Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

BAKALÁŘSKÁ PRÁCE

Program pro výuku binárních vyhledávacích stromů

2015

Martin Šebesta

Vedoucí práce: RNDr. Arnošt Večerka

Studijní obor: Aplikovaná informatika, prezenční forma

Bibliografické údaje

Autor:	Martin Šebesta
Název práce:	Program pro výuku binárních vyhledávacích stromů
Typ práce:	bakalářská práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2015
Studijní obor:	Aplikovaná informatika, prezenční forma
Vedoucí práce:	RNDr. Arnošt Večerka
Počet stran:	53
Přílohy:	1 CD/DVD
Jazyk práce:	český

Bibliograhic info

Author:	Martin Šebesta
Title:	Binary search tree demonstration program
Thesis type:	bachelor thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2015
Study field:	Applied Computer Science, full-time form
Supervisor:	RNDr. Arnošt Večerka
Page count:	53
Supplements:	1 CD/DVD
Thesis language:	Czech

Anotace

Výukový program Trees zobrazuje uživateli průběhy operací přidávání, odebírání a vyhledávání uzlů v AVL a červeno–černých stromech. Dále program porovnává časy operací těchto stromů a zobrazuje výsledky pomocí grafů.

Synopsis

Trees is an educational program which depicts progresses of particular operations such as inserting, deleting and searching of nodes in both AVL and Red-Black trees to a user. Further, the program compares the time operations of these trees and demostrates their results by grafs.

Klíčová slova: AVL stromy; červeno-černé stromy; Vyhledávaní ve stromech; Přidávání do stromů; Odebírání ze stromů;

Keywords: AVL trees; Red–Black trees; Search in trees; Insert to trees ; Delete from trees

Tímto bych rád poděkoval svému vedoucímu práce RNDr. Arnoštu Večerkovi za odbornou pomoc, cenné rady a poskytnuté materiály, které mi pomohly při zpracování bakalářské práce.

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 tabulek

Seznam vět

Seznam zdrojových kódů

1 Úvod

Tento program vznikl pro výuku binárních vyhledávacích stromů studentům. Zahrnuje AVL a červeno-černé stromy. Program demonstruje průběh operace *Vyhledávání, Přidávání* a *Odebírání* uzlů ze stromu. Při *Přidávání* a *Odebírání* program znázorňuje, jak probíhají transformace a obnovení vlastností stromů. Druhá část programu vzájemně srovnává rychlosti těchto operací v červeno-černých a AVL stromech. Výsledkem této části je graf, který uživateli dává vizuální představu o časovém rozdílu.

2 Kořenové stromy

Tato kapitola slouží k vysvětlení základních pojmů, které se vyskytuji při práci s binárními stromy. Jsou zde rozebrány principy operací nad binárními vyhledávacími stromy, AVL stromy a červeno-černými stromy. Tyto principy byly naprogramovány ve výsledném programu. Kapitola vychází z publikací [1], [2].

Věta 1 (Kořenový strom)
Kořenový strom je volný strom [1]*, ve kterém je vybrán speciální uzel, tzv. kořen [1].*

Důležité pojmy, které se vyskytují v textu:

- **úroveň uzlu *x*** je délka cesty od kořene do x
- jestliže poslední hrana na cestě [2] z kořene r do uzlu x je hrana (y, x), potom se uzel y nazývá **rodič** uzlu x a uzel x je **potomek** uzlu y
- **list** neboli **externí uzel** je uzel bez potomků
- **sourozenec** (anglicky sibling) je uzel, který má s dalším uzlem stejného rodiče
- **strýc** uzlu x je uzel, který je sourozenec rodiče uzlu x

Definice 2 (m–ární strom)
*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 stromy

Definice 3 (Binární strom)
Binární strom je struktura definovaná nad konečnou množinou uzlů, která:

- *neobsahuje žádný uzel*
- *je složena ze tří disjunktních množin uzlů: kořene, binárního stromu zvaného levý podstrom a binárního stromu tzv. pravého podstromu*

[1] Souvislý, acyklický, neorientovaný graf.
[2] Cesta je posloupnost po sobě jdoucích vrcholů, které jsou spojeny hranou.

Prázdný strom – takový strom, který neobsahuje žádné uzly.

Levý podstrom – pokud levý podstrom není prázdný, jeho kořen je levým potomkem kořene celého stromu.

Pravý podstrom – pokud pravý podstrom není prázdný, jeho kořen je pravým potomkem kořene celého stromu.

Uzel má maximálně 2 potomky, struktura uzlu obsahuje tyto vlastnosti:

- **klíč** – hodnota uložená v uzlu
- ukazatele na **levého** a **pravého** potomka
- ukazatel na **rodiče**

Kořen stromu je jediný uzel bez rodiče.

Vnitřní uzel je každý uzel který není list.

Průchody stromem slouží k průchodu všech uzlů x ve stromu T a pro každý uzel x může provést operaci (například výpis hodnoty uzlu). Uzly jsou navštěvovány v určitém pořadí. Rozeznáváme dva typy průchodů:

- **Do hloubky**
  - **V pořadí** – (anglicky *Inorder*) vnitřní průchod; nejdříve navštívíme levý podstrom, poté uzel a pravý podstrom
  - **Nejprve podstromy** – (anglicky *Postorder*) zpětný průchod; nejdříve navštívíme levý a pravý podstrom, poté uzel
  - **Nejprve uzel** – (anglicky *Preorder*) přímý průchod; nejdříve navštívíme uzel, poté levý a pravý podstrom
- **Do šířky**
  - **Průchod do šířky** – (anglicky *Breadth–first*); pořadí uzlů je dáno hloubkou stromu, začíná se v kořenu, poté se začíná zleva a prochází se strom po vrstvách

Ukázka pseudokódu operace *Inorder*, kde x je ukazatel na kořen stromu:

Algorithm 1 Inorder(x)

```
if x ≠ NIL then
  Inorder(levy[x]
  vytiskni uzel x
  Inorder(pravy[x]
end if
```

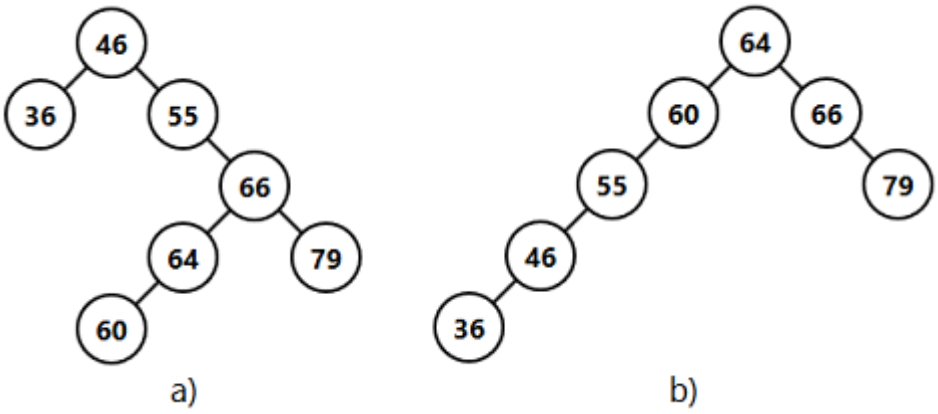
2.2 Binární vyhledávací stromy

Binární vyhledávající stromy (dále jen BVS) jsou speciální binární stromy, kde jsou uzly uspořádány následovně:

Definice 4 (Binární vyhledávací stromy)
Nechť x je uzel v binárním stromu. Jestliže y je z levého podstromu uzlu x, potom klíč[y] ≤ klíč[x]. Jestliže y je z pravého podstromu uzlu x, potom klíč[x] ≤ klíč[y].

Struktura BVS uzlů je stejná jako u binárních stromů.

Základní operace nad BVS mají časovou složitost $\theta(h)$, kde h je výška stromu. Výška BVS je hlavní nevýhodou, v nejhorším případě má BVS s n uzly hloubku (n - 1). Příklad takového stromu na obr. 1. Vidíme zde dva BVS stromy složené ze stejných uzlů, avšak vypadají různě.



Obrázek 1: Binární vyhledávací stromy

2.2.1 Vyhledávání

Operace *Vyhledávání* hledá ve stromu T hodnotu k. Výsledkem je buď ukazatel na uzel x, jehož hodnota je k, nebo hodnota *NIL*, která znamená, že hodnota k není ve stromu T. Operace *Vyhledávání* je nejpoužívanější operací při práci s binárními stromy. Podobné operace, jako jsou operace nalezení *Maxima*[3] nebo *Minima*[4] ve stromu, mají časovou složitost $\theta(h)$, kde h je výška stromu.

[3]Uzel s maximální hodnotou klíče.
[4]Uzel s minimální hodnotou klíče.

Ukázka pseudokódu operace *Minimum*, kde x je ukazatel na kořen stromu.

Algorithm 2 Minimum(x)

```
if x ≠ NIL then
  while pravy[x] ≠ NIL do
    x ← levy[x]
  end while
end if
return x
```

Vyhledávání ve stromu probíhá následovně:

- máme strom T s ukazatelem na kořen r a hledaný klíč k
- hledání začíná v kořenu r
- pokud kořen r je *NIL*, *Vyhledávání* končí
- v opačném případě porovnáváme klíč k s aktuálním klíčem kořene r aktuálního podstromu. Nastane jedna z následujících možností:
  - $k =$ klíč$[r]$ strom T obsahuje hledaný klíč k, výsledkem je ukazatel na uzel x, který obsahuje tento klíč k
  - $k <$ klíč$[r]$ klíč k se nachází v levém podstromu aktuálního kořene r, pokračujeme rekurzivně v levém podstromu
  - $k >$ klíč$[r]$ klíč k se nachází v pravém podstromu aktuálního kořene r, pokračujeme rekurzivně v pravém podstromu

Ukázka pseudokódu operace *Vyhledavani*, kde x je ukazatel na kořen stromu a *klic* je hledaný klíč.

Algorithm 3 Vyhledavani(x,*klic*)

```
if klic = klic[x] or x = NIL then
  return x
end if
if klic < klic[x] then
  return Vyhledavani(levy[x], klic)
else
  return Vyhledavani(pravy[x], klic)
end if
```

Následník uzlu x je uzel, který je minimem pravého podstromu uzlu x. Při průchodu *V pořadí* je to uzel, který byl navštíven hned po uzlu x.

Předchůdce uzlu x je uzel, který je maximem levého podstromu uzlu x. Při průchodu *V pořadí* je to uzel, který byl navštíven hned před uzlem x.

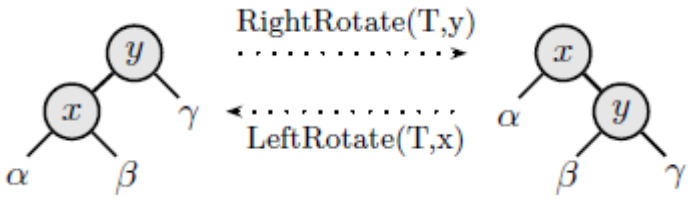
Ukázka pseudokódu operace *Naslednik*, kde x je ukazatel na kořen stromu.

