Problem on 2d Dp

Count ALL possible ways type of question is type of recursion and dp

Q.1 <https://www.naukri.com/code360/problems/ninja-s-training_3621003?leftPanelTabValue=PROBLEM>

* so basically here we have to provide 3 argument
* input array and ind also the last=previous element index which we took at starting of call we can provide 3 because nothing has take so it will calculate with both
* do recursion and memorization which task for loop which will run till 3 because there are 3 task and tabulation has 3 for loop

|  |
| --- |
| int helper(int ind,vector<vector<int>> &points,int last,vector<vector<int>> &dp)  {      if(ind==0)      {          int maxi=0;          for(int i=0;i<3;i++)          {              if(i!=last)              {                  maxi=max(maxi,points[ind][i]);              }          }          return maxi;      }      if(dp[ind][last]!=-1) return dp[ind][last];      int maxi=0;          for(int i=0;i<3;i++)          {              int point=0;              if(i!=last)              {                  point=points[ind][i]+helper(ind-1,points, i,dp);              }              maxi=max(maxi,point);          }          dp[ind][last]=maxi;          return dp[ind][last];  }  int ninjaTraining(int n, vector<vector<int>> &points)  {      // Write your code here.      vector<vector<int>> dp(n,vector<int>(4,0));      // return helper(n-1,points,3,dp);      dp[0][0]=max(points[0][1],points[0][2]);      dp[0][1]=max(points[0][0],points[0][2]);      dp[0][2]=max(points[0][0],points[0][1]);      dp[0][3]=max(points[0][1],max(points[0][2],points[0][0]));      for(int day=1;day<n;day++)      {          for(int last=0;last<4;last++)          {              // int maxi=0;              dp[day][last]=0;              for(int task=0;task<3;task++)              {                  if(task!=last)                  {                      int point=points[day][task]+dp[day-1][task];                      dp[day][last]=max(dp[day][last],point);                  }                }            }      }      return dp[n-1][3];  } |

TC:O(N\*4)\*3 recursion

SC:O(N)

TC:O(N\*4)\*3 Memorization

SC:O(N)+O(N\*4)

TC:O(N\*4)\*3 Tabulation

SC:O(n\*4)

Q.2 <https://leetcode.com/problems/unique-paths/>

* here we just went from m-1,n-1 to 0 0 so right and down changed to up and left
* base condition is row==0 && col==0 return 1
* also if row<0 || col<0 return 0
* up=fun(row-1,col)
* left=fun(row,col-1)
* return up+left;

|  |
| --- |
| class Solution {  public:      int helper(int row, int col)      {          if(row==0 && col==0) return 1;          if(row<0 || col<0) return 0;          int up=helper(row-1,col);          int left=helper(row,col-1);          return left+up;      }      int memo(int row,int col,vector<vector<int>> &dp)      {          if(row==0 && col==0) return 1;          if(row<0 || col<0) return 0;            if(dp[row][col]!=-1) return dp[row][col];          int up=memo(row-1,col,dp);          int left=memo(row,col-1,dp);          dp[row][col]=up+left;          return dp[row][col];      }      int uniquePaths(int m, int n) {          vector<vector<int>> dp(m,vector<int>(n,-1));          // return memo(m-1,n-1,dp);          dp[0][0]=1;          for(int row=0;row<m;row++)          {              for(int col=0;col<n;col++)              {                  int up=0,left=0;                  if(row==0 && col==0) continue;                  else                  {                        if(row>0)                      {                          up=dp[row-1][col];                      }                      if(col >0) left=dp[row][col-1];                  }                  dp[row][col]=up+left;              }          }          return dp[m-1][n-1];      }  }; |

Recursion TC:O(2^m\*n) SC:O(Len Path)

Memo TC:O(m\*n) SC:O(n-1+m-1) path length +O(N\*M) DP)

Tab TC:O(M\*N) SC:O(N\*M)

Q.3 <https://leetcode.com/problems/unique-paths-ii/description/>

* here we just went from m-1,n-1 to 0 0 so right and down changed to up and left
* base condition is row==0 && col==0 && input[row][col]!=1 return 1
* also if row<0 || col<0 return 0 also if input[row][col]==1 return 0;
* up=fun(row-1,col)
* left=fun(row,col-1)
* return left+up
* use same logic to covert into memo and tab

