为了做这道题，花了许多时间，重新回忆起动态规划的内容。动态规划的方法基本是学一遍忘一遍，为了下次再用到时可以快速回忆起来，在这里做一个笔记。

**动态规划基本思想**

动态规划是一种方法论，不是具体算法，不存在明确的数学表达式。它的核心思想是，把一个问题划分为多个互相联系的子问题，我们可以**用一个表记录下每个子问题的答案**，避免重复求解。

动态规划要求问题满足：**子问题重叠**、**[最优子结构](https://zh.wikipedia.org/w/index.php?title=%E6%9C%80%E4%BC%98%E5%AD%90%E7%BB%93%E6%9E%84&action=edit&redlink=1)**、**无后效性**。具体是什么含义还是到具体问题里体会吧。

**解题流程（适用于0-1背包类问题）**

1. 定义动态规划表DP，明确行和列的含义。为了优化空间复杂度，这里采用一维的滚动数组来表示表DP，用0填充。
2. 逐行扫描，基于上一行的信息来更新当前行。一般来说，扫描方向从左往右还是从右到左都可以。这里会有一个转移方程，需要根据具体问题来定义。
3. 从表中提取出最终结果。如果需要更具体的答案（例如背包问题中，选择哪些物品）可能需要进行回溯。

**0-1背包问题**

有一个容量为C的背包和N件物品。物品的价值P分别为1、3、2，体积V分别为2、5、1。选择放入背包的物品使价值最大。

我们定义一个动态规划表dp[N+1][C+1]，或者动态数组dp[C+1]

for i=1:N # i表示物品索引

odp = copy(dp) # 这里的拷贝不用也可以，但是为了结构清晰还是加上好

for j=0:C # j表示背包容量

if j < V[i]

continue

dp[j] = max(odp[j - V[i]] + P[i], odp[j])

result = max(dp)

**Energy Stones问题（小测试集）**

这个问题比较困难，首先，我们观察到：

1. 对于能量为0的石头，没有必要吃
2. 连续吃下能量非0的石头，则这些石头的L必然是降序

简化成0-1背包问题，子问题为：

对于前i个石头和时间范围j，能获得的最大能量dp[i][j]。

有dp[i+1][j] = max(dp[i][j], dp[i][j-1]+current\_e)

**其它非0-1背包类的动态规划问题**

**Maximum Subarray问题**

Given an integer array nums, find the contiguous subarray (containing at least one number) which has the largest sum and return its sum.

Example:

Input: [-2,1,-3,4,-1,2,1,-5,4], Output: 6

Explanation: [4,-1,2,1] has the largest sum = 6.

实际上这个问题关于动态规划的部分仅仅是求子序列和的时候，简化了运算。

穷举法 O(n^3)

answer = -inf

for i = 1:N

for j = i:N

sum = 0

for v in list[i:j]

sum += v

If sum > answer:

answer = sum

改进法O(n^2)

answer = -inf

for i = 1:N

sum = 0

for j = i:N

sum += list[j]

if sum > answer:

answer = sum

特殊性质 O(n)

负数的子序列不应该被包含

answer = -inf

sum = 0

for i = 1:N

sum += list[i]

if sum > answer:

answer = sum

elif sum < 0:

sum = 0

**爬楼梯问题**

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.

递归法

f(n) = f(n-1) + f(n-2)

