

# 【DP】-云短信平台优惠活动

## 题目描述与示例

### 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  $0 \leq M \leq 1000000$

第二行给出售价表， $P_1, P_2, \dots, P_n$ ，其中  $P_i$  为充值  $i$  元获得的短信条数。

$1 \leq P_i \leq 1000$ ， $1 \leq n \leq 100$

### 输出描述

最多获得的短信条数

### 示例一

#### 输入

```
1 6
2 10 20 30 40 60
```

#### 输出

```
1 70
```

### 说明

分两次充值最优，1 元、5 元各充一次。总条数  $10 + 60 = 70$

# 示例二

## 输入

```
1 15
2 10 20 30 40 60 60 70 80 90 150
```

## 输出

```
1 210
```

## 说明

分两次充值最优，10 元、5 元各充一次。总条数  $150 + 60 = 210$

## 解题思路

本题的每一种短信套餐都可以购买无数次，即物品的选取次数不做限制。另外只需要看最终套餐的购买结果，和中间过程无关。

因此，本题属于**路径无关、求和最大值的完全背包问题**。直接套模板即可。

## 代码

### Python

#### 1维dp哈希表

```
1 # 题目：2024E-云短信平台优惠活动
2 # 分值：200
3 # 作者：许老师-闭着眼睛学数理化
4 # 算法：背包DP/1维dp哈希表
5 # 代码看不懂的地方，请直接在群上提问
6
7
8 from collections import defaultdict
9
```

```

10 # 客户预算
11 target = int(input())
12 # 充值获得的短信条数
13 nums = list(map(int, input().split()))
14 # 把充值0元可以获得0条短信也视为一种情况
15 nums = [0] + nums
16 # 构建1维dp哈希表, dp[i]表示充值总钱数为i元时, 能获得的最大短信数
17 dp = defaultdict(int)
18 # 初始化dp哈希表, dp[0] = 0表示使用0元可以获得0条短信
19 dp[0] = 0
20
21 # 路径无关的完全背包
22 # 先遍历物品nums
23 # i为下标, 也为充值的钱数i; num为充值钱数i能获得的短信条数
24 for i, num in enumerate(nums[1:], 1):
25     # 再顺序遍历背包
26     for pre_sum in range(target+1):
27         # 考虑前一个可以获得的总金额为pre_sum
28         # 加上当前充值的钱数i, 可以得到当前总金额cur_sum
29         cur_sum = pre_sum + i
30         # 当前金额必须不超过预算
31         if cur_sum <= target:
32             # 更新新的哈希表中的值
33             dp[cur_sum] = max(dp[cur_sum], dp[pre_sum] + num)
34
35 # 输出dp哈希表中的最大值即为答案
36 print(max(dp.values()))

```

## 1维dp数组

```

1 # 题目: 2024E-云短信平台优惠活动
2 # 分值: 200
3 # 作者: 许老师-闭着眼睛学数理化
4 # 算法: 背包DP/1维dp数组
5 # 代码看不懂的地方, 请直接在群上提问
6
7
8 # 客户预算
9 target = int(input())
10 # 充值获得的短信条数
11 nums = list(map(int, input().split()))
12 # 把充值0元可以获得0条短信也视为一种情况
13 nums = [0] + nums
14 # 构建1维dp数组, dp[i]表示充值总钱数为i元时, 能获得的最大短信数
15 dp = [0] * (target + 1)

```

```

16
17 # 路径无关的完全背包
18 # 先遍历物品nums
19 # i为下标，也为充值的钱数i；num为充值钱数i能获得的短信条数
20 for i, num in enumerate(nums[1:], 1):
21     # 再顺序遍历背包
22     for pre_sum in range(0, target-i+1):
23         # 考虑前一个可以获得的总金额为pre_sum
24         # 加上当前充值的钱数i，可以得到当前总金额cur_sum
25         cur_sum = pre_sum + i
26         dp[cur_sum] = max(dp[cur_sum], dp[pre_sum] + num)
27
28 # 输出dp数组中的最大值即为答案
29 print(max(dp))

