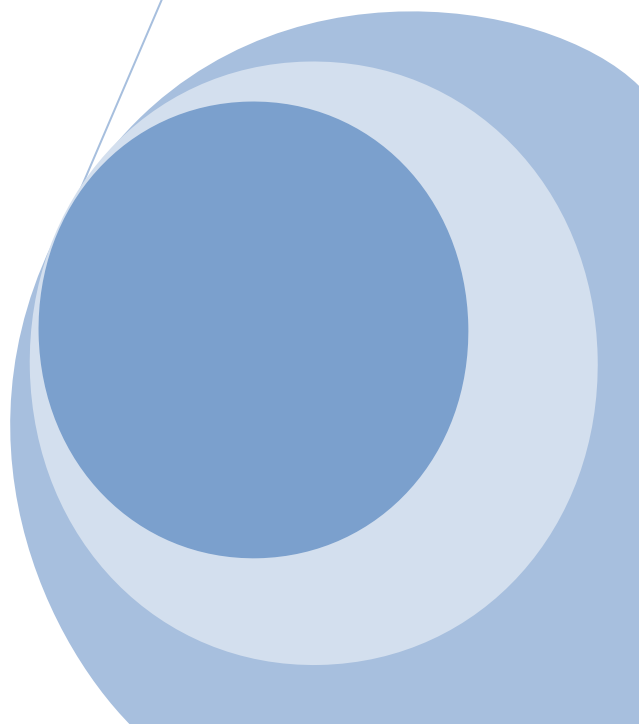
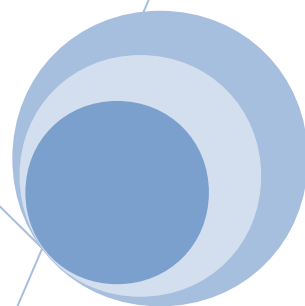
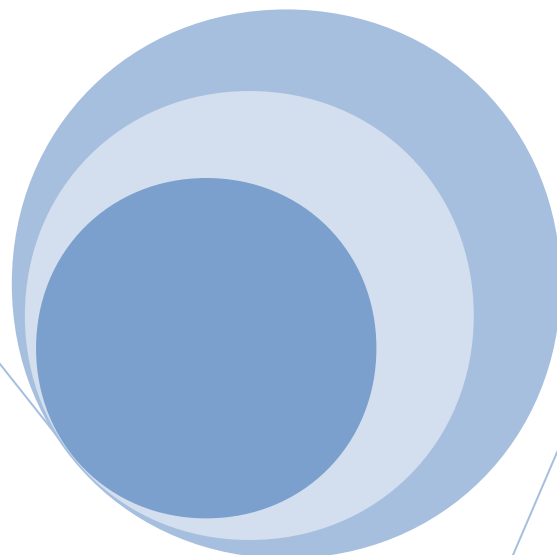


KHOA CÔNG NGHỆ THÔNG TIN  
ĐẠI HỌC THÁI NGUYÊN

**Đề tài thực tập chuyên ngành:**



# MỤC LỤC

<u>MỤC LỤC.....</u>	<u>2</u>
<u>DANH MỤC CÁC HÌNH.....</u>	<u>5</u>
<u>LỜI NÓI ĐẦU.....</u>	<u>6</u>
<u>TÓM TẮT NỘI DUNG.....</u>	<u>7</u>
<u>CHƯƠNG 1. TỔNG QUAN VỀ KIỂM THỬ PHẦN MỀM.....</u>	<u>9</u>
<u>1.1 Các khái niệm cơ bản về kiểm thử phần mềm.....</u>	<u>9</u>
<u>1.1.1 Kiểm thử phần mềm là gì?.....</u>	<u>9</u>
<u>1.1.2 Các phương pháp kiểm thử.....</u>	<u>10</u>
<u>1.1.2.1 Kiểm thử tĩnh – Static testing.....</u>	<u>10</u>

## THIẾT KẾ TEST-CASE TRONG KIỂM THỬ PHẦN MỀM

Sinh viên thực hiện : ***Phạm Thị Trang***

Lớp : ***ĐHCQ K4A***

Giáo viên hướng dẫn : ***Nguyễn Hồng Tân***

Bộ môn : ***Công nghệ phần mềm***

Thái Nguyên, tháng 9 năm 2009

1.1.2.2 Kiểm thử động – Dynamic testing.....	10
1.1.3 Các chiến lược kiểm thử.....	10
1.1.3.1 Kiểm thử hộp đen – Black box testing.....	11
1.1.3.2 Kiểm thử hộp trắng – White box testing.....	12
1.1.3.3 Kiểm thử hộp xám – Gray box testing.....	13
1.1.4 Các cấp độ kiểm thử phần mềm .....	13
1.1.4.1 Kiểm thử đơn vị – Unit test.....	14
1.1.4.2 Kiểm thử tích hợp – Intergration Test.....	15
1.1.4.3 Kiểm thử hệ thống – System Test .....	16
1.1.4.4 Kiểm thử chấp nhận sản phẩm – Acceptance Test.....	18
1.1.4.5 Một số cấp độ kiểm thử khác.....	19
1.1.5 Các phương pháp kiểm thử con người .....	20
1.1.5.1 Tổng duyệt – Walkthrough .....	21
1.1.5.2 Thanh tra mã nguồn – Code Inspection.....	21
1.2 Nguyên tắc kiểm thử phần mềm.....	22
CHƯƠNG 2. THIẾT KẾ TEST – CASE.....	23
2.1 Khái niệm.....	23
2.2 Vai trò của thiết kế test – case.....	23
2.3 Quy trình thiết kế test – case .....	23
2.3.1 Kiểm thử hộp trắng - Kiểm thử bao phủ logic .....	24
2.3.1.1 Bao phủ câu lệnh – Statement Coverage.....	25
2.3.1.2 Bao phủ quyết định – Decision coverage .....	26
2.3.1.3 Bao phủ điều kiện – Condition coverage .....	27
2.3.1.4 Bao phủ quyết định/điều kiện – Decision/condition coverage.....	29
2.3.1.5 Bao phủ đa điều kiện – Multiple condition coverage.....	30
2.3.2 Kiểm thử hộp đen.....	32
2.3.2.1 Phân lớp tương đương – Equivalence Patitioning.....	32
2.3.2.2 Phân tích giá trị biên – Boundary Value Analysis.....	35
2.3.2.3 Đồ thị nguyên nhân – kết quả - Cause & Effect Graphing.....	36
2.3.2.4 Đoán lỗi – Error Guessing.....	42
2.3.3 Chiến lược.....	42
CHƯƠNG 3. ÁP DỤNG.....	44
3.1 Đặc tả .....	44
3.2 Thiết kế test – case.....	46
3.2.1 Vẽ đồ thị nguyên nhân – kết quả.....	46
3.2.2 Phân lớp tương đương.....	49
3.2.2.1 Xác định các lớp tương đương .....	49
3.2.2.2 Xác định các ca kiểm thử.....	50
3.2.3 Phân tích giá trị biên.....	50
3.2.3.1 Xét các trạng thái đầu vào.....	50

3.2.3.2 Xét không gian kết quả.....	51
3.2.4 Các phương pháp hộp trắng.....	52
3.2.4.1 Bao phủ câu lệnh.....	52
3.2.4.2 Bao phủ quyết định.....	54
3.2.4.3 Bao phủ điều kiện.....	54
3.2.4.4 Bao phủ quyết định – điều kiện.....	54
3.2.4.5 Bao phủ đa điều kiện.....	55
KẾT LUẬN.....	56
TÀI LIỆU THAM KHẢO.....	57
NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN.....	58

# DANH MỤC CÁC HÌNH

Hình 1.1 Sơ đồ các cấp độ kiểm thử.....	13
Hình 2.1 Một chương trình nhỏ để kiểm thử .....	25
Hình 2.2 Mã máy cho chương trình trong Hình 2.1.....	29
Hình 2.3 Một mẫu cho việc liệt kê các lớp tương đương.....	33
Hình 2.4 Các ký hiệu đồ thị nguyên nhân – kết quả cơ bản.....	38
Hình 2.5 Các ký hiệu ràng buộc.....	39
Hình 2.6 Những xem xét được sử dụng khi dò theo đồ thị.....	40
Hình 3.1 Đồ thị nguyên nhân – kết quả:.....	47
Hình 3.2 Bảng quyết định.....	47

# LỜI NÓI ĐẦU

Trong ngành kỹ nghệ phần mềm, năm 1979, có một quy tắc nổi tiếng là: “Trong một dự án lập trình điển hình, thì xấp xỉ 50% thời gian và hơn 50% tổng chi phí được sử dụng trong kiểm thử các chương trình hay hệ thống đã được phát triển”. Và cho đến nay, sau gần một phần 3 thế kỷ, quy tắc đó vẫn còn đúng. Đã có rất nhiều ngôn ngữ, hệ thống phát triển mới với các công cụ tích hợp cho các lập trình viên sử dụng phát triển ngày càng linh động. Nhưng kiểm thử vẫn đóng vai trò hết sức quan trọng trong bất kỳ dự án phát triển phần mềm nào.

Rất nhiều các giáo sư, giảng viên đã từng than phiền rằng: “ Sinh viên của chúng ta tốt nghiệp và đi làm mà không có được những kiến thức thực tế cần thiết về cách để kiểm thử một chương trình. Hơn nữa, chúng ta hiếm khi có được những lời khuyên bổ ích để cung cấp trong các khóa học mở đầu về cách một sinh viên nên làm về kiểm thử và gỡ lỗi các bài tập của họ”.

Các tác giả của cuốn sách nổi tiếng “*The Art of Software Testing*” – *Nghệ thuật kiểm thử phần mềm*, Glenford J. Myers, Tom Badgett, Todd M. Thomas, Corey Sandler đã khẳng định trong cuốn sách của mình rằng: “ Hầu hết các thành phần quan trọng trong các thủ thuật của một nhà kiểm thử chương trình là kiến thức về cách để viết các ca kiểm thử có hiệu quả”. Việc xây dựng các test – case là một nhiệm vụ rất khó khăn. Để có thể xây dựng được tập các test – case hữu ích cho kiểm thử, chúng ta cần rất nhiều kiến thức và kinh nghiệm.

Đó là những lý do thúc đẩy em thực hiện đề tài này. Mục đích của đề tài là tìm hiểu những kiến thức tổng quan nhất về kiểm thử, và cách thiết kế test – case trong kiểm thử phần mềm. Việc thực hiện đề tài sẽ giúp em tìm hiểu sâu hơn và lĩnh vực rất hấp dẫn này, vận dụng được các kiến thức đã học để có thể thiết kế được các test – case một cách có hiệu quả và áp dụng vào những bài toán thực tế.

Bản báo cáo được hoàn thành dưới sự chỉ bảo tận tình của thầy giáo, ThS Nguyễn Hồng Tân, sự giúp đỡ nhiệt tình của các thầy cô trong bộ môn Công nghệ

phần mềm, và tất cả các bạn. Em hi vọng sẽ nhận được sự đóng góp ý kiến của các thầy cô và các bạn để bản báo cáo được hoàn thiện hơn. Những đóng góp đó sẽ là kinh nghiệm quý báu cho em. Và từ đó, em có thể tiếp tục phát triển đề tài này cho đợt thực tập tốt nghiệp và đồ án tốt nghiệp sắp tới, cũng như cho công việc trong tương lai.

*Em xin chân thành cảm ơn.*

Sinh viên

***Phạm Thị Trang***

## **TÓM TẮT NỘI DUNG**

Bản báo cáo được chia thành 3 chương với nội dung như sau:

- ***Chương 1: Tổng quan về kiểm thử phần mềm.***

Chương này là cái nhìn tổng quan về kiểm thử phần mềm: các khái niệm cơ bản về kiểm thử phần mềm, các quy tắc trong kiểm thử, và các phương pháp kiểm thử phần mềm tiêu biểu.

- ***Chương 2: Thiết kế test – case trong kiểm thử phần mềm.***

Trong chương này, em đi tìm hiểu các phương pháp thiết kế test – case có hiệu quả. Từ đó rút ra nhận xét về ưu nhược điểm của từng phương pháp.

- ***Chương 3: Áp dụng.***

Từ những phương pháp thiết kế test – case đã tìm hiểu trong Chương 2, em áp dụng để xây dựng tập các test –

case cho một bài toán cụ thể : Thiết kế các test – case cho chương trình “*Tam giác*”.



