IT3040- KỸ THUẬT LẬP TRÌNH
LỚP KSTN K62, NĂM HỌC 2018-2019

Giảng viên: PGS. TS.Huỳnh Quyết Thắng
BM Công nghệ phần mềm
Viện CNTT-TT, ĐHBK HN
https://users.soict.hust.edu.vn/thanghq
Điện thoại: 0913536752

Chương IV. CÁC KỸ THUẬT KIỂM THỬ VÀ GỠ RỐI CHƯƠNG TRÌNH

- 4.1. Kiểm thử chương trình
- 4.1.1 Phân loại các lỗi lập trình tiêu biểu
- 4.1.2 Vai trò của kiểm thử
- 4.1.3 Các kỹ thuật kiểm thử
- 4.1.4 Quy trình kiểm thử hiệu quả
- 4.2. Kỹ thuật gỡ rối
- 4.2.1 Tổng quan về gỡ rối chương trình
- 4.2.2 Tìm kiếm lỗi và gỡ rối lỗi
- 4.2.3 Một số kỹ thuật gỡ rối hiệu quả
- 4.2.4 Các công cụ hỗ trợ gỡ rối

Kiểm thử chương trình

- Kiểm thử phần mềm (KTPM) là gì ?
- Tại sao phải kiểm thử PM ?
- Lịch sử của Kiểm thử phần mềm
- Khái quát về các kĩ thuật kiểm thử
- Các kĩ thuật kiểm thử

Tại sao phải KTPM ?

- Đảm bảo chất lượng phần mềm sau khi đưa ra sử dụng
- Hoàn thiện & nâng cấp khả năng phần mềm
- Tránh rủi ro cho khách hàng và giảm bảo trì, bảo hành cho người viết phần mềm.

Kiểm thử phần mềm

Kiểm thử phần mềm là công việc sau cùng trước khi đưa phần mềm ra thị trường (release) , kiểm thử được thực hiện thông qua các bài Test.

- Thực tế là áp dụng các phương pháp, chiến lược cho các các tập giá trị cho đầu vào để xem kết quả đầu ra có như ý muốn hay không ?

