1) fibonacci number

02 December 2024 19:32

The **Fibonacci numbers**, commonly denoted F(n) form a sequence, called the **Fibonacci sequence**, such that each number is the sum of the two preceding ones, starting from 0 and 1. That is,
F(0) = 0, F(1) = 1
F(n) = F(n - 1) + F(n - 2), for n > 1.
Given n, calculate F(n).

From <https://leetcode.com/problems/fibonacci-number/description/>

//memoiation:

```
    int helper(int n, vector<int>&dp){
        if(n == 0 || n==1) return n;
        if(dp[n] != -1) return dp[n];
        return dp[n] = helper(n-1, dp) + helper(n-2, dp);
    }
    int fib(int n) {
        vector<int> dp(n+1, -1);
        return helper(n, dp);
    }
```

//tabulation

```
    int fib(int n) {
        if(n <= 1) return n;
        vector<int> dp(n+1, -1);
        dp[0] = 0;
        dp[1] = 1;
        for(int i = 2; i <= n; i++){
            dp[i] = dp[i-1] + dp[i-2];
        }
        return dp[n];
    }
```

//space optimization

```
    int fib(int n) {
        if(n <= 1) return n;
        int prev2 = 0;
        int prev = 1;
        for(int i = 2; i <= n; i++){
            int curr = prev + prev2;
            prev2 = prev;
            prev = curr;
        }
        return prev;
    }
```

2) Climbing stairs

02 December 2024 19:54

You are climbing a staircase. It takes n steps to reach the top.
Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

From <https://leetcode.com/problems/climbing-stairs/description/>

```
int climbStairs(int n) {
    if(n <= 1) return n;
    int prev2 = 1;
    int prev = 1;
    for(int i = 2; i <= n; i++){
        int curr = prev + prev2;
        prev2 = prev;
        prev = curr;
    }
    return prev;
}
```

From <https://leetcode.com/problems/climbing-stairs/submissions/1468383397/>

3) Geek Jump

03 December 2024 01:54

Geek wants to climb from the 0th stair to the (n-1)th stair. At a time the Geek can climb either one or two steps. A height[N] array is also given. Whenever the geek jumps from stair i to stair j, the energy consumed in the jump is abs(height[i]- height[j]), where abs() means the absolute difference. return the minimum energy that can be used by the Geek to jump from stair 0 to stair N-1.

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

//memoization

```
  int helper(vector<int>& height, vector<int> &dp, int n){
      if(n == 0)return 0;
      if(dp[n] != -1) return dp[n];
      int jump2 = INT_MAX;
      int jump1 = helper(height, dp, n-1) + abs(height[n] - height[n-1]);
      if(n > 1) jump2 = helper(height, dp, n-2) + abs(height[n] - height[n-2]);
      return dp[n] = min(jump1, jump2);
  }
  int minCost(vector<int>& height) {
      int n = height.size();
      vector<int> dp(n, -1);
      return helper(height, dp, n-1);
  }
```

//tabulation

```
  int minimumEnergy(vector<int>& height, int n) {
      vector<int>dp(n, -1);
      dp[0] = 0;

      for(int i = 1; i < n; i++){
          int jump1 = dp[i-1] + abs(height[i] - height[i-1]);
          int jump2 = INT_MAX;
          if(i > 1) jump2 = dp[i-2] + abs(height[i] - height[i-2]);

          dp[i] = min(jump1, jump2);
      }
      return dp[n-1];
  }
```

//space

```
  int minimumEnergy(vector<int>& height, int n) {
      int prev = 0;
      int prev2 = 0;

      for(int i = 1; i < n; i++){
          int jump1 = prev + abs(height[i] - height[i-1]);
          int jump2 = INT_MAX;
          if(i > 1) jump2 = prev2 + abs(height[i] - height[i-2]);

          int curr = min(jump1, jump2);
          prev2 = prev;
          prev = curr;
      }
      return prev;
  }
```

4) Geek Jump 2/ Minimal Cost

03 December 2024 02:52

There is an array **arr** of heights of stone and Geek is standing at the first stone and can jump to one of the following: Stone i+1, i+2, ... i+k stone, where k is the maximum number of steps that can be jumped and cost will be $|h_i - h_j|$ is incurred, where j is the stone to land on. Find the minimum possible total cost incurred before the Geek reaches the last stone.

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

```
//recursive solution

  int helper(int k, vector<int>&arr, int n){
    if(n == 0) return 0;
    int mini = INT_MAX;

    for(int i = 1; i <= k; i++){
      int jump = INT_MAX;
      if(n-i >= 0) jump = helper(k, arr, n-i) + abs(arr[n] - arr[n-i]);
      mini = min(mini, jump);
    }
    return mini;
  }
  int minimizeCost(int k, vector<int>& arr) {
    int n = arr.size();
    return helper(k, arr, n-1);
  }

//memoization

  int helper(int k, vector<int>&arr, int n ,vector<int> &dp){
    if(n == 0) return 0;
    if(dp[n] != -1) return dp[n];
    int mini = INT_MAX;

    for(int i = 1; i <= k; i++){
      int jump = INT_MAX;
      if(n-i >= 0) jump = helper(k, arr, n-i, dp) + abs(arr[n] - arr[n-i]);
      mini = min(mini, jump);
    }
    return dp[n] = mini;
  }
  int minimizeCost(int k, vector<int>& arr) {
    int n = arr.size();
    vector<int> dp(n, -1);
    return helper(k, arr, n-1, dp);
  }

//tabulation

  int helper(int k, vector<int>&arr, int n ,vector<int> &dp){
    dp[0] = 0;

    for(int i = 1; i < n; i++){
        int mini = INT_MAX;
        for(int j = 1; j <= k; j++){
            int jump = INT_MAX;
            if(i-j >= 0) jump = dp[i-j] + abs(arr[i] - arr[i-j]);
            mini = min(mini, jump);
        }
        dp[i] = mini;
    }
    return dp[n-1];
}
int minimizeCost(int k, vector<int>& arr) {
    int n = arr.size();
    vector<int> dp(n, -1);
    return helper(k, arr, n, dp);
}
```

5) House Robber

04 December 2024 16:30

You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed, the only constraint stopping you from robbing each of them is that adjacent houses have security systems connected and **it will automatically contact the police if two adjacent houses were broken into on the same night.**
Given an integer array nums representing the amount of money of each house, return *the maximum amount of money you can rob tonight* ***without alerting the police***.

From <https://leetcode.com/problems/house-robber/description/>

//memoization

```
    int helper(vector<int>& nums, int n, vector<int>& dp){
        if(n < 0) return 0;
        if(dp[n] != -1)return dp[n];
        int option1 = nums[n] + helper(nums, n-2, dp);
        int option2 = 0 + helper(nums, n-1, dp);
        return dp[n] = max(option1, option2);
    }
    int rob(vector<int>& nums) {
        int n = nums.size();
        vector<int>dp(n, -1);
        return helper(nums, n-1, dp);
    }
```

From <https://leetcode.com/problems/house-robber/submissions/1470053083/>

//tabulation

```
int solve(vector<int>& arr, int n, vector<int>& dp) {
    dp[0] = arr[0];

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

        int pick = arr[i];
        if (i > 1)
            pick += dp[i - 2];
        int nonPick = dp[i - 1];
        dp[i] = max(pick, nonPick);
    }
    return dp[n - 1];
}
    int rob(vector<int>& nums){
       int n = nums.size();
       int index = 0;
       vector<int> dp(n, -1);
       return solve(nums, n, dp);
    }
```

From <https://leetcode.com/problems/house-robber/submissions/1470075130/>

//space optimization

```
int solve(vector<int>& arr, int n) {
        int prev = arr[0];
        int prev2 = 0;

        for (int i = 1; i < n; i++) {
            int pick = arr[i] + prev2;
            int nonPick = prev;
            int curr = max(pick, nonPick);
            prev2 = prev;
            prev = curr;
        }
        return prev;
    }
    int rob(vector<int>& nums){
       int n = nums.size();
       return solve(nums, n);
    }
```

From <https://leetcode.com/problems/house-robber/submissions/1470077827/>

6) House Robber II

04 December 2024 16:33

You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed. All houses at this place are **arranged in a circle.** That means the first house is the neighbor of the last one. Meanwhile, adjacent houses have a security system connected, and **it will automatically contact the police if two adjacent houses were broken into on the same night.**
Given an integer array nums representing the amount of money of each house, return *the maximum amount of money you can rob tonight* ***without alerting the police***.

