Задача 9. Списки (List<T> – то же самое, что одномерные массивы, только лучше)

Входные данные для этой задачи должны читаться из текстового файла, а выходные – записываться в текстовый файл.

В файле каждый набор чисел (массив / список), а также другие параметры, если они предусмотрены условием задачи, должны быть записаны на отдельной строке.

Данная задача должна быть оформлена двумя способами:

1) в виде консольного приложения с разбором параметров командной строки,

2) в виде оконного приложения, где двумерный массив можно задать в JTable. При этом должна быть возможность загрузить данный из файла в JTable (реализованная в виде двух функций: чтение данных из файла в двумерный массив и отображение двумерного массива в JTable), а также сохранить данные из JTable в файл (реализованная в виде двух функций: чтение данных из JTable в двумерный массив и запись двумерного массива в файл).

Функции, реализующие логику задачи и чтение / запись данных из файлов / в файлы, должны быть оформлены в виде отдельного модуля (в отдельном файле). Этот модуль без каких-либо изменений должен использоваться в двух программах: с консольным интерфесом (файлы для чтения / записи задаются в параметрах командной строки) и оконным интерфейсом.

Заранее придумать не менее 10 различных тестов, охватывающих как типичные, так и все возможные граничные (наиболее невероятные и показательные) ситуации. (Сохранить в текстовых файлах input01.txt, input02.txt и т. д.)

Решение, естественно, должно быть оформлено в виде отдельной функции. В реализации обязательно использовать вспомогательные функции.

Запрещено использовать любые стандартные (уже реализованные в библиотеке языка Java) функции и методы массивов и списков (за исключением создания списка, получения размера и добавления нового элемента). Аналоги стандартных функций необходимо реализовать (какие – для каждого варианта указано отдельно).

Реализация в виде консольного приложения с разбором параметров командной строки

Имена файлов для чтения и записи должны передаваться в параметрах командной строки, например, так:

```
> java ru.vsu.cs.course1.Task8 .\input.txt .\output.txt
```

(Здесь "." означает текущую директорию, т.е. ".\" можно опустить. Также при обращении к файлам можно использовать "..", что будет означать родительскую директорию относительно текущей, например, "..\..\input05.txt")

Еще лучше будет, если вы реализуете разбор параметров командной строки, чтобы имена входных и выходных файлов можно было задавать в виде именованных параметров, например, так:

```
> java ru.vsu.cs.course1.Task8 -i .\input.txt -o .\output.txt
```

или так (а лучше и так и так)

```
> java ru.vsu.cs.course1.Task8 --input-file=.\input.txt --output-
file=.\output.txt
```

Если программа запускается без указания необходимых аргументов, либо отсутствует входной файл, то в поток ошибки (System.err.println()) должно печататься сообщение об ошибке и программа должна завершаться с кодом, отличным от 0 (код успешного

завершения).

Для разбора параметров командной строки реализовать функцию:

```
public static InputArgs parseCmdArgs(String[] args)
```

InputArgs – класс, в котором описаны поля inputFile и outputFile (и, возможно, какие-от другие параметры в зависимости от задачи).

По возможности перебирать элементы списка циклом for(Integer v: list).

Варианты:

1. Перевернуть числа в списке слева направо:

   { 2, 7, 3, 5, 100 } → { 100, 5, 3, 7, 2 }

   Реализовать в виде функции:

   ```
   public static void process(List<Integer> list)
   ```

   Дополнительный список или массив не заводить.

2. Реализовать функцию, которая будет формировать новый список путем циклического сдвига переданного списка на n позиций. Если n > 0, то сдвиг происходит вправо, n < 0 – влево, например:

   ({3, 5, 2, 4, 9, 1}, 2) → { 9, 1, 3, 5, 2, 4 }
   ({3, 5, 2, 4, 9, 1}, -15) → { 4, 9, 1, 3, 5, 2 }

   Реализовать в виде функции:

   ```
   public static List<Integer> createNewList(List<Integer> list, int n)
   ```

3. Составить новый список из первых n минимальных чисел переданного списка, взятых по одному разу. Если в переданном списке нет n различных чисел, то итоговый список будет короче n.

