МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО
ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
**"САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО"**
Институт компьютерных наук и технологий
Направление **02.03.01** : Математика и компьютерные науки

# ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ «Инспекция кода»

Исполнитель: _____ Яшнова Дарья Михайловна
группа 5130201/20002

Руководитель: _____ Курочкин Михаил Александрович

«___»_____ 2025г

Санкт-Петербург, 2025

# Содержание

**Введение** **3**

**1 Общие сведения о методах тестирования «белого» и «черного ящика»** **4**
1.1 Метод «черного ящика» . . . 4
1.2 Метод «белого ящика» . . . 5

**2 Тестирование методом «черного ящика»** **7**
2.1 Описание программы . . . 7
2.1.1 Постановка задачи . . . 7
2.1.2 Спецификация . . . 7
2.1.3 Блок-схема . . . 7
2.2 Классы эквивалентности . . . 9
2.3 Анализ граничных значений . . . 10
2.4 Причинно-следственная диаграмма . . . 13
2.5 Результаты тестирования методом черного ящика . . . 15

**3 Тестирование методом белого ящика** **16**
3.1 Описание программы . . . 16
3.2 Постановка задачи . . . 16
3.2.1 Спецификация . . . 16
3.2.2 Блок-схема . . . 17
3.3 Покрытие операторов . . . 17
3.4 Покрытие операторов . . . 18
3.5 Покрытие решений . . . 18
3.6 Покрытие условий . . . 18
3.7 Результаты тестирования методом белого ящика . . . 18

**4 Заключение** **19**

**Список литературы** **20**

## Введение

В рамках данной работы требуется провести исследование и практическое применение методологий тестирования «черного ящика» и «белого ящика». Для этого необходимо протестировать две программы, разработанные другими программистами:

Программа №1: Протестировать методом "черного ящика".

Программа №2: Протестировать методом "белого ящика".

# 1 Общие сведения о методах тестирования «белого» и «черного ящика»

## 1.1 Метод «черного ящика»

Проектирование тестов методом черного ящика базируется на анализе спецификации программы, то есть на понимании того, как она должна функционировать. Поскольку проверить все возможные комбинации входных данных невозможно, используются определённые подходы, которые позволяют охватить как можно больше различных сценариев. Это, в свою очередь, помогает выявить большее количество ошибок. Основными техниками являются:

1. Классы эквивалентности

   Для каждого из входных условий выделяются группы, называемые классами эквивалентности. Эти группы делятся на:

   - Допустимые классы эквивалентности — значения, которые программа должна корректно обработать.

   - Недопустимые классы эквивалентности — значения, которые должны вызывать ошибки или некорректное поведение.

   После определения классов составляются тесты:

   - Тесты, которые проверяют как можно больше различных допустимых классов.

   - Тесты, каждый из которых проверяет один недопустимый класс.

2. Граничные значения

   Для каждого входного условия определяются крайние значения диапазонов, как допустимые, так и недопустимые. Далее разрабатываются тесты:

   - Проверяющие корректность работы программы на границах допустимых значений.

   - Проверяющие поведение программы на недопустимых граничных значениях.

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. Разбить спецификацию: Разделить сложную спецификацию на более мелкие, управляемые части.

   2. Определить причины и следствия: Причины – это входные условия или классы эквивалентности. Следствия – это выходные условия или преобразования системы. Причины и следствия нумеруются.

   3. Построить причинно-следственную диаграмму: Создать булев граф, связывающий причины и следствия, отражая семантическое содержание спецификации.

   4. Добавить ограничения: Задать ограничения на комбинации причин и/или следствий, которые невозможны.

5. Преобразовать в таблицу решений: Преобразовать диаграмму в таблицу решений с ограниченными входами. Каждый столбец таблицы представляет собой тест.

6. Преобразовать в тесты: Преобразовать столбцы таблицы решений в конкретные тестовые случаи.

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”.

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

- Функция not устанавливает, что если a равно 1, то b равно 0; в против-ном 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.

