题目描述

给你一个整数数组nums，请计算数组的中心位置，数组的中心位置是数组的一个下标，其左侧所有元素相乘的积等于右侧所有元素相乘的积。数组第一个元素的左侧积为1，最后一个元素的右侧积为1。

如果数组有多个中心位置，应该返回最靠近左边的那一个，如果数组不存在中心位置，返回**-1**。

输入描述

输入只有一行，给出N个正整数用空格分隔：nums = 2 5 3 6 5 6

1 <= nums.length <= 1024

1 <= nums[i] <= 10

输出描述

输出：3

解释：中心位置是3

用例

输入	2 5 3 6 5 6
输出	3
说明	无

题目解析

很简单的单指针操作。

应该是考察优化操作，即我们不应该每次都重新计算中心位置左边所有值和右边所有值的各自乘积，应该利用差异剔除的方式。

比如中心位置右移一格到i位置后，左边所有值的乘积left 应该变为了 left *= arr[i-1]，右边所有值得乘积right 应该变为了 right /= arr[i]

JavaScript算法源码

感谢网友m0_71826536提出改进意见，本题中

1 <= nums.length <= 1024

1 <= nums[i] <= 10

也就是说单边乘积最大可以是10^1024，这个值已经远远超过了int,long范围，因此这里我们应该使用大数来处理。

JS可以使用BigInt来处理，需要注意的是BigInt类型数值只能和BigInt类型数值进行计算，否则会报错。而我们可以使用BigInt函数将Number数值转为BigInt类型，或者在字面量后面加n，比如1n

```
/* 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) => {
  const arr = line.split(" ").map(BigInt);

  let left = 1n;
  let right = arr.reduce((p, c) => p * c) / arr[0];

  if (left === right) {
    return console.log(0);
  }

  for (let i = 1; i < arr.length; i++) {
    left *= arr[i - 1];
    right /= arr[i];
```

JavaScript算法源码

感谢网友m0_71826536提出改进意见，本题中

1 <= nums.length <= 1024

1 <= nums[i] <= 10

也就是说单边乘积最大可以是10^1024，这个值已经远远超过了int,long范围，因此这里我们应该使用大数来处理。

JS可以使用BigInt来处理，需要注意的是BigInt类型数值只能和BigInt类型数值进行计算，否则会报错。而我们可以使用BigInt函数将Number数值转为BigInt类型，或者在字面量后面加n，比如1n

```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) => {
  const arr = line.split(" ").map(BigInt);

  let left = 1n;
  let right = arr.reduce((p, c) => p * c) / arr[0];

  if (left === right) {
    return console.log(0);
  }

  for (let i = 1; i < arr.length; i++) {
    left *= arr[i - 1];
    right /= arr[i];

    if (left === right) return console.log(i);
  }

  return console.log(-1);
});
```

Java算法源码

感谢网友m0_71826536提出改进意见，本题中

1 <= nums.length <= 1024

1 <= nums[i] <= 10

也就是说单边乘积最大可以是10^1024，这个值已经远远超过了int,long范围，因此这里我们应该使用大数来处理，Java可以使用BigInteger来处理。

```java
import java.math.BigInteger;
import java.util.Arrays;
import java.util.Scanner;

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

    Integer[] arr =
        Arrays.stream(sc.nextLine().split(" ")).map(Integer::parseInt).toArray(Integer[]::new);

    System.out.println(getResult(arr));
  }

  public static int getResult(Integer[] nums) {
    BigInteger fact = BigInteger.valueOf(1);
    for (Integer num : nums) {
      fact = fact.multiply(BigInteger.valueOf(num));
    }

    BigInteger left = BigInteger.valueOf(1);
    BigInteger right = fact.divide(BigInteger.valueOf(nums[0]));

    if (left.compareTo(right) == 0) {
      return 0;
    }

    for (int i = 1; i < nums.length; i++) {
      left = left.multiply(BigInteger.valueOf(nums[i - 1]));
      right = right.divide(BigInteger.valueOf(nums[i]));

      if (left.compareTo(right) == 0) return i;
    }

    return -1;
  }

  //  public static int getResult(Integer[] nums) {
  //    int fact = 1;
  //    for (Integer num : nums) {
  //      fact *= num;
```

