

Lecture - 52
Vertex and Edge Colouring

(Refer Slide Time: 00:26)

Lecture Overview

- Vertex coloring
 - ❖ Vertex chromatic number
- Edge coloring
 - ❖ Edge chromatic number

Hello everyone, welcome to this lecture; the plan for this lecture is as following. We will discuss about vertex colourings, vertex chromatic number and we will discuss about edge colouring and edge chromatic number.

(Refer Slide Time: 00:33)


Vertex Coloring

□ Exam Time-table scheduling

- ❖ Minimum number of time-slots needed to conduct exams for n subjects such that no student has two exams in the same time-slot

□ Graph-theoretic modelling

- ❖ n subjects form the nodes of the graph : $\{1, 2, \dots, n\}$
- ❖ Edge (i, j) if there is a student who has registered for both subjects i and j
- ❖ Assign colors to the nodes so that no two adjacent nodes get the same color
- ❖ Number of time-slots = minimum number of colors needed to color the vertices



4 slots needed

So, let us start with vertex colouring first and let us see some real world motivation for studying the vertex colouring problem; the problem is that of exam time table scheduling. So, there are n subjects in a college. Imagine there is a college with n subjects; multiple students

taking those subjects and we need to schedule the exams for those n subjects and we need to schedule the exams in such a way that it should not happen that a student has 2 exams in the same time slot appearing in the schedule.

So, for instance if I am a student and if I have taken subject number 1 as well as subject number 2 then it should not happen that I get a schedule where subject number 1 and subject number 2; their exams are scheduled on the same day and at the same time slot so that should not happen. So, one obvious way of doing the scheduling is that you schedule one exam in exactly one time slot and that will require you n time slots but we do not want to do that because that will be an overkill and that might be a wastage of resources.

Instead we would require or we would be interested to find out the minimum number of time slots where I can allow multiple exams to be conducted in the same time slot, but without violating the condition that no student has two exams in the same time slot. It should not happen that I schedule the exams in such a way that a student who has taken two subjects and both the subjects appear in the same time slot in my schedule that should not happen. So, how do we model this problem as a graph theoretic problem? So, the n subjects will form the n nodes of my graph.

And what will be the edge set so I will add an edge between node number i and node number j which you can interpret as subject number i and subject number j . So, subject number i and subject number j will have an edge among them if there is at least one student who has registered both for the subject i as well as the subject j . If there is no student who has taken both subject i as well as subject j that means I can interpret or treat subject i and subject j as kind of independent subjects.

And I can conduct the exam for both subjects number i as well as subject number j in the same time slot. But if there is an edge between subject number i and subject number j that means I cannot schedule the exam for subject number i as well as the subject number j in the same time slot. So, how do I model that requirement? So, basically I will be interested to colour the nodes of the graph by various colours.

And my colouring should satisfy the condition that no two adjacent nodes should get the same colour. And the number of time slots or minimum number of time slots I require is

same as the minimum number of colours needed to colour the vertices. Of course a trivial way to colour the vertices will you take n distinct colours and assign one distinct colour to each of the n vertices. I do not want to do that but because that is equivalent to saying that I conduct exams for the n subjects taking n slots.

I might like to assign the same colour to multiple vertices provided this condition is satisfied. And the minimum number of colours that I need to colour the vertices will give me the minimum number of time slots so for instance if this graph is given to me. So, I have subject number s_1 here, s_2 to here, s_3 , s_4 , s_5 , s_6 , and s_7 I have seven subjects. So, I can take 7 time slot and in each time slot I can schedule 1 exam I do not want to do that.

It turns out that just with 4 slots I can finish off all the 7 exams namely subject number 7 and subject number 4 can be scheduled together because there is no student who has taken simultaneously the subject number 7 and subject number 4. In the same way, subject number 5 and subject number 3 can be scheduled together because there is no student who has taken both the subjects and so on.

(Refer Slide Time: 05:20)

Vertex Coloring Problem

☐ Input: A simple graph

☐ Output: An assignment of a color to each vertex so that no two adjacent vertices are assigned the same color

☐ Vertex-chromatic-number of a graph ($\chi_0(G)$): χ

- ❖ Minimum number of colors needed to color the vertices of the graph

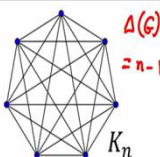
☐ Finding $\chi_0(G)$ for an arbitrary graph is a hard problem

☐ Upper bound on $\chi_0(G)$?

