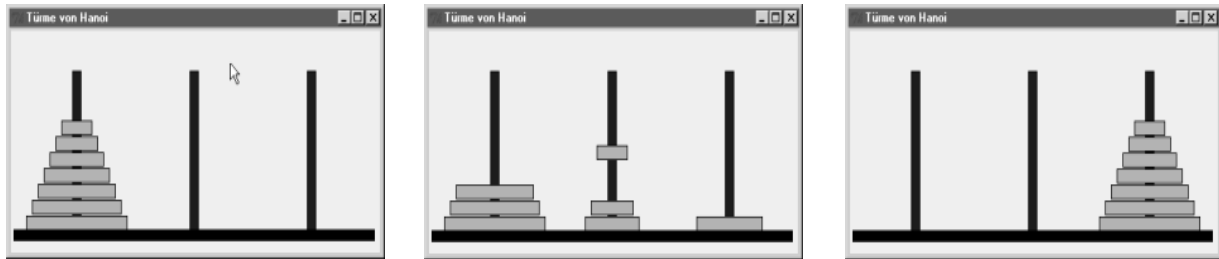


Die Türme von Hanoi



1883 wurde in Frankreich ein neuartiges Spiel verkauft. Es bestand aus einem Brett mit drei aufrecht stehenden Stäbchen. Auf einem der Stäbchen waren 8 verschieden große Scheiben aufgereiht, von der größten unten bis zur kleinsten oben. Als Erfinder zeichnete ein gewisser 'Prof. Claus' von der Universität 'Li-Sou-Stian' verantwortlich. Folgende Geschichte lag dem Spiel bei:

Eine alte indische Sage erzählt von einem Tempel in der Stadt Benares, wo Mönche am 'Turm des Brahma' arbeiten. Ihre Aufgabe ist es, einen Turm von 64 aufeinandergetürmten unterschiedlich großen Goldscheiben von einem Pflock auf einen anderen zu transferieren, wobei sie einen dritten Pflock als Ablage verwenden dürfen. Allerdings darf niemals eine größere Scheibe auf eine kleinere gelegt werden, und in jedem Zug darf nur eine einzige Scheibe bewegt werden. Der Sage nach wird noch vor Beendigung dieser Aufgabe der Tempel zu Staub zerfallen und das Universum mit einem Schlag im Nichts verschwinden.

Das Ziel des Spiels war es, diese Aufgabe mit 8 Scheiben durchzuführen und dabei die geringste nötige Anzahl von Zügen auszuführen.

Du kannst Dir ein solches Spiel leicht selbst basteln! Nimm einfach unterschiedlich große Münzen, Bücher, Teller oder ähnliches. Beginne vorerst mit einem Turm von 3, 4 oder 5 'Scheiben'!

Nebenbemerkung: Der Name 'Prof. Claus' aus 'Li-Sou-Stian' wurde rasch als Anagramm (Buchstabenvertauschung) von 'Prof. Lucas' von der Universität 'Saint Louis' erkannt. Du kennst diesen Mathematiker Edouard Lucas vermutlich von seiner Verallgemeinerung der Fibonacci-Folge aus dem Mathematikunterricht.

Die Herausforderung für uns: Wie lösen wir die Aufgabe mithilfe des Computers, und was ist die nötige Zuganzahl? Kühn wie wir sind wollen wir die Aufgabe natürlich für jede Anzahl von Scheiben lösen – egal ob 8, 64, 17 oder 100 Scheiben auf dem Anfangsstapel liegen. Und vielleicht bekommen wir auch noch heraus, wie lange unser Universum noch existieren wird...

Bezeichnungen

Wie immer sind präzise Sprechweisen äußerst hilfreich: Bezeichnen wir mit	
links,Mitte,rechts	die drei Pflöcke auf dem Brett nach ihrer Position
n	die Anzahl der Scheiben
A,B,C,... oder 1,2,3,...	die Scheiben, wobei A bzw. 1 die kleinste ist
Turm(n)	einen Hanoi-Turm mit n Scheiben (die kleinste oben)
start, hilf, ziel	die Namen der Pflöcke je nach ihrer Verwendung

Dann sehen wir : ein Turm(4) ist nichts anderes als eine vierte Scheibe (D) unten und ein Turm(3) darüber. Klingt stark nach Rekursion!

