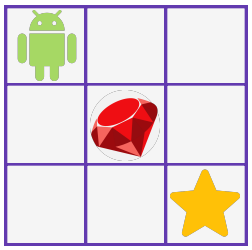
Dynammic Programming-2

Assignment Solutions

PW SKILLS

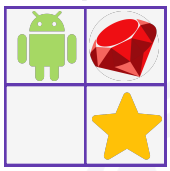
Q1. You are given an m x n integer array grid. There is a robot initially located at the top-left corner (i.e., grid[0][0]). The robot tries to move to the bottom-right corner (i.e., grid[m - 1][n - 1]). The robot can only move either down or right at any point in time.
An obstacle and space are marked as 1 or 0 respectively in the 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.

Example 1:



Input: obstacleGrid = [[0,0,0],[0,1,0],[0,0,0]]
Output: 2
Explanation: There is one obstacle in the middle of the 3x3 grid above.
There are two ways to reach the bottom-right corner:
1. Right -> Right -> Down -> Down
2. Down -> Down -> Right -> Right

Example 2:



Input: obstacleGrid = [[0,1],[0,0]]
Output: 1

Solution :

Code : ASS_Code1.java

Output :

```
The desired output is : 2
```

Approach :

- Array dp stores the number of paths which pass this point. The whole algorithm is to sum up the paths from left grid and up grid.
- 'if (row[j] == 1) dp[j] = 0;
- It means if there is an obstacle at this point, all the paths passing this point will no longer be valid.
- In other words, the grid right of the obstacle can be reached only by the grid which lies up to it.

Q2. 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.

Example 1:
Input: nums = [2,3,2]
Output: 3
Explanation: You cannot rob house 1 (money = 2) and then rob house 3 (money = 2), because they are adjacent houses.

Example 2:
Input: nums = [1,2,3,1]
Output: 4
Explanation: Rob house 1 (money = 1) and then rob house 3 (money = 3).
Total amount you can rob = 1 + 3 = 4.

Example 3:
Input: nums = [1,2,3]
Output: 3

Solution :
Code : ASS_Code2.java

Output :

```
The desired output is : 3
```

Approach :

- At every i-th house we have two choices to make, i.e., rob the i-th house or don't rob it.
- **Case1 :** Don't rob the i-th house - then we can rob the i-1 th house...so we will have max money robbed till i-1 th house
- **Case 2 :** Rob the i-th house - then we can't rob the i-1 th house but we can rob i-2 th house....so we will have max money robbed till i-2 th house + money of i-th house.
- If the array is [1,5,3] then the robber will rob the 1st index house because arr[1] > arr[0]+arr[2] (i.e., at last index, arr[i-1] > arr[i-2]+arr[i]).
- If the array is [1,2,3] then robber will rob the 0th and 2nd index house because arr[0]+arr[2] > arr[1] (i.e., at last index, arr[i-2] + arr[i] > arr[i-1]).
- Since you cannot rob both the first and last house, just create two separate vectors, one excluding the first house, and another excluding the last house.

Q3. 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.

Example 1:

1	3	1
1	5	1
4	2	1

Input: grid = [[1,3,1],[1,5,1],[4,2,1]]
Output: 7
Explanation: Because the path 1 → 3 → 1 → 1 → 1 minimizes the sum.

Example 2:
Input: grid = [[1,2,3],[4,5,6]]
Output: 12

Solution :
Code : ASS_Code3.java

Output :

```
The desired output is : 7
```

Approach :

- we can move either down or right.
- for the grid[0][0] we don't update it.
- so for the 0th row or column we will update the cost by adding the cost before that column :
  - for the 0th row --> grid[i][j] += grid[i][j-1];
  - for the 0th col --> grid[i][j] += grid[i-1][j];
- Now for the other elements we update them adding that block value to MIN of the value from the top element above them and the left element to that block (since we can from either down or right , therefore to reach any element we enter in it either from top of it or left of it).

Q4. 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.

Example 1:
Input: nums = [1,5,11,5]
Output: true
Explanation: The array can be partitioned as [1, 5, 5] and [11].

Example 2:
Input: nums = [1,2,3,5]

Output: **false**
Explanation: The array cannot be partitioned into equal sum subsets.

Solution :
Code : ASS_Code4.java

Output :

```
The desired output is : true
```

Approach :

- Since we have to make two subset both having equal sum then our first condition is to check whether the sum of a given array can be divided in two equal parts which is if total sum is odd then partition is not possible at all and if sum is even then there is chance.
- **For example:**
  1.) arr1-> [1,5,11,5] and 2.) arr2 -> [1,5,3,11]
- Both arr1 and arr2 have even sum but 1st can be partitioned into ([1,5,5] & [11]) and 2nd can not.
- In this case we have n elements in array and we have two choices to make whether to keep it in subset1 or subset2 (inclusion in one is direct exclusion in other) and weight of subset will be sum/2.
- So now what our target remains is we have to take care about only one subset because if one subset with weight sum/2 is possible then other subset will surely have the weight sum/2.
- So now using subset sum problem code we have to just check if it's possible to have a subset having sum = totalSum/2;

Q5.You are given an integer array of 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.

Example 1:
Input: amount = 5, coins = [1,2,5]
Output: 4
Explanation: there are four ways to make up the amount:
5=5
5=2+2+1
5=2+1+1+1
5=1+1+1+1+1

Example 2:
Input: amount = 3, coins = [2]
Output: 0
Explanation: the amount of 3 cannot be made up just with coins of 2.

Example 3:

Input: amount = 10, coins = [10]
Output: 1

Solution :
Code : ASS_Code5.java

Output :

```
The desired output is : 4
```

Approach :

- If the highest coin does not exceed the required sum, then the number of ways that the sum can be made will be equal to the number of ways that the same sum can be made without the highest coin + the number of ways in which a value of sum - highest coin can be made.
- Else, if the highest coin exceeds the required sum, then the number of ways this sum can be made is equal to the number of ways it could be made without this coin.

Note: Try solving this problem without taking an extra space.