f(1) = 1

f(2) = 2

时间复杂度似乎是O(2^n)，可以观察到实际上有很多重复运算，比方说f(6-1) = f(7-2)。

改进递归法

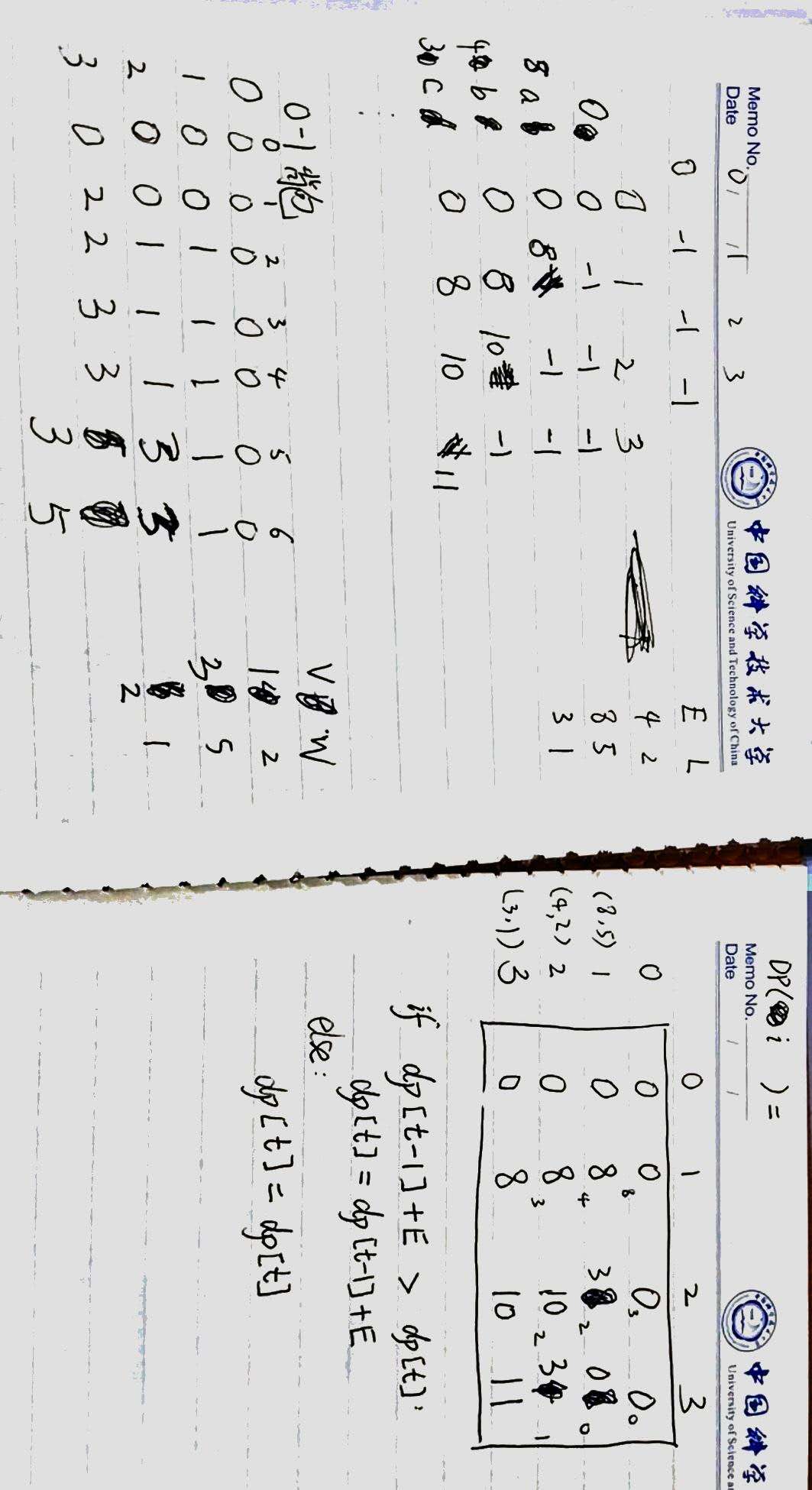