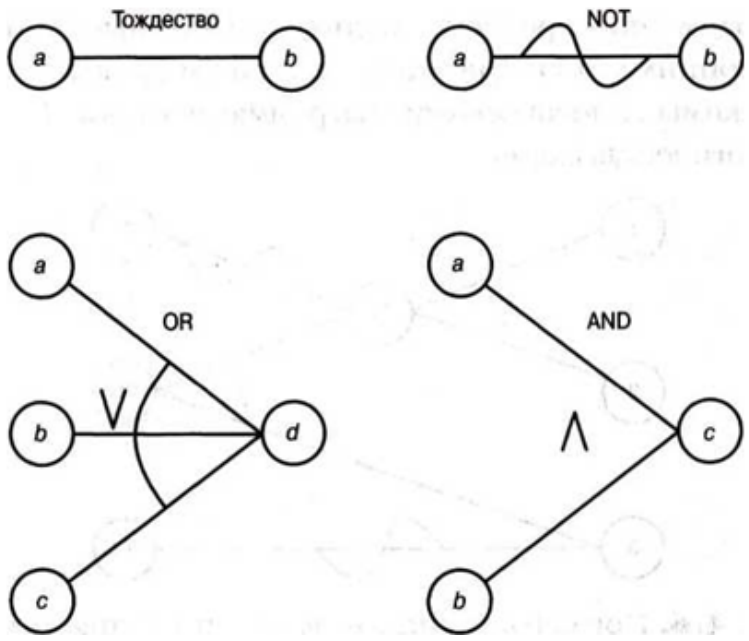


Рис. 1: Базовая нотация причинно-следственных диаграмм

Таким образом, применение этих техник позволяет создать набор тестов, который максимально полно охватывает возможные сценарии использования программы, увеличивая вероятность обнаружения ошибок.

## 1.2 Метод «белого ящика»

Проектирование тестов методом белого ящика основывается на детальном знании логики работы программы, что позволяет использовать её структуру, представленную в виде блок-схемы. Задача тестирования в этом подходе заключается в том, чтобы охватить тестами все возможные пути выполнения программы. Однако на практике это практически невозможно из-за сложности программ, поэтому применяются различные критерии, которые помогают систематизировать тестирование. Основные из них:

1. Покрытие операторов

   Тесты создаются так, чтобы каждый оператор программы исполнялся хотя бы один раз. Это минимальный и самый простой критерий, но его недостатком является слабая эффективность в обнаружении сложных ошибок.

2. Покрытие решений (ветвей)

   Тесты разрабатываются таким образом, чтобы каждая ветвь условного оператора исполнялась хотя бы один раз. Этот подход лучше, чем покрытие операторов, но не всегда достаточен, особенно если в программе используются составные условия.

3. Покрытие условий

   Каждое условие внутри условного оператора должно принимать все возможные значения (например, true и false) хотя бы один раз. Однако этот критерий не гарантирует, что все ветви программы будут выполнены.

4. Покрытие решений и условий

   Этот подход объединяет покрытие ветвей и покрытие условий. Он требует, чтобы:

   - Каждая ветвь условного оператора была выполнена хотя бы раз.

   - Все условия внутри операторов принимали все возможные значения.

   Проблема этого подхода заключается в том, что некоторые условия могут «маскировать» друг друга, и ошибки в таких случаях могут остаться незамеченными.

5. Комбинаторное покрытие условий

   Тесты составляются так, чтобы каждая возможная комбинация значений всех условий была проверена хотя бы один раз. Этот критерий является самым мощным, так как позволяет выявить максимум ошибок. Однако его основной недостаток — экспоненциальный рост числа тестов по мере увеличения количества условий.

# 2 Тестирование методом «черного ящика»

## 2.1 Описание программы

### 2.1.1 Постановка задачи

Автор: Черепнов Максим

Название программы: поиск максимальной суммы подмассива одномерного массива алгоритмом Кадане.

Дано:

1. n – положительное целое число, размер входного массива
2. arr – массив целых чисел

Требуется: вычислить максимальную сумму maxSum подмассива массива arr и индексы start и end первого и последнего значения подмассива.

Ограничения:

1. 1 <= n <= 1000
2. -2000 <= arr[i] <= 2000, где 0 <= i <= n - 1

### 2.1.2 Спецификация

| № | Входные данные | Выходные данные | Реакция программы |
|---|---|---|---|
| 1 | n = 9<br>arr = {-2, 1, -3, 4, -1, 2, 1, -5 , 4} | maxSum = 6, start = 3, end = 6. | Вывод на экран maxSum, start, end. Завершение работы программы |
| 2 | n = 5<br>arr = {1, 2, 3, 4, 5} | maxSum = 15, start = 0, end = 4. | Вывод на экран maxSum, start, end. Завершение работы программы |
| 3 | n = 4<br>arr = {-3, -2, -1, -4} | maxSum = -1, start = 2, end = 2. | Вывод на экран maxSum, start, end. Завершение работы программы |
| 4 | n < 1 или n > 1000 | Размер массива должен быть больше 0 и меньше 1001! | Вывод сообщения об ошибке: «Размер массива должен быть больше 0 и меньше 1001!». Возврат к вводу n |
| 5 | arr[i] < -2000 или arr[i] > 2000 | В массиве должны храниться числа, по модулю меньшие, чем 2001! | Вывод сообщения об ошибке: «В массиве должны храниться числа, по модулю меньшие, чем 2001!». Завершение работы программы |

