Wstęp do programowania

Rekurencja, metoda "dziel i zwyciężaj"

Paweł Daniluk

Wydział Fizyki

Jesień 2014

Rekurencja

Recursion

See “Recursion”.

Rekurencja

Recursion

If you still don't get it, see "Recursion".

Silnia

$$n! = 1 \cdot 2 \cdot \ldots \cdot n$$

$$silnia(i) = \begin{cases} 1 & i = 1 \\ silnia(i-1) \cdot i & \text{w p.p.} \end{cases}$$

Silnia

$$n! = 1 \cdot 2 \cdot \ldots \cdot n$$

$$silnia(i) = \begin{cases} 1 & i = 1 \\ silnia(i-1) \cdot i & \text{w p.p.} \end{cases}$$

Rozwiązanie rekurencyjne

```
def silnia(n):
    if n==1:
        return 1
    else:
        return silnia(n-1)*n
```

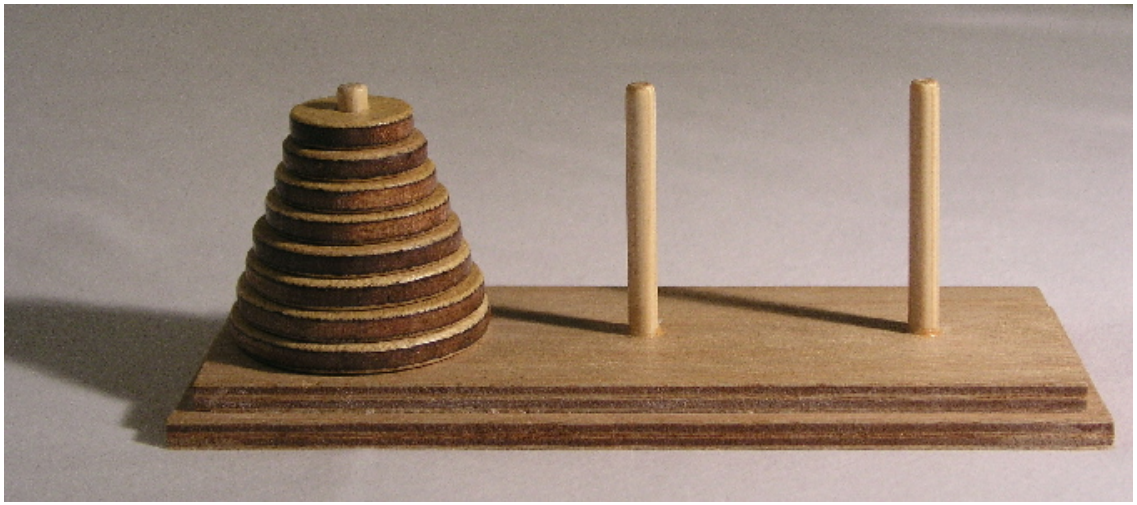
Liczby Fibonacciego

$$F_0 = 0$$
$$F_1 = 1$$
$$F_n = F_{n-1} + F_{n-2}$$

Rozwiązanie rekurencyjne

```
def fib(n):
    if n<2:
        return n
    else:
        return fib(n-1)+fib(n-2)
```

Wieże Hanoi



Jak przenieść krążki z lewego stosu na prawy, jeżeli wolno je przenosić po jednym i nie wolno kłaść większego na mniejszym?

Wieże Hanoi

Rozwiązanie

```
def move(n, f, t, h):
    if n==1:
        move_one(f, t)
    else:
        move(n-1, f, h, t)
        move_one(f, t)
        move(n-1, h, t, f)
```

Ile kosztuje rekurencja?

Niech H_n będzie liczbą przeniesień w przypadku z n krążkami.

$$H_{n+1} = 2H_n + 1$$

$$H_1 = 1$$
$$H_2 = 2H_1 + 1 = 3$$
$$H_3 = 7$$
$$H_4 = 15$$

Ile kosztuje rekurencja?

Niech H_n będzie liczbą przeniesień w przypadku z n krążkami.

$$H_{n+1} = 2H_n + 1$$

$$\begin{aligned} H_1 &= 1 \\ H_2 &= 2H_1 + 1 = 3 \\ H_3 &= 7 \\ H_4 &= 15 \end{aligned}$$

$$H_n = 2^n - 1$$

Lepiej się nie da.

