Semesterarbeit Teil 1a: Rekursion mit Python anhand der Fibonacci-Folge

Aufgabenstellung

Die Fibonaccifolge ist wie folgt definiert:

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

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

Implementieren Sie eine Python-Funktion fib(n) , die die n-te Fibonacci-Zahl bestimmt.

1. 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.
2. Vergleichen Sie die Anzahl der Funktionsaufrufe von fib zur Bestimmung einer Fibonacci-Zahl mit den Fibonacci-Zahlen selber. Können Sie eine Vermutung aufstellen?
3. Verwenden Sie die Funktion time() aus dem Modul time, um zu bestimmen, wie lange die Funktion fib benötigt, um eine Fibonacci-Zahl zu bestimmen.
4. 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.)

Analyse

Die Python-Funktion fib(n), welche für diese Aufgabe zu implementieren ist, muss folgende Bedingungen erfüllen:

- Die mathematische Formel für die Funktion lautet: $f_n := f_{n-1} + f_{n-2} \text{ für } n \geq 2$
- Für folgende Eingaben müssen folgende Ausgaben hervorgehen:

Eingabe	1	2	3	4	5	6	7	8	9	10
Ausgabe	1	1	2	3	4	8	13	21	34	55

- Die Implementation muss mit folgenden Lösungswegen implementiert werden:
  - Native Implementation: die mathematische Formel direkt umsetzen.
    - Für diese Implementation muss zusätzlich eine weitere Funktion implementiert werden, um die Funktionsaufrufe zu zählen.
    - Die Anzahl der Funktionsaufrufe muss mit den Fibonacci-Zahlen verglichen werden.
  - Rekursive Implementation (möglichst effizient)
  - Iterative Implementation (möglichst effizient)
- Für alle Implementationen muss eine zusätzliche Funktion implementiert werden, um zu bestimmen, wie lange die implementierte Funktion benötigt um die Fibonacci-Zahl zu bestimmen.
- Nicht in der Aufgabenstellung: Die Performance der verschiedenen Implementationen muss verglichen werden.

Implementation

Struktur

Für diese Übung habe ich 3 Python-Scripts erstellt:

- «fibonacci.py»: Dieses Script enthält die ganzen Implementationen aus der Analyse.
- «test.py»: Dieses Script dient zur Überprüfung der Implementationen.
- «tools.py»: Dieses Script enthält eine Funktion zur Überprüfung von 2 Arrays, diese Funktion wird im «test.py»-Script verwendet um die Soll- und Ist-Resultate zu vergleichen.

Tools-Script

Auf dieses Script werde ich nur ganz oberflächlich eingehen, da dieses nicht direkt von der Aufgabenstellung verlangt wird. Ich habe dieses Script nur implementiert, um meine Arbeitsqualität zu testen.

Dieses Script enthält nur die Funktion «check_arrays(results, expected_results, on_function = None)». Diese Funktion vergleicht das Array «results» mit dem Array «expected_results» und gibt Wahr oder Falsch zurück. Falls der Parameter «on_function» auch übergeben wird, wird das Resultat entsprechend in der Konsole angezeigt.

Test-Script

Auf dieses Script werde ich nur ganz oberflächlich eingehen, da dieses nicht direkt von der Aufgabenstellung verlangt wird. Ich habe dieses Script nur implementiert, um meine Arbeitsqualität zu testen.

Dieses Script importiert das Tools-Script und das Fibonacci-Script und wendet diese an, um die Aufgabestellung zu lösen. Das Script zeigt die verschiedenen Implementationen gemäss Analyse an und wertet diese aus. Ich habe dieses Script zuerst geschrieben und dann entsprechend die Funktionen implementiert, somit konnte ich direkt überprüfen, ob meine Funktionen korrekt implementiert wurden, ausserdem sind die Funktionen so unabhängig im Fibonacci-Script enthalten.

Fibonacci-Script

Dieses Script ist die Haupt-Implementation für diese Aufgaben, deshalb werde ich hier etwas genauer auf die verschiedenen Funktionen eingehen.

«fib(n)»

Wie die mathematische Funktion führt auch diese Funktion sich selbst immer wieder aus, bis «n» kleiner als 2 ist.

