DATOVÉ STRUKTURY – STROMY

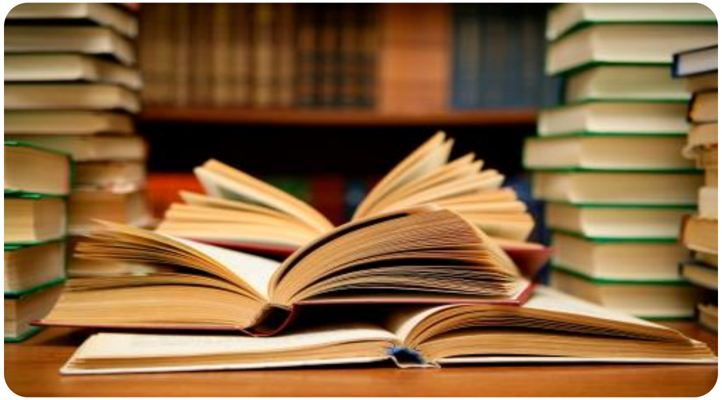
Kurz: **Datové struktury a algoritmy / Teoretická informatika**

Lektor: Doc. Ing. Radim Burget, Ph.D.
Autor: Doc. Ing. Radim Burget, Ph.D.

evropský sociální fond v ČR
EVROPSKÁ UNIE
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY
OP Vzdělávání pro konkurenceschopnost
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ
Vytvoření této videopřednášky bylo podpořeno projektem č. CZ.1.07/2.2.00/28.0098
Evropského sociálního fondu a státním rozpočtem České republiky.

Cíl přednášky

1. Stromová struktura dat
2. Stromy a příbuzné datové struktury
3. Implementace stromů
4. Procházení stromy
5. Vyvážené stromy

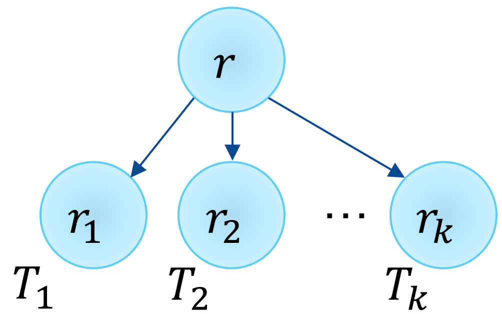


Stromová struktura dat

- Stromy a vyvážené stromy

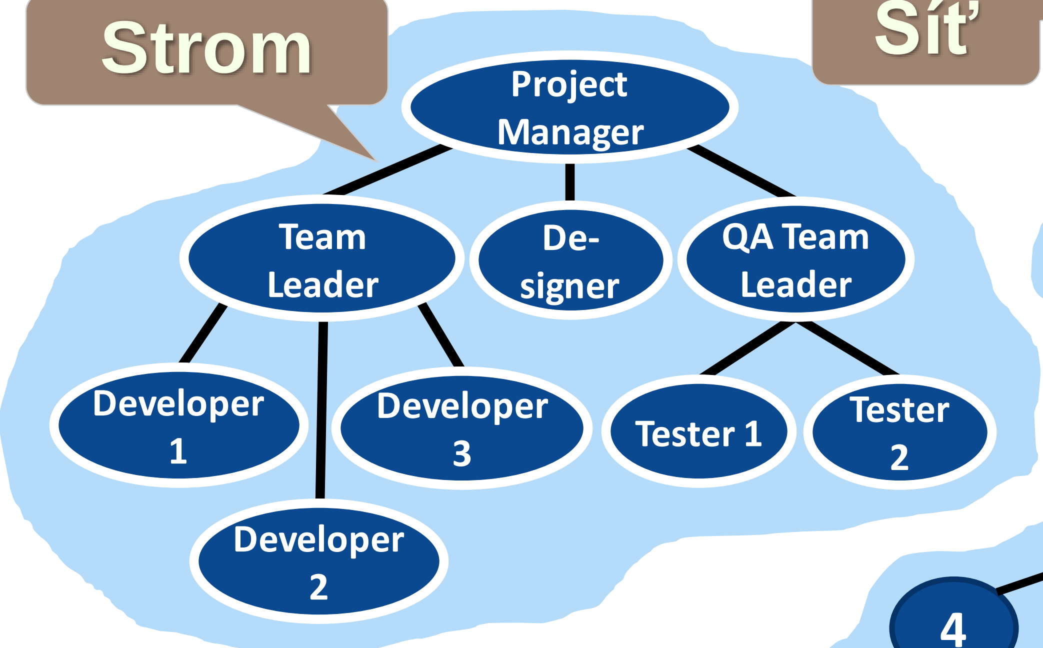
Stromová struktura dat

- Stromy – prostředek pro reprezentaci informace
- **Definice**:
  - Strom *T* je konečná množina nula nebo více prvků (uzlů), z nichž jeden je označen jako kořen *r* (root) a zbývající uzly jsou rozděleny do *k*≥0 disjunktních podmnožin $T_1, T_2, \ldots T_k$, které jsou také stromy a jejichž kořeny $r_1, r_2, \ldots, r_k$ jsou následníky kořene r

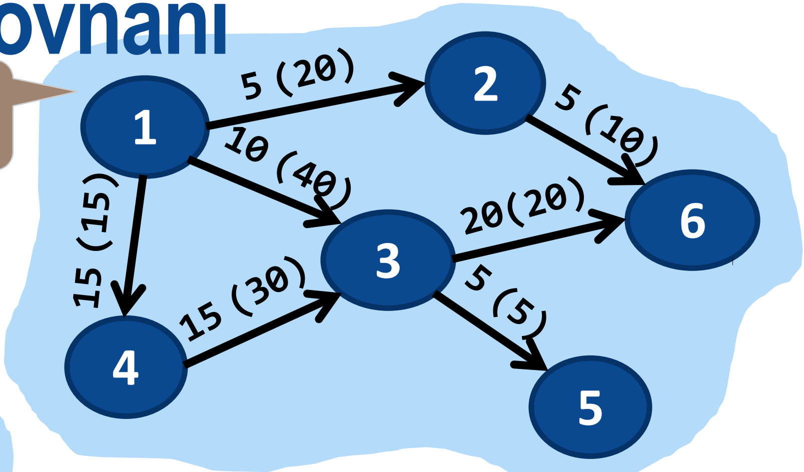


Stromová struktura dat - srovnání

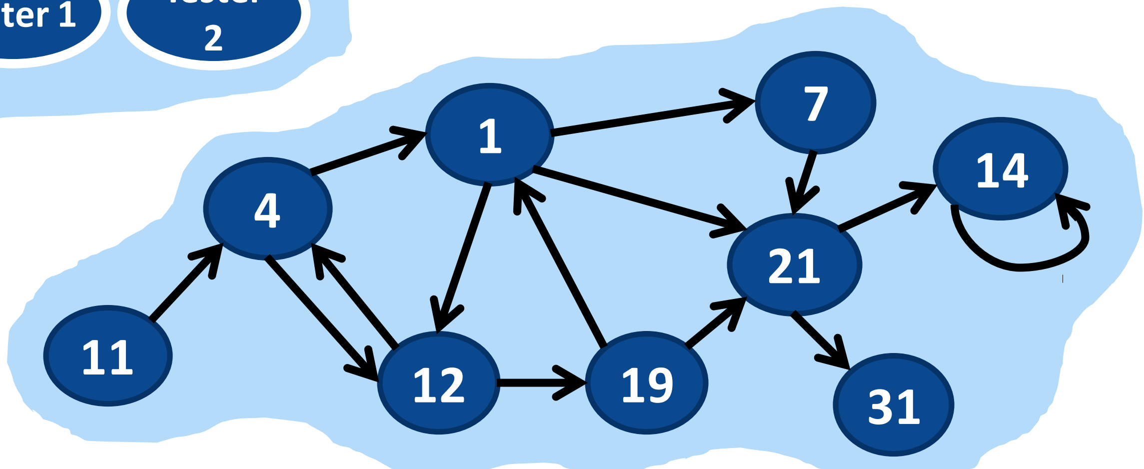
Strom



Síť



Graf



Stromy – příklady použití

- Jedna z nejčastějších datových struktur
- jeden z možných způsobů indexování klíčů v databázích (systémech řízení báze dat)
- reprezentace znalostí, stavového prostoru v umělé inteligenci
- metody distribuce klíčů v kryptografii (broadcast encryption)
- jakékoli řazené struktury, množiny, atp.
- popis scény v oblasti zpracování a analýza obrazu, počítačová grafika
- vyhledávací stromy v databázových systémech
- rozhodovací stromy – expertní systémy
- organizace adresářů a souborů v souborovém systému OS,
- komprese dat (Hufmannovy kódovací stromy, fraktálová komprese)
- atd.

Stromová struktura dat

- Stromové struktury jsou speciálním případem graf
- Významné vlastnosti:
    - Neobsahují cykly
    - Pokud jsou vyvážené a seřazené =>
      velice efektivní pro reprezentaci množin, vyhledávání prvků a řazení prvků

Stromy – rozdělení

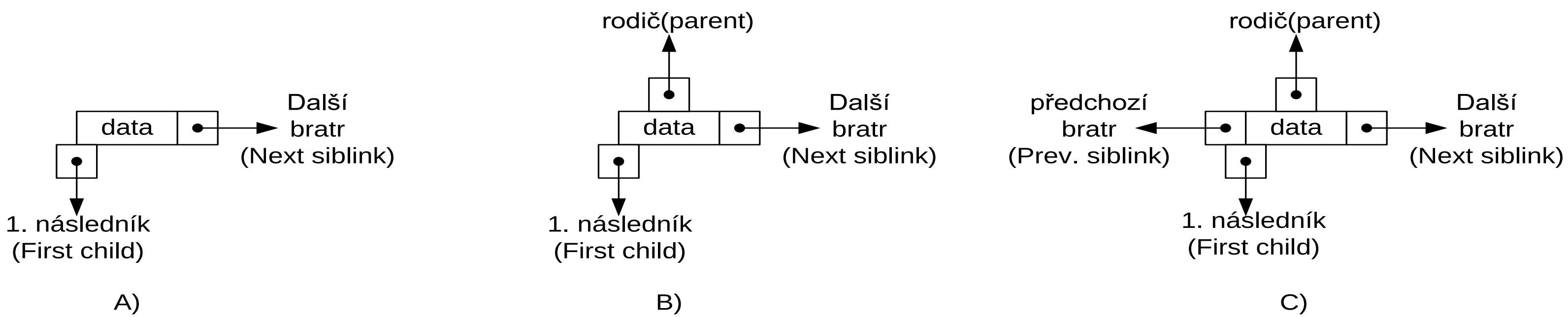
- Obecný strom
- N-ární strom

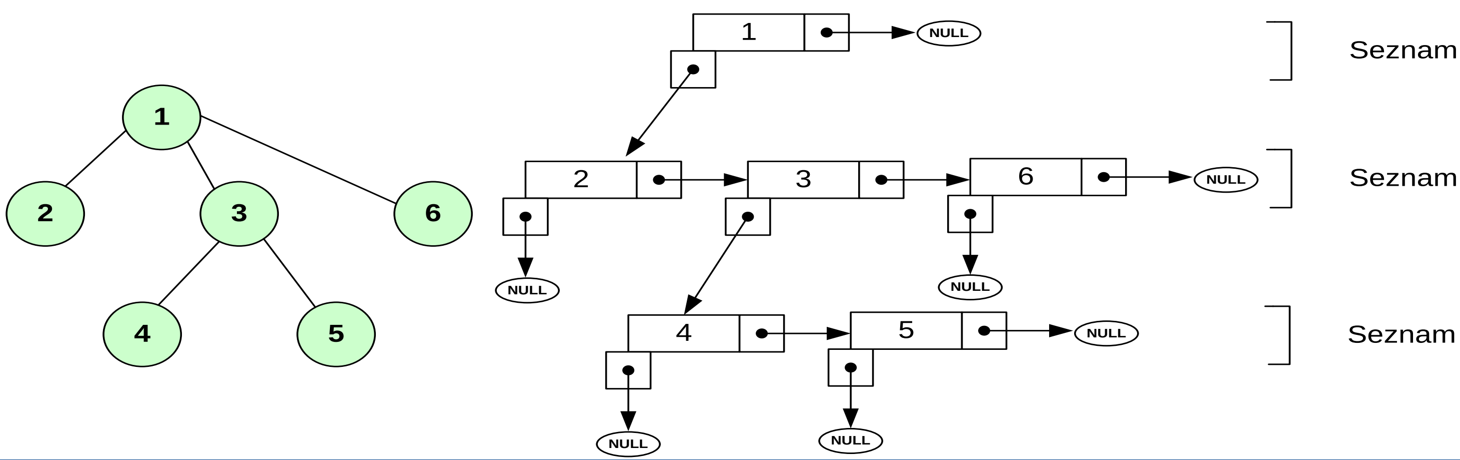
Vícecesté stromy (Multiway trees)

- Binární stromy
- Binární vyhledávací stromy
    - AVL stromy
    - Red-black stromy
- R stromy (prostorové vyhledávání)

Obecný strom

- Každý uzel libovolný počet potomků





Stromový ADT

- **getData( ):** vrací data na této pozici.
- **Obecné metody:**
  - integer **getSize( )**
  - boolean **isEmpty( )**
  - Iterator **iterator( )**
  - Iterable **positions( )**
- **Přístupové metody:**
  - Node **getRoot( )**
  - Node **getThisParent(p)**
  - Node **children(p)**

Dotazovací metody:

boolean **isInternal(p)**

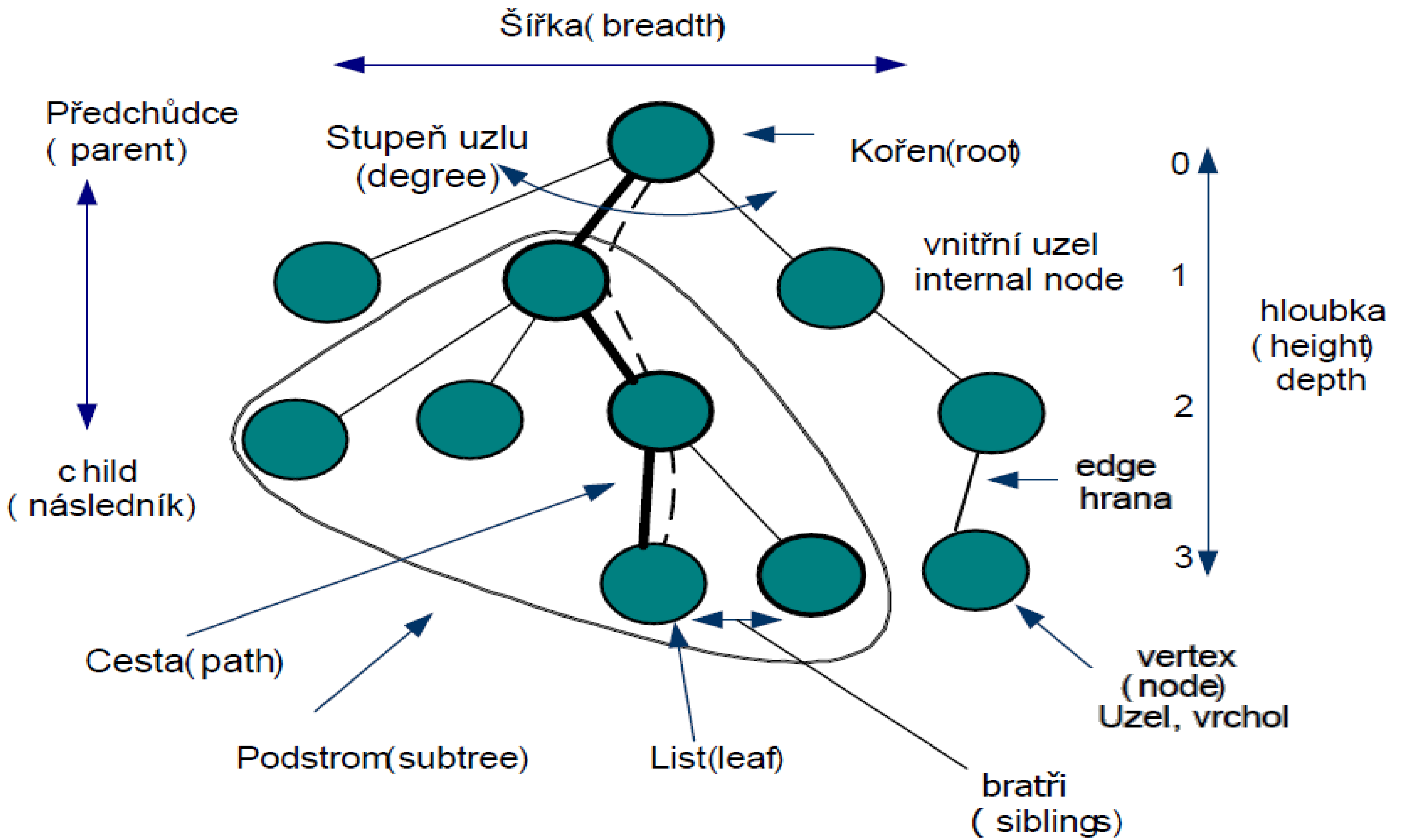
boolean **isLeaf(p)**

boolean **isRoot(p)**

Aktualizační metoda:

element **replace (p, o)**

Stromy a terminologie



Hloubka vrcholu

- Nechť ***v*** uzlem stromu **T**. Hloubka vrcholu ***v*** je počet předků ***v*** (s výjimkou ***v*** sebe sama)
  - Pokud ***v*** je prázdný (null), potom hloubka ***v*** je 0
  - Pokud ***v*** je kořen, potom hloubka ***v*** je 1
  - V opačném případě se hloubka ***v*** je hloubka rodiče ***v + 1***.

```
Algorithm depth(T, v):
if kořen je prázdný T then
        return 0
else if v je kořenem T then
        return 1
else
        return 1+depth(T, w), kde w je rodičem v ve
        stromu T
```

```
Algorithm depth(T, v):
depth ← 0
while v != null
   v = v.parent
   depth ← depth + 1
return depth
```

(Paměťově efektivnější)

- Časová složitost depth(T, v) je $\mathbf{O(d_v)}$,
  - kde $\mathbf{d_v}$ značí hloubku uzlu ***v*** ve stromě T.

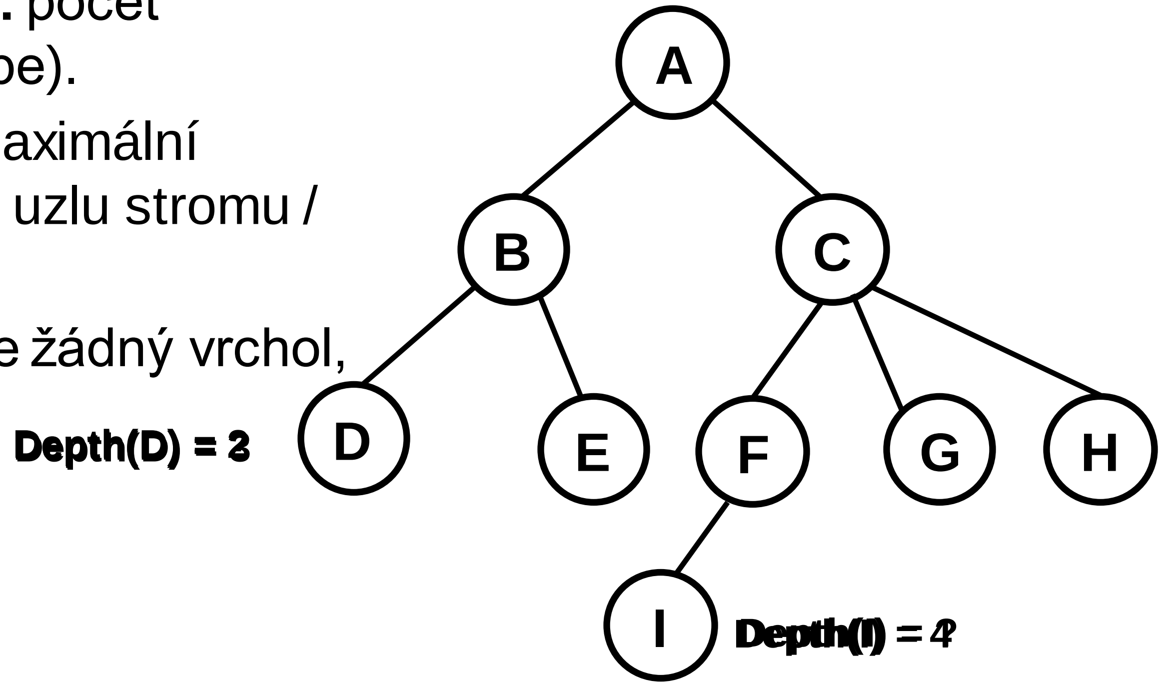
Velikost stromu – rekurzivní varianta

```
Algorithm size (Node root):
{
  if(root!=NULL)
          return(1 + size(root->child1)
            + size(root->child2) + … + size(root->childn));
  else return 0;
}
```

Doc. Ing. Radim Burget, Ph.D.

