



KIỂM THỬ PHẦN MỀM (5)

Lê Thị Mỹ Hạnh

Khoa Công nghệ Thông tin

Trường Đại học Bách khoa

Đại học Đà Nẵng



Nội dung

- Giới thiệu về kiểm thử
- Kiểm thử trong tiến trình phát triển
- Kiểm thử hộp đen
- Kiểm thử hộp trắng



Lỗi phần mềm

1. Tàu tham dò Sao Hỏa (Mars Climate Orbiter) bị rơi vào tháng 9 năm 1999 vì một "sai lầm ngớ ngẩn": sai đơn vị trong một chương trình.
2. Việc bắn hạ chiếc Airbus 320 của USS Vincennes năm 1988 được cho là do đầu ra khó hiểu và gây hiểu lầm được hiển thị bởi phần mềm theo dõi.
3. Cái chết do thử nghiệm không đầy đủ của phần mềm Dịch vụ xe cứu thương London.
4. Một số trường hợp tử vong trong giai đoạn 1985-7 của bệnh nhân ung thư là do điều trị quá liều phóng xạ do điều kiện tương tranh giữa các tác vụ đồng thời trong phần mềm Therac-25.
5. Lỗi trong phần mềm y tế đã gây ra cái chết. Chi tiết trong B.W. Boehm, "Software and its Impact: A Quantitative Assessment," *Datamation*, 19 (5), 48-59 (1973).
6. Một chiếc Airbus A320 gặp sự cố tại một triển lãm hàng không.
7. Một chiếc Airbus Industrie A300 của China Airlines gặp sự cố vào ngày 26 tháng 4 năm 1994 đã giết chết 264. Khuyến nghị bao gồm sửa đổi phần mềm.

List of 107 software failures that should have been caught by testing
<http://www.cs.tau.ac.il/~nachumd/verify/horror.html>

Lỗi phần mềm

1. Ngày 04/06/1996, chỉ 30 giây sau khi được khởi động, tên lửa Ariane 5 đã nổ tung trên bầu trời, giống như một trận bắn pháo hoa. Sự cố này là do lỗi mô phỏng hệ thống tương tự với phiên bản Ariane 4 cũ, biến 64-bit có số thập phân đã được chuyển thành biến 16-bit mà không có số thập phân. Vì sự khác biệt về kích cỡ nên đã gây ra hàng loạt lỗi trong bộ nhớ, ảnh hưởng đến máy tính trên tàu, làm tê liệt hệ thống và phát nổ, tiêu tốn 370 triệu USD chỉ do một phần mềm.



Tên lửa đẩy Ariane 5 nổ tung sau khi phóng 30s



Lỗi phần mềm

- Điều nhảy Gangnam Style làm YouTube điều đứng
 - Khi YouTube được phát triển, kiểu dữ liệu số nguyên có dấu 32 bit được sử dụng để lưu trữ số lượt xem của video.
 - không ai nghĩ rằng một video có thể đạt tới lượng view lớn hơn giới hạn tối đa của số nguyên có dấu 32 bit (khoảng 2,14 tỷ).
 - Gangnam Style đã phá kỷ lục này!
 - Sau khi vượt quá giới hạn, bộ đếm lượt truy cập hiển thị giá trị số âm!?
 - Các kỹ sư của Google đã sửa lỗi này bằng cách sử dụng kiểu số nguyên có dấu 64 bit để lưu trữ số lượt view, nâng con số tối đa lên khoảng 4,3 tỷ. Có lẽ một thời gian dài dài nữa giới hạn này mới bị xô đổ.

Top 10 bugs thốn nhất trong lịch sử ngành phần mềm

<https://laptrinhx.com/top-10-bugs-thon-nhat-trong-lich-su-nganh-phan-mem-1124661189/>



Kiểm thử là gì?

- IEEE: Kiểm thử là tiến trình vận hành hệ thống hoặc thành phần dưới những điều kiện xác định, quan sát hoặc ghi nhận kết quả và đưa ra đánh giá về hệ thống hoặc thành phần đó
- Myers: Kiểm thử là tiến trình thực thi chương trình với mục đích tìm thấy lỗi (The art of software testing)



Mục đích của kiểm thử

- Làm lộ ra một cách có hệ thống những lớp lỗi khác nhau (với thời gian và công sức tối thiểu)
 - Đảm bảo các thành phần của ứng dụng là ăn khớp với nhau, vận hành như mong đợi và phù hợp với các chuẩn thiết kế.
-
1. *Kiểm thử không chứng minh tính đúng của chương trình*
 - *Không thể kiểm thử triệt để*
 2. *Kiểm thử chỉ mang tính chất phát hiện ra được rằng tại thời điểm đó chỉ phát hiện ra bấy nhiêu lỗi thôi.*



Ai sẽ là người kiểm thử

- Lập trình viên
 - Kiểm tra code
- Nhóm kiểm thử độc lập
 - Kiểm tra tính đúng đắn của chương trình
- Người sử dụng
 - Thoả mãn yêu cầu người dùng



Kiểm thử là gì ?

- Kiểm thử \neq Gỡ rối (debug)
 - Kiểm thử
 - nhằm phát hiện lỗi
 - Gỡ rối
 - Xác định bản chất lỗi và định vị lỗi trong chương trình
 - tiến hành sửa lỗi

Các khái niệm

- Một sai sót (error) là một sự nhầm lẫn hay một sự hiểu sai trong quá trình phát triển phần mềm của người phát triển
- Một lỗi (fault, defect) xuất hiện trong phần mềm như là kết quả của một sai sót
- Một hỏng hóc (failure) là kết quả của một lỗi xuất hiện làm cho chương trình không hoạt động được hay hoạt động nhưng cho kết quả không như mong đợi

sai sót



lỗi



hỏng hóc



Các khái niệm: Ví dụ

- Sai sót (Error)
 - Một chỉ số vòng lặp bị lỗi
 - Lỗi biến con trỏ
- Lỗi (Fault) – kết quả của sai sót
 - Vòng lặp thực hiện quá nhiều lần
 - Tham số được lập trình lấy giá trị ở địa chỉ sai
- Hỏng hóc (Failure) – kết quả việc lỗi được “thực thi”
 - Tràn bộ nhớ
 - Giá trị sai



Các khái niệm

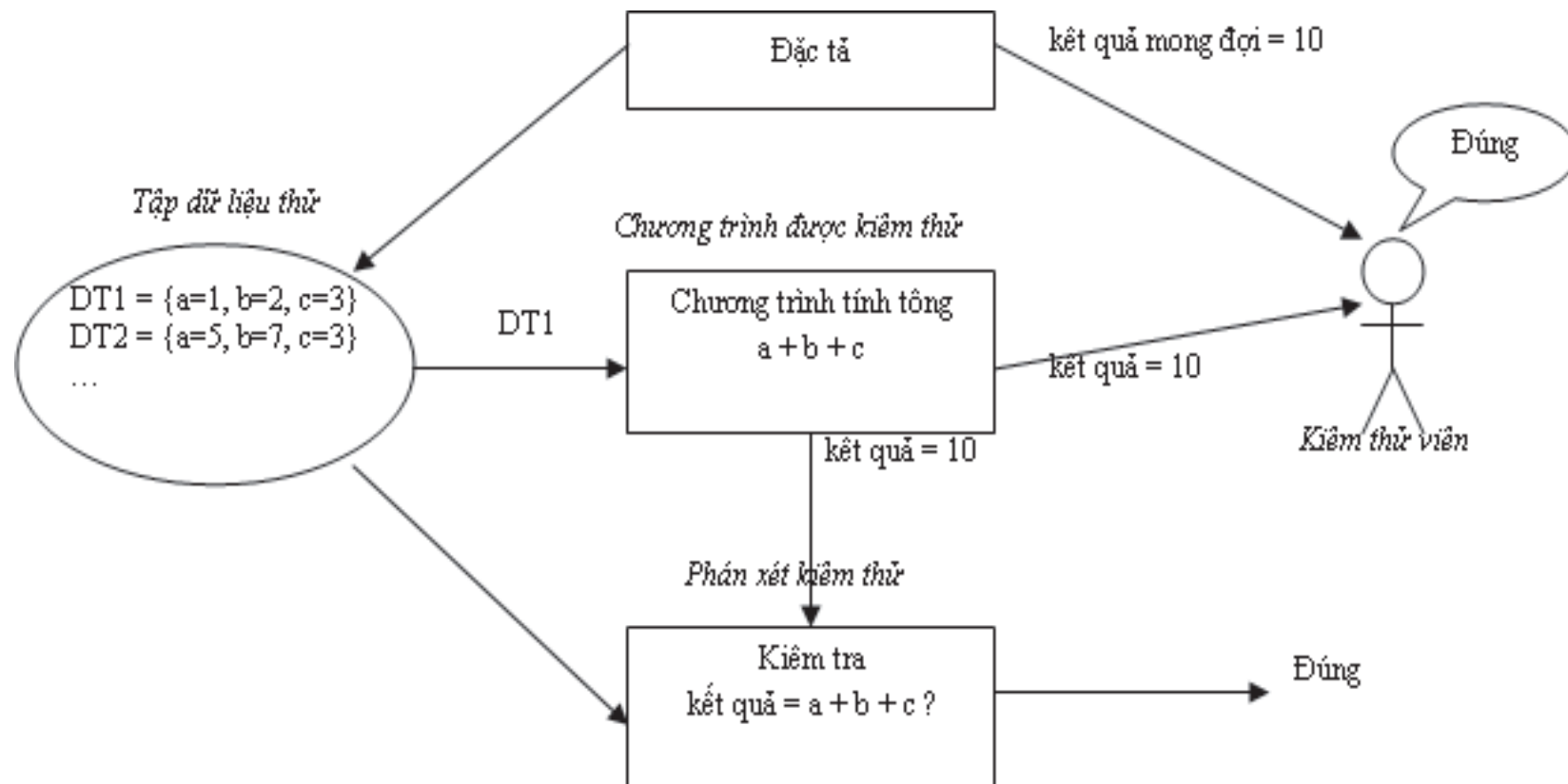
- Dữ liệu thử (test data)
 - dữ liệu vào cần cung cấp cho phần mềm trong khi thực thi
- Kịch bản kiểm thử (test scenario)
 - các bước thực hiện khi kiểm thử
- Phán xét kiểm thử (test oracle)
 - đánh giá kết quả của kiểm thử
 - tự động: chương trình
 - thủ công: con người



