Discrete Mathematics Project

PROJECT NO: 7

A Project Submitted

in Partial Fulfilment of the

Requirements for the Degree of

Bachelor of Technology

in

CSE

SUBMITTED BY: GROUP NO: 7

Group Members:

D Veera Harsha Vardhan Reddy

(210C2030061)

V Sai Sumanth (210C2030060)

K Sritej Vishnu (210C2030053)

SCHOOL OF ENGINEERING AND TECHNOLOGY

BML MUNJAL UNIVERSITY GURGAON

December 2023

Acknowledgment

We extend our deepest gratitude to our respected Professor Dr. Rishi Asthana, whose expert guidance were instrumental in the completion of this project.

I would also like to express our sincere appreciation to my group mates, Harsha, and Vishnu for their collaborative spirit, dedication, and significant contributions to the project. Their perspectives and diligence were vital in achieving our project goals.

The project helped us learning more about graph colouring, methods to implement and its applications.

Together, under our professor mentorship, we were able to navigate the complexities of this project and see it to its successful completion.

Contents

Problem Statement

Given Problem:

Discuss the Application of graph coloring in frequency assignment. Use it to determine how many different channels are needed for six stations located at the distances shown in the table, if two stations cannot use the same channel when they are within 150 miles of each other? Wherever required you can use available software or write your own codes in C/C++.

	1	2	3	4	5	6
1	—	85	175	200	50	100
2	85	—	125	175	100	160
3	175	125	—	100	200	250
4	200	175	100	—	210	220
5	50	100	200	210	—	100
6	100	160	250	220	100	—

Problem Description:

The project involves the application of graph coloring principles in frequency assignment for communication networks. Specifically, the objective is to determine the minimum number of different frequency channels required for a network of six stations.

These stations are placed at various distances from each other, as provided in a distance matrix. The constraint is that no two stations can use the same frequency channel if they are located within 150 miles of each other. This problem requires an understanding of discrete mathematics and graph theory, and it may be solved using available software tools or custom-coded solutions in programming languages such as C/C++. The solution should effectively demonstrate how graph coloring can be used to address real-world problems in frequency allocation for communication networks.

INTRODUCTION

The introduction to the mathematical theory and formulas for the given problem would involve graph theory and, more specifically, the concept of graph colouring.

Overview:

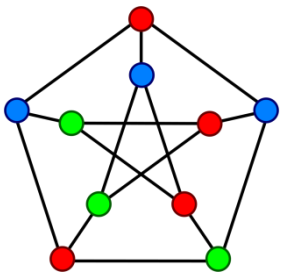
The problem at hand is a practical application of a concept in graph theory known as "graph colouring." The main goal is to assign the minimum number of colours (or frequencies, in the context of the problem) to the vertices of a graph such that no two adjacent vertices share the same colour. This is particularly important in frequency assignment for communication networks to avoid interference.

Graph Theory:

Graph theory is a field of combinatorial mathematics that studies graphs, which are mathematical structures used to model pairwise relations between objects. A graph is made up of vertices (or nodes) and edges (or links) that connect pairs of vertices. In the context of frequency assignment, each station is represented as a vertex, and an edge between two vertices indicates that those stations are within a certain distance that would require separate frequencies.

Graph Colouring:

Graph colouring is a fundamental concept in graph theory that involves assigning colours to the vertices of a graph subject to certain constraints. The primary objective is to colour the vertices in such a way that no two adjacent vertices (connected by an edge) have the same colour.



Chromatic Number: The chromatic number of a graph is the minimum number of colours needed to properly colour its vertices. Determining the chromatic number is an essential problem in graph theory.

There are several types and concepts related to graph colouring:

1. Vertex Colouring: This is the most common form of graph colouring. It involves assigning colours to each vertex of the graph such that no two adjacent vertices share the same colour.

2. **Edge Colouring:** In edge colouring, colours are assigned to the edges of the graph such that no two adjacent edges share the same colour. The objective is to use the fewest possible colours.

3. Planar Graph Colouring: Also known as Face and map colouring; Planar graphs are those that can be drawn on a plane without any edges crossing. The famous Four Colour Theorem states that any planar graph can be coloured using at most four colours, ensuring that no two adjacent regions have the same colour.