From <https://leetcode.com/problems/house-robber-ii/description/>

```
int helper(vector<int>& nums, int n, vector<int>& dp){
    if(n < 0) return 0;
    if(dp[n] != -1)return dp[n];
    int option1 = nums[n] + helper(nums, n-2, dp);
    int option2 = 0 + helper(nums, n-1, dp);
    return dp[n] = max(option1, option2);
}
int rob(vector<int>& nums) {
    int n = nums.size();
    if(n == 1) return nums[0];
    vector<int> temp1,temp2;
    vector<int>dp1(n-1, -1);
    vector<int>dp2(n-1, -1);
    for(int i = 1; i < n; i++){
        temp1.push_back(nums[i]);
    }
    for(int i = 0; i < n-1; i++){
        temp2.push_back(nums[i]);
    }
    int option1 = helper(temp1, n-2, dp1);
    int option2 = helper(temp2, n-2, dp2);
    return max(option1, option2);
}
```

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);
  }
```

16) Subset Sum Problem

06 December 2024 16:04

Given an array of positive integers, **arr[]** and a value, **target**, determine if there is a subset of the given set with sum equal to given **target.**

From <https://www.geeksforgeeks.org/problems/subset-sum-problem-1611555638/1>

```
bool helper(int index, vector<int>& arr, int target, vector<vector<int>> &dp){
    if(target == 0) return true;
    if(index == 0) return (arr[0] == target);
    if(dp[index][target] != -1) return dp[index][target];

    bool notTaken = helper(index-1, arr, target, dp);
    bool taken = false;
    if(arr[index] <= target) taken = helper(index-1, arr, target-arr[index], dp);

    return dp[index][target] = notTaken || taken;
}
bool isSubsetSum(vector<int>& arr, int target) {
    int n = arr.size();
    vector<vector<int>>dp(n, vector<int>(target+1, -1));
    return helper(n-1, arr, target, dp);
}
```

17) Partition Equal Subset Sum (based on q no 16)

06 December 2024 16:16

Given an integer array nums, return true *if you can partition the array into two subsets such that the sum of the elements in both subsets is equal or* false *otherwise.*

From <https://leetcode.com/problems/partition-equal-subset-sum/description/>

```
    bool helper(int index, vector<int>& arr, int target, vector<vector<int>> &dp){
        if(target == 0) return true;
        if(index == 0) return (arr[0] == target);
        if(dp[index][target] != -1) return dp[index][target];

        bool notTaken = helper(index-1, arr, target, dp);
        bool taken = false;
        if(arr[index] <= target) taken = helper(index-1, arr, target-arr[index],
dp);

        return dp[index][target] = notTaken || taken;
    }
    bool isSubsetSum(vector<int>& arr, int target) {
        int n = arr.size();
        vector<vector<int>>dp(n, vector<int>(target+1, -1));
        return helper(n-1, arr, target, dp);
    }
    bool canPartition(vector<int>& nums) {
        int totalSum = accumulate(nums.begin(), nums.end(), 0);
        if(totalSum % 2 != 0) return false;
        int target = totalSum/2;
        return isSubsetSum(nums, target);
    }
```

18) Minimum sum partition(based on q 16)

06 December 2024 18:33

Given an array **arr[]** containing **non-negative** integers, the task is to divide it into two sets **set1** and **set2** such that the absolute difference between their sums is minimum and find the **minimum** difference.

From <https://www.geeksforgeeks.org/problems/minimum-sum-partition3317/1?itm_source=geeksforgeeks&itm_medium=article&itm_campaign=practice_card>

```
bool helper(int index, vector<int>& arr, int target, vector<vector<int>> &dp){
    if(target == 0) return true;
    if(index == 0) return (arr[0] == target);
    if(dp[index][target] != -1) return dp[index][target];

    bool notTaken = helper(index-1, arr, target, dp);
    bool taken = false;
    if(arr[index] <= target) taken = helper(index-1, arr, target-arr[index], dp);

    return dp[index][target] = notTaken || taken;
}

int minDifference(vector<int>& arr) {
    int n = arr.size();
    int totSum = accumulate(arr.begin(), arr.end(), 0);
    int halfSum = totSum / 2;

    vector<vector<int>> dp(n, vector<int>(halfSum + 1, -1));

    int mini = 1e9;
    for (int s1 = halfSum; s1 >= 0; s1--) {
        int s2 = totSum-s1;
        if (helper(n - 1, arr, s1, dp)) {
            mini = min(mini, abs(s1-s2));
        }
    }

    return mini;
}
```

19) Perfect Sum Problem (same as q16, just the base case changes)

06 December 2024 19:24

Given an array **arr** of non-negative integers and an integer **target**, the task is to count all subsets of the array whose sum is equal to the given target.

From <https://www.geeksforgeeks.org/problems/perfect-sum-problem5633/1?utm_source=youtube&utm_medium=collab_striver_ytdescription&utm_campaign=perfect-sum-problem>

```
int findWaysUtil(int ind, int target, vector<int>& arr, vector<vector<int>>& dp) {
   // Base case: If the target sum is 0, we found a valid subset
   if (ind == 0){
      if (target == 0 && arr[0] == 0) return 2;
      if (target == 0 || target == arr[0])return 1;
      return 0;
   }

   if (dp[ind][target] != -1) return dp[ind][target];

   int notTaken = findWaysUtil(ind - 1, target, arr, dp);

   int taken = 0;
   if (arr[ind] <= target)
      taken = findWaysUtil(ind - 1, target - arr[ind], arr, dp);

   return dp[ind][target] = notTaken + taken;
}

int perfectSum(vector<int>& num, int k) {
   int n = num.size();
   vector<vector<int>> dp(n, vector<int>(k + 1, -1));
   return findWaysUtil(n - 1, k, num, dp);
}
```

20) Partitions with Given Difference (based on q no 19)

06 December 2024 19:43

Given an array **arr[]**, partition it into two subsets(possibly empty) such that each element must belong to only one subset. Let the sum of the elements of these two subsets be **sum1** and **sum2**. Given a difference **d**, count the number of partitions in which **sum1** is greater than or equal to **sum2** and the difference between **sum1** and **sum2** is equal to **d**.

From <https://www.geeksforgeeks.org/problems/partitions-with-given-difference/1?utm_source=youtube&utm_medium=collab_striver_ytdescription&utm_campaign=partitions-with-given-difference>

```
int findWaysUtil(int ind, int target, vector<int>& arr, vector<vector<int>>& dp) {
    // Base case: If the target sum is 0, we found a valid subset
    if (ind == 0){
        if (target == 0 && arr[0] == 0) return 2;
        if (target == 0 || target == arr[0])return 1;
        return 0;
    }

    if (dp[ind][target] != -1) return dp[ind][target];

    int notTaken = findWaysUtil(ind - 1, target, arr, dp);

    int taken = 0;
    if (arr[ind] <= target)
        taken = findWaysUtil(ind - 1, target - arr[ind], arr, dp);

    return dp[ind][target] = notTaken + taken;
}

int perfectSum(vector<int>& num, int k) {
    int n = num.size();
    vector<vector<int>> dp(n, vector<int>(k + 1, -1));
    return findWaysUtil(n - 1, k, num, dp);
}
int countPartitions(vector<int>& arr, int d) {
    int totalSum = accumulate(arr.begin(), arr.end(), 0);
    if(totalSum - d < 0 || (totalSum-d)%2 != 0) return 0;
    return perfectSum(arr, (totalSum-d)/2);
}
```

21) 0/1 Knapsack (based on q no 16)

06 December 2024 20:04

You are given the weights and values of items, and you need to put these items in a knapsack of capacity **capacity** to achieve the maximum total value in the knapsack. Each item is available in only one quantity.
In other words, you are given two integer arrays **val[]** and **wt[]**, which represent the values and weights associated with items, respectively. You are also given an integer **capacity**, which represents the knapsack capacity. Your task is to find the maximum sum of values of a subset of val[] such that the sum of the weights of the corresponding subset is less than or equal to **capacity**. You cannot break an item; you must either pick the entire item or leave it (0-1 property).

From <https://www.geeksforgeeks.org/problems/0-1-knapsack-problem0945/1?itm_source=geeksforgeeks&itm_medium=article&itm_campaign=practice_card>

```
int helper(int index, int W, vector<int> &val, vector<int> &wt, vector<vector<int>> &dp){
    if(index == 0){
        if(wt[0] <= W) return val[0];
        return 0;
    }
    if(dp[index][W] != -1) return dp[index][W];

    int notTaken = helper(index-1, W, val ,wt, dp);
    int taken = INT_MIN;
    if(wt[index] <= W) taken = val[index] + helper(index-1, W-wt[index], val, wt, dp);
    return dp[index][W] = max(notTaken, taken);
}
int knapSack(int capacity, vector<int> &val, vector<int> &wt) {
    int n = val.size();
    vector<vector<int>>dp(n, vector<int>(capacity+1, -1));
    return helper(n-1, capacity, val ,wt, dp);
}
```

22) Coin Change

06 December 2024 20:48

You are given an integer array coins representing coins of different denominations and an integer amount representing a total amount of money.
Return *the fewest number of coins that you need to make up that amount*. If that amount of money cannot be made up by any combination of the coins, return -1.
You may assume that you have an infinite number of each kind of coin.

From <https://leetcode.com/problems/coin-change/description/>

```
int helper(int index, vector<int> &coins, int T, vector<vector<int>> &dp){
    if(index == 0){
        if(T % coins[0] == 0) return T / coins[0];
        else return 1e9;
    }
    if(dp[index][T] != -1) return dp[index][T];
    int notTaken = 0 + helper(index-1, coins, T, dp);
    int taken = 1e9;
    if(coins[index] <= T) taken = 1 + helper(index, coins, T-coins[index], dp);
    return dp[index][T] = min(taken, notTaken);
}
int coinChange(vector<int>& coins, int amount) {
    int n = coins.size();
    vector<vector<int>> dp(n, vector<int>(amount+1, -1));
    int result = helper(n - 1, coins, amount, dp);
    return (result == 1e9) ? -1 : result;
}
```

23) Target Sum (exactly same as q no 20)

06 December 2024 20:59

You are given an integer array nums and an integer target.
You want to build an **expression** out of nums by adding one of the symbols '+' and '-' before each integer in nums and then concatenate all the integers.

- For example, if nums = [2, 1], you can add a '+' before 2 and a '-' before 1 and concatenate them to build the expression "+2-1".

Return the number of different **expressions** that you can build, which evaluates to target.

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