Tatsächlich: Wenn Du Dich spielerisch mit dem Turm(4)-Spiel auseinandergesetzt hast, wirst Du gemerkt haben, dass Du gelegentlich einen Turm(3) vor Dir auf einem Pflock hattest, gelegentlich hast Du auch die Versetzung von Turm(2)-Objekten überlegt!

Spiele die Sache nochmals durch (keine Sorge – in 15 Zügen kannst Du es schaffen!) und achte auf das Erscheinen von diesen kleineren 'Teiltürmen'!

Wichtig: Irgendwann ist 'ziel' leer, auf 'start' liegt die größte Scheibe alleine und auf 'hilf' sitzt ein Turm(3). Was ist Dein nächster Zug, und wie kannst Du die restliche Aufgabe kurz beschreiben?

Die gewonnene Erkenntnis ist leicht formuliert:

Wie versetzt man einen Turm(4) mit Scheiben ABCD von 'start' nach 'ziel'?

- 1.) wir versetzen Turm(3), also ABC, von 'start' nach 'hilf'
- 2.) Jetzt können wir die größte Scheibe D frei nach 'ziel' bewegen.
- 3.) Nun müssen wir noch Turm(3) von 'hilf' nach 'ziel' verschieben.

Allerdings benötigen wir zum Verschieben größerer Türme den Hilfspflock 'hilf'. Im ersten Schritt ist das kein Problem, aber im dritten ist 'hilf' anfänglich sicher besetzt. Doch da könnte einfach 'start' die Rolle des Hilfspflocks spielen, da es ja leer ist.

Nicht nur leere Pflöcke können Hilfe leisten: während des Umschichtens von Turm(3) in 1.) mussten wir gelegentlich eine Scheibe auf dem Startpflock zwischenlagern – kein Problem: für eine Scheibe gewisser Größe ist ein Teilturm, der oben eine größere Scheibe hat als die zu bewegende doch auch als Hilfspflock gut brauchbar! Das führt zu

Wie versetzt man einen Turm(4) mit Scheiben ABCD von start nach ziel (mit Ablage hilf)?

- 1.) wir versetzen Turm(3), also ABC, von start nach hilf, Hilfspflock ziel
- 2.) Jetzt können wir die größte Scheibe D frei von start nach ziel bewegen.
- 3.) Nun müssen wir noch Turm(3) von hilf nach ziel verschieben, Hilfspflock start

schiebe Turm(n) von start nach ziel mit Hilfspflock hilf

```
def Turmschieben(n,start,ziel,hilf):           Turm(n)
    Turmschieben(n-1,start,hilf,ziel)       Turm(n-1) weicht auf hilf aus
    versetze Scheibe n von start nach ziel  Scheibe ins Ziel schieben
    Turmschieben(n-1,hilf,ziel,start)       Turm(n-1) auf ziel holen
```

Wunderbar rekursiv formuliert!

Nur noch die richtige Abbruch-Bedingung einfügen, und wir haben es geschafft:

Wann sind wir fertig? Wenn die Anzahl der Scheiben Null ist!

Weiters wäre es nett, könnte man in jedem Teilschritt erfahren, welche Scheibe von wo nach wo transferiert wird!

	(alternativ mit weniger Rekursion)
<pre>def hanoi(n,start,ziel,hilf): if n==0: return hanoi(n-1,start,hilf,ziel) print n,'von',start,'nach',ziel hanoi(n-1,hilf,ziel,start)</pre>	<pre>def hanoi(n,start,ziel,hilf): if n>0: hanoi(n-1,start,hilf,ziel) print n,'von',start,'nach',ziel hanoi(n-1,hilf,ziel,start)</pre>
<pre>hanoi(4,'Links ','Rechts','Mitte ')</pre>	4 Scheiben von Pflock 'Links' nach Pflock 'Rechts' mit einem Hilfspflock 'Mitte' verschieben!

