习题课

TEACHING COURSEWARE POWPOINT

授课时间：2025.07.29

目录

01

题目描述

TEACHING
COURSEWARE

TEACH

题目描述

题目描述

给定一个正整数 N，然后将 N 分解成 3 个正整数之和，计算出共有多少种符合要求的分解方法。要求：

1. 分解的 3 个正整数各不相同；
2. 分解的三个正整数中**各个数位**都不含数字 3 和 7。

例如：N 为 8，可分解为 $(1,1,6)$、$(1,2,5)$、$(1,3,4)$、$(2,2,4)$、$(2,3,3)$，其中满足要求的分解方法有 1 种，为 $(1,2,5)$。

输入格式

输入一个正整数 N（$5 < N < 501$），表示要分解的正整数。

输出格式

输出一个整数，表示共有多少种符合要求的分解方法。

输入输出样例

输入 #1	复制	输出 #1	复制
8		1	

02

知识回归

TEACHING
COURSEWARE

TEACH

知识回归

“在计算机科学中，枚举算法（也叫‘穷举法’）是指：明确问题的所有可能解的范围（即‘解空间’），逐一列举所有候选解，对每个候选解验证是否满足问题的条件，最终得到所有有效解的算法。

简单说，就是让计算机‘按顺序把所有可能的情况过一遍，符合要求的就记下来。”

核心思想：

“枚举算法的核心可以浓缩成两个动作：

第一个是‘遍历’：系统地、不重不漏地列出解空间里的所有候选解；

第二个是‘验证’：对每个候选解，用问题的条件检查它是否有效。

03

题目解答

TEACHING
COURSEWARE

TEACH

题目解答

Q：如何避免重复枚举

Q：三重循环的枚举

Q：如何优化

题目解答

注意到数据范围不大（5<N<501），三重循环足矣。

注意到 k 可以直接通过 n-i-j 计算出来，我们可以减少一重循环

我们可以通过确保 i<j<k 来防止重复枚举。

04

标准程序

TEACHING
COURSEWARE

TEACH

标准程序


```cpp
#include<bits/stdc++.h>
using namespace std;
bool check(int x) {
    if (x == 0) {
        return true;
    }
    int n = x;
    while (n > 0) {
        int t = n % 10;  // 取最后一位
        if (t == 3 || t == 7) {
            return false;  // 发现3或7，返回false
        }
        n /= 10;  // 去掉最后一位
    }
    return true;  // 所有位都不是3或7
}
```

标准程序

三重循环

```
int main(){
    int n, ans = 0;
    cin >> n;
    for(int i = 1; i <= n; i++){
        for(int j = i + 1; j <= n; j++){
            for(int k = j + 1; k <= n; k++){
                if(i + j + k == n && check(i) && check(j) && check(k)){
                    ans += 1;
                }
            }
        }
    }
    cout << ans;
    return 0;
}
```

标准程序

二重循环

```cpp
int main() {
    int n;  // 存储输入的目标和
    cin >> n;  // 读取输入值
    int cnt = 0;  // 计数器，用于记录符合条件的三元组数量

    // 双重循环遍历可能的i和j的值
    for (int i = 1; i <= n; i++) {  // 第一个数i，从1开始
        for (int j = i + 1; j <= n; j++) {  // 第二个数j，必须大于i
            int k = n - i - j;  // 根据i和j计算第三个数k，满足i+j+k = n

            // 检查条件：
            // 1. i、j、k都不包含3或7
            // 2. 满足严格递增关系i < j < k
            if (check(i) && check(j) && check(k) && i < j && j < k) {
                cnt++;  // 符合条件则计数器加1
            }
        }
    }

    cout << cnt;  // 输出符合条件的三元组数量
    return 0;
}
```