МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»

Институт компьютерных наук и кибербезопасности

Направление 02.03.01 Математика и компьютерные науки

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №2

«Тестирование ПО методом белого и чёрного ящика»

Обучающийся: _____ Шклярова Ксения Алексеевна

Руководитель: _____ Курочкин Михаил Александрович

«_____»_____20___г.

Санкт-Петербург, 2025

Содержание

Введение

Модульное тестирование — это процесс тестирования отдельных блоков, подпрограмм, классов или процедур, образующих крупную программу. Другими словами, прежде чем тестировать программу в целом, необходимо сосредоточить внимание на ее меньших по размеру компонентах. Цель модульного тестирования - сравнение функций, реализуемых модулем, со спецификациями, описывающими его функциональные или интерфейсные характеристики.

В рамках данной работы требуется изучить методологии тестирования «черного ящика» и «белого ящика». После этого необходимо протестировать две программы, разработанные другими программистами методами «черного» и «белого ящика».

1 Описание методов тестирования

1.1 Метод «черного ящика»

Одной из важнейших стратегий тестирования является тестирование методом черного ящика. В соответствии с этим методом программа рассматривается как «черный ящик», внутреннее поведение и структура которого не имеют никакого значения. Вместо этого все внимание фокусируется на выяснении обстоятельств, при которых поведение программы не соответствует спецификации.

При таком подходе исходными данными являются спецификации требований, на основе которых выбираются тестовые данные, т.е. без привлечения каких-либо знаний о внутренней структуре программы.

Основными техниками являются:

1. Классы эквивалентности
2. Анализ граничных значений
3. Причинно-следственная диаграмма

1.1.1 Классы эквивалентности

Определение классов эквивалентности сводится к последовательному рассмотрению каждого из входных условий и разбиению его на две или несколько групп.

Эти группы делятся на:

• Допустимые классы эквивалентности, представляющие допустимые входные данные программы;

• Недопустимые классы эквивалентности, представляющие все остальные возможные состояния условий

Второй этап заключается в использовании классов эквивалентности для определения тестов. Этот процесс состоит из следующих этапов:

• Назначение каждому классу эквивалентности уникального номера.

• Запись новых тестов, охватывающих как можно большее количество оставшихся неохваченными допустимых классов эквивалентности, пока не будут покрыты все допустимые классы.

• Запись новых тестов, каждый из которых охватывает один и только один из оставшихся неохваченными недопустимых классов эквивалентности, пока не будут покрыты все недопустимые классы.

1.1.2 Анализ граничных значений

Граничные условия - это ситуации, возникающие в области граничных значений входных и выходных классов эквивалентности. Анализ граничных значений отличается от методики разбиения на классы эквивалентности в двух отношениях:

• Вместо того чтобы выбирать любой элемент класса эквивалентности в качестве представителя всего класса, анализ граничных значений требует выбирать такой элемент или элементы, которые обеспечивают тестирование каждой границы класса.

• При создании тестов внимание фокусируется не только на входных условиях, но и на пространстве результатов.

1.1.3 Причинно-следственная диаграмма

Тестирование всех комбинаций входных условий практически невозможно из-за их огромного количества. Метод причинно-следственных диаграмм систематически выбирает наиболее эффективные тесты и выявляет неполноту и неоднозначность в спецификациях. Спецификация транслируется в формальный язык(причинно-следственную диаграмму), аналог логической схемы, использующий простую нотацию. Требуется понимание базовых логических операторов (AND, OR, NOT).

Процесс построения тестов:

1. Спецификация разбивается на части, с которыми легче работать.

2. В спецификации определяются причины и следствия. Причина — это отдельное входное условие или класс эквивалентности входных условий. Следствие — это выходное условие или преобразование системы (долговременное воздействие, которое входное условие оказывает на состояние программы или системы). Каждой причине и каждому следствию присваивается уникальный номер.

3. Семантическое содержание спецификации анализируется и преобразуется в булев граф, связывающий причины и следствия. Полученный граф называется причинно-следственной диаграммой.

4. Диаграмма снабжается примечаниями, задающими ограничения и описывающими комбинации причин и (или) следствий, реализация которых невозможна из-за синтаксических или внешних ограничений.

5. Путем методичного прослеживания состояний условий диаграмма преобразуется в таблицу решений с ограниченными входами. Каждый столбец таблицы решений соответствует тесту.

6. Столбцы таблицы решений преобразуются в тесты.

Базовая нотация причинно-следственных диаграмм представлена на рис. 1. Каждый узел диаграммы может находиться в двух состояниях: 0 или 1, где 0 представляет состояние «отсутствует», а 1 — «присутствует».