Die Bildschirmausgabe:

```
1 von Links nach Mitte | | | ...Turm(1) A
2 von Links nach Rechts | | | ... Turm(2) Scheibe B
1 von Mitte nach Rechts | | | ...Turm(1) A
3 von Links nach Mitte | ... Turm(3) Scheibe C
1 von Rechts nach Links | | | ...Turm(1) A
2 von Rechts nach Mitte | | | ... Turm(2) Scheibe B
1 von Links nach Mitte | | | ...Turm(1) A
4 von Links nach Rechts | Scheibe D hinüber
1 von Mitte nach Rechts | | | ...Turm(1) A
2 von Mitte nach Links | | | ... Turm(2) Scheibe B
1 von Rechts nach Links | | | ...Turm(1) A
3 von Mitte nach Rechts | ... Turm(3) Scheibe C
1 von Links nach Mitte | | | ...Turm(1) A
2 von Links nach Rechts | | | ... Turm(2) Scheibe B
1 von Mitte nach Rechts | | | ...Turm(1) A
```

Wir wollten aber doch auch wissen, wie viele Schritte jeweils nötig sind!

Wenn man sich obige Erklärung ansieht, dann kann man es an der Zahl der Zeilen ablesen:

Turm(1) : 1 Schritt

Turm(2) : 3 Schritte

Turm(3): 7 Schritte

Turm(4): 15 Schritte

Turm(n) schieben = Turm(n-1) wegbringen + größte Scheibe ins Ziel + Turm(n-1) ins Ziel holen

Schritte(n) = 2*Schritte(n-1) + 1 ,

ein Hanoi Spiel mit n Scheiben braucht $2^n - 1$ Schritte.

Kann das der Computer auch berechnen?

```

def hanoi1(n,start,ziel,hilf):
    global schritte
    if n==0: return
    hanoi1(n-1,start,hilf,ziel)
    schritte+=1
    # print 'Schritt',schritte,':',n,'von',start,'nach',ziel
    hanoi1(n-1,hilf,ziel,start)

def hanoischritte(n,start,ziel,hilf):
    global schritte
    schritte = 0
    hanoi1(n,start,ziel,hilf)
    return schritte

>>> print hanoischritte(4,'Links ','Rechts','Mitte ')

```

Für diese Tests kommentiert man die Print-Anweisung aus Zeitgründen besser aus...
(Nicht zu große n versuchen!) n=20 etwa braucht bereits 1048575 Schritte und einige Denkzeit...

Und wie steht's im Tempel des Brahma? Was sagt Python zur Anzahl der Schritte:

```

>>> print 2**64-1
18446744073709551615

```

Und wieviele Jahre dauert das, etwa bei einer Bewegung pro Sekunde?

```

>>> print (2**64-1)/(60*60*24*365.25)
584542046091.0

```

über 584 Milliarden Jahre... Frag Deinen Physiklehrer, wie lange es das Universum bereits gibt und wie lange es unser Sonnensystem noch geben wird!

Einen Schönheitsfehler hat dieser Algorithmus aber doch: die Scheiben 'tun' eigentlich gar nichts. Nur eine langweilige print-Zeile informiert über Verschiebungen.

Dem können wir aber abhelfen. Definieren wir doch Scheiben als Objekte, die Fähigkeiten und Eigenschaften haben!

Eine objektorientierte Version

Die Idee:

So könnte die Lösung auch gelingen: Die größte Scheibe befiehlt den über ihr befindlichen, auf einen anderen Pflock zu gehen (hilf). Dann kann sie selbst auf den Zielpflock und ruft die restlichen Scheiben zu sich zurück. Verketteten wir die Scheiben der Größe nach von unten nach oben, so braucht eine Scheibe immer nur der über ihr befindlichen einen Befehl zu geben, die darauf liegenden sollen automatisch mitwandern. Die kleinste Scheibe ist kann immer ziehen, da sie auf jeder anderen Platz findet.

