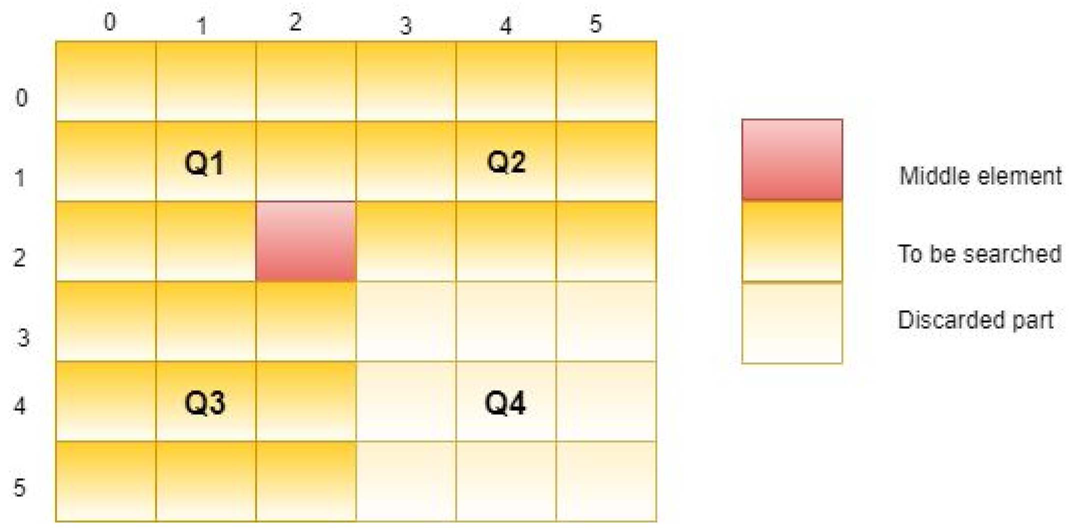
DAA

Sorted Matrix

(Find an element in order of n^(log3))

By Sai Venkatesh (IMT2017012) and George Abraham (IMT2017019)



Introduction

A matrix of size nXn is given as input. The matrix follows the below properties:

- The elements in the row of the matrix are in increasing order.
- The elements in the column of the matrix are in increasing order.

Objective

Find an element in order of n^(log3) which is given by user.

Approach

The following question can be solved by using divide and conquer approach and can be done using recursion as shown in the pseudo code below.(The approach is very similar to a binary search).

Pseudo code

R1-is starting row,c1-starting column,r2 - ending row,c2-ending column

(t)-element being searched for in matrix -A.

The position [r1,c1] and [r2,c2] give diagonally opposite points of the matrix.(Let function be called search)

Code:

```
search(r1,c1,r2,c2,t,A){

        r3=(r1+r2)/2;

        c3=(c1+c2)/2;

        if(A[r3][c3]=t) ----------------------------------------------------------------->condition 1

        {

                return r3 and c3

                (returning correct position of element in matrix)

        }

        else{
```

```
        if(A[r3][c3]>t){
            search(r1,c1,r3,c3,t,A)
            search(r1,c3,r3,c2,t,A)
            search(r3,c1,r2,c3,t,A)
        }
        if(A[r3][c3]<t){
            search(r3,c3,r2,c2,t,A)
            search(r1,c3,r3,c2,t,A)
            search(r3,c1,r2,c3,t,A)
        }
    }
}
```

Data Structures used:

A vector of size nXn as matrix.

Proof of correctness:

The approach used is divide and conquer.First we make an important observation. The row is sorted in increasing order and so are the columns.Since A[h][j]>A[i][j](by increasing order property). Similarly A[i][r]>A[i][j].So if any element A[i][j]>k where k is any number,we can say that A[h][r]is strictly greater than k where h>i and r>j. ----property(1)

By the same above logic we can conclude that if A[i][j]<k, A[h][r] is lesser than A[i][j] for all h<i and r<j. --property(2)

We can divide the matrix into four quarters(similar to quadrants) of equal size.

So first we take middle element in matrix A[n/2][n/2].If it is equal to the required element,we have found the position of the element.

If it is lesser than the required element then we can eliminate the top left quarter using the above property.-----------[Case 1]

Similarly if the middle element is greater,then we can eliminate the lower right quarter due to the above property. ---[Case 2]

So in both the cases we have to search for the element in 3 other matrices of size n/2Xn/2.

The divide part is splitting the nXn matrix into 3 n/2Xn/2 matrices and the conquer is bringing the solution together.

Let us use Induction on the number of elements in the matrix.

Base case:Matrix of size 1X1. By condition 1 in pseudo code if key is equal to element at position [0,0] it will return key otherwise it will return -999 . Hence it holds true for base case.

Induction hypothesis: Now we will assume that the algorithm search returns the position of the element in the matrix for all matrix sizes upto n-1Xn-1.So we can safely say that it works for matrices of size n/2 X n/2 too.

Need to prove that it also holds for matrix of size nXn.

We will use contradiction to prove that position of element is correctly found even after eliminating a quarter of the matrix.

Let us take

Case 1:

Assume that we have not found the element in the remaining three segments but it it exists in the top left segment(eliminated segment).It would follow that A[n/2][n/2]>key, by [property 1] but this would be a contradiction to the condition of case 1 i.e A[n/2][n/2]<key.Hence we can say that no element would be missed in Case 1.

Case 2:

A similar approach to Case 1.Assume that we have not found the element in the remaining three segments but it it exists in the bottom right segment(eliminated segment).It would follow that A[n/2][n/2]<key, by [property 1] but this would be a contradiction to the condition of case 1 i.e A[n/2][n/2]>key.Hence we can say that no element would be missed in Case 2.

Since the above arguments cover both the cases,we can conclude that no element existing in the matrix will be wrongly eliminated. It means that if our element is present in the matrix it should be present in one of the three matrices of size n/2Xn/2.

Since the search algo works correctly for matrices till size n-1 X n-1,we can conclude by the above argument that it works for matrix of size n X n.

Time Complexity:

For a matrix of size nXn (or) n^2 elements,

T(n^2)=1+3T((n^2)/4)

(n/2)^2 can go upto log(n)base 2 values

So

T(n^2)=1+3*3*.......(Log(n))T(1))

T(1) is constant

T(n^2)=3^(logn)

By property of logarithm

T(n^2)=n^log3

To find an element in nXn matrix (n^2 elements) takes θ(n^log3) time.

Read before testing code:

The code includes a random matrix generator satisfying the constraints of the question (sortedness) and only requires the input of matrix size to generate the required matrix .You have the option of printing out the matrix and selecting an element of your choice and querying for it or specifying the position of some random element in the matrix,this returns the number at that position of the matrix and you can query for it’s position .

The code returns the position of the element and if the element doesn’t exist in the matrix it prints out “number not found”.

Individual contributions:

Sai Venkatesh:

- Writing of the code.
- Running time analysis
- debugging

George:

- Algorithm.
- Proof of correctness
- Test case generation
- Debugging
- Pseudo code

6