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

Problem Definition

The given problem of generating a timetable for an academic institution can be classified as a scheduling problem. The problem essentially requires the process of allocation of a variety of courses in a stipulated time period based on the availability of resources such as teacher, student and lecture halls while satisfying a set of constraints.

A solution to the problem also depends on the constraints that have been imposed on the system, in terms of time constraints and constraints on faculty. The academic institution must be considered as a whole, while formulating the solution.

A need for two levels of accessibility has been identified:

- At the student level- To view the timetable of their section in their department.
- At the faculty level- To add courses, the number of sessions and its venue, to add a lecturer and the course taught and to assign lab slots.

The output would be a timetable corresponding to the inputs entered, for students and faculty.

Module 1: Database

The timetable scheduling problem deals with enormous amounts of data from an academic institution. These data include the courses available for each semester, the number of hours allotted to each course, the number of sections in each year and the faculty members teaching a course. The storage of these various and vast amounts of data calls for the use of a database.

A database is an organised collection of structured information, or data, typically stored electronically in a computer system. A database is usually controlled by a database management system(DBMS). Together, the data and the DBMS, along with the applications that are associated with them, are referred to as a database system, or database.

Databases let us work with large amounts of data efficiently. They make updating data easy and reliable, and they help to ensure accuracy. They offer security features to control access to information, and they help us avoid redundancy. It is also very easy to access, navigate and research data in a database efficiently with the help of Data Query Language(DQL). For this project, we will be using relational databases, which store the data in the form of tables or "relations". The database management system used is Oracle MySQL, where the data will be gathered and ordered. The data will then be exported to Microsoft Excel spreadsheets.
There will also be a connection established between MySQL and the programming language used, i.e. C, by including <mysql.h> as a pre-processor directive along with <stdio.h>.

For the purpose of this project, we have three tables:

1. Courses- This table contains a list of all the courses, which semester each course belongs to, and whether it is a theory, lab or elective course, which is represented by 0 or 1.

CourseID	Name	Semester	Theory	Lab	Elective
UMA2377	Discrete Mathematics	3	1	0	0
UHS2351	Humanities 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
UCS2501	Computer Networks	5	1	0	0
UCS2502	Microprocessors and Interfacing	5	1	0	0
UPE2551	Professional Elective	5	0	0	1
UCS2503	Theory of Computation	5	1	0	0
UCS2504	Artificial Intelligence	5	1	0	0
UCS2505	Logic Programming and Paradigms	5	1	0	0
UCS2511	Networks Lab	5	0	1	0
UCS2512	Microprocessors Lab PC	5	0	1	0
UCS2701	Distributed Systems	7	1	0	0
UCS2702	Mobile Computing	7	1	0	0
UCS2703	Graphics and Multimedia	7	1	0	0
UCS2704	Management and Ethical Practices	7	1	0	0
UCS2705	Data Warehousing and Data Mining	7	1	0	0
UPE2751	Professional Elective	7	0	0	1
UCS2711	Mobile Application Development Lab	7	0	1	0
UCS2712	Graphics and Multimedia Lab	7	0	1	0

2. Faculty- This table contains all the faculty members, the different courses they take, and the number of hours required for each course the faculty is teaching.

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

3. Sections- This table lists the courses for each semester along with the number of sections, i.e 2(A and B).

Semester	Section	Course1	Course2	Course3	Course4	Course5	Course6	Course7	Course8
3	A	UMA2377	UHS2351	UCS2301	UCS2302	UCS2303	UCS2311	UCS2312	UCS2313
3	B	UMA2377	UHS2351	UCS2301	UCS2302	UCS2303	UCS2311	UCS2312	UCS2313
5	A	UCS2501	UCS2502	UPE2551	UCS2503	UCS2504	UCS2505	UCS2511	UCS2512
5	B	UCS2501	UCS2502	UPE2551	UCS2503	UCS2504	UCS2505	UCS2511	UCS2512
7	A	UCS2701	UCS702	UPE2751	UCS2703	UCS2704	UCS2705	UCS2711	UCS2712
7	B	UCS2701	UCS702	UPE2751	UCS2703	UCS2704	UCS2705	UCS2711	UCS2712

Module 2: Graph Colouring

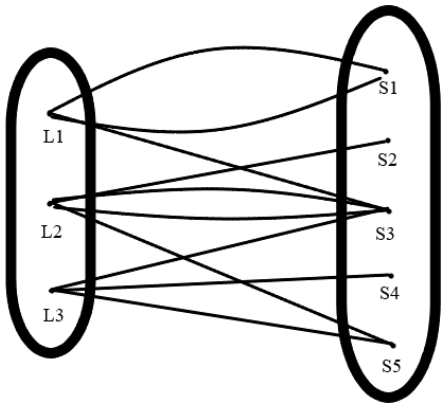
In order to schedule a timetable with no conflicts it is necessary to determine the classes which can occur parallely. *Edge colouring* is the process of assigning colours to the edges of a graph such that no two edges sharing a common vertex have the same colour. The edge colouring idea is being used 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.

