САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

Факультет: Программной инженерии и компьютерной техники
Направление подготовки: 09.03.04 (Программная инженерия)

Дисциплина «Алгоритмы и структуры данных»

Отчёт по лабораторной работе

Курс на Stepik

Группа: P3217
Выполнил: Минин Александр

г. Санкт-Петербург
2019г.

Задача на программирование: небольшое число Фибоначчи

Дано целое число $1 \le n \le 40$, необходимо вычислить n-е число Фибоначчи (напомним, что $F_0 = 0$, $F_1 = 1$ и $F_n = F_{n-1} + F_{n-2}$ при $n \ge 2$).

```
class Main {
    public static void main(String[] args) throws IOException {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        int n = Integer.valueOf(br.readLine());
        int[] f = new int[41];
        f[0] = 0;
        f[1] = 1;
        for (int i = 2; i < n + 1; i++) {
            f[i] = f[i - 1] + f[i - 2];
        }
        OutputStream os = new BufferedOutputStream(System.out);
        os.write((String.valueOf(f[n]).getBytes()));
        os.flush();
    }
}
```

Задача на программирование: последняя цифра большого числа Фибоначчи

Дано число $1 \le n \le 10^7$, необходимо найти последнюю цифру n-го числа Фибоначчи.

Как мы помним, числа Фибоначчи растут очень быстро, поэтому при их вычислении нужно быть аккуратным с переполнением. В данной задаче, впрочем, этой проблемы можно избежать, поскольку нас интересует только последняя цифра числа Фибоначчи: если $0 \le a, b \le 9$ — последние цифры чисел F_i и F_{i+1} соответственно, то $(a + b) \bmod 10$ — последняя цифра числа F_{i+2}.

```
import java.io.*;

class Main {
    public static void main(String[] args) throws IOException {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        int n = Integer.valueOf(br.readLine());
        int f[] = {0, 1};
        for (int i = 2; i < n + 1; i++) {
            f[i % 2] = (f[0] + f[1]) % 10;
        }
        OutputStream os = new BufferedOutputStream(System.out);
        os.write((String.valueOf(f[n % 2]).getBytes()));
        os.flush();
    }
```

```
}
```

Задача на программирование повышенной сложности: огромное число Фибоначчи по модулю

Даны целые числа $1 \le n \le 10^{18}$ и $2 \le m \le 10^5$, необходимо найти остаток от деления n-го числа Фибоначчи на m.

```
import java.io.*;
import java.util.ArrayList;
class Main {
    public static void main(String[] args) throws IOException {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        String[] line = br.readLine().split(" ");
        long time = System.currentTimeMillis();
        long n = Long.valueOf(line[0]);
        int m = Integer.valueOf(line[1]);
        OutputStream os = new BufferedOutputStream(System.out);
        os.write(getFibonacciRest(n, m).toString().getBytes());
        os.flush();
    }

    private static Long getFibonacciRest(long n, long m) {
        ArrayList<Long> s = getSequencePeriod(m);
        long period = s.size() - 2;
        int val = (int) (n % period);
        return s.get(val);
    }

    private static ArrayList<Long> getSequencePeriod(long m) {
        ArrayList<Long> s = new ArrayList();
        s.add(0l);
        s.add(1l);
        for (int i = 2; i < m * 6; i++) {
            s.add((s.get(i - 1) + s.get(i - 2)) % m);
            if (s.get(i) == 1 && s.get(i - 1) == 0) {
                break;
            }
        }
        return s;
    }
}
```

Задача на программирование: наибольший общий делитель

По данным двум числам $1 \le a, b \le 2 \cdot 10^9$ найдите их наибольший общий делитель.

```
import java.io.*;
```

```
import java.math.BigInteger;
import java.util.ArrayList;
class Main {
    public static void main(String[] args) throws IOException {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        String[] line = br.readLine().split(" ");
        BigInteger a = new BigInteger(line[0]);
        BigInteger b = new BigInteger(line[1]);
        OutputStream os = new BufferedOutputStream(System.out);
        os.write(a.gcd(b).toString().getBytes());
        os.flush();
    }
}
```

Задача на программирование: декодирование Хаффмана

Восстановите строку по её коду и беспрефиксному коду символов.

