

# Červeno-černý strom a B-strom

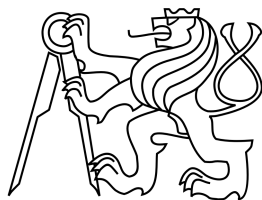
Karel Richta a kol.

Katedra počítačů  
Fakulta elektrotechnická  
České vysoké učení technické v Praze

© Karel Richta, Jan Drchal a kol., 2024

Datové struktury a algoritmy, B6B36DSA  
02/2024, Lekce 11

<https://moodle.fel.cvut.cz/course/view.php?id=5973>



Evropský sociální fond  
Praha & EU: Investujeme do vaší budoucnosti

# Obsah

- Červeno-černé stromy (Red-Black trees – RB)
- B-stromy (B-Trees)

Založeno na:

[Cormen, Leiserson, Rivest: Introduction to Algorithms, Chapter 14 and 19, McGraw Hill, 1990]

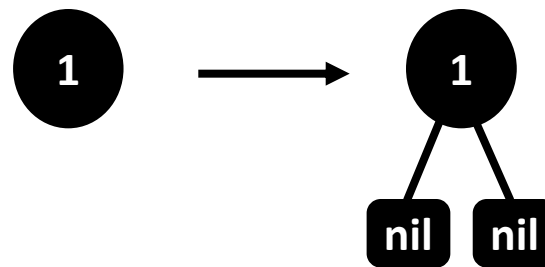
[Whitney: CS660 Combinatorial Algorithms, San Diego State University, 1996]

[Frederic Maire: An Introduction to Btrees, Queensland University of Technology, 1998]

<http://artax.karlin.mff.cuni.cz/~martin/ds/main.pdf>

# Vlastnosti červeno-černých stromů

- Jedná se o přibližně vyvážené BVS – platí:  
 $h_{RB} \leq 2x h_{BVS}$  (výška  $\leq 2x$  výška perfektně vyváženého stromu)
- Každý uzel obsahuje navíc bit pro barvu {red | black}
- Listy jsou doplněny odkazem na **nil** (neexistující potomek) = ukazatelem na **nil** - původní listy se stanou vnitřními uzly.

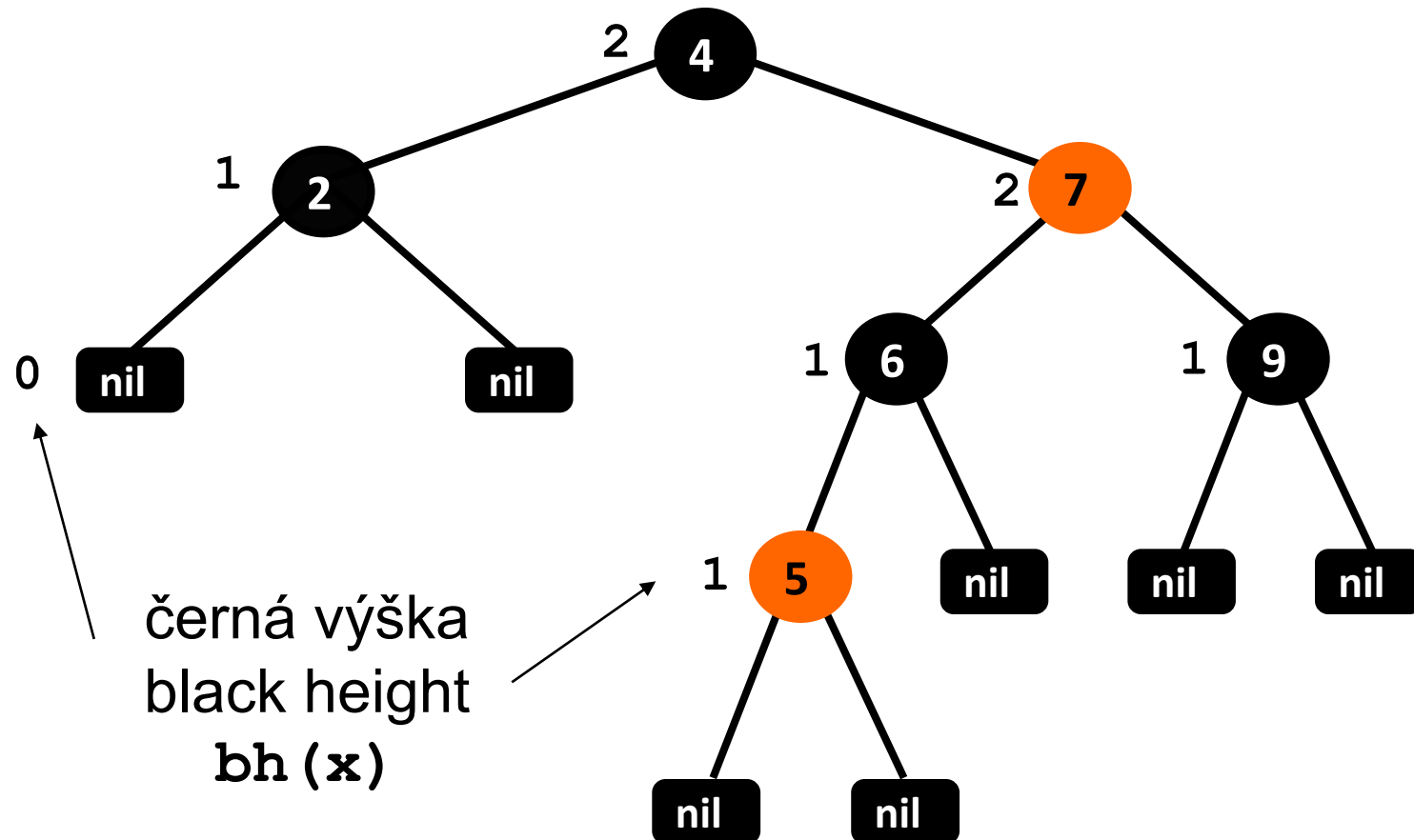


leaf → inner node

# Černá výška v červeno-černých stromech

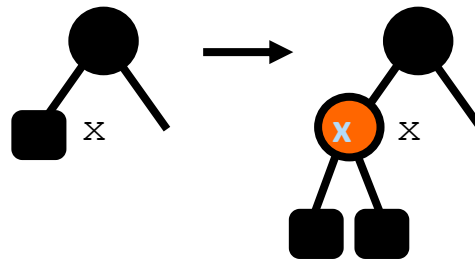
- Kořen RB-stromu je **černý**.
- Každá větev je zakončena (černým) uzlem **nil**.
- **Černá výška** uzlu  **$x$**  (**black-height**) se značí  **$bh(x)$**  a je definována jako počet černých uzlů na libovolné cestě z  **$x$**  do listu ( **$x$**  se nepočítá). Pozn.: Všechny tyto cesty jsou stejně dlouhé. Totéž platí pro kořen.
- Černá výška má rozsah od  **$h/2$**  (kde  **$h$**  je výška stromu), jestliže polovina uzlů je červených do  **$h$** , pokud jsou všechny uzly černé.
- **Výška  $h(x)$**  stromu s kořenem v  **$x$**  je maximálně dvakrát větší, než výška optimálně vyváženého stromu.
- **$h \leq 2\lg(n+1)$**                       ....  **$h \in \Theta(\lg(n))$**
- **$bh(x)$**  je v rozsahu od  **$\lg(n+1)$**  do  **$2\lg(n+1)$**

# Příklad černé výšky



# Operace INSERT pro RB-stromy

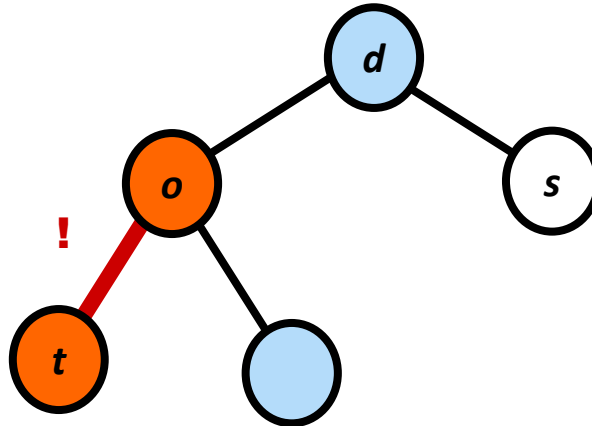
- 1) Nový uzel bude **červený**.
- 2) Vložíme uzel do stromu jako do standardního BVS.
- 3) Jestliže je předek černý, jsme hotovi → strom je **Červeno-Černý**.



- 4) Jestliže nikoliv, mohou nastat 3 následující případy.

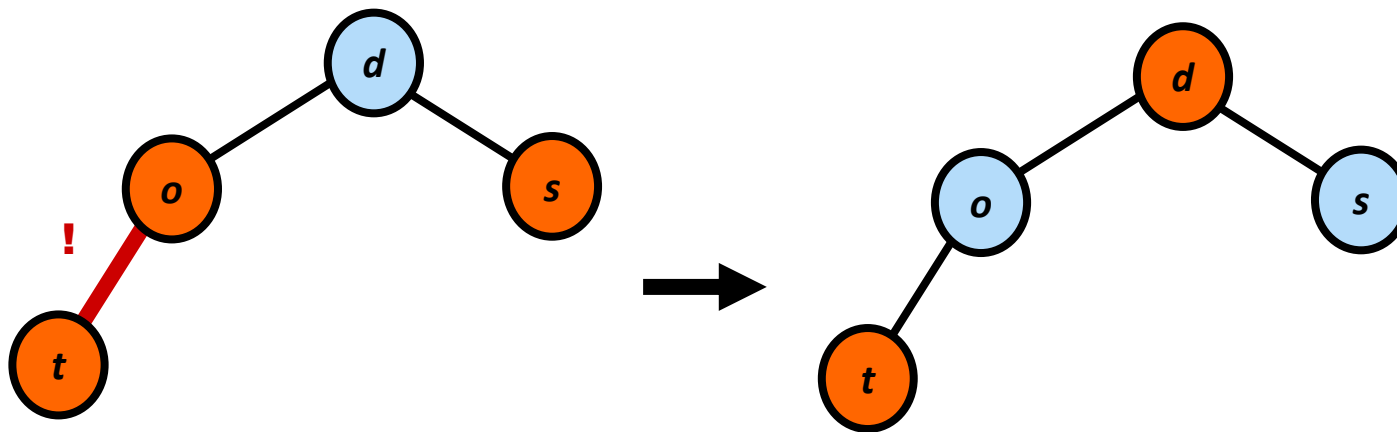
# Poruchy při vkládání

- Pokud je vrchol  $t$  červený a jeho otec je také červený, pak řekneme, že vložením  $t$  nastala porucha.
- Pokud nastala porucha, pak ji musíme nějak opravit. Situace je na obrázku - nejprve záleží na tom, jakou barvu má  $s$ , strýc  $t$ :



## Poruchy při vkládání (2)

1.  $s$  je **červený**. Pak pouze přebarvíme  $o$ ,  $d$  a  $s$  podle obrázku. Nyní  $d$  může být porucha, ovšem posunutá o 2 hladiny výše.



