Software Quality Assurance

Đỗ Thị Bích Ngọc

PTIT/FIT/SE

ngocdtb@ptit.edu.vn

Software Testing
Các kỹ thuật kiểm thử

Các kỹ thuật kiểm thử

1. Kiểm thử hộp đen – black box testing

2. Kiểm thử hộp trắng – white box testing

1. Kiểm thử hộp đen

- Black-box testing là phương pháp kiểm thử mà không cần biết cài đặt của chương trình.
  - Cần có một bản chương trình chạy được và tài liệu đặc tả
- Một số kỹ thuật thiết kế test:

a. phân lớp tương đương (Equivalence Class Partitioning).
b. phân tích các giá trị biên (Boundary value analysis).
c. dùng các bảng quyết định (Decision Tables)
d. kiểm thử theo cặp (Pairwise)
e. dùng bảng chuyển trạng thái (State Transition)
f. …

- Quan trọng trong công nghiệp

a. Phân lớp tương đương – *Equivalence Patitioning*

- Chia miền đầu vào thành các lớp dữ liệu, từ đó suy dẫn ra các ca kiểm thử.
  - Lớp tương đương biểu thị cho tập các trạng thái **hợp lệ** hay **không hợp lệ** đối với điều kiện vào.

- Thiết kế Test-case theo 2 bước:
  - Xác định các lớp tương đương.
  - Xác định các ca kiểm thử.

a. Phân lớp tương đương *(tiếp)*

- Xác định các lớp tương đương.
  - Lấy mỗi trạng thái đầu vào (thường là một câu hay một cụm từ trong đặc tả) và phân chia nó thành 2 hay nhiều nhóm.
- Các trường hợp:
  - Điều kiện đầu vào là ***một vùng giá trị.***

Ví dụ: "*Giá trị x chỉ có thể dao động từ 0 đến 100".*

lớp tương đương hợp lệ là: **0 <=x <= 100**

và 2 lớp tương đương không hợp lệ là: **x < 0** và **x > 100**.

  - Điều kiện đầu vào là ***một số giá trị.***

Ví dụ: "Chỉ một đến sáu người có thể được đăng ký".

lớp tương đương hợp lệ: "*Có từ một đến sáu người đăng ký*"

Và 2 lớp tương đương không hợp lệ: "*không người nào đăng ký*" và "*nhiều hơn sáu người đăng ký*".

a. Phân lớp tương đương *(tiếp)*

- Xác định các lớp tương đương *(tiếp)*
  - Điều kiện đầu vào là ***một tập các giá trị:*** Ta sẽ xác định mỗi giá trị trong tập đó là một lớp tương đương hợp lệ.

Ví dụ: "Các loại xe được đăng ký là xe bus, xe khách, xe tải, xe taxi và xe máy".

5 lớp tương đương hợp lệ tương ứng với 5 loại xe

1 lớp tương đương không hợp lệ: một loại xe khác các loại xe nêu trên ví dụ như "xe đạp".

  - Điều kiện đầu vào là ***một điều kiện đặc biệt.***

Ví dụ: "Ký tự đầu tiên phải là ký tự chữ"

1 lớp tương đương hợp lệ: "*ký tự đầu tiên là ký tự chữ*"

1 lớp tương đương không hợp lệ: "*không phải là ký tự chữ (có thể là số hoặc ký tự đặc biệt)*".

a. Phân lớp tương đương *(tiếp)*

- **Xác định các ca kiểm thử**
  - Gán số thứ tự cho mỗi lớp tương đương đã xác định.
  - Viết test case cho các giá trị nằm trong các lớp tương đương hợp lệ.
  - Viết test case cho các lớp tương đương không hợp lệ.

Ví dụ

- Cho một chức năng đăng kí đăng nhập gồm 2 trường dạng text là User và Password. Trong trường Password chỉ cho nhập số ký tự trong khoảng 8 đến 30 ký tự với các mức bảo mật như trong bảng

Số ký tự	Mức độ bảo mật
Từ 8 đến 12	Yếu
Từ 13 đến 18	Trung bình
Từ 19 đến 24	Khá
Từ 25 đến 30	Tốt

- Ta sẽ chia:
  - 4 vùng hợp lệ tương đương với 4 mức độ bảo mật
  - 3 vùng không hợp lệ là số kí tự lớn hơn 30, số kí tự nhỏ hơn 8 và trường Password để trống.
- Gọi x là số kí tự ta có bảng phân vùng:

Vùng tương đương	x
Không hợp lệ	x = 0
Không hợp lệ	0<=x<=7
Hợp lệ	8 <= x <= 12
Hợp lệ	13 <= x <= 18
Hợp lệ	19 <= x <= 24
Hợp lệ	25 <= x <= 30
Không hợp lệ	x > 30

Ví dụ(tiếp): Các ca kiểm thử được sinh ra từ các vùng tương đương

STT	Mô tả	Dữ liệu kiểm thử	Đầu ra mong muốn
1	Trường Password để trống	Không nhập gì	Hiện thông báo “Mật khẩu không được để trống”
2	Số kí tự trong [Password] từ 1 đến 7	Password: Sqa	Hiện thông báo “Mật khẩu dưới 8 ký tự, mời nhập lại”
3	Số kí tự trong [Password] từ 8 đến 12	Password: Sqa12345	Hiện ra dòng chữ “Mức độ bảo mật: yếu”
4	Số kí tự trong [Password] từ 13 đến 18	Password: SQA12345789	Hiện ra dòng chữ “Mức độ bảo mật: trung bình”
5	Số kí tự trong [Password] từ 19 đến 24	Password:Sqa12345678987654321	Hiện ra dòng chữ “Mức độ bảo mật: khá”
6	Số kí tự trong [Password] từ 25 đến 30 Ấn nút đăng kí	Password: Sqa01234567899876543210sqA	Hiện ra dòng chữ “Mức độ bảo mật: tốt” ngay dưới [Password]
7	Số kí tự trong [Password] > 30 Ấn nút đăng kí	Password: Sqa0123456789987654321000123456789	Hiện thông báo “Mật khẩu trên 30 ký tự, mời nhập lại” khi ấn nút đăng kí

b. Phân tích giá trị biên – Boundary Value Analysis

- Kinh nghiệm cho thấy lỗi thường xuất hiện ở các giá trị biên.
  - Giá trị biên là các tình huống ngay tại, trên và dưới các cạnh của các lớp tương đương đầu vào và các lớp tương đương đầu ra.
- **Những cách phân tích giá trị biên**
  - Điều kiện đầu vào là **một vùng giá trị:**

test case cho giá trị hợp lệ là điểm bắt đầu, kết thúc của vùng giá trị này; test case cho giá trị không hợp lệ là giá trị ở phía ngoài của 2 điểm này.

Ví dụ: "*Giá trị x dao động từ 0 đến 100*"

Ta sẽ viết test case cho các trường hợp: **0, 100, -1, 101.**

b. Phân tích giá trị biên (tiếp)

- **Những cách phân tích giá trị biên**
  - Điều kiện đầu vào là **một số giá trị.**

test case cho giá trị hợp lệ là số nhỏ nhất, lớn nhất của các giá trị này; test case cho giá trị không hợp lệ là giá trị ở phía ngoài của 2 số này.

Ví dụ: "*Chỉ một đến sáu người có thể đăng ký*"

Ta cần viết test case cho các trường hợp: **1, 6, 0** và **7.**

  - Quan tâm đến điều kiện xuất (kết quả)

Sử dụng cách 1 và 2 ở trên áp dụng cho điều kiện xuất.

Ví dụ: "*Màn hình hiển thị tóm tắt các tin tức mới nhất và hiển thị được nhiều nhất 4 tin*".

Ta viết test case cho các kết quả hợp lệ là: **0, 1** và **4 tin**.

