Search a 2D Matrix

Write an efficient algorithm that searches for a value in an *m* x *n* matrix. This matrix has the following properties:

- Integers in each row are sorted from left to right.
- The first integer of each row is greater than the last integer of the previous row.

For example,

Consider the following matrix:

```
[
  [1,   3,  5,  7],
  [10, 11, 16, 20],
  [23, 30, 34, 50]
]
```

Given **target** = `3`, return `true`.

Solution 1

Use binary search.

n * m matrix convert to an array => matrix[x][y] => a[x * m + y]

an array convert to n * m matrix => a[x] =>matrix[x / m][x % m];

```
class Solution {
public:
    bool searchMatrix(vector<vector<int> > &matrix, int target) {
        int n = matrix.size();
        int m = matrix[0].size();
        int l = 0, r = m * n - 1;
        while (l != r){
            int mid = (l + r - 1) >> 1;
            if (matrix[mid / m][mid % m] < target)
                l = mid + 1;
            else
                r = mid;
        }
        return matrix[r / m][r % m] == target;
    }
};
```

written by vaputa original link here

Solution 2

```
/**
 *  Do binary search in this "ordered" matrix
 */
public boolean searchMatrix(int[][] matrix, int target) {

    int row_num = matrix.length;
    int col_num = matrix[0].length;

    int begin = 0, end = row_num * col_num - 1;

    while(begin <= end){
        int mid = (begin + end) / 2;
        int mid_value = matrix[mid/col_num][mid%col_num];

        if( mid_value == target){
            return true;

        }else if(mid_value < target){
            //Should move a bit further, otherwise dead loop.
            begin = mid+1;
        }else{
            end = mid-1;
        }
    }

    return false;
}
```

written by liaison original link here

Solution 3

```
bool searchMatrix(vector<vector<int>>& matrix, int target) {
    // treat the matrix as an array, just taking care of indices
    // [0..n*m]
    // (row, col) -> row*n + col
    // i -> [i/n][i%n]
    if(matrix.empty() || matrix[0].empty())
    {
        return false;
    }
    int m = matrix.size(), n = matrix[0].size();
    int start = 0, end = m*n - 1;
    while(start <= end)
    {
        int mid = start + (end - start)/2;
        int e = matrix[mid/n][mid%n];
        if(target < e)
        {
            end = mid - 1;
        }
        else if(target > e)
        {
            start = mid + 1;
        }
        else
        {
            return true;
        }
    }
    return false;
}
```

written by xlingcc original link here