

ALLOCATING SUITABLE TIME SLOTS FOR A REPLACEMENT CLASS



TEAM MEMBERS

Muhammad Najwan Bin Nizarimi

209897@student.upm.edu.my

Bachelor of Software Engineering

Tuan Nurhusmiza Fazehah Binti Tuan Mat Zawai

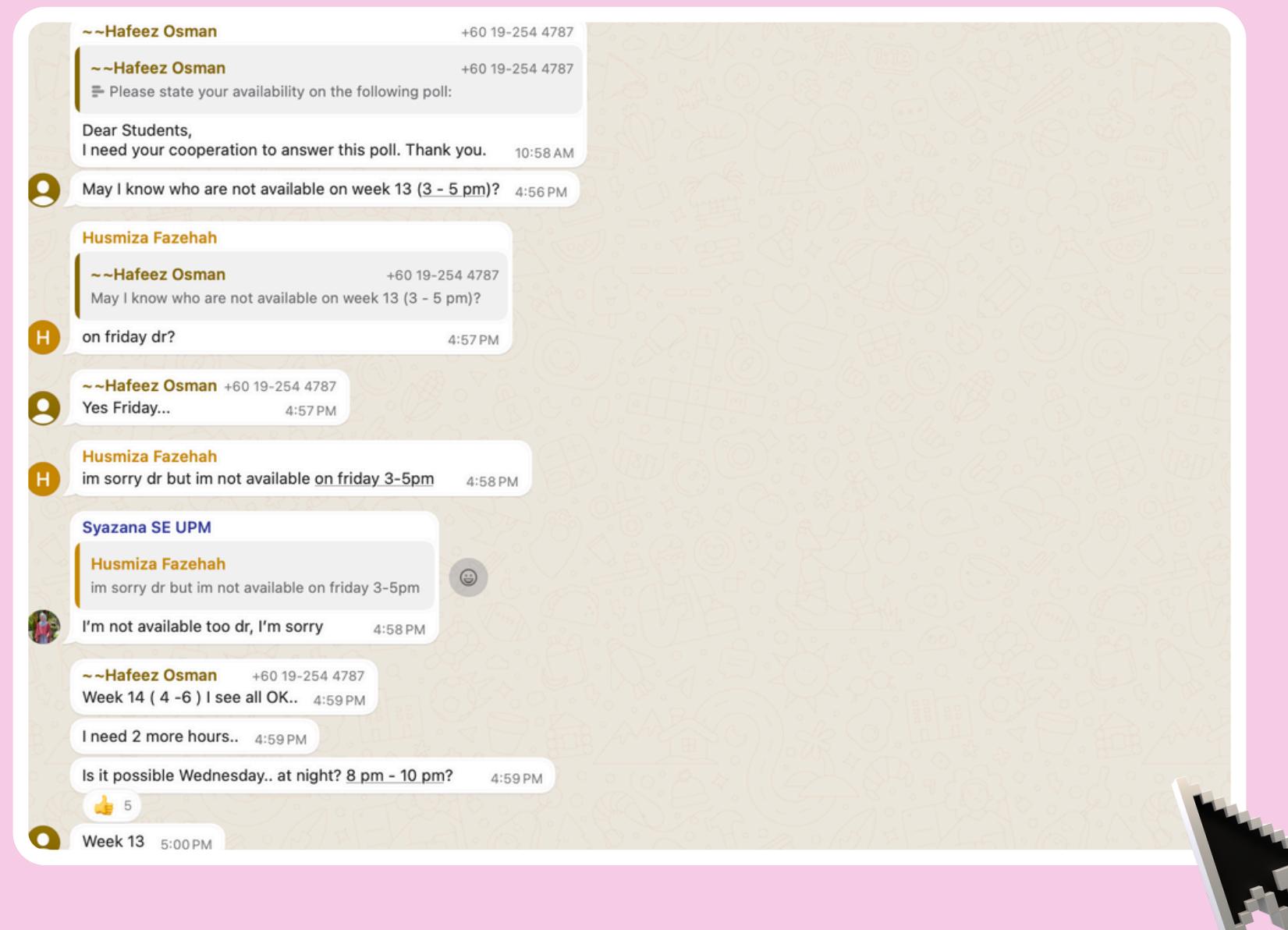
209861@student.upm.edu.my

Bachelor of Software Engineering

SCENARIO

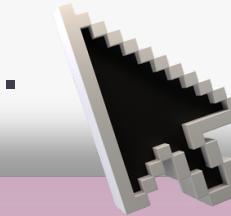


The illustration below represents students from the Secure Software Development (SWE4356) class at University Putra Malaysia (UPM), where the lecturer struggles to find replacement classes that accommodate all the students