Test case cho kết quả không hợp lệ là **5 tin.**

b. Phân tích giá trị biên (tiếp)

- **Những cách phân tích giá trị biên**
  - Danh sách có thứ tự

    Nếu đầu vào hay đầu ra của 1 chương trình là tập được sắp thứ tự (ví dụ 1 file tuần tự hay 1 danh sách định tuyến hay 1 bảng) tập trung chú ý vào các phần tử đầu tiên và cuối cùng của tập hợp.
  - Cuối cùng, tùy vào các trường hợp khác nữa, chúng ta cũng cần sự tư duy và kinh nghiệm của mình để tìm ra các biên cần test.

c. Kỹ thuật dùng bảng quyết định (decision table)

- Miêu tả các qui tắc nghiệp vụ phức tạp mà phần mềm phải thực hiện dưới dạng dễ đọc và dễ kiểm soát
- Ví dụ 1 chức năng nhỏ của công ty bảo hiểm : khuyến mãi cho những chủ xe nếu họ thỏa ít nhất 1 trong 2 điều kiện: đã lập gia đình / là sinh viên giỏi. Mỗi dữ liệu nhập là 1 giá trị luận lý, nên bảng quyết định chỉ cần có 4 cột, miêu tả 4 luật khác nhau :

c. Kỹ thuật dùng bảng quyết định (decision table)

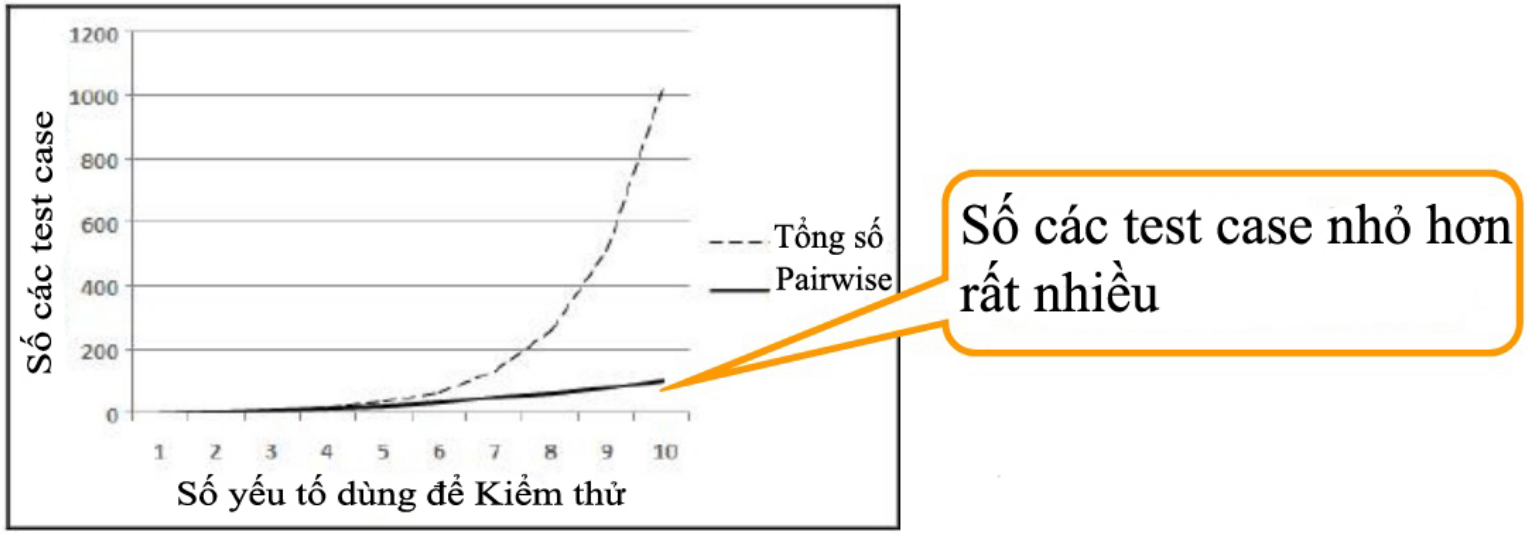
	Rule 1	Rule 2	Rule 3	Rule 4
Conditions				
Married?	Yes	Yes	No	No
Good Student?	Yes	No	Yes	No
Actions Discount ($)	60	25	50	0

Từ bảng quyết định chuyển thành bảng các testcase trong đó mỗi cột miêu tả 1 luật được chuyển thành 1 đến n cột miêu tả các testcase tương ứng với luật đó :

- nếu điều kiện nhập là trị luận lý thì mỗi cột luật được chuyển thành 1 cột testcase.
- nếu điều kiện nhập là 1 lớp tương đương (nhiều giá trị liên tục) thì mỗi cột luật được chuyển thành nhiều testcase dựa trên kỹ thuật lớp tương đương hay kỹ thuật giá trị biên.

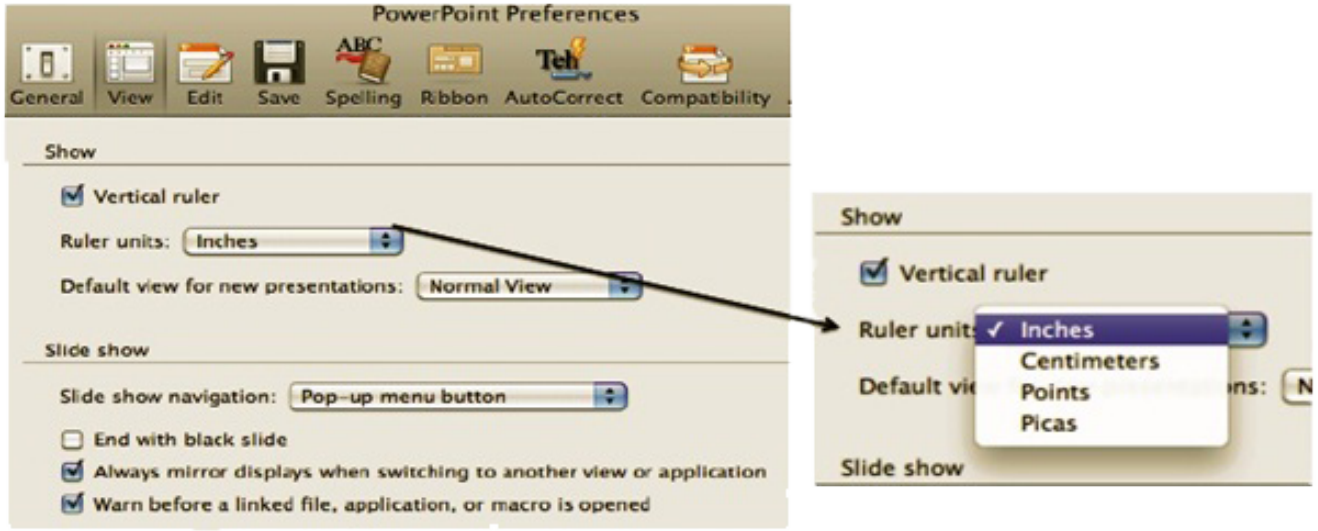
d. Kiểm thử theo cặp – pairwise testing

- Thực tế cho thấy hầu hết các lỗi đều được sinh ra từ sự kết hợp giá trị của các cặp tham số đầu vào.
- Phương pháp:
  - Lựa chọn tham số đầu vào và các giá trị tương ứng
  - Lấy tổ hợp (pairwise) của các giá trị giữa 2 tham số
  - Xây dựng bộ test sao cho bao phủ được tất cả các cặp xác định ở trên



d. Kiểm thử theo cặp – ví dụ

- Xét tab tùy chọn View từ một phiên bản của phần mềm Powerpoint Microsoft



(a) (b)

Vertical Ruler	Ruler Units	Default View	SS Navigation	End with Black	Always Mirror	Warn Before
Visible	Inches	Normal	Pop-up	Yes	Yes	Yes
Invisible	Centimeters	Slide	None	No	No	No
	Points	Outline				
	Picas					

(c)

d. Kiểm thử theo cặp – ví dụ

- Tab View_preference gồm bảy thuộc tính, mỗi thuộc tính lại bao gồm một trong các giá trị con khác nhau: Vertical_Ruler (Visible, InVisible), Ruler_units (Inches, Centimetes, Points, Picas), Default_View (Normal, Slide, Outline), Ss_Navigator (Popup, None), End_With_Black (Yes1, No1), Always_Mirror (Yes2, No2), Warn_Before (Yes3, No3).
- Ban đầu, chúng ta có tổng số 2*4*3*2*2*2*2 = 384 test case.
- Các cặp có thể có: (Visible*, Inches*), (Visible*, Centimetes*),… , (No2*, No3*)
- Test case *(Visible, Centimetes, Nomal, Non, No1, Yes2, Yes3)* bao gồm 21 cặp *(Visible, Centimetes), (Visble, Nomal),..., (Yes2, Yes3)*

