华为OD机试 - 代表团坐车（Java & JS & Python）

原创 伏城之外 已于 2023-05-21 18:05:07 修改 595 收藏 4 版权

分类专栏： 华为OD机试（Java & JS & Python） # 华为OD机试2023B # 华为OD机试2023A

文章标签： 算法 华为机试 Java Python JavaScript

目录

题目描述

某组织举行会议，来了多个代表团同时到达，接待处只有一辆汽车，可以同时接待多个代表团，为了提高车辆利用率，请帮接待员计算可以坐满车的接待方案，输出方案数量。

约束:

1. 一个团只能上一辆车，并且代表团人数 (代表团数量小于30，每个代表团人数小于30)小于汽车容量(汽车容量小于100)
2. 需要将车辆坐满

输入描述

第一行 代表团人数，英文逗号隔开，代表团数量小于30，每个代表团人数小于30
第二行 汽车载客量，汽车容量小于100

输出描述

坐满汽车的方案数量
如果无解输出0

用例

输入	5,4,2,3,2,4,9 10
输出	4
说明	解释 以下几种方式都可以坐满车，所以，优先接待输出为4 [2,3,5] [2,4,4] [2,3,5] [2,4,4]

题目解析

本题可以转化为01背包的装满背包的方案数问题。

解析可以参考：LeetCode - 494 目标和_伏城之外的博客-CSDN博客

也可以尝试下：华为校招机试 - 求和（Java & JS & Python）_伏城之外的博客-CSDN博客

Java算法源码

二维数组解法

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

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

    Integer[] nums =
        Arrays.stream(sc.nextLine().split(",")).map(Integer::parseInt).toArray

    int bag = Integer.parseInt(sc.nextLine());

    System.out.println(getResult(nums, bag));
  }

  private static int getResult(Integer[] nums, int bag) {
    int n = nums.length;

    int[][] dp = new int[n + 1][bag + 1];
    dp[0][0] = 1;

    for (int i = 1; i <= n; i++) {
      int num = nums[i - 1];
      for (int j = 0; j <= bag; j++) {
        if (j < num) {
          dp[i][j] = dp[i - 1][j];
        } else {
          dp[i][j] = dp[i - 1][j] + dp[i - 1][j - num];
        }
      }
    }

    return dp[n][bag];
  }
}
```

滚动数组优化解法

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

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

    Integer[] nums =
        Arrays.stream(sc.nextLine().split(",")).map(Integer::parseInt).toArray

    int bag = Integer.parseInt(sc.nextLine());

    System.out.println(getResult(nums, bag));
  }

  private static int getResult(Integer[] nums, int bag) {
    int n = nums.length;

    int[] dp = new int[bag + 1];
    dp[0] = 1;

    for (int i = 1; i <= n; i++) {
      int num = nums[i - 1];
      for (int j = bag; j >= num; j--) {
        dp[j] = dp[j] + dp[j - num];
      }
    }

    return dp[bag];
  }
}
```

JS算法源码

二维数组解法

```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 nums = lines[0].split(",").map(Number);
    const bag = lines[1] - 0;
    console.log(getResult(nums, bag));
    lines.length = 0;
  }
});

function getResult(nums, bag) {
  const n = nums.length;

  const dp = new Array(n + 1).fill(0).map(() => new Array(bag + 1).fill(0));
  dp[0][0] = 1;

  for (let i = 1; i <= n; i++) {
    const num = nums[i - 1];
    for (let j = 0; j <= bag; j++) {
      if (j < num) {
        dp[i][j] = dp[i - 1][j];
      } else {
        dp[i][j] = dp[i - 1][j] + dp[i - 1][j - num];
      }
    }
  }

  return dp[n][bag];
}
```

滚动数组优化解法

```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 nums = lines[0].split(",").map(Number);
    const bag = lines[1] - 0;
    console.log(getResult(nums, bag));
    lines.length = 0;
  }
});

function getResult(nums, bag) {
  const n = nums.length;

  const dp = new Array(bag + 1).fill(0);
  dp[0] = 1;

  for (let i = 1; i <= n; i++) {
    const num = nums[i - 1];
    for (let j = bag; j >= num; j--) {
      dp[j] = dp[j] + dp[j - num];
    }
  }

  return dp[bag];
}
```

Python算法源码

二维数组解法

```python
# 输入获取
nums = list(map(int, input().split(",")))
bag = int(input())


# 算法入口
def getResult():
    n = len(nums)

    dp = [[0] * (bag + 1) for _ in range(n+1)]
    dp[0][0] = 1

    for i in range(1, n + 1):
        num = nums[i - 1]
        for j in range(bag + 1):
            if j < num:
                dp[i][j] = dp[i - 1][j]
            else:
                dp[i][j] = dp[i - 1][j] + dp[i - 1][j - num]

    return dp[n][bag]


# 算法调用
print(getResult())
```

滚动数组优化解法

```python
# 输入获取
nums = list(map(int, input().split(",")))
bag = int(input())


# 算法入口
def getResult():
    n = len(nums)

    dp = [0] * (bag + 1)
    dp[0] = 1

    for i in range(1, n + 1):
        num = nums[i - 1]
        for j in range(bag, num-1, -1):
            dp[j] = dp[j] + dp[j - num]

    return dp[bag]


# 算法调用
print(getResult())
```

文章知识点与官方知识档案匹配，可进一步学习相关知识