DAY 14:

SESSION 1:

Pet Store Dogs

ID:11149 Solved By 637 Users

There are **N** dogs in a pet store. The pet store wants to keep the N dogs in cages. There can be two dogs in each cage. A dog can be eitheir passive or aggressive. If the dog is passive, it can be clubed with another passive dog. If the dog is aggressive, it has to be alone in a cage. The program must print the number ways W to put the N dogs in the cages as the output.
Note: The number of cages in the pet store is sufficient to keep N dogs according to the given condition.

Boundary Condition(s):
1 <= N <= 1000

Input Format:
The first line contains N.

Output Format:
The first line contains W.

Example Input/Output 1:
Input:
2

Output:
2

Explanation:
Here N = 2.
Way 1: If both the dogs are passive, only one cage is required.
Way 2: If both the dogs are aggressive or one dog is passive or aggressive, two cages are required.

Example Input/Output 2:
Input:
3

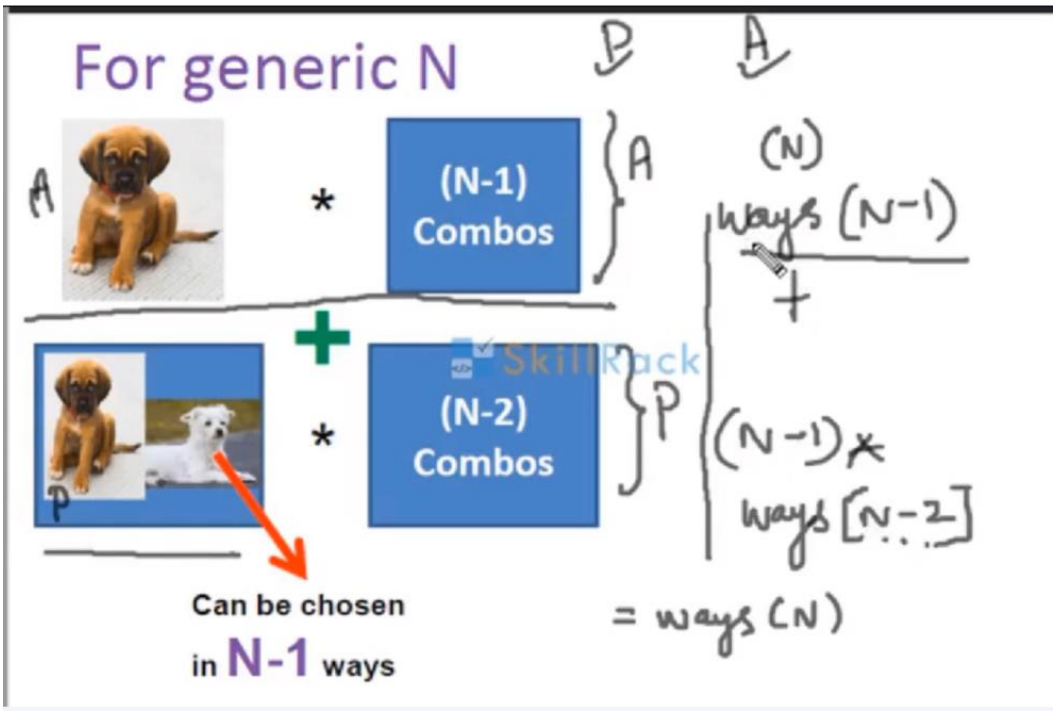
Output:
4

Example Input/Output 3:
Input:
10

Output:
9496

Max Execution Time Limit: 500 millisecs

Logic:



Code:

```
import java.util.*;
import java.math.*;

public class petStoreDogs {

    public static void main(String[] args) {
        // Your Code Here
        Scanner in = new Scanner(System.in);
        int N = in.nextInt();
        BigInteger ways[] = new BigInteger[N + 1];
        for (int ctr = 1; ctr <= N; ctr++) {
            if (ctr <= 2) { // for first two we have to initialize the value as
it will throw                               // IndexOutOfBoundException
                ways[ctr] = new BigInteger(ctr + "");
            } else {
                BigInteger passive = new BigInteger((ctr - 1) +
"").multiply(ways[ctr - 2]); // refer logic photo
                ways[ctr] = passive.add(ways[ctr - 1]);
            }
        }
        System.out.println(ways[N].toString());
    }
```

```
}
```

Session 2:

Minimum Edit Distance Two Strings

ID:11156 Solved By 635 Users

The program must accept two string values **S1** and **S2** as the input. The program must print the minimum cost required to convert the string S1 to S2 as the output. The cost of insertion, deletion and substitution of any character in the string S1 will be **1**.

Boundary Condition(s):
1 <= Length of S1, S2 <= 1000

Input Format:
The first line contains S1.
The second line contains S2.

