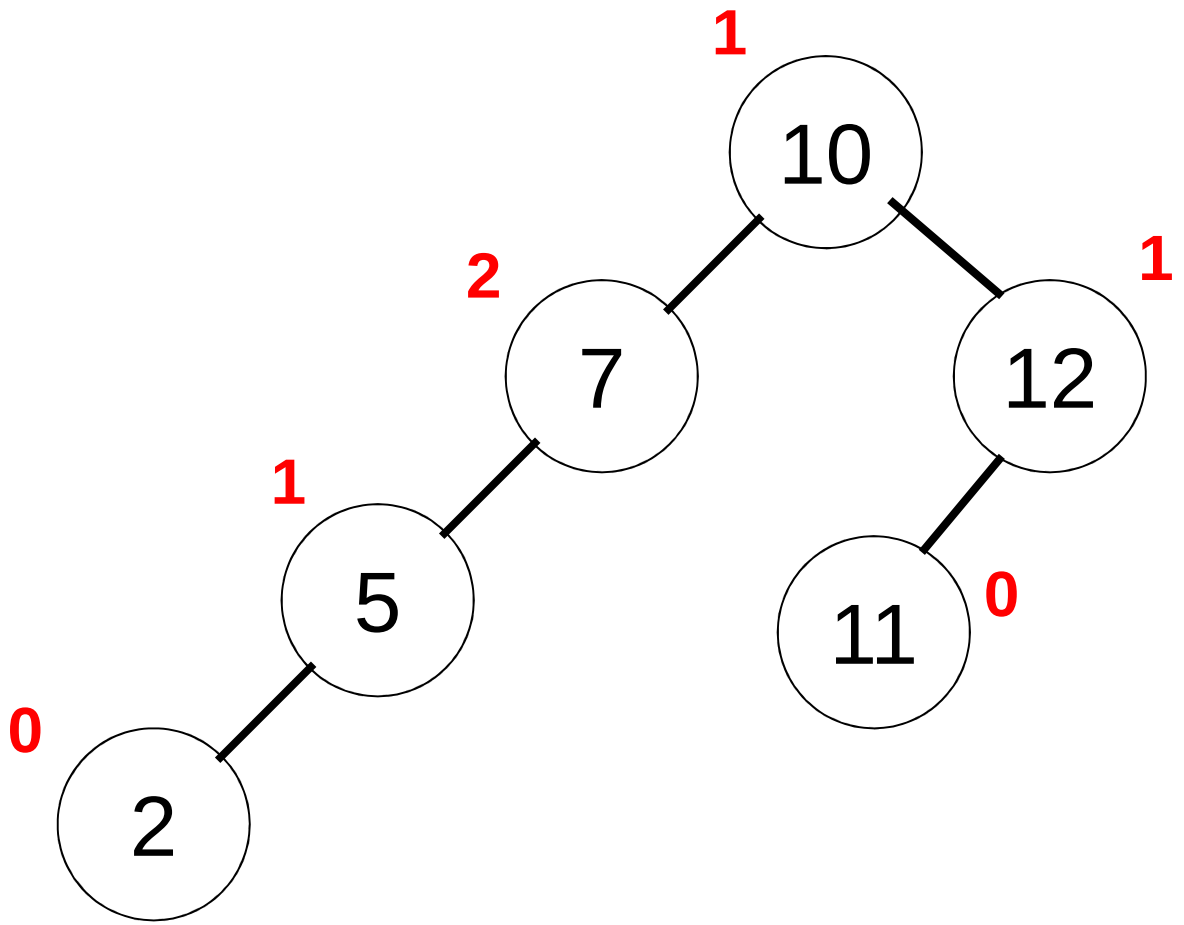
# Stromy 2

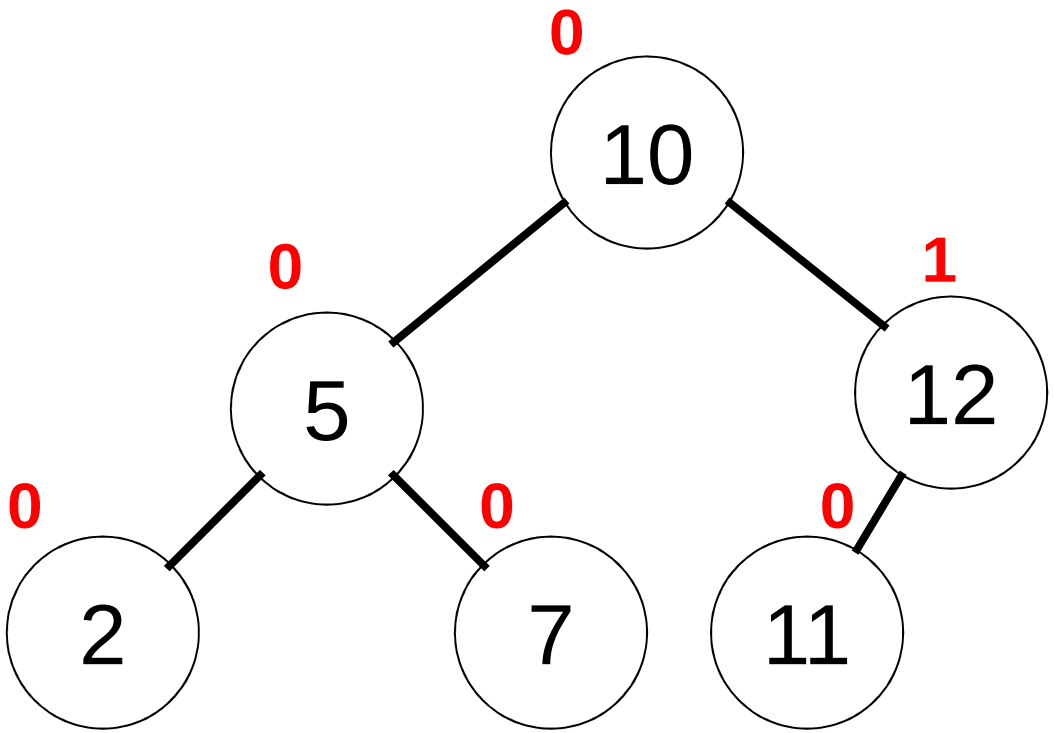
# AVL stromy

- AVL
  - jména tvůrců stromů: dva Rusové Adelson-Velskii, Landis
- vyvážené binární stromy
  - pro každý uzel *u* stromu platí, že rozdíl mezi výškou jeho levého a pravého podstromu je nejvýše 1
  - stromy jsou *samovyvažující*
    - po přidání a odebrání stromu může dojít k rozvážení – v operacích vlož, vymaž se spouští kontrola vyvážení a event. se vyvažuje

# AVL stromy

- implementace je obdobná jako u binárního (vyváženého) stromu
  - datová struktura pro uzel stromu obsahuje navíc proměnnou (tzv. ***faktor vyvážení - ballance***) reprezentující stupeň vyváženosti uzlu $b = h_L - h_P$ ($h_L$, $h_P$ – výšky levého a pravého podstromu):
    - 0 – oba podstromy jsou stejně vysoké
    - 1 – levý podstrom je o 1 vyšší (hlubší)
    - 2 – levý podstrom je o 2 vyšší (hlubší)
    - -1 – pravý podstrom je o 1 vyšší (hlubší)
    - -2 – pravý podstrom je o 2 vyšší (hlubší)


1
10
2
7
1
12
1
5
11
0
0
2


0
10
0
5
1
12
0
2
0
7
0
11

# Vložení prvku *x* do AVL stromu

1. Vyhledej prvek *x* ve stromu
2. Jestliže prvek existuje, konec
3. Vlož x pod list, kde se ukončilo vyhledávání
4. Přepočti stupně vyvážení od přidaného uzlu vzhůru (od listu ke kořenu).
5. Jestliže došlo k rozvážení (v místě, kde se poprvé vyskytne stupeň vyváženosti s hodnotou +/-2), jdi na bod 6, jinak konec
6. Vyvaž strom aplikací jedné ze dvou rotací:
   - jednoduché rotace RR, LL
   - dvojité rotace RL, LR

