

Dátové štruktúry a algoritmy

Pokročilé algoritmy vyhľadávania

10. 10. 2017

zimný semester
2017/2018

Opakovanie – Problém vyhľadávania

■ Vstup:

- Postupnosť: $a_1, a_2, a_3 \dots a_n$
 $k(a_i)$ označíme kľúč k_i prvku a_i
- Hľadaný kľúč x
- Čo sú kľúče?
Definičný obor D – reťazce, reálne čísla, dvojice celých čísel, ...
- Relácia = (rovnosti) – relácia ekvivalencie nad D
- Usporiadanie kľúčov $<$ (binárna relácia nad D)
Lineárne usporiadaná množina K (total ordering)
Pre $k_1, k_2 \in D$ budeme písať, že $k_1 \leq k_2$ ak $k_1 < k_2$ alebo $k_1 = k_2$.

■ Výstup:

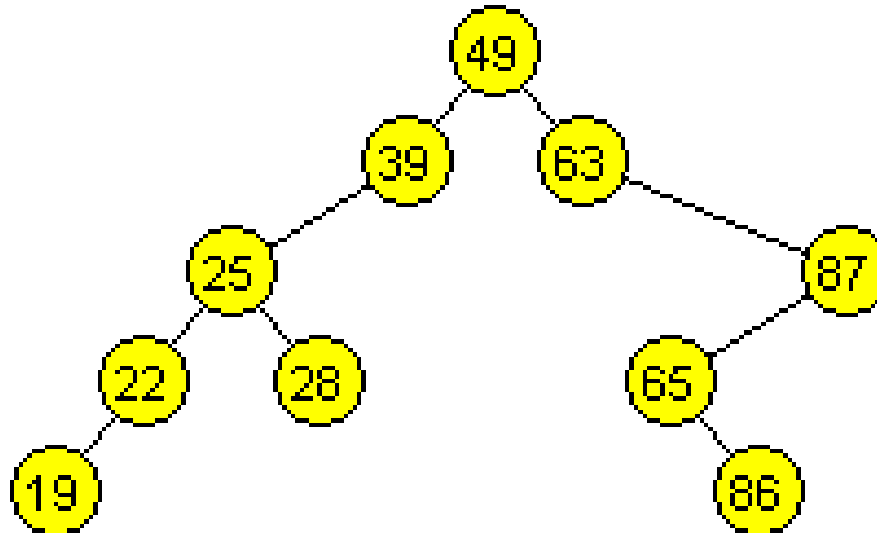
- Index $res \in \{1, 2, \dots, n\}$ takého prvku, že $k(a_{res}) = x$,
alebo 0 ak taký prvok neexistuje.

Opakovanie – Základné algoritmy

- Čím viac informácií o vstupnej postupnosti mám k dispozícii, tým rýchlejší algoritmus dokážem vytvoriť
 - Lineárne vyhľadávanie: $O(n)$
 - Binárne vyhľadávanie: $O(\log n)$
 - Interpoláčné vyhľadávanie: $O(\log \log n)$
- Binárne vyhľadávacie stromy
 - Priemerný prípad: $O(\log n)$
 - **Najhorší prípad: $O(n)$**
- Niektoré špecializované typy vyhľadávania
 - Prioritný front (vyhľadávam len najprioritnejší prvok):
insert / removeMax : $O(\log n)$

Nová operácia: nájsi k-ty prvok v strome

- Prvé riešenie:
Využiť in-order usporiadanie, zobrat' k-ty prvok
 - Zložitosť $O(k)$
- Napr. $k=5$



- In-order: 19, 22, 25, 28, **39**, 49, 63, 65, 86, 87

Nová operácia: nájsť k-ty prvok v strome

- Lepšie riešenie: využiť princíp QuickSelect algoritmu **pri porovnaní vo vrchole pokračovať len v podstrome, v ktorom sa k-ty prvok nachádza**
- Potrebujeme pre každý vrchol x poznať: **počet prvkov v podstrome strome s koreňom x**
- Implementácia ako **rozšírenie štandardnej dátovej štruktúry BVS**, rozšírime údaje pre vrchol:
 - ľavý, pravý, rodič, **počet** (prvkov v podstrome) tzv. **váha**
 - rekurzívna definícia váhy
$$\text{váha}(v) = \text{váha}(\text{ľavýPodstrom}(v)) + \text{váha}(\text{pravýPodstrom}(v)) + 1$$
- Hodnoty **váha** vo vrcholoch upravujeme pri každej operácii ktorá mení štruktúru stromu: zložitosť $O(h)$, kde h je výška stromu

Order statistic tree

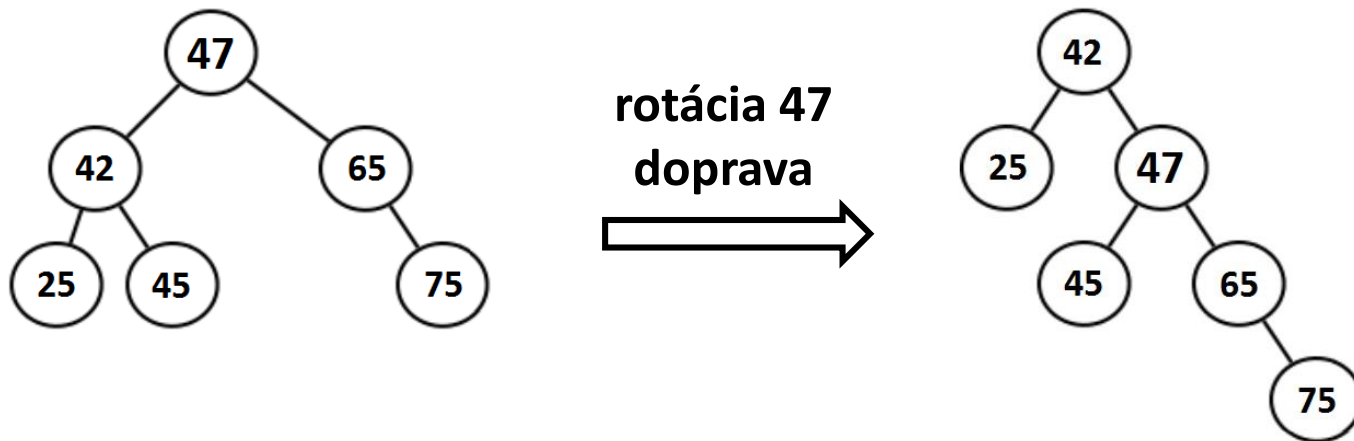
- Rozšírenie BVS stromu
- Pre každý vrchol BVS si navyše pamätáme **počet prvkov v podstrome vrcholu** tzv. váhu
- Hodnoty váhy vo vrcholoch upravujeme pri každej operácii ktorá mení štruktúru stromu (insert, delete)
- Rozšírený strom podporuje navyše operácie:
 - **select(k)** – nájsť k-ty najmenší prvok v množine
 - **rank(x)** – nájsť poradie prvku x v usporiadanej postupnosti prvkov stromu
- Zložitosť operácií $O(h)$, kde h je výška stromu

Ako vylepšiť všeobecné vyhľadávacie stromy?

- Obmedziť ich štruktúru, aby sme mohli o nej prehlásiť nejaké vlastnosti – napr. že bude vždy nízka výška stromu
- Z týchto garancií (na veľkosť výšky) vyplynú efektívne zložitosti operácií nad takýmito stromami
- Na získanie optimálnej zložitosti $O(\log n)$ musíme zabezpečiť, aby strom po vykonaní každej operácie zostal vyvážený
- Ako zabezpečiť vyváženie stromu?
 - Hodnoty v strome meniť nemôžeme :)
 - Musíme nejako **upravovať štruktúru** stromu

Rotácia stromu – doprava

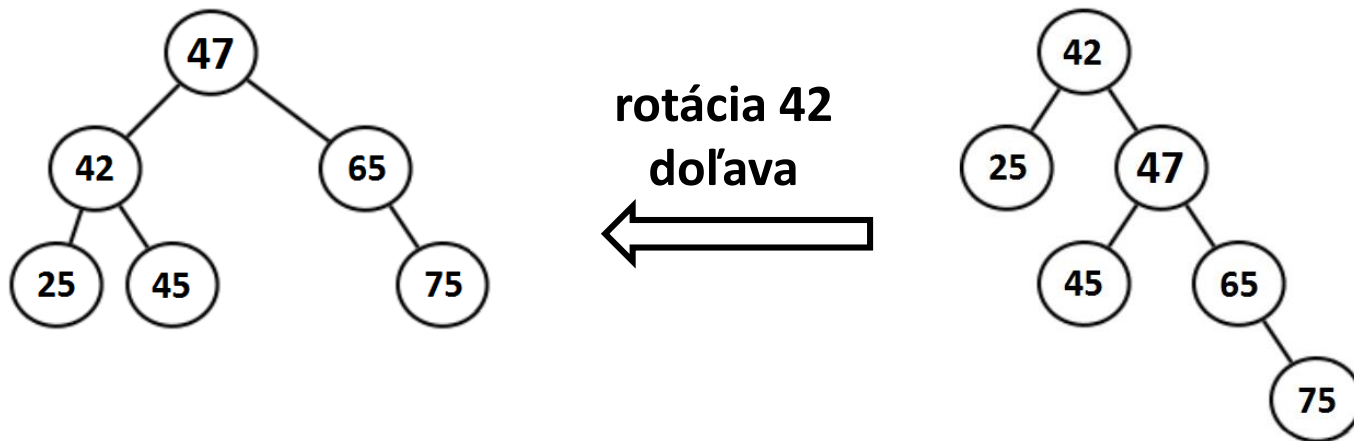
- Operácia, ktorá zmení štruktúru ale zachová usporiadanie
- Zmena tvaru stromu – zmena výšky stromu
- **Rotácia doprava:**
ľavé dieťa sa presunie (doprava hore) na miesto rodiča



- Zmena hĺbky 25(-1), 42(-1), 47(+1), 65(+1), 75(+1)
- In-order poradie (oba stromy): 25, 42, 45, 47, 65, 75

Rotácia stromu – doľava

- Operácia, ktorá zmení štruktúru ale zachová usporiadanie
- Zmena tvaru stromu – zmena výšky stromu
- **Rotácia doľava:**
pravé dieťa sa presunie (doľava hore) na miesto rodiča



- Zmena hĺbky 25(+1), 42(+1), 47(-1), 65(-1), 75(-1)
- In-order poradie (oba stromy): 25, 42, 45, 47, 65, 75

Výškovo vyvážené stromy (AVL stromy)

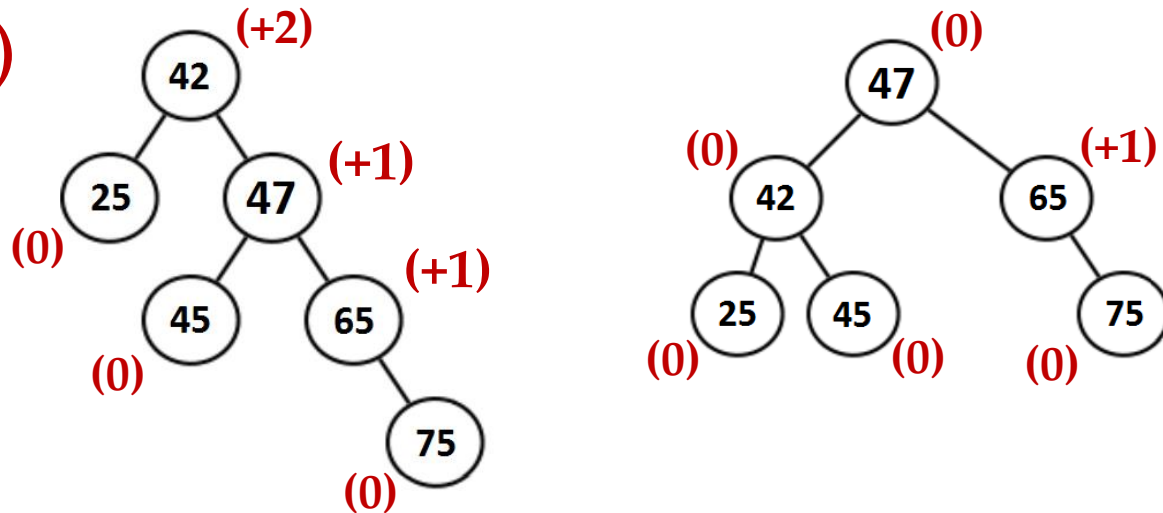
- Označme faktor vyváženia (balance factor) vo vrchole:

$$\text{bf}(v) = \text{výška}(\text{pravýPodstrom}(v)) - \text{výška}(\text{ľavýPodstrom}(v))$$

- Výškovo vyvážený strom (AVL strom):

$$|\text{bf}(v)| \leq 1, \text{ pre každý vrchol } v$$

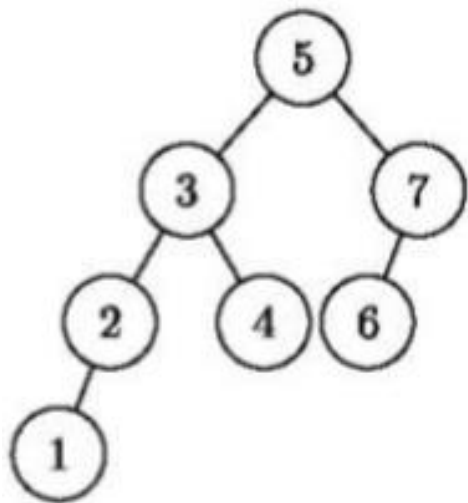
- Napr. (bf)



Výška AVL stromu

- Aké sú najhoršie prípady výškovo vyvážených stromov?
 - Pre danú výšku také, ktoré majú čo najmenej vrcholov, lebo
 - ak by sme pridali / odstránili nejaký list, tak by sme strom
 - a) viac vyvážili (čo nechceme), alebo
 - b) by to už nebol výškovo vyvážený strom (lebo $bf(v) > 1$)

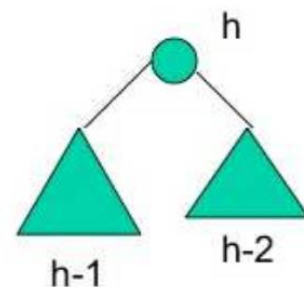
Napr. pre 7 vrcholov:



Označme N_h minimálny počet vrcholov AVL stromu výšky h .

AVL strom s N_h vrcholmi, musí mať jedno dieťa koreňa, ktoré má výšku $h-1$ a teda najmenej vrcholov v jeho podstrome je N_{h-1} . Minimálna výška druhého dieťaťa koreňa je $h-2$, a teda jeho podstrom môže mať najmenej N_{h-2} vrcholov:

$$N_h = N_{h-1} + N_{h-2} + 1$$



Výška AVL stromu (2)

- N_h – najmenší počet vrcholov „najhoršieho“ AVL stromu výšky h
- $N_1 = 1, N_2 = 2$. Rekurzívna konštrukcia „najhoršieho“ stromu pre väčšie výšky: $N_h = N_{h-1} + N_{h-2} + 1$

- **Odhadneme výšku h :**

$$N_h = N_{h-1} + N_{h-2} + 1$$

$$N_{h-1} = N_{h-2} + N_{h-3} + 1$$

$$N_h = (N_{h-2} + N_{h-3} + 1) + N_{h-2} + 1$$

$$N_h > 2N_{h-2} > 2 \cdot 2 \cdot N_{h-4} > \dots > 2^{\frac{h}{2}}$$

$$N_h > 2^{\frac{h}{2}}$$

$$\log N_h > \log 2^{\frac{h}{2}}$$

$$2 \log N_h > h$$

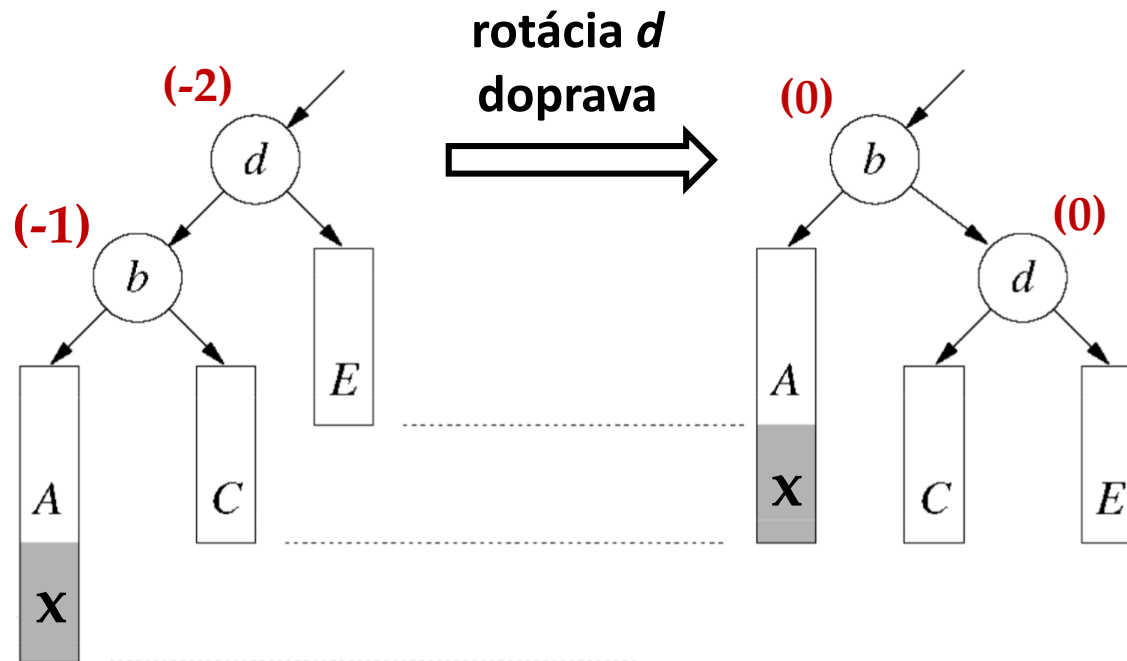
$$h = O(\log N_h)$$

Operácie nad AVL stromom

- **Ak operácia nemení štruktúru stromu**
(napr. min, max, select, succ, pred) vykonávame rovnako ako pri BVS, ale navyše máme garantovanú zložitosť $O(\log N)$, kde N je počet vrcholov v AVL strome.
- **Ak operácia mení štruktúru stromu**
(napr. insert, delete), vykonáme rovnako ako pri BVS a následne dodatočne vyvážíme strom rotáciami.
 - Časová zložitosť jednej rotácie $O(1)$
 - Vykonáme najviac h rotácií, kde $h = O(\log N)$, preto celková zložitosť insert aj delete je $O(\log N)$

Insert(x) do AVL stromu

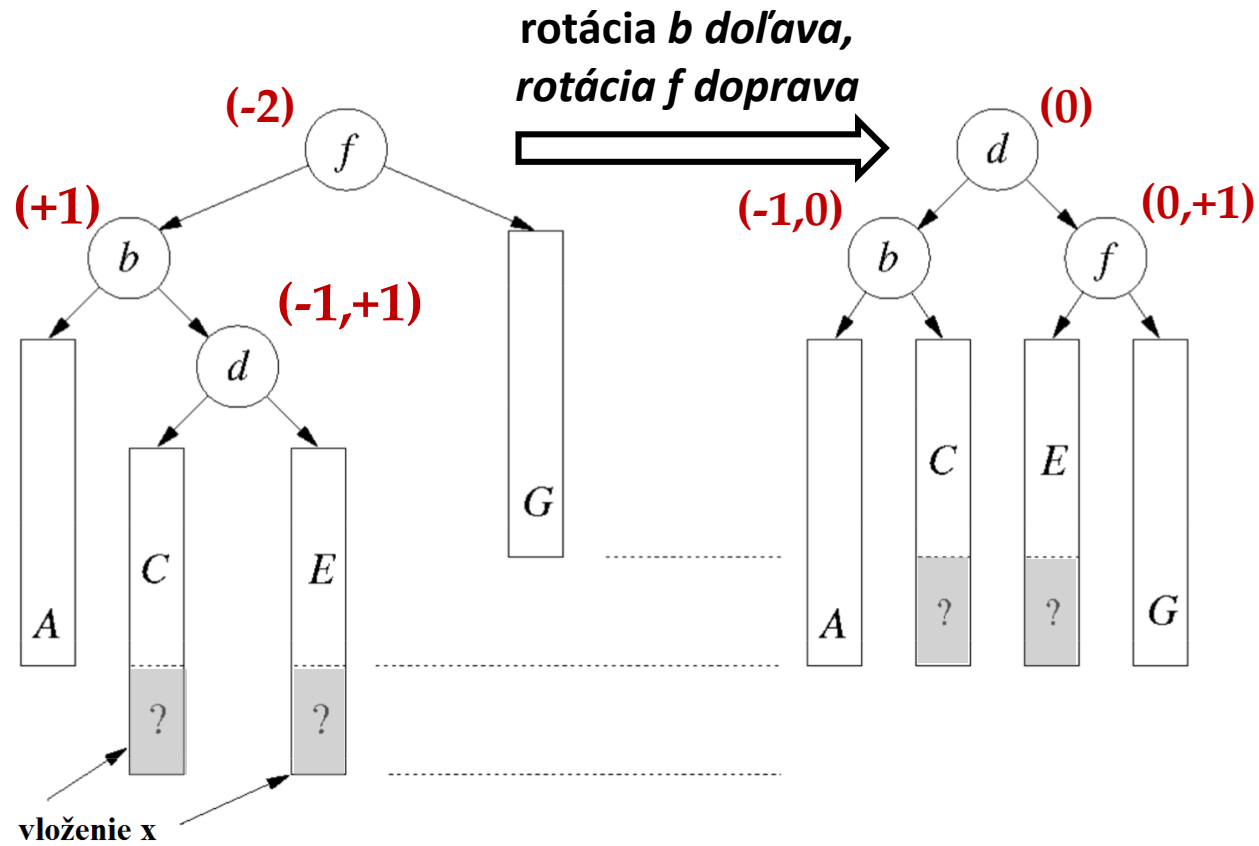
- Postupnosť rotácií závisí podľa typu nevyváženosti
- Uvažujme, že po pridaní je ľavý podstrom (b) ľavého dieťaťa (d) príliš hlboký: **rotácia doprava**