   Примеры:

   ({ 7, 2, 3, 2, 2, 6, 5, 7, 8, 8, 3 }, 4) → { 2, 3, 5, 6 }
   ({ 3, 3, 7, 3, 5, 3 }, 5) → { 3, 5, 7 }

   Реализовать в виде функции

   ```
   public static List<Integer> createNewList(List<Integer> list, int n)
   ```

4. Составить новый список из чисел, которые совпадают в совпадающих позициях 2-х списков, например:

   ({2, 3, 7, 0, 8, -1, 78, 8, 99, 4, 3 }, { -1, 3, 7, -5, -8, 2, 77, 9, 99, -4, 3}) → { 3, 7, 99, 3 }

   Реализовать в виде функции:

   ```
   public static List<Integer> createNewList(List<Integer> list1,
       List<Integer> list2)
   ```

5. Реализовать функцию:

   ```
   public static List<Integer> uniq(List<Integer> list)
   ```

   , принимающую список чисел и возвращающую новый список чисел, в который включены числа из первого списка по одному разу (в порядке появления в переданном списке). Для удобства реализовать дополнительную функцию:

   ```
   public static int indexOf(List<Integer> list, int value)
   ```

, которую использовать в реализации функции Uniq.

6. (*) Вернуть в виде списка самую длинную подпоследовательность подряд идущих чисел, сумма которых максимальна (задача имеет смысл, если в исходной последовательности будут встречаться, как положительные, так и отрицательные элементы). Если будет несколько таких подпоследовательностей с одинаковой суммой, выбрать самую короткую. Если и длина будет совпадать, то вернуть первую (от начала списка) такую подпоследовательность. Реализовать поиск такой подпоследовательности в один проход (один цикл, второй цикл будет использоваться для копирования элементов найденной подпоследовательности в результирующий список).

   Реализовать в виде функции

   ```
   public static List<Integer> createNewList(List<Integer> list)
   ```

7. Реализовать функцию:

   ```
   public static List<Integer> inList1NotInList2(
       List<Integer> list1, List<Integer> list2)
   ```

   которая вернет список элементов первого списка, которых нет во втором списке (в порядке появления в первом списке). Для удобства реализовать дополнительную функцию:

   ```
   public static int indexOf(List<Integer> list, int value)
   ```

   , которую использовать в реализации функции inList1NotInList2.

8. Реализовать функцию:

   ```
   public static List<Integer> inList1OrInList2(
       List<Integer> list1, List<Integer> list2)
   ```

   , которая вернет список чисел, представленных по одному разу, которые есть в первом или во втором списке (вначале числа, которые есть в первом списке в порядке появления, затем числа, которые есть во втором списке (но нет в первом), также в порядке появления). Для удобства реализовать дополнительную функцию:

   ```
   public static int indexOf(List<Integer> list, int value)
   ```

   , которую использовать в реализации функции inList1NotInList2.

9. Реализовать функцию:

   ```
   public static List<Integer> inList1XorInList2(
       List<Integer> list1, List<Integer> list2)
   ```

   , которая вернет список чисел, представленных по одному разу, которые есть или только в первом списке или только во втором списке, но не в двух списках одновременно (вначале числа, которые есть в первом списке в порядке появления, затем числа, которые есть во втором списке (но нет в первом), также в порядке появления). Для удобства реализовать дополнительную функцию:

   ```
   public static int indexOf(List<Integer> list, int value)
   ```

   , которую использовать в реализации функции inList1NotInList2.

10. Реализовать функцию:

    ```
    public static void process(List<Integer> list)
    ```

    , в которой все отрицательные элементы массива списка перенести в его начало, а все остальные – в конец, сохраняя исходное взаимное расположение как среди отрицательных, так и среди остальных элементов. Дополнительный список или массив

не заводить.

11. (*) У вас есть несколько гирь (<= 10) разного веса (по одной каждого веса). Вам необходимо найти все возможные способы, которым можно составить вес sumWeight. Решение задачи необходимо реализовать в виде функции:

```
public static List<List<Integer>> selectionCount(
    List<Integer> weights, int sumWeight)
```

, где

- weights – веса гирь;
- sumWeight – вес, который необходимо набрать.

Функция возвращает все подходящие варианты в виде списка вариантов, где подходящий вариант, в свою очередь, представлен списком чисел – весов гирь из которых составлен необходимый общий вес.

Подсказка: надо перебрать все возможные числа от 0 до $2^n - 1$, где n – кол-во весов различных гирь. В этом случае младшие биты соответствующих чисел будут образовывать все возможные комбинации гирь (1 – соответствующая гиря включается в набор, 0 – нет). Остается проверить вес каждой комбинации и, если он равен нужному, включить соответствующую комбинацию в итоговый набор (список вариантов).

12. Необходимо обработать список чисел следующим образом: отсортировать положительные

элементы, при этом отрицательные и 0 оставить на своих местах, например:

{ 9, -1, 3, 7, -5, -7, 1, 3, -3, 2, 1, -8 } → { 1, -1, 1, 2, -5, -7, 3, 3, -3, 7, 9, -8 }

Реализовать в виде функции:

```
public static void process(List<Integer> list)  ,
```

Дополнительный список или массив не заводить.