```
    int findWaysUtil(int ind, int target, vector<int>& arr, vector<vector<int>>& dp)
{
        if (ind == 0){
            if (target == 0 && arr[0] == 0) return 2;
            if (target == 0 || target == arr[0])return 1;
            return 0;
        }

        if (dp[ind][target] != -1) return dp[ind][target];

        int notTaken = findWaysUtil(ind - 1, target, arr, dp);

        int taken = 0;
        if (arr[ind] <= target)
            taken = findWaysUtil(ind - 1, target - arr[ind], arr, dp);

        return dp[ind][target] = notTaken + taken;
    }

    int perfectSum(vector<int>& num, int k) {
        int n = num.size();
        vector<vector<int>> dp(n, vector<int>(k + 1, -1));
        return findWaysUtil(n - 1, k, num, dp);
    }
    int countPartitions(vector<int>& arr, int d) {
        int totalSum = accumulate(arr.begin(), arr.end(), 0);
        if(totalSum - d < 0 || (totalSum-d)%2 != 0) return 0;
        return perfectSum(arr, (totalSum-d)/2);
    }
    int findTargetSumWays(vector<int>& nums, int target) {
        return countPartitions(nums, target);
    }
```

24) Coin Change II

06 December 2024 21:31

You are given an integer array coins representing coins of different denominations and an integer amount representing a total amount of money.
Return *the number of combinations that make up that amount*. If that amount of money cannot be made up by any combination of the coins, return 0.
You may assume that you have an infinite number of each kind of coin.
The answer is **guaranteed** to fit into a signed **32-bit** integer.

From <https://leetcode.com/problems/coin-change-ii/description/>

```
int helper(int index, vector<int> &coins, int T, vector<vector<int>> &dp){
    if(index == 0) return (T % coins[0] == 0);
    if(dp[index][T] != -1) return dp[index][T];
    int notTaken = 0 + helper(index-1, coins, T, dp);
    int taken = 0;
    if(coins[index] <= T) taken = helper(index, coins, T-coins[index], dp);
    return dp[index][T] = taken + notTaken;
}
int change(int amount, vector<int>& coins){
    int n = coins.size();
    vector<vector<int>> dp(n, vector<int>(amount+1, -1));
    return helper(n - 1, coins, amount, dp);
}
```

25) unbounded Knapsack (based on prev knapsack problem)

07 December 2024 17:39

Given a set of items, each with a weight and a value, represented by the array **wt** and **val** respectively. Also, a knapsack with a weight limit **capacity**.
The task is to fill the knapsack in such a way that we can get the maximum profit. Return the maximum profit.

From <https://www.geeksforgeeks.org/problems/knapsack-with-duplicate-items4201/1?utm_source=youtube&utm_medium=collab_striver_ytdescription&utm_campaign=knapsack-with-duplicate-items>

```
int helper(vector<int>& val, vector<int> &wt, int W, int index, vector<vector<int>> &dp){
    if(index == 0) return ((int)(W/wt[0]))*val[0];

    if(dp[index][W] != -1) return dp[index][W];

    int notTaken = helper(val, wt, W, index-1, dp);
    int taken = 0;
    if(wt[index] <= W){
        taken = val[index] + helper(val, wt, W-wt[index], index, dp);
    }
    return dp[index][W] = max(taken, notTaken);
}
int knapSack(vector<int>& val, vector<int>& wt, int capacity) {
    // code here
    int n = val.size();
    vector<vector<int>> dp(n, vector<int>(capacity+1, -1));
    return helper(val, wt, capacity, n-1, dp);
}
```

26) Rod cutting (q no 16)

07 December 2024 18:04

Given a rod of length **n** inches and an array of prices, **price**. price[i] denotes the value of a piece of length i. Determine the **maximum** value obtainable by cutting up the rod and selling the pieces.

From <https://www.geeksforgeeks.org/problems/rod-cutting0840/1>

```
int helper(int n, int index, vector<int> &price, vector<vector<int>> &dp){
    if(index == 0) return n*price[0];

    if(dp[index][n] != -1) return dp[index][n];

    int notTaken = 0 + helper(n, index-1, price, dp);
    int taken = INT_MIN;
    int rodLength = index+1;
    if(rodLength <= n) taken = price[index] + helper(n-rodLength, index, price, dp);

    return dp[index][n] = max(notTaken, taken);
}
int cutRod(vector<int> &price) {
    int n = price.size();
    vector<vector<int>> dp(n, vector<int>(n+1, -1));
    return helper(n, n-1, price, dp);
}
```

27) Longest Common Subsequence

07 December 2024 18:46

Given two strings text1 and text2, return *the length of their longest* ***common subsequence.*** If there is no **common subsequence**, return 0.

From <https://leetcode.com/problems/longest-common-subsequence/description/>

```
    int helper(int i, int j, string &s, string &t, vector<vector<int>> &dp){
        if(i < 0 || j < 0) return 0;
        if(dp[i][j] != -1) return dp[i][j];
        if(s[i] == t[j]) return 1 + helper(i-1, j-1, s, t, dp);
        return dp[i][j] = max(helper(i-1, j, s, t, dp), helper(i, j-1, s, t, dp));
    }
    int longestCommonSubsequence(string text1, string text2) {
        int n = text1.size();
        int m = text2.size();
        vector<vector<int>> dp(n, vector<int>(m, -1));
        return helper(n-1, m-1, text1, text2, dp);
    }
```

28) Print all LCS sequences (based on prev problem) (code not given by striver)

07 December 2024 19:32

You are given two strings **s** and **t**. Now your task is to print all longest common sub-sequences in lexicographical order.

From <https://www.geeksforgeeks.org/problems/print-all-lcs-sequences3413/1?utm_source=youtube&utm_medium=collab_striver_ytdescription&utm_campaign=print-all-lcs-sequences>

```
  void findAllLCS(int i, int j, string &s, string &t, vector<vector<int>> &dp, string currentLCS,
set<string> &lcsSet) {
      if (i == 0 || j == 0) {
          reverse(currentLCS.begin(), currentLCS.end());
          lcsSet.insert(currentLCS);
          return;
      }

      if (s[i - 1] == t[j - 1]) {
          currentLCS.push_back(s[i - 1]);
          findAllLCS(i - 1, j - 1, s, t, dp, currentLCS, lcsSet);
      } else {
          if (dp[i - 1][j] == dp[i][j])
              findAllLCS(i - 1, j, s, t, dp, currentLCS, lcsSet);
          if (dp[i][j - 1] == dp[i][j])
              findAllLCS(i, j - 1, s, t, dp, currentLCS, lcsSet);
      }
  }
  vector<string> all_longest_common_subsequences(string &s, string &t) {
      int n = s.size(), m = t.size();
      vector<vector<int>> dp(n + 1, vector<int>(m + 1, 0));

      // Fill the DP table
      for (int i = 1; i <= n; i++) {
          for (int j = 1; j <= m; j++) {
              if (s[i - 1] == t[j - 1]) {
                  dp[i][j] = 1 + dp[i - 1][j - 1];
              } else {
                  dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
              }
          }
      }

      set<string> lcsSet;
      findAllLCS(n, m, s, t, dp, "", lcsSet);

      // Convert set to vector
      vector<string> result(lcsSet.begin(), lcsSet.end());
      return result;
  }
```

29) Longest Common Substring

08 December 2024 15:51

You are given two strings **s1** and **s2**. Your task is to find the length of the **longest common substring** among the given strings.

From <https://www.geeksforgeeks.org/problems/longest-common-substring1452/1>

//same as q no 27, just isme tabulation lga ke
Dp[i][j] = 0; in else case

```
int longestCommonSubstr(string& s1, string& s2) {
    int n = s1.size();
    int m = s2.size();
    vector<vector<int>> dp(n + 1, vector<int>(m + 1, 0));
    int maxLen = 0;

    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            if (s1[i - 1] == s2[j - 1]) {
                dp[i][j] = dp[i - 1][j - 1] + 1;
                maxLen = max(maxLen, dp[i][j]);
            }
            else dp[i][j] = 0;
        }
    }
    return maxLen;
}
```

//memoized sol

```
int helper(int i, int j, string& text1, string& text2, vector<vector<int>>& dp) {
    if (i < 0 || j < 0) return 0;
    if (dp[i][j] != -1) return dp[i][j];

    if (text1[i] == text2[j]) return dp[i][j] = 1 + helper(i - 1, j - 1, text1, text2, dp);
    return dp[i][j] = 0;
}

int longestCommonSubstr(string& s1, string& s2) {
    int n = s1.size(), m = s2.size();
    vector<vector<int>> dp(n, vector<int>(m, -1));
    int maxLen = 0;

