■ソフトウェアテスト技法とはなにか？

なるべく少ないテストケースでなるべく多くのバグを発見するための技術

ソフトウェアテストにおいては、テストケースが多いほどバグは発見しやすくなる
しかし、闇雲にテストケースを増やしても意味がない

例：簡単な条件分岐のコード

```
if (a == 3) {}
elif (a == 10) {}
else {}
```

このようなコードがあったとき、必要なテストケースはなんだろうか？

×適切でない例　→　「a=0」「a=1」「a=2」
◎最適な例　→　「a=3」「a=10」「a=1」

上記2つのテストでは、どちらも3つのテストケースを用意している
上のテストケースでは3行目のelseブロックの処理しか確認できないのに対して、
下のテストケースではすべてのブロックの処理を確認することができる
これはテスト技法の「同値クラステスト」という考え方

以下の例ではどのようなテストケースが適当だろうか？

```
if (a <= 3) {}
elif (a >= 10) {}
else {}
```

△適切でない例　→　「a=3」「a=10」「a=5」
◎最適な例　→　「a=3」「a=2」「a=4」「a=10」「a=9」「a=11」

上の例も悪くはないが不安が残る
境界値付近にはバグが潜みやすいので、境界値付近は丁寧に調べるのが良い
「3」や「10」のような境界値に注目して、境界値とその前後をテストしている
これはテスト技法の「境界値テスト」という考え方

ソフトウェアテストのためのテストケースを設計する際に目指すべきことは、
現実的に実行できるレベルの量までテストケースを削減しつつ、
少ないテストケースでも品質を保証できるように最適なテストケースを設計する
ことである
⇒　そのための知識がソフトウェアテスト技法

以下のような、重要かつ最も基本的なテスト技法について学んでいく
技法を学ぶ前の基礎知識（網羅性とピンポイント、テスト設計アプローチ）
同値クラステストと境界値テスト
ドメイン分析テストとデシジョンテーブル
ペア構成テスト（直交表）
状態遷移テスト（状態遷移図と状態遷移表、Nスイッチカバレッジ）

Point!
テスト技法は、なるべく少ないテストケースで高い網羅性を確保するための知識
テスト技法を学ぶ際には、
どのようにして①テストケースを減らすのか、②網羅性を確保するのか
という2点を意識して学ぼう！

■技法を学ぶ前の基礎知識

具体的なソフトウェアテスト技法を学ぶ前に、
・網羅性とピンポイント
・テスト設計アプローチ
という重要な概念について理解しましょう

・網羅性とピンポイント

ソフトウェアテストで重要な2つの概念は「網羅性」と「ピンポイント」
網羅性 → テストの抜け漏れをなるべくなくす
ピンポイント → 怪しいところを入念に調べる

学校の試験勉強で高得点を取るには、試験範囲を網羅するのはもちろん大事だが、
それと同じくらい、試験に出そうな箇所を入念に勉強するのも大事

ソフトウェアテストもなるべく多くのバグを発見するには、
テスト技法によってなるべく網羅性を確保しつつ、
バグが潜んでいそうな箇所や、システムの機能的に特に重要なところについては、
他の箇所よりもテストケースを増やして入念に調べる必要がある

われわれの目標は、より少ないテストケースでより多くのバグを発見すること
網羅性を維持しつつテストケースを減らすテスト技法はもちろん役立つが、
怪しいところを詳しく調べても効率的にバグを見つけることができる

怪しいところを発見するには、テストの観点を広げることが重要
以下3つの方法で観点を広げて怪しいところを見つけ出す
①具体的な例を考え、その例の「間」「逆」「類推」「外側」を考える
②意地悪な条件を考える
③自分が「知っている」「当たり前」と思っていることこそ疑う

・テスト設計アプローチ

以下のようなプログラムのテストについて考えてみる

```
fun cubed(n: Int) {
    return n * n * n
}
```

どのようなテストケースを用意すればよいだろうか？
・整数値が渡された場合だけテストする
・文字列が渡されたときもテストする
・とても大きい整数でテストする
絶対の正解はない、時と場合によって判断する必要があるが、
ひとつの判断基準としてテストの設計アプローチがある

テスト設計アプローチには以下の2つがある
①契約によるテスト
②防御的テスト

①契約によるテスト
事前条件の範囲内だけでテストを行うアプローチ

事前条件とは？
⇒　渡される必要のある引数など、メソッド（ソフトウェアテストの用語ではモジュール）が正しく呼び出される際に満たすべき条件

例えば、上記のメソッドは1~100までの整数しか渡されない（事前条件）と
仕様で決まっていた場合、その範囲でしかテストを行わない

②防御的テスト
事前条件の範囲外もテストを行うアプローチ

例えば、上記のメソッドは1~100までの整数しか渡されない（事前条件）と
仕様で決まっているが、その範囲外（文字列など）のテストも行い、
例外やエラーが出ることを確認する

どちらのアプローチが正解ということはない
大事なのは、どちらのアプローチでテストをするのか？を明確にした上で、
テストを実行すること

Point!
テスト技法はなるべく多くのバグを発見する手段であって、目的ではない
⇒　テスト技法だけでなくピンポイントでもバグを探していこう
2つのテストアプローチを意識できれば、テストの指針になる

■同値クラスと境界値テスト

同値クラステストと境界値テストについてより詳しく見ていく

・同値クラステスト

同値クラスとは？
⇒　テスト的には同じ意味になる値のこと

```
if (a == 3) {}
elif (a == 10) {}
else {}
```

先程みた上の例では「a=1」と「a=2」はどちらもelseブロックの処理が行われる
⇒　テストという観点で見ると同じ意味なので同値クラス

・境界値テスト

境界値テストとは？
⇒　同値クラスの境界と、その前後を調べるテスト

以下のように書きたかったのに、

```
if (a <= 3) {}
elif (a >= 10) {}
else {}
```

以下のように書いてしまったケースは誰しもあるはず

```
if (a <= 3) {}
elif (a > 10) {}
else {}
```

境界値にはバグが潜みやすい
⇒　その付近を調べれば効率的にバグを見つけられる
⇒　これが境界値テスト

・欠陥の検出は**1**つずつ

テストを行う際には、欠陥は他の欠陥を隠すことに注意が必要

例えば、以下のように実装したかったが、

```
if (a <= 3) {//正しい処理}
```

以下のように実装してしまったとする

```
if (a < 3) {//誤った処理}
```

このとき「a=3」というテストケースを実行したときに、
「a<=3」とすべきところを「a<3」としていたことには気づけるが、
ブロックの内部で誤った処理を記述していることには気づけない

このように欠陥は他の欠陥を隠してしまうので、
一つずつ見つけて修正していく必要がある

Point!
テスト的に同じ意味を持つ同値クラスを意識してテストケースをつくろう
境界値にはバグが潜みやすい、境界値テストで確認しよう
欠陥は一つずつ検証することを意識しよう

・演習問題①

小数点第一位までの身長が入力されたとき、身長が165.0cm未満なら「Sサイズ」、
165.0cm以上175.0cm未満なら「Mサイズ」、175.0cm以上なら「Lサイズ」と表示されるシステムをテストするのに必要なテストケースを用意しましょう

■ドメイン分析テストとデシジョンテーブル

ソフトウェアは単純な論理だけでなく、複雑な論理も扱う必要がある
単純な論理　→　スイッチを押したら表示、もう一度押したら非表示
複雑な論理　→　書籍を4000円以上購入した離島以外に住む人は配送料無料

この2つの複雑さの違いは、「条件（変数）」の数の違いによる
単純な論理　→　条件は「スイッチ」のみ
複雑な論理　→　条件は「購入カテゴリ」「購入金額」「住所」の3つ

条件が増えて論理が複雑になると、テストケースの抜け漏れが発生しやすい
⇒　それを防ぐための2つの技法が、ドメイン分析テストとデシジョンテーブル

・ドメイン分析テスト
条件同士が相互作用を持つような場合
言い換えるなら、複数の条件が数式で結ばれている場合に有効

・デシジョンテーブル
条件同士は独立していて、込み入った論理関係を整理する場合
言い換えるなら、複数の条件が論理式で結ばれている場合に有効

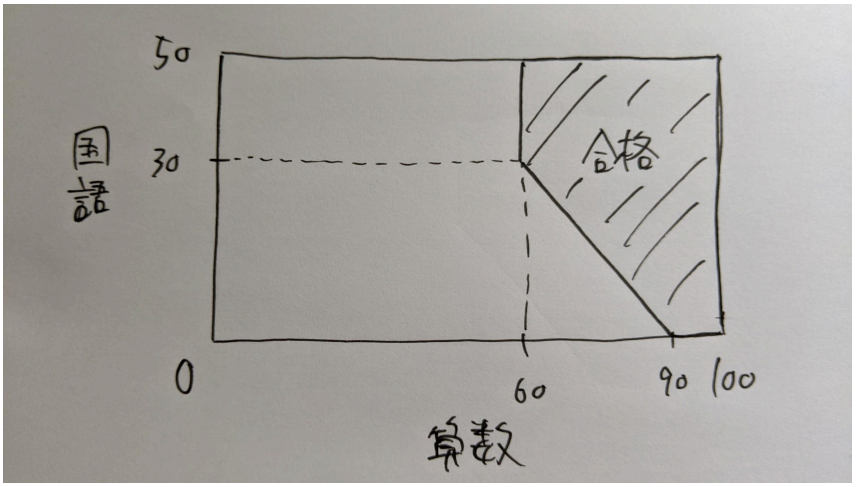
・ドメイン分析テスト

以下のような入試の結果を算出するプログラムについて考える

例題：
A中学の入試では、合格者は国語と算数の成績で決まる
A中学は算数を重視しているので、算数は100点満点で国語は50点満点
合計得点が150点満点中90点以上ならば合格になるが、90点以上の点数を取れていたとしても、算数の点数が60点未満なら不合格になる

合格のための条件を式で表すと以下のようになる
算数の点数 >= 60
算数の点数 + 国語の点数 >= 90

この条件を図で表すと以下のようになる



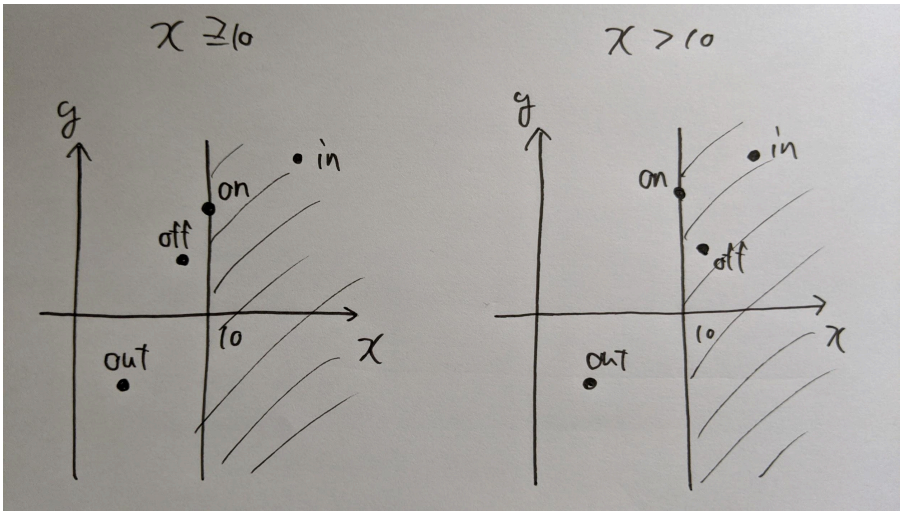
算数の得点によって合格するために必要な国語の点数が変わる、
つまり**2**つの条件（変数）が相互作用を持つので、
この場合はドメイン分析テストによってテストケースを設計するのが適切である