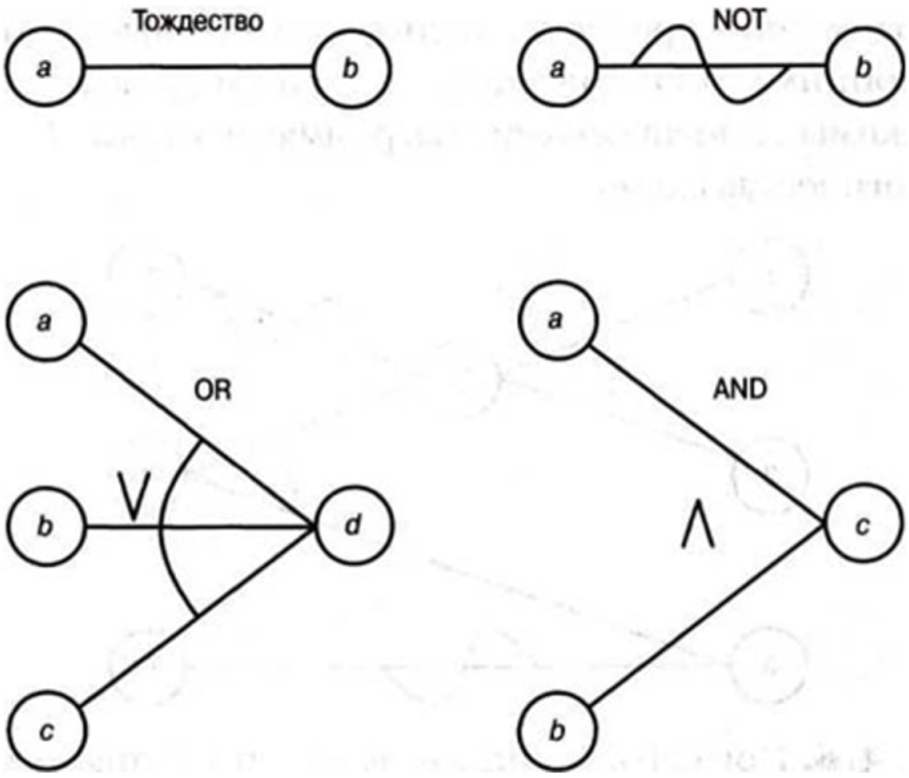

Рис. 2: Базовая нотация причинно-следственных диаграмм

• Функция тождество устанавливает, что если a равно 1, то и в равно 1; в противном случае в равно 0.

• Функция not устанавливает, что если a равно 1, то в равно 0; в противном случае b равно 1.

• Функция or устанавливает, что если a, или b, или c равно 1, то d равно 1; в противном случае d равно 0.

• Функция and устанавливает, что если и a, и b равно 1, то c равно 1; в противном случае c равно 0.

Необходимо преобразовать причинно-следственную диаграмму в таблицу решений, чтобы на ее основе разработать тесты. Для этого нужно:

1. Выбрать следствие, которое должно находиться в состоянии 1;

2. Продвигаясь вдоль линий диаграммы, ведущих к данному узлу, в обратном направлении, производится поиск комбинаций причин (подчиненных ограничениям), которые устанавливают данное следствие в 1;

• Если путь обратной трассировки проходит через узел типа OR, выход которого должен принимать значение 1, то не следует одновременно устанавливать в 1 более одного входа в этот узел.

• Если путь обратной трассировки проходит через узел типа AND, выход которого должен принимать значение 0, то, конечно же, должны быть перечислены все комбинации входов, приводящие к установке выхода в 0. Однако при исследовании ситуации, в которой один вход равен 0, а один или несколько других входов равны 1, перечислять все условия, при которых другие входы находятся в состоянии 1, не обязательно.

• Если путь обратной трассировки проходит через узел типа AND, выход которого должен принимать значение 0, то необходимо указать лишь одно условие, при котором все входы являются нулями.

3. Для каждой комбинации причин создается столбец в таблице решений;

4. Определить для каждой комбинации состояния всех других следствий и поместить их в соответствующий столбец таблицы.

После построения таблицы решений следующим шагом является создание тестов. Каждый столбец таблицы решений представляет собой уникальную комбинацию входных условий (причин) и соответствующих выходных значений (следствий). Таким образом, каждый столбец можно рассматривать как отдельный тест.

1.2 Метод белого ящика

Проектирование тестов методом белого ящика основывается на детальном знании логики работы программы, что позволяет использовать сё структуру, представленную в виде блок-схемы. Задача тестирования в этом подходе заключается в том, чтобы охватить тестами все возможные пути выполнения программы.

1.2.1 Покрытие операторов

Критерием покрытия является выполнение каждого оператора программы хотя бы один раз. Это необходимое, но недостаточное условие для приемлемого тестирования по принципу белого ящика.

1.2.2 Покрытие решений

В соответствии с этим критерием количество тестов должно быть таким, чтобы каждое условие в программе хотя бы раз принимало как значение true (истина), так и false (ложь). Иными

словами, каждая логическая ветвь каждой инструкции ветвления в программе должна быть выполнена хотя бы один раз.

1.2.3 Покрытие условий

В соответствии с этим критерием количество тестов должно быть таким, чтобы каждое из элементарных условий, образующих проверочное выражение в точке ветвления, принимало каждое из двух возможных логических значений, true и false, по крайней мере один раз.

1.2.4 Покрытие решений и условий

Согласно этому критерию набор тестов является достаточно полным, если удовлетворяются следующие требования: каждое условие в решении принимает каждое возможное значение по крайней мере один раз, каждый возможный исход решения проверяется по крайней мере один раз и каждой точке входа управление передается по крайней мере один раз.

1.2.5 Комбинаторное покрытие условий

Этот критерий требует создания такого количества тестов, при котором каждая возможная комбинация результатов вычисления условий в каждом решении и каждая точка входа проверяются по крайней мере один раз.

2 Программа №1

2.1 Описание программы

2.1.1 Постановка задачи

Название программы: Кодирование строк методом LZW.

Дано:

• String - строка, состоящая из символов ASCII.

Требуется: преобразовать строку в последовательность кодов, используя метод кодирования LZW (Lempel-Ziv-Welch). Вывести закодированную строку на экран.

Ограничения:

• Минимальная длина вводимой строки String: n - длина String, $1 \leq n$

• Максимальная длина вводимой строки String: n - длина String, $n \leq 10000$.

• Строка String состоит из символов ASCII: y - номер символа String[i] в таблице ASCII, $1 \leq y \leq 127$.

Дополнительно: Метод кодирования LZW преобразует входную строку в последовательность кодов, используя словарь символов. В процессе кодирования новые символьные последовательности добавляются в словарь.

