Пермский филиал федерального государственного автономного образовательного учреждения высшего образования «Национальный исследовательский университет

«Высшая школа экономики»

Факультет экономики, менеджмента и бизнес-информатики

Образовательная программа бакалавриата «Программная инженерия»

**ОТЧЕТ  
по учебной практике**

Выполнил студент группы ПИ-19-3 Никитин Андрей Владимирович

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

Проверил:

Руководитель практики   
преподаватель кафедры информационных

технологий в бизнесе

Ланин Вячеслав Владимирович

\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(оценка) (подпись)

\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата)

Пермь, 2019 год

**Аннотация**

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

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

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

Библиографический список включает в себя 4 источника.

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

**Введение**

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

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

Для достижения поставленной цели необходимо:

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

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

1. **Задача 1**

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

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

Входной файл: input.txt

Выходной файл: output.txt

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

* 1. **Анализ**

Для решения задачи нужно:

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

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

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

## **Проектирование**

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

Изображение выглядит как текст, карта

Автоматически созданное описание

Рисунок 1.1. Блок-схема для задачи 1

## **Тестирование**

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

Таблица 1.1. Таблица тестов для задачи 1

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Входные данные** | **Ожидаемые результаты** | **Реальные результаты** |
|
| Т1 | Файла input.txt нет Файла output.txt нет | 0 | 0 |
| Т2 | Файл input.txt пуст | 0 | 0 |
| Т3 | 2  255.255.255.255  255.255.255.0  3  192.168.31.1 192.168.31.2  192.168.31.3 192.168.31.4  192.168.31.1 192.167.31.2 | 1  1  0 | 1  1  0 |

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

Требования к скорости работы и занимаемой памяти также были выполнены.

1. **Задача 2**

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

Входной файл: input.txt

Выходной файл: output.txt

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

1. 1. **Анализ**

Для решения задачи нужно:

1. Считать количество множеств и чисел, которые может принять множество, из файла.
2. Считать количество операций.
3. Выяснить, что это за операция.
4. Выполнить соответствующую операцию.
5. Повторять, пока операций в файле не кончатся.

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

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

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

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

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

## **Проектирование**

Изображение выглядит как текст, карта

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

Рисунок 2.1. Блок-схема для задачи 2

## **Тестирование**

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

Таблица 1.1. Таблица тестов для задачи 1

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Входные данные** | **Ожидаемые результаты** | **Реальные результаты** |
|
| Т1 | Файла input.txt нет | 0 | 0 |
| Т2 | Файл input.txt пуст | 0 | 0 |
| Т3 | 10 10 5 ADD 1 1, ADD 1 2, ADD 2 1 LISTSET 1, LISTSETSOF 1 | 1 2  1 2 | 1 2  1 2 |

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

Требования к скорости работы и занимаемой памяти также были выполнены.

# **Задача 3**

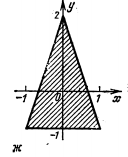
 Даны действительные числа х , у. Определить, принадлежит ли точка с координатами х , у заштрихованной части плоскости (рис. 3.1).

Рисунок 3.1. График к задаче 3

## **Анализ**

Данному графику (рис.3.1) соответствует функция

.

Для вычисления принадлежности точки .

Также необходимо производить проверку ввода на правильность ввода действительного числа.

## **Проектирование**

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

Изображение выглядит как стол, игра

Автоматически созданное описание

Рисунок 3.2. Блок-схема для задачи 3

## **Тестирование**

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

Таблица 3.1. Таблица тестов для задачи 2

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Входные данные** | **Ожидаемые результаты** | **Реальные результаты** |
|
| Т1 | h | Ошибка ввода | Ошибка ввода |
| Т2 | 0 0 | Принадлежит | Принадлежит |
| Т3 | -0,5 -1 | Принадлежит | Принадлежит |
| Т4 | -0,5 1 | Принадлежит | Принадлежит |
| Т5 | 0 -1 | Принадлежит | Принадлежит |
| Т6 | 4 0 | Не принадлежит | Не принадлежит |

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

# **Задача 4**

Даны действительные числа .

Получить , где – комплексные числа . (Определить процедуры выполнения арифметических операций над комплексными числами.)

## **Анализ**

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

Для проверки правильности ввода необходимо реализовать проверку входных данных.

