题目描述

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，找出是哪两个素数的乘积。

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

其次再来解析题目

> 给定一个 32 位正整数，请对其进行因数分解，找出是哪两个素数的乘积。
> 15 可以被分解为 3 5，由于是这两个素数乘积为15，所以判定为分解成功；
> 27 可以被分解为 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是，
给定的正整数，要支持只能分解为**两个素数因子**，且两个素数乘积要为给定的正整数，那么

- 若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

这可以帮助我们缩小了给定的正整数的范围只能是非素数。

另外如果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

```java
import java.util.Scanner;

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

 // 求解素数之积
 public static String getResult(int n) {
  // 如果n为素数，则必然不可能表两个素数之积
  if (isPrime(n)) {
   return "-1 -1";
  }

  // 质因子为n的因子
  for (int i = 2; i < n; i++) {
   // 当n不能整除i，则i不是n的因子，继续下次循环，效果剪枝
   // 当n可以整除i，则i就是n的因子
   if (n % i == 0) {
    // 记为n的另一因子
    int j = n / i;

    // 只有i,j因子都为素数时，n才是符合题意的素数之积
    if (isPrime(i) && isPrime(j)) {
     // 如果n为两个素数之积，则n只能分解为这两个因子，因为素数无法再次分解出其他因子，也就是说n不再有其他因子了（因子不包含1和自身）
     return i < j ? i + " " + j : j + " " + i;
    } else {
     // 如果i，j为一个不是素数的因子，则说明n存在非素数因子，此时n不可能是素数之积
     // 如果i，j为相等的素数因子，则n不是满足题意的素数之积
     // 这两种可以判定n不符合要求了，直接退出循环
     break;
    }
   }
  }

  return "-1 -1";
 }

 // 判断n是否为素数
 public static boolean isPrime(int n) {
  if (n <= 3) return n > 1;

  if (n % 6 != 1 && n % 6 != 5) return false;

  for (int i = 5; i <= Math.sqrt(n); i += 6) {
   if (n % i == 0 || n % (i + 2) == 0) {
    return false;
   }
  }

  return true;
 }
}
```

JS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) => {
 console.log(getResult(parseInt(line)));
});

/* 素数判定 */
function isPrime(n) {
 n = parseInt(n);

 if (n <= 3) {
  return n > 1;
 }

 if (n % 6 !== 1 && n % 6 !== 5) {
  return false;
 }

 for (let i = 5; i <= Math.sqrt(n); i += 6) {
  if (n % i === 0 || n % (i + 2) === 0) {
   return false;
  }
 }

 return true;
}

/* 素数之积 */
function getResult(n) {
 // 如果n为素数，则必然不可能表两个素数之积
 if (isPrime(n)) {
  return "-1 -1";
 }

 // 质因子为n的因子
 for (let i = 2; i < n; i++) {
  // 当n不能整除i，则i不是n的因子，继续下次循环，效果剪枝
  // 当n可以整除i，则i就是n的因子
  if (n % i === 0) {
   // 记为n的另一因子
   let j = n / i;
   // 只有i,j因子都为素数时，n才是符合题意的素数之积
   if (isPrime(i) && isPrime(j)) {
    // 如果n为两个素数之积，则n只能分解为这两个因子，因为素数无法再次分解出其他因子，也就是说n不再有其他因子了（因子不包含1和自身）
    return i < j ? `${i} ${j}` : `${j} ${i}`;
   } else {
    // 如果i，j为一个不是素数的因子，则说明n存在非素数因子，此时n不可能是素数之积
    // 如果i，j为相等的素数因子，则n不是满足题意的素数之积
    // 这两种可以判定n不符合要求了，直接退出循环
    break;
   }
  }
 }

 return "-1 -1";
}
```

Python算法源码

```python
import math

# 输入获取
n = int(input())


# 素数判断
def isPrime(n):
    if n <= 3:
        return n > 1

    if n % 6 != 1 and n % 6 != 5:
        return False

    for i in range(5, int(math.sqrt(n)) + 1, 6):
        if n % i == 0 or n % (i + 2) == 0:
            return False

    return True


# 素数之积
def getResult(n):
    # 如果n为素数，则必然不可能表两个素数之积
    if isPrime(n):
        return "-1 -1"

    # 质因子为n的因子
    for i in range(2, n):
        # 当n不能整除i，则i不是n的因子，继续下次循环，效果剪枝
        # 当n可以整除i，则i就是n的因子
        if n % i == 0:
            # 记为n的另一因子
            j = n // i
            # 只有i,j因子都为素数时，n才是符合题意的素数之积
            if isPrime(i) and isPrime(j):
                # 如果n为两个素数之积，则n只能分解为这两个因子，因为素数无法再次分解出其他因子，也就是说n不再有其他因子了（因子不包含1和自身）
                return f"{i} {j}" if i < j else f"{j} {i}"
            else:
                # 如果i，j为一个不是素数的因子，则说明n存在非素数因子，此时n不可能是素数之积
                # 如果i，j为相等的素数因子，则n不是满足题意的素数之积
                # 这两种可以判定n不符合要求了，直接退出循环
                break

    return "-1 -1"


# 执行结果
print(getResult(n))
```