FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Implementace a porovnání různých typů vyhledávacích stromů
Student:	Michal Štěpánek
Vedoucí:	doc. Ing. Ivan Šimeček, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Teoretická informatika
Katedra:	Katedra teoretické informatiky
Platnost zadání:	do konce zimního semestru 2022/2023

Pokyny pro vypracování

1) Nastudujte následující typy vyhledávacích stromů: AVL, červeno-černý, AA, splay, treap, (a,b), BB-α, scapegoat a nevyvážený BVS [1-2].
2) V jazyce C++ implementujte typy uvedené v bodu 1) pomocí spojových seznamů. Pro typy AA a nevyvážené BVS vytvořte více různých implementací.
3) Připravte různé typy vstupních dat, lišící se v množství operací vyhledávacích stromů a poměru operací a algoritmy, které využívají vyhledávací stromy (např. tree sort).
4) Porovnejte výkonnost jednotlivých implementací z bodu 2) a implementací vyhledávacích stromů v knihovnách jazyku C++ pro data a algoritmy z bodu 3).
5) Diskutujte výsledky z bodu 4).

[1] Sedgewick, R., & Wayne, K. (2011). Algorithms (4th Edition) (4th ed.). Addison-Wesley Professional.
[2] CORMEN, Thomas H. Introduction to algorithms. 3rd ed. Cambridge: MIT Press, c2009. ISBN 978-0-262-03384-8.

Elektronicky schválil/a doc. Ing. Jan Janoušek, Ph.D. dne 13. srpna 2021 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Implementace a porovnání různých typů vyhledávacích stromů

Michal Štěpánek

Katedra teoretické informatiky
Vedoucí práce: doc. Ing. Ivan Šimeček Ph.D.

11. května 2022

Poděkování

Poděkovat bych chtěl především panu docentovi Ing. Ivanu Šimečkovi Ph.D. za vedení práce a dále všem, kteří mě během studia na fakultě vyučovali a v neposlední řadě přátelům a rodině, která mě během doby studia podporovala.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu) licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 11. května 2022 . . . . . .

Odkaz na tuto práci

Štěpánek, Michal. *Implementace a porovnání různých typů vyhledávacích stromů.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2022.

Abstrakt

Tato práce se zabývá datovou strukturou vyhledávací strom, především jejími typy. V práci je detailně popsán vyhledávací strom, jednotlivé typy vyhledávacích stromů, jejich výhody, nevýhody, vlastnosti a příklady využití.

Literární rešerše se zabývá popisem a rozborem typů vyhledávacího stromu, praktická část práce navazuje na rešerši implementací uvedených typů v jazyku C++, přípravou dat, pomocí kterých budou implementace testovány, a porovnáním výkonností jednotlivých implementací. Výkonnost implementací uvedených typů je měřena pro připravená data, připravené scénáře a náhodná data na univerzitním výpočetním svazku STAR.

Vytvořené implementace dosahují v porovnání s implementacemi z knihoven C++ podobných, v některých případech i lepších, výsledků. Porovnání také ukazují případy, v kterých je vhodné typy využít nehledě pouze na asymptotické složitosti.

Klíčová slova vyhledávací strom, binární vyhledávací strom, AVL strom, červeno-černý strom, AA strom, splay strom, treap, scapegoat strom, váhově vyvážený strom, (a,b)-strom, C++

Abstract

This thesis is about data structure search tree, especially about different types of search trees. This thesis describes chosen search tree types, their advantages, disadvantages, traits and examples of their usage.

Analytical part contains description and analysis of search tree and different search tree types, practical part follows up with implementation of types researched in analytical part in C++ language, generation of data for testing and comparison of implemented types for generated data on university computer cluster STAR.

Created implementations achieve similar and in some cases better results than implementations imported from C++ libraries. Comparison also shows use cases of different search tree types notwithstanding only on asymptotic complexities.

Keywords search tree, binary search tree, AVL tree, red-black tree, AA tree, splay tree, treap, scapegoat tree, weight-balanced tree, (a,b)-tree, C++

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Jelikož je vyhledávací strom často používaná datová struktura a správný výběr typu vyhledávacího stromu je klíčový pro časovou a paměťovou složitost programu, je výběr správného typu vyhledávacího stromu velmi důležitý. Tato struktura se také často využívá pro implementaci abstraktních datových struktur (například pro implementaci prioritní fronty nebo pro implementaci asociativních polí), další využití nacházejí vyhledávací stromy v databázích nebo při řazení prvků množiny algoritmem `tree sort`.

V rámci teoretické části práce je představena datová struktura vyhledávací strom a možnosti implementace této datové struktury. Teoretická část je rozdělena na dvě kapitoly. V první kapitole se nachází definice, popis a možnosti implementace vyhledávacích stromů. Druhá kapitola pojednává o jednotlivých typech vyhledávacích stromů, jejich výhod, nevýhod a dalších vlastností.

Praktická část je také rozdělena na dvě kapitoly a začíná kapitolou, v které je popsána implementace typů uvedených v teoretické části a příprava vstupních dat pro měření rychlosti. V poslední kapitole dochází k porovnání vytvořených implementací pro připravená vstupní data a scénáře.

Cílem teoretické části bakalářské práce je rešerše následujících typů vyhledávacích stromů: AVL, červeno-černý, AA, splay, treap, (a,b), BB-α, scapegoat a nevyvážený BVS.

Dílčím cílem praktické části je implementace výše uvedených typů vyhledávacích stromů v jazyce C++. Dalším cílem je příprava vhodných vstupních dat a algoritmů pro testovaní vytvořených implementací. Navazujícím úkolem na předchozí body je porovnání výkonnosti implementací v knihovnách jazyku C++ a implementací vytvořených v rámci této práce pro připravená data a algoritmy a následná diskuze dosažených výsledků.

Přínosem této práce je srozumění čtenáře s jednotlivými typy vyhledávacích stromů, jejich vlastnostmi a případy, pro které je vhodné uvedené typy využít. Čtenář by po přečtení práce měl být schopen vybrat ideální typ vyhledávacího stromu při práci, kde je vhodné jejich využití. Dalším přínosem

je knihovna s uvedenými typy vyhledávacího stromu, která je v rámci práce vytvořena.

Kapitola 1

Vyhledávací strom

Pro definici, popis vyhledávacího stromu a popis jeho typů se musejí nejdříve uvést některé definice.

1.1 Základní pojmy

Definice 1 (Strom)**.** Graf G je strom právě tehdy, když je graf G souvislý a neobsahuje kružnici.

