Dynammic Programming-1

Assignment Solutions

PW SKILLS

Q1. There are n stairs, a person standing at the bottom wants to reach the top. The person can climb either 1,2,3...m stairs at a time where m is a user given integer. Count the number of ways the person can reach the top.

Input 1 : n = 5 , m = 3
Output 1: 7

Solution :

Code : ASS_Code1.java

Output :

```
Number of ways = 7
```

Approach :

- The following recurrence relation has been used :
  - ways(n, m) = ways(n-1, m) + ways(n-2, m) + ... ways(n-m, m)
- We create a table res[] in bottom up manner using the following relation:
- res[i] = res[i] + res[i-j] for every (i-j) >= 0
- Such that the ith index of the array will contain the number of ways required to reach the ith step considering all the possibilities of climbing (i.e. from 1 to i).

Q2. The Tribonacci sequence Tn is defined as follows:
T0 = 0, T1 = 1, T2 = 1, and Tn+3 = Tn + Tn+1 + Tn+2 for n >= 0.
Given n, return the value of nth tribonacci number.

Example 1:
Input: n = 4
Output: 4
Explanation:
T_3 = 0 + 1 + 1 = 2
T_4 = 1 + 1 + 2 = 4

Example 2:
Input: n = 25
Output: 1389537

Solution :

Code : ASS_Code2.java

Output :

```
Nth tribonacci number is : 1389537
```

Approach :

- We use an array to store the calculated values so that repeating values can be fetched without spending all that computing time on calculating the same thing again.

Q3. 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.

Example 1:
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 2:
Input: nums = [2,7,9,3,1]
Output: 12
Explanation: Rob house 1 (money = 2), rob house 3 (money = 9) and rob house 5 (money = 1).
Total amount you can rob = 2 + 9 + 1 = 12.

Solution :

Code : ASS_Code3.java

Output :

```
The maximum profit a thief can make : 4
```

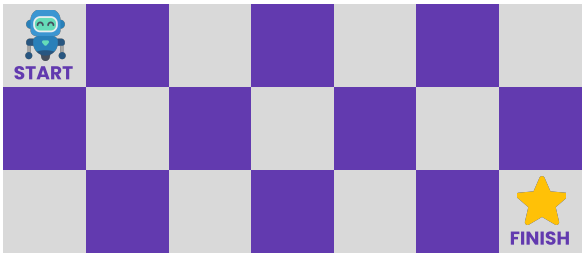
Approach :

- 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]).

Q4. 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.

Example 1:



Input: m = 3, n = 7
Output: 28

Example 2:
Input: m = 3, n = 2
Output: 3
Explanation: From the top-left corner, there are a total of 3 ways to reach the bottom-right corner:
1. Right -> Down -> Down
2. Down -> Down -> Right
3. Down -> Right -> Down

Solution :

Code : ASS_Code4.java

Output :

```
The desired output is : 35
```

Approach :

- For a path to be unique, at atleast 1 of move must differ at some cell within that path.
- At each cell we can either move down or move right.
- Choosing either of these moves could lead us to an unique path
- So we consider both of these moves.
- If the series of moves leads to a cell outside the grid's boundary, we can return 0 denoting no valid path was found.
- If the series of moves leads us to the target cell (m-1, n-1), we return 1 denoting we found a valid unique path from start to end.
- There are many cells which we reach multiple times and calculate the answer for it over and over again.
- However, the number of unique paths from a given cell (i,j) to the end cell is always fixed.
- So, we don't need to calculate and repeat the same process for a given cell multiple times. We can just store (or memoize) the result calculated for cell (i, j) and use that result in the future whenever required.
- Thus, here we use a 2d array dp, where dp[i][j] denote the number of unique paths from cell (i, j) to the end cell (m-1, n-1). Once we get an answer for cell (i, j), we store the result in dp[i][j] and reuse it instead of

recalculating it.

Q5. 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.

Example 1:
Input: triangle = [[2],[3,4],[6,5,7],[4,1,8,3]]
Output: 11
Explanation: The triangle looks like:
2
3 4
6 5 7
4 1 8 3
The minimum path sum from top to bottom is 2 + 3 + 5 + 1 = 11 (underlined above).

Example 2:
Input: triangle = [[-10]]
Output: -10

Solution :

Code : ASS_Code5.java

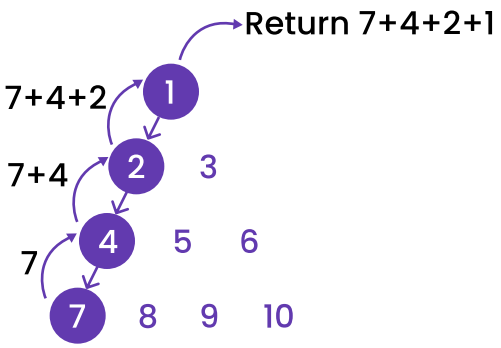
Output :

```
The desired output is : 6
```

Approach :

- We simply start from the root cell (i.e. the top of the triangle) and work our way down. The results of each individual cell will bubble all the way up to our root and we can simply return that answer.
- In this question, we start at the top level of the triangle and end at the bottom level. We'll do it the other way around for DP.
- In this question, each call represents a different level and at each level we look at different indices (cells) on that same level.
- This is basically just two for loops: one looping through each level, and another looping through each cell of that level.
- In this question, as aforementioned, we get the current cell value + the minimum path between the left and right cells on the next level.
- This is identical to how we do it in DP:
  - dp[level][i] = triangle.get(level).get(i) + Math.min(dp[level+1][i], dp[level+1][i+1]);

Top-down:



Bottom-up: