**Bài tập tổng hợp về Kỹ thuật Black-Box và White-Box Nâng cao**

**Phần 1: Lý thuyết**

**1. Bảng quyết định trong Black-Box Testing**

**a. Bảng quyết định là gì? Giải thích cách sử dụng bảng quyết định để kiểm thử các điều kiện logic phức tạp.**

Bảng quyết định (Decision Table) là một kỹ thuật kiểm thử Black-Box được sử dụng để mô tả các tổ hợp điều kiện logic phức tạp và kết quả tương ứng của chúng. Nó giúp kiểm thử viên xác định tất cả các trường hợp có thể xảy ra dựa trên các điều kiện đầu vào và đảm bảo rằng mọi tổ hợp đều được kiểm tra.

**Cách sử dụng bảng quyết định:**

1. **Xác định điều kiện và hành động**: Liệt kê tất cả các điều kiện logic (ví dụ: độ tuổi, thời gian) và các hành động hoặc kết quả tương ứng (ví dụ: giá vé).
2. **Liệt kê tổ hợp**: Tạo bảng với các cột đại diện cho các điều kiện và các hàng biểu thị các tổ hợp giá trị (thường là True/False hoặc các giá trị cụ thể).
3. **Xác định kết quả**: Ghi lại kết quả mong đợi cho mỗi tổ hợp điều kiện.
4. **Thiết kế test case**: Dựa trên bảng, tạo các test case để kiểm tra từng tổ hợp điều kiện và xác minh kết quả.

Bảng quyết định đặc biệt hữu ích khi hệ thống có nhiều điều kiện logic kết hợp, giúp giảm thiểu việc bỏ sót các trường hợp kiểm thử.

**b. Ví dụ về hệ thống có điều kiện logic phức tạp**

**Hệ thống**: Tính giá vé xem phim dựa trên độ tuổi và thời gian đặt vé.

* **Điều kiện**:
  + Độ tuổi: Trẻ em (<12 tuổi), Người lớn (12-60 tuổi), Người cao tuổi (>60 tuổi).
  + Thời gian: Giờ thường (trước 17:00), Giờ cao điểm (sau 17:00).
* **Kết quả**: Giá vé thay đổi theo tổ hợp (ví dụ: Trẻ em giờ thường: $5, Người lớn giờ cao điểm: $12).

**Cách tạo bảng quyết định**:

1. Liệt kê các điều kiện:
   * Độ tuổi: Trẻ em, Người lớn, Người cao tuổi.
   * Thời gian: Giờ thường, Giờ cao điểm.
2. Tạo bảng với tất cả các tổ hợp (3 độ tuổi x 2 thời gian = 6 trường hợp).
3. Ghi lại giá vé tương ứng cho mỗi tổ hợp.

Ví dụ bảng quyết định:

| **Độ tuổi** | **Thời gian** | **Giá vé** |
| --- | --- | --- |
| Trẻ em | Giờ thường | $5 |
| Trẻ em | Giờ cao điểm | $7 |
| Người lớn | Giờ thường | $10 |
| Người lớn | Giờ cao điểm | $12 |
| Người cao tuổi | Giờ thường | $7 |
| Người cao tuổi | Giờ cao điểm | $9 |

**2. Sơ đồ chuyển trạng thái**

**a. Sơ đồ chuyển trạng thái là gì? Giải thích cách sử dụng.**

Sơ đồ chuyển trạng thái (State Transition Diagram) là một kỹ thuật kiểm thử Black-Box mô tả các trạng thái của hệ thống và các chuyển đổi giữa chúng khi có sự kiện xảy ra. Nó được sử dụng để kiểm tra hành vi của hệ thống qua các trạng thái khác nhau, đảm bảo rằng các chuyển đổi trạng thái hoạt động đúng như mong đợi.

**Cách sử dụng**:

1. **Xác định trạng thái**: Liệt kê tất cả các trạng thái mà hệ thống có thể có (ví dụ: Giỏ hàng trống, Đang thanh toán).
2. **Xác định sự kiện**: Xác định các sự kiện gây ra chuyển đổi giữa các trạng thái (ví dụ: Thêm sản phẩm, Xác nhận đơn hàng).
3. **Vẽ sơ đồ**: Vẽ sơ đồ với các trạng thái (hình tròn) và các chuyển đổi (mũi tên) kèm nhãn sự kiện.
4. **Thiết kế test case**: Tạo test case để kiểm tra từng chuyển đổi trạng thái, bao gồm trạng thái ban đầu, sự kiện, và trạng thái kết quả.