PROBLEM DEFINITION

Dr. Hafeez is a dedicated lecturer in the Department of Software Engineering at Universiti Putra Malaysia (UPM), located in Serdang, Malaysia. Throughout the semester, his busy schedule and multiple public holidays occurring on class days have resulted in multiple postponed lectures. This has led to inconsistent student attendance and the risk of important topics being left uncovered before the final exams. To address these challenges, Dr. Hafeez seeks a solution to identify the most optimal time slots for replacement classes by analysing both his and his students' schedules. The proposed algorithm promises efficiency by quickly pinpointing the best time slots, accuracy by ensuring maximum possible attendance, and convenience by reducing the manual effort in coordination. The goal is to ensure that all missed lectures are rescheduled effectively, maximising student attendance and ensuring complete syllabus coverage. The expected output is a list of optimal time slots for replacement classes without conflicts. This solution aims to make sure all important lectures are completed and students are well-prepared for their final exams.



IMPORTANCES

- **Maintaining Academic Standards**

Each subject has designated credit hours and content that must be completed to meet educational standards.

- **Avoiding Scheduling Conflicts**

Replacement classes might be scheduled at inconvenient times which lead to conflicts with other classes or personal commitments.

- **Ensuring Fairness**

Ensure all students receive the same amount of learning time, as it can be unfair if certain students miss out on certain topics due to time constraints.

- **Minimising Disruption**

Properly scheduled replacement classes can help minimise additional disruptions to the academic calendar and ensure a smooth learning experience.



DEVELOPMENT OF THE MODEL

- **Data used:** Empty or available time slots from the lecture's and students' schedules for SWE3456 class
- **Objective:** To cover missed lecture hours during the holiday



CONSTRAINTS AND REQUIREMENTS



- Only on weekdays from 8 a.m to 6 p.m.
- Can be held online or physically
- Must cover for 2 hours
- Must not overlap with other scheduled classes
- A person is considered free when they do not have any classes during that time

DATA TYPE

Lecturer



Students



ACADEMIC DIVISION UNIVERSITI PUTRA MALAYSIA															
TIME TABLE SECOND SEMESTER 2023/2024															
NAME	TUAN NURHUSMIZA FAZEHAH BINTI TUAN MAT ZAWAI														
MATRIC NO.	208681														
FACTORIALITY	DEPARTMENT OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY														
PROGRAMME	480-320 BACHELOR OF SOFTWARE ENGINEERING WITH HONOURS														
MAJOR	NO MAJOR														
MINOR	NO MINOR														
TIME / DAY	8-9	9-10	10-11	11-12	12-13	13-14	14-15	15-16	16-17	17-18	18-19	19-20	20-21	21-22	22-23
MONDAY	SWE4355 (GROUP 1) BK2FSKTM (LECTURE) (LAB)	SWE4355 (GROUP 1) BK2FSKTM (LECTURE) (LAB)	SSE4304 (GROUP 1) MAK KP1 FSKTM (LAB)	SSE4304 (GROUP 1) MAK KP1 FSKTM (LAB)	SWE4357 (GROUP 1) BK2FSKTM (LECTURE)	SWE4356 (GROUP 1) JOGO KP1 (LECTURE)									
TUESDAY	SSE4340 (GROUP 1) MAK KP1 FSKTM (LAB)	SSE4340 (GROUP 1) MAK KP1 FSKTM (LAB)	SSE4340 (GROUP 1) MAK KP1 FSKTM (LAB)	SSE4340 (GROUP 1) MAK KP1 FSKTM (LAB)	SWE4356 (GROUP 1) JOGO KP1 (LECTURE)										
WEDNESDAY	CEB2105 (GROUP 6) DKS (LECTURE)	CEB2105 (GROUP 6) DKS (LECTURE)	CEB2105 (GROUP 6) DKS (LECTURE)	CEB2105 (GROUP 6) DKS (LECTURE)	SWE4355 (GROUP 1) BK2FSKTM (LECTURE)										
THURSDAY	CSC2020 (GROUP 6) BK1FSKTM (LECTURE)	SSE4340 (GROUP 1) MAK KP1 FSKTM (LAB)	SSE4340 (GROUP 1) MAK KP1 FSKTM (LAB)	SSE4340 (GROUP 1) MAK KP1 FSKTM (LAB)	SSE4340 (GROUP 1) MAK KP1 FSKTM (LAB)	SSE4340 (GROUP 1) MAK KP1 FSKTM (LAB)	SSE4340 (GROUP 1) MAK KP1 FSKTM (LAB)	SSE4340 (GROUP 1) MAK KP1 FSKTM (LAB)	SSE4340 (GROUP 1) MAK KP1 FSKTM (LAB)	SSE4340 (GROUP 1) MAK KP1 FSKTM (LAB)	SSE4340 (GROUP 1) MAK KP1 FSKTM (LAB)	SSE4340 (GROUP 1) MAK KP1 FSKTM (LAB)	SSE4340 (GROUP 1) MAK KP1 FSKTM (LAB)	SSE4340 (GROUP 1) MAK KP1 FSKTM (LAB)	
FRIDAY															
SATURDAY															
SUNDAY															



