### 算法实现题 众数问题

问题描述:

给定含有n个元素的多重集合S,每个元素在S中出现的次数称为该元素的重数。多重集S中重数最大的元素称为众数。例如,S={1,2,2,2,3,5}。多重集S的众数是2,其重数为3。

算法设计:

对于给定的由 n 个自然数组成的多重集S,编程计算 S的众数及其重数。

数据输入:

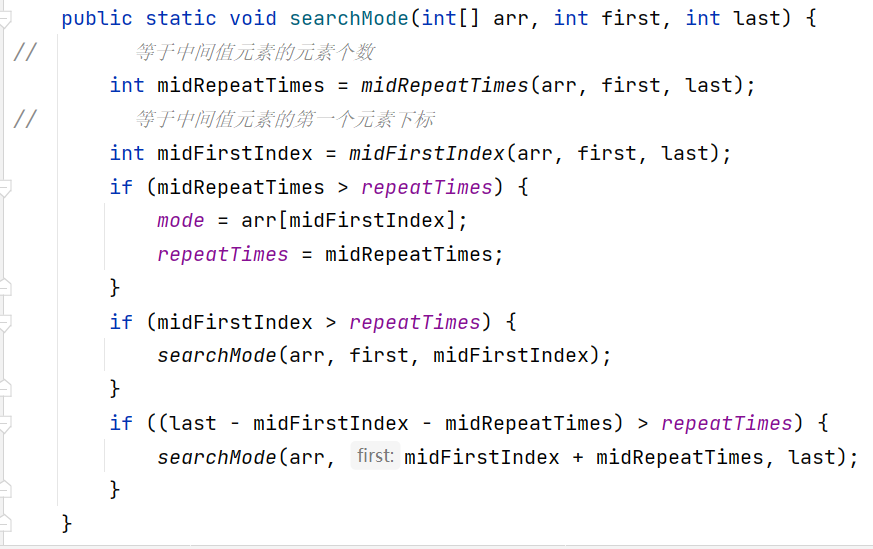
输人数据由文件名为input txt的文本文件提供。

文件的第1行为多重集 S中元素个数n;在接下来的n行中,每行有一个自然数。

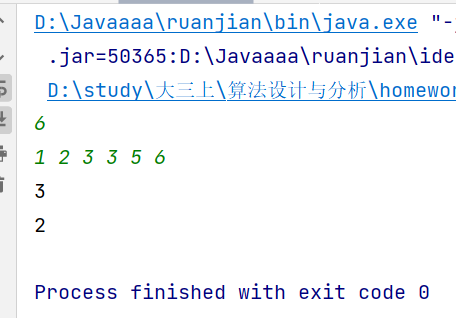
结果输出:

程序运行结束时,将计算结果输出到文件output.txt中。输出文件有2行,第1行是众数。第2行是重数。

解答：核心代码：



运行结果：



### 算法实现题 半数集问题

问题描述:

给定一个自然数n,由n开始可以依次产生半数集 set(n)中的数如下:

(1)n属于set(n);

(2)在n的左边加上一个自然数,但该自然数不能超过最近添加的数的一半;

(3)按此规则进行处理,直到不能再添加自然数为止。

例如,set(6)={6,16,26,126,36,136}。半数集set(6)中有6个元素。注意,该半数集是多重集。

算法设计:

对于给定的自然数n,编程计算半数集 set(m)中的元素个数。

数据输入:

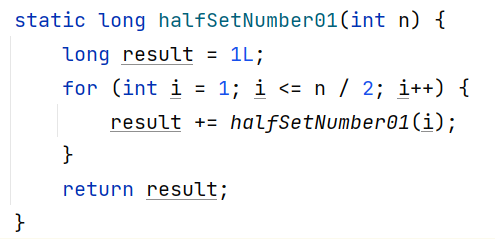
输入数据由文件名为input.txt的文本文件提供。每个文件只有一行,给出整数n(0<n<1000)。

结果输出:

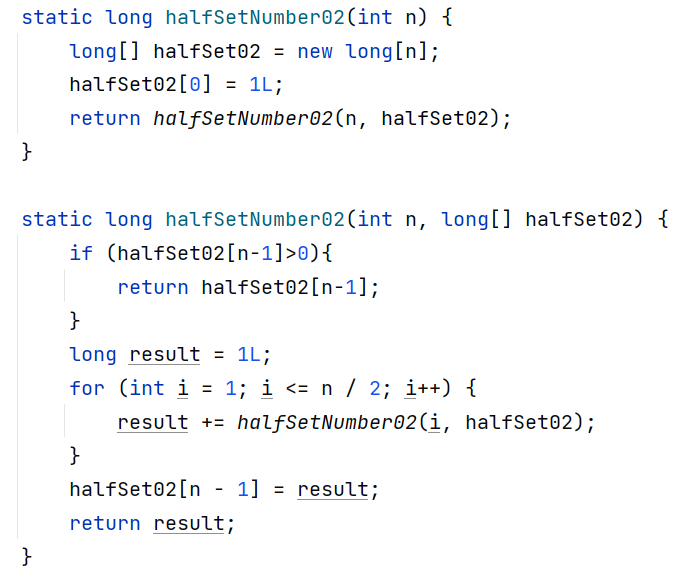
程序运行结束时,将计算结果输出到文件output.txt中。输出文件只有一行,给出半数集 set(n)中的元素个数。

