Übung 8 - Musterlösung

Abgabe 21.6.2012

Aufgabe 1 – Koch-Schneeflocke

14 Punkte

a) Diese Aufgabe lässt sich mit einfacher Vektorgeometrie anhand des Bildes lösen. Seien $\overrightarrow{p_1}, \overrightarrow{p_2}$ die Endpunte der Strecke, die verfeinert werden soll. Die Verfeinerung produziert fünf neue Punkte, wobei der erste und der fünfte mit den ursprünglichen Punkten übereinstimmen:

$$\overrightarrow{q_1} = \overrightarrow{p_1}, \quad \overrightarrow{q_5} = \overrightarrow{p_2}$$

Der zweite und der vierte Punkt liegen auf der Verbindungslinie der Endpunkte bei 1/3 und 2/3 der ursprünglichen Länge:

$$\vec{d} = \overrightarrow{p_2} - \overrightarrow{p_1}$$

$$\overrightarrow{q_2} = \overrightarrow{p_1} + \frac{1}{3}\vec{d}$$

$$\overrightarrow{q_4} = \overrightarrow{p_1} + \frac{2}{3}\vec{d}$$

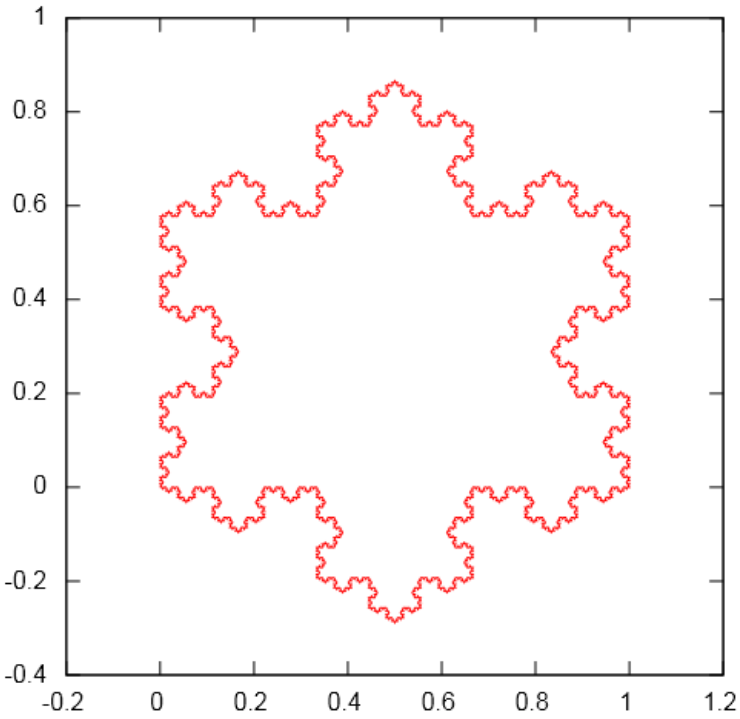
Der dritte Punkt liegt in der Mitte zwischen den ursprünglichen Punkten, aber um die Höhe des gleichseitigen Dreiecks nach außen versetzt. Deshalb brauchen wir den Normalenvektor:

$$\vec{n} = \begin{pmatrix} n_0 \\ n_1 \end{pmatrix} = \begin{pmatrix} d_1 \\ -d_0 \end{pmatrix}$$

Die Höhe eines gleichseitigen Dreiecks beträgt $\sqrt{3}/2$ der Seitenlänge, und diese wiederum muss 1/3 der ursprünglichen Streckenlänge sein. Der Vorfaktor der Normalenvektors ist deshalb $\sqrt{3}/6$:

$$\overrightarrow{q_3} = \overrightarrow{p_1} + \frac{1}{2}\vec{d} + \frac{\sqrt{3}}{6}\vec{n}$$

b) Eine mögliche Implementation befindet sich im Anhang unter `snowflake.py`.