Objekt Scheibe:

Eigenschaften:	name	sie hat einen Namen
	pflock	sie kennt den Pflock auf dem sie momentan sitzt
	kleiner	sie kennt die kleinere Scheibe, die auf ihr drauf sitzt
Fähigkeiten:	gehzu(ziel)	sie kann auf einen anderen Pflock wandern

- dieses 'Wandern' muss aber noch geklärt werden. Was denkt die Scheibe, wenn sie nach 'ziel' soll?
- Wenn ich die kleinste Scheibe bin, dann kann ich rüber. Ok, ich springe rüber.
 - Wenn jemand über mir ist, dann muss der fort. Aber wohin? Nun – der Pflock auf dem ich sitze, kann es nicht sein. Mein Zielpflock auch nicht. Also der, der dann noch übrigbleibt, ich nenne ihn 'hilf'.
 - Jetzt sage ich der Scheibe über mir, sie soll so nett sein, und nach 'hilf' gehen. Wie sie das macht ist ihre Sache. Aber vermutlich denkt sie genauso wie ich.
 - Nun kann ich endlich nach 'ziel' wandern
 - Und zuletzt bitte ich die Scheibe, die über mir war, wieder zu mir nach 'ziel' zurückzukommen (und ihre kleineren Kolleginnen selbstverständlich mitzubringen).

Tja, das war's eigentlich schon!

Mit passenden print-Anweisungen werden wir die Scheiben beim Nachdenken und bei ihren Gesprächen belauschen.

(Der Parameter self.tab ist nur ein Leerstring mit der Länge des Namens der Scheibe. Damit nur bei Personenwechsel der Name der Sprecherin angezeigt wird, so wie im Text eines Theaterstücks...)

<pre> class Scheibe: def __init__(self,name,pflock,kleiner): self.name = name self.pflock = pflock self.kleiner = kleiner self.tab = " "*len(name) def gehnach(self,ziel): print self.name+": hm...ich soll also von "+ \ self.pflock+" nach "+ziel if self.kleiner==None: print self.tab+": ok, ich springe rueber" self.pflock = ziel else: pfloecke = pflocknamen[:] pfloecke.remove(self.pflock) pfloecke.remove(ziel) hilf = pfloecke[0] print self.tab+": "+self.kleiner.name+ \ " bitte mach Platz und geh nach "+hilf self.kleiner.gehnach(hilf) print self.name+": ich gehe von "+ \ self.pflock+" nach "+ziel self.pflock = ziel print self.tab+": "+self.kleiner.name+ \ " komm zurueck zu mir nach "+self.pflock self.kleiner.gehnach(self.pflock) pflocknamen = ["links","Mitte","rechts"] # Start, Hilf, Ziel </pre>	<p>das denkt sich die Scheibe...</p> <p>ich bin die kleinste,;ich darf immer alles</p> <p>falls nicht</p> <p>ich suche den Zielpflock für die Scheibe über mir</p> <p>und bitte sie zu verschwinden</p> <p>jetzt wandere ich ins Ziel</p> <p>und hole die kleinere zurück zu mir</p>
---	--

Wir erzeugen 4 Scheiben mit den sprechenden Namen Maxi, Midi, Mini und Tiny. Die Reihenfolge ihrer Erschaffung muss von oben nach unten sein, da jede Scheibe einen Verweis auf die nächstkleinere mitbekommen muss.

Dann ersuchen wir die größte Scheibe, auf den Zielpflock zu wandern. Alles andere machen die Scheiben ganz von alleine!

```
def demo4():
    scheibe1 = Scheibe("Tiny",pflocknamen[0],None)
    scheibe2 = Scheibe("Mini",pflocknamen[0],scheibe1)
    scheibe3 = Scheibe("Midi",pflocknamen[0],scheibe2)
    scheibe4 = Scheibe("Maxi",pflocknamen[0],scheibe3)

    scheibe4.gehnach(pflocknamen[2])
```