Рис. 2: Спецификация программы

### 2.1.3 Блок-схема

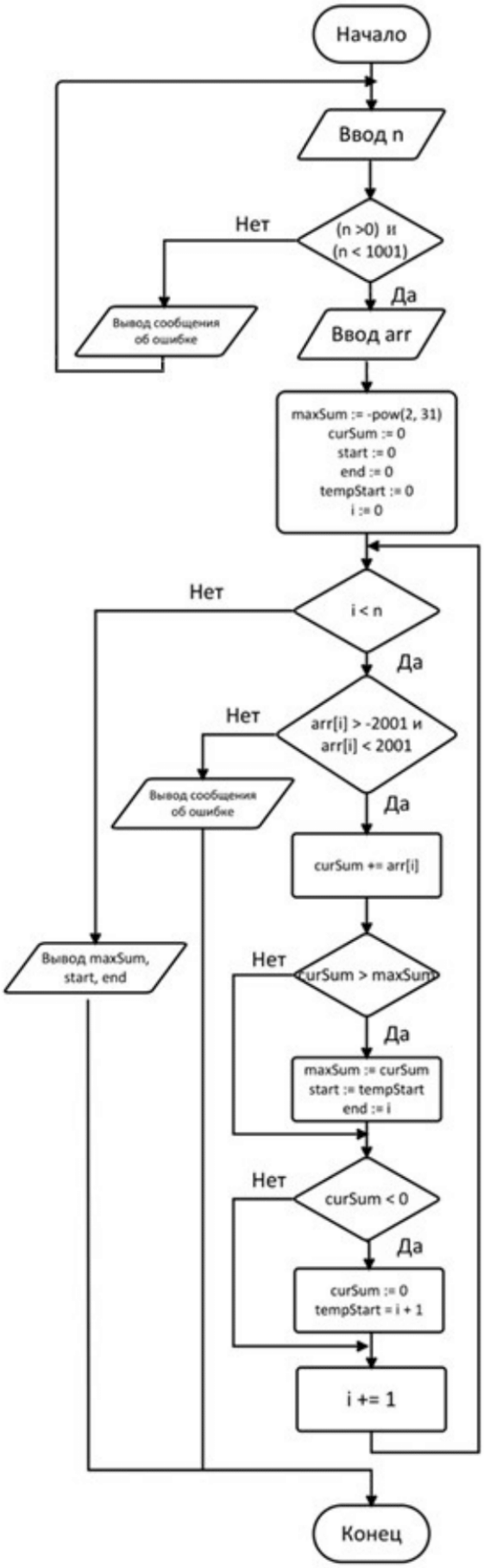


Рис. 3: Блок-схема

## 2.2 Классы эквивалентности

Входных условий в программе 2 - arr и n, где n - размер arr[i]. Исходя из ограничений для входных условий было проведено разбиение на классы эквивалентности.

Классы эквивалентности для n

| Входное условие | Допустимые классы | Недопустимые классы |
|---|---|---|
| n | 1 <= n <= 1000 (1) | $n \notin Z(2)$, n<1(3), n>1000(4) |

Классы эквивалентности для arr[i]

| Входное условие | Допустимые классы | Недопустимые классы |
|---|---|---|
| Элементы массива | -2000 <= arr[i] <= 2000 (5) | $arr[i] \notin Z(6)$, arr[i]<-2000(7), arr[i]>2000(8) |

В таблице 3 приведены тесты покрывающие все классы эквивалентности.

Таблица 3: Тесты, покрывающие все классы эквивалентности

| № | n | arr | Классы эквивалентности | Ожидаемый результат | Фактический результат | Результат тестирования |
|---|---|---|---|---|---|---|
| 1 | 5 | {1, 2, 3, 4, 5} | (1), (5) | maxSum = 15, start = 0, end = 4 | maxSum = 15, start = 0, end = 4 | Пройден |
| 2 | bb | {1, 2, 3, 4, 5} | (2) | «Размер массива должен быть целым числом больше 0 и меньше 1001», возврат к вводу n | «Размер массива должен быть целым числом больше 0 и меньше 1001», возврат к вводу n | Пройден |
| 3 | 0 | {1, 2, 3, 4, 5} | (3) | «Размер массива должен быть целым числом больше 0 и меньше 1001», возврат к вводу n | «Размер массива должен быть целым числом больше 0 и меньше 1001», возврат к вводу n | Пройден |
| 4 | 1001 | {1, 2, 3, 4, 5} | (4) | «Размер массива должен быть целым числом больше 0 и меньше 1001», возврат к вводу n | «Размер массива должен быть целым числом больше 0 и меньше 1001», возврат к вводу n | Пройден |