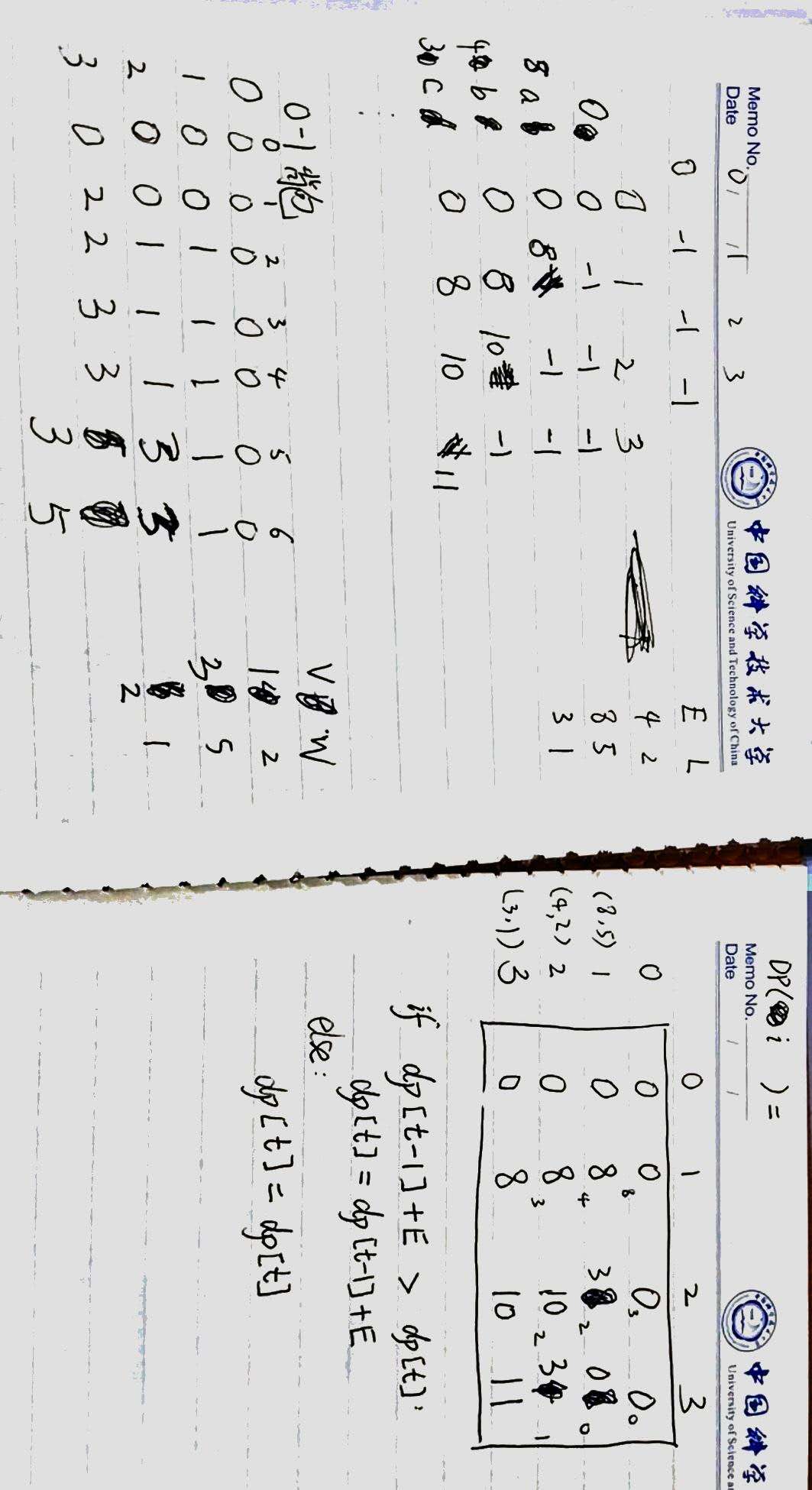
f(1) = 1

f(2) = 2

f(3) = f(1) + f(2)

f(4) = f(3) + f(2)

时间复杂度是O(n)，空间复杂度是O(n) (不保存前面的值的话就是O(1)了)

****