2.1.2 Спецификация

Входные данные	Результат	Реакция программы
Строка, состоящая из более чем 10 000 символов.	Enter the string to encode: fghfg...hj \\строка состоящая из более чем 10 000 символов. Error: The input message is too long and will not be encoded! The maximum number of characters is 10 000.	Вывод на экран ошибки: “Error: The input message is too long and will not be encoded! The maximum number of characters is 10 000.” Завершение программы.
String=’привет!’	Enter the string to encode: привет! Error: Non-ASCII character detected! Please enter only ASCII characters.	Вывод на экран ошибки: “Error: Non-ASCII character detected! Please enter only ASCII characters.” Завершение программы.
String=’hi! привет!’	Enter the string to encode: hi! привет!	Вывод на экран ошибки: “Error: Non-ASCII character
	Error: Non-ASCII character detected! Please enter only ASCII characters.	detected! Please enter only ASCII characters.” Завершение программы.
String=”	Enter the string to encode: Error: the string mustn’t be empty	Вывод на экран ошибки: «Error: the string mustn’t be empty». Завершение программы
String=’Hello 1234!’	Enter the string to encode: Hello 1234! Encoded string: 72 101 108 108 111 32 49 50 51 52 33	Вывод закодированного сообщения. Завершение программы.

Рис. 3: Спецификация программы №1

2.1.3 Блок-схема

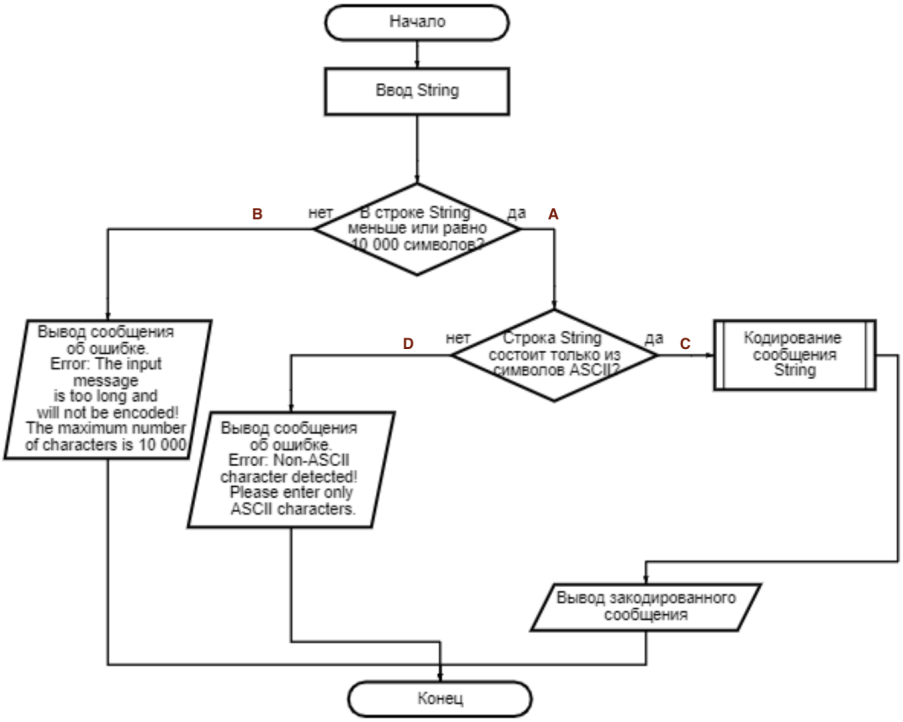

Рис. 4: Блок-схема программы №1

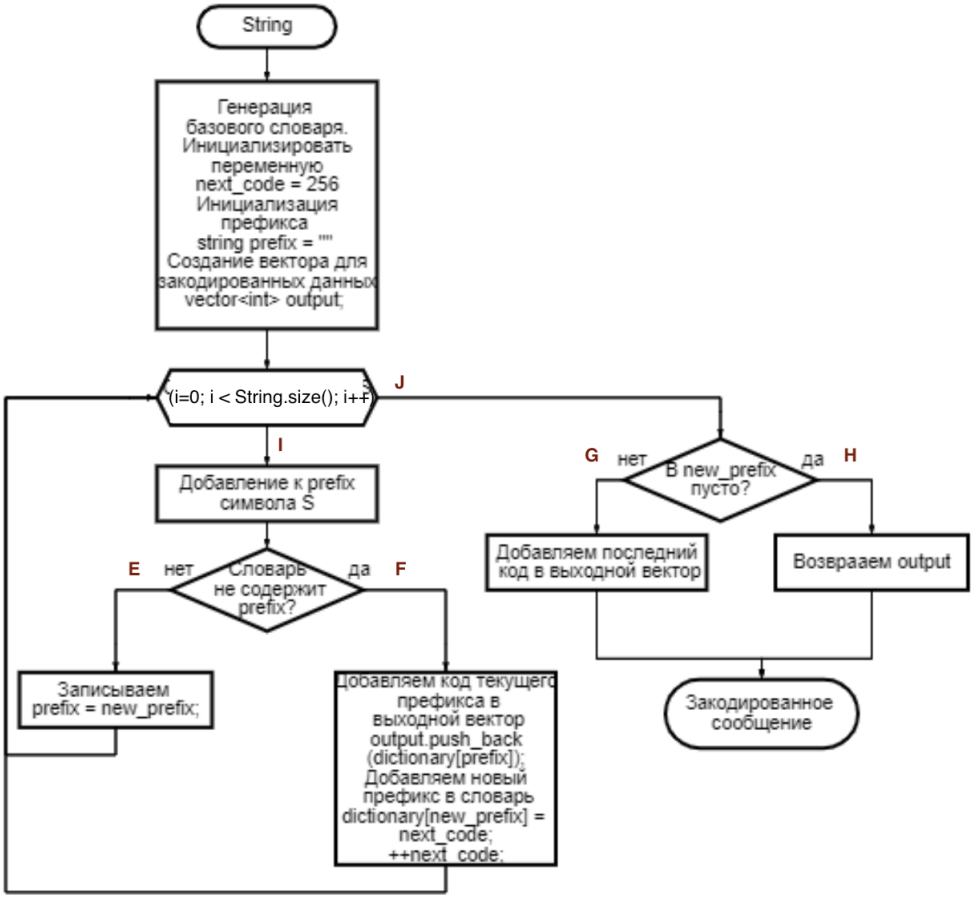

Рис. 5: Блок-схема программы №1

2.2 Тестирование методом черного ящика

2.2.1 Разбиение на классы эквивалентности

Составлено разбиение на классы эквивалентности исходя из ограничений для программы (n - длина введенной строки, $1 \leq n \leq 10000$, y - номер символа в таблице ASCII, $0 \leq y \leq 127$), представленное в таблице 1.

