【华为OD机考 统一考试机试C卷】最多购买宝石数目 （C++ Java JavaScript Python C语言）

华为OD机考:统一考试 C卷 + D卷 + B卷 +A卷

目前在考C卷，经过两个月的收集整理，C卷真题已基本整理完毕
抽到原题的概率为2/3到3/3，也就是最少抽到两道原题。请注意：大家刷完C卷真题，最好要把B卷的真题刷一下，因为C卷的部分真题来自B卷。
另外订阅专栏还可以联系笔者开通在线 OJ 进行刷题，提高刷题效率。
真题目录：华为OD机考机试 真题目录（C卷 + D卷 + B卷 + A卷） + 考点说明
专栏：2023华为OD机试(B卷+C卷+D卷)（C++JavaJSPy）
华为OD面试真题精选：华为OD面试真题精选
在线OJ：点击立即刷题，模拟真实机考环境

题目描述

橱窗里有一排宝石，不同的宝石对应不同的价格，宝石的价格标记为 gems[i]
0 ≤ i < n
n = gems.length
宝石可同时出售0个或多个，如果同时出售多个，则要求出售的宝石编号连续；

例如客户最大购买宝石个数为m，购买的宝石编号必须为：gems[i]，gems[i+1]，…，gems[i+m-1]
0 ≤ i < n
m ≤ n
假设你当前拥有总面值为 value 的钱，请问最多能购买到多少个宝石，如无法购买宝石，则返回0。

输入描述

第一行输入n，参数类型为int，取值范围：[0,10^6]，表示橱窗中宝石的总数量。

之后 n 行分别表示从第0个到第n-1个宝石的价格，即 gems[0] 到 gems[n-1] 的价格，类型为int，取值范围：(0,1000]。

之后一行输入v，类型为int，取值范围：[0,10^9]，表示你拥有的钱。

输出描述

输出int类型的返回值，表示最大可购买的宝石数量。

用例1

输入

```
7
8
4
6
3
1
6
7
10
```

输出

```
3
```

说明

> gems = [8,4,6,3,1,6,7], value = 10
> 最多购买的宝石为gems[2]至gems[4]或者gems[3]至gems[5]

用例2

输入

```
0
1
```

输出

```
0
```

说明

> gems = [], value = 1
> 因为没有宝石，所以返回0

用例3

输入

```
9
6
1
3
1
8
9
3
2
4
15
```

输出

```
4
```

说明

> gems = [6, 1, 3, 1, 8, 9, 3, 2, 4], value = 15
> 最多购买的宝石为gems[0]至gems[3]

用例4

输入

```
9
1

```

```
1
1
1
1
1
1
1
1
1
10
```

输出

```
9
```

说明

> gems = [1, 1, 1, 1, 1, 1, 1, 1, 1], value = 10
> 最多购买的宝石为gems[0]至gems[8]，即全部购买

解题思路

以下是详细的解题思路：

1. 首先，代码从控制台读取宝石的数量，每颗宝石的价格，以及你拥有的钱的总面值。
2. 然后，初始化滑动窗口的左右边界（left和right）为0，窗口内宝石的总价（sum），以及最大可购买的宝石数量（max）。
3. 在while循环中，首先将右边界的宝石价格加到总价中。如果总价超过你拥有的钱，那么就将左边界的宝石价格从总价中减去，并将左边界向右移动，直到总价不超过你拥有的钱。
4. 然后，更新最大可购买的宝石数量。这是通过比较当前最大可购买的宝石数量和滑动窗口内的宝石数量（right - left + 1）来实现的。
5. 最后，将右边界向右移动，扩大滑动窗口。当滑动窗口的右边界超过宝石数量时，结束循环。
6. 在while循环结束后，输出最大可购买的宝石数量。

