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

Факультет экономики, менеджмента и бизнес-информатики

ОТЧЕТ

ПО УЧЕБНОЙ ПРАКТИКЕ

Выполнил студент группы ПИ-17-2
Вальченко Антон Артемович

_____
(подпись)

Проверил:
Руководитель практики
Старший преподаватель кафедры высшей математики
М. В. Крючков

_____ _____
(оценка) (подпись)

_____
(дата)

Аннотация

В данной работе были решены 12 задач по учебной практике, реализована одна собственная библиотека.

Автор: студент НИУ-ВШЭ Пермь Вальченко Антон.

Кафедра информационных технологий в бизнесе.

Работа содержит 50 страниц формата А4 основного текста, включая 13 глав, в каждой из которых решается отдельная задача.

В основной части содержится 16 рисунков и 20 таблиц.

В работе содержится одно приложение – CD диск с исходными кодами.

В работе содержится один CD диск.

Библиографический список состоит из четырёх источников.

Оглавление

Введение

Цель учебной практики – это применение знаний и умений, полученных при изучении отдельных дисциплин первого курса, а также приобретение и развитие навыков алгоритмического мышления, путём решения и документирования набора задач разной сложности и размера.

Для достижения этой цели необходимо выполнить следующие задачи:

1. Изучить существующие алгоритмы и методы решения представленных задач из различных областей.
2. Провести анализ задач, выбрать наиболее подходящий алгоритм, определить функциональных требования, а также требования к входным и выходным данным.
3. На основе предыдущих двух пунктов описать решение задачи, спроектировать его и реализовать.
4. Провести тестирование решения по критериям полноты чёрного ящика.
5. Закрепить и углубить практические навыки работы с системой контроля версий Git, с помощью использования расширения GitHub Extension для Microsoft Visual Studio, использовать полученный опыт во время тестирования и отладки решения предложенных зачать посредством последовательного исправления ошибок и документирования процесса отладки.

В ходе решения предложенных задач будут приобретены навыки практического применения знаний из области дискретной математики, линейной алгебры и программирования, а также получен опыт применения динамического программирования, реализации сложных математических теорий.

Каждая последующая глава содержит описания решения одной задачи.

1. «Игра с монетами»

Гриша и Дима играют в следующую игру: они разложили однокопеечные монетки в стопки (в разных стопках может быть различное количество монет), а стопки расположили на столе перед собой в ряд слева направо. Затем Гриша и Дима по очереди делают ходы. На каждом ходе один из игроков берет слева несколько стопок, не меньше одной, но и не больше, чем перед этим взял его соперник. Первый игрок своим первым ходом берет не более K стопок. Игра заканчивается, когда стопок не остается.

Требуется написать программу, позволяющую вычислить, какое максимальное число монет может получить первый участник после окончания игры, если второй – тоже старается ходить так, чтобы получить как можно больше монет.

Входной файл INPUT. TXT состоит из одной строки, в которой записаны: число стопок N ($1 \le N \le 180$), за ним идут N чисел, задающих количество монет в стопках слева направо (количество монет в стопке – не менее 1 и не более 20000), а затем число K, ограничивающее количество стопок, которые первый игрок может взять на первом ходе ($1 \le K \le 80$). Все числа в строке разделены пробелом.

В выходной файл OUTPUT.TXT необходимо вывести одно число – максимальное количество монет, которое заведомо может получить первый игрок, как бы ни играл второй.

Максимальное время выполнения программы – 1 секунда, максимальная используемая память – 16 Мбайт.

1.1. Анализ задачи

Для нахождения максимального количества монет, которое заведомо можно получить необходимо вычислить таблицу максимальных выигрышей при каждом наборе действий. Эта задача решается с помощью динамического программирования методом, который описан в книге Томаса Х. Кормена «Алгоритмы: построение и анализ».

Динамическое программирование – это способ решения одной сложной задачи, путём разбиения её на более мелкие, но лёгкие задачи и запоминания результатов их выполнения.

Функциональные требования: программа должна выводить в файл «output.txt» верный ответ на входные данные из файла «input.txt».

Нефункциональные требования: максимальное время выполнения программы – одна секунда. Максимальная используемая память – 16 Мбайт.

Требования к входным данным: все данные – натуральные числа из диапазона от 1 до 20000.

Формат входных данных: N– число стопок, натуральное число формата int от 1 до 180, A[N] – количество монет в N-ой стопке, массив натуральные чисел типа int от 1 до 20000, K – число стопок, которые можно взять в первый ход, натуральное число формата int от 1 до 80.

Формат выходных данных: G – максимальное количество монет, число формата int.

1.2. Разработка алгоритма

Введем обозначения: s[r] − суммарное количество монет в стопках с r-той по n-ю; g[r, m] − максимальное количество монет, которое может получить игрок, если сейчас его ход, при условии, что остались стопки с r-той по n-ю, и он может взять не более m стопок. Элементы массива s[1..n] будут использоваться для вычисления элементов массива g[n, k], поэтому его нужно вычислить и сохранить сразу после ввода данных.

Далее вычисляем элементы массива g. Если игрок может взять все оставшиеся стопки, то он должен сделать это, иначе другой игрок возьмет не менее одной стопки из оставшихся стопок, и выигрыш игрока, чей сейчас ход, не будет максимальным. Следовательно, все числа g[r,m], где m ≥ n − r + 1, определены, в том числе все элементы g[n, m]. А именно, g[r,m]=s[r].

Будем находить элементы g[r, m] в порядке убывания r, при условии m < n − r + 1 . В этой ситуации игрок может взять от одной до m стопок. Пусть он взял i стопок. Тогда другой игрок окажется в ситуации (r + i, i) − его ход, разыгрываются стопки с (r + i) − той по n − ю, и можно взять не более i стопок. Он может взять максимальное количество монет, равное g[r+i, i]. Так как в стопках с r-той по n-ю было s[r] монет, то первый игрок , взяв i стопок, получит s[r] − g[r+i, i] монет. Максимальный выигрыш будет определяться по формуле (1).

$$g[r,m] = \max_{1 \le i \le m} (s[r] - g[r+i,i]) \quad (1)$$

Ответом задачи будет число g[1, k] – максимальное количество монет, которое может получить первый игрок, где k – первое максимально возможное число взятых стопок.

1.3. Реализация

Листинг программы прикреплен к приложению (приложение А), находится в папке «Игра с монетами».

Для ввода и вывода использованы методы класса File из стандартного пространства имён System.IO.

Переменная count хранит общее число разыгрываемых куч, одномерный массив stack хранит информацию о количестве монет в каждой куче, двумерный массив newStack хранит максимальные значения выигрышей.

Функция s вычисляет значение для ячейки выигрыша, функция f возвращает максимальный выигрыш, который находится в указанном положении, функция l вычисляет варианты разыгрывания кучек.

Для решения задачи собственный классы и структуры данных не использовались.

1.4. Тестирование

Результаты тестирования в системе acmp представлены ниже (рис. 1.1). Номер решения в системе: 8390225.

Тест	Результат	Время	Память
1	Accepted	0,03	1781 Кб
2	Accepted	0,062	1781 Кб
3	Accepted	0,062	1781 Кб
4	Accepted	0,062	1781 Кб
5	Accepted	0,124	1781 Кб
6	Accepted	0,062	1781 Кб
7	Accepted	0,092	1781 Кб
8	Accepted	0,062	1781 Кб
9	Accepted	0,092	1777 Кб
10	Accepted	0,156	1841 Кб

Рисунок 1.1 Результаты тестирования

2. «Игра Jammed»

Всем известна игра «Пятнашки», где надо выстроить изначально неупорядоченную последовательность чисел, перемещая фишки с нанесёнными числами от 1 до 15 в квадрате 4×4. На основе данной игры была разработана другая – поле в ней лишь 4×2 клетки, на поле 7 фишек, но на фишках изображены буквы английского алфавита и арабские цифры (на каждой фишке – один символ, но на разных фишках могут быть одинаковые символы). Цель игры прежняя – упорядочить в соответствии с образцом стартовую расстановку фишек за минимальное количество ходов.

Свободная клетка обозначается специальным символом «#» и используется для перемещения фишек по полю. Перемещать фишки на свободную клетку разрешается из соседних клеток, имеющих общую грань со свободной. Например (рис. 2.1), более правый символ «0» можно переместить вниз на свободную клетку, тогда «0» будет в нижней клетке, а пустой станет верхняя клетка, либо в свободную клетку переместить букву «C» или цифру «2».

