Пермский филиал федерального государственного автономного образовательного учреждения высшего образования «Национальный исследовательский университет «Высшая школа экономики»

Факультет экономики, менеджмента и бизнес-информатики

Образовательная программа бакалавриата «Программная инженерия»

ОТЧЕТ
по учебной практике

Выполнил студент группы ПИ-19-3
Никитин Андрей Владимирович

_____
(подпись)

Проверил:

Руководитель практики
преподаватель кафедры информационных технологий в бизнесе

Ланин Вячеслав Владимирович

_____ _____
(оценка) (подпись)

_____
(дата)

Пермь, 2020 год

Аннотация

В данном отчете описаны этапы проектирования, разработки и тестирования 12 задач. Программы реализованы на языке высокого уровня C#.

Работа выполнена студентом НИУ ВШЭ Пермь Никитиным Андреем Владимировичем. Кафедра информационных технологий в бизнесе.

Работа содержит 85 страниц формата А4 основного текста, включая 12 глав соответствующей каждой учебной задаче.

Библиографический список включает в себя 4 источника.

В каждой главе содержится анализ задачи (установка функциональных и не функциональных требований к задаче), проектирование алгоритмов и тестирования ПО. Приложения к задачам содержат реализацию задач.

Оглавление

Введение

Основная задача данной работы состоит в развитии практических навыков создания программных систем при помощи современных средств разработки, таких как MS Visual Studio, а также получения необходимого опыта проектирования, реализации и тестирования различных программных систем.

Цель данной работы заключается в закреплении знаний и навыков, полученных по дисциплинам «Дискретная математика», «Компьютерный практикум по основам алгоритмизации и методам программирования», «Линейная алгебра», «Программирование», «Введение в программную инженерию», а также приобретение опыта алгоритмизации задач.

Для достижения поставленной цели необходимо:

1. Наличие практических навыков работы с языком программирования высокого уровня C# и современных сред разработки для реализации предложенных программ.
2. Умение работать с системой контроля версий Git посредством использования веб-сервиса GitHub.
3. Провести тестирование всех разработанных систем по критериям черного ящика.
4. Развитие практических навыков объектно-ориентированного программирования.

Основная часть данной работы содержит результаты выполнения 12 заданий учебной практики и включает в себя: постановку задачи, формат входных и выходных данных, описание решения задачи, разработка функциональных и нефункциональных требований, разработку алгоритмов, процесса реализации и результатов тестирования программы.

1. Задача 1

Рассмотрим компьютерную сеть с настроенной TCP/IP маршрутизацией. Будем рассматривать некоторую ее модификацию. А именно в этой сети находить N подсетей. Каждая подсеть характеризуется своей маской. Маска подсети представляет собой 4 однобайтных числа, разделенных точкой. Причем для масок выполнено следующее свойство: если представить маску в двоичном виде, то сначала она будет содержать k единиц, а потом q нулей, причем k + q = 32. Например, 255.255.255.0 — маска подсети, а 192.168.0.1 — нет.

Вам даны M пар IP адресов компьютеров. Для каждой из них Вам надо определить, в скольких подсетях из заданных они лежат. Вывести результат в выходной файл.

Входной файл: input.txt

Выходной файл: output.txt

Программа должна выполняться не более чем за 0.4 секунды и требовать не более чем 16 МБ памяти.

1.1. Анализ

Для решения задачи нужно:

1. Считать количество подсетей из файла.
2. Записать маски подсетей.
3. Считать количество пар ip-адресов из файла.
4. Записать пары ip-адресов.
5. Выполнить проверку принадлежностей пары ip-адресов одной маске подсетей.
6. Записать результат в выходной файл.

Поясним, как получается двоичное представление IP-адреса. Для этого числа, составляющие IP-адрес, представляются в двоичной системе счисления (при этом каждое из них дополняется ведущими нулями до длины в 8 цифр), после чего удаляются точки. Получившееся 32-битное число и есть двоичное представление IP-адреса. Например, для адреса 192.168.0.1 этот процесс выглядит так: 192.168.0.1 → 11000000.10101000.00000000.00000001 → 11000000101010000000000000000001.

Таким образом, двоичным представлением IP-адреса 192.168.0.1 является 11000000101010000000000000000001.

Будем говорить, что два компьютера с IP1 и IP2 лежат в подсети, если IP1 /\ Mask = IP2 /\ Mask, где Mask — маска этой подсети, а /\ — операция побитового логического «и». IP компьютера представляет собой так же 4 однобайтных числа, разделенных точкой.

1.2. Проектирование

Согласно проведенному анализу и приведенному там алгоритму была составлена блок-схема для решения задачи (см. рис. 1.1).

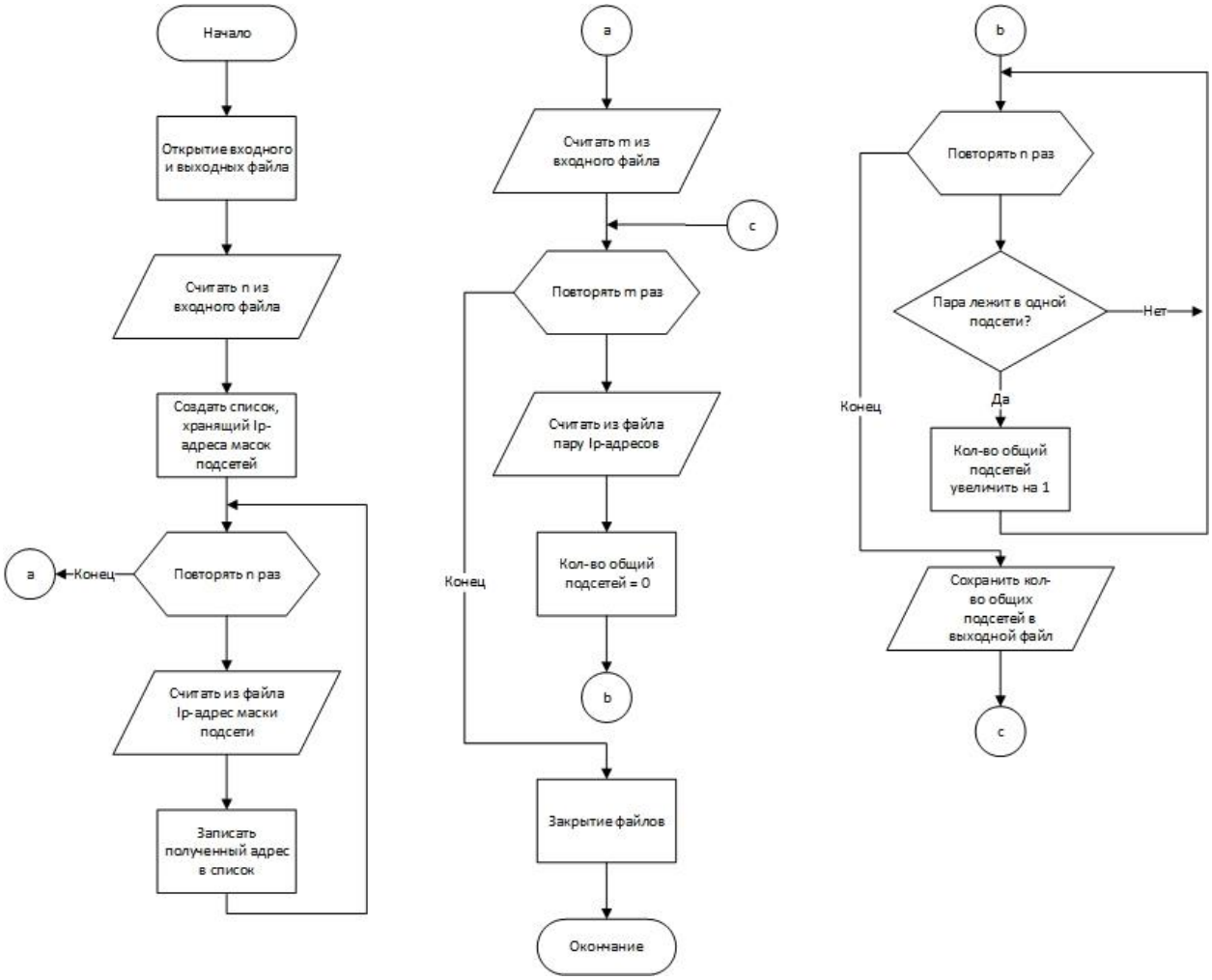


Рисунок 1.3.1. Блок-схема для задачи 1

1.3. Тестирование

Для тестирования была составлена следующая таблица тестов (табл. 1.1). Она содержит входные данные, ожидаемые и реальные результаты работы программы с такими данными:

Таблица 3.1. Таблица тестов для задачи 1

№	Входные данные	Ожидаемые результаты	Реальные результаты
T1	Файла input.txt нет Файла output.txt нет	0	0
T2	Файл input.txt пуст	0	0
T3	2 255.255.255.255 255.255.255.0 3 192.168.31.1 192.168.31.2 192.168.31.3 192.168.31.4 192.168.31.1 192.167.31.2	1 1 0	1 1 0

Данные тесты покрывают критерии черного ящика, представленные в таблице А.1 в приложении А.

Требования к скорости работы и занимаемой памяти также были выполнены.

2. Задача 2

Необходимо написать программу, реализующее структуру данных для хранения множества натуральных чисел. Данная структура должна хранить n множеств, в каждое из которых могут входить натуральные числа от 1 до m, при этом одно и то же число может принадлежать нескольким множествам одновременно. Необходимо реализовать операции добавления элемента в множество и вывода всех элементов множества и вывода номеров всех множеств, в которых лежит данный элемент. Вывести результат в выходной файл.

Входной файл: input.txt

Выходной файл: output.txt

Программа должна выполняться не более чем за 1 секунду и требовать не более чем 16 МБ памяти.

2.1. Анализ

Для решения задачи нужно:

7. Считать количество множеств и чисел, которые может принять множество, из файла.
8. Считать количество операций.
9. Выяснить, что это за операция.
10. Выполнить соответствующую операцию.
11. Повторять, пока операций в файле не кончатся.

Для определения типа считанной операции следует пользоваться конструкцией ЕСЛИ ИНАЧЕ, где в условие будет записано сравнение считанной операции с соответствующему названию операции, приведенного в постановке задачи.

Операция «ADD element set» добавляет элемент element ($1 \leq element \leq m$) в множество номер set ($1 \leq set \leq n$). Операция «LISTSET set» выводит все элементы множества номер set. Если такой коллекции нет, то вывести -1. Операция «LISTSETSOF element» выводит номера всех множеств, содержащих элемент element. Если такой коллекции нет, то вывести -1

Полученные в результате ввода множества типа Set необходимо хранить в коллекции типа Vector.

Для ошибкоустойчивости программы следует учесть ситуацию, при которой входной или выходной файл не будет существовать во время работы программы. То есть надо или создать файл с соответствующим названием, или сообщить пользователю об отсутствии искомого файла.

Также для удобства пользования необходимо вывести в консоль информацию о считанных из входного файла данных, выполненных операций и о загруженной в выходной файл информации.

2.2. Проектирование

Согласно проведенному анализу и приведенному там алгоритму была составлена блок-схема для решения задачи (см. рис. 2.1).

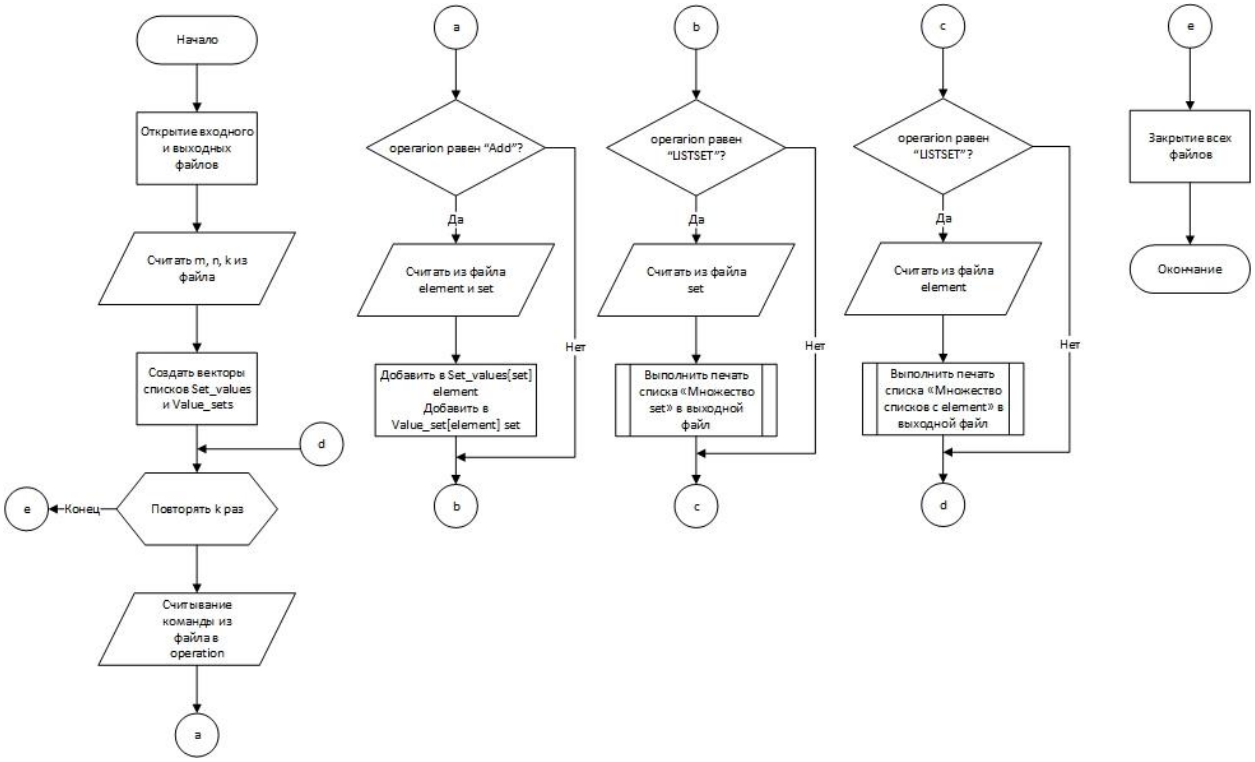


Рисунок 2.3.2. Блок-схема для задачи 2

2.3. Тестирование

Для тестирования была составлена следующая таблица тестов (табл. 1.1). Она содержит входные данные, ожидаемые и реальные результаты работы программы с такими данными:

Таблица 3.2. Таблица тестов для задачи 1

№	Входные данные	Ожидаемые результаты	Реальные результаты
T1	Файла input.txt нет	0	0
T2	Файл input.txt пуст	0	0
T3	10 10 5 ADD 1 1, ADD 1 2, ADD 2 1 LISTSET 1, LISTSETSOF 1	1 2 1 2	1 2 1 2

Данные тесты покрывают критерии черного ящика, представленные в таблице B.1 в приложении B.

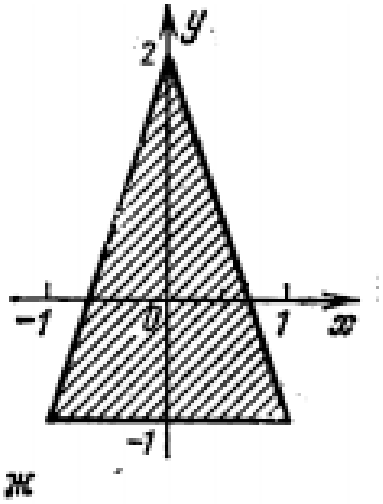
Требования к скорости работы и занимаемой памяти также были выполнены.

3. Задача 3

Даны действительные числа x , y. Определить, принадлежит ли точка с координатами x , y заштрихованной части плоскости (рис. 3.1).

Рисунок 3.1. График к задаче 3

3.1. Анализ



Данному графику (рис.3.1) соответствует функция

$$f(x) = \begin{cases} 3x + 2, -1 \le x \le 0 \\ -3x + 2, 0 \le x \le 1. \\ -1, -1 \le x \le 1 \end{cases}$$

Для вычисления принадлежности точки $f(x) \le y$.

Также необходимо производить проверку ввода на правильность ввода действительного числа.

3.2. Проектирование

Согласно проведенному анализу, была составлена блок-схема для решения задачи (см. рис. 3.2).

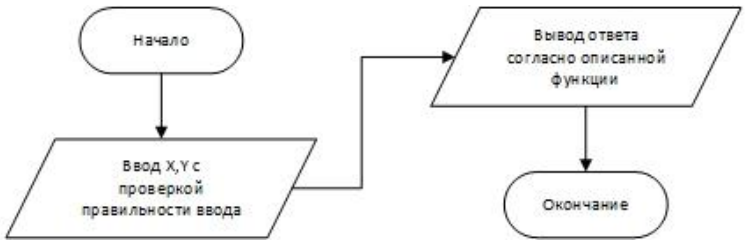


Рисунок 3.2. Блок-схема для задачи 3

3.3. Тестирование

Для тестирования была составлена следующая таблица тестов (табл. 3.1). Она содержит входные данные, ожидаемые и реальные результаты работы программы с такими данными:

Таблица 3.1. Таблица тестов для задачи 2

№	Входные данные	Ожидаемые результаты	Реальные результаты
T1	h	Ошибка ввода	Ошибка ввода
T2	0 0	Принадлежит	Принадлежит
T3	-0,5 -1	Принадлежит	Принадлежит
T4	-0,5 1	Принадлежит	Принадлежит
T5	0 -1	Принадлежит	Принадлежит
T6	4 0	Не принадлежит	Не принадлежит

Данные тесты покрывают критерии черного ящика, представленные в таблице С.1 в приложении С.

4. Задача 4

Даны действительные числа $u_1, u_2, v_1, v_2, w_1, w_2$.

Получить $2u + \frac{2uw}{2+w-v} - 7$, где u, v, w – комплексные числа $u_1 + iu_2$, $v_1 + iv_2, w_1 + iw_2$. (Определить процедуры выполнения арифметических операций над комплексными числами.)

4.1. Анализ

Для решения данной задачи необходимо реализовать такие функции как: умножение и деление комплексного числа на целое, вычитание из комплексного числа целое, сложение, разность и умножение комплексных чисел.

Для проверки правильности ввода необходимо реализовать проверку входных данных.

4.2. Проектирование

Согласно проведенному анализу, была составлена блок-схема для решения задачи (см. рис. 4.1).