4. List Colouring: In list colouring, each vertex is assigned a list of permissible colours. The challenge is to colour the vertices using only colours from their respective lists while adhering to the no-adjacent-vertices-same-colour rule.

5. Path Colouring: It is a specific type of graph colouring where the focus is on colouring the vertices of a path graph, ensuring that no two adjacent vertices share the same colour. A path graph is a simple graph that represents a path or a line and consists of vertices connected in a linear sequence.

6. Total Colouring: Total colouring involves colouring both the vertices and edges of a graph. The constraint is that for any edge, its two endpoints and the edge itself must all have different colours.

These are some of the key types of graph colouring. Depending on the specific problem or application, one type of colouring may be more relevant or practical than another.

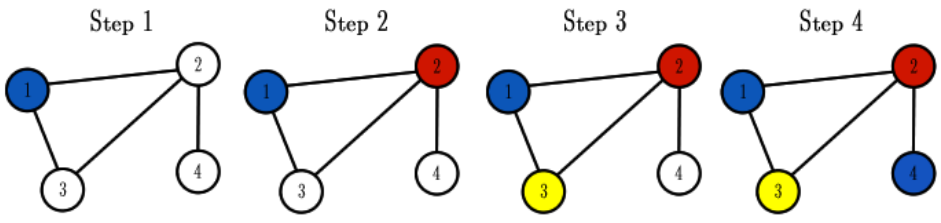
Mathematical Formulation:

- Let $G = (V, E)$ be a graph where V is the set of vertices and E is the set of edges.
- The distance constraint d is given as 150 miles.
- An adjacency matrix A can be formed from the given distances, where $a_{ij} = 1$ if the distance between stations i and j is less than or equal to d, and $a_{ij} = 0$ otherwise.
- A coloring function $c: V \rightarrow \{1, 2, ..., k\}$ assigns a color to each vertex.
- The objective is to minimize *the* number *of* colors k, such that for every edge $(i, j) \in E$, $c(i) \neq c(j)$.

Algorithms:

Several algorithms can be used for graph colouring, with varying levels of complexity and efficiency. A common approach is the "greedy colouring" algorithm, which iteratively assigns the lowest possible colour to each vertex. The algorithm's steps can be summarized as follows:

1. Initialize all vertices with no colour.
2. For each vertex, choose the lowest numbered colour that has not been used by its neighbours.
3. Repeat until all vertices are coloured.



The greedy algorithm does not always produce the chromatic number but gives an upper bound. Other algorithms, such as backtracking and heuristic methods, can be used to find the chromatic number but may require more computational resources.

In the case of the given problem, the application of these concepts and algorithms would provide the solution to the frequency assignment challenge, ensuring that the minimum

number of channels is used without interference between the stations.

To provide a mathematical solution to the problem of assigning frequencies to stations such that no two stations within 150 miles use the same frequency, we follow these steps:

1.Construct the Graph:

- Create a graph $G = (V, E)$, where each vertex $(v_i \in V)$ corresponds to a station.
- An edge $(e_{ij} \in E)$ exists between vertices v_i and v_j if the distance between station *(i)* and station *(j)* is 150 miles or less.

2. Adjacency Matrix:

- Formulate an adjacency matrix A based on the distances, where $A[i][j] = 1$ if the distance between stations *(i)* and *(j)* is less than or equal to 150 miles, otherwise $A[i][j] = 0$

3. Greedy Coloring Algorithm:

Initialize a color array *C* where $C[i] = -1$ indicating that no color has been assigned to vertex *(i)*

For each vertex (v_i), do the following:

- Create a temporary array of available colors, initialized to true.
- For each vertex (v_j) adjacent to (v_i) $(i.e., A[i][j] = 1)$, if a color has been assigned to (v_j) $(i.e.[j] \neq -1)$, set the corresponding color as unavailable (*i.e., available[C[j]] = false).*
- Find the first color in the available array that is still true and assign it to vertex (v_i) *(i.e., C[i] = the first true index in available).*

4. Determine the Number of Channels:

After assigning colours to all vertices, the number of different colours used gives the minimum number of channels needed, which is equivalent to the maximum value in the colour array *C plus 1 (since the colouring starts from 0).*

Mathematically, if *k* is the number of different colours used, then *k* is the minimum number of channels required to ensure that no two stations within *150 miles* of each other use the same channel.