ドメイン分析テストを理解するには、以下の「**4**つのポイント」の理解が必要

ポイント名	概要	x >= 10 の場合	x > 10 の場合
on	境界上の値	10	10
off	境界上にはない境界に隣接する値	9	11
in	on/off以外のドメイン内側の値	15など	15など
out	on/off以外のドメイン外側の値	5など	5など

offポイントは、onポイントがドメイン内ならドメイン外の隣接する値、
onポイントがドメイン外ならドメイン内の隣接する値にする

図で書くと以下のようになる



境界値テストは、onポイントとoffポイントのテストとも言える

```
if (a <= 3) {}
elif (a >= 10) {}
else {}
```

先程見たこの例では、境界値の前後のテストということで、
「a=3」「a=2」「a=4」「a=10」「a=9」「a=11」をテストケースとしたが、
onポイントとoffポイントの観点で言えば、「a=3」「a=4」「a=10」「a=9」の
4つのテストケースでも良い

今回やりたいのは、
算数の点数 >= 60
算数の点数 + 国語の点数 >= 90
という2つの条件を組み合わせて検証すること
⇒　欠陥は他の欠陥を隠すので、検証箇所は一箇所にする必要もある
⇒　着目するひとつの変数だけ**on**ポイントと**off**ポイントを動かして、その他の変数は**in**ポイントに入れておけば一つずつ検証する事ができる
⇒　これを網羅的に行うためにはドメインテストマトリクスを使う

表:ドメインテストマトリクス

変数	条件	ポイント	1	2	3	4
算数	算数の点数 >= 60	on	60			
		off		59		
		in			70	80
合計点（算数/国語）	算数の点数 + 国語の点数 >= 90	on			70/20	
		off				80/9
		in	60/40	59/41		
期待される結果			合格	不合格	合格	不合格

書き方
①条件（変数）をリストアップする
②行に条件（変数）を入れて、of/off/inポイントの行をその中に入れる
③一番下に期待される結果の行を入れる
④一つの条件（変数）以外はinポイントにして、着目する条件のon/offポイントを検証する

ドメインテストマトリクスを使うと、列がそのままテストケースになることに注目

Point!
複数の条件が式で表現できるなら、ドメイン分析テストのドメインテストマトリクスを用いて
条件の相互作用をテストしよう
ドメイン分析テストの4つのポイントを理解しよう

・デシジョンテーブル

「書籍を4000円以上購入した離島以外に住む人は配送料無料」
というケースについて考える

これは論理式で表すことができる
論理式はプログラミングならANDやORで表現することができる
⇒　書籍を購入している AND 4000円以上購入している AND 離島に住んでいない

このケースをデシジョンテーブルで表現すると以下のようになる

表：デシジョンテーブル

	1	2	3	4	5	6	7	8
条件								
書籍を購入している	Yes	Yes	Yes	No	Yes	No	No	No
4000円以上購入している	Yes	Yes	No	Yes	No	Yes	No	No
離島に住んでいない	Yes	No	Yes	Yes	No	No	Yes	No
アクション								
送料無料	Yes	No	No	No	No	No	No	No

条件とその時の結果（アクション）をそれぞれ埋めていくことで書くことができる
今回は2値条件（Yes/NO）が3つあるので、2の3乗で8通りの組み合わせがある
⇒　デシジョンテーブルが8列、列がそのままテストケースになる
⇒　もっと条件が増えた場合、テストケースが膨大になってしまう

場合によっては網羅性を維持しつつ、デシジョンテーブルを圧縮することができる
条件分岐の順番が判明している、かつ順番が一定ならば圧縮が可能

今回の例でもし、
書籍である→3000円以上である→離島ではないという順番で条件分岐が行われ、
どれか一つでも偽なら以降の判定は行わない
ということがわかっていれば以下のようなデシジョンテーブルに圧縮できる

表：圧縮したデシジョンテーブル

	1	2	3	4
条件				
書籍を購入している	Yes	Yes	Yes	No
4000円以上購入している	Yes	Yes	No	ー
離島に住んでいない	Yes	No	ー	ー
アクション				
送料無料	Yes	No	No	No

条件が上から順番に処理されていき、どれか一つでも偽なら以降の判定は行わないという
実装をデシジョンテーブルにも反映している

Point!
複数の条件がANDやORなどを使った論理式として表現できるなら、
デシジョンテーブルを用いて組み合わせを抜けもれなく調べよう
デシジョンテーブルはコードの中身に応じて圧縮しよう

・演習問題②

あるサッカーチームの入団には以下のようなルールが設けられている
・身長は1.70m以上
・体重は65.0kg以上
・BMIは25.0未満（BMI = 体重kg ÷ 身長m ÷ 身長m）
この条件をもとに入団の資格があるかどうかを
ドメインテストマトリクスで文書化して、必要なテストケースを見出しましょう

・演習問題③

整数値の入力に対して、以下の条件を満たすプログラムのテストケースを
デシジョンテーブルを用いて設計しましょう
・3の倍数が入力されると「Fizz」と返す
・5の倍数が入力されると「Buzz」と返す
・15の倍数が入力されると「FizzBuzz」と返す

ここまで見たケースに機能が追加されてさらに条件が増えた場合を考える
より複雑な組み合わせを網羅しようとするとテストケースが膨大になる
⇒　すべてを網羅するのは難しい、方針の切り替えが必要

すべてを網羅することは諦め、少ないテストケースである程度の網羅性を確保した上で、
重要な組み合わせやバグが潜んでいそうな組み合わせについて重点的に追加でテストを行えば、ほとんどのバグは検出できそう
少ないテストケースである程度の網羅性を確保するための技法がペア構成テスト

■ペア構成テスト

すべての条件のすべての組み合わせをテストするのではなく、
条件のすべてのペアをテストする技法

すべてのペアとは？
例：2値条件（0/1）が3つ（条件A, B, C）ある場合
すべての組み合わせ　→　8通り
すべてのペア　→「AとB」「AとC」「BとC」という3通りの条件の組み合わせそれぞれに、「0, 0」「0, 1」「1, 0」「1, 1」の4通りの値

すべてのペアをテストすることである程度の網羅性が確保できる理由は？
ほとんどの欠陥はシングルモード欠陥とダブルモード欠陥のどちらか
シングルモード欠陥　→　モジュールが正しく動かない欠陥
ダブルモード欠陥　　→　2つのモジュールが組み合わさったときに生じる欠陥

すべてのペアを調べれば、シングルモード欠陥とダブルモード欠陥については
網羅できることになるので、ある程度の網羅性を確保できる
ペア構成テストによって70~85%程度のバグを発見することが期待できる

これを見ただけだとすべてのペアを調べるほうが大変そうに思えるが、
すべてのペアを調べるとテストケースが大幅に減ることを直交表で確認しよう

・直交表

すべてのペアを抜けもれなく網羅する方法のひとつ
「2値条件（0/1）が3つ（条件A, B, C）ある場合」を直交表で書くと以下

表：直交表（2値条件が3つ）

	A	B	C
1	0	0	0
2	0	1	1
3	1	0	1
4	1	1	0

直交表を使うと行がテストケースになり、直交表で作られたテストケースを使えば
すべてのペアを網羅することができる
直交表の中にはすべての条件のペアにおいて「0, 0」「0, 1」「1, 0」「1, 1」が
現れていることと、テストケースが4つまで減っていることに注目

「3値条件が4つある場合」は以下のような直交表になる

表：直交表（3値条件が4つ）

	A	B	C	D
1	0	0	0	0
2	0	1	1	1
3	0	2	2	2
4	1	0	1	2
5	1	1	2	0
6	1	2	0	1
7	2	0	2	1
8	2	1	0	2
9	2	2	1	0

すべての組み合わせは3の4乗で81通りあるのに対して、
すべてのペアは直交表を使えば9通りのテストケースでテストできる

この場合でも「0,0」「0,1」「0,2」「1,0」「1,1」「1,2」「2,0」「2,1」「2,2」が
すべてのペアにおいて現れていることに注目

このように直交表を使えばすべてのペアを少ないテストケースで網羅できる
⇒　直交表を使えるようになろう！

直交表は自分でつくるものではない
すでにあるものの中から目的に応じて適切な直交表を選ぶのがテスト実行者の仕事

適切な直交表を選ぶためには、直交表の表記法を知る必要がある

表:L4(2^3)直交表（2値条件が3つ）

	A	B	C
1	0	0	0
2	0	1	1
3	1	0	1
4	1	1	0

上の直交表は4行で2値条件が3つなので「**L4(2^3)**」と表現する

では以下の直交表はどのように表現できるか？

表:9行で3値条件が4つの直交表

	A	B	C	D
1	0	0	0	0
2	0	1	1	1
3	0	2	2	2
4	1	0	1	2
5	1	1	2	0
6	1	2	0	1
7	2	0	2	1
8	2	1	0	2
9	2	2	1	0

解答
L9(3^4)直交表

では8行で2値条件が4つ、4値条件が1つの直交表は？

解答
L8(2^4 4^1)直交表

テストする条件に応じて、以下のような直交表のカタログが乗っているサイトから、適切な直交表を選択するのがテスト実行者の仕事
サイト例①
サイト例②
サイト例③

ここまでを踏まえると、直交表の使い方は以下のような流れになる
①条件をすべて洗い出す
②各条件が取りうる値の数を求める
③①②の条件に合う直交表を選ぶ、完全一致がないなら少し大きめを選ぶ
④直交表にテスト対象の条件を当てはめる

ケーススタディ
ある文書作成ソフトは、以下のような環境で使用される可能性があるとする
・ブラウザ
Google Chrome、Firefox
・OS
Windows、MacOS、Linux
・フォント
明朝体、ゴシック体、筆文字
・文字ぞろえ
左詰め、中央ぞろえ、右詰め

このとき直交表を用いて、すべてのペアのテストケースを作るにはどのようにしたらよいだろうか？

①条件をすべて洗い出す
今回条件は「ブラウザ」「OS」「フォント」「文字ぞろえ」の4つ
②各条件が取りうる値の数を求める
ブラウザが2値条件で、ほか3つは3値条件である
③①②の条件に合う直交表を選ぶ、完全一致がないなら少し大きめを選ぶ
ベストなサイズは「L?(2^1 3^3)」だが、この直交表はなさそう
一番近いのは「L9(3^4)」なのでこれを利用する
④直交表にテスト対象の条件を当てはめる
表：L9(3^4)直交表

	1	2	3	4
1	1	1	1	1
2	1	2	2	2
3	1	3	3	3
4	2	1	2	3
5	2	2	3	1
6	2	3	1	2
7	3	1	3	2
8	3	2	1	3
9	3	3	2	1

1の列をブラウザとしてセルの数字が1ならChrome、2ならFirefoxとして当てはめると以下のようになる

	ブラウザ	2	3	4
1	Chrome	1	1	1
2	Chrome	2	2	2
3	Chrome	3	3	3
4	Firefox	1	2	3
5	Firefox	2	3	1
6	Firefox	3	1	2
7	3	1	3	2
8	3	2	1	3
9	3	3	2	1

他の列も同様に当てはめていくと以下のようになる

	ブラウザ	OS	フォント	文字ぞろえ
1	Chrome	Windows	明朝体	左詰め
2	Chrome	MacOS	ゴシック体	中央ぞろえ
3	Chrome	Linux	筆文字	右詰め
4	Firefox	Windows	ゴシック体	右詰め
5	Firefox	MacOS	筆文字	左詰め
6	Firefox	Linux	明朝体	中央ぞろえ
7	3	Windows	筆文字	中央ぞろえ
8	3	MacOS	明朝体	右詰め
9	3	Linux	ゴシック体	左詰め

ブラウザの列の下3行が残っているが、ここはChromeかFirefoxのどちらかで埋める
この3行を何にしたとしても、すでにすべてのペアは登場しているのでなんでもよい

	ブラウザ	OS	フォント	文字ぞろえ
1	Chrome	Windows	明朝体	左詰め
2	Chrome	MacOS	ゴシック体	中央ぞろえ
3	Chrome	Linux	筆文字	右詰め
4	Firefox	Windows	ゴシック体	右詰め
5	Firefox	MacOS	筆文字	左詰め
6	Firefox	Linux	明朝体	中央ぞろえ
7	Chrome	Windows	筆文字	中央ぞろえ
8	Chrome	MacOS	明朝体	右詰め
9	Firefox	Linux	ゴシック体	左詰め

