# 【DP】-充电设备

# 题目描述与示例

# 题目描述

某个充电站,可提供 n 个充电设备,每个充电设备均有对应的输出功率。 任意个充电设备组合的输出功率总和,均构成功率集合 P 的一个元素。 功率集合 P 的最优元素,表示最接近充电站最大输出功率 p\_max 的元素。

#### 输入描述

输入为 3 行:

第一行: 充电设备个数 n

第二行:每个充电设备的输出功率

第三行: 充电站最大输出功率 p\_max

#### 输出描述

功率集合P的最优元素

补充说明:

- 1. 充电设备个数 n > 0
- 2. 最优元素必须**小于或等于**充电站最大输出功率 p\_max

#### 示例一

#### 输入

```
1 3
2 1 2 3
3 5
```

```
1 5
```

说明

## 示例二

## 输入

```
1 3
2 2 3 10
3 9
```

#### 输出

1 5

#### 说明

选择功率为 2 , 3 的设备构成功率集合,总功率为 5 ,最接近最大功率 9 。不能选择设备 10 ,因为已经超过了最大功率 9 。

# 解题思路

本题属于路径无关、判断是否能够取到某值的01背包问题。

具体讲解可见 目 2023/07/07 真题讲解 (动态规划背包问题专题)。

# 代码

**Python** 

```
1 # 题目: 2023A-充电设备
2 # 分值: 100
3 # 作者: 许老师-闭着眼睛学数理化
4 # 算法: 背包DP/二维DP数组
5 # 代码看不懂的地方,请直接在群上提问
6
7 # 输入充电设备数量
8 n = int(input())
9 # 输入n个充电设备的功率
10 devices = list(map(int, input().split()))
11 # 输入最大充电功率
12 p_max = int(input())
13
14
15 # 构建二维dp数组,长度为(n+1)*(p max+1),为布尔类型的二维数组
16 # dp[i][j]的含义为
17 # 在考虑第i个设备时,功率i是否能够取到
18 dp = [[False] * (p_max+1) for _ in range(n+1)]
19 # 初始化dp[0][0]为True,表示可以取到
20 dp[0][0] = True
21
22 # 遍历每一种设备的功率p
23 # 注意为了和二维dp数组的索引——对应,故i从1开始,取到n结束
24 for i in range(1, n+1):
     # i是从1开始计数的,故p应为devices[i-1]
25
26
     p = devices[i-1]
      # 遍历dp[i-1]数组,所有可以取到的功率总和,用pre_p表示
27
     for pre_p in range(0, p_max+1):
28
         # 假如某个功率总和pre_p可以取到,则更新到dp[i][pre_p]
29
         if dp[i-1][pre_p] == True:
30
            dp[i][pre_p] = True
31
            # 假如从某个功率总和pre p出发,加上当前的设备功率p
32
            # 能够取得的总设备功率为cur_p
33
34
            cur_p = p + pre_p
            # 如果cur 0没有超出最大功率限制
35
            if cur_p <= p_max:</pre>
36
                # 则cur_p是一个可以取得的方案,将dp[i][cur_p]修改为True
37
                dp[i][cur_p] = True
38
39
40 # 输出dp[-1]数组中为True的最大值,即为小于等于p_max的最大功率
41 print(max((i for i in range(p_max+1) if dp[-1][i])))
```

```
1 # 题目: 2024E-充电设备
2 # 分值: 100
3 # 作者: 许老师-闭着眼睛学数理化
4 # 算法: 背包DP/一维DP数组
5 # 代码看不懂的地方,请直接在群上提问
6
7
8 # 输入充电设备数量
9 n = int(input())
10 # 输入n个充电设备的功率
11 devices = list(map(int, input().split()))
12 # 输入最大充电功率
13 p_max = int(input())
14
15 # 构建一维dp数组,长度为(p_max+1),为布尔类型的数组
16 # dp[i]功率i是否能够取到
17 dp = [False] * (p_max+1)
18 # 初始化dp[0]为True,表示可以取到
19 dp[0] = True
20
21 # 遍历每一种设备的功率p
22 for p in devices:
      # 遍历当前dp数组,所有可以取到的功率总和,用pre_p表示
23
      # 这里必须使用拷贝,因为在本次遍历中, dp数组会发生改变
24
     # dp数组正在发生的改变,不能在遍历中被考虑
25
26
     # 另一种可行的方法是,逆序遍历dp数组
27
      # 这样可以保证大功率的修改总是发生在遇到小功率之前
28
     temp = dp[:]
29
     for pre_p in range(0, p_max+1):
30
         # 假如从某个功率总和pre_p出发,加上当前的设备功率p
31
         # 能够取得的总设备功率为cur p
32
         if temp[pre_p] == True:
33
            cur_p = p + pre_p
34
            # 如果cur_0没有超出最大功率限制
35
36
            if cur_p <= p_max:</pre>
                # 则cur_p是一个可以取得的方案,将dp[cur_p]修改为True
37
               dp[cur_p] = True
38
39
40 # 输出dp数组中为True的最大值,即为小于等于p_max的最大功率
41 print(max((i for i in range(p_max+1) if dp[i] == True)))
```