Analytical Solution

Here is a step-by-step mathematical solution using the distances provided:

- We construct an adjacency matrix A based on the distances given in the uploaded image.
- For example, since the distance between stations *1* and *2* is *85 miles*, and *85* is less than *150*, we set *A [1][2] = 1 (and symmetrically A [2][1] = 1 since the graph is undirected).* We repeat this process for all station pairs.
- Using the adjacency matrix, we apply the greedy coloring algorithm. We start at station *1* and assign the first color *(let us say color 0)*. We then move to station *2*, check the colors of all stations it is connected to, and assign the lowest color that has not been used by its neighbors.
- Continue this process until all stations have been assigned a color.
- The highest number assigned in the coloring process is the number of channels needed.

To summarize, the mathematical solution involves translating the problem into a graph colouring problem, constructing the graph, and applying a colouring algorithm to find the minimum number of colours needed. The chromatic number of the graph will be the minimum number of channels needed for the stations.

First, let us construct the adjacency matrix based on whether stations are within 150 miles of each other:

- Stations 1 and 2 are 85 miles apart, so they are adjacent.
- Stations 1 and 5 are 50 miles apart, so they are adjacent.
- Stations 1 and 6 are 100 miles apart, so they are adjacent.
- Stations 2 and 3 are 125 miles apart, so they are adjacent.
- Stations 2 and 5 are 100 miles apart, so they are adjacent.
- Stations 3 and 4 are 100 miles apart, so they are adjacent.
- Stations 5 and 6 are 100 miles apart, so they are adjacent.

→ **Station 1:** Adjacent to Stations 2, 5, 6

→ **Station 2:** Adjacent to Stations 1, 3, 5

→ **Station 3:** Adjacent to Stations 2, 4

→ **Station 4:** Adjacent to Station 3

→ **Station 5:** Adjacent to Stations 1, 2, 6

→ **Station 6:** Adjacent to Stations 1, 5

Now, let us start colouring:

- Assign Channel 1 to Station 1.
- Station 2 is adjacent to Station 1, so it cannot have Channel 1. Assign Channel 2 to Station 2.
- Station 3 is adjacent to Station 2 but not Station 1, so it can have Channel 1.
- Station 4 is adjacent to Station 3 (which has Channel 1), so it must have a different channel. Assign Channel 2 to Station 4.
- Station 5 is adjacent to Stations 1 (Channel 1) and 2 (Channel 2). It cannot have either of those channels, so we assign Channel 3 to Station 5.
- Station 6 is adjacent to Station 1 (Channel 1) and Station 5 (Channel 3). It can have Channel 2, which is different from its adjacent stations.

In the end, we have used three channels to ensure that no two adjacent stations share the same channel:

→ **Channel 1:** Stations 1, 3

→ **Channel 2:** Stations 2, 4, 6

→ **Channel 3:** Station 5

Therefore, the solution to the problem, calculated manually, is that a minimum of 3 different channels are required for the six stations.