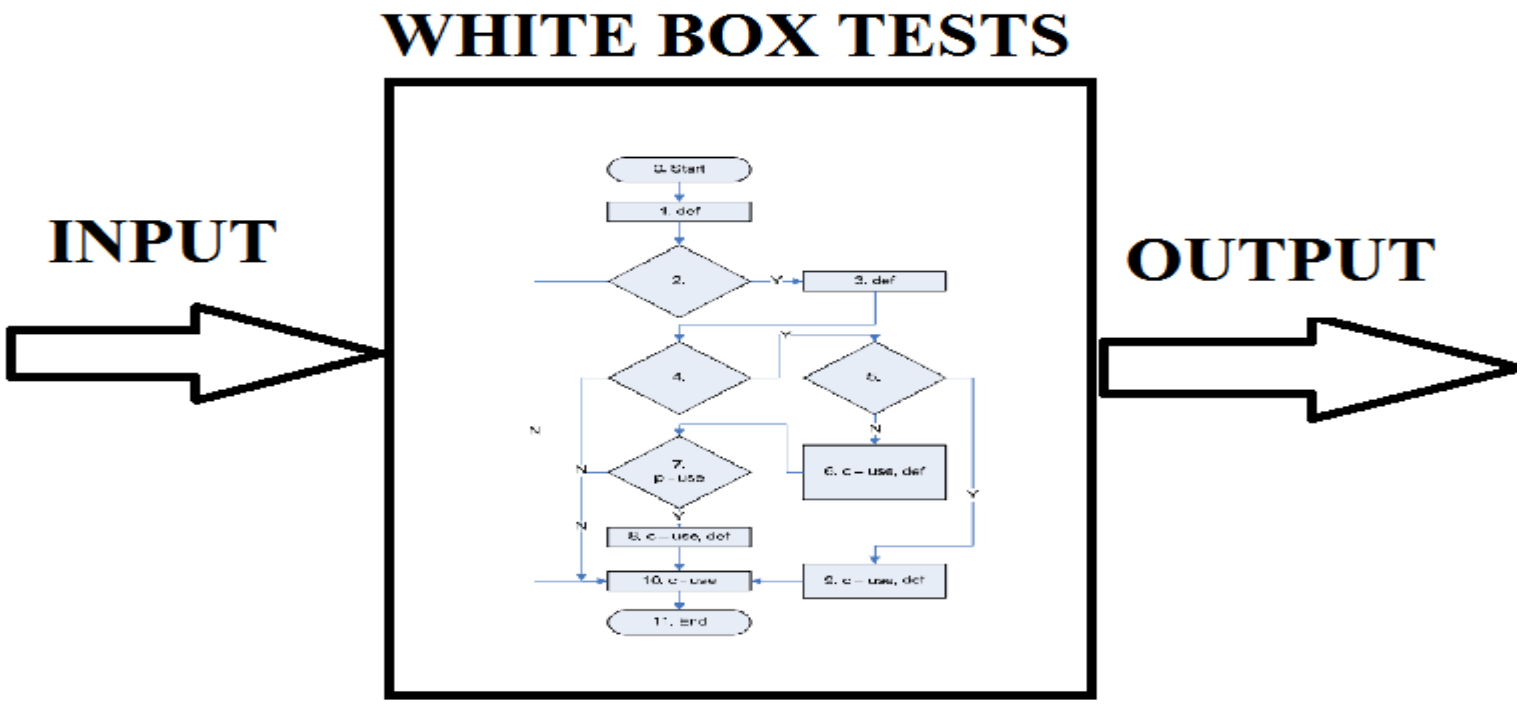
Khái quát các kĩ thuật KTPM

- **White Box Testing** (Hộp trắng)
- **Black Box Testing** (Hộp đen)
- **Grey box testing (Hộp xám)**

White Box Testing Khái niệm

- Còn được gọi là clear box testing, glass box testing, transparent box testing, or structural testing, thường thiết kế các *trường hợp kiểm thử* dựa vào *cấu trúc bên trong* của phần mềm.
- WBT đòi hỏi kĩ thuật lập trình am hiểu cấu trúc bên trong của phần mềm (các đường, luồng dữ liệu, chức năng, kết quả).
- *Phương thức :* Chọn các đầu vào và xem các đầu ra

White Box Testing Khái niệm



White Box Testing Khái niệm

- Phụ thuộc vào các cài đặt hiện tại của hệ thống và của phần mềm, nếu có sự thay đổi thì các bài test cũng cần thay đổi theo.
- Được ứng dụng trong các kiểm tra ở cấp độ mô đun(điển hình), tích hợp (có khả năng) và hệ thống của quá trình test phần mềm.

Tổng quát về kiểm thử hộp trắng

- Kiểm thử hộp trắng dựa vào thuật giải cụ thể, vào cấu trúc dữ liệu bên trong của đơn vị phần mềm cần kiểm thử để xác định đơn vị phần mềm đó có thực hiện đúng không.
- Do đó người kiểm thử hộp trắng phải có kỹ năng, kiến thức nhất định để có thể thông hiểu chi tiết về đoạn code cần kiểm thử.
- Thường tốn rất nhiều thời gian và công sức nếu mức độ kiểm thử được nâng lên ở cấp kiểm thử tích hợp hay kiểm thử hệ thống.
- Do đó kỹ thuật này chủ yếu được dùng để kiểm thử đơn vị. Trong lập trình hướng đối tượng, kiểm thử đơn vị là kiểm thử từng tác vụ của 1 class chức năng nào đó.
- Có 2 hoạt động kiểm thử hộp trắng :
  - Kiểm thử luồng điều khiển.
  - Kiểm thử dòng dữ liệu.

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.

