Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

BAKALÁŘSKÁ PRÁCE

Demonstrace práce s datovými strukturami

2018

Patrik Becher

Vedoucí práce: Mgr. Tomáš Kühr, Ph.D.

Studijní obor: Aplikovaná informatika, prezenční forma

Bibliografické údaje

Autor:	Patrik Becher
Název práce:	Demonstrace práce s datovými strukturami
Typ práce:	bakalářská práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2018
Studijní obor:	Aplikovaná informatika, prezenční forma
Vedoucí práce:	Mgr. Tomáš Kühr, Ph.D.
Počet stran:	30
Přílohy:	1 CD/DVD
Jazyk práce:	český

Bibliograhic info

Author:	Patrik Becher
Title:	Data Structure Demonstration
Thesis type:	bachelor thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2018
Study field:	Applied Computer Science, full-time form
Supervisor:	Mgr. Tomáš Kühr, Ph.D.
Page count:	30
Supplements:	1 CD/DVD
Thesis language:	Czech

Anotace

Cílem bakalářské práce bylo vytvořit nástroj pro podporu výuky algoritmizace, konkrétně práce se základními stromovými datovými strukturami (binární vyhledávací stromy, AVL stromy, červenočerné stromy). Výsledná aplikace podporuje vizualizaci vybraných datových struktur, včetně názorné demonstrace běžně prováděných operací s těmito datovými strukturami se souběžným zobrazením pseudokódu prováděné operace.

Synopsis

Toto mně doplní Míša... :D

Klíčová slova: Binární vyhledávací stromy, Binární strom, AVL strom, Červenočerný strom, Stromové animace Java, JavaFX

Keywords: Binary search trees, Binary Tree, AVL tree, Redblack tree, Tree animations, Java, JavaFX

Rád bych poděkoval panu Mgr. Tomáši Kührovi Ph.D. za vedení této bakalářské práce a panu RNDr. Arnoštu Večerkovi za odbornou pomoc a poskytnuté materiály k práci. Dále bych chtěl poděkoval mé rodině a přítelkyni za podporu při tvorbě.

Místopřísežně prohlašuji, že jsem celou práci včetně příloh vypracoval/a samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.

datum odevzdání práce podpis autora

Obsah

Seznam obrázků

Seznam zdrojových kódů

1 Úvod

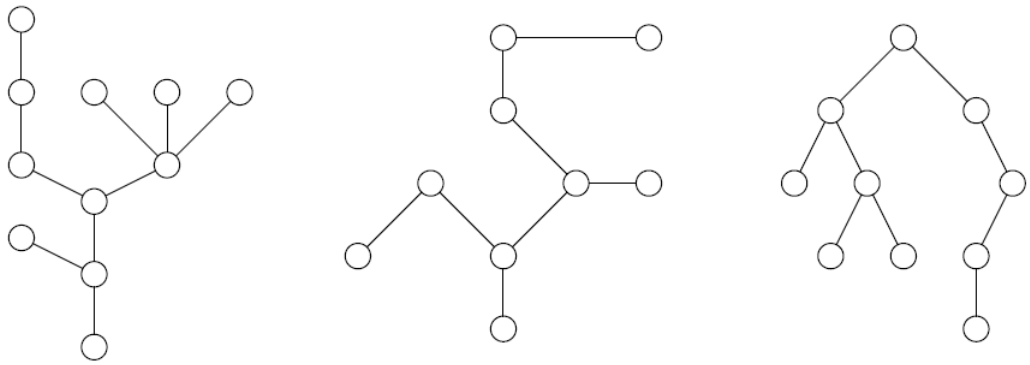
Tato aplikace vznikla za účelem výuky základních binárních stromů. Obsahuje podporu pro *Binární vyhledávací*, *AVL* a *Červenočerné stromy*. Program pomocí animací zobrazuje operace: *Vyhledávání*, *Vkládání* a *Odebírání* prvků ze stromů. Souběžně s animací zobrazuje stručný pseudokód aktuálně prováděné operace. Dále umožňuje *Opakovat poslední operaci* a *Generování náhodných stromů*.

Text samotné práce se dělí na dvě části: *Teoretickou část*, ve které se zabývám teorií vybraných stromů a *Programovou část*, která popisuje samotnou implementaci a funkcionalitu programu.

2 Stromy

V kapitole jsou vysvětleny základní pojmy, které jsou nezbytné k pochopení vlastností *stromů* obsažených v aplikaci. V podkapitolách jsou osvětleny principy pro tvorbu a následnou práci s konkrétními *binárními vyhledávacími stromy*. Využitím těchto principů byl naprogramován tento výukový nástroj.

Definice 1 (Strom)
Strom je neorientovaný [1] souvislý [2] graf bez kružnic [3] [1].



Obrázek 1: Příklady neorientovaných stromů

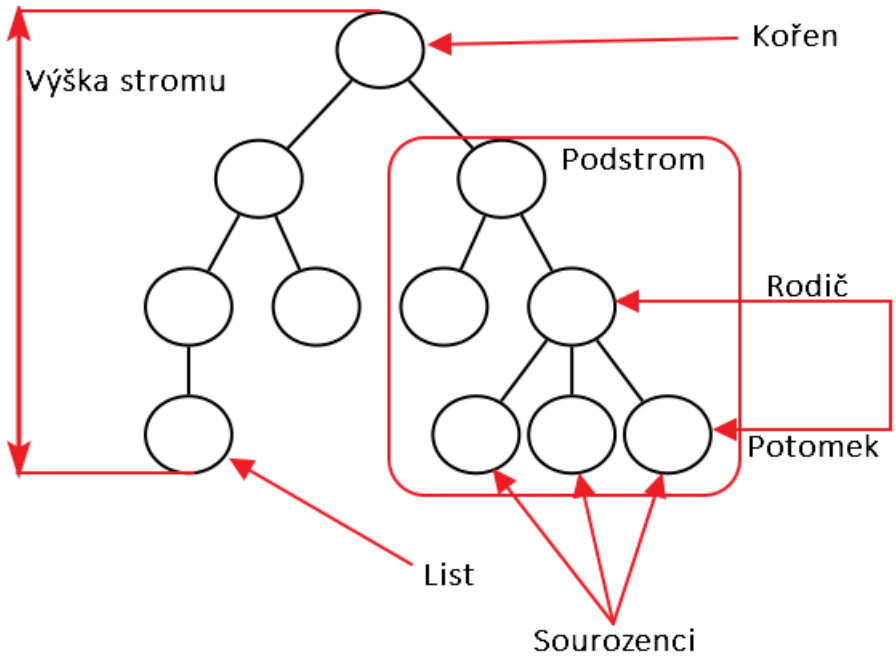
Strom je datová struktura, která představuje stromovou strukturu propojených *uzlů* [4]. Uzly jsou mezi sebou vzájemně spojeny pomocí *hran* [5]. Strom složený z uzlů U má $|U - 1|$ hran.

