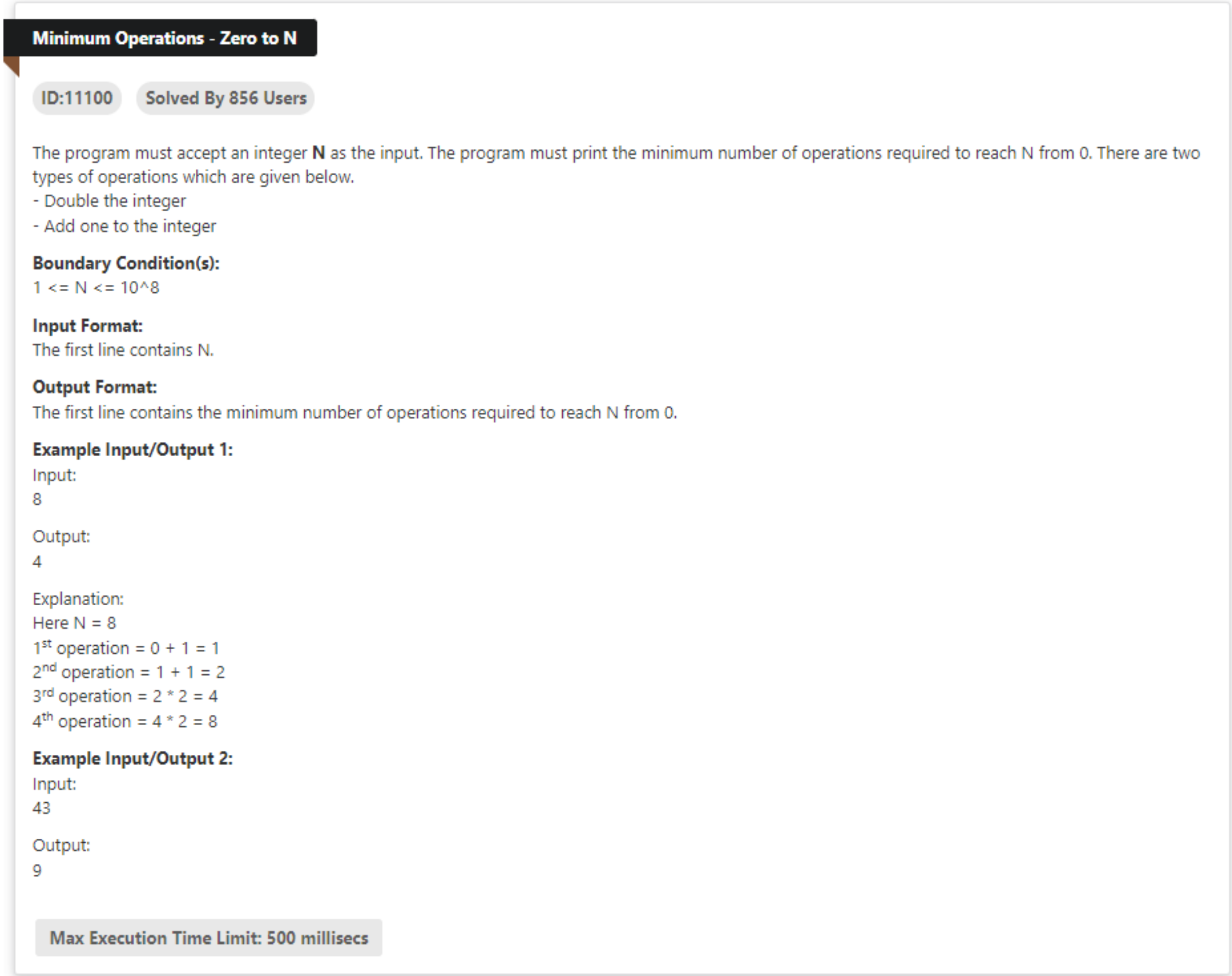
DAY 8:

SESSION 1:

Problem 1:



Minimum Operations - Zero to N

ID:11100 Solved By 856 Users

The program must accept an integer **N** as the input. The program must print the minimum number of operations required to reach N from 0. There are two types of operations which are given below.
- Double the integer
- Add one to the integer

Boundary Condition(s):
1 <= N <= 10^8

Input Format:
The first line contains N.

Output Format:
The first line contains the minimum number of operations required to reach N from 0.

Example Input/Output 1:
Input:
8

Output:
4

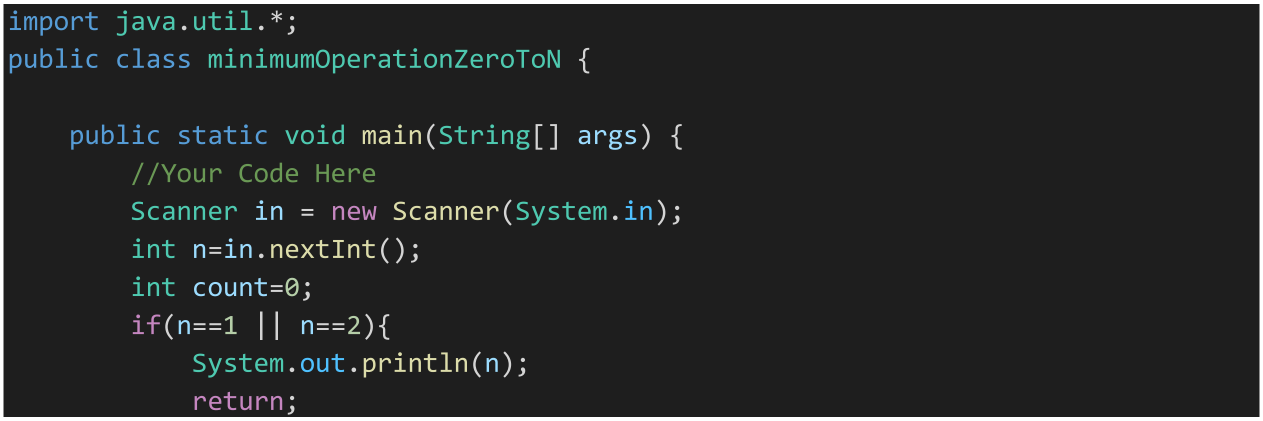
Explanation:
Here N = 8
1st operation = 0 + 1 = 1
2nd operation = 1 + 1 = 2
3rd operation = 2 * 2 = 4
4th operation = 4 * 2 = 8

Example Input/Output 2:
Input:
43

Output:
9

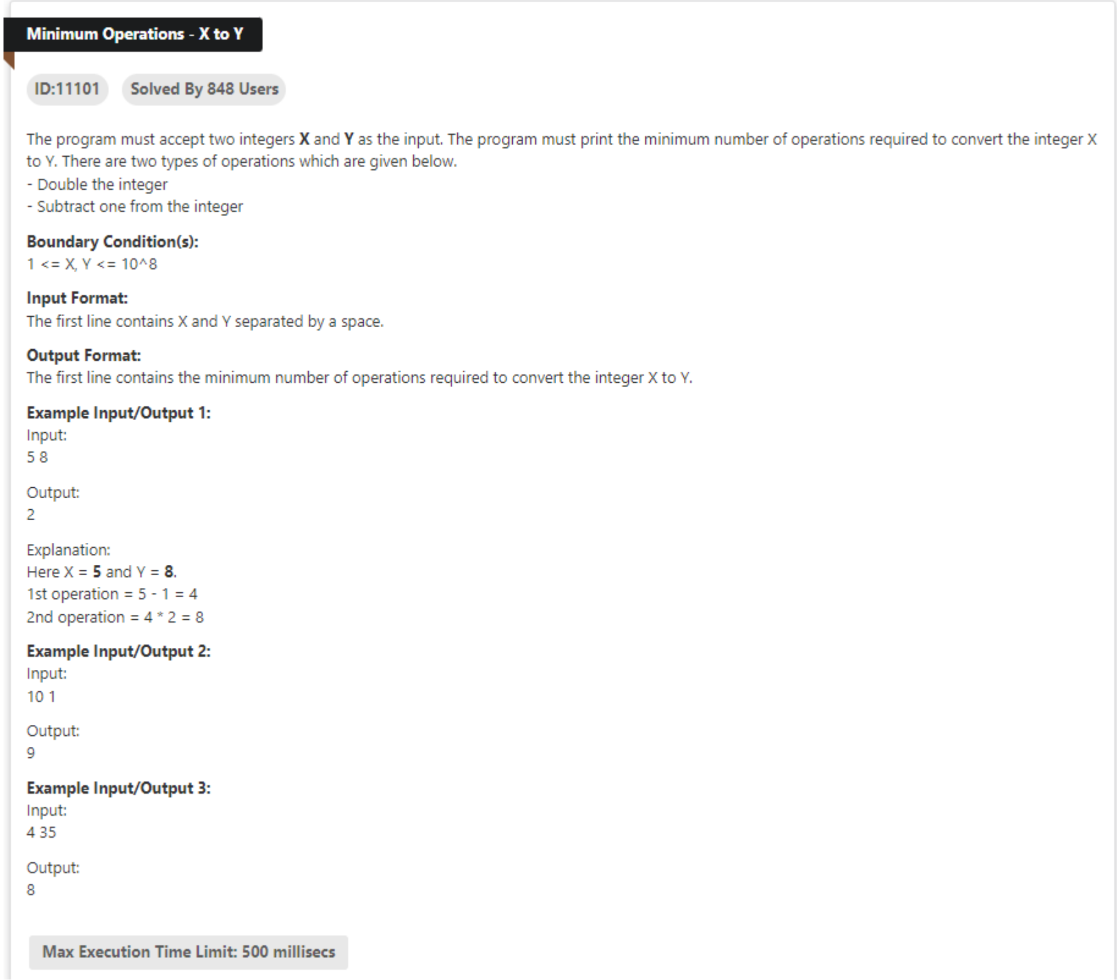
Max Execution Time Limit: 500 millisecs

Code:



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

    public static void main(String[] args) {
        //Your Code Here
        Scanner in = new Scanner(System.in);
        int n=in.nextInt();
        int count=0;
        if(n==1 || n==2){
            System.out.println(n);
            return;
        }
        else{
            while(n>2){
                if(n%2==0){  //we can also check for odd using binary like (n&1)
                    n=n/2;
                }
                else{
                    n=n-1;
                }
                count++; //increasing the count for one operation
            }
        }
        count=count+2;
        System.out.println(count); //adding two to count as minumum 2 operation is required from 0 to 2
    }}
```

Problem 2:



Minimum Operations - X to Y

ID:11101 Solved By 848 Users

The program must accept two integers **X** and **Y** as the input. The program must print the minimum number of operations required to convert the integer X to Y. There are two types of operations which are given below.
- Double the integer
- Subtract one from the integer

Boundary Condition(s):
1 <= X, Y <= 10^8

Input Format:
The first line contains X and Y separated by a space.

Output Format:
The first line contains the minimum number of operations required to convert the integer X to Y.

Example Input/Output 1:
Input:
5 8

Output:
2

Explanation:
Here X = **5** and Y = **8**.
1st operation = 5 - 1 = 4
2nd operation = 4 * 2 = 8

Example Input/Output 2:
Input:
10 1

Output:
9

Example Input/Output 3:
Input:
4 35

Output:
8

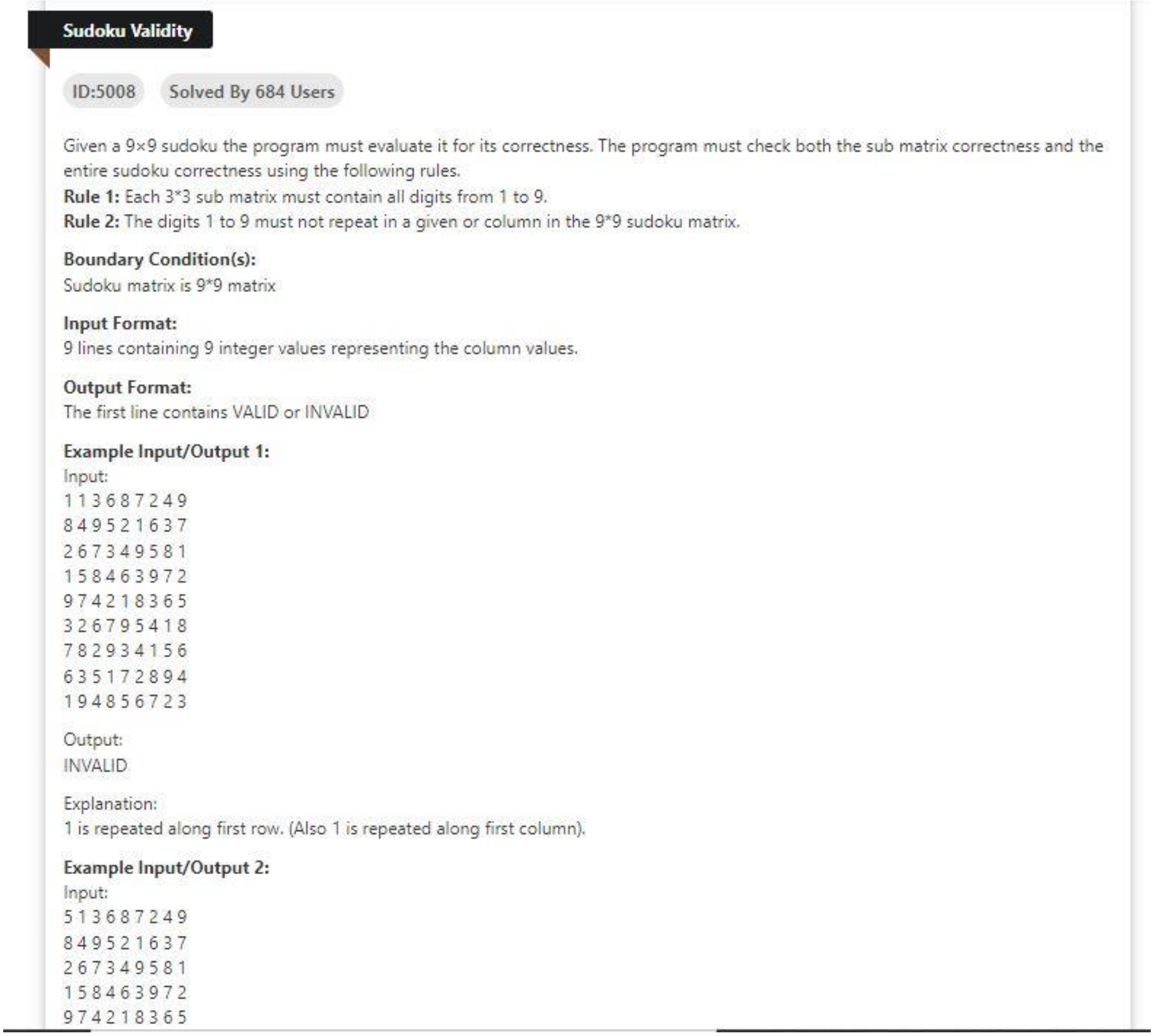
Max Execution Time Limit: 500 millisecs

Code:

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

    public static void main(String[] args) {
        //Your Code Here
        Scanner in = new Scanner(System.in);
        int x=in.nextInt();
        int y=in.nextInt();
        int count=0;
        while(y>x){
            if(y%2!=0){
                y=y+1;
            }
            else{
                y=y/2;
            }
            count++;
        }
        count=count+(x-y); //when the while loop exists then x>y so only 
operation to be done is x-y added with count
        System.out.println(count);
    }
}
```

SESSION 2:

Problem 1:



Sudoku Validity

ID:5008 Solved By 684 Users

Given a 9×9 sudoku the program must evaluate it for its correctness. The program must check both the sub matrix correctness and the entire sudoku correctness using the following rules.
Rule 1: Each 3*3 sub matrix must contain all digits from 1 to 9.
Rule 2: The digits 1 to 9 must not repeat in a given or column in the 9*9 sudoku matrix.

Boundary Condition(s):
Sudoku matrix is 9*9 matrix

Input Format:
9 lines containing 9 integer values representing the column values.

