Министерство образования и науки Российской Федерации
Вологодский государственный университет

Кафедра автоматики и вычислительной техники

ТЕСТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Методические указания к выполнению практических работ «Проектирование тестов»

Факультет электроэнергетический

Направления подготовки:
09.03.04 – Программная инженерия
09.03.01 – Информатика и вычислительная техника

Вологда
2014

УДК 681.3.06

Тестирование программного обеспечения: методические указания к выполнению практических работ «Проектирование тестов». - Вологда, ВоГУ, 2014. - 19 с.

Методические указания предназначены для проведения двух практических работ. В них имеется необходимый теоретический материал о способах подготовки наборов тестовых данных для функционального и структурного тестирования, приводятся примеры применения перечисленных методов и критериев тестирования, включены задачи для самостоятельной работы и вопросы для самоконтроля.

Утверждено редакционно-издательским советом ВоГУ

Составитель А.П. Сергушичева, канд. техн. наук, доцент

Рецензент А.Н. Швецов, д-р техн. наук, профессор

Практическая работа № 3. **Составление наборов тестовых данных для функционального тестирования. Стратегия "черного ящика"**

Основные теоретические положения

При функциональном тестировании программа рассматривается как «черный ящик», и целью тестирования является выяснение обстоятельств, в которых поведение программы не соответствует спецификации.

Исчерпывающее тестирование, т. е. тестирование на всех возможных наборах данных, в большинстве случаев невозможно (потребовались бы гигантские затраты времени и средств). А для программ, где исполнение команды зависит от предшествующих ей событий, необходимо также проверить и все возможные последовательности! Но, с другой стороны, необходимо обеспечить требуемое качество программного обеспечения. Возникает вопрос: как обнаружить максимальное количество ошибок, ограничив при этом подмножество возможных входных данных?

Существуют различные методы формирования тестовых наборов. В числе основных **критериев «черного ящика»** следующие:

1) **покрытие функций.** Для каждой из функций, реализуемых программой, требуется подобрать и выполнить хотя бы один тест;

2) **покрытие классов входных данных.** Критерий тестирования классов входных данных требует классифицировать входные данные, разделить их на классы таким образом, чтобы все данные из одного класса были равнозначны с точки зрения проверки правильности программы. Считается, что если программа работает правильно на одном наборе входных данных из этого класса, то она будет правильно работать на любом другом наборе данных из этого же класса. Критерий требует выполнения хотя бы одного теста для каждого класса входных данных;

3) **покрытие классов выходных данных.** Аналогичен предыдущему критерию, только проверяются не входные данные, а выходные;

4) **покрытие области допустимых значений** (тестирование границ класса). Для переменной, возможные значения которой перечислены (ноты, цвет, пол, диагноз и т.п.), следует убедиться, что на все указанные значения программа реагирует правильно и не принимает вместо них никаких иных значений. Если класс допустимых значений представляет собой числовой диапазон, то выделяются нормальные условия (в середине класса), гра-

ничные (экстремальные) условия и исключительные условия (выход за границу класса);

5) **тестирование длины набора данных** (можно считать частным случаем тестирования области допустимых значений). Определяется допустимое количество элементов в наборе. Если программа последовательно обрабатывает элементы некоторого набора данных, имеет смысл проверить следующие ситуации:
   - пустой набор (не содержит ни одного элемента);
   - единичный набор (состоит из одного-единственного элемента);
   - слишком короткий набор (если предусмотрена минимально допустимая длина);
   - набор минимально возможной длины (если такая предусмотрена); нормальный набор (состоит из нескольких элементов);
   - набор из нескольких частей (если такое возможно. Например, если программа читает литеры из текстового файла или печатает текст, то как она отнесется к переходу на следующую строку? На следующую страницу?);
   - набор максимально возможной длины (если такая предусмотрена);
   - слишком длинный набор (с длиной больше максимально допустимой);

6) тестирование **упорядоченности набора данных**. Важно для задач сортировки и поиска экстремумов. В этом случае имеет смысл проверить следующие ситуации (классы входных данных): данные неупорядочены; данные упорядочены в прямом порядке; данные упорядочены в обратном порядке; в наборе имеются повторяющиеся значения; экстремальное значение находится в середине набора; экстремальное значение находится в начале набора; экстремальное значение находится в конце набора; в наборе несколько совпадающих экстремальных значений.

Критерии покрытия функций, классов входных и выходных данных хорошо согласуются друг с другом и обычно каждой из функций ПП соответствуют свои классы входных и выходных данных. Например, программа учета кадров предприятия имеет функции: принять на работу, уволить с работы, перевести с одной должности на другую, выдать кадровую сводку. Соответственно классы входных данных – приказы о приеме, об увольнении, о переводе, заявка на кадровую сводку, а выходных – записи о приеме, об увольнении, о переводе, кадровая сводка.

