BỘ GIÁO DỤC VÀ ĐÀO TẠO

TRƯỜNG ĐẠI HỌC DÂN LẬP HẢI PHÒNG

PHẠM THỊ HÙY

KỸ THUẬT XÁC ĐỊNH CÁC CA KIỂM THỬ VÀ DỮ LIỆU KIỂM THỬ NHỜ MA TRẬN KIỂM THỬ

LUẬN VĂN THẠC SĨ CÔNG NGHỆ THÔNG TIN

CHUYÊN NGÀNH: HỆ THỐNG THÔNG TIN

MÃ SỐ: 60480104

NGƯỜI HƯỚNG DẪN KHOA HỌC:

TS. LÊ VĂN PHÙNG

Hải Phòng, 2017

LỜI CẢM ƠN

Trân trọng cảm ơn tất cả các Giáo sư, Phó giáo sư, tiến sĩ, các thầy giáo, cô giáo của khoa CNTT trường Đại học Dân lập Hải Phòng đã nhiệt tình giảng dạy, tạo điều kiện thuận lợi cho tác giả trong quá trình học tập, nghiên cứu, hoàn thành chương trình học tập của khóa học.

Tác giả xin trân trân trọng cảm ơn thầy TS. Lê Văn Phùng - Viện Công nghệ thông tin - Viện Hàn Lâm khoa học và công nghệ Việt Nam đã giành thời gian chỉ bảo tận tình giúp em hoàn thành luận văn.

Tác giả xin chân thành cảm ơn Ban giám hiệu trường THCS Thủy Sơn đã quan tâm giúp đỡ, tạo mọi điều kiện thuận lợi cho tác giả trong suốt quá trình học tập, nghiên cứu và hoàn thiện luận văn.

Tác giả xin cảm ơn gia đình, bạn bè, đồng nghiệp đã động viên tiếp thêm nghị lực để tác giả hoàn thành khóa học và luận văn.

Mặc dù đã có nhiều cố gắng, song luận văn khó tránh khỏi những thiếu sót. Tác giả rất mong sự chỉ bảo, góp ý của các nhà khoa học, các thầy cô giáo và các bạn đồng nghiệp.

Xin trân trọng cảm ơn!

Hải Phòng, ngày 12 tháng 11 năm 2017

Tác giả

Phạm Thị Hùy

LỜI CAM ĐOAN

Tôi xin cam đoan rằng, đây là công trình nghiên cứu của tôi trong đó có sự giúp đỡ rất lớn của thầy TS. Lê Văn Phùng. Các nội dung nghiên cứu và kết quả trong đề tài này là hoàn toàn trung thực.

Trong luận văn, tôi có tham khảo đến một số tài liệu của một số tác giả đã được liệt kê tại phần Tài liệu tham khảo ở cuối luận văn.

Hải Phòng, ngày 12 tháng 11 năm 2017

Tác giả

Phạm Thị Hùy

MỤC LỤC

DANH MỤC CÁC KÍ HIỆU, CHỮ CÁI VIẾT TẮT

CNTT	Công nghệ thông tin
CSDL	Cơ sở dữ liệu
Software technology	Công nghệ phần mềm
Software Engineering	Kỹ nghệ phần mềm
Software testing	Kiểm thử phần mềm
Software quality assurance (SQA)	Bảo đảm chất lượng phần mềm
Template	Tiêu bản
Test case	Ca kiểm thử
Comparision testing	Kiểm thử so sánh
Dynamic testing	Kiểm thử động
Acceptance testing	Kiểm thử chấp thuận
Statement Coverge	Bao phủ câu lệnh
Decision coverage	Bao phủ quyết định
Decision/ Condition coverage	Bao phủ quyết định/ điều kiện
Equivalence Patitioning	Phân lớp tương đương
Branch and Relational Operation	Chiến lược kiểm thử

DANH MỤC CÁC HÌNH VẼ

DANH MỤC CÁC BẢNG

PHẦN MỞ ĐẦU

1. Lý do chọn đề tài

Khoa học kỹ thuật ngày càng phát triển với tốc độ cao, hàng loạt các sản phẩn phần mềm được đưa ra phục vụ cho con người. Mỗi ngày chúng ta nghe đâu đó tin tức về vấn đề an toàn, bảo mật thông tin, một ngân hàng báo cáo số dư tài khản không chín xác, một đoàn tầu bị va chạm, một máy bay bị mất trong không gian hoặc một hacker truy cập đến hàng triệu thẻ tín dụng. Tại sao điều này xẩy ra? Có thể do lập trình viên máy tính không tìm ra cách để làm cho phần mềm đơn giản?

Phần mềm ngày càng trở nên phức tạp hơn, có được nhiều tính năng hơn, được thiết kế nối với nhau nhiều hơn và cũng có nhiều trục trặc hơn từ chương trình. Việc xây dựng và phát triển phần mềm ngày càng được nâng cao hơn bằng các công cụ hỗ trợ tiên tiến. Nhờ vào đó mà các chuyên gia phát triển thực hiện hiệu quả và đem lại nhiều lợi nhuận hơn trước. Tuy nhiên với công nghệ ngày càng cao thì đòi hỏi mức độ ứng dụng lớn và phát sinh ra sự phức tạp cùng với chi phí, thời gian tăng lên. Do đó phương pháp để cải thiện điều này chính là thực hiện kết hợp giữa xây dựng và quá trình kiểm thử. Hầu hết các công ty phần mềm lớn đều cam kết về chất lượng phần mềm do họ tạo ra, đó là một trong những nhiệm vụ khó khăn nhất hiện nay. Có nhiều lí do cho việc này:

Một sản phẩn phần mềm không phải là một đối tượng hữu hình có thể do được, cơ thể cảm thấy hoặc lấy mẫu vì vậy rất khó khăn để thử nghiệm một sản phẩm phần mềm.

Kiểm thử phần mềm vẫn không được coi là một trao đổi thương mại được công nhận, do đó việc tìm kiếm được những người chuyên nghiệp đủ điều kiện cho các công việc thử nghiệm là khó khăn.

Không giống như quá trình sản xuất đã được xác định và tiêu chuẩn hóa thiết kế sản phẩn trong quá trình, kiểm soát chất lượng, quy trình tương tự như tiêu chuẩn hóa vẫn chưa được xác định để thử nghiệm phần mềm.

Các công cụ tự động hóa hoạt động kiểm thử phần mềm vẫn còn trong giai đoạn mới bắt đầu, còn phải mất nhiều thời gian để có các công cụ tự động hóa tinh vi có sẵn cho các hoạt động kiểm thử phần mềm.

Nỗ lực tìm kỹ thuật mới cho các hoạt động thử nghiệm phần mềm vẫn đang được phát triển...

Tầm quan trọng của kiểm thử phần mềm là quá bao la bất cứ thất bại của sản phẩm phần mềm hoặc ứng dụng đều có thể gây thiệt hại hàng tỷ đồng cho công ty. Thậm chí nếu các lỗi phần mềm không phải là quá lớn, chi phí hỗ trợ để có thể chạy thử cũng mất cả chục tới trăm triệu trong vòng đời của sản phẩm phần mềm.

Một khiếm khuyết trong sản phẩn là gì? làm thế nào nó ảnh hưởng đến người sử dụng, những gì người dùng cảm thấy khi tìm thấy một khiếm khuyết trong sản phẩn sau khi mua và sử dụng nó, làm thế nào để ngăn ngừa nó, và cuối cùng là làm thế nào để xác định và loại bỏ các khuyết điểm đó.

Các phương pháp thiết kế ca kiểm thử và ứng dụng có vai trò cực kỳ quan trọng trong việc đưa một ứng dụng bảo đảm chất lượng vào thực tế.

Kiểm thử là một trong những giai đoạn của quá trình phát triển, hoàn thành sản phẩm. Trước khi sản phẩm được phát hành tất cả các chức năng cũng như giao diện, ứng dụng của sản phẩm đó đều cần qua kiểm thử. Một sản phẩm tuy được thiết kế tốt nhưng cũng không thể tránh khỏi các sai sót. Kiểm thử hiệu quả sẽ phát hiện ra được các sai sót này, tránh các lỗi trước khi phát hành sản phẩm. Kiểm thử đứng dưới vai trò của người sử dụng, sẽ giúp cho sản phẩm có sự thích ứng phù hợp hơn với thị hiếu và nhu cầu ngày càng cao của người dùng. Vì vậy, cần nghiên cứu về cách thiết kế ca kiểm thử để kiểm thử hiệu quả, đặc biệt là cách thiết kế có khả năng số hóa. Đó cũng là lý do mà tôi chọn đề tài "**Kỹ thuật xác định các ca kiểm thử và dữ liệu kiểm thử nhờ ma trận kiểm thử"** làm luận văn thạc sĩ của mình.

2. Mục đích nghiên cứu

Xác định vai trò của phương pháp ca kiểm thử và ứng dụng trong nghiên cứu khoa học và chế tạo sản phẩm. Trên cơ sở nghiên cứu và phân tích

các giải pháp an toàn trong việc kiểm thử sản phẩm, phân phối, trao đổi, cũng như các phương thức quản lý nhằm mang lại hiệu quả cao nhất trong quá trình đưa một sản phẩm nào đó vào ứng dụng thực tế.

Kiểm thử giúp rút ngắn thời gian và giảm chi phí cho sản phẩm phần mềm. Nó giúp cho các chuyên gia kiểm thử tìm ra lỗi trong quá trình tạo ra phần mềm và đảm bảo hơn về chất lượng. Kiểm thử thực hiện chặt chẽ sẽ hạn chế lỗi, tuy nhiên trong phần mềm vẫn còn tiềm ẩn các lỗi và có thể phát sinh bất cứ lúc nào dẫn đến khả năng gây thiệt hại cho nhà sản xuất vì vậy cần thực hiện quá trình kiểm thử liên tục, xuyên suốt trong các giai đoạn phát triển của phần mềm. Đó là phương pháp tốt nhất để đảm bảo cho các yêu cầu của người dùng về thiết kế và ứng dụng phần mềm được đáp ứng đầy đủ.

3. Nhiệm vụ nghiên cứu

Nghiên cứu các lý thuyết tổng quan về phương pháp ca kiểm thử và ứng dụng: Các khái niệm cơ bản, tiến trình kiểm thử, các phương pháp, kỹ thuật kiểm thử và ứng dụng.

Nghiên cứu về phương pháp thiết kế ca kiểm thử và ứng dụng, các vấn đề cần lưu ý khi thiết kế, kiểm thử và ứng dụng sản phẩm.

Ứng dụng thử nghiệm đối với phương pháp ca kiểm thử và ứng dụng, nêu các kết quả đạt được, chưa đạt được và hướng giải quyết các vấn đề khi kiểm thử và ứng dụng sản phẩm.

4. Đối tượng và phạm vi nghiên cứu

Đối tượng nghiên cứu của đề tài tập trung vào các phương pháp ca kiểm thử và ứng dụng phần mềm và phần cứng.

Phạm vi nghiên cứu của đề tài được giới hạn trong vấn đề kiểm thử hộp trắng và ma trận kiểm thử.

Để kiểm thử không mất nhiều thời gian và chi phí, người ta bắt đầu đưa nó vào trong các ứng dụng xây dựng phần mềm để có thể kết hợp chúng lại với nhau. Việc làm này biến quy trình kiểm thử thành quy trình bắt buộc thực hiện trong mỗi dự án phần mềm. Nhưng để có hiệu quả cao nhất người kiểm thử cần có phương án, kế hoạch hay chiến lược riêng cho từng ứng dụng phần mềm.

5. Đóng góp mới của luận văn

Xác định được các tiêu chuẩn thích hợp cho việc chọn phương pháp thiết kế ca kiểm thử và dữ liệu kiểm thử nhờ ma trân kiểm thử.

6. Phương pháp nghiên cứu

- Phương pháp tổng hợp phân tích các vấn đề liên quan đến đề tài.
- Phương pháp thống kê kết hợp với phương pháp chuyên gia.
- Phương pháp kết hợp lý thuyết với thực nghiệm trên máy tính.

7. Những nội dung chính của luận văn:

Bố cục của luận văn gồm 3 chương:

Chương 1: Tổng quan về kiểm thử phần mềm và ca kiểm thử: Khái niệm về kiểm thử phần mềm, chiến lược kiểm thử phần mềm, các phương pháp và kỹ thuật kiểm thử. Những nét chung nhất về ca kiểm thử: Khái niệm ca kiểm thử, vấn đề thiết kế ca kiểm thử.

Chương 2: Các kỹ thật thiết kế ca kiểm thử gồm: Kỹ thuật bao phủ câu lệnh, kỹ thuật phân lớp tương đương, kỹ thuật phân tích giá trị biên, kỹ thuật đồ thị nguyên nhân – kết quả, kỹ thuật đoán lỗi.

Chương 3: Phần mềm thử nghiệm thiết kế ca kiểm thử: Kỹ thuật bao phủ câu lệnh dựa vào ma trận kiểm thử để thiết kế ca kiểm thử và dữ liệu kiểm thử**,** áp dụng thiết kế tự động ca kiểm thử cho một số mô-đun chương trình trong bài giảng về câu lệnh có cấu trúc tại Trường THCS Thủy Sơn Hải Phòng.

CHƯƠNG 1
TỔNG QUAN VỀ KIỂM THỬ PHẦN MỀM VÀ CA KIỂM THỬ

1.1.Tổng quan về kiểm thử phần mềm

1.1.1.Khái niệm về kiểm thử phần mềm

Kiểm thử phần mềm (software testing) là một trong những yếu tố góp phần bảo đảm chất lượng phần mềm (SQA), là khâu điển hình kiểm soát đặc tả, thiết lập, lập mã.

Theo Glen Myers: *"Kiểm thử phần mềm là quá trình vận hành chương trình để tìm ra lỗi"*.

Cần vận hành như thế nào để:

- Hiệu suất tìm ra lỗi là cao nhất.
- Chi phí (thời gian, công sức) ít nhất.

(*Giáo trình kỹ nghệ phần mềm*. Nguyễn Văn Vỵ, Nguyễn Việt Hà. Nhà xuất bản Giáo dục Việt Nam năm 2009)

Kiểm thử phần mềm được đặt ra với những lý do:

- Muốn nhận diện phần mềm như một phần tử của hệ thống hoạt động.
- Hạn chế chi phí cho các thất bại do lỗi gây ra sau này (hiệu quả)
- Có kế hoạch tốt nâng cao chất lượng suốt quá trình phất triển (giải pháp).

Kiểm thử giữ vai trò lớn trong quá trình phát triển phần mềm. Xét theo tiêu chí về chi phí thì kiểm thử chiếm:

- 40% công sức phát triển;
- ≥ 30% tổng thời gian phát triển;
- Với các phần mềm có ảnh hưởng tới sinh mạng, chi phí có thể gấp từ 3 đến 5 lần tổng các chi phí khác cộng lại.
- Giảm chi phí phát triển;
- Tăng độ tin cậy của sản phẩm phần mềm.

Vấn đề đặt ra là cần vận hành phần mềm như thế nào để:

- Hiệu suất tìm ra lỗi là cao nhất?
- Chi phí (thời gian, công sức) ít nhất?

Mục tiêu trước mắt của kiểm thử phần mềm là tạo ra các ca kiểm thử để tìm ra lỗi của phần mềm.

Mục đích cuối cùng của kiểm thử phần mềm nhằm có một chương trình tốt, chi phí ít.

Glen Myers phát biểu một số quy tắc giống như mục đích kiểm thử:

① Kiểm thử là một tiến trình thực hiện một chương trình với ý định tìm ra lỗi.

②Một ca kiểm thử là một trường hợp kiểm thử có xác suất cao để tìm ra lỗi.

③Việc kiểm thử thành công là việc kiểm thử làm lộ ra một lỗi còn chưa được phát hiện.

Các mục đích trên dẫn đến một sự thay đổi lớn trong quan điểm.Chúng đi ngược lại quan điểm thông thường là một phép kiểm thử thành công là kiểm thử không tìm ra lỗi nào. Mục đích của chúng ta là thiết kế các ca kiểm thử để làm lộ ra một cách có hệ thống những lớp lỗi khác nhau và làm như vậy với một số lượng thời gian và công sức ít nhất.

Nếu kiểm thử được tiến hành thành công, thì nó sẽ làm lộ ra những lỗi trong phần mềm. Việc kiểm thử phần mềm làm việc theo đặc tả nên các yêu cầu hiệu năng dường như là được đáp ứng. Bên cạnh đó, dữ liệu thu thập được khi việc kiểm thử tiến hành đưa ra một chỉ dẫn tốt về độ tin cậy phần mềm và một chỉ dẫn nào đó về phẩm chất phần mềm với tư cách toàn cục.

Có một điều mà kiểm thử không thể làm được: Kiểm thử không thể chứng minh được việc không có khiếm khuyết, nó chỉ có thể chứng minh được khiếm khuyết phần mềm hiện hữu.

Khi kiểm thử, người ta đưa ra những khái niệm về ca kiểm thử “tốt” và “thắng lợi”:

- Ca kiểm thử tốt là ca kiểm thử có xác suất cao tìm ra 1 lỗi.

- Ca kiểm thử thắng lợi là ca kiểm thử làm lộ ra ít nhất một lỗi.

Vấn đề đặt ra ở chỗ nếu không tìm được lỗi nào thì có thể kết luận phần mềm hoàn hảo? Câu trả lời chung là chưa hẳn như vậy.

Kiểm thử có nhiều lợi ích, trong đó phải kể đến các lợi ích quan trọng:

- Ca kiểm thử thắng lợi làm lộ ra khiếm khuyết
- Kiểm thử mang lại các lợi ích phụ là thuyết minh:
  + Chức năng tương ứng với đặc tả,
  + Thực thi phù hợp yêu cầu và đặc tả,
  + Cung cấp các chỉ số tin cậy và chất lượng.

Tuy kiểm thử có nhiều lợi ích như trên nhưng chưa thể khẳng định phần mềm không còn khiếm khuyết.

1.1.2. Mục đích của kiểm thử phần mềm

Cũng giống như các sản phẩm máy móc và các hệ thống vật lý, mục đích của kiểm thử phần mềm là để đảm bảo hệ thống phần mềm có thể làm việc tốt như mong muốn khi chúng được đem ra thị trường tới tay khách hàng và người sử dụng. Cách thường dùng để đưa ra những kiểm chứng về chất lượng cho sản phẩm là đưa sản phẩm vào "chạy thử" hay được kiểm tra trong phòng thí nghiệm trước khi phân phối sản phẩm ra thị trường. Trong ngành Công Nghệ Phần Mềm, các sản phẩm phần mềm được kiểm tra, chạy thử được gọi chung là kiểm thử phần mềm.

Mục đích của việc kiểm thử phần mềm là gì? Kiểm thử phần mềm nhằm vào hai mục đích chính là: Đưa ra những chứng nhận về chất lượng và phát hiện sửa lỗi phần mềm.

Quy trình kiểm thử phần mềm

Những hành động chính của kiểm thử phần mềm là:

+ Chuẩn bị và lập kế hoạch kiểm thử: Đặt ra các mục tiêu cụ thể cho việc kiểm thử, chọn chiến lược kiểm thử, chuẩn bị các trường hợp kiểm thử cụ thể và các thủ tục kiểm thử.

+ Thực thi: Tiến hành kiểm thử theo kế hoạch, quan sát và thu thập các kết quả.

+ Phân tích và theo dõi: Trên những kết quả thu được, phân tích để tìm lỗi. Nếu một lỗi nào đó xuất hiện thì đưa ra giải pháp sửa đổi, sửa đổi và tiếp tục theo dõi tới khi lỗi đó được sửa.

Điểm quan trọng trong thực thi kiểm thử đó là tránh để lỗi từ một trường hợp kiểm thử ảnh hưởng đến các trường hợp kiểm thử khác.

1.1.3.Chiến lược kiểm thử phần mềm

1.1.3.1.Khái niệm chiến lược kiểm thử

Chiến lược kiểm thử là sự tích hợp các kỹ thuật thiết kế ca kiểm thử tạo thành một dãy các bước nhằm hướng dẫn quá trình kiểm thử phần mềm thành công.

Chiến lược kiểm thử được đặt ra với mục tiêu nhằm phác thảo một lộ trình để:

- Nhà phát triển tổ chức việc bảo đảm chất lượng bằng kiểm thử,
- Khách hàng hiểu được công sức, thời gian và nguồn lực cần cho kiểm thử.

Chiến lược kiểm thử cần phải đạt những yếu cầu sau:

- Tích hợp được các khâu như lập kế hoạch, thiết kế ca kiểm thử, tiến hành kiểm thử, thu thập và đánh giá các thong tin kết quả.
- Đủ mềm dẻo để cổ vũ óc sáng tạo, đáp ứng được nhu cầu khách hàng
- Thích ứng với mức kiểm thử cụ thể
- Đáp ứng các đối tượng quan tâm khác nhau.

Kiểm thử là một tập hợp những hoạt động mà có thể được lập kế hoạch trước và tiến hành một cách có hệ thống. Một tập các bước mà trong đó chúng ta có thể vận dụng những kỹ thuật thiết kế ca kiểm thử và phương pháp kiểm thử có những đặc trưng mang tính *"khuôn mẫu"*:

- Bắt đầu ở mức mô-đun và tiếp tục cho đến khi tích hợp ở mức hệ thống trọn vẹn.
- Các kỹ thuật kiểm thử khác nhau là thích hợp cho những thời điểm khác nhau.
- Được cả người phát triển và nhóm kiểm thử độc lập cùng tiến hành.
- Kiểm thử đi trước gỡ lỗi, song việc gỡ lỗi phải thích ứng với từng chiến lược kiểm thử.

Chiến lược cần thích ứng với từng mức kiểm thử và phải đưa ra hướng dẫn cho người thực hành và một tập các cột mốc cho người quản lý. Có hai mức kiểm thử:

- Kiểm thử mức thấp: thẩm định từng đoạn mã nguồn xem có tương ứng và thực thi đúng đắn hay không?
- Kiểm thử mức cao: thẩm định và xác minh các chức năng hệ thống chủ yếu có đúng đặc tả và đáp ứng yêu cầu của khách hàng hay không?

Mỗi chiến lược đáp ứng được yêu cầu cần quan tâm:

- Có các hướng dẫn cho người thực hiện tiến hành kiểm thử.
- Có các cột mốc cho các nhà quản lý kiểm soát hoạt động bảo đảm chất lượng.
- Có thước đo để đo và nhận ra các vấn đề càng sớm càng tốt.
- Khách hàng có thể nhận biết được quá trình kiểm thử.

Việc kiểm thử cung cấp một thành lũy cuối cùng để có thể thẩm định về chất lượng và có thể phát hiện ra lỗi.

Có một số quan điểm sai lầm:

- Người phát triển không nên tham gia kiểm thử.
- Cho phép người lạ kiểm thử một cách thô bạo.
- Người kiểm thử chỉ quan tâm khi kiểm thử bắt đầu.

Nên xuất phát từ thực tiễn mà phân công trách nhiệm thử:

- Người phát triển chịu trách nhiệm kiểm thử đơn vị do mình phát triển để bảo đảm thực hiện theo đúng thiết kế, có thể tham gia kiểm thử tích hợp; không khoán trắng chương trình cho người kiểm thử mà phải cùng làm việc với người kiểm thử xuyên suốt dự án.
- Nhóm kiểm thử độc lập bắt đầu làm việc khi các khoản mục cấu trúc phần mềm đã đầy đủ, giúp gỡ bỏ những thành kiến: "người xây dựng không thể kiểm thử tốt sản phẩm", gỡ bỏ mâu thuẫn giữa những người tham gia; đánh giá công sức người phát triển bỏ ra tìm lỗi; tạo ra báo cáo đầy đủ cho tổ chức bảo đảm chất lượng phần mềm.