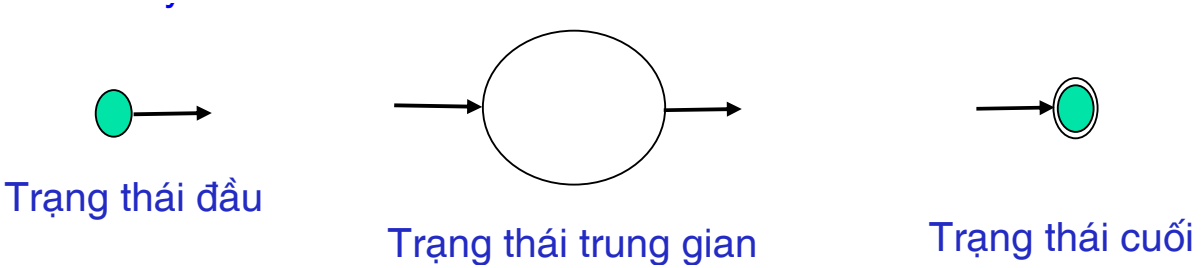
d. Kiểm thử theo cặp – ví dụ

- Một bộ test case bao phủ được tất cả các cặp được cho ở bảng dưới (được tạo bởi công cụ sinh pairwise PICT http://download.microsoft.com/download/f/5/5/f55484df-8494-48fa-8dbd-8c6f76cc014b/pict33.msi)

Vertical Ruler	Ruler units	Default View	Ss Navigator	End Black	With Always Mirror	Warn Before
Visible	Centimetes	Normal	None	No1	Yes2	Yes3
InVisible	Picas	Normal	Popup	Yes1	No2	No3
Visible	Picas	Slide	Popup	No1	No2	Yes3
InVisible	Inches	Normal	None	Yes1	Yes2	No3
InVisible	Points	Outline	None	No1	No2	No3
Visible	Centimetes	Slide	Popup	Yes1	Yes2	No3
Visible	Picas	Outline	None	Yes1	Yes2	Yes3
InVisible	Inches	Outline	Popup	No1	No2	Yes3
Visible	Points	Slide	None	Yes1	Yes2	Yes3
InVisible	Inches	Slide	None	No1	Yes2	Yes3
Visible	Inches	Normal	Popup	No1	Yes2	Yes3
Visible	Points	Normal	Popup	Yes1	Yes2	No3
InVisible	Centimetes	Outline	Popup	Yes1	No2	Yes3

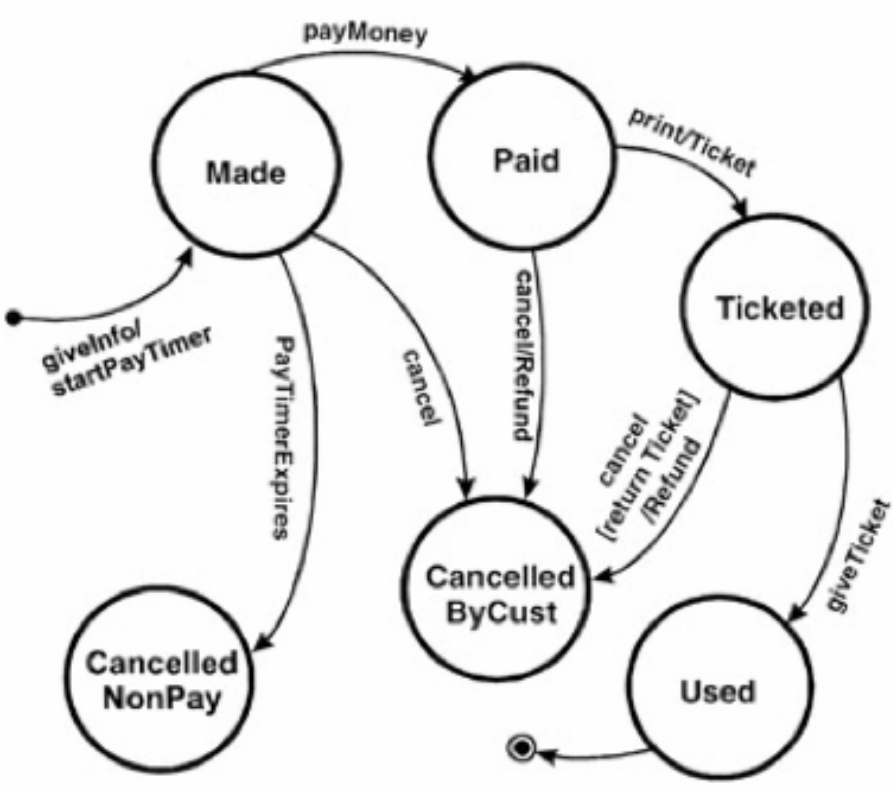
e. Kỹ thuật dùng lược đồ chuyển trạng thái

- là 1 công cụ rất hữu ích để đặc tả các yêu cầu phần mềm hoặc để đặc tả bảng thiết kế hệ thống phần mềm.
- lược đồ chuyển trạng thái ghi nhận các sự kiện xảy ra, rồi được hệ thống xử lý cũng như những đáp ứng của hệ thống.
- Khi hệ thống phải nhớ trạng thái trước đó của mình, hay phải biết trình tự các hoạt động nào là hợp lệ, trình tự nào là không hợp lệ thì lược đồ chuyển trạng thái là rất thích hợp.



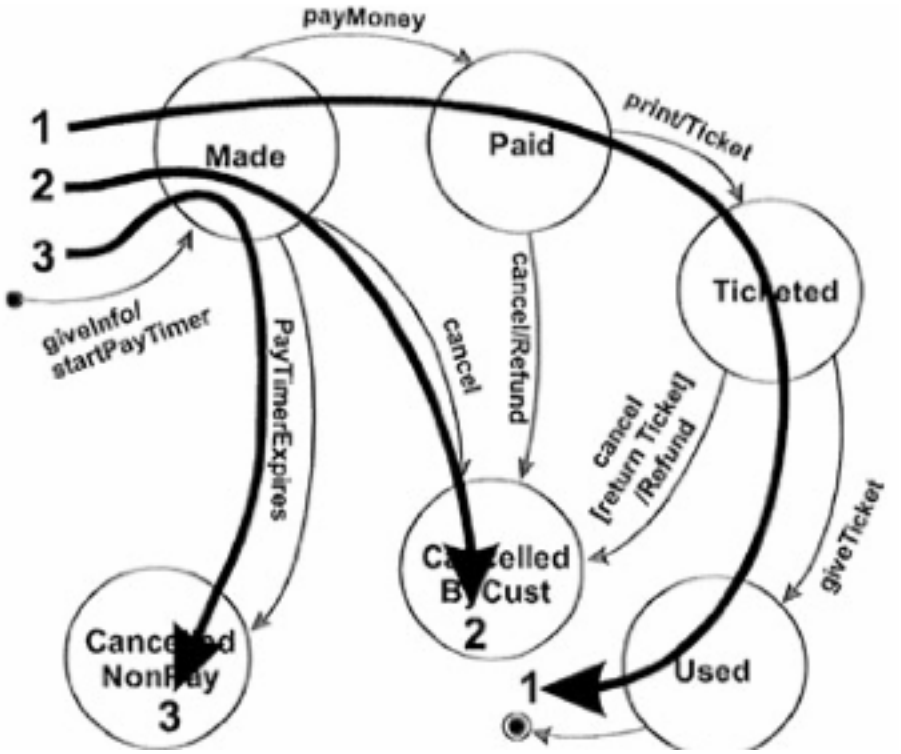
e. Ví dụ: module đặt mua vé máy bay có 6 trạng thái

Trạng thái	Điều kiện chuyển đến	Hành động cần thực hiện tiếp
made	sau khi người dùng đã nhập thông tin khách hàng.	khởi động timer T0 đếm thời gian giữ trạng thái
Cancelled (NonPay)	sau khi timer T0 đã hết.	null
3. Paid	sau khi người dùng đã thanh toán tiền.	null
Cancelled(byCust)	sau khi người dùng đã cancel	null
Ticketed	sau khi in vé xong.	null
6. Used	sau khi người dùng đã dùng	null



