Sudo Placement 2
Quick Links for Dynamic Programming Algorithms
Recent Articles
Practice Problems
Quizzes
Videos
Basic Concepts
Tabulation vs Memoizatation
Overlapping Subproblems Property
Optimal Substructure Property
How to solve a Dynamic Programming Problem ?
Explore more...
Advanced Concepts
Bitmasking and Dynamic Programming | Set 1 (Count ways to assign unique cap to every person)
Bitmasking and Dynamic Programming | Set-2 (TSP)
Digit DP | Introduction
Explore more...
Basic Problems
Ugly Numbers & Fibonacci numbers
nth Catalan Number & Bell Numbers
Binomial Coefficient & Permutation Coefficient
Tiling Problem & Gold Mine Problem & Coin Change
Friends Pairing Problem & Subset Sum Problem
Subset Sum Problem in O(sum) space & Subset with sum divisible by m
Largest divisible pairs subset & Perfect Sum Problem
Compute nCr % p & Choice of Area
Cutting a Rod & Painting Fence Algorithm
Tiling with Dominoes
Golomb sequence
Moser-de Bruijn Sequence
Newman-Conway Sequence
maximum length Snake sequence
Print Fibonacci sequence using 2 variables
Longest Repeated Subsequence
Explore more...
Intermediate Problems
Lobb Number & Eulerian Number
Delannoy Number & Entringer Number
Rencontres Number & Jacobsthal and Jacobsthal-Lucas numbers
Super Ugly Number &Floyd Warshall Algorithm
Bellman–Ford Algorithm & 0-1 Knapsack Problem

Printing Items in 0/1 Knapsack & Unbounded Knapsack
Temple Offerings & Egg Dropping Puzzle
Dice Throw & Word Break Problem
Vertex Cover Problem & Tile Stacking Problem
Box Stacking Problem
Highway Billboard Problem & Largest Independent Set Problem
Print equal sum sets of array (Partition problem) | Set 1 & Set 2
Longest Bitonic Subsequence
Printing Longest Bitonic Subsequence
Print Longest Palindromic Subsequence
Longest palindrome subsequence with O(n) space
Explore more...
Hard Problems
Palindrome Partitioning & Word Wrap Problem
Mobile Numeric Keypad Problem & The painter's partition problem
Boolean Parenthesization Problem & Bridge and Torch problem
A Space Optimized DP solution for 0-1 Knapsack Problem
Matrix Chain Multiplication & Printing brackets in Matrix Chain Multiplication Problem
Number of palindromic paths in a matrix
Largest rectangular sub-matrix whose sum is 0
Largest rectangular sub-matrix having sum divisible by k
Maximum sum bitonic subarray
K maximum sums of overlapping contiguous sub-arrays
Maximum profit by buying and selling a share at most k times
Maximum points from top left of matrix to bottom right and return back
Check whether row or column swaps produce maximum size binary sub-matrix with all 1s
Minimum number of elements which are not part of Increasing or decreasing subsequence in array
Count ways to increase LCS length of two strings by one
Count of AP (Arithmetic Progression) Subsequences in an array
Explore more...

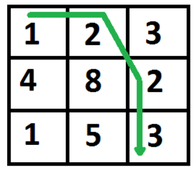
Min Cost Path | DP-6

Given a cost matrix cost[][] and a position (m, n) in cost[][], write a function that returns cost of minimum cost path to reach (m, n) from (0, 0). Each cell of the matrix represents a cost to traverse through that cell. Total cost of a path to reach (m, n) is sum of all the costs on that path (including both source and destination). You can only traverse down, right and diagonally lower cells 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 costs are positive integers.

For example, in the following figure, what is the minimum cost path to (2, 2)?

1	2	3
4	8	2
1	5	3

The path with minimum cost is highlighted in the following figure. The path is (0, 0) –> (0, 1) –> (1, 2) –> (2, 2). The cost of the path is 8 (1 + 2 + 2 + 3).



Recommended: Please solve it on "*PRACTICE*" first, before moving on to the solution.