M	0	0	A
8	C	#	2

Рисунок 2.1 Пример игровой решетки

Входной файл INPUT.TXT содержит четыре строки: две первые строки содержат стартовую комбинацию символов, следующие две - образец. Каждая строка содержит 4 символа (английский алфавит и арабские цифры), пустая клетка обозначается символом «#» (решетка).

В выходной файл OUTPUT.TXT выведите минимальное количество перемещений, необходимых для получения искомой комбинации. Если нужную комбинацию получить нельзя, выведите число -1.

2.1. Анализ задачи

Чтобы найти минимальное количество ходов для упорядочивания в соответствии с образцом стартовой расстановки фишек, необходимо проверять возможные состояния, в которые может перейти исходная расстановка, пока не будет получен требуемый результат. Эта задача решается с помощью обхода графа в ширину, который описан в пятой главе книги Левитина А. В. «Алгоритмы. Введение в разработку и анализ».

Функциональные требования: программа должна выводить в файл «output.txt» верный ответ на входные данные из файла «input.txt».

Нефункциональные требования: максимальное время выполнения программы – одна секунда. Максимальная используемая память – 16 Мбайт.

Требования к входным данным: все данные – символы.

Формат входных данных: начальное состояние – двумерный массив char длиной 8, конечное состояние – двумерный массив char длиной 8.

Формат выходных данных: минимальное количество ходов – целое число формата int.

2.2. Разработка алгоритма

Для реализации поиска по графу, каждое состояние будет приниматься за вершину, а возможность перехода из одного состояния в другое будет приниматься за ориентированное ребро графа.

Для вычисления следующего возможного состояния нужно рассмотреть все возможные сдвиги решётки в состоянии и, если они ещё не были рассмотрены, то пометить их соответствующим образом и занести в очередь обработки.

Блок-схема, демонстрирующая работу алгоритма представлена ниже (рис. 2.2).

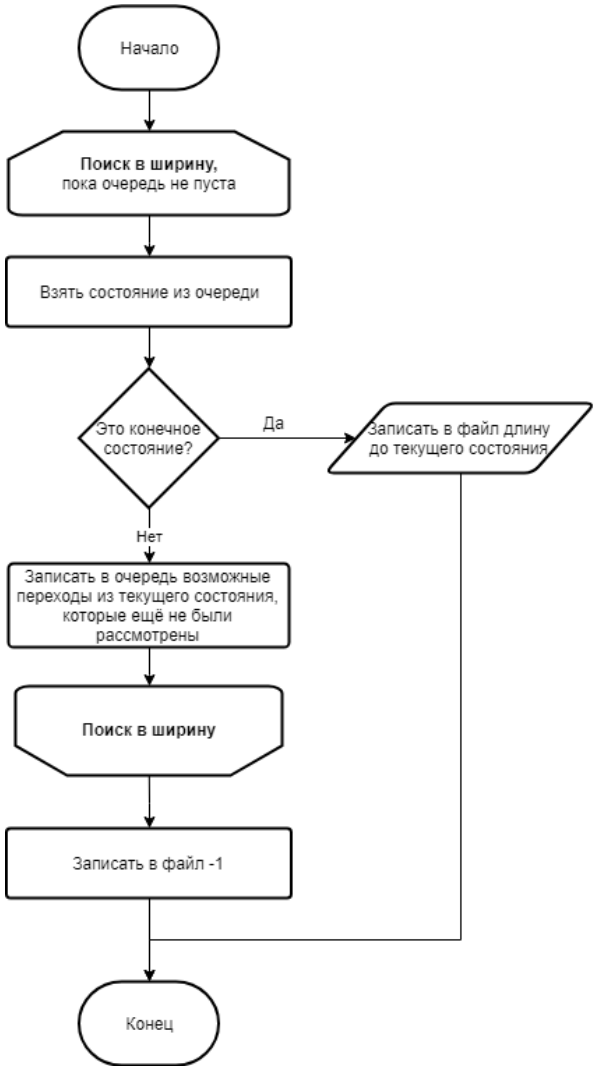

Рисунок 2.2 Поиск в ширину

2.3. Реализация

Листинг программы прикреплен к приложению (приложение А), находится в папке «Игра Jammed».

Для ввода и вывода использованы методы класса File из стандартного пространства имён System.IO.

Переменная start хранит исходное состояние, переменная finish хранит искомое состояние, переменная stateList хранит обработанные состояния и расстояние, которое необходимо пройти, чтобы попасть из исходного состояния в текущее, переменная queue используется как очередь для обработки состояний, переменная cur хранит текущее состояние, с которым и происходит действие.

Функция Initialize инициализирует все переменные пустыми значениями, функция Shift выполняет сдвиг решетки на заданное количество клеток, метод Swap меняет местами два передаваемых в неё значения, метод ReadStart получает из входной строки начальное состояние, метод ReadFinish получает из входной строки конечное состояние, функция ToString преобразует двумерный массив символов в строку.

2.4. Тестирование

Результаты тестирования в системе acmp представлены ниже (рис. 2.3). Номер решения: 8397993.

Тест	Результат	Время	Память
1	Accepted	0,124	5,3 Мб
2	Accepted	0,406	7 Мб
3	Accepted	0,062	3869 Кб
4	Accepted	0,062	2909 Кб
5	Accepted	0,062	2909 Кб
6	Accepted	0,03	2973 Кб
7	Accepted	0,062	2905 Кб
8	Accepted	0,062	2909 Кб
9	Accepted	0,03	2905 Кб
10	Accepted	0,342	7 Мб
11	Accepted	0,186	7 Мб
12	Accepted	0,5	7,7 Мб
13	Accepted	0,124	6,7 Мб
14	Accepted	0,218	6,7 Мб
15	Accepted	0,092	5085 Кб
16	Accepted	0,124	5,5 Мб
17	Accepted	0,062	5085 Кб
18	Accepted	0,03	3737 Кб
19	Accepted	0,092	6,7 Мб
20	Accepted	0,124	4889 Кб
21	Accepted	0,03	2909 Кб
22	Accepted	0,062	3421 Кб
23	Accepted	0,03	3545 Кб
24	Accepted	0,468	7,7 Мб
25	Accepted	0,186	6,7 Мб
26	Accepted	0,53	7,7 Мб
27	Accepted	0,25	7 Мб
28	Accepted	0,406	7,7 Мб
29	Accepted	0,062	3869 Кб
30	Accepted	0,436	7,7 Мб
31	Accepted	0,5	7,7 Мб
32	Accepted	0,468	7,7 Мб

Рисунок 2.3 Результаты тестирования

3. «Вычисление заданной функции»

Дано действительное число a. Для функции f(x) (рис. 3.1) вычислить f(a).

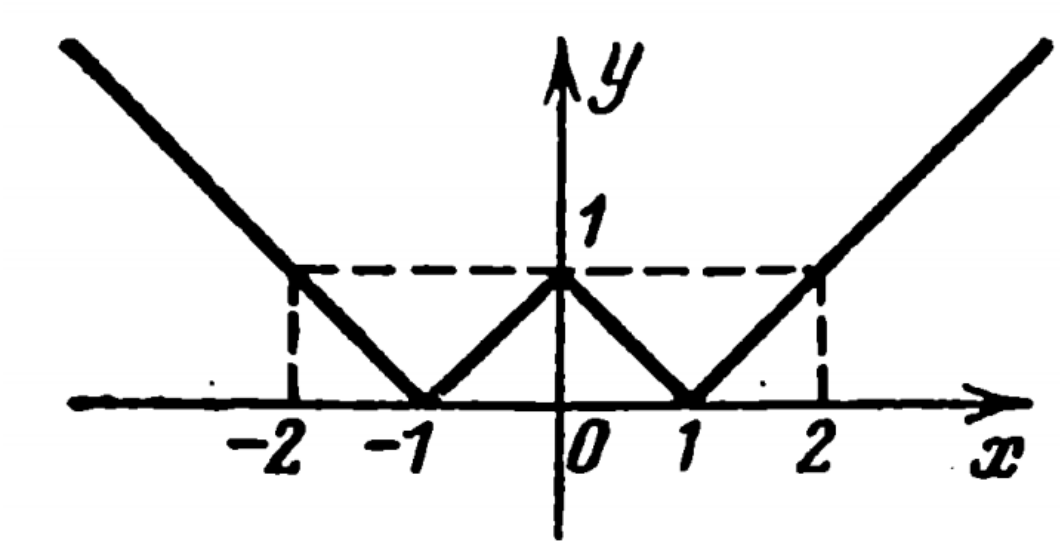

Рисунок 3.1 График функции f(x)

3.1. Анализ задачи

Для решения данной задачи нужно определить саму функцию графика. Проанализировав график, нетрудно догадаться, что его линия задаётся функцией, представленной формулой (2).

$$f(x) = ||x| - 1| \quad (2)$$

Функциональные требования: программа должна выводить на экран правильный ответ, принимать на ввод вещественные числа.

