



KIỂM THỬ PHẦN MỀM

(Software Testing)



GV: ThS. Nguyễn Thị Thanh Trúc
Khoa: Công nghệ Phần mềm
Email: tructt@uit.edu.vn

Bài 5: Các kỹ thuật kiểm thử



- **Test tĩnh (Static Verification)**
- **Test động (Dynamic Testing)**
- **5.1 Các kỹ thuật kiểm thử hộp đen**
- **5.2 Các kỹ thuật kiểm thử hộp trắng**

Các kỹ thuật kiểm thử



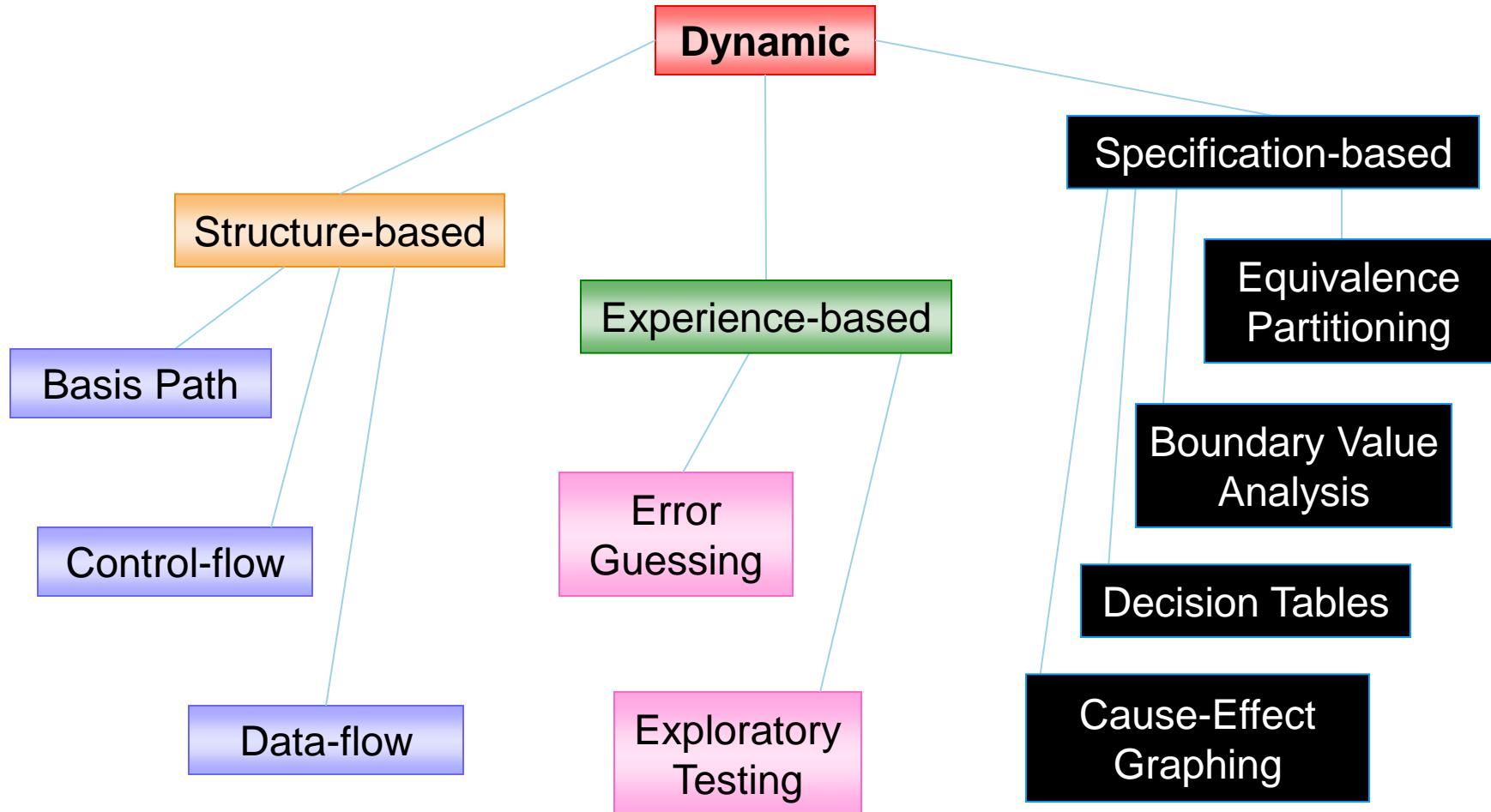
- **Test tĩnh (Static Verification)**

- Thực hiện kiểm chứng mà **không cần thực thi chương trình**
- Kiểm tra tính đúng đắn của các tài liệu có liên quan được tạo ra trong quá trình xây dựng ứng dụng
- Đạt được sự nhất quán và hiểu rõ hơn về hệ thống
- Giảm thời gian lập trình, thời gian và chi phí test,...

- **Test động (Dynamic Testing)**

- Thực hiện kiểm thử dựa trên việc thực thi chương trình

Dynamic Testing - Kiểm thử động



Các kỹ thuật kiểm thử hộp trắng

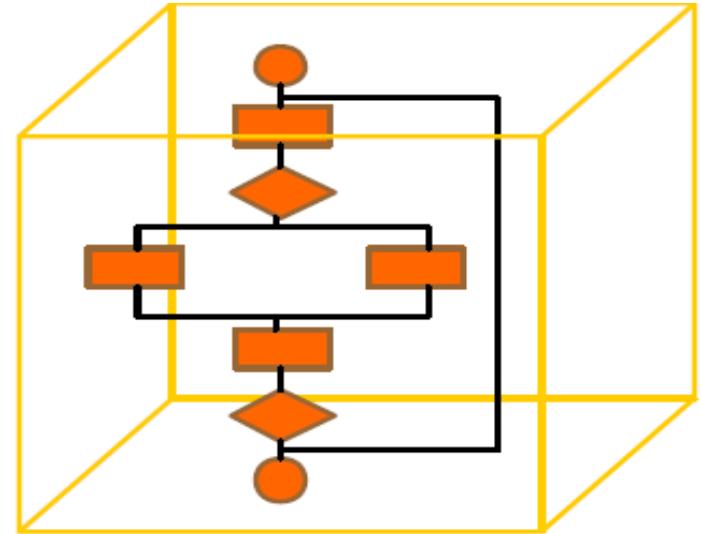


- Basis Path Testing
- Control-flow/Coverage Testing
- Data-flow Testing

Chiến lược kiểm thử hộp trắng



- Thiết kế test case dựa vào cấu trúc nội tại bên trong của đối tượng cần kiểm thử
- Đảm bảo tất cả các câu lệnh, các biểu thức điều kiện bên trong chương trình đều được thực hiện ít nhất một lần



Khái niệm



- Các tên gọi khác: kiểm thử cấu trúc (structural testing), kiểm thử hộp kính (glass box), kiểm thử rõ ràng (clear box testing).
- Đối tượng chính của kiểm thử hộp trắng là tập trung vào cấu trúc bên trong chương trình và tìm ra tất cả những lỗi bên trong chương trình.
- Việc kiểm tra tập trung chủ yếu vào:
 - Cấu trúc chương trình: Những câu lệnh và các nhánh, các loại đường dẫn chương trình.
 - Logic bên trong chương trình và cấu trúc dữ liệu.
 - Những hành động và trạng thái bên trong chương trình.

Ưu, nhược điểm



- **Ưu điểm:**

- Khi sử dụng kiểm thử hộp trắng, kiểm thử viên có thể chắc chắn rằng mọi đường xuyên qua phần mềm cần kiểm thử đã được xác định và kiểm thử

Nhược điểm



- 1. Không đủ khả năng kiểm thử hết các đường thi hành vì số lượng quá nhiều
- 2. Kiểm thử bằng hộp trắng không thể đảm bảo rằng chương trình đã tuân theo đặc tả
- 3. Không phát hiện ra chương trình sai do thiếu đường dẫn
- 4. Không phát hiện được lỗi do sai dữ liệu
- 5. Kiểm thử viên cần có các kỹ năng về lập trình để hiểu và đánh giá được phần mềm. Không may là hiện nay có nhiều kiểm thử viên không có được nền tảng tốt về lập trình

Các kỹ thuật kiểm thử hộp trắng



- Basis Path Testing
- Control-flow/Coverage Testing
- Data-flow Testing

Basis Path Testing



- Được McCabe đưa ra vào năm 1976
- Là phương pháp thiết kế **test case** đảm bảo rằng tất cả các **independent path** trong một code module đều được thực thi ít nhất một lần
- **Independent path:** là bất kỳ path nào trong code mà bổ sung vào ít nhất một tập các lệnh xử lý hay một biểu thức điều kiện (Pressman 2001)
- Cho biết số lượng **test case tối thiểu** cần phải thiết kế khi kiểm thử một code module

Các bước thực hiện

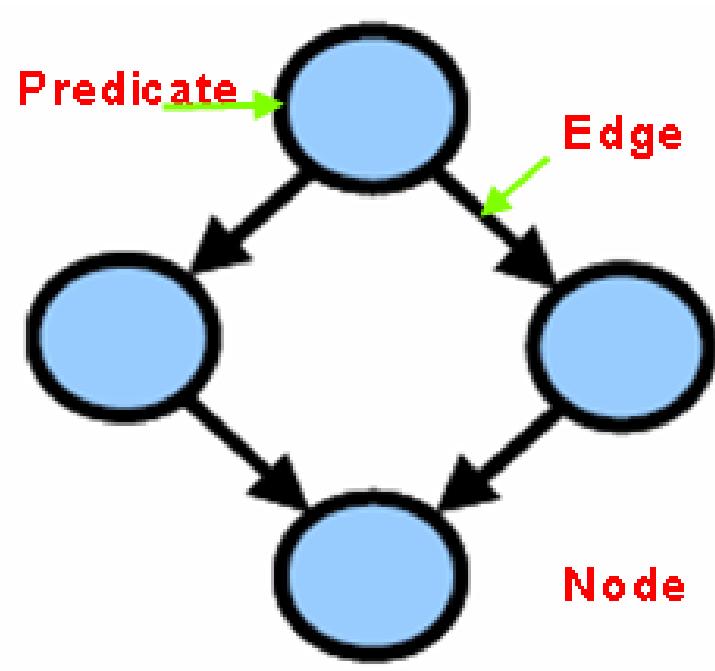


- Xây dựng đồ thị luồng điều khiển
- Tính toán độ phức tạp Cyclomatic
- Chọn ra tập path cơ sở cần test
- Phát sinh test case thực hiện kiểm tra từng path trong tập path cơ sở

Xây dựng đồ thị luồng điều khiển



- Edge: đại diện cho một luồng điều khiển
- Node: đại diện cho một hoặc nhiều câu lệnh xử lý
- Predicate node: đại diện cho một biểu thức điều kiện

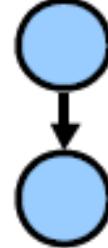


Xây dựng đồ thị luồng điều khiển

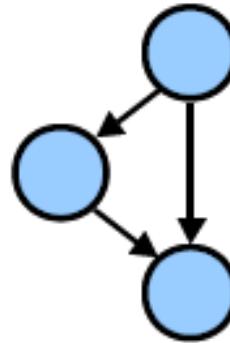


- Một số cấu trúc luồng điều khiển cơ bản

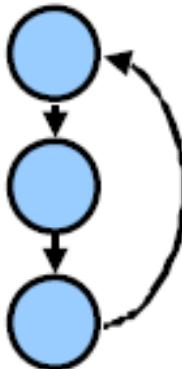
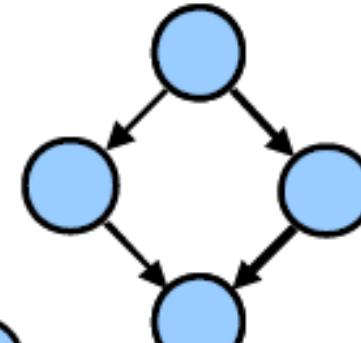
Straight
Line Code