これで行がテストケースになり、すべてのペアをテストすることができる

以下の流れを意識して、直交表を使えるようになりましょう
①条件をすべて洗い出す
②各条件が取りうる値の数を求める
③①②の条件に合う直交表を選ぶ、完全一致がないなら少し大きめを選ぶ
④直交表にテスト対象の条件を当てはめる

※補足
直交表を使わずに全ペアのテストケースを作成できるツールもある（PICTなど）
直交表を理解してペア構成テストの基本が分かったら、ぜひ調べてみてください

Point!
組み合わせの数が膨大になるときは、直交表を用いてすべてのペアを調べよう
すべてのペアをテストしただけでは網羅性は不十分、重要な組み合わせやバグが潜んでいそうな組み合わせについては追加でテストを行う必要があることを忘れずに

・演習問題④

あるカメラアプリは以下のような機能がある
・撮影モード
写真、動画、スロー動画
・フラッシュ
ON、OFF、自動、常時ON
・画面サイズ
16:9、4:3
・カメラ種類
インカメラ、アウトカメラ、広角インカメラ
・撮影オプション
夜景、ポートレート、文書スキャン、パノラマ

直交表を用いて、このカメラの機能のすべてのペアをテストするような
テストケースを作成しましょう

■状態遷移テスト

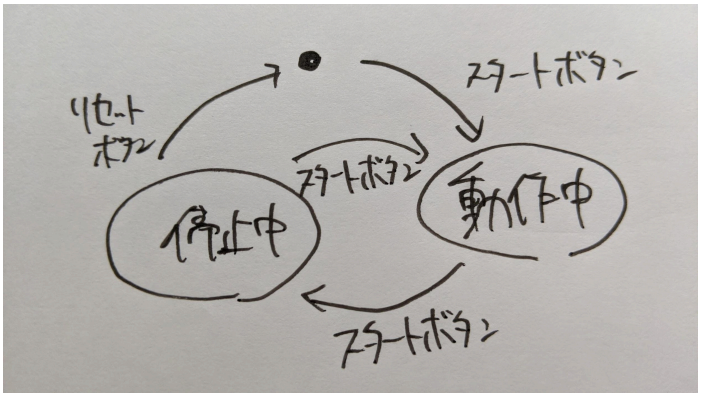
ここまでは条件が取る値によって振る舞いが変わるソフトウェアのテストだった
ここからは現在の状態とその遷移によって振る舞いが変わるようなソフトウェアをテストするための技法である、状態遷移テストについて見ていく

状態遷移テストには目的に応じて2種類あり、
簡単に全体を確認する状態遷移図と、深く詳細を確認する状態遷移表がある

・状態遷移図

以下のような仕様のストップウォッチについて考える
・初期状態でスタートボタンを押すとストップウォッチが動き出す
・動いているときにスタートボタンを押すと停止する
・停止しているときにスタートボタンを押すと再び動き出す
・停止しているときにリセットボタンを押すと初期状態になる

このような仕様があったとき、状態遷移図で表現すると以下のようになる



状態遷移は大きく「状態」「遷移」「イベント」の3つからなる
「イベント」は状態を遷移させるもののこと
初期状態は黒丸（●）、終了状態は目玉印（◉）で表現する

このストップウォッチをテストするには、
どのようなテストケースを用意すればよいだろうか？

ひとつの基準としてすべての遷移がテストされている必要はありそう
このストップウォッチにおけるすべての遷移は以下の4つ
・「初期状態」でスタートボタンを押して「動作中」に遷移する
・「動作中」にスタートボタンを押して「停止中」に遷移する
・「停止中」にスタートボタンを押して「動作中」に遷移する
・「停止中」にリセットボタンを押して「初期状態」に遷移する

これらをテストケースとすることで、すべての遷移だけでなく
すべての状態も一度は確認することができている
これですべてがテストできたのだろうか？
⇒　動作中にリセットボタンを押すなど、仕様にないケースのテストは未完了

仕様にはないが実際にはそのような操作をされる可能性はあるので対策が必要
ある状態でイベントが適用不可のことを**N/A**（**Not Applicable**）と呼ぶ

状態遷移図は簡単な書き方で状態遷移の全体感を把握するのには便利だが、
N/Aも含めて抜けもれなくテストケースを見出すのには向いていない
⇒　状態遷移表を使って網羅的に検討する必要がある

・状態遷移表

ストップウォッチの状態遷移表は以下のようになる

表：ストップウォッチの状態遷移表

イベント＼状態	①初期状態	②動作中	③停止中
スタートボタン	②	③	②
リセットボタン	N/A	N/A	①

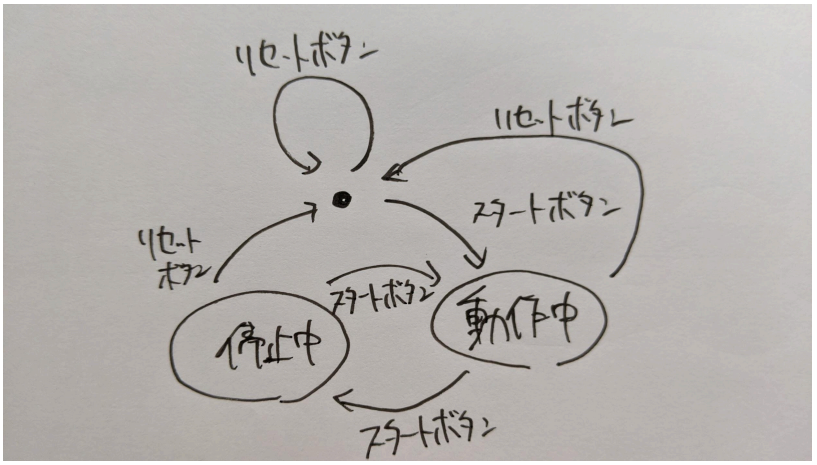
列の状態で行のイベントが起きたときの、遷移先の状態がセルに入っている
これでN/Aも含めた6つの遷移を網羅することができる
状態遷移表をつかうことで、仕様の抜け漏れのチェックもすることができる

今回はいかなる状態でもリセットボタンが押されたら初期状態に戻る仕様とする

表：仕様変更後の状態遷移表

イベント＼状態	①初期状態	②動作中	③停止中
スタートボタン	②	③	②
リセットボタン	①	①	①

状態遷移図も修正すると以下のようになる



仕様の抜け漏れにも気づけたし、これでテストとしては十分だろうか？
⇒ 「停止中」→「初期状態」→「動作中」という遷移のテストはできていない

「停止中」→「初期状態」と「初期状態」→「動作中」という遷移がそれぞれ
テストで動作を確認できていたとしても、状態遷移に伴う内部状態の予期せぬ
変更によって、「停止中」→「初期状態」→「動作中」が動作しないこともある
⇒ **N**スイッチカバレッジのテストを行う

・Nスイッチカバレッジ

Nスイッチカバレッジのテストを実施するには、以下のように
状態遷移表を前状態（行）×後状態（列）の関係行列の形に書き直す必要がある

表：関係行列

前状態＼後状態	①初期状態	②動作中	③停止中
①初期状態	R	S	
②動作中	R		S
③停止中	R	S	

簡単のためにスタートボタンをS、リセットボタンをRで表現している
この関係行列を2乗すると以下のようになる
※行列の計算方法はこの講座の範囲を超えるので説明は省略します。
　行列の計算を自動で行ってくれるExcelの無料ツールもありますので、
　計算の自信がないという方は以下のURLのツールなど使ってみてください
　https://ja.osdn.net/projects/testools/releases/31710

表：1スイッチカバレッジ

前状態＼後状態	①初期状態	②動作中	③停止中
①初期状態	RR+SR	RS	SS
②動作中	RR+SR	RS+SS	
③停止中	RR+SR	RS	SS

①×②セルのRSは、①初期状態の状態のときにリセットボタン（R）を押して、
さらにスタートボタン（S）を押すと、②動作中の状態になるという意味
②×②のセルのRS+SSは、②動作中の状態のときにリセットボタン（R）を押して、
さらにスタートボタン（S）を押すと、②動作中の状態になるし、
もしくは②動作中の状態のときにスタートボタン（S）を押して、
さらにスタートボタン（S）を押すしても、②動作中の状態になるという意味

「停止中」→「初期状態」→「動作中」というように、前状態から後状態までに経由する状態（スイッチ）が1つなので、このような表を**1**スイッチカバレッジという
スイッチが2つなら2スイッチカバレッジ、N個ならNスイッチカバレッジになる
スイッチの数Nに対して、関係行列をN+1乗することでNスイッチカバレッジの表が得られる

参考：2スイッチカバレッジ

前状態＼後状態	①初期状態	②動作中	③停止中
①初期状態	RRR+RSR+SRR+SSR	RRS+SRS+SSS	RSS
②動作中	RRR+RSR+SRR+SSR	RRS+SRS	RSS+SSS
③停止中	RRR+RSR+SRR+SSR	RRS+SRS+SSS	RSS

Nの数が大きくなるほどテストケースが増大することに注目
なのでまずは状態遷移表を使ったすべての遷移の確認を行い、
その上で必要に応じて1スイッチカバレッジ、2スイッチカバレッジというように、
段階的に必要なだけテストを行っていく

Point!
状態遷移図・状態遷移表・Nスイッチカバレッジのテストを、
それぞれの利用目的を理解した上で、求められる品質に応じて使い分けよう

・演習問題⑤

あるDVDプレイヤーは以下のような仕様になっている
・DVDディスクを挿入すると再生が開始される
・再生時に停止ボタンを押すと停止する
・再生時に2倍速ボタンを押すと2倍速再生になる
・2倍速再生時に停止ボタンを押すと停止する
・2倍速再生時に再生ボタンを押すと通常再生になる
・停止状態で再生ボタンを押すと再生される
・DVDディスクが挿入された状態で取り出しボタンを押すと、ディスクを取り出せる

このような場合において、以下の設問を考えましょう
①このDVDプレイヤーの状態遷移図と状態遷移表を作成して、すべての遷移を網羅するようなテストケース作成しましょう
②このDVDプレイヤーの関係行列を作成しましょう
③②の関係行列から1スイッチカバレッジの表を作成しましょう

■総合演習

問1

以下の仕様を持つカレンダーアプリのテストについて考えてみよう
・曜日や祝日によって日付が色分けされている
・祝日と日曜日は赤色
・土曜日は青色
・平日は白色

「曜日と色の対応が正しく設定できているか」を確認するためには、
どのようなテストケースを最低限用意する必要があるでしょうか？

問2

ある動物園の入園チケットを購入できるWebサイトについて考えてみよう
このWebサイトのチケット購入画面には以下2つの入力項目がある
・年齢：数値で入力する
・東京都在住かどうか：都内在住の場合はチェックを入れる
この2つの入力項目を入力すると、Webサイトにはチケットの値段が表示される

そして、動物園の入園料金は以下のルールで決まっている
・12歳以下は無料
・13歳～17歳は200円
・18歳～64歳は600円
・65歳以上は300円
・都内在住の18歳以上は一律300円

入力情報に応じた正しい値段が、Webサイトで表示されることを確認するには、
どのようなテストケースを最低限用意する必要があるでしょうか？

ただし、年齢の入力欄には0~120の整数値が入るものとし、
範囲外の整数値や整数値以外が入力されるケースは考慮しなくて良いものとします

問3

冷暖房と除湿機能を搭載した、あるエアコンについて考えてみよう
このエアコンは以下のような仕様になっている
・停止中に運転ボタンを押すと運転を開始する
・運転中に停止ボタンを押すと運転を停止する
・冷房、暖房、除湿の3つの運転モードがある
・運転中に運転切替ボタンを押すと、運転モードが冷房→暖房→除湿→冷房→...という順番で切り替わっていく
・運転停止時に運転停止前の状態を記憶しておくことで、運転再開時には停止前と同じモードで再開することができる。（例えば、暖房モードで運転停止された場合は、運転再開時には暖房モードで再開する）
・初回運転時は冷房モードで運転が開始される
・運転中に運転ボタンを押しても何も起こらない
・停止中に停止ボタンや運転切替ボタンを押しても何も起こらない

