



**FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY**

**GROUP PROJECT REPORT**

**CSC4202-1 (DESIGN AND ANALYSIS OF ALGORITHMS)**

**SEMESTER II 2022/2023**

**PROGRAMME :**

**BACHELOR OF COMPUTER SCIENCE WITH HONORS**

---

**COURSE : DESIGN AND ANALYSIS OF ALGORITHMS**

**COURSE CODE : CSC4202**

**GROUP : 1**

**ONLINE PORTFOLIO LINK: <https://github.com/Ramesh260402/CSC4202-G1-ProjectReport>**

**GROUP MEMBERS:**

| NAME                                     | MATRIC NUM |
|--|------------|
| RAMESH ARVIN A/L ANPALAGAN               | 213111     |
| PONMUGILLAN A/L MANI                     | 210301     |
| NIK QISTINA NURIN BINTI NIK SHAFUL ANWAR | 210839     |
| NOR SYAKILA BINTI SALIM                  | 212927     |

|   |           |
|---|-----------|
| <b>1.0 Original Scenario.....</b>   | <b>3</b>  |
| <b>2.0 Development of A Model.....</b>  | <b>3</b>  |
| <b>3.0 Importance of Optimal Solution.....</b>  | <b>4</b>  |
| <b>4.0 Suitability of Algorithms As Solution Paradigm (Specification of Algorithm).....</b> | <b>5</b>  |
| 3.1 Sorting.....  | 5         |
| 3.2 Divide and Conquer (DAC).....   | 6         |
| 3.3 Dynamic Programming (DP).....   | 6         |
| 3.4 Greedy Algorithms.....  | 6         |
| 3.5 Graph Algorithms.....   | 7         |
| 3.6 Final Choice of Algorithm.....  | 7         |
| 3.7 Consideration.....  | 8         |
| 3.7.1 Implementing Dynamic Programming.....   | 8         |
| 3.7.2 Sample of Implementation.....   | 9         |
| <b>4.0 Design The Algorithm.....</b>  | <b>11</b> |
| 4.1 Topological Sorting.....  | 11        |
| 4.1.1 Kahn's algorithm for topological sorting (Python).....                                | 11        |
| 4.1.2 Topological sorting using DFS (Java).....   | 13        |
| <b>5.0 Analysis of The Algorithm's Correctness.....</b>                                     | <b>14</b> |
| 5.1 Topological Sorting.....  | 14        |
| 5.1.1 Kahn's algorithm for topological sorting (Python).....                                | 14        |
| 5.1.2 Topological sorting using DFS (Java).....   | 15        |
| <b>8.0 Implementation of An Algorithm.....</b>  | <b>16</b> |
| <b>9.0 Program Testing.....</b>   | <b>16</b> |
| <b>10.0 Online Portfolio.....</b>   | <b>17</b> |
| <b>11.0 References.....</b>   | <b>17</b> |
| <b>11.0 Project Progress.....</b>   | <b>19</b> |
| 11.1 Week 10 , Milestone 1.....   | 19        |
| 11.2 Week 11 , Milestone 2.....   | 20        |
| 11.3 Week 12 , Milestone 3.....   | 21        |
| 11.4 Week 13 , Milestone 4.....   | 22        |
| 11.5 Week 14 , Milestone 6.....   | 23        |

## 1.0 Original Scenario

### Optimization of University Course Scheduling

Scenario: Timetabling issues can occur in a range of industries, such as healthcare, sports, transportation, and education. Here, we concentrate on a particular subset of scheduling issues known as the university course scheduling issue. This issue is frequently experienced in a lot of colleges around the world. Teacher assignment, course scheduling, class-teacher timetabling, student scheduling, and classroom assignment are the five sub-problems that can be further divided into.

The scheduling of students is the topic of this project. Imagine you are a student at university who has to come up with the **optimal course schedule** or best course plan for the forthcoming semester. The institution offers a wide range of courses with **various start times, requirements, and seat capacity**.

The number of desirable courses should be maximized while avoiding timing conflicts and fulfilling prerequisite requirements. An organized schedule ought to guarantee an equal distribution of professor and student groups. The planning of the class schedule is a crucial and challenging undertaking. Because there are so many courses, manual scheduling ultimately results in conflicts of various types, and its flaws become more obvious. The drawbacks of manual scheduling will be remedied by computer scheduling as computer science and technology advance.

## 2.0 Development of A Model

Data Type:

The courses and their dependencies would normally be represented as a directed graph, which is the data type needed to construct the scenario using Kahn's approach of Topological Sorting. The graph would represent each course as a node, and the connections between courses as directed edges.

Objective Function:

In this case, creating a proper course plan that complies with credit requirements and prerequisites would be the target function. The courses should be set up such that students can advance through the programme without going over their allotted credits or breaking any prerequisites.

Constraints:

1. Prerequisite Constraints: Each course may have one or more prerequisites, indicating that it must be taken after completing specific prerequisite courses. These constraints need to be satisfied in the generated schedule.

2. Credit Constraints: There are limitations on the number of credits a student can take in a given semester. The schedule should adhere to these credit constraints, ensuring that the total credits for each semester do not exceed the specified limit.

Examples:

1. Prerequisite Restrictions: Completion of Course A is a requirement for enrollment in Course B. Course A should be taken before Course B, according to the schedule.

2. Credit Constraint: A maximum of 15 credits may be taken per semester. The course allocation for each semester should be done in such a way that the maximum number of credits for that semester is 15.

Other requirements:

1. Main goal: To create a workable course plan that complies with prerequisite and credit restrictions.

2. Physical Space Restrictions: It is important to take into account the memory or storage needed to depict the graph and keep track of the schedule. To use as little space as possible, effective data structures should be used.

