Dynamic programming

1、问题目标

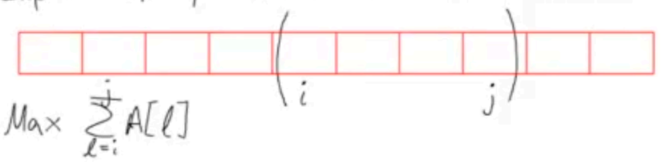
2、状态的定义： opt[n]

3、状态转移方程：opt[n] = best_of(opt[n-1], opt[n-2], ...)

1. 最大子序和

目标：

Input: Array A[1..n] of real numbers

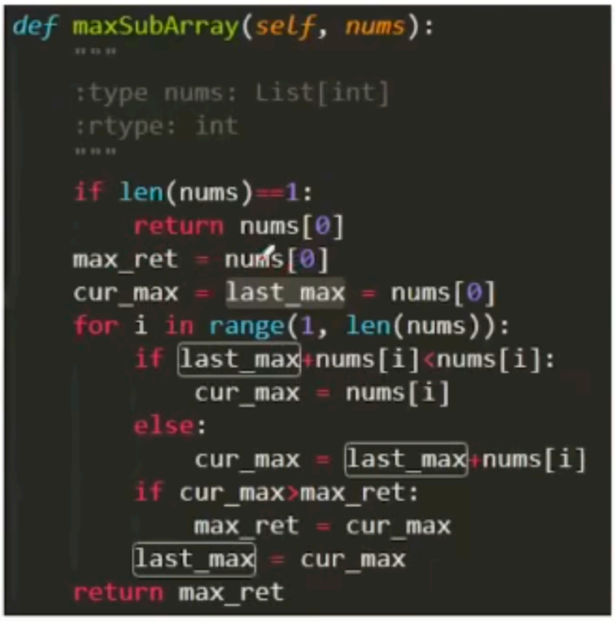


$\text{Max} \sum_{l=i}^{j} A[l]$

子问题：

M(j) : max sum over all windows ending at j.

$M(j) = \max\{M(j-1) + A[j],\ A[j]\}$



```
def maxSubArray(self, nums):
    """
    :type nums: List[int]
    :rtype: int
    """
    if len(nums)==1:
        return nums[0]
    max_ret = nums[0]
    cur_max = last_max = nums[0]
    for i in range(1, len(nums)):
        if last_max+nums[i]<nums[i]:
            cur_max = nums[i]
        else:
            cur_max = last_max+nums[i]
        if cur_max>max_ret:
            max_ret = cur_max
        last_max = cur_max
    return max_ret
```

2. 最长上升子序列

目标:

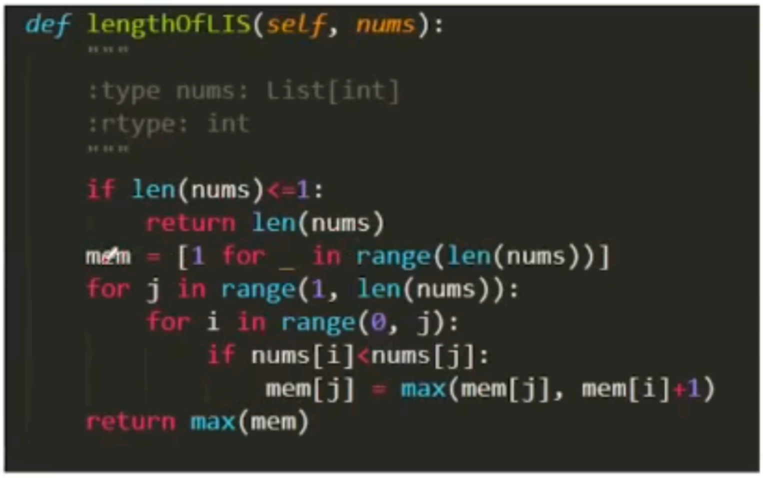
Input: Sequence $A_1 \ldots A_n$

Goal: find a longest strictly increasing subsequence (not necessarily contiguous).

