	Pimpri Chinchwad Education Trust's **Pimpri Chinchwad College of Engineering (PCCoE)** (An Autonomous Institute) Affiliated to Savitribai Phule Pune University (SPPU) ISO 21001:2018 Certified by TUV	
Department: Information Technology	**Academic Year**: 2025-26	**Semester:** V
DESIGN AND ANALYSIS OF ALGORITHMS LABORATORY		
Name : Avishkar Jadhav	**PRN** : 123B1F032	

ASSIGNMENT NO. 7

Aim:
Create an exam timetable by modeling courses as a conflict graph (edge = shared student), color the graph so no two adjacent courses share a time slot, compare three algorithms (Greedy, Welsh–Powell, DSATUR), and then allocate rooms (by capacity) into each timeslot.

Problem Statement:
Scenario: University Timetable Scheduling
A university is facing challenges in scheduling exam timetables due to overlapping student enrollments in multiple courses. To prevent clashes, the university needs to assign exam slots efficiently, ensuring that no two exams taken by the same student are scheduled at the same time.
To solve this, the university decides to model the problem as a Graph Coloring Problem, where:
● Each course is represented as a vertex.
● An edge exists between two vertices if a student is enrolled in both courses.
● Each vertex (course) must be assigned a color (time slot) such that no two adjacent vertices share the same color (no two exams with common students are scheduled in the same slot).
As a scheduling system developer, you must:
1. Model the problem as a graph and implement a graph coloring algorithm (e.g., Greedy Coloring or Backtracking).
2. Minimize the number of colors (exam slots) needed while ensuring conflict-free scheduling.
3. Handle large datasets with thousands of courses and students, optimizing performance.
4. Compare the efficiency of Greedy Coloring, DSATUR, and Welsh-Powell algorithms for better scheduling.
Extend the solution to include room allocation constraints where exams in the same slot should fit within available classrooms..

Code:

```
#include <bits/stdc++.h>
using namespace std;

// Add edge between two courses
void addEdge(vector<vector<int>> &graph, int u, int v) {
    graph[u].push_back(v);
    graph[v].push_back(u);
}

// Greedy Coloring algorithm
void greedyColoring(vector<vector<int>> &graph, int numCourses) {
    vector<int> result(numCourses, -1);
    result[0] = 0; // assign first slot to first course

    vector<bool> available(numCourses, true);

    for (int u = 1; u < numCourses; u++) {
        fill(available.begin(), available.end(), true);

        // mark unavailable colors
        for (int adj : graph[u]) {
            if (result[adj] != -1)
                available[result[adj]] = false;
        }

        // find first available color
        int color;
        for (color = 0; color < numCourses; color++) {
            if (available[color])
                break;
        }
        result[u] = color;
    }

    cout << "Exam Slot Assignment (Greedy Coloring):\n";
    for (int u = 0; u < numCourses; u++) {
```

```
        cout << "Course " << u << " → Slot " << result[u] << endl;
    }

    int maxColor = *max_element(result.begin(), result.end());
    cout << "\nTotal Exam Slots Used: " << (maxColor + 1) << endl;
}

int main() {
    int numCourses = 6;
    vector<vector<int>> graph(numCourses);

    // Add edges (conflicting courses with common students)
    addEdge(graph, 0, 1);
    addEdge(graph, 1, 2);
    addEdge(graph, 2, 3);
    addEdge(graph, 3, 4);
    addEdge(graph, 3, 5);
    addEdge(graph, 0, 5);

    greedyColoring(graph, numCourses);
    return 0;
}
```

Output:

Exam Slot Assignment (Greedy Coloring):
Course 0 → Slot 0
Course 1 → Slot 1
Course 2 → Slot 0
Course 3 → Slot 1
Course 4 → Slot 0
Course 5 → Slot 2

Total Exam Slots Used: 3

Conclusion:

In this assignment, the problem of university exam timetable scheduling was effectively modeled using the concept of **Graph Coloring**. Each course was represented as a vertex, and an edge was drawn between two vertices if they shared common students. The **Greedy Coloring Algorithm** was implemented to assign exam slots in such a way that no two conflicting courses were scheduled at the same time. The program successfully minimized the total number of exam slots while ensuring a **conflict-free schedule**. This approach demonstrates the practical application of graph theory in solving

real-life scheduling and resource allocation problems efficiently.