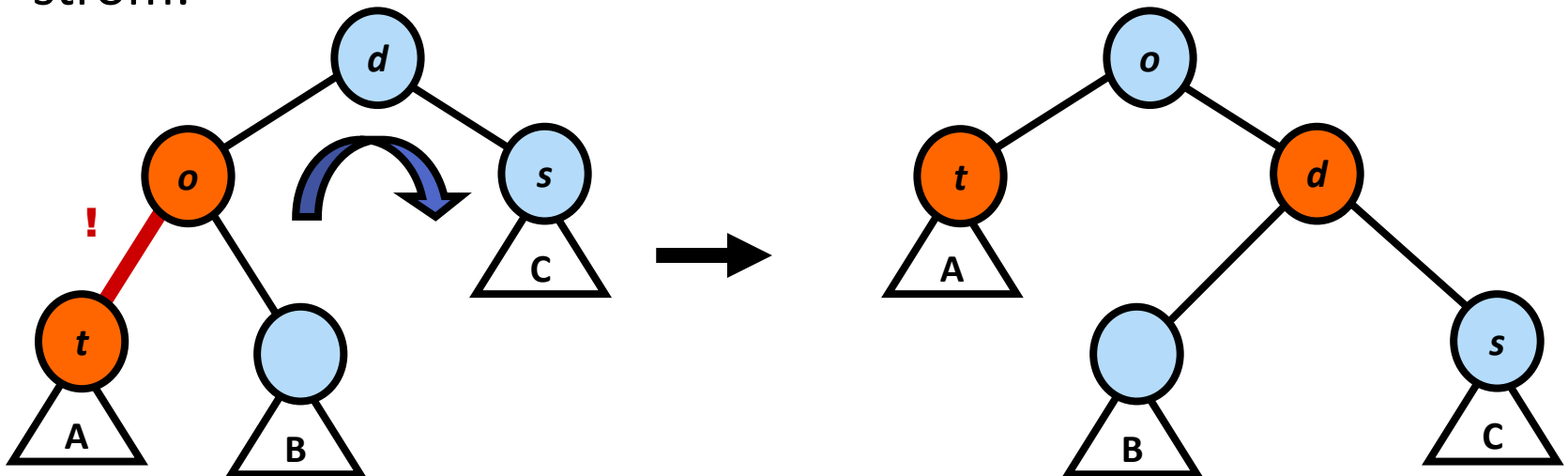
Nakonec je ještě nutné přebarvit kořen stromu na černo.



## Poruchy při vkládání (3)

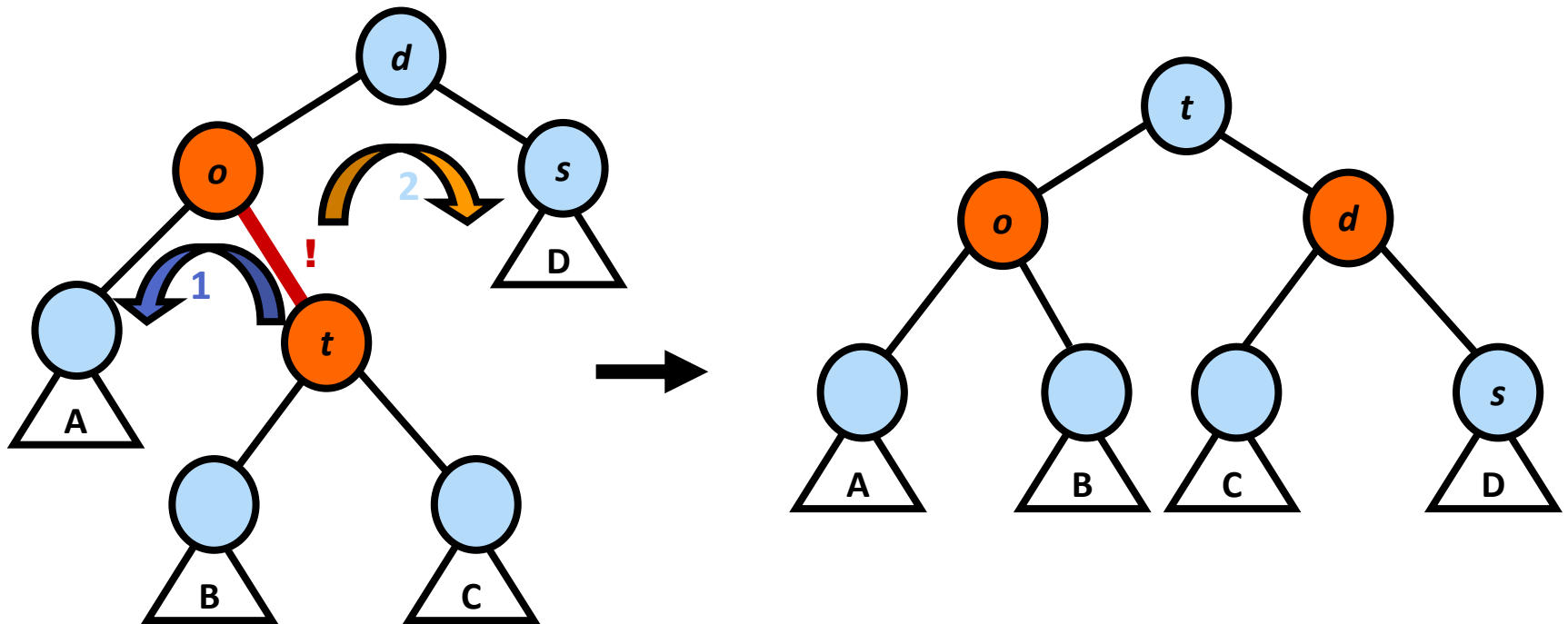
2.  $s$  je **černý**. Pak záleží na tom, zda hodnota  $t$  leží mezi hodnotami  $o$  a  $d$  nebo ne. Jinými slovy, zda cesta  $t$ - $o$ - $d$  obsahuje zatáčku.

(a) Bez zatáčky: Provedeme rotaci a přebarvíme podle obrázku. Splněny budou podmínky 1, 2 i 3, tedy máme **červeno**-černý strom:



## Poruchy při vkládání (4)

(b) Se zatáčkou. Provedeme dvojitou (*LR*) rotaci a přebarvíme podle obrázku. Splněny budou podmínky 1, 2 i 3, opět máme rovnou **červeno**-černý strom.



# Vkládání (Insert) v RB-stromech

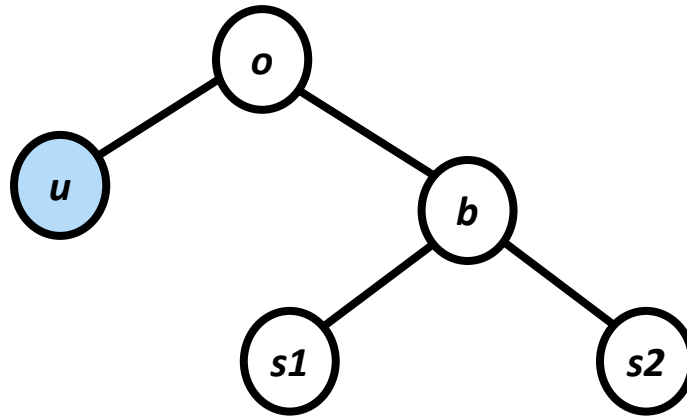
- Časová složitost je  $O(\log(n))$ .
- Vyžaduje maximálně dvě rotace.
- DEMO:
- <https://www.cs.usfca.edu/~galles/visualization/RedBlack.html>
- (Intuitivní, dobré pro porozumění)
- <http://www.youtube.com/watch?v=vDHFF4wjWYU>

# Operace DELETE pro RB-stromy

- Zatímco INSERT se příliš nelišil od své obdoby u AVL stromů, operace DELETE u **červeno**-černých stromů je oproti AVL stromům složitější mentálně, ovšem jednodušší časově.
- Situace: odstraňujeme vrchol  $t$  a jeho syna, který je list.
- Druhého syna  $t$ ,  $u$ , dáme na místo smazaného  $t$  a začerníme ho. Tím máme splněny podmínky 1 a 2. Pokud byl ale  $t$  **černý**, chybí nám na cestách procházejících nyní  $u$  jeden černý vrchol.

# Poruchy při mazání

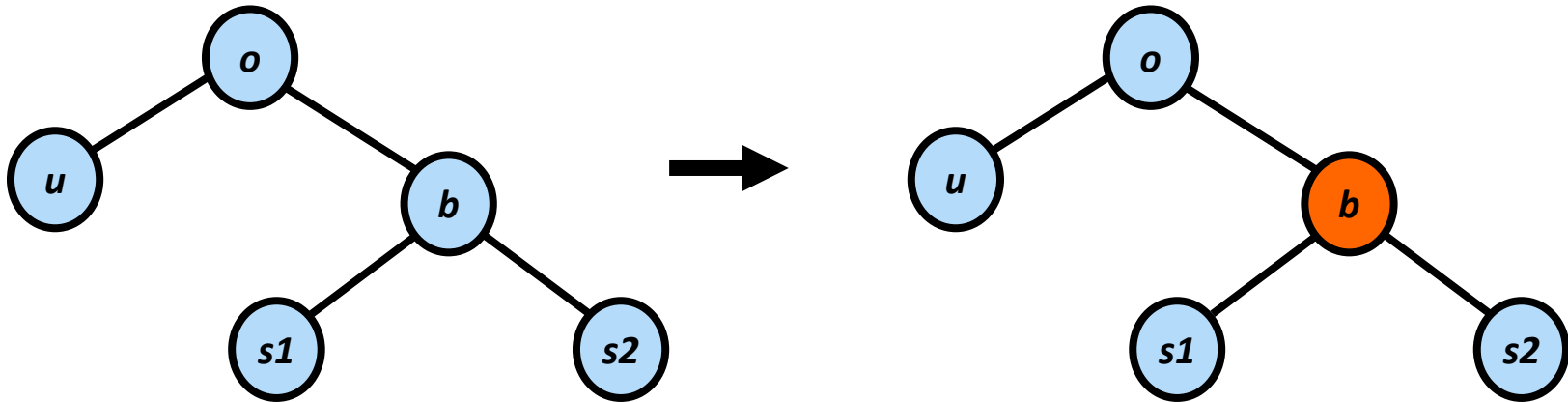
**Situace:** máme **červeno**-černý strom,  $u$  je porucha s otcem  $o$ , bratrem  $b$  a synovci  $s1$ ,  $s2$ , viz obrázek.



Oprava záležití na **barvě bratra** ( $b$ ):

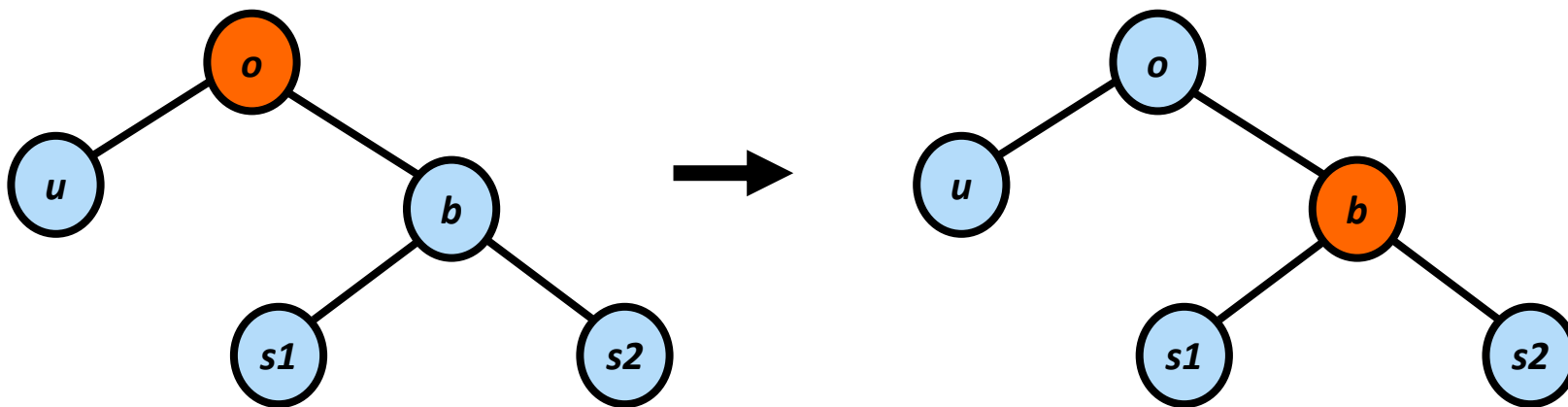
## Poruchy při mazání (2)

1. Bratr je **černý**. Rozlišujeme dále 4 případy, z nichž jeden způsobí poruchu o hladinu výše a ostatní skončí s **červeno-černým** stromem.
  - (a) Otec i synovci jsou **černí**. Přebarvíme  $b$  na **červeno**, viz obrázek, a tady porucha je o hladinu výše – provedli jsme tedy jakousi částečnou opravu.