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) doIsEqual();
// thiếu việc xử lý trường hợp a < 0 - if (a<0) doIsLess();
```

- 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 bệnh 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

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 1
(a==b)	Test Case 2 foo(1, 1, 1, 1) return value 0	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).

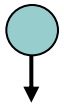
Kiểm thử các đường thi hành cơ bản

Dựa trên ý tưởng của Tom McCabe, gồm các bước công việc sau :

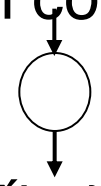
- Từ thủ tục cần kiểm thử, xây dựng đồ thị dòng điều khiển tương ứng.
- Tính độ phức tạp Cyclomatic của đồ thị (=C).
- Xác định C đường thi hành tuyến tính cơ bản cần kiểm thử.
- Tạo từng test case cho từng đường thi hành tuyến tính cơ bản.
- Thực hiện kiểm thử trên tứng test case.
- So sánh kết quả có được với kết quả được kỳ vọng.
- Lập báo cáo kết quả để phản hồi cho những người có liên quan.

Đồ thị dòng điều khiển

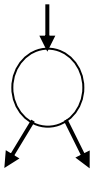
- Các thành phần cơ bản :



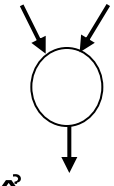
điểm xuất phát



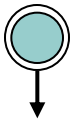
khối xử lý



điểm quyết định

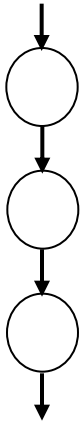


điểm nối

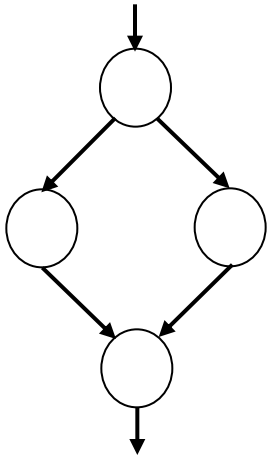


điểm kết thúc

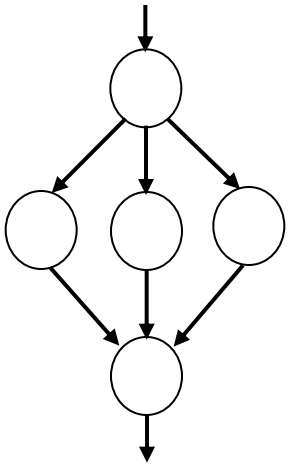
- Miêu tả các cấu trúc điều khiển phổ dụng :



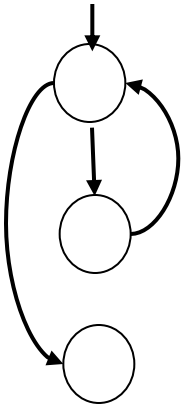
tuần tự



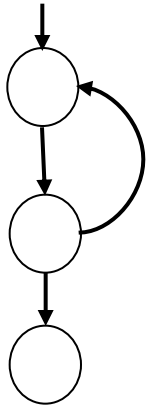
If



switch



while c do...

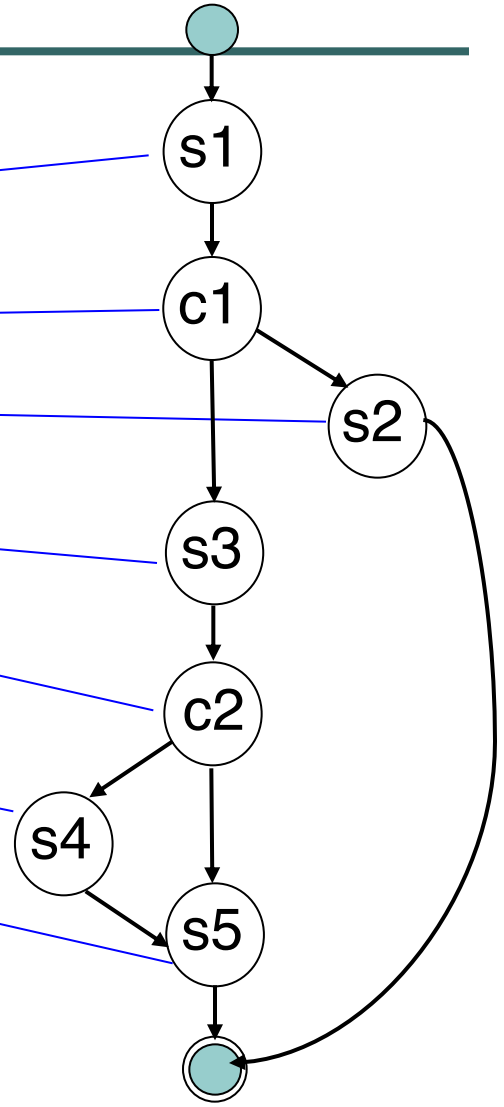


do ... while c

Đồ thị dòng điều khiển

- Thí dụ :

