SSN COLLEGE OF ENGINEERING
KALAVAKKAM-603110

Department of Computer Science and Engineering

Timetable Management System for an Academic Institution

A Sadhana - Reg No: 3122 21 5001 089
Pooja Premnath - Reg No: 3122 21 5001 066
Prahalad Rajagopal - Reg No: 3122 21 5001 067

Project Guide: Dr Shahul Hamead

Abstract

The intricate task of scheduling a timetable is a problem faced by every educational institution. Although this task seems to be very common, the rules that define its generation make it incredibly tedious to do it without the help of a well-defined algorithm. Computerizing the process of generating such a timetable saves a lot of time, provides clarity and avoids repetition as well as ambiguity. Numerous methodologies ranging from brute force techniques to graph theory algorithms can be used as a solution to this problem, generating a multitude of solutions. This project focuses on developing a solution using the concept of edge colouring, to eliminate conflicts in allocation. Since the primary hard constraint of conflict is handled early on, the number of comparisons to generate a viable timetable drastically reduces. The project also handles numerous constraints placed on lab and theory courses.

Problem Statement

The given problem of generating a timetable for an academic institution can be classified as a scheduling problem. A solution to the problem also depends on the constraints that have been imposed on the system, in terms of time constraints and availability of faculty. The solution involves parallelly handling classes across semesters, to find the right combination of them at different times on any given day. The system aims to generate timetables for both students and teachers.

Literature Survey

1. Mary Almond in “An Algorithm for Constructing University Timetables”, suggests a simple heuristic approach to develop a timetable for different departments. The resources are represented in the form of a 3-dimensional array. When an allocation is made for a class, there is a decrementation made in a Requirements matrix and a Lecturer Availability matrix. The solution generated may not be unique, and different variations can be obtained by scanning the Requirements matrix in different directions.
2. Samson Oluwasuen Fadiya et al. emphasize on the effective usage of relational databases to store and retrieve data in an organized manner for the development of a scheduling system in “University Time-table Scheduling System: Databases Design”. It elaborates on the required tables, and the interconnectivity that has to be established between them, such as the relationships between the faculty table and the course table.
3. Wathsala Samarasekara in “An Application of Graph Coloring Model to Course Timetabling Problem” explains a simple approach to this scheduling problem using the idea of graph colouring, especially edge colouring, to eliminate the occurrences of clashes among professors taking multiple classes across semesters.
4. Rakesh Prasad Badoni et al in “A Graph Edge Colouring Approach for School Timetabling Problems”, discuss various ways of implementing the timetabling system using graph colouring while taking into consideration various constraints and creating a constraint based system.
5. Timothy Anton Redl in “A Study of University Timetabling that Blends Graph Colouring with the Satisfaction of Various Essential and Preferential Conditions” gives a detailed explanation of two graph colouring approaches to this constraint based timetabling problem - vertex colouring as well as edge colouring to plot a conflict graph. Alternative approaches such as Mathematical Programming have also been discussed.

Methodology

Project Design

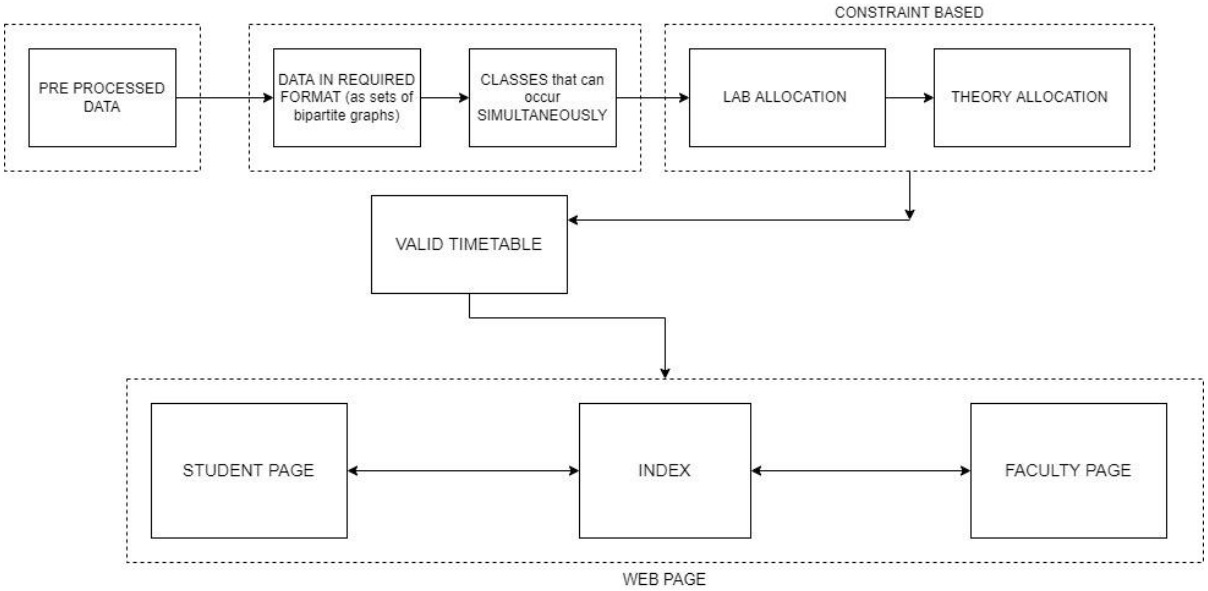


Figure 1: Block diagram of project design

Constraints

In order to generate a timetable that is cohesive, several constraints are to be put in place. Some constraints are absolutely non-negotiable and must be tackled immediately, while other constraints offer more flexibility. The non-negotiable constraints are Hard Constraints. The most important hard constraint is that no teacher should be allotted multiple classes simultaneously. Another important hard constraint is that lab courses have to be scheduled in bulk, with three hours for a lab being allotted consecutively. Only a single lab can be scheduled for a given semester per day. Soft constraints include not having more than two hours of a single subject per day, as well as allotting suitable timings for lunch, such that it does not interfere with the courses scheduled for the day. The remaining hours are filled up with supplementary periods like Library, Sports, PCD and Mentor. Another soft constraint is not having more than two free hours per day.

Table 1: Table of Constraints and the method of implementation

Constraint	**Nature of Constraint**	**Method of Implementation of Constraint**
Conflict Resolution- Multiple classes cannot be allotted to a single teacher simultaneously	Hard Constraint	Using Edge Colouring- Courses with the same edge colour can be allotted parallelly.
Allotting one lab course per day	Hard Constraint	Checking if a day has any course scheduled before choosing a day. (Lab courses are first allocated, and then theory courses).
Allocation of lab hours in bulk	Hard Constraint	Grouping all the lab hours for a given course, and handling them as a single entity. Checking is done to ensure that the three hours are scheduled within working hours, by using predefined lab spots, and then choosing a lab spot at random.
Maximum number of classes for a given subject in a single day is 2.	Hard Constraint	Iterative checking with a counter across each day. Priority is given to those days where the course has not been allotted at all.
Allocation of all the subjects meant for semesters	Hard Constraint	Under the given constraints, due to the random allocation of lab courses, it sometimes becomes impossible to schedule all the courses parallelly without conflict. Under such a situation, the lab allocation is declared to

		be invalid, and labs are allocated all over again, and then theory courses until a suitable timetable is arrived at. The process of generating a timetable may take anywhere between 1- 30 tries.
The maximum number of free hours in a day is 2.	Soft Constraint	Checking is done to count the number of courses scheduled per day. Priority is given to those days where the number of courses scheduled is less than four.
Allocation of Lunch break	Soft Constraint	Lunch break timings may vary depending on lab allocation. After generating the entire timetable, it is checked if the 11:30 am-12:30 pm slot is free or if the 12:30 pm- 1:30 pm slot is vacant. Lunch is scheduled in either of those two slots.
Allocation of supplementary hours like Sports, Mentor, Library and PCD.	Soft Constraint	After allocation of the entire timetable, a suitable number of hours are allotted for supplementary periods.

Module 1: Data Preprocessing

The data for the project has been given in the form of a Notepad file. This data must be processed to remove null values and to make it presentable in a tabularized form. It is also necessary to analyze other tables that might be required for this project that may be inferred from the primary table, and they must be formed. These tables must be presented in a CSV file(MS Excel in our project), since this data is to be used in the upcoming modules and thus needs to be easily transportable, exportable and executable.

Module 2: Edge Colouring

In order to schedule a timetable with no conflicts it is necessary to determine the classes which can occur in parallel. Edge colouring is the process of assigning colours to the edges of a graph such that no two edges sharing a common vertex have the same colour. The edge colouring idea is being used to find the courses which can occur simultaneously without a clash.

A bipartite graph is a graph whose vertices can be divided into two disjoint and independent sets such that every edge connects a vertex in one set to one in the other set. The relationship between lecturers and the

courses they take up to teach is in the form of a bipartite graph as there will be no edges between the lecturers or the courses themselves.

The edge colouring idea is being used to find the courses which can occur simultaneously without a clash. This will later enable the implementation of hard constraints, namely, one teacher can only be in one class at a time and a section of students can only attend one class at a given time.

The relationship between lecturers and the courses they take up to teach is in the form of a bipartite graph as there will be no edges between the lecturers or the courses themselves.

Consider sample data with {L1,L2,L3} as Lecturers and {S1,S2,S3,S4,S5} as the courses. Given below is one of the edge coloured solutions of the dataset considered.

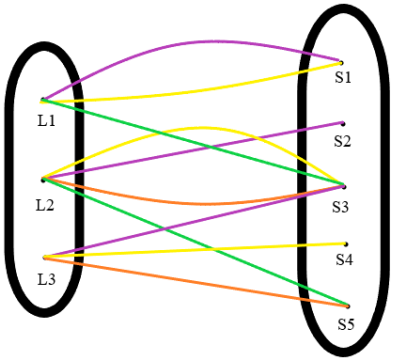


Figure 2: Edge Colouring of a sample dataset

The coloured edges can be labelled as 1,2,3… as per convention. According to Kőnig's Theorem, the minimum number of colours, known as the edge chromatic number or index, required to label the edges is equal to the maximum degree of the graph. Here, L2 has the maximum degree, i.e, L2 takes up the maximum number of classes, 4.

It is easier to arrive at a solution by starting with the largest degree first to obtain the chromatic number so that all possible colour labels can be produced. This can then be continued for the rest of the vertices. All the edges labelled with the same colour represent classes that occur simultaneously.

Module 3: Intermediary Processing of Data

The edge colouring process returns a nested list, where each element is a list containing the faculty name, Course ID, the semester, the section and the colour ID, the most vital outcome of the edge colouring process. This data must be transferred to a CSV file(MS Excel document in our project). In order to perform the subsequent steps of allotting lab courses and theory courses, the processed data must be

sorted, first by color ID to ensure that there are no clashes in the occurrence of classes, and then by semester.

Module 4: Allotment of Lab Courses

The first step in allotting courses is done by handling lab courses. Each lab course has a class once a week, with the class being three consecutive hours. Lab courses are allotted in the beginning, since they consume a majority of the hours on any given day. These lab courses are also subject to the constraint that only one can be scheduled per day. The allocation of three consecutive hours becomes difficult after theory courses are also scheduled. These courses can also only be scheduled in very specific combinations of consecutive hours, since any other combination would interfere with a lunch break, or would be scheduled outside working hours. The module that schedules the lab courses uses the random module to choose a day of the week, and also one of the possible pre-defined time slots. The module also checks the colour-ids of the courses scheduled across courses to ensure that no conflict takes place. The process of allocation is an iterative one, only terminating when a successful combination of classes across semesters is generated.

Module 5: Allotment of Theory Courses

After lab courses have been successfully allotted, theory classes must be scheduled, based on certain constraints imposed. Some of the constraints imposed include not having more than two hours of any subject in a single day. The allotment of theory courses also means that all such courses across semesters must be allotted and that none can be omitted from consideration. A soft constraint that has been imposed is that, on a given day, a class may have a maximum of two free hours with classes like PCD, Mentor, Library or Sports. Given the constraints imposed, especially ensuring that no conflict occurs in scheduling, it sometimes becomes impossible to generate a timetable with all the constraints in place. In such a scenario, all allotments are declared to be incorrect, including the lab courses allotted, and the allotment of both lab and theory courses is done from scratch until a successful outcome is achieved. Once a timetable has been generated, the vacant hours are filled with supplementary courses.

Module 6: Generation of Faculty Timetable

After a timetable has been successfully generated, timetables are also generated for faculty. This process involves the traversal of timetables for all the semesters, across days, and storing the course name, semester and time, in a dictionary structure. Further iteration through the dictionary is done, to position each course in the correct position in a nested list for each teacher.

Module 7: Modularity and Reuse of Code for different sections

Modularity of code helps simplify the problem into smaller blocks, which can be tackled independently, and then linked to form a complete solution. In this scheduling problem, certain tasks can be made modular, while others cannot. For example, the process of timetable allocation for any given section across semesters, cannot be modularized any further, since the allocation runs parallelly across semesters. However, the process of allocation of timetables for different sections can be simplified, since this process is equivalent to multiple program runs. Thus, rather than rewriting the entire program for different sections, the whole process of allocation of courses can be made into a user-defined library function, which can be called with the required parameters, as necessary. This principle of modular programming enhances efficiency, by reducing redundancy. It also simplifies the entire problem, and makes it much easier to make improvements and debug.

In Python, modularity is established by writing multiple user-defined functions. These functions are saved in a .py file, where the name of the .py file becomes the name of the module. Each user-defined function within the module, now becomes a library function. The .py files are stored in a directory, along with an __init__.py file, which helps the system recognize that the directory is now a Python package directory. Import statements can be used to call the module, and <modulename.functionname> can be used to call and utilize each library function.

Module 8: Construction of the Webpage

To make the timetable web-based, the backbones of a website are constructed using HTML. HTML is used to create the basic structure of the website, including links and buttons. User input is collected using forms, with dropdown menus. HTML tables display the output in a consolidated format. The values within the table are dynamic, as the courses and their positioning vary from semester to semester. This is controlled with the help of looping constructs that procure the value from the Python output.

Module 9: Design of the webpage

The webpage is made more user-friendly with the usage of CSS. CSS facilitates the styling of HTML elements, and helps position elements correctly, for greater readability. Navigation throughout the webpage is established using a nav bar. The styling of both the faculty and the student timetable is also done by applying CSS styles on the table built with HTML.

