Min Cost Path Problem

Send Feedback

An integer matrix of size (M x N) has been given. Find out the minimum cost to reach from the cell (0, 0) to (M - 1, N - 1).

From a cell (i, j), you can move in three directions:
1. ((i + 1), j) which is, "down"
2. (i, (j + 1)) which is, "to the right"
3. ((i+1), (j+1)) which is, "to the diagonal"

The cost of a path is defined as the sum of each cell's values through which the route passes.

Input format :

The first line of the test case contains two integer values, 'M' and 'N', separated by a single space. They represent the 'rows' and 'columns' respectively, for the two-dimensional array/list.

The second line onwards, the next 'M' lines or rows represent the ith row values.

Each of the ith row constitutes 'N' column values separated by a single space.

Output format :

Print the minimum cost to reach the destination.

Constraints :

1 <= M <= 10 ^ 2
1 <= N <= 10 ^ 2

Time Limit: 1 sec

Sample Input 1 :

3 4
3 4 1 2
2 1 8 9
4 7 8 1

Sample Output 1 :

13

Sample Input 2 :

3 4
10 6 9 0
-23 8 9 90
-200 0 89 200

Sample Output 2 :

76

Sample Input 3 :

5 6
9 6 0 12 90 1
2 7 8 5 78 6
1 6 0 5 10 -4
9 6 2 -10 7 4
10 -2 0 5 5 7

Sample Output 3 :

18

```
public class Solution {

	public static int minCostPath(int[][] input) {

		//Your code goes

    int m = input.length;

    int n = input[0].length;



    int [][] dp =new int[m+1][n+1];
```

```
        for(int i=0;i<dp.length;i++){
            for(int j=0;j<dp[0].length;j++){
                dp[i][j] = Integer.MAX_VALUE;
            }
        }
            for(int i=m-1;i>=0;i--){
            for(int j=n-1;j>=0;j--){
                if(i==m-1 && j==n-1){
                    dp[i][j] = input[i][j];
                    continue;
                }
                int ans1 = dp[i+1][j];
                int ans2 = dp[i][j+1];
                int ans3 = dp[i+1][j+1];

                dp[i][j]=input[i][j]+Math.min(ans1, Math.min(ans2, ans3));
                        }
        }
        return dp[0][0];
            }
}
```

LCS - Problem

Send Feedback

Given two strings, 'S' and 'T' with lengths 'M' and 'N', find the length of the 'Longest Common Subsequence'.

For a string 'str'(per se) of length K, the subsequences are the strings containing characters in the same relative order as they are present in 'str,' but not necessarily contiguous. Subsequences contain all the strings of length varying from 0 to K.

Example :

Subsequences of string "abc" are: ""(empty string), a, b, c, ab, bc, ac, abc.

Input format :

The first line of input contains the string 'S' of length 'M'.

The second line of the input contains the string 'T' of length 'N'.

Output format :

Return the length of the Longest Common Subsequence.

Constraints :

0 <= M <= 10 ^ 3
0 <= N <= 10 ^ 3

Time Limit: 1 sec

Sample Input 1 :

adebc
dcadb

Sample Output 1 :

3

Explanation of the Sample Output 1 :

Both the strings contain a common subsequence 'adb', which is the longest common subsequence with length 3.

Sample Input 2 :

ab
defg

Sample Output 2 :

0

Explanation of the Sample Output 2 :