Входное условие	Допустимые классы	Недопустимые классы
n	$1 \leq n \leq 10000(1)$	$n > 10000(2), n < 1(3)$
y	$0 \leq y \leq 127(4)$	$y > 127(5)$

Таблица 1: Разбиение на классы эквивалентности

В таблице 2 приведены тесты для допустимых классов эквивалентности. В таблице 3 приведены тесты для недопустимых классов эквивалентности.

№	n	y	String	Классы эквивалентности	Ожидаемый результат	Фактический результат	Статус теста
1	13	{71, 111, 100, 32, 109, 114, 105, 103, 33}	'Good morning!'	(1), (4)	Encoded string: 71 111 111 100 32 109 111 114 110 105 110 103 33 Завершение программы	Encoded string: 71 111 111 100 32 109 111 114 110 105 110 103 33 Завершение программы	Пройден
2	15	{49, 50, 55, 32, 43, 256, 61, 53, 48}	'127 + 122 = 250'	(1), (4)	Encoded string: 49 50 55 32 43 32 256 50 32 61 32 50 53 48 Завершение программы	Encoded string: 49 50 55 32 43 32 256 50 32 61 32 50 53 48 Завершение программы	Пройден

3	22	{49, 55, 32, 110, 111, 118, 101, 109, 98, 114, 44, 50, 48, 52, 121, 97}	’17 november, 2004 year’	(1), (4)	Encoded string: 49 55 32 110 111 118 101 109 98 101 114 44 32 50 48 48 52 32 121 101 97 114 Завершение программы	Encoded string: 49 55 32 110 111 118 101 109 98 101 114 44 32 50 48 48 52 32 121 101 97 114 Завершение программы	Пройден

Таблица 2: Тесты, покрывающие все допустимые классы эквивалентности

№	n	y	String	Классы эквивалентности	Ожидаемый результат	Фактический результат	Статус теста
4	25675	100	’d’*25765	(2)	Error: The input message is too long and will not be encoded! The maximum number of characters is 10 000. Завершение программы.	Error: The input message is too long and will not be encoded! The maximum number of characters is 10 000. Завершение программы.	Пройден
5	0	-	”	(3)	Error: string mustn’t be empty Завершение программы	Encoded string: 0. Завершение программы.	Не пройден
6	6	{ 207, 240, 232, 226, 229, 242}	Привет	(5)	Error: Non-ASCII character detected! Please enter only ASCII characters. Завершение программы.	Error: Non-ASCII character detected! Please enter only ASCII characters. Завершение программы.	Пройден

Таблица 3: Тесты, покрывающие недопустимые классы эквивалентности

Выводы

При тестировании недопустимых классов эквивалентности был обнаружен непройденный тест №5. При вводе пустого символа программа должна выдать сообщение об ошибке и завершиться, но она выдаёт результат кодирования - 0 и успешно завершается.

2.2.2 Анализ граничных значений

Исходя из ограничений в программе, выделим следующие граничные значения для входных данных:

1. Верхнее граничное значение: $n = 10000$
2. Нижнее граничное значение: $n = 1$

Для каждой из границ определим тесты, соответствующие граничному числу (верхнему, нижнему) и числу, выходящему за границу (верхнюю, нижнюю) на единицу.

Тесты для проверки граничных условий представлены в таблице 4.

№	String	n	Ожидаемый результат	Фактический результат	Статус теста
1	'hello'*2000	10000	Encoded string: 104 101 108 108 111 (*2000) Завершение программы	Encoded string: 104 101 108 108 111 (*2000) Завершение программы	Пройден
2	'a'	1	Encoded string: 97 Завершение программы	Encoded string: 97 Завершение программы	Пройден
3	'a'*10001	10001	Error: The input message is too long and will not be encoded! The maximum number of characters is 10 000. Завершение программы	Error: The input message is too long and will not be encoded! The maximum number of characters is 10 000. Завершение программы	Пройден
4	''	0	Error: string mustn't be empty Завершение программы	Encoded string: 0. Завершение программы.	Не пройден

Таблица 4: Тесты для проверки граничных условий

При тестировании граничных условий был обнаружен непройденный тест №4. При вводе пустого символа программа должна выдать сообщение об ошибке и завершиться, но она выдаёт

результат кодирования - 0 и успешно завершается.

2.2.3 Причинно-следственная диаграмма

Причины:

1. $1 \leq n$
2. $n \leq 10000$
3. Все символы из таблицы ASCII ($0 \leq y \leq 127$)

Промежуточные причины:

11. $1 \leq n \leq 10000$

Следствия:

21. Программа выдает закодированную строку и завершается.

22. Программа выводит сообщение об ошибке: "Введенное значение слишком длинное, максимальное количество символов в строке - 10000"и завершается.

23. Программа выводит сообщение об ошибке: "Все символы строки должны быть из таблицы ASCII"и завершается.

24. Программа выведет сообщение об ошибке: "Введенное значение не может быть пустым"

Причинно-следственная диаграмма представлена на рис. 6.