Aufgabe 2 – Fibonacci-Zahlen 10 Punkte

a) Eine mögliche Implementation befindet sich im Anhang unter `fibonacci.py`. Die Ausgabe zeigt, welche Algorithmen wie beendet werden:

```
_____________________Aufgabe a)_____________________________________________
overtime! N=40,time=21.6648619108, algorithm:fib1
------------------------------
exception! N=1212,time=0.000305838644458, algorithm:fib3
maximum recursion depth exceeded
------------------------------
overtime! N=276478,time=15.3032030953, algorithm:fib5
------------------------------
```

Die einzelnen Unterschiede sind wie folgt zu erklären: `fib1` wächst exponentiell und ist daher sehr ineffizient. Dieser Algorithmus erreicht bei einem sehr geringen `N` die 10 Sekunden Grenze. `Fib3` erreicht, wie die Exception sagt, die maximale Rekursiontiefe bei `N=995`. `fib5` hingegen ist effizienter und die Rechenzeit wächst zunächst linear bei wachsendem `N`. Da jedoch die Fibonaccizahlen selbst exponentiell wachsen, wird ihre Addition immer aufwändiger und das Wachstum verlangsamt sich bei großen N (nur dank des Python-Typs "long" für beliebig große ganze Zahlen können wir überhaupt so große Fibonacci-Zahlen berechnen, aber der aufwand für die arithmetischen Operationen hängt jetzt von der Größe der Zahlen ab.)

b) Eine mögliche Implementation befindet sich im Anhang unter `fibonacci.py`. Die 2x2 Matrizen wurden als Tupel der Länge 4 realisiert, dementsprechend wurde die Matrixmultiplikation umgesetzt. Der eigentliche Potenzierungsalgorithmus wurde 1 zu 1 übernommen. Die finden des `N`, für welches 10 Sekunden Berechnungszeit gebraucht werden, wurde durch binäre Suche umgesetzt (nach einer Idee von Saskia Klaus und Axel Wagner). Die Ausgabe:

```
_____________________Aufgabe b)_____________________________________________
fib6: time= 9.74141188665  N = 3670016 Stellen:  766987.634035
------------------------------
fib5 and fib6 produce equal results
```

`fib6` wächst zunächst logarithmisch, da jedoch die Fibonaccizahlen selbst exponentiell wachsen, nivelliert sich das Wachstum später auf eine lineare Rate.

Aufgabe 3 – Binäre Suche 16 Punkte

a) Der Aufwand der binären Suche lässt sich funktionell wie folgt darstellen:

$$T(n) = T\left(\frac{n}{2}\right) + f(n)$$

Die Funktion $f(n)$ beschreibt den Aufwand für das Aufteilen des Arrays. Für die beiden Varianten gilt:

$$f(n) \in O(n)\ (pass\ by\ value)$$

$$f(n) \in O(1)\ (pass\ by\ reference)$$

Nun muss ρ bestimmt werden. Mit $a = 1$ und $b = 2$ gilt:

$$\rho = \log_b a = \log_2 1 = 0$$

Nun können wir die einzelnen Fälle klassifizieren:

<u>Pass by Reference:</u>

Es ergibt sich Fall 2 des mster-Theorems:

$$f(n) \in O(n^\rho), mit\ \rho = 0$$

Daraus folgt also:

$$T(n) \in \Theta(n^\rho \cdot \log(n) \overset{\rho=0}{\Rightarrow} T(n) \in \Theta(\log(n))$$

