Тестирование программного обеспечения

Инспектирование ПО

- **Визуальный контроль** - проверка программ в ручную, без использования компьютера.
- **Статический контроль** -проверка программы по тексту без ее выполнения с помощью инструментальных средств.
- **Динамический контроль** заключается в проверке правильности работы программы на компьютере (тестировании ПО).

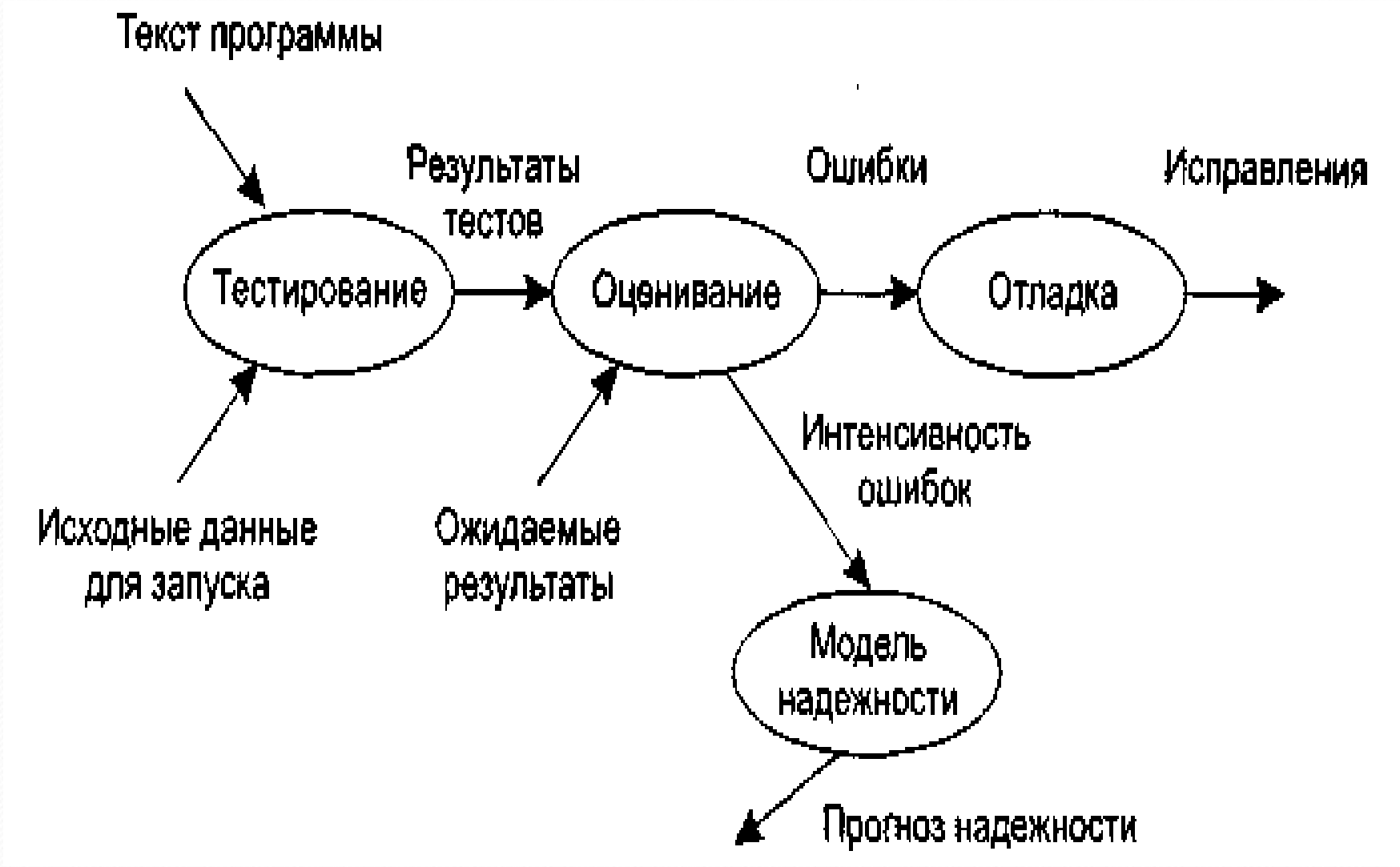
Виды программных ошибок и способы их обнаружения

Виды программных ошибок	Способы их обнаружения
Синтаксические	Статический контроль и диагностика компиляторами и компоновщиком
Ошибки выполнения, выявляемые автоматически: а) переполнение, защита памяти; б) несоответствие типов; в) зацикливание	Динамический контроль: а)аппаратурой процессора; б)run-time системы программирования в)операционной системой – по превышению лимита времени.
Программа не соответствует спецификации	Целенаправленное тестирование
Спецификация не соответствует требованиям	Испытания, бета-тестирование

