COP290 C Lab Report

2023CS10489, 2023CS10138, 2023CS50678

1 Introduction

This report details the design, implementation, and testing of a spreadsheet program that efficiently processes user commands to update and retrieve cell values. The spreadsheet supports arithmetic operations, statistical functions, and dependency tracking. It employs a hash table to optimize storage and lookup operations, ensuring efficient computations for large datasets.

2 Design Decisions

2.1 Data Structure: Hash Table

The spreadsheet utilizes a hash table for managing cell values dynamically, replacing a traditional 2D array. The key-value pair structure is as follows:

- **Key**: A unique combination of row and column indices.
- **Value**: A struct storing the cell's computed value, dependencies, and error status.

2.2 Hashing Strategy

To minimize collisions and ensure uniform distribution, we used:

- The unique address of the cell pointer is used as a key with the hash function key $\bmod$ table_size, where the table size allows dynamic reallocation whenever the load factor exceeds 0.5.
- Collision Handling: Linear probing is used to resolve collisions. When a collision occurs (i.e., a key maps to an already occupied index), the algorithm searches for the next available slot in a sequential manner until an empty position is found. This approach ensures that all entries remain within the hash table without requiring additional memory allocation.

3 Challenges Faced

3.1 Efficient Dependency Tracking

When a cell contains a formula (e.g., ‘C1=A1+B1‘), updating one cell triggers recalculations of all dependent cells. We implemented a dependency graph where:

- Each cell stores its precedents (cells it depends on).
- Each cell maintains a list of dependents that need recalculations upon change.

3.2 Handling Cyclic Dependencies

Cyclic dependencies (e.g., ‘A1 = B1 + 1, B1 = A1 - 1‘) can cause infinite loops. We implemented DFS-based cycle detection in the dependency graph. If a cycle is found, the operation is rejected, and an error message is displayed.

3.3 Managing Hash Collisions

Although a good hash function minimizes collisions, they still occur. Chaining with linked lists ensures collisions do not impact performance.

3.4 Parsing and Evaluating Complex Expressions

The spreadsheet supports operators (‘+‘, ‘-‘, ‘*‘, ‘/‘) and statistical functions (‘MIN‘, ‘MAX‘, ‘AVG‘, ‘SUM‘, ‘STDEV‘). This required:

- Tokenization: Breaking the input into meaningful tokens.

3.5 Ensuring Fast Lookups for Large Datasets

The hash table ensures O(1) average-time lookups, making updates and searches scalable for grids as large as ‘ZZZ999‘.

4 Program Structure

The program consists of several key files, each responsible for different aspects of the spreadsheet’s functionality:

- **driver.c**: This file acts as the main entry point of the program, handling user inputs and executing commands. It interprets various operations, such as assigning values to cells, performing arithmetic operations, navigating through the spreadsheet, and quitting the program. Upon receiving an input, it delegates tasks to the appropriate modules, ensuring that updates and recalculations are performed correctly.
- **dependency_graph_final.c / dependency_graph_final.h**: These files handle cell dependencies and automatic recalculations. Since many cells contain formulas referring to others, the program maintains a directed acyclic graph (DAG) to track dependencies. When a cell’s value is modified, this module ensures that all dependent cells are updated. Additionally, it includes cycle detection to prevent invalid calculations caused by circular references.
- **hash_table.c / hash_table.h**: These files implement the hash table used for storing spreadsheet data. Instead of maintaining a large 2D array, the program dynamically stores only non-empty cells in the hash table. Each cell’s row and column indices are hashed into a unique key, allowing O(1) average-time lookups, insertions, and deletions. Linear probing is used to resolve hash collisions efficiently.
- **input.c / input.h**: These files manage parsing and processing of user expressions. They handle direct assignments (A1=10), formulas (B1=A1+5), and functions (SUM(A1:A10)). The parser ensures correct syntax and updates the dependency graph when a formula is assigned. It also includes error handling mechanisms for invalid expressions.
- **cell.h**: This header file defines the structure of a spreadsheet cell. Each cell contains its computed value, an operation identifier indicating whether it holds a formula, and hash table of precedents (cells it depends on) and dependents (cells that rely on it). Additionally, an error flag is included to indicate invalid computations, such as division by zero.

