Vyvážené stromy

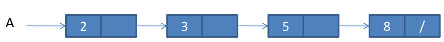
ako reprezentácia usporiadanej podmnožiny
2-3
B, 2-3-4
červeno-čierne
AVL

Usporiadaná množina

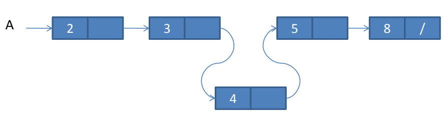
- Množina, pre ktorú je definovaná relácia usporiadania
- Operácie:
  - PRED: predchodca
  - SUCC: nasledovník
  - MIN: minimálny prvok
  - MAX: maximálny prvok
  - DELMIN: zmazanie min. prvku
  - DELMAX: zmazanie max. prvku
  - ALLMIN: zmazanie všetkých prvkov menších ako zadaný
  - ALLMAX: zmazanie všetkých prvkov väčších ako zadaný
  - ISINREL: test, či sú prvky v prelácií
  - Všetky operácie obyčajnej množiny

Usporiadaná množina – implementácia pomocou spájaného zoznamu

A - množina



Insert(A, 4)



Usporiadaná množina – implementácia pomocou stromov

- strom môže byť:
  - nevyvážený:
    - Zložitosť v priemere ~ log n
    - Zložitosť v najhoršom ~ n
    - napr. BVS
  - vyvážený:
    - hĺbky listov sa navzájom nelíšia o viac než určitú vzdialenosť
    - (hĺbka) h = $\log_2(n+1) - 1$
    - (počet vrcholov) n = 2^(h+1) – 1
    - Zložitosť v najhoršom ~ log n
    - Napr. B-stromy, 2-3 stromy, AVL stromy

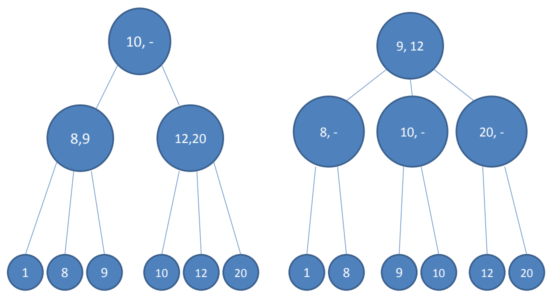
2-3 stromy

- Jedna trieda vyváženého stromu
- Platí:
  - Každý vnútorný vrchol má 2 alebo 3 nasledovníkov
  - Všetky listy sú rovnako vzdialené od koreňa
  - Dáta sú uložené (až) v listoch
    - varianta: dáta aj vo vnútorných vrcholoch
  - Všetky dáta sú v strome uložené v usporiadanom poradí

2-3 stromy

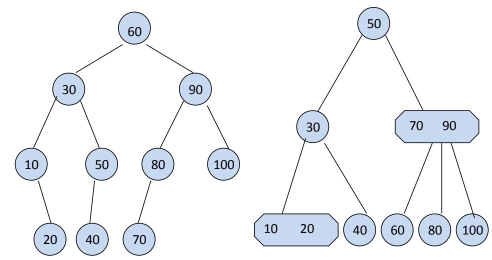
- Každý vnútorný prvok má priradenú dvojicu:
  - 1. prvok dvojice: Najmenší prvok spomedzi listov podstromu určeného jeho druhým nasledovníkom
  - 2. prvok dvojice: Ak ma 3 nasledovníkov, tak najmenší prvok spomedzi listov podstromu určeného jeho tretím nasledovníkom. Ak nemá 3 prvok, tak „-“

2-3 strom príklady



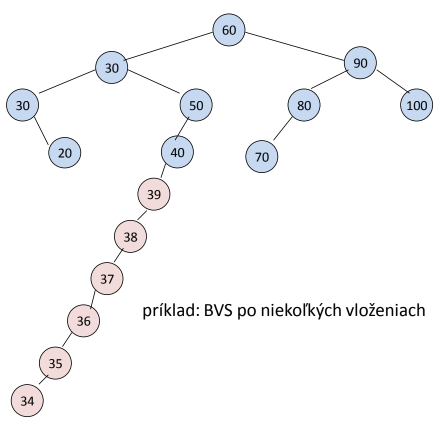
7

príklad: vyvážený BVS a 2-3 strom (varianta) s tými istými prvkami

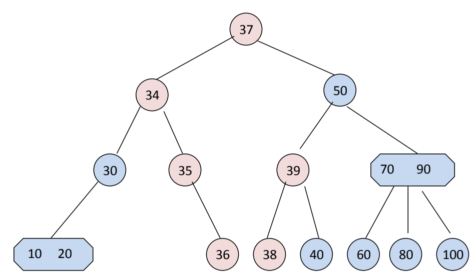


vyvážený BVS

2-3 strom (varianta)