1) Optimal Substructure

The path to reach (m, n) must be through one of the 3 cells: (m-1, n-1) or (m-1, n) or (m, n-1). So minimum cost to reach (m, n) can be written as "minimum of the 3 cells plus cost[m][n]".

minCost(m, n) = min (minCost(m-1, n-1), minCost(m-1, n), minCost(m, n-1)) + cost[m][n]

2) Overlapping Subproblems

Following is simple recursive implementation of the MCP (Minimum Cost Path) problem. The implementation simply follows the recursive structure mentioned above.

C

```c
/* A Naive recursive implementation of MCP(Minimum Cost Path) problem */
#include<stdio.h>
#include<limits.h>
#define R 3
#define C 3

int min(int x, int y, int z);

/* Returns cost of minimum cost path from (0,0) to (m, n) in mat[R][C]*/
int minCost(int cost[R][C], int m, int n)
{
   if (n < 0 || m < 0)
      return INT_MAX;
   else if (m == 0 && n == 0)
      return cost[m][n];
   else
      return cost[m][n] + min( minCost(cost, m-1, n-1),
                               minCost(cost, m-1, n),
                               minCost(cost, m, n-1) );
}

/* A utility function that returns minimum of 3 integers */
int min(int x, int y, int z)
{
   if (x < y)
      return (x < z)? x : z;
   else
      return (y < z)? y : z;
}

/* Driver program to test above functions */
int main()
{
   int cost[R][C] = { {1, 2, 3},
                      {4, 8, 2},
                      {1, 5, 3} };
   printf(" %d ", minCost(cost, 2, 2));
   return 0;
}
```

Run on IDE

Java

```java
/* A Naive recursive implementation of
MCP(Minimum Cost Path) problem */
public class GFG {

    /* A utility function that returns
    minimum of 3 integers */
    static int min(int x, int y, int z)
    {
        if (x < y)
            return (x < z) ? x : z;
        else
            return (y < z) ? y : z;
    }

    /* Returns cost of minimum cost path
    from (0,0) to (m, n) in mat[R][C]*/
    static int minCost(int cost[][], int m,
                                     int n)
    {
        if (n < 0 || m < 0)
            return Integer.MAX_VALUE;
        else if (m == 0 && n == 0)
            return cost[m][n];
        else
            return cost[m][n] +
                min( minCost(cost, m-1, n-1),
                     minCost(cost, m-1, n),
                     minCost(cost, m, n-1) );
```

```
    }

    // Driver code
    public static void main(String args[])
    {

        int cost[][] = { {1, 2, 3},
                         {4, 8, 2},
                         {1, 5, 3} };

        System.out.print(minCost(cost, 2, 2));
    }
}

// This code is contributed by Sam007
```

Run on IDE

Python3

```
# A Naive recursive implementation of MCP(Minimum Cost Path) problem
R = 3
C = 3
import sys

# Returns cost of minimum cost path from (0,0) to (m, n) in mat[R][C]
def minCost(cost, m, n):
    if (n < 0 or m < 0):
        return sys.maxsize
    elif (m == 0 and n == 0):
        return cost[m][n]
    else:
        return cost[m][n] + min( minCost(cost, m-1, n-1),
                                 minCost(cost, m-1, n),
                                 minCost(cost, m, n-1) )

#A utility function that returns minimum of 3 integers */
def min(x, y, z):
    if (x < y):
        return x if (x < z) else z
    else:
        return y if (y < z) else z


# Driver program to test above functions
cost= [ [1, 2, 3],
        [4, 8, 2],
        [1, 5, 3] ]
print(minCost(cost, 2, 2))

# This code is contributed by
# Smitha Dinesh Semwal
```

Run on IDE

C#

```
/* A Naive recursive implementation of
MCP(Minimum Cost Path) problem */
using System;

class GFG
{

    /* A utility function that
    returns minimum of 3 integers */
    static int min(int x,
                   int y, int z)
    {
        if (x < y)
            return ((x < z) ? x : z);
        else
            return ((y < z) ? y : z);
    }

    /* Returns cost of minimum
    cost path from (0,0) to
    (m, n) in mat[R][C]*/
    static int minCost(int [,]cost,
                       int m , int n)
    {
        if (n < 0 || m < 0)
            return int.MaxValue;
        else if (m == 0 && n == 0)
            return cost[m, n];
        else
            return cost[m, n] +
                   min(minCost(cost, m - 1, n - 1),
                   minCost(cost, m - 1, n),
                   minCost(cost, m, n - 1) );
    }

    // Driver code
    public static void Main()
    {

        int [,]cost = {{1, 2, 3},
                       {4, 8, 2},
                       {1, 5, 3}};

        Console.Write(minCost(cost, 2, 2));
    }
}
```

```
// This code is contributed
// by shiv_bhakt.
```