Definice 2 (Kořenový strom)
Kořenový strom je strom, ve kterém je vybrán jeden vrchol (kořen). Může to být kterýkoliv vrchol. Bývá to ale vrchol, který je v nějakém smyslu na vrcholu hierarchie objektů, která je stromem reprezentována. [1]

[1] Mezi každými dvěma vrcholy existuje právě jedna cesta.
[2] Vynecháním libovolné hrany vznikne nesouvislý graf.
[3] Přidáním jakékoli hrany vznikne graf s kružnicí.
[4] Prvek obsahující hodnotu.
[5] Představuje cestu mezi spojenými uzly.

Důležité pojmy:

- **Uzel** – Jednoduše jakýkoliv prvek stromu.
- **Kořen** – Jeden konkrétní uzel, který se nachází na vrcholu stromu. Pouze tento uzel nemá *rodiče*.
- **Potomek, následník** – Uzel, který je hranou přímo připojen k jinému uzlu, cestou od kořene.
- **Rodič, předchůdce** – Uzel, který má alespoň jednoho potomka.
- **Sourozenci** – Skupina uzlů, které mají stejného rodiče.
- **Podstrom** – Část stromu, která je úplným stromem, s tím, že kořen tohoto podstromu má svého rodiče.
- **Koncový uzel, list** – Uzel bez potomků.
- **Výška stromu** – Nejdelší délka cesty od kořene k uzlu.



Obrázek 2: Popis stromu

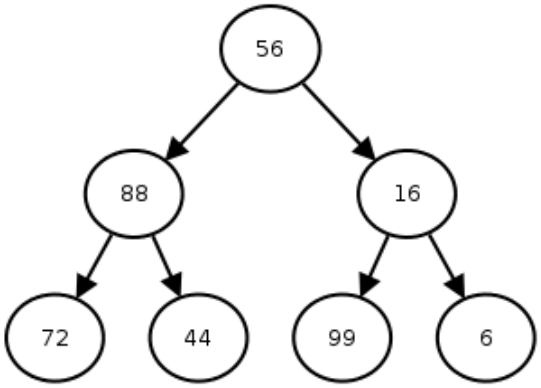
Definice 3 (m-ární stromy)
Kořenový strom se nazývá *m-ární*, právě když každý jeho vrchol má nejvýše m potomků. 2-ární strom se nazývá binární. Kořenový strom se nazývá úplný *m-ární*, právě když každý jeho vrchol nemá buď žádného nebo má právě m potomků.[2]

2.1 Binární strom

Definice 4 (Binární strom)
Binární strom je typ kořenových stromů, ve kterém každý obsažený uzel má maximálně 2 potomky.

Každý uzel obsahuje tyto vlastnosti:

- **Klíč** – Hodnota uložená v uzlu.
- **Ukazatel na levého potomka**
- **Ukazatel na pravého potomka**
- **Ukazatel na jednoho rodiče** – Tento ukazatel není povinný.



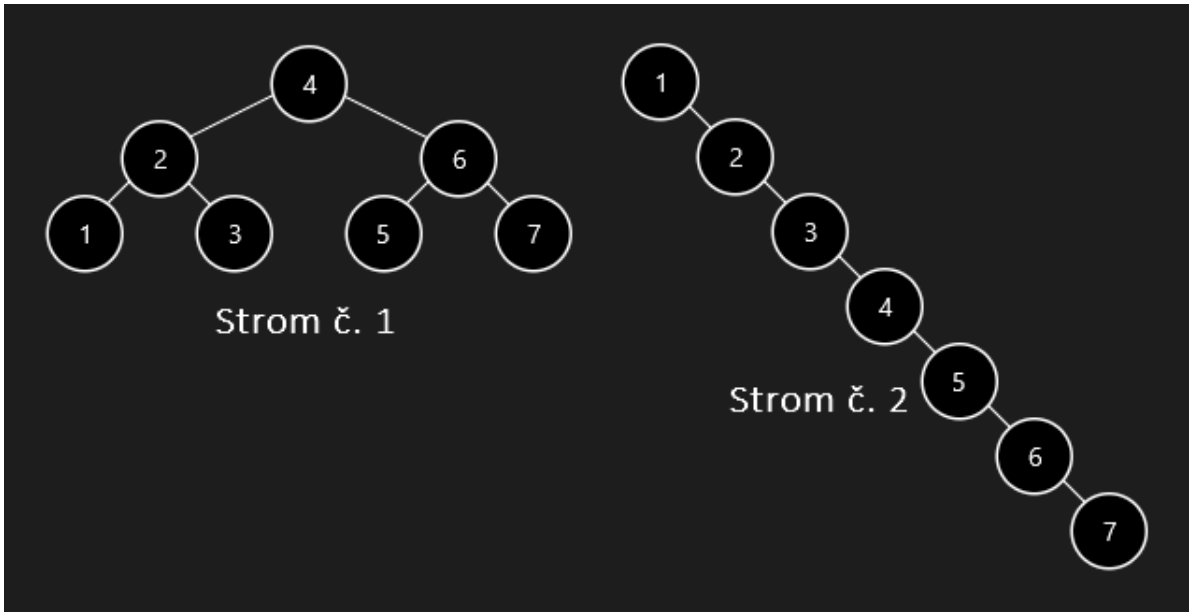
Obrázek 3: Binární strom

2.2 Binární vyhledávací strom

Binární *vyhledávací* strom (zkratka BVS), je speciální typ binárního stromu, kde platí následující:

- **Každý *pravý* potomek P rodiče R má vyšší hodnotu h než jeho rodič.** Platí tedy: $P.h > R.h \Rightarrow$ Pravý podstrom uzlu R obsahuje pouze uzly, které mají vyšší hodnotu než uzel R.
- **Každý *levý* potomek L rodiče R má nižší hodnotu h než jeho rodič.** Platí tedy: $P.h > L.h \Rightarrow$ Levý podstrom uzlu R obsahuje pouze uzly, které mají nižší hodnotu než uzel R.
- **Ve stromě se nenachází dva uzly se stejnou hodnotou.**

Toto uspořádání uzlů v BVS usnadňuje vyhledávání. Operace nad BVS stromem s výškou h mají časovou složitost $\theta(h)$. V nejhorším případě může mít BVS výšku rovnu n - 1, kde n je počet uzlů. Oba případy jsou zobrazeny na obrázku 4.