Výška stromu

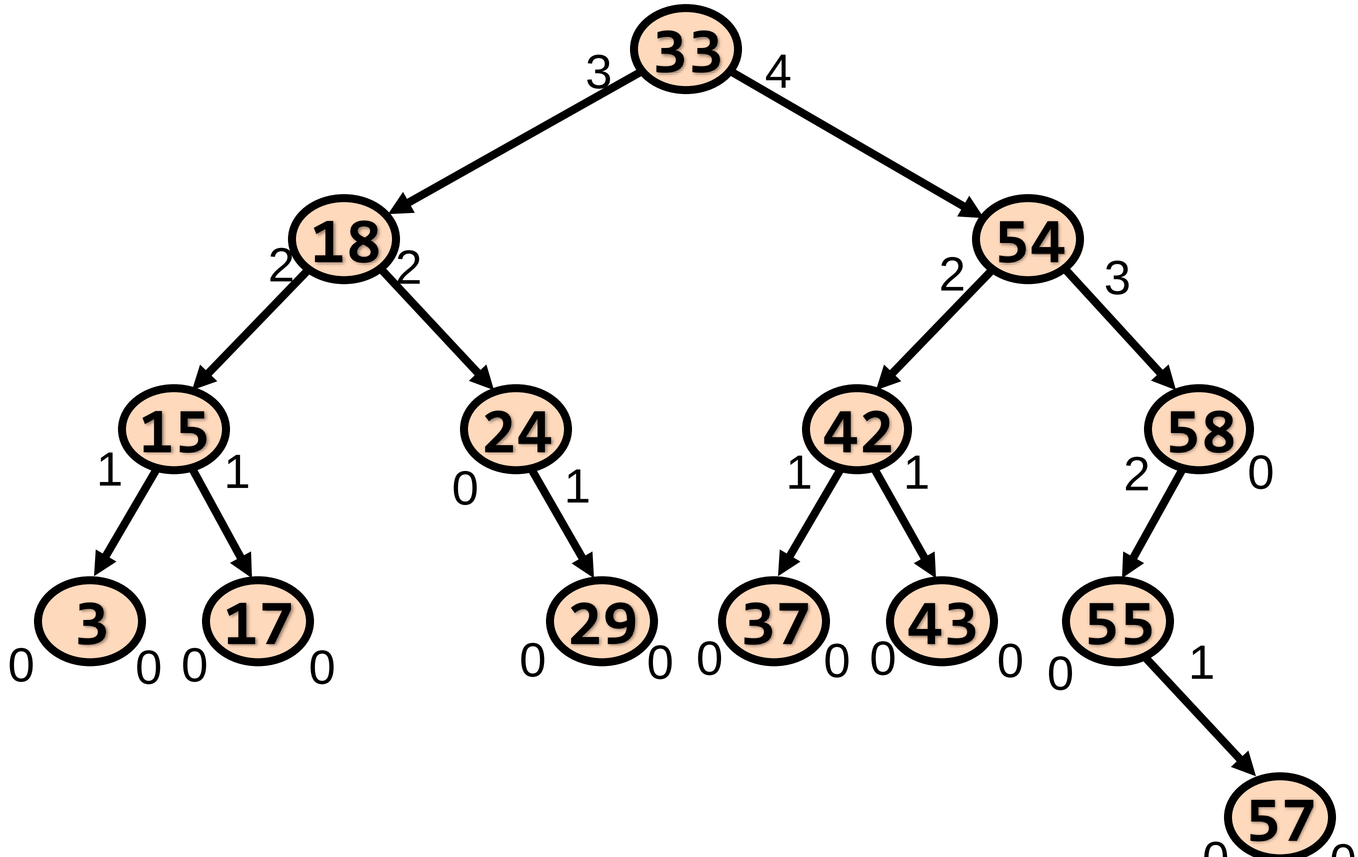
- **Hloubka vrcholu:** počet předků(včetně sebe).
- **Výška stromu:** maximální hloubka externího uzlu stromu / podstromu.
- Pokud neobsahuje žádný vrchol, tak je 0



Height = MAX[Depth(A), Depth(B), Depth(C), Depth(D), Depth(E), Depth(F), Depth(G), Depth(H), Depth(I)]

Height = MAX[1, 2, 2, 3, 3, 3, 3, 3, 4] = 4

Příklad: Určete výšku podstromů

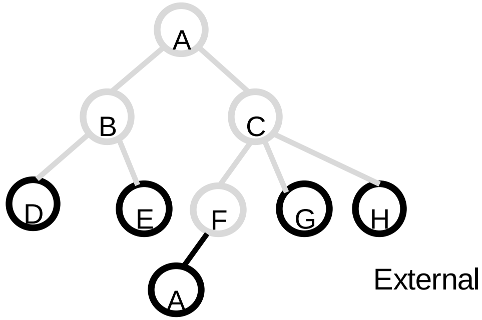


Výška stromu – neefektivní přístup

- Výška uzlu ***v*** ve stromě T může být určena jako:

Algoritmus height1(T):
$h \leftarrow 0$
for each vertex v in T **do**
  if v is an external node in T **then**
    $h \leftarrow \max(\mathrm{h}, \mathrm{depth}(\mathrm{T}, v))$
return h

- Algoritmus **height1** má časovou složitost O(**n²**)

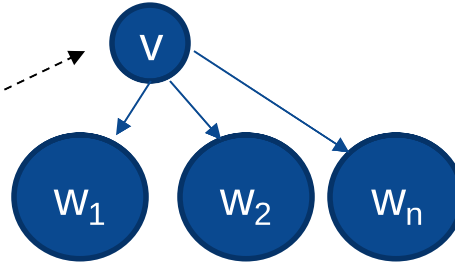


Doc. Ing. Radim Burget, Ph.D.

Výška stromu – efektivní přístup

- Výška uzlu ***v*** ve stromu T je rekurzivně definována:
  - Pokud ***v*** je listem, potom výška ***v*** je 1
  - V opačném případě ***v*** je jedna plus maximální výška potomka ***w***.

Algoritmus height2(T, *v*):
if *v* je externí uzel stromu T **then**
    return 0
else
    $h \leftarrow 0$
for each child *w* of *v* in T **do**
    $h \leftarrow \max(h, \text{height2}(T, w))$
return 1+*h*

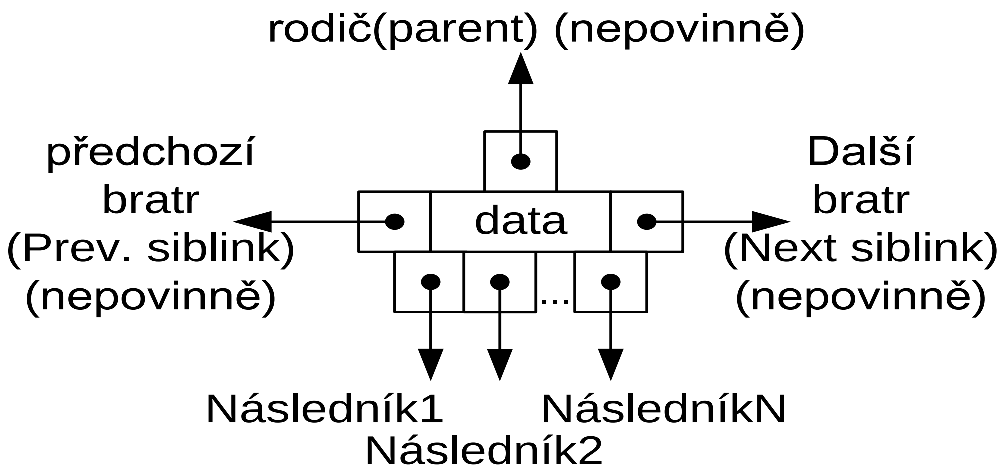


- algoritmus **height2** má časovou složitost O(**n**)

N-ární stromy

- maximálně N potomků
- N je konstanta

- Omezenější nežli obecný strom

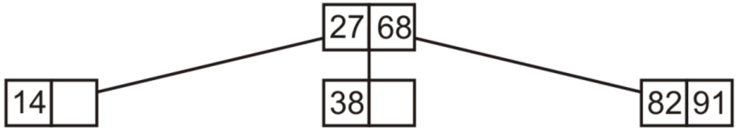


Ternární stromy

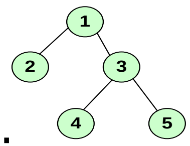
TCAM (Ternary Content Addressable Memory)

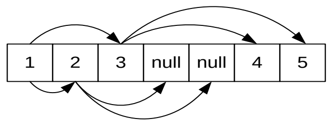
- Switche některých výrobců přepínaných síťových prvků akcelerují směrovací tabulky pomocí ternárních TCAM tabulek (až 10x zrychlení)

- Klasický princip je binární:

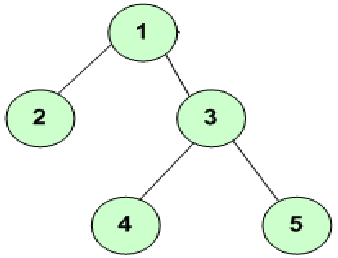


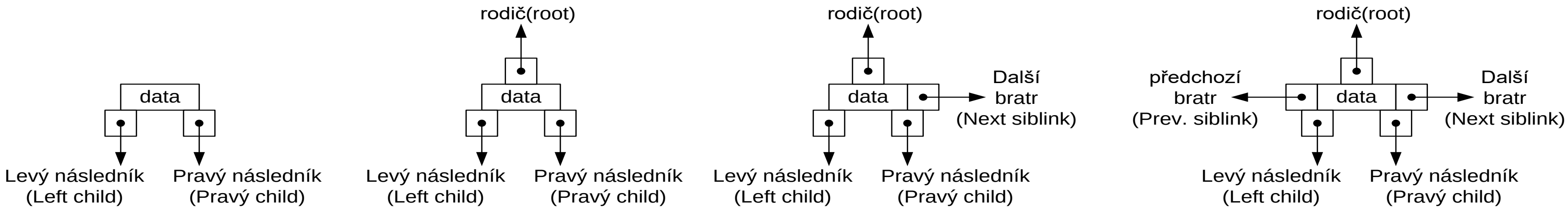
Binární stromy





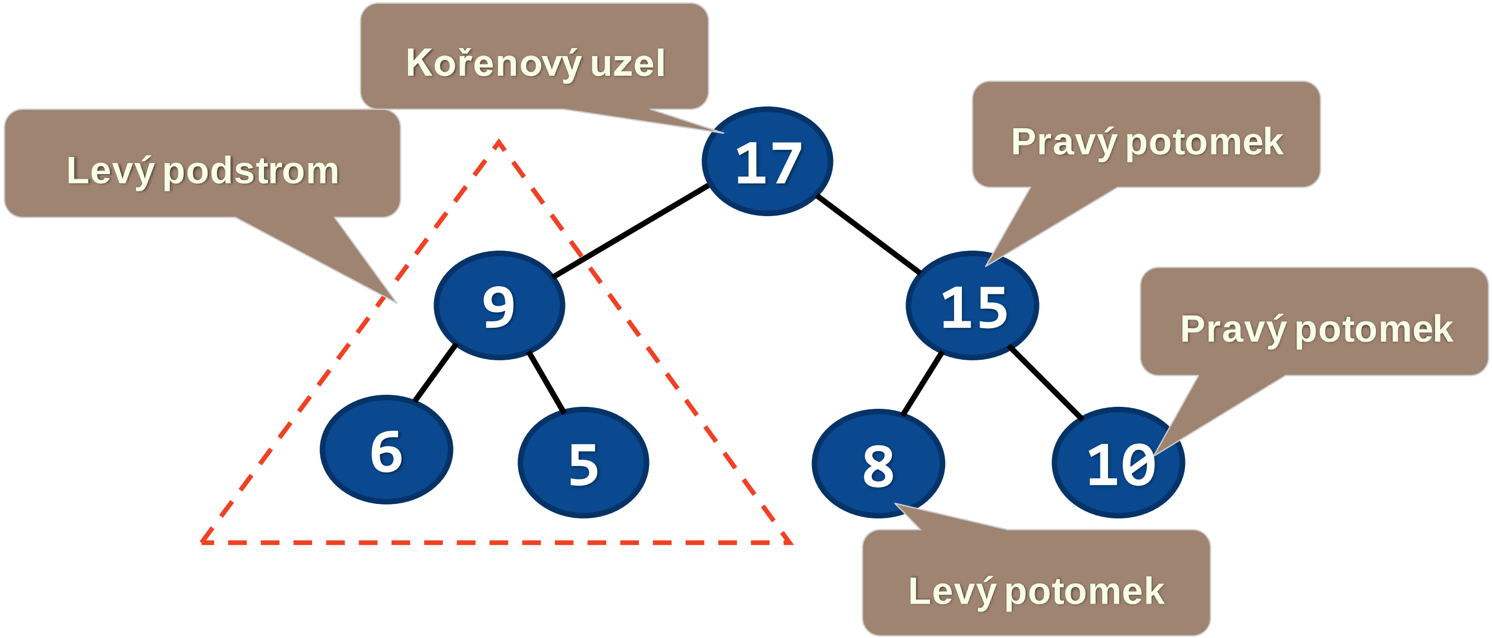
- N-ární strom, kde N = 2
- Většina stromových struktur použitých v informatice jsou binární stromy
- Prvky nejsou seřazeny
- Implementace má několik variant (záleží na způsobu použití)
  - Ovlivňuje schopnost se pohybovat v stromové struktuře (výkon)
  - Zvyšuje nároky na paměť





Binární stromy

- Binární stromy: speciální terminologie
  - Každý uzel má nejvýše 2 potomky

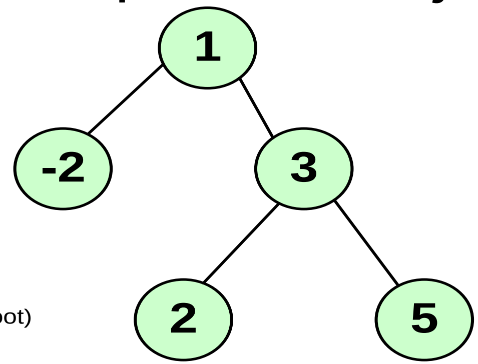


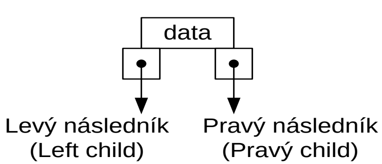
Binární vyhledávací stromy

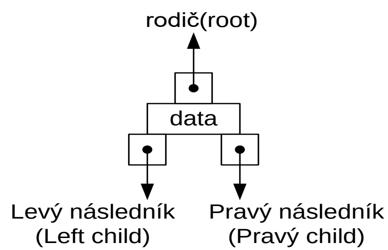
- Binární vyhledávací stromy jsou řazené
  - Pro každý uzel x ve stromě platí:
    - Všechny uzly v levém podstromu uzlu x jsou $\leq x$
    - Všechny uzly v pravém podstromu uzlu x jsou $> x$
- Binární stromy mohou být vyvážené
  - Vyvážené stromy mají hloubku $\sim \log_2(x)$
  - Vyvážené stromy mají pro každý uzel téměř stejný počet uzlů v jejich podstromech

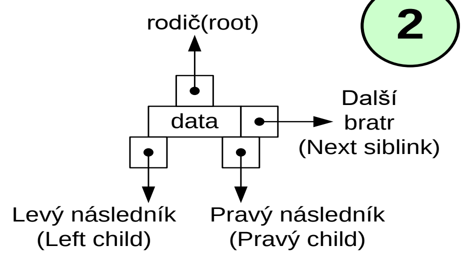
Binární vyhledávací stromy

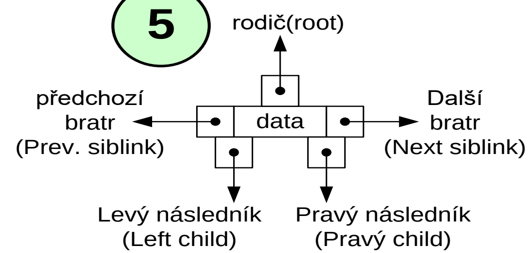
- (**Definice**) Binární vyhledávací stromy jsou stromy, kde musí pro každý uzel platit, že hodnota všech potomků z levého podstromu je menší než hodnota rodiče, a hodnota všech potomků z pravého podstromu je větší než hodnota rodiče.
- Tj. Prvky jsou seřazeny
- Vychází z binárního stromu





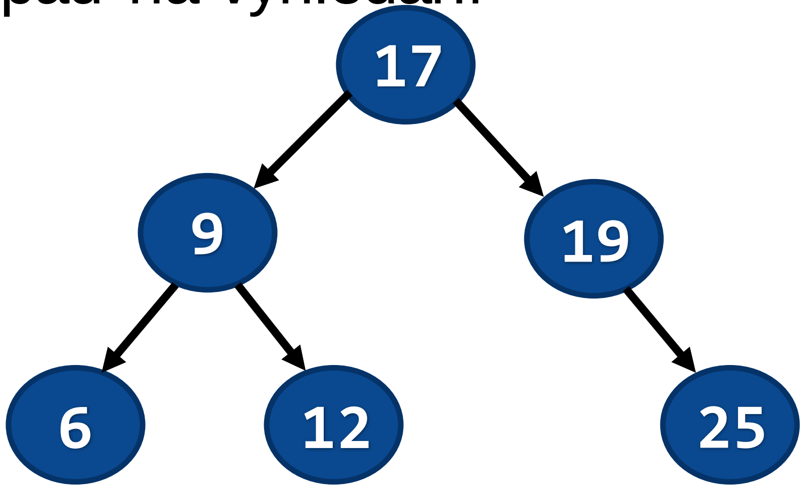




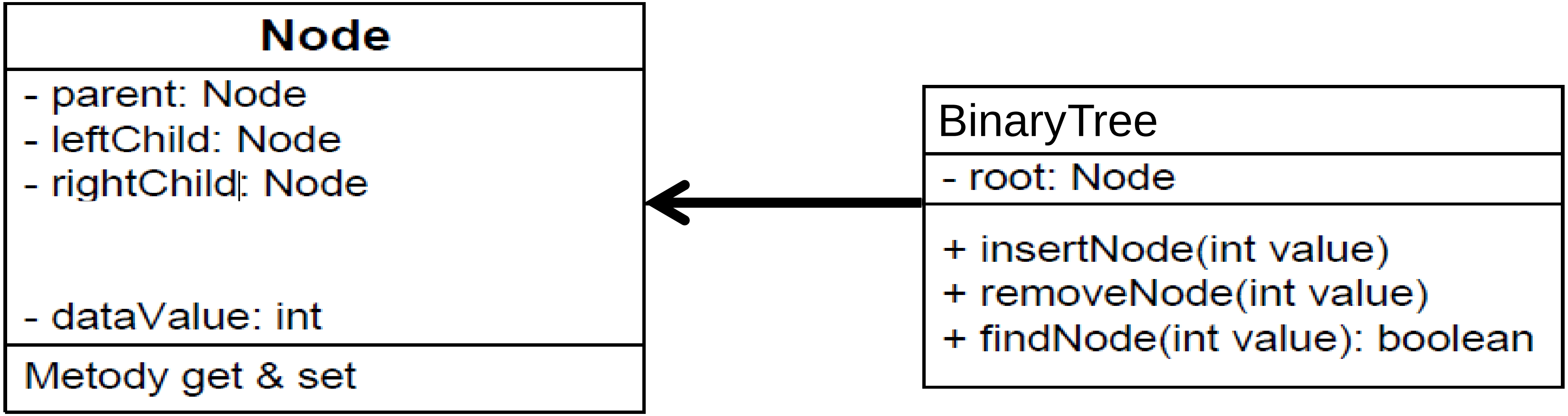


Binární vyhledávací stromy

- Příklad vyváženého binárního vyhledávacího stromu
- Příklad: Vložte do prázdného stromu prvky 17, 9, 6, 12, 19, 25
- Pozn: Co se stane pokud vložím prvky v pořadí 6, 9, 12, 17, 19, 25 ? Jaký to bude mít dopad na vyhledání libovolného prvku (výkon)?



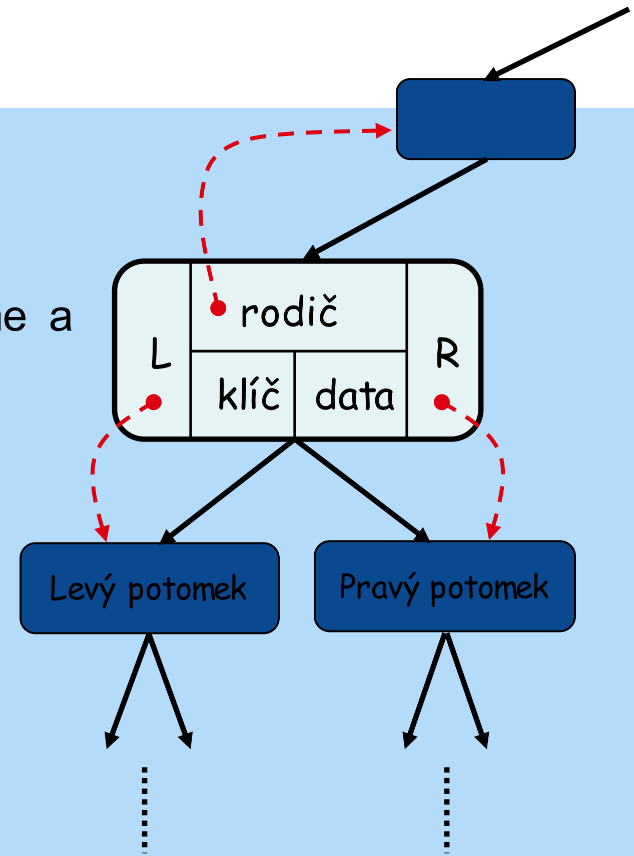
Binární vyhledávací stromy - implementace



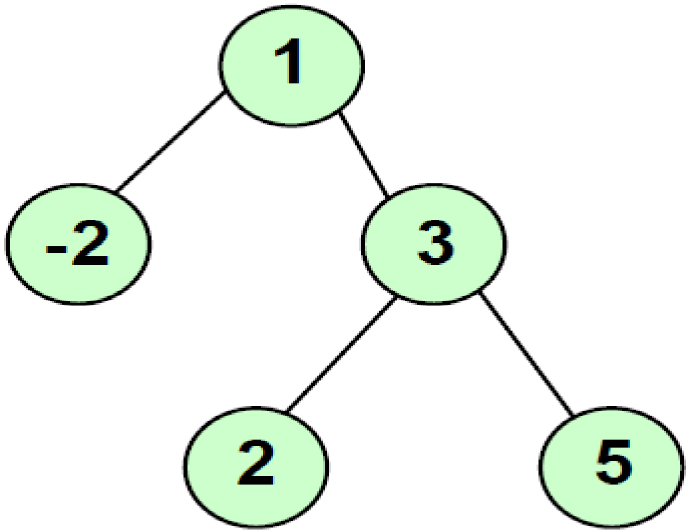
Binární vyhledávací stromy

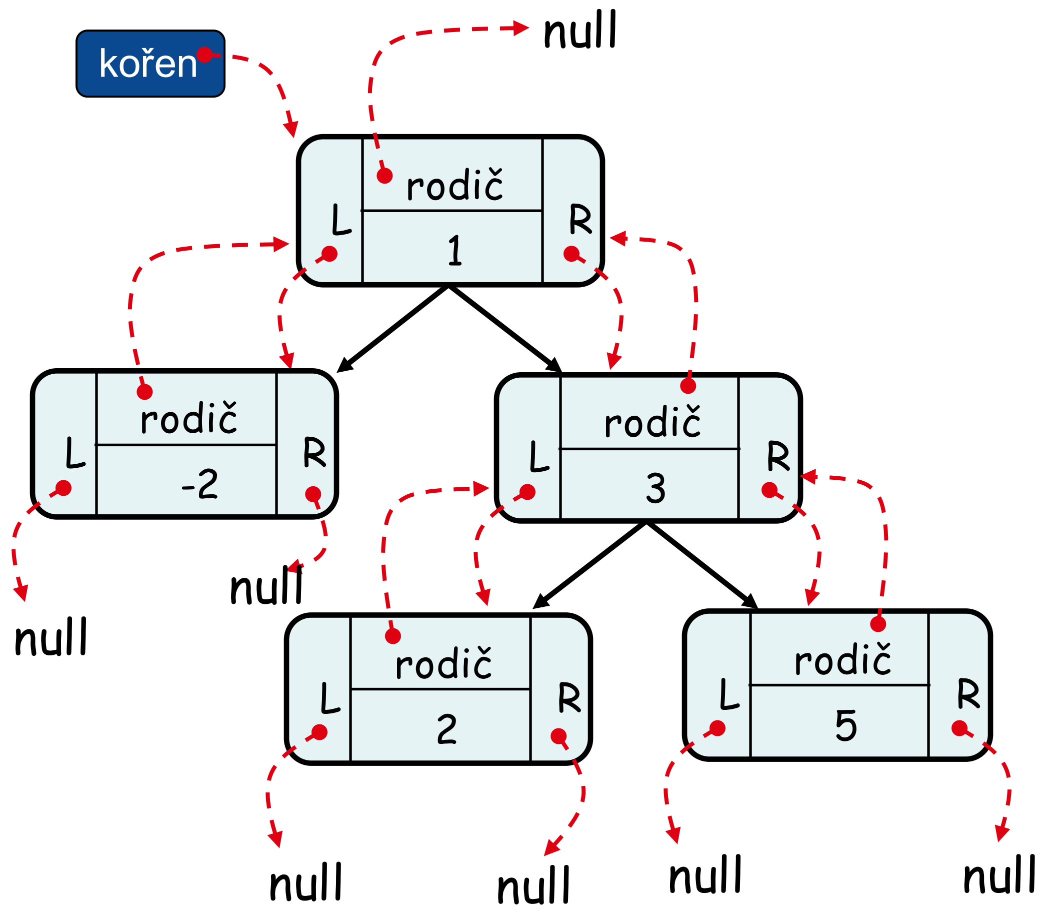
- Reprezentace stromových struktur:
  - Provázaná datová struktura
  - Rozšiřujeme často o klíč + hodnota (pro jednoduchost toto rozdělení zanedbáváme a budeme předpokládat, že klíč = data)
- Příklad - implementujte:
  - Key atribut
  - Data
  - Left: ukazatel na levý podstrom
  - Right: ukazatel na pravý podstrom
  - p: ukazatel na rodiče

    (p [root [T]] = NIL)



Binární vyhledávací stromy – implementace





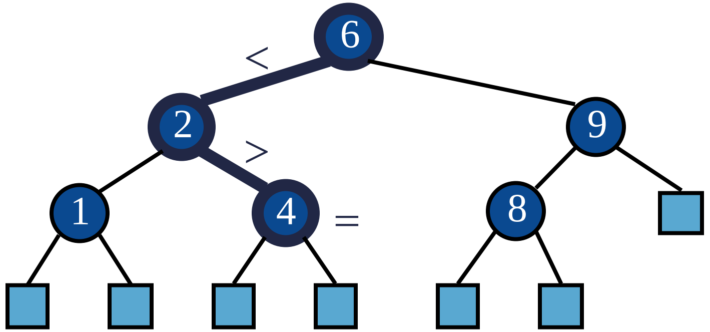
Vyhledávání v BVS

- Pro vyhledání hodnoty ***k***, prohledáváme BVS odspodu nahoru, kde začínáme v kořenu.
- Další navštívený uzel závisí na hodnotě ***k*** a hodonty aktuálního uzlu.
- Pokud soáhneme listu, hodnota neexistuje.