(pravý podstrom pravého dieťaťa je symetrická situácia: **rotácia doľava**)

Insert(x) do AVL stromu (2)

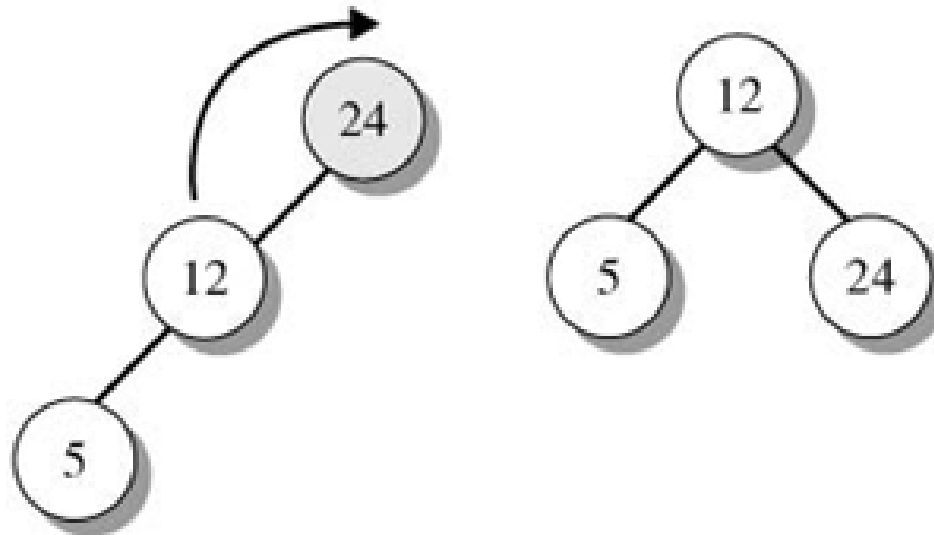
- Uvažujme, že po pridaní je pravý podstrom (d) ľavého dieťaťa (b) príliš hlboký: **dvojitá rotácia doprava**



(ľavý podstrom pravého dieťaťa je symetrická situácia: **dvojitá rotácia doľava**)

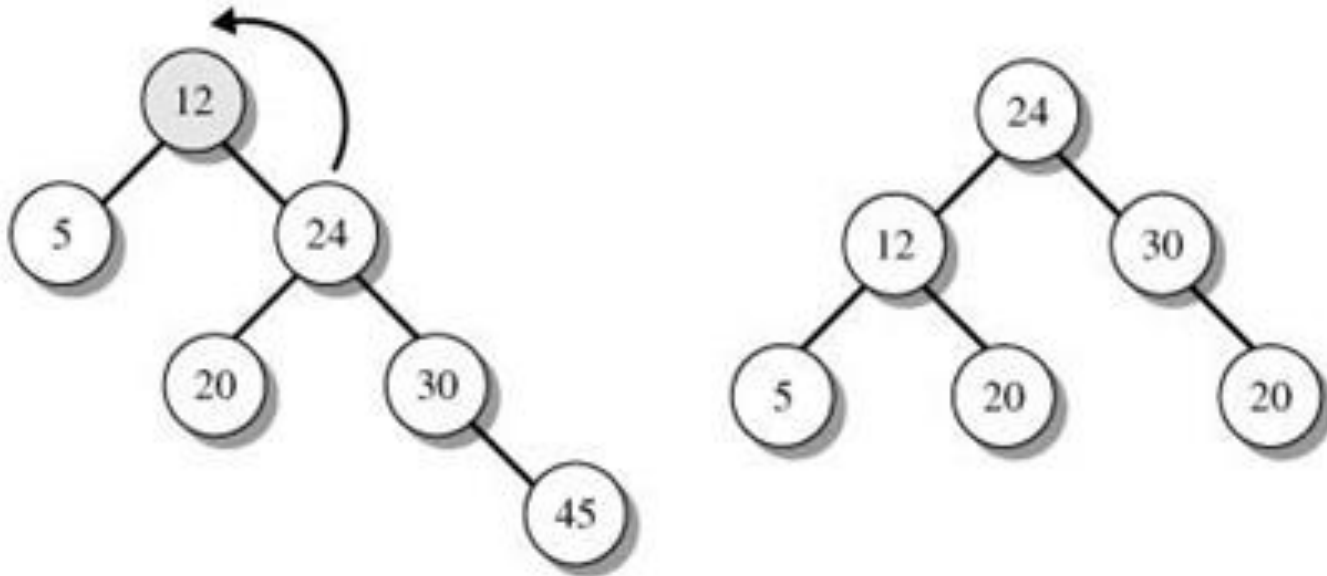
Insert do AVL stromu – Ukážka

- insert 24, insert 12, insert 5
- teraz je **bf(24) = -2**
- jednoduchá rotácia 24 doprava



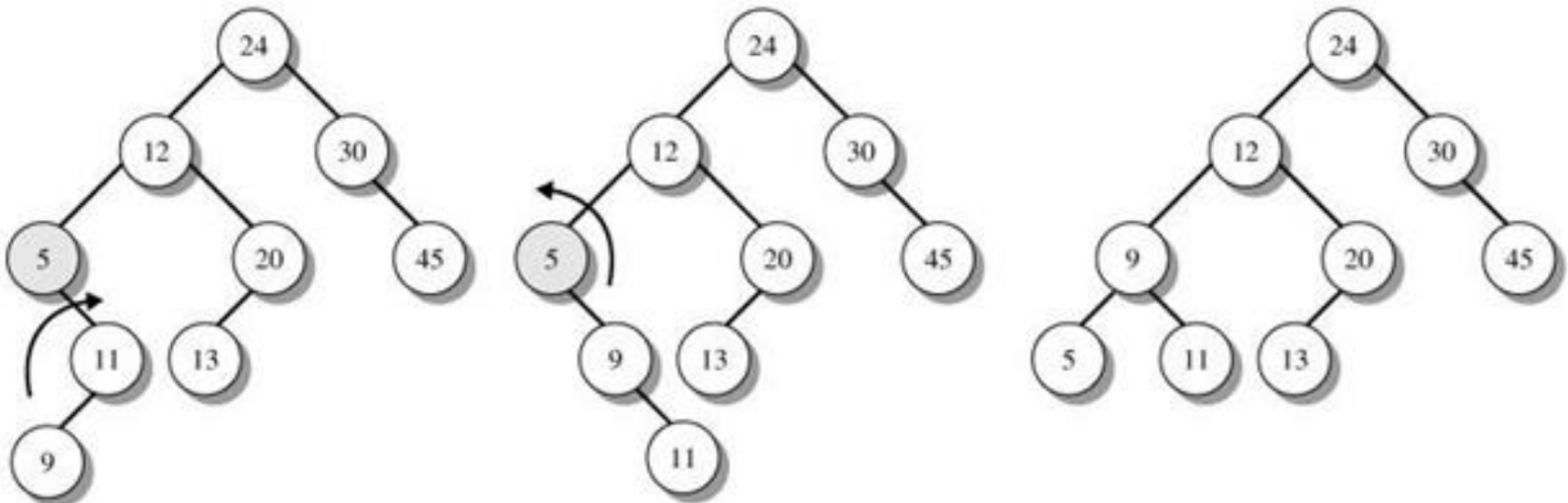
Insert do AVL stromu – Ukážka (2)

- ... insert 30, insert 20, insert 45
- teraz je **bf(12) = +2**
- jednoduchá rotácia 12 doľava



Insert do AVL stromu – Ukážka (3)

- ... insert 11, insert 13, insert 9
- teraz je **bf(5) = +2**
- Dvojitá rotácia:
 - jednoduchá rotácia 11 doprava
 - jednoduchá rotácia 5 doľava

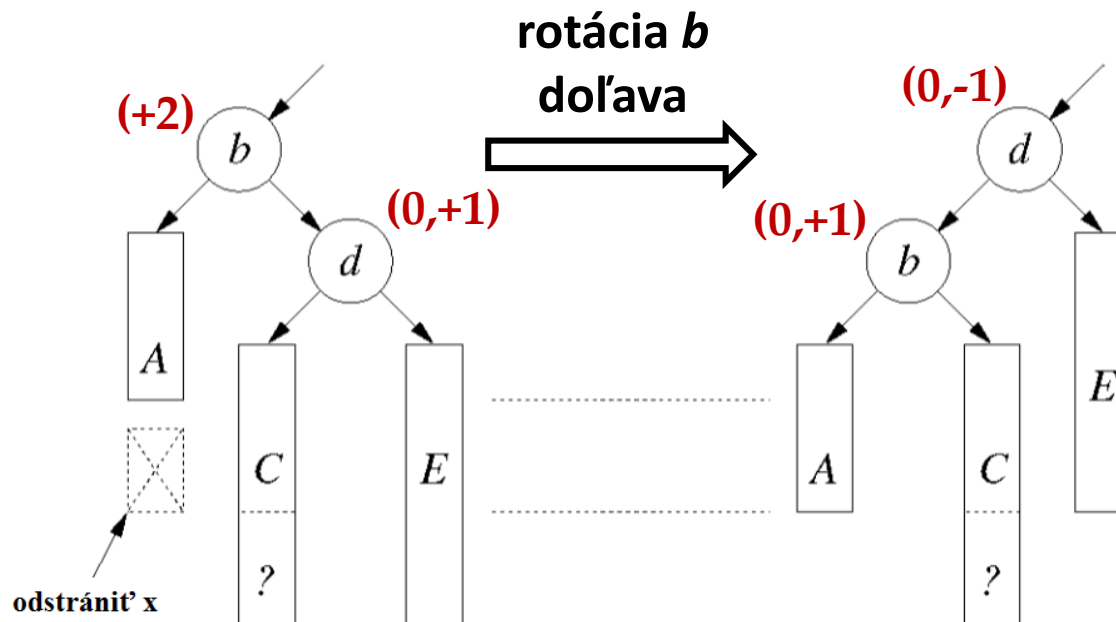


Odstránenie z AVL stromu

- Podobne ako pri vkladaní
- Vykonáme operáciu odstránenia nad obyčajným BVS a následne prechodom (od miesta odstráneného vrcholu) do koreňa upravujeme faktor vyváženia (bf) vrcholov, rotujúc vo vrcholoch, ktoré sú nevyvážené
- Môžeme rotovať viac krát, ale **vždy najviac $O(\log n)$ krát**

Delete(x) z AVL stromu

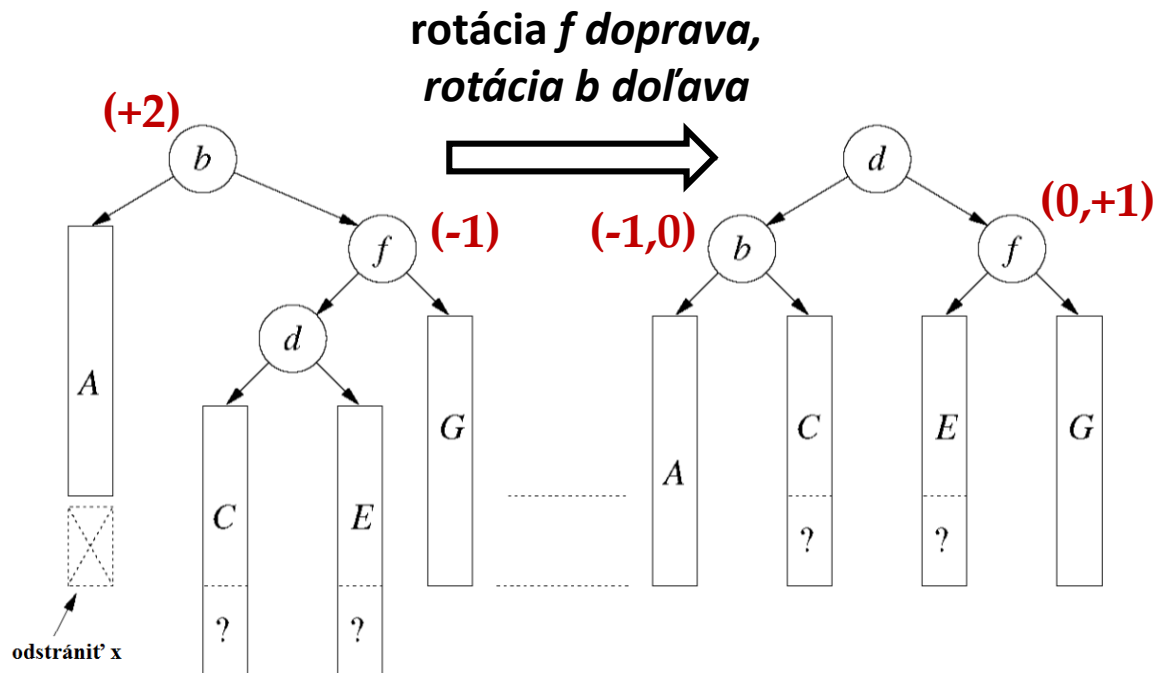
- Uvažujme, že po odstránení prvku z ľavého podstromu, niektorý jeho predok (b) zostane nevyvážený, pričom bf jeho pravého dieťaťa (d) je 0 alebo +1: **rotácia doľava**



(odstránenie z pravého podstromu je symetrická situácia: **rotácia doprava**)

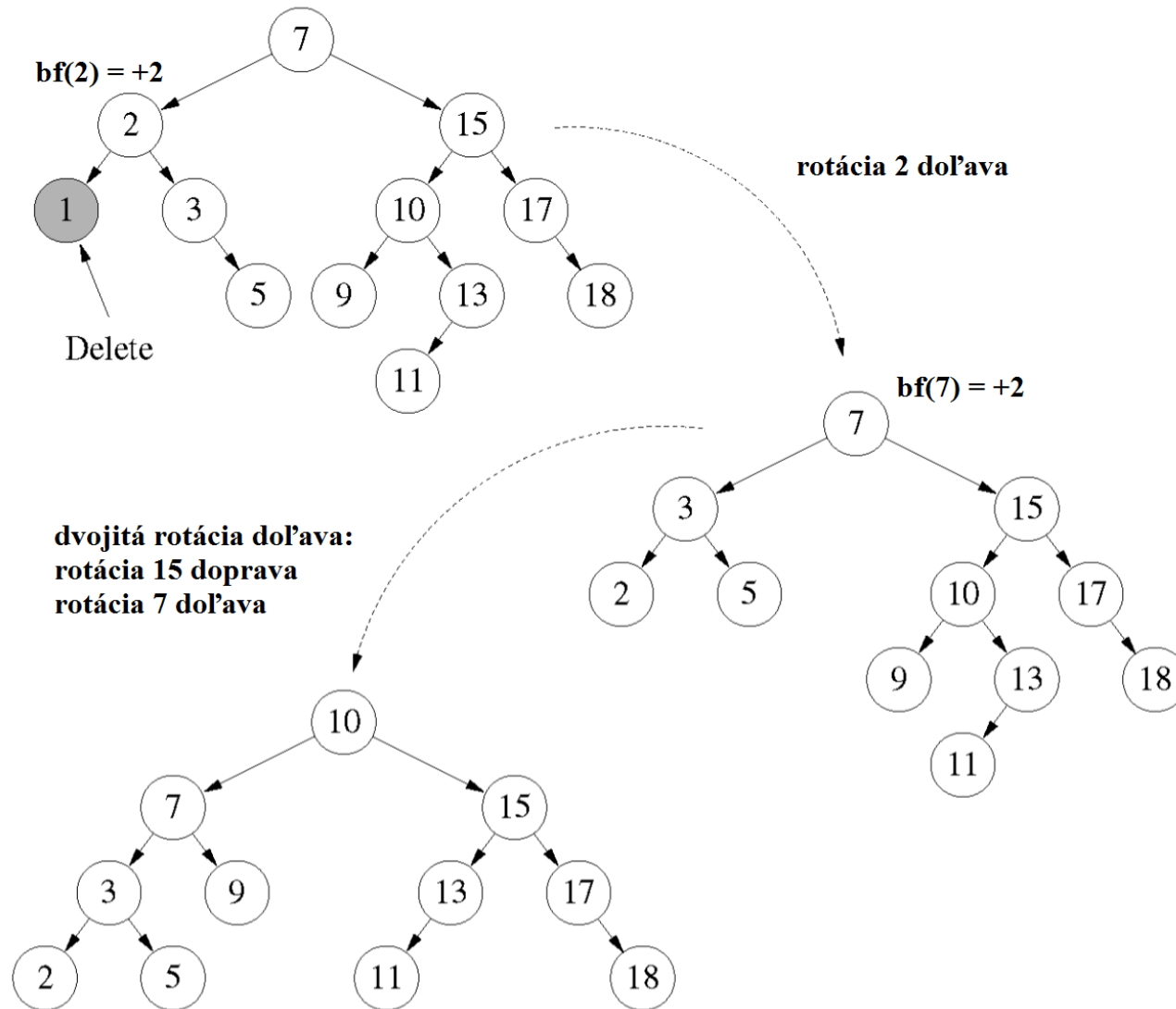
Delete(x) z AVL stromu (2)

- Uvažujme, že po odstránení prvku z ľavého podstromu, niektorý jeho predok (b) zostane nevyvážený, pričom bf jeho pravého dieťaťa (f) je -1: **dvojitá rotácia doľava**



(odstránenie z pravého podstromu je symetrická situácia: **dvojitá rotácia doprava**)

Delete z AVL stromu – Ukážka



Implementácia AVL stromov

- Rozšírenie štruktúry vrcholu o hodnotu **výšky** vrcholu
- Opakovanie – výpočet výšky vo vrchole realizujeme vychádzajúc z rekurzívnej definície:

$$\text{výška}(T) = \begin{cases} -1 & \text{ak podstrom } T \text{ je prázdný} \\ 1 + \max(\text{výška}(T_L), \text{výška}(T_R)) & \text{ak podstrom } T \text{ nie je prázdný} \end{cases}$$

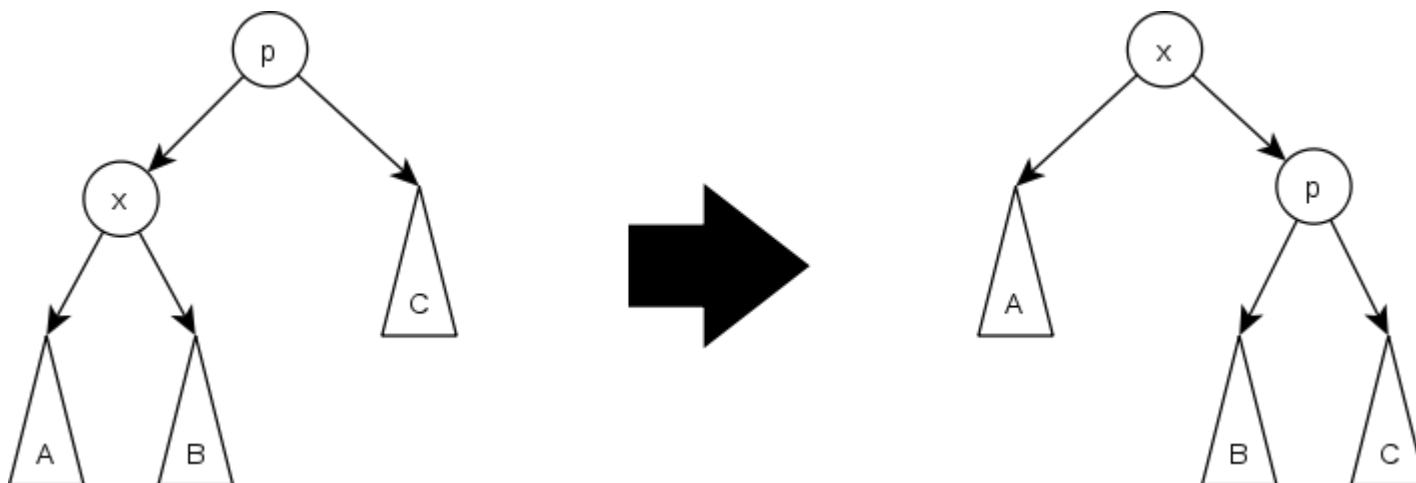
- Pri jednej rotácii sa zmenia výšky vrcholov „zachytených v rotáciach“ na ceste do koreňa
 - Najviac $O(\log n)$ zmien hodnôt výšok



Operácia splay(x) – presunúť hodnotu x do koreňa

- Rotácie nám umožňujú upravovať štruktúru stromu
- Ako dostať x do koreňa?
- Tri prípady:

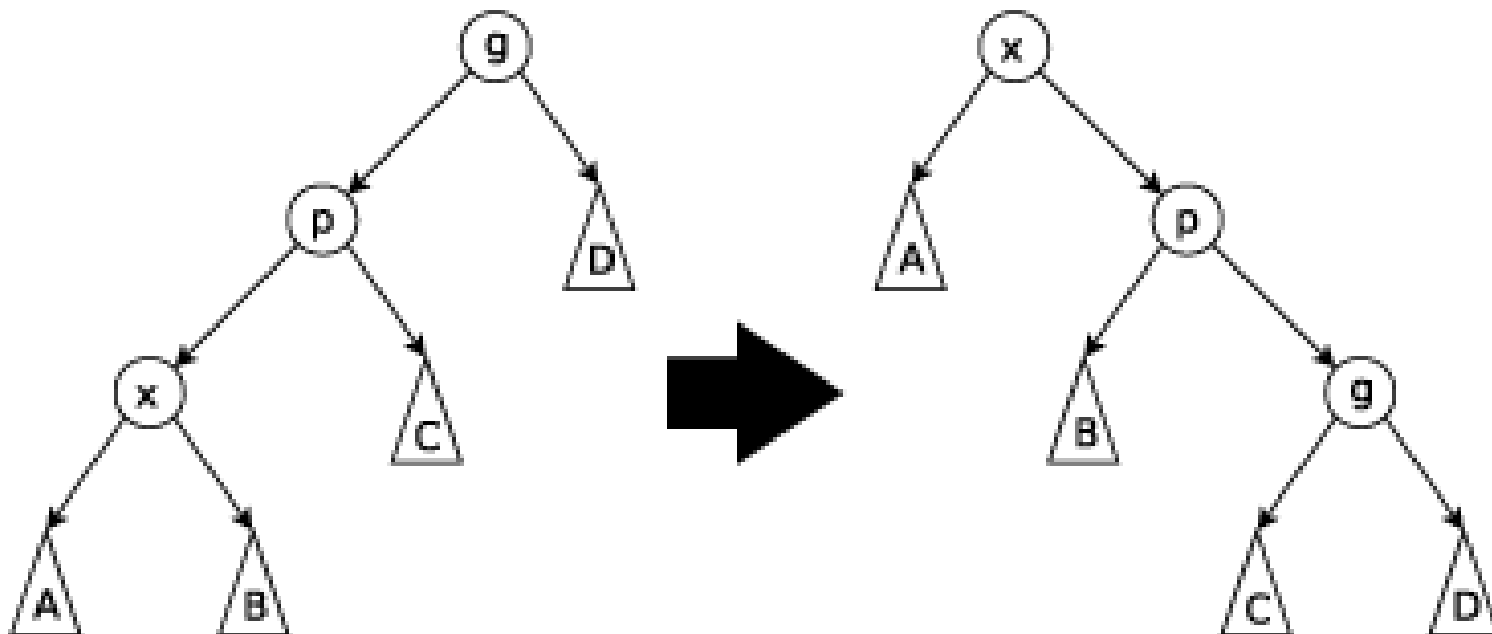
I. Parent(x) je koreň a x ľavé dieťa – **rotácia p doprava**
(zig)



(ak je x pravé dieťa je to symetrické: **rotácia p doľava**)

Operácia splay(x) – presunúť hodnotu x do koreňa

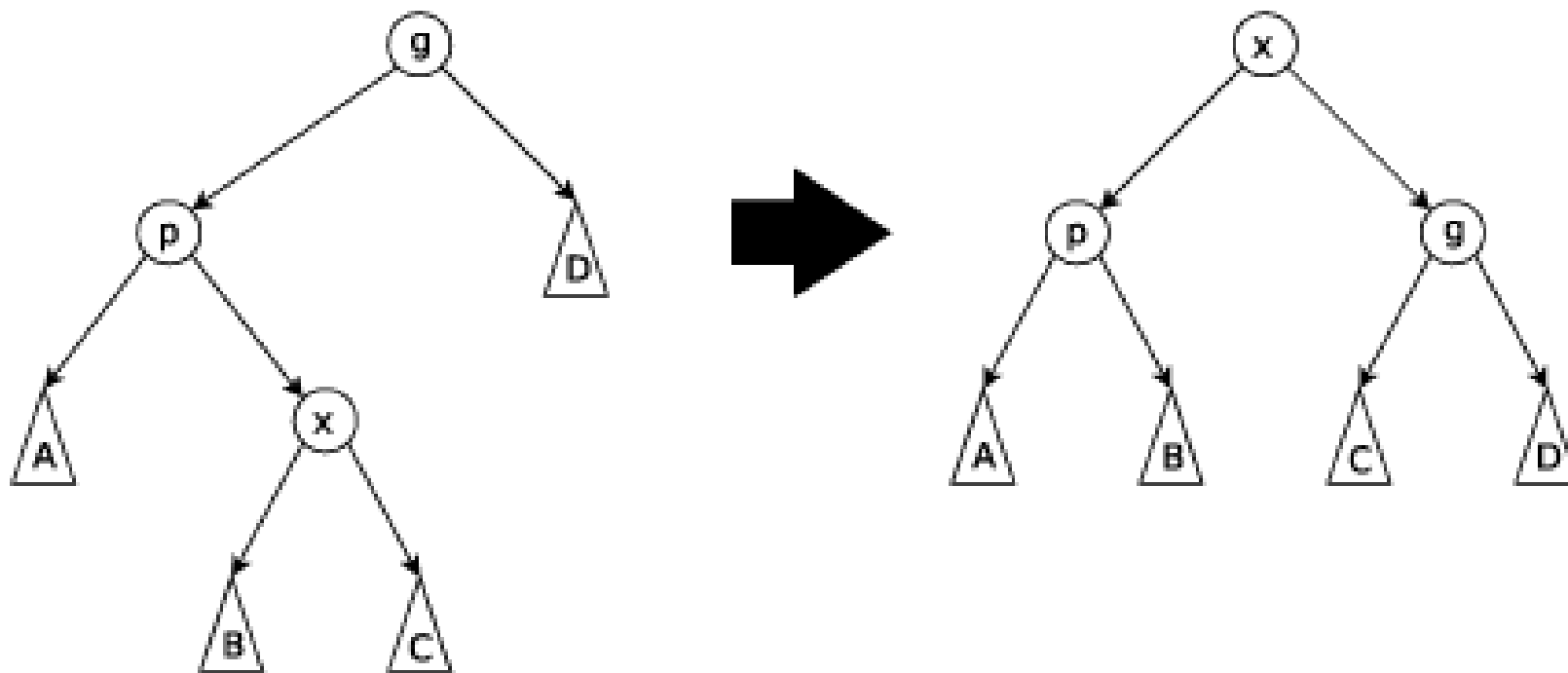
2. Parent(x) nie je koreň a Parent(x) a x sú obe ľavé deti – **2x rotácia doprava (zig-zig)**



(ak sú obe pravé deti je to symetrické: **2x rotácia doľava**)

Operácia splay(x) – presunúť hodnotu x do koreňa

3. Parent(x) nie je koreň a Parent(x) je ľavé dieťa a x je pravé dieťa, alebo opačne – **rotácia p doláva** a **rotácia g doprava** (zig-zag)



(analogicky v symetrickej situácii)

Ďalšie operácie nad BVS

- **insertRoot(x)** – vloženie hodnoty do koreňa:
 - insert(x) – štandardné vloženie x
 - splay(x) – presun hodnoty do koreňa
 - Zložitosť $O(\log n)$
- **join(a,b)** – spojenie dvoch vyhľadávacích stromov A a B
- **split(x)** – rozdelenie stromu na dva stromy, prvý s hodnotami $< x$, a druhý s hodnotami $> x$
- **Splay strom** je strom, v ktorom po každej operácii s x vykonáme splay(x), teda hodnota x sa dostane do koreňa
 - Štruktúra sa dynamicky prispôsobuje vykonávaným operáciám
 - Operácia pracujú v amortizovanej zložitosti $O(\log n)$

Optimálny binárny vyhľadávací strom

- Najmenší možný počet vykonaných krokov pre danú postupnosť vykonávania operácií (search, insert, delete)
- **Statická optimalita**
 - Pre dané pravdepodobnosti s akými sa budú vyhľadávať prvky, strom musí mať najnižšiu očakávanú zložitosť search operácií
 - Staticky optimálny strom sa dá zostrojiť dynamickým programovaním v čase $O(n^2)$ – využitím vážených dĺžok ciest
- **Dynamická optimalita**
 - Pre danú postupnosť prístupov k prvkom $X = x_1, \dots, x_m$ chceme minimalizovať celkový počet operácií (posun doľava/doprava, posun do rodiča, jednoduchá rotácia) – minimálny počet $OPT(X)$
 - Domnienka: **Splay stromy sú dynamicky optimálne – vykonajú $O(OPT(x))$ operácií.**

Váhovo vyvážené stromy

- Pôvodný názov: stromy s ohraničenou vyváženosťou (tzv. **bounded balance trees – BB[α]**)
- Vyváženie podľa váhy (počet prvkov v podstrome) tzv. **weight-balanced trees**
- Hodnoty váhy vo vrcholoch upravujeme pri každej operácii ktorá mení štruktúru stromu (insert, delete)
- Štruktúru stromu upravujeme jednoduchými rotáciami a dvojitými rotáciami (podobne ako pri vyvažovaní podľa výšky), ak je strom vo vrchole nevyvážený

Váhovo vyvážené stromy (2)

- Udržiadať vyváženosť podstromov na presnosť ± 1 ako pri vyvážení podľa výšky je náročnejšie ako $O(\log n)$
- Ohraničíme **pomer váhy** podstromu (ľavého aj pravého) k celkovej váhe vo vrchole:

$$\alpha \cdot \text{váha}(x) \leq \text{váha}(\text{Podstrom}(x)) \leq (1 - \alpha) \cdot \text{váha}(x)$$

- Strom výšky aspoň 2 má aspoň $\left(\frac{1}{1-\alpha}\right)^h$ listov
- Výška stromu:

$$h \leq \log_{\frac{1}{1-\alpha}} n = \frac{\log_2 n}{\log_2 \left(\frac{1}{1-\alpha}\right)} = O(\log n)$$

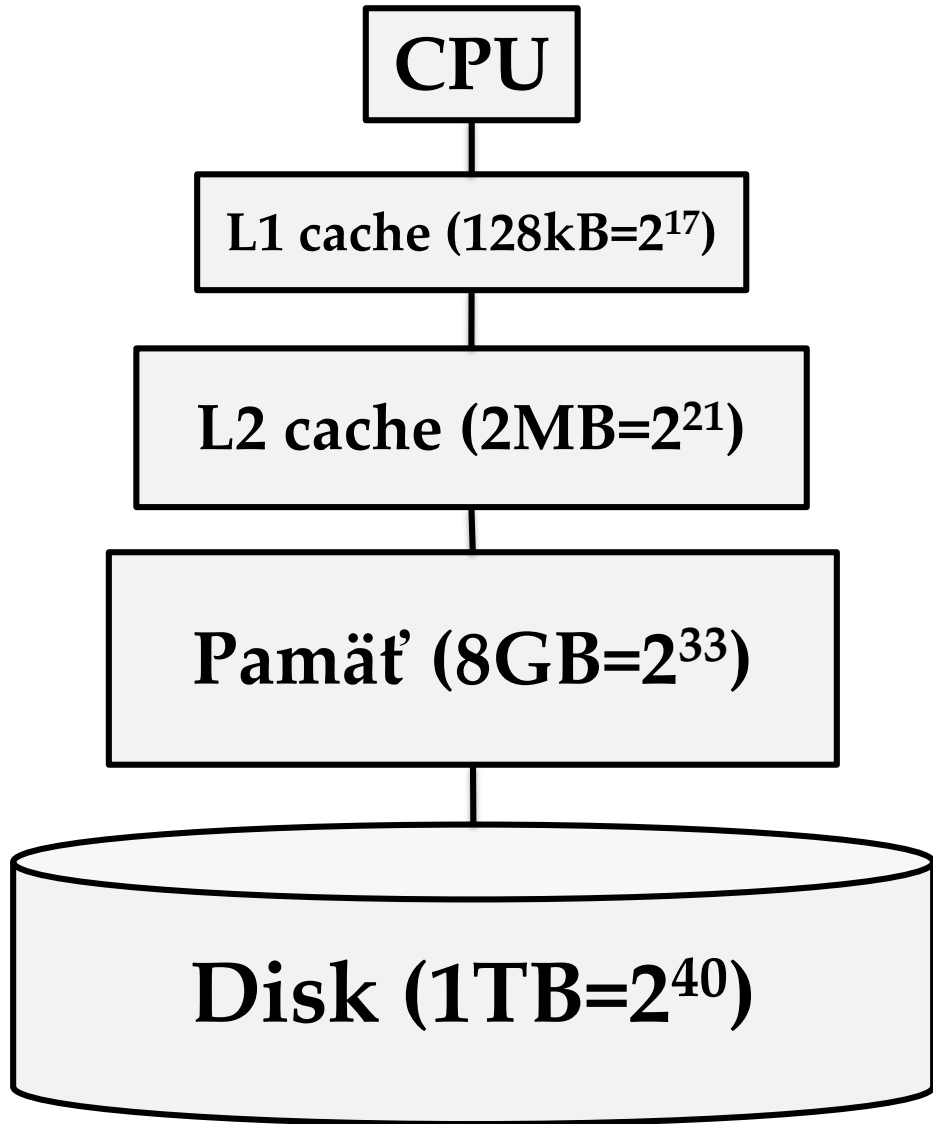
- Dobré hodnoty pre vyváženie: $\frac{2}{11} < \alpha < 1 - \frac{1}{\sqrt{2}}$



Čo keď máme veľa dát?

- Doteraz sme predpokladali, že sa dátová štruktúra zmestí do pracovnej pamäte
- Uvažujme, že máme tak veľa údajov, že ich nie je možné všetky naraz vložiť do pamäte
- Na uloženie musíme použiť pevný disk
 - čas vykonania podstatne narastie
- Diskové operácie (načítanie a zápis) sú výrazne pomalšie ako operácie s pamäťou

Aké rýchle sú operácie v počítači?



- 4GHz procesor $\approx 2^{32}/s$
- Načítať dáta do L1 cache trvá 2 inštrukcie: $\approx 2^{31}/s$
- Načítať dáta do L2 cache trvá 30 inštrukcií: $\approx 2^{27}/s$
- Načítať dáta do pamäte trvá 250 inštrukcií: $\approx 2^{24}/s$
- Načítať dáta z nového miesta na disku trvá asi 8M inštrukcií: $\approx 2^9/s$

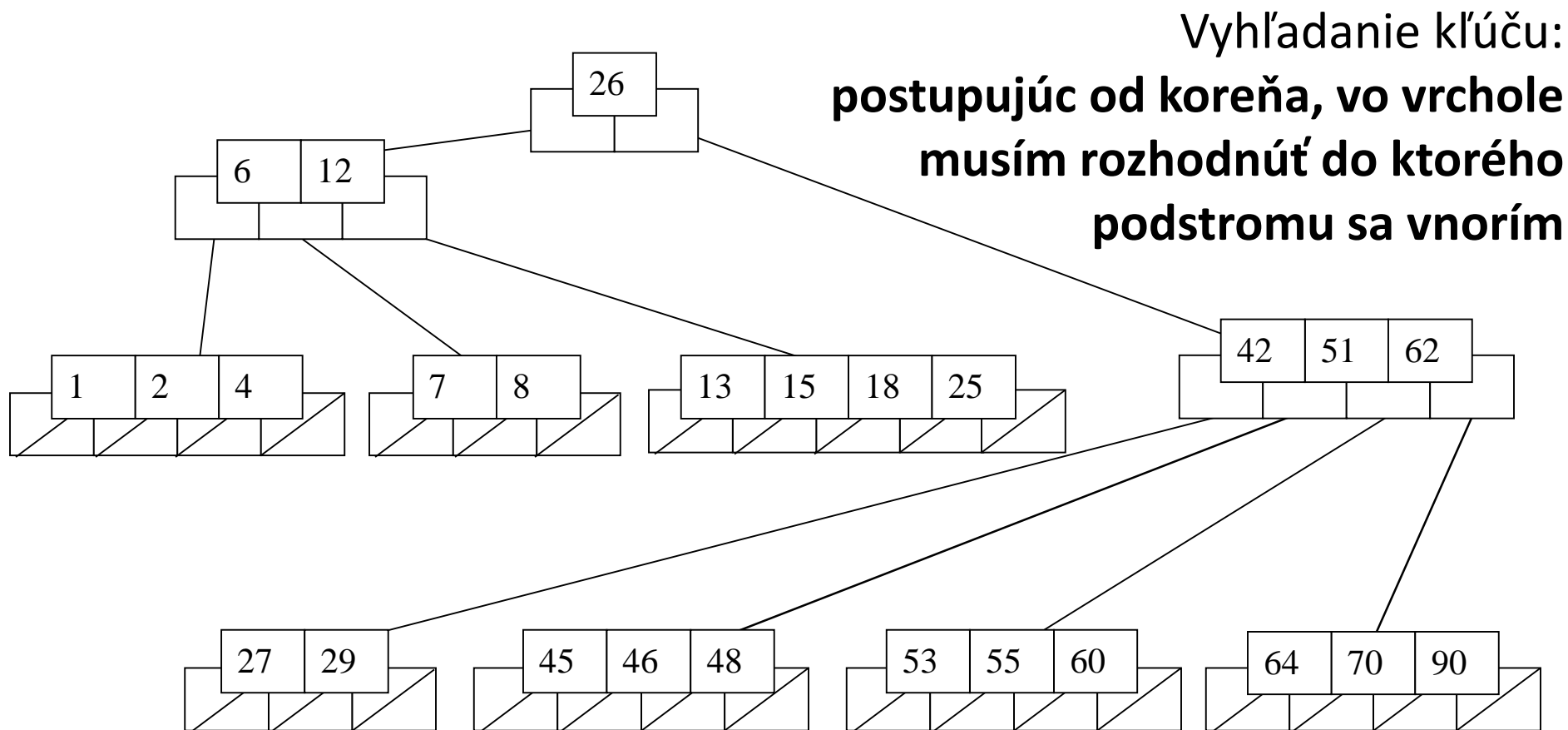
Náročnosť operácií

- Rýchlejšie ako JEDEN prístup na disk je:
 - 2 milióny aritmetických operácií,
 - 1000 L2 cache prístupov,
 - 200 prístupov do pamäte.
- Uvažujme teraz AVL strom s $n = 2^{38}$ (256GB) prvkami,
 - potom, výška stromu je okolo $1,44 \cdot 38 = 55$,
 - čiže každá operácia search/insert/delete vyžaduje okolo 55 prístupov na disk, čo môže trvať aj 0,5 sekundy, alebo
 - **asi 30 vykonaní operácií za minútu**
- Z disku sa číta po **veľkých** blokoch
 - Každý vrchol AVL stromu môže byť v inom bloku

B-strom (B-tree)

- Myšlienka: **vo vrcholech sa budeme viac vetviť**
 - Vyšší stupeň vetvenia vo vrcholech = menšia výška stromu
- B-strom rádu m je strom, v ktorom každý vrchol môže mať najviac m detí, pričom:
 - počet kľúčov vo vrchole je o 1 menší než počet jeho detí (kľúče rozdeľujú intervaly kľúčov v podstromoch)
 - všetky listy sú v rovnakej hĺbke
 - každý vnútorný vrchol okrem koreňa má aspoň $\left\lceil \frac{m}{2} \right\rceil$ detí
 - koreň je buď list alebo má 2 až m potomkov
 - list obsahuje najviac $m - 1$ kľúčov
- Rád B-stromu m je nepárne číslo

B-strom rádu 5 obsahujúci 26 prvkov – Ukážka



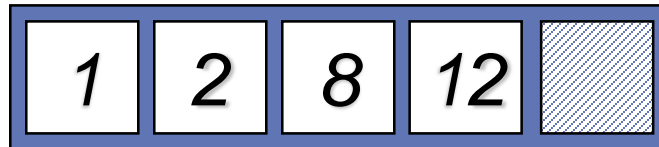
Všetky listy sú v (rovnakej) hĺbke 2

Insert do B-stromu

- Vyhľadám list, do ktorého by nový kľúč mal patriť
- Vložíme nový kľúč do tohto listu
- Opravím chyby:
 - Ak by list už obsahoval príliš veľa kľúčov, rozdelíme ho na dva vrcholy, a prostredný kľúč posunieme vyššie (do rodiča)
 - Rekurzívne pokračuj až do koreňa:
Ak by rodič už obsahoval príliš veľa kľúčov, rozdel' ho a stredný kľúč posun vyššie
 - V prípade koreňa (ak je to potrebné): koreň rozdelíme na dva vrcholy, a prostredný kľúč vytvorí nový koreň, čím sa zvýši celková výška B-stromu

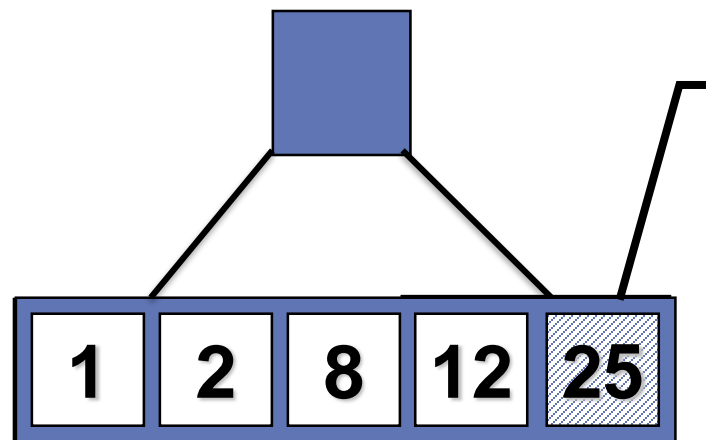
Konštrukcia B-stromu – Ukážka

- Na vstupe máme prvky: 1, 12, 8, 2, 25, 6, 14, 28, 17, 7, 52, 16, 48, 68, 3, 26, 29, 53, 55, 45
- Chceme zostrojiť B-strom rádu 5
- Vkladáme prvky po jednom
- Po vložení 4 prvkov bude koreň:



- Pridanie piateho prvku (25) by porušilo podmienky B-stromu rádu $m=5$, takže vrchol rozdelíme, strednú hodnotu (8) povýšime na nový koreň ...

