【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

```
3
1 2 3
5
```

输出

```
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 2023/07/07 真题讲解 （动态规划背包问题专题） 。

代码

Python

2维DP数组

```python
# 题目：2023A-充电设备
# 分值：100
# 作者：许老师-闭着眼睛学数理化
# 算法：背包DP/二维DP数组
# 代码看不懂的地方，请直接在群上提问

# 输入充电设备数量
n = int(input())
# 输入n个充电设备的功率
devices = list(map(int, input().split()))
# 输入最大充电功率
p_max = int(input())


# 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
# dp[i][j]的含义为
# 在考虑第i个设备时，功率j是否能够取到
dp = [[False] * (p_max+1) for _ in range(n+1)]
# 初始化dp[0][0]为True，表示可以取到
dp[0][0] = True

# 遍历每一种设备的功率p
# 注意为了和二维dp数组的索引一一对应，故i从1开始，取到n结束
for i in range(1, n+1):
    # i是从1开始计数的，故p应为devices[i-1]
    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
    for pre_p in range(0, p_max+1):
        # 假如某个功率总和pre_p可以取到，则更新到dp[i][pre_p]
        if dp[i-1][pre_p] == True:
            dp[i][pre_p] = True
            # 假如从某个功率总和pre_p出发，加上当前的设备功率p
            # 能够取得的总设备功率为cur_p
            cur_p = p + pre_p
            # 如果cur_0没有超出最大功率限制
            if cur_p <= p_max:
                # 则cur_p是一个可以取得的方案，将dp[i][cur_p]修改为True
                dp[i][cur_p] = True

# 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
print(max((i for i in range(p_max+1) if dp[-1][i])))
```

1维DP数组

```python
# 题目：2024E-充电设备
# 分值：100
# 作者：许老师-闭着眼睛学数理化
# 算法：背包DP/一维DP数组
# 代码看不懂的地方，请直接在群上提问


# 输入充电设备数量
n = int(input())
# 输入n个充电设备的功率
devices = list(map(int, input().split()))
# 输入最大充电功率
p_max = int(input())

# 构建一维dp数组，长度为(p_max+1)，为布尔类型的数组
# dp[i]功率i是否能够取到
dp = [False] * (p_max+1)
# 初始化dp[0]为True，表示可以取到
dp[0] = True

# 遍历每一种设备的功率p
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
    # 这里必须使用拷贝，因为在本次遍历中，dp数组会发生改变
    # dp数组正在发生的改变，不能在遍历中被考虑
    #
    # 另一种可行的方法是，逆序遍历dp数组
    # 这样可以保证大功率的修改总是发生在遇到小功率之前
    temp = dp[:]
    for pre_p in range(0, p_max+1):
        # 假如从某个功率总和pre_p出发，加上当前的设备功率p
        # 能够取得的总设备功率为cur_p
        if temp[pre_p] == True:
            cur_p = p + pre_p
            # 如果cur_0没有超出最大功率限制
            if cur_p <= p_max:
                # 则cur_p是一个可以取得的方案，将dp[cur_p]修改为True
                dp[cur_p] = True

# 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
print(max((i for i in range(p_max+1) if dp[i] == True)))
```

1维DP哈希表

```python
# 题目：2024E-充电设备
```

```python
# 分值：100
# 作者：许老师-闭着眼睛学数理化
# 算法：背包DP/一维DP哈希表
# 代码看不懂的地方，请直接在群上提问


# 输入充电设备数量
n = int(input())
# 输入n个充电设备的功率
devices = list(map(int, input().split()))
# 输入最大充电功率
p_max = int(input())


# 构建dp哈希集合，集合中的元素表示能够出现的功率总和
dp = set()
# 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
dp.add(0)


# 遍历每一种设备的功率p
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
    for pre_p in list(dp):
        # 假如从某个功率总和pre_p出发，加上当前的设备功率p
        # 能够取得的总设备功率为cur_p
        cur_p = p + pre_p
        # 如果cur_0没有超出最大功率限制
        if cur_p <= p_max:
            # 则cur_p是一个可以取得的方案，加入dp哈希集合中
            dp.add(cur_p)


