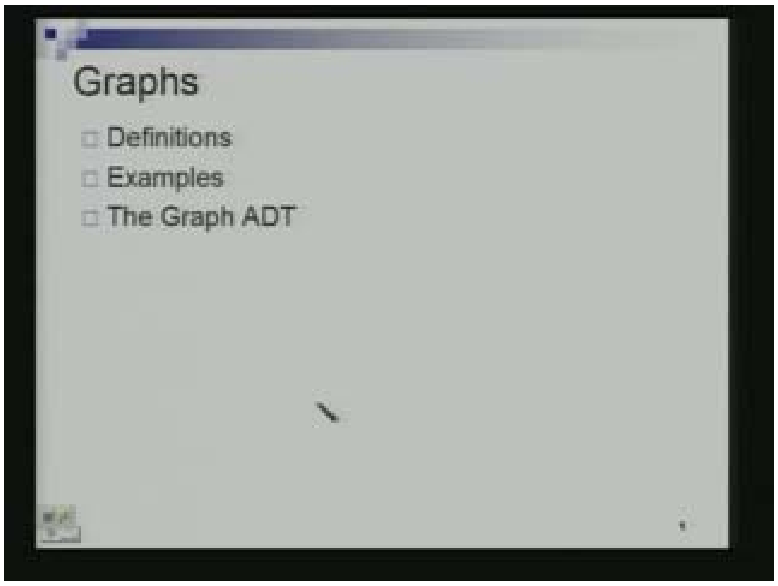
Data Structures and Algorithms

Dr. Naveen Garg
Department of Computer Science and Engineering
Indian Institute of Technology, Delhi

Lecture – 24
Graphs

Today we are going to start talking about graphs. We are going to spend quite a lot of time understanding the basic definition in terminologies associated with graphs, see some examples and then if time permits we are going to do the graph abstract data type or I think we will able to do the graph data type today.

(Refer Slide Time: 01:30)

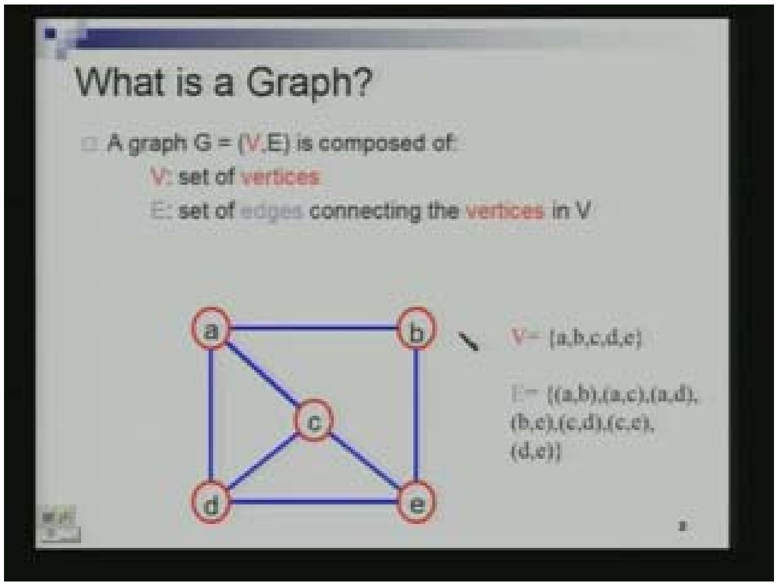


Graphs

- Definitions
- Examples
- The Graph ADT

So question is what is a graph? So pictorially this is what a graph is and what are terms we are going to have. So graph is always represented by a two tuple V and E typically, V's what we will call the set of vertices and E will call the set of edges. So set of vertices and a set of edges together specify a graph. In this picture these red circles are the vertices. I have given each of these vertices a name a b c d e to distinguish them and the blue lines are the edges, so edge really is a pair of vertices. An edge is a pair of vertices or an edge is specified by giving a pair of vertices so this edge is said to connect what is u and v or will not use the term connect but this edge is an edge between u and v; when I say e = u v is an edge then that means it's an edge between vertices u and vertex v, vertices u and v.

(Refer Slide Time: 2:08)



What is a Graph?

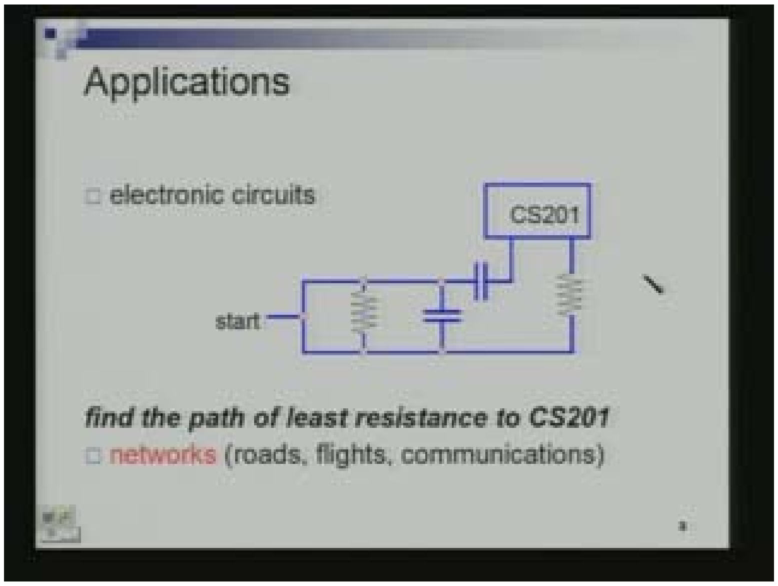
- A graph G = (V,E) is composed of:
  - V: set of vertices
  - E: set of edges connecting the vertices in V

V= {a,b,c,d,e}

E= {(a,b),(a,c),(a,d),
(b,e),(c,d),(c,e),
(d,e)}

So for instance in this example this graph could be specified either by giving this drawing or giving these this detail. As in v the set of vertices is 5 vertices a b c d e and what are the edges I have? Each edge as you can see is a pair of vertices, an unordered pair of vertices here, a comma b is the same as b comma a. All that specifies is it is an edge between vertices a and b. So a comma b, a comma c is this edge; a comma d is this edge, b comma e is this, c comma d is that, c comma e is this and d comma e is this. So there are 1, 2, 3, 4, 5, 6, 7 edges and there are the 7 pairs mentioned here. So set of vertices and a set of edges. What are they used for? They are for lots and lot of applications, you can model circuits as graphs, each of component of the circuit could be a vertex.

(Refer Slide Time: 04:18)



Applications

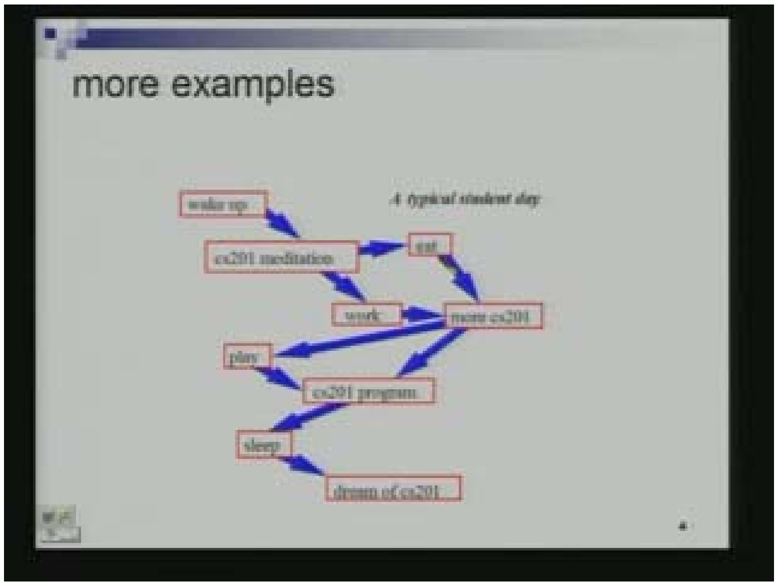
- electronic circuits

find the path of least resistance to CS201

- networks (roads, flights, communications)

So this could be a vertex, this could be a vertex, this could be a vertex, this could be a vertex, this is a vertex which is your CS201, you are trying to find out the path of these resistance to get CS201, they can be used to model networks. So I can take the map of the city and every intersection could be modeled as a vertex and the roads which are connecting to intersections could be modeled as an edge and then that could be a graph and then start asking various questions on whether how can I go from this place to this place by asking the corresponding question on a graph. So transportation networks, lots of this communication networks all of them are modeled as graphs.