3. Time Restraints: The algorithm's effectiveness is essential, particularly for complex scheduling issues. To ensure that schedules are generated on time, the implementation should aim for an acceptable runtime.

4. Value restrictions: The scheduling algorithm should take into account any additional restrictions that are relevant to the situation, such as particular course offers, restricted resource availability, or desired time slots.

Topological sorting (Kahn's algorithm), which can be established as a viable model for resolving the course scheduling problem, can be developed by taking into account these restrictions, aims, and other needs.

### **3.0 Importance of Optimal Solution**

#### **1. *Maximizing Course Availability***

The most number of suitable courses can be enrolled in by students with the best course scheduling. Students have a better chance of finding the courses they need or desire to take when the course offerings are organised well, which results in a more satisfying educational experience.

#### **2. *Avoiding Timing Conflicts***

When two or more courses that a student wants to take are offered at the same time, scheduling problems may occur. Conflicts can be reduced by optimising the course schedule, giving students greater freedom to choose their courses and lowering the need for time-consuming revisions or compromises.

3. *Fulfilling Prerequisite Requirements*

Some courses have prerequisites, which means that before enrolling in higher-level courses, students must successfully finish preparatory courses. These prerequisites are taken into consideration while creating a course schedule, which guarantees that students can complete the necessary requirements logically and effectively.

4. *Efficient Resource Allocation*

The course schedule can be optimised to make the most effective use of resources like classrooms, lecturers, and teaching assistants. The institution can make the most use of its resources and prevent potential bottlenecks or inefficiencies by equitably spreading these resources among various courses and time slots.

5. *Improved Student and Professor Satisfaction*

Both students and professors experience less stress and annoyance when the course schedule is well-organized. By successfully planning their semester, students can ensure a balanced workload and reduce conflicts between their courses. A more equitable distribution of student groups benefits professors by enabling more manageable class sizes and enhancing effective teaching and engagement.

6. *Time and Cost Savings*

When dealing with a lot of courses, students, and limits, manual scheduling can be a time-consuming and error-prone operation. Students, teachers, and administrative staff can save a lot of time and effort by automating the scheduling process and identifying the best option. The institution may then be able to save money as a result of this.

In the university course scheduling situation, determining the best course schedule is crucial for maximising course availability, avoiding conflicts, completing requirements, effectively allocating resources, increasing satisfaction, and achieving time and money savings. The shortcomings of manual scheduling can be solved, resulting in a more efficient and simplified scheduling process by utilising computer scheduling and optimisation approaches.

#### **4.0 Suitability of Algorithms As Solution Paradigm (Specification of Algorithm)**

### **3.1 Sorting**

1. *Strengths*

Sorting can be useful for certain aspects of the course scheduling problem, such as ordering the courses based on prerequisites or sorting them by certain criteria like course credits or semester availability.

2. *Weaknesses*

Sorting alone cannot handle the full complexity of the course scheduling problem, as it doesn't consider prerequisites or credit constraints. It may require additional algorithms or techniques to ensure that the sorted schedule satisfies the constraints.

### 3.2 Divide and Conquer (DAC)

1. *Strengths*

DAC could be used to break down the course scheduling problem into smaller subproblems, solving each subproblem independently, and then combining the solutions. This approach can be effective when dealing with complex dependency structures.

2. *Weaknesses*

DAC may not directly address the prerequisites or credit constraints, and additional steps would be needed to ensure the overall schedule satisfies the requirements. Handling the combination of solutions and maintaining consistency across subproblems can also be challenging.

### 3.3 Dynamic Programming (DP)

1. *Strengths*

DP can be useful when there are overlapping subproblems and optimal substructure properties. It could potentially handle the course scheduling problem by considering the dependencies and credits, building a schedule incrementally, and finding an optimal solution.

2. *Weaknesses*

DP can be computationally expensive for large input sizes, and it may require defining an appropriate state and recursive relationship that captures the course dependencies and credit constraints. It can also be challenging to handle real-time updates or changes to the schedule.

### 3.4 Greedy Algorithms

1. *Strengths*

Greedy algorithms can provide simple and efficient solutions by making locally optimal choices at each step. In the course scheduling problem, a greedy approach could involve selecting the courses based on certain criteria, such as the number of prerequisites or credits.

2. *Weaknesses*

Greedy algorithms may not always guarantee an optimal global solution, especially when considering complex constraints like prerequisites and credit limits. The locally optimal choices made by the greedy algorithm may lead to suboptimal or infeasible schedules.

### 3.5 Graph Algorithms

1. *Strengths*

Graph algorithms, such as topological sorting or shortest path algorithms, can directly address the dependencies and prerequisites in the course scheduling problem. They provide a natural representation for modeling the relationships between courses and can help in building a schedule that satisfies the constraints.

## 2. *Weaknesses*

Graph algorithms may not inherently consider credit constraints, and additional steps would be required to incorporate them into the scheduling process. They may also require careful consideration of edge cases and exceptional scenarios to ensure the correctness and efficiency of the schedule.

The course scheduling issue might be solved using a variety of strategies. Prerequisites and dependencies can be handled by graph algorithms like topological sorting, and extra constraints or scheduling process optimisation can be achieved by using sorting, DAC, DP, or greedy algorithms. The specific needs, problem complexity, available input data, and performance factors will all influence the technique that is chosen.