Các khái niệm

- Kiểm thử viên (tester)
 - người thực hiện kiểm thử
- Ca kiểm thử (test case)
 - tập dữ liệu thử
 - điều kiện thực thi
 - kết quả mong đợi

Các khái niệm

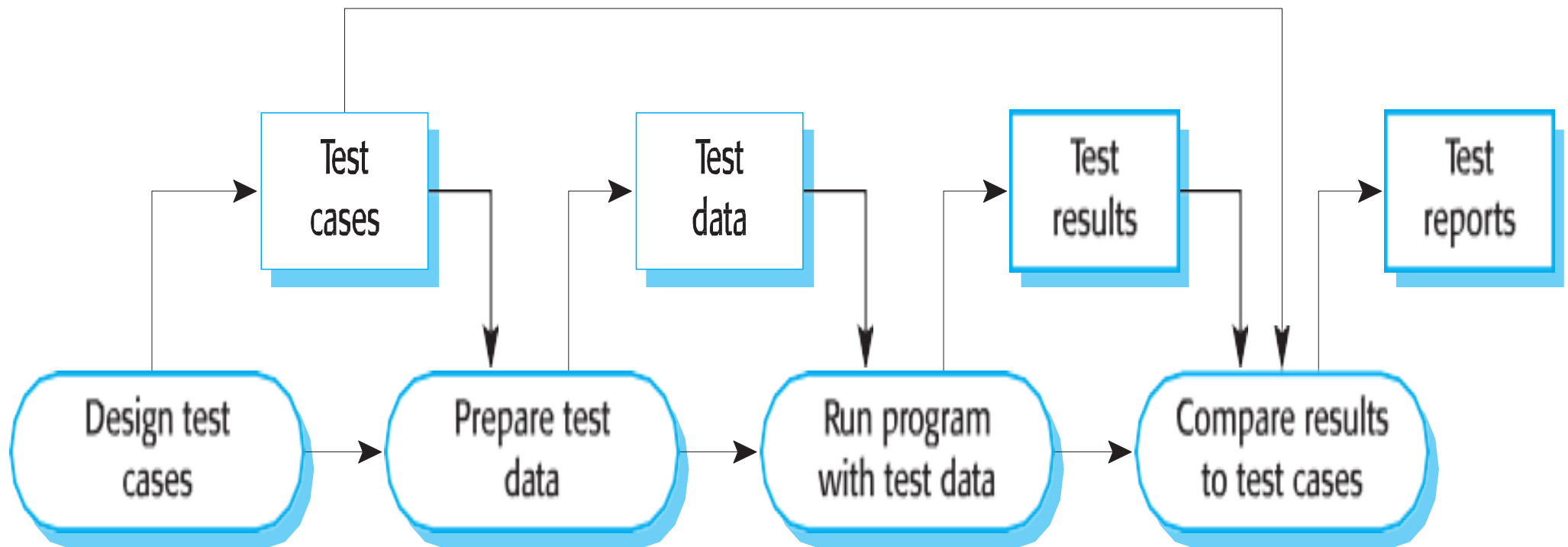




Tiến trình kiểm thử

- Kiểm thử thường bao gồm các bước
 - thiết kế các ca kiểm thử
 - bước tạo dữ liệu thử
 - kiểm thử với tất cả các dữ liệu vào là cần thiết
 - không thể kiểm thử “vét cạn”
 - chọn tập các dữ liệu thử đại diện từ miền dữ liệu vào
 - dựa trên các tiêu chuẩn chọn dữ liệu thử
 - bước thực thi chương trình trên dữ liệu thử
 - cung cấp dữ liệu thử
 - thực thi
 - ghi nhận kết quả
 - bước quan sát kết quả kiểm thử
 - Thực hiện trong khi hoặc sau khi thực thi
 - so sánh kết quả nhận được và kết quả mong đợi

Tiến trình kiểm thử





Khó khăn của kiểm thử

- Liên quan đến tiến trình phát triển
 - gồm nhiều giai đoạn phát triển
 - cái ra của một giai đoạn là cái vào của giai đoạn khác
 - mất mát thông tin
- Về mặt con người
 - thiếu đào tạo
 - ít chú trọng vai trò kiểm thử
- Về mặt kỹ thuật
 - không tồn tại thuật toán tổng quát có thể chứng minh sự đúng đắn hoàn toàn của bất kỳ một chương trình nào



Tại sao kiểm thử

- Hợp thức hóa (validation)
 - chỉ ra rằng sản phẩm đáp ứng được yêu cầu người sử dụng
- Xác minh (verification)
 - chỉ ra rằng sản phẩm thỏa mãn đặc tả yêu cầu
- Phân biệt hợp thức hóa và xác minh
 - “Verification: Are we building the product right ?”
 - “Validation: Are we building the right product ?”



Kiểm thử trong tiến trình phát triển

- Các kỹ thuật kiểm thử
 - kỹ thuật kiểm thử tĩnh (static testing)
 - kỹ thuật kiểm thử động (dynamic testing)
 - kiểm thử hộp đen (black-box testing)
 - kỹ thuật kiểm thử chức năng (functional testing)
 - kiểm thử hộp trắng (white-box testing)
 - kỹ thuật kiểm thử cấu trúc (structural testing)
- Các hoạt động kiểm thử/chiến lược kiểm thử
 - kiểm thử đơn vị (unit testing)
 - kiểm thử tích hợp (integration testing)
 - kiểm thử hợp thức hóa (validation testing)
 - kiểm thử hồi quy (regression testing)



Kiểm thử trong tiến trình phát triển

- Kiểm thử đơn vị (unit testing)
 - kiểm thử mỗi đơn vị phần mềm (mô-đun)
 - sử dụng kỹ thuật kiểm thử hộp đen
 - dữ liệu thử được tạo ra dựa trên tài liệu thiết kế
 - có thể sử dụng cả kiểm thử hộp trắng và kiểm thử tĩnh
 - phần mềm yêu cầu chất lượng cao
 - thường được thực hiện trên phần cứng phát triển phần mềm



Kiểm thử trong tiến trình phát triển

- Kiểm thử tích hợp (integration testing)
 - sau khi đã thực hiện kiểm thử đơn vị
 - ghép nối các đơn vị/thành phần phần mềm
 - kiểm thử sự ghép nối, trao đổi dữ liệu giữa các đơn vị/thành phần
 - sử dụng kỹ thuật kiểm thử hộp đen
 - một số trường hợp, sử dụng kỹ thuật kiểm thử hộp trắng
 - chi phí cao, khó khăn
 - dữ liệu thử được tạo ra dựa trên thiết kế tổng thể

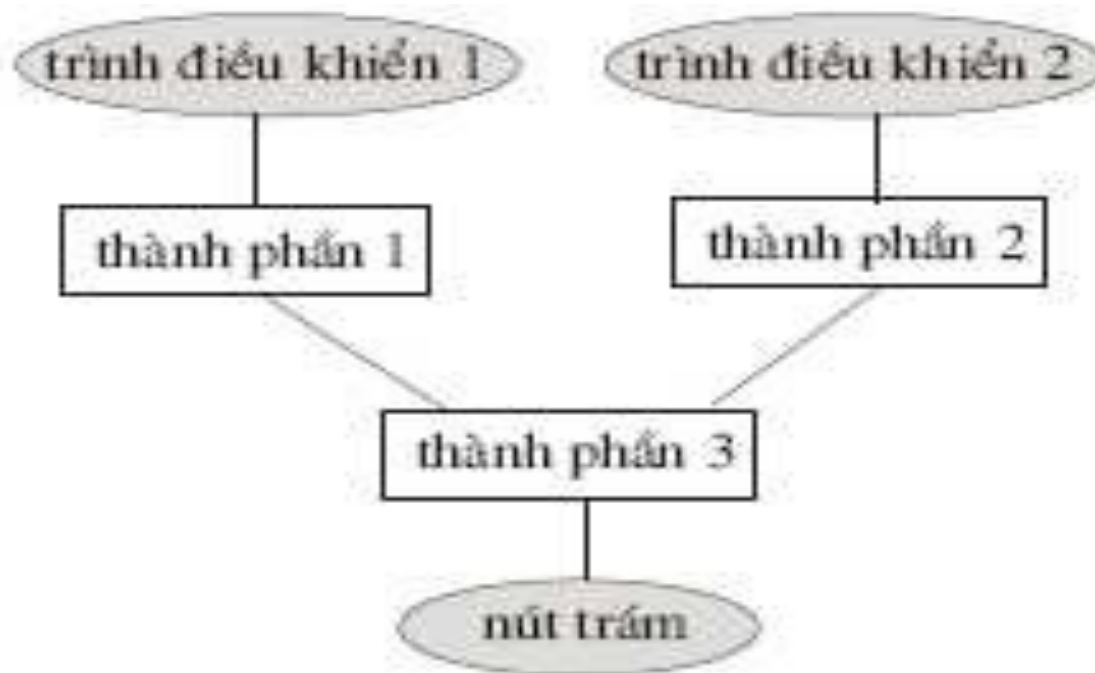


Kiểm thử trong tiến trình phát triển

- Kiểm thử tích hợp (2)
 - cần xây dựng thêm
 - nút trám (stub): các thành phần khác mô phỏng các thành phần phần mềm chưa được tích hợp
 - trình điều khiển (driver): các thành phần tạo ra các dữ liệu vào cho một vài các thành phần phần mềm trong tập hợp đang được kiểm thử