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            helper(i, j, s1, s2, dp);
            maxLen = max(maxLen, dp[i][j]);
        }
    }
    return maxLen;
}
```

30) Longest palindromic subsequence (same as q no 27)

08 December 2024 16:02

Given a string s, find *the longest palindromic* ***subsequence****'s length in* s.
A **subsequence** is a sequence that can be derived from another sequence by deleting some or no elements without changing the order of the remaining elements.

From <https://leetcode.com/problems/longest-palindromic-subsequence/description/>

```
int helper(int i, int j, string &s, string &t, vector<vector<int>> &dp){
    if(i < 0 || j < 0) return 0;
    if(dp[i][j] != -1) return dp[i][j];
    if(s[i] == t[j]) return 1 + helper(i-1, j-1, s, t, dp);
    return dp[i][j] = max(helper(i-1, j, s, t, dp), helper(i, j-1, s, t, dp));
}
int longestPalindromeSubseq(string s) {
    string s2 = s;
    reverse(s2.begin(), s2.end());
    int n = s.size();
    vector<vector<int>> dp(n, vector<int>(n, -1));
    return helper(n-1, n-1, s, s2, dp);
}
```

31) Minimum insertion steps to make a string palindrome (almost same as previous problem)

08 December 2024 16:12

Given a string s. In one step you can insert any character at any index of the string.
Return *the minimum number of steps* to make s palindrome.
A **Palindrome String** is one that reads the same backward as well as forward.

From <https://leetcode.com/problems/minimum-insertion-steps-to-make-a-string-palindrome/description/>

```
int helper(int i, int j, string &s, string &t, vector<vector<int>> &dp){
    if(i < 0 || j < 0) return 0;
    if(dp[i][j] != -1) return dp[i][j];
    if(s[i] == t[j]) return 1 + helper(i-1, j-1, s, t, dp);
    return dp[i][j] = max(helper(i-1, j, s, t, dp), helper(i, j-1, s, t, dp));
}
int minInsertions(string s) {
    string s2 = s;
    reverse(s2.begin(), s2.end());
    int n = s.size();
    vector<vector<int>> dp(n, vector<int>(n, -1));
    int lps = helper(n-1, n-1, s, s2, dp);
    return n-lps;
}
```

32) Minimum number of deletions and insertions (based on q no 27)

08 December 2024 16:20

Given two strings **s1** and **s2**. The task is to **remove or insert** the **minimum number** of characters from/in **s1** to transform it into **s2**. It could be possible that the same character needs to be removed from one point of **s1** and inserted into another point.

From <https://www.geeksforgeeks.org/problems/minimum-number-of-deletions-and-insertions0209/1?itm_source=geeksforgeeks&itm_medium=article&itm_campaign=practice_card>

```
int helper(int i, int j, string &s, string &t, vector<vector<int>> &dp){
      if(i < 0 || j < 0) return 0;

      if(dp[i][j] != -1) return dp[i][j];
      if(s[i] == t[j]) return 1 + helper(i-1, j-1, s, t, dp);
      return dp[i][j] = max(helper(i-1, j, s, t, dp), helper(i, j-1, s, t, dp));
  }
  int minOperations(string &s1, string &s2) {
      int n = s1.size();
      int m = s2.size();
      vector<vector<int>> dp(n, vector<int>(m, -1));
      int lcs = helper(n-1, m-1, s1, s2, dp);
      return n+m- 2*(lcs);
  }
```

33) Delete operation for two strings (exactly same code as prev problem)

08 December 2024 16:24

Given two strings word1 and word2, return *the minimum number of **steps** required to make* word1 *and* word2 *the same*.
In one **step**, you can delete exactly one character in either string.

From <https://leetcode.com/problems/delete-operation-for-two-strings/description/>

```
int helper(int i, int j, string &s, string &t, vector<vector<int>> &dp){
    if(i < 0 || j < 0) return 0;
    if(dp[i][j] != -1) return dp[i][j];
    if(s[i] == t[j]) return 1 + helper(i-1, j-1, s, t, dp);
    return dp[i][j] = max(helper(i-1, j, s, t, dp), helper(i, j-1, s, t, dp));
}
int minDistance(string s1, string s2){
    int n = s1.size();
    int m = s2.size();
    vector<vector<int>> dp(n, vector<int>(m, -1));
    int lcs = helper(n-1, m-1, s1, s2, dp);
    return n+m- 2*(lcs);
}
```

34) Shortest common supersequence (based on q no 27 and concept of printing of longest common subsequence)

08 December 2024 17:21

Given two strings str1 and str2, return *the shortest string that has both* str1 *and* str2 *as* ***subsequences***. If there are multiple valid strings, return **any** of them.
A string s is a **subsequence** of string t if deleting some number of characters from t (possibly 0) results in the string s.

From <https://leetcode.com/problems/shortest-common-supersequence/description/>

```
    string shortestCommonSupersequence(string s1, string s2) {
        int n = s1.size();
        int m = s2.size();
        vector<vector<int>> dp(n + 1, vector<int>(m + 1, 0));
        for (int ind1 = 1; ind1 <= n; ind1++) {
            for (int ind2 = 1; ind2 <= m; ind2++) {
                if (s1[ind1 - 1] == s2[ind2 - 1])
                    dp[ind1][ind2] = 1 + dp[ind1 - 1][ind2 - 1];
                else
                    dp[ind1][ind2] = max(dp[ind1 - 1][ind2], dp[ind1][ind2 - 1]);
            }
        }
        int i = n, j = m;
        string ans = "";
        while (i > 0 && j > 0) {
            if (s1[i - 1] == s2[j - 1]) {
                ans += s1[i - 1];
                i--;
                j--;
            } else if (dp[i - 1][j] > dp[i][j - 1]) {
                ans += s1[i - 1];
                i--;
            } else {
                ans += s2[j - 1];
                j--;
            }
        }
        while (i > 0) {
            ans += s1[i - 1];
            i--;
        }
        while (j > 0) {
            ans += s2[j - 1];
            j--;
        }
        reverse(ans.begin(), ans.end());
        return ans;
    }
```

```
// memoization
```

```
    int helper(string& s1, string& s2, int i, int j, vector<vector<int>>& dp){
        if(i < 0 || j < 0) return 0;
        if(dp[i][j] != -1) return dp[i][j];
        if(s1[i] == s2[j]) return dp[i][j] = 1 + helper(s1, s2, i-1, j-1, dp);
        return dp[i][j] = max(helper(s1, s2, i-1, j, dp), helper(s1, s2, i, j-1, 
dp));
    }
    string shortestCommonSupersequence(string s1, string s2) {
        int n = s1.length();
        int m = s2.length();
        vector<vector<int>> dp(n, vector<int>(m, -1));
        int len = helper(s1, s2, n-1, m-1, dp);
        string ans = "";
        int i = n-1; int j = m-1;
        while(i >= 0 && j >= 0){
            if(s1[i] == s2[j]){
                ans += s1[i];
                i--; j--;
            }
            else if (i > 0 && dp[i - 1][j] > (j > 0 ? dp[i][j - 1] : 0)){
                ans += s1[i];
                i--;
            }
            else{
                ans += s2[j];
                j--;
            }
        }
        while(i >= 0){
            ans += s1[i];
            i--;
        }
        while(j >= 0){
            ans += s2[j];
            j--;
        }
        reverse(ans.begin(), ans.end());
        return ans;
    }
```

35) Distinct Subsequence

09 December 2024 01:39

Given two strings s and t, return *the number of distinct* ***subsequences*** *of* s *which equals* t.
The test cases are generated so that the answer fits on a 32-bit signed integer.

From <https://leetcode.com/problems/distinct-subsequences/description/>

```
    int helper(int i, int j, string &s, string &t, vector<vector<int>> &dp){
        if(j < 0) return 1;
        if(i < 0) return 0;
        if(dp[i][j] != -1) return dp[i][j];
        if(s[i] == t[j]) return dp[i][j] = (helper(i-1, j, s, t, dp) + helper(i-1,
j-1, s, t, dp));
        return dp[i][j] = helper(i-1, j, s, t, dp);
    }
    int numDistinct(string s, string t) {
        int m = s.length();
        int n = t.length();
        vector<vector<int>>dp(m, vector<int>(n, -1));
        return helper(m-1, n-1, s, t, dp);
    }
```

36) Edit distance

09 December 2024 02:00

Given two strings word1 and word2, return *the minimum number of operations required to convert word1 to word2.*
You have the following three operations permitted on a word:

- Insert a character
- Delete a character
- Replace a character

From <https://leetcode.com/problems/edit-distance/description/>

```
    int solve(int i, int j, string &s1, string &s2, vector<vector<int>> &dp){
        if(i < 0) return j+1;
        if(j < 0) return i+1;
        if(dp[i][j] != -1) return dp[i][j];
        if(s1[i] == s2[j]) return dp[i][j] = solve(i-1, j-1, s1, s2, dp);
        return dp[i][j] = 1 + min(solve(i-1, j-1, s1, s2, dp), min(solve(i-1, j, s1,
s2, dp), solve(i, j-1, s1, s2, dp)));
    }
    int minDistance(string word1, string word2) {
        int m = word1.length();
        int n = word2.length();
        vector<vector<int>> dp(m, vector<int>(n, -1));
        return solve(m-1, n-1, word1, word2, dp);
    }
```

37) Wildcard matching

09 December 2024 02:31

Given an input string (s) and a pattern (p), implement wildcard pattern matching with support for '?' and '*' where:

- '?' Matches any single character.
- '*' Matches any sequence of characters (including the empty sequence).

The matching should cover the **entire** input string (not partial).

From <https://leetcode.com/problems/wildcard-matching/description/>