```
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  }
```



Độ phức tạp Cyclomatic C

Độ phức tạp Cyclomatic C = V(G) của đồ thị dòng điều khiển được tính bởi 1 trong các công thức sau :

- V(G) = E - N + 2, trong đó E là số cung, N là số nút của đồ thị.
- V(G) = P + 1, nếu đồ thị chỉ chứa các nút quyết định luận lý (chỉ có 2 cung xuất True/False) và P số nút quyết định.

Độ phức tạp Cyclomatic C chính là số đường thi hành tuyến tính độc lập cơ bản của thủ tục cần kiểm thử.

Nếu chúng ta chọn lựa được đúng C đường thi hành tuyến tính độc lập cơ bản của thủ tục cần kiểm thử và kiểm thử tất cả các đường thi hành này thì sẽ đạt được mức kiểm thử 6 như đã trình bày trong các slide trước.

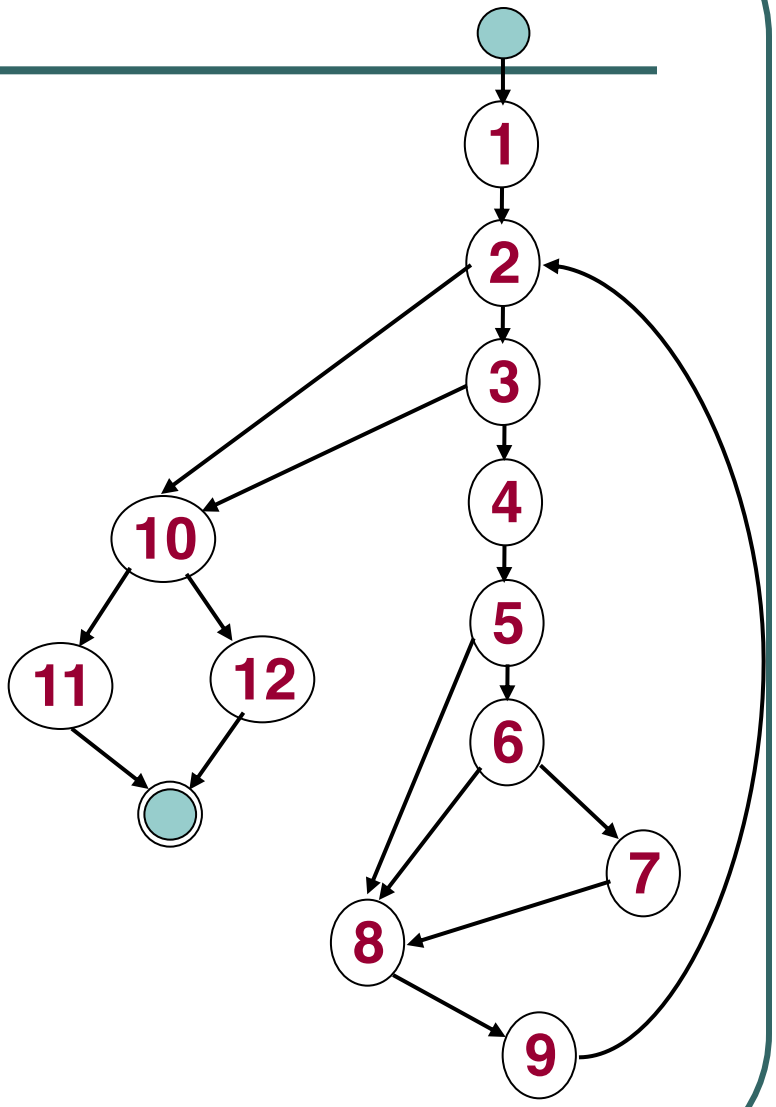
Qui trình xác định các đường tuyến tính độc lập

Tom McCabe đề nghị qui trình xác định C đường tuyến tính độc lập gồm các bước :

1. Xác định đường cơ bản, đường này nên là đường thi hành phố biến nhất.
2. Để chọn đường thứ 2, thay đổi cung xuất của nút quyết định đầu tiên và cố gắng giữ lại maximum phần còn lại.
3. 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.
4. 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.
5. 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).

Thí dụ

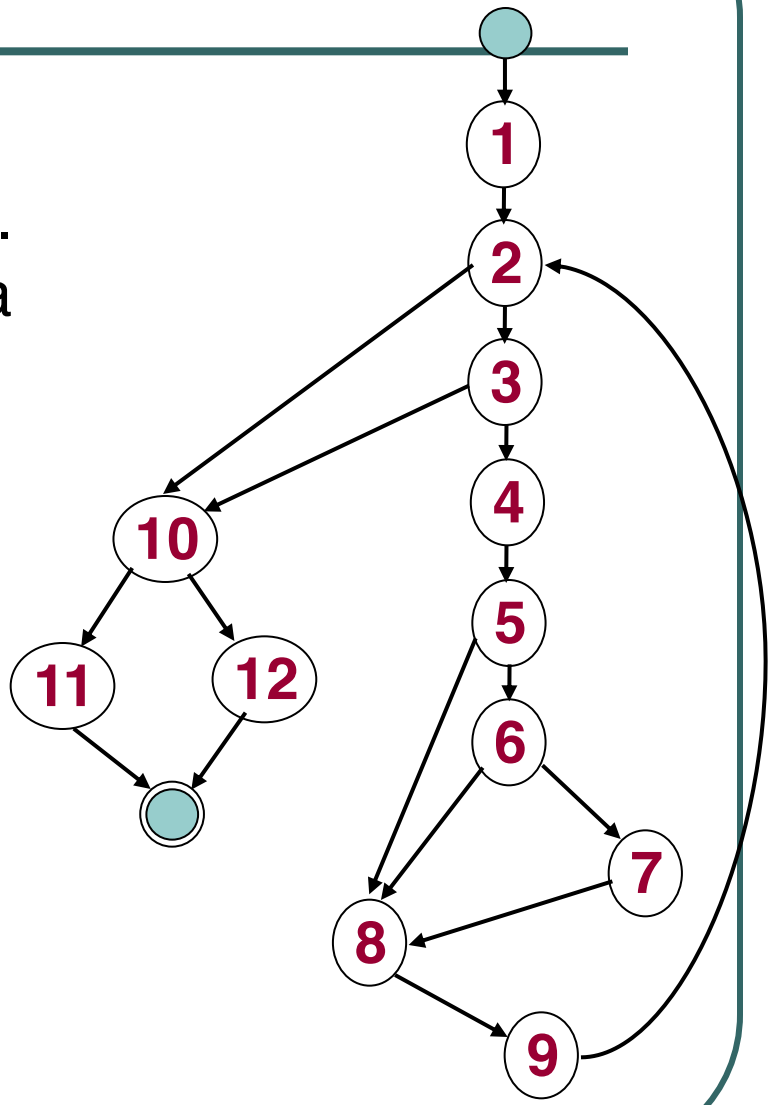
```
double average(double value[], double
    min, double max, int& tcnt, int& vcnt) {
double sum = 0;
int i = 1;
tcnt = vcnt = 0;
while (value[i] <> -999 && tcnt <100) {
  tcnt++;
  if (min<=value[i]  && value[i] <= max)
     sum += value[i];
     vcnt ++;
  }
  i++;
}
if (vcnt > 0) return sum/vcnt;
