【华为OD机考 统一考试机试C卷】 用连续自然数之和来表达整数（ C++ Java JavaScript python）

华为OD机考:统一考试 C卷 + D卷 + B卷 + A卷

2023年11月份，华为官方已经将 华为OD机考：OD统一考试（A卷 / B卷）切换到 OD统一考试（C卷）和 OD统一考试（D卷） 。根据考友反馈：目前抽到的试卷为B卷或C卷/D卷，其中C卷居多 ，按照之前的经验C卷部分考题会复用A卷，B卷题，博主正积极从考过的同学收集C卷和D卷真题。 可以先继续刷B卷， C卷和D卷的题目会放在现在大家购买的专栏内，不需要重新购买，请大家放心。

专栏：2023华为OD机试(A卷+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 华为OD机考B卷C卷华为OD机考华为OD机考B卷华为OD机试B卷华为OD机试C卷华为OD机考C卷华为OD机考D卷题目华为OD机考C卷/D卷答案华为OD机考C卷/D卷解析华为OD机考C卷和D卷真题华为OD机考C卷和D卷题解

题目描述：用连续自然数之和来表达整数 （本题分值100）

一个整数可以由连续的自然数之和来表示。

给定一个整数，计算该整数有几种连续自然数之和的表达式，且打印出每种表达式

输入描述

一个目标整数T (1 <=T<= 1000)

输出描述

该整数的所有表达式和表达式的个数。如果有多种表达式，输出要求为：

自然数个数最少的表达式优先输出

每个表达式中按自然数递增的顺序输出，具体的格式参见样例。

在每个测试数据结束时，输出一行”Result:X”，其中X是最终的表达式个数。

ACM输入输出模式

如果你经常使用**Leetcode**,会知道**letcode**是不需要编写输入输出函数的。但是华为OD机考使用的是 **ACM 模式**，需要手动编写输入和输出。

所以最好在牛-客上提前熟悉这种模式。例如C++使用 `cin/cout` ,python使用 `input()/print()` 。JavaScript使用node的 `readline()` 和 `console.log()` 。Java 使用 `sacnner/system.out.print()`

用例1

输入

```
9
```

输出

```
9=9
9=4+5
9=2+3+4
Result:3
```

说明

> 整数 9 有三种表示方法，第1个表达式只有1个自然数，最先输出，
> 第2个表达式有2个自然数，第2次序输出，
> 第3个表达式有3个自然数，最后输出。
> 每个表达式中的自然数都是按递增次序输出的。
> 数字与符号之间无空格

用例2

输入

```
10
```

输出

```
10=10
10=1+2+3+4
Result:2
```

解题思路

解题思路如下：

1. 首先，直接打印出目标整数本身作为一个表达式。
2. 然后，我们需要找出所有可能的连续自然数之和的表达式。我们可以通过枚举起始自然数来实现这一点。对于每一个起始自然数，我们从这个数开始，依次加上后面的自然数，直到总和超过目标整数。
3. 当总和等于目标整数时，我们就找到了一个有效的表达式。我们需要将这个表达式存储下来，以便稍后打印。在存储表达式的同时，我们还需要记录表达式中自然数的个数，因为我们需要按照这个数量对表达式进行排序。
4. 一旦我们找到了所有的表达式，我们就可以对它们进行排序。排序的依据是表达式中自然数的个数，自然数个数少的表达式优先输出。
5. 最后，我们按照排序后的顺序打印出所有的表达式，然后打印出表达式的总数。

这种方法的时间复杂度是O(n^2)，其中n是目标整数。因为我们需要枚举所有可能的起始自然数，并且对于每一个起始自然数，我们可能需要加到目标整数才能确定是否可以形成有效的表达式。在实际应用中，由于目标整数的范围是1到1000，所以这种方法的效率是可以接受的。

C++

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <sstream>

using namespace std;

int main() {
   int target;
cin >> target;
// 输出目标整数T
cout << target << "=" << target << endl;

// 存储所有表达式的 vector
vector<string> expressions;

// 枚举从 1 开始的连续自然数的个数
for (int i = 1; i < target; i++) {
```

```
    int sum = 0;
    stringstream ss;
    // 从第 i 个自然数开始累加
    for (int j = i; sum < target; j++) {
        sum += j;
        ss << j << "+";
        // 找到了一个表达式
        if (sum == target) {
            // 将表达式加入 vector
            expressions.push_back(to_string(target) + "=" + ss.str().substr(0, ss.str().length() - 1));
            break;
        }
    }
}