**b. Ví dụ về hệ thống cần kiểm thử chuyển trạng thái**

**Hệ thống**: Quy trình đặt hàng trực tuyến.

* **Trạng thái chính**:
  + Giỏ hàng trống: Không có sản phẩm trong giỏ.
  + Đã thêm sản phẩm: Có ít nhất một sản phẩm trong giỏ.
  + Đang thanh toán: Người dùng đang nhập thông tin thanh toán.
  + Đã xác nhận: Đơn hàng được xác nhận thành công.
  + Đã hủy: Đơn hàng bị hủy.

**Mô tả sơ đồ chuyển trạng thái**:

* Từ **Giỏ hàng trống** → **Đã thêm sản phẩm** (sự kiện: Thêm sản phẩm).
* Từ **Đã thêm sản phẩm** → **Đang thanh toán** (sự kiện: Tiến hành thanh toán).
* Từ **Đang thanh toán** → **Đã xác nhận** (sự kiện: Xác nhận thanh toán).
* Từ **Đang thanh toán** → **Đã hủy** (sự kiện: Hủy đơn hàng).
* Từ **Đã thêm sản phẩm** → **Giỏ hàng trống** (sự kiện: Xóa hết sản phẩm).

**3. Statement và Branch Coverage trong White-Box Testing**

**a. Định nghĩa và sự khác biệt**

* **Statement Coverage**: Đo lường tỷ lệ phần trăm các dòng mã (statement) được thực thi trong quá trình kiểm thử. Mục tiêu là đảm bảo mỗi dòng mã được chạy ít nhất một lần.
* **Branch Coverage**: Đo lường tỷ lệ phần trăm các nhánh (branch) trong các cấu trúc điều kiện (if, else, elif) được thực thi. Mục tiêu là kiểm tra tất cả các nhánh của mỗi cấu trúc điều kiện.

**Sự khác biệt**:

* Statement coverage chỉ đảm bảo rằng mỗi dòng mã được thực thi, nhưng không đảm bảo rằng tất cả các trường hợp logic trong các nhánh được kiểm tra.
* Branch coverage yêu cầu kiểm tra cả hai nhánh (True và False) của mỗi điều kiện, do đó bao quát hơn về mặt logic.

**b. Tại sao 100% branch coverage đảm bảo chất lượng tốt hơn?**

* Branch coverage đảm bảo rằng tất cả các nhánh logic trong mã được kiểm tra, bao gồm cả các trường hợp hiếm hoặc ngoại lệ, giúp phát hiện lỗi trong các điều kiện phức tạp.
* Statement coverage có thể đạt 100% mà không kiểm tra tất cả các nhánh (ví dụ: chỉ kiểm tra nhánh True của một điều kiện if). Điều này có thể bỏ sót các lỗi trong nhánh không được thực thi.
* Ví dụ: Trong đoạn mã if x > 0: y = 1 else: y = 0, statement coverage chỉ cần kiểm tra trường hợp x > 0 để thực thi dòng y = 1, nhưng branch coverage yêu cầu kiểm tra cả x > 0 và x <= 0.

**4. Phân tích mã để xác định điểm kiểm thử**

**a. Quy trình phân tích mã**

1. **Đọc mã nguồn**: Hiểu cấu trúc và logic của mã, xác định các câu lệnh điều kiện, vòng lặp, và các nhánh.
2. **Xác định điểm kiểm thử**:
   * **Statement coverage**: Đảm bảo mỗi dòng mã được thực thi ít nhất một lần.
   * **Branch coverage**: Xác định tất cả các nhánh trong các cấu trúc điều kiện (if, else, elif) và thiết kế test case để kiểm tra từng nhánh.
3. **Xác định giá trị đầu vào**: Chọn các giá trị đầu vào để kích hoạt từng nhánh hoặc dòng mã.
4. **Kiểm tra biên và trường hợp ngoại lệ**: Xác định các giá trị biên (boundary values) và các trường hợp ngoại lệ (ví dụ: giá trị âm, null).
5. **Thiết kế test case**: Tạo các test case dựa trên các điểm kiểm thử đã xác định.