Obrázek 4: Rozdílné výšky stromu

2.2.1 Vyhledávání

Operace *vyhledávání* patří k nejčastěji používané operaci s BVS. Při *vyhledávání* je potřeba zadat hodnotu x, kterou chceme vyhledat. Postupně dochází k porovnávání hodnot uzlů s x. Výsledkem vyhledávání je buď uzel, který obsahuje hodnotu x nebo takový uzel neexistuje.

Přesný postup vyhledávání:

1. **krok** – počáteční
   Na začátku vyhledávání je třeba určit aktuální uzel, který označíme u. Hledanou hodnotu označíme x.
   V tomto kroku u bude kořen stromu, pokud strom nemá kořen, hodnota x je nenalezena a tím vyhledávání končí.

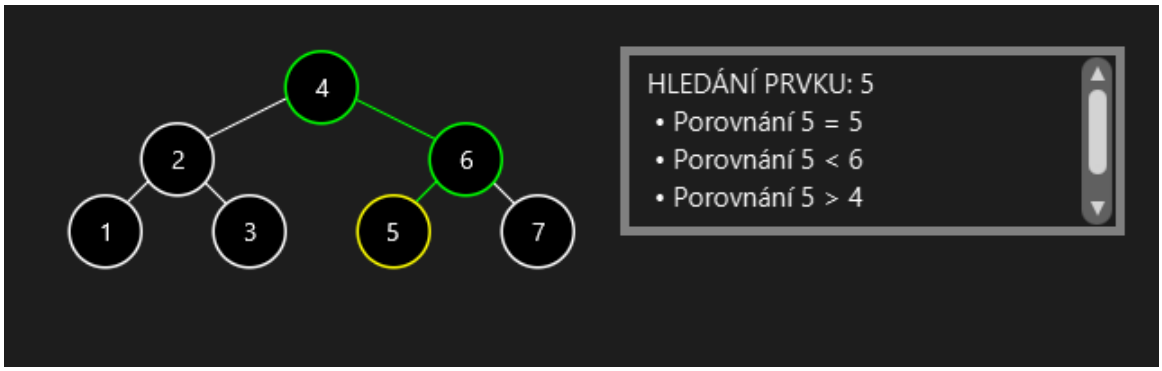
2. **krok** – průběžný
   Zde dochází k porovnávání x a hodnoty aktuálního uzlu u. Hodnotu u označíme $u.h$.
   Pokud je u prázdný, vyhledávání končí neúspěchem.

   Při porovnávání mohou nastat tyto možnosti:

   - $x > u.h$
     V tomto případě jako u nastavíme pravého potomka u. A znovu provedeme 2. krok.

- $x < u.h$
  V tomto případě jako u nastavíme levého potomka u. A znovu provedeme 2. krok.
- $x = u.h$
  Hledaná hodnota x byla nalezena v u. Vyhledávání tedy končí.

Na obrázku 5 je zvýrazněna cesta průchodů stromem při vyhledávání uzlu s hodnotou 5. Na obrázku je i zaznamenána historie porovnávání.



Obrázek 5: Postup vyhledávání hodnoty 5

Zdrojový kód vyhledávání v jazyku Java:

```
1  Node search(int value, Node node) { //value je hledaná hodnota, node
       je při prvním volání kořen
2    if (node == null) {
3      return null; //uzel nebyl nalezen
4    }
5
6    if (value > node.getValue()) { //pokud je hledaná hodnota vyšší ne
       ž má aktuální uzel
7      search(value, node.getRight()); //nastavím aktuální uzel na pravé
         ho potomka
8    } else if (value < node.getValue()) { //pokud je hledaná hodnota
       vyšší než má aktuální uzel
9      search(value, node.getLeft()); //nastavím aktuální uzel na levého
           potomka
10   } else { //pokud není vyšší ani nižší, tak se musí rovnat
11     return node; //vrátím nalezený uzel
12   }
13 }
```

Zdrojový kód 1: search

2.2.2 Vkládání

Při *vkládání* zadané hodnoty x je nejprve nutné prohledat strom, jestli se zde x již nenachází. Pokud je nalezen uzel s hodnotou x, je výsledkem operace nalezený uzel. V případě, že daný strom tento uzel neobsahuje, je jako potomek posledního prohledávaného uzlu vložen nový uzel s hodnotou x.

Přesný postup vkládání:

1. **krok** – počáteční
   Na začátku vkládání je třeba určit aktuální uzel, který označíme u. Vkládanou hodnotu označíme x.
   V tomto kroku u bude kořen stromu. Pokud strom nemá kořen, vytvoříme nový uzel s hodnotou x a ten vložíme do stromu. Uzel se stane kořenem stromu, čímž vkládání končí.

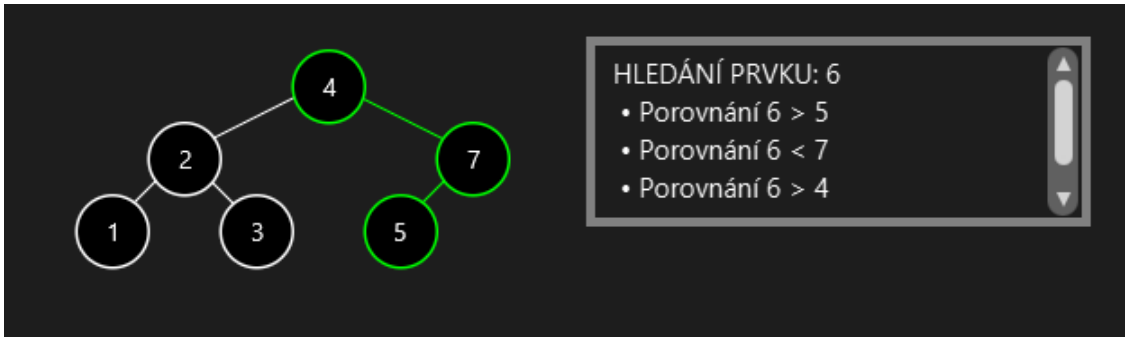
2. **krok** – vyhledávání, vkládání
   Před vkládáním je nejprve nutno ověřit, zde se ve stromu nenachází uzel s hodnotou x.

