الغرض من هذا الكود هو **جدولة اختبارات الكورسات بطريقة تضمن عدم وجود تعارض في المواعيد**، بحيث يتم تخصيص يوم مختلف للاختبارات لكل مجموعة من الكورسات التي تشترك في نفس الطلاب. يُستخدم هذا النوع من الجدولة لضمان أن الطالب الذي يأخذ أكثر من كورس لن يُطلب منه أداء اختبارين في نفس اليوم.

خطوات عمل الكود:

1. **إنشاء رسم بياني (Graph):**
والعلاقات بينها (المجموعات المشتركة (Nodes) يتم تمثيل الكورسات كعُقد (Edges) للطلاب) كأضلاع.

- إذا كان هناك طالبان يشتركان في أكثر من كورس، يتم إنشاء ضلع بين الكورسين.

2. **إضافة الأضلاع (AddEdge):**
يتم إدخال البيانات التي تمثل الكورسات المشتركة من قبل المستخدم، وتُضاف بين الكورسات في الرسم البياني (Edges) العلاقات.

3. **تخصيص جدول الاختبارات (GenerateSchedule):**

- يتم تخصيص يوم (أو لون) لكل كورس بناءً على القاعدة التالية:
إذا كان هناك كورسات متصلة بكورس معين (أي يشتركون في نفس الطلاب)، يتم تجنب استخدام ألوان تلك الكورسات، واختيار أول لون (يوم) متاح.

- يُمثل كل لون يومًا معينًا.

4. **إخراج الجدول:**
بعد تخصيص الأيام لكل كورس، يتم عرض اليوم المخصص لكل اختبار للمستخدم.

مثال عملي:

- إذا أدخل المستخدم:

- عدد الكورسات: ٤

- عدد المجموعات المشتركة: ٢

- العلاقات:
  - كورس ٠ وكورس ١
  - كورس ١ وكورس ٢
- فإن الكود سيُنتج جدولًا مثل:
  - كورس ٠: اليوم ١
  - كورس ١: اليوم ٢
  - كورس ٢: اليوم ١
  - كورس ٣: اليوم ١ (لأنه لا يتشارك طلابًا مع أي كورس آخر).

الفائدة العملية:

هذا النوع من الكود يُستخدم في الجامعات والمدارس لتنظيم مواعيد الاختبارات وتجنب التعارض، خاصةً في الأنظمة التي تعتمد على وجود مجموعات مشتركة من الطلاب في كورسات متعددة

ALGORITHM:

Conflict-Free Exam Scheduling Algorithm

Input:

- N: Number of courses.
- E: List of edges representing conflicts (shared students between courses).

Output:

- Schedule[]: An array where Schedule[i] represents the exam day assigned to course i.

Step 1: Initialize the Graph

1. Create an adjacency list Graph[] of size N to represent the relationships between courses.
2. For each edge (u, v) in E, add v to Graph[u] and u to Graph[v].

Step 2: Initialize Scheduling Data

1. Create an array Schedule[] of size N and initialize all values to -1.
    - Schedule[i] = -1 means course i has not been assigned a day yet.
2. Create a boolean array AvailableColors[] of size N initialized to false.
    - AvailableColors[c] = true indicates that day c is already used by a neighboring course.

Step 3: Assign Days to Courses

1. Assign day 0 to the first course: Schedule[0] = 0.
2. For each course u from 1 to N-1:
   a. For each neighbor v of u (i.e., v in Graph[u]):
    - If Schedule[v] != -1, mark AvailableColors[Schedule[v]] = true.
      b. Find the first available day d (smallest d where AvailableColors[d] = false).
    - Assign day d to course u: Schedule[u] = d.
      c. Reset AvailableColors[] to false for the next iteration.

Step 4: Output the Schedule

1. For each course i, output:
    - Course i will have the exam on day Schedule[i] + 1.

Example

Input:

- N = 4 (Courses)
- E = {(0, 1), (1, 2), (2, 3)}

Output:

- Course 0: Day 1
- Course 1: Day 2
- Course 2: Day 1
- Course 3: Day 2

Time Complexity Analysis

1. **Graph Construction:** O(E), where E is the number of edges.
2. **Day Assignment:** O(N × D), where D is the maximum degree of the graph.
3. **Overall Complexity:** O(N × D + E).

This is a **graph coloring algorithm** tailored for scheduling exams while avoiding conflicts.