|  |  |
| --- | --- |
| **Assignment** | |
| **Course Code** | CSC301A |
| **Course Name** | Discrete Mathematics-II |
| **Programme** | B.Tech |
| **Department** | CSE |
| **Faculty** | FET |

|  |  |
| --- | --- |
| **Name of the Student** | Satyajit Ghana |
| **Reg. No.** | 17ETCS002159 |
| **Semester/Year** | 5/2017 |
| **Course Leader(s)** | Ms. Pallavi R. Kumar |



|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Declaration Sheet | | | | | | | | |
| Student Name | Satyajit Ghana | | | | | | | |
| Reg. No | 17ETCS002159 | | | | | | | |
| Programme | B.Tech | | | | | Semester/Year | 5/2017 | |
| Course Code | CSC301A | | | | | | | |
| Course Title | Discrete Mathematics-II | | | | | | | |
| Course Date |  | | To |  | | | | |
| Course Leader | Ms. Pallavi R. Kumar | | | | | | | |
| **Declaration**  The assignment submitted herewith is a result of my own investigations and that I have conformed to the guidelines against plagiarism as laid out in the Student Handbook. All sections of the text and results, which have been obtained from other sources, are fully referenced. I understand that cheating and plagiarism constitute a breach of University regulations and will be dealt with accordingly. | | | | | | | | |
| Signature of the Student | |  | | | | | Date |  |
| Submission date stamp  (by Examination & Assessment Section) | |  | | | | | | |
| Signature of the Course Leader and date | | | | | Signature of the Reviewer and date | | | |
|  | | | | |  | | | |

# Contents

[Declaration Sheet ii](#_Toc24298947)

[Contents iii](#_Toc24298948)

[List Of Figures iv](#_Toc24298949)

[1 Question 1 5](#_Toc24298950)

[1.1 Specification of constrains for the scheduling problem 5](#_Toc24298951)

[1.2 Assumptions made, if any, along with justification 5](#_Toc24298952)

[1.3 The graph model used and the colouring algorithm for it 6](#_Toc24298953)

[1.4 The generation of a timetable based on the colouring carried out 10](#_Toc24298954)

[2 Question 2 11](#_Toc24298955)

[2.1 Specification of constraints for the external examiner assignment problem 11](#_Toc24298956)

[2.2 Assumptions made, if any, along with justification 11](#_Toc24298957)

[2.3 Formulation of the Linear Programming Problem (LPP) 12](#_Toc24298958)

[2.4 Method used to solve the LPP and the reason to choose the method 12](#_Toc24298959)

[Appendix A 17](#_Toc24298960)

[Bibliography 20](#_Toc24298961)

# List Of Figures

[Figure 1‑1 Lab1 dependency graph 7](#_Toc24298854)

[Figure 1‑3 Lab1 coloured graph 8](#_Toc24298855)

[Figure 1‑4 Lab2 dependency graph 9](#_Toc24298856)

[Figure 1‑5 Lab2 Coloured Graph 9](#_Toc24298857)

# Question 1

Solution to Question No. 1 Part A

## Specification of constrains for the scheduling problem

Given,

There are 200 students that need to take up Lab Exams for CN, CS and OS. There are 2 Labs available each day for the exam, 103C and 103D, and each lab can take upto 3 slots per day. We need to create a time table for the examination of each student in minimum number of days.

Hard Constraints:

* The constraints being, each student has to take up all the lab exams, and at a given time slot a student cannot give more than 1 exam simultaneously.

Soft Constraints:

* At a given time slot, only one subject examination can take place, for example at Slot #1 for both the Labs, only CN will be conducted. This is to reduce the number of question papers that the teachers have to prepare. In the answer the number of days of exams will be the number of question papers that the teacher of each subject has to prepare.

## Assumptions made, if any, along with justification

Some assumptions that we are going to make are, we are not constrained to conduct only one subject in a lab in a given day, i.e for example Lab 103C, we can conduct CN for slot #1, CS for slot #2, and OS for slot #3. This has been done to simplify the graph and reduce the number of constrained, it would be possible to solve the graph manually now.