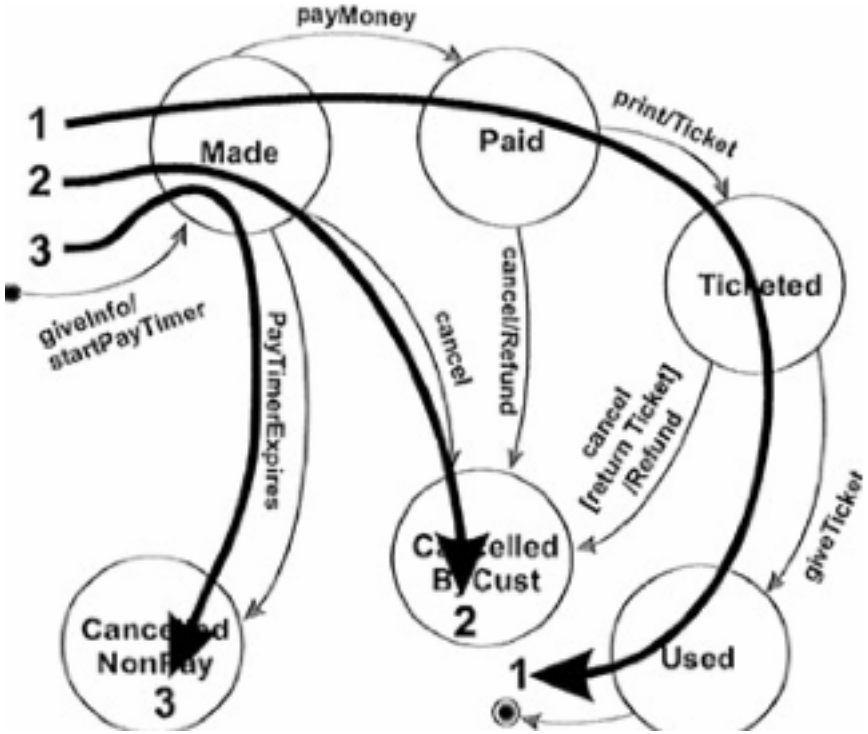
e. Ví dụ: module đặt mua vé máy bay có 6 trạng thái

- Dựa vào lược đồ chuyển trạng thái, ta có thể dễ dàng định nghĩa các testcase.
- Phủ cấp 1 : tạo các testcase sao cho mỗi trạng thái đều xảy ra ít nhất 1 lần. Thí dụ 3 tescase sau sẽ kiểm thử được TPPM đạt phủ cấp 1 :



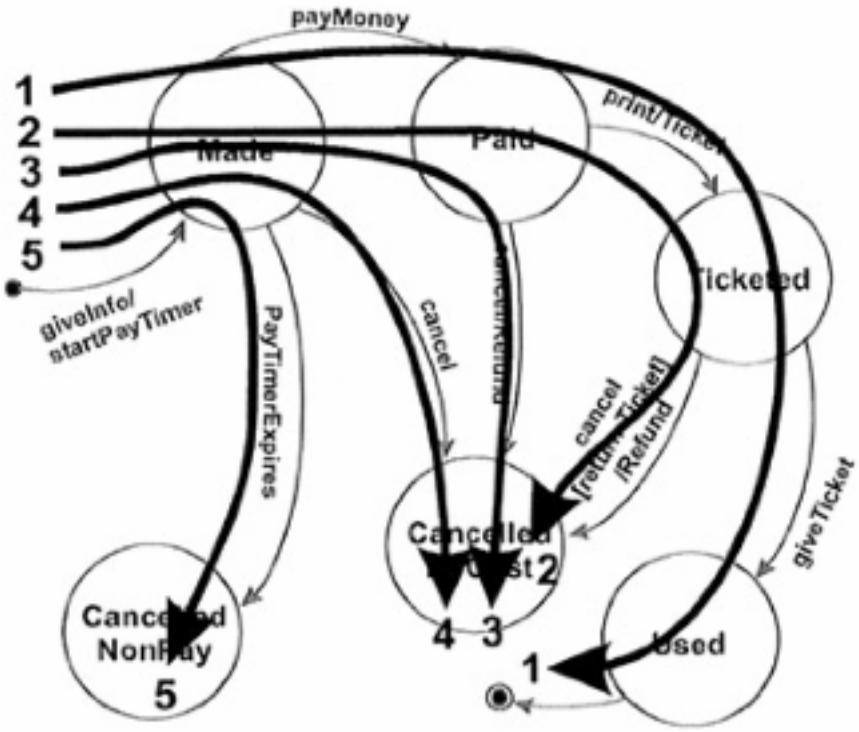
e. Ví dụ: module đặt mua vé máy bay có 6 trạng thái

- Phủ cấp 2 : tạo các testcase sao cho mỗi sự kiện đều xảy ra ít nhất 1 lần. Thí dụ 3 tescase sau sẽ kiểm thử được TPPM đạt phủ cấp 2



e. Ví dụ: module đặt mua vé máy bay có 6 trạng thái

- Phủ cấp 3 : tạo các testcase sao cho tất cả các path chuyển đều được kiểm thử. 1 path chuyển là 1 đường chuyển trạng thái xác định, bắt đầu từ trạng thái nhập và kết thúc ở trạng thái kết thúc.
  - Đây là phủ tốt nhất vì đã vét cạn mọi khả năng hoạt động của TPPM, tuy nhiên không khả thi vì 1 path chuyển có thể lặp vòng.
- Phủ cấp 4 : tạo các testcase sao cho mỗi path chuyển tuyến tính đều xảy ra ít nhất 1 lần.



2. Kiểm thử hộp trắng - White-box Testing (WBT)

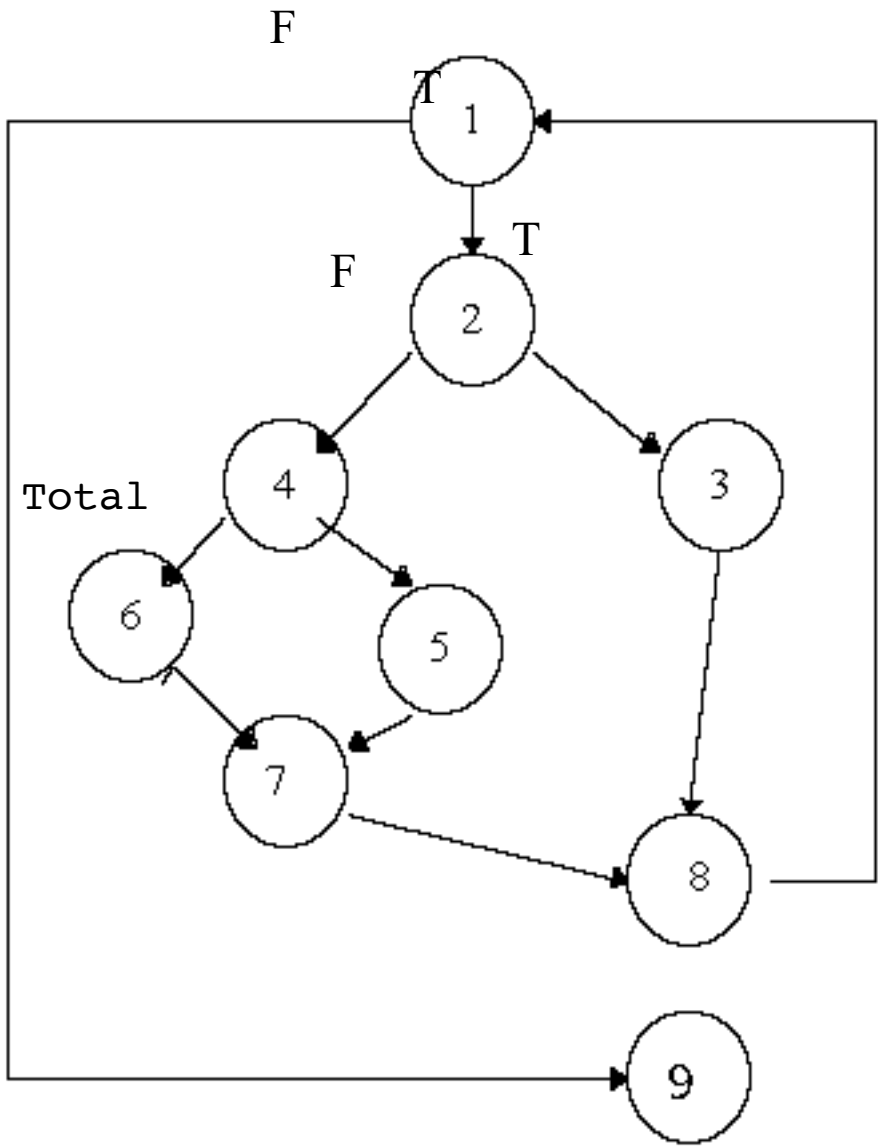
- WBT dựa vào thuật giải cụ thể, vào cấu trúc dữ liệu bên trong của module cần kiểm thử để xác định module đó có thực hiện đúng không.
- Do đó người WBT phải có kỹ năng, kiến thức để có thể thông hiểu chi tiết về đoạn code cần kiểm thử.
- Thường tốn rất nhiều thời gian và công sức
- Với các module quan trọng, thực thi việc tính toán chính của hệ thống, phương pháp này là cần thiết.
  - Cần các kỹ thuật cho kiểm thử hộp trắng
- Các phương pháp kiểm thử hộp trắng:
  - Kiểm thử luồng điều khiển
  - Kiểm thử luồng dữ liệu