ALGORITHMS REVIEW

Sorting



S: Straightforward way to organise schedules by date or availability

W: Doesn't address the complexity of schedules conflicts optimization for maximum attendances

DP



S: Can be used to find optimal time slots for replacement classes by recursively considering sub-problems

W: DP can require significant memory especially with large inputs.

Greedy



S: Easy to implement as it often requires less code due to its straightforward character

W: Do not always produce optimal results especially when future decisions are dependent on earlier decisions

DAC



S: Efficient as breaking down a problem into sub-problems, solve it, and combine it back together can speed up the computation

W: Complicated to implement for problems that highly dependant to other problems

Graph



S: Can represent the relationship between time slots, conflicts and dependencies to identify the time slots for replacement classes

W: Can be used under the assumption of static data and reflects that does not reflect real-word dynamic changes

ALGORITHM SPECIFICATION

Aspect	Greedy	Dynamic Programming
Approach	Makes locally optimal choices at each step without considering future consequences.	Solves problems by breaking them down into smaller overlapping subproblems.
Decision Making	Makes decisions based on immediate gains or local optimal solutions	Evaluates all possible decisions systematically to find the optimal solution
Backtracking	Does not involve backtracking	Does involve backtracking
Complexity	Lower time and space complexity	Higher time and space complexity
Application	Suitable for quick scheduling with fewer constraints	Suitable if multiple constraints and factors need thorough analysis
Examples	Minimum Spanning Tree, Shortest Path algorithms.	Fibonacci sequence, Longest Common Subsequence.

Therefore, Dynamic Programming algorithm is better suited as it provides multiple choices for the students and lecturers, resulting in an optimal allocation of time for replacement classes.

PSEUDOCODE

```
START  
DEFINE WORKING_HOURS_START AS 8  
DEFINE WORKING_HOURS_END AS 18  
DEFINE REQUIRED_FREE_HOURS AS 2  
  
CLASS DPAlgorithm  
  
FUNCTION findContinuousFreeSlot(lecturerAvailability,  
                                studentAvailability)  
    INITIALIZE dp AS ARRAY OF BOOLEAN WITH LENGTH  
(WORKING_HOURS_END - WORKING_HOURS_START + 1)  
  
    FOR hour FROM WORKING_HOURS_START TO  
        WORKING_HOURS_END  
        IF isSlotAvailable(hour, hour + REQUIRED_FREE_HOURS,  
                           lecturerAvailability)  
            AND isSlotAvailable(hour, hour +  
                REQUIRED_FREE_HOURS, studentAvailability)  
            SET dp[hour - WORKING_HOURS_START] TO TRUE  
        ELSE  
            SET dp[hour - WORKING_HOURS_START] TO FALSE  
        END IF  
    END FOR  
  
    printContinuousFreeSlots(dp)  
END FUNCTION  
  
FUNCTION isSlotAvailable(startHour, endHour, availability)  
    FOR hour FROM startHour TO endHour - 1  
        IF availability[hour] IS FALSE  
            RETURN FALSE  
        END IF  
    END FOR  
    RETURN TRUE  
END FUNCTION
```

```
FUNCTION printContinuousFreeSlots(dp)  
    PRINT "Free 2-hour slots:"  
    FOR hour FROM 0 TO LENGTH(dp) - REQUIRED_FREE_HOURS  
        IF dp[hour] IS TRUE AND dp[hour + 1] IS TRUE  
        PRINT "Free slot from", WORKING_HOURS_START + hour, "to",  
              WORKING_HOURS_START + hour + REQUIRED_FREE_HOURS  
        END IF  
    END FOR  
END FUNCTION  
  
END CLASS  
  
FUNCTION main()  
    INITIALIZE lecturerAvailability AS ARRAY OF BOOLEAN WITH  
        LENGTH 24  
    INITIALIZE studentAvailability AS ARRAY OF BOOLEAN WITH  
        LENGTH 24  
  
    FOR hour FROM 0 TO 23  
        SET lecturerAvailability[hour] TO TRUE  
        SET studentAvailability[hour] TO TRUE  
    END FOR  
  
    SET lecturerAvailability[10] TO FALSE  
    SET studentAvailability[15] TO FALSE  
  
    CREATE instance OF DPAlgorithm  
    CALL instance.findContinuousFreeSlot(lecturerAvailability,  
                                         studentAvailability)  
END FUNCTION  
  
END
```