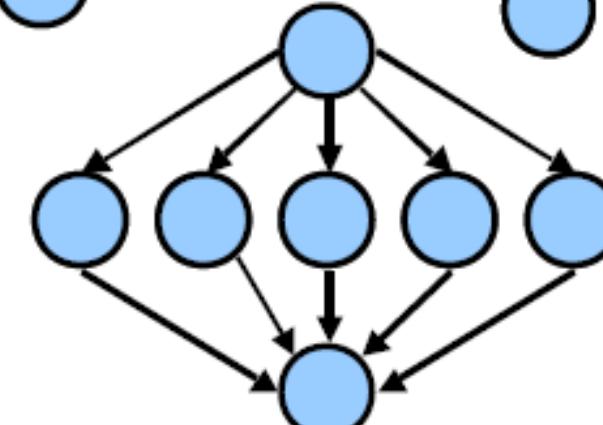
If - Then



If - Then - Else



Loop

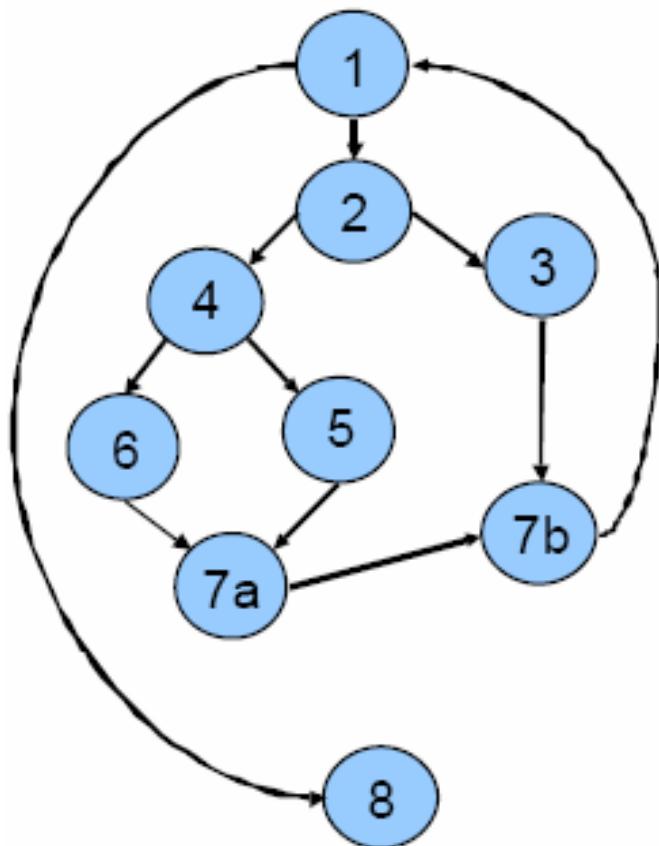


Case Statement

Xây dựng đồ thị luồng điều khiển



- Đồ thị luồng điều khiển từ một đoạn



1. do while (records remain)
read record;
2. if (record field 1 = 0) then
3. process record;
 store in buffer;
 increment counter;
4. elseif (record field 2 = 0) then
5. reset record;
6. else
 process record;
 store in file;
- 7a. endif;
- 7b. enddo;
8. end;

