7) Geeks Training

04 December 2024 19:25

Geek is going for a training program. He can perform any of these activities: Running, Fighting, and Learning Practice. Each activity has some point on each day. As Geek wants to improve all his skills, he can't do the same activity on two consecutive days. Help Geek to maximize his merit points as you are given a 2D array of points **arr,** corresponding to each day and activity.

From <https://www.geeksforgeeks.org/problems/geeks-training/1?utm_source=youtube&utm_medium=collab_striver_ytdescription&utm_campaign=geeks-training>

//recursive approach

```
class Solution {
  public:
    int helper(vector<vector<int>>& arr, int day, int last){
        if(day == 0){
            int maxi = 0;
            for(int task = 0; task < 3; task++){
                if(task != last){
                    maxi = max(maxi, arr[0][task]);
                }
            }
            return maxi;
        }

        int maxi = 0;
        for(int task = 0; task < 3; task++){
            if(task != last){
                int points = arr[day][task] + helper(arr, day-1, task);
                maxi = max(maxi, points);
            }
        }
        return maxi;
    }
    int maximumPoints(vector<vector<int>>& arr, int n) {
        return helper(arr, n-1, 3);
    }
};
```

//memoization

```
    int helper(vector<vector<int>>& arr, int day, int last, vector<vector<int>> &dp){
        if(day == 0){
            int maxi = 0;
            for(int task = 0; task < 3; task++){
                if(task != last){
                    maxi = max(maxi, arr[0][task]);
                }
            }
```

```
        return maxi;
    }

    if(dp[day][last] != -1) return dp[day][last];

    int maxi = 0;
    for(int task = 0; task < 3; task++){
        if(task != last){
            int points = arr[day][task] + helper(arr, day-1, task, dp);
            maxi = max(maxi, points);
        }
    }
    return dp[day][last] = maxi;
}
int maximumPoints(vector<vector<int>>& arr) {
    int n = arr.size();
    vector<vector<int>>dp(n, vector<int>(4, -1));
    return helper(arr, n-1, 3, dp);
}
```

8) Unique Paths

06 December 2024 00:38

There is a robot on an m x n grid. The robot is initially located at the **top-left corner** (i.e., grid[0][0]). The robot tries to move to the **bottom-right corner** (i.e., grid[m - 1][n - 1]). The robot can only move either down or right at any point in time.
Given the two integers m and n, return *the number of possible unique paths that the robot can take to reach the bottom-right corner*.
The test cases are generated so that the answer will be less than or equal to $2 * 10^9$.

From <https://leetcode.com/problems/unique-paths/description/>

```
int helper(int i, int j, vector<vector<int>> &dp){
    if(i == 0 && j == 0) return 1;
    if(i < 0 || j < 0) return 0;
    if(dp[i][j] != -1) return dp[i][j];
    int up = helper(i-1, j, dp);
    int left = helper(i, j-1, dp);
    return dp[i][j] = left + up;
}
int uniquePaths(int m, int n) {
    vector<vector<int>> dp(m, vector<int>(n, -1));
    return helper(m-1, n-1, dp);
}
```

9) Unique Paths II (almost same as unique path)

06 December 2024 00:52

You are given an m x n integer array grid. There is a robot initially located at the **top-left corner** (i.e., grid[0][0]). The robot tries to move to the **bottom-right corner** (i.e., grid[m - 1][n - 1]). The robot can only move either down or right at any point in time.
An obstacle and space are marked as 1 or 0 respectively in grid. A path that the robot takes cannot include **any** square that is an obstacle.
Return *the number of possible unique paths that the robot can take to reach the bottom-right corner*.
The testcases are generated so that the answer will be less than or equal to $2 * 10^9$.

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

```
    int helper(int i, int j, vector<vector<int>> &dp, vector<vector<int>>&
obstacleGrid){
        if(i == 0 && j == 0) return 1;
        if(i < 0 || j < 0) return 0;
        if(obstacleGrid[i][j] == 1) return 0;
        if(dp[i][j] != -1) return dp[i][j];
        int up = helper(i-1, j, dp, obstacleGrid);
        int left = helper(i, j-1, dp, obstacleGrid);
        return dp[i][j] = left + up;
    }
    int uniquePathsWithObstacles(vector<vector<int>>& obstacleGrid) {
        int m = obstacleGrid.size();
        int n = obstacleGrid[0].size();
        vector<vector<int>> dp(m, vector<int>(n, -1));
        if(obstacleGrid[0][0] == 1 || obstacleGrid[m-1][n-1] == 1) return 0;
        return helper(m-1, n-1, dp, obstacleGrid);
    }
```

10) Minimum path sum (based on q8)

06 December 2024 01:29

Given a m x n grid filled with non-negative numbers, find a path from top left to bottom right, which minimizes the sum of all numbers along its path.
Note: You can only move either down or right at any point in time.

