|  |  |
| --- | --- |
| 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-16-1  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (Фамилия, имя, отчество)  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) |
| Проверил:  Руководитель практики  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (должность, ученая степень, ученое звание)  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (Фамилия, имя, отчество)  \_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_  (оценка) (подпись)  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (дата) |  |

**Оглавление**

[Глава 1. Введение 4](#_Toc492373810)

[Глава 2. Задание 1 5](#_Toc492373811)

[Глава 3. Задание 2 9](#_Toc492373812)

[Глава 4. Задание 3 12](#_Toc492373813)

[Глава 5. Задание 4 17](#_Toc492373814)

[Глава 6. Задание 5 21](#_Toc492373815)

[Глава 7. Задание 6 24](#_Toc492373816)

[Глава 8. Задание 7 27](#_Toc492373817)

[Глава 9. Задание 8 30](#_Toc492373818)

[Глава 10. Задание 9 33](#_Toc492373819)

[Глава 11. Задание 10 37](#_Toc492373820)

[Глава 12. Задание 11 41](#_Toc492373821)

[Глава 13. Задание 12 44](#_Toc492373822)

[Глава 14. Заключение 47](#_Toc492373823)

[Библиографический список 48](#_Toc492373824)

[Приложение А. Исходный код программы 49](#_Toc492373825)

[Приложение Б. Блок-схема задания 1 50](#_Toc492373826)

[Приложение В. Блок-схема задания 2 51](#_Toc492373827)

[Приложение Г. Блок-схема задания 3 52](#_Toc492373828)

[Приложение Д. Блок-схема задания 4 53](#_Toc492373829)

[Приложение Е. Блок-схема задания 5 55](#_Toc492373830)

[Приложение Ж. Блок-схема задания 6 58](#_Toc492373831)

[Приложение И. Блок-схема задания 7 59](#_Toc492373832)

[Приложение К. Блок-схема задания 8 60](#_Toc492373833)

[Приложение Л. Блок-схема задания 9 61](#_Toc492373834)

[Приложение М. Блок-схема задания 10 62](#_Toc492373835)

[Приложение Н. Блок-схема задания 11 63](#_Toc492373836)

[Приложение П. Блок-схема задания 12 64](#_Toc492373837)

[Приложение Р. Тесты для задания 3 65](#_Toc492373838)

[Приложение С. Тесты для задания 4 66](#_Toc492373839)

[Приложение Т. Тесты для задания 5 67](#_Toc492373840)

[Приложение У. Тесты для задания 6 68](#_Toc492373841)

[Приложение Ф. Тесты для задания 7 69](#_Toc492373842)

[Приложение Х. Тесты для задания 8 70](#_Toc492373843)

[Приложение Ц. Тесты для задания 10 71](#_Toc492373844)

[Приложение Ш. Тесты для задания 11 72](#_Toc492373845)

[Приложение Щ. Тесты для задания 12 73](#_Toc492373846)

# Введение

*Целью* учебной практики является формирование алгоритмического мышления; закрепление, расширение, углубление и систематизация теоретических знаний, полученных при изучении дисциплин «Введение в программную инженерию», «Дискретная математика», «Компьютерный практикум по основам алгоритмизации и методам программирования», «Программирование»; приобретение навыков и опыта алгоритмизации задач, реализации построенных алгоритмов на языках высокого уровня, публичного выступления при защите отчета о прохождении практики.

*Задачами* учебной практики являются:

1. развитие и закрепление практических навыков построения и описания алгоритмов для решения задач из разных предметных областей (численные методы, дискретная математика, структуры данных и др.);
2. развитие и закрепление практических навыков использования языков высокого уровня и современных сред разработки для реализации построенных алгоритмов;
3. развитие и закрепление практических навыков объектно-ориентированного программирования;
4. развитие практических навыков оформления отчетов о проделанной работе,
5. публичного выступления с защитой проекта;
6. развитие интереса к научно-исследовательской деятельности.

Разработанная система предназначена для решения 12 указанных в варианте задач, а также для предоставления пользователю интерфейса ввода исходных данных этих задач и отображение результатов их решения.

Таким образом, основные функции системы:

1. Решение указанных задач
2. Предоставление пользователю интерфейса для ввода исходных данных
3. Реализация вывода результатов решения поставленных задач

Конечным результатом данной работы является набор Windows-приложений, решающих все данные в варианте 12 задач.

# Задание 1

В качестве задания 1 была предложена задача с сайта ACMP.RU, сайта-платформы, позволяющий решать олимпиадные задачи по программированию различной сложности.

## Анализ задачи

Данная задача представляет из себя типичную задачу на теорию графов. Условия задания представлены на рисунке ‎2.1.

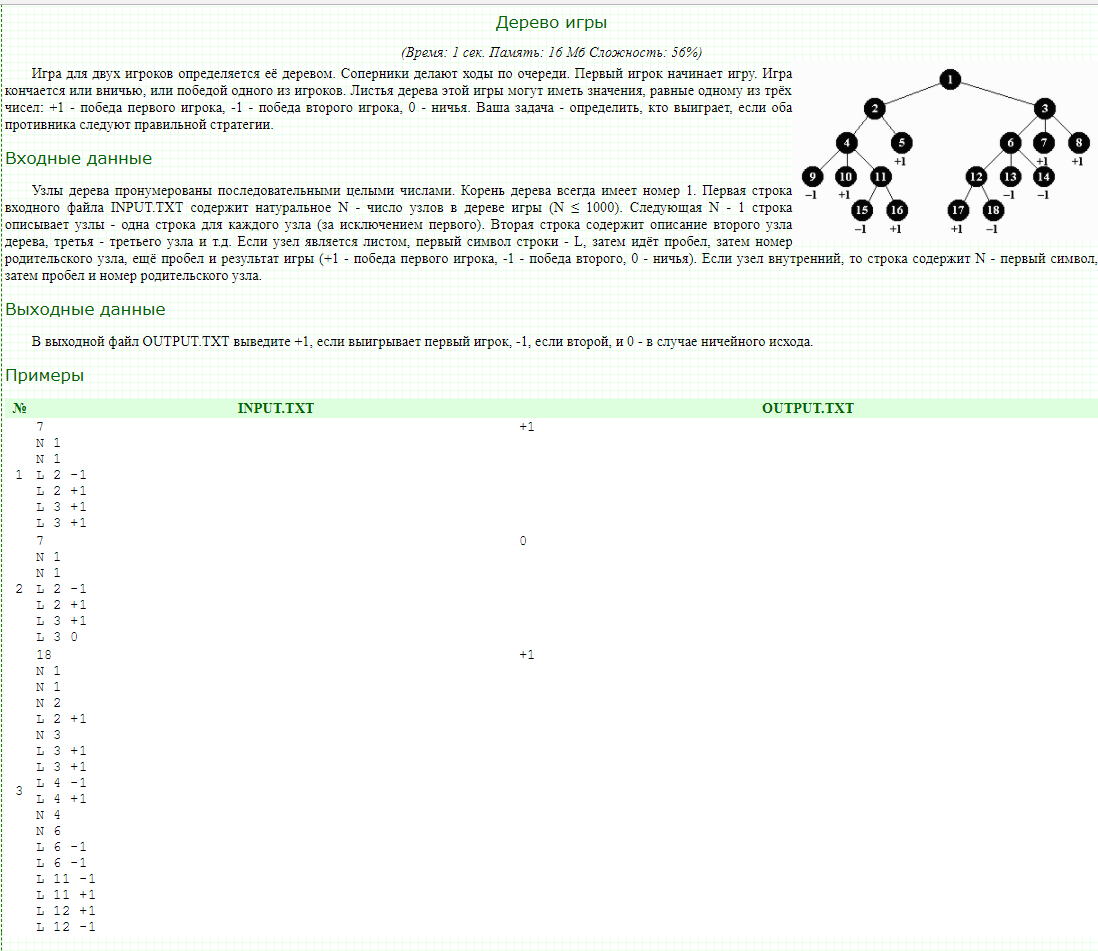


Рисунок ‎2.1. Условия задания 1

Как видно из условия задания, для такой игры возможны три исхода: победа первого игрока, победа второго игрока, ничья. Деревом игры является дерево с фиксированной вершиной, от которой идут узлы. Узлы, в свою очередь, могут быть двух типов: внутренние и концевые. Концевые узлы несут информацию о победителе.

### Входные и выходные данные

Входными данными является информация о каждом из узлов дерева, исключая корневой. В первой строке входного файла содержится натуральное число – суммарное количество узлов дерева, включая корневой узел. Далее в отдельных строках идет описание каждого узла: указывается его тип (внутренний, концевой) – символ («N» для внутреннего либо «L» для концевого), натуральное число – родительский узел, и, если узел является концевым, то победитель – одно из следующих чисел: плюс 1 для победы первого игрока, 0 для ничьей, минус 1 в случае победы второго игрока.

Выходные данные: информация о победителе при условии, что каждый игрок следует правильной стратегии (см. информацию о победителе в конце предыдущего абзаца).

## Разработка алгоритма

Для решения задачи был разработан алгоритм, основная работа которого выполняется с помощью рекурсивного обхода дерева в глубину. Блок-схема данного алгоритма представлена в приложении Б.

Перед тем, как выполнять рекурсивный обход, в программе должно быть создано дерево игры. В ходе считывания входных данных должно быть сконструировано дерево с фиксированным корнем. У каждого узла должен быть список дочерних узлов, который заполняется во время ввода исходных данных. Концевые узлы не имеют дочерних элементов, но с помощью них можно определить победителя. Именно это свойство используется при рекурсивном обходе дерева.

Таким образом, после создания дерева исполняется алгоритм рекурсивного обхода в глубину, задача которого заключается в том, чтобы собрать информацию о победителе при оптимальных стратегиях обоих игроков. Для этого алгоритм вызывается для каждого из дочерних узлов до тех пор, пока их не останется (до концевых узлов). Каждый вызов хранит информацию о текущем игроке, следовательно, становится возможным выбрать оптимальную стратегию для текущего игрока.

На основе информации о победителях, хранящихся в дочерних узлах, и информации о текущем игроке, формируется информация о победителе в текущем узле; так продолжается до тех пор, пока не будет достигнут корень дерева. Результатом работы алгоритма является идентификатор победителя в корневом узле.

## Реализация программной системы

Для реализации программной системы был выбран язык C#. Очевидно, что узел представляет структуру данных (класс), представленную на рисунке ‎2.2. Она имеет частные поля для хранения ссылок на дочерние элементы, ссылку на родительский элемент, информацию о победителе (если концевая вершина, значение «не определено» в случае внутреннего узла).

Класс также имеет методы для работы с ним, например, геттеры/сеттеры для получения информации об узле, метод FillWinners, который представляет собой практическую реализацию алгоритма, представленного ранее.

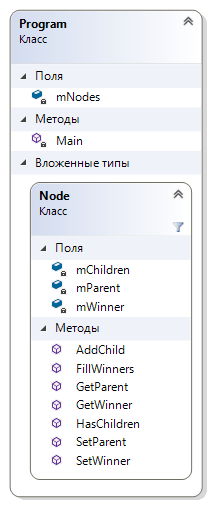


Рисунок ‎2.2. Диаграмма классов для задания 1

В программе дерево представлено массивом элементов, в котором начальный элемент является корневым. Такое решение было подсказано самой формулировкой задачи, так как с помощью такого подхода перевод исходных данных во внутреннюю структуру максимально прост и логичен.

По завершении ввода исходных данных у корневого узла вызывается метод FillWinners, а далее производится вывод идентификатора победителя в выходной файл.

## Тестирование и отладка приложения

Так как приложение было написано для решения задачи на сайте ACMP.RU, тестирование было выполнено на самом сайте. Приложение было протестировано на контрольных тестах из описания задачи, которые представлены на рисунке ‎2.3.

Все тесты успешно прошли проверку, поэтому было принято решение загрузить файл с исходным кодом на тестирующую систему. Из-за различия в компиляторе первый раз программа не скомпилировалась. Поэтому после этого были внесены правки и код был повторно загружен.

Исходный код был удачно скомпилирован, запущен. В ходе тестирования в проверяющей системе он прошел все тесты в установленные временные рамки. Это продемонстрированно на рисунке ‎2.4.

