

SAP Labs Coding Assessment

Question1:

Question

The current selected programming language is **Java**. We emphasize the submission of a fully working code over partially correct but efficient code. Once **submitted**, you cannot review this problem again. You can use `System.out.println()` to debug your code. The `System.out.println()` may not work in case of syntax/runtime error. The version of **JDK** being used is **1.8**.

Note: The main class name must be **"Solution"**.

Write an algorithm to find the number of occurrences of *needle* in a given positive number *haystack*.

Input
The first line of the input consists of an integer *needle*, representing a digit.
The second line consists of an integer *haystack*, representing the positive number.

Output
Print an integer representing the number of occurrences of *needle* in *haystack*.

Constraints
 $0 \leq \text{needle} \leq 9$

```
1 import java.util.*;
2 import java.lang.*;
3 import java.io.*;
4
5 /*
6  * needle, representing a digit.
7  * haystack, representing the positive number.
8  */
9 public class Solution
10 {
11     public static int countOccurrences(int needle, int haystack)
12     {
13         int answer = 0;
14         // Write your code here
15         while(haystack>0){
16             int haystackNum=haystack%10;
17             if(haystackNum==needle) answer++;
18             haystack/=10;
19         }
20
21         return answer;
22     }
23
24     public static void main(String[] args)
25     {
26         Scanner in = new Scanner(System.in);
27         // input for needle
28         int needle = in.nextInt();
29
30     }
```

Test Cases & Output

RUN SOLUTION NEXT

Question 2:

Question

A company sells its products at *N* outlets. All the outlets are connected to each other by a series of roads. There is only one way to reach from one outlet to another. Each outlet of the company has a unique outlet ID. Whenever the inventory of a certain product reaches a minimum limit then these *K* outlets make a request for extra inventory. The company sends the requested products from its warehouse to the outlets. In order to save on fuel, the warehouse supervisor directs the driver Mike to deliver the products to the outlets along the shortest and most direct path possible, without traveling any single road twice.

Write an algorithm to help Mike deliver his inventory to the maximum number of outlets without traveling any road twice.

Input
The first line of the input consists of an integer - *num*, representing the total number of outlets of the company including the warehouse (*N*).
The second line consists of an integer - *outletsCount*, representing the outlets that

```
1 import java.util.*;
2 import java.lang.*;
3 import java.io.*;
4
5 /*
6  * num is the total number of outlets of the company including the warehouse.
7  * reqOutletsIDs is a list representing the outlet IDs of the outlets that requested the extra inventory.
8  * roadCon is a grid where each row represents the outlets connected by a road.
9  */
10 public class Solution
11 {
12     public static int maxOutlets(int num, int[] reqOutletsIDs, int[][] roadCon)
13     {
14         int answer = 0;
15         // Write your code here
16
17         return answer;
18     }
19
20     public static void main(String[] args)
21     {
22         Scanner in = new Scanner(System.in);
23         // input for num
24         int num = in.nextInt();
25
26         //input for reqOutletsIDs
27         int reqOutletsIDs_size = in.nextInt();
28         int reqOutletsIDs[] = new int[reqOutletsIDs_size];
29         for(int idx = 0; idx < reqOutletsIDs_size; idx++)
30         {
31             reqOutletsIDs[idx] = in.nextInt();
32         }
33
34         //input for roadCon
35         int roadCon_rows = in.nextInt();
36         int roadCon_cols = in.nextInt();
37         int roadCon[][] = new int[roadCon_rows][roadCon_cols];
38         for(int row = 0; row < roadCon_rows; row++)
39         {
40             for(int col = 0; col < roadCon_cols; col++)
41             {
42                 roadCon[row][col] = in.nextInt();
43             }
44         }
45     }
```

Test Cases & Output

RUN SOLUTION SUBMIT ASSESSMENT

SHL

01:28

Show tipsHelpAccessibilityExit

Question

Input

The first line of the input consists of an integer - *num*, representing the total number of outlets of the company including the warehouse (N).

The second line consists of an integer - *routesCount*, representing the outlets that requested the extra inventory (K).

The third line consists of K space-separated integers representing the outlet IDs of the outlets that requested the extra inventory.

The fourth line consists of two space-separated integers - *numR* and *conOutlet*, representing the total number of roads between two outlets including the warehouse (*numR* (M) is always equal to N-1) and number of outlets connected by a road (*conOutlet* (X) is always equal to 2).

The next M lines consists of X space-separated integers representing the road between two outlets.

Output

Print an integer representing the maximum number of outlets that Mike can cover in a single trip without traveling any road twice.

Constraints

$0 \leq \text{routesCount} \leq \text{num} \leq 10^5$

18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46

```
visitedOutlet.add(outlet);
backTrack(outlet,roadCon,visitedRoad, visitedOutlet,maxOutlets );
}
return maxOutlets[0];
}

public static void backTrack(int currentOutlet,int[][] roadCon, boolean[] visitedRoad, List<Integer> visitedOutlet, int[] maxOutlets){
    maxOutlets[0]=Math.max(maxOutlets[0], visitedOutlet.size());
    for(int i=0; i<roadCon.length;i++){
        if(!visitedRoad[i]){
            int outlet1=roadCon[i][0];
            int outlet2=roadCon[i][1];

            if(currentOutlet==outlet1 && !visitedOutlet.contains(outlet2)){
                visitedRoad[i]=true;
                visitedOutlet.add(outlet2);
                backTrack(outlet2, roadCon,visitedRoad,visitedOutlet,maxOutlets);
                if(visitedOutlet.isEmpty()){
                    visitedOutlet.remove(visitedOutlet.size()-1);
                    visitedRoad[i]=false;
                }
            }else if(currentOutlet==outlet2 && !visitedOutlet.contains(outlet1)){
                visitedRoad[i]=true;
                visitedOutlet.add(outlet1);
                backTrack(outlet1, roadCon,visitedRoad,visitedOutlet,maxOutlets);
                if(visitedOutlet.isEmpty()){
                    visitedOutlet.remove(visitedOutlet.size()-1);
                    visitedRoad[i]=false;
                }
            }
        }
    }
}
```

Test Cases & Output

RUN SOLUTIONSUBMIT ASSESSMENT

1

2