Anotace

- Stromové datové struktury:
  - Vyvážené binární stromy
  - AVL-stromy,
  - červeno-černé stromy,
  - A-B stromy,
  - hashování,
  - haldy.

Reprezentace stromů

- Strom je (v této chvíli) datová struktura, ve které má každý vrchol nějaký počet synů a nejvýše jednoho rodiče.
- Vrchol bez rodiče je jen jeden a označujeme ho kořen.
- Ve stromě nesmějí vznikat cykly (ani slabé).
- Jak reprezentovat v Pascalu?
- Podobně jako spojový seznam, třeba takto:

```
type  strom:^vrchol;
      vrchol=record
               hod:longint;
               syn1:strom;
               syn2:strom;
               ...
      end;
```

Binární vyhledávací strom

- Jedná se o binární strom, ve kterém pro každý vrchol všechny prvky v levém podstromě mají klíč menší než je klíč tohoto vrcholu a prvky v pravém podstromě mají klíč větší (než současný vrchol).
- V tomto stromě půjde snadno vyhledávat.
  Co výhody a nevýhody?
- Pokud se dobře postaví, je mnohem efektivnější, než spojový seznam.
- Problém je nepostavit ho špatně a při přidávání a ubírání nestrávit příliš času.
- Vyváženost odkazuje k tomu, jak moc se liší počty prvků v jednotlivých podstromech. Počty prvků v synech vyváženého stromu se liší nejvýše o 1.

Postavení vyváženého BVS

- Najdi medián a zakořeň.
- Postav vyvážený BVS z menších prvků (rekurzívně),
- připoj jako levého syna kořene,
- postav vyvážený BVS z větších prvků,
- připoj jako pravého syna kořene.

BVS – datové struktury

- Pole, ze kterého budeme stavět (nebudeme řešit).
- Dynamická struktura reprezentující vrcholy stromu:

```
type pbvs:^bvs;
     bvs=record
          hod:longint;
          left:pbvs;
          right:pbvs;
```

Postavení vyváženého BVS

(pseudokód)

```
function postav(pole):pbvs;
begin
      if empty(pole) then postav:=nil; else begin
            med:=median(pole);
            men:=mensi(med,pole);
            vet:=vetsi(med,pole);
            new(koren);
            koren^.hod:=med;
            koren^.left:=postav(mensi);
            koren^.right:=postav(vetsi);
            postav:=koren;
      end;
end;
```

Další operace nad vyváženým BVS

`member, insert, delete`

- Operace `member` je snadná:

```
function member(co:longint,kde:pbvs):pbvs;
begin if kde=nil then member:=nil
      else  if kde^.hod=co then member:=kde
            else  if kde^.hod>co then
                        member:=member(kde^.left)
                  else  member:=member(kde^.right);
end;
```

- Pozor, algoritmus spekuluje na postranní efekt (trichotomie), tedy číslo je větší, menší nebo rovné jinému. U posledního `else` bychom měli udělat test, zda `kde^.hod>co`.
- Funkce `insert` a `delete` jsou téměř neimplementovatelné (byly by spojené s destrukcí celého stromu).

Binární vyhledávací strom – zdaleka ne vyvážený!

```
procedure insert(co,kam);
begin {Okrajové případy!}
      while(( co<kam^.hod) and (kam^.left<>nil)) or
            ((co>kam^.hod)and (kam^.right<>nil)))
            if(co<kam^.hod) then kam:=kam^.left
            else  kam:=kam^.right;
      if(co=kam^.hod) then error("Uz tam je!");
      if(co<kam^.hod) then
      begin new(kam^.left);
            kam:=kam^.left;
      end else symetricky pro pravy...
      kam^.left:=nil; kam^.right:=nil;
      kam^.hod:=co;
end;
```

BVS – delete – špatná varianta

- Najdi prvek,
- je-li výstupního stupně nejvýš 1, vyhoď ho (přemosti).
- Je-li výstupního stupně 2,
  přidej jeho levého syna jako levého syna nejlevějšího prvku v pravém podstromě,
  tím se mazaný prvek začne tvářit jako vrchol výstupního stupně 1 $\Rightarrow$ vyhoď ho (přemosti).
- Co je špatně?
- Strom se rychle rozsype (změní v něco blízkého spojovému seznamu).

