BINÁRNY VYHĽADÁVACÍ STROM

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.
Vidíme, že 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** je potrebné nové vrcholy vkladať podľa **presných pravidiel**:
• ak sa vkladaná hodnota už nachádza v koreni stromu, treba skončiť
• ak je vkladaná hodnota menšia ako hodnota v koreni, treba dáta vkladať do ľavého podstromu (rekurzívne voláme vkladanie pre ľavý podstrom)
• ak je vkladaná hodnota väčšia ako hodnota v koreni, treba 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
```

Vyhľadávanie v BVS

Hľadanie hodnoty v BVS (napr. metódou je_prvkom()) bude analogické vkladaniu novej hodnoty.

```
class BinarnyVyhladavaciStrom:
...
    def je_prvkom(self, hodnota):
        vrch = self.root
        while vrch is not None:
            if hodnota == vrch.data:
                return True
            if hodnota < vrch.data:
                vrch = vrch.left
            else:
                vrch = vrch.right
        return False
```

Vypisovanie hodnôt v BVS

Binárny vyhľadávací strom má jednu veľmi dôležitú vlastnosť, keď navštívime vrcholy v poradí inorder , resp. vytvoríme pole s hodnotami v poradí inorder, dostaneme usporiadanú postupnosť všetkých hodnôt.

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

        def __init__(self):
            self.root = None

     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

    def kresli(self):
        def kresli_rek(vrch, sir, x, y):
            if vrch.left is not None:
                    self.canvas.create_line(x, y, x-sir//2, y+40)
                    kresli_rek(vrch.left, sir//2, x-sir//2, y+40)
            if vrch.right is not None:
                    self.canvas.create_line(x, y, x+sir//2, y+40)
                    kresli_rek(vrch.right, sir//2, x+sir//2, y+40)
                    #self.canvas.create_oval(x-15, y-15, x+15, y+15, fill='lightgray
                    , outline='')
                    self.canvas.create_oval(x-15, y-15, x+15, y+15, fill='white')
                    #self.canvas.create_text(x, y, text=vrch.data)
            if self.canvas is None:
                    BinarnyVyhladavaciStrom.canvas = tkinter.Canvas(bg='white',
                    width=800, height=400)
                    self.canvas.pack()
                    self.canvas.delete('all')
        kresli_rek(self.root, 380, 400, 40)

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

    def preorder(self):
        def preorder_rek(vrch):
            if vrch is None:
                    return ''
        return repr(vrch.data) + ' ' + preorder_rek(vrch.left) +
        preorder_rek(vrch.right)
    return preorder_rek(self.root)

    def postorder(self):
        def postorder_rek(vrch):
            if vrch is None:
                    return ''
            return postorder_rek(vrch.left) + postorder_rek(vrch.right) +
            repr(vrch.data) + ' '
        return postorder_rek(self.root)
```

```
    def inorder(self):
        def inorder_rek(vrch):
            if vrch is None:
                return ''
            return inorder_rek(vrch.left) + repr(vrch.data) + ' ' +
            inorder_rek(vrch.right)
        return inorder_rek(self.root)

    def je_prvkom(self, hodnota):
        vrch = self.root
        while vrch is not None:
            if hodnota == vrch.data:
        return True
        if hodnota < vrch.data:
            vrch = vrch.left
        else:
            vrch = vrch.right
        return False
__contains__ = je_prvkom
```

Vyhodenie uzla z BVS

Pri uvažovaní o vlastnostiach a algoritmoch binárnych vyhľadávacích stromov sa často spomína aj vyhodenie niektorého vrcholu tak, aby strom ostal BVS a pritom sme ho nemuseli celý prerábať.

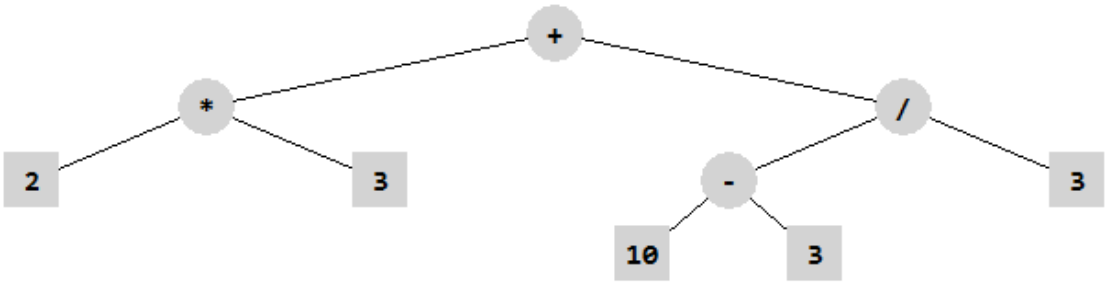
Základnou ideou vyhodenia uzla z BVS je:
• nájsť vyhadzovaný vrchol a ak je to list, jednoducho sa vyhodí (nahradiť None),
• ak je to vnútorný vrchol s jediným synom, tak tento vrchol môžeme vyhodiť tiež a namiesto neho otcovi nastavíme podstrom s týmto jedným existujúcim synom,
• ak je to vrchol, ktorý má oboch synov, tak v podstrome ľavého syna (tam sú všetci menší, ako vyhadzovaný vrchol) nájdeme vrchol s najväčšou hodnotou (stačí ísť v podstrome stále vpravo, kým sa dá): tento vrchol už určite nemá pravého syna (inak by nebol najväčší), tak ho vyhodíme a jeho hodnotu dáme namiesto pôvodne vyhadzovaného vrcholu.