**b. Ví dụ về lỗi tiềm ẩn**

**Đoạn mã**:

def divide(a, b):

return a / b

**Lỗi tiềm ẩn**: Nếu b = 0, mã sẽ gây ra lỗi chia cho 0 (ZeroDivisionError).  
**Phát hiện qua White-Box**: Phân tích mã cho thấy nhánh b == 0 không được xử lý. Test case với b = 0 sẽ phát hiện lỗi này.  
**Sửa lỗi**:

def divide(a, b):

if b == 0:

raise ValueError("Cannot divide by zero")

return a / b

**Phần 2: Bài tập thực hành**

**1. Tạo bảng quyết định**

**a. Bảng quyết định cho tính năng tính giá vé**

**Điều kiện**:

* Độ tuổi: Trẻ em (<12), Người lớn (12-60), Người cao tuổi (>60).
* Thời gian: Giờ thường (trước 17:00), Giờ cao điểm (sau 17:00).  
  **Kết quả**: Giá vé thay đổi theo tổ hợp.

**Bảng quyết định**:

| **Trường hợp** | **Độ tuổi** | **Thời gian** | **Giá vé** |
| --- | --- | --- | --- |
| 1 | Trẻ em | Giờ thường | $5 |
| 2 | Trẻ em | Giờ cao điểm | $7 |
| 3 | Người lớn | Giờ thường | $10 |
| 4 | Người lớn | Giờ cao điểm | $12 |
| 5 | Người cao tuổi | Giờ thường | $7 |
| 6 | Người cao tuổi | Giờ cao điểm | $9 |

**b. Thiết kế 4 test case**

1. **Test Case 1**:
   * **Mô tả**: Kiểm tra giá vé cho trẻ em vào giờ thường.
   * **Dữ liệu đầu vào**: Độ tuổi = 10, Thời gian = 14:00.
   * **Kết quả mong đợi**: Giá vé = $5.
2. **Test Case 2**:
   * **Mô tả**: Kiểm tra giá vé cho người lớn vào giờ cao điểm.
   * **Dữ liệu đầu vào**: Độ tuổi = 30, Thời gian = 18:00.
   * **Kết quả mong đợi**: Giá vé = $12.
3. **Test Case 3**:
   * **Mô tả**: Kiểm tra giá vé cho người cao tuổi vào giờ thường.
   * **Dữ liệu đầu vào**: Độ tuổi = 65, Thời gian = 15:00.
   * **Kết quả mong đợi**: Giá vé = $7.
4. **Test Case 4**:
   * **Mô tả**: Kiểm tra giá vé cho trẻ em vào giờ cao điểm.
   * **Dữ liệu đầu vào**: Độ tuổi = 8, Thời gian = 19:00.
   * **Kết quả mong đợi**: Giá vé = $7.

**2. Thiết kế sơ đồ chuyển trạng thái**

**a. Sơ đồ chuyển trạng thái**

**Hệ thống**: Ứng dụng đặt hàng trực tuyến.  
**Trạng thái**:

* Giỏ hàng trống
* Đã thêm sản phẩm
* Đang thanh toán
* Đã xác nhận
* Đã hủy

**Mô tả sơ đồ chuyển trạng thái (bằng văn bản)**:

* **Giỏ hàng trống** → **Đã thêm sản phẩm** (Sự kiện: Thêm sản phẩm).
* **Đã thêm sản phẩm** → **Giỏ hàng trống** (Sự kiện: Xóa hết sản phẩm).
* **Đã thêm sản phẩm** → **Đang thanh toán** (Sự kiện: Tiến hành thanh toán).
* **Đang thanh toán** → **Đã xác nhận** (Sự kiện: Xác nhận thanh toán).
* **Đang thanh toán** → **Đã hủy** (Sự kiện: Hủy đơn hàng).
* **Đã xác nhận** → **Đã hủy** (Sự kiện: Hủy đơn hàng sau xác nhận).