// 按表达式中自然数的个数排序
sort(expressions.begin(), expressions.end(), [](const string& s1, const string& s2) { return s1.length() < s2.length(); });

// 输出所有表达式
for (const auto& expression : expressions) {
    cout << expression << endl;
}

// 输出表达式的个数
cout << "Result:" << expressions.size() + 1 << endl;

return 0;
    return 0;
}
```

java

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int target = scanner.nextInt();
        // 输出目标整数T
        System.out.println(target + "=" + target);
```

```
        // 存储所有表达式的 vector
        List<String> expressions = new ArrayList<>();

        // 枚举从 1 开始的连续自然数的个数
        for (int i = 1; i < target; i++) {
            int sum = 0;
            StringBuilder sb = new StringBuilder();
            // 从第 i 个自然数开始累加
            for (int j = i; sum < target; j++) {
                sum += j;
                sb.append(j).append("+");
                // 找到了一个表达式
                if (sum == target) {
                    // 将表达式加入 vector
                    expressions.add(target + "=" + sb.substring(0, sb.length() - 1));
                    break;
                }
            }
        }

        // 按表达式中自然数的个数排序
        Collections.sort(expressions, Comparator.comparingInt(String::length));

        // 输出所有表达式
        for (String expression : expressions) {
            System.out.println(expression);
        }

        // 输出表达式的个数
        System.out.println("Result:" + (expressions.size() + 1));
    }
}
```

javaScript

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

```
});

let target;
rl.on('line', (answer) => {
  target = parseInt(answer);
  console.log(target + '=' + target);

  const expressions = [];
  for (let i = 1; i < target; i++) {
    let sum = 0;
    let expression = '';
    for (let j = i; sum < target; j++) {
      sum += j;
      expression += j + '+';
      if (sum === target) {
        expressions.push(target + '=' + expression.slice(0, -1));
        break;
      }
    }
  }

  expressions.sort((s1, s2) => s1.length - s2.length);
  expressions.forEach((expression) => console.log(expression));
  console.log('Result:' + (expressions.length + 1));

  rl.close();
});
```

python

```
import sys

target = int(input())
# 输出目标整数T
print(target, "=", target,sep='')

# 存储所有表达式的 list
expressions = []

```

```python
# 枚举从 1 开始的连续自然数的个数
for i in range(1, target):
    sum = 0
    ss = ""
    # 从第 i 个自然数开始累加
    for j in range(i, target):
        sum += j
        ss += str(j) + "+"
        # 找到了一个表达式
        if sum == target:
            # 将表达式加入 list
            expressions.append(str(target) + "=" + ss[:-1])
            break

# 按表达式中自然数的个数排序
expressions.sort(key=lambda s: len(s))

# 输出所有表达式
for expression in expressions:
    print(expression,sep='')

# 输出表达式的个数
print("Result:", len(expressions) + 1,sep='')
```

C语言

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// 定义一个结构体，用于存储表达式和表达式的长度
typedef struct {
    char expression[1000];
    int length;
} Expression;

// 比较函数，用于 qsort 函数，按照表达式的长度进行排序
int compare(const void *a, const void *b) {
    return ((Expression *)a)->length - ((Expression *)b)->length;
```

```c
}

int main() {
    int target;
    scanf("%d", &target); // 读入目标整数

    printf("%d=%d\n", target, target); // 输出目标整数本身的表达式

    Expression expressions[1000]; // 存储所有表达式的数组
    int count = 0; // 表达式的数量

    // 枚举从 1 开始的连续自然数的个数
    for (int i = 1; i < target; i++) {
        int sum = 0;
        char temp[1000] = ""; // 临时存储每个表达式的字符串

        // 从第 i 个自然数开始累加
        for (int j = i; sum < target; j++) {
            sum += j;
            char num[10];
            sprintf(num, "%d+", j); // 将数字转换为字符串，并添加"+"符号
            strcat(temp, num); // 将数字添加到表达式中

            // 找到了一个表达式
            if (sum == target) {
                temp[strlen(temp) - 1] = '\0'; // 去掉最后的"+"符号
                sprintf(expressions[count].expression, "%d=%s", target, temp); // 将表达式存入数组
                expressions[count].length = j - i + 1; // 记录表达式的长度
                count++;
                break;
            }
        }
    }

    // 按表达式中自然数的个数排序
    qsort(expressions, count, sizeof(Expression), compare);

    // 输出所有表达式
    for (int i = 0; i < count; i++) {
        printf("%s\n", expressions[i].expression);

```

```
    }

    // 输出表达式的个数
    printf("Result:%d\n", count + 1);

    return 0;
}
```

文章目录

机考C卷真题
华为OD