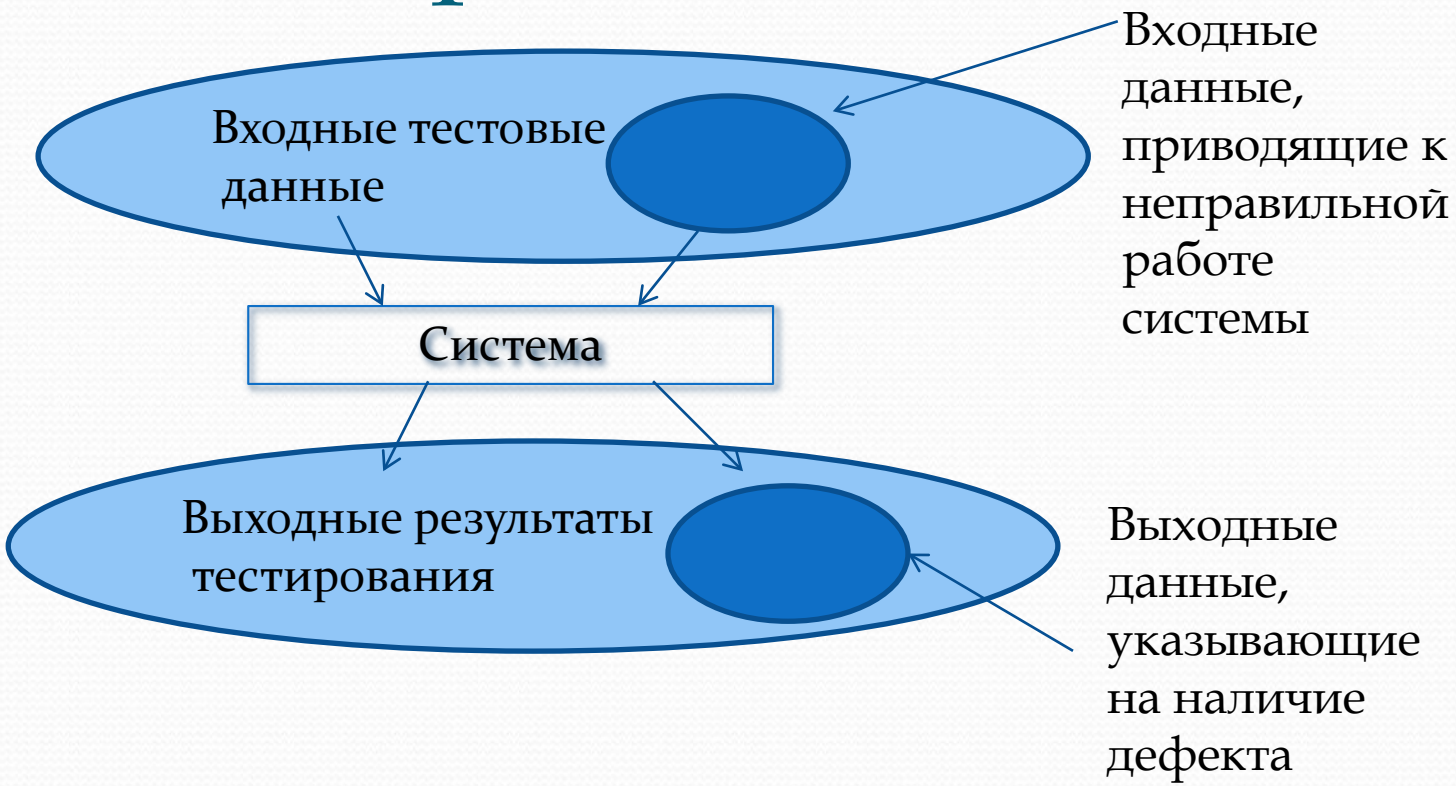
Уровни тестирования

- модульное тестирование;
- интеграционное тестирование.;
- системное тестирование.:
  - альфа-тестирование
  - бета-тестирование

Информационные потоки процесса тестирования

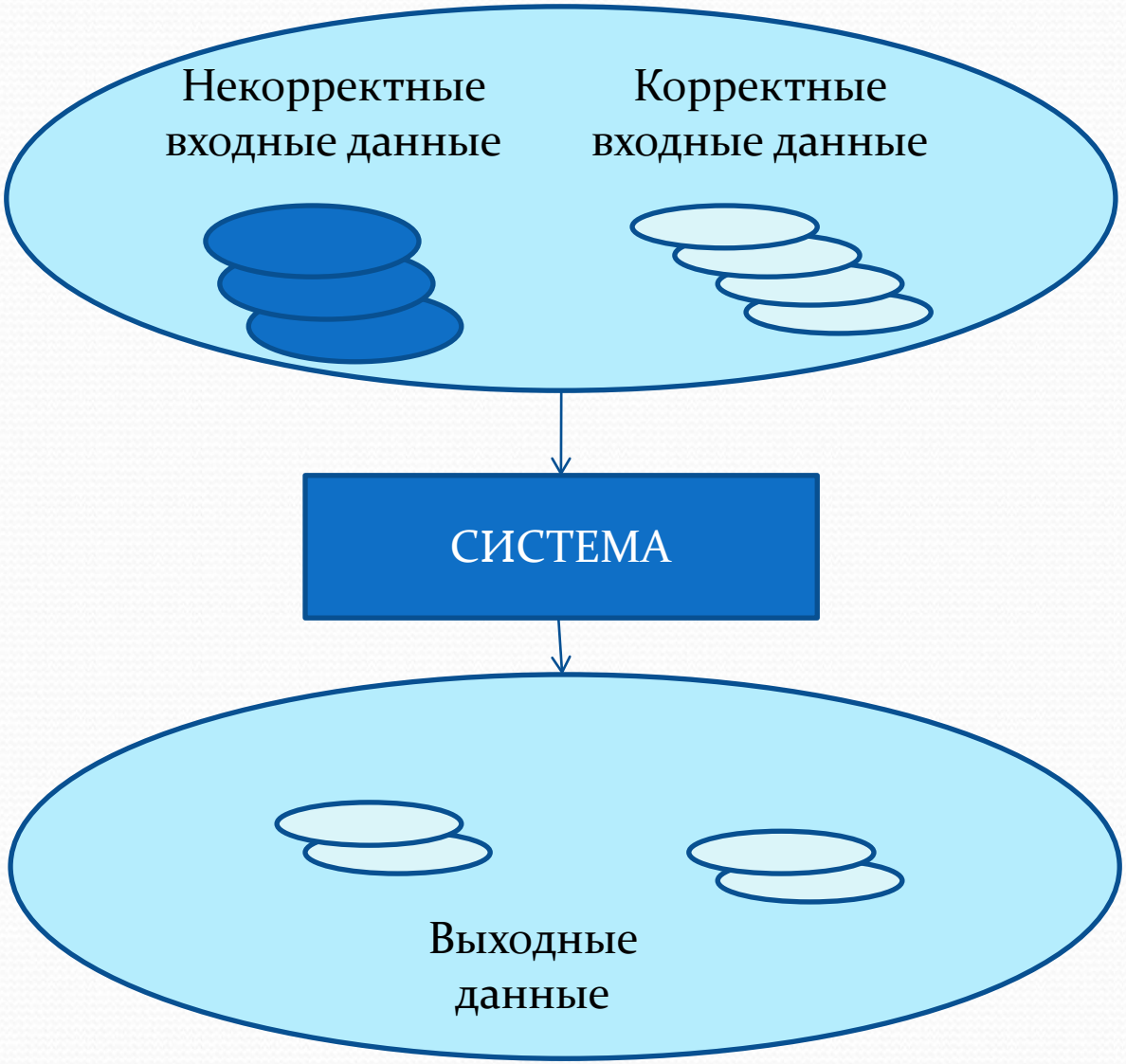


Функциональное тестирование



- Эквивалентное разбиение
- Анализ граничных значений
- Применение функциональных диаграмм
- Предположение об ошибке

Эквивалентное разбиение



Выделение классов эквивалентности

- Если входное условие описывает область значений, то определяются один правильный класс эквивалентности и два неправильных.
- Если входное условие описывает множество входных значений и есть основание полагать, что каждое значение программа трактует особо, то определяется правильный класс эквивалентности для каждого значения и один неправильный класс.
- Если входное условие описывает ситуацию "должно быть, то определяется один правильный класс эквивалентности и один неправильный.
- Если есть любое основание считать, что различные элементы класса эквивалентности трактуются программой неодинаково, то данный класс эквивалентности разбивается на меньшие классы эквивалентности

Пример построения теста

Спецификация

Оператор DIM используется для определения массивов.

DIM *ad* [, *ad*]…,

где *ad* есть описатель массива в форме *n*(*d*[,*d*]...),

n – символическое имя массива,

d – индекс массива.

Символические имена могут содержать от одного до шести символов - букв или цифр, причем первой должна быть буква.

Допускается от одного до семи индексов.

Форма индекса [lb:] ub,

где lb и ub задают нижнюю и верхнюю границы индекса массива.

Граница может быть либо константой, принимающей значения от - 65534 до 65535, либо целой переменной (без индексов).

Если lb не определена, то предполагается, что она равна единице. Значение ub должно быть больше или равно lb.

Если lb определена, то она может иметь отрицательное, нулевое или положительное значение.