| | | | | | | |
|---|---|---|---|---|---|---|
| 5 | 5 | {1, 2, «bb», 4, 5} | (6) | «В массиве должны храниться числа, по модулю меньшие 2001», Останов программы | maxSum = 12, start = 0, end = 4 | Не пройдено |
| 6 | 5 | {1, 2, -2001, 4, 5} | (7) | «В массиве должны храниться числа, по модулю меньшие 2001». Останов программы | «В массиве должны храниться числа, по модулю меньшие 2001». Останов программы | Пройден |
| 7 | 5 | {1, 2, 2001, 4, 5} | (8) | «В массиве должны храниться числа, по модулю меньшие 2001». Останов программы | «В массиве должны храниться числа, по модулю меньшие 2001». Останов программы | Пройден |

***Выводы***

При тестировании на данных из различных классов эквивалентности был обнаружен один непройденный тест - №5.

№5 : При вводе значения «bb» в качестве arr[i] программа должна остановиться, так как было введено недопустимое значение, однако ввод приводится к целому числу, arr[i] становится равным 0, программа успешно завершается и выводит результат.

## 2.3 Анализ граничных значений

Для входных данных определены следующие граничные значения:

1. n<=1000
2. n>=1
3. arr[i]>=-2000

4.arr[i]<=2000

Для каждой из границ определим тесты, соответствующие:

- граничному целому числу (верхнему/нижнему);
- целому числу, выходящему за границу (верхнюю/нижнюю) на единицу;
- дробному числу, на 0.0001 выходящему за границу (верхнюю/нижнюю). Тесты для проверки граничных условий приведены в таблице 4.

Таблица 4: Проверка граничных условий

| № | n | arr | Ожидаемый результат | Фактический результат | Результат тестирования |
|---|---|---|---|---|---|
| 1 | 0 | - | «Размер массива должен быть целым числом больше 0 и меньше 1001», возврат к вводу n | «Размер массива должен быть целым числом больше 0 и меньше 1001», возврат к вводу n | Пройден |
| 2 | 1 | {-2001} | «В массиве должны храниться числа, по модулю меньшие 2001». Останов программы | «В массиве должны храниться числа, по модулю меньшие 2001». Останов программы | Пройден |
| 3 | 1 | {-2000} | «maxSum = -2000, start = 0, end = 0», Останов программы | «maxSum = -2000, start = 0, end = 0», Останов программы | Пройден |
| 4 | 1 | {2000} | «maxSum = 2000, start = 0, end = 0», Останов программы | «maxSum = 2000, start = 0, end = 0», Останов программы | Пройден |
| 5 | 1 | {2001} | «В массиве должны храниться числа, по модулю меньшие 2001». Останов программы | «В массиве должны храниться числа, по модулю меньшие 2001». Останов программы | Пройден |
| 6 | 1000 | {-2000,...,-2000}, size=1000 | «maxSum = -2 000, start = 999, end = 999», Останов программы | «maxSum = -2 000, start = 999, end = 999», Останов программы | Пройден |
| 7 | 1000 | {-2001,..., -2001}, size=1000 | «В массиве должны храниться числа, по модулю меньшие 2001». Останов программы | «В массиве должны храниться числа, по модулю меньшие 2001». Останов программы | Пройден |
| 8 | 1000 | {2000,...,2000}, size=1000 | «maxSum = 2 000 000, start = 0, end = 999», Останов программы | «maxSum = 2 000 000, start = 0, end = 999», Останов программы | Пройден |