ALGORITHM ANALYSIS

ALGORITHM CORRECTNESS:

Analyzed through asymptotic complexity
and recurrence relations.

Asymptotic complexity examines
runtime behavior as input size grows.



ALGORITHM ANALYSIS

TIME COMPLEXITY



Best-Case:

- $O(n \times m)$
 - where n is the number of students and m is the length of the working hours
- Occurs when the first checked slot from 8 a.m. to 10 a.m. is free for both the lecturer and all student.

ALGORITHM ANALYSIS

TIME COMPLEXITY



Average-Case:

- $O(n \times m \times p)$
 - where n is the number of students and m is the length of the working hours, and p is the average length of the availability
- Occurs when the function will need to check multiple slots before finding a free one.

ALGORITHM ANALYSIS

TIME COMPLEXITY



Worst-Case:

- $O(n \times m \times p)$
 - where n is the number of students and m is the length of the working hours, and p is the average length of the availability
- Occurs when there are no frees slots available requiring the function to check all possible slots for each student.

ALGORITHM ANALYSIS

RECURRENCE RELATIONS

- Specifies how solutions to larger subproblems depend on smaller subproblems.
- Defines solution to a larger problem based on its previous values.
- Relation: ' $dp[i] = \text{lecturerFree} \&\& \text{studentsFree}$ '.
- $dp[i]$: whether a 2-hour free slot starting at hour ' i ' is feasible.
- Builds on smaller subproblems to solve the entire working period.



ALGORITHM ANALYSIS

IMPLEMENTATION

- Base case: Initialize ‘boolean[] dp = new boolean[WORKING_HOURS_END]’.
 - Serves as the dynamic programming table.
- Inductive step: Solve smaller subproblems to larger ones.
 - In ‘findContinuousFreeSlot’ method:
 - Loop iterates over each hour.
 - Checks if both lecturer and students are available for a 2-hour slot starting at hour ‘i’.
 - Sets ‘dp[i]’ to true if conditions are met, otherwise false.
 - After populating ‘dp’ array:
 - Another loop collects all hours where ‘dp[i]’ is true into ‘freeSlots’ list.
 - Represents valid 2-hour continuous time slots.

