Chương 6

Kiểm thử dòng điều khiển

Trong chương này, chúng ta sẽ tìm hiểu chi tiết về phương pháp kiểm thử dòng dữ liệu (control flow testing) nhằm phát hiện các lỗi tiềm ẩn bên trong chương trình/đơn vị chương trình cần kiểm thử. Các lỗi này thường khó phát hiện bởi các kỹ thuật kiểm thử hàm hay kiểm thử chức năng được trình bày trong chương 5. Để áp dụng phương pháp này, chúng ta cần phân tích mã nguồn và xây dựng các ca kiểm thử ứng với các dòng điều khiển của chương trình/đơn vị chương trình. Các độ đo hay tiêu chí kiểm thử cho phương pháp này cũng sẽ được giới thiệu.

6.1 Kiểm thử hộp trắng

Kiểm thử hộp trắng sử dụng các chiến lược cụ thể và sử dụng mã nguồn của chương trình/đơn vị phần mềm cần kiểm thử nhằm kiểm tra xem chương trình/đơn vị phần mềm có thực hiện đúng so với thiết kế và đặc tả hay không. Trong khi các phương pháp kiểm thử hộp đen hay kiểm thử hàm/chức năng chỉ cho phép phát hiện các lỗi/khiếm khuyết có thể quan sát được, kiểm thử hộp trắng cho phép phát hiện các lỗi/khiếm khuyết tiềm ẩn bên trong chương trình/đơn vị phần mềm. Các lỗi này thường khó phát hiện bởi các phương pháp kiểm thử hộp đen. Kiểm thử hộp đen và kiểm thử hộp trắng không thể thay thế cho nhau mà chúng cần được sử dụng kết hợp với nhau trong một quy trình kiểm thử thống nhất nhằm đảm bảo chất lượng phần mềm. Tuy nhiên, để áp dụng các phương pháp kiểm thử hộp trắng, người

kiểm thử không chỉ cần hiểu rõ giải thuật mà còn cần có các kỹ năng và kiến thức tốt về ngôn ngữ lập trình được dùng để phát triển phần mềm, nhằm hiểu rõ mã nguồn của chương trình/đơn vị phần mềm cần kiểm thử. Do vậy, việc áp dụng các phương pháp kiểm thử hộp trắng thường tốn thời gian và công sức nhất là khi chương trình/đơn vị phần mềm có kích thước lớn. Vì lý do này, các phương pháp kiểm thử hộp trắng chủ yếu được sử dụng cho kiểm thử đơn vị [D.95].

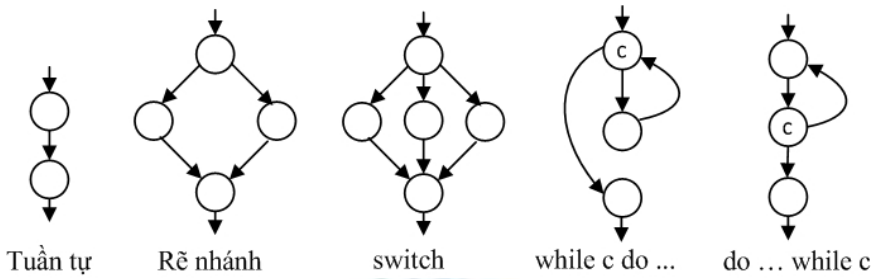
Hai phương pháp được sử dụng trong kiểm thử hộp trắng là kiểm thử dòng điều khiển (control flow testing) và kiểm thử dòng dữ liệu (data flow testing). Phương pháp kiểm thử dòng điều khiển tập trung kiểm thử tính đúng đắn của các giải thuật sử dụng trong các chương trình/đơn vị phần mềm. Phương pháp kiểm thử dòng dữ liệu tập trung kiểm thử tính đúng đắn của việc sử dụng các biến dữ liệu sử dụng trong chương trình/đơn vị phần mềm. Trong chương này, chúng ta sẽ tìm hiểu chi tiết về phương pháp kiểm thử dòng điều khiển. Phương pháp kiểm thử dòng dữ liệu sẽ được giới thiệu trong chương 7.

6.2 Đồ thị dòng điều khiển

Phương pháp kiểm thử dòng điều khiển dựa trên khái niệm đồ thị dòng điều khiển (control flow graph). Đồ thị này được xây dựng từ mã nguồn của chương trình/đơn vị chương trình. Đồ thị dòng điều khiển là một đồ thị có hướng gồm các đỉnh tương ứng với các câu lệnh/nhóm câu lệnh và các cạnh là các dòng điều khiển giữa các câu lệnh/nhóm câu lệnh. Nếu i và j là các đỉnh của đồ thị dòng điều khiển thì tồn tại một cạnh từ i đến j nếu lệnh tương ứng với j có thể được thực hiện ngay sau lệnh tương ứng với i.

Xây dựng một đồ thị dòng điều khiển từ một chương trình/đơn vị chương trình khá đơn giản. Hình 6.1 mô tả các thành phần cơ bản của đồ thị dòng điều khiển bao gồm điểm bắt đầu của đơn vị chương trình, khối xử lý chứa các câu lệnh khai báo hoặc tính toán, điểm quyết định ứng với các câu lệnh điều kiện trong các khối lệnh rẽ nhánh hoặc lặp, điểm nối ứng với các câu lệnh ngay sau các lệnh rẽ nhánh, và điểm kết thúc ứng với điểm kết thúc của đơn vị chương trình. Các cấu trúc điều khiển phổ biến của chương trình được mô tả trong Hình 6.2. Chúng ta sẽ sử dụng các thành phần cơ bản và các cấu trúc phổ biến này để dễ dàng xây dựng đồ thị dòng điều khiển cho

Hình 6.1: Các thành phần cơ bản của đồ thị chương trình.