# CHƯƠNG 1. TỔNG QUAN VỀ KIỂM THỬ PHẦN MỀM

## 1.1 Các khái niệm cơ bản về kiểm thử phần mềm

### 1.1.1 Kiểm thử phần mềm là gì?

*Kiểm thử phần mềm là quá trình khảo sát một hệ thống hay thành phần dưới những điều kiện xác định, quan sát và ghi lại các kết quả, và đánh giá một khía cạnh nào đó của hệ thống hay thành phần đó. (Theo Bảng chú giải thuật ngữ chuẩn IEEE của Thuật ngữ kỹ nghệ phần mềm- IEEE Standard Glossary of Software Engineering Terminology).*

*Kiểm thử phần mềm là quá trình thực thi một chương trình với mục đích tìm lỗi. (Theo “The Art of Software Testing” – Nghệ thuật kiểm thử phần mềm).*

*Kiểm thử phần mềm là hoạt động khảo sát thực tiễn sản phẩm hay dịch vụ phần mềm trong đúng môi trường chúng dự định sẽ được triển khai nhằm cung cấp cho người có lợi ích liên quan những thông tin về chất lượng của sản phẩm hay dịch vụ phần mềm ấy. Mục đích của kiểm thử phần mềm là tìm ra các lỗi hay khiếm khuyết phần mềm nhằm đảm bảo hiệu quả hoạt động tối ưu của phần mềm trong nhiều ngành khác nhau. (Theo Bách khoa toàn thư mở Wikipedia).*

Có thể định nghĩa một cách dễ hiểu như sau: *Kiểm thử phần mềm là một tiến trình hay một tập hợp các tiến trình được thiết kế để đảm bảo mã hóa máy tính thực hiện theo cái mà chúng đã được thiết kế để làm, và không thực hiện bất cứ thứ gì không mong muốn. Đây là một pha quan trọng trong quá trình phát triển hệ thống, giúp cho người xây dựng hệ thống và khách hàng thấy được hệ thống mới đã đáp ứng yêu cầu đặt ra hay chưa?*

## 1.1.2 Các phương pháp kiểm thử

Có 2 phương pháp kiểm thử chính là: Kiểm thử tĩnh và Kiểm thử động.

### 1.1.2.1 *Kiểm thử tĩnh – Static testing*

Là phương pháp thử phần mềm đòi hỏi phải duyệt lại các yêu cầu và các đặc tả bằng tay, thông qua việc sử dụng giấy, bút để kiểm tra logic, lần từng chi tiết mà không cần chạy chương trình. Kiểu kiểm thử này thường được sử dụng bởi chuyên viên thiết kế người mà viết mã lệnh một mình.

Kiểm thử tĩnh cũng có thể được tự động hóa. Nó sẽ thực hiện kiểm tra toàn bộ bao gồm các chương trình được phân tích bởi một trình thông dịch hoặc biên dịch mà xác nhận tính hợp lệ về cú pháp của chương trình.

### 1.1.2.2 *Kiểm thử động – Dynamic testing*

Là phương pháp thử phần mềm thông qua việc dùng máy chạy chương trình để điều tra trạng thái tác động của chương trình. Đó là kiểm thử dựa trên các ca kiểm thử xác định bằng sự thực hiện của đối tượng kiểm thử hay chạy các chương trình. Kiểm thử động kiểm tra cách thức hoạt động động của mã lệnh, tức là kiểm tra sự phản ứng vật lý từ hệ thống tới các biến luôn thay đổi theo thời gian. Trong kiểm thử động, phần mềm phải thực sự được biên dịch và chạy. Kiểm thử động thực sự bao gồm làm việc với phần mềm, nhập các giá trị đầu vào và kiểm tra xem liệu đầu ra có như mong muốn hay không. Các phương pháp kiểm thử động gồm có kiểm thử Unit – *Unit Tests*, Kiểm thử tích hợp – *Integration Tests*, Kiểm thử hệ thống – *System Tests*, và Kiểm thử chấp nhận sản phẩm – *Acceptance Tests*.

## 1.1.3 Các chiến lược kiểm thử

Ba trong số những chiến lược kiểm thử thông dụng nhất bao gồm: Kiểm thử hộp đen, Kiểm thử hộp trắng, và Kiểm thử hộp xám.

### **1.1.3.1**      *Kiểm thử hộp đen – Black box testing*

Một trong những chiến lược kiểm thử quan trọng là kiểm thử *hộp đen*, *hướng dữ liệu*, hay *hướng vào/ra*. Kiểm thử hộp đen xem chương trình như là một “hộp đen”. Mục đích của bạn là hoàn toàn không quan tâm về cách cư xử và cấu trúc bên trong của chương trình. Thay vào đó, tập trung vào tìm các trường hợp mà chương trình không thực hiện theo các đặc tả của nó.

Theo hướng tiếp cận này, dữ liệu kiểm tra được lấy chỉ từ các đặc tả.

#### *Các phương pháp kiểm thử hộp đen*

- Phân lớp tương đương – *Equivalence partitioning*.
- Phân tích giá trị biên – *Boundary value analysis*.
- Kiểm thử mọi cặp – *All-pairs testing*.
- Kiểm thử fuzz – *Fuzz testing*.
- Kiểm thử dựa trên mô hình – *Model-based testing*.
- Ma trận dấu vết – *Traceability matrix*.
- Kiểm thử thăm dò – *Exploratory testing*.
- Kiểm thử dựa trên đặc tả – *Specification-base testing*.

Kiểm thử dựa trên đặc tả tập trung vào kiểm tra tính thiết thực của phần mềm theo những yêu cầu thích hợp. Do đó, kiểm thử viên nhập dữ liệu vào, và chỉ thấy dữ liệu ra từ đối tượng kiểm thử. Mức kiểm thử này thường yêu cầu các ca kiểm thử triệt để được cung cấp cho kiểm thử viên mà khi đó có thể xác minh là đối với dữ liệu đầu vào đã cho, giá trị đầu ra (hay cách thức hoạt động) có giống với giá trị mong muốn đã được xác định trong ca kiểm thử đó hay không. Kiểm thử dựa trên đặc tả là cần thiết, nhưng không đủ để để ngăn chặn những rủi ro chắc chắn.

#### *Ưu, nhược điểm*

Kiểm thử hộp đen không có mối liên quan nào tới mã lệnh, và kiểm thử viên chỉ rất đơn giản tâm niệm là: một mã lệnh phải có lỗi. Sử dụng nguyên tắc “Hãy đòi hỏi và bạn sẽ được nhận”, những kiểm thử viên hộp đen tìm ra lỗi mà những lập trình viên đã không tìm ra. Nhưng, mặt khác, người ta cũng nói kiểm thử hộp đen “giống như là đi trong bóng tối mà không có đèn vậy”, bởi vì kiểm thử viên không biết các phần mềm được kiểm tra thực sự được xây dựng như thế nào. Đó là lý do mà có nhiều trường hợp mà một kiểm thử viên hộp đen viết rất nhiều ca kiểm thử để kiểm tra một thứ gì đó mà đáng lẽ có thể chỉ cần kiểm tra bằng 1 ca kiểm thử duy nhất, và/hoặc một số phần của chương trình không được kiểm tra chút nào.

Do vậy, kiểm thử hộp đen có ưu điểm của “một sự đánh giá khách quan”, mặt khác nó lại có nhược điểm của “thăm dò mù”.

### ***1.1.3.2 Kiểm thử hộp trắng – White box testing***

Là một chiến lược kiểm thử khác, trái ngược hoàn toàn với kiểm thử hộp đen, kiểm thử hộp trắng hay kiểm thử hướng logic cho phép bạn khảo sát cấu trúc bên trong của chương trình. Chiến lược này xuất phát từ dữ liệu kiểm thử bằng sự kiểm thử tính logic của chương trình. Kiểm thử viên sẽ truy cập vào cấu trúc dữ liệu và giải thuật bên trong chương trình (và cả mã lệnh thực hiện chúng).

#### ***Các phương pháp kiểm thử hộp trắng***

- ***Kiểm thử giao diện lập trình ứng dụng - API testing (application programming interface):*** là phương pháp kiểm thử của ứng dụng sử dụng các API công khai và riêng tư.
- ***Bao phủ mã lệnh – Code coverage:*** tạo các kiểm tra để đáp ứng một số tiêu chuẩn về bao phủ mã lệnh.
- ***Các phương pháp gán lỗi – Fault injection.***
- ***Các phương pháp kiểm thử hoán chuyển – Mutation testing methods.***

- **Kiểm thử tĩnh – Static testing:** kiểm thử hộp trắng bao gồm mọi kiểm thử tĩnh.

Phương pháp kiểm thử hộp trắng cũng có thể được sử dụng để đánh giá sự hoàn thành của một bộ kiểm thử mà được tạo cùng với các phương pháp kiểm thử hộp đen. Điều này cho phép các nhóm phần mềm khảo sát các phần của 1 hệ thống ít khi được kiểm tra và đảm bảo rằng những điểm chức năng quan trọng nhất đã được kiểm tra.

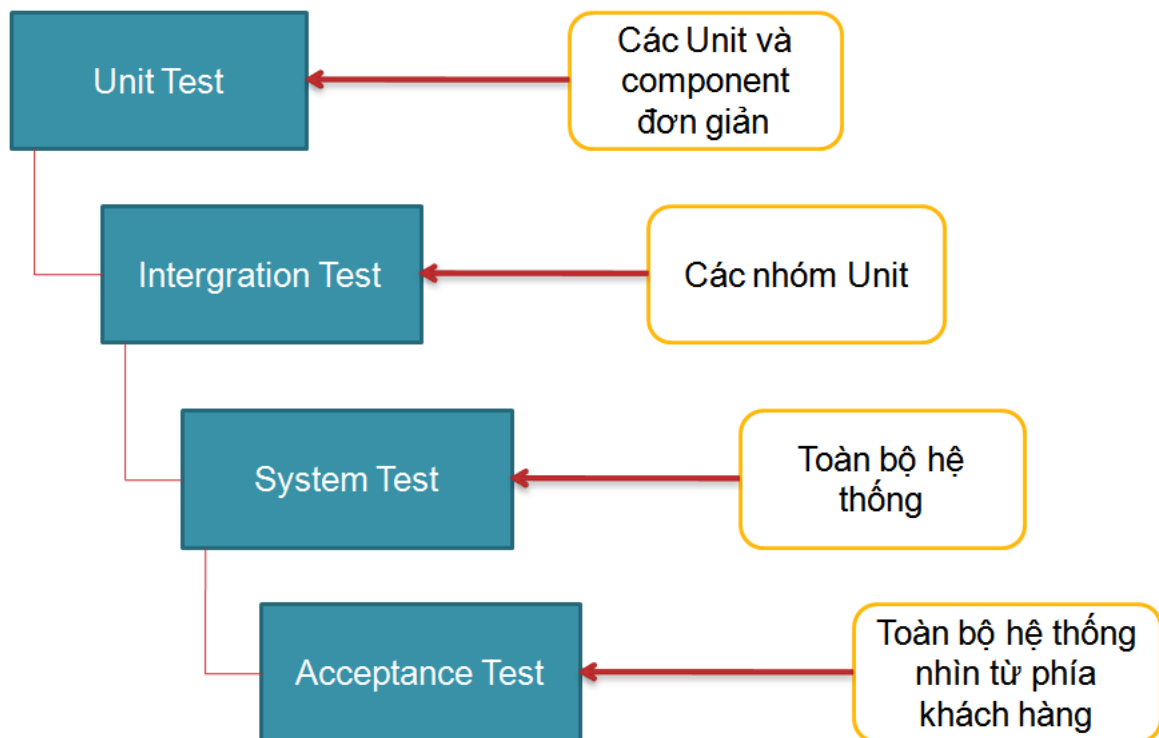
### **1.1.3.3 Kiểm thử hộp xám – Gray box testing**

Kiểm thử hộp xám đòi hỏi phải có sự truy cập tới cấu trúc dữ liệu và giải thuật bên trong cho những mục đích thiết kế các ca kiểm thử, nhưng là kiểm thử ở mức người sử dụng hay mức hộp đen. Việc thao tác tới dữ liệu đầu vào và định dạng dữ liệu đầu ra là không rõ ràng, giống như một chiếc “*hộp xám*”, bởi vì đầu vào và đầu ra rõ ràng là ở bên ngoài “*hộp đen*” mà chúng ta vẫn gọi về hệ thống được kiểm tra. Sự khác biệt này đặc biệt quan trọng khi quản lý kiểm thử tích hợp – *Intergartion testing* giữa 2 modul mã lệnh được viết bởi hai chuyên viên thiết kế khác nhau, trong đó chỉ giao diện là được đưa ra để kiểm thử. Kiểm thử hộp xám có thể cũng bao gồm cả thiết kế đối chiếu để quyết định, ví dụ, giá trị biên hay thông báo lỗi.

### **1.1.4 Các cấp độ kiểm thử phần mềm**

Kiểm thử phần mềm gồm có các cấp độ: Kiểm thử đơn vị, Kiểm thử tích hợp, Kiểm thử hệ thống và Kiểm thử chấp nhận sản phẩm.

**Hình 1.1 Sơ đồ các cấp độ kiểm thử**



#### 1.1.4.1 Kiểm thử đơn vị – Unit test

Một đơn vị là một thành phần phần mềm nhỏ nhất mà ta có thể kiểm thử được. Ví dụ, các hàm (*Function*), thủ tục (*Procedure*), lớp (*Class*) hay phương thức (*Method*) đều có thể được xem là Unit.