- **Příklad:** contains(4):

Rekurzivní varianta:

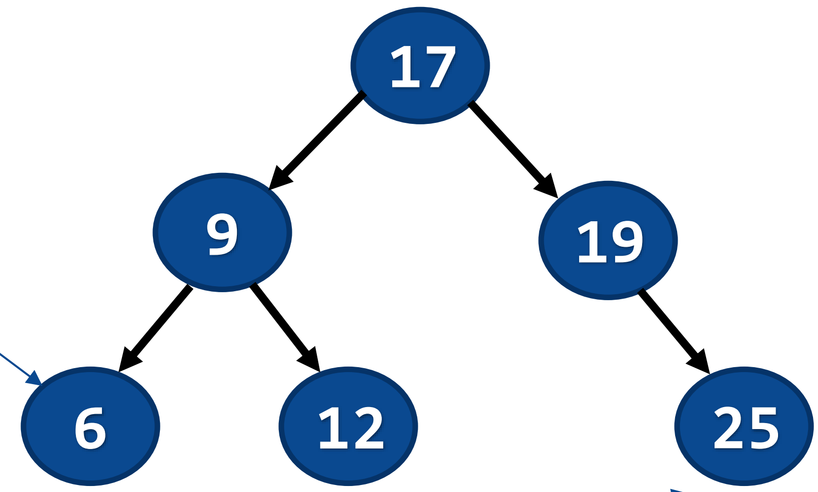
Algoritmus ***TreeSearch***(***k***, ***v***)
  if *T.isExternal* (*v*)
    return *v*
  if *k* < *key*(*v*)
    return *TreeSearch*(*k*, *T.left*(*v*))
  else if *k* = *key*(*v*)
    return *v*
  else { *k* > *key*(*v*) }
    return *TreeSearch*(*k*, *T.right*(*v*))



Nalezení minima / maxima (rekurzivní varianta)

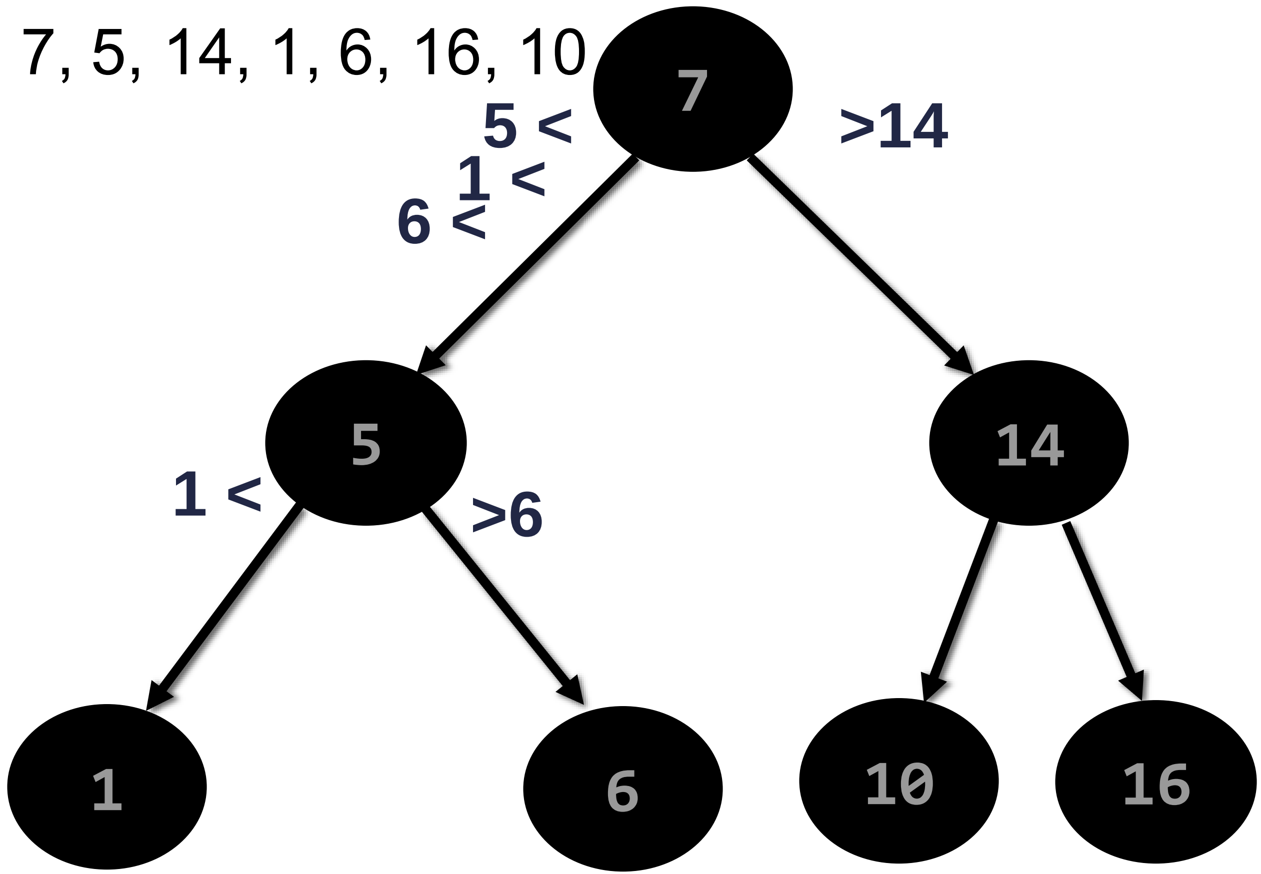
```
NalezniMinimum (T)
{
    if (t_left is not empty)
        return NalezniMinimum(levý podstrom);
    else
        return key(t);
}
```

```
NalezniMaximum (T)
{
    if (t_right is not empty)
        return NalezniMaximum(pravý podstrom);
    else
        return key(t);
}
```



Vložení prvku do prázdného BVS

- Vložte: 7, 5, 14, 1, 6, 16, 10



Algoritmus vložení: rekurzivní varianta

```
Insert (tree, new_item)
  if (tree je prázdný)
        vlož na kořen;
  else if  (shoduje se s aktuálním uzlem)
        nedělej nic; (duplicitní klíč)
  else if (nový klíč je menší než aktuální kořen)
        volej rekurzivně insert na levý pod-strom;
  else
        volej rekurzivně insert na pravý pod-strom
```

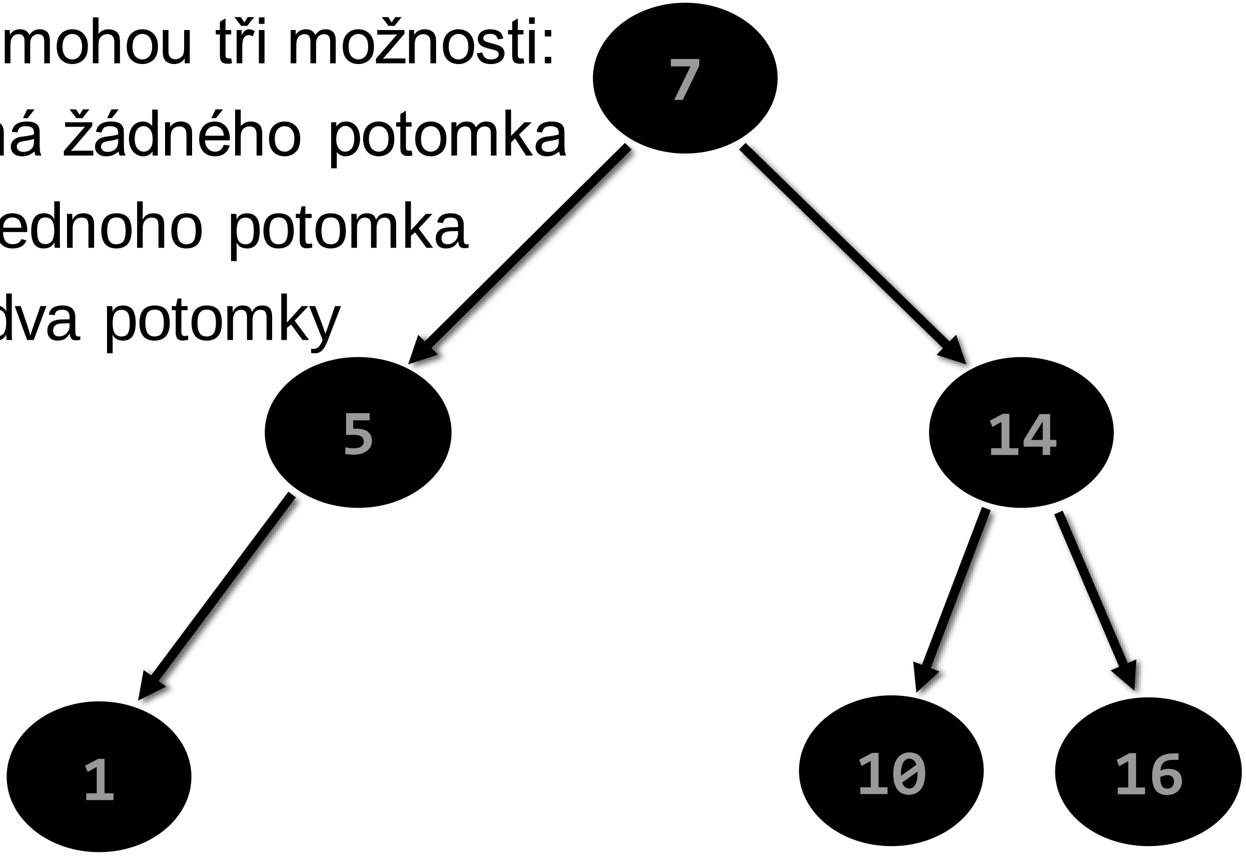
Odstranění prvku

- Nastat mohou tři možnosti:

A) Nemá žádného potomka

B) Má jednoho potomka

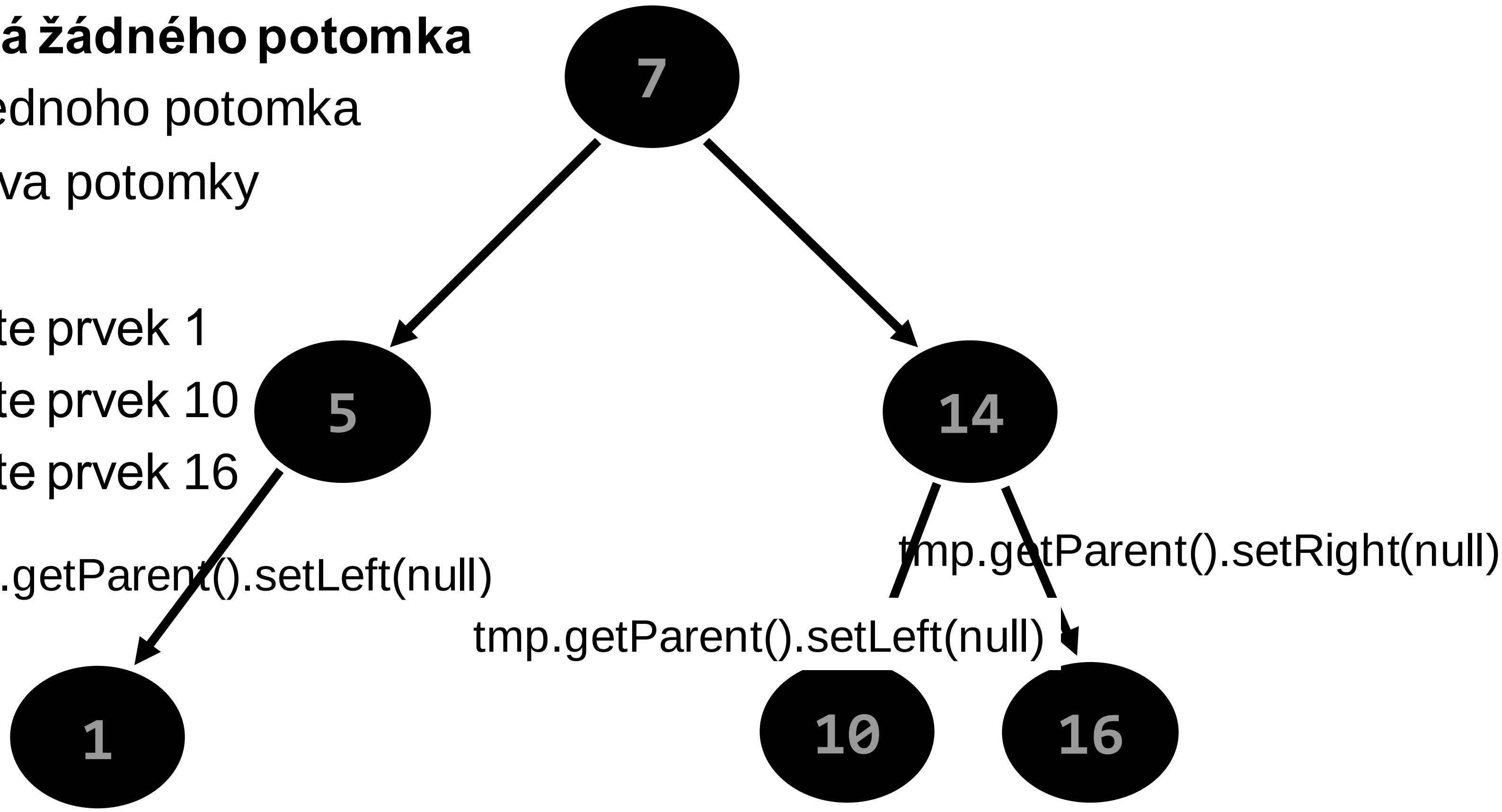
C) Má dva potomky



A) Odstranění prvku – nemá žádného potomka?

A) **Nemá žádného potomka**
B) Má jednoho potomka
C) Má dva potomky

- Odstraňte prvek 1
- Odstraňte prvek 10
- Odstraňte prvek 16



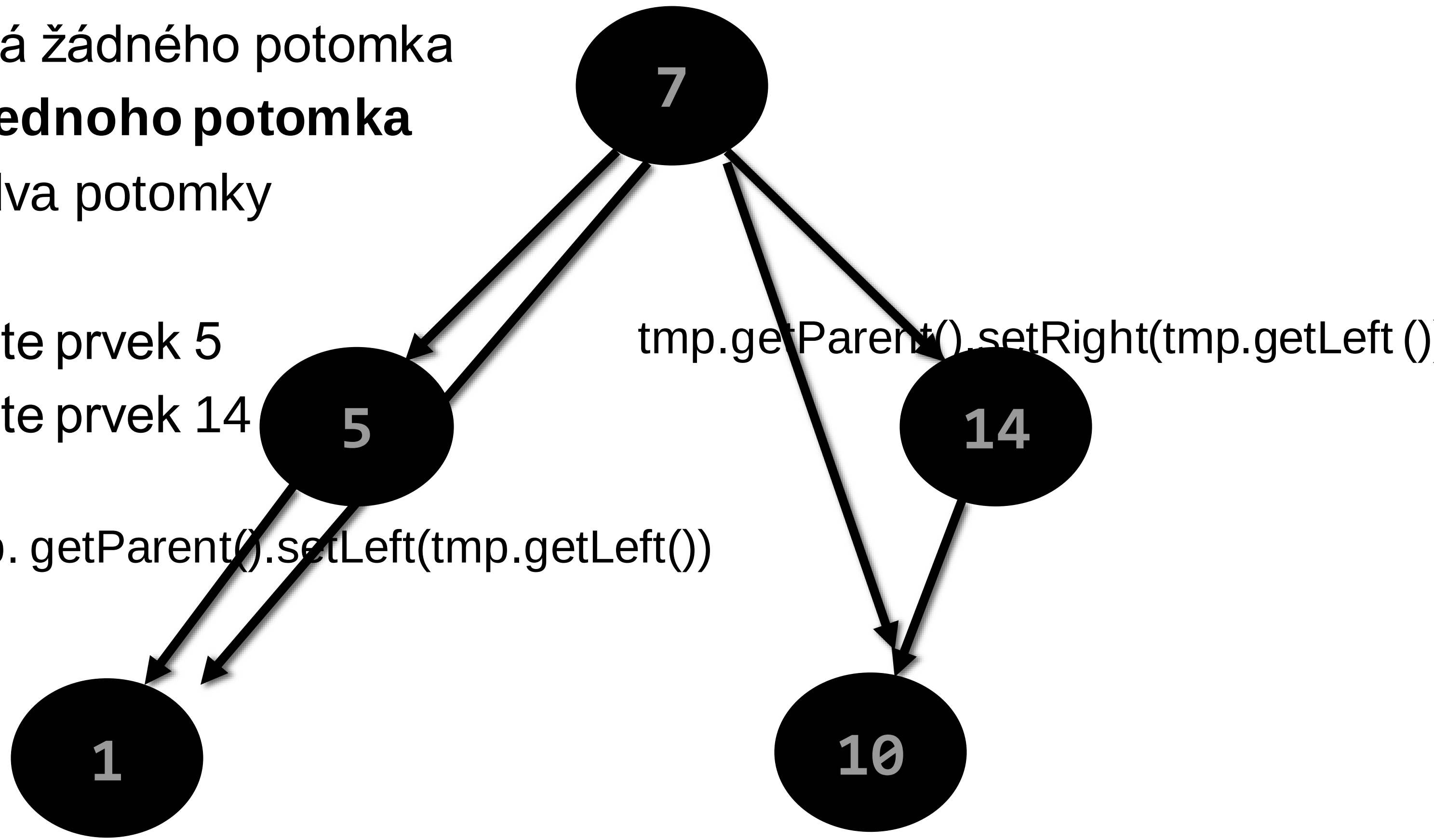
B) Odstranění prvku – má jednoho potomka

A) Nemá žádného potomka

B) **Má jednoho potomka**

C) Má dva potomky

- Odstraňte prvek 5
- Odstraňte prvek 14



C) Odstranění prvku – má oba potomky

A) Nemá žádného potomka

B) Má jednoho potomka

C) **Má dva potomky**

- Její součástí jsou kroky A anebo B – tzn. podařilo se vám porozumět krokům A) a B)?
- Existují 2 varianty – levá a pravá

Doc. Ing. Radim Burget, Ph.D.