Run on IDE

PHP

```
<?php
/* A Naive recursive implementation
of MCP(Minimum Cost Path) problem */

$R = 3;
$C = 3;

/* Returns cost of minimum
cost path from (0,0) to
(m, n) in mat[R][C]*/
function minCost($cost, $m, $n)
{
global $R;
global $C;
if ($n < 0 || $m < 0)
    return PHP_INT_MAX;
else if ($m == 0 && $n == 0)
    return $cost[$m][$n];
else
    return $cost[$m][$n] +
           min1(minCost($cost, $m - 1, $n - 1),
           minCost($cost, $m - 1, $n),
           minCost($cost, $m, $n - 1) );
}

/* A utility function that
returns minimum of 3 integers */
function min1($x, $y, $z)
{
if ($x < $y)
    return ($x < $z)? $x : $z;
else
    return ($y < $z)? $y : $z;
}

// Driver Code
$cost = array(array(1, 2, 3),
              array (4, 8, 2),
              array (1, 5, 3));
echo minCost($cost, 2, 2);

// This code is contributed by mits.
?>
```

Run on IDE

Output:

```
8
```

It should be noted that the above function computes the same subproblems again and again. See the following recursion tree, there are many nodes which apear more than once. Time complexity of this naive recursive solution is exponential and it is terribly slow.

```
mC refers to minCost()
                                  mC(2, 2)
                          /          |          \
                         /           |           \
               mC(1, 1)          mC(1, 2)            mC(2, 1)
            /     |     \      /     |     \          /     |     \
           /      |      \    /      |      \        /      |      \
      mC(0,0) mC(0,1) mC(1,0) mC(0,1) mC(0,2) mC(1,1) mC(1,0) mC(1,1) mC(2,0)
```

So the MCP problem has both properties (see this and this) of a dynamic programming problem. Like other typical Dynamic Programming(DP) problems, recomputations of same subproblems can be avoided by constructing a temporary array tc[][] in bottom up manner.

C++

```
/* Dynamic Programming implementation of MCP problem */
#include<stdio.h>
#include<limits.h>
#define R 3
#define C 3

int min(int x, int y, int z);

int minCost(int cost[R][C], int m, int n)
{
     int i, j;

     // Instead of following line, we can use int tc[m+1][n+1] or
     // dynamically allocate memory to save space. The following line is
     // used to keep the program simple and make it working on all compilers.
     int tc[R][C];

     tc[0][0] = cost[0][0];

     /* Initialize first column of total cost(tc) array */
     for (i = 1; i <= m; i++)
        tc[i][0] = tc[i-1][0] + cost[i][0];
```

```
    /* Initialize first row of tc array */
    for (j = 1; j <= n; j++)
       tc[0][j] = tc[0][j-1] + cost[0][j];

    /* Construct rest of the tc array */
    for (i = 1; i <= m; i++)
       for (j = 1; j <= n; j++)
           tc[i][j] = min(tc[i-1][j-1],
                          tc[i-1][j],
                          tc[i][j-1]) + cost[i][j];

    return tc[m][n];
}

/* A utility function that returns minimum of 3 integers */
int min(int x, int y, int z)
{
   if (x < y)
      return (x < z)? x : z;
   else
      return (y < z)? y : z;
}

/* Driver program to test above functions */
int main()
{
   int cost[R][C] = { {1, 2, 3},
                      {4, 8, 2},
                      {1, 5, 3} };
   printf(" %d ", minCost(cost, 2, 2));
   return 0;
}
```

