

Tartalomjegyzék

- 1 Max. min keresése
- 2 Határérték, derivált
- 3 Gráfok
- 4 Bináris kereszfák
- 5 Rekurzív algoritmus
- 6 Egy klasszikus rekurzió példa
- 7 A két Fibonacci-megoldó függvény futásidejei:

Max, min keresése

Maximumot vagy minimumot is kereshetünk numerikusan. Ez is feltételezi, hogy a bemenet egy egyváltozós folytonos függvény, és egy lokális maximumot keres meg közelít?leg.

```
find_local_maximum(start, end)
find_local_minimum(start, end)
```

Határérték, derivált

Kiszámíthatjuk egy kifejezés határértékét egy pontban, vagy a deriváltját, határérték.

Határérték:

```
sage: ((2**x - 1)/sin(x)).limit(x = 0)
log(2)
```

Derivált:

```
sage: (2**x - 1).derivative(x)
2^x*log(2)
```

Ellen?rzés:

```
sage: (2**x - 1).derivative(x).subs(x=0)
log(2)
```

Gráfok

Gráf: csúcsok (nodes), és a csúcsokat összeköt? élek(edges) halmaza.

Például:

$$V = 1, 2, 3, 4, 5, 6$$

Gráf1

$$E = (1,2), (1,5), (2,3), (2,5), (3,4), (4,5), (4,6)$$

Python-ban a legegyszer?bben egy szótárban tárolhatjuk a gráfokat.

```
G = {'1' : ['2', '5'], Gráf2
      '2' : ['3', '5'],
      '3' : ['4'],
      '4' : ['5', '6']}
```

Gráfok

Ha irányítatlan gráfot szeretnénk, akkor mindkét irányítással vegyük fel az éleket!

Sage-ben létezik egy Graph osztály. A Graph objektumok segítségével irányítatlan gráfokkal dolgozhatunk.

```
sage: G = Graph({'1': ['2', '5'],
               '2': ['3', '5'],
               '3': ['4'],
               '4': ['5', '6']})
sage: G
Graph on 6 vertices
```

Bináris kereszfák

A bináris kereszfa (binary search tree, BST) olyan adatstruktúra, amely a következő tulajdonságokkal rendelkezik:

- - bármely csúcs alatti bal részfa csak a csúcsnál kisebb kulcsú elemeket tartalmaz
- - bármely csúcs alatti jobb részfa csak a csúcsnál nagyobb kulcsú elemeket tartalmaz
- - a bal és jobboldali részfa is bináris kereszfa

BST

Rekurzív algoritmus

A rekurzív algoritmusok három fontos összetevője:

- Alapeset: a legegyszerűbb, redukált eset, amire triviális a megoldás és visszatérhetünk vele
- Általános eset: ezt egyel egyszerűbb esetre kell visszavezetni
- Bizonyítani kell, hogy az általános esetből mindig el fogjuk érni valamelyik alapesetet (különben végtelen rekurzió lehet!)

Bináris kereszfa rekurzív bejárása.

Keressük meg, hogy van-e a fában 5 kulcsú elem!

- A gyökértől (legfelső elem) indulunk
- ha megtaláltuk a keresett kulcsot, visszatérünk (első alapeset)
- ha levélhez értünk, visszatérünk (második alapeset)
- ha a keresett kulcs kisebb, mint az aktuális csúcs értéke, akkor balra megyünk lefelé, ha nagyobb, akkor jobbra

BST

```
def search_bst(tree, node, key):
    if node == key:          # 1. alapeset: megvan
        return True
    if node not in tree:    # 2. alapeset: nincs
        return False
    # rekurzív hívás balra vagy jobbra
    (left, right) = tree[node]
    if node > key:
        return search_bst(tree, left, key)
    else:
        return search_bst(tree, right, key)
```

Egy klasszikus rekurzió példa

Fibonacci sorozat:

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

Feladat: írjunk Sage függvényt, ami visszaadja az n-edik Fibonacci-számot!

Megoldás: Rekurzióval a legegyszerűbb a kód

```
def fibo_r(n):
    if n==1 or n==2:
        return 1
    else:
        return fibo_r(n-1) + fibo_r(n-2)
```

Jobb megoldás:

a ciklusok hatékonyabbak! (Hosszabb kód)

```
def fibo_for(n):
    if n==1 or n==2:
        return 1
    a=1; b=1
    f=a+b
    for i in range(n-3):
        a=b
        b=f
        f=a+b
    return f
```

A két Fibonacci-megoldó függvény futásidejei:

Bemenet (n) For ciklussal Rekurzióval

| | | |
|--------|---------|--------------|
| 10 | 0.020s | 0.020s |
| 20 | 0.020s | 0.034s |
| 30 | 0.021s | 0.952s |
| 40 | 0.021s | 1m 56.8s |
| 4000 | 0.023s | kb. 1068 év |
| 400000 | 16.375s | kb. végtelen |

További olvasnivalók

(nem kell tudni, de hasznos és érdekes):

- Bináris keresés: http://en.wikipedia.org/wiki/Binary_search_tree
- Fibonacci számok: http://en.wikipedia.org/wiki/Fibonacci_number
- Algoritmusok futásidője: http://en.wikipedia.org/wiki/Time_complexity
http://en.wikipedia.org/wiki/Big_O_notation