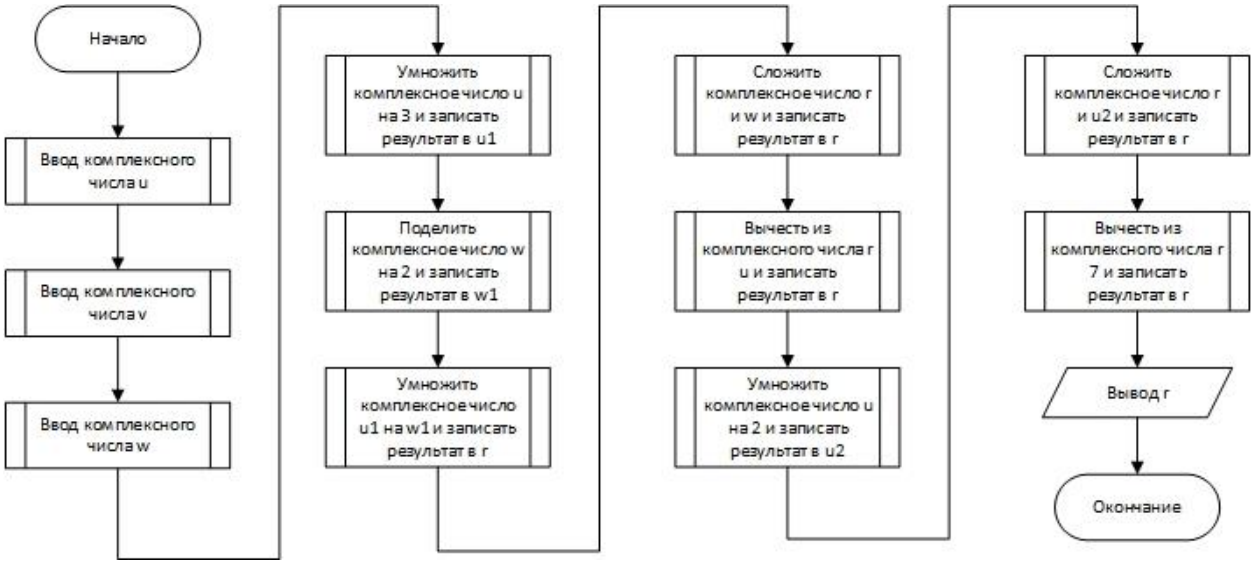


Рисунок 4.1. Блок-схема для задачи 4

4.3. Тестирование

Для тестирования была составлена следующая таблица тестов (табл. 4.1). Она содержит входные данные, ожидаемые и реальные результаты работы программы с такими данными:

Таблица 4.1. Таблица тестов для задачи 4

№	Входные данные	Ожидаемые результаты	Реальные результаты
T1	j	Ошибка ввода	Ошибка ввода
T2	-1 2 0.2 5 0 0	-8-2i	-8-2i

Данные тесты покрывают критерии черного ящика, представленные в таблице D.1 в приложении D.

5. Задача 5

Дана действительная квадратная матрица порядка n. Найти наибольшее из значений элементов, расположенных в заштрихованной части матрицы (рис 5.1).

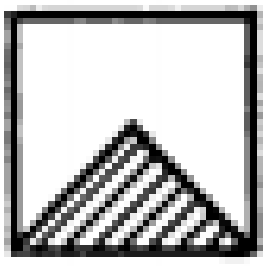


Рисунок 5.1. Заштрихованная часть к задаче 5

5.1. Анализ

Для нахождения искомой последовательности необходимо разработать алгоритм прохождения заштрихованной части квадратной матрицы по равнобедренному треугольнику.

Для работы данного алгоритма необходима переменная start, задающая стартовую и конечную позицию для прохождения строк матрицы, чтобы образовать область на рис 5.1.

Для движения влево необходимо задать у значение start. Движемся до тех пор, пока у < ширина матрицы - start. Во время движения проверяем текущий элемент на максимальное значение. Когда дошли до правой границы, необходимо сменить текущую строку матрицы и увеличить значение start на 1. Данный алгоритм повторить до тех пор, пока не дойдем до центральной строки.

Также необходимо сделать проверку ввода. Вводить можно только вещественные числа.

5.2. Проектирование

Согласно проведенному анализу, была составлена блок-схема для решения задачи (см. рис. 5.1).

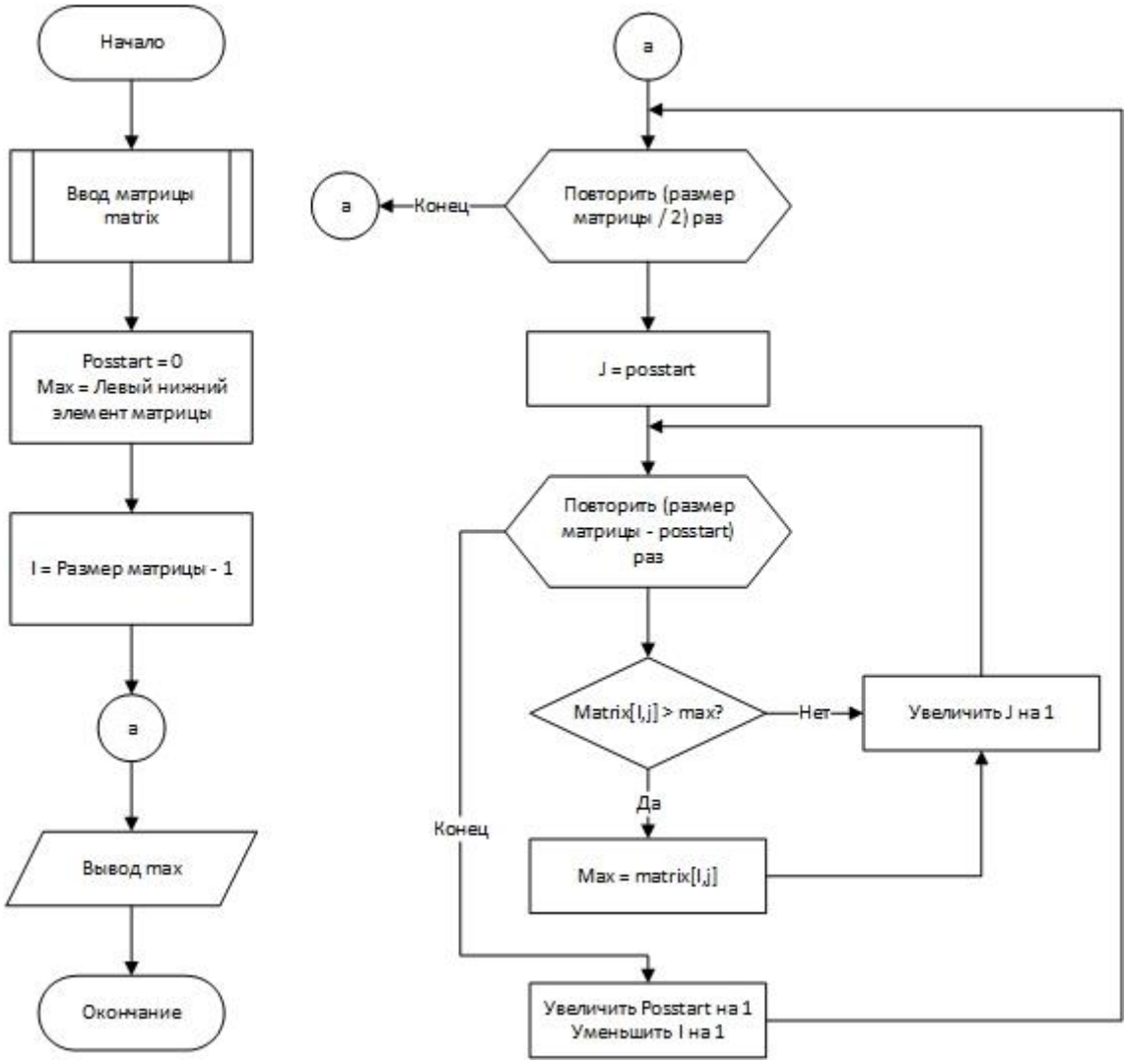


Рисунок 5.2. Блок-схема алгоритма для задачи 5

5.3. Тестирование

Для тестирования была составлена следующая таблица тестов (табл. 5.1). Она содержит входные данные, ожидаемые и реальные результаты работы программы с такими данными:

Таблица 5.1. Таблица тестов для задачи 5

№	Входные данные	Ожидаемые результаты	Реальные результаты
T1	ываыва	Ошибка ввода	Ошибка ввода

№	Входные данные	Ожидаемые результаты	Реальные результаты
Т2	-23	Ошибка ввода	Ошибка ввода
Т3	5 64 18 33 -50 -40 73 62 48 -84 -39 87 97 7 -37 8 -13 91 -97 22 -7 -70 -1 -71 -85 17	91	91
Т4	1 -1	-1	-1
Т5	4 33 -60 -54 96 -22 46 36 67 -79 37 -30 -9 28 61 52 -32	61	61

Данные тесты покрывают критерии черного ящика, представленные в таблице Е.1 в приложении Е.

6. Задача 6

Ввести a_1, a_2, a_3. Построить последовательность чисел $a_k = 13 * a_{k-1} - 10 * a_{k-2} + a_{k-3}$. Построить N элементов последовательности проверить, образуют ли элементы, стоящие на четных местах, возрастающую подпоследовательность.

6.1. Анализ

Так как последовательность задана рекуррентной формулой, то вычислять члены последовательности следует с помощью рекурсивной функции. Данная функция должна массив, в котором храниться последовательность, число n, отвечающее за номер элемента последовательности. На основе введенных членов последовательности вычисляется следующий член $a_к$. Если его номер меньше 3, то следует вернуть полученное значение, если же номер $a_к$ больше n, то необходимо найти предыдущий член.

Длина последовательности N может быть меньше 3 в том случае, если M меньше одного из первых трех элементов. Данный момент также следует учесть при проектировании.

Для того чтобы определить: образуют ли элементы, стоящие на четных местах, образуют возрастающую последовательность необходимо реализовать метод.

6.2. Проектирование

Согласно проведенному анализу, была сделана блок-схема программы (см. рис. 6.1)

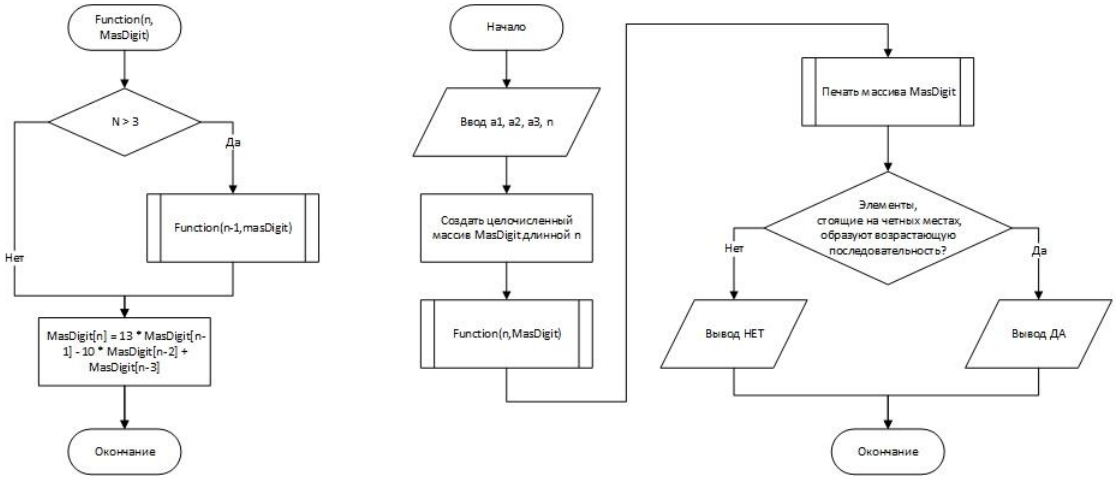


Рисунок 6.1. Блок-схема алгоритма расчета последовательности по рекуррентной формуле

Блок-схема алгоритма программы представляет значительно в меньшем масштабе, поэтому была вынесена в приложение F (см. рис. F.1)

6.3. Тестирование

Для тестирования была составлена следующая таблица тестов (см. табл. 6.1 и 6.2). Она содержит входные данные, ожидаемые и реальные результаты работы программы с такими данными:

Таблица 6.1. Таблица тестов для задачи 6

№	Входные данные	Ожидаемые результаты	Реальные результаты
T1	y	Ошибка ввода	Ошибка ввода
T2	6 y	Ошибка ввода	Ошибка ввода
T3	6 -8 h	Ошибка ввода	Ошибка ввода
T4	4 5,6 3 t	Ошибка ввода	Ошибка ввода
T5	0 0 0 8	Последовательность бесконечна, состоит из нулей	Последовательность бесконечна, состоит из нулей
T6	9 6 3 5	Последовательность: 9 6 7 40 456 Длина: 5 Элементы, стоящие на четных местах, образуют возрастающую последовательность	Последовательность: 9 6 7 40 456 Длина: 5 Элементы, стоящие на четных местах, образуют возрастающую последовательность

Данные тесты покрывают критерии черного ящика, представленные в таблице F.1 в приложении F.

7. Задача 7

Сгенерировать все сочетания из N элементов по K с повторениями и выписать их в лексикографическом порядке.

7.1. Анализ

В первую очередь следует уточнить, что при выполнении данного задания предполагалось, что пользователь введет сначала длину нужного ему алфавита, а после – длину каждого кодирующего слова.

Сочетаниями с повторениями называются наборы по M элементов, в которых каждый элемент множества N может участвовать несколько раз. При этом на соотношение значений M и N не накладывается никаких ограничений, а общее количество сочетаний с повторениями составляет $\frac{(N+M-1)!}{(N-1)!*M!}$.

Лексикографический порядок – порядок сочетаний, задаваемый следующими правилами: все сочетания храниться в алфавитном порядке. В данном случае от 1 до 9.

Следует отметить, что все вводимые в данной задаче данные являются натуральными числа, то есть необходимо ввести проверку на правильность данного ввода.

7.2. Проектирование

Блок-схема алгоритма программы представляет значительно меньший интерес, поэтому была вынесена в приложение G (см. рис. G.1)

Блок-схема функции создания сочетаний с повторениями программы представлена на рисунке 7.1:

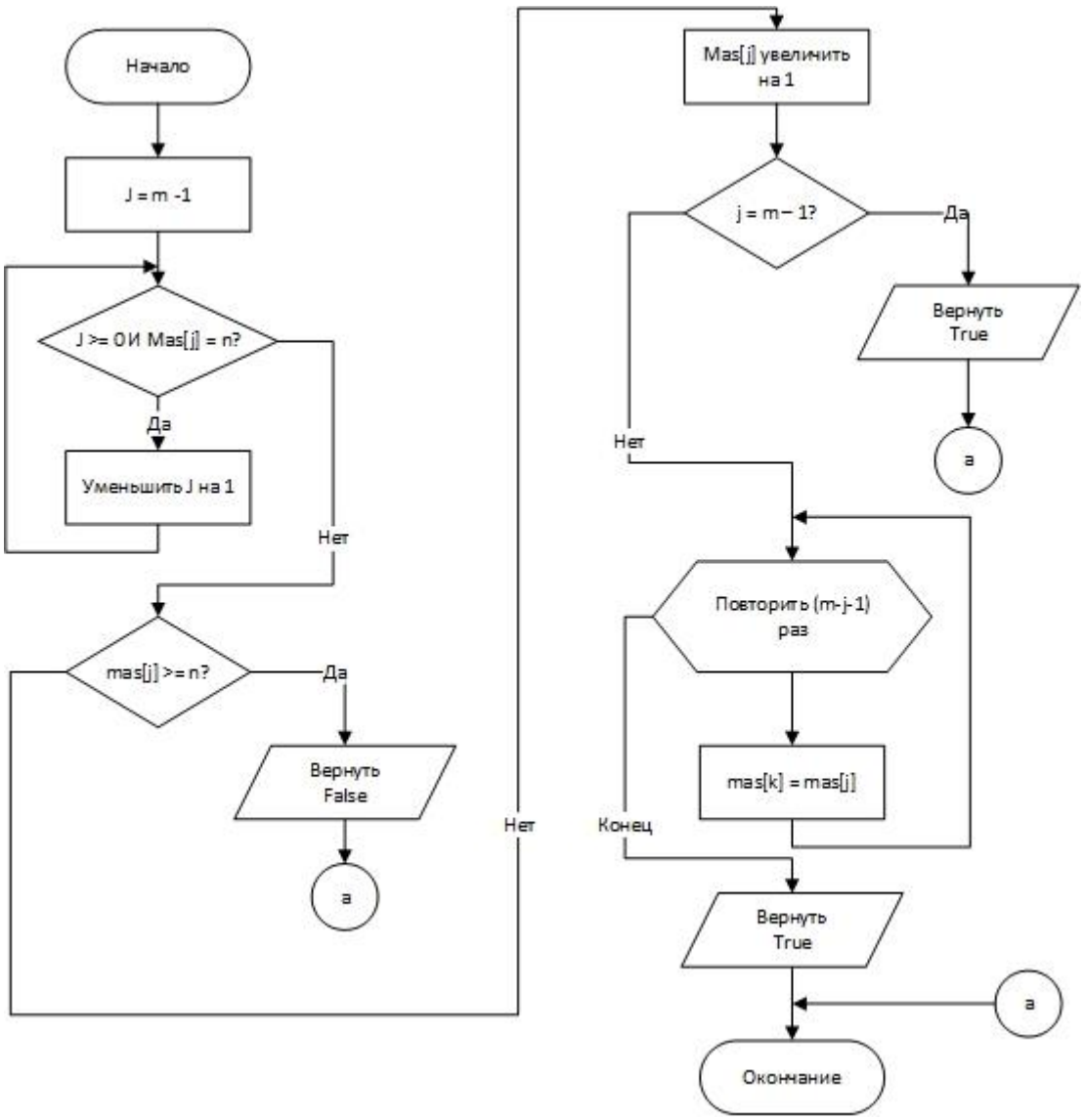


Рисунок 6.2. Блок-схема алгоритма создания сочетаний для задачи 7

7.3. Тестирование

Для тестирования была составлена следующая таблица тестов (см. табл. 7.1).:

Таблица 6.2. Таблица тестов для задачи 7

№	Входные данные	Ожидаемые результаты	Реальные результаты
T1	y	Ошибка ввода	Ошибка ввода
T2	-2	Ошибка ввода	Ошибка ввода
T3	2 3	1: 1 1 2 2: 1 2 2 3: 2 2 2	1: 1 1 2 2: 1 2 2 3: 2 2 2
T4	4 3	1: 1 1 2 2: 1 1 3 3: 1 1 4	1: 1 1 2 2: 1 1 3 3: 1 1 4

		4: 1 2 2 5: 1 2 3 6: 1 2 4 7: 1 3 3 8: 1 3 4 9: 1 4 4 10: 2 2 2 11: 2 2 3 12: 2 2 4 13: 2 3 3 14: 2 3 4 15: 2 4 4 16: 3 3 3 17: 3 3 4 18: 3 4 4 19: 4 4 4	4: 1 2 2 5: 1 2 3 6: 1 2 4 7: 1 3 3 8: 1 3 4 9: 1 4 4 10: 2 2 2 11: 2 2 3 12: 2 2 4 13: 2 3 3 14: 2 3 4 15: 2 4 4 16: 3 3 3 17: 3 3 4 18: 3 4 4 19: 4 4 4

В процессе тестирования была обнаруженная и исправлена ошибка с неправильной проверкой ввода чисел: проверку проходили все целые числа, а не только натуральные.