## **Проектирование**

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

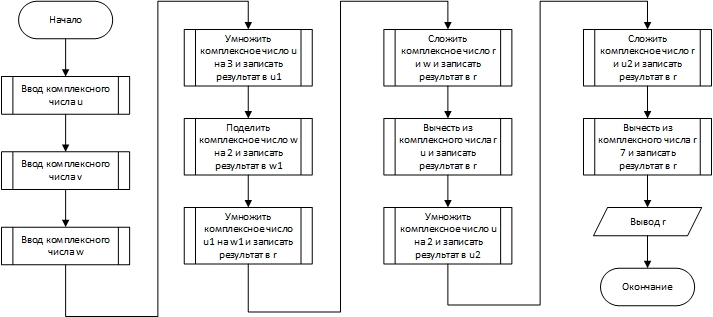


Рисунок 4.1. Блок-схема для задачи 4

## **Тестирование**

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

Таблица 4.1. Таблица тестов для задачи 4

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Входные данные** | **Ожидаемые результаты** | **Реальные результаты** |
|
| Т1 | j | Ошибка ввода | Ошибка ввода |
| Т2 | -1 2  0.2 5  0 0 | -8-2i | -8-2i |

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

# **Задача 5**

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

Рисунок 5.1. Заштрихованная часть к задаче 5

## **Анализ**

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

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

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

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

## **Проектирование**

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

Изображение выглядит как текст, карта

Автоматически созданное описание

Рисунок 5.2. Блок-схема алгоритма для задачи 5

## **Тестирование**

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

Таблица 5.1. Таблица тестов для задачи 5

| **№** | **Входные данные** | **Ожидаемые результаты** | **Реальные результаты** |
| --- | --- | --- | --- |
|
|  |
| Т1 | ываыва | Ошибка ввода | Ошибка ввода |
| Т2 | -23 | Ошибка ввода | Ошибка ввода |
| Т3 | 5 64 18 33 -50 -40  73 62 48 -84 -39  87 97 7 -37 8  -13 91 -97 22 -7  -70 -1 -71 -85 17 | 91 | 91 |
| Т4 | 1 -1 | -1 | -1 |
| Т5 | 4  33 -60 -54 96  -22 46 36 67  -79 37 -30 -9  28 61 52 -32 | 61 | 61 |

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

# **Задача 6**

Ввести а1, а2, а3. Построить последовательность чисел . Построить N элементов последовательности проверить, образуют ли элементы, стоящие на четных местах, возрастающую подпоследовательность.

## **Анализ**

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

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

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

## **Проектирование**

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

Изображение выглядит как текст, карта

Автоматически созданное описание

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

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

## **Тестирование**

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

Таблица 6.1. Таблица тестов для задачи 6

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Входные данные** | **Ожидаемые результаты** | **Реальные результаты** |
|
| Т1 | y | Ошибка ввода | Ошибка ввода |
| Т2 | 6 y | Ошибка ввода | Ошибка ввода |
| Т3 | 6 -8 h | Ошибка ввода | Ошибка ввода |
| Т4 | 4 5,6 3 t | Ошибка ввода | Ошибка ввода |
| Т5 | 0 0 0 8 | Последовательность бесконечна, состоит из нулей | Последовательность бесконечна, состоит из нулей |
| Т6 | 9 6 3 5 | Последовательность: 9 6 7 40 456 Длина: 5 Элементы, стоящие на четных местах, образуют возрастающую последовательность | Последовательность: 9 6 7 40 456 Длина: 5 Элементы, стоящие на четных местах, образуют возрастающую последовательность |

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

1. **Задача 7**

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

* 1. **Анализ**

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

Сочетаниями с повторениями называются наборы по M элементов, в которых каждый элемент множества N может участвовать несколько раз. При этом на соотношение значений M и N не накладывается никаких ограничений, а общее количество сочетаний с повторениями составляет .

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

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

* 1. **Проектирование**

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

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

Изображение выглядит как текст, карта

Автоматически созданное описание

***Рисунок 7.1. Блок-схема алгоритма создания сочетаний для задачи 7***

* 1. **Тестирование**

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

***Таблица 7.1. Таблица тестов для задачи 7***

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Входные данные** | **Ожидаемые результаты** | **Реальные результаты** |
|
| Т1 | y | Ошибка ввода | Ошибка ввода |
| Т2 | -2 | Ошибка ввода | Ошибка ввода |
| Т3 | 2 3 | 1: 1 1 2  2: 1 2 2  3: 2 2 2 | 1: 1 1 2  2: 1 2 2  3: 2 2 2 |
| Т4 | 4 3 | 1: 1 1 2  2: 1 1 3  3: 1 1 4  4: 1 2 2  5: 1 2 3  6: 1 2 4  7: 1 3 3  8: 1 3 4  9: 1 4 4  10: 2 2 2  11: 2 2 3  12: 2 2 4  13: 2 3 3  14: 2 3 4  15: 2 4 4  16: 3 3 3  17: 3 3 4  18: 3 4 4  19: 4 4 4 | 1: 1 1 2  2: 1 1 3  3: 1 1 4  4: 1 2 2  5: 1 2 3  6: 1 2 4  7: 1 3 3  8: 1 3 4  9: 1 4 4  10: 2 2 2  11: 2 2 3  12: 2 2 4  13: 2 3 3  14: 2 3 4  15: 2 4 4  16: 3 3 3  17: 3 3 4  18: 3 4 4  19: 4 4 4 |

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

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