$\chi_0(G) \leq \Delta(G) + 1$

 $\Delta(G)$: maximum degree of any vertex in G

Will give an algorithm which needs at most $\Delta(G) + 1$ colors for any arbitrary G



$\Delta(G) = n-1$
 K_n

So, now coming to the vertex colouring problem what is the input? The input here will be a simple graph it may or may not be connected and the output is basically an assignment of a colour to each vertex such that no two adjacent vertices are assigned the same colour, What is the vertex chromatic number of a graph? So we denote the vertex chromatic number of a graph by this quantity $\chi_0(G)$ this is a Greek character.

If you do Latex then this is the character χ . It is pronounced as “khi” not as “chi”. So, it is what is the vertex colour chromatic number of a graph. The vertex chromatic number is the minimum number of colours needed to colour the vertices of the graph such that no two adjacent vertices are assigned the same colour. It turns out that finding the chromatic number of a graph is indeed a hard problem. Hard problem in the sense we do not have in general efficient algorithms or practical algorithms for finding out the vertex chromatic number of a graph, if the number of vertices n in the graph is arbitrarily large or very large. Of course I can run exponential time algorithms, exponential in the number of vertices, and get vertex chromatic number. What I mean by hard problem is basically we do not have efficient algorithm, that is a very loose definition of a hard problem. So, what is an upper bound on the vertex chromatic number?

It turns out that the vertex chromatic number of a graph is always upper bounded by $1 + \Delta(G)$, the maximum degree of any vertex in your graph. You do not require more than these many number of colours in your graph to colour the vertices of the graph and to check whether indeed this bound is true or not just take the case of a complete graph of n vertices where the degrees of all the vertices are same.

And where $\Delta(G)$ namely the maximum degree is $n - 1$. So, for colouring all the n vertices in a complete graph we will need indeed n colours because there is an edge between every pair of nodes in the graph and that satisfies this upper bound. So, what we will do is we will give an algorithm which will indeed need at most these many number of colours to colour all the vertices of the graph. But that need not be the optimal colouring because it might be possible that your graph may not need $\Delta(G) + 1$ number of colours. So, in that sense my algorithm need not be an optimal algorithm.

(Refer Slide Time: 08:18)

Greedy Algorithm for Vertex Coloring

❑ Idea: Use the first available color at every step (if possible)

- ❑ Repeat till there are uncolored vertices in the graph

 - ❖ Arbitrarily choose an uncolored vertex, say v_i
 - ❖ Let T be the set of colors used till now for coloring
 - If v_i can be colored using one of the colors from T , then assign the least indexed color from those colors to v_i
 - Else assign a new color to v_i and add this new color to T

$T = \{c_1, c_2, \dots, c_k\}$
 c_{k+1}
- ❑ Will this algorithm always give the optimal coloring ?

 - ❖ Depends upon the sequence in which we choose the vertices for coloring

So, this algorithm is based on the greedy strategy which is a very popular strategy in algorithm designs. So, what exactly is the greedy strategy here: the greedy strategy is that you use the first available colour at every step if possible, if not possible then use a new colour. So, more specifically the algorithm is an iterative algorithm and in each iteration we will pick a new vertex for colouring.

And when we are picking the new vertex for colouring we have to follow a greedy strategy to decide whether we can use a new colour for colouring the next vertex or whether you should use an existing colour which you might have already assigned or given to some of the existing vertices depending upon some conditions. So, to be more specific we do the following repeat while loop so till we have uncoloured vertices left in the graph we do the following.

We arbitrarily choose any uncoloured vertex which is not yet assigned any colour. So, there might be several uncoloured vertices still left in your graph I arbitrarily choose one of them. And let T be the set of colours which I have used till now for colouring the various vertices in the graph. So, to begin with my T will be empty and all the n vertices are uncoloured. So, I will start with any of the vertex and my T will be empty.

But in general as my algorithm proceeds my T will keep on taking new values depending upon what colours I have chosen for various vertices. So, imagine I am at my current iteration where I have decided to colour the vertex number v_i because it has not been assigned any colour. Now what I will do is I will check that is it possible to use one of the existing colours

from the set T and assign it to the vertex v_i in the sense that there is no vertex incident with the vertex v_i or which is adjacent to the vertex v_i to be more specific, and that has been assigned the same colour which I am considering from the set T . That means your set T might have already taken the colour c_1 to c_k . So, you have already used k number of colours and then you will like to check whether it is possible to assign the colour number c_1 to the vertex v_i or not without violating the vertex colouring requirement if not then check for c_2 if not then check for c_3 and so on.