Algorithm 4 Naslednik(x)

```
if x = NIL then
  return NIL
end if
if pravy[x] ≠ NIL then
  return Minimum(pravy[x])
end if
y ← rodic[x]
while y ≠ NIL and pravy[y] = x do
  x ← y
  y ← rodic[y]
end while
return y
```

2.2.2 Rotace

Pravá a **levá** rotace slouží ke znovuobnovení vlastností například u AVL a červeno–černých stromů (obr. 2). Používají se při operacích *Přidávání* nebo *Odebírání*, kde dochází ke změnám ve stromu. Rotace pouze mění ukazatele na jednotlivé podstromy určitých uzlů, ostatní vlastnosti zůstávají zachovány. Časová složitost rotace je $\theta(1)$, protože dochází jen k výměně ukazatelů. Při rotaci se zachovává pořadí klíčů ve stromu:
$klíče[\alpha] < klíč[x] < klíče[\beta] < klíč[y] < klíče[\gamma]$



Obrázek 2: Levá a pravá rotace u BVS

2.2.3 Přidávání

Přidání nového uzlu do BVS je podobné operaci *Vyhledávání*. Výsledkem operace *Přidávání* je BVS s kořenem r, který je změněný původní strom. Po *Přidávání* se nezmění vlastnosti BVS, nový uzel je vložen správně.

Operace *Přidávání* probíhá následovně:

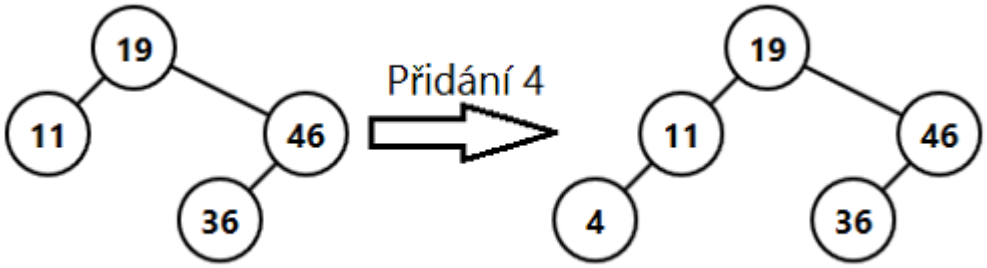
- Vstupem funkce je ukazatel na kořen r BVS a ukazatel na nový uzel x s klíčem k. Ukazatele na potomky (right[x] a left[x]) jsou nastaveny na NIL, ukazatel na rodiče (parent[x]) má též hodnotu NIL.
- Pokud kořen r je NIL, vloží se uzel x a stane se kořenem stromu.
- V opačném případě musíme najít místo, kam umístíme nový uzel. Operace *Vyhledávání*, kde hledaná hodnota je hodnota klíče k vkládaného uzlu x může skončit dvěma způsoby:
  - Funkce *Vyhledávání* nám vrátí ukazatel na uzel, tzn. uzel s klíčem k, který již ve stromu je.
  - Funkce *Vyhledávání* vrací NIL, na toto místo vložíme náš nový uzel x.

Časová složitost operace *Přidávání* uzlu x je $\theta(h)$, kde h je výška stromu, protože v nejhorším případě je vkládaný uzel list (cestujeme od kořene stromu až k listu). Ukázka přidání uzlu do BVS stromu (obr. 3).

Ukázka pseudokódu operace *Pridavani*, kde x je ukazatel na kořen stromu a z je ukazatel na nový uzel.

Algorithm 5 Pridavani(x, z)

```
y ← NIL
while x ≠ NIL do
  y ← x
  if klic[z] < klic[x] then
    x ← levy[x]
  else
    x ← pravy[x]
  end if
end while
rodic[z] ← y
if x = NIL then
  x ← z
else
  if klic[z] < klic[y] then
    levy[y] ← z
  else
    right[y] ← z
  end if
end if
```



Obrázek 3: Ukázka přidávání uzlu do BVS stromu

2.2.4 Odebírání

Operace *Odebírání* ruší v BVS námi požadovaný uzel x. Po zrušení tohoto uzlu x nedochází k porušení vlastností BVS. Výsledkem operace je pozměněný původní strom. Nejprve nalezneme rušený uzel x ve stromu a podle počtu potomků následují úpravy stromu.

Postup zrušení uzlu x ze stromu T:

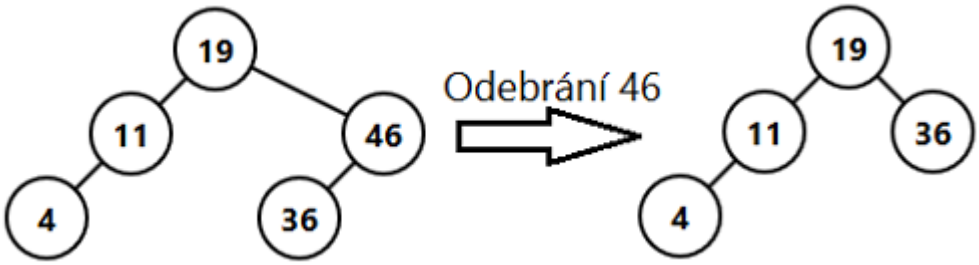
- Vyhledáme uzel x ve stromu T.
- Pokud se uzel x ve stromu T nenachází, operace *Odebírání* končí.
- V opačném případě postup závisí na počtu potomků uzlu x:
  1. Uzel x je bez potomků, tzn. že je list, jednoduše uzel x odebereme.
  2. Uzel x má jednoho potomka (pravého nebo levého), přepojíme ukazatele z rodiče uzlu x na potomka uzlu x a uzel x již můžeme odebrat.
  3. Uzel x má dva potomky. Uzel x musíme nahradit buď následovníkem nebo předchůdcem uzlu x. Záleží na implementaci. Když nahrazujeme uzel x následovníkem případně předchůdcem, zkopírujeme do uzlu x klíč z uzlu následovníka (předchůdce). Následovníka případně předchůdce odebereme podle předchozích bodů 1 nebo 2.

Časová složitost operace *Odebírání* v BVS je $\theta(h)$ způsobená hledáním odebíraného uzlu a hledáním následovníka nebo předchůdce (v nejhorším případě cestujeme z kořene stromu až k listu). Ukázka odebrání uzlu z BVS stromu (obr. 4).

Ukázka pseudokódu operace *Odebirani*, kde x je ukazatel na kořen stromu a z je ukazatel na nový uzel:

Algorithm 6 Odebirani(x, z)

$w \leftarrow NIL$
$y \leftarrow NIL$
if $levy[z] = NIL$ **or** $pravy[z] = NIL$ **then**
  $y \leftarrow z$
else
  $y \leftarrow Naslednik(z)$
end if
if $levy[z] \neq NIL$ **then**
  $w \leftarrow left[y]$
else
  $w \leftarrow right[y]$
end if
if $w \neq NIL$ **then**
  $rodic[w] \leftarrow rodic[y]$
end if
if $rodic[y] = NIL$ **then**
  $x \leftarrow w$
else
  if $y = levy[rodic[y]]$ **then**
    $levy[rodic[y]] \leftarrow w$
  else
    $pravy[rodic[y]] \leftarrow w$
  end if
end if
if $y \neq z$ **then**
  $klic[z] \leftarrow klic[y]$
end if
return y



Obrázek 4: Ukázka odebírání uzlu z BVS stromu

2.3 AVL stromy

AVL strom je vyvážený BVS s příznivou logaritmickou složitostí $\theta(\log(n))$ všech operací: *Vyhledávání*, *Přidávání*, *Odebírání*, *Následovník*, *Předchůdce*, *Minimum*, *Maximum*.

Vyváženost AVL stromu zformulovali v roce 1962 G. M. Adelson-Velskij a Je. M. Landis. Odtud název AVL.

Pojmy používané v další části textu:

- **h(l(x))** – výška levého podstromu uzlu x
- **h(p(x))** – výška pravého podstromu uzlu x
- **aktuální uzel *u*** – předchůdce uzlu x

Definice 5 (Vyvážený strom)
Strom je vyvážený tehdy, je–li rozdíl výšek každého uzlu nejvýše 1. Pro každý uzel platí:

$$|h(l(x)) - h(p(x))| \leq 1$$

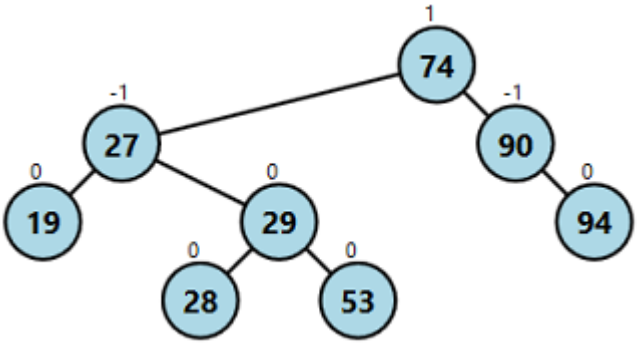
(výška levého i pravého podstromu se liší maximálně o 1)

Výška AVL stromu je maximálně $1.44 \cdot h$, kde h je výška úplně vyváženého binárního stromu se stejným počtem prvků n. Pro výšku úplně vyváženého stromu platí: $h = \lfloor \log_2(n) \rfloor$

Při práci s operacemi, které mění AVL strom (*Odebírání*, *Přidávání*) je nutné opravit tento strom, aby i nadále platila podmínka vyvážení. Proto si každý uzel uchovává tzv. **faktor vyvážení**, který si uchovává informaci o aktuálním vyvážení. Značíme ho **b** (z anglického slova **balance**), vypočteme ho následovně: ***b = h(l) - h(p)***. Nabývá pouze hodnot {-1, 0, 1}.

Struktura AVL uzlu obsahuje:

- **klíč** – hodnota uložená v uzlu
- **faktor vyvážení** uzlu
- ukazatele na **levého** a **pravého** potomka
- ukazatel na **rodiče**

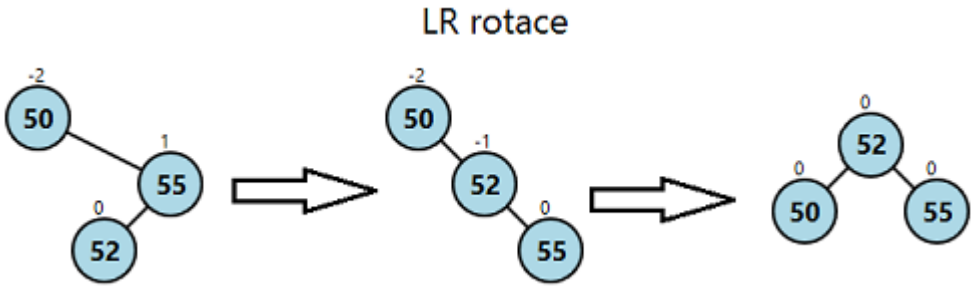


Obrázek 5: Ukázka AVL stromu

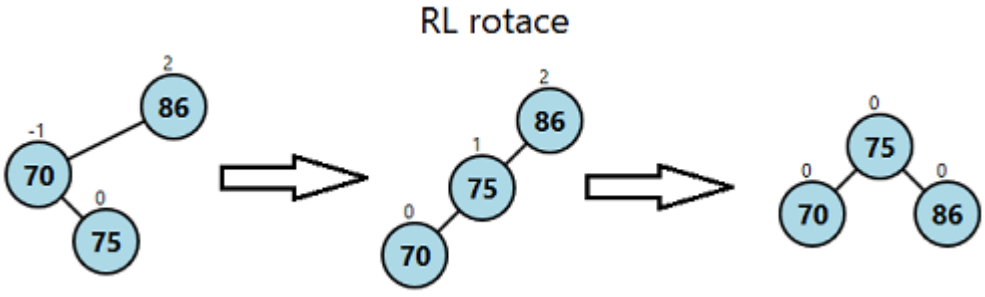
Ukázka AVL stromu na obr. 5.

Při provádění operací, které vyvažují strom, budeme kromě pravé a levé rotace (obr. 2) potřebovat tzv. *dvojité* rotace. Tyto rotace jsou poskládány z jednoduchých pravých a levých rotací:

- *dvojitá rotace LR* – pravá rotace, následuje levá rotace (obr. 6)
- *dvojitá rotace RL* – levá rotace, následuje pravá rotace (obr. 7)



Obrázek 6: Ukázka dvojité LR rotace u AVL stromů



Obrázek 7: Ukázka dvojité RL rotace u AVL stromů

2.3.1 Vyhledávání

Vyhledávání probíhá stejným způsobem jako u BVS. Rozdíl je ale v časové složitosti, která je nyní $\theta(\log(n))$, protože výška AVL stromu je $\log n$.