В первой строке входного файла заданы два целых числа k и l через пробел — количество различных букв, встречающихся в строке, и размер получившейся закодированной строки, соответственно. В следующих k строках записаны коды букв в формате "letter: code". Ни один код не является префиксом другого. Буквы могут быть перечислены в любом порядке. В качестве букв могут встречаться лишь строчные буквы латинского алфавита; каждая из этих букв встречается в строке хотя бы один раз. Наконец, в последней строке записана закодированная строка. Исходная строка и коды всех букв непусты. Заданный код таков, что закодированная строка имеет минимальный возможный размер.

В первой строке выходного файла выведите строку s. Она должна состоять из строчных букв латинского алфавита. Гарантируется, что длина правильного ответа не превосходит 10^4 символов.

```
import java.io.*;
import java.util.HashMap;
import java.util.Map;
class Main {
  public static void main(String[] args) throws IOException {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in), 8192);
        OutputStream os = new BufferedOutputStream(System.out);

        String[] metaData = br.readLine().split(" ");
        int k = Integer.valueOf(metaData[0]);
        int l = Integer.valueOf(metaData[1]);

        Map<String, String> letterMap = new HashMap<>();

        String[] line;
        for (int i = 0; i < k; i++) {
            line = br.readLine().split(": ");
            letterMap.put(line[1], line[0]);
        }

        String encodedMessage = br.readLine();
        StringBuilder decodedMessageBuilder = new StringBuilder(1024);
        int i = 0;
        while (i < l) {
```

```
            int j = i + 1;
            while (!letterMap.containsKey(encodedMessage.substring(i, j))) {
                j++;
            }
            decodedMessageBuilder.append(letterMap.get(encodedMessage.substring(i, j)));
            i = j;
        }
        os.write(decodedMessageBuilder.toString().getBytes());
        os.flush();
        os.close();
    }
}
```

Задача на программирование: очередь с приоритетами

Первая строка входа содержит число операций $1 \le n \le 10^5$. Каждая из последующих n строк задают операцию одного из следующих двух типов:

- `Insert x`, где $0 \le x \le 10^9$ — целое число;
- `ExtractMax`.