|  |
| --- |
| class Solution {  public:      int helper(int row,int col,vector<vector<int>> &grid)      {            if(row==0 && col==0 && grid[row][col]!=1) return 1;          if(row<0 || col<0) return 0;          if(grid[row][col]==1) return 0;            int up=helper(row-1,col,grid);          int down=helper(row,col-1,grid);          return up+down;      }       int memo(int row,int col,vector<vector<int>> &grid,vector<vector<int>> & dp)      {            if(row==0 && col==0 && grid[row][col]!=1) return 1;          if(row<0 || col<0) return 0;          if(grid[row][col]==1) return 0;            if(dp[row][col]!=-1) return dp[row][col];          int up=memo(row-1,col,grid,dp);          int down=memo(row,col-1,grid,dp);          dp[row][col]=up+down;          return dp[row][col];      }      int uniquePathsWithObstacles(vector<vector<int>>& obstacleGrid) {          int n=obstacleGrid.size();          int m=obstacleGrid[0].size();          vector<vector<int>> dp(n,vector<int>(m,0));          // return memo(n-1,m-1,obstacleGrid,dp);          if(obstacleGrid[0][0]==1 || obstacleGrid[n-1][m-1]==1) return 0;          dp[0][0]=1;          for(int row=0;row<n;row++)          {              for(int col=0;col<m;col++)              {                  int up=0,left=0;                  if(row==0 && col==0 || obstacleGrid[row][col]==1) continue;                  else                  {                      if(row>0 && obstacleGrid[row-1][col]!=1) up=dp[row-1][col];                      if(col>0 && obstacleGrid[row][col]!=1) left=dp[row][col-1];                  }                  dp[row][col]=up+left;              }          }          return dp[n-1][m-1];      }  }; |

Recursion TC:O(2^m\*n) SC:O(Len Path)

Memo TC:O(m\*n) SC:O(n-1+m-1) path length +O(N\*M) DP)

Tab TC:O(M\*N) SC:O(N\*M)

Q.4 <https://leetcode.com/problems/minimum-path-sum/>

🡪 logic is same as above question we have find all path but have to take minimum path

* base case i==0 && j==0 return grid[i][j] because here we have to return sum of path
* second base is if i<0 || j<0 return 1e9 maximum value so this path can’t be considered
* up=grid[i][j]+fun(i-1,j,grid)
* left=grid[i][j]+fun(i,j-1,grid)
* return min(up,left)
* covert this into memo and tab

|  |
| --- |
| class Solution {  public:      int helper(int row,int col,vector<vector<int>>& grid)      {              if(row==0 && col==0) return grid[row][col];              if(row<0 || col<0) return 1e9;              int up=grid[row][col]+helper(row-1,col,grid);              int left=grid[row][col]+helper(row,col-1,grid);              return min(up,left);      }      int Memo(int row,int col,vector<vector<int>>& grid,vector<vector<int>> &dp)      {              if(row==0 && col==0) return grid[row][col];              if(row<0 || col<0) return 1e9;              if(dp[row][col]!=-1) return dp[row][col];              int up=grid[row][col]+Memo(row-1,col,grid,dp);              int left=grid[row][col]+Memo(row,col-1,grid,dp);              dp[row][col]=min(up,left);              return dp[row][col];      }      int minPathSum(vector<vector<int>>& grid) {          int n=grid.size();          int m=grid[0].size();          vector<vector<int>> dp(n,vector<int>(m,-1));          // return Memo(n-1,m-1,grid,dp);          // return helper(n-1,m-1,grid);          dp[0][0]=grid[0][0];          for(int i=0;i<n;i++)          {              for(int j=0;j<m;j++)              {                  if(i==0 && j==0) continue;                  else                  {                      int up=grid[i][j];                      int left=grid[i][j];                      if(i>0)                      {                          up+=dp[i-1][j];                      }                      else                      {                          up+=1e9;                      }                      if(j>0) left+=dp[i][j-1];                      else left+=1e9;                      dp[i][j]=min(up,left);                  }              }          }          return dp[n-1][m-1];      }  }; |

Recursion TC:O(2^m\*n) SC:O(Len Path)

Memo TC:O(m\*n) SC:O(n-1+m-1) path length +O(N\*M) DP)

Tab TC:O(M\*N) SC:O(N\*M)

Q.5 <https://leetcode.com/problems/triangle/description/>

* in this question you have given fixed starting point and and ending point we don’t know
* so whenever we have given fixed starting point we’ll start recursion from their
* base condition row==n-1
* you have to do in down and diagonal right direction take out minimum and return

|  |
| --- |
| class Solution {  public:      int helper(int row,int col,vector<vector<int>> &traingle,int n,vector<vector<int>> &dp)      {         if(row==n-1) return traingle[row][col];         if(dp[row][col]!=-1) return dp[row][col];         int down=traingle[row][col]+helper(row+1,col,traingle,n,dp);         int digonal=traingle[row][col]+helper(row+1,col+1,traingle,n,dp);         return dp[row][col]=min(down,digonal);      }      int minimumTotal(vector<vector<int>>& triangle) {            int n=triangle.size();          vector<vector<int>> dp(n,vector<int>(n,-1));          // return helper(0,0,triangle,n,dp);          for(int j=0;j<n;j++)          {              dp[n-1][j]=triangle[n-1][j];          }          for(int row=n-2;row>=0;row--)          {              for(int col=row;col>=0;col--)              {                  int down=triangle[row][col]+dp[row+1][col];                  int digonal=triangle[row][col]+dp[row+1][col+1];                  dp[row][col]=min(down,digonal);              }          }          return dp[0][0];      }  }; |