From <https://leetcode.com/problems/minimum-path-sum/description/>

```
int helper(int i, int j, vector<vector<int>> &dp, vector<vector<int>>& grid){
    if(i == 0 && j == 0) return grid[0][0];
    if(i < 0 || j < 0) return 1e9;
    if(dp[i][j] != -1) return dp[i][j];

    int up = grid[i][j] + helper(i-1, j, dp, grid);
    int left = grid[i][j] + helper(i, j-1, dp, grid);
    return dp[i][j] = min(left, up);
}
int minPathSum(vector<vector<int>>& grid) {
    int m = grid.size();
    int n = grid[0].size();
    vector<vector<int>> dp(m, vector<int>(n, -1));
    return helper(m-1, n-1, dp, grid);
}
```

11) Triangle (based on q8)

06 December 2024 01:55

Given a triangle array, return *the minimum path sum from top to bottom*.
For each step, you may move to an adjacent number of the row below. More formally, if you are on index i on the current row, you may move to either index i or index i + 1 on the next row.

From <https://leetcode.com/problems/triangle/description/>

```
    int helper(int i, int j, int &m, vector<vector<int>>& triangle,
vector<vector<int>> &dp){
        if(i == m-1) return triangle[m-1][j];
        if(dp[i][j] != -1) return dp[i][j];
        int down = triangle[i][j] + helper(i+1, j, m, triangle, dp);
        int diagonal = triangle[i][j] + helper(i+1, j+1, m, triangle, dp);
        return dp[i][j] = min(down, diagonal);
    }
    int minimumTotal(vector<vector<int>>& triangle) {
        int m = triangle.size();
        vector<vector<int>>dp(m, vector<int>(m, -1));
        return helper(0, 0, m, triangle, dp);
    }
```

12) Maximum Path Sum in matrix

06 December 2024 02:54

Given a **n** x **n** matrix of positive integers. There are only three possible moves from a cell **mat[r][c]**.

1. mat[r+1] [c]
2. mat[r+1] [c-1]
3. mat [r+1] [c+1]

Starting from any column in row 0 return the largest sum of any of the paths up to row n -1. Return the highest maximum path sum.

From <https://www.geeksforgeeks.org/problems/path-in-matrix3805/1>

```
int helper(int i, int j, int m, int n, vector<vector<int>>& matrix, vector<vector<int>>& dp) {
    if (j < 0 || j >= n) return -1e9;
    if (i == m - 1) return matrix[m - 1][j];
    if (dp[i][j] != -1) return dp[i][j];

    int down = matrix[i][j] + helper(i + 1, j, m, n, matrix, dp);
    int leftDiag = matrix[i][j] + helper(i + 1, j - 1, m, n, matrix, dp);
    int rightDiag = matrix[i][j] + helper(i + 1, j + 1, m, n, matrix, dp);

    dp[i][j] = max(down, max(leftDiag, rightDiag));
    return dp[i][j];
}

int maximumPath(int N, vector<vector<int>> matrix)
{
    int m = matrix.size();
    int n = matrix[0].size();
    vector<vector<int>> dp(m, vector<int>(n, -1));
    int maxi = INT_MIN;

    for (int j = 0; j < n; j++) {
        maxi = max(maxi, helper(0, j, m, n, matrix, dp));
    }
    return maxi;
}
```

13) Minimum falling path sum

06 December 2024 02:55

Given an n x n array of integers matrix, return *the* ***minimum sum*** *of any* ***falling path*** *through* matrix.
A **falling path** starts at any element in the first row and chooses the element in the next row that is either directly below or diagonally left/right. Specifically, the next element from position (row, col) will be (row + 1, col - 1), (row + 1, col), or (row + 1, col + 1).

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

```
int helper(int i, int j, int m, int n, vector<vector<int>>& matrix, vector<vector<int>>& dp) {
    if (j < 0 || j >= n) return 1e9;
    if (i == m - 1) return matrix[m - 1][j];
    if (dp[i][j] != -1) return dp[i][j];
    int down = matrix[i][j] + helper(i + 1, j, m, n, matrix, dp);
    int leftDiag = matrix[i][j] + helper(i + 1, j - 1, m, n, matrix, dp);
    int rightDiag = matrix[i][j] + helper(i + 1, j + 1, m, n, matrix, dp);
    dp[i][j] = min(down, min(leftDiag, rightDiag));
    return dp[i][j];
}
int minFallingPathSum(vector<vector<int>>& matrix) {
    int m = matrix.size();
    int n = matrix[0].size();
    vector<vector<int>> dp(m, vector<int>(n, -1));
    int mini = INT_MAX;
    for (int j = 0; j < n; j++) {
        mini = min(mini, helper(0, j, m, n, matrix, dp));
    }
    return mini;
}
```

From <https://leetcode.com/problems/minimum-falling-path-sum/submissions/1471349005/>