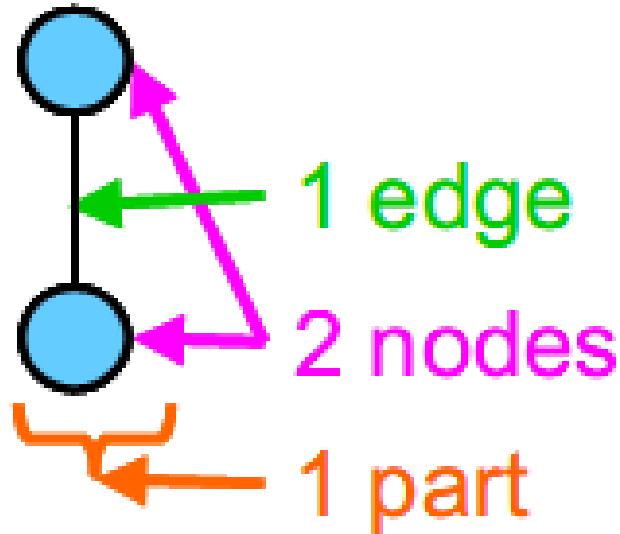
Tính toán độ phức tạp Cyclomatic



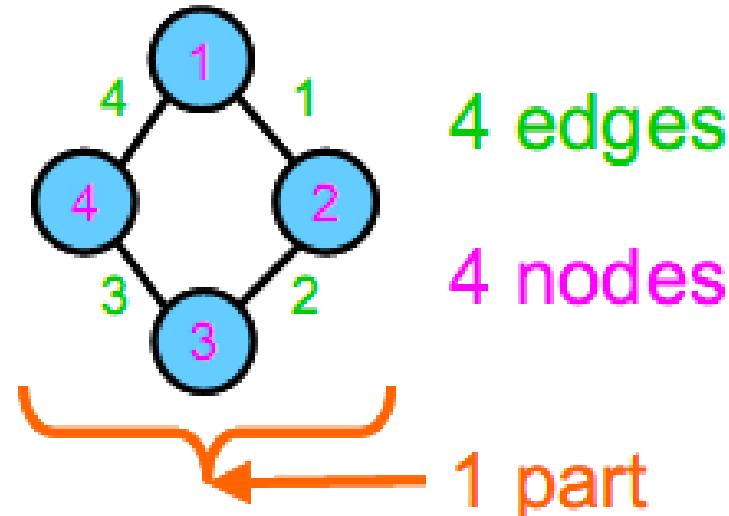
- Cách 1: Dựa trên công thức của McCabe

- $V(G) = \text{edges} - \text{nodes} + 2p$

- $p = \text{number of unconnected parts of the graph}$

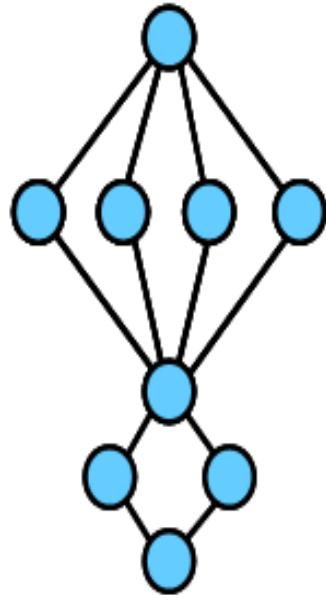


$$V(G) = 1 - 2 + 2 \times 1 = 1$$



$$V(G) = 4 - 4 + 2 \times 1 = 2$$

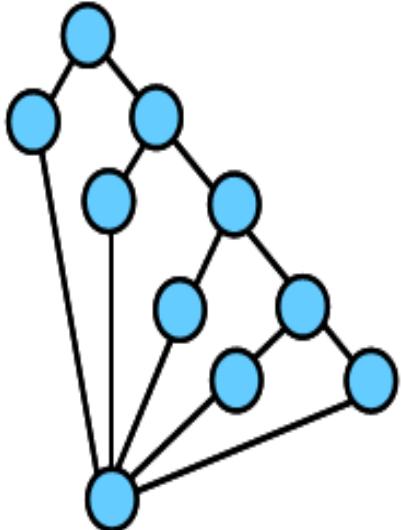
Tính toán độ phức tạp Cyclomatic



12 edges

9 nodes

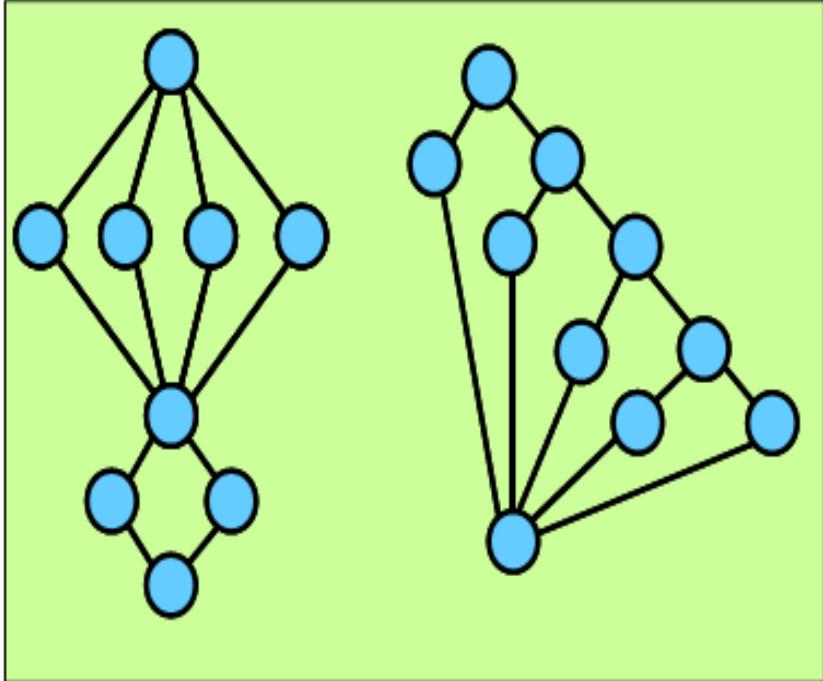
$$12 - 9 + 2(1) = 5$$



13 edges

10 nodes

$$13 - 10 + 2(1) = 5$$



25 edges

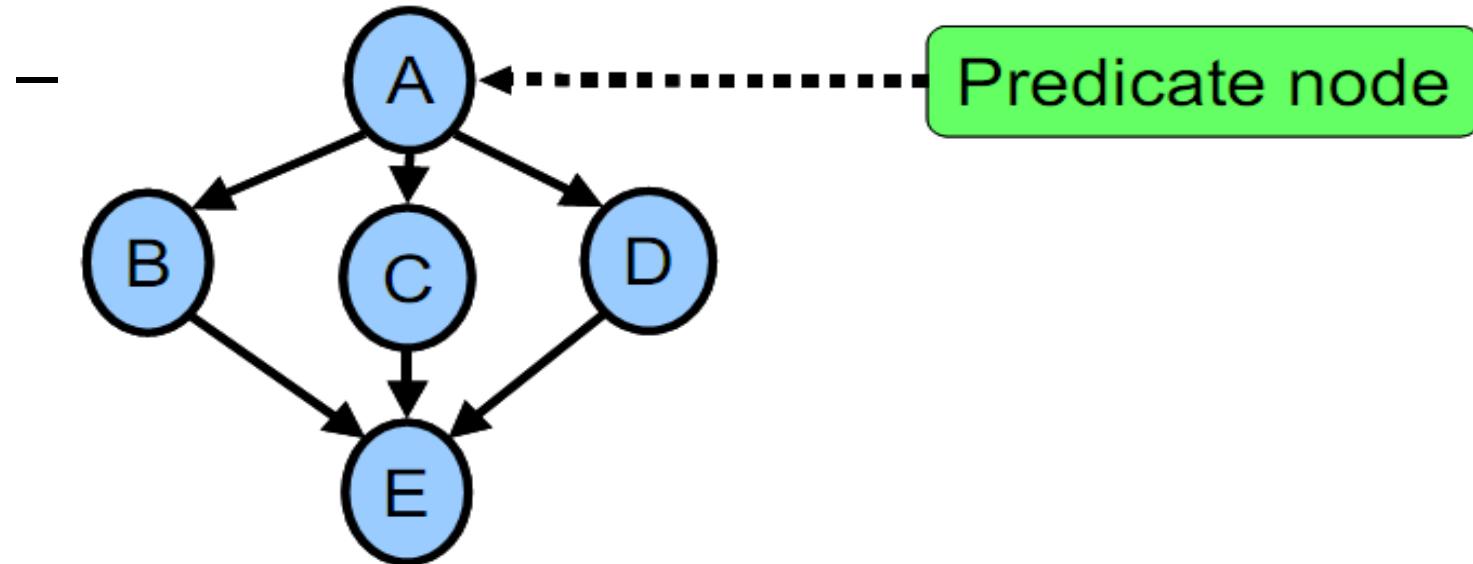
19 nodes

$$25 - 19 + 2(2) = 10$$

Tính toán độ phức tạp Cyclomatic

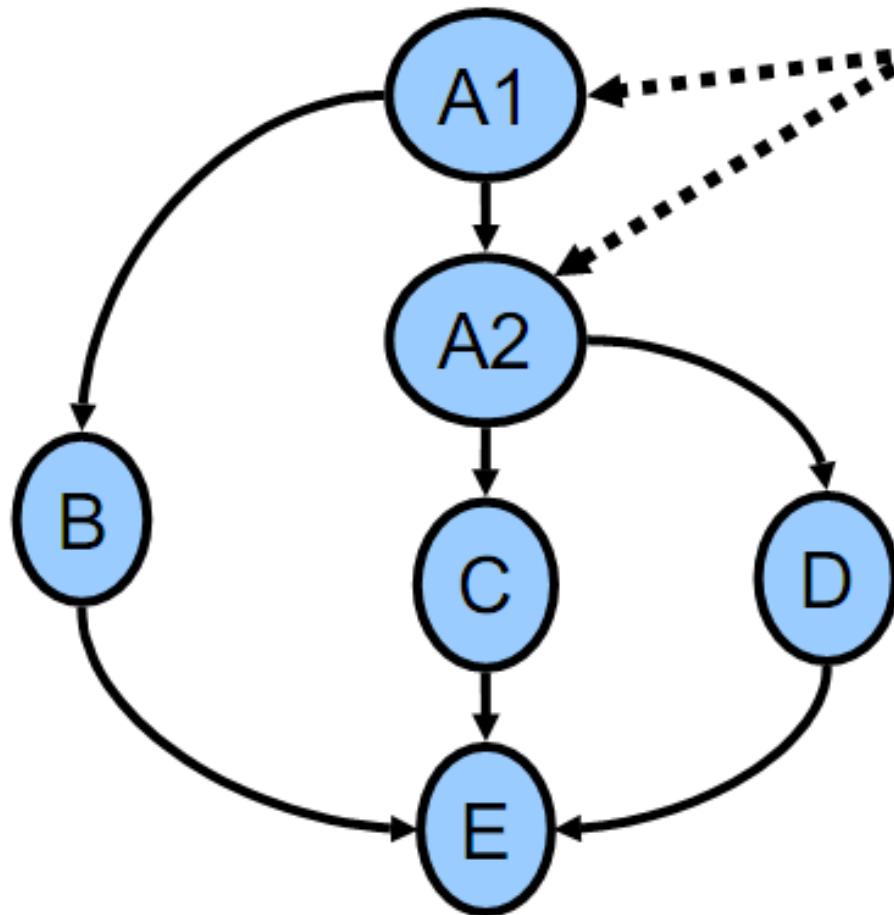


- Cách 2: Dựa vào số lượng Predicate Node



$$\text{McCabe: } V(G) = 6 - 5 + 2(1) = 3$$

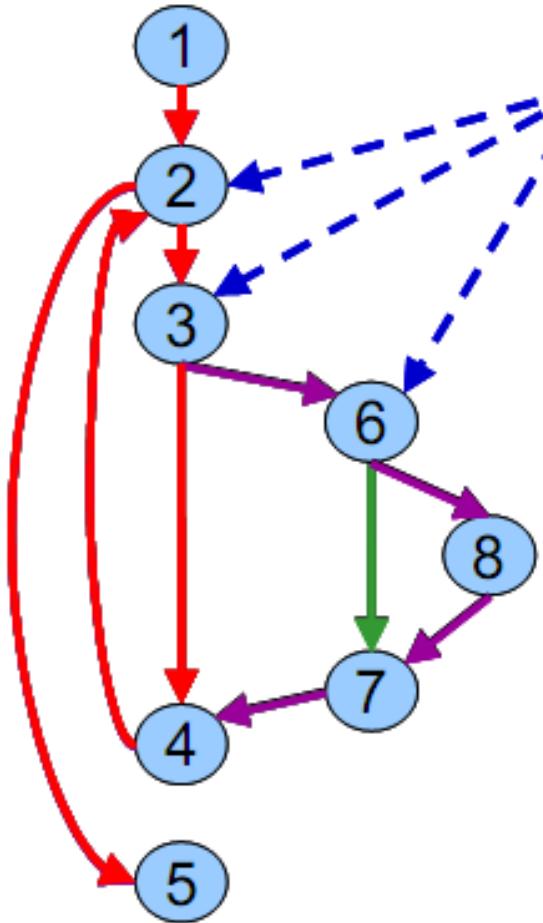
Tính toán độ phức tạp Cyclomatic



Predicate nodes

$$V(G) = 2 + 1 = 3$$

Chọn ra tập path cơ sở cần test



Predicate Nodes

Path 1: 1,2,5

Path 2: 1,2,3,4,2,5

Path 3: 1,2,3,6,7,4,2,5

Path 4: 1,2,3,6,8,7,4,2,5

Phát sinh test case



Test case 1	Path 1: 1,2,5
Test case 2	Path 2: 1,2,3,4,2,5
Test case 3	Path 3: 1,2,3,6,7,4,2,5
Test case 4	Path 4: 1,2,3,6,8,7,4,2,5

Độ phức tạp chu trình



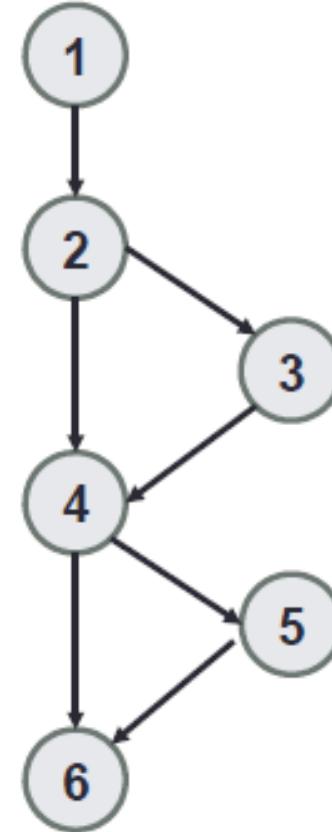
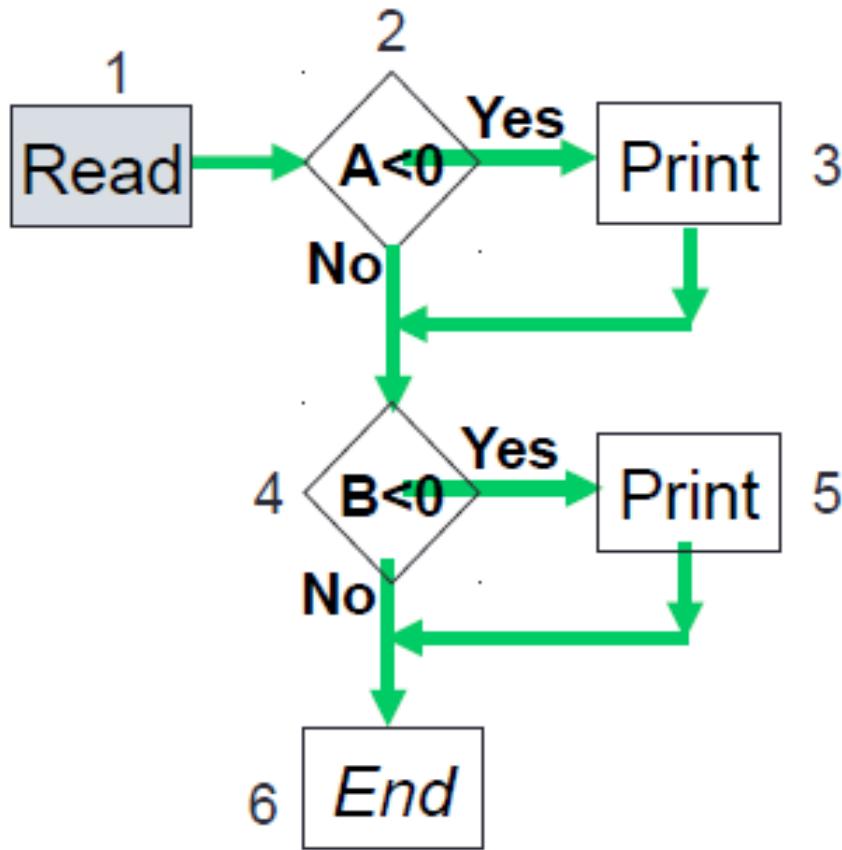
- Độ phức tạp chu trình
 - Là số đo sự phức tạp logic của chương trình.
 - Là số các đường đi độc lập cơ bản trong tập các con đường độc lập của một chương trình.
 - Là số đường độc lập nhỏ nhất phủ hết các cạnh của đồ thị luồng.
 - Số đo này là giới hạn trên của số ca kiểm thử cần phải tiến hành để đảm bảo rằng, tất cả các câu lệnh trong chương trình đều được thực hiện ít nhất một lần.

Độ phức tạp chu trình



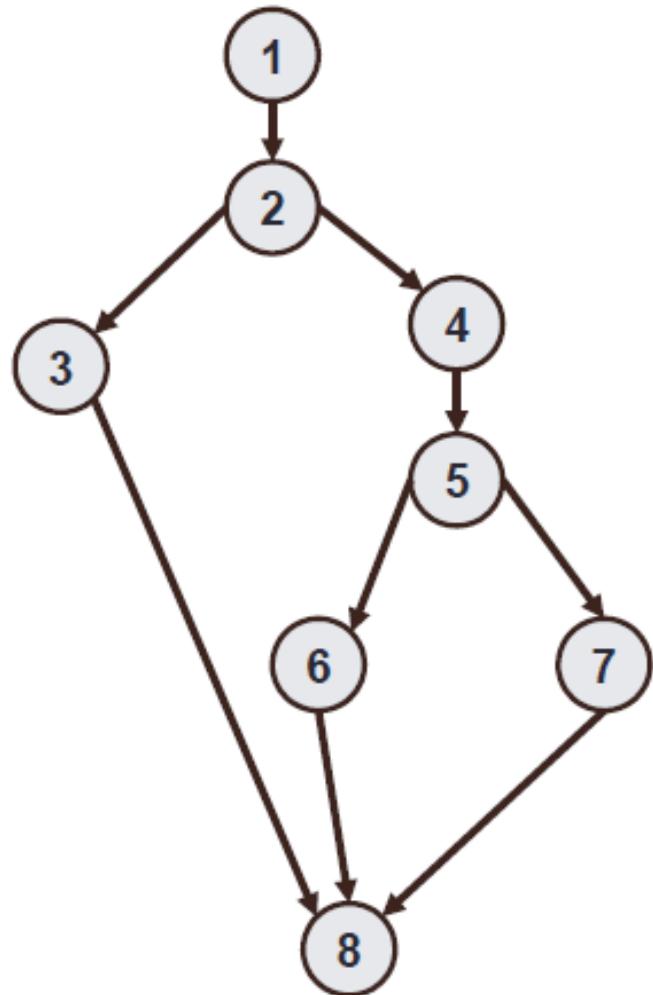
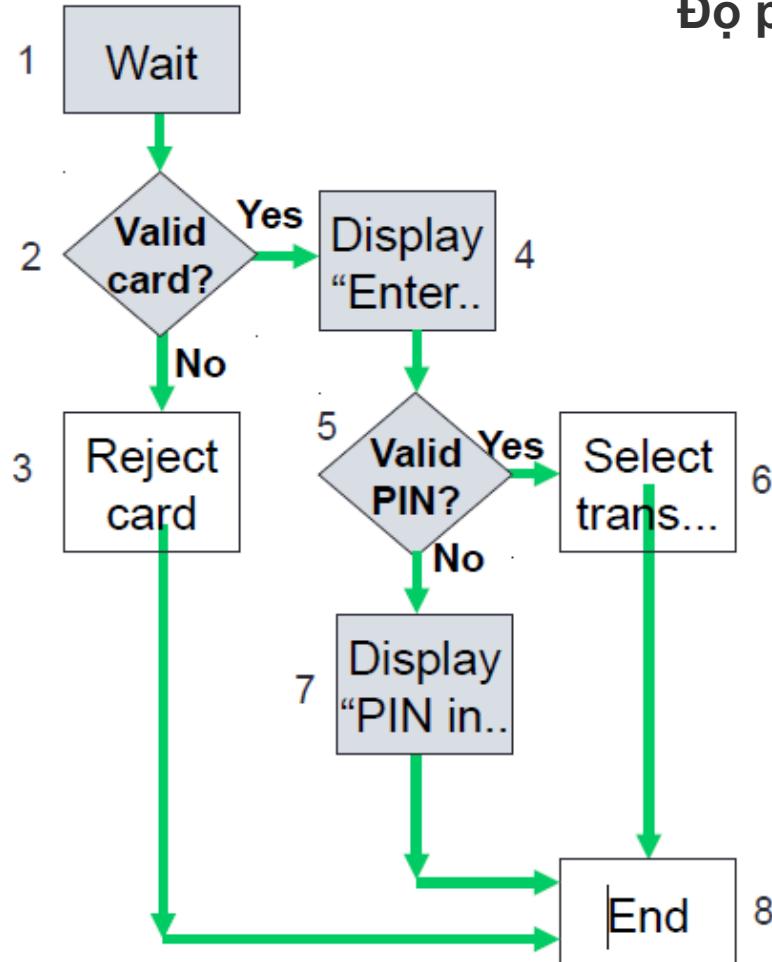
- Độ phức tạp Cyclomatic $C = V(G)$ của đồ thị dạng điều khiển được tính bởi 1 trong các công thức sau :
- $V(G) = E - N + 2$, trong đó E là số cung, N là số nút của đồ thị.
- $V(G) = P + 1$, P là số nút quyết định
- Độ phức tạp của chu trình (Cyclomatic) C chính là **số đường thi hành tuyến tính độc lập** của TPPM cần kiểm thử.