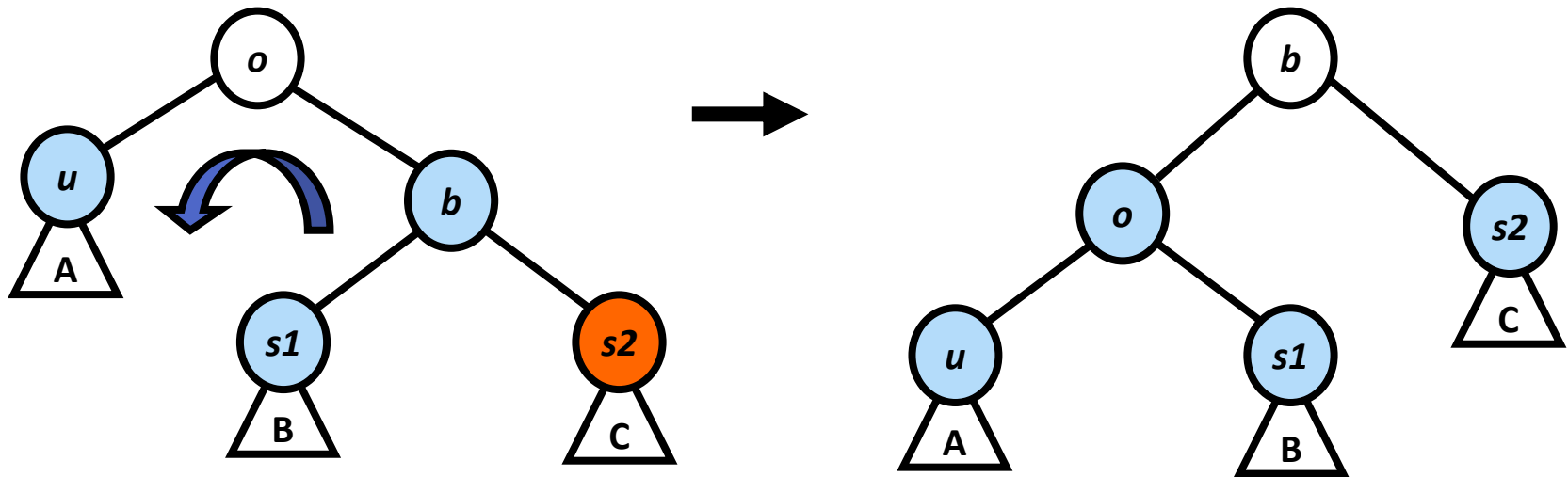
## Poruchy při mazání (3)

(b) Otec je **červený**, synovci **černí**. Přebarvíme otce a bratra podle obrázku a dostáváme **červeno-černý** strom.



## Poruchy při mazání (4)

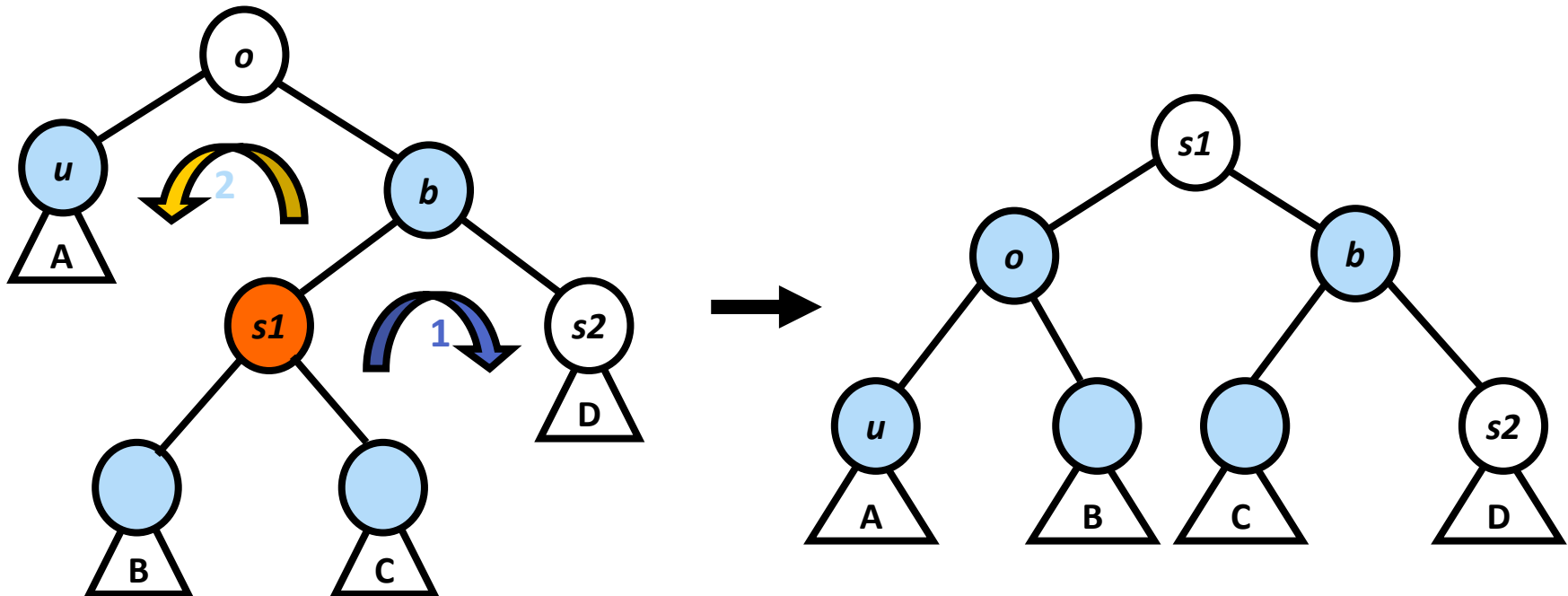
(c) Synovec  $s1$ , jehož hodnota leží mezi hodnotami otce a bratra, je **černý**, druhý synovec je **červený**. Přebarvíme a rotujeme podle obrázku, barva otce se nemění (tj., vrchol  $b$  bude mít barvu, kterou původně měl vrchol  $o$ ). Dostáváme **červeno-černý** strom.





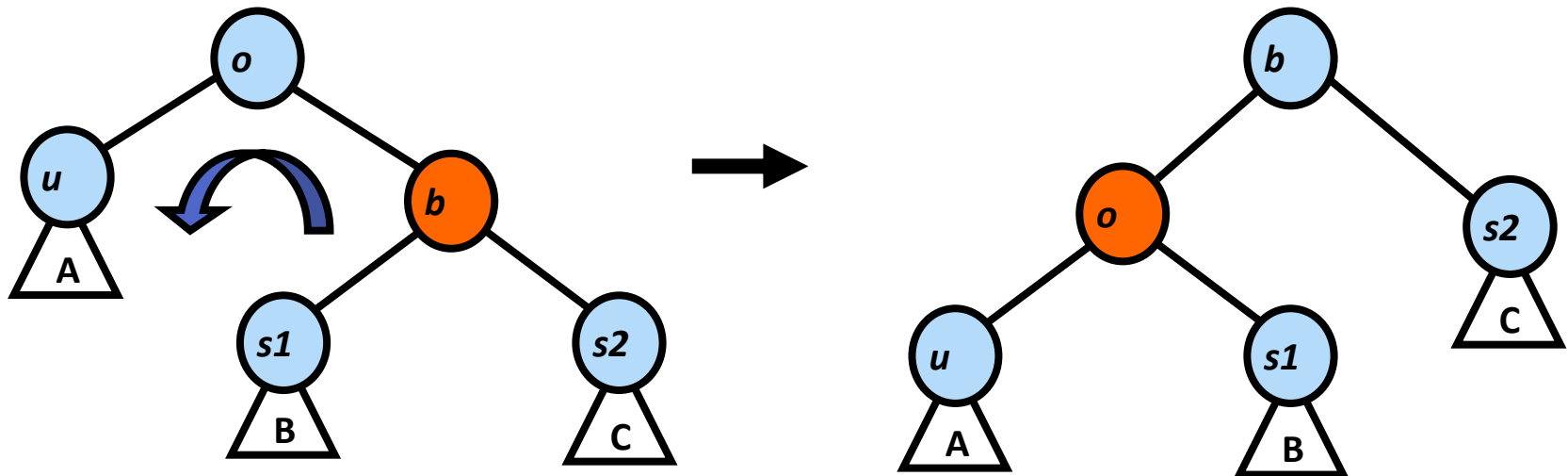
## Poruchy při mazání (5)

(d) Synovec  $s1$ , jehož hodnota leží mezi hodnotami otce a bratra, je **červený**, druhý synovec má libovolnou barvu. Přebarvíme a  $RL$  rotací upravíme podle obrázku (tj. vrchol  $s1$  bude mít barvu, kterou měl původně vrchol  $o$  a barva vrcholu  $s2$  se nezmění). Dostáváme **červeno-černý** strom.



## Poruchy při mazání (6)

2. Bratr je **červený**. Přebarvíme a rotujeme podle obrázku. Dostáváme **červeno**-černý strom s poruchou, přičemž porucha je o hladinu níže. I když to tak na první pohled nevypadá, máme vyhráno, protože bratr poruchy je **černý** a otec **červený**, tedy příští oprava bude případ 1b, 1c, nebo 1d a skončíme s **červeno**-černým stromem.



# Mazání v RB-stromě

- Časová složitost je  $O(\log(n))$ .
- Jsou zapotřebí maximálně tři rotace.

# Důkaz výšky RB-stromu

**Věta:** RB-strom s  $n$  interními uzly má výšku  $h$  nejvýše  $2\lg(n+1)$ .

Důkaz:

- Ukažme, že podstrom s kořenem v  $x$  obsahuje nejméně  $2^{bh(x)}-1$  interních uzlů. Indukcí dle výšky podstromu s kořenem v  $x$ :
  - Pokud je  $x$  list, pak  $bh(x) = 0$ ,  $2^{bh(x)}-1 = 0$  interních uzlů (uzel **nil**).
  - Předpokládejme, že výška  $x$  je  $h$ , potomci  $x$  mají výšku  $h-1$ :
    - černá výška potomků  $x$  je buď  $bh(x)-1$  nebo  $bh(x)$  (podle barvy),
    - indukční předpoklad je, že potomci  $x$  mají nejméně  $(2^{bh(x)-1})-1$  interních uzlů,
    - pak podstrom s kořenem v  $x$  obsahuje nejméně:
 
$$2^{bh(x)-1} - 1 + 2^{bh(x)-1} - 1 + 1 = 2^{bh(x)} - 1$$
 interních uzlů.
  - Necht'  $h$  je výška stromu, který rotujeme v  $x$ :
    - $bh(x) \geq h / 2$  (max  $\frac{1}{2}$  je červených),
    - pak  $n \geq 2h/2 - 1 \Leftrightarrow n + 1 \geq 2h/2 \Leftrightarrow \lg(n+1) \geq h / 2$  a
    - $h \leq 2\lg(n+1)$ .