```
    bool helper(int i, int j, string &s, string &p, vector<vector<int>> &dp){
        if(i < 0 && j < 0) return true;
        if(i < 0 && j >= 0) return false;
        if(i >= 0 && j < 0){
            for(int k = 0; k <= i; k++){
                if(p[k] != '*') return false;
            }
            return true;
        }
        if(dp[i][j] != -1) return dp[i][j];
        if(p[i] == s[j] || p[i] == '?') return dp[i][j] = helper(i-1, j-1, s, p, 
dp);
        if(p[i] == '*') return dp[i][j] = helper(i-1, j, s, p, dp) || helper(i, j-1, 
s, p, dp);
        return dp[i][j] = false;
    }
    bool isMatch(string s, string p) {
        int n = s.length();
        int m = p.length();
        vector<vector<int>>dp(m, vector<int>(n, -1));
        return helper(m-1, n-1, s, p, dp);
    }
```

38) Best time to buy and sell stock II

09 December 2024 03:03

You are given an integer array prices where prices[i] is the price of a given stock on the ith day.
On each day, you may decide to buy and/or sell the stock. You can only hold **at most one** share of the stock at any time. However, you can buy it then immediately sell it on the **same day**.
Find and return *the **maximum** profit you can achieve*.

From <https://leetcode.com/problems/best-time-to-buy-and-sell-stock-ii/description/>

```
    int helper(int day, int buy, vector<int>& prices, int &n,  vector<vector<int>>
&dp){
        if(day == n) return 0;
        if(dp[day][buy] != -1) return dp[day][buy];

        int profit = 0;
        if(buy){
            profit = max(-prices[day] + helper(day+1, 0, prices, n, dp), (0 +
helper(day+1, 1, prices, n, dp)));
        }
        else{
            profit = max(prices[day] + helper(day+1, 1, prices, n, dp), (0 +
helper(day+1, 0, prices, n, dp)));
        }
        return dp[day][buy] = profit;
    }
    int maxProfit(vector<int>& prices) {
        int n = prices.size();
        vector<vector<int>> dp(n, vector<int>(2, -1));
        return helper(0, 1, prices, n, dp);
    }
```

39) Best time to buy and sell stock III

09 December 2024 03:15

You are given an array prices where prices[i] is the price of a given stock on the ith day.
Find the maximum profit you can achieve. You may complete **at most two transactions**.
Note: You may not engage in multiple transactions simultaneously (i.e., you must sell the stock before you buy again).

From <https://leetcode.com/problems/best-time-to-buy-and-sell-stock-iii/description/>

```
    int helper(int day, int buy, vector<int>& prices, int
&n,  vector<vector<vector<int>>>& dp, int cap){
        if(cap == 0) return 0;
        if(day == n) return 0;
        if(dp[day][buy][cap] != -1) return dp[day][buy][cap];

        int profit = 0;
        if(buy){
            profit = max(-prices[day] + helper(day+1, 0, prices, n, dp, cap), (0 +
helper(day+1, 1, prices, n, dp, cap)));
        }
        else{
            profit = max(prices[day] + helper(day+1, 1, prices, n, dp, cap-1), (0 +
helper(day+1, 0, prices, n, dp, cap)));
        }
        return dp[day][buy][cap] = profit;
    }
    int maxProfit(vector<int>& prices) {
        int n = prices.size();
        int cap = 2;
        vector<vector<vector<int>>> dp(n, vector<vector<int>>(2, vector<int>
(3, -1)));
        return helper(0, 1, prices, n, dp, cap);
    }
```

40) Best time to buy and sell stock IV

09 December 2024 03:16

You are given an integer array prices where prices[i] is the price of a given stock on the ith day, and an integer k.
Find the maximum profit you can achieve. You may complete at most k transactions: i.e. you may buy at most k times and sell at most k times.
Note: You may not engage in multiple transactions simultaneously (i.e., you must sell the stock before you buy again).

From <https://leetcode.com/problems/best-time-to-buy-and-sell-stock-iv/description/>

```
    int helper(int day, int buy, vector<int>& prices, int
&n,  vector<vector<vector<int>>>& dp, int cap){
        if(cap == 0) return 0;
        if(day == n) return 0;
        if(dp[day][buy][cap] != -1) return dp[day][buy][cap];

        int profit = 0;
        if(buy){
            profit = max(-prices[day] + helper(day+1, 0, prices, n, dp, cap), (0 +
helper(day+1, 1, prices, n, dp, cap)));
        }
        else{
            profit = max(prices[day] + helper(day+1, 1, prices, n, dp, cap-1), (0 +
helper(day+1, 0, prices, n, dp, cap)));
        }
        return dp[day][buy][cap] = profit;
    }
    int maxProfit(int k, vector<int>& prices) {
        int n = prices.size();
        int cap = k;
        vector<vector<vector<int>>> dp(n, vector<vector<int>>(2, vector<int>(k+
1, -1)));
        return helper(0, 1, prices, n, dp, cap);
    }
```

41) Best time to buy and sell stock with cooldown

09 December 2024 03:22

You are given an array prices where prices[i] is the price of a given stock on the ith day.
Find the maximum profit you can achieve. You may complete as many transactions as you like (i.e., buy one and sell one share of the stock multiple times) with the following restrictions:

- After you sell your stock, you cannot buy stock on the next day (i.e., cooldown one day).

Note: You may not engage in multiple transactions simultaneously (i.e., you must sell the stock before you buy again).

From <https://leetcode.com/problems/best-time-to-buy-and-sell-stock-with-cooldown/description/>

```
    int helper(int day, int buy, vector<int>& prices, int &n,  vector<vector<int>>
&dp){
        if(day >= n) return 0;
        if(dp[day][buy] != -1) return dp[day][buy];

        int profit = 0;
        if(buy){
            profit = max(-prices[day] + helper(day+1, 0, prices, n, dp), (0 +
helper(day+1, 1, prices, n, dp)));
        }
        else{
            profit = max(prices[day] + helper(day+2, 1, prices, n, dp), (0 +
helper(day+1, 0, prices, n, dp)));
        }
        return dp[day][buy] = profit;
    }
    int maxProfit(vector<int>& prices) {
        int n = prices.size();
        vector<vector<int>> dp(n, vector<int>(2, -1));
        return helper(0, 1, prices, n, dp);
    }
```

42) Best Time to Buy and Sell Stock with Transaction Fee

09 December 2024 03:27

You are given an array prices where prices[i] is the price of a given stock on the ith day, and an integer fee representing a transaction fee.
Find the maximum profit you can achieve. You may complete as many transactions as you like, but you need to pay the transaction fee for each transaction.

From <https://leetcode.com/problems/best-time-to-buy-and-sell-stock-with-transaction-fee/description/>

```
    int helper(int day, int buy, vector<int>& prices, int &n,  vector<vector<int>>
&dp, int &fee){
        if(day == n) return 0;
        if(dp[day][buy] != -1) return dp[day][buy];

        int profit = 0;
        if(buy){
            profit = max(-prices[day] + helper(day+1, 0, prices, n, dp, fee), (0 +
helper(day+1, 1, prices, n, dp, fee)));
        }
        else{
            profit = max(prices[day] - fee + helper(day+1, 1, prices, n, dp, fee),
(0 + helper(day+1, 0, prices, n, dp, fee)));
        }
        return dp[day][buy] = profit;
    }
    int maxProfit(vector<int>& prices, int fee) {
        int n = prices.size();
        vector<vector<int>> dp(n, vector<int>(2, -1));
        return helper(0, 1, prices, n, dp, fee);
    }
```

43) Longest increasing subsequence

10 December 2024 02:28

Given an integer array nums, return *the length of the longest **strictly increasing** **subsequence***
.

From <https://leetcode.com/problems/longest-increasing-subsequence/description/>

```
    int helper(vector<int>& nums, int ind, int prev_ind, int n, vector<vector<int>>
&dp){
        if(ind == n) return 0;
        if(dp[ind][prev_ind+1] != -1) return dp[ind][prev_ind+1];
        int len = 0 + helper(nums, ind+1, prev_ind, n, dp);
        if(prev_ind == -1 || nums[ind] > nums[prev_ind]){
            len = max(len, 1 + helper(nums, ind+1, ind, n, dp));
        }
        return dp[ind][prev_ind+1] = len;
    }
    int lengthOfLIS(vector<int>& nums) {
        int n = nums.size();
        vector<vector<int>>dp(n, vector<int>(n+1, -1));
        return helper(nums, 0, -1, n, dp);
    }
```

//space optimization: very imp, as iss se related saari problem main yhi as a sample code use hoga