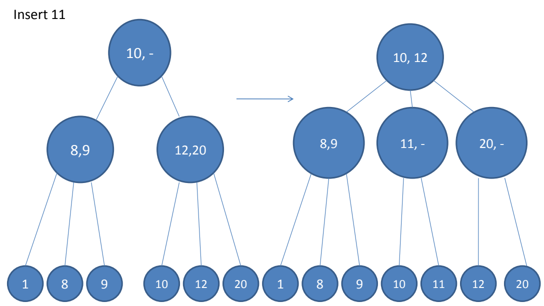
príklad: BVS po niekoľkých vloženiach



2-3 strom (varianta) po tých istých vloženiach

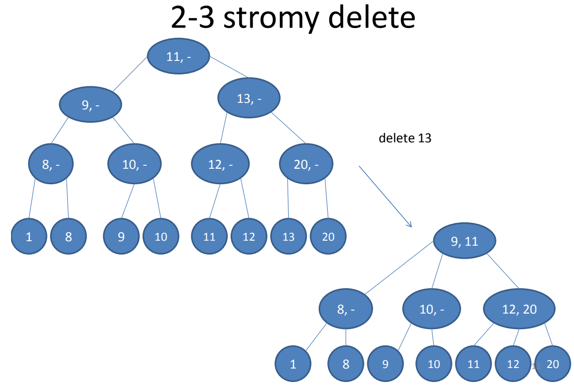
2-3 stromy insert

Insert 11



11

2-3 stromy delete



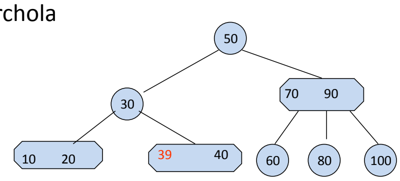
2-3 stromy vyhľadávanie

- Začni na koreni a porovnávaj hľadanú hodnotu s hodnotami vo vrchole. Ak nenastane zhoda, tak pokračuj v náležitom podstrome. Opakuj postup, až kým nenájdeš zhodu alebo nedosiahneš koniec podstromu.

13

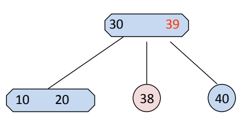
Insert do 2-3 stromu

- Insert 39. Hľadanie miesta pre 39 skončí v liste <40>. Keďže tento vrchol obsahuje iba jeden prvok, 39 sa jednoducho vloží do tohto vrchola



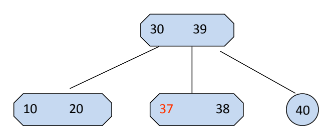
Insert do 2-3 stromu

- Insert 38: Hľadanie skončí vo vrchole <39 40>. Keďže vrchole nemôže mať 3 prvky, rozdelíme ich na najmenšiu (38), strednú (39) a najväčšiu(40). Strednú teraz posunieme do predchodcu.



Insert do 2-3 stromu

- Insert 37: Pre 37 sa nájde miesto v liste, ktorý obsahuje iba jeden prvok (38) – stačí ho tam pridať.

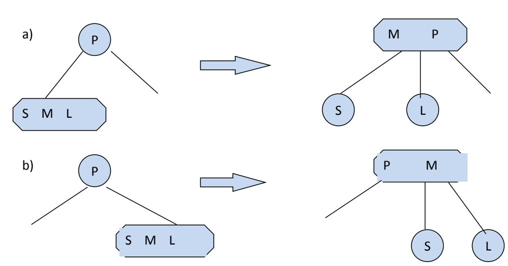


algoritmus vkladania do 2-3 stromu

- Nájdi list, v ktorom skončí hľadanie prvku p.
- Vlož prvok p do tohto listu.
- Ak list obsahuje teraz iba 2 prvky, skonči. Ak obsahuje 3 prvky, treba list rozdeliť.

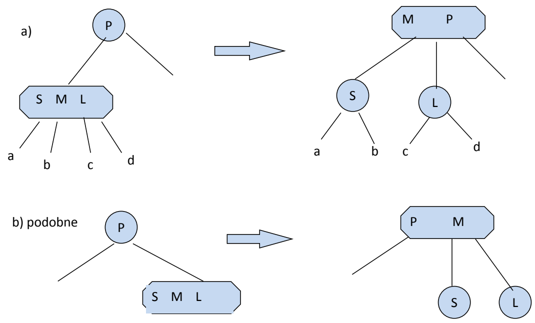
algoritmus vkladania do 2-3 stromu

- rozdelenie listu



algoritmus vkladania do 2-3 stromu

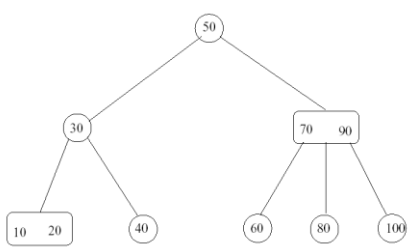
- rozdelenie vnútorného vrcholu



zrušenie prvku v 2-3 strome

- postup zrušenia prvku v 2-3 strome je opakom postupu vkladania. Tak ako sa pri vkladaní šíri účinok vloženia rozdeľovaním vrcholov, ktoré sa preplnia, pri rušení sa šíri účinok zrušenia spájaním vrcholov, ktoré sa vyprázdnili.
- nasleduje príklad:

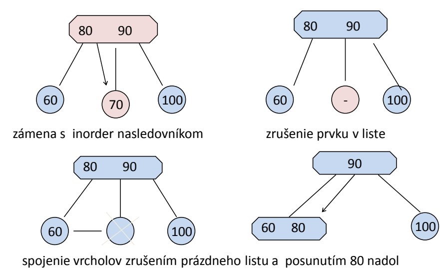
príklad 2-3 stromu



21

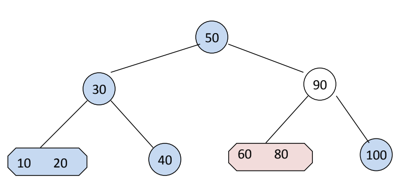
zrušenie prvku v 2-3 strome

- Delete 70



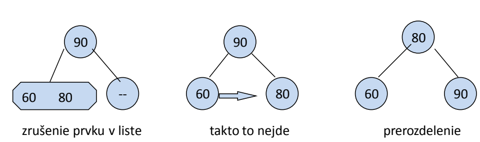
zrušenie prvku v 2-3 strome

- Delete 70



zrušenie prvku v 2-3 strome

- Delete 100



2-3 stromy zložitosť

- Vyhľadanie – O(log n)
- Vkladanie - O(log n)
- Vymazanie - O(log n)