C) Odstranění prvku – má oba potomky

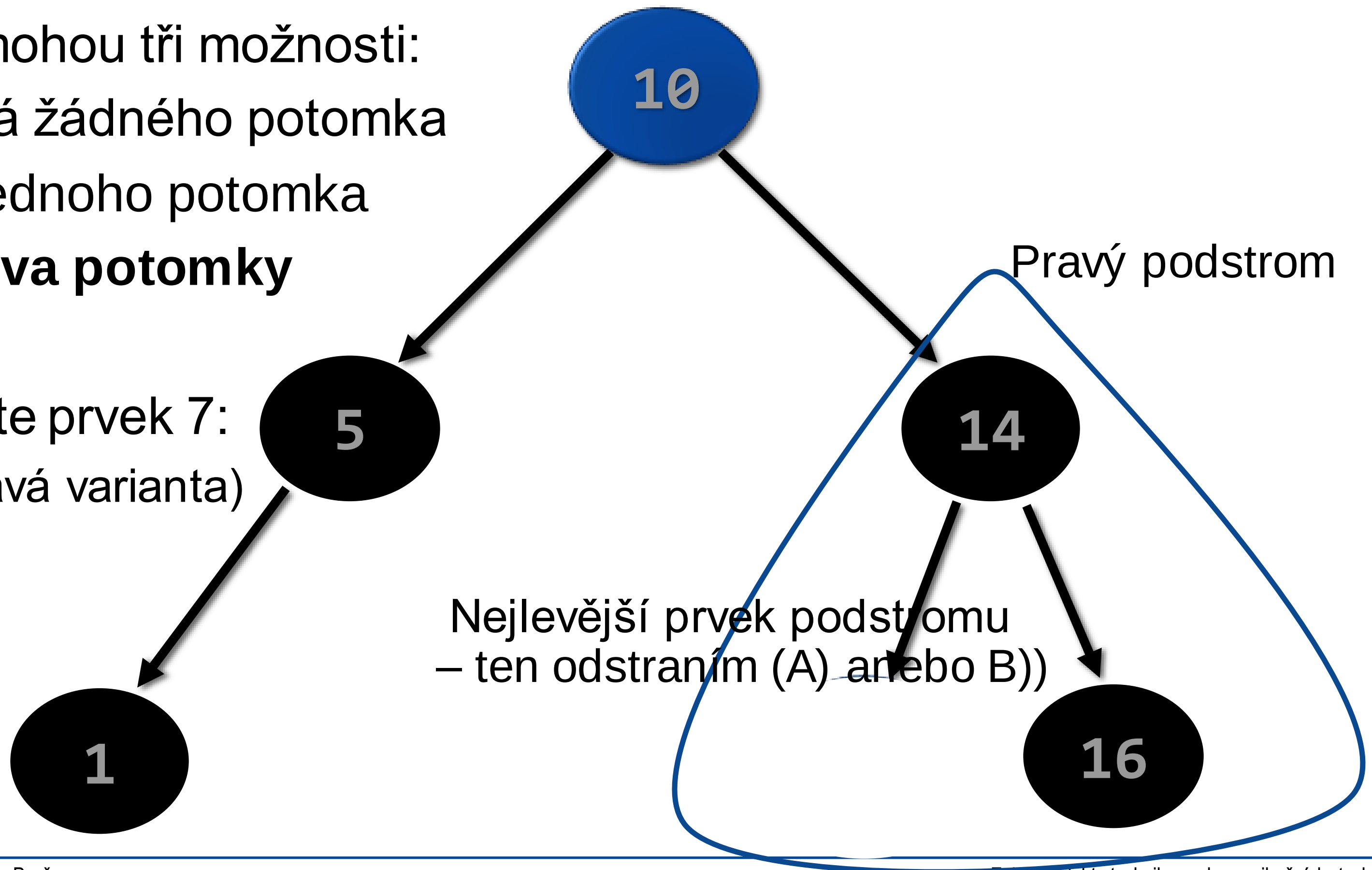
- Nastat mohou tři možnosti:

A) Nemá žádného potomka

B) Má jednoho potomka

C) **Má dva potomky**

- Odstraňte prvek 7:
- (např. pravá varianta)



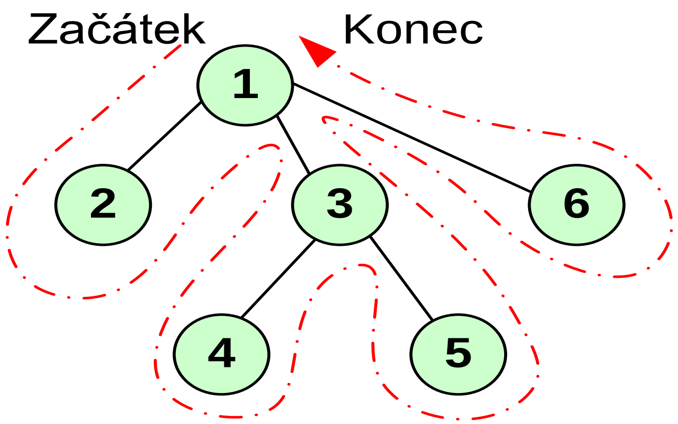
Průchody stromy

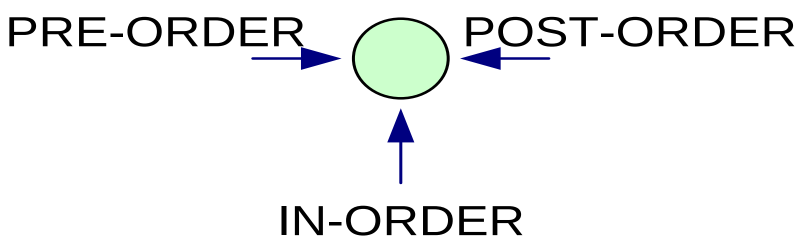
Doc. Ing. Radim Burget, Ph.D.

Algoritmy pro procházení stromů

- Pre-order: 1 2 3 4 5 6
- In-order: 2 1 4 3 5 6
- Post-order: 2 4 5 3 6 1

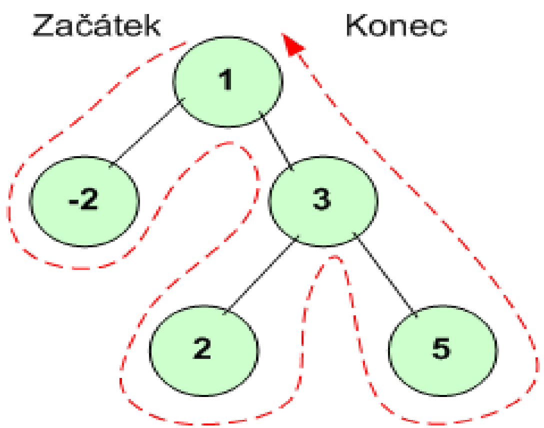
+ revresní varianty (zleva doprava)

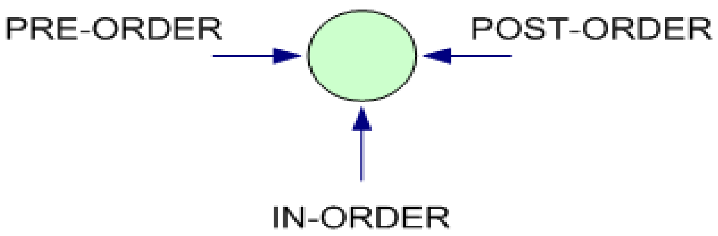




Algoritmy pro procházení stromů

- Jeli strom binární vyhledávací -> metoda in-order vrací prvky ve správném pořadí: -2 1 2 3 5





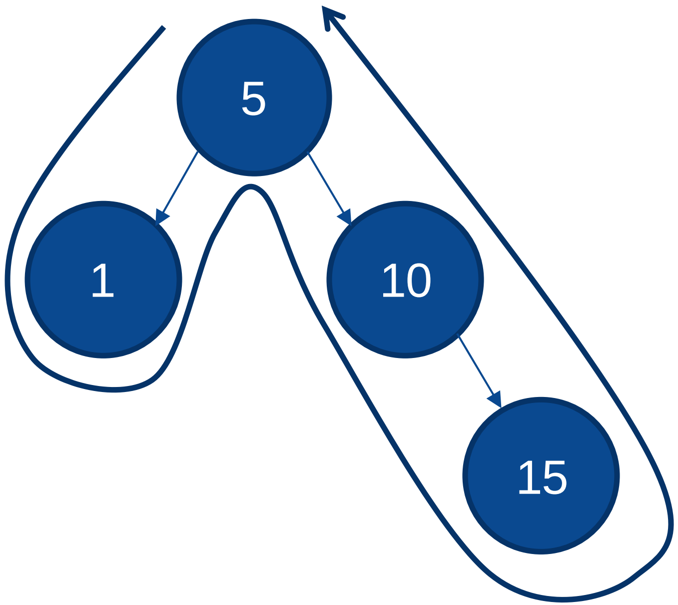
```java
public class Spustitelna {
    public static void main(String[] args) {
        MujStrom s = new MujStrom();
        s.pridej(5);
        s.pridej(10);
        s.pridej(1);
        s.pridej(15);

        s.vypis();
    }
}
```

```java
public void vypis() {
    vypis(koren);
}

private void vypis(Uzel u) {
    if(u == null) {
        return;
    }
    vypis(u.getLevy());
    System.out.println(u.getData());
    vypis(u.getPravy());
}
```

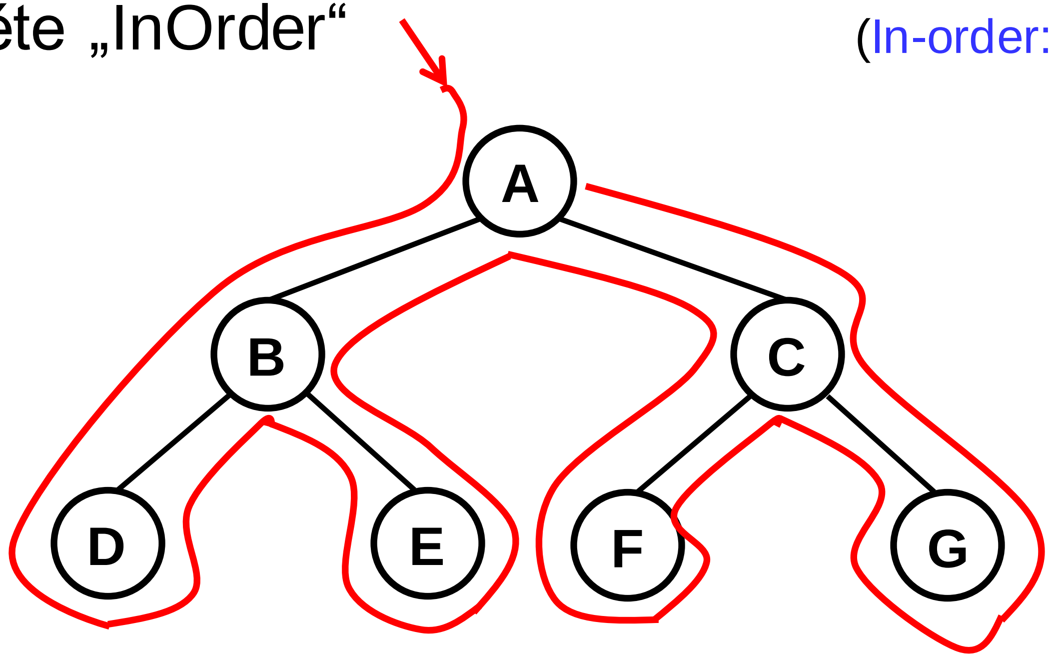
IN-ORDER



Příklad: Průchod stromem In-order

- Implementujte binární vyhledávací strom
- Projděte „InOrder“

(In-order: Levý ⇨ Uzel ⇨ Pravý)

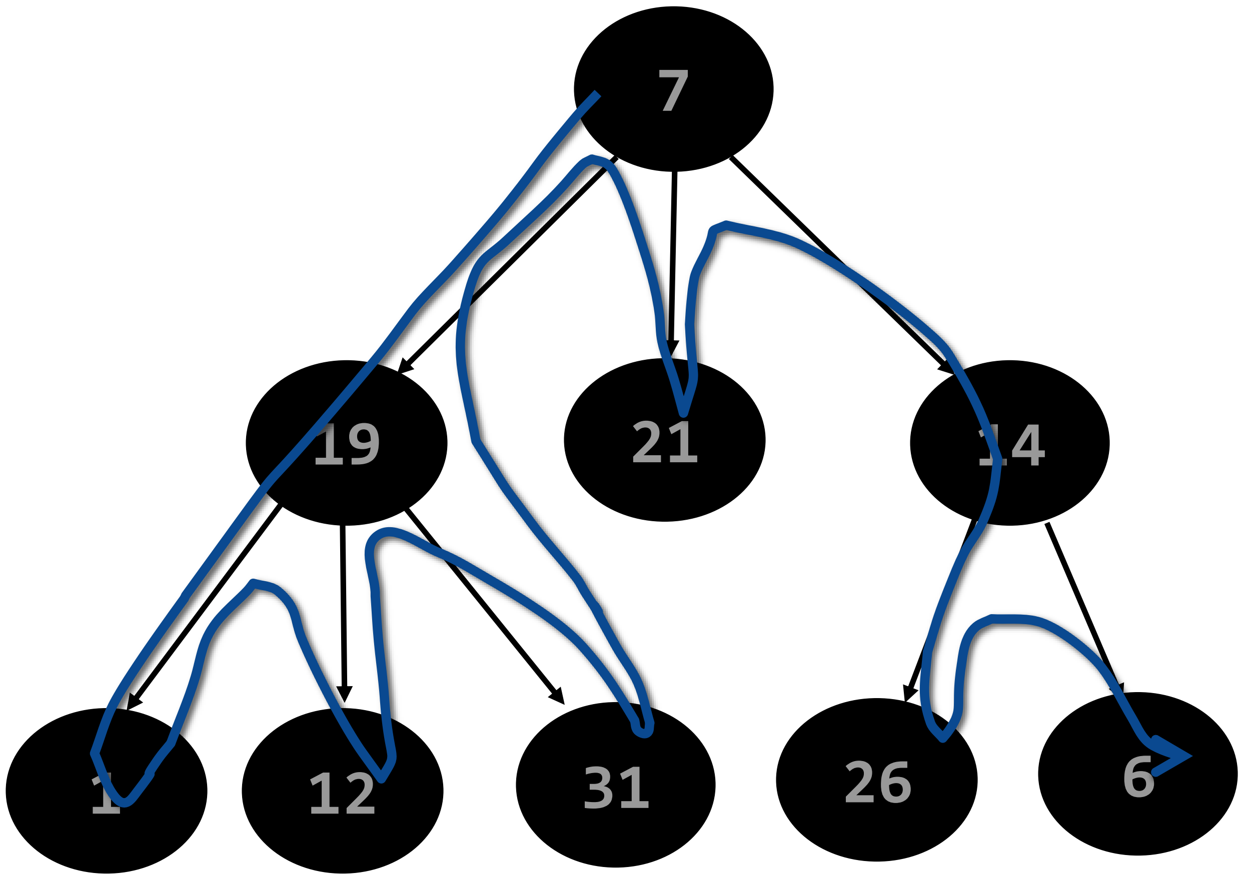


D B E A F C G

Algoritmy pro procházení stromů

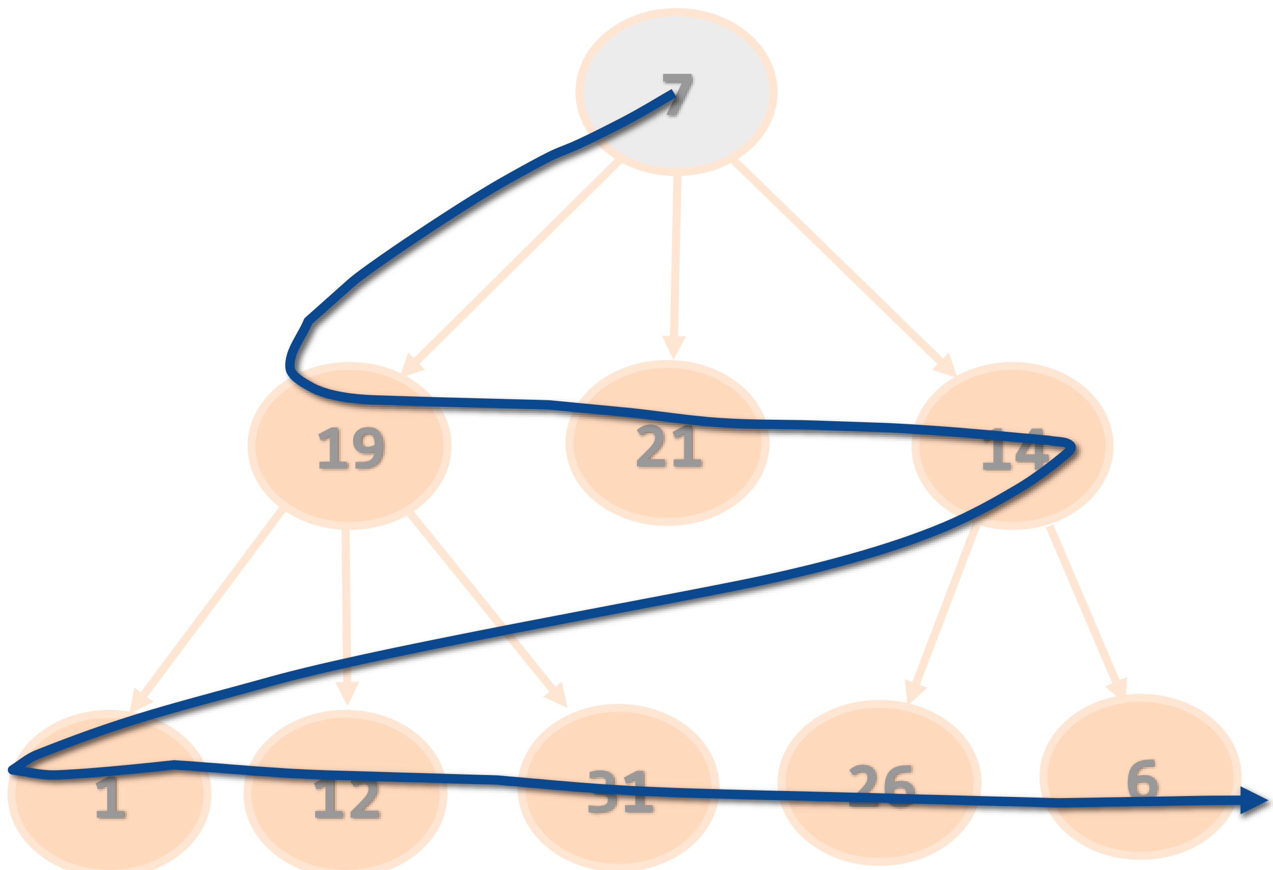
- Stromy jsou speciálním případem grafu a lze tedy na nich aplikovat veškeré algoritmy pro procházení/vyhledávání v grafech:
  - Prohledávání do hloubky – DFS (Depth-First Search)
    - Nejdříve se projde celá větev
  - Prohledávání do šířky – BFS (Breadth-First Search)
    - Nejdříve se prochází všichni potomci zkoumaného uzlu
  - Algoritmům procházení grafem se budeme věnovat detailněji později

Prohledávání do hloubky – DFS (Úvod)



Doc. Ing. Radim Burget, Ph.D.

Prohledávání do šířky – BFS (Úvod)



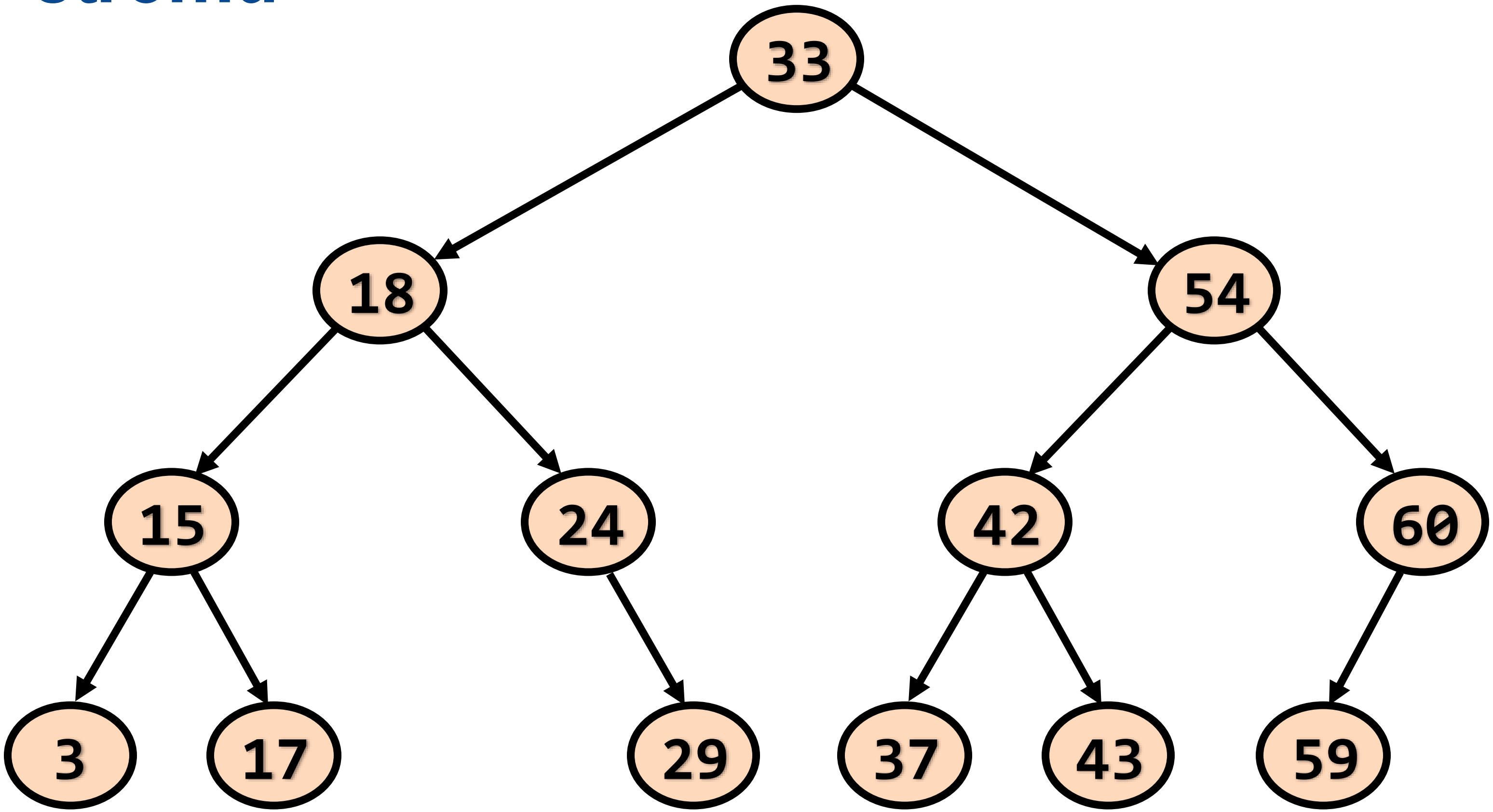
Vyvážené vyhledávací stromy

- AVL stromy, Red-Black stromy, (také B-stromy, B+-stromy, 2-3 stromy, 2-3-4 stromy, AA-stromy, (a, b) stromy, B-stromy, …)

Vyvážený binární vyhledávací strom

- Vyhledávací binární strom
  - Pro každý uzel *x* ve stromě platí:
    - Všechny uzly v levém podstromu uzlu x jsou $\leq x$
    - Všechny uzly v pravém podstromu uzlu x jsou $> x$
- Vyvážený strom
  - Rozdíl hloubky levého a pravého podstromu každého uzlu je vždy nula nebo jedna.
- Vyvážený binární vyhledávací strom
  - má výšku $\log_2(n)$ kde n je počet uzlů
  - Vyhledávání ve stromě tedy znamená přibližně $\log_2(n)$ operací