解答：核心代码：

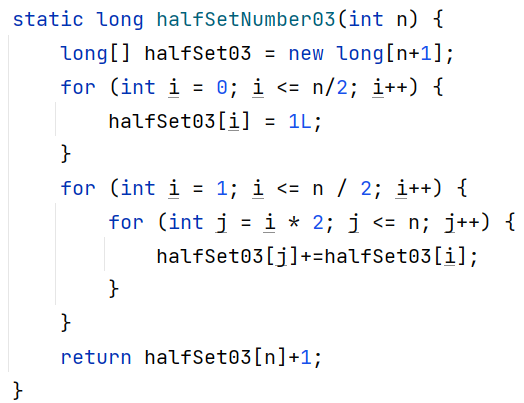
直接递归：



数组存储中间数据：



迭代：



运行结果：



代码附录：

众数：

import java.util.Scanner;

public class HW0102 {

public static int mode; //众数

public static int repeatTimes; //重数

//找到等于中间值元素的元素个数

public static int midRepeatTimes(int[] arr, int left, int right) {

int sum = 0;

int mid = (left + right) >>> 1;

int n = arr[mid];

for (int i = left; i < right; i++) {

if (arr[i] == n) {

sum++;

}

}

return sum;

}

//找出等于中间值元素的第一个元素下标

public static int midFirstIndex(int[] arr, int left, int right) {

int x = 0;

int mid = (left + right) >>> 1;

for (int i = left; i < right; i++) {

if (arr[mid] == arr[i]) {

x = i;

break;

}

}

return x;

}

public static void searchMode(int[] arr, int first, int last) {

// 等于中间值元素的元素个数

int midRepeatTimes = midRepeatTimes(arr, first, last);

// 等于中间值元素的第一个元素下标

int midFirstIndex = midFirstIndex(arr, first, last);

if (midRepeatTimes > repeatTimes) {

mode = arr[midFirstIndex];

repeatTimes = midRepeatTimes;

}

if (midFirstIndex > repeatTimes) {

searchMode(arr, first, midFirstIndex);

}

if ((last - midFirstIndex - midRepeatTimes) > repeatTimes) {

searchMode(arr, midFirstIndex + midRepeatTimes, last);

}

}

public static void main(String[] args) {

Scanner input = new Scanner(System.in);

int n = input.nextInt();

int[] arr = new int[n];

for (int i = 0; i < arr.length; i++) {

arr[i] = input.nextInt();

}

searchMode(arr, 0, n);

System.out.println(mode);

System.out.println(repeatTimes);

}

}

半数集：

import java.util.Scanner;

public class HalfSet {

public static Long[] halfSet02 = new Long[10000];

int k = 0;

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int n = scanner.nextInt();

long startTime = System.currentTimeMillis();

long endTime = System.currentTimeMillis();

while (n != 0) {

startTime = System.currentTimeMillis(); //获取开始时间

System.out.print(halfSetNumber01(n));

endTime = System.currentTimeMillis(); //获取结束时间

System.out.println(" 程序运行时间：" + (endTime - startTime) + "ms"); //输出程序运行时间

startTime = System.currentTimeMillis();

System.out.print(halfSetNumber02(n));

endTime = System.currentTimeMillis();

System.out.println(" 程序运行时间：" + (endTime - startTime) + "ms");

startTime = System.currentTimeMillis();

System.out.print(halfSetNumber03(n));

endTime = System.currentTimeMillis();

System.out.println(" 程序运行时间：" + (endTime - startTime) + "ms");

n=scanner.nextInt();

}

}

static long halfSetNumber01(int n) {

long result = 1L;

for (int i = 1; i <= n / 2; i++) {

result += halfSetNumber01(i);

}

return result;

}

static long halfSetNumber02(int n) {

long[] halfSet02 = new long[n];

halfSet02[0] = 1L;

return halfSetNumber02(n, halfSet02);

}

static long halfSetNumber02(int n, long[] halfSet02) {

if (halfSet02[n-1]>0){

return halfSet02[n-1];

}

long result = 1L;

for (int i = 1; i <= n / 2; i++) {

result += halfSetNumber01(i);

}

halfSet02[n - 1] = result;

return result;

}

static long halfSetNumber03(int n) {

long[] halfSet03 = new long[n+1];

for (int i = 0; i <= n/2; i++) {

halfSet03[i] = 1L;

}

for (int i = 1; i <= n / 2; i++) {

for (int j = i \* 2; j <= n; j++) {

halfSet03[j]+=halfSet03[i];

}

}

return halfSet03[n]+1;

}

}