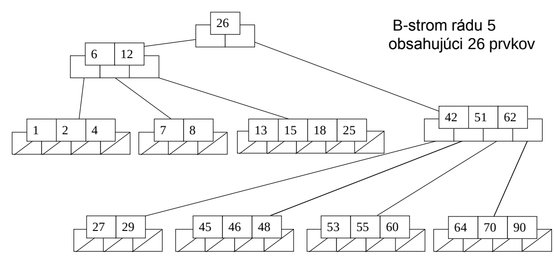
Definícia B-stromu

- B-strom rádu *m* je *m*-cestný strom (t.j. strom, v ktorom každý vrchol môže mať až *m* potomkov) v ktorom platí:
  1. počet kľúčov v každom vnútornom vrchole je o 1 menší než počet jeho potomkov a tieto kľúče rozdeľujú intervaly kľúčov v podstromoch
  2. každý list je v rovnakej hĺbke
  3. každý vnútorný vrchol okrem koreňa má najmenej $\lceil m / 2 \rceil$ potomkov
  4. koreň je buď list alebo má 2 až *m* potomkov
  5. list obsahuje najviac *m* – 1 kľúčov
- rád *m* je nepárne číslo

B stromy

- – kľúče sú uložené v neklesajúcej postupnosti
  – ak je x vnútorný vrchol s n(x) kľúčmi, tak obsahuje n(x) + 1 ukazovateľov na potomkov
  – kľúče vo vrcholoch rozdeľujú intervaly kľúčov v podstromoch
  – každý list je v rovnakej hĺbke
  – pre nejaké pevné t platí t>= 2 (tzv. minimálny stupeň)
  – každý vrchol okrem koreňa má aspoň t-1 kľúčov. Každý vnútorný vrchol okrem koreňa má aspoň t potomkov. Ak je strom neprázdny, má koreň aspoň 1 potomka.
  – každý vrchol má najviac 2t-1 kľúčov, t.j. najviac 2t potomkov.
- špeciálne
  – t=2: 2-3-4 stromy

príklad B-stromu

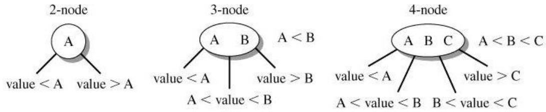


B-strom rádu 5 obsahujúci 26 prvkov

poznámka: všetky listy sú v rovnakej hĺbke

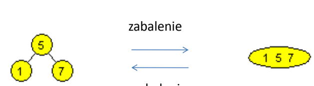
2-3-4 stromy

- 2-3-4 strom má 2-uzly, ktoré majú dvoch potomkov a jednu hodnotu, 3-uzly s tromi potomkami a dvomi hodnotami a 4-uzly so štyrmi potomkami a tromi hodnotami.



2-3-4 stromy

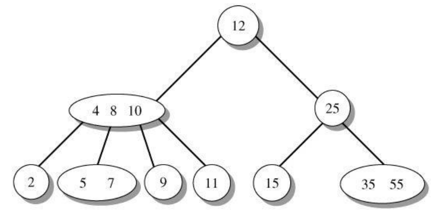
- Je to 2-3 strom, kde tri 2-uzly sú nahradené jedným 4-uzlom, čo zjednodušuje algoritmus vkladania a odstraňovania hodnôt.



prehľadávanie 2-3-4 stromu

- Začni v koreni a porovnávaj hľadanú hodnotu s hodnotami vo vrchole. Ak nenastane zhoda, tak pokračuj v náležitom podstrome. Opakuj postup, až kým nenájdeš zhodu alebo nedosiahneš koniec podstromu.

prehľadávanie 2-3-4 stromu



prehľadávanie B-stromu

B-Tree-Search(x, k)
i <- 1
while i <= n[x] and k > $key_i[x]$
  do i <- i + 1 // nájde sa najmenšie i také, že k ≤ $key_i[x]$ alebo sa priradí i n[x]+1
if i <= n[x] and k = $key_i[x]$ // našli sme už kľúč?
  then return (x, i) // ak áno, vracia sa ukazovateľ na uzol x a index i kľúča v ňom takého, že k = $key_i[x]$
if leaf[x]
  then return NIL // nenašli sme kľúč a uzol nemá potomka
  else Disk-Read($c_i[x]$) // nenašli sme kľúč ale uzol má potomka
return B-Tree-Search($c_i[x]$, k)

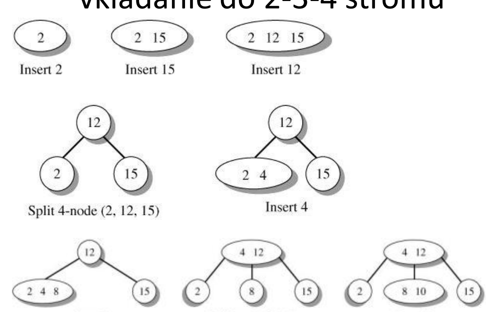
Vytvorenie prázdneho B-stromu

B-Tree-Create(T)
x <- Allocate-Node()
leaf[x] <- TRUE
n[x] <- 0
Disk-Write(x)
root[T] <- x

