**Problem link**- [Maximum Path Sum](https://www.codingninjas.com/codestudio/problems/maximum-path-sum-in-the-matrix_797998?source=youtube&campaign=striver_dp_videos&utm_source=youtube&utm_medium=affiliate&utm_campaign=striver_dp_videos&leftPanelTab=0)

\*\***1. recursive- solution: TC =** O(3^n), for every element (in n length path) we have 3 options for every element

**//SC** = O(N) recursion stack space

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

**//out of boundary for left or right diagonal**

**if(j<0 || j>=matrix[0].size())**

**return -1e8;**

**if(i==0) //if reached to first row**

**return matrix[i][j];**

**int u = matrix[i][j] + f(i-1, j, matrix);**

**int ul = matrix[i][j] + f(i-1, j-1, matrix);**

**int ur = matrix[i][j] + f(i-1, j+1, matrix);**

**return max(u, max(ul, ur));**

**}**

**int getMaxPathSum(vector<vector<int>> &matrix)**

**{**

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

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

**int maxi = -1e8;**

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

**maxi = max(maxi, f(n-1, j, matrix)); //start from last row**

**}**

**return maxi;**

**}**

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

// **sc** = O(n\*m) + O(n) for recursion stack space

**//memoization solution**

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

**//out of boundary for left or right diagonal**

**if(j<0 || j>=matrix[0].size())**

**return -1e8;**

**if(i==0) //if reached to first row**

**return matrix[i][j];**

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

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

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

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

**return dp[i][j] = max(u, max(ul, ur));**

**}**

**int getMaxPathSum(vector<vector<int>> &matrix)**

**{**

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

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

**int maxi = -1e8;**

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

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

**maxi = max(maxi, f(n-1, j, matrix, dp)); //start from last row**

**}**

**return maxi;**

**}**

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

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

**//tabulation solution**

**int getMaxPathSum(vector<vector<int>> &matrix)**

**{**

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

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

**vector<vector<int>> dp (n, vector<int>(m, 0));**

**//build base(first row)**

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

**dp[0][j] = matrix[0][j];**

**//build complete dp**

**for(int i=1; i<n; i++){**

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

**int d = matrix[i][j] + dp[i-1][j];**

**int dl = matrix[i][j];**

**if(j-1>=0) dl += dp[i-1][j-1];**

**else dl += -1e8;**

**int dr = matrix[i][j];**

**if(j+1<m) dr += dp[i-1][j+1];**

**else dr += -1e8;**

**dp[i][j] = max(d, max(dl, dr));**

**}**

**}**

**//get max value from last row**

**int maxi = -1e8;**

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

**maxi = max(maxi, dp[n-1][j]);**

**return maxi;**

**}**

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

**//use a prev[m] to keep track of [i-1] values and use curr[m] for storing current row**

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

**int getMaxPathSum(vector<vector<int>> &matrix)**

**{**

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

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

**//prev row**

**vector<int> prev(m);**

**//build base(first row)**

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

**prev[j] = matrix[0][j];**

**//build complete dp**

**for(int i=1; i<n; i++){**

**vector<int> curr(m);**

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

**int d = matrix[i][j] + prev[j];**

**int dl = matrix[i][j];**

**if(j-1>=0) dl += prev[j-1];**

**else dl += -1e8;**

**int dr = matrix[i][j];**

**if(j+1<m) dr += prev[j+1];**

**else dr += -1e8;**

**curr[j] = max(d, max(dl, dr));**

**}**

**prev = curr;**

**}**

**//get max value from last row stored in prev**

**int maxi = -1e8;**

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

**maxi = max(maxi, prev[j]);**

**return maxi;**

**}**