geeksforgeeks.org

Kth smallest element in a row-wise and column-wise sorted 2D array | Set 1 - GeeksforGeeks

GeeksforGeeks

8-10 minutes

Given an n x n matrix, where every row and column is sorted in non-decreasing order. Find the kth smallest element in the given 2D array.

Example,

```
Input:k = 3 and array =
        10, 20, 30, 40
        15, 25, 35, 45
        24, 29, 37, 48
        32, 33, 39, 50
Output: 20
Explanation: The 3rd smallest element is 20

Input:k = 7 and array =
        10, 20, 30, 40
        15, 25, 35, 45
        24, 29, 37, 48
        32, 33, 39, 50
Output: 30

Explanation: The 7th smallest element is 30
```

Approach: So the idea is to find the kth minimum element. Each row and each column is sorted. So it can be thought as C sorted lists and the lists have to be merged into a single list, the kth element of the list has to be found out. So the approach is similar, the only difference is when the kth element is found the loop ends.

Algorithm:

1. The idea is to use min heap. Create a Min-Heap to store the elements
2. Traverse the first row from start to end and build a min heap of elements from first row. A heap entry also stores row number and column number.
3. Now Run a loop k times to extract min element from heap in each iteration
4. Get minimum element (or root) from Min-Heap.
5. Find row number and column number of the minimum element.
6. Replace root with the next element from same column and min-heapify the root.
7. Print the last extracted element, which is the kth minimum element

Implementation:

- C++
- Java
- Python3
- C#
- Javascript

C++

```
#include <bits/stdc++.h>

using namespace std;

struct HeapNode {

    int val;
    int r;

    int c;

};

void minHeapify(HeapNode harr[], int i, int heap_size)

{

    int l = i * 2 + 1;

    int r = i * 2 + 2;

     if(l < heap_size&& r<heap_size && harr[l].val <
harr[i].val && harr[r].val < harr[i].val){

            HeapNode temp=harr[r];

            harr[r]=harr[i];

            harr[i]=harr[l];

            harr[l]=temp;

            minHeapify(harr ,l,heap_size);

            minHeapify(harr ,r,heap_size);

        }

          if (l < heap_size && harr[l].val <
harr[i].val){

            HeapNode temp=harr[i];

            harr[i]=harr[l];

            harr[l]=temp;

            minHeapify(harr ,l,heap_size);

        }

}

int kthSmallest(int mat[4][4], int n, int k)

{
    if (k < 0 || k >= n * n)
        return INT_MAX;
    HeapNode harr[n];
    for (int i = 0; i < n; i++)
        harr[i] = { mat[0][i], 0, i };
    HeapNode hr;
    for (int i = 0; i < k; i++) {
        hr = harr[0];
        int nextval = (hr.r < (n - 1)) ? mat[hr.r +
1][hr.c]: INT_MAX;
        harr[0] = { nextval, (hr.r) + 1, hr.c };
        minHeapify(harr, 0, n);
    }
    return hr.val;
}
int main()
{
    int mat[4][4] = {
        { 10, 20, 30, 40 },
        { 15, 25, 35, 45 },
        { 25, 29, 37, 48 },
        { 32, 33, 39, 50 },
    };
    cout << "7th smallest element is "
        << kthSmallest(mat, 4, 7);
    return 0;
```

```
}
```

Java

Python3

C#

Javascript

Output

7th smallest element is 30

The codes above are contributed by RISHABH CHAUHAN.
Complexity Analysis:

- **Time Complexity:** The above solution involves following steps.

1. Building a min-heap which takes O(n) time
2. Heapify k times which takes O(k Logn) time.

- **Auxiliary Space:** O(R), where R is the length of a row, as the Min-Heap stores one row at a time.

The above code can be optimized to build a heap of size k when k is smaller than n. In that case, the kth smallest element must be in first k rows and k columns.
We will soon be publishing more efficient algorithms for finding the kth smallest element.
This article is compiled by Ravi Gupta. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Using inbuilt priority_queue :

By using a comparator, we can carry out custom comparison in priority_queue. We will use priority_queue<pair<int,int>> for this.

Implementation :

- C++

C++

```
#include<bits/stdc++.h>

using namespace std;

int kthSmallest(int mat[4][4], int n, int k)

{

    auto cmp = [&](pair<int,int> a,pair<int,int> b){

        return mat[a.first][a.second] > mat[b.first]
[b.second];

    };

    priority_queue<pair<int,int>,vector<pair<int,int>>,decltype(c
pq(cmp);

    for(int i=0; i<n; i++){

        pq.push({i,0});

    }

    for(int i=1; i<k; i++){

        auto p = pq.top();

        pq.pop();

        if(p.second+1 < n) pq.push({p.first,p.second +
1});

    }

    return mat[pq.top().first][pq.top().second];

}

int main()

{

    int mat[4][4] = {
        { 10, 20, 30, 40 },
        { 15, 25, 35, 45 },
        { 25, 29, 37, 48 },
        { 32, 33, 39, 50 },
    };
    cout << "7th smallest element is "
        << kthSmallest(mat, 4, 7);
    return 0;
}
```