Vì Unit được chọn để kiểm tra thường có kích thước nhỏ và chức năng hoạt động đơn giản, chúng ta không khó khăn gì trong việc tổ chức kiểm thử, ghi nhận và phân tích kết quả kiểm thử. Nếu phát hiện lỗi, việc xác định nguyên nhân và khắc phục cũng tương đối dễ dàng vì chỉ khoanh vùng trong một đơn thể Unit đang kiểm tra. Một nguyên lý đúc kết từ thực tiễn: thời gian tốn cho Unit Test sẽ được đền bù bằng việc tiết kiệm rất nhiều thời gian và chi phí cho việc kiểm thử và sửa lỗi ở các mức kiểm thử sau đó.

Unit Test thường do lập trình viên thực hiện. Công đoạn này cần được thực hiện càng sớm càng tốt trong giai đoạn viết code và xuyên suốt chu kỳ phát triển phần mềm. Thông thường, Unit Test đòi hỏi kiểm thử viên có kiến thức về thiết kế và code của chương trình. Mục đích của Unit Test là bảo đảm thông tin được xử lý

và xuất (khỏi Unit) là chính xác, trong mối tương quan với dữ liệu nhập và chức năng của Unit. Điều này thường đòi hỏi tất cả các nhánh bên trong Unit đều phải được kiểm tra để phát hiện nhánh phát sinh lỗi. Một nhánh thường là một chuỗi các lệnh được thực thi trong một Unit. Ví dụ: chuỗi các lệnh sau điều kiện If và nằm giữa then ... else là một nhánh. Thực tế việc chọn lựa các nhánh để đơn giản hóa việc kiểm thử và quét hết Unit đòi hỏi phải có kỹ thuật, đôi khi phải dùng thuật toán để chọn lựa.

Cùng với các mục kiểm thử khác, Unit Test cũng đòi hỏi phải chuẩn bị trước các ca kiểm thử (*Test case*) hoặc kịch bản kiểm thử (*Test script*), trong đó chỉ định rõ dữ liệu đầu vào, các bước thực hiện và dữ liệu đầu ra mong muốn. Các Test case và Test script này nên được giữ lại để tái sử dụng.

#### **1.1.4.2      *Kiểm thử tích hợp – Intergration Test***

Integration test kết hợp các thành phần của một ứng dụng và kiểm thử như một ứng dụng đã hoàn thành. Trong khi Unit Test kiểm tra các thành phần và Unit riêng lẻ thì Integration Test kết hợp chúng lại với nhau và kiểm tra sự giao tiếp giữa chúng.

Hai mục tiêu chính của Integration Test:

- Phát hiện lỗi giao tiếp xảy ra giữa các Unit.
- Tích hợp các Unit đơn lẻ thành các hệ thống nhỏ (*Subsystem*) và cuối cùng là nguyên hệ thống hoàn chỉnh (*System*) chuẩn bị cho kiểm thử ở mức hệ thống (*System Test*).

Trong Unit Test, lập trình viên cố gắng phát hiện lỗi liên quan đến chức năng và cấu trúc nội tại của Unit. Có một số phép kiểm thử đơn giản trên giao tiếp giữa Unit với các thành phần liên quan khác, tuy nhiên mọi giao tiếp liên quan đến Unit chỉ thật sự được kiểm tra đầy đủ khi các Unit tích hợp với nhau trong khi thực hiện Integration Test.

Trừ một số ít ngoại lệ, Integration Test chỉ nên thực hiện trên những Unit đã được kiểm tra cẩn thận trước đó bằng Unit Test, và tất cả các lỗi mức Unit đã được sửa chữa. Một số người hiểu sai rằng Unit một khi đã qua giai đoạn Unit Test với các giao tiếp giả lập thì không cần phải thực hiện Integration Test nữa. Thực tế việc tích hợp giữa các Unit dẫn đến những tình huống hoàn toàn khác. Một chiến lược cần quan tâm trong Integration Test là nên tích hợp dần từng Unit. Một Unit tại một thời điểm được tích hợp vào một nhóm các Unit khác đã tích hợp trước đó và đã hoàn tất các đợt Integration Test trước đó. Lúc này, ta chỉ cần kiểm thử giao tiếp của Unit mới thêm vào với hệ thống các Unit đã tích hợp trước đó, điều này sẽ làm cho số lượng can kiểm thử giảm đi rất nhiều, và sai sót sẽ giảm đáng kể.

Có 4 loại kiểm thử trong Integration Test:

- **Kiểm thử cấu trúc (Structure Test):** Tương tự White Box Test, kiểm thử cấu trúc nhằm bảo đảm các thành phần bên trong của một chương trình chạy đúng và chú trọng đến hoạt động của các thành phần cấu trúc nội tại của chương trình chẳng hạn các câu lệnh và nhánh bên trong.
- **Kiểm thử chức năng (Functional Test):** Tương tự Black Box Test, kiểm thử chức năng chỉ chú trọng đến chức năng của chương trình, mà không quan tâm đến cấu trúc bên trong, chỉ khảo sát chức năng của chương trình theo yêu cầu kỹ thuật.
- **Kiểm thử hiệu năng (Performance Test):** Kiểm thử việc vận hành của hệ thống.
- **Kiểm thử khả năng chịu tải (Stress Test):** Kiểm thử các giới hạn của hệ thống.

#### **1.1.4.3 Kiểm thử hệ thống – System Test**

Mục đích System Test là kiểm thử thiết kế và toàn bộ hệ thống (sau khi tích hợp) có thỏa mãn yêu cầu đặt ra hay không.



System Test bắt đầu khi tất cả các bộ phận của phần mềm đã được tích hợp thành công. Thông thường loại kiểm thử này tốn rất nhiều công sức và thời gian. Trong nhiều trường hợp, việc kiểm thử đòi hỏi một số thiết bị phụ trợ, phần mềm hoặc phần cứng đặc thù, đặc biệt là các ứng dụng thời gian thực, hệ thống phân bố, hoặc hệ thống nhúng. Ở mức độ hệ thống, người kiểm thử cũng tìm kiếm các lỗi, nhưng trọng tâm là đánh giá về hoạt động, thao tác, sự tin cậy và các yêu cầu khác liên quan đến chất lượng của toàn hệ thống.

Điểm khác nhau then chốt giữa Integration Test và System Test là System Test chú trọng các hành vi và lỗi trên toàn hệ thống, còn Integration Test chú trọng sự giao tiếp giữa các đơn thể hoặc đối tượng khi chúng làm việc cùng nhau. Thông thường ta phải thực hiện Unit Test và Integration Test để bảo đảm mọi Unit và sự tương tác giữa chúng hoạt động chính xác trước khi thực hiện System Test.

Sau khi hoàn thành Integration Test, một hệ thống phần mềm đã được hình thành cùng với các thành phần đã được kiểm tra đầy đủ. Tại thời điểm này, lập trình viên hoặc kiểm thử viên bắt đầu kiểm thử phần mềm như một hệ thống hoàn chỉnh. Việc lập kế hoạch cho System Test nên bắt đầu từ giai đoạn hình thành và phân tích các yêu cầu.

System Test kiểm thử cả các hành vi chức năng của phần mềm lẫn các yêu cầu về chất lượng như độ tin cậy, tính tiện lợi khi sử dụng, hiệu năng và bảo mật. Mức kiểm thử này đặc biệt thích hợp cho việc phát hiện lỗi giao tiếp với phần mềm hoặc phần cứng bên ngoài, chẳng hạn các lỗi "tắc nghẽn" (deadlock) hoặc chiếm dụng bộ nhớ. Sau giai đoạn System Test, phần mềm thường đã sẵn sàng cho khách hàng hoặc người dùng cuối cùng kiểm thử chấp nhận sản phẩm (*Acceptance Test*) hoặc dùng thử (*Alpha/Beta Test*).

Đòi hỏi nhiều công sức, thời gian và tính chính xác, khách quan, System Test thường được thực hiện bởi một nhóm kiểm thử viên hoàn toàn độc lập với nhóm phát triển dự án. Bản thân System Test lại gồm nhiều loại kiểm thử khác nhau, phổ biến nhất gồm:

- **Kiểm thử chức năng (Functional Test):** Bảo đảm các hành vi của hệ thống thỏa mãn đúng yêu cầu thiết kế.
- **Kiểm thử hiệu năng (Performance Test):** Bảo đảm tối ưu việc phân bổ tài nguyên hệ thống (ví dụ bộ nhớ) nhằm đạt các chỉ tiêu như thời gian xử lý hay đáp ứng câu truy vấn...
- **Kiểm thử khả năng chịu tải (Stress Test hay Load Test):** Bảo đảm hệ thống vận hành đúng dưới áp lực cao (ví dụ nhiều người truy xuất cùng lúc). Stress Test tập trung vào các trạng thái tới hạn, các "điểm chết", các tình huống bất thường như đang giao dịch thì ngắt kết nối (xuất hiện nhiều trong kiểm tra thiết bị như POS, ATM...)...
- **Kiểm thử cấu hình (Configuration Test).**
- **Kiểm thử bảo mật (Security Test):** Bảo đảm tính toàn vẹn, bảo mật của dữ liệu và của hệ thống.
- **Kiểm thử khả năng phục hồi (Recovery Test):** Bảo đảm hệ thống có khả năng khôi phục trạng thái ổn định trước đó trong tình huống mất tài nguyên hoặc dữ liệu; đặc biệt quan trọng đối với các hệ thống giao dịch như ngân hàng trực tuyến...

Nhìn từ quan điểm người dùng, các cấp độ kiểm thử trên rất quan trọng: Chúng bảo đảm hệ thống đủ khả năng làm việc trong môi trường thực.

Lưu ý là không nhất thiết phải thực hiện tất cả các loại kiểm thử nêu trên. Tùy yêu cầu và đặc trưng của từng hệ thống, tùy khả năng và thời gian cho phép của dự án, khi lập kế hoạch, người Quản lý dự án sẽ quyết định áp dụng những loại kiểm thử nào.

#### **1.1.4.4      Kiểm thử chấp nhận sản phẩm – Acceptance Test**

Thông thường, sau giai đoạn System Test là Acceptance Test, được khách hàng thực hiện (hoặc ủy quyền cho một nhóm thứ ba thực hiện). Mục đích của

Acceptance Test là để chứng minh phần mềm thỏa mãn tất cả yêu cầu của khách hàng và khách hàng chấp nhận sản phẩm (và trả tiền thanh toán hợp đồng).

Acceptance Test có ý nghĩa hết sức quan trọng, mặc dù trong hầu hết mọi trường hợp, các phép kiểm thử của System Test và Acceptance Test gần như tương tự, nhưng bản chất và cách thức thực hiện lại rất khác biệt.

Đối với những sản phẩm dành bán rộng rãi trên thị trường cho nhiều người sử dụng, thông thường sẽ thông qua hai loại kiểm thử gọi là kiểm thử Alpha – ***Alpha Test*** và kiểm thử Beta – ***Beta Test***. Với Alpha Test, người dùng kiểm thử phần mềm ngay tại nơi phát triển phần mềm, lập trình viên sẽ ghi nhận các lỗi hoặc phản hồi, và lên kế hoạch sửa chữa. Với Beta Test, phần mềm sẽ được gửi tới cho người dùng để kiểm thử ngay trong môi trường thực, lỗi hoặc phản hồi cũng sẽ gửi ngược lại cho lập trình viên để sửa chữa.

Thực tế cho thấy, nếu khách hàng không quan tâm và không tham gia vào quá trình phát triển phần mềm thì kết quả Acceptance Test sẽ sai lệch rất lớn, mặc dù phần mềm đã trải qua tất cả các kiểm thử trước đó. Sự sai lệch này liên quan đến việc hiểu sai yêu cầu cũng như sự mong chờ của khách hàng. Ví dụ đôi khi một phần mềm xuất sắc vượt qua các phép kiểm thử về chức năng thực hiện bởi nhóm thực hiện dự án, nhưng khách hàng khi kiểm thử sau cùng vẫn thất vọng vì bố cục màn hình nghèo nàn, thao tác không tự nhiên, không theo tập quán sử dụng của khách hàng v.v...

Gắn liền với giai đoạn Acceptance Test thường là một nhóm những dịch vụ và tài liệu đi kèm, phổ biến như hướng dẫn cài đặt, sử dụng v.v... Tất cả tài liệu đi kèm phải được cập nhật và kiểm thử chặt chẽ.

#### ***1.1.4.5 Một số cấp độ kiểm thử khác***

Ngoài các cấp độ trên, còn một số cấp độ kiểm thử khác như:

#### ***Kiểm thử hồi quy – Regression Testing:***

Theo chuẩn IEEE610.12-90, kiểm thử hồi quy là “sự kiểm tra lại có lựa chọn của một hệ thống hay thành phần để xác minh là những sự thay đổi không gây ra những hậu quả không mong muốn...”. Trên thực tế, quan niệm này là chỉ ra rằng phần mềm mà đã qua được các kiểm tra trước đó vẫn có thể được kiểm tra lại. Beizer định nghĩa đó là sự lặp lại các kiểm tra để chỉ ra rằng cách hoạt động của phần mềm không bị thay đổi, ngoại trừ tới mức như yêu cầu. Hiển nhiên là sự thỏa hiệp phải được thực hiện giữa sự đảm bảo được đưa ra bởi kiểm thử hồi quy mỗi lần thực hiện một sự thay đổi và những tài nguyên được yêu cầu thực hiện điều đó.

