

МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

**«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ
ПЕТРА ВЕЛИКОГО»**

Институт Компьютерных Наук и Кибербезопасности

Высшая школа технологий искусственного интеллекта

Направление 02.03.01 Математика и Компьютерные Науки

Отчет по лабораторной работе №2.

По дисциплине:

«Методы тестирования программного обеспечения»

Тема Работы:

«Тестирование методами "белого" и "черного" ящиков»

Обучающийся: _____ Черепанов Михаил Дмитриевич

Руководитель: _____ Курочкин Михаил Александрович

«_____» _____ 20__ г.

Санкт-Петербург 2024

Содержание

Введение

Тестирование является важным этапом разработки программного обеспечения, направленным на обеспечение его качества и надежности. Целью тестирования является выявление ошибок и дефектов в программном продукте до его выпуска в эксплуатацию.

Существует два основных подхода к тестированию программного обеспечения: методом «Белого ящика» и методом «Черного ящика».

Тестирование «белым» ящиком основано на анализе внутренней структуры программного кода. При использовании этого подхода тестировщик имеет доступ к исходному коду программы и может осуществлять тестирование на уровне отдельных модулей, функций и логики программы.

Тестирование «черным» ящиком, напротив, основано на внешнем поведении программного продукта без знания его внутренней структуры. При использовании этого подхода тестировщик рассматривает программу как «черный ящик», не обращая внимания на детали реализации. Основная задача тестирования методом «черного ящика» - выявить случаи, в которых программа не соответствует спецификации.

Цель данной лабораторной работы - изучить обе методологии и получить практический опыт по их применению.

1 Постановка задачи

1. Изучить методологии тестирования «белым» и «черным» ящиками.
2. Разработать тесты и провести тестирование в соответствии с каждой из этих методологий для двух предоставленных программ.

2 Общие сведения о тестировании методами «Белого» и «Черного» ящиков

2.1 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 тестовые данные выбираются исключительно на основе требований спецификации (без привлечения каких-либо знаний о внутренней структуре программы).

Обнаружить все ошибки в программе при помощи данного метода можно было бы выполнив исчерпывающее входное тестирование. Однако в реальных задачах такое тестирование произвести невозможно ввиду большого (или бесконечного) пространства входных данных. В рамках рассматриваемого метода существует ряд эвристических методологий, направленных на то, чтобы минимальным количеством тестов обнаружить как можно большее число ошибок.

Далее рассматриваются основные методологии проектирования тестов, используемые в рамках метода “черного ящика”.

2.1.1 Разбиение на классы эквивалентности

Исчерпывающее тестирование программы на входных наборах данных принципиально неосуществимо. Следовательно, тестировщик всегда будет ограничиваться небольшим подмножеством всех возможных входных значений. А раз так, то хотелось бы выбрать “самое подходящее” подмножество тестов, т.е. такое, которое обеспечит наибольшую вероятность обнаружения большинства ошибок.

Определить такое “самое подходящее” подмножество можно используя разбиение входных данных программы на конечное число классов эквивалентности.

Это позволит с достаточной долей уверенности полагать, что тестирование одного конкретного значения из данного класса эквивалентно тестированию любого другого значения, принадлежащего тому же классу.

Определение классов эквивалентности

Определение классов эквивалентности сводится к последовательному рассмотрению каждого из входных условий (обычно это предложение или фраза, приведенные в спецификации) и разбиению его на две или несколько групп. Часто на основе спецификации выделяются два типа классов:

1. Допустимые классы эквивалентности, представляющие допустимые входные данные программы.
2. Недопустимые классы эквивалентности, представляющие все остальные возможные состояния условий (т.е. недопустимые входные значения).

2.1.2 Анализ граничных значений

Как показывает опыт, тесты, исследующие граничные условия, приносят большую пользу, чем тесты, которые не обеспечивают этого. 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.

1. 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.

2. При создании тестов внимание фокусируется не только на входных условиях (пространство входных значений), но и на пространстве результатов (выходные классы эквивалентности).

2.1.3 Применение причинно-следственных диаграмм

Одной из слабых сторон анализа граничных значений и разбиения данных на классы эквивалентности является то, что они не исследуют комбинаций входных условий.

Тестирование комбинаций входных условий — непростая задача, поскольку даже после разбиения входных условий на классы количество возможных комбинаций обычно выражается астрономическими цифрами. В отсутствие систематического способа выбора подмножества входных условий его, как правило, выбирают произвольным образом, что и становится причиной неэффективного тестирования.

Метод причинно-следственных диаграмм помогает систематически выбирать высокорезультативные тесты. Его дополнительным преимуществом является то, что он позволяет обнаруживать неполноту и неоднозначность исходных спецификаций.

Причинно-следственная диаграмма представляет собой формальный язык, на который транслируется спецификация, написанная на естественном языке. Фактически такая диаграмма является аналогом цифровой логической схемы, но для ее описания используется более простая нотация по сравнению с той, которая принята в электронике. Для того чтобы воспользоваться методом причинно-следственных диаграмм, никаких знаний электроники, кроме понимания правил булевой алгебры (т.е. умения работать с логическими операторами and, or и not), не требуется.

2.1.4 Стратегия разработки тестов в рамках методики «Черного ящика»

Каждый метод из перечисленных выше обеспечивает создание определенного набора полезных тестов, но ни один из них сам по себе не может обеспечить полный набор тестов.

Поэтому в рамках тестирования «Черного ящика» рекомендуется использовать следующую стратегию:

1. Если спецификация содержит комбинации входных условий, то начинать следует с применения метода причинно-следственных диаграмм.
2. В любом случае необходимо использовать анализ граничных значений. Этот метод включает анализ как входных, так и выходных данных. Анализ граничных значений дает набор дополнительных тестовых условий, но многие из них могут быть включены в тесты метода причинно-следственных диаграмм.
3. Определить допустимые и недопустимые классы эквивалентности для входных и выходных данных и в случае необходимости создать новые тесты в дополнение к уже имеющимся.
4. Для получения дополнительных тестов рекомендуется использовать метод прогнозирования ошибок. Метод состоит в выдвижении предположений об ошибках, а также определении тестов, связанных с допущениями, которые программист мог сделать при чтении спецификации (те моменты, которые не были включены в спецификацию либо случайно, либо из-за того, что автору спецификации они показались очевидными).

2.2 Метод «Белого ящика»

Тестирование "белым" ящиком основано на анализе внутренней структуры программного кода. При использовании этого подхода тестировщик имеет доступ к исходному коду программы и может осуществлять тестирование на уровне отдельных модулей, функций и логики программы.

При тестировании методом "белого ящика" учитывается степень покрытия логики (кода) программы. Исчерпывающее тестирование методом "белого ящика" требует прохождения каждого пути в программе, однако в случае программ, содержащих циклы, удовлетворение этого условия — задача нереальная. Поэтому существуют различные эвристические правила, позволяющие определить набор критериев, достаточных для приемлемого тестирования данным методом.

Эти критерии можно упорядочить по возрастанию качества покрытия логики программы следующим образом:

- Критерий покрытия операторов. В соответствии с данным критерием набор тестов является достаточным, если каждая инструкция в программе выполняется хотя бы один раз.
- Критерий покрытия условий. 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). Также каждая точка входа в программу или подпрограмму были пройдены хотя бы один раз.
- Комбинаторное покрытие условий. Этот критерий требует создания такого количества тестов, при котором каждая возможная комбинация результатов вычисления условий в каждом ветвлении и каждая точка входа проверяются по крайней мере один раз.

3 «Треугольник Паскаля»

3.1 Постановка задачи, спецификация

Название:

Печать N строк треугольника Паскаля

Дано:

Число N.

