# IEnumerable i yield

25 XI 2016

## 1 Zadanie

Zadanie polega na napisaniu funkcji operujących na potencjalnie nieskończonych ciągach liczb całkowitych reprezentowanych przez `IEnumerable`. W trakcie tego zadania zakładamy, że ciągi podane jako argumenty funkcji zawierają tylko elementy typu `int` – można wykonywać rzutowanie z `object` na `int` bez upewniania się co do zgodności typów.

Część operacji, które należy zaimplementować w ramach zadania została już zaimplementowana w bibliotece LINQ, w szczególności w klasie `Enumerable`. W tym zadaniu **nie wolno** używać elementów LINQ, całość należy zaimplementować tylko z użyciem pętli.

Zadane podzielone jest na 5 etapów. W pierwszej kolejności należy wykonać pierwszy etap, ponieważ jego elementy są używane do testowania dalszych etapów. Kolejne etapy zostały uszeregowane według przewidywanej trudności, ale można je wykonywać w dowolnej kolejności. Każdy etap jest za 1 punkt.

Wraz z wykonaniem każdego z etapów należy odkomentować odpowiedni fragment `Main`. Do weryfikacji wyników użyć dostarczonego pliku z przykładowym wyjściem programu.

> Wszystkie metody implementowane w trakcie tego zadania mają być statycznymi metodami statycznej klasy `Lab8a.Sequences`. Należy utworzyć tę klasę w oddzielnym pliku.

### 1.1 Etap 1

- Napisać metodę `PrintN`, która przyjmuje dwa argumenty: ciąg i liczbę całkowitą $n$. Metoda wypisze na standardowe wyjście $n$ pierwszych elementów podanego ciągu (lub mniej, jeśli ciąg jest krótszy).

- Napisać metodę `ArithmeticSequence`, która przyjmuje dwie liczby całkowite jako argumenty: $x_1$ i $d$ i zwraca nieskończony ciąg arytmetyczny, które pierwszym wyrazem jest $x_1$, a kolejne wyrazy różnią się o $d$ od siebie.

- Napisać metodę `SubtractX`, która przyjmuje dwa argumenty: ciąg $\{a_n\}$ i liczbę całkowitą $x$ i zwraca ciąg, którego każdy wyraz jest wyrazem ciągu wejściowego pomniejszonym o $x$, czyli ciąg $\{b_i\}$ dany wzorem: $b_i = a_i - x$.

## 1.2 Etap 2

- Napisać metodę `RepeatNTimes`, która przyjmuje dwa argumenty: ciąg $\{a_n\}$ i liczbę całkowitą $n$. Metoda ta zwraca ciąg $\{a_n\}$ powtórzony $n$-krotnie. Na przykład dla $n = 3$ i ciągu o długości $m$, wynikiem będzie ciąg: $a_1, a_2, \ldots, a_m, a_1, a_2, \ldots, a_m, a_1, a_2, \ldots, a_m$. *Uwaga: W przypadku, gdy ciąg jest nieskończony wynikowy ciąg będzie tożsamy z wejściowym.*

- Napisać metodę `LimitSeq`, która przyjmuje dwa argumenty: ciąg i liczbę całkowitą $n$ i zwraca ciąg, który składa się z pierwszych $n$ wyrazów ciągu wejściowego (lub krótszy), jeśli ciąg wejściowy jest krótszy niż $n$.

- Napisać metodę `LastOrDefault`, która przyjmuje dwa argumenty: ciąg i liczbę $a$. Metoda zwraca wartość ostatniego elementu ciągu lub $a$, gdy ciąg jest długości 0. *Uwaga: ta metoda ma działać tylko dla ciągów skończonych.*

## 1.3 Etap 3

- Napisać metodę `InnerProduct`, która przyjmuje dwa argumenty: dwa ciągi $\{a_n\}$ i $\{b_n\}$. Metoda ta zwraca iloczyn skalarny dwóch ciągów:

  $$\sum_{i=1,\ldots,k} a_i \cdot b_i$$

  , gdzie $k$ jest długością krótszego z ciągów. Zakładamy, że $\sum_{i\in\emptyset} x_i = 0$. *Metoda ma działać, gdy jeden (dowolny) z ciągów jest skończonej długości.*

- Napisać metodę `NotDivigingNeighboursSequence`, która przyjmuje ciąg $\{a_n\}$ i zwraca podciąg $\{b_n\}$ tego ciągu spełniający następujące własności:

  - Jeśli istnieje $a_1$, to $b_1 = a_1$.
  - Żaden element ciągu wyjściowego nie jest dzielnikiem swojego sąsiada: $\forall i(\neg b_i | b_{i+1} \wedge \neg b_{i+1} | b_i)$.
  - Ciąg $\{b_n\}$ budowany jest w sposób zachłanny: dla każdy kolejny element ciągu $\{a_i\}$ jest dodawany do ciągu wynikowego, jeśli tylko nie narusza warunków.