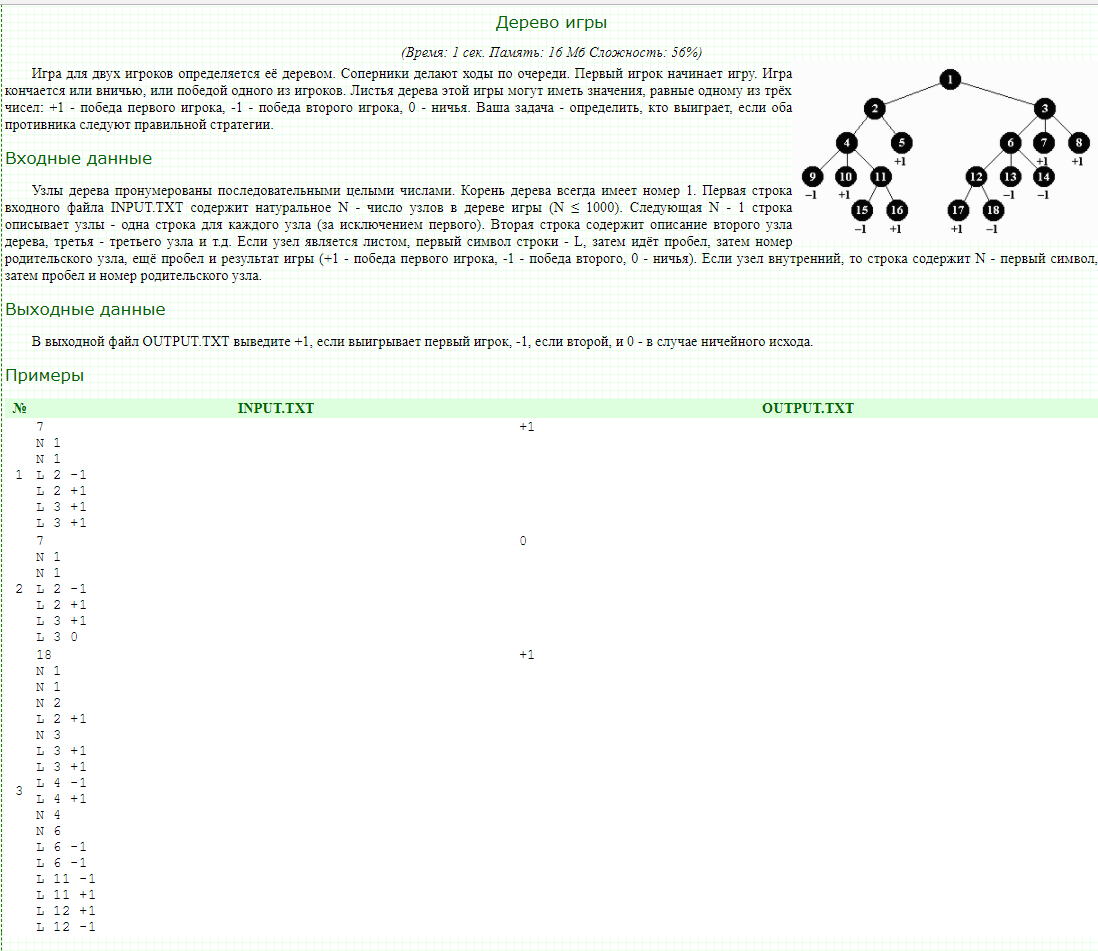


Рисунок ‎2.3. Контрольные тесты

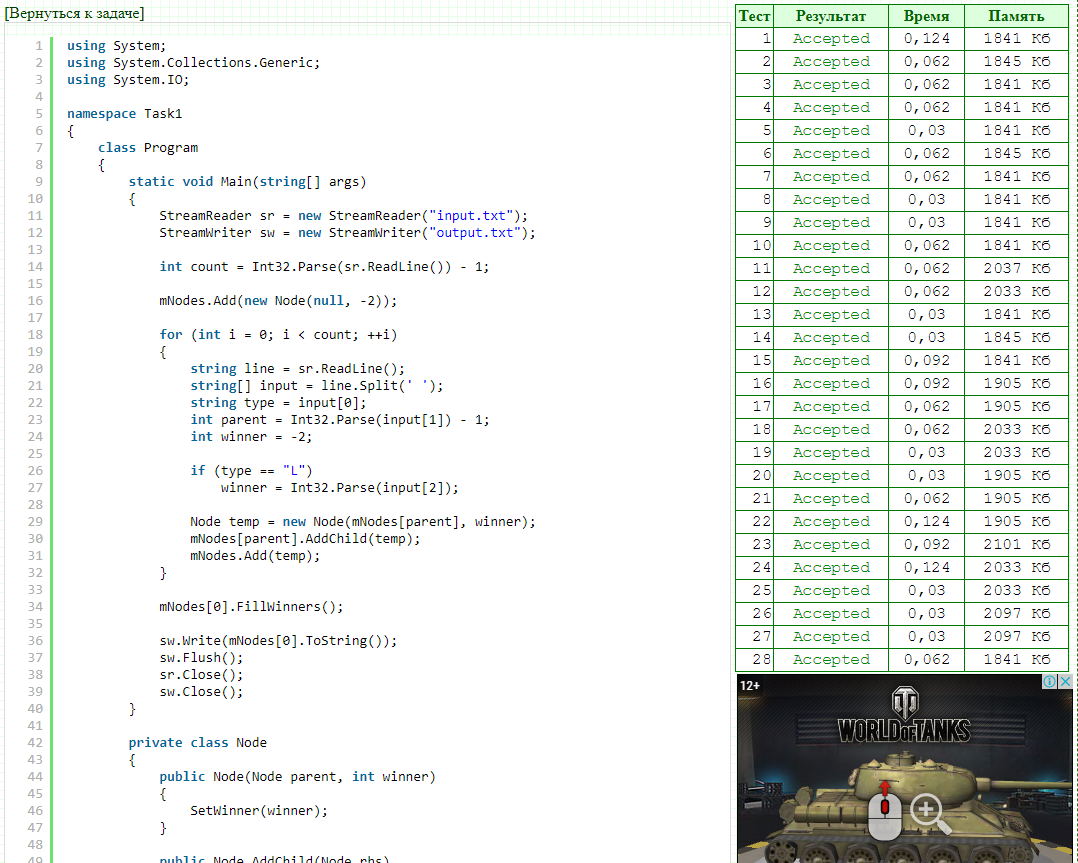


Рисунок ‎2.4. Результаты тестирования

# Задание 2

В качестве задания 2 была также предложена задача с сайта ACMP.RU. Эта задача отличается от предыдущей, по области она принадлежит линейной алгебре.

## Анализ задачи

При анализе условия задачи, представленном на рисунке ‎3.1, можно заметить формулу перемножения матриц. С первого взгляда достаточно реализовать алгоритм перемножения матриц.

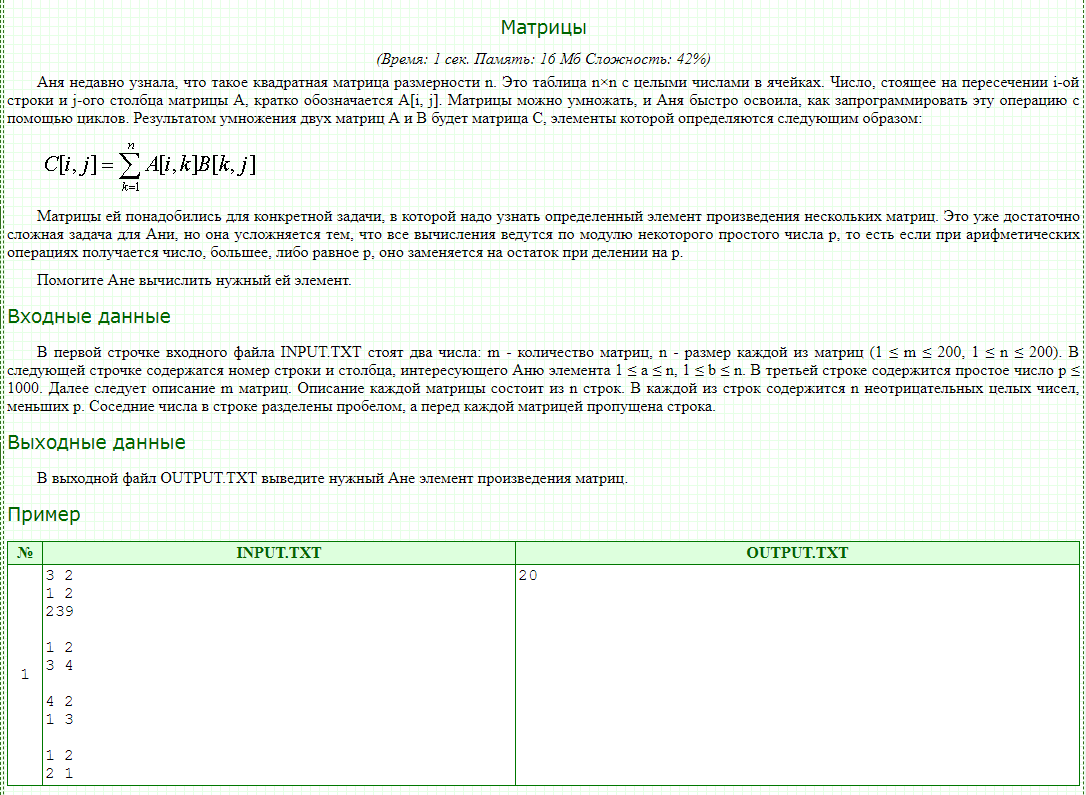


Рисунок ‎3.1. Условия задания 2

При детальном рассмотрении можно отметить, что наивный алгоритм перемножения матриц имеет временную сложность , что недостаточно приемлемо для данной задачи при максимальном количестве матриц и их размере в 200 соответственно. Кроме того, из определения формулы перемножения матриц можно заметить, что для того, чтобы узнать элемент в определенном положении, нужно перемножать лишь строку-вектор первой матрицы и вторую матрицу.

### Входные и выходные данные

Входными данными задачи являются информация о количестве и размере матриц, целевые строка и столбец, модуль, а также содержание самих матриц. В первой строке содержится натуральное число в диапазоне от 1 до 200 – количество матриц, натуральное число в том же диапазоне – размер каждой из матриц. В следующей строке содержатся два натуральных числа, не превышающих вышеописанный размер матриц – строка и столбец элемента, который нужно будет вывести.

В следующих строках содержится описания каждой из матриц. Все матрицы имеют одинаковую размерность.

## Разработка алгоритма

Ранее уже было сказано о недостаточной эффективности наивного алгоритма перемножения матриц, вследствие чего был разработан облегченный алгоритм, применимый для данной задачи. Блок-схема алгоритма приведена в приложении В. Также было отмечено, что для выполнения задачи можно ограничиться лишь перемножением требуемой строки-вектора первой матрицы на вторую матрицу.

В результате такого облегчения в алгоритме вместо трех вложенных циклов осталось лишь два. Также после перемножения выполняется замена значения результата на модуль этого результата, как и требует того задание.

## Реализация программной системы

Данная задача была решена на языке C. Данный язык выбран в качестве языка для решения олимпиадных задач из-за легкости программ, быструю скорость, малое потребление памяти.

Перемножение матриц выполняется функцией mulmodij, ввод матриц – readmatrix. В ходе выполнения программы память под элементы выделяется динамически: выделяется память на хранение массива 200 на 200 – максимального размера матрицы в данной задаче.

На вход главной функции программы mulmodij поступают первая матрица (она уже выделена и является матрицей назначения), вторая матрица, размер матриц, модуль, строка и столбец нужного элемента. Матрицы перемножаются «на лету», то есть выделена память для матрицы результата, она же является первой матрицей, и выделяется память для второй матрицы. Далее считывается матрица из исходного файла и сразу же происходит умножение и применение операции модуля.

На выходе получается строка-результат, вывести требуется лишь элемент с установленными строкой и столбцом, что и делается.

## Тестирование и отладка приложения

После написания приложения оно было протестировано на примерах, данных в условии задачи. На рисунке ‎3.2 видны исходные данные и соответствующие им выходные данные.

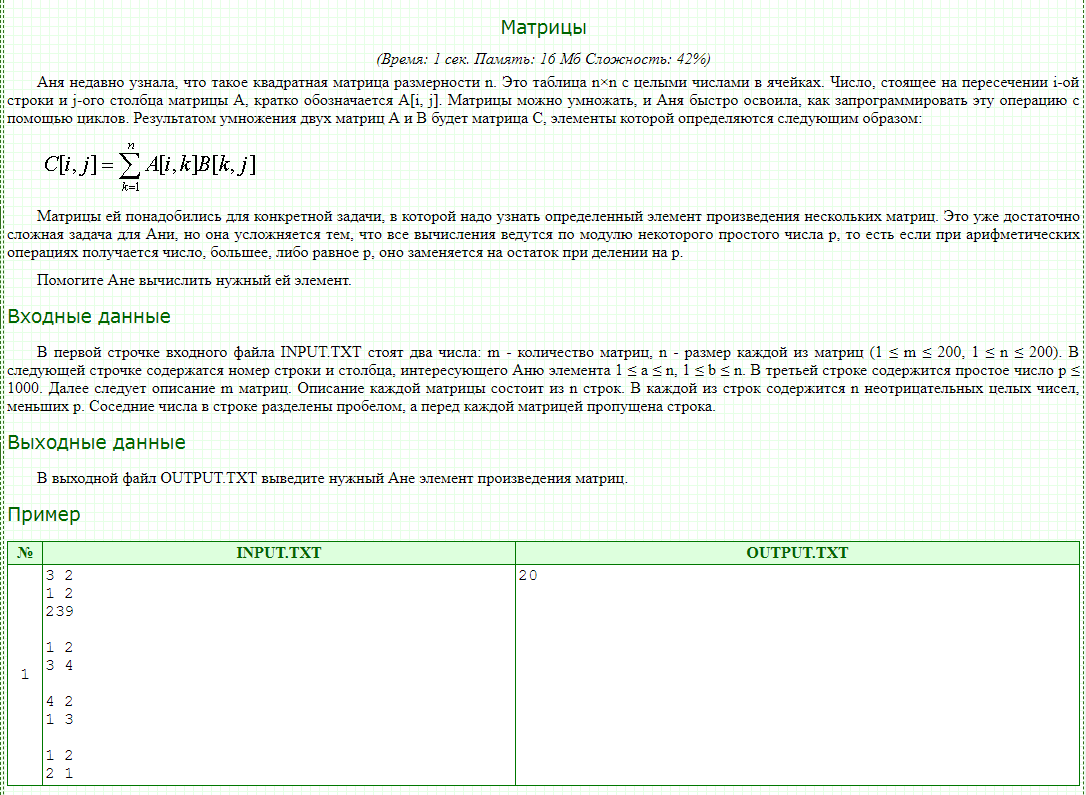


Рисунок ‎3.2. Контрольные примеры

После прохождения контрольных тестов исходный код решения был отправлен на тестирующую систему ACMP.RU. Результаты выполнения видны на рисунке ‎3.3.

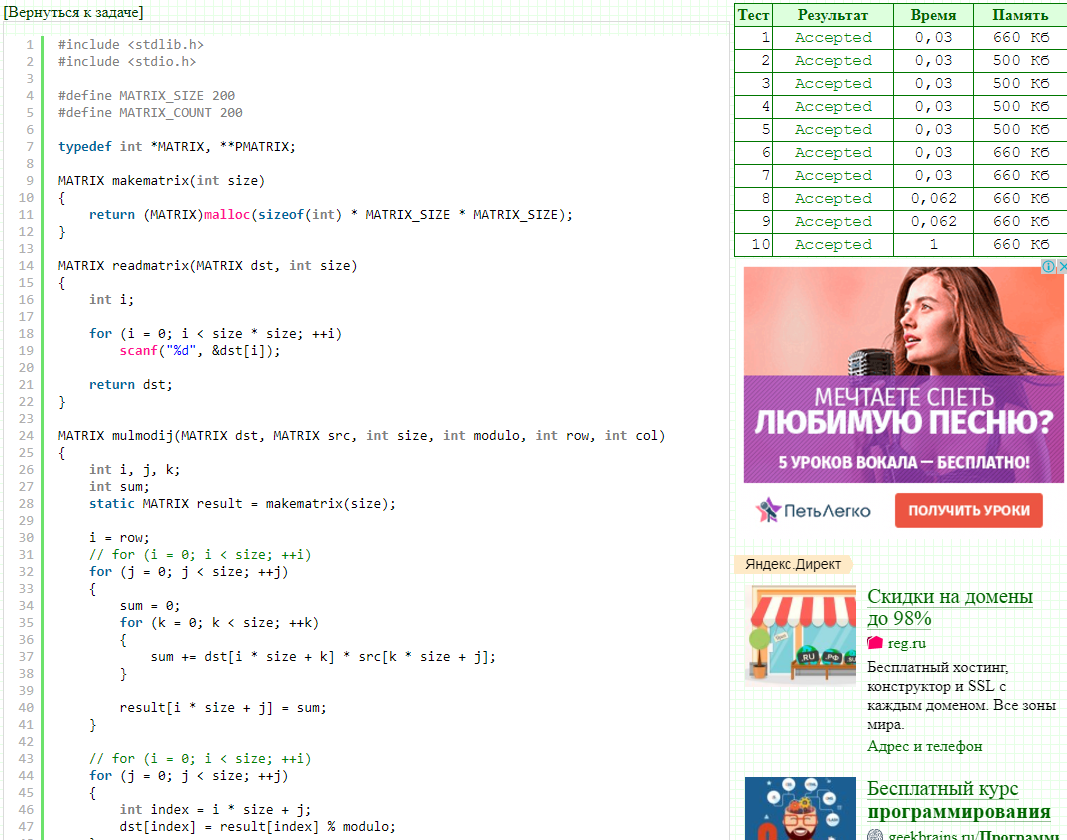


Рисунок ‎3.3. Результаты тестирования задания 2

Таким образом, задание 2 было решено и протестировано. Все тесты оказались успешными, программа завершилась в установленные временные рамки, а использование памяти не превысило допустимые пределы.

# Задание 3

Задание 3 представляет собой задачу на определение принадлежности той или иной точки определенной области плоскости. Так как задача геометрическая, то графическое решение более эргономично.

## Анализ задачи

Задача взята из книги [1], *вариант д*. Условие задачи можно видеть на рисунке 4.1.