BVS – delete – správná varianta

- Najdi prvek,
- je-li výstupního stupně nejvýš 1, vyhoď ho (přemosti).
- Jinak najdi nejlevější prvek v pravém podstromě a dočasně prohoď.
- Tím se poruší vlastnost být BVS, ale chvíli nám to nevadí!
- Nyní máme zrušit vrchol výstupního stupně nejvýš 1 $\Rightarrow$
- vyhoď ho (přemosti).
- Místo nejlevějšího v pravém podstromě můžeme použít nejpravější v levém podstromě. Proč? Protože je to prvek s hodnotou nejbližší vyhazovanému (a ještě k tomu s příjemnými vlastnostmi).

Vyváženost

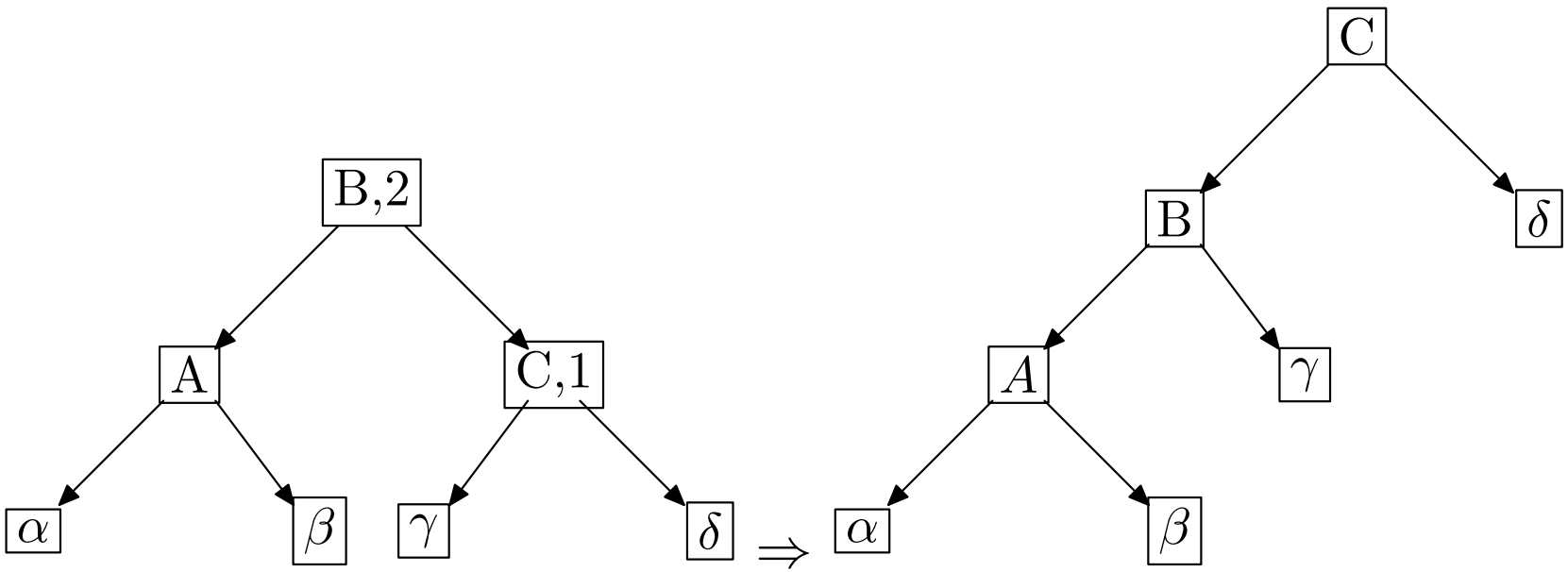
- Udržovat strom vyvážený je těžké, proto se spokojíme s něčím horším.
- AVL-strom je binární vyhledávací strom, ve kterém se pro každý vrchol hloubka jeho levého a pravého syna liší nejvýš o jedna.
- AVL – Adelson-Velskij a Landis.
- Operace `member`, `insert` a `delete` jako pro BVS, ale
- po `insert` a `delete` uděláme vyvažovací operace.
- Pro každý vrchol definujme hodnotu `balance`, která bude mít hodnotu 0 pokud jsou oba syny stejně hluboké,
  1 pokud levý syn má hloubku o 1 menší než pravý,
  -1 pokud levý syn má hloubku o 1 větší než pravý.

