Module -III

1) For each of the graphs below, complete the following.
(a) Find the chromatic number χ(G). Include an argument why fewer colors will not sufficient.
(b) Find the chromatic index χ'(G).
(c) Determine which graphs are perfect. Explain your answer.

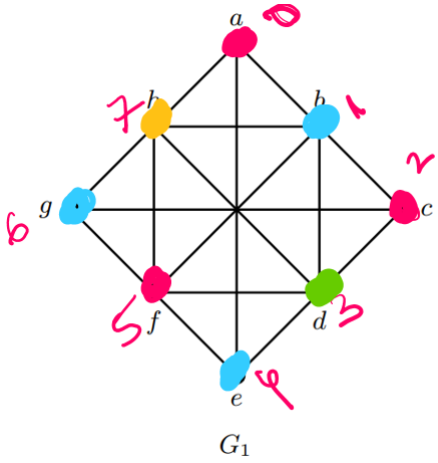
Answer:

a)

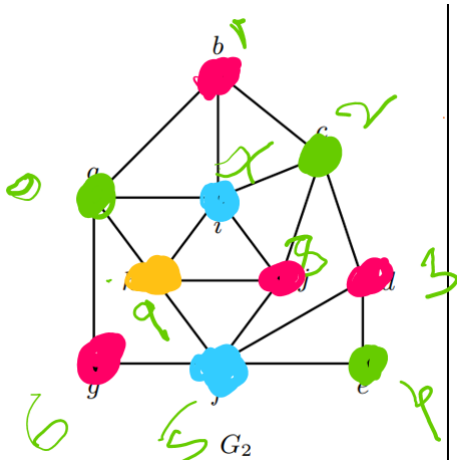
0

The *chromatic number* of a graph G, denoted by $\chi(G)$, is the minimum number of colors needed to color the vertices of G in such a way that no two adjacent vertices receive the same color. Clearly $\chi(G)$ is bounded from below by the size of a largest clique in G, denoted by $\omega(G)$. In 1960, Berge introduced the notion of a perfect graph. A graph G is *perfect*, if for every induced subgraph H of G, $\chi(H) = \omega(H)$.

The chromatic number X(G) for G1 and G2 is 4



G_1



G_2

b) **Chromatic index:** The ***chromatic index*** χ'(G) : χG is the least number of colours needed to colour the edges of GG so that any two edges that share a vertex have different colours.

The chromatic index is also 5 for both the graphs

c) A graph is considered perfect if, for every induced subgraph, the chromatic number equals the size of the largest clique. This property is known as the strong perfect graph theorem, proven by Maria Chudnovsky, Neil Robertson, Paul Seymour, and Robin Thomas in 2002.

For a graph is perfect:

- Identify Cliques:
- Calculate Chromatic Number:
- Compare Sizes: Compare the size of the largest clique in the graph with the chromatic number. If they are equal for every induced subgraph of the graph, then the graph is perfect.
- Repeat for Subgraphs: Repeat steps 1-3 for every induced subgraph of the original graph. An induced subgraph is a subset of vertices and the edges that connect them, where all edges between the selected vertices in the original graph are also present in the induced subgraph.
- Conclusion: If the chromatic number equals the size of the largest clique for every induced subgraph, then the graph is perfect. Otherwise, it is not perfect.

First graph is perfect graph since the $x(H)=w(H)=4$

Second graph is not perfect graph since chromatic number is 4 but the largest clique size is 3

2) Explain with an example the minimum number of colors needed to properly color the vertices of any planar graph according to the rules of vertex coloring?

The minimum number of colors needed to properly color the vertices of any planar graph is four. This is known as the Four-Color Theorem, which states that any planar map can be colored using at most four colors in such a way that no two adjacent regions (countries) have the same color.

Here's a simple example to illustrate this:

Imagine a planar graph resembling a map of countries. Each country is represented by a vertex, and if two countries share a border, there's an edge between their corresponding vertices. The

goal is to color each country (vertex) in such a way that no two adjacent countries (connected vertices) share the same color.

To illustrate the Four-Color Theorem, consider a simple map of four countries arranged in a square: A, B, C, and D. Each country shares a border with two others.