1.1.3.2.Mô hình chiến lược tổng thể

Về mặt kỹ nghệ, việc kiểm thử gồm một số bước được thực hiện tuần tự. Ban đầu, việc kiểm thử tập trung vào từng mô-đun riêng biệt bảo đảm nó ban hành đúng đắn như một đơn vị. Do đó mới có tên kiểm thử đơn vị. Kiểm thử đơn vị dùng rất nhiều các kỹ thuật kiểm thử hộp trắng, kiểm soát các đường đặc biệt trong cấu trúc điều khiển của một lớp mô-đun nhằm phát hiện tối đa các lỗi. Mặt khác, các mô-đun phải được lắp ghép hay tích hợp lại để tạo nên phần mềm hoàn chỉnh.

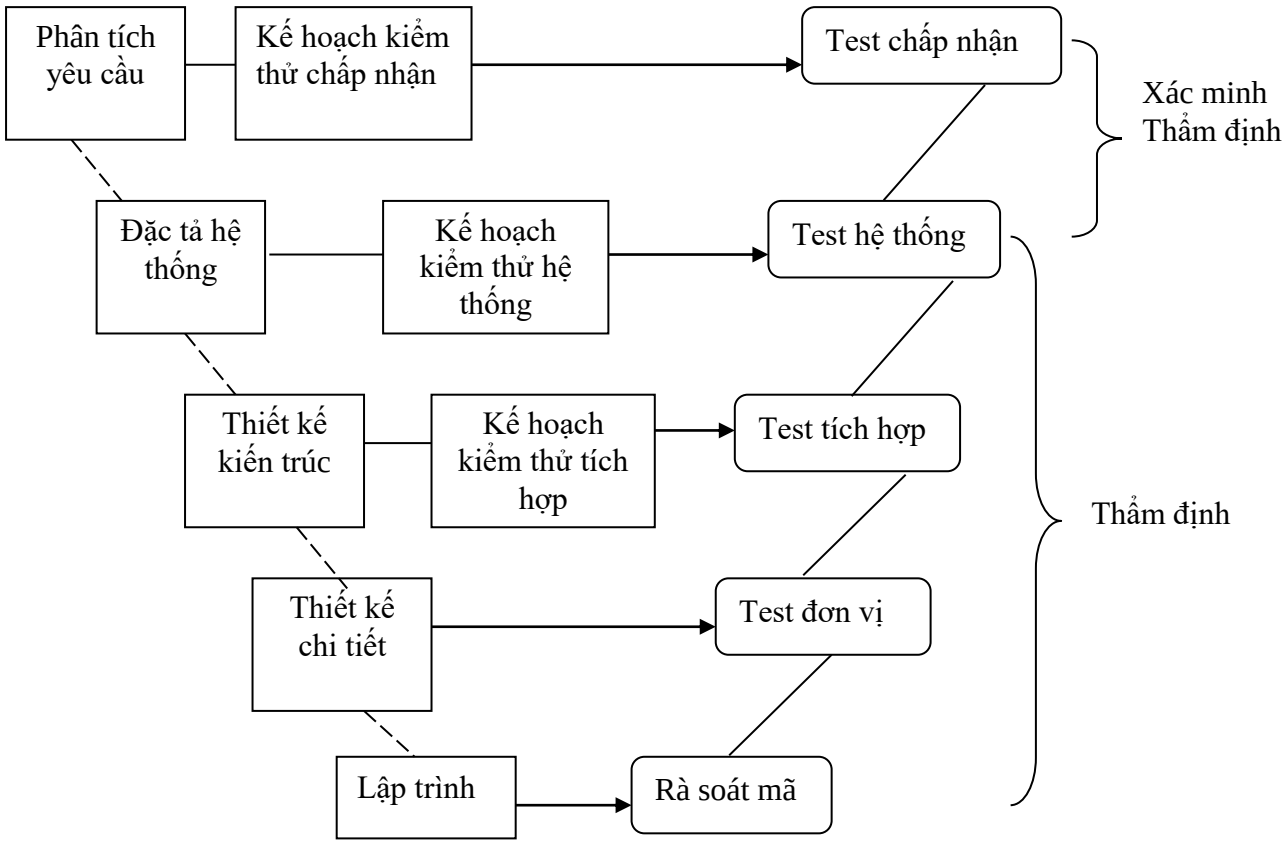
Việc kiểm thử tích hợp có liên quan đến thẩm định và xây dựng chương trình. Các kỹ thuật thiết kế kiểm thử hộp đen được dùng trong hầu hết quá trình tích hợp, mặc dù các kiểm thử hộp trắng cũng có thể được dùng để bao quát đa số các đường điều khiển. Sau khi phần mềm đã được dùng tích hợp (được xây dựng), một tập hợp các phép kiểm thử sẽ được tiến hành. Các tiêu chuẩn hợp lệ (được thiết lập trong phân tích yêu cầu) cũng phải được kiểm thử.Việc kiểm thử hợp lệ được tiến hành nhằm bảo đảm phần mềm đáp ứng đầy đủ các yêu cầu chức năng. Các kỹ thuật kiểm thử hộp đen được dùng chủ yếu trong kiểm thử hợp lệ.

Kiểm thử hệ thống nằm trong khung cảnh rộng hơn của kỹ nghệ hệ thống máy tính. Khi làm hợp lệ, phần mềm phải được tổ hợp với các phần tử

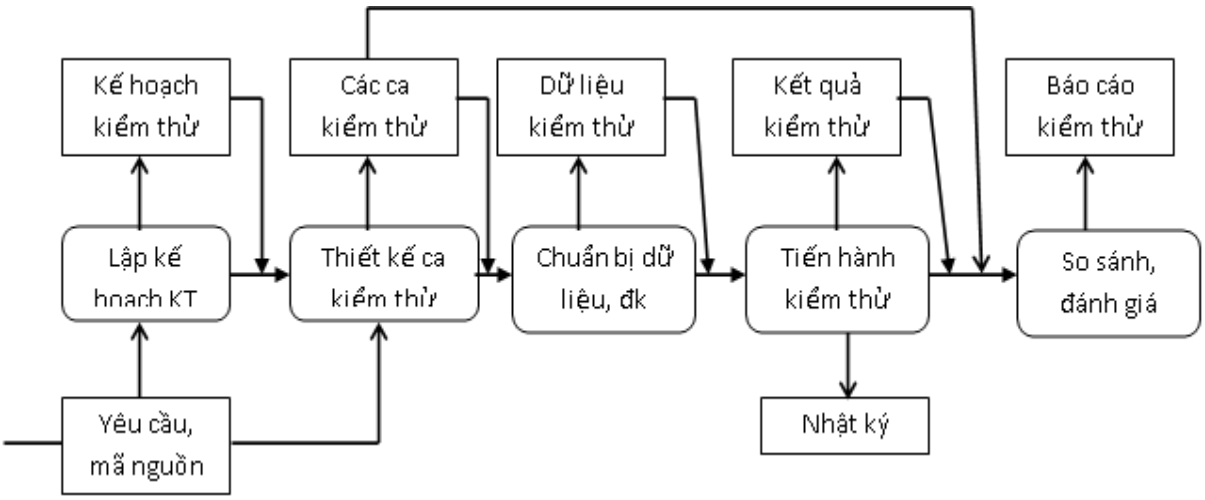
hệ thống khác (như phần cứng, con người, CSDL).Vì vậy, kiểm thử hệ thống là rất quan trọng.

(Trích trang 46. Kỹ thuật phần mềm - Lê Văn Phùng - Nhà xuất bản thông tin truyền thông năm 2010)

Cuối cùng, *kiểm thử chấp nhận* sẽ thẩm định lại rằng tất cả các thành phần có phối khớp với nhau không, cũng như chức năng hay độ hoàn thiện của hệ thống có đạt được không.



Hình 1.1.Mô hình chiến lược kiểm thử tổng thể



Hình 1.2. Mô hình tiến trình thực hiện kiểm thử

1.1.3.3. Một số chiến lược kiểm thử khác

Ngoài chiến lược kiểm thử tổng thể, người ta còn tiến hành một loạt các chiến lược kiểm thử bổ trợ khác như:

- Kiểm thử hệ thời gian thực
- Kiểm thử Alpha và Beta
- Kiểm thử so sánh.

1. Chiến lược kiểm thử hệ thời gian thực

Hệ thời gian thực là hệ thống đáp ứng đúng, chính xác các sự kiện của môi trường.

Kiểm thử hệ thống thời gian thực là rất khó. Những người thiết kế ca kiểm thử không chỉ phải xem xét các trường hợp kiểm thử hộp đen và hộp trắng mà còn phải xem xét cả việc chia thời gian cho dữ liệu cà cơ chế song song cho các nhiệm vụ (tiến trình) giải quyết dữ liệu đó.Trong nhiều tình huống, trạng thái của hệ thống cũng có thể dẫn tới lỗi.

Mối quan hệ mật thiết giữa phần mềm thời gian thực và môi trường phần cứng của nó cũng có thể gây ra các vấn đề cho kiểm thử.Việc kiểm thử

phần mềm phải xem xét tác động của các lỗi phần cứng đến xử lý phần mềm.những lỗi như thế rất khó mô phỏng sát thực tế.

Để khắc phục khó khăn trên, chiến lược kiểm thử được vạch ra theo 4 bước thực hiện:

Bước 1: Kiểm thử tác vụ

Kiểm thử từng tác vụ một cách độc lập với nhau (bằng kiểm thử hộp trắng và hộp đen).

Kiểm thử tác vụ cho phép phát hiện sai về logic và chức năng nhưng không phát hiện sai về thời gian và ứng xử.

Bước 2: Kiểm thử ứng xử

Trước hết sử dụng công cụ CASE tạo mô hình hệ thống để mô phỏng ứng xử, xem ứng xử như hậu quả của sự kiện tác động từ ngoài vào.

Sau đó dựng kết quả hoạt động phân tích này để thiết kế các ca kiểm thử (tương tự kỹ thuật đồ thị nhân quả).

Tiếp theo, phân lớp sự kiện (phân hoạch tương đương). Kiểm thử từng lớp sự kiện và nhận ứng xử của hệ thi hành để phát hiện các sai do xử lý đáp ứng các sự kiện đó.

Cuối cùng, kiểm thử mọi lớp sự kiện. Các sự kiện được đưa vào trong hệ thống theo thứ tự ngẫu nhiên và với tần xuất ngẫu nhiên nhằm phát hiện các lỗi ứng xử.

Bước 3: Kiểm thử liên tác

Kiểm thử liên tác là kiểm thử để tìm các sai liên quan đến thời gian đáp ứng do không đồng bộ:

- Các tác vụ không đồng bộ khi liên tác với các tác vụ khác. Vì thế cần kiểm thử với nhịp điệu dữ liệu và mức tải với các xử lý khác nhau.

- Các tác vụ không đồng bộ do giao tiếp phụ thuộc hàng đợi thông điệp hoặc truy nhập kho dữ liệu cũng được thử để bộc lộ các lỗi về quy mô dữ liệu.

Bước 4: Kiểm thử toàn hệ thống

Tích hợp phần cứng và phần mềm nhằm tìm lỗi trên các phương diện:

- Giao diện (giữa phần cứng và phần mềm)
- Môi trường (tác động từ môi trường, các sự kiện).
- Các ngắt (các loại ngắt và các xử lý xảy ra như hậu quả của ngắt).

2. Kiểm thử Alpha và Beta

Kiểm thử alpha (alpha testing) và kiểm thử beta (beta testing) đều do người dùng thực hiện và đều được thực hiện trong môi trường thực.

Người phát triển không thể nào lường hết được khách hàng sẽ dùng chương trình như thế nào. Các hướng dẫn sử dụng có thể bị hiểu sai, việc tổ hợp dữ liệu lạ lại có thể hay được dùng, cái ra dường như rõ ràng với người kiểm thử, nhưng có thể lại khó hiểu đối với người dùng.

+ Kiểm thử Alpha

Kiểm thử Alpha được khách hàng tiến hành tại cơ quan của người phát triển. Phần mềm được dùng theo sự sắp đặt tự nhiên với người phát triển (nhìn qua vai) người dùng và ghi lại các lỗi khi sử dụng. Kiểm thử Alpha còn có tên khác là kiểm thử "sau lưng" và được tiến hành trong một môi trường có kiểm soát.

Dữ liệu cho kiểm thử Alpha là dữ liệu mô phỏng.

+ Kiểm thử Beta

Kiểm thử Beta được người dùng cuối tiến hành tại một hay nhiều cơ quan khách hàng. Không giống kiểm thử Alpha, người phát triển thường không có mặt. Do đókiểm thử Beta là việc áp dụng "sống" của phần mềm trong một môi trường mà người phát triển không thể nào kiểm soát được. Khách hàng ghi lại tất cả các vấn đề (thực hay tưởng tượng) gặp phải trong khi kiểm thử Beta và báo cáo lại những vấn đề đó cho người phát triển trong những khoảng thời gian đều đặn. Xem như một kết quả của vấn đề được báo cáo trong kiểm thử Beta, người phát triển tiến hành sửa đổi rồi chuẩn bị đưa ra sản phẩm phần mềm cho toàn bộ khách hàng.

Trường hợp kiểm thử Alpha và Beta cho 1 khách:

- Khi các phần mềm dành cho một người đặt hàng, thì hoạt động kiểm thử chỉ cần một khách hàng tiến hành thẩm định mọi yêu cầu.

- Kiểm thử này do người sử dụng cuối cùng thực hiện, không phải là người đặt hàng.

- Kiểm thử chấp nhận có thể tiến hành vài tuần hoặc vài tháng một lần, nhờ đó mà bộc lộ được các lỗi tích lũy làm suy giảm hệ thống theo thời gian.

Trường hợp kiểm thử Alpha và Beta cho n khách:

Với phần mềm dành cho nhiều khách hàng, thì kiểm thử chấp nhận bằng một khách hàng là không thực tế.Quá trình kiểm thử Alpha và Beta cho nhiều người tiến hành là bắt buộc.

3. Kiểm thử so sánh

Có một số tình huống (như điều khiển máy bay, điều khiển nhà máy năng lượng hạt nhân) mà trong đó độ tin cậy của phần mềm là yếu tố hàng đầu.Trong những ứng dụng như vậy, phần cứng và phần mềm thường được yêu cầu tối thiểu hóa khả năng lỗi. Khi phần mềm được phát triển, nhóm kỹ nghệ phần mềm tách biệt sẽ phát triển những bản độc lập ứng dụng bằng cách dùng cùng đặc tả.Trong những tình huống như thế, mỗi bản có thể được kiểm thử với cùng dữ liệu để đảm bảo rằng tất cả đều cho kết quả giống nhau.

Các nhà nghiên cứu đã gợi ý rằng các bản phần mềm độc lập được phát triển. Những bản độc lập này tạo nền cho kỹ thuật kiểm thử hộp đen được gọi là kiểm thử so sánh (comparision testing) hay kiểm thử dựa vào nhau (back-to-back testing).

Khi tạo ra nhiều cài đặt cho cùng một đặc tả, các ca kiểm thử được thiết kế để dùng những kỹ thuật hộp đen khác (như phân hoạch tương đương) được xem như từng bản đầu vào của phần mềm. Nếu kết quả ra của mỗi bản là giống nhau thì người ta giả thiết rằng tất cả các cài đặt đều đúng. Nếu kết quả ra là khác nhau thì từng ứng dụng sẽ được nghiên cứu lại để xác định liệu một khiếm khuyết trong một hay nhiều bản có phải là nguyên nhân sinh ra sự khác

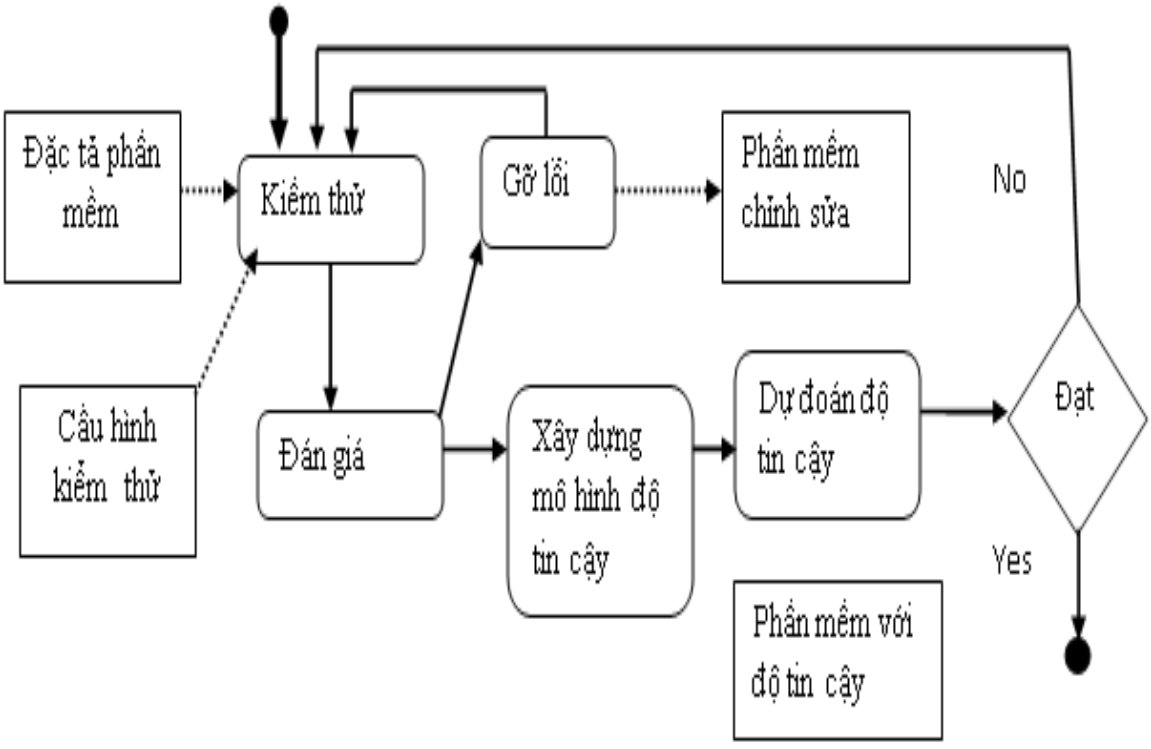
biệt hay không.Trong phần lớn các trường hợp, việc so sánh các kết quả ra có thể được thực hiện tự động.

Việc kiểm thử so sánh không đơn giản. Nếu đặc tả mà từ đó tất cả các phiên bản đã được xây dựng ra bị lỗi thì mọi bản đều có thể phản ánh lỗi đó.Mặt khác, nếu từng bản độc lập lại tạo ra kết quả giống nhau, nhưng không đúng, thì việc kiểm thử điều kiện sẽ không phát hiện được lỗi.

Trong quá trình kiểm thử so sánh, cần chú ý:

- Khi triển khai nhiều phiên bản phần mềm từ cùng một đặc tả, kiểm thử hộp đen cho các sản phẩm này được thực hiện với cùng ca kiểm thử và cùng các dữ liệu vào.

- Khi so sánh các kết quả thu được, nếu có khác biệt nghĩa là có sai trong một sản phẩm nào đó.



Hình 1.3. Sơ đồ thông tin toàn bộ tiến trình kiểm thử

(Trang 75. sách Kiểm thử và đảm bảo chất lượng phần mềm. của Thạc Bình Cường -NXB Bách khoa Hà Nội 2011)

1.1.4. Các phương pháp và kỹ thuật kiểm thử

Thiết kế kiểm thử cho phần mềm và các sản phẩm kỹ nghệ khác có thể có tính thách đố như việc thiết kế ban đầu cho chính bản thân sản phẩm. Người kỹ sư phần mềm thường kiểm thử hần mềm sau khi lập trình xong. Các ca kiểm thử đòi hỏi vừa đúng lại vừa đủ. Mặt khác, kiểm thử cần được thiết kế sao cho có khả năng cao nhất để tìm ra lỗi với thời gian và công sức ít nhất.

1.1.4.1.Một số phương pháp kiểm thử

Có rất nhiều phương pháp thiết kế trường hợp kiểm thử cho phần mềm. Những phương pháp này cung cấp cho người phát triển một cách tiếp cận hệ thống tới việc kiểm thử. Quan trọng hơn, những phương pháp này đưa ra một cơ chế có thể giúp đảm bảo tính đầy đủ của kiểm thử và đưa ra khả năng cao nhất để phát hiện ra lỗi trong phần mềm.

Bất kỳ sản phẩm kỹ nghệ nào đều có thể được kiểm thử một trong hai cách:

Cách 1: Kiểm thử chức năng/ hộp đen: cho dữ liệu đầu vào đúng/ sai, kiểm tra đầu ra đúng/sai, tức là kiểm thử xem từng chức năng có vận hành đúng không, không quan tâm đến cấu trúc bên trong của chức năng đó.

Cách 2: Kiểm thức cấu trúc/ hộp trắng: không những quan tâm đến mối quan hệ đầu vào và đầu ra của chức năng đó mà còn quan tâm đến cấu trúc bên trong, quan tâm chi tiết đến từng đầu vào đầu ra của các thành phần cấu thành trong đó và cả sự ăn khớp giữa chúng nữa, tức là đảm bảo rằng, sự vận hành bên trong thực hiện đúng theo đặc tả và tất cả các thành phần bên trong đều được quan tâm và được kiểm tra một cách chi tiết.

Đối với phần mềm máy tính, kiểm thử hộp đen biểu thị việc kiểm thử được tiến hành tại giao diện phần mềm. Mặc dầu chúng được thiết kế để phát hiện ra lỗi, kiểm thử hộp đen được dùng để thể hiện rằng các chức năng phần mềm đã vận hành, cái vào được chấp nhận đúng, và cái ra được tạo ra đúng, tính toàn vẹn thông tin ngoài (như tệp dữ liệu) là được duy trì. Phép kiểm thử

hộp đen xem xét một khía cạnh của hệ thống mà ít để ý tới cấu trúc logic bên trong của phần mềm.

Kiểm thử hộp trắng được hướng tới việc xem xét kỹ về chi tiết thủ tục. Các đường logic đi qua phần mềm được kiểm thử bằng cách đưa ra các trường hợp kiểm thử, vốn thực hiện trên một tập xác định các điều kiện và/ hoặc chu trình. "Trạng thái của chương trình" có thể được xem xét tại nhiều điểm khác nhau để xác định liệu trạng thái dự kiến hay khẳng định có tương ứng với trạng thái thực tại hay không.

Thoáng nhìn dường như là việc kiểm thử hộp trắng rất thấu đáo sẽ dẫn tới "chương trình chính xác 100%". Mọi điều ta cần là xác định tất cả các con đường logic, xây dựng các trường hợp kiểm thử để vét hết logic chương trình. Nhưng việc kiểm thử vét cạn lại có một số vấn đề. Ngay cả với những chương trình nhỏ, số các đường logic cũng có thể rất lớn.

Tuy nhiên, việc kiểm thử hộp trắng không nên bị bỏ qua không xét vì không thực tế. Người ta có thể chọn ra một số có giới hạn đường logic quan trọng và thực hiện chúng. Có thể thăm dò cấu trúc dữ liệu quan trọng về tính hợp lệ. Các thuộc tính của cả kiểm thử hộp đen lẫn hộp trắng có thể được tổ hợp để đưa ra một cách tiếp cận để làm hợp lệ cho giao diện phần mềm và bảo đảm có chọn lựa rằng công việc bên trong của phần mềm là đúng đắn.

1.1.4.2.Các kỹ thuật kiểm thử

Mục tiêu của kiểm thử là phải thiết kế các trường hợp kiểm thử có khả năng cao nhất trong việc phát hiện nhiều lỗi với thời gian và công sức tối thiểu. Do đó, có thể chia các kỹ thuật kiểm thử thành hai loại:

- Kỹ thuật kiểm thử hộp đen (Black – box testing) hay còn gọi là kỹ thuật kiểm thử chức năng.

- Kỹ thuật kiểm thử hộp trắng (white – box testing) hay còn gọi là kỹ thuật kiểm thử cấu trúc (structural testing).

1. Kỹ thuật kiểm thử hộp đen (Black – box testing)

Hay còn gọi là kiểm thử chức năng kiểm tra sự hoạt động đúng của các chức năng giao tiếp với bên ngoài của phần mềm thông qua sự quan sát chúng trong quá trình thực thi. Do phần mềm được coi như một hộp đen, với các luồng dữ liệu vào ra và một số đặc tính có thể quan sát được khác nên loại kiểm thử này còn được gọi là kiểm thử hộp đen.