Оператор может располагаться на нескольких строках

Входные условия	Правильные классы эквивалентности	Неправильные классы эквивалентности	Тестовые наборы для неправильных классов
Число описателей массивов	ОДИН (1), > ОДНОГО (2)	НИ ОДНОГО (3)	DIM
Длина имени массива	1-6(4)	0(5), > 6(6)	DIM (10) DIM A234567(2)
Имя массива	Имеет в своем составе буквы (7) и цифры (8)	содержит что-то еще (9)	DIM A.I(2)
Имя массива начинается с буквы	да (10)	нет (11)	DIM 1A(10)
Число индексов	1-7(12)	0(13), > 7(14)	DIM B DIM B (4,4,4,4,4,4,4,4)
Верхняя граница	Константа (15), целая переменная (16)	имя элемента массива (17), что-то иное (18)	DIM B(4,A(2)) DIM B(4,A(2))

Имя целой переменной	Имеет в своем составе буквы (19), и цифры (20)	состоит из чего-то еще (21)	DIM C(1.,10)
Целая переменная начинается с буквы	да (22)	нет (23)	DIM C(10,1J)
Константа	От -65534 до 65535 (24))	Меньше -65534 (25), больше 65535 (26)	DIM D (-65535:1) DIM D(65536)
Нижняя граница определена	да (27), нет (28)		
Верхняя граница по отношению к нижней границе	Больше (29), равна (30)	меньше (31)	DIM D(4:3)
Значение нижней границы	Отрицательное (32) Нуль (33), > 0 (34)		
Нижняя граница	Константа (35), целая переменная (36)	имя элемента массива (37), что-то иное (38)	DIM D(A(2):4) DIM D(.:4)
Оператор расположен на нескольких строках	да (39), нет (40)		

Например, тест DIM A(2) покрывает классы 1, 4, 7, 10, 12, 15, 24, 28, 29 и 40.
DIM A12345(I,9,J4XXXX.65535,1,KLM, X 100),
BBB (-65534 : 100,0 : 1000,10 : 10,1 : 65535) покрывает оставшиеся классы.

(3)	DIMENSION
(5)	DIMENSION(10)
((6)	DIMENSION A234567(2)
(9)	DIMENSION A.I(2)
(11)	DIMENSION1A(10)
(13)	DIMENSION B
(14)	DIMENSION B (4,4,4,4,4,4,4,4)
(17)	DIMENSION B(4,A(2))
(18)	DIMENSION B(4„7)
(21)	DIMENSION C(1.,10)
(23)	DIMENSION C(10,1J)
(25)	DIMENSION D (-65535:1)
(26)	DIMENSION D(65536)
(31)	DIMENSION D(4:3)
(37)	DIMENSION D(A(2):4)
(38)	DIMENSION D(.:4)

Анализ граничных значений

Основные отличия анализа граничных значений от разбиения по эквивалентности:

1) тестовые варианты создаются для проверки только ребер классов эквивалентности;

2) при создании тестовых вариантов учитывают не только условия ввода, но и область вывода.

Правила анализа граничных значений.

1. Если условие ввода задает диапазон *n...m*, то тестовые варианты должны быть построены:

- для значений *n* и *m;*
- для значений чуть левее *n* и чуть правее *m* на числовой оси.

Например, если задан входной диапазон -1,0...+1,0, то создаются тесты для значений - 1,0, +1,0, - 1,001, +1,001.

2. Если условие ввода задает дискретное множество значений, то создаются тестовые варианты:

- для проверки минимального и максимального из значений;
- для значений чуть меньше минимума и чуть больше максимума.

Так, если входной файл может содержать от 1 до 255 записей, то создаются тесты для О, 1, 255, 256 записей.

3. Если внутренние структуры данных программы имеют предписанные границы, то разрабатываются тестовые варианты, проверяющие эти структуры на их границах.
4. Если входные или выходные данные программы являются упорядоченными множествами, то надо тестировать обработку первого и последнего элементов этих множеств.

Протестировать программу бинарного поиска. Нам известна *спецификация* этой программы. Поиск выполняется в массиве элементов *M*, возвращается индекс *I* элемента массива, значение которого соответствует ключу поиска *Key*.

Входные условия:

1) массив должен быть упорядочен;
2) массив должен иметь не менее одного элемента;
3) нижняя граница массива (индекс) должна быть меньше или равна его верхней границе.

Результаты:

1) если элемент найден, то флаг Result=True, значение I — номер элемента;
2) если элемент не найден, то флаг Result=False, значение I не определено.

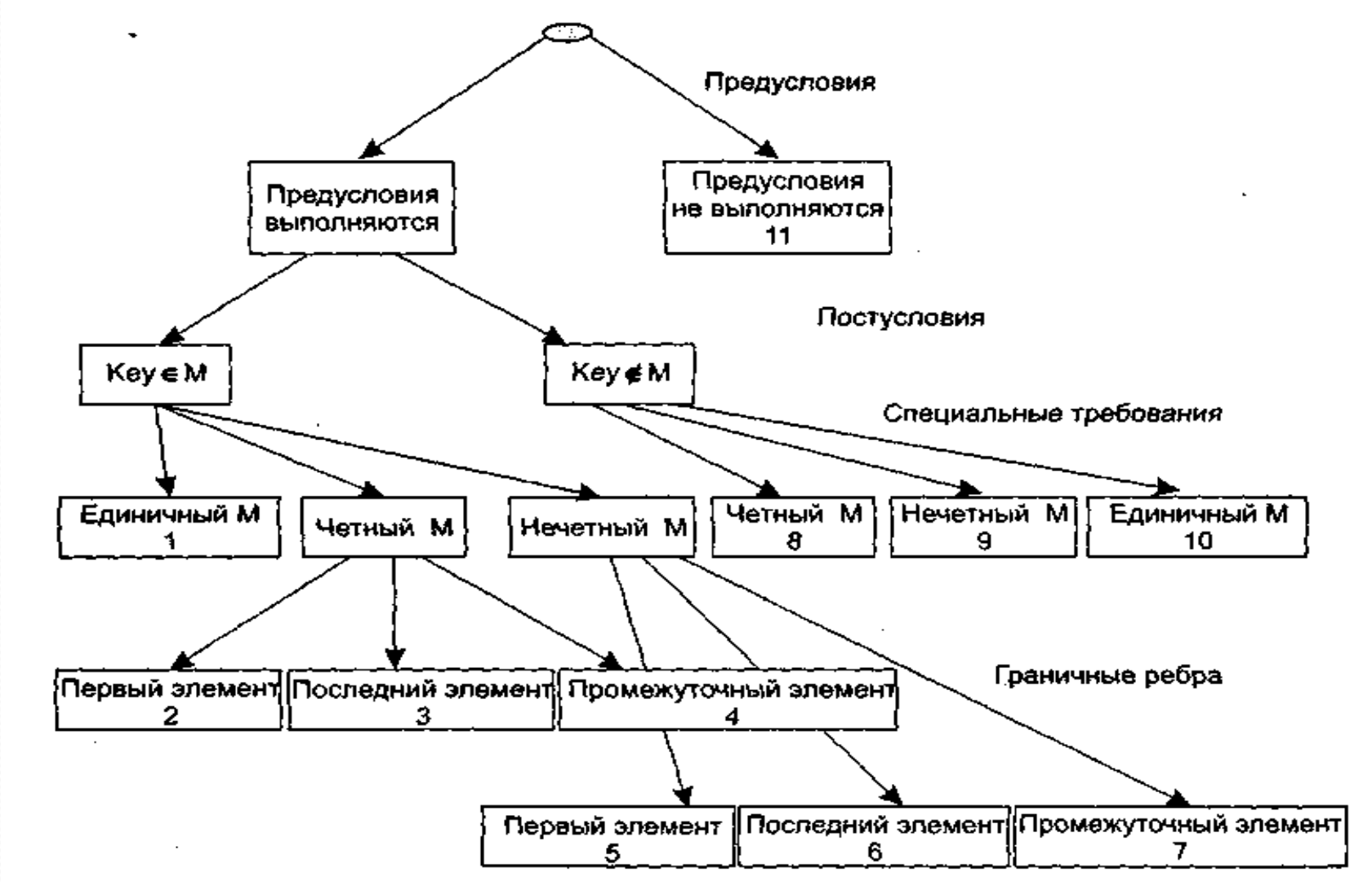
Специальные требования :

1) массив из одного элемента;
2) массив из четного количества элементов;
3) массив из нечетного количества элементов, большего единицы.