Требуется:

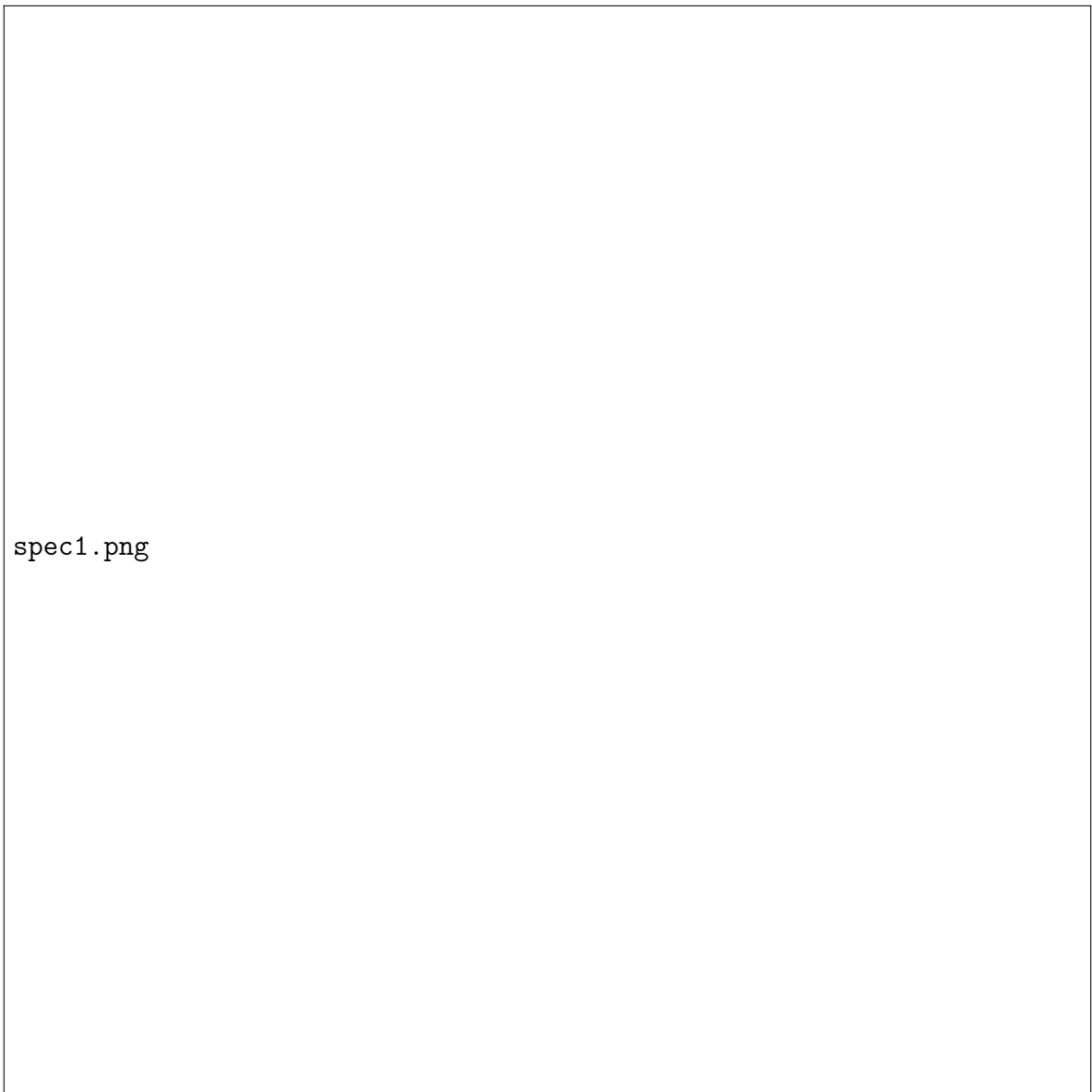
Вывести на экран N строк треугольника Паскаля или сообщение об ошибке.

Ограничение:

$0 \leq N \leq 100$

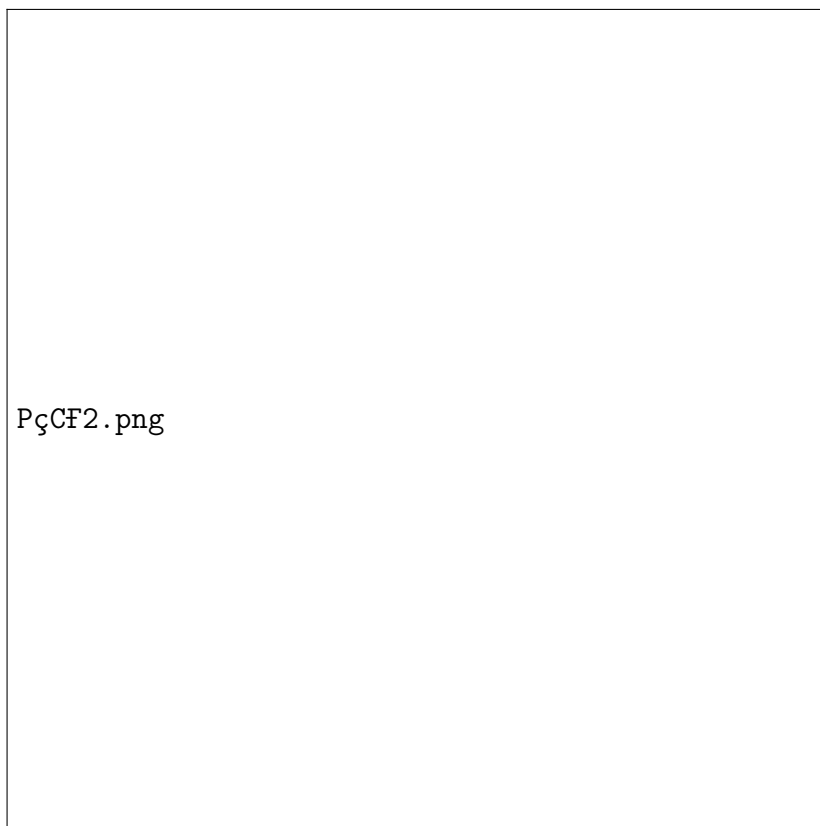
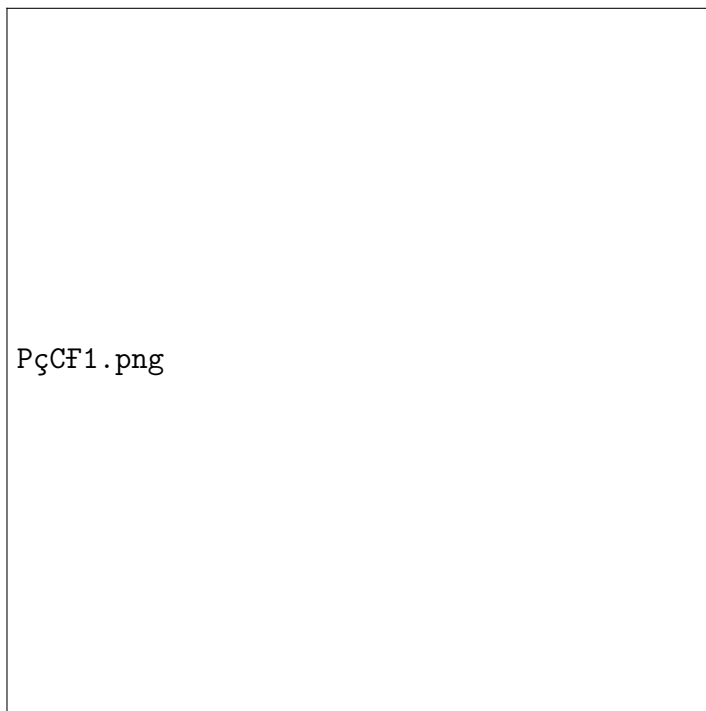
N - целое число

Спецификация:



spec1.png

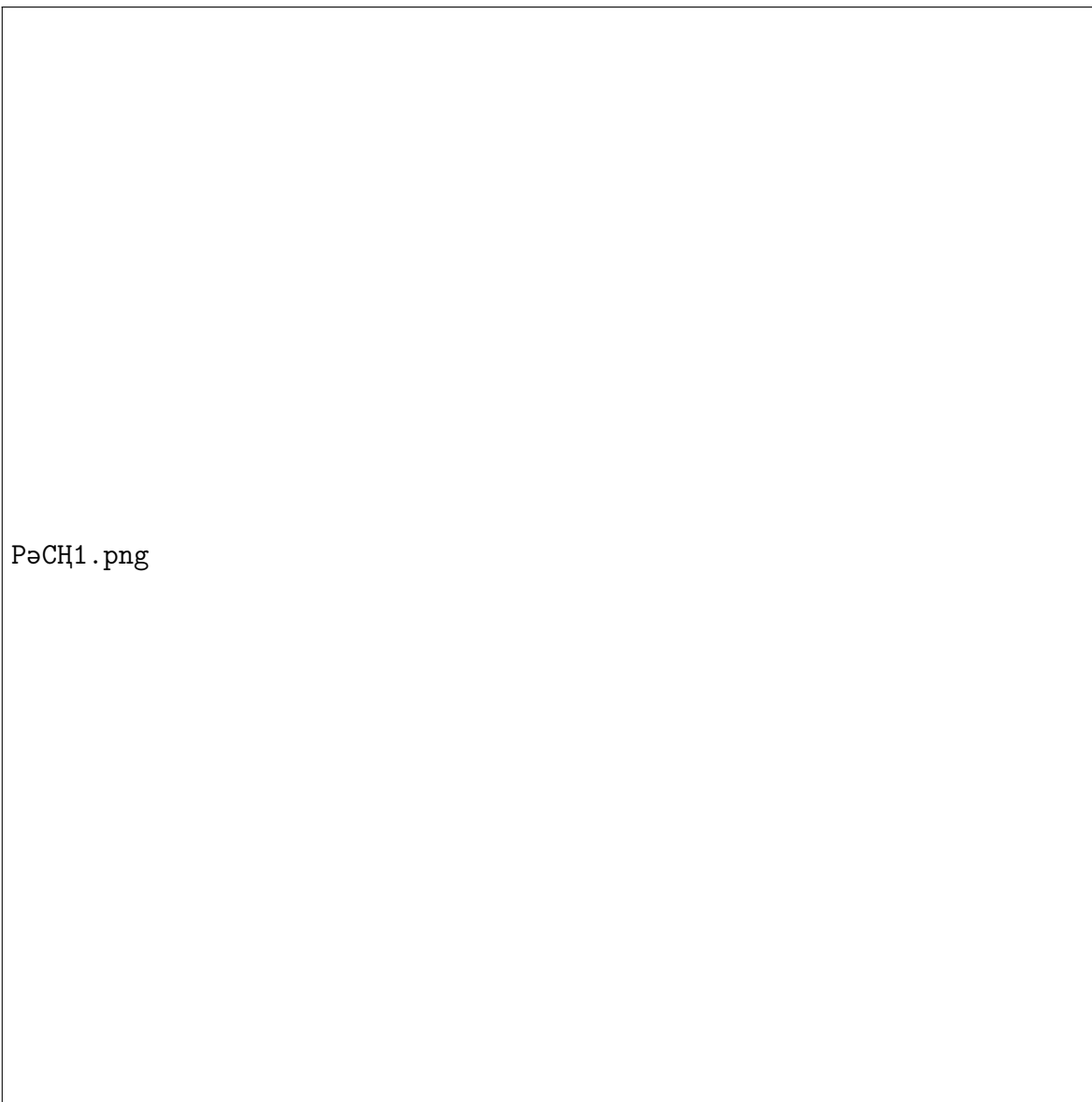
Блок-схема алгоритма:



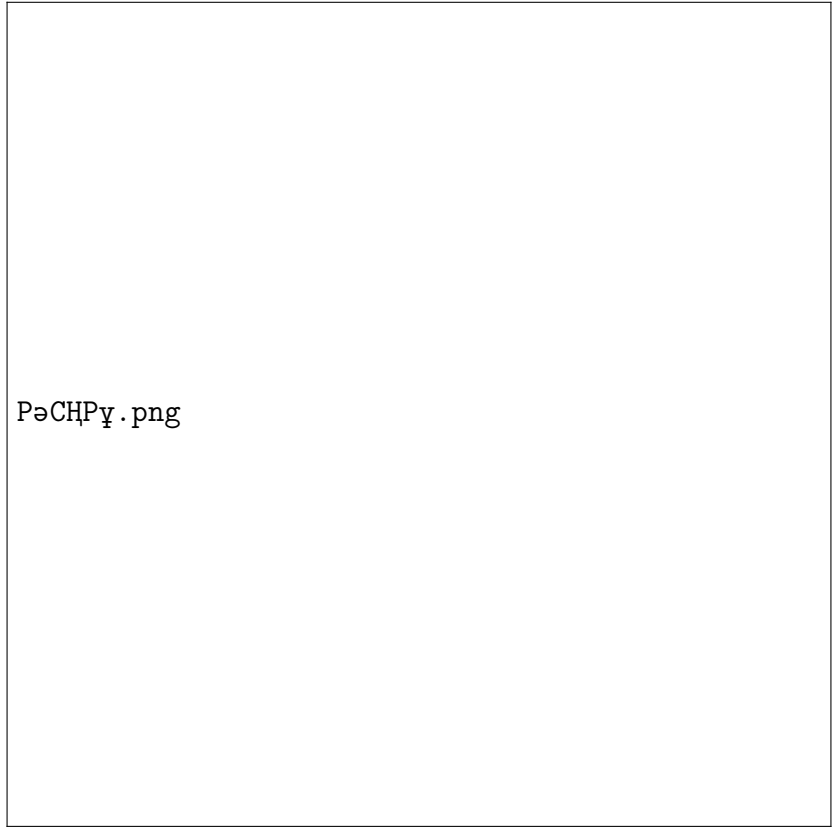
3.2 Составление тестов методом «черного ящика»

Классы эквивалентности

Исходя из ограничений для программы составим следующие классы эквивалентности:



Составим тесты для допустимых классов эквивалентности $\{1,3\}$. Нет оснований предполагать, что некоторые значения из данных классов эквивалентности будут обрабатываться по-разному, поэтому выберем случайные значения:



В выбранных случаях программа отработала корректно. Составим тесты для недопустимых классов:

P,,PəCH.png

Проверка граничных условий:

В спецификации указано единственное условие:

$$0 \leq N \leq 100$$

Для каждой из границ напишем по 5 тестов: граница, наиболее близкое к границе целое число(с низу и сверху), наиболее близкое к границе дробное число(с низу и сверху).

РҮСІСҢСРҢ11.png

Рисунок 12.png

Причинно-следственная диаграмма:

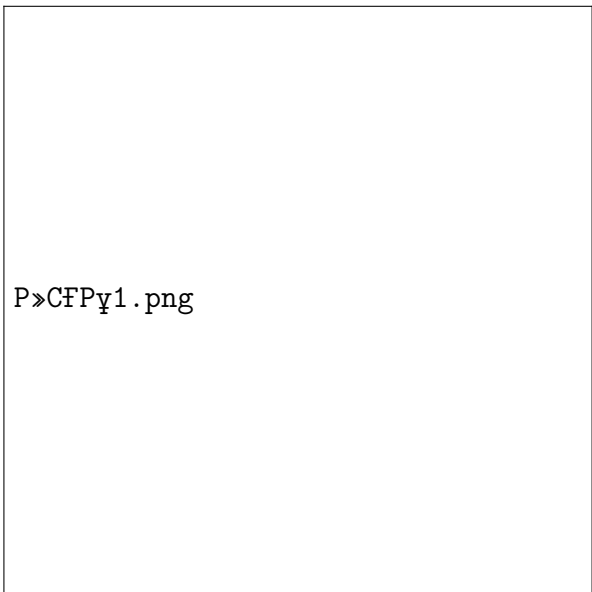
Для построения причинно-следственной диаграммы еще раз внимательно прочитаем спецификацию и выделим из нее отдельно три причины:

- 1: $n \geq 0$;
- 2: $n < 100$;
- 3: n - целое число;

Выделим два следствия:

- 11: На экран выводится N строк треугольника Паскаля.
- 12: На экран выводится сообщение об ошибке.

