  to target  $v_t$ ?
- Look at the picture and see if  $v_s$  and  $v_t$  are connected
- How do we get an algorithm to “look at the picture”?

So, the problem that we have abstractly, let us stick to undirected graphs, since that we are given at this point an undirected graph and we are given a source vertex  $v_s$  and a target vertex  $v_t$  and we asked, whether there is a way to go from  $v_s$  to  $v_t$  in this graph. Now, what we did in the previous graph and what we can do as a human beings is to take a look at the graph and see, if you can visually identify such a path, just see  $v_s$  and  $v_t$  are connected.

But, when we write an algorithm to manipulate a graph, how do we get the algorithm to look at the picture. So, for us a graph is a picture and we can easily look at the picture, if the graph is not very complicated and try to understand the situation. But, how do we make this picture available to an algorithm? We need a way to represent this picture that gives the graph as input to our algorithm.

(Refer Slide Time: 02:42)

## Representing graphs

- Let  $V$  have  $n$  vertices
  - We can assume vertices are named  $1, 2, \dots, n$
  - Each edge is now a pair  $(i, j)$ , where  $1 \leq i, j \leq n$
  - Let  $A(i, j) = 1$  if  $(i, j)$  is an edge and 0 otherwise
  - $A$  is an  $n \times n$  matrix describing the graph
- **Adjacency matrix**

So, let us make some assumptions, in any graph that we consider, there is only be a finite set of vertices. So, if there are  $n$  vertices to simplify life, let us just name these vertices 1, 2 up to  $n$ . So, our vertices are always going to be 1, 2, 3, 4 up to  $n$ . So, therefore now an edge is a pair of numbers  $i$  comma  $j$ . So, the first representation we can have is to just record which pairs  $i$  and  $j$  are connected. So, this is called an adjacency matrix, we say in this matrix  $A_{ij}$  is 1, if and only if  $i, j$  is an edge.

(Refer Slide Time: 03:24)

## Adjacency matrix

	1	2	3	4	5	6	7	8	9	10
1	1	1	1	1	1	0	0	0	0	0
2	1	1	1	1	0	0	0	0	0	0
3	1	1	1	1	0	0	0	0	0	0
4	1	1	1	1	1	1	0	1	0	0
5	1	0	0	1	1	0	1	1	0	0
6	0	0	0	1	1	1	1	1	1	0
7	0	0	0	1	1	1	1	0	0	0
8	0	0	0	1	0	1	0	1	0	0
9	0	0	0	0	0	1	0	1	1	0
10	0	0	0	0	0	0	0	1	0	1

So, when we write such a matrix, then if we take a graph like we had before, we would now rename the vertices 1 to 3 up to 10, because there are actually 10 vertices in this graph and then we would write this matrix which says for instant there is an edge from 1



to 3 and therefore, the entry  $A_{1,3}$  is 1. There is no edge from 1 to 5 and therefore, the entry  $A_{1,5}$  is 0.

So, in this way for every edge that we find in the graphs, say for example,  $A_{4,5}$ , we will find an entry in the graph which says  $A_{4,5}$  is 1. Now, remember that this is undirected. So, if there is an edge  $5,4$ , there is also an edge  $4,5$  and so we will see that actually there is a matching edge  $5,4$ . So, this graph is actually this symmetry, so it is actually to be symmetric across this diagonal. So, if I see a 1 above the line, I will see a 1 below the line, because  $8,9$  is as same as  $9,8$  as for as our undirected edges. So, this is an adjacency matrix.