Анализ ребер классов эквивалентности:

1) работа с первым элементом массива;
2) работа с последним элементом массива;
3) работа с промежуточным (ни с первым, ни с последним) элементом массива.

Структура дерева разбиений



Тестовые варианты

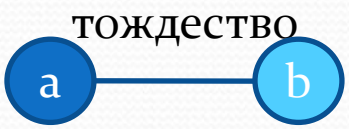
Номер теста	Исходные данные	Ожидаемый результат
1	M=15; Key=15	Result=True; I=1
2	M=15, 20, 25,30,35,40; Key=15	Result=True; I=1
3	M=15, 20, 25, 30, 35, 40; Key=40	Result=True; I=6
4	M=15,20,25,30,35,40; Key=25	Result-True; I=3
5	M=15, 20, 25, 30, 35,40, 45; Key=15	Result=True; I=1
6	M=15, 20, 25, 30,35, 40,45; Key=45	Result=True; I=7
7	M=15, 20, 25, 30,35, 40, 45; Key=30	Result=True; I=4
8	M=15, 20, 25, 30, 35,40; Key=23	Result=False; I=?
9	M=15, 20, 25, 30, 35, 40, 45; Key=24	Result=False; I=?
10	M=15; Key=0	Result=False; I=?
11	M=15, 10, 5, 25, 20, 40, 35; Key=35	Аварийное завершение: Массив не упорядочен

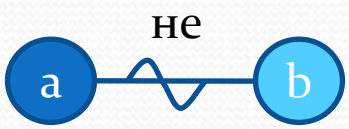
Применение функциональных диаграмм

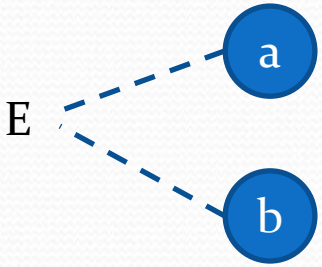
Шаги способа:

- для каждого модуля перечисляются причины и следствия;
- каждой причине и следствию присваивается свой идентификатор;
- разрабатывается граф причинно-следственных связей;
- граф преобразуется в таблицу решений;
- столбцы таблицы решений преобразуются в тестовые варианты

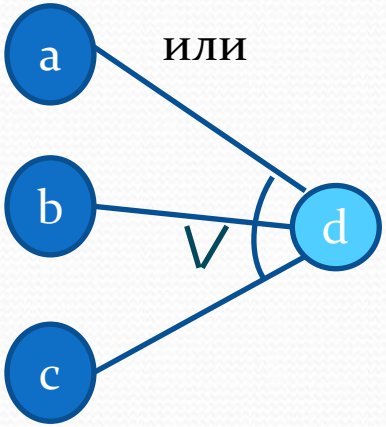
Базовые символы для записи функциональных диаграмм

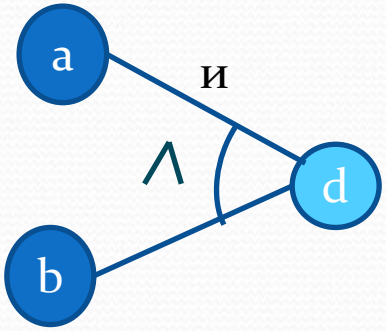


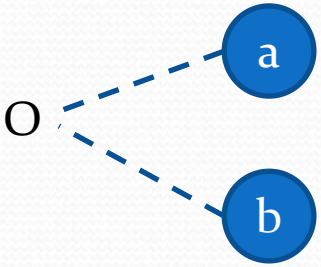




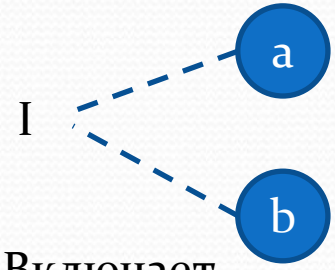
Исключает





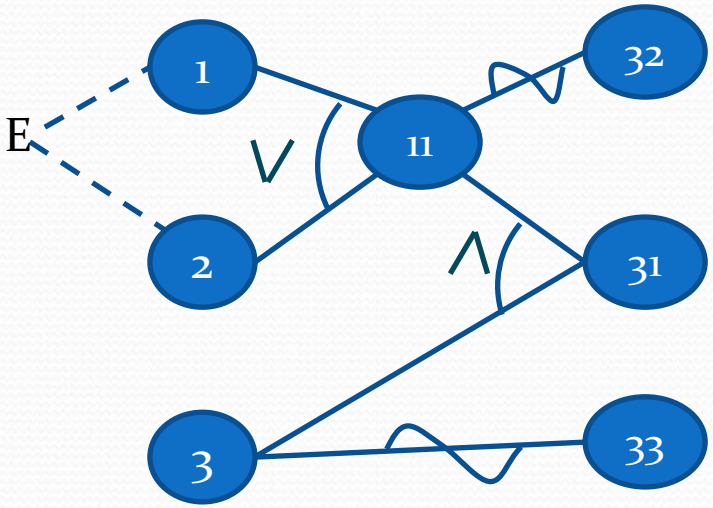


Только одно



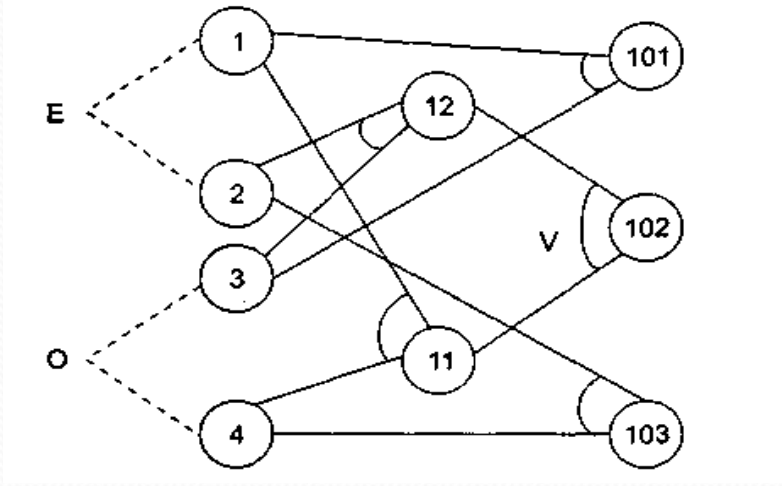
Включает

Функция f принимает два параметра. Первый может принимать значение в диапазоне 0÷10 или 20÷100. Второй параметр должен быть типа char. В этом случае возвращается код успешного завершения функции f - O'k. Если первый параметр неправильный, то выдается сообщений ErrMes1. Если второй параметр неправильный, то выдается сообщений ErrMes2.