The only subsequence that is common to both the given strings is an empty string("") of length 0.

```
public class Solution {

        public static int lcs(String s, String t) {

                //Your code goes here

    int[][] dp = new int[s.length()+1][t.length()+1];

    for (int i=0;i<dp.length;i++)

    {

       for (int j=0;j<dp[0].length;j++)

       {

          dp[i][j]=-1;

       }

    }

    return lcsHelper(s,0,t,0,dp);

}



private static int lcsHelper(String s, int i, String t, int j, int[][] dp)

{

    if (i==s.length() || j== t.length())

    {

       return 0;
```

```
    }
        if (s.charAt(i)==t.charAt(j))
    {
       if (dp[i+1][j+1]==-1)
       {
          dp[i+1][j+1]=lcsHelper(s,i+1,t,j+1,dp);
       }
       dp[i][j]=1+dp[i+1][j+1];
    }
    else
    {
       if(dp[i+1][j]==-1)
       {
          dp[i+1][j]=lcsHelper(s,i+1,t,j,dp);
       }
       int ans1=dp[i+1][j];

       if(dp[i][j+1]==-1)
       {
          dp[i][j+1]=lcsHelper(s,i,t,j+1,dp);
       }
       int ans2=dp[i][j+1];

       dp[i][j]=Math.max(ans1,ans2);
    }
    return dp[i][j];
  }     }
```

Edit Distance

You are given two strings S and T of lengths M and N, respectively. Find the 'Edit Distance' between the strings.

Edit Distance of two strings is the minimum number of steps required to make one string equal to the other. In order to do so, you can perform the following three operations:
1. Delete a character
2. Replace a character with another one
3. Insert a character

Note :

Strings don't contain spaces in between.

Input format :

The first line of input contains the string S of length M.

The second line of the input contains the String T of length N.

Output format :

Print the minimum 'Edit Distance' between the strings.

Constraints :

0 <= M <= 10 ^ 3
0 <= N <= 10 ^ 3

Time Limit: 1 sec

Sample Input 1 :

abc
dc

Sample Output 1 :

2

Explanation to the Sample Input 1 :

In 2 operations we can make string T to look like string S.
First, insert character 'a' to string T, which makes it "adc".

And secondly, replace the character 'd' of string T with 'b' from the string S. This would make string T as "abc" which is also string S.

Hence, the minimum distance.

Sample Input 2 :

whgtdwhgtdg
aswcfg

Sample Output 2 :

9

```
public class Solution {

public static int editDistance(String s, String t) {

        //Your code goes here

 //Find the lengths of both strings

int m=s.length();

int n=t.length();



int[][] dp = new int[m+1][n+1];

//Initializing dp for iterative approach

for (int i=n;i>=0;i--)

   dp[m][i]=n-i;

     for (int i=m;i>=0;i--)
```

```
    dp[i][n]=m-i;


for (int i=m-1;i>=0;i--)
{
   for (int j=n-1;j>=0;j--)
   {
      if (s.charAt(i)==t.charAt(j))
      {
         dp[i][j]=dp[i+1][j+1];
      }
      else
      {
         int ans1=1+dp[i+1][j+1];
         int ans2=1+dp[i][j+1];
         int ans3=1+dp[i+1][j];

         dp[i][j]=Math.min(ans1,Math.min(ans2,ans3));
      }
   }
}
return dp[0][0];
      }   }
```

0 1 Knapsack - Problem

Send Feedback

A thief robbing a store can carry a maximal weight of W into his knapsack. There are N items, and i-th item weigh 'Wi' and the value being 'Vi.' What would be the maximum value V, that the thief can steal?

Input Format :

The first line of the input contains an integer value N, which denotes the total number of items.

The second line of input contains the N number of weights separated by a single space.

The third line of input contains the N number of values separated by a single space.

The fourth line of the input contains an integer value W, which denotes the maximum weight the thief can steal.

Output Format :

Print the maximum value of V that the thief can steal.

Constraints :

1 <= N <= 20
1<= Wi <= 100
1 <= Vi <= 100

Time Limit: 1 sec

Sample Input 1 :

```
4
1 2 4 5
5 4 8 6
5
```

Sample Output 1 :

```
13
```

Sample Input 2 :

```
5
12 7 11 8 9
24 13 23 15 16
26
```

Sample Output 2 :

```
51
```

```
public class Solution {



        public static int knapsack(int[] weights, int[] values, int n, int maxWeight) {

                //Your code goes

    int [][] dp = new int[n+1][maxWeight+1];



    for(int i=n-1;i>=0;i--){

       for(int j=0;j<=maxWeight;j++){

          if(weights[i]<=j){

             int ans1 = dp[i+1][j];

             int ans2 = dp[i+1][j-weights[i]]+values[i];

             dp[i][j] = Math.max(ans1,ans2);

          }

          else{

             dp[i][j] = dp[i+1][j];

          }

       }

    }

    return dp[0][maxWeight];   }    }
```

Ways To Make Coin Change

Send Feedback

For the given infinite supply of coins of each of denominations, D = {D0, D1, D2, D3, ... Dn-1}. You need to figure out the total number of ways W, in which you can make the change for Value V using coins of denominations D.