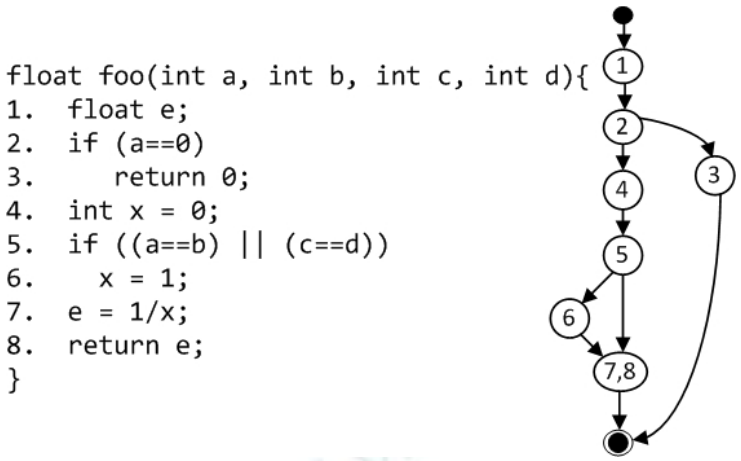
Hình 6.2: Các cấu trúc điều khiển phổ biến của chương trình.

mọi đơn vị chương trình viết bằng mọi ngôn ngữ lập trình.

Chúng ta thử xem cách dựng đồ thị dòng điều khiển cho đơn vị chương trình có mã nguồn bằng ngôn ngữ C như Hình 6.3. Chúng ta đánh số các dòng lệnh của đơn vị chương trình và lấy số này làm đỉnh của đồ thị. Điểm xuất phát của đơn vị chương trình ứng với câu lệnh khai báo hàm `foo`. Đỉnh 1 ứng với câu lệnh khai báo biến e. Các đỉnh 2 và 3 ứng với câu lệnh `if`. Đỉnh 4 ứng với câu lệnh khai báo biến x trong khi các đỉnh 5 và 6 ứng với câu lệnh `if`. Đỉnh 7,8 đại diện cho hai câu lệnh 7 và 8. Trong trường hợp này, chúng ta không tách riêng thành hai đỉnh vì đây là hai câu lệnh tuần tự nên chúng ta ghép chúng thành một đỉnh nhằm tối thiểu số đỉnh của đồ thị dòng điều khiển.

6.3 Các độ đo kiểm thử

Kiểm thử hàm (kiểm thử hộp đen) có hạn chế là chúng ta không biết có thừa hay thiếu các ca kiểm thử hay không so với chương trình cài đặt và thiếu thừa ở mức độ nào. Độ đo kiểm thử là một công cụ giúp ta đo mức độ bao phủ chương trình của một tập ca kiểm thử cho trước. Mức độ bao phủ của



Hình 6.3: Mã nguồn của hàm `foo` và đồ thị dòng điều khiển của nó.

Bảng 6.1: Các ca kiểm thử cho độ đo C_1 của hàm `foo`

ID	Inputs	EO	RO	Note
tc1	0, 1, 2, 3	0		
tc2	1, 1, 2, 3	1		

một bộ kiểm thử (tập các ca kiểm thử) được đo bằng tỷ lệ các thành phần thực sự được kiểm thử so với tổng thể sau khi đã thực hiện các ca kiểm thử. Thành phần liên quan có thể là câu lệnh, điểm quyết định, điều kiện con, đường thi hành hay là sự kết hợp của chúng. Độ bao phủ càng lớn thì độ tin cậy của bộ kiểm thử càng cao. Độ đo này giúp chúng ta kiểm soát và quản lý quá trình kiểm thử tốt hơn. Mục tiêu của chúng ta là kiểm thử với số ca kiểm thử tối thiểu nhưng đạt được độ bao phủ tối đa. Có rất nhiều độ đo kiểm thử đang được sử dụng hiện nay, dưới đây là ba độ đo kiểm thử đang được sử dụng phổ biến nhất trong thực tế [Lee03].

Độ đo kiểm thử cấp 1 (C_1): mỗi câu lệnh được thực hiện ít nhất một lần sau khi chạy các ca kiểm thử (test cases). Ví dụ, với hàm `foo` có mã nguồn như trong Hình 6.3, ta chỉ cần hai ca kiểm thử như Bảng 6.1 là đạt 100% độ phủ cho độ đo C_1 với EO (expected output) là giá trị đầu ra mong đợi và RO (real output) là giá trị đầu ra thực tế (giá trị này sẽ được điền khi thực hiện ca kiểm thử).

Bảng 6.2: Các trường hợp cần kiểm thử của độ đo C_2 với hàm `foo`

Điểm quyết định	**Điều kiện tương ứng**	**Đúng**	**Sai**
2	a==0	tc1	tc2
5	(a == b) || (c == d)	tc2	?

Bảng 6.3: Các ca kiểm thử cho độ đo C_2 của hàm `foo`

ID	**Inputs**	**EO**	**RO**	**Note**
tc1	0, 1, 2, 3	0		
tc2	1, 1, 2, 3	1		
tc3	1, 2, 1, 2	Lỗi chia cho 0		

Độ đo kiểm thử cấp 2 (C_2): các điểm quyết định trong đồ thị dòng điều khiển của đơn vị kiểm thử đều được thực hiện ít nhất một lần cả hai nhánh đúng và sai. Ví dụ, Bảng 6.2 mô tả các trường hợp cần kiểm thử để đạt được 100% độ phủ của độ đo C_2 ứng với hàm `foo` được mô tả trong Hình 6.3.

Như vậy, với hai ca kiểm thử trong độ đo kiểm thử cấp 1 (tc1 và tc2), ta chỉ kiểm thử được $3/4 = 75\%$ ứng với độ đo kiểm thử cấp 2. Chúng ta cần một ca kiểm thử nữa ứng với trường hợp sai của điều kiện `(a == b) || (c == d)` nhằm đạt được 100% độ phủ của độ đo C_2. Bảng 6.3 mô tả các ca kiểm thử cho mục đích này.