   Vyhledávání může dopadnout těmito způsoby:

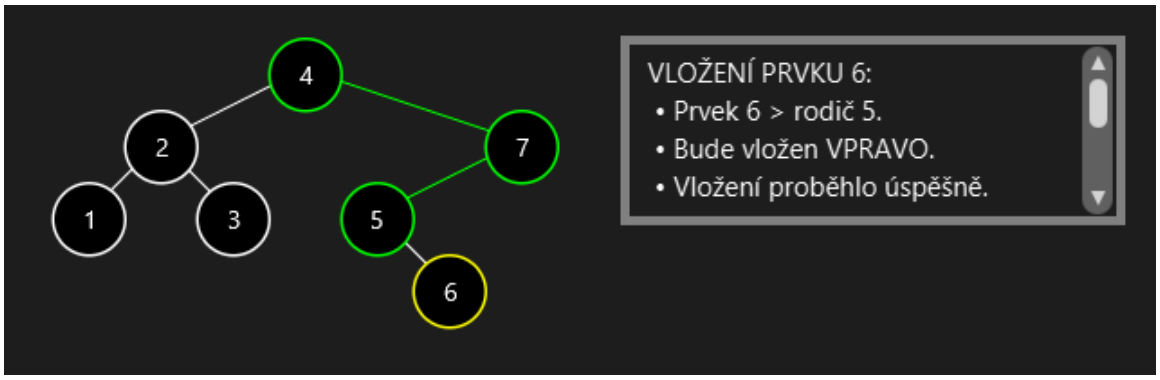
   - Prvek byl nalezen.
     Pokud byl uzel s hodnotou x nalezen, vkládání končí neúspěchem.
   - Prvek nebyl nalezen, přičemž platí $x > u.h$[6].
     Vytvoříme nový uzel s hodnotou x a vložíme jako pravého potomka u[7].
   - Prvek nebyl nalezen, přičemž platí $x < u.h$[6].
     Vytvoříme nový uzel s hodnotou x a vložíme jako pravého potomka u[7]

Na obrázku 6 je zvýrazněna cesta průchodů stromem při vyhledávání uzlu s hodnotou 6. Samotné vložení je na obrázku 7.

[6]Hodnota posledního navštíveného uzlu při hledání.
[7]Poslední navštívený uzel při hledání.



Obrázek 6: Postup vkládání hodnoty 6 - hledání



Obrázek 7: Postup vkládání hodnoty 6 - vložení

Zdrojový kód vkládání v jazyku Java:

Před samotnou funkcí `insert` je třeba vytvořit třídu `Result`, která bude dále použita:

```
1 Class Result() {
2   private Node node; //poslední navštívený uzel (hledaný/rodič)
3   private boolean isFind; //parametr pro určení zda byl uzel nalezen
4
5   public Result(Node node, boolean isFind) { //konstruktor
6     ...
7   }
8   ...
9 }
```

Zdrojový kód 2: Result - Třída

Dále je potřeba trochu poupravit již známou funkci `search` 1:

```
1  Result search(int value, Node node) { //value je hledaná hodnota,
       node je při prvním volání kořen
2    if (value > node.getValue()) {
3      if (node.getRight() != null) {
4        search(value, node.getRight()); //pokud má pravého potomka
5      } else {
6        return new Result(node, false); //pokud nemá pravého potomka,
             node = rodič vkládaného
7      }
8    } else if (value < node.getValue()) {
9      if (node.getLeft() != null) {
10       search(value, node.getLeft()); //pokud má levého potomka
11     } else {
12       return new Result(node, false); //pokud nemá levého potomka,
             node = rodič vkládaného
13     }
14   } else {
15     return new Result(node, true);
16   }
17 }
```

Zdrojový kód 3: search - úprava

Nyní funkce `insert`:

```
1  Node insert(int value) {
2    Result result = search(value); //nejprve vyhledáme value
3    if (result.isFind()) { //pokud byl uzel s danou hodnotou nalezen
4      return result.getNode();
5    } else { //dále vkládáme nově vytvořený uzel:
6      if (value > result.getNode().getValue()) {
7        result.getNode().setRight(new Node(value));//bude pravý potomek
8      } else {
9        result.getNode().setLeft(new Node(value));//bude levý potomek
10     }
11   }
12   return null;
```

Zdrojový kód 4: insert

2.2.3 Odebírání

Při *odebírání* zadané hodnoty x je stejně jako u vkládání nejprve nutné prohledat strom, zda se odebíraný uzel s hodnotou x ve stromě nachází. Pokud je uzel nalezen, je následně smazán a struktura stromu případně upravena.

Přesný postup odebírání:

1. **krok** – počáteční
   Na začátku odebírání je třeba určit aktuální uzel, který označíme u. Odebíranou hodnotu označíme x.
   V tomto kroku u bude kořen stromu. Pokud strom nemá kořen odebírání končí, uzel s hodnotou x nebyl nalezen.

2. **krok** – vyhledávání
   Před odebíráním je nejprve nutno uzel s hodnotou x vyhledat.

   Vyhledávání může dopadnout těmito způsoby:

   - Prvek nebyl nalezen.
     Pokud se uzel s hodnotou x ve stromě nenachází, odebírání končí neúspěchem.
   - Prvek byl nalezen.
     Nyní můžeme nalezený uzel u odebrat.