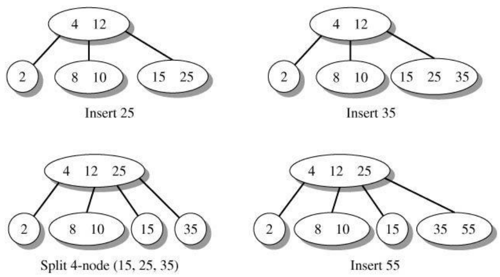
vkladanie do 2-3-4 stromu

- Nájdi list, do ktorého sa bude hodnota vkladať.
- Počas hľadania, keď narazíš na 4-uzol, tak ho rozbaľ.
- Ak je list, do ktorého vkladáme 2-uzol alebo 3-uzol, tak vlož do listu.
- Ak je list 4-uzol, tak ho rozbaľ tak, že prostrednú hodnotu vlož do rodičovského uzla a vkladanú hodnotu vlož do príslušného listu. Miesto v rodičovskom uzle sa určite nájde, keďže sme pri ceste dole rozbalili všetky 4-uzly. Preto nemusíme rekurzívne postupovať hore do ďalších uzlov ako to bolo pri 2-3 stromoch.

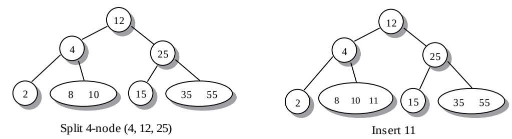
vkladanie do 2-3-4 stromu



vkladanie do 2-3-4 stromu



vkladanie do 2-3-4 stromu



vloženie uzla s kľúčom k do B-stromu

B-Tree-Insert(T, k)
r <- root[T]
if n[r] = 2t - 1 // koreň je plný?
  then s <- Allocate-Node() // plný, treba ho rozdeliť
    root[T] <- s // vytvorí sa nový koreň
    leaf[s] <- FALSE
    n[s] <- 0
    c_1 <- r
    B-Tree-Split-Child(s, 1, r)
    B-Tree-Insert-Nonfull(s, k)
  else B-Tree-Insert-Nonfull(r, k)

rozdelenie uzla v B-strome

B-Tree-Split-Child(x, i, y) // y je i-tý potomok uzla x, y sa rozdeľuje
z <- Allocate-Node()
leaf[z] <- leaf[y]
n[z] <- t - 1
for j <- 1 to t - 1 do key_j[z] <- key_{j+t}[y]
if not leaf[y] then for j <- 1 to t
do c_j[z] <- c_{j+t}[y]
n[y] <- t - 1
for j <- n[x] + 1 downto i + 1 do c_{j+1}[x] <- c_j[x]
c_{i+1} <- z
for j <- n[x] downto i do key_{j+1}[x] <- key_j[x]
key_i[x] <- key_t[y]
n[x] <- n[x] + 1
Disk-Write(y) ; Disk-Write(z) ; Disk-Write(x)

pomocné vloženie do neplného uzla v B-strome

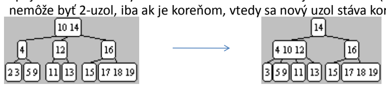
B-Tree-Insert-Nonfull(x, k)
i <- n[x]
if leaf[x] // uzol x je list,
  then while i >= 1 and k < key_i[x] // vkladá sa do listu
    do key_{i+1}[x] <- key_i[x]
      i <- i - 1
      key_{i+1}[x] <- k
      n[x] <- n[x] + 1
      Disk-Write(x)
  else while i >= and k < key_i[x] do i <- i - 1 // vkladá sa do vnútorného uzla
    i <- i + 1
    Disk-Read(c_i[x])
    if n[c_i[x]] = 2t - 1 then B-Tree-Split-Child(x, i, c_i[x]) //plný?
      if k > key_i[x] then i <- i + 1
    B-Tree-Insert-Nonfull(c_i[x], k)

odstraňovanie vrchola z 2-3-4 stromu

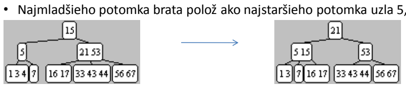
- Nájdi hodnotu, ktorá sa bude vymazávať a nahraď ju hodnotou inorder nasledovníka alebo predchodcu.
- Počas hľadania hodnoty a jeho nasledovníka zabaľuj 2-uzly do 3-uzlov alebo 4-uzlov. Takto zabezpečíme, že odoberaná hodnota bude v 3-uzle alebo 4-uzle.

odstraňovanie vrchola z 2-3-4 stromu

- Prípady zbalenia:
  - Odstraňujeme uzol 2, nachádzame sa na uzle 4, ktorého pravý brat je 2-uzol:
    - Spoj susedné hodnoty a deliacu hodnotu rodiča do jedného uzla (otec nemôže byť 2-uzol, iba ak je koreňom, vtedy sa nový uzol stáva koreňom)

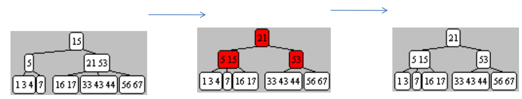


  - Odstraňujeme uzol 4, sme na uzle 5, ktorého pravý brat je 3-uzol:
    - Deliacu hodnotu otca (15) vlož do uzla 5 a na jeho miesto vlož najmenšiu hodnotou brata.
    - Najmladšieho potomka brata polož ako najstaršieho potomka uzla 5,15



odstraňovanie vrchola z 2-3-4 stromu

Delete 4



2-3-4 stromy zložitosť (max.)

