COP290
C-LAB

Submitted by:

Savya Goel	Harshit Kansal	Varun Subramaniam
2023CS50115	2023CS10498	2023CS50497

Department of Computer Science and Engineering
IIT Delhi

---

1 Design Decisions

1.1 Data Structures and Algorithms

Step 1: Sheet Representation and Dependency Tracking
The spreadsheet is represented using a 2D dynamic array. For tracking dependencies among cells, an adjacency structure based on AVL trees is used. Each cell maintains its dependency in a separate AVL tree . The AVL tree implementation optimizes the time complexity such that insertion, deletion, and lookup operations have an overall complexity of $O(\log n)$. In addition, we used bitfields to store the operation type along with the dependent cell coordinates in a memory-efficient way.

Step 2: Recalculation Strategy
To propagate updates efficiently, a topological sort-based recalculation is implemented. Functions like `toposort` and `recalculate` traverse the dependency graph ensuring that cells are updated in the correct order – i.e., every cell is recalculated only after all cells it depends on have been updated.

2 Challenges Faced

The main challenge was handling large range operations. For example, the command `A1=SUM(A2:ZZZ999)` involves almost 18000000 cells. The challenge was because of the large number of memory allocations and AVL tree insertions required and the cycle detection in the dependency graph. Also, we had to convert recursive implementations to iterative implementations to avoiding stack memory to be exceeded and the program to crash. This required a lot of time in debugging.

3 Structure of the Program

3.1 Modules and Interactions

The program is modularized into the following key files:

- **main.c**: Controls the program flow; initializes the sheet; reads input from the user; times execution and prints output; and calls other modules for processing.
- **input_process.c**: Parses user input and distinguishes among control commands, value assignments, arithmetic expressions, and function assignments.
- **sheet.c**: Manages the spreadsheet state including coordinate conversions. It provides routines to print the sheet and executes commands.
- **functions.c**: Implements various range functions (e.g., `MIN`, `MAX`, `SUM`, `AVG`, `STDEV`) as well as sleep functionality.
- **recalculations.c**: Handles dependency management using AVL trees, cycle detection (`has_cycle`), and the propagation of updates via topological sorting. All updates to dependent cells are scheduled here.

Execution Flow:
The execution flow begins with input read by `main.c`, moves to parsing in `input_process.c`, and then branches to processing commands in either `sheet.c` or `functions.c`. Any changes that affect other cells trigger dependency updates in `recalculations.c` and finally, the updated sheet is printed as output.

4 Test Suite

5 Diagram of Software

A high-level flowchart of the program is outlined below:

Input Block	`main.c` reads user commands and sends them to `process_input`
Processing Block	`input_process.c` splits the command into control, assignment, arithmetic, or function types and calls the respective module
Dependency Block	`recalculations.c` updates the dependency graph (AVL trees) and recalculates affected cells using topological sort
Output Block	`sheet.c` prints the updated sheet via `print_sheet`

This modular structure and clear separation of responsibilities enhance both the maintainability and scalability of the program.

6 Conclusion

In summary, the program implements a spreadsheet application with arithmetic and range operations. We implemented the sheet using dynamic arrays, dependency management using AVL trees, and topological sort for recalculation in order to achieve efficiency and scalability even for very large inputs. While there were issues with performance and memory management during large operations, they were resolved using iterative implementations and the use of explicit stacks.