| | | | | | |
|---|---|---|---|---|---|
| 9 | 1000 | {2001,..., 2001}, size=1000 | «В массиве должны храниться числа, по модулю меньшие 2001». Останов программы | «В массиве должны храниться числа, по модулю меньшие 2001». Останов программы | Пройден |
| 10 | 1001 | - | «Размер массива должен быть целым числом больше 0 и меньше 1001», возврат к вводу n | «Размер массива должен быть целым числом больше 0 и меньше 1001», возврат к вводу n | Пройден |
| 11 | 1.0001 | {1} | «Размер массива должен быть целым числом больше 0 и меньше 1001», возврат к вводу n | maxSum = 1, start = 0, end = 0 | Не пройден |
| 12 | 0.9999 | - | «Размер массива должен быть целым числом больше 0 и меньше 1001», возврат к вводу n | «Размер массива должен быть целым числом больше 0 и меньше 1001», возврат к вводу n | Пройден |
| 13 | 1000.0001 | {2000,...,2000}, size = 1000 | «Размер массива должен быть целым числом больше 0 и меньше 1001», возврат к вводу n | «maxSum = 2 000 000, start = 0, end = 999», Останов программы | Не пройден |
| 14 | 999.9999 | {2000,...,2000}, size = 999 | «Размер массива должен быть целым числом больше 0 и меньше 1001» | «maxSum = 1998000, start = 0, end = 998», Останов программы | Не пройден |

| | | | | | |
|---|---|---|---|---|---|
| 15 | 1 | {2000.0001} | «maxSum = 2000, start = 0, end = 0», Останов программы | «В массиве должны храниться числа, по модулю меньшие 2001». Останов программы | Не пройден |
| 16 | 1 | {1999.9999} | «maxSum = 1999, start = 0, end = 0», Останов программы | «В массиве должны храниться числа, по модулю меньшие 2001». Останов программы | Не пройден |
| 17 | 1 | {-1999.9999} | «maxSum = -1999, start = 0, end = 0», Останов программы | «В массиве должны храниться числа, по модулю меньшие 2001». Останов программы | Не пройден |
| 18 | 1 | {-2000.0001} | «maxSum = -2000, start = 0, end = 0», Останов программы | «В массиве должны храниться числа, по модулю меньшие 2001». Останов программы | Не пройден |

***Выводы***

При тестировании граничных условий были обнаружены 7 непройденных тестов - №11, №13, №14, №15, №16, №17, №18.

№11 : При вводе значения 1.0001 для n программа должна вернуться к вводу n, однако она сокращает вещественное число до целого 1, успешно завершается и выводит результат.

№13 : При вводе значения 1000.1001 для n программа должна вернуться к вводу n, однако она сокращает вещественное число до целого 1000, успешно завершается и выводит результат.

№14 : При вводе значения 999.9999 для n программа должна вернуться к вводу n, однако она сокращает вещественное число до целого 999, успешно завершается и выводит результат.

№14-18 : При вводе одного значений 2000.0001, 1999.9999, -2000.0001, -1999.9999 для n программа должна завершиться , так как вводится нецелое число, однако она сокращает вещественное число до целого, успешно завершается и выводит результат.

## 2.4 Причинно-следственная диаграмма

Для причинно-следственной диаграммы введем следующие обозначения:

1: n — целое;

2: 1 <= n <= 1000;

3: все arr[i] — целое;

4: для всех arr[i] -2000 <= arr[i] <= 2000;

5: Программа вычисляет максимальную сумму подмассива;

6: Программа выводит сообщение об ошибке;

11: n — целое и 1 <= n <= 1000 и все arr[i] — целое идля всех arr[i] -2000 <= arr[i] <= 2000;