a. Kiểm thử luồng điều khiển

- Đường thi hành (Execution path) : là 1 kịch bản thi hành đơn vị phần mềm tương ứng : danh sách có thứ tự các lệnh được thi hành ứng với 1 lần chạy cụ thể của đơn vị phần mềm, bắt đầu từ điểm nhập của đơn vị phần mềm đến điểm kết thúc của đơn vị phần mềm.
- Mục tiêu của phương pháp kiểm thử luồng điều khiển là đảm bảo mọi đường thi hành của đơn vị phần mềm cần kiểm thử đều chạy đúng. Rất tiếc trong thực tế, công sức và thời gian để đạt mục tiêu trên đây là rất lớn, ngay cả trên những đơn vị phần mềm nhỏ.

Ví dụ

```
1: WHILE NOT EOF LOOP
2:     Read Record;
2:    IF field1 equals 0 THEN
3:            Add field1 to Total
3:            Increment Counter
4:        ELSE
4:            IF field2 equals 0 THEN
5:                Print Total, Counter
5:                Reset Counter
6:            ELSE
6:                Subtract field2 from Total
7:            END IF
8:        END IF
8:        Print "End Record"
9:      END LOOP
9:      Print Counter
```



Ví dụ các đường thi hành:

1, 9

1, 2, 3, 8, 1, 9

1, 2, 4, 5, 7, 8, 1, 9

1, 2, 4, 6, 7, 8, 1, 9

Kiểm thử luồng điều khiển

- Thí dụ đoạn code sau :

```
for (i=1; i<=1000; i++)
   for (j=1; j<=1000; j++)
       for (k=1; k<=1000; k++)
          doSomethingWith(i,j,k);
```

có 1 đường thi hành dài 1000*1000*1000 = 1 tỉ lệnh gọi `doSomethingWith(i,j,k)` khác nhau.

- Thí dụ đoạn code gồm 32 lệnh if else sau :

```
if (c1) s11 else s12;
if (c2) s21 else s22;
if (c3) s31 else s32;

...

if (c32) s321 else s322;
```

có 2^32 = 4 tỉ đường thi hành khác nhau.

Kiểm thử luồng điều khiển

- Mà cho dù có kiểm thử hết được toàn bộ các đường thi hành thì vẫn không thể phát hiện những đường thi hành cần có nhưng không (chưa) được hiện thực :

```
if (a>0) doIsGreater();
  if (a==0) dolsEqual();
  // thiếu việc xử lý trường hợp a < 0 - if (a<0)
  dolsLess();
```

- Một đường thi hành đã kiểm tra là đúng nhưng vẫn có thể bị lỗi khi dùng thiệt (trong 1 vài trường hợp đặc biệt) :

```
int blech (int a, int b) { return a/b; }
```

khi kiểm tra, ta chọn b <> 0 thì chạy đúng, nhưng khi dùng thật trong trường hợp b = 0 thì hàm blech bị lỗi.

Phủ kiểm thử

- Do đó, ta nên kiểm thử số test case tối thiểu mà kết quả độ tin cậy tối đa. Nhưng làm sao xác định được số test case tối thiểu nào có thể đem lại kết quả có độ tin cậy tối đa ?
- Phủ kiểm thử (Coverage) : là tỉ lệ các thành phần thực sự được kiểm thử so với tổng thể sau khi đã kiểm thử các test case được chọn. Phủ càng lớn thì độ tin cậy càng cao.
- Thành phần liên quan có thể là lệnh, điểm quyết định, điều kiện con, đường thi hành hay là sự kết hợp của chúng.

Phủ cấp 0 & 1

- Phủ cấp 0 : kiểm thử những gì có thể kiểm thử được, phần còn lại để người dùng phát hiện và báo lại sau. Đây là mức độ kiểm thử không thực sự có trách nhiệm.
- Phủ cấp 1 : kiểm thử sao cho mỗi lệnh được thực thi ít nhất 1 lần.

Với hàm foo bện cạnh, ta chỉ cần 2 test case sau đây là đạt 100% phủ cấp 1 :

1. foo(0,0,0,0), trả về 0
2. foo(1,1,1,1), trả về 1 6

nhưng không phát hiện lỗi chia 0 ở hàng lệnh 8

```
1. float foo(int a, int b, int c, int
   d) {
2. float e;
3. if (a==0)
4.  return 0;
5. int x = 0;
6. if ((a==b) || ((c==d) && bug(a)))
7.  x = 1;
8. e = 1/x;
9. return e;
10.}
```

Phủ cấp 2

- Phủ cấp 2 : kiểm thử sao cho mỗi điểm quyết định đều được thực hiện ít nhất 1 lần cho trường hợp TRUE lẫn FALSE. Ta gọi mức kiểm thử này là phủ các nhánh (Branch coverage). Phủ các nhánh đảm bảo phủ các lệnh.

Line	Predicate	True	False
3	(a == 0)	Test Case 1 foo(0, 0, 0, 0) return 0	Test Case 2 foo(1, 1, 1, 1) return 1
6	((a==b) OR ((c == d) AND bug(a)))	Test Case 2 foo(1, 1, 1, 1) return 1	Test Case 3 foo(1, 2, 1, 2) return 1

Với 2 test case xác định trong slide trước, ta chỉ đạt được 3/4 x 75% phủ các nhánh. Nếu thêm test case 3 :

3. foo(1,2,1,2), thì mới đạt 100% phủ các nhánh.

Phủ cấp 3

- Phủ cấp 3 : kiểm thử sao cho mỗi điều kiện luận lý con (subcondition) của từng điểm quyết định đều được thực hiện ít nhất 1 lần cho trường hợp TRUE lẫn FALSE. Ta gọi mức kiểm thử này là phủ các điều kiện con (subcondition coverage). Phủ các điều kiện con chưa chắc đảm bảo phủ các nhánh.

Predicate	True	False
a==0	Test Case 1 foo(0, 0, 0, 0) return 0	Test Case 2 foo(1, 1, 1, 1) return value 0
a==b	Test Case 2 foo(1, 1, 1, 1) return 1	Test Case 3 foo(1, 2, 1, 2) division by zero!
c==d		Test Case 3 foo(1, 2, 1, 2) division by zero!
bug(a)		

Phủ cấp 4

- Phủ cấp 4 : kiểm thử sao cho mỗi điều kiện luận lý con (subcondition) của từng điểm quyết định đều được thực hiện ít nhất 1 lần cho trường hợp TRUE lẫn FALSE & điểm quyết định cũng được kiểm thử cho cả 2 nhánh. Ta gọi mức kiểm thử này là phủ các nhánh & điều kiện con (branch & subcondition coverage).

b Phương pháp kiểm thử các đường thi hành cơ bản - Basis Path Testing (by Tom McCabe)

Các bước:

- Từ module cần kiểm thử, xây dựng đồ thị luồng điều khiển G tương ứng.
- Tính độ phức tạp Cyclomatic của đồ thị (=C).
  - V(G) = E - N + 2, trong đó E là số cung, N là số nút của đồ thị.
  - V(G) = P + 1, với P là số nút quyết định luận lý
  - Lưu ý: thông thường, nếu V(G) >10, ta nên chia module thành các module nhỏ hơn để giảm xác suất gây lỗi
- Xác định C đường thi hành tuyến tính cơ bản cần kiểm thử.
- Tạo từng test case cho từng đường thi hành tuyến tính cơ bản.