ARITMETICKÝ STROM

Ďalším príkladom využitia binárnych stromov, sú tzv. aritmetické stromy.
Môžete ich nájsť aj pod názvom binary expression tree na stránke (http://en.wikipedia.org/wiki/Binary_expression_tree).

Základné princípy aritmetických stromov:

- vo vnútorných vrcholoch znamienka operácií (napr. ‘+’ , ‘- ‘, ‘*’ , ‘/’ , ...) a v listoch sú operandy, napr. celé čísla alebo mená premenných
- takýto strom jednoznačne popisuje štandardný aritmetický výraz
- vyhodnocovanie takéhoto aritmetického výrazu (teda binárneho stromu) je jednoduché - použitím rekurzívnej vyhodnocovacej funkcie



Aritmetický strom zodpovedá výrazu 2 * 3 + (10 - 3) / 3.

```
import tkinter
class AritmetickyStrom:
    canvas = None
    class Vrchol:
        def __init__(self, data, left=None, right=None):
            self.data = data
            self.left = left
            self.right = right
        def __init__(self, root=None):
                self.root = root

    def kresli(self):
        def kresli_rek(vrch, sir, x, y):
            if vrch.left is not None:
                self.canvas.create_line(x, y, x-sir//2, y+40)
            kresli_rek(vrch.left, sir//2, x-sir//2, y+40)
            if vrch.right is not None:
                self.canvas.create_line(x, y, x+sir//2, y+40)
            kresli_rek(vrch.right, sir//2, x+sir//2, y+40)
            if vrch.left is None and vrch.right is None:
                self.canvas.create_rectangle(x-15, y-15, x+15, y+15, fill=
                'lightgray', outline='')
            else:
```

```
                self.canvas.create_oval(x-15, y-15, x+15, y+15, fill=
                'lightgray', outline='')
            self.canvas.create_text(x, y, text=vrch.data, font='consolas 12 bold')

        if self.canvas is None:
            AritmetickyStrom.canvas = tkinter.Canvas(bg='white', width=800,
                height=400)
            self.canvas.pack()
        self.canvas.delete('all')
        kresli_rek(self.root, 380, 400, 40)

    def preorder(self):
        def preorder_rek(vrch):
            if vrch is None:
                return ''
            return str(vrch.data) + ' ' + preorder_rek(vrch.left) +
            preorder_rek(vrch.right)
        return preorder_rek(self.root)

    def postorder(self):
        def postorder_rek(vrch):
            if vrch is None:
                return ''
            return postorder_rek(vrch.left) + postorder_rek(vrch.right) +
            str(vrch.data) + ' '
        return postorder_rek(self.root)

    def inorder(self):
        def inorder_rek(vrch):
            if vrch is None:
                return ''
            return inorder_rek(vrch.left) + str(vrch.data) + ' ' +
            inorder_rek(vrch.right)
        return inorder_rek(self.root)
```

Teraz sa nebudeme zaoberať algoritmami na vytváranie takýchto stromov (sú podobné algoritmom na prevody infixu a postfixu). Do inicializácie __init__() sme pridali parameter root, vďaka čomu môžeme jednoducho vytvoriť celý strom ‘ručne’. Pomôžeme si funkciou *v()* na definovanie jedného vrcholu stromu:

```
v = AritmetickyStrom.Vrchol
strom = AritmetickyStrom(v('+', v('*', v(2), v(3)), v('/', v('-', v(10),v(3)), v(3))))
strom.kresli()
```

Metóda hodnota(), ktorá vypočíta hodnotu aritmetického výrazu, najprv zistí, či je vrchol list (vtedy je to operand, teda číslo), alebo vnútorný vrchol (je to operácia). Pre operáciu rekurzívne spustí vyhodnocovanie pre ľavý aj pravý podstrom a oba tieto medzivýsledky spojí príslušnou operáciou:

```
def hodnota(self):
    def hodnota_rek(vrch):
        if vrch is None:
            return 0
        if vrch.left is None and vrch.right is None:
            return vrch.data
        vlavo = hodnota_rek(vrch.left)
        vpravo = hodnota_rek(vrch.right)
        if vrch.data == '+':
            return vlavo + vpravo
        if vrch.data == '*':
            return vlavo * vpravo
        if vrch.data == '-':
            return vlavo - vpravo
        if vrch.data == '/':
            return vlavo / vpravo
    return hodnota_rek(self.root)
```