«fib_count(n)»

Diese Funktion berechnet die Fibonacci-Zahl, zusätzlich zählt sie jedoch noch die Funktionsaufrufe. Sie gibt ein Array zurück mit den Werten «[Fibonacci-Zahl, Anzahl-Aufrufe]».

«fib_time(n)»

Diese Funktion speichert die Zeit vor dem «fib(n)»-Aufruf und die Zeit nach dem «fib(n)»-Aufruf.

Sie gibt ein Array zurück mit den Werten «[Fibonacci-Zahl, Zeitdifferenz]». Die Zeitdifferenz ist in Millisekunden.

«fib_fast_recursive(n, a = 1, b = 1)»

Dies ist die rekursive Implementation gemäss Analyse. Die Parameter «a» und «b» sind optional und werden rekursiv übergeben um den aktuellen Status zu merken.

Aus mathematischer Perspektive wird folgendes gemacht: Die Zahl «n» wird heruntergezählt bis diese auf «0» steht. Für jeden Zählschritt wird «a» auf «b» gesetzt und «b» auf «a + b». Sobald alle Zählschritte durch sind, wird «a» zurückgegeben, da dies die gesuchte Fibonacci-Zahl ist.

«fib_fast_recursive_time(n)»

Diese Funktion speichert die Zeit vor dem «fib_fast_recursive (n)»-Aufruf und die Zeit nach dem «fib_fast_recursive (n)»-Aufruf.

Sie gibt ein Array zurück mit den Werten «[Fibonacci-Zahl, Zeitdifferenz]». Die Zeitdifferenz ist in Millisekunden.

«fib_fast_iterative(n)»

Grundsätzlich wird hier das selbe System angewendet wie bei der rekursiven Implementation. Der Unterschied hier ist, dass die Berechnungen in einer Loop ausgeführt werden.

Beide «fast_*»-Implementationen sind bei weitem schneller als die Standard Implementation. Dies liegt daran, dass die erste Implementierung sich exponentiell selbst aufruft. Bei kleinen Zahlen stellt dies kein Problem dar, sind jedoch die Zahlen grösser, werden so viele Funktionsaufrufe gestartet, dass bereits die Berechnung der 50.ste Fibonacci-Zahl nicht mehr vertretbar ist. Da für die rekursive und iterative Variante der Code nur so viele Male ausgeführt wird, wie die Fibonacci-Zahl gross ist, ist auch die Berechnung für die 500.ste Fibonacci-Zahl kein Problem. Die Ausgabe ist jedoch für beide Varianten dieselbe.

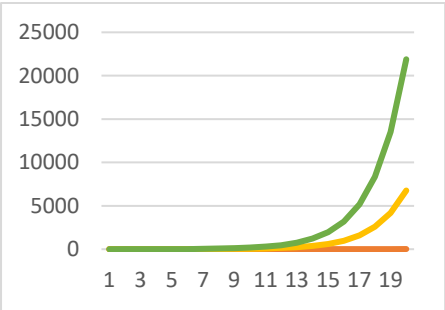
«fib_fast_iterative_time(n)»

Diese Funktion speichert die Zeit vor dem «fib_fast_iterative (n)»-Aufruf und die Zeit nach dem «fib_fast_iterative (n)»-Aufruf.

Sie gibt ein Array zurück mit den Werten «[Fibonacci-Zahl, Zeitdifferenz]». Die Zeitdifferenz ist in Millisekunden.

Vergleich: Zahl und Aufrufe

Für den Vergleich von Fibonacci-Zahl und Funktionsaufrufe habe ich ein Diagramm für die ersten 20 Zahlen erstellt:



* Eingabe
* Fibonacci-Zahl
* Funktionsaufrufe

(Excel-Datei ist auch vorhanden)

Auf dem Diagramm kann man erkennen, dass der exponentielle Faktor zwischen der Fibonacci-Zahl und den Funktionsaufrufen ungefähr 300% beträgt. Das heisst, die Funktionsaufrufe betragen ungefähr 3-mal die Fibonacci-Zahl. Dieses Verhältnis wird immer grösser, je grösser die Zahl wird. Hierzu könnte man bestimmt einen Grenzwert aufstellen, dazu habe ich jedoch nicht genügend Zahlen gesammelt, da die Zählung der Funktionsaufrufe extrem lange dauert bei höheren Zahlen. (Ungefähr 3.23x)