Лабораторная работа № 9 Тестирование программ методами белого ящика

Цель работы: Усвоение студентами методов тестирования логики программы, формализованного описания результатов тестирования и стандартов по составлению схем программ**.**

Основные понятия

Тестирование программного обеспечения охватывает целый ряд видов деятельности, аналогичных последовательности процессов разработки программного обеспечения. В него входят /1/:

а) постановка задачи для теста,

б) проектирование теста,

в) написание тестов,

г) тестирование тестов,

д) выполнение тестов,

е) изучение результатов тестирования.

Решающую роль играет проектирование тестов. Возможны разные подходы к проектированию тестов. Первый состоит в том, что тесты проектируются на основе внешних спецификаций программ и модулей, либо спецификаций сопряжения программы или модуля. Программа при этом рассматривается как черный ящик (стратегия ‘черного ящика’). Существо такого подхода - проверить соответствует ли программа внешним спецификациям. При этом логика модуля совершенно не принимается во внимание.

Второй подход основан на анализе логики программы (стратегия ‘белого ящика’). Существо подхода - в проверке каждого пути, каждой ветви алгоритма. При этом внешняя спецификация во внимание не принимается.

Ни один из этих подходов не является оптимальным. Из анализа существа первого подхода ясно, что его реализация сводится к проверке всех возможных комбинаций значений на входе программы. Тестирование любой программы для всех значений входных данных невозможно, так как их бесконечное множество, поэтому ограничиваются меньшим. При этом исходят из максимальной отдачи теста по сравнению с затратами на его создание. Она измеряется вероятностью того, что тест выявит ошибки, если они имеются в программе. Затраты измеряются временем и стоимостью подготовки, выполнения и проверки результатов теста.

Проанализируем теперь второй подход к тестированию. Даже если предположить, что выполнены тесты для всех путей программы, нельзя с полной уверенностью утверждать, что модуль не содержит ошибок.

Очевидное основание этого утверждения состоит в том, что выполнение всех путей не гарантирует соответствия программы ее спецификациям. Допустим, если требовалось написать программу для вычисления кубического корня, а программа фактически вычисляет корень квадратный, то программа будет совершенно неправильной, даже если проверить все

пути. Вторая проблема - отсутствующие пути. Если программа реализует спецификации не полностью (например, отсутствует такая специализированная функция, как проверка на отрицательное значение входных данных программы вычисления квадратного корня), никакое тестирование существующих путей не выявит такой ошибки. И, наконец, проблема зависимости результатов тестирования от входных данных. Путь может правильно выполняться для одних данных и неправильно для других. Например, если для определения равенства 3 чисел программируется выражение вида:

IF (A+B+C)/3=A,

то оно будет верным не для всех значений A, B и C (ошибка возникает в том случае, когда из двух значений B или C одно больше, а другое на столько же меньше A). Если концентрировать внимание только на тестировании путей, нет гарантии, что эта ошибка будет выявлена.

Таким образом, полное тестирование программы невозможно. Тест для любой программы будет обязательно неполным, то есть тестирование не гарантирует полное отсутствие ошибок в программе. Стратегия проектирования тестов заключается в том, чтобы попытаться уменьшить эту неполноту насколько это возможно.

2.1 Методы стратегии ‘белого ящика’

Тестирование по принципу белого ящика характеризуется степенью, какой тесты выполняют или покрывают логику (исходный текст программы).

2.1.1 Метод покрытия операторов

Целью этого метода тестирования является выполнение каждого оператора программы хотя бы один раз.

Пример:

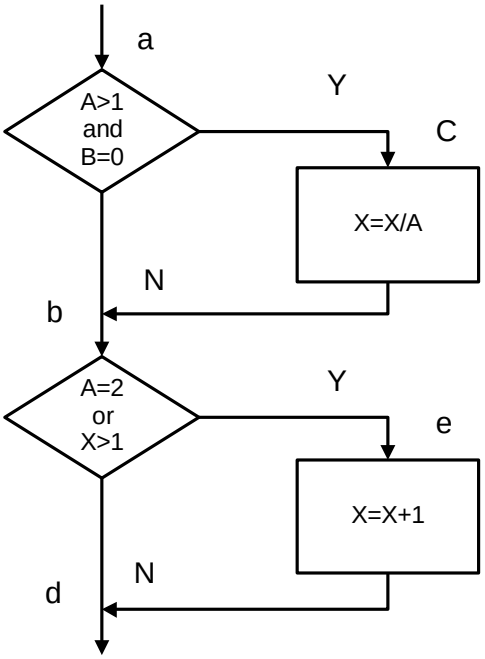

Рисунок 2.1

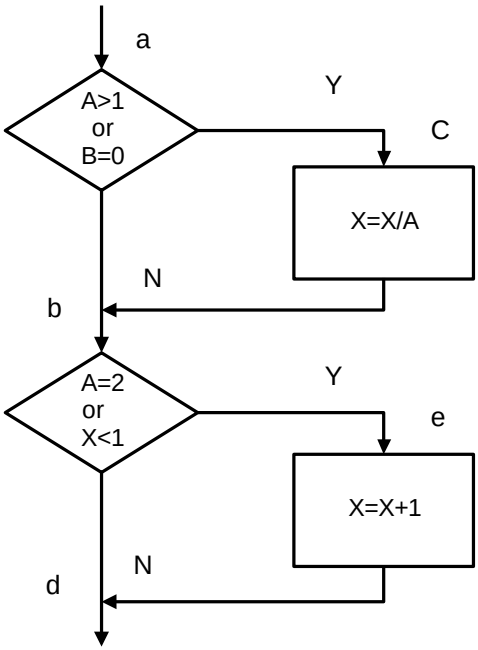

Рисунок 2.2

В этой программе можно выполнить каждый оператор, записав один единственный тест, который реализовал бы путь *ace*. Т.е., если бы на входе было: A=2, B=0, X=3, каждый оператор выполнился бы один раз. Но этот критерий на самом деле хуже, чем он кажется на первый взгляд. Пусть в первом условии вместо "*and*"→"*or*" и во втором вместо *"x>1"→"x<1"* (блок-схема правильной программы приведена на рисунке 2.1, а неправильной - на рисунке 2.2). Результаты тестирования приведены в таблице 2.1. Обратите внимание: ожидаемый результат определяется по алгоритму на рисунке 2.1, а фактический - по алгоритму рисунка 2.2, поскольку определяется чувствительность метода тестирования к ошибкам программирования. Как видно из этой таблицы, ни одна из внесенных в алгоритм ошибок не будет обнаружена .

Таблица 2.1 - Результат тестирования методом покрытия операторов

Тест	Ожидаемый результат	Фактический результат	Результат тестирования
A=2, B=0, X=3	X=2,5	X=2,5	неуспешно

2.2 Метод покрытия решений (покрытия переходов)

Более сильный метод тестирования известен как покрытие решений (покрытие переходов). Согласно данному методу каждое направление перехода должно быть реализовано по крайней мере один раз.

Покрытие решений обычно удовлетворяет критерию покрытия операторов. Поскольку каждый оператор лежит на некотором пути, исходящем либо из оператора перехода, либо из точки входа программы, при выполнении каждого направления перехода каждый оператор должен быть выполнен.

Для программы приведенной на рисунке 2.2 покрытие решений может быть выполнено двумя тестами, покрывающими пути {*ace, abd*}, либо {*acd,abe*}. Пути {*acd,abe*}покроим, выбрав следующие исходные данные: {A=3, B=0, X=3} и {A=2, B=1, X=1} (результаты тестирования - в таблице 2.2).

Таблица 2.2 - Результат тестирования методом покрытия решений

Тест	Ожидаемый результат	Фактический результат	Результат тестирования
A=3, B=0, X=3	X=1	X=1	неуспешно
A=2, B=1, X=1	X=2	X=1,5	успешно

2.3 Метод покрытия условий

Лучшие результаты по сравнению с предыдущими может дать метод покрытия условий. В этом случае записывается число тестов, достаточное для того, чтобы все возможные результаты каждого условия в решении выполнялись по крайней мере один раз.

В предыдущем примере имеем четыре условия: {A>1, B=0}, {A=2, X>1}. Следовательно, требуется достаточное число тестов, такое, чтобы реализовать ситуации, где A>1, A≤1, B=0 и B≠0 в точке *a* и A=2, A≠2, X>1 и X≤1 в точке B. Тесты, удовлетворяющие критерию покрытия условий и соответствующие им пути:

а) A=2, B=0, X=4 *ace*

б) A=1, B=1, X=0 *abd*

Таблица 2.3 - Результаты тестирования методом покрытия условий

Тест	Ожидаемый результат	Фактический результат	Результат тестирования
A=2, B=0, X=4	X=3	X=3	неуспешно
A=1, B=1, X=0	X=0	X=1	успешно

2.4 Критерий решений (условий)

Критерий покрытия решений/условий требует такого достаточного набора тестов, чтобы все возможные результаты каждого условия в решении выполнялись по крайней мере один раз, все результаты каждого решения выполнялись по крайней мере один раз и, кроме того, каждой точке входа передавалось управление по крайней мере один раз.

Два теста метода покрытия условий

а) A=2, B=0, X=4 *ace*

б) A=1, B=1, X=0 *abd*

отвечают и критерию покрытия решений/условий. Это является следствием того, что одни условия приведенных решений скрывают другие условия в этих решениях. Так, если условие A>1 будет ложным, транслятор может не

проверять условия B=0, поскольку при любом результате условия B=0, результат решения ((A>1)&(B=0)) примет значение *ложь*. Следовательно, недостатком критерия покрытия решений/условий является невозможность его применения для выполнения всех результатов всех условий.