Output Format:
The first line contains VALID or INVALID

Example Input/Output 1:
Input:

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

Output:

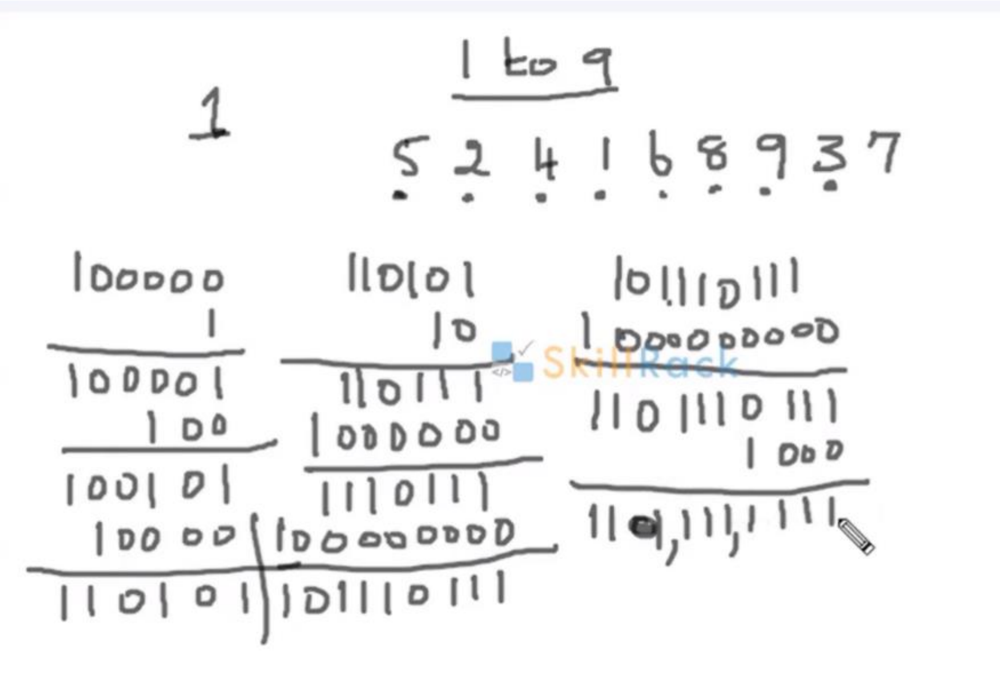
```
INVALID
```

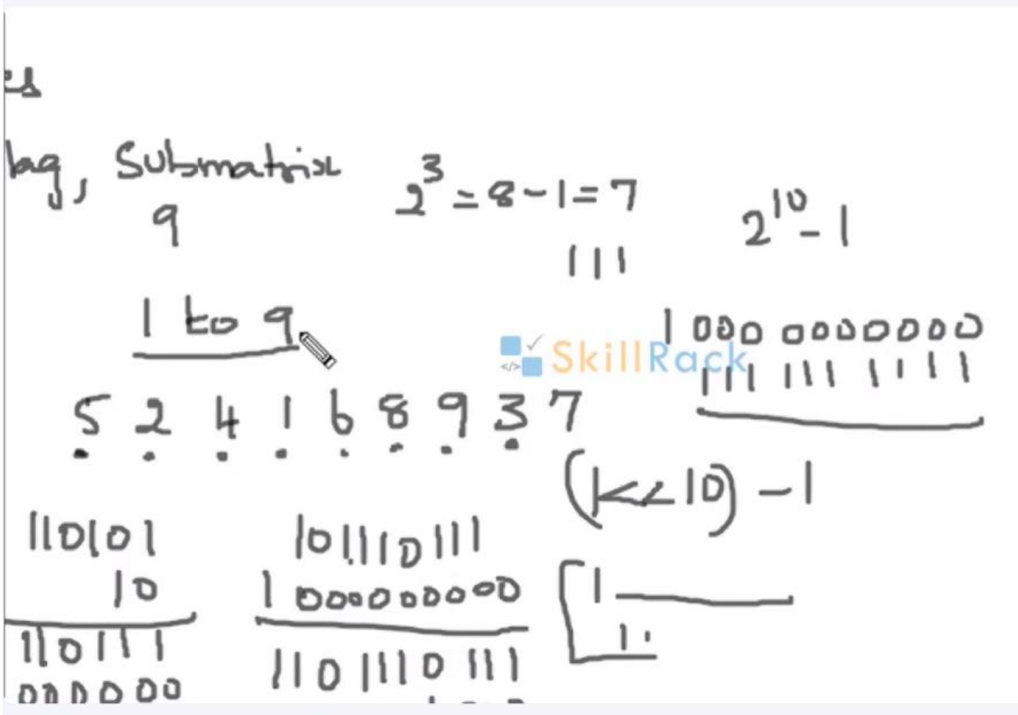
Explanation:
1 is repeated along first row. (Also 1 is repeated along first column).

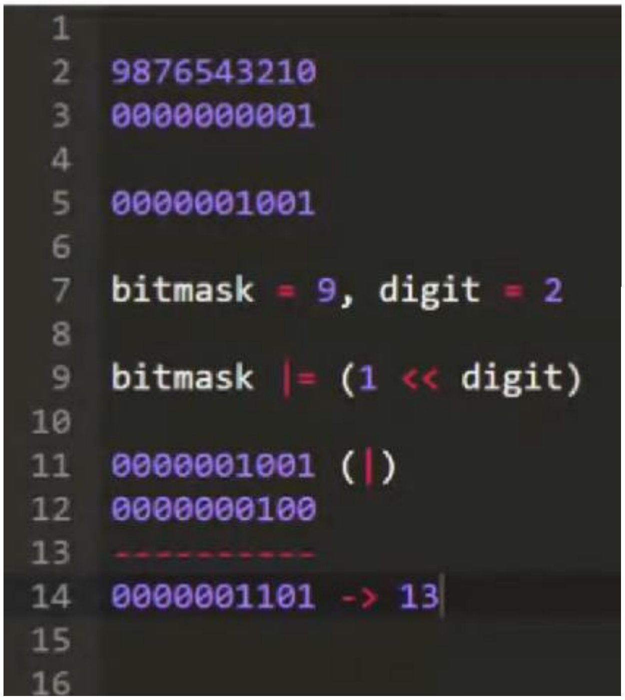
Example Input/Output 2:
Input:

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

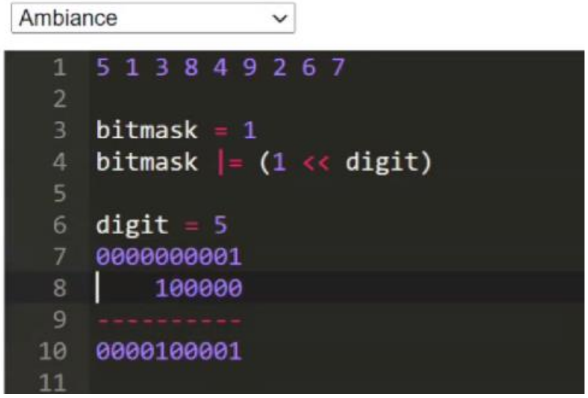
Bit masking technique





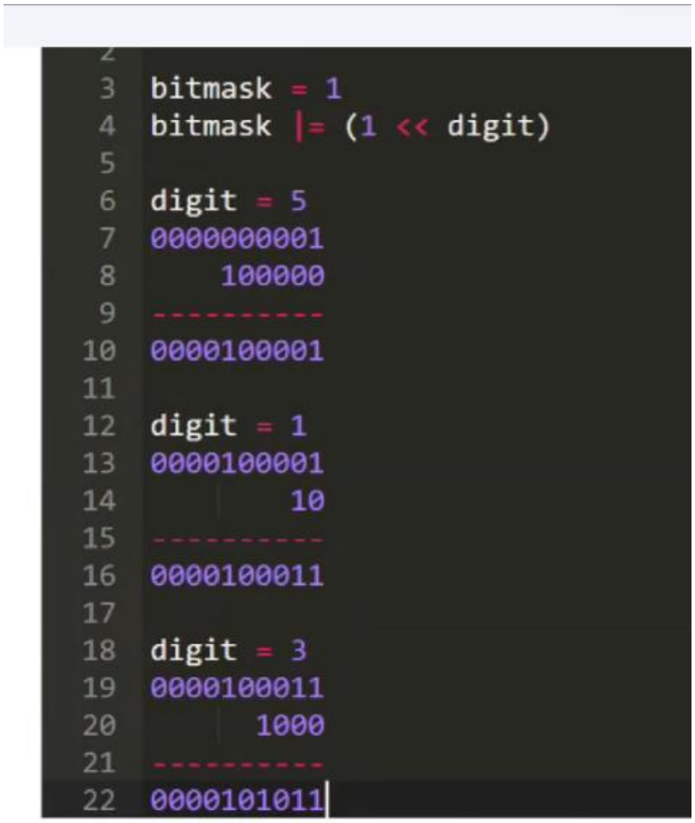


```

9876543210
0000000001

0000001001

bitmask = 9, digit = 2

bitmask |= (1 << digit)

0000001001 (|)
0000000100
----------
0000001101 -> 13


