Searching a 2D Sorted Matrix Part III

October 13, 2010 by 1337c0d3r 18 Replies

> Write an efficient algorithm that searches for a value in an *n* x *m* table (two-dimensional array). This table is sorted along the rows and columns — that is,
>
> Table[i][j] ≤ Table[i][j + 1],
> Table[i][j] ≤ Table[i + 1][j]

Note:

This is Part III of the article: Searching a 2D Sorted Matrix. Please read Part I and Part II for more background information.

Test Data Sets:

I created three test input files (Download all of them below) by myself for testing and studying purposes. Each file contains multiple test cases. The easy input contains 15 test cases, with **M** and **N** = 10, **K** = 50. The hard input contains 23 test cases, but with **M** and **N** = 100, **K** = 100,000. The performance input has **M** and **N** = 1000, and **K** = 1,000,000.

Here are the run times of the five different algorithms using the performance input data set.

```
Binary Search:             31.62s
Diagonal Binary Search:    32.46s
Step-wise Linear Search:   10.71s
Quad Partition:            17.33s
Binary Partition:          10.93s
Improved Binary Partition:  6.56s
```

As you can see from the results above, the *Improved Binary Partition* method is clearly the winner here.

Input:

Each test case starts with two integers, **M** and **N**. **M** is the number of rows and **N** is the number of columns of the matrix. The subsequent **M** lines would be the contents of the matrix in row-order. Each line would contain **N** integers separated by a single space. Following that is a blank line, with an integer, **K**, the total number of targets to be searched in the matrix. The next line contains **K** integers in one line, each of them separated by a single space. After that is a blank line. Then followed by the next case. **M**=0 and **N**=0 and **K**=0 indicates that there are no more inputs.

Output:

For each test case, output "Test case *#x*", where *x* is the test case number starting from 1. After that you should have **K** lines, each line telling if the target is found in the matrix or not. If the target is found in the matrix, output "Target *y* found at (*i*,*j*)", where *y* is the searched element, while *i* and *j* are the row and column number of the found element respectively. If the target could not be found, then output "Target *y* not found". The next test case should be separated by a blank line from the above test case.

Sample Input:

```
1 1
1

3
0 1 2

1 2
-1 3

8
-2 -1 0 1 2 3 4 5

0 0
0
```

Sample Output:

```
Test case #1
Target 0 not found
Target 1 found at (1,1)
Target 2 not found

Test case #2
Target -2 not found
Target -1 found at (1,1)
Target 0 not found
```

```
Target 1 not found
Target 2 not found
Target 3 found at (1,2)
Target 4 not found
Target 5 not found
```

Attachments:

» Download Easy Input
» Download Easy Output
» Download Hard Input
» Download Hard Output
» Download Performance Input

Further Thoughts:

A variation of this problem had been asked in Amazon interviews:

> 2D Matrix(*n* * *n*) of positive and negative numbers is given. Matrix is sorted rowwise and columnwise. You have to return the count of -ve numbers in most optimal way.

Rating: 4.8/**5** (15 votes cast)

Searching a 2D Sorted Matrix Part III, 4.8 out of 5 based on 15 ratings

← Searching a 2D Sorted Matrix Part II

Implement strstr() to Find a Substring in a String →

18 thoughts on “Searching a 2D Sorted Matrix Part III”

Chimpanzee
November 10, 2010 at 2:24 pm

Hi, what does "-ve numbers" mean? I don't quite understand.

Reply ↓ -1

1337c0d3r
November 10, 2010 at 2:26 pm