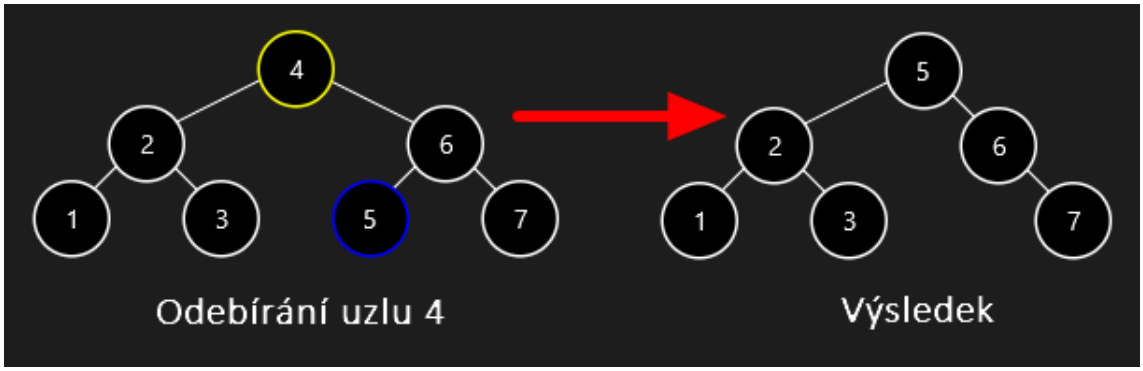
3. **krok** – odebírání

   Odebírání u má tyto možnosti:

   - Pokud u je list[8].
     List u může být odebrán.
   - u má jednoho potomka.
     u bude nahrazen podstromem potomka.
   - u má dva potomky.
     - u bude nahrazen *nejlevějším* prvkem z *pravého* podstromu.
     - u bude nahrazen *nejpravějším* prvkem z *levého* podstromu.

V následujícím obrázku 8 je ukázka mazání uzlu s hodnotou 4, který je následně nahrazen uzlem s hodnotou 5, který je *nejlevější* prvek z *pravého* podstromu.

[8] Nemá žádné potomky.



Obrázek 8: Postup odebírání hodnoty 4

Zdrojový kód[9] odebírání v jazyku Java:

[9]Tento kód je pouze zjednodušená implementace k lepšímu pochopení odebírání.

```
1  boolean delete(int value) {
2    Node removeNode = result.getNode();
3    Node helpNode; //pomocná proměnná
4
5    Result result = search(value); //vyhledám hodnotu
6
7    if (!result.isFind()) { //pokud ho nenajdu
8      return false;
9    }
10
11   if ((removedNode.getLeft() != null) && (removedNode.getRight() !=
         null)) { //Pokud má dva potomky
12     helpNode = removeNode.getRight(); //dosadím pravého potomka
13
14     while (helpNode.getLeft() != null) { //a hledám nejlevějšího
15       helpNode = helpNode.getLeft();
16     }
17
18     removeNode.setNode(helpNode);
19   } else if (removedNode.getLeft() != null) { //pouze levý potomek
20     removeNode.setNode(removedNode.getLeft());
21   } else if (removedNode.getRight() != null) { //pouze pravý potomek
22     removeNode.setNode(removedNode.getRight());
23   } else { //pokud je to list
24     removeNode.delete(); //odstraním list ze stromu
25   }
26
27   return true;
28 }
```

Zdrojový kód 5: delete

2.3 AVL strom

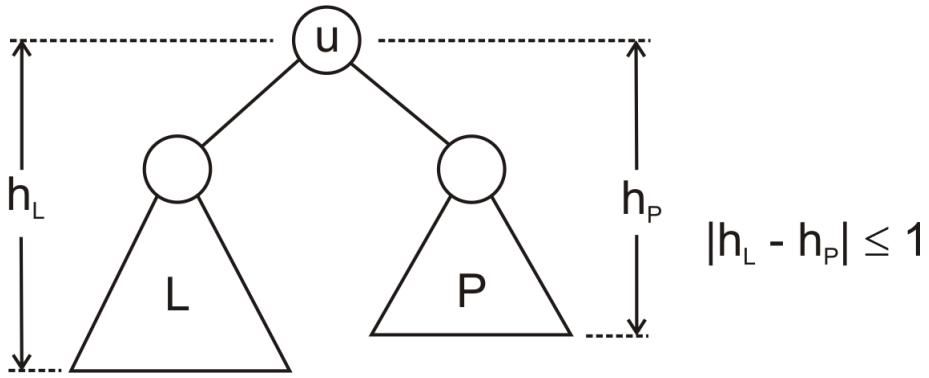
Určitým způsobem by se dalo tvrdit, že *AVL strom* je binární vyhledávací strom, který má ale jednu zásadní odlišnost. Složitost všech operací u BVS je závislá na výšce stromu, je tedy žádoucí udržovat výšku stromu co nejnižší. *AVL strom* je tedy vyvážený binární vyhledávací strom.

Definice 5 (Vyvážený strom)
Strom je vyvážený tehdy a jen tehdy, je-li rozdíl výšek každého uzlu nejvýše 1.[6][4]

Kvůli udržování vyváženosti stromu mají operace *vkládání* a *odebírání* složitost $\theta(log(n))$.

V následujícím obrázku 9 je předchozí definice o vyváženém stromu názorně vysvětlena. Nejprve je třeba zavést pojmy:

- u – aktuální uzel.
- L – levý podstrom u.
- P – pravý podstrom u.
- h_L – výška levého podstromu.
- h_P – výška pravého podstrom.

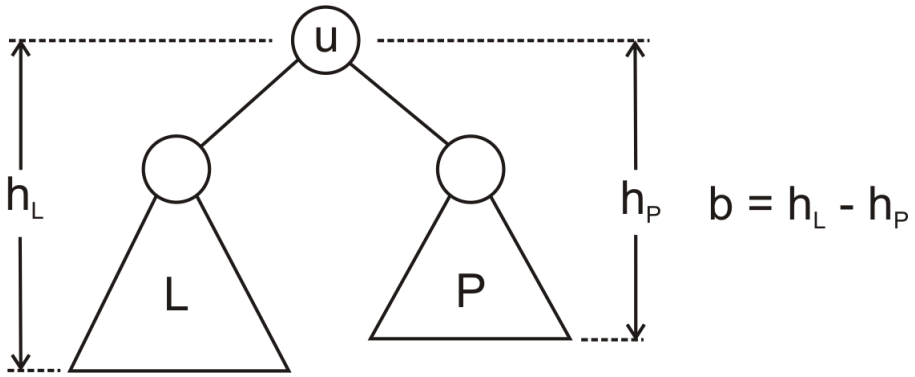


Obrázek 9: Vyvážený strom

Faktor vyvážení

Pro kontrolu dodržení pravidla o vyvážení, je třeba pro každý uzel zavést novou vlastnost *faktor vyvážení uzlu*. Tuto vlastnost budeme značit b (anglicky balance). b obsahuje informaci o aktuálním vyvážení daného podstromu.

