

Discrete Mathematics
Prof. Ashish Choudhury
Department of Mathematics and Statistics
International Institute of Information Technology, Bangalore

Lecture -47
Various Operations on Graphs

Hello everyone, welcome to this lecture. The plan for this lecture is as follows. In this lecture we will discuss.

(Refer Slide Time: 00:27)

Lecture Overview

- Operations on graphs
- Representing graphs
- Graph isomorphism
- Connectivity in a graph

4

Various operations that we can perform on graphs. We will see various mechanisms of representing graphs. We will discuss the graph isomorphism problem and we will define the connectivity in a graph.

(Refer Slide Time: 00:40)

New Graphs from Old Graphs

□ **Subgraph of a graph:**
 ❖ Let $G = (V, E)$ be a graph. Then a graph $H = (W, F)$ is a subgraph of G , if:
 $W \subseteq V$ and $F \subseteq E$

□ **Proper subgraph of a graph** --- $H = (W, F)$ is a proper subgraph of G , if:
 $W \subseteq V, F \subseteq E$ and $G \neq H$

G H

So, it turns out that since graph is nothing but a collection of two sets we can perform various set theoretic operations on a graph and obtain new graphs. So, let us discuss some of the important operations which we can perform on the graphs. So, we will first define what we call as the subgraph of a graph, if you are given a graph $G = (V, E)$ with the vertex set being V and edge set being E . Then, a graph $H = (W, F)$ with vertex set W and edge set F will be called as a subgraph of G , if the vertex set W of H is a subset of the vertex set V of G , namely all the vertices of H should be the vertices of G and the edge set F of H should be an edge set, there should be a subset of the edge set E of G , that means all the edges of F should be present in. So, that is a simple straight forward definition of a subgraph. Now let us define what we call a proper subgraph of a graph.

So, we will first give an intuitive definition what exactly we can think of a proper subgraph and then we will see that definition is not correct. So, remember when we define the proper subset of a set we say that $A \subset B$, if A is a subset of B and there is something extra which is always there in B which is not there in A . So, let us try to extend that definition in the context of a proper subgraph.

So, say my definition is that H graph H will be called as a proper subgraph of G if either the vertex set of H is a proper subgraph, $W \subset V$ it is a proper subset of the vertex set of G and the edge set of H , $F \subset E$ it is a proper subset of the edge set of G , suppose that is my definition. But this is my definition, then with respect to this definition if I take my graph G and H to be this then H will not

be considered as a proper subset of G. This is because all the vertices of H are the vertices of G as well, so this conditions that $W \subset V$ is not satisfied. So, as per this definition, I will say that H is not a proper subgraph of G but this is a proper subgraph of G. Because there is something extra in G which is not there in H, namely the edge between the vertex b and c is there in G but that is not there in the graph H.

So, that means the definition that I gave here is not the correct definition, so that is why the right definition of the proper subgraph is the following. I will say that H is a proper subgraph of graph G if it is a subgraph of G because that is definitely the requirement. And it is a subgraph which is different from the graph G or the parent graph. If that is the case then I will say that my graph H is a proper subgraph of the graph G. So this condition that G is not equal to H takes care of the fact that there is something extra in the graph G which is not there in the graph H.

(Refer Slide Time: 04:16)

New Graphs from Old Graphs

□ Induced subgraph:

❖ Let $G = (V, E)$ be a graph and $W \subseteq V$. Then G' is the subgraph induced by the vertex set W , if:

$G' = (W, E')$, where $E' \subseteq E$ and $E' = E \cap \{ \overset{v_i, v_j}{(a_i, a_j)} : \overset{v_i, v_j}{a_i, a_j} \in W \}$

All edges from E with both end-points in W

Subgraph induced by $\{b, c\}$
Subgraph induced by $\{a\}$

Now let us next define what we call as induced subgraph. So, if you are given a graph $G = (V, E)$ with vertex set V and edge set E and if I take a subset of vertices W , then G' is called the induced subgraph or the induced by the vertex set W such that the vertex set of G' is W and the edge set E' of G' consists of only those edges whose both the end points are within the subset W .