Данные тесты покрывают критерии черного ящика, представленные в таблице G.1 в приложении G.

8. Задача 8

Эйлеров граф задан матрицей смежностей. Найти в нем какой-либо эйлеров цикл.

8.1. Анализ

Сначала проверим, существует ли эйлеров путь. Затем найдём все простые циклы и объединим их в один — это и будет эйлеровым циклом. Если граф таков, что эйлеров путь не является циклом, то, добавим недостающее ребро, найдём эйлеров цикл, потом удалим лишнее ребро.

Чтобы проверить, существует ли эйлеров путь, нужно воспользоваться следующей теоремой. Эйлеров цикл существует тогда и только тогда, когда степени всех вершин чётны. Эйлеров путь существует тогда и только тогда, когда количество вершин с нечётными степенями равно двум (или нулю, в случае существования эйлерова цикла).

Кроме того, конечно, граф должен быть достаточно связным (т.е. если удалить из него все изолированные вершины, то должен получиться связный граф).

Искать все циклы и объединять их будем одной не рекурсивной процедурой:

```
stack St;
в St кладём любую вершину (стартовая вершина);
пока St не пустой
       пусть V - значение на вершине St;
       если степень(V) = 0, то
              добавляем V к ответу;
              снимаем V с вершины St;
       иначе
              находим любое ребро, выходящее из V;
              удаляем его из графа;
              второй конец этого ребра кладём в St;
```

Также необходимо ввести проверку на правильность ввода. В матрице смежности должны быть только нули и единицы.

8.2. Проектирование

Сначала программа проверяет степени вершин: если вершин с нечётной степенью нет, то в графе есть эйлеров цикл, если есть 2 вершины с нечётной степенью, то в графе есть только эйлеров путь (эйлерова цикла нет), если же таких

вершин больше 2, то в графе нет ни эйлерова цикла, ни эйлерова пути. Чтобы найти эйлеров путь (не цикл), поступим таким образом: если V1 и V2 — это две вершины нечётной степени, то просто добавим ребро (V1,V2), в полученном графе найдём эйлеров цикл (он, очевидно, будет существовать), а затем удалим из ответа "фиктивное" ребро (V1,V2). Эйлеров цикл будем искать в точности так, как описано выше (не рекурсивной версией), и заодно по окончании этого алгоритма проверим, связный был граф или нет (если граф был не связный, то по окончании работы алгоритма в графе останутся некоторые рёбра, и в этом случае нам надо вывести -1). Наконец, программа учитывает, что в графе могут быть изолированные вершины. Блок-схема тела программы представлена на рисунке 8.1:

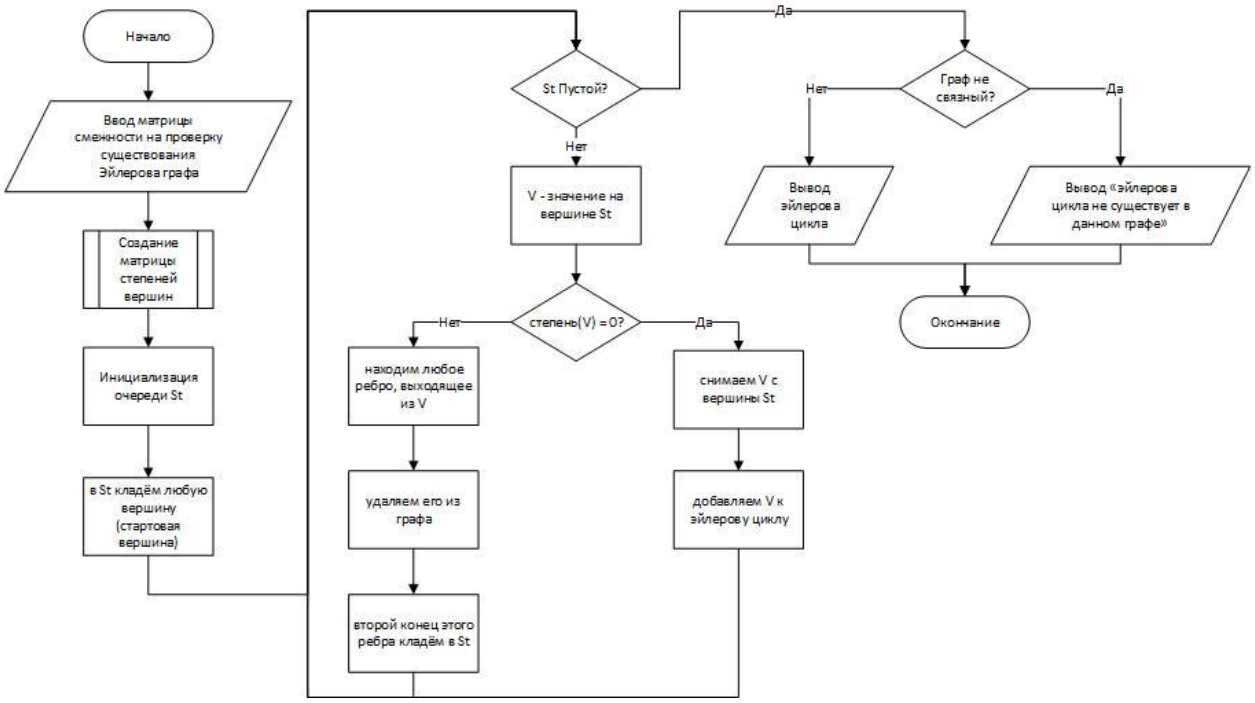


Рисунок 6.3. Блок-схема алгоритма для задачи 8

8.3. Тестирование

Для тестирования была составлена следующая таблица тестов (см. табл. 8.2). Она содержит входные данные, ожидаемые и реальные результаты работы программы с такими данными:

Таблица 6.3. Таблица тестов для задачи 8

№	Входные данные	Ожидаемые результаты	Реальные результаты
T1	5 6 п 8 7 5	Ошибка ввода	Ошибка ввода

№	Входные данные	Ожидаемые результаты	Реальные результаты
Т2	5 0 1 0 0 1 1 0 1 0 0 0 1 0 1 0 0 0 1 0 1 1 0 0 1 0	5 => 4 => 3 => 2 => 1	5 => 4 => 3 => 2 => 1
Т3	5 0 1 0 0 0 1 0 1 0 0 0 1 0 1 0 0 0 1 0 1 0 0 0 1 0	Данный граф не содержит эйлеров цикл	Данный граф не содержит эйлеров цикл

Данные тесты покрывают критерии черного ящика, представленные в таблице Н.1 в приложении Н.

9. Задача 9

Данное задание необходимо выполнить без использования коллекций языка C#.

В программе построен линейный список. Напишите процедуру, создающую два новых линейных списка: список, включающий все положительные значения из элементов исходного списка, и список, включающий все отрицательные значение из элементов исходного списка; при включении элементов в новые списки, они удаляются из исходного списка.

9.1. Анализ

Связный список (LinkedList) представляет набор связанных узлов (Node), каждый из которых хранит собственно данные и ссылку на следующий узел. В реальной жизни связный список можно представить в виде поезда, каждый вагон которого может содержать некоторый груз или пассажиров и при этом может быть связан с другим вагоном.

Со списком из нескольких элементов без дополнительных функций работать можно, но очень неудобно. Поэтому следует создать также коллекцию для работы со списком. Ее свойствами будут первый и последний элемент списка, а также будут реализованы методы, данные в задании.

Также N, до которого необходимо заполнить список, очевидно должно быть целым и больше или равно 1.

9.2. Проектирование

В классе элемента списка (LinkedList) необходимо, помимо свойств, описать инициализатор без аргументов и с вводом значения. Также реализовать функцию печати списка для удобного строкового представления и добавить метод для вывода элемента в консоль. Подробнее данные функции и свойства описаны в таблицах I.1 и I.2 Приложения I.

В классе коллекции (LinkedList), дополнительно к описанным выше свойствам и методам, добавить метод добавления элемента в список, так как в линейном списке это требует некоторого изменения ссылок, а значит добавление

такого метода облегчит написание кода с данным классом. Также необходимо реализовать метод для удаления элемента из списка. Подробнее данные функции описаны в таблицах I.3 и I.4 Приложения I.

Блок-схема для основного тела программы представлена на рисунке 9.1:

Рисунок 9.1. Блок-схема алгоритма для задачи 9

9.3. Тестирование

Для тестирования была составлена следующая таблица тестов (см. табл. 9.1). Она содержит входные данные, ожидаемые и реальные результаты работы программы с такими данными:

Таблица 9.1. Таблица тестов для задачи 9

№	Входные данные	Ожидаемые результаты	Реальные результаты
T1	y	Ошибка ввода	Ошибка ввода
T2	9,8	Ошибка ввода	Ошибка ввода
T3	9 -44 -58 98 98 -37	Оригинальный список: Полученный Positive список: 98 98 Полученный Positive список: -44 -58 -37	Оригинальный список: Полученный Positive список: 98 98 Полученный Positive список: -44 -58 -37

Данные тесты покрывают критерии черного ящика, представленные в таблице I.5 в приложении I.

10. Задача 10

Данное задание необходимо выполнить без использования коллекций языка C#.

Даны натуральное число n, действительные числа $x_1 \dots, x_n$. Получить последовательность $x_1 - x_n, \dots, x_{n-1} - x_n$. Для решения этой задачи полезен список, изображенный на рис. 10.1.

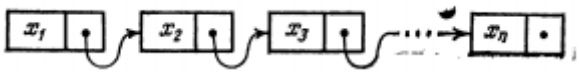


Рисунок 10.1. Список

10.1. Анализ

Связный список (LinkedList) представляет набор связанных узлов (Node), каждый из которых хранит собственно данные и ссылку на следующий узел. В реальной жизни связный список можно представить в виде поезда, каждый вагон которого может содержать некоторый груз или пассажиров и при этом может быть связан с другим вагоном.

Со списком из нескольких элементов без дополнительных функций работать можно, но очень неудобно. Поэтому следует создать также коллекцию для работы со списком. Ее свойствами будут первый и последний элемент списка, а также будут реализованы методы, данные в задании.

Также N, до которого необходимо заполнить список, очевидно должно быть целым и больше или равно 1.

10.2. Проектирование

В классе элемента списка (LinkedList) помимо описанных свойств следует реализовать инициализацию без аргументов, а также метод Print. Подробнее данные свойства и методы описаны в таблицах J.1 и J.2 Приложения J.

В классе коллекции (LinkedList), дополнительно к описанным выше свойствам и методам, добавить метод добавления элемента в список, так как в линейном списке это требует некоторого изменения ссылок, а значит добавление такого метода облегчит написание кода с данным классом.

Создание нового списка предполагает использовать цикл foreach, поэтому необходимо реализовать методы интерфейса IEnumerable. Подробнее данные свойства и методы описаны в таблицах J.1 и J.2 Приложения J. Само создание нового списка было вынесено в отдельную статическую функцию класса.

Блок-схема для основного тела программы со статической функций создания нового списка представлена на рисунке 10.2:

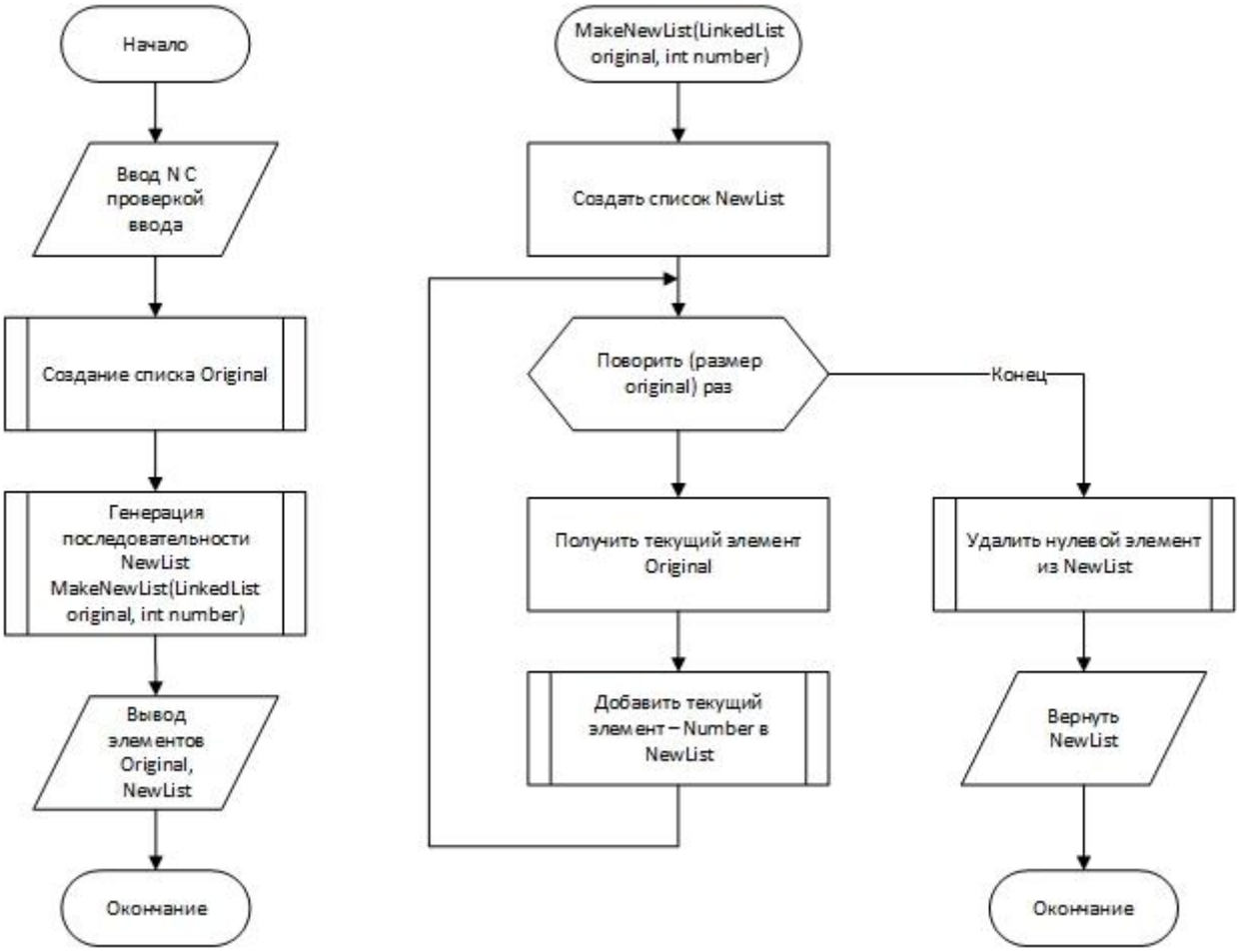


Рисунок 10.2. Блок-схема алгоритма для задачи 10

10.3. Тестирование

Для тестирования была составлена следующая таблица тестов (см. табл. 10.1). Она содержит входные данные, ожидаемые и реальные результаты работы программы с такими данными:

Таблица 10.1. Таблица тестов для задачи 10

№	Входные данные	Ожидаемые результаты	Реальные результаты
T1	апап	Ошибка ввода	Ошибка ввода

№	Входные данные	Ожидаемые результаты	Реальные результаты
T2	-2	Ошибка ввода	Ошибка ввода
T3	1.23	Ошибка ввода	Ошибка ввода
T4	6 1 2 3 4 5 6	-5 -4 -3 -2 -1	-5 -4 -3 -2 -1

Данные тесты покрывают критерии черного ящика, представленные в таблице J.3 в приложении J.

11. Задача 11

Зафиксируем натуральное k и перестановку чисел 1, ..., k (ее можно задать с помощью последовательности натуральных чисел p_1, ..., p_k, в которую входит каждое из чисел 1, ..., k). При шифровке в исходном тексте к каждой из последовательных групп по k символов применяется зафиксированная перестановка. Пусть k = 4 и перестановка есть 3. 2, 4, 1. Тогда группа символов s_1 s_2, s_3, s_4 заменяется на s_3, s_2, s_4, s_1. Если в последней группе меньше четырех символов, то к ней добавляются пробелы. Пользуясь изложенным способом:

a) зашифровать данный текст;
b) расшифровать данный текст.

11.1. Анализ

Исходя из условия, для удобства кодирования и декодирования необходимо идти по тексту с помощью каретки длинной k. Пока не достиг ли конца текста необходимо получить подстроку, входящую в каретку. Затем зашифровать данный участок текста и записать результат в новый текст. После чего передвинуть каретку и повторить алгоритм.

Функция расшифрования зашифрованного текста является рекурсивной функций, так как чтобы расшифровать текст, необходимо расшифровать его k-1.

Длина каретки k – натуральное число, поэтому необходимо реализовать проверку на входные данные.

Так же стоит отметить, что в случае пустого входного текста программа должна выдавать ошибку.

11.2. Проектирование

Согласно проведенному анализу, была составлена блок-схема для решения задачи (см. Приложение К, рис. К.1).

11.3. Тестирование

Для тестирования была составлена следующая таблица тестов (см. табл. 11.1, 11.2). Она содержит входные данные, ожидаемые и реальные результаты работы программы с такими данными:

Таблица 10.2. Таблица тестов для задачи 11

№	Входные данные	Ожидаемые результаты	Реальные результаты
T1	аывп	Ошибка ввода	Ошибка ввода
T2	4 12341234	Шифр 4123 Зашифрованный текст 34123412 Расшифрованный текст 12341234	Шифр 4123 Зашифрованный текст 34123412 Расшифрованный текст 12341234

Данные тесты покрывают критерии черного ящика, представленные в таблице К.1 в приложении К.

12. Задача 12

Выполнить сравнение двух предложенных методов сортировки одномерных массивов, содержащих *n* элементов, по количеству пересылок и сравнений.

Для этого необходимо выполнить программную реализацию двух методов сортировки, включив в нее подсчет количества пересылок (т.е. перемещений элементов с одного места на другое) и сравнений.

Провести анализ методов сортировки для трех массивов: упорядоченного по возрастанию, упорядоченного по убыванию и неупорядоченного.