Definice 2 (Zakořeněný graf)**.** Dvojice (G, v), kde $G = (V, E)$ je graf a $v \in V$, je zakořeněný graf s kořenem v.

Definice 3 (Cesta v grafu)**.** Cesta v grafu $G = (V, E)$ mezi vrcholy v_0 a v_k je posloupnost vrcholů $v_0, v_1, \ldots, v_k$, kde $\forall i \in \{0, \ldots, k\} : v_i \in V$, taková, že pro $\forall j \in \{0, \ldots, k-1\} : \{v_j, v_{j+1}\} \in E$ a její délka je k. Pokud jsou vrcholy cesty $v_0 = v_k$, pak je délka této cesty 0.

Pokud ve stromu existuje cesta $v_0, v_1, \ldots, v_{k-1}, v_k$, kde v_0 je kořen stromu, je vrchol v_{k-1} označován jako rodič vrcholu v_k a vrchol v_k synem vrcholu v_{k-1}. Existuje-li i vrchol v_{k-2} je tento vrchol označován jako prarodič vrcholu v_k.

Definice 4 (Hloubka vrcholu)**.** Hloubka vrcholu $v \in V$ ve stromu $T = (V, E)$, označována $h(v)$, je délka cesty mezi vrcholem v a kořenem stromu T.

Definice 5 (Hloubka stromu)**.** Pro hloubku h stromu $T = (V, E)$ platí $h = \max\{h(v) | v \in V\}$.

Definice 6 (Podstrom stromu)**.** Podstrom $T' = (V', E')$ ve vrcholu v, označován jako $T(v)$, stromu $T = (V, E)$ je zakořeněný strom, pro který platí, že V' obsahuje kořen v a vrcholy, mezi kterými v grafu T existuje taková cesta, jejíž všechny vrcholy mají větší nebo stejnou hloubku jako v.

Nyní je již možné uvést grafovou definici vyhledávacího stromu. Pro lepší pochopitelnost a konzistenci bude práce obsahovat všechny definice a popisy operací vyhledávacího stromu pro operátor $<$, vyhledávací stromy lze použít i pro některé jiné relační operátory.

Definice 7 (Vyhledávací strom)**.** Vyhledávací strom je zakořeněný strom $G = (V, E)$ s určeným pořadím synů každého vrcholu a vrcholy vyhledávacího stromu se dělí na vnitřní a vnější.

Vnitřní vrcholy mají libovolný nenulový počet klíčů. Vrchol obsahující klíče $x_1, \ldots, x_k$ má $k+1$ synů $s_0, \ldots, s_k$, pro tyto klíče a jejich syny platí: $T(s_0) < x_1 < T(s_1) < x_2 < \cdots < x_{k-1} < T(s_{k-1}) < x_k < T(s_k)$, kde $T(s_i)$ je množina všech klíčů v podstromu určeném vrcholem s_i.

Vnější vrcholy neobsahují žádná data ani klíče, nemají žádné syny a jsou to listy vyhledávacího stromu.

Vnější vrcholy se v některých zdrojích, například v [1, 2], neuvádí, jelikož neobsahují klíče. Pro přehlednost nebudou znázorněny ani na ilustracích v této práci.

Dalšími důležitými pojmy pro popis vlastností vyhledávacího stromu a operací jeho typů jsou asymptotické, jejichž definice a popis se nachází v [3], a amortizované složitosti, které jsou popsány v [4].

1.2 Operace vyhledávacího stromu

Dle [5] jsou základními operacemi vyhledávacího stromu T operace:

- **insert(x)** – operace sloužící pro vložení nového prvku do stromu. Pokud strom T neobsahuje vrchol s klíčem x, pak je vytvořen nový vrchol a vložen do stromu T na příslušné místo určené podle pravidel stromu, pokud již strom T obsahuje vrchol s klíčem x, pak se do stromu nový vrchol nevkládá a struktura zůstává nezměněna.

- **delete(x)** – operace sloužící pro odstranění prvku ze stromu. Pokud strom T neobsahuje vrchol s klíčem x, pak se žádný vrchol neodstraňuje a struktura se neupravuje, pokud ovšem strom T obsahuje vrchol s klíčem x, pak je tento vrchol odstraněn a strom je upraven tak, aby i nadále splňoval pravidla příslušného typu.

- **search(x)** (někdy označována jako **find(x)**) – operace pro vyhledávání ve stromu. Pokud strom T obsahuje vrchol s klíčem x, pak je tento vrchol operací vrácen.

Vzhledem k průchodu stromem po cestě mezi kořenem a vnitřním vrcholem, jejíž délka je maximálně h. Je spodní asymptotická časová složitost základních operací $\Omega(h)$.

Strom může obsahovat i rozšiřující operace, [6] uvádí jako nejčastější rozšiřující operace:

- **min()**, respektive **max()** – operace, která získá minimální, respektive maximální, klíč stromu T.
- **pred(v)**, respektive **succ(v)** – operace vrací předchůdce (nejvyšší nižší klíč), respektive následníka (nejnižší vyšší klíč), vrcholu v ve stromu T.
- **preorder()** – vypíše nebo uloží klíče při rekurzivním průchodu stromem, začne v kořeni stromu T a pro každý klíč, který je navštíven, nejdříve vypíše nebo uloží hodnotu navštíveného klíče, pak je tato operace provedena pro levého syna a na závěr je provedena pro pravého syna určeného navštíveným klíčem.
- **inorder()** – vypíše nebo uloží klíče při rekurzivním průchodu stromem, začne v kořeni stromu T a pro každý navštívený klíč se operace nejdříve provede pro levého syna určeného navštíveným klíčem, poté se vypíše nebo uloží hodnota navštíveného klíče, a na závěr je tato operace provedena i pro pravého syna, výstupem `inorder` průchodu je seřazená posloupnost klíčů ve vyhledávacím stromu.
- **postorder()** – vypíše nebo uloží klíče při rekurzivním průchodu stromem, funguje jako předchozí operace, ale nejdříve je operace provedena pro syny a až na závěr je vypsán nebo uložen navštívený klíč.