```

## Java

### 1维dp哈希表

```

1 import java.util.HashMap;
2 import java.util.Map;
3 import java.util.Scanner;
4
5 public class Main {
6     public static void main(String[] args) {
7         Scanner scanner = new Scanner(System.in);
8
9         // 客户预算
10        int target = scanner.nextInt();
11        scanner.nextLine(); // 消费掉换行符
12
13        // 充值获得的短信条数
14        String[] numsStr = scanner.nextLine().split(" ");
15        int[] nums = new int[numsStr.length + 1];
16        for (int i = 1; i <= numsStr.length; i++) {
17            nums[i] = Integer.parseInt(numsStr[i - 1]);
18        }
19
20        // 构建1维dp哈希表，dp[i]表示充值总钱数为i元时，能获得的最大短信数
21        Map<Integer, Integer> dp = new HashMap<>();
22        // 初始化dp哈希表，dp[0] = 0表示使用0元可以获得0条短信
23        dp.put(0, 0);
24
25        // 路径无关的完全背包
26        // 先遍历物品nums

```

```

27      // i为下标, 也为充值的钱数i; num为充值钱数i能获得的短信条数
28      for (int i = 1; i < nums.length; i++) {
29          int num = nums[i];
30          // 再顺序遍历背包
31          for (int pre_sum = 0; pre_sum <= target; pre_sum++) {
32              // 考虑前一个可以获得的总金额为pre_sum
33              // 加上当前充值的钱数i, 可以得到当前总金额cur_sum
34              int cur_sum = pre_sum + i;
35              // 当前金额必须不超过预算
36              if (cur_sum <= target) {
37                  // 更新新的哈希表中的值
38                  dp.put(cur_sum, Math.max(dp.getOrDefault(cur_sum, 0),
dp.getOrDefault(pre_sum, 0) + num));
39              }
40          }
41      }
42
43      // 输出dp哈希表中的最大值即为答案
44      int maxSms = 0;
45      for (int value : dp.values()) {
46          if (value > maxSms) {
47              maxSms = value;
48          }
49      }
50      System.out.println(maxSms);
51  }
52 }
53

```

## 1维dp数组

```

1  import java.util.Scanner;
2
3  public class Main {
4      public static void main(String[] args) {
5          Scanner scanner = new Scanner(System.in);
6
7          // 客户预算
8          int target = scanner.nextInt();
9          scanner.nextLine(); // 消费掉换行符
10
11         // 充值获得的短信条数
12         String[] numsStr = scanner.nextLine().split(" ");
13         int[] nums = new int[numsStr.length + 1];
14         for (int i = 1; i <= numsStr.length; i++) {

```

```

15         nums[i] = Integer.parseInt(numsStr[i - 1]);
16     }
17
18     // 构建1维dp数组，dp[i]表示充值总钱数为i元时，能获得的最大短信数
19     int[] dp = new int[target + 1];
20
21     // 路径无关的完全背包
22     // 先遍历物品nums
23     // i为下标，也为充值的钱数i；num为充值钱数i能获得的短信条数
24     for (int i = 1; i < nums.length; i++) {
25         int num = nums[i];
26         // 再顺序遍历背包
27         for (int pre_sum = 0; pre_sum <= target - i; pre_sum++) {
28             // 考虑前一个可以获得的总金额为pre_sum
29             // 加上当前充值的钱数i，可以得到当前总金额cur_sum
30             int cur_sum = pre_sum + i;
31             dp[cur_sum] = Math.max(dp[cur_sum], dp[pre_sum] + num);
32         }
33     }
34
35     // 输出dp数组中的最大值即为答案
36     int maxSms = 0;
37     for (int sms : dp) {
38         if (sms > maxSms) {
39             maxSms = sms;
40         }
41     }
42     System.out.println(maxSms);
43 }
44 }
45