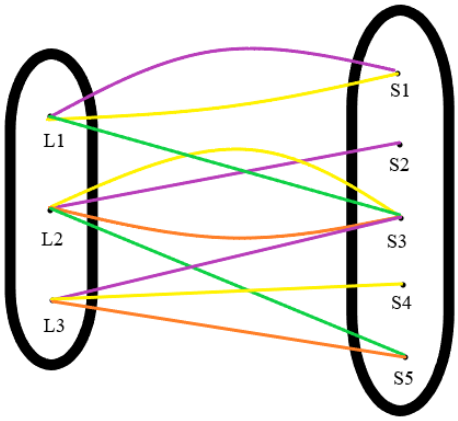
The major components of a bipartite graph are its vertices and edges which represent entities and the relationship amongst them.
The *vertices* are in the form of two sets- (i) set of lecturers and (ii) set of courses.
The *edges* between a lecturer vertex, say L11, and a course vertex, say UCS1501 indicates 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



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.
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.

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
```

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
```

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))
```

Output:

The column under each colour represents the classes which can occur at once without any conflict of Lecturers as well as Courses.

Time slot (color) / Lecturer	1	2	3	4
L1	S1	S1	-	S3
L2	S2	S3	S3	S5
L3	S3	S4	S5	-

```
>>>
============= RESTART: C:\COLLEGE\studio project\edgecolouring#2.py ============
Making a list of courses taught by Lith Lecturer in the (i-1)th position: [['S1','S3'],['S2','S3','S5'],['S3','S4','S5']]
Enter the number of hours taught by each lecturer for a subject in matrix format: [[2,0,1,0,0],[0,1,2,0,1],[0,0,1,1,1]]
[['L2', 'S2', 1, 1], ['L2', 'S3', 1, 2], ['L2', 'S3', 2, 3], ['L2', 'S5', 1, 4], ['L1', 'S1', 1, 1], ['L1', 'S1', 2, 2], ['L1', 'S3', 1, 4], ['L3', 'S3', 1, 1], ['L3', 'S4', 1, 2], ['L3', 'S5', 1, 3]]
>>>
```

Similarly, the same can be done for a larger dataset:

These are the first twenty rows of the larger data set. In all there are 79 records.

Sl No	Course ID	Lecturer	Color ID	Semester	Type
1	UCS1501	L11	1	5	T
2	UCS1701	L15	1	7	T
3	UCS1504	L17	1	5	T
4	UCS1711	L13	1	7	P
5	UCS1703	L19	1	7	T
6	UHS2351	L3	1	3	T
7	UCS2301	L5	1	3	T
8	UCS2302	L7	1	3	T
9	UCS1704	L9	1	7	T
10	UMA2377	L1	2	3	T
11	UCS1524	L11	2	5	T
12	UCS1503	L15	2	5	T
13	UCS1711	L13	2	7	P
14	UCS1729	L21	2	7	T
15	UCS1703	L19	2	7	T
16	UCS2302	L7	2	3	T
17	UCS1704	L9	2	7	T
18	UCS2301	L5	2	3	T
19	UCS1701	L15	3	7	T
20	UCS2303	L9	3	3	T

After performing graph colouring on the larger dataset, it can be identified that the chromatic index of the graph is 9. Classes with the same chromatic index can be scheduled simultaneously.

	1	2	3	4	5	6	7	8	9
L11	UCS1511	UCS1511	UCS1511	UCS1501	UCS1524	UCS1501	UCS1524	UCS1501	UCS1524
L13	UCS1502	UCS1711	UCS1711	UCS1711	UCS1526	UCS1502	UCS1526	UCS1502	UCS1526
L15	UCS1701	UCS1503	UCS1512	UCS1512	UCS1512	UCS1701	UCS1503	UCS1701	UCS1503
L17		UCS1504		UCS1712	UCS1712	UCS1712	UCS1504	UCS1504	
L5	UCS2301	UCS1505	UCS2301	UCS1505	UCS2311	UCS2311	UCS2311	UCS1505	UCS2301
L7	UCS2302	UCS2302	UCS1723	UCS1723	UCS2302	UCS1723	UCS2312	UCS2312	UCS2312
L9	UCS1704	UCS1704	UCS1704	UCS2303	UCS2303	UCS2313	UCS2313	UCS2313	UCS2303
L1	UMA2377	UMA2377	UMA2377						UMA2377
L19	UCS1703		UCS1722		UCS1722	UCS1703		UCS1703	UCS1722
L21		UCS1729		UCS1729	UCS1702		UCS1729	UCS1702	UCS1702
L3			UHS2351				UHS2351	UHS2351	

Module 3: Allocation of Courses

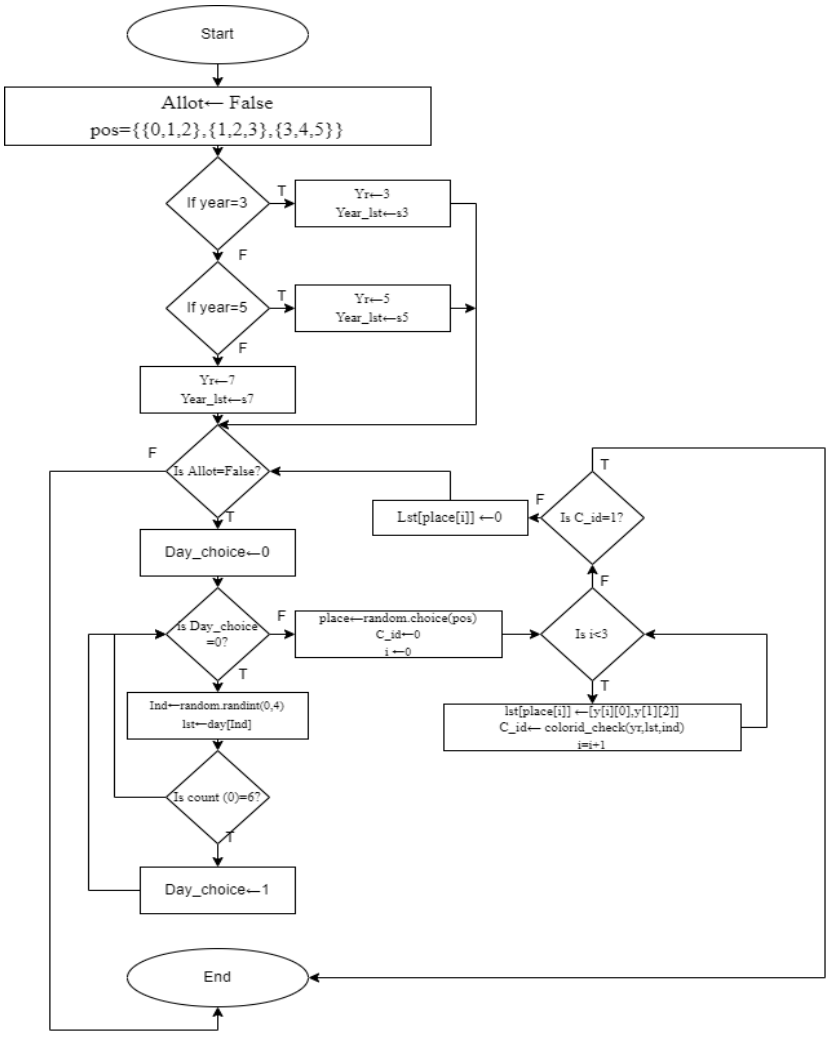
The allotment of courses into slots is done in two parts. The labs are initially allocated, since they require three hours in bulk, and must also satisfy the constraint that each day has only one lab course. In order to achieve this, the dataset consisting of all of the courses, lecturer ids and colour ids is sorted on the basis of the *type of course, lecturer id and then colour id.*

The first twenty rows of the sorted dataset would be 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

Part 1: Choosing days to conduct labs, and their positions

Flowchart



Algorithm

Algorithm-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 lab with colour ids

Algorithm

Algorithm-verifying positioning of lab (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
```