В данной практической работе предлагается рассмотреть два **метода формирования тестовых наборов**: 1) на основе классов эквивалентности и 2) на основе граничных значений классов эквивалентности. Оба метода основаны на исследовании входных данных. Они не позволяют проверять результаты, получаемые при различных сочетаниях данных. Для построения тестов, проверяющих сочетания данных, применяют методы, использующие булеву алгебру.

Эквивалентное разбиение. Метод эквивалентного разбиения заключается в следующем. Область всех возможных наборов входных данных программы по каждому параметру разбивают на конечное число групп - классов эквивалентности. Наборы данных такого класса объединяют по принципу обнаружения одних и тех же ошибок: если набор какого-либо класса обнаруживает некоторую ошибку, то предполагается, что все другие тесты этого класса эквивалентности тоже обнаружат эту ошибку и наоборот.

Разработку тестов методом эквивалентного разбиения осуществляют в два этапа: на первом выделяют классы эквивалентности, а на втором - формируют тесты.

Формирование классов эквивалентности является эвристическим процессом, однако целесообразным считают выделять в отдельные классы эквивалентности наборы, содержащие допустимые и недопустимые значения некоторого параметра. При этом существует ряд правил:

- если некоторый параметр x может принимать значения в интервале [1, 999], то выделяют один правильный класс $1 < x < 999$ и два неправильных: $x < 1$ и $x > 999$;
- если входное условие определяет диапазон значений порядкового типа, например, «в автомобиле могут ехать от одного до шести человек», то определяется один правильный класс эквивалентности и два неправильных: ни одного и более шести человек;
- если входное условие описывает множество входных значений и есть основания полагать, что каждое значение программист трактует особо, например, «типы графических файлов: bmp, jpeg, vsd», то определяют правильный класс эквивалентности для каждого значения и один неправильный класс, например, txt;
- если входное условие описывает ситуацию «должно быть», например, «первым символом идентификатора должна быть буква», то определяется один правильный класс эквивалентности (первый символ - буква) и один неправильный (первый символ - не буква);
- если есть основание считать, что различные элементы класса эквивалентности трактуются программой неодинаково, то данный класс разбивается на меньшие классы эквивалентности.

Таким образом, классы эквивалентности выделяют, перебирая ограничения, установленные для каждого входного значения в техническом задании или при уточнении спецификации. Каждое ограничение разбивают на две или более групп. При этом используют специальные бланки - таблицы классов эквивалентности:

Ограничение на значение параметра	Правильные классы эквивалентности	Неправильные классы эквивалентности

Правильные классы включают правильные данные, неправильные классы - неправильные данные. Для правильных и неправильных классов тесты проектируют отдельно. При построении тестов правильных классов учитывают, что каждый тест должен проверять по возможности максимальное количество различных входных условий. Такой подход позволяет минимизировать общее число необходимых тестов. Для каждого неправильного класса эквивалентности формируют свой тест. Последнее обусловлено тем, что определенные проверки с ошибочными входами скрывают или заменяют другие проверки с ошибочными входами.

Анализ граничных значений. *Граничные значения* - это значения на границах классов эквивалентности входных значений или около них. Анализ показывает, что в этих местах резко увеличивается возможность обнаружения ошибок. Например, если в программе анализа вида треугольника было записано A + B ≥ C вместо A + B > C, то задание граничных значений приведет к ошибке: линия будет отнесена к одному из видов треугольника.

Применение метода анализа граничных значений требует определенной степени творчества и специализации в рассматриваемой проблеме. Тем не менее, существует несколько общих правил для применения этого метода:

- если входное условие описывает область значений, то следует построить тесты для границ области и тесты с неправильными входными данными для ситуаций незначительного выхода за границы области, например, если описана область [-1.0, +1.0], то должны быть сгенерированы тесты: -1.0, +1.0,-1.001 и +1.001;
- если входное условие удовлетворяет дискретному ряду значений, то следует построить тесты для минимального и максимального значений и тесты, содержащие значения большие и меньшие этих двух значений, например, если входной файл может содержать от 1 до 255 записей, то следует проверить О, 1, 255 и 256 записей;
- если существуют ограничения выходных значений, то целесообразно аналогично тестировать и их: конечно, не всегда можно получить ре-

зультат вне выходной области, но тем не менее стоит рассмотреть эту возможность;

– если некоторое входное или выходное значение программы является упорядоченным множеством, например, это последовательный файл, линейный список или таблица, то следует сосредоточить внимание на первом и последнем элементах этого множества.

Помимо указанных граничных значений, целесообразно поискать другие. Однако следует помнить, что граничные значения могут быть едва уловимы и определение их связано с большими трудностями, что является недостатком этого метода.

Пример. Пусть необходимо выполнить тестирование программы, определяющей точку пересечения двух прямых на плоскости. При этом она должна определять параллельность прямой одной из осей координат.