Return 0 if the change isn't possible.

Input Format

The first line of the input contains an integer value N, which denotes the total number of denominations.

The second line of input contains N values, separated by a single space. These values denote the value of denomination.

The third line of the input contains an integer value, that denotes the value of V.

Output Format

Print the total total number of ways i.e. W.

Constraints :

1 <= n <= 10

1 <= V <= 1000

Time Limit: 1sec

Sample Input 1 :

```
3
1 2 3
4
```

Sample Output 1 :

```
4
```

Explanation to Sample Input 1 :

Number of ways are - 4 total i.e. (1,1,1,1), (1,1, 2), (1, 3) and (2, 2).

Sample Input 2 :

```
6
1 2 3 4 5 6
250
```

Sample Output 2 :

```
13868903
```

```
public class Solution {

		public static int countWaysToMakeChange(int denominations[], int value){

	// Write your code here

	int[] dp = new int[value+1];

	dp[0]=1;

	for (int i=0;i<denominations.length;i++)

	{

	   for (int j=0;j<=value;j++)

	   {

	      if (j>=denominations[i])

	      {

	         dp[j]=dp[j]+dp[j-denominations[i]];
```

```
        }       }     }
    return dp[value];
        }
        }
```

Assignment

Maximum Square Matrix With All Zeros

Send Feedback

Given an NxM matrix that contains only 0s and 1s, find out the size of the maximum square sub-matrix with all 0s. You need to return the size of the square matrix with all 0s.

Input format :

The first line of the test case contains two integer values, 'N' and 'M', separated by a single space. They represent the 'rows' and 'columns' respectively.

Second-line onwards, the next 'N' lines or rows represent the ith row values.

Each of the ith rows constitutes column values separated by a single space (Either 0 or 1).

Output Format:

Print the size of maximum square sub-matrix.

Constraints :

0 <= N <= 10^4
0 <= M <= 10^4

Time Limit: 1 sec

Sample Input 1:

```
3 3
1 1 0
1 1 1
1 1 1
```

Sample Output 1:

```
1
```

Sample Input 2:

```
4 4
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
```

Sample Output 2:

```
4
```

```
public class Solution {

    public static int findMaxSquareWithAllZeros(int[][] input){

    /* Your class should be named Solution.
     * Don't write main() function.
     * Don't read input, it is passed as function argument.
```

```
        * Return output and don't print it.
        * Taking input and printing output is handled automatically.
        */
int m = input.length;
if(m==0)
   return 0;

int n = input[0].length;
if(n==0)
   return 0;

int [][] dp = new int[m][n];
int maxVal = 0;

for(int i=0;i<n;i++){
   if(input[0][i]==0){
      dp[0][i]= 1;
   }
}
for(int i=0;i<m;i++){
   if(dp[i][0]==0){
      dp[i][0]= 1;
   }
}

for(int i=1;i<m;i++){
   for(int j=1;j<n;j++){
      if(input[i][j]==0){
       int ans1=dp[i-1][j];
```

```
            int ans2=dp[i][j-1];

            int ans3=dp[i-1][j-1];


            dp[i][j]=Math.min(ans1,Math.min(ans2,ans3))+1;

           }

           if (dp[i][j]>maxVal)

          maxVal=dp[i][j];

       }

    }

    return maxVal;


         }          }
```

Smallest Super-Sequence

Send Feedback

Given two strings S and T with lengths M and N. Find and return the length of their shortest 'Super Sequence'.

The shortest 'Super Sequence' of two strings is defined as the smallest string possible that will have both the given strings as its subsequences.

