【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 <= M <= 1000000`

第二行给出售价表，`P1，P2，…，Pn`,其中 `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

示例一

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`

示例二

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`

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

```python
# 题目：2024E-云短信平台优惠活动
# 分值：200
# 作者：许老师-闭着眼睛学数理化
# 算法：背包DP/1维dp哈希表
# 代码看不懂的地方，请直接在群上提问


from collections import defaultdict

```

```python
# 客户预算
target = int(input())
# 充值获得的短信条数
nums = list(map(int, input().split()))
# 把充值0元可以获得0条短信也视为一种情况
nums = [0] + nums
# 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
dp = defaultdict(int)
# 初始化dp哈希表，dp[0] = 0表示使用0元可以获得0条短信
dp[0] = 0

# 路径无关的完全背包
# 先遍历物品nums
# 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
for i, num in enumerate(nums[1:], 1):
    # 再顺序遍历背包
    for pre_sum in range(target+1):
        # 考虑前一个可以获得的总金额为pre_sum
        # 加上当前充值的钱数i，可以得到当前总金额cur_sum
        cur_sum = pre_sum + i
        # 当前金额必须不超过预算
        if cur_sum <= target:
            # 更新新的哈希表中的值
            dp[cur_sum] = max(dp[cur_sum], dp[pre_sum] + num)

# 输出dp哈希表中的最大值即为答案
print(max(dp.values()))
```

1维dp数组

```python
# 题目：2024E-云短信平台优惠活动
# 分值：200
# 作者：许老师-闭着眼睛学数理化
# 算法：背包DP/1维dp数组
# 代码看不懂的地方，请直接在群上提问


# 客户预算
target = int(input())
# 充值获得的短信条数
nums = list(map(int, input().split()))
# 把充值0元可以获得0条短信也视为一种情况
nums = [0] + nums
# 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
dp = [0] * (target + 1)
```

```python

# 路径无关的完全背包
# 先遍历物品nums
# 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
for i, num in enumerate(nums[1:], 1):
    # 再顺序遍历背包
    for pre_sum in range(0, target-i+1):
        # 考虑前一个可以获得的总金额为pre_sum
        # 加上当前充值的钱数i，可以得到当前总金额cur_sum
        cur_sum = pre_sum + i
        dp[cur_sum] = max(dp[cur_sum], dp[pre_sum] + num)

# 输出dp数组中的最大值即为答案
print(max(dp))
```

Java

1维dp哈希表

```java
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;

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

        // 客户预算
        int target = scanner.nextInt();
        scanner.nextLine();  // 消费掉换行符

        // 充值获得的短信条数
        String[] numsStr = scanner.nextLine().split(" ");
        int[] nums = new int[numsStr.length + 1];
        for (int i = 1; i <= numsStr.length; i++) {
            nums[i] = Integer.parseInt(numsStr[i - 1]);
        }

        // 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
        Map<Integer, Integer> dp = new HashMap<>();
        // 初始化dp哈希表，dp[0] = 0表示使用0元可以获得0条短信
        dp.put(0, 0);

        // 路径无关的完全背包
        // 先遍历物品nums
```

```
        // 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
        for (int i = 1; i < nums.length; i++) {
            int num = nums[i];
            // 再顺序遍历背包
            for (int pre_sum = 0; pre_sum <= target; pre_sum++) {
                // 考虑前一个可以获得的总金额为pre_sum
                // 加上当前充值的钱数i，可以得到当前总金额cur_sum
                int cur_sum = pre_sum + i;
                // 当前金额必须不超过预算
                if (cur_sum <= target) {
                    // 更新新的哈希表中的值
                    dp.put(cur_sum, Math.max(dp.getOrDefault(cur_sum, 0),
dp.getOrDefault(pre_sum, 0)+ num));
                }
            }
        }

        // 输出dp哈希表中的最大值即为答案
        int maxSms = 0;
        for (int value : dp.values()) {
            if (value > maxSms) {
                maxSms = value;
            }
        }
        System.out.println(maxSms);
    }
}

