Лабораторна робота №4

Тема: Тестування методом «білого ящика».

Мета: Навчитися здійснювати тестування логіки програми та формалізовано описувати результати тестування.

Теоретичні відомості

Можливі різні підходи до проектування тестів. Перший полягає в тому, що тести проектуються на основі зовнішніх специфікацій програм і модулів, або специфікацій взаємозв'язку програми або модуля. Програма при цьому розглядається як чорний ящик (метод «чорного ящика»). Особлиість такого підходу - перевірити чи відповідає програма зовнішнім специфікаціям. При цьому логіка модуля зовсім не береться до уваги.

Другий підхід базується на аналізі логіки програми (метод «білого ящика»). Особливість такого підходу - в перевірці кожного шляху, блоку та кожної гілки алгоритму. При цьому зовнішня специфікація до уваги не береться.

Жоден з цих підходів не є оптимальним. З аналізу першого підходу ясно, що його реалізація зводиться до перевірки всіх можливих комбінацій значень на вході програми. Тестування будь-якої програми для всіх значень вхідних даних неможливо, так як їх безліч. При цьому виходять з максимальної віддачі тесту в порівнянні з витратами на його створення. Вона вимірюється ймовірністю того, що тест виявить помилки, якщо вони є в програмі. Витрати вимірюються часом і вартістю підготовки, виконання та перевірки результатів тесту.

Проаналізуємо тепер другий підхід до тестування. Навіть якщо припустити, що виконані тести для всіх шляхів програми, не можна з повною впевненістю стверджувати, що модуль не містить помилок.

Очевидна підстава цього твердження полягає в тому, що виконання всіх шляхів не гарантує відповідності програми її специфікаціям. Припустимо, якщо потрібно написати програму для обчислення кубічного кореня, а програма фактично обчислює корінь квадратний, то програма буде абсолютно неправильною, навіть якщо перевірити всі шляхи. Друга проблема - відсутні шляхи. Якщо програма не реалізує специфікації в повному обсязі (наприклад, відсутня така спеціалізована функція, як перевірка на від'ємне значення вхідних даних програми обчислення квадратного кореня), ніяке тестування існуючих шляхів не виявить такої помилки. І, нарешті, проблема залежності результатів тестування від вхідних даних. Шлях може правильно виконуватися для одних даних і неправильно для інших. Наприклад, якщо для визначення рівності 3 чисел програмується вираз виду:

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

Він буде вірним не для всіх значень A, B і C (помилка виникає в тому випадку, коли з двох значень B або C одне більше, а інше на стільки ж менше від A). Якщо концентрувати увагу лише на тестуванні шляхів, немає гарантії, що ця помилка буде виявлена.

Таким чином, повне тестування програми неможливо. Тест для будь-якої програми буде обов'язково неповним, тобто тестування не гарантує повну відсутність помилок в програмі. Стратегія проектування тестів полягає в тому, щоб спробувати зменшити цю неповноту наскільки це можливо.

Тестування за принципом білого ящика характеризується тим, наскільки тести виконують або покривають логіку (вихідний текст програми).

Критерій покриття операторів (C0): кожен оператор програми повинен бути виконаний (покритий) хоча б один раз.

Приклад:

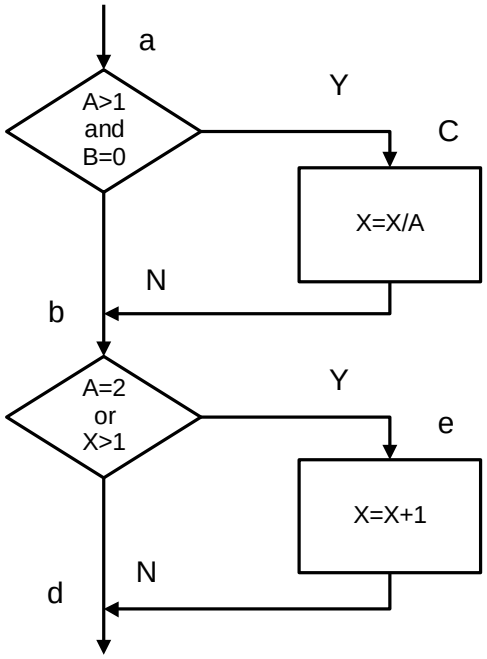

Рис.1

У цій програмі можна виконати кожен оператор, записавши один єдиний тест, який реалізував би шлях {ace}. Тобто, якби на вході було: A = 2, B = 0, X = 3, кожен оператор виконався б один раз. Але цей критерій насправді є гіршим, ніж він здається на перший погляд. Нехай в першій умові замість "and" буде "or" і в другому замість "x> 1" буде "x <1" (блок-схема на рис.2).

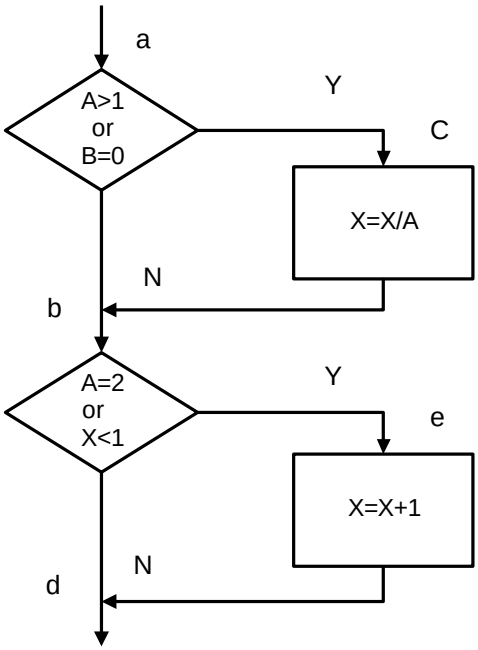

Рис.2

Результати тестування наведені в таблиці 1. Зверніть увагу: очікуваний результат визначається за алгоритмом на рис.1, а фактичний - за алгоритмом рис.2, оскільки визначається чутливість методу тестування до помилок програмування. Як видно з цієї таблиці, жодна з внесених в алгоритм помилок не буде виявлена.

Таблиця 1 - Результат тестування методом покриття операторів

Тест	Очікуваний результат	Фактичний результат	Результат тестування
A = 2, B = 0, X = 3	X = 2,5	X = 2,5	неуспішно

Критерій покриття рішень(C1): кожна гілка алгоритму (кожний перехід між вершинами) має бути пройдена (виконана) хоча б один раз. Виконання даного критерію, у загальному випадку, забезпечує й покриття операторів, проте критерій С1 не є ідеальним. Так, наприклад, він не забезпечує перевірку правильності обробки операторів логічних переходів та циклів.

Покриття рішень зазвичай відповідає критерію покриття операторів. Оскільки кожен оператор лежить на деякому шляху, що виходить або з оператора переходу, або з точки входу програми, при виконанні кожного напряму переходу кожен оператор повинен бути виконаний.

Для програми наведеної на рис.2 покриття рішень може бути виконано двома тестами, які покривають шляхи {ace, abd}, або {acd, abe}. Шляхи {acd, abe} покриєм, вибравши такі вихідні дані: {A = 3, B = 0, X = 3} і {A = 2, B = 1, X = 1} (результати тестування - в таблиці 2).

Таблиця 2 - Результат тестування методом покриття рішень

Тест	Очікуваний результат	Фактичний результат	Результат тестування
A = 3, B = 0, X = 3	X = 1	X = 1	неуспішно
A = 2, B = 1, X = 1	X = 2	X = 1,5	успішно

Критерій покриття умов. Кращі результати в порівнянні з попередніми може дати критерій покриття умов. В цьому випадку записується кількість тестів, що є достатньою для того, щоб всі можливі результати кожної умови в рішенні виконувалися принаймні один раз.

