Daily Test   Happy Coding from necse   SkillRack

Time Left: 00:00:00

Sub Matrix Sum

The program must accept an integer **S** and an integer matrix of size **R*C** as the input. Then the program must print the sub matrices which have the sum as S as the output.

Boundary Condition(s):
1 <= R, C <= 20
1 <= Matrix element value <= 1000

Input Format:
The first line contains S, R and C separated by a space.
The next R lines, each contains C integer values separated by a space.

Output Format:
The lines contain the submatrices having the sum as S. Each sub-matrix is separated by a new line containing the string value END.

Example Input/Output 1:
Input:
```
26 4 4
1 2 3 4
20 3 1 2
7 2 6 1
5 4 6 1
```

Output:
```
1 2
20 3
END
20 3 1 2
END
3 1 2
2 6 1
4 6 1
```

Explanation:
Here **S=26**, the submatrices which have the sum as 26 are highlighted below.
1st sub matrix:
1 2 3 4
20 3 1 2
7 2 6 1
5 4 6 1
2nd sub matrix:
1 2 3 4
20 3 1 2
7 2 6 1
5 4 6 1
3rd sub matrix:
1 2 3 4
20 **3 1 2**
7 **2 6 1**
5 **4 6 1**

Example Input/Output 2:
Input:
```
26 4 4
1 2 3 4
2 3 1 2
7 2 6 1
5 4 6 1
```

Output:
```
1 2
2 3
7 2
5 4
END
```

```
3 1 2
2 6 1
4 6 1
```

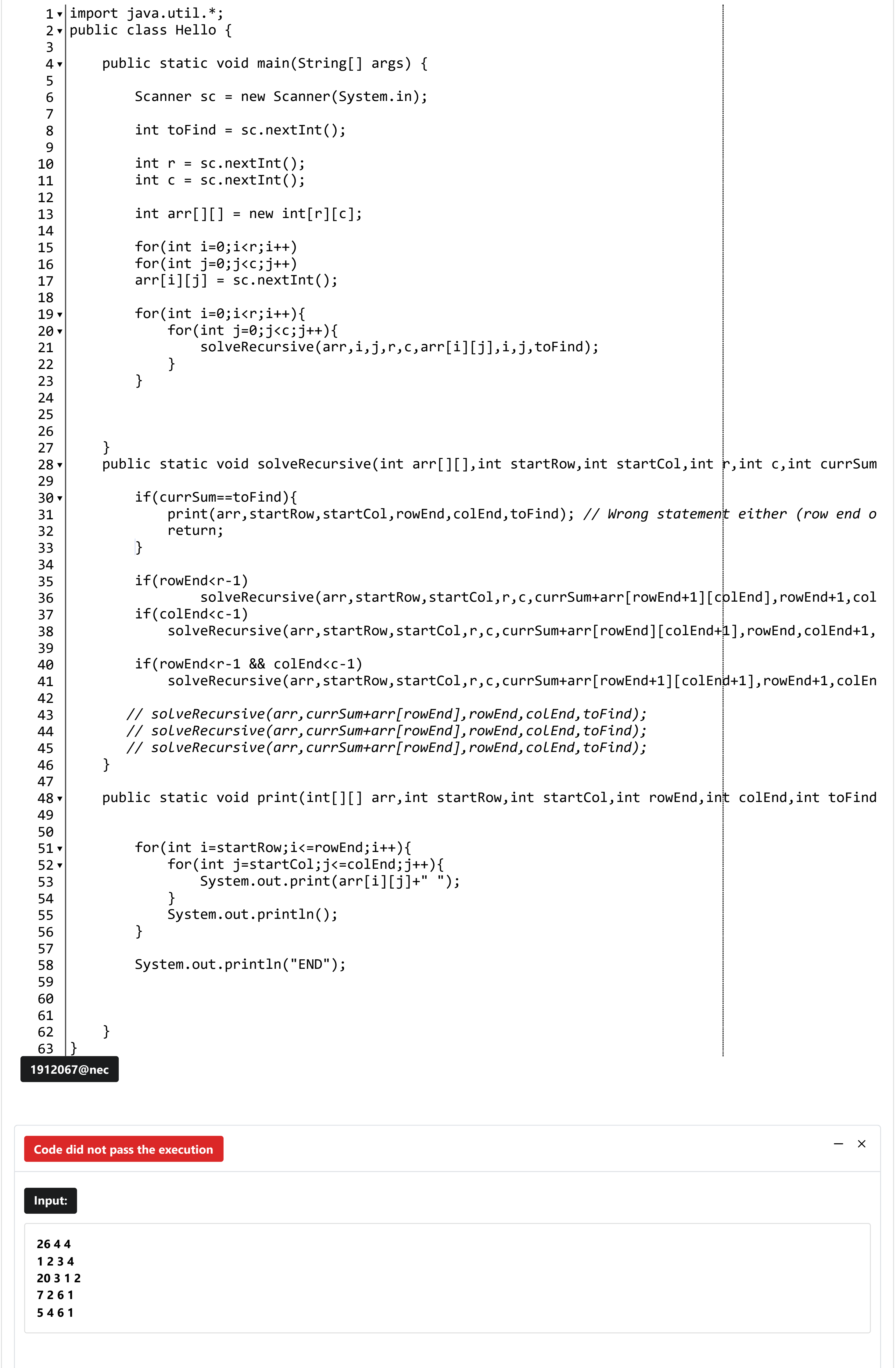
Example Input/Output 3:
Input:
```
95 3 4
25 11 18 13
22 19 18 16
11 24 10 19
```

