STROM

Spájaná dátová štruktúra strom je zovšeobecnením spájaného zoznamu:
• skladá sa z množiny vrcholov, v každom vrchole sa nachádza nejaká informácia (atribút data)
• vrcholy sú navzájom pospájané hranami:
– jeden vrchol má špeciálne postavenie: je prvý a od neho sa vieme dostať ku všetkým zvyšným vrcholom
– každý vrchol okrem prvého má práve jedného predchodcu
– každý vrchol môže mať ľubovoľný počet nasledovníkov (nie iba jeden ako pri spájaných zoznamoch)

Zvykne sa používať takáto terminológia z botaniky:
• **koreň** (root) je špeciálny prvý vrchol - jediný nemá svojho predchodcu
• **vetva** (branch) je hrana (budeme ju kresliť šípkou) označujúca nasledovníka
• **list** (leaf) označuje vrchol, ktorý už nemá žiadneho nasledovníka

Okrem botanickej terminológie sa pri stromoch používa aj takáto terminológia rodinných vzťahov:
• predchodca každého vrcholu (okrem koreňa) sa nazýva otec, resp. rodič, predok (ancestor, parent)
• nasledovník vrcholu sa nazýva syn, resp. dieťa alebo potomok (child, descendant)
• vrcholy, ktoré majú spoločného predchodcu (otca) sa nazývajú súrodenci (siblings)

Ďalej sú dôležité ešte aj tieto pojmy:
• okrem listov sú v strome aj vnútorné vrcholy (interior nodes), to sú tie, ktoré majú aspoň jedného potomka
• medzi dvoma vrcholmi môžeme vytvárať cesty, t.j. postupnosti hrán, ktoré spájajú vrcholy - samozrejme, že len v smere od predkov k potomkom (v smere šípok) - dĺžkou cesty je počet týchto hrán (medzi otcom a synom je cesta dĺžky 1)
• úroveň, alebo hĺbka vrcholu (level, depth) je dĺžka cesty od koreňa k tomuto vrcholu, teda koreň má hĺbku 0,
jeho synovia majú hĺbku 1, ich synovia (vnuci) ležia na úrovni 2, atď.
• výška stromu (height) je maximálna úroveň v strome, t.j. dĺžka najdlhšej cesty od koreňa k listom
• podstrom (subtree) je časť stromu, ktorá začína v nejakom vrchole a obsahuje všetkých jeho potomkov (nielen synov, ale aj ich potomkov, ...)
• šírka stromu (width) je počet vrcholov v úrovni, v ktorej je najviac vrcholov

Niekedy sa všeobecný strom definuje aj takto rekurzívne:

- strom je buď prázdny, alebo
- obsahuje koreň a množinu podstromov, ktoré vychádzajú z tohto koreňa, pričom každý podstrom je opäť všeobecný strom (má svoj koreň a svoje podstromy)

Binárny strom

Aj binárny strom môžeme definovať rekurzívne:
• je buď prázdny,
• alebo sa skladá z koreňa a z dvoch jeho podstromov: z ľavého podstromu a pravého podstromu, tieto sú opäť binárne stromy

Počet vrcholov v jednotlivých úrovniach:
• v 0. úrovni môže byť len koreň, teda len 1 vrchol
• koreň môže mať dvoch synov, teda v 1. úrovni môžu byť maximálne 2 vrcholy
• v 2. úrovni môžu byť len synovia predchádzajúcich dvoch vrcholov v 1. úrovni, teda maximálne 4 vrcholy
• v každej ďalšej úrovni môže byť maximálne dvojnásobok predchádzajúcej, teda v i-tej úrovni môže byť maximálne 2**i vrcholov
• strom, ktorý má výšku h môže mať maximálne 1 + 2 + 4 + 8 + ... + 2**h vrcholov, teda spolu 2**(h+1)-1 vrcholov
– strom s maximálnym počtom vrcholov s hĺbkou h sa nazýva **úplný strom**

Binárny strom budeme realizovať podobne ako spájaný zoznam, t.j. najprv definujeme triedu Vrchol , v ktorej definujeme atribúty každého prvku binárneho stromu:

```
class Vrchol:
def __init__(self, data, left=None, right=None):
self.data = data
self.left = left
self.right = right
```

Na rozdiel od spájaného zoznamu, tento vrchol binárneho stromu má namiesto jedného nasledovníka next dvoch nasledovíkov: ľavého left a pravého right. Vytvorme najprv 3 izolované vrcholy:

```
a = Vrchol('A')
b = Vrchol('B')
c = Vrchol('C')
```

Tieto tri vrcholy môžeme chápať ako 3 jednovrcholové stromy. Pomocou priradení môžeme pripojiť stromy b a c ako ľavý a pravý podstrom a:

```
a.left = b
a.right = c
```

To isté sa dá zapísať ako:

```
a = Vrchol('A', Vrchol('B'), Vrchol('C'))
```

Vykreslenie stromu

Strom nakreslíme rekurzívne. Prvá verzia bez kreslenia čiar:

```
import tkinter
canvas = tkinter.Canvas(bg='white', width=400, height=400)
canvas.pack()

def kresli(v, sir, x, y):
    if v is None:
        return
    canvas.create_text(x, y, text=v.data)
    kresli(v.left, sir//2, x-sir//2, y+40)
    kresli(v.right, sir//2, x+sir//2, y+40)

kresli(strom, 200, 200, 40)
```