Insert 25 do B-stromu rádu 5

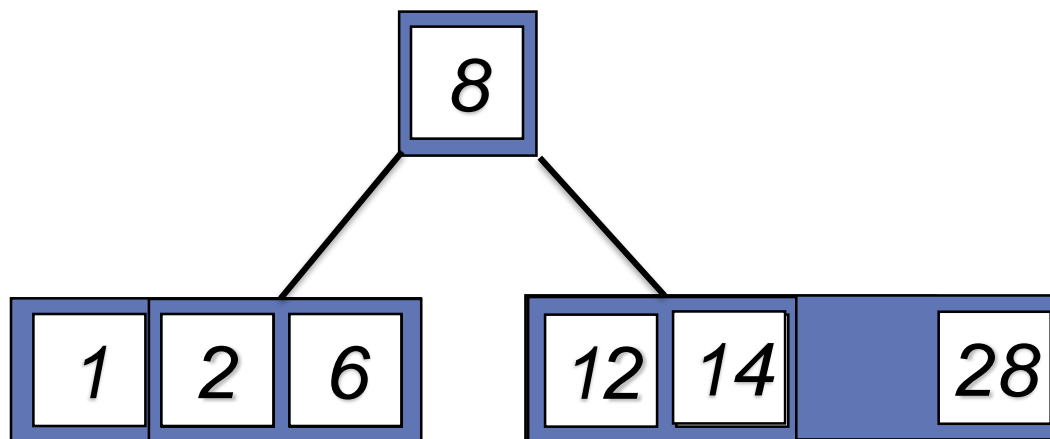
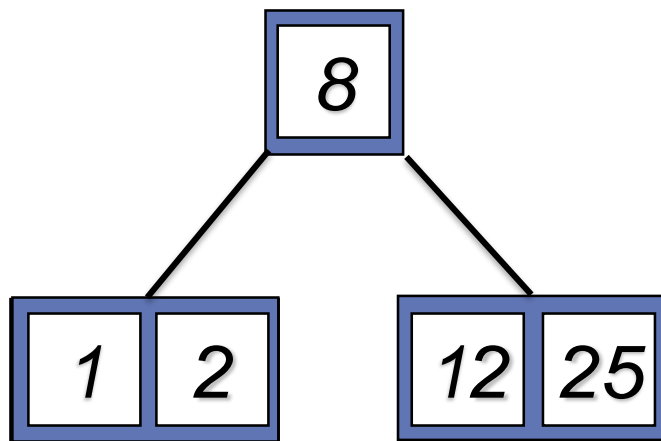


Presahuje rád $m=5$.

Rozdeliť vrchol a
povýšiť stredný prvok
(8) na nový koreň.

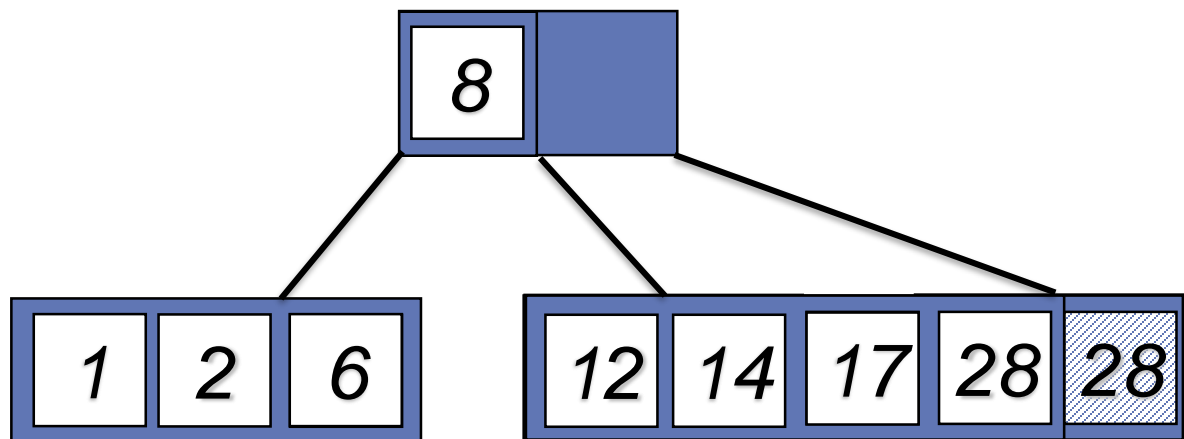
Insert 6, 14, 28 do listov

1
12
8
2
25
6
14
28
17
7
52
16
48
68
3
26
29
53
55
45



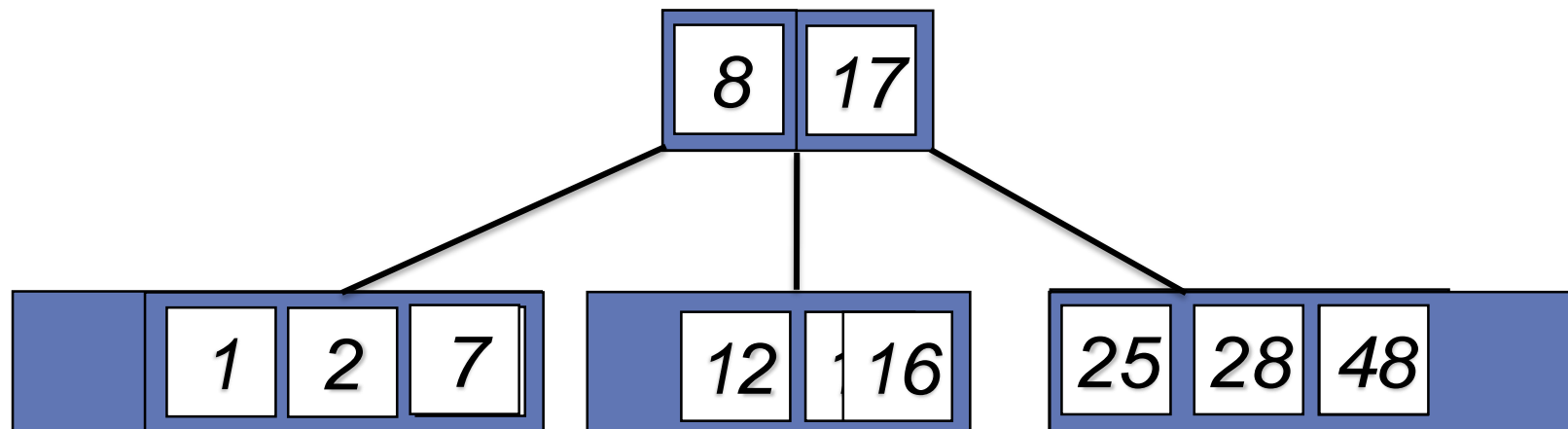
Insert 17 do B-stromu

Vložíme 17 do pravého listu, keďže bude už obsahovať príliš veľa prvkov, rozdelíme podľa strednej hodnoty, ktorú posunieme do rodiča



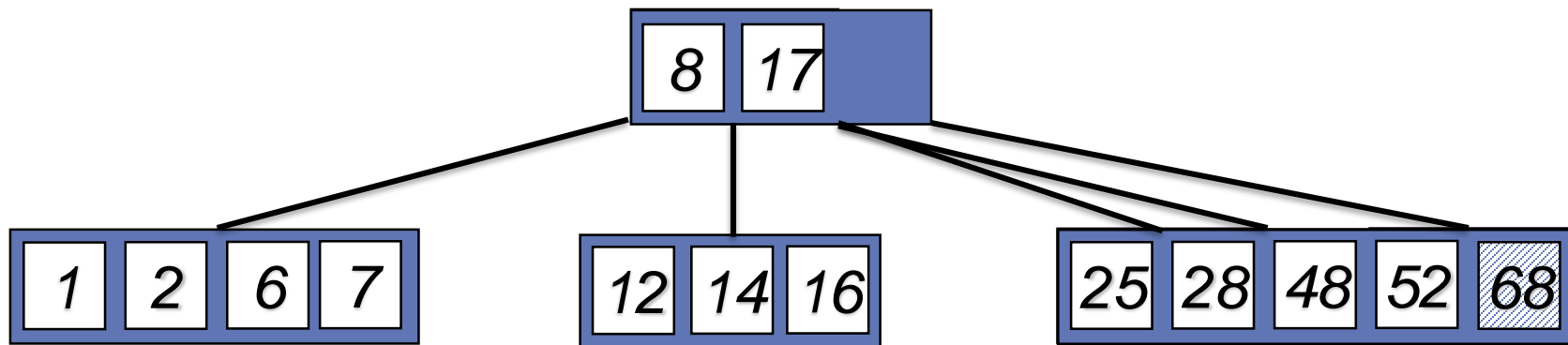
Insert 7, 52, 16, 48 do listov

1
12
8
2
25
6
14
28
17
7
52
16
48
68
3
26
29
53
55
45



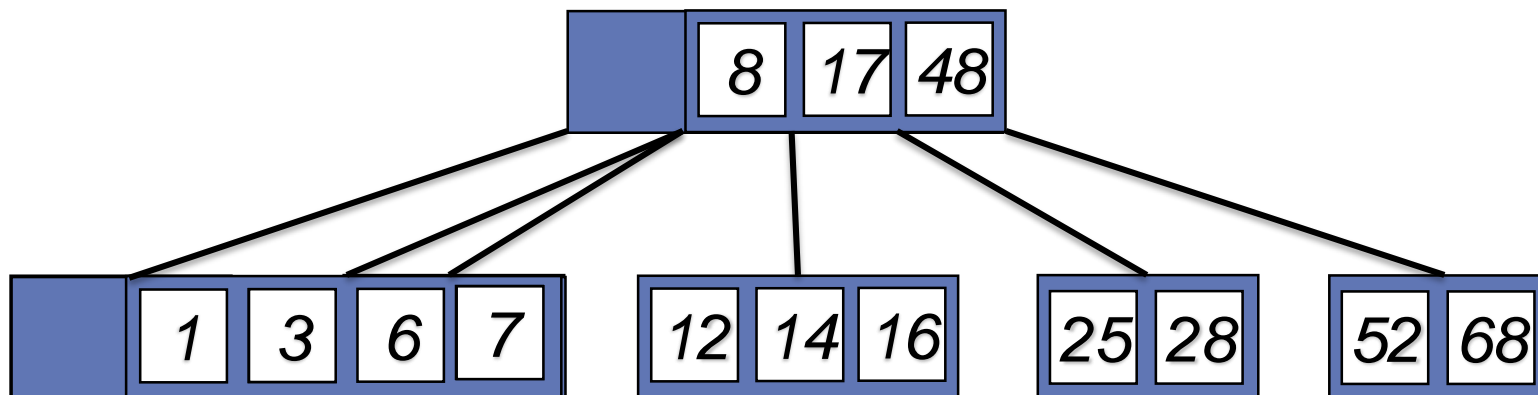
Insert 68 do B-stromu

Vloženie 68 spôsobí rozdelenie pravého listu,
prvok 48 bude povýšený do rodiča

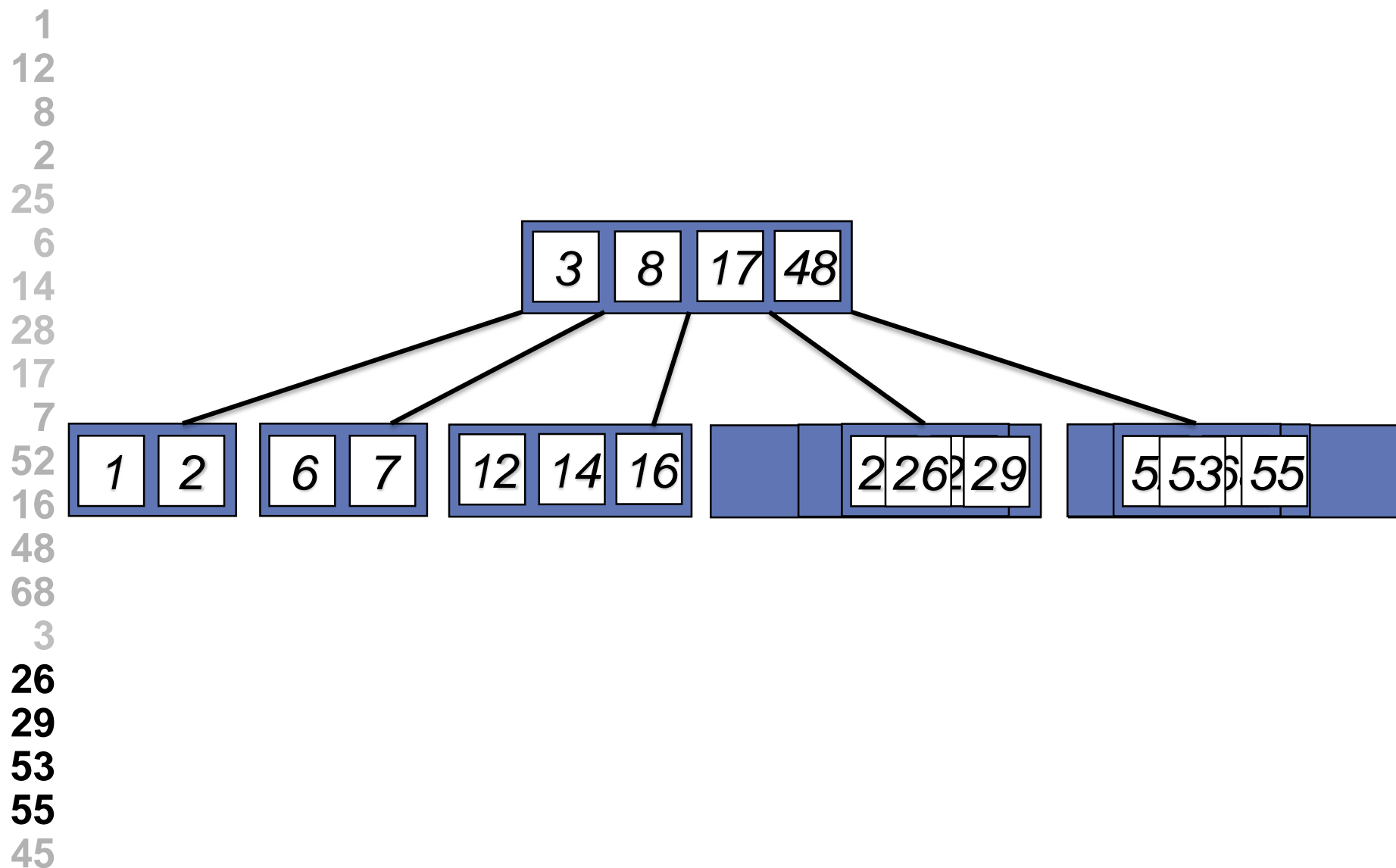


Insert 3 do B-stromu

Vloženie 3 spôsobí rozdelenie ľavého listu



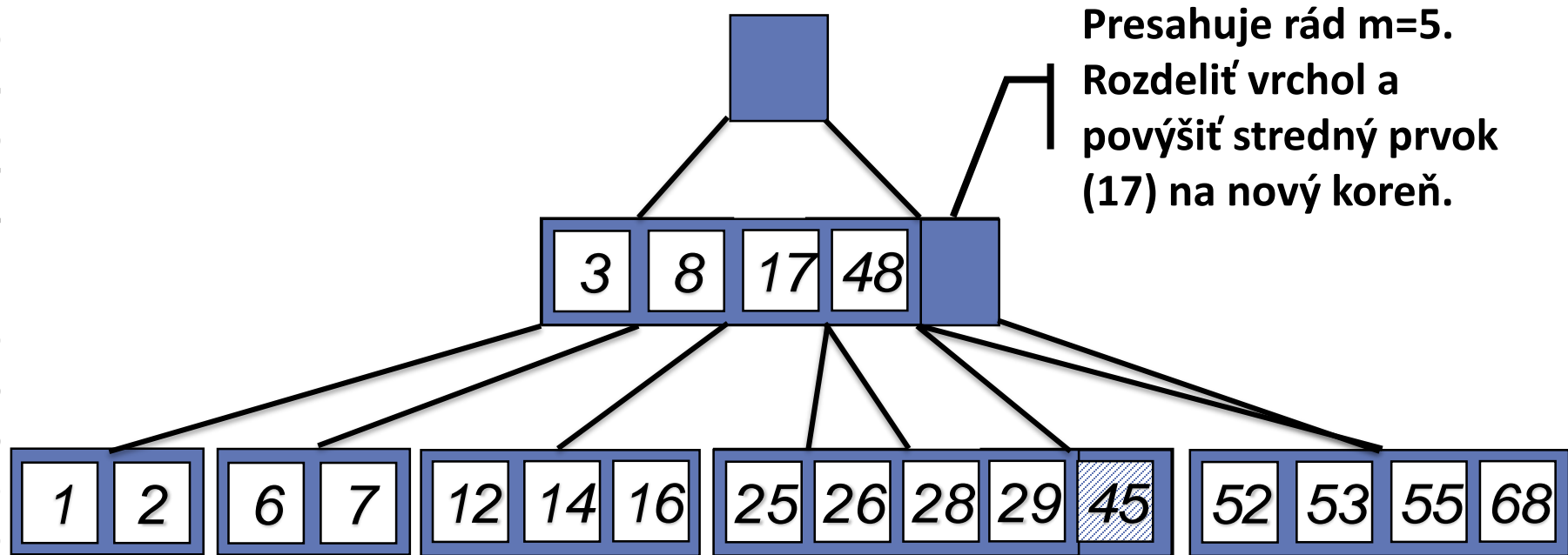
Insert 26, 29, 53, 55 do listov



Insert 45 do B-stromu

Vloženie 45 zvýši výšku celého stromu

1
12
8
2
25
6
14
28
17
7
52
16
48
68
3
26
29
53
55
45



Presahuje rád $m=5$.
Rozdeliť vrchol a
povýšiť stredný prvok
(17) na nový koreň.

Presahuje rád $m=5$.
Rozdeliť a povýšiť
stredný kľúč do rodiča.