IMPLEMENTATION

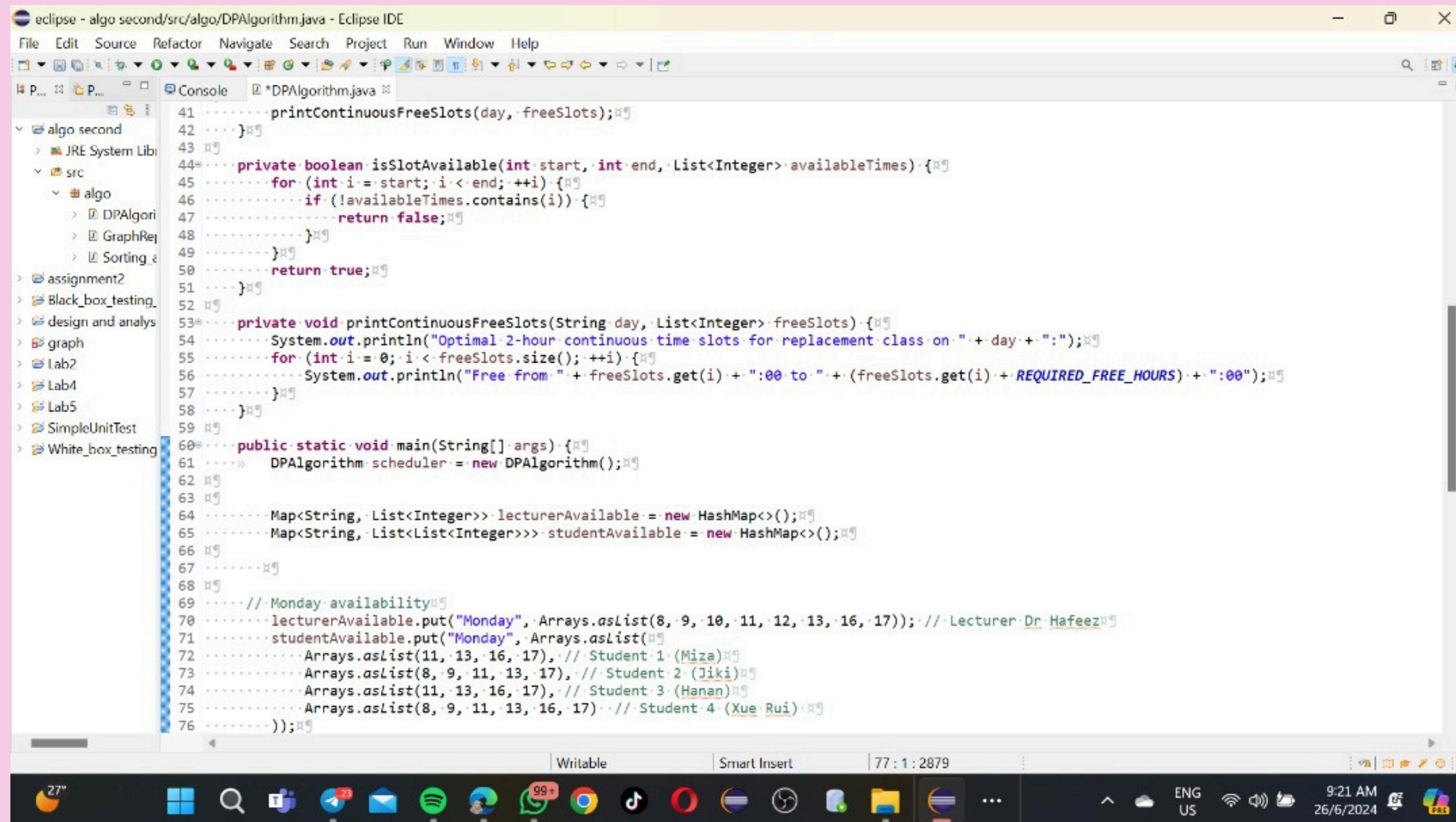
The screenshot shows the Eclipse IDE interface with the following details:

- Title Bar:** eclipse - algo second/src/algo/DPAlgorithm.java - Eclipse IDE
- File Menu:** File Edit Source Refactor Navigate Search Project Run Window Help
- Project Explorer:** Shows the project structure with packages like algo second, SRC, and various sub-directories.
- Console Tab:** Shows the Java code for the DPAlgorithm class.
- Code:** The code implements a dynamic programming solution to find a continuous free slot. It defines working hours from 8am to 6pm and requires 2 hours of free time. The code initializes a boolean array 'dp' and fills it based on lecturer and student availability. Finally, it finds and prints the first available slot.

```
1 package algo;
2
3 import java.util.ArrayList;
4
5 public class DPAlgorithm {
6
7     private static final int WORKING_HOURS_START = 8; // 8am
8     private static final int WORKING_HOURS_END = 18; // 6pm
9     private static final int REQUIRED_FREE_HOURS = 2; // 2 hours time slots
10
11     public DPAlgorithm() {
12
13     }
14
15     public void findContinuousFreeSlot(String day, List<Integer> lecturerAvailable, List<List<Integer>> studentAvailable) {
16
17         boolean[] dp = new boolean[WORKING_HOURS_END];
18
19         for (int i = WORKING_HOURS_START; i <= WORKING_HOURS_END - REQUIRED_FREE_HOURS; ++i) {
20             boolean lecturerFree = isSlotAvailable(i, i + REQUIRED_FREE_HOURS, lecturerAvailable);
21             boolean studentsFree = true;
22             for (List<Integer> studentAvailability : studentAvailable) {
23                 if (!isSlotAvailable(i, i + REQUIRED_FREE_HOURS, studentAvailability)) {
24                     studentsFree = false;
25                     break;
26                 }
27             }
28         }
29         dp[i] = lecturerFree && studentsFree;
30
31     }
32
33     List<Integer> freeSlots = new ArrayList<>();
34     for (int i = WORKING_HOURS_START; i <= WORKING_HOURS_END - REQUIRED_FREE_HOURS; ++i) {
35         if (dp[i]) {
36             freeSlots.add(i);
37         }
38     }
39
40 }
```