Первая операция добавляет число x в очередь с приоритетами, вторая — извлекает максимальное число и выводит его.

```
import java.io.*;
import java.math.BigDecimal;
import java.math.BigInteger;
import java.math.RoundingMode;

class Main {

    public static void main(String[] args) throws IOException {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        OutputStream os = new BufferedOutputStream(System.out);

        int n  = Integer.valueOf(br.readLine());
        Heap<HeapNode<Integer>> heap = new Heap((int) Math.pow(10, 5) + 1);
        for (int i = 0; i < n; i++) {
            String line = br.readLine();
            if (line.startsWith("Insert")) {
                int el = Integer.parseInt(line.split(" ")[1]);
                heap.insert(new HeapNode<>(null, el));
            } else {
                // extract
                os.write(String.valueOf(heap.getMin().key + "\n").getBytes());
                heap.removeMin();

            }
        }
        os.flush();
    }

    private interface Keyable {

        public int getKey();
```

```
    public void setKey(int key);

    public int getPos();

    public void setPos(int pos);

}

public static class HeapNode<T> implements Keyable {
    private T obj;
    private int key;
    private int pos;

    HeapNode(T obj, int key) {
        this.obj = obj;
        this.key = key;
    }

    private T getObject() {
        return obj;
    }

    public int getKey() {
        return key;
    }

    public void setKey(int key) {
        this.key = key;
    }

    public int getPos() {
        return pos;
    }

    public void setPos(int pos) {
        this.pos = pos;
    }

    public void setObj(T obj) {
        this.obj = obj;
    }

}

public static class Heap<T extends Keyable> {
    private Object[] h;
    private int heapSize = 0;

    public Heap(int capacity) {
        h = new Object[capacity];
    }

    /**
     * Constructs a HEAP from the array O(N)
     *
     * @param array
     */
    public Heap(T[] array) {
        h = new Object[array.length];
        heapSize = array.length;
```

```
    for (int i = 0; i < array.length; i++) {
        h[i] = array[i];
    }
    for (int i = heapSize / 2; i >= 0; i--) {
        siftDown(i);
    }
}

public void insert(T t) {
    t.setPos(heapSize);
    h[heapSize] = t;
    heapSize++;
    siftUp(heapSize - 1);
}

public T getMin() {
    return get(0);
}

public void removeMin() {
    swap(0, --heapSize);
    siftDown(0);
}

private void siftUp(int i) {
    while (i > 0 && get(i).getKey() > get((i - 1) / 2).getKey()) {
        swap(i, (i - 1) / 2);
        i = (i - 1) / 2;
    }
}

private void siftDown(int i) {
    while (2 * i + 1 < heapSize) {
        int j = 2 * i + 1;
        if (j + 1 < heapSize && get(j).getKey() < get(2 * i + 2).getKey()) {
            j += 1;
        }
        if (get(j).getKey() > get(i).getKey()) {
            swap(i, j);
            i = j;
        } else {
            break;
        }
    }
}

private T get(int i) {
    return (T) h[i];
}


private void swap(int i, int j) {
    get(i).setPos(j);
    get(j).setPos(i);

    T tmp = get(i);
    h[i] = get(j);
    h[j] = tmp;
}
```

```
    public void changeKey(int i, int newKey) {
        get(i).setKey(newKey);
        siftDown(i);
        siftUp(i);
    }

    public void print() {
        BigDecimal dec = new BigDecimal(Math.log(heapSize) / Math.log(2));
        dec = dec.setScale(1, RoundingMode.UP);
        int levels = dec.intValue();
        int lastLevel = (int) Math.pow(2, levels - 1);
        System.out.println("*** HEAP *** SIZE(" + heapSize + ")");
        for (int i = 0, k = 0, lev = 0; i <= (heapSize - 1); i++) {
            if (lev == 4) {
                System.out.println(" ");
            } else {
                for (int l = 0; l < (2 * lastLevel - 1) / Math.pow(2, lev); l++) {
                    System.out.print(" ");
                }
            }
            System.out.print(get(i).getKey());
            if (i == k) {
                k = 2 * k + 2;
                lev += 1;
                System.out.println();
            }
        }
    }
}
```

Задача на программирование: двоичный поиск

В первой строке даны целое число $1 \le n \le 10^5$ и массив $A[1 \dots n]$ из n различных натуральных чисел, не превышающих 10^9, в порядке возрастания, во второй — целое число $1 \le k \le 10^5$ и k натуральных чисел $b_1, \dots, b_k$, не превышающих 10^9. Для каждого i от 1 до k необходимо вывести индекс $1 \le j \le n$, для которого $A[j] = b_i$, или -1, если такого j нет.

```
import java.io.*;
import java.util.Arrays;
class Main {
  public static void main(String[] args) throws IOException {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in), 8192);
        OutputStream os = new BufferedOutputStream(System.out);

        int[] a = Arrays.stream(br.readLine().split(" ")).mapToInt(s -> Integer.parseInt(s)).toArray();
        int[] k = Arrays.stream(br.readLine().split(" ")).mapToInt(s -> Integer.parseInt(s)).toArray();

        for (int i = 1; i < k.length; i++) {
            os.write((binarySearch(k[i], a) + " ").getBytes());
        }

        br.close();
        os.flush();
```

```
        os.close();
    }

    public static int binarySearch(int x, int[] arr) {
        int ind = binarySearchLeftIterative(x, arr);
        if (ind == arr.length || arr[ind] != x) {
            return -1;
        } else {
            return ind;
        }
    }

    public static int binarySearchLeftIterative(int x, int[] arr) {
        int l = 1; // cuz first number in array - is n of elements!!!(((
        int r = arr.length;
        while (l < r) {
            int mid = l + (r - l) / 2;
            if (arr[mid] < x) {
                l = mid + 1;
            } else {
                r = mid;
            }
        }
        return r;
    }
}
```