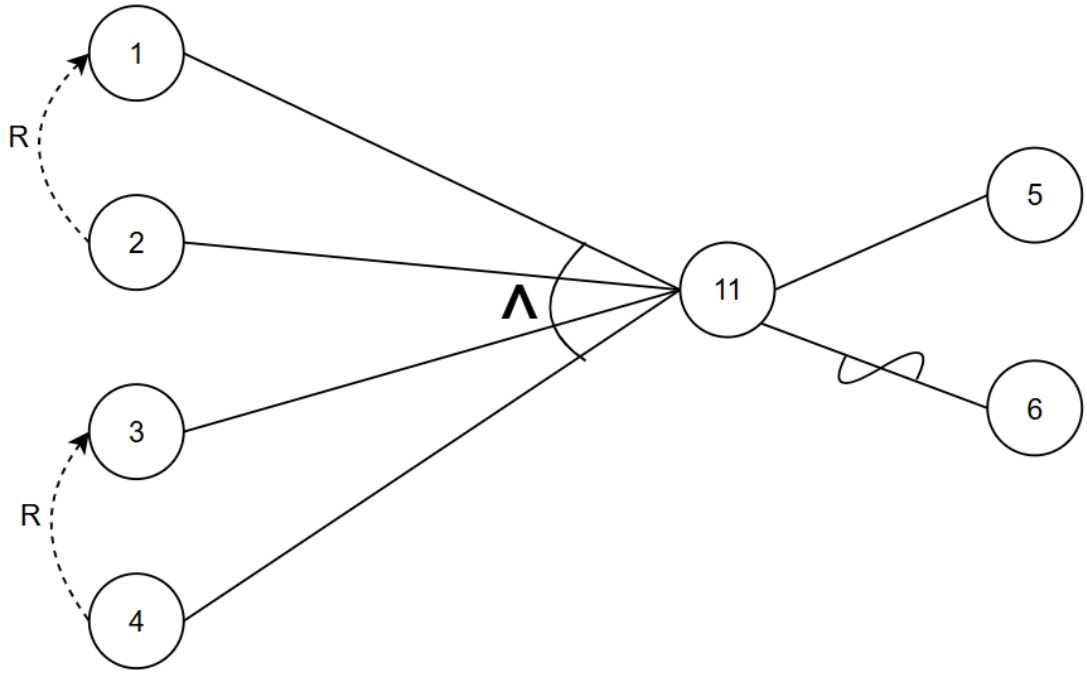


Рис. 4: Причинно-следственная диаграмма

Таблица решений для диаграммы представлена в таблице 5

Таблица 5: Таблица решений для диаграммы

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 1 | 0 | 1 | 1 | 1 |
| 2 | 1 | 1 | 0 | 1 | 1 |
| 3 | 1 | 1 | 1 | 0 | 1 |
| 4 | 1 | 1 | 1 | 1 | 0 |
| 11 | 1 | 0 | 0 | 0 | 0 |
| 5 | 1 | 0 | 0 | 0 | 0 |
| 6 | 0 | 1 | 1 | 1 | 1 |

Таблица 6:Тесты на основе таблицы решений

| № | n | arr | Ожидаемый результат | Фактический результат | Результат тестирования |
|---|---|---|---|---|---|
| 1 | 5 | {1, 2, 3, 4, 5} | maxSum = 15, start = 0, end = 4 | maxSum = 15, start = 0, end = 4 | Пройден |
| 2 | 999.9999 | {2000,...,2000}, size = 999 | «Размер массива должен быть целым числом больше 0 и меньше 1001» | «maxSum = 1998000, start = 0, end = 998», Останов программы | Не пройден |
| 3 | 1001 | - | «Размер массива должен быть целым числом больше 0 и меньше 1001», возврат к вводу n | «Размер массива должен быть целым числом больше 0 и меньше 1001», возврат к вводу n | Пройден |
| 4 | 5 | {1, 2, «bb», 4, 5} | «В массиве должны храниться числа, по модулю меньшие 2001», Останов программы | maxSum = 12, start = 0, end = 4 | Не пройдено |
| 5 | 1 | {2001} | «В массиве должны храниться числа, по модулю меньшие 2001». Останов программы | «В массиве должны храниться числа, по модулю меньшие 2001». Останов программы | Пройден |

***Выводы***

При тестировании методом причинно-следственных диаграмм было обнаружено два непройденных теста №2, №4.

№2: При вводе значения 999.9999 для n программа должна вернуться к вводу n, однако она сокращает вещественное число до целого 999, успешно завершается и выводит результат.

№4: При вводе значения «bb» в качестве arr[i] программа должна остановиться, так как было введено недопустимое значение, однако ввод приводится к целому числу, arr[i] становится равным 0, программа успешно завершается и выводит результат.

## 2.5 Результаты тестирования методом черного ящика

В результате тестирования методом черного ящика было составлено 30 тестов, из которых было не пройдено 10 и пройдено 20. Ошибки, из-за которых тесты не были пройдены, связаны с отсутствием проверки входных значений: программа выдает численный результат как на дробные входные данные, так и на строковые, что не соответствует требованиям, описанным в задаче.

# 3 Тестирование методом белого ящика

## 3.1 Описание программы

## 3.2 Постановка задачи

Автор: Емешкин Максим

Название программы: транспонирование матрицы.

Входные данные:

1. Число строк матрицы A — M
2. Число столбцов матрицы A — N.
3. Матрица A размером M x N.

Требуется: вычислить транспонированную матрицу $A\hat{T}$ и вывести её.