```



Ambiance

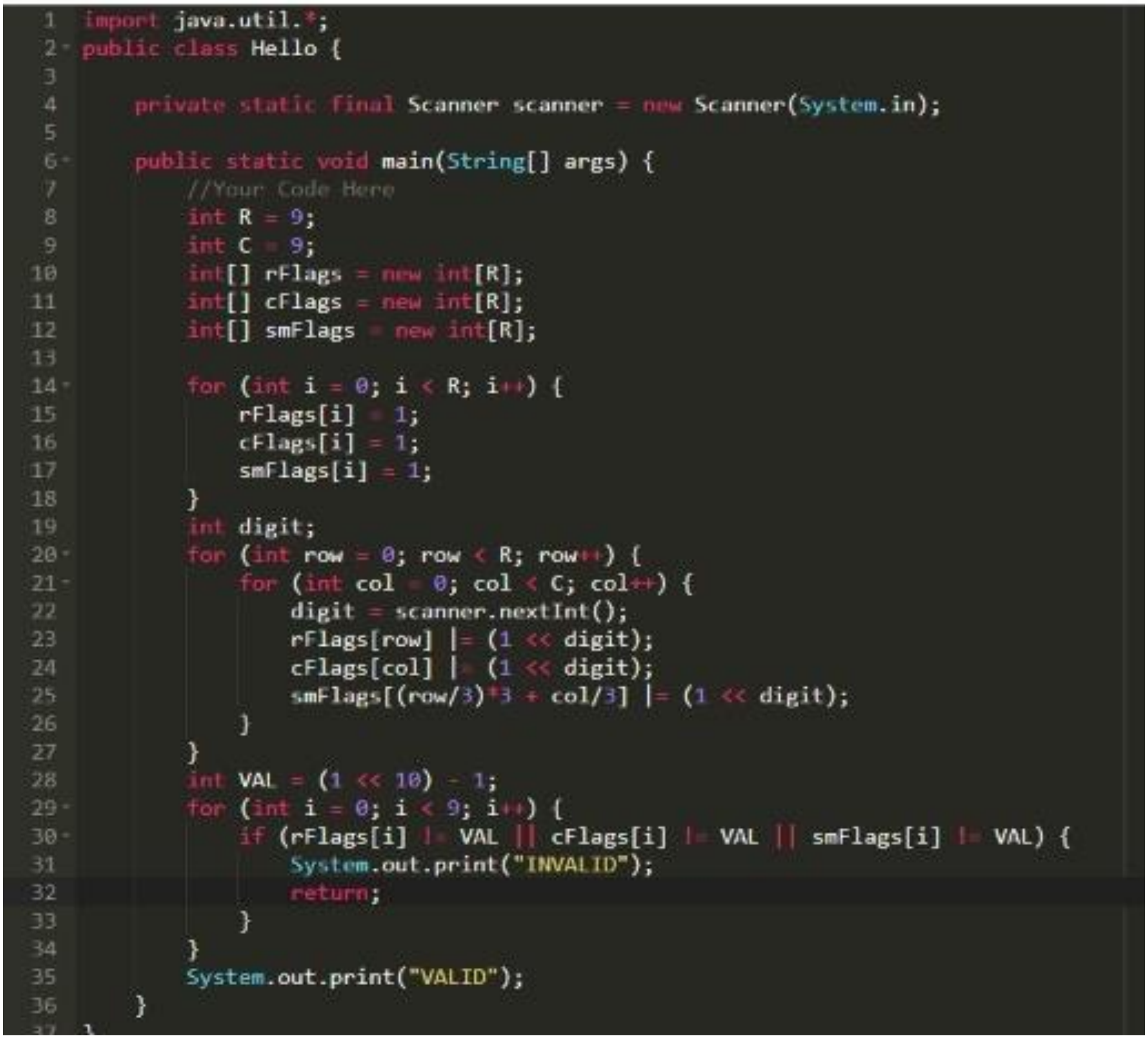
```
5 1 3 8 4 9 2 6 7

bitmask = 1
bitmask |= (1 << digit)

digit = 5
0000000001
    100000
----------
0000100001

```



```
bitmask = 1
bitmask |= (1 << digit)

digit = 5
0000000001
    100000
----------
0000100001

digit = 1
0000100001
        10
----------
0000100011

digit = 3
0000100011
      1000
----------
0000101011
```

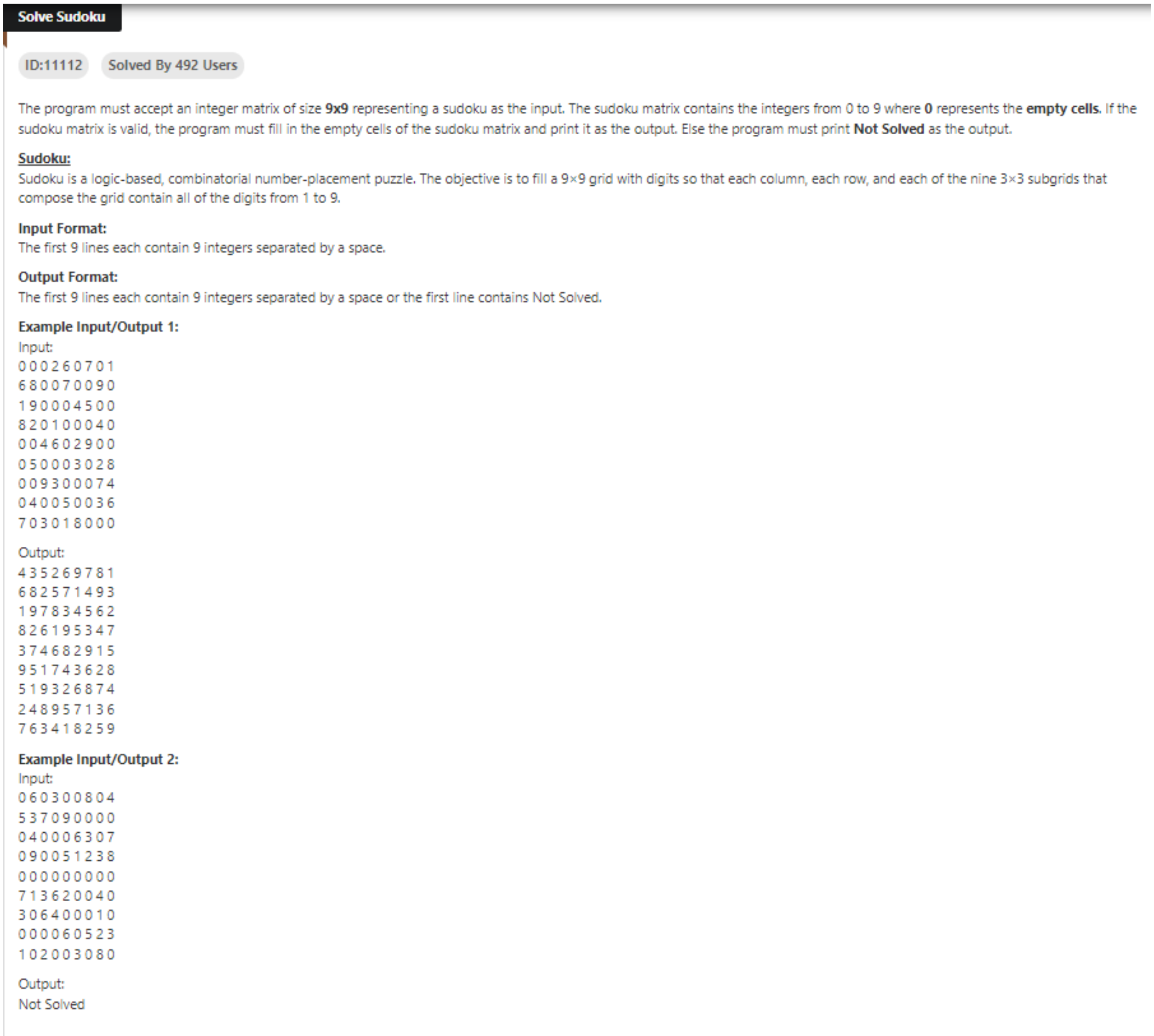
Code:

```
#include<stdio.h>
#include<stdlib.h>
#define R 9
#define C 9
int main()
{
int rflags[9],cflags[9],smflags[9];
for(int index =0;index<9;index++){
    rflags[index]=cflags[index]=smflags[index]=1;
}
int digit;
for(int row=0;row<R;row++){
    for(int col=0;col<C;col++){
        scanf("%d",&digit);
        rflags[row] |= (1<<digit);
        cflags[col] |= (1<<digit);
        smflags[(row/3)*3+col/3] |= (1<<digit);
    }
}
int val=(1<<10)-1;
for(int index=0;index<9;index++){
    if(rflags[index]!=val || cflags[index] !=val || smflags[index] !=val){
        printf("INVALID");
        return;
    }
}
printf("VALID");
}
```