通过这种方式，代码能够找出在给定预算下，最多可以购买的宝石数量。

以例3中的宝石数量是9，宝石的价格分别是6、1、3、1、8、9、3、2、4，你拥有的钱的总面值是15。

以下是详细的计算步骤：

1. 初始化：left=0，right=0，sum=0，max=0。
2. 第一步：将右边界的宝石价格加到总价中，sum=6，right=1，left=0，max=1。
3. 第二步：将右边界的宝石价格加到总价中，sum=7，right=2，left=0，max=2。
4. 第三步：将右边界的宝石价格加到总价中，sum=10，right=3，left=0，max=3。
5. 第四步：将右边界的宝石价格加到总价中，sum=11，right=4，left=0，max=4。
6. 第五步：将右边界的宝石价格加到总价中，sum=19，总价超过了你拥有的钱，所以将左边界的宝石价格从总价中减去，并将左边界向右移动，sum=13，right=4，left=1，max=4。
7. 第六步：将右边界的宝石价格加到总价中，sum=22，总价超过了你拥有的钱，所以将左边界的宝石价格从总价中减去，并将左边界向右移动，sum=21，right=5，left=2，max=4。
8. 第七步：继续将左边界的宝石价格从总价中减去，并将左边界向右移动，sum=18，right=5，left=3，max=4。
9. 第八步：继续将左边界的宝石价格从总价中减去，并将左边界向右移动，sum=17，right=5，left=4，max=4。
10. 第九步：继续将左边界的宝石价格从总价中减去，并将左边界向右移动，sum=9，right=5，left=5，max=4。
11. 第十步：将右边界的宝石价格加到总价中，sum=12，right=6，left=5，max=4。
12. 第十一步：将右边界的宝石价格加到总价中，sum=14，right=7，left=6，max=4。
13. 第十一步：将右边界的宝石价格加到总价中，sum=18，总价超过了你拥有的钱，所以将左边界的宝石价格从总价中减去，并将左边界向右移动，sum=9，right=8，left=7，max=4。
14. 第十四步：滑动窗口的右边界已经超过了宝石数量，结束循环。

因此，最大可购买的宝石数量是4。

0	1	2	3	4	5	6	7	8
6	1	3	1	8	9	3	2	4
6	1	3	1	8	9	3	2	4
6	1	3	1	8	9	3	2	4
6	1	3	1	8	9	3	2	4
6	1	3	1	8	9	3	2	4
6	1	3	1	8	9	3	2	4
6	1	3	1	8	9	3	2	4
6	1	3	1	8	9	3	2	4
6	1	3	1	8	9	3	2	4
6	1	3	1	8	9	3	2	4
6	1	3	1	8	9	3	2	4
6	1	3	1	8	9	3	2	4

C++

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

int main() {
    // 创建一个对象来读取输入
    int n;
    cin >> n;
    // 创建一个数组来存储每个宝石的价格
    vector<int> gems(n);
    // 读取每个宝石的价格
    for (int i = 0; i < n; i++) {
        cin >> gems[i];
    }
    // 读取你拥有的钱的总面值
    int value;
    cin >> value;

    // 初始化滑动窗口的左右边界和窗口内宝石的总价
    int left = 0, right = 0, sum = 0;
    // 初始化最大可购买的宝石数量
```

```
    int maxT = 0;
    // 当滑动窗口的右边界没有超过宝石数量时，继续循环
    while (right < gems.size()) {
        // 将右边界的宝石价格加到总价中
        sum += gems[right];
        // 当总价超过你拥有的钱时，将左边界的宝石价格从总价中减去，并将左边界向右移动
        while (sum > value) {
            sum -= gems[left];
            left++;
        }
        // 更新最大可购买的宝石数量
        maxT = max(maxT, right - left + 1);
        // 将右边界向右移动
        right++;
    }
    // 输出最大可购买的宝石数量
    cout << maxT << endl;

    return 0;
}
```

Java

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        // 创建一个 Scanner 对象来读取输入
        Scanner scanner = new Scanner(System.in);
        // 读取宝石的数量
        int n = scanner.nextInt();
        // 创建一个数组来存储每个宝石的价格
        int[] gems = new int[n];
        // 读取每个宝石的价格
        for (int i = 0; i < n; i++) {
            gems[i] = scanner.nextInt();
        }
        // 读取你拥有的钱的总面值
        int value = scanner.nextInt();

