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 `Lab8b.Sequences`. Należy utworzyć tę klasę w oddzielnym pliku.

1.1 Etap 1

- Napisać metodę `PrintSeq`, która przyjmuje jeden argument – ciąg i wypisuje na standardowe wyjście wszystkie jego wyrazy.
- Napisać metodę `LimitSequence`, która przyjmuje dwa argumenty: ciąg i liczbę całkowitą n i zwraca ciąg składający się z n pierwszych wyrazów ciągu wejściowego lub mniej, jeśli ciąg wejściowy jest krótszy.
- Napisać bezparametrową metodę `NaturalNumbers`, która zwraca nieskończony ciąg kolejnych liczb naturalnych (rozpoczynając od 0).
- Napisać metodę `Intersperse`, która przyjmuje ciąg $\{a_n\}$ i liczbę całkowitą x. I zwraca ciąg, zawierający naprzemiennie kolejny wyraz a_i i x.

Dokładniej: zwraca ciąg o długości $2m$, gdzie m jest długością ciągu wejściowego taki, że:

$$b_i = \begin{cases} a_{(i+1)/2} & \text{, jeśli } i \text{ jest nieparzyste} \\ x & \text{, jeślijj } i \text{ jest parzyste} \end{cases}$$

. Jeśli ciąg wejściowy jest nieskończony, zwraca nieskończony ciąg o takiej samej własności.

1.2 Etap 2

- Napisać metodę `Cycle`, która jeden argument – ciąg. Metoda zwraca nieskończony ciąg powstały poprzez powtarzanie ciągu wejściowego $\{a_i\}$: $a_1, a_2, \ldots, a_n, a_1, a_2, \ldots, a_n, a_1, \ldots$. *Uwaga: dla nieskończonego ciągu wejściowego, ciąg wyjściowy będzie tożsamy z wejściowym.*

- Napisać metodę `IndexOfFirstOccurrence`, która przyjmuje dwa argumenty – ciąg $\{a_n\}$ i liczbę całkowitą x i zwraca liczbę całkowitą – indeks pierwszego wystąpienia x w ciągu. Jeśli x nie występuje w skończonym ciągu, zwraca -1.

- Napisać metodę `SkipN`, która przyjmuje dwa argumenty: ciąg i liczbę całkowitą n. Metoda zwraca ciąg uzyskany poprzez pominięcie pierwszych n wyrazów ciągu wejściowego. Jeśli ciąg wejściowy ma nie więcej niż n wyrazów – zwraca ciąg pusty.

1.3 Etap 3

- Napisać metodę `SequenceSum`, która przyjmuje dwa ciągi $\{a_n\}$ i $\{b_n\}$ jako argumenty i zwraca ciąg, którego wyrazy są sumą odpowiadających elementów ciągów wejściowych.

  $$c_i = a_i + b_i$$

  Ciąg wyjściowy ma długość krótszego (nie dłuższego) z ciągów wejściowych. Jeśli oba ciągi wejściowe są nieskończone, ciąg wyjściowy też jest nieskończony.

- Napisać metodę `ArithmeticSubsequence`, która przyjmuje jeden argument – ciąg i zwraca ciąg, który spełnia następujące warunki:
  - Jest podciągiem ciągu wejściowego.
  - Jest ciągiem arytmetycznym
  - Pierwszy jego element to a_1, drugi to a_2, jeśli te elementy istnieją w ciągu wejściowym.
  - Jest zbudowany zachłannie poprzez dodawanie do ciągu wejściowego tych elementów ciągu wyjściowego, które spełniają warunek ciągu arytmetycznego zdefiniowanego przez poprzednie elementy.

1.4 Etap 4

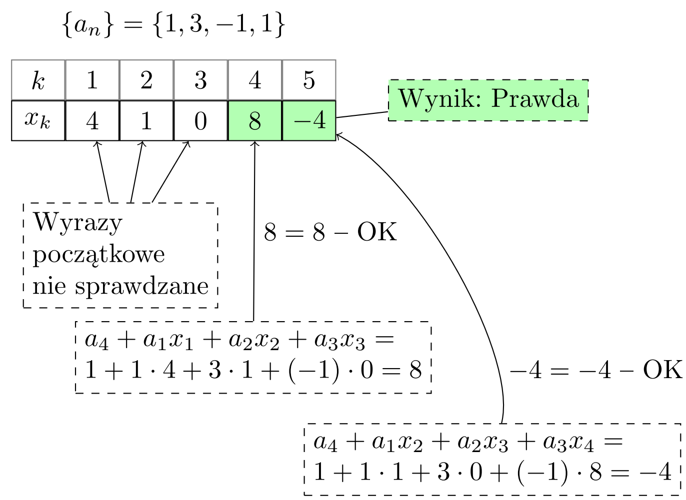
- Napisać metodę `SequenceSumWithTail`, która zwraca ciąg podobny do metody `SequenceSum`, ale w przypadku, gdy ciągi są różnej długości, dopisuje na koniec ciągu wynikowego pozostałem elementy dłuższego z ciągów.

1.5 Etap 5

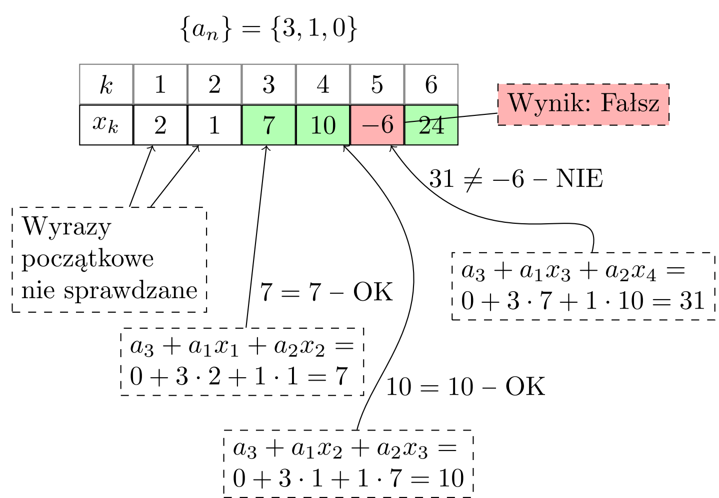
- Napisać metodę `IsRecurrenceEquation`, która przyjmuje dwa argumenty – ciąg $\{x_k\}$ i tablicę typu int i zwraca bool. Metoda sprawdza, czy ciąg jest ciągiem rekurencyjnym ze współczynnikami zdefiniowanymi w tablicy. k-ty element ciągu jest kombinacją afiniczną $n - 1$ poprzednich elementów. Oznaczmy elementy tablicy współczynników: $a_1, \ldots, a_n$, gdzie $n \geqslant 1$ – długość tablicy. Ciąg ma spełniać następujący warunek:

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

  Pierwsze $n - 1$ elementów ciągu może być dowolne. Rysunki 1 i 2 przedstawiają przykładowe ciągi i odpowiedzi.



Rysunek 1: Przykład obliczenia Etapu 5



Rysunek 2: Przykład obliczenia Etapu 5