**Advanced Data Structures and Algorithms (ADSA)**

**Spreadsheet Problem**

Name: **Aneri Sheth**

Roll no. : **1401072**

Course Instructor: **Prof. Ratnik Gandhi**

**Scenario**

Consider a Google spreadsheet or Microsoft Excel File with lots of data entries. Each data entry (cell) is either a number or a formula made from those cells (for example, cell B1 = A1 + A7). Any update in a particular cell will affect the other cells directly or indirectly.

**Objective**

To design, implement and analyze this problem and suggest/propose an optimal algorithm for the same.

**Design**

When a cell is updated, all the other cells dependent on the updated cell get updated. Thus, there is a dependency among various cells. This can be interpreted as a Dependency Tree. To implement this problem, we must first understand the design tools which are Directed Acyclic Graph, Topological Sorting and Depth First Search.

Directed Acyclic Graph (DAG)

This is a type of graph where there are no directed cycles. This means that for a node x with dependencies y or z, there is no way for the dependency to loop back to x.

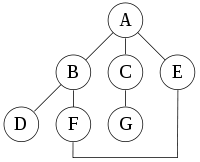
For this spreadsheet problem, the order of update should be such that the dependent cell should always be after the updated cell. This particular ordering is called Topological Ordering.

Topological Sorting

In this, for every node (Ci,Cj) , Cj should always come after Ci. Topological sorting in this case will be performed using Depth-first search algorithm.

Depth-first search (DFS)

Consider the following example:



Starting from node A, with dependencies B, C and E it randomly decides a node. Say it chooses B. Then again it randomly decides node D or F. It decides D and ends there because D has no further dependencies. Similarly, it does this for all the nodes and the order becomes: A, B, D, F, E, C, G

**Data Structures**

Directed Acyclic Graph – Used for the dependency solving.

Stack – For editing/updating the cells.

**Implementation**

As discussed in the Design part, we use Depth-first Search to implement topological sorting of the cells.

* Topological sorting can be implemented as a stack. The first cell is the top element of the cell and the last cell is the last element of the stack
* Any cell is pushed to the stack only when all the neighbors are visited. After the depth first search has generated a sequence, this sorting will be implemented. Also, if a cell is pushed only when all the neighbors are visited, the neighbors are already in the stack. Thus the topological sorting is achieved
* Once topological sorting is done, the cells are popped out and formulae is evaluated

**Code**

The code has already been uploaded on GitHub.

Below is the example of how logically this will be implemented in a 2x2 matrix.

The following can be implemented for nxn matrix within the code.