  *Metoda ma działać tylko dla ciągów o elementach dodatnich.*

## 1.4 Etap 4

- Napisać metodę `Interleave`, która przyjmuje dwa ciągi $\{a_n\}$ i $\{b_n\}$ jako argumenty i zwraca ciąg zawierający naprzemiennie elementy ciągów wejściowych: $a_1, b_1, a_2, b_2, \ldots$. Gdy jeden z ciągów jest krótszy, pozostałem elementy drugiego ciągu są dopisywane na koniec ciągu wynikowego.

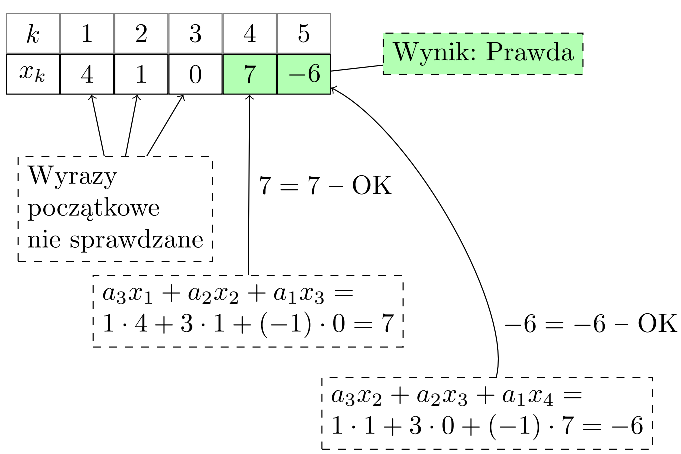
## 1.5 Etap 5

- Napisać metodę `IsRecurrenceEquation`, która przyjmuje dwa argumenty – ciąg $\{x_k\}$ i tablicę typu int i zwraca bool. Metoda sprawdza, czy ciąg jest liniowym ciągiem rekurencyjnym ze współczynnikami zdefiniowanymi w tablicy ($k$-ty element ciągu jest kombinacją liniową poprzednich $n$ elementów). Oznaczmy elementy tablicy współczynników: $a_1, \ldots, a_n$, gdzie $n$ – długość tablicy. Ciąg ma spełniać następujący warunek:

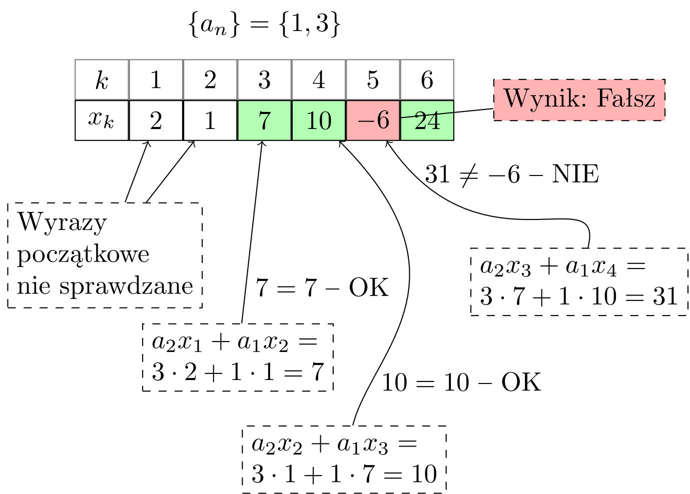
$$(\forall k > n) x_k = \sum_{i=1}^{n} a_i \cdot x_{k-i}$$

  Pierwsze $n$ wyrazów może być dowolne. Rysunki 1 i 2 przedstawiają przykładowe ciągi i odpowiedzi.

$\{a_n\} = \{-1, 3, 1\}$




Rysunek 1: Przykład obliczenia Etapu 5




Rysunek 2: Przykład obliczenia Etapu 5