Kiểm thử trong tiến trình phát triển

- Kiểm thử tích hợp (3)





Kiểm thử trong tiến trình phát triển

- Kiểm thử tích hợp (4)

- chiến lược từ trên xuống (top-down)

- kiểm thử tích hợp các thành phần chính trước, sau đó thêm vào các thành phần nữa gọi trực tiếp bởi các thành phần vừa kiểm thử
 - cho phép xác định sớm các lỗi về kiến trúc
 - các bộ dữ liệu thử có thể được tái sử dụng cho các bước tiếp theo
 - tuy nhiên chiến lược này đòi hỏi phải xây dựng nhiều nút trám

- chiến lược từ dưới lên (bottom-up)

- kiểm thử các thành phần không gọi các thành phần khác, sau
 - nó thêm vào các thành phần gọi các thành phần vừa kiểm thử
 - ít sử dụng các nút trám
 - nhưng lại xác định lỗi trễ hơn



Kiểm thử trong tiến trình phát triển

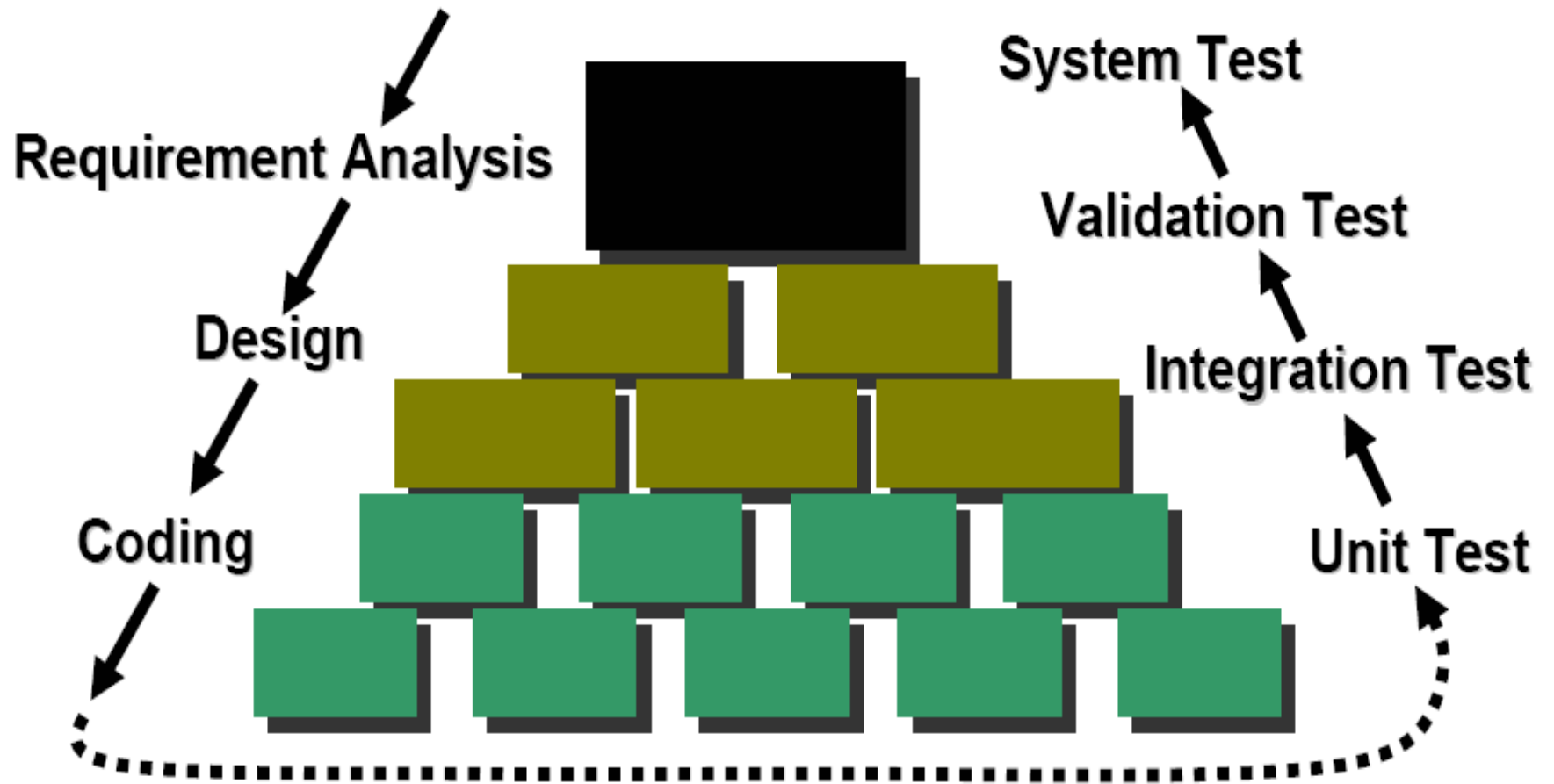
- Kiểm thử hợp thức hóa (validation testing)
 - còn gọi là kiểm thử hệ thống (system testing)
 - thực hiện sau khi kiểm thử tích hợp kết thúc
 - chứng minh phần mềm thực hiện đúng mong đợi của người sử dụng
 - dựa vào yêu cầu người sử dụng
 - chỉ sử dụng kỹ thuật kiểm thử hộp đen
 - nên thực hiện trong môi trường mà phần mềm sẽ được sử dụng



Kiểm thử trong tiến trình phát triển

- Kiểm thử hồi quy (regression testing)
 - phần mềm sau khi đưa vào sử dụng, có thể có các chỉnh sửa
 - có thể phát sinh lỗi mới
 - cần kiểm thử lại: kiểm thử hồi quy
 - thường tái sử dụng các bộ dữ liệu thử đã sử dụng trong các giai đoạn trước

Kiểm thử trong tiến trình phát triển





Các kỹ thuật kiểm thử

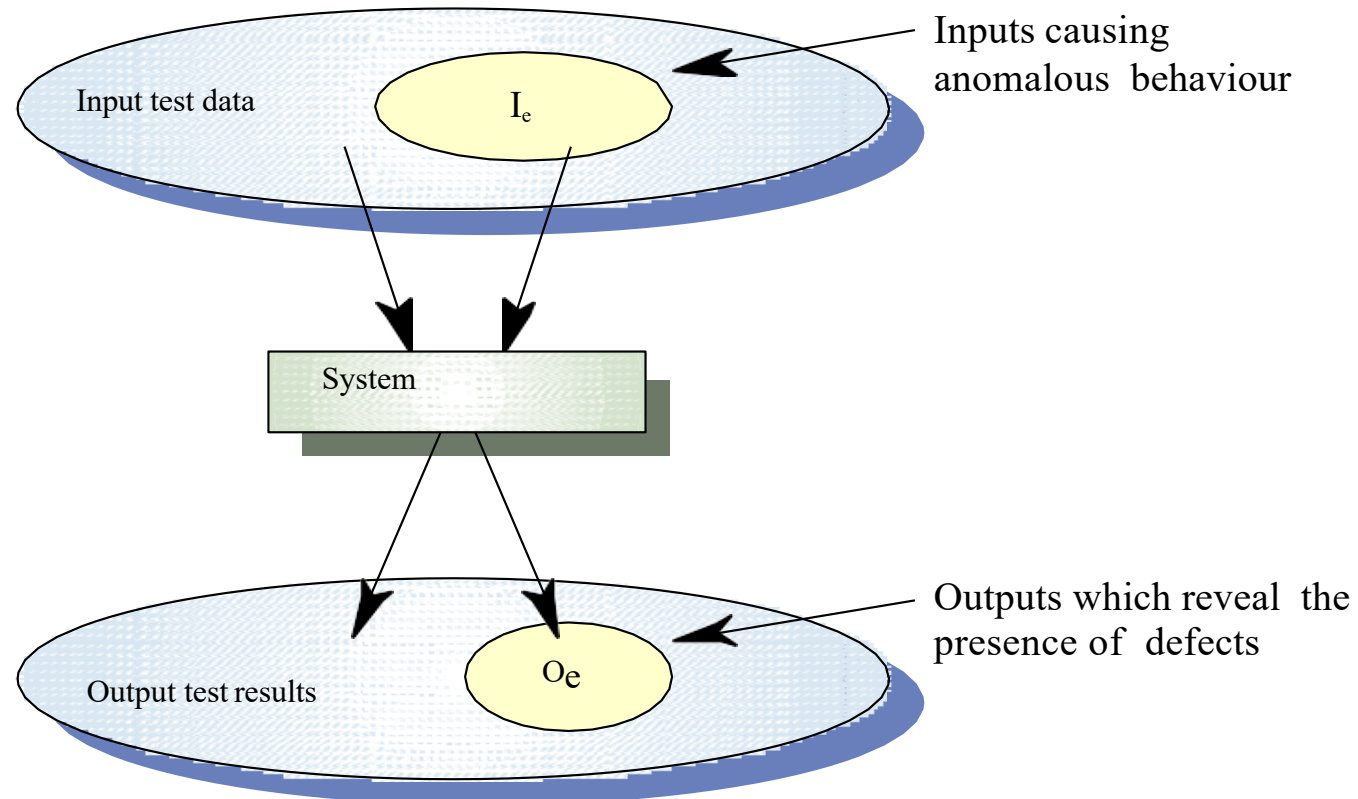
- kỹ thuật kiểm thử tĩnh (static testing)
 - không thực thi chương trình
- kỹ thuật kiểm thử động (dynamic testing)
 - kiểm thử hộp đen (black-box testing)
 - kỹ thuật kiểm thử chức năng (functional testing)
 - kiểm thử hộp trắng (white-box testing)
 - kỹ thuật kiểm thử cấu trúc (structural testing)