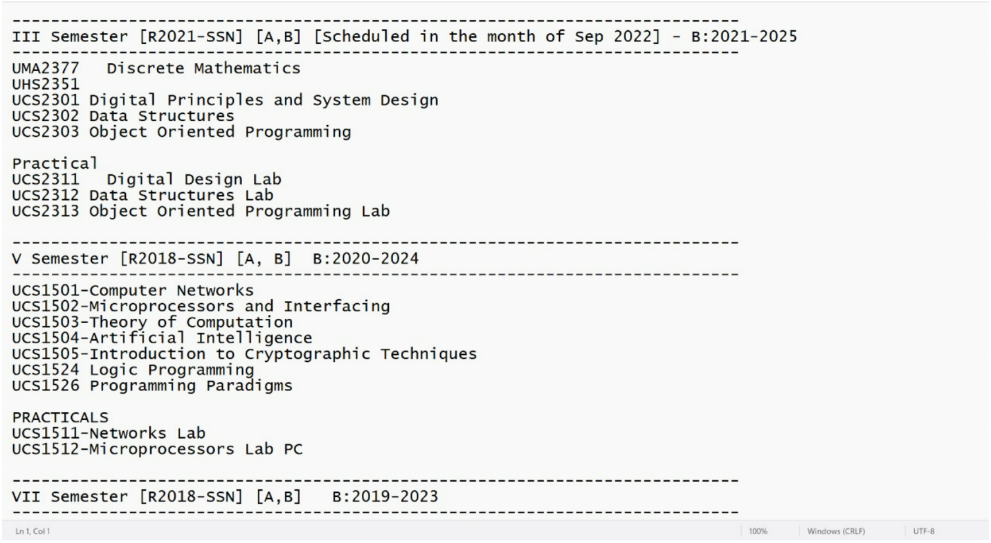
Module 10: Establishing Connectivity to the Python Backend using Flask

Flask is a web framework that helps to integrate the front end of a web application with a Python backend. The Web Server Gateway Interface also called the WSGI is the interface that is used when web pages are made with Python. Flask uses the WSGI to establish this connectivity. Using Flask, a data source from Python can be passed to the HTML, to make a web page with dynamic values.

The output from the program which allots slots to courses is combined with HTML using Flask. In this manner, a multitude of timetables (that of faculty and students, across semesters) can dynamically be displayed.

Implementation

Module 1: Data Preprocessing



```
--------------------------------------------------------------------------------
III Semester [R2021-SSN] [A,B] [Scheduled in the month of Sep 2022] - B:2021-2025
--------------------------------------------------------------------------------
UMA2377   Discrete Mathematics
UHS2351
UCS2301 Digital Principles and System Design
UCS2302 Data Structures
UCS2303 Object Oriented Programming

Practical
UCS2311   Digital Design Lab
UCS2312 Data Structures Lab
UCS2313 Object Oriented Programming Lab

--------------------------------------------------------------------------------
V Semester [R2018-SSN] [A, B]  B:2020-2024
--------------------------------------------------------------------------------
UCS1501-Computer Networks
UCS1502-Microprocessors and Interfacing
UCS1503-Theory of Computation
UCS1504-Artificial Intelligence
UCS1505-Introduction to Cryptographic Techniques
UCS1524 Logic Programming
UCS1526 Programming Paradigms

PRACTICALS
UCS1511-Networks Lab
UCS1512-Microprocessors Lab PC

--------------------------------------------------------------------------------
VII Semester [R2018-SSN] [A,B]   B:2019-2023
--------------------------------------------------------------------------------
```

Figure 3: Original data received as a text file

Algorithm: To write the data into a CSV file

Input: text file with rows of data
Output: CSV(MS Excel) file with the rows of data stored in it

1. Check the given data and remove null values, spaces and tabs and mention whether it is theory, lab or elective using 0/1.
2. myfile←open("data.txt", "r")
3. da←myfile.read()
4. da_l←da.split("\n")
5. da_nl←[]
6. for i in range(len(da_l)):
   6.1: st←da_l[i].split(",")
   6.2: da_nl+=[st]
   END FOR
7. print(da_nl)
8. df←pd.DataFrame(da_nl)
9. writer ← pd.ExcelWriter('edata.xlsx', engine='xlsxwriter')

10. df.to_excel(writer, sheet_name='data1', index=False)
11. writer.save()

```
UMA2377,Discrete Mathematics,3,1,0,0
UHS2351,Elective,3,0,0,1
UCS2301,Digital Principles and System Design,3,1,0,0
UCS2302,Data Structures,3,1,0,0
UCS2303,Object Oriented Programming,3,1,0,0
UCS2311,Digital Design Lab,3,0,1,0
UCS2312,Data Structures Lab,3,0,1,0
UCS2313,Object Oriented Programming Lab,3,0,1,0
UCS1501,Computer Networks,5,1,0,0
UCS1502,Microprocessors and Interfacing,5,1,0,0
UCS1503,Theory of Computation,5,1,0,0
UCS1504,Artificial Intelligence,5,1,0,0
UCS1505,Introduction to Cryptographic Techniques,5,1,0,0
UCS1524,Logic Programming,5,1,0,0
UCS1526,Programming Paradigms,5,1,0,0
UCS1511,Networks Lab,5,0,1,0
UCS1512,Microprocessors Lab PC,5,0,1,0
UCS1701,Distributed Systems,7,1,0,0
UCS1702,Mobile Computing,7,1,0,0
UCS1703,Graphics and Multimedia,7,1,0,0
UCS1704,Management and Ethical Practices,7,1,0,0
UCS1722,Social Network Analysis,7,1,0,0
UCS1723,Deep Learning,7,1,0,0
UCS1729,Data Warehousing and Data Mining,7,1,0,0
UCS1711,Mobile Application Development Lab,7,0,1,0
UCS1712,Graphics and Multimedia Lab,7,0,1,0
```

Figure 4: The file after the removal of Null values and spaces

```
Python 3.10.2 (tags/v3.10.2:a58ebcc, Jan 17 2022, 14:12:15) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: D:\Sem 2\Software Dev\Project- Uni TT\txt2list.py ==========
[['UMA2377', 'Discrete Mathematics', '3', '1', '0', '0'], ['UHS2351', 'Elective', '3', '0', '0', '1'], ['UCS2301', 'Digital Principles and System Design', '3', '1', '0', '0'], ['UCS230
2', 'Data Structures', '3', '1', '0', '0'], ['UCS2303', 'Object Oriented Programming', '3', '1', '0', '0'], ['UCS2311', 'Digital Design Lab', '3', '0', '1', '0'], ['UCS2312', 'Data Str
uctures Lab', '3', '0', '1', '0'], ['UCS2313', 'Object Oriented Programming Lab', '3', '0', '1', '0'], ['UCS1501', 'Computer Networks', '5', '1', '0', '0'], ['UCS1502', 'Microprocessor
s and Interfacing', '5', '1', '0', '0'], ['UCS1503', 'Theory of Computation', '5', '1', '0', '0'], ['UCS1504', 'Artificial Intelligence', '5', '1', '0', '0'], ['UCS1505', 'Introduction
to Cryptographic Techniques', '5', '1', '0', '0'], ['UCS1524', 'Logic Programming', '5', '1', '0', '0'], ['UCS1526', 'Programming Paradigms', '5', '1', '0', '0'], ['UCS1511', 'Networks
Lab', '5', '0', '1', '0'], ['UCS1512', 'Microprocessors Lab PC', '5', '0', '1', '0'], ['UCS1701', 'Distributed Systems', '7', '1', '0', '0'], ['UCS1702', 'Mobile Computing', '7', '1',
'0', '0'], ['UCS1703', 'Graphics and Multimedia', '7', '1', '0', '0'], ['UCS1704', 'Management and Ethical Practices', '7', '1', '0', '0'], ['UCS1722', 'Social Network Analysis', '7',
'1', '0', '0'], ['UCS1723', 'Deep Learning', '7', '1', '0', '0'], ['UCS1729', 'Data Warehousing and Data Mining', '7', '1', '0', '0'], ['UCS1711', 'Mobile Application Development Lab',
'7', '0', '1', '0'], ['UCS1712', 'Graphics and Multimedia Lab', '7', '0', '1', '0']]
>>>
```

Figure 5: Data transfer from a text file to list

FacultyID	Name	CourseID1	CourseID2	CourseID3	Hours1	Hours2	Hours3
MAL1	L1	UMA2377	0	0	4	0	0
MAL2	L2	UMA2377	0	0	4	0	0
HSL3	L3	UHS2351	0	0	3	0	0
HSL4	L4	UHS2351	0	0	3	0	0
CSL5	L5	UCS2301	UCS2501	UCS2711	3	3	3
CSL6	L6	UCS2301	UCS2501	UCS2711	3	3	3
CSL7	L7	UCS2302	UCS2502	UCS2712	3	3	3
CSL8	L8	UCS2302	UCS2502	UCS2712	3	3	3
CSL9	L9	UCS2311	UCS2503	UCS2701	3	3	3
CSL10	L10	UCS2311	UCS2503	UCS2701	3	3	3
CSL11	L11	UCS2312	UCS2504	UCS2702	3	3	3
CSL12	L12	UCS2312	UCS2504	UCS2702	3	3	3
CSL13	L13	UCS2313	UCS2505	UCS2703	3	3	3
CSL14	L14	UCS2313	UCS2505	UCS2703	3	3	3
CSL15	L15	UCS2303	UCS2511	UCS2704	3	3	3
CSL16	L16	UCS2303	UCS2511	UCS2704	3	3	3
CSL17	L17	UPE2551	UCS2512	UCS2705	3	3	3
CSL18	L18	UPE2551	UCS2512	UCS2705	3	3	3
CSL19	L19	UPE2751	UPE2751	0	3	3	0

Figure 6: Data transfer from nested list to CSV file:

Module 2: Edge Colouring

The relationship between lecturers and the courses they take up to teach is in the form of a bipartite graph as there will be no edges between the lecturers or the courses themselves.

- The vertices are in the form of two sets- (i) set of lecturers and (ii) set of courses.
- The edges between a lecturer vertex, say L11, and a course vertex, say UCS1501 indicates that L11 teaches UCS1501.
- The number of edges from each vertex in the Lecturer set corresponds to the number of hours taught by each lecturer for a week across courses.

For constructing the structure of the graph described above the following data is required from the database:

1. List of lecturers
   - the L‘i’th lecturer is stored in the ‘i-1’th position
2. Corresponding list of courses taught by them
   - the course taught by the L‘i’th lecturer is stored in the ‘i-1’th position
3. List of the number of hours per course
   - each element in this two dimensional array is the number of hours taught by the ‘i’th Lecturer for the ‘j’th subject.

Algorithm

Input: Records from the above mentioned csv file
Output: List of lecturers, Corresponding list of courses taught by them, List of the number of hours per course

```
import csv
Define a function getdata() which performs the following tasks
   file←"main.csv"
   data←[]
   with open(file,"r") as csvfile:
      reader←csv.reader(csvfile)
      for every row in reader
         Add row to data
      H←[]
      L←[]
      teach←[]
      for i i=1 to len(data))
         Add data[i][1] to L
         if i%2=0
            Then, Add ([data[i][2]+" B "+str(data[i][5]),data[i][3]+" B "+str(data[i][6]),data[i][4]+" B
"+str(data[i][7])]) to teach
         Otherwise,
            Add ([data[i][2]+" A "+str(data[i][5]),data[i][3]+" A "+str(data[i][6]),data[i][4]+" A
"+str(data[i][7])]) to teach
      distinct_class←[]
```

```
for every element, i in teach
   for every element j in i:
      if (j not in distinct_class) and ('NULL' not in j)
         Then, Add j to distinct_class
for i=0 to len(L)
   k←[]
   c←0
   for j=0 to len(distinct_class)
      if distinct_class[j] in teach[i]:
         Add (int(teach[i][teach[i].index(distinct_class[j])][-1])) to k
      Otherwise
         Add 0 to k
       End if
      C←c+1
    End for
   Add k to H
End for
return H,L,teach
```

Given below are the H, L and teach arrays respectively. There are the 3 important arrays based on which further processes are carried out:

1. List of lecturers
   - the L‘i’th lecturer is stored in the ‘i-1’th position

Table 2: List of lecturers

Index (i-1)	0	1	2
‘i’th Lecturer	Lecturer 1	Lecturer 2	Lecturer 3

2. Corresponding list of courses taught by them
   - the course taught by the L‘i’th lecturer is stored in the ‘i-1’th position

Table 3: List of courses taught by each lecturer

Index (i-1)	0	1	2
‘i’th Lecturer Details	Array of courses taught by Lecturer 1	Array of courses taught by Lecturer 2	Array of courses taught by Lecturer 3

3. List of the number of hours per course
   - Each element in this two dimensional array is the number of hours taught by the ‘i+1’th Lecturer for the ‘j+1’th subject. Here, ‘S’ denotes a course for a particular semester for a particular section.

Table 4: List of number of hours per course

‘i+1’th Lecturer→	0	1

‘j+1’th Subject ↓		
0	Number of hours L1 is teaching S1	Number of hours L1 is teaching S2
1	Number of hours L2 is teaching S1	Number of hours L2 is teaching S2

Algorithm

The algorithm has been decomposed into 3 functions, each carrying out necessary processes for the Edge Colouring process.

Part 1:

Identifying and applying edge colouring on the vertex with the maximum degree:

- It is easier to perform edge colouring on a graph after identifying the vertex with the maximum degree as the Kőnig's Theorem theorem states that the minimum number of colours or the chromatic index is equal to the maximum degree of the graph. Hence, the ‘largest degree first’ idea is used.
- The function Colouring() implements this aspect of the algorithm

Algorithm

Input: list of lecturers, corresponding list of courses taught by them, number of hours per course
Output: maximum degree of the graph, the lecturer corresponding to the maximum degree, all possible chromatic numbers, array consisting of edge details such as vertices, hour and colour label

Defining a function Colouring() with H1,teach1 as parameters

```
1.max_degree←sum(H[0])
2.//initialising a random possible value for the maximum degree
3.max_lec←0
4.//initialising a random possible value for the lecturer with the maximum degree
5.//to obtain maximum degree of the bipartite graph
6.index←0
7.for every element i in matrix H
        if sum(i)>max_degree
        max_degree←sum(i)
        max_lec←index
        index←index+1

8.//max_degree represents the chromatic number according to Konig's Theorem
9.Colours←[]
10.for n = 1 to max_degree
        Add n to Colours array
11.Edges←[]
```

```
12.k←0#represents each subject - dimension of each small array
13.temp←0
14.for every j in array H[max_lec]:
15.if j!=0:
for hour=1 to j
        Add (["L"+str(max_lec+1),teach1[max_lec][k],hour,Colours[temp]]) to Edges
        temp←temp+1
  k←k+1
16.return max_lec,Edges,Colours
```

Given below is the Colours array:

1. **Colours :** array of all possible colour IDs.

