МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий

Кафедра информатики и систем управления

Разреженные матрицы
Отчет по лабораторной работе
Вариант № 17

по дисциплине

Алгоритмы и структуры данных

РУКОВОДИТЕЛЬ:

_____ Капранов С.Н.

СТУДЕНТ:

_____ Сапожников В.О.

19-ИВТ-3

Работа защищена «___» _____

С оценкой _____

Нижний Новгород 2020

Текст задания

Дана разреженная ленточная матрица. Найти сумму её элементов.

Используемый язык программирования: Java

Текст программы

Main.java

```
import java.io.File;                        //класс оболочка для
                                            //представление пути к
                                            //файлу/директории

import java.io.FileNotFoundException;       //класс - исключение,
                                            //сообщает о
                                            //невозможности открытия
                                            //файла

import java.util.Scanner;                   //класс отвечающий за
                                           //чтение с потока ввода

/**
 * Класс, содержащий точку входа в программу - метод main.
 * Язык: java
 *
 * Реализация второй лабораторной работы по дисциплине:
 * Алгоритмы и структуры данных. Вариант№17.
 *
 * Текст задания: Дана разреженная ленточная матрица. Найти
 * сумму её элементов.
 *
 * @release: 27.06.20
 * @last_update: 27.06.20
 *
 * @author Vladislav Sapozhnikov 19-IVT-3
 */
public class Main
{
    public static void main (String[] args)
    {
        try
        {
```

```
        //создание объекта класса, предназначенного
        //для хранения разреженной ленточной матрицы
        BandSpareMatrix spareMatrix = new BandSpareMatrix();


         //буфер, куда помещается прочитанная с файла
         //матрица
        Integer[][] initMatrix =readMatrixFromFile(args[0]);

        System.out.println("\nИсходная матрица: ");
        for (Integer[] i: initMatrix)
        {
            for (Integer j: i)
            {
                System.out.print(j + " ");
            }
            System.out.println();
        }

        System.out.println("\nОбработанная матрица (ленточно
                             + строчный формат): ");

        //перезапись прочитанной матрицы
        //в ленточно строчный формат
        spareMatrix.fillMatrix(initMatrix);

        //вывод матрицы в ленточно строчном формате
        spareMatrix.printMatrix();

        //вывод суммы элементов матрицы
        System.out.println();

        System.out.println("Сумма элементов матрицы: " +
                          +spareMatrix.getSumMatrixElement());
    }
    catch (FileNotFoundException e)
    {
        System.err.println("Нет такого файла: " + args[0]);
        System.err.println("Укажите путь к существующему +
                             + файлу и повторите попытку.");
    }
    catch (NumberFormatException e)
    {
```

```
            System.err.println("Неверное содержимое файла.");
            System.err.println("Измените переданный файл или +
                               + его содержимое и повторите +
                               + попытку.");
        }
    }

    /**
     * Метод, считавыющий матрицу с файла.
     *
     * @param path                    - путь к файлу из которого
     *                                  следует читать матрицу.
     *
     * @throws FileNotFoundException - исключение, уведомляющие
     *                                  о невозможности открытия
     *                                  файла.
     *                                 Данное исключение не
     *                                  обработается внутри
     *                                  метода, а делегируется
     *                                  методу выше.
     *
     *
     * @return initMatrix             - метод возвращает
     *                                  двумерный массив
     *                                  ссылочного типа Integer.
     * */
    public static Integer[][] readMatrixFromFile (String path)
throws FileNotFoundException
    {
        SinglyLinkedList<String[]> strList = readFile(path);
        int size = strList.getSize();

        Integer[][] initMatrix = new Integer[size][size];

        for (int i = 0; i < size; i++)
        {
            for (int j = 0; j < size; j++)
            {
                initMatrix[i][j] =
Integer.parseInt(strList.get(i)[j]);
            }
        }
        return initMatrix;}
```

```
    /**
     * Метод считавающий файл посторочно. Дополнение метода
     * readMatrixFromFile.
     *
     * @param path                    - путь к файлу из которого
     *                                  следует читать матрицу.
     *
     * @throws FileNotFoundException - исключение, уведомляющие
     *                                  о невозможности открытия
     *                                  файла.
     *                                 Данное исключение не
     *                                  обработается внутри
     *                                  метода, а делегируется
     *                                  методу выше.
     *
     * @return strList               - возвращает список
     *                                  массивов строк типа
     *                                  String.
     * */
    private static SinglyLinkedList<String[]> readFile(String
path) throws FileNotFoundException
    {
        Scanner scanner = new Scanner(new File(path));
        SinglyLinkedList<String[]> strList = new
SinglyLinkedList<>();

        while (scanner.hasNextLine())
        {
            //для разделения по пробелам используется регулярное
            //выражение "//s"
            strList.push_back(scanner.nextLine().split("\\s"));
        }
        scanner.close();
        return strList;
    }
}
```