Độ phức tạp của chu trình



Độ phức tạp của chu trình = $7 - 6 + 2 = 3$

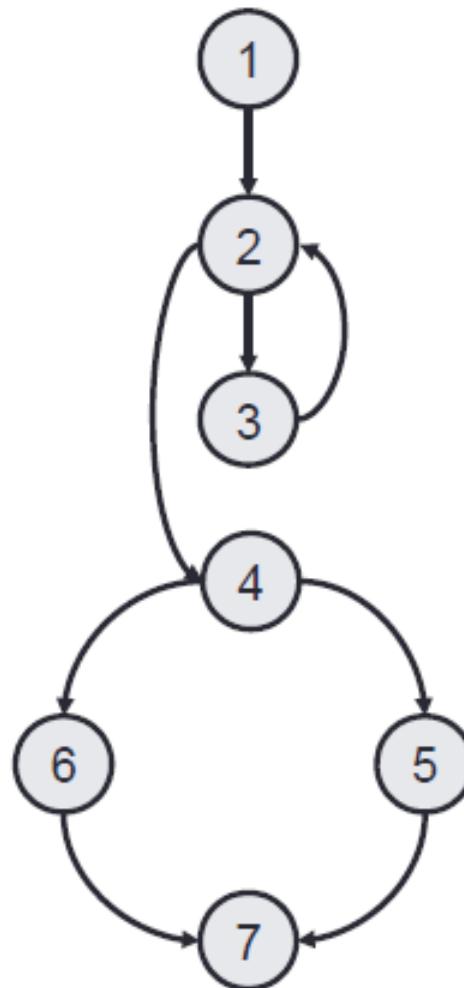
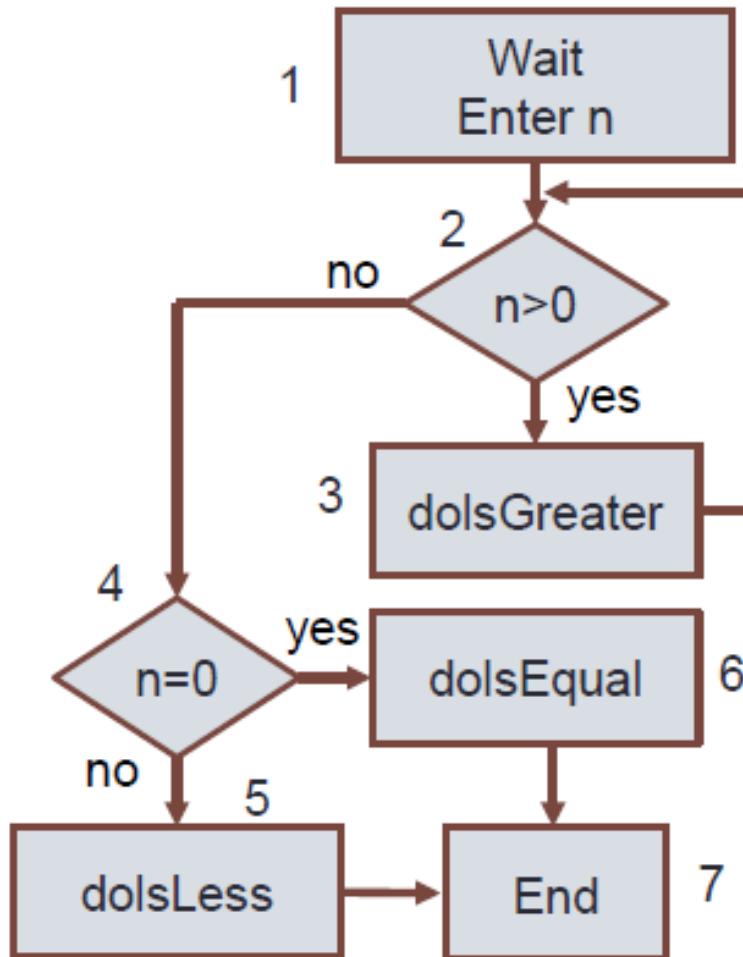
Chuyển sang đồ thị dòng tính độ phức tạp



Ví dụ



Độ phức tạp của chu trình C=8-7+2=3



Ví dụ



Read A

IF A > 0 THEN

IF A = 21 THEN

 Print “Key”

ENDIF

ENDIF

1. Vẽ lưu đồ

2. Xác định độ phức tạp chương trình=3

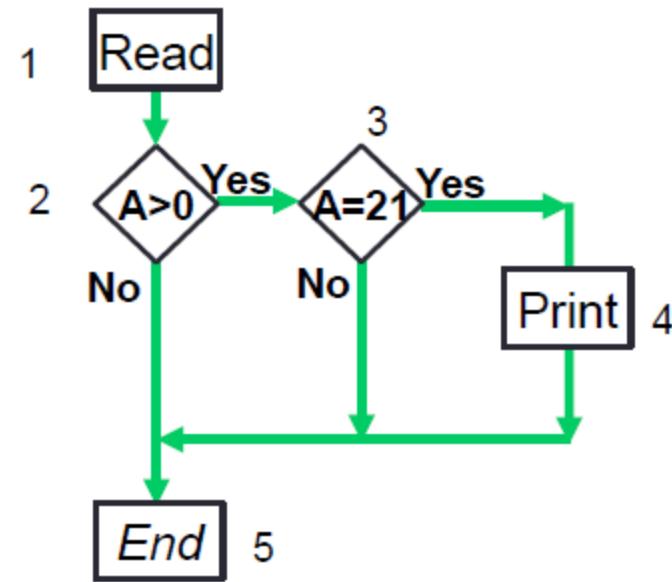
3. Xác định đường thi hành cơ bản

1. 1-2-3-4-5

2. 1-2-5

3. 1-2-3-5

4. Các ca kiểm thử



TC	Đầu vào	Đầu ra mong đợi	Kết quả
1	A=21	In ra “Key”	
2	A=-5	Kết thúc CT	

Các cấp bao phủ kiểm thử



- **Phủ kiểm thử (coverage)**: tỉ lệ các thành phần thực sự được kiểm thử so với tổng thể các thành phần.
- Các thành phần bao gồm: lệnh thực thi, điểm quyết định, điều kiện con hay sự kết hợp của chúng.
- Độ phủ càng lớn thí độ tin cậy càng cao.

Các cấp bao phủ kiểm thử



- *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 kiểm thử không có trách nhiệm
- *Phủ cấp 1: Bao phủ câu lệnh (statement coverage):* Các câu lệnh được thực hiện ít nhất 1 lần
- *Phủ cấp 2: Bao phủ nhánh (branch coverage):* tại các điểm quyết định thì các nhánh đều được thực hiện ở cả hai phía T,F
- *Phủ cấp 3: Bao phủ điều kiện(condition coverage):* Các điều kiện con của các điểm quyết định được thực hiện ít nhất 1 lần
- *Phủ cấp 4: Kết hợp phủ nhánh và điều kiện (branch & condition coverage)*

Control-flow/Coverage Testing



- Là kỹ thuật thiết kế test case đảm bảo “cover” được tất cả các câu lệnh, biểu thức điều kiện trong code module cần test
- Có bốn tiêu chí đánh giá độ bao phủ
 - Method Coverage (phương thức)
 - Statement Coverage (câu lệnh)
 - Decision/Branch Coverage (biểu thức điều kiện)
 - Condition Coverage (biểu thức điều kiện đơn)

Method Coverage



- Tỷ lệ phần trăm các **phương thức** trong chương trình được gọi thực hiện bởi các test case
- Test case cần phải đạt được 100% method coverage

Ví dụ - Method Coverage

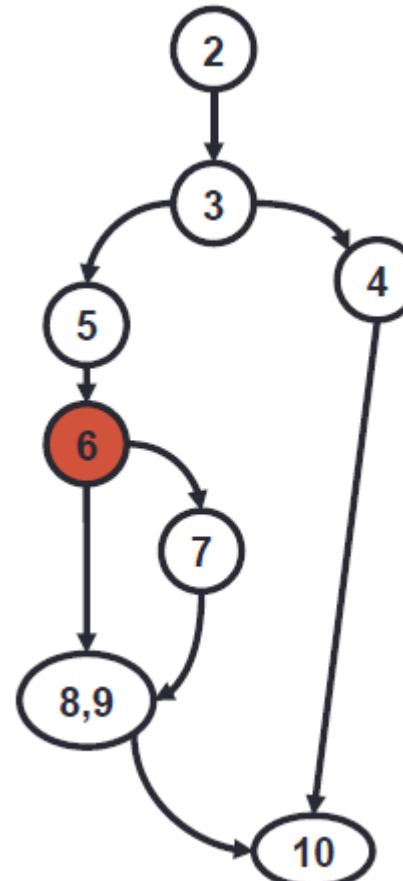
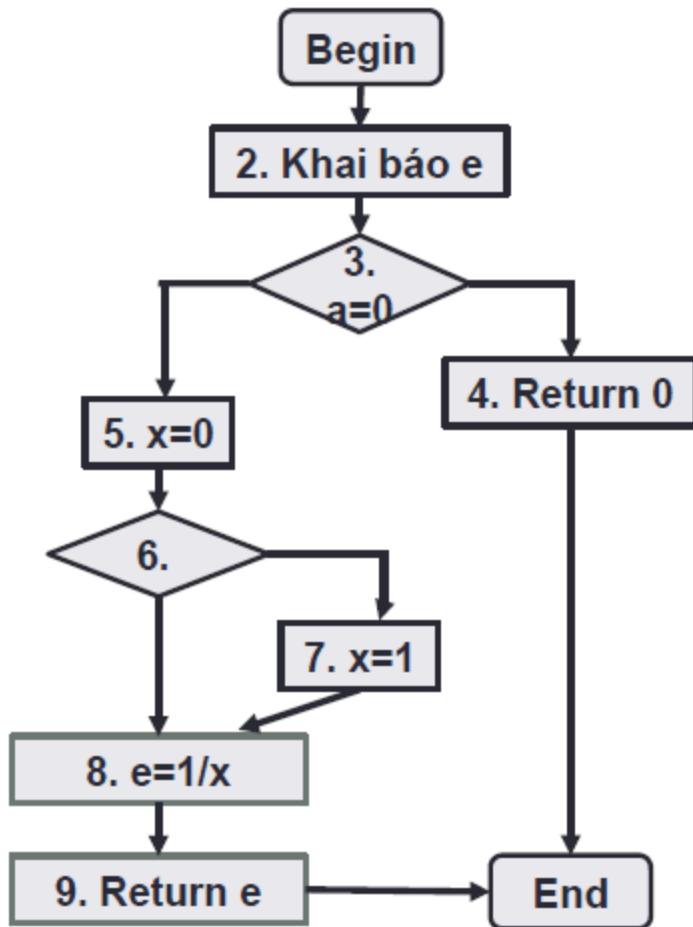


- Xét đoạn chương trình

- Test case 1: foo (0,0,0,0,0)
- 100% method coverage

```
1 float foo (int a, int b, int c, int d, float e) {  
2     float e;  
3     if (a == 0) {  
4         return 0;  
5     }  
6     int x = 0;  
7     if ((a == b) OR ((c == d) AND bug(a))) {  
8         x = 1;  
9     }  
10    e = 1/x;  
11    return e;  
12 }
```

Vd: Đồ thị dòng



Statement Coverage



- Tỷ lệ phần trăm các **câu lệnh** trong chương trình được gọi thực hiện bởi các test case
- Test case 1 thực hiện các lệnh từ 1→5 trong 12 câu lệnh đạt 42% Statement Coverage
- Để đạt 100% Statement Coverage
→Test case 2: foo (1,1,1,1,1)

```
1 float foo (int a, int b, int c, int d, float e) {  
2     float e;  
3     if (a == 0) {  
4         return 0;  
5     }  
6     int x = 0;  
7     if ((a==b) OR ((c == d) AND bug(a))) {  
8         x=1;  
9     }  
10    e = 1/x;  
11    return e;  
12 }
```

Decision/Branch Coverage



- Tỷ lệ phần trăm các **biểu thức điều kiện** trong chương trình được ước lượng giá trị trả về (**true**, **false**) khi thực thi các test case
- Một biểu thức điều kiện (**cho dù là single hay complex**) phải được kiểm tra trong cả hai trường hợp giá trị của biểu thức là **true** hay **false**
- Đối với các hệ thống lớn, thường chỉ đạt từ **75%** → **85% độ bao phủ**

Decision/Branch Coverage



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

→ Đạt 75% coverage

→ Test case 3: **foo (1,2,1,2,1)** → 100% coverage

Condition Coverage



- Tỷ lệ phần trăm các biểu thức điều kiện đơn trong biểu thức điều kiện phức của chương trình được ước lượng giá trị trả về (true, false) khi thực thi các test case
- Ví dụ: 50% coverage

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

```
1 float foo (int a, int b, int c, int d, float e) {  
2     float e;  
3     if (a == 0) {  
4         return 0;  
5     }  
6     int x = 0;  
7     if ((a==b) OR ((c == d) AND bug(a))) {  
8         x=1;  
9     }  
10    e = 1/x;  
11    return e;  
12 }
```

Condition Coverage



- Thiết kế thêm Test case 4, 5 để đạt 100%

Predicate	True	False
(a==b)	Test Case 2 foo(1, 1, x, x, 1) return value 0	Test Case 3 foo(1, 2, 1, 2, 1) division by zero!
(c==d)	Test Case 4 foo(1, 2, 1, 1, 1) return value 1	Test Case 3 foo(1, 2, 1, 2, 1) division by zero!
bug(a)	Test Case 4 foo(1, 2, 1, 1, 1) return value 1	Test Case 5 foo(3, 2, 1, 1, 1) division by zero!

```
1 float foo (int a, int b, int c, int d, float e) {  
2     float e;  
3     if (a == 0) {  
4         return 0;  
5     }  
6     int x = 0;  
7     if ((a==b) OR ((c == d) AND bug(a) )) {  
8         x=1;  
9     }  
10    e = 1/x;  
11    return e;  
12 }
```

Data-flow Testing



- Là kỹ thuật thiết kế **test case** dựa vào việc khảo sát sự thay đổi trạng thái trong chu kỳ sống của các biến trong chương trình
- **Ví dụ:** Một số pattern lỗi thường gặp
 - Sử dụng biến mà chưa khai báo
 - Sử dụng biến đã hủy trước đó
 - ...

