Semesterarbeit Analyis mit Python Teil 1

Einleitung: Rekursion mit Python anhand der Fibonacci-Folge

Zu ehren Leonardo Fibonaccis welcher im Jahr 1202 wohl kaum dachte welche Signifikants seine Beschreibung zum Wachstum einer Kanichenpopulation erreichen würde. Die Fibonacci-Folge wurde von ihm wie folgt definiert:

$f_0 := 0$
$f_1 := 1$
$f_n := f_{n-1} + f_{n-2}$ für $n \geq 2$

Die ersten Fibonacci-Zahlen sind folglich 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55,...

Die Aufgabenstellung gliedert sich in fünf Teile:

1. Implementieren Sie eine Python-Funktion fib(n) , die die n-te Fibonacci-Zahl bestimmt.
2. Eine naive Implementierung setzt die obige Rekursionsgleichung direkt um. Schreiben Sie eine weitere Python-Funktion, die berechnet, wie viele Funktionsaufrufe on fib notwendig sind, um die n-te Fibonacci-Zahl zu berechnen.
3. Vergleichen Sie die Anzahl der Funktionsaufrufe von fib zur Bestimmung einer Fibonacci-Zahl mit den Fibonacci-Zahlen selber. Können Sie eine Vermutung aufstellen?
4. Verwenden Sie die Funktion time() aus dem Modul time, um zu bestimmen, wie lange die Funktion fib benötigt, um eine Fibonacci-Zahl zu bestimmen.
5. Implementieren Sie eine weitere Python-Funktion zur Berechnung der n-ten Fibonacci-Zahl, die möglichst effizient ist. (Hinweis: das kann rekursiv oder iterativ gelöst werden.)

Theoretische Beschreibung des Lösungsansatzes

Die erste Aufgabe ist gelöst in dem die Definition der Fibonacci-Folge sozusagen in die Phytonsprache übersetzt wird. Es werden drei verscheidene Fälle abgefangen in der Funktion fib() für f_0, f_1 und f_n. Die Fälle haben dann den Rückgabewert gemäss Definition.

Für die zweite Aufgabe fügen wir obiger fib() Funktion noch einen Zähler hinzu welcher bei jedem Aufruf erhöht wird.

In Aufgabe drei um die Anzahl Funktionsaufrufe mit den Fibonacci-Zahlen selber zu untersuchen geben wir mit einer for-Schleife einemal die Position, Fibonacci-Zahl und Anzahl Funktionsaufrufe der ersten 20 Zahlen aus. Somit können die Zahlen beurteilt werden. Danach lässt sich eventuell ein Folge finden für die Anzahl Funktionsaufrufe. Diese wäre mit dem Induktionsalgorithmus auf Richtigkeit zu prüfen.

Die Aufgabe 4. und 5. wird zusammen gefasst. Die Funktion fib() wird sauber neu aufgebaut, wie ich das professionel tun würde. Dazu erstellen wir eine Klasse in welcher wir die die Fibonacci-Zahlen iterativ und rekursiv berechnen und dabei die Zeit messen. Danach wollen wir die beiden Funktionen vergelichen und die schnellere bestimmen.

Die iterative Funktion lässt sich mit einer Schleife realisieren welche so oft die beiden Vorgänger zusammen zählt und die $f_{n-1} + f_{n-2}$, bis die gewünschte n-te Zahl erreicht ist.

Implementierungsidee und Programm Code

1. n-te Fibonacci-Zahl bestimmen

Die Funktion fib() fängt die beiden Startwerte a_0 und a_1 mit einem "if" und "elif" ab. Als letzte Option geht die Funktions ins "else" und ruft sich selbst auf. Die Funktion ruft sich solange selbst auf bis alle Selbstaufrufe bei 0 und 1 enden.

In [1]:
```
#set the n-the fibonaccinumber you would like to get.
print("\n")
print("Please set the n-the Fibonacci number you would like to get!")
n = int(input())

def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-1) + fib(n-2)

print("\n")
print(f"At position n={n} is Fibonacci number: {fib(n)}")
```

```
Please set the n-the Fibonacci number you would like to get!
3


At position n=3 is Fibonacci number: 2
```

2. fib() Aufrufe

Zur simplen Implementierung fügen wier einen Zähler hinzu welcher bei jedem Funktionsaufruf um eins erhöht wird. Damit wissen wir wie oft fib() aufgerufen wird. Dazu wird hier eine Variabel "count" eingeführt.

In [2]:
```
#set the n-the fibonaccinumber you would like to get.
print("\n")
print("Please set the n-the Fibonacci number you would like to get!")
n = int(input())

#count of function calls
count = 0

def fib(n):
    global count
    count += 1

    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-1) + fib(n-2)

print("\n")
print(f"At position n={n} is Fibonacci number: {fib(n)}")
print(f"It took {count} calls of fib()")
```

```
Please set the n-the Fibonacci number you would like to get!
4


At position n=4 is Fibonacci number: 3
It took 9 calls of fib()
```

3. Anzahl Funktionsaufrufe

Die for-Schleife plotet die ersten 20 Fibonacci-Zahlen. Der Zähler muss vor jedem Funktionsaufruf wieder zurück gesetzt werden. So lassen sich nun die Zahlen untersuchen