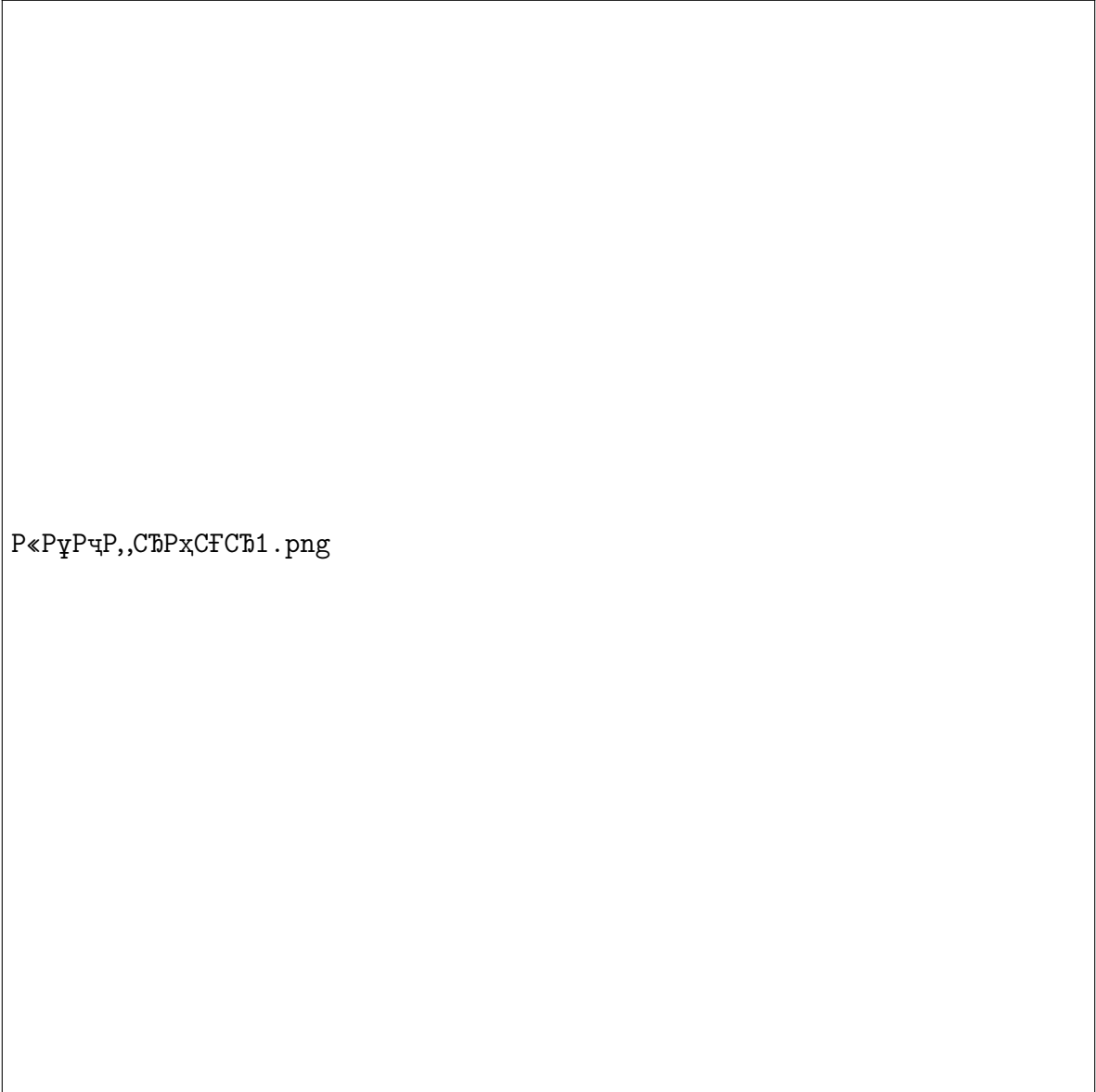
Построим причинно-следственную диаграмму:



P»CFPҫ1 .png

В соответствии с методологией причинно следственных диаграмм необходимо для единственного желаемого условия рассмотреть все комбинации порождающих его условий.

Таким образом, требуется разработать один тест, проверяющий выполнимость одновременно 1,2 и 3 условий.

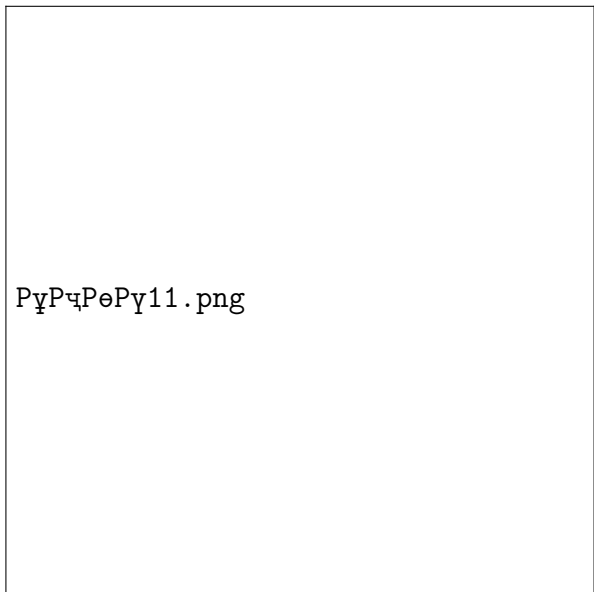


Р«РҫРҫР,СЃРҫСЃСЃ1.png


3.3 Составление тестов методом «белого ящика»

Алгоритм составления тестов методом белого ящика предполагает обход всех возможных путей в теле программы и проверка выполнения каждого оператора не менее одного раза.

Для этого обозначим каждый из блоков условий, и блоки, находящиеся в разных ветках программы отдельным символом.



P̄P̄P̄P̄P̄11.png



P̄P̄P̄P̄P̄12.png

Выделим следующие пути в теле программы, которые необходимо покрыть тестами:

ABC

ABDE

ABDFG

ABDFHIJ

ABDFHIKNPM

ABDFHIKNORM

Другие пути во время исполнения программы невозможны.

Каждый из выделенных путей покроем тестами:

ABC: строка "q"

ABDE: 1.5 (Любое значение, не являющееся целым числом)

ABDFG: -1 (Любое отрицательное число)


ABDFHIJ: 0

ABDFHIKNPMJ: 4 (Любое целое число $\in [1, 100]$)

ABDFHIKNORMJ: 4 (Любое целое число $\in [1, 100]$)

Входные данные для двух последних выделенных путей совпадают, так как в любом случае при прохождении одного из них, программа зайдет во второй.

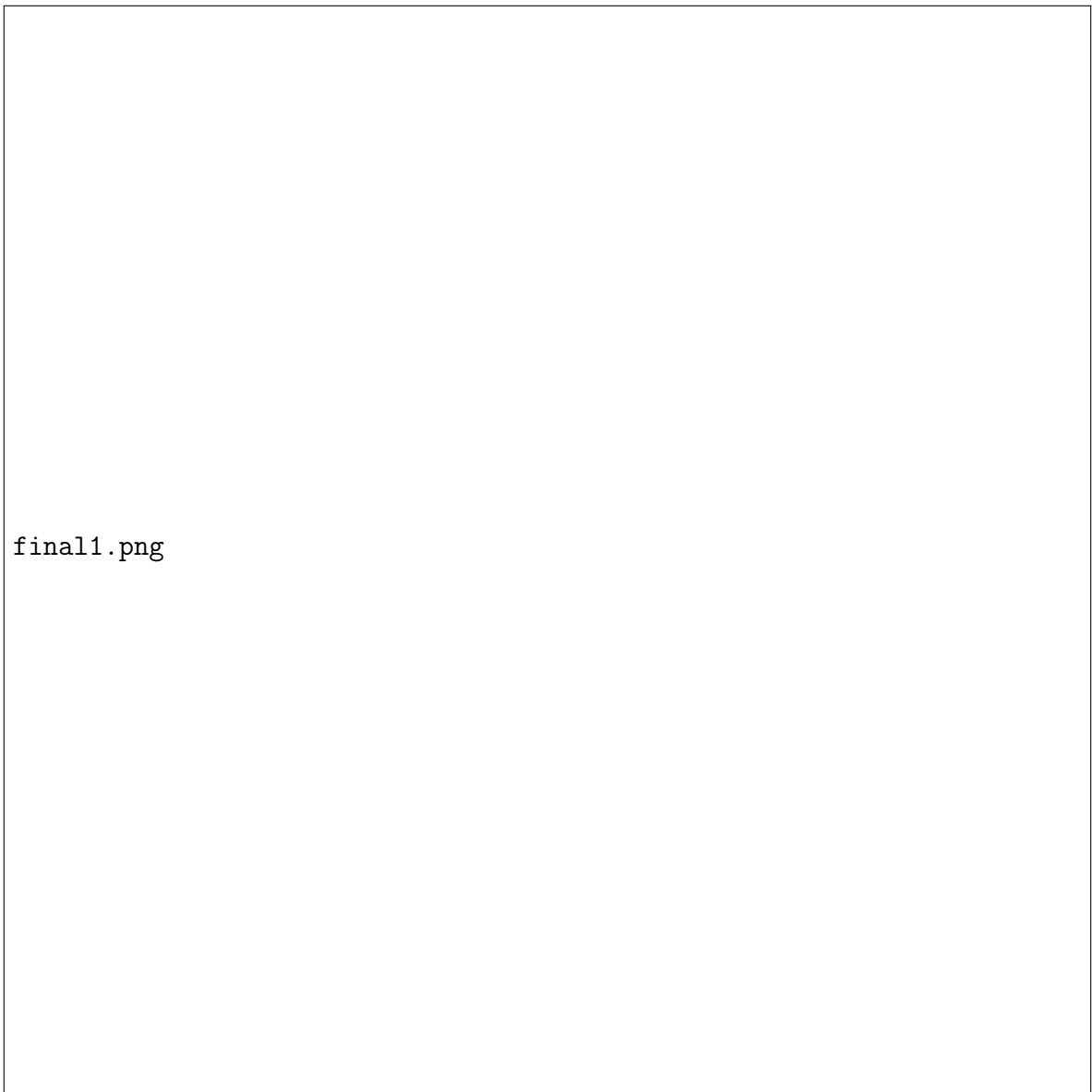
Итак, в ходе тестирования методом «белого ящика» были разработаны следующие тесты:



tablewb1.png

3.4 Тестирование

Таким образом, пользуясь методами «Черного» и «Белого» ящиков, можно составить следующий набор тестов, покрывающий граничные условия, все выделенные классы эквивалентности, все пути в теле программы.