Prvým parametrom funkcie je koreň stromu (teda referencia na najvrchnejší vrchol stromu). Druhým parametrom je polovičná šírka grafickej plochy. Ďalšie dva parametre sú súradnicami, kde sa nakreslí koreň stromu. Všimnite si, že v rekurzívnom volaní:

- smerom vľavo budeme vykresľovať ľavý podstrom, preto x-ovú súradnicu znížime o polovičnú šírku plochy, y-ovú zvýšime o nejakú konštantu, napr. 40
- smerom vpravo budeme kresliť pravý podstrom a teda x-ovú súradnicu zväčšíme tak, aby bola v polovici šírky oblasti

Nakreslenie samotného vrcholu (bude to teraz farebný krúžok s textom) presťahujeme až na koniec funkcie, teda až potom, keď sa vykreslia čiary (vychádzajúce z tohto vrcholu aj vchádzajúce do tohto vrcholu):

```
import tkinter
canvas = tkinter.Canvas(bg='white', width=400, height=400)
canvas.pack()
def kresli(v, sir, x, y):
    if v is None:
        return
    if v.left is not None:
        canvas.create_line(x, y, x-sir//2, y+40)
        kresli(v.left, sir//2, x-sir//2, y+40)
    if v.right is not None:
        canvas.create_line(x, y, x+sir//2, y+40)
        kresli(v.right, sir//2, x+sir//2, y+40)
    canvas.create_oval(x-15, y-15, x+15, y+15, fill='lightgray', outline='')
    canvas.create_text(x, y, text=v.data, font='consolas 12 bold')
```

Ďalšie rekurzívne funkcie

Tieto funkcie postupne (rekurzívne) prechádzajú všetky vrcholy v strome: najprv v ľavom podstrome a potom aj v pravom.
Uvádzame dve verzie súčtu hodnôt vo vrcholoch. Prvá verzia, keď zistí, že je strom neprázdny, vypočíta súčet najprv v ľavom podstrome a potom aj v pravom.
Výsledok celej funkcie je potom súčet hodnoty v koreni stromu + súčet všetkých hodnôt v ľavom podstrome + súčet všetkých hodnôt v pravom podstrome:

```
def sucet(vrch):
    if vrch is None:
        return 0
    vlavo = sucet(vrch.left)
    vpravo = sucet(vrch.right)
    return vrch.data + vlavo + vpravo
print('sucet =', sucet(koren))
```

Na podobnom princípe pracuje aj zisťovanie celkového počtu vrcholov v strome:

```
def pocet(vrch):
    if vrch is None:
        return 0
    return 1 + pocet(vrch.left) + pocet(vrch.right)
print('pocet =', pocet(koren))
```

Funkcia najprv rieši situáciu, keď je strom prázdny: vtedy dáva výsledok 0. Inak sa vypočíta výška ľavého podstromu, potom výška pravého a na záver sa z týchto dvoch výšok vyberie tá väčšia a k tomu sa ešte pripočíta 1, lebo celkový strom okrem dvoch podstromov obsahuje aj koreň, ktorý je o úroveň vyššie.

```
def vyska(vrch):
    if vrch is None:
        return -1
    vyska_vlavo = vyska(vrch.left)
    vyska_vpravo = vyska(vrch.right)
return 1 + max(vyska_vlavo, vyska_vpravo)
```

Výpis hodnôt vo vrcholoch v nejakom poradí:

```
def vypis(vrch):
    if vrch is None:
        return
    print(vrch.data, end=' ')
    vypis(vrch.left)
    vypis(vrch.right)
 vypis(koren)
```

Trieda BinarnyStrom

Prirodzené a programátorsky správne je zadefinovanie triedy Strom, ktorá okrem definície jedného vrchol obsahuje aj všetky užitočné funkcie ako svoje metódy. Zapuzdrime všetky funkcie spolu s dátovou štruktúrou do jedného „obalu“. Pre prístup ku stromu (k jeho vrcholom) si musíme pamätať referenciu na jeho koreň - ten bude atribút self.root, ktorý bude mať pri inicializácii hodnotu None. Zapíšme základ:

```
class BinarnyStrom:
    class Vrchol:
        def __init__(self, data, left=None, right=None):
            self.data = data
            self.left = left
            self.right = right
        def __repr__(self):
            return repr(self.data)

    def __init__(self):
        self.root = None

    def __len__(self):
        def pocet(vrch):
            if vrch is None:
                return 0
        return 1 + pocet(vrch.left) + pocet(vrch.right)
    return pocet(self.root)

    def sucet(self):
        def sucet_rek(vrch):
            if vrch is None:
                return 0
            return vrch.data + sucet_rek(vrch.left) + sucet_rek(vrch.right)
    return sucet_rek(self.root)

    def vyska(self):
        def vyska_rek(vrch):
            if vrch is None:
                return -1
            return 1 + max(vyska_rek(vrch.left), vyska_rek(vrch.right))
        return vyska_rek(self.root)

    def __contains__(self, hodnota):
        def nachadza_sa(vrch):
            if vrch is None:
                return False
            return vrch.data==hodnota or nachadza_sa(vrch.left) or
                                    nachadza_sa(vrch.right)
    return nachadza_sa(self.root)
```