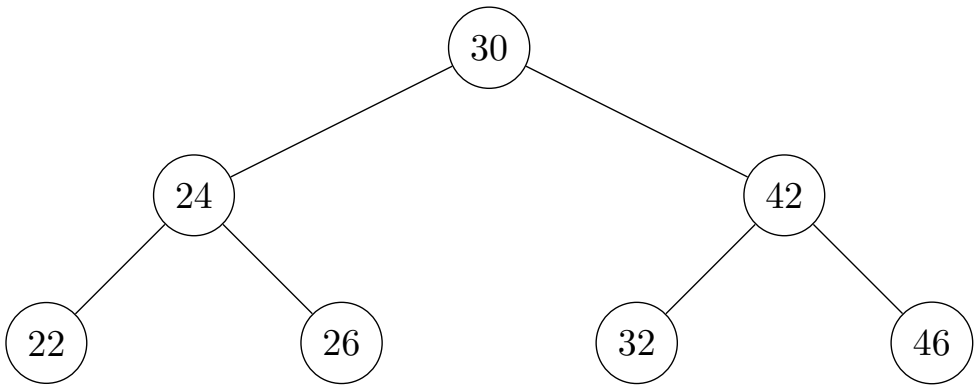
1.3 Využití

Vyhledávací strom je vhodná datová struktura pro reprezentaci dat, která by měla být po dobu používání seřazena, aby bylo umožněno výhodnější vyhledávání v množině dat. Jeho hlavní výhodou, pokud se jedná o vyvážený vyhledávací strom (jeho hloubka je $O(\log n)$), je asymptotická časová složitost základních operací $O(\log n)$, která je dle [7] nejlepší dosažitelná pro vyhledávání v množině. Výhodou oproti seřazenému poli je rychlejší provádění operací pro vkládání a odebírání. Vyhledávací stromy se proto často využívají v databázích. Další využití nacházejí například jako datová struktura pro implementaci abstraktních datových struktur (například slovník, množina, prioritní fronta) a v algoritmu `tree sort`, který seřadí data postupným vkládáním všech dat do vyhledávacího stromu a seřazená data pak získá `inorder` průchodem.

1.4 Možnosti implementace

O časové a paměťové složitosti konečné implementace vyhledávacího stromu rozhoduje hned několik faktorů, jedním z hlavních faktorů je způsob implementace. Způsob implementace rozhoduje o několika dolních hranicích časových

a paměťových složitostí operací, proto je i výběr způsobu implementace důležitý.



Obrázek 1.1: Příklad vyhledávacího stromu

1.4.1 Jednosměrný spojový seznam

Tento princip je velice jednoduchý na pochopení i na implementování. Stačí vytvořit jednoduchou strukturu či třídu, která bude reprezentovat vnitřní vrcholy a bude obsahovat atributy pro hodnoty klíčů a ukazatele na syny, hrany jsou realizovány pomocí nastavení ukazatelů na syny, které tvoří stejná struktura se správně nastavenými hodnotami atributů, pokud je nějaký syn vnitřního vrcholu vrchol vnější, pak je použit jako ukazatel na tohoto syna ukazatel nulový. Při práci s touto implementací si musí programátor pamatovat pouze ukazatel na kořen stromu.

Výhodou implementace pomocí jednosměrného spojového seznamu je jednoduchost implementace a dobrá rozšiřitelnost stromu o další vrcholy. Nevýhodou jsou neoptimální paměťové nároky na každý vrchol (pro každý vrchol se musí vytvořit dva ukazatele a jeden klíč) a zvýšení spodní časové složitosti některých operací (například vrácení seřazeného pole). Při implementaci některých vyvažovacích vyhledávacích stromů je také nevýhodný fakt, že vrcholy neobsahují informaci o rodičovi a je proto nutné ukládat cestu, kterou se při průchodu stromem postupuje, do paměti.

1.4.2 Obousměrný spojový seznam

Implementace obousměrným spojovým seznamem se odlišuje od implementace jednosměrným spojovým seznamem pouze tím, že vytvořená struktura obsahuje i referenci na rodiče, kořen stromu má jako ukazatel na rodiče nulovou referenci a vnější vrcholy zůstávají nulovými referencemi.

Obousměrný spojový seznam přebírá výhody a nevýhody od jednosměrného s drobnými odlišnostmi. Velikost struktury pro vrchol je větší o jeden

ukazatel, ale při implementaci složitějších vyhledávacích stromů již není nutné ukládat cestu, kterou se prochází, do paměti.

1.4.3 Seřazené pole

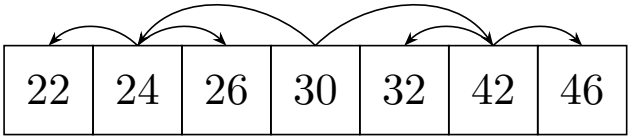
Jeden způsob implementace pomocí pole udržuje seřazené vrcholy podle klíčů, které strom obsahuje, v poli. Vzhledem k seřazení pole je jisté, že všechny prvky z levého podstromu vybraného klíče jsou v poli uloženy na nižším indexu, než má vybraný klíč, a všechny prvky pravého podstromu vybraného klíče jsou v poli na vyšším indexu.

V poli lze spočítat index kořene stromu vzhledem k typu vyhledávacího stromu. Také platí, že podstrom stromu je stromem stejného typu a je možné spočítat i index kořene daného podstromu, tedy index daného syna. Tato vlastnost se využívá pro průchod stromem. Pro ukázku uložení dat v paměti při implementaci pomocí seřazeného pole viz obrázek 1.2, ve kterém jsou uloženy klíče z binárního stromu na obrázku 1.1, šipky v obrázku ukazují na syny vrcholu.

Implementace vyhledávacího stromu, který bude ve všech svých hladinách maximálně naplněn po celou dobu používání, pomocí pole umožňuje ukládat pouze klíče vrcholů a dosáhnou tak optimálního uložení dat do paměti. Pak je ale nutné upravit některé operace, například nahrazením operací odstraňování a vkládání operací, která upraví klíč vrcholu a pak upraví strom tak, aby splňoval podmínky. Pokud strom nebude vždy maximálně naplněn, je nutno označit vnější vrcholy, které se nenachází v poslední hladině stromu.

Hlavní výhodou této implementace je možnost dosažení efektivního uložení klíčů do paměti bez zbytečného využití paměti navíc. Další dílčí výhodou je seřazenost klíčů v poli, tato vlastnost je vhodná například pro algoritmus `tree sort` a přímý přístup do množiny (výběr n-tého největšího prvku).

Hlavní nevýhodou je nutnost velikosti pole podle hloubky stromu, nikoliv podle počtu vrcholů jako u implementace pomocí spojových seznamů, a tedy přidáním jednoho vrcholu se mohou požadavky na paměť exponenciálně zvýšit. Další nevýhodou je velmi neefektivní využití paměti pro stromy, které jsou zároveň řídké a hluboké.