Причины \Тесты	1	2	3	4	5	6
1	0	0	0	0	1	1
2	0	0	1	1	0	0
3	0	1	0	1	0	1
11	0	0	1	1	1	1
...						
31	0	0	0	1	0	1
32	1	1	0	0	0	0
33	1	0	1	0	1	0

Пример 2



Номера столбцов — >			1	2	3	4
Условия	Причины	1	1	0	1	0
		2	0	1	0	1
		3	1	1	0	0
		4	0	0	1	1
	Вторичные причины	11	0	0	1	0
		12	0	1	0	0
Действия	Следствия	101	1	0	0	0
		102	0	1	1	0
		103	0	0	0	1

N пп	**Исходные данные**	**Ожидаемый результат**
1	расчет по среднему тарифу; месячное потребление электроэнергии 75 кВт/ч.	минимальная месячная стоимость
2	расчет по переменному тарифу; месячное потребление электроэнергии 90 кВт/ч	процедура *A* планирования расчета
3	расчет по среднему тарифу; месячное потребление электроэнергии 100 кВт/ч	процедура *A* планирования расчета
4	расчет по переменному тарифу; месячное потребление электроэнергии 100 кВт/ч	процедура *B* планирования расчета

Структурное тестирование

- базируется на анализе логики ПО (ПО рассматривается как "белый ящик")
- тестированием по «маршрутам»

Структурный подход к тестированию имеет ряд недостатков

- не обнаруживают пропущенных маршрутов;
- не обнаруживают ошибок, зависящих от обрабатываемых данных, например, в операторе if (a - b) < eps - пропуск функции абсолютного значения abs проявится только, если a < b;
- не дают гарантии, что программа правильна, например, если вместо сортировки по убыванию реализована сортировка по возрастанию.

Управляющий граф программы

УГП **-** отображает поток управления программы. Это граф G(V, A), где V(V1,… Vm) – множеств вершин (операторов), A(A1,… An) – множество дуг (управлений), соединяющих вершины.

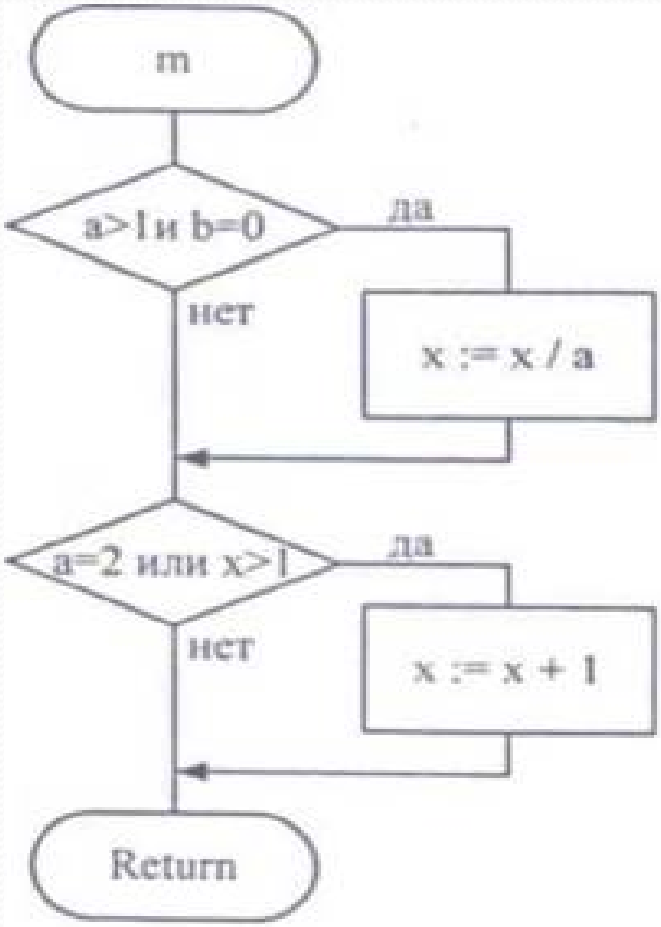
Путь – последовательность вершин и дуг УГП, в которой любая дуга выходит из вершины Vi и приходит в вершину Vj

Ветвь – *путь*(V1, V2, … Vk), где V1 - либо первый, либо условный оператор, Vk - либо условный оператор, либо оператор выхода из программы, а все остальные операторы – безусловные. Ветви - линейные участки программы, их конечное число.

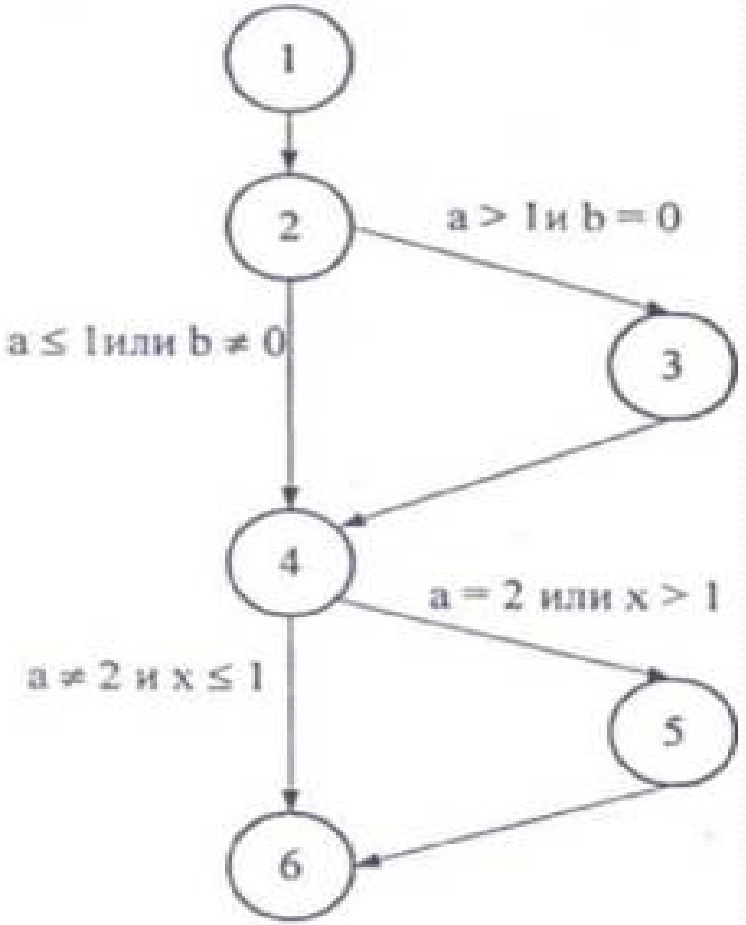
Число путей в программе может быть не ограничено (пути, различающиеся хотя бы числом прохождений цикла – разные).

Существуют реализуемые и нереализуемые пути в программе, в нереализуемые пути в обычных условиях попасть нельзя.