(Refer Slide Time: 04:53)



more examples

One more example. So this is typically student day so you wake up, you mediate first 201 then you eat, may be you work then more CS201, play CS201 programming sleep and you dream of CS201 cycles. [Student: so idealistic] There is no room for any other course. This is the day before mine. So this is slightly different from the graph that I had shown in the previous example. Why [student: directed] directed. So this is what we call directed graph because we can't do any meditation before you wake up. So there is clearly an edge going from wake up to meditation. So every edge has a direction associated with it, we will call such graphs directed graphs. So we also consider directed graphs but in the rest of the lecture I am going to spend most of time with undirected graphs. Whatever things I define will carry over in a straight forward way to directed graphs as well so I will tell you what the difference is.

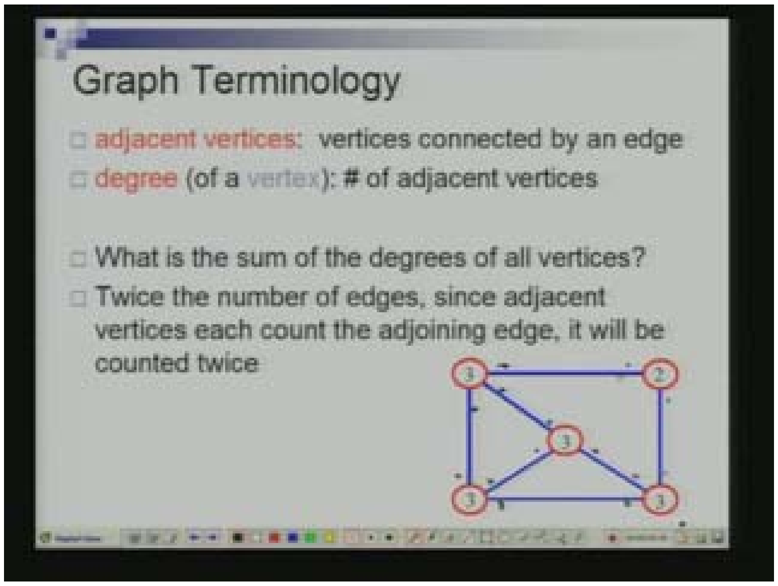
So to begin with let me go back to the previous slide. In this example or in this definition where would the difference be when I am talking of a directed graph? So e comma u v is not just a pair, it is an ordered pair let's say. So the ordering is important, the first vertex typically specify what the start of the edges is or the origin of the edge and the other would specify the destination of the edge where the edge is going from, so what is the start and what is the end.

So as I said today is a fairly simple lecture, we are going to look at lots of terminologies. So now you have understood what a graph is. So there are two kinds of graphs a directed graph and an undirected graph. So graph which is not directed is called undirected graph and you understand what a vertex is, what vertices are, what edges are. Adjacent vertices, so two vertices so this is all terminologies associated with an undirected graph so two vertices which are connected by an edge are called adjacent. Is this vertex and this vertex, these two vertices are they adjacent? No, they are not connected by an edge while this and this are adjacent and this and this are not adjacent either. So what is it which are connected by an edge are called vertices, the degree of a vertex.

The degree of the vertex is the number of adjacent vertices it has. So what is the degree of this vertex? 3. So in fact I have written down the degrees of the various vertices on these so this vertex is degree is 2, this vertex is degree 3, this is degree 3, this is degree 3, everyone understands the degree of the vertex. It is the number of adjacent vertices. Sometimes we say that this edge is incident to these two vertices. Should I write down the word? So this edge, let’s say this vertex is vertex a and vertex b and this edge is e, so e equals a b is incident to vertices a and b. So this edge is incident to these two vertices similarly this edge is incident into this vertex as well as this vertex. So degree of a vertex can also be defined as the number of edges which are incident to that vertex. There are three edges which are incident to this vertex, so the degree of this vertex is 3. These are equivalent ways of saying the same thing. So question is what is the sum of the degrees of all the vertices, [Hindi Conversation] twice the number of edges. Because when I am counting, so let’s think of it in the following manner. So the answer is right, twice the number of edges and the argument is actually half a line of an argument.

So pictorially I would say the following; when I am counting three for this, I am counting three because I am counting this one edge, this edge and this edge. So let me put 3 stones, one on each of these three edges then when I am counting 3 here I am counting this edge, this edge let me put down 3 stones. Then here I am putting down 2 stones, here I am putting down 3 stones, here I am putting down 3 stones.

(Refer Slide Time: 10:50)

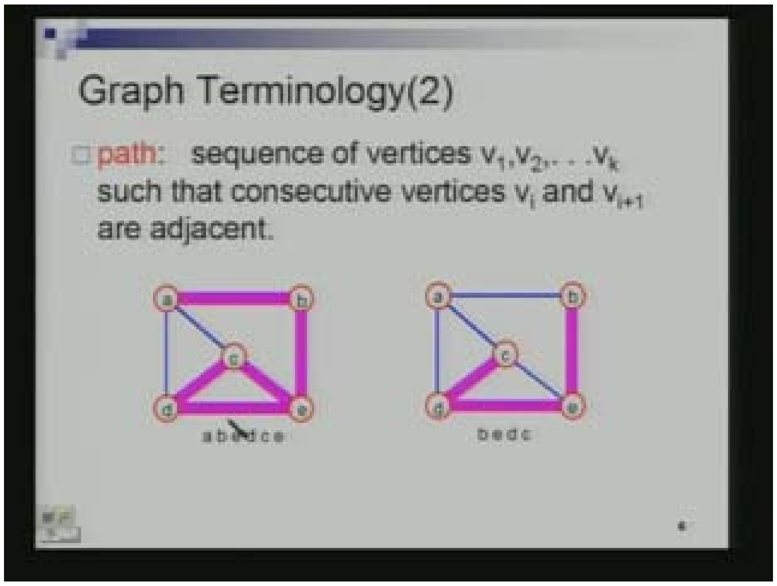


Graph Terminology

- adjacent vertices: vertices connected by an edge
- degree (of a vertex): # of adjacent vertices
- What is the sum of the degrees of all vertices?
- Twice the number of edges, since adjacent vertices each count the adjoining edge, it will be counted twice

So I have to put as many stones or pebbles, if you want as many peppules as the sum of the degrees of the vertices. Now if I look at any edge, how many pebbles are there on that edge? Exactly 2, so the sum of the degrees of the vertices equals two times the number of edges. So that's degree and you understand what degree is, you understand what adjacent vertices are. Now let's define the notion of a path.