このようなエアコンがあったとき、
①N/Aも含めたすべての遷移
②1スイッチカバレッジ
をテストするために必要なテストケースを用意してみましょう

問4

アメリカのとある大学院における、留学生の入試制度について考えてみよう
この大学院では受験生の足切りとして、TOEFL iBTのスコアを採用している
TOEFL iBTは以下のようなテストである
・Reading、Listening、Writing、Speakingの4つのパートからなる
・各パート30点満点で、合計120点満点
・点数は0以上の整数値

この大学院の足切りを突破するための基準は以下のようになっている
・合計得点が100点以上ならば合格
・ただし、4つのパートの中で1つでも19点以下のパートがあった場合には、合計得点が100点以上でも不合格（例えば、R30, L30, W30, S15なら不合格）

このような入試制度があったときに、各パートの点数を入力にして合否を判定するシステムの動作をテストするために必要な、最低限のテストケースはどのようなものでしょうか？

ただし、各パートの点数の入力として、
0~30の整数値以外は考慮しなくてよいものとします

問5

ある配送会社の配送料は、以下のような料金体系になっている

重量	規格内	規格外
50.0g以下	120円	200円
200.0g以下	240円	320円
500.0g以下	390円	510円
1.0kg以下	取り扱いません	710円
2.0kg以下		1,040円

サイズによって規格内と規格外に分かれており、以下の条件を満たすと規格内
・長辺25.0cm以下
・短辺15.0cm以下
・厚さ3.0cm以下
・重量500.0g以下

このような料金体系になっていたときに、配送物のサイズ（長辺・短辺・厚さ）と重量を入力にして料金を出力するシステムをテストするのに必要なテストケースはどのようなものでしょうか？

ただし、以下の条件があるとします
・「長辺 >= 短辺」が常に成り立つものとし、「長辺 < 短辺」のケースについては考慮する必要はないものとします
・重量は0.0~2,000.0gの範囲で、小数第1位までの値とします
・サイズはそれぞれ0.0~100.0cmの範囲で、小数第1位までの値とします
・重量とサイズに範囲外の数値が入力されるケースについては、考慮しなくて良いものとします
・テストケースは50個以内に収めなければいけないものとします

■カバレッジ（coverage）

ここまで「なるべく少ないテストケースでなるべく多くのバグを発見する」ようなテストケースの作成方法である、ソフトウェアテスト技法について学んできた
テストケースを用意するためのその他の観点として「カバレッジ」がある
カバレッジは、ホワイトボックステストの考え方に基づいている

・ブラックボックステストとホワイトボックステスト

ソフトウェアテストはテストケースの作り方によって大きく2種類に分かれる

ブラックボックステスト
ソフトウェアのコード内部のロジックを考慮せず、
外部から見たときの仕様のみからテストケースを作成する
プログラミングの知識よりも業務に関する知識が必要

ホワイトボックステスト
コード内部のロジックを考慮してテストケースを作成する
ロジックなどを読み取れる必要があるので、ある程度プログラミングの知識が必要

これまで見てきたテスト技法は、ブラックボックステストに分類される
これから見ていくカバレッジは、ホワイトボックステストの考え方

・データフローテストと制御フローテスト

ホワイトボックステストは、データフローテストと制御フローテストに分類される

データフローテスト
それぞれの変数において「生成→使用→廃棄」というサイクルを確認するテスト
変数を生成していないのに使用しようとしていないか？、生成しているのに使用していない変数はないか？など、変数ごとに生成→使用→廃棄の順番が守られているか？を確認する

データフローテストを行えば以下のようなプログラムのエラーを発見できる

```
int x;
if (x == 1) {...}; //値が代入される前に使用してしまっているのでエラー
```

最近では、IDEなどで自動的にやってくれることがほとんどですが、
概念としては一応知っておきましょう

制御フローテスト
コード中の処理がどれだけ実行されたか？を基準に行うテスト
例えば、100行のコードがあった場合、100行全てが一度は実行されるようにテストケースを設計する、もしくは100行のうち何%が実行されたかに注目してテストを行うのが制御フローテスト

以下のようなコードがあったとする

```
if (a > 0) {
  print("aは0より大きい")
}
```

このコードにおいて、すべての処理を実行させるにはどのようなテストケースが必要か？を考えるのが、制御フローテストの観点によるテストケースの設計方法
例えば、「a=1」というテストケースを用意すればすべての処理が実行される

境界値テストの観点でテストケースを作った場合と比較してみよう

カバレッジは制御フローテストの考え方

・カバレッジとは

テスト可能なコードのうち、実際にテストされたコードの割合のこと

例えば、用意したいくつかのテストケースによって、
100行のコードのうち80行が実際に実行されたなら、カバレッジは80%になる

カバレッジはこのような「実行された行数 / 実行可能な行数」という式以外にも、
テストの網羅性の高さ（カバレッジレベル）ごとにいくつか求め方が存在している

代表的なものとしては、以下５つのカバレッジレベルである

・ステートメントカバレッジ
・デシジョンカバレッジ
・コンディションカバレッジ
・複合コンディションカバレッジ
・MC/DCカバレッジ

・5つのカバレッジレベル

レベル1：命令網羅（**statement coverage**）
実行可能な行のうち何行を実行したか、で求められるカバレッジレベル
C0とも表記される

ステートメントカバレッジの網羅率（%）
＝　実行した行数 ÷ 実行可能な行数 × 100

例えば、以下のようなコードについて考えてみる

```
if (a > 0) {
  x = 1
}
if (b == 2) {
  y = 2
}
```

このような場合は、例えば「a=1, b=2」のテストケースを用意すれば、
100%ステートメントカバレッジを達成できる

空行・「 } 」・コメント行などの、
テストする必要のない行（実行可能でない行）は含めずに割合を算出するので、
「a=0, b=2」のテストケースなら、75%ステートメントカバレッジ
「a=0, b=1」のテストケースなら、50%ステートメントカバレッジ

レベル2：判定網羅（**decision coverage**）
判定の分岐をどれだけ網羅したか、で求められるカバレッジレベル
分岐網羅（branch coverage）とも呼ばれたり、C1と表記されることもある

デシジョンカバレッジの網羅率（%）
＝　実現した判定結果の数 ÷ すべての判定の判定結果の合計数 × 100

もう一度、以下のようなコードについて考えてみる

```
if (a > 0) {
  x = 1
}
if (b == 2) {
  y = 2
}
```

このコードには、「a>0」と「b==2」という2つの判定（条件分岐）がある
判定の結果には真と偽の2通りがあるので、2つの判定に2通りの結果で、
合計4通りの以下のような判定結果が考えられる
「a>0」が真、「a>0」が偽、「b==2」が真、「b==2」が偽

この4通りのうち、用意したテストケースによって
何通りが実現したか？を表すのがデシジョンカバレッジ

先程、100%ステートメントカバレッジを達成するために用意した、
「a=1, b=2」のテストケースでは、この4通りの判定結果のうち
「a>0」が真、「b==2」が真、の2つの判定結果だけが実現しているので、
2 ÷ 4 × 100 ＝ 50%デシジョンカバレッジになる

「a>0」と「b==2」の判定が偽のケースをテストできていないので、
「a=1, b=2」に加えて、「a=0, b=1」のテストケースを追加すれば、
100%デシジョンカバレッジを達成することができる

さらに、以下のようなコードについて考えてみる

```
if (a > 0 && c == 1) { // 一つ目の判定
  x = 1
}
if (b == 2 || d < 0) { // 二つ目の判定
  y = 2
}
```

先程とは異なり、一つの判定の中に複数の条件式があるような場合を考えてみよう

このような場合、
100%デシジョンカバレッジを達成するために必要なテストケースの一例は
「a=1, b=2, c=1, d=-1」（一つ目の判定が真、二つ目の判定が真）
「a=0, b=1, c=1, d=2」（一つ目の判定が偽、二つ目の判定が偽）
のような2つなどがある

100%デシジョンカバレッジを達成すると、
自動的に**100%**ステートメントカバレッジも達成できる

レベル3：条件網羅（**condition coverage**）
判定の中の条件式の真偽をどれだけ網羅したか、で求められるカバレッジレベル
C2と表記されることもある

コンディションカバレッジの網羅率（%）
＝　実現した条件式の結果の数 ÷ すべての条件式の結果の合計数 × 100

もう一度、以下のコードについて考えてみる

```
if (a > 0 && c == 1) { // 一つ目の判定
  x = 1
}
if (b == 2 || d < 0) { // 二つ目の判定
  y = 2
}
```

2つの条件判定の中に、条件式が合計で4つ含まれている
それぞれの条件式ごとに真と偽の2通りの結果があるので、
合計8通りの結果がある

先程、このコードで100%デシジョンカバレッジを達成するために用意した
「a=1, b=2, c=1, d=-1」（一つ目の判定が真、二つ目の判定が真）
「a=0, b=1, c=1, d=2」（一つ目の判定が偽、二つ目の判定が偽）
のような2つのテストケースだと、
「c==1」が偽になるパターンがテストされていないので、
7 ÷ 8 × 100 ＝ 87.5%コンディションカバレッジになる

なので100%コンディションカバレッジを達成するには、
以下のようなテストケースを最低限用意する必要がある
「a=1, b=2, c=1, d=-1」（一つ目の判定が真、二つ目の判定が真）
「a=0, b=1, c=2, d=2」（一つ目の判定が偽、二つ目の判定が偽）
こうすると、100%デシジョンカバレッジも同時に達成している

ただし、**100%**コンディションカバレッジだからといって、**100%**ステートメントカバレッジと、
100%デシジョンカバレッジが保証されるわけではないことに注意
例えば、以下のようなテストケースでは、100%デシジョンカバレッジにならない
「a=1, b=2, c=2, d=-1」（一つ目の判定が偽、二つ目の判定が真）
「a=0, b=1, c=1, d=2」（一つ目の判定が偽、二つ目の判定が偽）

すべての判定が一つの条件式からなる場合には、
デシジョンカバレッジとコンディションカバレッジの値は一致する

レベル4：複合条件網羅（**multiple condition coverage**）
条件式の真偽の組み合わせをどれだけ網羅したか、で求められるカバレッジレベル
MCCと表記されることもある

複合コンディションカバレッジの網羅率（%）
＝　実現した条件式の真偽の組み合わせの数 ÷ 条件式の真偽の組み合わせの合計数 × 100

```
if (a > 0 && c == 1) { // 一つ目の判定
  x = 1
}
if (b == 2 || d < 0) { // 二つ目の判定
  y = 2
}
```

例えば、一つ目の「a > 0 && c == 1」という判定は、
①「a > 0」と、②「c == 1」という2つの条件式からなるので、
2つの条件式の真偽の組み合わせは以下の4通りある

	1	2	3	4
①	真	真	偽	偽
②	真	偽	真	偽

「b == 2 || d < 0」という判定も同様に4通りの組み合わせがある

これらの組み合わせをすべて確認する以下のようなテストケースがあれば、
100%複合コンディションカバレッジを達成できる
「a=1, b=2, c=1, d=-1」：真、真、真、真（判定は一つ目が真、二つ目が真）
「a=0, b=1, c=1, d=0」：偽、偽、真、偽（判定は一つ目が偽、二つ目が偽）
「a=1, b=2, c=0, d=0」：真、真、偽、偽（判定は一つ目が偽、二つ目が真）
「a=0, b=1, c=0, d=-1」：偽、偽、偽、真（判定は一つ目が偽、二つ目が真）

100%複合コンディションカバレッジを達成できると、以下が保証される
・100%ステートメントカバレッジ
・100%コンディションカバレッジ
・100%デシジョンカバレッジ

条件式の数が2つならテストケースは4つ、3つなら8つというように、
必要なテストケースが倍々で増えていくのが特徴
網羅性は申し分ないが、テストケースが必要以上に増えてしまう場合が多いため、
実際には複合コンディションカバレッジの代わりに**MC/DC**カバレッジが使われる