Delete z B-stromu

- Odstránenie kľúča X zo stromu
- Ak kľúč nie je v liste, tak predchádzajúci alebo nasledujúci prvok P stromu je v liste, odstránime X a prvok P povýšime na miesto X
- Ak je kľúč X v liste:
 - Ak ho môžeme odstrániť bez toho, aby v liste zostalo príliš málo prvkov, odstránime ho.
 - Ak máme list, v ktorom po odstránení X zostane príliš málo kľúčov, hľadáme možnosti, ako by sme listy-súrodencov spojili (a v rodičovi prípadne znížili počet kľúčov) ...
 - Ak v rodičovi zostane málo kľúčov pokračujeme rekurzívne do koreňa

Zložitosť operácií nad B-stromom

- Podstatné sú prístupy na disk
- Maximálny počet prvkov v B-strome rádu m a výšky h :

root	$m - 1$
level 1	$m(m - 1)$
level 2	$m^2(m - 1)$
• • •	
level h	$m^h(m - 1)$



- Celkový počet prvkov:

$$N_{m,h} = (1 + m + m^2 + m^3 + \dots + m^h)(m - 1) = \left[\frac{(m^{h+1} - 1)}{(m - 1)} \right] (m - 1) = m^{h+1} - 1$$

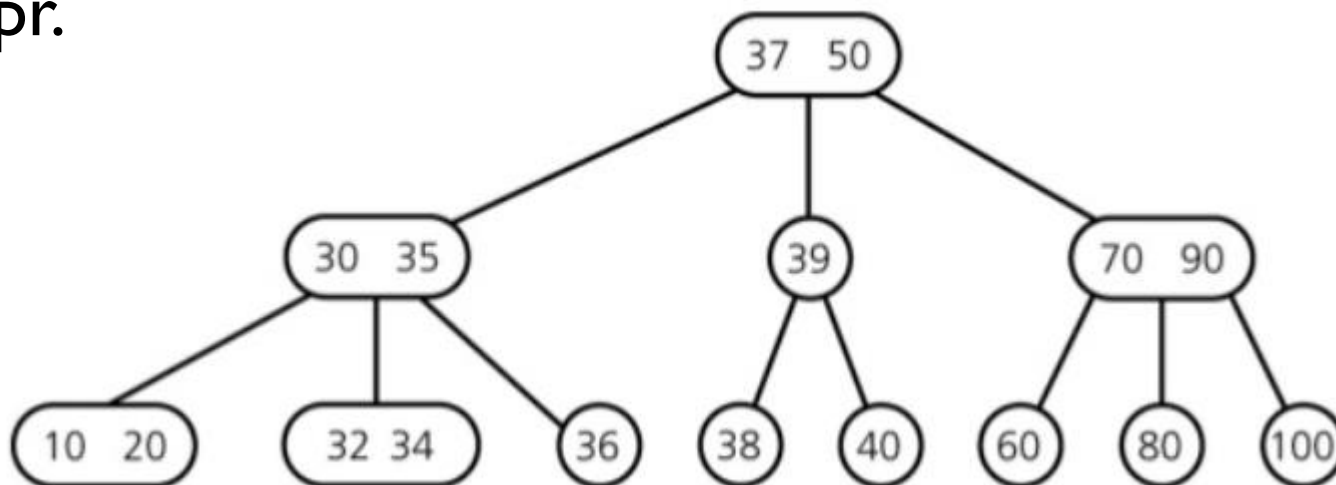
- Pre $m = 5, h = 2$ počet $N_{5,2} = 5^3 - 1 = 124$
- Pre $m = 101, h = 3$ počet $N_{101,3} = 101^4 - 1 = \text{cca } 100\text{M}$

(a,b) stromy

- Stupeň vrchola B-stromu je od **a** do **$b=2a-1$**
 - Pri insert/delete je potrebných $\Theta(\log n)$ úprav blokov
- Ak umožníme ešte väčší stupeň vetvenia (**$b \geq 2a$**), tak vyvažovanie pracuje efektívnejšie – **(a,b) stromy**:
 - Pri insert/delete postačuje upraviť amortizovane **$O(1)$** blokov
- V praxi preferovaný typ stromu v porovnaní s klasickým B-stromom

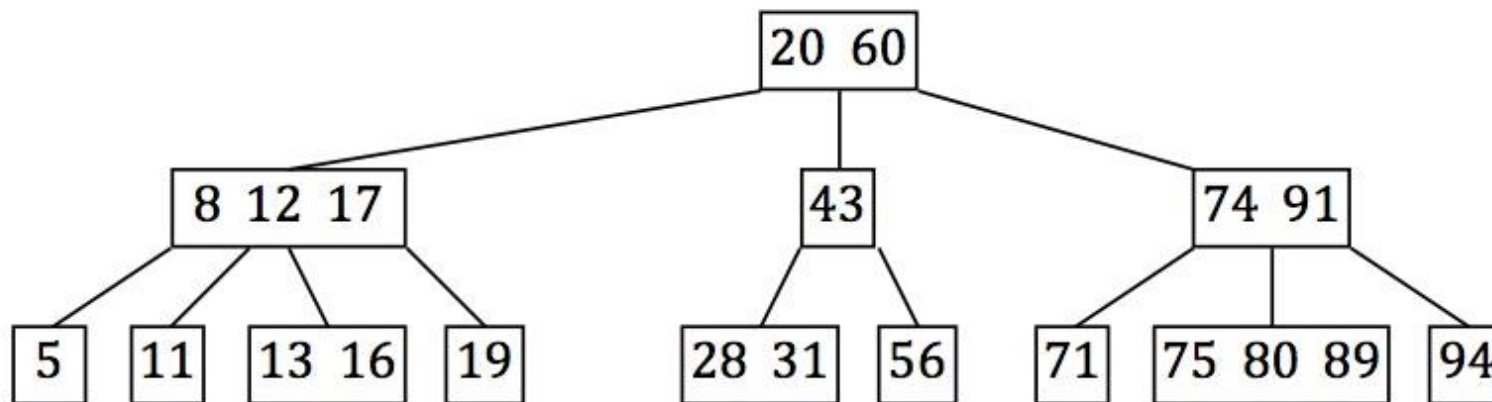
2-3 stromy (2-3 trees)

- Špeciálny prípad B-stromu pre $m=3$
 - Každý vnútorný vrchol má dve alebo tri deti
 - Všetky listy sú v rovnakej hĺbke a každý obsahuje najviac 2 hodnoty
- B-stromy sú vždy vyvážené, takže 2-3 strom je dobrý príklad vyváženého vyhľadávacieho stromu
- Napr.



2-3-4 stromy (2-3-4 trees)

- Špeciálny prípad B-stromu, resp. (a,b) stromu pre (2,4)
 - Každý vnútorný vrchol má dve, tri alebo štyri deti
 - Všetky listy sú v rovnakej hĺbke a každý obsahuje najviac 3 hodnoty



- Insert je možné spraviť jedným prechodom od koreňa
 - Vždy, keď prechádzame cez 4-vrchol, rozdelíme ho na dva 2-vrcholy
 - 2-3 strom môže (po vložení do listu) vyžadovať spätný prechod

Insert do 2-3-4 stromu

- Nájdi list, do ktorého sa bude hodnota vkladat’.
- Počas hľadania, keď narazíš na 4-vrchol, tak ho rozbal’.
- Ak je list, do ktorého vkladáme 2-vrchol alebo 3-vrchol, tak vlož do listu.
- Ak je list (po vložení) 4-vrchol, tak ho rozbal’ tak, že prostrednú hodnotu vlož do rodiča a vkladajú hodnotu vlož do príslušného listu.
 - Miesto v rodičovi sa určite nájde, keďže sme pri ceste dole rozbalili všetky 4-vrcholy. Preto nemusíme rekurzívne postupovať hore ako v prípade 2-3 stromov.

Farebné stromy

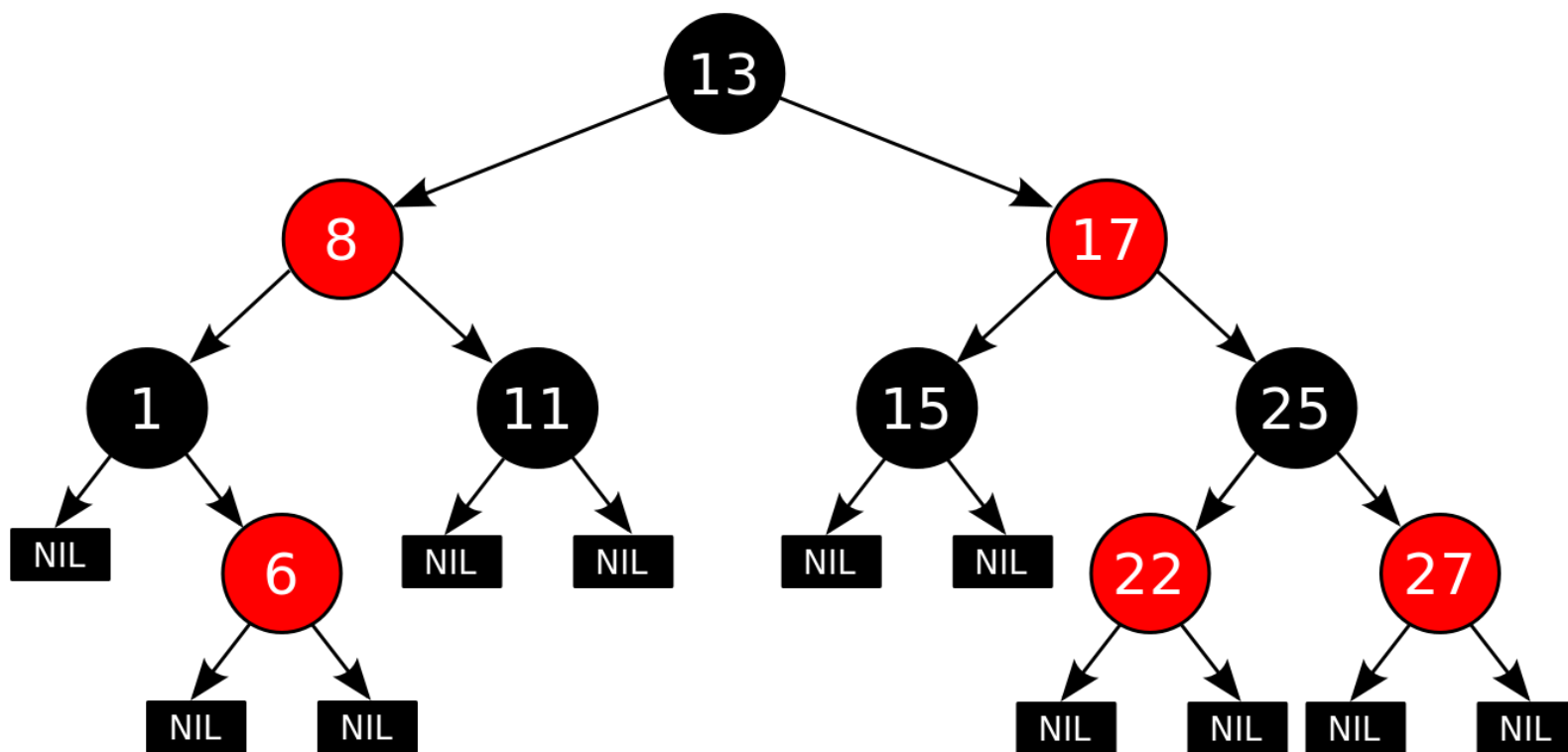
- Binárne stromy sú implementačne výhodné
 - Dajú sa využiť operácie ako nad štandardným BVS
- Vrcholy obsahujúce viac ako jeden kľúč sú implementačne komplikované
 - Operácie majú veľa špeciálnych prípadov
- Návrh: **farbenie vrcholov binárneho stromu dvomi farbami**
- Aké farby zvolíme? **červenú** – **čiernu**
 - (1978) **červená** bola najkrajšia farba, ktorú vedeli farebné laserové tlačiarne vytlačiť :)
 - Dobrá dostupnosť **červených** a **čiernych** pier na kreslenie na papier ...

Červeno-čierny strom (red-black tree)

- Binárny vyhľadávací strom s vrcholmi ofarbenými na **červeno** alebo **čierno**, taký že:
 1. Koreň je **čierny**
 2. Listy neobsahujú dáta a sú **čierne**
 3. Cesty z koreňa do listov majú rovnaký počet **čiernych** vrcholov a tento počet označujeme čierna výška stromu
 4. Ak je vrchol **červený**, tak jeho deti sú **čierne**
- Vlastnosti:
 - Na žiadnej ceste nie sú dva **červené** vrcholy za sebou
 - Dĺžka cesty z koreňa do najvzdialenejšieho listu nie je viac ako dvakrát dlhšia ako cesta do najbližšieho listu
 - Každý vnútorný vrchol má dvoch potomkov

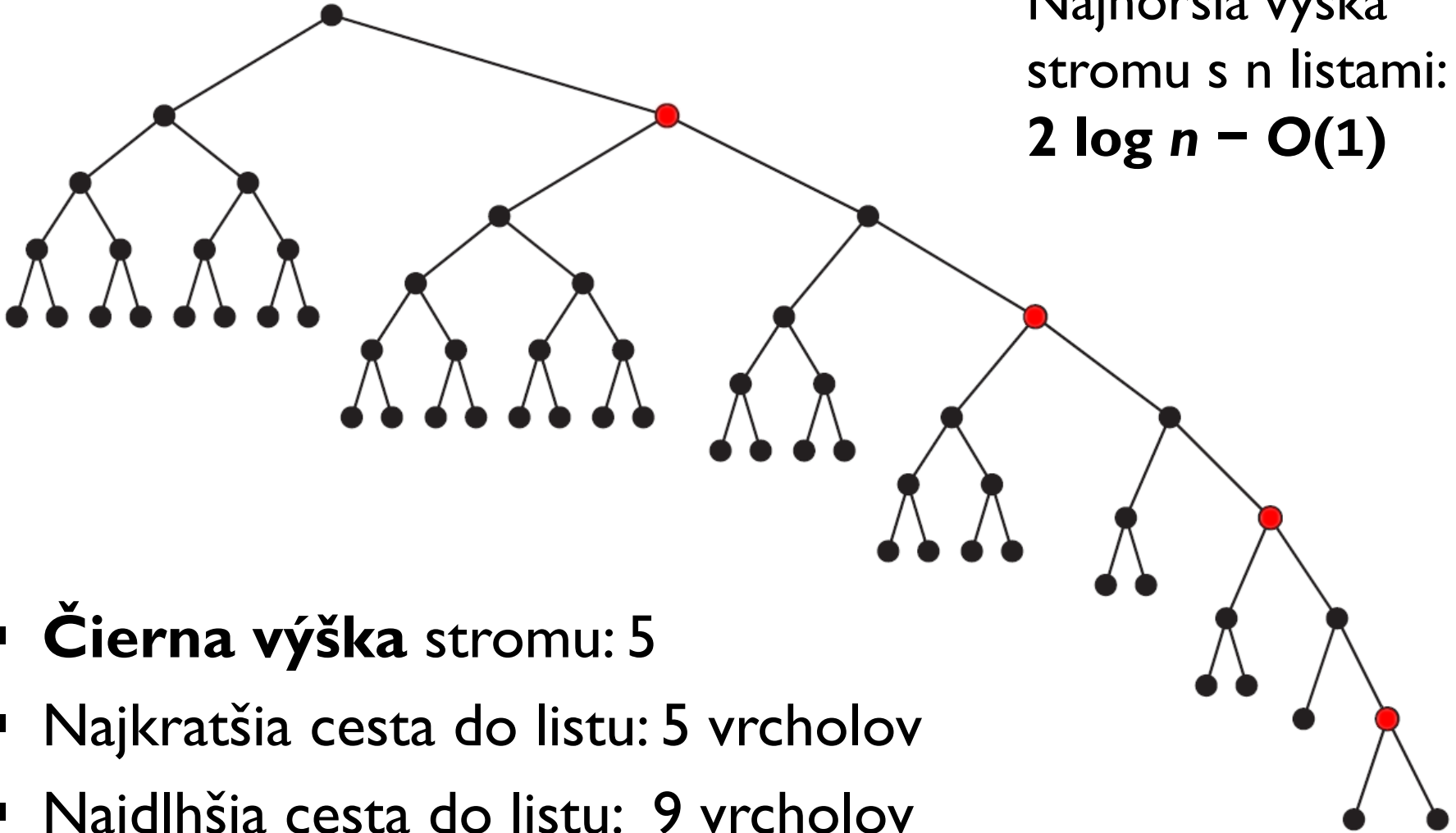
Červeno-čierny strom – Ukážka

- Čierna výška vrcholu – počet čiernych vrcholov na ceste z vrcholu do listu



Najmenej vyvážený **červeno**-čierny strom

Najhoršia výška
stromu s n listami:
 $2 \log n - O(1)$



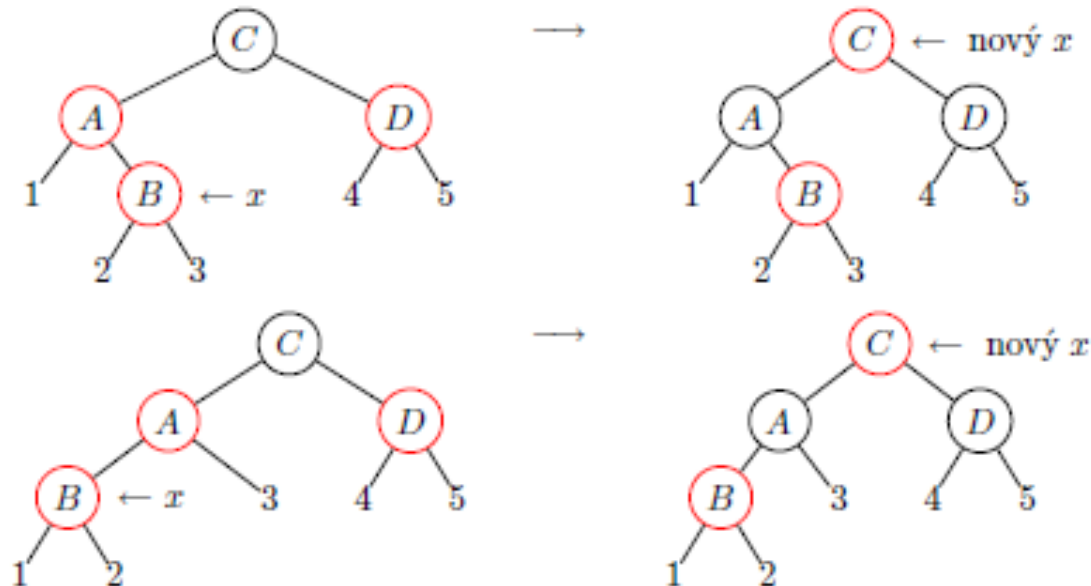
- **Čierna výška** stromu: 5
- Najkratšia cesta do listu: 5 vrcholov
- Najdlhšia cesta do listu: 9 vrcholov

Insert(x) do **červeno**-čierneho stromu

- Vrchol x vložíme ako do štandardného BVS a zafarbíme ho na **červeno**
- Ktorú vlastnosť stromov sme mohli týmto porušiť?
- Ak je x koreň, zafarbíme ho na **čierno**
- Ak je $\text{Parent}(x)$ **čierny**, strom je v poriadku
- Ak je $y = \text{Parent}(x)$ **červený**, tak (keďže nie je koreň) má $z = \text{Parent}(y)$, ktorý je **čierny**:
 1. súrodenec vrcholu y (strýko vrcholu x) je **červený**
 2. súrodenec vrcholu y (strýko vrcholu x) je **čierny**

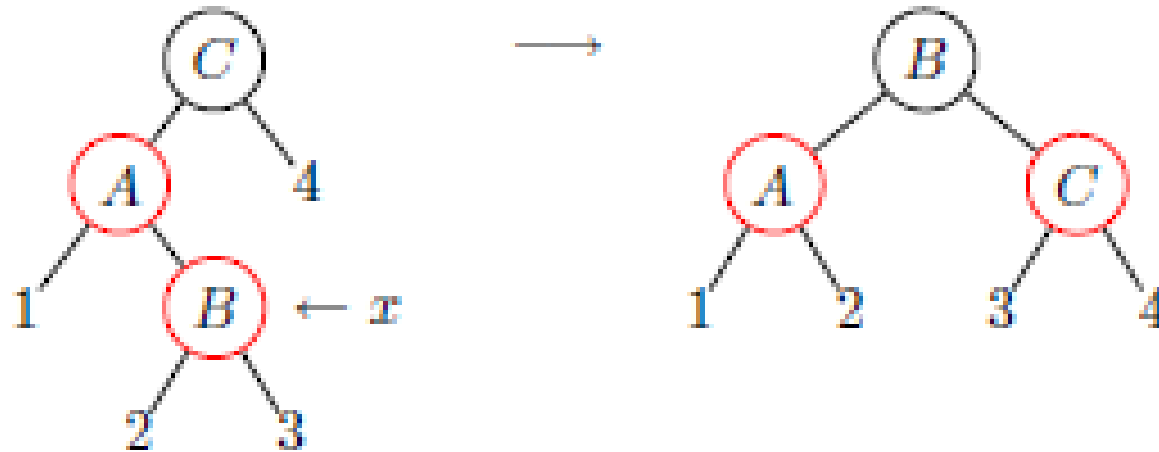
Insert(x) do **červeno**-čierneho stromu (2)

- $y = \text{Parent}(x)$ je **červený**, strýko vrcholu x je **červený**
- Vrcholy y a strýko x prefarbíme na **čierne**, $z = \text{Parent}(y)$ na **červeno**. Ak je z koreň alebo má **čierneho** rodiča, končíme, inak vyriešime „chybu“ rekurzívne vyššie. (Nakoniec koreň zafarbíme na **čierne**.)