У попередньому прикладі маємо чотири умови: {A> 1, B = 0}, {A = 2, X> 1}. Відповідно, потрібна достатня кількість тестів, така, щоб реалізувати ситуації, де A> 1, A≤1, B = 0 і B≠0 в точці *а* і A = 2, A≠2, X> 1 і X≤1 в точці *в*. Тести, що задовольняють критерій покриття умов і відповідні їм шляхи:

а) A = 2, B = 0, X = 4 {ace}

б) A = 1, B = 1, X = 0 {abd}

Таблиця 3 - Результати тестування методом покриття умов

Тест	Очікуваний результат	Фактичний результат	Результат тестування
A=2, B=0, X=4	X=3	X=3	неуспішно
A=1, B=1, X=0	X=0	X=1	успішно

Критерій покриття умов/рішень. Критерій покриття умов/рішень вимагає такого достатнього набору тестів, щоб всі можливі результати кожної умови в рішенні виконувалися принаймні один раз, всі результати кожного

рішення виконувалися принаймні один раз і, крім того, кожній точці входу передавалося управління щонайменше один раз.

Два тести методу покриття умов

а) A = 2, B = 0, X = 4 {ace}

б) A = 1, B = 1, X = 0 {abd}

відповідають і критеріям покриття умов/рішень. Це є наслідком того, що одні умови наведених рішень приховують інші умови в цих рішеннях. Так, якщо умова A> 1 буде хибною, транслятор може не перевіряти умову B = 0, оскільки при будь-якому результаті умови B = 0, результат рішення ((A> 1) & (B = 0)) прийме значення брехня. Отже, недоліком критерію покриття умов/рішень є неможливість його застосування для виконання всіх результатів всіх умов.

Інша реалізація даного прикладу приведена на рис.3. Рішення з багатьма умовами вихідної програми розбиті на окремі рішення і переходи. Найбільш повне покриття тестами в цьому випадку виконується так, щоб виконувалися всі можливі результати кожного простого рішення. Для цього потрібно покрити шляхи 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) .

Протестувавши алгоритм на рис.3, неважко переконатися в тому, що критерії покриття умов і критерії покриття умов/рішень недостатньо чутливі до помилок в логічних виразах.

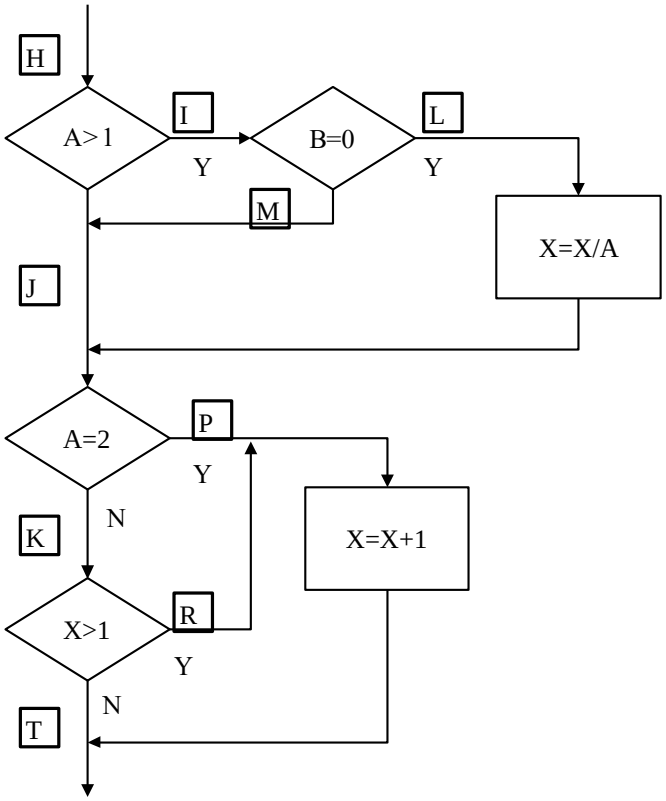

Рис.3

<u>Комбінаторний критерій покриття умов/рішень.</u> Критерієм, який вирішує ці та деякі інші проблеми, є комбінаторний критерій покриття умов. Він вимагає створення такої кількості тестів, щоб всі можливі комбінації результатів умови в кожному рішенні виконувалися принаймні один раз. Набір тестів, що задовольняє критерій комбінаторного покриття умов, задовольняє також і критерії покриття рішень, покриття умов і покриття умов/рішень.