Nabývá hodnot <-2, 2>, přičemž uzel je vyvážený pokud jeho faktor b je 1, 0 nebo -1. Při operaci přidávání nebo odebírání, může být b 2 nebo -2, pak je ale nutné provést transformaci, která dosáhne vyvážení daného uzlu.

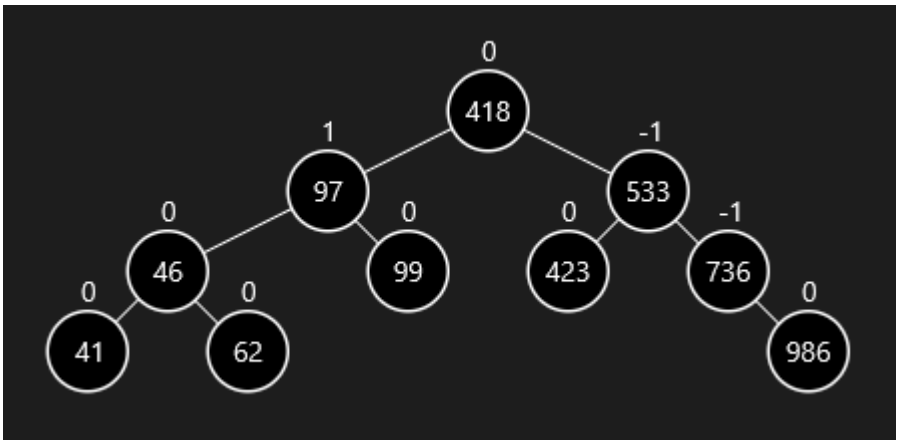


Obrázek 10: Výpočet faktoru vyvážení pro uzel u

Zdrojový kód rekurzivního výpočtu faktoru v jazyku Java:

```
1  int computeFactor() {
2    int lHeight = 0; //výška levého podstromu
3    int rHeight = 0; //výška pravého podstromu
4
5    if (left != null) { //pokud má levý podstrom
6      lHeight = left.computeFactor();
7    }
8
9    if (right != null) { //pokud má pravý podstrom
10     rHeight = right.computeFactor();
11   }
12
13   factor = lHeight - rHeight;
14
15   return Math.max(rHeight, lHeight) + 1;
16 }
```

Zdrojový kód 6: computeFactor



Obrázek 11: Ukázka vyváženého AVL stromu

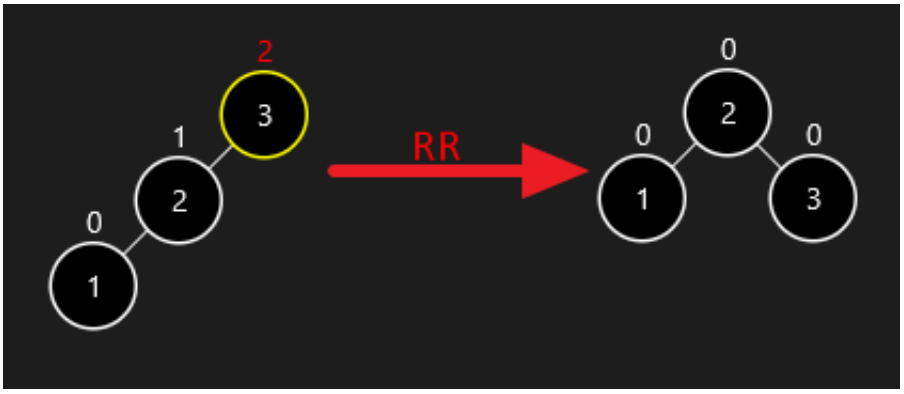
Každý uzel ALV stromu nyní obsahuje tyto vlastnosti:

- **Klíč** – Hodnota uložená v uzlu.
- **Faktor vyvážení** – Faktor vyvážení daného uzlu.
- **Ukazatel na levého potomka**
- **Ukazatel na pravého potomka**
- **Ukazatel na jednoho rodiče**

2.3.1 Rotace

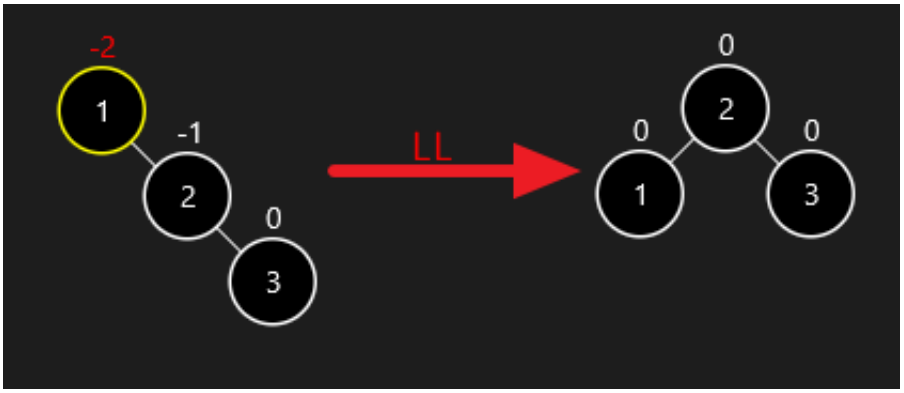
K obnově vyváženosti uzlů se používají *rotace*. *Rotace* pouze změní ukazatele uzlů v nevyvážené části stromu, aby došlo k opětovnému vyvážení. V AVL stromu se k vyvažování podle typu nevyvážené části stromu, používají tyto rotace:

Jednoduchá rotace RR
Nevyvážený uzel má $b = 2$, jeho *levý potomek* má $b = 0$ nebo $b = 1$.



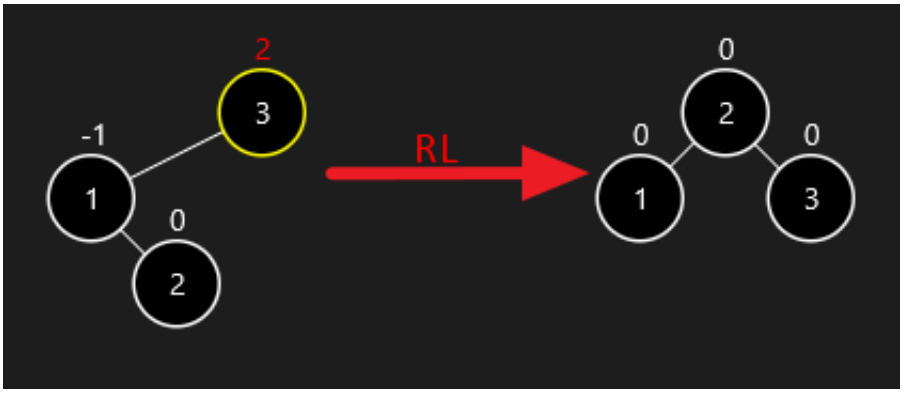
Obrázek 12: Rotace RR

Jednoduchá rotace LL
Nevyvážený uzel má $b = -2$, jeho *pravý potomek* má $b = -1$ nebo $b = 0$.