Подсказка: надо в цикле с начала списка (с позиции 0) и до конца (точнее до позиции size() - 2) для каждого положительного элемента искать позицию наименьшего положительного элемента списка, начиная с текущей позиции и правее. Затем обменять два элемента – текущий и найденный минимальный положительный.

13. Реализовать функцию:

```
public static List<Integer> createNewList(List<Integer> list, int n)
```

, которая вернет список чисел, представленных по одному разу, которые встречаются в переданном списке n или более раз (в новом списке числа должны следовать в порядке появления в списке list). Для удобства реализовать дополнительную функцию:

```
public static int countOfValue(List<Integer> list, int value)
```

, которую использовать в реализации функции createNewList.

14. (*) Перевернуть числа в списке между первым максимальным элементом списка и последним минимальным, слева направо, например:

{ 2, 11, 8, 11, 2, 3, 2, 2, 6, 7, 2, 3, 5, 10 } → { 2, 11, 7, 6, 2, 2, 3, 2, 11, 8, 2, 3, 5, 10 }

Реализовать в виде функции:

```
public static void process(List<Integer> list)
```

Дополнительный список или массив не заводить. Для удобства реализовать дополнительные функции:

```
public static int findMax(List<Integer> list)
public static int findMin(List<Integer> list)
public static int firstIndexOf(List<Integer> list, int value)
public static int lastIndexOf(List<Integer> list, int value)
```

, которые использовать в реализации функции process.

15. Необходимо реализовать функцию:

```
public static List<Integer> createNewList(List<Integer> list)
```

, которая из списка чисел создаст новый список, в котором подряд идущие одинаковые числа будут заменены одним вхождением, например:

{ 2, 11, 11, 11, 2, 3, 2, 2, 6, 6, 2, 3, 3, 3, 10 } → { 2, 11, 2, 3, 2, 6, 2, 3, 10 }

16. (*) Необходимо из списка чисел путем вычеркивания некоторых чисел, не меняя порядка остальных, составить максимально длинный новый список чисел, образующих арифметическую прогрессию. Будем считать, что два любых числа (или одно для списка из одного элемента) всегда образуют такую прогрессию. Если можно составить несколько таких списков одинаковой максимальной длины, разрешается составить любой, например:

{ 11, 3, 2, 5, 4, 7, 7, 1, 4, 6, 9, 8, 8, 7, 1, 3, 2 } → { 3, 5, 7, 9 }
или { 2, 4, 6, 8 }
или { 5, 4, 3, 2 }

Реализовать в виде функции:

```
public static List<Integer> createNewList(List<Integer> list)
```

17. Реализовать метод пузырьковой сортировки элементов списка с индексами от index1 до index2 включительно:

```
public static void sort(List<Integer> list, int index1, int index2)
```

В собственной реализации не использовать существующие методы sort, а также не создавать новых списков или массивов. Если index1 < index2, до числа в диапазоне [index1 – index2] должны быть упорядочены по возрастанию, если index1 > index2, до числа в диапазоне [index2 – index1] должны быть упорядочены по убыванию, например:

({ 4, 7, 1, 3, 5, 6, 5, 2, 10, 7 }, 1, 4) → { 4, 1, 3, 5, 7, 6, 5, 2, 10, 7 }
({ 4, 7, 1, 3, 5, 6, 5, 2, 10, 7 }, 8, 2) → { 4, 7, 10, 6, 5, 5, 3, 2, 1, 7 }

Также учесть возможность задания index1 и index2 за пределами допустимых значений индексов элементов для переданного списка. В этом случае заменить неправильные индексы первым или, соответственно, последним индексом из допустимых для переданного списка.

18. Составить новый список чисел из двух переданных списков следующим образом. На четную позицию ставить элементы из первого списка, на нечетную – из второго. В случае, если один список короче другого, то на недостающие позиции (неважно, четные или нечетные) ставить элементы из более длинного списка (т.е. итоговый список обязательно будет длиной, равной длине более длинного списка), например:

({ 1, 2, 3, 4 }, { 101, 102, 103, 104, 105, 106, 107 }) → { 1, 102, 3, 104, 105, 106, 107 }

Реализовать в виде функции:

```
public static List<Integer> createNewList(List<Integer> list1,
    List<Integer> list2)
```

19. Необходимо из списка чисел составить новый список, состоящий из чисел, которые останутся на своем месте в переданном списке, если этот список отсортировать по возрастанию, например:

    { 3, 1, 3, 7, 7, 5, 9, 9, 15, 12, 10 } → { 3, 7, 9, 9, 12 }

    Переданный список не должен измениться (самый простой вариант – создать дополнительный список из элементов переданного списка, новый список отсортировать, а затем сравнивать элементы двух списков).

