

322. Coin Change

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.

Example 1:

Input: `coins = [1,2,5]`, `amount = 11`

Output: 3

Explanation: $11 = 5 + 5 + 1$

Example 2:

Input: `coins = [2]`, `amount = 3`

Output: -1

Example 3:

Input: `coins = [1]`, `amount = 0`

Output: 0

Constraints:

- $1 \leq n \leq 12$
- $1 \leq \text{coins}_i \leq 2^{31} - 1$
- $0 \leq \text{amount} \leq 10^4$

Solution

$$\begin{aligned}
 f(0) &= 0, \\
 f(a) &= \min_{c \in \text{coins}} (1 + f(a - c)) \text{ for } a > 0, \\
 f(a) &= \infty \text{ for } a < 0
 \end{aligned}$$

```

import math
class Solution:
    def rec(self, curr_amount, coins):
        if curr_amount < 0:
            return math.inf
        if curr_amount == 0:
            return 0
        return min([self.rec(curr_amount - coin, coins) + 1 for coin in coins])
    def coinChange(self, coins: List[int], amount: int) → int:
        ans = self.rec(curr_amount = amount, coins = coins)
        return ans if ans != math.inf else -1

```

$$f(a) = \begin{cases} M[a] & a \in M \\ 0 & a = 0 \\ \infty & a < 0 \\ \min_{c \in \text{coins}} (1 + f(a - c)) & \text{otherwise (store in } M[a]) \end{cases}$$

```

import math
class Solution:
    def mem(self, curr_amount, coins, d):
        if curr_amount < 0:
            return math.inf
        if curr_amount == 0:
            return 0
        if curr_amount not in d:
            d[curr_amount] = min([self.mem(curr_amount - coin, coins, d) + 1 for coin in coins])
        return d[curr_amount]
    def coinChange(self, coins: List[int], amount: int) → int:

```

```

d = {}
ans = self.mem(curr_amount = amount, coins = coins, d = d)
return ans if ans != math.inf else -1

```

$$dp[i] = \min_{c \in \text{coins}, c \leq i} (dp[i - c] + 1), \quad \forall i \in [1, \text{amount}]$$

```

import math
class Solution:
    def tab(self, coins, amount):
        dp = [math.inf for _ in range(amount + 1)]
        dp[0] = 0
        for i in range(1, len(dp)):
            dp[i] = 1 + min([dp[i - coin] for coin in coins if i - coin >= 0],
                           default=math.inf)
        return dp[amount]
    def coinChange(self, coins: List[int], amount: int) → int:
        ans = self.tab(coins = coins, amount = amount)
        return ans if ans != math.inf else -1

```

Overnight

Your task is to calculate how many ways you can get the sum n by rolling the dice. Each roll of the dice produces a result between 1 and 6.

For example, if $n = 3$, the options are:

- $1 + 1 + 1$
- $1 + 2$
- $2 + 1$
- 3

Input

The input is a number, n : target amount.

Printout

Your program should print one integer: how many ways can you get the sum.

Bounds

- $1 \leq n \leq 50$

Example

Input:

3

Printout:

4

Solution

$$f(n) = \begin{cases} 1 & n = 0 \\ \sum_{n-d \geq 0} f(n-d) & n > 0 \end{cases}$$

```
n = int(input())
dice = [1,2,3,4,5,6]
def rec(n, dice):
    if n == 0:
        return 1
    return sum([rec(n - d, dice) for d in dice if n - d >= 0])
print(rec(n, dice))
```

$$f(n) = \begin{cases} 1 & n = 0 \\ M[n] & n \in M \\ \sum_{n-d \geq 0} f(n-d) & n \notin M \end{cases}$$

```
n = int(input())
dice = [1,2,3,4,5,6]
def mem(n, dice, M):
    if n == 0:
        return 1
    if n not in M:
        M[n] = sum([mem(n - d, dice, M) for d in dice if n - d >= 0])
    return M[n]
M = {}
print(mem(n, dice, M))
```

$$dp[n] = \begin{cases} 1 & n = 0 \\ \sum_{n-d \geq 0} dp[n-d] & n > 0 \end{cases}$$

```
n = int(input())
dice = [1,2,3,4,5,6]
def tab(n, dice):
    dp = [0 for _ in range(n + 1)]
    dp[0] = 1
    for i in range(1, len(dp)):
        dp[i] = sum([dp[i - d] for d in dice if i-d>=0])
    return dp[n]
print(tab(n, dice))
```

Reset

You have been given a number, n and your task is to get 0 from it. You get to subtract one of the digits from the number at each step.

What is the minimum number of steps required to successfully reset?

Input

The input is a number, n .

Printout

Your program should print the smallest number of steps to reset the number to zero.

Bounds

- $1 \leq n \leq 10^6$

Example

Input:

27

Printout:

5

Explanation: The reset is done like this: $27 \rightarrow 20 \rightarrow 18 \rightarrow 10 \rightarrow 9 \rightarrow 0$

Solution

Given that $D(n)$ is set of digits of n

$$f(n) = \begin{cases} 1 & n < 10 \\ 1 + \min_{d \in D(n), d \neq 0} f(n - d) & n \geq 10 \end{cases}$$

```
import math
n = int(input())
def rec(n):
    if n < 10:
        return 1
    N = n
    digits = []
    while N:
        digits.append(N%10)
        N = N // 10
    return 1 + min([rec(n - digit) for digit in digits if digit != 0],
        default = math.inf)
print(rec(n))
```

$$f(n) = \begin{cases} 1 & n < 10 \\ M[n] & n \in M \\ 1 + \min_{d \in D(n), d \neq 0} f(n - d) & n \notin M \end{cases}$$

```
import math
n = int(input())
def mem(n, M):
    if n < 10:
        return 1
    if n not in M:
        N = n
        digits = []
        while N:
```



```

        digits.append(N%10)
        N = N // 10
    M[n] = 1 + min([mem(n - digit, M) for digit in digits if digit != 0],
        default = math.inf)
    return M[n]
M = {}
print(mem(n, M))

```

$$dp[n] = \begin{cases} 1 & 0 \leq n \leq 9 \\ 1 + \min_{d \in D(n), d \neq 0} dp[n - d] & n \geq 10 \end{cases}$$

```

import math
n = int(input())
def tab(n):
    dp = [0 for _ in range(n + 1)]
    for d in range(min(n, 9) + 1):
        dp[d] = 1
    for i in range(10, len(dp)):
        N = i
        digits = []
        while N:
            digits.append(N%10)
            N = N // 10
        dp[i] = 1 + min([dp[i - digit] for digit in digits if digit > 0],
            default = math.inf)
    return dp[n]
print(tab(n))

```