# Zhodnocení pro RB-stromy

- Pro binární vyhledávací červeno-černé stromy lze implementovat SEARCH, INSERT a DELETE tak, že vyžadují čas  $O(\log n)$  a INSERT používá nejvýše jednu (dvojitou) rotaci a DELETE používá nejvýše dvě rotace nebo rotaci a dvojitou rotaci.
- RB-stromy jsou lepší než AVL stromy, které pro DELETE potřebují až  $\log n$  rotací. Oproti váhově vyváženým stromům i proti AVL stromům jsou červeno-černé stromy sice jen konstantně lepší, ale i to je dobré.
- Při použití binárních vyhledávacích stromů ve výpočetní geometrii nese informaci i rozložení prvků ve stromě, a tato informace se musí po provedení rotace nebo dvojitě rotace aktualizovat. To znamená prohledání celého stromu a tedy čas  $O(n)$  za každou rotaci a dvojitou rotaci navíc.
- Pro tyto problémy jsou červeno-černé stromy obzvláště vhodné, protože minimalizují počet použitých rotací a dvojitých rotací.

# B-stromy (B-trees)

1. Motivace
2. Vícecestné vyhledávací stromy (multiway search trees)
3. Definice B-stromu
4. Search
5. Insert
6. Delete (jen přehledově...)

# Motivace pro B-stromy

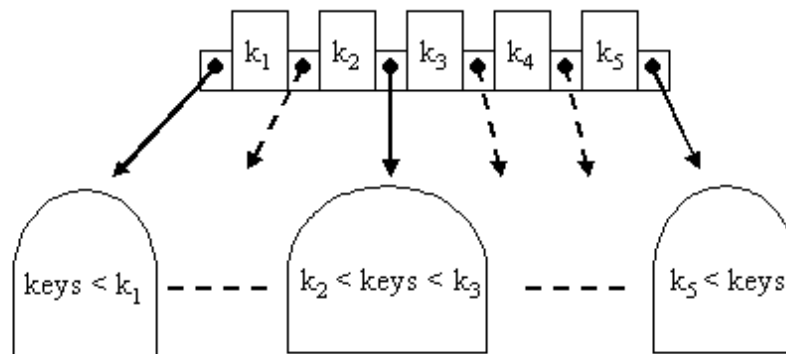
- Velká data se nevejdou do operační paměti -> používáme disky (vnější paměti s přímým přístupem).
- Čas přístupu na disk je určen HW a OS.
- Přístup na disk je MNOHEM pomalejší, v porovnání s přístupem do operační paměti:
  - 1 diskový přístup ~ 13 000 000 instrukcí!!!!
  - Počet diskových operací dominuje v odhadu časové náročnosti.
- Přístup na disk = Disk-Read, Disk-Write
  - Disk je rozdělen do bloků (512, 2048, 4096, 8192 bytes), které se přenášejí jako celek.
  - Proto se může hodit návrh vícecestných vyhledávacích stromů (*multiway search trees*), kde každý uzel stromu „pasuje“ do jednoho bloku na disku.

DISK : 16 ms  
Seek 8ms + rotational  
delay 7200rpm 8ms

Instruction:  
800 MHz 1,25ns

# Vícecestné vyhledávací stromy

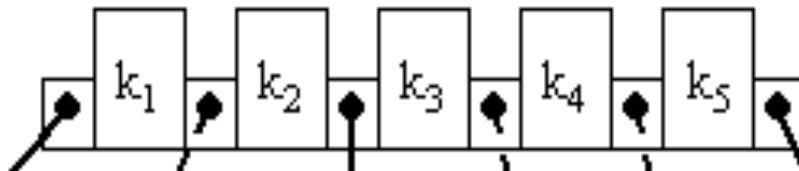
- Zobecnění binárních vyhledávacích stromů, kde počet následníků byl  $m=2$ .
- Každý uzel vícecestného stromu má nejvýše  $m$  potomků ( $m>2$ ).
- Interní uzly obsahující  $n$  klíčů mají  $n+1$  následníků,  $n < m$  (výjimkou může být kořen stromu).
- Listy nemají následníky.
- Strom je uspořádán.
- Klíče v uzlech vymezují intervaly v podstromech - viz obrázek:





# Listy vícecestného vyhledávacího stromu

- Listy vícecestného vyhledávacího stromu nemají žádné následníky a neobsahují žádné ukazatele na podstromy. Obvykle obsahují vlastní hodnoty.
- Platí:  $k_1 \leq k_2 \leq \dots \leq k_5$

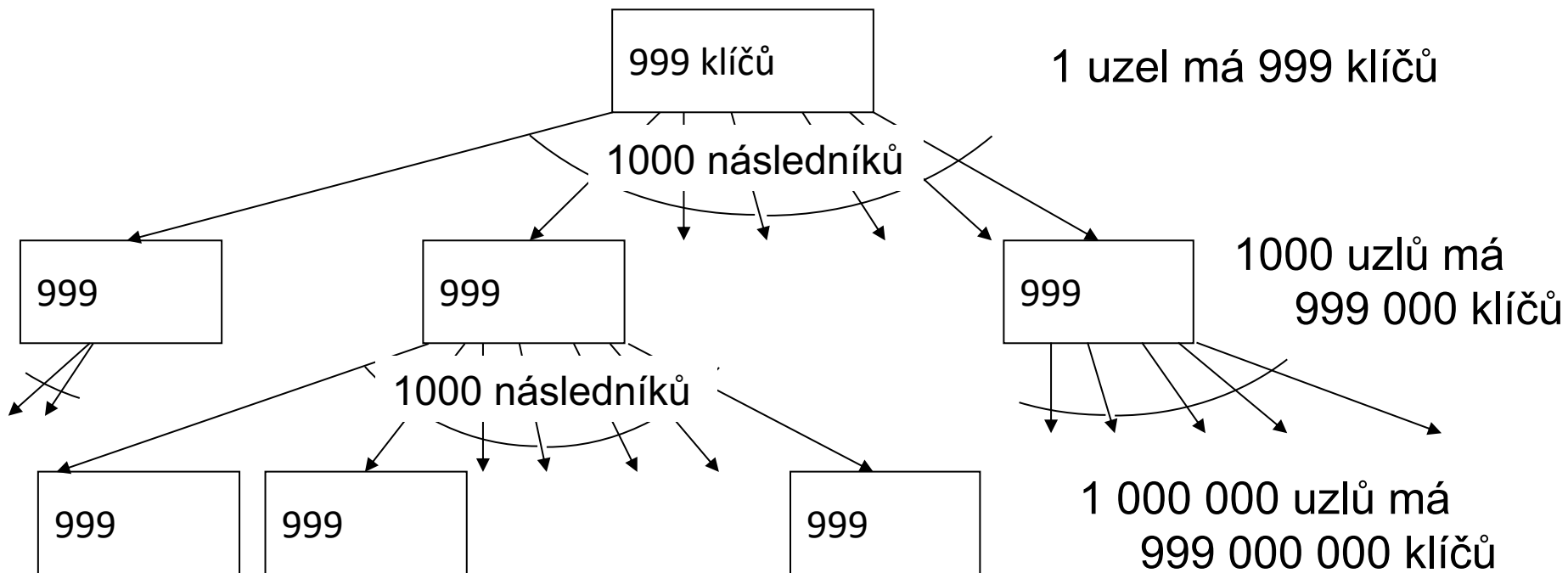


© Frederic Maire, QUT

# B-stromy (B-trees)

- **B-strom** je vícecestný vyhledávací strom minimálního stupně  $t$ , který splňuje další podmínky:
- Všechny listy mají stejnou výšku (B-strom je vyvážený).
- Kořen může mít 0 až  $2t$  potomků (max  $2t-1$  klíčů)
- Pro všechny ostatní uzly platí:
  - mají nejméně:  $t-1$  klíčů (vnitřní uzly, krom kořene,  $t$  potomků)
  - mají nejvýše:  $2t-1$  klíčů (vnitřní uzly  $2t$  potomků).

# Příklad B-stromu

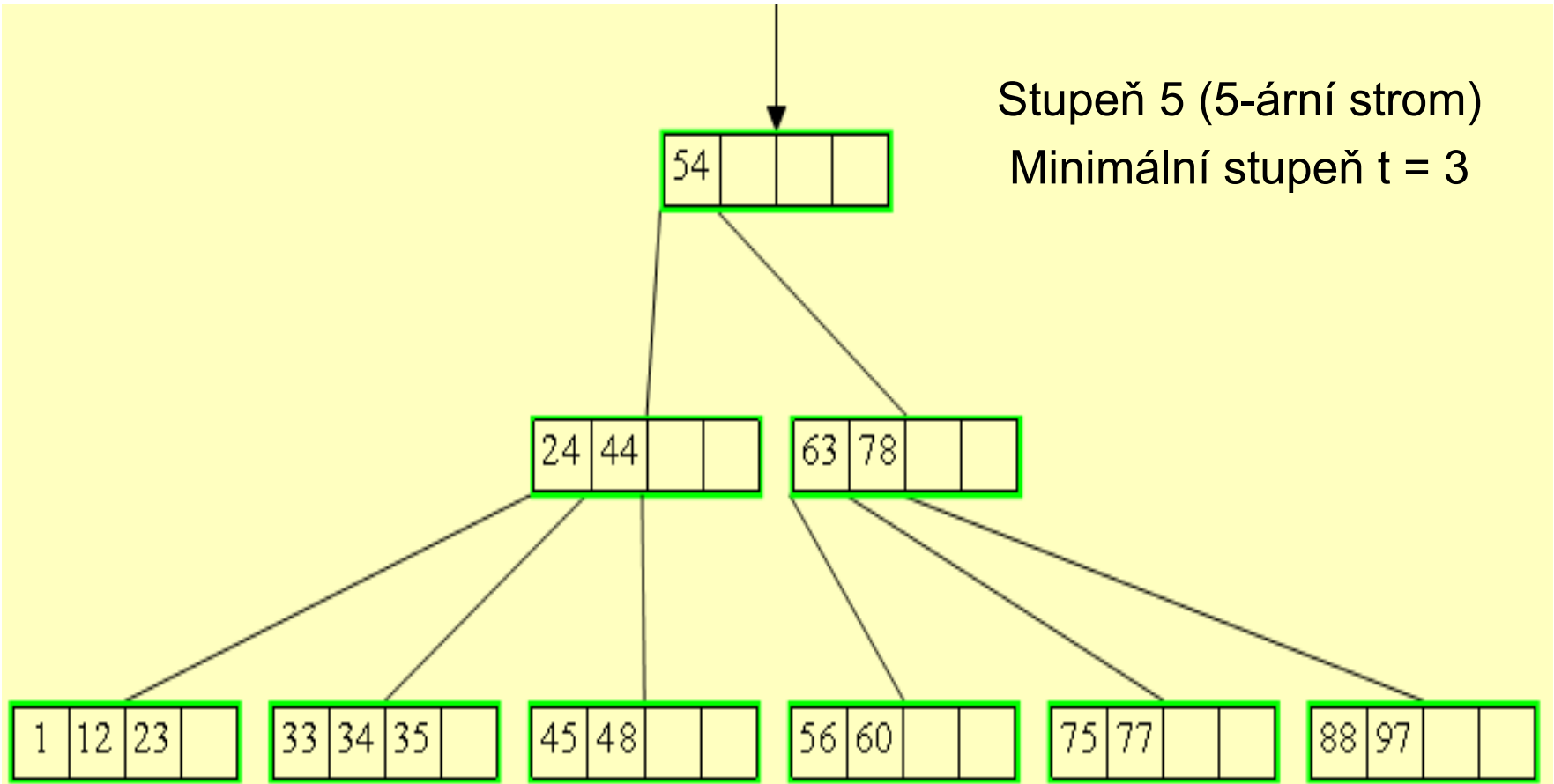


B-strom minimálního stupně  $t = 500$ , výšky 2 obsahuje maximálně  
 1 001 001 uzlů ( $1+1000 + 1\,000\,000$ )  
 999 999 999 klíčů ~ 1 miliarda klíčů