### ***Kiểm thử tính đúng – Correctness testing:***

Tính đúng đắn là yêu cầu tối thiểu của phần mềm, là mục đích chủ yếu của kiểm thử. Kiểm thử tính đúng sẽ cần một kiểu người đáng tin nào đó, để chỉ ra cách hoạt động đúng đắn từ cách hoạt động sai lầm. Kiểm thử viên có thể biết hoặc không biết các chi tiết bên trong của các modun phần mềm được kiểm thử, ví dụ luồng điều khiển, luồng dữ liệu, v.v .... Do đó, hoặc là quan điểm hộp trắng, hoặc là quan điểm hộp đen có thể được thực hiện trong kiểm thử phần mềm.

#### **1.1.5 Các phương pháp kiểm thử con người**

Hai phương pháp kiểm thử con người chủ yếu là *Code Inspections* và *Walkthroughs*. Hai phương pháp này bao gồm một nhóm người đọc và kiểm tra theo mã lệnh của chương trình. Mục tiêu của chúng là để tìm ra lỗi mà không gỡ lỗi.

Một *Inspection* hay *Walkthrough* là 1 sự cải tiến của phương pháp kiểm tra mà lập trình viên đọc chương trình của họ trước khi kiểm thử nó. *Inspections* và *Walkthroughs* hiệu quả hơn là bởi vì những người khác sẽ kiểm thử chương trình tốt hơn chính tác giả của chương trình đó.

*Inspections/Walkthroughs* và kiểm thử bằng máy tính bổ sung cho nhau. Hiệu quả tìm lỗi sẽ kém đi nếu thiếu đi 1 trong 2 phương pháp. Và đối với việc sửa đổi

chương trình cũng nên sử dụng các phương pháp kiểm thử này cũng như các kỹ thuật kiểm thử hồi quy.

#### **1.1.5.1 Tổng duyệt – Walkthrough**

*Walkthrough* là một thuật ngữ mô tả sự xem xét kỹ lưỡng của một quá trình ở mức trừu tượng trong đó nhà thiết kế hay lập trình viên lãnh đạo các thành viên trong nhóm và những người có quan tâm khác thông qua một sản phẩm phần mềm, và những người tham gia đặt câu hỏi, và ghi chú những lỗi có thể có, sự vi phạm các chuẩn phát triển và các vấn đề khác. Walkthrough mã lệnh là 1 tập các thủ tục và các công nghệ dò lỗi cho việc đọc nhóm mã lệnh. Trong một *Walkthrough*, nhóm các nhà phát triển – với 3 hoặc 4 thành viên là tốt nhất – thực hiện xét duyệt lại. Chỉ 1 trong các thành viên là tác giả của chương trình.

Một ưu điểm khác của *walkthroughs*, hiệu quả trong chi phí gỡ lỗi, là 1 thực tế mà khi một lỗi được tìm thấy, nó thường được định vị chính xác trong mã lệnh. Thêm vào đó, phương pháp này thường tìm ra 1 tập các lỗi, cho phép sau đó các lỗi đó được sửa tất cả với nhau. Mặt khác, kiểm thử dựa trên máy tính, chỉ tìm ra triệu chứng của lỗi (chương trình không kết thúc hoặc đưa ra kết quả vô nghĩa), và các lỗi thường được tìm ra và sửa lần lượt từng lỗi một.

#### **1.1.5.2 Thanh tra mã nguồn – Code Inspection**

Thanh tra mã nguồn là 1 tập hợp các thủ tục và các kỹ thuật dò lỗi cho việc đọc các nhóm mã lệnh. Một nhóm kiểm duyệt thường gồm 4 người. Một trong số đó đóng vai trò là người điều tiết – một lập trình viên lão luyện và không được là tác giả của chương trình và phải không quen với các chi tiết của chương trình. Người điều tiết có nhiệm vụ: phân phối nguyên liệu và lập lịch cho các buổi kiểm duyệt, chỉ đạo phiên làm việc, ghi lại tất cả các lỗi được tìm thấy và đảm bảo là các lỗi sau đó được sửa. Thành viên thứ hai là một lập trình viên. Các thành viên còn lại

trong nhóm thường là nhà thiết kế của chương trình ( nếu nhà thiết kế khác lập trình viên) và một chuyên viên kiểm thử.

## **1.2 Nguyên tắc kiểm thử phần mềm**

Để kiểm thử đạt hiệu quả thì khi tiến hành kiểm thử phần mềm cần phải tuân thủ một số quy tắc sau:

***Quy tắc 1: Một phần quan trọng của 1 ca kiểm thử là định nghĩa của đầu ra hay kết quả mong muốn.***

***Quy tắc 2: Lập trình viên nên tránh tự kiểm tra chương trình của mình.***

***Quy tắc 3: Nhóm lập trình không nên kiểm thử chương trình của chính họ.***

***Quy tắc 4: Kiểm tra thấu đáo mọi kết quả của mỗi kiểm tra.***

***Quy tắc 5: Các ca kiểm thử phải được viết cho các trạng thái đầu vào không hợp lệ và không mong muốn, cũng như cho các đầu vào hợp lệ và mong muốn.***

***Quy tắc 6: Khảo sát 1 chương trình để xem liệu chương trình có thực hiện cái mà nó cần thực hiện chỉ là 1 phần, phần còn lại là xem liệu chương trình có thực hiện cái mà nó không cần phải thực hiện hay không.***

***Quy tắc 7: Tránh các ca kiểm thử băng quơ trừ khi chương trình thực sự là 1 chương trình băng quơ.***

***Quy tắc 8: Không dự kiến kết quả của kiểm thử theo giả thiết ngầm là không tìm thấy lỗi.***

***Quy tắc 9: Xác suất tồn tại lỗi trong 1 đoạn chương trình là tương ứng với số lỗi đã tìm thấy trong đoạn đó.***

***Quy tắc 10: Kiểm thử là 1 nhiệm vụ cực kỳ sáng tạo và có tính thử thách trí tuệ.***

## CHƯƠNG 2. THIẾT KẾ TEST – CASE

### 2.1 Khái niệm

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.

### 2.2 Vai trò của thiết kế test – case

- 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.

### 2.3 Quy trình thiết kế test – case

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:

<i>Hộp đen</i>	<i>Hộp trắng</i>
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.

### 2.3.1 Kiểm thử hộp trắng - Kiểm thử bao phủ logic

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. Các tiêu chuẩn trong kiểm thử bao phủ logic gồm có:



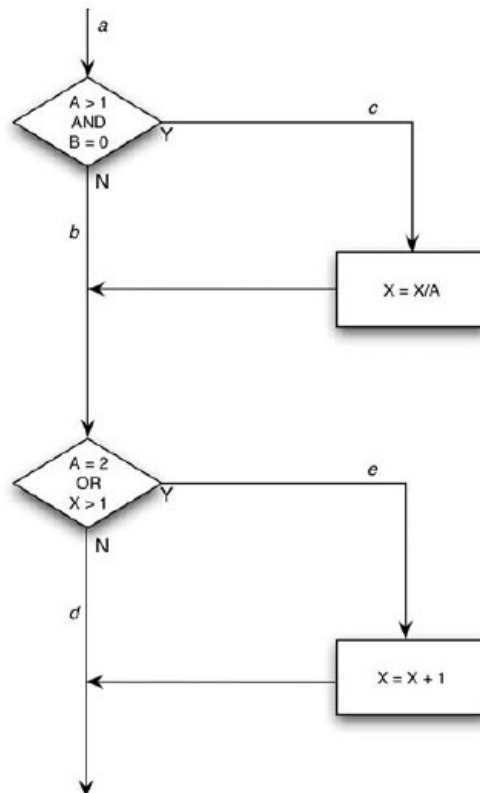
### 2.3.1.1 Bao phủ câu lệnh – Statement Coverage

**Tư tưởng:** Thực hiện mọi câu lệnh trong chương trình ít nhất 1 lần.

Xét ví dụ với đoạn mã lệnh JAVA sau:

```
public void foo (int a, int b, int x){  
    if (a>1 && b==0) {  
        x=x/a;}  
    if (a==2||x>1){  
        x=x+1;  
    }  
}
```

**Hình 2.1** Một chương trình nhỏ để kiểm thử



Bạn có thể thực hiện mọi câu lệnh bằng việc viết 1 ca kiểm thử đơn đi qua đường *ace*. Tức là, bằng việc đặt  $A=2$ ,  $B=0$  và  $X=3$  tại điểm *a*, mỗi câu lệnh sẽ được thực hiện 1 lần (thực tế,  $X$  có thể được gán bất kỳ giá trị nào).

Thường tiêu chuẩn này khá kém. Ví dụ, có lẽ nếu quyết định đầu tiên là phép *or* chứ không phải phép *and* thì lỗi này sẽ không được phát hiện. Hay nếu quyết định thứ hai mà bắt đầu với  $x>0$ , lỗi này cũng sẽ không được tìm ra. Cũng vậy, có 1 đường đi qua chương trình mà ở đó  $x$  không thay đổi (đường đi *abd*). Nếu đây là 1 lỗi, thì lỗi này có thể không tìm ra. Hay nói cách khác, tiêu chuẩn bao phủ câu lệnh quá yếu đến nỗi mà nó thường là vô ích.

### 2.3.1.2 Bao phủ quyết định – *Decision coverage*

**Tư tưởng:** 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.

Các ví dụ về câu lệnh rẽ nhánh hay quyết định là các câu lệnh switch, do-while, và if-else. Các câu lệnh đa đường GOTO thường sử dụng trong một số ngôn ngữ lập trình như FORTRAN.

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 1 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:

- Những chương trình không có quyết định.
- 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.
- 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.

Phương pháp này chỉ xem xét những quyết định hay những sự phân nhánh 2 đường và phải được sửa đổi cho những chương trình có chứa những quyết định đa đường. Ví dụ, các chương trình JAVA có chứa các lệnh *select (case)*, các chương trình FORTRAN chứa các lệnh số học (ba đường) *if* hoặc các lệnh tính toán hay số học *GOTO*, và các chương trình COBOL chứa các lệnh *GOTO* biến đổi hay các lệnh *GO-TO-DEPENDING-ON* (các lệnh goto phụ thuộc). Với những chương trình như vậy, tiêu chuẩn này đang sử dụng mỗi kết luận có thể của tất cả các quyết định ít nhất 1 lần và gọi mỗi điểm vào tới chương trình hay thường trình con ít nhất 1 lần.

Trong hình 2.1, bao phủ quyết định có thể đạt được bởi ít nhất 2 ca kiểm thử bao phủ các đường *ace* và *abd* hoặc *acd* và *abe*. Nếu chúng ta chọn khả năng thứ hai, thì 2 đầu vào test-case là  $A=3, B=0, X=3$  và  $A=2, B=1, X=1$ .

Bao phủ quyết định là 1 tiêu chuẩn mạnh hơn bao phủ câu lệnh, nhưng vẫn khá yếu. Ví dụ, chỉ có 50% cơ hội là chúng ta sẽ tìm ra con đường trong đó  $x$  không bị thay đổi (ví dụ, chỉ khi bạn chọn khả năng thứ nhất). Nếu quyết định thứ hai bị lỗi (nếu như đáng lẽ phải nói là  $x < 1$  thay vì  $x > 1$ ), lỗi này sẽ không được phát hiện bằng 2 ca kiểm thử trong ví dụ trước.

### 2.3.1.3 Bao phủ điều kiện – *Condition coverage*

**Tư tưởng:** 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 một lần.

Vì vậy, như với bao phủ quyết định, thì bao phủ điều kiện không phải luôn luôn dẫn tới việc thực thi mỗi câu lệnh. Thêm vào đó, trong tiêu chuẩn bao phủ điều

kiện, mỗi điểm vào chương trình hay thường trình con, cũng như các ON-unit, được gọi ít nhất 1 lần. Ví dụ, câu lệnh rẽ nhánh do  $k=0$  to 50 while  $(j+k < \text{quest})$  có chứa 2 điều kiện là  $k \leq 50$ , và  $j+k < \text{quest}$ . Do đó, các ca kiểm thử sẽ được yêu cầu cho những tình huống  $k \leq 50$ ,  $k > 50$  (để đến lần lặp cuối cùng của vòng lặp),  $j+k < \text{quest}$ , và  $j+k \geq \text{quest}$ .