2.3.2 Přidávání

Přidávání uzlu do AVL stromu probíhá stejným způsobem jako u BVS. Nový uzel má nastaven faktor vyvážení na 0. Na konci přidání uzlu x do stromu T musíme u určitých uzlů přepočítat faktor vyvážení. Podle něj následně buď operace *Přidávání* končí nebo strom T opravíme příslušnými rotacemi.

Operace *Přidávání* probíhá následovně:

- uzel x vložíme stejným způsobem jako u BVS
- od uzlu x postupujeme ke kořeni a přepočítáváme faktor vyvážení
- pokud jsme vložili uzel x do levého podstromu aktuálního uzlu u, jeho faktor zvýšíme o 1 ($b = b + 1$)
- pokud jsme vložili uzel x do pravého podstromu aktuálního uzlu u, jeho faktor zmenšíme o 1 ($b = b - 1$)
- po přidání uzlu x mohou nastat 3 případy u aktuálního uzlu u:
  1. b = 0 – Přidání uzlu x nemá vliv na další uzly ve stromu, *Přidávání* končí (obr. 8).
  2. b = 1 nebo b = -1 – Je–li u kořen, končíme. Jinak učiníme u rodiče aktuálního uzlu u a opakujeme postup (obr. 9).
  3. b = 2 nebo b = -2 – Vyvážíme vhodnou rotací a přepočítáme faktor vyvážení příslušných uzlů. Je-li u kořen nebo $b = 0$, končíme. Jinak učiníme u rodiče aktuálního uzlu u a opakujeme postup (obr. 10).

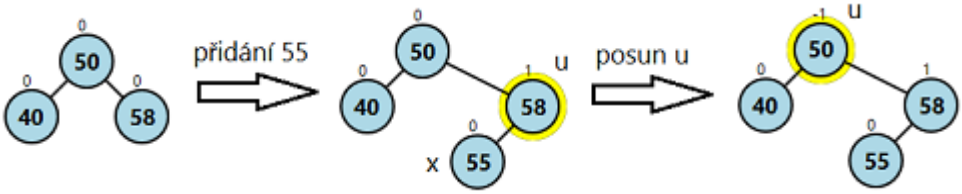
Provedení rotací a přepočítání faktoru vyvážení je provedeno v konstantním čase $\theta(1)$. Výsledná časová složitost *Přidávání* je $\theta(\log(n))$, opět v nejhorším případě cestujeme od kořene stromu k listu.

2.3.3 Odebírání

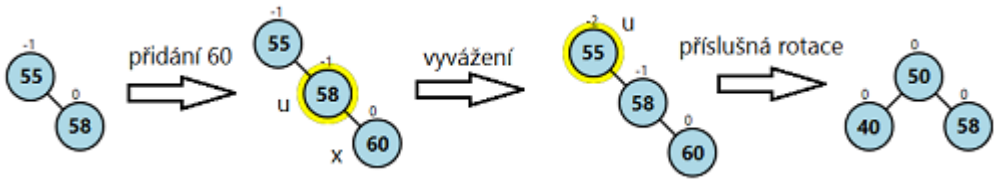
Zrušení uzlu v AVL je podobný operaci přidávání uzlu do stromu. Postupujeme stejným způsobem jako u odebráni uzlu u BVS . Poté přepočítáme faktor vyvážení v příslušných částí stromu jednotlivých uzlů a je-li potřeba, provedeme vyvážení.

Podrobnější postup:

Obrázek 8: Příklad *Přidání* uzlu do AVL stromu, kde $b = 0$



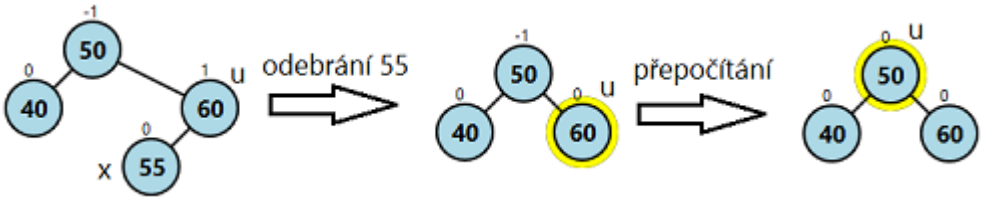
Obrázek 9: Příklad *Přidání* uzlu do AVL stromu, kde $b = 1$



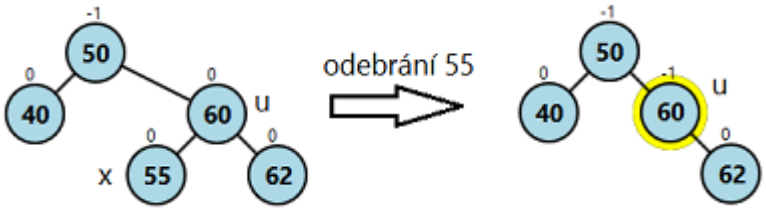
Obrázek 10: Příklad *Přidání* uzlu do AVL stromu, kde $b = -2$

- uzel x odebereme stejným způsobem jako u BVS
- pokud jsme zrušili uzel x v levém podstromu aktuálního uzlu u, jeho faktor zmenšíme o 1 ($b = b - 1$)
- pokud jsme zrušili uzel x v pravém podstromu aktuálního uzlu u, jeho faktor zvětšíme o 1 ($b = b + 1$)
- po zrušení uzlu x mohou nastat 3 případy u aktuálního uzlu u:
  1. b = 0 – Je–li uzel u kořenem stromu, končíme. V opačném případě učiníme u rodiče aktuálního uzlu u a opakujeme postup (obr. 8).
  2. b = 1 nebo b = -1 – Zrušený uzel nemá vliv na další uzly, operace končí (obr. 9).
  3. b = 2 nebo b = -2 – Vyvážíme vhodnou rotací a přepočítáme faktor vyvážení příslušných uzlů. Je-li u kořen, $b = 1$ nebo $b = -1$, končíme. Jinak učiníme u rodiče aktuálního uzlu u a opakujeme postup (obr. 10).

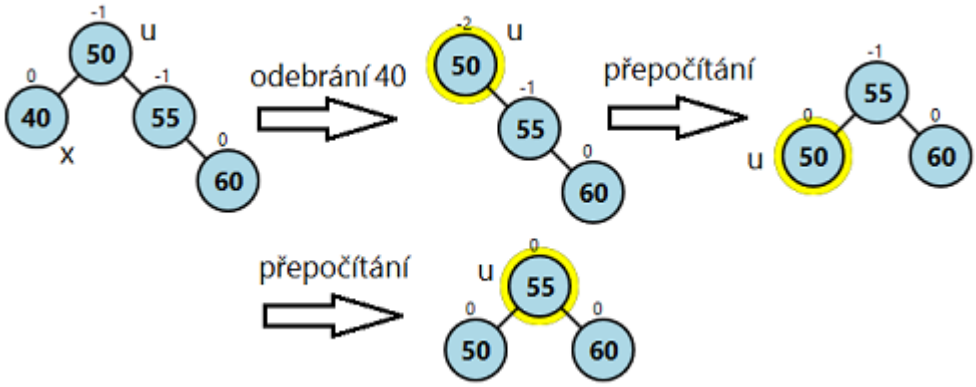
Časová složitost operace *Odebírání* je $\theta(\log\ n)$.



Obrázek 11: Příklad odebrání uzlu z AVL stromu, kde $b = 0$



Obrázek 12: Příklad odebrání uzlu z AVL stromu, kde $b = -1$



Obrázek 13: Příklad odebrání uzlu z AVL stromu, kde $b = -2$

2.4 Červeno-černé stromy

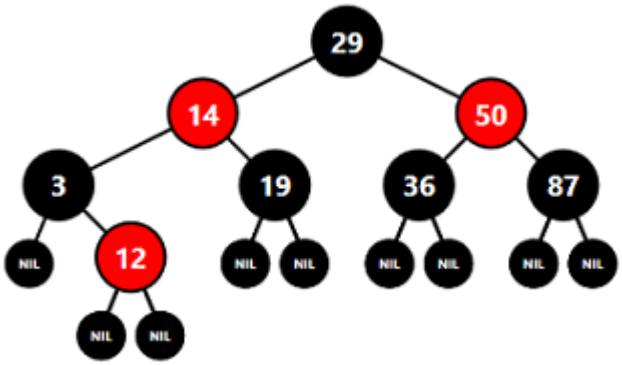
Červeno-černé stromy (dále už jen ČČ) jsou **samovyvažovací** BVS. Vynalezl je Rudolf Bayer v roce 1972 (10 let od uvedení AVL stromů) a jmenovaly se *Symetrické binární B–stromy* (Symmetric binary B–tree). Název *červeno–černý strom* (Red–Black tree) dostaly až v roce 1978 od Leonidase J. Guibase a Roberta Sedgewicka.

ČČ stromy se od BVS liší jedním dvouhodnotovým příznakem v uzlu navíc. Tento příznak se nejčastěji implementuje pomocí typu *bool* [5]. Představuje barvu uzlu, která je buď **červená** nebo **černá**.

[5]Nabývá hodnot true nebo false (1 nebo 0).

Definice 6 (Červeno-černé stromy)
Každý červeno–černý strom musí splňovat následující podmínky:

1. *Každý uzel je buď černý, nebo červený.*
2. *Každý list (Nil) je černý.*
3. *Každý červený uzel má pouze černé potomky.*
4. *Kořen je vždy černý.*
5. *Každá cesta z libovolného uzlu do listu (Nilu) obsahuje stejný počet černých uzlů.*



Obrázek 14: Ukázka ČČ stromu

Struktura ČČ uzlu obsahuje:

- **klíč** – hodnota uložená v uzlu
- **barva** – nastavuje barvu uzlu
- ukazatele na **levého** a **pravého** potomka
- ukazatel na **rodiče**

Černá výška uzlu *x* – značíme *bh(x)*, je počet černých uzlů na cestě z uzlu x do listu (mimo uzel x).

Černá výška stromu – černá výška kořene stromu.

Věta 7 (Výška ČČ stromu)
***Výška** ČČ stromu s n vnitřními uzly* [6] *je nejvýše* $2log_2(n+1) = \theta\ (log\ (n))$.

Protože výška ČČ stromu je nejvýše $2log_2(n+1)$ a strom je vyvážený, operace prováděné nad tímto stromem mají nejhorší časovou složitost $\theta(\log\ (n))$.

[6] Všechny uzly, které nejsou listy (Nil).

2.4.1 Vyhledávání

Vyhledávání probíhá stejným způsobem jako u BVS. Časová složitost je u ČČ stromů $\theta(\log\ (n))$.