Run on IDE

Java

```
/* Java program for Dynamic Programming implementation
   of Min Cost Path problem */
import java.util.*;

class MinimumCostPath
{
    /* A utility function that returns minimum of 3 integers */
    private static int min(int x, int y, int z)
    {
        if (x < y)
            return (x < z)? x : z;
        else
            return (y < z)? y : z;
    }

    private static int minCost(int cost[][], int m, int n)
    {
        int i, j;
        int tc[][]=new int[m+1][n+1];

        tc[0][0] = cost[0][0];

        /* Initialize first column of total cost(tc) array */
        for (i = 1; i <= m; i++)
            tc[i][0] = tc[i-1][0] + cost[i][0];

        /* Initialize first row of tc array */
        for (j = 1; j <= n; j++)
            tc[0][j] = tc[0][j-1] + cost[0][j];

        /* Construct rest of the tc array */
        for (i = 1; i <= m; i++)
            for (j = 1; j <= n; j++)
                tc[i][j] = min(tc[i-1][j-1],
                               tc[i-1][j],
                               tc[i][j-1]) + cost[i][j];

        return tc[m][n];
    }

    /* Driver program to test above functions */
    public static void main(String args[])
    {
        int cost[][]= {{1, 2, 3},
                       {4, 8, 2},
                       {1, 5, 3}};
        System.out.println(minCost(cost,2,2));
    }
}
// This code is contributed by Pankaj Kumar
```

Run on IDE

Python

```
# Dynamic Programming Python implementation of Min Cost Path
# problem
R = 3
C = 3

def minCost(cost, m, n):

    # Instead of following line, we can use int tc[m+1][n+1] or
    # dynamically allocate memoery to save space. The following
    # line is used to keep te program simple and make it working
    # on all compilers.
```

```
    tc = [[0 for x in range(C)] for x in range(R)]

    tc[0][0] = cost[0][0]

    # Initialize first column of total cost(tc) array
    for i in range(1, m+1):
        tc[i][0] = tc[i-1][0] + cost[i][0]

    # Initialize first row of tc array
    for j in range(1, n+1):
        tc[0][j] = tc[0][j-1] + cost[0][j]

    # Construct rest of the tc array
    for i in range(1, m+1):
        for j in range(1, n+1):
            tc[i][j] = min(tc[i-1][j-1], tc[i-1][j], tc[i][j-1]) + cost[i][j]

    return tc[m][n]

# Driver program to test above functions
cost = [[1, 2, 3],
        [4, 8, 2],
        [1, 5, 3]]
print(minCost(cost, 2, 2))

# This code is contributed by Bhavya Jain
```

Run on IDE

C#

```
// C# program for Dynamic Programming implementation
// of Min Cost Path problem
using System;

class GFG
{
    // A utility function that
    // returns minimum of 3 integers
    private static int min(int x, int y, int z)
    {
        if (x < y)
            return (x < z)? x : z;
        else
            return (y < z)? y : z;
    }

    private static int minCost(int [,]cost, int m, int n)
    {
        int i, j;
        int [,]tc=new int[m+1,n+1];

        tc[0,0] = cost[0,0];

        /* Initialize first column of total cost(tc) array */
        for (i = 1; i <= m; i++)
            tc[i, 0] = tc[i - 1, 0] + cost[i, 0];

        /* Initialize first row of tc array */
        for (j = 1; j <= n; j++)
            tc[0, j] = tc[0, j - 1] + cost[0, j];

        /* Construct rest of the tc array */
        for (i = 1; i <= m; i++)
            for (j = 1; j <= n; j++)
                tc[i, j] = min(tc[i - 1, j - 1],
                               tc[i - 1, j],
                               tc[i, j - 1]) + cost[i, j];

        return tc[m, n];
    }

    // Driver program
    public static void Main()
    {
        int [,]cost= {{1, 2, 3},
                      {4, 8, 2},
                      {1, 5, 3}};
        Console.Write(minCost(cost,2,2));
    }
}

// This code is contributed by Sam007.
```