final1.png

В ходе тестирования программы несоответствий спецификации и ошибок в логике программы обнаружено не было.

4 «Проверка правильности скобочной последовательности»

4.1 Постановка задачи, спецификация

Название:

Проверка правильности скобочной последовательности.

Дано:

Строка, в которой могут содержаться скобочные символы:

« {, }, [,], (,) », а так же другие символы.

Требуется:

Проверить правильность скобочных последовательностей внутри строки.

Проверяются скобочные последовательности трех типов:

состоящие из круглых скобок: ();

состоящие из фигурных скобок: { };

состоящие из квадратных скобок: [];

В случае, если каждый из трех типов последовательностей верен - вывести на экран «Да».

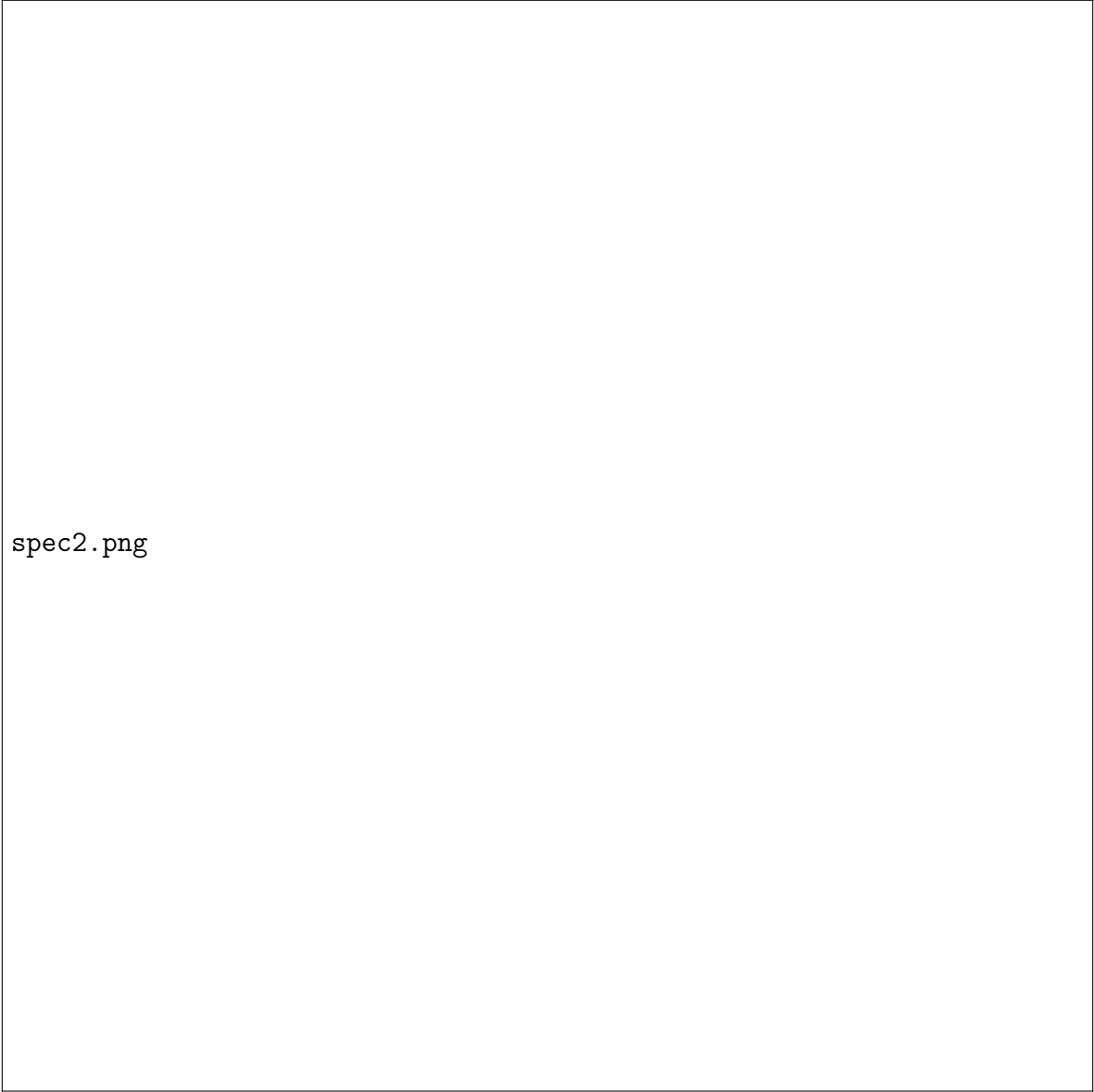
Иначе - «Нет».

В случае, если размер строки превышает 10000 символов - вывести сообщение об ошибке.

Ограничение:

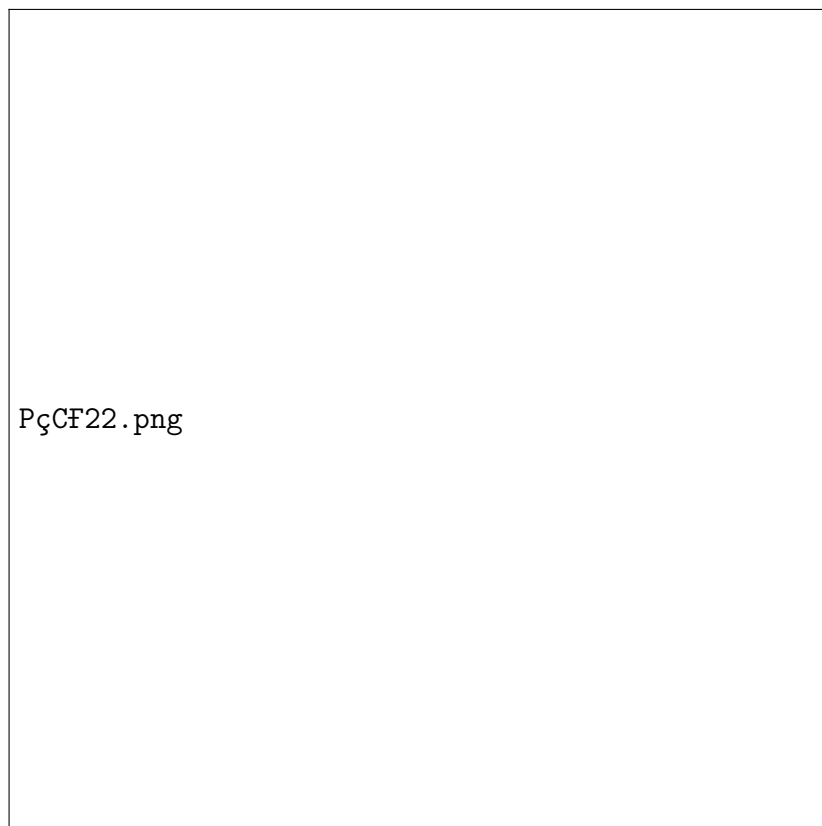
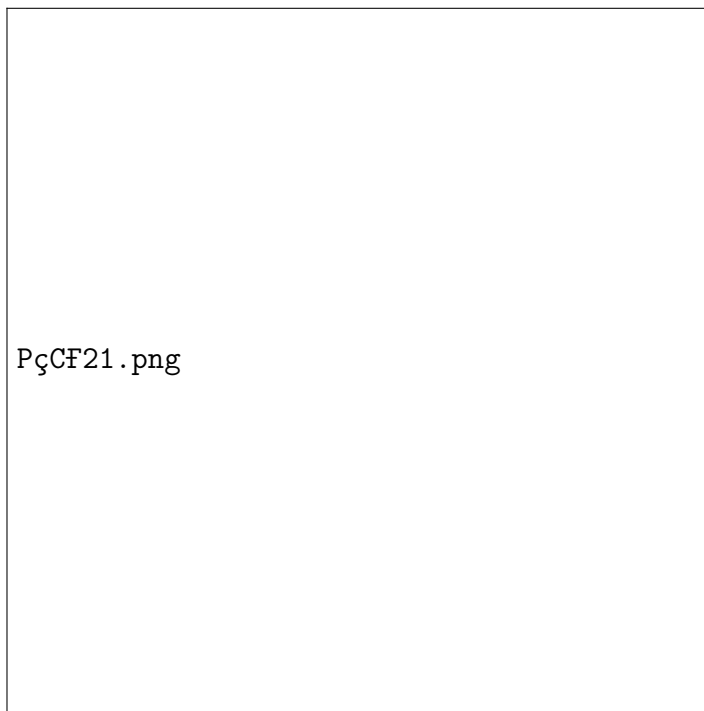
Длина строки не более 10000 символов.