2.4.2 Přidávání

Přidávání do ČČ stromu probíhá stejným způsobem jako u BVS. Každý nově vložený uzel je obarven na **červeno**. Poté se pokračuje následujícími kroky:

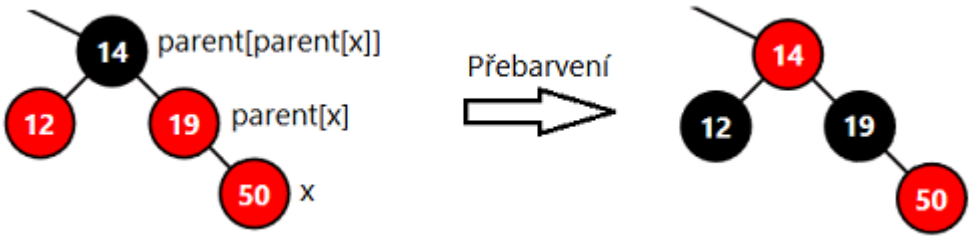
- rodič (dále používán anglický překlad parent) vkládaného uzlu je černý, vkládání končí
- jinak je porušena 3. podmínka u vlastností ČČ stromů a musí se provést oprava stromu za použití přebarvení uzlů nebo pomocí rotací
- na konci operace přebarvíme kořen na černou barvu

U *Přidávání* rozlišujeme 3 případy, které mohou nastat po vložení nového uzlu:

Případ 1: platí následující podmínky:

- rodič vkládaného uzlu x, parent[x] je červený
- strýc uzlu x je červený

Oprava spočívá v přebarvení rodiče uzlu x a strýce uzlu x na černou barvu. Dále přebarvíme rodiče uzlu parent[x] (parent[parent[x]]) na červeno. Tento krok nemusí vyřešit náš problém dvou červených uzlů, pouze posouvá problém blíže ke kořenu. Rekurzivně opakujeme postup. Ukázka příkladu na obr. 15.



Obrázek 15: Ukázka obnovení vlastností ČČ stromu při přidávání; *Případ 1*

Případ 2: platí následující podmínka:

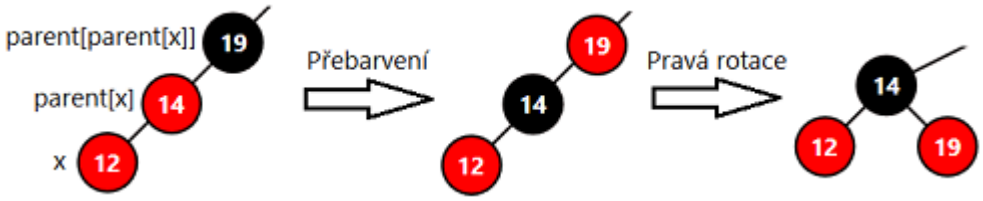
- vkládaný uzel x je levý potomek, rodič vkládaného uzlu x (*parent[x]*) je červený a levým potomkem svého rodiče (*parent[parent[x]]*)

- pokud existuje strýc uzlu x je černý (když neexistuje je to Nil uzel a ten je černý)
- symetricky pro druhou stranu

Tento případ opravíme následovně:

- rodiče vkládaného uzlu x (*parent[x]*) přebarvíme na černo
- uzel parent[parent[x]] přebarvíme na červeno
- následuje pravá rotace
- po těchto úpravách je vkládaní kompletní, platí symetricky pro druhou stranu

Ukázka příkladu na obr. 16.

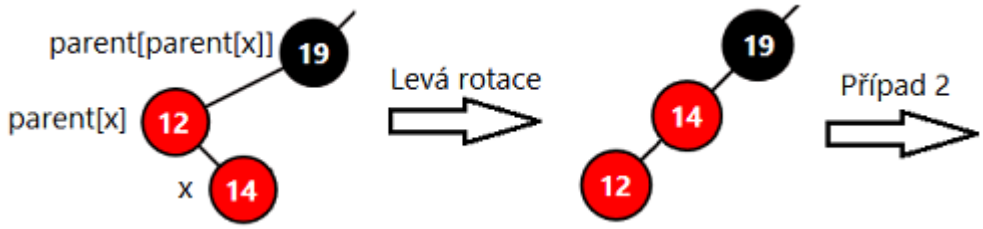


Obrázek 16: Ukázka obnovení vlastností ČČ stromu při přidávání; *Případ 2*

Případ 3: platí následující podmínka:

- vkládaný uzel x je pravý potomek, rodič vkládaného uzlu x (*parent[x]*) je červený a je levým potomkem svého rodiče
- symetricky pro druhou stranu

Pro opravení této podmínky je zapotřebí provést levou rotaci. Tento krok neobnoví vlastnosti ČČ stromu, pokračujeme *Případem 2*. Symetricky postupujeme pro druhou stranu. Ukázka příkladu na obr. 17.



Obrázek 17: Ukázka obnovení vlastností ČČ stromu při přidávání; *Případ 3*

Každý z uvedených příkladů je proveden v konstantním čase $\theta(1)$. U *Případu 1* se pouze přebarví uzly a problém se posune blíže ke kořeni, maximálně se provedou 2 rotace a to u *Případu 2* a *Případu 3*. Výsledná složitost operace *Přidávání* je $\theta(\log\ (n))$, způsobená nalezením místa, kam se uzel vloží a cyklickým opakováním *Případu 1*.

2.4.3 Odebírání

Zrušení uzlu v ČČ stromu je náročnější než operace *Přidávání* uzlu do stromu. Závisí zde na barvě skutečně odebíraného uzlu. Pokud je odebíraný uzel červený, je zrušení uzlu snadné. Ale v opačném případě, zrušením černého uzlu způsobíme, že podmínka 5 (každá cesta z libovolného uzlu do listu obsahuje stejný počet černých uzlů) nebude platit.

Pojmy používané v této části textu:

- **sourozenec S** – u obrázků anglický překlad *sibling*
- **rodič P** – u obrázků anglický překlad *parent*, zkratka *P*
- zkratka **Sl** – levý potomek sourozence uzlu x
- zkratka **Sp** – pravý potomek sourozence uzlu x
- **Nil** uzel – potomek rušeného uzlu

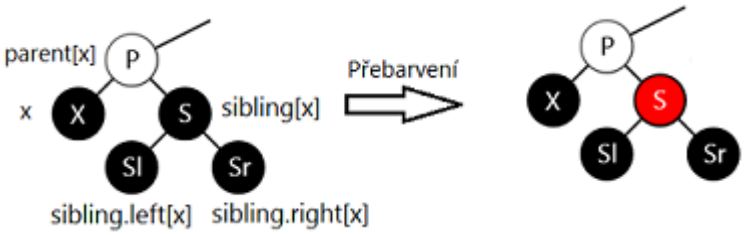
Podrobný postup zrušení uzlu v ČČ stromu:

- uzel odstraníme stejným způsobem jako u BVS
- dále hledíme jen na skutečně odebíraný uzel (náš rušený uzel může mít následovníka)
- pokud rušený uzel je červený, žádná z podmínek u vlastností ČČ stromů není porušena
- jinak odebíraný uzel je černý a postupujeme následovně:
  - označíme uzel x jako potomka rušeného uzlu (pokud není potomek, je to Nil uzel) [1]
  - pokud je uzel x červený, přebarvíme ho na černou barvu
  - jinak označíme uzel x jako dvojitě černý a jedné černé barvy se snažíme zbavit po cestě ke kořeni stromu
  - této černé barvy na uzlu x se zbavíme příslušnými rotacemi nebo přebarvením podle možných *Případů 1–4*

Případ 1: platí následující podmínky:

- sourozenec S uzlu x je černý, x je levým potomkem svého rodiče
- oba potomci sourozence S jsou černí
- rodič P uzlu x je červený nebo černý
- symetricky pro druhou stranu

Oprava spočívá v přebarvení sourozence S na červenou barvu. Tento krok pouze posouvá problém blíže ke kořenu. Pokračujeme vhodnou úpravou podle vhodných případů. Ukázka příkladu na obr. 18.

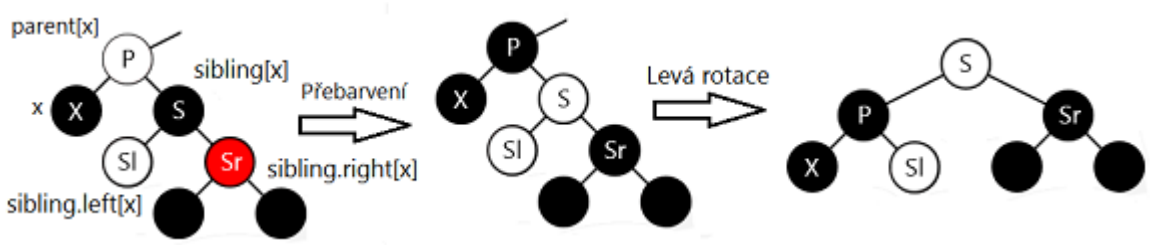


Obrázek 18: Ukázka obnovení vlastností ČČ stromu při odebírání; *Případ 1*

Případ 2: platí následující podmínky:

- uzel x je levým potomkem
- rodič uzlu x je červený nebo černý
- sourozenec S uzlu x je černý
- pravý potomek uzlu S je červený
- levý potomek uzlu S je červený nebo černý
- symetricky pro druhou stranu

Přebarvíme sourozence S podle barvy rodiče uzlu x. Následně přebarvíme na černou barvu jak rodiče uzlu x, tak pravého potomka sourozence S. Po přebarvení provedeme levou rotaci. Symetricky platí pro druhou stranu. Ukázka příkladu na obr. 19.

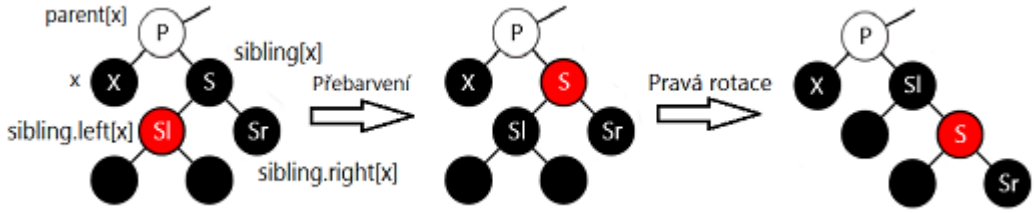


Obrázek 19: Ukázka obnovení vlastností ČČ stromu při odebírání; *Případ 2*

Případ 3: platí následující podmínky:

- uzel x je levým potomkem
- sourozenec S uzlu x je černý
- levý potomek uzlu S je červený
- rodič uzlu x je červený nebo černý
- symetricky pro druhou stranu

V prvním kroku opravy přebarvíme sourozence S na červenou barvu a levého potomka uzlu S na černou. Dále provedeme pravou rotaci. Tento případ se zjednoduší a pokračujeme *Případem 2*. Symetricky platí pro druhou stranu. Ukázka příkladu na obr. 20.

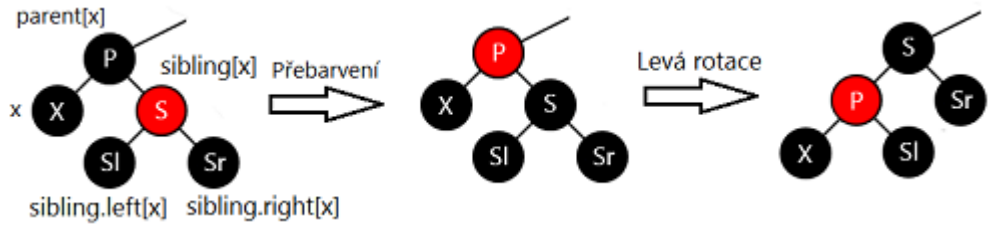


Obrázek 20: Ukázka obnovení vlastností ČČ stromu při odebírání; *Případ 3*

Případ 4: platí následující podmínky:

- sourozenec S uzlu x je červený
- rodič uzlu x je černý

Přebarvíme uzel S na černou barvu a rodiče uzlu x na červenou. Poté provedeme levou rotaci. Tento případ neobnoví vlastnosti ČČ stromů, posouvá problém o krok dále od kořene. Symetricky platí pro druhou stranu.
Ukázka příkladu na obr. 21.