(Refer Slide Time: 04:20)

**Adjacency matrix**

- Neighbours of  $i$ 
  - Any column  $j$  in row  $i$  with entry 1
  - Scan row  $i$  from left to right to identify all neighbours
- Neighbours of 4 are  $\{1, 5, 8\}$

	1	2	3	4	5	6	7	8	9	10
1	1	0	1	1	1	0	0	0	0	0
2	1	1	0	1	0	0	0	0	0	0
3	1	1	1	0	0	0	0	0	0	0
4	1	0	0	1	0	0	1	0	0	0
5	0	0	0	1	1	0	1	1	0	0
6	0	0	0	0	1	0	1	1	1	0
7	0	0	0	0	1	1	0	0	0	0
8	0	0	0	1	0	1	0	0	1	0
9	0	0	0	0	0	1	0	1	0	1
10	0	0	0	0	0	0	0	0	1	0

So, now what can we do with the adjacency matrix? Well, one thing we can do for example is find all the neighbors all of a vertex. Suppose, if we want the neighbors of a vertex  $i$ , then we want to look at the row  $i$  and look at all the entries 1 in that row. For example, we want to look at the neighbors of vertex 4, we look at the row 4, then 4, the entry 4 comma 1 indicates whether or not 4 1 is an edge. It is, so we get 1 of our vertex, then we walk for the... And then the next one is a 4 comma 5, so 5 is a neighbor.

And then, the next one is an 8, so 8 is a neighbor, so in order to find the neighbors of a vertex  $i$ , we go to the row labeled  $i$  and we scan the row from left to right and each one that we find, the corresponding column is a neighbor.

(Refer Slide Time: 05:09)

**Finding a path**

- Start with  $v_s$
- New Delhi is 1
- Mark each neighbour as reachable
- Explore neighbours of marked vertices
- Check if target is marked
- $v_t = 10$  = Trivandrum

	1	2	3	4	5	6	7	8	9	10
1	0	1	1	1	0	0	0	0	0	0
2	1	0	1	0	0	0	0	0	0	0
3	1	1	0	0	0	0	0	0	0	0
4	1	0	0	0	1	0	0	1	0	0
5	0	0	0	1	0	1	1	0	0	0
6	0	0	0	0	1	0	1	1	1	0
7	0	0	0	0	1	1	0	0	0	0
8	0	0	0	1	0	1	0	0	1	0
9	0	0	0	0	0	1	0	1	0	1
10	0	0	0	0	0	0	0	0	1	0

Now, how do we find the path? Then, we can now look at neighbors and the neighbors of neighbors and so on. So, we start with the source vertex in our problem, if you look at the numbering, let me give, New Delhi corresponded to the vertex labeled 1 and Trivandrum was the labeled vertex labeled 10. And we wanted to know, whether we can get from vertex 1 New Delhi to vertex 10 Trivandrum.

So, what we do is, we start at vertex 1 and we know how to find the neighbors of 1. So, we will say that every vertex any neighbor of 1 can be reached from one by one. So, we start with the vertex 1 which we have already started, we can reach, because we start there. Now, we scan it is neighbors and then we find that there these 3 neighbors, 2, 3 and 4. So, therefore now we can conclude that if you start at 1, we can definitely reach 2, 3 and 4.

So, let us color these rows also green, so now we have indicated that in one step from 1, we can reach 2, 3 and 4. Now, anything that we can reach from 2, 3 and 4 can also be reached from 1. So, we now focus one by one to the neighbors that we have already explored and look at their neighbors. So, we moved down to the neighbors of 2. So, 2 has only 2 neighbors, 1 and 3 and it turns out that 1 and 3 have already been marked as being reachable from 1, we do nothing about this.

So, the cities reachable from 2 do not add any information to our problem. So, you move on to 3, likewise 3 can reach 1 and 2, both of which I have already been listed as visited or which can be reached from 1. So, we can again skip over to 4, because 3 has nothing

new to tell us. Now, when we reach vertex 4, it has something new to tell us, because now from 4, it says you can reach 1 of course, we will know, then we can also reach 5 and 8.