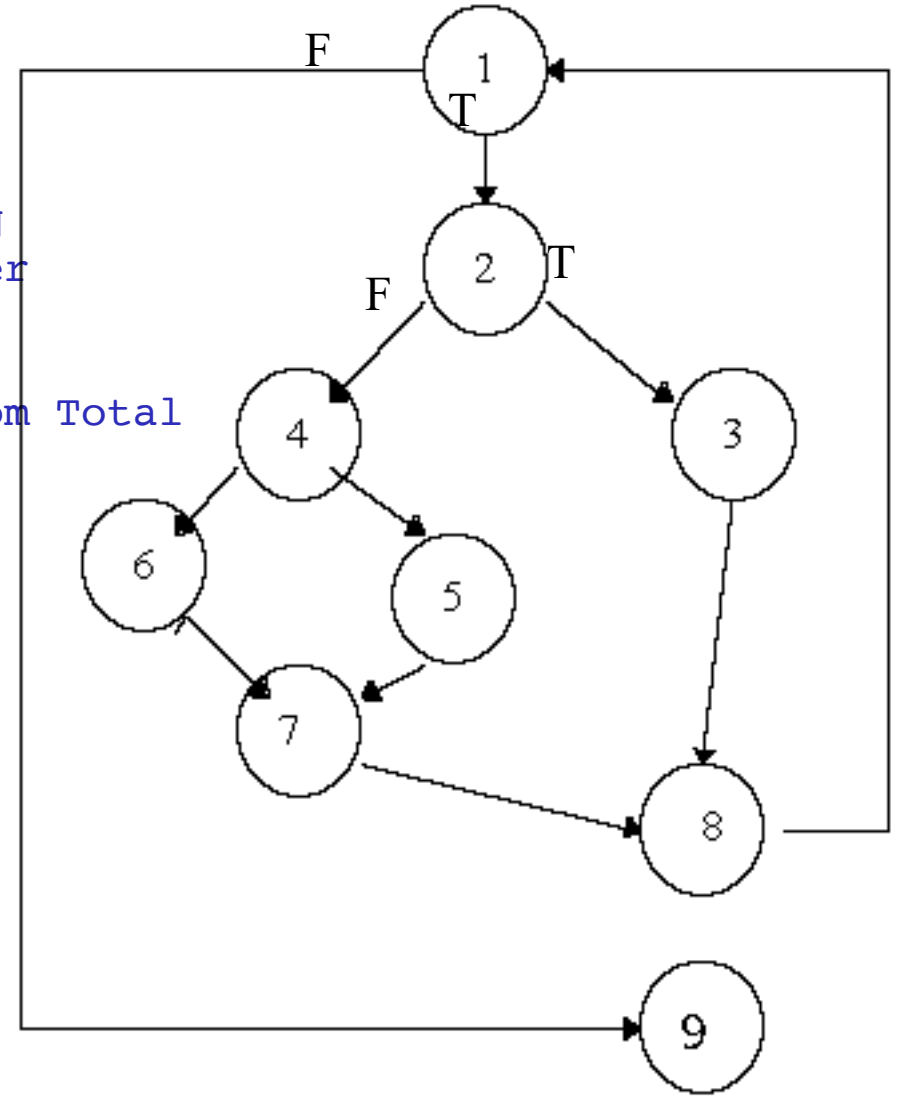
Basis Path Testing (tiếp)

Các bước xác định đường tuyến tính độc lập:

- Xác định đường cơ bản, đường này nên là đường thi hành phố biến nhất.
- Để chọn đường thứ 2, thay đổi cung xuất của nút quyết định đầu tiên và cố gắng giữ lại maximum phần còn lại.
- Để chọn đường thứ 3, dùng đường cơ bản nhưng thay đổi cung xuất của nút quyết định thứ 2 và cố gắng giữ lại maximum phần còn lại.
- Tiếp tục thay đổi cung xuất cho từng nút quyết định trên đường cơ bản để xác định đường thứ 4, 5,... cho đến khi không còn nút quyết định nào trong đường cơ bản nữa.
- Lặp dùng tuần tự các đường tìm được làm đường cơ bản để xác định các đường mới xung quanh nó y như các bước 2, 3, 4 cho đến khi không tìm được đường tuyến tính độc lập nào nữa (khi đủ số C).

Ví dụ

```
1: WHILE NOT EOF LOOP
2:      Read Record;
2:     IF field1 equals 0 THEN
3:             Add field1 to Total
3:             Increment Counter
4:         ELSE
4:             IF field2 equals 0 THEN
5:                 Print Total, Counter
5:                 Reset Counter
6:             ELSE
6:                 Subtract field2 from Total
7:             END IF
8:         END IF
8:         Print "End Record"
9:       END LOOP
9:       Print Counter
```



Ta có C = 3 + 1 = 4, đường độc lập:

1, 9

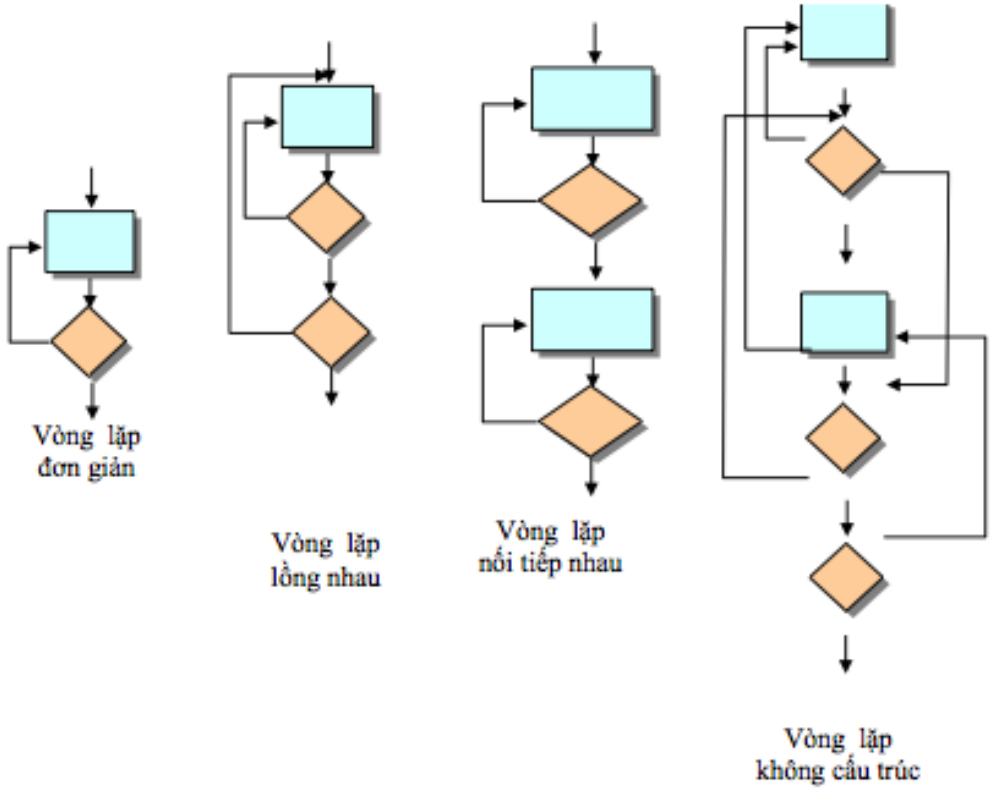
1, 2, 3, 8, 1, 9

1, 2, 4, 5, 7, 8, 1, 9

1, 2, 4, 6, 7, 8, 1, 9

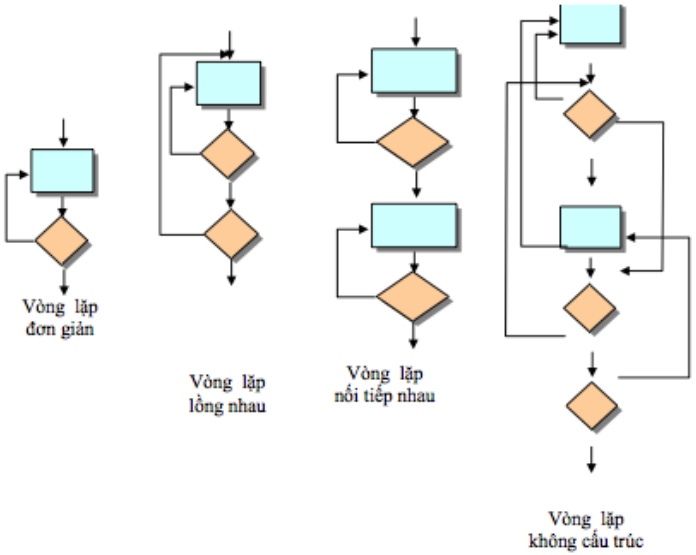
c. Kiểm thử theo vòng lặp (loop testing)

- Tập trung vào kiểm tra tính hợp lệ của cấu trúc vòng lặp.
- Với vòng lặp đơn:
  - bỏ qua vòng lặp
  - lặp 1 lần
  - lặp 2 lần
  - lặp k lần (k <n)
  - lặp n-1, n, n + 1 lần
  - Với n là số lần lặp tối đa.



