4.5 Collatz algoritmen, rekursiv

Lothar Collatz (1910-1990) var professor för tillämpad matematik vid Hamburgs Universitet på 60-talet. Som ung student ställde han upp följande uppgift:

> *”Tänk dig ett positivt heltal (startvärde). Är talet udda multiplicera det med 3 och addera 1. Är talet jämnt dividera det med 2. Gör samma sak med resultatet. Fortsätt tills du fått 1.”*

Det visar sig att talföljderna i denna algoritm alltid slutar med 1 oavsett startvärde. Dock är detta påstående matematiskt hittills obevisat *.

Tidigare har Collatz algoritmen implementerats *iterativt* dvs med en loop, både i modulär och icke-modulär version. Den modulariserade varianten har dessutom behandlats både som **void**-metod och formulerats som metod med returvärde i en övningsuppgift. Följande klass definierar nu algoritmen som en *rekursiv* metod:

```
// Collatz_reku.cs
// Definierar metoden Collatz() rekursivt.
// Får ett startvärde som parameter, tar det gånger 3 + 1,
// om det är udda, annars delar det med 2.
// Avslutar när talföljden har nått 1.
using System;
class Collatz_reku
{
   public static int Collatz(int n)      // Rekursiv metod
   {
      Console.Write(n + "\t");
      if (n == 1)
         return n;
      else
      {
         if (n % 2 == 1)
            return Collatz(3 * n + 1);  // Rekursivt anrop
         else
            return Collatz(n / 2);      // Rekursivt anrop
      }
   }
}
```

* Man kan testa Collatz algoritmen i appen *Mattekollen* där den är kodad i Python. Ladda ned appen eller kör den som Webbapp: **app.mattekollen.se → En mobil pythonmiljö**. Eller kör den direkt som webbapp: beta.mattekollen.se/#/app/coding. Kör koden med olika startvärden för att kolla om algoritmens talföljder alltid slutar med 1. I denna variant är Collatz algoritmen förstås inte rekursiv utan iterativ, närmare bestämt kodad med en **while**-loop. Du kan nu gärna försöka att koda en rekursiv version i Python i denna miljö.

Metoden **`Collatz()`** som är definierad i klassen **`Collatz_reku`** ovan, är rekursiv därför att den anropar sig själv i sin egen definition. De rekursiva anropen står i den nästlade **`if-else`**-satsens **`return`**-satser. I själva verket skickas i dessa anrop de nya elementen i algoritmens talsekvens till metoden igen, efter att de bestämts enligt algoritmens regler. På så sätt simulerar de rekursiva anropen loopen som fanns i de tidigare iterativa implementationerna.

Vi testar den rekursiva metoden **`Collatz()`** ovan i följande klass:

```
// Collatz_reku_Test.cs
// Läser in ett startvärde och skickar det till den rekursiva
// metoden Collatz() definierad i klassen Collatz_reku.
using System;

class Collatz_reku_Test
{
   static void Main()
   {
      Console.Write("\n\tMata in ett positivt heltal:\t");
      int number = int.Parse(Console.ReadLine());  // Start-
      Console.Write("\n\t");                       // värde

      Collatz_reku.Collatz(number);                // Anrop

      Console.WriteLine("\n");
   }
}
```

Körexemplet nedan skiljer sig förstås inte från de tidigare testerna med de iterativa lösningarna:

```
        Mata in ett positivt heltal:    135

135     406     203     610     305     916     458     229
688     344     172     86      43      130     65      196

98      49      148     74      37      112     56      28
14      7       22      11      34      17      52      26
13      40      20      10      5       16      8       4
2       1
```