Dynamické datové struktury

a jejich aplikace při programování

- Na algoritmizaci byly spojové seznamy implementované pomocí objektů,
- stromy a algoritmy pro práci s nimi.
- V různých programovacích jazycích jsou předpřipravena řešení,
- my si však ukážeme, jak to implementovat samostatně, abychom věděli, co od předpřipravených řešení očekávat.

Datové struktury

a typické operace nad nimi

- Typické operace jsou `init`, `member`, `insert`, `delete`,
- některé struktury jsou ale navržené na jiné operace,
- kupř. `extract_min`, `merge`, ...
- Někde se dobře známé operace jmenují jinak:
- `push`, `pop`, `enqueue`, `dequeue`, ...

Zásobník

a jeho předpřipravená implementace:

```
class zasobnik:
    def init(this):
        this.vrchol=[]
    def __init__(this):
        init()
    def push(this,co):#a.k.a. insert
        this.vrchol.append(co)
    def pop(this):
        return this.vrchol.pop()
```

Spojové seznamy

které si samostatně implementujeme

```
class vagon:
    def __init__(this,hod=0,next=None):
        this.hod=hod
        this.next=next
```

Takto bude vypadat jeden prvek spojového seznamu. Má hodnotu a následníka.

Samotné funkce

spoják obhospodařující

```
class spojak:
    def __init__(this):
        this.zac=None
        this.kon=None
    def insertz(this,co):
        if(this.zac==None):
            this.zac=vagon(co)
            this.kon=zac
        else:
            this.zac=vagon(co,this.zac)
```

insertk

bude také jednoduchá

```
def insertk(this,co):
    if(this.zac!=None):
        this.kon.next=vagon(co)
        this.kon=this.kon.next
    else: insertz(this,co)
```

delete

půjde jen ze začátku

```
def delete(this):
    if(this.zac==None):
        return None
    else:
        navrhod=this.zac.hod
        this.zac=this.zac.next
        if(this.zac==None):
            this.kon=None
```

Typologie spojových seznamů

alias zvěřinec a jeho využití

- Jak odebírat z konce? Těžko. $\Rightarrow$
- Obousměrný spojový seznam (prvky ukazují i na předchůdce),
- cyklický (z konce ukazuje next na začátek),
- s hlavou (v prvním prvku není hodnota $\Rightarrow$ nenarazíme tam na None),
- s ocasem ("hlava"na konci) – lze využít jako zarážku.
- Hledání se zarážkou: Na konec dáme, co hledáme (pak se nemusíme bát, že to nenajdeme).

K AVL-stromům

ať se dozvíme něco navíc oproti Algoritmizaci

- Toto více méně bylo na Algoritmizaci,
- spojové seznamy umožňují vyhledávat unárně. My chceme vyhledávat binárně.
- Definujeme binární vyhledávací stromy.
- Datová struktura osazena dvěma ukazateli, pro každý vrchol platí, že hodnoty v levém podstromě jsou menší, v pravém větší (než současná hodnota).
- Pro pohodlí můžeme přidat i ukazatel na rodiče.
- Binární vyhledávací strom je vhodné udržovat vyvážený.
- Možností je několik, velmi jednoduché jsou AVL-stromy,
- `https://gist.github.com/girish3/a8e3931154af4da89995`

Jak ty stromy fungují?

V té implementaci se bez výkladu nelze orientovat

- AVL-strom je binární vyhledávací strom, kde pro každý vrchol se hloubka jeho podstromů liší nejvýše o 1.
- Definujeme balanci jako hloubu pravého syna - hloubka levého syna.
- Vkládáme a mažeme jako u normálního BVS (vkládáme tam, kde prvek chybí, mazat musíme tam, kde prvek je),
- následně případně provedeme vyvažovací operace,
- těmi jsou tzv. rotace a dvojrotace.