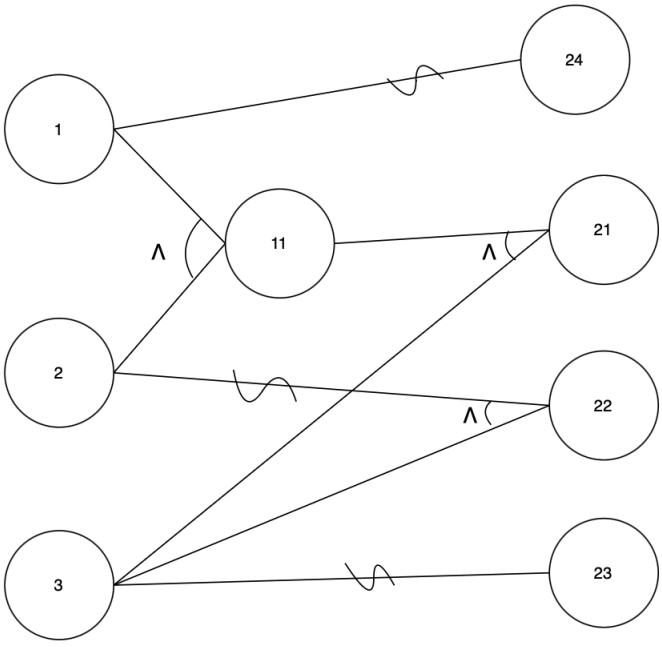

Рис. 6: Причинно-следственная диаграмма

Таблица решений на основе причинно-следственной диаграммы представлена в таблице 5.

	1	2	3	4
1	1	1	1	0
2	1	0	1	1
3	1	1	0	1
11	1	0	0	0
21	1	0	0	0
22	0	1	0	0
23	0	0	1	0
24	0	0	0	1

Таблица 5: Таблица решений на основе причинно-следственной диаграммы

Тесты на основе таблицы решений представлены в таблице 6.

№	Причины	Входные значения		Ожидаемый результат	Фактический результат	Статус теста
		String	n			
1	1, 2, 3	"1+1"	3	Encoded string: 49 43 49 Завершение программы	Encoded string: 49 43 49 Завершение программы	Пройден
2	1, 3	"b"*10001	10001	Error: The input message is too long and will not be encoded! The maximum number of characters is 10 000. Завершение программы	Error: The input message is too long and will not be encoded! The maximum number of characters is 10 000. Завершение программы	Пройден
3	1, 2	"Яблоко"	6	Error: Non-ASCII character detected! Please enter only ASCII characters. Завершение программы	Error: Non-ASCII character detected! Please enter only ASCII characters. Завершение программы	Пройден
4	2, 3	”	0	Error: string mustn’t be empty Завершение программы	Encoded string: 0. Завершение программы	Не пройден

Таблица 6: Тесты на основе таблицы решений

При тестировании причинно-следственных диаграмм был обнаружен непройденный тест №4. При вводе пустого символа программа должна выдать сообщение об ошибке и завершиться, но

она выдаёт результат кодирования - 0 и успешно завершается.

2.2.4 Результаты тестирования методом черного ящика

В результате тестирования методом «черного ящика» было составлено 14 тестов, из которых не пройдено 3 и пройдено 14. Ошибки, из-за которых не были пройдены тесты, связаны с некорректной проверкой длины входных значений: программа проверяет только верхнюю границу, поэтому при вводе пустой строки выдает результат кодирования, что не соответствует требованиям, описанным в спецификации.

2.3 Тестирование методом белого ящика

Алгоритм составления тестов методом «белого» ящика предполагает обход всех возможных путей в теле программы и проверку выполнения каждого оператора не менее одного раза. Для этого обозначим всевозможные пути, порожденные блоками условия, символами A-J.

Условия, указанные в ветвлениях программы:

1. n <= 10000
2. Входное значение состоит из символов таблицы ASCII ($0 \leq y \leq 127$)
3. Словарь содержит префикс?(dictionary.find(new_prefix) != dictionary.end())
4. i < String.size()
5. new_prefix == null

2.3.1 Покрытие операторов

Критерием покрытия является выполнение каждого оператора программы хотя бы раз. Это необходимое условие для приемлемого тестирования по принципу «белого» ящика.

Чтобы достичь покрытия всех операторов, предлагается воспользоваться тестом, описанным в таблице 7, который проходит по пути ACIEIFIFIFIFJH. Однако такой подход не учитывает множество возможных входных данных, что делает его недостаточно эффективным для поиска несоответствий спецификации.

№	String	n	y	Путь	Ожидаемый результат	Фактический результат	Статус теста
1	hello	5	{104, 101, 108, 108, 111}	ACIEIFIFIFIFJH	Encoded string: 104 101 108 108 111 Завершение программы	Encoded string: 104 101 108 108 111 Завершение программы	Пройден

Таблица 7: Покрытие операторов

2.3.2 Покрытие решений

В соответствии с этим критерием необходимо составить такое число тестов, при которых каждое условие в программе примет как истинное значение, так и ложное. Таким образом, к тестам, составленным для метода покрытия операторов, необходимо добавить тесты, которые будут проверять все возможные переходы.

Тесты, покрывающие все решения программы представлены в таблице 8.

№	String	n	y	Путь	Условия					Ожидаемый результат	Фактический результат	Статус теста
					1	2	3	4	5			
1	's'*20000	20000	115	B	F	-	-	-	-	Error: The input message is too long and will not be encoded! The maximum number of characters is 10 000. Завершение программы	Error: The input message is too long and will not be encoded! The maximum number of characters is 10 000. Завершение программы	Пройден

№	String	n	y	Путь	Условия					Ожидаемый результат	Фактический результат	Статус теста
					1	2	3	4	5			
2	’sд’	2	{100, 228}	AD	T	F	-	-	-	Error: Non-ASCII character detected! Please enter only ASCII characters. Завершение программы	Error: Non-ASCII character detected! Please enter only ASCII characters. Завершение программы	Пройден
3	’sd’	2	{100, 115}	ACIE IFJG	T	T	T/F	T/F	F	Encoded string: 115 100 Завершение программы	Encoded string: 115 100 Завершение программы	Пройден
4	’@’	1	64	ACI EJH	T	T	T/F	F	T	Encoded string: 64 Завершение программы	Encoded string: 64 Завершение программы	Пройден

Таблица 8: Покрытие решений

2.3.3 Покрытие условий

В соответствии с эттим критерием количество тестов должно быть таким, чтобы каждое из элементарных условий, образующих проверочное выражение в точке ветвления, принимало каждое из двух логических значений, true и false, по крайней мере один раз. Уже написанные тесты покрывают каждое условие хотя бы 1 раз.