(Refer Slide Time: 10:57)



Graph Terminology(2)

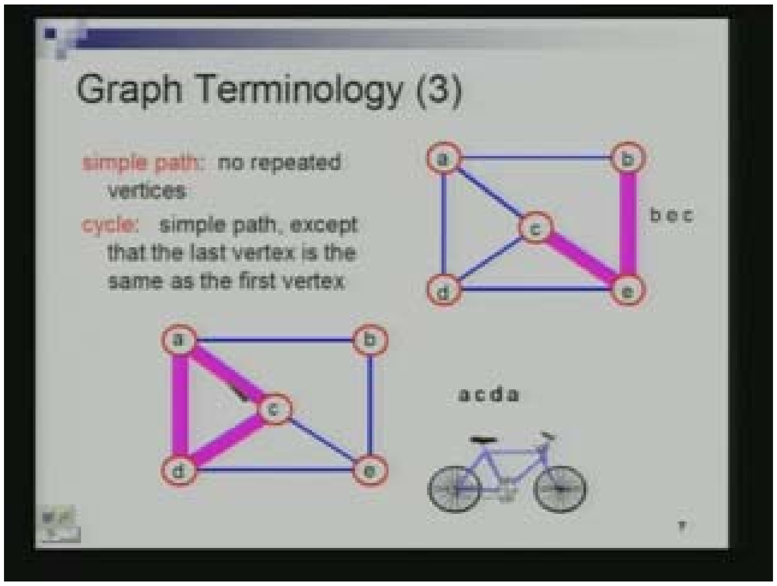
- path: sequence of vertices $v_1, v_2, \ldots v_k$ such that consecutive vertices v_i and v_{i+1} are adjacent.

So a path in a graph is a sequence of vertices let's say V_1, V_2, V_k such that consecutive vertices have an edge between them. So if I take vertex V_i and V_{i+1} then these two vertices are adjacent there is an edge between this vertices. So there are two examples here. So this is my graph, the same graph as before recall that there is an edge between c and e also. So this is the path a b e d c e. Why is this a path? Because there is an edge between a and b, there is an edge between b and e, there is an edge between e and d, between d and c and c and e, so this is a path. Similarly this is the path b e d c because there is an edge between b and e, between d and e, between d and c. it is easy to construct examples which are not paths.

Suppose I had written down a b c, a b c is not a path in this graph. Why because while there is an edge from a to b there is no edge from b to c, so everyone understands what a path is. A simple path is a path in which no vertex is repeated so this is an example of a simple path b e c.

These three vertices are all distinct so it is a simple path. A cycle is a simple path in which the first and the last vertices are the same. So a c d a is a cycle, d a c d is the same cycle, c d a c is also a same cycle. So you can read the cycle anywhere, this is a cycle this is a simple path. In the previous slide we had an example of a path which is not simple. This is not a simple path. Why, this is not a simple path because vertex e is repeated here.

(Refer Slide Time: 13:05)



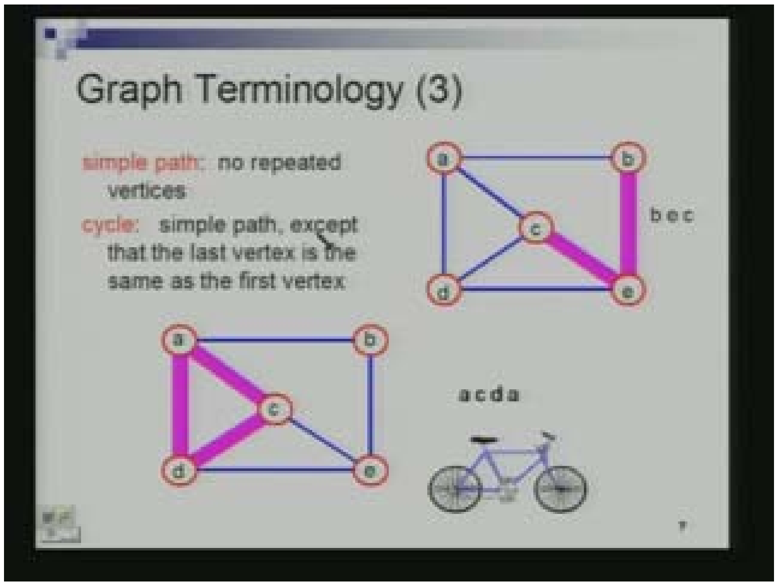
Graph Terminology (3)

simple path: no repeated vertices

cycle: simple path, except that the last vertex is the same as the first vertex

So this is a simple path except that the first and the last vertices are the same. That’s what a cycle is. A graph is said to be connected if there is a path between every pair of vertices in the graph, [Hindi conversation] that the graph is connected.

(Refer Slide Time: 13:48)



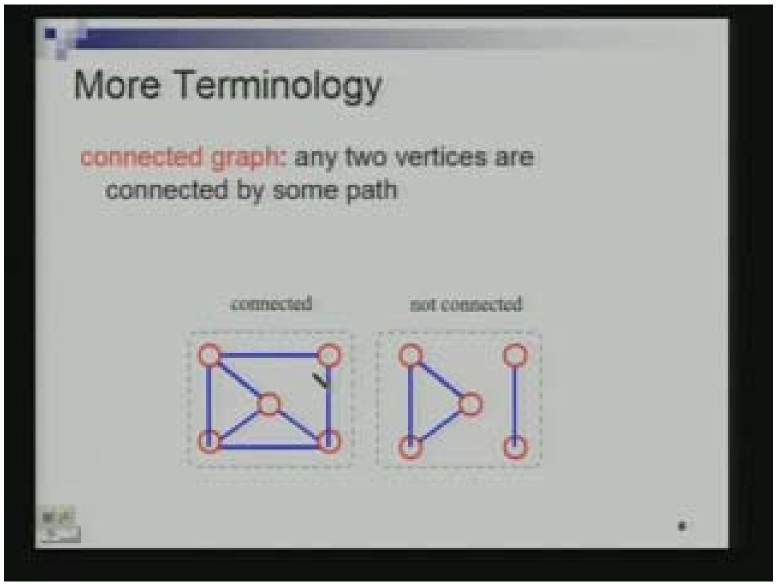
Graph Terminology (3)

simple path: no repeated vertices

cycle: simple path, except that the last vertex is the same as the first vertex

Is this graph connected? [Student: yes, the path] path [Hindi Conversation]. So this graph is connected, this is not connected there is no path from here to here, so this is connected this second one is not connected and this is the common mistake connected [Hindi conversation] there should be a path between every pair of vertices.

(Refer Slide Time: 15.37)

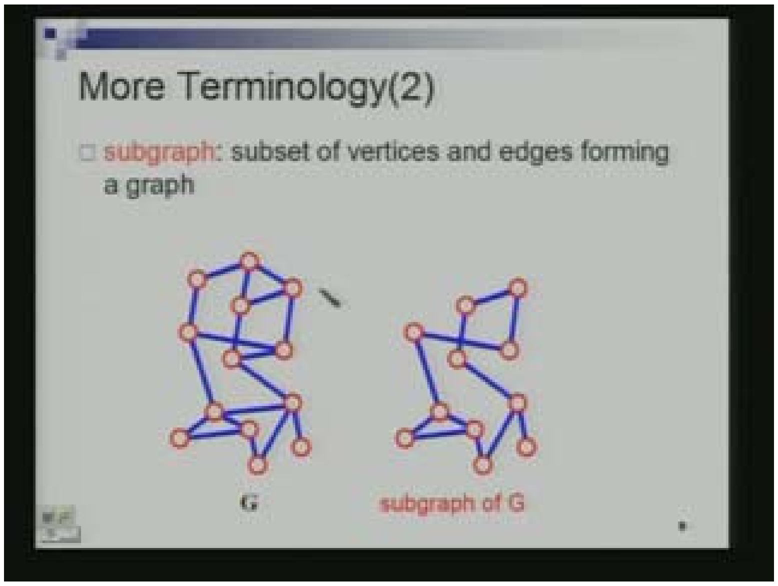


More Terminology

connected graph: any two vertices are connected by some path

If there is a path then it is connected, if there is no path it's not connected. So these two vertices so again this is the common mistake when you are writing a minus especially you are going to say these two vertices are not connected because you don't see an edge between them that's wrong terminology. These two vertices do not have an edge between them but they are connected because there is a path between these two vertices. So we say two vertices are connected if there is a path between them and a graph is connected if there is a path between every pair of vertices. Is this clear to everyone? Let's understand the notion of a sub graph, so this is a graph on the left hand side suppose I take a subset of the vertices and of the edges such that the resulting thing is also a graph.

