Data structures and Algo in Java - Day 29

public class day29

{

public static void main (String [] args)

{

// int arr [] [] = {

// {1,2,5,1,4,5},

// {2,9,3,2,3,2},

// {1,7,6,0,1,3},

// {3,6,2,3,7,2}

// };

// int ans [] = findPeakElementIn2DMatrix(arr);

// System.out.println(Arrays.toString(ans));

// int arr [] [] = {

// {1,5,7,9,11},

// {2,3,4,5,10},

// {9,10,12,14,16}

// };

int testcase [] [] = {

{1,5,7,9,11},

{2,3,4,8,9},

{4,11,14,19,20},

{6,10,22,99,100},

{7,15,17,24,28}

};

findMedianIn2DMatrix(testcase);

}

public static int [] findPeakElementIn2DMatrix(int arr [] [])

{

int n = arr.length;

int m = arr[0].length;

int low = 0;

int high = m-1;

int left = -1;

int right = -1;

while(low<=high)

{

int mid = (low+high)/2;

int row = maxElement(arr,n,m,mid);

if(mid-1>=0) // int left = (mid-1>=0) ? arr[row][mid-1] : -1;

{

left = arr[row][mid-1];

}

else

{

left = -1;

}

if(mid+1<m)

{

right = arr[row][mid+1];

}

else

{

right = -1;

}

if(arr[row][mid]>left && arr[row][mid]>right)

{

return new int [] {row,mid};

}

else if(arr[row][mid]<left)

{

high = mid-1;

}

else

{

low = mid+1;

}

}

return new int [] {left,right};

}

public static int maxElement(int arr [] [], int n, int m, int column)

{

int max = -1;

int index = -1;

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

{

if(arr[i][column]>max)

{

max = arr[i][column];

index = i;

}

}

return index;

}

public static void findMedianIn2DMatrix(int arr [] [])

{

int n = arr.length;

int m = arr[0].length;

int req = (n\*m)/2;

int low = getLow(arr,n);

int high = getHigh(arr,n,m);

while(low<=high)

{

int mid = (low+high)/2;

int elementsLessThanMid = getLessThanMid(arr,n,m,mid);

if(elementsLessThanMid<=req)

{

low = mid +1;

}

else

{

high = mid-1;

}

}

System.out.println("The median is: "+low);

}

public static int getLow(int arr [][] , int n)

{

int low =Integer.MAX\_VALUE;

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

{

low = Math.min(low,arr[i][0]);

}

return low;

}

public static int getHigh(int arr [][] , int n,int m)

{

int high =Integer.MIN\_VALUE;

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

{

high = Math.max(high,arr[i][m-1]);

}

return high;

}

public static int getLessThanMid(int arr [][], int n, int m , int mid)

{

int sum = 0;

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

{

sum = sum + upperBound(arr[i],mid);

}

return sum;

}

public static int upperBound(int arr [], int x)

{

int low = 0;

int high = arr.length-1;

while(low<=high)

{

int mid = (low+high)/2;

if(arr[mid]<=x)

{

low = mid+1;

}

else

{

high = mid -1;

}

}

return low;

}

}