1	2	3	4	5	6	7	8	9

Part 2:

Checking whether the vertices of a given edge are coinciding with the vertices of an already coloure edge:

- If an edge is already coloured, then, the same colour must not be assigned to another edge sharing at least one vertex with the existing one. Hence, a thorough checking must be done to avoid incorrect labelling of edges.
- The function check() implements this aspect of the algorithm

Algorithm

Input: list of lecturers, corresponding list of courses taught by them, number of hours per course, maximum degree of the graph, the lecturer corresponding to the maximum degree, all possible chromatic numbers, array consisting of edge details such as vertices, hour and colour label
Output: a valid colour label

```
1.Defining a function check() with parameters temp,index,k,Edges,Colours,teach1
    2.arr←[]
    3.for every element x in 2D array Edges
        4.if ((x[0]="L"+str(index+1)) or (x[1]=teach1[index][k])):
        n←x[3]
        Add n to arr
        5.if Colours[temp] not in arr
            return Colours[temp]
        Otherwise
            return check((temp+1)%len(Colours),index,k,Edges,Colours,teach1)
```

Part 3:

Continuing the edge colouring process for the rest of the graph excluding the vertex with maximum degree for which the process was already performed:

- If an edge is already coloured, then, the same colour must not be assigned to another edge sharing at least one vertex with the existing one. A thorough checking is done to avoid incorrect labelling of edges.
- Then, the valid colour label is assigned to the new edge created.
- The function Continue() implements this aspect of the algorithm

Algorithm
Input: list of lecturers, corresponding list of courses taught by them, number of hours per course, maximum degree of the graph, the lecturer corresponding to the maximum degree, all possible chromatic numbers, array consisting of edge details such as vertices, hour and colour label, a valid colour label
Output: array consisting of edge details such as vertices, hour and colour label

Defining a function Continue() with parameters (max_lec,Edges,H,teach1,Colours)

```
1.index←0
2.for every i in matrix H
3.k←0
4.temp←0
5.if i!=H[max_lec]:
for every j in array i
        if j!=0:
        for hour=1 to j+1:
        res←check(temp,index,k,Edges,Colours,teach1)
        Add (["L"+str(index+1),teach1[index][k],hour,res]) to Edges
        temp←temp+1
        k←k+1
6.index←index+1
7.return Edges
```

Given below is the Edges arrays:
Edges : array containing details of vertices (i.e, the Lecturer, the course, the semester, the section, total number of hours for the course, the hour signified by the given edge and the colour ID (Example indicates random edges from actual output from the above data)

Table 5: Array with details about each vertex of the graph

Index of Edges Array (Represents every detail for an edge→) (Represents each Edge ↓)	0 Lecturer Name	1 - Course - Semester - Section - Total number of hours for the course	2 hour signified by the given edge	3 Colour ID

0	L5	UCS2301, 3, A, 3	1	1
1	L6	UCS2711, 7, B, 3	1	7
2	L16	UCS2704, 7, B, 3	2	8

Main program for implementing the functions

1.//Array of courses taught by the Lith Lecturer stored in the (i-1)th position
2. teach←eval(input("Making a list of courses taught by Lith Lecturer in the (i-1)th position: "))
3. H_main←eval(input("Enter the number of hours taught by each lecturer for a subject in matrix format: "))
4. a,L,b←Colouring(H_main,teach)
5. Display (Continue(a,L,H_main,teach,b))

Flowcharts:

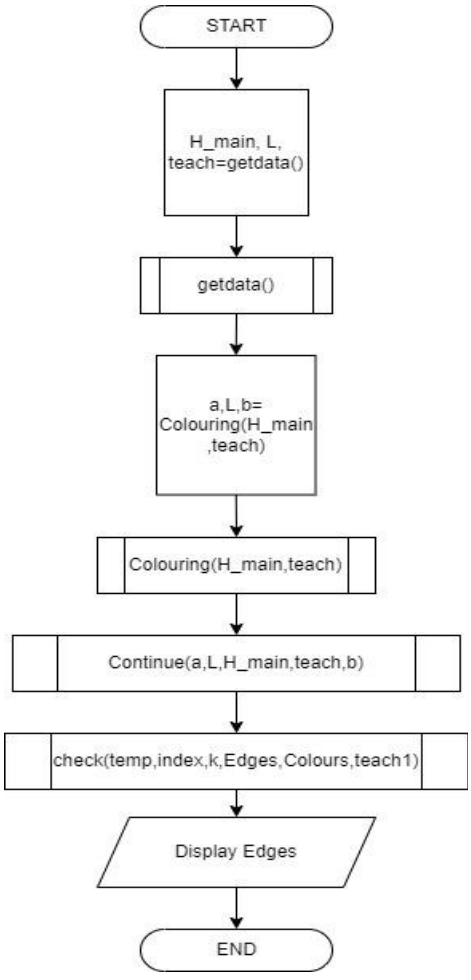


Figure 7: Flowchart depicting the functions called to perform Edge Coloring

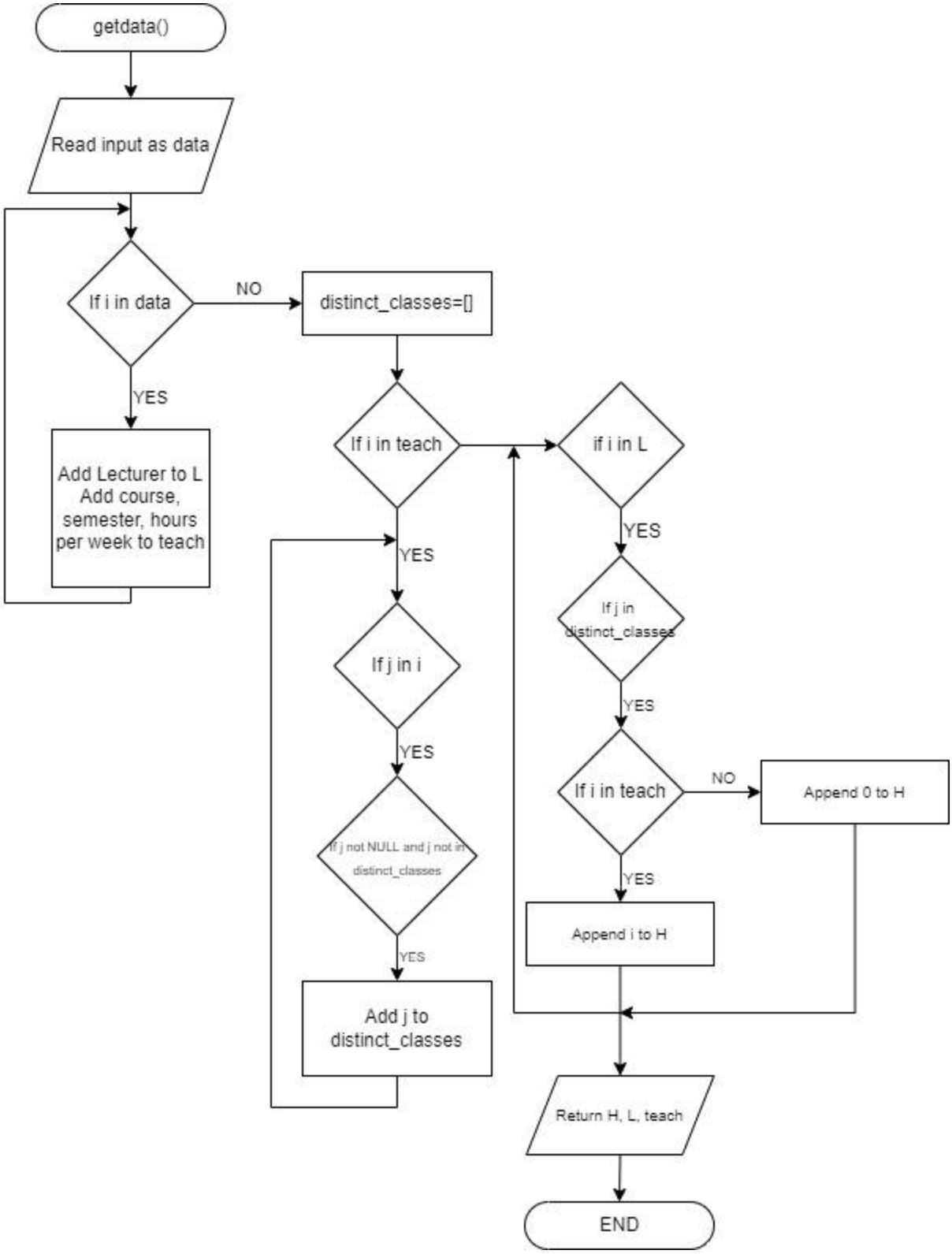


Figure 8: Flowchart for getdata() function

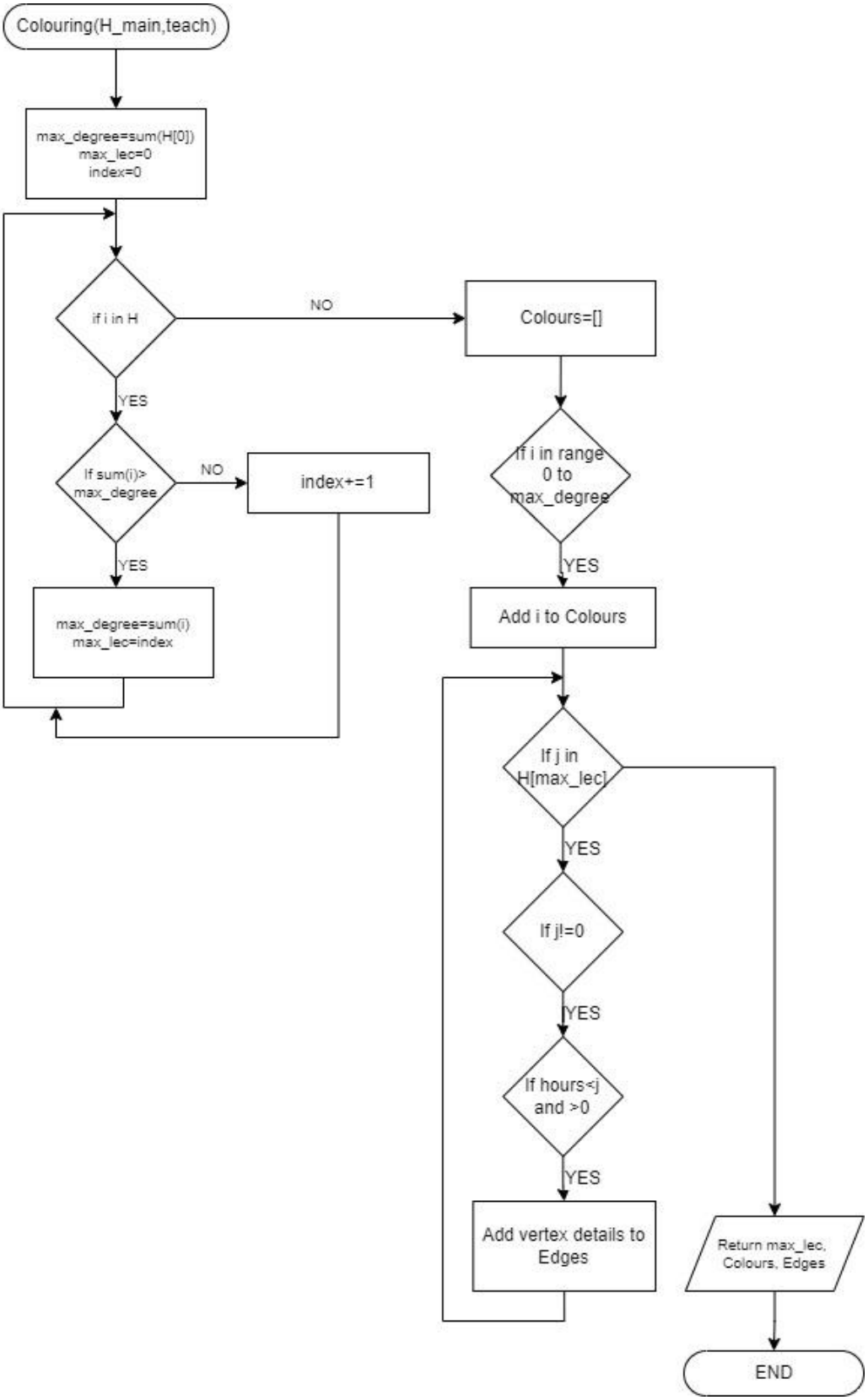


Figure 9: Flowchart for the Continue() function

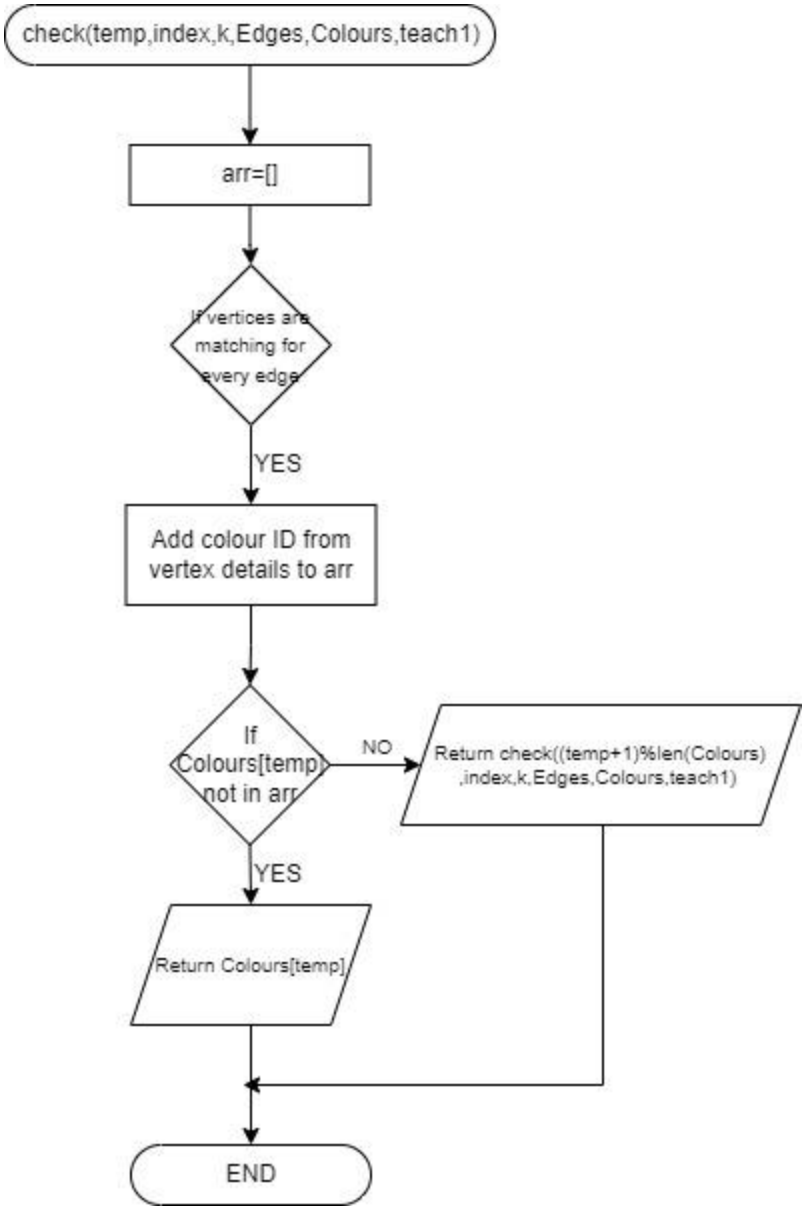


Figure 10: Flowchart for the check() function

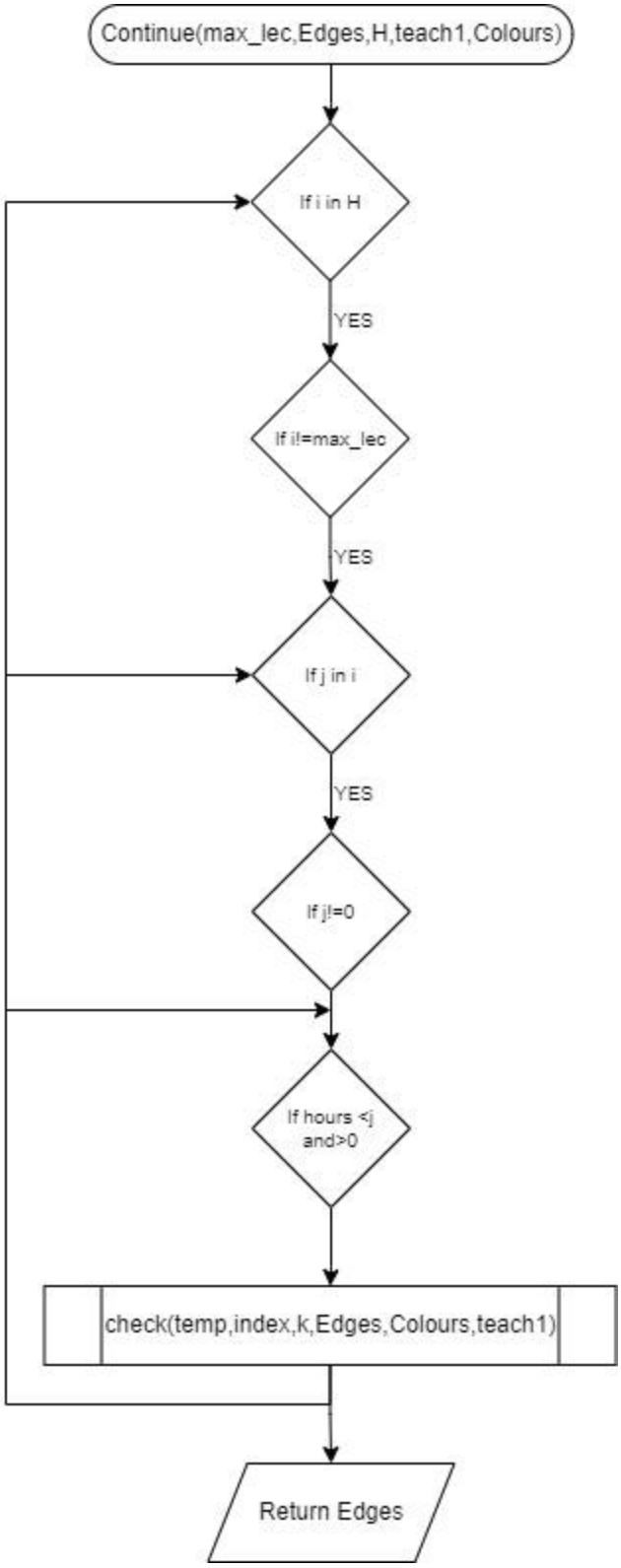


Figure 11: Flowchart for the Continue() function

Output:

The output of the above code is the ‘Edges’ array containing all the necessary details of the graph, namely, the Lecturer, the course, the semester, the section, total number of hours for the course, the hour signified by the given edge and the colour ID.

This array can be transformed into a table in the database and sorted based on colour IDs. The column under each colour represents the classes which can occur at once without any conflict of Lecturers as well as Courses.

Figure 12: Hand Trace Outcome for Checking ‘L’ and ‘teach’ arrays:

```
============== RESTART: C:\COLLEGE\studio project\edgecolouring.py =============
['L1', 'L2', 'L3', 'L4', 'L5', 'L6', 'L7', 'L8', 'L9', 'L10', 'L11', 'L12', 'L13', 'L14', 'L15', 'L16', 'L17', 'L18', 'L19']



[['UMA2377 3 A 4', 'NULL 5 A 0', 'NULL 7 A 0'], ['UMA23773 B 4', 'NULL 5 B 0', 'NULL 7 B 0'], ['UHS2351 3 A 3', 'NULL 5 A 0', 'NULL 7 A 0'], ['UHS23513 B 3', 'NULL 5 B 0', 'NULL 7 B 0'], ['UCS2301 3 A 3', 'UCS2501 5 A 3', 'UCS2711 7 A 3'], ['UCS23013 B 3', 'UCS2501 5 B 3', 'UCS2711 7 B 3'], ['UCS2302 3 A 3', 'UCS2502 5 A 3', 'UCS2712 7 A 3'], ['UCS23023 B 3', 'UCS2502 5 B 3', 'UCS2712 7 B 3'], ['UCS2311 3 A 3', 'UCS2503 5 A 3', 'UCS2701 7 A 3'], ['UCS23113 B 3', 'UCS2503 5 B 3', 'UCS2701 7 B 3'], ['UCS2312 3 A 3', 'UCS2504 5 A 3', 'UCS2702 7 A 3'], ['UCS23123 B 3', 'UCS2504 5 B 3', 'UCS2702 7 B 3'], ['UCS2313 3 A 3', 'UCS2505 5 A 3', 'UCS2703 7 A 3'], ['UCS23133 B 3', 'UCS2505 5 B 3', 'UCS2703 7 B 3'], ['UCS2303 3 A 3', 'UCS2511 5 A 3', 'UCS2704 7 A 3'], ['UCS23033 B 3', 'UCS2511 5 B 3', 'UCS2704 7 B 3'], ['UPE2551 3 A 3', 'UCS2512 5 A 3', 'UCS2705 7 A 3'], ['UPE25513 B 3', 'UCS2512 5 B 3', 'UCS2705 7 B 3'], ['UPE2751 3 A 3', 'UPE2751 5 A 3', 'NULL 7 A 0']]
```

Figure 13: Hand Trace Outcome for Checking ‘H’ array:

```
[['4', 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, '4', 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, '3', 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, '3', 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, '3', '3', '3', 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, '3', '3', '3', 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, '3', '3', '3', 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, '3', '3', '3', 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, '3', '3', '3', 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, '3', '3', '3', 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, '3', '3', '3', 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, '3', '3', '3', 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, '3', '3', '3', 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, '3', '3', '3', 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, '3', '3', '3', 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, '3', '3', '3', 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, '3', '3', '3', 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, '3', '3', '3', 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, '3', '3']]
```

Figure 14: Hand Trace Outcome for Checking if all the unique classes, that is, course per semester per section, in the table are recognised in ‘distinct_class':

```
48
['UMA2377 3 A 4', 'UMA23773 B 4', 'UHS2351 3 A 3', 'UHS23513 B 3', 'UCS2301 3 A 3', 'UCS2501 5 A 3', 'UCS2711 7 A 3', 'UCS23013 B 3', 'UCS2501 5 B 3', 'UCS2711 7 B 3', 'UCS2302 3 A 3', 'UCS2502 5 A 3', 'UCS2712 7 A 3', 'UCS23023 B 3', 'UCS2502 5 B 3', 'UCS2712 7 B 3', 'UCS2311 3 A 3', 'UCS2503 5 A 3', 'UCS2701 7 A 3', 'UCS23113 B 3', 'UCS2503 5 B 3', 'UCS2701 7 B 3', 'UCS2312 3 A 3', 'UCS2504 5 A 3', 'UCS2702 7 A 3', 'UCS23123 B 3', 'UCS2504 5 B 3', 'UCS2702 7 B 3', 'UCS2313 3 A 3', 'UCS2505 5 A 3', 'UCS2703 7 A 3', 'UCS23133 B 3', 'UCS2505 5 B 3', 'UCS2703 7 B 3', 'UCS2303 3 A 3', 'UCS2511 5 A 3', 'UCS2704 7 A 3', 'UCS23033 B 3', 'UCS2511 5 B 3', 'UCS2704 7 B 3', 'UPE2551 3 A 3', 'UCS2512 5 A 3', 'UCS2705 7 A 3', 'UPE25513 B 3', 'UCS2512 5 B 3', 'UCS2705 7 B 3', 'UPE2751 3 A 3', 'UPE2751 5 A 3']
```

Figure 15: Final Outcome of ‘Edges’ array:

```
================================ RESTART: C:\COLLEGE\studio project\edgecolouring#2.py ================================
[['L5', 'UCS2301 3 A 3', 1, 1], ['L5', 'UCS2301 3 A 3', 2, 2], ['L5', 'UCS2301 3 A 3', 3, 3], ['L5', 'UCS2501 5 A 3', 1, 4], ['L5', 'UCS2501 5 A 3', 2, 5], ['L5', 'UCS2501 5 A 3', 3, 6], ['L5', 'UCS2711 7 A 3', 1, 7], ['L5', 'UCS2711 7 A 3', 2, 8], ['L5', 'UCS2711 7 A 3', 3, 9], ['L1', 'UMA2377 3 A 4', 1, 1], ['L1', 'UMA2377 3 A 4', 2, 2], ['L1', 'UMA2377 3 A 4', 3, 3], ['L1', 'UMA2377 3 A 4', 4, 4], ['L2', 'UMA23773 B 4', 1, 1], ['L2', 'UMA23773 B 4', 2, 2], ['L2', 'UMA23773 B 4', 3, 3], ['L2', 'UMA23773 B 4', 4, 4], ['L3', 'UHS2351 3 A 3', 1, 1], ['L3', 'UHS2351 3 A 3', 2, 2], ['L3', 'UHS2351 3 A 3', 3, 3], ['L4', 'UHS23513 B 3', 1, 1], ['L4', 'UHS23513 B 3', 2, 2], ['L4', 'UHS23513 B 3', 3, 3], ['L6', 'UCS23013 B 3', 1, 1], ['L6', 'UCS23013 B 3', 2, 2], ['L6', 'UCS23013 B 3', 3, 3], ['L6', 'UCS2501 5 B 3', 1, 4], ['L6', 'UCS2501 5 B 3', 2, 5], ['L6', 'UCS2501 5 B 3', 3, 6], ['L6', 'UCS2711 7 B 3', 1, 7], ['L6', 'UCS2711 7 B 3', 2, 8], ['L6', 'UCS2711 7 B 3', 3, 9], ['L7', 'UCS2302 3 A 3', 1, 1], ['L7', 'UCS2302 3 A 3', 2, 2], ['L7', 'UCS2302 3 A 3', 3, 3], ['L7', 'UCS2502 5 A 3', 1, 4], ['L7', 'UCS2502 5 A 3', 2, 5], ['L7', 'UCS2502 5 A 3', 3, 6], ['L7', 'UCS2712 7 A 3', 1, 7], ['L7', 'UCS2712 7 A 3', 2, 8], ['L7', 'UCS2712 7 A 3', 3, 9], ['L8', 'UCS23023 B 3', 1, 1], ['L8', 'UCS23023 B 3', 2, 2], ['L8', 'UCS23023 B 3', 3, 3], ['L8', 'UCS2502 5 B 3', 1, 4], ['L8', 'UCS2502 5 B 3', 2, 5], ['L8', 'UCS2502 5 B 3', 3, 6], ['L8', 'UCS2712 7 B 3', 1, 7], ['L8', 'UCS2712 7 B 3', 2, 8], ['L8', 'UCS2712 7 B 3', 3, 9], ['L9', 'UCS2311 3 A 3', 1, 1], ['L9', 'UCS2311 3 A 3', 2, 2], ['L9', 'UCS2311 3 A 3', 3, 3], ['L9', 'UCS2503 5 A 3', 1, 4], ['L9', 'UCS2503 5 A 3', 2, 5], ['L9', 'UCS2503 5 A 3', 3, 6], ['L9', 'UCS2701 7 A 3', 1, 7], ['L9', 'UCS2701 7 A 3', 2, 8], ['L9', 'UCS2701 7 A 3', 3, 9], ['L10', 'UCS23113 B 3', 1, 1], ['L10', 'UCS23113 B 3', 2, 2], ['L10', 'UCS23113 B 3', 3, 3], ['L10', 'UCS2503 5 B 3', 1, 4], ['L10', 'UCS2503 5 B 3', 2, 5], ['L10', 'UCS2503 5 B 3', 3, 6], ['L10', 'UCS2701 7 B 3', 1, 7], ['L10', 'UCS2701 7 B 3', 2, 8], ['L10', 'UCS2701 7 B 3', 3, 9], ['L11', 'UCS2312 3 A 3', 1, 1], ['L11', 'UCS2312 3 A 3', 2, 2], ['L11', 'UCS2312 3 A 3', 3, 3], ['L11', 'UCS2504 5 A 3', 1, 4], ['L11', 'UCS2504 5 A 3', 2, 5], ['L11', 'UCS2504 5 A 3', 3, 6], ['L11', 'UCS2702 7 A 3', 1, 7], ['L11', 'UCS2702 7 A 3', 2, 8], ['L11', 'UCS2702 7 A 3', 3, 9], ['L12', 'UCS23123 B 3', 1, 1], ['L12', 'UCS23123 B 3', 2, 2], ['L12', 'UCS23123 B 3', 3, 3], ['L12', 'UCS2504 5 B 3', 1, 4], ['L12', 'UCS2504 5 B 3', 2, 5], ['L12', 'UCS2504 5 B 3', 3, 6], ['L12', 'UCS2702 7 B 3', 1, 7], ['L12', 'UCS2702 7 B 3', 2, 8], ['L12', 'UCS2702 7 B 3', 3, 9], ['L13', 'UCS2313 3 A 3', 1, 1], ['L13', 'UCS2313 3 A 3', 2, 2], ['L13', 'UCS2313 3 A 3', 3, 3], ['L13', 'UCS2505 5 A 3', 1, 4], ['L13', 'UCS2505 5 A 3', 2, 5], ['L13', 'UCS2505 5 A 3', 3, 6], ['L13', 'UCS2703 7 A 3', 1, 7], ['L13', 'UCS2703 7 A 3', 2, 8], ['L13', 'UCS2703 7 A 3', 3, 9], ['L14', 'UCS23133 B 3', 1, 1], ['L14', 'UCS23133 B 3', 2, 2], ['L14', 'UCS23133 B 3', 3, 3], ['L14', 'UCS2505 5 B 3', 1, 4], ['L14', 'UCS2505 5 B 3', 2, 5], ['L14', 'UCS2505 5 B 3', 3, 6], ['L14', 'UCS2703 7 B 3', 1, 7], ['L14', 'UCS2703 7 B 3', 2, 8], ['L14', 'UCS2703 7 B 3', 3, 9], ['L15', 'UCS2303 3 A 3', 1, 1], ['L15', 'UCS2303 3 A 3', 2, 2], ['L15', 'UCS2303 3 A 3', 3, 3], ['L15', 'UCS2511 5 A 3', 1, 4], ['L15', 'UCS2511 5 A 3', 2, 5], ['L15', 'UCS2511 5 A 3', 3, 6], ['L15', 'UCS2704 7 A 3', 1, 7], ['L15', 'UCS2704 7 A 3', 2, 8], ['L15', 'UCS2704 7 A 3', 3, 9], ['L16', 'UCS23033 B 3', 1, 1], ['L16', 'UCS23033 B 3', 2, 2], ['L16', 'UCS23033 B 3', 3, 3], ['L16', 'UCS2511 5 B 3', 1, 4], ['L16', 'UCS2511 5 B 3', 2, 5], ['L16', 'UCS2511 5 B 3', 3, 6], ['L16', 'UCS2704 7 B 3', 1, 7], ['L16', 'UCS2704 7 B 3', 2, 8], ['L16', 'UCS2704 7 B 3', 3, 9], ['L17', 'UPE2551 3 A 3', 1, 1], ['L17', 'UPE2551 3 A 3', 2, 2], ['L17', 'UPE2551 3 A 3', 3, 3], ['L17', 'UCS2512 5 A 3', 1, 4], ['L17', 'UCS2512 5 A 3', 2, 5], ['L17', 'UCS2512 5 A 3', 3, 6], ['L17', 'UCS2705 7 A 3', 1, 7], ['L17', 'UCS2705 7 A 3', 2, 8], ['L17', 'UCS2705 7 A 3', 3, 9], ['L18', 'UPE25513 B 3', 1, 1], ['L18', 'UPE25513 B 3', 2, 2], ['L18', 'UPE25513 B 3', 3, 3], ['L18', 'UCS2512 5 B 3', 1, 4], ['L18', 'UCS2512 5 B 3', 2, 5], ['L18', 'UCS2512 5 B 3', 3, 6], ['L18', 'UCS2705 7 B 3', 1, 7], ['L18', 'UCS2705 7 B 3', 2, 8], ['L18', 'UCS2705 7 B 3', 3, 9], ['L19', 'UPE2751 3 A 3', 1, 1], ['L19', 'UPE2751 3 A 3', 2, 2], ['L19', 'UPE2751 3 A 3', 3, 3], ['L19', 'UPE2751 5 A 3', 1, 4], ['L19', 'UPE2751 5 A 3', 2, 5], ['L19', 'UPE2751 5 A 3', 3, 6]]
```

Module 3: Intermediary Processing of Data

The intermediary processing of data involves the storing of the edge colouring output into a CSV files. The data is first separated into two CSV files, based on whether it is a theory course or a practical courses, based on the label associated with it. In each file, the records are sorted by colour id and then by semester, so that efficient allocation can be made.