Způsob údržby AVL-stromů – rotace

- Problém začneme řešit tam, kde balance dosáhne BÚNO 2,
- probereme jen dvě varianty, ostatní jsou symetrické.
- Strom se hroutí buďto "ke straně" (prohloubili jsme BÚNO pravého vnuka pravého syna),
- nebo "dovnitř" (prohloubili jsme levého vnuka pravého syna).
- V prvním případě nasadíme rotaci, ve druhém dvojrotaci.

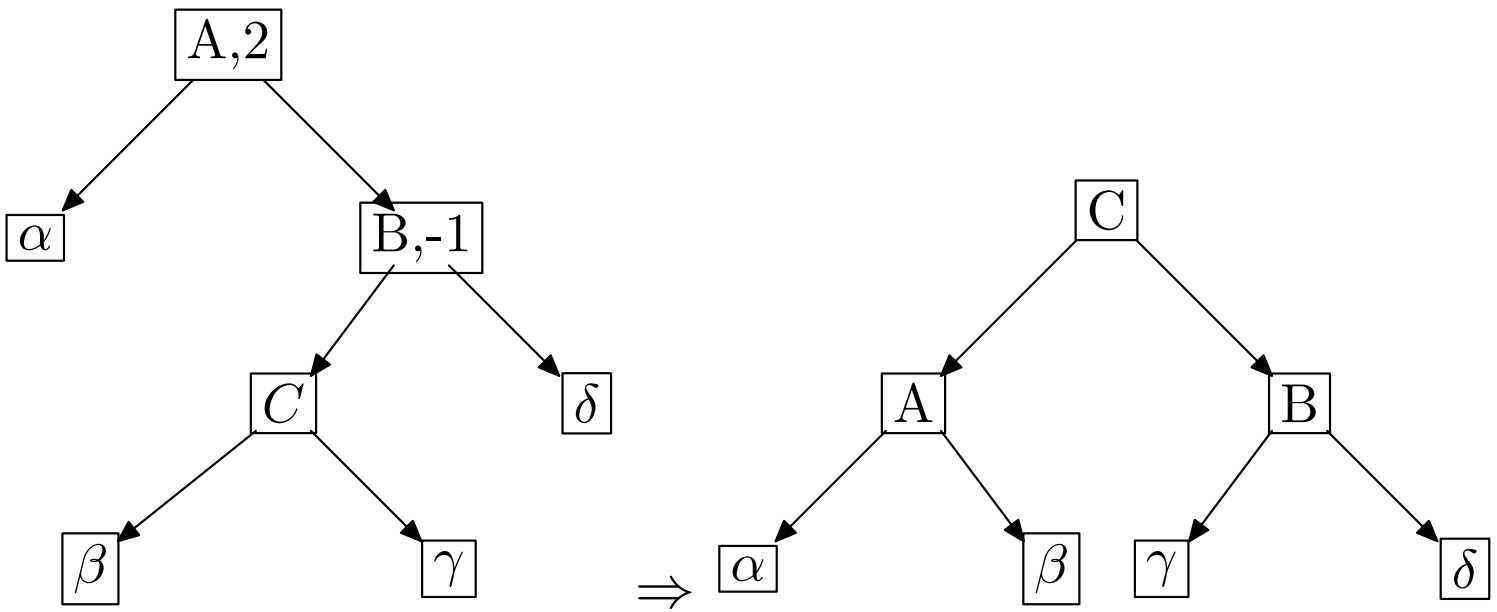
Rotace

Strom se hroutí "ke straně".



Dvojrotace

Strom se hroutí "dovnitř".