## 3.6 Final Choice of Algorithm

We decided to choose the graph algorithms which are topologically sorting to implement our scenario. This is because Graph algorithms, particularly topological sorting, are well-suited for solving the Course Scheduling problem that involves considering prerequisites and credit constraints. Here are the reasons why topological sorting is a suitable solution paradigm for this problem:

### 1. *Prerequisite dependencies*

The Course Scheduling problem inherently involves prerequisites, where certain courses must be completed before taking others. Graph algorithms, such as topological sorting, are designed to handle dependencies between nodes in a graph. By representing the courses and prerequisites as nodes and edges in a directed graph, topological sorting can efficiently determine the order in which the courses should be taken.

### 2. *Credit constraints*

In addition to prerequisites, credit constraints are an essential consideration when creating a course schedule. Topological sorting does not directly address credit constraints, but it provides the foundation for incorporating such constraints into the scheduling algorithm. After obtaining the topological order, additional checks and algorithms can be implemented to assign courses to specific semesters while considering credit limits.

### 3. *Efficiency*

Topological sorting has a time complexity of  $O(|V| + |E|)$ , where  $|V|$  represents the number of courses and  $|E|$  represents the number of prerequisites. This time complexity is generally efficient for the Course Scheduling problem, especially considering that the number of prerequisites and courses is typically manageable. By utilizing topological

sorting, the algorithm can generate a valid course schedule in a reasonable amount of time.

4. *Correctness*

Topological sorting guarantees that the courses are scheduled in a valid order, satisfying all prerequisites. It ensures that no course is taken before its prerequisites are completed, thus maintaining the correctness of the schedule. By relying on a well-established algorithm, the Course Scheduling solution can provide accurate and reliable results.

5. *Scalability*

Graph algorithms, including topological sorting, are scalable and can handle larger instances of the Course Scheduling problem. As long as the number of courses and prerequisites remains within a reasonable range, topological sorting can efficiently generate the course schedule without significant performance issues.

Topological sorting is a suitable solution paradigm for the Course Scheduling problem due to its ability to handle prerequisite dependencies, potential integration with credit constraints, efficiency, correctness, and scalability. By utilizing this graph algorithm, you can generate a valid course schedule while considering prerequisites and credit constraints efficiently.

### **3.7 Consideration**

#### **3.7.1 Implementing Dynamic Programming**

However one consideration when we are completing this project is to implement Dynamic Programming for Course Scheduling Problem in the subset of Class Capacity Problem since in the original scenario that we created, seat capacity is also one of the criterias of optimized schedule that we defined. Dynamic programming can be a suitable solution paradigm for the Classroom Selection problem.

1. *Optimal substructure*

Dynamic programming is effective when a problem can be divided into overlapping subproblems, and the optimal solution can be constructed from the optimal solutions of its subproblems. In the Classroom Selection problem, the goal is to find the class with the minimum excess capacity, which can be seen as finding the optimal solution by considering the capacities of individual classes. Thus, the problem exhibits optimal substructure, making dynamic programming a viable approach.

2. *Overlapping subproblems*

The dynamic programming approach in the code snippet utilizes a 2D table (dp) to store intermediate results. Each cell in the table represents the minimum number of classrooms required to accommodate a certain number of students for a subset of



classes. By filling in this table iteratively, the algorithm effectively solves overlapping subproblems and avoids redundant computations.

### 3. *Time complexity*

The time complexity of the dynamic programming solution is  $O(n * m)$ , where  $n$  is the number of classes and  $m$  is the total number of students. The nested loops in the code iterate over the class capacities and the number of students, respectively. Since the dimensions of the table are proportional to the number of classes and the total number of students, the time complexity remains manageable even for larger problem instances.

### 4. *Efficiency*

Dynamic programming offers an efficient solution for the Classroom Selection problem. By computing and storing intermediate results in the dp table, the algorithm avoids redundant computations and optimizes the search for the class with the minimum excess capacity. The use of dynamic programming ensures that the solution is obtained in an efficient and systematic manner.

### 5. *Correctness*

The dynamic programming approach in the code snippet correctly determines the class with the minimum excess capacity. It computes the minimum number of classrooms required to accommodate different numbers of students for each subset of classes and identifies the class with the closest capacity to the total number of students.

Dynamic programming is a suitable solution paradigm for the Classroom Selection problem. It leverages the problem's optimal substructure and overlapping subproblems to efficiently find the class with the minimum excess capacity. The approach ensures correctness, efficiency, and manageable time complexity, making it a viable solution strategy for this problem.

## 3.7.2 Sample of Implementation

This is the example of implementation of Dynamic Programming for Course Scheduling Problem in the subset of class capacity

```
import java.util.Arrays;

public class ClassroomSelection {
    public static void main(String[] args) {
        int[] classCapacity = {10, 20, 30, 40, 50}; // Capacity of each class
        int totalStudents = 50; // Total number of students

        String[] classNames = {"A", "B", "C", "D", "E"}; // Class names

        int[][] dp = new int[classCapacity.length + 1][totalStudents + 1];

        // Initialize the table with a large value
```

```

int largeValue = totalStudents + 1;
for (int i = 0; i <= classCapacity.length; i++) {
    Arrays.fill(dp[i], largeValue);
}

// Base case: If there are no students, no classrooms are needed
for (int i = 0; i <= classCapacity.length; i++) {
    dp[i][0] = 0;
}

// Fill the table using dynamic programming
for (int i = 1; i <= classCapacity.length; i++) {
    for (int j = 1; j <= totalStudents; j++) {
        // If the current class can accommodate j students
        if (classCapacity[i - 1] <= j) {
            dp[i][j] = Math.min(dp[i - 1][j], 1 + dp[i][j - classCapacity[i - 1]]);
        } else {
            dp[i][j] = dp[i - 1][j];
        }
    }
}

// Find the class with the minimum excess capacity (closest to the total number
of students)
int bestClassIndex = -1;
int minExcessCapacity = Integer.MAX_VALUE;
for (int i = 0; i < classCapacity.length; i++) {
    int excessCapacity = classCapacity[i] - totalStudents;
    if (excessCapacity >= 0 && excessCapacity < minExcessCapacity) {
        minExcessCapacity = excessCapacity;
        bestClassIndex = i;
    }
}

// Print the best class if found
if (bestClassIndex != -1) {
    String bestClass = classNames[bestClassIndex];
    System.out.println("Best Class: " + bestClass);
    System.out.println("Minimum Classrooms Required: 1");
} else {
    System.out.println("No valid class found.");
}
}
}