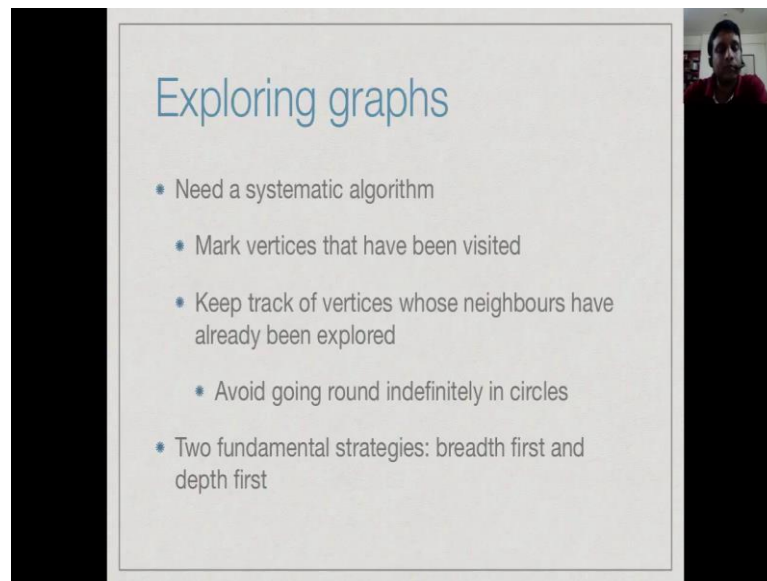
So, now we will now, in our representation we will indicate that 5 and 8 are colored green to indicate that these are also reachable from 1 by an indirect path, I can go from 1 to 4 and then from 4 to 5 and 8. So, I have already processed 1, 2, 3 and 4, so the next I look at for vertex 5 to see 5 can give us any new neighbor. So, you look at 5 and now 5 has a neighbor we have already seen 4, but it has a new neighbor 6 and other new neighbor 7, so if I process 5, I get 6 and 7.

Now, I can look at any of 6, 7 or 8, so let us look at 8. So, 8 has neighbors 4, which I have already seen, 6 which I have already seen, but it has new neighbor 9, which I have not seen. So, I add 9 to the list of neighbors which can be reached eventually from 1. And now if I look at 6, I find that 6 can reach 5, 7, 8 and 9; all of which are already known to be neighbors are reachable from 1.

So, I do not have any new information, likewise when I go to 7, I can reach 5 and 6, which I have already know are reachable. So, there is no new information. And finally, the vertex which are not yet examined is 9, so from 9 I can reach 6 which I know, 8 which I know and there is a new vertex 10, so I can reach 10 from 9, so I mark 10 and once I marked 10, my problem is solved. I have found that there is a way to go from 1 to 10, but systematically expanding the set of neighbors I can reach one level at a time.

So, this give us some algorithm, we will make this algorithm more precise as we go long, but you can see that using this representation as an adjusting C matrix, we can take this matrix and use it to actually explore the problem that we have in hand and make it into assign as a reasonable procedure which we can execute effectively.

(Refer Slide Time: 08:50)



## Exploring graphs

- Need a systematic algorithm
  - Mark vertices that have been visited
  - Keep track of vertices whose neighbours have already been explored
  - Avoid going round indefinitely in circles
- Two fundamental strategies: breadth first and depth first

So, to make this algorithm more precise, we need to make this algorithm more systematic, we have to keep track of the vertices which have been visited, so that we do not keep exploring the same vertex again. So, we do not want to keep going around in circle and doing the same problem again and again. So, it will turn out that there are two fundamental strategies to solve this particular problem which is one of the most basic problem in graphs, which will find out, what is connected to what.

So, the strategy that we executed is what is called breadth first, that is we first explore all the neighbors of the starting vertex from all the neighbors of these things which are one step away, all the neighbors that thing that two step away and so on. The other strategy to go as far as possible in one direction, so you pick one neighbor at starting vertex, then you pick one neighbor of new vertex, then you pick one neighbor of that new vertex and so on. And we cannot find a new vertex, then you go back and explore and other neighbors of previous vertices and so on and this is called breadth first. So, we will see these algorithms in detail in a later lecture.