# Obsah uzlů B-stromu

- $n$  ... počet klíčů  $k_i$  uložených v uzlu.
- Uzel, kde  $n = 2t-1$  se nazývá **plný uzel (full-node)**.
- $k_i$  ...  $n$  klíčů, uložených v neklesajícím pořadí
- $k_1 \leq k_2 \leq \dots \leq k_n$
- *list* ... booleovská hodnota, `true` pro list, `false` pro ostatní.
- $c_i$  ...  $n+1$  ukazatelů na následníky (nedefinováno pro listy).
- Pro klíče  $k_i$  v podstromech platí:
- $keys_1 \leq k_1 \leq keys_2 \leq k_2 \leq \dots \leq k_n \leq keys_{n+1}$

# B-strom



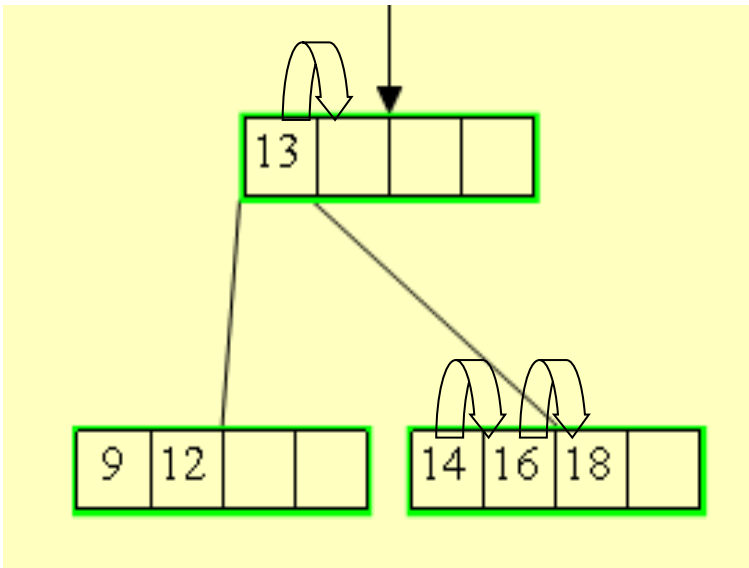
Based on [Cormen] and [Maire]

# Hledání v B-stromech

- Obdobné hledání v BVS.
- V rámci uzlů se hledá sekvenčně, nebo binárně.
- Vstup: ukazatel na kořen stromu a klíč  $k$
- Výstup: uspořádaná dvojice  $(y, i)$ , kde  $y$  je uzel a  $i$  je index v rámci uzlu takový, že  $y.k[i] = k$  nebo hodnota NIL, pokud klíč  $k$  nebyl nalezen.

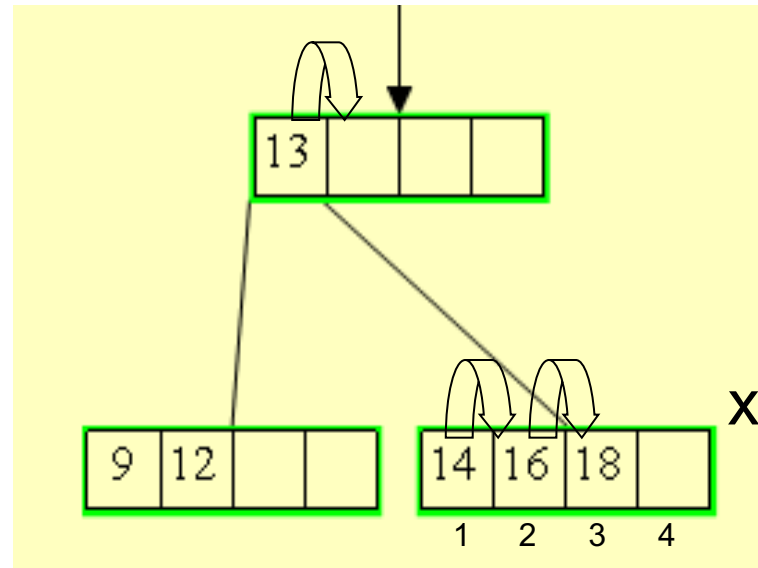
# Příklad hledání v B-stromu

- Hledej 17 17



17 not found => return NIL

- Hledej 18 18



18 found => return (x, 3)

# Operační složitost hledání v B-stromu

- Počet čtení z disku:
- $O(h) = O(\log_t n)$
- kde  $h$  je výška stromu a  $t$  je minimální stupeň stromu a  $n$  je počet uzlů stromu.
- Protože počet klíčů je  $x.n < 2t$ , celá smyčka má horní odhad  $O(t)$  a celková časová složitost je  $O(t \log_t n)$ .

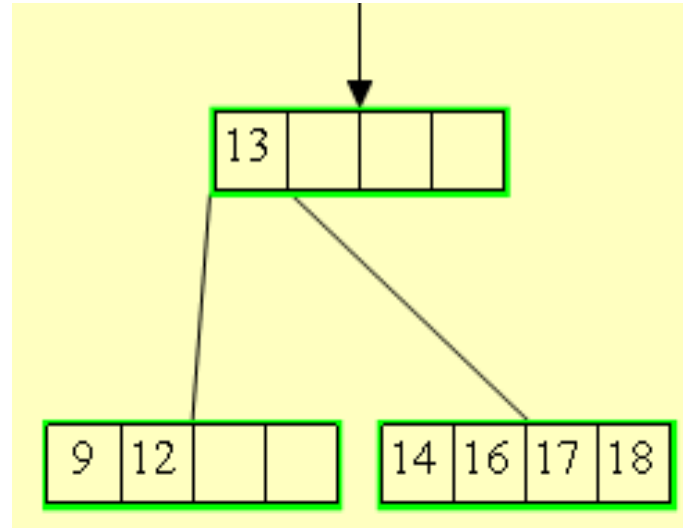
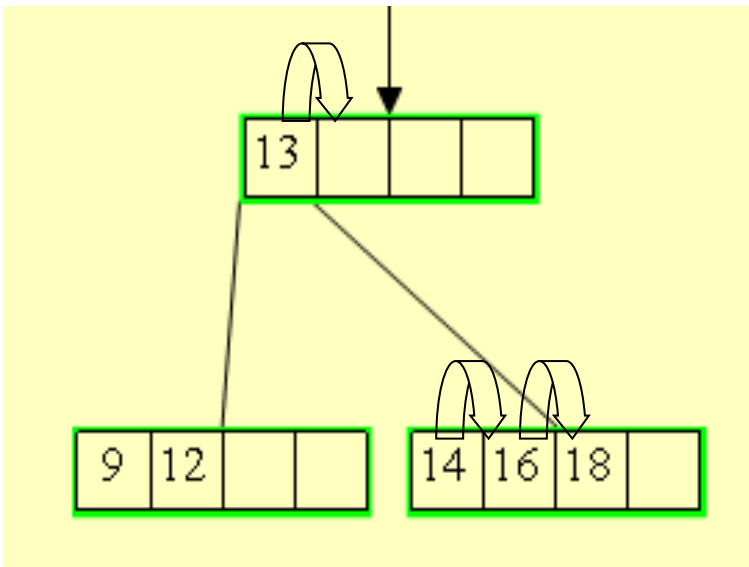


# B-stromy: insert - 1. vícefázová strategie

Insert do uzlu, který není zcela naplněn (**not full node**)

- Insert 17

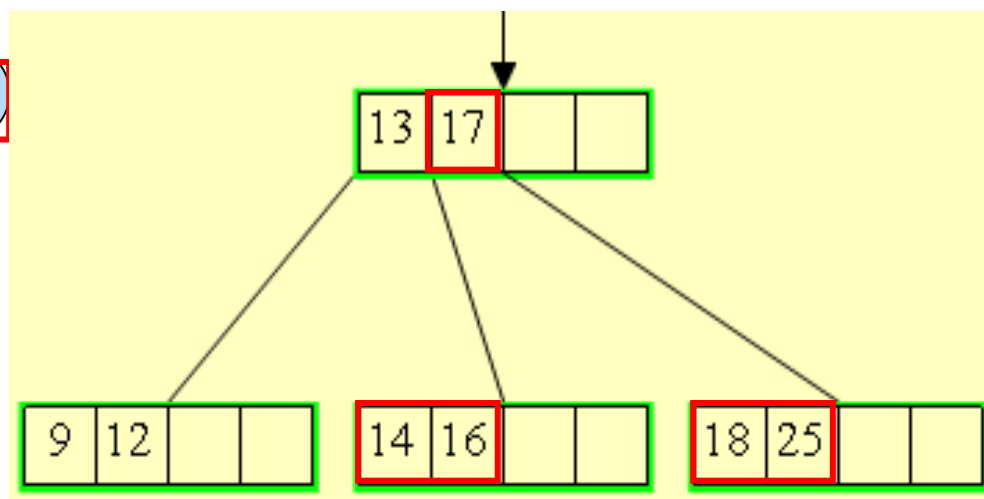
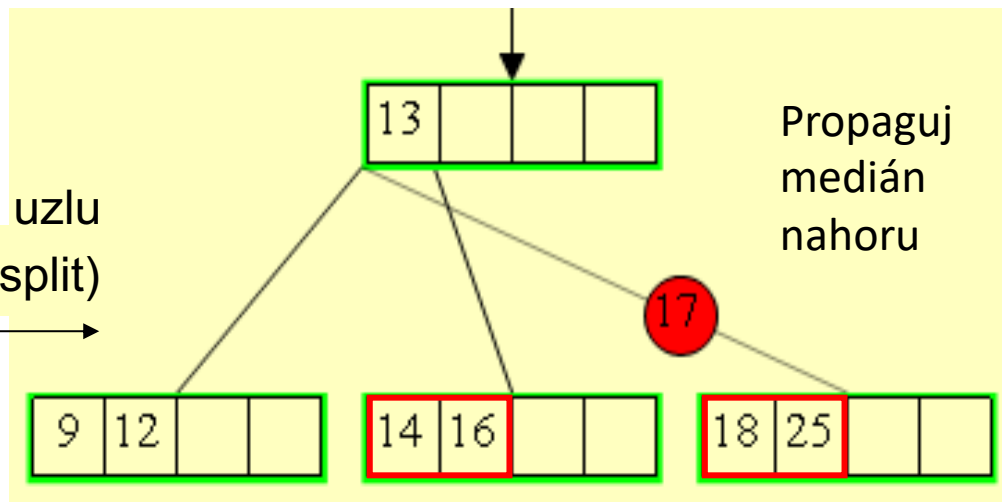
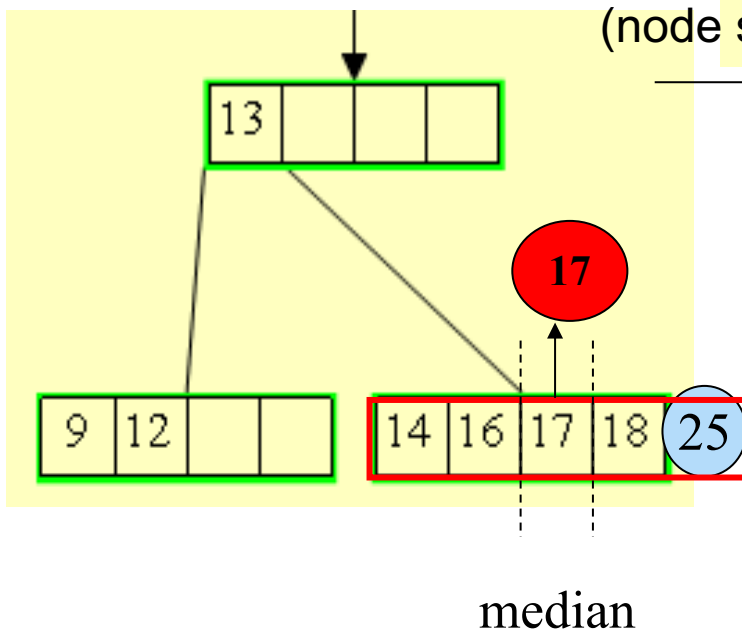
17