Insert(x) do **červeno**-čierneho stromu (3)

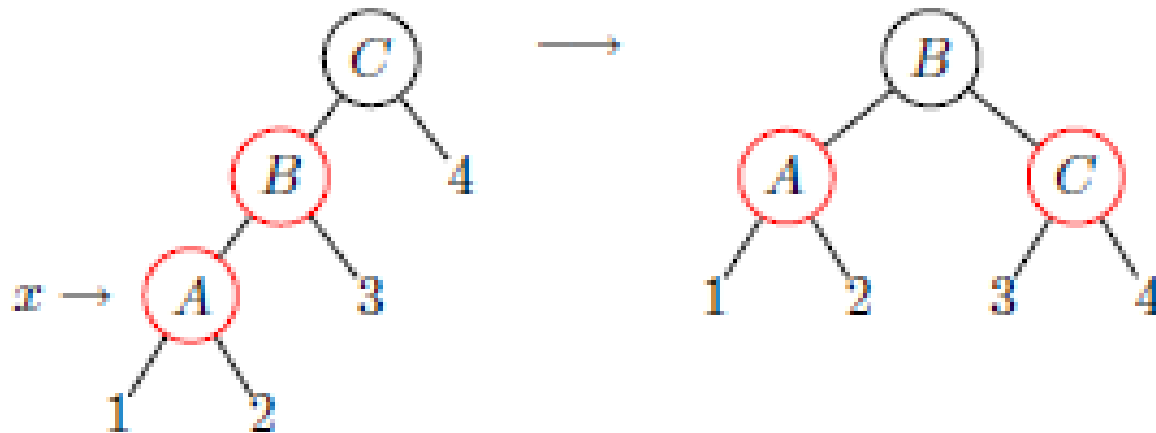
- $y = \text{Parent}(x)$ je **červený**, strýko vrcholu x je **čierny**
- Ak x je opačným dieťaťom y ako je y dieťaťom $z = \text{Parent}(y)$
- Uvažujme, že x je pravým dieťaťom a y je ľavým
rotácia y doľava, a **rotácia z doprava** a prefarbenie:



- Symetricky ak x je ľavým dieťaťom a y je pravým

Insert(x) do **červeno**-čierneho stromu (4)

- $y = \text{Parent}(x)$ je **červený**, strýko vrcholu x je **čierny**
- Ak x je rovnakým dieťaťom y ako je y dieťaťom $z = \text{Parent}(y)$
- Uvažujme, že x je ľavým dieťaťom a y je tiež ľavým
rotácia y doprava, a prefarbíme y na **čierno**, z na **červeno**:



- Symetricky ak sú pravými deťmi: **rotácia y doľava**

Insert do **červeno**-čierneho stromu – Ukážka

- Vlož 10

10

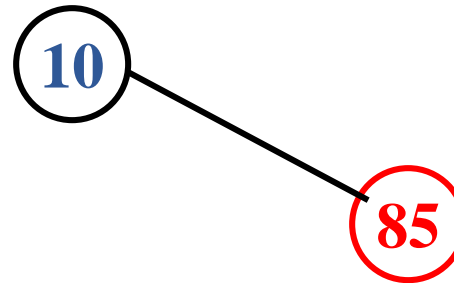
Insert do **červeno**-čierneho stromu – Ukážka

- Prefarbit' koreň na **čierno**



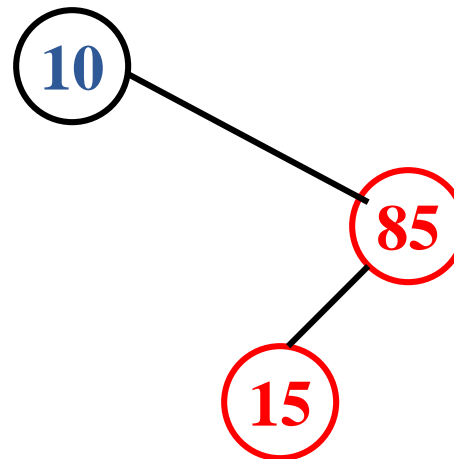
Insert do **červeno**-čierneho stromu – Ukážka

- Vlož 85



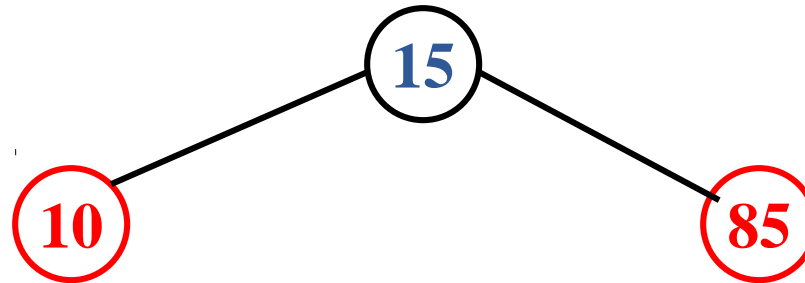
Insert do **červeno**-čierneho stromu – Ukážka

- Vlož 15



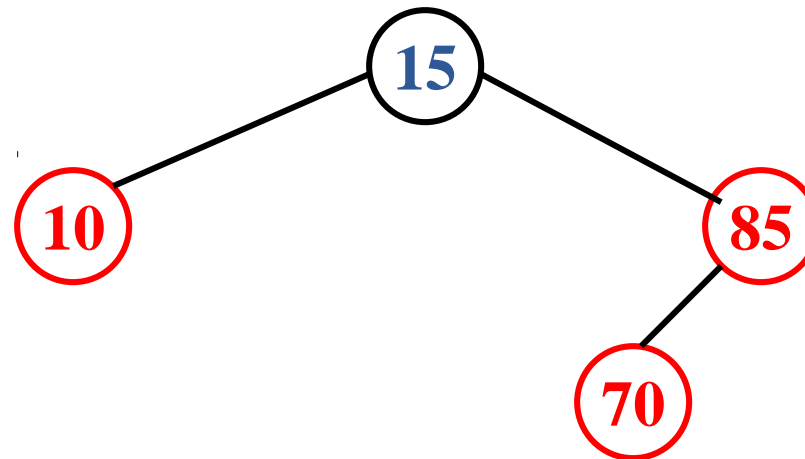
Insert do **červeno**-čierneho stromu – Ukážka

- Porušená podmienka!
 - Rotácia 85 doprava
 - Rotácia 15 doľava
 - Prefarbenie



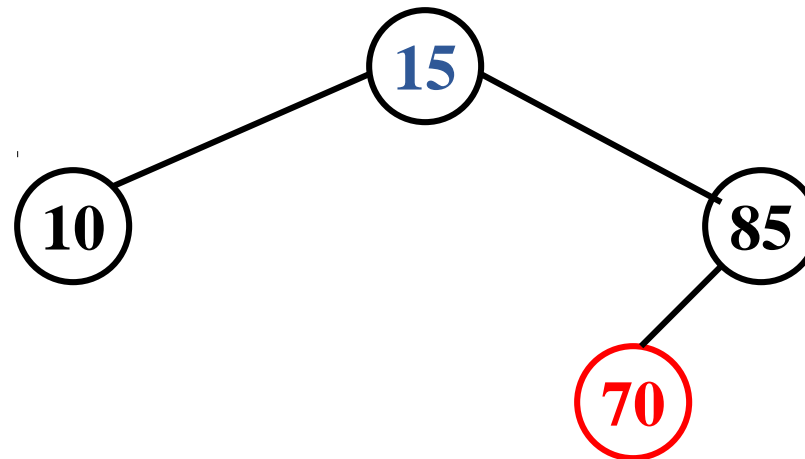
Insert do **červen**o-čierneho stromu – Ukážka

- Vlož 70



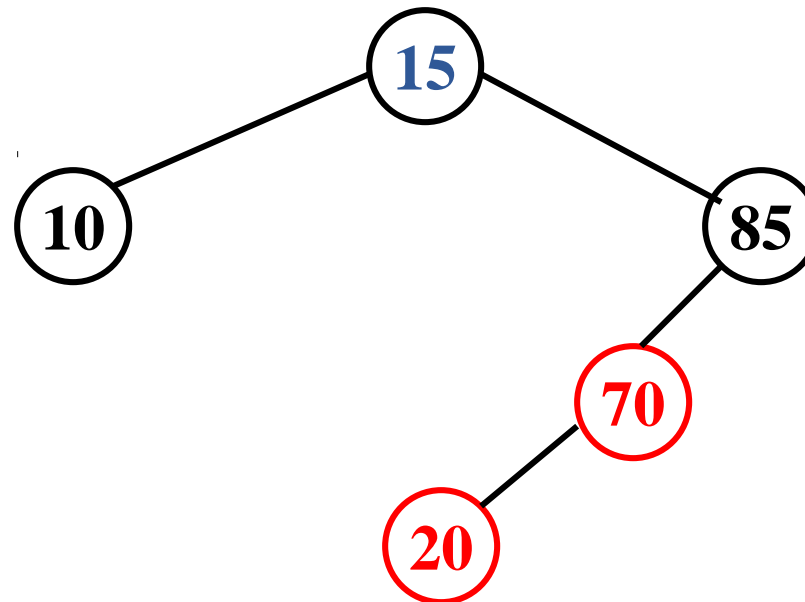
Insert do **červeno**-čierneho stromu – Ukážka

- Porušená podmienka!
 - Prefarbenie



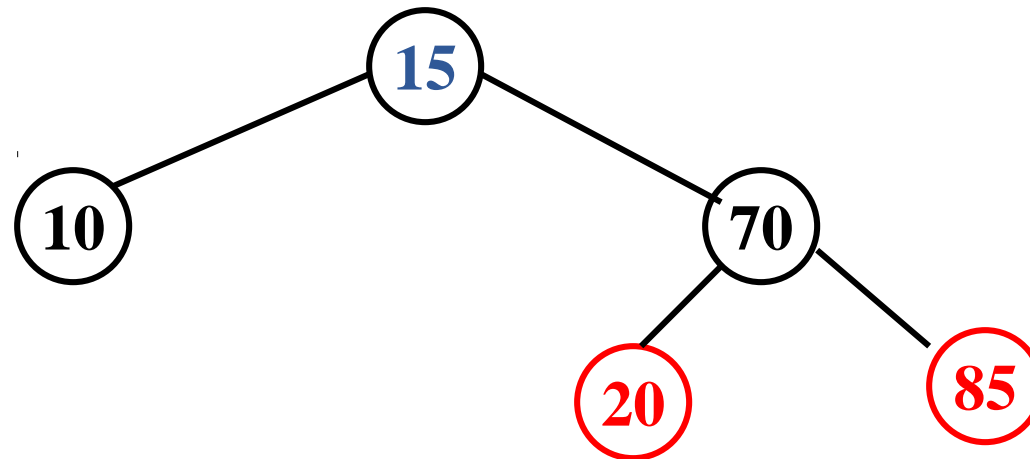
Insert do **červen**o-čierneho stromu – Ukážka

- Vlož 20



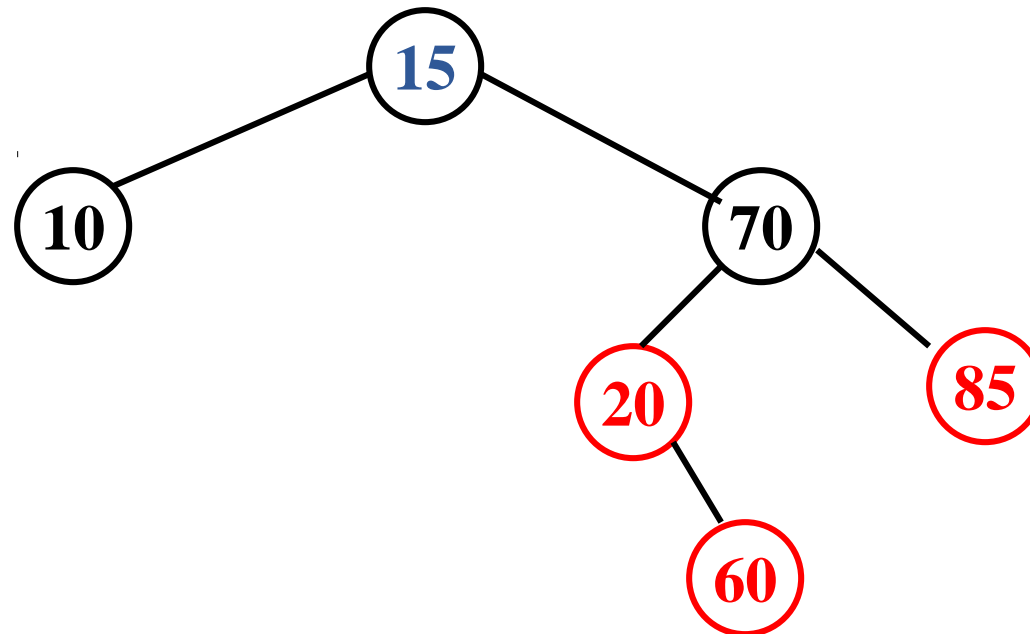
Insert do **červeno**-čierneho stromu – Ukážka

- Porušená podmienka!
 - Rotácia 70 doprava
 - Prefarbenie



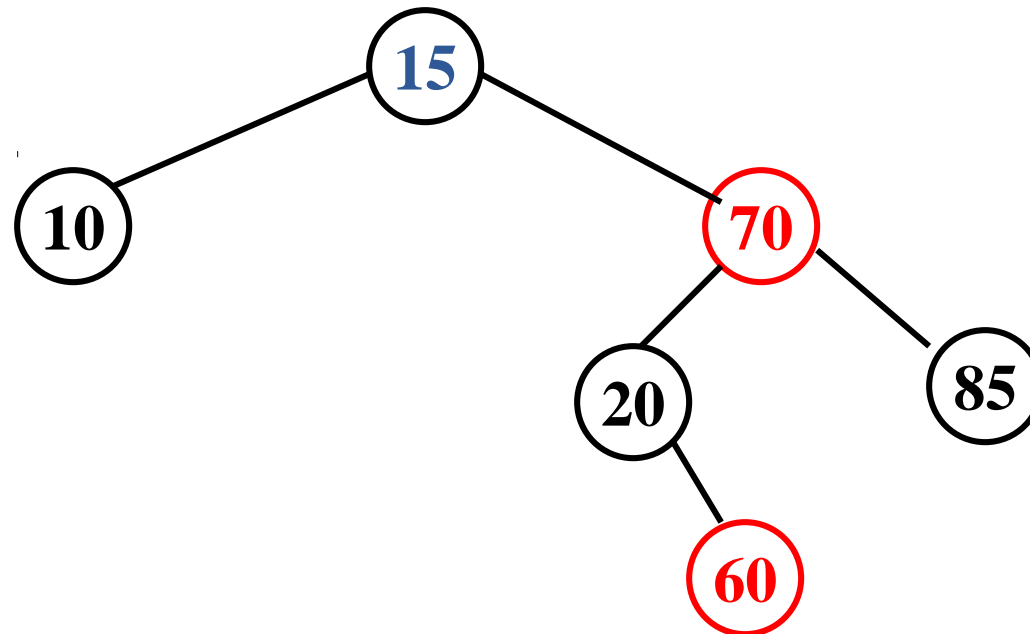
Insert do **červeno**-čierneho stromu – Ukážka

- Vlož 60



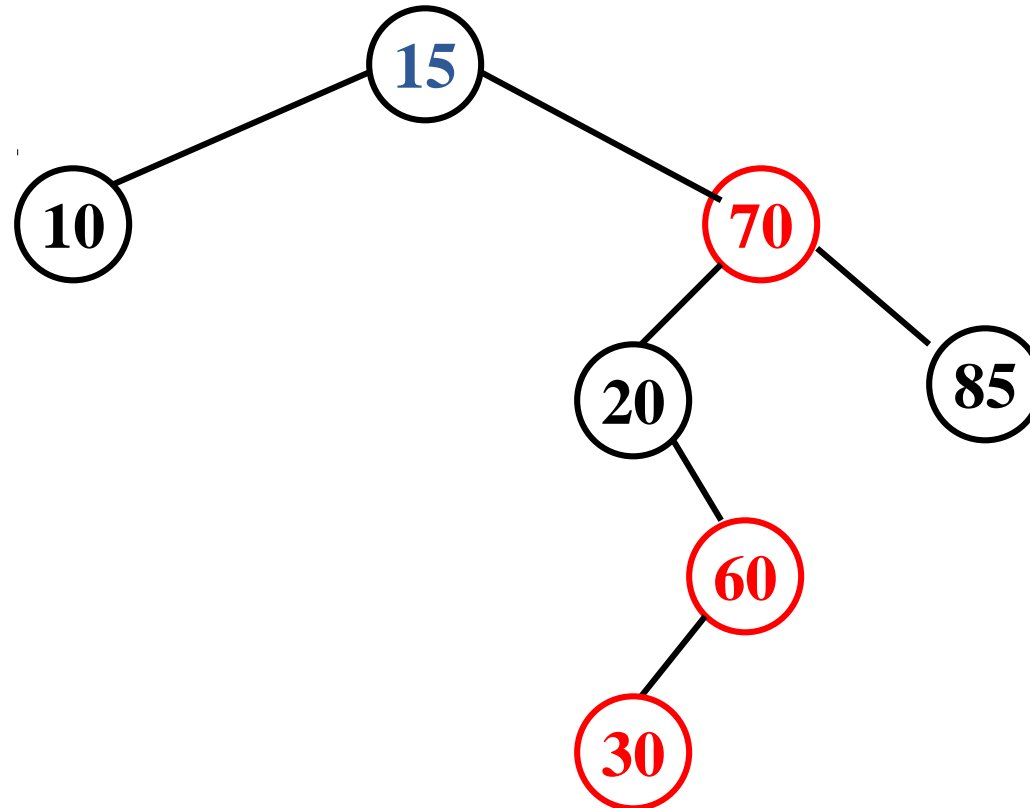
Insert do **červeno**-čierneho stromu – Ukážka

- Porušená podmienka!
 - Prefarbenie



Insert do **červeno**-čierneho stromu – Ukážka

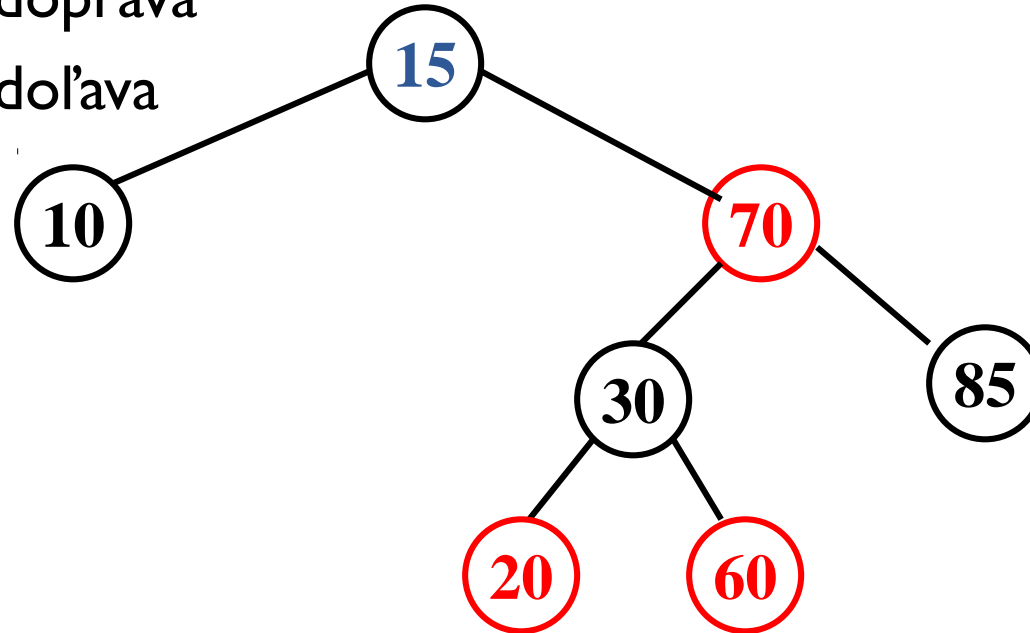
- Vlož 30



Insert do **červeno**-čierneho stromu – Ukážka

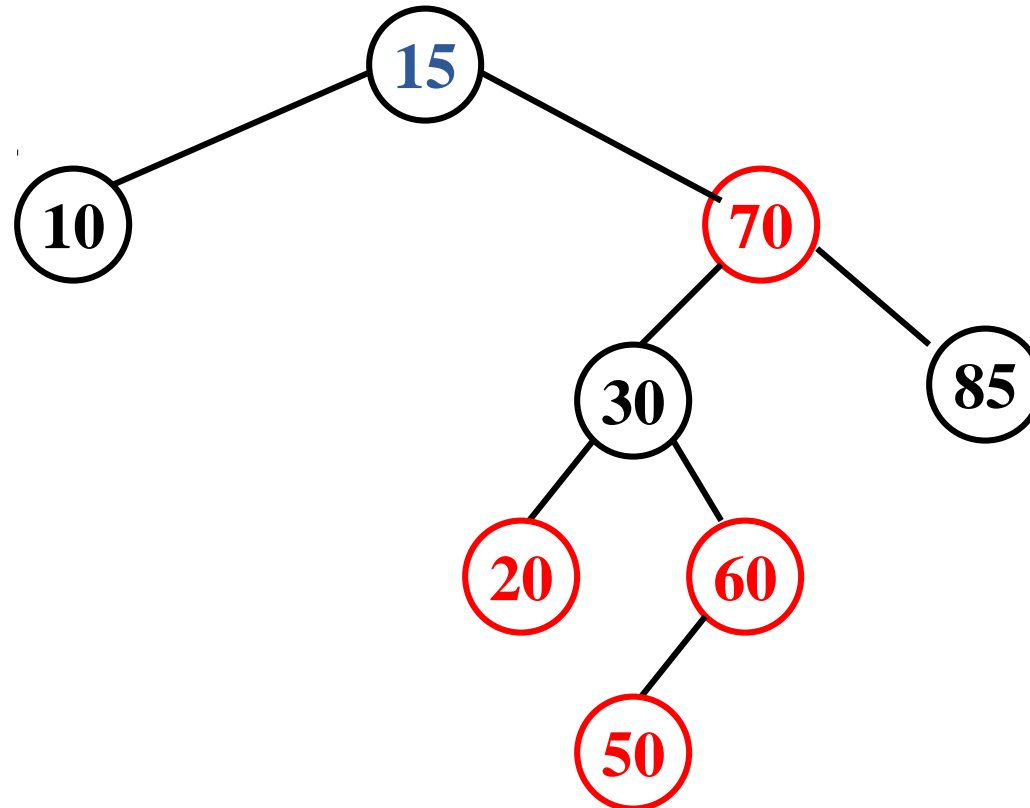
■ Porušená podmienka!