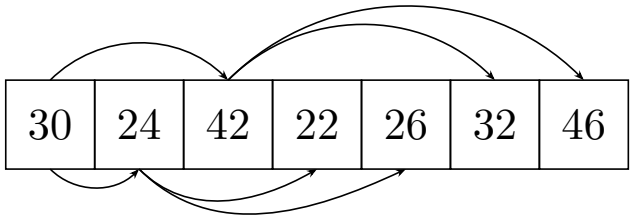


Obrázek 1.2: Uložení dat ze stromu na obrázku 1.1 při implementaci pomocí seřazeného pole

1.4.4 Pole po hloubkách

Další způsob implementace pomocí pole je ukládání po hloubkách stromu. Na první pozici pole je uložen kořen stromu, na dalších pozicích jsou jeho synové, dále synové synů atd. (pro binární strom je levý syn vrcholu na indexu n uložen na indexu $2n + 1$ a pravý na indexu $2n + 2$). Pro ukázku uložení dat v paměti při implementaci pomocí pole po hloubkách viz obrázek 1.3, ve kterém jsou uloženy klíče z binárního stromu na obrázku 1.1, šipky v obrázku ukazují na syny vrcholu.

Výhodou je opět možnost efektivního uložení klíčů do paměti. Další výhodou je jednodušší nalezení rodiče vrcholu a hlavní nevýhody jsou totožné s hlavními nevýhodami uchovávání dat v seřazeném poli, tedy neoptimální využití dat pro řídké a hluboké stromy a exponenciální nárůst paměťových požadavků.



Obrázek 1.3: Uložení dat ze stromu na obrázku 1.1 při implementaci pomocí pole po hloubkách

Kapitola 2

Typy vyhledávacích stromů

2.1 Nevyvážený binární vyhledávací strom

Definice 8 (Binární strom)**.** Binární strom je strom, který je zakořeněný a každý vrchol má nejvýše dva syny, z nichž jeden je levý a jeden je pravý.

Definice 9 (Binární vyhledávací strom)**.** Binární vyhledávací strom je binární strom, v jehož každém vrcholu v je uložen unikátní klíč $k(v)$. Dále platí $\forall a \in T(l(v)) : k(a) < k(v)$ a $\forall b \in T(r(v)) : k(v) < k(b)$, kde $l(v)$, respektive $r(v)$ představují levého, respektive pravého syna vrcholu v.

2.1.1 Základní operace

Dle [8] všechny základní operace BVS stromu postupně procházejí od kořenu k listům stromu a na každé hladině je navštíven pouze jeden vrchol. Postup BVS je důkladněji popsán níže.

Vyhledávání v nevyváženém BVS

Při vyhledávaní v nevyváženém BVS se postupně prochází stromem podle hodnoty hledaného klíče, pokud je hledaný klíč menší než klíč současného vrcholu, pokračuje vyhledávání v levém podstromu současného vrcholu, pokud je hledaný klíč větší než klíč současného vrcholu, pak je navštíven podstrom pravý a nenastane-li ani jedna z předchozích možností, pak je hledaný klíč roven současnému vrcholu a vrchol byl nalezen. Pokud je při vyhledávaní navštíven vrchol vnější, pak strom hledaný klíč neobsahuje.

Vkládání do nevyváženého BVS

Při vkládání nového prvku do nevyváženého BVS se postupně prochází stromem podle hodnoty vkládaného klíče jako při vyhledávání. Při nalezení vrcholu s klíčem totožným s vkládaným klíčem se operace neprovede, pokud

ale při průchodu stromem bude navštíven vrchol vnější, bude vložen na jeho pozici vrchol s vkládaným klíčem.

Odstraňování z nevyváženého BVS

Při odstraňování prvku z nevyváženého BVS se ve stromu vyhledává vrchol se shodným klíčem. Pokud nebyl vrchol nalezen, operace je neúspěšná a strom zůstává nezměněn. Pokud ale byl takový vrchol nalezen, pak je ze stromu odstraněn a strom je upraven tak, aby obsahoval podstromy odstraněného vrcholu. Toho je docíleno tím, že je-li jeden syn odstraňovaného vrcholu vrchol vnější, pak je nahrazen druhým synem, jsou-li oba synové vrcholy vnitřní, pak je možné použít pro nahrazení předchůdce nebo následníka v příslušném podstromu a jeden z nich je vybrán, a pokud nemá vrchol žádného potomka, pak není třeba vrchol nahrazovat a vrchol je ze stromu odstraněn bez nahrazení.

Nalezení předchůdce vrcholu v podstromu je stejné jako nalezení maxima v jeho levém podstromu, z toho že se jedná o maximum v podstromu je jisté, že tento vrchol nemá pravého syna a stačí tedy u rodiče přesouvaného vrcholu nahradit pravého syna za levého syna přesouvaného vrcholu, k přesouvanému vrcholu jsou pak přesunuti také synové vrcholu určeného k odstranění. Poté již není vrchol obsažen ve stromu a může být odstraněn. Při nahrazení vrcholu následníkem se postupuje symetricky, tedy jde o hledání minima v pravém podstromu a přesouvání pravého podstromu na jeho pozici u rodiče.

2.1.2 Vlastnosti nevyváženého BVS

Všechny operace postupují od kořene maximálně k listům, jejich časová složitost je určena hloubkou stromu a složitost těchto operací je v nejhorším případu $O(h)$. V ideálním případě je hloubka stromu $O(\log n)$ (viz další sekce), pak je složitost operací velmi příznivá, ovšem hloubka nevyváženého BVS může být až n, například pokud bude do stromu vždy vloženo nové maximum. Složitost operací proto může být až $O(n)$.

Vzhledem k závislosti složitosti operací na hloubce stromu je žádoucí vést kroky pro minimalizaci hloubky. Jedním způsobem je předem upravit pořadí vkládaných dat tak, aby byla hloubka co nejnižší, množinu lze seřadit tak, aby vytvořený nevyvážený BVS měl hloubku $O(\log n)$, viz 2.7.1, ovšem při dalším vkládání a odstraňování vrcholů může dojít ke změně hloubky a strom může mít nepříznivou hloubku $O(n)$. Proto se nevyvážený BVS využívá pouze pro velmi specifické případy a pro větší množiny dat se více vyplatí využít vyvažovací stromy, které kontrolují, zda strom splňuje své podmínky, které zaručují hloubku stromu $O(\log n)$.

Nevyvážený BVS, do kterého jsou vkládána náhodná data, mívá dle [9] hloubku $O(\log n)$, ovšem stále může dojít k případům, kdy bude hloubka stromu $O(n)$, avšak pro velká n je tato možnost velmi nepravděpodobná.

2.2 AVL strom

V roce 1962 byl AVL strom poprvé popsán ruskými matematiky Adělson-Veĺskiim a Landisem v [10] a jméno obdržel podle příjmení jeho autorů. AVL strom na rozdíl od nevyvážené BVS provádí vyvažovací operace, které zaručují hloubku v řádu $O(\log n)$, což vede v mnoha případech k rychlejšímu průběhu operací než při použití nevyváženého BVS.