Ограничения

1. M>0 и N>0 (число строк и столбцов должно быть положительным).
2. Элементы матрицы должны быть целыми числами.

### 3.2.1 Спецификация

| Входные значения | Выходные значения | Реакция программы |
|---|---|---|
| A = {{1, 2, 3}, {4, 5, 6}}, M = 2, N = 3 | A^T = {{1, 4}, {2, 5}, {3, 6}} Останов программы | Транспонированная матрица A^T будет иметь размер N x M. |
| A = {{1, 2}, {3, 4}, {5, 6}}, M = 3, N = 2 | A^T = {{1, 3, 5}, {2, 4, 6}} Останов программы | Транспонированная матрица A^T будет иметь размер N x M. |
| Если M <= 0 | Вывод: "Ошибка ввода M" Возврат к вводу M | Число строк и столбцов в матрице должно быть больше 0. |
| Если N <= 0 | Вывод: "Ошибка ввода N" Возврат к вводу N | Число строк и столбцов в матрице должно быть больше 0. |

Рис. 5: Спецификация программы №2

### 3.2.2 Блок-схема

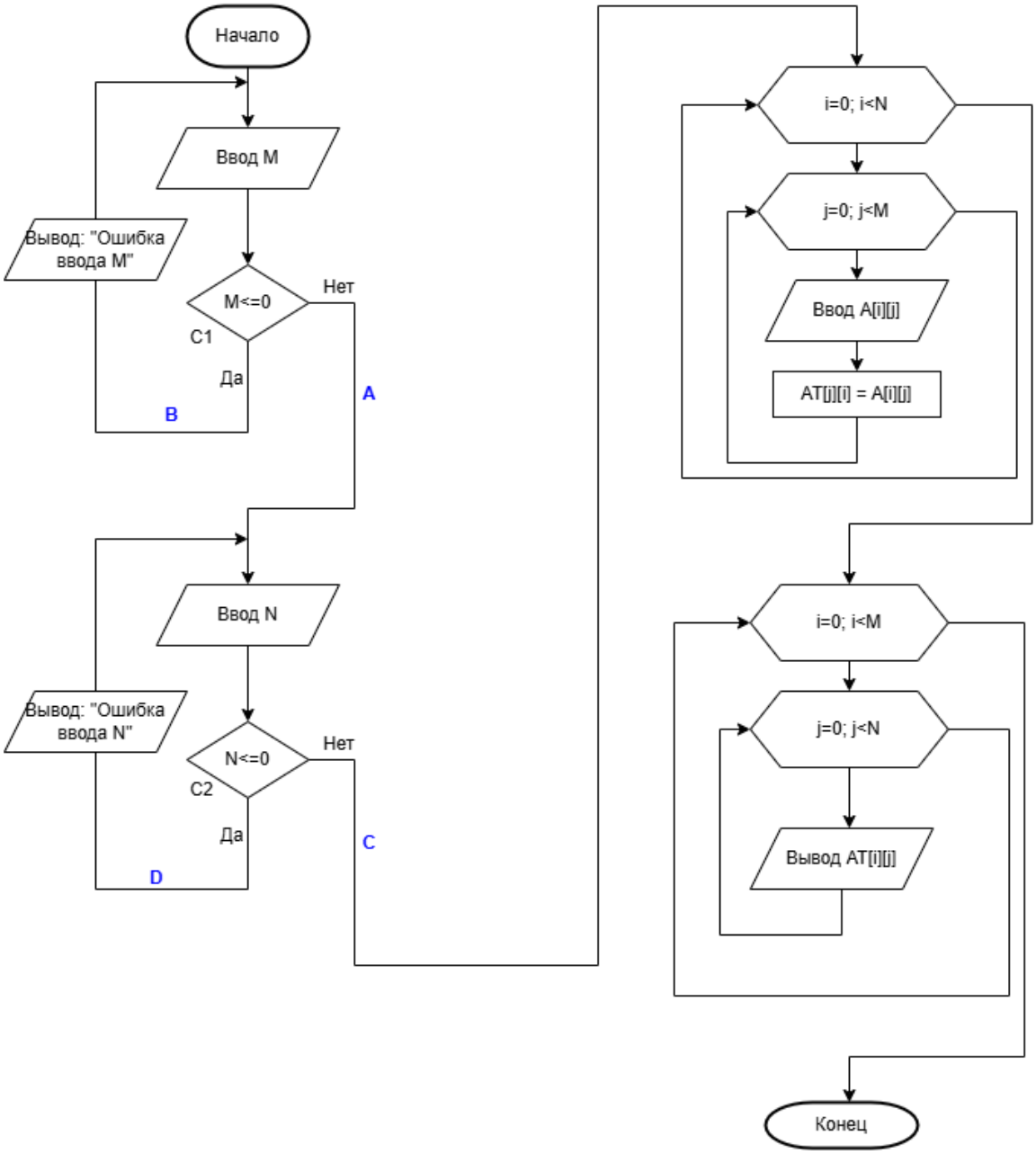


Рис. 6: Блок-схема программы №2

## 3.3 Покрытие операторов

Алгоритм тестирования по методу «белого» ящика подразумевает, что для каждой ветви в программе нужно пройти все потенциальные пути, проверив при этом каждый оператор хотя бы один раз. С этой целью присвоим всем возможным ветвям, возникающим из условных блоков, обозначения A, B, C и D.