Преобразование схемы алгоритма в УГП



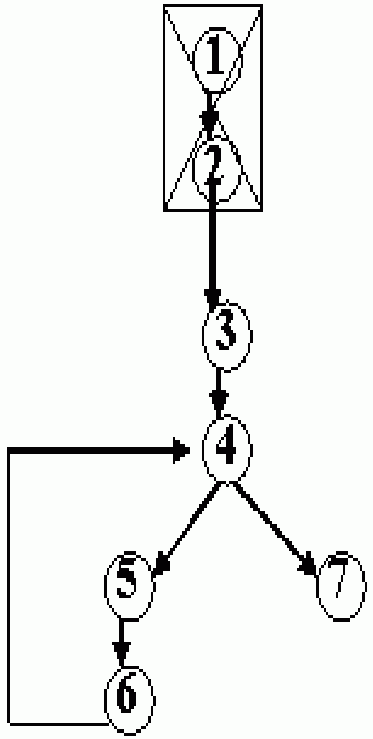
а



б

Управляющий граф г

```
/* Функция вычисляет неотрицательную
     степень n числа x */
1    double Power(double x, int n){
2    double z=1; int i;
3    for ( i = 1;
4          i <= n;
5          i++ )
6            { z = z*x; }  /* Возврат в п.4 */
7     return z;
     }
```



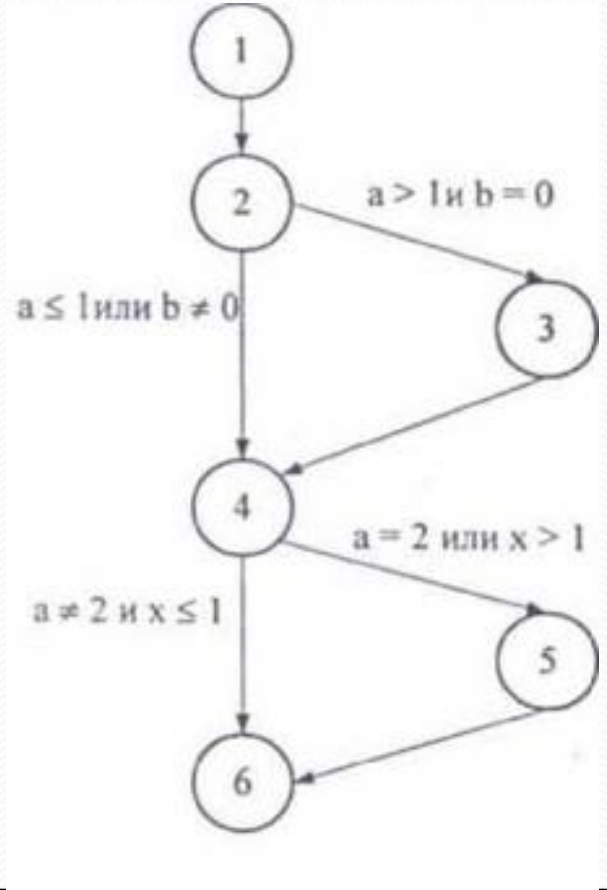
примеры путей: (3,4,7), (3,4,5,6,4,5,6), (3,4), (3,4,5,6)
примеры ветвей: (3,4) (4,5,6,4) (4,7)

Критерии структурного тестирования

- покрытие операторов;
- покрытие решений;
- покрытие условий;
- покрытие решений/условий;
- комбинаторное покрытие условий

Критерий покрытие операторов

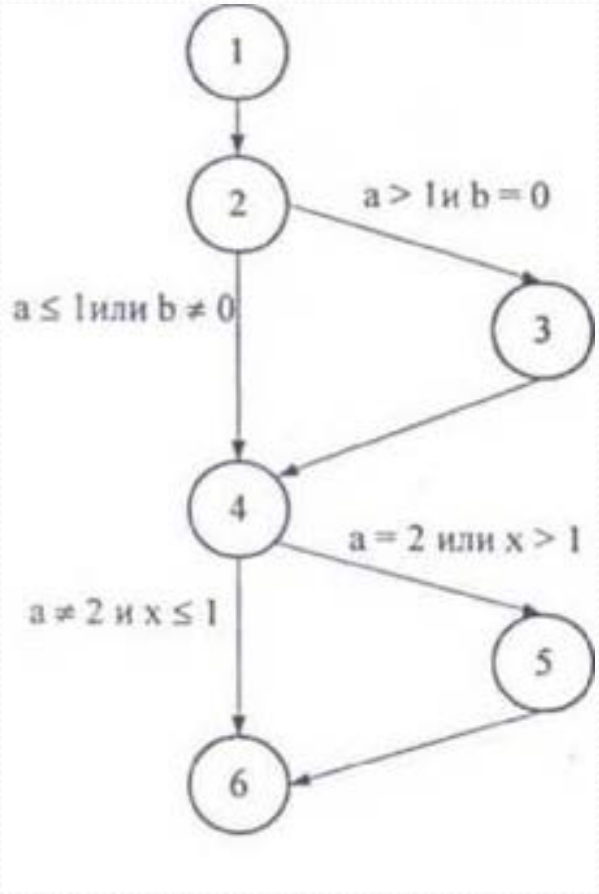
Тест	Ожидаемый результат	Фактический результат	Результат тестирования
A = 2, B = 0, X=3	*X*=2,5	*X*=2,5	Неуспешно