Run on IDE

PHP

```
<?php
// DP implementation
// of MCP problem
$R = 3;
$C = 3;

function minCost($cost, $m, $n)
{
    global $R;
    global $C;
    // Instead of following line,
    // we can use int tc[m+1][n+1]
    // or dynamically allocate
    // memory to save space. The
    // following line is used to keep
    // the program simple and make
```

```
    // it working on all compilers.
    $tc;
    for ($i = 0; $i <= $R; $i++)
    for ($j = 0; $j <= $C; $j++)
    $tc[$i][$j] = 0;

    $tc[0][0] = $cost[0][0];

    /* Initialize first column of
       total cost(tc) array */
    for ($i = 1; $i <= $m; $i++)
        $tc[$i][0] = $tc[$i - 1][0] +
                     $cost[$i][0];

    /* Initialize first
       row of tc array */
    for ($j = 1; $j <= $n; $j++)
        $tc[0][$j] = $tc[0][$j - 1] +
                     $cost[0][$j];

    /* Construct rest of
       the tc array */
    for ($i = 1; $i <= $m; $i++)
        for ($j = 1; $j <= $n; $j++)

            // returns minimum of 3 integers
            $tc[$i][$j] = min($tc[$i - 1][$j - 1],
                              $tc[$i - 1][$j],
                              $tc[$i][$j - 1]) +
                              $cost[$i][$j];

    return $tc[$m][$n];
}


// Driver Code
$cost = array(array(1, 2, 3),
              array(4, 8, 2),
              array(1, 5, 3));
echo minCost($cost, 2, 2);

// This code is contributed by mits
?>
```

Run on IDE

Output:

```
8
```

Time Complexity of the DP implementation is O(mn) which is much better than Naive Recursive implementation.

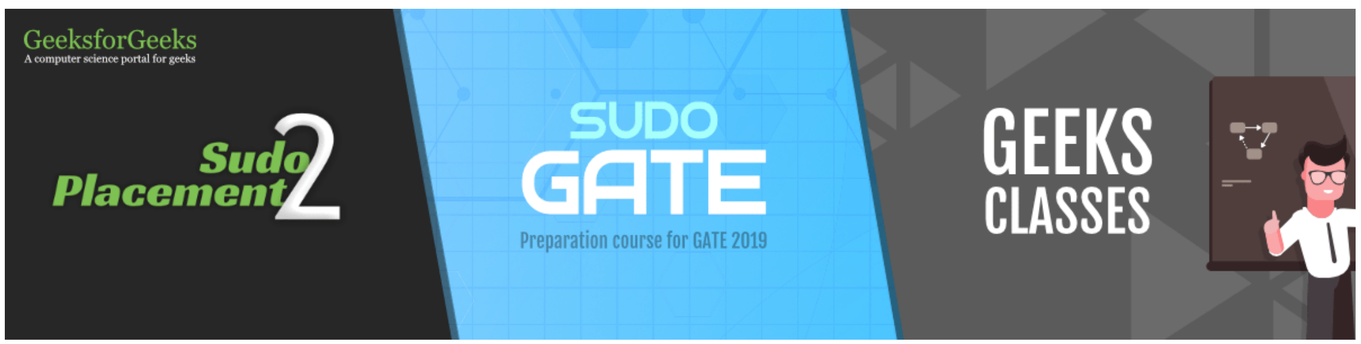
Asked in: Amazon

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Improved By : Sam007, Mithun Kumar, shiv_bhakt, maveriek

Practice Tags : Amazon, Dynamic Programming, Mathematical, Matrix

Article Tags : Dynamic Programming, Mathematical, Matrix, Amazon



Improve this Article

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

Recommended Posts:

Coin Change | DP-7
Edit Distance | DP-5
Length of the longest substring without repeating characters
Minimum Cost Path with Left, Right, Bottom and Up moves allowed
Matrix Chain Multiplication | DP-8
Paytm Interview experience for FTE (On-Campus)
Balanced expressions such that given positions have opening brackets | Set 2
Number of ways a convex polygon of n+2 sides can split into triangles by connecting vertices
Alternate Fibonacci Numbers
Find the largest area rectangular sub-matrix whose sum is equal to k

Logged in as **Dragan Nikolic**(Logout)

2.4 Average Difficulty : **2.4/5.0**
Based on **356** vote(s)

Basic Easy Medium Hard Expert

Add to TODO List
Mark as DONE

Give Feedback | Add your Notes | Improve Article

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments | Share this post!