So, basically what this induced subgraph tries to do is the following. You are given a collection of vertices W , that W could be empty or it could be the entire set of vertices V . So, you are given

some subset W and you are focusing only on that part of the graph G where all the edges have endpoints within the subset W only. Even if there are all the edges where one of the end points is outside the subset W are not focusing on those edges.

So, that is the definition of the induced subgraph, so its vertex set will be W and the edge set will be only those edges, this should be (v_i, v_j) , not (e_i, e_j) , such that both the end points are members of the subset W . So, for instance if this is my graph G and if I take my $W = \{b, c\}$, then I am focusing only on that part of the graph G where the edges have their end points restricted within the subset W .

So, that means I cannot take this edge, this edge is not allowed because one of the end points is a and that a is not within my subset W . Similarly, I cannot take the edge between a and c because one of the end points of this edge is the set a which is not there in my W . Whereas if I take my W to be the node a only, then I get an empty graph. Empty graph in the sense which has no edges because this edge will not be there as b is not within my W , this edge will not be there because the node c is not within my W and this edge also will not be there.

(Refer Slide Time: 07:03)

New Graphs from Old Graphs

□ Let $G = (V, E)$ be a graph

- ❖ $G - e = (V, E - \{e\})$ ❖ $G - E' = (V, E - E')$
- ❖ $G + e = (V \cup \{v_i, v_j\}, E \cup \{e\})$, where $e = (v_i, v_j)$
- ❖ $G - v = (V - \{v\}, E')$, where $E' = E - \{(v, *)\}$
- ❖ $G - V' = (V - V', E')$, where $E' = E - \{(v, *) : v \in V'\}$

The diagram illustrates three graphs derived from the original graph G . Graph G is a triangle with vertices a (blue), b (yellow), and c (yellow), and edges e_1 (between a and b), e_2 (between b and c), and e_3 (between a and c). Graph $G - a$ is a single vertex b (yellow). Graph $G - e_1$ is a V-shape with vertices a (blue), b (yellow), and c (yellow), and edges e_2 (between b and c) and e_3 (between a and c).

So, now let us see some set theoretic operations that we can perform on an existing graph to get new graphs, so imagine you are given a graph then the deletion of an edge is denoted by this operation. So, imagine a small e is an edge, so if I delete an edge then the vertex set does not get

disturbed, it remains the same it is only the edge set which gets affected. That means my new edge set will be the old edge set minus the edge e which I am excluding.

Whereas if I am removing a collection of edges, even in that case my vertex set remains intact, it is only the edge set which gets affected, that means my new edge set will be the difference of the old edge set and the edge set E' which I am deleting from the graphs. So, it is like saying the following; imagine your graph represents a computer network where the nodes are the computers and the edge represents a cable connecting two computers. So, if you remove a cable that does not mean that the corresponding computers also get deleted, the computers are still there, it is only the cables which are getting removed. Similarly, if I add a new edge that is expressed by this operation then my edge set gets affected, so I will be including a new edge, so I will be including a new edge and the end points of the edge e will be included in my vertex set.

On the other hand, if I delete a vertex from my graph G , then definitely the vertex that gets affected and also the edge set gets affected. So, I have to remove all the edges whose one of the endpoints is v from my graph, it is like saying the following: again, if I take the fact that my graph represents a computer network then deleting a vertex is equivalent to saying that I am deleting or removing a computer itself from my network. So, if I remove a computer from the network then whichever cable has one of its end points as that computer, those edges will not be there anymore in my computer network, so I have to modify my edge set as well. Whereas if I remove a set of vertices V' then my new vertex set will be the difference of the old vertex set and the vertex of V' and edge set E' will be the following.