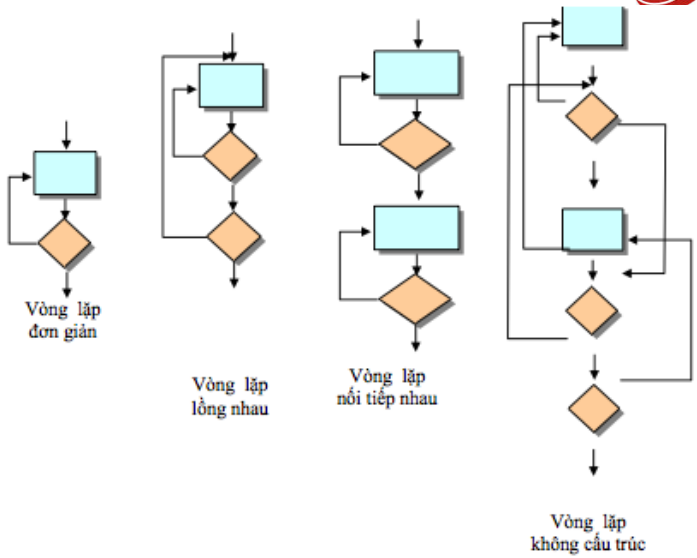
d. Kiểm thử theo vòng lặp (loop testing)

- Với vòng lặp lồng nhau:
  - khởi đầu với vòng lặp trong cùng. Thiết lập các tham số lặp cho các vòng lặp bên ngoài về giá trị nhỏ nhất
  - kiểm tra tham số min+1, một giá trị tiêu biểu, max -1, max cho vòng lặp trong cùng khi các tham số lặp của các vòng lặp bên ngoài là nhỏ nhất
  - tiếp tục với các vòng lặp bên ngoài cho tới khi tất cả các vòng lặp được kiểm tra



d. Kiểm thử theo vòng lặp (loop testing)

- Với vòng lặp lồng nhau:
  - khởi đầu với vòng lặp trong cùng. Thiết lập các tham số lặp cho các vòng lặp bên ngoài về giá trị nhỏ nhất
  - kiểm tra tham số min+1, một giá trị tiêu biểu, max -1, max cho vòng lặp trong cùng khi các tham số lặp của các vòng lặp bên ngoài là nhỏ nhất
  - tiếp tục với các vòng lặp bên ngoài cho tới khi tất cả các vòng lặp được kiểm tra



d. Kiểm thử theo vòng lặp (loop testing)

Ví dụ

```
// LOOP TESTING EXAMPLE PROGRAM import java.io.*;
class LoopTestExampleApp {
// ------------------ FIELDS ----------------------
public static BufferedReader keyboardInput = new BufferedReader(new
   InputStreamReader(System.in));
   private static final int MINIMUM = 1;
   private static final int MAXIMUM = 10;
// ------------------ METHODS ---------------------
/* Main method */
public static void main(String[] args) throws IOException {
   System.out.println("Input an integer value:");
   int input = new Integer(keyboardInput.readLine()).intValue();
   int numberOfIterations=0;
   for(int index=input;index >= MINIMUM && index <= MAXIMUM;index++)
   {
    numberOfIterations++;
   } // Output and end
   System.out.println("Number of iterations = " + numberOfIterations); }
}
```

d. Kiểm thử theo vòng lặp (loop testing)

Giá trị đầu vào	Kết quả
11	0 (bỏ qua vòng lặp)
10	1 (lặp 1 lần)
9	2 (lặp 2 lần)
5	6 (lặp k lần)
2	9 (lặp n – 1 lần)
1	10 (lặp n lần)
0	0 (bỏ qua vòng lặp)

e. kiểm thử dòng dữ liệu – data flow testing

- là 1 công cụ mạnh để phát hiện việc dùng không hợp lý các biến do lỗi coding phần mềm gây ra :

  →Phát biểu gán hay nhập dữ liệu vào biến không đúng.

  →Thiếu định nghĩa biến trước khi dùng

  →Tiên đề sai (do thi hành sai luồng thi hành).

  →...

- Mỗi biến nên có chu kỳ sống tốt thông qua trình tự 3 bước : được tạo ra, được dùng và được xóa đi.
- Chỉ có những lệnh nằm trong phạm vi truy xuất biến mới có thể truy xuất/xử lý được biến.
  - Phạm vi: toàn cục, cục bộ

kiểm thử dòng dữ liệu – data flow testing

Phân tích đời sống của 1 biến

- Các lệnh truy xuất 1 biến thông qua 1 trong 3 hành động sau :
  - →d (define): định nghĩa biến, gán giá trị xác định cho biến (nhập dữ liệu vào biến cũng là hoạt động gán trị cho biến).
  - →r(reference) : tham khảo giá trị của biến (thường thông qua biểu thức).
  - →u(undefine) : hủy (xóa bỏ) biến đi.
- Như vậy nếu ký hiệu ~ là miêu tả trạng thái mà ở đó biến chưa tồn tại, ta có 3 khả năng xử lý đầu tiên trên 1 biến :
  - →~d : biến chưa tồn tại rồi được định nghĩa với giá trị xác định.
  - →~r : biến chưa tồn tại rồi được dùng ngay (trị nào ?)
  - →~u : biến chưa tồn tại rồi bị hủy (không bình thường).

Phân tích đời sống của 1 biến (tiếp)

3 hoạt động xử lý biến khác nhau kết hợp lại tạo ra 9 cặp đôi :

→dd : biến được định nghĩa rồi định nghĩa nữa : hơi lạ, có thể đúng và chấp nhận được, nhưng cũng có thể có lỗi lập trình.

→dr : biến được định nghĩa rồi được dùng : trình tự đúng và bình thường.

→du : biến được định nghĩa rồi bị xóa bỏ : hơi lạ, có thể đúng và chấp nhận được, nhưng cũng có thể có lỗi lập trình.

→rd : biến được dùng rồi định nghĩa giá trị mới : hợp lý.

→rr : biến được dùng rồi dùng tiếp : hợp lý.

→ru : biến được dùng rồi bị hủy : hợp lý.

→ud : biến bị xóa bỏ rồi được định nghĩa lại : chấp nhận được.

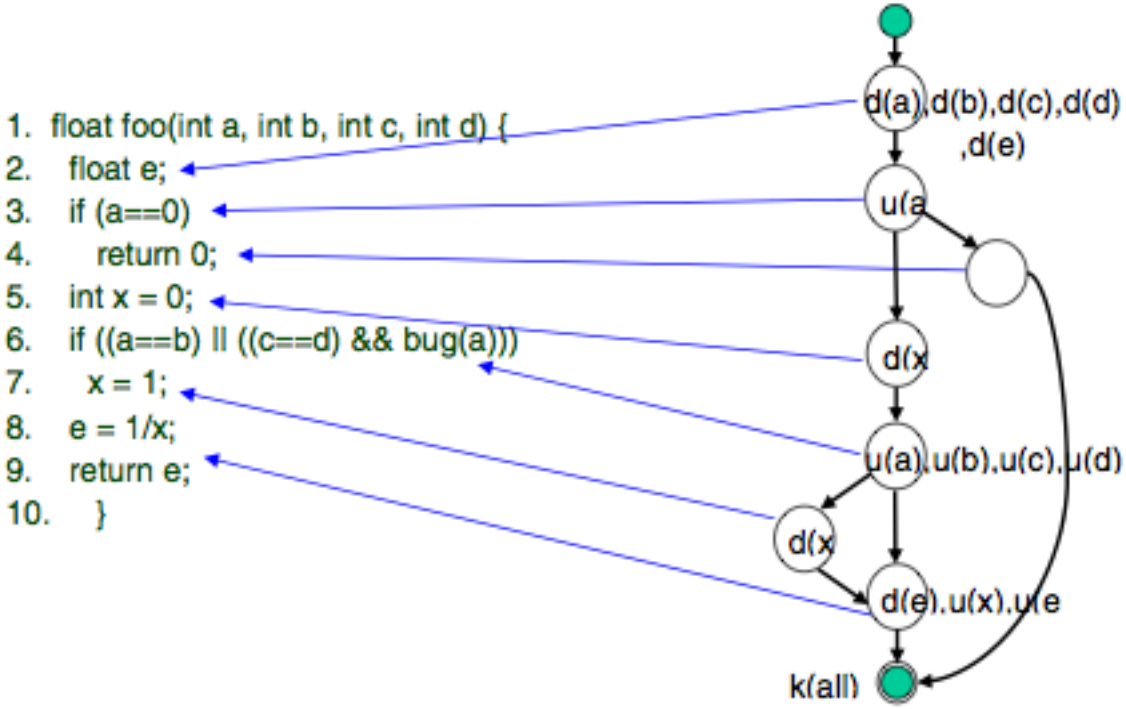
→ur : biến bị xóa bỏ rồi được dùng : lỗi.