Нефункциональные требования: программа должна быть защищена от неправильного ввода, предупреждать пользователя о неправильном вводе, ожидать ответа пользователя перед завершением.

Требования к входным данным: все данные – вещественные числа.

Формат входных данных: A – вещественное число.

Формат выходных данных: Вычисленная функция – вещественное число.

3.2. Разработка алгоритма

Блок-схема алгоритма вычисления функции представлена на рисунке 3.2.

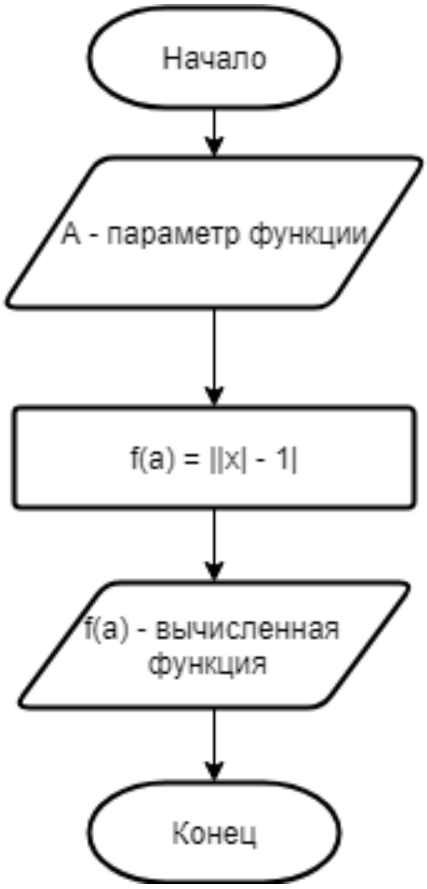

Рисунок 3.2 Вычисление функции f(x)

3.3. Реализация

Листинг программы прикреплен к приложению (приложение А), находится в папке «Вычисление заданной функции».

Реализация меню, ввода, вывода, ожидания действий пользователя используется собственная библиотека «InputOutputLib» (см. глава 13).

3.4. Тестирование

Классы и методы, предоставляемые собственной библиотекой, не тестировались. Излишние критерии, которые не несут пользы при проверке полноты тестирования, были опущены. Приведённое тестирование (табл. 3.1) удовлетворяет критериям чёрного ящика по М. А. Плаксину (табл. 3.2).

Таблица 3.1 Тесты

	Входные данные	Ожидаемые выходные	Выходные данные
T1	A = F	Неверный ввод	Неверный ввод
T2	A = -1.5	0.5	0.5
T3	A = 1	0	0

Таблица 3.2 Черный ящик

	T1	T2	T3
1. Тестирование входных данных			
1.1 Ввод целого A			+
1.2 Ввод вещественного A		+	
1.3 Ввод символов в A	+		
2. Тестирование выходных данных			
2.1 Ошибка ввода	+		
2.2 Вычисленная функция		+	+

4. «Системы счисления»

Даны натуральное число n, целые числа $a_0, \dots, a_n$ такие, что каждое a_i равно нулю или единице и $a_n \neq 0$. Последовательность $a_0, \dots, a_n$ задаёт двоичное представление некоторого числа $p = a_n * 2^n + \cdots + a_1 * 2 + a_0$. Получить последовательность нулей и единиц, задающую двоичное представление числа $3p$.

4.1. Анализ задачи

Задание требует двоичное число умножить на десятичное. Сделать это можно одним из следующих вариантов: представить вводимое пользователем число в десятичной системе, либо представить множитель в двоичной системе, так как умножение в обеих системах счисления подчиняется одинаковым законам. Число p в данной задаче представляется в двоичной системе счисления таким образом, что a_n – первый значащий элемент, значит перед тем, как производить операцию умножения или перевода, необходимо последовательность отразить.

Функциональные требования: программа должна выводить на экран верный ответ.

Нефункциональные требования: программа должна быть защищена от неправильного ввода, предупреждать пользователя о неправильном вводе, ожидать ответа пользователя перед завершением.

Требования к входным данным: Натуральные целые числа 0 или 1.

Формат входных данных: P – натуральное целое число в двоичном представлении.

Формат выходных данных: 3p – натуральное целое число в двоичном представлении.

4.2. Разработка алгоритма

Блок-схема к алгоритму представлена ниже (рис. 4.1).

Рисунок 4.1 Умножение числа P на 3

4.3. Реализация

Листинг программы прикреплен к приложению (приложение А), находится в папке «Системы счисления».

Реализация меню, ввода, вывода, ожидания действий пользователя используется собственная библиотека «InputOutputLib» (см. глава 13).

Переменная p хранит число P в десятичном представлении, переменная oldP хранит число P в двоичном представлении.

Функция GetArray получает строку, представляющую число P в двоичном представлении.

4.4. Тестирование

Классы и методы, предоставляемые собственной библиотекой, не тестировались. Излишние критерии, которые не несут пользы при проверке полноты тестирования, были опущены. Приведённое тестирование (табл. 4.1) удовлетворяет критериям чёрного ящика (табл. 4.2).

Таблица 4.1 Тесты

	Входные данные	Ожидаемые выходные	Выходные данные
T1	P = D	Неверный ввод	Неверный ввод
T2	A = 321	Неверный ввод	Неверный ввод
T3	A = 100	1100	1100

Таблица 4.2 Черный ящик

	T1	T2	T3
1. Тестирование входных данных			
1.1 Ввод последовательности из 0 и 1			+
1.2 Ввод отличных от 0 и 1 цифр		+	
1.3 Ввод символов	+		
2. Тестирование выходных данных			
2.1 Ошибка ввода	+	+	
2.2 Вычисленная функция			+

5. «Матричная задача»

Дана действительная квадратная матрица порядка n. Получить $x_1x_n + x_2x_{n-1} + \cdots + x_nx_1$, где x_k – наибольшее значение элементов k-ой строки данной матрицы.

5.1. Анализ задачи

Для решения данного задания нужно прочитать матрицу и в каждой строке найти максимальный элемент.

Задача относится к области линейной алгебры. Матрица является квадратной, если число её строк и столбцов совпадает.

Функциональные требования: программа должна выводить на экран верный ответ.

Нефункциональные требования: программа должна быть защищена от неправильного ввода, предупреждать пользователя о неправильном вводе, ожидать ответа пользователя перед завершением.

Требования к входным данным: Натуральные целые и вещественные числа.

Формат входных данных: N – натуральное целое число, порядок матрицы. N^2 вещественных чисел.

Формат выходных данных: Вещественное число – вычисленное по заданной формуле.

5.2. Разработка алгоритма

Блок-схема к алгоритму представлена ниже (рис. 5.1).

Рисунок 5.1 Обход строк матрицы

5.3. Реализация

Листинг программы прикреплен к приложению (приложение А), находится в папке «Матричная задача».

Реализация меню, ввода, вывода, ожидания действий пользователя используется собственная библиотека «InputOutputLib» (см. глава 13).

Функция GetMatrix получает матрицу от пользователя, функция GetAnswer вычисляет ответ по заданной формуле в матрце.

5.4. Тестирование

Классы и методы, предоставляемые собственной библиотекой, не тестировались. Излишние критерии, которые не несут пользы при проверке полноты тестирования, были опущены. Приведённое тестирование (табл. 5.1) удовлетворяет критериям чёрного ящика (табл. 5.2).