Độ đo kiểm thử cấp 3 (C_3): Với các điều kiện phức tạp (chứa nhiều điều kiện con cơ bản), việc chỉ quan tâm đến giá trị đúng sai là không đủ để kiểm tra tính đúng đắn của chương trình ứng với điều kiện phức tạp này. Ví dụ, nếu một điều kiện phức tạp gồm hai điều kiện con cơ bản, chúng ta có bốn trường hợp cần kiểm thử chứ không phải hai trường hợp đúng sai như độ đo C_2. Với các đơn vị chương trình có yêu cầu cao về tính đúng đắn, việc tuân thủ độ đo C_3 là hết sức cần thiết. Điều kiện để đảm bảo độ đo này là các điều kiện con thuộc các điều kiện phức tạp tương ứng với các điểm quyết định trong đồ thị dòng điều khiển của đơn vị cần kiểm thử đều được thực hiện ít nhất một lần cả hai nhánh đúng và sai. Ví dụ, Bảng 6.4 mô tả các trường hợp cần kiểm thử để đạt được 100% độ phủ của độ đo C_3 ứng với

Bảng 6.4: Các trường hợp cần kiểm thử của độ đo C_3 với hàm `foo`

Điểm quyết định	**Điều kiện tương ứng**	**Đúng**	**Sai**
2	a==0	tc1	tc2
5	(a == b)	tc2	tc3
5	(c == d)	?	tc2

Bảng 6.5: Các ca kiểm thử cho độ đo C_3 của hàm `foo`

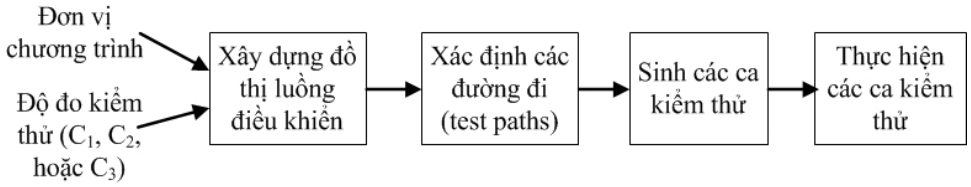
ID	**Inputs**	**EO**	**RO**	**Note**
tc1	0, 1, 2, 3	0		
tc2	1, 1, 2, 3	1		
tc3	1, 2, 1, 2	Lỗi chia cho 0		
tc4	1, 2, 1, 1	1		

hàm `foo` được mô tả trong Hình 6.3.

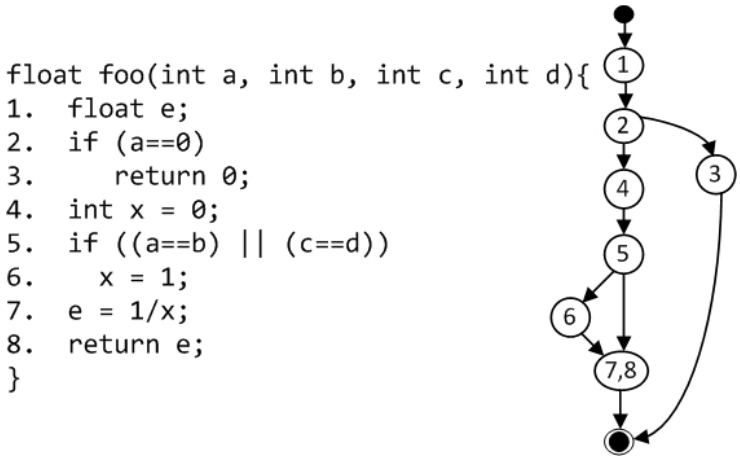
Như vậy, với ba ca kiểm thử trong độ đo kiểm thử cấp 2 (tc1, tc2 và tc3), ta chỉ kiểm thử được 7/8 = 87,5% ứng với độ đo kiểm thử cấp 3. Chúng ta cần một ca kiểm thử nữa ứng với trường hợp sai của điều kiện con cơ bản `(c == d)` nhằm đạt được 100% độ phủ của độ đo C_3. Bảng 6.5 mô tả các ca kiểm thử cho mục đích này.

6.4 Kiểm thử dựa trên độ đo

Kiểm thử dựa trên độ đo là phương pháp chạy mã nguồn sao cho bao phủ một độ đo nào đó. Hình 6.4 mô tả quy trình kiểm thử dựa trên độ đo cho các đơn vị chương trình. Với mỗi đơn vị chương trình, đồ thị dòng điều khiển ứng với các độ đo C_1 và C_2 là giống nhau trong khi chúng khác với đồ thị dòng điều khiển ứng với độ đo C_3. Với mỗi đơn vị chương trình và mỗi độ đo kiểm thử, chúng ta tiến hành xây dựng đồ thị dòng điều khiển tương ứng. Các đường đi của chương trình (xuất phát từ điểm bắt đầu, đi qua các đỉnh của đồ thị và kết thúc ở điểm cuối) được xác định sao cho khi chúng được thực hiện thì độ đo kiểm thử tương ứng được thỏa mãn. Dựa trên ý tưởng



Hình 6.4: Quy trình kiểm thử đơn vị chương trình dựa trên độ đo.



Hình 6.5: Mã nguồn của hàm `foo` và đồ thị dòng điều khiển của nó.

của T. J. McCabe [McC76, WM96], số đường đi chương trình ứng với đồ thị dòng điều khiển của nó được tính bằng một trong các phương pháp sau:

- Số cạnh – số đỉnh + 2
- Số đỉnh quyết định + 1

Sau khi có được các đường đi của đơn vị chương trình cần kiểm thử, với mỗi đường đi, chúng ta sẽ sinh một ca kiểm thử tương ứng. Cuối cùng, các ca kiểm thử được thực hiện trên đơn vị chương trình nhằm phát hiện các lỗi.