Hình 2.1 có 4 điều kiện:  $A > 1$ ,  $B = 0$ ,  $A = 2$ ,  $X > 1$ . Do đó các ca kiểm thử đầy đủ là cần thiết để thúc đẩy những trạng thái mà  $A > 1$ ,  $A \leq 1$ ,  $B = 0$  và  $B \neq 0$  có mặt tại điểm  $a$  và  $A = 2$ ,  $A \neq 2$ ,  $X > 1$ ,  $X \leq 1$  có mặt tại điểm  $b$ . Số lượng đầy đủ các ca kiểm thử thỏa mãn tiêu chuẩn và những đường đi mà được đi qua bởi mỗi ca kiểm thử là:

1.  $A=2, B=0, X=4$     *ace*
2.  $A=1, B=1, X=1$     *abd*

Chú ý là, mặc dù cùng số lượng các ca kiểm thử được tạo ra cho ví dụ này, nhưng bao phủ điều kiện thường tốt hơn bao phủ quyết định là vì nó có thể (nhưng không luôn luôn) gây ra mọi điều kiện riêng trong 1 quyết định để thực hiện với cả hai kết quả, trong khi bao phủ quyết định lại không. Ví dụ trong cùng câu lệnh rẽ nhánh: *DO K=0 TO 50 WHILE (J+K < QUEST)* là 1 nhánh 2 đường (thực hiện thân vòng lặp hay bỏ qua nó). Nếu bạn đang sử dụng kiểm thử quyết định, thì tiêu chuẩn này có thể được thỏa mãn bằng cách cho vòng lặp chạy từ  $K=0$  tới 51, mà chưa từng kiểm tra trường hợp trong đó mệnh đề *WHILE* bị sai. Tuy nhiên, với tiêu chuẩn bao phủ điều kiện, 1 ca kiểm thử sẽ cần phải cho ra 1 kết quả sai cho những điều kiện  $J+K < \text{QUEST}$ .

Mặc dù nếu mới nhìn thoáng qua, tiêu chuẩn bao phủ điều kiện xem ra thỏa mãn tiêu chuẩn bao phủ quyết định, nhưng không phải lúc nào cũng vậy. Nếu quyết định *IF (A&B)* được kiểm tra, thì tiêu chuẩn bao phủ điều kiện sẽ cho phép bạn viết 2 ca kiểm thử - A đúng, B sai, và A sai, B đúng – nhưng điều này sẽ không làm cho mệnh đề *THEN* của câu lệnh *IF* được thực hiện.

Ví dụ, 2 ca kiểm thử khác:

1.  $A=1, B=0, X=3$

2.  $A=2, B=1, X=1$

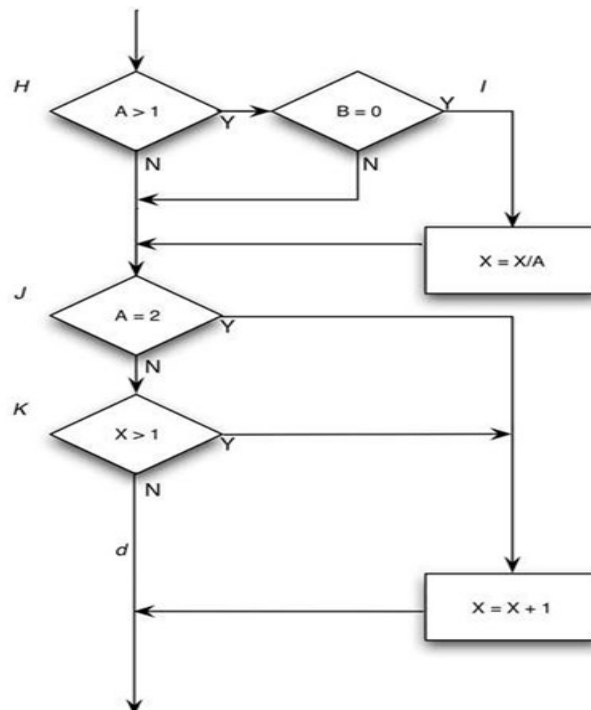
bao phủ tất cả các kết quả điều kiện, nhưng chúng chỉ bao phủ 2 trong 4 kết quả quyết định (cả 2 đều bao phủ đường đi  $abd$  và do đó, không sử dụng kết quả  $true$  của quyết định đầu tiên và kết quả  $false$  của quyết định thứ hai).

#### 2.3.1.4 Bao phủ quyết định/điều kiện – *Decision/condition coverage*

**Tư tưởng:** Thực hiện đủ các ca kiểm thử mà mỗi điều kiện trong 1 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.

**Hình 2.2** Mã máy cho chương trình trong Hình 2.1



Biểu đồ tiến trình trong hình 2.2 là cách 1 trình biên dịch tạo ra mã máy cho chương trình trong Hình 2.1. Các quyết định đa điều kiện trong chương trình nguồn

đã bị chia thành các quyết định và các nhánh riêng vì hầu hết các máy không được chế tạo để có thể thực hiện các quyết định đa điều kiện. Khi đó 1 bao phủ kiểm thử tỉ mỉ hơn xuất hiện là việc sử dụng tất cả các kết quả có thể của mỗi quyết định gốc. Hai ca kiểm thử bao phủ quyết định trước không làm được điều này; chúng không thể sử dụng kết quả *false* của quyết định H và kết quả *true* của quyết định K.

Lí do, như đã được chỉ ra trong hình 2.2, là những kết quả của các điều kiện trong các biểu thức *and* và *or* có thể cản trở hay ngăn chặn việc ước lượng các quyết định khác. Ví dụ, nếu 1 điều kiện *and* là sai, không cần kiểm tra các điều kiện tiếp theo trong biểu thức. Tương tự như vậy, nếu 1 điều kiện *or* là đúng thì cũng không cần kiểm tra các điều kiện còn lại. Do đó, các lỗi trong biểu thức logic không phải lúc nào cũng được phát hiện bằng các tiêu chuẩn bao phủ điều kiện và bao phủ quyết định/điều kiện.

#### 2.3.1.5 Bao phủ đa điều kiện – *Multiple condition coverage*

**Tư tưởng:** 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.

Ví dụ, xét chuỗi mã lệnh sau:

```
NOTFOUND = TRUE;  
DO I=1 TO TABSIZE WHILE (NOTFOUND); /*SEARCH TABLE*/  
    ...searching logic...;  
END;
```

Bốn tình huống để kiểm thử là:

1.  $I \leq \text{TABSIZE}$  và *NOTFOUND* có giá trị đúng (đang duyệt)
2.  $I \leq \text{TABSIZE}$  và *NOTFOUND* có giá trị sai (tìm thấy mục vào trước khi gặp cuối bảng).

3.  $I > TABSIZE$  và  $NOTFOUND$  có giá trị đúng (gấp cuối bảng mà không tìm thấy mục vào).
4.  $I > TABSIZE$  và  $NOTFOUND$  có giá trị sai (mục vào là cái cuối cùng trong bảng).

Dễ nhận thấy là tập hợp các ca kiểm thử thỏa mãn tiêu chuẩn đa điều kiện cũng thỏa mãn các tiêu chuẩn bao phủ quyết định, bao phủ điều kiện và bao phủ quyết định/điều kiện.

Quay lại hình 2.1, các ca kiểm thử phải bao phủ 8 sự kết hợp:

1.  $A > 1, B = 0$
2.  $A > 1, B < > 0$
3.  $A \leq 1, B = 0$
4.  $A \leq 1, B < > 0$
5.  $A = 2, X > 1$
6.  $A = 2, X \leq 1$
7.  $A < > 2, X > 1$
8.  $A < > 2, X \leq 1$

Vì là ca kiểm thử sớm hơn, nên cần chú ý là các trường hợp từ 5 đến 8 biểu diễn các giá trị tại vị trí câu lệnh *IF* thứ hai. Vì  $X$  có thể thay đổi ở trên câu lệnh *IF* này, nên giá trị cần tại câu lệnh *IF* này phải được sao dự phòng thông qua tính logic để tìm ra các giá trị đầu vào tương ứng.

Những sự kết hợp để được kiểm tra này không nhất thiết ngụ ý rằng cần thực hiện cả 8 ca kiểm thử. Trên thực tế, chúng có thể được bao phủ bởi 4 ca kiểm thử. Các giá trị đầu vào kiểm thử, và sự kết hợp mà chúng bao phủ, là như sau:

$A=2, B=0, X=4$	Bao phủ trường hợp 1, 5
$A=2, B=1, X=1$	Bao phủ trường hợp 2, 6
$A=1, B=0, X=2$	Bao phủ trường hợp 3, 7
$A=1, B=1, X=1$	Bao phủ trường hợp 4, 8

Thực tế là việc có 4 ca kiểm thử và 4 đường đi riêng biệt trong hình 2.1 chỉ là sự trùng hợp ngẫu nhiên. Trên thực tế, 4 ca kiểm thử này không bao phủ mọi đường đi, chúng bỏ qua đường đi *acd*. Ví dụ, bạn sẽ cần 8 ca kiểm thử cho quyết định sau mặc dù nó chỉ chứa 2 đường đi:

$$\begin{aligned} & \text{If } (x==y \&\& \text{length}(z)==0 \&\& \text{FLAG}) \{ \\ & \quad J=1; \} \\ & \text{Else } \{ \\ & \quad I=1; \} \end{aligned}$$

Trong trường hợp các vòng lặp, số lượng các ca kiểm thử được yêu cầu bởi tiêu chuẩn đa điều kiện thường ít hơn nhiều số lượng đường đi.

Tóm lại, đối với những chương trình chỉ chứa 1 điều kiện trên 1 quyết định, thì 1 tiêu chuẩn kiểm thử nhỏ nhất là một số lượng đủ các ca kiểm thử để (1) gọi tất cả các kết quả của mỗi quyết định ít nhất 1 lần và (2) gọi mỗi điểm của mục vào (như là điểm vào hay ON-unit) ít nhất 1 lần, để đảm bảo là tất cả các câu lệnh được thực hiện ít nhất 1 lần. Đối với những chương trình chứa các quyết định có đa điều kiện thì tiêu chuẩn tối thiểu là số lượng đủ các ca kiểm thử để gọi tất cả những sự kết hợp có thể của các kết quả điều kiện trong mỗi quyết định, và tất cả các điểm vào của chương trình ít nhất 1 lần.

## 2.3.2 Kiểm thử hộp đen

### 2.3.2.1 Phân lớp tương đương – *Equivalence Partitioning*

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 cách xác định tập con này là để nhận ra rằng 1 ca kiểm thử được lựa chọn tốt cũng nên có 2 đặc tính khác:

1. Giảm thiểu số lượng các ca kiểm thử khác mà phải được phát triển để hoàn thành mục tiêu đã định của kiểm thử “hợp lý”.
2. Bao phủ một tập rất lớn các ca kiểm thử có thể khác. Tức là, nó nói cho chúng ta một thứ gì đó về sự có mặt hay vắng mặt của những lỗi qua tập giá trị đầu vào cụ thể.

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

- (1). Xác định các lớp tương đương và
- (2). Xác định các ca kiểm thử.

### ***Xác định các lớp tương đương***

Các lớp tương đương được xác định 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 (có thể sử dụng bảng 2.3 để liệt kê các lớp tương đương).

***Hình 2.3 Một mẫu cho việc liệt kê các lớp tương đương***

Điều kiện bên ngoài	Các lớp tương đương hợp lệ	Các lớp tương đương không hợp lệ

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 1 đầu vào hay điều kiện bên ngoài đã cho, việc xác định các lớp tương đương hầu như là 1 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 1 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 1 trạng thái đầu vào xác định số giá trị, xác định 1 lớp tương đương hợp lệ và 2 lớp tương đương bất hợp lệ.
3. Nếu 1 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 1 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 1 lớp là như nhau, thì hãy chia lớp tương đương đó thành các lớp tương đương nhỏ hơn.

### ***Xác định các ca kiểm thử***

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 1 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 1 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 1 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 test – case 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.

#### **2.3.2.2      *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 1 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 1 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 1 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 1 kết quả bên ngoài giới hạn đầu ra, nhưng tuy nhiên rất đáng để xem xét tiềm ẩn đó.
4. Sử dụng nguyên tắc 2 cho mỗi trạng thái đầu ra.
5. Nếu đầu vào hay đầu ra của 1 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.
6. Sử dụng sự khéo léo của bạn để tìm các điều kiện biên.

