Spread Sheet: Algorithm and Analysis

Roll No: 1401117 Name: Kishan Raval

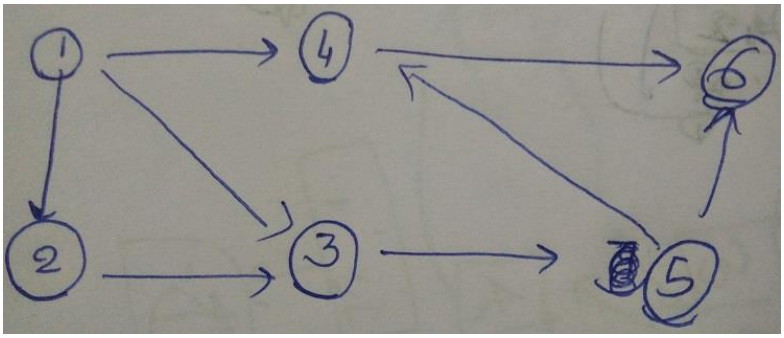
Spread-sheet contains ArrayList of cells (cell is represented by Node), which will store data corresponding to that cell, i.e. either constant values or String that contains formula. Address of each node can be obtained by given cell location using **Hashing** as each cell address is unique.

Node:

- Node represents each cell in the spreadsheet.
- It contains
  - A String which stores data
  - A list which stores Node (address) of those nodes who are dependent on this node

```
public class Node
{
    private String data;
    private ArrayList<Node> dependent;
```

E.g. for given graph below, 1, 2,…, 6 are Node and dependent Node list for Node 5 is [Node4, Node6].



Creating a new Node:

1. Get data string which contains formula, say ***f***
2. For all nodes listed in ***f***, say Node ***i***
   - Find address of Node ***i*** using Hash map
   - Add ***f*** to dependent list of ***I***

This would cost **O(number of dependent node)** as step-2 would run for all nodes on which new node is dependent. Here, number of dependent node cannot exceed 15.

Updating a node:

1. Get old data string which contains formula, say ***f1***
2. Get old data string which contains formula, say ***f2***

3. For all nodes listed in ***f2-f1***, say Node ***i*** (here, ***f2-f1*** is used for cells which are not in ***f1***, but present in ***f2)***
    - Find address of Node ***i*** using Hash map
    - Add ***f*** to dependency list of i
4. For all nodes listed in ***f1-f2***, say Node ***p*** (here, ***f2-f1*** is used for cells which are not in ***f1***, but present in ***f2)***
    - Find address of Node ***p*** using Hash map
    - Remove the Node from list of dependent Node of ***p***
5. Evaluate expression
6. Call updateNode method (described below)

Deleting a Node:

1. Get data string which contains formula, say ***f***
2. For all nodes listed in ***f***, say Node ***i***
    - Find address of Node ***i*** using Hash map
    - Remove Node ***f*** from list of dependent Node of ***i***
3. Call updateNode method (described below)

Evaluate Expression:

1. Get Expression String as a parameter: say ***exp***
2. Replace all cell address with respective cell value from ***exp***. Cell value can be obtained by extracting cell address one by one, finding it's address using HashMap and getting value from the Node address
3. Evaluate infix expression using Stack
   (Reference: http://faculty.cs.niu.edu/~hutchins/csci241/eval.htm)

getParentDependent method:

This method returns ArrayList of Nodes on Which given node is dependent

1. Get Node as parameter: say ***currNode***
2. Create an empty ArrayList of distinct node
3. Find all Cell Address one by one and add respective cell's Node address to ArrayList. Finding cell's Node address can be done by HashMap.
4. Return ArrayList

UpdateNode method (without using parallel List):

- Get address of node as parameter : say ***upNode***
- Create a queue which will store Nodes which are remaining to update
- Add ***upNode*** to the queue
- while(queue != null) //Run till all dependent Nodes are updated
- {
- Node t = queue.dequeue(); //Taking an element
- foreach (Node i = t.getParentDependent())
- { //If parentDependent node is not updated
- if(queue.exists(i)) //Then, wait for parent's upudation
- { //by again entering the queue
- queue.enqueue(t);
- t = null;
- break;
- }
- }
- if(t != null) //If all parentDependents are updated,
- { //then update the node
- update(t);
- foreach (Node i = t.getDependent()) //start updating its child nodes
- {
- queue.enqueue(i);
- }
- }
- }

Limitation: No cycles are allowed here. If such case happens, then the dead loop will occur.

UpdateNode method (By adding a new Variable to Node):

We add a new boolean variable to the Node which stores Nodes and checks whether the Node is been updated or not. While enqueuing the node, we make this Boolean variable false. And after updating, we make it true. Now, new Node structure is:

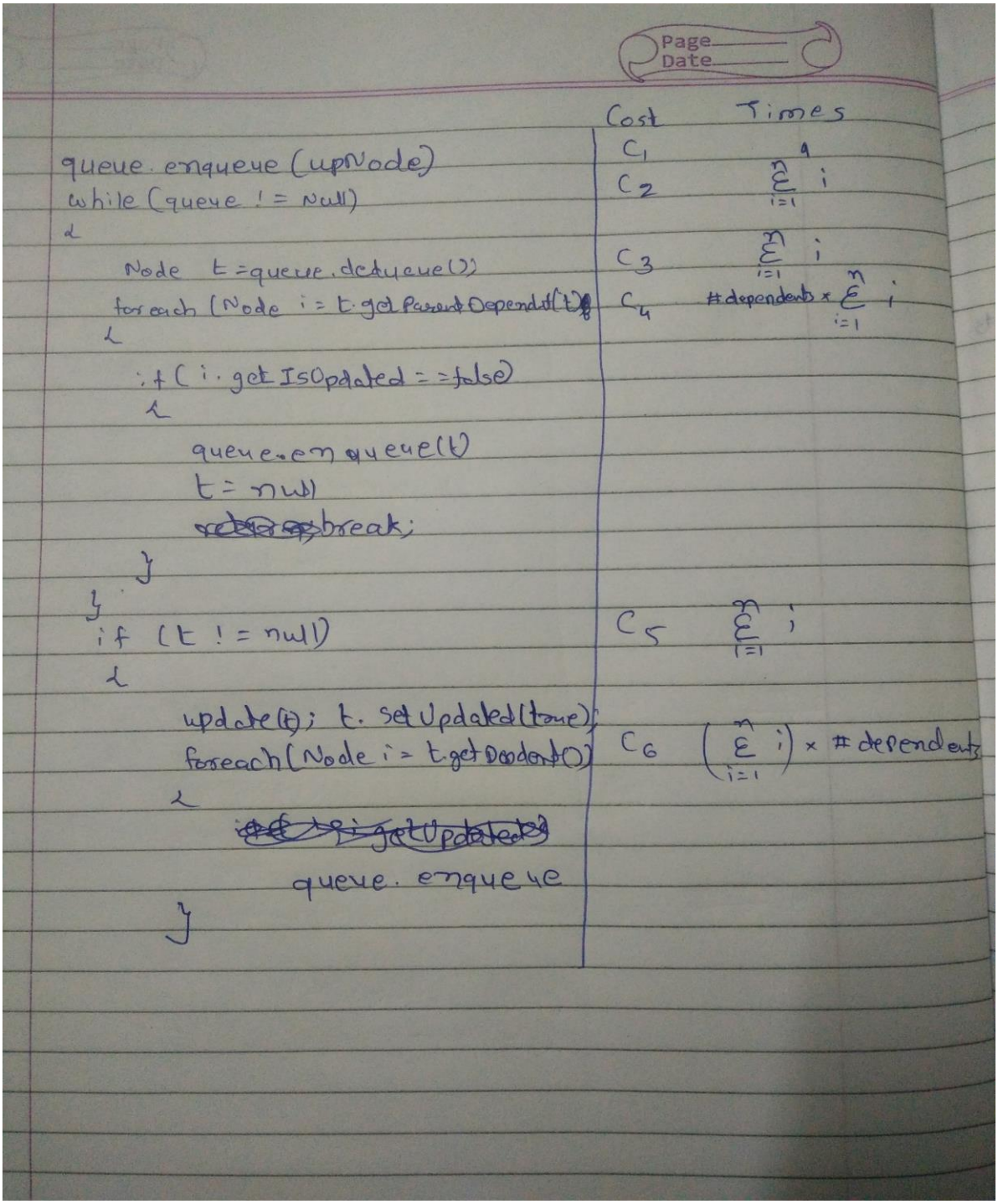
```
public class Node
{
    private String data;
    private ArrayList<Node> dependent;
    private boolean isUpdated;
```

- Get address of node as parameter : say ***upNode***

- Create a queue which will store Nodes which are remaining to update
- Add ***upNode*** to the queue and make upNode.isUpdated = false;
- while(queue != null)
- {
- Node t = queue.dequeue();
- foreach (Node i = t.getParentDependent())
- {
- if(i.getIsUpdated == false) //This will cost O(1)
- {
- queue.enqueue(t);
- t = null;
- break;
- }
- }
- if(t != null)
- {
- update(t);
- t.setIsUpdated(true);
- foreach(Node i = t.getDependent())
- {
- queue.enqueue(i);
- }
- }
- }

[P.T.O.]

Analysis



```
queue.enqueue(upNode)                                   Cost    Times
while (queue != null)                                   C1      n+1
{                                                       C2      Σ_{i=1}^{n} i
    Node t = queue.dequeue()                            C3      Σ_{i=1}^{n} i
    foreach (Node i = t.getParentDependent(t)           C4      #dependents × Σ_{i=1}^{n} i
    {
        if (i.getIsUpdated == false)
        {
            queue.enqueue(t)
            t = null
            break;
        }
    }
    if (t != null)                                      C5      Σ_{i=1}^{n} i
    {
        update(t); t.setUpdated(true);
        foreach(Node i = t.getDependent())              C6      (Σ_{i=1}^{n} i) × #dependents
        {
            i.getUpdated()
            queue.enqueue
        }
```

$$T(n) = C_1(1) + C_2 \times \sum_{i=1}^{n} i + C_3 \sum_{i=1}^{n} i$$

$$+ C_4 * \#\text{dependents} * \sum_{i=1}^{n} i$$

$$+ C_5 \sum_{i=1}^{n} i + C_6 \left(\sum_{i=1}^{n} i\right) * \#\text{dependents}$$

$$= C_1 + \sum_{i=1}^{n} i \left(C_2 + C_3 + C_4 + C_5 + C_6\right)$$

$$+ \sum_{i=1}^{n} i * \#\text{dependencies} \left(C_4 + C_6\right)$$

$$= C_1 + \left(\sum_{i=1}^{n} i\right)(C_7) +$$

$$C_8 \left(\sum_{i=1}^{n} i\right) * \#\text{dependencies}$$

$$= C_1 + C_9 \left(\sum_{i=1}^{n} i\right)$$

$$= C_1 + C_9 \frac{(n)(n-1)}{2}$$

$$= C_1 + C_9 \frac{n^2}{2} - C_9 \frac{n}{2}$$

$$= C_2 + C_{10} n^2 + C_{11} \frac{n}{2}$$

$$= O(n^2)$$