BVS - uzel má dva syny
levy podstrom obsahuje mensi nez klic
pravy podstrom vetsi

máme více případů ale zase jen max 3 rotace (delete)

haldy - stromová struktura kde platí:
otec≥syn (max-heap)
otec≤syn (min-heap)

binární haldy

Implemetace v poli: kořen na pozici 0 potomci vždy na 2n+1 a 2n+2 rodiče na ceil(n/2)-1

Pravidla: je perfektně vyváženým binárním stromem, nebo pokud je poslední úroveň stromu nekompletní, uzly plní strom zleva doprava.

Hloubka: zřejmě max (log n)+1

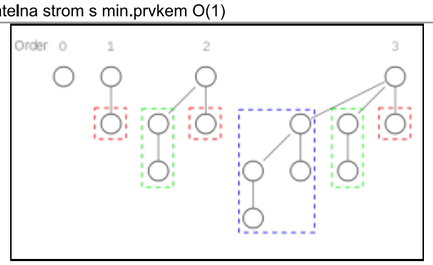
Insert — **přidám na konec haldy** a když nesedí prohazujeme s rodičem (tedy **posouvám nahoru**) O(log n)

Delete — odeberu, **nahradím posledním prvkem** a pokud nesedí prohazuji s potomky kteří porušují vlastnost víc (tedy **posouvám dolů**) O(log n)

Vytvoření haldy z n prvků — dá se v O(n)

binomiální haldy

Pravidla: les binomiálních stromů - každý řád max.jednou, bim.strom řádu i se skládá z kořene a i synů ze stromů 0 až i-1 - chová se jako malá halda
-používá se ukazatelna strom s min.prvkem O(1)



umožňuje rychlou implemetaci Insert O(log n) (amotizovaně O(1)) a Merge (dvou hald) O(log n)

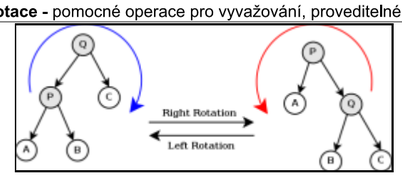
fibonacciho haldy

Pravidla: les haldových stromů, každý uzel haldy s n prvky má podstrom velikosti min. F_n+2 má stupeň max log n
-při odebírání prvků lze oddelit z nekořenového prvku max 1 syna, jinak se oddeli cely uzel do noveho stromu
-při odebrání minima se počet stromů naopak snižuje - spojují se

vychází z binomiálních

Insert je amortizovaně O(1) Delete O(log n)

(pravá/levá) rotace - pomocné operace pro vyvažování, proveditelné v konst.čase



Pravidla: - ∀uzel platí: **výška** jeho **levého a pravého podstromu se liší nejvýše o 1**, uchováváme si v uzlu o tom info {-1,0,1}

AVL stromy

Hloubka: a0=0,a1=1,a2=2... pro k ≥ 2:
(min.počet vrcholů v hloubce k)=$a_k=1+a_{n-1}+a_{k-2} > 2^{k-1}/2+2^{k-2}/2 = 2^k/2\,(2^{-1/2}+2^{-1}) > 2^k/2$ na každé hladině je min. exponenciálně vrcholů => hloubka O(log n)

Insert (max 2 rot.)
- postupujeme od nově přidaného uzlu směrem nahoru a cestou opravujeme balance uzlů podle hloubky podstromů
- pokud se balance uzlu změnila na 2 nebo −2 (silně nevyvážený vrchol) - > je nutná reorganizace stromu … operace rotace (LR a RL rotace jde brat jako jednu)
- zrotovaný podstrom má stejnou výšku jako původní, takže není potřeba postupovat dále nahoru ke kořeni stromu (**tzn. rotace 2x a dost**)
- časová složitost je nejvýše rovna výšce stromu, tzn. O(log N)

Delete (může mít rotace až do kořene)

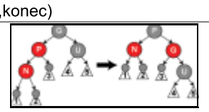
Find,Insert,Delete vždy v O(log n)