```

1维dp数组

```
import java.util.Scanner;

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

        // 客户预算
        int target = scanner.nextInt();
        scanner.nextLine();  // 消费掉换行符

        // 充值获得的短信条数
        String[] numsStr = scanner.nextLine().split(" ");
        int[] nums = new int[numsStr.length + 1];
        for (int i = 1; i <= numsStr.length; i++) {
```

```
            nums[i] = Integer.parseInt(numsStr[i - 1]);
        }

        // 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
        int[] dp = new int[target + 1];

        // 路径无关的完全背包
        // 先遍历物品nums
        // 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
        for (int i = 1; i < nums.length; i++) {
            int num = nums[i];
            // 再顺序遍历背包
            for (int pre_sum = 0; pre_sum <= target - i; pre_sum++) {
                // 考虑前一个可以获得的总金额为pre_sum
                // 加上当前充值的钱数i，可以得到当前总金额cur_sum
                int cur_sum = pre_sum + i;
                dp[cur_sum] = Math.max(dp[cur_sum], dp[pre_sum] + num);
            }
        }

        // 输出dp数组中的最大值即为答案
        int maxSms = 0;
        for (int sms : dp) {
            if (sms > maxSms) {
                maxSms = sms;
            }
        }
        System.out.println(maxSms);
    }
}

```

C++

1维dp哈希表

```
#include <iostream>
#include <vector>
#include <unordered_map>
#include <algorithm>
using namespace std;

int main() {
    // 客户预算
    int target;
```

```cpp
    cin >> target;

    // 充值获得的短信条数
    vector<int> nums;
    nums.push_back(0);  // 把充值0元可以获得0条短信也视为一种情况
    int num;
    while (cin >> num) {
        nums.push_back(num);
        if (cin.peek() == '\n') break;
    }

    // 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
    unordered_map<int, int> dp;
    // 初始化dp哈希表，dp[0] = 0表示使用0元可以获得0条短信
    dp[0] = 0;

    // 路径无关的完全背包
    // 先遍历物品nums
    // 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
    for (int i = 1; i < nums.size(); i++) {
        int num = nums[i];
        // 再顺序遍历背包
        for (int pre_sum = 0; pre_sum <= target; pre_sum++) {
            if (dp.find(pre_sum) != dp.end()) {
                // 考虑前一个可以获得的总金额为pre_sum
                // 加上当前充值的钱数i，可以得到当前总金额cur_sum
                int cur_sum = pre_sum + i;
                // 当前金额必须不超过预算
                if (cur_sum <= target) {
                    // 更新新的哈希表中的值
                    dp[cur_sum] = max(dp[cur_sum], dp[pre_sum] + num);
                }
            }
        }
    }

    // 输出dp哈希表中的最大值即为答案
    int maxSms = 0;
    for (const auto& pair : dp) {
        if (pair.second > maxSms) {
            maxSms = pair.second;
        }
    }
    cout << maxSms << endl;

    return 0;
}
```

```
57
```

1维dp数组

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int main() {
    // 客户预算
    int target;
    cin >> target;

    // 充值获得的短信条数
    vector<int> nums;
    nums.push_back(0);  // 把充值0元可以获得0条短信也视为一种情况
    int num;
    while (cin >> num) {
        nums.push_back(num);
        if (cin.peek() == '\n') break;
    }

    // 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
    vector<int> dp(target + 1, 0);

    // 路径无关的完全背包
    // 先遍历物品nums
    // 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
    for (int i = 1; i < nums.size(); i++) {
        int num = nums[i];
        // 再顺序遍历背包
        for (int pre_sum = 0; pre_sum <= target - i; pre_sum++) {
            // 考虑前一个可以获得的总金额为pre_sum
            // 加上当前充值的钱数i，可以得到当前总金额cur_sum
            int cur_sum = pre_sum + i;
            dp[cur_sum] = max(dp[cur_sum], dp[pre_sum] + num);
        }
    }

    // 输出dp数组中的最大值即为答案
    cout << *max_element(dp.begin(), dp.end()) << endl;

    return 0;
}
```

```
42
```

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 = len(dp)`。

空间复杂度：`O(M)`。dp数组/哈希表所占空间。