To color this map the fewest colors according to the four color theorem, country A with one color say red . Then, since A and B share a border, B must be colored with a different color, say blue. Similarly, since B and C share a border, C must be colored with a third color, say green. Finally, D, which shares borders with both A and C, must be colored with the fourth color, say yellow.

3. Explain edge coloring and how does it differ from vertex coloring?

Edge coloring and vertex coloring are both concepts in graph theory used to assign colors to the elements of a graph, but they target different components of the graph.

Vertex Coloring:

- In vertex coloring, the objective is to assign colors to the vertices of a graph such that no two adjacent vertices (connected by an edge) have the same color.
- The minimum number of colors required to vertex-color a graph is called its chromatic number.
- Vertex coloring is often used to solve scheduling problems, register allocation in compilers, and various other optimization problems.

Edge Coloring:

- In edge coloring, the aim is to assign colors to the edges of a graph such that no two adjacent edges (sharing a common vertex) have the same color.
- The minimum number of colors required to edge-color a graph is called its chromatic index.
- Edge coloring is employed in scheduling problems, frequency assignment in telecommunications, and designing maps.

Differences:

Target:

- Vertex coloring focuses on coloring the vertices.
- Edge coloring focuses on coloring the edges.

Constraint:

- **In vertex coloring**, the constraint is that adjacent vertices must have different colors.
- **In edge coloring,** the constraint is that adjacent edges must have different colors.
- **Chromatic number** refers to the minimum number of colors needed to vertex-color a graph.
- **Chromatic index** refers to the minimum number of colors needed to edge-color a graph.

Applications:

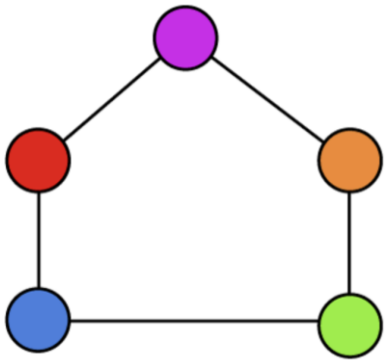
- Vertex coloring is used in problems where resources (vertices) need to be allocated efficiently without conflicts.
- Edge coloring is used in problems where interactions or conflicts occur between pairs of resources (edges).

4) Explain chromatic number in the context of vertex coloring with an example?

The chromatic number of a graph, denoted by $\chi(G)$, is the minimum number of colors needed to color the vertices of a graph such that no two adjacent vertices (connected by an edge) share the same color. In simpler terms, it's the minimum number of colors required to vertex-color a graph without any adjacent vertices having the same color.

Consider the following graph:

In this graph, vertices A, B, C, D, and E are represented by the letters A, B, C, D, and E respectively. The edges are represented by lines connecting the vertices.

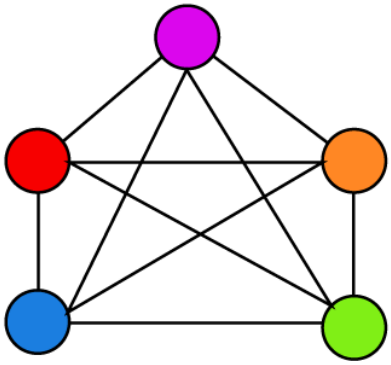


- The same color is not used to color the two adjacent vertices.
- The minimum number of colors of this graph is 3, which is needed to properly color the vertices.
- Hence, in this graph, the chromatic number = 3
- If we want to properly color this graph, in this case, we are required at least 3 colors.

Chromatic number:

- The chromatic number in a cycle graph will be 2 if the number of vertices in that graph is even.

- The chromatic number in a cycle graph will be 3 if the number of vertices in that graph is odd.
- In a planner graph, the chromatic Number must be Less than or equal to 4.
- In a complete graph, the chromatic number will be equal to the number of vertices in that graph.



- In any bipartite graph, the chromatic number is always equal to 2.
- In any tree, the chromatic number is equal to 2.

Explain the procedure for First-Fit coloring algorithm with an example graph?

First-Fit Coloring Algorithm

Input: Graph G with vertices ordered as $x_1 \prec x_2 \prec \cdots \prec x_n$.