BandSpareMatrix.java

```
/**
 * Класс, реализующий хранение матрицы в ленточно строчном
 * формате.
 * Можно хранить только матрицы целочисленных значений.
 *
 * @release: 27.06.20
 * @last_update: 27.06.20
 *
 * @author Vladislav Sapozhnikov 19-IVT-3
 * */

public class BandSpareMatrix {
    private int[][] matrix;          //массив для хранения
                                     //матрицы в ленточно
                                     //строчном формате

    private int p;                   //поле, хранящее значение
                                     //нижней ширины ленты

    private int q;                   //поле, хранящее значение
                                     //верхней ширины ленты
    private int n;                   //поле, хранящее размер
                                     //квадратной матрицы

    private int m;                   //поле, хранящее общую
                                     //ширину ленты

    /**
     * Внутриклассовый метод, получающий значения полей –
     * характеристик матрицы.
     *
     * @param initMatrix - двумерный массив ссылочного типа
     *                     Integer.
     * */
    private void getParameters(Integer[][] initMatrix)
    {
        int buffQ = 0;
        int buffP = 0;
        for (Integer[] integers : initMatrix) {
            for (int j = 0; j < integers.length; j++) {
                if ((j < n) & (integers[j] != 0)) buffP = n - j;
                if ((j > n) & (integers[j] != 0)) buffQ = j - n;

                if (buffQ > q) q = buffQ;
                if (buffP > p) p = buffP;
            }
            n++;
        }
        m = p + q + 1;   }
```

```
    /**
     * Конструктор класса. Создает пустую матрицу с нулевыми
     * параметрами.
     * */
    BandSpareMatrix()
    {
        matrix = null;
        p = 0;
        m = 0;
        q = 0;
        n = 0;
    }

    /**
     * Метод, заполняющий уже ранее созданную ленто строчную
     * матрицу.
     *
     * @param initMatrix - двумерный массив ссылочного типа
     *                     Integer.
     * */
    public void fillMatrix(Integer[][] initMatrix)
    {
        //перед заполнением необходимо получить характеристики
        //матрицы
        getParameters(initMatrix);
        matrix = new int[n][m];

        for (int i = 0; i < initMatrix.length; i++)
        {
            for (int j = 0; j < initMatrix[i].length; j++)
            {
                //Алгоритм заполнения матрицы ленточно строчного
                //формата.
                if (((i <= j) && ((j - i) <= q )) || ((i > j) &&
((i - j) <= p)))
                {
                    matrix[i][j-i+p] = initMatrix[i][j];
                }
            }
        }
    }

    /**
     * Метод, выводящий данную матрицу ленточно строчно формата
     * в стандартный поток вывода.
     * */
    public void printMatrix()
    {
        for (int[] i: matrix)
        {
```

```
            for (int j: i)
            {
                System.out.print(j + " ");
            }
            System.out.println();
        }
    }

    /**
     * Метод, позволяющий получить сумму всех элементов данной
     * матрицы.
     *
     * @return size - целочисленное значение, равняющееся сумме
     *                всех элементов в матрице.
     * */
    public int getSumMatrixElement()
    {
        int sum = 0;

        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < m; j++)
            {
                sum += matrix[i][j];
            }
        }

        return sum;
    }
}
```

```
/**
 * Реализация односвязного списка (Данный класс взят из
 * Лабораторной работы №1).
 *
 * @author  Vladislav Sapozhnikov
 * @param <E> тип, содержащийся в данной коллекции
 */
public class SinglyLinkedList<E>
