0/1背包问题

题目描述

现有n个物品和一个背包。

已知：物品i的重量是W_i，价值是V_i，背包最大承重为W_{max}。

为简化起见，假设：

1、每个物品只有一件且不可分割，即对任意一个物品i，背包中或存在0个，或存在1个；

2、除承重约束外，无需考虑体积等其他任何约束。

目标：求满足约束的条件下，背包可装下物品的最大价值和。

输入格式

一行，每个数以空格隔开，行尾无空格或其他字符。

第1个数为物品个数n

第2个数为背包承重W_{max}

第3至第$n+2$个数为物品的重量W_i

第$n+3$至第$2n+2$个数为物品的价值V_i

输出格式

一个数字，代表满足约束的条件下，背包可承载的最大价值

样例 #1

样例输入 #1

```
6 21 1 2 3 4 5 6 6 5 4 3 2 1
```

样例输出 #1

```
21
```

提示

$1 \leq n \leq 100$

$0 \leq W_{max} \leq 2147483647$

$0 \leq W_i \leq 100$

$0 \leq V_i \leq 100$

以上均为整数

迭代法

```
from collections import namedtuple


class Solution(namedtuple("solution", "i_max w_max w v")):
    answers = {}
    __hash__ = staticmethod(lambda: None)

    def maximize(self, i, w_max):
        ans = self.answers.get((i, w_max), None)
        if ans is not None:
            return ans

        if i == self.i_max:
            ans = self.v[i] if w_max >= self.w[i] else 0
        else:
```

```
            ans = max(
                self.maximize(i + 1, w_max),
                self.maximize(i + 1, w_max - self.w[i]) + self.v[i]
            ) if w_max >= self.w[i] else self.maximize(i + 1, w_max)

        self.answers[i, w_max] = ans
        return ans

    @property
    def answer(self):
        return self.maximize(0, self.w_max)


if __name__ == '__main__':
    values = map(int, input().split())
    n = next(values)
    print(Solution(n - 1, next(values),
```

```
                    [next(values) for _ in range(n)],
                    [next(values) for _ in range(n)]).answer)
```

递归法

```
from functools import cache


class Solution:
    def __init__(self, i_max, w_max, w, v):
        self.i_max = i_max
        self.w_max = w_max
        self.w = w
        self.v = v

    @cache
```

```
    def maximize(self, i, w_max):
        w, v = self.w, self.v
        if i == self.i_max:
            return v[i] if w_max >= w[i] else 0
        else:
            return max(
                self.maximize(i + 1, w_max),
                self.maximize(i + 1, w_max - w[i]) + v[i]
            ) if w_max >= w[i] else self.maximize(i + 1, w_max)

    @property
    def answer(self):
        return self.maximize(0, self.w_max)


if __name__ == '__main__':
    values = map(int, input().split())
```

```
    n = next(values)
    print(Solution(n - 1, next(values),
                   [next(values) for _ in range(n)],
                   [next(values) for _ in range(n)]).answer)
```