В основе программы лежит решение системы линейных уравнений:

$$Ax + By = C,\ \ Dx + Ey = F.$$

По методу эквивалентных разбиений формируем для каждого коэффициента один правильный класс эквивалентности (коэффициент –вещественное число) и один неправильный (коэффициент – не вещественное число). Откуда генерируем 7 тестов:

1) все коэффициенты - вещественные числа (1 тест);

2-7) поочередно каждый из коэффициентов - не вещественное число (6 тестов).

По методу граничных значений можно считать, что для исходных данных граничные значения отсутствуют, т. е. коэффициенты - «любые» вещественные числа.

Для результатов получаем, что возможны варианты: единственное решение, прямые сливаются - множество решений, прямые параллельны - отсутствие решений. Следовательно, целесообразно предложить тесты с результатами внутри областей возможных значений результатов:

8) результат - единственное решение ($\delta \neq 0$ (определитель δ=AE-BD));

9) результат - множество решений ($\delta = 0$ и $\delta_X = \delta_Y = 0$, (δ_X=CE-BF, δ_Y=AF-CD));

10) результат - отсутствие решений ($\delta = 0$, но $\delta_X \neq 0$ или $\delta_Y \neq 0$);

и с результатами на границе:

11) $\delta = 0{,}01$;

12) $\delta = -0{,}01$;

13) $\delta = 0$, $\delta_X = 0{,}01$, $\delta = 0$;

14) $\delta = 0$, $\delta_Y = -0{,}01$, $\delta_X = 0$.

Задачи

Задача 3.1. Есть программа, которая интерпретирует три целых числа, вводимых с клавиатуры, как длины сторон треугольника и выводит сообщение, о том, какой это треугольник: равносторонний, равнобедренный или неравносторонний. Напишите на листе бумаги тесты (последовательности входных данных и ожидаемые результаты), которые, как вам кажется, будут адекватно проверять эту программу. Рекомендуется запись тестов в виде таблицы:

Критерий формирования теста	№ теста	Исходные данные			Ожидаемый результат
		a	b	c	

Задача 3.2. В программе «Деканат» переменная, обозначающая количество студентов в группе, может принимать значения от 1 до 30. Составьте тестовый набор для этой переменной.

Задача 3.3. В компьютерной обучающей системе тестовые задания для контроля знаний берутся из файла типа .txt, где каждое задание занимает одну строку. Для формирования теста указывается имя файла, количество заданий в тесте и количество вариантов теста (не более 10). Варианты должны различаться не менее чем тремя заданиями. Если заданий в файле недостаточно для реализации этого требования, выдается сообщение «Недостаточно заданий», если файл не найден - «файл отсутствует». Увидеть составленный вариант теста для контроля знаний можно, нажав на соответствующую кнопку «Вариант №».

Составьте тестовые наборы для проверки перечисленных функций. Рекомендуется оформить их в виде таблицы, подобно задаче 3.1.

Задача 3.4. Используя методы эквивалентного разбиения и граничных значений, ***составьте*** тестовые наборы для проверки перечисленных ниже функций программы "Геометрические фигуры". Протестируйте указанную программу.

В программе "Геометрические фигуры" (файл geometry.exe) после ее запуска пользователю предоставляется выбрать вариант геометрической программы. Для выбора пользователь должен нажать на кнопку с названием нужного варианта. Предусмотрены следующие варианты:

Вариант 1. Программа определяет тип треугольника. Возможные результаты: прямоугольный, остроугольный, тупоугольный, равнобедренный, правильный (равносторонний)

Вариант 2. Четырехугольник задается координатами вершин. Программа проверяет, является ли он квадратом

Вариант 3. Четырехугольник задается координатами вершин. Программа проверяет, является ли он ромбом

Вариант 4. Определение взаимного положения прямой и окружности. Прямая описывается уравнением Y=kX+b. Окружность с центром в начале координат задается радиусом R. Результат – линии не пересекаются, пересекаются в двух точках, прямая линия является касательной к окружности.

Вариант 5. Определение взаимного положения двух окружностей. Окружности задаются координатами центра X,Y и радиусом R. Результат – линии не пересекаются, пересекаются в двух точках, касаются в одной точке, совпадают.

Контрольные вопросы

1. Почему стратегия функционального тестирования называется также стратегией "черного ящика"?
2. По каким критериям осуществляется функциональное тестирование?
3. В чем заключается метод эквивалентного разбиения?
4. По какому принципу формируют классы эквивалентности?
5. Почему анализ граничных значений считается одним из наиболее полезных методов проектирования тестов?.