return -999;
}
```



Độ phức tạp Cyclomatic C

Đồ thị bên có 5 nút quyết định nhị phân nên có độ phức tạp C = 5+1 = 6.
6 đường thi hành tuyến tính độc lập là :

1. 1→2→10→11
2. 1→2→3→10→11
3. 1→2→10→12
4. 1→2→3→4→5→8→9
5. 1→2→3→4→5→6→8→9
6. 1→2→3→4→5→6→7→8→9



Thiết kế các test case

Test case cho đường 1 :

value(k) <>-999, với 1< k < i

value(i) = -999 với 2 ≤ i ≤ 100

Kết quả kỳ vọng : (1) average=Giá trị trung bình của k giá trị hợp lệ. (2) tcnt = k. (3) vcnt = k

Chú ý : không thể kiểm thử đường 1 này riêng biệt mà phải khiểm thử chung với đường 4 hay 5 hay 6.

Test case cho đường 2 :

value(k) <>-999, với $\forall k < i$, i >100

Kết quả kỳ vọng : (1) average=Giá trị trung bình của 100 giá trị hợp lệ. (2) tcnt = 100. (3) vcnt = 100

Test case cho đường 3 :

value(1) = -999

Kết quả kỳ vọng : (1) average = -999. (2) tcnt = 0 (3) vcnt = 0

Thiết kế các test case

Test case cho đường 4 :

value(i) <> -999 ∀i <= 100
và value(k) < min với k < i
Kết quả kỳ vọng : (1) average=Giá trị trung bình của n giá trị hợp lệ. (2) tcnt = 100. (3) vcnt = n (số lượng giá trị hợp lệ)

Test case cho đường 5 :

value(i) <>-999 với ∀i <= 100
và value(k) > max với k <= i
Kết quả kỳ vọng : (1) average=Giá trị trung bình của n giá trị hợp lệ. (2) tcnt = 100. (3) vcnt = n (số lượng giá trị hợp lệ)

Test case cho đường 6 :

value(i) <>-999 và min <= value(i) <= max với ∀i <= 100
Kết quả kỳ vọng : (1) average=Giá trị trung bình của 100 giá trị hợp lệ. (2) tcnt = 100. (3) vcnt = 100

Kiểm thử vòng lặp

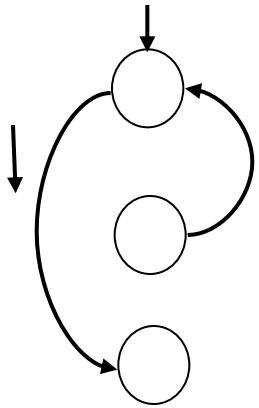
Tập trung riêng vào các lệnh lặp. Có 4 loại :