Obrázek 21: Ukázka obnovení vlastností ČČ stromu při odebírání; *Případ 4*

Jelikož *Případy 2–4* přebarvují uzly a provedou maximálně 3 rotace je složitost těchto případů konstantní $\theta(1)$. *Případ 1* přesouvá korekci barvy postupně ke kořeni stromu. Jelikož výška stromu je $\theta(\log(n))$, v nejhorším případě je i časová složitost *Případu 1* $\theta(\log(n))$. Proto operace *Odebírání* z ČČ stromu má časovou složitost $\theta(\log(n))$.

3 Programovací jazyk a použité nástroje

Aplikace je napsána v jazyce C# s použitím .NET Frameworku 4.5, pro grafické zpracování byla použita technologie Windows Presentation Foundation (dále jen WPF). Aplikace je vytvořená na operačním systému Windows 8.1 ve vývojovém prostředí Visual Studio 2013. Při volbě programovacího jazyka hrála roli dobrá podpora jazyka k vytvoření grafické aplikace. V druhé řadě pak zkušenost s tímto jazykem. Tato kapitola vychází především z publikací [3].

3.1 C#

C# je objektově orientovaný programovací jazyk, vytvořený firmou Microsoft, určený k vývoji aplikací pro platformu .NET. C# je následníkem jazyka C++ a podobný jazyku Java. Oproti C++ je C# jednoduší, chybí zde ukazatelé, které jsou zde skryty. Další výhodou je například automatická alokace paměti[7] nebo rozsáhlý systém knihoven. Jazyk C# se používá k tvorbě formulářových, databázových, webových, mobilních aplikací a podobně.

3.2 .NET Framework

Framework .NET je platforma vytvořena firmou Microsoft v roce 2000 k vývoji aplikací. Pracuje na principu řízeném běhovém prostředí. To znamená, že při převodu zdrojového kódu do strojového kódu využívá jedné mezivrstvy. Tento kód z mezivrstvy (mezikód) je do strojového kódu převeden až na cílové platformě při spuštění programu. .NET se skládá z několika částí:

- **Microsoft Intermediate Language (MSIL)** – označuje mezikód, který je spouštěn pomocí **CLR**.
- **Common Language Runtime (CLR)** – zajišťuje běh programů přeložených z různých programovacích jazyků do mezikódu
- **Common Type System (CTS)** – sada datových typů pro programovací jazyky
- **Common Language Specification (CLS)** – spolu s **CTS** určují specifikaci, jakou musí mít každý jazyk, který využívá tuto platformu (například C#, J#, Visual Basic .NET)

3.3 WPF

Technologie WPF je rozhraní pro návrh, tvorbu a zobrazení uživatelského prostředí formulářových aplikací. Poprvé bylo součástí .NET Frameworku verze

[7] Obstarává ji *Garbage Collector*, který se stará o uvolňování zdrojů spuštěného programu.

3.0 jako WPF3. S každou novou verzí .NET Frameworku vyšla i nová verze WPF, dnes je k dostání WPF4.5. WPF vzniklo jako nástupce WF[8], které je v dnešní době již zastaralé a nezvládá nové technologie u stále náročnějších grafických aplikacích. Hlavní nevýhodou WF je neschopnost zobrazit na více zařízeních aplikaci ve stejném rozlišení. Proto je celé WPF vektorové, lze díky tomu elementy, které aplikace obsahuje zmenšovat, zvětšovat a různě transformovat. Výsledná aplikace je nezávislá na rozlišení zobrazovacího zařízení. Pro vykreslování formulářů využívá WPF techniku *Direct3D*[9]. Díky tomu jsou aplikace rychlejší a méně náročné na procesor. Hlavní výhodou, proč jsem si k bakalářské práci zvolil WPF je možnost tvorby jednoduchých animací přes *StoryBoard*.

3.4 XAML

EXtensible Application Markup Language (dále už jen XAML) je značkovací jazyk vycházející z jazyka XML[10]. Tento jazyk je určený k návrhu uživatelského prostředí v technologii WPF. Definujeme v něm jednotlivé elementy, jejich vlastnosti a vazby mezi nimi. Vše, co lze zapsat pomocí XAML, lze zapsat i v jiných jazycích, konkrétně v jazyce C# a Visual Basic .NET.

3.5 Storyboard

Objekt *Storyboard* je typ kontejneru obsahující objekty typu *DoubleAnimation*[11] nebo objekty typu *TimeLine*[12]. *Storyboard* dále definuje časovou osu, dobu trvání, druh animace (zvětšování, posun, zviditelňování, změny barev, ...) a mnoho dalšího. V bakalářské práci je *Storyboard* využíván k posunu uzlů po *Canvasu*[13] a k zneviditelňování uzlů (u odebírání uzlů ze stromu).

3.6 Microsoft Excel

Data získána ze vzájemného srovnání rychlostí obou stromů jsou zobrazena v grafu, vytvořeného pomocí MS Excel[14](dále už jen Excel). Data jsou zpracována programem, poté jsou vložena do Excelu. V něm proběhne vytvoření grafu, který je potom jako obrázek exportován zpátky do programu. Excel umí vytvářet přehledné grafy, což byl první důvod zvolení tohoto postupu. Druhým

[8]Windows Forms – První framework z .NET sloužící k vytváření grafických formulářových aplikací.

[9]Rozhraní, které obsahuje funkce pro práci s 3D grafikou.

[10]EXtensible Markup Language – Rozšiřitelný značkovací jazyk, slouží k popisu dat.

[11]Animuje hodnotu *Double* mezi dvěma hodnotami: začátkem a koncem.

[12]Objekty, které vytváří časovou osu.

[13]Kreslící plátno, slouží k zobrazení stromů uživateli.

[14]Tabulkový editor sloužící k vytváření a úpravám tabulek.

důvodem bylo vzájemné propojení mezi Excelem a MS Visual Studiem. Jelikož jsou při programování používané knihovny z Exelu, výsledný program je nespustitelný bez nainstalovaného Excelu na počítači.

4 Implementace

V této části textu si blíže nastíníme programy využité pro implementaci jednotlivých částí bakalářské práce. Dále se budeme zabývat způsobem implementace jednotlivých částí programu a řešením vzniklých problémů, především v grafické části.

4.1 Stromy

Stromy jsou implementované ve třídách `RBTree` a `AVLTree`. Tyto třídy si jsou podobné, obsahují vlastnosti stromů a hlavní metody k *Vyhledávání*, přidávání a *Odebírání*. Vlastnosti stromů můžeme rozdělit na základní a vlastnosti potřebné ke grafické části. Mezi základní vlastnosti tříd `RBTree` a `AVLTree` patří:

- **root** – kořen stromu
- **count** – počet uzlů ve stromu
- **deleteNode, replaceNode** – mazaný uzel, uzel který nahrazuje odebíraný uzel (důležité při odebírání uzlů v animaci)

Mezi vlastnosti potřebné ke grafické části programu patří:

- **listLine** – seznam uzlů ve stromu
- **whoMove** – seznam uzlů, u kterých byly změněny souřadnice při vyvažování
- **history** – sem se ukládají postupné změny ve stromu, s tímto seznamem pak pracuje třída `MainWindow`

Dále jsou zde potřebné metody k určení a přepočítání souřadnic uzlů ve stromu. Po každé rotaci, vložení nebo odebrání uzlu ze stromu, musí dojít k přepočítání souřadnic. Například vložení uzlu do stromu proběhne velice rychle a provedou se potřebné rotace k vyvážení stromů, souřadnice uzlů se změní a zůstanou pouze ty, které se vypočítaly naposledy. Pro správnou animaci je ovšem zapotřebí mít správné počáteční a konečné souřadnice. Potřeba jsou ovšem všechny souřadnice daných uzlů, protože animací se provádí více za sebou. Proto jsou v průběhu rotací a při přepočítávání nových souřadnic dělané kopie daných uzlů, které se mění. Tyto kopie jsou ukládány do seznamu **history**. Ve výsledku máme uloženou historii, jak probíhalo přidání nebo odebrání do/ze stromu.

Ve třídách `RBTree` a `AVLTree` jsou implementovány různé průchody stromem. Opět se ukládají uzly do seznamu, aby bylo možné je zanimovat a zobrazit průchod stromem v *Canvasu*. Tyto uzly se ze seznamu odebírají a vkládají do *DoubleAnimation*.

Základní princip AVL stromů vychází z rekurzivní implementace z [4], princip odebírání u ČČ stromů pak z [5].

4.2 Uzly

Stromy jsou reprezentovány pomocí uzlů, u kterých v této části nastíníme jejich vlastnosti a implementaci bez grafické části. Každý uzel obsahuje tyto vlastnosti:

- **value** – hodnota uzlu
- **left, right, parent** – odkaz na levého a pravého potomka, dále pak na rodiče
- **coordinateOLD** – stará souřadnice, výchozí bod u animace
- **coordinateNEW** – nová souřadnice, cílový bod u animace
- **nodeButton** – k zobrazení uzlu, důležité je pojmenování tohoto *Buttonu*, při každé animaci se musí toto jméno registrovat, slouží k identifikaci

Další vlastnosti závisí na druhu uzlu, buď AVL nebo ČČ uzel.

4.2.1 AVL

Uzly AVL stromu navíc obsahují následující vlastnosti:

- **balance** – faktor vyvážení uzlu
- **balanceLa** – *Label*, slouží k zobrazení faktoru vyvážení v *Canvasu*

4.2.2 Červeno-černé

Červeno-černý uzly navíc obsahují tuto vlastnost:

- **colorRed** – boolovská hodnota, zda je uzel červený (true) nebo černý (false)

4.3 Grafické rozhraní

Hlavní třída pro zobrazení hlavního okna programu je třída `MainWindow`. Elementy tohoto okna jsou nadefinovány v souboru `MainWindow.xaml`. Třída `MainWindow` je spuštěna jako první při startu programu. Obstarává funkčnost jednotlivých elementů okna, převážně pak obsahuje metody určené k animacím stromu (podrobněji v podsekci `Animace`). Podle výběru typu stromu, je formulář pozměněn; například zobrazení *CheckBoxu* pro Nil uzly a podobně.

Dále třída `MainWindow` obsahuje implementaci *Expanderu* [15] **Teorie stromů**. *Expander* se mění podle výběru mezi AVL a ČČ stromy. Obsahuje tutoriál pro jednotlivé operace na stromech.

Pro znázornění a pochopení studentů jsou zde znázorněny průchody stromem. Jsou implementovány pomocí již dříve zmíněné animace za použití *Storyboardu*. Ve třídě `RBTree` nebo `AVLTree` jsou pomocné seznamy, které obsahují hodnoty uzlů v pořadí podle výběru průchodu. Tyto uzly ze seznamu jsou zvýrazněny ve správném pořadí. Na stejném principu je založeno i *Vyhledávání* ve stromu.

Další okno programu je k druhé části bakalářské práce, pro časovou analýzu. Obsluhují ho třídy `TimeAnalyst` a `TimeAnalystGUIMan`. Pro jednoduchost ovládání obsahuje tento formulář pouze nejnutnější tlačítka a *Image*, do kterého se zobrazuje výsledný graf. Třída `TimeAnalystGUIMan` jako jediná pracuje s Excelem. Metoda pro práci s Excelem převzata z [6]. Vytváří graf z naměřených hodnot. Tento graf je potřeba uložit do počítače, aby se zobrazil v programu. Data pro tvorbu grafu jsou získána z objektů typu *GraphPoint*. Tento objekt je tvořen třídou `GraphPoint`, která obsahuje pouze informace o počtu uzlů a časové době. Jelikož získání dat trvá delší dobu (podle čísla, které zadá uživatel), běží tento proces ve druhém vlákně, aby nezamrzl celý program.