Ile kosztuje rekurencja?

Niech T_n będzie liczbą dodawań potrzebnych do obliczenia `fib(n)`.

$$T_{n+1} = T_n + T_{n-1} + 1$$

$$T_{1,2,3,\ldots} = 0, 0, 1, 2, 4, 7, 12, 20, 33, 54, 87, \ldots$$

Ile kosztuje rekurencja?

Niech T_n będzie liczbą dodawań potrzebnych do obliczenia `fib(n)`.

$$T_{n+1} = T_n + T_{n-1} + 1$$

$$T_{1,2,3,\ldots} = 0, 0, 1, 2, 4, 7, 12, 20, 33, 54, 87, \ldots$$

$$F_{1,2,3,\ldots} = 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 88, \ldots$$

$$T_n = F_n - 1 = \left\lfloor \frac{\varphi^n}{\sqrt{5}} + \frac{1}{2} \right\rfloor$$

$$\varphi = \frac{1 + \sqrt{5}}{2}$$

Ile kosztuje rekurencja?

Niech O_n będzie liczbą dodawań potrzebnych do obliczenia F_n metodą iteracyjną.

Wiemy, że wystarczy dla F_n wystarczy $n-2$ dodawań.

$$O_n = O_{n-1} + 1$$

Czy można poprawić funkcję fib?

Wystarczy zapamiętywać obliczone liczby.

```
fib_list=[0,1]

def fib(n):
    if len(fib_list)>n:
        return fib_list[n]
    else:
        res=fib(n-2)+fib(n-1)
        fib_list.append(res)
        return res
```

Rekurencja ogonowa

Ostatnią operacją funkcji rekurencyjnej jest wywołanie samej siebie lub zwrócenie wyniku.

Przykład

```
def f(i):
    k=foo(i)
    return f(k)
```

albo

```
def f(i):
    k=foo(i)
    res=f(k)
    return res
```

Tak nie

```
def f(i):
    k=foo(i)
    return f(k)+1
```

Rekurencja ogonowa

Zwykła

```
def silnia(n):
    if n==1: return 1

    return silnia(n-1)*n
```

Ogonowa

```
def sil(n, acc):
    if n==1: return acc

    return sil(n-1, n*acc)

def silnia(n):
    return sil(n, 1)
```

Rekurencja ogonowa

Zwykła

```
Wywołaj silnia(3)
  Wywołaj silnia(2)
    Wywołaj silnia(1)
      Wywołaj silnia(0)
      Zwróć 1
    Zwróć 1
  Zwróć 2
Zwróć 6
```

Ogonowa

```
Wywołaj silnia(3)
  Wywołaj sil(3,1)
  Zastąp argumenty (3,1) przez (2,3), skocz na początek sil
  Zastąp argumenty (2,3) przez (1,6), skocz na początek sil
  Zastąp argumenty (1,6) przez (0,6), skocz na początek sil
  Zwróć 6 jako wynik
```

Konwersja na pętlę

Przed

```
def f(par, res):
    if done(par):
        return res

    par, res=one_step(par, res)

    f(par, res)
```

Po

```
def f(par, res):
    while not done(par):
        par, res=one_step(par, res)

    return res
```

Dziel i zwyciężaj

1. Podział problemu na 2 lub więcej części
2. Rozwiązanie każdej z części niezależnie
3. Scalenie rozwiązań części w rozwiązanie problemu pierwotnego

Każdy z podproblemów rozwiązywany jest tą samą metodą.

Naturalna implementacja przy pomocy rekurencji.

Hanoi

1. Podział: Hanoi(n, f, t, h) $\longrightarrow$ Hanoi(n-1, f, h, t), Hanoi(n-1, h, t, f)
2. Scalenie: Hanoi(n-1, f, h, t), f $\rightarrow$ t, Hanoi(n-1, h, t, f)

Mergesort

Scalanie dwóch posortowanych list

L_1, L_2 – scalane listy L – lista wynikowa

1. jeżeli L_1 i L_2 niepuste
   - niech a i b będą odpowiednio pierwszymi elementami L_1 i L_2
   - jeżeli $a \leq b$ usuń a z L_1 i dopisz na koniec L
   - jeżeli $a > b$ usuń b z L_2 i dopisz na koniec L
   - skocz do 1
2. dopisz niepustą listę do L

Scalenie list o łącznej długości N wymaga co najwyżej $N - 1$ porównań i N przepisań.