6.4.1 Kiểm thử cho độ đo C_1

Xét lại hàm `foo` có mã nguồn như Hình 6.5. Chúng ta xây dựng đồ thị dòng điều khiển ứng với độ phủ C_1 cho hàm này như Hình 6.5. Để đạt được 100%

Bảng 6.6: Các ca kiểm thử cho độ đo C_1 của hàm `foo`

ID	Test Path	Inputs	EO	RO	Note
tc1	1; 2; 4; 5; 6; 7,8	2, 2, 3, 5	1		
tc2	1; 2; 3	0, 3, 2, 7	0		

độ phủ của độ đo C_1, ta chỉ cần hai đường đi như sau để đảm bảo được tất cả các câu lệnh của hàm `foo` được kiểm thử ít nhất một lần. Để kiểm tra việc đảm bảo độ đo C_1, chúng ta cần kiểm tra tất cả các lệnh/khối lệnh (1-8) đều được xuất hiện ít nhất một lần trong các đường đi này. Rõ ràng, hai đường đi này thỏa mãn điều kiện trên nên chúng ta đạt được 100% độ phủ C_1.

1. 1; 2; 4; 5; 6; 7,8
2. 1; 2; 3

Với đường đi 1; 2; 4; 5; 6; 7,8, ta sẽ sinh một ca kiểm thử để nó được thực thi khi thực hiện ca kiểm thử này. Ý tưởng của việc sinh ca kiểm thử này là tìm một bộ giá trị đầu vào cho a, b, c và d sao cho điều kiện ứng với điểm quyết định 2 `(a == 0)` là sai và điều kiện ứng với điểm quyết định 5 `((a == b) || (c == d))` là đúng. Giá trị đầu ra mong đợi (EO) của ca kiểm thử này là 1. Tương tự, ta sẽ sinh ca kiểm thử ứng với đường đi 1; 2; 3 với đầu ra mong đợi là 0. Chúng ta sẽ tìm một bộ đầu vào sao cho điều kiện `(a == 0)` là đúng. Bảng 6.6 là một ví dụ về hai ca kiểm thử được sinh ra bằng ý tưởng trên.

6.4.2 Kiểm thử cho độ đo C_2

Như chúng ta đã biết, với mỗi đơn vị chương trình, đồ thị dòng điều khiển ứng với các độ đo C_1 và C_2 là giống nhau. Vì vậy, đồ thị dòng điều khiển ứng với độ đo C_1 của hàm `foo` được mô tả ở Hình 6.5 cũng là đồ thị dòng điều khiển của hàm này ứng với độ đo C_2. Tuy nhiên, để được 100% độ phủ của độ đo C_2 chúng ta cần tối thiểu ba đường đi. Tại sao chúng ta biết được điều này? Như đã trình bày ở mục 6.4, chúng ta có hai cách để tính được con số này. Ví dụ, đồ thị dòng điều khiển của hàm `foo` có hai điểm quyết định là 2 và 5 nên chúng ta cần $2 + 1 = 3$ đường đi để đạt được 100% độ phủ của độ

Bảng 6.7: Các ca kiểm thử cho độ đo C_2 của hàm `foo`

ID	Test Path	Inputs	EO	RO	Note
tc1	1; 2; 4; 5; 6; 7,8	2, 2, 3, 5	1		
tc2	1; 2; 3	0, 3, 2, 7	0		
tc3	1; 2; 4; 5; 7,8	2,3,4,5	lỗi chia cho 0		

đo C_2. Các đường đi cần thiết được liệt kê như sau. Rõ ràng với ba đường đi này, cả hai nhánh đúng và sai của hai điểm quyết 2 và 5 đều được kiểm tra.

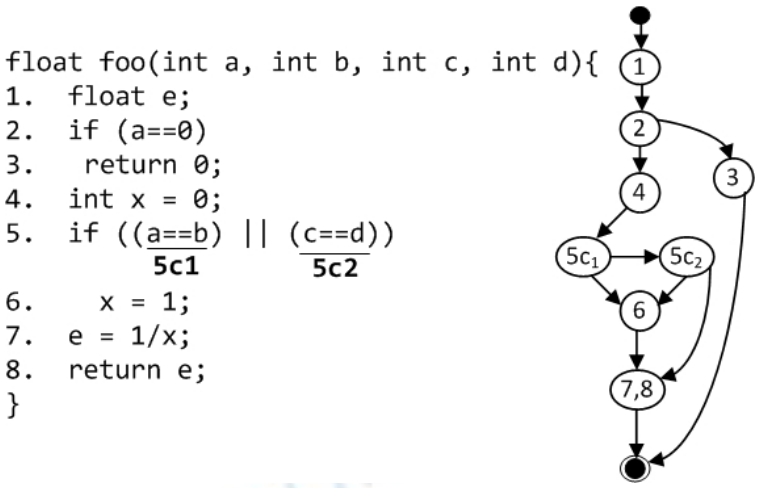
1. 1; 2; 4; 5; 6; 7,8
2. 1; 2; 3
3. 1; 2; 4; 5; 7,8

Để sinh các ca kiểm thử ứng với các đường đi trên, chúng ta chỉ cần quan tâm đến đường đi (3) vì việc sinh các ca kiểm thử cho các đường đi (1) và (2) đã được trình bày ở mục kiểm thử cho độ đo C_1 (mục 6.4.1). Với đường đi (3), ta chỉ cần chọn một bộ đầu vào sao cho điều kiện ứng với điểm quyết định 2 `(a == 0)` là sai và điều kiện ứng với điểm quyết định 5 `((a == b) || (c == d))` cũng là sai. Giá trị đầu ra mong đợi của đường đi này là lỗi chia cho 0. Bảng 6.7 là một ví dụ về ba ca kiểm thử được sinh ra bằng ý tưởng trên ứng với các đường đi (1), (2), và (3).

