Spreadsheet Program Report

Your Name

March 2, 2025

1 Introduction

The spreadsheet application supports both simple and advanced formulas. This is implemented through dependencies and dependents for cell structs. The program maintains a separate list of advanced formulas. This report describes the design decisions behind this approach, the program structure, the challenges faced during development, the data structures used for various operations, and how various edge cases are handled.

2 Design Decisions

2.1 Advanced Formulas List

Instead of storing every cell within a range as a dependency, the program maintains a dedicated list for advanced formulas (defined as a member of the spreadsheet struct). We store only the endpoints of the range and iterate over this global structure to check which ranges include our updated cell. This design has several advantages:

- **Memory Efficiency:** Large ranges do not lead to a massive dependency list.
- **Performance:** Recalculations are performed by processing only the advanced formulas in the list via topological sorting, thus recalculating a cell at most once.

2.2 Dependency Management

Simple formulas manage dependencies via AVL trees. Each cell maintains two trees:

- **Dependencies:** The cells on which the current cell depends.
- **Dependents:** The cells that depend on the current cell.

These trees ensure efficient lookup, insertion, and deletion operations.

2.3 Recalculation Process

Recalculation is performed in two stages:

1. **Local Recalculation:** The function `recalc_cell()` computes the cell's new value based on its operation (arithmetic or advanced formula).
2. **Propagation:** `recalcUsingTopoOrder()` propagates the change to dependent cells in topologically sorted order. This ensures that each cell is updated only after all its dependencies have been recalculated.

3 Challenges Faced

After finalizing the design decisions, several challenges were encountered during implementation:

1. **Memory Overhead:** The normal approach of storing every cell in the dependency list for range functions consumed a large amount of memory. To address this, we moved to using a dedicated global structure (the advanced formulas list) that stores only the endpoints of the range.
2. **Cycle Detection Efficiency:** Implementing cycle detection was challenging because, with the global advanced formulas structure, not all cells were stored as dependencies. The normal DFS approach for checking cycles in long dependency chains proved too slow. We therefore switched to a BFS approach, which was better suited for this design and improved performance significantly.

4 Data Structures Used for Various Operations

The efficiency and robustness of our spreadsheet application are driven by carefully selected data structures:

- **AVL Trees:**
  - Used for managing dependencies and dependents for each cell.
  - They provide balanced search trees for fast insertions, deletions, and lookups.
- **Linked Lists (Queues):**
  - Employed in the BFS-based cycle detection algorithm.
  - These queues help traverse the dependency graph efficiently during cycle checks and topological ordering.
- **Arrays:**
  - The advanced formulas list is maintained as a dynamic array, which stores pointers to cells that contain advanced formulas.
  - This structure provides quick access and iteration when recalculating advanced formulas.
- **Contiguous Memory Blocks:**
  - The spreadsheet table is allocated in one contiguous block, improving cache locality and performance during recalculations.

5 Program Structure and Key Components

The application is organized into several modules:

- **cell.c/h:** Contains the `Cell` structure and functions for initializing and freeing cells.
- **spreadsheet.c/h:** Implements the spreadsheet logic including user input parsing, cell recalculation, and management of advanced formulas.
- **avl_tree.c/h:** Provides an AVL tree implementation for managing dependency relationships.
- **input_parser.c/h:** Handles parsing of user commands.
- **scrolling.c/h:** Implements scrolling for displaying a portion of the spreadsheet.

Key functions related to the advanced formulas list include:

- `addAdvancedFormula()`: Adds a cell to the advanced formulas list.
- `removeAdvancedFormula()`: Removes a cell from the list.
- `recalcAllAdvancedFormulas()`: Recalculates all advanced formula cells using topological sorting.

6 Architecture Diagram

Figure 1 illustrates the high-level architecture of the application, highlighting the use of a dedicated advanced formulas list.

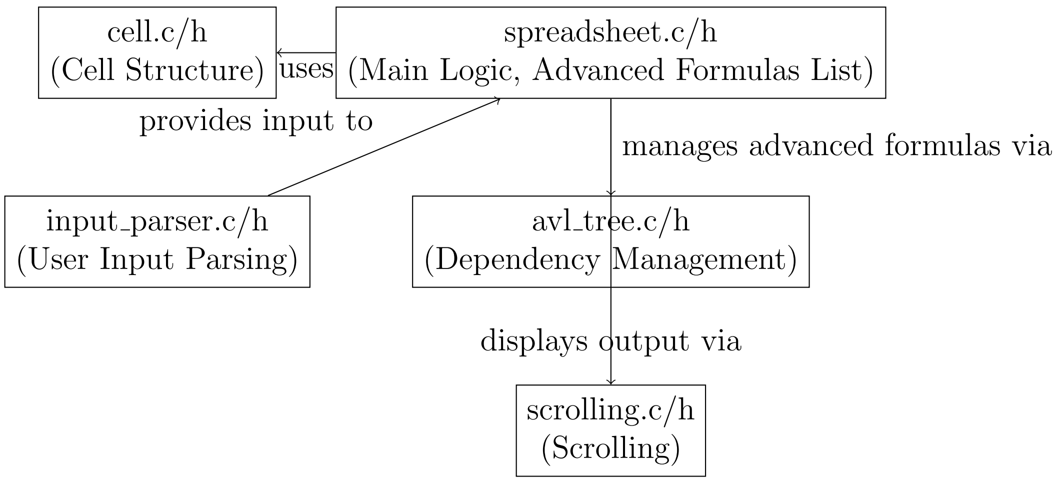

Figure 1: Software Architecture Diagram

7 Error Scenarios and Edge Cases

The system handles several error and edge-case scenarios:

- **Cyclic Dependencies:** DFS and BFS routines detect cycles and prevent operations that would create them.

- **Division by Zero:** Cells performing division by zero are marked with an error.
- **Invalid Input Formats:** The parser in `handleOperation()` validates commands and flags malformed inputs.
- **SLEEP Function:** A negative value passed to the SLEEP function results in the cell storing the negative value while executing no delay (display time is set to 0.0).
- **Direct Assignment:** When a constant is assigned to a cell (e.g., `A1=2`), any stale dependency pointers are cleared so that previous errors are removed.

8 Testing Strategy

- Our testing strategy relied on a comprehensive test suite that generated a large number of commands to verify the correct recalculation and cycle detection.
- The output of the spreadsheet was compared with expected values to ensure accuracy.

9 Demo and Repository Links

- **Video Demo:** `https://yourvideodemo.link`
- **GitHub Repository:** `https://github.com/mayankgoel2005/lab1_2023CS10204_2023CS10186_2023CS10076`

10 Conclusion

The spreadsheet application is designed to handle both simple and advanced formulas efficiently. By maintaining a dedicated advanced formulas list, the program avoids the overhead of storing every cell in a range as a dependency. The design, supported by AVL trees for dependency management and a robust recalculation mechanism based on topological sorting, ensures that even complex dependency chains are updated correctly. This document has detailed the structure of the program, key functions, and error-handling strategies.