Results of Simulations

Online GDB: https://onlinegdb.com/B4CVcF95V

```
#include <iostream>
#include <vector>

const int N = 6;  // Number of stations

// Function to print the station colors (channels)
void printStationsChannels(int colors[]) {
    std::vector<int> channels[N];  // Vector to store stations with the same
channel

    // Group stations by color
    for (int i = 0; i < N; i++) {
        channels[colors[i]].push_back(i + 1);  // +1 to match station
numbering starting at 1
    }

    // Print the stations for each channel
    for (int i = 0; i < N; i++) {
        if (!channels[i].empty()) {
            std::cout << "Channel " << i + 1 << ": Stations ";
            for (int j = 0; j < channels[i].size(); j++) {
                std::cout << channels[i][j];
                if (j < channels[i].size() - 1) std::cout << ", ";
            }
            std::cout << std::endl;
        }
    }
}

// Function to find the minimum color for a node
int getMinColor(bool graph[N][N], int colors[], int node) {
    bool available[N] = {false};  // Colors available for the node

    // Check colors of adjacent nodes and mark them as unavailable
    for (int i = 0; i < N; ++i) {
        if (graph[node][i] && colors[i] != -1) {
            available[colors[i]] = true;
        }
    }

    // Find the first color that is not assigned to adjacent nodes
    int color;
```

```
    for (color = 0; color < N; color++) {
        if (!available[color]) break;
    }

    return color;  // Return the minimum available color
}

// Function to apply graph coloring
int graphColoring(bool graph[N][N], int colors[]) {
    for (int i = 0; i < N; i++) colors[i] = -1;  // Initialize colors for all
nodes to -1

    // Assign colors to nodes
    for (int i = 0; i < N; i++) {
        colors[i] = getMinColor(graph, colors, i);
    }

    // Find the maximum color ID used which is the number of channels
    int maxColor = 0;
    for (int i = 0; i < N; i++) {
        if (colors[i] > maxColor) maxColor = colors[i];
    }

    return maxColor + 1;  // +1 because colors start from 0
}

int main() {
    // Distance matrix as provided
    int distances[N][N] = {
        {0, 85, 175, 200, 50, 100},
        {85, 0, 125, 175, 100, 160},
        {175, 125, 0, 100, 200, 250},
        {200, 175, 100, 0, 210, 220},
        {50, 100, 200, 210, 0, 100},
        {100, 160, 250, 220, 100, 0}
    };

    // Create graph from distance matrix, true if distance is within 150 miles
    bool graph[N][N];
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            graph[i][j] = (i != j && distances[i][j] <= 150);
        }
    }

    int colors[N];  // Array to store the color assigned to each station
    int numChannels = graphColoring(graph, colors);
```

```
    std::cout << "Minimum number of channels needed: " << numChannels <<
std::endl;

    // Print the stations that share the same channel
    printStationsChannels(colors);

    return 0;
}
```

```
Minimum number of channels needed: 3
Channel 1: Stations 1, 3
Channel 2: Stations 2, 4, 6
Channel 3: Stations 5
```