Độ đo C_1 đảm bảo các câu lệnh được "viếng thăm" ít nhất một lần khi thực hiện tất cả các ca kiểm thử được sinh ra ứng với độ đo này. Đây là độ đo khá tốt và việc đảm bảo độ đo này trong thực tế cũng khá tốn kém. Tuy nhiên, qua ví dụ trên, chúng ta thấy rằng nếu chỉ sử dụng độ đo C_1 với hai ca kiểm thử như trong bảng 6.6, lỗi chia cho không sẽ không được phát hiện. Chỉ khi kiểm tra cả hai nhánh đúng sai của tất cả các điểm quyết định (các lệnh điều khiển) thì lỗi này mới được phát hiện như ca kiểm thử tc3 trong bảng 6.7.

6.4.3 Kiểm thử cho độ đo C_3

Như đã trình bày ở mục 6.3, ứng với mỗi đơn vị chương trình, đồ thị dòng điều khiển ứng với độ đo C_3 khác với đồ thị dòng điều khiển ứng với các



Hình 6.6: Hàm `foo` và đồ thị dòng điều khiển của nó ứng với độ đo C_3.

độ đo C_1 và C_2. Ví dụ, đồ thị dòng điều khiển của hàm `foo` ứng với độ đo C_3 được xây dựng như Hình 6.6. Với câu lệnh điều kiện 5, vì đây là điều kiện phức tạp nên ta phải tách thành hai điều kiện con cơ bản là `(a == b)` và `(c == d)` ứng với hai điểm quyết định 5c1 và 5c2 trong đồ thị dòng điều khiển. Từ câu lệnh 4, nếu điều kiện con `(a == b)` đúng, ta không cần kiểm tra điều kiện con còn lại (vì điều kiện phức tạp là hoặc của hai điều kiện con cơ bản) và thực hiện câu lệnh 6. Nếu điều kiện con `(a == b)` là sai, ta cần tiến hành kiểm tra điều kiện con cơ bản còn lại `(c == d)`. Nếu điều kiện này đúng, ta tiến hành câu lệnh 6. Ngược lại, chúng ta thực hiện các câu lệnh 7 và 8. Trong đồ thị này, chúng ta gộp hai lệnh 7 và 8 trong một đỉnh (đỉnh (7,8)) vì đây là hai câu lệnh tuần tự. Mục đích của việc này là nhằm tối thiểu số đỉnh của đồ thị dòng điều khiển. Một đồ thị có số đỉnh càng nhỏ thì chúng ta càng dễ dàng trong việc sinh các đường đi của chương trình và tránh các sai sót trong quá trình này.

Đồ thị dòng điều khiển của hàm `foo` ứng với độ đo C_3 như Hình 6.6 có ba điểm quyết định là 2, 5c1 và 5c2 nên chúng ta cần $3 + 1 = 4$ đường đi để được 100% độ phủ của độ đo C_3. Các đường đi cần thiết được liệt kê như sau:

1. 1; 2; 4; 5c1; 6; 7,8
2. 1; 2; 4; 5c1; 5c2; 6; 7,8

Bảng 6.8: Các ca kiểm thử cho độ đo C_3 của hàm `foo`

ID	Test Path	Inputs	EO	RO	Note
tc1	1; 2; 4; 5c1; 6; 7,8	0, 2, 3, 5	0		
tc2	1; 2; 4; 5c1; 5c2; 6; 7,8	2, 2, 2, 7	1		
tc3	1; 2; 4; 5c1; 5c2; 7,8	2,3,4,5	lỗi chia cho 0		
tc4	1; 2; 3	2,3,4,4	1		

3. 1; 2; 4; 5c1; 5c2; 7,8

4. 1; 2; 3

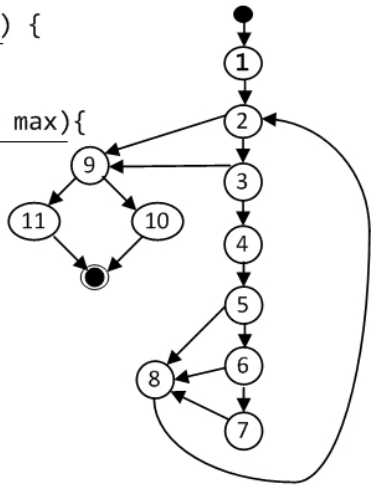
Tương tự như các phương pháp kiểm thử độ đo C_1 và C_2, chúng ta dễ dàng sinh các ca kiểm thử tương ứng cho các đường đi chương trình như đã mô tả trên. Bảng 6.8 là một ví dụ về các ca kiểm thử cho hàm `foo` ứng với độ đo C_3.

6.4.4 Kiểm thử vòng lặp

Cho dù chúng ta tiến hành kiểm thử các đơn vị chương trình với độ đo C_3 (độ đo với yêu cầu cao nhất), phương pháp kiểm thử dòng điều khiển không thể kiểm thử các vòng lặp xuất hiện trong các đơn vị chương trình. Lý do là các đường đi sinh ra từ đồ thị dòng điều khiển không chứa các vòng lặp. Trong thực tế, lỗi hay xảy ra ở các vòng lặp. Vì lý do này, chúng ta cần sinh thêm các ca kiểm thử cho các vòng lặp nhằm giảm tỷ lệ lỗi của các đơn vị chương trình. Với mỗi đơn vị chương trình có vòng lặp, chúng ta cần quan tâm đến ba trường hợp sau:

- **Lệnh lặp đơn giản:** đơn vị chương trình chỉ chứa đúng một vòng lặp (thân của vòng lặp không chứa các vòng lặp khác).
- **Lệnh lặp liền kề:** đơn vị chương trình chỉ chứa các lệnh lặp kế tiếp nhau.
- **Lệnh lặp lồng nhau:** đơn vị chương trình chỉ chứa các vòng lặp chứa các lệnh lặp khác.

```
double average(double value[], double min, double
     max, int& tcnt, int& vcnt) {
     double sum = 0;
     int i = 1;          |1
     tcnt = vcnt = 0;    |
     while (value[i] <> -999 && tcnt <100) {
                 2              3
          tcnt++; |4
          if (min<=value[i] && value[i]<= max){
                    5              6
                    sum += value[i];|7
                    vcnt ++;        |
          }
          i++;|8
     }//end while
     if (vcnt > 0)
           9
          return sum/vcnt; |10
     return -999; |11
}//end
```



Hình 6.7: Hàm `average` và đồ thị dòng điều khiển của nó ứng với độ đo C_3.