Definice 10 (Dokonale vyvážený binární vyhledávací strom)**.** Binární vyhledávací strom je dokonale vyvážený, pokud pro každý jeho vrchol v platí: $||T(l(v))| - |T(r(v))|| \leq 1$.

Dokonale vyvážený binární vyhledávací strom s n vrcholy má hloubku $\lfloor \log_2 n \rfloor$, jelikož na každé cestě z kořene do listu je s každým vrcholem blíže k listu velikost jeho podstromu maximálně poloviční oproti podstromu určeného jeho rodičem.

Definice 11 (Hloubkově vyvážený binární vyhledávací strom)**.** Binární vyhledávací strom je hloubkově vyvážený, pokud pro každý jeho vrchol v platí: $|h(l(v)) - h(r(v))| \leq 1$.

Jelikož je udržování dokonale vyváženého BVS velmi náročné, často je používán právě hloubkově vyvážený BVS, kterému se říká AVL strom a který má pro n vrcholů maximálně o 45 % větší hloubku než dokonale vyvážený BVS, přesněji: $\lfloor \log_2 n \rfloor \leq h(n) \leq 1{,}4404 \log(n+2) - 0{,}328$, důkaz uveden v [11], a hloubka stromu je proto vždy $O(\log n)$.

Pro rozpoznání porušení vyvážení a na jaké straně stromu nebo podstromu je vyšší hloubka se používá vyvažovací faktor vrcholu $\delta(v) = h(l(v)) - h(r(v))$. Z definice 11 vyplývá, že aby vyhledávací strom byl AVL mohou být vyvažovací faktory všech vrcholů pouze -1, 0 nebo 1 a tedy pro $\delta(v)$ v AVL platí:

$$\delta(v) = \begin{cases} 1 & \text{pokud je levý podstrom vrcholu hlubší,} \\ 0 & \text{oba podstromy vrcholu mají stejnou hloubku,} \\ -1 & \text{pokud je pravý podstrom vrcholu hlubší.} \end{cases}$$

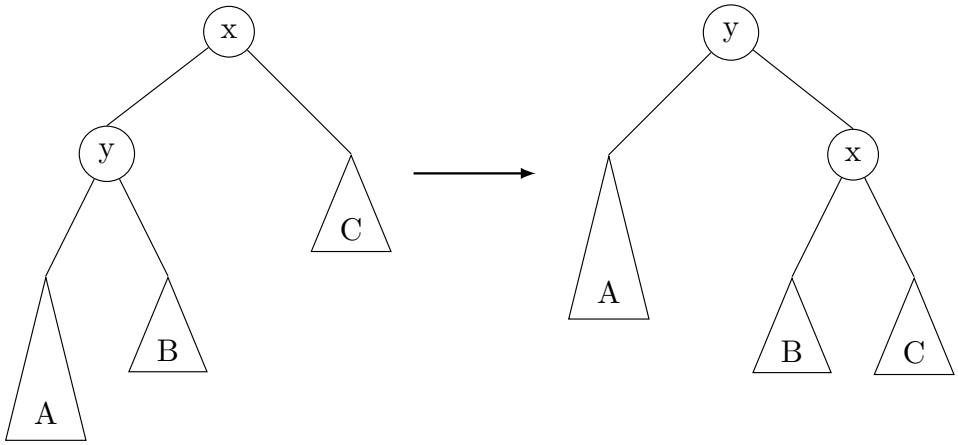
2.2.1 Vyvažovací operace

Pokud nastane situace, při které má vyvažovací faktor některého vrcholu jinou hodnotu, než které jsou povolené v AVL stromu, je nutné strom upravit tak, aby byl znovu hloubkově vyvážený. Tyto situace jsou řešeny rotacemi, které pomocí prohození rodiče se synem a přesunutím jejich podstromů tak, aby strom zůstal nadále vyhledávacím stromem, ze stromu opět udělají AVL strom.

Jednoduché rotace

Jednoduché rotace fungují jako otočení orientace hrany a následné přesunutí podstromu tak, aby výsledný strom splňoval podmínky stromu určené z definice.

- **Pravá rotace** – přesune pravý podstrom levého syna kořenu podstromu, na kterém je rotace prováděna, na pozici levého podstromu kořenu a levého syna přesune na pozici kořenu. Následně nastaví vrchol, ve kterém je rotace prováděna, jako pravého syna nového kořenu, kterým je jeho původní levý syn. Pravé rotace je znázorněna na obrázku 2.1.
- **Levá rotace** – je symetrická k pravé rotaci. Pro představu lze použít obrázek 2.1 s opačným pořadím grafů.

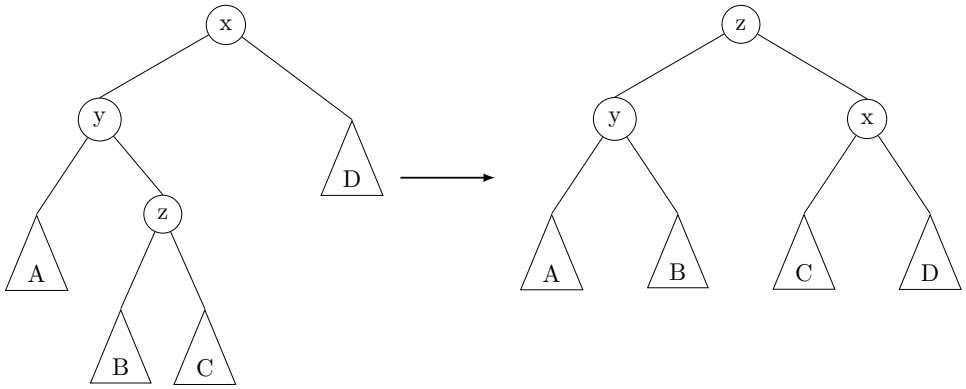


Obrázek 2.1: Pravá rotace

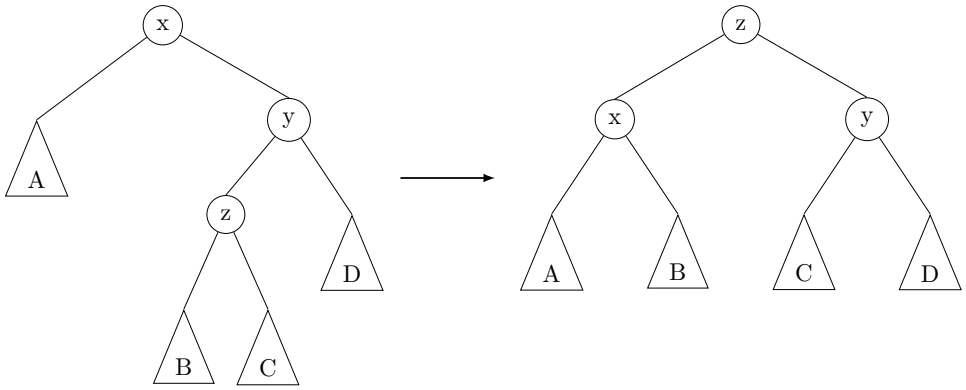
Dvojité rotace

Může nastat situace, kdy ani jedna z jednoduchých operacích neudělá z nevyváženého stromu AVL strom, příkladem takové situace jsou výchozí pozice v obrázcích 2.2 a 2.3. Tyto situace se dají řešit těmito kombinace jednoduchých operací:

- **LR rotace** – nejdříve se provede levá rotace v levém synovi a poté se na vrchol, ve kterém je LR rotace prováděna, provede pravá rotace, viz obrázek 2.2.
- **RL rotace** – je symetrická k LR rotaci a v této rotaci se nejdříve provede pravá rotace v pravém synovi s následnou levou rotací v kořeni podstromu, viz obrázek 2.3.



Obrázek 2.2: LR rotace



Obrázek 2.3: RL rotace

Při rotacích může dojít ke změně hloubky podstromu a případnou změnu hloubky je nutné propagovat směrem ke kořenu, také cestou ke kořenu může nastat situace, při které podstrom v daném vrcholu nebude hloubkově vyvážen, pak je nutné provést další rotace pro vyvážení stromu.

2.2.2 Základní operace

Základní operace probíhají jako v nevyváženém BVS pouze s tím rozdílem, že v některých operacích může nastat změna vyvažovacího faktoru vrcholu, pak je nutné zkontrolovat, zdali je vyvažovací faktor validní i po změně, nebo zda je nutné provést rotaci ve vrcholu. Vzhledem k principu vyhledávací operace nemůže být při vyhledávání změněna hloubka žádného podstromu a operace zůstane nezměněna. Je nutné upravit pouze operace pro odstraňování a přidávání do stromu. Tyto operace se upraví tak, že se na konci operací provede kontrola, zdali není nutné provést rotaci a zdali nedošlo ke změně hloubky podstromu. Pokud došlo ke změně hloubky tohoto podstromu, pak se změna hloubky musí propagovat směrem ke kořenu stromu, dokud se bude hloubka

podstromů měnit. I během propagace může dojít k porušení hloubkové vyváženosti a strom v takové situaci musí být opraven další rotací.

Vkládání do AVL stromu

Po vložení do stromu mohou ve vrcholu x, nastat následující situace:

- $\delta(x) = -2$ – pravý podstrom vrcholu má vyšší hloubku než levý podstrom o 2 úrovně, pravý syn vrcholu x je označen jako y, pokud je $\delta(y) = -1$, pak je provedena levá rotace a podstrom je opět vyvážený, pokud je $\delta(y) = 1$, pak nestačí provést jednoduchou rotaci a musí být provedena RL rotace a situace, kde $\delta(y) = 0$ nemůže nikdy nastat, protože by se nezměnila hloubka podstromu určeného vrcholem y a nedošlo by k propagaci změny hloubky k vrcholu x.

- $\delta(x) = -1$, $\delta(x) = 0$ nebo $\delta(x) = 1$ – podstrom určený vrcholem x je hloubkově vyvážený i bez provádění rotací.

- $\delta(x) = 2$ – levý podstrom vrcholu má vyšší hloubku než pravý podstrom o 2 úrovně, levý syn vrcholu x je označen jako y, pokud je $\delta(y) = 1$, pak je provedena pravá rotace a podstrom je opět vyvážený, pokud je $\delta(y) = -1$, pak nestačí provést jednoduchou rotaci a musí být provedena LR rotace a situace, kde $\delta(y) = 0$ opět nemůže nikdy nastat.

Odstraňování z AVL stromu

Po odstranění vrcholu ze stromu mohou ve vrcholu x, který je rodičem odstraněného vrcholu či vrcholu, který byl použit k nahrazení, nastat následující situace:

- $\delta(x) = -2$ – pravý podstrom vrcholu má vyšší hloubku než levý podstrom o 2 úrovně, pravý syn vrcholu x je označen jako y, pokud je $\delta(y) = -1$ nebo $\delta(y) = 0$, pak je provedena levá rotace a podstrom je opět vyvážený, pokud je $\delta(y) = 1$, pak nestačí provést jednoduchou rotaci a musí být provedena RL rotace.

- $\delta(x) = -1$, $\delta(x) = 0$ nebo $\delta(x) = 1$ – podstrom určený vrcholem x je hloubkově vyvážený i bez provádění rotací.

- $\delta(x) = 2$ – levý podstrom vrcholu má vyšší hloubku než pravý podstrom o 2 úrovně, levý syn vrcholu x je označen jako y, pokud je $\delta(y) = 1$ nebo $\delta(y) = 0$, pak je provede pravá rotace a podstrom je opět vyvážený, pokud je $\delta(y) = -1$, pak nestačí provést jednoduchou rotaci a musí být provedena LR rotace.

2.2.3 Vlastnosti AVL stromu

Vzhledem k zaručené hloubce AVL stromu s n vrcholy $O(\log n)$ a nezměněné operaci vyhledávání zůstává složitost této operace $O(h) = O(\log n)$. Dále byly přidány rotace, jejichž složitost je vzhledem k tomu, že jde pouze o prohození synů, konstantní. Další operace, která byla přidána je výpočet $\delta(v)$. Pokud bude v každém vrcholu udržována informace o hloubce podstromu tvořeného tímto vrcholem, pak bude mít výpočet $\delta(v)$ složitost $O(1)$.

Operacím odstraňování a vkládání byla přidána propagace nové hloubky, která se propaguje cestou ke kořenu a propagace má tedy složitost maximálně $O(h) = O(\log n)$ a po cestě se může provést maximálně $O(h) = O(\log n)$ rotací a výpočtů $\delta(v)$, obě tyto operace mají konstantní časovou složitost, a tudíž mají operace vkládání a odstraňování nezměněnou složitost $O(h)$, což je vhledem k hloubce AVL stromu $O(\log n)$.