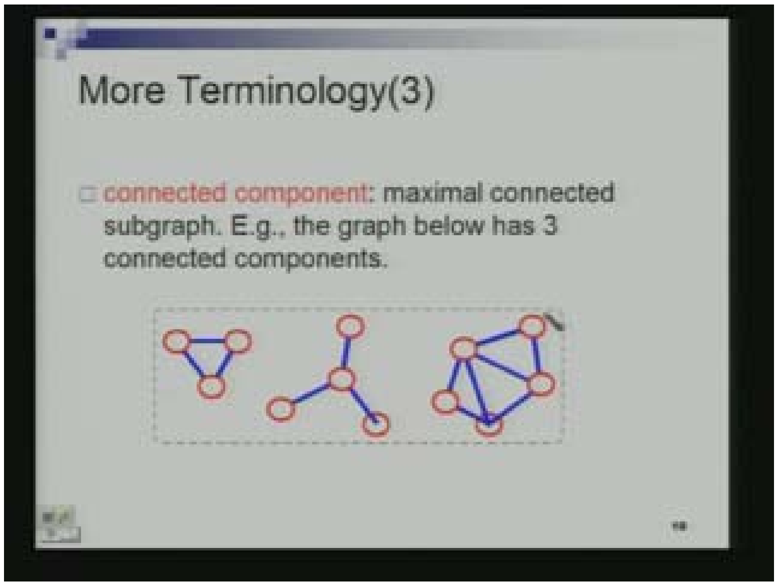
(Refer Slide Time: 16.23)


More Terminology(2)

- subgraph: subset of vertices and edges forming a graph

So I took some vertices from here, this vertex you can see it's corresponding. I took 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 vertices from here. There are 13 vertices in here I took 11 of them and I took some of the edges between these vertices. I am not taken all the edges, as you can see this edges is not here, this would be called a sub graph of this graph. I cannot takes this edge because the other point of this edge is not there, I have not included here at all. For an edge, the two vertices between which the edge is running are also called the end points of that edge. Each edge has two end points and those are the two end points so this is called the sub graph of this graph.

(Refer Slide Time: 17:39)



More Terminology(3)

- connected component: maximal connected subgraph. E.g., the graph below has 3 connected components.

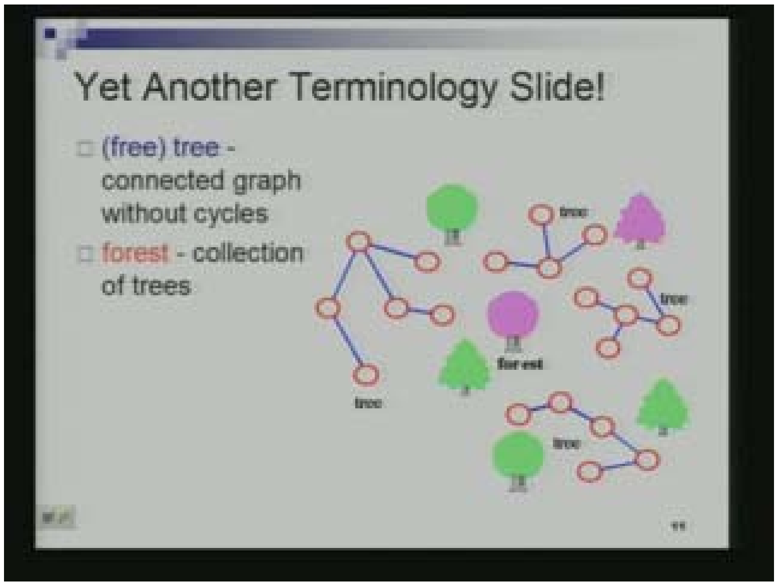
Now let's understand what a connected component is. A connected component is a maximal connected graph. Suppose this is one graph, it is not 3 graphs I have drawn just one graph in. This is not a connected graph. Is this connected? [Hindi conversation] [Student: this is not connected] this is not a connected graph. Why, because there is no path from here to here [Hindi conversation]. This is not connected because there is no path from here to here, there is no path from here to here, so it is not a connected graph. If I look at this sub graph it is connected just this sub graph. These three vertices and these three edges it's connected.

These 4 vertices and these 3 edges are also connected, these 5 vertices and the 7 edges on them are also connected. These 3 are the connected components of this graph. Now what's the definition of connect? It's a maximal connected sub graph. What does a maximal connected sub graph mean? This needs to be understood more carefully. Suppose I were to take this vertex and this vertex and I were to take this edge and this edge. This is a sub graph, yes or no? This is a sub graph of the original graph but this is not a connected component, I am not going to call this a connected component. Why? Because it is not maximal so what does maximal mean? So when we say maximal in this class, we mean a set is called maximal if we cannot increase the size of the set while retaining the property. So a set is said to be maximal with respect to a certain property.

If we cannot add more elements to the set and retain the property that's not true here I can add more elements to this set, I can add more edges or I can add more vertices and both. So I can add this edge and it is still connected I can this vertex and this edge and it is still connected, I can add this vertex and this edge and it is still connected, I can add this edge now it is still connected, I can add this edge now it is still connected. Now if I add any other vertex or any other edge, suppose I decided to add this vertex, I add this but it is not connected anymore. So this is a maximal connected sub graph and so we will call this a connected componenent so this entire thing is the connected component. This is also a connected component and this is also a connected component. [Hindi Conversation] I cannot add any other vertices and still have the property of it being connected.

So essentially intuitively how do you think of connected component? You just see which are the pieces which are connected among each other, each of them is a connected component as simple as that. So this graph is 3 connected components. More terminologies; what is a forest? Forest is a jungle, jungle is a collection of trees and animals but we will leave out the animals. So we are thinking of forest as a collection of trees so these are trees in the forest now what is a tree here.

(Refer Slide Time: 23:38)



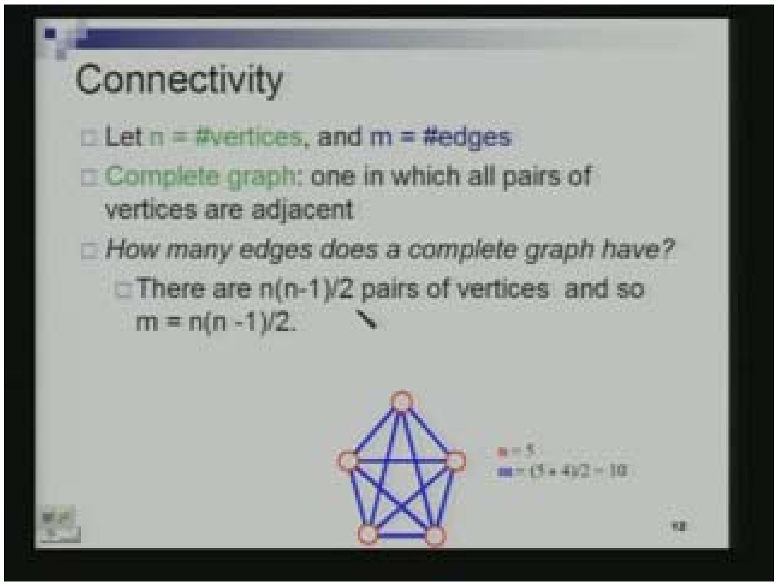
Yet Another Terminology Slide!

- (free) tree - connected graph without cycles
- forest - collection of trees

tree
tree
tree
forest
tree

A tree here is a connected graph which does not have any cycles in it. It's the same as the tree that we till now except the [Hindi conversation] (Refer Slide Time: 23:00). So this is an example of the tree it is a connected sub graph as we can see and it does not have any cycle in it. This is also a tree, this is also a tree, this is also a tree when you have collection of trees it is a forest. So forest is a collection of trees so everyone understands this. What a trees? Tree is a connected sub graph which does not have any cycle in it. So I am typically going to use n to denote the number of vertices and m to denote the number of edges in any graph. So what is the complete graph? A complete graph is one in which there is an edge between every pair of vertices, between every pair of vertices there is an edge. This is an example of a complete graph.

