题目描述

某云短信厂商，为庆祝国庆，推出充值优惠活动。
现在给出客户预算，和优惠售价序列，求最多可获得的短信总条数。

输入描述

第一行客户预算M，其中 0 ≤ M ≤ 10^6
第二行给出售价表， P1, P2, ... Pn , 其中 1 ≤ n ≤ 100 ,
Pi为充值 i 元获得的短信条数。1 ≤ Pi ≤ 1000 , 1 ≤ n ≤ 100

输出描述

最多获得的短信条数

用例

输入	6 10 20 30 40 60
输出	70
说明	分两次充值最优， 1 元、 5 元各充一次。总条数 10 + 60 = 70

输入	15 10 20 30 40 60 60 70 80 90 150
输出	210
说明	分两次充值最优， 10 元 5 元各充一次，总条数 150 + 60 = 210

题目解析

本题是 完全背包问题 。

如果大家不是很清楚 完全背包 的求解，可以看算法设计 - 01背包问题的状态转移方程优化，以及完全背包问题_伏城之外的博客-CSDN博客

本题中：

- 客户预算M相当于背包的承重，
- 出售价表：

1. i 元相当于物品的重量，
2. Pi 短信条数相当于物品的价值

JavaScript算法源码

```
/* JavaScript Node ACM模式 控制台输入获取 */
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

const lines = [];
rl.on("line", (line) => {
  lines.push(line);

  if (lines.length === 2) {
    const m = lines[0] - 0;
    const p = lines[1].split(" ").map(Number);
    console.log(getResult(m, p));
    lines.length = 0;
  }
```

题目解析

本题是完全背包问题。

如果大家不是很清楚完全背包的求解，可以看算法设计 - 01背包问题的状态转移方程优化，以及完全背包问题_伏城之外的博客-CSDN博客

本题中：

- 客户预算M相当于背包的承重，
- 出售价表：

1. i 元相当于物品的重量，
2. Pi 短信条数相当于物品的价值

JavaScript算法源码

```javascript
/* JavaScript Node ACM模式 控制台输入获取 */
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

const lines = [];
rl.on("line", (line) => {
  lines.push(line);

  if (lines.length === 2) {
    const m = lines[0] - 0;
    const p = lines[1].split(" ").map(Number);
    console.log(getResult(m, p));
    lines.length = 0;
  }
});

function getResult(m, p) {
  const dp = new Array(m + 1).fill(0);

  for (let i = 0; i <= p.length; i++) {
    for (let j = 0; j <= m; j++) {
      if (i == 0 || j == 0 || j < i) continue;
      dp[j] = Math.max(dp[j], dp[j - i] + p[i - 1]);
    }
  }

  return dp[m];
}
```

Java算法源码

```java
import java.util.Arrays;
import java.util.Scanner;

public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    int m = Integer.parseInt(sc.nextLine());
    Integer[] p =
        Arrays.stream(sc.nextLine().split(" ")).map(Integer::parseInt).toArray(Integer[]::new);

    System.out.println(getResult(m, p));
  }

  public static int getResult(int m, Integer[] p) {
    int[] dp = new int[m + 1];

    for (int i = 0; i <= p.length; i++) {
      for (int j = 0; j <= m; j++) {
        if (i == 0 || j == 0 || j < i) continue;
        dp[j] = Math.max(dp[j], dp[j - i] + p[i - 1]);
      }
    }

    return dp[m];
  }
}
```

Python算法源码

```python
# 输入获取
m = int(input())
p = list(map(int, input().split()))


# 算法入口
def getResult(m, p):
```

```javascript
    const p = lines[1].split(" ").map(Number);
    console.log(getResult(m, p));
    lines.length = 0;
  }
});

function getResult(m, p) {
  const dp = new Array(m + 1).fill(0);

  for (let i = 0; i <= p.length; i++) {
    for (let j = 0; j <= m; j++) {
      if (i == 0 || j == 0 || j < i) continue;
      dp[j] = Math.max(dp[j], dp[j - i] + p[i - 1]);
    }
  }

  return dp[m];
}
```

Java算法源码

```java
import java.util.Arrays;
import java.util.Scanner;

public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    int m = Integer.parseInt(sc.nextLine());
    Integer[] p =
        Arrays.stream(sc.nextLine().split(" ")).map(Integer::parseInt).toArray(Integer[]::new);

    System.out.println(getResult(m, p));
  }

  public static int getResult(int m, Integer[] p) {
    int[] dp = new int[m + 1];

    for (int i = 0; i <= p.length; i++) {
      for (int j = 0; j <= m; j++) {
        if (i == 0 || j == 0 || j < i) continue;
        dp[j] = Math.max(dp[j], dp[j - i] + p[i - 1]);
      }
    }

    return dp[m];
  }
}
```

Python算法源码

```python
# 输入获取
m = int(input())
p = list(map(int, input().split()))


# 算法入口
def getResult(m, p):
    dp = [0 for i in range(m + 1)]

    for i in range(len(p) + 1):
        for j in range(m + 1):
            if i == 0 or j == 0 or j < i:
                continue

            dp[j] = max(dp[j], dp[j - i] + p[i - 1])

    return dp[m]


# 算法调用
print(getResult(m, p))
```