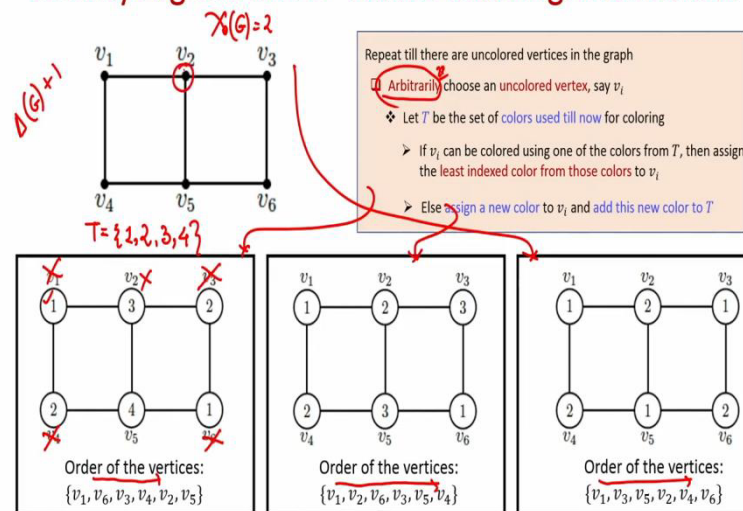
So, if at all you stuck or you find some colour which can be assigned to the vertex v_i and if there are multiple colours from the set T which can be assigned to the vertex v_i without violating the vertex colouring requirement then assign the least index colour from the set T to the vertex v_i . If not then pick a new colour which is different from all the colours c_1 to c_k and assign it to the vertex v_i and add that colour c_{k+1} to the set of colours T and repeat this process that is the idea here.

So, now the question is will this algorithm always give the optimal colouring and what do I mean by optimal colouring? By optimal colouring I mean the minimum number of colours that is indeed required to colour all the vertices of my graph namely the number of colours is exactly the vertex chromatic number. So, it turns out that this algorithm may not always give you the optimal colouring because it depends upon the sequence in which you pick the uncoloured vertices in every iteration.

So, remember in each iteration we are arbitrarily picking one vertex from my current set of uncoloured vertices and then deciding to colour it. Now depending upon in what order you pick it you may or may not get an optimal colouring.

(Refer Slide Time: 12:42)

Greedy Algorithm for Vertex Coloring: Illustration



So, let me demonstrate my point here so imagine this is a graph given to you and we want to assign colours to the vertices by following this algorithm. So, if I follow the vertex ordering $\{v_1, v_6, v_3, v_4, v_2, v_5\}$ then I will need 4 colours why so? So suppose I start with the vertex number 1 my set T is empty so I will assign colour number 1 then as per my sequencing vertex v_1 is done then I have 5 vertices left.

So, suppose I decide to colour vertex number 6. Vertex number 6 can be assigned one of the colours from my set T . So, remember my set T has now taken colour number 1 so I can assign colour number 1 to vertex number 6. So, vertex 6 is also done. Now I am left with 4 vertices my T has only 1 colour as of now. Then out of the 4 vertices which are not yet coloured suppose I decide vertex number 3. Now vertex number 3 cannot be assigned colour number 1 because colour number 1 has been assigned to vertex 6 which is adjacent to vertex 3.

So that is why now I have to use a new colour for vertex number 3; it is done. Now out of the 3 vertices as per my sequencing I am deciding to colour vertex 4. The vertex 4 cannot be coloured with colour number 1 because colour 1 has been assigned to vertex 1 which is adjacent to vertex 4 but colour 2 can be assigned to vertex 4 so I do not need a new colour, vertex 4 done.

Now I am left with 2 vertices, vertex 2 and vertex 5. Suppose I choose vertex 2 then I cannot use colour 1 I cannot use colour 2 so I have to add a new colour give it to vertex 2. And now only vertex left is vertex 5 which cannot be given colour 1 which cannot be given colour 2

which cannot be given colour 3. So, the only option is to give a new colour namely colour number 4. So, if this is the order in which I picked the vertices in every iteration I will need 4 colours.