```

## 4.0 Design The Algorithm

### 4.1 Topological Sorting

#### 4.1.1 Kahn's algorithm for topological sorting (Python)

1. Initialize the in-degree dictionary to store the in-degree of each node in the `course_graph` as 0.
2. Calculate the in-degree for each node by iterating over the nodes and their neighbors, incrementing the in-degree of each neighbor in the `in_degree` dictionary. This step does not require any recurrence.
3. Enqueue nodes with an in-degree of 0 into the queue by iterating over the nodes in the `in_degree` dictionary and adding the nodes with an in-degree of 0 to the queue. This step also does not require any recurrence.
4. Perform the topological sorting using Kahn's algorithm. This part does not involve recurrence.
  - Initialize an empty list `sorted_nodes` to store the sorted nodes.
  - While the queue is not empty, dequeue a node from the front of the queue.
  - Append the dequeued node to the `sorted_nodes` list.
  - Iterate over the neighbors of the dequeued node:
    - Decrement their in-degree in the `in_degree` dictionary.
    - If the in-degree of a neighbor becomes 0, enqueue it into the queue.
5. Create a new directed graph `new_graph` of type `nx.DiGraph()`.
6. Add nodes to `new_graph` from the `sorted_nodes` list. This step does not require recurrence.
7. Add edges to `new_graph` by copying the edges from the original `course_graph`. This step does not require recurrence.
8. Copy node data from the original `course_graph` to the corresponding nodes in `new_graph`. This step does not require recurrence.
9. Assign `new_graph` to the `self.top_graph` variable.

The algorithm uses an iterative approach and does not require any recurrence. The main idea is to calculate the in-degrees of nodes, enqueue nodes with an in-degree of 0, perform topological sorting using Kahn's algorithm, create a new graph with the sorted nodes and edges, and finally copy the node data from the original graph to the sorted graph. There is no specific optimization function mentioned in the given code.

#### Pseudocode:

1. Start
2. Declare and initialize variables:

- `classCapacity = {10, 20, 30, 40, 50}`
- `totalStudents = 50`
- `classNames = {"A", "B", "C", "D", "E"}`
- `dp = 2D array of size [classCapacity.length + 1][totalStudents + 1]`
- `largeValue = totalStudents + 1`
- `bestClassIndex = -1`
- `minExcessCapacity = Integer.MAX_VALUE`

3. Initialize the dp table with a large value using nested loops:

- Iterate over rows (i) from 0 to `classCapacity.length`:
  - Iterate over columns (j) from 0 to `totalStudents`:
    - Set `dp[i][j]` to `largeValue`

4. Set the base case for dp table when there are no students (`j = 0`):

- Iterate over rows (i) from 0 to `classCapacity.length`:
  - Set `dp[i][0]` to 0

5. Fill the dp table using dynamic programming:

- Iterate over rows (i) from 1 to `classCapacity.length`:
  - Iterate over columns (j) from 1 to `totalStudents`:
    - Check if `classCapacity[i - 1] <= j`:
      - If true, set `dp[i][j]` to the minimum of `dp[i - 1][j]` and `1 + dp[i][j - classCapacity[i - 1]]`
      - If false, set `dp[i][j]` to `dp[i - 1][j]`

6. Find the class with the minimum excess capacity:

- Iterate over classes (i) from 0 to `classCapacity.length - 1`:
  - Calculate `excessCapacity = classCapacity[i] - totalStudents`

- Check if `excessCapacity >= 0` and `excessCapacity < minExcessCapacity`:
  - If true, update `minExcessCapacity` to `excessCapacity` and set `bestClassIndex` to `i`
7. Check if a valid class is found (`bestClassIndex != -1`):
- If true, retrieve the name of the best class using `classNames[bestClassIndex]`
  - Print "Best Class: [bestClass]" and "Minimum Classrooms Required: 1"
8. If no valid class is found (`bestClassIndex == -1`), print "No valid class found."
9. End

#### **4.1.2 Topological sorting using DFS (Java)**

1. The `CourseScheduler2` class is initialized with the number of courses (`numCourses`) and the prerequisites for each course (`prerequisites`).
2. The constructor initializes the necessary data structures and populates the `prerequisites` list with the prerequisite courses for each course.
3. The `canFinish()` method is used to check if all the courses can be finished without any cyclic dependencies. This method calls the `hasCycle()` method for each course that has not been visited yet.
4. The `hasCycle()` method uses a recursive approach to check for cycles in the course dependencies. It marks the current course as visited and adds it to the `recursionStack` to keep track of the current path. It recursively checks the prerequisite courses and returns `true` if a cycle is detected.
5. The `printCourseSchedule()` method prints the course schedule if it is possible to schedule all the courses without cyclic dependencies. It retrieves the course names using the `getCourseName()` method and prints them in reverse order.
6. The `getCourseName()` method can be modified to map course IDs to their respective names according to a specific naming convention.