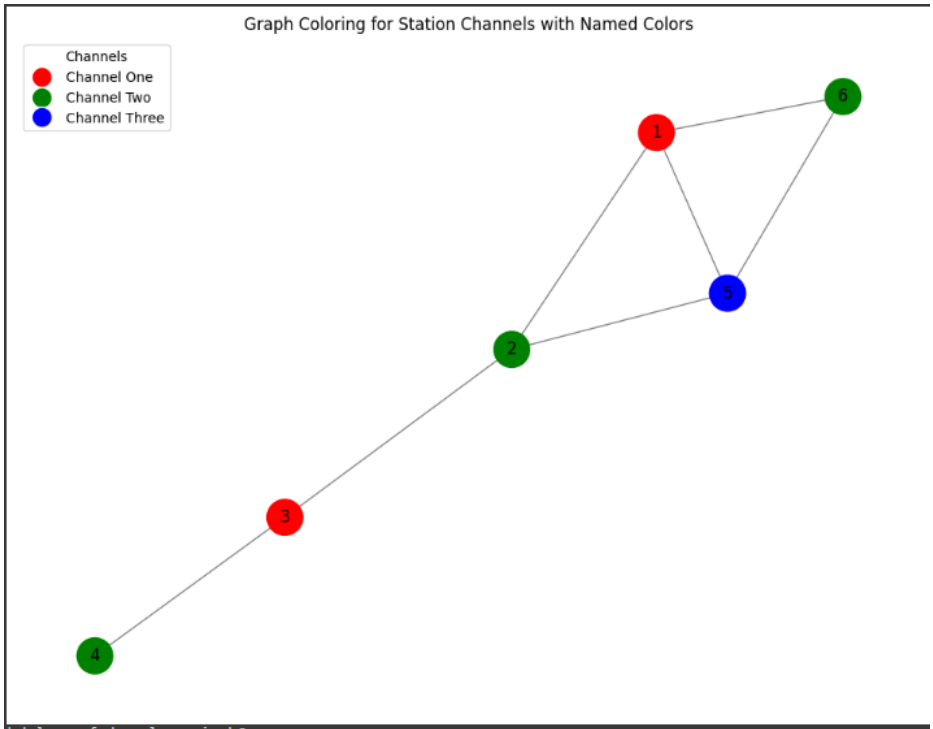
Graph Representation:

```
import matplotlib.pyplot as plt
import networkx as nx

# Distances between the stations as given by the user's table
distances = {
    (1, 2): 85,
    (1, 3): 175,
    (1, 4): 200,
    (1, 5): 50,
    (1, 6): 100,
    (2, 3): 125,
    (2, 4): 175,
    (2, 5): 100,
    (2, 6): 160,
    (3, 4): 100,
    (3, 5): 200,
    (3, 6): 250,
    (4, 5): 210,
    (4, 6): 220,
    (5, 6): 100,
}
# Create a graph
G = nx.Graph()

# Add all nodes
G.add_nodes_from(range(1, 7))

# Add edges with distances less than or equal to 150 miles
```

```
for (i, j), distance in distances.items():
    if distance <= 150:
        G.add_edge(i, j)

# Apply graph coloring
coloring = nx.coloring.greedy_color(G, strategy='largest_first')

# Define color names for channels
color_names = {0: 'Channel One', 1: 'Channel Two', 2: 'Channel Three'}

# Map the color names to the coloring
named_coloring = {node: color_names[color] for node, color in
coloring.items()}

# Create a color map for the plot, with specific colors for each channel
color_map_plot = {'Channel One': 'red', 'Channel Two': 'green', 'Channel
Three': 'blue'}
colors_for_plot = [color_map_plot[named_coloring[node]] for node in G.nodes()]

# Draw the graph
plt.figure(figsize=(12, 9))
pos = nx.spring_layout(G)  # positions for all nodes

# Draw the nodes with assigned colors
nx.draw_networkx_nodes(G, pos, node_size=700, node_color=colors_for_plot,
label=named_coloring)

# Draw the edges
nx.draw_networkx_edges(G, pos, width=1.0, alpha=0.5)

# Draw the labels
nx.draw_networkx_labels(G, pos, font_size=12)

# Display the legend for channel colors
plt.legend(handles=[plt.Line2D([0], [0], marker='o', color='w', label=key,
                           markersize=15, markerfacecolor=val) for key, val in
color_map_plot.items()],
           title="Channels")

# Set plot title and turn off the axis
plt.title("Graph Coloring for Station Channels with Named Colors")
plt.axis("off")  # Turn off the axis

# Display the graph
plt.show()

# Count the number of channels needed
num_channels_needed = len(set(coloring.values()))
```

```
print(f"total no. of channels required {num_channels_needed} ")

# Print the channel assignment for each station
for node, channel_name in named_coloring.items():
    print(f"Station {node} is assigned to {channel_name}")
```

Result:




```
total no. of channels required 3
Station 1 is assigned to Channel One
Station 2 is assigned to Channel Two
Station 5 is assigned to Channel Three
Station 3 is assigned to Channel One
Station 6 is assigned to Channel Two
Station 4 is assigned to Channel Two
```

Conclusion

The study successfully demonstrated a practical application of discrete mathematics by solving the frequency assignment problem for a network of six stations using a greedy graph colouring technique. The findings we obtained indicate the algorithm's capability to efficiently allocate communication channels, ensuring that stations within 150 miles of each other operate without disruption. Even while the greedy technique does not always result in the best answer, in this case it worked well and was sufficient, demonstrating the usefulness of graph colouring in addressing discrete constraint network design challenges in the real world. This project highlights how discrete mathematics is essential to creating algorithms that are both practical and mathematically interesting.

References

1. Rosen, K. H. (2012). Discrete Mathematics and Its Applications (7th ed.). McGraw-Hill Education.
2. Sotak, Roman, and Miroslav Skovira. "Vertex list coloring in products of graphs." Discussions Mathematics Graph Theory 31.3 (2011): 447-457. Retrieved from https://intapi.sciendo.com/pdf/10.2478/v10209-011-0012-y
3. GeeksforGeeks. NetworkX: Python software package for the study of complex networks. Retrieved from https://www.geeksforgeeks.org/networkx-python-software-package-study-complex-networks/
4. GeeksforGeeks. Graph Coloring | Set 2 (Greedy Algorithm). Retrieved from https://www.geeksforgeeks.org/graph-coloring-set-2-greedy-algorithm/