### **2.3.2.3 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 bạn phân lớp tương đương các trạng thái đầu vào, thì số lượng sự kết hợp thường là rất lớn. Nếu bạn không có cách lựa chọn có hệ thống một tập con các trạng thái đầu vào, có lẽ bạn sẽ 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

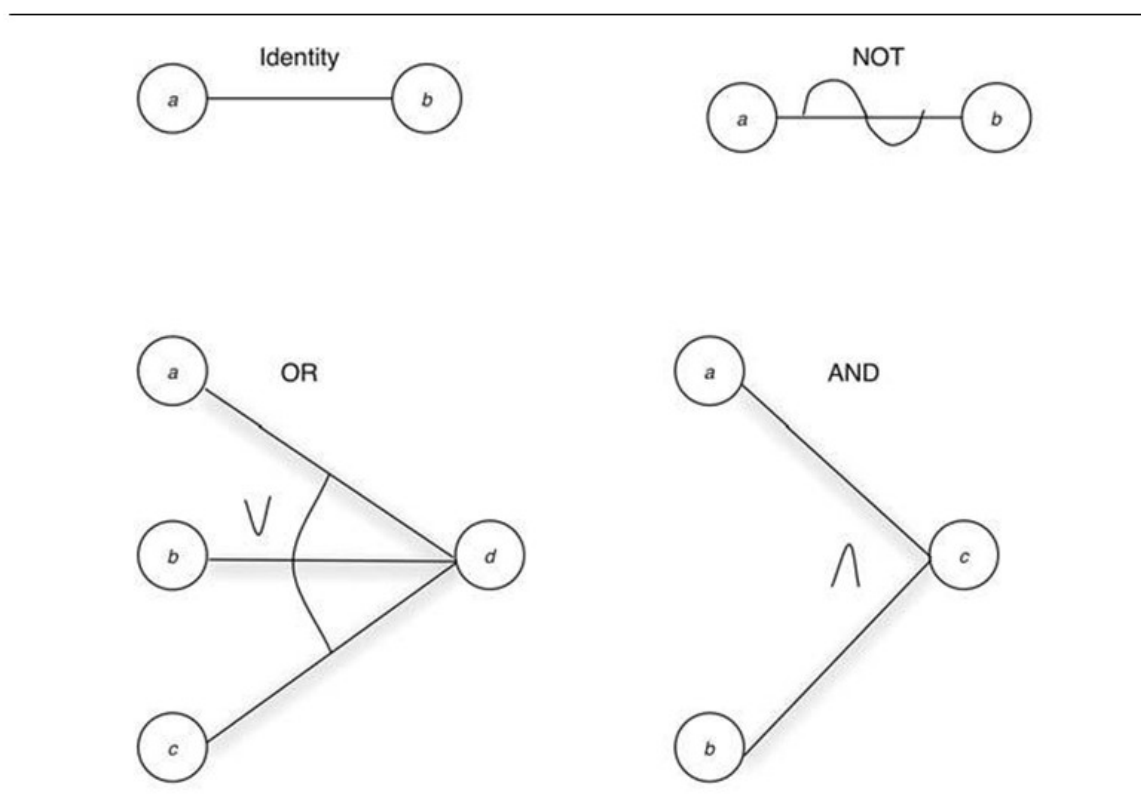
Quá trình dưới đây được sử dụng để xây dựng được các test – case:

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 1 sự biến đổi hệ thống (kết quả còn lại mà 1 đầu vào có trạng thái của 1 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 1 số 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.
6. Các cột trong bảng quyết định được chuyển thành các ca kiểm thử.

Ký hiệu cơ bản cho đồ thị được chỉ ra trong hình 2.4. Tưởng tượng mỗi nút có giá trị là 0 hoặc 1; 0 mô tả trạng thái vắng mặt và 1 mô tả trạng thái có mặt. Hàm

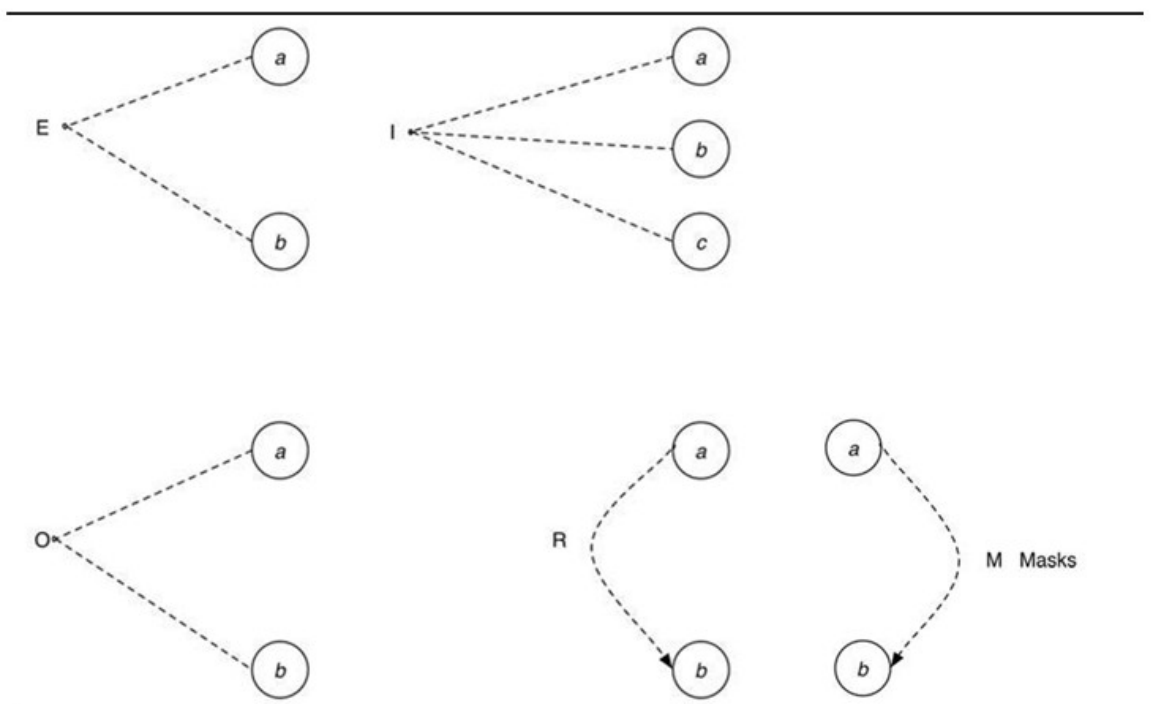
đồng nhất nói là nếu  $a$  là 1 thì  $b$  là 1; ngược lại,  $b$  là 0. Hàm *not* là nói nếu  $a$  là 1 thì  $b$  là 0; ngược lại thì  $b$  là 1. Hàm *or* khẳng định rằng nếu  $a$  hoặc  $b$  hoặc  $c$  là 1, thì  $d$  là 1; ngược lại  $d$  là 0. Hàm *and* khẳng định nếu cả  $a$  và  $b$  là 1 thì  $c$  là 1; ngược lại  $c$  là 0. Hai hàm *or* và *and* được phép có số lượng đầu vào bất kỳ.

**Hình 2.4** Các ký hiệu đồ thị nguyên nhân – kết quả cơ bản



Trong hầu hết các chương trình, sự kết hợp nào đó của một số nguyên nhân là không thể bởi vì lý do ngữ nghĩa và môi trường (ví dụ, một ký tự không thể đồng thời vừa là “A” vừa là “B”). Khi đó, ta sử dụng ký hiệu trong Hình 2.5. Ràng buộc E (Exclude – loại trừ) khẳng định rằng tối đa, chỉ có hoặc  $a$  hoặc  $b$  có thể là 1 ( $a$  và  $b$  không thể đồng thời là 1). Ràng buộc I (Include – bao hàm) khẳng định ít nhất một trong  $a$ ,  $b$  hoặc  $c$  phải luôn luôn là 1 ( $a$ ,  $b$  hoặc  $c$  không thể đồng thời là 0). Ràng buộc O (Only – chỉ một) khẳng định một và chỉ một hoặc  $a$  hoặc  $b$  phải là 1. Ràng buộc R (Request – yêu cầu) khẳng định rằng khi  $a$  là 1, thì  $b$  phải là 1 (ví dụ, không thể có trường hợp  $a$  là 1, còn  $b$  là 0). Ràng buộc M (Mask – mặt nạ) khẳng định là nếu kết quả  $a$  là 1, kết quả  $b$  sẽ bắt buộc phải là 0.

**Hình 2.5** Các ký hiệu ràng buộc



Bước tiếp theo là tạo bảng quyết định mục vào giới hạn – *limited-entry decision table*. Tương tự với các bảng quyết định, thì nguyên nhân chính là các điều kiện và kết quả chính là các hành động. Quy trình được sử dụng là như sau:

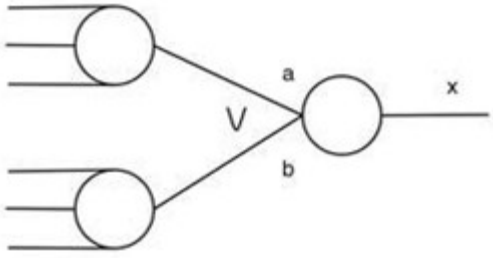
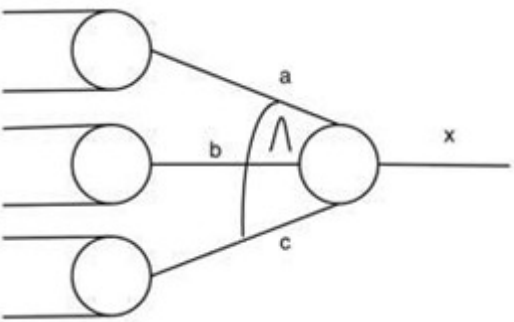
1. Chọn một kết quả để là trạng thái có mặt (1).
2. Lần ngược trở lại đồ thị, tìm tất cả những sự kết hợp của các nguyên nhân (đối tượng cho các ràng buộc) mà sẽ thiết lập kết quả này thành 1.
3. Tạo một cột trong bảng quyết định cho mỗi sự kết hợp nguyên nhân.
4. Với mỗi sự kết hợp, hãy quy định trạng thái của tất cả các kết quả khác và đặt chúng vào mỗi cột.

Trong khi biểu diễn bước 2, cần quan tâm các vấn đề sau:

1. Khi lần ngược trở lại qua một nút *or* mà đầu ra của nó là 1, không bao giờ thiết lập nhiều hơn 1 đầu vào cho nút *or* là 1 một cách đồng thời. Điều này được gọi là *path sensitizing* – *làm nhạy đường đi*. Mục tiêu của nó là để ngăn chặn dò lỗi thất bại vì một nguyên nhân che đi một nguyên nhân khác.

2. Khi lần ngược trở lại qua một nút *and* mà đầu ra của nó là 0, dĩ nhiên, phải liệt kê tất cả các sự kết hợp đầu vào dẫn tới đầu ra 0. Tuy nhiên, nếu bạn đang khảo sát trạng thái mà 1 đầu ra là 0 và một hay nhiều đầu ra khác là 1, thì không nhất thiết phải liệt kê tất cả các điều kiện mà dưới điều kiện đó các đầu vào khác có thể là 1.
3. Khi lần ngược trở lại qua một nút *and* mà đầu ra của nó là 0, chỉ cần liệt kê 1 điều kiện trong đó tất cả đầu vào bằng 0. (Nếu nút *and* ở chính giữa của đồ thị như vậy thì tất cả các đầu vào của nó xuất phát từ các nút trung gian khác, có thể có quá nhiều trạng thái mà trong trạng thái đó tất cả các đầu vào của nó bằng 0.)

**Hình 2.6** Những xem xét được sử dụng khi dò theo đồ thị

	<ul style="list-style-type: none"> <li>• Nếu <math>x=1</math>, không quan tâm về trường hợp <math>a=b=1</math> (sự xem xét thứ 1)</li> <li>• Nếu <math>x=0</math>, liệt kê tất cả các trường hợp trong đó <math>a=b=0</math>.</li> </ul>
	<ul style="list-style-type: none"> <li>• Nếu <math>x=1</math>, liệt kê tất cả các trường hợp trong đó <math>a=b=c=1</math>.</li> <li>• Nếu <math>x=0</math>, bao gồm chỉ 1 trường hợp mà <math>a=b=c=0</math> (sự xem xét 3). Đối với các trạng thái mà <math>abc</math> là 001, 010, 100, 011, 101 và 110, bao gồm chỉ 1 trường hợp mỗi trạng thái (sự xem xét 2).</li> </ul>

Những sự xem xét này có thể xuất hiện thất thường, nhưng chúng có một mục đích rất quan trọng: để giảm bớt các kết quả được kết hợp của đồ thị. Chúng liệt kê các trường hợp mà hướng về các ca kiểm thử ít có lợi. Nếu các ca kiểm thử ít có lợi không được liệt kê, một đồ thị nguyên nhân – kết quả lớn sẽ tạo ra một số lượng ca kiểm thử cực kỳ lớn. Nếu số lượng các ca kiểm thử trên thực tế là quá lớn, bạn sẽ



chọn ra 1 tập con nào đó, nhưng không đảm bảo là các ca kiểm thử ít có lợi sẽ là những ca kiểm thử được liệt kê. Do đó, tốt hơn hết là liệt kê chúng trong suốt quá trình phân tích của đồ thị.

## ***NHẬN XÉT***

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 1 sự lựa chọn kết hợp không thể dự tính trước, nhưng khi thực hiện như vậy, có vẻ như bạn sẽ bỏ sót nhiều ca kiểm thử “thứ 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 1 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, bạn 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 1 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 2 mục tiêu.

Chú ý là việc vẽ đồ thị nguyên nhân – kết quả phù hợp với một số quy tắc trong Chương 1. Việc xác định đầu ra mong đợi cho mỗi ca kiểm thử là một phần

cổ 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 1 chương trình. Trên thị trường đã có một vài chương trình thương mại tồn tại giúp cho quá trình chuyển đổi này.

