10.1 - Subset Sum

子集和问题（Subset Sum Problem）

给定一个包含 n 个物品的集合 $\{1, 2, ..., n\}$，每个物品 i 有一个非负权重 w_i；给定一个界限 W

我们的目标是找到一个子集 S 使得 $\sum_{i \in S} w_i \leq W$，并且 $\sum_{i \in S} w_i$ 最大化

贪心算法

- 按递减顺序排列：从最大的权重开始，逐一放入子集，直到无法再放入更多物品。这种方法适用于某些情况下的最大化问题
- 按递增顺序排列：从最小的权重开始，逐一放入子集，直到达到或超过界限 W。这种方法有助于确保可以尽可能多地选择物品

这两个算法当然都不是最优的

动态规划

我们需要确定适当的子问题，以解决主要问题

- 设 $\mathrm{OPT}(i)$ 是子集和问题的最优解，使用集合 $\{1, 2, ..., i\}$
  - 设 O_i 是子问题最优解的值，它是最优物品集合的权重和，因此 O 是 O_n
- 项目 n 是否应该在最优解 O 中
  - 如果不是，那么 $\mathrm{OPT}(n-1) = \mathrm{OPT}(n)$
  - 如果是，那么?

$n \in O$

在加权区间调度问题中，我们可以移除所有与 n 重叠的区间
我们能在这里做类似的事情吗?

- 没有理由预先排除任何剩余的物品，除非添加它会超过重量
- 我们真正得到的唯一信息是我们现在剩下的重量：$W - w_n$

要找到 $\mathrm{OPT}(n)$ 的最优值，我们需要：

- 如果 n 不在 O 中，$\mathrm{OPT}(n-1)$ 的最优值
- 输入集合 $\{1, 2, ..., n-1\}$ 与 $w = W - w_n$ 的最优值

我们需要多少个子问题?

- 每个初始集合 $\{1, 2, ..., n-1\}$ 和剩余重量 w 的每个可能值的一个子问题

子问题

我们将为每个 $i = 0, 1, ..., n$ 和每个整数 $0 \leq w \leq W$ 定义一个子问题：
设 $\mathrm{OPT}(i, w)$ 为在允许的最大重量 w 下，在子集 $\{1, 2, ..., i\}$ 上的最优解的值
我们要寻找：$\mathrm{OPT}(n, W)$

对于第 n 个物品是否应该在 O 中：

- 如果不在，则 $\mathrm{OPT}(n, W) = \mathrm{OPT}(n-1, W)$
- 如果在，则 $\mathrm{OPT}(n, W) = w_n + \mathrm{OPT}(n-1, W-w_n)$，对于 $W \geq w_n$

则 $\mathrm{OPT}(j, w) = \max\left(w_j + \mathrm{OPT}(j-1, w-w_j), \mathrm{OPT}(j-1, w)\right)$

算法

```
SubsetSum(n, W)
Array M = [[0 ... n], [0 ... W]]
Init M[0, w] = 0, for each w = 0, 1, ..., W

for i = 1, 2, ..., n
    for w = 0, ..., W
        if (w_i > w)
            M[i, w] = M[i-1, w]
        else
            M[i, w] = max(M[i-1, w], w_i + M[i-1, w-w_i])
return M[n, W]
```

1. 创建一个二维数组 M，其中 $M[i][w]$ 表示在前 i 个物品中选择，且总重量不超过 w 时的最优值
2. 初始化 $M[0][w] = 0$ 表示当没有物品可选时，无论重量限制多少，最优解临值都为 0
3. 对于每个物品 i 从 1 到 n：
    1. 如果物品的重量 $w_i > w$：
        1. 不选择这个物品，$M[i][w] = M[i-1][w]$
    2. 如果物品的重量 $w_i \leq w$：
        1. 不选择当前物品时的最优值，即 $M[i][w] = M[i-1][w]$（递归调用算法计算）
        2. 选择当前物品时的最优值，即当前物品的重量加上剩余重量 $w - w_i$ 下的最优值 $w_i + M[i-1][w-w_i]$（递归调用算法计算）
        3. 取 1, 2 中较大的值
4. 返回 $M[n][W]$

- n=3, W=6, $w_1 = w_2 = 2$ and $w_3 = 3$.

Optimal value

3	0	0	2	3	4	5	5
2	0	0	2	2	4	4	4
1	0	0	2	2	2	2	2
0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6

非常类似于加权区间调度问题，这是一个从值到解决方案的过程

运行时间

类似于加权区间调度问题，我们构建的是一个解决方案表 M，而不是数组

计算 M 中的每个值 $M(i, w)$ 都利用了之前的值，这需要 $O(1)$ 的时间

因此 `SubsetSum(n, W)` 算法的运行时间是 $O(nW)$，但是：

- 这不是一个多项式时间算法：它依赖于 W，因此是一个伪多项式算法
- 如果输入的数字相对较小，它十分的有效

子集问题是 NP-Hard 的，设计一个子集问题的多项式算法难如登天

伪多项式时间：如果算法的运行时间是输入值的大小的多项式（而不是输入的位数），则称该算法为伪多项式时间
伪多项式时间算法在输入值较小的情况下是有效的，但当输入值很大时，运行时间可能变得不可接受