子问题:

$L(j)$: longest strictly increasing subsequence ending at position j.

$$L(j) = \max_{\substack{i<j \\ A[i]<A[j]}} \{L(i)\} + 1$$



```python
def lengthOfLIS(self, nums):
    """
    :type nums: List[int]
    :rtype: int
    """
    if len(nums)<=1:
        return len(nums)
    mem = [1 for _ in range(len(nums))]
    for j in range(1, len(nums)):
        for i in range(0, j):
            if nums[i]<nums[j]:
                mem[j] = max(mem[j], mem[i]+1)
    return max(mem)
```

3. 零钱兑换

目标:

Input: n denominations of coins of values $1 = v_1 < v_2 < v_3 < \cdots < v_n$

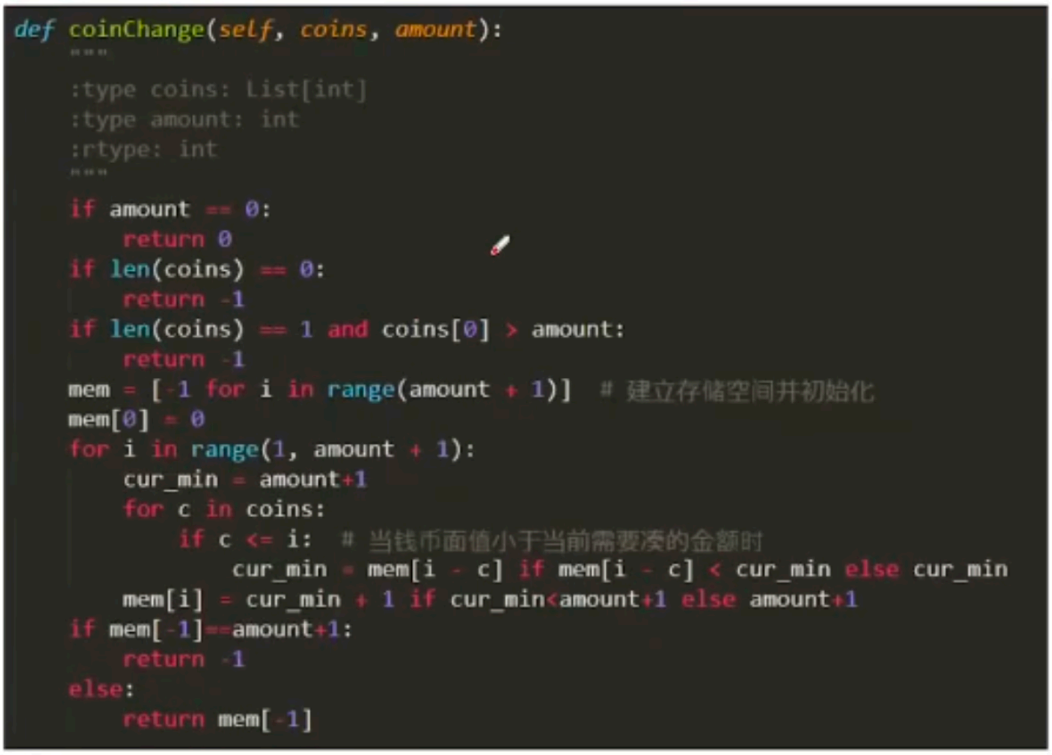
Goal: make change for amount of money C. Use as few coins as possible.

子问题:

$M(j)$: minimum # coins required to make change for amount of money j.

$$M(j) = \min_i \{ M(j - v_i) \} + 1$$

有一些零钱，给定一个数值。用已有的硬币，去组成这个数值。用最少数量的硬币。

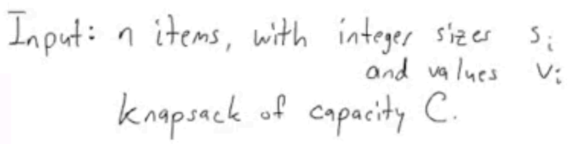


```python
def coinChange(self, coins, amount):
    """
    :type coins: List[int]
    :type amount: int
    :rtype: int
    """
    if amount == 0:
        return 0
    if len(coins) == 0:
        return -1
    if len(coins) == 1 and coins[0] > amount:
        return -1
    mem = [-1 for i in range(amount + 1)]  # 建立存储空间并初始化
    mem[0] = 0
    for i in range(1, amount + 1):
        cur_min = amount+1
        for c in coins:
            if c <= i:  # 当钱币面值小于当前需要凑的金额时
                cur_min = mem[i - c] if mem[i - c] < cur_min else cur_min
        mem[i] = cur_min + 1 if cur_min<amount+1 else amount+1
    if mem[-1]==amount+1:
        return -1
    else:
        return mem[-1]
```