Output

The output of lab courses scheduled across Semester 3, 5 and 7 without there being a conflict of teachers and after satisfying all of the lab constraints is as follows:

```
= RESTART: C:\Users\pooja\Documents\SSN\Semester 2\Fundamentals and Practice of Software Develop
ment\Studio Project- Timetable\colorid_slot.py
Semester 3
[[0, 0, 0, 0, 0, 0], [('UCS2311', '5'), ('UCS2311', '6'), ('UCS2311', '7'), 0, 0, 0], [('UCS2312
', '7'), ('UCS2312', '8'), ('UCS2312', '9'), 0, 0, 0], [('UCS2313', '6'), ('UCS2313', '7'), ('UC
S2313', '8'), 0, 0, 0], [0, 0, 0, 0, 0, 0]]

Semester 5
[[('UCS1512', '3'), ('UCS1512', '4'), ('UCS1512', '5'), 0, 0, 0], [0, 0, 0, 0, 0, 0], [0, 0, 0,
0, 0, 0], [0, 0, 0, ('UCS1511', '1'), ('UCS1511', '2'), ('UCS1511', '3')], [0, 0, 0, 0, 0, 0]]

Semester 7
[[0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0], [0, 0, 0, ('UCS1711', '2'), ('UCS1711', '3'), ('UCS1711
', '4')], [0, 0, 0, 0, 0, 0], [0, ('UCS1712', '4'), ('UCS1712', '5'), ('UCS1712', '6'), 0, 0]]
```

Part 3: Scheduling of Theory Courses

Algorithm

Input: A set of theory courses, along with their color 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 allocation is valid
                if c_id=1:
                   return
                else:
                   Day[k]  ←0
//Removing value if course is improperly placed and then repeating the process
          End for
```

End for
End while

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.