Kiểm thử tĩnh

- Thanh tra mã nguồn (code inspection)
- Chứng minh hình thức
- Thực thi hình thức (symbolic execution)
- Đánh giá độ phức tạp
 - McCabe
 - Nejmech

Kiểm thử hộp đen





Kiểm thử hộp đen

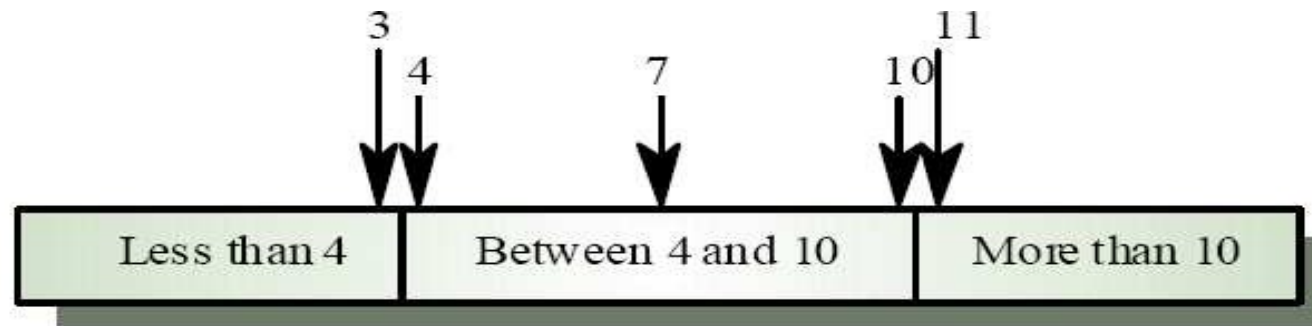
- Chỉ cần dựa vào đặc tả chương trình
 - Xây dựng dữ liệu thử trước khi mã hóa/lập trình
- Thường phát hiện các lỗi đặc tả yêu cầu, thiết kế
 - Dễ dàng thực hiện
 - Chi phí thấp
- Các kỹ thuật kiểm thử hộp đen
 - Kiểm thử giá trị biên (boundary value analysis)
 - Kiểm thử lớp tương đương (equivalence class testing)
 - Kiểm thử ngẫu nhiên (random testing)
 - Đồ thị nhân-quả (cause-effect graph)
 - Kiểm thử cú pháp



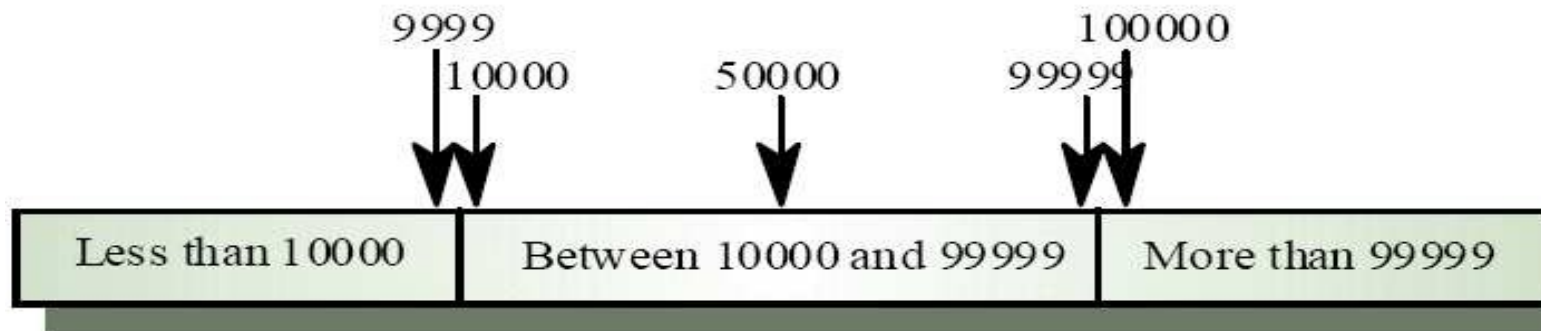
Kiểm thử giá trị biên

- Cơ sở
 - lỗi thường xuất hiện gần các giá trị biên của miền dữ liệu
- Tập trung phân tích các giá trị biên của miền dữ liệu
 - để xây dựng dữ liệu kiểm thử
- Nguyên tắc: kiểm thử các dữ liệu vào gồm
 - giá trị nhỏ nhất
 - giá trị gần kề lớn hơn giá trị nhỏ nhất
 - giá trị bình thường
 - giá trị gần kề nhỏ hơn giá trị lớn nhất
 - giá trị lớn nhất

Kiểm thử giá trị biên



Number of input values



Input values



Kiểm thử giá trị biên

- Nguyên tắc chọn dữ liệu thử
 - Nếu dữ liệu vào thuộc một khoảng, chọn
 - 2 giá trị biên
 - 4 giá trị = giá trị biên \pm sai số nhỏ nhất
- Nếu giá trị vào thuộc danh sách các giá trị, chọn
 - phần tử thứ nhất, phần tử thứ hai, phần tử kế cuối và phần tử cuối
- Nếu dữ liệu vào là điều kiện ràng buộc số giá trị, chọn
 - số giá trị tối thiểu, số giá trị tối đa và một số các số giá trị không hợp lệ
- Tự vận dụng khả năng và thực tế để chọn các giá trị biên cần kiểm thử



Kiểm thử giá trị biên

- Ví dụ (1)

- Chương trình nhận vào ba số thực, kiểm tra ba số thực có là độ dài ba cạnh một tam giác. Nếu là độ dài ba cạnh của một tam giác, thì kiểm tra xem đó là tam giác thường, cân, đều cũng như kiểm tra đó là tam giác nhọn, vuông hay tù.



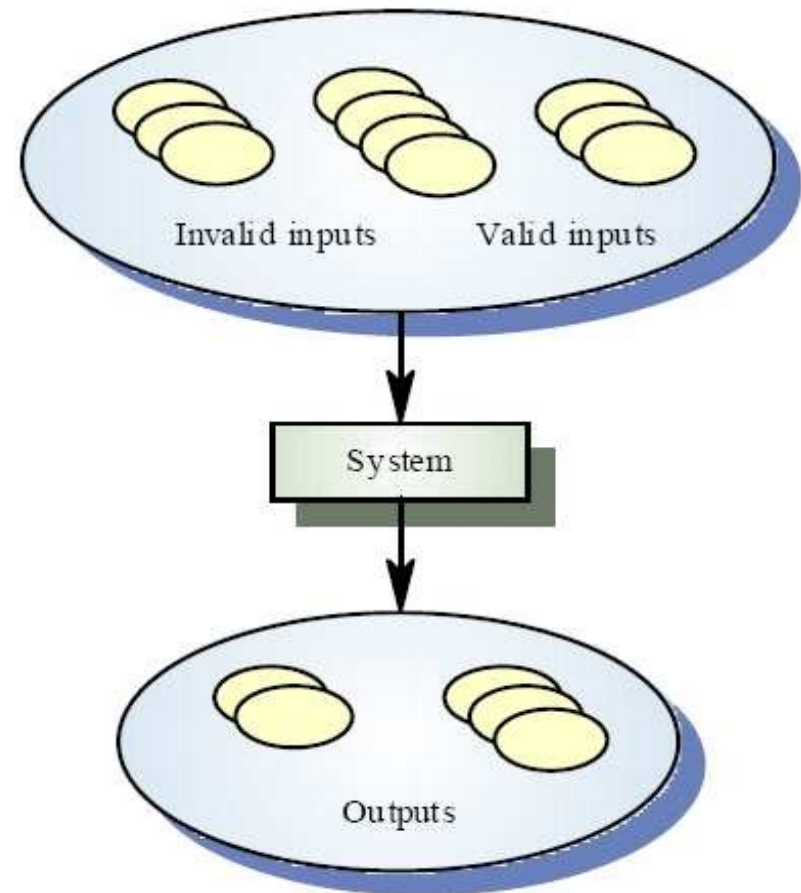
Kiểm thử giá trị biên

- Ví dụ (2): Dữ liệu thử

1, 1, 2	Không là tam giác
0, 0, 0	Chỉ một điểm
4, 0, 3	Một cạnh bằng không
1, 2, 3.00001	Gần là một tam giác
0.001, 0.001, 0.001	Tam giác rất nhỏ
99999, 99999, 99999	Tam giác rất lớn
3.00001, 3, 3	Tam giác gần đều
2.99999, 3, 4	Tam giác gần cân
3, 4, 5.00001	Tam giác gần vuông
3, 4, 5, 6	Bốn giá trị
3	Chỉ một giá trị
	Dữ liệu vào rỗng
-3, -3, 5	Giá trị âm