Hệ thống ký hiệu trạng thái dữ liệu



Hệ thống ký hiệu

d	defined, created, initialized
k	killed, terminated, undefined
u	used c – used in a computation (sử dụng trong biểu thức tính toán) p – used in a predicate (sử dụng trong các biểu thức điều kiện)
$\sim x$	Cho biết trước khi tất cả hành động liên quan đến x
$x\sim$	Cho biết tất cả hành động không có thông báo liên quan đến x

Một số ví dụ



- $v = \text{expression}$
 - $c - \text{use}$ của các biến trong biểu thức
 - definition của v
- $\text{read}(v_1, v_2, \dots, v_n)$
 - definitions của v_1, \dots, v_n
- $\text{write}(v_1, v_2, \dots, v_n)$
 - $c - \text{uses}$ của v_1, \dots, v_n
- $\text{method call: } P(c_1, \dots, c_n)$
 - definition của mỗi tham số
- $\text{While } B \text{ do } S$
 - $p - \text{use}$ của mỗi biến trong biểu thức điều kiện

Ví dụ



```
1. read (x, y);  
2. z = x + 2;  
3. if (z < y)  
4.     w = x + 1;  
    else  
5.     y = y + 1;  
6. print (x, y, w, z);
```

<i>Def</i>	<i>C-use</i>	<i>P-use</i>
x, y		
z	x	
		z, y
w	x	
y	y	
		x, y,
		w, z

Các chiến lược thiết kế test case



- All-du paths (ADUP)
- All-Uses (AU)
- All-p-uses (APU)
- All-c-uses (ACU)
- All-p-uses / Some-c-uses (APU+C)
- All-c-uses / Some-p-uses (ACU+P)
- All-definition (AD)

Ví dụ



```
public static double calculateBill (int usage)
{
    double bill = 0;

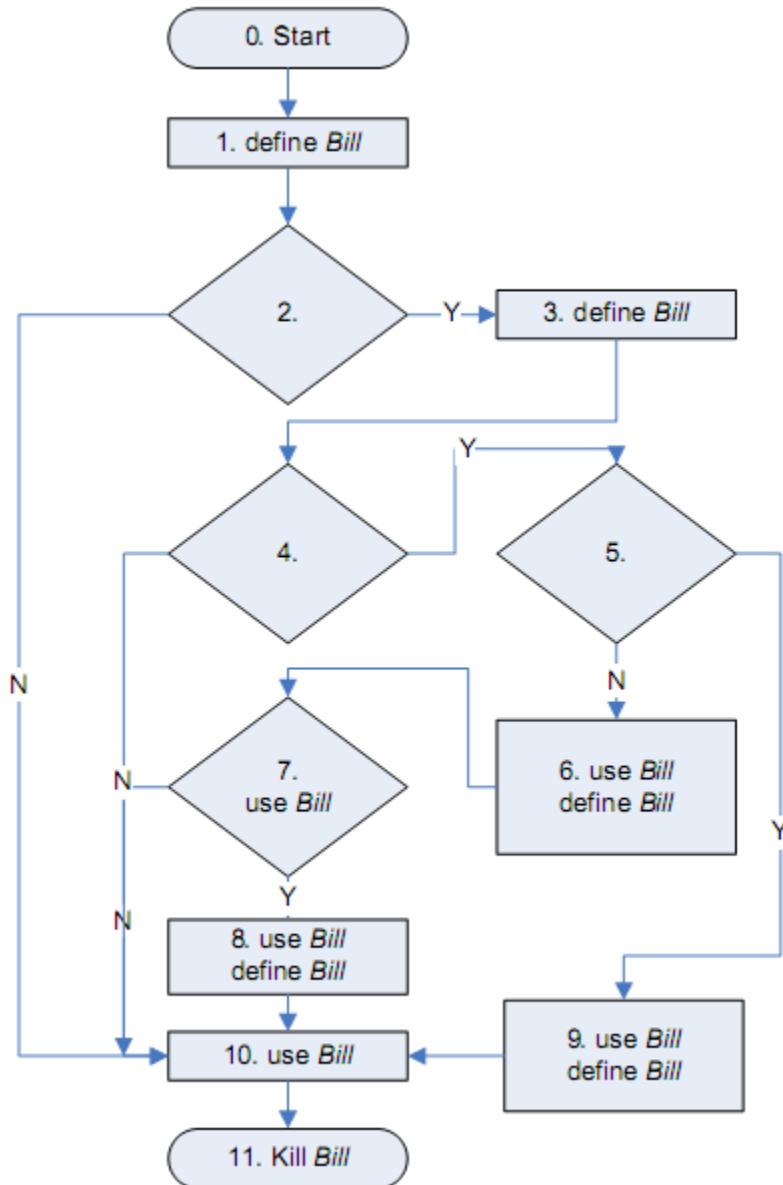
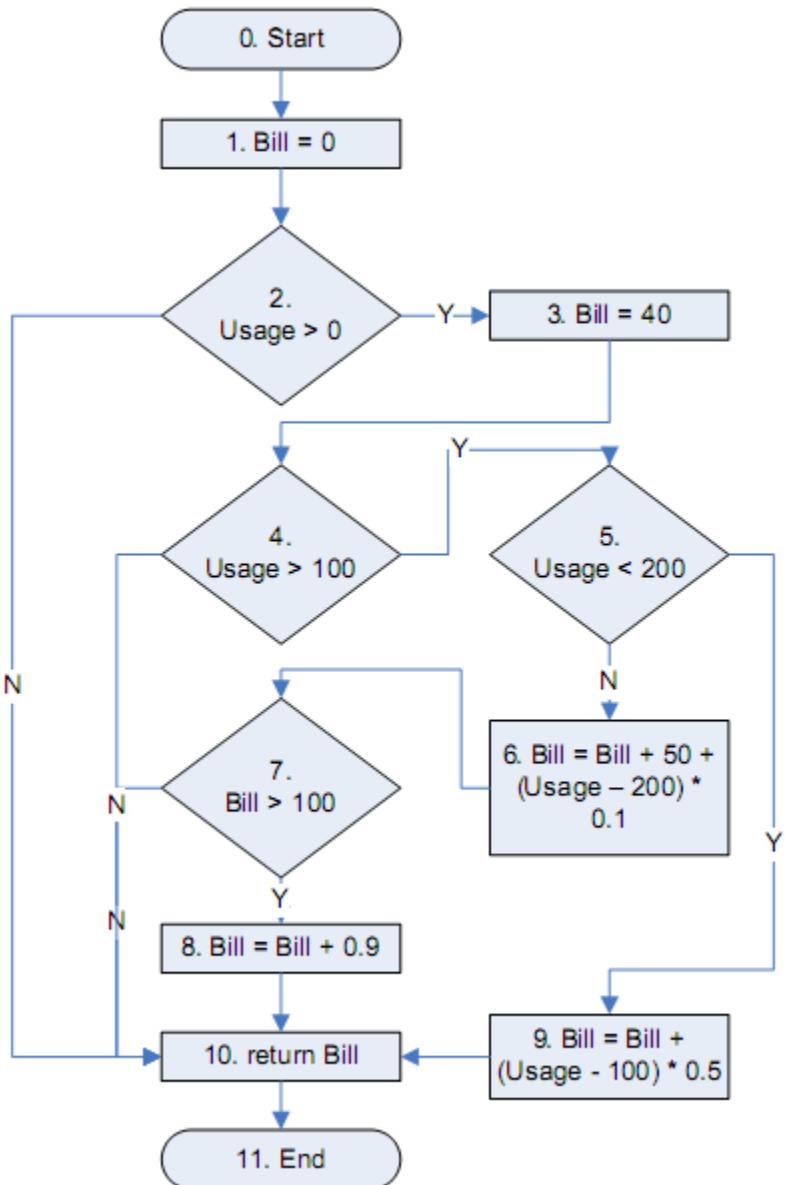
    if(usage > 0)
    {
        bill = 40;
    }

    if(usage > 100)
    {
        if(usage <= 200)
        {
            Bill = bill + (usage - 100) * 0.5;
        }
        else
        {
            Bill = bill + 50 + (usage - 200) * 0.1;

            if(bill >= 100)
            {
                bill = bill * 0.9;
            }
        }
    }

    return bill;
}
```

Xét biến “Bill”

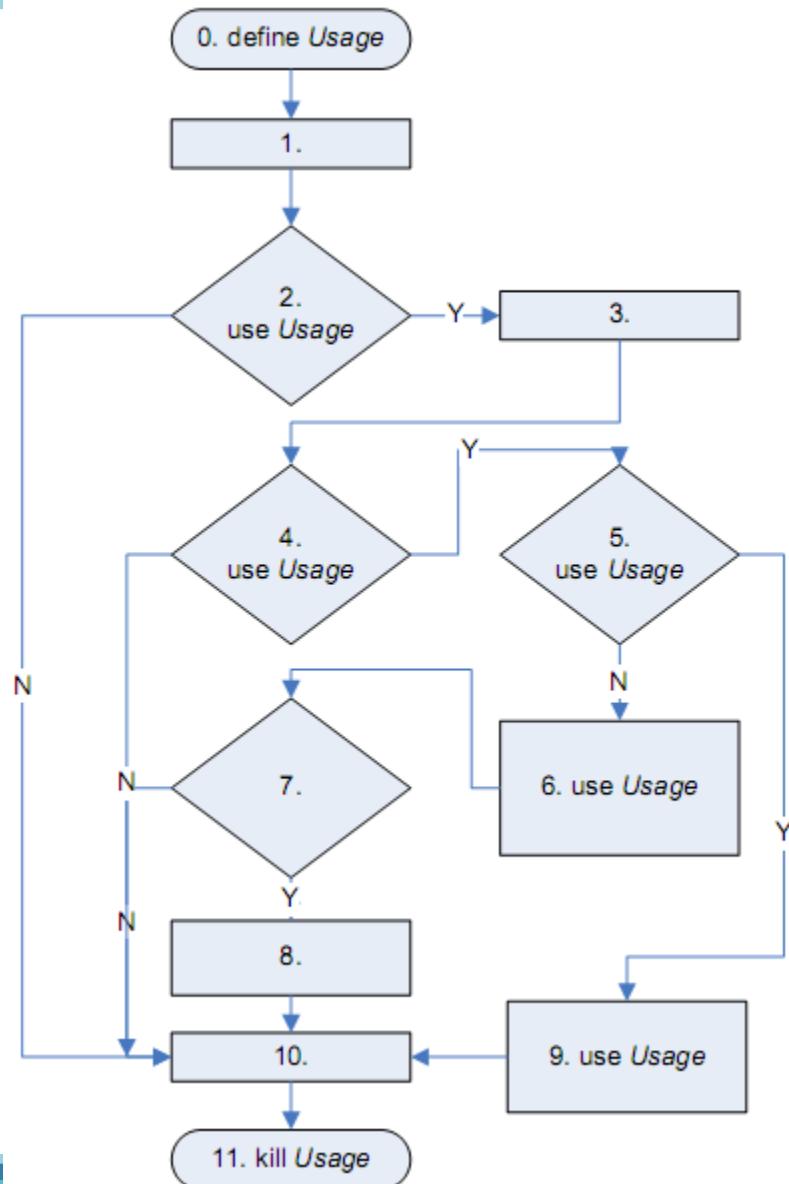
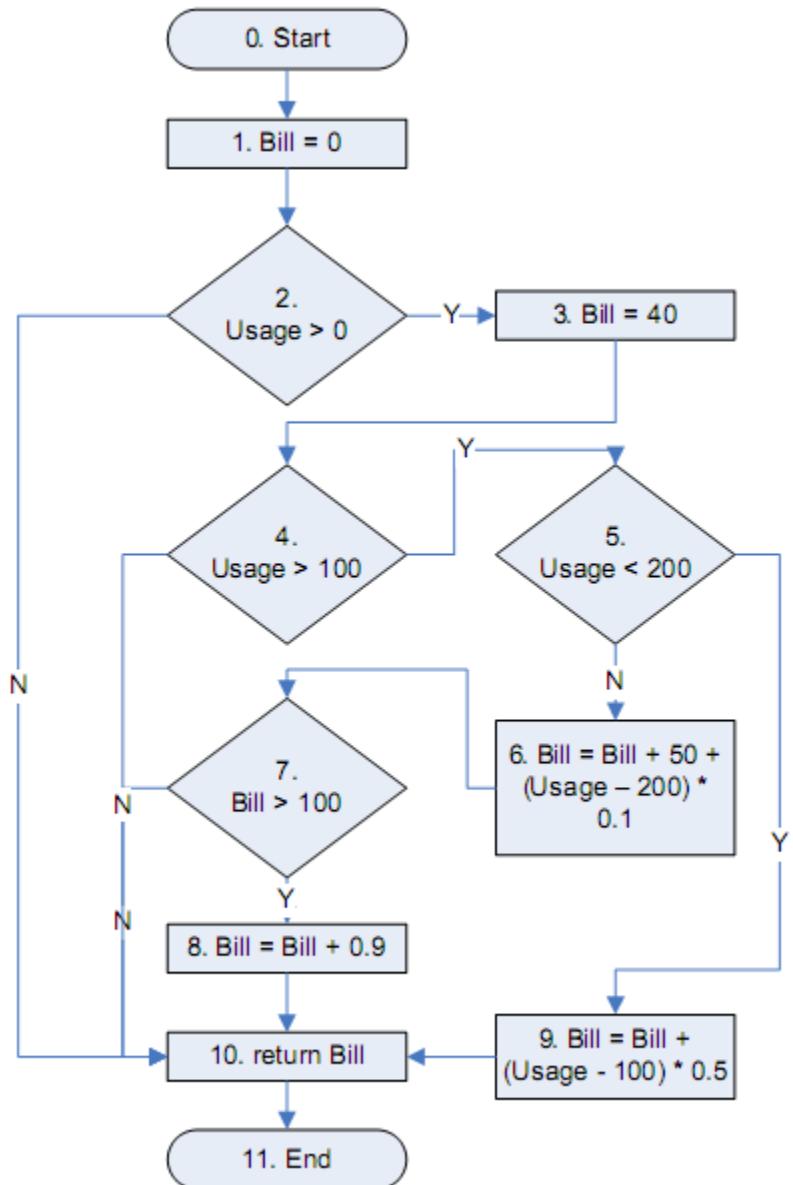