- Vyhľadanie – int((log$_2$n)+1) = O(log n)
- Vkladanie = max. počet rozdeľovania uzlov * rozdeľovanie uzlov

  = int((log$_2$n)+1)*O(1)

  = O(log n)
- Vymazanie – O(log n)

Červeno-čierne stromy

Červeno-čierny strom je BVS, kde každý vrchol má navyše farbu (atribút s možnými hodnotami červený alebo čierny, implementačne 1 bit).

BVS je č-č strom, ak spĺňa tieto vlastnosti:

1.Každý vrchol je buď červený alebo čierny

2.Každý list (tzv. vonkajšie listy sú tu akoby pridané prázdne stromy, tj ukazovatele na nil v pôvodných listoch) je čierny

3.Každý červený vrchol má oboch potomkov čiernych

4.Každá cesta z (ľubovoľného pevne zvoleného) vrcholu x do listov v podstrome s koreňom x obsahuje **rovnaký počet** čiernych vrcholov

vlastnosti:

- každý vrchol okrem listov má práve dvoch potomkov
- na žiadnej ceste (od koreňa k listu) nie sú dva červené vrcholy za sebou (pozri 3.)
- každá cesta (od koreňa k listu) má rovnaký počet čiernych vrcholov (pozri 4.)
- najdlhšia cesta je najviac dvakrát tak dlhá jako najkratšia cesta – strom je „vyvážený“

definície:

Definície:

- výška vrchola: h(x) = počet vrcholov (nepočítajúc x) na najdlhšej ceste z x do listu v podstrome s koreňom x
- čierna výška vrchola: bh(x) = počet čiernych vrcholov (nepočítajúc x) na nejakej ceste z x do listu v podstrome s koreňom x

vlastnosti:

Lemma 1: Nech x je ľubovoľný vrchol. Potom podstrom s koreňom vo vrchole x má aspoň $2^{bh(x)}$-1 vnútorných vrcholov.

Lemma 2: Červeno-čierny strom s n (vnútornými) vrcholmi má výšku najviac 2 $\log_2(n+1)$.

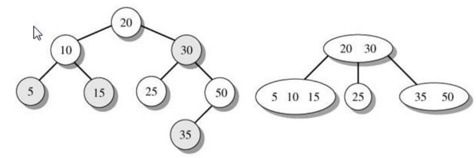
Dôkaz . Podľa lemmy 1 podstrom s koreňom vo vrchole x má aspoň $2^{bh(x)}$-1 vnútorných vrcholov. Použitím na koreň:
n>= $2^{h/2}$ -1 a z toho vyplýva h =< 2 log (n + 1).

Dôsledok: Dopytovacie operácie (Find, Min, Max, Succ, Predec) pre BVS majú na Č-Č stromoch garantovanú zložitosť O(log n) bez toho, aby ich (stromy) bolo treba meniť (nemôžu pokaziť žiadnu vlastnosť Č-Č stromov, pretože strom nemenia)

49

Č-č stromy a 2-3-4 stromy

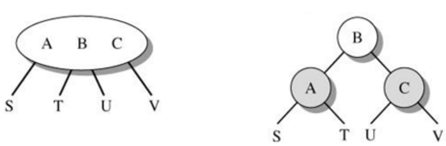
- Č-č strom ako reprezentácia 2-3-4 stromu



50

Č-č stromy a 2-3-4 stromy

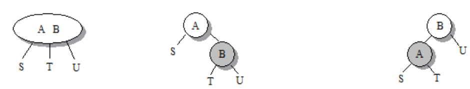
- 2-vrchol je vždy čierny
- 4-vrchol (A, B, C) sa zapíše ako vrchol s hodnotou B, ktorý má dvoch potomkov s hodnotami A a C.



51

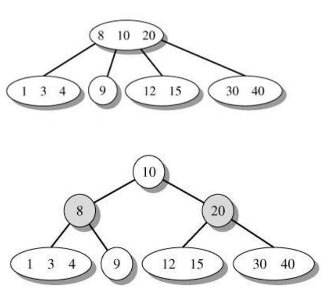
Č-č stromy a 2-3-4 stromy

- 3-vrchol (A, B) sa zapíše
  - buď ako čierny predchodca s A a väčší červený r-nasledovník s B
  - alebo ako čierny predchodca s B a menší čierny r-nasledovník s B



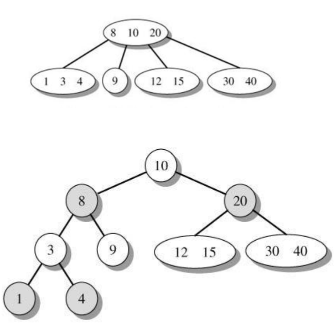
52

Príklad zapísania 2-3-4 stromu ako č-č stromu - 1



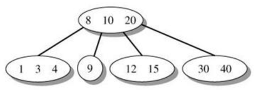
53

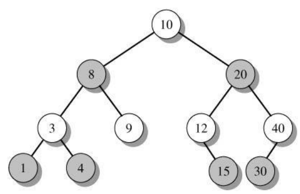
Príklad zapísania 2-3-4 stromu ako č-č stromu - 2