Все три массива следует отсортировать обоими методами сортировки.

Найти в литературе теоретические оценки сложности каждого из методов и сравнить их с оценками, полученными на практике.

Сделать выводы о том, насколько отличаются теоретические и практические оценки количества операций, объяснить почему это происходит. Сравнить оценки сложности двух алгоритмов.

Предложенные методы сортировки:

1. Сортировка с помощью двоичного дерева.
2. Сортировка слиянием.

12.1. Анализ

Бинарное (двоичное) дерево поиска – это бинарное дерево, для которого выполняются следующие дополнительные условия (свойства дерева поиска):

оба поддерева – левое и правое, являются двоичными деревьями поиска;

у всех узлов левого поддерева произвольного узла X значения ключей данных меньше, чем значение ключа данных самого узла X;

у всех узлов правого поддерева произвольного узла X значения ключей данных не меньше, чем значение ключа данных узла X.

Данные в каждом узле должны обладать ключами, на которых определена операция сравнения меньше. Для сортировки с помощью дерева исходная сортируемая последовательность представляется в виде структуры данных «дерево».

Корнем дерева будет начальный элемент последовательности. Далее все элементы, меньшие корневого, располагаются в левом поддереве, все элементы, большие корневого, располагаются в правом поддереве. Причем это правило должно соблюдаться на каждом уровне.

После того, как все элементы размещены в структуре «дерево», необходимо вывести их, используя инфиксную форму обхода.

Теоретическое количество пересылок в лучшем случае равно O(log(n)), в худшем случае – $O(n^2)$.

Сортировка слиянием предполагает разделение массива на более мелкие части и их сортировка, а после – слияние обратно в единый массив. В реализации это выглядит как деление массива пополам и вызов данной функции для половинок массива, если они длиннее одного элемента, и дальнейшее слияние данных массивов в правильном порядке.

Теоретическое количество сравнений равно количеству перестановок, не зависит от случая и вычисляется по рекуррентной формуле C(1)=0, C(n) = C(n/2) + n.

12.2. Проектирование

Блок-схемы алгоритмов сортировки пузырьком и сортировки слиянием представлены на рисунках 12.1 и 12.2 соответственно.

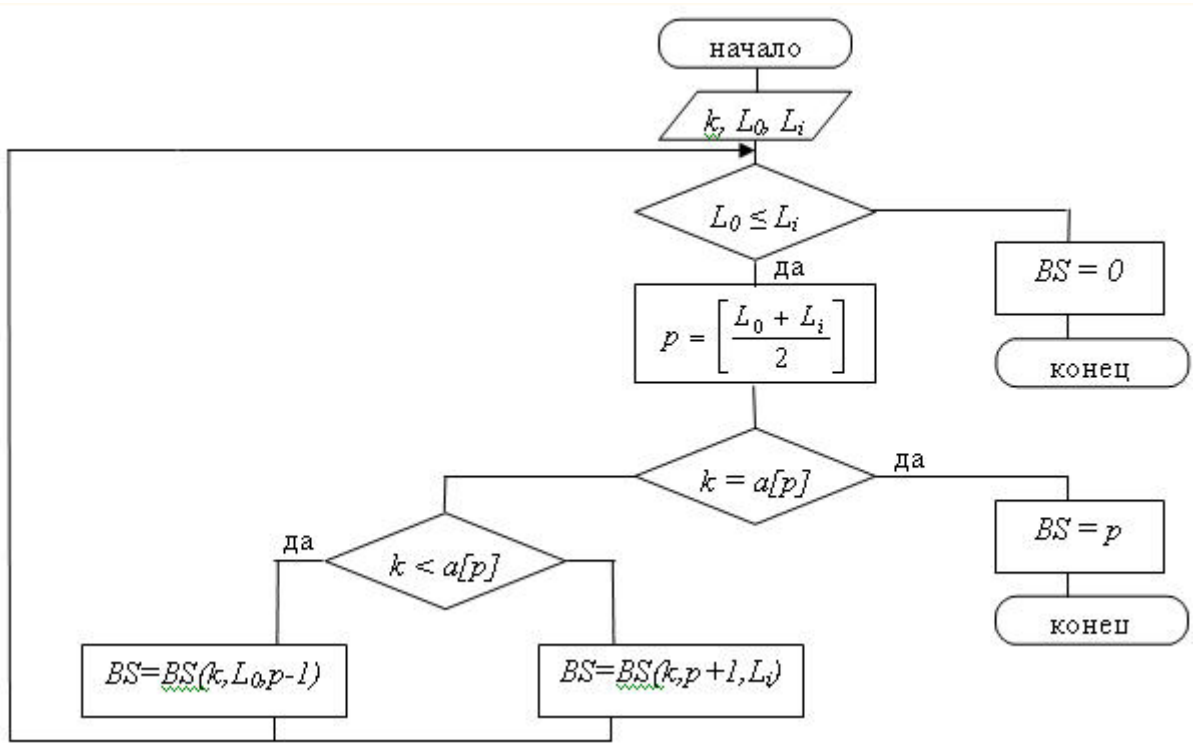


Рисунок 10.3. Блок-схема алгоритма сортировки бинарным деревом

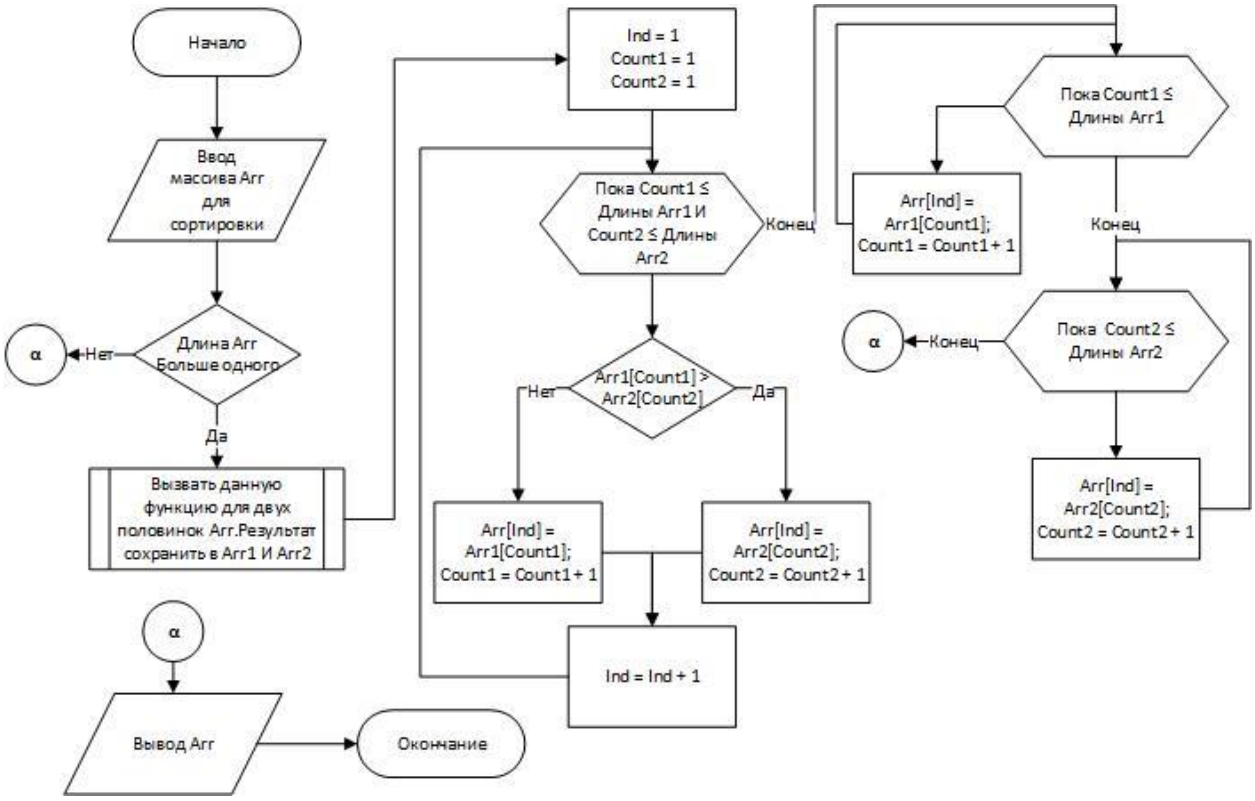


Рисунок 10.4. Блок-схема алгоритма сортировки слиянием

Для подсчета количества сравнений и перестановок необходимо добавить переменные-счетчики и увеличивать из на один после каждого сравнения или перестановки соответственно.

12.3. Тестирование

Для тестирования была составлена следующая таблица тестов (см. табл. 11.1, 11.2). Она содержит входные данные, ожидаемые и реальные результаты работы программы с такими данными:

Таблица 10.3. Таблица тестов для задачи 12

№	Входные данные	Ожидаемые результаты	Реальные результаты
T1	3 5 a	Ошибка ввода	Ошибка ввода
T2	1 5 6 8 9 34 56 78	1 5 6 8 9 34 56 78 1 5 6 8 9 34 56 78	1 5 6 8 9 34 56 78 1 5 6 8 9 34 56 78
T3	99 65 43 21 7 5 3 2 1	1 2 3 5 7 21 43 65 99 1 2 3 5 7 21 43 65 99	1 2 3 5 7 21 43 65 99 1 2 3 5 7 21 43 65 99
T4	8 3 99 5 73 5 79 1 31	1 3 5 5 8 31 73 79 99 1 3 5 5 8 31 73 79 99	1 3 5 5 8 31 73 79 99 1 3 5 5 8 31 73 79 99
T5		Массив пуст	Массив пуст
T6	6	6 6	6 6

Данные тесты покрывают критерии черного ящика, представленные в таблице L.1 в приложении L.

12.4. Сравнение алгоритмов сортировки

Практические данные по количеству пересылок и количеству сравнений при сортировке представлены в таблице 12.2:

Таблица 10.4. Сравнение двух сортировок

	Число сравнений		Число пересылок	
	Деревом	**Слиянием**	**Деревом**	**Слиянием**
Упорядоченный по возрастанию массив	45	34	18	34
Упорядоченный по убыванию массив	90	34	36	34
Неупорядоченный массив	135	34	54	34

Теоретические данные возможно найти для худших и лучших результатов только для сортировки слиянием, и для обеих сортировок они совпадают с реальными результатами.

Неупорядоченный массив при сортировке пузырьком имеет 135 сравнений, как и любой иной случай сортировки массива из 10 элементов, так как при данной сортировке количество сравнений является константой при определенном N.

При сортировке слиянием количество пересылок и сравнений является константой при определенном N, поэтому в случае неупорядоченного массива их число не изменилось.

При сравнении двух сортировок заметно, что даже на таком маленьком массиве число сравнений при сортировке слиянием заметно меньше, чем при сортировке деревом. Число пересылок зависит от случая, так как сортировка слиянием не имеет худшего случая, в отличии от сортировки деревом. В худшем случае двоичной сортировки производится больше пересылок, чем при сортировке слиянием.

Теоретические данные показывают, что чем более крупный набор данных необходимо отсортировать, тем заметнее будет превосходство сортировки слиянием над сортировкой деревом.

Заключение

Подводя итоги, были спроектированы, разработаны и протестированы двенадцать задач; развиты и закреплены практические навыки построения и описания алгоритмов для решения задач из разных предметных областей (численные методы, дискретная математика, структуры данных и др). Были развиты и закреплены практические навыки использования языков высокого уровня и современных сред разработки для реализации построенных алгоритмов. Были развиты и закреплены практические навыки объектно-ориентированного программирования. Были развиты практические навыки оформления отчетов о проделанной работе и публичного выступления с защитой проекта.

По окончании работы получены приложения выполняющие поставленные задачи. Таким образом, все задачи были выполнены, и как следствие – достигнута цель.

Библиографический список

1. Абрамов С.А. Задачи по программированию // С.А. Абрамов, Г.Г. Гнездилова, Е.Н. Капустина, М.И. Селюн — М.: Наука, 1988. — 578 с.
2. Плаксин М. А. Тестирование и отладка программ для профессионалов будущих и настоящих / М. Плаксин - М.: Бином, 2013. – 20-45 с.
3. Терехов А.Н. Технология программирования: учебное пособие. / А.Н. Терехов: – 2-е изд. – М.: БИНОМ. Лаборатория знаний, 2012. — 130 с.
4. Троелсен Э. Язык программирования C# 2010 и платформа .NET 4 // пер. с англ. – М.: Издательский дом "Вильямс", 2012. — 1412 с.

Приложение А. К задаче 1

Таблица 10.5. Черный ящик к задаче 1

№	Проверяемая ситуация	Тесты		
		T1	**T2**	**T3**
1	Классы входных данных			
1.1	Отсутствие файла input.txt	+		
1.2	Отсутствие файла output.txt	+		
1.3	Файл input.txt пуст		+	
1.4	Файл output.txt пуст		+	
1.5	Файл output.txt не пуст			+
1.6	В файле input.txt есть символ:			
1.6.1	Целые числа			+
1.6.2	.			+
3	Классы выходных данных			
3.1	Количество масок подсетей, в которых лежит пара ip-адресов	+	+	+

Листинг программы:

```
using System;
using System.Collections.Generic;
using System.Text;

namespace Task12_v4_5
{
    public class MergeSortArray
    {
        // Метод для слияния массивов
        public static void Merge(int[] array, int lowIndex, int middleIndex, int
highIndex)
        {
            var left = lowIndex;
            var right = middleIndex + 1;
            var tempArray = new int[highIndex - lowIndex + 1];
            var index = 0;

            while ((left <= middleIndex) && (right <= highIndex))
            {
                if (array[left] < array[right])
                {
                    tempArray[index] = array[left];
                    left++;
                }
                else
                {
                    tempArray[index] = array[right];
                    right++;
                }
```

```
                index++;
            }

            for (var i = left; i <= middleIndex; i++)
            {
                tempArray[index] = array[i];
                index++;
            }

            for (var i = right; i <= highIndex; i++)
            {
                tempArray[index] = array[i];
                index++;
            }

            for (var i = 0; i < tempArray.Length; i++)
            {
                array[lowIndex + i] = tempArray[i];
            }
        }

        // Сортировка слиянием
        public static int[] MergeSort(int[] array, int lowIndex, int highIndex)
        {
            if (lowIndex < highIndex)
            {
                var middleIndex = (lowIndex + highIndex) / 2;
                MergeSort(array, lowIndex, middleIndex);
                MergeSort(array, middleIndex + 1, highIndex);
                Merge(array, lowIndex, middleIndex, highIndex);
            }

            return array;
        }

        public static int[] MergeSort(int[] array)
        {
            return MergeSort(array, 0, array.Length - 1);
        }

    }
}
```

Приложение В. К задаче 2

Таблица 10.6. Черный ящик к задаче 2

№	Проверяемая ситуация	Тесты		
		Т1	**Т2**	**Т3**
1	Классы входных данных			
1.1	Отсутствие файла input.txt	+		
1.2	Отсутствие файла output.txt	+		
1.3	Файл input.txt пуст		+	
1.4	Файл output.txt пуст		+	
1.5	Файл output.txt не пуст			+
1.6	В файле input.txt первая строка это:			
1.6.1	число			+
1.6.2	Команды			+
2	Классы выходных данных			
2.1	Ошибка: в первой строке файла не число		+	
2.2	Число			+
2.3	Файл пуст	+		

Листинг программы:

```
#include <fstream>
#include <set>
#include <vector>
#include <iostream>

using namespace std;

void print(const vector<set<size_t>>& sets, size_t key, ostream& ost)
{
    // Если список не содерджит множество с адресом key
    if (sets.size() <= key || sets[key].empty())
    {
        ost << "-1" << endl;
        return;
    }

    for (auto value : sets[key])
    {
        ost << value << " ";
    }

    ost << endl;
}

int main()
{
    ifstream ifst("input.txt");
    ofstream ofst("output.txt");

    size_t n, m, k;
```

```
    vector<set<size_t>> set_values; // Вектор Список множеств
    vector<set<size_t>> value_sets; // Вектор Список элементов ссылками на множества, в
которых они встречаются

    // Ввод максимального числа множества, количетсва множеств, количества операций
    ifst >> m >> n >> k;

    cout << "Программа получла следующее данные" << endl
         << "Максимальное число в множестве - " << m << endl
         << "Количество множеств - " << n << endl
         << "Количество операций - " << k << endl;

    set_values.resize(n + 1);
    value_sets.resize(m + 1);

    cout << "Операции" << endl;
    for (size_t i = 0; i < k; ++i)
    {
        string operation;
        size_t element, set;

        ifst >> operation;

        if (operation == "ADD")
        {
            ifst >> element >> set;

            if (element >= 1 && element <= m)
            {
                set_values[set].insert(element);
                value_sets[element].insert(set);
                cout << "Операция ADD " << element << " " << set << " была выполнена."<<
endl;
            }
            else
            {
                cout << "Операция ADD " << element << " " << set << " не была выполнена,
так как введеый элемент превышает максимальное разрешенное." << endl;
            }
        }
        if (operation == "LISTSET")
        {
            ifst >> set;
            print(set_values, set, ofst);
            cout << "Операция LISTSET " << set << " была выполнена." << endl;
        }
        if (operation == "LISTSETSOF")
        {
            ifst >> element;
            print(value_sets, element, ofst);
            cout << "Операция LISTSETSOF " << element << " была выполнена." << endl;
        }
    }

    cout << "Результаты выполнения операций хранятся в файле output.txt" << endl;
    system("Pause");
}
```

Приложение С. К задаче 3

Таблица 10.7. Черный ящик к задаче 3

№	Проверяемая ситуация	Тесты					
		T1	**T2**	**T3**	**T4**	**T5**	**T6**
1	Классы входных данных						
1.1	Число		+	+	+	+	+
1.2	Не число	+					
2	Классы выходных данных						
2.1	Ошибка ввода	+					
2.2	Значение функции		+	+	+	+	+
3	входных данные (x, y)						
3.1	x = 0, y = 0, f(x) <= y		+				
3.2	x = -0.5, y = -1, f(x) <= y			+			
3.3	x = 0.5, y = -1, f(x) <= y				+		
3.4	x = 0, y 0 -1, f(x) <= y					+	
3.5	x = 4, y =0, f(x) > y						+