Output Format:
The first line contains the minimum cost required to convert the string S1 to S2.

Example Input/Output 1:
Input:
hello
hail

Output:
3

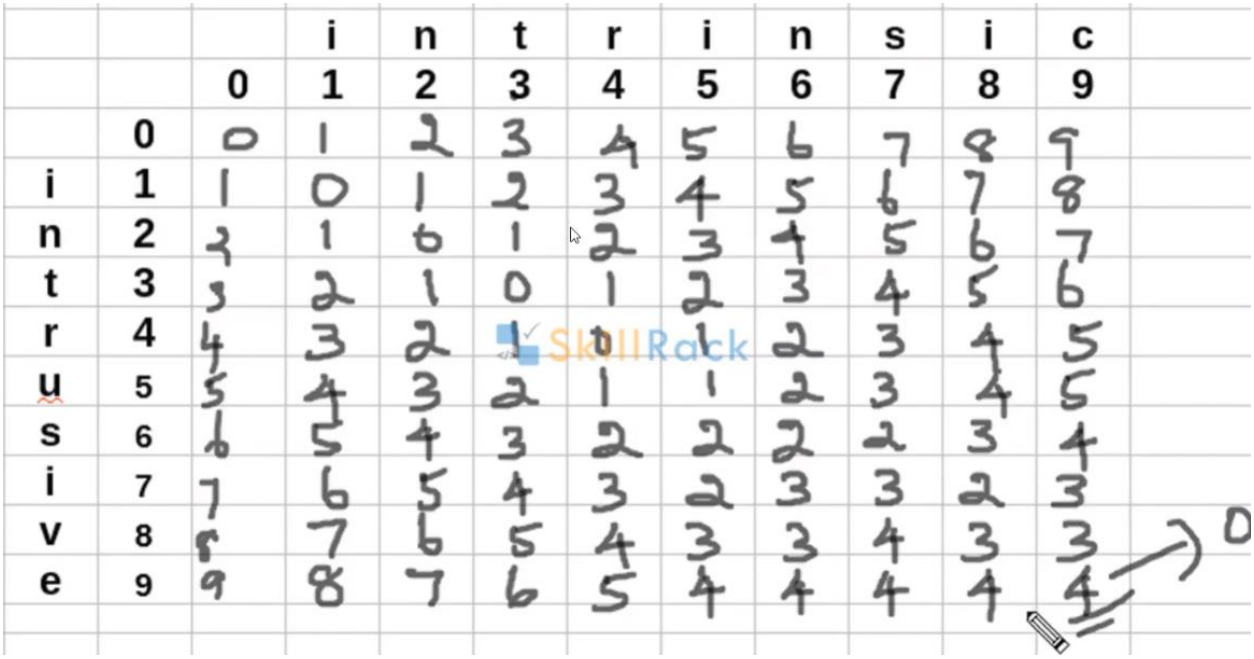
Explanation:
Here S1 = **hello** and S2 = **hail**
The minimum cost required to convert the string **hello** to **hail** is **3**.
The 3 operations are given below.
1 - **a** is substituted in place of **e**. Now the string S1 becomes **hallo**.
2 - **i** is substituted in place of **l** (first occurring l). Now the string S1 becomes **hailo**.
3 - **o** is deleted. Now the string S1 becomes **hail**.

Example Input/Output 2:
Input:
intrinsic
intrusive

Output:
4

Max Execution Time Limit: 500 millisecs

Logic:



Code:

```
import java.util.*;

public class minEditDistanceTwoStrings {

    public static void main(String[] args) {
        // Your Code Here
        Scanner in = new Scanner(System.in);
        String str1 = in.nextLine();
        String str2 = in.nextLine();
```

```
        int R = str1.length();
        int C = str2.length();

        int matrix[][] = new int[R + 1][C + 1];
        for (int row = 1; row <= R; row++) { //filling the row values
            matrix[row][0] = row;
        }
        for (int col = 0; col <= C; col++) { //filling the col values
            matrix[0][col] = col;
        }
        for (int row = 1; row <= R; row++) {
            for (int col = 1; col <= C; col++) {
                if (str1.charAt(row - 1) == str2.charAt(col - 1)) { //refer the
logic photo
                    matrix[row][col] = matrix[row - 1][col - 1];
                } else {
                    int top = matrix[row - 1][col];
                    int topleft = matrix[row - 1][col - 1];
                    int left = matrix[row][col - 1];
                    matrix[row][col] = Math.min(left, (Math.min(top, topleft))) +
1; //adding one refer photo
                }
            }
        }
        System.out.println(matrix[R][C]);
    }
}
```

Session 3:

Single Source Shortest Path

ID:11151 Solved By 603 Users

There are **N** cities in a country which are numbered from 1 to N. The N cities are connected by **L** links. Each link contains the **source** city, the **destination** city and the **distance** between them. The program must accept the values of N and L links as the input. The program must print the shortest distance from the city 1 to all the other cities as the output.