→uu : biến bị xóa bỏ rồi bị xóa nữa : có lẽ là lỗi lập trình.

kiểm thử dòng dữ liệu – data flow testing

Đồ thị dòng dữ liệu

- Là một trong nhiều phương pháp miêu tả các kịch bản đời sống khác nhau của các biến.
- Qui trình xây dựng đồ thị dòng dữ liệu dựa trên qui trình xây dựng đồ thị dòng điều khiển của module cần kiểm thử.



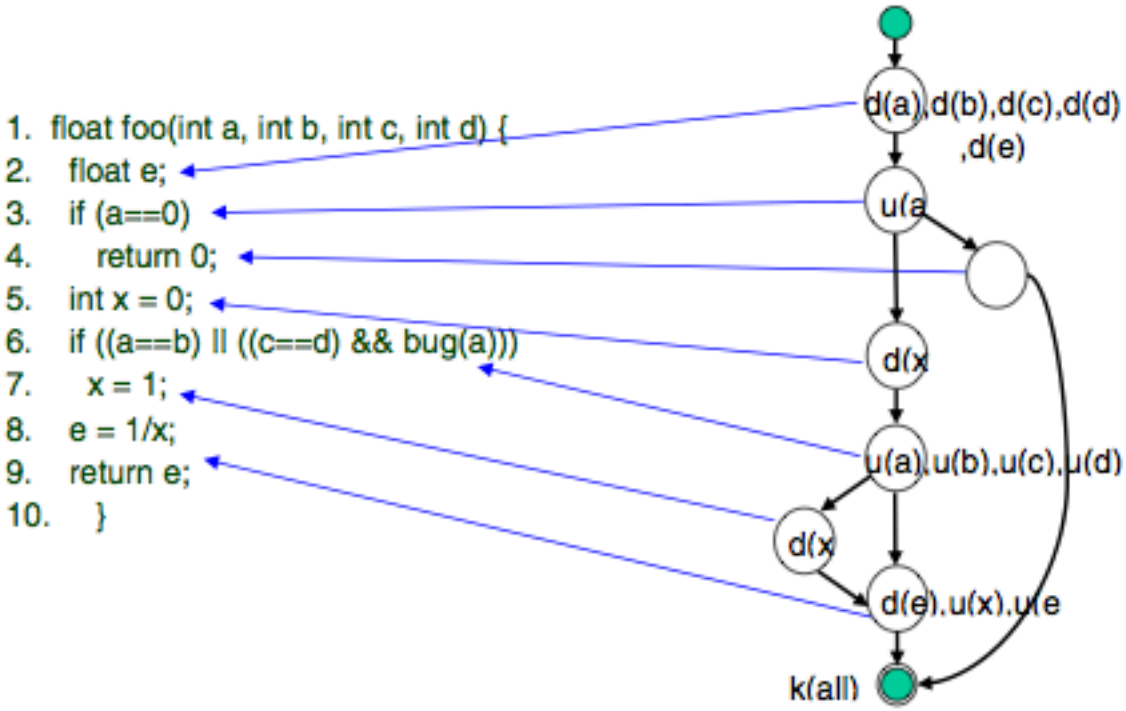
kiểm thử dòng dữ liệu – data flow testing

Qui trình kiểm thử dòng dữ liệu của 1 module gồm các bước công:

- Từ TPPM cần kiểm thử, xây dựng đồ thị dòng điều khiển tương ứng, rồi chuyển thành đồ thị dòng điều khiển nhị phân, rồi chuyển thành đồ thị dòng dữ liệu.
- Tính độ phức tạp Cyclomatic của đồ thị (C = P +1).
- Xác định C đường thi hành tuyến tính độc lập cơ bản cần kiểm thử (theo thuật giải chi tiết ở chương 3).
- Lặp kiểm thử đời sống từng biến dữ liệu :
  - mỗi biến có thể có tối đa C kịch bản đời sống khác nhau.
  - trong từng kịch bản đời sống của 1 biến, kiểm thử xem có tồn tại cặp đôi hoạt động không bình thường nào không ? Nếu có hãy ghi nhận để lập báo cáo kết quả và phản

kiểm thử dòng dữ liệu – data flow testing

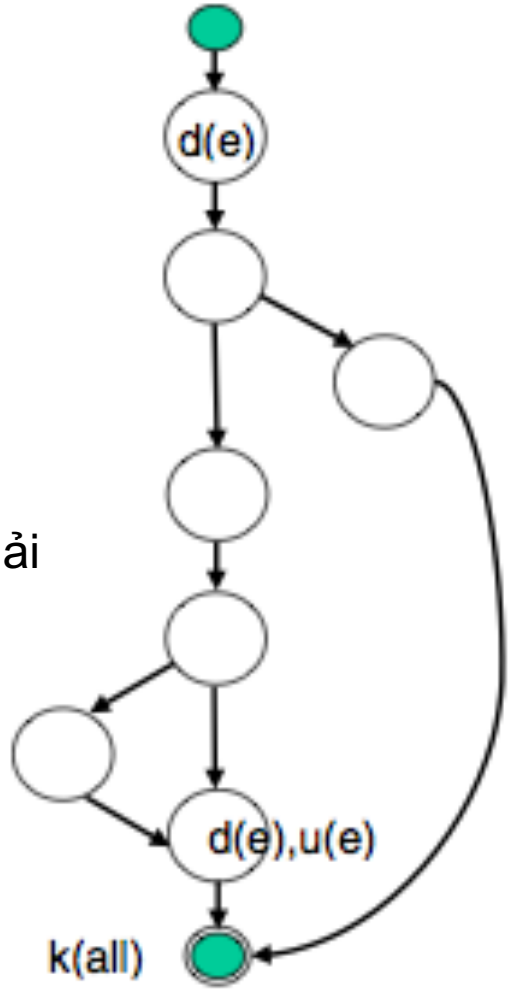
- Đồ thị ở slide trước có 2 nút quyết định nhị phân nên có độ phức tạp C = 2 +1 = 3.
- Nó có 4 biến đầu vào (tham số) và 2 biến cục bộ. Hãy lập kiểm thử đời sống từng biến a, b, c, d, e, x.



Kiểm thử đời sống biến e

- Kịch bản 1 : ~dduk
- Kịch bản 2: ~dduk (giống kịch bản 1).
- Kịch bản 3: ~dk

Trong 3 kịch bản trên, kịch bản 1 & 2 có chứa cặp đôi dd bất thường nên cần tập trung chú ý kiểm tra xem có phải là lỗi không



Tool

http://pathcrawler-online.com:8080

- PathCrawler's principal functionality, and the one demonstrated in this online version, is to automate structural unit testing by generating test inputs for full coverage of the C function under test. Full coverage can mean all feasible execution paths or *k-path* coverage which restricts the all-path criterion to paths with at most *k* consecutive loop iterations. These are available in this online version. PathCrawler can also be used to satisfy other coverage criteria (such as branch coverage, MC-DC,...), to generate supplementary tests to improve coverage of an existing functional test suite or to generate just the tests necessary to cover the part of the code which has been impacted by a modification.