Dạng đơn giản nhất của kiểm thử hộp đen là bắt đầu chạy chương trình và quan sát để tìm ra những hành vi, những hoạt động mong muốn và không mong muốn. Dạng kiểm thử này được gọi là kiểm thử vô thể thức (ad hoc testing).

Một dạng phổ biến khác của kiểm thử hộp đen là sử dụng một danh sách kiểm tra, danh sách kiểm tra là một danh sách các chức năng ngoài với các thông tin trạng thái mong muốn và thông tin vào ra. Thông tin vào ở đây bao gồm tất cả những hành động, công cụ và tài nguyên được cung cấp cho chương trình tại thời điểm trước lúc chạy chương trình hoặc tại bất kì thời điểm nào trong lúc chương trình chạy. Tương tự thông tin ra là tất các những hành vi và kết quả của chương trình tại thời điểm chương trình kết thúc hay tại bất kì thời điểm nào trong lúc chương trình đang chạy. Ví dụ thông tin vào cho một chương trình tính toán là các con số từ bàn phím, hành động là chia hai số đó cho nhau, thông tin ra có thể là kết quả đúng hay có thể là một thông báo lỗi – trong trường hợp chia cho không.

Kiểm thử hộp đen có thể tuân theo quy trình kiểm thử ở trên, bao gồm 3 hoạt động chính là lên kế hoạch, thực thi, phân tích và theo dõi.

- Lên kế hoạch tập trung vào việc xác định các chức năng ngoài, thông tin đầu vào, thông tin đầu ra và những trạng thái mong muốn của người sử dụng.Ví dụ: Một chương trình dịch; thông tin đầu vào là mã nguồn chương trình cần dịch, thông tin đầu ra là các đối tượng hay mã thực thi, trạng thái mong mà người sử dụng là chương trình sẽ kết thúc trong một khoảng thời gian giới hạn, hay yêu cầu nhiều hơn đó là với một chương trình đầu vào không hợp lệ, các đối tượng hay mã thực thi sẽ không được sinh ra đồng thời

đưa ra các thông báo lỗi. Trong ví dụ này nếu đầu vào là một tập các chương trình thì ta sẽ có một tập các trường hợp kiểm thử.

- Thực thi kiểm thử tập trung vào việc quan sát cách hoạt động của chương trình, đảm bảo các trường hợp kiểm thử thực hiện đúng thứ tự, ghi nhận kết quả để phân tích. Nếu sự quan sát không tìm thấy các lỗi trực tiếp, nhưng thông tin ghi nhận được vẫn cần được lưu lại cho việc phân tích về sau. Như trong ví dụ trên, thông tin đầu ra, những thông tin về quá trình dịch cũng như các tham số cấu hình cần được lưu lại.

- Thông tin thu được trong quá trình thực thi kiểm thử được so sánh với những thông tin chuẩn để xác định một chức năng nào đó thỏa mãn hay không thỏa mãn yêu cầu. Một chức năng nào đó không thỏa mãn yêu cầu sẽ được theo dõi cô lập, phát hiện và sửa chữa.

Kiểm thử hộp đen hay còn gọi kiểm thử hướng dữ liệu (data driven) hay là kiểm thử hướng vào/ra (input/output driven).

Trong kỹ thuật này, người kiểm thử xem phần mềm như là một hộp đen. Người kiểm thử hoàn toàn không quan tâm đến cấu trúc và hành vi bên trong của chương trình. Người kiểm thử chỉ cần quan tâm đến việc tìm các hiện tượng mà phần mềm không hành xử theo đúng đặc tả của nó. Do đó, dữ liệu kiểm thử sẽ xuất phát từ đặc tả.

2. Kỹ thuật kiểm thử hộp trắng (white – box testing)

Hay còn gọi là kiểm thử cấu trúc kiểm tra sự cài đặt đúng của những đơn vị bên trong chương trình phần mềm như các câu lệnh, các cấu trúc dữ liệu, các khối ... và quan hệ giữa chúng. Việc kiểm tra được thực hiện thông qua việc quan sát kết quả của sự thực thi các đơn vị đó và mối quan hệ với các đơn vị định trước. Phần mềm được xem như là một chiếc hộp trắng (thực chất là hộp trong suốt), các đơn vị bên trong chương trình được nhìn thấy cùng với các mối tương tác giữa chúng.

Kiểm thử cấu trúc được hỗ trợ bởi một số công cụ phần mềm, dạng đơn giản nhất là kiểm thử mọi dòng lệnh thông qua một số công cụ gỡ lỗi,

(debugging tool hay debugger) giúp dò vết khi thực hiện chương trình. Do đó người kiểm thử có thể biết được khi một lệnh được thực thi, kết quả của nó có như mong muốn hay không. Ưu điểm của cách kiểm thử này là khi phát hiện được lỗi đồng thời cũng xác định được lỗi ngay, tuy nhiên nó yêu cầu người kiểm thử phải thông thạo mã nguồn và các lỗi về sự thiếu sót, sự sai trong thiết kế rất khó được phát hiện. Kiểm thử cấu trúc nên được thực hiện bởi chính những người viết chương trình đó thì việc phát hiện và sửa lỗi mới dễ dàng.

Kiểm thử cấu trúc cũng có thể tuân theo quy trình kiểm thử phần mềm chung, tuy nhiên do yêu cầu về sự thông thạo mã nguồn chương trình, bao quát toàn bộ sự cài đặt của hệ thống – điều này là rất khó, nên kiểm thử cấu trúc thường được giới hạn với quy mô nhỏ. Đối với các sản phẩm phần mềm nhỏ là không cần thiết phải có một quy trình đầy đủ cho việc kiểm thử, phát hiện và sửa lỗi. Đối với các chương trình lớn, việc kiểm thử cấu trúc được thực hiện trong một framework hoàn chỉnh do vậy việc lập kế hoạch kiểm thử giữ vai trò kém quan trọng hơn so với kiểm thử chức năng. Thêm vào đó việc phát hiện và sửa lỗi cũng dễ dàng do có sự liên kết chặt chẽ giữa chức năng của chương trình với các đơn vị chương trình, thêm vào đó là vai trò kép vừa là người kiểm thử vừa là người viết chương trình. Do vậy kiểm thử cấu trúc không đòi hỏi một quy trình chặt chẽ như kiểm thử chức năng.

Kiểm thử hộp trắng hay còn gọi là kiểm thử hướng logic, cho phép kiểm tra cấu trúc bên trong của phần mềm với mục đích bảo đảm rằng tất cả các câu lệnh và điều kiện sẽ được thực hiện ít nhất một lần. Người kiểm thử truy nhập vào mã nguồn chương trình và có thể kiểm tra nó, lấy đó làm cơ sở để hỗ trợ việc kiểm thử.

3. So sánh kiểm thử hộp đen và kiểm thử hộp trắng.

Kiểm thử chức năng coi các đối tượng kiểm thử như một hộp đen, chú trọng vào việc kiểm tra các quan hệ vào ra và những chức năng giao diện bên ngoài. Trong đó kiểm thử cấu trúc coi các đối tượng như một hộp trong suốt,

các thành phần cài đặt bên trong được nhìn thấy và kiểm tra. Dưới đây là một số đặc điểm so sánh:

- Đối tượng: Kiểm thử cấu trúc được sử dụng cho các đối tượng kiểm thử nhỏ như các chương trình nhỏ hay các đơn vị chương trình nhỏ của một chương trình lớn. Còn kiểm thử chức năng thích hợp hơn cho các hệ thống phần mềm lớn hay các thành phần quan trọng của chúng.

- Thời gian: Kiểm thử cấu trúc được sử dụng nhiều trong giai đoạn đầu như kiểm thử đơn vị và kiểm thử thành phần. Trong đó kiểm thử chức năng được dùng trong các giai đoạn sau như kiểm thử hệ thống và kiểm thử chấp thuận (acceptance testing).

- Kiểu lỗi: Trong kiểm thử chức năng những lỗi xuất hiện liên quan tới các chức năng bên ngoài. Những lỗi xuất hiện trong kiểm thử cấu trúc liên quan tới sự cài đặt bên trong.

- Phát hiện và sửa lỗi: Những lỗi được phát hiện thông qua kiểm thử cấu trúc dễ dàng được sửa hơn so với kiểm thử chức năng bởi vì có sự liên quan trực tiếp giữa những lỗi quan sát được, các đơn vị chương trình và sự cài đặt chi tiết. Tuy nhiên kiểm thử chức năng rất khó phát hiện kiểu lỗi thiếu sót và lỗi trong thiết kế - những lỗi mà có thể phát hiện bởi kiểm thử chức năng. Kiểm thử chức năng hiệu quả trong việc phát hiện và sửa các lỗi về giao diện và tương tác, còn kiểm thử cấu trúc hiệu quả cho các vấn đề cục bộ trong một đơn vị nhỏ.

- Người kiểm thử: Người kiểm thử trong kiểm thử chức năng là các chuyên gia kiểm thử hay có thể là đơn vị thứ ba, còn đối với kiểm thử cấu trúc người kiểm thử là người phát triển chương trình.

1.2. Những nét chung nhất về ca kiểm thử

1.2.1.Khái niệm ca kiểm thử

Ca kiểm thử (test case) là một tình huống kiểm thử tương ứng với một mạch hoạt động của chương trình. Nó bao gồm một tập các giá trị đầu vào và một danh sách các kết quả đầu ra mong muốn và thực tế.

Mục tiêu thiết kế ca kiểm thử nhằm:

- Tìm ra nhiều lỗi nhất

- Với chi phí và thời gian ít nhất

Trong các thập kỷ 80-90 của thế kỷ XX, người ta đã nghiên cứu nhiều loại phương pháp thiết kế ca kiểm thử. Trong các phương pháp này, phương pháp thiết kế ca kiểm thử. Trong các phương pháp này, phương pháp thiết kế được chọn theo cơ chế:

- Bảo đảm tính đầy đủ (không sót phần nào)

- Cung cấp khả năng phát hiện lỗi nhiều nhất

Việc thiết kế 1 ca kiểm thử được đặt ra với lý do sau là chủ yếu:

- Số con đường logic/ mạch thực hiện trong chương trình là rất lớn

- Nhiều trạng thái dữ liệu khác nhau: số đại lượng, giá trị, sự thay đổi trong tiến trình, sự kết hợp giữa chúng.

Câu hỏi đặt ra là khi nào thì kiểm thử xong? Làm thế nào để biết rằng kiểm thử đã đủ? Về nguyên tắc:

- Không bao giờ kiểm thử được tất cả

- Vận hành chương trình là đang kiểm thử

- Kiểm thử tiếp tục khi chương trình còn hoạt động

Kỹ sư phần mềm cần các tiêu chuẩn nghiêm ngặt để xác định có cần phải tiếp tục kiểm thử không.

1.2.2.Vấn đề thiết kế ca kiểm thử

Thiết kế test – case trong kiểm thử phần mềm là quá trình xây dựng các phương pháp kiểm thử có thể phát hiện lỗi, sai sót, khuyết điểm của phần mềm để xây dựng phần mềm đạt tiêu chuẩn.

Thiết kế test-case giữ vai trò quan trọng trong việc nâng cao chất lượng phần mềm vì những lý do sau đây:

- Tạo ra các ca kiểm thử tốt nhất có khả năng phát hiện ra lỗi, sai sót của phần mềm một cách nhiều nhất.

- Tạo ra các ca kiểm thử có chi phí rẻ nhất, đồng thời tốn ít thời gian và công sức nhất.

Một trong những lý do quan trọng nhất trong kiểm thử chương trình là thiết kế và tạo ra các ca kiểm thử - các Test case có hiệu quả. Với những ràng buộc về thời gian và chi phí đã cho, thì vấn đề then chốt của kiểm thử trở thành: Tập con nào của tất cả ca kiểm thử có thể có khả năng tìm ra nhiều lỗi nhất?

Thông thường, phương pháp kém hiệu quả nhất là kiểm tra tất cả đầu vào ngẫu nhiên – quá trình kiểm thử một chương trình bằng việc chọn ngẫu nhiên một tập con các giá trị đầu vào có thể. Về mặt khả năng tìm ra nhiều lỗi nhất, tập hợp các ca kiểm thử được chọn ngẫu nhiên có rất ít cơ hội là tập hợp tối ưu hay gần tối ưu. Sau đây là một số phương pháp để chọn ra một tập dữ liệu kiểm thử một cách thông minh.

Để kiểm thử hộp đen và kiểm thử hộp trắng một cách thấu đáo là không thể. Do đó, một chiến lược kiểm thử hợp lý là chiến lược có thể kết hợp sức mạnh của cả hai phương pháp trên: Phát triển 1 cuộc kiểm thử nghiêm ngặt vừa bằng việc sử dụng các phương pháp thiết kế ca kiểm thử hướng hộp đen nào đó và sau đó bổ sung thêm những ca kiểm thử này bằng việc khảo sát tính logic của chương trình, sử dụng phương pháp hộp trắng.

Những chiến lược kết hợp đó bao gồm:

Hộp đen	***Hộp trắng***
1. Phân lớp tương đương	1. Bao phủ câu lệnh
2. Phân tích giá trị biên	2. Bao phủ quyết định
3. Đồ thị nguyên nhân – kết quả	3. Bao phủ điều kiện
4. Đoán lỗi	4. Bao phủ điều kiện – quyết định
	5. Bao phủ đa điều kiện.

Mỗi phương pháp có những ưu điểm cũng như khuyết điểm riêng, do đó để có được tập các ca kiểm thử tối ưu, chúng ta cần kết hợp hầu hết các phương pháp. Quy trình thiết kế các ca kiểm thử sẽ bắt đầu bằng việc phát

triển các ca kiểm thử sử dụng phương pháp hộp đen và sau đó phát triển bổ sung các ca kiểm thử cần thiết với phương pháp hộp trắng.

Khi nào kết thúc ca kiểm thử?

Câu hỏi "Khi nào kết thúc việc kiểm thử?" được phân thành hai dạng khác nhau:

- Đối với các chương trình nhỏ hoặc quy mô nhỏ, có thể hỏi "Khi nào kết thúc hành động kiểm thử?"

- Đối với các chương trình có quy mô lớn, có thể đặt câu hỏi "Khi nào thì kết thúc tất cả các hoạt động kiểm thử chính?". Do kiểm thử thường là khâu cuối cùng của quá trình phát triển phần mềm, trước khi phần mềm được phân phối đến người sử dụng, nên cũng có thể đặt câu hỏi tương đương "Khi nào thì kết thúc việc kiểm thử và phân phối sản phẩm?".

Có nhiều câu trả lời khác nhau cho những câu hỏi trên, mỗi câu trả lời hướng tới những kĩ thuật và hành động khác nhau. Khi không có một sự đánh giá chuẩn "formal assessment", thì quyết định dừng kiểm thử sản phẩm phần mềm dựa trên hai dạng chính:

- Tài nguyên: Quyết định có tiếp tục quá trình kiểm thử hay không dựa trên sự tiêu tốn tài nguyên, hai tiêu chuẩn trạng thái để dừng kiểm thử là: Vượt quá thời gian và tiêu tốn quá nhiều tiền.

- Hành động: Quyết định dừng kiểm thử khi đã hoàn thành tất cả các hoạt động kiểm thử theo như kế hoạch kiểm thử đã đề ra.

Trên phương diện tổng thể, kết thúc quá trình kiểm thử gắn với sự chuyển giao sản phẩm, đồng thời thể hiện mức độ chất lượng mà khách hàng hay người sử dụng mong đợi. Trên phương diện Công Nghệ Phần Mềm, quyết định dừng kiểm thử gắn với sự thỏa mãn các tiêu chuẩn về chất lượng và mục đích của dự án trong tổng thể quá trình phát triển phần mềm. Do vậy cách trực tiếp và rõ ràng nhất để đưa ra quyết định có dừng quá trình kiểm thử hay không đó là sử dụng những đánh giá, kiểm chứng tin cậy. Khi mà môi trường đánh giá sản phẩm giống với môi trường sử dụng thực của khách

hàng, thì kết quả đánh giá là đáng tin cậy nhất và đó cũng là kết quả của kiểm thử hướng sử dụng. Trong trường hợp với số lượng khách hàng lớn, tình huống và kịch bản sử dụng chương trình khác nhau việc thống kê hết là điều không thể thì đó chính là nội dung của kiểm thử thống kê hướng sử dụng.

Đối với các pha đầu của quá trình kiểm thử hay các tiêu chuẩn kết thúc kiểm thử liên quan tới các hành động kiểm thử cục bộ, những tiêu chuẩn tin cậy dựa trên những kịch bản sử dụng của khách hàng và tần suất sử dụng có thể không còn ý nghĩa nhiều. Ví dụ trong một số hệ thống phần mềm, một số thành phần không bao giờ được sử dụng trực tiếp bởi người sử dụng, và một số thành phần thì có tần suất sử dụng rất thấp. Do vậy việc lựa chọn các tiêu chuẩn khác là cần thiết. Những tiêu chuẩn mới này là các tiêu chuẩn bao phủ, nó bao hàm các tiêu chuẩn khác nhau, định nghĩa các kĩ thuật dùng trong tình huống kiểm thử và các thành phần khác có liên quan. Các kĩ thuật kiểm thử hướng tới các tiêu chuẩn này được gọi là các kĩ thuật kiểm thử hướng bao phủ.

Ngoài các ràng buộc về tài nguyên và giới hạn của con người, có hai loại tiêu chuẩn để kết thúc hành động kiểm thử đó là dựa trên thống kê sử dụng của người dùng để xác định các thành phần cũng như các yếu tô liên quan để kiểm thử- theo các tiêu chuẩn này có dạng kiểm thử thống kê hướng sử dụng. Tiêu chuẩn thứ hai bao gồm các tiêu chuẩn về mọi vấn đề liên quan tới đơn vị kiểm thử, xét tới mọi khả năng gây lỗi của chương trình – theo tiêu chuẩn này để kết thúc kiểm thử có các kĩ thuật kiểm thử hướng bao phủ. Trong phần tiếp theo sẽ trình bày khái quát và so sánh về hai dạng kiểm thử này.

CHƯƠNG 2
CÁC KỸ THUẬT THIẾT KẾ CA KIỂM THỬ

Một trong những lý do quan trọng nhất trong kiểm thử phần mềm là thiết kế và tạo ra các *ca kiểm thử (Test case)* có hiệu quả (ca kiểm thử tốt nhất có khả năng phát hiện ra lỗi, sai sót của phần mềm một cách nhiều nhất). Với những ràng buộc về thời gian và chi phí đã cho, thì vấn đề then chốt của kiểm thử là phải trả lời câu hỏi: *Tập con nào của tất cả ca kiểm thử có thể có khả năng tìm ra nhiều lỗi nhất?*

Thông thường, phương pháp kém hiệu quả nhất là kiểm tra tất cả đầu vào ngẫu nhiên – quá trình kiểm thử một chương trình bằng việc chọn ngẫu nhiên một tập con các giá trị đầu vào có thể. Về mặt khả năng tìm ra nhiều lỗi nhất, tập hợp các ca kiểm thử được chọn ngẫu nhiên có rất ít cơ hội là tập hợp tối ưu hay gần tối ưu. Sau đây là một số phương pháp để chọn ra một tập dữ liệu kiểm thử một cách thông minh.

Để kiểm thử hộp đen và kiểm thử hộp trắng một cách thấu đáo là không thể. Do đó, một chiến lược kiểm thử hợp lý là chiến lược có thể kết hợp sức mạnh của cả hai phương pháp trên: Phát triển một cuộc kiểm thử nghiêm ngặt vừa bằng việc sử dụng các phương pháp thiết kế ca kiểm thử hướng hộp đen nào đó và sau đó bổ sung thêm những ca kiểm thử này bằng việc khảo sát tính logic của chương trình, sử dụng phương pháp hộp trắng.

Những chiến lược kết hợp đó bao gồm:

Hộp đen	***Hộp trắng***
1. Phân lớp tương đương 2. Phân tích giá trị biên 3. Đồ thị nguyên nhân – kết quả 4. Đoán lỗi	1. Bao phủ câu lệnh 2. Bao phủ quyết định 3. Bao phủ điều kiện 4. Bao phủ điều kiện – quyết định 5. Bao phủ đa điều kiện.

Mỗi phương pháp có những ưu điểm cũng như khuyết điểm riêng, do đó để có được tập các ca kiểm thử tối ưu, chúng ta cần kết hợp hầu hết các

phương pháp. Quy trình thiết kế các ca kiểm thử sẽ bắt đầu bằng việc phát triển các ca kiểm thử sử dụng phương pháp hộp đen và sau đó phát triển bổ sung các ca kiểm thử cần thiết với phương pháp hộp trắng.

Kiểm thử hộp trắng có liên quan tới mức độ mà các ca kiểm thử thực hiện hay bao phủ tính logic (mã nguồn) của chương trình. Kiểm thử hộp trắng cơ bản là việc thực hiện mọi đường đi trong chương trình, nhưng việc kiểm thử đầy đủ đường đi là một mục đích không thực tế cho một chương trình với các vòng lặp.

Chúng ta sẽ xem xét một số cách thiết kế các ca kiểm thử dưới đây.

2.1. Kỹ thuật bao phủ câu lệnh (Statement Coverge)

Tư tưởng của phương pháp Thiết kế các ca kiểm thử dựa vào ý tưởng phương pháp bao phủ câu lệnh (Statement Coverage) là thực hiện mọi câu lệnh trong chương trình ít nhất 1 lần. Phương pháp này bao gồm:

2.1.1. Kỹ thuật bao phủ quyết định

Ý tưởng của phương pháp bao phủ quyết định (Decision coverage) là viết đủ các ca kiểm thử mà mỗi quyết định có kết luận đúng hay sai ít nhất 1 lần. Nói cách khác, mỗi hướng phân nhánh phải được xem xét kỹ lưỡng ít nhất 1 lần.

Bao phủ quyết định thường thỏa mãn bao phủ câu lệnh. Vì mỗi câu lệnh là trên sự bắt nguồn một đường đi phụ nào đó hoặc là từ một câu lệnh rẽ nhánh hoặc là từ điểm vào của chương trình, mỗi câu lệnh phải được thực hiện nếu mỗi quyết định rẽ nhánh được thực hiện. Tuy nhiên, có ít nhất 3 ngoại lệ:

1. Những chương trình không có quyết định.
2. Những chương trình hay thường trình con/phương thức với nhiều điểm vào. Một câu lệnh đã cho có thể được thực hiện nếu và chỉ nếu chương trình được nhập vào tại 1 điểm đầu vào riêng.

3. Các câu lệnh bên trong các ON-unit. Việc đi qua mỗi hướng rẽ nhánh sẽ là không nhất thiết làm cho tất cả các ON-unit được thực thi.

Vì chúng ta đã thấy rằng bao phủ câu lệnh là điều kiện cần thiết, nên một chiến lược tốt hơn là bao phủ quyết định nên được định nghĩa bao hàm cả bao phủ câu lệnh. Do đó, bao phủ quyết định yêu cầu mỗi quyết định phải có kết luận đúng hoặc sai, và mỗi câu lệnh đó phải được thực hiện ít nhất 1 lần.

Bao phủ quyết định là một tiêu chuẩn mạnh hơn bao phủ câu lệnh, nhưng vẫn khá yếu.

2.1.2. Kỹ thuật bao phủ điều kiện (Condition Coverage)

Ý tưởng của phương pháp bao phủ điều kiện (Condition coverage) là viết đủ các ca kiểm thử để đảm bảo rằng mỗi điều kiện trong một quyết định đảm nhận tất cả các kết quả có thể ít nhất 1 lần.

2.1.3. Kỹ thuật bao phủ quyết định/ điều kiện (Decision/Condition coverage)

Ý tưởng của phương pháp bao phủ quyết định/ điều kiện (Decision/ condition coverage) là thực hiện đủ các ca kiểm thử mà mỗi điều kiện trong một quyết định thực hiện trên tất cả các kết quả có thể ít nhất 1 lần, và mỗi điểm vào được gọi ít nhất 1 lần.