Спецификация:



spec2.png

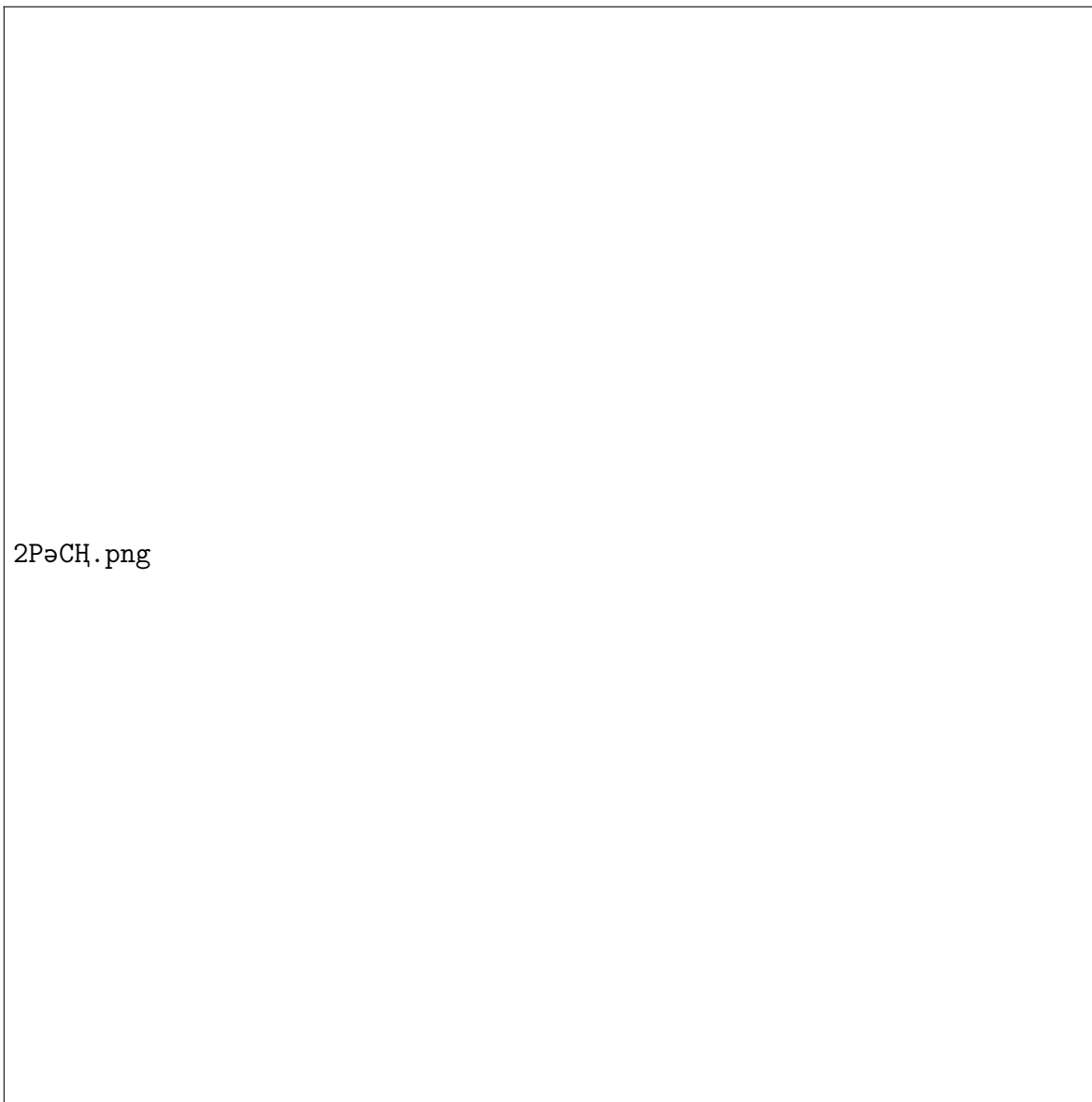
Блок-схема алгоритма:



4.2 Составление тестов методом «черного ящика»

Классы эквивалентности

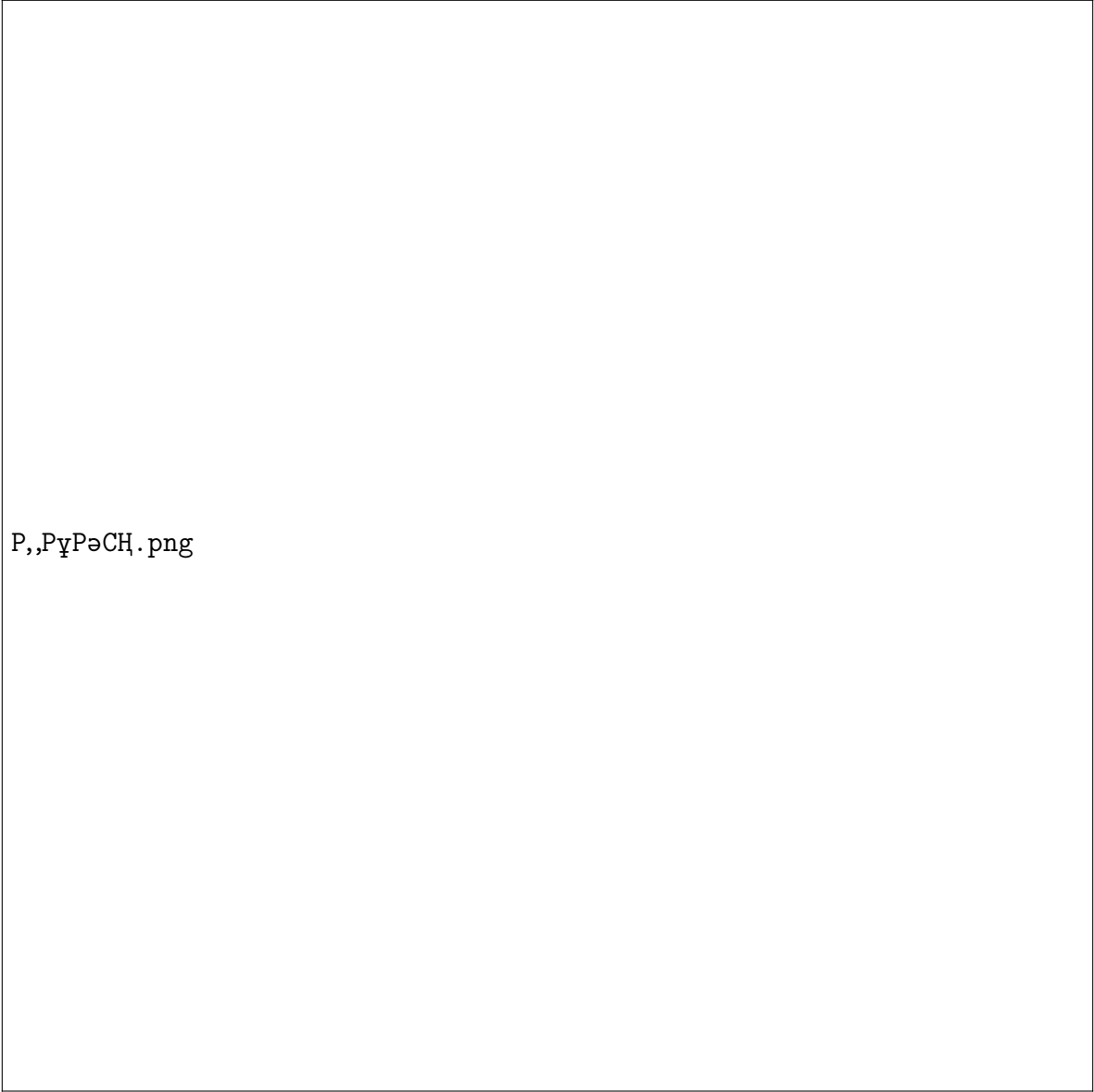
Исходя из спецификации составим следующие классы эквивалентности:



Составим тесты для допустимых классов эквивалентности $\{1,3,5,7\}$. Нет оснований предполагать, что некоторые значения из данных классов эквивалентности будут обрабатываться по-разному, поэтому выберем случайные значения для каждого из допустимых классов:

2P̄P̄CH̄.png

Составим тесты для недопустимых классов:



P,,P̸P̸CH̸.png

Проверка граничных условий

В спецификации указано единственное граничное условие:

Длина строки $< 10\,000$ символов. Т.к. длина не может быть отрицательным числом, граничным условием так же является пустая строка(длины 0).

Поэтому проверим верхнюю границу, наиболее близкое к ней значение сверху и снизу, нижнюю границу, наиболее близкое к ней верхнее значение:

2РyCГCГCГPь.png

Причинно-следственная диаграмма:

Для составления диаграммы выделим следующие причины из спецификации:

1. Корректная расстановка в строке символов: «(» или «)»;
2. Корректная расстановка в строке символов: «{» или «}»;
3. Корректная расстановка в строке символов: «[» или «]»;
4. Длина строки ≤ 10000 символов;

Были выделены следующие следствия:

11. Вывод «Да» при корректности скобочной последовательности.
12. Вывод «Нет» при некорректности скобочной последовательности.
13. Вывод сообщения об ошибке при длине строки превышающей 10 000.

Построим причинно следственную диаграмму:



2P>CTCTPъPy.png

Применим метод обратной трассировки графа, который заключается в обходе диаграммы от следствий к условиям:

Указания к методу обратной трассировки графа:

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.

2. 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.

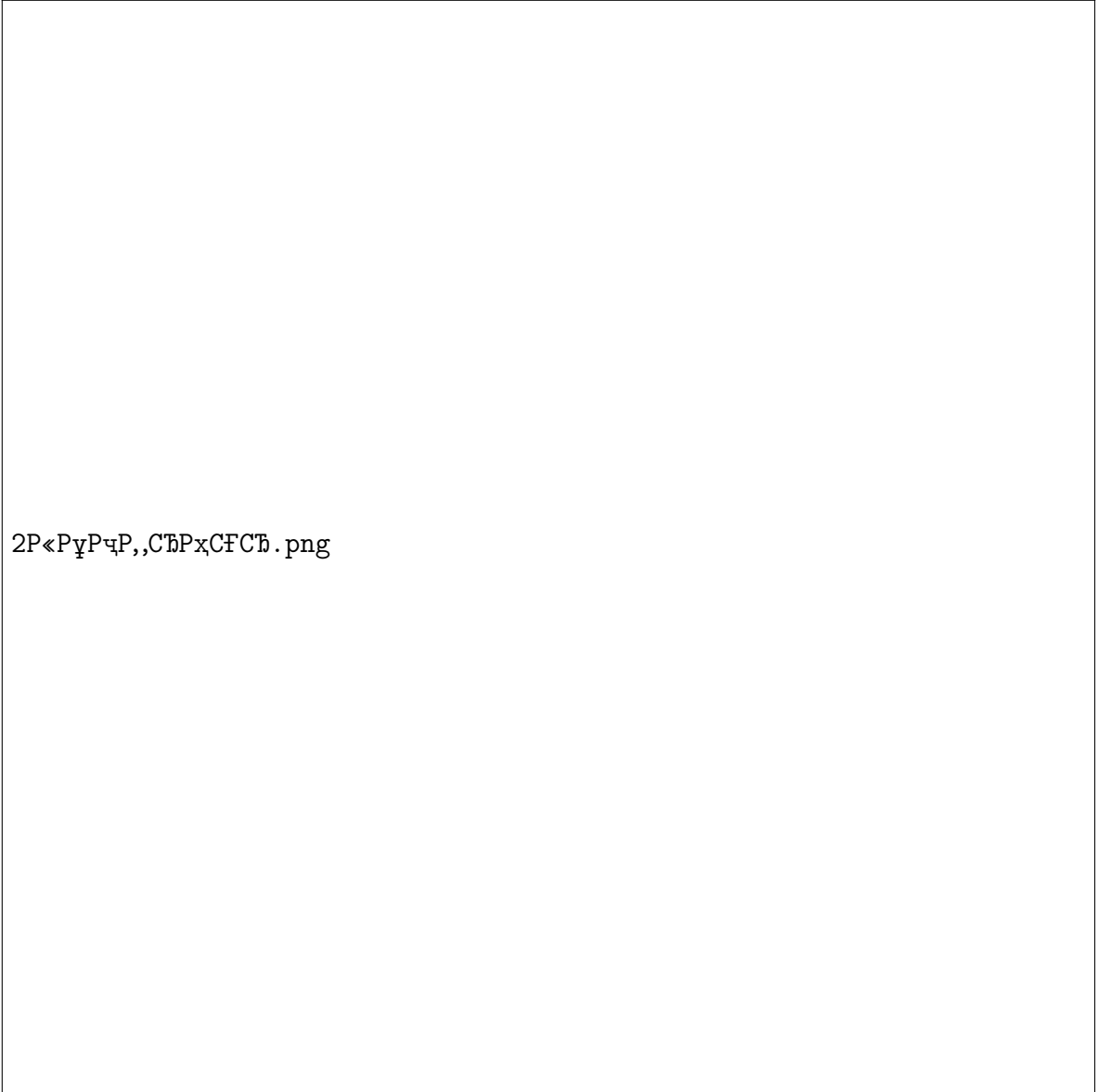
3. Если путь обратной трассировки проходит через узел типа `and`, выход которого должен принимать значение 0, то необходимо указать лишь одно условие, при котором все входы являются нулями.

Исходя из вышеперечисленных правил построим тесты, проверяющие условия 1,2,3,4 - поотдельности друг от друга;

Выполнение условий 1,2,3,4 - одновременно.

Данные тесты покрывающие данные условия уже были сгенерированы в предыдущих пунктах.