Bảng mô tả biến “Bill”



Anomaly		Explanation
~d	0-1	Allowed. Normal case
dd	0-1-2-3	Potential bug. Double definition.
du	3-4-5-6	Allowed. Normal case.
ud	6	Allowed. Data is used and then redefined.
uk	10-11	Allowed.
dd	1-2-3	Potential bug. Double definition.
uu	7-8	Allowed. Normal case.
k~	11	Allowed. Normal case.

Xét biến “Usage”



Bảng mô tả biến “Usage”



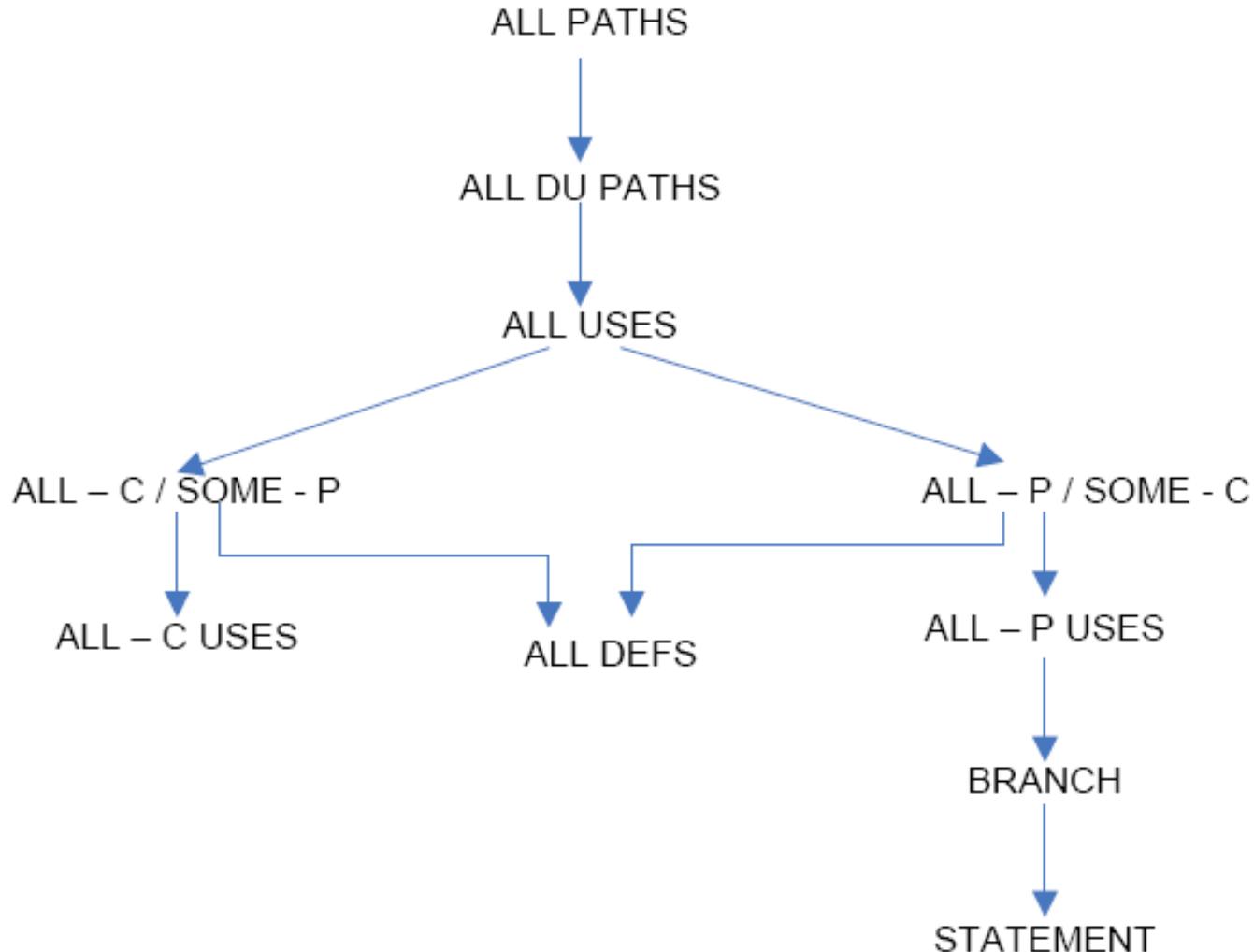
Anomaly	Explanation
~d	0
du	0-1-2
uk	9-10-11
uu	5-6
k~	11

Data-flow testing paths for each variable



Strategy	Bill	Usage	Strategy	Bill	Usage
All uses (AU)	3-4-5-6 6-7 6-7-8 8-10 3-4-5-9	0-1-2 0-1-2-3-4 0-1-2-3-4-5 0-1-2-3-4-5-6 0-1-2-3-4-5-9	All p – use/ some c (APU + C)	1-2-3-4-5-6-7 3-4-5-6-7 6-7 8-10 9-10	0-1-2 0-1-2-3-4 0-1-2-3-4-5
All p – uses (APU)	1-2-3-4-5-6-7 3-4-5-6-7 6-7	0-1-2 0-1-2-3-4 0-1-2-3-4-5	All c – use/ some p (ACU + P)	1-2-10 3-4-5-6 3-4-5-9 6-7-8 8-10 9-10	0-1-2-3-4-5-6 0-1-2-3-4-5-9
All c – uses (ACU)	1-2-10 3-4-5-6 3-4-5-9 3-4-10 6-7-8 6-7-10 8-10 9-10	0-1-2-3-4-5-6 0-1-2-3-4-5-9	All du (ADUP)	(ACU+P) + (APU+C)	(ACU+P) + (APU+C)
			All definition (AD)	1-2-10 3-4-5-6 6-7 8-10 9-10	0-1-2

Mối quan hệ giữa các chiến lược data-flow test



Các công cụ hỗ trợ kiểm thử



- Các công cụ hỗ trợ quản lý quá trình kiểm thử
- Các công cụ hỗ trợ thực hiện các kỹ thuật kiểm thử
 - Công cụ kiểm thử hiệu năng (Performance)
 - Công cụ kiểm thử chức năng (Functional)
 - Công cụ kiểm thử bảo mật (Security)
 - Công cụ kiểm thử đơn vị (UnitTesting)

Các công cụ hỗ trợ quản lý quy trình kiểm thử phần mềm (1)



- Các đối tượng cần quản lý của 1 công cụ kiểm thử PM
 - Project
 - User
 - User Role
 - Requirement
 - Release: Phiên bản của project.
 - Test Plan: Kế hoạch test.
 - Test types: Các loại test.
 - Test cases: Các trường hợp test
 - Teststep: Các bước thực hiện cho mỗi test case
 - Result: Kết quả thực thi test.
 - Bug: Lỗi
 - Reports: Các thông báo về tình trạng của tiến trình: Tình trạng lỗi, tiến triển của công việc: ...
 - Các tài liệu hướng dẫn sử dụng chương trình (Help)

Các công cụ hỗ trợ quản lý quy trình kiểm thử phần mềm (2)



- Các chức năng cần phải có
 - Quản lý project.
 - Quản lý User.
 - Phân quyền User.
 - Quản lý requirement theo phiên bản.
 - Quản lý release.
 - Quản lý các thành phần của release: build, component,..
 - Quản lý testplan.
 - Quản lý testcase.
 - Cập nhật kết quả cho test case.
 - Cập nhật tình trạng lỗi.
 - Thống kê lỗi cho mỗi release hoặc mỗi thành phần của release.
 - Tự động cập nhật kết quả kiểm thử

Các công cụ hỗ trợ quản lý quy trình kiểm thử phần mềm (3)



No	Name	Desc	REQ	Download
1	TestLink		Apache, MySQL, PHP	48797
2	Fitnessse		Mac, Windows, POSIX	24475
3	QATraq		Windows, BSD, Linux, SunOS/Solaris	21992
4	Bugzilla Test Runner		Bugzilla 2.16.3 or above	17291
5	rth		All 32-bit MS Windows (95/98/NT/2000/XP), All POSIX (Linux/BSD/UNIX-like OSes), IBM AIX	9563
6	TestMaster		Linux, Apache, PostgreSQL	6728
7	TCW		Any (PHP/SQL/Apache)	4488
8	Tesly		OS Independent	3327
9	qaProjectManager		Platform Independent	3133
10	Testitool		Apache, PHP, MySQL	701

Công cụ kiểm thử hiệu năng



- Là một dạng kiểm tra tự động nhằm tìm ra những điểm “thắt cổ chai” của phần mềm, giúp cho người phát triển có những thay đổi thích hợp để tăng khả năng thực thi, tốc độ xử lý của phần mềm
- Giúp người kiểm tra xác định được những thông số ngưỡng của phần mềm, đề ra tiêu chuẩn cho những lần kiểm tra sau
- Thường được áp dụng đối với các PM được triển khai trên môi trường nhiều người dùng (ví dụ: ứng dụng web)
- Kết quả mong đợi của việc kiểm thử hiệu năng phải được định nghĩa một cách rõ ràng
- Ví dụ:
 - Số kết nối (session) đồng thời mà server có thể phục vụ
 - Thời gian (bao nhiêu phút/giây) mà trình duyệt nhận được kết quả từ server
 -

Công cụ kiểm thử hiệu năng



No	Name	Requirements	Download
1	<u>OpenSTA</u>	Windows 2000, NT4 and XP	251965
2	<u>Grinder</u>	OS Independent	156458
3	<u>TPTEST</u>	MacOS/Carbon and Win32	108036
4	<u>Database Opensource Test Suite</u>	Linux, POSIX	103484
5	<u>Sipp</u>	Linux/Unix/Win32-Cygwin	102111
6	<u>WebLOAD</u>	32-bit MS Windows (NT/2000/XP), Linux, Windows Server 2003	39401
7	<u>OpenWebLoad</u>	Linux, DOS	31204
8	<u>Hammerhead 2 - Web Testing Tool</u>	Hammerhead has been used with Linux, Solaris and FreeBSD.	24814
9	<u>Dieseltest</u>	Windows	14618
10	<u>DBMonster</u>	OS Independent	13710

Các công cụ hỗ trợ kiểm thử đơn vị



- Có rất nhiều công cụ kiểm thử đơn vị được viết bằng nhiều ngôn ngữ khác nhau
 - ADA
 - C++
 - HTML
 - Java
 - .NET
 - Pert
 - PHP
 - SQL
 - XML
 - Ruby
 - ...