Таблица 5.1 Тесты

	Входные данные	Ожидаемые выходные	Выходные данные
T1	"a 1"	Ошибка ввода	Ошибка ввода: неверно введено число
T2	"-1 0 3" "4 5 6" "7 8 9"	90	90

Таблица 5.2 Черный ящик

	T1	T2
Тестирование входных данных		
Одна строка матрицы короче		
Элемент – не целое число	+	
Элемент меньше нуля		+
Элемент ноль		+
Простой ввод		+
Тестирование выходных данных		
Вывод значения		+
Ошибка ввода	+	

6. «Последовательность чисел»

Ввести a_1, a_2, a_3, M, N, L. Используя рекурсию построить последовательность чисел $a_к = (7/3* a_{к-1}+ a_{к-2}) /2 * a_{к-3}$. Построить N элементов последовательности, либо найти первые M ее элементов, большие числа L (в зависимости от того, что выполнится раньше). Напечатать последовательность и причину остановки.

6.1. Анализ задачи

Рекурсия – это вызов функции в ней же самой. В данной задаче используется простая рекурсия.

При решении задания необходимо учесть, что заданная последовательность является монотонной, то есть если элементы отрицательные, а границы положительные – возникает переполнение.

Функциональные требования: программа должна выводить на экран верный ответ.

Нефункциональные требования: программа должна быть защищена от неправильного ввода, предупреждать пользователя о неправильном вводе, ожидать ответа пользователя перед завершением.

Требования к входным данным: Вещественные и натуральные числа.

Формат входных данных: a_1, a_2, a_3, M – вещественные числа, N, L – натуральное число.

Формат выходных данных: Последовательность элементов, причина остановки построения.

6.2. Разработка алгоритма

Блок-схема к алгоритму представлена ниже (рис. 6.1).

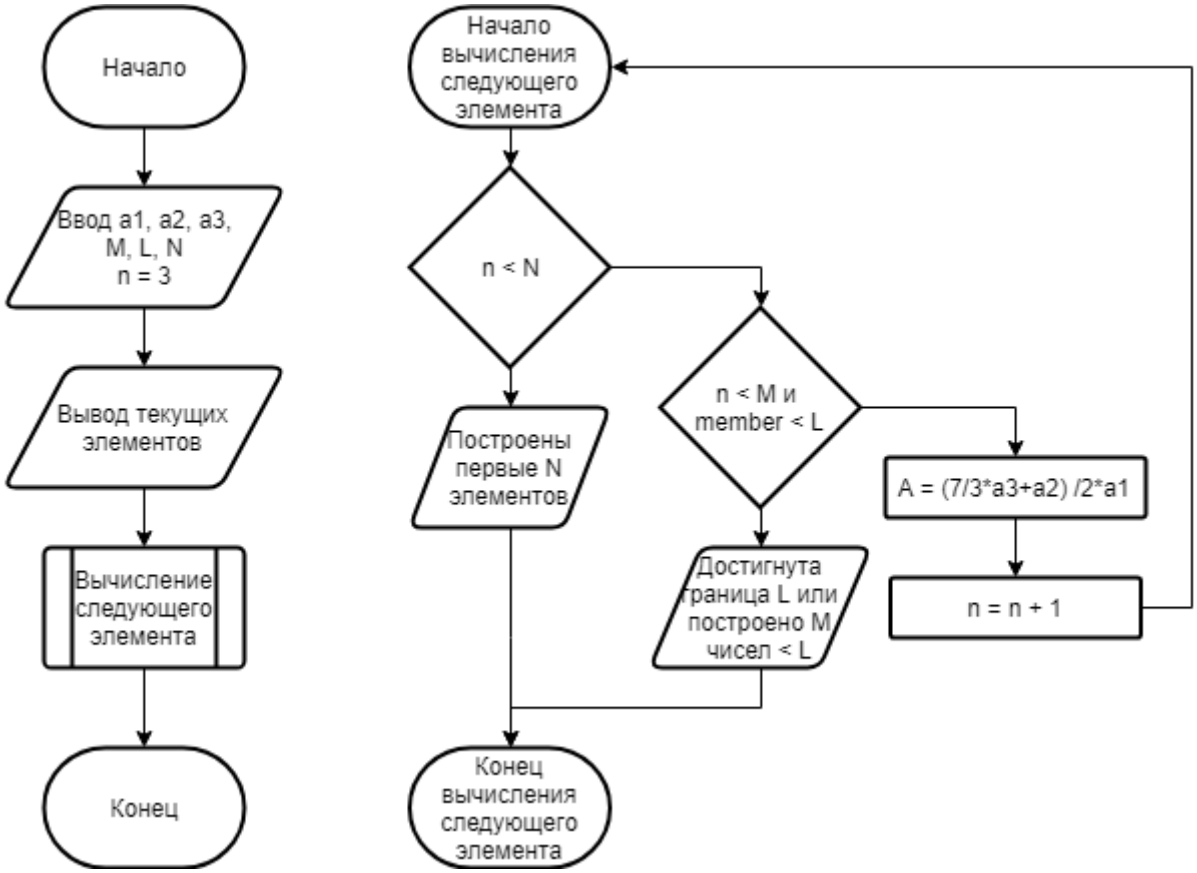

Рисунок 6.1 Построение последовательности

6.3. Реализация

Листинг программы прикреплен к приложению (приложение А), находится в папке «Последовательность чисел».

Реализация меню, ввода, вывода, ожидания действий пользователя используется собственная библиотека «InputOutputLib» (см. глава 13).

Функция Next вычисляет следующее число последовательности, проверяя его по оговоренным условиям задачи.

6.4. Тестирование

Классы и методы, предоставляемые собственной библиотекой, не тестировались. Излишние критерии, которые не несут пользы при проверке полноты тестирования, были опущены. Приведённое тестирование (табл. 6.1) удовлетворяет критериям чёрного ящика (табл. 6.2).

Таблица 6.1 Тесты

	Входные данные	Ожидаемые выходные	Выходные данные
T1	a 1 1 2	Неверный ввод	Неверный ввод
T2	1 2 3 50 25 10	1 2 3 4 11 Достигнута граница L или построено M чисел < L	1 2 3 4 11 Постройка последовательности остановлена. Достигнута граница L или построено M чисел < L
T3	1 2 3 4 25 10	1 2 3 4 Построено N первых элементов последовательсти	1 2 3 4 Постройка последовательности остановлена. Построено N первых элементов последовательсти

Таблица 6.2 Черный ящик

	T1	T2	T3
Тестирование входных данных			
Ввод не числа	+		
Обычный ввод		+	+
Тестирование выходных данных			
Неверный ввод	+		
Построено N первых		+	
Достигнута граница			+

7. «Булевы функции»

Выписать все булевы функции от 3 аргументов, которые самодвойственны. Выписать их вектора в лексикографическом порядке.

7.1. Анализ задачи

Эта задача относится к области предмета Дискретная математика. Булева функция может быть задана, как в виде схемы, как в виде вектора, так и в виде какого-то выражения. Очевидно, что существует бесконечное множество различных выражений, которые задают одну и ту же функцию. Следовательно, при поиске функций удобно работать с вектором.

$f(\{x1, x2, x3\}) = \sim f(\sim\{x1, x2, x3\})$ – на противоположных аргументах функция принимает противоположные значения. Не трудно заметить, что $\{000\}$ противоположен $\{111\}, \{001\} — \{110\}$, n-ой строчке от начала схемы – n-ая строчка от конца схемы. То же и с вектором: n-ому биту от начала вектора соответствует n-ый бит от конца вектора.

Функциональные требования: программа должна выводить на экран верный ответ. Дополнительная функция – проверка заданного пользователем вектора на самодвойственность.

Нефункциональные требования: программа должна быть защищена от неправильного ввода, предупреждать пользователя о неправильном вводе, ожидать ответа пользователя перед завершением.

Требования к входным данным: Все данные – символы.

Формат входных данных: Строка – последовательность из восьми символов, задающая вектор функции.

Формат выходных данных: Самодвойственный вектор или нет.

7.2. Разработка алгоритма

Блок-схема к алгоритму представлена ниже (рис. 7.1).