Kiểm thử lớp tương đương

- Ý tưởng
 - phân hoạch miền dữ liệu vào thành các lớp các dữ liệu có quan hệ với nhau (lớp tương đương)
 - mỗi lớp dùng để kiểm thử một chức năng, gọi là lớp tương đương





Kiểm thử lớp tương đương

- Ba bước
 - đối với mỗi dữ liệu vào, xác định các lớp tương đương từ miền dữ liệu vào
 - chọn dữ liệu đại diện cho mỗi lớp tương đương
 - kết hợp các dữ liệu thử bởi tích đề-các để tạo ra bộ dữ liệu kiểm thử



Kiểm thử lớp tương đương

- Nguyên tắc phân hoạch các lớp tương đương
 - Nếu dữ liệu vào thuộc một khoảng, xây dựng
 - 1 lớp các giá trị lớn hơn
 - 1 lớp các giá trị nhỏ hơn
 - n lớp các giá trị hợp lệ
- Nếu dữ liệu là tập hợp các giá trị, xây dựng
 - 1 lớp với tập rỗng
 - 1 lớp quá nhiều các giá trị
 - n lớp hợp lệ
- Nếu dữ liệu vào là điều kiện ràng buộc, xây dựng
 - 1 lớp với ràng buộc được thỏa mãn
 - 1 lớp với ràng buộc không được thỏa mãn



Kiểm thử lớp tương đương

- Ví dụ
 - Bài toán tam giác

	Nhọn	Vuông	Tù
Thường	6,5,3	5,6,10	3,4,5
Cân	6,1,6	7,4,4	$\sqrt{2}, 2, \sqrt{2}$
đều	4,4,4	không thể	không thể
Không là tam giác		-1,2,8	



Bài tập

- Kiểm thử giá trị biên

- Viết một chương trình thống kê phân tích một tệp chứa tên và điểm của sinh viên trong một năm học. Tệp này chứa nhiều nhất 100 trường. Mỗi trường chứa tên của mỗi sinh viên (20 ký tự), giới tính (1 ký tự) và điểm của 5 môn học (từ 0 đến 10). Mục đích chương trình:
 - tính điểm trung bình mỗi sinh viên
 - tính điểm trung bình chung (theo giới tính và theo môn học)
 - tính số sinh viên lên lớp (điểm trung bình trên 5)
- Xây dựng dữ liệu thử cho chương trình trên bởi kiểm thử giá trị biên



Bài tập

- Kiểm thử lớp tương đương
 - Viết chương trình dịch, trong đó có câu lệnh FOR, đặc tả câu lệnh FOR như sau: “Lệnh FOR chỉ chấp nhận một tham số duy nhất là biến đếm. Tên biến không được sử dụng quá hai ký tự khác rỗng. Sau ký hiệu = là cận dưới và cận trên của biến đếm. Các cận trên và cận dưới là các số nguyên dương và được đặt giữa từ khóa TO”.
 - Xây dựng dữ liệu thử để kiểm thử câu lệnh FOR theo kỹ thuật kiểm thử lớp tương đương



Kiểm thử hộp trắng

- Dựa vào mã nguồn/cấu trúc chương trình
 - Xây dựng dữ liệu thử sau khi mã hóa/lập trình
- Thường phát hiện các lỗi lập trình
- Khó thực hiện
- Chi phí cao



Các kỹ thuật kiểm thử hộp trắng

- Kiểm thử dựa trên đồ thị luồng điều khiển
- Kiểm thử dựa trên đồ thị luồng dữ liệu
- Kiểm thử đột biến (mutation testing)



Đồ thị luồng điều khiển

- Đồ thị luồng điều khiển (Control Flow Graph -ĐTLĐK) là đồ thị có hướng, biểu diễn một chương trình
 - Đỉnh: biểu diễn lệnh tuần tự hay khối lệnh
 - cung: biểu diễn các rẽ nhánh
 - Một đỉnh vào và một đỉnh ra được thêm vào để biểu diễn điểm vào và ra của chương trình
- Lộ trình (path) trong ĐTLĐK xuất phát từ đỉnh vào đi qua các đỉnh và cung trong đồ thị và kết thúc ở đỉnh ra

Đồ thị luồng điều khiển

- Ví dụ 1

```
if x <= 0 then
    x := -x
else x := 1 - x;
if x = -1 then
    x = 1
else x := x + 1;
writeln(x);
```

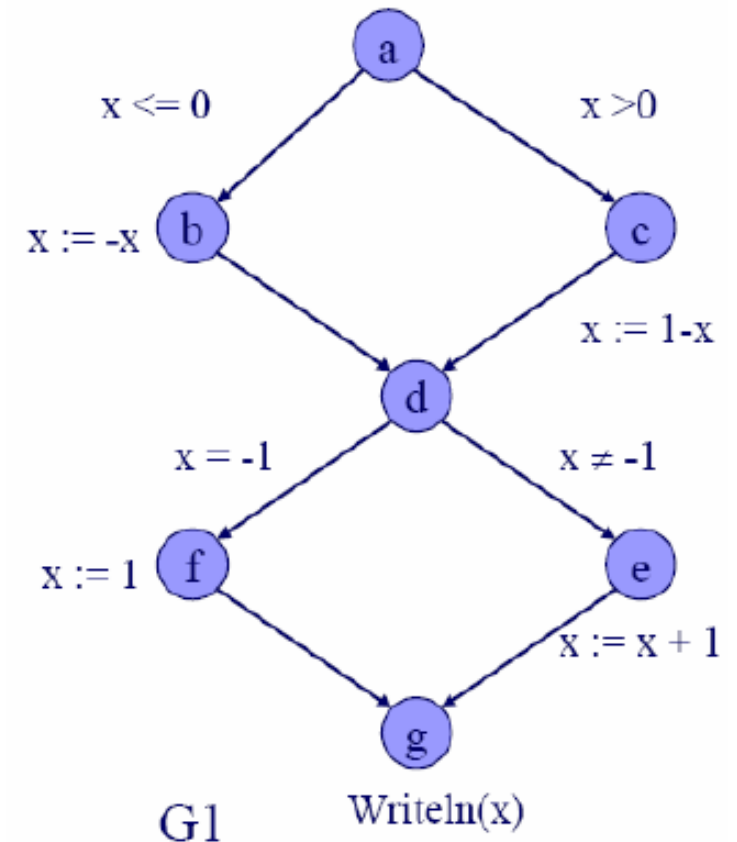
- Có 4 lộ trình

- [a, b, d, f, g]
- [a, b, d, e, g]
- [a, c, d, f, g]
- [a, c, d, e, g]

- Đồ thị G1 có thể biểu diễn dạng biểu thức chính quy:

$$G1 = abdfg + abdeg + acdfg + acdeg$$

- Hay đơn giản: $G1 = a(bdf + bde + cdf + cde)g$
 $G1 = a(b + c)d(e + f)g$

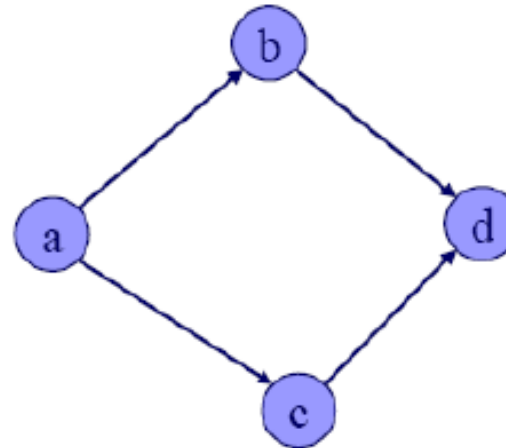


Đồ thị luồng điều khiển

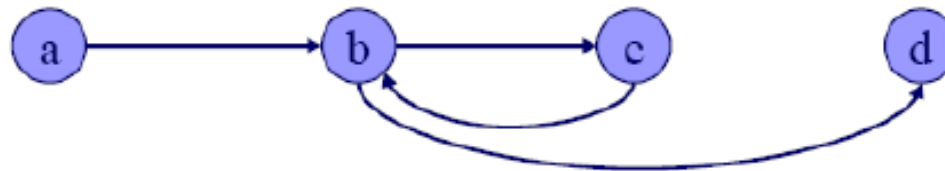
Biểu diễn các cấu trúc



Cấu trúc tuần tự: ab



Cấu trúc rẽ nhánh: $b(a + d)c$

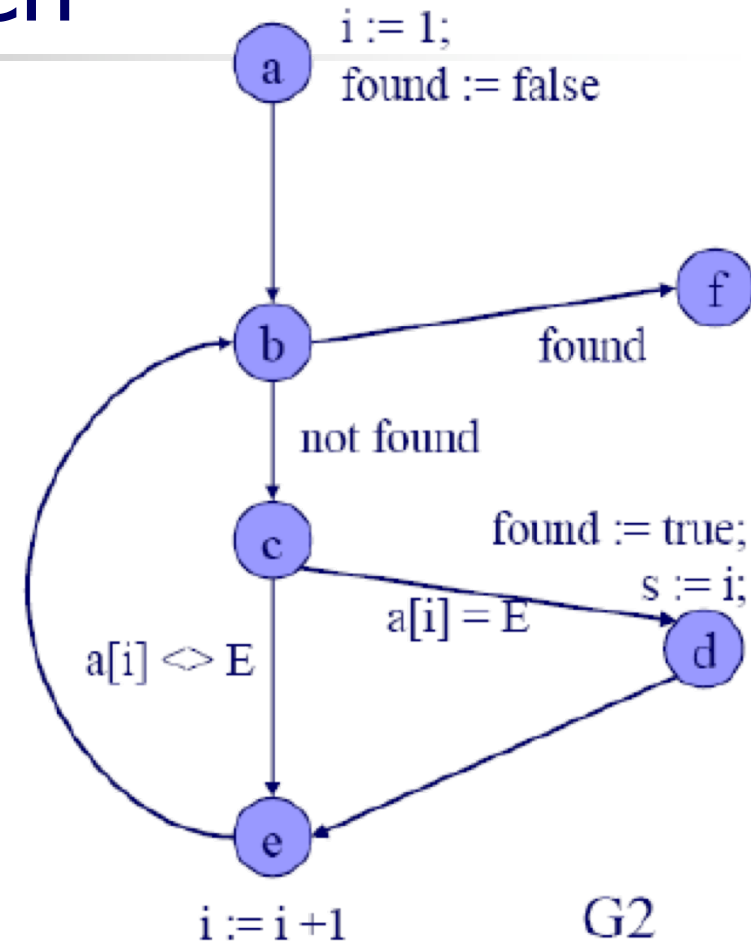


Cấu trúc lặp: $ab(cb)^*d$

Đồ thị luồng điều khiển

■ Ví dụ 2:

```
i := 1;  
found := false;  
while (not found) do  
  begin  
    if (a[i] = E) then  
      begin  
        found := true;  
        s := i;  
      end;  
    i := i + 1;  
  end;
```



$$G2 = ab(c(e + d)eb)^*f$$



Đồ thị luồng điều khiển

- Bài tập 2
 - Vẽ đồ thị luồng điều khiển
 - Xây dựng biểu thức chính quy biểu diễn đồ thị

```
read(i);  
s := 0;  
while(i <= 3) do begin  
    if a[i] > 0 then s := s + a[i];  i := i + 1;  
end
```



Kiểm thử dựa trên ĐTLĐK

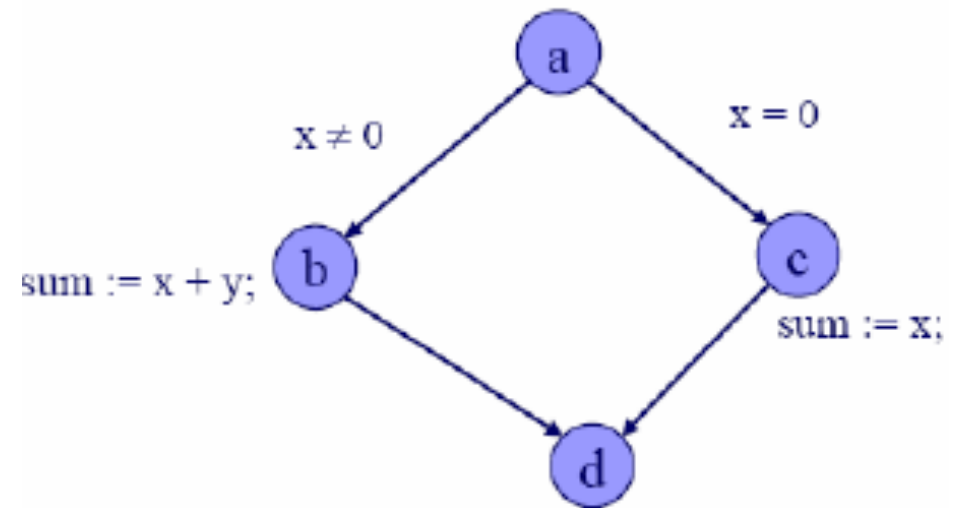
- Các tiêu chuẩn bao phủ
 - Phủ tất cả các đỉnh/lệnh
 - Phủ tất cả các cung
 - Phủ tất cả các quyết định
 - Phủ tất cả các đường đi

Kiểm thử dựa trên ĐTLĐK

Phủ tất cả các đỉnh/lệnh

- Cho phép phủ tất cả các đỉnh/lệnh
 - Mỗi lệnh được thực thi ít nhất một lần
 - Tiêu chuẩn tối thiểu

```
function sum(x, y : integer) : integer;  
begin  
    if (x = 0) then  
        sum := x  
    else  
        sum := x + y  
    end;  
end;
```



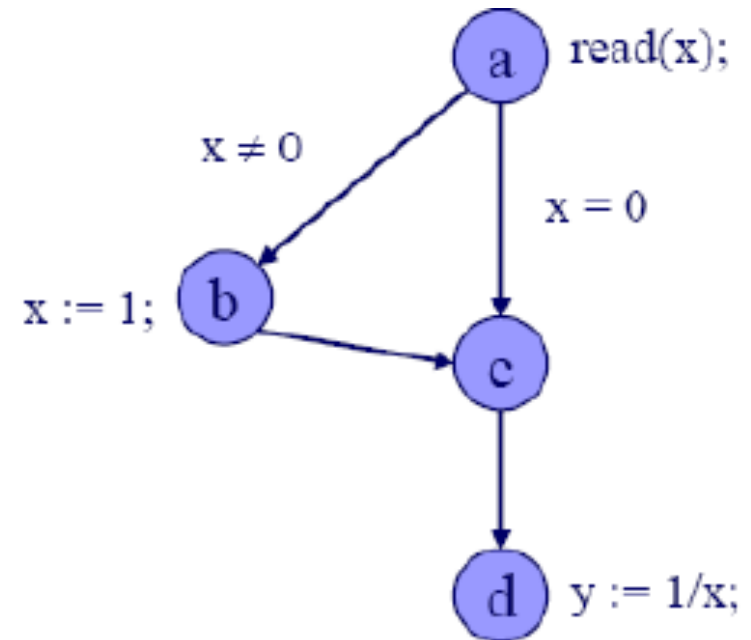
Khi thực thi lộ trình acd sẽ phát hiện lỗi

Kiểm thử dựa trên ĐTLĐK

Phủ tất cả các đỉnh/lệnh

Hạn chế của tiêu chuẩn

```
read(x);  
...  
if (x <> 0) then x := 1;  
...  
y := 1/x;
```



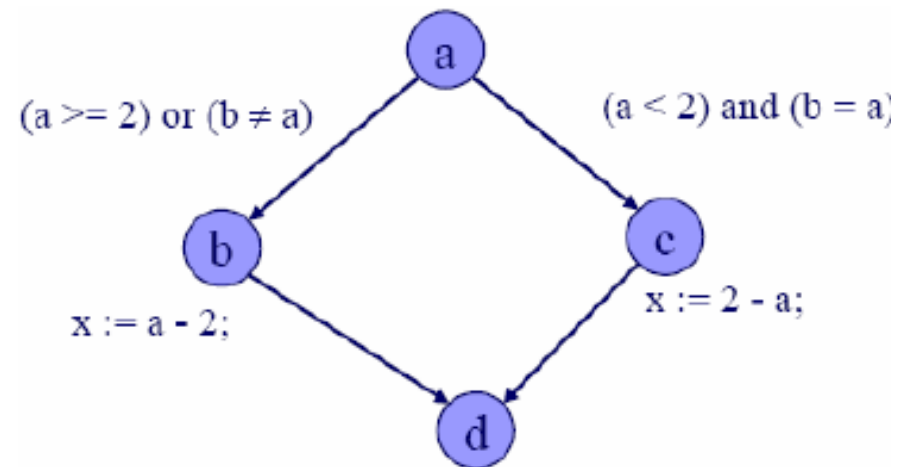
Phủ tất cả các đỉnh không phát hiện được lỗi

Kiểm thử dựa trên ĐTLĐK

Phủ tất cả các cung

- Phủ tất cả các cung ít nhất một lần
 - Phủ tất cả các giá trị đúng sai của một biểu thức lô-gíc
 - Phủ tất cả các cung kéo theo phủ tất cả các đỉnh

```
if ((a < 2) and (b = a)) then
    x := 2 - a
else
    x := a - 2
```



Dữ liệu thử DT1 = {a=b=1} và DT2 = {a=b=3} thỏa mãn phủ tất cả các cung, nhưng không phủ tất cả các quyết định,
Chẳng hạn DT3 = {a=3, b=2}



