DAY 31

1. Minimum Cost Problem

You are provided with a map to go from one place to another in the form of a cost matrix named cost[][], where each cell represents the amount of time required to pass through that cell. The robot starts from position 0,0 and has to reach location m,n on the map. You have to find the minimum time required to reach the destination m,n. Each cell of the matrix represents a cost to traverse through that cell. The total cost of a path to reach (m, n) is the sum of all the costs on that path (including both the source and destination). You can only traverse down, right, and to diagonally lower cells to right from a given cell, i.e. from a given cell (i, j), cells (i + 1, j), (i, j + 1), and (i + 1, j + 1) can be traversed. You may assume that all the costs are positive integers.

Now, say you're given the following:

1	2	3
4	8	2
1	5	3

So, the minimum cost path for [2][2] is 1 + 2 + 2 + 3 = 8.

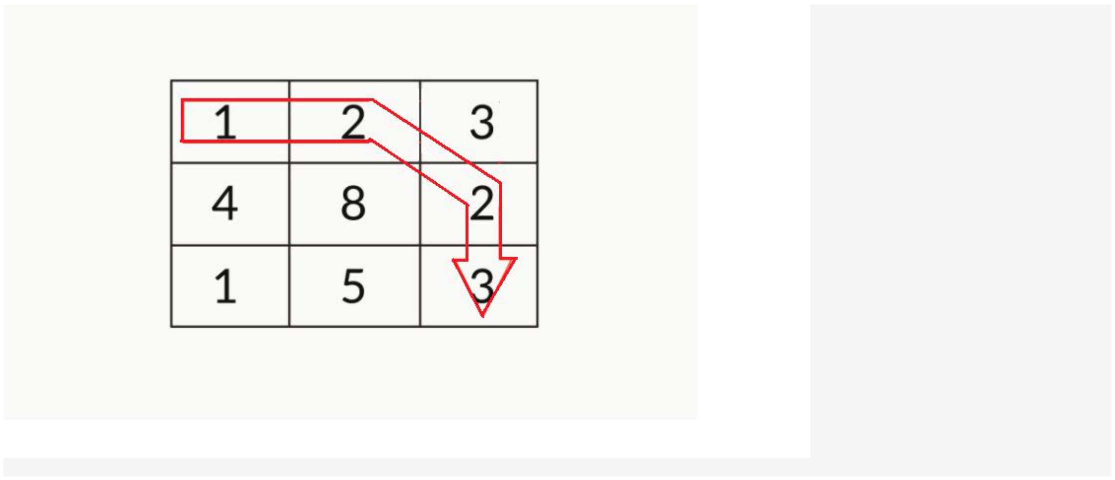

Image-MinimumCost

So, the problem is this: given a cost matrix cost[][] and a position (m, n) in cost[][], write a function that returns the cost of the minimum cost path, to reach (m, n) from (0, 0).

The input format should take values as the x coordinate of the destination in the first line followed by its y coordinate in the next.
The output should print the minimum cost to reach the destination taken from the input.

Sample input
1
1
Sample output
9

Sample input
2
2
Sample output
8

Note that matrix cost[][] with values has already been inserted.

CODE :

JAVA

```
import java.lang.*;
import java.util.Scanner;
import java.util.Arrays;
class Source
{

    private static int minCost(int cost[][], int m, int n)
    {
        int minCost[][]=new int[m+1][n+1];
        minCost[0][0]=cost[0][0];
        /* Initialise the first column of the minimum cost (table) array */
        for(int i=1;i<=m;i++){
            minCost[i][0]=cost[i][0]+minCost[i-1][0];
        }
        /* Initialise the first row of the table array */
        for(int j=1;j<=n;j++){
            minCost[0][j]=cost[0][j]+minCost[0][j-1];
        }
        /* Constructing the rest of the table array from the recursion relation */
        for(int i=1;i<=m;i++){
            for(int j=1;j<=n;j++){

minCost[i][j]=Math.min(Math.min(minCost[i-1][j-1],minCost[i-1][j]),minCost[i][j-1])+cost[i]
[j];
            }
        }
        return minCost[m][n];
    }

    public static void main(String args[])
    {
        int cost[][]= {{1, 2, 3},
                       {4, 8, 2},
                       {1, 5, 3}};
        Scanner scan = new Scanner(System.in);
        int xCoordinate = scan.nextInt();
        int yCoordinate = scan.nextInt();
        System.out.println(minCost(cost,xCoordinate,yCoordinate));
    }
}
```

2. Superhero Problem

Marvel is coming up with a new superhero named Jumping Jack. The co-creator of this superhero is a mathematician, and he adds a mathematical touch to the character's powers.

So, one of Jumping Jack's most prominent powers is jumping distances. But, this superpower comes with certain restrictions. Jumping Jack can only jump —
A. To the number that is one kilometre lesser than the current distance. For example, if he is 12 km away from the destination, he won't be able to jump directly to it since he can only jump to a location 11 km away.
B. To a distance that is half the current distance. For example, if Jumping Jack is 12 km away from the destination, again, he won't be able to jump directly to it since he can only jump to a location 6 km away.
C. To a distance that is ⅓rd the current distance. For example, if Jumping Jack is 12 km away from the destination, once more, he won't be able to jump directly to it since he can only jump to a location 4 km away.

So, you need to help the superhero develop an algorithm to reach the destination in the minimum number of steps. The destination is defined as the place where the distance becomes 1. Jumping Jack should cover the last 1 km running! Also, he can only jump to a destination that is an integer distance away from the main destination. For example, if he is at 10 km, by jumping 1/3rd the distance, he cannot reach 10/3rd the distance. He has to either jump to 5 or 9.
So, you have to find the minimum number of jumps required to reach a destination. For instance, if the destination is 10 km away, there are two ways to reach it:
10 -> 5 -> 4 -> 2 -> 1 (four jumps)
10 -> 9 -> 3 -> 1 (three jumps)
The minimum of these is three, so the superhero takes a minimum of three jumps to reach the point.
Note: Assume that the distance entered will be a positive integer always.

1. The input takes in the value of the location away from the destination.
2. The output displays the value of minimum needed to reach the destination.

Sample Input
6
Sample Output
2

CODE :

JAVA

```
import java.util.Scanner;
import java.util.Arrays;
class Source
{
    public static int minJump(int n)
    {
      /* Declare an array to store the minimum jumps. */
        int f[] = new int[n+1];
        /* Initialise the base condition */
        f[0]=0;
        f[1]=0;
        /* Fill in the array to find Minimum Jumps from a certain distance */
        for(int i=2;i<=n;i++){
            if(i%6==0){
                f[i]=Math.min(Math.min(f[i/2],f[i-1]),f[i/3])+1;
            }
            else if(i%3==0){
                f[i]=Math.min(f[i/3],f[i-1])+1;
            }
            else if(i%2==0){
                f[i]=Math.min(f[i/2],f[i-1])+1;
            }
            else{
                f[i]=1+f[i-1];
            }
        }
        return f[n];
    }

    public static void main (String args[])
    {
        Scanner scan = new Scanner(System.in);
        int distance = scan.nextInt();
        if(distance>0){
            System.out.println(minJump(distance));
        }
        else {
           System.out.println("Distance should be a positive integer");
        }
    }
}
```