testerzy.pl

Testy jednostkowe
Wybrane problemy testowania metod rekurencyjnych

Artykuł przeznaczony jest dla osób związanych z testowaniem, programowaniem, jakością oraz wytwarzaniem oprogramowania, wymaga jednak znajomości podstaw programowania obiektowego.

Testy jednostkowe przeprowadzane są na poziomie kodu podczas procesu wytwarzania oprogramowania. Na ich temat można znaleźć wiele publikacji, książek i artykułów naukowych. Również w Internecie jest sporo wiadomości na ten temat. Nieczęsto jednak znajdziemy informacje o testowaniu metod wywoływanych rekurencyjnie. Okazuje się, że testy jednostkowe dla takich metod nie są proste i dodatkowo wymagają twórczego podejścia od autora.

1. Zjawisko rekurencji w naukach ścisłych

Definicja 1.1 [1] Rekurencja jest techniką programowania, dzięki której procedura, funkcja lub podprogram jest w stanie w swoim ciele wywołać sam siebie. Dzięki rekurencji łatwo można „wykonać" zadanie, w którym potrzebne są wyniki cząstkowe do obliczenia całości. Rekurencja składa się z podania **wartości brzegowych** (początkowych) i z równania wyrażającego **ogólną wartość** za pomocą wartości wcześniejszych wyrazów. Klasycznym przykładem w zagadnieniu rekurencji jest liczenie silni dla dowolnej liczby naturalnej n, czyli $(n!)$:

Przykład 1.1

$$n! = \begin{cases} 1 & , n = 0 \\ n(n-1)! & , n \geq 1 \end{cases}.$$

Rekurencja może być z dobrym skutkiem stosowana w najefektywniejszych algorytmach sortowania. Rekurencyjny opis obliczeń (funkcji czy metod) jest na ogół bardziej zwarty (krótszy, przejrzysty) niż opis tych samych obliczeń bez użycia rekurencji. Taki opis jest stosowany np. przy opisie fraktali, które są definiowane, jako obiekty podobne do swoich części (obiekty samopodobne). Zwartość opisu rekurencyjnego nie zawsze idzie w parze z efektywnością realizacji algorytmów. Dlatego algorytmy rekurencyjne powinny być stosowane rozsądnie. Istnieje także pewna klasa funkcji rekurencyjnych, które można przedstawić jawnym wzorem typu $f(n)$, gdzie $n \in N$. Nie zawsze jednak jest to możliwe.

2. Sformułowanie problemu

Załóżmy, że w kodzie pewnego oprogramowania istnieją metody zaimplementowane rekurencyjnie. Do tych metod należy napisać testy jednostkowe. Wobec tego potrzebne są sposoby, dzięki którym będzie można otrzymać dane oczekiwane w celu sprawdzenia poprawności rezultatu testów jednostkowych. W takiej sytuacji z pewnością można napotkać następujące problemy:

a) metoda jest zaimplementowana, jako funkcja jednej lub wielu zmiennych liczb naturalnych,

b) metoda nie jest funkcją, która da się opisać wyrażeniem matematycznym,

W artykule skupimy się na metodzie matematycznej. Załóżmy, że pewna firma handlowa potrzebuje aplikację ($C\#$), która ułatwi zarządzanie wynagrodzeniami dla pracowników (w tym również systemem wypłaty premii). Należy zaimplementować metodę, która ma liczyć premię od ilości sprzedanych napojów, które są pakowane w pakiety. Premia może być wypłacona w dowolnym czasie (ale od całych pakietów z napojami), o czym decyduje osoba, której ta premia się należy. Kolejna premia jest pomniejszana o kwotę wypłaconą wcześniej.

Zarząd firmy chce, aby:

i) premia od sprzedaży uzależniona była od ilości poprzednio sprzedanych pakietów,

ii) do bieżącej sprzedaży dodawane było wyrażanie $m-1$, gdzie m – ilość wszystkich sprzedanych pakietów.

Wobec tego, równanie przedstawiające kwotę premii będzie miało postać:

$$f(m) = \begin{cases} 0 & , m = 1 \\ f(m-1) + m - 1, & m > 1 \end{cases} \quad (1).$$

Równania rekurencyjne o postaci (1) jak wyżej zazwyczaj dają się rozwiązać, a dzięki temu można otrzymać jawny wzór $f(m)$. Wobec tego niech $m \geq 0$. Rozpisując lewą stronę równania otrzymujemy wyrażenie:

$$f(m) = f(m-1) + m - 1 = f(m-2) + m - 1 - 1 + m - 1 = f(m-2) + (m-2) + (m-1) =$$
$$= f(m-3) + (m-3) + (m-2) + (m-1) = (*) \ (2)$$

Podczas kolejnych cofnięć do poprzednich argumentów funkcji f dochodzimy do kroku:

$m - k = 1$, czyli $m = k + 1$.

$$(*) = f(m-k) + km - \sum_{i=1}^{m} i = (m-1)m - \frac{m(m-1)}{2} = m^2 - m - \frac{m^2 - m}{2} = \frac{m^2 - m}{2}. \quad (3)$$

Dla zachowania poprawności postępowania wykażemy indukcyjnie, że rozwiązanie $\frac{m^2-m}{2}$ jest poprawne.