```

## C++

### 1维dp哈希表

```

1 #include <iostream>
2 #include <vector>
3 #include <unordered_map>
4 #include <algorithm>
5 using namespace std;
6
7 int main() {
8     // 客户预算
9     int target;

```

```

10     cin >> target;
11
12     // 充值获得的短信条数
13     vector<int> nums;
14     nums.push_back(0); // 把充值0元可以获得0条短信也视为一种情况
15     int num;
16     while (cin >> num) {
17         nums.push_back(num);
18         if (cin.peek() == '\n') break;
19     }
20
21     // 构建1维dp哈希表, dp[i]表示充值总钱数为i元时, 能获得的最大短信数
22     unordered_map<int, int> dp;
23     // 初始化dp哈希表, dp[0] = 0表示使用0元可以获得0条短信
24     dp[0] = 0;
25
26     // 路径无关的完全背包
27     // 先遍历物品nums
28     // i为下标, 也为充值的钱数i; num为充值钱数i能获得的短信条数
29     for (int i = 1; i < nums.size(); i++) {
30         int num = nums[i];
31         // 再顺序遍历背包
32         for (int pre_sum = 0; pre_sum <= target; pre_sum++) {
33             if (dp.find(pre_sum) != dp.end()) {
34                 // 考虑前一个可以获得的总金额为pre_sum
35                 // 加上当前充值的钱数i, 可以得到当前总金额cur_sum
36                 int cur_sum = pre_sum + i;
37                 // 当前金额必须不超过预算
38                 if (cur_sum <= target) {
39                     // 更新新的哈希表中的值
40                     dp[cur_sum] = max(dp[cur_sum], dp[pre_sum] + num);
41                 }
42             }
43         }
44     }
45
46     // 输出dp哈希表中的最大值即为答案
47     int maxSms = 0;
48     for (const auto& pair : dp) {
49         if (pair.second > maxSms) {
50             maxSms = pair.second;
51         }
52     }
53     cout << maxSms << endl;
54
55     return 0;
56 }

```

## 1维dp数组

```

1  #include <iostream>
2  #include <vector>
3  #include <algorithm>
4  using namespace std;
5
6  int main() {
7      // 客户预算
8      int target;
9      cin >> target;
10
11     // 充值获得的短信条数
12     vector<int> nums;
13     nums.push_back(0); // 把充值0元可以获得0条短信也视为一种情况
14     int num;
15     while (cin >> num) {
16         nums.push_back(num);
17         if (cin.peek() == '\n') break;
18     }
19
20     // 构建1维dp数组, dp[i]表示充值总钱数为i元时,能获得的最大短信数
21     vector<int> dp(target + 1, 0);
22
23     // 路径无关的完全背包
24     // 先遍历物品nums
25     // i为下标,也为充值的钱数i; num为充值钱数i能获得的短信条数
26     for (int i = 1; i < nums.size(); i++) {
27         int num = nums[i];
28         // 再顺序遍历背包
29         for (int pre_sum = 0; pre_sum <= target - i; pre_sum++) {
30             // 考虑前一个可以获得的总金额为pre_sum
31             // 加上当前充值的钱数i,可以得到当前总金额cur_sum
32             int cur_sum = pre_sum + i;
33             dp[cur_sum] = max(dp[cur_sum], dp[pre_sum] + num);
34         }
35     }
36
37     // 输出dp数组中的最大值即为答案
38     cout << *max_element(dp.begin(), dp.end()) << endl;
39
40     return 0;
41 }

```



## 时空复杂度

时间复杂度： $O(NM)$ 。需要遍历  $N$  个不同的数字，每个数字都要考虑  $M$  种前置情况。其中  $M = \text{len}(dp)$ 。

空间复杂度： $O(M)$ 。dp数组/哈希表所占空间。