DAA Assignment 2

Nitin Raj Singh IIT2016132
Chirag Meel ISM2016006.
Lekhika Dugtal ITM2016008
Saurabh Kumar IWM2016502

1 PROBLEM

Write an efficient algorithm for the following purpose. For a given value of n (n>3), generate an (n x n) matrix whose cells are filled with randomly generated digits 0,.., 9. Now, scan the matrix with a (3 x 3) mask and find out those masks (mask positions) which contain a specific pattern (for example successive numbers are in a non-decreasing order). Note that, you can report a mask position by only mentioning the top-left position of the mask. Do the necessary experimentation and analysis with your algorithm.

2 INTRODUCTION AND LITERATURE SURVEY

In this problem we have to first generate a matrix of size n X n where n ¿ 3. The matrix should contain random values between 0 to 9. Now from the given matrix we have to search for some specific pattern. The pattern corresponds to some 3 X 3 matrix. We will have to scan through all the 3 X 3 submatrix in the given matrix and note those indexes where the particular pattern is present.

Now, the approach towards this algorithm, depends a lot on the type of pattern we are provided with. For example, we can use hashing for some kinds of patterns which would be the fastest method provided we can form some hashing function for the given pattern, another approach we can use is Depth First Search (DFS) and ofcourse the most simple one would be looping across the given pattern. So we see that we have many ways to approach this problem.

3 ALGORITHM DESIGN

3.1 Algorithm Approach

In this problem we are taking the pattern that the successive numbers in the mask should be in non-decreasing order.

The index of any 3 X 3 mask will be in the following manner.

(x, y)	(x+1, y)	(x+2, y)
(x, y+1)	(x+1, y+1)	(x+2, y+1)
(x, y+2)	(x+1, y+2)	(x+2, y+2)

So now as it is provided in the question that the successive numbers should be in the increasing order, then we can have many ways to traverse successive numbers in the mask. But whatever be the order of traversal the index of the elements in the mask will remain the same as above.

Now depending on the way of traversing, we can create two arrays, lets say nextx[] and nexty[], where nextx[] refers to the next value of the x co-ordinate and in the same way nexty[] refers to the next value of the y co-ordinate.

For example we have to travel the mask in the following way.

1	2	3
4	5	6
7	8	9

In the above mask we are traveling from 1 to 9, left to right in each row.

In this case the values of the nextx[] and nexty[] will be:-

Nexty[] = $\{0, 0, 0, 1, 1, 1, 2, 2, 2\}$
Nextx[] = $\{0, 1, 2, 0, 1, 2, 0, 1, 2\}$

Similarly, if we want to travel the mask in some other way, lets say we want to travel from left to right then right to left and at last again from left to right

1	2	3
6	5	4
7	8	9

In this mask the value of the nextx[] and nexty[] will be:-

Nexty[] = $\{0, 0, 0, 1, 1, 1, 2, 2, 2\}$
Nextx[] = $\{0, 1, 2, 2, 1, 0, 0, 1, 2\}$

3.2 Algorithm Pseudo Code

Nexty[] = {0, 0, 0, 1, 1, 1, 2, 2, 2}
Nextx[] = {0, 1, 2, 2, 1, 0, 0, 1, 2}

```
check(y, x)
  for i = 2 to 9                                   3
    Cur = mat[y+nexty[i]][x+nextx[i]]              4
    Prev = mat[y+nexty[i-1]][x+nextx[i-1]]         4
    if(Cur < Prev)                                 1
      return false                                 1
    return true                                    1
```

```
findIndex(n, m)
  for i = 1 to (n-2)                               3
    for j = 1 to (m-2)                             3
      if( Check(i, j) == true)                     m + 1
        print
      else
        continue
```

3.3 C Implmentation

```c
#include <stdio.h>
int mat[100][100];
int next_y[] = {0, 0, 0, 1, 1, 1, 2, 2, 2};
int next_x[] = {0, 1, 2, 2, 1, 0, 0, 1, 2};

int check(int starty, int startx)
{
        for(int i=1; i<=8; i++)
        {
                if(mat[starty+next_y[i]][startx + next_x[i]] < mat[starty + next_y[i-1]][startx + next_x[i-1]])
                {
                        return 0;
                }
        }
        return 1;
}

void findIndex(int n, int m)
{
        for(int i=1; i<=n-2; i++)
        {
                for(int j=1; j<=m-2; j++)
                {
                        if(check(i, j) == 1)
                        {
                                printf("%d %d\n", i, j);
                        }
                }
        }
}

int main(void) {
        int n = 5;
        int m = 5;
        for(int i=1; i<=n; i++)
        {
                for(int j=1; j<=m; j++)
                {
                        mat[i][j] = rand()%10;
                }
        }
        findIndex(n, m);
        return 0;
}
```