Листинг программы:

```
using System;

namespace Task3_v59z
{
    class DotAccessory
    {
        static bool Check(double x, double y)
        {
            if (x >= -1 && x <= 0 && y <= 3 * x + 2 || x >= 0 && x <= 1 && y <= -3 * x +
2)
                return true;
            return false;
        }

        static double numberInput()
        {
            double number;
            while(!double.TryParse(Console.ReadLine(), out number))
            {
                Console.WriteLine("Ошибка. Вы не верно ввели координату. Повторите
ввод");
            }
            return number;
        }
        static void Main(string[] args)
        {
            Console.WriteLine("Ввелите координату X:");
            double x = numberInput();

            Console.WriteLine("Ввелите координату Y:");
            double y = numberInput();

            if (Check(x, y))
            {
```

```
                Console.WriteLine($"Точка A({x};{y}) принедлежит закрашенной области");
            }
            else
            {
                Console.WriteLine($"Точка A({x};{y}) не принедлежит закрашенной
области");
            }
        }
    }
}
```

Приложение D. К задаче 4

Таблица 10.8. Черный ящик к задаче 4

№	Проверяемая ситуация	Тесты	
		T1	T2
1	Классы входных данных		
1.1	Строка	+	
1.2	Действительные числа		+
2	Классы выходных данных		
2.1	Ошибка ввода	+	
2.2	Комплексное число		+

Листинг программы:

```
using System;

namespace Task4_v439
{
    class ComplexNumber
    {
        public struct Complex
        {
            public double re;
            public double im;
        }

        // Печать комплексного числа
        static void Print(Complex complex)
        {
            if (complex.re < 0)
            {
                Console.WriteLine(complex.re + "-" + complex.im + "i");
            }
            else
            {
                Console.WriteLine(complex.re + "+" + complex.im + "i");
            }
        }

        // Умножение комплексного на целое
        static void UmComplex(Complex complex, out Complex complex1, int k)
        {
            complex1.re = complex.re * k;
            complex1.im = complex.im * k;
        }

        // Деление комплексного на целое
        static void DelComplex(Complex complex, out Complex complex1, int k)
        {
            complex1.re = complex.re / k;
            complex1.im = complex.im / k;
        }

        // Вычитание из комплексного целого
        static void VchComplex(ref Complex complex, int k)
        {
            complex.re = complex.re - k;
```

```
}

// Сложение комплексных
static void Summa(ref Complex complex,Complex complex1)
{
    complex.re = complex.re + complex1.re;
    complex.im = complex.im + complex1.im;
}

// Разность комплексных
static void Razn(ref Complex complex, Complex complex1)
{
    complex.re = complex.re - complex1.re;
    complex.im = complex.im - complex1.im;
}

// Умножение комплексных
static void Proizved(out Complex complex, Complex complex1, Complex complex2)
{
    complex.re = complex1.re * complex2.re - complex1.im * complex2.im;
    complex.im = complex1.im * complex2.re + complex1.re * complex2.im;
}

// Ввод комплексного числа
static Complex complexInput()
{
    Complex complex = new Complex();
    Console.WriteLine("Введите действительную часть комплексного числа:");
    complex.re = numberInput();
    Console.WriteLine("Введите мнимую часть комплексного числа:");
    complex.im = numberInput();

    return complex;
}

// Ввод действительного числа
static double numberInput()
{
    double number;
    while (!double.TryParse(Console.ReadLine(), out number))
    {
        Console.WriteLine("Ошибка. Вы не верно ввели число. Повторите ввод");
    }
    return number;
}

static void Main(string[] args)
{
    Console.WriteLine("ВВедите комплексное число u");
    Complex u = complexInput();

    Console.WriteLine("ВВедите комплексное число v");
    Complex v = complexInput();

    Console.WriteLine("ВВедите комплексное число w");
    Complex w = complexInput();

    Complex u1 = new Complex();
    Complex u2 = new Complex();
    Complex w1 = new Complex();
```

```
            Complex r = new Complex();

            UmComplex(u, out u1, 3);
            DelComplex(w, out w1, 2);
            Proizved(out r, u1, w1);
            Summa(ref r, w);
            Razn(ref r, u);
            UmComplex(u, out u2, 2);
            Summa(ref r, u2);
            VchComplex(ref r, 7);

            Console.Write("2u+(3u*w/2+w-u)-7=");
            Print(r);
        }
    }
}
```

Приложение Е. К задаче 5

Таблица 10.9. Черный ящик к задаче 5

№	Проверяемая ситуация	Тесты				
		T1	**T2**	**T3**	**T4**	**T5**
1	Классы входных данных					
1.1	Размер матрицы строка	+				
1.2	Размер матрицы < 0		+			
1.3	Размер матрицы >= 0			+	+	+
1.4	Максимум в середине			+		
1.5	Максимум в начале				+	
1.6	Максимум вквонце					+
2	Классы выходных данных					
2.1	Ошибка ввода	+	+			
2.2	Целое число			+	+	+

Листинг программы:

```
using System;

namespace Task5_v692g
{
    class MaxElemMatrix
    {
        static int sizeInput()
        {
            int number;
            Console.WriteLine("Введите размер матрицы");
            while (!int.TryParse(Console.ReadLine(), out number) || number < 0)
            {
                Console.WriteLine("Ошибка. Вы неверно ввели размер матрицы. Повторите
ввод");
            }
            return number;
        }

        static int[,] matrixGeneration(int size)
        {
            int[,] matrix = new int[size,size];
            Random random = new Random();

            for(int i = 0; i < size; i++)
            {
                for(int j = 0; j < size; j++)
                {
                    matrix[i, j] = random.Next(-100, 100);
                }
            }

            return matrix;
```

```
        }

        static void matrixPrint(int[,] matrix)
        {
            Console.WriteLine("Матрица:");
            for(int i = 0; i < matrix.GetLength(0); i++)
            {
                for (int j = 0; j < matrix.GetLength(0); j++)
                {
                    if (matrix[i, j] >= 0)
                    {
                        Console.Write(" {0}\t", matrix[i, j]);
                    }
                    else
                    {
                        Console.Write("{0}\t", matrix[i, j]);
                    }
                }
                Console.WriteLine();
            }
        }

        static int maxElemMatrix(int[,] matrix)
        {
            int max = matrix[matrix.GetLength(0)-1,0];
            int postart = 0;
            for(int i=matrix.GetLength(0)-1; i >= matrix.GetLength(0) / 2; i--)
            {
                for(int j = postart; j < matrix.GetLength(1) - postart; j++)
                {
                    if (matrix[i,j] > max)
                    {
                        max = matrix[i, j];
                    }
                }
                postart++;
            }
            return max;
        }
        static void Main(string[] args)
        {
            int size = sizeInput();
            int[,] matrix = matrixGeneration(size);
            Console.WriteLine("Матрица сгенерировна");
            matrixPrint(matrix);
            Console.WriteLine("Максимальное значение в нижнем треугольнике матрицы = " +
maxElemMatrix(matrix));

        }
    }
}
```

Приложение F. К задаче 6

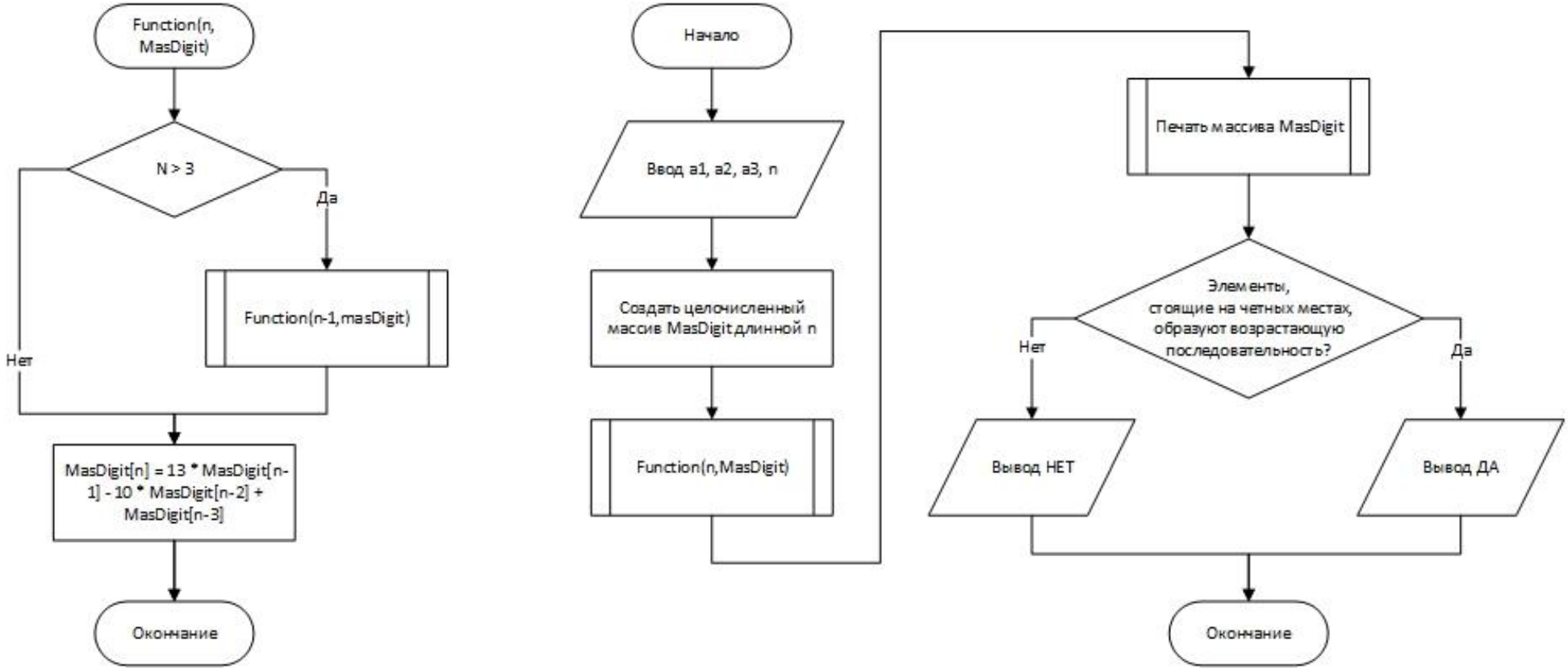


Рисунок 10.5. Блок-схема алгоритма для задачи 6

Таблица 10.10. Черный ящик к задаче 6

№	Проверяемая ситуация	Тесты					
		Т 1	**Т 2**	**Т 3**	**Т 4**	**Т 5**	**Т 6**
1	Классы входных данных						
1.1	A1 - число		+	+	+	+	+
1.2	A1 - не число	+					
1.3	A2 - число			+	+	+	+
1.4	A2 - не число		+				
1.5	A3 - число				+	+	+
1.6	A3 - не число			+			
1.7	N - число					+	+
1.8	N - не число				+		
1.9	A1=A2=A3=0					+	
1.10	A1 != 0 || A2 != 0 || A3 != 0						+
2	Классы выходных данных						
2.1	Ошибка ввода	+	+	+	+		
2.2	Последовательность, ее длина и элементы, стоящие на четных местах, образуют возрастающую последовательность					+	+

Листинг программы:

```
using System;

namespace Task6_v10
{
    class Sequence
    {
        static void Function(int n, long[] MasDigit)
        {
            if (n > 3)
            {
                Function(n - 1, MasDigit);
            }

            MasDigit[n] = 13 * MasDigit[n - 1] - 10 * MasDigit[n - 2] + MasDigit[n - 3];
        }
```

```
        static bool Parity(long[] masDigit)
        {
            for (int i = 3; i < masDigit.Length; i += 2)
            {
                if (masDigit[i-2] > masDigit[i])
                {
                    return false;
                }
            }

            return true;
        }

        static int NumberInput()
        {
            int number;
            while(!int.TryParse(Console.ReadLine(), out number))
            {
                Console.WriteLine("Ошибка. Вы ввели не целое число, повторите ввод");
            }
            return number;
        }

        static int SizeInput()
        {
            int number;
            while (!int.TryParse(Console.ReadLine(), out number) || number < 4)
            {
                Console.WriteLine("Ошибка. Вы неверно ввели размер последовательности,
повторите ввод");
            }
            return number;
        }

        static bool Check(int a1, int a2, int a3)
        {
            if (a1 == 0 && a2 == 0 && a3 == 0)
            {
                return false;
            }
            return true;
        }

        static void Main(string[] args)
        {
            Console.WriteLine("Ведите первый член последовательности");
            int a1 = NumberInput();
            Console.WriteLine("Ведите второй член последовательности");
            int a2 = NumberInput();
            Console.WriteLine("Ведите член член последовательности");
            int a3 = NumberInput();
            Console.WriteLine("Ведите размер последовательности больше 3");
            int n = SizeInput();

            if (Check(a1, a2, a3))
            {

                long[] MasDigit = new long[n];
                MasDigit[0] = a1;
```

```
                MasDigit[1] = a2;
                MasDigit[2] = a3;
                Function(n - 1, MasDigit);

                Console.WriteLine($"Полученная последовательность: {String.Join(" ",
MasDigit)}");
                Console.WriteLine(Parity(MasDigit) ?
                    "Элементы, стоящие на четных местах, образуют возрастающую
последовательность" :
                    "Элементы, стоящие на четных местах, не образуют возрастающую
последовательность");

            }
            else
            {
                Console.WriteLine($"Каждый член последовательности равен 0");
            }
            Console.ReadKey();
        }
    }
}
```

Приложение G. К задаче 7

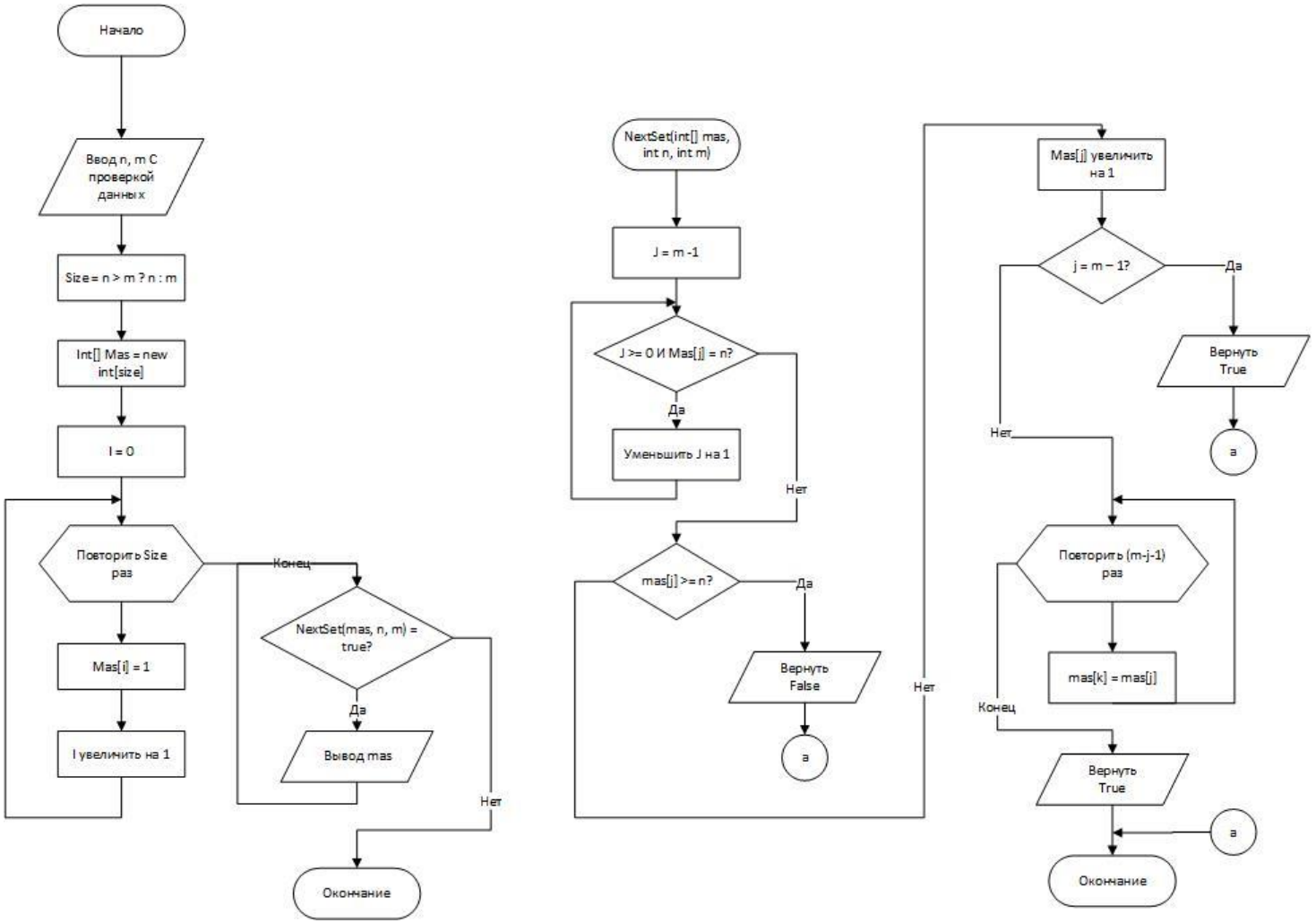


Рисунок 10.6. Блок-схема алгоритма кодирования алфавита

Таблица 10.11. Черный ящик к задаче 7

№	Проверяемая ситуация	Тесты						
		T1	**T2**	**T3**	**T4**	**T5**	**T6**	**T7**
1	Классы входных данных							
1.1	Длина массива - натуральное число			+	+	+	+	+
1.2	Длина массива - не натуральное число		+					
1.3	Длина массива - не число	+						
1.4	Элемент массива - натуральное число					+	+	+
1.5	Элемент массива - не натуральное число				+			
1.6	Элемент массива - не число			+				
2	Классы выходных данных							
2.1	Ошибка ввода	+	+	+	+	+		
2.2	Кодирующий алфавит в лексикографическом порядке						+	+
3	Критерии длины массива							
3.1	Количество слов длины N в массиве больше 2^N					+		
3.2	Количество слов длины N в массиве меньше 2^N							+
3.3	Количество слов длины N в массиве равно 2^N						+	

Листинг программы:

```
using System;

namespace Task7_v17
{
    class CombinationsWithRepetitions
    {
        static int num = 1;
        static bool NextSet(int[] mas, int n, int m)
        {
            int j = m - 1;
            while (j >= 0 && mas[j] == n) j--;

            if (j < 0) return false;

            if (mas[j] >= n)
            {
                j--;
            }

            mas[j]++;
            if (j == m - 1) return true;

            for(int k = j + 1; k < m; k++)
            {
                mas[k] = mas[j];
            }

            return true;
        }

        static void Print(int[] mas, int n)
        {
            Console.Write($"{num++}: ");
            for(int i = 0; i < n; i++)
            {
                Console.Write(mas[i] + " ");
```

```
            }
            Console.WriteLine();
        }

        static int numberInput()
        {
            int number;
            while(!int.TryParse(Console.ReadLine(), out number) || number <= 0)
            {
                Console.WriteLine("Ошибка. Вы не верно ввели число. Повторите ввод");
            }

            return number;
        }

        static void Main(string[] args)
        {
            int n, m;
            Console.WriteLine("Введите N(колиечсто чисел множества):");
            n = numberInput();

            Console.WriteLine("Введите M(Количество чесел в наборе):");
            m = numberInput();

            int size = n > m ? n : m; // размер массива а выбирается как max(n,m)

            int[] mas = new int[size];

            for (int i = 0; i < size; i++)
                mas[i] = 1;

            //Print(mas, m);

            while (NextSet(mas, n, m))
                Print(mas, m);
        }
    }
}
```