I have to remove all the edges where one of the end points of the edge is in my subset V' . So, let me demonstrate these operations with respect to this graph, so imagine this is my graph G , if I remove the vertex a , then this edge e_1 and edge e_3 will no longer be there and I will get this reduced graph, whereas if I remove the cable connecting the node a and node b or the computer a and computer b then only the edge e_1 vanishes, the vertex set remains the same.

(Refer Slide Time: 10:40)

Representing Graphs

Adjacency matrix

$G = (V, E)$ $|V| = n$

- ❖ Boolean matrix $\rightarrow n \times n$ matrix with entries $v_{i1}, v_{i2}, v_{i3}, \dots, v_{in}$
- ❖ Preferred for dense graphs

(i, j) th entry is 1 iff $(v_i, v_j) \in E$

Adjacency list

- ❖ Space efficient for sparse graphs

Incidence matrix

- ❖ $|V| \times |E|$ matrix
- ❖ Shows the relationship between an edge and its end-points
- ❖ $(v_i, e), (v_j, e)$ entries 1 $\Leftrightarrow e = (v_i, v_j)$

Now let us discuss the various data structures that we can use to represent graphs and people who have studied data structure, they must be knowing two of the common representations that we use to represent graphs. So, again I am explaining in the context of undirected graphs, but you can easily generalize this data structure to represent directed graphs as well, so we have what we call as the adjacency matrix representation, so this is a boolean matrix.

So, if your graph G is consisting of vertex set V and edge set E and if the cardinality of the vertex set is n , then this matrix is an $n \times n$ boolean matrix and the (i, j) th entry = 1 iff $(v_i, v_j) \in E$, otherwise it will be 0. And this representation is preferred for dense graph. What do we mean by dense graph? A graph which has a lot of edges, that means it is not the case that you have very few edges, that means you have lots of edges in the graph in which case a majority of the entries in your matrix will be 1.

Whereas a sparse graph is the graph which has lots of vertices but very few edges, for such a graph the adjacency list is the preferred data structure for representation of the graph. So, what is this adjacency list? So, it is a collection of linked list, each link is basically a collection of n -linked lists, so you will have the first linked list with v_1 as the starting node, second linked list as with v_2 as the starting node and n th linked list with v_n as the starting node and in v_1 you will now link all the nodes which are incident.

You link all the nodes which are the endpoints of an edge with v_1 as its one of the end points. That means in your graph G , you focus on the edges which have v_1 as one of the end points, so for those edges find out the other end points and you put them in the linked list with v_1 as the end point. So, basically the linked list starting with v_1 list down all the neighbors of v_1 , similarly the linked list with v_2 list down all the neighbors of v_2 and so on.

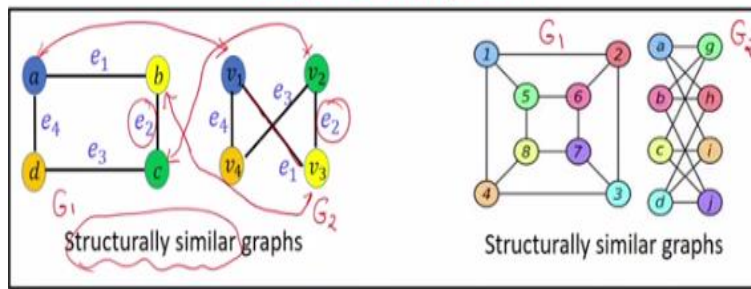
So, now you can see why this representation is very popular for representing sparse graph, that means if graphs which have only a very few edges, if you have very few edges then the size of each linked list will be very small. You do not need a huge matrix with lots of entries being zeros and only few entries being one, so these are the two popular representations for representing an undirected graph, they can be used even for representing directed graphs as well and there is this third data structure which is called as the incidence matrix.

So, what is this incidence matrix? It is so again I am explaining, assuming an undirected graph, so this will be a matrix with $|V| \times |E|$. And it basically represents the relationship between the edge and its endpoint. So, what it means is the following if you have an edge $e = (v_i, v_j)$ then the entry number $(v_i, e) = 1$ and $(v_j, e) = 1$, otherwise and the remaining entries will be 0.

