**Format for Micro- Project Report**

**(Even semester 2022-23)**

**Title of Micro-Project**

**Brief Introduction** *(Importance of the Project)*

**Aim of the Micro Project**

1 ......................................................................................................................................................

2 ......................................................................................................................................................

3 ......................................................................................................................................................

4 ......................................................................................................................................................

5 .....................................................................................................................................................

**Course Outcomes Integrated**

1 ......................................................................................................................................................

2 ......................................................................................................................................................

3 ......................................................................................................................................................

**Action Plan**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.No.** | **Details of activity** | **Planned start date** | **Planned finish date** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Name of responsible team members/persons**

**Actual Procedure Followed**  *(Write step wise the work was done, including which team member did what work and how the data was analysed (if any)*

**Actual Resources Used**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S.No.** | **Name of resource/material** | **Specifications** | **Quantity** | **Remark** |
| **1.** |  |  |  |  |
| **2.** |  |  |  |  |
| **3.** |  |  |  |  |
| **4.** |  |  |  |  |

**Outputs of the Micro project (**Skill developed/learning out of this micro project)

**Submitted to:**

**Submitted by:**

**Project Report: Comparative Analysis of Brute Force and Backtracking Approaches for Solving the 5x5 N-Queens Problem**

**1. Introduction: The N-Queens problem is a classic combinatorial puzzle with applications in various fields. In this project, we compare the execution time of two approaches, brute force and backtracking, for solving the 5x5 N-Queens problem using Python as the programming language.**

**2. Problem Definition: The task is to place 5 queens on a 5x5 chessboard in such a way that no two queens threaten each other, i.e., no two queens share the same row, column, or diagonal.**

**3. Methodology:**

***3.1 Brute Force Approach:* The brute force approach systematically explores all possible configurations of queens on the chessboard until a valid solution is found.**

***3.2 Backtracking Approach:* The backtracking approach is more efficient, systematically exploring potential solutions and backtracking as soon as an invalid configuration is detected.**

**4. Implementation: The code was implemented in Python, and it includes functions for both the brute force and backtracking approaches. The execution time for each approach is measured and compared.**

**5. Code Implementation:**

**# Importing required libraries**

**import time**

*# Function to check if placing a queen at (row, col) is safe*

*def is\_safe(board, row, col):*

*# Check for queens in the same row*

*if any(board[row]):*

*return False*

*# Check for queens in the same column*

*if any(board[i][col] for i in range(len(board))):*

*return False*

*# Check for queens in the same diagonal (left to right)*

*if any(board[i][j] for i, j in zip(range(row, -1, -1), range(col, -1, -1))):*

*return False*

*# Check for queens in the same diagonal (right to left)*

*if any(board[i][j] for i, j in zip(range(row, -1, -1), range(col, len(board)))):*

*return False*

*return True*

*# Function to solve N-Queens problem using brute force*

*def brute\_force\_n\_queens(board, row):*

*if row == len(board):*

*return True # All queens are placed successfully*

*for col in range(len(board)):*

*if is\_safe(board, row, col):*

*board[row][col] = 1 # Place queen*

*if brute\_force\_n\_queens(board, row + 1):*

*return True # If placing queen at (row, col) leads to a solution*

*board[row][col] = 0 # If placing queen at (row, col) does not lead to a solution*

*return False*

*# Function to solve N-Queens problem using backtracking*

*def backtracking\_n\_queens(board, row):*

*if row == len(board):*

*return True # All queens are placed successfully*

*for col in range(len(board)):*

*if is\_safe(board, row, col):*

*board[row][col] = 1 # Place queen*

*if backtracking\_n\_queens(board, row + 1):*

*return True # If placing queen at (row, col) leads to a solution*

*board[row][col] = 0 # If placing queen at (row, col) does not lead to a solution*

*return False*

*# Function to measure execution time for a given approach*

*def measure\_execution\_time(solution\_method, board):*

*start\_time = time.time()*

*solution\_found = solution\_method(board, 0)*

*end\_time = time.time()*

*execution\_time = end\_time - start\_time*

*return solution\_found, execution\_time*

*# Main function*

*def main():*

*# Initialize a 5x5 chessboard*

*board\_size = 5*

*chessboard = [[0] \* board\_size for \_ in range(board\_size)]*

*# Measure execution time for brute force approach*

*brute\_force\_solution, brute\_force\_time = measure\_execution\_time(brute\_force\_n\_queens, chessboard)*

*# Measure execution time for backtracking approach*

*backtracking\_solution, backtracking\_time = measure\_execution\_time(backtracking\_n\_queens, chessboard)*

*# Print results*

*print("Brute Force Solution:", "Found" if brute\_force\_solution else "Not Found")*

*print("Brute Force Execution Time:", brute\_force\_time, "seconds")*

*print("\nBacktracking Solution:", "Found" if backtracking\_solution else "Not Found")*

*print("Backtracking Execution Time:", backtracking\_time, "seconds")*

*if \_\_name\_\_ == "\_\_main\_\_":*

*main()*

**6. Execution and Testing: The experiments were conducted on a machine with standard specifications. The code solves the 5x5 N-Queens problem using both brute force and backtracking approaches, measuring the execution time for each.**

**7. Results: The results clearly show the difference in execution time between the brute force and backtracking approaches. Backtracking is expected to outperform brute force due to its ability to prune the search space.**

**8. Analysis: The backtracking approach is anticipated to have a significantly lower execution time than the brute force approach. This is due to the backtracking algorithm's ability to eliminate invalid configurations early in the process, reducing the search space and improving efficiency.**

**9. Conclusion: In conclusion, the project provides a comparative analysis of the brute force and backtracking approaches for solving the 5x5 N-Queens problem. The results demonstrate the advantages of using backtracking in terms of execution time efficiency.**

**10. Future Work: Future work may involve extending the analysis to larger N values, exploring optimizations in the implementation, and considering alternative algorithms for solving the N-Queens problem.**

**11. References: No external references were used for this project. The project is based on fundamental concepts of the N-Queens problem and algorithmic approaches.**