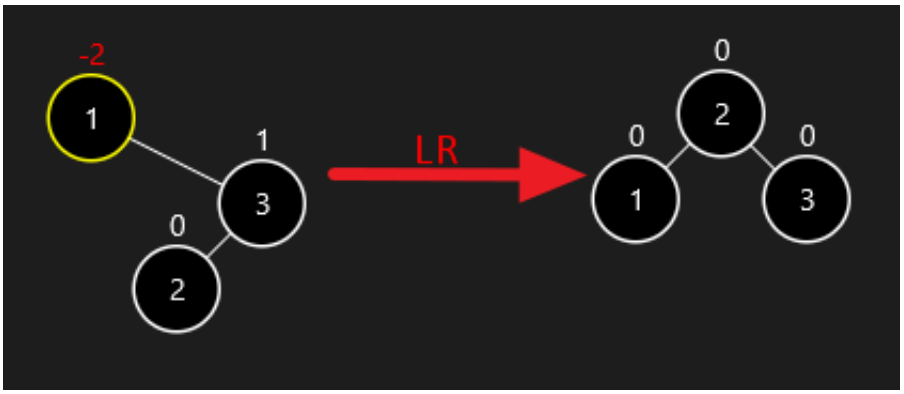
Obrázek 13: Rotace LL

Dvojitá rotace RL
Nevyvážený uzel má $b = 2$, jeho *levý potomek* má $b = 1$.



Obrázek 14: Rotace RL

Dvojitá rotace LR
Nevyvážený uzel má $b = -2$, jeho *pravý potomek* má $b = 1$.



Obrázek 15: Rotace LR

2.3.2 Vyhledávání

Vyhledávání u AVL stromu se nijak neliší od toho v BVS.

2.3.3 Vkládání

Vkládání je založeno na *vkládání* u BVS. Navíc je zde přidán krok aktualizace faktoru vyvážení, kde od nově vloženého uzlu postupně u určitých uzlů přepočítáváme b. Pokud se strom, kvůli vložení nového uzlu stává nevyváženým, tak provedeme jednu z rotací, čímž daný strom vyvážíme.

Přesný postup vkládání:

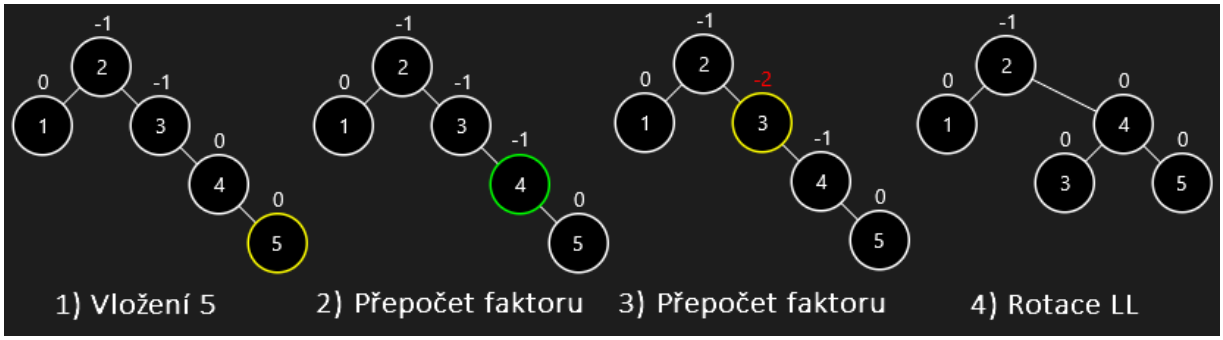
1. **krok** – počáteční
   Stejně jako u BVS.

2. **krok** – vyhledávání, vkládání
   Novému uzlu po vložení nastavíme faktor vyvážení na 0 a u nastavíme na předchůdce nově vloženého uzlu.
   Pokud nově vložený uzel je kořen vkládání úspěšně končí.

3. **krok** – výpočet faktoru vyvážení
   V tomto kroku v aktuálním uzlu u aktualizujeme faktor vyvážení b.

   Po aktualizace $u.b$ mohou nastat tyto případy:

   - $u.b = 1$ nebo $u.b = -1$
     Pokud u je kořen, vkládání úspěšně končí. Jinak u nastavíme na rodiče u a znovu opakujeme krok 3.
   - $u.b = 2$ nebo $u.b = -2$
     Provede se příslušná rotace.
     Pokud po provedení rotace je $u.b = 0$ nebo je nyní u kořen, vkládání úspěšně končí. Jinak u nastavíme na předchůdce u a znovu opakujeme krok 3.
   - $u.b = 0$
     Vkládaní úspěšně končí.



Obrázek 16: Postup vkládání hodnoty 5

2.3.4 Odebírání

Odebírání založeno na stejnojmenné operaci u BVS. Navíc je zde jako u *vkládání* přidán krok aktualizace faktoru vyvážení, kde od odstraněného uzlu postupně u určitých uzlů přepočítáváme b. Pokud se strom, kvůli odebrání uzlu stává nevyváženým, tak provedeme jednu z rotací, čímž daný strom vyvážíme.

Přesný postup odebírání:

1. **krok** – počáteční
   Stejně jako u BVS.

2. **krok** – vyhledávání
   Stejně jako u BVS.