(Refer Slide Time: 24.41)



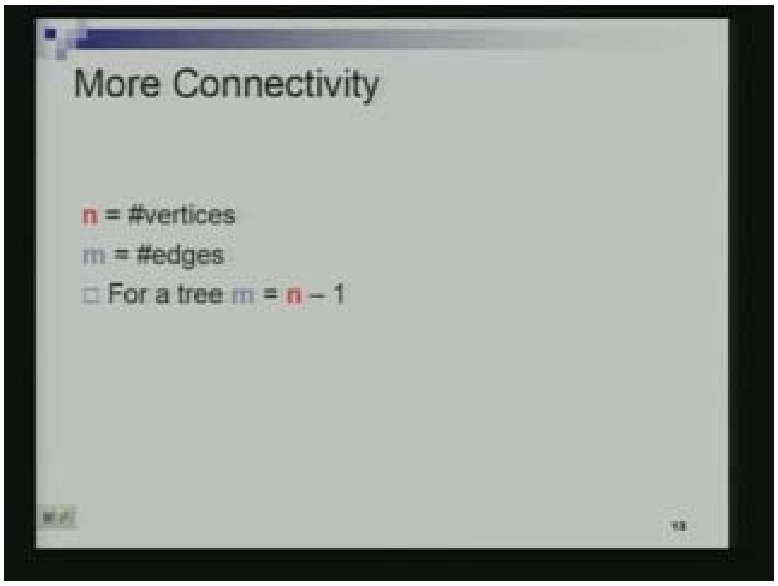
Connectivity

- Let n = #vertices, and m = #edges
- Complete graph: one in which all pairs of vertices are adjacent
- *How many edges does a complete graph have?*
    - There are n(n-1)/2 pairs of vertices and so m = n(n -1)/2.

This is a graph on 5 vertices between every pair of vertices there is an edge. So how many edges does a complete graph have? n_c2, because there are n_22 pairs of vertices and there is an edge between every pair and so you will have so many edges. How many edges does a complete directed graph have? [Student: three by two two] two times n_c2 a directed and a complete. So basically there will have to be and edge in both directions right so it will become twice. If a graph is not complete then the number of edges going to be strictly less than n chose two. So in an undirected graph this is the maximum number of edges that a graph can have, n chose two [Hindi conversation].

Suppose I give you a graph on n vertices, zero, it might not have any edge at all. So the minimum number of edges in a graph on n vertices is zero and the maximum number of edges is n chose two. So once again we have n number of vertices, m number of edges. [Student: minimum elements connected in graph]. That's the slide, suppose in a tree so what is a tree? Recall a tree is the connected graph which does not have any cycle in it. How many edges are there in a tree? I have said number of edges in the tree is n – 1, why? [Student: every pair of] [Student: start from a node and we end and we cannot have a like a cycle so starting] So [student: we can have about one two one two two three after and the number of edges these vertices two] each vertex is degree two. In a tree every vertex is degree two, no. [Hindi conversation][Student: nodes we write a so that will be n minus one you can't have repetition sir we get we can count the edges by, we will take the direction so the edge coming to a node is one].

(Refer Slide Time: 25.54)



More Connectivity

n = #vertices

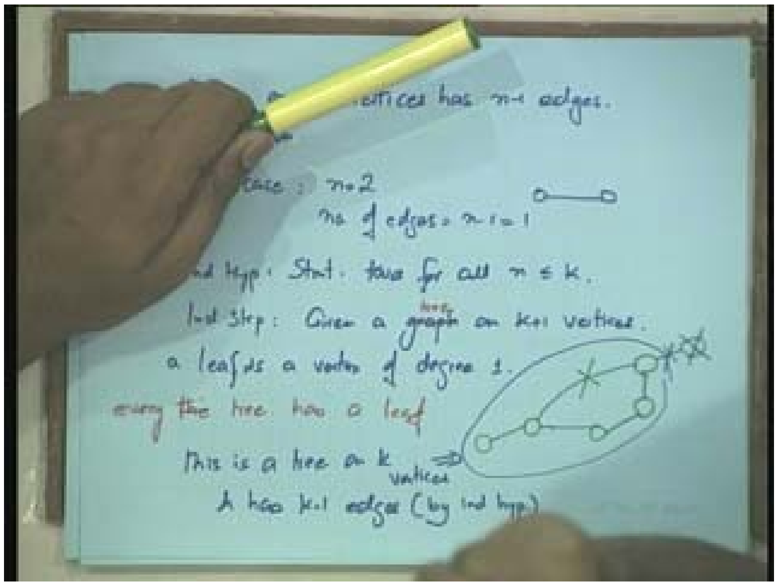
m = #edges

- For a tree m = n – 1

What do you mean coming to a node? So is this edge coming into this node or this edge coming into this node? [student: sir we take this as even starting from if you start from any particular node you don't have whether number of node have to have a] Let's prove this. [Hindi conversation] it is a true statement [Hindi] so let's prove that so what will be the proof? We have to prove that a tree on n vertices has n - 1 edges, induction [Hindi] as simple as that. So proof by induction, so what should be the base case? Let's say n equals two so suppose I have a connected graph on two vertices [Hindi] statement is true. So number of edges equals n - 1 equals one. So induction hypothesis [Hindi] statement true for all n less than or equal to k let's say.

So now the induction step. So given a graph on k + 1 vertices. Why should this have k edges? [Hindi] one leaf good. So he is saying something useful, he is saying there is no cycle in the graph. We have to use somewhere the fact there is no cycle in the graph [Hindi]. He says that there has to be one leaf [Hindi] [student: degree one] good. So let's define a leaf, now as a vertex, a leaf is a vertex of degree one. So his claim is that given a tree on k + 1 vertices. We are given a tree, we are proving this. The tree or every tree has a leaf [Hindi] so maybe we come back to one of the vertices we have already visited [Hindi] so it is not a tree [Hindi] because that was a leaf [Hindi]. [Student: there should be part of the path relating these vertices] exactly this edge cannot be part of any simple path between any two vertices because [hindi] this edge cannot be part of any simple path and so even after I remove this edge and this vertex this there is a path between every pair of vertices. So this is still connected, this is connected and by removing an edge and a vertex I cannot create a cycle [Hindi]. I can apply my induction hypothesis on it so [Hindi] we have removed only one vertex. So this is a tree on k vertices and has k - 1 edges, this is by induction hypothesis [Hindi] and so we prove that [Hindi].

(Refer Slide Time: 35.40)



...ertices has n-1 edges.

...case : n=2

no. of edges = n-1 = 1

Ind hyp : Stat. true for all n ≤ k.