#### **2.3.2.4      *Đoán lỗi – Error Guessing***

Một kỹ thuật thiết kế test-case khác là *error guessing* – *đoán lỗi*. Tester được đưa cho 1 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, bạn liệt kê những trường hợp đặc biệt đó mà có thể đã bị bỏ sót khi chương trình được thiết kế.

#### **2.3.3      Chiến lược**

Các phương pháp thiết kế test-case đã được thảo luận có thể được kết hợp thành một chiến lược toàn diện. Vì mỗi phương pháp có thể đóng góp 1 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. Chiến lược hợp lý 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.

Tuy việc sử dụng chiến lược này sẽ không đảm bảo rằng tất cả các lỗi sẽ được tìm thấy, nhưng nó đã được xác minh là đại diện cho một sự thỏa thuận hợp lý.

## CHƯƠNG 3.     ÁP DỤNG

Từ những phương pháp thiết kế test – case đã tìm hiểu ở trên, em vận dụng chúng vào thiết kế test – case cho chương trình “*Tam giác*”.

### 3.1     Đặc tả

Chương trình đọc vào 3 giá trị nguyên từ hộp thoại vào. Ba giá trị này tương ứng với chiều dài 3 cạnh của 1 tam giác. Chương trình hiển thị 1 thông điệp cho biết tam giác đó là tam giác thường, cân, hay đều.

Ba giá trị nhập vào thỏa mãn là 3 cạnh của một tam giác khi và chỉ khi cả 3 số đều là số nguyên dương, và tổng của 2 số bất kỳ trong 3 số phải lớn hơn số thứ 3. Khi đó, một tam giác đều là tam giác có 3 cạnh bằng nhau, tam giác cân là tam giác có 2 trong 3 cạnh bằng nhau, và tam giác thường thì có 3 cạnh khác nhau.

#### ***Mã lệnh của chương trình:***

```
unit main;
interface
uses
    Windows, Messages, SysUtils, Variants, Classes, Graphics,
    Controls, Forms, Dialogs, StdCtrls, ExtCtrls;
type
    TMainForm = class(TForm)
        AEdit: TLabelEdit;
        BEdit: TLabelEdit;
        CEdit: TLabelEdit;
        btnTest: TButton;
        procedure btnTestClick(Sender: TObject);
    private
        { Private declarations }
    public
```

```

        { Public declarations }
    end;
var
    MainForm: TMainForm;
implementation
    {$R *.dfm}
    Procedure TMainForm.btnTestClick(Sender: TObject);
    var
        a, b, c: Integer;
    begin
        try
            a := StrToInt(AEdit.Text);
            b := StrToInt(BEdit.Text);
            c := StrToInt(CEdit.Text);
            if (a < 0) Or (b < 0) Or (c < 0) then
                ShowMessage('3 canh A, B, C khong thoa man la 3
                canh cua mot tam giac.')
            else
                if (a + b > c) And (a + c > b) And (b + c > a) then
                    begin
                        if (a = b) And (b = c) then
                            ShowMessage('3 canh A, B, C lap thanh
                            mot tam giac deu.')
                        else
                            if (a=b) Or (b=c) Or (c=b) then
                                ShowMessage('3 canh A, B, C lap
                                thanh mot tam giac can.')
                            else
                                ShowMessage('3 canh A, B, C lap
                                thanh mot tam giac thuong.');
```

```
        ShowMessage('Loi dinh dang du lieu. De nghi ban xem  
        va nhap lai.');
```

end;  
end;  
end.

## 3.2 Thiết kế test – case

Áp dụng chiến lược kiểm thử đã trình bày trong Chương 2, các ca kiểm thử được xây dựng như sau:

### 3.2.1 Vẽ đồ thị nguyên nhân – kết quả

Do đặc tả có sự kết hợp đầu vào nên trước tiên, áp dụng phương pháp vẽ đồ thị nguyên nhân – kết quả.

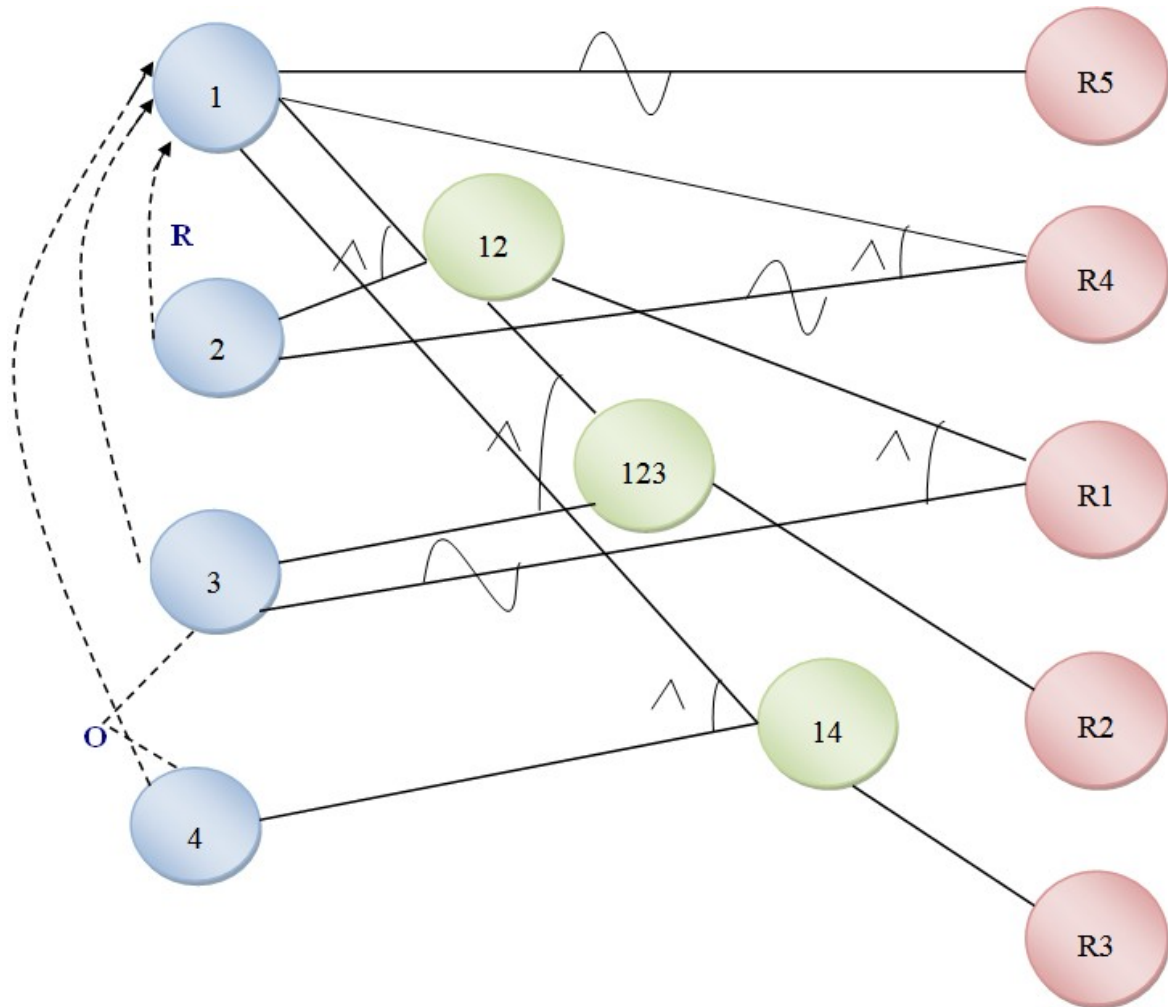
Nguyên nhân là:

1. Cả 3 giá trị nhập vào đều là số nguyên dương.
2. Tổng 2 số bất kỳ trong 3 số lớn hơn số còn lại.
3. Hai trong 3 số có giá trị bằng nhau.
4. Ba số có giá trị bằng nhau.

Kết quả là:

- R1. Thông báo ba giá trị nhập vào lập thành tam giác thường.
- R2. Thông báo ba giá trị nhập vào lập thành tam giác cân.
- R3. Thông báo ba giá trị nhập vào lập thành tam giác đều.
- R4. Thông báo ba giá trị nhập vào không lập thành một tam giác.
- R5. Thông báo lỗi nhập dữ liệu.

**Hình 3.1 Đồ thị nguyên nhân – kết quả:**



Bước tiếp theo là tạo bảng quyết định mục vào giới hạn. Chọn kết quả R1 là đầu tiên. R1 có mặt nếu nút các nút 12 và 3 = 1,0. Nút 12 = 1 khi 1 và 2 = 1,1.

Áp dụng lần lượt cho sự có mặt của từng kết quả đầu vào, ta được bảng quyết định như sau:

**Hình 3.2 Bảng quyết định**

	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>5</i>
<i>1</i>	1	1	1	1	0

<b>2</b>	1	1		0	
<b>3</b>	0	1			
<b>4</b>			1		
<b>R1</b>	1	0	0	0	0
<b>R2</b>	0	1	0	0	0
<b>R3</b>	0	0	1	0	0
<b>R4</b>	0	0	0	1	0
<b>R5</b>	0	0	0	0	1

Bước cuối cùng là chuyển đổi bảng quyết định thành các ca kiểm thử. Các ca kiểm thử thu được như sau:

STT	Các điều kiện	Ca kiểm thử	Hành động
1	Cả 3 giá trị nhập vào đều là số nguyên dương, và tổng của 2 số bất kỳ trong 3 số luôn lớn hơn số thứ 3, và không có cặp 2 số bất kỳ nào trong 3 số đó là = nhau.	2,3,4 2,4,3 3,2,4 3,4,2 4,2,3 4,3,2	R1
2	Cả 3 giá trị nhập vào đều là số nguyên dương, và tổng của 2 số	3,3,4	R2



	bất kỳ trong 3 số luôn lớn hơn số thứ 3, và tồn tại một cặp 2 số trong 3 số đó là = nhau.	3,4,3 4,3,3	
3	Cả 3 giá trị nhập vào đều là số nguyên dương, và cả 3 số có giá trị bằng nhau.	3,3,3	R3
4	Cả 3 giá trị nhập vào đều là số nguyên dương, và tồn tại 2 số trong 3 số có tổng nhỏ hơn hoặc bằng số còn lại.	1,2,4 Và 5 hoán vị của nó	R4
5	Tồn tại một giá trị nhập vào không phải là số nguyên dương.	A,2,2 -1,1,1 1.1,1,1 Và 2 hoán vị của mỗi trường hợp	R5

### 3.2.2 Phân lớp tương đương

#### 3.2.2.1 Xác định các lớp tương đương

Các giá trị nhập vào là số	Cả 3 giá trị đều là số (1)	Tồn tại 1 giá trị không phải là số (2)
Các giá trị là nguyên	Cả 3 giá trị đều nguyên (3)	Tồn tại 1 giá trị không nguyên (4)
Các giá trị là dương	Cả 3 giá trị đều dương (5)	Tồn tại 1 giá trị $\leq 0$ (6)

Hàng số	-32768 : 32767 (7)	<-32768 (8), >32767 (9)
Tổng 2 số bất kỳ so với số thứ 3	Lớn hơn (10)	Nhỏ hơn hoặc bằng (11)

### 3.2.2.2 *Xác định các ca kiểm thử*

#### *Các ca kiểm thử bao phủ các lớp tương đương hợp lệ:*

Ca kiểm thử 2,3,4 và 5 hoán vị bao phủ các lớp (1), (3), (5), (7), (10).

#### *Các ca kiểm thử tương ứng với từng ca kiểm thử không hợp lệ:*

- (2) A, 1, 1 và 2 hoán vị.
- (4) 1.1, 1, 1 và 2 hoán vị.
- (6) -1, 1, 1 và 2 hoán vị.
- (8) -32769, 1, 1 và 2 hoán vị.
- (9) 32768, 1, 1 và 2 hoán vị.
- (11) 1, 2, 4 và 5 hoán vị.

### 3.2.3 *Phân tích giá trị biên*

#### 3.2.3.1 *Xét các trạng thái đầu vào*

Xét các trạng thái đầu vào thu được các ca kiểm thử như sau:

1. 1, 1, 1
2. A, 1, 1 và 2 hoán vị.
3. 1.1, 1, 1 và 2 hoán vị.
4. 0, 1, 1 và 2 hoán vị.

5. -1, 1, 1 và 2 hoán vị.
6. -32768, 1, 1 và 2 hoán vị.
7. -32769, 1, 1 và 2 hoán vị.
8. 32767, 1, 1 và 2 hoán vị.
9. 32768, 1, 1 và 2 hoán vị.
10. 1, 2, 3 và 5 hoán vị.
11. 1, 2, 4 và 5 hoán vị.

### 3.2.3.2 *Xét không gian kết quả*

Xét không gian kết quả, thu được các ca kiểm thử như sau:

Ba số đầu vào thỏa mãn là 3 cạnh của một tam giác đều

12. 1, 1, 1
13. 32767, 32767, 32767

Ba số đầu vào thỏa mãn là 3 cạnh của một tam giác cân

14. 1, 1, 2 và 2 hoán vị.
15. 32767, 32767, 32766 và 2 hoán vị.