レベル5：**MC/DC（modified condition / decision coverage）**
複合コンディションカバレッジを改良（**modified**）して、
必要なテストケースの数を削減したカバレッジレベル

100%複合コンディションカバレッジでは、
条件式の真偽の組み合わせを網羅することで
100%デシジョン/コンディションカバレッジを保証することができた

100%MC/DCカバレッジは、条件式の真偽の組み合わせを網羅することなく、
100%デシジョン/コンディションカバレッジを達成するカバレッジレベル

もう一度、以下のコードについて考えてみる

```
if (a > 0 && c == 1) { // 一つ目の判定
  x = 1
}
if (b == 2 || d < 0) { // 二つ目の判定
  y = 2
}
```

この2つの判定と4つの条件式があったときに、
すべての判定結果（デシジョンカバレッジ）と、
すべての条件式の真偽（コンディションカバレッジ）は、
以下の3つのテストケースだけでも確認することが可能
「a=1, b=2, c=1, d=-1」：真、偽、真、真（判定は一つ目が真、二つ目が真）
「a=0, b=1, c=1, d=0」：偽、偽、真、偽（判定は一つ目が偽、二つ目が偽）
「a=1, b=2, c=0, d=0」：真、真、偽、偽（判定は一つ目が偽、二つ目が真）

MC/DCカバレッジではこのように、判定の中の条件式のそれぞれが、
単独で全体の判定の結果を左右するようにテストケースを用意する必要がある

ANDなら一つでも条件式に偽があれば、判定も偽になるし、
ORなら一つでも条件式に真があれば、判定も真になる

条件式の数が1つ増えるごとにテストケースが倍になる、
複合コンディションカバレッジとは異なり、MC/DCカバレッジなら、
基本的に「条件式の数**+1**」の数だけテストケースを用意すればよい

例えば、3つの条件式がANDでつながっているなら、真偽の組み合わせとして
「真、真、真」「真、真、偽」「真、偽、真」「偽、真、真」の4つ
そして、3つの条件式がORでつながっているなら、真偽の組み合わせとして
「偽、偽、偽」「真、偽、偽」「偽、真、偽」「偽、偽、真」の4つ

航空機ソフトウェアのテスト時に使用する網羅度として開発されたカバレッジ
航空機ソフトウェアのほかにも、自動車システムや銀行システムなどのような、
ソフトウェアのバグ発生が、人命に関わる・社会的に大きな影響になるなど、
ミッションクリティカルなソフトウェアに採用されるカバレッジレベル

このように、100%複合コンディションカバレッジよりも少ないテストケースで、
・100%ステートメントカバレッジ
・100%デシジョンカバレッジ
・100%コンディションカバレッジ
を達成できるので、100%MC/DCカバレッジの方がよく使われる

Point!
テスト技法以外のテストケースを設計する観点としてカバレッジがある
これは、ホワイトボックステストの制御フローテストに基づくもので、
テスト可能なコードのうち、実際にテストされたコードの割合のこと
カバレッジは網羅度の高さごとに、以下5つのカバレッジレベルに分かれている

・ステートメントカバレッジ（**C0**）
実行可能な行のうち何行を実行したか
・デシジョンカバレッジ（**C1**）
判定の分岐をどれだけ網羅したか
・コンディションカバレッジ（**C2**）
判定の中の条件式の真偽をどれだけ網羅したか
・複合コンディションカバレッジ（**MCC**）
条件式の真偽の組み合わせをどれだけ網羅したか
・**MC/DC**カバレッジ
複合コンディションカバレッジを改良して、必要なテストケースの数を削減

航空機システムなどのミッションクリティカルなシステムでは、
MC/DCが採用すべきカバレッジレベルになるが、
通常のWeb・モバイルのアプリケーションなど、
ミッションクリティカルでないシステムでは、**C0**か**C1**が基準になることが多い

カバレッジを自動で計算してくれるカバレッジ計測ツールが存在するので、
実用の場面では、カバレッジは手計算ではなく、ツールを用いて求めるのが基本

各カバレッジの数値を見た時に、
その数値の意味がわかるようになっておきましょう

・カバレッジの3つの特徴

①カバレッジが高いからといって、品質を保証できているとは限らない

例えば、以下のコードについて考えてみる

```
int a = b / c;
```

このようなコードがあったとき、例えば「b=6, c=2」という一つのテストケースを用意すれば、**100%**ステートメントカバレッジを達成できるが、実際には「c=0」のテストケースで、ゼロ除算によるエラーが出てしまうバグが潜んでいる

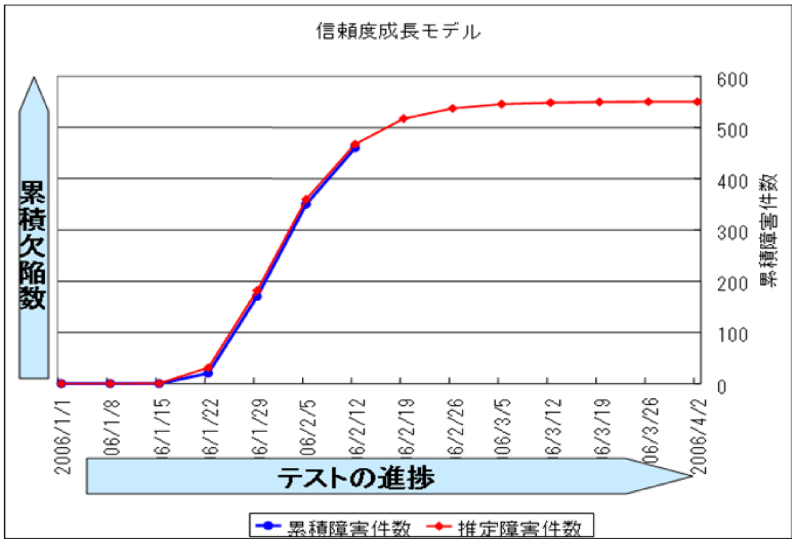
→　100%ステートメントカバレッジだからと言って品質が保証されるとは限らない

他にも、以下のような欠陥を見逃してしまう

・書かれているべきコードが書かれていない

・境界値などの実装を間違えてしまっている

②カバレッジを**100%**に近づけるほど、バグ検出の費用対効果が下がる



※https://www.ipa.go.jp/files/000070064.pdf

実施したテスト数と検出したバグの数で軸を取りグラフにすると、

上図のようなグラフの形になることが一般的

テストの初期段階では、行ったテストに対して多くのバグが検出されるが、

ある程度テストを実施すると徐々にバグが検出されなくなっていき、

最後はテストケースを増やしても、ほとんどバグが見つからなくなる

③カバレッジの数値がテストの目的にすり替わりがち
例えば、100%ステートメントカバレッジを達成することをテストの目標にしてしまうと、カバレッジの数値は本来、品質を保証するための手段であるにもかかわらず、100%という数値そのものが目標になってしまいがち

先述の通り、カバレッジの数値と品質は必ずしも一致しないので、
カバレッジの数値をやみくもに追い求めるのは危険

・カバレッジをどう活用する？

このような特徴からカバレッジは以下のように活用するのがおすすめ

①テストケースはカバレッジではなく、テスト技法の考え方を元に作成する
カバレッジはあくまでも、テスト技法の補助として使おう
テスト技法を活用して用意したテストケースでテストを行った上で、
それでもまだカバレッジが低いならテストケースの追加を検討する

具体的なテストの進め方
テスト技法を用いてテストケースを設計する
→そのテストケースを実行してみてカバレッジを計算する
→低いようならピンポイントの観点でテストケースを追加する

カバレッジを上げることを目的にしないで、効果の高いテストを行った結果、
高いカバレッジが副次的に得られる、という理解をしておく

②**100%**のカバレッジを目指さず、**85%**程度のカバレッジを「努力目標」にする
Googleでは85%ステートメントカバレッジが一つの目安※

コンパイル時にバグが発覚する、Javaのような静的型付け言語よりも、
実行するまでバグが発覚しない、Pythonのような動的型付け言語の方が
基準となるカバレッジを高くすることが多い

ミッションクリティカルなサービスでなければ、
ユーザーに使ってもらってバグを見つけてもらうという方針にするのもあり

カバレッジをテスト終了の基準や、ソフトウェアの品質の基準にはせず、
達成した方がいいけど、達成しなくてもいい努力目標として活用しよう

・演習問題⑥

以下のようなコードについて考えてみましょう

```
if (a == 0 && b > 0 && c < 0) { //一つ目の判定
  print("あ")
}
if (x == 0 || y > 0 || z < 0) { //二つ目の判定
  print("い")
}
```

このコードにおいて、以下のそれぞれのカバレッジを達成するためには、
どのようなテストケースが最低限必要でしょうか？
①100%デシジョンカバレッジ
②100%コンディションカバレッジ
③100%MC/DCカバレッジ

また、このコードをテストするのに最低限必要なテストケースを、
テスト技法を用いて設計してみましょう
そして、その時に得られる①～③のカバレッジレベルをそれぞれ求めましょう

ただし、以下のような条件があるとします
・すべての変数には、整数しか入らず、上限値と下限値のテストは不要です
・条件式の処理の順番を考慮して、なるべくテストケースを削減してください

■まとめと注意点

・テストの目的を常に意識しよう

テストを行う目的を見失って以下のような過ちが起こりがち
・テストケースの作成が目的になってしまう
・テスト技法の使用が目的になってしまう
・カバレッジの数値が目的になってしまう
・テストを行うこと自体が目的になってしまう

あくまでもソフトウェアテストは少しでも多くのバグを発見して、
ソフトウェアの品質を向上させるためのものであり、
テスト技法はなるべく少ないテストケースでなるべく多くのバグを発見するための技術であることを忘れずに

・品質の向上すら手段である

品質の向上が目的になってもいけない
品質の向上は、ソフトウェアでビジネスを成功させるための手段である

常にテストを行うそもそもの目的を意識して、
適切なテストを適切な量だけ行えるようになろう

・実際にやらないと身につかない

テスト技法は実際に使っていかないと身につかない
実際の開発で使うのはもちろん、以下書籍など活用して演習を積んでいこう
ソフトウェアテスト技法練習帳 ~知識を経験に変える40問~

■演習問題の解答

・演習問題①

小数点第一位までの身長が入力されたとき、身長が165.0cm未満なら「Sサイズ」、
165.0cm以上175.0cm未満なら「Mサイズ」、175.0cm以上なら「Lサイズ」と表示されるシステムをテストするのに必要なテストケースを用意しましょう

同値クラスは3つ（Sサイズ、Mサイズ、Lサイズ）
境界値は2つ（165cm、175cm）なので、必要なテストケースの一例としては
「164.9→S」「165.0→M」「165.1→M」「174.9→M」「175.0→L」「175.1→L」の6つが必要

・演習問題②

あるサッカーチームの入団には以下のようなルールが設けられている
・身長は1.70m以上
・体重は65.0kg以上
・BMIは25.0未満（BMI = 体重kg ÷ 身長m ÷ 身長m）
この条件をもとに入団の資格があるかどうかを
ドメインテストマトリクスで文書化して、必要なテストケースを見出しましょう

BMIは身長と体重によって決まるので、複数の条件が相互作用を持つ
⇒ ドメイン分析が最適

表：サッカーチーム入団可否のドメインテストマトリクス

変数	条件	ポイント	1	2	3	4	5	6
身長（m）	身長 >= 1.70	on	1.70					
		off		1.69				
		in			1.80	1.75	1.80	1.90
体重（kg）	体重 >= 65.0	on			65.0			
		off				64.9		
		in	70.0	66.0			81.0	89.9
BMI（体重kg ÷ 身長m ÷ 身長m）	BMI < 25	on					25.0	
		off						24.9
		in	24.2	23.1	20.1	21.2		
期待される結果			可	不可	可	不可	不可	可

6つの列が最低限のテストケースになる

・演習問題③

整数値の入力に対して、以下の条件を満たすプログラムのテストケースを
デシジョンテーブルを用いて設計しましょう
・3の倍数が入力されると「Fizz」と返す
・5の倍数が入力されると「Buzz」と返す
・15の倍数が入力されると「FizzBuzz」と返す