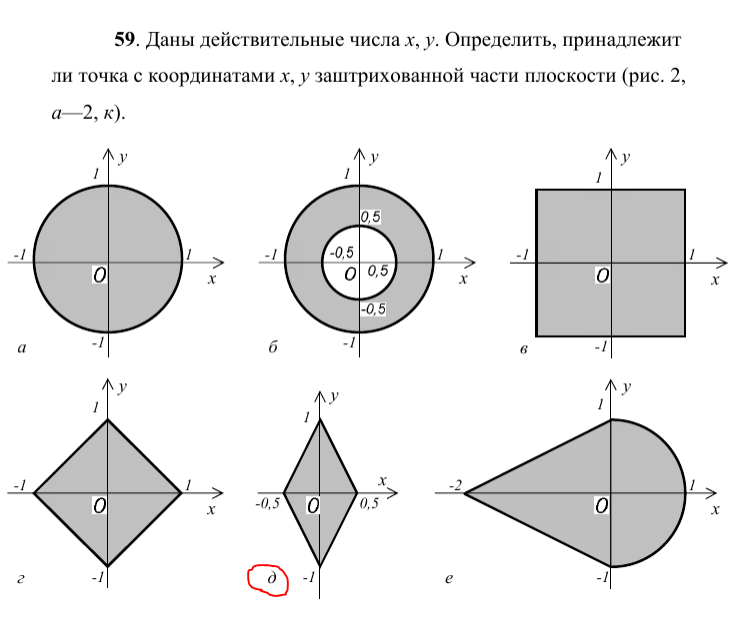


Рисунок ‎4.1. Условия задания 3

В качестве фигуры дан ромб высотой 2, шириной 1. Середина ромба лежит в начале координат. В качестве возможного решения можно разделить ромб на две симметричные относительно оси абсцисс части – верхнюю и нижнюю. Нетрудно заметить, что каждая из таких частей представляет из себя график модуля, смещенного по оси ординат. Для предварительной проверки попадания можно использовать окружающий прямоугольник, ориентированный по осям.

### Входные и выходные данные

В качестве **входных данных** служит пара действительных чисел – абсцисса и ордината точки, для которой требуется проверить принадлежность фигуре. Фигура, по условию задачи, является константой.

На **выходе** получаем единственное булевое значение – информация о том, принадлежит ли точка данной фигуре.

## Разработка алгоритма

Исходя из предположений выше, был разработан алгоритм определения принадлежности точки данной фигуре. Блок-схему данного алгоритма можно увидеть в приложении Г.

Для определения принадлежности точки заданному ромбу вычисляются координаты окружающего прямоугольника. Если точка находится внутри окружающего прямоугольника, то вычисляется значение функции модуля в данной точке. Так как ромб симметричен, то значение функции также симметрично относительно прямой, проходящей через центр ромба и параллельной оси абсцисс.

Таким образом, после вычисления значения функции модуля в точке, абсцисса которой равна абсциссе заданной точки, ордината заданной точки сравнивается с полученным значением функции. Если ордината не превышает верхней половины ромба и не ниже нижней половины ромба, то точка принадлежит данной фигуре. В любом другом случае точка лежит вне фигуры.

## Реализация программной системы

Для практической реализации задачи был выбран язык VB.NET. Обладая всеми преимуществами платформы .NET, данный объектно-ориентированный язык не уступает языку C# по функционалу. Так как по концепции языки схожи, то было принято решение писать на VB.NET для расширения опыта и применения конструкций, которые присутствуют в языке C#, в VB.NET.

Так как задание предполагает использование знаний геометрии, было принято решение реализовать программную систему на Windows Forms, сделав удобный графический интерфейс. Вместо введения координат точек вручную предложено решение с тем же функционалом, но упрощающим введение точек – графическим интерфейсом. В центре окна находится фигура, в данном случае, ромб, видны оси координат и разметка. Дополнительно вид можно масштабировать. Внешний вид приложения приведен на рисунке ‎4.2.

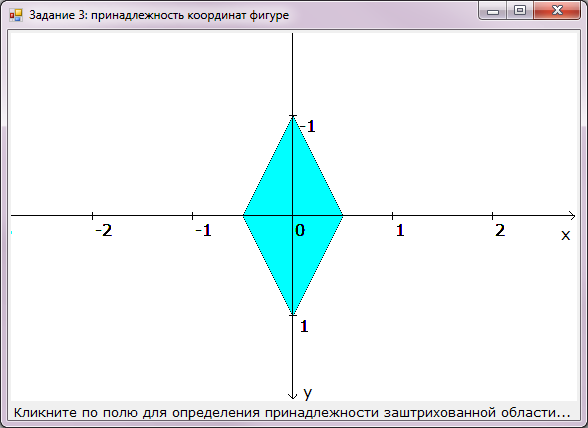


Рисунок ‎4.2. Внешний вид программы

В начале работы пользователю предлагается ввести точку путем щелчка на поле. Соответственно, после щелчка выводится информация о принадлежности точки данной фигуре в строке статуса.

Для выполнения задачи было реализовано несколько классов: Diamond, Plane. Вся логика, касающаяся физического расположения ромба, находится в классе Diamond.

Класс Plane представляет из себя модель математической плоскости. Класс рисует на графической поверхности оси координат, отметки на них, и фигуры, которые добавляются с помощью AddShape. Таким образом, возможно иметь несколько ромбов на плоскости, но задание требует лишь одного. Класс Plane имеет метод Test, который в свою очередь вызывает одноименный метод у экземпляра Diamond. Этот метод инкапсулирует всю проверку на принадлежность и возвращает булевое значение.

Класс Diamond содержит метод отрисовки Draw и проверки на принадлежностьTest, который принимает координаты и возвращает булевое значение. Частные поля данного класса содержат информацию о местоположении, габаритах фигуры. Диаграмма классов представлена на рисунке ‎4.3.

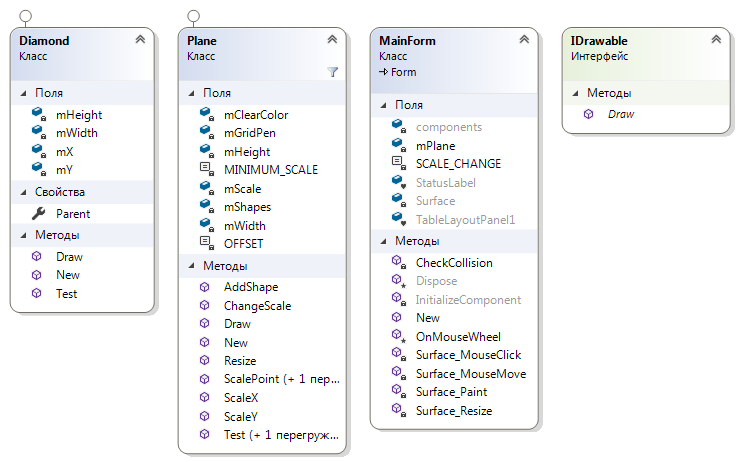


Рисунок ‎4.3. Диаграмма классов задания 3

Таким образом, в качестве решения задачи была написана программная система с графическим интерфейсом и наглядной демонстрацией заданной фигуры. Система была расширена с помощью дополнительного функционала, что увеличило удобство пользования и возможности пользователя.

## Тестирование и отладка приложения

После написания основных модулей программы было проведено ее тестирование в соответствии с условиями задания. От программы ожидается положительный результат, если заданная точка лежит внутри ромба шириной 1, длиной 2, отрицательный результат во всех остальных случаях. Так как ширина и высота являются целыми числами, то точки, совпадающие с координатами вершин ромба, должны возвращать положительный результат.

Так как входными данными являются пары , где первое значение – абсцисса точки, а второе – ордината, то были подобраны тесты по следующим критериям:

1. Начало координат
2. Точка внутри ромба
3. Точка, совпадающая с одной из вершин ромба
4. Точка, находящаяся вне ромба

Таблица тестов для задания 3 приведена в приложении Р.

# Задание 4

Задание 4 – задача на нахождение десятичных цифр длинного числа. Как известно, факториал числа возрастает очень быстро, поэтому при небольших значениях исходного числа его факториал не может быть помещен в стандартные целочисленные типы.

## Анализ задачи

Задача так же, как и предыдущая, взята из книги [1]. Ее условия приведены на рисунке ‎5.1. При рассмотрении задачи один из вариантов решения – реализация длинной арифметики.

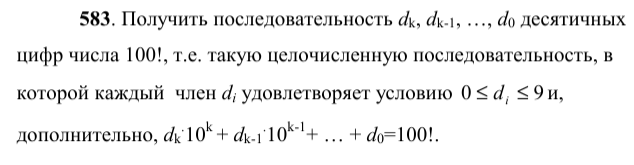


Рисунок ‎5.1. Условие задания 4

Из формулировки задания можно сделать вывод, что реализация длинной арифметики – подходящий подход.

### Входные и выходные данные

В исходной формулировке задания входных данных как таковых нет. На вход поступает число «100» – константа. Для обеспечения большей гибкости было принято решение получать на вход целое неотрицательное число – число, для которого будет рассчитываться факториал.

Выходные данные: набор десятичных цифр числа «100!» в исходной формулировке, набор десятичных цифр введенного числа в измененной формулировке.

## Разработка алгоритма

Для данной задачи был разработан алгоритм – алгоритм нахождения длинного факториала. Главная работа выполняется при выполнении сложения и умножения чисел. Блок-схема этих действий представлена в приложении Д.

Для достижения поставленной цели структура данных, представляющая из себя длинное число, должна иметь массив разрядов, начиная со младшего. При сложении и умножении, как и в математике, происходят переносы разрядов. Для этого требуется процедура переноса (вместе с другими представлена в приложении Д).

## Реализация программной системы

Для реализации задачи был выбран язык VB.NET. Как уже говорилось ранее, язык основан на платформе .NET, поэтому перенос знаний языка C# в другой похожий язык является получением новых знаний и полезной практикой.

Было принято оформить интерфейс при помощи Windows Forms. Внешний вид программы представлен на рисунке ‎5.2.

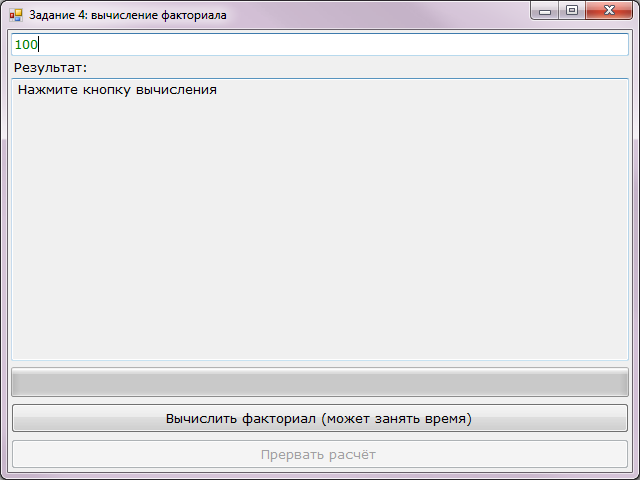


Рисунок ‎5.2. Внешний вид задания 4

Вычисление факториала числа 100 – довольно длительная процедура (по времени), а вычисление факториалов больших чисел занимает еще больше времени. Вследствие этого было принято решение вынести вычисление факториала в другой поток, для того чтобы не создавать неудобство пользователю. Если вычисление кажется пользователю слишком долгим, в интерфейсе предусмотрена кнопка «Прервать расчет». По ее нажатии расчет прерывается, и пользователь уведомляется, что получение искомого результата было прервано.

На форме присутствует компонент ProgressBar, который используется для индикации прогресса. При вычислении факториала числа 100 его использование может быть неоправданным, но при вычислении больших чисел это дает пользователю информацию о текущем прогрессе.

В качестве дополнительного функционала реализованы счетчик использования центрального процессора системой, памяти и параллельный алгоритм вычисления факториала. Внешний вид окна в процессе вычисления представлен на рисунке ‎5.3.

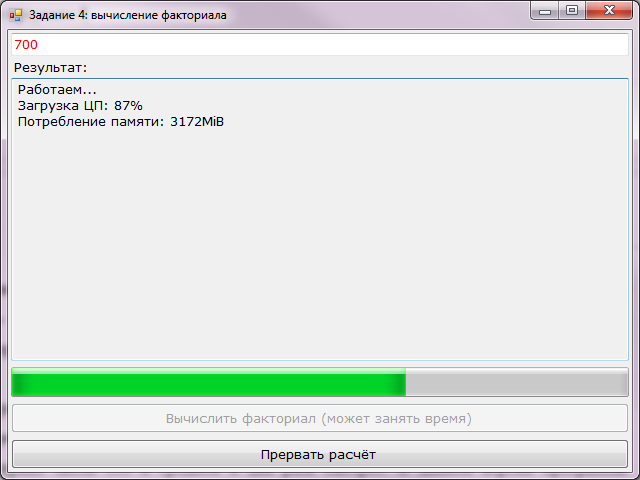


Рисунок ‎5.3. Внешний вид в процессе вычисления значения

По результатам тестов на 4-ядерном процессоре параллельный алгоритм вычисляет факториал числа 100 в среднем в два раза быстрее. В данной версии программы факториалы чисел больше 20 вычисляются параллельным алгоритмом, однопоточный алгоритм работает с маленькими числами.

## Тестирование и отладка приложения

Так как приложение вычисляет факториал длинного числа, было принято решение проводить тест, а затем сравнивать ответ с эталоном. В качестве эталона был выбран онлайн-сервис Wolfram Alpha (<http://www.wolframalpha.com>). Тесты для программы подбирались по следующим критериям:

1. Не число (строка символов)
2. Отрицательное число
3. Ноль
4. Число 100
5. Натуральное число

Тесты для данного приложения приведены в приложении С.

# Задание 5

Задание 5 взято из книги [1] и представляет собой задачу на работу с массивами, в частности, двумерным массивом фиксированного размера.

## Анализ задачи

Условия задачи (для *варианта в*) предполагают заполнение матрицы 8 на 8 определенным образом. В данном случае заполнение должно происходить диагонально с возрастанием номеров от верхнего левого края до правого нижнего края. Условия задачи приведены на рисунке ‎6.1.

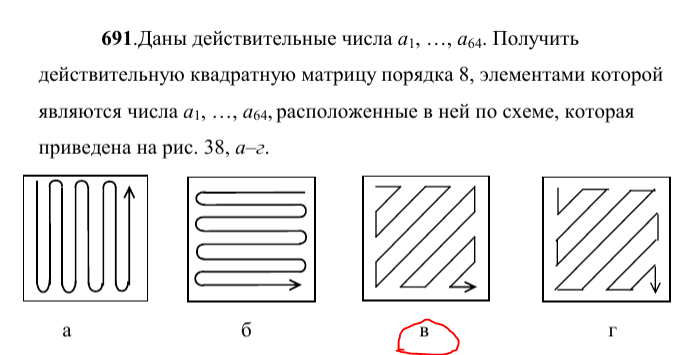


Рисунок ‎6.1. Условия задания 5, вариант в

Как видно по иллюстрации, движение по матрице должно происходить определенным образом до достижения препятствия этому движению. После достижения препятствия в правом верхнем углу движение продолжается, но только по измененной траектории.

### Входные и выходные данные

Входные данные: 64 действительных числа, которые следует расположить в матрице.

Выходные данные: матрица 8 на 8, построенная заданным образом.

## Разработка алгоритма

Для решения задачи был разработан алгоритм заполнения матрицы произвольного размера заданными элементами вышеописанным образом. Блок-схему алгоритма можно увидеть в приложении Е.

Алгоритм состоит из двух важных частей: основной алгоритм и алгоритм изменения направления движения указателя в матрице при заполнении. Основной алгоритм перебирает все элементы входного массива и записывает их в матрицу 8 на 8 с помощью алгоритма смены движения.

## Реализация программной системы

Программа написана на языке VB.NET. Большинство оснований применения данного языка программирования приведены в двух предыдущих главах. Одна из главнейших причин – возможность практики написания программы на другом объектно-ориентированном языке программирования с применением знаний, полученных в ходе изучения языка C# и платформы .NET.

Интерфейс приложения графический, создан с использованием Windows Forms. Пользователю предлагаются элементы для обработки по умолчанию, у него есть возможность их модифицировать. Внешний вид приложения представлен на рисунке ‎6.2.