```

```java
        // 初始化滑动窗口的左右边界和窗口内宝石的总价
        int left = 0, right = 0, sum = 0;
        // 初始化最大可购买的宝石数量
        int max = 0;
        // 当滑动窗口的右边界没有超过宝石数量时，继续循环
        while (right < gems.length) {
            // 将右边界的宝石价格加到总价中
            sum += gems[right];
            // 当总价超过你拥有的钱时，将左边界的宝石价格从总价中减去，并将左边界向右移动
            while (sum > value) {
                sum -= gems[left];
                left++;
            }
            // 更新最大可购买的宝石数量
            max = Math.max(max, right - left + 1);
            // 将右边界向右移动
            right++;
        }
        // 输出最大可购买的宝石数量
        System.out.println(max);
    }
}
```

javaScript

```javascript
const readline = require('readline').createInterface({
    input: process.stdin,
    output: process.stdout
});

let input = [];

readline.on('line', (line) => {
    input.push(line);
});

readline.on('close', () => {
    // 读取宝石的数量
    const n = parseInt(input[0]);
    // 创建一个数组来存储每个宝石的价格
```

```
    const gems = input.slice(1, n + 1).map(Number);
    // 读取你拥有的钱的总面值
    const value = parseInt(input[n + 1]);

    // 初始化滑动窗口的左右边界和窗口内宝石的总价
    let left = 0, right = 0, sum = 0;
    // 初始化最大可购买的宝石数量
    let maxT = 0;
    // 当滑动窗口的右边界没有超过宝石数量时，继续循环
    while (right < gems.length) {
        // 将右边界的宝石价格加到总价中
        sum += gems[right];
        // 当总价超过你拥有的钱时，将左边界的宝石价格从总价中减去，并将左边界向右移动
        while (sum > value) {
            sum -= gems[left];
            left++;
        }
        // 更新最大可购买的宝石数量
        maxT = Math.max(maxT, right - left + 1);
        // 将右边界向右移动
        right++;
    }
    // 输出最大可购买的宝石数量
    console.log(maxT);
});
```

Python

```
# 创建一个对象来读取输入
n = int(input())
# 创建一个列表来存储每个宝石的价格

# 创建一个列表来存储每个宝石的价格
gems = []
# 读取每个宝石的价格
for i in range(n):
    gems.append(int(input()))
# 读取你拥有的钱的总面值
value = int(input())

```

```python
# 初始化滑动窗口的左右边界和窗口内宝石的总价
left = 0
right = 0
sum = 0
# 初始化最大可购买的宝石数量
maxT = 0
# 当滑动窗口的右边界没有超过宝石数量时，继续循环
while right < len(gems):
    # 将右边界的宝石价格加到总价中
    sum += gems[right]
    # 当总价超过你拥有的钱时，将左边界的宝石价格从总价中减去，并将左边界向右移动
    while sum > value:
        sum -= gems[left]
        left += 1
    # 更新最大可购买的宝石数量
    maxT = max(maxT, right - left + 1)
    # 将右边界向右移动
    right += 1
# 输出最大可购买的宝石数量
print(maxT)
```

C语言

```c

#include <stdio.h>
#include <stdlib.h>

int main() {
    int n;
    // 读取宝石的总数量
    scanf("%d", &n);

    // 分配一个动态数组来存储每个宝石的价格
    int *gems = (int *)malloc(n * sizeof(int));


    // 读取每个宝石的价格
    for (int i = 0; i < n; i++) {
        scanf("%d", &gems[i]);
    }
```

```
    // 读取用户拥有的钱的总面值
    int value;
    scanf("%d", &value);

    // 初始化滑动窗口的左右边界和窗口内宝石的总价
    int left = 0, right = 0, sum = 0;
    // 初始化最大可购买的宝石数量
    int maxT = 0;

    // 当滑动窗口的右边界没有超过宝石数量时，继续循环
    while (right < n) {
        // 将右边界的宝石价格加到总价中
        sum += gems[right];
        // 当总价超过你拥有的钱时，将左边界的宝石价格从总价中减去，并将左边界向右移动
        while (sum > value) {
            sum -= gems[left];
            left++;
        }
        // 更新最大可购买的宝石数量
        if (right - left + 1 > maxT) {
            maxT = right - left + 1;
        }
        // 将右边界向右移动
        right++;
    }

    // 输出最大可购买的宝石数量
    printf("%d\n", maxT);

    // 释放动态分配的内存
    free(gems);

    return 0;
}
```

完整用例

用例1

0
0

用例2

1
500
1000

用例3

1
1000
500

用例4

3
300
200
500
1000

用例5

3
300
200
500
700

用例6

3
300
200
500
200

用例7

3
300
200
500
100

用例8

5
100
200
300
400
500
900

用例9

5
100
200
300
400
500
600

用例10

5
200
200
200
200
200
1000

文章目录


机考真题
华为OD
No No No
CSDN @算法大师