On the other hand imagine that I choose the vertices in this order $\{v_1, v_2, v_6, v_3, v_5, v_4\}$ then it is easy to see that I will be needing 3 colours which is 1 less than the number of colours that I used in the previous ordering, whereas, if I use this ordering $\{v_1, v_3, v_5, v_2, v_4, v_6\}$ then I need just 2 colours. And it is easy to see that 2 is indeed the vertex chromatic number of this graph. Indeed you need 2 colours here because you have at least 1 edge. So, you cannot give the same colour to the endpoints of an edge.

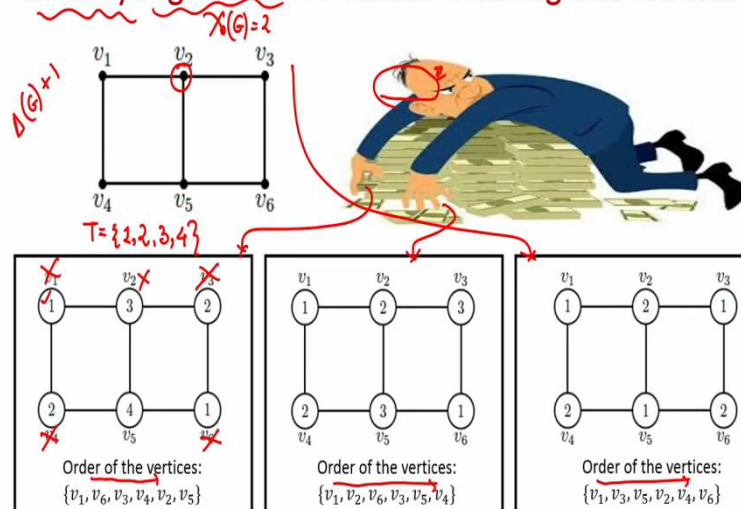
So, definitely 2 is the minimum number of colours needed to colour all the vertices of this graph. And indeed I am giving you now colouring which requires 2 colours here. So, the vertex chromatic number of this graph is 2. And I can get an optimal colouring from this algorithm but this algorithm may also give me non optimal colouring namely it can give me a colouring where I required 4 colours or this algorithm can also give me a colouring which requires me to use 3 colours.

And this algorithm can also give me the optimal colouring. Now I do not know in what order I should use or decide the next set of uncoloured vertices because if my graph has million number of nodes to structure the pictorial representation of the graph may not be given to me. In this example you can say it is having 6 vertices I can always follow this optimal strategy but imagine that you are given an arbitrary graph which has a million number of nodes.

And you are not given a pictorial representation of the graph but just say that adjacency matrix or adjacency list representation then in every iteration you have to just pick the edges arbitrarily. And it may turn out that your ordering does not lead you to the optimal colouring. What this algorithm guarantees is that you do not need more than maximum degree plus 1 number of edges why so? So, what is our maximum degree in this graph? The maximum degree is 3 and this algorithm guarantees you that you do not need more than 4 colours. So, let us argue that formally.

(Refer Slide Time: 17:28)

Greedy Algorithm for Vertex Coloring: Illustration



So, what this algorithm basically says is that the greedy approach is not always optimal. So that is why a greedy approach is used very carefully in algorithm design. There are algorithms which can be always solved and greedy approach can always give you optimal solutions, but vertex colouring is not one such problem instance. The greedy approach may not give you the optimal colouring here.

(Refer Slide Time: 17:56)

Greedy Algorithm for Vertex Coloring

- Repeat till there are uncolored vertices in the graph
 - ❖ Arbitrarily choose an **uncolored vertex**, say v_i
 - ❖ Let T be the set of **colors used till now** for coloring
 - If v_i can be colored using one of the colors from T , then assign the **least indexed color from those colors** to v_i
 - Else **assign a new color** to v_i and **add this new color to T**
- The above algorithm needs **at most $\Delta(G) + 1$ colors**
 - ❖ While coloring v_i , it can have **at most $\Delta(G)$ colored neighbors**
 - **Either a new color** may be needed to color v_i
 - Or an **existing color from T — colors of the neighbors of v_i** can be used

So, what we now want to argue here is that if we follow the greedy algorithm for vertex colouring then even if you do not get optimal colouring at most the colouring that you obtain will require you to use maximum degree plus 1 number of colours, You do not need more than these many number of colours. And the proof is very simple. The proof is based on the fact that when you are deciding a colour for a vertex v_i .

