IDAA432C Assignment-6 (Group: 41)

Aditya Vallabh	Kiran Velumuri	Neil Syiemlieh	Tauhid Alam
IIT2016517	ITM2016009	IIT2016125	BIM2015003

Question

You are given a 2D array such that elements occur at only certain indices while the remaining array is empty. Write a program to find the location or indices of the elements present in the array by recursively dividing the matrix into 4 parts(quad1, quad2, quad3, quad4) discarding at each step the quads that are clean or empty.

I. Introduction

In this problem, we are presented with a partially filled 2D array(matrix) wherein only some of the values in the matrix are non-zero(filled) and the remaining values are zero(empty). The aim of this problem is to find the non-zero occurrences of elements in the 2D array by dividing the matrix recursively into 4 sub-matrices such that the sub-matrices not containing any non-zero elements are discarded at each step.

Using the relevant test cases, we obtain the time complexity of the algorithms for each of best, average and worst cases. Then we plot the graphs for the above mentioned cases.

II. Algorithm Design

A. Quadrants of a Matrix

Given an $m \times n$ matrix, the upper-left quadrant of the matrix would be a submatrix consisting of:

1) Rows: 0 to $\frac{m}{2}$
2) Columns: 0 to $\frac{n}{2}$

The other quadrants would be defined in the same fashion, with the appropriate start and end indices set.

B. Empty Quadrants

The cells of the matrix could either be filled or empty. In our implementation, a cell is considered filled if it has a value of 1 and is considered empty if it has a value of 0.

The algorithm requires the ability to test if a quadrant is empty. An empty quadrant is one in which all cells are empty. For the algorithm in question, it is assumed that this test is of $O(1)$ time. This assumption has to be made, or we'd be distracted from the analysis of the main algorithm.

For reality's sake, our implementation of this test could not be performed in $O(1)$ time. However, we've made our measurements for the time taken considering this to be of $O(1)$ complexity. The test's pseudo code is given below.

Algorithm 1 isEmpty Method

```
Input: matrix, a, b, c, d
for i ← a to c do
    for i ← b to d do
        if matrix[i][j] ≠ 0 then return 0
return 1
```

C. Find Algorithm

Our algorithm has been implemented in the procedure called $Find$. The steps of the algorithm have been summarized in the steps below. Step 2 refers to the base case of the recursive function.

1) If input matrix has just one cell, go to 2. Else, go to 3.
2) If that cell is filled, print the cell's location and terminate. Else, just terminate.
3) Call $Find$ on the upper-left quadrant
4) Call $Find$ on the upper-right quadrant
5) Call $Find$ on the lower-left quadrant
6) Call $Find$ on the lower-right quadrant

Algorithm 2 find Method

```
Input: matrix, m, n, a, b, c, d
if a ≥ m or c ≥ m or b ≥ n or d ≥ n then return
if a == c and b == d then
    if matrix[a][b] ≠ 0 then
        print(a,b)
    return
if !isEmpty(matrix, a, b, c, d) then
    find(matrix, m, n, a, b, (a + c)/2, (b + d)/2)
    find(matrix, m, n, a, (b + d)/2 + 1, (a + c)/2, d)
    find(matrix, m, n, (a + c)/2 + 1, b, c, (b + d)/2)
    find(matrix, m, n, (a + c)/2 + 1, (b + d)/2 + 1, c, d)
```

III. Analysis and Discussion

A. Time Complexity

The time taken by the algorithm in the worst case can be described by the following recurrence relation, where $T(m, n)$ is the time taken to operate on an $m \times n$ matrix.

$$T(m,n) = 4\ T(\frac{m}{2}, \frac{n}{2}) + O(1)$$
$$T(1,1) = O(1)$$

Here, the $4\ T(\frac{m}{2}, \frac{n}{2})$ refers to the recurrent procedure on the four quadrants of the matrix, while $O(1)$ is the complexity of the checking if the current sub-matrix is empty. The base

case refers to the operation on a 1×1 matrix and hence takes $O(1)$ time.