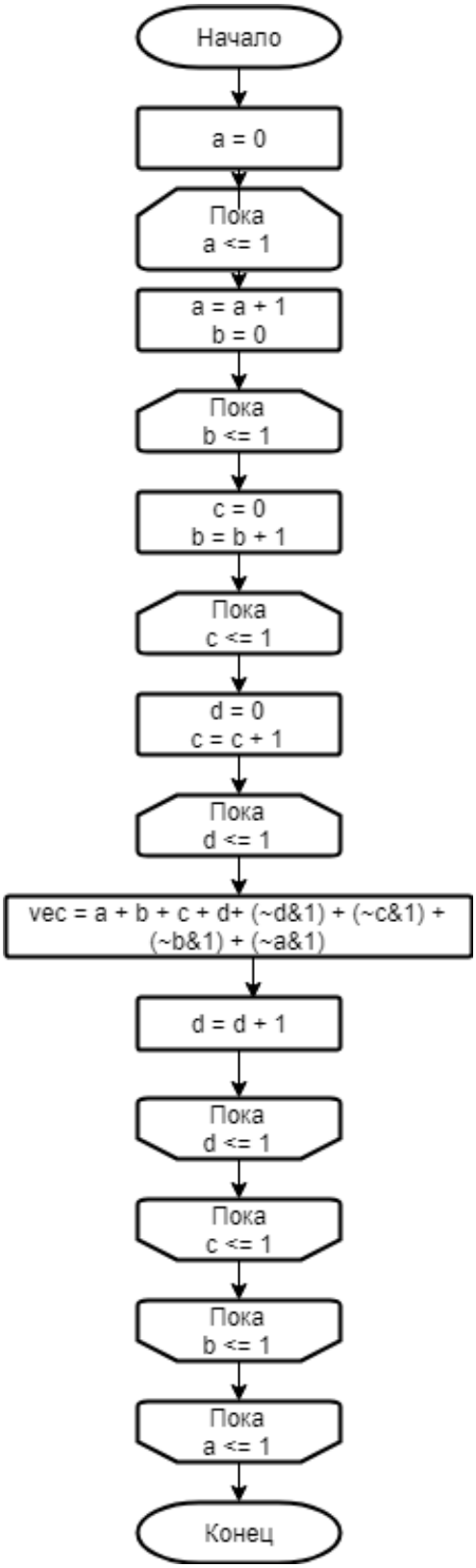

Рисунок 7.1 Вычисление самодвойственных векторов

7.3. Реализация

Листинг программы прикреплен к приложению (приложение А), находится в папке «Булевы функции».

Реализация меню, ввода, вывода, ожидания действий пользователя используется собственная библиотека «InputOutputLib» (см. глава 13).

Функция GenerateVector генерирует самодвойственные вектора от трех аргументов, функция GenerateFuntion генерирует таблицы истинности для функций от заданных векторов, функция IsDouble проверяет вектор на самодвойственность.

7.4. Тестирование

Классы и методы, предоставляемые собственной библиотекой, не тестировались. Излишние критерии, которые не несут пользы при проверке полноты тестирования, были опущены. Приведённое тестирование (табл. 7.1) удовлетворяет критериям чёрного ящика (табл. 7.2).

Таблица 7.1 Тесты

	Входные данные	Ожидаемые выходные	Выходные данные
T1	123123123	Неверный ввод	Неверный ввод
T2	11101110	Не самодвойственный вектор	Не самодвойственный вектор
T3	10100101	Самодвойственный вектор	Самодвойственный вектор

Таблица 7.2 Черный ящик

	T1	T2	T3
Тестирование входных данных			
Ввод не вектора	+		
Обычный ввод		+	+
Тестирование выходных данных			
Неверный ввод	+		
Не самодвойственный вектор		+	
Самодвойственный вектор			+

8. «Двудольный граф»

Граф задан матрицей списком вершин и ребер. Выяснить, является ли он двудольным.

8.1. Анализ задачи

Данная задача относится к области теории графов. Граф – это множество вершин, соединённых рёбрами. Двудольный граф – это такой граф, множество вершин которого можно разбить на две части таким образом, что каждое ребро в одной части соединяет вершину из другой части, но не соединяет ни одно ребро из текущей части.

Для определения необходимо проверить можно ли раскрасить заданный граф двумя цветами.

Для генератора достаточно создать любой граф.

Во внутренней памяти граф будет храниться в виде квадратной матрицы смежности.

Функциональные требования: программа должна выводить на экран верный ответ.

Нефункциональные требования: программа должна быть защищена от неправильного ввода, предупреждать пользователя о неправильном вводе, ожидать ответа пользователя перед завершением.

Требования к входным данным: Все данные – натуральные целые числа.

Формат входных данных: Количество вершин, начало и конец ребра.

Формат выходных данных: Двудольный граф или нет.

8.2. Разработка алгоритма

Блок-схема к алгоритму представлена ниже (рис. 8.1).

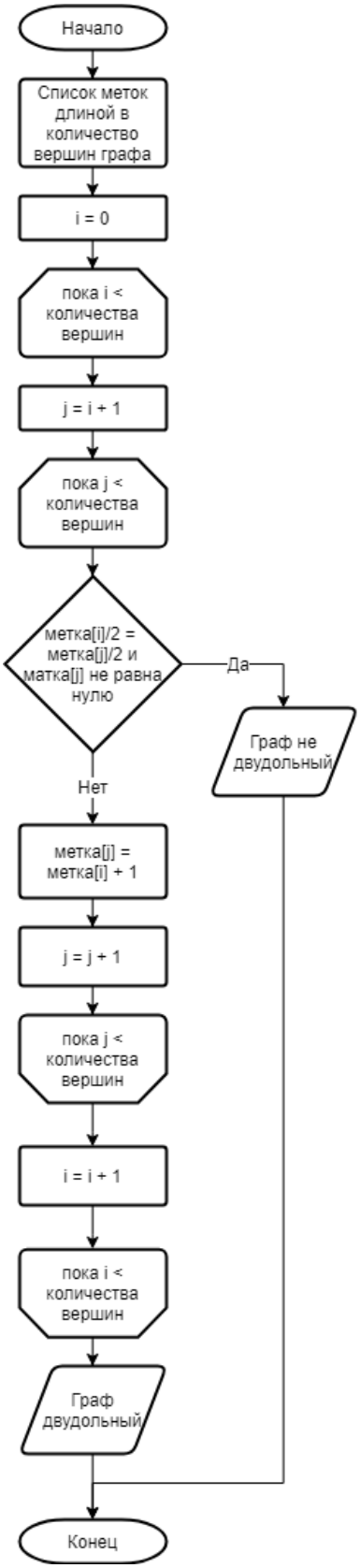

Рисунок 8.1 Блок-схема алгоритма

8.3. Реализация

Листинг программы, класса Graph, класса Testing прикреплен к приложению (приложение А), находится в папке «Двудольный граф».

Реализация меню, ввода, вывода, ожидания действий пользователя используется собственная библиотека «InputOutputLib» (см. глава 13).

Функция MakeGraph создаёт граф с указанным количеством вершин, функция IsBipartite проверять граф на двудольность по заданному алгоритму.

8.4. Тестирование

Классы и методы, предоставляемые собственной библиотекой, не тестировались. Излишние критерии, которые не несут пользы при проверке полноты тестирования, были опущены. Приведённое тестирование (табл. 8.1) удовлетворяет критериям чёрного ящика (табл. 8.2).

Таблица 8.1 Тесты

	Входные данные	Ожидаемые выходные	Выходные данные
T1	Ввод кол-ва вершин: -2 Неверный ввод, повтор: 2.1 Неверный ввод, повтор: ы	Ошибка ввода	+
T2	Вершин: 3 Рёбра: 01 12 20	Не двудольный граф	+
T3	Вершин: 2 Рёбра: 01 10	Двудольный граф	+

Таблица 8.2 Черный ящик

	T1	T2	T3
Тестирование входных данных			
Ввод отрицательного количества вершин	+		
Ввод нецелого количества вершин	+		
Ввод символа вместо количества вершин	+		
Тестирование выходных данных			
Неверный ввод	+		
Двудольный граф			+
Не двудольный граф		+	

9. «Циклический список»

Напишите метод создания циклического списка, в информационные поля элементов которого последовательно заносятся номера с 1 до N (N водится с клавиатуры). Первый включенный в список элемент, имеющий номер 1, оказывается в голове списка (первым). Разработайте методы поиска и удаления элементов списка.

9.1. Анализ задачи

Данная задача относится к области предмета Программирования, а именно к теме «Структуры данных».

Список – это динамическая структура, которая состоит из элементов, имеющих собственное значение и ссылку на следующий элемент списка. В циклическом списке последний элемент ссылается на первый, таким образом создавая «кольцо».