Các công cụ hỗ trợ kiểm thử đơn vị



No	Name	Requirements	Download
1	JUnit	OS Independent	2151874
2	Findbugs	JRE (or JDK) 1.4.0 or later	379779
3	PMD	JDK 1.3 or higher	344688
4	Checkstyle	OS Independent	216780
5	EclEmma	Eclipse	209153
6	Dbunit	JUnit	129300
7	StrutsTestCase for JUnit v1.9.5	OS Independent	106860
8	Emma	Java	59435
9	MockObjects	OS independent	55457
10	JUnitEE	JUnit	54618

www.opensourcetestingtools.org

Các công cụ hỗ trợ kiểm thử đơn vị



No	Name	Requirements	Download
1	<u>NUnit</u>	Windows NT/2000	1061875
2	<u>NUnitAsp</u>	Windows NT/2000	72724
3	<u>NUnit Addin for Visual Studio.NET</u>	Windows	58588
4	<u>NUnitForms</u>	Windows NT/2000	46880
5	<u>csUnit</u>	csUnit has been tested using the Microsoft .NET framework 1.0 Service Pack 2 runtime on an Intel-compatible platform.	31483
6	<u>NCover</u>	All 32-bit MS Windows (95/98/NT/2000/XP)	14264
7	<u>VSNUnit</u>	All 32-bit MS Windows (95/98/NT/2000/XP)	8763
8	<u>dotUnit</u>	All 32-bit MS Windows (95/98/NT/2000/XP)	6230
9	<u>.NETUnit</u>	OS Independent (Written in an interpreted language)	5558
10	<u>ASPUnit</u>	Microsoft Internet Information Server 5.0 or 5.1	5197



Một số công cụ hỗ trợ kiểm thử chức năng

No	Name	Desc	Req	Download
1	<u>Software Testing Automation Framework (STAF)</u>		Windows, Linux, Solaris, AS/400, AIX, HP-UX, Irix	212018
2	<u>soapui</u>		Java 1.5	178985
3	<u>Linux Test Project</u>		Linux	103484
4	<u>jWebUnit</u>		OS Independent	56526
5	<u>Abbot Java GUI Test Framework</u>		TBC	56118
6	<u>Software Automation Framework Support</u>		All 32-bit MS Windows (95/98/NT/2000/XP)	43735
7	<u>Jameleon</u>		OS Independent, JDK 1.4 or higher	43507
8	<u>WebInject</u>		Windows, OS Independent, Linux	40891
9	<u>Marathon</u>		Java 1.3 or later	30328
10	<u>Solex</u>		Eclipse 2.1 or above	29591

Các công cụ kiểm thử thương mại



Vendor	Tool	Name of testing suite or companion tools
Compuware	TestPartner	QACenter Enterprise Edition+
Empirix	e-Tester	e-TEST suite
IBM Rational	Functional Tester	Test Manager, Manual Tester, Performance Tester
Mercury	QuickTest Professional	Quality Center
RadView	WebFT	TestView Suite
Seapine	QA Wizard	TestTrack Pro
Segue	SilkTest	SilkCentral, SilkPerformer

Các công cụ kiểm thử thương mại



Technical and nontechnical users

Technical users

IBM Rational
Functional Tester
Segue SilkTest
RadView WebFT

Nontechnical users

Mercury
QuickTest Pro
Compuware
TestPartner

Empirix e-Tester
Seapine QA Wizard

Tài liệu tham khảo



- Software Testing, A Craftsman's Approach, Paul C.Jorgensen
- Practical Software Testing, EleneBurnstein
- Slides: Software Testing ISEB Foundation Certificate Course
- Slides: Software Testing, Dr. Balla Katalin
- Slide: Equivalence Class Testing, Prof. Schlingloff & Dr. M Roggenbach
- Slide: Decision Table Based Testing, Neelam Gupta, The University of Arizona Tucson, Arizona, USA
- Object Oriented Testing, Ali Kamandi, Sharif University of Technology

Bài tập 1



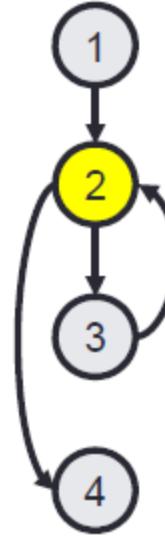
Xét đoạn code sau, yêu cầu: thiết kế các ca kiểm thử đạt bao phủ mức 4.

```
using namespace std;  
# include<iostream>  
main() {  
    int n;  
    cout<< "Nhập n"<<endl;  
    cin >>n;  
    for (n; n>0; n--) {  
        cout<< n<< ",";  
    }  
    cout<< "Kết thúc";  
    return 0;  
}
```

BT1



- Vẽ đồ thị dòng
- Tính độ phức tạp
 - $C=4-4+2=2$
- Xác định đường độc lập
 - 1-2-3-2
 - 1-2-4
- Xác định ca kiểm thử



TC	Đầu vào	Đầu ra among đợi
1	$N=10$	In ra “Nhập n” In ra “10,9,8,7,6,5,4,3,2,1, Kết thúc”
2	$N=0$	In ra “Nhập n” In ra “Kết thúc”

Bài tập 2



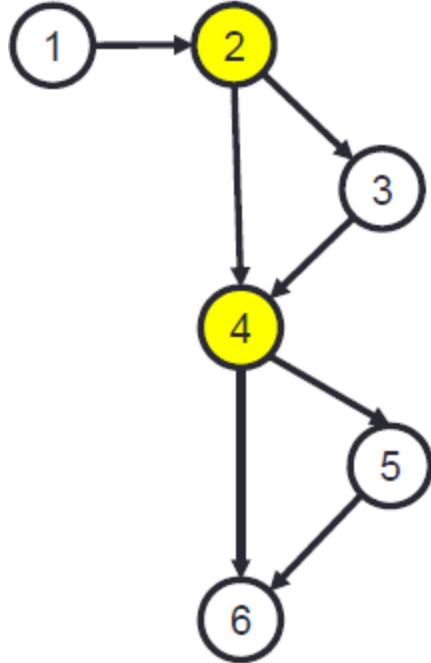
a. Xét đoạn code sau, yêu cầu: thiết kế các ca kiểm thử đạt bao phủ mức 2.

```
using namespace std;  
#include <iostream>  
main() {  
    int a,b,c,d,x,y;  
    cout<<"Nhập a, b, c, d, x, y"<<endl;  
    cin>>a>>b>>c>>d>>x>>y;  
    if (a>0&&b==1){x=x+1;}  
    if (c==3 || d<0) {y=0;}  
    cout<<"x = "<<x<<endl;  
    cout<<"y = "<<y<<endl;  
}
```

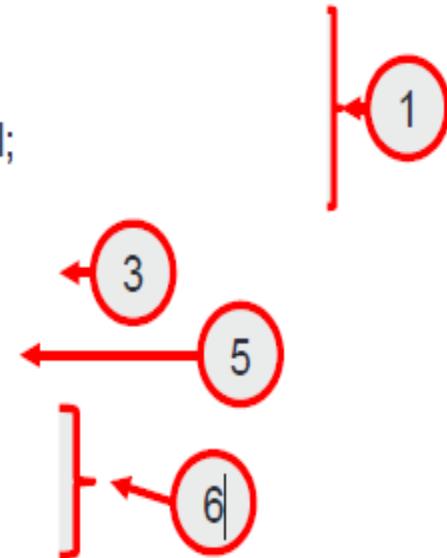
BT2



- Vẽ đồ thị dòng



```
using namespace std;  
#include <iostream>  
  
main() {  
    int a,b,c,d,x,y;  
  
    cout<<"Nhập a, b, c, d, x, y"<<endl;  
    cin>>a>>b>>c>>d>>x>>y;  
    if (a>0&&b==1){x=x+1;}  
    if (c==3 || d<0) {y=0;}  
    cout<<"x = "<<x<<endl;  
    cout<<"y = "<<y<<endl;  
}
```



- Độ phức tạp của chu trình C=3

BT2



- Xác định đường độc lập
 - 1-2-3-4-5-6
 - 1-2-4-5-6
 - 1-2-3-4-6
- Xác các ca kiểm thu

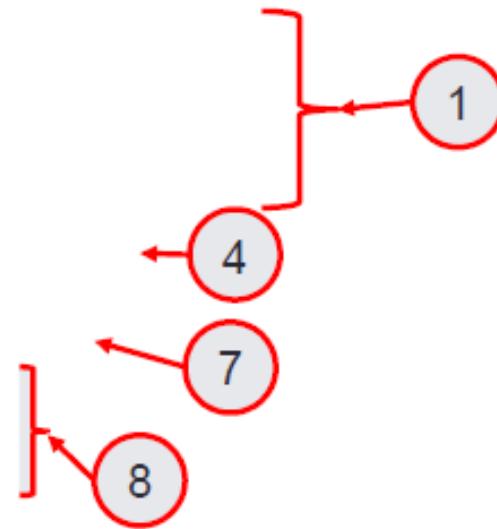
TC	Đầu vào	Đầu r among đợi
11	$a=5, b=1, c=3, d=2, x=7, y=7$	Nhap a, b, c, d, x, y $x=8, y=0$
22	$a=5, b=3, c=3, d=2, x=7, y=7$	Nhap a, b, c, d, x, y $x=7, y=0$
33	$a=5, b=1, c=0, d=2, x=7, y=7$	Nhap a, b, c, d, x, y $x=8, y=7$

BT2.b



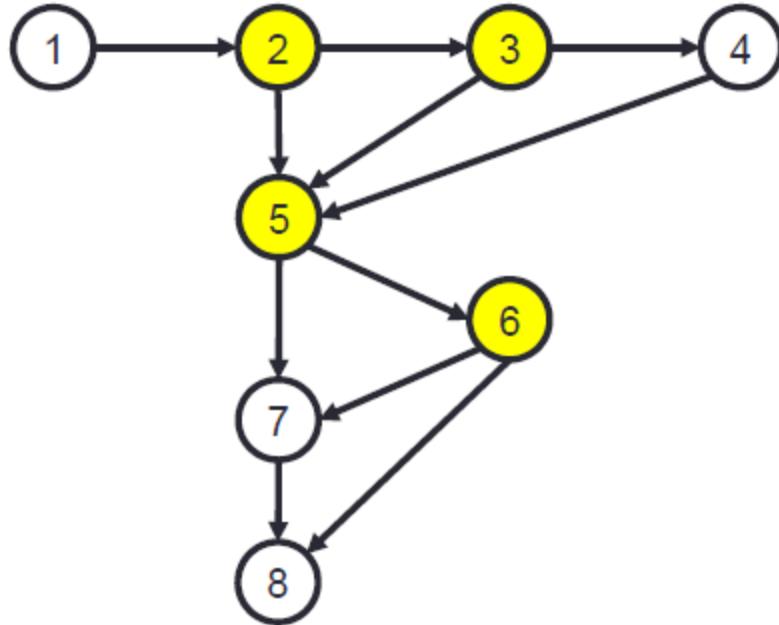
- Xét đoạn code yêu cầu thiết kế ca kiểm thử bao phủ mức 4

```
using namespace std;  
  
#include <iostream>  
  
main() {  
  
    int a,b,c,d,x,y;  
  
    cout<<"Nhập a, b, c, d, x, y"<<endl;  
    cin>>a>>b>>c>>d>>x>>y;  
  
    if (a>0&&b==1) {x=...}  
    if (c==3 || d<0) {y=...}  
  
    cout<<"x = "<<x<<endl;  
    cout<<"y = "<<y<<endl;  
  
}
```





- Vẽ đồ thị dòng
- Tính độ phức tạp của
 - Chu trình $C=11-8=2=5$
- Xác định các đường độc lập
 - 1-2-3-4-5-6-7-8
 - 2. 1-2-5-6-7-8
 - 3. 1-2-3-5-6-7-8
 - 4. 1-2-3-4-5-7-8
 - 5. 1-2-3-4-5-6-8



BT2



- Xác định các trường hợp kiểm thử

TC	Đầu vào	Đầu ra mong đợi
1	$a=5, b=1, c=4, d=-2, x=7, y=7$	Nhap a, b, c, d, x, y $x=8, y=0$
2	$a=-3, b=1, c=4, d=-2, x=7, y=7$	Nhap a, b, c, d, x, y $x=7, y=0$
3	$a=5, b=10, c=4, d=-2, x=7, y=7$	Nhap a, b, c, d, x, y $x=7, y=0$
4	$a=5, b=1, c=3, d=5, x=7, y=7$	Nhap a, b, c, d, x, y $x=8, y=0$
5	$a=5, b=1, c=4, d=5, x=7, y=7$	Nhap a, b, c, d, x, y $x=8, y=7$