(Refer Slide Time: 09:50)

### An alternative representation

- Adjacency matrix has many 0's
- Size of the matrix is  $n^2$  regardless of number of edges
- Maximum size of E is  $n(n-1)/2$  if we disallow self loops
- Typically E is much smaller

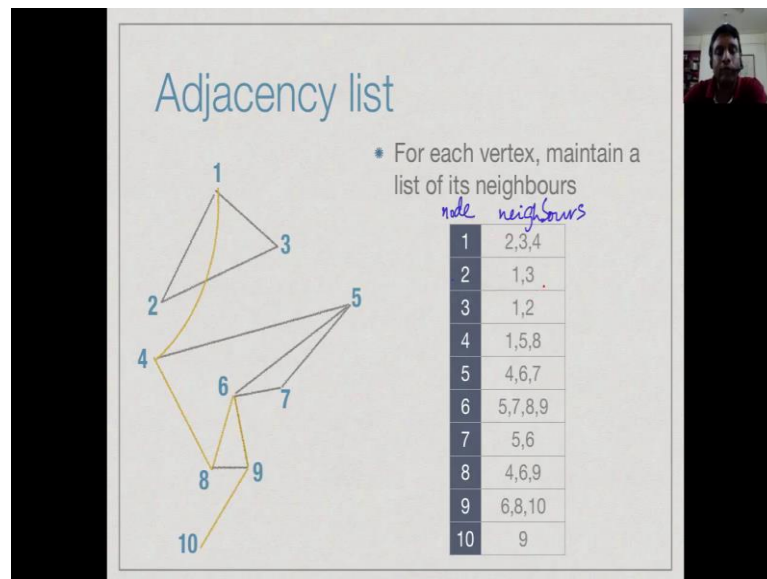
	1	2	3	4	5	6	7	8	9	10
1	0	1	1	1	0	0	0	0	0	0
2	1	0	1	0	0	0	0	0	0	0
3	1	1	0	0	0	0	0	0	0	0
4	1	0	0	0	1	0	0	1	0	0
5	0	0	0	1	0	1	1	0	0	0
6	0	0	0	0	1	0	1	1	1	0
7	0	0	0	1	1	0	0	0	0	0
8	0	0	0	0	1	1	0	0	1	0
9	0	0	0	0	0	1	0	1	0	1
10	0	0	0	0	0	0	0	0	1	0

So, one of the thing that you can observe is that most of the entries in this matrix are actually 0. So, remember that if you have  $n$  vertices, this matrixes size  $n$  square, because I have  $n$  rows and  $n$  columns. Now, if you count the number of the edges in an undirected graph, then each pair of vertices can be an edge, we normally disallows self loops, we normally do not consider edges from  $i$  to  $i$  for vertex  $i$ . And the number of different pairs is  $n$  choose 2, these are how many ways you can pick two vertices.

So, you pick all the pairs, then it is  $n$  into  $n$  minus 1 by 2, so this is the basic common oriented fact. So, we could have about  $n$  square edges no more and if you have heard about  $n$  square edges, then many of the entries in the matrix would be 1, but it most situations the number of edges is much less than  $n$  square. So, this is the wasteful representation in some sense, because we are recording a lot of 0's in order to capture the positive information which is in the 1's.

And also remember that these 1's are symmetric, so for every 1 about the line I have, so if I draw this diagonal, that every 1 above the diagonal, I have a symmetric 1 below the diagonal. So, actually half of this matrix are useless, if I just know this top half is enough and in this top half I will have mostly 0. So, we can think of another representation, where we only keep the relevant information at hand.