1) Dla $m = 1$ mamy $f(m) = 0$.
2) Załóżmy, że równanie jest poprawnie rozwiązane dla każdego $m \in N$. Wykazać należy, że dla każdego $(m+1) \in N$. Wobec tego:

$$f(m+1) = \frac{(m+1)^2 - (m+1)}{2} = \frac{m^2 + 2m + 1 - m - 1}{2} = \frac{m^2 - m}{2} + m.$$

Z drugiej strony wstawiając do postaci rekurencyjnej m+1 za m mamy

$$f(m+1-1) + m + 1 - 1 = f(m) - m,$$

co należało wykazać.

Kod funkcji $f(m)$ umieszczony w pewnej klasie w języku $C\#$ ma postać:

```
public class Funkcje
{
        public virtual int Premia(int m)
        {
                if (m == 1) return 0;
                else return Premia(m - 1) + m + 1;
        }
}

// pochodna klasy Funkcje utworzona w celu zaprojektowania testów

public class TesterFuncji : Funkcje
{
    public int liczbaWywolan { get; set; }

    public override int Premia(int m)
    {
        liczbaWywolan ++;
        return base.Premia(m);
    }

        // metoda pomocnicza z rozwiąnym wzorem

        public int WzorFunkcji(int m)
        {
                return (m*m-m)/2;
        }
}

// klasa testowa
```

```
public class TestyJednostkowe
{
	// test sprawdza, czy wartości zwrócone rekurencyjnie są równe z
	// oczekiwanym rezultatem, czyli wzorem danym jawnie
	[Test]
	Public void CzySaRowne
	{
		var tester = new TesterFunkcji();
		for (int j = 1; j <= 100; j++)
			Assert.AreEqual(tester.WzorFunkcji(j),tester.Premia(j));
	}
	// test sprawdza, czy metoda się wywołuje co najmniej raz
	[Test]
	Public void CzyWywolujeSieRaz
	{
		var tester = new TesterFunkcji();
		tester.Premia(1);
		Assert.GreaterOrEqual(1, tester.liczbaWywolan);
	}

}
```

Wyżej przedstawiona metoda Premia w dość prosty sposób pozwoliła na zaimplementowanie testów niej samej. Istnieje również tzw. metoda szeregów funkcyjnych, która ma zastosowanie w rozwiązywaniu funkcji zdefiniowanych rekurencyjnie, np. takich jak ciąg Fibonacciego, ale nie należy do najprostszych sposobów rozwiązywania równań.

Nie zawsze jednak musi tak być, że metoda rekurencyjna ma parametr, który jest liczbą naturalną. Nie zawsze też można taką metodę rozwiązań algebraicznie. W takim przypadku podejście do testów musi być dostosowane do metody, którą chcemy sprawdzić.

3. W czym jest dokładnie problem z rekurencją?

Cały proces wywołania rekurencji musi pamiętać wszystkie wartości zmiennych na każdym poziomie wywołania. 1000 - razy wykonana rekurencja oznacza 1000 - krotne zapamiętanie wszystkich zmiennych, które są używane w metodzie, a to może powodować problem z wydajnością. Dokładna implementacja i rozmieszczenie pamięci do obliczeń rekurencyjnych są zależne od języka programowania, ale zazwyczaj dzieje się to w stosie, który przetrzymuje wartości funkcji. Następnie stos jest czyszczony od góry, co daje możliwość wykonywania obliczeń w odwrotnej kolejności. Należy jednak pamiętać, że pamięć na stosie jest ograniczona. Istnieją pewne metody optymalizacji wywołań rekurencyjnych np. optymalizacja ogonowa, która polega na usuwaniu operacji wracających z powrotem. Taka optymalizacja jest przydatna, gdy pojedyncze sekwencje wywołań metody są bardzo długie.

Języki programowania (np. $C\#$) często zawierają instrukcje, które umożliwiają optymalizację rekurencji. Jednak po zastosowaniu tych instrukcji nie ma pewności, że wszystko zadziała poprawnie w każdym przypadku (środowisko uruchomieniowe $C\#$ ma instrukcje optymalizacji rekurencji, ale kompilator ich nie zweryfikuje [3]), dlatego dobre testy jednostkowe spełnią swoją rolę.

Wnioski:

i) testy jednostkowe dla metod rekurencyjnych powinny sprawdzać nie tylko wartości wyjściowe, ale również aspekty techniczne wywołania tych metod, np. wydajność, testy pamięci zależne od platformy,
ii) osoba pisząca testy powinna wiedzieć, jak działa język programowania w przypadku implementacji metod rekurencyjnych, co jest w stanie sprawdzić kompilator, a czego nie sprawdza, jak po implementacji takich metod podejść do testowania,
iii) testy jednostkowe dla metod rekurencyjnych powinny pisać osoby z dobrą techniczną znajomością języka programowania oraz wywołań rekurencyjnych.

4. Referencje

[1]https://www.bryk.pl/wypracowania/pozosta%C5%82e/informatyka/15879rekurencjaiiteracjar%C3%B3%C5%BCniceipodobie%C5%84stwa.html

[2]http://stackoverflow.com/questions/2241759/how-to-write-a-mockist-test-of-a-recursive-method

[3]https://msdn.microsoft.com/pl-pl/library/optymalizacja-kodu-c-sharp--czesc-3.aspx

Marek Żukowicz jest absolwentem matematyki na Uniwersytecie Rzeszowskim. Obecnie pracuje jako tester. Jego zainteresowania skupiają się wokół testowania, matematyki, zastosowania algorytmów ewolucyjnych oraz zastosowania matematyki w procesie testowania. Interesuje się również muzyką, grą na akordeonie oraz perkusji.