2.3 Červeno-černý strom

Dalším typem, který udržuje vyhledávací strom vyvážený pro dosažení lepší hloubky, je červeno-černý strom, který byl poprvé popsán v [12] jako upravení symetrických B-stromů (definice B-stromů se nachází v sekci 2.9) popsaných v [13], na binární stromy. Červeno-černý strom je tedy realizací (2,4)-stromu (viz 2.9) pomocí binárního stromu, kde červeně obarvené vrcholy představují klíče, které by ve (2,4)-stromu byly v jednom vrcholu.

Červeno-černý strom není ani dokonale ani hloubkově vyvážený, avšak pro hloubku h červeno-černého stromu s n vrcholy platí $h \leq 2\log(n+1)$, důkaz tvrzení je uveden v [1], a tudíž má červeno-černý strom hloubku $O(\log n)$. Na rozdíl od BVS je také nutné udržovat v každém vrcholu informaci o barvě vrcholu, která může být černá nebo červená.

Definice 12 (Červeno-černý strom)**.** Červeno-černý strom je binární vyhledávací strom, který splňuje následující podmínky:

1. Každý vrchol je obarven buď červeně nebo černě.

2. Kořen stromu je obarven černě.

3. Každý vnější vrchol je obarven černě.

4. Pokud je vnitřní vrchol obarven červeně, pak jsou jeho synové obarveni černě.

5. Pro každý vrchol u platí, že všechny cesty z vrcholu u do vnějších vrcholů stromu obsahují stejný počet černých uzlů.

2.3.1 Vyvažovací operace

Červeno-černý strom jako vyvažovací operace používá stejné rotace jako AVL strom a jejich popis lze nalézt v 2.2.1.

2.3.2 Základní operace

Červeno-černý strom musí splňovat všechna pravidla z definice 12 a operace, které mění strukturu stromu, mohou tato pravidla porušit, pro zachování hloubky $O(\log n)$ musí být při porušení těchto pravidel strom upraven tak, aby byl opět červeno-černým stromem.

Vyhledávací operace nijak strom neupravuje, tudíž může zůstat zůstat nezměněna. Upravit se ovšem musí operace vkládání a odstraňování, operace jsou upraveny tak, že po provedení operace stejným způsobem jako u BVS dojde ke kontrole, zdali nedošlo k porušení některého z pravidel a zdali není nutné strom upravit tak, aby opět splňoval podmínky červeno-černého stromu.

Vkládání do červeno-černého stromu

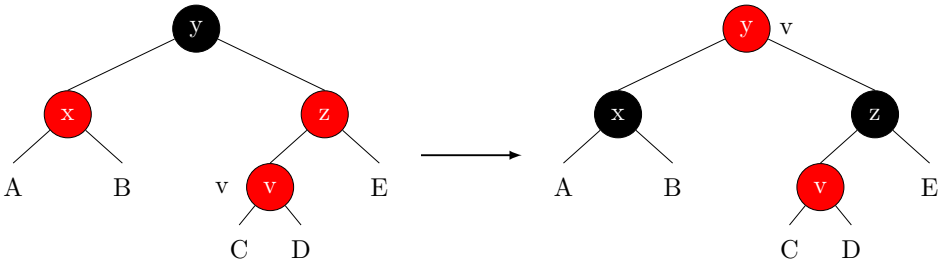
Vrchol je do stromu vložen stejným způsobem jako v nevyváženém BVS a po vložení je vrchol obarven červeně. Vzhledem k vložení červeně obarveného vnitřního vrcholu nemůže nikdy dojít k porušení pravidla 3, protože je vložený vrchol vrcholem vnitřním, a pravidel 1 a 5, protože je nově vložený vrchol obarven červeně. Tudíž jediná pravidla, která mohou být při vkládání porušena, jsou pravidla 2 a 4.

Do stromu je vložen vrchol v. Pokud je nově vložený vrchol kořenem stromu, pak došlo k porušení pravidla 2. Kořen je obarven černě a strom je červeno-černý. Dalším možným stavem je situace, kdy je rodič vloženého vrcholu obarven černě, pak nebylo porušeno žádné pravidlo a strom je stále červeno-černý, a stavy, které mohou nastat, pokud je rodič nově vkládaného vrcholu obarven červeně, jsou v [6] popsány takto:

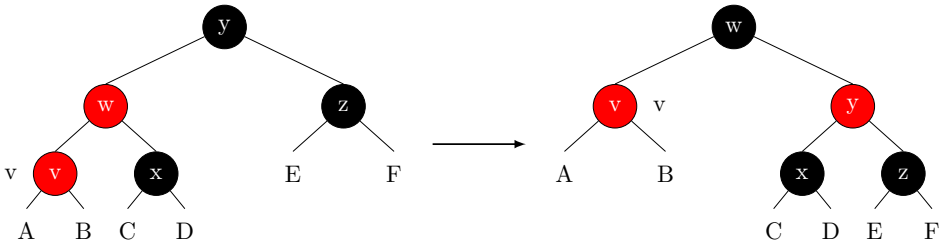
- Strýc vrcholu je také obarven červeně, jelikož byl strom před vložením červeno-černý, pak musí být prarodič nově vloženého vrcholu obarven černě. V tomto případu je přebarven prarodič červenou barvou a jeho synové obarveni černě, tímto byl podstrom upraven tak, aby znovu splňoval pravidlo 4 pro vrchol v. Vrchol, pro který bude oprava pokračovat je prarodič vrcholu v. Řešení tohoto stavu je znázorněno na obrázku 2.4.
- Strýc vrcholu je obarven černě a vrchol v je levým synem. Pokud je rodič vrcholu levým synem, pak je v prarodiči vrcholu v provedena pravá rotace, rodič vrcholu je obarven černou barvou a nový sourozenec vrcholu je přebarven červenou barvou. Pokud je rodič vrcholu levým synem, pak je nutné provést na prarodiče vrcholu v RL rotaci, po které je vrchol v přebarven černě a jeho pravý syn je obarven červeně. Řešení stavu pravou rotací je znázorněno na 2.5.

- Strýc vrcholu je obarven černě a vrchol v je pravým synem. Pokud je rodič vrcholu pravým synem, pak je v prarodičovi vrcholu provedena levá rotace, rodič vrcholu je přebarven černě a nový sourozenec vrcholu je přebarven červenou barvou. Pokud je rodič vrcholu levým synem, pak je nutné na prarodiče vrcholu v provést LR rotaci, po které je vrchol v obarven černě a jeho pravý syn je obarven červeně. Pro znázornění řešení stavu LR rotací viz 2.6.