Điểm yếu của bao phủ quyết định/điều kiện là mặc dù xem ra nó có thể sử dụng tất cả các kết quả của tất cả các điều kiện, nhưng thường không phải vậy vì những điều kiện chắc chắn đã cản các điều kiện khác.

2.1.4. Kỹ thuật bao phủ đa điều kiện (Multiple Condition Coverage)

Ý tưởng của phương pháp bao phủ đa điều kiện (Multiple condition coverage) là viết đủ các ca kiểm thử mà tất cả những sự kết hợp của các kết quả điều kiện có thể trong mỗi quyết định, và tất cả các điểm vào phải được gọi ít nhất 1 lần.

2.1.5. Kiểm thử vòng lặp

Vòng lặp là các lệnh phổ biến và là cơ bản trong các ngôn ngữ lập trình. Tuy nhiên, việc kiểm thử đối với các vòng lặp là rất phức tạp, việc kiểm thử tất cả các trường hợp của vòng lặp nhiều khi là không thể vì số lượng các trường hợp kiểm thử là rất lớn. Trong phần này sẽ trình bày một số cải tiến của CFT để kiểm thử cho các vòng lặp.

Vòng lặp là một đường dẫn trong CFG chứa một hay nhiều nút được lặp lại nhiều hơn một lần, bao gồm các thuộc tính:

- Thân vòng lặp: Thực hiện một chức năng nào đó lặp đi lặp lại một số lần. Thân vòng lặp được biểu diễn trên đồ thị là một nút hay một số các nút bị chứa..
- Điều khiển lặp: Đưa ra các quyết định lặp, hoặc thực hiện thân vòng lặp hoặc là kết thúc vòng lặp.
- Nút khởi tạo/kết thúc: Một vòng lặp phải có ít nhất một hay nhiều nút khởi tạo và nút kết thúc.
- Hai hay nhiều vòng lặp có thể kết hợp tuần tự hay chứa lẫn nhau.

Hai vòng lặp được sử dụng nhiều nhất trong các ngôn ngữ lập trình là for và while.

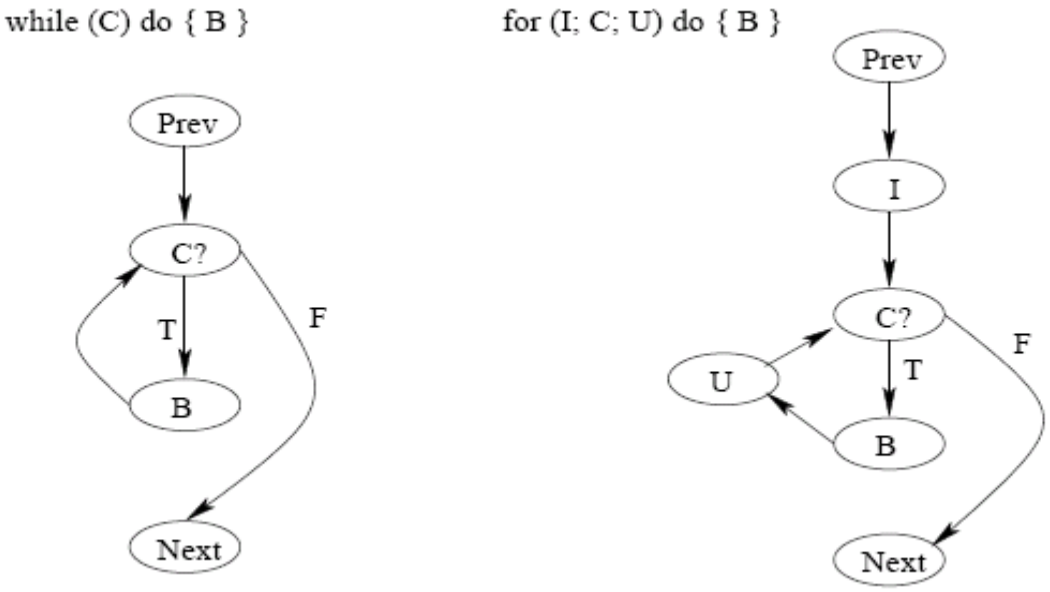
Các kiểu vòng lặp trên là các vòng lặp có cấu trúc, đối với các vòng lặp không có cấu trúc (có sử dụng lệnh go to) sẽ không kiểm thử mà sẽ thiết kế lại tương ứng với sử dụng việc xây dựng chương trình có cấu trúc.

Mỗi đường dẫn tại một lần lặp là một trường hợp kiểm thử và gọi đó là một đường dẫn phân biệt. Khi có nhiều vòng lặp kết hợp với nhau thì số

lượng các trường hợp kiểm thử sẽ tăng lên rất nhiều, như đối với trường hợp hai vòng lặp chứa nhau số các đường dẫn phân biệt là:

$$\sum_{i=0}^{M-1} N^i = \frac{N^M - 1}{N - 1}$$

Đối với M, N nhỏ thì có thể kiểm thử được tất cả mọi trường hợp, nhưng khi M, N lớn thì điều đó là không thể. Sử dụng một số kĩ thuật sau:



Ví dụ vòng lặp while và vòng lặp for

(Trich trang 62 Kỹ nghệ phần mềm (bản dịch tiếng việt bản dịch tiếng việt: Roger S.Pressman. Software Engineering, a Practitioner's Approach. 3th Edition, McGraw-Hill,1992), 3 tập, Nhà xuất bản Giáo dục.)

Vòng lặp đơn: Với vòng lặp đơn trong đó N là số lần lặp tối đa, các trường hợp kiểm thử sau được sử dụng để kiểm tra mỗi điều kiện sau:

- Bỏ qua vòng lặp
- Chỉ một lần lặp
- Hai lần lặp
- Lần lặp thứ N-1, N và N+1.

Các vòng lặp chứa nhau và kết hợp tuần tự: Theo một số bước sau:

- Bắt đầu tại vòng lặp trong cùng. Đặt tất cả các vòng lặp khác giá trị nhỏ nhất.

- Xây dựng các kiểm thử vòng lặp đơn cho vòng lặp trong cùng, trong khi đó giữ vòng lặp ngoài cùng tại các giá trị tham số lặp nhỏ nhất của chúng.

- Phát triển ra phía ngoài, xây dựng các kiểm thử cho vòng lặp tiếp theo, nhưng giữ tất cả các vòng lặp bên ngoài với giá trị nhỏ nhất và các vòng lặp lồng nhau khác giá trị "đặc biệt".

- Tiếp tục cho đến khi tất cả các vòng lặp được kiểm thử.

2.1.6. Kỹ thuật Điều kiện logic

Điều kiện lôgic có thể là:

- Điều kiện đơn: 1 biến Bool (cả toán tử phủ định): X

- Biểu thức quan hệ của 2 biểu thức số học C = (A Θ B), với Θ là phép so sánh: $<, \leq, =, >, \geq$ hay $\neq$ và A, B là biểu thức số học

- Điều kiện phức hợp cấu thành từ hơn một điều kiện đơn nhờ các toán tử Bool: *hoặc* ($\cup$), *và* ($\cap$)**,** *phủ định* ($\lrcorner$)

$D = X_1 \,\&\, X_2 \,\&\, \ldots\, X_n$, *trong đó* X_i là điều kiện đơn *và* là toán tử bool.

Kiểu sai trong điều kiện lôgic có thể là:

- Sai biến Bool.
- Sai toán tử Bool.
- Sai số hạng trong biểu thức toán tử Bool.
- Sai toán tử quan hệ.
- Sai biểu thức số học.

Chiến lược kiểm thử phân nhánh bao gồm:

- Kiểm thử từng điều kiện trong chương trình.
- Kiểm thử điều kiện không chỉ là phát hiện sai trong điều kiện mà còn phát hiện sai khác của chương trình liên quan.

Kiểm thử nhánh được thực hiện theo nguyên tắc: với mỗi điều kiện phức hợp C, thì với mỗi nhánh "true" và "false" của C, mỗi điều kiện đơn trong C phải được kiểm thử ít nhất một lần.

Chiến lược kiểm thử miền cần 3 hoặc 4 kiểm thử cho 1 biểu thức quan hệ bao gồm **<, >, =** và có thể ≠ nữa.

Nếu biểu thức Bool có n biến, mà n nhỏ thì thuận lợi, song n lớn thì khó thực hiện tất cả trường hợp !

Người ta đưa ra chiến lược cho các phép thử nhạy cảm bằng cách áp dụng kết hợp chiến lược *kiểm thử nhánh và kiểm thử miền (quan hệ).*

Để trả lời câu hỏi "Làm sao chỉ ra tất cả các trường hợp cần kiểm thử?", chúng ta xem xét một chiến lược có tên là *chiến lược kiểm thử BRO* (*Branch and Relational Operation*).

Chiến lược BRO bao gồm kiểm thử nhánh và toán tử quan hệ.

BRO dùng "ràng buộc điều kiện làm điều kiện cần thử" để phát hiện sai ở nhánh và toán tử khi xẩy ra 1 lần và không có biến chung.

Giả sử: $D = X_1 \& X_2 \& \ldots X_n$,

X_i*:* điều kiện đơn, *&:* toán tử bool

Cần đặc tả ràng buộc đầu ra của mỗi $\boldsymbol{X_i}$ tương ứng với điều kiện D đã xác định ?

Ta nói rằng, ràng buộc $\mathbf{X_i}$ của điều kiện D được phủ bởi một sự thực thi của C nếu khi đó, đầu ra của mỗi điều kiện đơn $\mathbf{X_i}$ trong D thoả mãn các ràng buộc tương ứng. Điều này có nghĩa là: Khi giá trị của D đã cho, ta cần tìm các điều kiện ràng buộc mà mỗi $\mathbf{X_i}$ (một thành phần của D) cần thỏa mãn để bảo đảm nhận được giá trị của D đã cho.

Với 1 biến Bool B, thì ràng buộc đầu ra của B là t (true) hoặc f (false).

Với 1 biểu thức quan hệ (A Θ B) thì ràng buộc đầu ra của nó là toán tử quan hệ:

Θ có thể nhận 1 trong 4 giá trị >, <, =, # (lớn hơn, nhỏ hơn, bằng hoặc khác).

Ví dụ:

Xét điều kiện C = A ∩ B, trong đó A và B là hai biến Bool. Khi đó ràng buộc đầu ra của C là: t (đúng) hoặc f (sai).

Chiến lược BRO đòi hỏi kiểm thử tập ba cặp giá trị: (t,t), (t,f) và (f,t) tương ứng với A và B trong C để phủ được các giá trị thực thi của C. Để thực hiện điều này, người ta thành lập bảng 2.1.

Bảng 2.1. Bảng kiểm thử kết quả ra

Đầu vào của các biến			Kết quả ra	Kết quả ra
A	**∩**	**B**	**= C**	**Thực tế**
t	∩	**t**	**t**	
t	∩	**f**	**t**	
f	∩	**t**	**f**	

Xét điều kiện C = (B = E). Khi đó ràng buộc đầu vào là " = " tương ứng với C là t và " < , >" tương ứng với C là f

Bảng 2.2. Bảng kiểm thử có ràng buộc

Đầu vào của toán tử			**Kết quả ra**	**Kết quả ra**
B	**Θ**	**E**	**= C**	**thực tế**
x	=	**y**	**t**	
x	<	**y**	**f**	
x	>	**y**	**f**	

Phủ của các ràng buộc này bảo đảm phát hiện được sai của toán tử quan hệ (B **Θ** E) trong C.

Xét điều kiện C = A∩(B = E). Khi đó, các ràng buộc của C là các cặp (t,t), (t,f) và (f,t); với (B = E) có giá trị t tương ứng với "=", và giá trị f tương ứng với "<" hoặc ">"; Bởi vậy tập các giá trị bảo đảm các ràng buộc đầu ra của C phải gồm 4 phần tử:

(t,=), (t,<), (t,>) và (f,=).

Bảng 2.3. Tập các giá trị bảo đảm các ràng buộc đầu ra

Đầu vào của biến và toán tử					**Kết quả ra**	**Kết quả ra**
A	**∩**	**(B**	**Θ**	**E)**	**= C**	**Thực tế**

t	∩	**x**	=	**y**	**t**	
t	∩	**x**	<	**y**	**f**	
t	∩	**x**	>	**y**	**f**	
f	∩	**x**	=	**y**	**f**	

Phủ của các ràng buộc này bảo đảm phát hiện được sai biến Bool A hoặc toán tử quan hệ (B **Θ** E) trong C.

Xét điều kiện C = A∩(B = E). Khi đó, các ràng buộc của C là các cặp (t,t), (t,f) và (f,t); với (B = E) có giá trị t tương ứng với "=", và giá trị f tương ứng với "<" hoặc ">"; Bởi vậy tập các giá trị bảo đảm các ràng buộc đầu ra của C phải gồm 4 phần tử (t,=), (t,<), (t,>) và (f,=).

Bảng 2.4. Tập các giá trị bảo đảm các ràng buộc đầu ra của C

Đầu vào của biến và toán tử					**Kết quả ra**	**Kết quả ra**
A	∩	**(B**	**Θ**	**E)**	**= C**	Thực tế
t	∩	**x**	=	**y**	**t**	
t	∩	**x**	<	**y**	**f**	
t	∩	**x**	>	**y**	**f**	
f	∩	**x**	=	**y**	**f**	

Phủ của các ràng buộc này bảo đảm phát hiện được sai biến Bool A hoặc toán tử quan hệ (B Θ E) trong C.

Xét điều kiện C là hội của hai biểu thức quan hệ:

$$C = (A>B) \cup (E=F)$$

Tập ràng buộc của C là (t,t), (t,f) và (f,t) và tương ứng sẽ là các đầu ra cần xác định cho việc kiểm thử: **(>,=); (>,<); (>,>); (=,=)** và **(<,=)**.

Bảng 2.5. Xác định các đầu ra để kiểm thử

Đầu vào các toán tử so sánh							**Kết quả ra**	**Kết quả ra**
(A	**Θ**	**B)**	**∪**	**(E**	**Θ**	**F)**	**= C**	**Thực tế**
z	>	**w**	**∪**	**x**	=	**y**	**t**	
z	>	**w**	**∪**	**x**	<	**y**	**f**	

z	>	w	U	x	>	y	f	
z	=	w	U	x	=	y	f	
z	<	w	U	x	=	y	f	

Phủ của các ràng buộc này bảo đảm phát hiện được các sai trong hai toán tử quan hệ (A>B) và (E = F) của C.

Hình 2.1. Xác định các ca kiểm thử bằng đường cơ bản và điều kiện

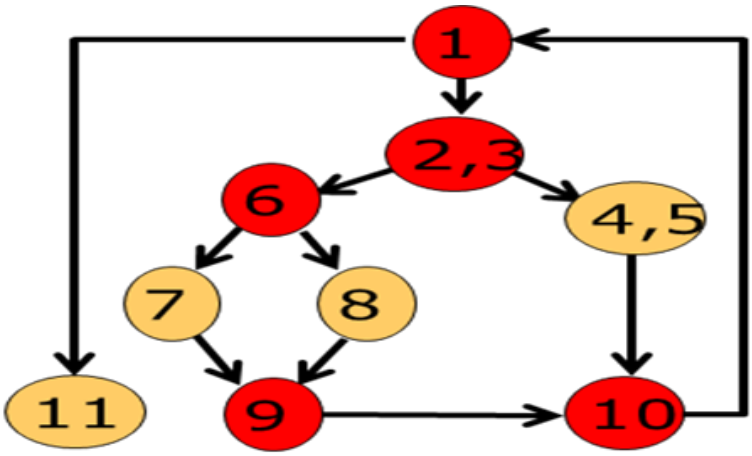
Ví dụ:

Xét 1 cấu trúc điều khiển chương trình

```
    Do while records remain
2       Read record
3       If record field1 = 0
4           then process record;
                store in buffer;
5       increment counter;
6       Else if record field2 = 0
7           then reset record;
8           Else process record;
                    store in file;
9           Endif
```

10 Endif

11 Enddo

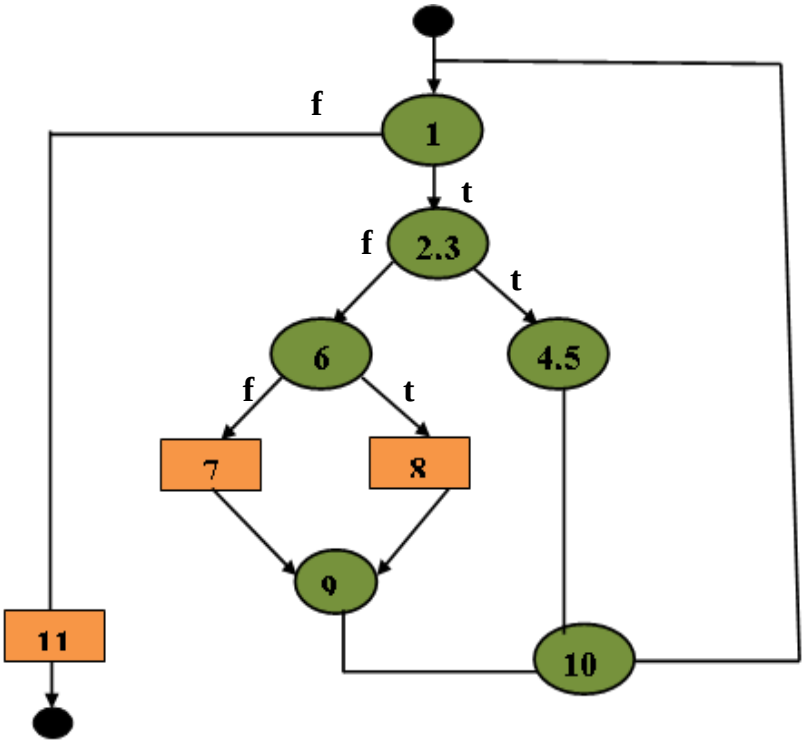


Hình 2.2. Đồ thị dòng để xác định tập đường cơ bản nhỏ nhất phủ các lệnh

Độ phức tạp được đo bằng số miền = 4.

Bảng 2.3. Tập đường cơ bản nhỏ nhất phủ các lệnh

Tập các đường cơ bản							
a	1	11					
b	1	2-3	4-5	10	1	11	
c	1	2-3	6	7	9	10	1
d	1	2-3	6	8	9	10	1



Hình 2.4. Xác định điều kiện cho đường cơ bản

Bảng 2.6. Dữ liệu kiểm thử theo tập đường cơ bản

Xác định dữ liệu kiểm thử							
Đường cơ bản	Điều kiện			Giá trị biến vào			Đầu ra
				số bản ghi	field1	field2	
b	**t**	**t**	**_**	**1**	**0**	-	Buffer #0
c	**t**	**f**	**t**	**2**	**#0**	**0**	Field1,2= ∅
d	**t**	**f**	**f**	**2**	**#0**	**#0**	Bản ghi thay đổi

2.1.7. Kỹ thuật ma trận kiểm thử

- Ma trận kiểm thử là 1 ma trận vuông có kích thước bằng số các nút đồ thị dòng:

+ Mỗi dòng cột ứng với 1 tên nút

+ Mỗi ô là tên 1 cung nối nút dòng đến nút cột

- Nhân liên tiếp k ma trận này ta được ma trận có số ở mỗi ô chỉ số con đường k cung từ nút dòng tới nút cột.

- Ma trận kiểm thử được sử dụng như 1 dữ liệu có cấu trúc để kiểm tra các con đường cơ bản: số đường đi qua nút (có thể tính cả trọng số của chúng).

2.1.8. Ma trận kiểm thử có trọng số:

Để ma trận kiểm thử là một công cụ mạnh trong việc đánh giá cấu trúc điều khiển chương ttrình. Khi kiểm thử ta thêm trọng số cho các cung của ma trận kiểm thử như sau:

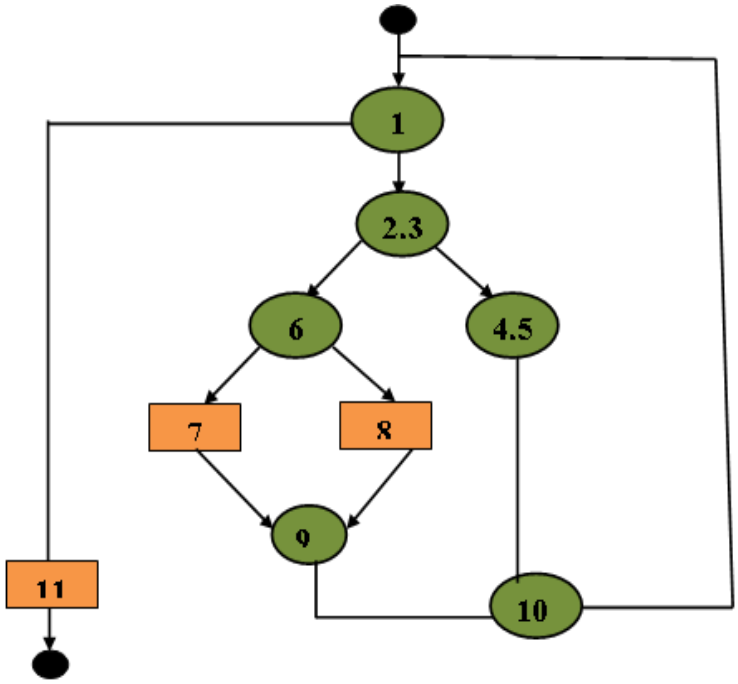
- Xác xuất cung đó được thực thi.

- Thời gian xử lý tiến trình đi qua cung đó

- Bộ nhớ đòi hỏi của tiến trình đi qua cung đó

- Nguồn lực đòi hỏi của tiến trình đi qua cung đó

Ví dụ ma trận kiểm thử:



Đỉnh	1	23	45	6	7	8	9	10	11	V(G)=4
1	1								1	1+1-1=1
23		1	1							1+1-1=1
45								1		1-1=0
6					1	1				1+1-1=1
7							1			1-1=0
8							1			1-1=0
9								1		1-1=0
10	1									1-1=0
11										∑+1= 3+1=4

2.2. Kỹ thuật phân lớp tương đương (Equivalence Patitioning)

Phân lớp tương đương là một phương pháp kiểm thử hộp đen chia miền đầu vào của một chương trình thành các lớp dữ liệu, từ đó suy dẫn ra các ca kiểm thử. Phương pháp này cố gắng xác định ra một ca kiểm thử mà làm lộ ra một lớp lỗi, do đó làm giảm tổng số các trường hợp kiểm thử phải được xây dựng.

Thiết kế ca kiểm thử cho phân lớp tương đương dựa trên sự đánh giá về các lớp tương đương với một điều kiện vào. Lớp tương đương biểu thị cho tập các trạng thái hợp lệ hay không hợp lệ đối với điều kiện vào.

Một ca kiểm thử được lựa chọn tốt nên có 2 tính chất:

1. Giảm thiểu được số lượng các ca kiểm thử không cần khác để hoàn thành mục tiêu kiểm thử "hợp lý".
2. Bao phủ được một tập rất lớn các ca kiểm thử khác. Tức là, chấp nhận sự có mặt hay vắng mặt của một ít lỗi qua tập giá trị đầu vào cụ thể.

Thiết kế Test-case bằng phân lớp tương đương tiến hành theo 2 bước:

Bước 1: Xác định các lớp tương đương.

Bước 2: Xác định các ca kiểm thử.

Các lớp tương đương được xác định bằng bằng cách lấy mỗi trạng thái đầu vào (thường là 1 câu hay 1 cụm từ trong đặc tả) và phân chia nó thành 2 hay nhiều nhóm.