Part 1- Storing the records into a CSV File

Algorithm

Input: nested list

Output: CSV(MS Excel) file with the rows of data stored in it.

1. START
2. Import pandas.
3. nl=[]
4. for i in range(len(edgeColor_op))
   - 1.1: st←edgeColor_op[i].split(",")
   - 1.2: nl+=[st]

   END FOR
5. print(nl)
6. def inEx(list,fname,sheetname):
   - df←pd.DataFrame(list)
   - writer ← pd.ExcelWriter(fname, engine='xlsxwriter')
   - df.to_excel(writer, sheet_name=sheetname, index=False)

writer.save()

7. nlt=[], nlp=[]

Part 2: Splitting of courses as Theory (T) and Practical (P) courses

```
for i in range(len(edgeColor_op))
        7.1:  st←edgeColor_op[i].split(",")
        7.2: if st[5]=='T'
                10.2.1:  nlt+=[st]
        7.3: if st[5]=='P'
                10.3.1: nlp+=[st]
        END IF
END FOR
```

Part 3: Sorting the data based on the colour id

1. inEx(nlt, 'edata.xslx', 'theory')
2. inEx(nlp, 'edata.xslx', 'prac')
3. i=0,j=0
4. for i in range(len(edgeColor_op))
        11.1: if edgeColor_op[i][2]>edgeColor_op[i+1][5]
                11.1.1: t=edgeColor_op[i][2]
                11.1.2: edgeColor_op[i][2]=edgeColor_op[i+1][2]
                11.1.3: edgeColor_op[i+1][2]=t
        END IF
    END FOR
5. for i in range(len(edgeColor_op))
        12.1: if edgeColor_op[i][1][7]]>edgeColor_op[i+1][1][7]
                12.1.1: t=edgeColor_op[i][1][7]
                12.1.2: edgeColor_op[i][1][7]=edgeColor_op[i+1][1][7]
                12.1.3: edgeColor_op[i+1][1][7]=t
        END IF
    END FOR
6. inEx(edgeColor_op, 'edata.xslx', 'sem&id')
7. STOP

Module 4: Allotment of Lab courses

Lab courses can only be allotted in certain pre-defined positions, to avoid classes from overlapping with lunch hours, and also to ensure that the scheduling is done within working hours. The possible positions where labs can be scheduled are: {{0,1,2},{1,2,3},{3,4,5}} .

The first twenty rows of the CSV file containing the details of lab courses are as follows:

Course ID	Lecturer	Color ID	Semester	Type
UCS1511	L11	1	5	P
UCS1511	L11	2	5	P
UCS1511	L11	3	5	P
UCS1711	L13	2	7	P
UCS1711	L13	3	7	P
UCS1711	L13	4	7	P
UCS1512	L15	3	5	P
UCS1512	L15	4	5	P
UCS1512	L15	5	5	P
UCS1712	L17	4	7	P
UCS1712	L17	5	7	P
UCS1712	L17	6	7	P
UCS2311	L5	5	3	P
UCS2311	L5	6	3	P
UCS2311	L5	7	3	P
UCS2312	L7	7	3	P
UCS2312	L7	8	3	P
UCS2312	L7	9	3	P
UCS2313	L9	6	3	P
UCS2313	L9	7	3	P

Figure 16: Details of Lab courses

Algorithm

Part 1: Choosing Lab slots
Input: A three-hour lab course
Output: Allocation of the lab course in the right semester, satisfying the set constraints

```
Allot← False
pos={{0,1,2},{1,2,3},{3,4,5}}    //Possible hours in which labs can be allocated
If year=3:
        Yr←3
        Year_lst←s3
Else if year=5:
        Yr←5
        Year_lst←s5
Else
        Yr←7
```

```
        Year_lst←s7

while Allot=False:
        Day_choice←0                    //Choosing a day to schedule the lab
        while day_choice=0:
                Ind←random.randint(0,4)
                lst←day[Ind]
                If count(0) in lst=6:           //Checking if no other lab is scheduled that day
                        Day_choice←1
        place←random.choice(pos)                //Choosing one of the possible lab positions
        C_id←0
        for i= 0 to 3:
                lst[place[i]] ←[y[i][0],y[1][2]]   //Allocate period to slot
                C_id← colorid_check(yr,lst,ind)

        //Checks if the same hours on the same day of the week are either unoccupied or have the same
        colour id
        if c_id=1:
                End while
        else:
                Lst[place[i]] ←0
        //In the case of unsuccessful allotment, undo allotment, repeat process
End while
```

Part 2: Validating positioning of a lab with colour ids
(Function: colorid_check(yr,lst,ind)

Input: Year of allotment (Yr), Day of allotment (lst), index (ind)
Output: Boolean True or False

```
If Yr=3:
        Set1 ←s5[ind]
        Set2 ←s7[ind]

Elif Yr=5:
        Set1 ←s3[ind]
        Set2 ←s7[ind]
Else:
        Set1 ←s3[ind]
        Set2 ←s5[ind]
Counter ←0
i ←0
for i=0 to 6:
        if lst[i]!=0:
       if (set1[i]=0 and set2[i]=0):          //Different combinations of values and zeroes
```

```
            counter ←counter+1
        elif type(set1[i])=tuple and type(set2[i])=tuple:
            if set1[i][1]=lst[i][1] and set2[i][1]=lst[i][1]:
                counter ←counter+1
        elif type(set1[i])=tuple and type(set2[i])=int:
            if set1[i][1]=lst[i][1]:
                counter ←counter+1
        elif type(set1[i])=int and type(set2[i])=tuple:
            if set2[i][1]=lst[i][1]:
                counter ←counter+1
        else:
            pass
End for
if counter==3:
      return 1
else:
     return 0
```

Case-wise Analysis of Lab Course Scheduling

The following steps are followed during the scheduling of lab courses:

1. The semester to which the course belongs to, is first checked.
2. A random day is chosen from that semester's timetable.
3. It is checked if another lab has already been scheduled on the same day.
   3.1 If a lab already exists on the chosen day, another day is chosen. This process continues until a suitable day is chosen.
4. A slot is chosen out of all the possible combinations. The course is put into the chosen slot.
5. To verify whether the allocation is correct, the following cases are considered.

Case 1: No other courses occur concurrently in the other two semesters

S3	0	0	0	0	0	0
S5	0	UCS1511,1	UCS1511,2	UCS1511,3	0	0
S7	0	0	0	0	0	0

Figure 17: Case where no other courses occur concurrently

In this situation, the allocation is perfectly valid, as no conflict arises.

Case 2: Other courses are scheduled on the same day, in the same time slots, but the colour ids match.

In this scenario, while attempting to schedule a course for the fifth semester (S5), it is seen that a lab occurs parallelly in the seventh semester (S7). However, the colour ids of the labs match, meaning that they can be scheduled without conflict.

S3	0	0	0	0	0	0
S5	0	UCS1511,1	UCS1511,2	UCS1511,3	0	0
S7	0	0	UCS1711,2	UCS1711,3	UCS1711,4	0

Figure 18: Occurrence of valid concurrent courses

Case 3: Other courses are scheduled on the same day, in the same time slots, but the colour ids do not match.

In this case, it is seen that another lab occurs at the same time as the fifth semester lab to be scheduled, and their colour ids do not match. This means that the two courses cannot be scheduled simultaneously. The allocation made is incorrect, and it has to be undone. The process is repeated until a suitable time slot is obtained for the course.

S3	0	0	0	0	0	0
S5	0	UCS1511,1	UCS1511,2	UCS1511,3	0	0
S7	UCS1711,2	UCS1711,3	UCS1711,4	0	0	0

Figure 19: Occurrence of invalid concurrent courses

S3	0	0	0	0	0	0
S5	0	0	0	0	0	0
S7	UCS1711,2	UCS1711,3	UCS1711,4	0	0	0

Figure 20: Complete reallotment of labs after a failed allocation

Flowchart:

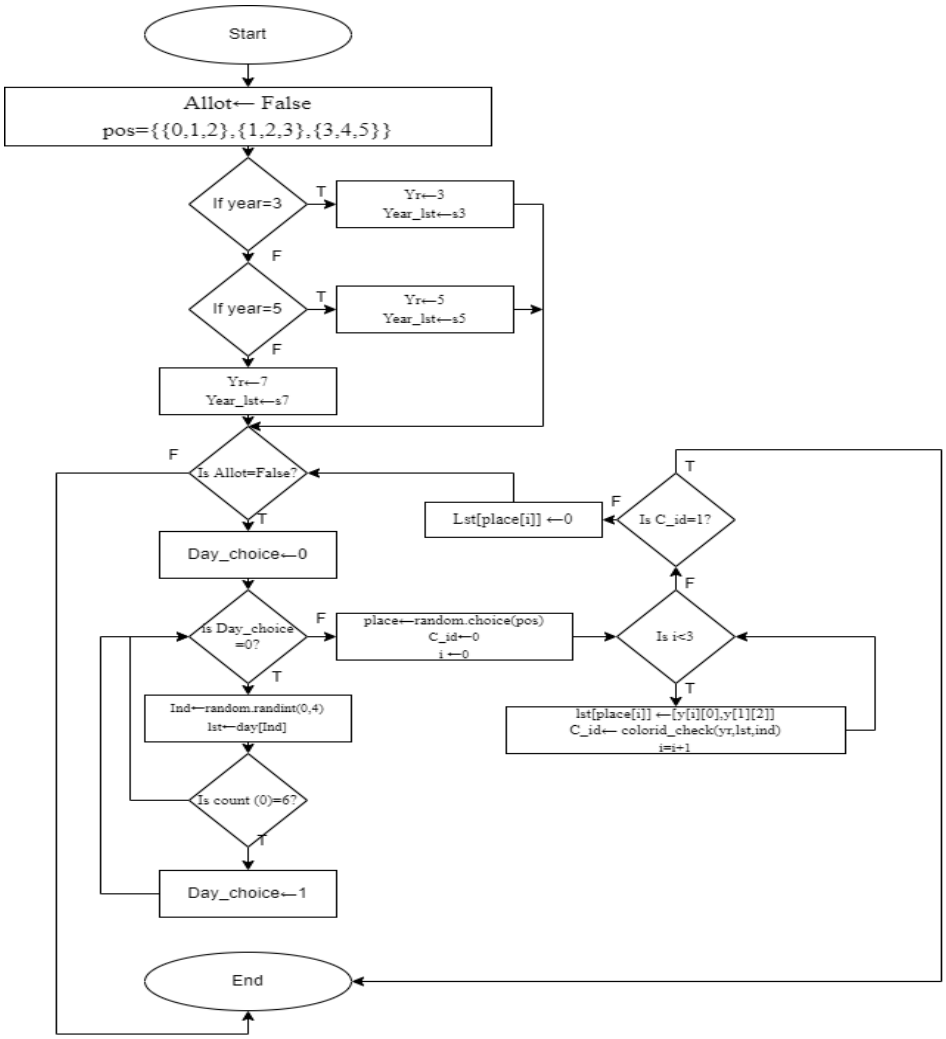


Figure 21: Flowchart depicting allotment of lab courses

Output

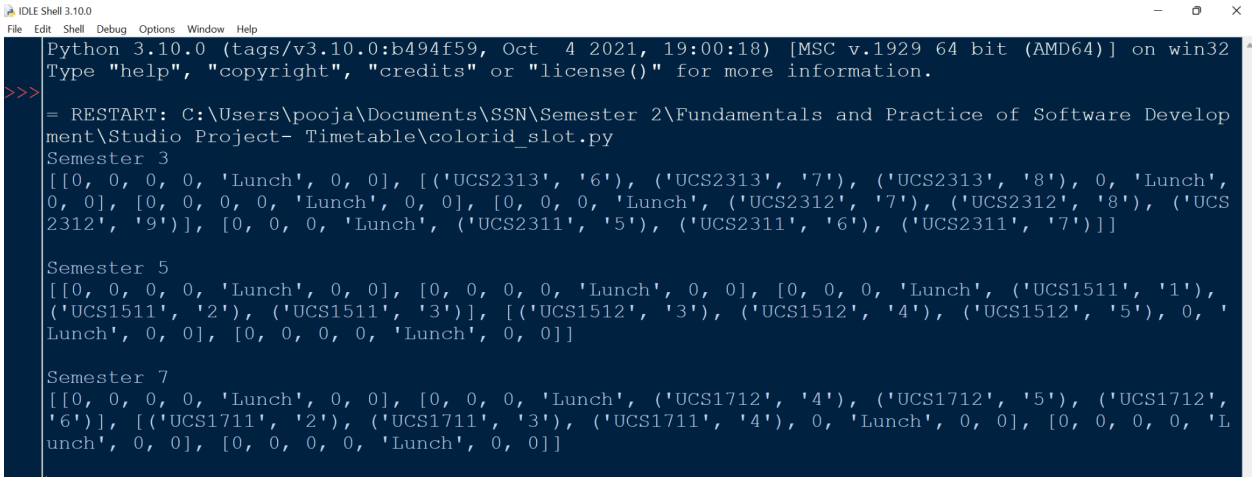


Figure 22: Output of Lab Course Allotment

Module 5: Allotment of Theory Courses

Steps to Allot Theory Courses

1. Sort the list of theory courses by colour IDs and then by semester.
2. Identify the semester the course belongs to.
3. The first level of checking involves the checking of colour IDs, across semesters
   2.1 In order to fill up spaces efficiently, it is initially checked if the same colour ID is already present in the other semesters. .
      2.1.1 Check if the course already occurs on the same day. If not, the same slot is filled in the table corresponding to the semester of the course, after validating if the slot is empty.. Proceed to the allocation of the next course.
      2.1.2 Otherwise, proceed to step 4.
4. Maintain an array consisting of colour IDs that have already been used up.
5. In the case that a course is unable to be allotted based on colour id, check if there are vacant slots corresponding to colour ids that have already been used up.
   5.1 If so, check whether the courses across semesters, having different colour ids are taught by the same lecturer.
   5.2 If they are taught by different lecturers if the same slot in the semester corresponding to the chosen course is empty and if the subject already doesn't occur on that day, a successful allocation can be made.