- Rotácia 60 doprava
- Rotácia 30 doľava
- Prefarbenie



Insert do **červeno**-čierneho stromu – Ukážka

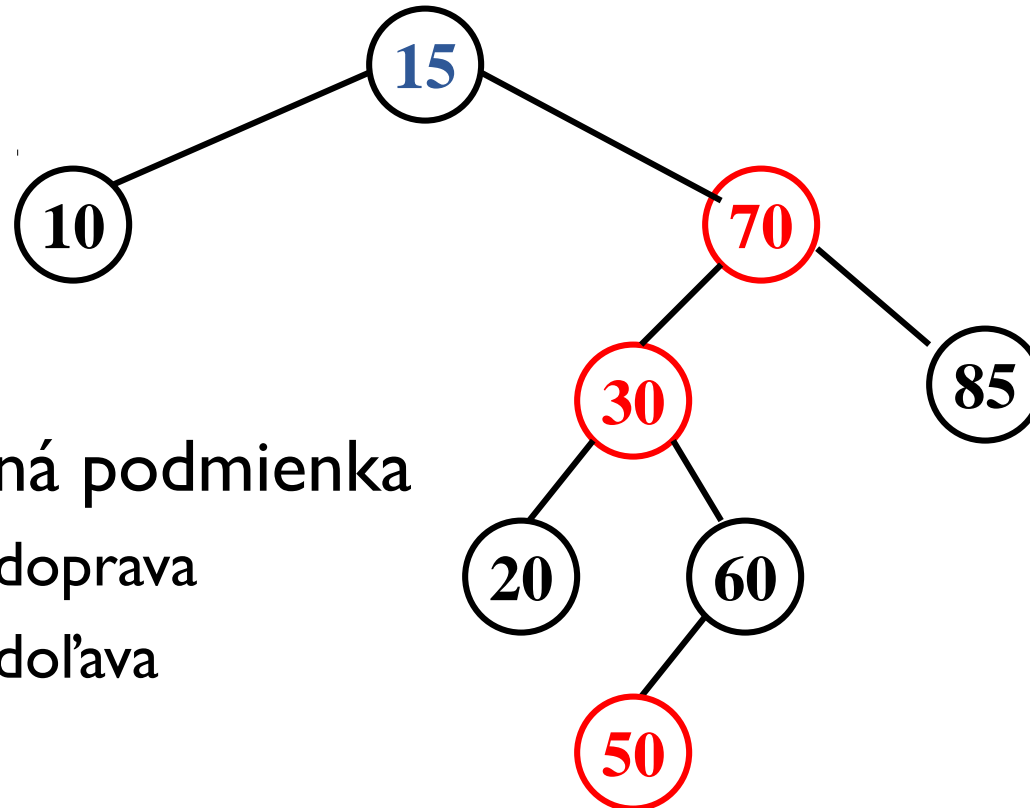
- Vlož 50



Insert do **červeno**-čierneho stromu – Ukážka

- Porušená podmienka!

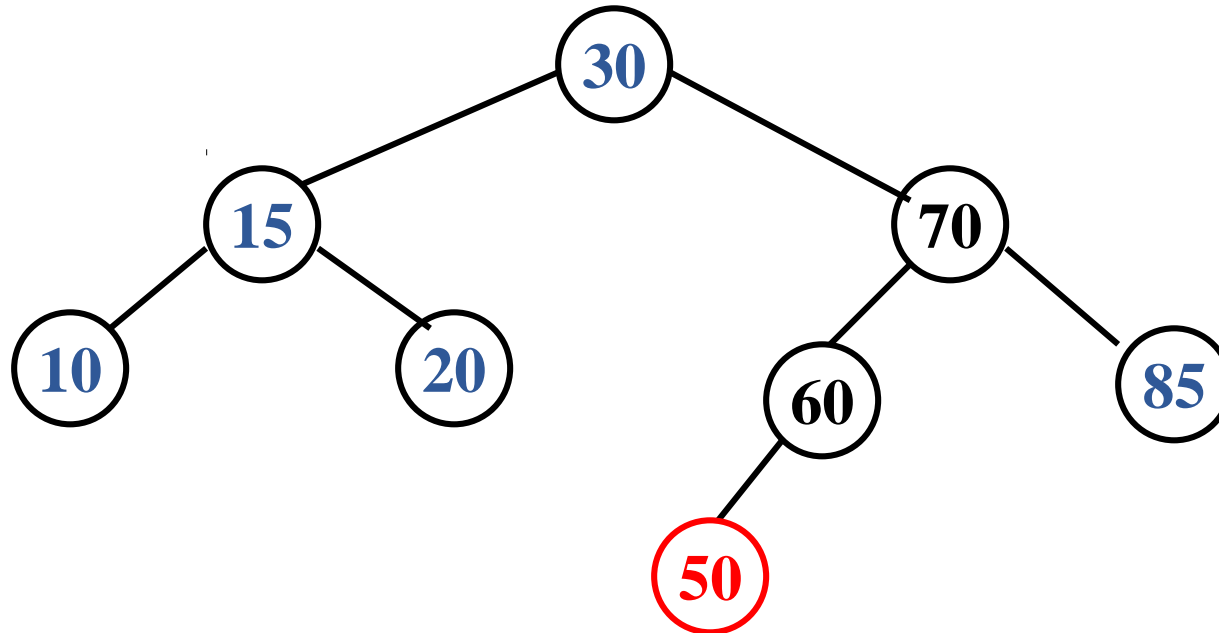
- Prefarbenie



- Stále porušená podmienka

- Rotácia 70 doprava
- Rotácia 15 doľava
- Prefarbenie

Insert do **červeno**-čierneho stromu – Ukážka

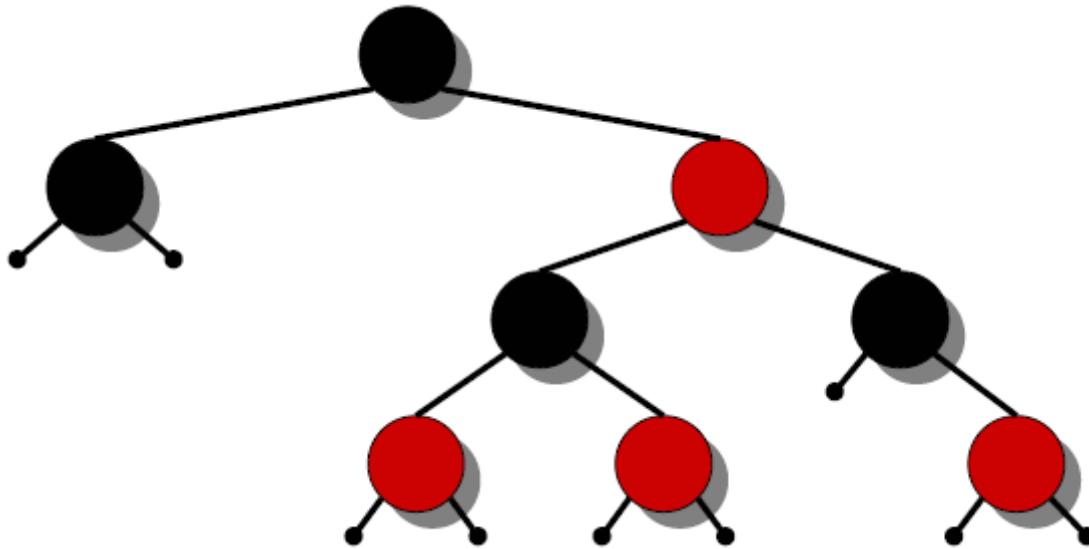


Delete z **červeno**-čierneho stromu

- Podobne ako v prípade štandardných BVS – vrchol na odstránenie nahradíme inorder predchodcom
- Musíme teda vyriešiť len prípad odstránenia vrcholu s jedným dieťaťom – nahradíme ho dieťaťom, a prípadnú „chybu“ riešime rekurzívne, postupujúc ku koreňu
- Odstránením vrcholu môžeme porušiť podmienku (3) „cesty do listov majú rovnaký počet **čiernych** vrcholov“
- Sú dva jednoduché prípady, ak zostal / sme v červenom vrchole alebo sme v koreni, prefarbíme na **čierno**
- Inak nastavíme vrchol ako **dvojito čierny**, a prefarbovaním smerom ku koreňu túto chybu vyriešime

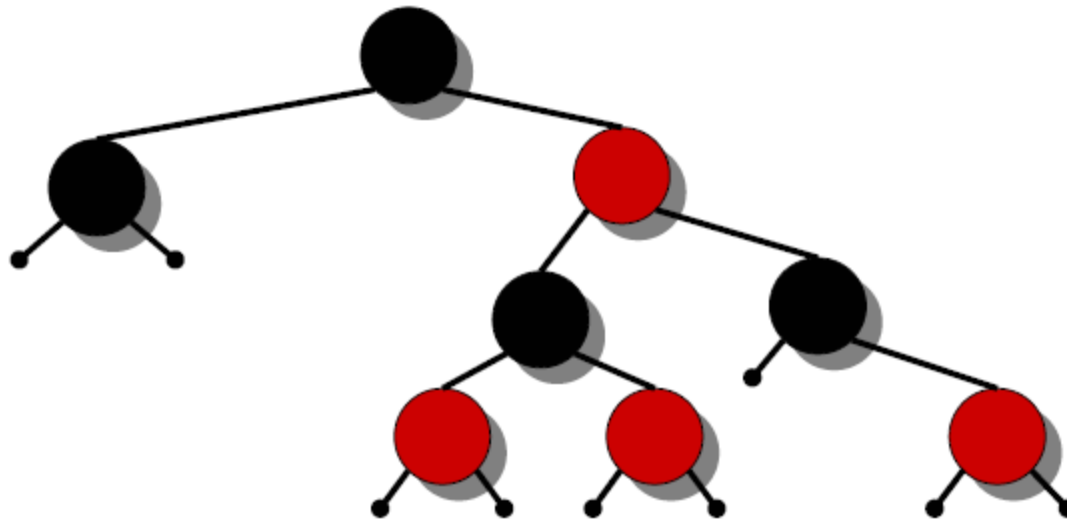
Červeno-čierny strom ako 2-3-4 strom

- Spojíme **červené** vrcholy do ich **čiernych** rodičov



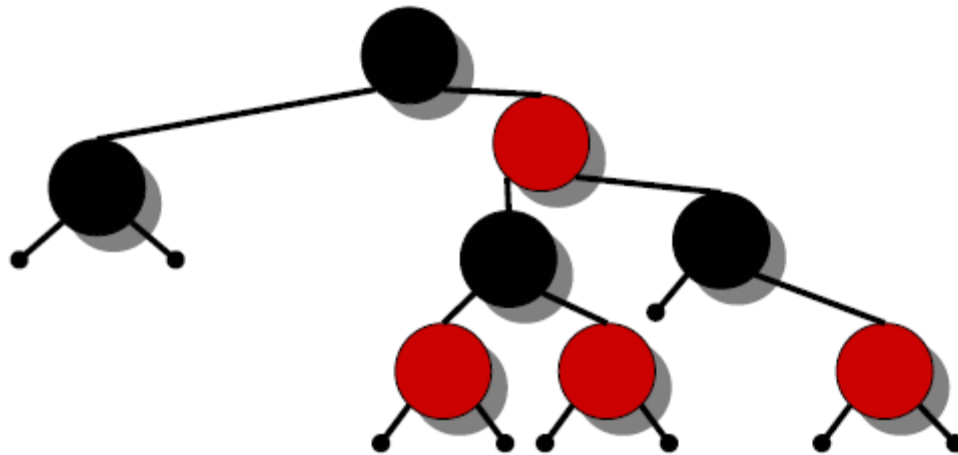
Červeno-čierny strom ako 2-3-4 strom

- Spojíme **červené** vrcholy do ich **čiernych** rodičov



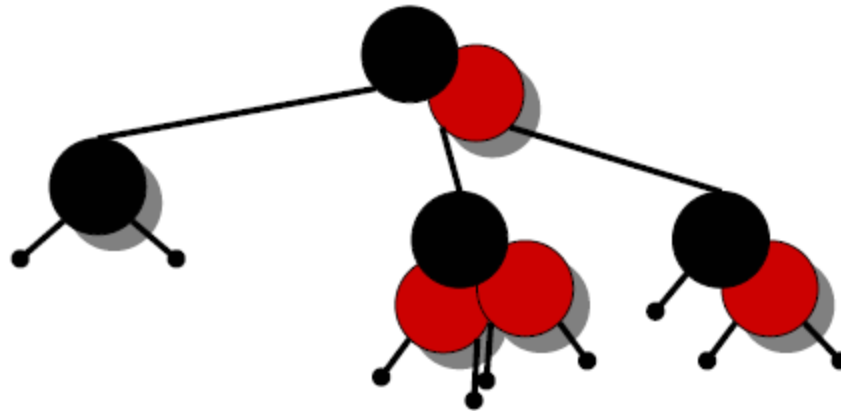
Červeno-čierny strom ako 2-3-4 strom

- Spojíme **červené** vrcholy do ich **čiernych** rodičov



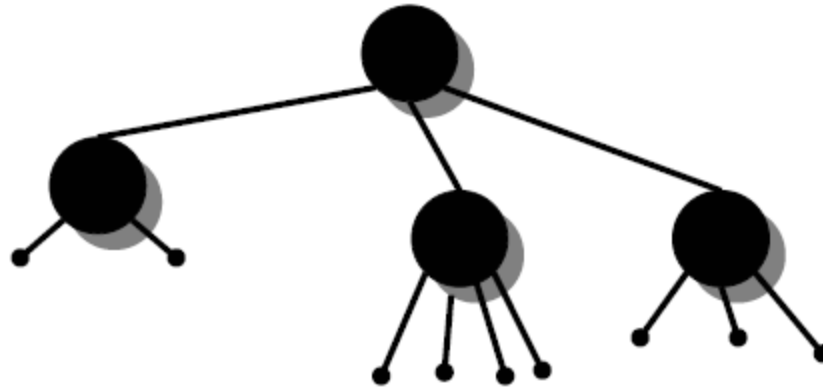
Červeno-čierny strom ako 2-3-4 strom

- Spojíme **červené** vrcholy do ich **čiernych** rodičov



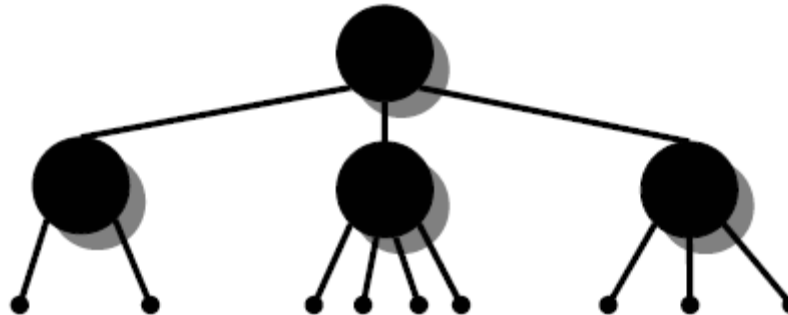
Červeno-čierny strom ako 2-3-4 strom

- Spojíme **červené** vrcholy do ich **čiernych** rodičov



Červeno-čierny strom ako 2-3-4 strom

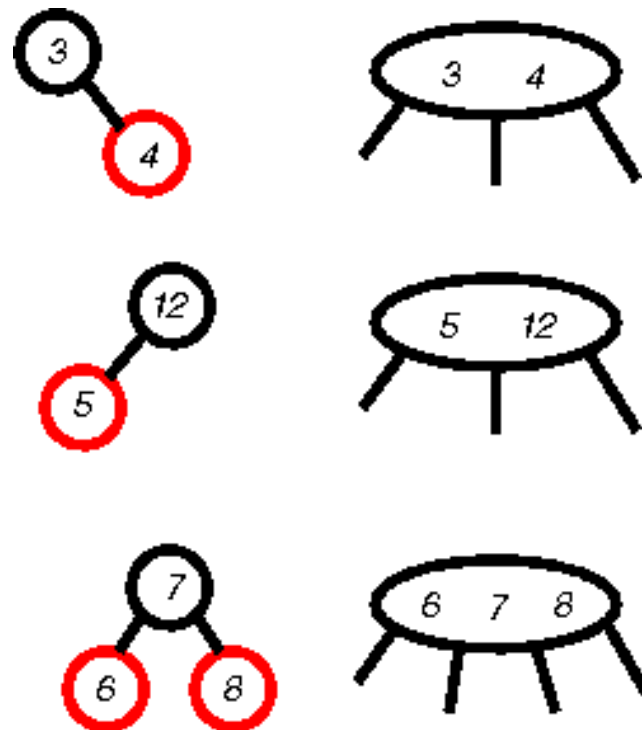
- Spojíme **červené** vrcholy do ich **čiernych** rodičov



- Vznikne strom, v ktorom vnútorné vrcholy majú 2, 3 alebo 4 deti – **2-3-4 strom**

Červeno-čierne stromy ako iné stromy

- Červeno-čierne stromy sú ako reprezentácia stromov s väčšími vrcholmi využitím binárneho stromu
- V závislosti od varianty implementácie zodpovedajú
 - 2-3-4 stromom, alebo
 - 2-3 stromom

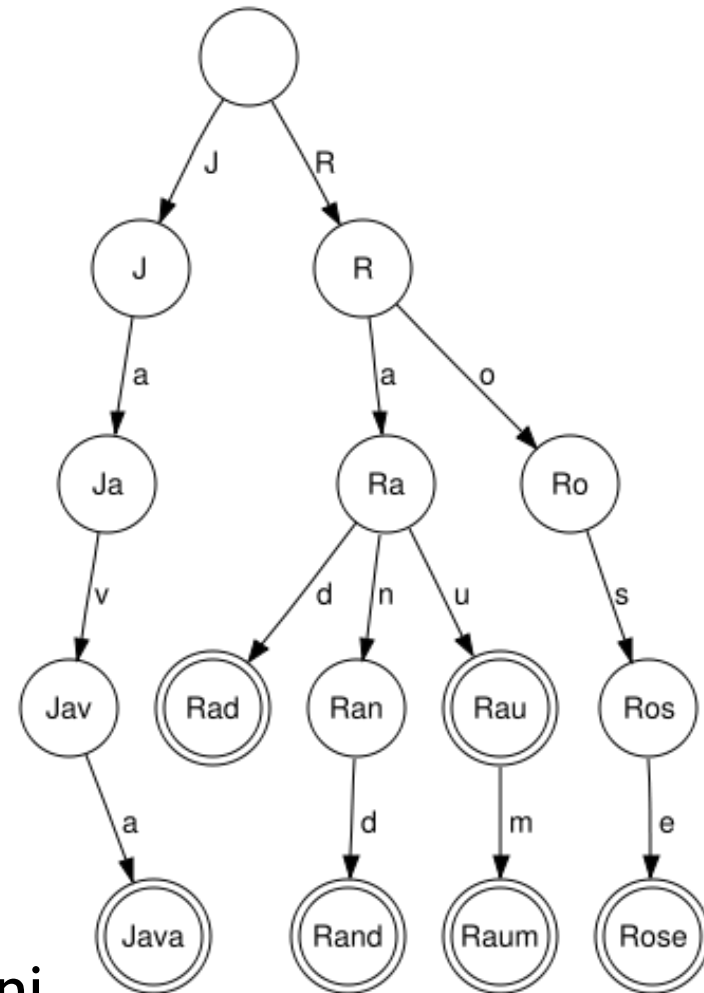


Porovnanie hĺbok vyvážených stromov

- h_n – hĺbka binárneho vyhľadávacieho stromu s n listami
- Dolné ohraničenie (úplný BVS): $\log n \leq h_n$
- Výškovo vyvážené (AVL) stromy: $h_n \leq 1.44 \log n$
- Červeno-čierne stromy: $h_n \leq 2 \log n$
- Váhovo vyvážené stromy: $2 \log n \leq h_n \leq O(\log n)$
- Iné teoretické modely dosahujú lepšie garancie, ale pre praktickú implementáciu sú komplikované
 - Malé zrýchlenie pri vyhľadávaní
 - Výrazné komplikácie pri úpravách (insert, delete)
 - Napr. 2-3 a 2-3-4 stromy s n listami majú výšku $\log n$

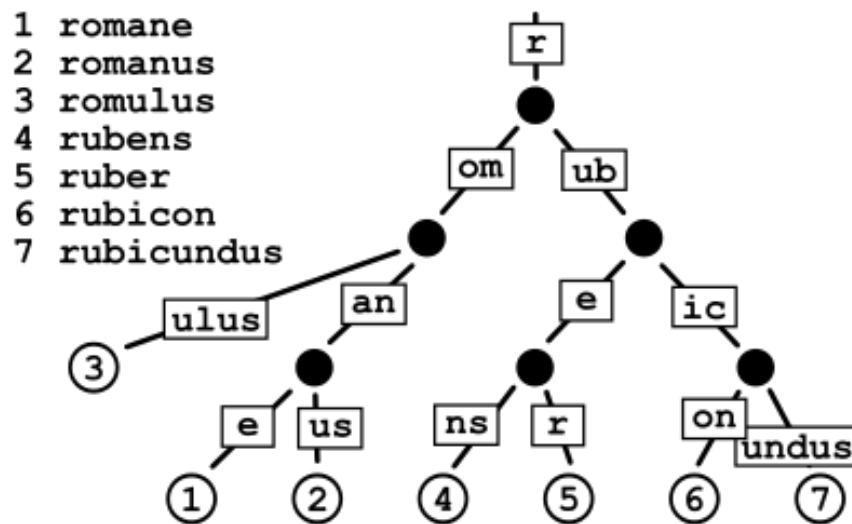
Trie – dynamická množina reťazcov

- Strom prefixov reťazcov
- **Vrcholy zodpovedajú prefixom hodnôt prvkov v množine**
 - Nasledovníci vrcholu majú spoločný prefix
 - Koreň je prázdny reťazec
 - Nie každý vrchol zodpovedá reťazcu z množiny
 - Každý z listov zodpovedá nejakej hodnote v množine
- **Každá hrana (prechod) má priradené písmenko**
- Vykonávanie operácií: začnem v koreni a postupujem po jednom písmenku