Chú ý là hai kiểu lớp tương đương được xác định: lớp tương đương hợp lệ mô tả các đầu vào hợp lệ của chương trình, và lớp tương đương không hợp lệ mô tả tất cả các trạng thái có thể khác của điều kiện (ví dụ, các giá trị đầu vào không đúng). Với một đầu vào hay điều kiện bên ngoài đã cho, việc xác định các lớp tương đương hầu như là một quy trình mang tính kinh nghiệm. Để xác định các lớp tương đương có thể áp dụng tập các nguyên tắc dưới đây:

1. Nếu một trạng thái đầu vào định rõ giới hạn của các giá trị, xác định 1 lớp tương đương hợp lệ và 2 lớp tương đương không hợp lệ.
2. Nếu một trạng thái đầu vào xác định số giá trị, xác định một lớp tương đương hợp lệ và 2 lớp tương đương bất hợp lệ.
3. Nếu một trạng thái đầu vào chỉ định tập các giá trị đầu vào và chương trình sử dụng mỗi giá trị là khác nhau, xác định 1 lớp tương đương hợp lệ cho mỗi loại và 1 lớp tương đương không hợp lệ.
4. Nếu một trạng thái đầu vào chỉ định một tình huống "chắc chắn – must be", xác định 1 lớp tương đương hợp lệ và 1 lớp tương đương không hợp lệ.

Nếu có bất kỳ lý do nào để tin rằng chương trình không xử lý các phần tử trong cùng một lớp là như nhau, thì hãy chia lớp tương đương đó thành các lớp tương đương nhỏ hơn.

Với các lớp tương đương xác định được ở bước trên, bước thứ hai là sử dụng các lớp tương đương đó để xác định các ca kiểm thử. Quá trình này như sau:

1. Gán một số duy nhất cho mỗi lớp tương đương.
2. Cho đến khi tất cả các lớp tương đương hợp lệ được bao phủ bởi (hợp nhất thành) các ca kiểm thử, viết một ca kiểm thử mới bao phủ càng nhiều các lớp tương đương đó chưa được bao phủ càng tốt.

3. Cho đến khi các ca kiểm thử của bạn đã bao phủ tất cả các lớp tương đương không hợp lệ, viết một ca kiểm thử mà bao phủ một và chỉ một trong các lớp tương đương không hợp lệ chưa được bao phủ.
4. Lý do mà mỗi ca kiểm thử riêng bao phủ các trường hợp không hợp lệ là vì các kiểm tra đầu vào không đúng nào đó che giấu hoặc thay thế các kiểm tra đầu vào không đúng khác.

Mặc dù việc phân lớp tương đương là rất tốt khi lựa chọn ngẫu nhiên các ca kiểm thử, nhưng nó vẫn có những thiếu sót. Ví dụ, nó bỏ qua các kiểu ca kiểm thử có lợi nào đó. Hai phương pháp tiếp theo, phân tích giá trị biên và đồ thị nguyên nhân – kết quả, bao phủ được nhiều những thiếu sót này.

Kỹ thuật Phân hoạch tương đương là một kỹ thuật của kiểm thử hộp đen.

Về mặt nguyên tắc, kỹ thuật này chia (phân hoạch) miền vào của chương trình thành các lớp dữ liệu để lập ca kiểm thử theo mỗi lớp đó.

Cơ sở của kỹ thuật này là dữ liệu trong một lớp tương đương tác động như nhau lên chương trình, tạo ra cùng một trạng thái: đúng hay sai của chương trình.

Mục tiêu của phương pháp phân hoạch là tìm ra một ca kiểm thử để bộc lộ một lớp sai, từ đó rút gọn số ca kiểm thử cần phát triển. Ca kiểm thử được thiết kế cho từng lớp tương đương.

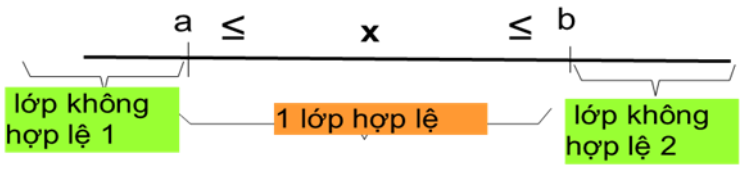
Vấn đề đặt ra là chọn lớp tương đương như thế nào ?

Phương châm xác định các lớp tương đương:

1. Điều kiện vào là một vùng rộng giới hạn một miền hay những giá trị đặc biệt thì cần xác định:
   - 1 lớp tương đương hợp lệ.
   - 2 lớp tương đương không hợp lệ.
2. Điều kiện vào một thành phần của một tập hoặc điều kiện Bool thì cần xác định:
   - 1 lớp tương đương hợp lệ.
   - 1 lớp tương đương không hợp lệ.

Ví dụ:

1. Điều kiện là 1 vùng giới hạn 1 miền giữa a và b:



Hình 2.5. Phân chia lớp tương đương

2. Điều kiện là một phần của một tập: x nhận giá trị nguyên dương sao cho:
   - 1 lớp tương đương hợp lệ: *các số nguyên dương*
   - 1 lớp tương đương không hợp lệ: *các số nguyên âm.*

2.3. Kỹ thuật phân tích giá trị biên (Boundary Value Analysis)

Kinh nghiệm cho thấy các ca kiểm thử mà khảo sát tỷ mỷ các điều kiện biên có tỷ lệ phần trăm cao hơn các ca kiểm thử khác. Các điều kiện biên là những điều kiện mà các tình huống ngay tại, trên và dưới các cạnh của các lớp tương đương đầu vào và các lớp tương đương đầu ra. Phân tích các giá trị biên là phương pháp thiết kế ca kiểm thử bổ sung thêm cho phân lớp tương đương, nhưng khác với phân lớp tương đương ở 2 khía cạnh:

1. Phân tích giá trị biên không lựa chọn phần tử bất kỳ nào trong 1 lớp tương đương là điển hình, mà nó yêu cầu là 1 hay nhiều phần tử được lựa chọn như vậy mà mỗi cạnh của lớp tương đương đó chính là đối tượng kiểm tra.

2. Ngoài việc chỉ tập trung chú ý vào các trạng thái đầu vào (không gian đầu vào), các ca kiểm thử cũng nhận được bằng việc xem xét không gian kết quả (các lớp tương đương đầu ra).

Phân tích giá trị biên yêu cầu óc sáng tạo và lượng chuyên môn hóa nhất định và nó là một quá trình mang tính kinh nghiệm rất cao. Tuy nhiên, có một số quy tắc chung như sau:

1. Nếu một trạng thái đầu vào định rõ giới hạn của các giá trị, hãy viết các ca kiểm thử cho các giá trị cuối của giới hạn, và các ca kiểm thử đầu vào không hợp lệ cho các trường hợp vừa ra ngoài phạm vi.

2. Nếu một trạng thái đầu vào định rõ số lượng giá trị, hãy viết các ca kiểm thử cho con số lớn nhất và nhỏ nhất của các giá trị và một giá trị trên, một giá trị dưới những giá trị này.

3. Sử dụng quy tắc 1 cho mỗi trạng thái đầu vào. Ví dụ, nếu một chương trình tính toán sự khấu trừ FICA hàng tháng và nếu mức tối thiểu là 0.00$, và tối đa là 1,165.25$, hãy viết các ca kiểm thử mà khấu trừ 0.00$ và 1,165.25, khấu trừ âm và khấu trừ lớn hơn 1,165.25$. Chú ý là việc xem xét giới hạn của không gian kết quả là quan trọng vì không phải lúc nào các biên của miền đầu vào cũng mô tả cùng một tập sự kiện như biên của giới hạn đầu ra (ví dụ, xét chương trình con tính SIN). Ngoài ra, không phải lúc nào cũng có thể tạo ra một kết quả bên ngoài giới hạn đầu ra, nhưng tuy nhiên rất đáng để xem xét tiềm ẩn đó.

1. Sử dụng nguyên tắc 2 cho mỗi trạng thái đầu ra.
2. Nếu đầu vào hay đầu ra của một chương trình là tập được sắp thứ tự (ví dụ,1 file tuần tự hay 1 danh sách định tuyến hay 1 bảng) tập trung chú ý vào các phần tử đầu tiên và cuối cùng của tập hợp.
3. Sử dụng sự khéo léo để tìm các điều kiện biên.

Người ta nhận thấy rằng:

- Các sai có xu hướng xuất hiện ở biên của vùng dữ liệu (hơn là ở "trung tâm").
- Các sai có thể cả trong và ngoài biên.
- Kiểm thử không chỉ chú ý đến các dữ liệu biên mà còn chú ý đến các dữ liệu sát biên (trong, ngoài).
- Các ca kiểm thử được xác định nhằm thực hiện các giá trị biên và sát biên.

Việc chọn lớp tương đương giá trị biên theo cách phân tích giá trị biên:

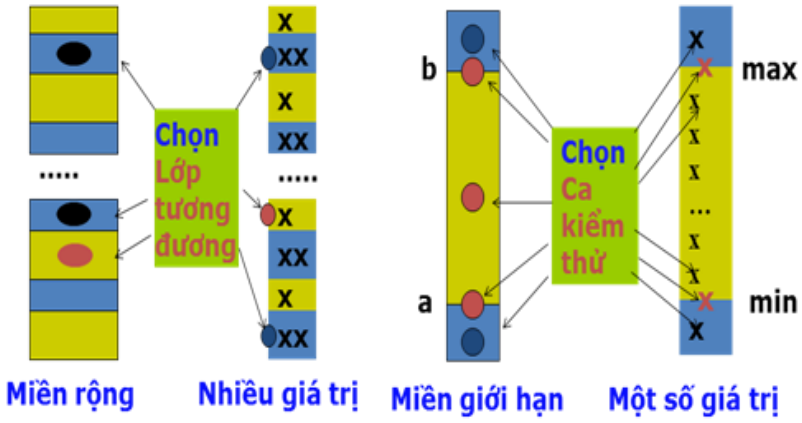
1. Nếu điều kiện vào là một miền giới hạn bởi a và b thì cần thiết kế các ca kiểm thử cho cả a và b, và cả trên, dưới a và b.

2. Nếu điều kiện vào đặc tả một số giá trị thì thiết kế các ca kiểm thử cho cả các số trên và dưới số nhỏ nhất và lớn nhất.

Phạm vi áp dụng cho giá trị biên ra:

-Áp dụng cách 1 và 2 cho cả điều kiện ra.

-Áp dụng điều kiện giá trị biên cho cả chương trình trung gian có các biên của cấu trúc dữ liệu được mô tả.



Hình 2.6. Mô hình phân hoạch và phân tích giá trị biên

2.4. Kỹ thuật đồ thị nguyên nhân – kết quả (Cause – Effect Graphing)

Một yếu điểm của phân tích giá trị biên và phân lớp tương đương là chúng không khảo sát sự kết hợp của các trường hợp đầu vào. Việc kiểm tra sự kết hợp đầu vào không phải là một nhiệm vụ đơn giản bởi vì nếu phân lớp tương đương các trạng thái đầu vào, thì số lượng kết hợp thường là rất lớn. Nếu không có cách lựa chọn có hệ thống một tập con các trạng thái đầu vào thì khi chọn ra một tập tùy hứng các điều kiện, điều này có thể dẫn tới việc kiểm thử không có hiệu quả.

Đồ thị nguyên nhân – kết quả hỗ trợ trong việc lựa chọn một cách có hệ thống tập các ca kiểm thử có hiệu quả cao. Nó có tác động có lợi ảnh hưởng tới việc chỉ ra tình trạng chưa đầy đủ và nhập nhằng trong đặc tả. Nó cung cấp cả cách biểu diễn chính xác cho các điều kiện logic và hành động tương ứng

Quá trình dưới đây được sử dụng để xây dựng được các ca kiểm thử:

1. Đặc tả được chia thành các phần có thể thực hiện được. Điều này là cần thiết bởi vì đồ thị nguyên nhân – kết quả trở nên khó sử dụng khi được sử dụng trên những đặc tả lớn.

2. Nguyên nhân và kết quả trong các đặc tả được nhận biết. Một nguyên nhân là một trạng thái đầu vào nhất định hay một lớp tương đương của các trạng thái đầu vào. Một kết quả là một trạng thái đầu ra hay một sự biến đổi hệ thống (kết quả còn lại mà một đầu vào có trạng thái của một chương trình hay hệ thống). Bạn nhận biết nguyên nhân và kết quả bằng việc đọc từng từ của đặc tả và gạch chân các từ hoặc cụm từ mô tả nguyên nhân và kết quả. Khi được nhận biết, mỗi nguyên nhân và kết quả được gán cho mộtsố duy nhất.

3. Xây dựng đồ thị nguyên nhân – kết quả bằng cách phát triển và biến đổi nội dung ngữ nghĩa của đặc tả thành đồ thị Boolean nối giữa nguyên nhân và kết quả.

4. Đồ thị được được diễn giải với các ràng buộc mô tả những sự kết hợp của nguyên nhân và/hoặc kết quả là không thể vì các ràng buộc ngữ nghĩa và môi trường.

5. Bằng việc dò theo các điều kiện trạng thái trong đồ thị một cách cẩn thận, bạn chuyển đổi đồ thị thành một bảng quyết định mục vào giới hạn. Mỗi cột trong bảng mô tả một ca kiểm thử.

1. Các cột trong bảng quyết định được chuyển thành các ca kiểm thử.

Vẽ đồ thị nguyên nhân – kết quả là phương pháp tạo các ca kiểm thử có hệ thống mô tả sự kết hợp của các điều kiện. Sự thay đổi sẽ là một sự lựa chọn kết hợp không thể dự tính trước, nhưng khi thực hiện như vậy sẽ bỏ sót nhiều ca kiểm thử "thú vị" được xác định bằng đồ thị nguyên nhân – kết quả.

Vì vẽ đồ thị nguyên nhân – kết quả yêu cầu chuyển một đặc tả thành một mạng logic Boolean, nó cung cấp một triển vọng khác và sự hiểu biết sâu sắc hơn nữa về đặc tả. Trên thực tế, sự phát triển của một đồ thị nguyên nhân – kết quả là cách hay để khám phá sự mơ hồ và chưa đầy đủ trong các đặc tả.

Mặc dù việc vẽ đồ thị nguyên nhân – kết quả tạo ra tập các ca kiểm thử hữu dụng, nhưng thông thường nó không tạo ra tất cả các ca kiểm thử hữu dụng mà có thể được nhận biết. Ngoài ra, đồ thị nguyên nhân – kết quả không

khảo sát thỏa đáng các điều kiện giới hạn. Dĩ nhiên, có thể cố gắng bao phủ các điều kiện giới hạn trong suốt quá trình.

Tuy nhiên, vấn đề trong việc thực hiện điều này là nó làm cho đồ thị rất phức tạp và dẫn tới số lượng rất lớn các ca kiểm thử. Vì thế, tốt nhất là xét một sự phân tích giá trị giới hạn tách rời nhau.

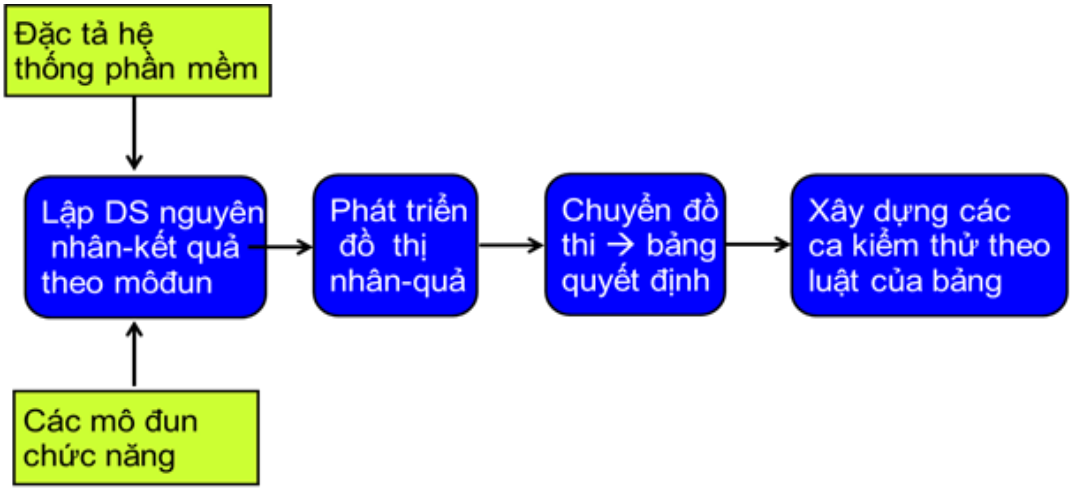
Vì đồ thị nguyên nhân – kết quả làm chúng ta mất thời gian trong việc chọn các giá trị cụ thể cho các toán hạng, nên các điều kiện giới hạn có thể bị pha trộn thành các ca kiểm thử xuất phát từ đồ thị nguyên nhân – kết quả. Vì vậy, chúng ta đạt được một tập các ca kiểm thử nhỏ nhưng hiệu quả mà thỏa mãn cả hai mục tiêu.

Chú ý là việc vẽ đồ thị nguyên nhân – kết quả phù hợp với một số quy tắc nhất định. Việc xác định đầu ra mong đợi cho mỗi ca kiểm thử là một phần cố hữu của kỹ thuật (mỗi cột trong bảng quyết định biểu thị các kết quả được mong đợi). Cũng chú ý là nó khuyến khích chúng ta tìm kiếm các kết quả có tác dụng không mong muốn.

Khía cạnh khó nhất của kỹ thuật này là quá trình chuyển đổi đồ thị thành bảng quyết định. Quá trình này có tính thuật toán, tức là bạn có thể tự động hóa nó bằng việc viết một chương trình. Trên thị trường đã có một vài chương trình thương mại tồn tại giúp cho quá trình chuyển đổi này.

Kỹ thuật đồ thị nhân- quả là một kỹ thuật để thiết kế ca kiểm thử. Nó cung cấp một biểu diễn chính xác giữa các điều kiện logic (đầu vào) và các hành động tương ứng (đầu ra- kết quả).

Kỹ thuật đồ thị nhân- quả được xây dựng dựa trên các mô-đun chức năng, lôgíc tiến trình và đặc tả hệ thống.



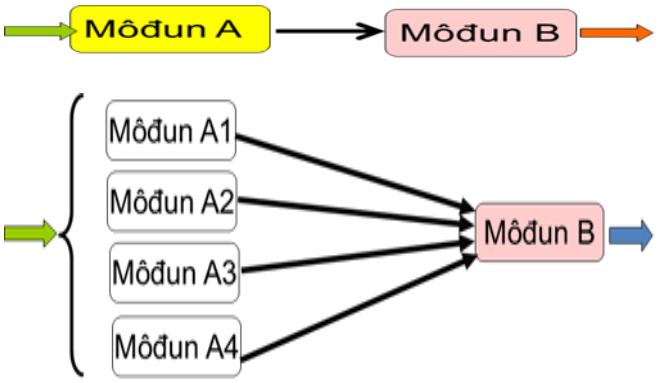
Hình 2.7. Các bước tiến hành theo kỹ thuật đồ thị nhân- quả

Ví dụ:

Bảng 2.7. Danh sách nhân- quả theo mô-đun

Mô-đun	Nguyên nhân	Kết quả	Định danh nhân - quả
A	Số > a	đúng	A1
	Số ≥ a	nghi ngờ	A2
	Số = a	nghi ngờ	A3
	Số < a	sai	A4
B	Số nguyên	đúng	B

Có nhiều công cụ để xây dựng đồ thị nhân- quả. Đồ thị có hướng hay được dùng hơn cả.

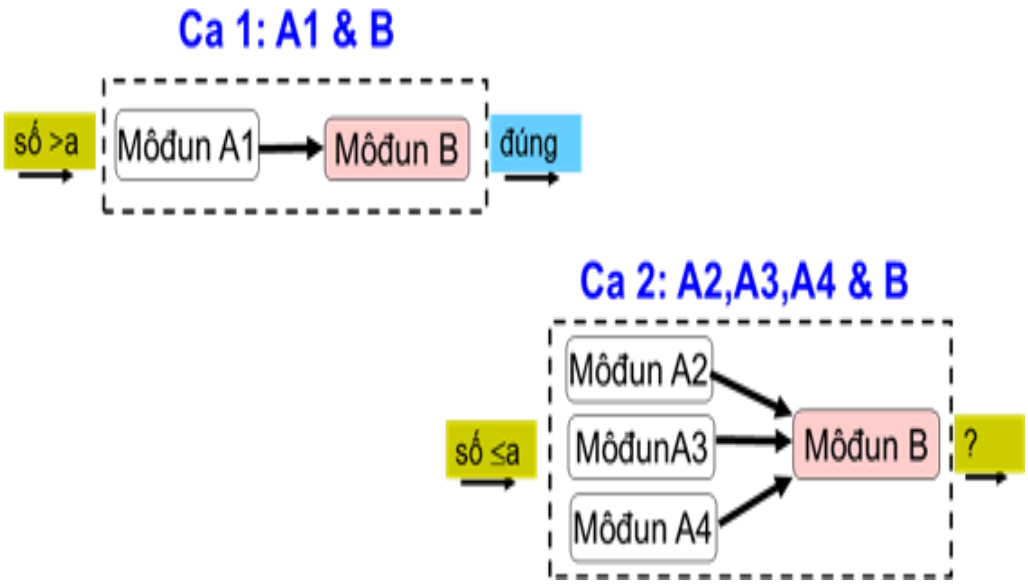


Hình 2.8. Xây dựng đồ thị nhân -quả bằng đồ thị

Bảng 2.8. Bảng quyết định của đồ thị nhân - quả

Định danh	Điều kiện	Đúng	Nghi ngờ	Sai
A1	Số > a	X		
A2,A3	Số ≥ a		X	
A4	Số < a			X
B	Số nguyên	X		
…	…	..	..	..

Chọn 2 ca kiểm thử



Hình 2.9. Các phương án lựa chọn ca kiểm thử

2.5. Kỹ thuật đoán lỗi (Error Guessing)

Một kỹ thuật thiết kế ca kiểm thử khác là *error guessing – đoán lỗi*. Người kiểm thử được đưa cho một chương trình đặc biệt. Họ phỏng đoán, cả bằng trực giác và kinh nghiệm, các loại lỗi có thể và sau đó viết các ca kiểm thử để đưa ra các lỗi đó.

Thật khó để đưa ra một quy trình cho kỹ thuật đoán lỗi vì nó là một quy trình có tính trực giác cao và không thể dự đoán trước. Ý tưởng cơ bản là liệt kê một danh sách các lỗi có thể hay các trường hợp dễ xảy ra lỗi và sau đó

viết các ca kiểm thử dựa trên danh sách đó. Một ý tưởng khác để xác định các ca kiểm thử có liên đới với các giả định mà lập trình viên có thể đã thực hiện khi đọc đặc tả (tức là, những thứ bị bỏ sót khỏi đặc tả, hoặc là do tình cờ, hoặc là vì người viết có cảm giác những đặc tả đó là rõ ràng). Nói cách khác, cần liệt kê những trường hợp đặc biệt đó mà có thể đã bị bỏ sót khi chương trình được thiết kế.

Các phương pháp thiết kế ca kiểm thử có thể được kết hợp thành một "kinh nghiệm" thiết kế ca kiểm thử. Vì mỗi phương pháp có thể đóng góp một tập riêng các ca kiểm thử hữu dụng, nhưng không cái nào trong số chúng tự nó đóng góp một tập trọn vẹn các ca kiểm thử. Kinh nghiệm thiết kế ca kiểm thử hợp lý có thể tóm lược như sau:

1. Nếu đặc tả có chứa sự kết hợp của các điều kiện đầu vào, hãy bắt đầu với việc vẽ đồ thị nguyên nhân – kết quả.
2. Trong trường hợp bất kỳ, sử dụng phương pháp phân tích giá trị biên. Hãy nhớ rằng đây là một sự phân tích của các biên đầu vào và đầu ra. Phương pháp phân tích giá trị biên mang lại một tập các điều kiện kiểm tra bổ sung, và rất nhiều hay toàn bộ các điều kiện này có thể được hợp nhất thành các kiểm thử nguyên nhân – kết quả.
3. Xác định các lớp tương đương hợp lệ và không hợp lệ cho đầu vào và đầu ra, và bổ sung các ca kiểm thử được xác định trên nếu cần thiết.
4. Sử dụng kỹ thuật đoán lỗi để thêm các ca kiểm thử thêm vào.
5. Khảo sát tính logic của chương trình liên quan đến tập các ca kiểm thử. Sử dụng tiêu chuẩn bao phủ quyết định, bao phủ điều kiện, bao phủ quyết định/điều kiện, hay bao phủ đa điều kiện (trong đó bao phủ đa điều kiện là được sử dụng nhiều nhất). Nếu tiêu chuẩn bao phủ không đạt được bởi các ca kiểm thử được xác định trong bốn bước trước, và nếu việc đạt được tiêu chuẩn là không thể (tức là, những sự kết hợp chắc chắn của các điều kiện có thể là không thể tạo vì bản

chất của chương trình), hãy thêm vào các ca kiểm thử có khả năng làm cho thỏa mãn tiêu chuẩn.