```
    int lengthOfLIS(vector<int>& arr) {
        int n = arr.size();
        vector<int> dp(n,1);

        int maxi = 1;

        for(int i=0; i<=n-1; i++){

            for(int prev_index = 0; prev_index <=i-1; prev_index ++){

                if(arr[i] > arr[prev_index] && 1 + dp[prev_index] > dp[i]){
                    dp[i] = 1 + dp[prev_index];
                }
            }

            if(dp[i] > maxi)
                maxi = dp[i];
        }
        return maxi;
    }
```

44) Print longest increasing subsequence

11 December 2024 01:56

Given an integer **n** and an array of integers **arr**, return the **Longest Increasing Subsequence** which is *Index-wise* lexicographically smallest.
Note - A subsequence S1 is **Index-wise lexicographically smaller** than a subsequence S2 if in the first position where S1 and S2 differ, subsequence S1 has an element that appears **earlier** in the array arr than the corresponding element in S2.

From <https://www.geeksforgeeks.org/problems/printing-longest-increasing-subsequence/1?utm_source=youtube&utm_medium=collab_striver_ytdescription&utm_campaign=printing-longest-increasing-subsequence>

```
int solve(int i, int prev, vector<int>& nums, vector<vector<int>>& dp) {
    if (i == nums.size()) return 0;
    if (dp[i][prev + 1] != -1) return dp[i][prev + 1];

    int len = solve(i + 1, prev, nums, dp);
    if (prev == -1 || nums[i] > nums[prev]) {
        len = max(len, 1 + solve(i + 1, i, nums, dp));
    }

    return dp[i][prev + 1] = len;
}

vector<int> longestIncreasingSubsequence(int n, vector<int>& nums) {
    vector<vector<int>> dp(n, vector<int>(n + 1, -1));
    int len = solve(0, -1, nums, dp); // Fill DP table

    // Reconstruct the LIS
    vector<int> lis;
    int i = 0, prev = -1;

    while (i < n && len > 0) {
        if (prev == -1 || nums[i] > nums[prev]) {
            int take = 1 + (i + 1 < n ? dp[i + 1][i + 1] : 0);
            if (take == len) {
                lis.push_back(nums[i]);
                prev = i;
                len--; // Reduce target length
            }
        }
        i++;
    }

    return lis;
}
```

45) *Longest Increasing Subsequence (special stl used)

11 December 2024 02:13

Given an array **arr[]** of integers, the task is to find the **length** of the **Longest Strictly Increasing Subsequence (LIS)**.

A subsequence is considered **strictly increasing** if each element in the subsequence is strictly less than the next element.

From <https://www.geeksforgeeks.org/problems/longest-increasing-subsequence-1587115620/1?utm_source=youtube&utm_medium=collab_striver_ytdescription&utm_campaign=longest-increasing-subsequence>

```
int longestSubsequence(vector<int>& arr) {
    int n = arr.size();
    vector<int> temp;
    temp.push_back(arr[0]);

    int len = 1;

    for (int i = 1; i < n; i++) {
        if (arr[i] > temp.back()) {
            temp.push_back(arr[i]);
            len++;
        } else {
            int ind = lower_bound(temp.begin(), temp.end(), arr[i]) - temp.begin();
            temp[ind] = arr[i];
        }
    }

    return len;
}
```

46) Largest Divisible subset

11 December 2024 02:48

Given a set of **distinct** positive integers nums, return the largest subset answer such that every pair (answer[i], answer[j]) of elements in this subset satisfies:

- answer[i] % answer[j] == 0, or
- answer[j] % answer[i] == 0

If there are multiple solutions, return any of them.

From <https://leetcode.com/problems/largest-divisible-subset/description/>

```
    int solve(int i, int prev, vector<int>& nums, vector<vector<int>>& dp) {
        if (i == nums.size()) return 0;
        if (dp[i][prev + 1] != -1) return dp[i][prev + 1];
        int notTake = solve(i + 1, prev, nums, dp);
        int take = 0;
        if (prev == -1 || nums[i] % nums[prev] == 0) {
            take = 1 + solve(i + 1, i, nums, dp);
        }
        return dp[i][prev + 1] = max(take, notTake);
    }
    vector<int> largestDivisibleSubset(vector<int>& nums) {
        int n = nums.size();
        sort(nums.begin(), nums.end()); // Important for divisibility checks
        vector<vector<int>> dp(n, vector<int>(n + 1, -1));
        solve(0, -1, nums, dp);
        // Reconstruct subset
        vector<int> res;
        int i = 0, prev = -1;
        int len = dp[0][0]; // Max length
        while (i < n && len > 0) {
            if (prev == -1 || nums[i] % nums[prev] == 0) {
                int take = 1 + (i + 1 < n ? dp[i + 1][i + 1] : 0);
                if (take == len) {
                    res.push_back(nums[i]);
                    prev = i;
                    len--;
                }
            }
            i++;
        }
        return res;
    }
```

47) Longest Increasing Subsequence

11 December 2024 03:09

You are given an array of words where each word consists of lowercase English letters.
word$_A$ is a **predecessor** of word$_B$ if and only if we can insert **exactly one** letter anywhere in word$_A$ **without changing the order of the other characters** to make it equal to word$_B$.

- For example, "abc" is a **predecessor** of "abac", while "cba" is not a **predecessor** of "bcad".

A **word chain** is a sequence of words [word$_1$, word$_2$, ..., word$_k$] with k >= 1, where word$_1$ is a **predecessor** of word$_2$, word$_2$ is a **predecessor** of word$_3$, and so on. A single word is trivially a **word chain** with k == 1.

From <https://leetcode.com/problems/longest-string-chain/description/>

```
    bool compare(string& s1, string& s2){
        if(s1.size() != s2.size() + 1) return false;

        int first = 0;
        int second = 0;

        while(first < s1.size()){
            if(second < s2.size() && s1[first] == s2[second]){
                first ++;
                second ++;
            }
            else first ++;
        }
        if(first == s1.size() && second == s2.size()) return true;
        else return false;
    }
    static bool comp(string& s1, string& s2){
        return s1.size() < s2.size();
    }

    int longestStrChain(vector<string>& arr){
        int n = arr.size();

        //sort the array

        sort(arr.begin(), arr.end(),comp);
        vector<int> dp(n,1);

        int maxi = 1;

        for(int i=0; i<=n-1; i++){

            for(int prev_index = 0; prev_index <=i-1; prev_index ++){

                if( compare(arr[i], arr[prev_index]) && 1 + dp[prev_index] > dp[i]){
                    dp[i] = 1 + dp[prev_index];
                }
            }

            if(dp[i] > maxi)
                maxi = dp[i];
        }
        return maxi;
    }
```

48) Longest Bitonic Sequence

11 December 2024 03:36

A Bitonic Sequence is a sequence of numbers that is first strictly increasing and then strictly decreasing.

A strictly ascending order sequence is also considered bitonic, with the decreasing part as empty, and same for a strictly descending order sequence.

From <https://www.naukri.com/code360/problems/longest-bitonic-sequence_1062688?source=youtube&campaign=striver_dp_videos&utm_source=youtube&utm_medium=affiliate&utm_campaign=striver_dp_videos&leftPanelTabValue=PROBLEM>

```
#include <bits/stdc++.h>
int longestBitonicSubsequence(vector<int>& arr, int n)
{
    vector<int> dp1(n, 1);
    vector<int> dp2(n, 1);
    for (int i = 0; i < n; i++) {
        for (int prev_index = 0; prev_index < i; prev_index++) {
            if (arr[prev_index] < arr[i]) {
                dp1[i] = max(dp1[i], 1 + dp1[prev_index]);
            }
        }
    }
    for (int i = n - 1; i >= 0; i--) {
        for (int prev_index = n - 1; prev_index > i; prev_index--) {
            if (arr[prev_index] < arr[i]) {
                dp2[i] = max(dp2[i], 1 + dp2[prev_index]);
            }
        }
    }
    int maxi = -1;
    for (int i = 0; i < n; i++) {
        maxi = max(maxi, dp1[i] + dp2[i] - 1);
    }
    return maxi;
}
```

49) No of longest increasing subsequence

11 December 2024 03:52

Given an integer array nums, return *the number of longest increasing subsequences.*
Notice that the sequence has to be **strictly** increasing.

From <https://leetcode.com/problems/number-of-longest-increasing-subsequence/description/>

```
    int findNumberOfLIS(vector<int>& arr) {
        int n = arr.size();
        vector<int> dp(n, 1);
        vector<int> ct(n, 1);
        int maxi = 1;
        for (int i = 0; i < n; i++) {
            for (int prev_index = 0; prev_index < i; prev_index++) {
                if (arr[prev_index] < arr[i] && dp[prev_index] + 1 > dp[i]) {
                    dp[i] = dp[prev_index] + 1;
                    ct[i] = ct[prev_index];
                } else if (arr[prev_index] < arr[i] && dp[prev_index] + 1 == dp[i])
{
                    ct[i] = ct[i] + ct[prev_index];
                }
            }
            maxi = max(maxi, dp[i]);
        }
        int numberOfLIS = 0;
        for (int i = 0; i < n; i++) {
            if (dp[i] == maxi) {
                numberOfLIS += ct[i];
            }
        }
        return numberOfLIS;
    }
```

50) Matrix Chain Multiplication

13 December 2024 05:48

Given a sequence of matrices, find the most efficient way to multiply these matrices together. The efficient way is the one that involves the least number of multiplications. The dimensions of the matrices are given in an array **arr[]** of size **n** (such that **n** = number of matrices + 1) where the ith matrix has the dimensions **(arr[i-1] x arr[i])**.

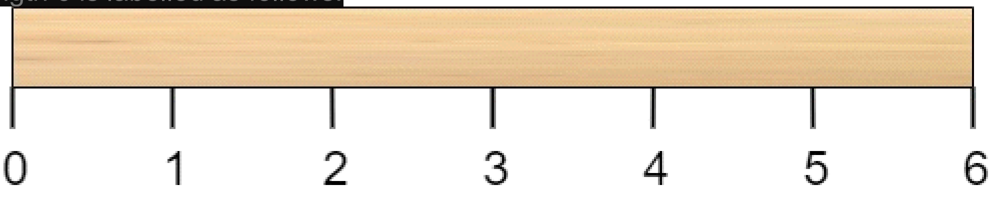
From <https://www.geeksforgeeks.org/problems/matrix-chain-multiplication0303/1?utm_source=youtube&utm_medium=collab_striver_ytdescription&utm_campaign=matrix-chain-multiplication>