So, for instance if this is my graph G , so there are 3 vertices, so three rows and there are three edges, so three columns. What are the endpoints of e_1 ? The endpoints of e_1 are a and b, so under the column e_1 , I only mark the entry for a row and b row as 1 to show that the end points of the edge e_1 are a and b and all other column entries under e_1 will be 0. Similarly, the end points of e_2 are b and c, so only the entry in row number b and column number e_2 will be 1. And entry in row number c and column number e_2 will be 1 and all other entries in column number e_2 will be 0 and so on, so that is the incidence matrix.

(Refer Slide Time: 16:16)

Graph Isomorphism



- Graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are called **isomorphic** ($G_1 \cong G_2$), if:
 - there is a bijection $\pi: V_1 \rightarrow V_2$, such that $(u, v) \in E_1 \Leftrightarrow (\pi(u), \pi(v)) \in E_2$
- Naïve algorithm** to check graph isomorphism ?
 - ❖ Check all $n!$ bijections from V_1 to V_2

Now let us define a graph isomorphism. So, if you see these two graphs, pictorially they are drawn in a different way. So, the first graph is a rectangle whereas the other graph does not look like a rectangle graph because you might be saying that there are two edges which are crossing each other. But if you see closely structurally, they are similar graphs. What do I mean by structurally they are similar graphs, so both the graphs have the same number of nodes namely 4, same number of edges namely 4 edges and even though the vertex names are different in the two graphs.

So, if I call this as graph G_1 and this as graph G_2 , the name of the vertices of G_1 are $\{a, b, c, d\}$ whereas the name of the vertices of G_2 are $\{v_1, v_2, v_3, v_4\}$, so you will say that how can they be the same graph because the vertex names are different. If I do not focus on the name of the vertices and the name of the edges but mentally think in my mind whether there exists a one-to-one correspondence between the vertices of the graph and the edges, then I find that the 2 graphs are structurally same. What does that mean?

If I think in my mind that vertex a of G_1 corresponds to vertex v_1 of graph G_2 and if I consider vertex b of graph G_1 corresponding to vertex v_3 of G_2 , then you can see that there exists an edge between a and b in graph G_1 and similarly I can consider this edge as the edge e_1 between the mapping of a and the mapping of b . Similarly, this node c , I can associate with the node v_2 here, and you can see that if that is the case then the edge e_2 in the graph G_1 corresponds to the edge e_2 of this graph G_2 and so on.

So, in that sense they are structurally the same graphs but drawn in a different fashion. In the same way this graph G_1 and this graph G_2 are structurally same, so the node 1 of graph G_1 corresponds to the node a of graph G_2 and so on and you can then verify that structurally if I reinterpret graph G_2 , then I can redraw it in the same way as the graph G_1 . So, in that sense these pairs of graphs are isomorphic, they are structurally drawn differently, they are drawn differently but structurally they represent the same information.

That means I can always redraw one of the graphs as another graph. So how do I formally define whether two graphs are isomorphic or not? So, imagine you are given two graphs G_1 and G_2 , they are called as isomorphic and isomorphic graphs are represented by this notation. So, I will say that the two graphs are isomorphic, if I can define a bijective mapping between the vertex set V_1 and the vertex at V_2 such that the following holds.

If you have an edge between the node u and v in the first graph, then you focus on the mapped u vertex in G_2 and the mapped v vertex in G_2 and there should be an edge between the mapped u vertex and the mapped v vertex in E_2 as well and this implication is bi-implication, that means other way around should also hold. So, if you can find one such mapping, one such bijection between the vertex sets, then we will say that two graphs are isomorphic and isomorphism is denoted by the bijection, $\pi: V_1 \rightarrow V_2$.