- Annotations:** Three callout boxes with arrows point to specific parts of the code:
 - Initialisation of 'dp' array:** Points to the line `boolean[] dp = new boolean[WORKING_HOURS_END];`.
 - Filling the 'dp' array:** Points to the loop where the array is filled based on lecturer and student availability.
 - Utilisation of results:** Points to the final part of the code where the results are printed.

IMPLEMENTATION

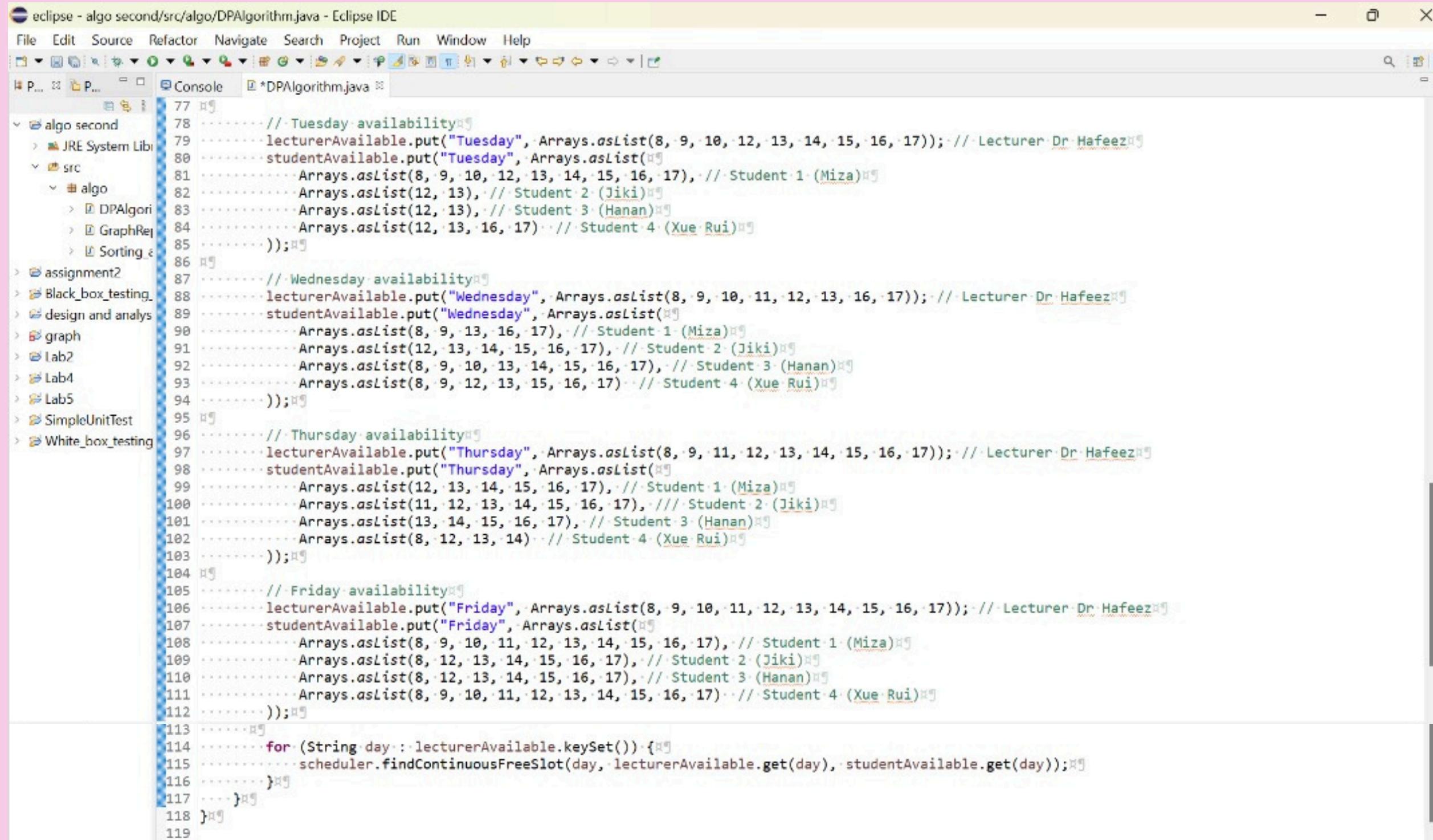


The screenshot shows the Eclipse IDE interface with the title bar "eclipse - algo second/src/algo/DPAAlgorithm.java - Eclipse IDE". The menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help. The toolbar has various icons for file operations. The left sidebar shows the project structure under "algo second": JRE System Library, src (containing algo, assignment2, Black_box_testing, design and analysis, graph, Lab2, Lab4, Lab5, SimpleUnitTest, White_box_testing). The main editor window displays the Java code for DPAAlgorithm.java:

```
41     .....printContinuousFreeSlots(day, freeSlots);}
42 }
43 
44     private boolean isSlotAvailable(int start, int end, List<Integer> availableTimes) {
45         for (int i = start; i < end; ++i) {
46             if (!availableTimes.contains(i)) {
47                 return false;