Для поиска и удаления элементов в таком списке необходимо учитывать, что каждый элемент имеет ссылку на следующий, а значит, что если перебирать элементы по порядку, то программа войдёт в бесконечный цикл.

Функциональные требования: программа должна выводить на экран созданный циклический список, поддерживать поиск и удаление элементов в созданном списке.

Нефункциональные требования: программа должна быть защищена от неправильного ввода, предупреждать пользователя о неправильном вводе, ожидать ответа пользователя перед завершением.

Требования к входным данным: Все данные – целые положительные числа.

Формат входных данных: Целое положительное число – количество элементов в списке.

Формат выходных данных: Последовательность элементов в списке.

9.2. Разработка алгоритма

Блок-схема к алгоритму представлена ниже (рис. 9.1).

Рисунок 9.1 Алгоритм создания списка

9.3. Реализация

Листинг программы и класса El прикреплен к приложению (приложение А), находится в папке «Циклический список».

Реализация меню, ввода, вывода, ожидания действий пользователя используется собственная библиотека «InputOutputLib» (см. глава 13).

Функция Add добавляет элемент в список, функция Remove удаляет элемент из списка, функция Search выполняет поиск элемента в списке, функция MakeList создаёт список заданного размера.

9.4. Тестирование

Классы и методы, предоставляемые собственной библиотекой, не тестировались. Излишние критерии, которые не несут пользы при проверке полноты тестирования, были опущены. Приведённое тестирование (табл. 9.1) удовлетворяет критериям чёрного ящика (табл. 9.2).

Таблица 9.1 Тесты

	Входные данные	Ожидаемые выходные	Выходные данные
T1	Кол-во: -5 Неверный ввод. Повтор: 2.1 Неверный ввод. Повтор: a Неверный ввод. Повтор: 0	Ошибка ввода	Ошибка ввода
T2	Кол-во: 5	1 2 3 4 5	1 2 3 4 5

Таблица 9.2 Черный ящик

	T1	T2
Тестирование входных данных		
Ввод отрицательного кол-ва	+	
Ввод кол-ва символом	+	
Ввод нецелого кол-ва	+	
Ввод количества		+
Тестирование выходных данных		
Ошибка ввода	+	
Вывод последовательности		+
Тестирование ОДЗ		
Нулевая последовательность	+	
Простая последовательность		+

10. «Удаление вершин из графа»

Написать метод удаления из графа всех вершин с заданным значением информационного поля.

10.1. Анализ задачи

Задача относится к теории графов. Сложность зависит от программного представления графа. Чтобы удалить вершину из графа в матрице смежности, нужно удалить строку и ряд с номером данной вершины.

Функциональные требования: программа должна выводить на экран верный ответ.

Нефункциональные требования: программа должна быть защищена от неправильного ввода, предупреждать пользователя о неправильном вводе, ожидать ответа пользователя перед завершением.

Требования к входным данным: Все данные – натуральные целые числа.

Формат входных данных: Количество вершин, начало и конец ребра.

Формат выходных данных: Граф с удалённой вершиной.

10.2. Разработка алгоритма

Блок-схема к алгоритму представлена ниже (рис. 10.1).

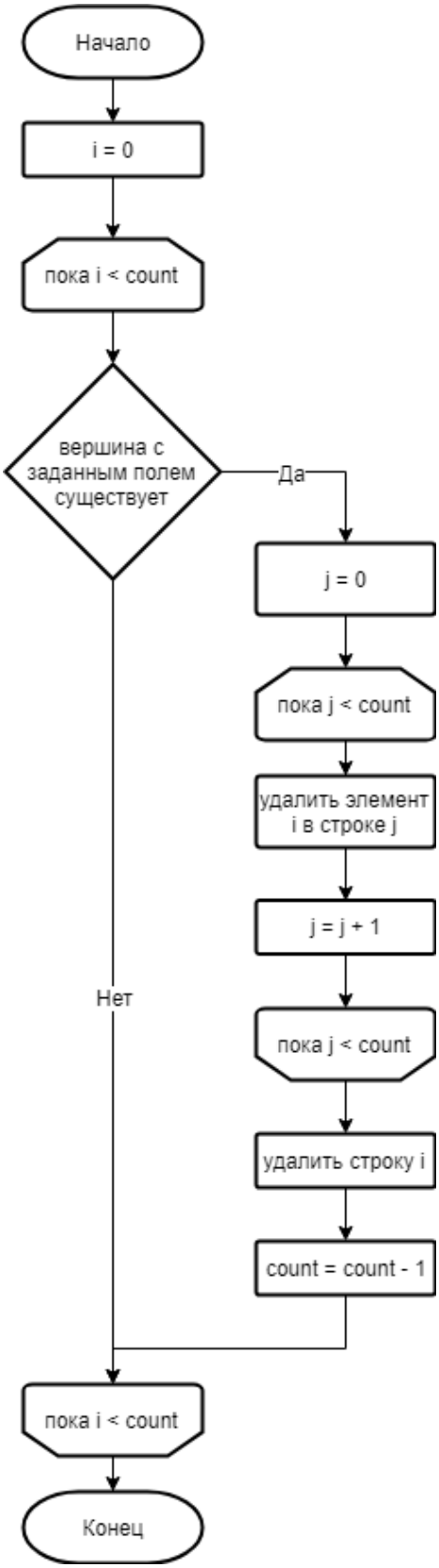

Рисунок 10.1 Блок-схема удаления вершин из графа

10.3. Реализация

Листинг программы и класса Graph прикреплен к приложению (приложение А), находится в папке «Удаление вершин из графа».

Реализация меню, ввода, вывода, ожидания действий пользователя используется собственная библиотека «InputOutputLib» (см. глава 13).

Функция MakeMatrix создаёт матрицу смежности графа, функция DelPoint удаляет из графа вершины с заданным информационным полем.

10.4. Тестирование

Классы и методы, предоставляемые собственной библиотекой, не тестировались, создание графа тестировалось в предыдущей задачи с использованием класса Graph. Излишние критерии, которые не несут пользы при проверке полноты тестирования, были опущены. Приведённое тестирование (табл. 10.1) удовлетворяет критериям чёрного ящика (табл. 10.2).

Таблица 10.1 Тесты

	Входные данные	Ожидаемые выходные	Выходные данные
T1	Несуществующее информационное поле	Ошибка ввода	+
T2	Вершин: 3 Поля: 1 2 3, ввод: 2	Граф без вершины 2	+
T3	Вершин: 3 Поля: 2 2 3, ввод: 2	Граф без вершин 2	+

Таблица 10.2 Черный ящик

	T1	T2	T3
Тестирование входных данных			
Ввод несуществующего поля	+		
Ввод вершина, которая содержится один раз		+	
Ввод вершина, которая содержится не один раз			+
Тестирование выходных данных			
Неверный ввод	+		
Граф с удалёнными вершинами		+	+

11. «Криптография»

Чтобы зашифровать текст из 121 буквы, его можно записать в квадратную матрицу порядка 11 по строкам, а затем прочитать по спирали, начиная с центр (т. е. с элемента, имеющего индексы 6, 6). Зашифровать текст по данному алгоритму. Расшифровать текст, зашифрованный данным алгоритмом.

11.1. Анализ задачи

Данная задача относится к Дискретной математике, шифрованию. Для шифрования, после получения строки, каждый символ нужно последовательно записать в матрицу, а после прочитать её по спирали. Для расшифровки, после получения строки, нужно каждый символ последовательно записать в матрицу по спирали, а после последовательно прочитать элементы полученной матрицы.

Функциональные требования: программа должна выводить на экран верный ответ.

Нефункциональные требования: программа должна быть защищена от неправильного ввода, предупреждать пользователя о неправильном вводе, ожидать ответа пользователя перед завершением.

Требования к входным данным: Все данные – строки длиной 121 символ.

Формат входных данных: Строка, длиной 121 символ.

Формат выходных данных: Зашифрованная или дешифрованная строка из 121 символа.

11.2. Разработка алгоритма

Блок-схема к алгоритму представлена ниже (рис. 11.1).

