1. Unique Paths
2. Climbing Stairs
3. Coin Change
4. 가장긴 서브시퀀스 증가
   (Longest Increasing Subsequence)

문제1) Unique Path

Problem

A robot is located at the top-left corner of a *m* x *n* grid
(marked 'Start' in the diagram below).
The robot can only move either down or right at any point in time.
The robot is trying to reach the bottom-right corner of the grid
(marked 'Finish' in the diagram below).
How many possible unique paths are there? (7*3)

start						
						Finish

Note: *m* and *n* will be at most 100.
Example 1:
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 -> Right -> Down
2. Right -> Down -> Right
3. Down -> Right -> Right

Unique Path

Problem

1 (0,0)	**1 (0,1)**	**1 (0,2)**
1 (1,0)	2 (1,1)	3 (1,2)

Intput : m=3 , n=2
Output : 3

1. Right -> Right -> Down
2. Right -> Down -> Right
3. Down -> Right -> Right

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

Example

1	**1**	**1**
1	2	3
1	3	6
1	4	10

1_UniquePath

```
public class UniquePath {

	public int uniquePaths(int m, int n)  {
	}
}
```

문제 2) Climbing Stairs

Problem

You are climbing a stair case. It takes n steps to reach to the top.

Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

Note: Given n will be a positive integer.

Example 1:

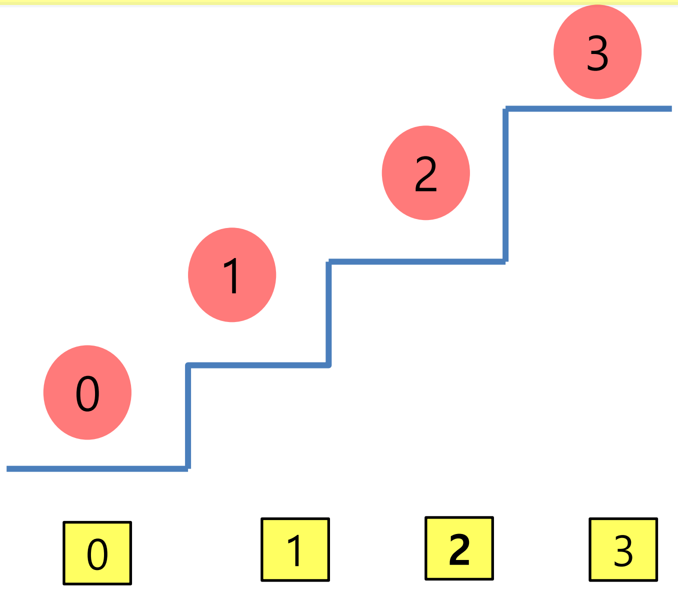
Input: 2
Output: 2
Explanation: There are two ways to climb to the top.
1. 1 step + 1 step
2. 2 steps

Climbing Stairs

Problem

Input: 3
Output: 3

1. 1 step + 1 step + 1 step
2. 1 step + 2 steps
3. 2 steps + 1 step



Solution

dp[3] = dp[2] + dp[1]

dp[i] = dp[i-1] + dp[i-2]

2_ClimbingStairs

```
public class ClimbingStairs {

    public int climbStairs(int n) {
    }
}
```

문제 3) Coin Change

Problem

You are given coins of different denominations and a total amount of money amount. Write a function to compute 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.

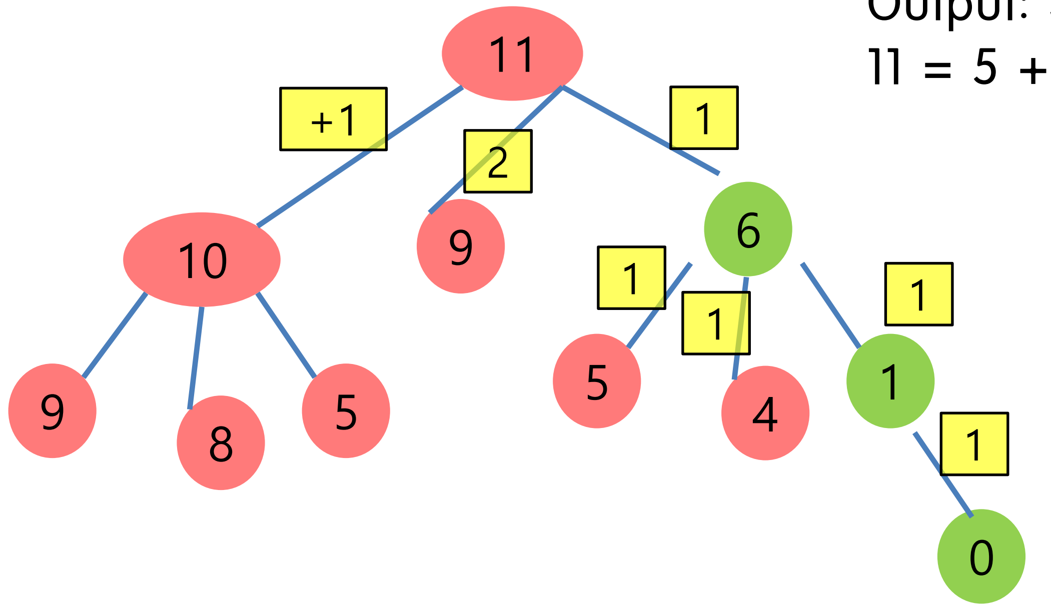
Example 1:

Input: coins = [1, 2, 5], amount = 11
Output: 3
Explanation: 11 = 5 + 5 + 1

Coin Change

Problem

Input: coins = [1, 2, 5], amount = 11
Output: 3
11 = 5 + 5 + 1



dp[0]=0 을 기준으로 +1

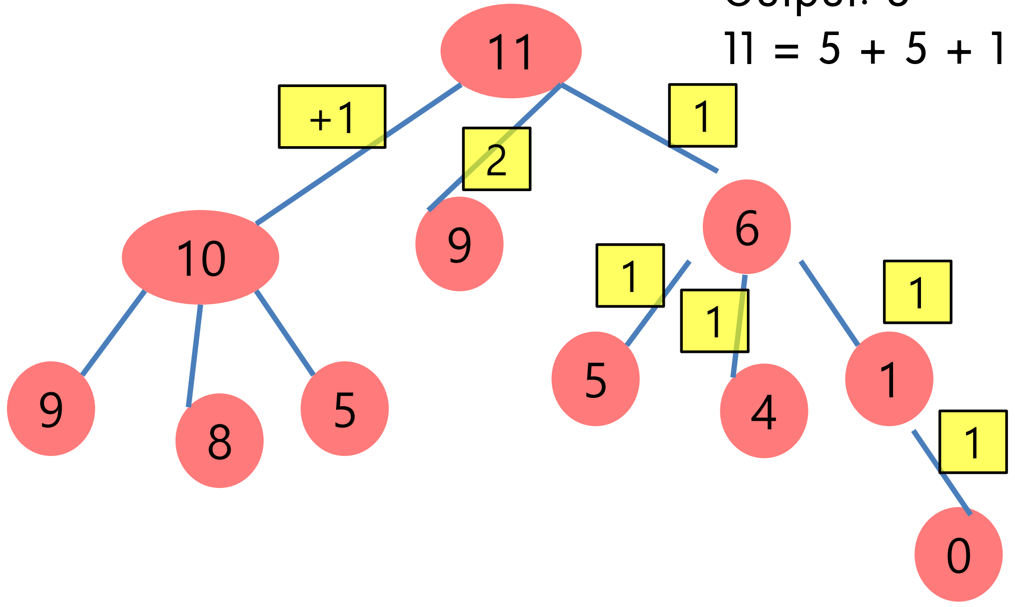
Solution

0	1	2	3	4	5	6	7	8	9	10	11
0	1 2	1 2	1 2	1 2	1 2	1 2	1 2	1 2	1 2	1 2	1 2
0	1	1	2	2	1	2	2	3	3	2	3

Coin Change

Problem

Input: coins = [1, 2, 5], amount = 11
Output: 3
11 = 5 + 5 + 1



dp[i] = Math.min(dp[i], dp[i - coins[j]] + 1);

dp[1] = Math.min(dp[1], dp[1 - coins[j]] + 1)

1-1 =dp[0] =0
1-2 (X)
1-5 (X)

dp[5] = Math.min(dp[5], dp[5 - coins[j]] + 1

5-1 =dp[4] =2
5-2=dp[3] =2
5-5=dp[0] =0

Solution

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
0	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**
0	**1**	**1**	**2**	**2**	**1**	**2**	**2**	**3**	**3**	**2**	**3**

3_CoinChange

```
public class CoinChange {

    public int coinChange(int[] coins, int amount) {
    }
}
```

가장긴 서브시퀀스 증가 (Longest Increasing Subsequence)

설명

정수 배열 nums가 주어집니다.
정수가 증가하는 가장 긴 sub sequence의 길이를 반환합니다.

정수의 배열에서 연속적으로 증가하는 부분만 선택하면 됩니다.
예를 들어 [3,6,2,7]는 [0,3,1,6,2,2,7] 배열의 하위 시퀀스입니다

입출력

Input: nums = [1,2,3,4,5,2,6,10,4,12]
Output: 8

Input: nums = [0,1,0,3,2,3]
Output: 4

Input: nums = [7,7,7,7,7,7,7]
Output: 1

제한사항

$1 <= \text{nums.length} <= 2500$
$-10^4 <= \text{nums[i]} <= 10^4$

4_Longest Increasing Subsequence

```
public class Lis{

    public int bottomUp(int[] nums) {     }
}
```