Ind Step : Given a ~~graph~~ tree on k+1 vertices.

a leaf is a vertex of degree 1.

every the tree has a leaf

This is a tree on k vertices ⇒

A has k-1 edges (by ind hyp)

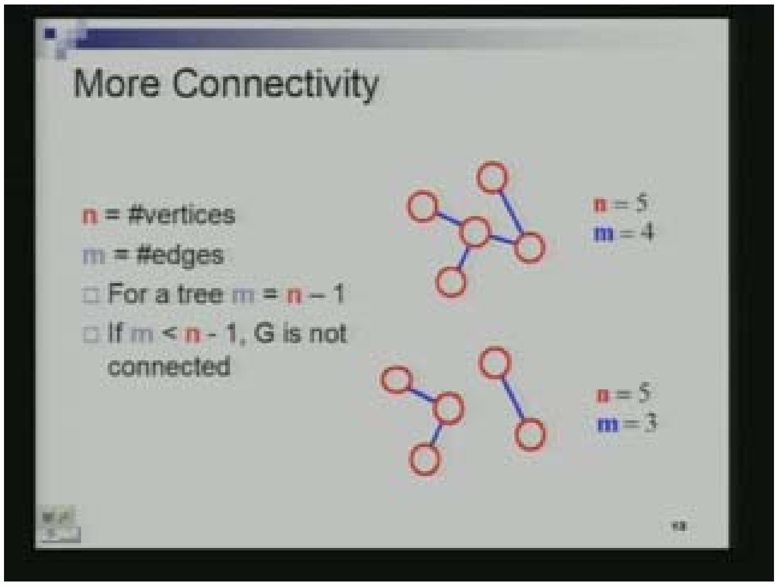
You have to use the fact, both the facts are critical that it is a connected graph and it does not have a cycle in it. Otherwise you will not be able to argue that it has k - 1 edges [Hindi]. That’s the proof for this, everyone follows this. Most text books would have this proof also, you can also go back and and look at one of the text. So if the number of edges is less than n - 1 in a graph then the graph cannot be connected at all. Why? This statement, if the number of edges is less than n - 1 then the graph is not connected proof by contradiction.

Suppose if it is connected then so let’s follow this argument. So suppose it is connected, if it is connected then why is it not a tree? It is not a tree because it has a cycle. So let’s take lets remove an edge from the cycle [Hindi] I should have switched but okay so what what are we trying to argue? If number of edges is less than n - 1 then G is not connected. So this is another useful thing to remember that suppose I have a cycle, G is a graph. Suppose I have a graph in which there is a cycle [Hindi] if you have a cycle and if you remove any edge from the cycle you cannot make the graph disconnected by doing that.

So what is the argument that to prove this claim? If suppose I have a graph on less than n - 1 on less than n - 1 which is connected. Why it is not a tree? It is not a tree because there is a cycle in let me remove an edge from the cycle I only reduce the number of edges and it’s still connected. If there is another cycle let me still remove another edge so I will only get less than n - 1 edges and the graph will remain connected eventually I will get a tree after removing all of this. So I am contradicting the earlier claim which says that any tree has to have exactly n - 1 edges in it.

It cannot have less than n - 1 edges so any graph which has less than n -1 edges cannot be connected [Hindi]. Is there something that is not clear? So couple of examples n =5, m=4 this is a tree on 5 vertices.

(Refer Slide Time: 40.46)



More Connectivity

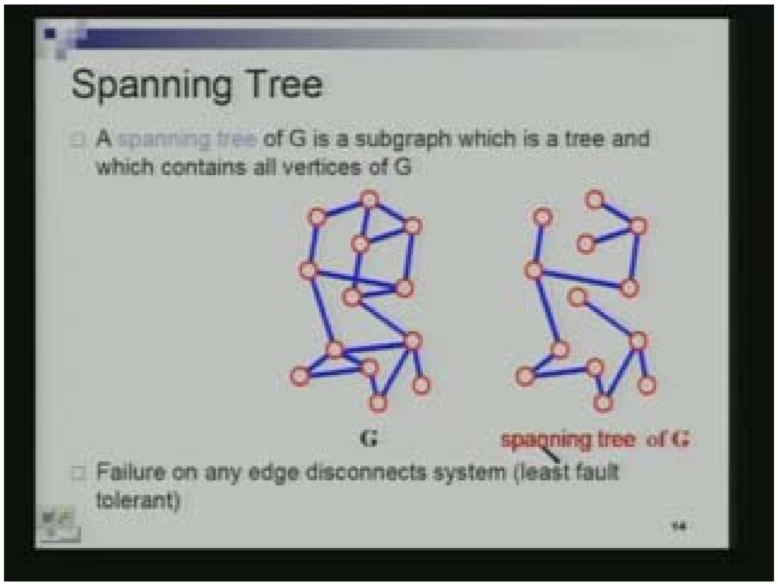
n = #vertices

m = #edges

- For a tree m = n – 1
- If m < n - 1, G is not connected

It has to have four edges, this is a graph on 5 vertices and 3 edges and it cannot be a tree, it cannot be a connected graph at all. Let me ask you a question suppose I have graph on n vertices and it has n - k edges n - k edges. How many connected components do you think it has? I have a graph on n vertices and n - k edges, how many connected components it has? k or more, k when there would be no cycle and if there were cycles then it could have more number of connected components, try to prove this. This is a very simple exercise. So a given a graph on n vertices and n - k edges how many connected components does it have? So more terms; a spanning tree is a sub graph which means you are given a graph so it is a sub graph of a graph and this sub graph has to be a tree and it should include all the vertices of the graph.

(Refer Slide Time: 42.13)



Spanning Tree

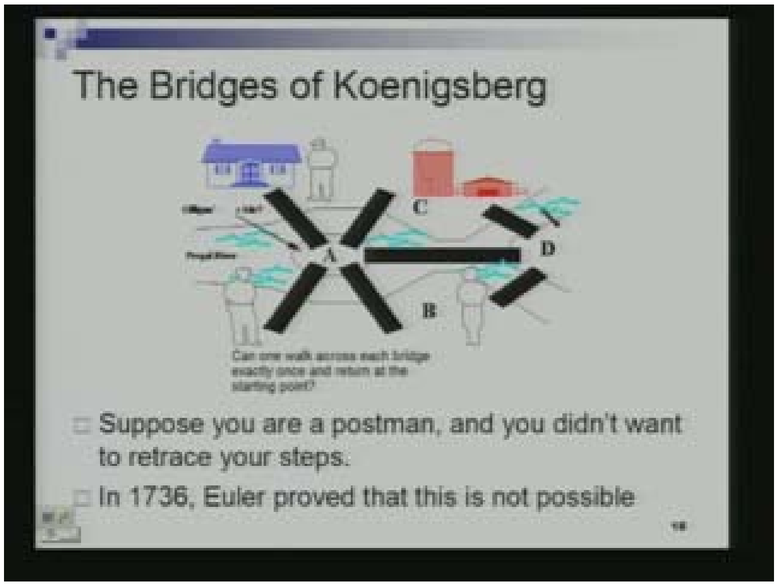
- A spanning tree of G is a subgraph which is a tree and which contains all vertices of G
- Failure on any edge disconnects system (least fault tolerant)

G
spanning tree of G
14

So spanning tree [Hindi] tree which means the sub graph has to be a tree and [hindi] it should include everything; include everything here means include all the vertices. So as you can see this sub graph includes all the 4 3 7 and 3 10 13 vertices that are there and it is a tree. There is no cycle here so this is the spanning tree of this graph, this is the graph and this is the spanning tree of this graph. G has to be connected if G is not connected then there is no notion of the spanning tree. If G is not connected then no sub graph of the graph of G cannot be a spanning tree [Hindi]. So this is a useful thing to have, quite often your network could be a just spanning tree.

Suppose these are points I want to connect so these are cities, these are possible roads that I can build but I just want to put the minimum amount of effort, I want to build has few roads as possible so that all these cities are still connected so I could built a spanning tree but this does not provide you any fault tolerance what does that mean [Hindi] you cannot reach from some city to some other city now. As you can see if I cut of this link then these 4 vertices would be disconnected from the other 8 vertices [Hindi] these 6 vertices would be disconnected from the other 7. Spanning tree is a useful but they provide don’t provide much fault tolerance.

(Refer Slide Time: 44.43)



The Bridges of Koenigsberg

Can one walk across each bridge exactly once and return at the starting point?

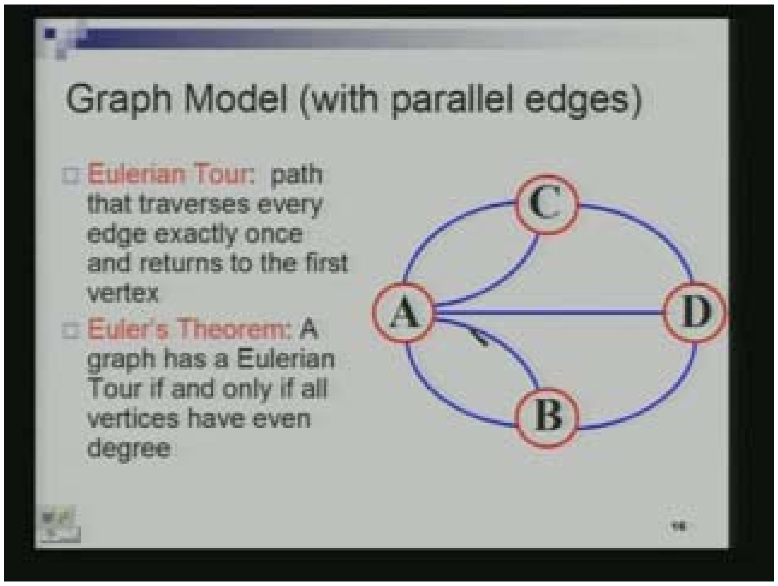
- Suppose you are a postman, and you didn't want to retrace your steps.
- In 1736, Euler proved that this is not possible

Let's talk about bridges. Koenigsberg, this is a city in Germany or Austria I don't remember where. So pragal river okay I don't remember where this is. This city has this nice thing, there is a river flowing through the city and there is an island in the river and there are bridges in this manner so A is this island and there is a bridge from here to here, here so there are 7 bridges in all. This black bar are the edges so question is can you start from here let's say or any point. So can one across each bridge exactly once and return to the starting point. Why no, so suppose I start from here I can take this bridge go here [student: it will land up] and you can go on land up [Hindi] so on and see.

Let's see whether we can solve this problem or not? Suppose this would have been useful if you were a postmen who had to visit the various brides and you did not want to retrace the steps. So this is also known as koenigsberg problem and Euler proved that this is not a problem and we will give a simple proof for that one. So we can model this thing as a graph, there is this island A so these are the going to be the vertices of my graph. This island A this is one piece of land and there is this part B because I can go from anywhere to here. This is one vertex, there is a vertex D and there is a vertex C which is this part. So I will have a graph with 4 vertices in it A B C D and then depending upon so since there is a bridge from B to A.

In fact there are two bridges from B to A so I will put two edges between B and A. similarly there are two bridges between A and C so I will put two edges between A and C. There is one bridge from A to D so I will put one edge between A and D, there is an bridge between D and B so I will put one edge between B and D and an edge between C and B so I will get. This is not a graph. Why is this not a graph? Because we did not define a notion of two edges between pair of vertices, we just talked about pair of vertices. The edges don't form a set, they form a multi set so this is called multi graph.

(Refer Slide Time: 48.43)



Graph Model (with parallel edges)

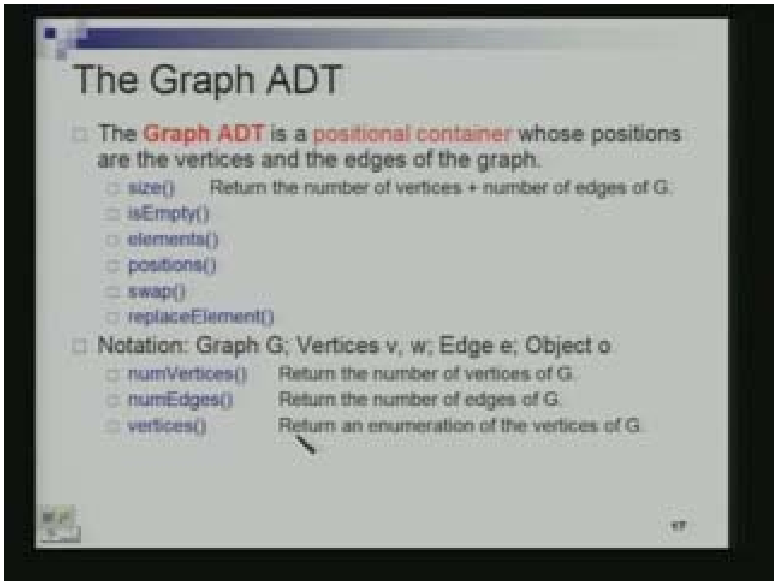
- Eulerian Tour: path that traverses every edge exactly once and returns to the first vertex
- Euler's Theorem: A graph has a Eulerian Tour if and only if all vertices have even degree

What is a multi-graph? In which they put the many edges between a pair of vertices is called a multi graph but this captures that problem in certain sets. So eulerian tour is a path that traverse every edge exactly once and returns to the first vertex and that's exactly what we want to do. Because these are the bridges so we want to traverse each bridge exactly once and return to the starting vertex.

Can you do that on this graph? So same problem can now be thought of here, can is start from A and come back to A and and visit each or traverse each exactly once. So the same question a same, can you draw this picture without lifting your pencil or redrawing an edge, you know coming back over a line twice. So [Hindi] Euler theorem says that you can do this if and only if every vertex has even degree [Hindi]. When you come to a vertex, you come by one edge and then you have to go by another edge and if you come again then you will need another edge to or fresh edge to go off by so every vertex has to have an even degree for this to work but here there are all vertices of odd degrees so clearly this cannot be done.

Now let's quickly do the uninteresting part, the abstract data type. The graph can be thought of as a container of positions. So you have the regular methods for any positional container like queues and stacks. We always had this methods called size and Is Empty and elements; elements would return all the vertices and the edges that's in and you can have some methods like swap which can swap two positions replaceElement those kind of thing, these are methods associated with the regular positional container swap is the generic method for any positional container. When you are saying that provide two positions and swap the contents at those two positions that's the swap method. So here I am not saying it specifically to the graph abstract data type, you will have to think of what it would mean. So you could decide what it means here for this particular data type but I am saying it is a generic methods. These are all generic methods for positional container and I am just saying in that context. So here I have methods which are specific to graphs so numVertices would be a method which returns number of vertices numEdges number of edges vertices would know be an enumeration of all the vertices.

(Refer Slide Time: 51.05)



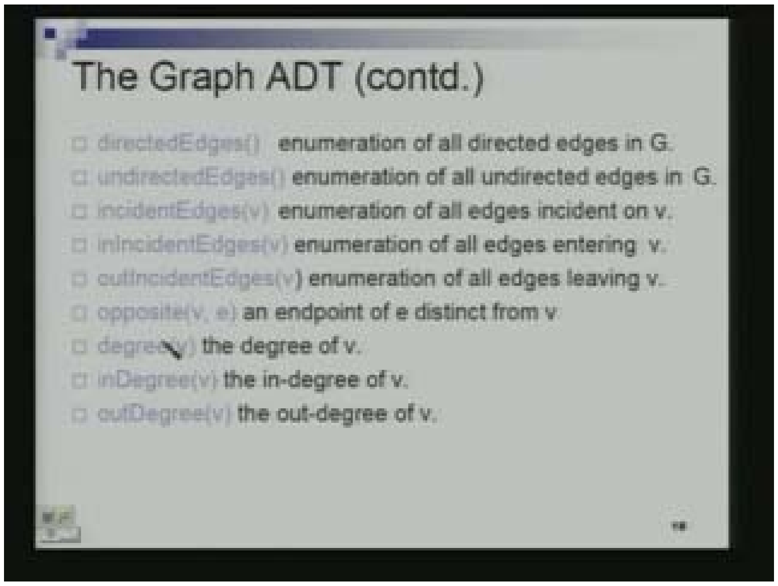
The Graph ADT

- The Graph ADT is a positional container whose positions are the vertices and the edges of the graph.
  - size() Return the number of vertices + number of edges of G.
  - isEmpty()
  - elements()
  - positions()
  - swap()
  - replaceElement()
- Notation: Graph G; Vertices v, w; Edge e; Object o
  - numVertices() Return the number of vertices of G.
  - numEdges() Return the number of edges of G.
  - vertices() Return an enumeration of the vertices of G.

