

RSA加密算法在网络安全世界中无处不在，它利用了极大整数因数分解的困难度。数据越大，安全系数越高，给定一个 32 位正整数，请对其进行因数分解，找出任意两个素数的乘积。

输入描述

一个正整数 num 0 < num <= 2147483647

输出描述

如果成功找到，以单个空格分割，从小到大输出两个素数，分解失败，请输出 -1, -1

用例

输入	15
输出	3 5
输入	27
输出	-1 -1

题目解析

首先，要了解素数概念，及素数判定方法

其它语言解析题目

给定一个 32 位正整数，请对其进行因数分解，找出任意两个素数的乘积。  
15 可以分解为 3 5，由于这两个素数乘积为 15，所以判定为分解成功；  
27 可以分解为 3 3 3，由于不是两个素数的乘积，所以判定为分解失败；

这里，我理解题目想表达的意思。  
给定的正整数，能支持只能分解为两个素数因子，且两个素数因子都为给定的正整数。那么

- 假给定的正整数为素数，则只能分解为 1 和自身，而 1 不是素数，所以判定为分解失败

这可以理解为缩小了给定的正整数的范围只能为素数。

另外如果是一个正整数为两个素数的乘积，比如 11 \* 13 = 143，则必然只能分解为这两个素数，因为这两个素数无法再次分解，所以该正整数就没有其他的素数因子了，所以，一旦我们得到一个可以被正整数整除的素数因子，则另一个因子只能为素数。

Java算法源码

```
1 import java.util.Scanner;
2
3 public class Main {
4     public static void main(String[] args) {
5         Scanner sc = new Scanner(System.in);
6         System.out.println(getResult(sc.nextInt()));
7     }
8 }
9
10 // 素数判定方法
11 public static String getResult(int n) {
12     // 如果 n 为素数，则返回两个素数因子乘积之和
13     if (isPrime(n)) {
14         return "-1 -1";
15     }
16
17     // 遍历所有可能的因子
18     for (int i = 2; i <= Math.sqrt(n); i++) {
19         // 如果 i 能整除 n，则 i 和 n/i 都是素数，返回它们的乘积
20         // 也可以理解为，返回素数因子
21         if (n % i == 0) {
22             // 返回两个素数
23             int j = n / i;
24
25             // 如果 i 和 j 都是素数，则返回它们的乘积之和
26             if (isPrime(i) && isPrime(j)) {
27                 // 返回两个素数之和，即返回两个素数因子之和，因为素数无法再次分解成其他因子，这意味着 n 不再有其他因子了（因子只有 1 和自身）
28                 return i + " " + j + " " + i * j + " " + i * j;
29             } else {
30                 // 如果 i 不是一个素数因子，则返回两个素数因子之和，此时 n 可能不是素数之和
31                 // 如果 i 和 j 都不是素数，则返回两个素数之和，此时 n 可能不是素数之和
32                 // 这时可以判定 n 不是素数了，直接返回 -1
33                 break;
34             }
35         }
36     }
37     return "-1 -1";
38 }
39
40 // 素数判定方法
41 public static boolean isPrime(int n) {
42     if (n <= 1) return n >= 1;
43
44     if (n % 2 == 0 && n % 2 != 2) return false;
45
46     for (int i = 3; i <= Math.sqrt(n); i += 2) {
47         if (n % i == 0) {
48             return false;
49         }
50     }
51     return true;
52 }
53 }
```

JS算法源码

```
1 // 读取输入并处理
2 const readline = require("readline");
3
4 const rl = readline.createInterface({
5     input: process.stdin,
6     output: process.stdout,
7 });
8
9 rl.on("line", (line) => {
10     console.log(getResult(parseInt(line)));
11 });
12
13 // 素数判定方法
14 function isPrime(n) {
15     n = parseInt(n);
16
17     if (n <= 1) {
18         return n >= 1;
19     }
20
21     if (n % 2 == 0 && n % 2 != 2) {
22         return false;
23     }
24
25     for (let i = 3; i <= Math.sqrt(n); i += 2) {
26         if (n % i == 0) {
27             return false;
28         }
29     }
30     return true;
31 }
32
33 // 素数判定方法
34 function getResult(n) {
35     // 如果 n 为素数，则返回两个素数因子乘积之和
36     if (isPrime(n)) {
37         return "-1 -1";
38     }
39
40     // 遍历所有可能的因子
41     for (let i = 2; i <= Math.sqrt(n); i++) {
42         // 如果 i 能整除 n，则 i 和 n/i 都是素数，返回它们的乘积
43         // 也可以理解为，返回素数因子
44         if (n % i == 0) {
45             // 返回两个素数
46             let j = n / i;
47
48             // 如果 i 和 j 都是素数，则返回它们的乘积之和
49             if (isPrime(i) && isPrime(j)) {
50                 // 返回两个素数之和，即返回两个素数因子之和，因为素数无法再次分解成其他因子，这意味着 n 不再有其他因子了（因子只有 1 和自身）
51                 return i + " " + j + " " + i * j + " " + i * j;
52             } else {
53                 // 如果 i 不是一个素数因子，则返回两个素数因子之和，此时 n 可能不是素数之和
54                 // 如果 i 和 j 都不是素数，则返回两个素数之和，此时 n 可能不是素数之和
55                 // 这时可以判定 n 不是素数了，直接返回 -1
56                 break;
57             }
58         }
59     }
60     return "-1 -1";
61 }
62 }
```

Python算法源码

```
1 import math
2
3 # 输入正整数
4 n = int(input())
5
6 # 素数判定
7 def isPrime(n):
8     if n <= 1:
9         return n >= 1
10
11     if n % 2 == 0 and n % 2 != 2:
12         return False
13
14     for i in range(3, int(math.sqrt(n)) + 1, 2):
15         if n % i == 0:
16             return False
17
18     return True
19
20 # 素数判定
21 def getResult(n):
22     # 如果 n 为素数，则返回两个素数因子乘积之和
23     if isPrime(n):
24         return "-1 -1"
25
26     # 遍历所有可能的因子
27     for i in range(2, 0):
28         # 如果 i 能整除 n，则 i 和 n/i 都是素数，返回它们的乘积
29         # 也可以理解为，返回素数因子
30         if n % i == 0:
31             // 返回两个素数
32             j = n // i
33
34             // 如果 i 和 j 都是素数，则返回它们的乘积之和
35             if isPrime(i) && isPrime(j):
36                 // 返回两个素数之和，即返回两个素数因子之和，因为素数无法再次分解成其他因子，这意味着 n 不再有其他因子了（因子只有 1 和自身）
37                 return f"{i} {j} {i * j} {i * j}"
38             else:
39                 // 如果 i 不是一个素数因子，则返回两个素数因子之和，此时 n 可能不是素数之和
40                 // 如果 i 和 j 都不是素数，则返回两个素数之和，此时 n 可能不是素数之和
41                 // 这时可以判定 n 不是素数了，直接返回 -1
42                 break
43
44     return "-1 -1"
45
46 # 输出结果
47 print(getResult(n))
```