Das Ergebnis ist folgender Dialog:

```
>>> demo4()
Maxi: hm...ich soll also von links nach rechts
      : Midi bitte mach Platz und geh nach Mitte
Midi: hm...ich soll also von links nach Mitte
      : Mini bitte mach Platz und geh nach rechts
Mini: hm...ich soll also von links nach rechts
      : Tiny bitte mach Platz und geh nach Mitte
Tiny: hm...ich soll also von links nach Mitte
      : ok, ich springe rüber
Mini: ich gehe von links nach rechts
      : Tiny komm zurueck zu mir nach rechts
Tiny: hm...ich soll also von Mitte nach rechts
      : ok, ich springe rüber
Midi: ich gehe von links nach Mitte
      : Mini komm zurueck zu mir nach Mitte
Mini: hm...ich soll also von rechts nach Mitte
      : Tiny bitte mach Platz und geh nach links
Tiny: hm...ich soll also von rechts nach links
      : ok, ich springe rüber
Mini: ich gehe von rechts nach Mitte
      : Tiny komm zurueck zu mir nach Mitte
Tiny: hm...ich soll also von links nach Mitte
      : ok, ich springe rüber
Maxi: ich gehe von links nach rechts
      : Midi komm zurueck zu mir nach rechts
Midi: hm...ich soll also von Mitte nach rechts
      : Mini bitte mach Platz und geh nach links
Mini: hm...ich soll also von Mitte nach links
      : Tiny bitte mach Platz und geh nach rechts
Tiny: hm...ich soll also von Mitte nach rechts
      : ok, ich springe rüber
Mini: ich gehe von Mitte nach links
      : Tiny komm zurueck zu mir nach links
Tiny: hm...ich soll also von rechts nach links
      : ok, ich springe rüber
Midi: ich gehe von Mitte nach rechts
      : Mini komm zurueck zu mir nach rechts
Mini: hm...ich soll also von links nach rechts
      : Tiny bitte mach Platz und geh nach Mitte
Tiny: hm...ich soll also von links nach Mitte
      : ok, ich springe rüber
Mini: ich gehe von links nach rechts
      : Tiny komm zurueck zu mir nach rechts
```

Tiny: hm...ich soll also von Mitte nach rechts
: ok, ich springe rüber

Es geht auch ganz anders

Eine iterative Version gibt es auch. Allerdings ist es schwierig (sogar für Mathematiker) zu beweisen, warum sie funktioniert! Dafür besteht sie aus einer so simplen Anweisungsfolge, dass man diese sogar kleinen Kindern beibringen kann. Die sitzen dann da und versetzen ganz lässig die Scheiben, wo ein Erwachsener lange nachdenken würde....

Vorschrift:

solange der Turm nicht vollständig bewegt wurde:

- bewege die kleinste Scheibe einen Platz zyklisch nach rechts
- führe die einzig mögliche Bewegung einer nicht kleinsten Scheibe aus

Erstaunlich!

Unter 'zyklisch bewegen' versteht man ein gedachtes kreisförmiges Schließen der Pflock-Kette. Also ...-links-Mitte-rechts-links-Mitte-rechts-....

Einziger Punkt zum Aufpassen: bei ungerader Scheibenzahl ist das 'rechts' in der Vorschrift durch 'links' zu ersetzen!

Schlampig formuliert:

solange Du nicht fertig bist:

- bewege die kleinste nach rechts (gerades n) / links(ungerades n)
- bewege eine Nichtkleinste

Und mit Grafik

In Gregor Lings wunderbarem Buch 'Python for Kids' findet sich ein hübsches Hanoi-Programm, das die Bewegungen der Scheiben in Form einer Animation vorführt!
(Die Bilder am Anfang dieses Textes stammen von Gregors Programm!)

Nachbemerkung für Leute mit starken Nerven!

Benennen wir die n Scheiben von oben nach unten mit A,B,C,... und stellen wir uns einen n-Würfel im n-dimensionalen Raum vor (Seitenlänge = 1 Schritt). Wenn dann jeder Buchstabe einer Raumrichtung entspricht, dann lässt sich jede Buchstabenfolge wie etwa ABDCA... deuten als: Beginne auf einer beliebigen Würfecke. Geh einen Schritt in die erste Raumrichtung (A), dann einen in die zweite (B), dann in die vierte (D), dann in die dritte (C), dann in die erste (A) usw. Führt man dies mit der richtigen Lösung einer Hanoi-Aufgabe durch, so erzeugen die n Scheiben eine sogenannte offene Hamilton-Rundreise auf dem n-Würfel: Man besucht jeden Eckpunkt des n-dimensionalen Würfels genau ein Mal und kehrt dabei nicht an den Anfangsort zurück.

Details : Mathematik / Graphentheorie