题目描述

信号传播过程中会出现一些误码，不同的数字表示不同的误码ID，取值范围为1~65535，用一个数组记录误码出现的情况，
每个误码出现的次数代表误码频度，请找出记录中包含频度最高误码的最小子数组长度。

输入描述

误码总数目：取值范围为0~255，取值为0表示没有误码的情况。
误码出现频率数组：误码ID范围为1~65535，数组长度为1~1000。

输出描述

包含频率最高的误码最小子数组长度

用例

输入	5 1 2 2 4 1
输出	2
说明	频度最高的有1和2，频度是2（出现的次数都是2）。 可以包含频度最高的记录数组是[2 2]和[1 2 2 4 1]， 最短是[2 2]，最小长度为2。

输入	7 1 2 2 4 2 1 1
输出	4
说明	频度最高的是1和2，最短的是[2 2 4 2]

题目解析

简单的排序题。

首先，我们统计出误码数组各个误码的出现过的索引值，假设统计到idxs对象中，属性是误码，属性值是数组，记录误码出现过的索引位置。

然后将idxs对象的所有属性值（各个误码出现过的索引位置数组）拎出来，即Object.values，然后对这些索引位置数组，进行排序，先按照索引位置数组长度进行排序，长度越长，说明频率越高，排序越靠前，如果两个数组长度相同，则看索引跨度，即索引数组的头元素索引和尾元素索引的差距，差距越小，越靠前。这样排序后，得到的首元素数组的首尾索引跨度就是题解。

JavaScript算法源码

```
/* 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,
});

const lines = [];
rl.on("line", (line) => {
  lines.push(line);

  if (lines.length === 2) {
    const total = lines[0] - 0;
    const arr = lines[1].split(" ").map(Number);

    console.log(getResult(total, arr));

    lines.length = 0;
  }
});

/**
```

JavaScript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,
});

const lines = [];
rl.on("line", (line) => {
  lines.push(line);

  if (lines.length === 2) {
    const total = lines[0] - 0;
    const arr = lines[1].split(" ").map(Number);

    console.log(getResult(total, arr));

    lines.length = 0;
  }
});

/**
 *
 * @param {*} total 误码总数目
 * @param {*} arr 误码出现频率数组
 */
function getResult(total, arr) {
  const idxs = {};

  for (let i = 0; i < arr.length; i++) {
    const code = arr[i];
    idxs[code] ? idxs[code].push(i) : (idxs[code] = [i]);
  }

  const countArr = Object.values(idxs).sort((a, b) => {
    if (a.length !== b.length) {
      return b.length - a.length;
    } else {
      const alen = a.at(-1) - a.at(0);
      const blen = b.at(-1) - b.at(0);
      return alen - blen;
    }
  });

  const start = countArr[0].at(0);
  const end = countArr[0].at(-1);
  return end - start + 1;
}
```

Java算法源码

```java
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Scanner;
import java.util.stream.Collectors;

public class Main {
  static ArrayList<Integer[]> nodes;

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

    int n = sc.nextInt();

    int[] arr = new int[n];
    for (int i = 0; i < n; i++) {
      arr[i] = sc.nextInt();
    }

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

  /**
   * @param arr 误码出现频率数组
   * @return 包含频率最高的误码最小子数组长度
   */
  public static int getResult(int[] arr) {
    HashMap<Integer, ArrayList<Integer>> idxs = new HashMap<>();

    for (int i = 0; i < arr.length; i++) {
      Integer code = arr[i];
      idxs.putIfAbsent(code, new ArrayList<>());
      idxs.get(code).add(i);
    }

    ArrayList<Integer> countList =
        idxs.values().stream()
            .sorted(
                (a, b) -> {
```

Java算法源码

```java
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Scanner;
import java.util.stream.Collectors;

public class Main {
  static ArrayList<Integer[]> nodes;

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

    int n = sc.nextInt();

    int[] arr = new int[n];
    for (int i = 0; i < n; i++) {
      arr[i] = sc.nextInt();
    }

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

  /**
   * @param arr 误码出现频率数组
   * @return 包含频率最高的误码最小子数组长度
   */
  public static int getResult(int[] arr) {
    HashMap<Integer, ArrayList<Integer>> idxs = new HashMap<>();

    for (int i = 0; i < arr.length; i++) {
      Integer code = arr[i];
      idxs.putIfAbsent(code, new ArrayList<>());
      idxs.get(code).add(i);
    }

    ArrayList<Integer> countList =
        idxs.values().stream()
            .sorted(
                (a, b) -> {
                  if (a.size() != b.size()) {
                    return b.size() - a.size();
                  } else {
                    int alen = a.get(a.size() - 1) - a.get(0);
                    int blen = b.get(b.size() - 1) - b.get(0);
                    return alen - blen;
                  }
                })
            .limit(1)
            .collect(Collectors.toList())
            .get(0);

    return countList.get(countList.size() - 1) - countList.get(0) + 1;
  }
}
```

Python算法源码

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


# 算法入口
def getResult(total, arr):
    """
    :param total: 误码总数目
    :param arr: 误码出现频率数组
    :return: 包含频率最高的误码最小子数组长度
    """
    idxs = {}

    for i in range(len(arr)):
        code = arr[i]
        if idxs.get(code) is None:
            idxs[code] = [i]
        else:
            idxs[code].append(i)

    countArr = list(idxs.values())

    countArr.sort(key=lambda x: (-len(x), x[-1] - x[0]))

    start = countArr[0][0]
    end = countArr[0][-1]
    return end - start + 1


# 调用算法
print(getResult(total, arr))
```

Python算法源码

复制

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


# 算法入口
def getResult(total, arr):
    """
    :param total: 误码总数目
    :param arr: 误码出现频率数组
    :return: 包含频率最高的误码最小子数组长度
    """
    idxs = {}

    for i in range(len(arr)):
        code = arr[i]
        if idxs.get(code) is None:
            idxs[code] = [i]
        else:
            idxs[code].append(i)

    countArr = list(idxs.values())

    countArr.sort(key=lambda x: (-len(x), x[-1] - x[0]))

    start = countArr[0][0]
    end = countArr[0][-1]
    return end - start + 1


# 调用算法
print(getResult(total, arr))
```