Приложение Н. К задаче 8

Таблица 10.12. Черный ящик к задаче 8

№	Проверяемая ситуация	Тесты		
		T1	T2	T3
1	Классы входных данных			
1.1	В матрице есть не числа	+		
1.2	Граф связный		+	
1.3	Граф не связный			+
1.4	Граф имеет <2 нечетный вершины		+	
1.5	Граф имеет >= 2 нечетный вершины			+
2	Классы выходных данных			
2.1	Ошибка ввода	+		
2.2	Эйлеров цикл		+	
2.2	Граф не содержит эйлеров цикл			+

Рисунок 10.7. Блок-схема алгоритма поиска эйлерового пути

Листинг программы:

```
using System;
using System.Collections.Generic;

namespace Tas8_v19
{
    public class EulerCycle
    {
        // Матрица смежности
        static int[,] matrix = null;

        // Функция для проверки ввода размера матрицы смежности
        static int SizeInput()
        {
            int number;
            Console.WriteLine("Введите размер матрицы смежности");

            while(!(int.TryParse(Console.ReadLine(), out number)) || number < 0)
            {
                Console.WriteLine("Ошибка. Вы ввели неверный размер матрицы, повторите
ввод.");
            }

            return number;
        }

        // Функция для корректного ввода элементов матрицы смежности
        static int ElemInput(int i, int j)
        {
            Console.WriteLine($"Введите {j} элемент {i} строки матрицы смежности");
            int number;
            while (!(int.TryParse(Console.ReadLine(), out number)) || number < 0 ||
number > 1)
            {
                Console.WriteLine("Ошибка. Вы ввели элемент матрицы, повторите ввод.");
```

```
    }

    return number;

}

// Функция для заполения матрицы смежности
static int[,] MatrixInput(int size)
{
    int[,] _matrix = new int[size, size];
    for(int i = 0; i < size; i++)
    {
        for(int j = 0; j < size; j++)
        {
            _matrix[i, j] = ElemInput(i + 1, j + 1);
        }
    }

    return _matrix;
}

static void Main(string[] args)
{
    int size = SizeInput();
    matrix = MatrixInput(size);

    List<int> deg = new List<int>();
    for(int i = 0; i < size; i++)
    {
        deg.Add(0);
    }

    for(int i = 0; i < size; i++)
    {
        for(int j = 0; j < size; j++)
        {
            deg[i] += matrix[i, j];
        }
    }

    int first = 0;
    while (deg[first] == 0) ++first;

    int v1 = -1; // Первая вершина нечетной степени
    int v2 = -1; // Вторая вершина нечетной степени

    bool linked = false; // Несвязный ли граф

    for (int i = 0; i < size; ++i)
    {
        if ((deg[i] & 1) != 0)
        {
            if (v1 == -1)
            {
                v1 = i;
            }
            else
            {
                if (v2 == -1)
                    v2 = i;
```

```
                else
                    linked = true;
            }
        }
    }

    if(v1 != -1)
    {
        ++matrix[v1, v2];
        ++matrix[v2, v1];
    }

    Stack<int> stack = new Stack<int>();
    stack.Push(first);

    List<int> res = new List<int>();

    while(stack.Count != 0)
    {
        int v = stack.Peek();
        int pos;

        for(pos = 0; pos < size; pos++)
        {
            if (matrix[v, pos] != 0)
            {
                break;
            }
        }

        if(pos == size)
        {
            res.Add(v);
            stack.Pop();
        }
        // Удаляем фиктивное ребро
        else
        {
            --matrix[v, pos];
            --matrix[pos, v];
            stack.Push(pos);
        }
    }

    if(v1 != -1)
    {
        for (int i=0; i+1 < res.Count; i++)
        {
            if(res[i] == v1 && res[i+1] == v2 || res[i] == v2 && res[i + 1] ==
v1)
            {
                List<int> res2 = new List<int>();
                for(int j = i + 1; j < res.Count; j++)
                {
                    res2.Add(res[j]);
                }

                for (int j = 1; j <= i; j++)
                {
                    res2.Add(res[j]);
```

```
                    }

                    res = res2;
                    break;
                }
            }
        }

        for(int i = 0; i < size; i++)
        {
            for(int j = 0; j < size; j++)
            {
                if(matrix[i,j] != 0)
                {
                    linked = true;
                }
            }
        }

        // Если граф несвязный
        if (linked)
        {
            Console.WriteLine("Данный граф не содержит эйлеров цикл");
        }
        else
        {
            Console.WriteLine("Данный граф содержит эйлеров цикл");
            for(int i = 0; i < res.Count; i++)
            {
                Console.Write($"{res[i] + 1} ");
            }
        }
    }
}
}
```

Приложение I. К задаче 9

Таблица 10.13. Проект свойств для класса Node

№	Свойство	Тип свойства	Описание
1	Data	int, автоматическое	Значение элемента
2	Next	Node, автоматическое	Ссылка на следующий элемент списка

Таблица 10.14. Проект методов для класса LinkedList

№	Метод	Тип возвращаемого объекта	Аргументы	Описание
1	LinkedList	инициализатор	—	Инициализатор без аргументов
2	Add	void	Int data	Добавляет элемент в список
	Remove	bool	Int data	Удаляет элемент из списка
3	PrintList	void	—	Вывод списка в консоль
4	MakeList	LinkedList	Int size	Создает список с заданным размером
5	MakePositiveAndNegativeList	void	Ref LinkedList original Ref LinkedList positive Ref LinkedList onegative	Создает списки положительных и отрицательных чисел из основного списка. Удаляет добавленный элементы в списки из основного

Таблица 10.15. Проект свойств для класса LinkedList

№	Свойство	Тип свойства	Описание
1	Head	Node, автоматическое	Первый элемент списка
2	Tail	Node, автоматическое	Последний элемент списка
3	Count	int, только для чтения	Количество элементов в списке

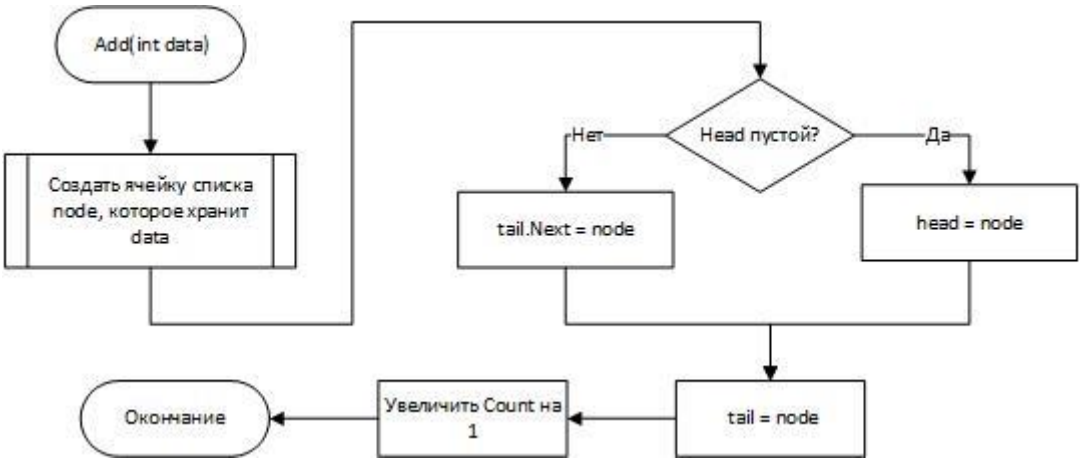


Рисунок 10.1. Блок-схема для метода Add

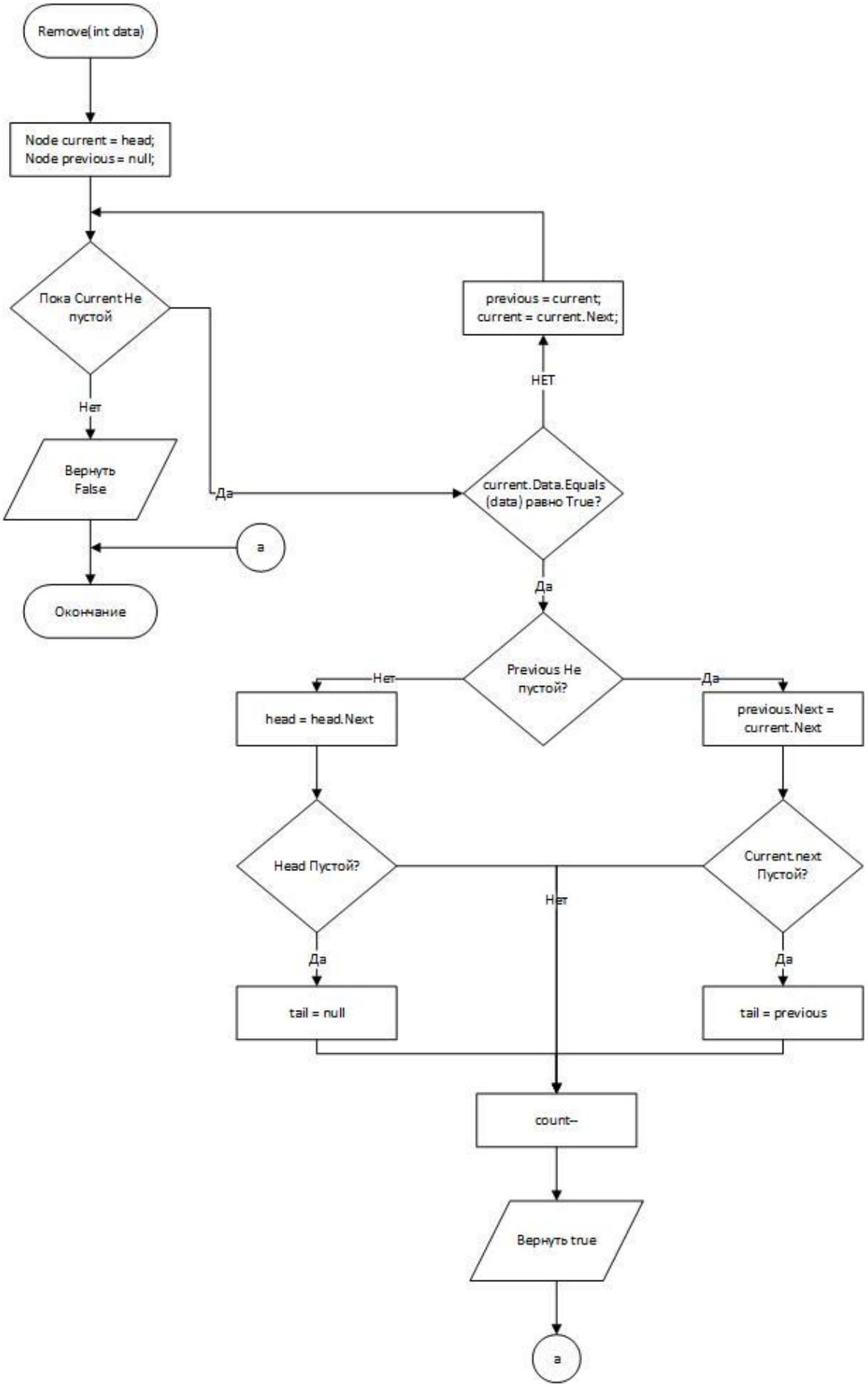


Рисунок 10.2. Блок-схема для метода Remove

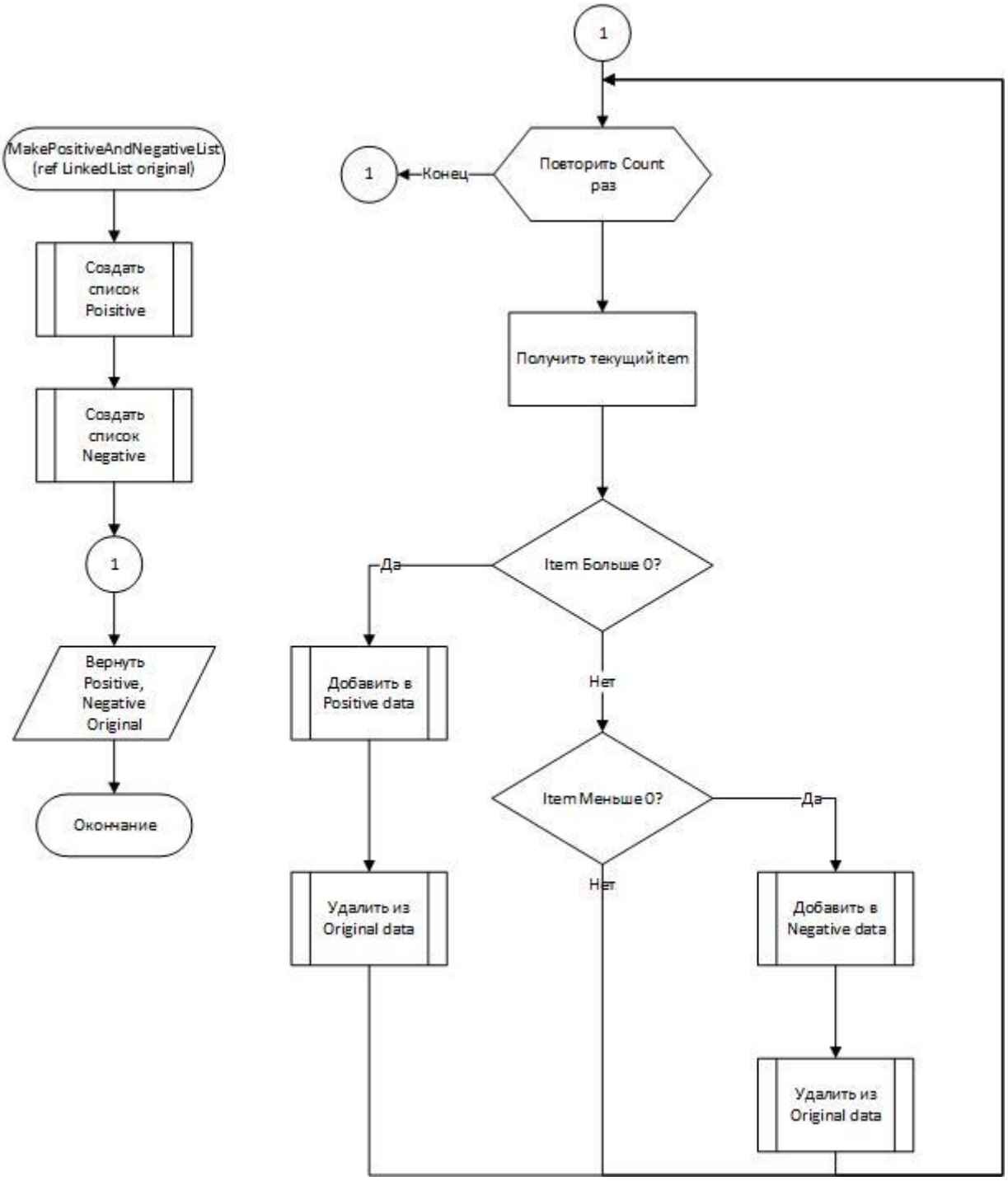


Рисунок 10.3. Блок-схема для метода MakePositiveAndNegativeList

Таблица 10.16. Черный ящик к задаче 9

№	Проверяемая ситуация	Тесты		
		Т1	**Т2**	**Т3**
1	Классы входных данных			
1.1	N - натуральное число			+
1.2	N - не натуральное число		+	
1.3	N - не число	+		
2	Классы выходных данных			
2.1	Ошибка ввода	+	+	
2.2	Списки			+

№	Проверяемая ситуация	Тесты		
		T1	T2	T3
3	Функции и свойства			
3.1	Инициализация LinkedList			+
3.2	Инициализация Node			+
3.3	LinkedList.Print()			+
3.4	LinkedList Add()			+
3.5	LinkedListRemove()			+
3.9	LinkedList MakePositiveAndNefativeList()			+
3.10	LinkedList MakeList()			+
3.11	LinkedList Head			+
3.12	LinkedList Tail			+
3.13	Node Data			+
3.14	Node Next			+

Листинг Node и LinkedList:

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Text;

namespace task9_v37
{
    public class Node
    {
        public Node(int data)
        {
            Data = data;
        }

        public int Data { get; set; }

        public Node Next { get; set; }
    }

    public class LinkedList : IEnumerable  // односвязный список
    {
        private static Random random = new Random();

        Node head;  // головной/первый элемент
        Node tail;  // последний/хвостовой элемент
        int count;  // количество элементов в списке

        public int Count { get { return count; } }

        public bool IsEmpty { get { return count == 0; } }

        // добавление элемента
       public void Add(int data)
        {
            Node node = new Node(data);

            if (head == null)
                head = node;
            else
                tail.Next = node;
```

```
            tail = node;

            count++;
        }

        // удаление элемента
        public bool Remove(int data)
        {
            Node current = head;
            Node previous = null;

            while (current != null)
            {
                if (current.Data.Equals(data))
                {
                    // Если узел в середине или в конце
                    if (previous != null)
                    {
                        // убираем узел current, теперь previous ссылается не на current,
а на current.Next
                        previous.Next = current.Next;

                        // Если current.Next не установлен, значит узел последний,
                        // изменяем переменную tail
                        if (current.Next == null)
                            tail = previous;
                    }
                    else
                    {
                        // если удаляется первый элемент
                        // переустанавливаем значение head
                        head = head.Next;

                        // если после удаления список пуст, сбрасываем tail
                        if (head == null)
                            tail = null;
                    }
                    count--;
                    return true;
                }

                previous = current;
                current = current.Next;
            }
            return false;
        }

        // очистка списка
        public void Clear()
        {
            head = null;
            tail = null;
            count = 0;
        }

        // содержит ли список элемент
        public bool Contains(int data)
        {
            Node current = head;
            while (current != null)
```

```
            {
                if (current.Data.Equals(data))
                    return true;
                current = current.Next;
            }
            return false;
        }

        // добвление в начало
        public void AppendFirst(int data)
        {
            Node node = new Node(data);
            node.Next = head;
            head = node;
            if (count == 0)
                tail = head;
            count++;
        }