//but the above gives TLE, so we do tabulation

```
int minFallingPathSum(vector<vector<int>>& matrix) {
    int m = matrix.size();
    int n = matrix[0].size();

    vector<int> dp = matrix[m-1];
    for (int i = m - 2; i >= 0; --i) {
        vector<int> temp(n);
        for (int j = 0; j < n; ++j) {
            int down = dp[j];
            int leftDiag = (j > 0) ? dp[j-1] : INT_MAX;
            int rightDiag = (j < n - 1) ? dp[j+1] : INT_MAX;

            temp[j] = matrix[i][j] + min(down, min(leftDiag, rightDiag));
        }
        dp = temp;
    }
    return *min_element(dp.begin(), dp.end());
}
```

From <https://leetcode.com/problems/minimum-falling-path-sum/submissions/1471350638/>

14) Minimum falling path sum II

06 December 2024 03:30

Given an n x n integer matrix grid, return *the minimum sum of a* ***falling path with non-zero shifts***.
A **falling path with non-zero shifts** is a choice of exactly one element from each row of grid such that no two elements chosen in adjacent rows are in the same column.

From <https://leetcode.com/problems/minimum-falling-path-sum-ii/description/>

```
    int solve(int &n, int &m, int row, int col, vector<vector<int>>& grid,
vector<vector<int>> &dp) {
        if (row == n - 1) {
            return grid[row][col];
        }
        if(dp[row][col] != -1) {
            return dp[row][col];
        }

        int ans = INT_MAX;
        for (int nextCol = 0; nextCol < n; nextCol++) {
            if (nextCol != col) {
                ans = min(ans, solve(n, m, row + 1, nextCol, grid, dp));
            }
        }
        return dp[row][col] = grid[row][col] + ans;
    }

    int minFallingPathSum(vector<vector<int>>& grid) {
        int n = grid.size();
        int m = grid[0].size();
        vector<vector<int>>dp(n, vector<int>(m, -1));
        int result = INT_MAX;
        for (int col = 0; col < n; col++) {
            result = min(result, solve(n, m, 0, col, grid, dp));
        }
        return result;
    }
```

//same typa soln, just tle on 1 test case

```
    int helper(int &m, int &n, int i, int j, vector<vector<int>>& grid,
vector<vector<int>>& dp){
        if(i == m-1) return grid[i][j];
        if(dp[i][j] != -1) return dp[i][j];
        int mini = INT_MAX;
        for(int col = 0; col < m; col++){
            if(col != j){
                int points = grid[i][j] + helper(m, n, i+1, col, grid, dp);
                mini = min(mini, points);
            }
        }
        return dp[i][j] = mini;
    }
    int minFallingPathSum(vector<vector<int>>& grid) {
```

```
    int m = grid.size();
    int n = grid[0].size();
    vector<vector<int>>dp(m, vector<int>(n, -1));
    int mini = INT_MAX;
    for(int j = 0; j < n; j++){
        mini = min(mini, helper(m, n, 0, j, grid, dp));
    }
    return mini;
}
```

15) Chocolate pickup

06 December 2024 04:05

You are given an **n** rows and **m** cols matrix **grid** representing a field of chocolates where grid[i][j] represents the number of chocolates that you can collect from the (i, j) cell. You have two robots that can collect chocolates for you:

- **Robot #1** is located at the **top-left corner** (0, 0), and
- **Robot #2** is located at the **top-right corner** (0, cols - 1).

Return the maximum number of chocolates collection using both robots by following the rules below:

- From a cell (i, j), robots can move to cell (i + 1, j - 1), (i + 1, j), or (i + 1, j + 1).
- When any robot passes through a cell, It picks up all chocolates, and the cell becomes an empty cell.
- When both robots stay in the same cell, only one takes the chocolates.
- Both robots cannot move outside of the grid at any moment.
- Both robots should reach the bottom row in grid.

From <https://www.geeksforgeeks.org/problems/chocolates-pickup/1?utm_source=youtube&utm_medium=collab_striver_ytdescription&utm_campaign=chocolates-pickup>

```
int helper(int i, int j1, int j2, int n, int m, vector<vector<int>>& grid, vector<vector<vector<int>>> &dp){
    if (j1 < 0 || j2 < 0 || j1 >= m || j2 >= m) return -1e9;

    if(i == n-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 = -1e9;
    for(int k = -1; k <= 1; k++){
       for(int l = -1; l <= 1; l++){
          int value = 0;
          if(j1 == j2) value = grid[i][j1];
          else value = grid[i][j1] + grid[i][j2];
          value += helper(i+1, j1+k, j2+l, n, m, grid, dp);
          maxi = max(maxi ,value);
       }
    }
    return dp[i][j1][j2] = maxi;
}

int solve(int n, int m, vector<vector<int>>& grid) {
    vector<vector<vector<int>>> dp(n, vector<vector<int>>(m, vector<int>(m, -1)));
    return helper(0, 0, m-1, n, m, grid, dp);
}
```