54

Príklad zapísania 2-3-4 stromu ako č-č stromu - 3





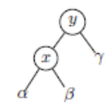
Rotácia (ľavá a pravá)

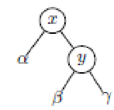
pomocné operácie potrebné pre implementáciu operácií Insert a Delete. Spĺňajú tieto vlastnosti:

• zachovávajú vlastnosť BVS – pre každý vrchol x platí, že kľúče v ľavom podstrome sú menšie než kľúč vrchola x a kľúče v pravom podstrome sú väčšie než kľúč vrchola x

• len presmerujú konštantne veľa ukazovateľov a teda vykonajú sa v O(1)

-> pravá rotácia (T; y)
<- ľavá rotácia (T; x)





Vkladanie vrchola

•Ak je koreň Č-Č stromu červený, tak sa dá prefarbiť na čierny bez toho, aby sa porušila ktorákoľvek vlastnosť Č-Č stromov.

•Preto môžeme predpokladať, že pred operáciou vkladania vrchola je koreň vždy čierny.

•Čiernotu koreňa budeme udržiavať.

Vkladanie vrchola

Predspracovanie:

•vrchol x vložíme insertom$_{BVS}$ a zafarbíme na červeno.

Ktorú vlastnosť Č-Č stromov môže predspracovanie porušiť?

rozbor možných prípadov:

1. x je koreň: prefarbiť na čierno
2. predchodca(x) je čierny: strom je po vložení v poriadku
3. predchodca (x) je červený: keďže y=predchodca (x) je červený, preto nemôže byť koreňom stromu, takže musí mať ešte predchodcu z (ktorý je určite čierny).

   •porušuje sa vlastnosť 3: nielen vložený vrchol x, ale aj jeho predchodca y sú červené.

porucha pri vkladaní

porucha nastáva v prípade:

y=predchodca (x) je červený,

z=predchodca (y)=predchodca(predchodca(x)) existuje a je čierny

ošetrenie:

a) súrodenec vrchola y (strýko vrchola x) je červený.

b) súrodenec vrchola y (strýko vrchola x) je čierny.

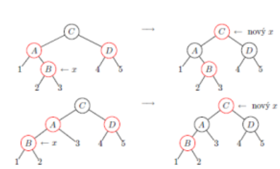
   a) x je opačným potomkom (Ľ/R) y než je y potomkom (R/Ľ) z.

   b) x je rovnakým potomkom (Ľ/R) y ako je y potomkom(Ľ/R) z.

porucha pri vkladaní

a) súrodenec vrchola y (strýko vrchola x) je červený.

Vrcholy prefarbíme (y a súrodenec(y) na čierno, z na červeno). Ak má vrchol z čierneho predchodcu, tak končíme, ak má červeného predchodcu, tak „chybu“ presúvame vyššie (opäť sú 3 možnosti). Ak vrchol z nemá predchodcu (tj. je to koreň), tak ho prefarbíme na čierno a končíme.

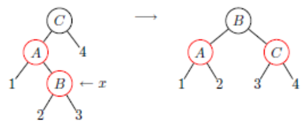


porucha pri vkladaní

b) súrodenec vrcholа y (strýko vrcholа x) je čierny.

a) x je opačným potomkom y než je y potomkom z.

Ak je x pravým potomkom y a y je ľavým potomkom z, tak Ľavá Rotácia(y), v opačnom prípade Pravá Rotácia(y).



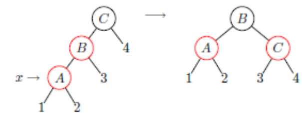
zdá sa, že tu niečo nesedí...
(vrcholy A a B?)

porucha pri vkladaní

b) súrodenec vrcholа y (strýko vrcholа x) je čierny.

b) x je rovnakým potomkom y ako je y potomkom z.

Ak je x pravým potomkom y a y je pravým potomkom z, tak Ľavá Rotácia(y) a prefarbiť y na čierno a z na červeno, v opačnom prípade Pravá Rotácia(y) atď. symetricky.



zložitosť vkladania

je O(log n) :

- predspracovanie (obyčajný insert$_{BVS}$) je O(log n)
- akcia prípadu 1. je O(1) a vykoná sa O(log n) krát
- akcie prípadov 2. a 3. sú obe O(1) a vykonajú sa každá nejviac raz

Odstránenie vrcholа

Predspracovanie: delete$_{BVS}$(y), y je vrchol, ktorý sa odstraňuje

•Odstraňovanie v BVS kopíruje do odstraňovaného vrcholа najmenšiu väčšiu hodnotu, tj nasledovníka(y).

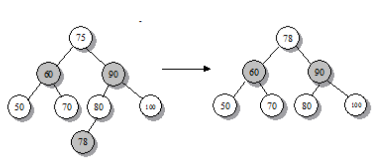
•Vrchol, v ktorom bol nasledovník(y), má najviac jedného (vnútorného) potomka (l-nasledovníka alebo r-nasledovníka) x.

• Ak nemá vnútorného potomka, x označuje jedného z jeho vonkajších potomkov.

Odstránenie vrcholа

Ak vrchol, v ktorom bol nasledovník(y), je červený, po jeho odstránení netreba nič robiť, lebo vlastnosti Č-Č stromov platia aj naďalej.

delete 75. nasledovník je 78, bol v červenom vrchole.