#### 1维DP哈希表

```
2 # 分值: 100
3 # 作者: 许老师-闭着眼睛学数理化
4 # 算法: 背包DP/一维DP哈希表
5 # 代码看不懂的地方,请直接在群上提问
6
7
8 # 输入充电设备数量
9 n = int(input())
10 # 输入n个充电设备的功率
11 devices = list(map(int, input().split()))
12 # 输入最大充电功率
13 p_max = int(input())
14
15
16 # 构建dp哈希集合,集合中的元素表示能够出现的功率总和
17 dp = set()
18 # 初始化dp哈希集合包含0,表示不选择任何设备的方案,此时功率总和为0
19 dp.add(0)
20
21
22 # 遍历每一种设备的功率p
23 for p in devices:
      # 遍历当前dp哈希集合中,所有可以取到的功率总和,用pre_p表示
24
     for pre_p in list(dp):
25
         # 假如从某个功率总和pre_p出发,加上当前的设备功率p
26
         # 能够取得的总设备功率为cur_p
27
         cur_p = p + pre_p
28
         # 如果cur O没有超出最大功率限制
29
        if cur_p <= p_max:</pre>
30
            # 则cur_p是一个可以取得的方案,加入dp哈希集合中
31
32
            dp.add(cur_p)
33
34
35 # 输出dp数组中的最大值,即为小于等于p_max的最大功率
36 print(max(dp))
```

#### Java

```
1 import java.util.*;
2
3 public class Main {
4    public static void main(String[] args) {
5         Scanner input = new Scanner(System.in);
```

```
6
 7
            int n = input.nextInt();
            int[] devices = new int[n];
 8
 9
            for (int i = 0; i < n; i++) {
                devices[i] = input.nextInt();
10
11
            }
            int p_max = input.nextInt();
12
13
14
            boolean[][] dp = new boolean[n + 1][p_max + 1];
15
            dp[0][0] = true;
16
            for (int i = 1; i <= n; i++) {
17
                int p = devices[i - 1];
18
                for (int pre_p = 0; pre_p <= p_max; pre_p++) {</pre>
19
                    if (dp[i - 1][pre_p]) {
20
21
                        dp[i][pre_p] = true;
22
                        int cur_p = p + pre_p;
23
                        if (cur_p <= p_max) {</pre>
24
                             dp[i][cur_p] = true;
25
                        }
26
                    }
                }
27
            }
28
29
            int maxPower = -1;
30
            for (int i = 0; i <= p_max; i++) {
31
32
                if (dp[n][i]) {
                    maxPower = i;
33
                }
34
            }
35
36
            System.out.println(maxPower);
37
       }
38
39 }
```

```
1 import java.util.*;
2
3 public class Main {
4    public static void main(String[] args) {
5         Scanner input = new Scanner(System.in);
6
7         int n = input.nextInt();
8         List<Integer> devices = new ArrayList<>();
```

```
for (int i = 0; i < n; i++) {
9
                devices.add(input.nextInt());
10
            }
11
            int pMax = input.nextInt();
12
13
            boolean[] dp = new boolean[pMax + 1];
14
15
            dp[0] = true;
16
17
            for (int p : devices) {
                boolean[] temp = dp.clone();
18
                for (int preP = 0; preP <= pMax; preP++) {</pre>
19
                    if (temp[preP]) {
20
                         int curP = p + preP;
21
22
                         if (curP <= pMax) {</pre>
                             dp[curP] = true;
23
24
                         }
                    }
25
26
                }
27
            }
28
29
            int maxP = -1;
            for (int i = 0; i <= pMax; i++) {
30
                if (dp[i]) {
31
32
                    maxP = i;
                }
33
            }
34
35
            System.out.println(maxP);
36
37
       }
38 }
```