- lệnh lặp đơn giản : thân của nó chỉ chứa các lệnh khác.
- lệnh lặp lồng nhau : thân của nó chứa lệnh lặp khác...
- lệnh lặp liền kề : 2 hay nhiều lệnh lặp kế tiếp nhau
- lệnh lặp giao nhau : 2 hay nhiều lệnh lặp giao nhau.

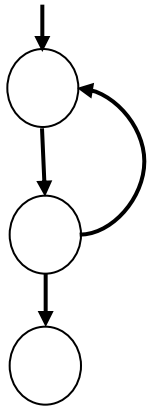
Kiểm thử vòng lặp đơn giản

Nên chọn các test case sau đây để kiểm thử lệnh lặp n lần :

- chạy 0 lần.
- chạy 1 lần.
- chạy 2 lần.
- chạy k lần, k là giá trị nào đó thỏa 2 < k < n-1.
- chạy n-1 lần
- chạy n lần
- chạy n+1 lần.



while c do...

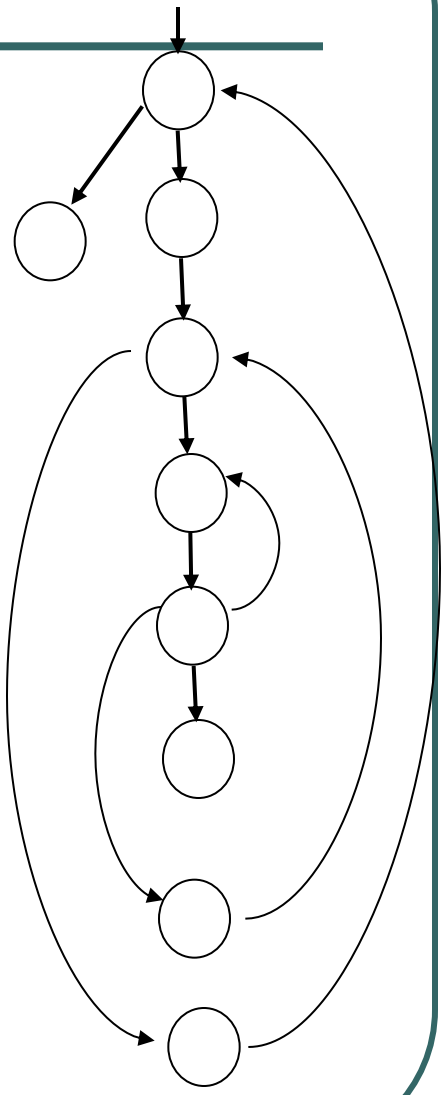


do ... while c

Kiểm thử vòng lặp lồng nhau

Kiểm thử tuần tự từng vòng lặp từ trong ra ngoài theo đề nghị sau đây :

1. kiểm thử vòng lặp trong cùng : cho các vòng ngoài chạy với giá trị min, kiểm thử vòng lặp trong cùng bằng 7 test case đã giới thiệu.
2. kiểm thử vòng lặp còn lại : cho các vòng ngoài nó chạy với giá trị min, các vòng bê trong nó chạy với giá trị điển hình, kiểm thử nó bằng 7 test case đã giới thiệu.



Kiểm thử các vòng lặp liền kề

Kiểm thử tuần tự từng vòng lặp từ trên xuống, mỗi vòng thực hiện kiểm thử bằng 7 test case đã giới thiệu.

Riêng các vòng lặp giao nhau thường do việc viết code chưa tốt tạo ra $\Rightarrow$ nên cấu trúc lại đoạn code sao cho không chứa dạng giao nhau này.

White Box Testing Trường hợp hỏng 'rác' / Failure 'Dirty' Case Test

- ***Cách kiểm thử***
- Tạo ra tất cả các trường hợp test mà người dùng thường mắc lỗi (dựa vào kinh nghiệm thực tế)
- Kiểm tra các lỗi toán học, số học, phạm vi biến, kiểu biến ....

Black Box Testing