3. **krok** – odebírání

   Odebírání u má tyto možnosti:

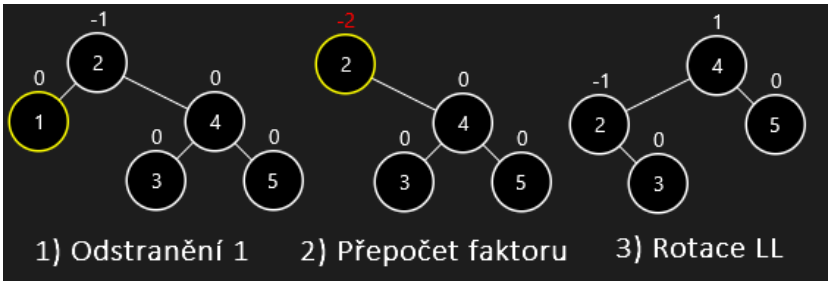
   - Pokud u je list.
     List u může být odebrán.
     Pokud byl u kořen, odebírání úspěšně končí. Jinak nastavíme u na rodiče u.
   - u má jednoho potomka.
     u bude nahrazen podstromem potomka. Pokud byl u kořen, odebírání úspěšně končí. Jinak nastavíme u na rodiče u.
   - u má dva potomky.
     - u bude nahrazen *nejlevějším* prvkem z *pravého* podstromu, zároveň tento prvek získá faktor vyvážení z u.
     - u bude nahrazen *nejpravějším* prvkem z *levého* podstromu, zároveň tento prvek získá faktor vyvážení z u.

     Rodiče uzlu, který nahradil u uložíme do u.

4. **krok** – výpočet faktoru vyvážení
   V tomto kroku v aktuálním uzlu u aktualizujeme faktor vyvážení b.

   Po aktualizace $u.b$ mohou nastat tyto případy:

   - $u.b =< -1, 1 >$
     Pokud u je kořen, odebírání úspěšně končí. Jinak u nastavíme na rodiče u a znovu opakujeme krok 4.
   - $u.b = 2$ nebo $u.b = -2$
     Provede se příslušná rotace.
     Pokud po provedení rotace je $u.b = 0$ nebo je nyní u kořen, odebírání úspěšně končí. Jinak u nastavíme na předchůdce u a znovu opakujeme krok 4.



Obrázek 17: Postup odebírání hodnoty 1

2.4 Červeno-černý strom

Červeno-černý strom (zkratka ČČ)

Závěr

Závěr práce v „českém“ jazyce.

Conclusions

Thesis conclusions in “English”.

A První příloha

Text první přílohy

B Druhá příloha

Text druhé přílohy

C Obsah přiloženého CD/DVD

Na samotném konci textu práce je uveden stručný popis obsahu přiloženého CD/DVD, tj. jeho závazné adresářové struktury, důležitých souborů apod.

bin/
: Instalátor INSTALATOR programu, popř. program PROGRAM, spustitelné přímo z CD/DVD. / Kompletní adresářová struktura webové aplikace WEBOVKA (v ZIP archivu) pro zkopírování na webový server. Adresář obsahuje i všechny runtime knihovny a další soubory potřebné pro bezproblémový běh instalátoru a programu z CD/DVD / pro bezproblémový provoz webové aplikace na webovém serveru.

doc/
: Text práce ve formátu PDF, vytvořený s použitím závazného stylu KI PřF UP v Olomouci pro závěrečné práce, včetně všech příloh, a všechny soubory potřebné pro bezproblémové vygenerování PDF dokumentu textu (v ZIP archivu), tj. zdrojový text textu, vložené obrázky, apod.

src/
: Kompletní zdrojové texty programu PROGRAM / webové aplikace WEBOVKA se všemi potřebnými (příp. převzatými) zdrojovými texty, knihovnami a dalšími soubory potřebnými pro bezproblémové vytvoření spustitelných verzí programu / adresářové struktury pro zkopírování na webový server.

readme.txt
: Instrukce pro instalaci a spuštění programu PROGRAM, včetně všech požadavků pro jeho bezproblémový provoz. / Instrukce pro nasazení webové aplikace WEBOVKA na webový server, včetně všech požadavků pro její bezproblémový provoz, a webová adresa, na které je aplikace nasazena pro účel testování při tvorbě posudků práce a pro účel obhajoby práce.

Navíc CD/DVD obsahuje:

data/
: Ukázková a testovací data použitá v práci a pro potřeby testování práce při tvorbě posudků a obhajoby práce.

`install/`
: Instalátory aplikací, runtime knihoven a jiných souborů potřebných pro provoz programu Program / webové aplikace Webovka, které nejsou standardní součástí operačního systému určeného pro běh programu / provoz webové aplikace.

`literature/`
: Vybrané položky bibliografie, příp. jiná užitečná literatura vztahující se k práci.

U veškerých cizích převzatých materiálů obsažených na CD/DVD jejich zahrnutí dovolují podmínky pro jejich šíření nebo přiložený souhlas držitele copyrightu. Pro všechny použité (a citované) materiály, u kterých toto není splněno a nejsou tak obsaženy na CD/DVD, je uveden jejich zdroj (např. webová adresa) v bibliografii nebo textu práce nebo v souboru `readme.txt`.

Literatura

[1] BĚLOHLÁVEK, Radim. Algoritmická matematika 2 - část 1 [online]. 2012-05-15. [cit. 2018-07-07]. Dostupné z: http://belohlavek.inf.upol.cz/vyuka/algoritmicka-matematika-2-1.pdf

[2] BĚLOHLÁVEK, Radim; VYCHODIL, Vilém. Diskrétní matematika pro informatiky II [online]. 2010-10-16. [cit. 2018-07-07]. Dostupné z: http://belohlavek.inf.upol.cz/vyuka/dm2.pdf

[3] VEČERKA, Arnošt. Studijní materiály ALM-2

[4] DVORSKÝ, Jiří. Algoritmy I.[online]. 2007-02-27. [cit. 2018-07-07]. Dostupné z: http://www.cs.vsb.cz/dvorsky/Download/SkriptaAlgoritmy/Algoritmy.pdf

[5] BINARY SEARCH TREES Dostupné z: http://cs.umw.edu/ finlayson/class/fall12/cpsc230/not binary-search-trees.html

[6] ADELSON-VELSKII, G. M.; LANDIS, E. M. An algorithm for the or- ganization of information. Soviet Mathematics Doklady, 3:1259–1263, 1962. [online]. [cit. 2018-07-07] Dostupné z: http://professor.ufabc.edu.br/ jesus.mena/courses/mc3305-2q-2015/AED2-10-avl-paper.pdf

[7] GALLES, David. Data Structure Visualizations [online]. [cit. 2018-07-07] Dostupné z: https://www.cs.usfca.edu/ galles/visualization/Algorithms.html