Output:
```
11 18 13
19 18 16
```

Max Execution Time Limit: 50 millisecs

Ambiance

Java (12.0)



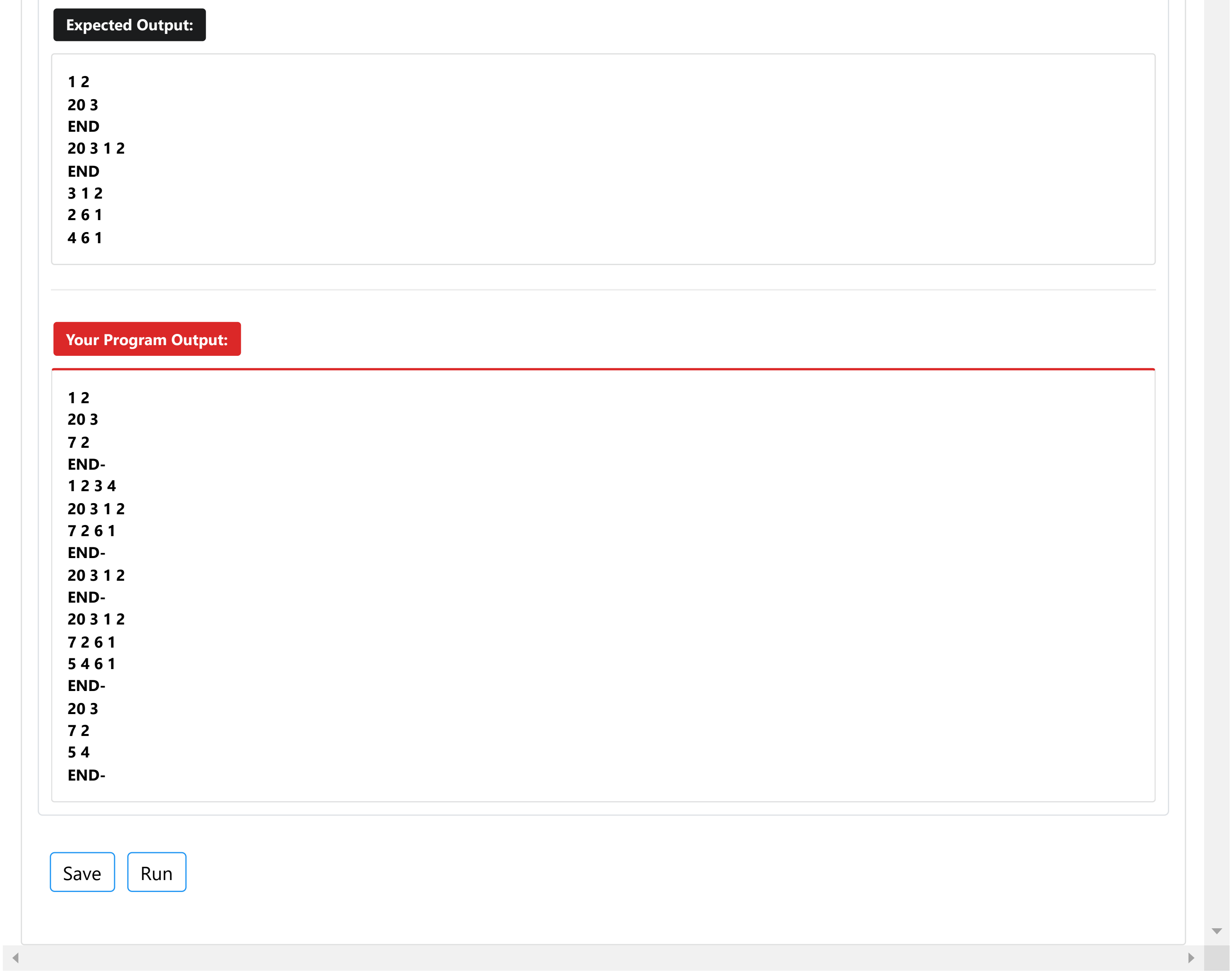
```java
import java.util.*;
public class Hello {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        int toFind = sc.nextInt();

        int r = sc.nextInt();
        int c = sc.nextInt();

        int arr[][] = new int[r][c];

        for(int i=0;i<r;i++)
        for(int j=0;j<c;j++)
        arr[i][j] = sc.nextInt();

        for(int i=0;i<r;i++){
            for(int j=0;j<c;j++){
                solveRecursive(arr,i,j,r,c,arr[i][j],i,j,toFind);
            }
        }



    }
    public static void solveRecursive(int arr[][],int startRow,int startCol,int r,int c,int currSum

        if(currSum==toFind){
            print(arr,startRow,startCol,rowEnd,colEnd,toFind); // Wrong statement either (row end o
            return;
        }

        if(rowEnd<r-1)
                solveRecursive(arr,startRow,startCol,r,c,currSum+arr[rowEnd+1][colEnd],rowEnd+1,col
        if(colEnd<c-1)