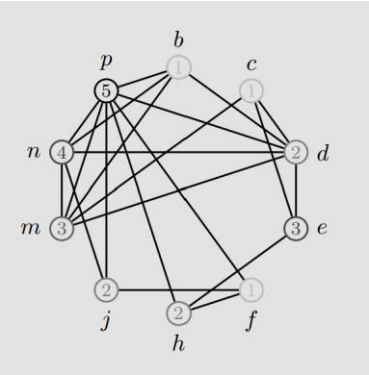
Steps:

1. Assign x_1 color 1.
2. Assign x_2 color 1 if x_1 and x_2 are not adjacent; otherwise, assign x_2 color 2.
3. For all future vertices, assign x_i the least number color available to x_i in $G[x_1, \ldots, x_i]$; that is, give x_i the first color not used by any neighbor of x_i that has already been colored.

Output: Coloring of G.

- Initialization: Begin with an empty list of colors and an empty dictionary (or array) to keep track of the colors assigned to each vertex.
- Start with the first vertex: Choose any vertex from the graph to start the coloring process.

- Assign color: Assign the first available color to the chosen vertex. The "first available color" typically means the smallest color index that is not being used by any of its adjacent vertices.
- Move to the next vertex: Select the next uncolored vertex in the graph.
- Check adjacent vertices: Examine the colors assigned to the adjacent vertices of the current vertex.
- Assign color: Assign the first available color to the current vertex that is not being used by its adjacent vertices.
- Repeat steps 4-6: Continue this process for all uncolored vertices in the graph until all vertices are colored.



Step 1: Color b with 1.

Step 2: Color c with 1 since b and c are not adjacent.

Step 3: Color d with 2 since d is adjacent to a vertex of color 1

Step 4: Color e with 3 since e is adjacent to a vertex of color 1 (c) and a

vertex of color 2 (d).

Step 5: Color f with 1 since f is not adjacent to any previous vertices.

Step 6: Color h with 2 since h is adjacent to a vertex of color 1 (f).

Step 7: Color j with 2 since j is adjacent to a vertex of color 1 (f).

Step 8: Color m with 3 since m is adjacent to vertices of color 1 (b, c) and a vertex of color 2 (d).

Step 9: Color n with 4 since n is adjacent to vertices of color 1, 2, and 3.

Step 10: Color p with 5 since p is adjacent to vertices of color 1, 2, 3, and 4.

6) Consider a simple graph with vertices A, B, C, and D connected as follows: A-B, A-C, A-D, B-C, and C-D. Explain first-fit vertex coloring algorithm?

First-Fit Coloring Algorithm

Input: Graph G with vertices ordered as $x_1 \prec x_2 \prec \cdots \prec x_n$.

Steps:

1. Assign x_1 color 1.
2. Assign x_2 color 1 if x_1 and x_2 are not adjacent; otherwise, assign x_2 color 2.
3. For all future vertices, assign x_i the least number color available to x_i in $G[x_1, \ldots, x_i]$; that is, give x_i the first color not used by any neighbor of x_i that has already been colored.

Output: Coloring of G.

Step 1: Color A with Yellow.

Step 2: Color B with Blue since A and C are adjacent.

Step 3: Color C with Green since C is adjacent to a vertex of color Red and B of blue color

Step 4: Color D with Black since is adjacent to a vertex of color red, blue and green

7) Describe with an example graph the first-fit coloring algorithm colors its vertices? You can select 5 question answer or sixth question Answer

8) Write a python program for vertex coloring using a simple graph representation and the first-fit coloring algorithm?

class Graph:

```
    def __init__(self, vertices):
        self.vertices = vertices
        self.adjacency_list = {v: [] for v in range(vertices)}

    def add_edge(self, u, v):
        self.adjacency_list[u].append(v)
        self.adjacency_list[v].append(u)

    def first_fit_coloring(self):
        colored = {}
        colors = {}
        for vertex in range(self.vertices):
            adjacent_colors = {colors[colored[v]] for v in self.adjacency_list[vertex] if v in colored}
            for color in range(self.vertices):
                if color not in adjacent_colors:
                    colored[vertex] = color
                    colors[color] = vertex
                    break
        return colored

    def print_colored_vertices(self, colored):
        for vertex, color in colored.items():
            print(f"Vertex {vertex} is colored with color {color}")


# Example usage:
if __name__ == "__main__":
    # Create a graph
    graph = Graph(6)
    graph.add_edge(0, 1)
    graph.add_edge(0, 2)
    graph.add_edge(1, 3)
    graph.add_edge(2, 3)
    graph.add_edge(3, 4)
    graph.add_edge(4, 5)
    # Perform first-fit coloring
    colored_vertices = graph.first_fit_coloring()
    # Print the colored vertices
    graph.print_colored_vertices(colored_vertices
```

9) Explain the edge coloring process with the following graph given below.