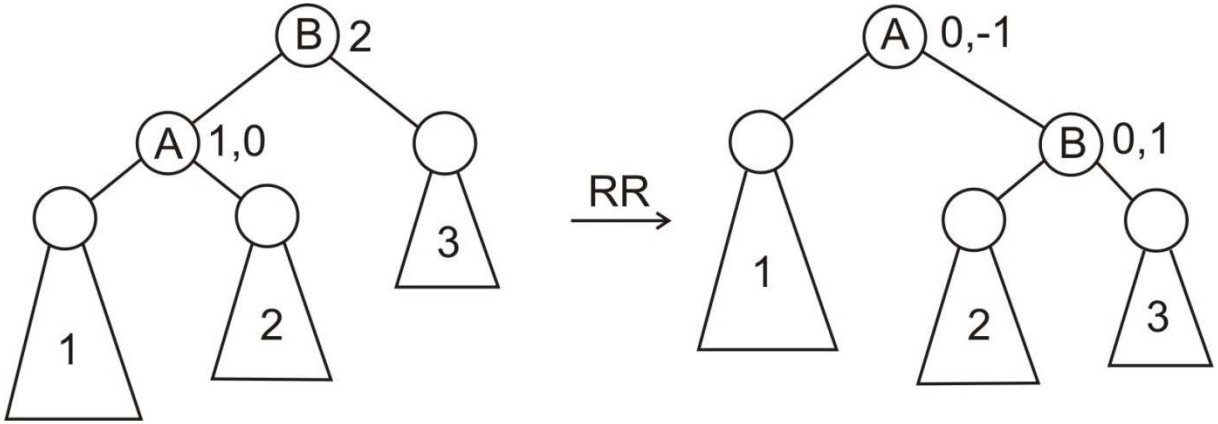
## *Jednoduché rotace*

- používají se, pokud jsou znaménka faktoru nevyváženosti stejná
  - vyvažujeme přímou větev

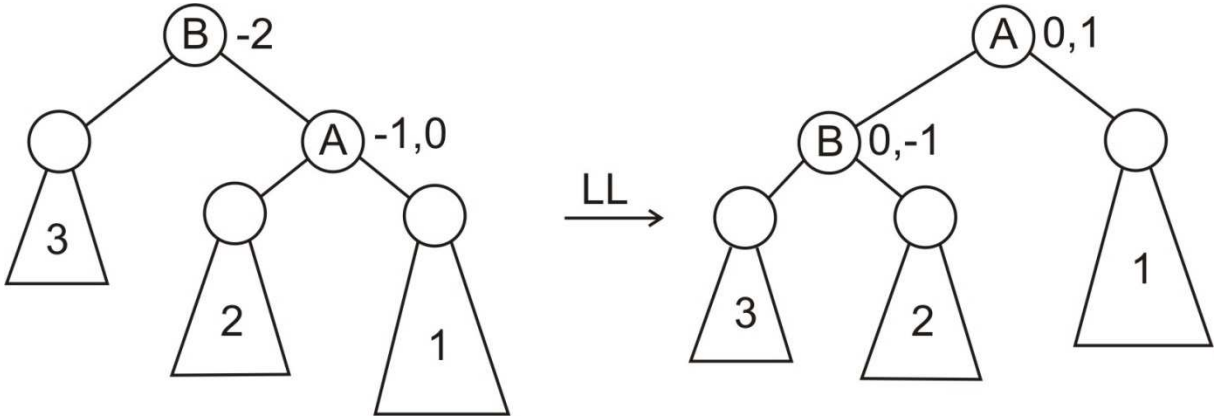
## *Dvojité rotace*

- používají se, pokud jsou znaménka faktoru nevyváženosti různá
  - vyvažujeme zalomenou větev

