Problem Statement :

- Given :
  - A cost matrix cost[][] (All costs are positive)
  - Position (m, n) in cost[][]

- Write a function that returns cost of minimum cost path to reach (m, n) from (0, 0).
  - Total cost of a path to reach (m, n) is sum of all the costs on that path (including both source and destination)

Allowed movements :

Can only traverse **up**, **right** and **diagonally upper** cells from a given cell.

- From a given cell (i, j)
  - (i+1 , j)
  - (i, j+1)
  - (i+1, j+1)


2
1
0
1
5
3
4
8
2
1
2
3
0
1
2

Minimum cost :

1 + 2 + 2 + 3 = 8

Optimal Substructure :

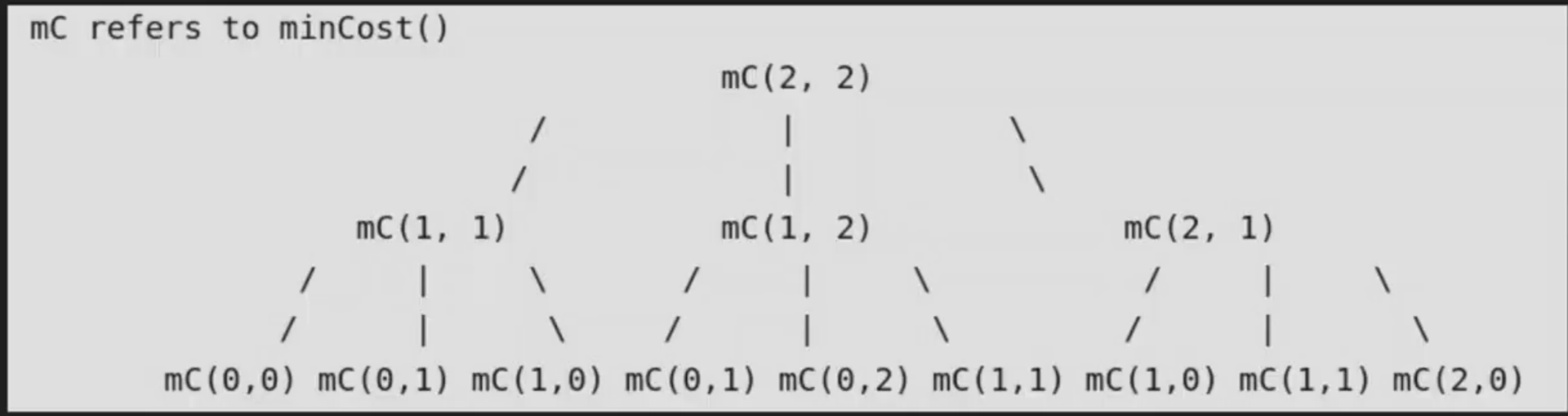
- The path to reach (m, n) :
  - Through one of the 3 cells: (m-1, n-1) or (m-1, n) or (m, n-1).

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

```
/* Returns cost of minimum cost path from (0,0) to (m, n
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) );
}
```

Time complexity of this naive recursive solution is exponential !

Overlapping Subproblems



So the problem has :

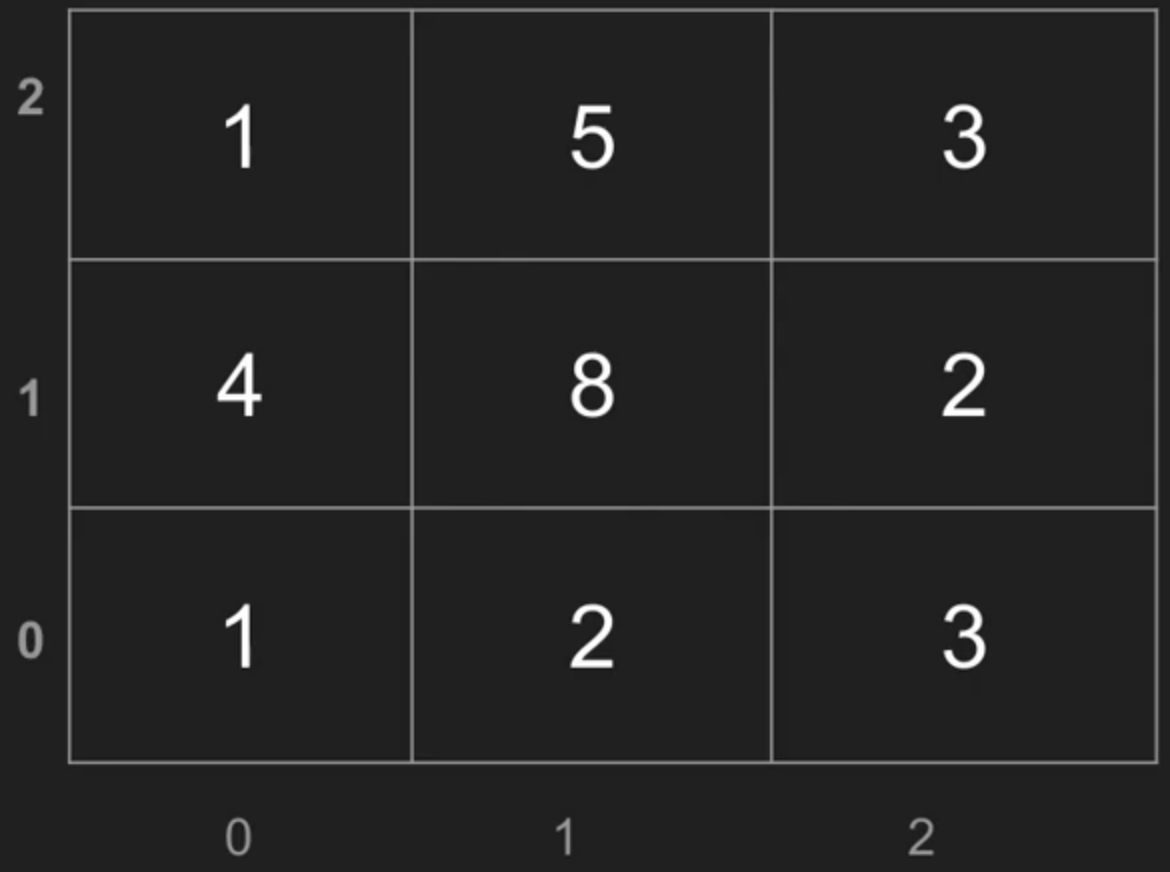
Optimal Substructure
and
Overlapping subproblems

Dynamic Programming

Constructing a temporary array tc[][] in bottom up manner.


2
1
0
1
5
3
4
8
2
1
2
3
0
1
2

Cost



tc

```
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 compile
    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];
}
```

Time
Complexity

O(m * n)

Thank You for watching !
Please leave us your likes and comments.