        // Печать связного списка
        public void PrintList()
        {
            foreach (var item in this)
            {
                Console.Write(item + " ");
            }
            Console.WriteLine();
        }

        // реализация интерфейса IEnumerable
        IEnumerator IEnumerable.GetEnumerator()
        {
            Node current = head;
            while (current != null)
            {
                yield return current.Data;
                current = current.Next;
            }
        }

        // Генерация связного списка с помощью ДСЧ
        public static LinkedList MakeList(int size)
        {
            LinkedList list = new LinkedList();
            for (int i = 0; i < size; i++)
            {
                list.Add(random.Next(-100, 100));
            }

            return list;
        }

        public static void MakePositiveAndNegativeList(ref LinkedList original, out
LinkedList positive, out LinkedList negative)
        {
            positive = new LinkedList();
            negative = new LinkedList();

            foreach (var item in original)
            {
```

```
                if ((int)item > 0)
                {
                    positive.Add((int)item);
                    original.Remove((int)item);
                }
                if ((int)item < 0)
                {
                    negative.Add((int)item);
                    original.Remove((int)item);
                }
            }
        }
    }
}
```

Листинг программы:

```
using System;

namespace task9_v37
{
    class GenerationLinkedLists
    {
        static int SizeInput()
        {
            int number;
            Console.WriteLine("Введите количество элементов связного списка:");
            while(!(int.TryParse(Console.ReadLine(), out number)) || number < 0)
            {
                Console.WriteLine("Ошибка. Вы неверно ввели количество элементов списка.
Повторите ввод");
            }
            return number;
        }
        static void Main(string[] args)
        {
            LinkedList original = new LinkedList();
            LinkedList positive;
            LinkedList negative;

            int size = SizeInput();

            original = LinkedList.MakeList(size);
            Console.WriteLine("Список сгенерировался\nСписок:");
            original.PrintList();

            LinkedList.MakePositiveAndNegativeList(ref original, out positive, out
negative);
            Console.WriteLine("Оригальный список после выполнения процедуры, создающей
два новых линейных списка");
            original.PrintList();

            Console.WriteLine("Список с положительными числами:");
            positive.PrintList();

            Console.WriteLine("Список с отрицательными числами:");
            negative.PrintList();

        }
    }
}
```

Приложение J. К задаче 10

Таблица 10.17. Проект свойств для класса Node

№	Свойство	Тип свойства	Описание
1	Data	int, автоматическое	Значение элемента
2	Next	Node, автоматическое	Ссылка на следующий элемент списка

Таблица 10.18. Проект методов для класса LinkedList

№	Метод	Тип возвращаемого объекта	Аргументы	Описание
1	LinkedList	инициализатор	—	Инициализатор без аргументов
2	Add	void	Int data	Добавляет элемент в список
	Remove	bool	Int data	Удаляет элемент из списка
3	PrintList	void	—	Вывод списка в консоль
4	MakeList	LinkedList	Int size	Создает список с заданным размером
5	MakeNewList	void	LinkedList original Int number	Создает новый список

Таблица 10.19. Проект свойств для класса LinkedList

№	Свойство	Тип свойства	Описание
1	Head	Node, автоматическое	Первый элемент списка
2	Tail	Node, автоматическое	Последний элемент списка
3	Count	int, только для чтения	Количество элементов в списке

Таблица 10.20. Черный ящик к задаче 10

№	Проверяемая ситуация	Тесты			
		T1	T2	T3	T4
1	Классы входных данных				
1.1	Строка	+			
1.2	Отрицательные число		+		
1.3	Действительное число			+	
1.4	Целое положительное число				
1.5	Входной файл содержит не только числа				+
2	Классы выходных данных				
2.1	Ошибка				+
2.2	Список	+	+	+	
3	Функции и свойства				
3.1	Инициализация ListEntry				+
3.2	Multiplication				+
4.1	Инициализация LinkedList				+
4.2	Инициализация Node				+
4.3	LinkedList.Print()				+
4.4	LinkedList Add()				+
4.5	LinkedListRemove()				+
4.6	LinkedList MakeNewListt()				+

№	Проверяемая ситуация	Тесты			
		T1	T2	T3	T4
4.7	LinkedList MakeList()				+
4.8	LinkedList Head				+
4.9	LinkedList Tail				+
4.10	Node Data				+
4.11	Node Next				+

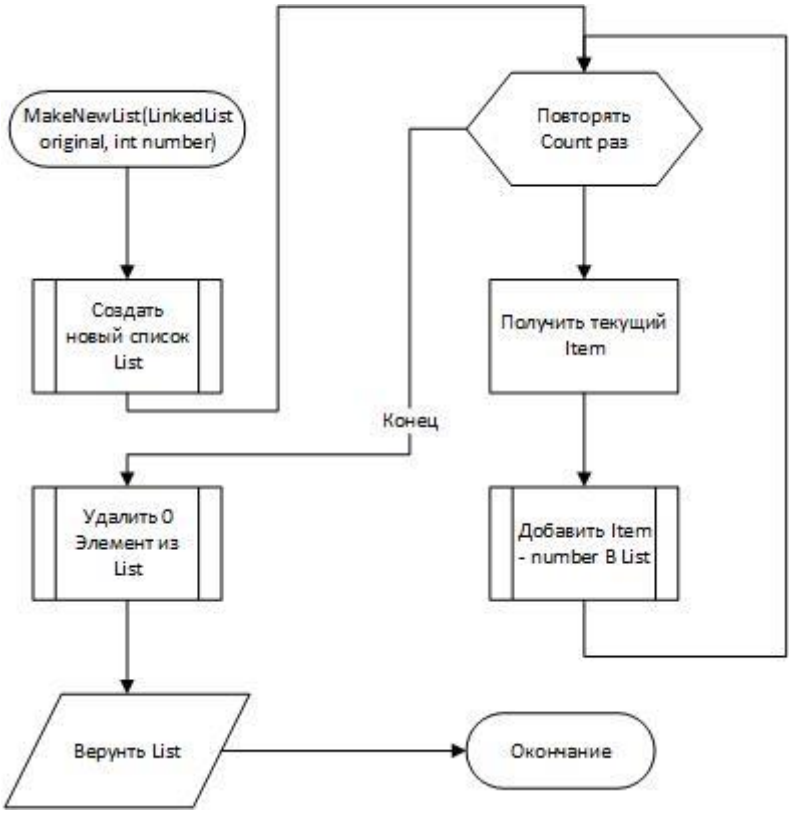


Рисунок 10.8. Блок-схема создания последовательности

Листинг Node и LinkedList:

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Text;

namespace Task10_v531
{
    public class Node
    {
        public Node(int data)
        {
            Data = data;
        }

        public int Data { get; set; }

        public Node Next { get; set; }
    }

    public class LinkedList : IEnumerable  // односвязный список
```

```
    {
        private static Random random = new Random();

        Node head;  // головной/первый элемент
        Node tail;  // последний/хвостовой элемент
        int count;  // количество элементов в списке

        public int Count { get { return count; } }

        public bool IsEmpty { get { return count == 0; } }

        // добавление элемента
        public void Add(int data)
        {
            Node node = new Node(data);

            if (head == null)
                head = node;
            else
                tail.Next = node;
            tail = node;

            count++;
        }

        // удаление элемента
        public bool Remove(int data)
        {
            Node current = head;
            Node previous = null;

            while (current != null)
            {
                if (current.Data.Equals(data))
                {
                    // Если узел в середине или в конце
                    if (previous != null)
                    {
                        // убираем узел current, теперь previous ссылается не на current,
а на current.Next
                        previous.Next = current.Next;

                        // Если current.Next не установлен, значит узел последний,
                        // изменяем переменную tail
                        if (current.Next == null)
                            tail = previous;
                    }
                    else
                    {
                        // если удаляется первый элемент
                        // переустанавливаем значение head
                        head = head.Next;

                        // если после удаления список пуст, сбрасываем tail
                        if (head == null)
                            tail = null;
                    }
                    count--;
                    return true;
                }
```

```
            previous = current;
            current = current.Next;
        }
        return false;
    }

    // очистка списка
    public void Clear()
    {
        head = null;
        tail = null;
        count = 0;
    }

    // содержит ли список элемент
    public bool Contains(int data)
    {
        Node current = head;
        while (current != null)
        {
            if (current.Data.Equals(data))
                return true;
            current = current.Next;
        }
        return false;
    }

    // добвление в начало
    public void AppendFirst(int data)
    {
        Node node = new Node(data);
        node.Next = head;
        head = node;
        if (count == 0)
            tail = head;
        count++;
    }

    // Печать связного списка
    public void PrintList()
    {
        foreach (var item in this)
        {
            Console.Write(item + " ");
        }
        Console.WriteLine();
    }

    // реализация интерфейса IEnumerable
    IEnumerator IEnumerable.GetEnumerator()
    {
        Node current = head;
        while (current != null)
        {
            yield return current.Data;
            current = current.Next;
        }
    }
```

```
        // Генерация связного списка с помощью ДСЧ
        public static LinkedList MakeList(int size)
        {
            LinkedList list = new LinkedList();
            for (int i = 1; i <= size; i++)
            {
                list.Add(i);
            }

            return list;
        }

        public static LinkedList MakeNewList(LinkedList original, int number)
        {
            LinkedList list = new LinkedList();
            foreach(var item in original)
            {
                list.Add((int)item - number);
            }

            list.Remove(0);
            return list;
        }
    }
}
```

Листинг программы

```
using System;

namespace Task10_v531
{
    class GenerationSequence
    {
        static int SizeInput()
        {
            int number;
            Console.WriteLine("Введите количество элементов связного списка:");
            while (!(int.TryParse(Console.ReadLine(), out number)) || number < 2)
            {
                Console.WriteLine("Ошибка. Вы неверно ввели количество элементов списка.
Повторите ввод");
            }
            return number;
        }

        static void Main(string[] args)
        {
            int number = SizeInput();

            LinkedList original = LinkedList.MakeList(number);
            LinkedList linked = LinkedList.MakeNewList(original, number);

            Console.WriteLine("Список сгенерировался\nСписок:");
            original.PrintList();

            Console.WriteLine("Последовательность");
            linked.PrintList();
        }
    }
```