    Реализовать в виде функции

    ```
    public static List<Integer> createNewList(List<Integer> list)
    ```

20. Вернуть в виде списка самую длинную неубывающую подпоследовательность подряд идущих чисел в переданном списке. Если можно выделить несколько таких подпоследовательностей одинаковой длины, то вернуть ту, сумма элементов которой будет максимальна. Если и сумма элементов совпадает, то вернуть первую (от начала списка) такую подпоследовательность.

    Реализовать в виде функции

    ```
    public static List<Integer> createNewList(List<Integer> list)
    ```

    Для удобства реализовать дополнительные функции:

    ```
    public static void sort(List<Integer> list)
    ```

21. Составить новый список чисел, отсортированных по возрастанию, из чисел переданного списка, встречающихся в исходном списке максимальное число раз, например:

    { 9, 3, 1, 3, 7, 7, 5, 3, 9, 9, 15, 12, 10 } → { 3, 9 }

    Реализовать в виде функции

    ```
    public static List<Integer> createNewList(List<Integer> list)
    ```

    Для удобства реализовать дополнительные функции:

    ```
    public static void sort(List<Integer> list)
    ```

22. Реализовать функцию:

    ```
    public static List<Integer> createNewList(
        List<Integer> list1, List<Integer> list2)
    ```

    , которая создаст новый список из элементов первого списка, взятых по одному разу, таких, для которых во втором списке есть делители данного числа (числа, отличные по модулю от самого числа, а также 0 и 1), например:

    ({ 7, 12, 0, 8, -21, 145, 100, 17 }, { 4, 3, 0, -1, -8, 29, 7 }) → { 12, 8, -21, 145, 100 }

    Для удобства реализовать дополнительную функцию:

    ```
    public static int indexOf(List<Integer> list, int value)
    ```

    , которую использовать в реализации функции createNewList.

23. (*) Составить максимально длинный новый список из чисел переданного списка, такой, что числа в новом списке будут идти по порядку (т.е. отличаться на 1). Если можно составить несколько чисел одинаковой длины, выбрать список меньших чисел, например:

    { 9, 7, -1, 5, 4, 12, 7, 8, 1, 10, 3 } → { 3, 4, 5 }

    Реализовать в виде функции

    ```
    public static List<Integer> createNewList(List<Integer> list)
    ```

Для удобства реализовать дополнительные функции:

```
public static void sort(List<Integer> list)
```

Переданный список не должен измениться (т.е. для сортировки надо создать новый список).

24. Создать новый список из элементов переданного списка, которые встречаются четное количество раз, взятых по одному разу. Элементы должны располагаться в созданном списке в порядке их появления в переданном списке, если элементы последнего просматривать с конца списка, например:

    { 7, 1, 17, 3, 10, 10, 5, 17, 5, 7, 8, 2, 3, 10, 4, 5, 3, 10, 5 } → { 5, 10, 7, 17 }

    Реализовать в виде функции

    ```
    public static List<Integer> createNewList(List<Integer> list)
    ```

    Для удобства реализовать дополнительную функцию:

    ```
    public static int countOf(List<Integer> list, int value)
    ```

25. Реализовать функцию:

    ```
    public static List<Integer> createNewList(List<Integer> list1,
       List<Integer> list2)
    ```

    , которая создаст новый список элементов из четных элементов первого списка, которых нет во втором списке, и нечетных элементов второго списка, которых нет в первом списке. Элементы в возвращаемом списке должны быть представлены в порядке возрастания значений.

    Для удобства реализовать дополнительные функции:

    ```
    public static int indexOf(List<Integer> list, int value)
    public static void sort(List<Integer> list)
    ```

26. Реализовать функцию:

    ```
    public static List<Integer> createNewList(List<Integer> list1, List<Integer>
    list2)
    ```

    , которая создаст новый список элементов из элементов первого списка, каждый из которых повторяется такое кол-во раз, которое данный элемент встречается во втором списке, например:

    ({ 5, 3, 1, 2, 3, 3, 7, 4, 8 }, { 5, 3, 7, 3, 7, 2, 2, 8, 7 }) → { 5, 3, 3, 2, 2, 3, 3, 3, 3, 7, 7, 7, 8 }

    Для удобства реализовать дополнительную функцию:

    ```
    public static int countOf(List<Integer> list, int value)
    ```

    , которую использовать в реализации функции createNewList.