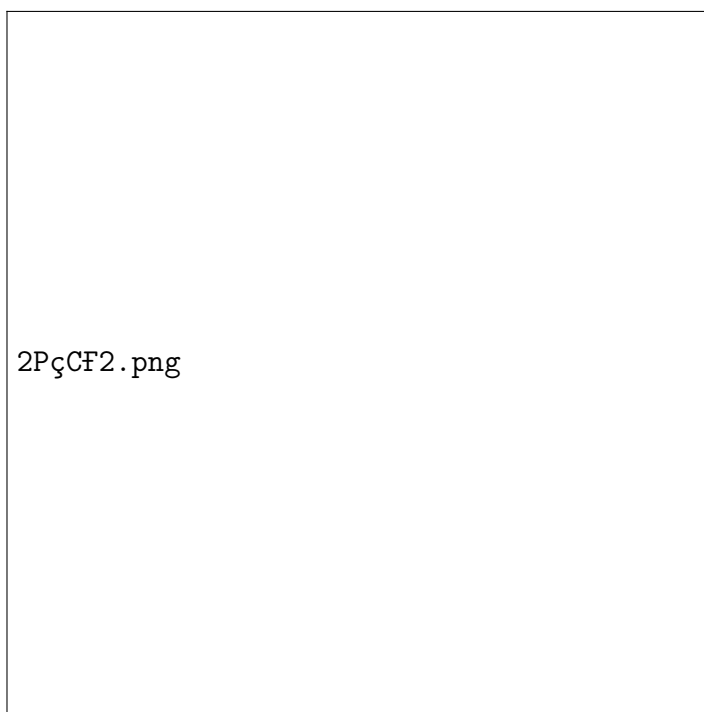
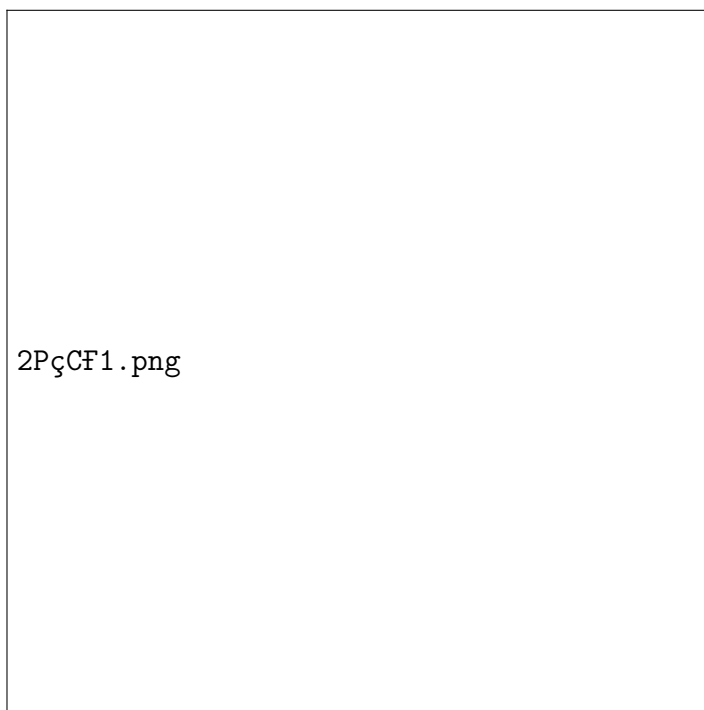
Так как в данном случае строка, не содержащая скобок интерпретируется как правильная скобочная последовательность, и есть основания предполагать, что строка в которой присутствуют скобки обрабатывается отлично от пустой строки - добавим следующий тест:



2Р«РҫРҫР,СЃРҫСҫСЃ. png

4.3 Составление тестов методом «белого ящика»

Для составления тестов обозначим каждый из блоков условий и блоки, находящиеся в разных ветках программы отдельными символами на блок-схеме:



После исследования блок-схемы были выделены следующие маршруты:
ABC

$ABDE$

$ABDA_1B_1C_1G_1FG$

$ABDA_1B_1D_1E_1F_1FH$

$ABDA_1B_1D_1E_1C_1G_1H_1FG$

$ABDA_1B_1D_1E_1C_1G_1F_1FH$

Другие маршруты во время выполнения программы невозможны.

Каждый из выделенных маршрутов покроем тестами:

ABC – «:q»

$ABDE$ – «.....» (длина строки $>10\ 000$)

$ABDA_1B_1C_1G_1FG$ – «aaa» (отсутствуют скобки)

$ABDA_1B_1D_1E_1F_1FH$ – «(aaa]» (после скобки одного вида, идет скобка другого вида)

$ABDA_1B_1D_1E_1C_1G_1H_1FG$ – «(aaa)» (корректная последовательность)

$ABDA_1B_1D_1E_1C_1G_1F_1FH$ – «(aa» (после открывающейся скобки отсутствует закрывающаяся)

В результате анализа был сгенерирован следующий набор тестов:

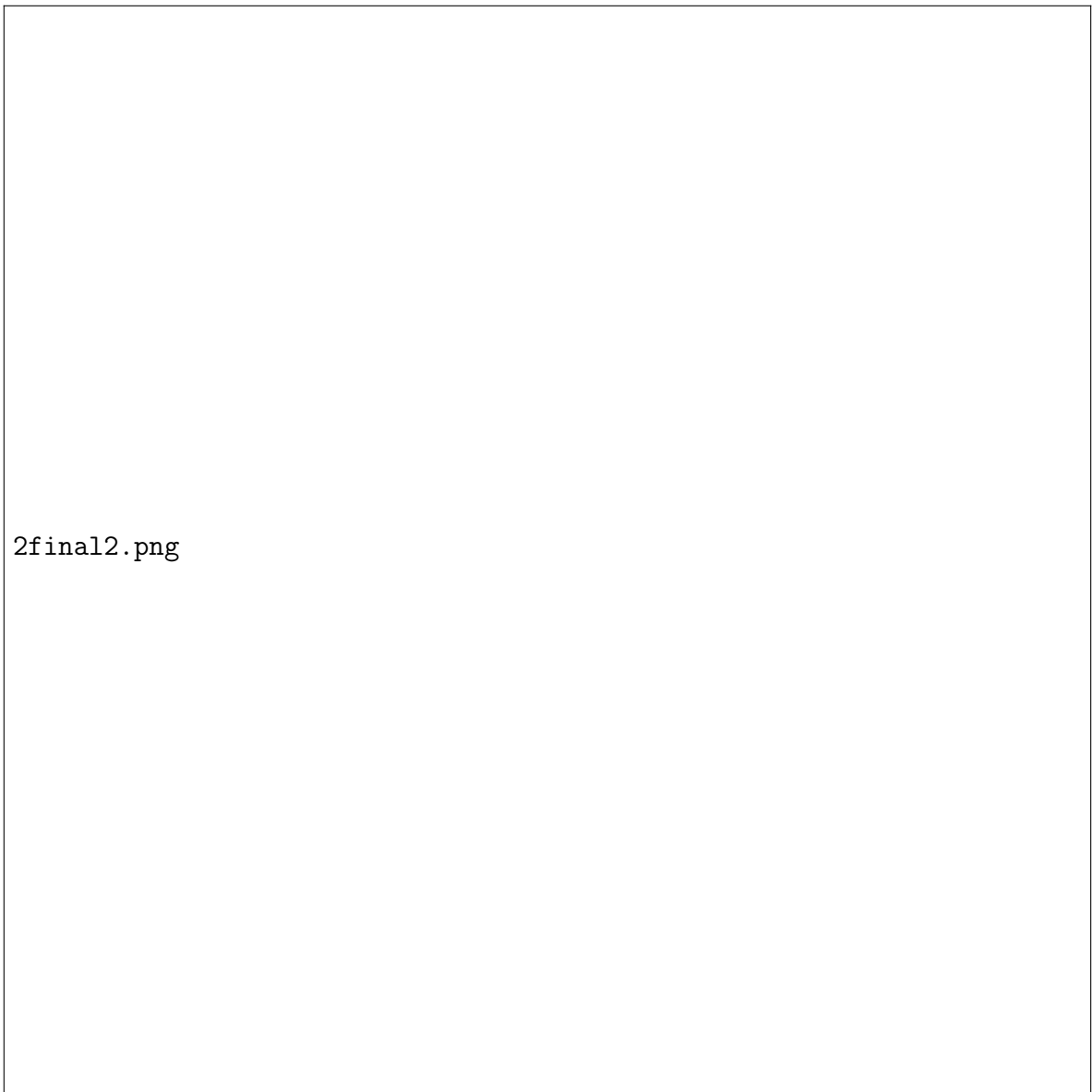


2P5CSC7.png

4.4 Тестирование

Таким образом, пользуясь методами «Черного» и «Белого» ящиков, можно составить следующий набор тестов, покрывающий граничные условия, все выделенные классы эквивалентности, все пути в теле программы.

2final1.png



2final2.png

В ходе тестирования программы несоответствий спецификации и ошибок в логике программы обнаружено не было.

Заключение

В ходе выполнения работы были изучены стратегии тестирования методами «белого» и «черного» ящиков и их совместное применение для генерации лучшего покрытия кода тестами.

Были разработаны тесты для двух предоставленных программ, путем последовательного применения стратегий: сначала «черного» ящика, затем «белого ящика».

В ходе тестирования ошибок в логике программ и несоответствий спецификации обнаружено не было, но это может свидетельствовать о том, что в данных программах действительно нет ошибок, поскольку предоставленные программы не являются объемными. Тестирование проводилось исключительно в учебных целях. На реальных проектах примененные стратегии тестирования должны дать положительные результаты.

Список использованных источников

- [1] ИСКУССТВО ТЕСТИРОВАНИЯ ПРОГРАММ. Третье издание. Гленфорд Майерс, Том Баджетт, Кори Сандлер