条件は「3の倍数」「5の倍数」「15の倍数」の3つ
これら3つの条件が論理式の関係で表現できるのでデシジョンテーブルを使う
デシジョンテーブルで表現すると以下のようになる

表：FizzBuzzのデシジョンテーブル

	1	2	3	4	5	6	7	8
条件								
3の倍数	Yes	Yes	Yes	No	Yes	No	No	No
5の倍数	Yes	Yes	No	Yes	No	Yes	No	No
15の倍数	Yes	No	Yes	Yes	No	No	Yes	No
アクション								
戻り値	Fizz Buzz	✖	✖	✖	Fizz	Buzz	✖	入力値

テスト設計アプローチに応じて、
有効な4ケースのみについてテスト（契約によるテスト）するか、
無効な4ケースも含めてテスト（防御的テスト）するかを決定する

防御的テストのためのテストケースを作ることは出来ないので、
このようなケースが起こらないことをソースコードで確認するなどする

特に、条件判定の順番が「15の倍数」→「3の倍数」→「5の倍数」ということが
分かっているなら以下のように圧縮できる

表：圧縮したデシジョンテーブル

	1	2	3	4
条件				
15の倍数	Yes	No	No	No
3の倍数	―	Yes	No	No
5の倍数	―	―	Yes	No
アクション				
戻り値	FizzBuzz	Fizz	Buzz	入力値

「15の倍数」は「3の倍数 かつ 5の倍数」なので、
以下のようなデシジョンテーブルも考えられる

	1	2	3	4
条件				
3の倍数	Yes	Yes	No	No
5の倍数	Yes	No	Yes	No
アクション				
戻り値	FizzBuzz	Fizz	Buzz	入力値

内部のロジックに応じて最適なデシジョンテーブルを選択しよう

・演習問題④

あるカメラアプリは以下のような機能がある
・撮影モード
写真、動画、スロー動画
・フラッシュ
ON、OFF、自動、常時ON
・画面サイズ
16:9、4:3
・カメラ種類
インカメラ、アウトカメラ、広角インカメラ
・撮影オプション
夜景、ポートレート、文書スキャン、パノラマ

直交表を用いて、このカメラの機能のすべてのペアをテストするような
テストケースを作成しましょう

---

①条件をすべて洗い出す
条件は以下の5つ
「撮影モード」「フラッシュ」「画面サイズ」「カメラ種類」「撮影オプション」

②各条件が取りうる値の数を求める
2値条件が1つ、3値条件が2つ、4値条件が2つなので、
「L?(2^1 3^2 4^2)」が最小サイズ

③①②の条件に合う直交表を選ぶ、完全一致がないなら少し大きめを選ぶ
ジャストサイズはないので、最も近い「L16(4^5)」を用いる

表:L16(4^5)直交表

	1	2	3	4	5
1	1	1	1	1	1
2	1	2	2	2	2
3	1	3	3	3	3
4	1	4	4	4	4
5	2	1	2	3	4
6	2	2	1	4	3
7	2	3	4	1	2
8	2	4	3	2	1
9	3	1	3	4	2
10	3	2	4	3	1
11	3	3	1	2	4
12	3	4	2	1	3
13	4	1	4	2	3
14	4	2	3	1	4
15	4	3	2	4	1
16	4	4	1	3	2

④直交表にテスト対象の条件を当てはめる

	撮影モード	フラッシュ	画面サイズ	カメラ種類	オプション
1	写真	ON	16:9	インカメラ	夜景
2	写真	OFF	4:3	アウトカメラ	ポートレート
3	写真	自動	3	広角インカメラ	文書スキャン
4	写真	常時ON	4	4	パノラマ
5	動画	ON	4:3	広角インカメラ	パノラマ
6	動画	OFF	16:9	4	文書スキャン
7	動画	自動	4	インカメラ	ポートレート
8	動画	常時ON	3	アウトカメラ	夜景
9	スロー動画	ON	3	4	ポートレート
10	スロー動画	OFF	4	広角インカメラ	夜景
11	スロー動画	自動	16:9	アウトカメラ	パノラマ
12	スロー動画	常時ON	4:3	インカメラ	文書スキャン
13	4	ON	4	アウトカメラ	文書スキャン
14	4	OFF	3	インカメラ	パノラマ
15	4	自動	4:3	4	夜景
16	4	常時ON	16:9	広角インカメラ	ポートレート

当てはめられていないセルにも当てはめると以下のようになる

	撮影モード	フラッシュ	画面サイズ	カメラ種類	オプション
1	写真	ON	16:9	インカメラ	夜景
2	写真	OFF	4:3	アウトカメラ	ポートレート
3	写真	自動	16:9	広角インカメラ	文書スキャン
4	写真	常時ON	4:3	インカメラ	パノラマ
5	動画	ON	4:3	広角インカメラ	パノラマ
6	動画	OFF	16:9	広角インカメラ	文書スキャン
7	動画	自動	4:3	インカメラ	ポートレート
8	動画	常時ON	16:9	アウトカメラ	夜景
9	スロー動画	ON	16:9	インカメラ	ポートレート
10	スロー動画	OFF	4:3	広角インカメラ	夜景
11	スロー動画	自動	16:9	アウトカメラ	パノラマ
12	スロー動画	常時ON	4:3	インカメラ	文書スキャン
13	写真	ON	4:3	アウトカメラ	文書スキャン
14	写真	OFF	16:9	インカメラ	パノラマ
15	動画	自動	4:3	アウトカメラ	夜景
16	スロー動画	常時ON	16:9	広角インカメラ	ポートレート

この16行がテストケースになり、すべてのペアをテストできる

・演習問題⑤

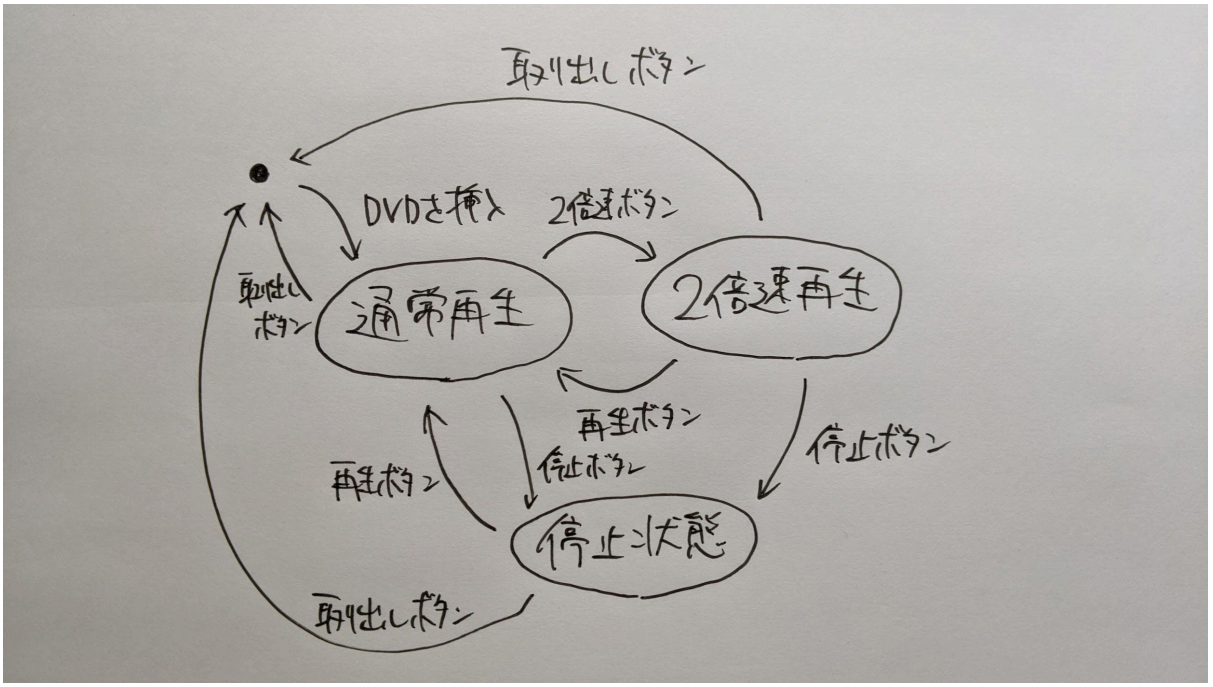
あるDVDプレイヤーは以下のような仕様になっている
・DVDディスクを挿入すると再生が開始される
・再生時に停止ボタンを押すと停止する
・再生時に2倍速ボタンを押すと2倍速再生になる
・2倍速再生時に停止ボタンを押すと停止する
・2倍速再生時に再生ボタンを押すと通常再生になる
・停止状態で再生ボタンを押すと再生される
・DVDディスクが挿入された状態で取り出しボタンを押すと、ディスクを取り出せる

このような場合において、以下の設問を考えましょう
①このDVDプレイヤーの状態遷移図と状態遷移表を作成して、すべての遷移を網羅するようなテストケース作成しましょう
②このDVDプレイヤーの関係行列を作成しましょう
③②の関係行列から1スイッチカバレッジの表を作成しましょう

---

①このDVDプレイヤーの状態遷移図と状態遷移表を作成して、すべての遷移を網羅するようなテストケース作成しましょう

図：DVDプレイヤーの状態遷移図



表：DVDプレイヤーの状態遷移表

イベント \ 状態	⓪初期状態	①通常再生	②停止状態	③2倍速再生
DVD挿入	①	N/A	N/A	N/A
再生ボタン	N/A	N/A	①	①
停止ボタン	N/A	②	N/A	②
2倍速ボタン	N/A	③	N/A	N/A
取り出しボタン	N/A	⓪	⓪	⓪

②このDVDプレイヤーの関係行列を作成しましょう

表:DVDプレイヤーの関係行列

前状態 \ 後状態	⓪初期状態	①通常再生	②停止状態	③2倍速再生
⓪初期状態		D		
①通常再生	T		S	2X
②停止状態	T	P		
③2倍速再生	T	P	S	

D → DVD挿入
P → 再生ボタン(Play)
S → 停止ボタン(Stop)
2X → 2倍速ボタン
T → 取り出しボタン(Take Out)

③②の関係行列から1スイッチカバレッジの表を作成しましょう

表:1スイッチカバレッジ

前状態 \ 後状態	⓪初期状態	①通常再生	②停止状態	③2倍速再生
⓪初期状態	DT		DS	D2X
①通常再生	ST+2XT	TD+SP+2XP	2XS	
②停止状態	PT	TD	PS	P2X
③2倍速再生	PT+ST	TD+SP	PS	P2X

D → DVD挿入
P → 再生ボタン(Play)
S → 停止ボタン(Stop)
2X → 2倍速ボタン
T → 取り出しボタン(Take Out)

・演習問題⑥

以下のようなコードについて考えてみましょう

```
if (a == 0 && b > 0 && c < 0) { //一つ目の判定
  print("あ")
}
if (x == 0 || y > 0 || z < 0) { //二つ目の判定
  print("い")
}
```

このコードにおいて、以下のそれぞれのカバレッジを達成するためには、
どのようなテストケースが最低限必要でしょうか？
①100%デシジョンカバレッジ
②100%コンディションカバレッジ
③100%MC/DCカバレッジ

また、このコードをテストするのに最低限必要なテストケースを、
テスト技法を用いて設計してみましょう
そして、その時に得られる①～③のカバレッジレベルをそれぞれ求めましょう

ただし、以下のような条件があるとします
・すべての変数には、整数しか入らず、上限値と下限値のテストは不要です
・条件式の処理の順番を考慮して、なるべくテストケースを削減してください

それぞれのカバレッジを達成するために必要なテストケースの一例は以下
①100%デシジョンカバレッジ（判定の真偽を網羅）
「a=0, b=1, c=-1, x=0, y=0, z=0 → あ、い」：判定は一つ目が真、二つ目が真
「a=1, b=0, c=0, x=1, y=0, z=0 → 表示なし」：判定は一つ目が偽、二つ目が偽

