Advanced Data Structure and Algorithms

Assignment 1

Jay Joshi

10/8/2016 1401005

Problem Statement

Let M be an n×n matrix (think of Microsoft Excel file) with following structure

• Each cell of M is indexed by iσ, where $i \in N \setminus \{0\}$ and σ is an alphabet or set of alphabets.

• Every cell iσ is either a constant number or a formula whose variables are other cells of M.

Design, implement and analyse a space and time efficient algorithm and necessary data structure for the cells of M such that when a particular cell $I_0\ \sigma_0$ is updated there is least amount of computation required to update the cells depending on $I_0\ \sigma_0$ and thus producing an efficient matrix update mechanism.

Clearly list down approach(s), required function(s) with inputs and outputs.

Make necessary assumptions (Size of the matrix: 100,000× 100,000, having at most 10,000 formulas and every formula has at most 15 variables) for the same.

Understanding:-

- Append a spreadsheet with minimum space and time complexity.
- Spreadsheet is designed in a way such that each cell has variables and formula assigned with it. There are total 15 variables that defines a formula. Change in one variable should change the value of its dependent nodes with minimum traversing and memory allocations.

Possible processing approaches :-

1) Lazy Propagation in segment tree
2) Shunting Yard Algorithm

- **Sample spreadsheet :-**

	A	B	C	D	E	F	1,00,000
1							
2							
3							
4							
5							
1,00,000						…….	100000 × 100000

- **Pre-processing :- Adjacency List**

- At the time of insertion of value and formula make a adjacency list.
- It gives an edge in time and space complexity over adjacency matrix with **Efficiency of O(n)** relative to $O(n^2)$ for adjacency matrix.
- Root of adjacency list are those 15 root variables which can cause change in value of other cells.
- EXAMPLE :-

Root variable	Dependent cells
1	A1,B4,C3,D9
2	A8,G4,E9,N7
3	A1,R4,V3,L3
4	K1,B4,J7,C9
5	A1,B4,U3,M9
6	A1,S4,T3,D9

- Root node is connected to its daughter nodes by dedicated edges. Thus, to traverse through the parsed tree would take time as function of number of root nodes.

Processing approaches :-

Step 1 :-

Lazy Propagation Approach :- A Junction Tree Inference Algorithm

- When there are many updates and updates are done on a range, we can postpone some updates (avoid recursive calls in update) and do those updates only when required.
- Thus**, only required cells will be updated through this approach**.

How it works :

- It traverses through all the cells once. Checks formula for each cell and assigns a flag (you can say it calls a cell 'dirty' if an update is required in that cell).
- Then during updation it only goes through those cells which requires updation.

Eg. : Suppose Cell C1 , D4, G5 needs update and G5 is dependent on C1 and D4.

Algorithm : How to find a reachable variable in a cell

Let there be an adjacency list of variables where each variable is connected other cells via edges. To determine the set of variables R connected to J do:

1. Set R = 0;
2. Make a tree of dependent variables connected by edges.
3. Check if value of cell is not constant but a formula
4. Mark that cell as 1 (flag == 1)
5. Return R.
6. When update is made in a cell, parse through all the connected parent node on which value of cell is dependent . Update only those cells which requires update instead of traversing the whole 10^{10} cells.

   Structure of tree is of specialized binary tree that avoids unnecessary recursive programming .

   Time complexity = O(nlogn)

Step 2 : Calculating / updating the value :-

- **Shunting yard algorithm** : A stack based approach

- It converts formula from infix form to Reverse polish notation.

- **How is it edgy ?**

- Instead of solving a formula by storing operators in nodes , it stores variable in its nodes.
- Each variable containing node is connected to other variable containing node by weighted edge having operational operator defining relation between those nodes.
- On updating value of a variable , formula does not change, it only updates the final result by applying the same equation.

Infix notation	**Reverse Polish Notation**
A + B	A B +
A ^ 2 + 2 * A * B + B ^ 2	A 2 ^ 2 A * B * + B 2 ^ +
((1 + 2) / 3) ^ 4	1 2 + 3 / 4 ^
(1 + 2) * (3 / 4) ^ (5 + 6)	1 2 + 3 4 / 5 6 + ^ *

- **Conclusion :-**

- These algorithms though rare but gives optimal output to spreadsheet implementation.
- Efficiency of pre-processing and processing algorithms provide accurate traversing and updating of cells for large number of cells of order of 10^{10}.