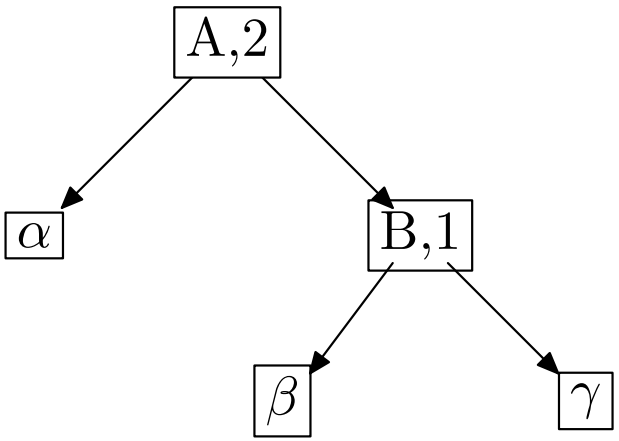
Náznak implementace BVS

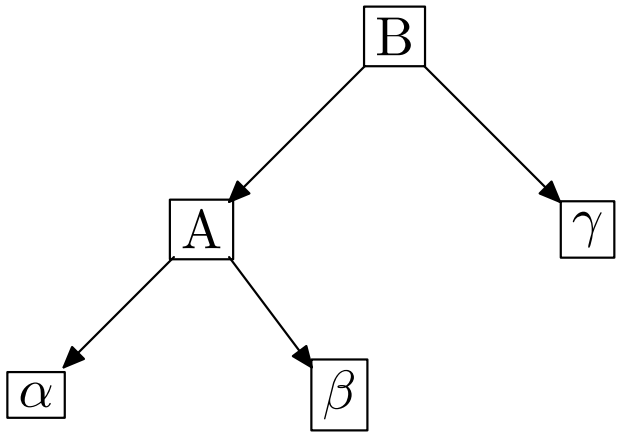
s funkcí member

```
class vagon:
    def __init__(this,hod,left=None,right=None):
        this.hod,this.left,this.right=hod,left,right
class avl:
    def __init__(this):
        koren=None
    def member(this,hod):
        pom=this          while((pom!=None)
and(pom.hod!=hod)):
            if(hod<pom.hod): pom=pom.left
            else: pom=pom.right
        return pom
```

Rotace

a její fungování

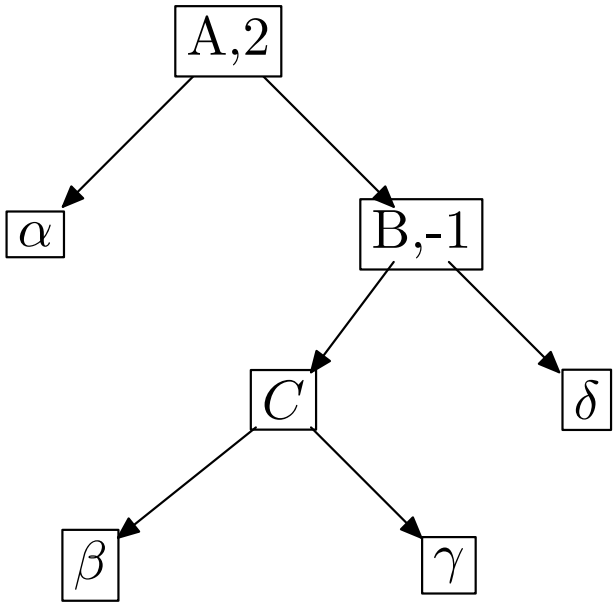


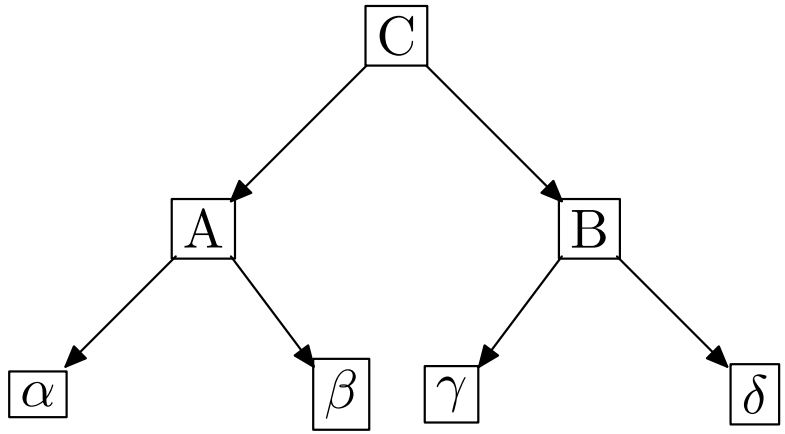


Rotace funguje, kácí-li se strom ke kraji (tedy balance syna vrcholu s balancí 2 nemá jiné znaménko než rodič)

Dvojrotace

Strom se kácí "dovnitř":





Další rotace a dvojrotace jsou zrcadlově obrácené (pro balanci -2).