Практическая работа № 4. Составление наборов тестовых данных для структурного тестирования. Стратегия "белого (прозрачного) ящика"

Основные теоретические положения

При использовании стратегии «белого ящика» тестовые наборы формируют путем анализа маршрутов, предусмотренных алгоритмом. Под ***маршрутами*** при этом понимают последовательности операторов программы, которые выполняются при конкретном варианте исходных данных. Тестирование, проводимое по составленным таким образом тестам, называют структурным или тестированием по маршрутам.

В основе структурного тестирования лежит концепция максимально полного тестирования всех маршрутов программы. Так, если алгоритм программы включает ветвление, то при одном наборе исходных данных может быть выполнена последовательность операторов, реализующая действия, которые предусматривает одна ветвь, а при втором – другая. Соответственно, для программы будут существовать маршруты, различающиеся выбранным при ветвлении вариантом.

Считают, что программа проверена полностью, если с помощью тестов удается осуществить выполнение программы по всем возможным маршрутам передач

управления. Однако нетрудно видеть, что даже в программе среднего уровня сложности число неповторяющихся маршрутов может быть очень велико, и, следовательно, полное или исчерпывающее тестирование маршрутов, как правило, невозможно.

Последовательность составления тестов следующая:

1. На основе алгоритма (текста) программы формируется потоковый граф (или граф-схема). Узлы (вершины) потокового графа соответствуют линейным участкам программы, включают один или несколько операторов программы. Дуги (ориентированные ребра) потокового графа отображают поток управления в программе (передачи управления между операторами). Различают операторные и предикатные узлы. Из операторного узла выходит одна дуга, а из предикатного – две дуги. *Предикатные узлы соответствуют* ***простым*** *условиям в программе.* Составное условие программы (условие, в котором используется одна или несколько булевых операций (OR, AND)) отображается в несколько предикатных узлов. На рис. 4.1 показан пример такого преобразования для фрагмента программы: *if a OR b then x else y end if.*

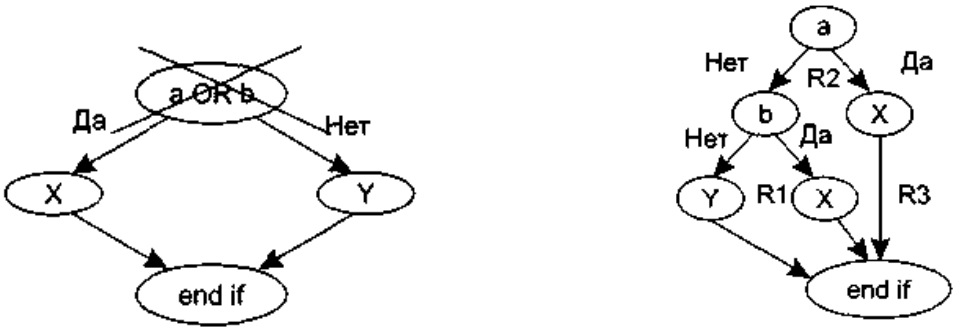

Рис. 4.1. Прямое отображение фрагмента программы в потоковый граф и преобразованный потоковый граф

2. Выбирается критерий тестирования. Формирование тестовых наборов для тестирования маршрутов может осуществляться по нескольким критериям: покрытие маршрутов, покрытие операторов, покрытие решений (переходов), покрытие условий, покрытие решений/условий, комбинаторное покрытие условий, покрытие потоков данных, покрытие циклов.

3. Подготавливаются тестовые варианты, инициирующие выполнение каждого пути. Каждый тестовый вариант формируется в следующем виде:

Исходные данные (ИД):

Ожидаемые результаты (ОЖ.РЕЗ.):

Тесты, составленные по критерию **покрытие маршрутов** и обеспечивающие проверку базового множества, гарантируют однократное выполнение каждого оператора и выполнение каждого условия по True-ветви и по False-ветви. Базовое множество образуют все независимые пути графа. Независимым называется любой путь от начального до конечного узла графа, который вводит новый оператор обработки или новое условие. В терминах потокового графа независимый путь должен содержать дугу, не входящую в ранее определенные пути. Независимые пути формируются в порядке от самого короткого к самому длинному.

Число независимых линейных путей в базовом множестве определяется ***цикломатической сложностью алгоритма***, которая вычисляется одним из трех способов:

1) цикломатическая сложность V(G) равна количеству регионов потокового графа; Регион - замкнутая область, образованная дугами и узлами. Окружающая граф среда рассматривается как дополнительный регион. Например, показанный на рис. 4.1 граф имеет три региона – Rl, R2, R3;

2) цикломатическая сложность определяется по формуле

$$V(G) = E-N+2,$$

где E – количество дуг, N – количество узлов потокового графа;

3) цикломатическая сложность V(G) = p +1, где p – количество предикатных узлов G.

Пример 4.1.

Цикломатическая сложность алгоритма на рис.4.1.:

1) V(G) = 3 региона;
2) V(G) = 7 дуг - 6 узлов + 2 = 3;
3) V(G) = 2 предикатных узлов + 1 = 3.