②100%コンディションカバレッジ（条件式の真偽を網羅）
「a=0, b=1, c=-1, x=0, y=1, z=-1 → あ、い」：真、真、真、真、真、真
「a=1, b=0, c=0, x=1, y=0, z=0 → 表示なし」：偽、偽、偽、偽、偽、偽

③100%MC/DCカバレッジ（判定と条件式の真偽を網羅）
「a=0, b=1, c=-1, x=1, y=0, z=0 → あ」
「a=1, b=1, c=-1, x=0, y=0, z=0 → い」
「a=0, b=0, c=-1, x=1, y=1, z=0 → い」
「a=0, b=1, c=0, x=1, y=0, z=-1 → い」

条件式が3つあるので、4つのテストケースが最低限必要

一つ目の判定:「真、真、真」「偽、真、真」「真、偽、真」「真、真、偽」
二つ目の判定:「偽、偽、偽」「真、偽、偽」「偽、真、偽」「偽、偽、真」

続いて、テスト技法を用いてテストケースを設計していきましょう

このコードは「a==0」や「b>0」といった境界値を持つ条件式が、
ANDや**OR**の論理式で複数組み合わさっているので、以下2つの技法が活用できる
①境界値テスト・同値クラステスト
②デシジョンテーブル

①境界値テスト・同値クラステスト
6つの条件式それぞれが境界値を1つ持つので、
それぞれの境界値を調べるためには、以下のテストケースが必要
on **off**
「a=0」「a=-1」「a=1」
「b=0」「b=1」
「c=0」「c=-1」
「x=0」「x=-1」「x=1」
「y=0」「y=1」
「z=0」「z=-1」

「a==0」のような等号の式においては、**off** ポイントは2つある

②デシジョンテーブル
それぞれの判定には、3値条件が1つ、2値条件が2つあることになるので
これらの条件の組み合わせをデシジョンテーブルで網羅すると以下のようになる

一つ目の判定のデシジョンテーブル

	1	2	3	4	5	6	7	8	9	10	11	12
条件												
a	0	0	0	0	-1	-1	-1	-1	1	1	1	1
b	1	1	0	0	1	1	0	0	1	1	0	0
c	-1	0	-1	0	-1	0	-1	0	-1	0	-1	0
アクション												
出力	あ	ー	ー	ー	ー	ー	ー	ー	ー	ー	ー	ー

二つ目の判定のデシジョンテーブル

	1	2	3	4	5	6	7	8	9	10	11	12
条件												
x	0	0	0	0	-1	-1	-1	-1	1	1	1	1
y	1	1	0	0	1	1	0	0	1	1	0	0
z	-1	0	-1	0	-1	0	-1	0	-1	0	-1	0
アクション												
出力	い	い	い	い	い	い	い	―	い	い	い	―

条件式の処理の順番を考慮して、デシジョンテーブルを圧縮したものが以下

一つ目の判定のデシジョンテーブル（圧縮ver.）

	1	2	3	4	5
条件					
a	0	0	0	-1	1
b	1	1	0	―	―
c	-1	0	―	―	―
アクション					
出力	あ	―	―	―	―

二つ目の判定のデシジョンテーブル（圧縮ver.）

	1	2	3	4	5	6	7
条件							
x	0	−1	−1	−1	1	1	1
y	―	1	0	0	1	0	0
z	―	―	-1	0	―	-1	0
アクション							
出力	い	い	い	―	い	い	―

「ー」の箇所を、「2」もしくは「−2」で適当に埋めて、
この2つのデシジョンテーブルをまとめると以下のようになる

	1	2	3	4	5	6	7
条件							
a	0	0	0	-1	1	**0**	**0**
b	1	1	0	**2**	**-2**	**1**	**1**
c	-1	0	**2**	**-2**	**2**	**0**	**-1**
x	0	−1	−1	−1	1	1	1
y	**2**	1	0	0	1	0	0
z	**-2**	**2**	-1	0	**-2**	-1	0
アクション							
出力	あ、い	い	い	―	い	い	あ

※太字は適当に埋めた部分

この7列がテスト技法を用いて設計したテストケースになる
このテストケースでどれだけのカバレッジが得られるか確認してみよう

①デシジョンカバレッジ
それぞれの判定の真偽は「あ」と「い」が出力されているかで判断できるので、
それぞれ出力される場合と、されない場合の両方があるので、
100%デシジョンカバレッジは達成できている

②コンディションカバレッジ
すべての条件式の真偽が一度は確認されているので、
100%コンディションカバレッジも達成できている

③MC/DCカバレッジ
この7つのテストケースで、以下の組み合わせが実現しているかを確認する
一つ目の判定:「真、真、真」「偽、真、真」「真、偽、真」「真、真、偽」
二つ目の判定:「偽、偽、偽」「真、偽、偽」「偽、真、偽」「偽、偽、真」

それぞれの値を条件式の真偽に書き換えると以下のようになる

	1	2	3	4	5	6	7
条件							
a	真	真	真	偽	偽	真	真
b	真	真	偽	真	偽	真	真
c	真	偽	偽	真	偽	偽	真
x	真	偽	偽	偽	偽	偽	偽
y	真	真	偽	偽	真	偽	偽
z	真	偽	真	偽	真	真	偽
アクション							
出力	あ、い	い	い	―	い	い	あ

MC/DCの必要な8通りのテストケースのうち、6通りはテストできているので
6 ÷ 8 × 100 ＝ 75%MC/DCカバレッジになる
このように、デシジョンテーブルを圧縮するとカバレッジが低下することもある

「―」の部分や、一つ目の判定の6,7列目を適当に埋めずに、
MC/DCカバレッジを意識して埋めると以下のようになり、
100%MC/DCカバレッジを達成できる

	1	2	3	4	5	6	7
条件							
a	0	0	0	-1	1	0	0
b	1	1	0	2	-2	1	**0**
c	-1	0	2	-2	2	0	-1
x	0	−1	−1	−1	1	1	1
y	**-2**	1	0	0	1	0	0
z	**2**	2	-1	0	-2	-1	0
アクション							
出力	あ、い	い	い	―	い	い	―

※太字は変更部分

■総合演習の解答

問1

以下の仕様を持つカレンダーアプリのテストについて考えてみよう
・曜日や祝日によって日付が色分けされている
・祝日と日曜日は赤色
・土曜日は青色
・平日は白色

「曜日と色の対応が正しく設定できているか」を確認するためには、
どのようなテストケースを最低限用意する必要があるでしょうか？

・祝日かどうか
・日曜日かどうか
・土曜日かどうか
・平日かどうか
といった複数の条件が、「祝日でないかつ土曜日」というような形で、
ANDを用いた論理式で結ばれているので、
デシジョンテーブルを用いて複数の条件の組み合わせを網羅的に調べると良さそう

この4つの条件の組み合わせを、起こりえないものも含めて
デシジョンテーブルで整理すると以下のようになる

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
条件																
祝日	Yes	Yes	Yes	Yes	No	Yes	Yes	No	Yes	No	No	Yes	No	No	No	No
日曜日	Yes	Yes	Yes	No	Yes	Yes	No	Yes	No	Yes	No	No	Yes	No	No	No
土曜日	Yes	Yes	No	Yes	Yes	No	Yes	Yes	No	No	Yes	No	No	Yes	No	No
平日	Yes	No	Yes	Yes	Yes	No	No	No	Yes	Yes	Yes	No	No	No	Yes	No
アクション																
色	ー	ー	ー	ー	ー	赤	赤	ー	赤	ー	ー	ー	赤	青	白	ー

Yes/Noの2値条件が4つなので、2の4乗で16通り
防御的テストを行うならこの16列がテストケースになる

「土曜日かつ平日」のような起こりえないものを取り除くと、以下のようになる

	1	2	3	4	5	6
条件						
祝日	Yes	Yes	Yes	No	No	No
日曜日	Yes	No	No	Yes	No	No
土曜日	No	Yes	No	No	Yes	No
平日	No	No	Yes	No	No	Yes
アクション						
色	赤	赤	赤	赤	青	白

契約によるテストならば、この6列のテストケースでよい

条件判定が「祝日かどうか」→「曜日は何か」の順番で行われることが分かっているなら、
以下のように圧縮することもできる

	1	2	3	4
条件				
祝日	Yes	No	No	No
日曜日	―	Yes	No	No
土曜日	―	No	Yes	No
平日	―	No	No	Yes
アクション				
色	赤	赤	青	白

また、「土曜日かどうか」「日曜日かどうか」という条件を、
「曜日」という条件で表現するとデシジョンテーブルは以下のように書ける

	1	2	3	4	5	6
条件						
祝日	Yes	Yes	Yes	No	No	No
曜日	日	土	平日	日	土	平日
アクション						
色	赤	赤	赤	赤	青	白

デシジョンテーブルはこのような書き方もできることも知っておこう

問2

ある動物園の入園チケットを購入できるWebサイトについて考えてみよう
このWebサイトのチケット購入画面には以下2つの入力項目がある
・年齢：数値で入力する
・東京都在住かどうか：都内在住の場合はチェックを入れる
この2つの入力項目を入力すると、Webサイトにはチケットの値段が表示される

そして、動物園の入園料金は以下のルールで決まっている
・12歳以下は無料
・13歳～17歳は200円
・18歳～64歳は600円
・65歳以上は300円
・都内在住の18歳以上は一律300円

入力情報に応じた正しい値段が、Webサイトで表示されることを確認するには、
どのようなテストケースを最低限用意する必要があるでしょうか？

ただし、年齢の入力欄には0~120の整数値が入るものとし、
範囲外の整数値や整数値以外が入力されるケースは考慮しなくて良いものとします

「年齢」と「都内在住かどうか」という2つの条件から料金が決定されます
なので、以下の2つのことについて確認する必要があります
①年齢の境界値と同値クラスが正しく設定されているか？
②**2**つの条件の組み合わせを網羅できているか？

①には、境界値テスト・同値クラステストの考え方、
②には、デシジョンテーブルが役立ちそうです

まずは都内在住かどうかは置いておいて、年齢の条件だけに注目してみよう
境界値は13、18、65の3つなので、
同値クラスは4つ（無料、200円、600円、300円）

なので必要そうなテストケースとしては、on/offポイントの観点で考えると、
「12歳→無料」「13歳→200円」「17歳→200円」「18歳→600円」「64歳→600円」「65歳→300円」の、6つのテストケースは最低限必要

※入力値の下限0と上限120の場合のテストケースを追加してもよい

この年齢の条件に加えて、都内在住かどうかという条件もあるので、
境界値テストの観点で見出した、年齢の6つのテストケースに対して、
都内在住の場合とそうでない場合の2通りのテストをそれぞれ作成する必要がある

「年齢」という**6**値条件が**1**つと、「都内在住かどうか」という**2**値条件が**1**つある
なので、6×2=12通りのテストケースが最低限必要

デシジョンテーブルにすると以下のようになる

	1	2	3	4	5	6	7	8	9	10	11	12
条件												
都内在住	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No	No	No	No
年齢	12	13	17	18	64	65	12	13	17	18	64	65
アクション												
料金	無料	200円	200円	300円	300円	300円	無料	200円	200円	600円	600円	300円

この12列のテストケースは最低限実施する必要がある

このような、複数のテスト技法を組み合わせたテストの設計もできるようになっておきましょう！

問3

冷暖房と除湿機能を搭載した、あるエアコンについて考えてみよう
このエアコンは以下のような仕様になっている
・停止中に運転ボタンを押すと運転を開始する
・運転中に停止ボタンを押すと運転を停止する
・冷房、暖房、除湿の3つの運転モードがある
・運転中に運転切替ボタンを押すと、運転モードが冷房→暖房→除湿→冷房→...という順番で切り替わっていく
・運転停止時に運転停止前の状態を記憶しておくことで、運転再開時には停止前と同じモードで再開することができる。（例えば、暖房モードで運転停止された場合は、運転再開時には暖房モードで再開する）
・初回運転時は冷房モードで運転が開始される
・運転中に運転ボタンを押しても何も起こらない
・停止中に停止ボタンや運転切替ボタンを押しても何も起こらない