Bài tập 3



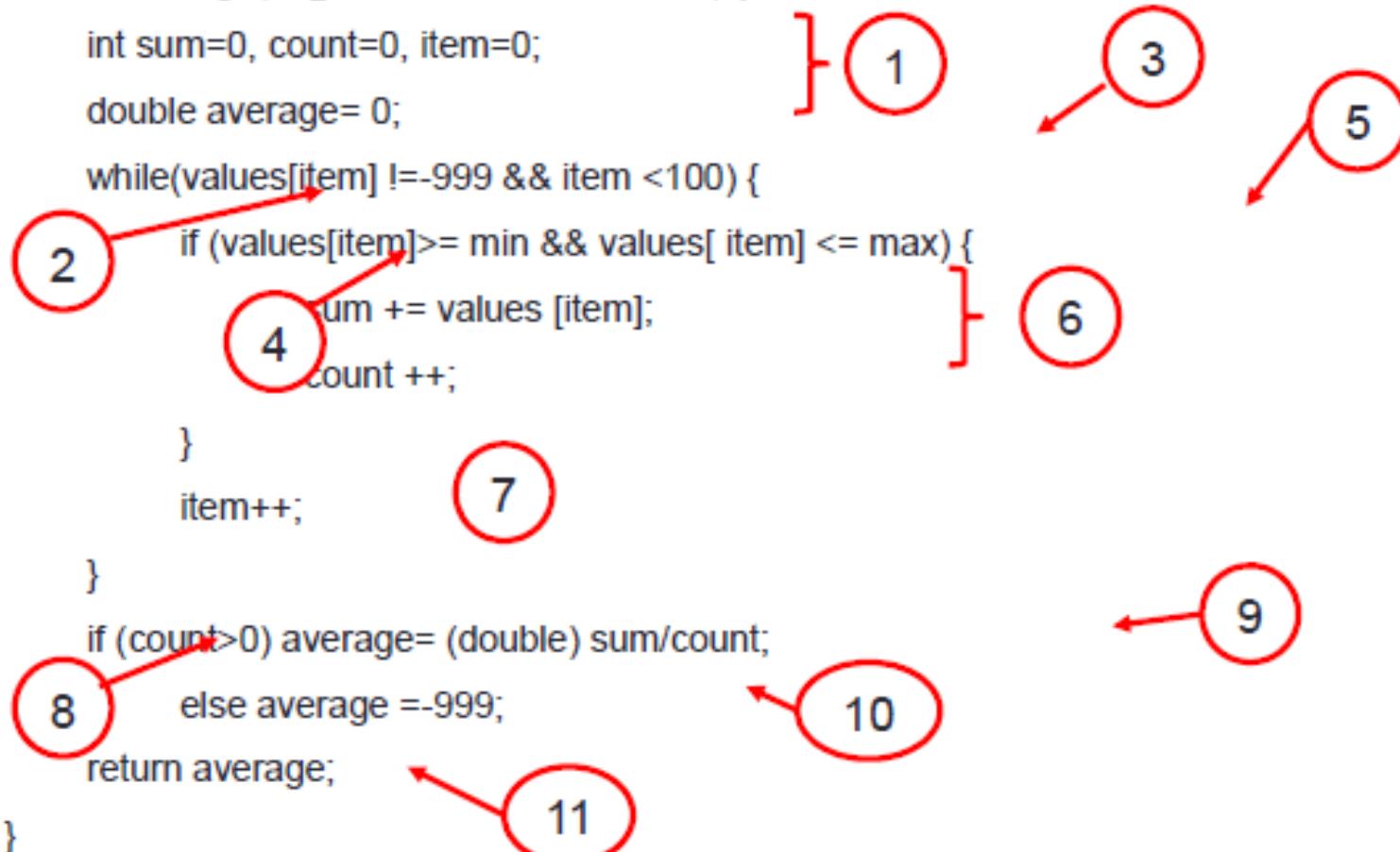
Thiết kế các ca kiểm thử thỏa mãn tiêu chuẩn phủ cấp 4

```
double average(int[] values, int min, int max) {  
    int sum=0, count=0, item=0;  
    double average= 0;  
    while(values[item] !=-999 && item <100) {  
        if (values[item]>= min && values[ item] <= max) {  
            sum += values [item];  
            count ++;  
        }  
        item++;  
    }  
    if (count>0) average= (double) sum/count;  
    else average =-999;  
    return average;  
}
```



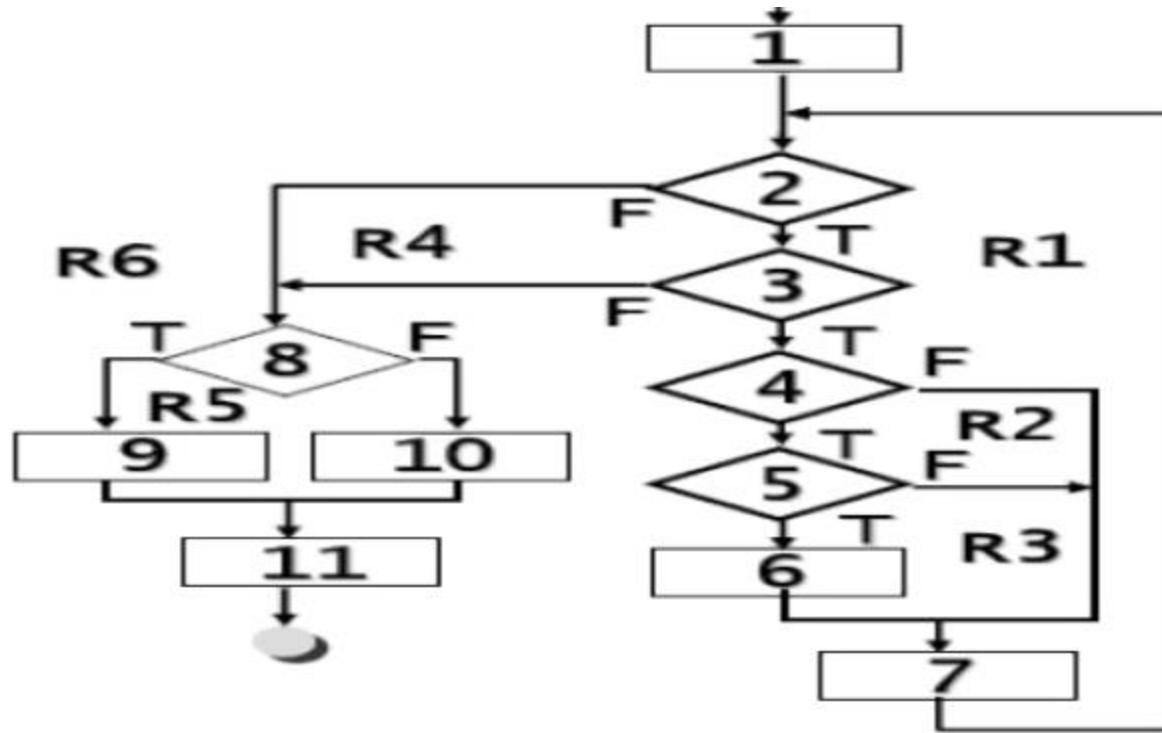
- Thiết kế ca kiểm thử thỏa mãn phủ cấp 4

```
double average(int[] values, int min, int max) {  
    int sum=0, count=0, item=0;  
    double average= 0;  
    while(values[item] !=-999 && item <100) {  
        if (values[item]>= min && values[ item] <= max) {  
            sum += values [item];  
            count ++;  
        }  
        item++;  
    }  
    if (count>0) average= (double) sum/count;  
    else average =-999;  
    return average;  
}
```



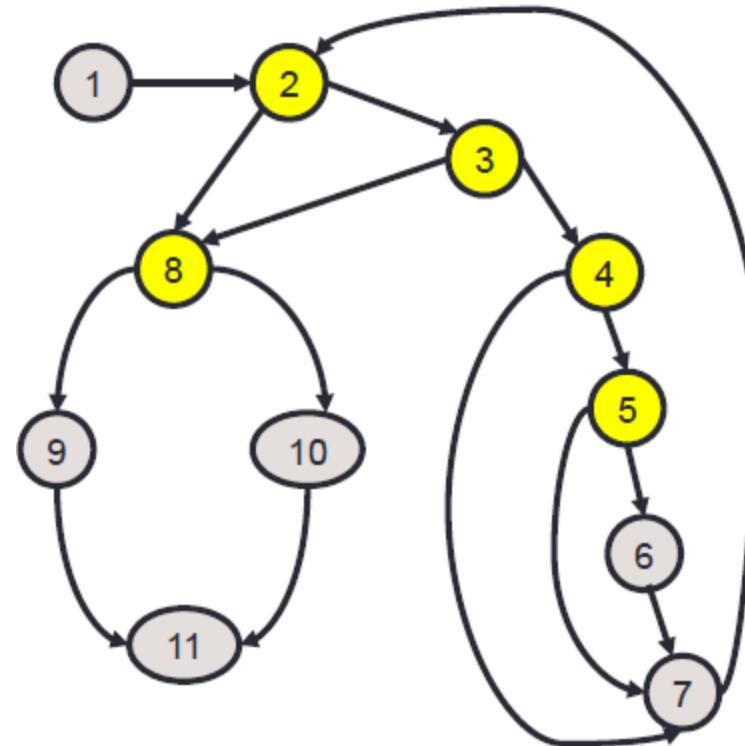


- Vẽ lưu đồ





- Vẽ sơ đồ dòng chảy
- Xác định độ phức tạp của chu trình
 - C=6
- Số đường path
 - 1. 1-2-8-9-11
 - 2. 1-2-8-10-11
 - 3. 1-2-3-8-9-11
 - 4. 1-2-3-4-7-2...
 - 5. 1-2-3-4-5-7-2...
 - 6. 1-2-3-4-5-6-7-2...



BT3



• Xác định ca kiểm thử

TC	Đầu vào	Đầu r among đợi
1	$M = \{1, 9, -999, 37\}$, min = 0, max = 10	Giá trị trung bình $\text{avg} = (1+9)/2$
2	$M = \{-999, 1, 7, 21\}$, min = 0, max = 10	$\text{avg} = -999$
3	$M = \{5, 6, 7, 120, \dots, 100\}$ (>100 phần tử) min = 0, max = 100	$\text{avg} = $ giá trị trung bình của 100 phần tử đầu tiên thỏa m.n điều kiện max, min
4	$M = \{7, 8, 18, 20, -999\}$ min = 10, max = 100	$\text{avg} = (18+20)/2$
6	$M = \{7, 80, 9, 20, 8, -999, 45\}$ min = 0, max = 10	$\text{avg} = (7+8+9)/3$
7	$M = \{7, 8, 9, \dots, 100\}$ ($M[i] <> -999$ và min $\leq M[i] \leq$ max với mọi $i \leq 100$) min = 0, max = 100	$\text{avg} = $ giá trị trung bình của 100 phần tử hợp lệ

Bài tập 4



- Thiết kế các ca kiểm thử thỏa mãn tiêu chuẩn phủ cấp 4

```
function goodstring(var count: integer): boolean;
```

```
var ch: char;
```

```
begin
```

```
    goodstring:= false;
```

```
    count:=0;
```

```
    read(ch);
```

```
    if ch='a' then
```



```
        while (ch='b') or (ch='c') do
```

```
            begin
```

```
                count:= count+1;
```

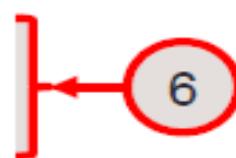
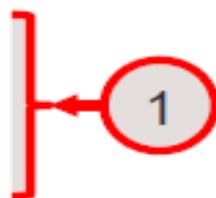
```
                read(ch);
```

```
            end;
```

```
            if ch='x' then goodstring= true;
```

```
        end;
```

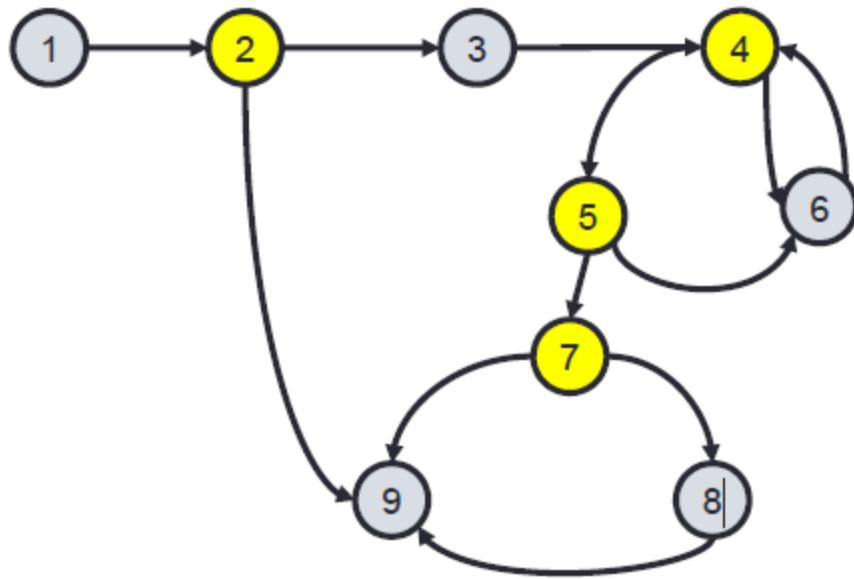
```
    end;
```



BT4



- Vẽ đồ thị
- Tính độ phức tạp
 - $C=12 \cdot 9 + 2 = 5$
- Xác định các đường độc lập
 - 1-2-9
 - 1-2-3-4-64
 - 1-2-3-4-5-6-4
 - 1-2-3-4-5-7-8-9
 - 1-2-3-4-5-7-9



BT 4



- Xác định các ca kiểm thử

TC	Đầu vào	Đầu r among đợi
1	Nhập “d”	Goodstring= false Count=0
2	Nhập “abbbx”	Goodstring= true Count=3
3	Nhập “ accccd”	Goodstring= false Count=4
4	Nhập “ax”	Goodstring= true Count=0
5	Nhập “ad”	Goodstring= false Count=0