Pro větší efektivitu, jak časovou, tak paměťovou, jsou pro časovou analýzu znovu napsané hlavní třídy pro práci se stromy. Jsou to třídy `RBTreeTime` a `AVLTreeTime`. Obsahují pouze důležité metody. Chybí zde oproti třídám `RBTree` a `AVLTree` všechny metody, které počítaly posuny uzlů volané v rotacích při vyvažování nebo při vkládání/odebírání uzlů do/ze stromů. AVL implementace byla z rekurzivního postupu předělána. Faktor vyvážení je počítán uvnitř rotací. Implementace rotací převzaty z [7]. Aby uzly zabíraly co nejméně paměti, obsahují pouze vlastnosti nutné k funkčnosti přidání a odebrání ze stromu. Obsahují:

- **value**
- **left, right, parent**
- **balance** – u AVL uzlu
- **colorRed** – u ČČ uzlu

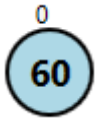
4.3.1 Grafická reprezentace uzlů a hran

Každý uzel je zobrazen pomocí *Buttonu*[16]. Je to z důvodu možnosti animace, transformace a možnosti nadefinování vlastního stylu tlačítka pomocí XAML. Tento styl obsahuje dvě elipsy (jedna jako okraj uzlu, druhá výplň), nastavení barvy pozadí, velikosti elips a vycentrování kontextu (zobrazuje hodnotu uzlu). Celkem je nadefinováno pět stylů:

[15] Rozbalující menu.
[16] Obyčejné tlačítko na formulářových aplikacích.

- **RedNode** – styl pro červený uzel v ČČ stromu
- **BlackNode** – styl pro černý uzel v ČČ stromu
- **SearchNode** – styl pro vyhledávací uzel v ČČ a AVL stromu
- **NilNode** – styl pro Nil uzel v ČČ stromu
- **AVLNode** – styl pro uzel v AVL stromu

K AVL uzlu je přidán navíc `Label`, který zobrazuje aktuální faktor vyvážení daného uzlu.
Ukázka stylů uzlů:



Obrázek 22: Příklad reprezentace AVLNode

Obrázek 23: Příklad reprezentace SearchNode

Obrázek 24: Příklad reprezentace RedNode

Obrázek 25: Příklad reprezentace BlackNode

Obrázek 26: Příklad reprezentace NilNode

Pro zobrazení hrany používám vykreslení *Line*[17]. Po každém posunu uzlů, se hrany musí překreslit podle aktuálních souřadnic uzlů. Všechny tyto elementy jsou zobrazovány v *Canvasu*.

4.4 Animace

Vlastní animace je tvořena objektem *DoubleAnimation*. Tomuto objektu nastavíme následující vlastnosti:

- **Duration** – doba délky animace
- **From** – souřadnice začátku animace
- **To** – souřadnice konce animace

Když máme vytvořenou *DoubleAnimation*, přidáme ji do kontejneru *Storyboard*, v něm dále nastavíme:

- **BeginTime** – zpoždění animace
- **Children** – zde přidáme naši vytvořenou *DoubleAnimation* animaci
- **SetTargetName -** zde přidáme jméno objektu, který budeme animovat (v našem případě *Button*, který je identifikován podle nastaveného jména, tudíž získáme správný uzel)
- **SetTargetProperty** – nastavení podle požadavku animace
  - **PropertyPath(Canvas.LeftProperty)** – pro horizontální animaci
  - **PropertyPath(Canvas.TopProperty)** – pro vertikální animaci
  - **PropertyPath(Button.OpacityProperty)** – ke zprůhlednění uzlu

Princip, který využívám při znázornění animací, je následující. Nejprve získáme uzly ze seznamu **history**. Vytvoříme příslušnou animaci a spustíme. Po začátku animace jsou tyto uzly odebrány ze seznamu. Po skončení animace se rekurzivně volají obslužné metody; například `DeleteCompletAVL()`. Podle stavu seznamu **history** je animace ukončena nebo pokračuje.

[17] Úsečka z bodu do bodu, nastavena na černou barvu a tloušťku 2 pixely.

5 Časová analýza

V této kapitole je analyzováno vzájemné srovnání rychlostí AVL a červeno-černých stromů. Z důvodu proměnlivých výsledků měření byl postup měření následující:

- každé jednotlivé měření bylo provedeno 10 krát
- poté v Excelu byly jednotlivé hodnoty zprůměrovány, zapsány do tabulky a z nich byl vytvořen sloupcový graf

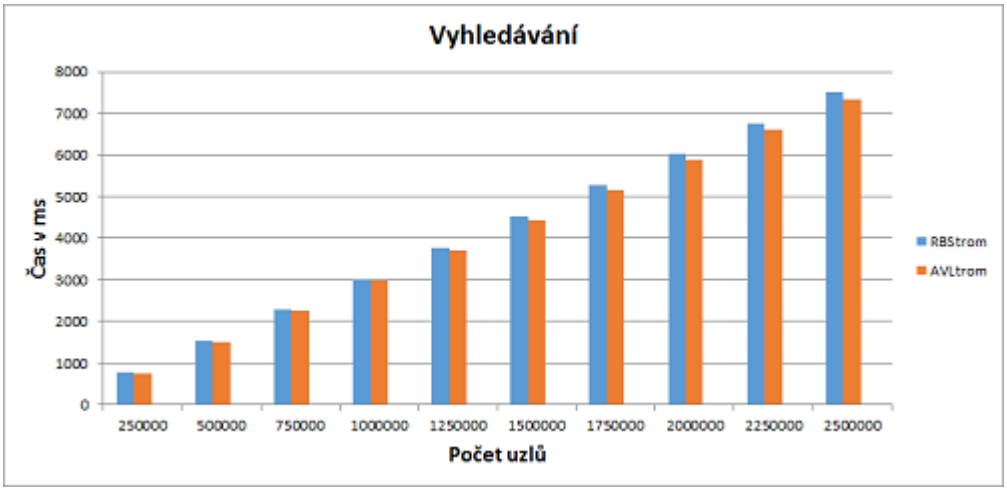
Zadávaná hodnota je omezena na hodnotu 10 000 - 10 000 000. Hodnota 10 000 je z důvodu, že pro nižší hodnotu je naměřený výsledek buď nulový (musely by se měřit tiky procesoru), anebo maximálně jednotky milisekund. Horní omezení zadávané hodnoty je kvůli velké paměťové náročnosti programu.

5.1 Vyhledávání

Pro měření dat ve *Vyhledávání*, byl postup následující:

- vygenerováno pole o 500 000 náhodných čísel v rozmezí hodnoty *Integeru*
- do každého stromu byla tato čísla vložena, aby byly stromy stejné
- poté bylo vyhledáváno 2,5 mil. uzlů

Souhrn naměřených výsledků v tabulce 1.
Pro vizuální představu ukázka grafu *Vyhledávání* obr.27.



Obrázek 27: Graf *Vyhledávání* reprezentující data z Tabulky 1

Naměřené hodnoty odpovídají našemu očekávání, jelikož AVL stromy mají lepší hloubku stromu a s tím spojenou i rychlost vyhledávání.

Počet uzlů	**ČČ[ms]**	**AVL[ms]**
250 000	777	747
500 000	1 525	1 500
750 000	2 278	2 245
1 000 000	3 025	2 976
1 250 000	3 773	3 705
1 500 000	4 526	4 434
1 750 000	5 271	5 160
2 000 000	6 018	5 886
2 250 000	6 763	6 617
2 500 000	7 514	7 348

Tabulka 1: Tabulka srovnávající časy *Vyhledávání* v jednotlivých stromech

5.2 Přidávání

Pro měření dat formou přidávání, byl postup následující:

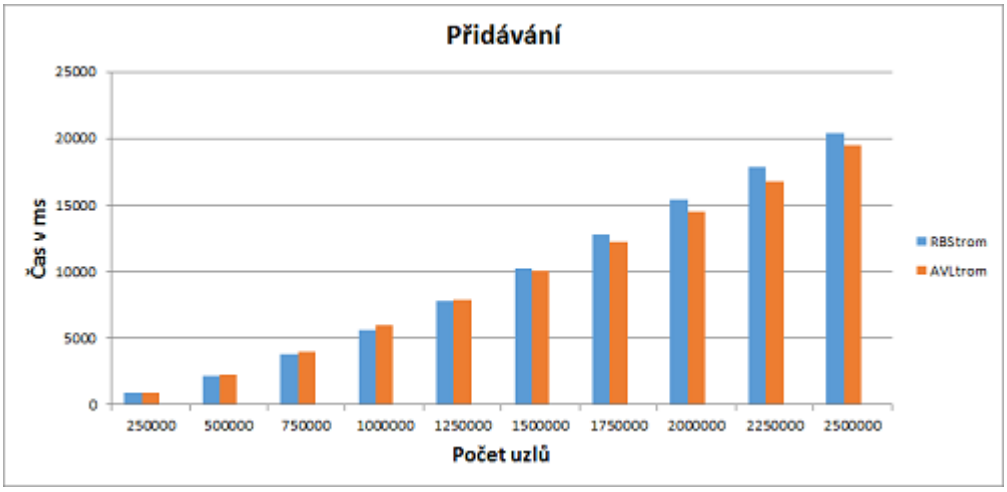
- bylo vytvořeno pole *Integeru* o velikosti 2,5 mil.
- do každého stromu byla tato čísla z pole vložena, aby byly stromy stejné
- měření probíhalo 10 krát, jak bylo řečeno na začátku kapitoly

Souhrn naměřených výsledků operace *Přidávání* uzlů do stromů v tabulce 2. Pro vizuální představu ukázka grafu *Přidávání* obr. 28.

Pro zajímavost, vložení 10 mil. uzlů trvá u AVL stromu zhruba 100 251 ms (1 min 40 s) a u červeno-černého stromu zhruba 107 384 ms (1 min 47 s), paměťová náročnost je přibližně 580 MBajtů RAM.

Počet uzlů	ČČ[ms]	AVL[ms]
250 000	890	927
500 000	2 174	2 286
750 000	3 778	3 967
1 000 000	5 630	5 962
1 250 000	7 807	7 884
1 500 000	10 210	10 018
1 750 000	12 759	12 232
2 000 000	15 428	14 524
2 250 000	17 835	16 749
2 500 000	20 423	19 521

Tabulka 2: Tabulka srovnávající časy *Přidávání* v jednotlivých stromech



Obrázek 28: Graf znázorňující srovnání rychlosti jednotlivých stromů při *Přidávání* uzlů reprezentující data z Tabulky 2

5.3 Odebírání

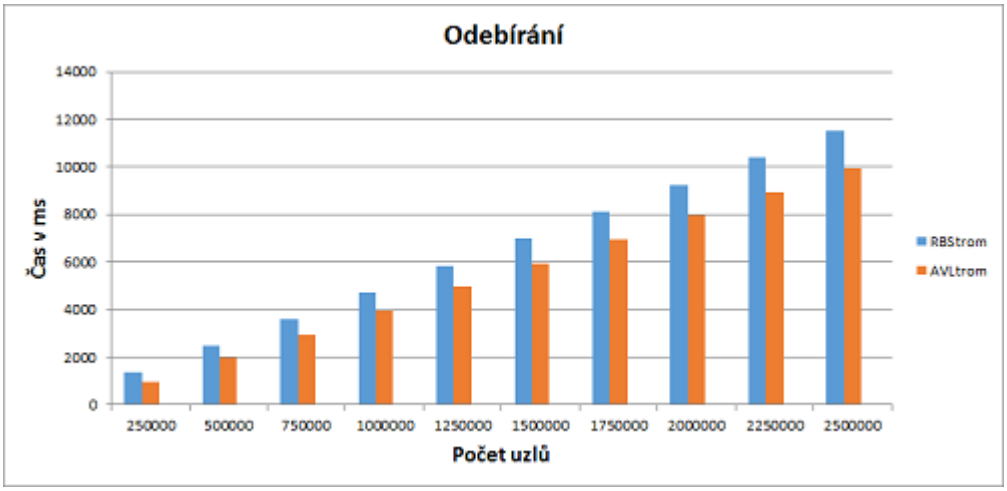
Pro měření dat v operaci *Odebírání* byl postup následující:

- nejprve bylo do každého stromu vloženo 5 mil. uzlů
- poté odebráno 2,5 mil. uzlů
- stromy byly opět naplněny na 5 mil. uzlů
- postup byl opakován

Souhrn naměřených výsledků *Odebírání* uzlů ze stromů v tabulce 5.3.
Pro vizuální představu ukázka grafu *Odebírání* obr. 29.