Thus, the allocation of courses is done giving priority to colour ID, and then by checking the conflict of lecturers.

Algorithm

Input: A set of theory courses, along with their colour ids (theory_allot(y)), where y is a course to be scheduled
Output: Positioning of theory courses, satisfying the set constraints

```
Val ←[y[0],y[2]]                    //Obtaining the course ID and the color id
Allot ←False
  if y[3]='3':                      //Checking the year in which the course has to be scheduled
    Yr ←3
    Year_lst ←s3
  elif y[3]='5':
    Yr ←5
    Year_lst ←s5
  else:
    Yr ←7
    Year_lst ←s7

while allot =False:
    for i=0 to 6:
      if val in year_lst[i]:
        continue
      else:
        Day ←year_lst[i]
        Ind ←i
         break
     End for

    if yr=3:        //Determining which other years the current year must be checked with
      Set1 ←s5[ind]
      Set2 ←s7[ind]
    elif yr=5:
      Set1 ←s3[ind]
      Set2 ←s7[ind]
    else:
      Set1 ←s3[ind]
      Set2 ←s5[ind]

   for j =0 to 6:
      if type(set1[j])=tuple:
          if set1[j][1]=val[1]:
          pos←j
```

```
                if day[pos]=0:
                   Day[pos] ←val
                   return
          if type(set2[j])=tuple:
                if set2[j][1]=val[1]:
                pos←j
                if day[pos]=0:
                   Day[pos] ←val
                   return
       else:
          for k=0 to 6):
             if day[k]=0:
                Day[k] ←val
                C_id ←0
                colorid_theory(yr, day,ind)

  //Call the colorid function to check if the allocation is valid
                if c_id=1:
                   return
                else:
                   Day[k]  ←0
//Removing value if the course is improperly placed and then repeating the process
           End for
     End for
End while
```

Hand tracing of Theory Allocation Algorithm

The colour ids of the courses are associated with the following colours. The courses are sorted on the basis of colour id and then based on semester

1	
2	
3	
4	
5	
6	
7	
8	
9	

Semester 3

	1	2	3	4	5	6
Monday	UHS2351	UMA2377	UMA2377	UCS2312,7	UCS2312,8	UCS2312,9
Tuesday	UCS2313,6	UCS2313,7	UCS2313,8	UHS2351	UCS2301	
Wednesday	UCS2301	UMA2377	UCS2301		UCS2302	
Thursday	UCS2311,5	UCS2311,6	UCS2311,7		UCS2303	UCS2302
Friday	UHS2351	UCS2303	UMA2377	UCS2302	UCS2303	

Semester 5

	1	2	3	4	5	6
Monday	UCS1512,3	UCS1512,4	UCS1512,5	UCS1526	UCS1526	UCS1501
Tuesday	UCS1502	UCS1505		UCS1511,1	UCS1511,2	UCS1511,3
Wednesday	UCS1501	UCS1524	UCS1503	UCS1502	UCS1504	UCS1503
Thursday	UCS1524	UCS1526	UCS1504	UCS1505	UCS1501	
Friday	UCS1505	UCS1502	UCS1503	UCS1504	UCS1524	

Semester 7

	1	2	3	4	5	6
Monday	UCS1701	UCS1722	UCS1729	UCS1702	UCS1702	UCS1702
Tuesday	UCS1722	UCS1723	UCS1722	UCS1701	UCS1703	UCS1729
Wednesday	UCS1703	UCS1711,2	UCS1711,3	UCS1711,4	UCS1704	UCS1729
Thursday	UCS1723	UCS1723		UCS1704	UCS1703	
Friday	UCS1712,4	UCS1712,5	UCS1712,6	UCS1704	UCS1701	

UHS2351	L3	1	3	T
UCS2301	L5	1	3	T
UCS2302	L7	1	3	T
UCS1501	L11	1	5	T
UCS1504	L17	1	5	T
UCS1701	L15	1	7	T
UCS1703	L19	1	7	T
UCS1704	L9	1	7	T
UMA2377	L1	2	3	T
UCS2301	L5	2	3	T
UCS2302	L7	2	3	T
UCS1524	L11	2	5	T
UCS1503	L15	2	5	T
UCS1703	L19	2	7	T
UCS1729	L21	2	7	T
UCS1704	L9	2	7	T
UHS2351	L3	3	3	T
UCS2301	L5	3	3	T
UCS2303	L9	3	3	T
UCS1501	L11	3	5	T
UCS1504	L17	3	5	T
UCS1701	L15	3	7	T
UCS1703	L19	3	7	T
UMA2377	L1	4	3	T
UCS1524	L11	4	5	T
UCS1502	L13	4	5	T
UCS1505	L5	4	5	T
UCS1701	L15	4	7	T
UCS1722	L19	4	7	T
UCS1729	L21	4	7	T
UCS1704	L9	4	7	T
UMA2377	L1	5	3	T
UCS1524	L11	5	5	T
UCS1502	L13	5	5	T
UCS1503	L15	5	5	T
UCS1729	L21	5	7	T
UCS1502	L13	6	5	T
UCS1503	L15	6	5	T
UCS1722	L19	6	7	T
UCS1723	L7	6	7	T
UHS2351	L3	7	3	T
UCS1526	L13	7	5	T
UCS1702	L21	7	7	T
UCS1723	L7	7	7	T
UCS2303	L9	8	3	T
UCS1526	L13	8	5	T
UCS1505	L5	8	5	T
UCS1722	L19	8	7	T
UCS1702	L21	8	7	T
UCS1723	L7	8	7	T
UMA2377	L1	9	3	T
UCS2302	L7	9	3	T
UCS2303	L9	9	3	T
UCS1501	L11	9	5	T
UCS1526	L13	9	5	T
UCS1504	L17	9	5	T
UCS1505	L5	9	5	T
UCS1702	L21	9	7	T

Figure 23: Handtrace outcome of theory and lab courses

The number of constraints imposed on the generation of the timetable means that the required timetable may not be generated in the first run. In the dataset used, there are 79 hours of classes in all, across three semesters. 58 of these are Theory classes, while the remaining 21 are Practical classes. The 21 Practical hours are initially allotted in the module meant for lab allocation. For a given lab allocation, there may not be a plausible solution to allocate all the remaining 58 theory hours. Trial and error across multiple numerous program runs show that successful allotment can vary anywhere between 50 and 58 hours of classes. A timetable is valid, only if all 58 hours of theory classes are also allotted. In the eventuality that anything lesser than 58 hours of classes is scheduled, the allocation is considered to be invalid, and the nested list maintaining the timetable is completely cleared. Allocation is done again from scratch, right from the lab allocation. The program terminates only once all 79 hours of classes have been scheduled.

The following screenshot shows a scenario where the allotment is scrapped multiple times before a valid solution is arrived at.

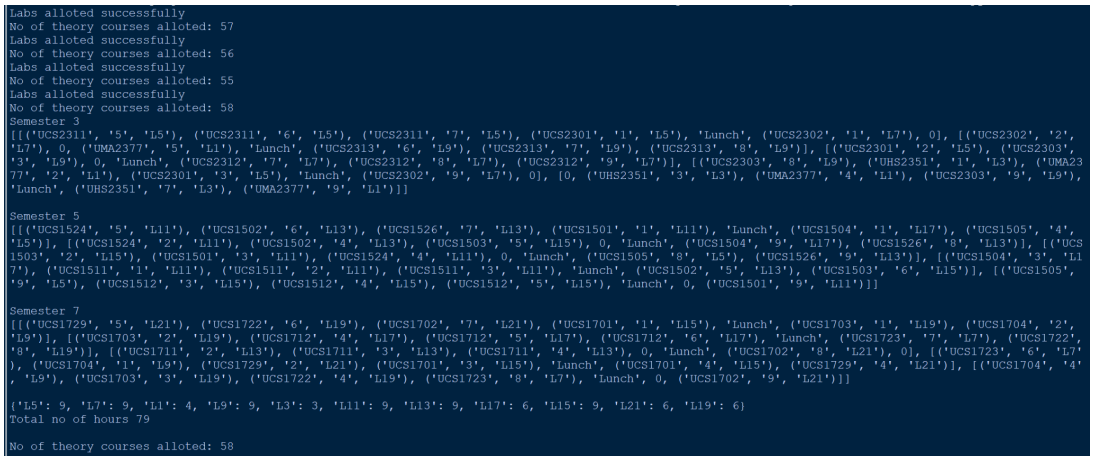


Figure 24: Demonstration of program reruns until a successful allotment is made

In this situation, the program generates a timetable 3 times before a valid timetable is arrived at. The first few attempts allot lesser than 58 hours, and hence the process is continued until all the 58 hours are allotted. Another example is as follows:

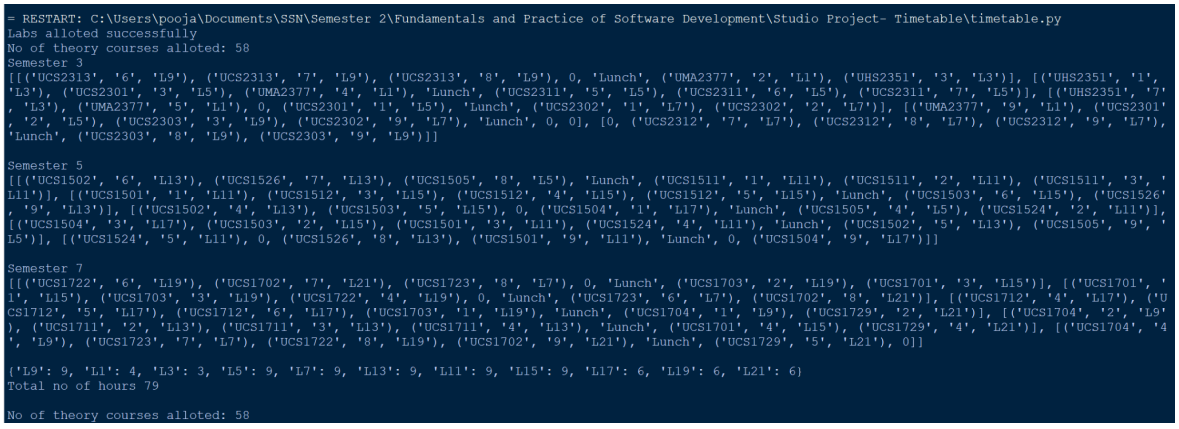


Figure 25: Outcome where all courses are allotted on the very first try

This is the most idealistic scenario, where all courses are put in, in the very first generation.

It is due to the degree of randomness associated with the allocation of lab courses, where days and positions are chosen at random, that such a situation occurs.

Thus, the successful implementation of numerous constraints is seen by the fact that the timetable cannot be generated on the very first try.

Module 6: Generation of Faculty Timetable

After the generation of the timetable across semesters, a function is defined to iterate through all of the student timetables. A dictionary with the keys as the lecturer IDs, and values as a nested list of all the hours of classes a teacher has, is created. Iteration through the dictionary, while dynamically updating and mapping the course, along with semester at the right day and time, in a nested list, generates the faculty timetable.

A sample of the dictionary that is generated after the creation of the timetable is shown below:

Figure 26: A dictionary showing allocation of classes and time slots to teachers

The faculty timetables of two faculty- L3 and L9, in the form of a nested list are displayed below.

```
L3 [[0, 0, 0, 0, 0, 0, 0], [0, 'UHS2351', 0, 0, 0, 0, 0], [0, 'UHS2351', 0, 0, 0, 0, 0], [0, 0, 0, 'UHS2351', 0, 0, 0], [0, 0, 0, 0, 0, 0, 0]]
L9 [[0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 'UCS1704', 'UCS2303'], ['UCS2303', 'UCS1704', 0, 0, 0, 0, 'UCS1704'], [0, 0, 0, 0, 0, 0, 0], ['UCS2313', 'UCS2
313', 'UCS2313', 0, 0, 0, 'UCS2303']]
```

Figure 27: Arrays showing examples of faculty timetables

Module 7: Modularity and Reuse of Code for Different Sections

The programs for each section are made into modules and then called upon. Within each module, a particular printing function alone is called. This means that all the computation happens inside the module, and only the output is passed to the main program body.

The two modules used are module_a.py and module_b.py. The functions to print the outputs are called from each as shown below.

```
import module_b
import module_a
module_b.printing_b()

module_a.printing_b()
```

Figure 28: Importing user-defined library functions

Module 8: Construction of the Webpage

HTML is used for building the structure of the web application. The webpage has the following components:

1. A navigation bar made from unordered list <ul> elements
2. A section with the project's problem statement, constraints and documentation
3. Dynamic links leading to the faculty and student timetables

HTML is also used to design forms to collect user input on the semester and the section, or the faculty ID. It is also used to make the structure of the tables that display the timetable.

The following are the HTML pages created.