Các kinh nghiệm trên tuy chưa bảo đảm rằng tất cả các lỗi sẽ được tìm thấy nhưng nó rất hữu ích cho việc thực hành thiết lập các ca kiểm thử.

2.6. Kỹ thuật mô hình hóa

Trong một số tình huống, người ta dùng phương pháp mô hình hóa để kiểm thử hệ thống. Mỗi loại phương pháp sử dụng các mô hình khác nhau. Có thể phải kết hợp hoặc dùng nhiều mô hình cho một loại kiểm thử. Song việc kiểm thử hiệu quả tùy thuộc khả năng và kinh nghiệm của người tiến hành.

Ví dụ (về thiết kế kiểm thử hệ thống cầu thang máy):

Xét hệ thống cầu thang máy 4 tầng, đây là hệ thống thời gian thực có đặc trưng là các tầng (1,2,3,4), và 4 vị trí phòng cầu thang (cabinet) dừng ở các tầng.

Các yêu cầu bảo đảm hệ thống hoạt động bình thường:

- Từ một tầng cầu thang có thể đi đến một tầng bất kỳ.
- Để đến một tầng cần chọn số tầng tương ứng trong bảng điều khiển khi thang máy ở một tầng đang dừng.
- Khi cầu thang đang di chuyển lên (xuống), thì tại tầng dừng chỉ được chọn số tầng >(<) tầng đó.
- Tại tầng xuất phát: hướng di chuyển lên/ xuống tùy thuộc vào số của tầng được chọn đầu tiên.
- Cầu thang sẽ bắt đầu chuyển động sau khi chọn nút "khởi hành".
- Khi cầu thang di chuyển đến một tầng:

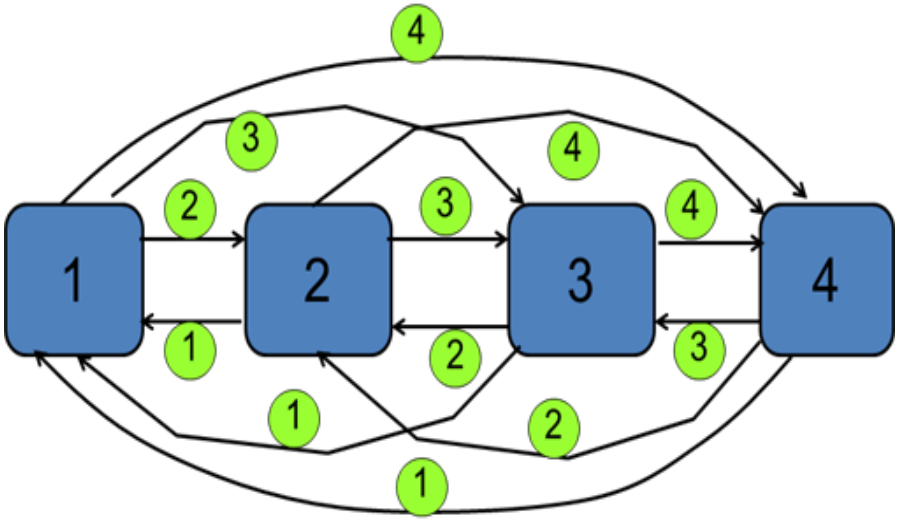
  + Nếu là tầng được chọn (số trong bảng điều khiển đã chọn hay phím tầng cùng chiều được chọn), thì cầu thang dừng và các phím đã chọn của tầng được trả về trạng thái bình thường (không chọn).

  + Nếu ngược lại thì cầu thang đi qua không dừng, và nếu phím tầng đã chọn theo hướng ngược lại thì nó cũng được trả về trạng thái không chọn.
- Khi nút ở tầng được chọn, 3 tình huống có thể:

1. Cầu thang đang đứng tại tầng được chọn. Khi đó cầu thang bắt đầu hoạt động, cửa được mở:
2. Nếu cầu thang đang đứng ở một tầng khác: tầng trên hay dưới tầng được chọn, cầu thang cần di chuyển xuống (lên) đến tầng được chọn thì dừng lại và mở cửa.
3. Cầu thang đang di chuyển (đã mô tả ở trên).

Hãy thiết kế các ca kiểm thử để kiểm thử hệ thống này ?

Mô hình hóa hệ thống bằng sơ đồ trạng thái có 4 trạng thái và có 12 đường mũi tên chuyển từ 1 trạng thái (tầng) đến 3 trạng thái (các tầng) còn lại. Kết quả ta được đồ thị có hướng (4 nút và 12 cung):



Hình 2.10. Sơ đồ trang thái hệ thống thang máy

- Cần thiết kế các ca kiểm thử để kiểm thử cầu thang thực hiện các chức năng:
- Ở 4 vị trí 4 tầng (phủ mọi trạng thái).
- Từ một tầng thang máy có thể đến mọi tầng khác, (phủ mọi chuyển trạng thái).
- Việc chuyển đến một trạng thái khác (tầng) được quyết định bằng phím được chọn (điều kiện).

- khi bắt đầu chọn hướng di chuyển ở tầng, cầu thang đang ở vị trí khác (tầng trên/dưới), cần thực hiện chuyển trạng thái từ tầng đang đứng về trạng thái tầng được chọn.

- ***Ca kiểm thử 1***: kế hoạch sử dụng đường đi: 1-2-3-4 sẽ phủ được 4 trạng thái của hệ thống

Bảng 2.9. Bảng dữ liệu phục vụ cho ca kiểm thử 1

Dữ liệu vào		**Kết quả ra**			
Trạng thái xuất phát	Sự kiện chon tầng	Trạng thái		Chuyển trạng thái	
		Dự kiến	Thực tế	Dự kiến	Thực tế
1	2, 3, 4	2		1,2	
		3		2,3	
		4		3,4	

Để bao phủ mọi chuyển trạng thái, tìm một chu trình ơle chứa đoạn đầu tiên 1-2-3-4 (Điều này có thể thực hiện được, vì mỗi đỉnh của đồ thị đều có số cung vào bằng số cung ra).

Chu trình ơle là đường đi qua mọi cạnh của đồ thị và đi qua đúng 1 lần. Nó là cơ sở thiết kế ca kiểm thử 2 (bao gồm cả ca kiểm thử 1).

- ***Ca kiểm thử 2***: 1-2-3-4-2-1-4-3-2-4-1-3-1

Theo tiền điều kiện đặt ra, mỗi lần di chuyển chỉ theo 1 hướng lên/xuống, nên cần chia đường đi thành nhiều đoạn kiểm thử liên tục với điều kiện trên.

Bảng 2.10. Bảng dữ liệu phục vụ cho ca kiểm thử 2:

Dữ liệu vào		**Kết quả ra**			
Trạng thái xuất phát	Sự kiện chon tầng	Trạng thái đến		Chuyển trạng thái	
		Dự kiến	Thực tế	Dự kiến	Thực tế
1	2, 3, 4	2		1,2	
		3		2,3	
		4		3,4	
4	2, 1	2		4,2	
		1		2,1	
1	4	4		1,4	
4	3, 2	3		4,3	
		2		3,2	
2	4	4		2,4	
4	1	1		4,1	
1	3	3		1,3	
3	1	1		3,1	

Để kiểm thử phủ các nút tầng, ta chọn đại diện là tầng giữa 2 và 3, Để kiểm thử sự chuyển trạng thái đồng bộ giữa tầng và cầu thang ta chọn 2 trạng thái nút tầng khi cầu thang ở các tầng khác: cả phiá trên và dưới.

- ***Ca kiểm thử 3,4***:

Bảng 2.11. Kế hoạch kiểm thử hai trạng thái tầng (lên, xuống) và đồng bộ:

Trạng thái cầu thang	Trạng thái tầng, nút chọn	Trạng thái đến		Chuyển trạng thái	
		Dự kiến	Thực tế	Dự kiến	Thực tế
1	T3, nút lên	3		1,3	
3	T2, nút xuống	2		3,2	

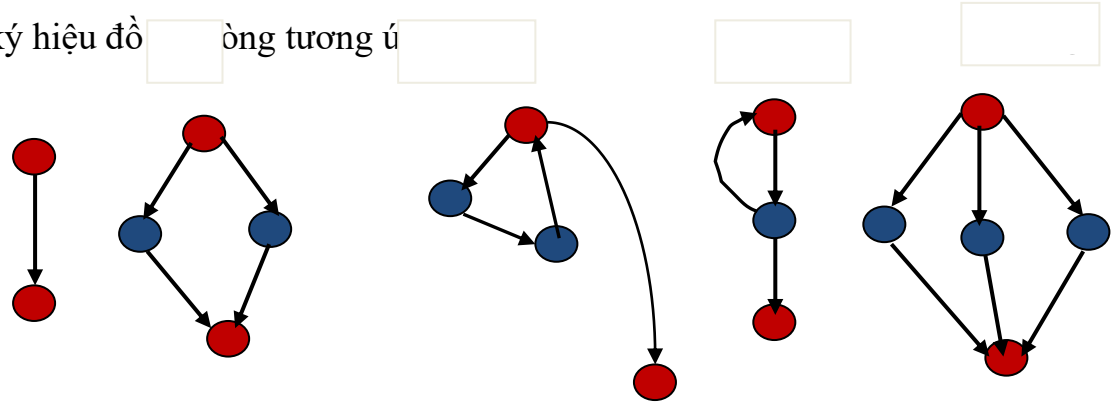
CHƯƠNG 3
PHẦN MỀM THỬ NGHIỆM THIẾT KẾ CA KIỂM THỬ

3.1 Phương pháp và kỹ thuật áp dụng thử nghiệm

Phần lý thuyết để thực hiện ví dụ lập trình thử nghiệm thiết kế ca kiểm thử là xác định độ phức tạp chu trình (số các ca kiểm thử).

Kỹ thuật đồ thị dòng là một kỹ thuật trong phương pháp kiểm thử hộp trắng đầu tiên được Tom McCabe đề nghị là *"kiểm thử đường cơ sở"*. Phương pháp xác định đường cơ sở giúp cho người thiết kế ca hợp kiểm thử có thể suy dẫn ra một cách đo độ phức tạp logic của thiết kế thủ tục và dùng cách đo này như một hướng dẫn để xác định một tập cơ sở các đường thực hiện. Các trường hợp kiểm thử được suy dẫn ra để thực hiện một tập cơ sở, được đảm bảo để thực hiện mọi câu lệnh trong chương trình ít nhất một lần trong khi kiểm thử.

Trong phương pháp xác định tập đường cơ sở, một hệ thống *ký pháp đơn giản được dùng để biểu diễn cho luồng điều khiển, được gọi là đồ thị dòng* (hay đồ thị chương trình). Đồ thị dòng mô tả cho dòng điều khiển logic dùng ký pháp được minh hoạ trong hình 3.1.. *Mỗi kết cấu có cấu trúc* đều có một ký hiệu đồ òng tương ú



Hình 3.1. Các cấu trúc cơ bản của đồ thị dòng (sequence,if, while, until, case)

Đồ thị dòng thực chất là một kỹ thuật dựa trên cấu trúc điều khiển của chương trình. Nó gần giống đồ thị luồng điều khiển của chương trình.

Đồ thị dòng nhận được từ đồ thị luồng điều khiển bằng cách:

- Gộp các lệnh tuần tự và điều khiển liên tiếp thành một lệnh;
- Thay lệnh rẽ nhánh của các đường điều khiển bằng một nút *"vị tự"*.

Cấu trúc đồ thị dòng gồm:

- Mỗi nút (hình tròn) biểu thị 1 hay một số lệnh tuần tự và rẽ nhánh, hoặc thay cho điểm phân nhánh hay hội tụ các đường điều khiển.
- Mỗi cạnh nối hai nút biểu diễn dòng điều khiển.

Kết quả đồ thị dòng thể hiện:

- Chia mặt phẳng thành nhiều miền.
- Có nút vị tự biểu thị sự phân nhánh của các cung.
- Mỗi cung nối từng cặp nút biểu diễn luồng điều khiển.

Ví dụ:

Đầu vào:

Xét 1 cấu trúc điều khiển chương trình

```
  Do while records remain
       Read record
       If record field1 = 0
            then process record;
                    store in buffer;
            increment counter;
       Else if record field2 = 0
             then reset record;
              Else process record;
                            store in file;
              Endif
      Endif
 Enddo
```

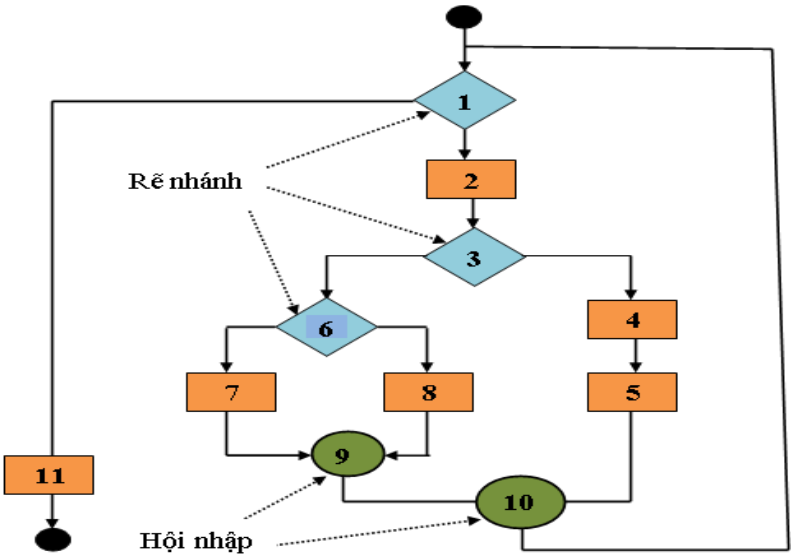
Quá trình giải:

Bước 1: Đánh số (mã hóa) lệnh:

```
1   Do while records remain
2        Read record
3        If record field1 = 0
```

```
4            then process record;
                    store in buffer;
5            increment counter;
6        Else if record field2 = 0
7                    then reset record;
8             Else process record;
                            store in file;
 9             Endif
10       Endif
11  Enddo
```

Bước 2: Vẽ s*ơ đồ điều khiển của chương trình*

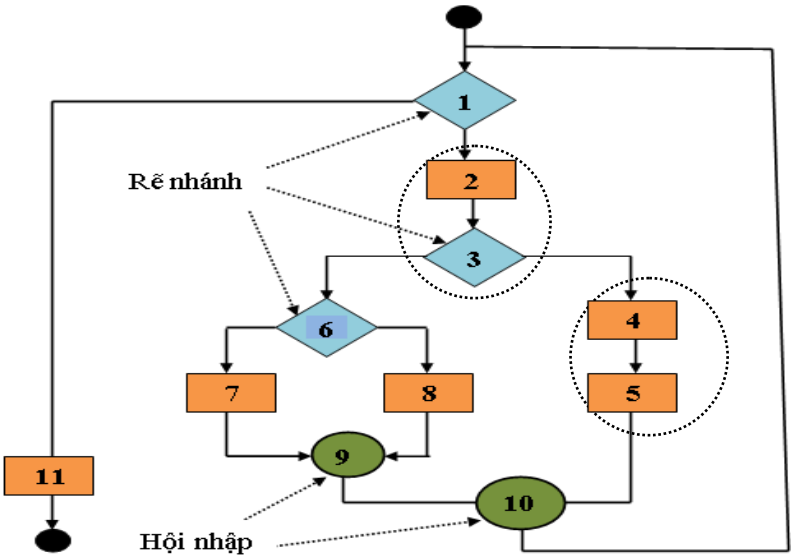


Hình 3.2. Sơ đồ điều khiển của chương trình

Trong lập trình, nếu vẽ được đồ thị thì tốt, nhưng cần xác định tổng số các nút và tổng số các cung.

Bước 3: vẽ *Sơ đồ luồng điều khiển*

Trong chương trình : rút gọn số nút và cung



Hình 3.3. Sơ đồ luồng điều khiển

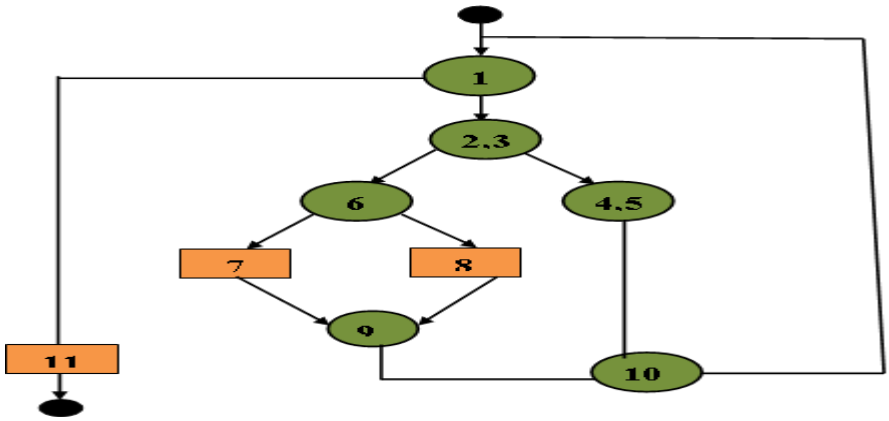
Bước 4: tính các thông số của đồ thị dòng theo cả 2 công thức (1) và (3)

Các thông số của đồ thị dòng trong ví dụ gồm:

9 nút (=N), trong đó:

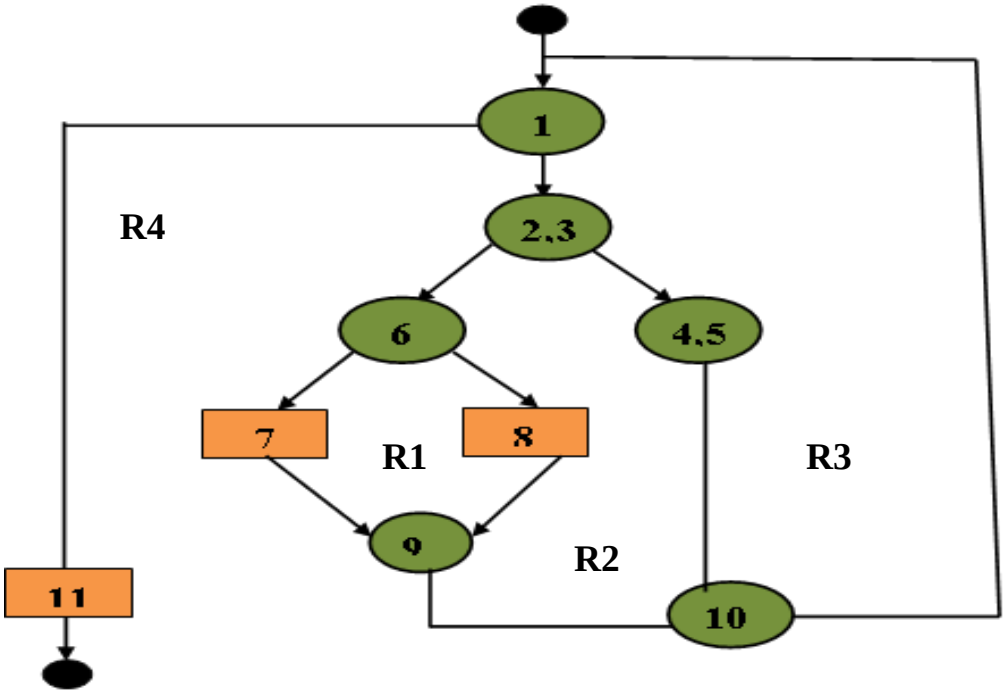
3 nút là vị tự (= P)

11cung (= E)



Hình 3.4. Đồ thị dòng

Đồ thị dòng chia mặt phẳng thành 4 miền: R1, R2, R3, R4



Hình 3.5. Độ phức tạp chu trình xác định từ đồ thị dòng

Để bảo đảm mọi câu lệnh đều được kiểm thử ít nhất 1 lần, cần tìm được tất cả các đường điều khiển độc lập trong chương trình (khác nhau ít nhất 1 lệnh).

Số các đường độc lập của một chương trình là giới hạn trên số ca kiểm thử cần tiến hành. Nó được gọi là độ phức tạp chu trình của chương trình.

Tập các đường độc lập/cơ bản (*basic paths*) của một chương trình trùng với các đường độc lập của đồ thị dòng (tìm đơn giản hơn).

Bước 5: Tính độ phức tạp chu trình để xác định số đường (ca) kiểm thử

Tính toán độ phức tạp chu trình:

Độ phức tạp chu trình V(G) của đồ thị G được tính theo các cách sau:

(1) V(G) = E - N + 2 (= 11-9+2 = 4)

(2) V(G) = số miền phẳng (= 4)

(3) V(G) = P + 1 (= 3+ 1 =4)

Trong đó: E = số cung; N = số nút; P= số nút vị tự

Với ví dụ về đồ thị dòng ở trên ta có: V(G) = 4

Xác định các ca kiểm thử:

Từ đồ thị dòng, xác định được độ phức tạp chu trình V(G)=4 và suy ra cần thiết kế 4 đường kiểm thử, tạo thành tập đường cơ bản trong bảng 3.1.