- **Black-box testing** sử dụng mô tả bên ngoài của phần mềm để kiểm thử, bao gồm các đặc tả (specifications), yêu cầu (requirements) và thiết kế (design) .
- Không có sự hiểu biết cấu trúc bên trong của phần mềm
- Các dạng đầu vào có dạng hàm hoặc không , hợp lệ và không không hợp lệ và biết trước đầu ra.

Black Box Testing

- Được sử dụng để kiểm thử phần mềm tại mức : mô đun, tích hợp, hàm, hệ thống và chấp nhận.
- - Lợi điểm của kiểm thử hộp đen là khả năng đơn giản hoá kiểm thử tại các mức độ được đánh giá là khó kiểm thử
- - Yếu điểm là khó đánh giá còn bộ giá trị nào chưa được kiểm thử hay không

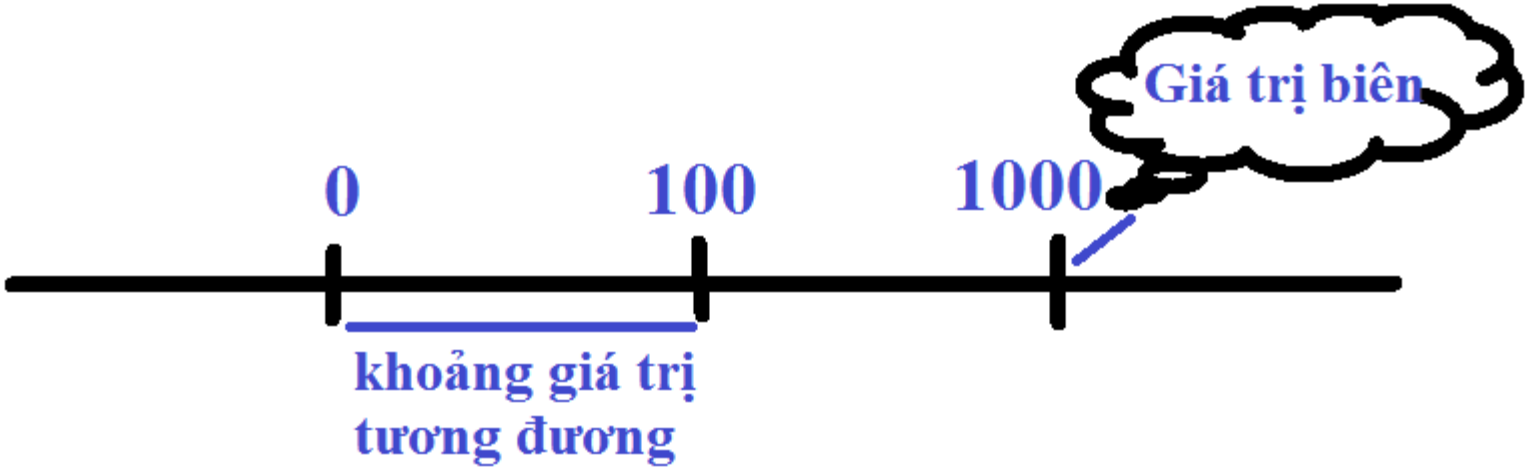
Ví dụ minh hoạ

Test ID	Description	Expected Results	Actual Results
1	Player 1 rolls dice and moves.	Player 1 moves on board.	
2	Player 2 rolls dice and moves.	Player 2 moves on board.	

Black Box Testing

- Các kí thuật chính của kiểm thử hộp đen :
- + Decision Table testing
- + Pairwise testing
- + State transition tables
- + Tests of Customer Requirement
- + Equivalence partitioning
- + Boundary value analysis
- + Failure Test Cases

White Box Testing Khoảng giá trị tương đương – Phân tích giá trị biên



- Giảm thiểu số trường hợp kiểm thử
- Phân vùng giá trị kiểm thử

Black Box Testing

- ***Decision Table Testing***
- Là cách xây dựng một bộ các giá trị kiểm thử đầy đủ không cần biết cấu trúc bên trong của phần mềm.
- Bảng quyết định được xây dựng dựa vào
- Trong đó :
- Condition : input
- Action : output

Decision Table Testing

Stub	*Rule 1*	*Rule 2*	*Rule 3*	*Rule 4*	*Rule 5*	*Rule 6*
c1	T	T	T	F	-	T
c2	F	T	T	T	-	T
c3	T	T	-	T	T	T
c4	T	F	F	T	T	T
a1	X	X		X	X	X