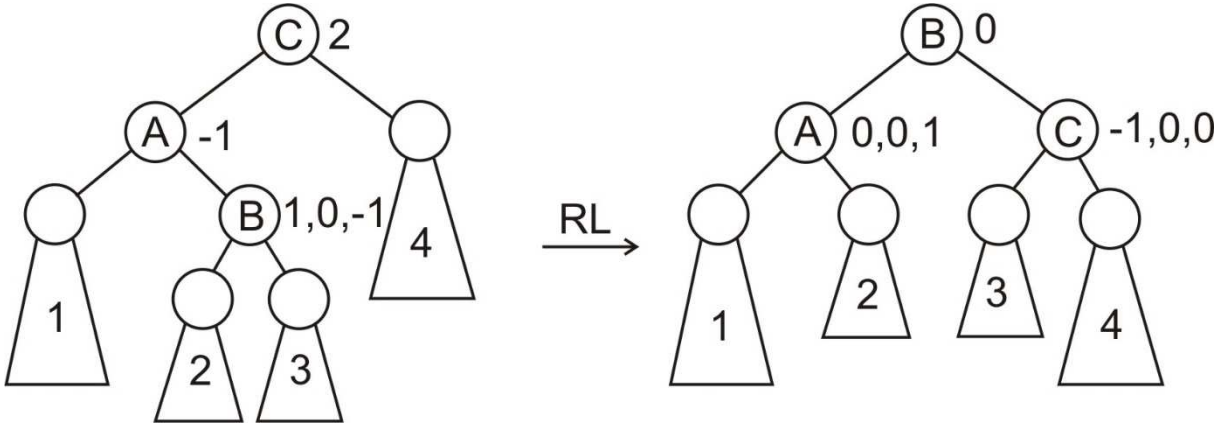
# Jednoduchá rotace RR



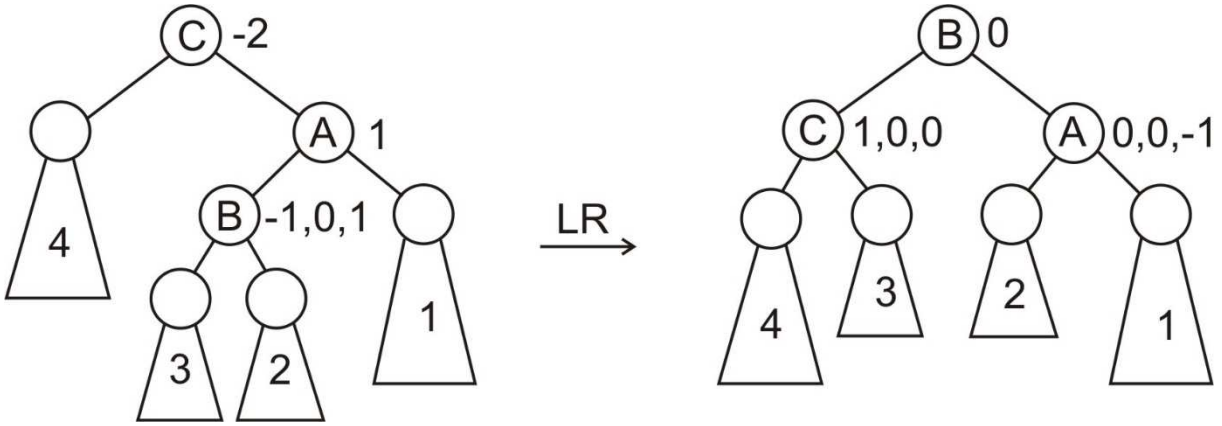

# Jednoduchá rotace LL



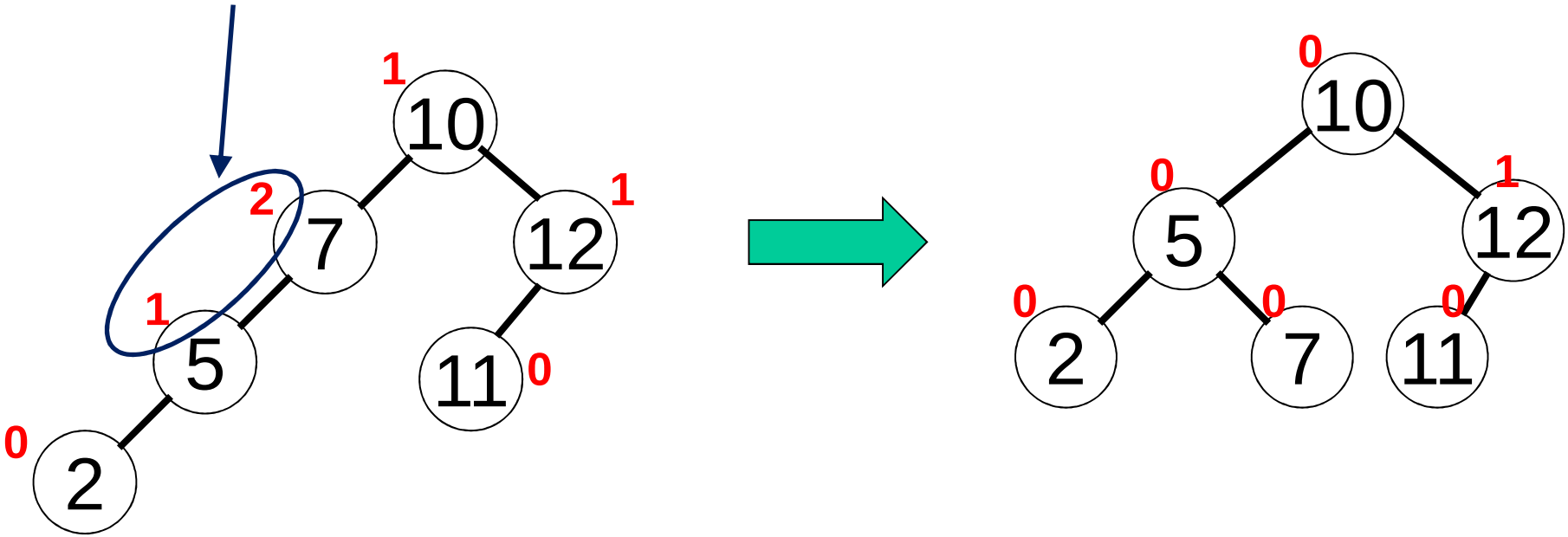
# Dvojitá rotace RL

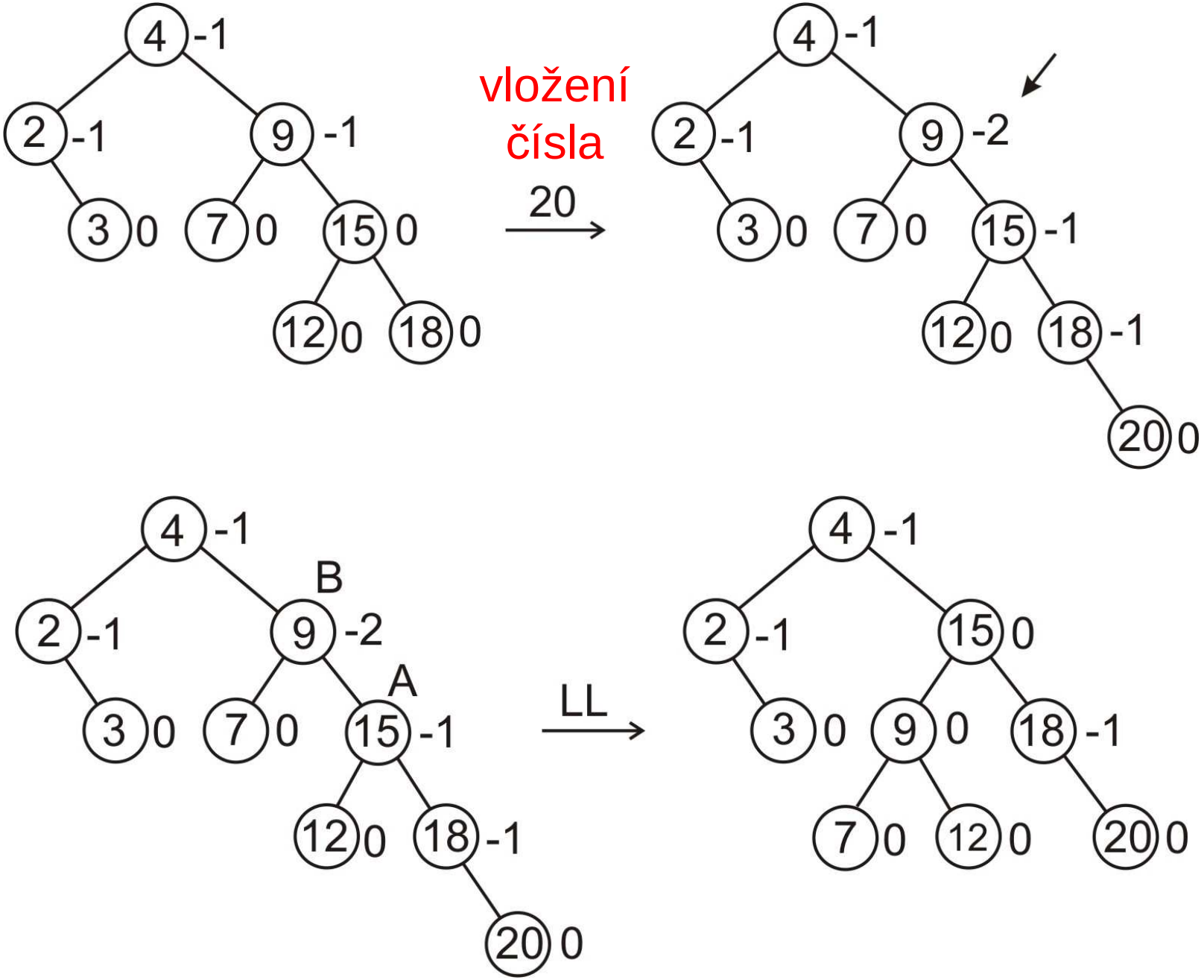



# Dvojitá rotace LR



stejná znaménka = jednoduchá rotace




4 -1
2 -1
9 -1
3 0
7 0
15 0
12 0
18 0
vložení
čísla
20
4 -1
2 -1
9 -2
3 0
7 0
15 -1
12 0
18 -1
20 0
4 -1
B
2 -1
9 -2
A
3 0
7 0
15 -1
12 0
18 -1
20 0
LL
4 -1
2 -1
15 0
3 0
9 0
18 -1
7 0
12 0
20 0

# Operační složitost operací

- výška AVL stromu je max. 1.44 $\times$ h, kde *h* je výška úplně vyváženého binárního stromu se stejným počtem prvků *n*
  - pro výšku úplně vyváženého stromu platí:

$$h = \log_2(n)$$

- všechny operace nad AVL stromem – *vyhledání, přidání i odebrání* prvku mají **logaritmickou složitost**

# Red - Black stromy

- červeno-černý strom
  - uzly jsou obarveny červeně nebo černě
- autor algoritmu
  - Rudolf Bayer,

    nejprve jej nazval symetrický binární B-strom
- značení RB získal až v práci Lea J. Guibase a Roberta Sedgewicka (1978)

# Red - Black stromy

*Vlastnosti:*

Každý uzel je červený nebo černý

1. Každý list (nil, NULL) je černý
2. Jestliže je uzel červený, oba jeho potomci jsou černí
3. Na kterékoliv cestě z kořene do listu leží stejný  počet černých uzlů
4. (Kořen je černý)