So, when we say that show me an isomorphism between graph G_1 and G_2 basically I am asking you to show the one-to-one correspondence between the two vertex sets but namely the vertex set of G_1 and the vertex set of G_2 such that this bi-implication is true. So, the graph isomorphism problem is the following: you are given two graphs G_1 and G_2 and you have to check whether they are isomorphic or not.

Checking whether they are isomorphic or not is equivalent to checking whether there exists a bijection between the vertex sets of the two graphs such that bi-implication is true. So, what will be the naive algorithm to check whether two given graphs are isomorphic or not? If $|V_1| = |V_2| =$

n , the cardinality of the vertex sets of the two graphs is n . By the way a simple necessary condition for the two graphs being isomorphic is that they should have the same number of vertices.

If one graph has more number of vertices than another how at the first place they can be isomorphic, they can never be isomorphic. So, imagine that the vertex set of both the graphs is of cardinality n , then a naive algorithm to check whether the two graphs are isomorphic is to try all possible $n!$ bijection between the vertex at V_1 and the vertex at V_2 and for each of those bijections check this implication is true or not.

So, indeed if the two graphs are isomorphic one of these $n!$ bijections will satisfy the bi-implication and hence you can declare that the two graphs are isomorphic. But if all the $n!$ bijections fail to satisfy this bi-implication, we will say that the two graphs are not isomorphic. But then what is the running time of this algorithm? You have to try $n!$ bijections and $n!$ is an enormously large quantity.

So, that is why this naive algorithm will work only for the small values of n and it is still a big open problem or to come up with efficient algorithms or feasible algorithms to check whether two graphs are isomorphic or not.

(Refer Slide Time: 22:45)

How to Show Two Graphs are Not Isomorphic

□ Graph invariant properties --- Properties preserved by isomorphic graphs

- Ex: Number of vertices, edges, number of vertices of same degree, etc
- ❖ Violation of graph invariant properties \Rightarrow non-isomorphic graphs
- ❖ Graph invariant properties are only necessary for isomorphic graphs

G_1

G_2

- ❖ a must correspond to either t or u or x or y
- ❖ Each of t, u, x, y is adjacent to another vertex of degree 2, which is not true for a

So, how do we verify whether two graphs are not isomorphic or not and we can verify whether two graphs are not isomorphic or not by checking for graph invariant properties. So, what are graph invariant properties? They are the properties which should be preserved by isomorphic graphs, that means these are the properties which should be there both in graph G_1 as well as in graph G_2 if at all they are isomorphic, that means if any of these properties is violated then you can declare that the two graphs are not isomorphic.

So, some of the naive graph invariant properties which should be preserved by the isomorphic graphs are the following: they should have the same number of vertices, same number of edges number of vertices of a particular degree should be the same in both the graphs and so on. So, for instance if in your graph G_1 , there are two vertices of degree 2, then in G_2 also there should be exactly two vertices of degree 2.

Otherwise, the two graphs can never be isomorphic because you would like to associate a vertex of degree 2 in G_1 with another vertex of degree 2 in G_2 and vice versa. However, it turns out that if any of these graph invariant properties is violated and you can immediately declare that the two graphs are not isomorphic. However, it turns out that the graph invariant properties are the only necessary condition for the existence of isomorphic graphs.

And we do not have an exhaustive list of graph invariant properties, we do not know that all these properties or you do not have a list of properties such that if those properties are preserved both in graph G_1 and G_2 , then you can declare that the two graphs are isomorphic. Unfortunately, we do not have such graph invariant properties. So, for instance if I take these two graphs G_1 and G_2 , then it is slightly difficult to identify a graph invariant property which is present in G_1 but not present in G_2 .

In fact the graph G_1 and G_2 here are not isomorphic and we have to identify here a graph invariant property which is present in one graph but not present in the other graph. So, if you see closely here, if at all graph G_1 is isomorphic to graph G_2 , then I need to associate the vertex a of G_1 with some vertex in G_2 and that association or the vertex which could be associated with vertex a could be either the vertex t or the vertex u or the vertex x or the vertex y .