27. Реализовать функцию:

    ```
    public static void process(List<Integer> list1, List<Integer> list2)
    ```

    , которая изменит первый список следующим образом: перенесет в конец списка элементы, которые не встречаются во втором списке и при этом порядок переставленных элементов не должен быть нарушен, например:

    ({ 4, 5, 2, 1, 7, 5, 2, 9, 6 }, { 5, 1, 8, 9, 1 }) → ({ 5, 1, 5, 9, 4, 2, 7, 2, 6 }, { 5, 1, 8, 9, 1, 6 })

    Дополнительный список или массив не заводить.

    Для удобства реализовать дополнительную функцию:

```
public static int indexOf(List<Integer> list, int value)
```

, которую использовать в реализации функции createNewList.

28. (*) Если путем перестановки элементов из списка чисел можно получить арифметическую прогрессию, выполнить такую перестановку (в противном случае оставить список как есть). Реализовать в виде функции:

```
public static void process(List<Integer> list)
```

Подсказка: найти минимальный и максимальный элемент, вычислить разность прогрессии d (проверить, что разность – целое число) и проверить на наличие числа $a_{min} + d$, $a_{min} + 2d$, $a_{min} + 3d$ … $a_{min} + (n - 1) * d$, где n – кол-во элементов списка. Если все указанные числа присутствуют, то для образования арифметической прогрессии достаточно список отсортировать. Для удобства реализовать дополнительные функции:

```
public static int findMax(List<Integer> list)
public static int findMin(List<Integer> list)
public static int indexOf(List<Integer> list, int value)
public static void sort(List<Integer> list)
```

29. (*) Реализовать функцию, которая создаст новый список из переданного списка, упорядочив элементы первого списка в порядке встречаемости элементов (от наиболее встречаемых до наименее встречаемых). Если какие-то элементы встречаются одинаковое кол-во раз, то в этом случае упорядочить элементы по возрастанию. Например:

{ 5, 3, 5, 8, 7, 7, 1, 8, 9, -1, 0, 8, -1, 3, 4 } → { 8, 8, 8, -1, -1, 3, 3, 5, 5, 7, 7, 0, 1, 4, 9 }

Реализовать в виде функции

```
public static List<Integer> createNewList(List<Integer> list)
```

Для удобства реализовать дополнительные функции:

```
public static int countOf(List<Integer> list, int value)
public static void sort(List<Integer> list, int index1, int index2)
```

, которые использовать в реализации функции createNewList.

Подсказка: если не обращать внимание на эффективность алгоритма, то задачу можно решать так:

1) находим наибольшую встречаемость (для каждого элемента вызываем countOf);

2) для каждой встречаемости от максимальной до 1:

   а) добавляем в новый список элементы с данной встречаемостью;

   б) сортируем добавленные на данном шаге элементы в новом списке с помощью sort

30. (*) Необходимо, заменив минимальное кол-во чисел в списке, сделать его арифметической прогрессией, например:

{ 1, 16, 4, 10, 7, 11, 1, -2 } → { 19, 16, 13, 10, 7, 4, 1, -2 } (заменены 3 элемента)

Реализовать в виде функции:

```
public static void process(List<Integer> list)
```

В случае нескольких подходящих вариантов замены минимального кол-ва элементов можно использовать любой.

Подсказка: необходимо для каждой пары элементов списка посчитать, сколько элементов списка придется изменить, не меняя выбранную пару элементов, чтобы список стал арифметической прогрессией (очевидно, что в целых числах некоторые пары элементов сразу же не будут подходить для построения арифметической прогрессии;

например, для приведенного примера в качестве опорной пары не подходят числа 4 и 7, т.к. между ними не может быть дробного числа 5,5).

31. Реализовать функцию, которая составляет новый список чисел, состоящий из элементов двух переданных списков, расположенных по возрастанию, например:

    ({ 3, 9, 2, 7}, { 8, 3, 5 }) → { 2, 3, 3, 5, 7, 8, 9 }

    Реализовать в виде функции:

```
public static List<Integer> createNewList(List<Integer> list1,
    List<Integer> list2)
```

    Передаваемые списки не изменять.

    Для удобства реализовать дополнительные функции:

```
public static int copy(
    List<Integer> src,  // список, с которого копируются элементы
    int srcIndex,    // позиция, с которой копируются из src
    List<Integer> dst,  // список, в который копируются элементы
    int dstIndex,    // позиция, с которой копируются элементы в dst
    int count        // кол-во копируемых элементов
)
public static void sort(List<Integer> list)
```