Analýza a poznámky

rotace, dvojrotace, hloubky

- Při přidání prvku stačí jedna rotace (resp. dvojrotace).
- Při ubírání prvku můžeme potřebovat kaskádu rotací, tedy testy musíme spouštět od místa, kde nastala porucha, až ke kořeni.
- Velikost (počet prvků) stromu hloubky n:
- Hloubka synů se liší nejvýš o jedna, tedy:
  $T(n) \geq T(n-1) + T(n-2)$,
- tedy počet prvků jsou alespoň Fibonacciho čísla (tedy exponenciální v n),
- tedy hloubka AVL-stromu je logaritmická oproti počtu prvků

Červeno-černé stromy

- Jiná metoda jak udržet strom dostatečně košatý.
- Každému vrcholu přidělíme červenou nebo černou barvu.
- Červené vrcholy nesmějí být dva za sebou,
- černých musí být na cestách z každého vrcholu do všech jeho listů stejně.
- Tudíž jeden podstrom musí mít hloubku nejvýše dvakrát větší než druhý.
- K údržbě se používají rotace, dvojité rotace a přebarvování.
- Pravidla jsou velmi komplikovaná.
- Hloubka červeno-černého stromu je též logaritmická vůči počtu vrcholů.

A-B Stromy

úplně jiný systém správy stromu

- A-B-strom není binární, ale B-ární (tedy každý vrchol má nejvýš B synů)
- V A-B-stromu je každý vrchol buďto list, nebo kořen, nebo má alespoň A a nejvýše B synů. List, který není kořen, obsahuje aspoň $A - 1$ a nejvýše $B - 1$ hodnot.
- Parametry A a B jsou přirozená čísla. Aby bylo možno A-B-strom postavit, je třeba, aby $B \geq 2A - 1$.
- V každém vrcholu je klíč vždy mezi dvěma pointery a stále platí vlastnost vyhledávacího stromu.
- A-B-strom obhospodařujeme tak, že je-li vrchol přeplněný, rozštípneme ho na dva a medián pošleme do otce.
- Tímto přesunem "do otce" můžeme spustit kaskádu.

Vkládání a ubírání z A-B-stromu

- A-B-strom je vyhledávací strom, tudíž poloha prvku je určena hodnotami prvků v rodičích.
- Při vkládání najdeme list, do kterého by prvek patřil. Pokud se prvek vejde, přidáme ho. Pokud ne, vrchol rozdělíme na dva a medián převedeme do rodiče (jako pivot).
- Štěpení může vyvolat kaskádu vedoucí až ke kořeni (včetně).
- Při rušení prvku problém převedeme na rušení listového vrcholu (ekvivalent "najdi nejlevější v pravém podstromě").
- Pokud vrchol začne být podkritický, buďto si "půjčíme" od bratra, pokud je podkritický i bratr, vrcholy sloučíme (a přidáme pivot z rodiče mezi nimi).

Analýza a poznámky

- Všechny listy jsou ve stejné hloubce.
- A-B-strom "obsahuje" vyvážený binární strom (tedy lze z něj takový vybrat), proto je hloubka logaritmická vůči počtu prvků.
- Přidáváme i ubíráme v logaritmickém čase.
- Typický příklad je 2-3 strom (každý vrchol obsahuje 1 nebo 2 prvky).
- Nepříjemná implementace (je třeba prohledávat více prvků ve vrcholu, evidovat počty prvků ve vrcholu).
- A-B-stromy jsou prakticky často používány.
- Pokud je $A = \lfloor \frac{B+1}{2} \rfloor$, hovoříme o B-stromech.
- Existují různé modifikace (zvané B+ strom, B*, někdy na listech uděláme spojový seznam, jindy posilujeme "výměny se sourozenci"). Prakticky jsou zajímavé, teoreticky až tolik ne.

A-sort

- *A*-sort: Třídění pomocí *B*-stromu s prstem.
- Prst ukazuje na vrchol *B*-stromu, se kterým jsme pracovali jako s posledním.
- Vkládat nezačínáme od kořene, ale od "prstu".
- Dobré výsledky pokud je vstup předtříděný (nebubláme často až do kořene).