Независимые пути:

Путь 1: a - x – end if.
Путь 2: a - b - x – end if.
Путь 3: a - b - y – end if.

Критерий **покрытия операторов** подразумевает такой подбор тестов, чтобы каждый оператор программы выполнялся, по крайней мере, один раз. Это необходимое, но недостаточное условие для приемлемого тестирования.

Для реализации критерия **покрытие решений (переходов)** необходимо такое количество и состав тестов, чтобы результат проверки каждого условия (т.е. решение) принимал значения «истина» или «ложь», по крайней мере, один раз. Нетрудно видеть, что критерий покрытия решений удовлетворяет критерию покрытия операторов, но является более «сильным».

Критерий **покрытия условий** является еще более «сильным» по сравнению с предыдущими. В этом случае формируют некоторое количество тестов, достаточное для того, чтобы все возможные результаты каждого условия в решении были выполнены, по крайней мере, один раз. Однако, как и в случае покрытия решений, этот критерий не всегда приводит к выполнению каждого оператора, по крайней мере, один раз. К критерию требуется дополнение, заключающееся в том, что каждой точке входа управление должно быть передано, по крайней мере, один раз.

Согласно **покрытию решений/условий** тесты должны составляться так, чтобы, по крайней мере, один раз выполнились все возможные результаты каждого условия и все результаты каждого решения, и каждому оператору управление передавалось, по крайней мере, один раз.

Критерий **комбинаторное покрытие условий** требует создания такого множества тестов, чтобы все возможные комбинации результатов условий в каждом решении и все операторы выполнялись, по крайней мере, один раз.

Пример 4.2.

Требуется выполнить структурное тестирование текста программы, которая определяет значение x в зависимости от значений параметров процедуры.

```
Procedure m (a, b:real; var x:real);
    begin
        if(a>1) and (b=0) then x:=x /a;
          if(a=2) or (x>l) then x:=x+1;
    end;
```

Для формирования тестов программу представляют в виде графа, вершины которого соответствуют операторам программы, а дуги представляют возможные варианты передачи управления (рис.4.2).

Покрытие операторов будет реализовано при a = 2, b = 0, x = 3.

Однако, хотя исходные данные заданы так, чтобы все операторы программы были выполнены хотя бы один раз, для проверки программы этого явно недостаточно. Например, из второго условия следует, что переменная x может принимать любое значение, и в некоторых версиях языка Pascal это значение проверяться не будет. Кроме того, если при написании программы в первом условии указано, что (a > 1) or (b = 0), или, если во втором условии вместо x > 1 записано x > 0, то эти ошибки обнаружены не будут. Также существует путь 1-2-4-6, в котором x вообще не меняется и, если здесь есть ошибка, она не будет обнаружена.