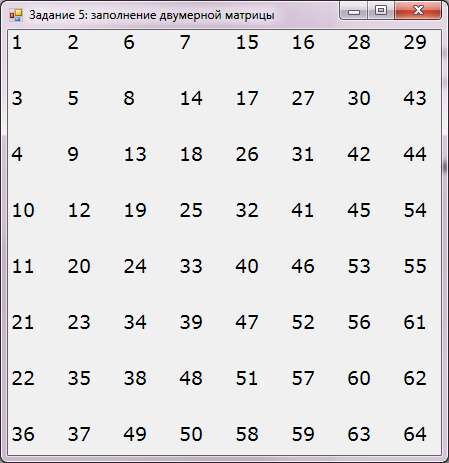


Рисунок ‎6.2. Внешний вид задания 5

В ходе реализации программной системы было создано несколько классов: SquareMatrix, абстрактный Inserter, классы DefaultInserter, DiagonalInserter. Диаграмма классов приведена на рисунке ‎6.3.

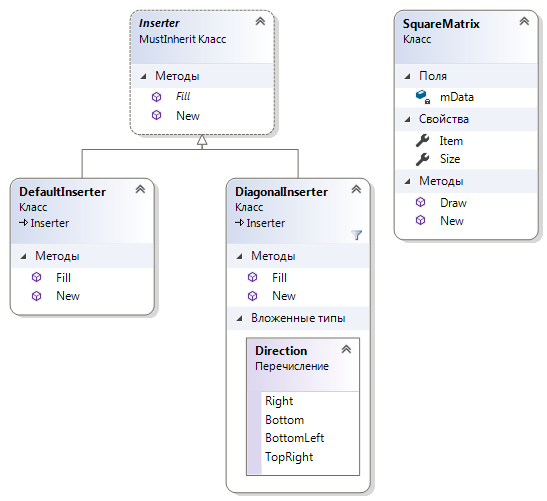


Рисунок ‎6.3. Диаграмма классов задания 5

Класс Inserter содержит абстрактный метод Fill, который является интерфейсом для класса SquareMatrix. SquareMatrix использует экземпляр одного из потомков класса Inserter для заполнения самой себя с помощью метода Fill. Этот метод принимает на вход ссылку на матрицу и массив элементов, которыми матрицу следует заполнить.

## Тестирование и отладка приложения

# Задание 6

Задание 6 – задача на составление рекуррентной последовательности. Самым очевидным и удобочитаемым способом реализовать данную задачу является использование рекурсивных функций.

## Анализ задачи

Последовательность, представленная в задаче, является расходящейся. Условия задачи приведены на рисунке ‎7.1.



Рисунок ‎7.1. Условия задания 6

Задачей является продолжить последовательность рекурсивно. На практике рекурсивный способ, как правило, более удобочитаем, чем итеративный, но значительно уступает в эффективности. Так как при рекурсии функция вызывает саму себя, то при большой рекурсивной вложенности стек приложения может переполниться и вызвать ошибку времени исполнения (хотя во многих современных компиляторах ЯП встроена оптимизация хвостовой рекурсии при выборе режима оптимизации кода, что дает рекурсивному способу еще один аргумент «за»).

### Входные и выходные данные

Входные данные: три действительных числа – первые три члена последовательности, действительное число – лимит последовательности.

Выходные данные: продолженная последовательность, натуральное число – номер элемента, при котором последователь впервые становится не меньше лимита, булевое значение – информация о том, является ли член последовательности, впервые ставший большим или равным лимиту, равен ли он лимиту.

## Разработка алгоритма

Алгоритм для решения задачи довольно прост, так как большая часть его работы – продолжение последовательности по вышеуказанной формуле и проверка элементов на достижение предела. Блок-схема алгоритма приведена в приложении Ж.

При построении последовательности изначально проводятся проверки исходных элементов на предмет достижения предела. Далее эта же проверка выполняется для каждого следующего элемента продолженной прогрессии.

## Реализация программной системы

Программа реализована на языке C#. Из-за простоты входных данных было отдано предпочтение консольному интерфейсу. Все четыре исходных числа должны быть введены в одну строку.

После ввода исходных данных исполняется главный алгоритм и выводится результат. Пример такого взаимодействия можно наблюдать на рисунке ‎7.2.

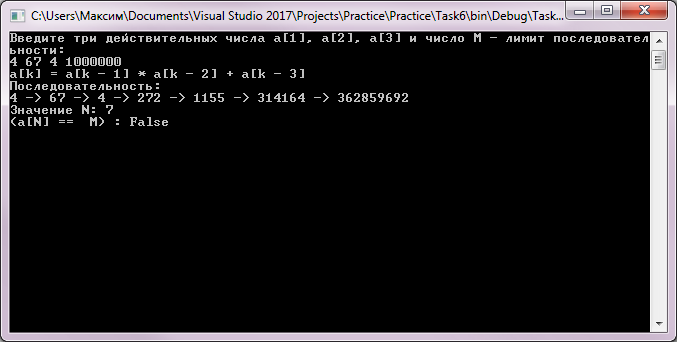


Рисунок ‎7.2. Пример взаимодействия пользователя с консольным интерфейсом задания 6

После ввода данных пользователь получает информацию о рассчитываемой последовательности в символьном виде, а далее и в конкретном числовом виде для заданных значений. Выводится значение , предел сравнивается с элементом .

## Тестирование и отладка приложения

Приложение принимает на вход 4 значения: 3 начальных значения последовательности и предел. Данная в условии последовательность всегда расходится, исключение составляет случай, когда все 3 начальных элемента равны нулю.

Основываясь на этих данных, можно выделить следующие критерии для тестирования:

1. Одно или более параметров не являются числами
2. Начальные значения, равные нулю и такой же предел
3. Начальные значения, равные нулю, предел больше нуля
4. Начальные значения, равные нулю, предел меньше нуля
5. Начальные значения, при которых предел меньше первого элемента
6. Начальные значения, при которых предел меньше второго элемента
7. Начальные значения, при которых предел меньше третьего элемента
8. Начальные значения, при которых предел равен одному из элементов последовательности
9. Начальные значения, при которых предел не равен одному из элементов последовательности

Тесты для данного задания представлены в приложении У.

# Задание 7

Задание 7 представляет собой задачу на кодирование информации. При решении был использован материал из курса «Дискретная математика», раздел «Кодирование».

## Анализ задачи

Условия задачи представлены на рисунке ‎8.1. Задание требует добавить контрольные разряды и заполнить их. Возможно разделить это задание на два шага: собственно, добавление битов и их заполнение.



Рисунок ‎8.1. Условия задания 7

### Входные и выходные данные

Входные данные: содержание информационных разрядов кодового слова кода Хэмминга, представленное в двоичном виде.

Выходные данные: кодовое слово с добавленными и заполненными контрольными битами.

## Разработка алгоритма

Алгоритм, разработанный для решения данной задачи, предполагает ввод данных как булевый массив. Его блок-схема представлена в приложении И. Алгоритм состоит из двух главных частей: добавление контрольных разрядов и их заполнение.

После формирования массива с исходными данными он посылается алгоритму. Алгоритм копирует поэлементно исходный массив в массив-результат, при этом оставляя место для контрольных битов (каждый бит).

После заполнения массив посылается процедуре заполнения контрольных битов, которая заполняет их в соответствии с требованиями кода Хэмминга. -ый бит контролирует информационных битов включая себя, информационных битов через такое же количество битов, и так далее, до конца информационного массива.

## Реализация программной системы

Программа реализована на языке C#, интерфейс создан при помощи Windows Forms. Внешний вид приложения можно наблюдать на рисунке ‎8.2.

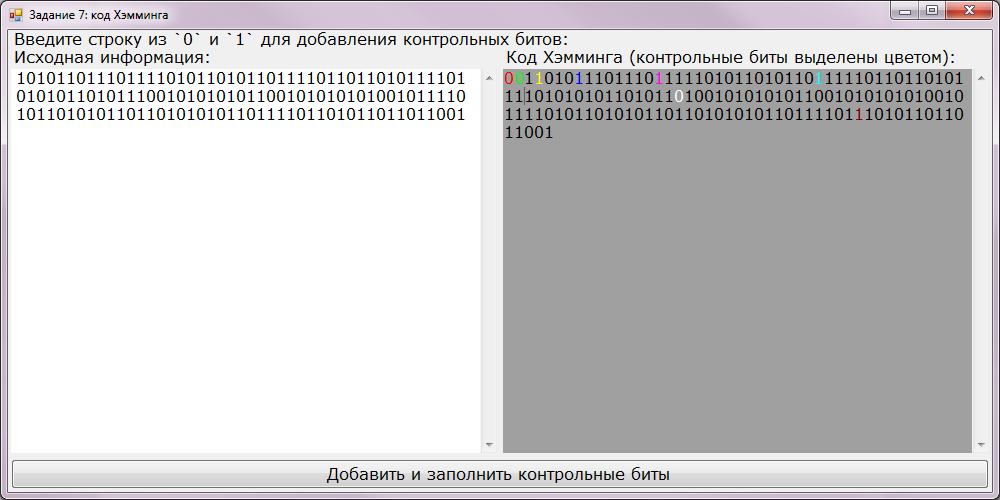


Рисунок ‎8.2. Внешний вид задания 7

Для поля ввода предусмотрена строгая фильтрация ввода: можно вводить лишь «0», «1», стирать символы. При нажатии на кнопку происходит добавление и заполнение контрольных битов. В качестве дополнительного функционала была реализована подсветка контрольных битов для удобства пользования программой.

Подсветка осуществляется благодаря использованию элемента управления RichTextBox вместо обычного TextBox. Каждый бит будет иметь разный цвет в зависимости от его положения.

Класс Utility был создан для того, чтобы избежать повтора кода. Статический метод IsPowerOfTwo используется как в классе HammingBuffer, так и в классе MainForm. Этот метод помогает определить, является ли целое число степенью двойки с помощью так называемой «битовой магии»: если число является степенью двойки, то результат побитового «И» с числом в битовом представлении даст ноль.

Основную работу в программе выполняет класс HammingBuffer. Он имеет конструктор с параметром, являющимся булевым массивом. При построении данного класса используются вышеописанные алгоритмы. Метод Init копирует содержание исходного массива и добавляет контрольные биты. Метод SetControlBits отвечает за заполнение контрольных битов в полученном массиве Хэмминга. Диаграмма классов представлена на рисунке ‎8.3.

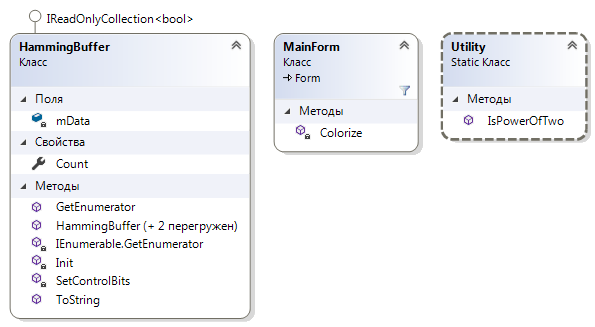


Рисунок ‎8.3. Диаграмма классов задания 7

HammingBuffer реализует интерфейс IReadOnlyCollection<bool>, что позволяет использовать его в циклах foreach, но не позволяет модификацию данных коллекции, в том числе и контрольных битов.

## Тестирование и отладка приложения

Для данного приложения был проведен ряд тестов. Входные данные можно разделить на следующие категории:

1. Пустое поле ввода
2. Ввод, содержащий помимо «0» и «1» другие символы
3. Ввод нуля
4. Ввод единицы
5. Ввод корректной битовой строки

Так как ввод фильтруется, то в поле ввода можно печатать лишь разрешенные символы, так что пользовательский ввод имеет мало шансов принадлежать категории 2[[1]](#footnote-1). Тесты для данной программы представлены в приложении Ф.

# Задание 8

Задание 8 представляет из себя задачу на теорию графов. Данные задачи решались на курсе «Дискретная математика» в разделе «Графы». Как и в случае с проверкой точки на принадлежность плоскости, создание визуального редактора графа намного облегчает ввод информации пользователем.

## Анализ задачи

Условие задачи представлено на рисунке ‎9.1. В графе, заданном матрицей инциденций, нужно найти пустой подграф размера , то есть такой граф, в котором нет ребер.



Рисунок ‎9.1. Условия задания 8

Так как граф задан матрицей инциденций, то решение довольно очевидно: перебирать все вершины и добавлять в пустой подграф те из них, которые не имеют ребер.

### Входные и выходные данные

Входные данные: граф, заданный матрицей инциденций, натуральное число – количество вершин в пустом подграфе.

Выходные данные: информация о пустом подграфе заданного размера, список вершин, образующих его.

## Разработка алгоритма

Для выполнения задания был разработан алгоритм, представленный в приложении К. Процедура нахождения пустого подграфа использует процедуру определения наличия ребер у определенной вершины.

После создания графа можно определять наличие у него пустого подграфа. Алгоритм является ленивым, так как при наличии двух и более пустых подграфов заданного размера алгоритм выводит вершины первого попавшегося подграфа. Жадные алгоритмы просты в реализации и довольно просты для понимания. Так как условие задачи не противоречит использованию такого алгоритма, то он и был выбран для решения задачи.

## Реализация программной системы

Задача реализована на языке C#, интерфейс создан при помощи Windows Forms. Внешний вид приложения можно видеть на рисунке ‎9.2. В дополнение к основной функции программы было принято решение сделать удобный редактор графа, так как ввод графа в текстовом виде не является наглядным и таким быстрым, как графический редактор, к тому же возможность ошибиться при вводе графа в виде текста довольно велика, а графический редактор позволяет удалять ошибочные вершины и создавать новые вместо них.

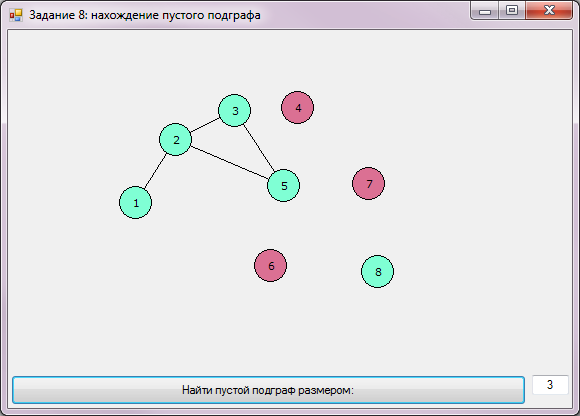


Рисунок ‎9.2. Внешний вид и нахождение пустого подграфа

На рисунке выше представлен построенный граф, в нем найден пустой подграф заданного в элементе TextBox размера – 3 (подсвечен фиолетово-красным цветом). Как видно из полученного пустого подграфа, номера его вершин минимальны, хотя вершина 8 так же могла быть бы частью пустого подграфа. Это объясняется использованием жадного алгоритма, как уже было оговорено ранее. Для достижения такого результата был разработан класс Graph.

Класс Graph выполняет всю работу, связанную с графом: добавление вершин, удаление вершин, выделение вершин цветом, перемещение, соединение и т.д. Конечно же, главная функция этого класса: нахождение пустого подграфа заданного графа. В ней реализован вышеописанный алгоритм нахождения пустого класса. Диаграмма классов представлена рисунке 9.3.

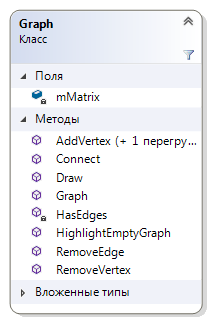


Рисунок ‎9.3. Диаграмма классов задания 8

Метод класса Graph HighlightEmptyGraph реализует вышеописанный алгоритм нахождения пустого подграфа заданного размера. Частный метод-помощник HasEdges проверяет, имеет ли заданная вершина ребра.

## Тестирование и отладка приложения

После разработки приложение было оттестировано, для него был составлен ряд тестов. Критерии входных данных:

1. Отсутствие вершин
2. Размер пустого подграфа равен 0
3. Наличие вершин, но отсутствие пустого подграфа
4. Наличие как вершин, так и пустого подграфа

Тесты приведены в приложении Ф.