1. index.html

Figure 29: Header page of website

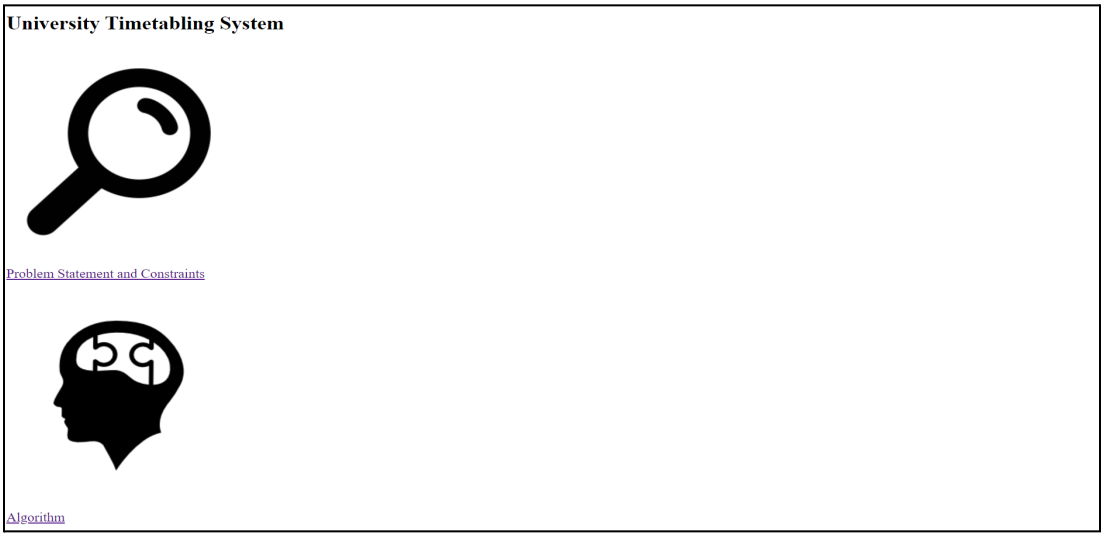


Figure 30: Links to Problem Statement and Algorithm

Figure 31: Buttons for generating student and faculty timetables

2. student.html

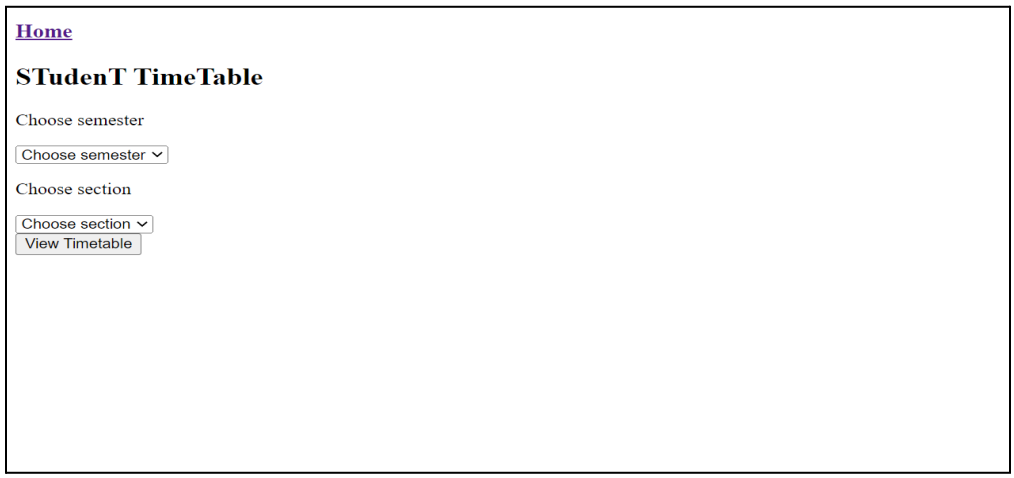


Figure 32: Menu for displaying student timetable

3. faculty.html

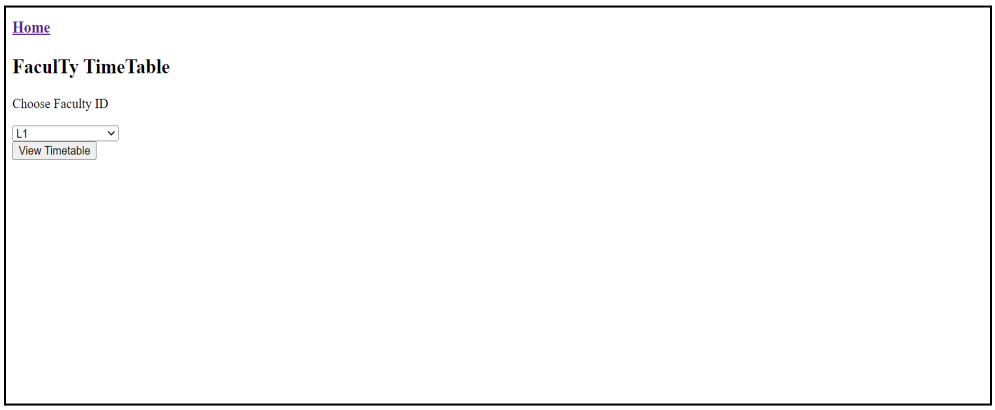


Figure 33: Menu for display faculty timetable

Module 9: Design of the Webpage

The webpage is styled using CSS to make the system more user-friendly, as well as aesthetically pleasing.

The following screenshots show the design of the webpage, after applying CSS styles across all the pages.

1. index.html

Figure 34: Header page of website

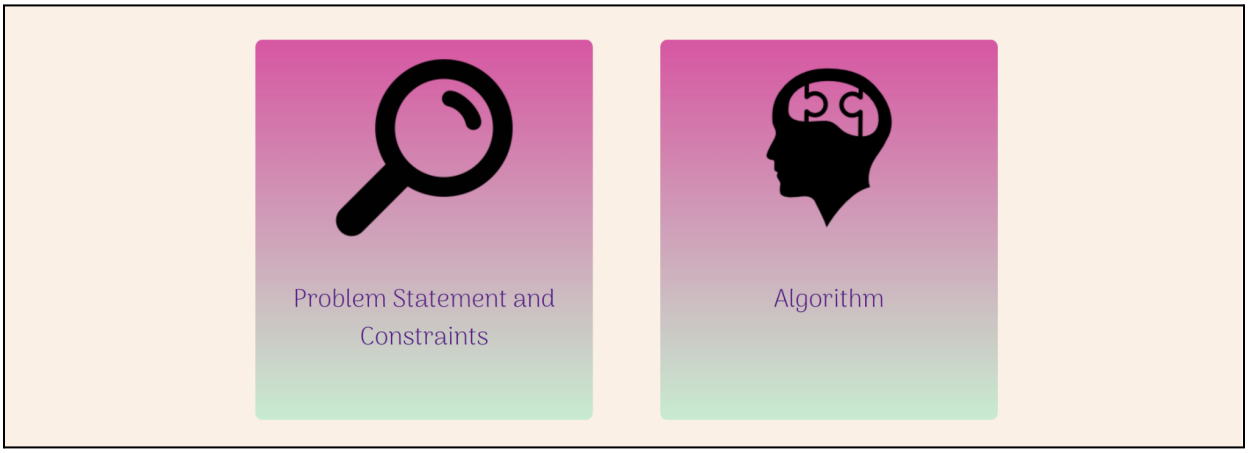


Figure 35: Links to Problem Statement and Algorithm

Figure 36: Buttons for generating student and faculty timetables

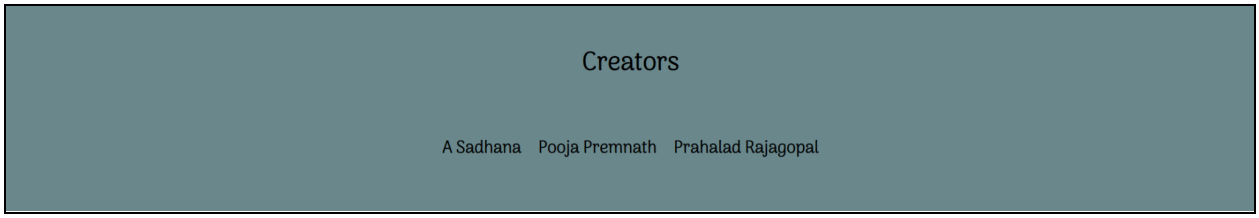


Figure 37 : Footer of website

2. student. html

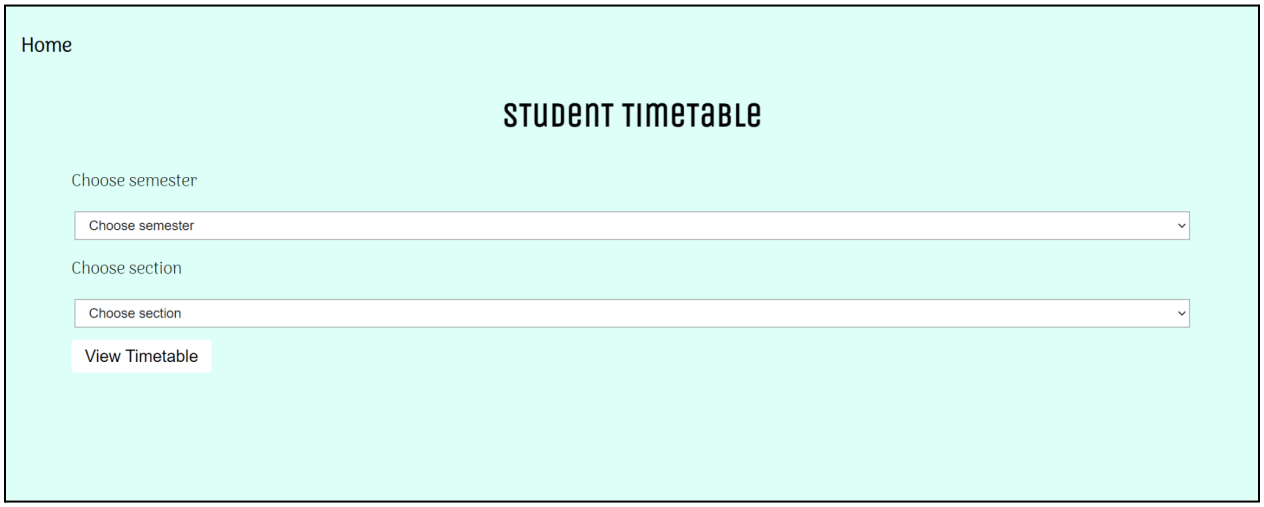


Figure 38: Menu for displaying student timetable

3. faculty.html

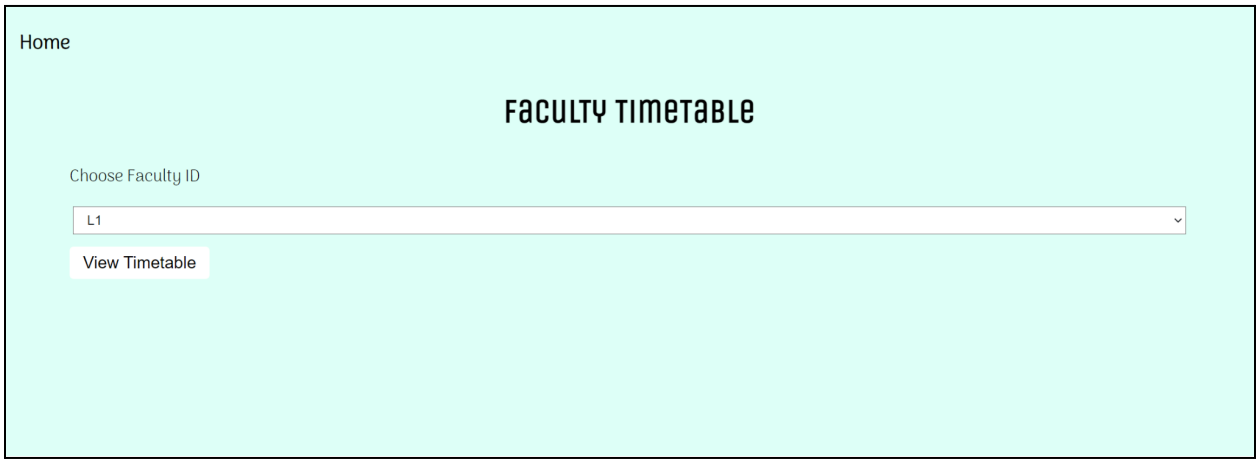


Figure 39: Menu for display faculty timetable

Module 10: Establishing Connectivity to the Python Backend using Flask

The interface between Python and HTML is written using Flask- a web framework. All of the code written in Flask is contained within a file flask_app.py, within which the library functions that generate the timetables are called. Several specific methods are used to perform operations in Flask. Some of them are as follows:

1. Usage of the @app.route() function: This predefined function redirects control to different pages of the site. The program is made modular by defining the tasks to be carried out by each webpage under different @app.route() functions.

2. Usage of the render_template() method: This function calls a template, that is, a specific HTML file to display. This function can be used with conditions, meaning that redirection to different pages can be set up conditionally.

3. Usage of POST and GET: POST and GET are both HTTP methods. The POST method sends information to the web server. This means that the responses to the HTML forms filled up, once submitted are sent to the server, using the POST method. This triggers another action. Otherwise, the GET method is used to render the HTML page.

Result

The output, in the form of student and faculty timetables, is shown below:

Semester 3

Semester: 3 Section: A

8:15am-9:15am	9:15am-10:15am	10:30am-11:30am	11:30am-12:30pm	12:30pm-1:30pm	1:30pm-2:30pm	2:30pm-3:30pm
UHS2351	UCS2303	UMA2377	LIBRARY	Lunch	UCS2301	UCS2302
UCS2301	UCS2303	UMA2377	UCS2302	Lunch	SPORTS	UCS2301
UCS2312	UCS2312	UCS2312	PCD	Lunch	UMA2377	UHS2351
UHS2351	SPORTS	UMA2377	Lunch	UCS2313	UCS2313	UCS2313
UCS2302	UCS2303	PCD	Lunch	UCS2311	UCS2311	UCS2311

Figure 40: Semester 3- Section A Class Timetable

Semester: 3 Section: B

8:15am-9:15am	9:15am-10:15am	10:30am-11:30am	11:30am-12:30pm	12:30pm-1:30pm	1:30pm-2:30pm	2:30pm-3:30pm
UCS2312	UCS2312	UCS2312	MENTOR	Lunch	UMA2377	UHS2351
UHS2351	UMA2377	SPORTS	Lunch	UCS2313	UCS2313	UCS2313
UCS2301	UCS2311	UCS2311	UCS2311	Lunch	UCS2302	MENTOR
UCS2302	UCS2301	MENTOR	UHS2351	Lunch	UMA2377	UCS2303
UMA2377	UCS2301	UCS2303	MENTOR	Lunch	UCS2302	UCS2303

Figure 41: Semester 3- Section B Class Timetable

It can be seen that the timings of Lunch can be varied. By default, Lunch is between 12:30 pm and 1:30 pm. However, if a lab course is scheduled in the afternoon, Lunch would occur between 11:30 pm and 12:30 pm.

To make the timetable more readable, the course code, course name, the type of course and the course instructor are specified in another table, just below the class timetable.