RB stromy

Pravidla:
- **kořen** (a **externí vrcholy**) jsou **černé**
- **otec červeného** uzlu je **černý** a jeho **synové** jsou **černí**
- Na cestě **od kořene do lib. uzlu** s jedním nebo žádným synem je **stejný počet černých** uzlů

Hloubka: k-poč.černých vrcholů,n-počet vrcholů $2^k-1 \le n \le 2^{2k}-1$
nejmenší strom má všechny vrcholy černé=>hloubka k-1 ; největší: střídavě černé a červ. =>hloubky 2k-1
takže k ≤ hloubka ≤ 2k => hloubka O(log n)

Insert (max 2 rot.)
- vložíme **nový vrchol N** jako do standatního BVS (jako list) a **obarvíme červeně**, první a 3 pravidlo jsme neporušili
- pokud je **otec černý** tak ani druhé (->konec)
- pokud je **otec červený**
  - **strýc** N **je červený** → **přebarvíme otce a strýce na cerno** a **dědu** na **červeno** a posuneme chybu na dědu
  - **strýc** N **je černý**
    - **N je levý syn** provedeme pravou rotaci na dědovi a přebarvíme (=pravidla splněna,konec)
    - **N je pravý syn** levá rotace na otci → přechozí případ



Find,Insert,Delete vždy v O(log n)

B-Stromy

Pravidla:
- ne všechny vrcholy mají stejný počet synů
- počet datových boxů v uzlu je alespoň m/2 a nejvýše m pro nějaké m>1
- kořen stupně alespoň 2
- všechny listy ve stejné hladině

Insert probíhá tak, že se najde místo, kam záznam vložit, pokud není uzel plný, prostě se záznam vloží, jinak se uzel rozštěpí, půlka prvků se dá vlevo, půlka vpravo a prostřední se vloží („mezi ně“) do otce. Pokud v otci není místo, pokračuje se stejným způsobem až do kořene, kde se případně vytvoří nový uzel a udělá se z něj kořen.

Delete prvků je opačný postup, v případě podtečení uzlu musím přebírat data od sousedních uzlů nebo slévat. V redundantních B-stromech není nutné při mazání odstraňovat vyhledávací klíč z vnitřních uzlů -- prvek s touto hodnotou se ve stromě už nebude nacházet, ale vyhledávat podle jeho klíče je dál možné.

Find,Insert,Delete vždy v O(log n)

vhodný pro externí paměť protože více dat v uzlech snižuje přstupy do paměti, a pokud máme data pouze v listech (**redundantní B-Stromy**) je to ještě lepší

B+ stromy jsou mírným vylepšením B-stromů pro zrychlení intervalových dotazů: všechny uzly ve stejné úrovni (a nebo jenom listy) jsou spojeny do spojového seznamu (možná je jednosměrná i obousměrná varianta).

B* stromy (řádu m) jsou úpravou B-stromů na základě vyvažování stránek. Druhá podmínka B-stromů se upraví tak, že každý uzel kromě kořene má minimálně 2m/3 zaplnění. Při vkládání prvků se stěpení odkládá opět do té doby, dokud nejsou plní i sourozenci daného listu; potom se štěpí buď 2 listy do 3, nebo 3 do 4 (buď s pomocí jednoho nebo dvou sousedních sourozenců). Odebírání podobně zahrnuje slévání 3 uzlů do 2 (nebo 4 do 3). Při obém lze ve složitější variantě zapojit ještě více uzlů.

Splay-trees

Splay - zarotuj určený uzel až do kořene

Insert - vlož jako v bvs a pak splay na přidaný

Delete - smaž jako v BVS(nahraď nejlevějším z pravého, nebo nejpravějším z levého) a splay na parenta

\+ poslední přistupované prvky jsou rychle dostupné

− nejhorším případ až O(n) jinak O(log n), mění se jenom hledáním

Optimání BVS - minimalizují dobu hledání když víme předem co budem hledat častěji