建立一个 memory，一直完善这个 memory。

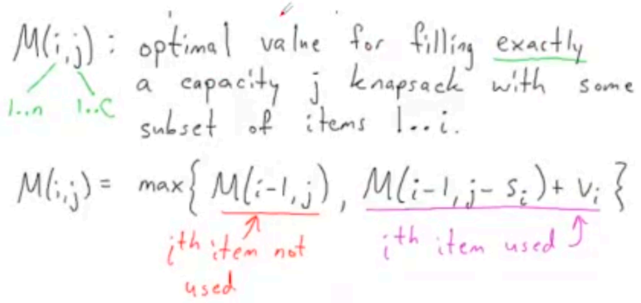
4. 背包问题

目标：



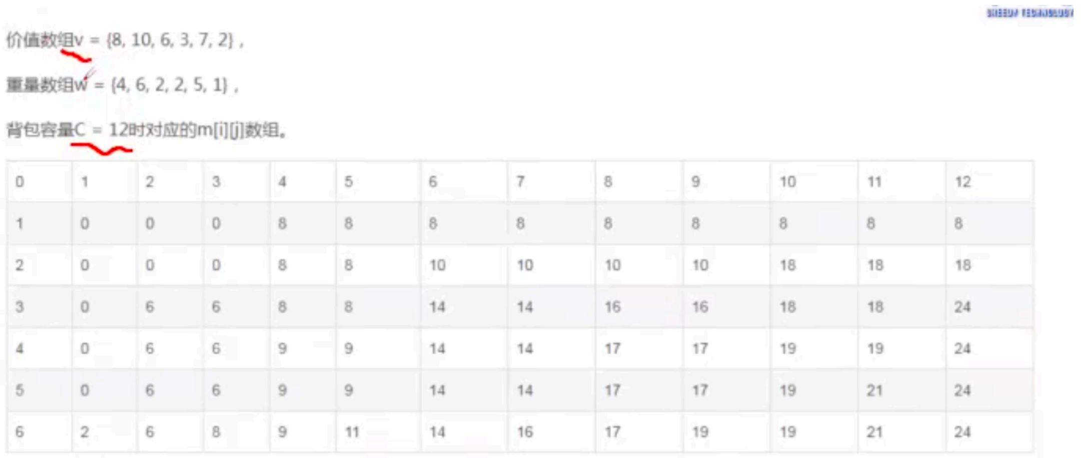
Input: n items, with integer sizes s_i
and values v_i
knapsack of capacity C.

子问题：



$\underset{1..n}{M(i},\underset{1..C}{j)}$: optimal value for filling exactly a capacity j knapsack with some subset of items 1..i.

$$M(i,j) = \max\{\underbrace{M(i-1,j)}_{i\text{th item not used}}, \underbrace{M(i-1,j-s_i)+v_i}_{i\text{th item used}}\}$$



价值数组v = {8, 10, 6, 3, 7, 2}，

重量数组w = {4, 6, 2, 2, 5, 1}，

背包容量C = 12时对应的m[i][j]数组。

0	1	2	3	4	5	6	7	8	9	10	11	12
1	0	0	0	8	8	8	8	8	8	8	8	8
2	0	0	0	8	8	10	10	10	10	18	18	18
3	0	6	6	8	8	14	14	16	16	18	18	24
4	0	6	6	9	9	14	14	17	17	19	19	24
5	0	6	6	9	9	14	14	17	17	19	21	24
6	2	6	8	9	11	14	16	17	19	19	21	24