Příklad vyváženého vyhledávacího binárního stromu



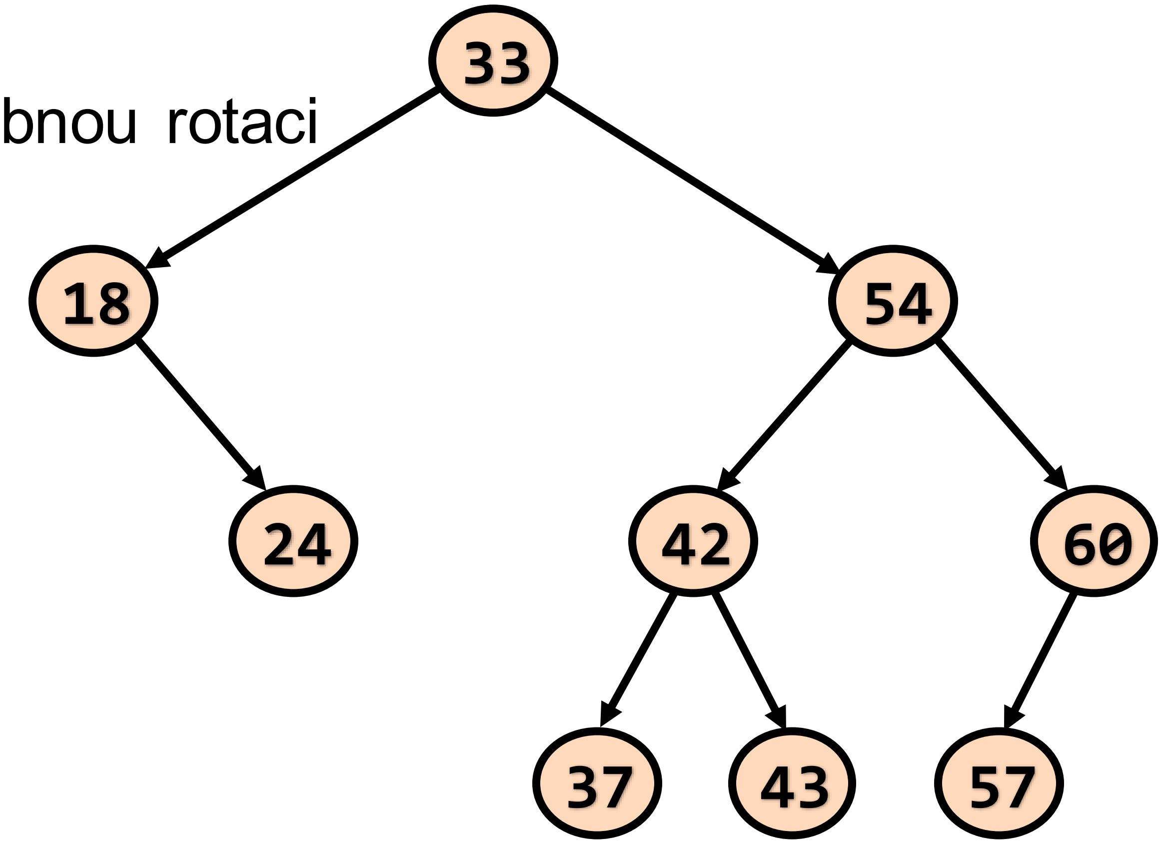
Vyvážený binární vyhledávací strom

- Vyvažování po vkládání a mazání je komplexní operace

- Příklady implementace vyvážených binárních vyhledávacích stromů:
  - AVL stromy – lépe vyvážené, hrozí problém mnohonásobné rotace
    - Efektivnější vyhledávání, méně efektivní vkládání
  - Red-black stromy – zhruba vyvážené, řeší problém mnohonásobné rotace
    - Efektivnější vkládání, méně efektivní vyhledávání

Mnohonásobná rotace

- Vložte hodnotu 58
- Způsobí mnohonásobnou rotaci



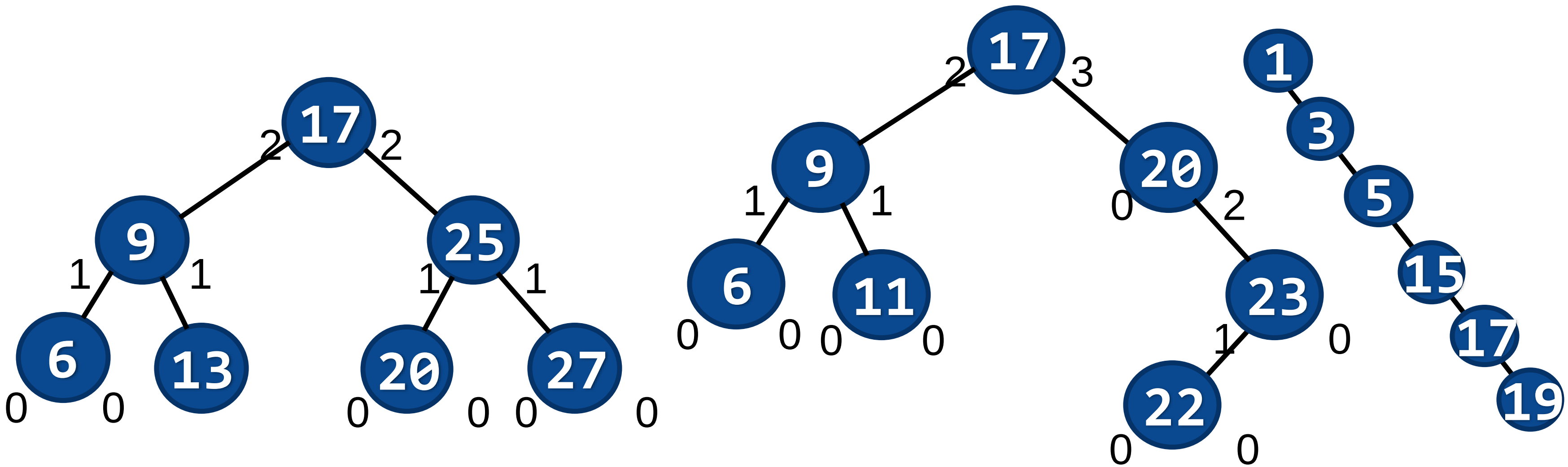
AVL stromy a vkládání

- Vkládání má dvě fáze:
  - Vložení jako u vyhledávacího stromu
  - Následuje kontrola vyváženosti
    - Pokud není strom vyvážený, provede se některá rotace

Doc. Ing. Radim Burget, Ph.D.

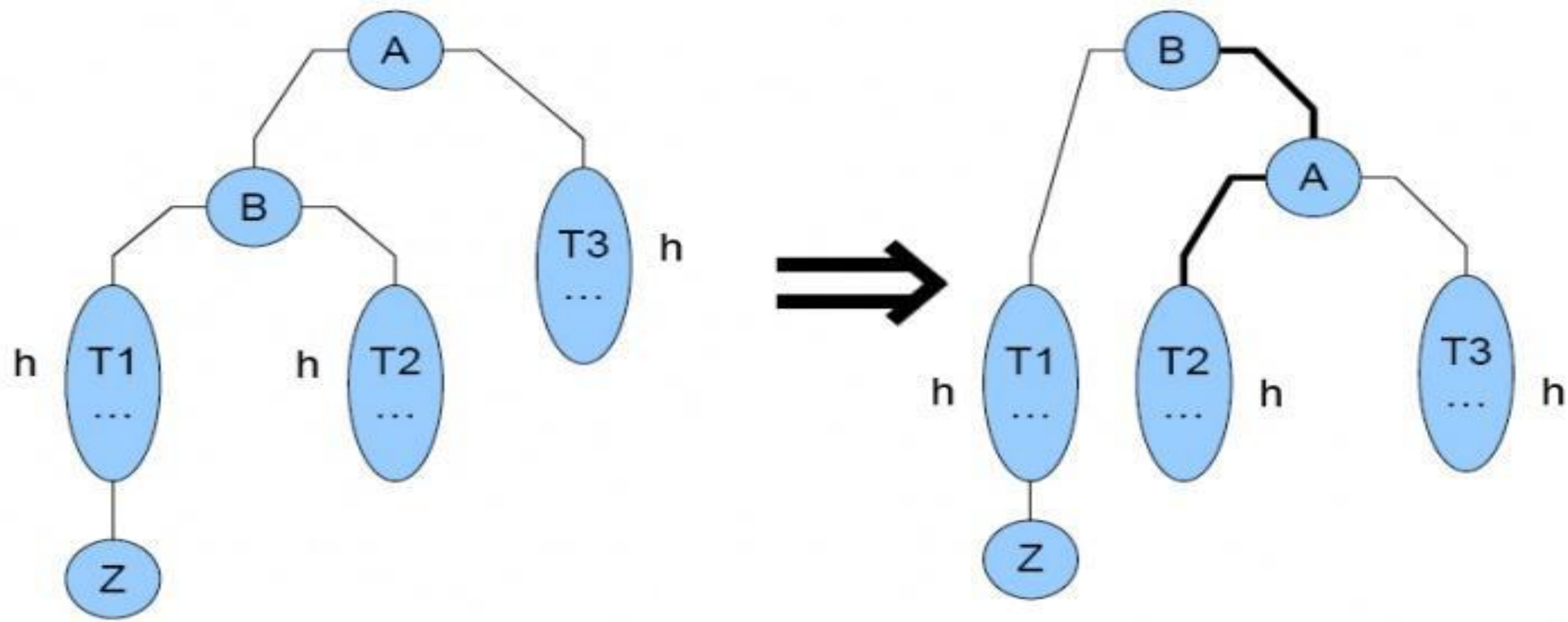
Vyvážené binární stromy

- Binární strom označíme za ***dokonale vyvážený***, jestliže pro jeho libovolný vrchol *v* platí, že počet vrcholů v levém a pravém podstromu vrcholu se liší nejvýše o 1.
- *Strom je **vyvážený**, právě když se hloubky obou podstromů každého uzlu připojených k jeho libovolnému vrcholu, liší nejvýše o 1.*



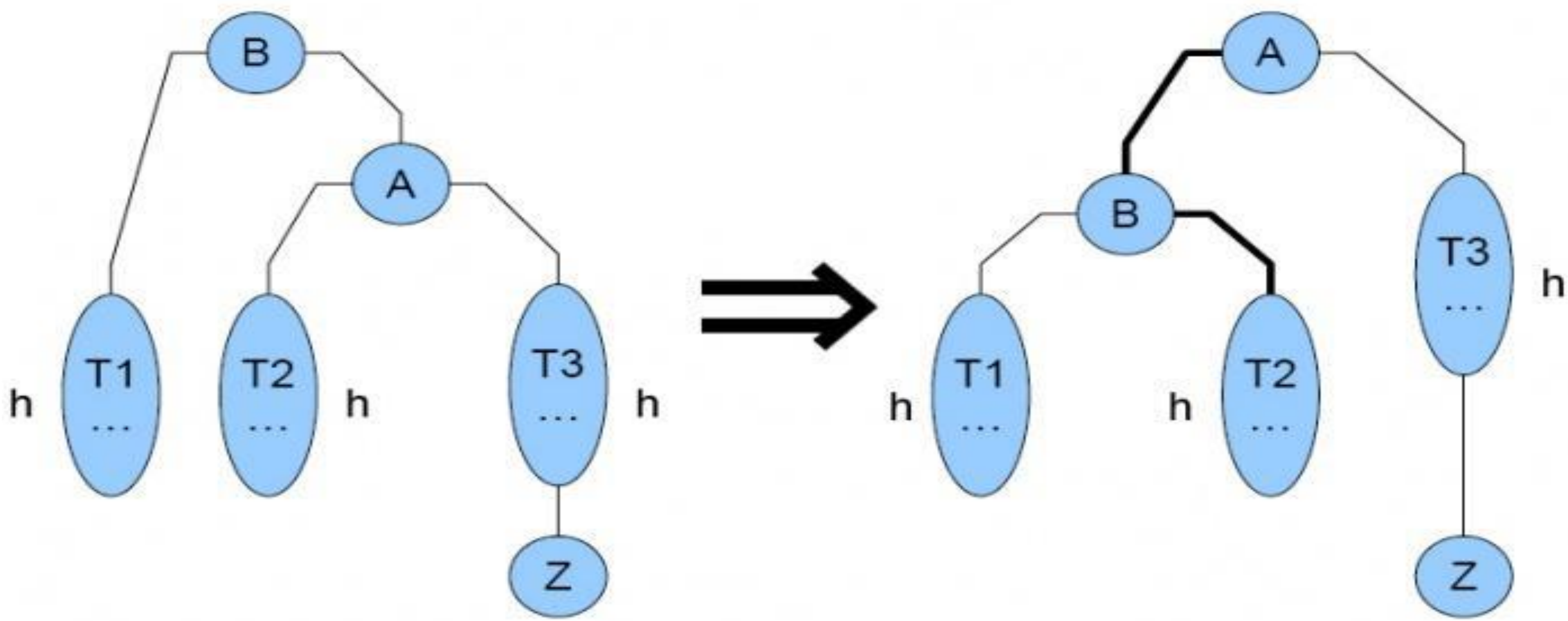
Vyvažování stromů

Jednoduchá levá rotace – SLR (Single Left Rotation)



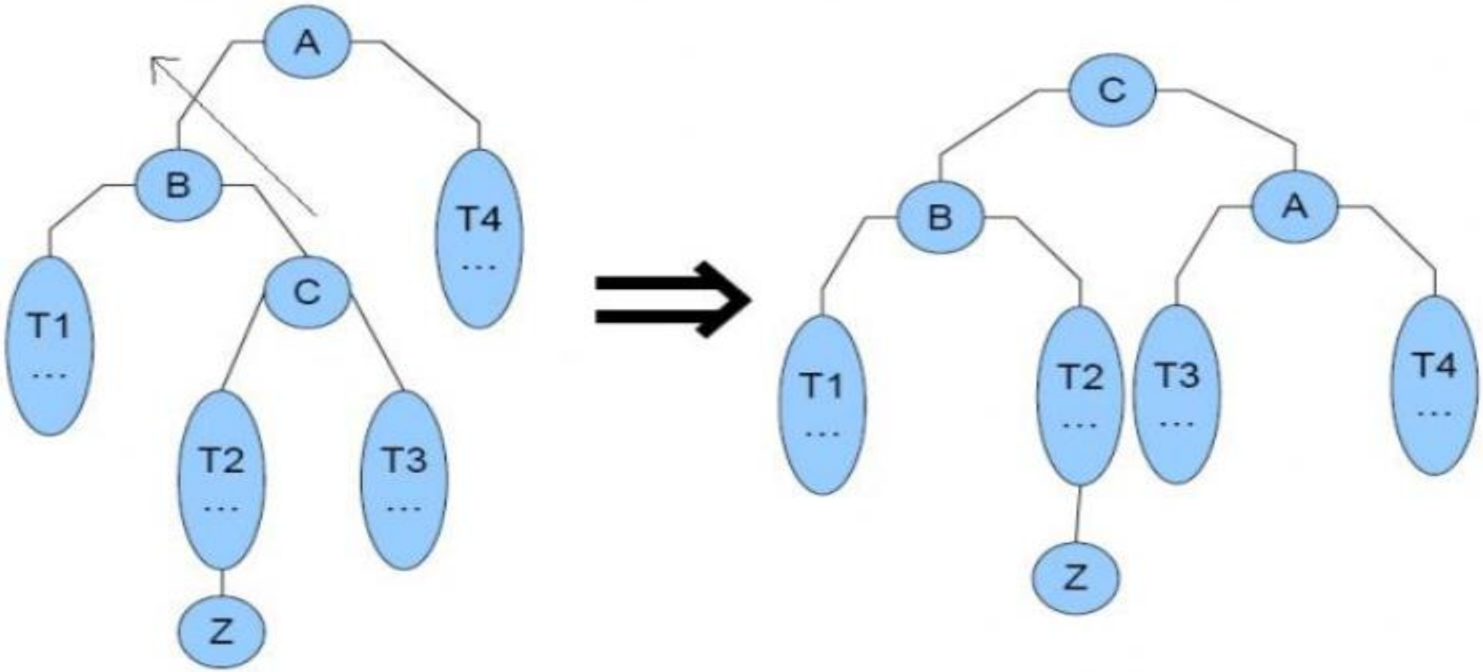
Vyvažování stromů

Jednoduchá pravá rotace – SRR (Single Right Rotation)



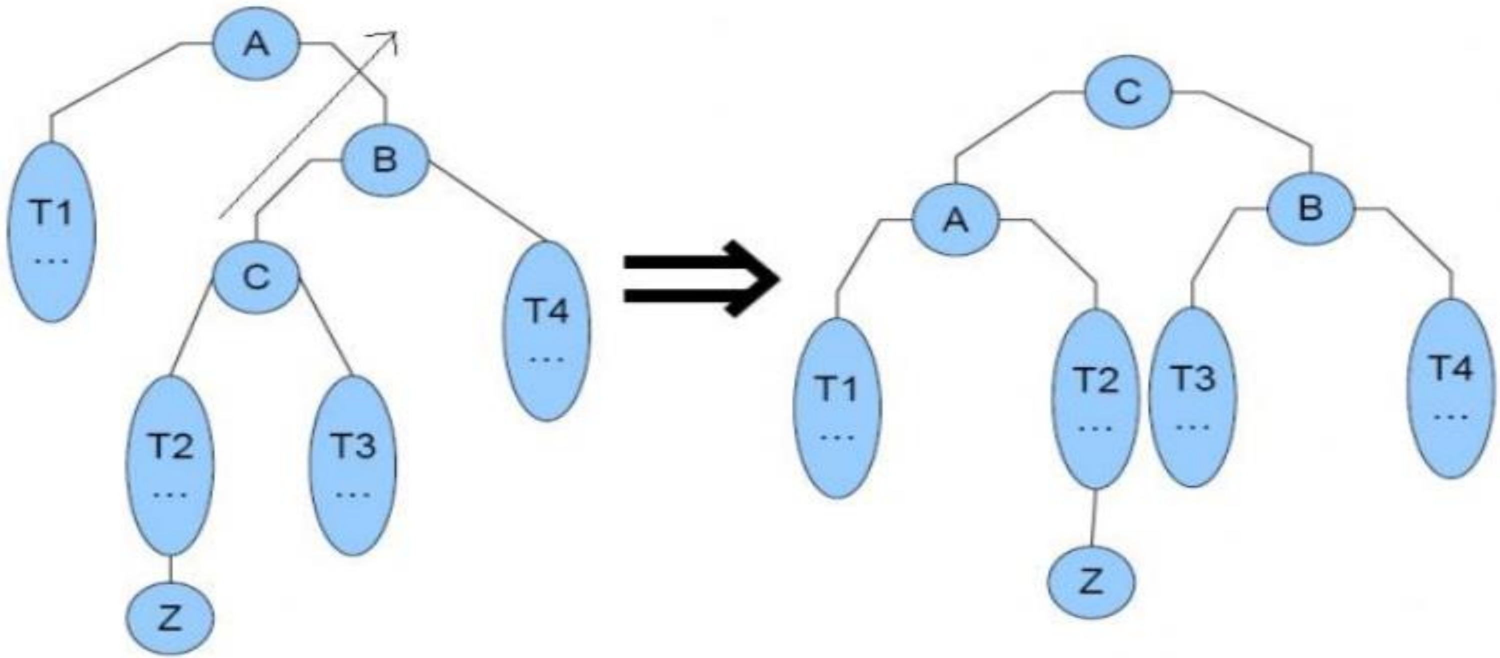
Vyvažování stromů

Dvojnásobná levá rotace – DLR (Double Left Rotation)



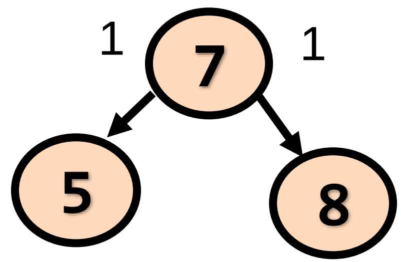
Vyvažování stromů

Dvojnásobná pravá rotace – DRR (Double Right Rotation)

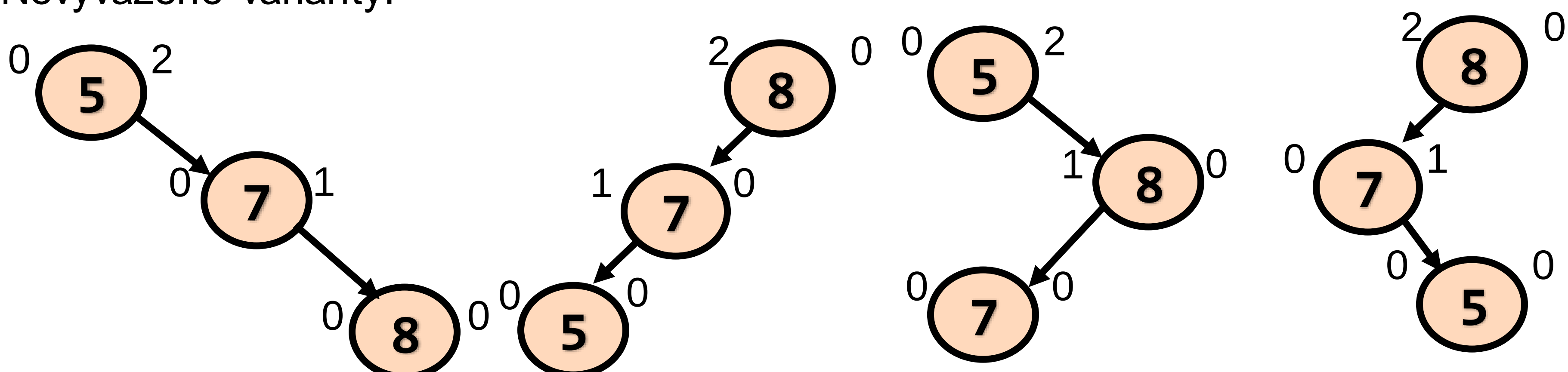


Mějme 5, 7, 8 v BVS? Jak mohou vypadat?