            solveRecursive(arr,startRow,startCol,r,c,currSum+arr[rowEnd][colEnd+1],rowEnd,colEnd+1,

        if(rowEnd<r-1 && colEnd<c-1)
            solveRecursive(arr,startRow,startCol,r,c,currSum+arr[rowEnd+1][colEnd+1],rowEnd+1,colEn

       // solveRecursive(arr,currSum+arr[rowEnd],rowEnd,colEnd,toFind);
       // solveRecursive(arr,currSum+arr[rowEnd],rowEnd,colEnd,toFind);
       // solveRecursive(arr,currSum+arr[rowEnd],rowEnd,colEnd,toFind);
    }

    public static void print(int[][] arr,int startRow,int startCol,int rowEnd,int colEnd,int toFind


        for(int i=startRow;i<=rowEnd;i++){
            for(int j=startCol;j<=colEnd;j++){
                System.out.print(arr[i][j]+" ");
            }
            System.out.println();
        }

        System.out.println("END");



    }
}
```

1912067@nec

Code did not pass the execution

Input:

```
26 4 4
1 2 3 4
20 3 1 2
7 2 6 1
5 4 6 1
```



Expected Output:

```
1 2
20 3
END
20 3 1 2
END
3 1 2
2 6 1
4 6 1
```

Your Program Output:

```
1 2
20 3
7 2
END-
1 2 3 4
20 3 1 2
7 2 6 1
END-
20 3 1 2
END-
20 3 1 2
7 2 6 1
5 4 6 1
END-
20 3
7 2
5 4
END-
```