- počet *černých uzlů* na cestě z uzlu *x* do listu (mimo uzel *x*) nazýváme **černou výškou** uzlu *x*, píšeme *bh*(*x*)
- černá výška stromu = černá výška kořene

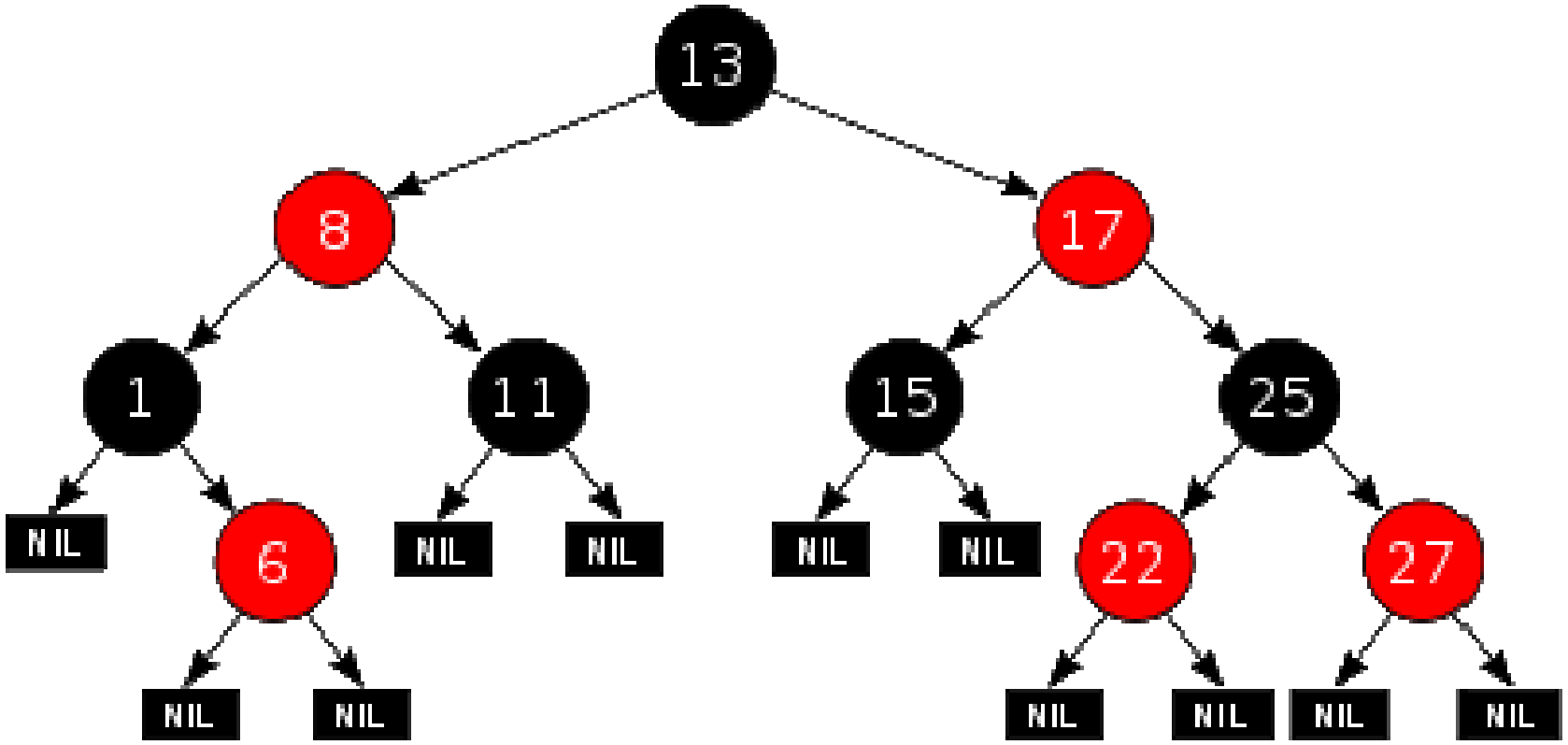
- výška RB stromu s n vnitřními uzly je nejvýše 2 log(n + 1)
  - strom je přibližně vyvážený