Prechádzanie vrcholov stromu

```
class Strom:
    def vypis_preorder(self):
        def vypis(vrch):
            if vrch is None:
                return
            print(vrch.data, end=' ')
            vypis(vrch.left)
            vypis(vrch.right)
    vypis(self.root)
    print()
```

Vytvorili sme metódu vypis_preorder(), ktorá v svojom tele opäť obsahuje len tri príkazy:

- definíciu vnorenej (lokálnej) funkcie vypis(),
- jej zavolanie s referenciou na koreň stromu
- a ukončenie vypisovaného riadka.

Metóda **preorder** (podobne ako skoro všetky doterajšie) obíde všetky vrcholy v strome tak, že najprv spracuje koreň stromu (vypíše ho príkazom print()), potom rekurzívne celý ľavý podstrom a na záver celý pravý podstrom. Takémuto poradiu hovoríme spracovanie preorder.

Okrem poradia preorder poznáme ešte tieto základné poradia obchádzania vrcholov stromu:
• **inorder** - najprv ľavý podstrom, potom samotný vrchol a na záver pravý podstrom
• **postorder** - najprv ľavý podstrom, potom pravý podstrom a na záver samotný vrchol
• **po úrovniach** - vrcholy sa spracovávajú v poradí ako sú vzdialené od koreňa, t.j. najprv koreň, potom obaja jeho synovia, potom všetci vnuci, atď.

Výpis prvkov stromu pomocou inorder a postgorder:

```
def vypis_inorder(self):
    def vypis(vrch):
        if vrch is None:
            return
        vypis(vrch.left)
        # spracuj samotný vrchol vrch
        print(vrch.data, end=' ')
        vypis(vrch.right)

    vypis(self.root)
    print()
```

```
def vypis_postorder(self):
    def vypis(vrch):
        if vrch is None:
            return
        vypis(vrch.left)
        vypis(vrch.right)
        # spracuj samotný vrchol vrch
        print(vrch.data, end=' ')

    vypis(self.root)
    print()
```

Binárny vyhľadávací strom

V niektorých situáciách je vhodné ukladať nové hodnoty do stromu tak, aby sme ich neskôr vedeli čo najrýchlejšie nájsť. Jedna z možnností je ukladať ich tak, aby sme sa na základe hodnoty v koreni vedeli jednoznačne rozhodnúť, či pokračujeme v hľadaní do ľavého alebo pravého podstromu. Ak by sme takéto pravidlo zabezpečili pre všetky vrcholy v strome, hľadanie by bolo jednoduché: kým nenájdeme, resp. už sa ďalej pokračovať nedá (prišli sme na None), budeme sa vnárať sa do ľavej alebo pravej strany stále hlbšie a hlbšie.
Takto organizovaný binárny strom sa nazýva binárny vyhľadávací strom, skrátene BVS (binary search tree http://en.wikipedia.org/wiki/Binary_search_tree) a má tieto vlastnosti (rekurzívna definícia):
• všetky hodnoty v ľavom podstrome sú menšie ako hodnota v koreni
• všetky hodnoty v pravom podstrome sú väčšie ako hodnota v koreni
• všetky podstromy sú tiež BVS, t.j. aj pre ne platí, že vľavo sú iba menšie hodnoty a vpravo iba väčšie, atď.
V takomto strome sa rovnaká hodnota nemôže nachádzať v rôznych vrcholoch (v strome musia byť všetky hodnoty rôzne) - ak by sme to niekedy potrebovali, treba to zabezpečiť nejako inak.

Do **existujúceho BVS** musíme nové vrcholy **vkladať podľa presných pravidiel**:
• ak sa vkladaná hodnota už nachádza v koreni stromu, môžeme skončiť
• ak je vkladaná hodnota menšia ako hodnota v koreni, zrejme budem vkladať do ľavého podstromu (rekurzívne voláme vkladanie pre ľavý podstrom)
• ak je vkladaná hodnota väčšia ako hodnota v koreni, zrejme budem vkladať do pravého podstromu (rekurzívne voláme vkladanie pre pravý podstrom)
• ak takýto podstrom ešte neexistuje a my sme sa do neho chceli rekurzívne vnoriť, tak na jeho mieste vytvoríme nový vrchol aj s vkladanou hodnotou (podstrom s jedným vrcholom).

```
def pridaj(self, hodnota):
```

```
if self.root is None:
    self.root = self.Vrchol(hodnota)
else:
    vrch = self.root
    while vrch.data != hodnota:
        if vrch.data > hodnota:
            if vrch.left is not None:
                vrch = vrch.left
            else:
                vrch.left = self.Vrchol(hodnota)
        else:
            if vrch.right is None:
                vrch.right = self.Vrchol(hodnota)
            else:
                vrch = vrch.right
```