2.3.4 Покрытие решений и условий

Согласно этому критерию набор тестов является достаточно полным, если удовлетворяются следующие требования: каждое условие в решении принимает каждое возможное значение по крайней мере один раз, каждый возможный исход решения проверяется по крайней мере один раз и каждой точке входа управление передается по крайней мере один раз.

Совокупность всех раннее написанных тестов дает покрытие условий и решений.

2.3.5 Комбинаторное покрытие условий

Этот критерий требует создания такого количества тестов, при котором каждая возможная комбинация результатов вычислений условий в каждом решении и каждая точка входа проверяются по крайней мере один раз.

Совокупность всех ранее написанных тестов дает комбинаторное покрытие условий.

2.3.6 Результаты тестирования методом «белого» ящика

В ходе тестирования методом белого ящика были разработаны 5 тестов, и все они были успешно пройдены. Это может одновременно указывать на корректность программы, а также на то, что разработанные тесты не учитывают какие-то слчаи, в которых могла бы проявиться ошибка, например, ошибки ввода пустой строки.

3 Программа №2

3.1 Описание программы

3.1.1 Постановка задачи

Название программы: сумма кубов чисел от 1 до N

Дано:

• N - целое число в десятичной системе счисления.

Требуется: найти сумму кубов целых чисел от 1 до заданного N и вывести ее на экран.

Ограничения:

• $1 \leq N \leq 77935$

3.1.2 Спецификация

Входные данные	Результат	Реакция программы
N = 21	53361	Вывод на экран суммы кубов чисел от 1 до N. Завершение программы.
N = -21	N is incorrect	Вывод на экран сообщения 'N is incorrect'. Завершение программы.
N = heck	Input is not a number	Вывод на экран сообщения 'Input is not a number'. Завершение программы.
N = 2.1	Input is not a number	Вывод на экран сообщения 'Input is not a number'. Завершение программы.
N = 77936	Overflow at i = 77936	Вывод на экран сообщения 'Overflow at i = 77936'. Завершение программы.

Рис. 7: Спецификация программы №2

3.1.3 Блок-схема

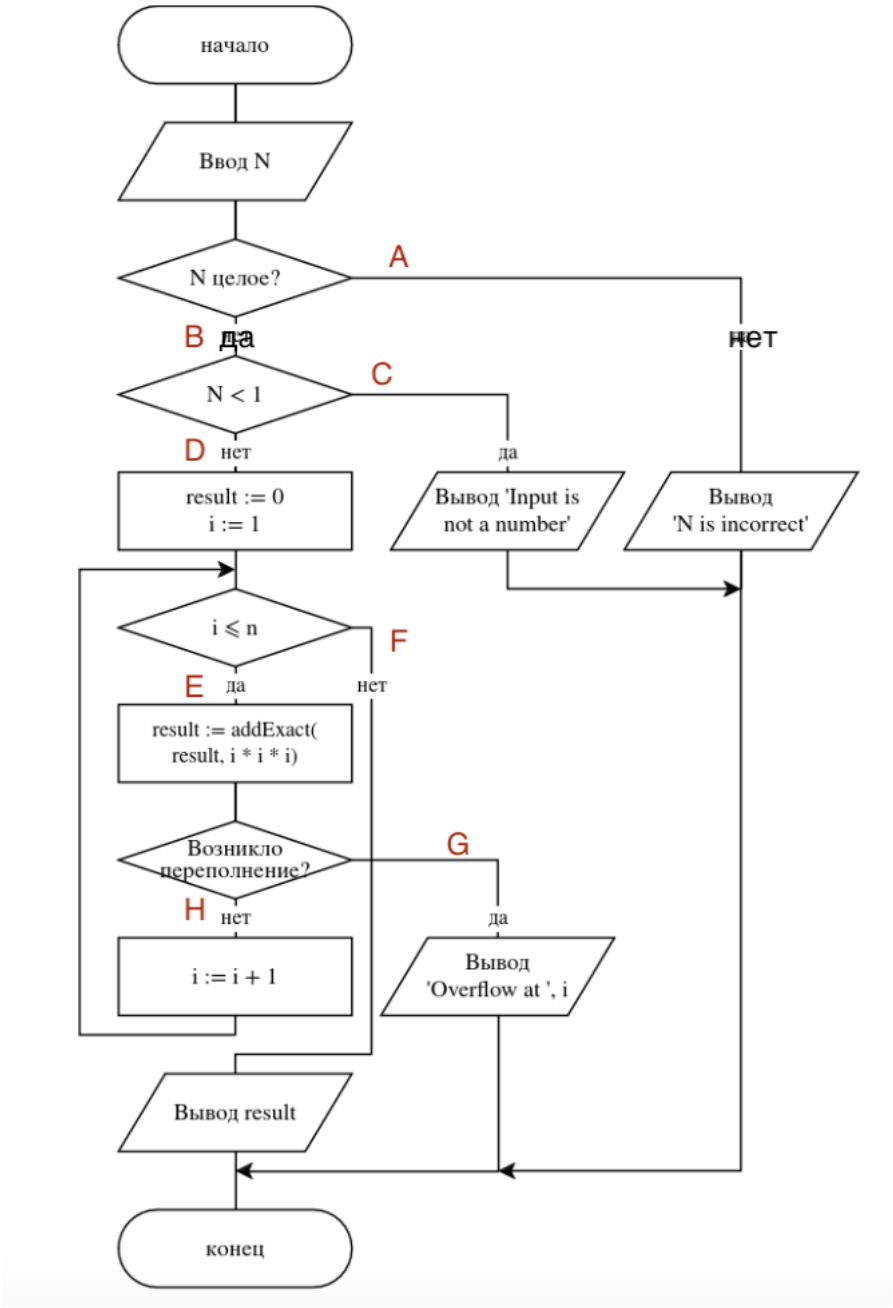

Рис. 8: Блок-схема программы №2

3.2 Тестирование методом черного ящика

3.2.1 Разбиение на классы эквивалентности

Составлено разбиаение на классы эквивалентности исходя из ограничений для программы ($1 \leq N \leq 77935$), представленное в таблице 9.