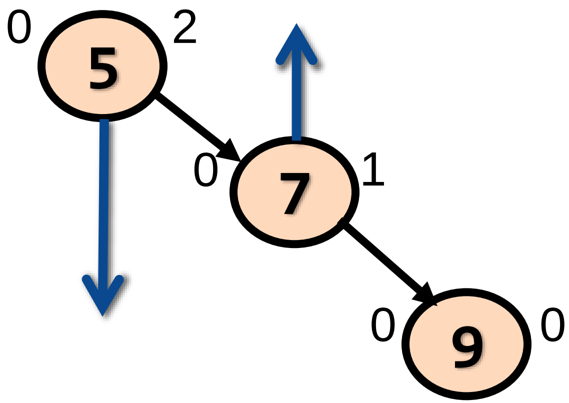
Vyvážená varianta:

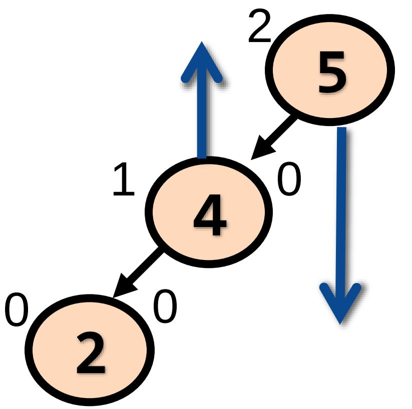


Nevyvážené varianty:

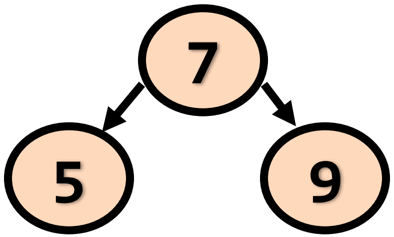


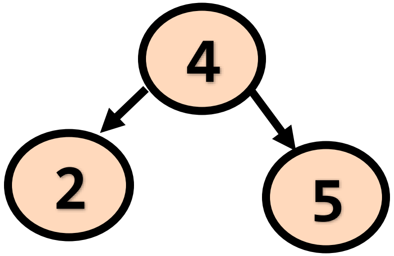
Jednoduché příklady – varianta do / anebo \



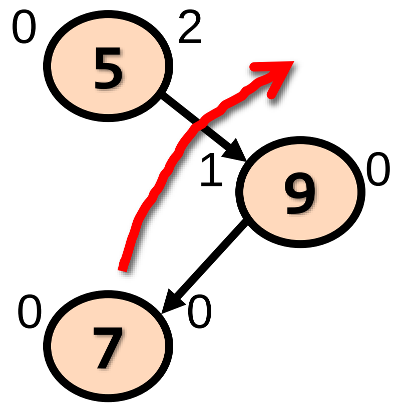


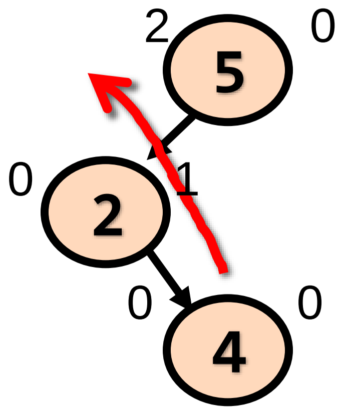
Výsledek po vyvážení pomocí AVL:



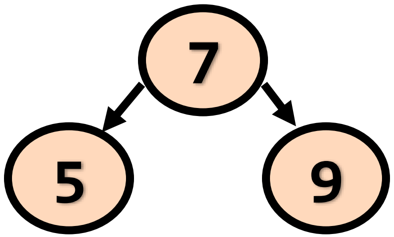


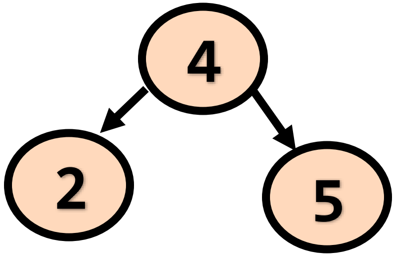
Jednoduché příklady – varianta do < anebo >





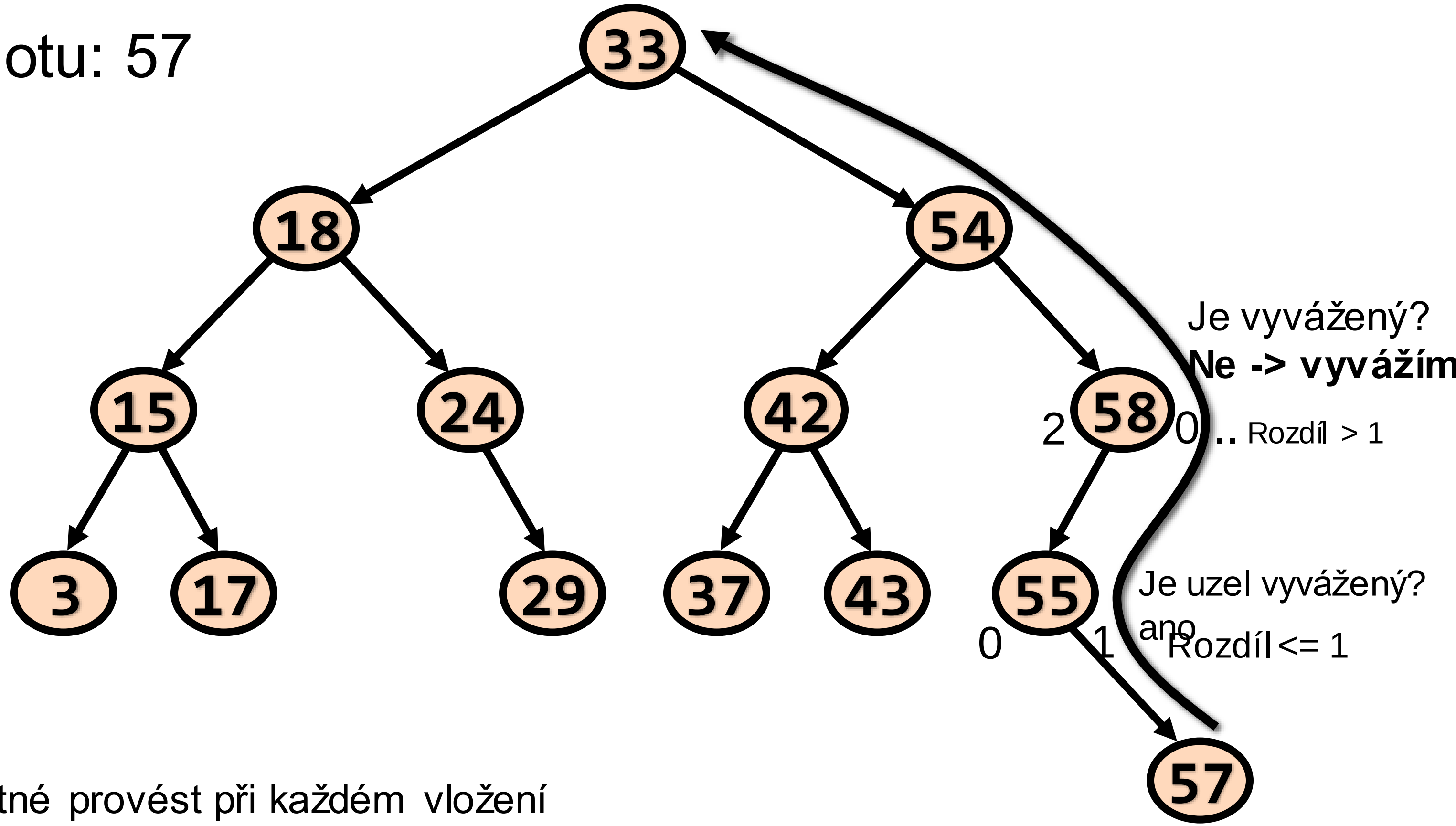
Výsledek po vyvážení pomocí AVL:





Kontrola vyváženosti

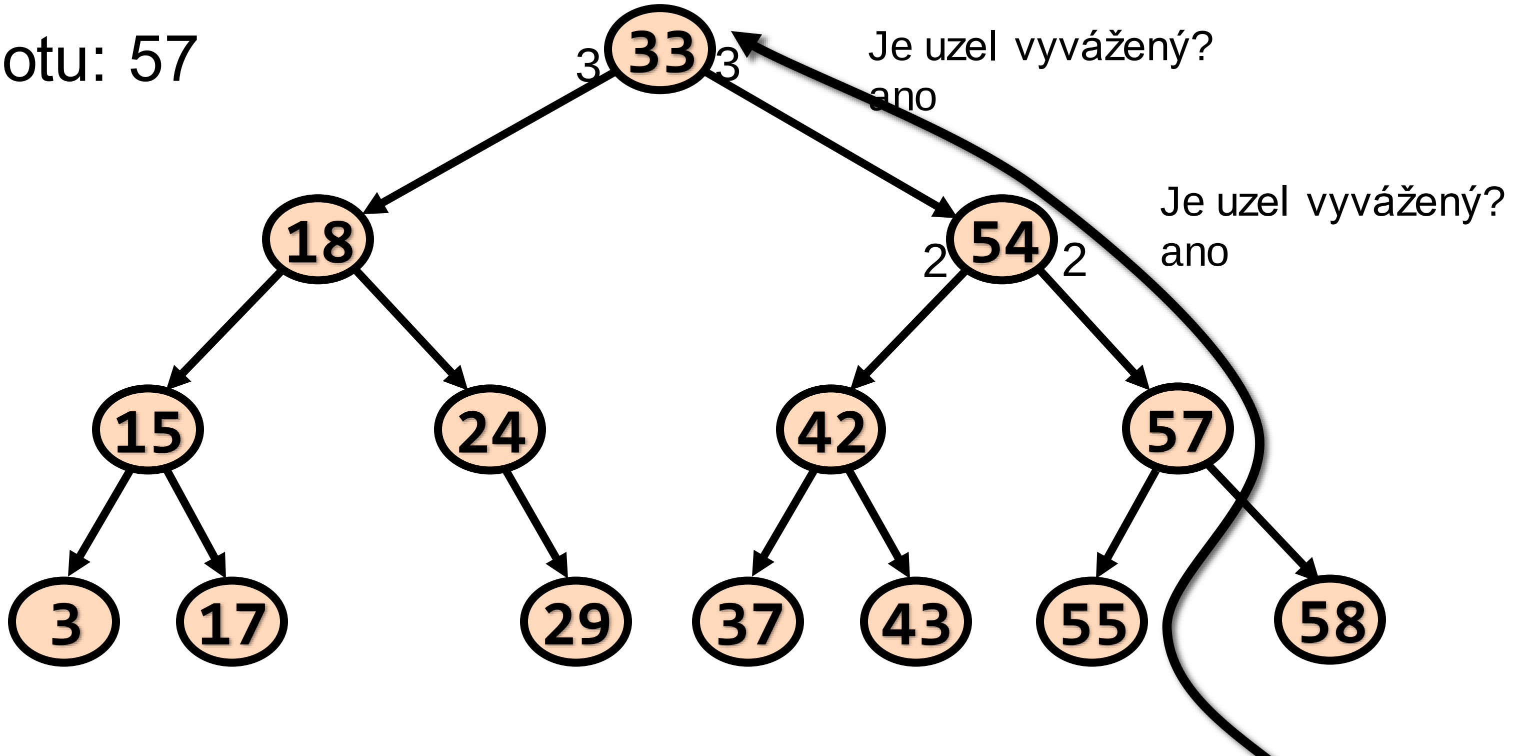
Vložte hodnotu: 57



Vyvážení je nutné provést při každém vložení

Kontrola vyváženosti

Vložte hodnotu: 57



Hotovo ☺

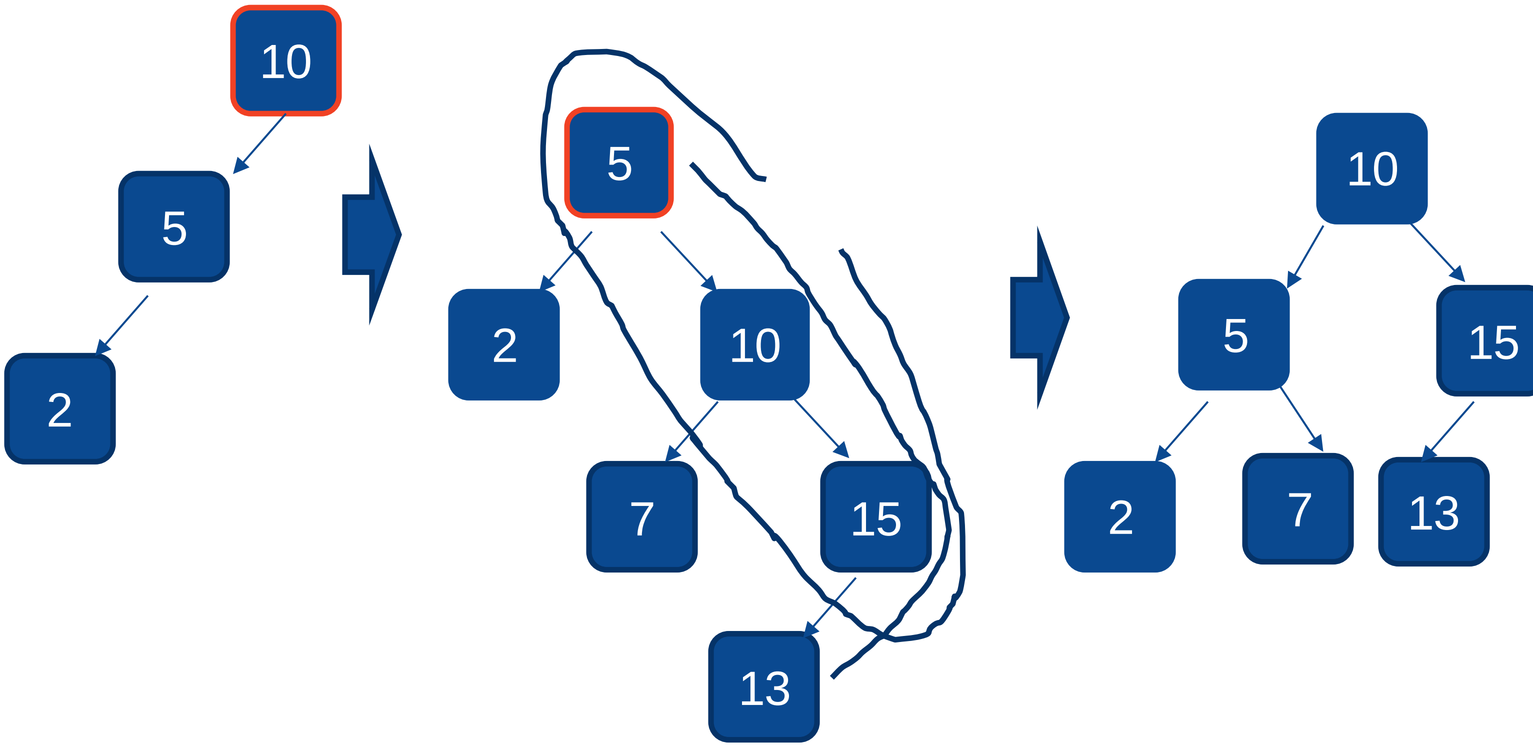
Příklad: Vyvažování stromů

- Vložte prvky do prázdného vyhledávacího stromu (vyvažovaného algoritmem AVL) v tomto pořadí:
- Jednoduché příklady:
    - 4, 6, 8
    - 7, 4, 1
    - 5, 3, 4
    - 6, 9, 8

  Pokročilejší příklady:
    - 5, 3, 8, 10, 12
    - 7, 4, 8, 10, 13
    - 8, 3, 10, 5, 4, 7

  Extrémní příklad:
    - 52, 8, 2, 11, 18, 36, 39, 38, 28, 71, 61, 66, 67, 58, 75, 87, 72, 95

Vložte: 10, 5, 2



Jednoduché příklady

- 4, 6, 8; **7, 4, 1**; 5, 3, 4; **6, 9, 8**

Pokročilejší příklady

- 5, 3, 8, 10, 12; **7, 4, 8, 10, 13**; 8, 3, 10, 5, 4, 7

Pokročilejší příklad

- 52, 8, 2, 11, 18, 36, 39, 38, 28, 71, 61, 66, 67, 58, 75, 87, 72, 95

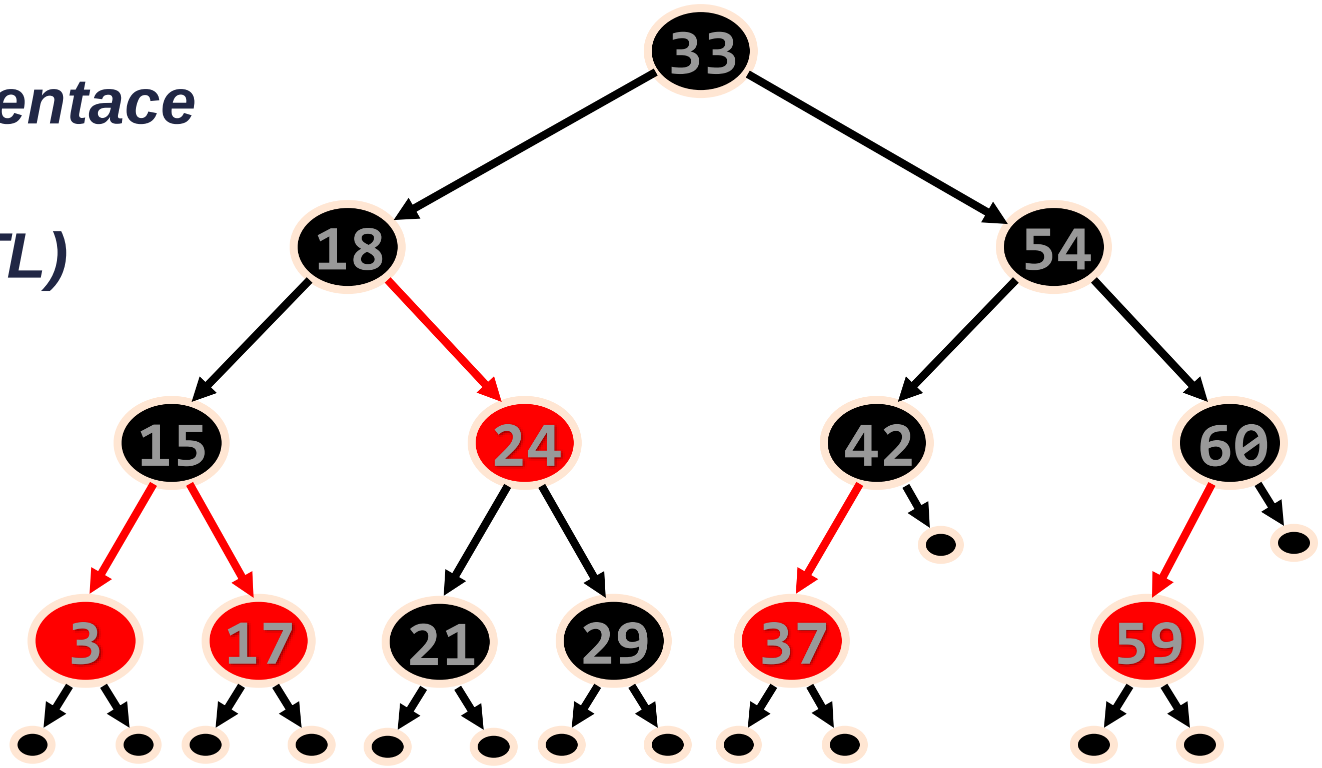
Red-Black stromy

Pro Red-Black stromy platí tato pravidla:

- Každý uzel je vždy červený nebo černý
- Kořen stromu je černý
- Všechny listy jsou černé
- Oba potomci červeného uzlu jsou černé
- Každá cesta z libovolného uzlu do jeho podřízených listů obsahuje stejný počet černých uzlů

Red – Black stromy – ukázka

Implementace JAVA, C++ (STL)



Red – Black stromy

Odkazy na příklady

Red-Black stromy nebudou součástí příkladu v rámci zkoušky (je ale nutné vědět co to je).

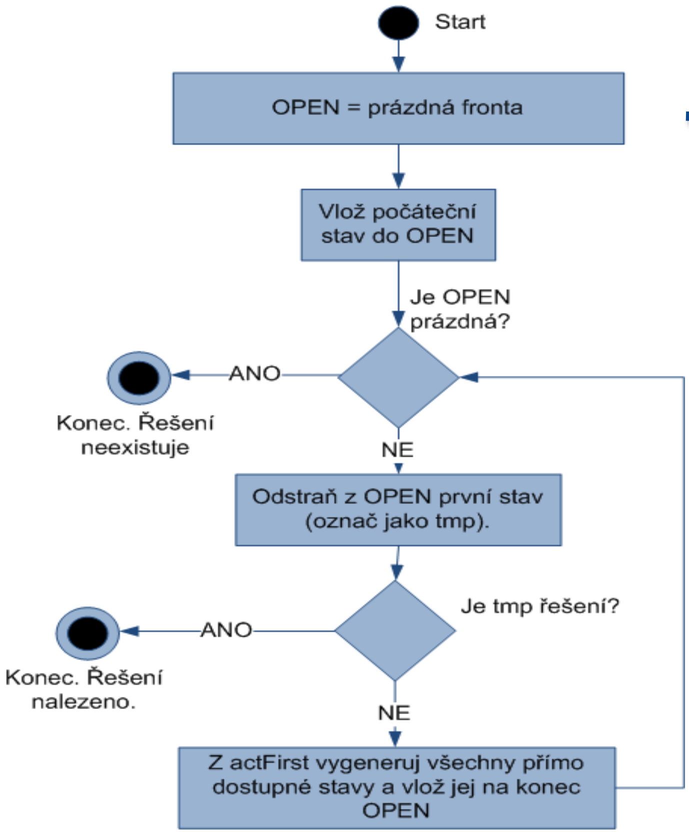
Více informací:

- Animace binárních vyhledávacích stromů
- Vkládání do Red - Black stromu - nejhorší případ
- Zdrojové kódy Red - Black stromu v Javě a mnoho dalších

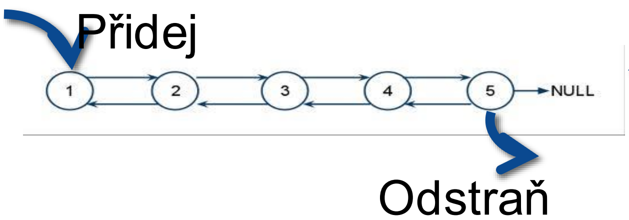
Prohledávání do šířky - algoritmus

Prohledávání do šířky – BFS – Implementace

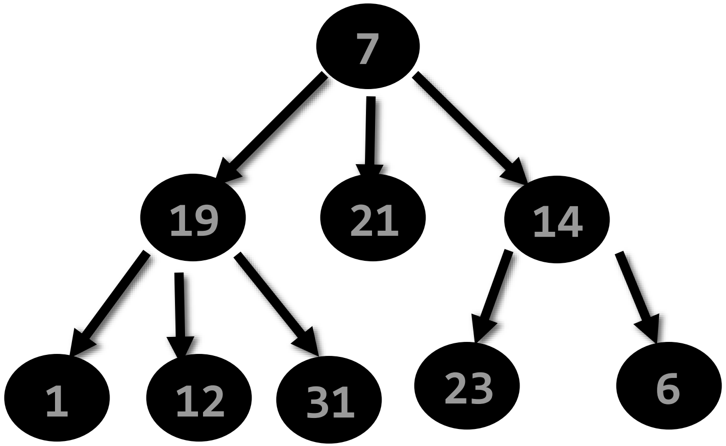
Jak se dostanu ze stavu 7 do stavu **6 s nejmenším počtem tahů** ?