Another simplification done is, the batch is assigned to a lab room, all the subject exams for that batch has to be done in that room, this is to reduce the room changes that the student has to do for the exam, the seating arrangement hence can be constant, making it easier for the examination department, for example B1, B3, B5 and B7 will give exam in Lab 103C, so the computers assigned to each student is same throughout the subjects lab exams.

## The graph model used and the colouring algorithm for it

We are to allocate 200 students in batches of 30 each, since the capacity of each lab is 30, hence we will have 7 batches of capacities [30 30 30 30 30 30 20] named [B1, B2, B3, B4, B5, B6, B7]. B1-B6 have 30 students each, and B7 has 20 students.

Now we have 2 Labs, based on our soft constrains, we’ll have to conduct CN, OS and CS exams in each of the labs, we’ll call the exams in Lab1 as [CN1, OS1, CS1] and in Lab2 as [CN2, OS2, CS2].

The 7 Batches are round-robin divided into the two labs, since our soft constraints says that we have to conduct all the exams for one batch in the same lab. We’ll assign [B1, B3, B5, B7] to Lab1 and [B2, B4, B6] to Lab2.

The following dependency table is generated for the subjects and the batches.

Table Lab1 Dependency Table

|  |  |
| --- | --- |
| Lab1-Subject | Batches |
| CN1 | B1, B3, B5, B7 |
| OS1 | B1, B3, B5, B7 |
| CS1 | B1, B3, B5, B7 |

Table Lab2 Dependency Table

|  |  |
| --- | --- |
| Lab2-Subject | Batches |
| CN2 | B2, B4, B6 |
| OS2 | B2, B4, B6 |
| CS2 | B2, B4, B6 |

For Table1 Dependency Table we generate the Undirected Graph,

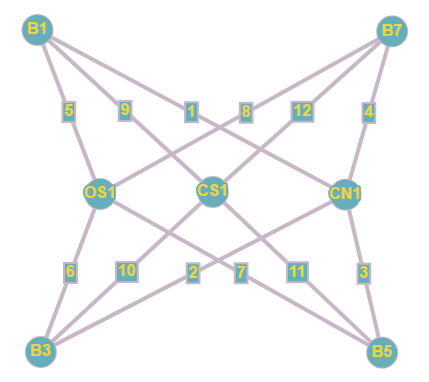


Figure ‑ Lab1 dependency graph

For the above dependency graph, the allocation can be done by edge colouring, but we’ll convert the edge colouring problem into vertex coloring problem, by assigning number to edges and creating an adjacency edges as adjacency vertex graph as below.

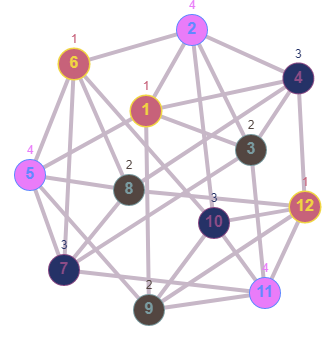


Figure ‑ Lab1 coloured graph

Similarly, we generate the dependency graph from the table and convert the edge colouring problem into vertex colouring problem.

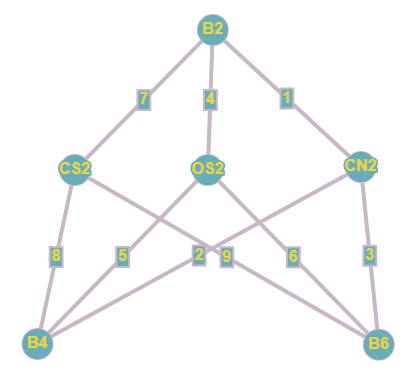


Figure ‑ Lab2 dependency graph

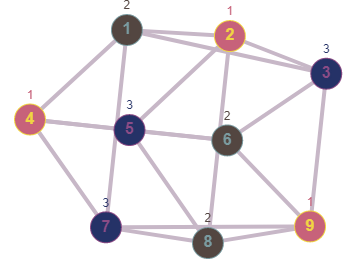


Figure ‑ Lab2 Coloured Graph

The vertex colouring problem was solved using Backtracking algorithm as described below.