# B-stromy: insert - 1. vícefázová strategie

Insert do „plného uzlu“

- Insert 25



1. Vícefázová strategie:  
“řeš problém, až když se vyskytne”

# B-stromy: insert - 1. vícefázová strategie

Insert ( $x, T$ ) - pseudokód

- Najdi list pro  $x$
- Pokud uzel není plný, insert  $x$  a konec
- **while** (je aktuální uzel plný)
- najdi medián (z klíčů po vložení  $x$ )
- rozděl uzel (split node) do dvou uzlů
- propaguj medián směrem nahoru
- aktuální uzel = rodič aktuálního uzlu

Fáze top-down

(node overflow)

Fáze bottom-up

# B-stromy: insert - 2. jednofázová strategie

Princip: “řeš problémy, než nastanou”

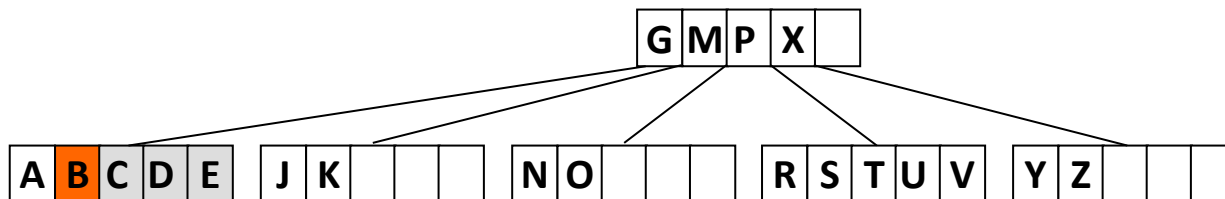
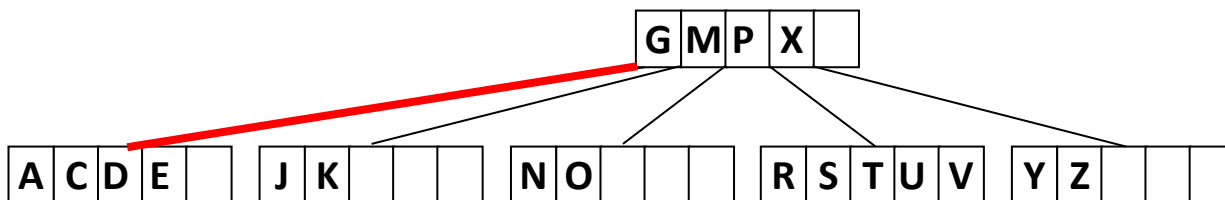
Pouze fáze top-down

- Rozděl před vstupem naplněný uzel.
- Tím si vytvoříš prostor pro budoucí medián z potomků.
- Fáze „bottom-up“ není zapotřebí.
- Rozdělení:
  - kořen => strom naroste o úroveň,
  - vnitřní uzel nebo list => rodič obdrží klíč, který je medián

# B-stromy: insert - 2.jednofázová strategie

Insert do nenaplněného uzlu (not full)

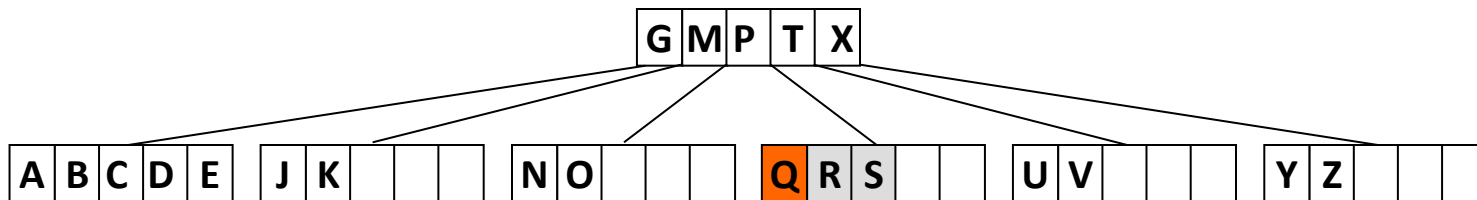
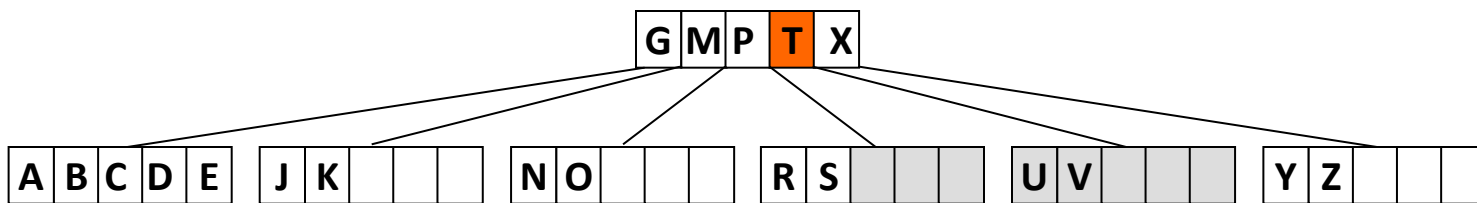
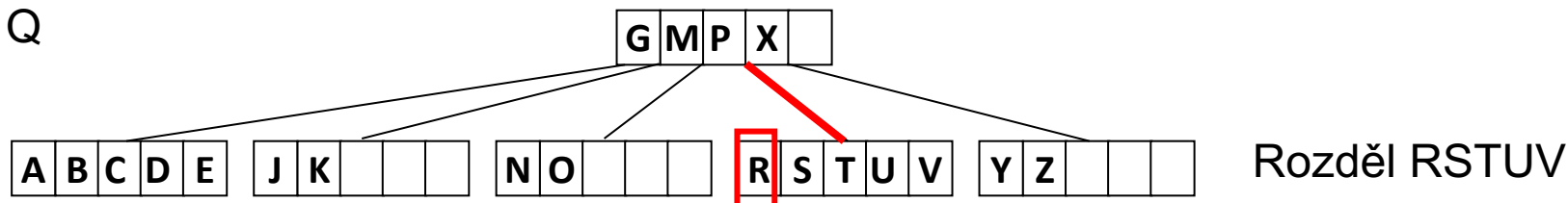
Vlož B



# B-stromy: insert - 2.jednofázová strategie

Přeskok „full node“ a vložení do „not full“

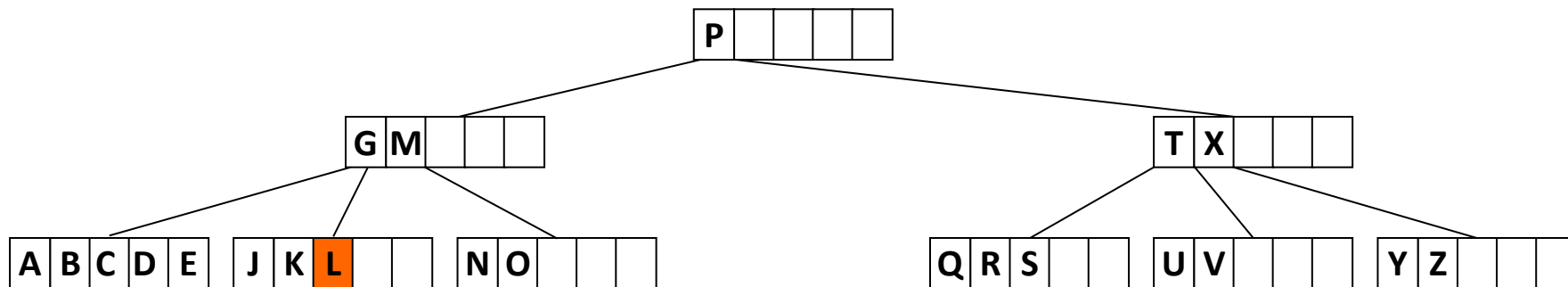
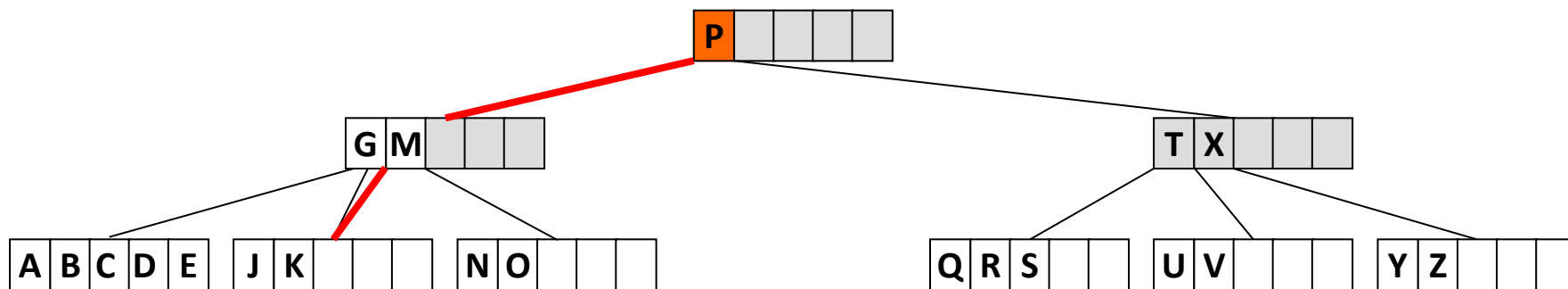
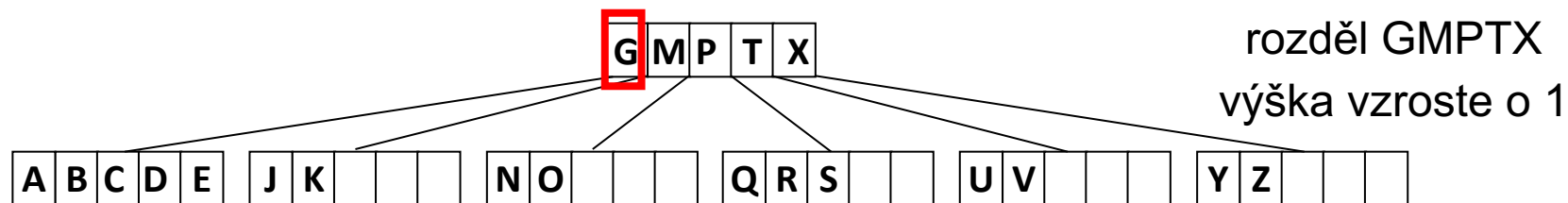
Vlož Q