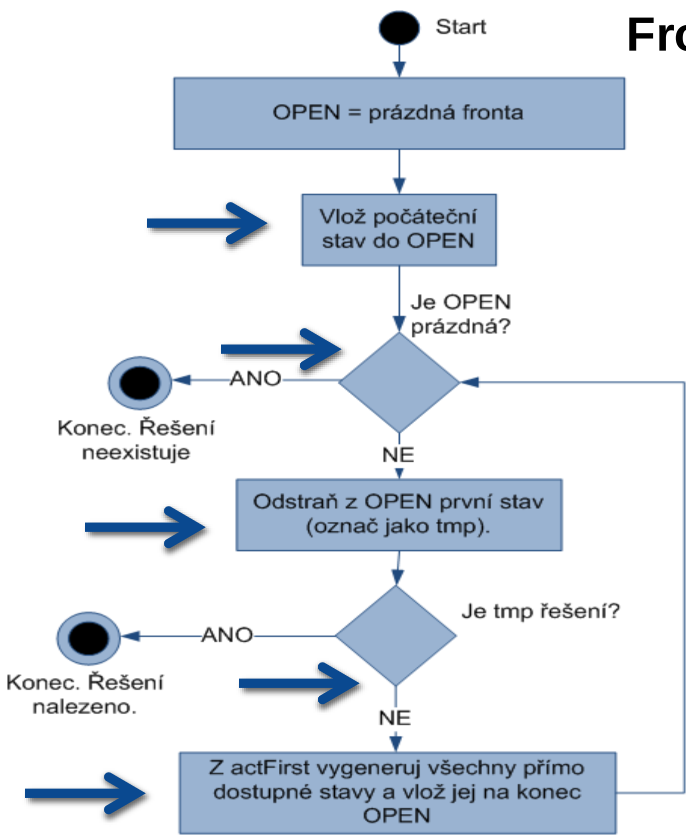
Fronta OPEN:



Jediná odlišnost oproti DFS

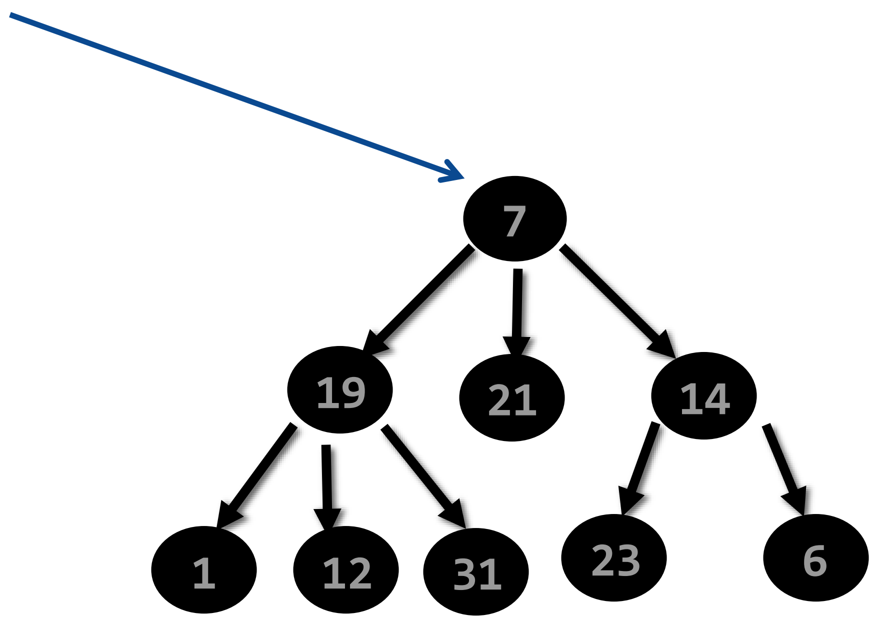


Prohledávání do šířky – BFS - Implementace

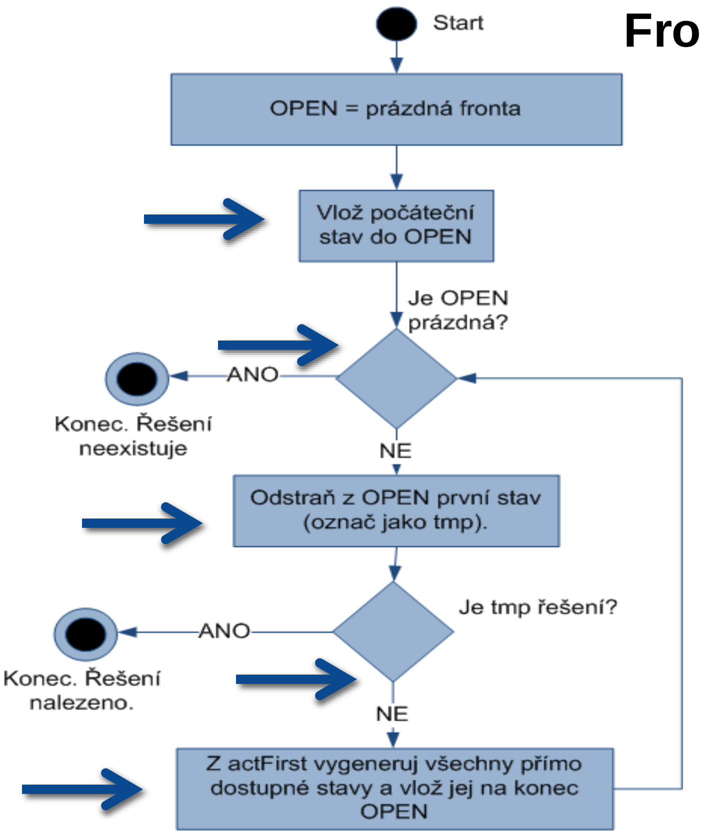


Fronta OPEN: ~~7~~ 19 21 14

tmp:

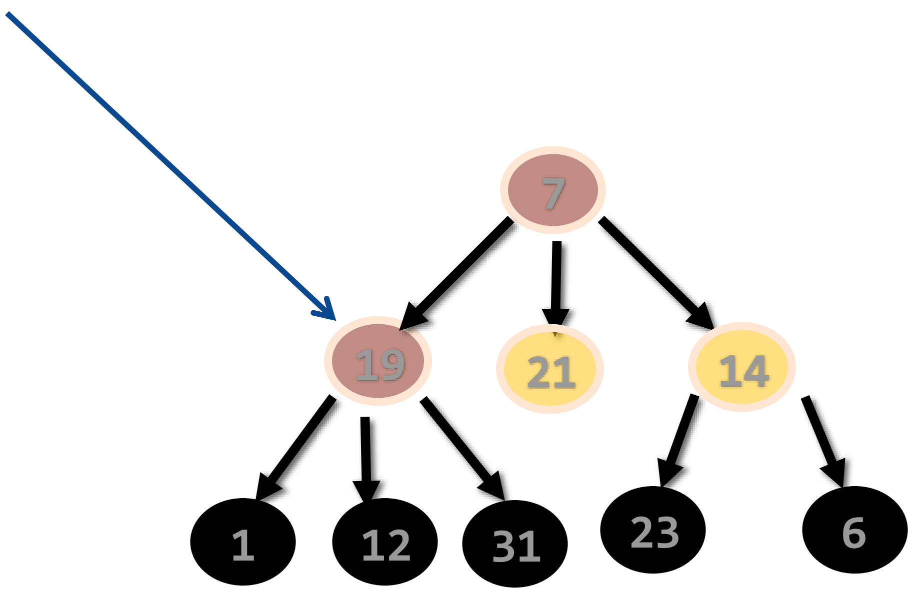


Prohledávání do šířky – BFS - Implementace



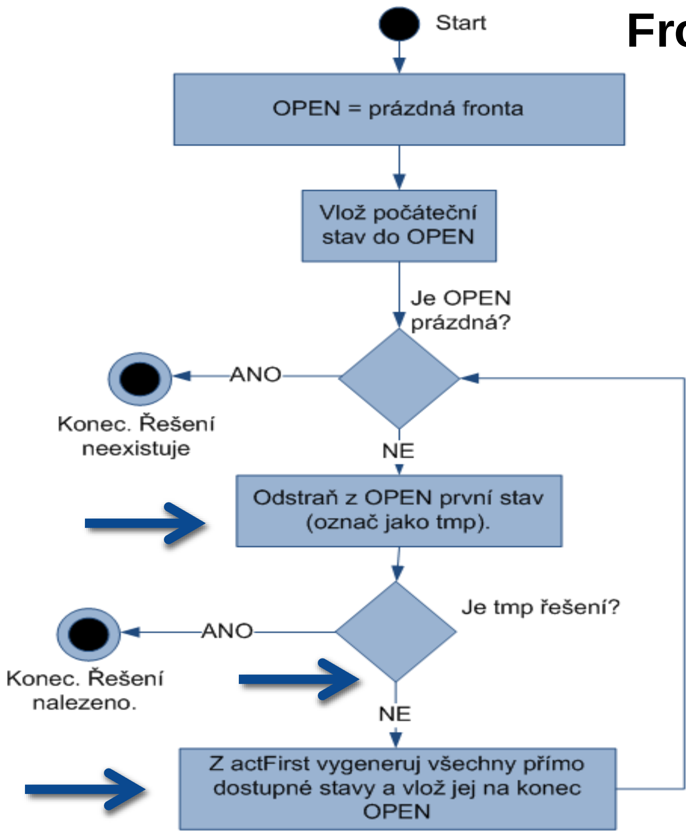
Fronta OPEN: ~~7~~ ~~19~~ 21 14 1 12 31

tmp:



Doc. Ing. Radim Burget, Ph.D.

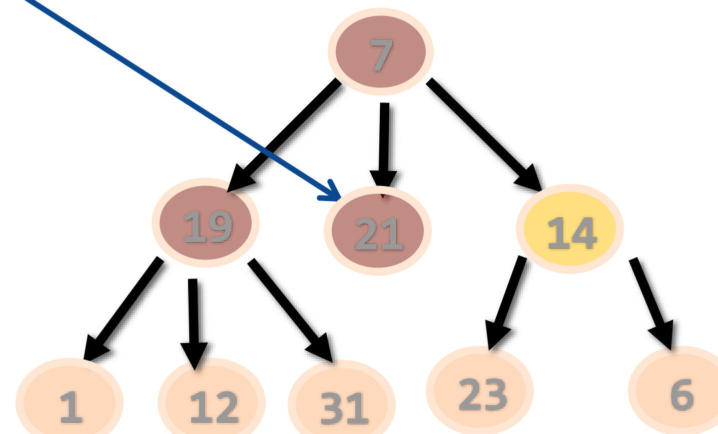
Prohledávání do šířky – BFS - Implementace



Fronta OPEN: ~~7~~ ~~19~~ ~~21~~ 14 1 12 31

tmp:

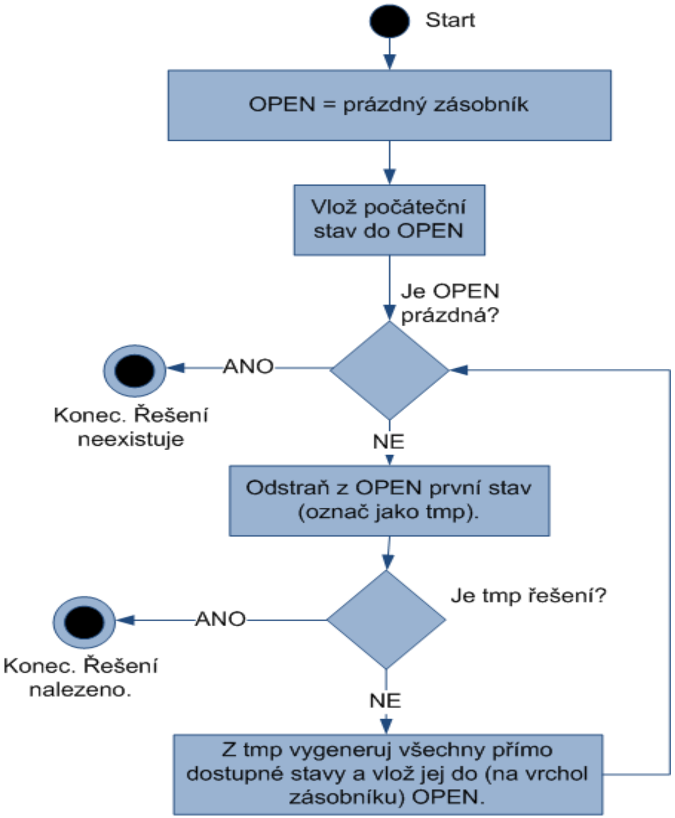
A tak dále, dokud není fronta prázdná



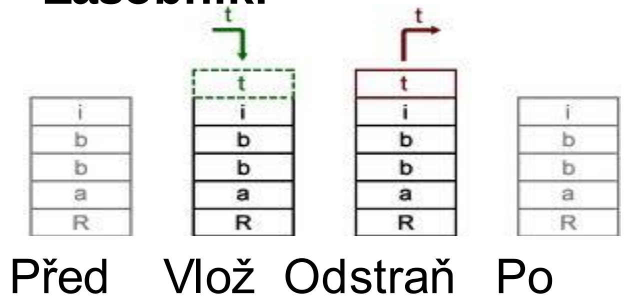
Prohledávání do hloubky - algoritmus

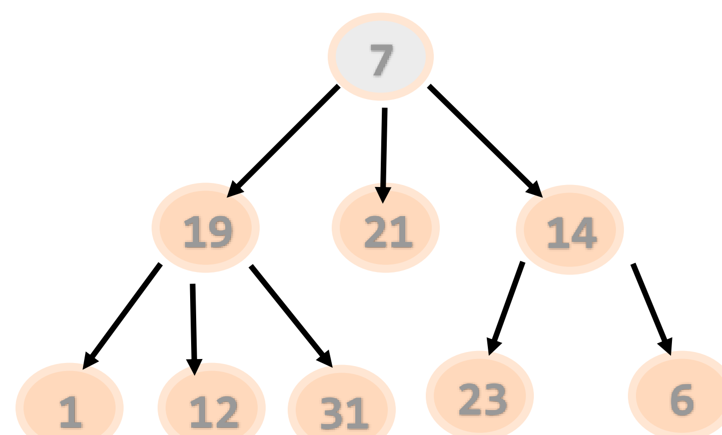
Prohledávání do hloubky – DFS - Implementace

Jak se dostanu ze stavu 7 do stavu 6 (existuje taková cesta) ?



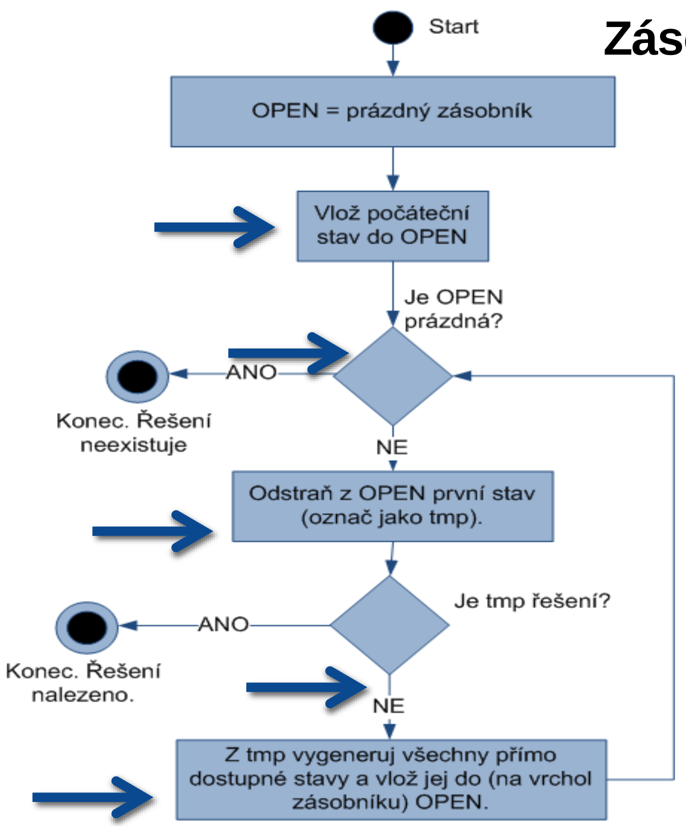
Zásobník:





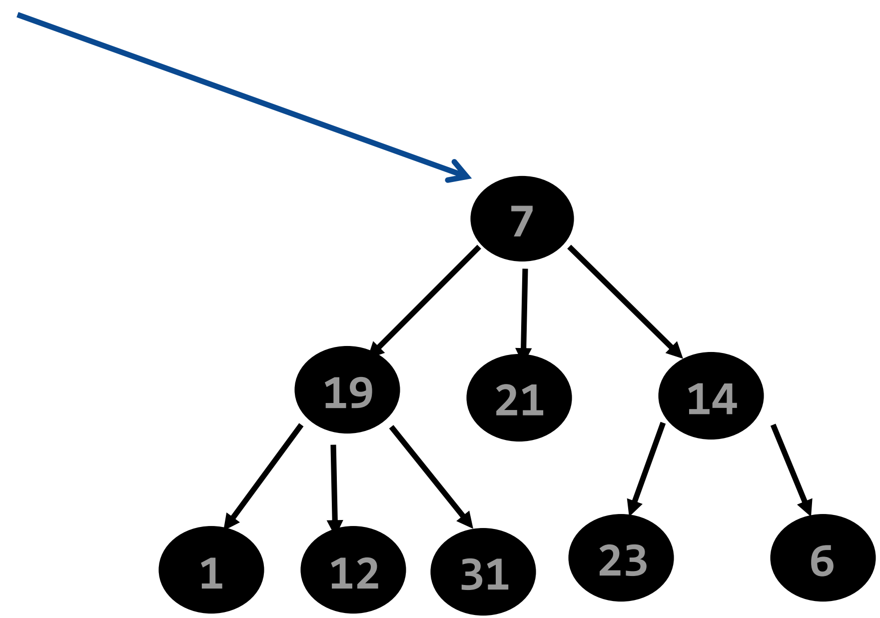
Doc. Ing. Radim Burget, Ph.D.

Prohledávání do hloubky – DFS - Implementace

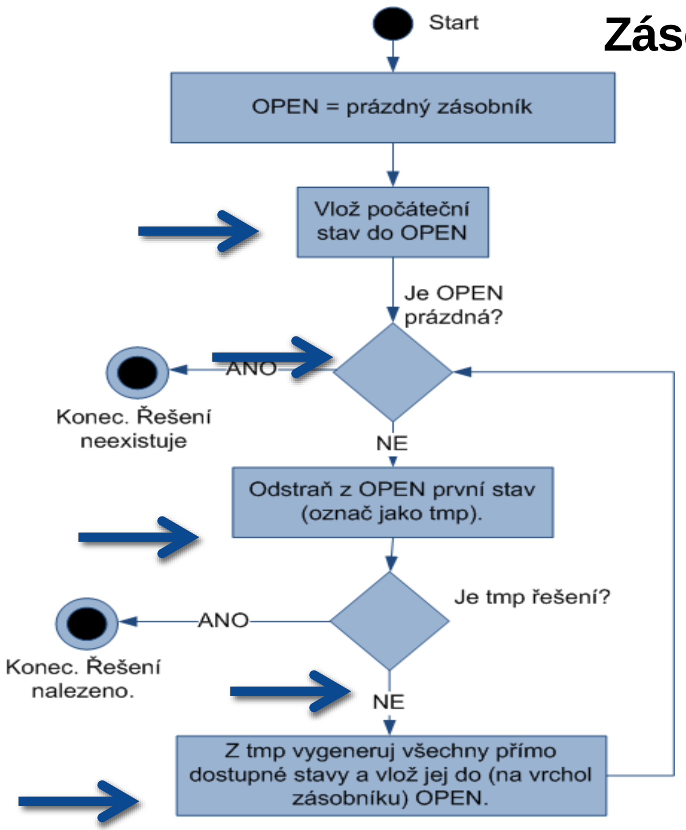


Zásobník OPEN: ~~7~~ 19 21 14

tmp:

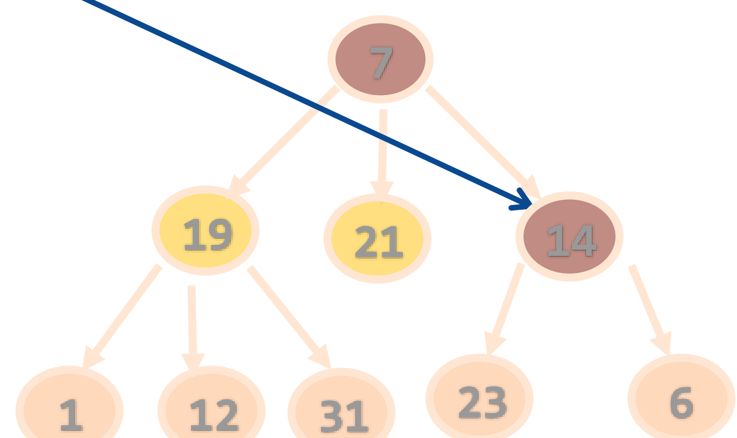


Prohledávání do hloubky – DFS - Implementace



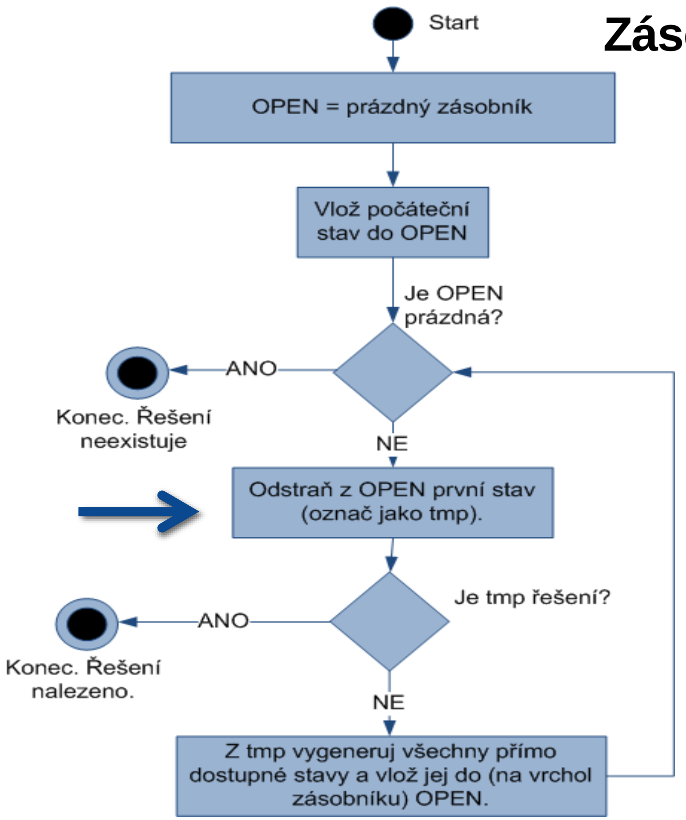
Zásobník OPEN: ~~7~~ 19 21 ~~14~~ 23 6

tmp:



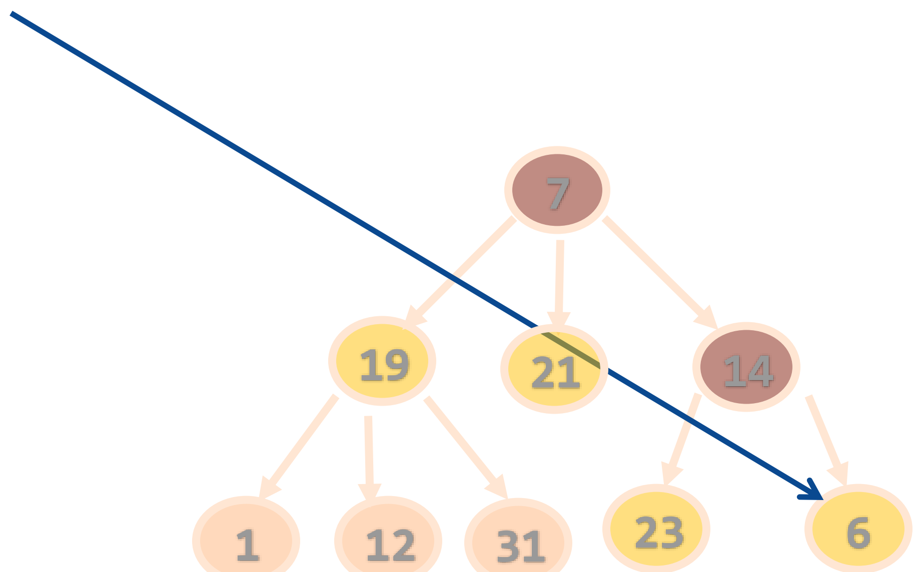
Doc. Ing. Radim Burget, Ph.D.