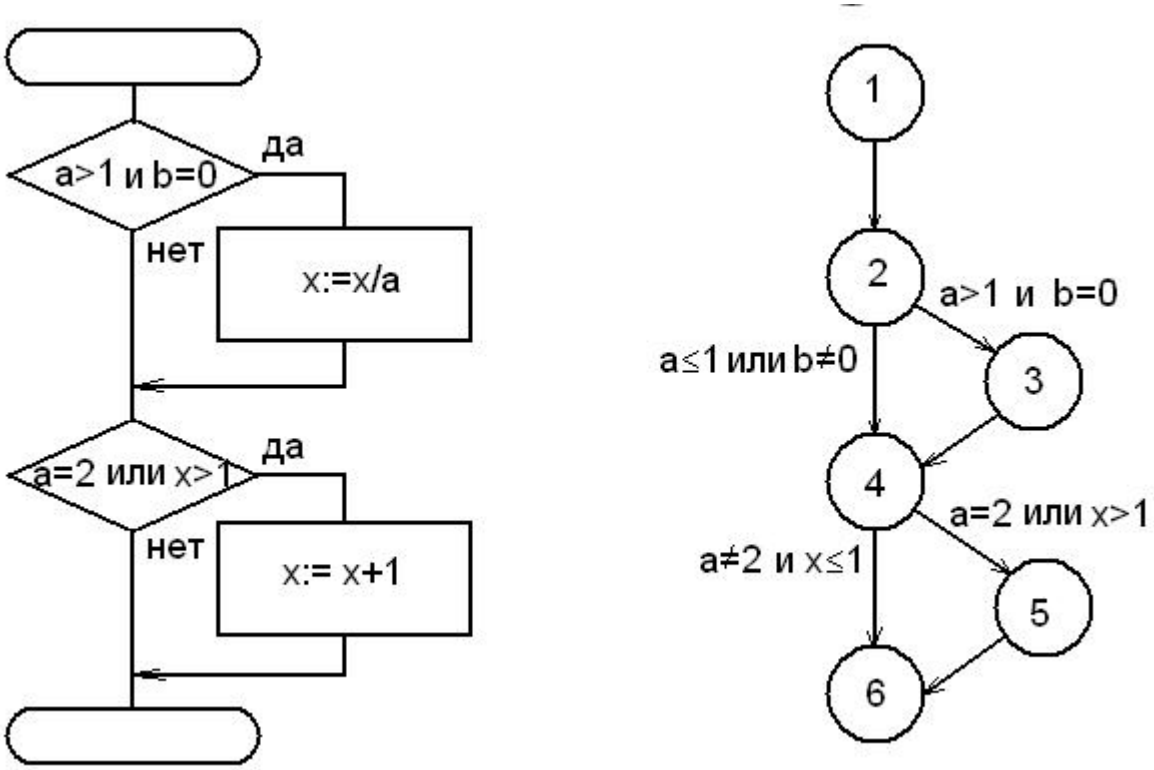

Рис. 4.2. Схема алгоритма процедуры (слева) и ее граф передач управления

По методу **покрытия решений (переходов)** рассматриваемую программу можно протестировать двумя тестами, покрывающими либо пути: 1-2-4-6, 1-2-3-4-5-6, либо пути: 1-2-3-4-6, 1-2-4-5-6, например:

a = 3, b = 0, x = 3 — путь 1-2-3-4-5-6;

a = 2, b = 1 , x = 1 — путь 1-2-4-5-6

Однако путь, где x не меняется, будет проверен с вероятностью 50 %: если во втором условии вместо x > 1 записано x < 1, то этими двумя тестами ошибка обнаружена не будет.

Покрытие условий проверяет четыре условия:

l) a>l; 2) b = 0; 3) a = 2; 4) x>1.

Необходимо реализовать все возможные ситуации:

Тесты, удовлетворяющие этому условию:

a= 2, b = 0, x = 4 — путь 1-2-3-4-5-6, условия: 1 - да, 2 - да, 3-да, 4-да

a = 1, b = 1, x = 1 — путь 1-2-4-6, условия: 1 - нет, 2 - нет, 3 – нет, 4-нет.

Критерий покрытия условий часто удовлетворяет критерию покрытия решений, но не всегда. Тесты критерия покрытия условий для ранее рассмотренных примеров покрывают результаты всех решений, но это случайное совпадение. Например, тесты:

a=1,Ь = 0, x = 3 — путь 1-2-3-6, условия: 1 - нет, 2 - да, 3 - нет, 4 - да;

a = 2, b = 1, x = 1 — путь 1-2-3-4-5-6, условия: 1 - да, 2 - нет, 3 - да, 4 - нет

покрывают результаты всех условий, но только два из четырех результатов решений: не выполняется результат «истина» первого решения и результат «ложь» второго.

Основной недостаток метода – недостаточная чувствительность к ошибкам в логических выражениях.

Покрытие решений/условий

Анализ, проведенный выше, показывает, что этому критерию удовлетворяют тесты:

a = 2, Ь=0, x=4 – путь 1-2-3-4-5-6, условия: 1 - да, 2 - да, 3 - да, 4 - да;
a=1, b=1, x=1 – путь 1-2-4-6, условия: 1 - нет, 2 - нет, 3 - нет, 4 - нет.

Комбинаторное покрытие условий требует покрыть тестами восемь комбинаций:

1) a>1,b = 0;
2) a>1, b ≠ 0;
3) a<1, b = 0;
4) a<1, b ≠ 0
5) a = 2,x>1;
6) a = 2, x<1;
7) a ≠ 2, x>1;
8) a ≠ 2, x<1.

Эти комбинации можно проверить четырьмя тестами:

a = 2, b = 0, x = 4 — проверяет комбинации (1), (5);
a = 2, b = 1, x = 1 — проверяет комбинации (2), (6);
a= 1, b = 0, x = 2 — проверяет комбинации (3), (7);
a=1,Ь=1,x=1— проверяет комбинации (4), (8).

В данном случае то, что четырем тестам соответствуют четыре пути, является совпадением. Представленные тесты не покрывают всех путей, например, acd. Поэтому иногда необходима реализация восьми тестов.

Таким образом, для программ, содержащих только одно условие на каждое решение, минимальным является набор тестов, который проверяет все результаты каждого решения и передает управление каждому оператору, по крайней мере, один раз.

Для программ, содержащих вычисления, каждое из которых требует проверки более чем одного условия, минимальный набор тестов должен:

- генерировать все возможные комбинации результатов проверок условий для каждого вычисления;
- передавать управление каждому оператору, по крайней мере, один раз.

Термин «возможных» употреблен здесь потому, что некоторые комбинации условий могут быть нереализуемы. Например, для комбинации k<0 и k>40 задать k невозможно.