In terms of algorithm paradigm, the main part of the algorithm that involves recurrence is the `hasCycle()` method. It uses recursion to traverse the prerequisite courses and check for cycles. The recursion allows the algorithm to explore the dependencies in a depth-first manner. Regarding optimization, the algorithm could potentially benefit from memoization or caching techniques to avoid redundant calculations or lookups of course names. However, the given code does not incorporate any specific optimization function.

Overall, the algorithm uses a recursive approach to detect cycles in course dependencies and determine if all courses can be scheduled without cyclic dependencies.

## **5.0 Analysis of The Algorithm's Correctness**

### **5.1 Topological Sorting**

#### **5.1.1 Kahn's algorithm for topological sorting (Python)**

The provided code implements a course scheduling algorithm using a directed graph representation and performs topological sorting to create a course schedule.

##### ***Correctness***

1. Creating the graph: The ``create_graph`` method constructs a directed graph based on the input course data. It iterates over each course and its prerequisites, adding nodes and edges to the graph accordingly. The course data is stored as attributes for each node. This step correctly represents the course dependencies in the graph.

2. Topological sorting: The ``create_top_sort`` method performs topological sorting on the course graph using Kahn's algorithm. It calculates the in-degree of each node, enqueues nodes with in-degree 0, and then performs the sorting by processing nodes in the queue. The resulting sorted nodes are used to create a new graph. This step correctly generates a valid order of courses to satisfy prerequisites.

3. Creating the schedule: The ``create_schedule`` method creates a course schedule based on the sorted graph. It iterates over the sorted nodes, considering the prerequisites and credits for each course. It assigns courses to semesters while taking into account the desired credits and the starting semester. The schedule is generated as a dictionary representing years and semesters. This step correctly schedules the courses based on the given constraints.

Overall, the algorithm correctly creates a course graph, performs topological sorting, and generates a valid course schedule considering prerequisites, desired credits, and starting semester.

##### ***Time Complexity***

1. Creating the graph: The code iterates over the course input and prerequisites, adding nodes and edges to the graph. This step takes  $O(|V| + |E|)$  time, where  $|V|$  is the number of courses and  $|E|$  is the number of prerequisites.

2. Topological sorting: The topological sorting algorithm used is Kahn's algorithm, which has a time complexity of  $O(|V| + |E|)$ . It involves calculating in-degrees, enqueueing nodes with

in-degree 0, and processing nodes in the queue to update in-degrees and generate the sorted order.

3. Creating the schedule: The code iterates over the sorted nodes and performs various operations based on the course data and constraints. In the worst case, it may iterate over all the courses and perform checks for each prerequisite and semester. Thus, the time complexity is dependent on the number of courses and their attributes.

Therefore, the overall time complexity of the algorithm is  $O(|V| + |E|)$ , where  $|V|$  is the number of courses and  $|E|$  is the number of prerequisites.

Note: The space complexity of the algorithm is  $O(|V| + |E|)$  as it stores the course graph, topologically sorted graph, and the course schedule.

In summary, the algorithm has a time complexity of  $O(|V| + |E|)$ , where  $|V|$  is the number of courses and  $|E|$  is the number of prerequisites. It correctly creates the course graph, performs topological sorting, and generates a valid course schedule considering prerequisites, desired credits, and the starting semester.

### **5.1.2 Topological sorting using DFS (Java)**

The algorithm presented in the code solves the problem of determining whether a set of courses can be scheduled without any cyclic dependencies.

1. The `CourseScheduler2` class is initialized with the number of courses (`numCourses`) and the prerequisites for each course (`prerequisites`).
2. The constructor initializes the necessary data structures and populates the `prerequisites` list with the prerequisite courses for each course.
3. The `canFinish()` method is used to check if all the courses can be finished without any cyclic dependencies. This method calls the `hasCycle()` method for each course that has not been visited yet.
4. The `hasCycle()` method uses a recursive approach to check for cycles in the course dependencies. It marks the current course as visited and adds it to the `recursionStack` to keep track of the current path. It recursively checks the prerequisite courses and returns true if a cycle is detected.
5. The `printCourseSchedule()` method prints the course schedule if it is possible to schedule all the courses without cyclic dependencies. It retrieves the course names using the `getCourseName()` method and prints them in reverse order.
6. The `getCourseName()` method can be modified to map course IDs to their respective names according to a specific naming convention.

In terms of algorithm paradigm, the main part of the algorithm that involves recurrence is the `hasCycle()` method. It uses recursion to traverse the prerequisite courses and check for cycles. The recursion allows the algorithm to explore the dependencies in a depth-first manner.

Regarding optimization, the algorithm could potentially benefit from memoization or caching techniques to avoid redundant calculations or lookups of course names. However, the given code does not incorporate any specific optimization function.

Overall, the algorithm uses a recursive approach to detect cycles in course dependencies and determine if all courses can be scheduled without cyclic dependencies.

## 8.0 Implementation of An Algorithm

In our initial attempt, we implemented Topological Sorting using DFS in Java, and we were able to obtain the basic output successfully. However, as we aimed to enhance its functionality by incorporating visualization using JavaFX, we encountered challenges in resolving issues and obtaining the desired output.