# Задание 9

Задание 9 представляет собой задачу на создание коллекции определенным способом, в данном случае, нужно создать коллекцию типа «двусвязный список» с помощью рекурсии.

## Анализ задачи

Условие задания можно видеть на рисунке ‎10.1. Как видно из формулировки задания, задача заключается в создании двунаправленного списка рекурсивным способом.

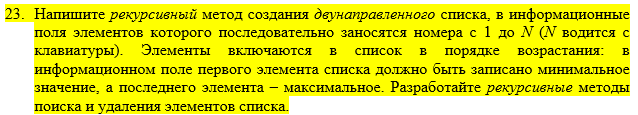


Рисунок ‎10.1. Условие задания 9

Так как элементы списка имеют в качестве информационных полей последовательность натуральных чисел, то при создании списка единственный параметр – количество элементов.

### Входные и выходные данные

Входные данные: количество элементов в целевом списке.

Выходные данные: сконструированный список с заданным количеством элементов.

## Разработка алгоритма

Для решения поставленной задачи был разработан алгоритм построения списка, алгоритм нахождения заданного элемента. Удаление элемента использует алгоритм поиска. Блок-схема алгоритмов представлена в приложении Л.

Список строится следующим образом: формируется элемент с текущим значением, далее вызывается функция конструирования со значением на единицу меньше текущего. Новая вершина формируется перед ранее сформированной. Эти действия выполняются до тех пор, пока значение, подаваемое в функцию, не окажется равным нулю.

Алгоритм поиска проверяет, является ли текущая вершина той, которая содержит заданный элемент. Если является, то возвращается индекс данной вершины, иначе действие повторяется до достижения нулевой вершины.

Алгоритм удаления используем реализацию поиска – с его помощью находится элемент для удаления. Если элемент найден, то он удаляется, функция удаления вызывает себя повторно для удаления всех таких элементов, иначе не удаляется ничего.

## Реализация программной системы

Задача реализована на языке C#, интерфейс создан при помощи Windows Forms. Внешний вид приложения можно увидеть на рисунке ‎10.2. На форме представлено текстовое поле для вывода содержимого списка, кнопки для управления процессами создания, удаления элементов в списке, кнопка для нахождения элемента, а также цифровые поля для указания размера и искомого элемента.

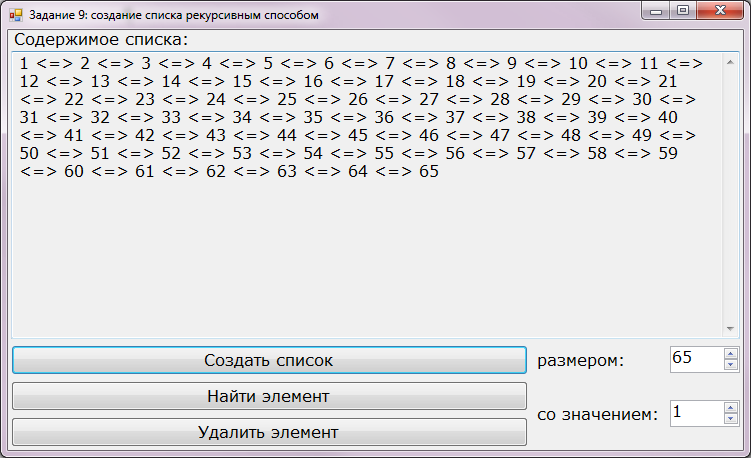


Рисунок ‎10.2. Внешний вид задания 9

Как видно из рисунка выше, в программе был создан список размером 65. Содержимое списка выводится в текстовое поле. Также видны элементы управления, отвечающие за функции поиска и удаления элементов с определенным значением.

Для реализации функционала приложения был создан класс IntList. Он содержит методы конструирования, нахождения требуемого элемента и удаления элемента. Он реализует интерфейс ICollection<int>, что позволяет ему быть использованным в циклах foreach. При попытке найти или удалить несуществующий элемент пользователь уведомляется о его отсутствии с помощью диалогового окна. Диаграмма классов представлена на рисунке ‎10.3.

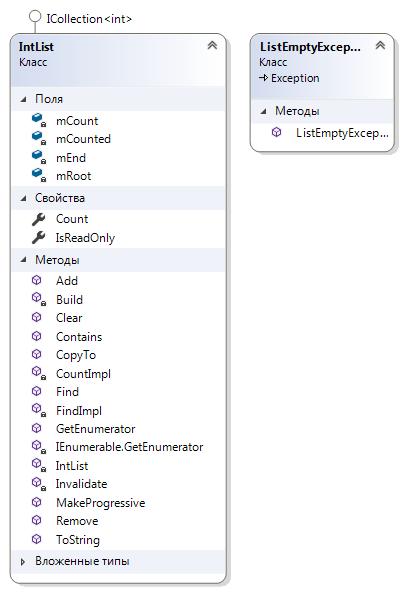


Рисунок ‎10.3. Диаграмма классов задания 9

Класс IntList представляет методы для создания экземпляра (MakeProgressive), для поиска (Find, реализация FindImpl) и удаления элемента (Remove, основная работа по поиску в FindImpl).

## Тестирование и отладка приложения

После разработки программы было проведено тестирование программы. Тесты представляют из себя сценарии. Так как функции работы со списком должны работать только после его создания, их вызов перед тем, как список создан, должен породить ошибку. Именно поэтому в программе кнопки функций работы со списком являются неактивными до того момента, пока список не будет создан. Размер списка – натуральное число, некорректный ввод предотвращается элементом управления NumericUpDown.

Сценарии тестирования:

1. Попытка поиска отсутствующего элемента:
   1. Создать список размером 1
   2. Найти элемент со значением 2
   3. Ошибка, элемент не найден
2. Попытка удаления несуществующего элемента:
   1. Создать список размером 1
   2. Удалить элемент со значением 2
   3. Ошибка, элемент для удаления не найден
3. Удаление элемента, далее попытка поиска того же элемента:
   1. Создать список размером 10
   2. Удалить элемент со значением 5
   3. Найти элемент со значением 5
   4. Ошибка, элемент не найден
4. Удаление элемента, далее попытка поиска существующего элемента:
   1. Создать список размером 10
   2. Удалить элемент со значением 5
   3. Найти элемент со значением 6
   4. Элемент найден на позиции 5

# Задание 10

Задание 10 представляет собой задачу на создание специализированной коллекции – кольцевого списка для имитации людей, стоящих в одном круге и предоставляющей возможность для человека «выйти» из круга.

## Анализ задачи

Условие задачи приведено на рисунке ‎11.1. Задание предполагает создание специальной структуры данных, а затем удаление элементов по очереди через равный промежуток.

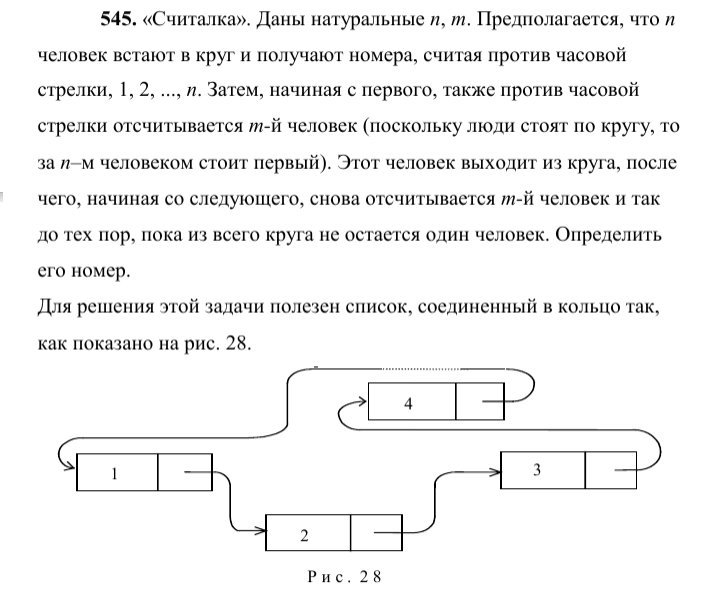


Рисунок ‎11.1. Условия задания 10

При создании подобной структуры нужно учитывать количество оставшихся после удаления узлов. Как только оно достигнет единицы, то решение задачи найдено.

### Входные и выходные данные

Входные данные: два натуральных числа: одно отвечает за количество человек, другое за номер выходящего из круга.

Выходные данные: натуральное число – номер человека, не вышедшего из круга.

## Разработка алгоритма

Блок-схема алгоритма удаления элемента из кольцевого списка приведена в приложении М. Так как список кольцевой, алгоритм удаления использует этот факт: если входной индекс больше количества узлов в списке, то значение индекса приравнивается модулю от него.

По условию задачи, если размер списка равен единице, то выводится решение задания, но в алгоритме предусмотрена проверка на удаление единственного значения – в таком случае список оказывается пустым и дальнейшее удаление или обращение к (несуществующему) элементу вызовет ошибку.

В случае, если элементов больше единицы, то от ссылки на первый элемент списка отсчитывается требуемое количество элементов, для того чтобы был выбран узел для удаления. Узел затем удаляется, ссылки предыдущего и следующего элементов устанавливаются друг на друга. Если удален головной узел, то новым головным элементов становится тот, который следует сразу же после старого.

## Реализация программной системы

Программа реализована на языке C#, интерфейс выбран консольный вследствие простоты ввода исходных данных. Пример взаимодействия пользователя с программой представлен на рисунке ‎11.2.

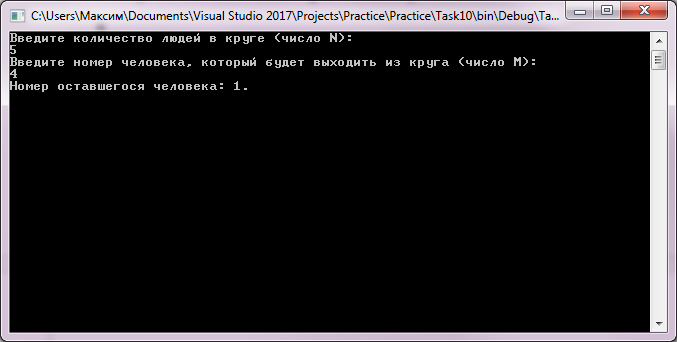


Рисунок ‎11.2. Пример взаимодействия пользователя с заданием 10

Для достижения требуемого результата был создан класс LoopedList, который является двусвязным списком, концевой элемент которого имеет ссылку на головной элемент.

Класс LoopedList можно сконструировать, задав количество людей в группе. Таким образом, список будет заполнен натуральными числами от 1 до количества людей. Внутри класса LoopedList объявлен класс Node, представляющий собой отдельный узел списка. Каждый экземпляр этого класса содержит в себе информационное поле (для хранения номера человека), ссылки на предыдущий и следующий элементы. Диаграмму классов можно наблюдать на рисунке ‎11.3.

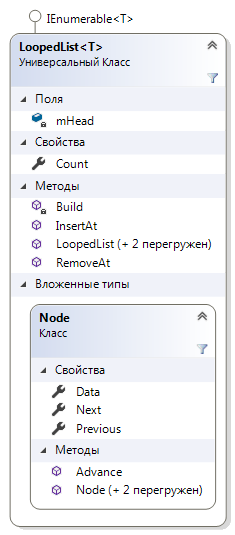


Рисунок ‎11.3. Диаграмма классов задания 10

С помощью последовательных вызовов метода RemoveAt имитируется «выход человека из круга». Когда количество элементов достигает единицы, приложение выводит его порядковый номер.

## Тестирование и отладка приложения

После разработки решения задания было проведено тестирование приложения. Входными данными служат 2 натуральных числа, так что следует проверять вводимые числа на натуральность, а также на соответствие числовому формату.

Критерии входных данных:

* + 1. Один из параметров – не число
    2. Оба параметра – числа, не являющиеся натуральными
    3. Один из параметров – не натуральное число
    4. Оба параметра равны единице
    5. Оба параметра – натуральные числа

Тесты для задания приведены в приложении Ц.

# Задание 11

Задание 11 взято из [1]. Оно является задачей на кодирование информации и имеет определенную практическую ценность. При передаче информации часто возникают помехи, поэтому при ошибках очень ценно иметь возможность восстановиться.

## Анализ задачи

Как видно из условия задачи, представленного на рисунке ‎12.1, задание похоже на передачу информации по каналу связи. При связи может произойти изменение сигнала на противоположный, поэтому ценой расширения сообщения оно утраивается.

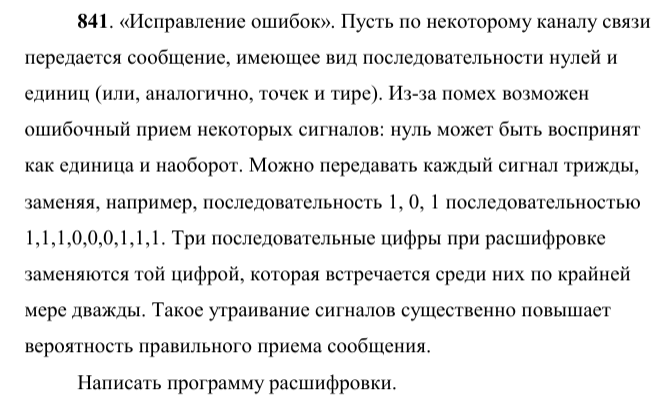


Рисунок ‎12.1. Условия задания 11

Для моделирования канала связи можно представить его как булевый массив, помехи можно моделировать с применением псевдослучайных чисел.

### Входные и выходные данные

Входные данные: массив с утроенными двоичными данными (то есть с возможными помехами).

Выходные данные: массив с наиболее вероятной исходной информации.

## Разработка алгоритма

Для решения задачи был разработан алгоритм для создания наиболее вероятной информации из утроенной информации. Его блок-схема находится в приложении Н. Алгоритм довольно прост. Он действует по принципу подсчета единиц в каждой тройке битов. Если количество единиц больше полутора, то значит на выходе получается единица, иначе получается ноль.

## Реализация программной системы

В качестве языка программирования был выбран C#. Интерфейс представлен консолью с дополнительной подсветкой битов: в выходной информации зеленым цветом подсвечиваются корректно восстановленные биты, красным – ошибки. Внешний вид представлен на рисунке ‎12.2.

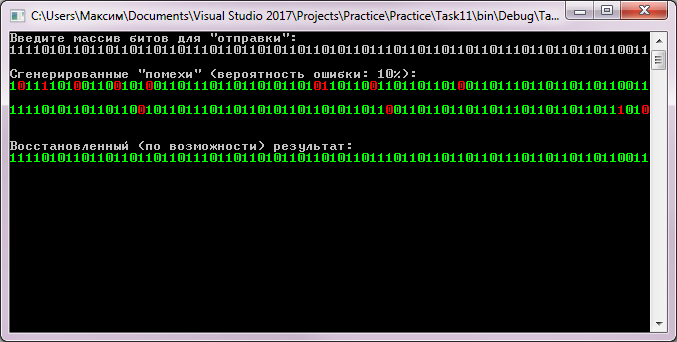


Рисунок ‎12.2. Внешний вид задания 11

Как видно из этого примера, при вероятности ошибки в 10% информация может быть доставлена невредимой. Программа довольно проста, поэтому основную работу делает функция Decode статического класса Decoder. Именно в ней реализован вышеописанный алгоритм.

## Тестирование и отладка приложения

После проектирования и разработки приложения оно было протестировано и отлажено. Так как входные данные – двоичная строка, для которой автоматически генерируются помехи, то пользователь ответственен лишь за ввод исходной строки.

Типы исходных данных:

* + 1. Пустая строка
    2. Строка, не являющаяся представлением двоичной информации
    3. Строка, начинающаяся как двоичное представление, но имеющая внутри себя символы, отличные от «0» и «1».
    4. Строка, являющаяся корректной

Тесты для задания 11 представлены в приложении Ш.