Để kiểm thử các đơn vị chương trình chỉ có lệnh lặp đơn giản, ta xét hàm average với mã nguồn đồ thị dòng điều khiển tương ứng với độ đo C_3 như Hình 6.7.

Đồ thị dòng điều khiển của hàm `average` ứng với độ đo C_3 như Hình 6.7 có năm điểm quyết định là 2, 3, 5, 6 và 9 nên chúng ta cần $5 + 1 = 6$ đường đi để được 100% độ phủ của độ đo C_3. Các đường đi cần thiết được liệt kê như sau:

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

Bảng 6.9: Các ca kiểm thử cho độ đo C_3 của hàm `average`

ID	Test Path	Inputs	EO	RO	Note
tc1	1; 2; 9; 10				tc6
tc2	1; 2; 9; 11	[-999, ...], 1, 2	-999		
tc3	1; 2; 3; 9; 10				tc6
tc4	1; 2; 3; 4; 5; 8; 2; 9; 11	[0,-999], 1, 2	-999		
tc5	1; 2; 3; 4; 5; 6; 8; 2; 9; 11	[3,-999], 1, 2	-999		
tc6	1; 2; 3; 4; 5; 6; 7; 8; 2; 9; 10	[1,-999], 1, 2	1		

Với mỗi đường đi, chúng ta sẽ sinh một ca kiểm thử tương ứng. Bảng 6.9 mô tả các ca kiểm thử cho hàm `average` ứng với độ đo C_3. Với mỗi ca kiểm thử, bộ dữ liệu đầu vào (inputs) gồm ba thành phần: `double value[]`, `double min`, và `double max`. Với đường đi 1; 2; 9; 10, chúng ta không thể tìm được bộ dữ liệu đầu vào để đường đi này được thực thi. Thực vậy, điều kiện để thực thi đường đi này là `value[0] = -999` (tức là điều kiện 2 là sai) và điều kiện 9 đúng (tức là `vcnt > 0`). Điều này không bao giờ xảy ra vì nếu `value[0]` = -999 thì `vcnt = 0`. Tương tự, chúng ta cũng không thể sinh bộ kiểm thử với đường đi 1; 2; 3; 9; 10. Trong các trường hợp này, chúng ta không cần sinh các ca kiểm thử cho những đường đi này. Chúng sẽ được kiểm thử bởi các đường đi khác (Ví dụ: đường đi 1; 2; 3; 4; 5; 6; 7; 8; 2; 9; 10 chứa các đỉnh của các đường dẫn trên).

Với các đường đi trên, vòng lặp while của hàm `average` chỉ được thực hiện tối đa một lần lặp nên chúng ta rất khó để phát hiện các lỗi tiềm ẩn (có thể có) bên trong vòng lặp này. Các lỗi này có thể xảy ra khi vòng lặp này được thực hiện nhiều lần lặp. Ví dụ trên đã chỉ ra những hạn chế của phương pháp kiểm thử dòng điều khiển khi áp dụng cho các chương trình/đơn vị chương trình có chứa vòng lặp. Để giải quyết vấn đề này, chúng ta cần sinh thêm bảy ca kiểm thử ứng với bảy trường hợp sau:

1. Vòng lặp thực hiện 0 lần
2. Vòng lặp thực hiện 1 lần
3. Vòng lặp thực hiện 2 lần

Bảng 6.10: Các ca kiểm thử cho cho kiểm thử vòng lặp `while` của hàm `average`

ID	Số lần lặp	Inputs	EO	RO	Note
tcl0	0	[-999, ...], 1, 2	-999		
tcl1	1	[1,-999], 1, 2	1		
tcl2	2	[1,2,-999], 1, 2	1.5		
tclk	5	[1,2,3,4,5,-999], 1, 10	3		
tcl(n-1)	99	[1,2,...,99,-999], 1, 100	50		
tcln	100	[1,2,...,100], 1, 2	50.5		
tcl(n+1)					

4. Vòng lặp thực hiện k lần, 2 < k < n - 1, với n là số lần lặp tối đa của vòng lặp
5. Vòng lặp thực hiện n - 1 lần
6. Vòng lặp thực hiện n lần
7. Vòng lặp thực hiện n + 1 lần

Chú ý rằng trong một số trường hợp chúng ta có thể không xác định được số lần lặp tối đa của các vòng lặp. Trong trường hợp này, chúng ta chỉ cần sinh bốn ca kiểm thử đầu tiên. Tương tự, trong một số các trường hợp khác, chúng ta không thể sinh ca kiểm thử để vòng lặp thực hiện n + 1 lần (trường hợp thứ 7). Khi đó, chúng ta chỉ cần sinh sáu ca kiểm thử còn lại (các trường hợp từ 1–6). Ví dụ, với vòng lặp `while` trong hàm `average` như Hình 6.7, vòng lặp này chỉ thực hiện lặp tối đa 100 lần nên chúng ta không thể sinh ca kiểm thử để nó thực hiện n + 1 = 101 lần. Kết quả là chúng ta chỉ cần sinh sáu ca kiểm thử đầu tiên như trong Bảng 6.10 nhằm kiểm thử vòng lặp này.