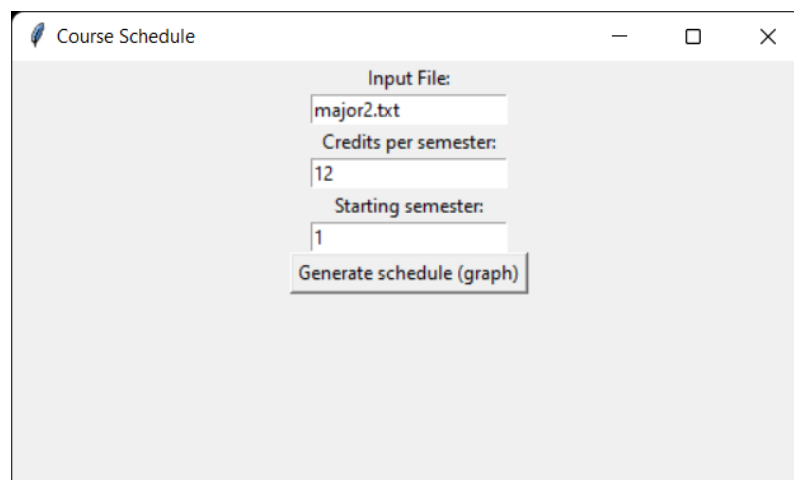
To address these difficulties, we decided to explore Kahn's Algorithm for Topological Sorting, opting to implement it using Python. Python, being a simpler language compared to Java, offered advantages in terms of ease of implementation and convenient libraries for visualization. By utilizing Python's libraries, we anticipated a smoother integration of the visualization component into our course scheduling solution.

By utilizing Matplotlib for visualization and NetworkX for graph representation, we were able to achieve an effective and visually informative course scheduling solution using Kahn's Algorithm in Python.

We think that Python is better to complete this project due to its concise syntax and the availability of specialized visualization libraries. Python's libraries often provide higher-level abstractions, enabling users to create visualizations quickly and easily with fewer lines of code.

The implementation of both Java and Python is compiled in the online portfolio.

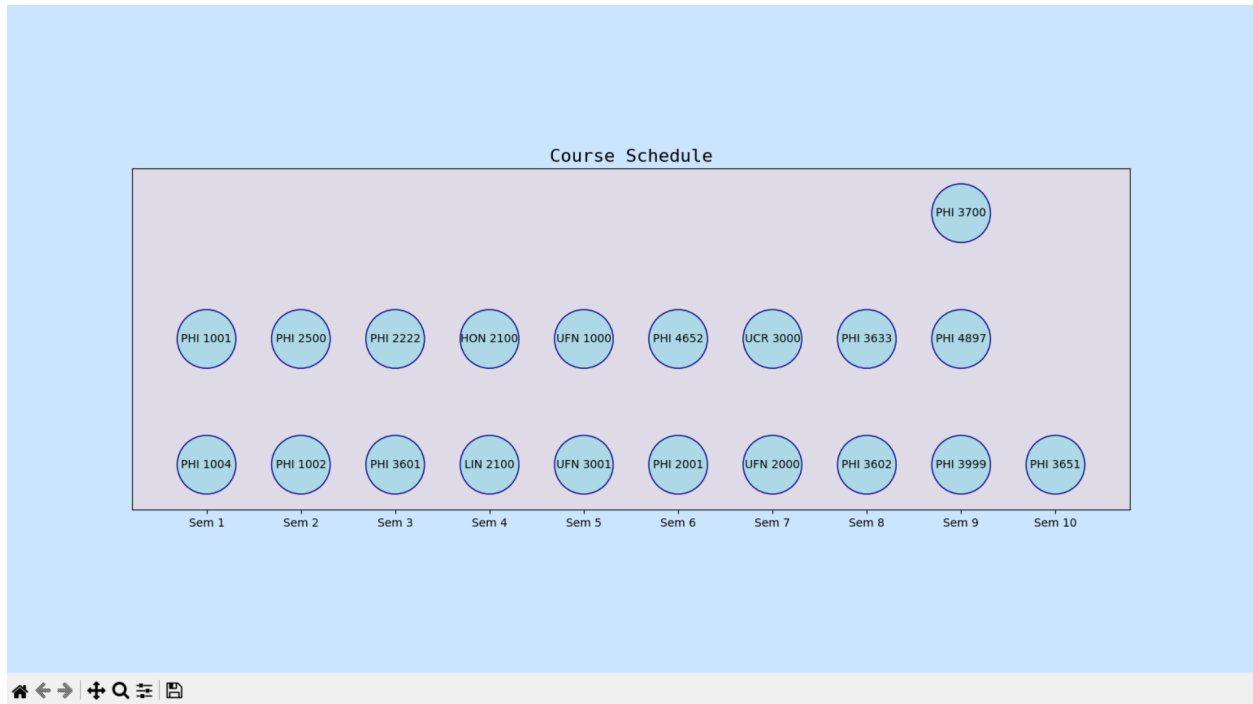
## 9.0 Program Testing



The screenshot shows a JavaFX window titled "Course Schedule". Inside the window, there is a light gray background with a central form. The form contains the following elements:

- A label "Input File:" followed by a text input field containing "major2.txt".
- A label "Credits per semester:" followed by a text input field containing "12".
- A label "Starting semester:" followed by a text input field containing "1".
- A button labeled "Generate schedule (graph)" at the bottom of the form.