Output

```
7th smallest element is 30
```

Time Complexity: O(n log n)

Auxiliary Space: O(n)

Using Binary Search over the Range:

This approach uses binary search to iterate over possible solutions. We know that

1. answer >= mat[0][0]
2. answer <= mat[N-1][N-1]

So we do a binary search on this range and in each iteration determine the no of elements greater than or equal to our current middle element. The elements greater than or equal to current element can be found in O(n logn) time using binary search.

- C++
- Java
- Python3
- C#

- Javascript

C++

```
#include <bits/stdc++.h>

using namespace std;

int getElementsGreaterThanOrEqual(int num, int n, int
mat[4][4]) {

    int ans = 0;

    for (int i = 0; i < n; i++) {

        if (mat[i][0] > num) {

            return ans;

        }

        if (mat[i][n - 1] <= num) {

            ans += n;

            continue;

        }

        int greaterThan = 0;

        for (int jump = n / 2; jump >= 1; jump /= 2) {

            while (greaterThan + jump < n &&

                   mat[i][greaterThan + jump] <= num)
{

                greaterThan += jump;

            }

        }

        ans += greaterThan + 1;

    }

    return ans;
}
int kthSmallest(int mat[4][4], int n, int k) {
    int l = mat[0][0], r = mat[n - 1][n - 1];
    while (l <= r) {
        int mid = l + (r - l) / 2;
        int greaterThanOrEqualMid =
getElementsGreaterThanOrEqual(mid, n, mat);
        if (greaterThanOrEqualMid >= k)
            r = mid - 1;
        else
            l = mid + 1;
    }
    return l;
}
int main() {
    int n = 4;
    int mat[4][4] = {
        {10, 20, 30, 40},
        {15, 25, 35, 45},
        {25, 29, 37, 48},
        {32, 33, 39, 50},
    };
    cout << "7th smallest element is " <<
kthSmallest(mat, 4, 7);
    return 0;
}
```

Java

Python3

C#

Javascript

Output:

7th smallest element is 30

Complexity Analysis

- **Time Complexity** : O(y * n*logn)

  Where y = log(abs(Mat[0][0] - Mat[n-1][n-1]))

1. We call the getElementsGreaterThanOrEqual function log (abs(Mat[0][0] – Mat[n-1][n-1])) times
2. Time complexity of getElementsGreaterThanOrEqual is O(n logn) since there we do binary search n times.

- **Auxiliary Space**: O(1)

USING ARRAY:

We will make a new array and will copy all the contents of matrix in this array. After that we will sort that array and find kth smallest element. This will be so easier.

- C++
- Java
- Python3
- C#
- Javascript

C++

```
#include <bits/stdc++.h>
```

```
using namespace std;
int kthSmallest(int mat[4][4], int n, int k)
{
  int a[n*n];
  int v = 0;
  for(int i = 0; i < n; i++)
  {
    for(int j = 0; j < n; j++)
    {
      a[v] = mat[i][j];
      v++;
    }
  }
  sort(a, a + (n*n));
  int result = a[k - 1];
  return result;
}
int main()
{
  int mat[4][4] = { { 10, 20, 30, 40 },
                    { 15, 25, 35, 45 },
                    { 25, 29, 37, 48 },
                    { 32, 33, 39, 50 } };
  int res = kthSmallest(mat, 4, 7);
  cout <<  "7th smallest element is " << res;
  return 0;
}
```

Java

Python3

C#

Javascript

Output

7th smallest element is 30

Time Complexity: O(n^2)

Auxiliary Space: O(n^2)

Using Priority queue approach

- C++14
- Java
- Python3
- C#

C++14

```
#include <bits/stdc++.h>

using namespace std;

int kthSmallest(vector<vector<int>>& matrix, int k) {

  int i,j,n=matrix.size();

  priority_queue<int> maxH;

  if(k==1)

    return matrix[0][0];

  for(i=0;i<n;i++)

  {
    for(j=0;j<n;j++)
    {
      maxH.push(matrix[i][j]);
      if(maxH.size()>k)
        maxH.pop();
    }
  }
  return maxH.top();
}
int main() {
    vector<vector<int>> matrix = {{1,5,9},{10,11,13},
{12,13,15}};
      int k = 8;
    cout << "8th smallest element is " <<
kthSmallest(matrix,k);
    return 0;
}
```

Java

Python3

C#

Output

```
8th smallest element is 13
```

Time Complexity: O(n^2)
Auxiliary Space: O(n)

This article is contributed by Aarti_Rathi. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above