Odstránenie vrcholа

Ak vrchol, v ktorom bol nasledovník(y), je červený, po jeho odstránení netreba nič robiť, lebo vlastnosti Č-Č stromov platia aj naďalej. Ak je čierny, tak jeho odstránením sa poruší vlastnosť 4 (okrem prípadu, že je koreň).

Ak je x červený, prefarbiť x na čierno. Tým sa obnovia vlastnosti Č-Č stromov.

Ak je x čierny - ?

Odstránenie vrchola

Ak je x čierny - ?

vrchol x urobiť „dvojito čierny“ (tým sa splní vlastnosť 4) a túto druhú čiernu farbu posúvať vyššie:

ak je x koreň stromu, tak druhú čiernu farbu zrušíme.

ak x nie je koreň, tak rodič(x) má aj jedného vnútorného potomka (označíme si ho w). Prečo? vlastnosť 4 (externý potomok vrchola rodič(x) by mal menšiu čiernu výšku než x)

Predpokladajme, že x je ľavý potomok vrchola rodič(x) (pre opačný prípad je riešenie symetrické). Treba rozlíšiť 4 prípady podľa farby w a jeho prípadných potomkov:

1. vrchol w je červený (a teda má 2 čiernych potomkov)

vymeniť farbu w a jeho rodiča a vykonať Ľavú Rotáciu(rodič(x)). teraz je to jeden z prípadov 2 nebo 3 nebo 4

2. vrchol w je čierny a má 2 čiernych potomkov

odstrániť jednu čiernu farbu z x a prefarbiť w na červeno. Ak je ich spoločný rodič červený, prefarbiť na čierno a skončiť. Ak je čierny, tak ho prefarbiť na dvojitú čiernu. Takéto posúvanie dvojitej čiernej nahor sa určite skončí najneskôr v koreni.

3. vrchol w je čierny, jeho ľavý potomok je červený a pravý potomok je čierny

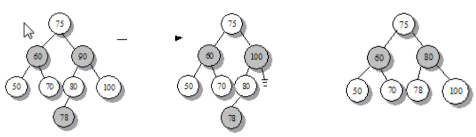
vymeniť farbu w a jeho ľavého potomka a vykonať Pravú Rotáciu(w). teraz je to prípad 4

4. vrchol w je čierny a jeho pravý potomok je červený

prefarbiť pravého potomka w na čierno a x odfarbiť od jednej čiernej farby. Ak bol rodič(x) červený, tak ho prefarbiť na čierno a vrchol w na červeno. Vykonať Ľavú Rotáciu(rodič(x)).

Odstránenie vrchola

- delete 90. nasledovník je 100, bol v čiernom vrchole.
- treba prefarbenie aj rotáciu doprava okolo 80.



Odstraňovanie vrchola

je O(log n):

- predstracovanie (delete$_{BVS}$) je O(log n)
- prípad 2 je O(1) a vykoná sa O(log n) krát
- prípady 1, 3 a 4 sú O(1) a vykonajú sa najviac raz

Georgij Maximovič Adeľson-Veľskij

- (* 8. január 1922, Samara, Sovietske Rusko)
- 1948 – PhD, Moskovská štátna univerzita
- sovietsky matematik
- príspevok do umelej inteligencie: programovanie šachu
- príspevok do informatiky: údajová štruktúra AVL strom
- Adelson-Velskii, G.; E. M. Landis (1962). "An algorithm for the organization of information". *Proceedings of the USSR Academy of Sciences* **146**: 263–266. **(rusky)** anglický preklad: Myron J. Ricci in *Soviet Math. Doklady*, 3:1259–1263, 1962.

Jevgenij Michailovič Landis

- (* 6. október 1921, Charkov, Sovietska Ukrajina - † 12. december 1997, Moskva , Ruská federácia)
- študoval Moskovská štátna univerzita
- sovietsky matematik
- výskum: diferenciálne rovnice
- príspevok do informatiky: údajová štruktúra AVL strom
- Adelson-Velskii, G.; E. M. Landis (1962). "An algorithm for the organization of information". *Proceedings of the USSR Academy of Sciences* **146**: 263–266. **(rusky)** anglický preklad: Myron J. Ricci in *Soviet Math. Doklady*, 3:1259–1263, 1962.

AVL stromy

Definícia (Adelson-Velskii, Landis) BVS je AVL strom (vyvážený strom) práve vtedy, ak pre každý vrchol x v strome platí

|(výška ľavého podstromu vrchola x) - (výška pravého podstromu vrchola x)| ≤ 1

Vlastnosť: Výška AVL stromu s n vrcholmi je O(log n).

Dôsledok Všetky dopytovacie operácie nad BVS (Find, Min, Max, Succ, Predec), ktoré nemenia prehľadávaný strom , majú na AVL strome zložitosť O(log n).

Operácie Insert a Delete, ktoré strom menia, pracujú rovnako ako nad BVS. Prípadne treba dodatočne vyvažovať strom rotáciami.