Note :

If the two strings do not have any common characters, then return the sum of the lengths of the two strings.

Input Format:

The first only line of input contains a string, that denotes the value of string S. The following line contains a string, that denotes the value of string T.

Output Format:

Length of the smallest super-sequence of given two strings.

Constraints :

0 <= M <= 10 ^ 3
0 <= N <= 10 ^ 3

Time Limit: 1 sec

Sample Input 1 :

ab
ac

Sample Output 1 :

3

Explanation of Sample Output 1 :

Their smallest super sequence can be "abc" which has length = 3.

Sample Input 2 :

pqqrpt
qerepct

Sample Output 2 :

9

```
public class Solution {
```

```
    public static int smallestSuperSequence(String str1, String str2) {


/* Your class should be named Solution
                * Don't write main().
                * Don't read input, it is passed as function argument.
                * Return output and don't print it.
                * Taking input and printing output is handled automatically.
*/
int n = str1.length();
int m = str2.length();


int [][] dp = new int[n+1][m+1];
for(int i=n;i>=0;i--){
   dp[i][m] = n-i;
}
for(int i=m;i>=0;i--){
   dp[n][i] = m-i;
}


for(int i=n-1;i>=0;i--){
   for(int j=m-1;j>=0;j--){
      if(str1.charAt(i)==str2.charAt(j)){
         dp[i][j] = dp[i+1][j+1]+1;
      }
      else{
         int ans1=dp[i+1][j];
         int ans2=dp[i][j+1];


         dp[i][j]=Math.min(ans1,ans2)+1;
```

```
            }

          }

        }

        return dp[0][0];

}       }
```

Magic Grid

Send Feedback

You are given a magic grid A with R rows and C columns. In the cells of the grid, you will either get magic juice, which increases your strength by |A[i][j]| points, or poison, which takes away |A[i][j]| strength points. If at any point of the journey, the strength points become less than or equal to zero, then you will die.

You have to start from the top-left corner cell (1,1) and reach at the bottom-right corner cell (R,C). From a cell (i,j), you can only move either one cell down or right i.e., to cell (i+1,j) or cell (i,j+1) and you can not move outside the magic grid. You have to find the minimum number of strength points with which you will be able to reach the destination cell.

Input format:

The first line contains the number of test cases T. T cases follow. Each test case consists of R C in the first line followed by the description of the grid in R lines, each containing C integers. Rows are numbered 1 to R from top to bottom and columns are numbered 1 to C from left to right. Cells with A[i][j] < 0 contain poison, others contain magic juice.

Output format:

Output T lines, one for each case containing the minimum strength you should start with from the cell (1,1) to have a positive strength through out his journey to the cell (R,C).

Constraints:

1 ≤ T ≤ 5
2 ≤ R, C ≤ 500
-10^3 ≤ A[i][j] ≤ 10^3
A[1][1] = A[R][C] = 0
Time Limit: 1 second

Sample Input 1:

```
3
2 3
0 1 -3
1 -2 0
2 2
0 1
2 0
3 4
0 -2 -3 1
-1 4 0 -2
1 -2 -3 0
```

Sample Output 1:

```
2
1
2
```

```
public class Solution{


   public static int getMinimumStrength(int[][] grid) {
```

```
/* Your class should be named Solution
                * Don't write main().
                * Don't read input, it is passed as function argument.
                * Return output and don't print it.
                * Taking input and printing output is handled automatically.
*/
int row=grid.length;
if (row==0)
   return row;

int col=grid[0].length;
if (col==0)
   return col;

int[][] dp=new int[row][col];
dp[row-1][col-1]=1;

for (int i=col-2;i>=0;i--)
{
     dp[row-1][i]=dp[row-1][i+1]-grid[row-1][i];

}
for (int i=row-2;i>=0;i--)
{
   dp[i][col-1]=dp[i+1][col-1]-grid[i][col-1];
}

for(int i=row-2;i>=0;i--)
```

```
    {
        for (int j=col-2;j>=0;j--)
        {
            int ans1=dp[i+1][j];
            int ans2=dp[i][j+1];

            dp[i][j]=Math.max(1,Math.min(ans1,ans2)-grid[i][j]);
        }
    }

    return dp[0][0];

  }
}
```