Jak rotace implementovat?

Až nečekaně snadno!

```
def rotujr(ktery):
    a=ktery
    b=a.right
    r=a.rodic
    alfa=a.left
    beta=b.left
    gamma=b.right
    if(a==r.left): b=r.left
    else: b=r.right
    a.right=beta
    b.left=a
```

Mravní naučení

rotaci lze zoptimalizovat

- Naše implementace je didaktická,
- ve skutečnosti proměnné `alfa` a `gamma` nejsou nutné,
- protože se s nimi nepracuje!

Rekurze

a její aplikace při programování

- Další položka z Algoritmizace cvičená u nás,
- jde o zamaskovanou indukci,
- problém řešíme tak, že instanci zmenšíme,
- najdeme řešení menších instancí (rekurzívně)
- z řešení menších instancí zjistíme řešení větší instance.

Typické příklady

uváděné nyní na Algoritmizaci

- Fibonacciho čísla (Přednášející jde do posluchárny, Mrkev a petržel,...),
- platná uzávorkování (nesmíme zavřít více závorek, než jsme otevřeli) a.k.a. Dyckovy cesty,...,
- rozklad čísla na součet posloupnosti nerostoucích čísel (a.k.a. Youngovy tabulky, Ferrersovy diagramy,...),
- kam kůň (král a další harampádí na šachovnici) nemůže (aneb rekurze v dostupném programovacím jazyce),
- vyhodnocení výrazu v prefixní či infixní notaci.
- Motto:
  - R. Kryl: Rekurze je dobře navržena, pokud žádná část programu sama o sobě nedělá nic a dohromady to funguje.
  - Já: Rekurze obvykle vypadá tak, že napřed neuděláme nic, pak uděláme to samé a nakonec to udělá, co chceme.

Rekurze

mravní naučení

- Při práci s rekurzí je důležité říct **co** zavolání funkce (se zadanými parametry) udělá (ne jak – to budeme řešit pak).

Evaluace prefixní notace

aneb čistá vydestilovaná rekurze bez parametru

- Jak vyhodnotit výraz v prefixní notaci?
- Co zavolání funkce udělá: Zajistí vyhodnocení výrazu v prefixní notaci začínající zde (tam, kde je zrovna vstup).
- Základní krok: číslo.
- Indukční krok: Operátor.
- Je-li na vstupu číslo, vyhodnotíme ho (Hornerovo schéma).
- Jinak přečteme operátor a necháme mu vyhodnotit dva operandy (na to již máme odborníka – sebe).

Vyhodnocení

naznačeno, detaily (mezery) se na slide nevejdou!

```
import sys
def evalpref():
    znak=sys.stdin.read(1)
    if((ord(znak)>=48) and (ord(znak)<=57)):
        return vyhodnotcislo(znak)
    else:
        a,b=evalpref(),evalpref()
        if(znak=='+'): return a+b
        ...
```

Pořádný příklad

Vyhodnocení výrazu v infixní notaci

Zdrojáky zvlášť, na slide se nevejdou.
Algoritmus: Najdeme poslední operátor, který se provede a schramstneme ho. O zbytek ať se postará někdo jiný (ve skutečnosti opět my).

Prohledávání

taktéž bylo na Algoritmizaci

- Další z možných aplikací rekurze,
- alias *kam kráľ nemuže,* alias Thesseus hledá Minotaura,...
- Dostaneme startovní pole označíme sousedy. Ty ať prohledá někdo, kdo to umí!
- Alternativa: Sousedy dáme na zásobník.
- Anebo do fronty....
- Anebo do prioritní fronty,...