

Problem Solving Workshop

Phase 1: Foundations

Instructor: Abhishek Bansal
Academic Year: 2025

Course Overview

This two-week workshop introduces foundational problem-solving techniques through classic puzzles, algorithmic strategies, and real-world modeling exercises. Students will build confidence in recursion, search algorithms, data structures, and collaborative design.

Phase 1 Curriculum Overview

1. Sudoku – Baseline Quiz

- Sudoku, logic-grid, and handshake-count puzzles ($N(N - 1)/2$)
- Survey: "How did you solve each?"

Outcome: Gauge current skills and problem-solving intuition.

2. N-Queens Backtracking

- Explain rules
- Code stub: `def place_queens(row, board): ...`
- Test on $N = 4$ and $N = 8$

Outcome: Hands-on recursion and pruning.

3. Sudoku Solver

- Constraint propagation and backtracking
- Fill in logic for provided Python function signature

Outcome: Reinforce state-space search.

4. Graphs via Word Ladders – Mazes

- Introduction to graphs (nodes, edges)
- BFS/DFS on word ladders
- Maze-path visualization

Outcome: Bridge puzzles to graph algorithms.

5. Group Challenge – Extensions

- Teams choose from N-Queens, Sudoku, word ladders, mazes, or optional puzzles (Lights Out, Josephus, Tower of Hanoi, Coin Change, Word Search)
- Sketch and pitch an improved solution

Outcome: Collaboration and peer teaching.

6. Queues – Real-World Simulation

- Live-code a bank-line simulator using a queue
- Stack demo: undo/redo

Outcome: Apply data structures in real-world contexts.

7. Game AI – Tic-Tac-Toe

- Provide stub: `def minimax(board, player): ...`
- Implement and play versus AI

Outcome: Expose decision-tree logic.

8. Event-Driven Modeling: Traffic / Queue

- Model traffic lights or a bank queue with events
- Discuss real-world modeling challenges

Outcome: Experience event-driven thinking.

9. Ideation Workshop

- Brainstorm campus/community problems (library booking, canteen queues)
- Structured voting on top ideas

Outcome: Nurture interdisciplinary vision.

10. Mini-Pitch – Reflection

- Three-minute team pitches with solution outlines
- Reflection: "What did we learn?"

Outcome: Build confidence and prepare for Phase 2 capstone.

Setup Instructions

Students should follow these steps to access materials:

- Sign up or log into Replit: replit.com
- Take the Quiz on Google Forms: [Click here](#)
- View the Progress Tracker: [Click here](#)

Puzzles & Baseline Quiz

Ice-Breaker Puzzles

Complete the following exercises in 20 minutes:

- a. **Sudoku (4×4 grid):** Fill each row, column, and 2×2 block with numbers 1–4 exactly once.
- b. **Logic Grid Puzzle:** Three students (A, B, C) each solved *one* of three puzzles (maze, riddle, sudoku-mini) in *one* of three times (5, 10, 15 minutes), and no two students chose the same puzzle or time.
 - A did **not** take 5 minutes.
 - The riddle was solved in 10 minutes.
 - C solved the maze.
 - The sudoku-mini was completed in the shortest time (5 minutes).

Determine who solved which puzzle and in what time.

- c. **Handshake Count:** In a group of 7, every pair shakes hands exactly once. How many total handshakes occur?

Baseline Quiz

Answer the following in the next 10 minutes:

1. Compute the handshake count for $N = 7$ and verify using the formula $\frac{N(N-1)}{2}$.
2. Write pseudocode for computing handshake count for a general N :

```
function handshakeCount(N):  
    # Each of N people shakes hands with N-1 others  
    # Divide by 2 to avoid double-counting  
    return N * (N - 1) / 2
```

3. Reflect: Which problem-solving heuristic did you use most (decomposition, pattern recognition, abstraction)?

Backtracking I – N-Queens

Problem Description

Place one queen in each row of an $N \times N$ chessboard so that no two queens attack each other. A queen attacks along its row, column, and both diagonals.

Recursive Pseudocode

Use the following stub to implement your solution:

```
function placeQueens(row, board):
    if row > N:
        printSolution(board)
        return
    for col = 1 to N:
        if isSafe(row, col, board):
            board[row] = col
            placeQueens(row + 1, board)
            board[row] = 0 # backtrack

function isSafe(r, c, board):
    for prevRow = 1 to r - 1:
        prevCol = board[prevRow]
        if prevCol == c or
            abs(prevCol - c) == abs(prevRow - r):
            return false
    return true
```

Try it online: Run the N-Queens demo on Replit

Exercise

- Implement the pseudocode and print one valid arrangement for $N = 4$.
- Extend your code to count all solutions for $N = 4$ and report the total.
- *(Optional)* Test your solver for $N = 8$ and note the number of solutions.

Sudoku Solver Session

Problem Description

Students will implement a backtracking algorithm to solve a 4×4 Sudoku. Each row, column, and 2×2 block must contain numbers 1–4 exactly once.

Recursive Backtracking Stub

Use this function signature:

```
function solveSudoku(grid):
    if no empty cells:
        printSolution(grid)
        return true
    pick an empty cell (r,c)
    for num = 1 to 4:
        if isValid(grid, r, c, num):
            grid[r][c] = num
            if solveSudoku(grid): return true
            grid[r][c] = 0    # backtrack
    return false
```

Exercises

1. Fill in the body of `solveSudoku(grid)` using the stub above.
2. Test your solver on a sample 4×4 puzzle:

```
[ [0,2,0,4],
  [3,0,1,0],
  [0,1,0,3],
  [4,0,3,0] ]
```

3. *(Optional)* Extend your code to handle 9×9 puzzles.

Outcome: Reinforce backtracking and constraint propagation.