```
int helper(int i, int j, vector<int> &arr, vector<vector<int>> &dp){
    if(i == j) return 0;
    int mini = 1e9;
    if(dp[i][j] != -1) return dp[i][j];
    for(int k = i; k < j; k++){
        int steps = arr[i-1]*arr[j]*arr[k] + helper(i, k, arr, dp) + helper(k+1, j, arr, dp);
        mini = min(steps, mini);
    }
    return dp[i][j] = mini;
}
int matrixMultiplication(vector<int> &arr) {
    int n = arr.size();
    vector<vector<int>> dp(n, vector<int>(n, -1));
    return helper(1, n-1, arr, dp);
}
```

51) Minimum cost to cut a stick

13 December 2024 06:23

Given a wooden stick of length n units. The stick is labelled from 0 to n. For example, a stick of length **6** is labelled as follows:



Given an integer array cuts where cuts[i] denotes a position you should perform a cut at.
You should perform the cuts in order, you can change the order of the cuts as you wish.
The cost of one cut is the length of the stick to be cut, the total cost is the sum of costs of all cuts.
When you cut a stick, it will be split into two smaller sticks (i.e. the sum of their lengths is the length of the stick before the cut). Please refer to the first example for a better explanation.

From <https://leetcode.com/problems/minimum-cost-to-cut-a-stick/description/>

```
    int helper(int i, int j, vector<int>& cuts, vector<vector<int>> &dp){
        if(i > j) return 0;
        int mini = 1e9;
        if(dp[i][j] != -1) return dp[i][j];
        for(int k = i; k <= j; k++){
            int cost = cuts[j+1] - cuts[i-1] + helper(i, k-1, cuts, dp) + helper(k+
1, j, cuts, dp);
            mini = min(mini, cost);
        }
        return dp[i][j] = mini;
    }
    int minCost(int n, vector<int>& cuts) {
        int c = cuts.size();
        cuts.push_back(n);
        cuts.insert(cuts.begin(), 0);
        sort(cuts.begin(), cuts.end());
        vector<vector<int>>dp(c+1, vector<int>(c+1, -1));
        return helper(1, c, cuts, dp);
    }
```

52) Burst Balloons

13 December 2024 06:52

You are given n balloons, indexed from 0 to n - 1. Each balloon is painted with a number on it represented by an array nums. You are asked to burst all the balloons.
If you burst the ith balloon, you will get nums[i - 1] * nums[i] * nums[i + 1] coins. If i - 1 or i + 1 goes out of bounds of the array, then treat it as if there is a balloon with a 1 painted on it.
Return *the maximum coins you can collect by bursting the balloons wisely*.

From <https://leetcode.com/problems/burst-balloons/description/>

```
    int helper(int i, int j, vector<int>& nums, vector<vector<int>> &dp){
        if(i > j) return 0;
        int maxi = -1e9;
        if(dp[i][j] != -1) return dp[i][j];
        for(int k = i; k <= j; k++){
            int cost = nums[i-1]*nums[k]*nums[j+1] + helper(i, k-1, nums, dp) +
helper(k+1, j, nums, dp);
            maxi = max(maxi, cost);
        }
        return dp[i][j] = maxi;
    }
    int maxCoins(vector<int>& nums) {
        int n = nums.size();
        nums.push_back(1);
        nums.insert(nums.begin(), 1);
        vector<vector<int>>dp(n+1, vector<int>(n+1, -1));
        return helper(1, n, nums, dp);
    }
```

53) Boolean Parenthesization

14 December 2024 01:53

Given a boolean expression **s** of length **n** with following symbols.
Symbols
  'T' ---> true
  'F' ---> false
and following operators filled between symbols
Operators
  & ---> boolean AND
  | ---> boolean OR
  ^ ---> boolean XOR
Count the number of ways we can parenthesize the expression so that the value of expression evaluates to true.
Note: The answer can be large, so return it with **modulo 1003**

From <https://www.geeksforgeeks.org/problems/boolean-parenthesization5610/1?itm_source=geeksforgeeks&itm_medium=article&itm_campaign=practice_card>

```
#define ll long long
const int mod = 1003;

int f(int i, int j, int isTrue, string &exp, vector<vector<vector<ll>>> &dp) {
  // Base cases
  if (i > j) return 0;

  if (i == j) {
    if (isTrue) return exp[i] == 'T' ? 1 : 0;
    else return exp[i] == 'F' ? 1 : 0;
  }

  // Check DP table
  if (dp[i][j][isTrue] != -1) return dp[i][j][isTrue];

  ll ways = 0;

  // Iterate through operators
  for (int ind = i + 1; ind <= j - 1; ind += 2) {
    ll lT = f(i, ind - 1, 1, exp, dp) % mod;
    ll lF = f(i, ind - 1, 0, exp, dp) % mod;
    ll rT = f(ind + 1, j, 1, exp, dp) % mod;
    ll rF = f(ind + 1, j, 0, exp, dp) % mod;

    if (exp[ind] == '&') {
      if (isTrue) ways = (ways + (lT * rT) % mod) % mod;
      else ways = (ways + (lF * rT) % mod + (lT * rF) % mod + (lF * rF) % mod) % mod;
    } else if (exp[ind] == '|') {
      if (isTrue) ways = (ways + (lT * rT) % mod + (lT * rF) % mod + (lF * rT) % mod) % mod;
      else ways = (ways + (lF * rF) % mod) % mod;
    } else if (exp[ind] == '^') {
      if (isTrue) ways = (ways + (lT * rF) % mod + (lF * rT) % mod) % mod;
      else ways = (ways + (lT * rT) % mod + (lF * rF) % mod) % mod;
    }
  }

  // Store result in DP table
  return dp[i][j][isTrue] = ways;
}

int countWays(int n, string s) {
  vector<vector<vector<ll>>> dp(n, vector<vector<ll>>(n, vector<ll>(2, -1)));
  return f(0, n - 1, 1, s, dp);
}
```

54) Palindrome Partitioning II

14 December 2024 02:40

Given a string s, partition s such that every
substring
 of the partition is a
palindrome
.
Return *the **minimum** cuts needed for a palindrome partitioning of* s.

From <https://leetcode.com/problems/palindrome-partitioning-ii/description/>

```
bool isPalindrome(int i, int j, string &s) {
    while (i < j) {
        if (s[i] != s[j]) return false;
        i++;
        j--;
    }
    return true;
}
int minPartitions(int i, int n, string &str, vector<int> &dp) {
    if (i == n) return 0;
    if (dp[i] != -1) return dp[i];
    int minCost = INT_MAX;
    for (int j = i; j < n; j++) {
        if (isPalindrome(i, j, str)) {
            int cost = 1 + minPartitions(j + 1, n, str, dp);
            minCost = min(minCost, cost);
        }
    }
    return dp[i] = minCost;
}
int minCut(string s) {
    int n = s.size();
    vector<int> dp(n, -1);
    return minPartitions(0, n, s, dp) - 1;
}
```

55) Partition array for maximum sum

14 December 2024 02:50

Given an integer array arr, partition the array into (contiguous) subarrays of length at most k. After partitioning, each subarray has their values changed to become the maximum value of that subarray.

From <https://leetcode.com/problems/partition-array-for-maximum-sum/description/>

```
int f(int ind, vector<int> &num, int k, vector<int> &dp) {
    int n = num.size();
    if (ind == n) return 0;
    if (dp[ind] != -1) return dp[ind];
    int len = 0;
    int maxi = INT_MIN;
    int maxAns = INT_MIN;
    for (int j = ind; j < min(ind + k, n); j++) {
        len++;
        maxi = max(maxi, num[j]);
        int sum = len * maxi + f(j + 1, num, k, dp);
        maxAns = max(maxAns, sum);
    }
    return dp[ind] = maxAns;
}
int maxSumAfterPartitioning(vector<int>& num, int k) {
    int n = num.size();
    vector<int> dp(n, -1);
    return f(0, num, k, dp);
}
```

56) Count square submatrices with all ones

14 December 2024 03:02

Given a m * n matrix of ones and zeros, return how many square submatrices have all ones.

From <https://leetcode.com/problems/count-square-submatrices-with-all-ones/description/>