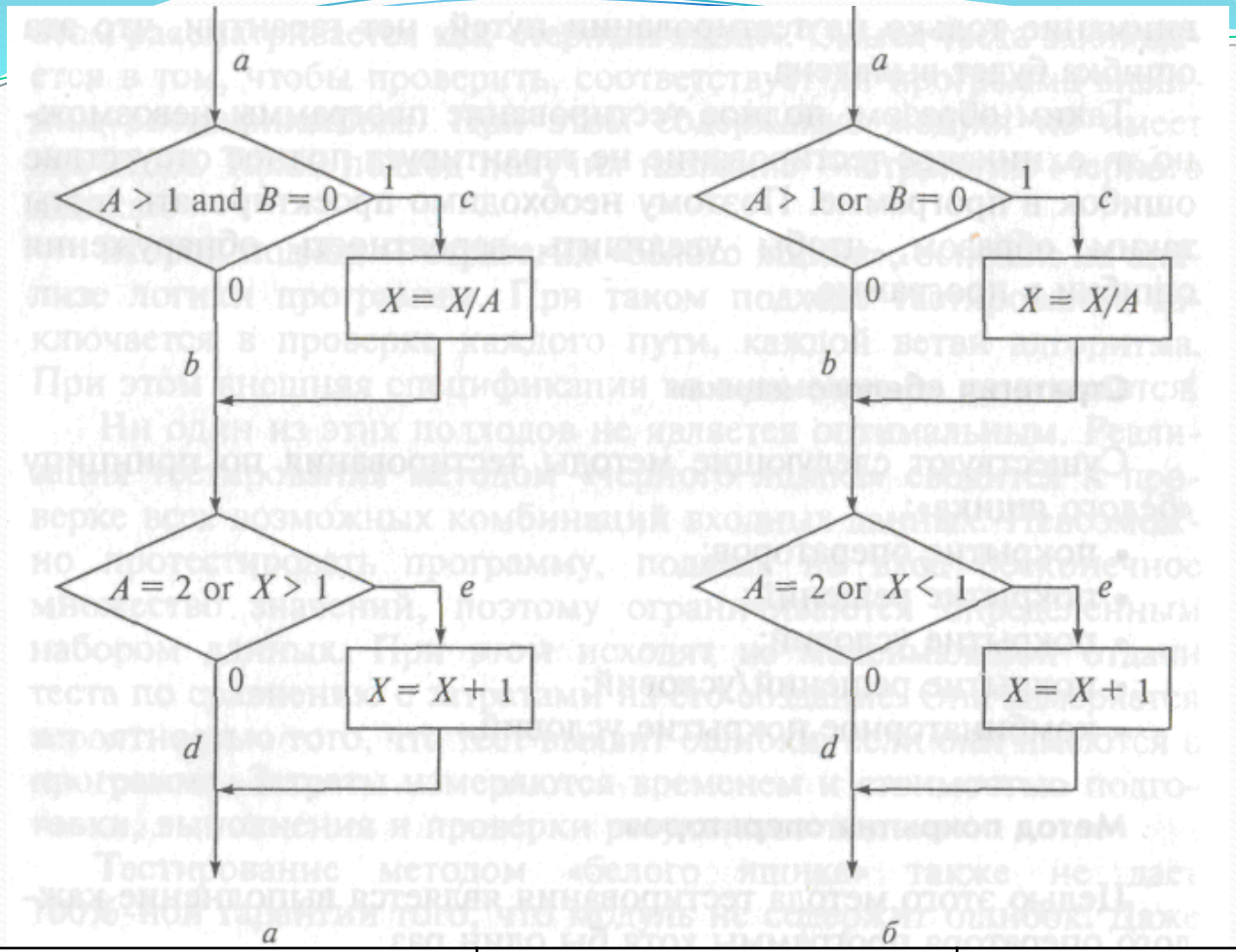


Критерий покрытия решений

- пути: 1-2-4-6, 1-2-3-4-5-6,
- пути: 1-2-3-4-6, 1-2-4-5-6.

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

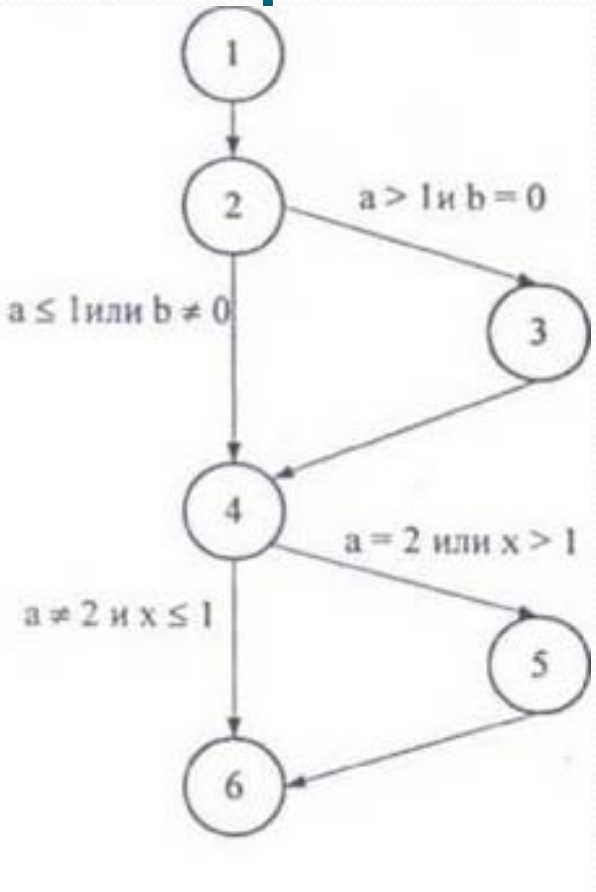




а б

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=1*	*X=1,5*	Успешно

Покрытие условий



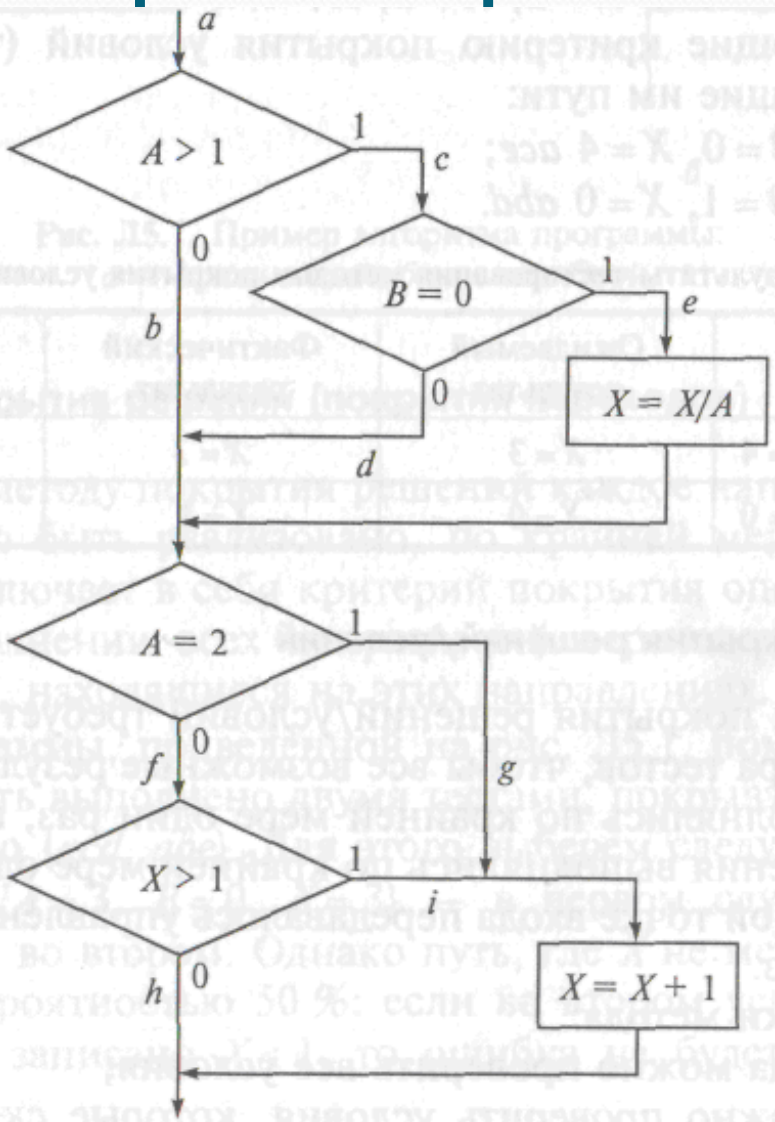
1) a>1	2) b=0	3) a=2	4) x>1
a>1, a<=1	b=0, b<>0	a=2, a<>2	x>1, x<=1

Тесты, удовлетворяющие этому условию:

a=2,b=0,x=4 – путь1-2-3-4-5-6 условия 1 – «1»
2 – «1»
3 – «1»
4 – «1»

a=1,b=1,x=0 – путь1-2-4-6 условия 1 – «0»
2 – «0»
3 – «0»
4 – «0»

Покрытие решений/условий

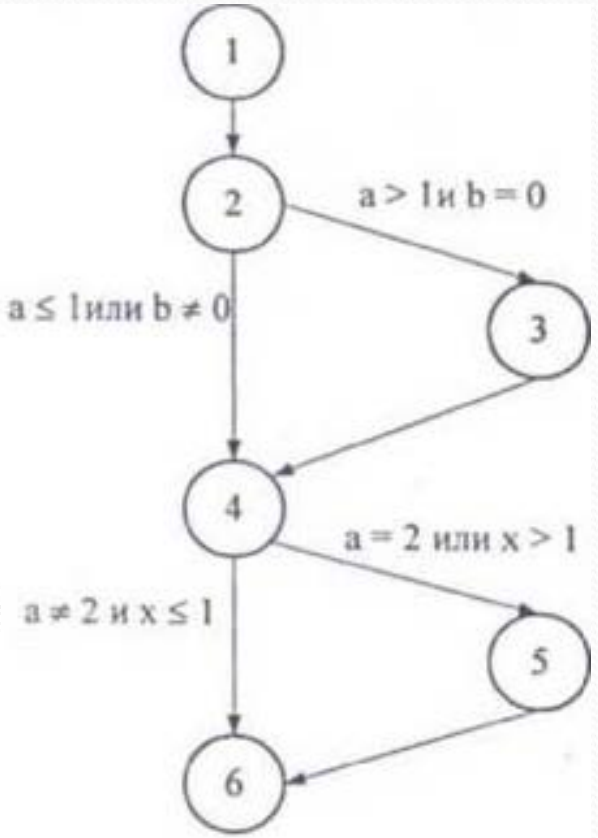


- *aceg* (тест *A* = 2, B=0, X=4),
- *acdfi* (тест A = 3, B= 1,X=0),
- *abflh* (тест A = 0, B = 0, *X*= 0),
- *abfi*(тест *A* = 0, B= 0, *X*= 2).