Задача на программирование: число инверсий

Первая строка содержит число $1 \le n \le 10^5$, вторая — массив $A[1 \dots n]$, содержащий натуральные числа, не превосходящие 10^9. Необходимо посчитать число пар индексов $1 \le i < j \le n$, для которых $A[i] > A[j]$. (Такая пара элементов называется инверсией массива. Количество инверсий в массиве является в некотором смысле его мерой неупорядоченности: например, в упорядоченном по неубыванию массиве инверсий нет вообще, а в массиве, упорядоченном по убыванию, инверсию образуют каждые два элемента.)

```
import java.io.*;
import java.util.Scanner;
class Main {
     private static long inversionsNumber = 0;

    public static void main(String[] args) throws IOException {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in), 8192);
        OutputStream os = new BufferedOutputStream(System.out);

        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int a[] = new int[n];
        for (int i = 0; i < n; i++) {
            a[i] = sc.nextInt();
        }

        mergeSort(a, 0, a.length);
```

```
        os.write((inversionsNumber + "\n").getBytes());
        br.close();
        os.flush();
        os.close();
    }

    // l1 < l2; r1 < r2
    public static void merge(int[] array, int l1, int r1, int l2, int r2) throws IOException {
        int i = 0;
        int size = r2 - l1;
        int destStart = l1;
        int[] tmp = new int[size];
        while (i < size) {
            while (l1 < r1 && (l2 >= r2 || array[l1] <= array[l2])) { // stable
                tmp[i++] = array[l1];
                l1++;
            }
            while (l2 < r2 && (l1 >= r1 || array[l2] < array[l1])) {
                tmp[i++] = array[l2];
                inversionsNumber += r1 - l1;
                l2++;
            }
        }

        for (int k = 0; k < size; k++) {
            array[destStart + k] = tmp[k];
        }
    }

    public static void mergeSort(int[] array, int l, int r) throws IOException {
        if (l == r - 1) {
            return;
        }
        mergeSort(array, l, l + (r - l) / 2);
        mergeSort(array, l + (r - l) / 2, r);
        merge(array, l, l + (r - l) / 2, l + (r - l) / 2, r);
    }
}
```

Задача на программирование: сортировка подсчётом

Первая строка содержит число $1 \le n \le 10^4$, вторая — n натуральных чисел, не превышающих 10. Выведите упорядоченную по неубыванию последовательность этих чисел.

```
import java.io.*;
import java.util.Arrays;
import java.util.Scanner;
class Main {
    public static void main(String[] args) throws IOException {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in), 8192);
        OutputStream os = new BufferedOutputStream(System.out);

        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
```

```
        int[] count = new int[11];
        for (int i = 0; i < n; i++) {
            count[sc.nextInt()]++;
        }
        for (int i = 0; i < count.length; i++) {
            for (int k = 0; k < count[i]; k++) {
                os.write((i + " ").getBytes());
            }
        }

        br.close();
        os.flush();
        os.close();
    }


}
```

Задача на программирование: наибольшая последовательнократная подпоследовательность