Prohledávání do hloubky – DFS - Implementace



Zásobník OPEN: ~~7~~ 19 21 ~~14~~ 23 ~~6~~

tmp:



Binární vyhledávací strom - shrnutí

- Operace nad BVS:
  - SEARCH $O(h)$
  - MINIMUM $O(h)$
  - MAXIMUM $O(h)$
  - INSERT $O(h)$
  - DELETE $O(h)$
  - BFS $O(v+e)$
  - DFS $O(v+e)$

h… výška stromu
h =$\log_2(n)$

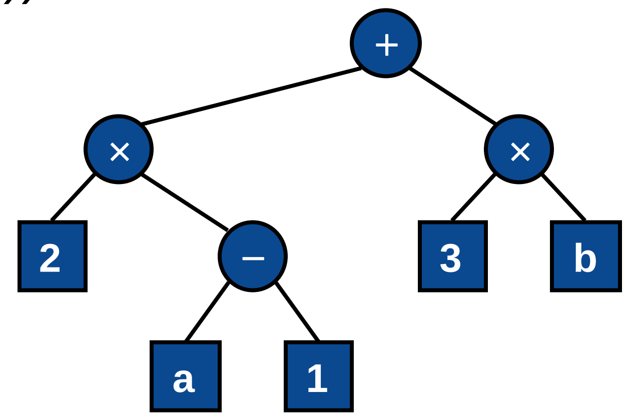
v … počet vrcholů
e … počet hran

Výška náhodně postaveného **binárního vyhledávacího stromu** na **n** různých klíčích je **O(2 $\log_2$ n)**

Aritmetický výrazový strom

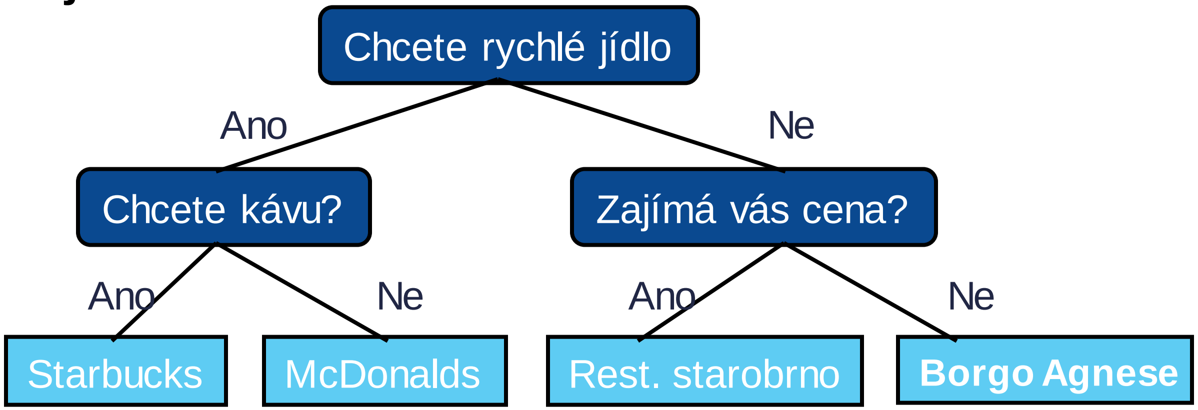
- Binární strom asociovaný s aritmetickým výrazem:
  - Interní uzly: operátory
  - Externí uzly: operandy
- **Příklad:** Aritmetický výrazový strom pro výraz

$(2 \times (a - 1) + (3 \times b))$



Rozhodovací stromy

- Binární rozhodovací strom asociovaný s rozhodovacím procesem
  - Vnitřní uzly: otázka s odpovědí ano/ne
  - Listy: rozhodnutí
- Příklad: rozhodnutí o jídle



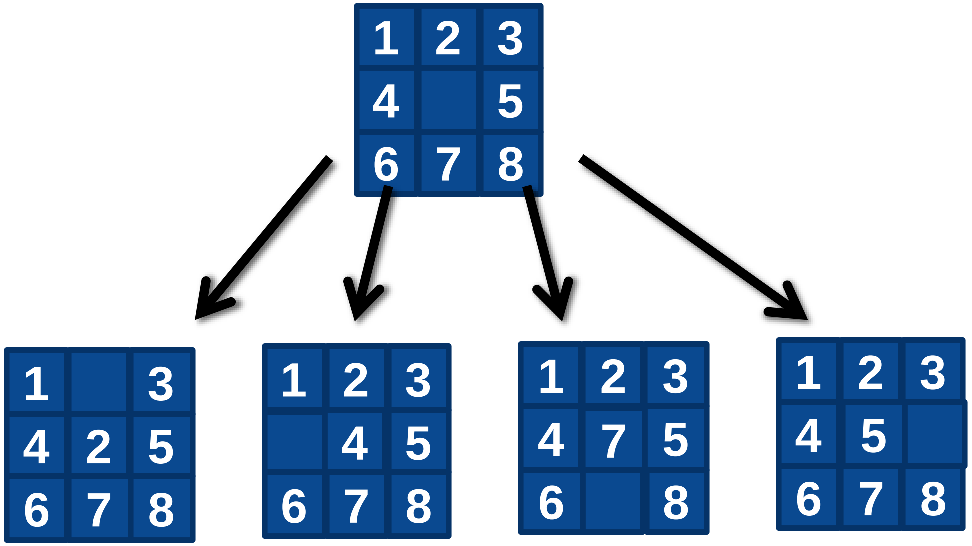
Huffmanovo kódování

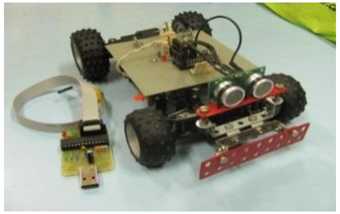
- **Huffmanovo kódování** je technika pro bezeztrátovou kompresi dat.
- Konvertuje znaky vstupního souboru do bitových řetězců různé délky.

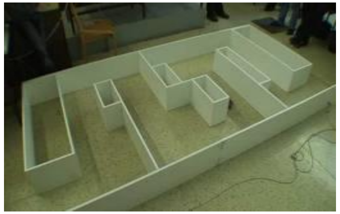
Doc. Ing. Radim Burget, Ph.D.

Význam prohledávání stromů

- Přestože byl algoritmus představen z pohledu teorie, má mnoho praktických využití – zejména v kombinaci s grafy (budeme se jimi zabývat později)
- Mnoho problémů lze převést do podoby stromové struktury







představen

Příklad I.

- Pracujeme na projektu pro analýzu sémantického obsahu textových dokumentů. Vytvořme filtr, který získá seznam vyskytujících se slov v seřazeném pořadí.
- Analyzovat se budou miliony dokumentů
- Lineární struktury (lineární složitost – odstrašující případ)
- Hash – výkonnostně dobré, neřadí
- TreeSet – **správná volba**

Příklad II.

- V rámci Real-time protokolu chodí každou sekundu přibližně 50 paketů. Jelikož jsou pakety přenášeny skrze UDP protokol, je potřeba zajistit: řazení, eliminaci duplicit
- Možnosti:
  - Pole anebo seznamy (vyhledávání či řazení časově náročné)
  - HashSet (výkonostně dobré, neřadí prvky)
  - TreeSet (**správná volba**)

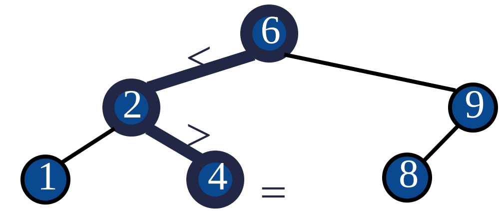
Shrnutí

- Obecný, n-ární, binární
- Vyhledávací
- Vyvážený

strom

- Operace nad stromy:
  - Vkládání, odstranění, vyhledávání
- Průchody:
  - In-order, pre-order, post-order
  - DFS
  - BFS (po úrovních)
- Vyvažování:
  - AVL vyvažování
  - V praxi se používá: Red-Black strom (vědět, že existuje + umět srovnat, nebude předmětem zkoušení)

K zamyšlení



- Proč nastává ClassCastException?
  - = nemohu přetypovat na třídu java.lang.Comparable
  - A jak ji opravit?

```
public class Car {
    public int vin;
}
```

(Vehicle Identification Number)

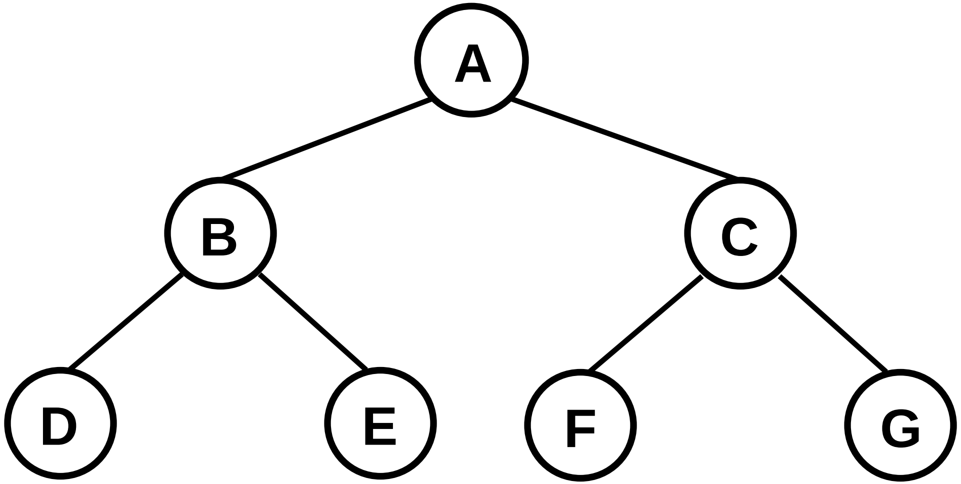
```
TreeSet<Car> mySet = new TreeSet<>();
mySet.add(new Car());
```

```
Exception in thread "main" java.lang.ClassCastException: cz.vutbr.feec.dsa.Car cannot be cast to java.lang.Comparable
	at java.util.TreeMap.compare(Unknown Source)
	at java.util.TreeMap.put(Unknown Source)
	at java.util.TreeSet.add(Unknown Source)
	at cz.vutbr.feec.dsa.Runnable.main(Runnable.java:8)
```

Děkuji za pozornost

Otázka z přijímacího pohovoru

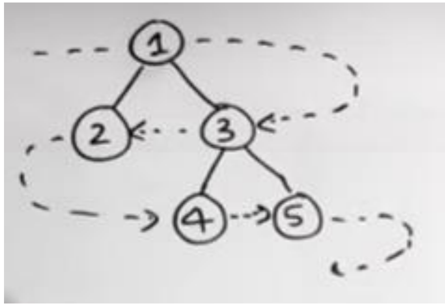
- Převeďte **binární vyhledávací strom** na seřazený **obousměrně vázaný seznam**. Nesmí být vytvářena nová paměť, pouze lze měnit ukazatele



D B E A F C G

Otázka z přijímacího pohovoru

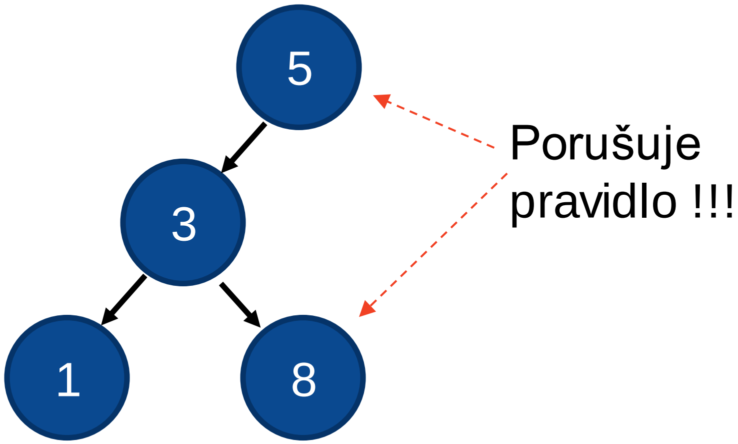
- Nalezněte nejhlubší uzel v binárním stromu.
- Napište funkci, která rozhodne, zdali je binární strom vyvážený.
- Nalezněte všechny uzly ve vzdálenosti k od kořene.
- Zig-zag průchod stromem



- Ze seřazeného pole vytvořte binární vyhledávací strom
- Ověřte, že všechny listy stromu jsou stejné úrovně

Otázka z přijímacího pohovoru

- Napište funkci k určení, zda daný binární strom (odlišná celá čísla) je platný binární vyhledávací strom.
  - Složitější problém
    - Přímá rekurze nefunguje.



Otázka z přijímacího pohovoru

- Napište funkci pro průchod binárním vyhledávacím stromem
  - bez rekurze / zásobníku / paměti navíc.
  - Tip - můžete rozšířit strukturu dat uzlu. Do uzlu však nemůžete přidat navštívené pole.

- Vytiskněte strom jako:

(Rodič (leftchild (leftchild, rightchild), rightchild (leftchild, rightchild)))

Otázka z přijímacího pohovoru

- Napište kód pro Huffmanovo kódování

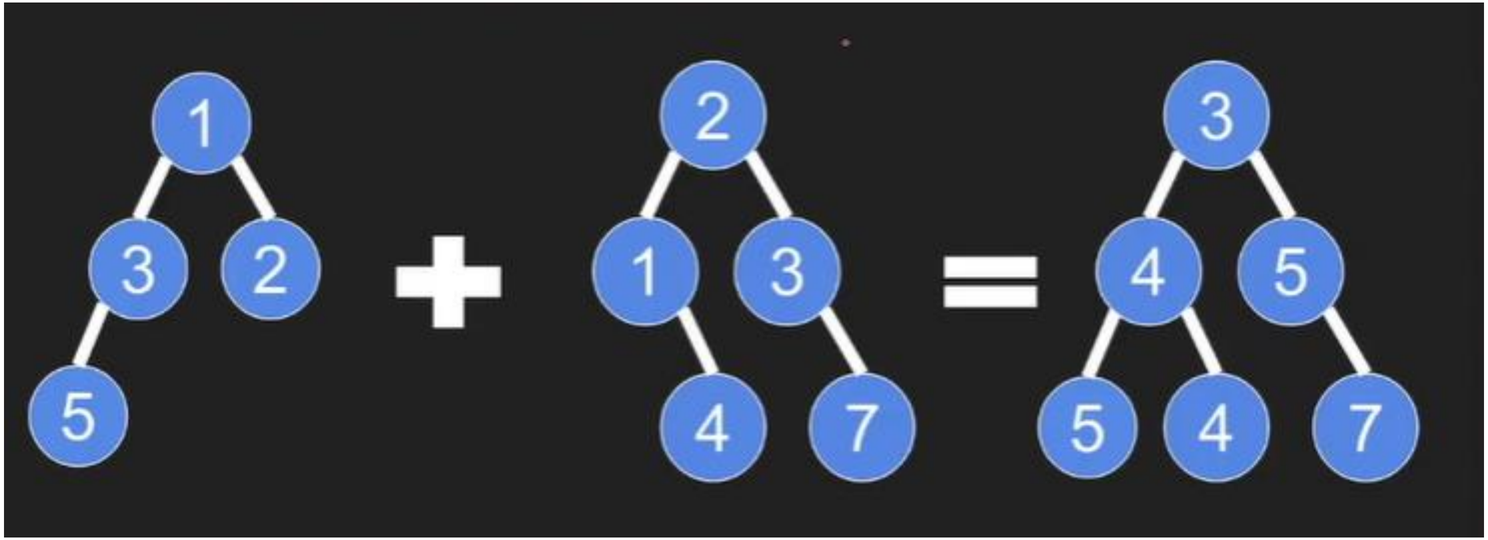
- Pozn: Hufmanovo kódování je stromová struktura

Char	četnost
E	125
T	93
A	80
O	76
I	73
N	71
S	65
R	61
H	55
L	41
D	40
C	31
U	27

Otázka z přijímacího pohovoru

Google

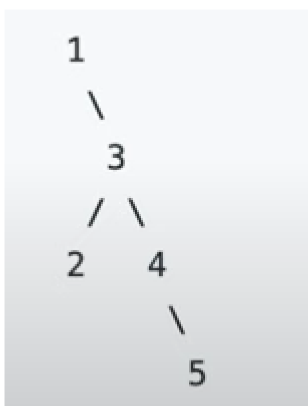
- Slučte 2 stromy tak, aby vznikl binární vyhledávací strom



Otázka z přijímacího pohovoru

- Najděte nejdelší po sobě jdoucí sekvenci v binárním rozhodovacím stromě.

- Příklad



Nejdelší je 3, 4, 5
odpověď je tedy 3.

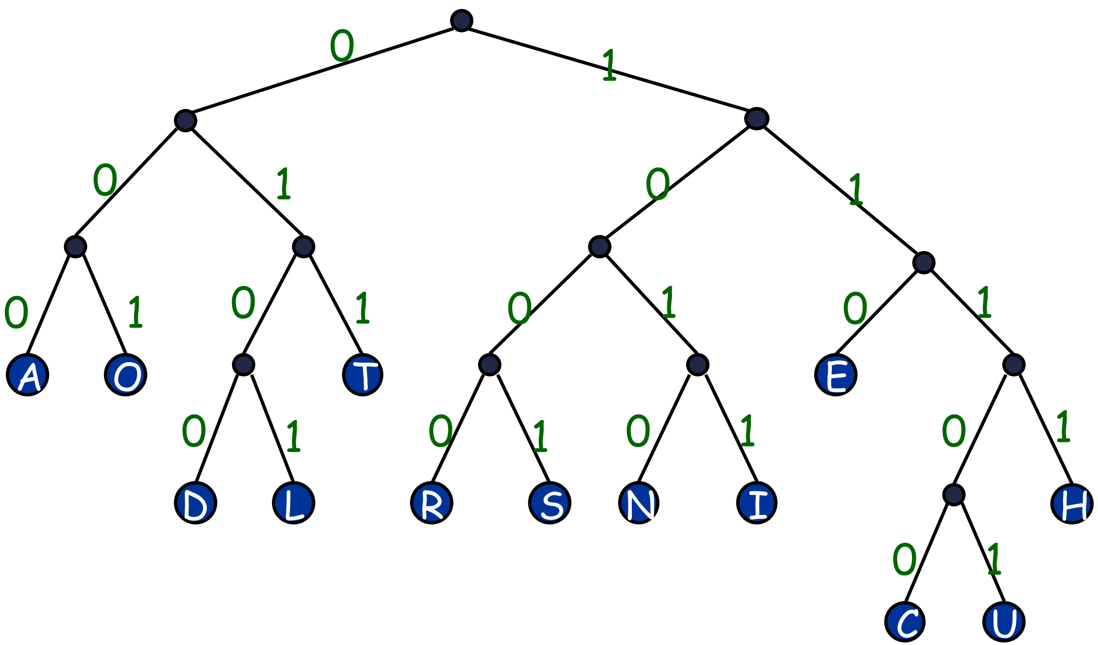
Otázka z přijímacího pohovoru

- Nalezněte nejbližšího společného předchůdce v binárním stromu
  - Určete vzdálenost mezi dvěma uzly v binárním stromu (obecném, nejen vyhledávacím)
- Rozhodněte, zdali dva binární stromy jsou identické, či nikoli

Rovnost dvou stromů

- Navrhněte algoritmus, který ověří, že dva binární stromy jsou identické.
- Nelze alokovat novou paměť, pouze jednorozměrné pomocné reference.

Řešení: Otázka z přijímacího pohovoru



Char	Freq	Fixed	Huff
E	125	0000	110
T	93	0001	011
A	80	0010	000
O	76	0011	001
I	73	0100	1011
N	71	0101	1010
S	65	0110	1001
R	61	0111	1000
H	55	1000	1111
L	41	1001	0101
D	40	1010	0100
C	31	1011	11100
U	27	1100	11101
Total	838	4.00	3.62