So it would be a method returns an iterator which will let you iterate through the various vertices of the graph, edges could be a method which returns all the edges. DirectedEdges would be a method if you had a directed graph, it would return all the enumerated all the directed edges in the graph. What does enumerator do and an iterator? It basically returns an object which has two methods associated with it, one method is next and the other method is whether there is anything left, has next whether there is a next method next element at all or not.

So as you every time you call next it gives you a next object in the enumeration so when you are enumerating edges I call next once it will give me one edge, when I call next again it will give me another edge. What order this edge is come in that you typically do not know. It depends upon how you implemented the iterator. UndirectedEdges could similarly enumerate all the undirected edges incident Edges, if I specify a vertex it would enumerate all the edges incident at that vertex. This is for an undirected graph incident Edges; for a directed graph right there are two kinds of edges either there would be edges which start from this vertex or there would be vertex which end at this vertex. So it could have a notion of any incident edges which are edges entering a vertexV which are ending at vertex V and you could have an out incident edges which are edges which are starting from vertex V going out of vertex, opposite. so I specify an edge e, all of these are objects an edge is also an object and I specify one end point on the edge so this method gives me the other end point of that edge.

(Refer Slide Time: 53.24)

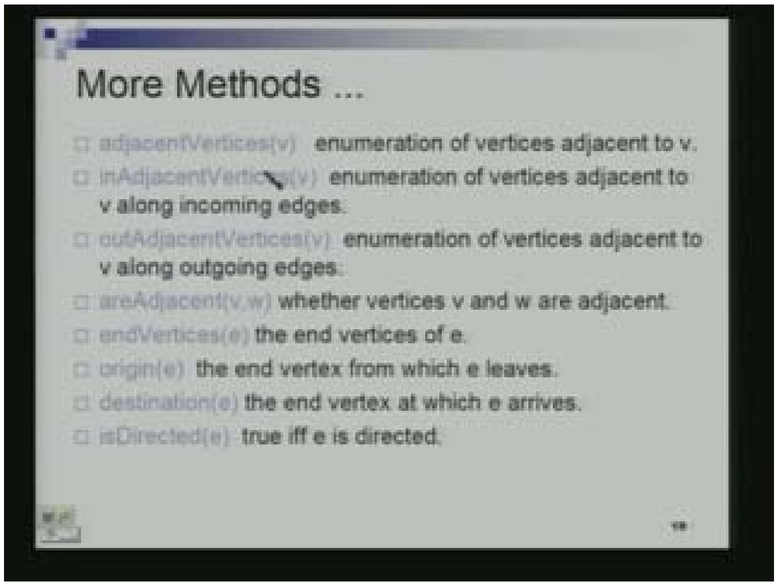


The Graph ADT (contd.)

- directedEdges() enumeration of all directed edges in G.
- undirectedEdges() enumeration of all undirected edges in G.
- incidentEdges(v) enumeration of all edges incident on v.
- inIncidentEdges(v) enumeration of all edges entering v.
- outIncidentEdges(v) enumeration of all edges leaving v.
- opposite(v, e) an endpoint of e distinct from v
- degree(v) the degree of v.
- inDegree(v) the in-degree of v.
- outDegree(v) the out-degree of v.

Degree gives me the degree of the vertex, inDegree so degree would be for an undirected graph, for a directed graph there would be the notion of in degree and an outdegree. Indegree would be n number of edges coming into the vertex outDegree would be the number of edges leaving the vertex. Similarly I could have adjacent vertices, adjacent vertices would be a method which will turns an iterator over all the vertices which are adjacent to this particular vertex. This would be for the undirected graph, for a directed graph you could similarly have a notion of inAdjacent and an outAdjacentVertices. Then you could have a method areAdjacent whether vertices two vertices v and w are adjacent or not. So this would be return a Boolean value; endVertices given an edge it will return the two end points of that edge.

(Refer Slide Time: 53.31)



More Methods ...

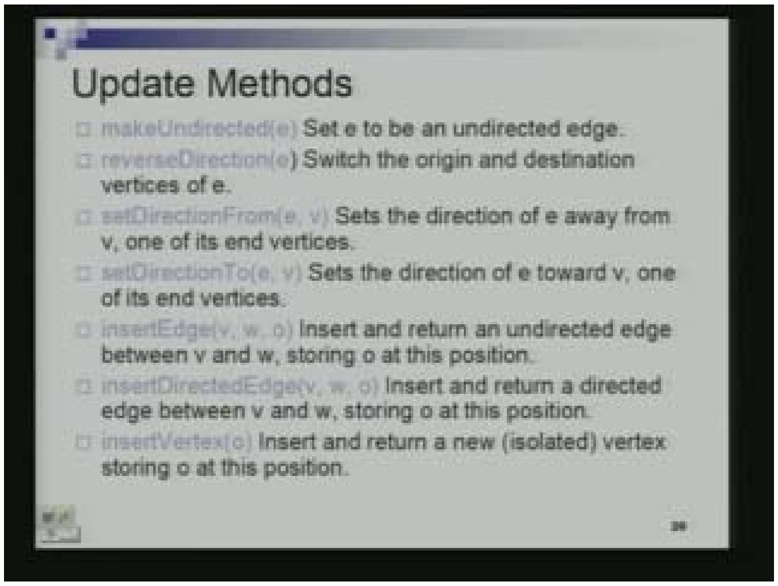
- adjacentVertices(v) enumeration of vertices adjacent to v.
- inAdjacentVertices(v) enumeration of vertices adjacent to v along incoming edges.
- outAdjacentVertices(v) enumeration of vertices adjacent to v along outgoing edges.
- areAdjacent(v,w) whether vertices v and w are adjacent.
- endVertices(e) the end vertices of e.
- origin(e) the end vertex from which e leaves.
- destination(e) the end vertex at which e arrives.
- isDirected(e) true iff e is directed.

Origin, for a directed edge e it would return where the edge is starting from, destination for a directed edge e it would return where the edge is ending. Given an edge e it will tell whether it is directed or not. This method would be useful when you have, what are called mixed graphs, mixed graphs some edges are directed and some are undirected. Can you give me a setting where it would be useful to have a mixed graph, what kind of a problem setting can you imagine there?

It would be natural to have a [student: roads] roads, traffic network once again where you have some roads are one ways. So you are bi directed edges, roads which are two way could be undirected edges and roads which are only one way could be directed edges. There such a methods could be useful because given an edge you can then determine whether it is a directed edge or an undirected edge. I will just take, I guess this is last slide yes it is.

Make Undirected e, so you are given edge and you set it to be an undirected edge. You can have a method which reverses the direction so you can have tonnes and tonnes of update method also. You can have methods to create the graph, change remove an edge, remove a vertex do whatever you want. So set direction from, so you can set the direction of an edge suitably we just look through this these slides that I have given. So this is just a subset of method depending upon what application you have, you could design your own set of methods.

(Refer Slide Time: 55.50)



Update Methods

- makeUndirected(e) Set e to be an undirected edge.
- reverseDirection(e) Switch the origin and destination vertices of e.
- setDirectionFrom(e, v) Sets the direction of e away from v, one of its end vertices.
- setDirectionTo(e, v) Sets the direction of e toward v, one of its end vertices.
- insertEdge(v, w, o) Insert and return an undirected edge between v and w, storing o at this position.
- insertDirectedEdge(v, w, o) Insert and return a directed edge between v and w, storing o at this position.
- insertVertex(o) Insert and return a new (isolated) vertex storing o at this position.

So graph can be thought of as data type, is an abstract datatype on which you can have a bunch of methods which you can use to update and modify the data type. So with that we will end our discussion on graphs we will continue in next class however to see how to actually represent a graph what kind of data structures can you use to represent graphs.