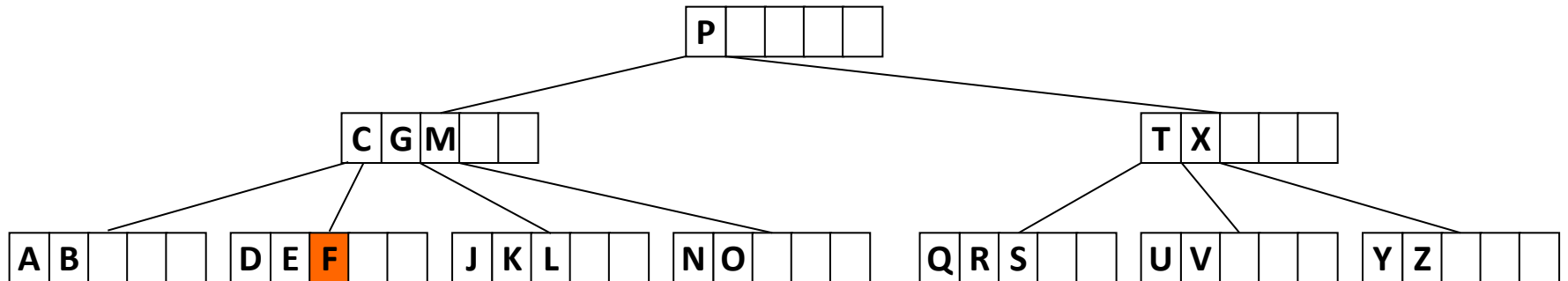
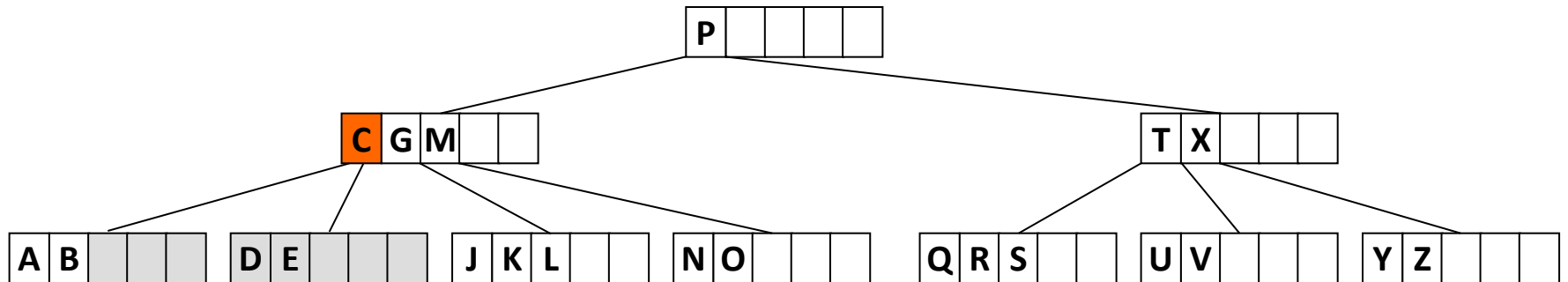
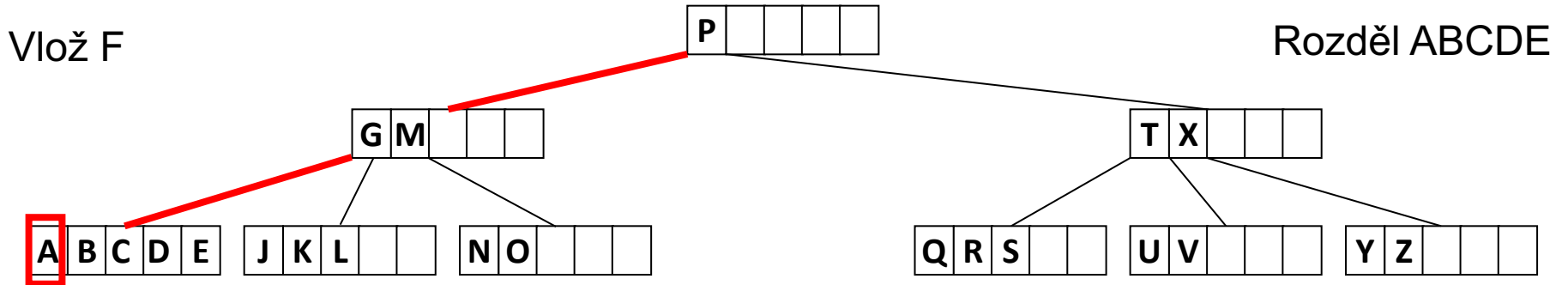
# B-stromy: insert - 2.jednofázová strategie

Ignorování „full node“ a vložení do „not full“ uzlu

Vlož L



# B-stromy: insert - 2.jednofázová strategie





# B-stromy: insert - 2.jednofázová strategie

Insert ( $x, T$ ) - pseudokód

- While hledej listy  $x$
- if (je uzel plný)
- najdi medián (mezi klíči naplněného uzlu)
- rozděl (split) uzel na dva
- vlož (insert) medián do rodiče (parent).
- Insert  $x$  a zastav.

Pouze fáze top-down

# Strategie pro úpravy B-stromu

## Dvě principiální strategie

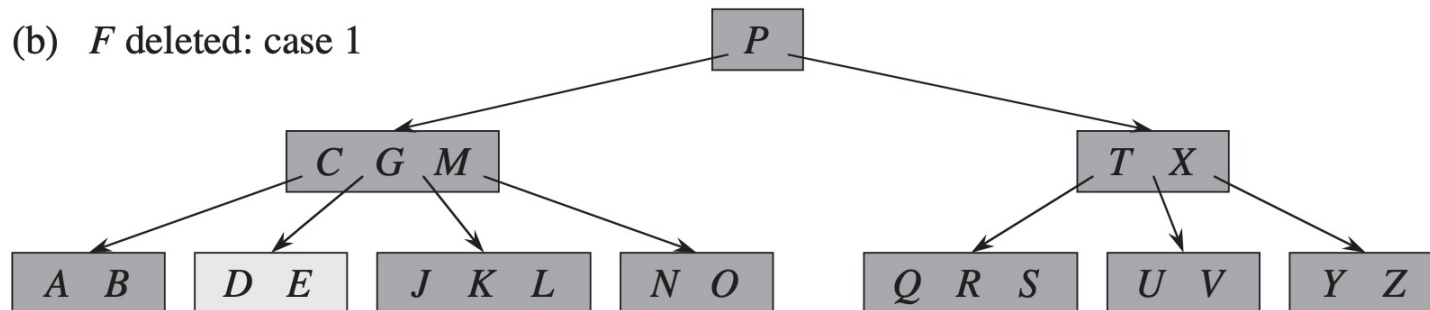
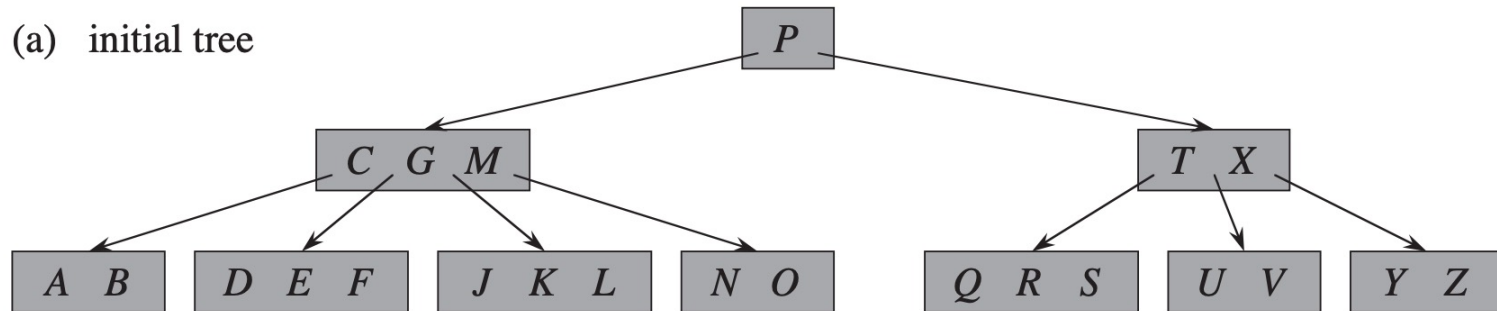
1. Více-fázová strategie:
  - “řeš problém, až když se vyskytne”
2. Jedno-fázová strategie: [Cormen]
  - “řeš problémy, než nastanou”
  - Akce:
    - Rozděl plné uzly
    - Sluč uzly, které mají minimální obsah (méně než minimum položek).

# Delete pro B-stromy

## Delete ( $x, T$ ) - principy

- Při průchodu opravujeme uzly, které mají minimální počet klíčů ( $t-1$ )
- Pokud mají oba potomci jen  $t-1$  klíčů => operace *merge*: opačná operace k *split*
- Pokud jen jeden => přesun předchůdce/následníka do rodiče
- Pokud kořen nemá žádný klíč, zrušíme jej a nahradíme jeho jediným potomkem => snížení výšky stromu

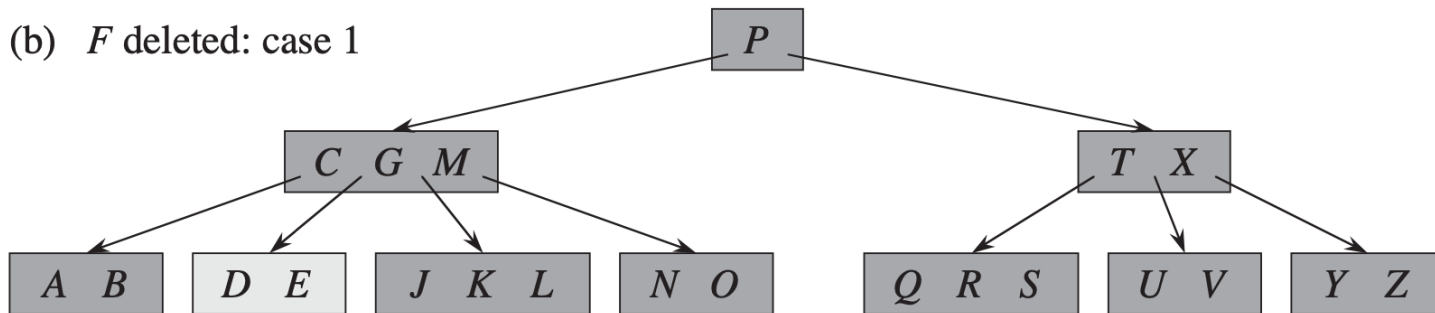
# Delete pro B-stromy (ukázka)