Входное условие	Допустимые классы	Недопустимые классы
N	$1 \leq N \leq 77935(1)$	$N > 77935(2), N < 1(3), N \notin Z(4)$

Таблица 9: Разбиение на классы эквивалентности

В таблице 10 приведены тесты для допустимых классов эквивалентности.

№	N	Классы эквивалентности	Ожидаемый результат	Фактический результат	Статус теста
1	5	(1)	"225"Завершение программы	"225"Завершение программы	Пройден
2	12	(1)	"6084"Завершение программы	"6084"Завершение программы	Пройден

Таблица 10: Тесты, покрывающие все допустимые классы эквивалентности

В таблице 11 приведены тесты для недопустимых классов эквивалентности.

№	N	Классы эквивалентности	Ожидаемый результат	Фактический результат	Статус теста
3	80000	(2)	"Overflow at i = 77936"Завершение программы	"Overflow at i = 77936"Завершение программы	Пройден
4	-10	(3)	"N is incorrect"Завершение программы	"N is incorrect"Завершение программы	Пройден
5	3.4	(4)	"Input is not a number"Завершение программы	"Input is not a number"Завершение программы	Пройден
6	abcd	(4)	"Input is not a number"Завершение программы	"Input is not a number"Завершение программы	Пройден

Таблица 11: Тесты, покрывающие недопустимые классы эквивалентности

3.2.2 Анализ граничных значений

Исходя из ограничений в программе, выделим следующие граничные значения для входных данных:

• Нижнее граничное значение: $N = 1$

• Верхнее граничное значение: $N = 77935$

Для каждой из границ определим тесты, соответствующие граничному числу (верхнему, нижнему) и числу, выходящему за границу (верхнюю, нижнюю) на единицу.

Тесты для проверки граничных условий представлены в таблице 12.

№	N	Ожидаемый результат	Фактический результат	Статус теста
1	1	"1"Завершение программы	"1"Завершение программы	Пройден
2	77935	"9223193340756366400"_ Завершение программы	"9223193340756366400"_ Завершение программы	Пройден
3	0	"N is incorrect"Завершение программы	"N is incorrect"Завершение программы	Пройден
4	77936	"Overflow at i = 77936"Завершение программы	"Overflow at i = 77936"Завершение программы	Пройден

Таблица 12: Тесты для проверки граничных условий

3.2.3 Причинно-следственная диаграмма

Причины:

1. $N \in Z$
2. $1 \leq N$
3. $N \leq 77935$

Промежуточные причины:

11. $1 \leq N \leq 77935$

Следствия:

21. Программа выдает результат вычисления и завершается

22. Программа выводит сообщение об ошибке: "Некорректное значение N"и завершается

23. Программа выводит сообщение об ошибке: "Введенное значение N не является числом"и завершается

24. Программа выводит сообщение об ошибке переполнения: "Переполнение при значении..."и завершается

Причинно-следственная диаграмма представлена на рис. 9. Таблица решений на основе причинно-следственной диаграммы представлена в таблице 13.

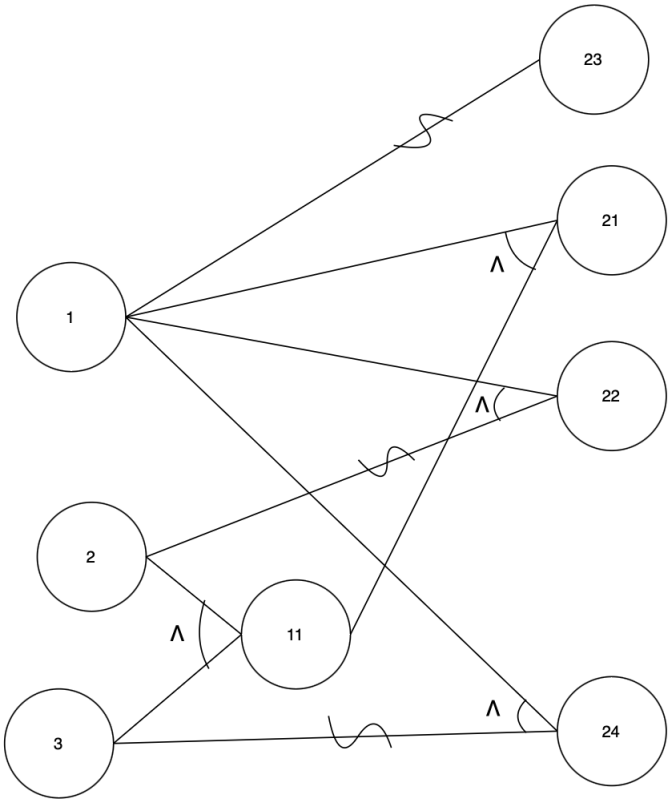

Рис. 9: Причинно-следственная диаграмма

	1	2	3	4
1	1	1	0	1
2	1	0	1	1
3	1	1	1	0
11	1	0	1	0
21	1	0	0	0
22	0	1	0	0
23	0	0	1	0
24	0	0	0	1

Таблица 13: Таблица решений на основе причинно-следственной диаграммы

Тесты на основе таблицы представлены в таблице 14.