Then there are 2 possible cases: either the vertex v_i is the vertex which has the maximum degree and say that all the neighbours of that vertex v_i have been already coloured. So, in that case it can have already $\Delta(G)$ colours used, in which case you have to use a new colour for the vertex v_i or it might be possible that you can use one of the existing colours from your currently used set of colours T because remember that the colours of the neighbours of the node v_i can be used.

So, in any case it does not matter whether you are in this case or in this case you would need more than $\Delta(G) + 1$ number of colours. That is a simple proof for the fact that this greedy algorithm will require at most these many number of colours.

(Refer Slide Time: 19:21)

Edge Coloring

☐ Round-robin tournament scheduling

- ❖ n teams $\{1, 2, \dots, n\}$ --- for simplicity, let n be even
- ❖ Every team has to play against each other once
- ❖ A team does not want to play more than one game on any day ✓✓}}
- ❖ Goal: to come up with a fixture, requiring minimum number of days

☐ Graph-theoretic formulation :

- ❖ Input: complete graph K_n
- ❖ Output: an edge coloring with least number of colors
- Adjacent edges e_i and e_j assigned different colors

So that is all about vertex colouring. Now let us see a related concept which is called as edge colouring. And again let us first to see a motivation for studying this problem and then we will discuss the general theory. So, the motivation here is how to schedule a round robin tournament. So, what do we mean by that? Imagine you have n teams say n cricket teams representing n countries and again for simplicity assume n to be even but that is not necessary.

And in a round robin tournament each team has to play against each other. And then based on the results we decide the semi final and then final. So, now we want to schedule the matches, so it is possible to schedule all the matches in 1 single day. So, you will be having $\frac{n(n-1)}{2}$

number of matches and you may schedule all the matches. In that case each team has to play multiple matches but you do not want to put too much stress on the teams.

So, you want to schedule the matches in such a way that no team is forced to play more than a single game on any day. And the goal is to come up with a schedule so that you finish all the required matches satisfying this condition in minimum number of days; again you can do the following: each day just schedule 1 match. And in that case you will require $\frac{n(n-1)}{2}$ number of days to schedule all the matches.

But that will be overkill because audience may not be so much patient, you might schedule multiple matches in parallel and simultaneously satisfy this requirement. So, what will be the graph theoretic formulation here so the input will be a complete graph with n nodes and output will be an edge colouring with least number of colours and what is an edge colouring here? We want to now colour the edges not the vertices.

And we want to colour the edges in such a way that if you have an edge e_i and e_j which are adjacent, adjacent in the sense they are incident on a common node then they should get different colours you cannot assign the same colour simultaneously to such edges e_i and e_j . So, for instance if I take a complete graph of 4 nodes then I can do the colouring in the following way. I can colour the edge between 1 and 4 and 2 and 3 with the same colour.

Because those 2 edges are not adjacent because their endpoints are completely distinct. I can assign the same colour to the edges between 4 and 3 and 1 and 2 because the endpoints are completely distinct. And I can assign the same colour to the edges between 1 and 3 and 4 and 2. So, I need 3 colours here and that corresponds to the fact that I can do the scheduling in 3 days: the first day we can schedule the match between team number 1 and 4 and team number 2 and 3 that would not force any of the teams to play more than 1 match.

On day number 2 you schedule the match between team 4 and 3 and team 1 and 2. Again that would not force any team to play more than 1 match. And on the third day you schedule the match between team 4 and 2 and team 1 and 3.

(Refer Slide Time: 23:11)

Edge Coloring Problem

❑ Input: A graph without loops

❑ Output: An assignment of a color to each edge so that no two adjacent edges (incident with a common vertex) are assigned the same color

❑ Edge-chromatic-number of a graph ($\chi_1(G)$):

❖ Minimum number of colors needed to color the ~~vertices~~ ^{edges} of the graph

❑ Finding $\chi_1(G)$ for an arbitrary graph is a hard problem


❑ Lower bound on $\chi_1(G)$?

$\chi_1(G) \geq \Delta(G)$

$\Delta(G)$: maximum degree of any vertex in G

❖ Consider the vertex v with degree $\Delta(G)$

➤ Need $\Delta(G)$ colors for the edges incident with v



So, now coming to the general problem of edge colouring you are given a graph without loops. And what we want here as an output? We want to output an assignment of colour to the each edges of the graph so that no two adjacent edges and by adjacent edges I mean who have common incident vertex. So, I need a colouring of the edges in such a way that no two incident edges are assigned the same colour.