<u>Pass by Value:</u>

Hier haben wir Fall 3 des Mster-Theorems:

$$f(n) \in O(n^{\rho+\epsilon}), mit\ \rho = 0\ und\ \epsilon = 1$$

Daraus folgt

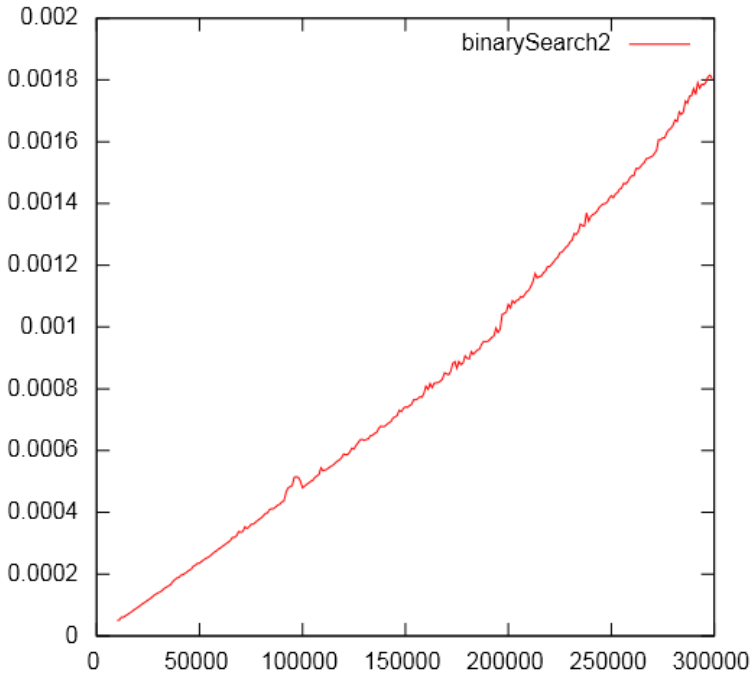
$$T(n) \in \Theta\big(f(n)\big) \in \Theta(n)$$

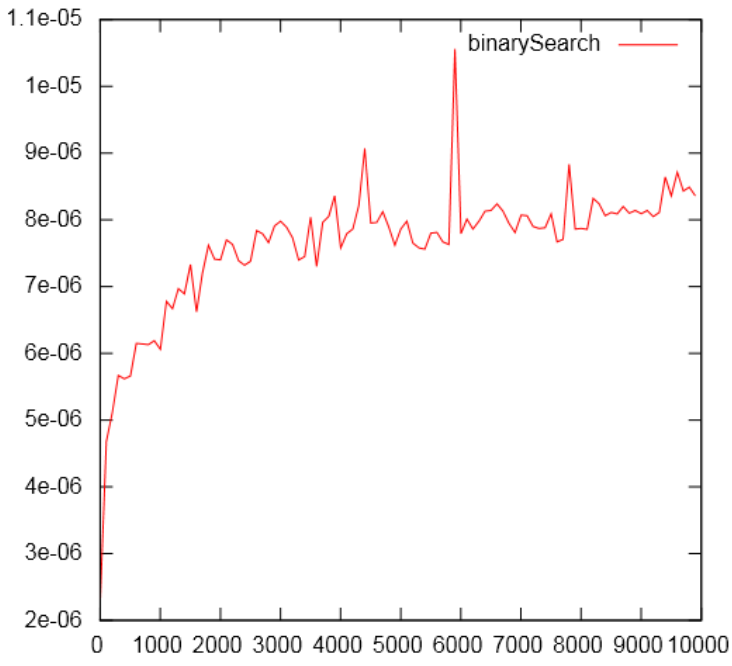
wenn die folgende Zusatzbedingung erfüllt werden kann:

$$\exists\, c\ \forall\, n \in \mathbb{N} : f\left(\frac{n}{2}\right) \leq c \cdot f(n)$$

Hier kann $c = \frac{1}{2}$ gewählt werden, da es sich um einen Kopiervorgang handelt.

b) Mittels des Timeit-Moduls gewinnt man die Daten für folgende Plots:





Die lineare bzw. Logarithmische Abhängigkeit ist sehr gut zu erkennen.

c) Eine mögliche Implementation befindet sich im Anhang unter `binarySearch.py`.