If all colors are assigned,

print vertex assigned colors

Else

a. Trying all possible colors, assign a color to the vertex

b. If color assignment is possible, recursivelty assign colors to next vertices

c. If color assignment is not possible, de-assign color, return False

The source code for the C++ program and the output for the graph colourings is attached in Appendix A.

## The generation of a timetable based on the colouring carried out

Based on the colours assigned to the edges, we go to that edge number in the dependency graph and see the vertex associated with that edge and hence generate the time table.

Table Allocation for Lab1 - 103C

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Day 1 | Day 2 | Day 3 | Day 4 |
| Slot #1 – OS | B3 | B1 | B7 | B5 |
| Slot #2 – CN | B5 | B7 | B1 | B3 |
| Slot #3 – CS | B7 | B5 | B3 | B1 |

Table Allocation for Lab2 - 103D

|  |  |  |  |
| --- | --- | --- | --- |
|  | Day 1 | Day 2 | Day 3 |
| Slot #1 – OS | B2 | B6 | B4 |
| Slot #2 – CN | B4 | B2 | B6 |
| Slot #3 – CS | B6 | B4 | B2 |

# Question 2

Solution to Question 1 Part B

## Specification of constraints for the external examiner assignment problem

The external examiner assignment problem is to minimize the Total Enumeration paid to the examiners. The question is a continuation of the previous scheduling problem so the constraints needs to be formulated accordingly.

The remuneration table is as follow:

|  |  |  |
| --- | --- | --- |
| Teacher | Availability | Renumeration per Day |
| A | 3 Days | Rs. 2000 |
| B | 1 Day | Rs. 3400 |
| C | 4 Days | Rs. 3000 |
| D | 2 Days | Rs. 2500 |

* No examiner can be invited for more than three days.
* The examiners are paid per day.
* An examiner cannot examine for two classes simultaneously.

## Assumptions made, if any, along with justification

Hard Constraints:

From Section 1, we have 2 labs, 4 days of examination in Lab1 and 3 days of examination in Lab2. At a time, an examiner can only examine for one Lab, so we need another examiner for the other lab, so it can be thought of Slots, or 7 “Examiner Days”, that’s what we will call it. 4 Days of exams in our time table needs 7 Examiner Days, since there are parallel exams going on.

Soft Constraints:

The same examiner can examine for OS, CN and CS, this is to make the work more simpler, also this constrain should have be presumed to be true as well.

## Formulation of the Linear Programming Problem (LPP)

Let the number of days Examiner A comes =

Let the number of days Examiner B comes =

Let the number of days Examiner C comes =

Let the number of days Examiner D comes =

From the cost table in 2.1 we need to minimize the total cost, i.e.

No Examiner can come more than 3 days, so

From the assumption made in 2.2 we need the examiners for at least 7 days,

The LPP Problem can thus be presented as,

MIN

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

## Method used to solve the LPP and the reason to choose the method

Simplex Minimization is used to solve the problem, because it is simple and can be programmed easily. Dual wasn’t taken here because the number of slack variables is not reduced by converting, so there wasn’t any significant benefit to convert and solve.

The problem is converted to canonical form by adding slack variables, surplus and artificial variables as appropriate.

1. As the constraint-1 is of type we should add slack variable

2. As the constraint-2 is of type we should add slack variable

3. As the constraint-3 is of type we should add slack variable

4. As the constraint-4 is of type we should add slack variable

5. As the constraint-5 is of type we should subtract surplus variable and add artificial variable

After introducing these, the equations becomes,

subject to

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Iter 1: |  |  | 2 | 3.4 | 3 | 2.5 | 0 | 0 | 0 | 0 | 0 | M |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | 0 | 3 | (1) | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |  |
|  | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | -- |
|  | 0 | 3 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | -- |
|  | 0 | 2 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | -- |
|  | M | 7 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | -1 | 1 |  |
|  |  |  |  |  |  |  | 0 | 0 | 0 | 0 | -M | M |  |
|  |  |  | M-2 | M-3.4 | M-3 | M-2.5 | 0 | 0 | 0 | 0 | -M | 0 |  |

Positive Maximum is and its column index is 1. So, the entering variable is .