# Задание 12

Различные алгоритмы сортировки имеют различные требования к памяти и времени исполнения. «Быстрым» алгоритмом сортировки называется тот алгоритм, который имеет временную сложность и выше. Примерами таких алгоритмов служат быстрая сортировка, сортировка Шелла, пирамидальная сортировка и т.д.

## Анализ задачи

Задание 12 предполагает реализацию и сравнение двух алгоритмов сортировки: сортировки с помощью двоичного дерева и сортировки подсчетом. Полное условие представлено на рисунке ‎13.1.

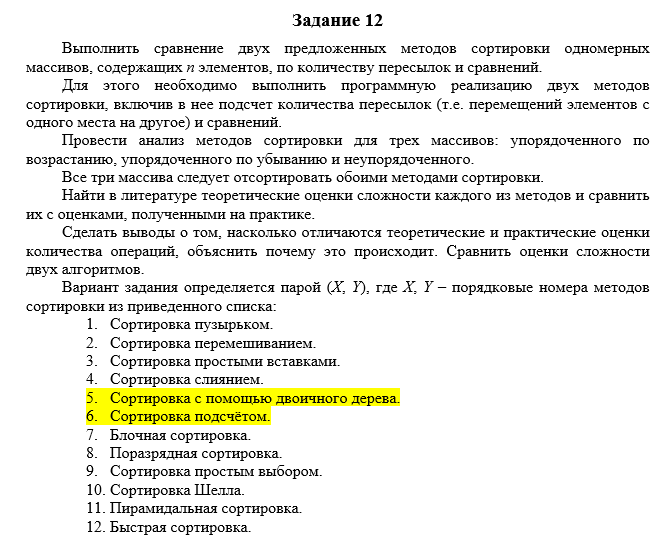


Рисунок ‎13.1. Условие задания 12 (варианты 5 и 6)

Сортировка с помощью двоичного дерева предполагает реализацию упорядоченной структуры данных – основная работа алгоритма происходит при построении экземпляра такой структуры. При оптимальных входных данных временная оценка данного алгоритма близка к , но при неблагоприятном входе временная сложность деградирует до . Примером такого случая можно назвать сортировку последовательности убывающих натуральных чисел в порядке возрастания. Потребление памяти является линейным, то есть растет пропорционально количеству сортируемых элементов.

Сортировка подсчетом является довольно специфичным алгоритмом, его применение возможно только к примитивным цифровым данным. Алгоритм показывает временную оценку в , где n – количество элементов, k – количество чисел в диапазоне. Несмотря на кажущуюся быстроту, алгоритм крайне требователен к памяти, причем потребление памяти зависит не от количества элементов, а от диапазона входных данных. Например, при сортировке двух 32-битных целых чисел, равняющимся минимальному и максимальному значению для этого типа данных, потребление памяти составит около 4 ГиБ, что недопустимо практически в любом случае.

Алгоритм может быть практичен для применения на длинных последовательностях с малым диапазоном значений, например, на массивах байтовых данных. Более того, при таком применении даже не самый оптимальный код будет выдавать лучшие результаты по сравнению с другими алгоритмами сортировки.

### Входные и выходные данные

Входные данные: три типа диапазонов для сортировки двумя заданными методами: возрастающий, убывающий, неупорядоченный.

Выходные данные: статистические данные о количестве обращений к памяти обоих алгоритмов.

## Разработка алгоритма

Так как данные алгоритмы были разработаны, то их остается лишь реализовать. Блок-схема этих алгоритмов сортировки приведена в приложении П. В отличие от блок-схемы, в реализацию должны быть добавлены счетчики пересылок значений и счетчики сравнений.

## Реализация программной системы

Для реализации программы выбран язык программирования C#. Интерфейс приложения – консоль. Пользователь вводит количество элементов в каждом из вышеописанных диапазонов, сами диапазоны генерируются последовательным заполнением их целочисленными элементами В случае неупорядоченного диапазона дополнительно применяется функция случайной расстановки.

После ввода длины массивы сортируются, на экран выводятся результаты замеров данных о доступе к памяти. Пример такого взаимодействия представлен на рисунке ‎13.2.

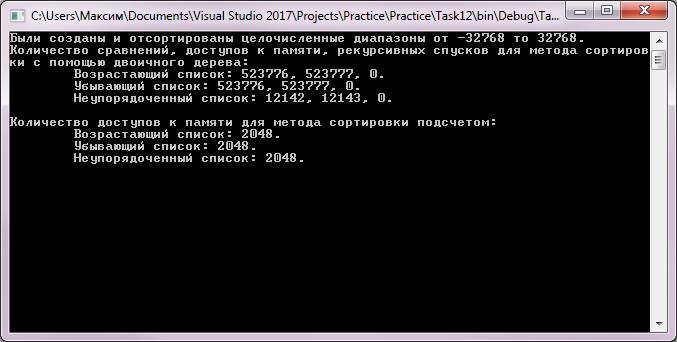


Рисунок ‎13.2. Пример взаимодействия пользователя с заданием 12

Практические результаты во многом зависят от введенных данных. Это отчетливо видно на алгоритме сортировки с помощью двоичного дерева. Сортировки неупорядоченного списка и возрастающего/убывающего списков различаются по показателям эффективности более чем в 40 раз, хотя значения всех трех диапазонов были одинаковы, различен лишь порядок.

## Тестирование и отладка приложения

# Заключение

Результатом выполнения исследования и разработки является программная система, в рамках которой решены все 12 задач, заданных в варианте.

Функции, которые были реализованы:

1. Решение указанных задач
2. Предоставление пользователю интерфейса для ввода исходных данных
3. Реализация вывода результатов решения поставленных задач

В ходе реализации системы были достигнуты цели и решены задачи, описанные во введении.

В качестве дальнейшего развития системы возможно улучшение производительности, например, задания 4 – подсчета факториала. Также возможно добавление дополнительного функционала, выходящего за рамки поставленных в заданиях учебной практики условий, улучшение интерфейса пользователя.

# Библиографический список

1. Абрамов С.А., Гнездилова Г.Г., Капустина Е.Н., Селюн М.И. Задачи по программированию. М.: Наука, 1988.
2. Мацяшек Л.А. Практическая программная инженерия на основе учебного примера: пер. с англ. – М.: БИНОМ. Лаборатория знаний, 2011.
3. Плаксин М.А. Тестирование и отладка программ – для профессионалов будущих и настоящих. – М.: БИНОМ. Лаборатория знаний, 2007.
4. Терехов А.Н. Технология программирования: учебное пособие. / А.Н. Терехов: – 2-е изд. – М.: БИНОМ. Лаборатория знаний, 2012.
5. Троелсен Э. Язык программирования C# 2010 и платформа .NET 4: пер. с англ. – М.: Издательский дом "Вильямс", 2012.

Приложение А. Исходный код программы

Исходный код программной системы, созданной в результате работы над практикой, находится на сервисе GitHub по следующему адресу: <https://github.com/SitadziMado/practice>.