Anhang – snowflake.py

```
from math import sqrt
from numpy import ndarray
from Gnuplot import Gnuplot

def kochSnowflake(level=4):

    level = int(level)
    points = [] #initialize result array and init points
    i1 = ndarray(2)
    i1[0:2] = [0, 0]
    i2 = ndarray(2)
    i2[0:2] = [1, 0]
    i3 = ndarray(2)
    i3[0:2] = [0.5, 0.5 * sqrt(3)]
    points.append(i1) #append init points
    points.append(i2)
    points.append(i3)
    for i in range(level): #for each lvl iter through all points, and add points of next level
        tmp = []
        for j in range(len(points)):
            p1 = points[j] #edgepoints
            p2 = points[(j+1)%len(points)]
            d = p2 - p1
            n = ndarray(2)
            n[0], n[1] = d[1], -d[0] #normal vector
            q1 = p1+1.0 / 3 * d #new points
            q2 = p1+1.0 / 2 * (d + 1 / sqrt(3) * n)
            q3 = p1+2.0 / 3 * d
            tmp += [p1,q1, q2, q3] #append old and new points, minus one edge
                                                    #point
        points = tmp            #so there are no multiple points
    tmp.append(tmp[0]) #append one last point for the closing edge
    return tmp

if __name__ == "__main__":

    import sys
    points = kochSnowflake(sys.argv[1]) #get points
    plot = Gnuplot() #get the plot
    plot('set term svg enhanced') #set graph properties
    plot('set out "snowflake.svg"')
    plot('set size square')
    plot('set xrange [-0.2:1.2]')
    plot('set yrange [-0.4:1.0]')
    plot('set style data lines')
    plot.plot(points) #plot!
```

Anhang – fibonacci.py

```
from timeit import Timer
from math import log10

#Implementation
def fib1(n):
    if n <= 1:
        return n
    return fib1(n-1) + fib1(n-2)
def fib3Impl(n):
    if n == 0:
        return 1, 0
    else:
        f1, f2 = fib3Impl(n-1)
        return f1 + f2, f1
def fib3(n):
    f1, f2 = fib3Impl(n)
    return f2
def fib5(n):
    f1, f2 = 1, 0
    while n > 0:
        f1, f2 = f1 + f2, f1
        n -= 1
    return f2

# multiplication of two symmetric 2x2 matrices represented as vectors of length 4
def matmul(a, b):
    t = a[0]*b[1] + a[1]*b[3]
    return [a[0]*b[0] + a[1]*b[2], t,
            t, a[2]*b[1] + a[3]*b[3]]

# power of two symmetric 2x2 matrices represented as vectors of length 4
def matpow(a, n):
    if n == 0:
        return [1, 0, 0, 1]
    elif n == 1:
        return a
    elif n % 2 == 0:
        return matpow(matmul(a, a), n / 2)
    else:
        return matmul(a, matpow(matmul(a, a), n / 2))

# use the matrix power
def fib6(n):
    f = matpow([1, 1, 1, 0], n)
    return f[1]
```

```
#test implementations
def timeFib(fib):
    N = 2
    lastTime = 0
    while True:
        t = Timer(fib.__name__+"("+str(N)+")","from __main__ import "+fib.__name__)
        try: #try to do the computation
            time = t.timeit(1)
            #increment lineary or exponentially, depending on growth rate
            if lastTime/time < 0.5:
                N += 2
            else:
                N *= 2
            lastTime = time
            if time > 10: #if it takes too long
                print 'overtime! N=%s,time=%s, algorithm:%s'%(N,time,fib.__name__)
                print '------------------------------'
                break
        except Exception as e: # if an exception occurs
            print 'exception! N=%s,time=%s, algorithm:%s'%(N,time,fib.__name__)
            print e
            print '------------------------------'
            break

def timeCompare(): #find the N with computation time ca 10 s
    N = 1
    flag = 0
    # a simple binary search
    while True:
        T = Timer(fib6.__name__+"("+str(N)+")","from __main__ import "+fib6.__name__)
        time = T.timeit(1)
        #first, reach 10 s
        if flag != 1:
            if time < 10:
                minN = N
                N *= 2
                maxN = N
            elif time > 10:
                minN = int(N/2)
                maxN = N
                N = int(minN+(maxN-minN)/2)
                flag = 1
                continue
        #then, binary search until...
        else:
            if time > 10:
                maxN = N
                N = int(minN+(maxN-minN)/2)
```

```
        elif time < 10:
            minN = N
            N = int(minN+(maxN-minN)/2)
        else:
            break
        #...the tolerance limit is reached
        if abs(time-10)< 0.3:
            break
    #print the results!
    print 'fib6: time=',time,' N =',N,'Stellen: ',log10(fib6(N))
    #test fib5 vs fib6
    for i in range(1,1000):
        assert fib6(i)==fib5(i)
    print '------------------------------'
    print 'fib5 and fib6 produce equal results'
if __name__ == "__main__":
    print '_____Aufgabe a)_____'
    timeFib(fib1)
    timeFib(fib3)
    timeFib(fib5)
    print '_____Aufgabe b)_____'
    timeCompare()
```