48             }
49         }
50         return true;
51     }
52 
53     private void printContinuousFreeSlots(String day, List<Integer> freeSlots) {
54         System.out.println("Optimal 2-hour continuous time slots for replacement class on " + day + ":");
55         for (int i = 0; i < freeSlots.size(); ++i) {
56             System.out.println("Free from " + freeSlots.get(i) + ":00 to " + (freeSlots.get(i) + REQUIRED_FREE_HOURS) + ":00");
57         }
58     }
59 
60     public static void main(String[] args) {
61         DPAalgorithm scheduler = new DPAalgorithm();
62     }
63 
64     Map<String, List<Integer>> lecturerAvailable = new HashMap<>();
65     Map<String, List<List<Integer>>> studentAvailable = new HashMap<>();
66 
67     // Monday availability
68     lecturerAvailable.put("Monday", Arrays.asList(8, 9, 10, 11, 12, 13, 16, 17)); // Lecturer Dr. Hafeez
69     studentAvailable.put("Monday", Arrays.asList(
70         Arrays.asList(11, 13, 16, 17), // Student 1 (Miza)
71         Arrays.asList(8, 9, 11, 13, 17), // Student 2 (Jiki)
72         Arrays.asList(11, 13, 16, 17), // Student 3 (Hanan)
73         Arrays.asList(8, 9, 11, 13, 17) // Student 4 (Xue Rui)
74     ));
75 }
```

The status bar at the bottom shows: Writable, Smart Insert, 77:1:2879, ENG US, 9:21 AM, 26/6/2024.