5 Software Architecture

The diagram below illustrates the software structure:

Click here to view the image: Software Architecture Diagram

6 Edge Cases and Error Handling

6.1 Error Scenarios

The spreadsheet system must handle various error conditions to ensure correctness and prevent crashes. The following scenarios are explicitly handled:

- **Division by Zero**: Expressions such as ‘A1/0‘ result in an error message instead of an invalid computation or crash. This prevents undefined numerical operations from propagating through dependent cells.
- **Cyclic Dependencies**: If a circular dependency is introduced, such as:

$$A1 = B1 + 1, \quad B1 = A1 - 1$$

  the system detects and rejects the assignment before execution. This prevents infinite loops and incorrect calculations.
- **Invalid Syntax**: Commands with incorrect syntax, such as ‘SUM(A1:A2‘ (missing closing parenthesis) or ‘D1=*‘ (an incomplete expression), return syntax errors instead of being evaluated incorrectly.
- **Missing Cell References**: If a formula references an undefined cell, such as ‘C1 = A1 + B1‘ when ‘B1‘ has not been assigned a value, the system correctly handles the case instead of assuming a default value or leading to an invalid operation.
- **Undefined Operations**: Expressions containing unsupported or ambiguous operators, such as ‘F1=INVALID INPUT‘, are detected and flagged as errors to prevent unintended behavior.
- **Unmatched Parentheses**: Functions such as ‘SUM(A1:A2‘ without a closing parenthesis or incorrectly formatted expressions lead to an error message rather than an incomplete evaluation.
- **Out-of-Bounds Access**: Assignments or references to non-existent cells, such as ‘ZZZ1000‘, are detected, and the system prevents memory access violations.
- **Sleep Function Handling**: The ‘SLEEP(n)‘ function introduces a delay in execution. If an invalid value (negative or non-numeric) is provided, the system flags it as an error instead of attempting execution.

6.2 Test Suite Coverage

- **Basic Assignments**: The system successfully evaluates simple assignments such as ‘A1=10‘ and ‘B1=A1+5‘, ensuring that numerical values are correctly stored and retrieved.
- **Arithmetic Operations**: The test suite includes addition, subtraction, multiplication, and division, verifying that expressions like ‘C1=A1*A2‘ and ‘D1=A1/A2‘ yield correct results.
- **Statistical Functions**: The spreadsheet supports aggregate functions such as ‘SUM‘, ‘AVG‘, ‘STDEV‘, ‘MAX‘, and ‘MIN‘. These operations are tested across single-cell values and cell ranges, such as ‘SUM(A1:A6)‘.
- **Error Cases**: The test suite includes cases that trigger error handling, such as:
  - ‘A1=B1/0‘ (division by zero)
  - ‘A1=SUM(A1:A2‘ (syntax error)

  - ‘B2=MIN(A1:F1)‘ where ‘F1‘ is out of bounds

- **Dependency and Recalculation Handling**: Cases where one cell’s value depends on others (‘C6=SUM(B3:F3)‘, ‘D5=MAX(B3:C6)‘) are tested to confirm correct recalculations.
- **Sleep Function Timing**: The execution delay introduced by ‘SLEEP(n)‘ is validated to ensure that dependent cells update only after the specified duration.
- **Invalid and Corrupt Inputs**: The spreadsheet is tested against malformed expressions, missing values, and non-numeric inputs to verify robust error detection.
- **Scrolling and Display**: The `scroll_to` command, such as `scroll_to C3`, ensures that the viewport updates correctly, allowing users to navigate efficiently through large spreadsheets without losing track of the displayed data.

7 Demo and Code Repository

- **Demo Video:** `https://drive.google.com/file/d/1JUq9z9THuBc3g5gOOdYhEx8XA0ikjaST/view?usp=drive_link`
- **GitHub Repository:** `https://github.com/roymustang12/cop290_private/tree/main`