4 Analysis

4.1 Worst Case Analysis

Analysis of the check() function

The maximum time taken by the check function will be when th for loop in the check function loops completely

$$T \ \alpha \ 3*8+9*8+1$$

$$T \ \alpha \ 97$$

Analysis of findIndex() function

The time taken by the above function is same every time for a particular value of n.
In this case the complexity would be

$$T \ \alpha \ (n-2)*(n-2)*(4+m)+(n-2)*3$$
$$T \ \alpha \ (n-2)*\{(n-2)*(4+m)+3\}$$

As calculated above, in the worst case the value of m would be 97, putting m = 97, we get

$$T \ \alpha \ (n-2)*\{(n-2)*101+3\}$$
$$T \ \alpha \ (101n*n-401n+402)$$

4.2 Best Case Analysis

Analysis of the check() function

The minimum time taken by the check function would be when the if condition would be true in the first iteration only and false is returned.
The time complexity in this case would be

$$T \ \alpha \ 3+9+1$$

$$T \ \alpha \ 13$$

Analysis of the findIndex() function The time taken by the above function is same everytime for a particular value of n.
In this case the complexity would be

$$T \ \alpha \ (n-2)*(n-2)*(4+m)+(n-2)*3$$
$$T \ \alpha \ (n-2)*\{(n-2)*(4+m)+3\}$$

For the best case the value of m = 13, we get

$$T \ \alpha \ (n-2)*\{(n-2)*(17)+3\}$$
$$T \ \alpha \ (17n*n-65n+62)$$

We see here that the order of the algorithm is n*n but the coefficient involved with the value is very large so those coefficient would also affect the complexity to a great extent.
For very large values of n the complexity would by proportional to n*n.

In this case we see that the best case complexity and the worst case complexity evaluates to be equal.

$$O(n*n) = \Omega(n*n)$$

Thus, here we represent the complexity as

$$\theta(n*n)$$

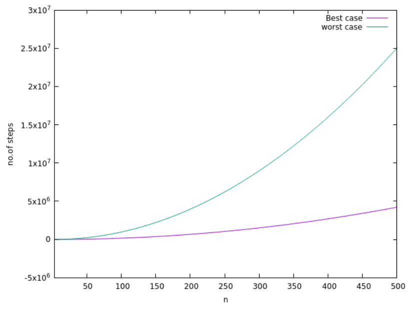
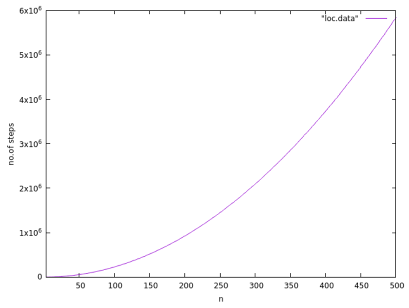


Figure 1: Theoretically calculated best and worst case complexity.

5 Experimental Study



The above graph was obtained for different values of n which had random values inserted into them and the time was calculated by taking a global variable which was increased by the number of units of time consumed by each step of the algorithm. As expected the time taken is more than the best case and less than the worst case which can be noticed by comparing it to the graph obtained theoretically shown above.

6 Discussion and Conclusion

A better approach or optimization would be that we skip a mask if at the present mask the condition for the pattern fails at the middle column i.e, at the second column element as the condition check at the second element in the present mask is a condition to be checked in the next mask but since it is false in the present mask it need not be checked in the next mask and we can skip that mask. This wouldnt reduce the overall complexity but it would reduce the factor of time complexity in case of best case.

In the approach discussed above implementation wouldnt be much different for different patterns as one could change the nextx and nexty arrays accordingly for different patterns and check for the masks as the nextx and nexty arrays indicate how the x and y in increased in the mask and the only thing that needs to be changed is the checking of the condition.

7 References

- How to solve it by Computer By R.G. Dromney
- Introduction to Algorithm 3rd Edition By Thomas H.Cormen
- overleaf.com
- tex.stackexchange.com
- latex.org