Тестовое покрытие циклов

Цикл — наиболее распространенная конструкция алгоритмов, реализуемых в программном обеспечении. При проверке циклов основное внимание обращается на правильность конструкций циклов. Количество наборов тестов для проверки циклов по принципу «белого ящика» зависит от типа цикла. Различают 4 типа циклов: простые, вложенные, объединенные, неструктурированные.

Для проверки ***простых*** циклов с количеством повторений п может использоваться один из следующих наборов тестов:

1) прогон всего цикла;
2) только один проход цикла;
3) два прохода цикла;
4) т проходов цикла, где т<п;
5) п - 1, п, п + 1 проходов цикла.

С увеличением уровня **вложенности** циклов количество возможных путей резко возрастает. Это приводит к нереализуемому количеству тестов. Для сокращения количества тестов применяется специальная методика, использующая понятия объемлющего и вложенного циклов (рис. 4.3).

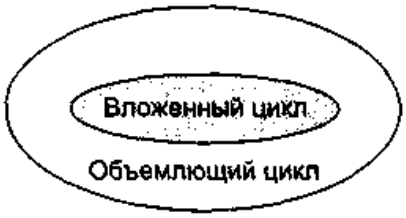

Рис. 4.3. Объемлющий и вложенный циклы

Порядок тестирования вложенных циклов следующий:.

– Выбирается самый внутренний цикл. Устанавливаются минимальные значения параметров всех остальных циклов.

– Для внутреннего цикла проводятся тесты простого цикла. Добавляются тесты для исключенных значений и значений, выходящих за пределы рабочего диапазона.

– Переходят в следующий по порядку объемлющий цикл. Выполняют его тестирование. При этом сохраняются минимальные значения параметров для всех внешних (объемлющих) циклов и типовые значения для всех вложенных циклов.

– Работа продолжается до тех пор, пока не будут протестированы все циклы.

Если каждый из циклов независим от других **(объединенные циклы)**, то используется техника тестирования простых циклов. При наличии зависимости (например, конечное значение счетчика первого цикла используется как начальное значение счетчика второго цикла) используется методика для вложенных циклов.

Неструктурированные циклы тестированию не подлежат. Этот тип циклов должен быть переделан с помощью структурированных программных конструкций.

Задачи

Задача 5.1. Определить цикломатическую сложность потоковых графов, представленных на рис.4.4.

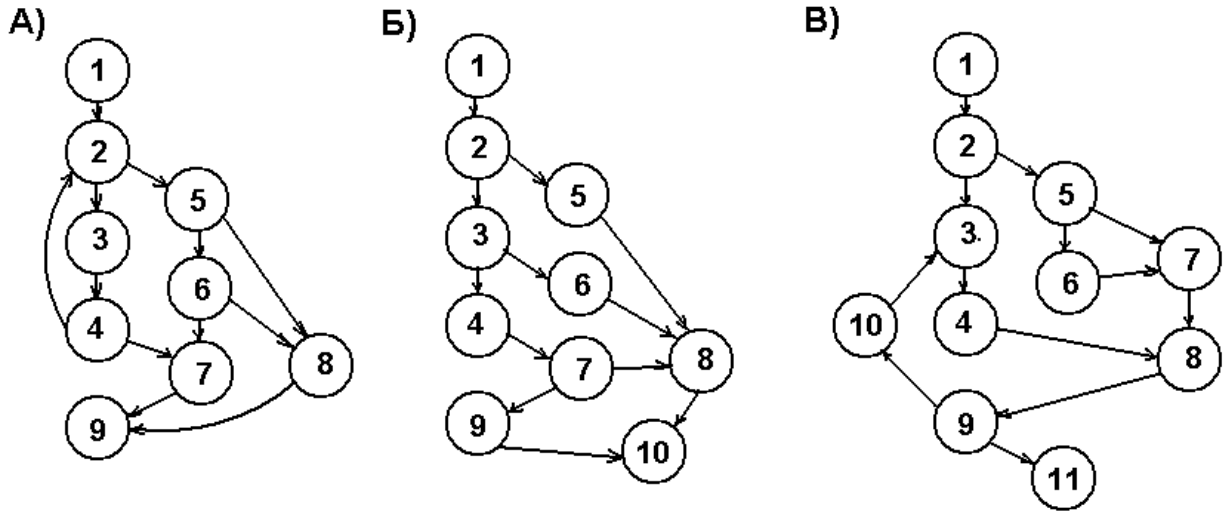

Рис. 4.4. Потоковые графы

Задача 5.2. В соответствии **с** концепцией максимально полного тестирования всех маршрутов программы определить независимые пути потоковых графов, представленных на рис.4.4.

Задача 5.3. Для заданной процедуры (варианты 1-6) составить потоковый граф, определить цикломатическую сложность потокового графа по каждой из трех формул и составить тестовые наборы по критерию покрытия маршрутов.