Počet uzlů	**ČČ[ms]**	**AVL[ms]**
250 000	1 370	959
500 000	2 466	1 953
750 000	3 577	2 955
1 000 000	4 700	3 950
1 250 000	5 844	4 942
1 500 000	6 976	5 948
1 750 000	8 110	6 941
2 000 000	9 244	7 941
2 250 000	10 385	8 927
2 500 000	11 516	9 927

Tabulka 3: Tabulka srovnávající časy operace *Odebírání* v jednotlivých stromech



Obrázek 29: Graf znázorňující srovnání rychlosti jednotlivých stromů při *Odebírání* uzlů reprezentující data z Tabulky 5.3

Odebírání všech prvků ze stromu o velikosti 10 mil. uzlů trvalo zhruba 41 sekund, u ČČ stromu pak 49 sekund.

5.4 Závěr časové analýzy

Z naměřených dat vyplývá, že AVL stromy jsou rychlejší v hlavních operacích (*Vyhledávání, Přidávání, Odebírání*) než ČČ stromy. Je to způsobenou výslednou výškou AVL stromů, která je menší než u ČČ stromů. Při menším počtu uzlů ve stromech jsou rychlosti skoro stejné, ale při větším počtu uzlů mají AVL stromy jednoznačně lepší čas. Při měření se může stát, že výsledný čas vyjde lépe pro ČČ stromy, proto jsou měření opakována a zprůměrována.

Při měření operace *Přidávání* a *Odebírání*, kde byly stromy naplněny na 10 mil. uzlů, a poté odebrány, je časový rozdíl zhruba 1 minuta ve prospěch *Odebírání*. Je to způsobeno vytvářením nových uzlů při operaci *Přidávání*, vlastnostmi jazyka C# a automatickou správou paměti.

Optimalizací algoritmů u používaných operacích by se jistě měřený čas zmenšil. Například u operace *Vyhledávaní* by se místo rekurzivního algoritmu mohl použít cyklus, u operace *Přidávání* a *Odebírání* by jsme mohli využívat minimálního nebo maximálního uzlu (získáme ho operací *Minimum, Maximum*). Tento uzel by jsme si pamatovali a při vyhledávání, které probíhá na začátku těchto operací, by se začínalo u tohoto uzlu.

6 Uživatelská příručka

Následující část textu popisuje uživatelskou část programu. Popisuje program od jeho instalace a požadavků až po ovládací části v grafickém rozhraní.

6.1 Požadavky a instalace

Pro správnou instalaci aplikace je zapotřebí:

- operační systém Windows 7 a novější (doporučen Windows 8.1)
- .NET Framework ve verzi 4.5 a vyšší

Instalaci spustíme instalačním souborem Setup.exe ve složce bin na instalačním CD.
Program spustíme souborem Trees.exe z adresáře určeného při instalaci nebo z plochy ikonou Stromy.exe.
Pro zapnutí programu a běh časové analýzy je nutné mít nainstalovaný MS Excel v počítači!.

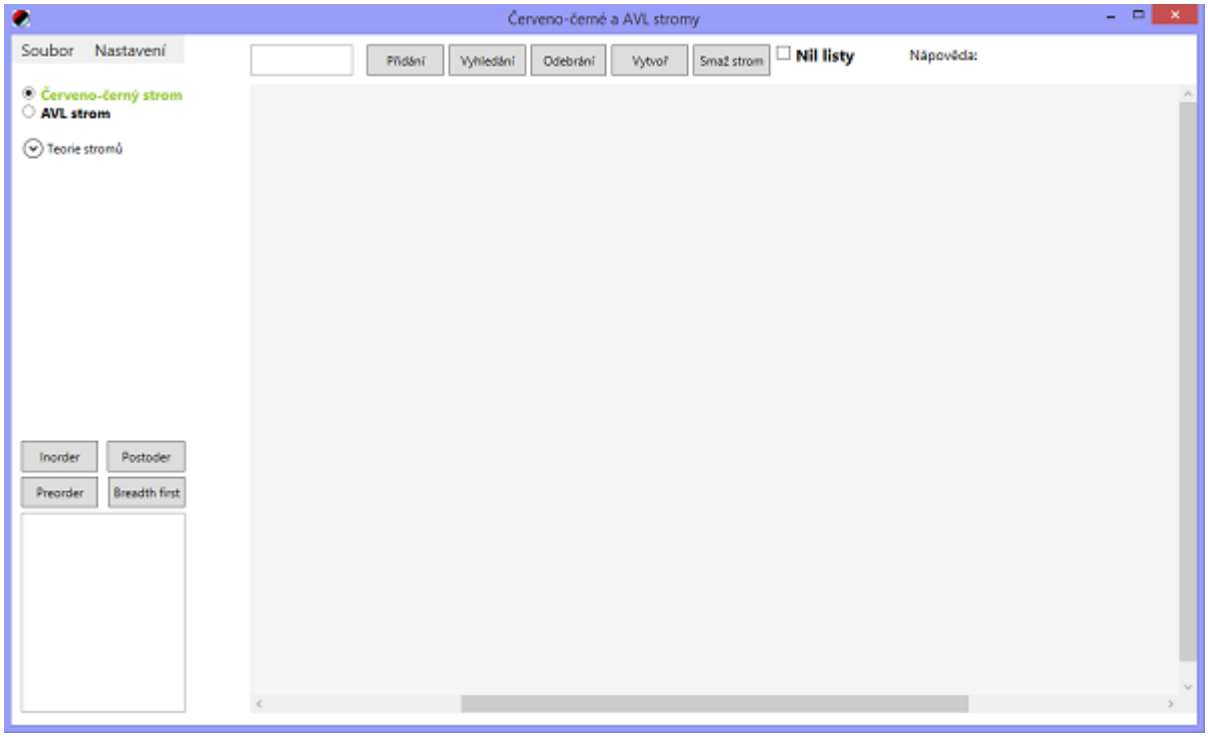
6.2 Hlavní okno programu

Po zapnutí programu se nám zobrazí hlavní okno (obr. 30). Jsou zde všechny důležité informační a ovládací prvky uspořádané do skupin. Tyto skupiny rozdělíme následovně:

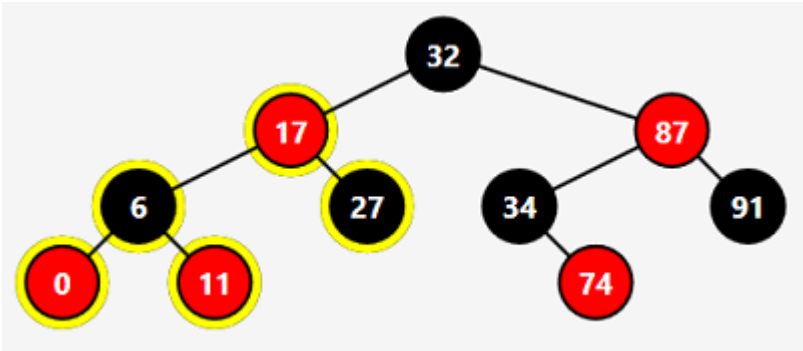
- Menu (obr. 32)
- Ovládací prvky plátna (obr. 35)
- Teorie (obr. 36)
- Průchody stromem (obr. 37)

6.2.1 Canvas

Pro uživatele nejzajímavější část programu. Probíhá zde výsledek celého programu. Jsou zde zobrazeny stromy a jejich práce s nimi. Pozadí tohoto plátna je šedou barvu, aby vznikl kontrast mezi *Canvasem* a ovládacími prvky. Pro uživatele je tu možnost přiblížení a oddálení uzlů, které se provádí kolečkem u myši. Když se velikost stromu blíží velikosti kreslícího plátna, plátno se zvětší a provede se posun stromu. Ukázka *Canvasu* s červeno-černým stromem na obr. 31.



Obrázek 30: Hlavní okno programu



Obrázek 31: Ukázka kreslícího plátna z programu

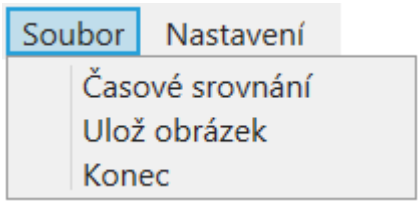
6.2.2 Menu

Hlavní menu aplikace obsahuje pouze potřebné funkce. Struktura menu obsahuje:

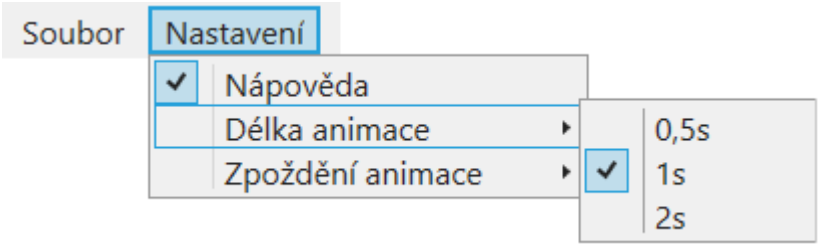
- **Soubor** (obr. 32)
  - **Časové srovnání** – K otevření nového okna pro časovou analýzu.
  - **Ulož obrázek** – Slouží k uložení *Canvasu*. Výsledkem je obrázek s bílým pozadím ve formátu *.png*. Pro uložení celého *Canvasu* je zapotřebí nemít přiblížené nebo oddálené uzly.

- **Konec** – ukončí program

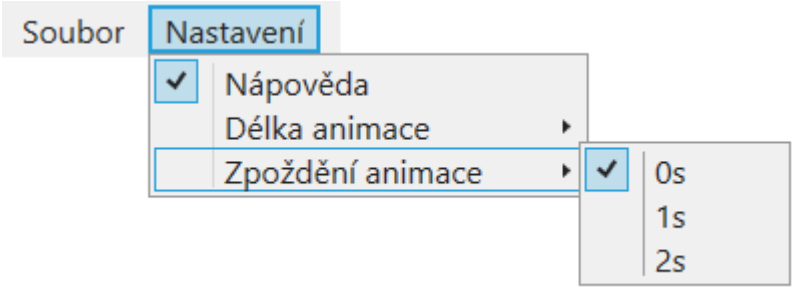
- **Nastavení** (obr. 33, obr. 34)
  - **Nápověda** – Zobrazuje postup operací při práci se stromem, lze ji vypnout/zapnout.
  - **Délka animace** – Čas potřebný k provedení animace.
  - **Zpoždění animace** – Začátek animace je posunut o nastavený čas.