Minimum ration is 3 and its row index is 1. So the leaving basis variable is

Pivot element is 1

Now,

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Iter 2: |  |  | 2 | 3.4 | 3 | 2.5 | 0 | 0 | 0 | 0 | 0 | M |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | 2 | 3 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | -- |
|  | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | -- |
|  | 0 | 3 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | -- |
|  | 0 | 2 | 0 | 0 | 0 | (1) | 0 | 0 | 0 | 1 | 0 | 0 |  |
|  | M | 4 | 0 | 1 | 1 | 1 | -1 | 0 | 0 | 0 | -1 | 1 |  |
|  |  |  | 2 |  |  |  | -M+2 | 0 | 0 | 0 | -M | M |  |
|  |  |  | 0 | M-3.4 | M-3 | M-2.5 | -M+2 | 0 | 0 | 0 | -M | 0 |  |

Positive Maximum is and its column index is 4. So, the entering variable is .

Minimum ration is 2 and its row index is 4. So, the leaving basis variable is

The pivot element is 1

Now,

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Iter 3: |  |  | 2 | 3.4 | 3 | 2.5 | 0 | 0 | 0 | 0 | 0 | M |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | 2 | 3 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | -- |
|  | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | -- |
|  | 0 | 3 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |  |
|  | 2.5 | 2 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | -- |
|  | M | 2 | 0 | 1 | (1) | 0 | -1 | 0 | 0 | -1 | -1 | 1 |  |
|  |  |  | 2 |  |  | 2.5 | -M+2 | 0 | 0 | -M+2.5 | -M | M |  |
|  |  |  | 0 | M-3.4 | M-3 | 0 | -M+2 | 0 | 0 | -M+2.5 | -M | 0 |  |

Positive Maximum is and its column index is 3. So, the entering variable is .

Minimum ration is 2 and its row index is 5. So, the leaving basis variable is

The pivot element is 1

Now,

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Iter 4: |  |  | 2 | 3.4 | 3 | 2.5 | 0 | 0 | 0 | 0 | 0 |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | 2 | 3 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |  |
|  | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |  |
|  | 0 | 1 | 0 | -1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 |  |
|  | 2.5 | 2 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |  |
|  | 3 | 2 | 0 | 1 | 1 | 0 | -1 | 0 | 0 | -1 | -1 |  |
|  |  |  | 2 | 3 | 3 | 2.5 | -1 | 0 | 0 | -0.5 | -3 |  |
|  |  |  | 0 | -0.4 | 0 | 0 | -1 | 0 | 0 | -0.5 | -3 |  |

Since all

Hence the optimal solution is arrived with the value of variables as:

MIN

The Teacher allocation will be as follows

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Day 1 | Day 2 | Day 3 | Day 4 |
| Slot #1 – OS | A | A | A | D |
| Slot #2 – CN | A | A | A | D |
| Slot #3 – CS | A | A | A | D |

|  |  |  |  |
| --- | --- | --- | --- |
|  | Day 1 | Day 2 | Day 3 |
| Slot #1 – OS | C | C | D |
| Slot #2 – CN | C | C | D |
| Slot #3 – CS | C | C | D |

The total cost being Rs. 17,000

# Appendix A

#include <stdbool.h>

#include <stdio.h>

*// Number of vertices in the graph*

#define V 12

*void* print\_colors(*int* **color**[]);

*bool* is\_safe(*int* **v**, *bool* **graph**[V][V], *int* **color**[], *int* **c**) {