#### 1维DP哈希表

```
1 import java.util.HashSet;
 2 import java.util.Scanner;
3 import java.util.Set;
4
 5 public class Main {
       public static void main(String[] args) {
 6
           Scanner scanner = new Scanner(System.in);
 7
 8
           int n = scanner.nextInt();
           int[] devices = new int[n];
9
           for (int i = 0; i < n; i++) {
10
               devices[i] = scanner.nextInt();
11
           }
12
```

```
13
            int pMax = scanner.nextInt();
14
            Set<Integer> dp = new HashSet<>();
15
            dp.add(0);
16
17
            for (int p : devices) {
18
                Set<Integer> temp = new HashSet<>(dp);
19
20
                for (int preP : temp) {
21
                    int curP = p + preP;
                    if (curP <= pMax) {</pre>
22
                        dp.add(curP);
23
                    }
24
                }
25
            }
26
27
            int maxPower = dp.stream().mapToInt(Integer::intValue).max().orElse(0);
28
            System.out.println(maxPower);
29
30
       }
31 }
32
```

#### C++

```
1 #include <iostream>
 2 #include <vector>
 3 using namespace std;
 4
 5 int main() {
       int n;
 6
 7
       cin >> n;
 8
 9
       vector<int> devices(n);
       for (int i = 0; i < n; i++) {
10
           cin >> devices[i];
11
       }
12
13
       int pMax;
14
       cin >> pMax;
15
16
17
       vector<vector<bool>> dp(n + 1, vector<bool>(pMax + 1, false));
       dp[0][0] = true;
18
19
20
       for (int i = 1; i <= n; i++) {
```

```
int p = devices[i - 1];
21
            for (int preP = 0; preP <= pMax; preP++) {</pre>
22
                if (dp[i - 1][preP]) {
23
24
                     dp[i][preP] = true;
                     int curP = p + preP;
25
                    if (curP <= pMax) {</pre>
26
27
                         dp[i][curP] = true;
                     }
28
29
                }
            }
30
       }
31
32
       int maxP = -1;
33
        for (int i = 0; i <= pMax; i++) {
34
            if (dp[n][i]) {
35
                maxP = i;
36
           }
37
38
        }
39
        cout << maxP << endl;</pre>
40
41
       return 0;
42
43 }
```

```
1 #include <iostream>
 2 #include <vector>
 3 using namespace std;
 5 int main() {
       int n;
       cin >> n;
 7
 8
       vector<int> devices(n);
 9
10
       for (int i = 0; i < n; i++) {
           cin >> devices[i];
11
12
       }
13
14
       int pMax;
       cin >> pMax;
15
16
       vector<bool> dp(pMax + 1, false);
17
18
       dp[0] = true;
19
```

```
for (int p : devices) {
20
            vector<bool> temp = dp;
21
            for (int preP = 0; preP <= pMax; preP++) {</pre>
22
                 if (temp[preP]) {
23
                     int curP = p + preP;
24
                     if (curP <= pMax) {</pre>
25
26
                         dp[curP] = true;
                     }
27
28
                }
            }
29
        }
30
31
        int maxP = -1;
32
33
        for (int i = 0; i <= pMax; i++) {
            if (dp[i]) {
34
                maxP = i;
35
           }
36
37
        }
38
        cout << maxP << endl;</pre>
39
40
41
        return 0;
42 }
```

## 1维DP哈希表

```
1 #include <iostream>
 2 #include <vector>
 3 #include <unordered_set>
 4 #include <algorithm>
 5
 6 using namespace std;
 7
 8 int main() {
       int n;
 9
10
       cin >> n;
11
       vector<int> devices(n);
       for (int i = 0; i < n; i++) {
12
           cin >> devices[i];
13
       }
14
15
       int p_max;
       cin >> p_max;
16
17
       unordered_set<int> dp;
18
19
       dp.insert(0);
```

```
20
       for (int p : devices) {
21
22
            vector<int> temp(dp.begin(), dp.end());
23
           for (int pre_p : temp) {
                int cur_p = p + pre_p;
24
25
                if (cur_p <= p_max) {</pre>
26
                    dp.insert(cur_p);
27
                }
           }
28
       }
29
30
       int maxPower = *max_element(dp.begin(), dp.end());
31
       cout << maxPower << endl;</pre>
32
33
34
       return 0;
35 }
```

## 时空复杂度

时间复杂度: O(N\*p\_max)。需要遍历 N 种不同功率的设备,每个设备都要考虑 len(dp) 种前置情况。

空间复杂度: O(p\_max) 。dp哈希集合最多储存 p\_max+1 个元素。