**b. Thiết kế 3 test case**

1. **Test Case 1**:
   * **Mô tả**: Kiểm tra chuyển từ Giỏ hàng trống sang Đã thêm sản phẩm.
   * **Hành động**: Thêm một sản phẩm vào giỏ hàng.
   * **Kết quả mong đợi**: Trạng thái hệ thống chuyển sang Đã thêm sản phẩm.
2. **Test Case 2**:
   * **Mô tả**: Kiểm tra chuyển từ Đã thêm sản phẩm sang Đang thanh toán và sau đó sang Đã xác nhận.
   * **Hành động**: Thêm sản phẩm, nhấn "Tiến hành thanh toán", nhập thông tin thanh toán hợp lệ, nhấn "Xác nhận".
   * **Kết quả mong đợi**: Trạng thái hệ thống chuyển sang Đã xác nhận.
3. **Test Case 3**:
   * **Mô tả**: Kiểm tra chuyển từ Đang thanh toán sang Đã hủy.
   * **Hành động**: Thêm sản phẩm, nhấn "Tiến hành thanh toán", nhấn "Hủy đơn hàng".
   * **Kết quả mong đợi**: Trạng thái hệ thống chuyển sang Đã hủy.

**3. Phân tích mã và xác định điểm kiểm thử**

**a. Phân tích đoạn mã**

**Đoạn mã**:

def calculate\_ticket\_price(age):

if age < 12:

price = 5

elif age <= 60:

price = 10

else:

price = 7

return price

**Phân tích để đạt 100% statement và branch coverage**:

* **Statement coverage**: Cần thực thi tất cả các dòng mã:
  + Dòng if age < 12: Kiểm tra với tuổi < 12.
  + Dòng elif age <= 60: Kiểm tra với 12 ≤ tuổi ≤ 60.
  + Dòng else: Kiểm tra với tuổi > 60.
  + Dòng return price: Được thực thi trong mọi trường hợp.
* **Branch coverage**: Cần kiểm tra tất cả các nhánh của các điều kiện:
  + Nhánh age < 12 (True).
  + Nhánh age < 12 (False) dẫn đến kiểm tra age <= 60.
  + Nhánh age <= 60 (True).
  + Nhánh age <= 60 (False) dẫn đến else.

**Điểm kiểm thử**:

* Tuổi < 12 (ví dụ: 10).
* 12 ≤ Tuổi ≤ 60 (ví dụ: 30).
* Tuổi > 60 (ví dụ: 65).
* Giá trị biên: Tuổi = 12, 60 (điểm chuyển tiếp giữa các nhánh).

**b. Thiết kế 4 test case**

1. **Test Case 1**:
   * **Mô tả**: Kiểm tra giá vé cho trẻ em.
   * **Dữ liệu đầu vào**: Tuổi = 10.
   * **Kết quả mong đợi**: Giá vé = $5.
2. **Test Case 2**:
   * **Mô tả**: Kiểm tra giá vé cho người lớn.
   * **Dữ liệu đầu vào**: Tuổi = 30.
   * **Kết quả mong đợi**: Giá vé = $10.
3. **Test Case 3**:
   * **Mô tả**: Kiểm tra giá vé cho người cao tuổi.
   * **Dữ liệu đầu vào**: Tuổi = 65.
   * **Kết quả mong đợi**: Giá vé = $7.
4. **Test Case 4**:
   * **Mô tả**: Kiểm tra giá vé tại giá trị biên (tuổi = 12).
   * **Dữ liệu đầu vào**: Tuổi = 12.
   * **Kết quả mong đợi**: Giá vé = $10.

**c. Xác định lỗi tiềm ẩn và đề xuất sửa đổi**

**Lỗi tiềm ẩn**: Đoạn mã không xử lý trường hợp tuổi âm (ví dụ: age = -1), có thể dẫn đến giá vé không hợp lý (trong trường hợp này, -1 < 12 sẽ trả về giá vé $5).

**Đề xuất sửa đổi**:

def calculate\_ticket\_price(age):

if age < 0:

raise ValueError("Age cannot be negative")

if age < 12:

price = 5

elif age <= 60:

price = 10

else:

price = 7

return price

* Thêm kiểm tra age < 0 để ném ngoại lệ nếu tuổi âm.
* Đảm bảo rằng chỉ các giá trị tuổi hợp lệ được xử lý.