Why so? Because the degree of a here is 2, so it can be associated only with a vertex of degree 2 in G_2 and the only vertices of degree 2 in G_2 are t, u, x and y. You cannot associate vertex a with vertex s because a has degree 2 where s has degree 3 and so on. But it turns out that in graph G_2 the vertex t, the vertex u, the vertex x, the vertex y all of them are adjacent to another vertex of degree 2. So, for instance; if you take t, t is adjacent to u and u has degree 2.

If you take u, u is adjacent to t and t has degree 2. If you take x, x is adjacent to y which has degree 2 and similarly y is adjacent to x which has degree 2. But if you see graph G_1 , all the neighbors of the node a have degree 3, so b has degree 3 and t has degree 3. So, that means this is now a graph invariant property which is present in G_2 but not present in G_1 , the graph invariant property is that all the vertices of degree 2 in G_2 , they have a neighbour which also have degree 2.

But that property is not satisfied in G_1 , namely I have a vertex of degree 2, namely the vertex a and none of its neighbours has degree 2. So, that shows that I can never find out isomorphism or a bijection between the vertex set of G_1 and the vertex set of G_2 because I cannot find a vertex corresponding to the vertex a in the graph G_2 .

(Refer Slide Time: 27:50)

Connectivity in a Graph

Path of length n from node u to node v in a graph G (n : non-negative integer)

➤ Sequence of n edges e_1, e_2, \dots, e_n of G , such that:

- ❖ $e_1 = (x_0, x_1), e_2 = (x_1, x_2), \dots, e_i = (x_{i-1}, x_i), \dots, e_n = (x_{n-1}, x_n)$ and
- ❖ $x_0 = u$ and $x_n = v$

A path from u to v is also called a walk from u to v

If $u = v$, then the path is called a circuit (closed walk)

Simple Path (trail)

➤ A path with no repeated edges ($e_1 \neq e_2 \neq \dots \neq e_n$)

Simple Circuit : circuit with distinct edges

Now let us next define the connectivity in a graph, so for that we first recall the definition of a path of length n between the node u and the node v in an undirected graph, where n is a non-negative

integer. So, a path of length n between the node u and the node v is a sequence of n edges in the graph and say $e_1 = (x_0, x_1)$, $e_2 = (x_1, x_2)$, \dots , $e_i = (x_{i-1}, x_i)$, \dots , $e_n = (x_{n-1}, x_n)$ such that the starting vertex or the first end point of the edge e_1 is u and x_n is equal to v . If that is the case then I say that I have a path of length n between the node u and the node v . So, remember there is no restriction on whether the edges in this sequence of n edges are distinct or they are allowed to be repeated or not, no such restriction is there when I consider a path of length n .

A path is also called as a walk from u to v , because we are just traversing edges and going from the node u to node v , if the starting vertex u and the end vertex v of the path are same, then the path is also called as a circuit or a closed walk and if all the edges in my path between u to v are distinct, ($e_1 \neq e_2 \neq \dots \neq e_n$). That means no edges are repeated then I call the path as a simple path and if I have a circuit where the starting vertex and the end vertex are the same and if all the edges are distinct, then the circuit is called a simple circuit.

By the way, in a simple path the vertices are allowed to be repeated, it is only the edges which are not allowed to be repeated.

(Refer Slide Time: 29:57)

Connected Graphs and Components

□ An undirected graph $G = (V, E)$ is **connected**, if there exists a path between every pair of **distinct vertices** in the graph

□ **Connected component** of a graph $G = (V, E)$

- ❖ **Maximal connected subgraph** of G
- ❖ A **connected subgraph** of G , which is **not a proper subgraph** of another connected subgraph of G

So, once we have the definition of a path, let us now define what we call connected graphs and components, so an undirected graph is called connected if there exists a path between every pair of distinct vertices, I stress here between distinct vertices, I am not interested to check whether

there exists a path from a node to itself, but between every pair of distinct vertices there should exist at one path, there could be multiple paths as well but at least one path should be there between every pair of distinct vertices.