Recursion : TC:- 1,2,3,4 …n row have m colums and for every row col there are 2 choices go down or diagonal so TC is 2^ that above equation that is exponential in nature

SC: O(N) recursive stack space

Memo: TC:-O(N\*N) traversing SC:-O(N\*N) dp+ O(N)) recursion

Tab : TC:O(N\*N) SC:O(N\*N)

Q.6 <https://leetcode.com/problems/minimum-falling-path-sum/description/>

* so we don’t know either starting and ending points so here we’ll start from last row and for each column we call recursion and take minimum out of it
* base condition first out of bound condition if column go < 0 or >m
* second base condition if it row==0
* call on 3 direction
* take out minimum and return

|  |
| --- |
| class Solution {  public:      int helper(int row,int col,vector<vector<int>> &matrix,int m,vector<vector<int>>&dp)      {          if(col<0 || col>=m) return 1e9;          if(row==0 ) return matrix[row][col];          if(dp[row][col]!=1e9) return dp[row][col];          int up=matrix[row][col]+helper(row-1,col,matrix,m,dp);          int digo\_left=matrix[row][col]+helper(row-1,col-1,matrix,m,dp);          int digo\_right=matrix[row][col]+helper(row-1,col+1,matrix,m,dp);          return dp[row][col]=min(up,min(digo\_left,digo\_right));      }      int minFallingPathSum(vector<vector<int>>& matrix) {          int n=matrix.size();          int m=matrix[0].size();          vector<vector<int>> dp(n,vector<int>(m,1e9));          int mini=INT\_MAX;          // for(int j=0;j<m;j++)          // {          //     int sum=helper(n-1,j,matrix, m,dp);          //     mini=min(sum,mini);          // }          for(int j=0;j<m;j++)          {              dp[0][j]=matrix[0][j];          }          for(int row=1;row<n;row++)          {              for(int col=0;col<m;col++)              {                  int digoL=1e9,digoR=1e9;                  int up=matrix[row][col]+dp[row-1][col];                  if(col-1>=0)  digoL=matrix[row][col]+dp[row-1][col-1];                  if(col+1 < m)  digoR=matrix[row][col]+dp[row-1][col+1];                  dp[row][col]=min(up,min(digoL,digoR));              }          }           for(int j=0;j<m;j++)          {              mini=min(dp[n-1][j],mini);          }          return mini;      }  }; |

Recursion : TC:- O(M\*3^ no of element in grid ) SC: O(N) at max we can traval N rows

Memo : TC:-O(N\*M) SC:O(N\*M) Dp+O(N) recursion

Tab : TC:-O(N\*M) SC:O(N\*M) Dp

Q.7 <https://naukri.com/code360/problems/ninja-and-his-friends_3125885?leftPanelTabValue=PROBLEM>

🡪so here we have given starting point and variable ending point

* so here we have to move alice and bob together
* so their will be common i and j1 ,j2 is different because I is going to same for each for both alice and bob
* base condition j1<0 || j1>=m || j2<0 || j2>=m return -1e8
* also if i==n-1 then check if both point to same cell that is j1==j2 return value only one else return matrix[i][j1]+matrix[i][j2]
* for one move alice there can be 3 moves of bob
* so for that dj1 for loop -1 till 1
* inside it dj2 for loop -1 till 1
* take value ==0
* check j1==j2 then value=matrix[i][j1]
* else value =matrix[i][j1]+[i][j2];
* do recursive call value+= helper(i+1,j1+dj1,j2+dj2,matrix)
* store maximum at last return maximum

|  |
| --- |
| #include <bits/stdc++.h>  int helper(int i,int j1,int j2,int r,int c,vector<vector<int>> &grid,vector<vector<vector<int>>> &dp)  {      if(j1<0 || j1>=c || j2<0 || j2>=c) return -1e8;      if (i == r - 1) {        if (j1 == j2)          return grid[i][j1];        else          return grid[i][j1] + grid[i][j2];      }      if(dp[i][j1][j2]!=-1) return dp[i][j1][j2];          int maxi=-1e8;          for(int dj1=-1;dj1<=1;dj1++)          {              for(int dj2=-1;dj2<=1;dj2++)              {                  int value=0;                  if(j1==j2) value=grid[i][j1];                  else value=grid[i][j1]+grid[i][j2];                  value+=helper(i+1,j1+dj1,j2+dj2,r,c,grid,dp);                  maxi=max(maxi,value);              }          }      return dp[i][j1][j2]=maxi;  }  int maximumChocolates(int r, int c, vector<vector<int>> &grid) {      // Write your code here.      vector<vector<vector<int>>> dp(r,vector<vector<int>>(c,vector<int>(c,-1)));      return helper(0,0,c-1,r,c,grid,dp);  } |

Recursion : TC:-O(3^n\*3^n) for every step alice and bob have 3 option and their N step SC: O(N) recursive call

Memo: TC:O(N\*M\*M) SC:O(N\*M\*M) dp+O(N) recursion