Приложение Б. Блок-схема задания 1

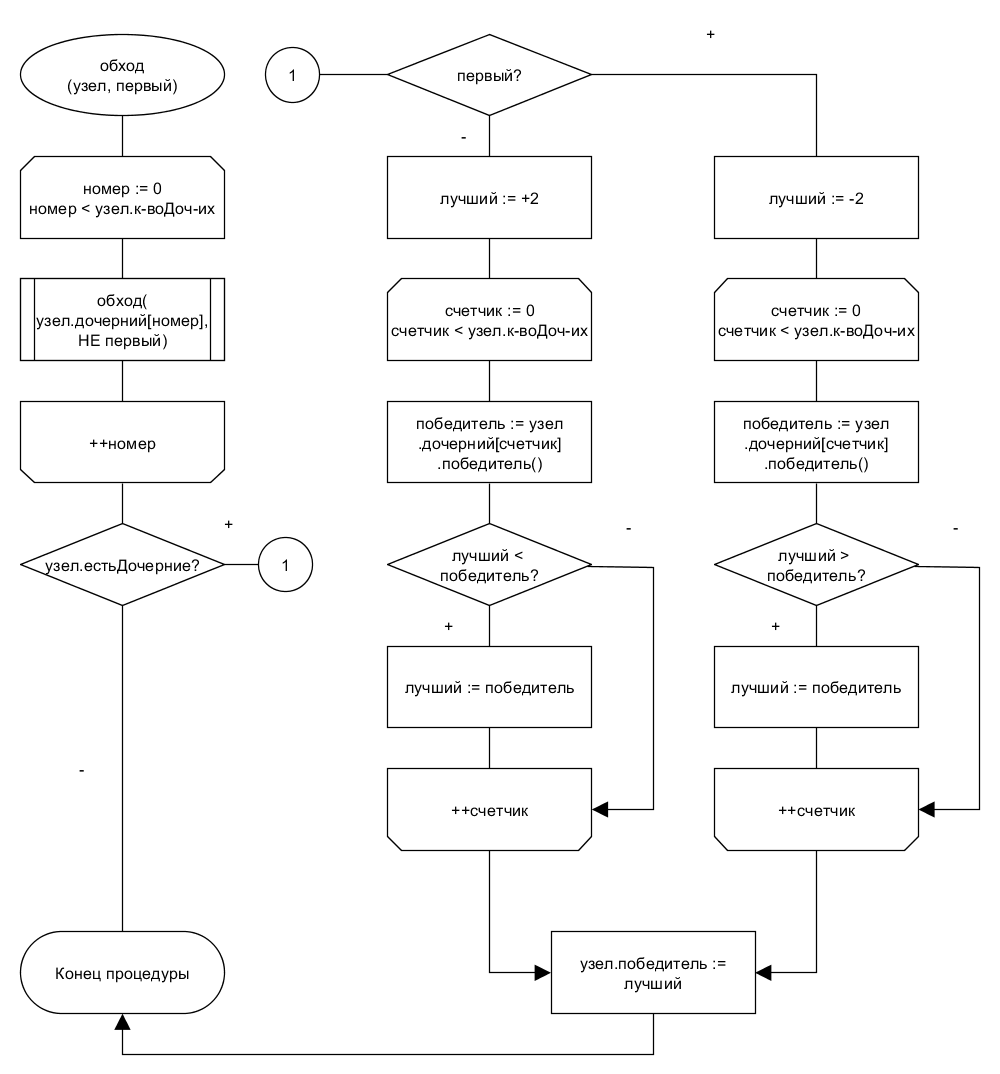


Рисунок 1. Блок-схема задания 1

Приложение В. Блок-схема задания 2

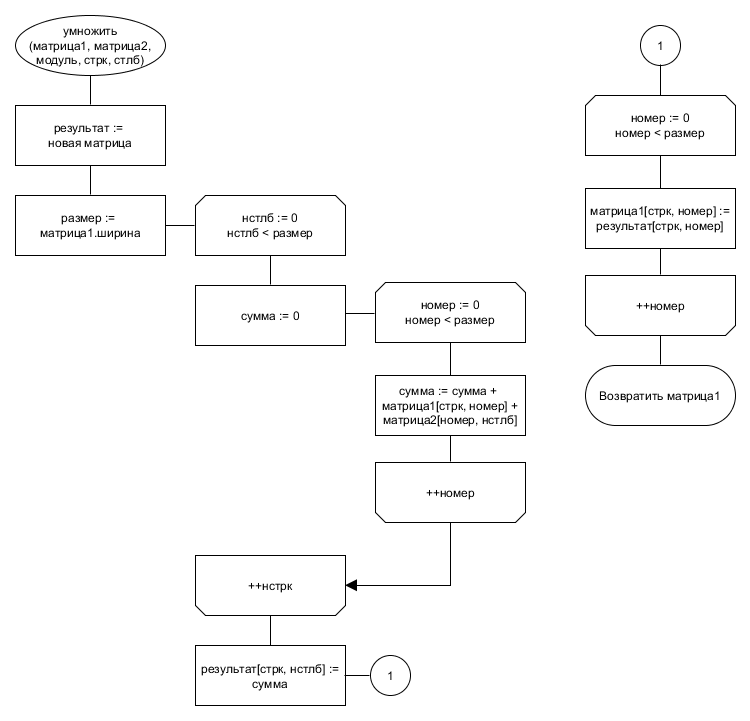


Рисунок 2. Блок-схема задания 2

Приложение Г. Блок-схема задания 3

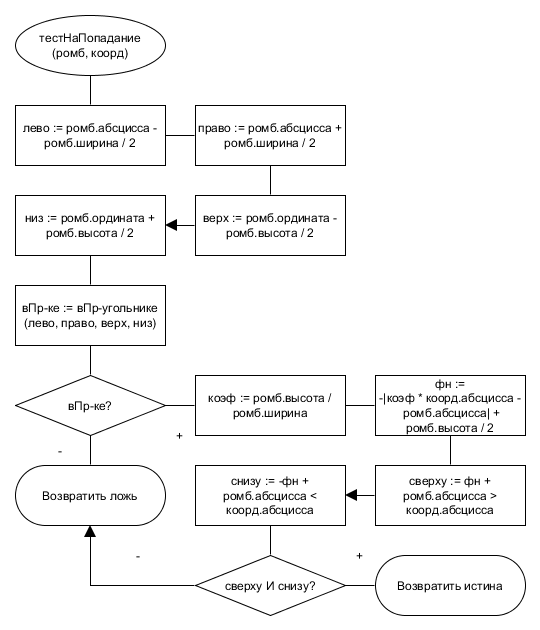


Рисунок 3. Блок-схема задания 3

Приложение Д. Блок-схема задания 4

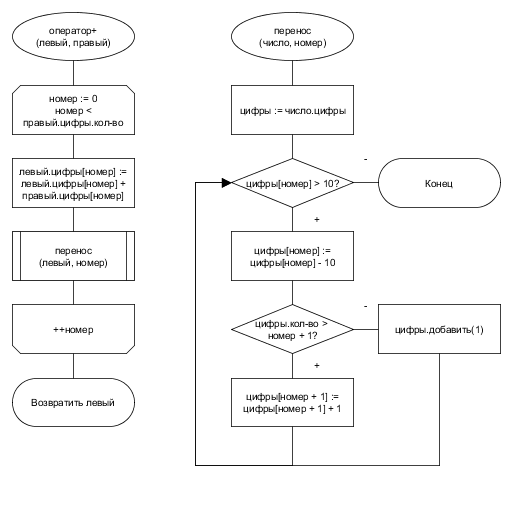


Рисунок 4. Блок-схема задания 4, часть 1

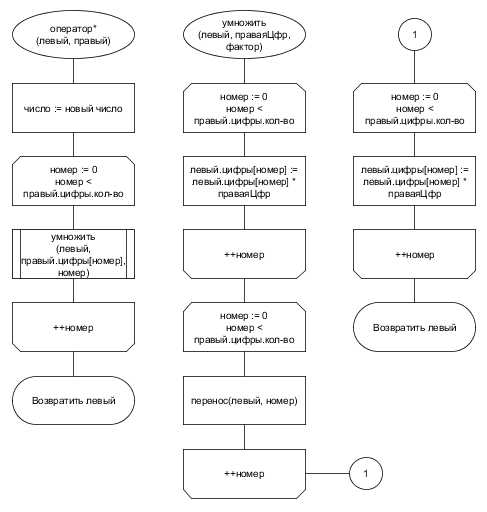


Рисунок 5. Блок-схема задания 4, часть 2

Приложение Е. Блок-схема задания 5

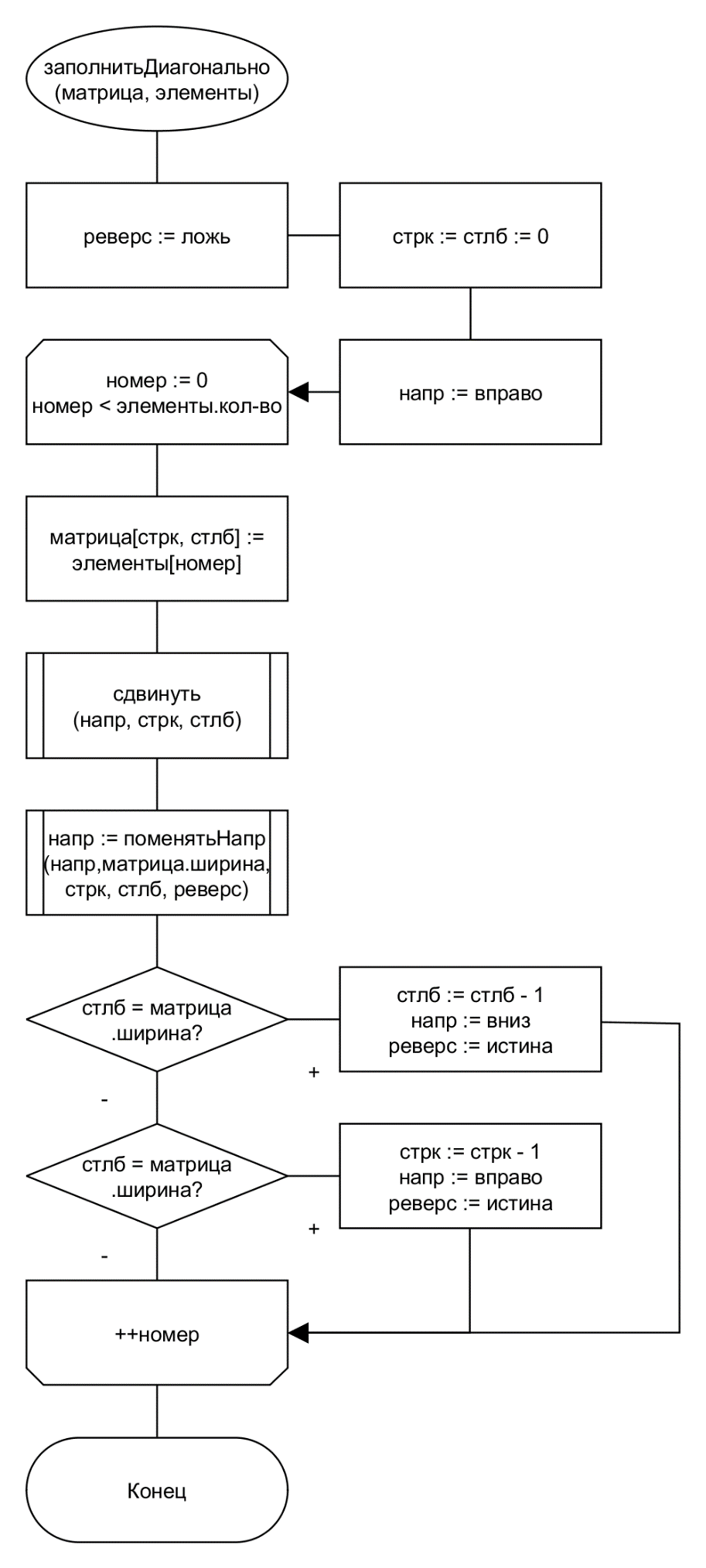


Рисунок 6. Блок-схема основного алгоритма задания 5

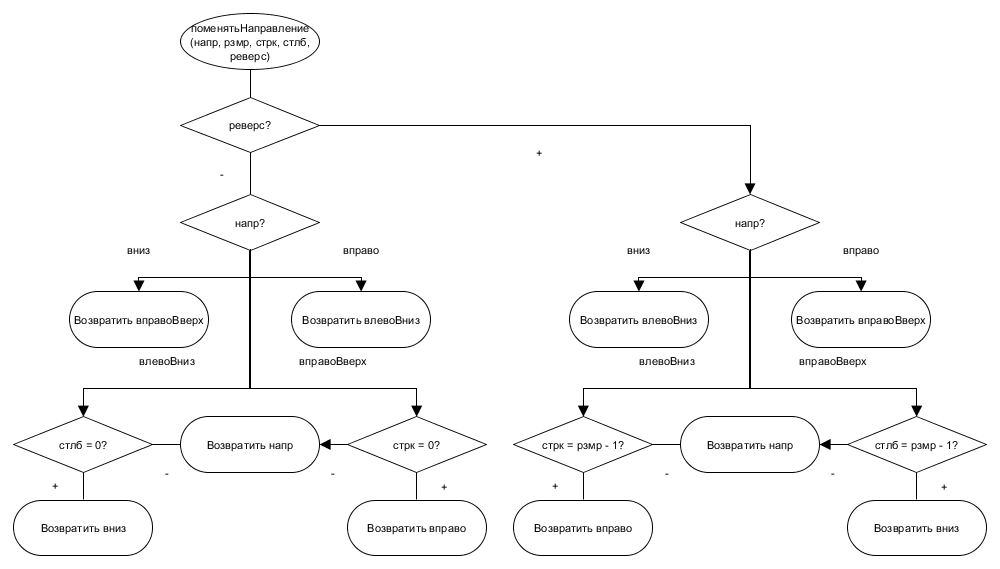


Рисунок 7. Блок-схема алгоритма смены направления

Приложение Ж. Блок-схема задания 6

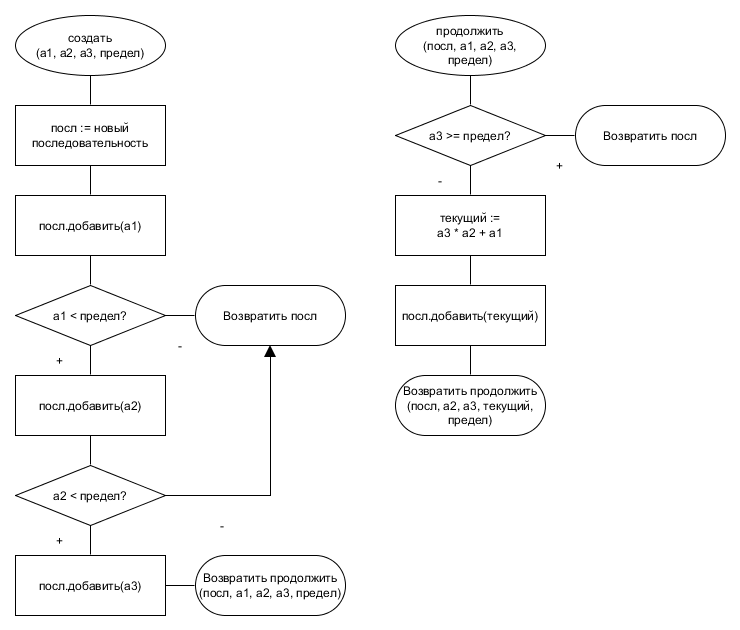


Рисунок 8. Блок-схема задания 6

Приложение И. Блок-схема задания 7

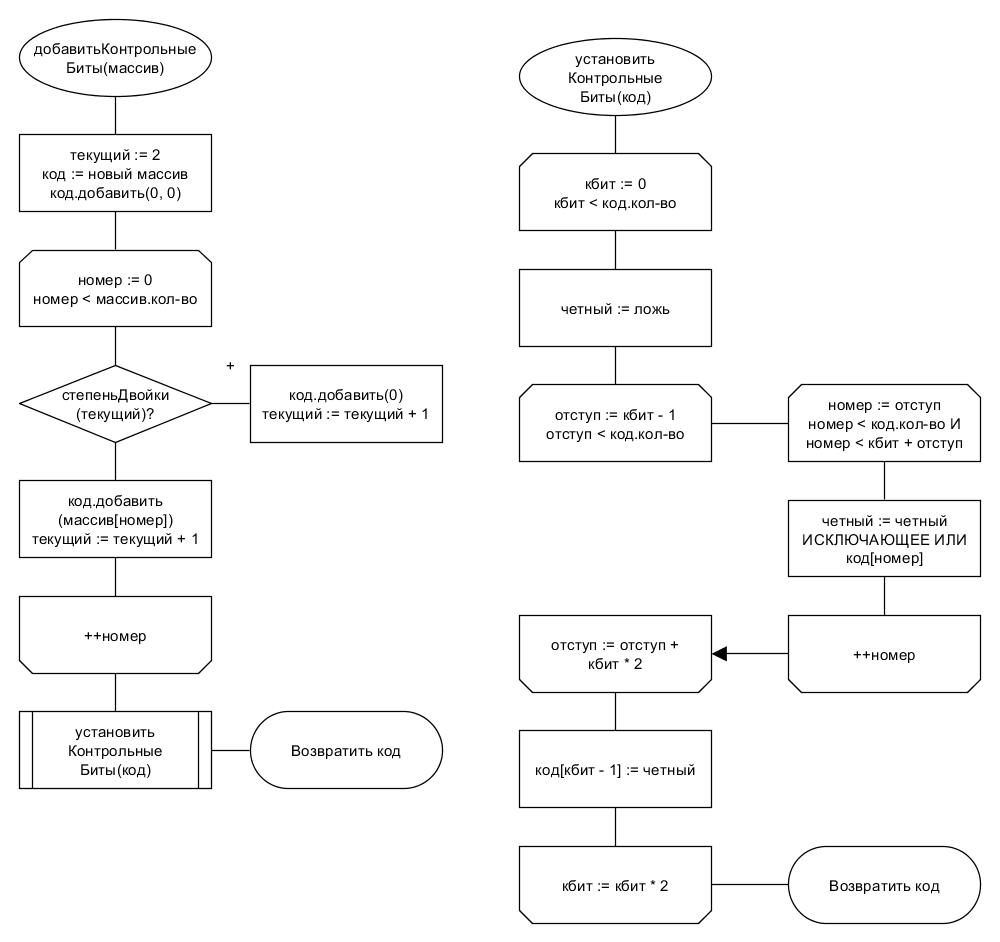


Рисунок 9. Блок-схема задания 7

Приложение К. Блок-схема задания 8

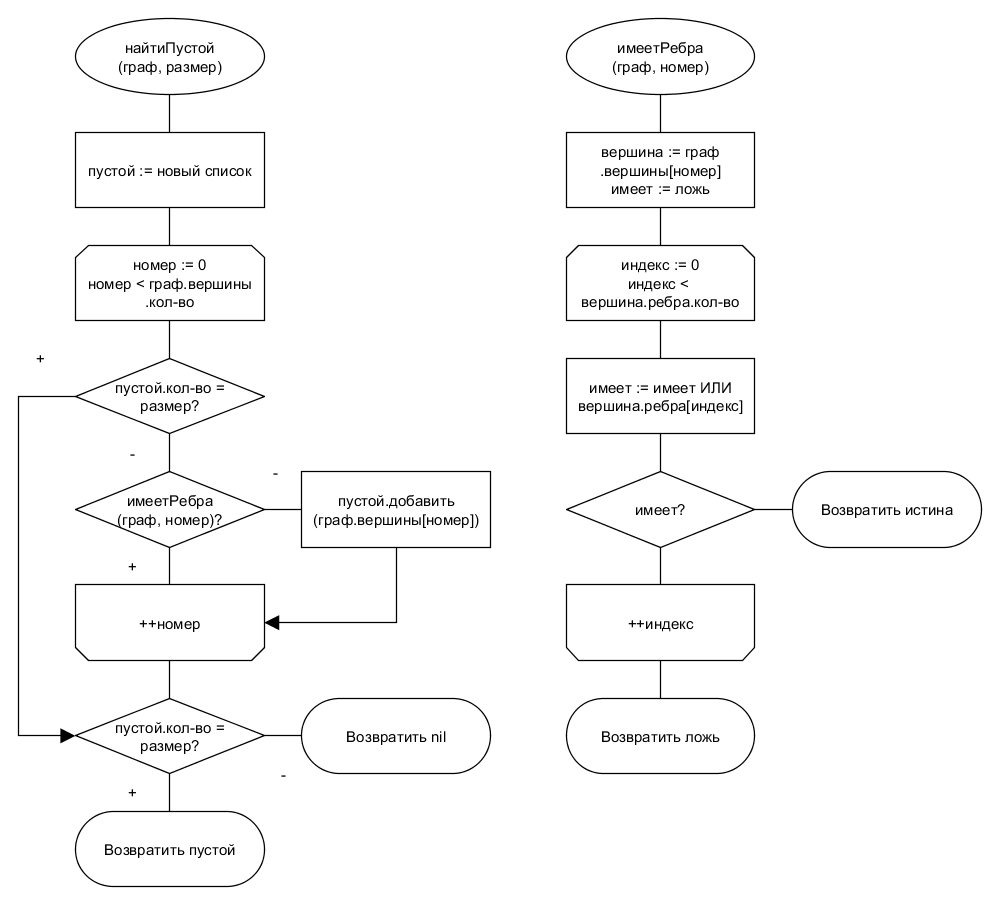


Рисунок 10. Блок-схема задания 8

Приложение Л. Блок-схема задания 9

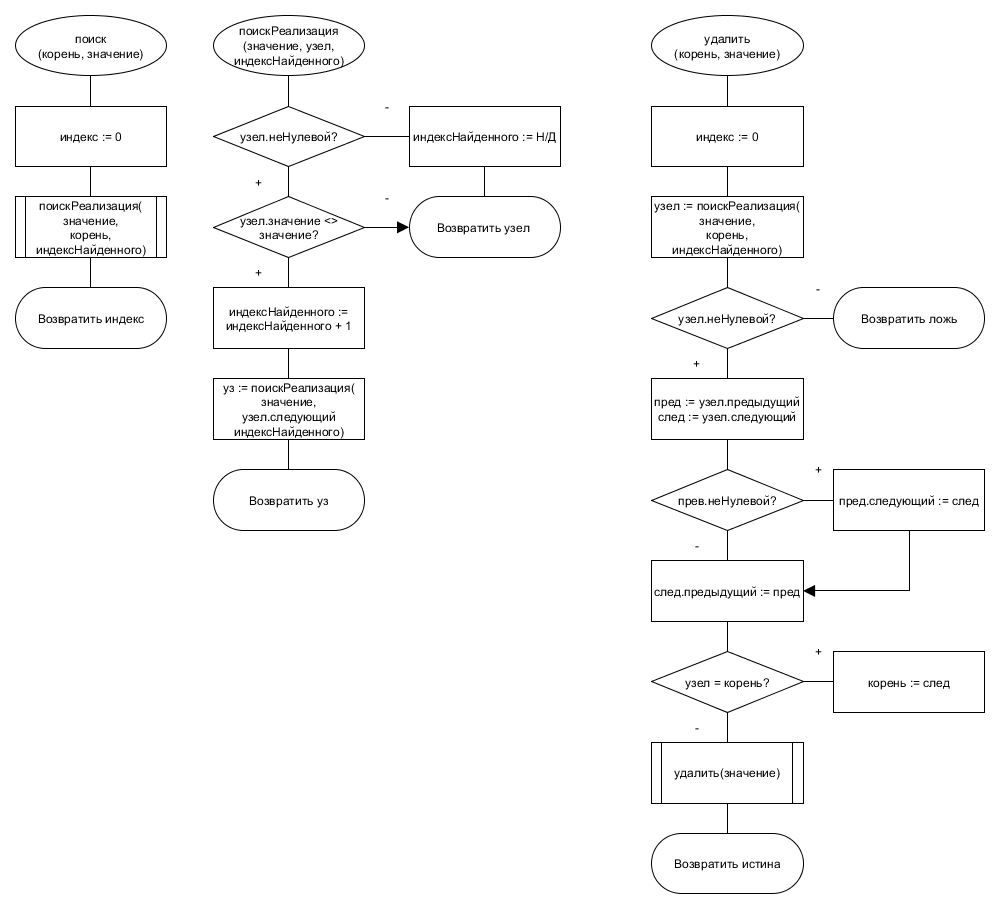


Рисунок 11. Блок-схема задания 9

Приложение М. Блок-схема задания 10

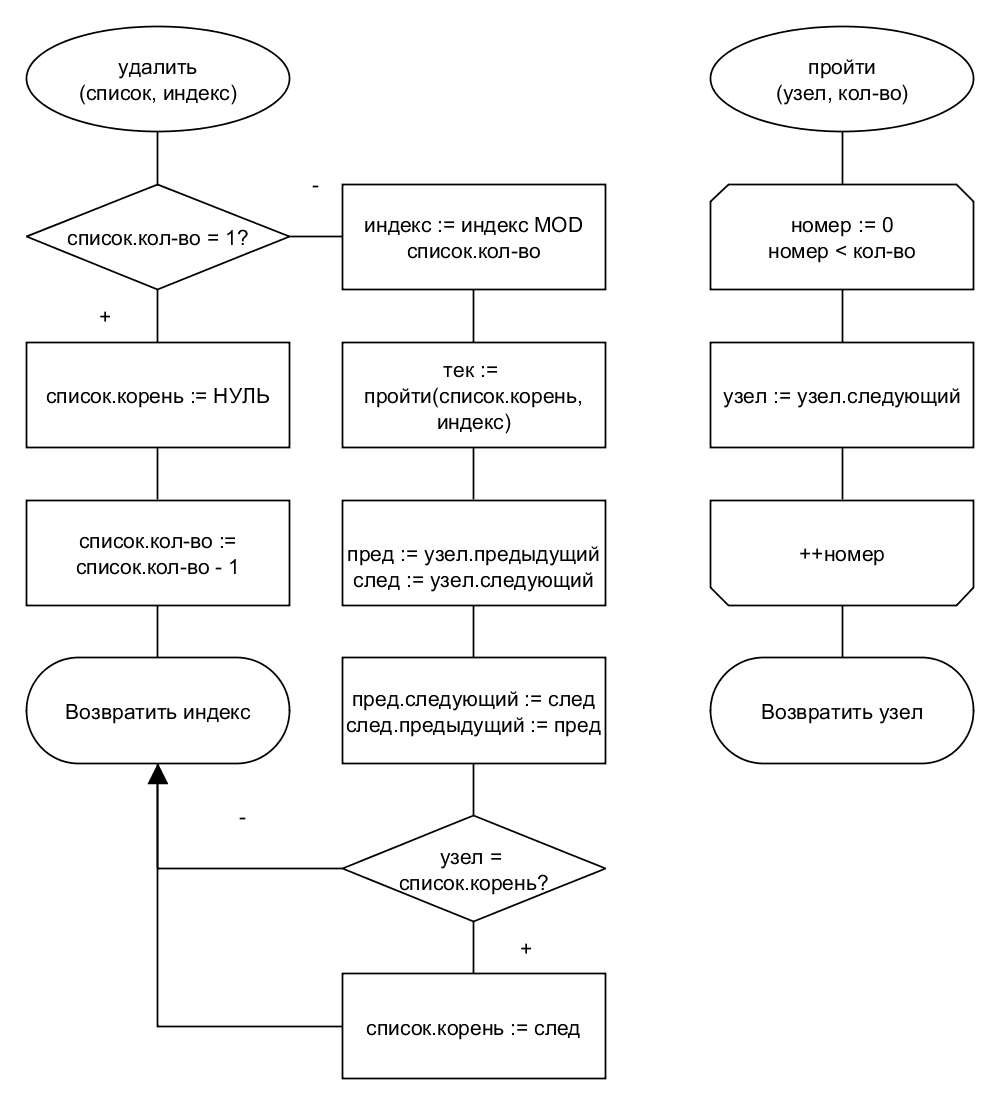


Рисунок 12. Блок-схема задания 10

Приложение Н. Блок-схема задания 11

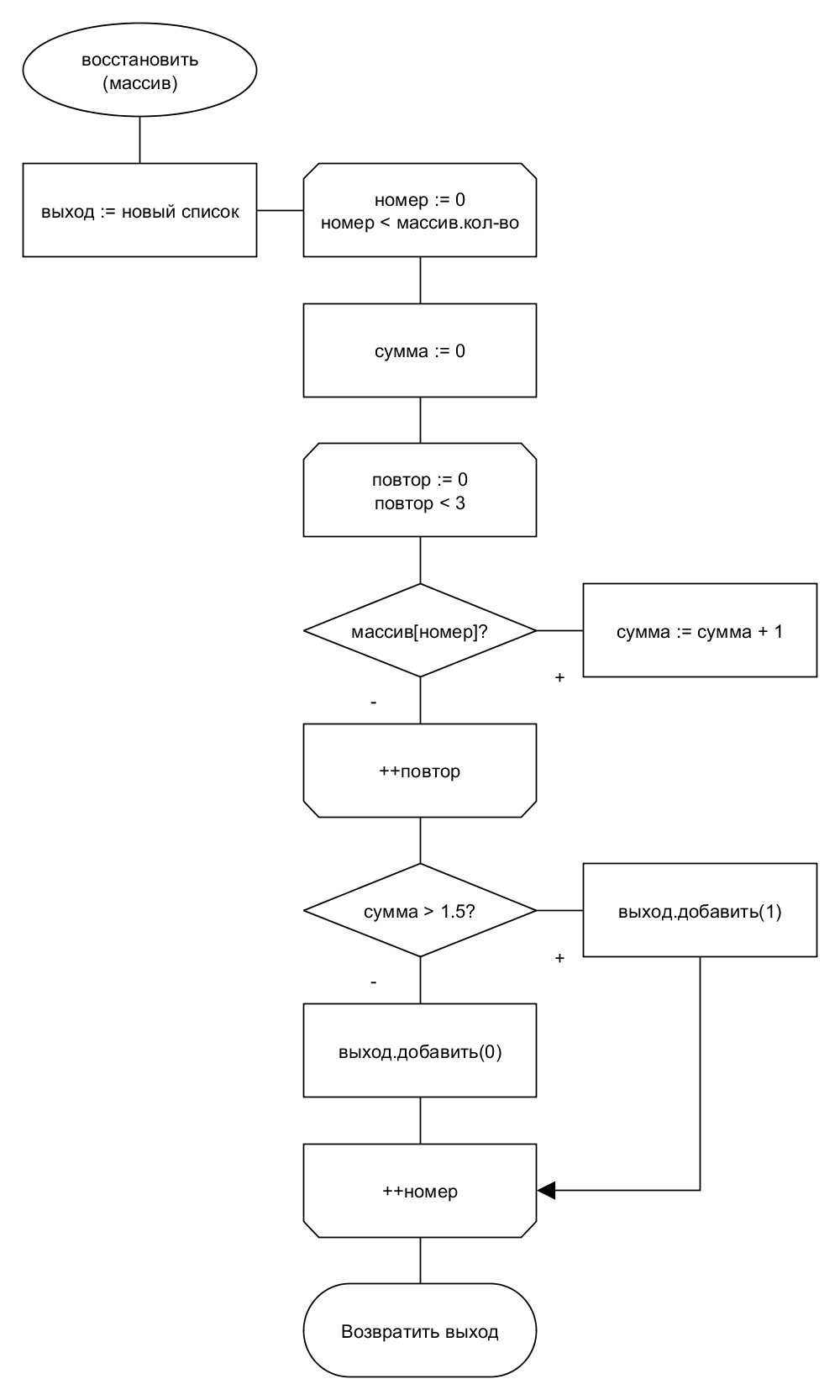


Рисунок ‎13. Блок-схема задания 11

Приложение П. Блок-схема задания 12

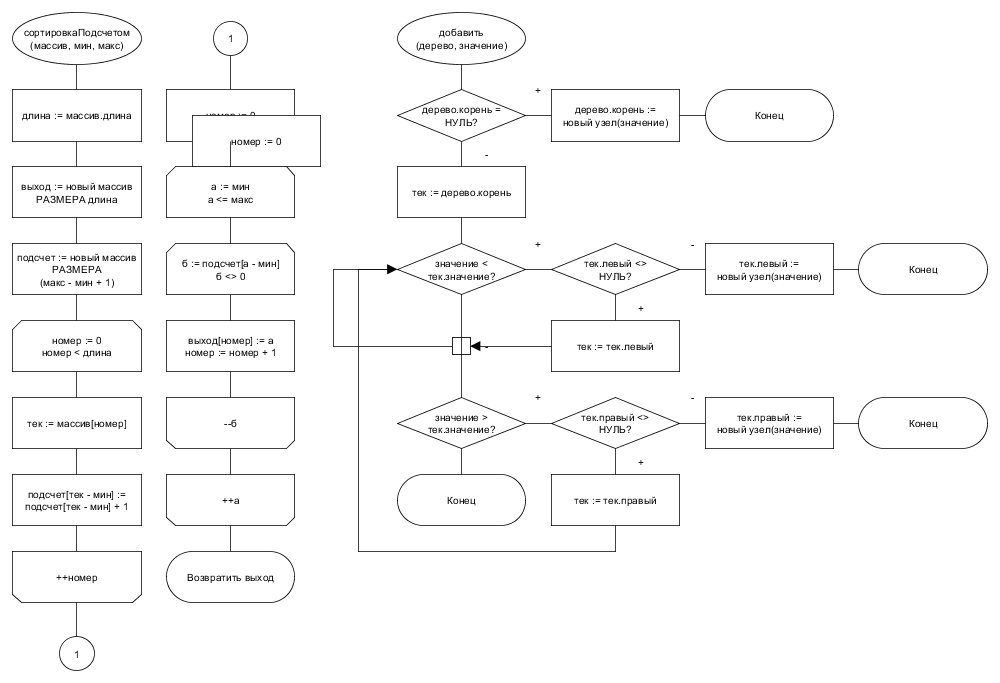


Рисунок 14. Блок-схема алгоритмов сортировки

Приложение Р. Тесты для задания 3

Таблица 1. Тесты для задания 3

|  |  |  |  |
| --- | --- | --- | --- |
| Номер теста | Входные данные | Ожидаемый выход | Реальный выход |
| Т1 | (0.0; 0.0) | Принадлежит | + |
| Т2 | (0.0; -0.5) | Принадлежит | + |
| Т3 | (0.2; 0.2) | Принадлежит | + |
| Т4 | (-0.5; 0.0) | Принадлежит | + |
| Т5 | (2.0; 0.0) | Не принадлежит | + |
| Т6 | (0.0; 10.0) | Не принадлежит | + |

Приложение С. Тесты для задания 4

Таблица 2. Тесты для задания 4

|  |  |  |  |
| --- | --- | --- | --- |
| Номер теста | Входные данные | Ожидаемый выход | Реальный выход |
| Т1 | Скрскр | Ошибка, ожидалось число | + |
| Т2 | -256 | Ошибка, ожидалось неотрицательное число | + |
| Т3 | 0 | 0 | + |
| Т4 | 100 | 93326215443944152681699238856266700490  71596826438162146859296389521759999322  99156089414639761565182862536979208272  23758251185210916864000000000000000000  000000 | + |
| Т5 | 25 | 15511210043330985984000000 | + |
| Т6 | 10 | 3628800 | + |

Приложение Т. Тесты для задания 5

Таблица 3. Тесты для задания 5

|  |  |  |  |
| --- | --- | --- | --- |
| Номер теста | Входные данные | Ожидаемый выход | Реальный выход |
| Т1 | Скрскр | Ошибка, ожидалось число | + |
| Т2 | -256 | Ошибка, ожидалось неотрицательное число | + |
| Т3 | 0 | 0 | + |