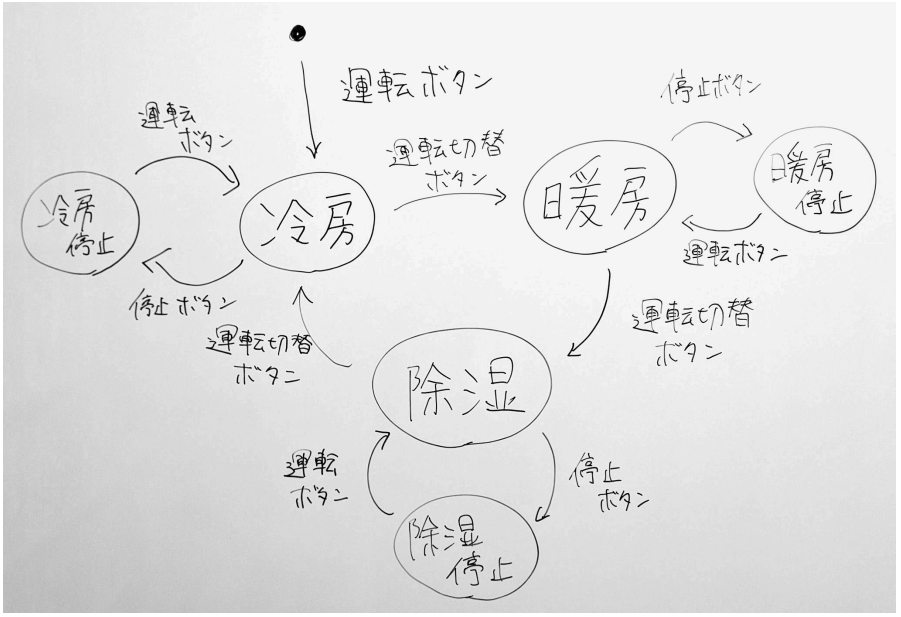
このようなエアコンがあったとき、
①N/Aも含めたすべての遷移
②1スイッチカバレッジ
をテストするために必要なテストケースを用意してみましょう

冷房運転時に運転切替ボタンを押すと暖房運転になるが、
暖房運転時に運転切替ボタンを押すと除湿運転になる、というように、
このエアコンは現在の状態とその遷移によって振る舞いが変わるようなシステムと言えるので、状態遷移テストを行う必要がありそう

このエアコンの「状態」と「イベント」には以下のようなものがある
・状態
初期状態
冷房運転
暖房運転
除湿運転
冷房停止（冷房運転から停止された状態）
暖房停止（暖房運転から停止された状態）
除湿停止（除湿運転から停止された状態）

・イベント
運転ボタン
停止ボタン
運転切替ボタン

これを踏まえてこのエアコンの状態遷移図を書くと以下のようになる



状態遷移表で、N/Aも含めたすべての遷移を洗い出すと以下のようになる

イベント＼状態	⓪初期状態	①冷房運転	②暖房運転	③除湿運転	④冷房停止	⑤暖房停止	⑥除湿停止
運転ボタン	①	N/A	N/A	N/A	①	②	③
停止ボタン	N/A	④	⑤	⑥	N/A	N/A	N/A
運転切替ボタン	N/A	②	③	①	N/A	N/A	N/A

なので、「①N/Aも含めたすべての遷移」をテストするためには、
この21個のテストケースが必要

以下のようにテストケースを表にまとめてもいい（6件だけ掲載）

No	遷移前の状態	イベント	期待する結果
1	初期状態	運転ボタンを押す	冷房運転に遷移
2	初期状態	停止ボタンを押す	遷移しない
3	初期状態	運転切替ボタンを押す	遷移しない
4	冷房運転	運転ボタンを押す	遷移しない
5	冷房運転	停止ボタンを押す	冷房停止に遷移
6	冷房運転	運転切替ボタンを押す	暖房運転に遷移

さらに、1スイッチカバレッジをテストするために、
状態遷移表を関係行列に書き直すと以下のようになる

前状態＼後状態	⓪初期状態	①冷房運転	②暖房運転	③除湿運転	④冷房停止	⑤暖房停止	⑥除湿停止
⓪初期状態		W					
①冷房運転			C		S		
②暖房運転				C		S	
③除湿運転		C					S
④冷房停止		W					
⑤暖房停止			W				
⑥除湿停止				W			

W：運転ボタンを押す
S：停止ボタンを押す
C：運転切り替えを押す

この関係行列をもとに1スイッチカバレッジを求めると以下のようになる

前状態＼後状態	⓪初期状態	①冷房運転	②暖房運転	③除湿運転	④冷房停止	⑤暖房停止	⑥除湿停止
⓪初期状態			WC		WS		
①冷房運転		SW		CC		CS	
②暖房運転		CC	SW				CS
③除湿運転			CC	SW	CS		
④冷房停止			WC		WS		
⑤暖房停止				WC		WS	
⑥除湿停止		WC					WS

なので、「②1スイッチカバレッジ」をテストするためには、
この17個のテストケースが必要

以下のようにテストケースを表にまとめてもいい（6件だけ掲載）

No	遷移前の状態	イベント1	イベント2	期待する結果
1	初期状態	運転ボタンを押す	運転切替ボタンを押す	暖房運転に遷移
2	初期状態	運転ボタンを押す	停止ボタンを押す	冷房停止に遷移
3	冷房運転	停止ボタンを押す	運転ボタンを押す	冷房運転に遷移
4	冷房運転	運転切替ボタンを押す	運転切替ボタンを押す	除湿運転に遷移
5	冷房運転	運転切替ボタンを押す	停止ボタンを押す	暖房停止に遷移
6	暖房運転	運転切替ボタンを押す	運転切替ボタンを押す	冷房運転に遷移

問4

アメリカのとある大学院における、留学生の入試制度について考えてみよう
この大学院では受験生の足切りとして、TOEFL iBTのスコアを採用している
TOEFL iBTは以下のようなテストである
・Reading、Listening、Writing、Speakingの4つのパートからなる
・各パート30点満点で、合計120点満点
・点数は0以上の整数値

この大学院の足切りを突破するための基準は以下のようになっている
・合計得点が100点以上ならば合格
・ただし、4つのパートの中で1つでも19点以下のパートがあった場合には、合計得点が100点以上でも不合格（例えば、R30, L30, W30, S15なら不合格）

このような入試制度があったときに、各パートの点数を入力にして合否を判定するシステムの動作をテストするために必要な、最低限のテストケースはどのようなものでしょうか？

ただし、各パートの点数の入力として、
0~30の整数値以外は考慮しなくてよいものとします

4つのパートを以下の省略記号で表現するものとします
Reading→R
Listening→L
Writing→W
Speaking→S

足切りを突破するためには、以下の条件を同時に満たす必要がある
・合計得点（R+L+W+S） >= 100
・R >= 20
・L >= 20
・W >= 20
・S >= 20

Rの点数によって、L+W+Sで必要な点数が変わるので、
これらの条件は相互作用を持つ　⇒　ドメイン分析テストが最適

これらの5条件をもとに、ドメインテストマトリクスを作成すると以下のようになる

変数	条件	ポイント	1	2	3	4	5	6	7	8	9	10
合計得点 R+L+W+S	合計得点 >= 100	on	100									
		off		99								
		in			105	104	105	104	105	104	110	109
Reading（R）	R >= 20	on			20							
		off				19						
		in	25	24			30	30	25	25	30	30
Listening（L）	L >= 20	on					20					
		off						19				
		in	25	25	30	30			30	30	30	30
Writing（W）	W >= 20	on							20			
		off								19		
		in	25	25	30	30	25	25			30	30
Speaking（S）	S >= 20	on									20	
		off										19
		in	25	25	25	25	30	30	30	30		
期待される結果			合格	不合格	合格	不合格	合格	不合格	合格	不合格	合格	不合格

※inポイントの値は自由に設定しましょう

このドメインテストマトリクスの10列が最低限必要なテストケースになる

問5

ある配送会社の配送料は、以下のような料金体系になっている

重量	規格内	規格外
50.0g以下	120円	200円
200.0g以下	240円	320円
500.0g以下	390円	510円
1.0kg以下	取り扱いません	710円
2.0kg以下		1,040円

サイズによって規格内と規格外に分かれており、以下の条件を満たすと規格内
・長辺25.0cm以下
・短辺15.0cm以下
・厚さ3.0cm以下
・重量500.0g以下

このような料金体系になっていたときに、配送物のサイズ（長辺・短辺・厚さ）と重量を入力にして料金を出力するシステムをテストするのに必要なテストケースはどのようなものでしょうか？

ただし、以下の条件があるとします
・「長辺 >= 短辺」が常に成り立つものとし、「長辺 < 短辺」のケースについては考慮する必要はないものとします
・重量は0.0~2,000.0gの範囲で、小数第1位までの値とします
・サイズはそれぞれ0.0~100.0cmの範囲で、小数第1位までの値とします
・重量とサイズに範囲外の数値が入力されるケースについては、考慮しなくて良いものとします
・テストケースは50個以内に収めなければいけないものとします

今回の条件は「重量」「長辺」「短辺」「厚さ」の4つ
それぞれの条件に境界値があるので、境界値テストの考え方が必要そうです

それぞれの条件の境界値は以下
「重量」：50.0g, 200.0g, 500.0g, 1.0kg（1,000.0g）
「長辺」：25.0cm
「短辺」：15.0cm
「厚さ」：3.0cm

なので境界値テストの観点で、それぞれの条件に最低限必要なテストケースは以下
・重量（g）
50.0, 50.1, 200.0, 200.1, 500.0, 500.1, 1000.0, 1000.1 の8通り
・長辺（cm）
25.0, 25,1 の2通り
・短辺（cm）
15.0, 15,1 の2通り
・厚さ（cm）
3.0, 3,1 の2通り

これらの条件の組み合わせをすべて調べようとすると、
8×2×2×2=64通りあり、テストケースが50個を超えてしまう
⇒　ペア構成テストでテストケースを削減しよう

今回は「8値条件が1つ、2値条件が3つ」なので、
サイト例①にある「L16(8^1 2^8)」の直交表を使ってみよう
一番左の8値条件と、2値条件の左3列を取り出すと以下のようになる

	A	B	C	D
1	1	1	1	1
2	1	2	2	2
3	2	1	1	1
4	2	2	2	2
5	3	1	1	2
6	3	2	2	1
7	4	1	1	2
8	4	2	2	1
9	5	1	2	1
10	5	2	1	2
11	6	1	2	1
12	6	2	1	2
13	7	1	2	2
14	7	2	1	1
15	8	1	2	2
16	8	2	1	1

ここにそれぞれの条件と、それぞれの条件が取る値を入れた上で、
一番右の列に配送料を追加すると、直交表は以下のようになる

	重量	長辺	短辺	厚さ	配送料
1	50.0	25.0	15.0	3.0	120円
2	50.0	25.1	15.1	3.1	200円
3	50.1	25.0	15.0	3.0	240円
4	50.1	25.1	15.1	3.1	320円
5	200.0	25.0	15.0	3.1	320円
6	200.0	25.1	15.1	3.0	320円
7	200.1	25.0	15.0	3.1	510円
8	200.1	25.1	15.1	3.0	510円
9	500.0	25.0	15.1	3.0	510円
10	500.0	25.1	15.0	3.1	510円
11	500.1	25.0	15.1	3.0	710円
12	500.1	25.1	15.0	3.1	710円
13	1000.0	25.0	15.1	3.1	710円
14	1000.0	25.1	15.0	3.0	710円
15	1000.1	25.0	15.1	3.1	1,040円
16	1000.1	25.1	15.0	3.0	1,040円

この直交表の行がそのままテストケースになる

直交表だけだと規格内の**200.0g**の境界値がテストされていないので、
以下のテストケースを追加するとさらに安心

17	200.0	25.0	15.0	3.0	240円
18	200.1	25.0	15.0	3.0	390円

必要に応じてピンポイントの観点でテストケースを補いましょう！