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

    private static final Scanner scanner = new Scanner(System.in);

    public static void main(String[] args) {
        //Your Code Here
        int R = 9;
        int C = 9;
        int[] rFlags = new int[R];
        int[] cFlags = new int[R];
        int[] smFlags = new int[R];

        for (int i = 0; i < R; i++) {
            rFlags[i] = 1;
            cFlags[i] = 1;
            smFlags[i] = 1;
        }
        int digit;
        for (int row = 0; row < R; row++) {
            for (int col = 0; col < C; col++) {
                digit = scanner.nextInt();
                rFlags[row] |= (1 << digit);
                cFlags[col] |= (1 << digit);
                smFlags[(row/3)*3 + col/3] |= (1 << digit);
            }
        }
        int VAL = (1 << 10) - 1;
        for (int i = 0; i < 9; i++) {
            if (rFlags[i] != VAL || cFlags[i] != VAL || smFlags[i] != VAL) {
                System.out.print("INVALID");
                return;
            }
        }
        System.out.print("VALID");
    }
}
```

Problem 2:



Solve Sudoku

ID:11112 Solved By 492 Users

The program must accept an integer matrix of size **9x9** representing a sudoku as the input. The sudoku matrix contains the integers from 0 to 9 where **0** represents the **empty cells**. If the sudoku matrix is valid, the program must fill in the empty cells of the sudoku matrix and print it as the output. Else the program must print **Not Solved** as the output.

Sudoku:
Sudoku is a logic-based, combinatorial number-placement puzzle. The objective is to fill a 9×9 grid with digits so that each column, each row, and each of the nine 3×3 subgrids that compose the grid contain all of the digits from 1 to 9.

Input Format:
The first 9 lines each contain 9 integers separated by a space.

Output Format:
The first 9 lines each contain 9 integers separated by a space or the first line contains Not Solved.

Example Input/Output 1:
Input:

```
0 0 0 2 6 0 7 0 1
6 8 0 0 7 0 0 9 0
1 9 0 0 0 4 5 0 0
8 2 0 1 0 0 0 4 0
0 0 4 6 0 2 9 0 0
0 5 0 0 0 3 0 2 8
0 0 9 3 0 0 0 7 4
0 4 0 0 5 0 0 3 6
7 0 3 0 1 8 0 0 0
```

Output:

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

Example Input/Output 2:
Input:

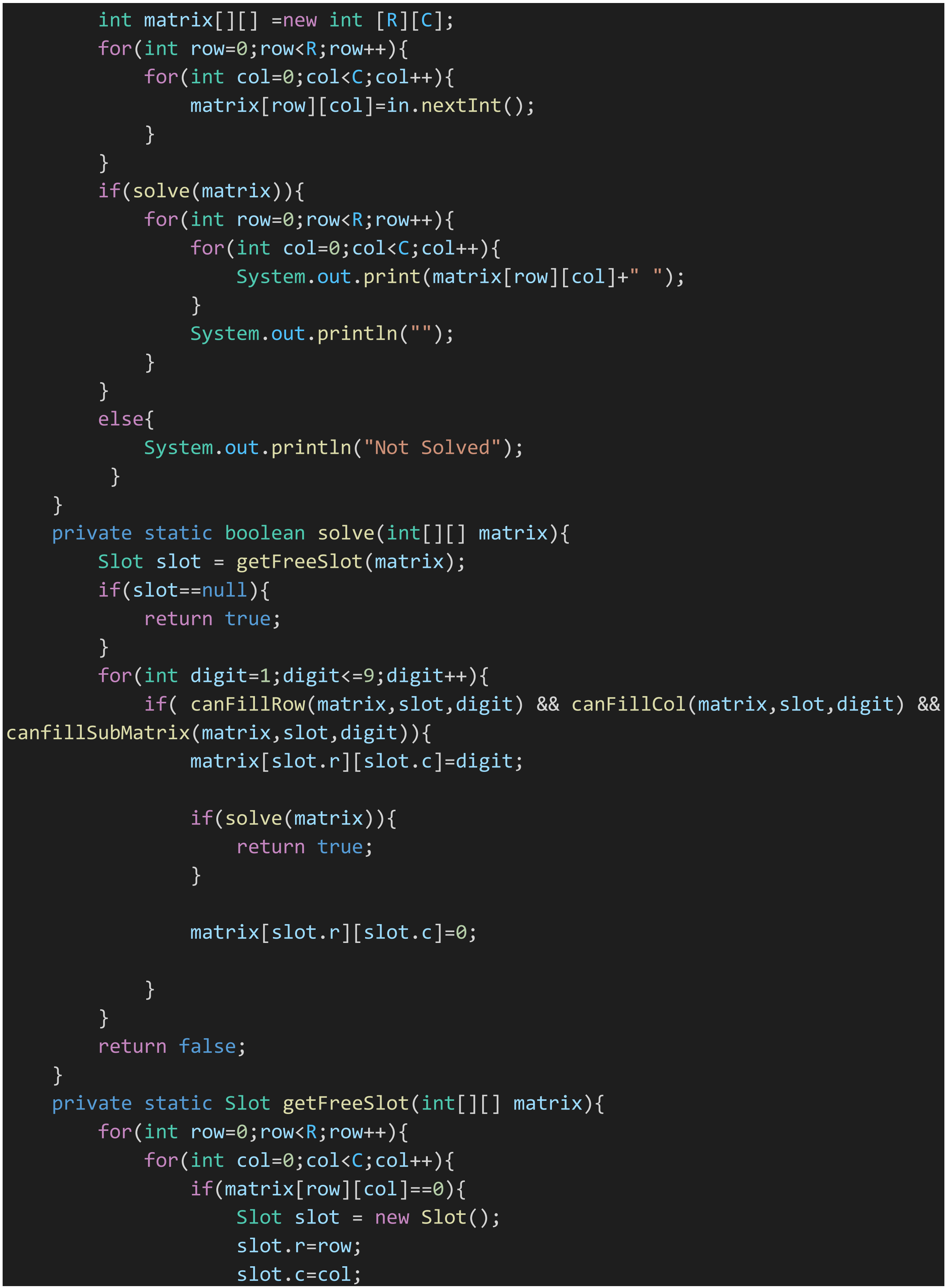
```
0 6 0 3 0 0 8 0 4
5 3 7 0 9 0 0 0 0
0 4 0 0 0 6 3 0 7
0 9 0 0 5 1 2 3 8
0 0 0 0 0 0 0 0 0
7 1 3 6 2 0 0 4 0
3 0 6 4 0 0 0 1 0
0 0 0 0 6 0 5 2 3
1 0 2 0 0 3 0 8 0
```

Output:

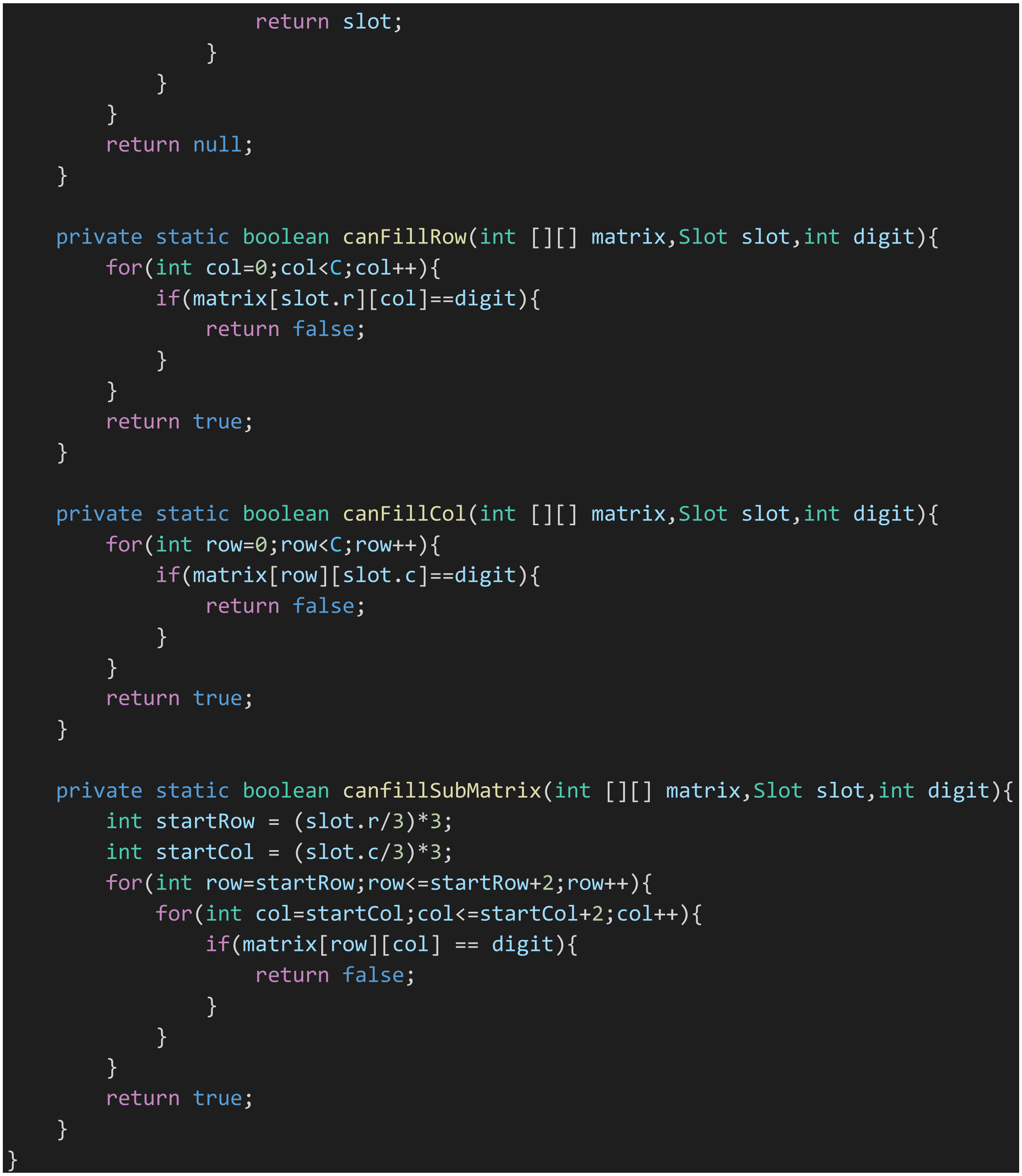
```
Not Solved
```

Code:

```
import java.util.*;
class Slot{
    int r,c;
}

public class solveSudoku {

    static final int R=9,C=9;
    public static void main(String[] args) {
        //Your Code Here
        Scanner in = new Scanner(System.in);
```



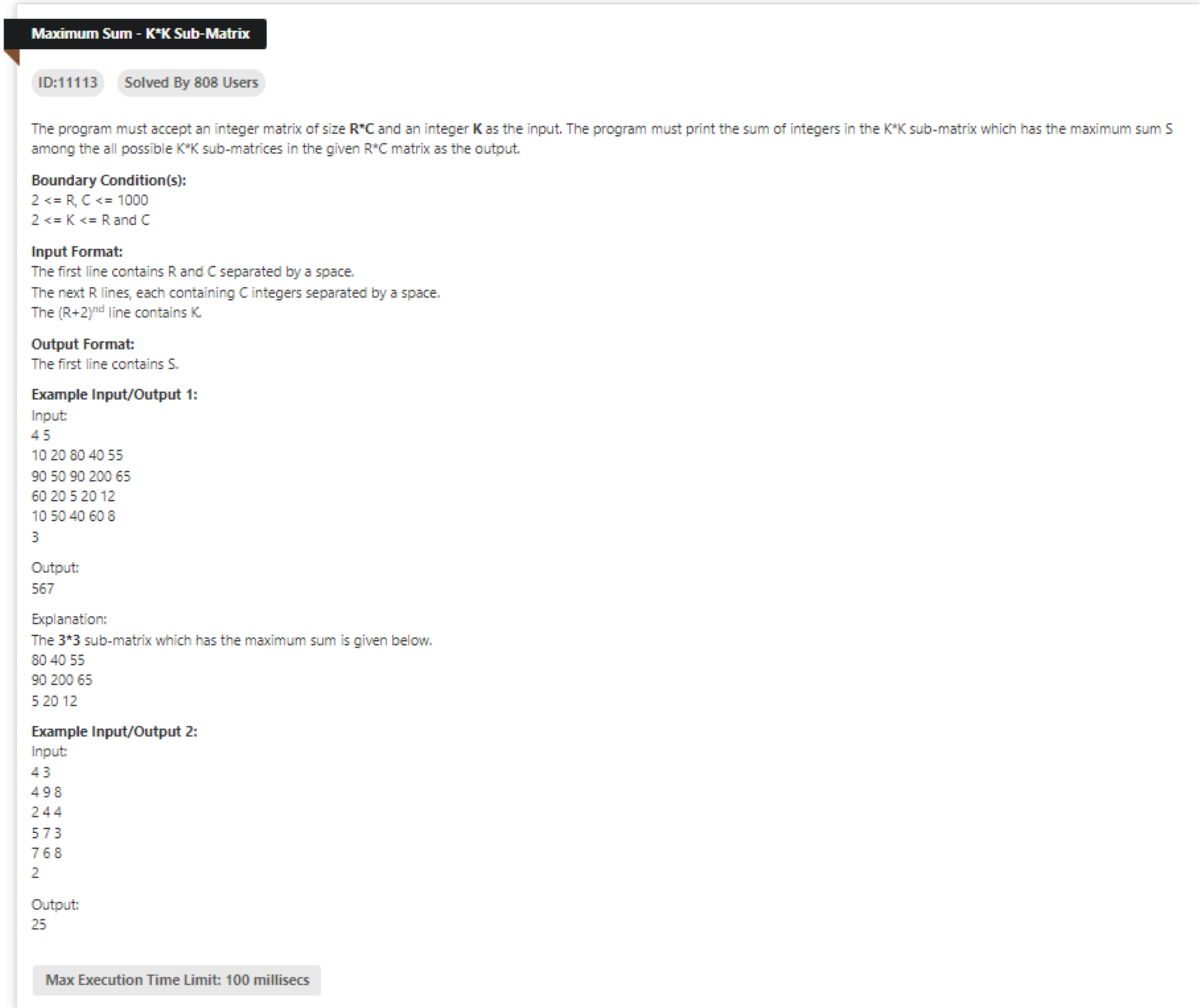
```
        int matrix[][] =new int [R][C];
        for(int row=0;row<R;row++){
            for(int col=0;col<C;col++){
                matrix[row][col]=in.nextInt();
            }
        }
        if(solve(matrix)){
            for(int row=0;row<R;row++){
                for(int col=0;col<C;col++){
                    System.out.print(matrix[row][col]+" ");
                }
                System.out.println("");
            }
        }
        else{
            System.out.println("Not Solved");
         }
    }
    private static boolean solve(int[][] matrix){
        Slot slot = getFreeSlot(matrix);
        if(slot==null){
            return true;
        }
        for(int digit=1;digit<=9;digit++){
            if( canFillRow(matrix,slot,digit) && canFillCol(matrix,slot,digit) &&
canfillSubMatrix(matrix,slot,digit)){
                matrix[slot.r][slot.c]=digit;

                if(solve(matrix)){
                    return true;
                }

                matrix[slot.r][slot.c]=0;

            }
        }
        return false;
    }
    private static Slot getFreeSlot(int[][] matrix){
        for(int row=0;row<R;row++){
            for(int col=0;col<C;col++){
                if(matrix[row][col]==0){
                    Slot slot = new Slot();
                    slot.r=row;
                    slot.c=col;
```



```
                    return slot;
                }
            }
        }
        return null;
    }

    private static boolean canFillRow(int [][] matrix,Slot slot,int digit){
        for(int col=0;col<C;col++){
            if(matrix[slot.r][col]==digit){
                return false;
            }
        }
        return true;
    }

    private static boolean canFillCol(int [][] matrix,Slot slot,int digit){
        for(int row=0;row<C;row++){
            if(matrix[row][slot.c]==digit){
                return false;
            }
        }
        return true;
    }