IMPLEMENTATION



The screenshot shows the Eclipse IDE interface with the title bar "eclipse - algo second/src/algo/DPAlgorithm.java - Eclipse IDE". The menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. The left sidebar displays the project structure under "algo second": JRE System Library, src, algo (containing DPAlgorithm.java), GraphRej, Sorting, assignment2, Black_box_testing, design and analys, graph, Lab2, Lab4, Lab5, SimpleUnitTest, and White_box_testing. The main editor window shows Java code for a "DPAlgorithm.java" file. The code defines four arrays: lecturerAvailable, studentAvailable, lecturerFreeSlot, and studentFreeSlot, each mapping days of the week (Tuesday through Friday) to lists of time slots (8:00 to 17:00). It then iterates over these days to find continuous free slots for both lecturer and student.

```
77 // Tuesday availability
78 lecturerAvailable.put("Tuesday", Arrays.asList(8, 9, 10, 12, 13, 14, 15, 16, 17)); // Lecturer Dr Hafeez
79 studentAvailable.put("Tuesday", Arrays.asList(
80     Arrays.asList(8, 9, 10, 12, 13, 14, 15, 16, 17), // Student 1 (Miza)
81     Arrays.asList(12, 13), // Student 2 (Jiki)
82     Arrays.asList(12, 13), // Student 3 (Hanan)
83     Arrays.asList(12, 13, 16, 17) // Student 4 (Xue Rui)
84 )); // Wednesday availability
85 lecturerAvailable.put("Wednesday", Arrays.asList(8, 9, 10, 11, 12, 13, 16, 17)); // Lecturer Dr Hafeez
86 studentAvailable.put("Wednesday", Arrays.asList(
87     Arrays.asList(8, 13, 16, 17), // Student 1 (Miza)
88     Arrays.asList(12, 13, 14, 15, 16, 17), // Student 2 (Jiki)
89     Arrays.asList(8, 9, 10, 13, 14, 15, 16, 17), // Student 3 (Hanana)
90     Arrays.asList(8, 9, 12, 13, 15, 16, 17) // Student 4 (Xue Rui)
91 )); // Thursday availability
92 lecturerAvailable.put("Thursday", Arrays.asList(8, 9, 11, 12, 13, 14, 15, 16, 17)); // Lecturer Dr Hafeez
93 studentAvailable.put("Thursday", Arrays.asList(
94     Arrays.asList(12, 13, 14, 15, 16, 17), // Student 1 (Miza)
95     Arrays.asList(11, 12, 13, 14, 15, 16, 17), // Student 2 (Jiki)
96     Arrays.asList(13, 14, 15, 16, 17), // Student 3 (Hanana)
97     Arrays.asList(8, 12, 13, 14) // Student 4 (Xue Rui)
98 )); // Friday availability
99 lecturerAvailable.put("Friday", Arrays.asList(8, 9, 10, 11, 12, 13, 14, 15, 16, 17)); // Lecturer Dr Hafeez
100 studentAvailable.put("Friday", Arrays.asList(
101     Arrays.asList(8, 9, 10, 11, 12, 13, 14, 15, 16, 17), // Student 1 (Miza)
102     Arrays.asList(8, 12, 13, 14, 15, 16, 17), // Student 2 (Jiki)
103     Arrays.asList(8, 12, 13, 14, 15, 16, 17), // Student 3 (Hanana)
104     Arrays.asList(8, 9, 10, 11, 12, 13, 14, 15, 16, 17) // Student 4 (Xue Rui)
105 )); // Saturday availability
106 lecturerAvailable.put("Saturday", Arrays.asList(8, 9, 10, 11, 12, 13, 14, 15, 16, 17)); // Lecturer Dr Hafeez
107 studentAvailable.put("Saturday", Arrays.asList(
108     Arrays.asList(8, 9, 10, 11, 12, 13, 14, 15, 16, 17), // Student 1 (Miza)
109     Arrays.asList(8, 12, 13, 14, 15, 16, 17), // Student 2 (Jiki)
110     Arrays.asList(8, 12, 13, 14, 15, 16, 17), // Student 3 (Hanana)
111     Arrays.asList(8, 9, 10, 11, 12, 13, 14, 15, 16, 17) // Student 4 (Xue Rui)
112 )); // Sunday availability
113 for (String day : lecturerAvailable.keySet()) {
114     scheduler.findContinuousFreeSlot(day, lecturerAvailable.get(day), studentAvailable.get(day));
115 }
116 }
117 }
118 }
```

OUTPUT

eclipse - Eclipse IDE

File Edit Navigate Search Project Run Window Help

Console * DPAalgorithm.java

```
<terminated> GraphReplacementClass2 [Java Application] C:\Users\ASUS A409I\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_14.0.2.v20200815-0932\jre\bin\javaw.exe (26 Jun 2024, 7:21:15 a
```

Optimal 2-hour continuous time slots for replacement class on Monday:
Optimal 2-hour continuous time slots for replacement class on Thursday:
Free from 13:00 to 15:00
Optimal 2-hour continuous time slots for replacement class on Friday:
Free from 12:00 to 14:00
Free from 13:00 to 15:00
Free from 14:00 to 16:00
Free from 15:00 to 17:00
Free from 16:00 to 18:00
Optimal 2-hour continuous time slots for replacement class on Wednesday:
Free from 16:00 to 18:00
Optimal 2-hour continuous time slots for replacement class on Tuesday:
Free from 12:00 to 14:00

algo second

JRE System Library

src

algo

DPAAlgorithm

GraphReplacementClass2

Sorting

assignment2

Black_box_testing

design and analysis

graph

Lab2

Lab4

Lab5

SimpleUnitTest

White_box_testing

27°

ENG US

9:25 AM

26/6/2024

PRES

