Kth smallest element in sorted matrix

This is an interview question.

Find the K^{th} smallest element in a matrix with sorted rows and columns.
Is it correct that the K^{th} smallest element is one of `a[i, j]` such as `i + j = K` ?

arrays algorithm data-structures matrix multidimensional-array

edited Mar 3 '13 at 7:38

Grijesh Chauhan
37.6k 9 72 124

asked Mar 2 '13 at 21:17

Michael
11.5k 32 99 187

how is the matrix sorted? only that in each row or column is the number increasing? – V-X Mar 2 '13 at 21:19

Yes, the numbers in each row and column are sorted in increasing order. – Michael Mar 2 '13 at 21:21

It's very easy to come up with a counterexample to show that the statement is false. – NPE Mar 2 '13 at 21:21

the solution is obviously incorrect. eg. first element can be found at the corner but the second number can be one of the two neighbors. the third may be at one of 5 possible indices. you have to employ some modification of binary search. – V-X Mar 2 '13 at 21:23

6 Answers

False.

Consider a simple matrix like this one:

```
1 3 5
2 4 6
7 8 9
```

9 is the largest (9th smallest) element. But 9 is at A[3, 3], and 3+3 != 9. (No matter what indexing convention you use, it cannot be true).

You can solve this problem in O(k log n) time by merging the rows incrementally, augmented with a heap to efficiently find the minimum element.

Basically, you put the elements of the first column into a heap and track the row they came from. At each step, you remove the minimum element from the heap and push the next element from the row it came from (if you reach the end of the row, then you don't push anything). Both removing the minimum and adding a new element cost O(log n). At the jth step, you remove the `j` th smallest element, so after `k` steps you are done for a total cost of `O(k log n)` operations (where n is the number of rows in the matrix).

For the matrix above, you initially start with `1,2,7` in the heap. You remove `1` and add `3` (since the first row is `1 3 5`) to get `2,3,7` . You remove `2` and add `4` to get `3,4,7` . Remove `3` and add `5` to get `4,5,7` . Remove `4` and add `6` to get `5,6,7` . Note that we are removing the elements in the globally sorted order. You can see that continuing this process will yield the `k` th smallest element after k iterations.

(If the matrix has more rows than columns, then operate on the columns instead to reduce the running time.)

edited Sep 23 '13 at 18:53

Trying
7,058 1 32 74

answered Mar 2 '13 at 21:28

nneonneo
99.7k 19 123 222

that is Good..give an example when matrix is a set. No repeat elements – Grijesh Chauhan Mar 2 '13 at 21:30

very trivial, done. – nneonneo Mar 2 '13 at 21:33

PLease check my answer If I did correct. With my assumptions – Grijesh Chauhan Mar 2 '13 at 21:35

@GrijeshChauhan: well, with that assumption it is right. But that assumption is too restrictive. – nneonneo Mar 2 '13 at 21:39

2 This solution works best if only row or column is sorted (essentially, it is n-way merge in external sorting). @user1987143's is better since it leverages the fact that both row and column are sorted. – sinoTrinity May 4 '15 at 22:41

|

`O(k log(k))` solution.

- Build a minheap.
- Add `(0,0)` to the heap. While, we haven't found the `kth` smallest element, remove the top element `(x,y)` from heap and add next two elements `[(x+1,y)` and `(x,y+1)]` if they haven't been visited before.

We are doing `O(k)` operations on a heap of size `O(k)` and hence the complexity.

edited Sep 23 '13 at 18:49

BartoszKP
22.5k 8 41 67

answered Sep 23 '13 at 18:18

user1987143
151 1 3

Can you give this some formatting? kind of hard to read as-is – StormeHawke Sep 23 '13 at 18:39

Are you sure this is correct? I mean even I think the same, just amazed by the no of votes received on your answer in contrast to the other, even though the complexity of your solution is better than the other. – Akashdeep Saluja Sep 27 '13 at 5:41

I think this is correct, can someone expert please confirm it ? – Harry Feb 17 '14 at 7:25

i think this is correct and the runtime is better than the accepted answer. – Meow Apr 3 '14 at 16:17

@SixinLi Not sure whether it is better than `O(klog(n))` as `0 <= k <= n^2` and k has `n-1/n` chances to be bigger than n – Jackson Tale Apr 16 '14 at 20:51

|

Start traversing the matrix from the top-left corner (0,0) and use a binary heap for storing the "frontier" - a border between a visited part of the matrix and the rest of it.

Implementation in Java:

```
private static class Cell implements Comparable<Cell> {

    private final int x;
    private final int y;
    private final int value;

    public Cell(int x, int y, int value) {
        this.x = x;
        this.y = y;
        this.value = value;
    }

    @Override
    public int compareTo(Cell that) {
        return this.value - that.value;
    }

}

private static int findMin(int[][] matrix, int k) {

    int min = matrix[0][0];

    PriorityQueue<Cell> frontier = new PriorityQueue<>();
    frontier.add(new Cell(0, 0, min));

    while (k > 1) {

        Cell poll = frontier.remove();

        if (poll.y + 1 < matrix[poll.x].length) frontier.add(new Cell(poll.x, poll.y + 1,
matrix[poll.x][poll.y + 1]));
        if (poll.x + 1 < matrix.length) frontier.add(new Cell(poll.x + 1, poll.y,
matrix[poll.x + 1][poll.y]));

        if (poll.value > min) {
            min = poll.value;
            k--;
        }

    }
```

```
    return min;

}
```

answered Apr 11 '15 at 23:55

bedrin
2,113 11 24

As people mentioned previously the easiest way is to build a `min heap` . Here's a Java implementation using PriorityQueue:

```
private int kthSmallestUsingHeap(int[][] matrix, int k) {

    int n = matrix.length;

    // This is not necessary since this is the default Int comparator behavior
    Comparator<Integer> comparator = new Comparator<Integer>() {
        @Override
        public int compare(Integer o1, Integer o2) {
            return o1 - o2;
        }
    };

    // building a minHeap
    PriorityQueue<Integer> pq = new PriorityQueue<>(n*n, comparator);
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            pq.add(matrix[i][j]);
        }
    }

    int ans = -1;
    // remove the min element k times
    for (int i = 0; i < k; i++) {
        ans = pq.poll();
    }

    return ans;
}
```

answered Sep 16 at 20:25

Yar
1,370 9 23

`k` is bounded by `n*m`. So a solution that runs in `O(k*log(k))` is worse than `O(k)` which is in fact `O(n*m)`, the trivial search.

answered Sep 27 '13 at 11:40

aUserHimself
268 1 8

1 trivial search can't do the work. although the matrix is sorted by rows and columns, overall they are not totally sorted. So if you want to map to `n*m`, you need to read them all and then sort them all, which is `n*m*Log(n*m)` – Jackson Tale Apr 16 '14 at 20:07

We don't actually need to sort to find the k-th largest then. It can be done using randomized selection. For length 'n' it runs in 'O(n)'. For length 'n*m' it can be done in 'O(n*m)'. The only issue is that by doing this, we are ignoring the sorted-ness of the matrix. – Ehsan Sep 29 '14 at 21:27

Seems this just uses the feature: every row is sorted, but not use its column-wise sorted feature.

answered Jul 22 '14 at 4:04

jeffery.yuan
328 1 7 16