Anhang – binarySearch.py

```
from Gnuplot import Gnuplot,PlotItems
from timeit import Timer
from math import log
import unittest
from random import randint
#implement Search algorithms
#by reference
def binarySearch(a, key, start, end):
    size = end-start
    if size <= 0:
        return None
    center = (start + end) / 2
    if key == a[center]:
        return center
    elif key < a[center]:
        return binarySearch(a, key, start, center)
    else:
        return binarySearch(a, key, center+1, end)
#by value
def binarySearch2(a, key):
    if len(a) == 0:
        return None
    center = len(a) / 2
    if key == a[center]:
        return center
    elif key < a[center]:
        return binarySearch2(a[:center], key)
    else:
        res = binarySearch2(a[center+1:], key)
    if res is None:
        return None
    else:
        return res + center + 1

def binarySearchI(a, value, start, end):
    while start <= end: #while start is left of end
        mid = (start + end) / 2; #calculate mid in the current array
        if a[mid] > value: #if the value in the mid position is larger then value
            end = mid - 1; #new search array searches through left half
        elif a[mid] < value: #likewise
            start = mid + 1;
        else:
            return mid; #if a[mid] == value, return position
    return -1 #if value not in array, return error code!

#testclass for iterative binary search. Use 'nosetests binarySearch.py'
```

```
class BinaryTest(unittest.TestCase):
    def test_Iterative(self):
        n = 1000
        for i in range(4,n):
            array = range(2,i)
            value = randint(2,i-1)
            self.assertEqual(binarySearchI(array,value,0,len(array)),array.index(value),\
                    'Binary Iterative Search algortihm did not work.')

if __name__ == "__main__":
    #get data
    data1 = []
    data2 = []
    for N in range(10000,300000,1000):
        t2 = Timer("binarySearch2(a,str(randint(0,"+str(N)+")))","from random import randint\nfrom
__main__ import binarySearch2\na=range("+str(N)+")")
        time2 = t2.timeit(100)/100
        data2.append((N,time2))
    for N in range(2,10000,100):
        t1 = Timer("binarySearch(a,str(randint(0,"+str(N)+")),0,len(a))","from random import
randint\nfrom __main__ import binarySearch\na=range("+str(N)+")")
        time1 = t1.timeit(100)/100
        print time1
        data1.append((N,time1))
    plot = Gnuplot()
    plot('set term svg enhanced')
    plot('set out "binary.svg"')
    plot('set size square')
    plot('set style data lines')
    plot1 = PlotItems.Data(data1,title="binarySearch")
    plot2 = PlotItems.Data(data2,title="binarySearch2")
    plot.plot(plot1)
    plot('set out "binary2.svg"')
    plot.plot(plot2)
```