Bước 6: xác định tập đường cơ bản

Bảng 3.1. Tập đường cơ bản

Tập đường cơ bản							
a	1	11					
b	1	2-3	4-5	10	1		
c	1	2-3	6	7	9	10	1
d	1	2-3	6	8	9	10	1

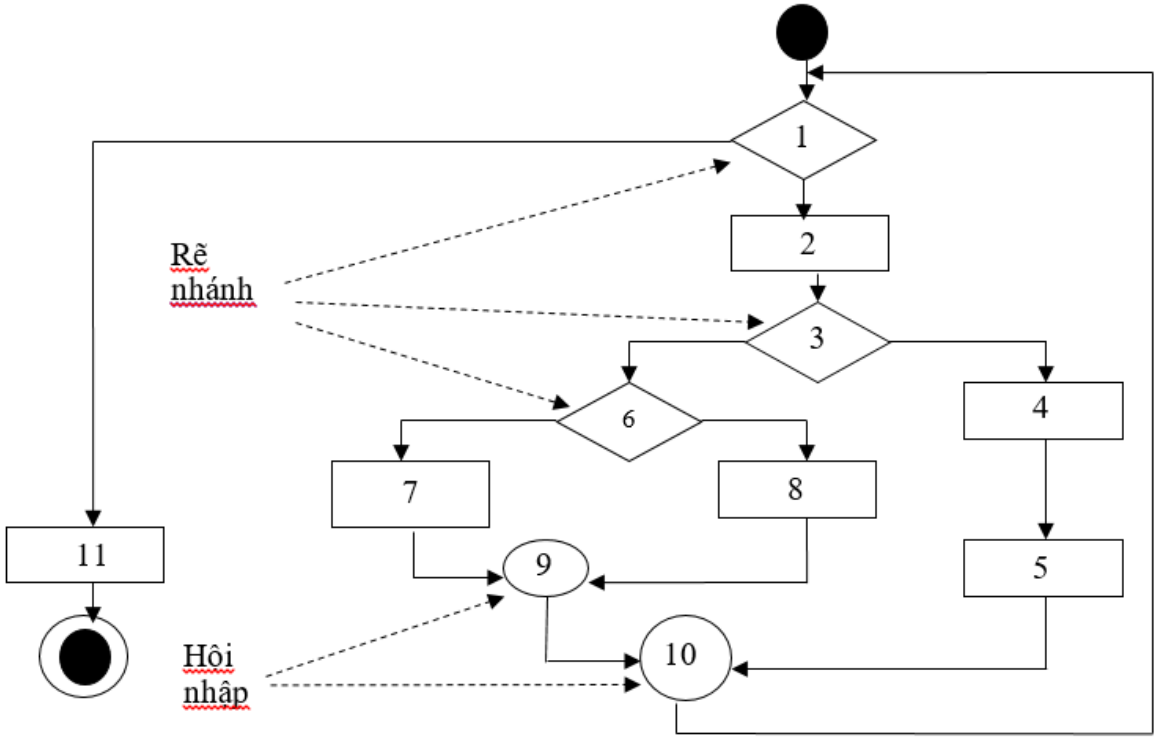
Bước 7: **Phương pháp Ma trận kiểm thử để xác định tập đường kiểm thử**

Ví dụ:

Xét một cấu trúc điều khiển chương trình

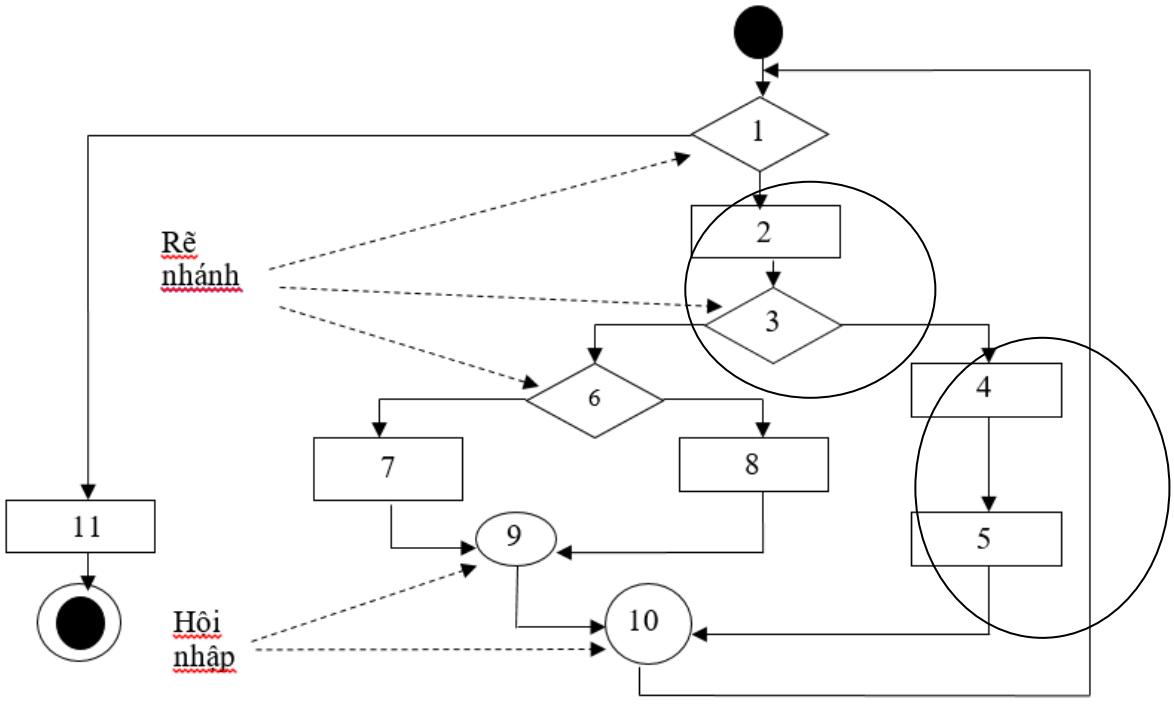
```
1 Do while còn bản ghi chưa xử lý
2       Read bản ghi chưa xử lý
3       if giá trị trường thứ nhất của bản ghi = 0
4               then xử lý bản ghi;
                        Cất vào bộ nhớ;
5               tăng biến đếm;
6       else if giá trị trường thứ hai của bản ghi = 0
7               then tạo lại bản ghi;
8                       else xử lý bản ghi;
                                Cất vào tệp;
9               end if
10      end if
11 end do
```

Các dòng lệnh có liên quan đến xử lý dữ liệu đều được đánh số thứ tự (1, 2, .., 11), từ đó sẽ được sơ đồ điều khiển của chương trình

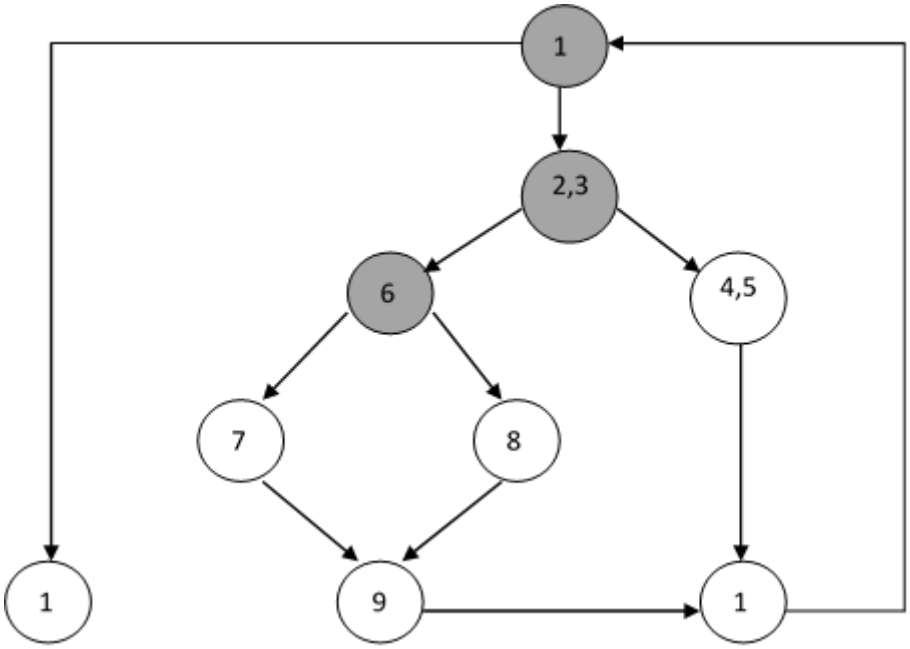


Hình 3.6. Sơ đồ điều khiển của chương trình

Trên cơ sở gộp các lệnh thực hiện tuần tự liền kề với nhau hoặc lệnh thực hiện liền kề rẽ nhánh, ta có sơ đồ luồng điều khiển (hình 3.7)



Hình 3.7. Sơ đồ luồng điều khiển gộp



Hình 3.8. Đồ thị dòng

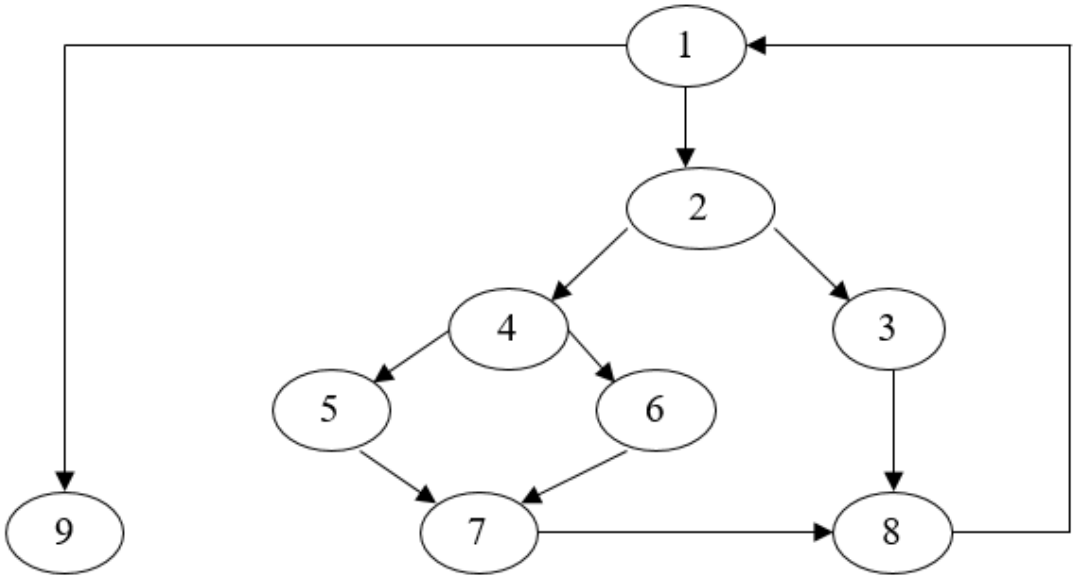
Từ sơ đồ luồng điều khiển gộp xác định được đồ thị dòng (hình 3.8). Các thông số của đồ thị dòng gồm:

9 nút (= N), trong đó:

3 nút là vị tự (= P) (nút được đánh dấu tô màu sẫm).

11 cung (= E).

Cho đơn giản, chúng ta đánh số lại đồ thị dòng sẽ được đồ thị dòng mới



Hình 3.9. Đồ thị dòng dùng để xác định ma trận kiểm thử

Từ đồ thị dòng này, xác định "ma trận kiểm thử":

Ma trận kiểm thử, ký hiệu là A=(aij) với i,j=1,2,3,...,n, là một ma trận vuông cấp n, có kích thước bằng số các nút (n) trên đồ thị dòng:

- Mỗi dòng/ cột ứng với tên một nút;
- Mỗi ô: là tên một cung nối nút dòng đến nút cột.

Nhân liên tiếp k ma trận này ta được ma trận có số ở mỗi ô chỉ số con đường k cung từ nút dòng tới nút cột.

Ma trận kiểm thử được sử dụng như một dữ liệu có cấu trúc để kiểm tra các con đường cơ bản: số đường đi qua nút (có thể tính cả trọng số của chúng).

Ma trận kiểm thử là một công cụ mạnh trong việc đánh giá cấu trúc điều khiển chương trình. Khi kiểm thử, ta thêm trọng số cho các cung của ma trận kiểm thử (ma trận kiểm thử có trọng số) như sau:

- Xác suất cung đó được thực thi.
- Thời gian xử lý của tiến trình đi qua cung đó.
- Bộ nhớ đòi hỏi của tiến trình đi qua cung đó.
- Nguồn lực đòi hỏi của tiến trình đi qua cung đó.

Bảng 3.2 Bảng tính độ phức tạp của đồ thị dòng V(G):

nút	1	2	3	4	5	6	7	8	9	V(G) = **4**
1		1							1	1 + 1 -1 = 1
2			1	1						1 + 1 -1 = 1
3								1		1 – 1 = 0
4					1	1				1 + 1 -1 = 1
5							1			1 – 1 = 0
6							1			1 – 1 = 0
7								1		1 – 1 = 0
8	1									1 – 1 = 0
9										V(G)= ∑ theo cột+1=3 + 1 = **4**

(Các số 1 trong ma trận đánh dấu nút dòng đi tới nút cột, ví dụ nút dòng 2 xác định chuyển tới nút cột 3 nên có tọa độ (2,3) được đánh dấu 1).

Do đó ta có bảng ma trận kiểm thử A= (aij) với i,j=1,2,3,4,...9:

	1							1

		1	1					
							1	
				1	1			
						1		
						1		
							1	
1								

Bảng 3.3 Bảng ma trận kiểm thử A = (aij)

Từ ma trận A, xác định độ phức tạp của đồ thị trong ví dụ bằng công thức:

$$V(G) = (\sum_{i=1}^{9}((\sum_{j=1}^{9} a\ ij) - 1) + 1 = 4$$

Để thuận tiên cho việc lập trình, chúng ta xác định cách tính độ phức tạp theo quy trình sau:

Bước 1: tính các tổng theo hàng của ma trận A trừ dòng cuối cùng

T1=a11+ a12+ a13+... +a19=2

T2=a21+ a22+ a23+...+ a29 =2

T3=a31+ a32+ a33+... +a39 =1

T4=a41+ a42+ a43+...+ a49 =2

T5=a51+ a52+ a53+... +a59 =1

T6=a61+ a62+ a33+...+ a69 =1

T7=a71+ a72+ a73+... +a79 =1

T8=a81+ a82+ a83+... +a89 = 1

T9= a91+ a92+ a93+... +a99 = 0

Bước 2: Tính

T1:=T1-1=1

T2:=T2-1=1

T3:=T3-1=0

T4:=T4-1=1

T5:=T5-1= 0

T6:=T6-1 = 0

T7:=T7-1 = 0

T8:=T8-1 = 0

T9 = 0 (vì là điểm kết thúc)

Bước 3: Tính V(G)= (T1+T2+...+T9) + 1

Trong đó: (T1+T2+...+T9) chính là số nút vị tự P

V(G)= P+1= (T1+T2+...+T9) + 1= 3+1 = 4

Như vậy, xác định được số đường phải kiểm thử là 4

Từ đây, chúng ta dễ dàng xác định được tập đường kiểm thử:

Tập đường kiểm thử:							
a	1	9					
b	1	2	3	8	1		
c	1	2	4	5	7	8	1
d	1	2	4	6	7	8	1
Từ kết quả trên, xác định được tập đường kiểm thử cho bài toán ban							
a	1	11					
b	1	2-3	4-5	10	1		
c	1	2-3	6	7	9	10	1
d	1	2-3	6	8	9	10	1

Bảng 3.4 Tập đường kiểm thử

3.2. Áp dụng thiết kế tự động ca kiểm thử cho một số mô-đun chương trình trong bài giảng về câu lệnh có cấu trúc tại Trường THCS Thủy Sơn Hải Phòng.

3.2.1. Chọn lọc một số bài tập lập trình về câu lệnh có cấu trúc tại trường THCS Thủy Sơn Hải Phòng.

Một số bài tập lập trình về câu lệnh có cấu trúc được chọn thử nghiệm

gồm:

1-Viết chương trình tính tổng S = 1+2+...+N bằng cách dùng vòng lặp WHILE;

2-Viết chương trình giải phương trình bậc nhất ax+b=0;

3-Viết chương trình giải phương trình bậc hai: $ax^2 + bx + c = 0$, $a \neq 0$.

3.2.2. Đặc tả các mô-đun chương trình theo các bài toán đã chọn (input) theo 3 cấp độ dễ, trung bình, khó

Bài tập 1 (dễ)

Viết chương trình tính tổng S = 1+2+...+N bằng cách dùng vòng lặp WHILE:

Program TinhTong;

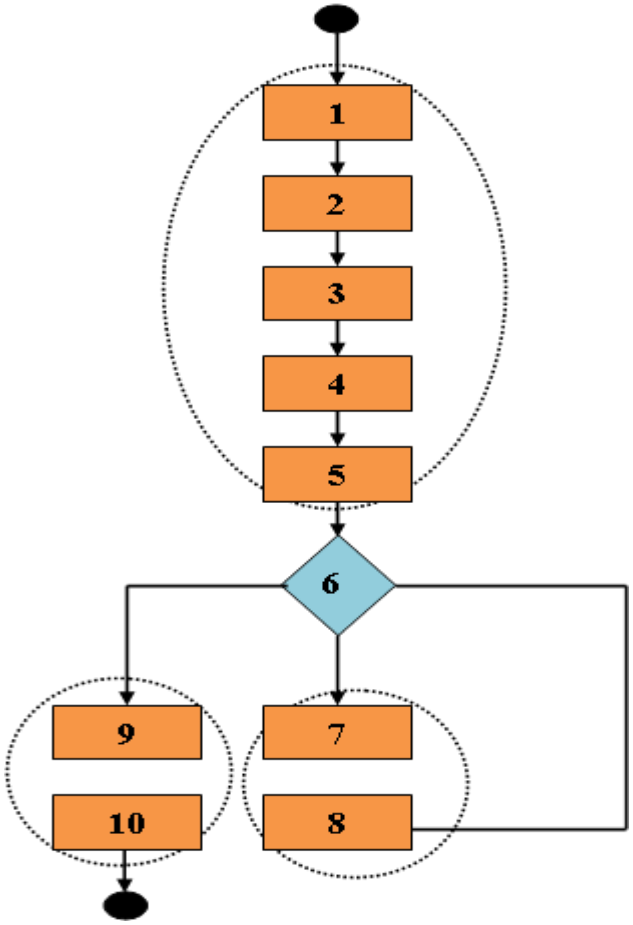
```
Uses crt;
Var
N,i,S:integer;
Begin
Clrscr;
Write('Nhap vao gia tri cua N :');
Readln(N);
S:=0; i:=1;
  While i<=N Do
        Begin
        S:=S+i; i:=i+1;
     End;
Writeln('Ket qua la ',S);
Readln;
End.
```

Bước 1: Gán nhãn

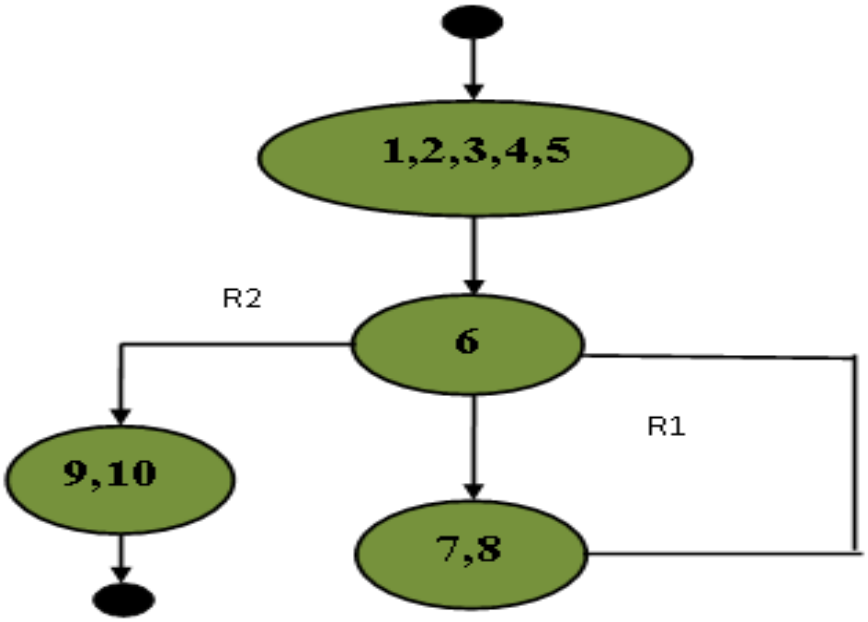
```
Begin
1.Clrscr;
```

```
2.Write('Nhap vao gia tri cua N :');
3.Readln(N);
4.S:=0;
5.i:=1;
  6.While i<=N Do
        Begin
        7.S:=S+i;
        8.i:=i+1;
      End;
9.Writeln('Ket qua la ',S);
10.Readln;
End.
```

<u>Bước 2</u>: Sơ đồ điều khiển



Bước 3: Đồ thị dòng để xác định ma trận kiểm thử



Bước 4: Ma trận kiểm thử và Độ phức tạp V(G)

Nút	1,2,3,4,5	6	7,8	9,10	V(G)=2
1,2,3,4,5		1			1-1=0
6			1	1	1+1-1=1
7,8		1			1-1=0
9,10					Σ+1=1+1=2

Bước 5: Tập đường kiểm thử

Tập đường cơ bản				
A	12345	6	9,10	
B	12345	6	7,8	6

Bài tập 2 (trung bình)

Viết chương trình giải phương trình bậc nhất ax+b=0:

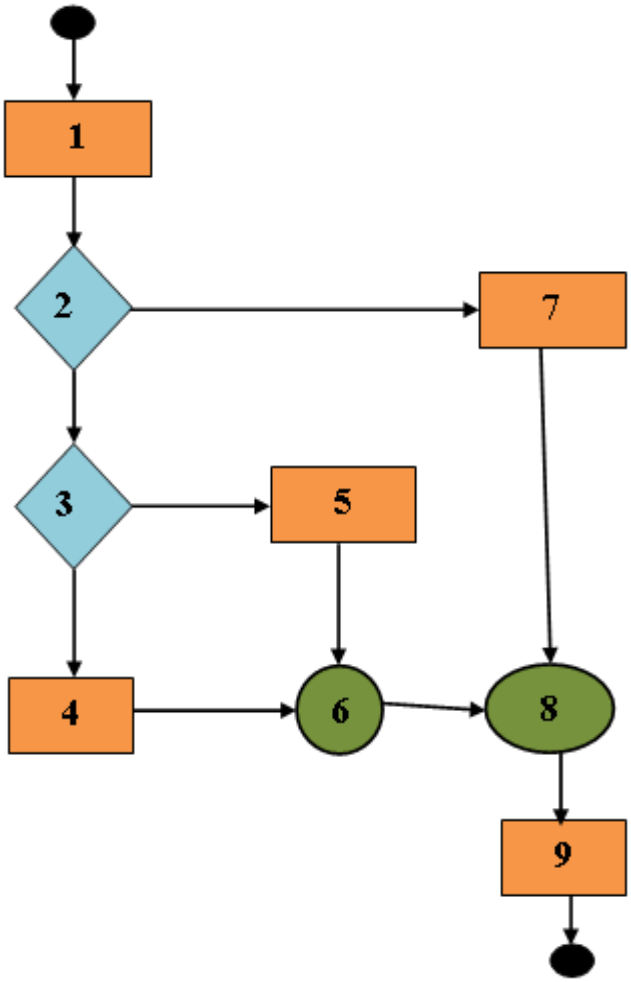
```
Program ptbacnhat;
  Uses crt;
  Var a,b,x:real;
  Begin
        Clrscr;
        Write('Nhap a,b: ');
        Readln(a,b);
        if (a=0 )and (b=0) then
        writeln('Phuong trinh co vo so nghiem')
        else
           if (a=0) and (b<>0) then
              writeln('Phuong trinh vo nghiem')
           else
        writeln('Pt co nghiem x= ',-b/2*a:3:2);
        readln;
  End.
```

<u>Bước 1</u>: Gán nhãn

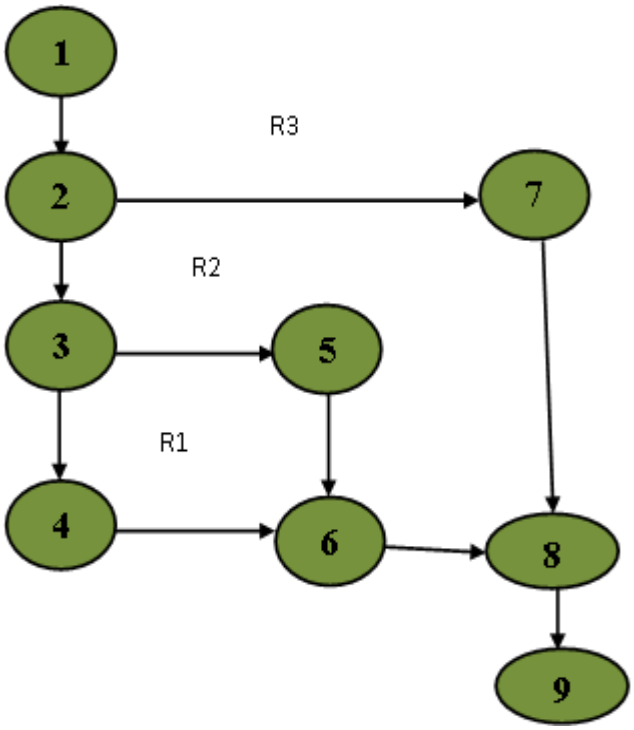
```
Begin
1.        Clrscr;
2.       Write('Nhap a,b: ');
3.       Readln(a,b);
4.       if (a=0 )and (b=0) then
5.        writeln('Phuong trinh co vo so nghiem')
        else
6.           if (a=0) and (b<>0) then
7.             writeln('Phuong trinh vo nghiem')
           else
8.       writeln('Pt co nghiem x= ',-b/2*a:3:2);
```

```
9.          readln;
  End.
```

<u>Bước 2</u>: Sơ đồ điều khiển



<u>Bước 3</u>: Đồ thị dòng để xác định ma trận kiểm thử



<u>Bước 4</u>: Ma trận kiểm thử và Độ phức tạp V(G)

