Appello TAP del 8/09/2017

Scrivere nome, cognome e matricola sul foglio protocollo, indicando anche se avete nel piano di studi TAP da 6 CFU (quello attuale) o da 8 CFU (quello "vecchio"). Avete a disposizione due ore.

Esercizio 1 (10 punti)

Scrivere l'extension-method `TakePrime` che estrae sottosequenze di elementi di tipo `T`. Il metodo dovrà prendere come parametro "this" `s`, la sequenza sorgente, e un intero `count` che determina la dimensione massima della sequenza restituita. Nota: la sequenza di input può anche essere infinita. Il metodo `TakePrime` invocato su `s` e un intero `count`:

- restituisce la sequenza contenente i primi `count` elementi di `s` (nello stesso ordine) la cui posizione è un numero primo, se tali elementi esistono;
- se in `s` non ci sono abbastanza elementi, restituisce tutti gli elementi della sequenza sorgente la cui posizione è un numero primo.

Per esempio, il seguente frammento di codice

```
var seq = new string[] { "a 0", "b 1", "c 2", "d 3", "e 4", "f 5", "g 6" };
Console.Write("First 2 elements on some prime position are: ");
foreach (var s in seq.TakePrime(2))
     Console.Write("{0} ,", s);
Console.WriteLine();
Console.Write("First 100 elements on some prime position are: ");
foreach (var s in seq.TakePrime(100))
      Console.Write("{0} ,", s);
Console.WriteLine();
```

stampa:

```
First 2 elements on some prime position are: c 2, d 3,
First 100 elements on some prime position are: c 2, d 3, f 5,
```

Il metodo deve sollevare l'eccezione...

- `ArgumentNullException` se `s` è `null`;
- `ArgumentOutOfRangeException` se `count` non è strettamente positivo

Esercizio 2 ([3+3+4] = 10 punti)

Implementare, usando NUnit ed eventualmente Moq, i seguenti test relativi al metodo `TakePrime`, dell'esercizio precedente.

1. Input della chiamata sotto test: `s` deve essere la sequenza dei primi 20 interi a partire da 0 e `count` deve essere 6.

   Output atteso: la sequenza $2, 3, 5, 7, 11, 13$.

2. Test parametrico con parametro intero `b`.

   Input della chiamata sotto test: `s` deve essere la sequenza (illimitata) delle potenze di `b` e `count` deve essere 0.

   Output atteso: deve essere sollevata un'eccezione di tipo `ArgumentOutOfRangeException`.

3. Test parametrico con parametro intero `size`.

   Input della chiamata sotto test: `s` deve essere la sequenza (illimitata) dei numeri interi a partire da 0 e `count` deve essere `size`.

   Il test deve verificare che il risultato sia una sequenza strettamente crescente di lunghezza `size`.

Esercizio 3 (10 punti)

- Si definisca un'interfaccia generica per rappresentare le relazioni uno a molti fra elementi di un generico tipo `T1` (lato 1) e di un generico tipo `T2` (lato molti).

  Le operazioni richieste sono le seguenti.

  - Aggiungere un link alla relazione, con tipo di ritorno booleano;
  - Togliere un link alla relazione, con tipo di ritorno booleano;
  - Restituire il dominio della relazione, ovvero la collezione di elementi di tipo `T1` per cui esiste almeno un link nella relazione;
  - Restituire l'immagine della relazione, ovvero la collezione di elementi di tipo `T2` per cui esiste almeno un link nella relazione;
  - Restituire la collezione di elementi di tipo `T2` che sono in relazione con un elemento di tipo `T1` dato come parametro;
  - Comporre con una relazione uno a molti da `T2` a un ulteriore tipo generico `T` e restituire la relazione uno a molti da `T1` a `T`;

  Uno o più dei tipi generici utilizzati si possono dichiarare come co/controvarianti? perché?

- Implementare l'interfaccia definita, in maniera tale che

  - gli oggetti della classe rispettino il vincolo *uno a molti*, ovvero non ci possano essere simultaneamente due link `(x1,y)` e `(x2,y)` con `x1!=x2`.
  - per quanto riguarda le operazioni di inserimento e cancellazione di un link, il valore restituito sia `true` se e solo se la relazione è stata effettivamente modificata;
  - tutte le collezioni restituite dalle varie operazioni non contengano duplicati.