{
    /**
     * Класс - узел/ячейка списка.
     * Стандартный класс для большинства динамических структур.
     *
     * @param <E> тип, содержащийся в данном узле
     * */
    private static class Node<E>
    {
        E value;                //поле, хранящее данные
        Node<E> next;           //указатель на след. узел

        Node(Node<E> next, E value)
        {
            this.value = value;
            this.next  = next;
        }
    }

    private Node<E> head;       //указатель на начало списка
    private int size;           //поле для хранения размерности
                                //списка необходим для работы
                                //данной программы.
    /**
     * private метод для получения узла по заданному списку.
     *
     * @param index                     - индекс, по которому
     *                                    необходимо получить
     *                                    узел.
     *
     * @throws IndexOutOfBoundsException- исключение,
     *                                    сообщающее о выход за
     *                                    пределы списка.
     *                                   Данное исключение не
     *                                    обрабатывается внутри
     *                                    функции, а
     *                                    делегируется
     *                                   методу выше.
     *
     * @return Node<E>                  - возвращает узел списка
     * */
```

```
private Node<E> getByIndex(int index)
{
    if (index < 0)
    {
        throw new IndexOutOfBoundsException("Index must be a
                                            +positive number.");
    }

    Node<E> buffNode = head;
    for(int i = 0; i < index; i++)
    {
        if (buffNode.next == null)
        {
            throw new IndexOutOfBoundsException("Index [" +
                                + index + "] out of list.");
        }
        buffNode = buffNode.next;
    }
    return buffNode;
}

public SinglyLinkedList()
{
    head = null;
    size = 0;
}

/**
 * Метод, добавляющий новый узел в конец списка.
 *
 * @param value - данные, которые следует добавить в новый
 *                узел списка.
 * */
public void push_back(E value)
{
    if (head == null)
    {
        head = new Node<>(null, value);
    }
    else
    {
        Node<E> buffNode = head;
        while(buffNode.next != null)
        {
            buffNode = buffNode.next;
        }
        buffNode.next = new Node<>(null, value);
    }

    size++;
}
```

```
    /**
     * Перегруженный метод от класса Object.
     * Отвечает за вывод объекта в консоль.
     *
     * @return String - строка, содержащая строковое
     *                  представление объекта.
     * */
    @Override
    public String toString()
    {
        StringBuilder result = new StringBuilder("[");

        Node<E> buffNode = head;

        while(buffNode.next != null)
        {
            result.append(buffNode.value).append(", ");
            buffNode = buffNode.next;
        }
        result.append(buffNode.value).append("]");

        return new String(result);
    }

    /**
     * Метод, возвращающий значение из узла по переданному
     * индексу.
     *
     * @param index - индекс узла, содержимое которого
     *                необходимо получить.
     * */
    public E get(int index)
    {
        Node <E> buffNode = getByIndex(index);
        return buffNode.value;
    }

    /**
     * Метод, возвращающий размерность данного списка.
     *
     * @return size - целочисленное значение равное размерности
     *                текущего списка.
     * */
    public int getSize()
    {
        return size;
    }
}
```