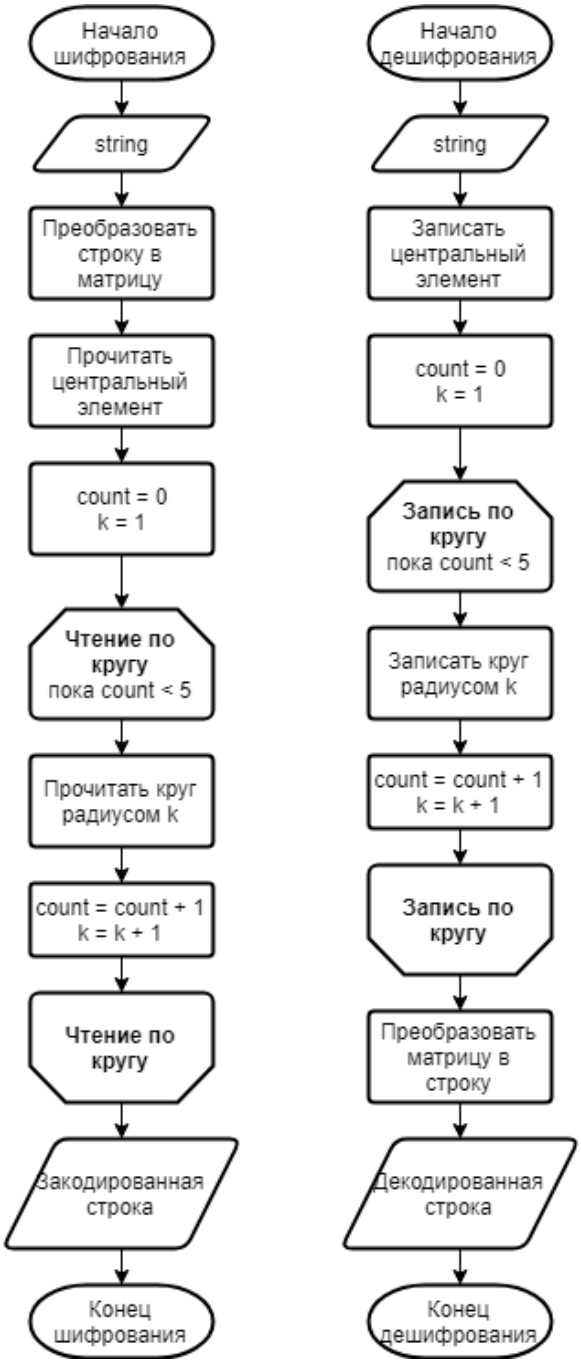

Рисунок 11.1 Блок-схема шифровки и дешифровки

11.3. Реализация

Листинг программы прикреплен к приложению (приложение А), находится в папке «Криптография».

Реализация меню, ввода, вывода, ожидания действий пользователя используется собственная библиотека «InputOutputLib» (см. глава 13).

Функция GetString получает строку и делает проверку длины, функция RemakeString преобразует строку в матрицу, функция RemakeArray преобразует матрицу в строку, функция Encode кодирует строку с помощью матрицы по заданному алгоритму, функция Decode декодирует строку с помощью матрицы по заданному алгоритму.

11.4. Тестирование

Классы и методы, предоставляемые собственной библиотекой, не тестировались. Излишние критерии, которые не несут пользы при проверке полноты тестирования, были опущены. Приведённое тестирование (табл. 11.1) удовлетворяет критериям чёрного ящика (табл. 11.2).

Таблица 11.1 Тесты

	Входные данные	Ожидаемые выходные	Выходные данные
T1	Ma quande lingues coalesce, li grammatica del resultant lingue es plu simplic e regulari quam ti del coalescent lingues.a	Ошибка ввода	Ошибка ввода
T2	Li Europan lingues es membres del sam familie. Lor separat existentie es un myth. Por scientie, musica, sport etc., li to	xr i seeo famssuroP . Lmbres dietn m ,eitnhet amngues eselpe sutrops ,acetia.seii Europan lianmcs ot il ,.cteiiytre mlL	Закодированная строка: xr i seeo famssuroP . Lmbres dietn m ,eitnhet amngues eselpe sutrops ,acetia.seii Europan lianmcs ot il ,.cteiiytre mlL
T3	xr i seeo famssuroP . Lmbres dietn m ,eitnhet amngues eselpe sutrops ,acetia.seii Europan lianmcs ot il ,.cteiiytre mlL	Li Europan lingues es membres del sam familie. Lor separat existentie es un myth. Por scientie, musica, sport etc., li to	Декодировка закодированной строки: Li Europan lingues es membres del sam familie. Lor separat existentie es un myth. Por scientie, musica, sport etc., li to

Таблица 11.2 Черный ящик

	T1	T2	T3
Тестирование входных данных			
Ввод строки не из 121 символа	+		
Ввод строки из 121 символа		+	+
Тестирование выходных данных			
Ошибка ввода	+		
Закодированное сообщение	+	+	
Декодированное сообщение			+

12. «Сортировки»

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.

Сортировки: простыми вставками, блочная сортировка.

12.1. Анализ задачи

Массивы заранее заготовлены в программе и не требуют пользовательского ввода.

Сортировка простыми вставками сортирует массив, проходя по нему один раз и перемещая элемент, «вставляя» его на своё место, путем сравнения с уже отсортированными элементами. Вычислительная сложность – $O(n^2)$.

Блочная сортировка разбивает отрезок $[0, 1)$ на n одинаковых отрезков и делит по этим блокам n входных величин, затем каждый из карманов последовательно сортируется и перемещается обратно в исходный массив. Вычислительная сложность – $O(n)$.

Функциональные требования: реализация двух методов сортировки на трёх различных массивах. Дополнительная функция – сортировка массива из 10000 элементов.

Нефункциональные требования: программа должна быть защищена от неправильного ввода, предупреждать пользователя о неправильном вводе, ожидать ответа

пользователя перед завершением, выводить количество сравнений и перестановок. Дополнительная функция – программа должна считать тики, затраченные на сортировку.

Требования к входным данным: -

Формат входных данных: -

Формат выходных данных: Отсортированные массивы, количество сравнений и количество перестановок.

12.2. Разработка алгоритма

Блок-схема к алгоритму представлена ниже (рис. 12.1).

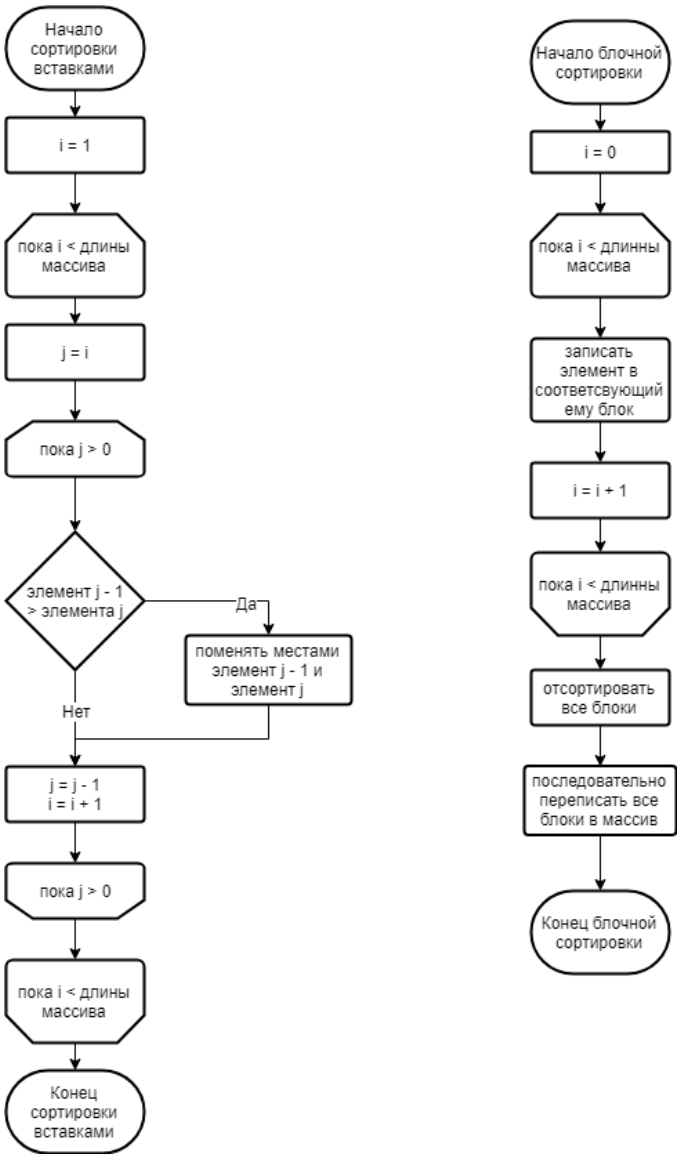

Рисунок 12.1 Блок-схема алгоритмов сортировки

12.3. Реализация

Листинг программы прикреплен к приложению (приложение А), находится в папке «Сортировки».