a2		X				
a3	X			X		
a4			X			X

Figure 2.2 a Typical Structure of a Decision Table

Black box testing Pairwise testing

- Pairwise testing là cách phối hợp các đầu vào để tạo ra bộ giá trị kiểm thử

Table 1: Pairwise test cases for System S

Test ID	Input X	Input Y	Input Z
TC_1	1	Q	5
TC_2	1	R	6
TC_3	2	Q	6
TC_4	2	R	5

- Ở ví dụ này Bộ có thể chọn của X=1 | 2
- Y= Q | R , Z= 5 | 6

Black box testing Pairwise Testing

- ***Hạn chế :*** số lượng giá trị của mỗi đầu vào tăng tạo ra sự tăng nhanh trong các trường hợp thử
- Có thể gặp phải lỗi trong việc kết hợp các giá trị đôi khi không xảy ra
- ***Ưu điểm :*** Xét được hết các trường hợp đầu vào kể cả trường hợp ngẫu nhiên của người dùng

Black box Testing

- Testing based on Customer Requirements
- Dựa vào các yêu cầu của khách hàng để tạo ra các bộ giá trị kiểm thử.

Black box Testing

- *State transition tables :* Là bảng mô tả sự chuyển trạng thái tương ứng với giá trị đầu vào tương ứng.

Table 1. Transition Table

State Name	Inputs					on entry	always
	open button	close button	door open	floor sensor	door closed		
Stopped	A	B	ignore	can't happen	C	call Stopping	none
Moving	ignore	ignore	can't happen	D	can't happen	none	none
Stopping	ignore	ignore	return	can't happen	can't happen	A	none
Going Home	ignore	ignore	can't happen	E	can't happen	none	F

Grey Box Testing

- Là sự kết hợp của kiểm thử hộp đen và kiểm thử hộp trắng khi mà người kiểm thử biết được một phần cấu trúc bên trong của phần mềm
- Như vậy không phải là KT hộp đen
- Là dạng kiểm thử tốt và có sự kết hợp các kĩ thuật của cả kiểm thử hộp đen và hộp trắng