## 10.0 Online Portfolio

The following is the link to the Online Portfolio:

<https://github.com/Ramesh260402/CSC4202-G1-ProjectReport>

## 11.0 References

- Ajwani, D., Friedrich, T. (2010). Average-case analysis of incremental topological ordering. *\*Discrete Applied Mathematics\**, 158(4), 240–250.
- Bellman, R. E., & Dreyfus, S. E. (2015). *Applied dynamic programming* (Vol. 2050). Princeton university press.
- Haeupler, B., Kavitha, T., Mathew, R., Sen, S., Tarjan, R.E. (2012). Incremental cycle detection, topological ordering, and strong component maintenance. *\*ACM Transactions on Algorithms\**, 8(1), 3:1–3:33.
- Goodrich, M. T., & Tamassia, R. (2015). *Algorithm design and applications* (Vol. 363). Hoboken: Wiley.
- Kaveh, A., & Rahami, H. (2006). Analysis, design and optimization of structures using force method and genetic algorithm. *International Journal for Numerical Methods in Engineering*, 65(10), 1570-1584.
- Pearce, D. J., & Kelly, P. H. (2007). A dynamic topological sort algorithm for directed acyclic graphs. *Journal of Experimental Algorithmics (JEA)*, 11, 1-7.
- Prabhumoye, S., Salakhutdinov, R., & Black, A. W. (2020). Topological sort for sentence ordering. *arXiv preprint arXiv:2005.00432*.
- Sherine, A., Jasmine, M., Peter, G., & Alexander, S. A. (2023). *\*Algorithm and Design Complexity\**. CRC Press.
- Vince, A. (2002). A framework for the greedy algorithm. *Discrete Applied Mathematics*, 121(1-3), 247-260.

## 11.0 Project Progress

### 11.1 Week 10 , Milestone 1

|                             |  |                             |                             |  |                            |  |                      |                         |                             |                             |                             |                             |
|-----------------------------|--|-----------------------------|-----------------------------|--|----------------------------|--|----------------------|-------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|
| Milestone 1                 | Planning   |                             |                             |  |                            |  |                      |                         |                             |                             |                             |                             |
| Date (Wk)                   | 26/05/23 (Week 10)   |                             |                             |  |                            |  |                      |                         |                             |                             |                             |                             |
| Description/<br>sketch      | 1. Established scenario refinement<br><br>2. Enhance accuracy and realism of the project   |                             |                             |  |                            |  |                      |                         |                             |                             |                             |                             |
| Role                        | <table><tr><td>Ramesh Arvin A/L Anpalagan</td><td>Nik Qistina Nurin Binti NikShaiful Anwar</td><td>Ponmugillan A/L Mani</td><td>Nor Syakila binti Salim</td></tr><tr><td>Documentati on, Researching</td><td>Documentati on, Researching</td><td>Documentati on, Researching</td><td>Documentati on, Researching</td></tr></table> |                             |                             |  | Ramesh Arvin A/L Anpalagan | Nik Qistina Nurin Binti NikShaiful Anwar | Ponmugillan A/L Mani | Nor Syakila binti Salim | Documentati on, Researching | Documentati on, Researching | Documentati on, Researching | Documentati on, Researching |
| Ramesh Arvin A/L Anpalagan  | Nik Qistina Nurin Binti NikShaiful Anwar   | Ponmugillan A/L Mani        | Nor Syakila binti Salim     |  |                            |  |                      |                         |                             |                             |                             |                             |
| Documentati on, Researching | Documentati on, Researching  | Documentati on, Researching | Documentati on, Researching |  |                            |  |                      |                         |                             |                             |                             |                             |

## 11.2 Week 11 , Milestone 2

|                             |   |                             |                             |  |                            |   |                      |                         |                             |                             |                             |                             |
|-----------------------------|---|-----------------------------|-----------------------------|--|----------------------------|---|----------------------|-------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|
| Milestone 4                 | Exploring   |                             |                             |  |                            |   |                      |                         |                             |                             |                             |                             |
| Date (Wk)                   | 02/06/23 (Week 11)  |                             |                             |  |                            |   |                      |                         |                             |                             |                             |                             |
| Description/<br>sketch      | <div>1. Explore various example of solutions</div> <div>2. Identify the solutions most suitable for the project/scenario</div> <div>3. Determine why the selected solutions and the example of problems are closely related to our problem scenario and project's objective</div>   |                             |                             |  |                            |   |                      |                         |                             |                             |                             |                             |
| Role                        | <table><tr><td>Ramesh Arvin A/L Anpalagan</td><td>Nik Qistina Nurin Binti Nik Shaiful Anwar</td><td>Ponmugillan A/L Mani</td><td>Nor Syakila binti Salim</td></tr><tr><td>Documentati on, Researching</td><td>Documentati on, Researching</td><td>Documentati on, Researching</td><td>Documentati on, Researching</td></tr></table> |                             |                             |  | Ramesh Arvin A/L Anpalagan | Nik Qistina Nurin Binti Nik Shaiful Anwar | Ponmugillan A/L Mani | Nor Syakila binti Salim | Documentati on, Researching | Documentati on, Researching | Documentati on, Researching | Documentati on, Researching |
| Ramesh Arvin A/L Anpalagan  | Nik Qistina Nurin Binti Nik Shaiful Anwar   | Ponmugillan A/L Mani        | Nor Syakila binti Salim     |  |                            |   |                      |                         |                             |                             |                             |                             |
| Documentati on, Researching | Documentati on, Researching   | Documentati on, Researching | Documentati on, Researching |  |                            |   |                      |                         |                             |                             |                             |                             |