| Т4 | 100 | 93326215443944152681699238856266700490  71596826438162146859296389521759999322  99156089414639761565182862536979208272  23758251185210916864000000000000000000  000000 | + |
| Т5 | 25 | 15511210043330985984000000 | + |
| Т6 | 10 | 3628800 | + |

Приложение У. Тесты для задания 6

Таблица 4. Тесты для задания 6

|  |  |  |  |
| --- | --- | --- | --- |
| Номер теста | Входные данные | Ожидаемый выход | Реальный выход |
| Т1 | Скрскр 5 6 8 | Ошибка, ожидалось число | + |
| Т2 | 0 0 0 0 | N = 1, M == a[N]: Истина | + |
| Т3 | 0 0 0 1 | Последовательность никогда не превысит предела | + |
| Т4 | 0 0 0 -88 | N = 1, M == a[N]: Ложь | + |
| Т5 | 4 9 16 2 | N = 1, M == a[N]: Ложь | + |
| Т6 | 4 9 16 6 | N = 2, M == a[N]: Ложь | + |
| Т7 | 4 9 16 13 | N = 3, M == a[N]: Ложь | + |
| Т8 | 4 9 16 148 | N = 4, M == a[N]: Истина | + |
| Т9 | 4 9 16 175 | N = 5, M == a[N]: Ложь | + |

Приложение Ф. Тесты для задания 7

Таблица 5. Тесты для задания 7

|  |  |  |  |
| --- | --- | --- | --- |
| Номер теста | Входные данные | Ожидаемый выход | Реальный выход |
| Т1 |  | Введите что-либо | + |
| Т2 | 01010010одинноль | Запрет на ввод | + |
| Т3 | 0 | 000 | + |
| Т4 | 1 | 111 | + |
| Т5 | 0100010000111101 | 100110000100001011101 | + |

Приложение Х. Тесты для задания 8

Таблица 6. Тесты для задания 8

|  |  |  |  |
| --- | --- | --- | --- |
| Номер теста | Входные данные | Ожидаемый выход | Реальный выход |
| Т1 | -  K = 3 | Ошибка, отсутствуют вершины | + |
| Т2 | 1  1  0  K = 0 | Ошибка, размер пустого подграфа должен быть натуральным числом | + |
| Т3 | 10  11  01  K = 1 | Пустой подграф не найден | + |
| Т4 | 1  1  0  0  K = 2 | Пустой подграф размером 2 найден: вершины 3 и 4 | + |

Приложение Ц. Тесты для задания 10

Таблица 7. Тесты для задания 10

|  |  |  |  |
| --- | --- | --- | --- |
| Номер теста | Входные данные | Ожидаемый выход | Реальный выход |
| Т1 | Скр 2 | Ошибка, требуется натуральное число | + |
| Т2 | -5 0 | Ошибка, число должно быть натуральным | + |
| Т3 | 30 0 | Ошибка, число должно быть натуральным | + |
| Т4 | 1 1 | 1 | + |
| Т5 | 7 3 | 4 | + |
| Т6 | 5 4 | 1 | + |

Приложение Ш. Тесты для задания 11

Таблица 8. Тесты для задания 11

|  |  |  |  |
| --- | --- | --- | --- |
| Номер теста | Входные данные | Ожидаемый выход | Реальный выход |
| Т1 |  | Ошибка, введите непустую строку, состоящую только из `0` и `1` | + |
| Т2 |  | Ошибка, введите непустую строку, состоящую только из `0` и `1` | + |
| Т3 |  | Ошибка, введите непустую строку, состоящую только из `0` и `1` | + |
| Т4 | 0101110010101011 | ОК | + |

Приложение Щ. Тесты для задания 12

Таблица 9. Тесты для задания 12

|  |  |  |  |
| --- | --- | --- | --- |
| Номер теста | Входные данные | Ожидаемый выход | Реальный выход |
| Т1 | -1 | Ошибка | + |
| Т2 | 0 | Ошибка | + |
| Т3 | 1 | ОК | + |
| Т4 | 1024 | ОК | + |

1. Этого можно добиться с помощью сторонних программ, таких, как Spy++ от Microsoft [↑](#footnote-ref-1)