Obrázek 32: Nabídka menu



Obrázek 33: Nastavení – délka animace



Obrázek 34: Nastavení – délka zpoždění

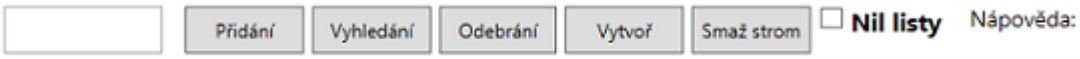
6.2.3 Ovládací prvky pro stromy

Ovládací prvky jsou závislé na výběru typu stromu. Tento výběr se provede přepnutím *Checkboxu* mezi AVL a červeno-černými stromy. Jako výchozí nastavení jsou nastaveny červeno-černé. Dále je zde *Textbox*, do kterého uživatel zapisuje hodnotu uzlu pro libovolnou operaci se stromem. Tato hodnota je omezena

na rozsah 1 – 100. Při špatném zadání je uživatel informován. Hlavní ovládací prvky jsou tlačítka pro provádění operací se stromem:

- **Přidání** – vytvoří a vloží uzel se zadanou hodnotou. Pokud je *Textbox* prázdný, vloží se uzel s náhodnou hodnotou v rozmezí 0 – 100. Omezení je z důvodu velikosti písma u hodnoty uzlů a přehlednosti.
- **Odebrání** – odebere ze stromu zadanou hodnotu pokud se ve stromu vyskytuje
- **Vyhledání** – zanimuje hledání zadané hodnoty ve stromu, o výsledku operace je uživatel informován
- **Vytvoř** – vytvoří daný typ stromu bez animací, počet uzlů je dán hodnotou získanou z *Textboxu*, která je omezena na hodnotu 1 – 20. Je to z důvodu výsledné přehlednosti stromu, uživatel dále může pracovat s tímto stromem.
- **Smaž** – smaže aktuální strom

Jednotlivé postupy operací při práci se stromy se zapisují do **Nápovědy**, lze ji vypnout nebo zapnout z menu **Nastavení**.



Obrázek 35: Ovládací prvky hlavního okna

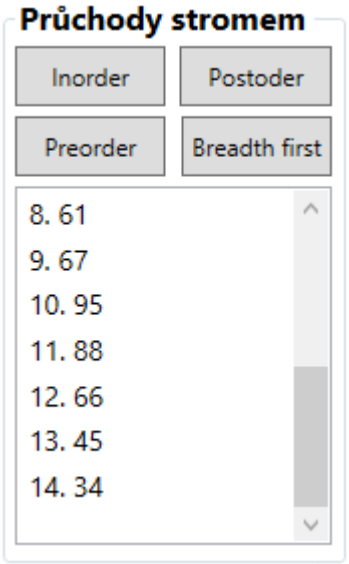
6.2.4 Teorie stromů

Pro teorii stromů se uživateli otevře nové okno. V něm jsou informace o stromech a složitostech jednotlivých operací. Jsou zde ukázkové příklady pro vysvětlení principu při *Přidávání* nebo *Odebírání* uzlů ze stromu. Dále je zde princip vyhledávání a rotací. Ukázka z programu na obr. 36.

6.2.5 Průchody stromem

Do této sekce patří tlačítka: **Inorder**, **Preorder**, **Postorder** a **Breadth first**. Slouží k postupnému průchodu stromem a zobrazení tohoto průchodu s výpisem čísel do pomocného *Listboxu*. Hodnoty uzlů jsou zapsány za číslo pořadí. Na konci operace je uživatel informován o konci průchodu. Všechny tyto průchody byly probírány v předmětu *ALM2*.

Obrázek 36: Teorie stromů – obsahuje informace o stromech, principy postupů operací, časové složitosti



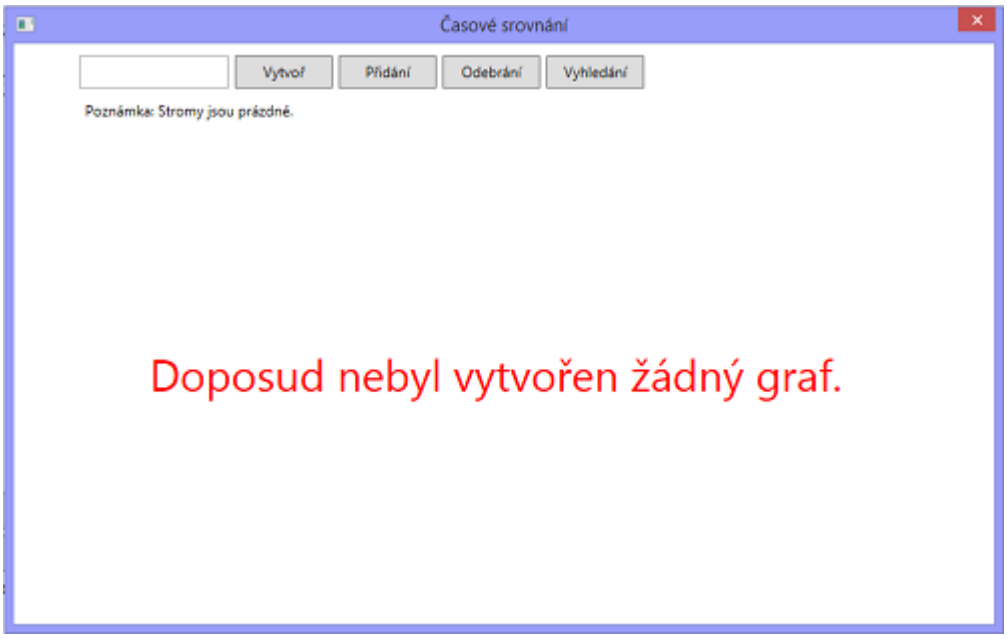
Obrázek 37: Příklad průchodu stromem

6.3 Formulář pro časové porovnání

Hlavní okno pro časovou analýzu obsahuje části jako je *Textbox* sloužící pro zadání maximálního počtu uzlů v grafu a část se čtyřmi tlačítky pro nastavení výstupu grafu:

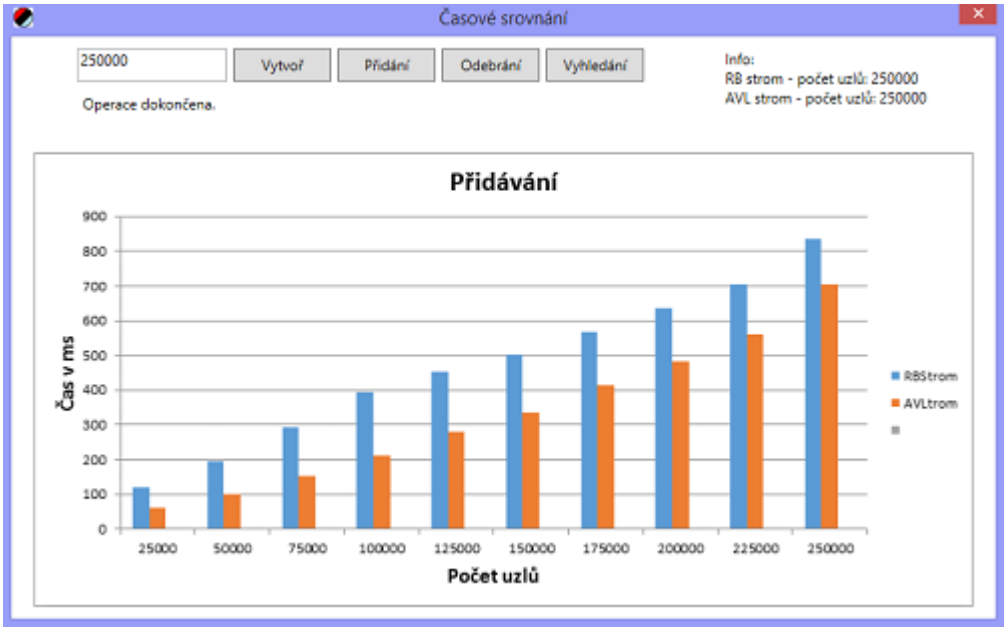
- **Textbox** – k zadání čísla od uživatele

- **Vytvoř** – podle zadaného čísla vytvoří stromy o zadaném počtu uzlů, nevytváří graf
- **Přidání** – podle zadaného čísla vytvoří stromy o zadaném počtu uzlů, vytváří graf
- **Odebírání** – podle zadaného čísla odebere tento počet ze stromů, pokud je tohle číslo větší jak počet uzlů ve stromu, uživatel je upozorněn
- **Vyhledání** – stromy musí obsahovat alespoň 10 000 uzlů, pak podle zadaného čísla je tento počet vyhledáván



Obrázek 38: Okno pro časovou analýzu

Po vytvoření stromů nebo různých grafů se objeví informační text. Je zde vypsán počet uzlů ve stromech. Podle těchto informací se může uživatel informovat o počtu uzlů a na základě těchto čísel odebírat nebo vyhledávat ve stromech.



Obrázek 39: Ukázka příkladu časové analýzy

Závěr

Výsledkem práce je jednoduchý a přehledný program, který studentům demonstruje operace na binárních vyhledávacích stromech, konkrétně AVL a červeno–černých stromech. Program znázorňuje průběhy operací *Vyhledávání, Přidávání, Odebírání* a dalších pomocných operací. Z naměřených dat získaných z časové analýzy vidíme, že AVL stromy jsou méně časově náročné, než červeno–černé stromy. Při programování tohoto programu jsem získal nové zkušenosti o tvorbě grafických programů s novější technologií WPF. Výsledný program lze využít k výukovým účelům.

Conclusions

The result of the bachelor's thesis is a simple and clear program that demonstrates operations on binary search trees to students, specifically the AVL and red-black trees. The program shows the flow of operations *Search, Insert, Delete* and other auxiliary operations. From the data obtained from the timing analysis we can see that AVL trees are less time-consuming than red-black trees. During programming this program, I have gained new experience in creating of graphic programs with newer technology WPF. The final program can be used for educational purposes.

Bibliografie

[1] Dvorský J. *Algoritmy I.* 2007
Dostupný také z: ⟨*http : //www.cs.vsb.cz/dvorsky/Download/SkriptaAlgoritmy/-Algoritmy.pdf*⟩

[2] Bělohlávek R., Vychodil V. *Diskrétní matematika pro informatiky 2* 2006
Dostupný také z: ⟨*http : //belohlavek.inf.upol.cz/vyuka/dm2.pdf*⟩

[3] Microsoft MSDN
Dostupný také z: ⟨*http : //msdn.microsoft.com*⟩

[4] AVL rekurzivní implementace
Dostupný také z: ⟨*http : //blog.blackbam.at/2012/05/04/avl − tree − implementation − in − java/*⟩

[5] Odebírání z ČČ stromu
Dostupný také z: ⟨*http : //en.literateprograms.org/Red − black_tree_(Java)-#chunkdef nodeconstructor*⟩

[6] Excel export
Dostupný také z: ⟨*http : //csharp.net − informations.com/excel/csharp − excel − chart − picturebox.htm*⟩

[7] AVL rotace
Dostupný také z: ⟨*http : //www.superstarcoders.com/blogs/posts/efficient − avl − tree − in − c − sharp.aspx*⟩

[8] Robert Sedgewick *Algorithms in C++: Parts 1–4: Fundamentals, Data Structure, Sorting, Searching*

Addison-Wesley 1998. ISBN: 0-201-35088-2.

[9] Stolerman A. *Red-Black Trees – Insertion, Deletion*
Dostupný také z: ⟨*http : //www.stolerman.net/studies/cs521/red_black_trees.pdf*⟩

[10] Red-Black Tree
Dostupný také z: ⟨http://www.cs.nthu.edu.tw/ wkhon/algo08-tutorials/tutorial-redblack.pdf⟩

[11] AVL Trees
Dostupný také z: ⟨*http : //www.dcs.gla.ac.uk/ pat/52233/slides/AVLTrees1x1.pdf*⟩

7 Obsah přiloženého CD/DVD

bin/
: Obsahuje složku `instalace/`, ve které je instalátor Setup.exe programu s potřebnými soubory pro správné nainstalování. Složka `spustit/` obsahuje program Trees.exe spustitelný přímo z CD a potřené soubory.

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).

src/
: Kompletní zdrojové kódy programu se všemi potřebnými soubory pro bezproblémové spuštění programu (v ZIP archivu).

readme.txt
: Instrukce pro instalaci a spuštění programu.