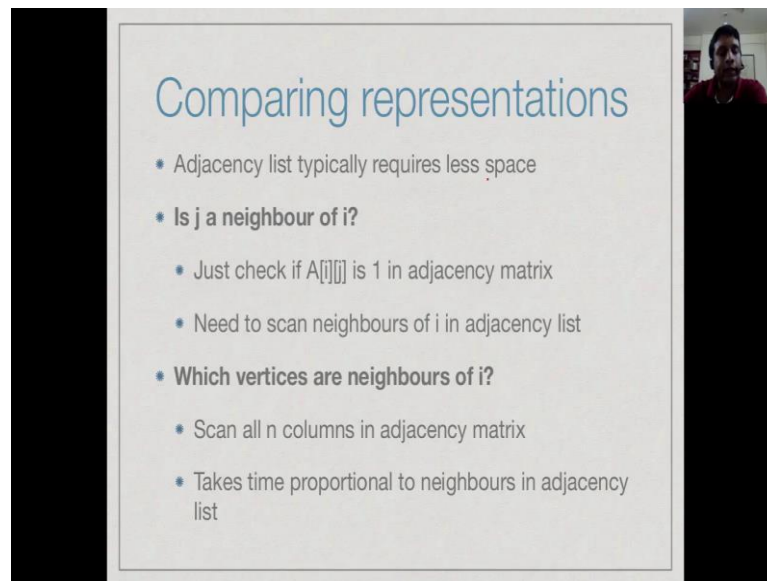
(Refer Slide Time: 11:20)



So, this is what is called an adjacency list, so what we do in an adjacency list is that we explicitly maintain for every vertex, the list of its neighbors. So, we see that one is connected to 2, 3 and 4. So, we say that the list connected to the 1 is 2, 3, 4. Similarly, 2 is connected 1 and 3, so we have the list connected to is 1 and 3, so this is the node and these are the neighbors.

So, now, in this representation, whenever I have node, I just go to that entry for that node and I look up the list and I can scan it neighbors and I can get it in time proportional to the number of neighbors of that. So, there is an advantage that I do not want any useless information, because I am not storing all the 0 is adjacency matrix here, I am not keeping the norm information which vertex not connect.

(Refer Slide Time: 12:12)



### Comparing representations

- Adjacency list typically requires less space
- Is  $j$  a neighbour of  $i$ ?
  - Just check if  $A[i][j]$  is 1 in adjacency matrix
  - Need to scan neighbours of  $i$  in adjacency list
- Which vertices are neighbours of  $i$ ?
  - Scan all  $n$  columns in adjacency matrix
  - Takes time proportional to neighbours in adjacency list

So, these two representation have their advantages and disadvantages, in the adjacency matrix, we need much more space and then, adjacency list, but some questions are easier to answer an adjacency matrix than in the adjacency list. If you want to know, whether vertex  $j$  is a neighbor of vertex  $i$ , we just have to prove one entries in the matrix, we just check it if  $A[i][j]$  is 1.

On the other hand in an adjacency list, if you want to find out  $j$  is a neighbor of  $i$ , we need to go to the entry for  $i$  and scan the entire list. So, this is similar to the original discussion we had in the beginning of the sorting module on the difference between matrix and this. So, here we can probe this entry  $A[i][j]$  in unit time, whereas we need propositional to the number of neighbors of  $i$ , to find out whether or adjacent neighbor of  $i$  or adjacency list.

On the other hand, if you want actually find all the neighbors of  $i$ , then regardless of how many neighbours of needed in the adjacency matrix, we have to scan the entire row. So, there are  $n$  vertex in the graph, when even if a node had only 2 or 3 neighbors we will have to look at all  $n$  entries in the row to determined which of these  $n$  entries are the actual neighbors. On the other hand, in an adjacency list as we have seen, we record exactly those but which a neighbors. So, time to scan the neighbors is directly prepositional to neighbors. So, if each vertex only a small number of neighbors, then the adjacency list to quickly give as those neighbors, whereas adjacency matrix you required scan order  $n$  entries to find out the small number.