Реализация меню, ввода, вывода, ожидания действий пользователя используется собственная библиотека «InputOutputLib» (см. глава 13).

Функция EasyEnterSort сортирует массив методом простых вставок, функция BucketSort сортирует массив методом блочной сортировки.

12.4. Тестирование

В данной задаче входные данные отсутствуют, тестирование по критериям чёрного ящика не может быть проведено.

13. Библиотека «InputOutputLib»

Требуется создать библиотеку методов для удобной работы.

В библиотеке необходимо реализовать методы запроса информации у пользователя, метод ожидания действия пользователя, методы создания меню и работы с ним.

13.1. Анализ задачи

Библиотека должна включать в себя требуемые методы, а их вызов должен быть коротким и быстрым.

Функциональные требования: реализация считывания и записи, ожидания ответа, вывода меню, запросов к пользователю.

Нефункциональные требования: защита от некорректного ввода.

Требования к входным данным: -

Формат входных данных: -

Формат выходных данных: -

13.2. Разработка алгоритма

Блок-схема к алгоритму представлена ниже (рис. 13.1).

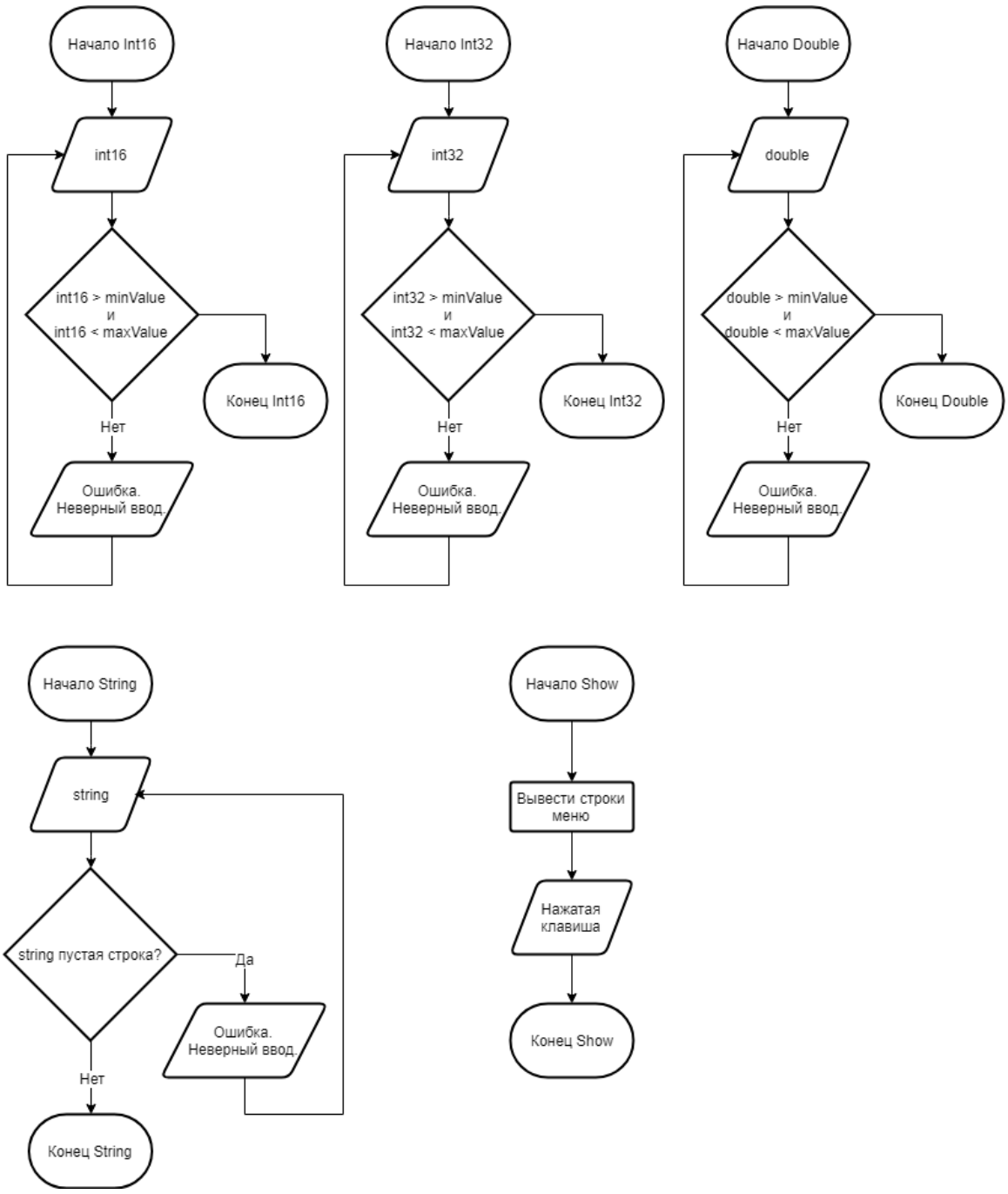

Рисунок 13.1 Блок-схемы методов библиотеки

13.3. Реализация

Листинг библиотеки прикреплен к приложению (приложение А), находится в папке «InputOutputLib».

Класс Get содержит следующие методы: Int16 – получает от пользователя число в формате Int16, Int32 – получает от пользователя число в формате Int32, Double – получает от

пользователя число в формате Double, String – получает от пользователя строку в формате string, Wait – выводит текст ожидания и ждёт нажатия любой клавиши.

Класс Menu содержит конструктор, который принимает в себя массив строк и метод Show, который выводит меню, ждёт нажатие клавиши и возвращает её код.

13.4. Тестирование

Излишние критерии, которые не несут пользы при проверке полноты тестирования, были опущены. Приведённое тестирование (табл. 13.1) удовлетворяет критериям чёрного ящика (табл. 13.2).

Таблица 13.1 Тесты

	Входные данные	Ожидаемые выходные	Выходные данные
T1	ввод int (от 0 до 100): в ошибка Повтор: 3.2 ошибка Повтор: -3 ошибка Повтор: 101 ошибка Повтор: 1 ввод double (от 0 до 100): а ошибка Повтор: -3 ошибка Повтор: 101 ошибка Повтор: 1 ввод string: " " ошибка Повтор: "" ошибка Повтор: "в"	Ошибки ввода int: 1 double: 1 string: в	+
T2	ввод int: 1 ввод double: 1,6 ввод string: один	int: 1 double: 1,6 string: один	+

Таблица 13.2 Черный ящик

	T1	T2
Символ вместо числа	+	
Нецелое число int	+	
Символ вместо double	+	
Пустая строка в string	+	
Строка из пробелов в string	+	
Число меньше минимального int	+	
Число меньше минимального double	+	
Число больше максимального int	+	
Число больше максимального double	+	
Корректное считывание		+
Ошибка	+	

Заключение

В ходе выполнения работы были получены, углублены и расширены знания и навыки создания и описания алгоритмов, их реализации на языке программирования высокого уровня C#.

Были изучены такие методы решения задач, как динамическое программирование, поиск по графи в ширину, поиск по графу в глубину, проход по графу, использование комбинаций и переводом между системами счисления, реализация рекурсивных функций. Рассмотрены различные области принадлежности задач: дискретная математика, математическая статистика, линейная алгебра, дискретная математика.

Во время разработки, тестирования и отладки были приобретены навыки работы с системой контроля версий Git, восстановления, систематизации и документирования этапов реализации решения задач.

Библиографический список

1. Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. Алгоритмы: построение и анализ. — 3-е издание. — М.: «Вильямс», 2013.
2. Левитин А. В. Алгоритмы. Введение в разработку и анализ. — М.: «Вильямс», 2006.
3. Плаксин М. А. Тестирование и отладка программ для профессионалов будущих и настоящих. — 2-е изд. — М.: БИНОМ. Лаборатория знаний, 2013.
4. Морозенко В.В. Дискретная математика: учебное пособие. — Перм. гос. ун-т. - Пермь, 2008. - 244с.

Приложение А. CD диск с исходными кодами