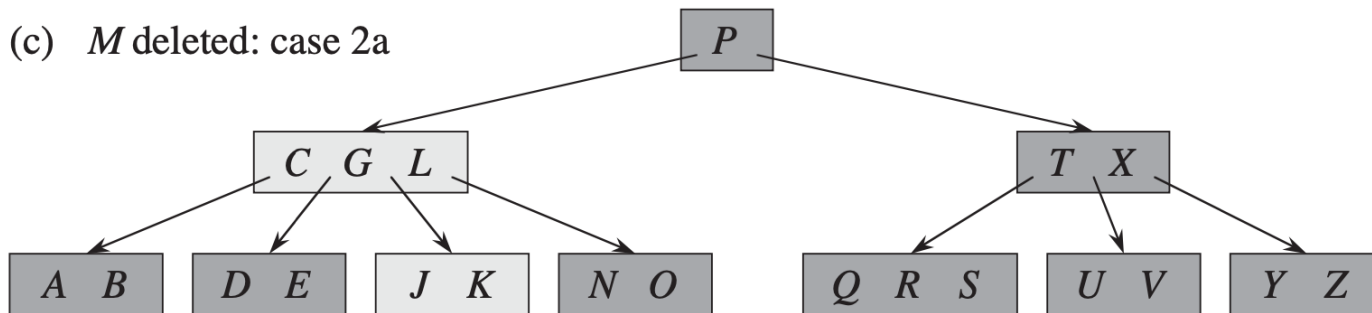
Minimální stupeň  $t = 3$

# Delete pro B-stromy (ukázka)

(b) *F* deleted: case 1

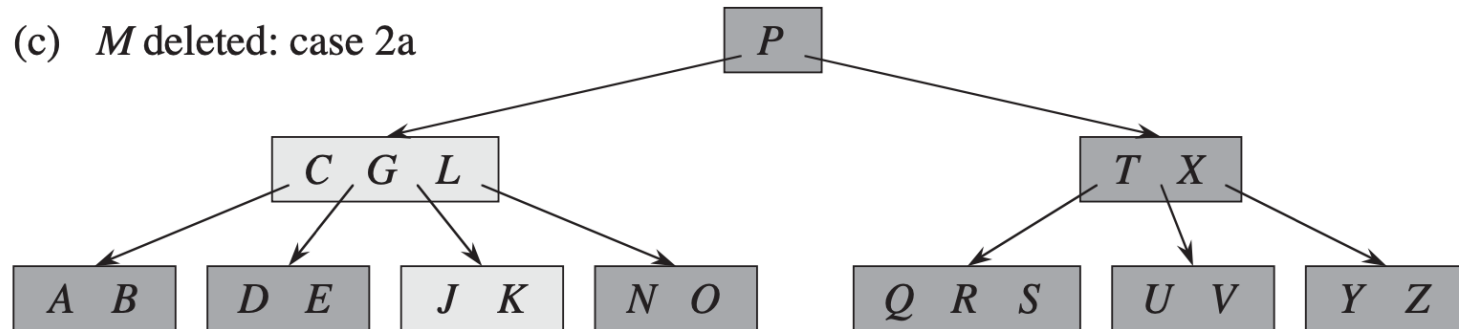


(c) *M* deleted: case 2a

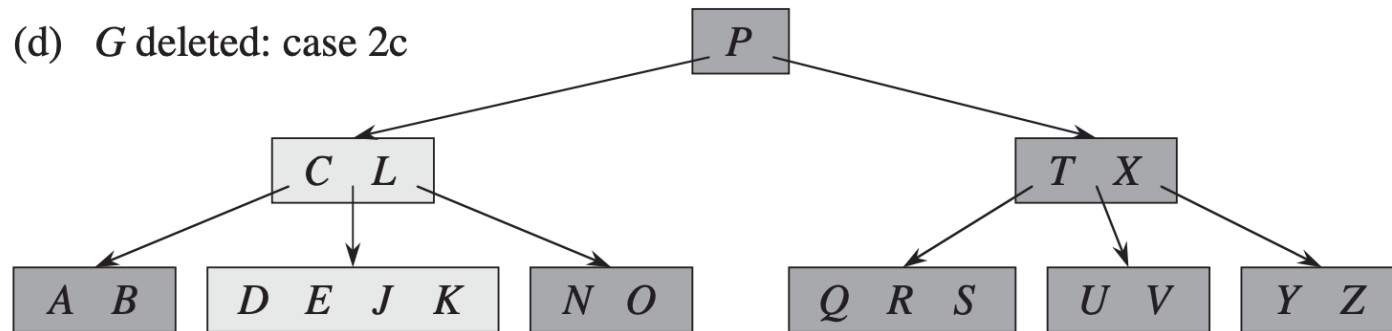


# Delete pro B-stromy (ukázka)

(c) *M* deleted: case 2a

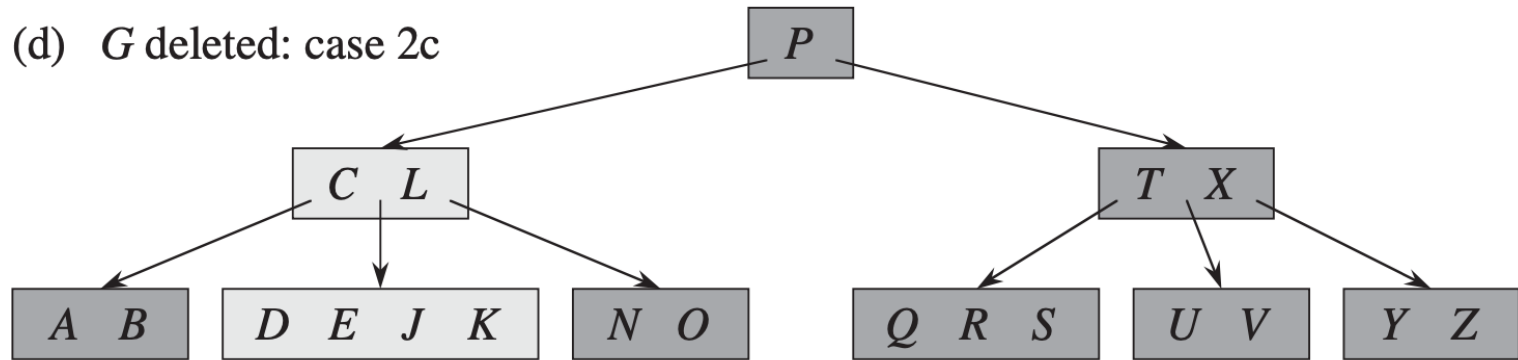


(d) *G* deleted: case 2c

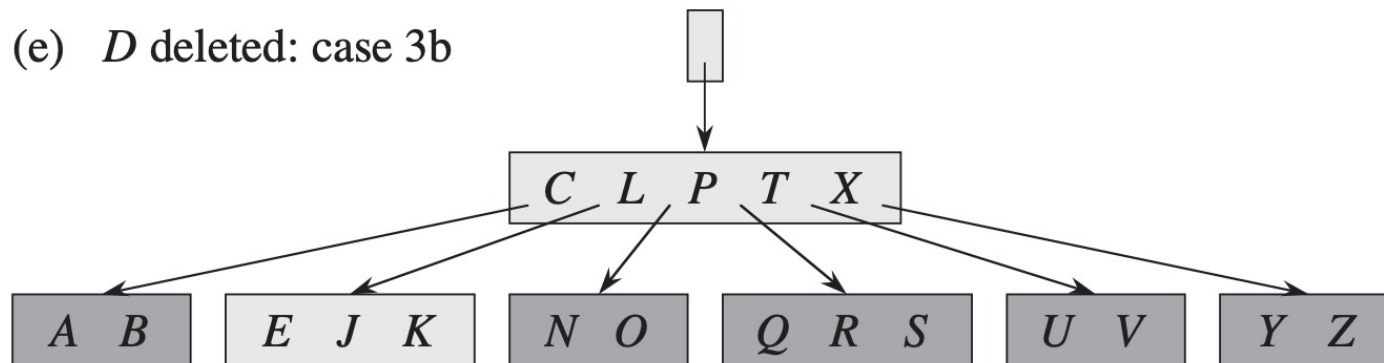


# Delete pro B-stromy (ukázka)

(d) *G* deleted: case 2c

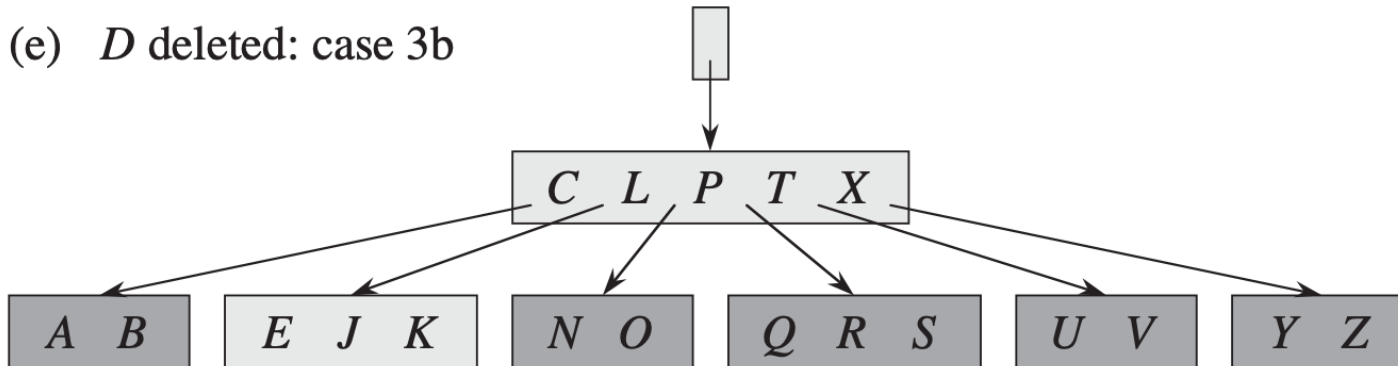


(e) *D* deleted: case 3b

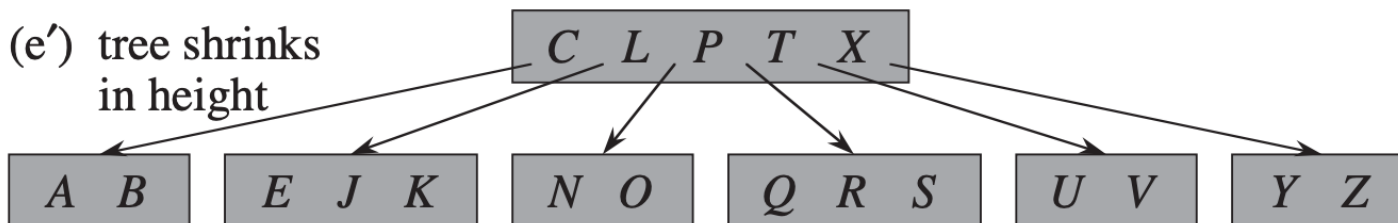


# Delete pro B-stromy (ukázka)

(e) *D* deleted: case 3b

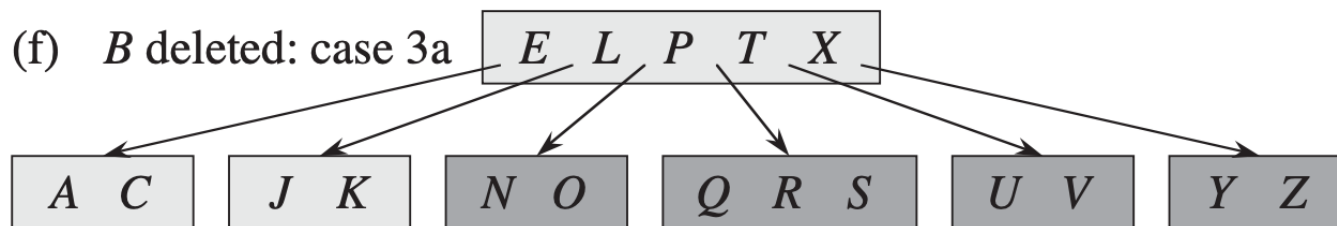
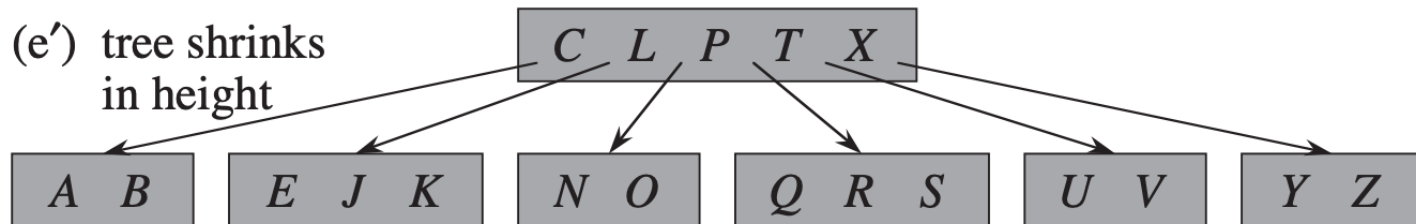


(e') tree shrinks  
in height





# Delete pro B-stromy (ukázka)



# Maximální výška B-stromu

- $h \leq \log_{\lceil m/2 \rceil} ((N+1)/2)$
- Dává horní hranici přístupů na disk.
- Viz [Maire] nebo [Cormen] podrobnosti

# References

- [Cormen] Cormen, Leiserson, Rivest: Introduction to Algorithms, Chapter 14 and 19, McGraw Hill, 1990
- [Whitney]: CS660 Combinatorial Algorithms, San Diego State University, 1996], RedBlack, B-trees  
<http://www.eli.sdsu.edu/courses/fall96/cs660/notes/redBlack/redBlack.html#RTFToC5>
- [Wiki] B-tree. (2006, November 24). In Wikipedia, The Free Encyclopedia. Retrieved 20:25, December 12, 2006, from  
<http://en.wikipedia.org/w/index.php?title=B-tree&oldid=89805120>
- [Maire] Frederic Maire: An Introduction to Btrees, Queensland University of Technology, 1998]  
<http://sky.fit.qut.edu.au/~maire/baobab/lecture/>
- [RB tree] John Franco - [java applet](#)
- <http://www.ececs.uc.edu/~franco/C321/html/RedBlack/redblack.html>
- [Jones] Jeremy Jones: B-Tree animation - [java applet](#)  
<https://www.cs.tcd.ie/Jeremy.Jones/vivio/trees/B-tree.htm>

# The End