Boundary Condition(s):
2 <= N <= 100
1 <= L <= 1000
1 <= Distance between any two cities <= 10^5

Input Format:
The first line contains N and L separated by a space.
The next L lines, each containing three integers representing the source city, the destination city and the distance between them.

Output Format:
The first line contains N-1 integers representing the shortest distance from the city 1 to all the other cities.

Example Input/Output 1:
Input:
6 7
1 2 20
1 6 5
6 5 2
5 4 3
4 3 2
5 2 10
3 2 2

Output:
14 12 10 7 5

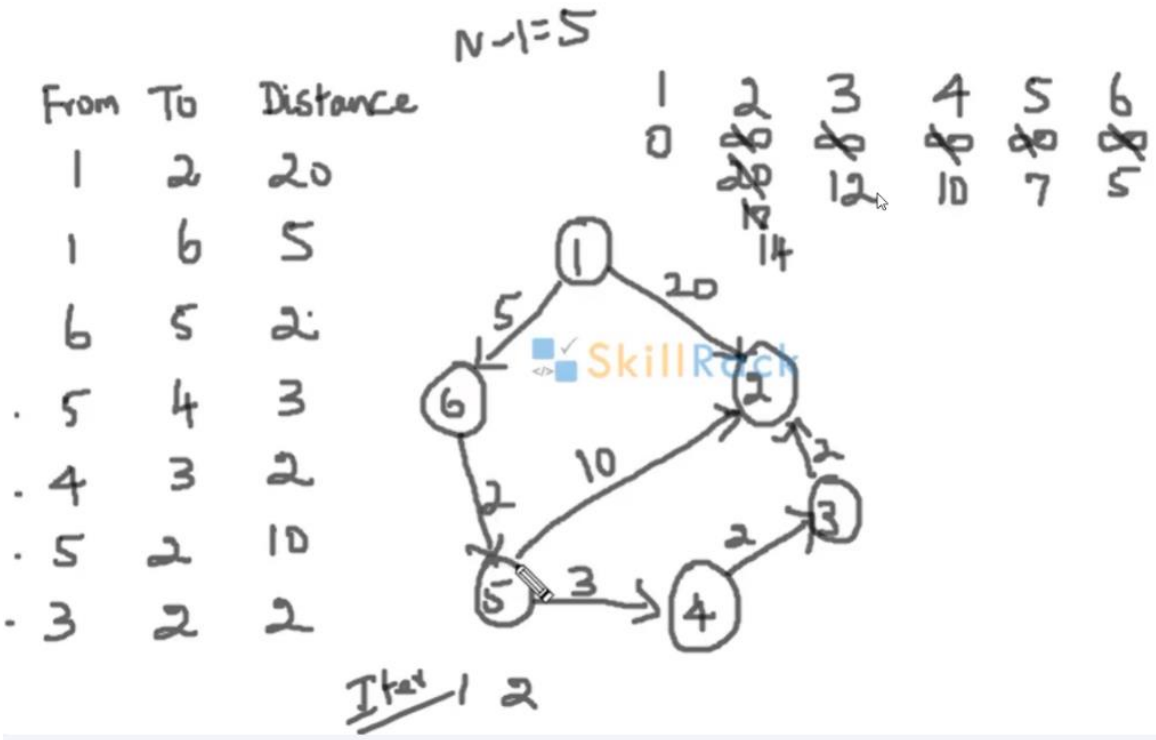
Explanation:
The shortest distance from the city **1** to **2** is **14** (1 -> 6 -> 5 -> 4 -> 3 -> 2).
The shortest distance from the city **1** to **3** is **12** (1 -> 6 -> 5 -> 4 -> 3).
The shortest distance from the city **1** to **4** is **10** (1 -> 6 -> 5 -> 4).
The shortest distance from the city **1** to **5** is **7** (1 -> 6 -> 5).
The shortest distance from the city **1** to **6** is **5** (1 -> 6).