За цим критерієм в розглянутому прикладі повинні бути покриті тестами наступні вісім комбінацій:

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 щоб протестувати ці комбінації, необов'язково використовувати всі 8 тестів. Фактично вони можуть бути покриті чотирма тестами:

A = 2, B = 0, X = 4 {покриває а, д};

A = 2, B = 1, X = 1 {покриває б, е};

A = 0,5, B = 0, X = 2 {покриває в, ж};

A = 1, B = 0, X = 1 {покриває г, з}.

Таблиця 4 - Результати тестування методом комбінаторного покриття умов

Тест	Очікуваний результат	Фактичний результат	Результат тестування
A=2, B=0, X=4	X=3	X=3	неуспішно
A=2, B=1, X=1	X=2	X=1,5	успішно
A=0,5 B=0, X=2	X=3	X=4	успішно
A=1, B=0, X=1	X=1	X=1	неуспішно

Порядок виконання роботи:

Згідно індивідуального завдання:

1. Написати програму, що реалізує заданий викладачем алгоритм обробки даних (див. індивідуальне завдання).
2. Показати алгоритм розв'язання задачі у вигляді блок-схеми програми та графу управління програми.
3. Позначити буквами або цифрами гілки алгоритму.
4. Вибрати критерій тестування, який на Вашу думку може дати найбільшу ймовірність виявлення помилок в програмі (або використати всі).
5. Виписати шляхи алгоритму, які повинні бути перевірені тестами для обраного критерію тестування.
6. Записати тести, які дозволять пройти шляхами алгоритму, обраним Вами в п.5.
7. Протестувати розроблену Вами програму. Результати оформити у вигляді таблиць (див. таблиці 1-4).
8. Перевірити чи тести виконуються при помилках в програмі (намалювати блок-схему програми з навмисно неправильно зміненими умовами та протестувати її на тестах, що розроблені вами в п.6). Результати оформити у вигляді таблиць (див. таблиці 1-4).

9. Оформити звіт по лабораторній роботі.

Індивідуальні завдання (згідно варіанту):

1. Ідентифікувати трикутник за трьома сторонами (гострокутний, прямокутний, тупокутний, рівносторонній, рівнобедрений).

2. Ідентифікувати чотирикутник по чотирьох сторонах (квадрат або ромб, прямокутник, трапеція або звичайний чотирьохкутник).

3. Ідентифікувати трикутник за двома сторонами і кутом між ними (гострокутний, прямокутний, тупокутний, рівносторонній, рівнобедрений).

4. Визначити, чи є задане з клавіатури шестизначне число парним та щасливим (сума перших трьох цифр дорівнює сумі останніх трьох цифр) або ділиться на 13.

5. Ідентифікувати трикутник за трьома кутами (гострокутний, прямокутний, тупокутний, рівносторонній, рівнобедрений).

6. Ідентифікувати трапецію по двох сторонах і куту між ними (квадрат, рівнобедрена, звичайна).

7. Визначити скільки додатніх і скільки від’ємних чисел є з трьох чисел, що введені з клавіатури.

8. З клавіатури введено номер року (додатнє ціле число). Визначити кількість днів в цьому році, враховуючи, що звичайний рік може мати або 365 або 366 днів. 366 днів мають роки, що діляться на 4, за вийнятком тих років що діляться на 100 і не діляться на 400 (наприклад роки 300,1300,1900 мають 365 днів, а 1200 і 2000 мають 366).

9. Дано три числа. Знайти суму двох найбільших з них.

10. Дано три числа. Знайти середнє з них (тобто число, що міститься між найменшим і найбільшим).

11.Дано ціле число. Якщо воно додатнє – то збільшити його на 1. Якщо від’ємне – то зменшити на 2 і якщо воно = 0 то замінити його на 20.

12. Дано координати точки, що не лежить на координатних осях. Визначити номер координатної чверті, в якій знаходиться дана точка (перша, друга, третя або четверта).