Mergesort c.d.

Sortowanie przez scalanie

1. podziel listę na dwie listy o równej długości
2. posortuj listy oddzielnie
3. scal posortowane listy

Czas potrzebny na posortowanie listy długości N:

$$T_1 = 0$$
$$T_n = 2T_{\frac{n}{2}} + n$$

Mergesort c.d.

Załóżmy, że $n = 2^k$:

$$S_k = T_n$$
$$S_0 = 0$$
$$S_k = 2S_{k-1} + 2^k$$

Ostatecznie:

$$S_k = k2^k$$
$$T_n = n \log_2 n$$

Liczby Fibonacciego

Łatwo zauważyć

$$\begin{pmatrix} F_{n+1} \\ F_n \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} F_n \\ F_{n-1} \end{pmatrix}$$

$$\begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} F_n & F_{n-1} \\ F_{n-1} & F_{n-2} \end{pmatrix} =$$

$$= \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n-1} \begin{pmatrix} F_2 & F_1 \\ F_1 & F_0 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n}$$

$$\begin{pmatrix} F_{n+m+1} & F_{n+m} \\ F_{n+m} & F_{n+m-1} \end{pmatrix} = \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} \begin{pmatrix} F_{m+1} & F_m \\ F_m & F_{m-1} \end{pmatrix}$$

Liczby Fibonacciego c.d.

$$\begin{pmatrix} F_{2n+1} & F_{2n} \\ F_{2n} & F_{2n-1} \end{pmatrix} = \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix}$$

Ostatecznie

$$F_{2n+1} = F_{n+1}^2 + F_n^2$$
$$F_{2n} = 2F_{n+1}F_n - F_n^2$$

Algorytm obliczania F_n

1. obliczyć $F_{\lfloor \frac{n}{2} \rfloor + 1}$ i $F_{\lfloor \frac{n}{2} \rfloor}$
2. zastosować wzory

Liczby Fibonacciego c.d.

Czas obliczeń

$$T_1 = 0$$
$$T_{2n} = T_n + 1$$

$$T_n = \log_2 n$$

Zadanie 1 – labirynt

Labirynt jest zdefiniowany na prostokątnej planszy o rozmiarze n na m pól. Pole może być oznaczone jako korytarz lub ściana. Dozwolone jest przechodzenie pomiędzy mającymi wspólny bok polami korytarza. Napisz program rozstrzygający, czy w zadanym labiryncie istnieje droga pomiędzy wskazanymi polami.

Wskazówka

Labirynt w Pythonie można zaimplementować jako listę list. Np.:

```
map=[[0,0,1,1,1,0,0,0],
     [1,1,1,0,1,0,1,1],
     [0,0,0,1,1,0,1,0],
     [0,0,0,1,1,1,1,0]]
```

Zadanie 2 – wyszukiwanie binarne

Zaimplementuj algorytm wyszukiwania binarnego w posortowanej liście z zastosowaniem rekurencji.

Zadanie 3 – Quicksort

Sortowanie szybkie polega na rozdzieleniu sortowanej listy na dwie: zawierającą elementy mniejsze od pewnego wybranego i zawierającą elementy pozostałe. Obie listy sortowane są niezależnie, a następnie następuje ich złączenie. Cały proces można (i należy) przeprowadzić w miejscu stosując do rozdzielania algorytm analogiczny do użytego w zadaniu o fladze polskiej.

Zadanie 4 – wieże Hanoi z ograniczeniami

Napisz program tworzący sekwencję ruchów w problemie wież Hanoi przy założeniu, że nie wszystkie ruchy są dozwolone. Przykładowo, jeżeli zakazany jest ruch z A na C, konieczne jest przenoszenie krążka z A na B, a następnie z B na C.

Strona wykładu

http://bioexploratorium.pl/wiki/Wstęp_do_programowania_
_2014z