Với các chương trình/đơn vị chương trình có các vòng lặp liền kề, chúng ta tiến hành kiểm thử tuần tự từ trên xuống. Mỗi vòng lặp được kiểm thử bằng bảy ca kiểm thử như vòng lặp đơn giản (như đã mô tả ở trên). Trong trường hợp các vòng lặp lồng nhau, chúng ta tiến hành kiểm thử tuần tự các vòng lặp theo thứ tự từ trong ra ngoài (mỗi vòng lặp cũng dùng bảy ca kiểm thử như đã mô tả ở trên).

6.5 Tổng kết

Kiểm thử dòng điều khiển là một trong những phương pháp kiểm thử quan trọng nhất của chiến lược kiểm thử hộp trắng cho các chương trình/đơn vị chương trình. Phương pháp này cho phép phát hiện ra các lỗi (có thể có) tiềm ẩn bên trong chương trình/đơn vị chương trình bằng cách kiểm thử các đường đi của nó tương ứng với các dòng điều khiển có thể có. Để áp dụng phương pháp này, chúng ta cần xác định độ đo kiểm thử (cần kiểm thử với độ đo nào?). Tiếp theo, đồ thị dòng điều khiển của chương trình/đơn vị chương trình ứng với độ đo kiểm thử sẽ được tạo ra. Dựa vào đồ thị này, chúng ta sẽ sinh ra các đường đi độc lập. Số đường đi độc lập này là các trường hợp tối thiểu nhất để đảm bảo 100% độ bao phủ ứng với độ đo yêu cầu. Với mỗi đường đi, chúng ta sẽ sinh ra một ca kiểm thử sao cho khi nó được dùng để kiểm thử thì đường đi này được thực thi. Việc sinh các đầu vào cho các ca kiểm thử này là một bài toán thú vị. Chúng ta sẽ chọn một bộ đầu vào sao cho thỏa mãn các điểm quyết định có trong đường đi tương ứng. Giá trị đầu ra mong muốn ứng với mỗi bộ đầu vào của mỗi ca kiểm thử cũng sẽ được tính toán. Đây là bài toán khó và thường chỉ có các chuyên gia phân tích chương trình mới có thể trả lời chính xác giá trị này. Cuối cùng, các ca kiểm thử được chạy nhằm phát hiện ra các lỗi của chương trình/đơn vị chương trình cần kiểm thử. Khi một lỗi được phát hiện bởi một ca kiểm thử nào đó, nó sẽ được thông báo tới lập trình viên tương ứng. Lập trình viên sẽ tiến hành sửa lỗi (phát hiện vị trí của lỗi ở câu lệnh nào và sửa nó). Trong trường hợp này, chúng ta không chỉ thực hiện lại ca kiểm thử phát hiện ra lỗi này mà phải thực hiện lại tất cả các ca kiểm thử của đơn vị chương trình. Lý do chúng ta phải thực hiện công việc này là vì khi sửa lỗi này có thể gây ra một số lỗi khác.

Việc áp dụng phương pháp kiểm thử dòng điều khiển là khó và tốn kém hơn các phương pháp kiểm thử hộp đen (phân hoạch tương đương, phân tích giá trị biên, bảng quyết định, ...). Để áp dụng kỹ thuật này, chúng ta cần đội ngũ nhân lực về kiểm thử có kiến thức và kỹ năng tốt. Hơn nữa, chúng ta cần một sự đầu tư lớn về các nguồn lực khác (tài chính, thời gian, ...) mới có thể thực hiện tốt phương pháp này. Đây là một yêu cầu khó và không nhiều công ty phần mềm đáp ứng được. Kiểm thử dòng điều khiển tự động hứa hẹn sẽ là một giải pháp tốt nhằm giúp cho các công ty giải quyết những khó khăn này. Hiện nay, đã có nhiều công cụ hỗ trợ một phần hoặc hoàn toàn các bước trong phương pháp này. Một môn trường lập trình mới nơi

mà các công cụ lập trình trực quan được tích hợp với công cụ kiểm thử tự động nhằm giải quyết những khó khăn nêu trên. Eclipse[1] là một ví dụ điển hình về xu hướng này. Trong cá phiên bản hiện nay của mô trường này đã được tích hợp công cụ kiểm thử tự động có tên là JUnit[2]. Công cụ kiểm thử này chưa hỗ trợ việc sinh các ca kiểm thử tự động nhưng nó cho phép chúng ta viết các kịch bản kiểm thử độc lập với mã nguồn và thực thi chúng một cách tự động trên một môi trường thống nhất.

6.6 Bài tập

1. Tại sao chúng ta cần thực hiện kiểm thử hộp trắng?

2. Phân biệt kiểm thử hộp trắng và kiểm thử hộp đen.

3. Tại sao kiểm thử hộp trắng thường có chi phí và độ khó cao hơn kiểm thử hộp đen?

4. Thế nào là đồ thị dòng điều khiển của một chương trình/đơn vị chương trình?

5. Trình bày các độ đo kiểm thử cho kiểm thử dòng điều khiển.

6. Chứng minh rằng độ đo C_2 đảm bảo độ đo C_1.

7. Chứng minh rằng độ đo C_3 đảm bảo độ đo C_2.

8. Trình bày các bước nhằm kiểm thử một đơn vị chương trình theo phương pháp kiểm thử dòng điều khiển với một độ đo kiểm thử cho trước.