Minimum Number of Chocolates

Send Feedback

Miss. Noor Rashid is a teacher. She wants to give some chocolates to the students in her class. All the students sit in a line, and each of them has a score according to performance. Noor wants to give at least one chocolate to each student. She distributes chocolates to them such that If two students sit next to each other, then the one with the higher score must get more chocolates. Miss. Noor wants to save money, so she wants to minimize the total number of chocolates.

Note :

When two students have an equal score, they are allowed to have a different number of chocolates.

Input Format:

The first line of the input contains an integer value of N. It denotes the total number of students in Noor's class.

The second line of the input contains N integer values denoting the score of each of the students. A single space will separate them.

Output Format:

Print the minimum number of chocolates Noor must give.

Constraints

1 <= N <= 10^5
1 <= score <= 10^5

Time Limit: 1 sec

Sample Input 1 :

```
4
1 4 4 6
```

Sample Output 1 :

```
6
```

Explanation:

One of the ways in which the chocolates can be distributed, such Noor has to give minimum number of chocolates, are: The first student can be given one chocolate, second student can be given two chocolates, third student can be one chocolate and fourth can be given two chocolates.

Sample Input 2 :

```
3
8 7 5
```

Sample Output 2 :

```
6
```

```
public class Solution {



        public static int getMin(int arr[], int N){



    /* Your class should be named Solution

                * Don't write main().

                * Don't read input, it is passed as function argument.

                * Return output and don't print it.

                * Taking input and printing output is handled automatically.

    */

    int[] arr1 = new int[N];



    // Distribute 1 chocolate to each

    for (int i = 0; i < N; i++) {

       arr1[i] = 1;

    }



    // Traverse from left to right

    for (int i = 1; i < N; i++) {

       if (arr[i] > arr[i - 1])

          arr1[i] = arr1[i - 1] + 1;

       else

          arr1[i] = 1;

    }
```

```
    // Traverse from right to left
    for (int i = N - 2; i >= 0; i--) {
      if (arr[i] > arr[i + 1])
        arr1[i] = Math.max(arr1[i + 1] + 1, arr1[i]);
      else
        arr1[i] = Math.max(arr1[i], 1);
    }

    // Initialize sum
    int sum = 0;

    // Find total sum
    for (int i = 0; i < N; i++) {
      sum += arr1[i];
    }

     return sum;
    // System.out.print(sum + "\n");

  }
}
```

Subset Sum

Send Feedback

You are given a set of N integers. You have to return true if there exists a subset that sum up to K, otherwise return false.

Input Format

The first line of the test case contains an integer 'N' representing the total elements in the set.

The second line of the input contains N integers separated by a single space.

The third line of the input contains a single integer, denoting the value of K.

Output Format

Output Yes if there exists a subset whose sum is k, else output No.

Constraints :

1 <= N <= 10^3
1 <= a[i] <= 10^3, where a[i] denotes an element of the set
1 <= K <= 10^3

Time Limit: 1 sec

Sample Input 1 :

```
4
4 3 5 2
13
```

Sample Output 1 :

```
No
```

Sample Input 2 :

```
5
4 2 5 6 7
14
```

Sample Output 2 :

```
Yes
```

```
public class Solution{

   static boolean isSubsetPresent(int[] arr, int n, int sum) {



      /* Your class should be named Solution

                  * Don't write main().

                  * Don't read input, it is passed as function argument.

                  * Return output and don't print it.

                  * Taking input and printing output is handled automatically.

      */

      if(sum==0)

         return true;

      if(n==0)

         return false;



      if(arr[n-1]>sum){

         return isSubsetPresent(arr,n-1,sum);

      }

      return isSubsetPresent(arr, n - 1, sum)

         || isSubsetPresent(arr, n - 1, sum - arr[n - 1]);



   }

}
```