В программе имеется 2 условных оператора (2, 4 на рис.2)

1)C1: $M \leq 0$

2)C2: $N \leq 0$

## 3.4 Покрытие операторов

В качестве критерия покрытия выступает выполнение каждого оператора программы как минимум один раз. Хотя это условие и необходимо, оно не является достаточным для полноценного тестирования по принципу белого ящика. Чтобы достичь покрытия всех операторов, предлагается воспользоваться тестом, описанным в таблице 7, который проходит по пути AC. Однако такой подход не учитывает множество возможных входных данных, что делает его недостаточно эффективным для поиска несоответствий спецификации.

Таблица 7: Покрытие операторов

| № | M | N | A | Путь | C1 | C2 | Ожидаемый результат | Фактический результат | Статус теста |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 2 | {{1, 2}, {3, 4}} | AC | 0 | 0 | {{1, 3}, {2, 4}} | {{1, 3}, {2, 4}} | Пройден |

## 3.5 Покрытие решений

Согласно этому критерию требуется сформировать такой набор тестов, чтобы каждое условие в программе могло принимать и истинное, и ложное значения. Следовательно, помимо тестирования, ориентированного на покрытие всех операторов, необходимо добавить тесты, охватывающие все возможные ветвления. Тесты, демонстрирующие полное покрытие решений программы, представлены в таблице 8.

Таблица 8: Покрытие решений

| № | M | N | A | Путь | C1 | C2 | Ожидаемый результат | Фактический результат | Статус теста |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 2 | {{1, 2}, {3, 4}} | AC | 0 | 0 | {{1, 3}, {2, 4}} | {{1, 3}, {2, 4}} | Пройден |
| 2 | 2 | -100 | - | AD | 0 | 1 | «Ошибка ввода N» | «Ошибка ввода N» | Пройден |
| 3 | -100 | 2 | - | B | 1 | - | «Ошибка ввода M» | «Ошибка ввода M» | Пройден |

## 3.6 Покрытие условий

Согласно этому критерию, необходимо составить столько тестов, чтобы в точке ветвления каждое из входящих в проверочное выражение элементарных условий хотя бы один раз принимало значения true и false.

Таблица 9: Покрытие условий

| № | M | N | A | Путь | C1 | C2 | Ожидаемый результат | Фактический результат | Статус теста |
|---|---|---|---|---|---|---|---|---|---|
| 1 | -2 | 2 | - | B | 1 | - | «Ошибка ввода M» | «Ошибка ввода M» | Пройден |
| 2 | 2 | -2 | - | AD | 0 | 1 | «Ошибка ввода N» | «Ошибка ввода N» | Пройден |
| 3 | 2 | 2 | {{1, 2}, {3, 4}} | AC | 0 | 0 | {{1, 3}, {2, 4}} | {{1, 3}, {2, 4}} | Пройден |

## 3.7 Результаты тестирования методом белого ящика

В ходе тестирования методом белого ящика были разработаны 3 теста, и все они оказались успешно пройдены. Это может одновременно указывать на корректность программы, а

также на то, что выбранная методика не учитывает какие-то специфические случаи, в которых могла бы проявиться ошибка, например, ошибки ввода элементов в матрицу.

# 4 Заключение

В рамках данной лабораторной работы были изучены методы тестирования «белым ящиком» и «черным ящиком».

Тестирование программы №1 по методу «черного ящика» выявило 10 непройденных тестов из 30, позволив обнаружить несоответствия спецификации. Были выявлены непройденные тесты из-за того, что программа не подразумевает проверку введенных данных.

При тестировании программы №2 с помощью «белого ящика» все 3 теста завершились успешно. Это означает, что скорее всего данный метод не затронул какой-то сценарий, где могла бы возникнуть ошибка.

Можно сделать вывод, что оба метода тестирования способны помочь в нахождении ошибок, но определить, какой из них эффективнее для тестирования каждой конкретной программы невозможно. По этой причине лучше сочетать оба метода в процессе тестирования.

# Список литературы

[1 ] Майерс, Г. Искусство тестирования программ. – Санкт-Петербург: Диалектика, 2012. – С. 272.