Course Code	Course Name	Type	Course Instructor
UMA2377	Discrete Mathematics	Theory	L2
UHS2351	Elective	Theory	L4
UCS2301	Digital Principles and System Design	Theory	L6
UCS2302	Data Structures	Theory	L8
UCS2303	Object Oriented Programming	Theory	L10
UCS2311	Digital Design Lab	Practical	L6
UCS2312	Data Structures Lab	Practical	L8
UCS2313	Object Oriented Programming Lab	Practical	L10

Figure 42: List of courses for Semester 3

Semester 5

Semester: 5 Section: A

8:15am-9:15am	9:15am-10:15am	10:30am-11:30am	11:30am-12:30pm	12:30pm-1:30pm	1:30pm-2:30pm	2:30pm-3:30pm
UCS1501	UCS1512	UCS1512	UCS1512	Lunch	UCS1504	UCS1502
UCS1524	UCS1501	UCS1505	UCS1503	Lunch	UCS1526	UCS1504
UCS1526	UCS1505	UCS1501	Lunch	UCS1511	UCS1511	UCS1511
UCS1504	UCS1524	UCS1502	PCD	Lunch	UCS1505	UCS1526
UCS1524	UCS1503	UCS1503	LIBRARY	Lunch	UCS1502	PCD

Figure 43: Semester 5- Section A Class Timetable

Semester: 5 Section: B

8:15am-9:15am	9:15am-10:15am	10:30am-11:30am	11:30am-12:30pm	12:30pm-1:30pm	1:30pm-2:30pm	2:30pm-3:30pm
UCS1526	UCS1505	UCS1501	Lunch	UCS1511	UCS1511	UCS1511
UCS1501	UCS1524	UCS1502	SPORTS	Lunch	UCS1505	UCS1526
UCS1504	UCS1524	UCS1502	LIBRARY	Lunch	UCS1505	UCS1503
UCS1524	UCS1512	UCS1512	UCS1512	Lunch	UCS1504	UCS1503
PCD	UCS1503	UCS1501	UCS1502	Lunch	UCS1526	UCS1504

Figure 44: Semester 5- Section B Class Timetable

Course Code	Course Name	Type	Course Instructor
UCS1503	Theory of Computation	Theory	L24
UCS1504	Artificial Intelligence	Theory	L26
UCS1501	Computer Networks	Theory	L12
UCS1502	Microprocessors and Interfacing	Theory	L14
UCS1505	Introduction to Cryptographic Techniques	Theory	L6
UCS1524	Logic Programming	Theory	L12
UCS1526	Programming Paradigms	Theory	L14
UCS1511	Networks Lab	Practical	L12
UCS1512	Microprocessors Lab PC	Practical	L16

Figure 45: List of courses for Semester 5

Semester: 7 Section: A

8:15am-9:15am	9:15am-10:15am	10:30am-11:30am	11:30am-12:30pm	12:30pm-1:30pm	1:30pm-2:30pm	2:30pm-3:30pm
UCS1701	UCS1729	UCS1722	UCS1723	Lunch	UCS1703	UCS1704
UCS1711	UCS1711	UCS1711	UCS1703	Lunch	UCS1704	UCS1701
UCS1702	UCS1722	PCD	SPORTS	Lunch	UCS1729	UCS1703
UCS1701	UCS1712	UCS1712	UCS1712	Lunch	UCS1723	UCS1702
UCS1704	UCS1729	UCS1723	SPORTS	Lunch	UCS1722	UCS1702

Figure 46: Semester 7- Section A Class Timetable

Semester: 7 Section: B

8:15am-9:15am	9:15am-10:15am	10:30am-11:30am	11:30am-12:30pm	12:30pm-1:30pm	1:30pm-2:30pm	2:30pm-3:30pm
UCS1702	UCS1722	LIBRARY	MENTOR	Lunch	UCS1703	UCS1701
UCS1701	UCS1712	UCS1712	UCS1712	Lunch	UCS1723	UCS1702
UCS1703	UCS1729	UCS1722	UCS1723	Lunch	UCS1704	UCS1704
UCS1729	UCS1703	UCS1701	UCS1723	Lunch	UCS1722	UCS1729
UCS1704	UCS1711	UCS1711	UCS1711	Lunch	UCS1702	LIBRARY

Figure 47: Semester7- Section B Class Timetable

Course Code	Course Name	Type	Course Instructor
UCS1701	Distributed Systems	Theory	L26
UCS1702	Mobile Computing	Theory	L22
UCS1703	Graphics and Multimedia	Theory	L20
UCS1704	Management and Ethical Practices	Theory	L10
UCS1722	Social Network Analysis	Theory	L20
UCS1723	Deep Learning	Theory	L8
UCS1729	Data Warehousing and Data Mining	Theory	L22
UCS1711	Mobile Application Development Lab	Practical	L14
UCS1712	Graphics and Multimedia Lab	Practical	L18

Figure 48: List of courses for Semester 7

Faculty Timetable

The faculty timetables for a few lecturers are as shown below:

FACULTY ID: L17

8:15am-9:15am	9:15am-10:15am	10:30am-11:30am	11:30am-12:30pm	12:30pm-1:30pm	1:30pm-2:30pm	2:30pm-3:30pm
-	-	-	-	-	UCS1504 (Sem 5 A)	-
-	-	-	-	-	-	UCS1504 (Sem 5 A)
-	-	-	-	-	-	-
-	UCS1712 (Sem 7 A)	UCS1712 (Sem 7 A)	UCS1712 (Sem 7 A)	-	-	-
-	-	-	-	-	-	-

Figure 49: Faculty Timetable of L17

FACULTY ID: L7

8:15am-9:15am	9:15am-10:15am	10:30am-11:30am	11:30am-12:30pm	12:30pm-1:30pm	1:30pm-2:30pm	2:30pm-3:30pm
-	-	-	UCS1723 (Sem 7 A)	-	-	UCS2302 (Sem 3 A)
-	-	-	UCS2302 (Sem 3 A)	-	-	-
UCS2312 (Sem 3 A)	UCS2312 (Sem 3 A)	UCS2312 (Sem 3 A)	-	-	-	-
-	-	-	-	-	UCS1723 (Sem 7 A)	-
UCS2302 (Sem 3 A)	-	UCS1723 (Sem 7 A)	-	-	-	-

Figure 50: Faculty Timetable of L7

Validation of Output

Since the generation of the timetables across multiple semesters and across two different sections yields such a large number of possible outputs, it becomes very tedious to check whether the key hard constraint of avoiding conflict in scheduling is satisfied in all instances. To overcome this problem, and to check the correctness of the output, a dictionary with the faculty id as the key, and the courses they take, along with the semester, the day of the week and the time slot is created. The semester, day of the week and time slot are all represented as numerical quantities, with the latter two parameters being represented by indices.

For example, a combination of 3 0 3, would imply that the class would be with the third semester, on Monday, in the fourth period. By checking if all of these three number combinations are unique for a given teacher, the doubt of any conflict in scheduling can be eradicated. For example, consider the dictionary keys and corresponding nested list values for a few teachers:

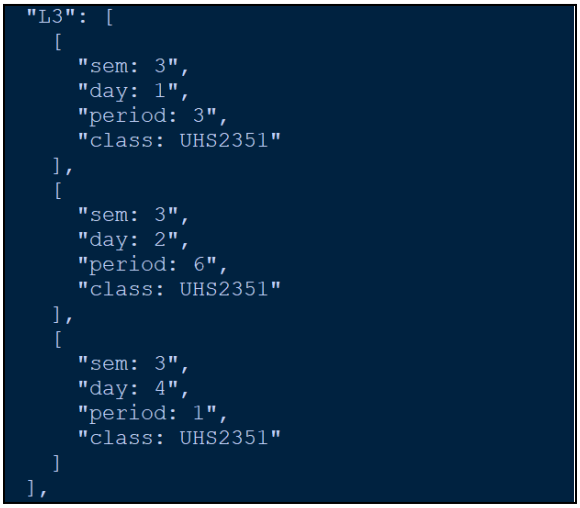


Figure 51 a: Dictionary with allocation of classes for each teacher

For the teacher L3, the different combinations of semesters, days of the week, and time slots are [3 1 3], [3 2 6] and [3 4 1], all of which are unique. This means that L3 does not have any conflict in his/her schedule.

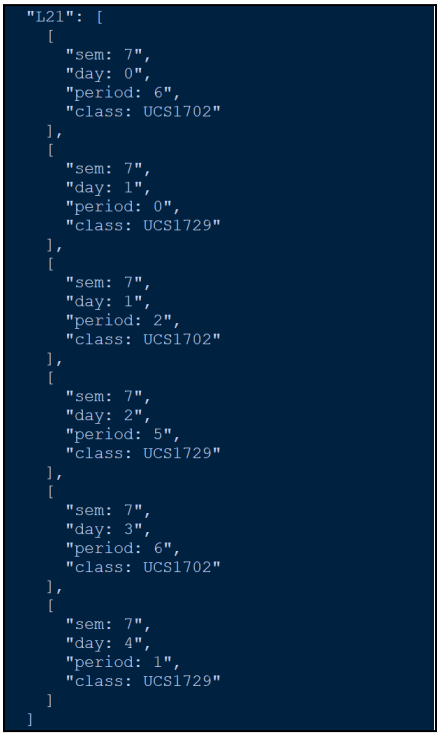


Figure 51 b: Dictionary with allocation of classes for each teacher

For the teacher L21, the different combinations are [7 0 6], [7 1 0], [7 1 2], [7 2 5], [7 3 6] and [7 4 1], all of which are unique, once again.

Thus, the hard constraint of resolving clashes is successfully solved.

Analysis

Efficiency of code

The solution to the problem makes use of graph colouring- specifically edge colouring. This process helps in finding out the courses that can occur simultaneously. Once the edge colours in the form of colour ids are generated, the number of comparisons is greatly reduced, as compared to a brute force approach, wherein checking has to be done at every iteration.

Certain processes in the program such as the process of course allocation for every section across semesters constitutes a basic function. This process of allocation of timetables for different sections can be simplified as it is equivalent to multiple program runs. Thus, rather than rewriting the entire program for different sections, the whole process of allocation of courses can be made into a user-defined library function, which can be called when necessary. This principle of modular programming enhances efficiency, by reducing redundancy.

An Analysis on the number of possible solutions

Based on the constraints implemented in our Time Tabling System, an estimate of the upper bound of the number of solutions generated by the system can be obtained. This estimate includes the possible solutions, however, not all of these solutions are viable. The system developed keeps executing the code until it arrives at a viable solution. This estimate gives us a general idea of how large the solution set is although only the set of viable solutions is obtained as an output from the program rather than the set of intermediary solutions where all of the constraints are not completely satisfied.

Based on the nature of the time table generated by this system, the following calculations are made.

The total number of working hours in a day is 6.
Considering 5 working days in a week, the total number of working hours in a week is, 5x6=30

Table 6: Distribution of lab and theory courses across semesters

Semesters →	Semester 3	Semester 5	Semester 7
Number of Lab Courses	3	2	2
Number of Theory Courses	5	7	7
Total Number of Courses	8	9	9

To calculate the number of ways of allotting lab courses, we must consider each semester.
For semester 3, any 3 out of the 5 working days can be selected and there are 3 possible positions for the allotment of lab courses in a day.

Hence, the number of ways of allotting lab courses to semester 3 is $C_3^5 \times 3$.

Similarly, for the 5th and 7th semesters, any 2 out of the 5 working days can be selected and there are 3 possible positions for the allotment of lab courses in a day.

Hence, the number of ways of allotting lab courses to each semester is $C_2^5 \times 3$.

All these allotments must be made simultaneously, hence, the upper bound of number of possible ways of allotment of lab courses to all the three semesters is $C_3^5 \times 3 \times C_2^5 \times 3 \times C_2^5 \times 3 = 3^3 \times 10^3 =$ 27,000 ways. These allocations must be done twice, for each of the two sections in each semester, hence, 2 x 27,000 = 54,000 ways.

In this timetabling system developed, it is sufficient to consider the allotment of lab courses while estimating the total number of ways the timetable can be generated as the element of randomness exists only during the lab allocation, that is, the days in which the lab courses occur as well as the selection of any of the 3 valid time slots available for lab courses are done in a random manner. The allocation of theory courses revolves around this random allocation of lab courses and is systematic in nature. However, the estimate only indicates an upper limit estimate as it includes the count of certain intermediary solutions wherein constraints such as matching of colour IDs or the total number of course hours allotted are not completely satisfied. Hence, in this system it is safe to say that the total number of ways in which timetables can be generated is less than 54,000 ways.

A generalised system can be formulated based on the above calculations for this timetabling system.

Suppose,
The total number of working hours in a day is ‘n’.
Considering ‘w’ working days in a week, the total number of working hours in a week is, $n \times w$

Table 7: Generalized distribution of courses across semesters

Semesters →	Semester 3	Semester 5	Semester 7
Number of Lab Courses	S3L	S5L	S7L
Number of Theory Courses	S3T	S5T	S7T
Total/ Final Number of Courses	S3F	S5F	S7F

To calculate the number of ways of allotting lab courses, we must consider each semester.

Let there be 'x' possible slots available for lab courses for semester 3, semester 5 and semester 7 in a day without hindering the lunch break or extending the working hours.

For semester 3, any 'S3L' out of the 'n' working days can be selected and there are 'x' possible positions for the allotment of lab courses in a day.
Hence, the number of ways of allotting lab courses to semester 3 is $C_{S3L}^{n} \times x$.

Similarly, for semester 5, any 'S5L' out of the 'n' working days can be selected and there are 'x' possible positions for the allotment of lab courses in a day.
Hence, the number of ways of allotting lab courses to semester 5 is $C_{S5L}^{n} \times x$.

Similarly, for semester 7, any 'S7L' out of the 'n' working days can be selected and there are 'x' possible positions for the allotment of lab courses in a day.
Hence, the number of ways of allotting lab courses to semester 7 is $C_{S7L}^{n} \times x$.

All these allotments must be made simultaneously, hence, the upper bound of number of possible ways of allotment of lab courses to all the three semesters is $C_{S3L}^{n} \times x \times C_{S5L}^{n} \times x \times C_{S7L}^{n} \times x$ ways.
Let there be 'i' number of sections in each semester, then, the upper limit of the number of possible ways of timetable generation is $C_{S3L}^{n} \times x \times C_{S5L}^{n} \times x \times C_{S7L}^{n} \times x \times i$ ways

Suppose, the number of semesters is generalised and varies as s=1,3,5,...,K or s=2,4,6,..,K as only the either the odd semesters or the even semesters can occur simultaneously, then, the upper limit of the total

number of possible timetable generation is

$C^n_{S3L} \times x \times C^n_{S5L} \times x \times C^n_{S7L} \times x \times \ldots \times C^n_{SKL} \times x \times i$ ways where K is always odd, or ,

$C^n_{S2L} \times x \times C^n_{S4L} \times x \times C^n_{S6L} \times x \times \ldots \times C^n_{SKL} \times x \times i$ ways where K is always even.

In this way, it is possible to generalise the timetabling system developed by further incremental development into one that preferentially considers values such as the number of working hours in a day, number of working days in a week, number of semesters, number of sections per semester etc.

Conclusion

Thus, the problem of scheduling a timetable for an academic institution has been successfully solved. The usage of graph colouring to resolve conflicts drastically reduces the number of checks that have to be made, in comparison to using mere brute force. The colour ids assigned to every occurrence of a class helps resolve the most important constraint in this scheduling problem of not having clashes between courses. The problem runs repeatedly, with it starting over from scratch in the cases where all of the courses cannot be allocated without conflict, and only terminates once a valid solution is obtained.

The given problem can be further improved by expanding the system for the entire department, and then the entire college. Further fine-tuning can be done to ensure that free hours occur in the latter half of the day alone, and do not occur in the mornings. A more complex model could even facilitate teachers to choose slots for classes at the most convenient time. In such a scenario, the constraints would have to be imposed after meeting the requirements of the teacher. In all, this solution to the problem satisfies all the hard constraints, and numerous soft constraints. It automates the cumbersome task of manual timetable scheduling quite effectively.

References

1. “An Application of Graph Colouring Model to Course Timetabling Problem” by Wathsala Samasekara, Sri Lanka Institute of Advanced Technological Education, Sri Lanka, published in International Journal of Science and Research(IJSR).
2. “A Graph Edge Colouring Approach for School Timetabling Problems” by Rakesh Prasad Badoni and D.K. Gupta, Indian Institute of Technology Kharagpur, published in International Journal of Mathematics in Operational Research.
3. “A Study of University Timetabling that Blends Graph Colouring with the Satisfaction of Various Essential and Preferential Conditions” by Timothy Anton Redl, Rice University.