# 输出dp数组中的最大值，即为小于等于p_max的最大功率
print(max(dp))
```

Java

2维DP数组

```java
import java.util.*;

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

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

1维DP数组

```
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++) {
            devices.add(input.nextInt());
        }
        int pMax = input.nextInt();

        boolean[] dp = new boolean[pMax + 1];
        dp[0] = true;

        for (int p : devices) {
            boolean[] temp = dp.clone();
            for (int preP = 0; preP <= pMax; preP++) {
                if (temp[preP]) {
                    int curP = p + preP;
                    if (curP <= pMax) {
                        dp[curP] = true;
                    }
                }
            }
        }

        int maxP = -1;
        for (int i = 0; i <= pMax; i++) {
            if (dp[i]) {
                maxP = i;
            }
        }

        System.out.println(maxP);
    }
}
```

1维DP哈希表

```
import java.util.HashSet;
import java.util.Scanner;
import java.util.Set;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt();
        int[] devices = new int[n];
        for (int i = 0; i < n; i++) {
            devices[i] = scanner.nextInt();
        }
```

```
        int pMax = scanner.nextInt();

        Set<Integer> dp = new HashSet<>();
        dp.add(0);

        for (int p : devices) {
            Set<Integer> temp = new HashSet<>(dp);
            for (int preP : temp) {
                int curP = p + preP;
                if (curP <= pMax) {
                    dp.add(curP);
                }
            }
        }

        int maxPower = dp.stream().mapToInt(Integer::intValue).max().orElse(0);
        System.out.println(maxPower);
    }
}

```

C++

2维DP数组

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

int main() {
    int n;
    cin >> n;

    vector<int> devices(n);
    for (int i = 0; i < n; i++) {
        cin >> devices[i];
    }

    int pMax;
    cin >> pMax;

    vector<vector<bool>> dp(n + 1, vector<bool>(pMax + 1, false));
    dp[0][0] = true;

    for (int i = 1; i <= n; i++) {
```

```
        int p = devices[i - 1];
        for (int preP = 0; preP <= pMax; preP++) {
            if (dp[i - 1][preP]) {
                dp[i][preP] = true;
                int curP = p + preP;
                if (curP <= pMax) {
                    dp[i][curP] = true;
                }
            }
        }
    }

    int maxP = -1;
    for (int i = 0; i <= pMax; i++) {
        if (dp[n][i]) {
            maxP = i;
        }
    }

    cout << maxP << endl;

    return 0;
}
```

1维DP数组

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

int main() {
    int n;
    cin >> n;

    vector<int> devices(n);
    for (int i = 0; i < n; i++) {
        cin >> devices[i];
    }

    int pMax;
    cin >> pMax;

    vector<bool> dp(pMax + 1, false);
    dp[0] = true;

```

```
    for (int p : devices) {
        vector<bool> temp = dp;
        for (int preP = 0; preP <= pMax; preP++) {
            if (temp[preP]) {
                int curP = p + preP;
                if (curP <= pMax) {
                    dp[curP] = true;
                }
            }
        }
    }

    int maxP = -1;
    for (int i = 0; i <= pMax; i++) {
        if (dp[i]) {
            maxP = i;
        }
    }

    cout << maxP << endl;

    return 0;
}
```

1维DP哈希表

```
#include <iostream>
#include <vector>
#include <unordered_set>
#include <algorithm>

using namespace std;

int main() {
    int n;
    cin >> n;
    vector<int> devices(n);
    for (int i = 0; i < n; i++) {
        cin >> devices[i];
    }
    int p_max;
    cin >> p_max;

    unordered_set<int> dp;
    dp.insert(0);
```

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