Kiểm thử dựa trên ĐTLĐK

Phủ tất cả các quyết định

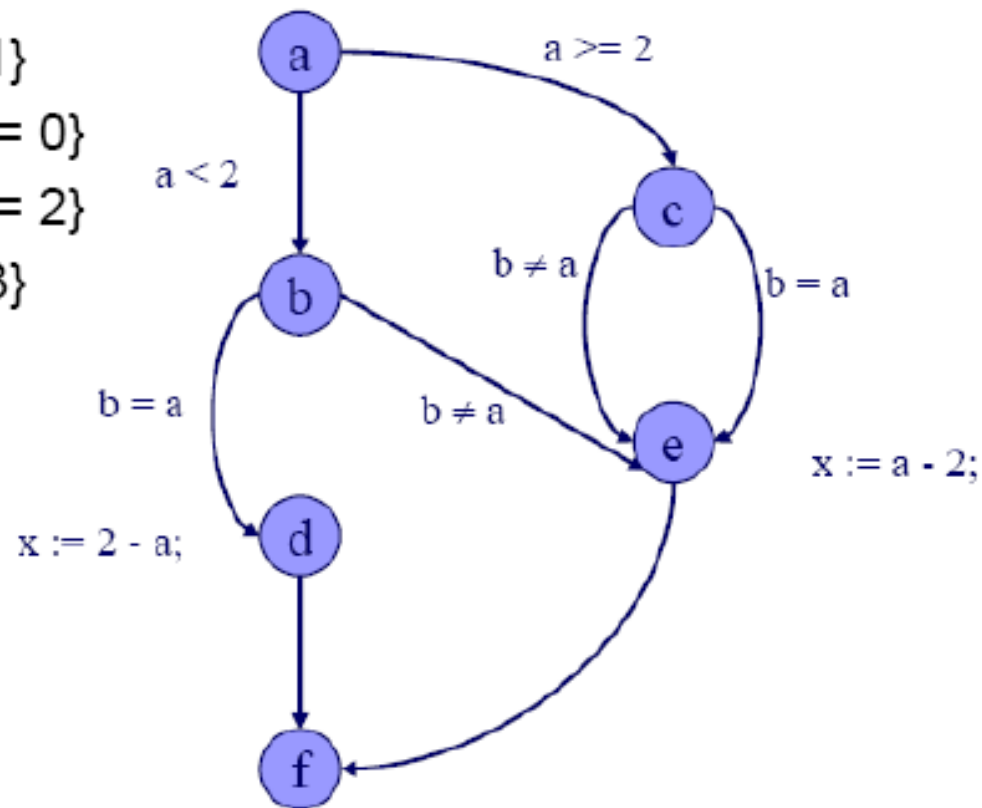
- Phủ tất cả các quyết định được thỏa mãn khi:
 - tiêu chuẩn phủ tất cả các cung được thỏa mãn và
 - Mọi biểu thức con của biểu thức điều kiện được thử với tất cả các giá trị có thể
- Nếu (a AND b)
 - $a = b = \text{true}$
 - $a = b = \text{false}$
 - $a = \text{true}, b = \text{false}$
 - $a = \text{false}, b = \text{true}$

Kiểm thử dựa trên ĐTLĐK

Phủ tất cả các quyết định

■ Dữ liệu thử

- $DT1 = \{a = b = 1\}$
- $DT2 = \{a = 1, b = 0\}$
- $DT3 = \{a = 3, b = 2\}$
- $DT4 = \{a = b = 3\}$



Kiểm thử dựa trên ĐTLĐK

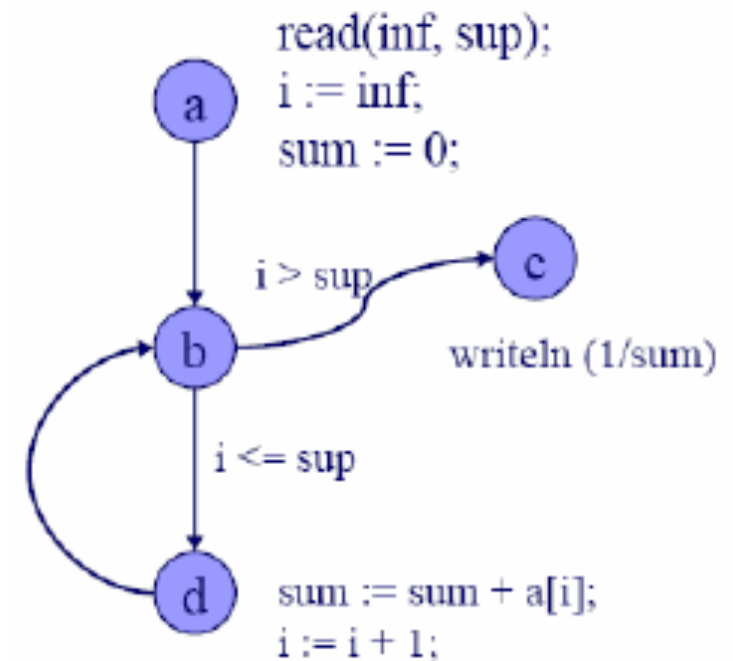
Phủ tất cả các quyết định

- Hạn chế

- Không phát hiện lỗi trong trường hợp không thực thi vòng lặp

```
read(inf, sup);  
i := inf;  
sum := 0;  
while(i <= sup) do  
  begin  
    sum := sum + a[i];  
    i := i + 1;  
  end;  
writeln(1/sum);
```

Dữ liệu thử DT1 = {a[1]=50, a[2]=60, a[3]=80, inf=1, sup=3}
phủ tất cả các cung/quyết định, nhưng không phát hiện lỗi





Kiểm thử dựa trên ĐTLĐK

Phủ tất cả các lộ trình

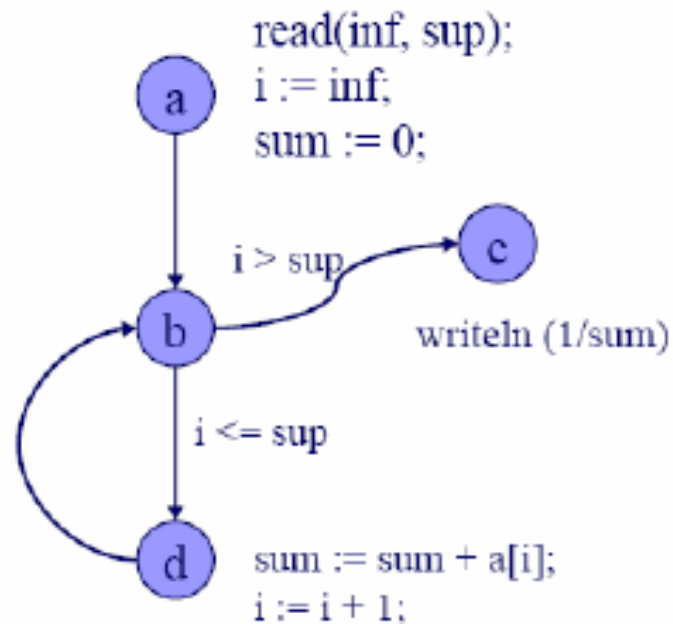
- Mỗi lộ trình phải được thực hiện ít nhất một lần
- Gặp khó khăn khi số lần lặp vô hạn
 - Chỉ thực hiện một số lần lặp nhất định
 - Hoặc chỉ thực hiện hai loại lộ trình
 - các lộ trình vượt qua vòng lặp nhưng không lặp
 - các lộ trình chỉ lặp n lần (chẳng hạn $n = 1$)

Kiểm thử dựa trên ĐTLĐK

Phủ tất cả các lộ trình

■ Dữ liệu thử

- DT1 = {a[1]=50, a[2]=60, a[3]=80, inf=1, sup=3}
- DT2 = {a[1]=50, a[2]=60, a[3]=80, inf=3, sup=2}





Kiểm thử dựa trên ĐT LĐK

- Xây dựng dữ liệu thử thỏa mãn các tiêu chuẩn
 - phủ tất cả các đỉnh
 - phủ tất cả các cung
 - phủ tất cả các lộ trình

```
if  $n \leq 0$  then  
     $n := 1 - n$   
end;  
if  $(n \bmod 2) = 0$  then  
     $n := n / 2$   
else  
     $n := 3 * n + 1$   
end ;  
write(n);a
```



Kiểm thử dựa trên ĐT LĐK

- Xây dựng dữ liệu thử thỏa mãn các tiêu chuẩn phủ tất cả các lộ trình

```
function goodstring(var count : integer) : boolean;  
var ch : char;  
begin  
  goodstring := false; count := 0; read(ch);  
  if ch = 'a' then  
    begin  
      read(ch)  
      while(ch = 'b') or (ch = 'c') do begin count := count + 1;  
        read(ch);  
      end;  
      if ch = 'x' then goodstring = true;  
    end;  
  end;  
end;
```



KIỂM THỬ ĐƯỜNG DẪN CƠ SỞ

- Cho phép người kiểm thử thực hiện các trường hợp kiểm thử để đảm bảo mọi câu lệnh trong chương trình được thực hiện ít nhất 1 lần
- Trong quá trình kiểm thử người kiểm thử cần thực hiện theo các bước sau:

B1: Vẽ đồ thị lưu trình

B2: Xác định độ phức tạp Cyclomat

Tìm C bằng 1 trong 3 cách sau:

+ $C = \text{Số lượng vòng}$

+ $C = \text{số cạnh} - \text{số đỉnh} + 2$

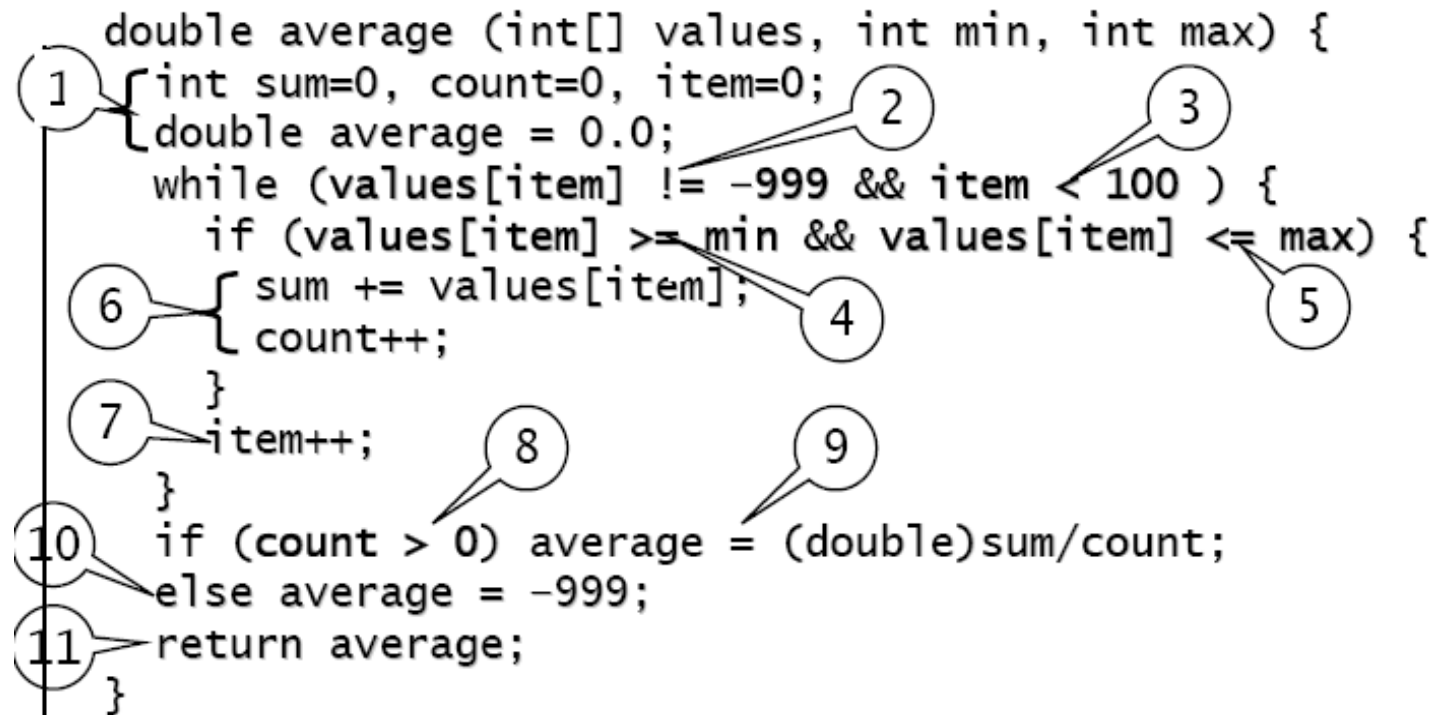
+ $C = \text{Số đỉnh điều kiện} + 1$

B3: Xác định 1 tập các đường dẫn cơ sở độc lập

B4: Thiết kế các trường hợp kiểm thử với mọi đường dẫn độc lập

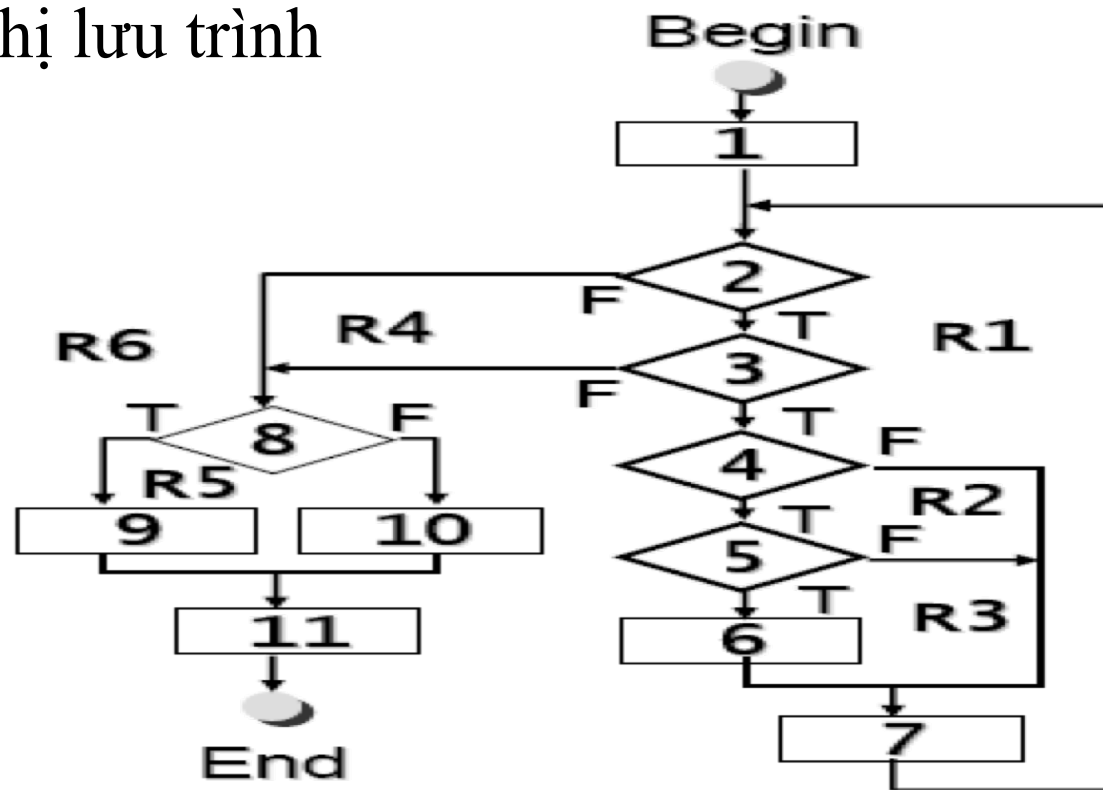
KIỂM THỬ ĐƯỜNG DẪN CƠ SỞ

■ Ví dụ:



KIỂM THỬ ĐƯỜNG DẪN CƠ SỞ

- B1: Vẽ đồ thị lưu trình





KIỂM THỬ ĐƯỜNG DẪN CƠ SỞ

B2: Xác định độ phức tạp Cyclomat(C)

+ $C = \text{Số lượng vòng} = 6$

+ Hoặc $C = \text{số cạnh} - \text{số đỉnh} + 2 = 15 - 11 + 2 = 6$

+ Hoặc $C = \text{Số đỉnh điều kiện} + 1 = 5 + 1 = 6$



KIỂM THỬ ĐƯỜNG DẪN CƠ SỞ

B3: Xác định các đường dẫn độc lập

- Dựa vào độ phức tạp đưa ra đường dẫn độc lập
- Có 6 đường dẫn độc lập:

Path1: $1 \rightarrow 2 \rightarrow 8 \rightarrow \mathbf{9} \rightarrow 11$

Path2: $1 \rightarrow 2 \rightarrow 8 \rightarrow \mathbf{10} \rightarrow 11$

Path3: $1 \rightarrow 2 \rightarrow \mathbf{3} \rightarrow 8 \rightarrow 9 \rightarrow 11$

Path4: $1 \rightarrow 2 \rightarrow 3 \rightarrow \mathbf{4} \rightarrow 7 \rightarrow 2 \dots$

Path5: $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow \mathbf{5} \rightarrow 7 \rightarrow 2 \dots$

Path6: $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow \mathbf{6} \rightarrow 7 \rightarrow 2 \dots$



KIỂM THỬ ĐƯỜNG DẪN CƠ SỞ

- B4: Thiết kế trường hợp kiểm thử cho từng đường dẫn:

Test Case1 cho Path1:

Đầu vào: $a = \{1, 7, -999, 37\}$, $\min=0$, $\max=100$

Đầu ra mong đợi: $\text{avg} = (1+7)/2$

Test Case2 cho Path2:

Đầu vào: $a = \{-999, 32, 34\}$, $\min=0$, $\max=100$

Đầu ra mong đợi: $\text{avg} = -999$

Test Case3 cho Path3:

Đầu vào: $a = \{7, 5, \dots, 50\}$ (>100 phần tử), $\min=0$, $\max=100$

Đầu ra mong đợi: avg = giá trị trung bình của 100 phần tử đầu tiên



KIỂM THỬ ĐƯỜNG DẪN CƠ SỞ

- Bài tập

- Xây dựng dữ liệu thử thoả mãn tiêu chuẩn phủ đường dẫn cơ sở cho các hàm sau:
 - Tìm ước số chung lớn nhất của 02 số tự nhiên a, b : USCLN(a, b)
 - Tìm kiếm có hay không giá trị x trong 01 mảng có thứ tự