In [3]:
```
#Funktion welche die ersten 20 Fibonacci Zahlen plotet

for i in range(20):
    count = 0
    print(f"Position n={i}, Fibonacci number: {fib(i)}, calls {count}")
```

```
Position n=0, Fibonacci number: 0, calls 1
Position n=1, Fibonacci number: 1, calls 1
Position n=2, Fibonacci number: 1, calls 3
Position n=3, Fibonacci number: 2, calls 5
Position n=4, Fibonacci number: 3, calls 9
Position n=5, Fibonacci number: 5, calls 15
Position n=6, Fibonacci number: 8, calls 25
Position n=7, Fibonacci number: 13, calls 41
Position n=8, Fibonacci number: 21, calls 67
Position n=9, Fibonacci number: 34, calls 109
Position n=10, Fibonacci number: 55, calls 177
Position n=11, Fibonacci number: 89, calls 287
Position n=12, Fibonacci number: 144, calls 465
Position n=13, Fibonacci number: 233, calls 753
Position n=14, Fibonacci number: 377, calls 1219
Position n=15, Fibonacci number: 610, calls 1973
Position n=16, Fibonacci number: 987, calls 3193
Position n=17, Fibonacci number: 1597, calls 5167
Position n=18, Fibonacci number: 2584, calls 8361
Position n=19, Fibonacci number: 4181, calls 13529
```

4. und 5. Zeit und Effizienz

Hier ist eine Klasse welche die Fibonacci-Zahlen iterativ und rekursiv berechnet und dabei die Zeit misst.

In [4]:
```
#import fast timer
from timeit import default_timer as timer

import sys
sys.setrecursionlimit(10000)

#Get Input from user not implemented
#set the n-the fibonaccinumber you would like to get.
number = 4

# 1. Implementation of two functions which calculate the n-the Fibonacci num
ber. One iterative the other recursive.
# class to calculate the fibonacci numbers
class Fibonacci:

    _usedTime = 0

    def getNTheFibonacciNumberRecursive(self, n):
        start = timer()
        result = self._fibonacciRecursive(n)
        stop = timer()
        fib.calcTime(start,stop)
        return result

    # Get the n-the fibonaccinumber iterativ
    def getNTheFibonacciNumberIterative(self, n):
        start = timer()

        nextterm = 0
        present = 1
        previous = 0

        # Get only values greater than 0
        if n < 0:
            print("Incorrect input")

        i = 0
        while i < n:
            nextterm = present + previous
            present = previous
            previous = nextterm
            i += 1

        stop = timer()
        self.calcTime(start,stop)
        return nextterm

    def calcTime(self, startTime, stopTime):
        self._usedTime = (stopTime-startTime)

    def getUsedTimeToCalculateNtheNumber(self):
        return self._usedTime

    # private get the n-the fibonaccinumber recursive
    def _fibonacciRecursive(self, n):
        retVal = 0
        # Get only values greater than 0
        if n < 0:
            print("Incorrect input, n has to be greater than 0")
        # First Fibonacci number is 0
        elif n == 0:
            retVal = 0
        # Second Fibonacci number is 1
        elif n == 1:
            retVal = 1
        else:
            retVal = self._fibonacciRecursive(n - 1) + self._fibonacciRecurs
ive(n - 2)
```

```
Recursive calculation of the Fibonacci Number
At position 4 is Fibonacci number: 3
Process time: 0.0026879997676587664ms


Iterative calculation of the Fibonacci Number:
At position 4 is Fibonacci number: 3
Process time: 0.0010710000424296595ms
```

Diskussion der Ergebnisse

Aufgabe 1 und 2

Wie wir im Plot bei Aufgabe 3 sehen stimmen die Rückgabewerte der funktion fib() für die Startwerte $f_0 := 0 \text{ und } f_1 := 1$ sowie für die weiteren Werte f_n.

Aufgabe 3

Anhand der geploteten Zahlen sehen wir, dass die Anzahl Funktionsaufrufe immer etwa 3x grösser ist als die Fibonacci Zahl. Daher gehen wir mit einem "educated guess" vor und nehmen an es ist eine Folge welche ähnlich der Fibonacci-Folge ist, also ähnlich wie $a_n := a_{n-1} + a_{n-2}$

Testen wir dies für $a_2 = a_0 + a_1 a_2 = 1 + 1 = 2$ gemäss unseres plott sollte aber $a_2 = 3$ sein.

Wir passen die Formel an zu $a_n := a_{n-1} + a_{n-2} + 1$

Nun is $a2 = 3$

Testen wir $a3 = a2 + a1 + 1 = 3 + 1 + 1 = 5$ dies entspricht unseren geplotteten Daten.

Gehen wir nun mit der Aussage $a_n := a_{n-1} + a_{n-2} + 1$ den Induktionsalgorithmus (1) durch.

1. Induktionsanfang $a_0 = 1$
   $a_1 = 1$

1. Induktionsvoraussetzung $a_2 = 3$ Die Aussage gilt für ein $n \in \mathbb{N}$ n 0 .
2. Induktionsbehauptung Die Aussage gilt für n als auch für $n + 1$
3. Induktionsschluss:

$a_{n+1} = a_{n+1-1} + a_{n+1-2} + 1$

$a_{n+1} = a_n + a_{n-1} + 1 \Longleftrightarrow a_n := a_{n-1} + a_{n-2} + 1 \square$

$a_3 = a_2 + a_1 + 1$

Quelle:
(1) Vollsändige Induktion Dalwigk F. ISBN 978-3-662-58632-7

Aufgabe 4 und 5

Wir sehen sehr deutlich, dass die iterative Methode wesentlich kürzere Rechenzeiten beansprucht und wesentlich efizienter ist. Dies verwundert nicht da die Anzahl Funktionsaufrufe bei der rekursiven Methode exponentiell grösser wird im Gegensatz zur iterativen Methode bei welcher die Rechenzeit linear länger wird.