Приложение К. К задаче 11

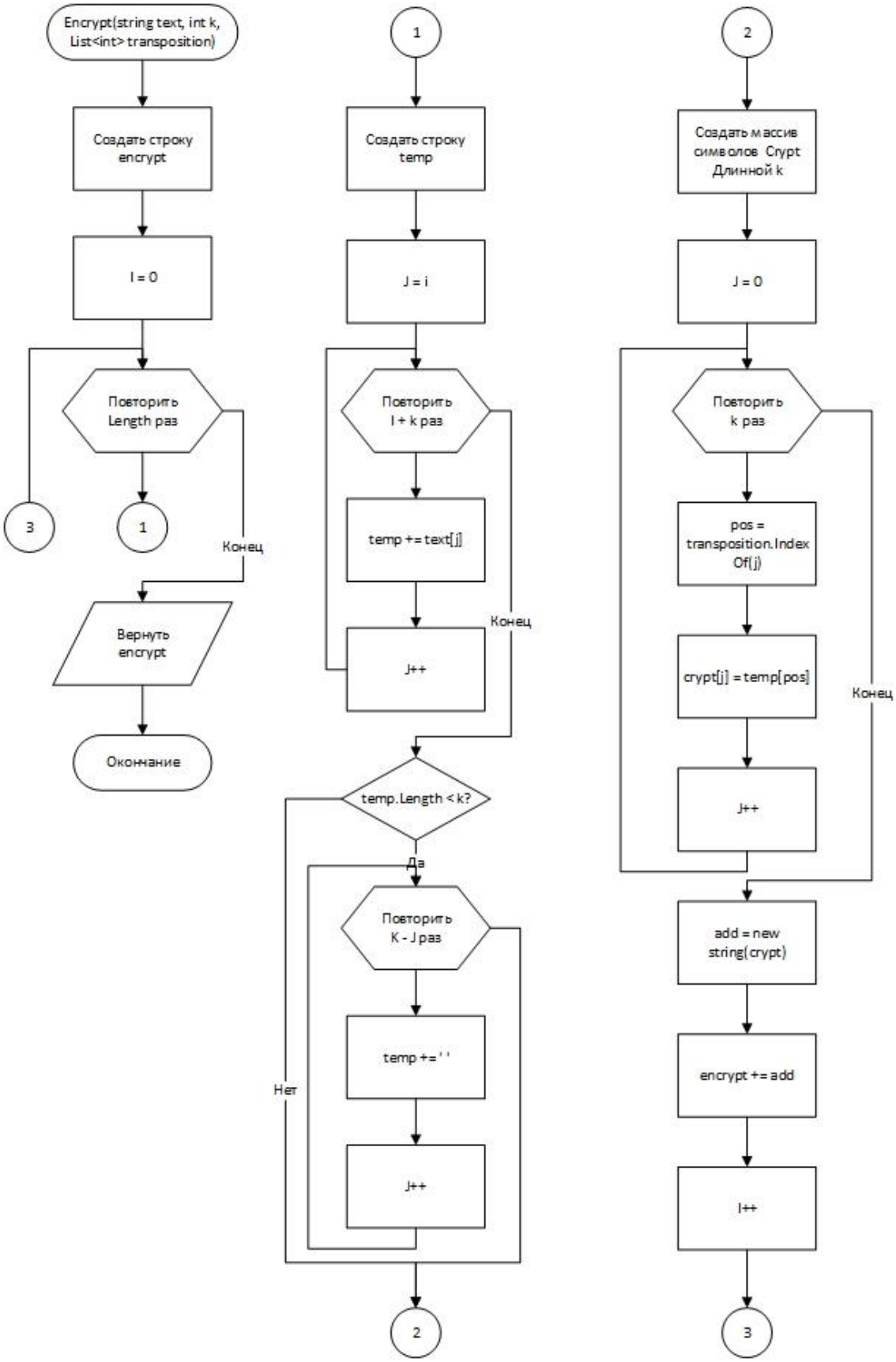


Рисунок 10.9. Блок-схема алгоритма кодирования для задачи 11

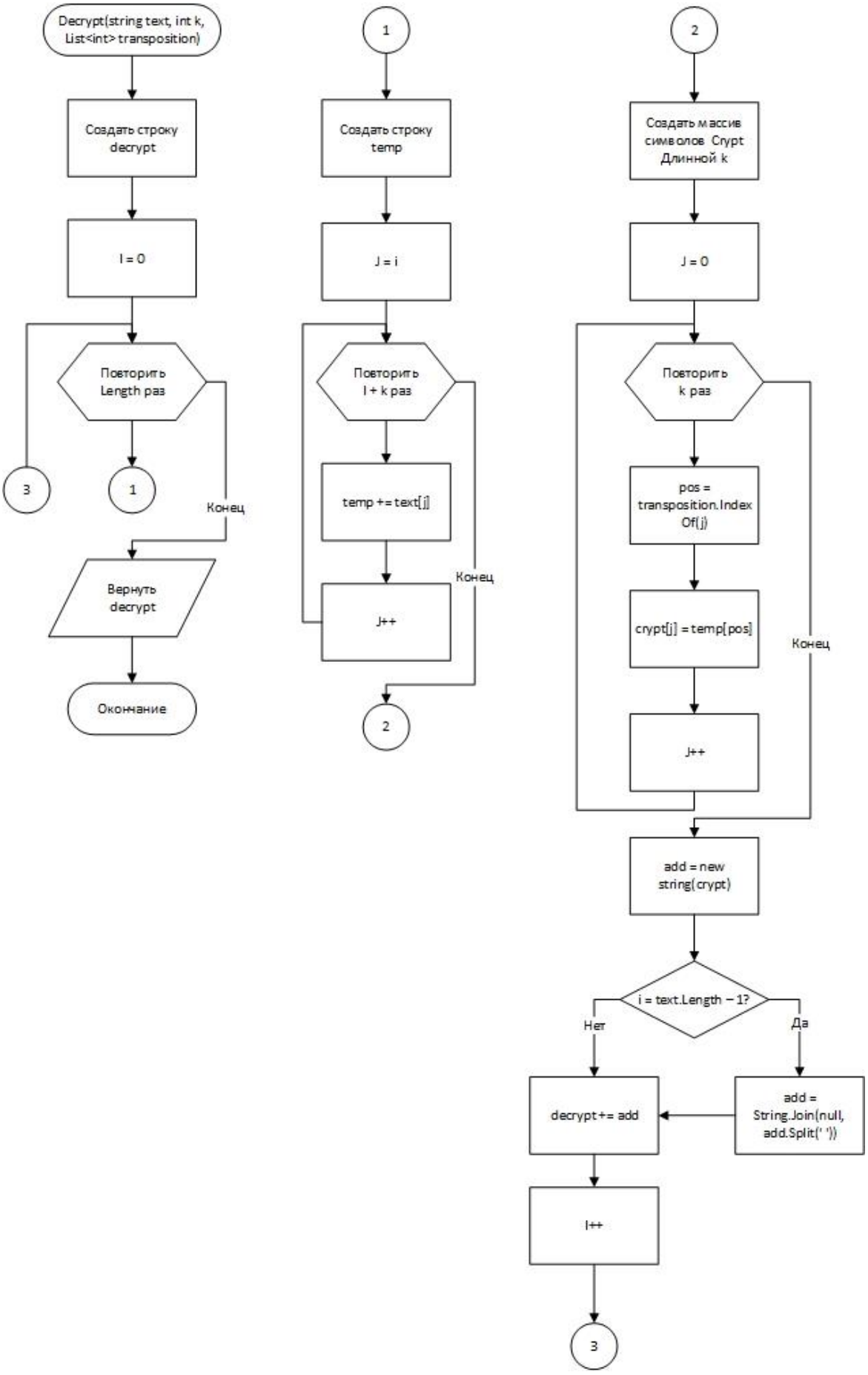


Рисунок 10.2.. Блок-схема алгоритма декодирования для задачи 11

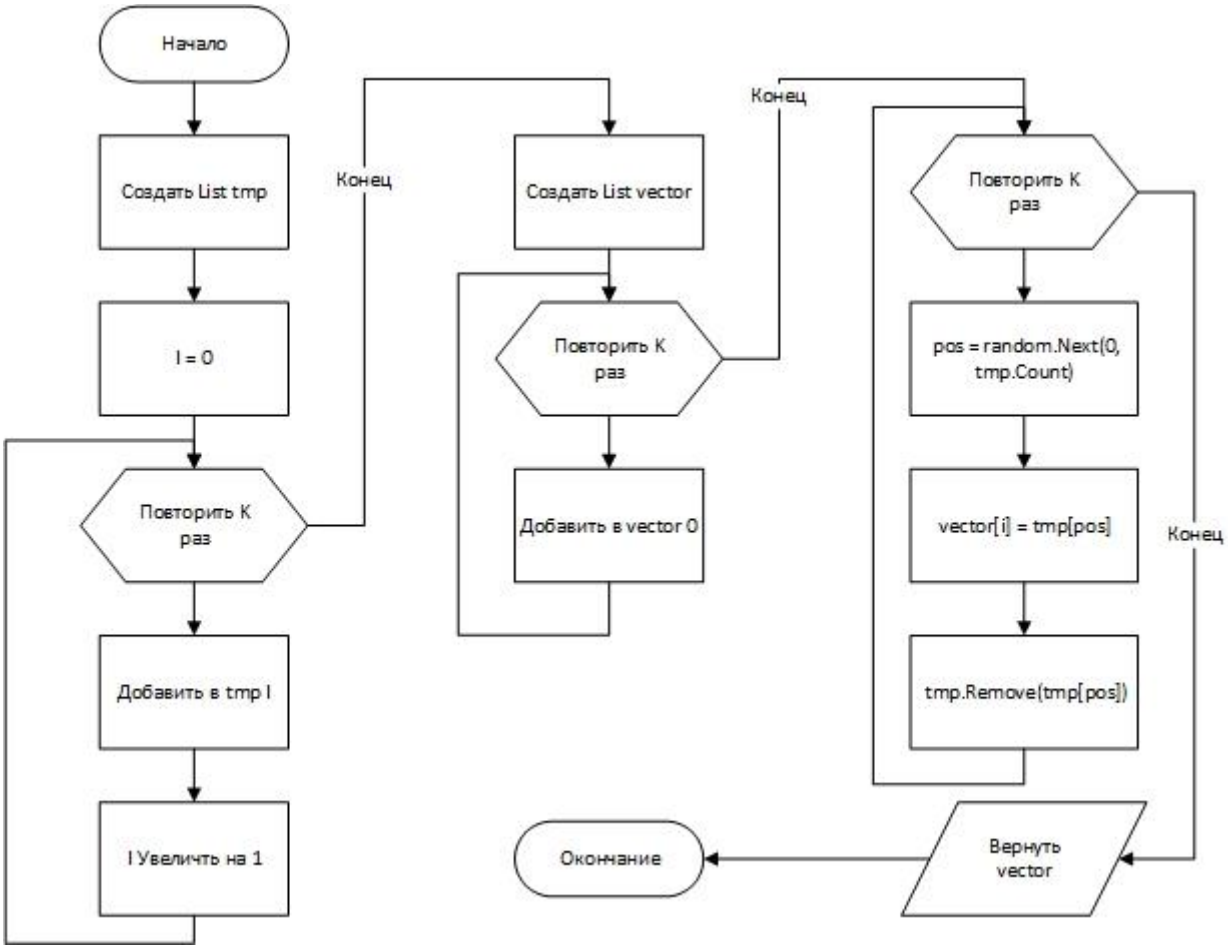


Рисунок 10.3.. Блок-схема алгоритма для создания последовательности перестановок для задачи 11

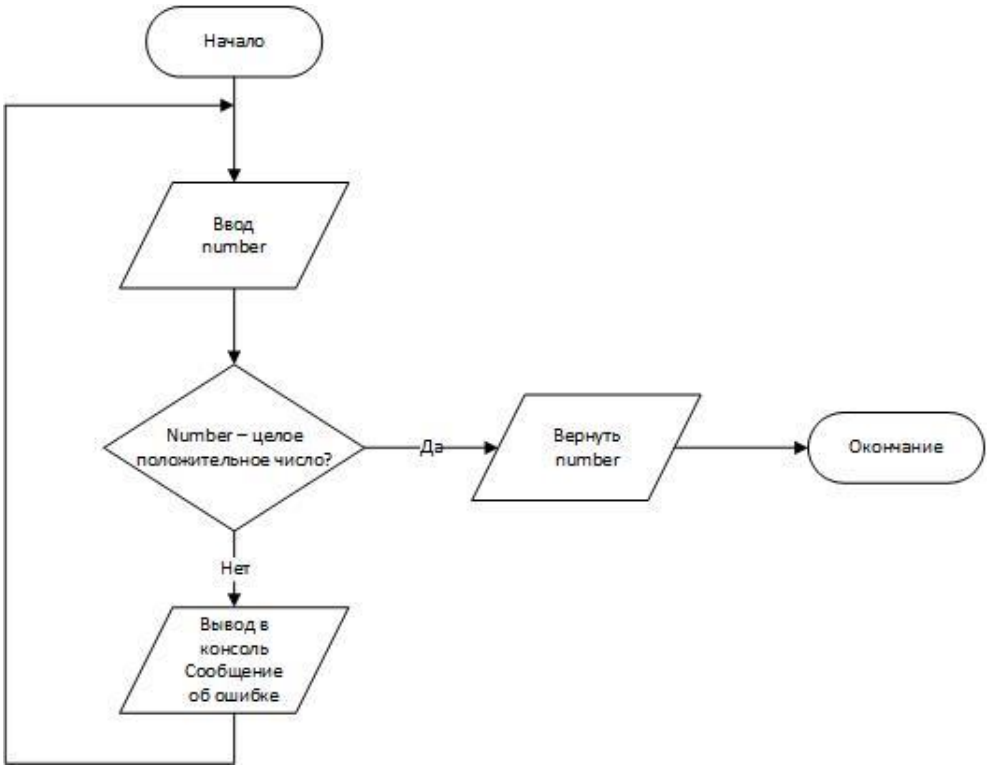


Рисунок 10.4. Блок-схема алгоритма проверки данных для задачи 11

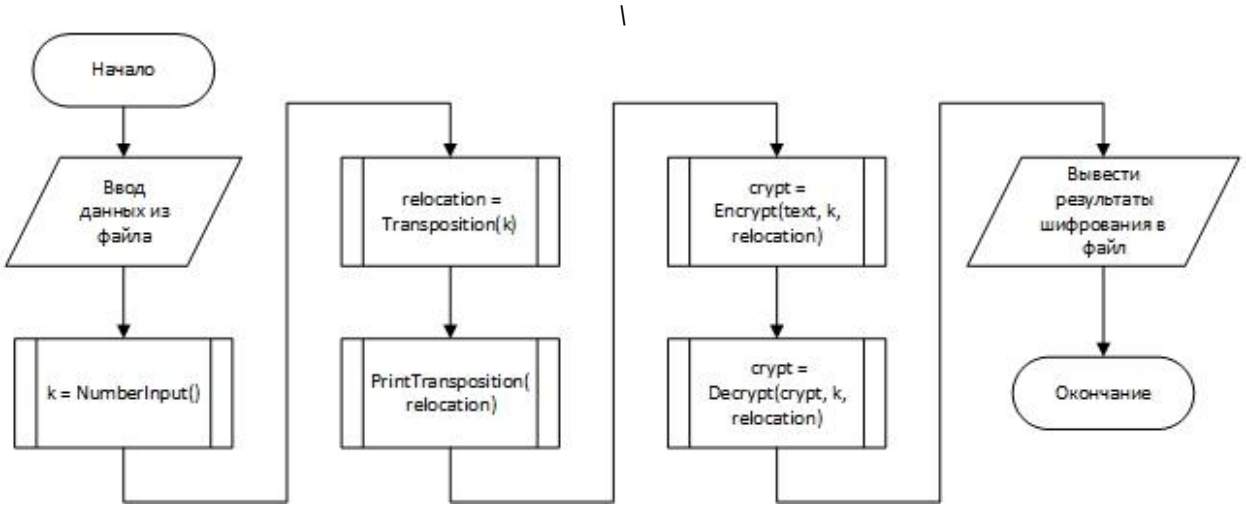


Рисунок 10.5. Блок-схема алгоритма для задачи 11

Таблица 10.21. Черный ящик к задаче 11

№	Проверяемая ситуация	Тесты		
		T1	T2	T3
1	Классы входных данных			
1.1	Файл пустой		+	
1.2	Файл не пустой	+		+
1.3	Целое положительное число			+
1.4	Не целое положительное число	+		
2	Классы выходных данных			
2.1	Ошибка ввода	+		
2.2	Ошибка данных		+	
	Шифрованный / Дешифрованный текст			+

Листинг программы:

```
using System;
using System.Collections.Generic;
using System.IO;

namespace Task11_v839
{
    class Crypter
    {
        static Random random = new Random();

        static StreamReader reader = new StreamReader("input.txt");  // Файл для чтения
данных
        static StreamWriter writer = new StreamWriter("output.txt"); // Файл для записи
данных

        // Ввод длины последовательности перестановки
        static int NumberInput()
        {
            int number;
            while(!int.TryParse(Console.ReadLine(), out number) || number < 0)
            {
```

```
                Console.WriteLine("Ошибка. Вы ввели неверно длину перестановки. Повторите
ввод");
            }
            return number;
        }

        // Создание последовательности перестановок
        static List<int> Transposition(int k)
        {
            List<int> tmp = new List<int>();
            for(int i = 0; i < k; i++)
            {
                tmp.Add(i);
            }

            List<int> vector = new List<int>();
            for (int i = 0; i < k; i++)
            {
                vector.Add(0);
            }

            for (int i = 0; i < k; i++)
            {
                int pos = random.Next(0, tmp.Count);
                vector[i] = tmp[pos];

                tmp.Remove(tmp[pos]);
            }

            return vector;
        }

        static void PrintTransposition(List<int> transposition)
        {
            Console.WriteLine("Шифр");
            int pos = 1;
            for(int i = 0; i < transposition.Count; i++)
            {
                Console.Write($"{pos} -> {transposition[i]+1}; ");
                pos++;
            }
            Console.WriteLine();
        }

        // Зашифровка текста
        static string Encrypt(string text, int k, List<int> transposition)
        {
            string encrypt = "";

            for(int i = 0; i < text.Length; i += k)
            {
                string temp = "";
                for (int j = i; (j < i + k) && j < text.Length; j++)
                {
                    temp += text[j];
                }

                if (temp.Length < k)
                {
                    for(int j = temp.Length -1; j < k; j++)
```

```
            {
                temp += ' ';
            }
        }

        char[] crypt = new char[k];

        for (int j = 0; j < k; j++)
        {
            int pos = transposition.IndexOf(j);
            crypt[j] = temp[pos];
        }
        string add = new string(crypt);
        encrypt += add;
    }

    return encrypt;
}

// Расшифровка текста
static string Decrypt(string text, int k, List<int> transposition)
{
    string decrypt = "";

    for (int i = 0; i < text.Length; i += k)
    {
        string temp = "";
        for (int j = i; j < i + k; j++)
        {
            temp += text[j];
        }

        char[] crypt = new char[k];

        for (int j = 0; j < k; j++)
        {
            crypt[j] = temp[transposition.IndexOf(j)];
        }

        string add = new string(crypt);


        if (i == text.Length - 1) add = String.Join(null, add.Split(' '));

        decrypt += add;
    }

    return decrypt;
}

static void Main(string[] args)
{
    // Длина последовательности перестановки
    Console.WriteLine("Введите длину последовательности перестановки");
    int k = NumberInput();

    // Последовательность перестановок
    List<int> relocation = Transposition(k);

    // Текст который нужно засшифровать
```

```
            string text = reader.ReadToEnd();

            PrintTransposition(relocation);

            // Зашифрованный текст
            string crypt = Encrypt(text, k, relocation);

            Console.WriteLine("Текст был зашифрован и записан в файл");
            writer.WriteLine("Зашифрованный текст: ");
            writer.WriteLine(crypt);

            // Расшифрованный текст
            crypt = Decrypt(crypt, k, relocation);

            Console.WriteLine("Текст был расшифрован и записан в файл");
            writer.WriteLine("Расшифрованный текст: ");
            writer.WriteLine(crypt);

            writer.Close();
        }
    }
}
```

Приложение L. К задаче 12

Таблица 10.22. Черный ящик к задаче 12

№	Проверяемая ситуация	Тесты					
		T1	**T2**	**T3**	**T4**	**T5**	**T6**
1	Классы входных данных						
1.1	В массиве только числа		+	+	+	+	+
1.2	В массиве есть не числа	+					
2	Классы выходных данных						
2.1	Ошибка ввода	+					
2.2	Массив пуст					+	
2.3	Массив, отсортированный обоими способами		+	+	+		+
3	Критерий длины						
3.1	Массив нулевой длинны					+	
3.2	Массив с одним элементов						+
3.3	Массив длиннее одного элемента		+	+	+		
4	Критерий сортировки						
4.1	Упорядочен по возрастанию		+				
4.2	Упорядочен по убыванию			+			
4.3	Неупорядочен				+		
5	Функции						
5.1	Сортировка деревом		+	+	+	+	+
5.2	Сортировка слиянием		+	+	+	+	+
5	Критерий поиска						
5.1	Элемент не имеет повторений		+	+			+
5.2	Какой-то элемент повторяется				+		

Листинг TreeNode:

```
using System;
using System.Collections.Generic;
using System.Text;

namespace Task12_v4_5
{
    // Простая реализация бинарного дерева
    public class TreeNode
    {
        public TreeNode(int data)
        {
            Data = data;
        }

        // Данные
        public int Data { get; set; }

        // Левая ветка дерева
        public TreeNode Left { get; set; }

        // Правая ветка дерева
        public TreeNode Right { get; set; }
```

```
        // Рекурсивное добавление узла в дерево
        public void Insert(TreeNode node, ref int count_compare, ref int count_changed)
        {
            count_compare++;
            if (node.Data < Data)
            {
                if (Left == null)
                {
                    Left = node;
                    count_changed++;
                }
                else
                {
                    Left.Insert(node, ref count_compare, ref count_changed);
                }
            }
            else
            {
                if (Right == null)
                {
                    Right = node;
                    count_changed++;
                }
                else
                {
                    Right.Insert(node, ref count_compare, ref count_changed);
                }
            }
        }

        // Преобразование дерева в отсортированный массив
        public int[] Transform(List<int> elements, ref int count_compare, ref int
count_changed)
        {
            if (elements == null)
            {
                elements = new List<int>();
            }

            if (Left != null)
            {
                count_changed++;
                Left.Transform(elements, ref count_compare, ref count_changed);
            }

            elements.Add(Data);

            if (Right != null)
            {
                count_changed++;
                Right.Transform(elements, ref count_compare, ref count_changed);

            }
            return elements.ToArray();

        }

        // Метод для сортировки с помощью двоичного дерева
```

```
        public static int[] TreeSort(int[] array, ref int count_compare, ref int
count_changed)
        {
            var treeNode = new TreeNode(array[0]);
            for (int i = 1; i < array.Length; i++)
            {
                treeNode.Insert(new TreeNode(array[i]), ref count_compare, ref
count_changed);
            }

            return treeNode.Transform(new List<int>(), ref count_compare, ref
count_changed);
        }
    }
}
```

Листинг программы:

```
using System;

namespace Task12_v4_5
{
    class Program
    {
        public static int count_compare = 0;
        public static int count_changed = 0;

        // Копирование массива
        public static int[] IntCopy(int[] sourse, int sourse_index, int length)
        {
            int[] ans = new int[length];
            Array.Copy(sourse, sourse_index, ans, 0, length);
            return ans;
        }

        public static string ArrToString(int[] arr)
        {
            string str = "";
            for (int i = 0; i < arr.Length; i++)
            {
                str += arr[i] + " ";
            }
            return str.Trim();
        }

        // Сортировка слиянием
        public static int[] MergeSort(int[] arr)
        {
            int[] ans = IntCopy(arr, 0, arr.Length);
            if (ans.Length > 1)
            {
                int index = ans.Length / 2;
                int[] arr1 = MergeSort(IntCopy(ans, 0, index));
                int[] arr2 = MergeSort(IntCopy(ans, index, ans.Length - index));

                int count1 = 0, count2 = 0, i = 0;
                for (i = 0; count1 != arr1.Length && count2 != arr2.Length; i++)
                {
                    if (arr1[count1] > arr2[count2]) ans[i] = arr2[count2++];
                    else ans[i] = arr1[count1++];
```

```
                    count_compare++;
                    count_changed++;
                }
                while (count1 != arr1.Length) { ans[i++] = arr1[count1++];
count_compare++; count_changed++; }
                while (count2 != arr2.Length) { ans[i++] = arr2[count2++];
count_compare++; count_changed++; }
            }
            return ans;
        }

        // Старт сортировки слиянием
        public static int[] StartMergeSort(int[] arr)
        {
            count_changed = 0;
            count_compare = 0;
            return MergeSort(arr);
        }

        static void Main(string[] args)
        {
            int[] arr1 = new int[] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
            int[] arr2 = new int[] { 10, 9, 8, 7, 6, 5, 4, 3, 2, 1 };
            int[] arr3 = new int[] { 5, 8, 1, 3, 6, 2, 9, 4, 10, 7 };

            Console.WriteLine($"Упорядоченный по возрастанию массив:
{ArrToString(arr1)}");
            Console.WriteLine($"Упорядоченный по убыванию массив: {ArrToString(arr2)}");
            Console.WriteLine($"Неупорядоченный массив: {ArrToString(arr3)}");

            Console.WriteLine($"\nУпорядоченный по возрастанию массив. Сортировка
пузырьком.\n{ArrToString(TreeNode.TreeSort(arr1,ref count_compare, ref
count_changed))}\nКоличество пересылок: {count_changed}. Количество сравнений:
{count_compare}");

            Console.WriteLine($"\nУпорядоченный по убыванию массив. Сортировка
пузырьком.\n{ArrToString(TreeNode.TreeSort(arr1, ref count_compare, ref
count_changed))}\nКоличество пересылок: {count_changed}. Количество сравнений:
{count_compare}");

            Console.WriteLine($"\nНеупорядоченный массив. Сортировка
пузырьком.\n{ArrToString(TreeNode.TreeSort(arr1, ref count_compare, ref
count_changed))}\nКоличество пересылок: {count_changed}. Количество сравнений:
{count_compare}");

            Console.WriteLine($"\nУпорядоченный по возрастанию массив. Сортировка
слияниями.\n{ArrToString(StartMergeSort(arr1))}\nКоличество пересылок: {count_changed}.
Количество сравнений: {count_compare}");

            Console.WriteLine($"\nУпорядоченный по убыванию массив. Сортировка
слияниями.\n{ArrToString(StartMergeSort(arr2))}\nКоличество пересылок: {count_changed}.
Количество сравнений: {count_compare}");

            Console.WriteLine($"\nНеупорядоченный массив. Сортировка
слияниями.\n{ArrToString(StartMergeSort(arr3))}\nКоличество пересылок: {count_changed}.
Количество сравнений: {count_compare}");

            Console.ReadKey();
        }
```

```
    }
}
```