Ba số đầu vào thỏa mãn là 3 cạnh của một tam giác thường

16. 2, 3, 4 và 5 hoán vị.
17. 32767, 32766, 32765 và 5 hoán vị.

Ba số đầu vào không thỏa mãn là 3 cạnh của 1 tam giác

18. 1, 2, 3 và 5 hoán vị.
19. 1, 2, 4 và 5 hoán vị.
20. 32767, 1, 32765 và 5 hoán vị.

21. 32767, 1, 32766 và 5 hoán vị.

Lỗi định dạng dữ liệu vào

22. A, 1, 1 và 2 hoán vị.

23. 0, 1, 1 và 2 hoán vị.

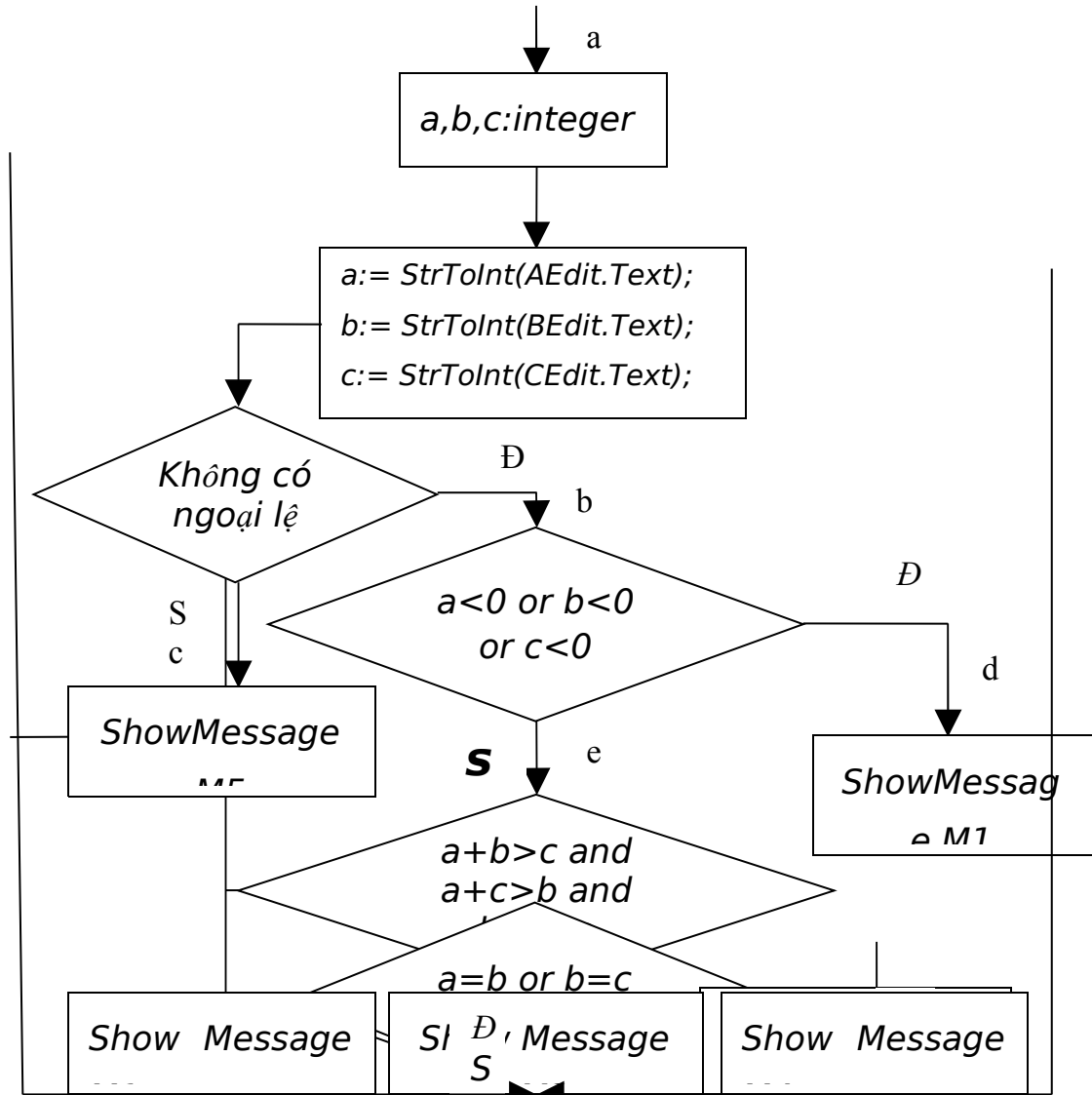
24. -1, 1, 1 và 2 hoán vị.

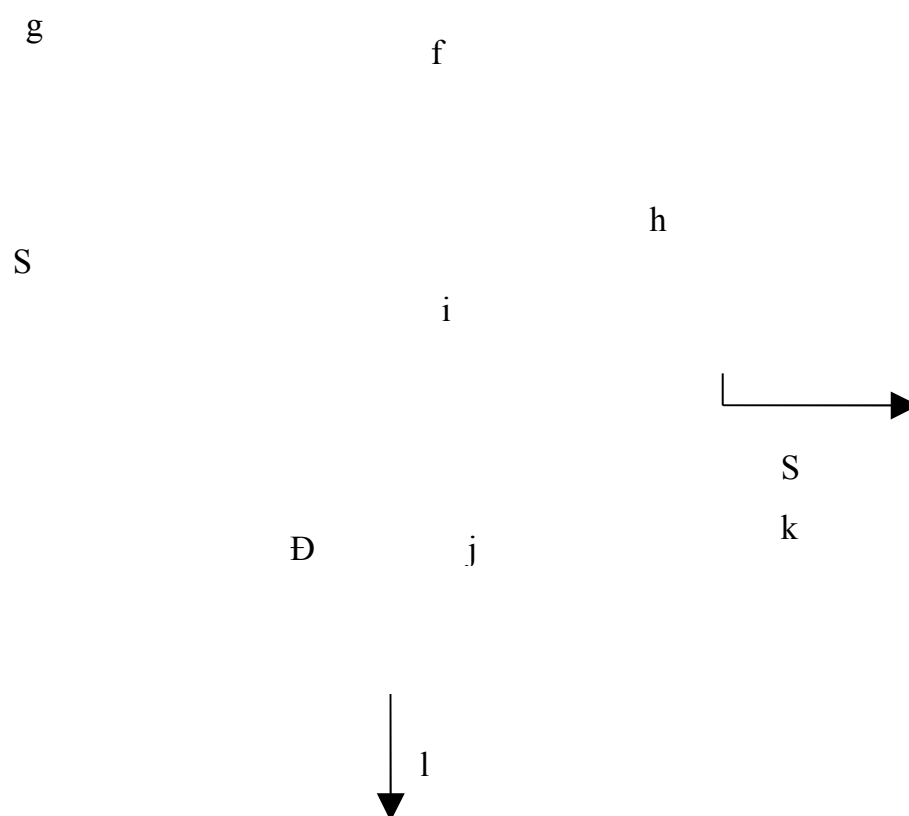
25. 1.1, 1, 1 và 2 hoán vị.

### 3.2.4 Các phương pháp hộp trắng

#### 3.2.4.1 Bao phủ câu lệnh

Lưu đồ thuật toán cho chương trình tam giác:





Trong đó:

M1: *Ba cạnh A, B, C không thỏa mãn là 3 cạnh của 1 tam giác.*

M2: *Ba cạnh A, B, C lập thành 1 tam giác đều.*

M3: *Ba cạnh A, B, C lập thành 1 tam giác cân.*

M4: *Ba cạnh A, B, C lập thành 1 tam giác thường.*

M5: *Lỗi định danh dữ liệu. Đề nghị bạn xem và nhập lại.*

Các ca kiểm thử thu được:

1. -1, 1, 1 và các hoán vị (abdl)
2. 1, 1, 1 (abefhl)
3. 2, 2, 1 và các hoán vị (abefjil)
4. 2, 3, 4 và các hoán vị (abefikl)
5. 1, 2, 4 và các hoán vị (abegl)
6. A, 1, 1 và các hoán vị (acl)

7. 1.1, 1, 1 và các hoán vị (a,c,l)

### 3.2.4.2 *Bao phủ quyết định*

Các ca kiểm thử thu được:

- |    |                        |                |           |
|----|------------------------|----------------|-----------|
| 1. | -32768, -32768, -32768 | và các hoán vị | (abdl)    |
| 2. | 32767, 32767, 32767    | và các hoán vị | (abefhl)  |
| 3. | 32767, 32767, 327676   | và các hoán vị | (abefijl) |
| 4. | 32767, 32766, 32765    | và các hoán vị | (abefikl) |
| 5. | 32767, 1, 2            | và các hoán vị | (abeghl)  |
| 6. | A, 1, 1                | và các hoán vị | (acl)     |
| 7. | 1.1, 1, 1              | và các hoán vị | (a,c,l)   |

### 3.2.4.3 *Bao phủ điều kiện*

Các ca kiểm thử thu được là:

1. -1, 1, 1 và các hoán vị.
2. 2, 3, 4 và các hoán vị.
3. 1, 2, 4 và các hoán vị.
4. 2, 2, 1 và các hoán vị.
5. 1, 1, 1 và các hoán vị.
6. A, 1, 1 và các hoán vị.
7. 1.1, 1, 1 và các hoán vị.

### 3.2.4.4 *Bao phủ quyết định – điều kiện*

Các ca kiểm thử thu được là:

1. -1, 1, 1 và các hoán vị.
2. 2, 3, 4 và các hoán vị.

3. 1, 2, 4 và các hoán vị.
4. 2, 2, 1 và các hoán vị.
5. 1, 1, 1 và các hoán vị.
6. A, 1, 1 và các hoán vị.
7. 1.1, 1, 1 và các hoán vị.

#### **3.2.4.5 Bao phủ đa điều kiện**

Các ca kiểm thử thu được là:

1. -1, 1, 1 và các hoán vị.
2. 2, 3, 4 và các hoán vị.
3. 1, 2, 4 và các hoán vị.
4. 2, 2, 1 và các hoán vị.
5. 1, 1, 1 và các hoán vị.
6. A, 1, 1 và các hoán vị.
7. 1.1, 1, 1 và các hoán vị.

# KẾT LUẬN

Kiểm thử phần mềm, một hướng đi không còn mới mẻ trên thế giới, nhưng lại là một hướng đi rất mới ở Việt Nam. Nó hứa hẹn một tương lai mới cho các học sinh, sinh viên ngành CNTT.

Qua tìm hiểu và xây dựng đề tài này, em thấy mình đã đạt được một ưu điểm cũng như vẫn còn một số tồn tại.

Những điểm đạt được:

- ✓ Nắm được tổng quan về kiểm thử phần mềm: Các khái niệm cơ bản, các phương pháp kiểm thử phần mềm, và các vấn đề liên quan ...
- ✓ Tìm hiểu và nắm được các phương pháp và chiến lược thiết kế test – case trong kiểm thử phần mềm, và áp dụng được các phương pháp đã tìm hiểu để xây dựng các test – case cho 1 bài toán cụ thể - Chương trình “*Tam giác*”.
- ✓ Bổ sung và rèn luyện thêm kỹ năng sử dụng phần mềm Word và Powerpoint.
- ✓ Nâng cao khả năng đọc hiểu tài liệu Tiếng Anh.

Những điểm chưa đạt:

- ✓ Sự áp dụng những kiến thức tìm hiểu được mới chỉ dừng lại ở một bài toán nhỏ, mà vẫn chưa thử áp dụng cho các bài toán hay ứng dụng lớn.

Từ những điểm đạt và chưa đạt ở trên, em hi vọng sẽ nhận được sự góp ý chân thành của các thầy cô và các bạn để bản báo cáo được hoàn thiện hơn.

Sinh viên

***Phạm Thị Trang***



# TÀI LIỆU THAM KHẢO

1. *The Art of Software Testing*, Glenford J. Myers, Second Edition, John Wiley and Sons, Inc.
2. *Software Engineering - A Practitioner's Approach*, Roger S. Pressman, Sixth Edition, Ph.D, McGraw-Hill, Inc.
3. *A Practitioner's Guide to Software Test Design*, Lee Copeland, First Edition, Artech House Publishers Boston, London.
4. *Effective methods for Software Testing*, William E. Perry, 3rd Edition, Wiley Publishing, Indian.
5. *Software Testing*, Ron Patton, Second Edition, Sam Publishing.
6. <http://www.vietnamesetestingboard.org/>
7. [http://en.wikipedia.org/wiki/Software\\_testing](http://en.wikipedia.org/wiki/Software_testing)
8. Một số trang web về kiểm thử phần mềm khác.

## This image shows a full page of white paper with horizontal dotted lines, typical of primary school handwriting practice paper. The lines are evenly spaced and run across the entire width of the page. There are no margins, text, or other markings present.

Giáo viên hướng dẫn

58