# Red - Black stromy

*Datová struktura uzlu:*

- data
- barva
- ukazatel na levý podstrom
- ukazatel na pravý podstrom
- ukazatel na rodič

# Red - Black stromy



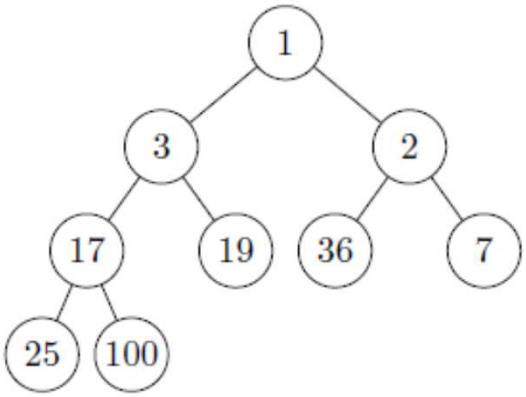
# Vkládání prvku

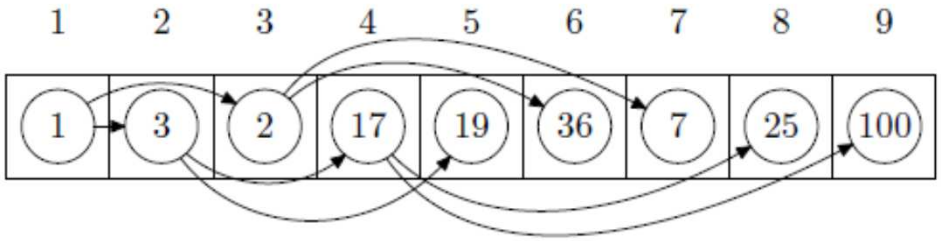
- nový uzel se vkládá jako červený stejně jako do binárního uspořádaného stromu
- pokud dojde k porušení vlastností (lze porušit pouze vlastnost 2), provádí se operace rotace (jednoduchá vlevo/vpravo) a přebarvení
- operace mají logaritmickou složitost

- AVL strom se používá tam, kde převažuje operace hledání nad vkládáním/mazáním, RB v případech, kde se častěji vkládá/maže

# Ukládání binárního stromu do binární haldy (pomocí pole)

- indexy pole musí začínat od 1
- uzel *u* je uložen v prvku s indexem
- *levý potomek v prvku s indexem 2i*
- *pravý potomek v prvku s indexem 2i+1*



# Rozptylovací funkce (hash)

- hledání s rozptylovací funkcí se využívá, když množina univerza je značně větší než očekáváná množina, ve které se hledá
- funkce, která ke každému klíči (vyhledávané hodnotě) vypočítá řádek v rozptylovací tabulce
  - rozdělí data do *k* ekvivaletních tříd
  - v každém řádku jsou uložena data patřící k dané třídě (spojový seznam, strom, …)

# Příklad

*Máme navrhnout způsob hledání v množině řetězů (bez diakritiky) pomocí rozptylovací funkce.*

- *anglická abeceda má 26 písmen*
  - *rozptylovací tabulka bude mít 26 řádek*
  - *rozptylovací funkce přiřadí hledanému řetězci index řádku dle prvního písmene v řetězci (A → 0, B → 1, …, Z → 25)*

# Příklad

- tedy
  - *v nultém řádku tabulky budou uložena všechna slova začínající na A, v prvním řádku tabulky budou uložena všechna slova začínající na B atd.*
    - *např. spojovým seznamem*
- příklad: hledám slovo AJAX
  - slovo začíná na písmeno A: rozptylovací funkce vypočítá řádek 0
  - dále hledám v seznamu slov na řádku 0