We can solve the recurrence relation as follows:

$$\begin{aligned}
T(m,n) &= 4\ T(\frac{m}{2},\frac{n}{2}) + 1 \\
&= 4\ [4\ T(\frac{m}{4},\frac{n}{4}) + 1] + 1 \\
&= 16\ T(\frac{m}{4},\frac{n}{4}) + 4 + 1 \\
&= 16\ [4\ T(\frac{m}{8},\frac{n}{8}) + 1] + 4 + 1 \\
&= 64\ T(\frac{m}{8},\frac{n}{8}) + 16 + 4 + 1 \\
&\vdots \\
T(m,n) &= (2^k)^2\ T(\frac{m}{2^k},\frac{n}{2^k}) + \sum_{i=1}^{k} 4^{i-1} \\
&= (2^k)(2^k)\ T(\frac{m}{2^k},\frac{n}{2^k}) + \sum_{i=1}^{k} 4^{i-1}
\end{aligned}$$

For $T(1,1)$:

$$\begin{aligned}
&\implies (\frac{m}{2^k},\frac{n}{2^k}) = (1,1) \\
&\implies m = 2^k\ and\ n = 2^k \\
&\implies k = \log_2 n
\end{aligned}$$

Entering the value of 2^k in the equation above, we obtain the solution:

$$\begin{aligned}
T(m,n) &= mn + \sum_{i=1}^{\log_2 n} 4^{i-1} \\
&= 2mn
\end{aligned}$$

Hence, the time complexity of the algorithm is $O(m \times n)$.

The worse case occurs when all the cells of the matrix are filled, requiring a recursive call for every quadrant of every sub-matrix. The best case occurs when the entire matrix is empty. This would prevent any recursive calls in the algorithm, resulting in a time complexity of $O(1)$.

B. Space Complexity

Since no auxiliary space is used by the algorithm, the space complexity for an $m \times n$ input matrix is $O(m \times n)$.

IV. Experimental Analysis

The algorithm was run against various inputs randomly generated for values of m and n ranging from 0 to 100. The total number of computations have been plotted against the size of the input in Fig. 1.

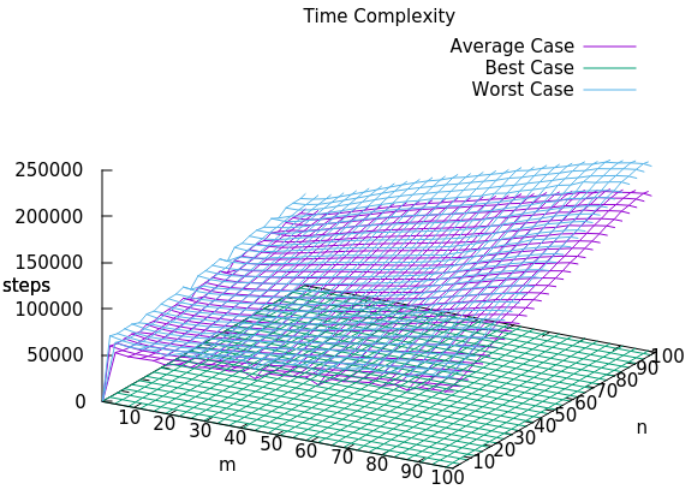

Fig. 1. Time complexity of finding filled cells with recursion

We see that the function grows in the order or $m \times n$, which is expected as the time complexity of the algorithm is $O(m \times n)$ where $m \times n$ are the dimensions of the input matrix.

We also see that the average case is closer to the worst case than it is to the best case. This is because randomly generated matrices would have filled cells that are more evenly distributed throughout the matrix. So, most of the quadrants have to be explored, almost as much as in the worst case.

V. Conclusion

Through this paper we proposed the algorithm to find the location of non-zero elements in a 2D array by recursively dividing the matrix into 4 quadrants. In this process, we discarded those quadrants which are filled only with zeros. Moreover our algorithm has both time and space complexities of $O(m \times n)$ where m, n are the dimensions of the matrix.

References

[1] Introduction to Algorithms Book by Charles E. Leiserson, Clifford Stein, Ronald Rivest, and Thomas H. Cormen