AVL stromy

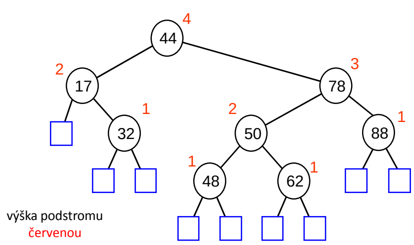
|(výška ľavého podstromu vrchola x) - (výška pravého podstromu vrchola x)| ≤ 1

faktor vyváženia bf:

bf (x) = výška(ľavý podstrom(x)) – výška(pravý podstrom(x))

AVL stromy

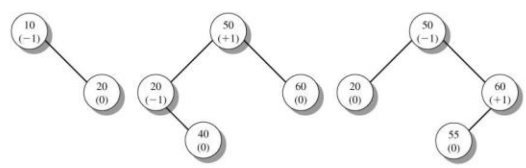
|(výška ľavého podstromu vrchola x) - (výška pravého podstromu vrchola x)| ≤ 1



AVL stromy

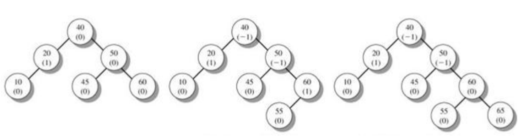
faktor vyváženia bf:

bf (x) = výška(ľavý podstrom(x)) – výška(pravý podstrom(x))



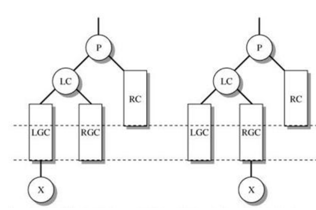
vkladanie do AVL stromu

- insert 55
- insert 65
- ani jeden insert neporušil vyváženosť AVL



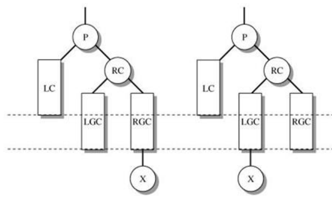
vkladanie do AVL stromu

- insert x doľava môže pokaziť vyváženosť predchodcu P (bf(P)=2)



vkladanie do AVL stromu

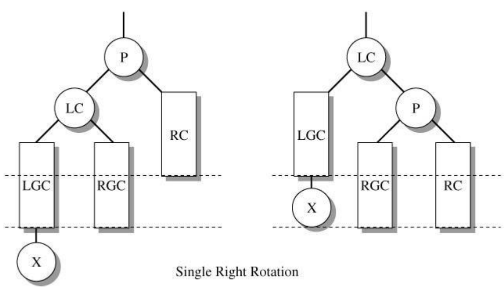
- insert x doprava môže pokaziť vyváženosť predchodcu P (bf(P)=-2)



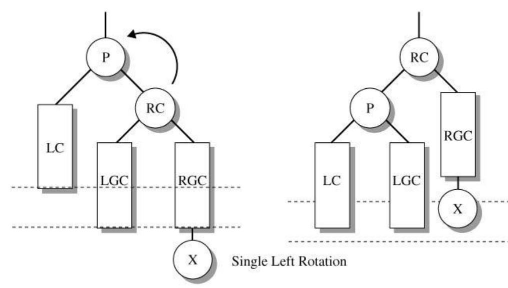
algoritmická schéma operácií nad AVL stromom

nájdi / vlož / zruš obdobne ako v BVS
po každom vložení / zrušení v aktuálnom vrchole
repeat
    if výšky potomkov aktuálneho vrchola sa líšia viac než o 1
    then
        rotuj potomky dovtedy, až sú podstromy vyvážené
        aktuálny vrchol <- predchodca(aktuálny vrchol)
    else exit
until
    aktuálny vrchol = nil (naposledy sa testoval koreň)

rotácia AVL stromu doprava

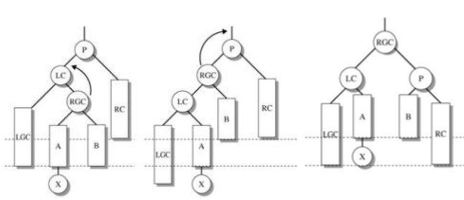


rotácia AVL stromu doľava



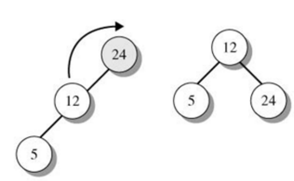
dvojitá rotácia AVL stromu doprava

- pozostáva z jednoduchej ľavej rotácie okolo LC a jednoduchej pravej rotácie okolo P



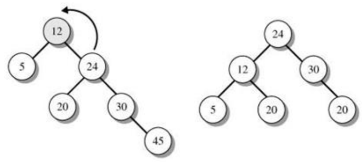
vkladanie do AVL stromu – príklad 1

- insert 24, insert 12, insert 5
- teraz je bf(24)=2
- jednoduchá pravá rotácia okolo 24



vkladanie do AVL stromu – príklad 2

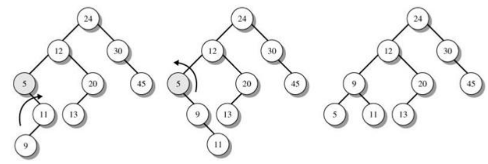
- ... insert 30, insert 20, insert 45
- teraz je bf(12)=-2
- jednoduchá ľavá rotácia okolo 12



85

vkladanie do AVL stromu – príklad 3

- ... insert 11, insert 13, insert 9
- teraz je bf(5)=-2
- jednoduchá pravá rotácia okolo 11
- jednoduchá ľavá rotácia okolo 5



86

AVL stromy zložitosť

- Vyhľadanie – O(log n)
- Vkladanie - O(log n)
- Vymazanie - O(log n)

87