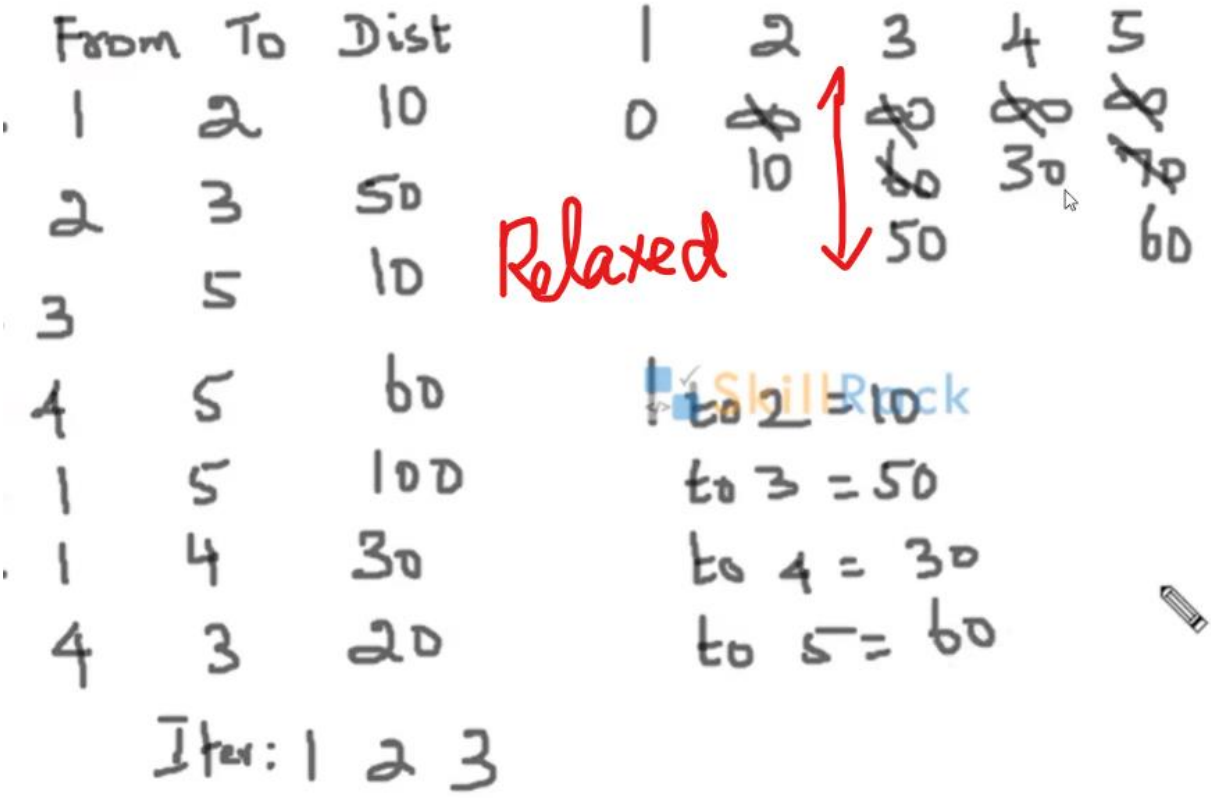
Example Input/Output 2:
Input:
5 7
1 2 10
2 3 50
3 5 10
4 5 60
1 5 100
1 4 30
4 3 20

Output:
10 50 30 60

Max Execution Time Limit: 500 millisecs

Logic:


N-1=5
From To Distance
1 2 20
1 6 5
6 5 2
5 4 3
4 3 2
5 2 10
3 2 2
1 2 3 4 5 6
0
20
14 12 10 7 5
Iter 1 2


From To Dist
1 2 10
2 3 50
3 5 10
4 5 60
1 5 100
1 4 30
4 3 20
Iter: 1 2 3
1 2 3 4 5
0
10 60 30 60
50
Relaxed
to 2 = 10
to 3 = 50
to 4 = 30
to 5 = 60

Code:

```
import java.util.*;
class Link{
    int source,dest,dist; //dest- destination, dist-distance
}
public class singleSourceShortestPath {
    public static void main(String[] args) {
        //Your Code Here
        Scanner in = new Scanner(System.in);
        int N=in.nextInt(),L=in.nextInt();
        List<Link> links = new ArrayList<>();
        for(int ctr=1;ctr<=L;ctr++){
            Link link = new Link(); //creating a class and an instance for that
class so we can add three attributes to the LIST
            link.source = in.nextInt();
            link.dest = in.nextInt();
            link.dist = in.nextInt();
            links.add(link);
        }
        Integer shortest[]  = new Integer[N+1];//Integer is a wrapper by default
java assign all values to null. in case of normal int array it assign all values
to zero
        shortest[1]=0; //distance from 1-1 is zero
        boolean relaxed = true; //relaxed means that no values changes after
iteration refer photo
        for(int iter=1;iter<N && relaxed;iter++){
            relaxed=false;
            for(Link link:links){
                if(shortest[link.source]==null){
                    continue;
                }
                if(shortest[link.dest]==null || shortest[link.source]+link.dist <
shortest[link.dest]){
                    shortest[link.dest] = shortest[link.source]+link.dist;
                    relaxed=true;
                }
            }
        }
        for(int city=2;city<=N;city++){ //not including the source so city starts
from 2
            System.out.print(shortest[city]+" ");
        }
    }
}
```