**for** (*int* i **=** 0; i **<** V; i**++**)

**if** (graph[v][i] **&&** c **==** color[i])

**return** false;

**return** true;

}

*bool* graph\_coloring\_recurse(*bool* **graph**[V][V], *int* **m**, *int* **color**[], *int* **v**) {

**if** (v **==** V)

**return** true;

*/\* Consider this vertex v and try different colors \*/*

**for** (*int* c **=** 1; c **<=** m; c**++**) {

*/\* Check if assignment of color c to v is fine\*/*

**if** (is\_safe(v, graph, color, c)) {

            color[v] **=** c;

*/\* recur to assign colors to rest of the vertices \*/*

**if** (graph\_coloring\_recurse(graph, m, color, v **+** 1) **==** true)

**return** true;

*/\* If assigning color c doesn't lead to a solution*

*then remove it \*/*

            color[v] **=** 0;

        }

    }

*/\* If no color can be assigned to this vertex then return false \*/*

**return** false;

}

*bool* graph\_coloring(*bool* **graph**[V][V], *int* **m**) {

*// Initialize all color values as 0. This initialization is needed*

*// correct functioning of is\_safe()*

*int* color[V];

**for** (*int* i **=** 0; i **<** V; i**++**)

        color[i] **=** 0;

*// Call graph\_coloring\_recurse() for vertex 0*

**if** (graph\_coloring\_recurse(graph, m, color, 0) **==** false) {

        printf("Solution does not exist");

**return** false;

    }

*// Print the solution*

    print\_colors(color);

**return** true;

}

*void* print\_colors(*int* **color**[]) {

    printf(

        "Solution Exists:"

        " Following are the assigned colors \n");

**for** (*int* i **=** 0; i **<** V; i**++**)

        printf(" *%d* ", color[i]);

    printf("\n");

}

*// driver program to test above function*

*int* main() {

*bool* graph[V][V] **=** {0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0,

                        1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0,

                        1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0,

                        1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1,

                        1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0,

                        0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0,

                        0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0,

                        0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1,

                        1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1,

                        0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1,

                        0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1,

                        0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0};

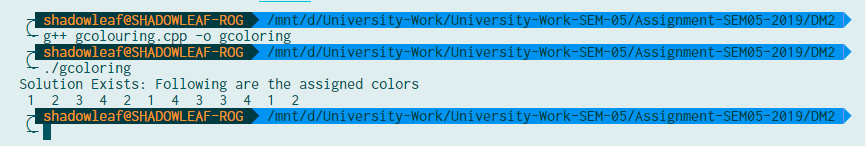
*int* m **=** 4;*// Number of colors*

    graph\_coloring(graph, m);

**return** 0;

}

OUTPUT for Graph 1



As we can see 4 colours were used

Source Code changes for Graph 2

*bool* graph[V][V] **=** {0, 1, 1, 1, 0, 0, 1, 0, 0,

                        1, 0, 1, 0, 1, 0, 0, 1, 0,

                        1, 1, 0, 0, 0, 1, 0, 0, 1,

                        1, 0, 0, 0, 1, 1, 1, 0, 0,

                        0, 1, 0, 1, 0, 1, 0, 1, 0,

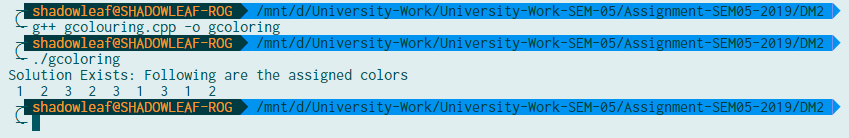
                        0, 0, 1, 1, 1, 0, 0, 0, 1,

                        1, 0, 0, 1, 0, 0, 0, 1, 1,

                        0, 1, 0, 0, 1, 0, 1, 0, 1,

                        0, 0, 1, 0, 0, 1, 1, 1, 0,};

Output for Graph 2



As we can see only 3 colours were used

# Bibliography

1. <https://www.geeksforgeeks.org/m-coloring-problem-backtracking-5/>
2. <http://stephanie-w.github.io/brainscribble/m-coloring-problem.html>