    private static boolean canfillSubMatrix(int [][] matrix,Slot slot,int digit){
        int startRow = (slot.r/3)*3;
        int startCol = (slot.c/3)*3;
        for(int row=startRow;row<=startRow+2;row++){
            for(int col=startCol;col<=startCol+2;col++){
                if(matrix[row][col] == digit){
                    return false;
                }
            }
        }
        return true;
    }
}
```

Session 3:

Problem 1:



Maximum Sum - K*K Sub-Matrix

ID:11113 Solved By 808 Users

The program must accept an integer matrix of size **R*C** and an integer **K** as the input. The program must print the sum of integers in the K*K sub-matrix which has the maximum sum S among the all possible K*K sub-matrices in the given R*C matrix as the output.

Boundary Condition(s):
2 <= R, C <= 1000
2 <= K <= R and C

Input Format:
The first line contains R and C separated by a space.
The next R lines, each containing C integers separated by a space.
The $(R+2)^{nd}$ line contains K.

Output Format:
The first line contains S.

Example Input/Output 1:
Input:
4 5
10 20 80 40 55
90 50 90 200 65
60 20 5 20 12
10 50 40 60 8
3

Output:
567

Explanation:
The **3*3** sub-matrix which has the maximum sum is given below.
80 40 55
90 200 65
5 20 12

Example Input/Output 2:
Input:
4 3
4 9 8
2 4 4
5 7 3
7 6 8
2

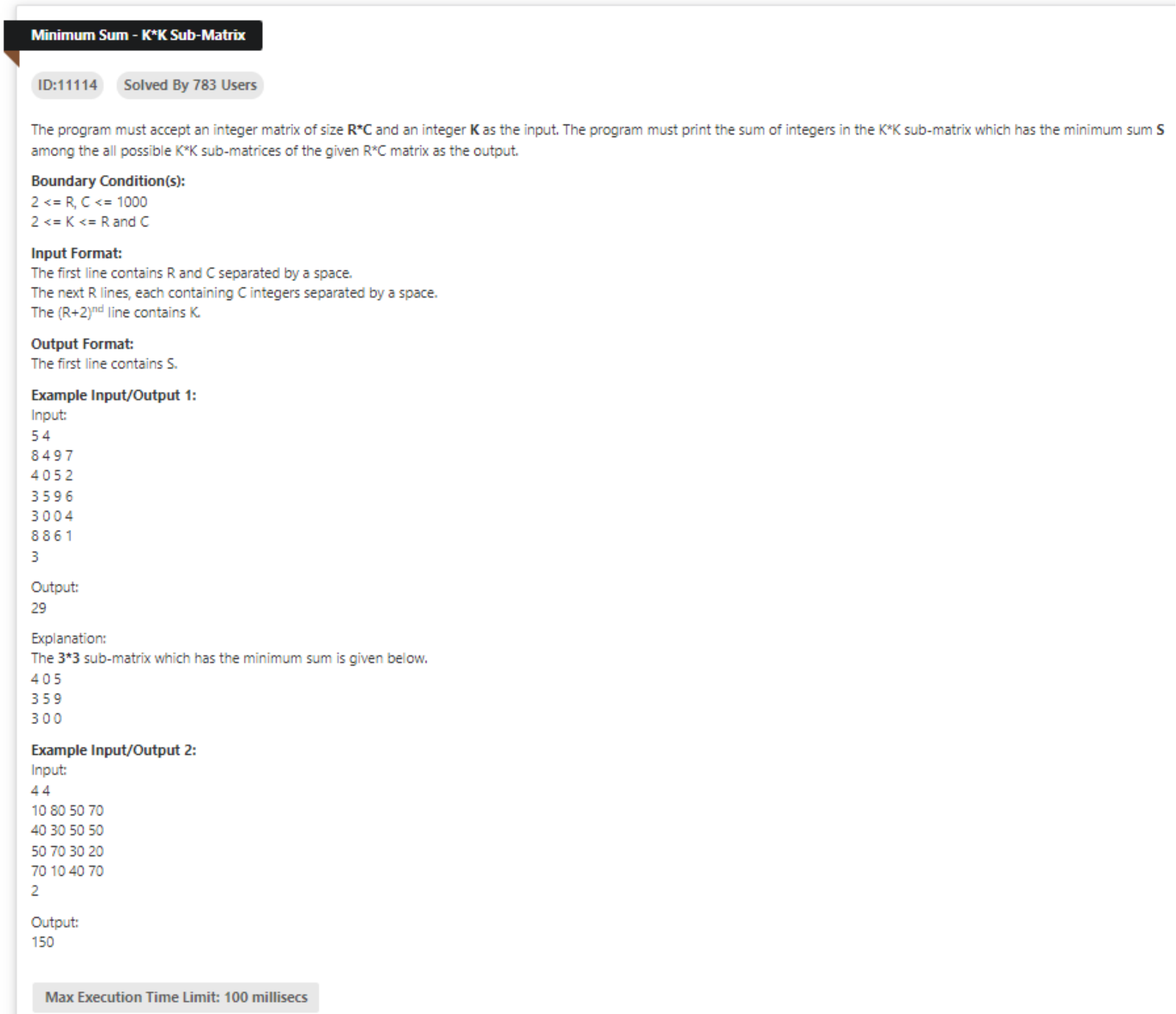
Output:
25

Max Execution Time Limit: 100 millisecs

Code:

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

    public static void main(String[] args) {
        //Your Code Here
        Scanner in =new Scanner(System.in);
        int R=in.nextInt();
        int C=in.nextInt();
        int [][] rowSum = new int[R][C+1]; //if solving in c language we have to
initialize first column of the matrix to zero

        for(int row=0;row<R;row++){
            for(int col=1;col<=C;col++){
                int curr =in.nextInt();
                rowSum[row][col] = curr+rowSum[row][col-1]; //adding previous
column value to the present column value each time
            }
        }
        int K=in.nextInt();

        int maxSum=Integer.MIN_VALUE; //assigning min value
        for(int row=0;row<=R-K;row++){
            for(int col=1; col<=C-K+1;col++){
                int sum=0;
                for(int srow=row;srow<row+K;srow++){
                    sum+=rowSum[srow][col+K-1]- rowSum[srow][col-1];
//subtracting 3th column value with 0th column value --one iteration
                }
                maxSum = Math.max(maxSum,sum);
            }
        }
        System.out.println(maxSum);
    }
}
```

Problem 2:



Minimum Sum - K*K Sub-Matrix

ID:11114 Solved By 783 Users

The program must accept an integer matrix of size **R*C** and an integer **K** as the input. The program must print the sum of integers in the K*K sub-matrix which has the minimum sum **S** among the all possible K*K sub-matrices of the given R*C matrix as the output.

Boundary Condition(s):
2 <= R, C <= 1000
2 <= K <= R and C

Input Format:
The first line contains R and C separated by a space.
The next R lines, each containing C integers separated by a space.
The (R+2)nd line contains K.

Output Format:
The first line contains S.

Example Input/Output 1:
Input:

```
5 4
8 4 9 7
4 0 5 2
3 5 9 6
3 0 0 4
8 8 6 1
3
```

Output:

```
29
```

Explanation:
The **3*3** sub-matrix which has the minimum sum is given below.

```
4 0 5
3 5 9
3 0 0
```

Example Input/Output 2:
Input:

```
4 4
10 80 50 70
40 30 50 50
50 70 30 20
70 10 40 70
2
```

Output:

```
150
```

Max Execution Time Limit: 100 millisecs

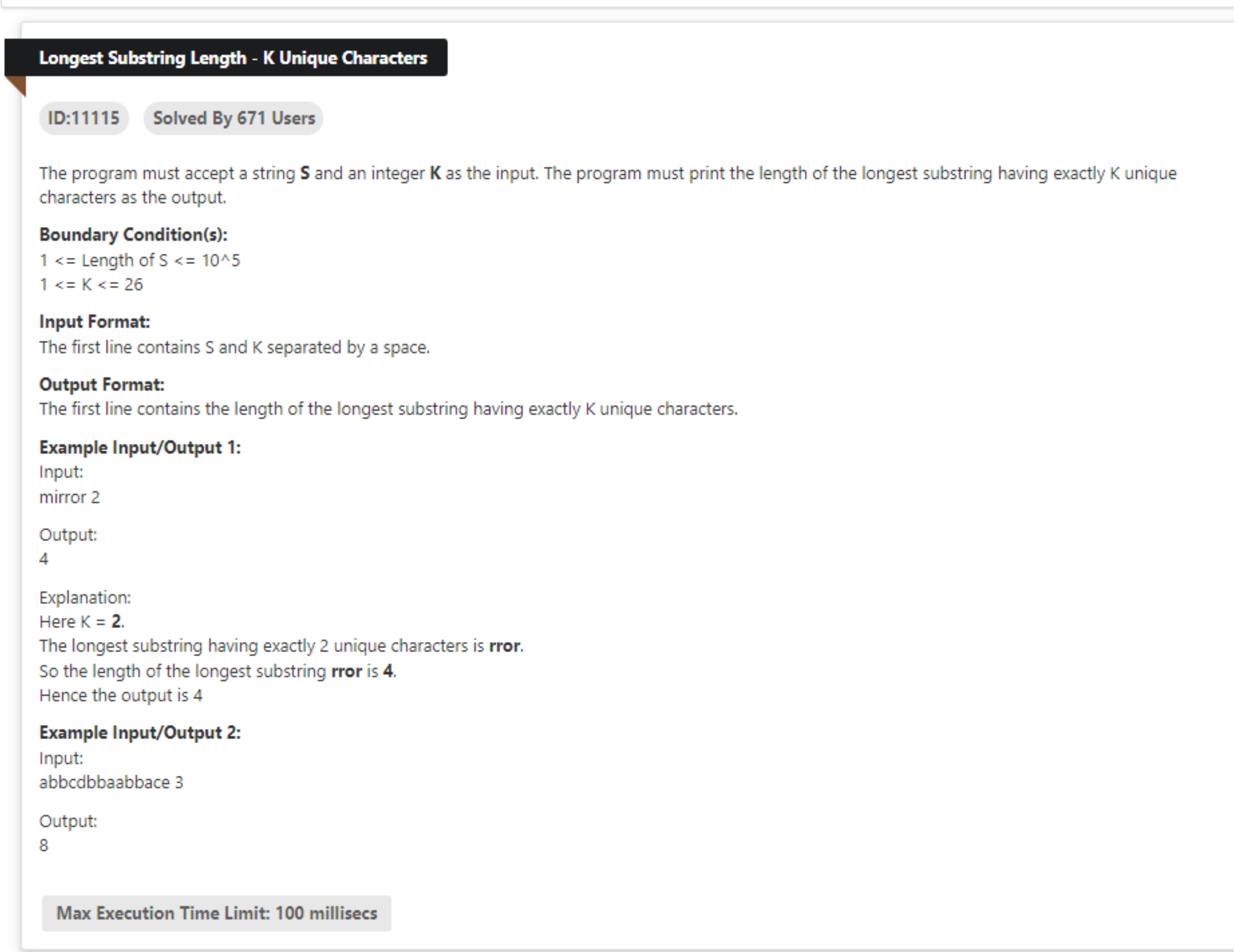
Code:

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

    public static void main(String[] args) {
        //Your Code Here
        Scanner in = new Scanner(System.in);
        int R=in.nextInt();
        int C=in.nextInt();

        int[][] rowSum= new int[R][C+1]; //if solving in c language we have to
initialize first column of the matrix to zero
```

```
        for(int row=0;row<R;row++){
            for(int col=1;col<=C;col++){
                int curr=in.nextInt();
                rowSum[row][col] = curr+rowSum[row][col-1]; //adding previous 
column value to the present column value each time
            }
        }
        int K=in.nextInt();

        int minSum = Integer.MAX_VALUE; //assigning max value
        for(int row=0;row<=R-K;row++){
            for(int col=1;col<=C-K+1;col++){
                int sum=0;
                for(int srow=row;srow<row+K;srow++){
                    sum+=rowSum[srow][col+K-1]-rowSum[srow][col-
1];  //subtracting 3th column value with 0th column value --one iteration
                }
                minSum =Math.min(minSum,sum);
            }
        }
        System.out.println(minSum);
    }
}
```

SESSION 4:

Problem 1:



Longest Substring Length - K Unique Characters

ID:11115 Solved By 671 Users

The program must accept a string **S** and an integer **K** as the input. The program must print the length of the longest substring having exactly K unique characters as the output.

Boundary Condition(s):
1 <= Length of S <= 10^5
1 <= K <= 26

Input Format:
The first line contains S and K separated by a space.

Output Format:
The first line contains the length of the longest substring having exactly K unique characters.

Example Input/Output 1:
Input:
mirror 2

Output:
4

Explanation:
Here K = **2**.
The longest substring having exactly 2 unique characters is **rror**.
So the length of the longest substring **rror** is **4**.
Hence the output is 4

Example Input/Output 2:
Input:
abbcdbbaabbace 3

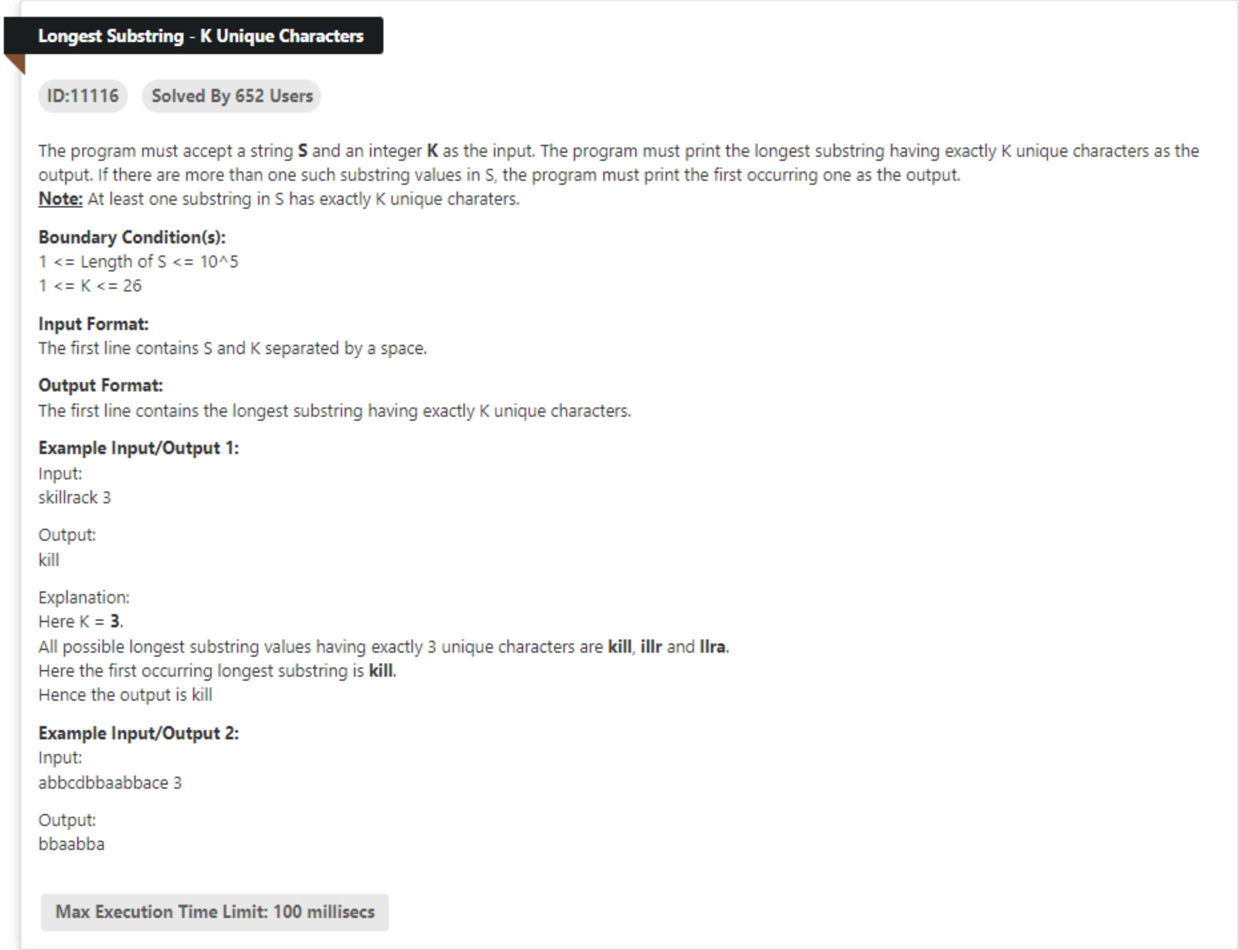
Output:
8

Max Execution Time Limit: 100 millisecs

Code:

```
#include<stdio.h>
#include<stdlib.h>

int main()
{
    char str[100000];
    int K;
    scanf("%s%d",str,&K);

    int start=0,end=0,unique=0,max=0;
    int len=strlen(str),arr[128]={0};
    arr[str[end]]=1;
    unique=1;
    while(end<len){
        if(K==unique){
            int curr=end-start+1;
            if(curr>max){
                max=curr;
            }
        }
        if(unique<=K){
            end++;
            arr[str[end]]++;
            if(arr[str[end]]==1){
                unique++;
            }
        }else{
            arr[str[start]]--;
            if(arr[str[start]]==0){
                unique--;
            }
            start++;
        }
    }
    printf("%d",max);
}
```

Problem 2:



Longest Substring - K Unique Characters

ID:11116 Solved By 652 Users

The program must accept a string **S** and an integer **K** as the input. The program must print the longest substring having exactly K unique characters as the output. If there are more than one such substring values in S, the program must print the first occurring one as the output.
Note: At least one substring in S has exactly K unique charaters.

Boundary Condition(s):
1 <= Length of S <= 10^5
1 <= K <= 26

Input Format:
The first line contains S and K separated by a space.

Output Format:
The first line contains the longest substring having exactly K unique characters.

Example Input/Output 1:
Input:
skillrack 3

Output:
kill

Explanation:
Here K = **3**.
All possible longest substring values having exactly 3 unique characters are **kill**, **illr** and **llra**.
Here the first occurring longest substring is **kill**.
Hence the output is kill

Example Input/Output 2:
Input:
abbcdbbaabbace 3

Output:
bbaabba

Max Execution Time Limit: 100 millisecs

Code:

```c
#include<stdio.h>
#include<stdlib.h>

int main()
{
    char str[100000];
    int K;
    scanf("%s%d",str,&K);
    int start=0,end=0,unique=0,max=0;
    int maxStart=0,maxEnd=0;
    int len=strlen(str),arr[128]={0};
    arr[str[end]]=1;
    unique=1;
    while(end<len){
        if(K==unique){
            int curr=end-start+1;
            if(curr>max){
                max=curr;
                maxStart=start;
                maxEnd=end;
            }
        }
        if(unique<=K){
            end++;
            arr[str[end]]++;
            if(arr[str[end]]==1){
                unique++;
            }
        }
        else{
            arr[str[start]]--;
            if(arr[str[start]]==0){
                unique--;
            }
            start++;
        }
    }
    for(int index=maxStart;index<=maxEnd;index++){
        printf("%c",str[index]);
    }
}
```