Комбинаторное покрытие условий

1) $a > 1,\ b = 0;$
2) $a > 1,\ b \neq 0;$
3) $a \leq 1,\ b = 0;$
4) $a \leq 1;\ b \neq 0;$
5) $a = 2,\ x > 1;$
6) $a = 2,\ x \leq 1;$
7) $a \neq 2,\ x > 1;$
8) $a \neq 2,\ x \leq 1.$

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

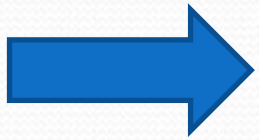


Построение набора тестов

1)конструирование управляющего графа программы;

2) выбор тестовых путей;

- *Статические методы*
  - *Эвристические методы*
  - *Адаптивные методы*
- *Динамические методы*
- *Методы реализуемых путей*



3)генерация тестов, соответствующих тестовым путям.

Каждый путь описывается конъюнктивным предикатом:

Pred(i)=C1^C2^…^Cn,

где предикат Ci - соответствует переходу из вершины – решения и задаётся в этой вершине.

Методы построения множества тестов

- **Статические методы.** Построение каждого пути посредством постепенного его удлинения за счет добавления дуг, пока не будет достигнута выходная вершина.
  Недостатки

  – не учитывается возможная нереализуемость построенных путей тестирования (непредсказуемый процент брака).
  - трудоемкость (переход от покрывающего множества путей к полной системе тестов осуществляется вручную)
  Достоинство

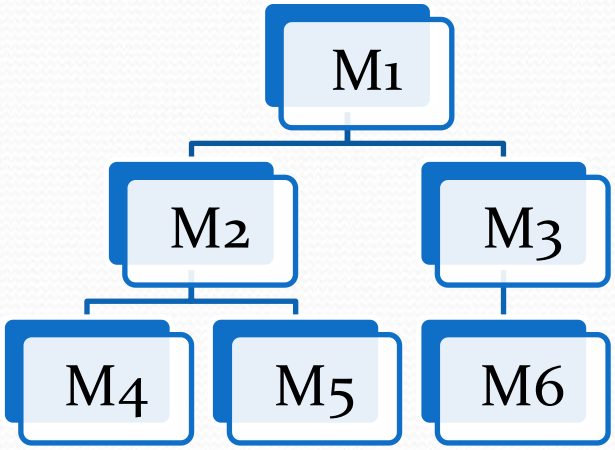
  - сравнительно небольшое количество необходимых ресурсов
- **Динамические методы.** Построение полной системы тестов, удовлетворяющих заданному критерию, путем одновременного решения задачи построения покрывающего множества путей и тестовых данных. При этом можно автоматически учитывать реализуемость или нереализуемость ранее рассмотренных путей или их частей.
  Достоинство

  - некоторый качественный уровень

  - реализуемость путей.
- **Методы реализуемых путей.** Выделение из множества путей подмножества всех реализуемых путей, из которых строится покрывающее множество путей.

Методика тестирования ПС

Тестирование многомодульного ПО



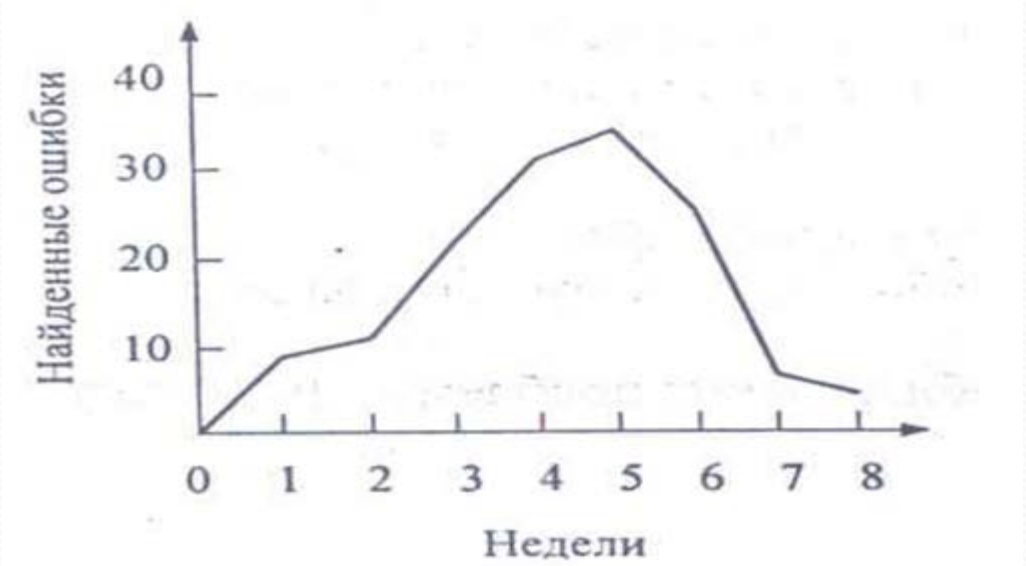
- *Монолитное* тестирование
- *Пошаговое* тестирование
  - Нисходящее
  - Восходящее
  - Комбинированное

Оценочное тестирование

- *Тестирование реальности использования*
- *Тестирование на предельных объемах*
- *Тестирование на предельных нагрузках*
- *Тестирование удобства эксплуатации*
- *Тестирование безопасности*
- *Тестирование производительности*
- *Тестирование требований к памяти*
- *Тестирование конфигурации оборудования*
- *Тестирование совместимости*
- *Тестирование удобства установки*
- *Тестирование надежности*
- *Тестирование восстановления*
- *Тестирование удобства обслуживания*
- *Тестирование пользовательской документации*
- *Тестирование процедур*

Критерии завершения тестирования и отладки.

- основанные на методологиях проектирования тестов ;
- основанные на оценке возможного количества ошибок;
- основанные на исследовании результатов тестирования.



ОТЛАДКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Методы отладки программного обеспечения

- ручного тестирования;
- индукции;
- дедукции;
- обратного прослеживания.

Индукция

Дедукция

Общая методика отладки программного обеспечения

- 1 этап - изучение проявления ошибки
- 2 этап - локализация ошибки
  - путем отсечения частей программы
  - с использованием отладочных средств
- 3 этап - определение причины ошибки
- 4 этап - исправление ошибки
- 5 этап - повторное тестирование