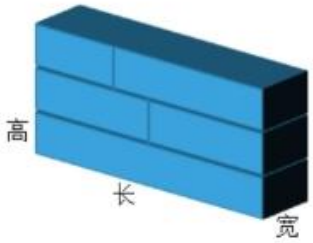
题目描述

有一堆长方体积木，它们的高度和宽度都相同，但长度不一。

小橙想把这堆积木叠成一面墙，墙的每层可以放一个积木，也可以将两个积木拼接起来，要求每层的长度相同。

若必须用完这些积木，叠成的墙最多为多少层?

如下是叠成的一面墙的图示，积木仅按宽和高所在的面进行拼接。



输入描述

输入为一行，为各个积木的长度，数字为正整数，并由空格分隔。积木的数量和长度都不超过5000。

输出描述

输出一个数字，为墙的最大层数，如果无法按要求叠成每层长度一致的墙，则输出-1。

用例

输入	3 6 3 3 3
输出	3
说明	以 6 为底的墙，第一层为 6，第二层为 3 + 3，第三层 3 + 3。

输入	9 9 9 5 3 2 2 2 2
输出	5
说明	5+2+2=9 3+2+2+2=9 9,9,9 共五层

题目解析

本题算是

的变种题，本题同样也是求 将数组划分为k个和相等的子集。

但是本题并未直接给出k，而是让我们求出最大的k。

这里k的求解很简单，首先，我们可以猜想下k的上限是多少？

比如数组所有元素都相等，则k === arr.length，即每个元素都能作为一层，因此我们可以让k从arr.length开始尝试，如果不行，则k--，直到k=2，即最少分两层。

而验证arr是否可以划分为k层，其实就是判断arr是否可以划分为k个和相等的子集，这个判断逻辑可以复用

中的逻辑。

需要注意的是，本题的用例似乎和题目描述冲突，比如题目描述中："墙的每层可以放一个积木，也可以将两个积木拼接起来"，我理解是一层只能用一个积木，或最多两个积木，但是用例2中

> 5+2+2=9
> 3+2+2+2=9
> 9,9,9
> 共五层

红色标记的层，却可以用超过两个积木。。。。这里我以用例为准，即一层可以用多个积木。

另外，本题输出描述说："输出一个数字，为墙的最大层数，如果无法按要求叠成每层长度一致的墙，则输出-1。"

我是这么想的，既然可以一层用多个积木，那我把所有积木都盖第一层不行吗？那最大层级至少也可以为1吧，那岂不是说永远不能输出-1？

这里我理解题目应该是不允许只盖一层，因此我默认最大层数至少为2。

JavaScript算法源码

```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,
});

rl.on("line", (line) => {
  const nums = line.split(" ").map(Number);
  console.log(getResult(nums));
});

function getResult(arr) {
  let m = arr.length;
  const sum = arr.sort((a, b) => b - a).reduce((p, c) => p + c);

  while (m > 1) {
    if (canPartitionMSubsets([...arr], sum, m)) return m;
    m--;
  }

  return -1;
}

```

```javascript
function canPartitionMSubsets(arr, sum, m) {
  if (sum % m !== 0) return false;

  const subSum = sum / m;

  if (subSum < arr[0]) return false;

  while (arr[0] === subSum) {
    arr.shift();
    m--;
  }

  const buckets = new Array(m).fill(0);

  return partition(arr, 0, buckets, subSum);
}

function partition(arr, index, buckets, subSum) {
  if (index === arr.length) return true;

  const select = arr[index];

  for (let i = 0; i < buckets.length; i++) {
    if (i > 0 && buckets[i] === buckets[i - 1]) continue;
    if (select + buckets[i] <= subSum) {
      buckets[i] += select;
      if (partition(arr, index + 1, buckets, subSum)) return true;
      buckets[i] -= select;
    }
  }

  return false;
}
```

Java算法源码

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

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

    Integer[] arr =
        Arrays.stream(sc.nextLine().split(" ")).map(Integer::parseInt).toArray(Integer[]::new);

    System.out.println(getResult(arr));
  }

  public static int getResult(Integer[] arr) {
    Arrays.sort(arr, (a, b) -> b - a);

    int sum = 0;
    for (Integer ele : arr) {
      sum += ele;
    }

    int m = arr.length;

    while (m > 1) {
      LinkedList<Integer> link = new LinkedList<>();
      Collections.addAll(link, arr);
      if (canPartitionMSubsets(link, sum, m)) return m;
      m--;
    }

    return -1;
  }

```

```java
  public static boolean canPartitionMSubsets(LinkedList<Integer> link, int sum, int m) {
    if (sum % m != 0) return false;

    int subSum = sum / m;

    if (subSum < link.get(0)) return false;

    //    while (link.get(0) == subSum) { // 此段代码可能超界
    while (link.size() > 0 && link.get(0) == subSum) {
      link.removeFirst();
      m--;
    }

    int[] buckets = new int[m];
    return partition(link, 0, buckets, subSum);
  }

  public static boolean partition(LinkedList<Integer> link, int index, int[] buckets, int subSum) {
    if (index == link.size()) return true;

    int select = link.get(index);

    for (int i = 0; i < buckets.length; i++) {
      if (i > 0 && buckets[i] == buckets[i - 1]) continue;
      if (select + buckets[i] <= subSum) {
        buckets[i] += select;
        if (partition(link, index + 1, buckets, subSum)) return true;
        buckets[i] -= select;
      }
    }

    return false;
  }
}
```

Python算法源码

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


# 算法入口
def getResult():
    link.sort(reverse=True)

    sumV = 0
    for ele in link:
        sumV += ele

    m = len(link)
    while m > 1:
        if canPartitionMSubsets(link[:], sumV, m):
            return m
        m -= 1

    return -1


```

```python
def canPartitionMSubsets(link, sumV, m):
    if sumV % m != 0:
        return False

    subSum = sumV / m

    if subSum < link[0]:
        return False

    while len(link) > 0 and link[0] == subSum:
        link.pop(0)
        m -= 1

    buckets = [0] * m

    return partition(link, 0, buckets, subSum)


def partition(link, index, buckets, subSum):
    if index == len(link):
        return True

    select = link[index]

    for i in range(len(buckets)):
        if i > 0 and buckets[i] == buckets[i - 1]:
            continue

        if select + buckets[i] <= subSum:
            buckets[i] += select
            if partition(link, index + 1, buckets, subSum):
                return True
            buckets[i] -= select

    return False


# 真后调用
print(getResult())
```