# Deliverable 2: Initial Implementation & Data Structure Integration
# Sudoku Solver

Shaikh Muhammad Irtiza, Shahram Khan, Komail Lokhandwala

April 11, 2025

## 1 Introduction

This report presents the initial implementation of a terminal-based Sudoku Solver for the ES221 Data Structures and Algorithms course. The focus is on implementing core data structures and demonstrating their functionality, as required for Deliverable 2. The solver uses a 9x9 grid to represent a Sudoku puzzle and employs backtracking to find solutions.

## 2 Data Structures Implementation

### 2.1 Vectors

The Sudoku grid is represented using a `std::vector<std::vector<int>>`, a dynamic 2D array. This structure allows efficient indexing and iteration, with each cell storing an integer (0 for empty, 1-9 for filled). The vector is initialized as:

```
std::vector<std::vector<int>> grid(9, std::vector<int>(9, 0));
```

Vectors provide $O(1)$ access time and are flexible for grid manipulation.

### 2.2 Unordered Sets and Maps

To enforce Sudoku constraints, we use `std::unordered_set` within `std::unordered_map` to track numbers in rows, columns, and 3x3 boxes. For example:

```
unordered_map<int, unordered_set<int>> rowConstraints;
```

Unordered sets offer $O(1)$ average-case lookup and insertion, making constraint checking efficient.

## 3 Functional Demonstration

The implemented code allows users to input a puzzle manually, validates the input, and solves it using backtracking. The `displayGrid` function visually represents the grid, with dots for empty cells and numbers for filled ones. The `validateGrid` function ensures no duplicate numbers in rows, columns, or boxes. A sample run shows the grid before and after solving.

## 4 Efficiency Analysis

- **Time Complexity**:
  - Grid access: $O(1)$ per cell.
  - Constraint checking: $O(1)$ average case per lookup.
  - Backtracking: $O(9^n)$ worst case, where $n$ is the number of empty cells.
- **Space Complexity**:
  - Grid: $O(81)$ for the 9x9 vector.
  - Constraints: $O(81)$ for sets storing at most 9 numbers per row/column/box.

## 5 Application of Class Concepts

This implementation applies key concepts from the course:

- **Dynamic Arrays**: Vectors for flexible grid storage.
- **Hashing**: Unordered sets/maps for fast constraint lookups.
- **Recursion**: Backtracking to explore solution space.

## 6 Conclusion

The initial implementation successfully demonstrates the use of vectors and unordered sets/maps in a Sudoku solver. The code is functional, with efficient data structures and clear output. Future deliverables will enhance the solver with optimizations like MRV heuristics and file I/O.