Другая реализация рассматриваемого примера приведена на рисунке 2.3. Многоусловные решения исходной программы разбиты на отдельные решения и переходы. Наиболее полное покрытие тестами в этом случае выполняется так, чтобы выполнялись все возможные результаты каждого простого решения. Для этого нужно покрыть пути HILP (тест A=2,B=0,X=4), HIMKT (тест A=3, B=1, X=0), HJKT (тест A=0, B=0, X=0), HJKR (тест A=0, B=0, X=2)..

Протестировав алгоритм на рисунке 2.3, нетрудно убедиться в том, что критерии покрытия условий и критерии покрытия решений/условий недостаточно чувствительны к ошибкам в логических выражениях.

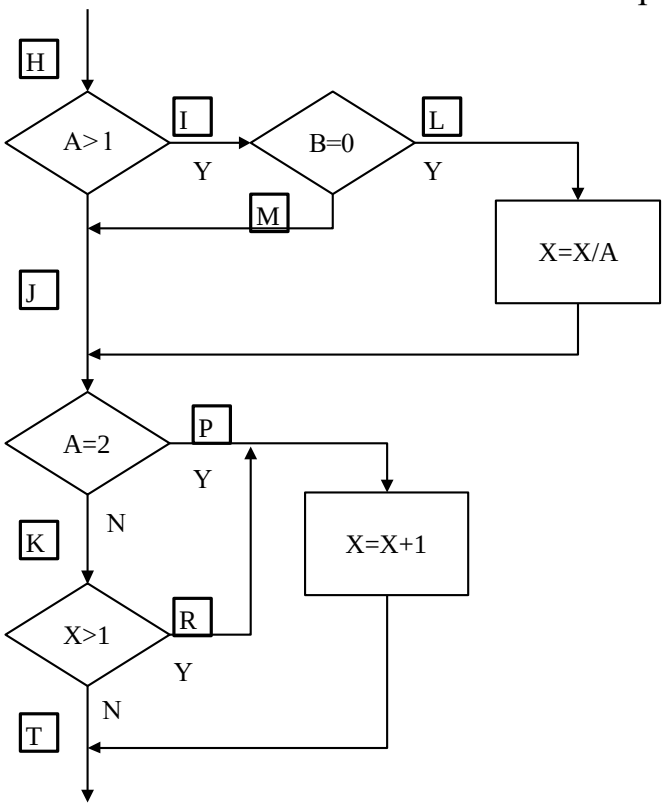

Рисунок 2.3

2.5 Метод комбинаторного покрытия условий.

Критерием, который решает эти и некоторые другие проблемы, является комбинаторное покрытие условий. Он требует создания такого числа тестов, чтобы все возможные комбинации результатов условия в каждом решении выполнялись по крайней мере один раз. Набор тестов, удовлетворяющих критерию комбинаторного покрытия условий, удовлетворяет также и критериям покрытия решений, покрытия условий и покрытия решений/условий.

По этому критерию в рассматриваемом примере должны быть покрыты тестами следующие восемь комбинаций:

а) A>1, B=0;
б)A>1, B≠0;
в) A≤1, B=0;
г) A≤1, B≠0;
д) A=2, X>1;
е) A=2, X≤1;
ж) A≠2, X>1;
з) A≠2, X≤1;

Для того чтобы протестировать эти комбинации, необязательно использовать все 8 тестов. Фактически они могут быть покрыты четырьмя тестами:

- A=2, B=0, X=4 {покрывает а, д};
- A=2, B=1, X=1 {покрывает б, е};
- A=0,5, B=0, X=2 {покрывает в, ж};
- A=1, B=0, X=1 {покрывает г, з}.

Таблица 2.4 - Результаты тестирования методом комбинаторного покрытия условий

Тест	Ожидаемый результат	Фактический результат	Результат тестирования
A=2, B=0, X=4	X=3	X=3	неуспешно
A=2, B=1, X=1	X=2	X=1,5	успешно
A=0,5 B=0 X=2	X=3	X=4	успешно
A=1, B=0, X=1	X=1	X=1	неуспешно

Методика выполнения лабораторной работы

1. Проанализировать программу, реализующую заданный алгоритм обработки данных – по индивидуальным вариантам из сборника задач. (https://coolcode.ru/sostavnyie-tipyi-dannyih-v-protsedurah-i-funktsiyah-gruppa-param/) Номер задачи соответствует номеру студента в журнале учёта посещаемости.
2. Отобразить алгоритм решения задачи в виде схемы программы (см. /2,3/).
3. Обозначить буквами или цифрами ветви алгоритма

4. Отладить программу и провести тестирование программного продукта рассмотренными методами.

5. Выписать пути алгоритма, которые должны быть проверены тестами для рассматриваемого метода тестирования.

6. Записать тесты, которые позволят пройти по путям алгоритма,.

7. Протестировать разработанную Вами программу. Результаты оформить в виде таблиц (таблицы 2.1-2.4).

8. Оформить отчет по лабораторной работе.

4 Содержание отчета

1. Цель работы.
2. Программа решения поставленной Вам задачи.
3. Схема программы (см. пп.2,3).
4. Таблицы тестирования программы (п.7).
6. Выводы по результатам тестирования (целью тестирования является обнаружение ошибок в программе).