Metropolitan University of Tirana
Data Structures
Final Project

1 Final Project

The goal of a final project is to provide you with an opportunity to apply and extend your knowledge of data structures and algorithms in a practical context. By working on a project, you can demonstrate your understanding of key concepts, develop your programming skills, and gain experience in problem-solving, design, and implementation. The final project should help you to work collaboratively, communicate effectively, and reflect on your learning experience.

1.1 Formalities

- The project is worth **20 points** of your final grade.
- The project **deadline is two weeks before the end of the semester**.
- The project can be completed in **teams of up to three** people.

1.2 General Requirements

- Design and implement an original program that utilizes a variety of common data structures such as lists, arrays, maps, trees, and graphs.
- The project should be original and not plagiarized from any existing codebase or resource.
- Present your solutions in a clear and organized manner, highlighting the effectiveness and complexity of the data structures used.
- Properly comment your code and push it to a Github repository for review and feedback.
- Test your program thoroughly to ensure its correctness and efficiency.
- Be prepared to explain and justify your design decisions.
- Explore more advanced data structures and algorithms beyond those covered in the course to demonstrate your understanding and creativity.
- Take into consideration the effectiveness and complexity of the data structures used in your program.
- Provide a written report that documents your program's design, implementation, and testing process.

1.3 Project Ideas

The final project should cover and extend the knowledge and topics covered in the data structures in Java course. The topic you choose should encourage you to think creatively and critically and push you to explore new ways of using data structures to solve real-world problems. Remember, the goal of this project is to demonstrate your mastery of the subject matter and develop your problem-solving skills. You may choose from the following list of topics, or a similar (pre-approved) idea

1.3.1 Management Systems

- **To-do-list application**: The to-do-list application should allow users to create and manage a list of tasks to be completed, using a data structure such as a linked list or stack to store the tasks. The application should allow users to add new tasks, mark tasks as complete, and reorder tasks based on priority or due date. The application should also provide a way for users to view and manage completed tasks.
- **Decision tree for a specific scenario**: The decision tree project should involve creating a decision-making tool for a specific scenario, such as choosing a college major or selecting a vacation destination. The application should use a decision tree data structure to model the decision-making process, with nodes representing different choices and outcomes. The application should allow users to traverse the decision tree by selecting different choices, and should provide feedback and recommendations based on the user's selections. The application should also allow for the addition or removal of nodes as needed to modify the decision tree.
- **Music library manager**: Develop a program that allows users to store and manage a collection of music tracks, using a data structure such as an array or linked list to store the track information. The application should allow users to add new tracks, search for tracks based on various criteria such as artist or genre, and create playlists of favorite tracks.
- **Inventory management system**: Create a program that helps businesses manage their inventory. The application should allow users to add new products, track inventory levels, and generate reports on inventory trends and sales performance. The system can also include alerts for low inventory levels and automated restocking.
- **Health tracker**: Develop an application that allows users to track their health metrics such as weight, blood pressure, and exercise activity, using data structures. The application should allow users to set goals for their health metrics, track their progress, and generate reports on their health trends. The system can also include features such as reminders for medication and doctor appointments.

1.3.2 Games

- **Sudoku solver**: A Sudoku solver is a program that takes as input a partially filled 9x9 Sudoku grid and uses an algorithm that implements a (backtracking) technique to fill in the empty cells with numbers from 1 to 9. The solver should ensure that each row, column, and 3x3 sub-grid contains all the digits from 1 to 9 without any duplicates. If the input grid is invalid and cannot be solved, the solver should output a message. If the grid has multiple solutions, the solver may output all possible solutions. The solver should use appropriate data structures such as arrays, lists, or maps to represent the Sudoku grid and track the candidate numbers for each empty cell.

- **Hangman**: Hangman is a word-guessing game where one player thinks of a word and the other player tries to guess the word by suggesting letters. The player has a limited number of incorrect guesses before they lose the game. To implement a Hangman game in Java, we need to define several requirements. First, the program should select a random word from a list of words. Second, the program should allow the user to enter letters as guesses. Third, the program should check if the guessed letter is in the word and display the letters in their correct position if they are. Fourth, the program should keep track of the number of incorrect guesses made by the user and end the game if they exceed a predetermined number. Finally, the program should display a message to the user if they win or lose the game. The implementation of Hangman in Java requires the use of data structures such as arrays, lists, and maps to store and manipulate the game state and track the user's guesses.

- **Towers of Hanoi**: The Towers of Hanoi is a mathematical puzzle that consists of three pegs and a series of disks of different sizes. The disks are initially stacked on one peg in decreasing order of size, with the largest at the bottom and the smallest at the top. The goal of the puzzle is to transfer the entire stack of disks to another peg, following three simple rules:

  - Only one disk can be moved at a time.
  - A larger disk cannot be placed on top of a smaller disk.
  - The stack must be transferred from the starting peg to the target peg.

  To implement a solution to the Towers of Hanoi problem, we need to define several requirements. First, the program should take as input the number of disks and the starting and target pegs. Second, the program should display each move of the disks as they are transferred between the pegs. Third, the program should check that the rules of the game are not violated, such as trying to place a larger disk on top of a smaller disk. Finally, the program should terminate when the entire stack of disks has been successfully transferred to the target peg. The implementation of the Towers of Hanoi problem requires the use of recursive algorithms and appropriate data structures to keep track of the state of the game and the movement of the disks between the pegs.

- **The N-Queen problem** The `N-Queen` problem is a classic computer science problem that involves placing `N` chess queens on an `N x N` chessboard in such a way that no two queens attack each other. Implement a solution to the `N-Queen` problem in Java, which involves generating all possible arrangements of queens on the board and then checking each arrangement to see if it is a valid solution. The program should allow the user to specify the value of `N`, and it should output all valid solutions to the problem. Additionally, the program should be optimized to minimize the number of checks required to find all solutions, as the brute-force approach can become prohibitively slow for larger values of N. The project should also include appropriate testing and documentation to ensure the correctness and usability of the program.

1.3.3 Data Structures for Path Planning

- **Maze solver**: To implement a maze solver, the program should take as input a maze represented as a two-dimensional array and find a path from the start to the end of the maze. The program should use appropriate data structures such as stacks or queues to keep track of visited cells and candidate paths. The solution should be found using algorithms such as depth-first search or breadth-first search.
- **Flight path planning**: To implement a flight path planning system, the program should take as input a list of airports and flights between them, and allow the user to find the shortest path between two airports. The program should use graph data structures to represent the airports and the flights between them, and algorithms such as Dijkstra's or Bellman-Ford to find the shortest path between two airports.
- **Robot path planning with A-star**: To implement a robot path planning system with A-star, the program should take as input a grid representing the environment, the robot's starting position, and the goal position. The program should use A-star algorithm and heuristic functions to find the optimal path from the starting position to the goal position while avoiding obstacles in the environment.
- **Bus route planning**: To implement a bus route planning system, the program should take as input a list of bus stops, the routes connecting them, and the schedule of bus arrivals and departures. The program should use graph data structures to represent the bus stops and the routes between them, and algorithms such as Dijkstra's or Bellman-Ford to find the shortest path or the fastest route between two stops. The program should also display the schedule of bus arrivals and departures for each stop.