Nút	1	2	3	4	5	6	7	8	9	V(G)=3
1.		1								1-1=0
2.			1				1			1+1-1=1
3.				1	1					1+1-1=1
4.						1				1-1=0
5.						1				1-1=0
6.								1		1-1=0
7.								1		1-1=0
8.									1	1-1=0
9.										Σ+1=2+1=3

<u>Bước 5</u>: Tập đường kiểm thử

Tập đường cơ bản							
A	1	2	3	4	6	8	9

B	1	2	3	5	6	8	9
C	1	2	7	8	9		

Bài tập 3 (Khó)

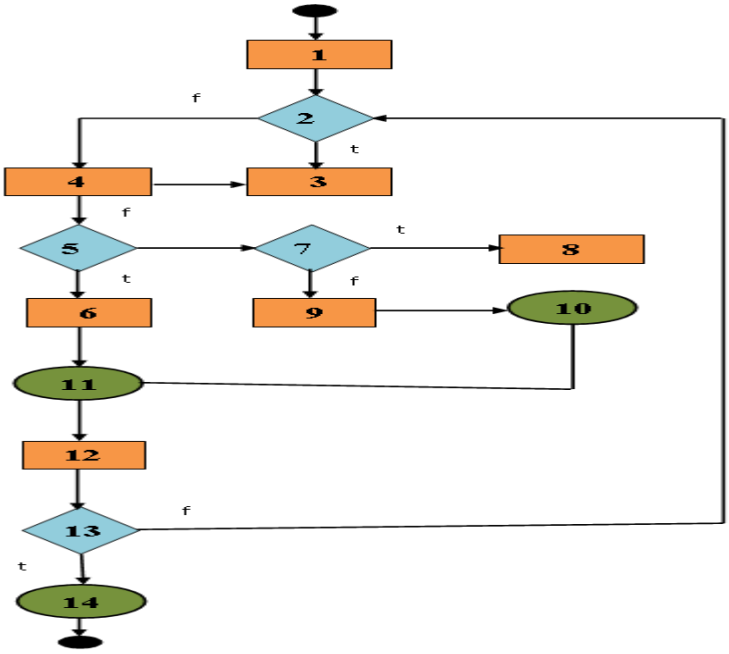
Viết chương trình giải phương trình bậc hai: $ax^2+ bx + c = 0$, a#0:

```
Program ptbac2;
  Uses crt;
  Var a,b,c,x1,x2,dt:real; ch:char;
  Begin
      Repeat
      Clrscr;
      Write('Nhap a,b,c: ');
      Readln(a,b,c);
      While a=0 do
      Begin
      Writeln('Ban phai nhap lai he so a <>0');
        readln(a);
        end;
        dt:=b*b-4*a*c;
      if dt<0 then
      writeln('Phuong trinh vo nghiem')
      else
       if dt=0 then
       writeln('Pt co no kep x1=x2= ',-b/(2*a):3:2)
       else
        Begin
          writeln('Pt co no phan biet: ');
          writeln('x1= ',(-b+(sqrt(dt)))/(2*a):3:2);
          writeln('x2=',(-b-(sqrt(dt)))/(2*a):3:2);
         End;
       writeln('Ban co muon tiep tuc nua ko? c/k ');
       readln(ch);
       Until ch='k' ;
       readln;
End.
```

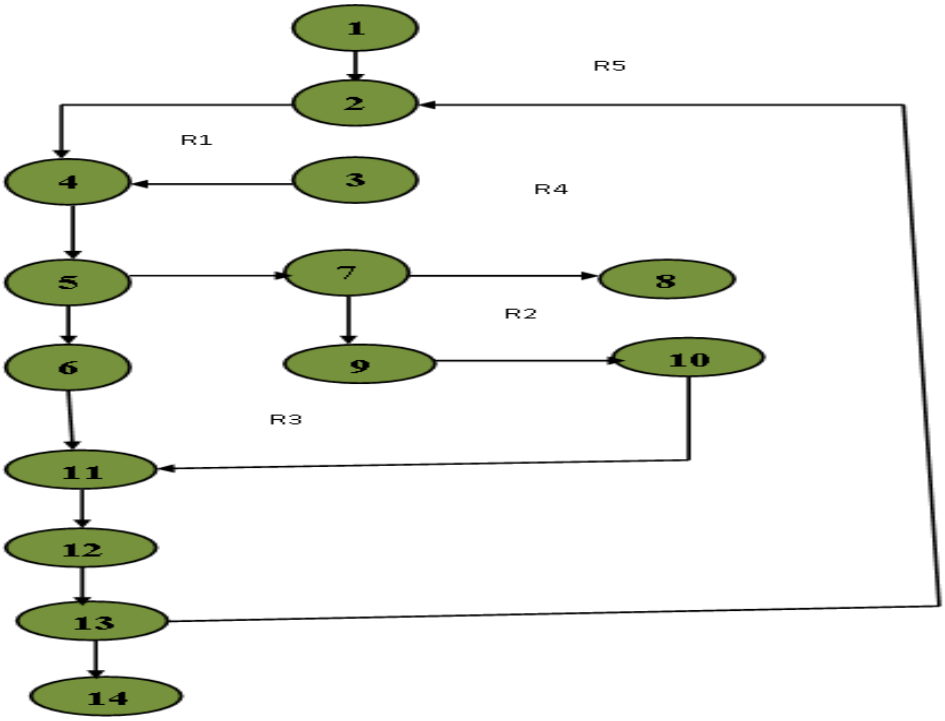
Bước 1: Gán nhãn

```
    Begin
1.   Repeat
     Clrscr;
     Write('Nhap a,b,c: ');
     Readln(a,b,c);
2.   While a=0 do
      Begin
3.      Writeln('Ban phai nhap lai he so a <>0');
        readln(a);
       end;
4.        dt:=b*b-4*a*c;
5.      if dt<0 then
6.     writeln('Phuong trinh vo nghiem')
     else
7.      if dt=0 then
8.        writeln('Pt co no kep x1=x2= ',-b/(2*a):3:2)
        else
        Begin
9.         writeln('Pt co no phan biet: ');
10          writeln('x1= ',(-b+(sqrt(dt)))/(2*a):3:2);
           writeln('x2=',(-b-(sqrt(dt)))/(2*a):3:2);
          End;
11.      writeln('Ban co muon tiep tuc nua ko? c/k ');
12.        readln(ch);
13.        Until ch='k' ;
14.        readln;
        End.
```

Bước 2: Sơ đồ điều khiển



<u>Bước 3</u>: Đồ thị dòng để xác định ma trận kiểm thử



<u>Bước 4</u>: Ma trận kiểm thử và Độ phức tạp V(G)

Nút	1	2	3	4	5	6	7	8	9	10	11	12	13	14	V(G)=5
1.		1													**1-1=0**
2.			1	1											**1+1-1=1**

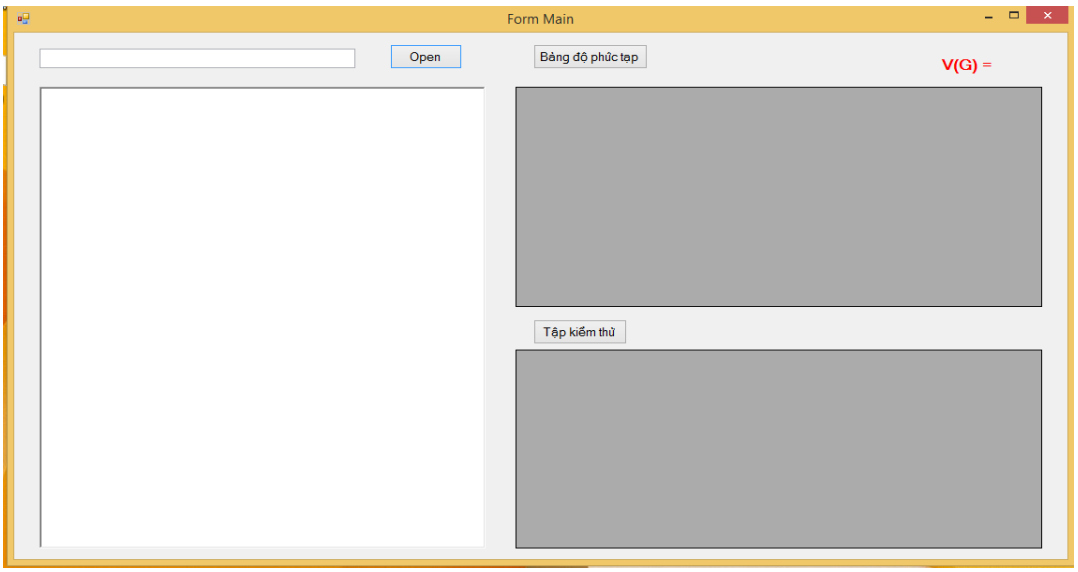
3.				1										**1-1=0**
4.					1									**1-1=0**
5.						1	1							**1+1-1=1**
6.											1			**1-1=0**
7.								1	1					**1+1-1=1**
8.										1				**1-1=0**
9.										1				**1-1=0**
10.											1			**1-1=0**
11.												1		**1-1=0**
12.													1	**1-1=0**
13.	1												**1**	**1+1-1=1**
14.														Σ+1=4+1=5

<u>Bước 5</u>: Tập đường kiểm thử

Tập đường cơ bản												
a	1	2	4	5	6	11	12	13	2			
b	1	2	4	5	6	11	12	13	14			
c	1	2	3	4	5	6	11	12	13	14		
d	1	2	4	5	7	9	10	11	12	13	14	
e	1	2	4	5	7	8	10	11	12	13	14	

3.3. Một số giao diện chính của chương trình

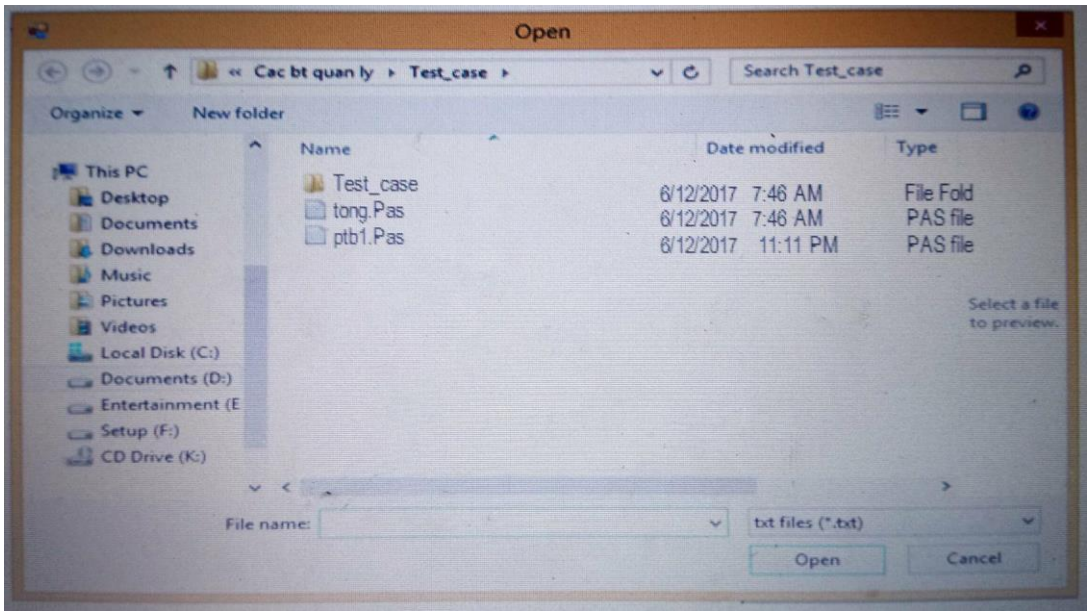
3.3.1. Form chính



Hình 3.1. Form chính của chương trình

3.3.2. Form chọn dường dẫn tới dữ liệu

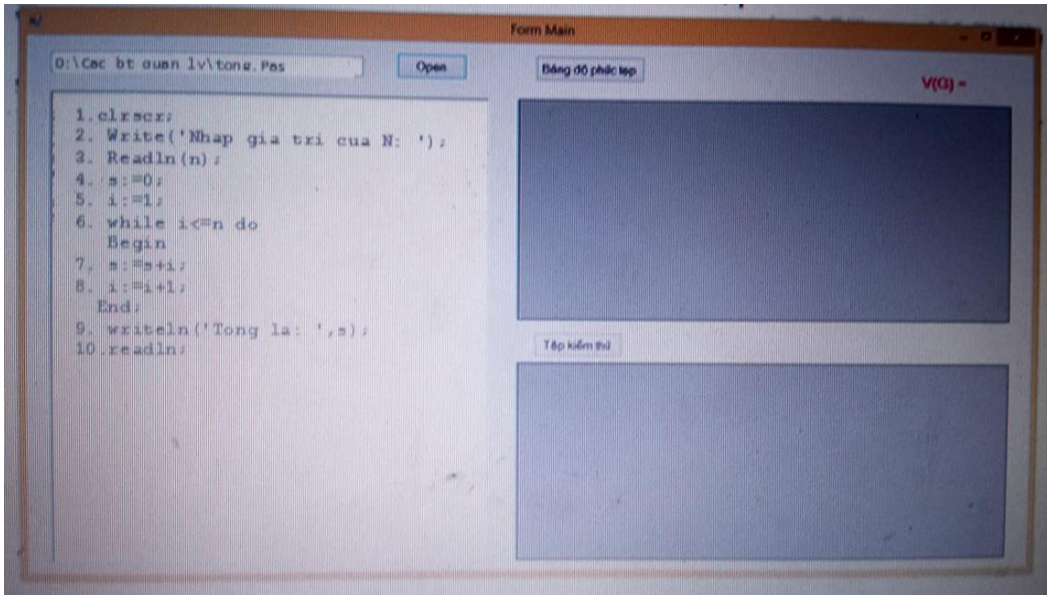
Nhấn nút "OPEN" để chọn đường dẫn.



Hình 3.2. Form chọn dường dẫn tới dữ liệu

3.3.3. Hiển thị dữ liệu

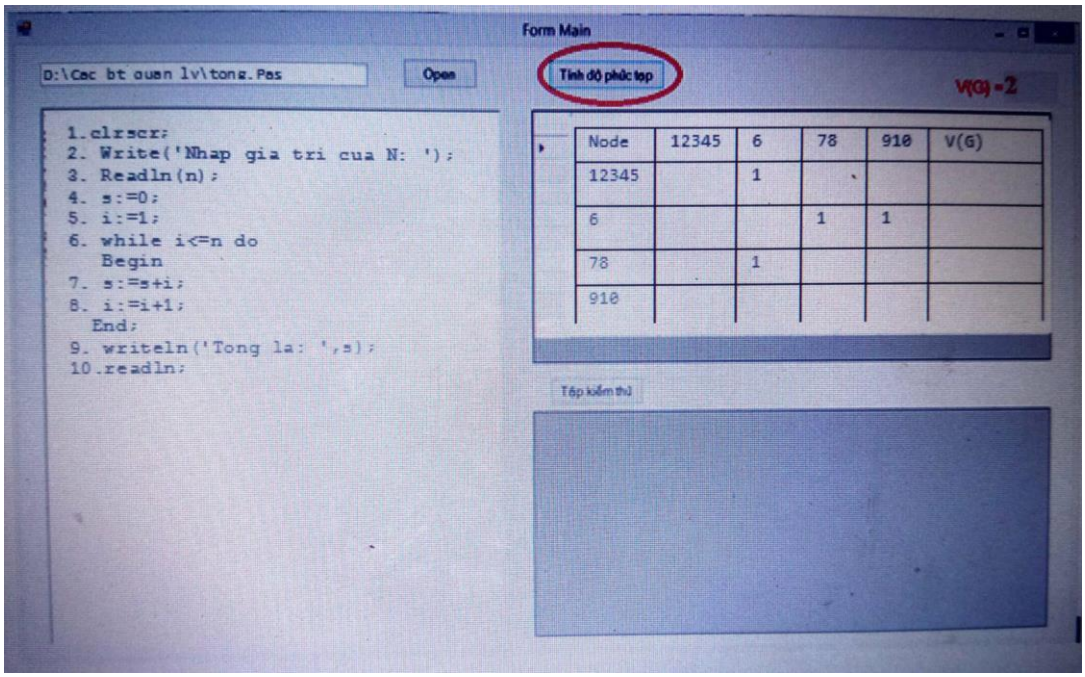
Sau khi chọn file dữ liệu, Form main hiển thị:



Hình 3.3. Form hiển thị dữ liệu

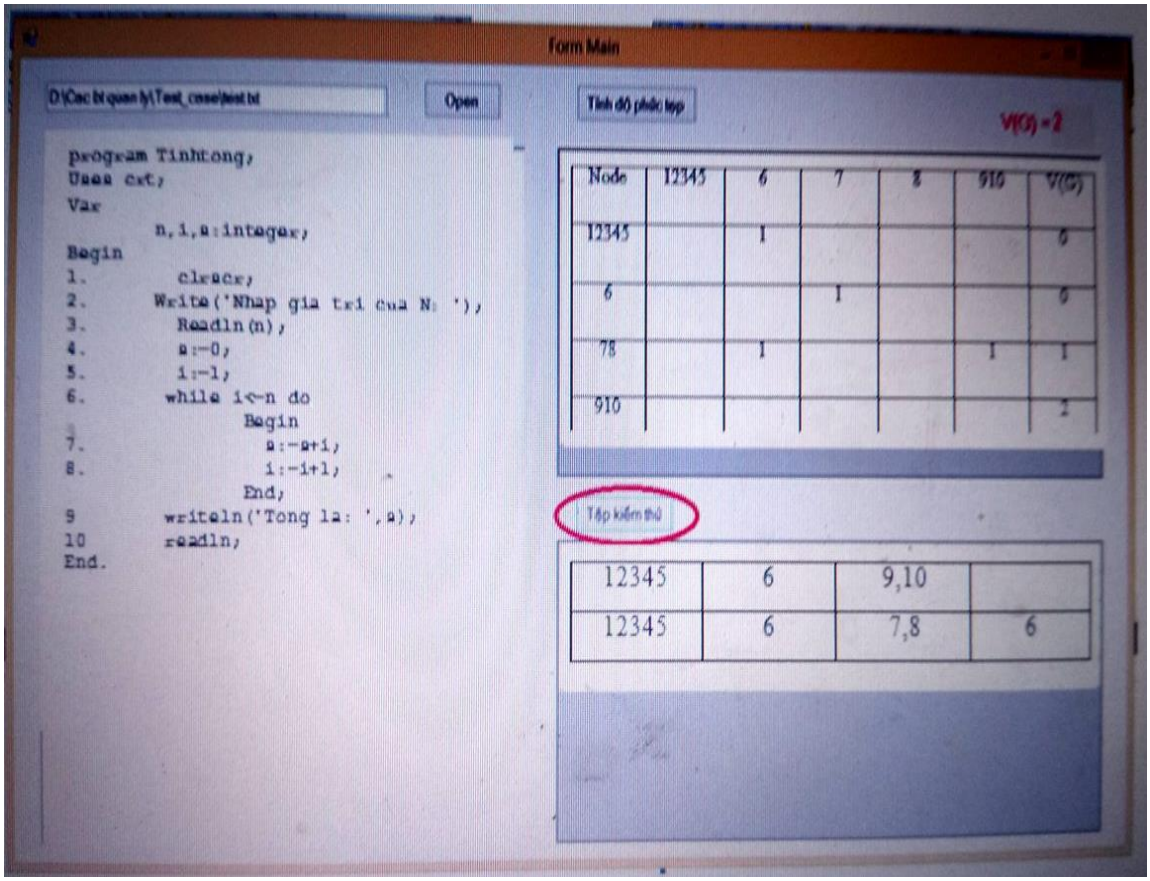
3.3.4. Tính toán độ phức tạp

Nhấn vào nút "**Tính độ phức tạp**"



3.3.5. Xuất ra các phương án kiểm thử

Nhấn nút "Tập kiểm thử"



3.4. Đánh giá kết quả thử nghiệm và hướng phát triển

3.4.1. Đánh giá

- Kết quả thử nghiệm cho ra một tập các đường cơ bản đúng như ví dụ trên.

- Chương trình thực hiện cho kết quả đúng theo thuật toán trên.

3.4.2. Hướng phát triển

- Phát triển chương trình có thể xử lý được với giải thuật với ngôn ngữ C, C++, ...

- Phát triển chương trình có thể tìm ra các tập kiểm thử với các số liệu cụ thể.

KẾT LUẬN

Ở Việt Nam ngành Công Nghệ Phần Mềm vẫn được coi là khá non trẻ, chưa có nhiều thành tựu lớn. Kiểm Thử Phần Mềm vẫn còn là một vấn đề khá mới mẻ.

Việc nghiên cứu lựa chọn các kỹ thuật và chiến lược Kiểm Thử Phần Mềm phù hợp giúp cho việc kiểm thử có hiệu quả, giảm chi phí và thời gian. Việc xây dựng tài liệu Kiểm Thử Phần Mềm hợp lý sẽ giúp cho việc tổ chức, quản lý và thực hiện kiểm thử có hiệu quả.

Trong khuôn khổ một luận văn thạc sĩ, tác giả nghiên cứu các kĩ thuật xác định các ca kiểm thử và dữ liệu kiểm thử dựa trên ma trận kiểm thử và ứng dụng kiểm thử các bài toán Pascal trong chương trình THCS tại trường THCS Thủy Sơn.

Do thời gian giới hạn nên chương trình mới chỉ dừng lại ở kiểm thử các bài bài toán cơ bản trong chương trình Pascal THCS, trong thời gian tới tác giả sẽ tiếp tục phát triển chương trình để trở thành một ứng kiểm thử được tất cả các bài toán Pascal và các bài của các ứng dụng khác, hoàn chỉnh, bổ sung thêm các chức năng kiểm thử cú pháp, kiểm thử các java scripts, kiểm thử khuôn dạng dữ liệu và kiểm thử các giao dịch web.

Hiện nay vấn đề Kiểm Thử Phần Mềm hầu như vẫn chưa được đầu tư và quan tâm đúng mức. Việt Nam ta đang trong quá trình xây dựng một ngành công nghiệp phần mềm thì không thể xem nhẹ việc Kiểm Thử Phần Mềm vì xác suất thất bại sẽ rất cao, hơn nữa, hầu hết các công ty phần mềm có uy tín đều đặt ra yêu cầu nghiêm ngặt là nếu một phần mềm không có tài liệu kiểm thử đi kèm thì sẽ không được chấp nhận.

TÀI LIỆU THAM KHẢO

1. Tiếng việt

[1]. Thạc Bình Cường, Nguyễn Đức Mận (2011), *Kiểm thử và đảm bảo chất lượng phần mềm*. NXB Bách Khoa Hà Nội.

[2]. Lê Văn Phùng (2010), *Kỹ nghệ phần mềm*. Nhà xuất bản thông tin và truyền thông.

[3]. Lê Văn Phùng (2013), *Kỹ nghệ phần mềm nâng cao*. Nhà xuất bản thông tin và truyền thông.

[4]. Ngô Trung Việt (2000), *Kỹ nghệ phần mềm* (bản dịch tiếng việt: Roger S.Pressman. Software Engineering, a Practitioner's Approach. 3th Edition, McGraw-Hill,1992), 3 tập, Nhà xuất bản Giáo dục.

[5]. Nguyễn Văn Vỵ, Nguyễn Việt Hà (2009), *Giáo trình kỹ nghệ phần mềm*. Nhà xuất bản Giáo dục Việt Nam.

2. Tiếng Anh

[6]. Berard M. (2001), *Systems and software verification.Model-Checking techniques and tools*. Springer.

[7]. Boehm B. (1981), *Software Engineering Economics*, Prentice Hall.

[8]. *John* C.Munson, ph.D. (2005), *Software specification and design. An Engineering Approach*, Taylor & Francis.

[9]. *John* C.Munson, ph.D. (2003), ***S**oftware Engineering measurement*, Taylor & Francis.

[10]. Roger S.Pressman (1992). *Software Engineering, a Practitioner's Approach*. 3th Edition, McGraw-Hill, Inc.

[11]. Sommerville Ian (2004), *Software Engineering*, Addison-Wesley, 4th,1992, 6th, 2001, 7th Editor Chapter 19.