№	Причины	Входные значения	Ожидаемый результат	Фактический результат	Статус теста
		N			
1	1, 2, 3	8	"1296"Завершение программы	"1296"Завершение программы	Пройден
2	1, 3	-10	"N is incorrect"Завершение программы	"N is incorrect"Завершение программы	Пройден
3	2, 3	5.5	"Input is not a number"Завершение программы	"Input is not a number"Завершение программы	Пройден
4	1, 2	80000	"Overflow at i = 77936"Завершение программы	"Overflow at i = 77936"Завершение программы	Пройден

Таблица 14: Тесты на основе таблицы решений

3.2.4 Результаты тестирования методом «черного» ящика

В результате тестирования методом «черного ящика» было составлено 14 тестов, из которых были пройдены все 14. Это может свидетельствовать как о корректности программы, так и о том, что разработанные тесты не рассматривают какую-либо ситуацию, в которой бы возникла ошибка.

3.3 Тестирование методом «белого» ящика

Алгоритм составления тестов методом «белого» ящика предполагает обход всех возможных путей в теле программы и проверку выполнения каждого оператора не менее одного раза. Для этого обозначим всевозможные пути, порожденные блоками условия, символами A-H.

Условия, указанные в ветвлениях программы:

1. $N \in Z$
2. N < 1
3. $i \leq N$
4. i > 77935

3.3.1 Покрытие операторов

Критерием покрытия является выполнение каждого оператора программы хотя бы раз. Это необходимое условие для приемлемого тестирования по принципу «белого» ящика.

Чтобы достичь покрытия всех операторов, предлагается воспользоваться тестом, описанным в таблице 15, который проходит по пути BDEHEHEHEHF. Однако такой подход не учитывает множество возможных входных данных, что делает его недостаточно эффективным для поиска несоответствий спецификации.

№	N	Путь	Ожидаемый результат	Фактический результат	Статус теста
1	4	BDEHE HEHEHF	"100"Завершение программы	"100"Завершение программы	Пройден

Таблица 15: Покрытие операторов

3.3.2 Покрытие решений

В соответствии с этим критерием необходимо составить такое число тестов, при которых каждое условие в программе примет как истинное значение, так и ложное. Таким образом, к тестам, составленным для метода покрытия операторов, необходимо добавить тесты, которые будут проверять все возможные переходы.

Тесты, покрывающие все решения программы представлены в таблице 16.

№	N	Путь	Условия				Ожидаемый результат	Фактический результат	Статус теста
			1	2	3	4			
1	3.5	A	F	-	-	-	"N is incorrect"_ Завершение программы	"N is incorrect"_ Завершение программы	Пройден
2	-5	BC	T	T	-	-	"Input is not a number"_ Завершение программы	"Input is not a number"_ Завершение программы	Пройден
3	2	BDEHEHF	T	F	T/F	F	"9"Завершение программы	"9"Завершение программы	Пройден

№	N	Путь	Условия				Ожидаемый результат	Фактический результат	Статус теста
			1	2	3	4			
4	80000	BD $(EH)^{77935}$ G	T	F	T/F	F/T	"Overflow at i = 77936"Завершение программы	"Overflow at i = 77936"Завершение программы	Пройден

Таблица 16: Покрытие решений

3.3.3 Покрытие условий

В соответствии с этим критерием количество тестов должно быть таким, чтобы каждое из элементарных условий, образующих проверочное выражение в точке ветвления, принимало каждое из двух логических значений, true и false, по крайней мере один раз. Уже написанные тесты покрывают каждое условие хотя бы 1 раз.

3.3.4 Покрытие решений и условий

Согласно этому критерию набор тестов является достаточно полным, если удовлетворяются следующие требования: каждое условие в решении принимает каждое возможное значение по крайней мере один раз, каждый возможный исход решения проверяется по крайней мере один раз и каждой точке входа управление передается по крайней мере один раз.

Совокупность всех раннее написанных тестов дает покрытие условий и решений.

3.3.5 Комбинаторное покрытие условий

Этот критерий требует создания такого количества тестов, при котором каждая возможная комбинация результатов вычислений условий в каждом решении и каждая точка входа проверяются по крайней мере один раз.

Совокупность всех ранее написанных тестов дает комбинаторное покрытие условий.

3.3.6 Результаты тестирования методом «белого» ящика

В ходе тестирования методом белого ящика были разработаны 5 тестов, и все они были успешно пройдены. Это может одновременно указывать на корректность программы, а также на то, что разработанные тесты не учитывают какие-то случаи, в которых могла бы проявиться ошибка, например, ошибки ввода пустой строки.

4 Заключение

В ходе данной лабораторной работы были изучены методы модульного тестирования - «белый ящик» и «черный ящик». Для двух программ были разработаны тесты с использованием обоих методов.

В результате тестирования программы №1 методом «черного ящика» было обнаружено 3 непройденных тестов из 14. Ошибки, из-за которых тесты не были пройдены, связаны с некорректной проверкой длины входных значений: программа проверяет только верхнюю границу, поэтому при вводе пустой строки выдает результат кодирования, что не соответствует требованиям, уписанным в спецификации. При тестировании методом «белого ящика» все 5 тестов были успешно пройдены, что свидетельствует о том, что разработанные тесты не рассматривают какую-либо ситуацию, в которой бы возникла ошибка. Таким образом, при тестировании программы №1 разработанные тесты для метода «черного ящика» оказались лучше.

В результате тестирования программы №2 методом «черного ящика» были успешно пройдены все 14 тестов, а при тестировании методом «белого ящика» - успешно пройдены все 5 тестов. Такие результаты могут свидетельствовать как о корректности программы, так и о том, что разработанные тесты не рассматривают какую-либо ситуацию, в которой бы возникла ошибка.

Список литературы

[1] Майерс, Г. Искусство тестирования программ / Г. Майерс, Т. Баджетт, К. Санд-лер. — Изд. 3-е. — Санкт-Петербург : Диалектика, 2012. — 272 с.