Now what is the connected component of a graph? A connected component of a graph is the maximal connected subgraph of the graph. What does the maximal connected subgraph means, well it is a connected subgraph of the graph G and it is maximal in the sense that it cannot be further extended. What does that mean? What does it mean that I cannot further extend that connected subgraph, well that means that you cannot have another subgraph of the graph G which is also connected such that the connected subgraph is a proper subgraph of that connected subgraph.

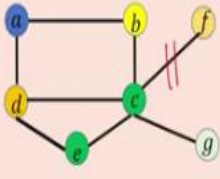
That means, it is maximal in the sense that it is not a proper subgraph of any other connected subgraph of the G . So, what does this mean? Imagine I take this graph, then can I call this as the connected component and can I call this graph as the connected component. So, if I take this triangle involving d , c and e , it is not a connected component because even though it is a connected subgraph of your graph G , it is a proper subgraph of this connected subgraph of the G , so that is why this is not maximally connected.

So, that is why the connected component of this graph is the original graph itself, because the whole graph itself is connected at the first point, so if my graph G is connected at the first point then the only connected component of the graph will be G itself. Whereas if my graph G is disconnected, then the connected component will be the collection of maximal connected subgraphs.

(Refer Slide Time: 32:46)

Cut Vertex and Cut Edge

Cut Vertex (articulation point)



- ❖ How "critical" is the node c ?
- ❖ Deleting node c disconnects the graph
- ❖ Vertex v is a **cut vertex** for a **connected graph** if the number of connected components of $G - v$ is at least one more than that of G

Cut Edge (bridge)

- ❖ How "critical" is the edge (c, f) ? --- deleting it disconnects the graph
- ❖ An edge e is a **cut edge** for a **connected graph** if the number of **connected components of $G - e$** is at least one more than that of G

Now let us next define cut vertex and cut edge. So, cut vertex are also called as articulation point or critical vertices, what does it mean when I say a vertex is critical, so if I take this graph G , the node c is very critical here, because if I delete this node c then it will disconnect the entire graph. So, in that sense it is a critical vertex or articulation point, so my definition here is the following: I will say a vertex v in a graph G is a cut vertex and this cut vertex is defined with respect to a connected graph.

So, this vertex v will be called as a cut vertex if deleting the vertex will disconnect the graph or equivalently the number of connected components of the graph $G - v$ is at least one more than that of G , because if the number of connected components increases for the graph $G - v$, then that is possible only if my graph $G - v$ becomes disconnected. Because I started with a connected graph and even after deleting the vertex v , my new graph still has only one connected component namely it is still connected. That means the vertex v is not a cut vertex, that means deleting the vertex v does not disconnect the graph. Similarly, I can define a critical edge which is also called as a bridge or cut edge. So again, if I take the same graph as above and focus on this edge c and that connecting the node c and f , then the edge connecting the node c and f is very critical because if I delete that edge then the whole graph gets disconnected.

So, that gives me the definition of a cut edge, I will say an edge is a cut edge for a connected graph if deleting that edge disconnects the graph which is equivalent to saying that the number of

connected components of the reduced graph, namely the graph obtained after removing the edge e is at least one more than the number of connected components of G . So, the number of connected components of G was 1 because it was a connected graph, but now in the new graph namely in the graph $G - e$, the number of connected components is 2 or more than 2. That means my graph got disconnected because of deleting the edge e and hence I will call the edge e as a critical edge or a bridge.

(Refer Slide Time: 35:29)

References for Today's Lecture



So, that brings me to the end of this lecture, these are the references. Just to summarize, in this lecture we discussed various set theoretic operations on the graph. We discussed the various data structures which we can use to represent graphs and we discussed the cut vertex and the cut edge. Thank you!