Задача 5.4. Для заданной процедуры (варианты 1-6) составить граф-схему и тестовые наборы для тестирования маршрутов по критериям:

1. покрытия операторов;
2. покрытия решений (переходов);
3. покрытия условий;
4. покрытия решений/условий;
5. комбинаторного покрытия условий.

Проанализируйте целесообразность каждого из критериев для своей программы, укажите их недостатки, достоинства и преимущества над другими критериями.

Задача 5.5. Определить типы циклов в потоковых графах, представленных на рис.4.4, 4.5.

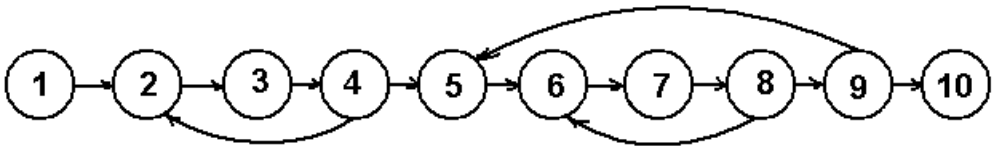

Рис. 4.5. Потоковый граф

Задача 5.5. Сколько наборов тестов необходимо для тестирования программы, потоковый граф которой представлен на рис. 4.5.

Варианты заданий для структурного тестирования

Вариант 1.

```
procedure m(a,b: real; var x: real)
begin
    if (a>0)and(b<0) then x:=x+1;
    if ((a=2)or(x>3))and(b>-10) then x:=x-1;
end;
```

Вариант 2.

```
procedure m(a,b,c: real; var x: real)
begin
    if (a>0)and(b<0)and(x>6) then x:=x+1;
    if (a=4)or(c<0) then x:=x+11;
end;
```

Вариант 3.

```
procedure m(a,b: real; var x: real)
begin
    if (a<=6)and(b<0) then x:=x+1;
    if (a=7)      then x:=x-1
                  else if (x>3) then  x=x*2;
end;
```

Вариант 4.

```
procedure m(a,b: real; var x: real)
begin
    if (a>0) then
      if(b<0)     then x:=x+1
                  else x=x*2;
    if (a>2)or(x=0) then x:=x+1;
end;
```

Вариант 5.

```
procedure m(a,b: real; var x: real)
begin
    if (a>0)and(b<0)  then x:=x+1
                      else if ((a=2)or(x>3))and(b>-10) then x:=x-1;
end;
```

Вариант 6.

```
procedure m(a,b,c: real; var x: real)
begin
    if  (a>0)and(b<0)and(x>6)
                then x:=x+1
                else if (a=4)or(c<0)
                                then x:=x+11;
end;
```

Контрольные вопросы

1. Почему стратегия структурного тестирования называется также стратегией "белого ящика"?
2. Что показывает цикломатическая сложность алгоритма?
3. В чем отличие критериев покрытия условий и покрытия решений/условий
4. Какой из критериев "белого ящика" считается самым сильным и почему?
5. Приведите порядок тестирования вложенных циклов.

Библиографический список

1. Бейзер, Б. Тестирование черного ящика : технологии функционального тестирования программного обеспечения и систем / Б. Бейзер . - СПб.: Питер , 2004 . - 317 с.

2. Гагарина, Л. Г. Технология разработки программного обеспечения : учебное пособие / Л. Г. Гагарина, Е. В. Кокорева, Б. Д. Виснадул; под ред. Л. Г. Гагариной . - М. : ФОРУМ , 2011. - 399 с.

3. Дастин,Э. Автоматизированное тестирование программного обеспечения/ Э.Дастин, Д.Рэшка, Д.Пол - М.: Лори, 2003.-567с.

4. Иванова, Г.С. Технология программирования: учебник для вузов / Г.С.Иванова. - М.:КНОРУС, 2011.-333с.

5. Орлов, С. Технологии разработки программного обеспечения: учебник / С.Орлов. – СПб.:Питер,2002. – 464с.

6. Орлов, С. А. Технологии разработки программного обеспечения: современный курс по программной инженерии: учебник для вузов / С.А. Орлов, Б. Я. Цилькер. - СПб.: Питер , 2012 . - 608 с.

7. Плаксин, М.А. Тестирование и отладка программ - для профессионалов будущих и настоящих/ М.А.Плаксин. - М.:БИНОМ, 2007.-167с.

8. Сергушичева, А.П. Жизненный цикл программного продукта: учебное пособие / А.П. Сергушичева. – Вологда: ВоГТУ, 2010. – 148 с

Подписано в печать 23.06.2014. Усл. печ. л. . Тираж экз.
Печать офсетная. Бумага офисная. Заказ № _____

Отпечатано: РИО ВоГУ, г. Вологда, ул. С. Орлова, 6.