Java算法源码

感谢网友m0_71826536提出改进意见，本题中

1 <= nums.length <= 1024

1 <= nums[i] <= 10

也就是说单边乘积最大可以是10^1024，这个值已经远远超过了int,long范围，因此这里我们应该使用大数来处理，Java可以使用BigInteger来处理。

```java
import java.math.BigInteger;
import java.util.Arrays;
import java.util.Scanner;

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

    Integer[] arr =
        Arrays.stream(sc.nextLine().split(" ")).map(Integer::parseInt).toArray(Integer[]::new);

    System.out.println(getResult(arr));
  }

  public static int getResult(Integer[] nums) {
    BigInteger fact = BigInteger.valueOf(1);
    for (Integer num : nums) {
      fact = fact.multiply(BigInteger.valueOf(num));
    }

    BigInteger left = BigInteger.valueOf(1);
    BigInteger right = fact.divide(BigInteger.valueOf(nums[0]));

    if (left.compareTo(right) == 0) {
      return 0;
    }

    for (int i = 1; i < nums.length; i++) {
      left = left.multiply(BigInteger.valueOf(nums[i - 1]));
      right = right.divide(BigInteger.valueOf(nums[i]));

      if (left.compareTo(right) == 0) return i;
    }

    return -1;
  }

  //  public static int getResult(Integer[] nums) {
  //    int fact = 1;
  //    for (Integer num : nums) {
  //      fact *= num;
  //    }
  //
  //    int left = 1;
  //    int right = fact / nums[0];
  //
  //    if (left == right) {
  //      return 0;
  //    }
  //
  //    for (int i = 1; i < nums.length; i++) {
  //      left *= nums[i - 1];
  //      right /= nums[i];
  //
  //      if (left == right) return i;
  //    }
  //
  //    return -1;
  //  }
}
```

Python算法源码

感谢网友m0_71826536提出改进意见，本题中

1 <= nums.length <= 1024

1 <= nums[i] <= 10

也就是说单边乘积最大可以是10^1024，这个值已经远远超过了int,long范围，因此这里我们应该使用大数来处理。

关于Python处理大数的说明在Python中处理非常大的数字 Dovov编程网

即Python3支持任意大的整数运算。

但是Python3整数相除时得到float小数，而float不支持任意大的数，如果数字过大会溢出，即报错如下：

python OverflowError: integer division result too large for a float_tz_zs的博客-CSDN博客

此时，我们应该使用整除 // 来保证 超大整数相除得到结果也是整数，而不是浮点数，避免溢出。

而本题，超大整数逻辑上是可以整除的。

Python算法源码

感谢网友m0_71826536提出改进意见，本题中

1 <= nums.length <= 1024

1 <= nums[i] <= 10

也就是说单边乘积最大可以是10^1024，这个值已经远远超过了int,long范围，因此这里我们应该使用大数来处理。

关于Python处理大数的说明在Python中处理非常大的数字 Dovov编程网

即Python3支持任意大的整数运算。

但是Python3整数相除时得到float小数，而float不支持任意大的数，如果数字过大会溢出，即报错如下：

python OverflowError: integer division result too large for a float_tz_zs的博客-CSDN博客

此时，我们应该使用整除 // 来保证 超大整数相除得到结果也是整数，而不是浮点数，避免溢出。

而本题，超大整数逻辑上是可以整除的。

```python
# 输入获取
arr = list(map(int, input().split()))


# 算法入口
def getResult(arr):
    left = 1
    right = 1

    for i in arr[1:]:
        right *= i

    if left == right:
        print(0)
        return

    for i in range(1, len(arr)):
        left *= arr[i - 1]
        right //= arr[i]

        if left == right:
            print(i)
            return

    print(-1)


# 算法调用
getResult(arr)
```