Edge coloring is a fundamental concept in graph theory, where the goal is to assign colors to the edges of a graph such that no two adjacent edges share the same color. This process is similar to vertex coloring, where colors are assigned to vertices so that adjacent vertices have different colors.

Here's a basic explanation of the edge coloring process:

- **Start with a Graph**: You begin with a graph, which consists of vertices (nodes) connected by edges (lines). The graph can be directed or undirected.
- **Select Colors:** Determine the number of colors you will use to color the edges. This number is referred to as the "chromatic index" or "edge chromatic number" of the graph. Finding the minimum number of colors needed for edge coloring is often a challenging problem.
- **Color the Edges:** Assign colors to the edges according to the chosen coloring strategy. The goal is to ensure that no two adjacent edges share the same color. There are different algorithms and strategies to accomplish this.
- **Check for Validity:** After coloring, verify that no two adjacent edges have the same color. If there are adjacent edges with the same color, then the edge coloring is invalid, and you need to adjust the colors accordingly.
- **Optimization:** In many cases, the goal is to minimize the number of colors used (i.e., minimize the chromatic index). Finding the minimum edge coloring for a graph is a complex problem and often requires sophisticated algorithms.

There are various approaches to edge coloring, including greedy algorithms, backtracking algorithms, and mathematical formulations. the complexity of edge coloring depends on the characteristics of the graph, such as its structure, connectivity, and degree distribution.

edge coloring has applications in various fields, including scheduling problems, wireless communication, register allocation in compilers, and designing frequency assignment in radio networks.

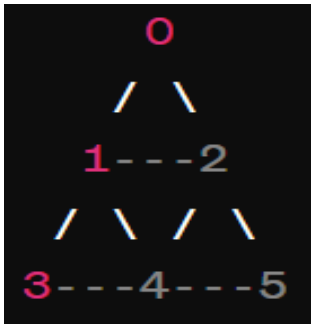
10) Explain the vertex coloring process with the following graph given below.

Vertex coloring is a fundamental concept in graph theory, where the objective is to assign colors to the vertices of a graph in such a way that no two adjacent vertices (connected by an edge) share the same color. This process is crucial in various applications, including scheduling, register allocation in compilers, map coloring, and many others.

Here's a breakdown of the vertex coloring process:

- **Start with a Graph:** You begin with a graph, which consists of vertices (nodes) connected by edges (lines). The graph can be directed or undirected.
- **Select Colors:** Determine the number of colors you will use to color the vertices. This number is referred to as the "chromatic number" of the graph. Finding the minimum number of colors needed for vertex coloring is often a challenging problem.

- **Color the Vertices:** Assign colors to the vertices according to the chosen coloring strategy. The goal is to ensure that no two adjacent vertices (connected by an edge) share the same color. There are different algorithms and strategies to accomplish this.
- **Check for Validity:** After coloring, verify that no adjacent vertices have the same color. If there are adjacent vertices with the same color, then the vertex coloring is invalid, and you need to adjust the colors accordingly.
- **Optimization:** In many cases, the goal is to minimize the number of colors used (i.e., minimize the chromatic number). Finding the minimum vertex coloring for a graph is a complex problem and often requires sophisticated algorithms.



- The complexity of vertex coloring depends on the characteristics of the graph, such as its structure, connectivity, and degree distribution.
- Vertex coloring has applications in various fields, including scheduling problems (where each vertex represents a task and adjacent vertices represent conflicting tasks),
- map coloring (where regions on a map are represented by vertices and adjacent regions should have different colors),
- register allocation in compilers (where vertices represent variables and conflicts between variables are represented by edges), and many others.

References: Data collected from online resources and textbook