So, like vertex chromatic number we have a related concept called edge chromatic number and this is denoted by χ_1 . So, χ_0 was for vertex chromatic number and χ_1 is for edge chromatic number. So, what is edge chromatic number? Edge chromatic number of a graph is the minimum number of colours needed to colour the edges of the graph satisfying this condition that no two adjacent edges are assigned the same colour.

And again like vertex colouring, finding the edge chromatic number of an arbitrary graph with large number of vertices is a hard problem, you do not have efficient algorithms or practical time algorithms for finding the minimum number of colours. Of course you can do a brute force and try to see whether 1 colour is sufficient, 2 colour is sufficiently, 3 colour is sufficient, 4 colour is sufficient, 5 colour sufficient.

And then you will hit upon the right answer but that will require you enormous amount of time so that is not an efficient algorithm. Now can we find a lower bound on the edge chromatic number that means what can I say that definitively these many colours are indeed required: it turns out that the lower bound is nothing but the maximum degree you take the vertex which has the maximum degree say the vertex v has the maximum degree.

So, I have the vertex v and it has the maximum degree and how many edges are there? $\Delta(G)$ number of edges are incident with the vertex v . So, definitely I need these many number of colours to colour all the edges incident with the vertex v because none of these edges can be assigned the same colours because all of them are incident with a common vertex namely v . Definitely these many number of colours are required but I may need more than these many colours as well.

(Refer Slide Time: 26:12)

Edge Coloring Problem

❑ Input: A graph **without loops**

❑ Output: An assignment of a color to each edge so that **no two adjacent edges** (incident with a common vertex) **are assigned the same color**

❑ **Edge-chromatic-number** of a graph ($\chi_1(G)$):

❖ **Minimum** number of colors needed to color the vertices of the graph

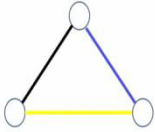
❑ Finding $\chi_1(G)$ for an arbitrary graph is a **hard problem**

❑ **Upper bound on $\chi_1(G)$?**

$\Delta(G) \leq \chi_1(G) \leq \Delta(G) + 1$

Gupta-Vizing theorem

For simple graphs

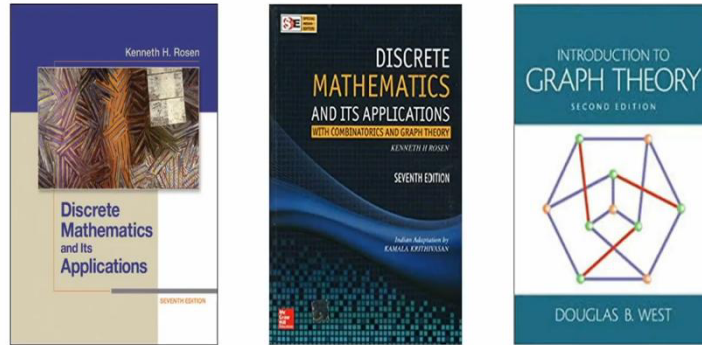


And what can I say about upper bound so there is a very interesting theorem called as the Gupta-Vizing theorem which says that if you have a simple graph then you do not need more than these many number of colours: $\Delta(G) + 1$. So, basically we get a range of values for edge chromatic number the lower bound was the maximum degree and upper bound is 1 plus the maximum degree.

Now finding the exact value is the hard problem. So, again if you want to verify whether this theorem is true or not take the case of a triangle graph where you need 3 colours basically to colour all the edges of the graph. So, due to interest of time I am not going into the exact proof of the Gupta Vizing theorem because it is slightly advanced but if you are interested you can refer to any standard reference.

(Refer Slide Time: 27:11)

References for Today's Lecture



□ Professor S. A. Choudum's NPTEL course on graph theory

So, these are the references for today's lecture. And with that I conclude today's lecture; just to summarize in this lecture we introduced the problems of vertex colouring and edge colouring. We saw the notion of vertex chromatic number, edge chromatic number, we discussed greedy algorithm for vertex colouring which may not give you the optimal colouring always. And we discussed various bounds for vertex chromatic number and edge chromatic number. Thank you.