### 11.3 Week 12 , Milestone 3

|                               |   |                      |                         |  |                            |   |                      |                         |                               |                               |                      |                      |
|-------------------------------|---|----------------------|-------------------------|--|----------------------------|---|----------------------|-------------------------|-------------------------------|-------------------------------|----------------------|----------------------|
| Milestone 4                   | Developing  |                      |                         |  |                            |   |                      |                         |                               |                               |                      |                      |
| Date (Wk)                     | 09/06/23 (Week 12)  |                      |                         |  |                            |   |                      |                         |                               |                               |                      |                      |
| Description/<br>sketch        | 1. Develop the program<br>2. Edit the codes if necessary<br>3. Completing the implementation<br>4. Debugging if encounter any issues  |                      |                         |  |                            |   |                      |                         |                               |                               |                      |                      |
| Role                          | <table><tr><td>Ramesh Arvin A/L Anpalagan</td><td>Nik Qistina Nurin Binti Nik Shaiful Anwar</td><td>Ponmugillan A/L Mani</td><td>Nor Syakila binti Salim</td></tr><tr><td>Documentati on, Code Debugger</td><td>Documentati on, Code Debugger</td><td>Developer of Program</td><td>Developer of Program</td></tr></table> |                      |                         |  | Ramesh Arvin A/L Anpalagan | Nik Qistina Nurin Binti Nik Shaiful Anwar | Ponmugillan A/L Mani | Nor Syakila binti Salim | Documentati on, Code Debugger | Documentati on, Code Debugger | Developer of Program | Developer of Program |
| Ramesh Arvin A/L Anpalagan    | Nik Qistina Nurin Binti Nik Shaiful Anwar   | Ponmugillan A/L Mani | Nor Syakila binti Salim |  |                            |   |                      |                         |                               |                               |                      |                      |
| Documentati on, Code Debugger | Documentati on, Code Debugger   | Developer of Program | Developer of Program    |  |                            |   |                      |                         |                               |                               |                      |                      |

#### 11.4 Week 13 , Milestone 4

|                            |   |                      |                         |  |                            |   |                      |                         |                |                |                      |                      |
|----------------------------|---|----------------------|-------------------------|--|----------------------------|---|----------------------|-------------------------|----------------|----------------|----------------------|----------------------|
| Milestone 5                | Analyzing   |                      |                         |  |                            |   |                      |                         |                |                |                      |                      |
| Date (Wk)                  | 16/06/23 (Week 13)  |                      |                         |  |                            |   |                      |                         |                |                |                      |                      |
| Description/<br>sketch     | 1. Conduct an analysis of correctness of the implementation<br>2. Verify the accuracy and reliability of the code<br>3. Perform analysis of the time complexity for the implementation<br>4. Carefully examine the algorithm utilized in the code to determine efficiency of the code       |                      |                         |  |                            |   |                      |                         |                |                |                      |                      |
| Role                       | <table><tr><td>Ramesh Arvin A/L Anpalagan</td><td>Nik Qistina Nurin Binti Nik Shaiful Anwar</td><td>Ponmugillan A/L Mani</td><td>Nor Syakila binti Salim</td></tr><tr><td>Documentati on</td><td>Documentati on</td><td>Developer of Program</td><td>Developer of Program</td></tr></table> |                      |                         |  | Ramesh Arvin A/L Anpalagan | Nik Qistina Nurin Binti Nik Shaiful Anwar | Ponmugillan A/L Mani | Nor Syakila binti Salim | Documentati on | Documentati on | Developer of Program | Developer of Program |
| Ramesh Arvin A/L Anpalagan | Nik Qistina Nurin Binti Nik Shaiful Anwar   | Ponmugillan A/L Mani | Nor Syakila binti Salim |  |                            |   |                      |                         |                |                |                      |                      |
| Documentati on             | Documentati on  | Developer of Program | Developer of Program    |  |                            |   |                      |                         |                |                |                      |                      |

### 11.5 Week 14 , Milestone 6

|                            |   |                           |                           |  |                            |   |                      |                         |                           |                           |                           |                           |
|----------------------------|---|---------------------------|---------------------------|--|----------------------------|---|----------------------|-------------------------|---------------------------|---------------------------|---------------------------|---------------------------|
| Milestone 6                | Finalization  |                           |                           |  |                            |   |                      |                         |                           |                           |                           |                           |
| Date (Wk)                  | 23/06/23 (Week 14)  |                           |                           |  |                            |   |                      |                         |                           |                           |                           |                           |
| Description/<br>sketch     | <div>1. Prepare online portfolio (by using GitHub, Google Sites or Google Colab)</div> <div>2. Showcase project progress, compiling the codes, pseudocodes, illustrating the problem and describe the algorithm analysis</div> <div>3. Preparing for the presentation</div> <div>4. Deliver the presentation</div>          |                           |                           |  |                            |   |                      |                         |                           |                           |                           |                           |
| Role                       | <table><tr><td>Ramesh Arvin A/L Anpalagan</td><td>Nik Qistina Nurin Binti Nik Shaiful Anwar</td><td>Ponmugillan A/L Mani</td><td>Nor Syakila binti Salim</td></tr><tr><td>Documentati on, Presentor</td><td>Documentati on, Presentor</td><td>Documentati on, Presentor</td><td>Documentati on, Presentor</td></tr></table> |                           |                           |  | Ramesh Arvin A/L Anpalagan | Nik Qistina Nurin Binti Nik Shaiful Anwar | Ponmugillan A/L Mani | Nor Syakila binti Salim | Documentati on, Presentor | Documentati on, Presentor | Documentati on, Presentor | Documentati on, Presentor |
| Ramesh Arvin A/L Anpalagan | Nik Qistina Nurin Binti Nik Shaiful Anwar   | Ponmugillan A/L Mani      | Nor Syakila binti Salim   |  |                            |   |                      |                         |                           |                           |                           |                           |
| Documentati on, Presentor  | Documentati on, Presentor   | Documentati on, Presentor | Documentati on, Presentor |  |                            |   |                      |                         |                           |                           |                           |                           |