Radixový strom (radix tree)

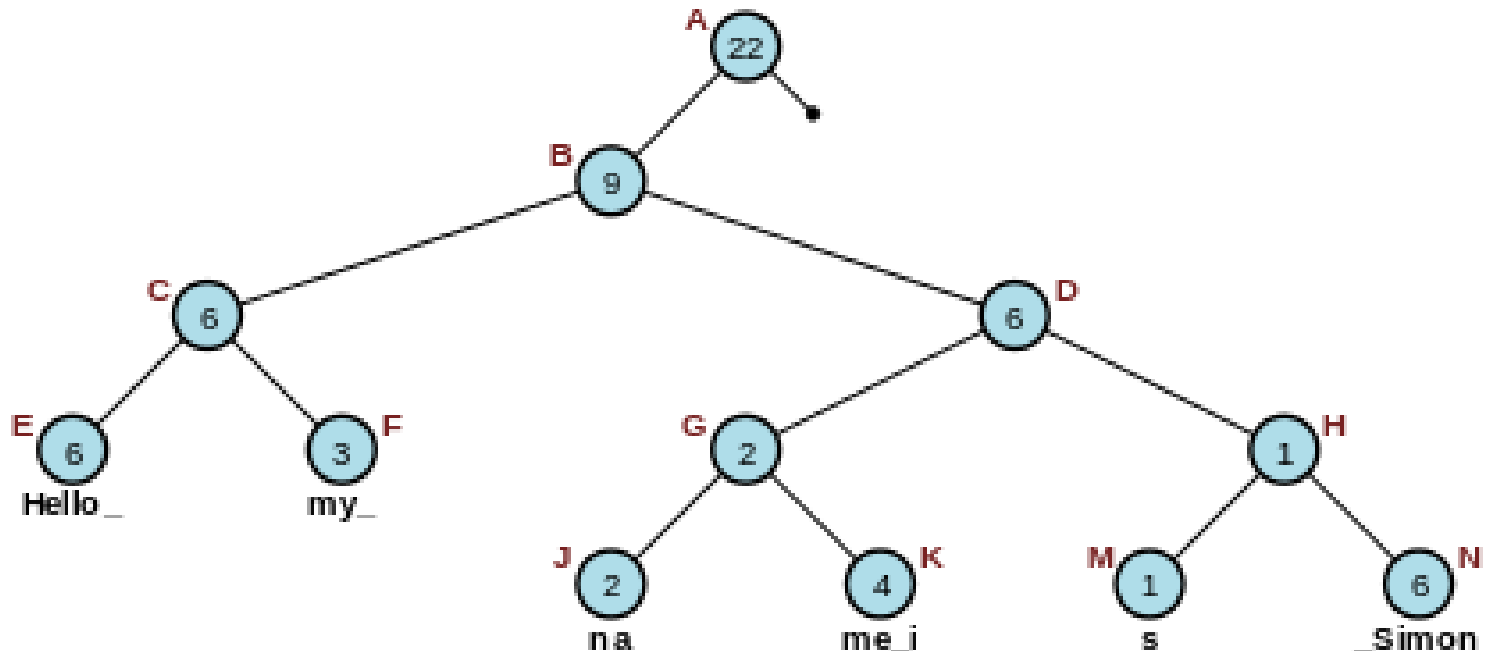
- Priestorovo optimalizovaný trie
- Také vrcholy, ktoré sú jedinými deťmi svojho rodiča sú spojené s rodičom do jedného vrcholu
- **Hrany môžu mať priradený reťazec**



- Efektívna dynamická množina reťazcov

Lano – efektívna manipulácia dlhých reťazcov

- Lano (rope) – binárny vyhľadávací strom
vo vrchole x je váha = počet znakov reťazca vo vrchole x a počet znakov v ľavom podstrome x
- Napr. Hello_my_name_is_Simon

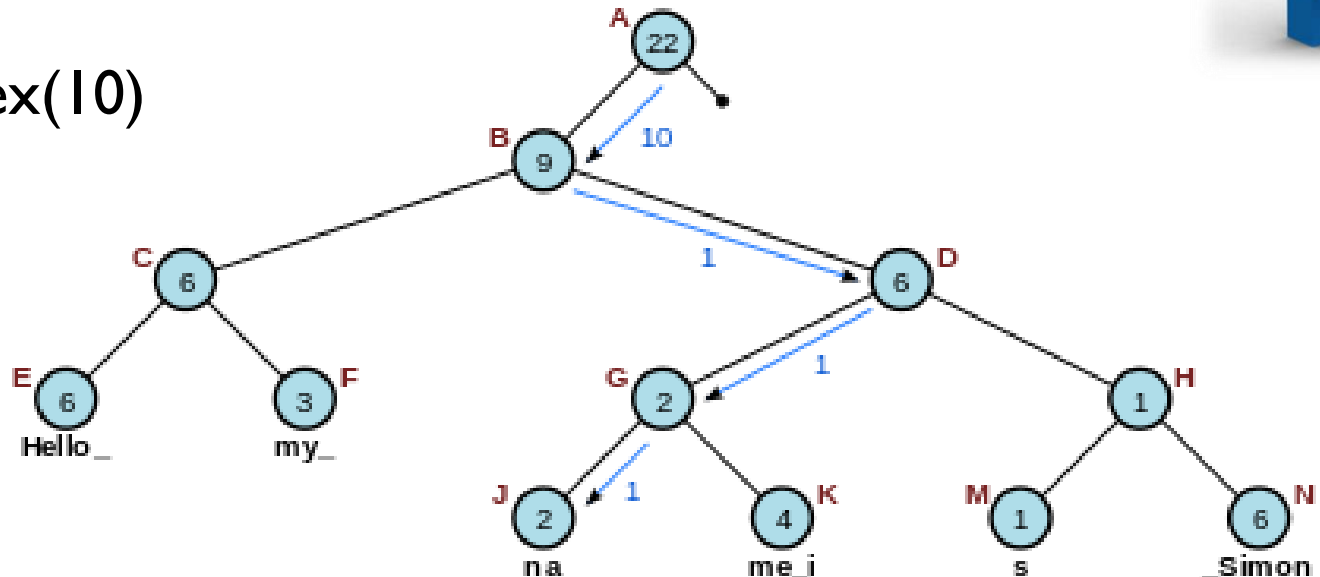


Lano – efektívna manipulácia dlhých reťazcov

- Efektívne manipulácie s dlhými reťazcami – $O(\log n)$
 - **Index** – znak na k-tej pozícii
 - **Concat** – spojenie dvoch reťazcov do jedného
 - **Insert** – vložiť podreťazec
 - **Delete** – odstrániť podreťazec



Napr. index(10)

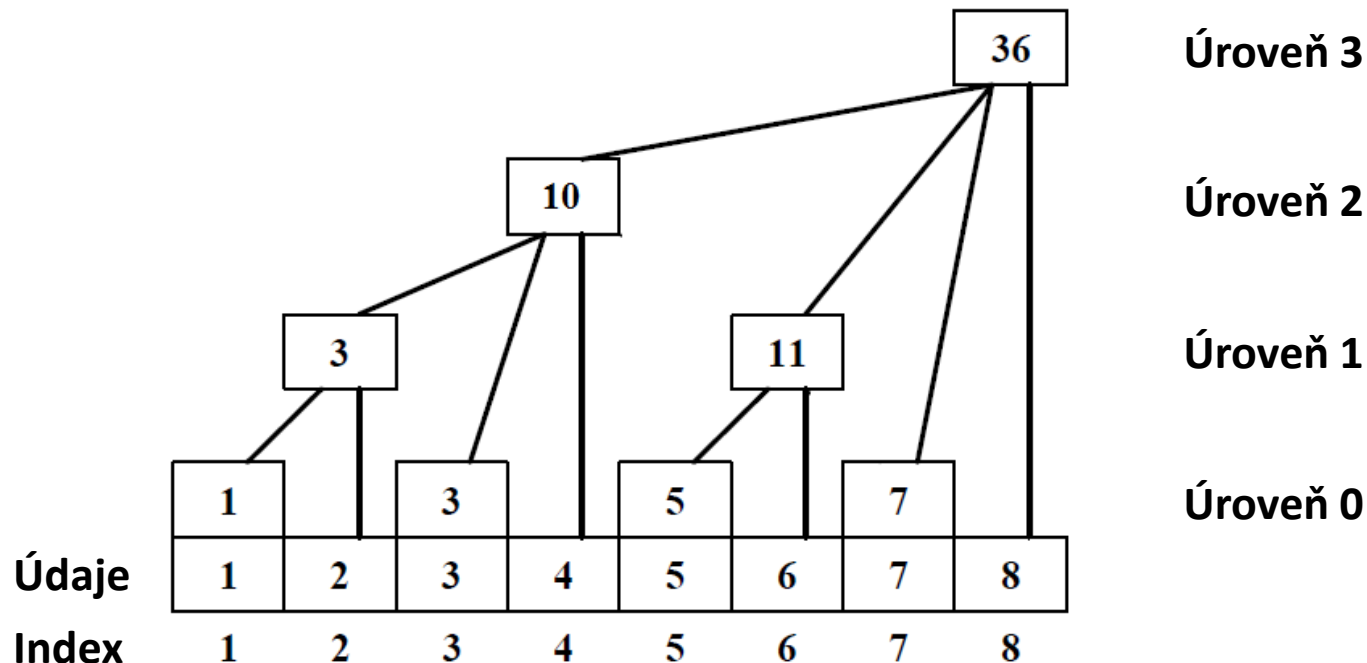


Binárne indexované stromy (Fenwick tree)

- Uvažujme úlohu určiť súčet prvkov intervalu v postupnosti a_1, a_2, \dots, a_n
- Chceme podporovať dve operácie:
 - **update(i,value)** – úprava i-tej hodnoty
 - **sum(i,j)** – súčet prvkov a_i, \dots, a_j
- Jednoduchá reprezentácia v poli: update $O(1)$, sum $O(n)$
- Binárne indexované stromy (tiež len v rovnakom poli)
 - Umožňujú efektívne vykonávať update, a zistiť **prefixovú sumu**, teda súčet a_1, \dots, a_i (pre ľubovoľné i)
 - Operácie: update $O(\log n)$, sum $O(\log n)$

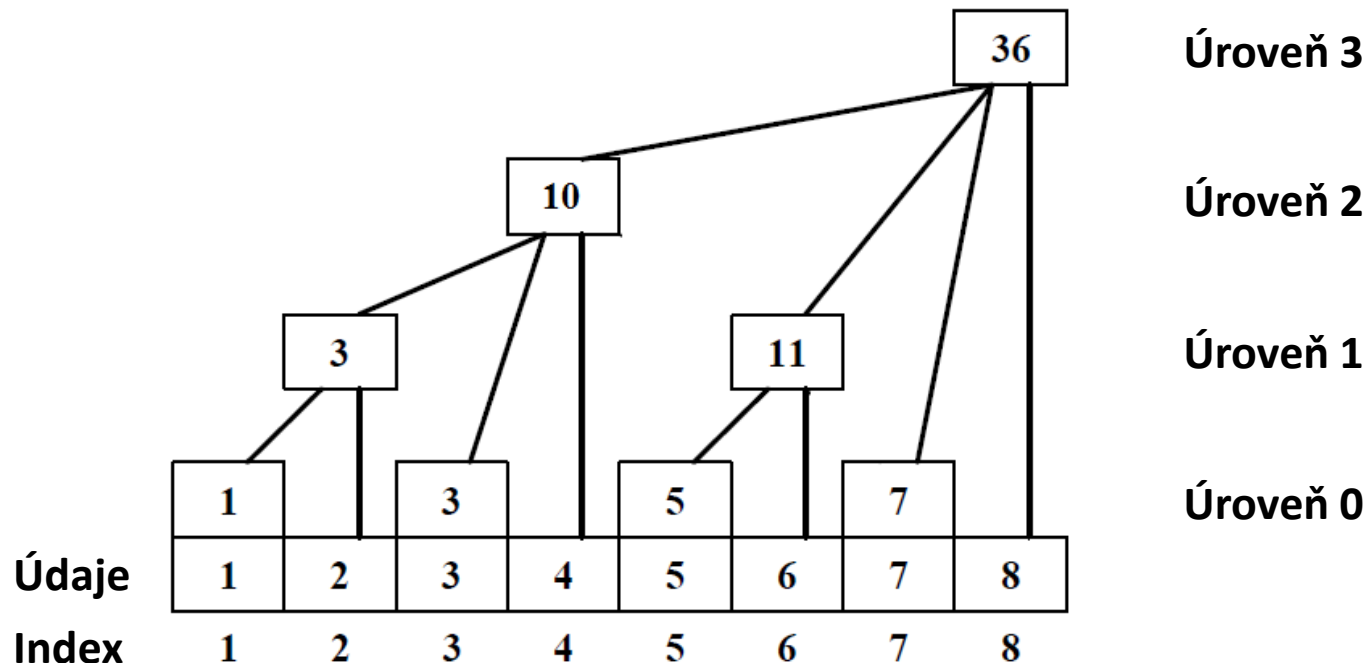
Binárne indexované stromy (2)

- Implementované v poli (podobne ako binárna halda)
 - Index v poli reprezentuje vrchol stromu
 - Rodiča vrcholu získame bitovými operáciami s indexom
 - V poli si pamätáme prefixový súčet prislúchajúcej časti poľa:



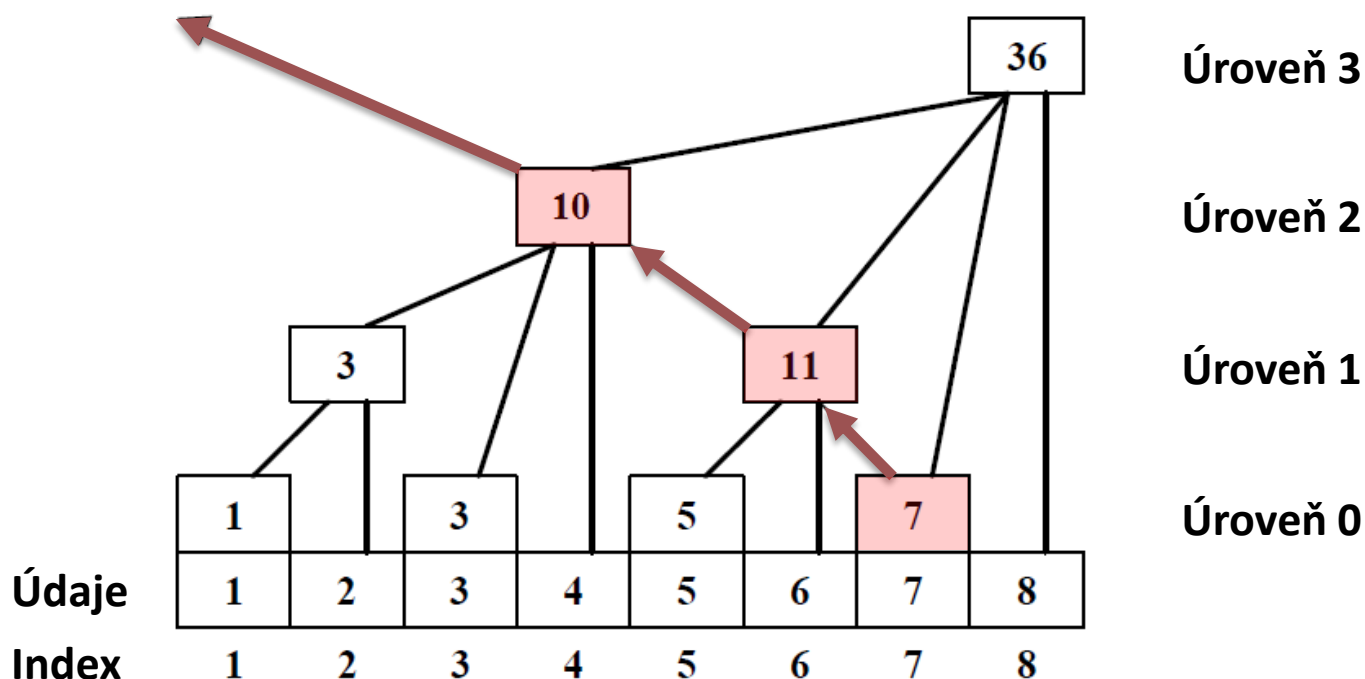
Binárne indexované stromy (3)

- Vo vrchole V si pamätáme súčet prvkov z intervalu $[V-2^K+1, V]$, kde K je počet núl na konci binárneho zápisu indexu V – napr. pre $V=4=100_2$ sú to čísla $[4-2^2+1, 4]$, teda $[1...4]$: $T[4]=a[1]+a[2]+a[3]+a[4]=10$



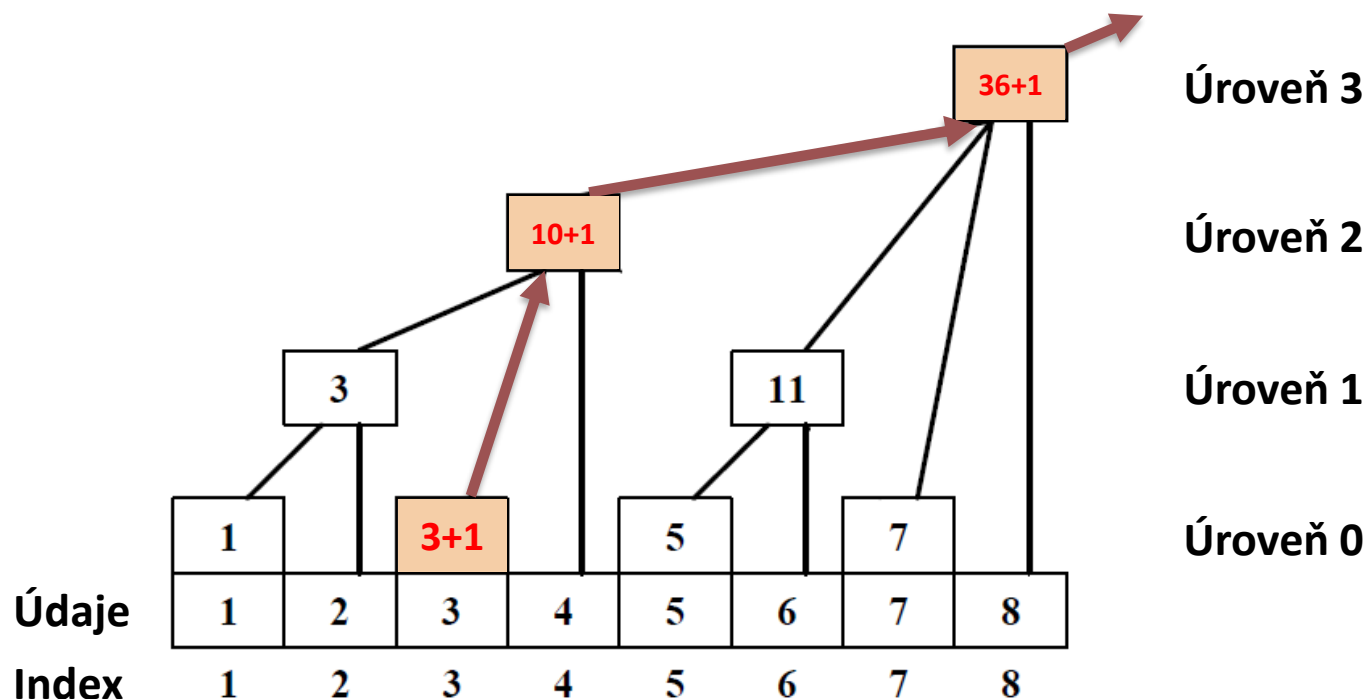
Operácia prefixSum

- Súčet hodnôt počas **prechodu dol'ava hore** (posun: v indexe zmažeme posledný nenulový bit)
- Napr. prefixSum(7): $7 \rightarrow 6 \rightarrow 4 \rightarrow \text{koniec}$
 $T[7] + T[6] + T[4] = 7 + 11 + 10 = 28$



Operácia update

- **Prechodom doprava hore** upravíme hodnoty v strome, ktoré zahŕňajú hodnotu upravovaného prvku (posun: k indexu pripočítame posledný nenulový bit)
- Napr: $\text{update}(3,+1)$: $3 \rightarrow 4 \rightarrow 8 \rightarrow \text{koniec}$



Ďalšie typy vyhľadávacích stromov

- Obsahujú intervaly
 - **Segmentové stromy** – vhodné pre operáciu: nájsť intervaly, v ktorých sa nachádza daný bod
 - **Intervalové stromy** – vhodné pre intervalové operácie: napr. nájsť intervaly, ktoré prekrývajú daný interval
- Obsahuje body:
 - **Range trees** – vhodné na operáciu: nájsť body, ktoré sa spadajú do daného intervalu
- Obsahujú skalárne hodnoty:
 - **Binárne indexované stromy** – vhodné na operáciu: koľko bodov je v danom intervale