**Problem link**- [Minimum Path Sum](https://leetcode.com/problems/minimum-path-sum/)

\*\***1. recursive- solution: TC =** O(2^ (m\*n), for every index we have 2option(left/up)

**//SC** = O(m-1 + n-1) recursion stack space

**class Solution {**

**public:**

**int minPathSum(vector<vector<int>>& grid) {**

**int n = grid.size();**

**int m = grid[0].size();**

**return f(n-1, m-1, grid);**

**}**

**int f(int i, int j, vector<vector<int>>& grid){**

**if(i==0 && j==0) //return value if reached to goal**

**return grid[i][j];**

**if(i<0 || j<0) //don't consider this path so return a max value**

**return 1e6;**

**int up = grid[i][j] + f(i-1, j, grid);**

**int left = grid[i][j] + f(i, j-1, grid);**

**return min(up, left); //take min sum**

**}**

**};**

\*\***2. DP- memoization solution: //tc =** O(m\*n) for all new calls (0, 0 to m-1, n-1)

// **sc** = O(m-1 + n-1) for recursion stack space + O(m\*n) for dp[]

**class Solution {**

**public:**

**int minPathSum(vector<vector<int>>& grid) {**

**int n = grid.size();**

**int m = grid[0].size();**

**vector<vector<int>> dp(n, vector<int>(m, -1)); //step1**

**return f(n-1, m-1, grid, dp);**

**}**

**int f(int i, int j, vector<vector<int>>& grid, vector<vector<int>> &dp){**

**if(i==0 && j==0) //return value if reached to goal**

**return grid[i][j];**

**if(i<0 || j<0) //don't consider this path so return a max value**

**return 1e6;**

**if(dp[i][j] != -1) return dp[i][j]; //step2**

**int up = grid[i][j] + f(i-1, j, grid, dp);**

**int left = grid[i][j] + f(i, j-1, grid, dp);**

**return dp[i][j] = min(up, left); //take min sum(step3)**

**}**

**};**

\*\***3. DP- tabulation: //TC = O(m\*n), SC = O(m\*n)**

**//DP\_tabulation (bottom- up approach)**

**class Solution {**

**public:**

**int minPathSum(vector<vector<int>>& grid) {**

**int n = grid.size();**

**int m = grid[0].size();**

**vector<vector<int>> dp(n, vector<int>(m, -1)); //step1**

**//build dp from values calculated initially**

**for(int i=0; i<n; i++){**

**for(int j=0; j<m; j++){**

**if(i==0 && j==0) dp[i][j]= grid[0][0];**

**else{**

**int up = grid[i][j]; //go up, consider current value in sum**

**if(i>0)**

**up+= dp[i-1][j]; //base cases**

**else**

**up+= 1e6;**

**int left = grid[i][j]; //go left, consider current value in sum**

**if(j>0)**

**left += dp[i][j-1]; //base case**

**else**

**left += 1e6;**

**//cout<<min(up, left)<<endl;**

**dp[i][j] = min(up, left);**

**}**

**}**

**}**

**return dp[n-1][m-1];**

**}**

**};**

\***\*4. DP- optimized space:** //TC = O(m\*n), SC = O(2m)

**//use a prev[m] to keep track of [i-1][j] value and use curr[m] for [i][j-1] value**

**//DP\_tabulation (space- optimized)**

**class Solution {**

**public:**

**int minPathSum(vector<vector<int>>& grid) {**

**int n = grid.size();**

**int m = grid[0].size();**

**//store previous row to take care of dp[i-1][j] element**

**vector<int> prev(m, -1);**

**//build dp from values calculated initially**

**for(int i=0; i<n; i++){**

**vector<int> curr(m, -1); //stores current row**

**for(int j=0; j<m; j++){**

**if(i==0 && j==0) curr[j] = grid[i][j];**

**else{**

**int up = grid[i][j]; //go up, consider current value in sum**

**if(i>0)**

**up+= prev[j]; //add up previous row value**

**else**

**up+= 1e6;**

**int left = grid[i][j]; //go left, consider current value in sum**

**if(j>0)**

**left += curr[j-1]; //base case**

**else**

**left += 1e6;**

**// cout<<min(up, left)<<endl;**

**curr[j] = min(up, left);**

**}**

**}**

**prev = curr;**

**}**

**return prev[m-1];**

**}**

**};**