Дано целое число $1 \le n \le 10^3$ и массив $A[1 \dots n]$ натуральных чисел, не превосходящих $2 \cdot 10^9$. Выведите максимальное $1 \le k \le n$, для которого найдётся подпоследовательность $1 \le i_1 < i_2 < \dots < i_k \le n$ длины k, в которой каждый элемент делится на предыдущий (формально: для всех $1 \le j < k$, $A[i_j] \mid A[i_{j+1}]$).

```
import java.io.*;
import java.util.Arrays;
class Main {
    public static void main(String[] args) throws IOException {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in), 8192);
        OutputStream os = new BufferedOutputStream(System.out);

        int n = Integer.valueOf(br.readLine());
        int[] a = Arrays.stream(br.readLine().split(" ")).mapToInt(s -> Integer.parseInt(s)).toArray();
        int[] d = new int[n];

        for (int i = 0; i < n; i++) {
            d[i] = 1;
            for (int j = 0; j < i; j++) {
                if (a[i] % a[j] == 0) {
                    d[i] = Integer.max(d[i], 1 + d[j]);
                }
            }
        }

        os.write((String.valueOf(Arrays.stream(d).max().getAsInt()) + "\n").getBytes());

        br.close();
        os.flush();
        os.close();
    }

}
```

Задача на программирование повышенной сложности: наибольшая невозрастающая подпоследовательность

Дано целое число $1 \le n \le 10^5$ и массив $A[1 \dots n]$, содержащий неотрицательные целые числа, не превосходящие 10^9. Найдите наибольшую невозрастающую подпоследовательность в A. В первой строке выведите её длину k, во второй — её индексы $1 \le i_1 < i_2 < \dots < i_k \le n$ (таким образом, $A[i_1] \ge A[i_2] \ge \dots \ge A[i_n]$).

```
import java.io.*;
import java.util.Arrays;
import java.util.stream.Collectors;
class Main {
  public static void main(String[] args) throws IOException {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in), 8192);
        OutputStream os = new BufferedOutputStream(System.out);

        int n = Integer.valueOf(br.readLine());
        int[] arr = Arrays.stream(br.readLine().split(" ")).mapToInt(s -> Integer.parseInt(s)).toArray();
        int[] tail = new int[n];
        int[] previous = new int[n];
        int sequenceLength = 0;
        for (int i = 0; i < n; i++) {
            int pos = binarySearchRight(arr, tail, sequenceLength, arr[i]);
            if (pos == sequenceLength) {
                sequenceLength++;
            }
            previous[i] = pos > 0 ? tail[pos - 1] : -1;
            tail[pos] = i;
        }

        os.write((String.valueOf(sequenceLength) + "\n").getBytes());

        int[] result = new int[sequenceLength];
        for (int i = tail[sequenceLength - 1]; i >= 0; i = previous[i]) {
            result[--sequenceLength] = i + 1;
        }

        for (int i = 0; i < result.length; i++) {
            os.write(String.valueOf(result[i] + " ").getBytes());
        }

        br.close();
        os.flush();
        os.close();
    }

    static int binarySearchRight(int[] a, int[] tail, int sequenceLength, int key) {
        int l = -1;
        int r = sequenceLength;
        while (l < r - 1) {
            int mid = (l + r) >>> 1;
            if (a[tail[mid]] >= key) {
```

```
                l = mid;
            } else {
                r = mid;
            }
        }
        return r;
    }

}
```

Задача на программирование: различные слагаемые

По данному числу $1 \le n \le 10^9$ найдите максимальное число k, для которого n можно представить как сумму k различных натуральных слагаемых. Выведите в первой строке число k, во второй — k слагаемых.

```
import java.io.*;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;
class Main {
       public static void main(String[] args) throws IOException {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in), 8192);
        OutputStream os = new BufferedOutputStream(System.out);

        int n = Integer.valueOf(br.readLine());
        ArrayList<Integer> addends = new ArrayList<>(1024);
        int i = 0;
        while (true) {
            i++;
            if (i * 2 < n) {
                addends.add(i);
                n = n - i;
            } else {
                addends.add(n);
                break;
            }
        }

        os.write((addends.size() + "\n").getBytes());
        for (int k = 0; k < addends.size(); k++) {
            os.write((addends.get(k) + " ").getBytes());
        }

        os.flush();
        os.close();
    }
}
```