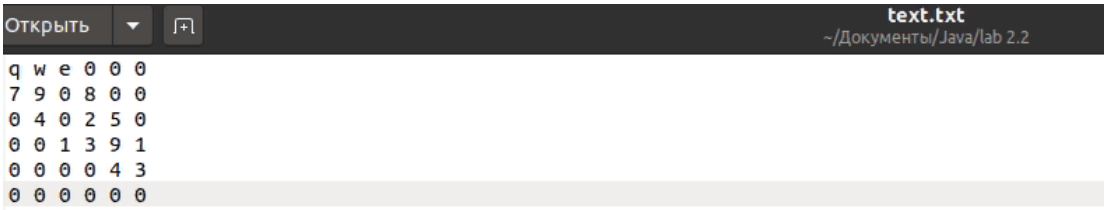
Скриншоты

1. Неверно указанный файл для чтения.

```
vladislav@progerSapog:~/Документы/Java/lab 2.2/out/production/lab 2.2$ java Main wjefnjew
Нет такого файла: wjefnjew
Укажите путь к существующему файлу и повторите попытку.
```

2. Неверное содержимое матрицы.

Входные параметры

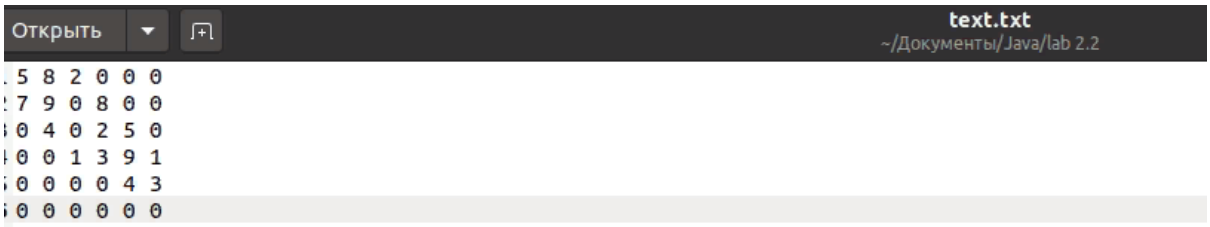


Поведение программы при неверных входных параметрах

```
vladislav@progerSapog:~/Документы/Java/lab 2.2/out/production/lab 2.2$ java Main text.txt
Неверное содержимое файла.
Измените переданный файл или его содержимое и повторите попытку.
```

3. Верные входные параметры (вариант 1)

Входные параметры

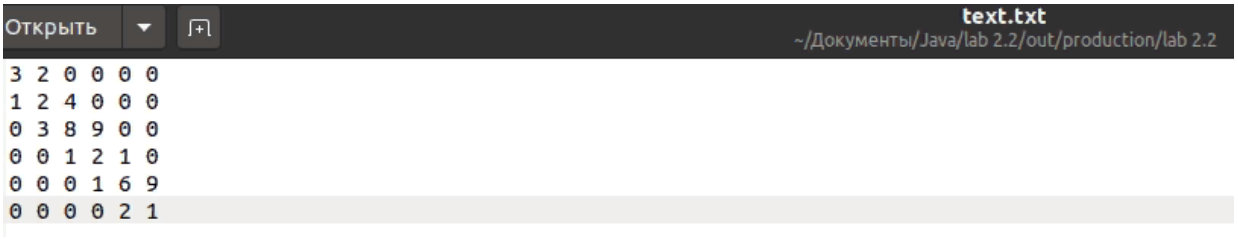


Поведение программы

```
vladislav@progerSapog:~/Документы/Java/lab 2.2/out/production/lab 2.2$ java Main text.txt

Исходная матрица:
5 8 2 0 0 0
7 9 0 8 0 0
0 4 0 2 5 0
0 0 1 3 9 1
0 0 0 0 4 3
0 0 0 0 0 0

Обработанная матрица (ленточно строчный формат):
0 5 8 2
7 9 0 8
4 0 2 5
1 3 9 1
0 4 3 0
0 0 0 0

Сумма элементов матрицы: 71
```

4. Верные входные параметры (вариант 2)

Входные параметры



Поведение программы

```
vladislav@progerSapog:~/Документы/Java/lab 2.2/out/production/lab 2.2$ java Main text.txt

Исходная матрица:
3 2 0 0 0 0
1 2 4 0 0 0
0 3 8 9 0 0
0 0 1 2 1 0
0 0 0 1 6 9
0 0 0 0 2 1

Обработанная матрица (ленточно строчный формат):
0 3 2
1 2 4
3 8 9
1 2 1
1 6 9
2 1 0

Сумма элементов матрицы: 55
```