ⓘ **Code Saved Successfully!!**

DP-S014 (E041)

Solved Challenges **0/1**

Back To Challenges List

Max Coins - Bottom Row Cannot Pick

ID:11143 **Solved By 478 Users**

In a room, the **R*C** boxes are arranged as a matrix where each box contains gold coins. A person is allowed to take gold coins from the room with the following conditions.
- He must pick only one box from a row.
- If he has picked a particular box then he cannot pick up the box in the bottom row of the same colomn.
The program must accept the number of gold coins in each box for **N** such rooms. For each room, the program must print the maximum number of gold coins that can be collected by the person as the output.

Boundary Condition(s):
1 <= N <= 100
2 <= R, C <= 100

Input Format:
The first line contains N.
The following lines containing the integers representing the N matrices.

Output Format:
The first N lines, each containing an integer representing the maximum number of gold coins that can be collected by the person.

Example Input/Output 1:
Input:
1
4 4
20 50 100 120
200 100 60 400
60 50 70 900
500 100 90 200

Output:
1720

Explanation:
The maximum number of gold coins that can be collected by the person is **1720**.
1st Row - **120**
2nd Row - **200**
3rd Row - **900**
4th Row - **500**

Example Input/Output 2:
Input:
2
5 5
25 98 74 11 89
53 68 36 48 23
4 14 99 48 41
40 22 97 72 1
29 67 61 92 49
2 6
45 10 12 78 66 90
9 1 3 15 12 95

Output:
395
173

Example Input/Output 3:
Input:
3
4 2
30 69
95 7
57 28
80 79
3 4
44 3 16 56
2 88 81 51
18 87 26 59
10 2
55 57
87 32
93 28
26 9
13 87
44 63
84 97
26 63
60 91
41 97

Output:
272
224
584

Max Execution Time Limit: 500 millisecs

Ambiance

Java (12.0)

Reset

```
import java.util.*;
public class Hello {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int T = sc.nextInt();
        while(T-->0)
        {
            int R = sc.nextInt(), C = sc.nextInt();
            int matrix[][] = new int[R][C];
            for(int row=0;row<R;row++)
            {
                for(inr col=0;col<C;col++)
                    matrix[row][col]=sc.nextInt();
            }

            int dp[][] = new int[R][C];
            for(int col=0;col<C;col++)
            dp[0][col] = matrix[0][col];

            for(int row=1;row<R;row++)
            {
                int prevRow[] = Arrays.copyOf(dp[row-1],C);
                Arrays.sort(prevRow);
                int firstMax = prevRow[C-1];
                int secMax = prevRow[C-2];
                for(int col=0;col<C;col++)
                {
                    if(dp[row-1][col]!=firstMax)
                    {
                        dp[row][col] = matrix[row][col]+firstM
                    }
                    else
                        dp[row][col] = matrix[row][col]+secMax
                }
            }
            Arrays.sort(dp[R-1]);
            System.out.println(dp[R-1][C-1]);
        }


    }
}
```

Save Run

Run with a custom test case (Input/Output)