```
int helper(int i, int j, vector<vector<int>>& matrix, vector<vector<int>>& dp){
    if(i >= matrix.size() || j >= matrix[0].size()) return 0;
    if(matrix[i][j] == 0) return 0;
    if(dp[i][j] != -1) return dp[i][j];
    int down = helper(i+1, j, matrix, dp);
    int diag = helper(i+1, j+1, matrix, dp);
    int right = helper(i, j+1, matrix, dp);
    return dp[i][j] = 1 + min({down, diag, right});
}
int countSquares(vector<vector<int>>& matrix) {
    int m = matrix.size();
    int n = matrix[0].size();
    vector<vector<int>>dp(m, vector<int>(n, -1));
    int result = 0;
    for(int i = 0; i < m; i++){
        for(int j = 0; j < n; j++){
            if(matrix[i][j] == 1){
                result += helper(i, j, matrix, dp);
            }
        }
    }
    return result;
}
```

57) Maximise the cut segment

13 February 2025 16:50

Given an integer **n** denoting the Length of a line segment. You need to cut the line segment in such a way that the cut length of a line segment each time is either **x** , **y** or **z**. Here x, y, and z are integers.
After performing all the cut operations, your total number of cut segments must be maximum. Return the maximum number of cut segments possible.
Note: if no segment can be cut then return 0.

From <https://www.geeksforgeeks.org/problems/cutted-segments1642/1>

```
int helper(int n, int x, int y, int z, vector<int> &dp){
    if(n == 0) return 0;
    if(n < 0) return INT_MIN;
    if(dp[n] != -1) return dp[n];

    int a = 1+helper(n-x, x, y, z, dp);
    int b = 1+helper(n-y, x, y, z, dp);
    int c = 1+helper(n-z, x, y, z, dp);
    return dp[n] = max(a, (max(b, c)));
}
int maximizeTheCuts(int n, int x, int y, int z) {
    vector<int> dp(n+1, -1);
    if(helper(n, x, y, z, dp) < 0) return 0;
    else return helper(n, x, y, z, dp);

}
```

58) Number of dice rolls with target sum

14 February 2025 17:16

You have n dice, and each dice has k faces numbered from 1 to k.
Given three integers n, k, and target, return *the number of possible ways (out of the* k^n *total ways) to roll the dice, so the sum of the face-up numbers equals* target. Since the answer may be too large, return it **modulo** $10^9 + 7$.

From <https://leetcode.com/problems/number-of-dice-rolls-with-target-sum/description/>

```
int helper(int n, int k, int target, vector<vector<int>>& dp) {
    if (target < 0) return 0;
    if (target == 0 && n != 0) return 0;
    if (target != 0 && n == 0) return 0;
    if (target == 0 && n == 0) return 1;
    if (dp[n][target] != -1) return dp[n][target];
    int ans = 0;
    for (int i = 1; i <= k; i++) {
        ans = (ans + helper(n - 1, k, target - i, dp)) % 1000000007;
    }
    return dp[n][target] = ans;
}
int numRollsToTarget(int n, int k, int target) {
    vector<vector<int>> dp(n + 1, vector<int>(target + 1, -1));
    return helper(n, k, target, dp);
}
```

59) Perfect Sq

14 February 2025 17:35

Given an integer n, return *the least number of perfect square numbers that sum to* n.
A **perfect square** is an integer that is the square of an integer; in other words, it is the product of some integer with itself. For example, 1, 4, 9, and 16 are perfect squares while 3 and 11 are not.

From <https://leetcode.com/problems/perfect-squares/description/>

```
int helper(int n, vector<int> &dp){
    if(n == 0) return 0;
    int minCount = INT_MAX;
    if(dp[n] != -1) return dp[n];
    for(int i = 1; i*i <= n; i++){
        int result = 1 + helper(n-i*i, dp);
        minCount = min (minCount, result);
    }
    return dp[n] = minCount;
}
int numSquares(int n) {
    vector<int>dp(n+1, -1);
    return helper(n, dp);
}
```

60) Min cost for tickets

14 February 2025 17:46

You have planned some train traveling one year in advance. The days of the year in which you will travel are given as an integer array days. Each day is an integer from 1 to 365.
Train tickets are sold in **three different ways**:

- a **1-day** pass is sold for costs[0] dollars,
- a **7-day** pass is sold for costs[1] dollars, and
- a **30-day** pass is sold for costs[2] dollars.

The passes allow that many days of consecutive travel.

- For example, if we get a **7-day** pass on day 2, then we can travel for 7 days: 2, 3, 4, 5, 6, 7, and 8.

Return *the minimum number of dollars you need to travel every day in the given list of days*.

From <https://leetcode.com/problems/minimum-cost-for-tickets/description/>

```
int helper(vector<int>& days, vector<int>& costs, int i, vector<int>&dp){
    if(i >= days.size()) return 0;
    if(dp[i] != -1) return dp[i];
    int cost1 = costs[0] + helper(days, costs, i+1, dp);
    int j = i;
    int endDay = days[i] + 7 - 1;
    while(j < days.size() && days[j] <= endDay){
        j++;
    }
    int cost7 = costs[1] + helper(days, costs, j, dp);
    j = i;
    endDay = days[i] + 30 - 1;
    while(j < days.size() && days[j] <= endDay){
        j++;
    }
    int cost30 = costs[2] + helper(days, costs, j, dp);
    return dp[i] = min(cost1, min(cost7, cost30));
}
int mincostTickets(vector<int>& days, vector<int>& costs) {
    vector<int> dp(days.size(), -1);
    return helper(days, costs, 0, dp);
}
```

61) Painting the Fence

14 February 2025 18:30

Given a fence with **n** posts and **k** colours, find out the number of ways of painting the fence so that **not more than two** consecutive posts have the same colours.
Answers are guaranteed to be fit into a 32 bit integer.

From <https://www.geeksforgeeks.org/problems/painting-the-fence3727/1>

```
int helper(int n, int k, vector<int> &dp){
    if(n == 1) return k;
    if(n == 2) return k*(k-1) + k;
    if(dp[n] != -1) return dp[n];
    return dp[n] = (k-1)*(helper(n-1, k, dp) + helper(n-2, k, dp));
}
int countWays(int n, int k) {
    vector<int> dp(n+1, -1);
    return helper(n, k, dp);
}
```

62) Maximal Square

15 February 2025 14:13

Given an m x n binary matrix filled with 0's and 1's, *find the largest square containing only* 1's *and return its area*.

From <https://leetcode.com/problems/maximal-square/description/>

```
    int solve(vector<vector<char>>& matrix, int i, int j, int row, int col, int&
maxi, vector<vector<int>>&dp){
        //base case
        if(i >= row || j >= col) return 0;
        if(dp[i][j] != -1) return dp[i][j];
        //explore all 3 directions
        int right = solve(matrix, i, j+1, row, col, maxi, dp);
        int diagonal = solve(matrix, i+1, j+1, row, col, maxi, dp);
        int down = solve(matrix, i+1, j, row, col, maxi, dp);
        //check can we build square from current position
        if(matrix[i][j] == '1'){
            int ans = 1 + min(right, min(down, diagonal));
            //ye imp h
            maxi = max(maxi, ans);
            return dp[i][j] = ans;
        }
        else return dp[i][j] = 0; //agr zero pe hi khade h to answer bhi 0 hoga
    }
    int maximalSquare(vector<vector<char>>& matrix) {
        int i = 0, j = 0;
        int row = matrix.size();
        int col = matrix[0].size();
        vector<vector<int>>dp(row+1, vector<int>(col+1, -1));
        int maxi = 0;
        int ans = solve(matrix, i , j, row, col, maxi, dp);
        return maxi*maxi;
    }
```

28)

17 February 2025 19:21

You are given two strings ***'s1'*** and ***'s2'***.

Return the longest common subsequence of these strings.

If there's no such string, return an empty string. If there are multiple possible answers, return any such string.

Note:
Longest common subsequence of string 's1' and 's2' is the longest subsequence of 's1' that is also a subsequence of 's2'. A 'subsequence' of 's1' is a string that can be formed by deleting one or more (possibly zero) characters from 's1'.

From <https://www.naukri.com/code360/problems/print-longest-common-subsequence_8416383?leftPanelTabValue=PROBLEM>

```
int solve(int n, int m, string &s1, string &s2, vector<vector<int>> &dp) {
    if (n < 0 || m < 0) {
        return 0;
    }
    if (dp[n][m] != -1) {
        return dp[n][m];
    }
    if (s1[n] == s2[m]) {
        return dp[n][m] = 1 + solve(n - 1, m - 1, s1, s2, dp);
    }
    return dp[n][m] = max(solve(n - 1, m, s1, s2, dp), solve(n, m - 1, s1, s2, dp));
}
string findLCS(int n, int m, string &s1, string &s2) {  // Accepts four arguments
    vector<vector<int>> dp(n, vector<int>(m, -1));
    // Get LCS length using the solve function
    int lcs_length = solve(n - 1, m - 1, s1, s2, dp);

    // Reconstruct the LCS string
    string lcs = "";
    int i = n - 1, j = m - 1;
    while (i >= 0 && j >= 0) {
        if (s1[i] == s2[j]) {
            lcs += s1[i];
            i--;
            j--;
        } else if (i > 0 && dp[i - 1][j] >= (j > 0 ? dp[i][j - 1] : 0)) {
            i--;
        } else {
            j--;
        }
    }
    // Reverse the LCS string since we built it from the end
    reverse(lcs.begin(), lcs.end());
    return lcs;
}
```