9. Cho hàm được viết bằng ngôn ngữ C như Hình 6.8.

    - Hãy xây dựng đồ thị dòng điều khiển cho hàm `Grade` ứng với độ đo C_1 và C_2.
    - Hãy sinh các đường đi và các ca kiểm thử với độ đo C_1.
    - Hãy sinh các đường đi và các ca kiểm thử với độ đo C_2.

```
char Grade(int score){
      int res;
      if(score < 0 || score > 10)
            return 'I';
      if(score>=9)
            res = 'A';
      else
            if(score >=8)
                  res ='B';
            else
                  if(score >=6.5)
                        res = 'C';
                  else
                        if(score >=5)
                              res = 'D';
                        else
                              res = 'F';
      return res;
}
```

Hình 6.8: Mã nguồn của hàm `Grade`.

10. Cho hàm được viết bằng ngôn ngữ C như Hình 6.9.

    - Hãy xây dựng đồ thị dòng điều khiển cho hàm `BinSearch` ứng với độ đo C_1 và C_2.
    - Hãy sinh các đường đi và các ca kiểm thử với độ đo C_1.
    - Hãy sinh các đường đi và các ca kiểm thử với độ đo C_2.
    - Hãy sinh các ca kiểm thử để kiểm thử vòng lặp `while`.

11. Cho hàm được viết bằng ngôn ngữ C như Hình 6.10.

    - Hãy xây dựng đồ thị dòng điều khiển cho hàm `LaSoNguyenTo` ứng với độ đo C_1 và C_2.
    - Hãy sinh các đường đi và các ca kiểm thử với độ đo C_1.
    - Hãy sinh các đường đi và các ca kiểm thử với độ đo C_2.
    - Hãy sinh các ca kiểm thử để kiểm thử vòng lặp `do ... while`.

[1] www.eclipse.org
[2] http://junit.org/

```
int Binsearch(int X, int V[], int n){
     int low, high, mid;
     low = 0;
     high = n - 1;
     while (low <= high) {
          mid = (low + high)/2;
          if (X < V[mid])
               high = mid - 1;
          else
               if (X > V[mid])
                    low = mid + 1;
               else
                    return mid;
     }//end while
     return -1;
}//the end
```

Hình 6.9: Mã nguồn của hàm `BinSearch`.

12. Cho hàm được viết bằng ngôn ngữ C như Hình 6.11.

    - Hãy xây dựng đồ thị dòng điều khiển cho hàm `UCLN` ứng với độ đo C_1 và C_2.
    - Hãy sinh các đường đi và các ca kiểm thử với độ đo C_1.
    - Hãy sinh các đường đi và các ca kiểm thử với độ đo C_2.
    - Hãy sinh các ca kiểm thử để kiểm thử vòng lặp `while`.

13. Cho hàm được viết bằng ngôn ngữ C như Hình 6.12.

```
//tra lai 1 neu n la so nguyen to, nguoc lai tra lai 0
int LaSoNguyenTo(int n){
     int i=2;
     do{
          if((n % i) == 0)
               return 0;
          i++;
     } while(i <= n/2);
     return 1;
}
```

Hình 6.10: Mã nguồn của hàm `LaSoNguyenTo`.

```
int UCLN(int m, int n){
     if (m < 0) m = -m;
     if (n < 0) n = -n;
     if (m == 0)   return n;
     if (n == 0)   return m;
     while (m != n) {
          if(m > n)
               m = m - n;
          else
               n = n - m;
     }//end while
     return m;
 }
```

Hình 6.11: Mã nguồn của hàm `UCLN`.

```
int sum(int a[], int n){
     int i;
     int total = 0;
     for(i=0; i<n; i++)
          total = total + a[i];
     return total;
}
```

Hình 6.12: Mã nguồn của hàm `Sum`.

- Hãy xây dựng đồ thị dòng điều khiển cho hàm `sum` ứng với độ đo C_1 và C_2.
- Hãy sinh các đường đi và các ca kiểm thử với độ đo C_1.
- Hãy sinh các đường đi và các ca kiểm thử với độ đo C_2.
- Hãy sinh các ca kiểm thử để kiểm thử vòng lặp `for`.

14. Cho hàm được viết bằng ngôn ngữ C như Hình 6.13.

- Hãy xây dựng đồ thị dòng điều khiển cho hàm `Power` ứng với độ đo C_1 và C_2.
- Hãy sinh các đường đi và các ca kiểm thử với độ đo C_1.
- Hãy sinh các đường đi và các ca kiểm thử với độ đo C_2.
- Hãy sinh các ca kiểm thử để kiểm thử vòng lặp `for`.

```
double    Power(double x, int n)
{
     double    result = 1;
     for(int i=1; i< n; i++)
          result *= x;

     return result;
}
```

Hình 6.13: Mã nguồn của hàm `Power`.

```
//tra ve 1 neu year la nham nhuan, nguoc lai tra ve 0
int LaNamNhuan(int year){
     if( (year%400==0) || (year%4==0 && year%100!=0))
          return 1;

     return 0;
}
```

Hình 6.14: Mã nguồn của hàm `LaNamNhuan`.

15. Cho hàm được viết bằng ngôn ngữ C như Hình 6.14.

    - Hãy xây dựng đồ thị dòng điều khiển cho hàm `LaNamNhuan` ứng với độ đo C_1 và C_2.
    - Hãy sinh các đường đi và các ca kiểm thử với độ đo C_1.
    - Hãy sinh các đường đi và các ca kiểm thử với độ đo C_2.
    - Hãy xây dựng đồ thị dòng điều khiển cho hàm `LaNamNhuan` ứng với độ đo C_3.
    - Hãy sinh các đường đi và các ca kiểm thử với độ đo C_3.

16. Cho hàm được viết bằng ngôn ngữ C như Hình 6.15.

    - Hãy xây dựng đồ thị dòng điều khiển cho hàm `SoKyTu` ứng với độ đo C_1 và C_2.
    - Hãy sinh các đường đi và các ca kiểm thử với độ đo C_1.
    - Hãy sinh các đường đi và các ca kiểm thử với độ đo C_2.
    - Hãy xây dựng đồ thị dòng điều khiển cho hàm `LaNamNhuan` ứng với độ đo C_3