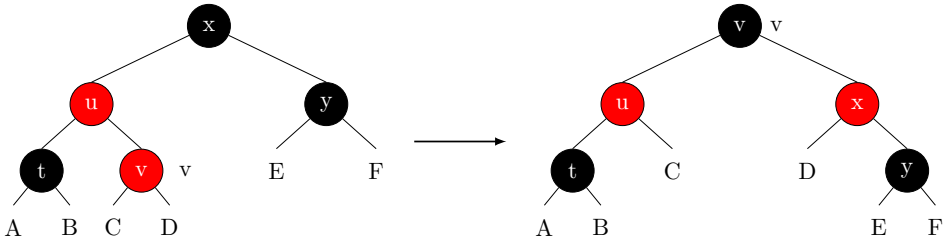
Pokud byl rodič vloženého prvku obarven červeně, pak oprava postupuje směrem ke kořenu, dokud strom nesplňuje pravidlo 4, nebo dokud se při opravě nedojde až ke kořenu, který, není-li obarven černě, bude tak obarven.



Obrázek 2.4: Znázornění řešení 1. stavu kontroly po vložení



Obrázek 2.5: Znázornění řešení 2. stavu kontroly po vložení pravou rotací



Obrázek 2.6: Znázornění řešení 3. stavu kontroly po vložení LR rotací

Odstraňování z červeno-černého stromu

Při odstraňování vrcholu v červeno-černém stromu se postupuje jako při odstraňování vrcholu v nevyváženém BVS s tím, že je-li vrchol v nahrazován, je vrchol přesunutý na jeho pozici, obarven barvou vrcholu v. Další úpravou je následná kontrola, zdali se jedná o červeno-černý strom, s případnými opravami.

Při vkládání je u opravy stromu přihlíženo k barvě strýce vrcholu, u odstraňování se přihlíží k barvě sourozence vrcholu. U operace odstraňování může dojít nejen k porušení pravidel 2 a 4, ale i k porušení pravidla číslo 5.

Ze stromu má být odstraněn vrchol u a pokud nejsou jeho synové vnější vrcholy, je nahrazen vrcholem v. Jednoduchá situace nastane, je-li odstraňovaný vrchol červený, jehož synové nejsou dva vnitřní vrcholy. Jsou-li synové vrcholu u pouze vnější vrcholy, pak je vrchol odstraněn bez náhrady, pokud je jeden jeho syn vrcholem vnitřním, pak je nahrazen tímto vrcholem. Dalším jednoduchým případem je situace, kdy je vrchol v červený, pak stačí přesunout vrchol v na pozici vrcholu u. V žádném z uvedených jednoduchých případů nemohlo dojít, vzhledem k zachování počtu a struktury černých vrcholů, k porušení pravidla červeno-černého stromu.

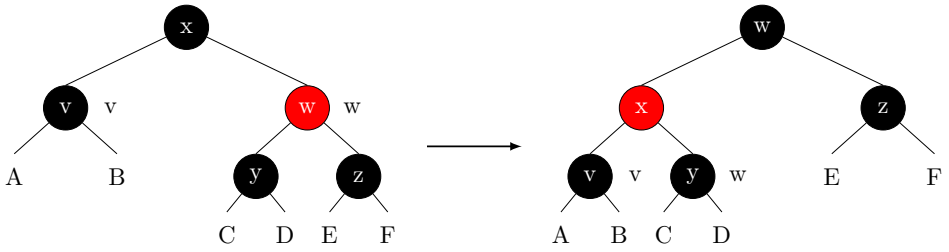
V ostatních případech se provede níže popsaná kontrola začínající na původní pozici vrcholu, jenž byl použit k nahrazení odstraňovaného vrcholu. Během kontroly vrcholu, zdali vrchol v, který má sourozencem w, splňuje pravidla červeno-černého stromu, se dle [6] postupuje tímto způsobem:

1. Pokud je w červený, vrchol w je obarven černě a rodič vrcholů v a w je obarven na červeně, následně je provedena jednoduchá rotace na rodiče vrcholů v a w. Orientace rotace je levá, pokud je v levým synem, nebo pravá, pokud je v pravým synem. Následně se vrcholem w stane nový sourozenec vrcholu v. Ukázka postupu v této situaci je znázorněna na 2.7.

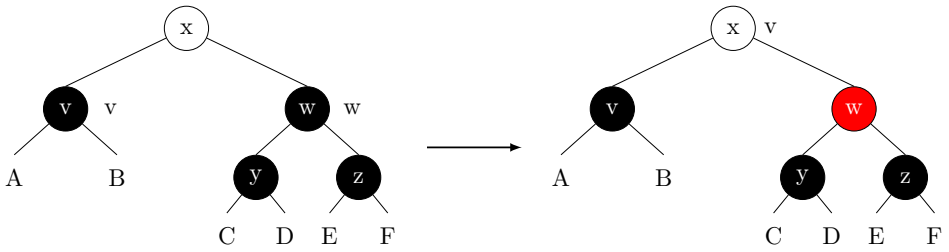
2. Jsou-li oba synové w obarveni černě, pak je vrchol w přebarven na červeno a podstrom určený vrcholem v je validní červeno-černý strom. Ovšem touto úpravou mohlo dojít k porušení některého pravidla cestou ke kořenu, proto je nutné pokračovat kontrolou rodiče vrcholu v. Znázornění řešení se nachází na obrázku 2.8.

3. Pokud je w obarven černě a pouze jeden ze synů vrcholu w je obarven černě, pak je obarven červený syn černě a vrchol w červeně. Následně, pokud byl levý syn vrcholu w původně červený a pravý syn černý, je provedena pravá rotace na vrchol w a symetricky, pokud byl pravý syn červený, je provedena rotace levá na w. Po rotaci musí být přeznačen vrchol w, jelikož došlo ke změně sourozence vrcholu v. Pro znázornění postupu viz 2.9.

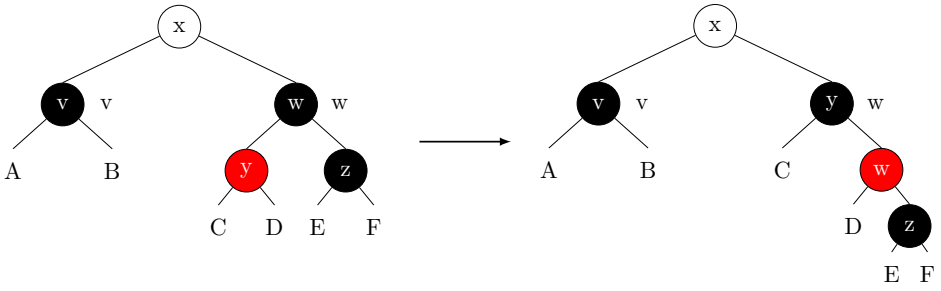
4. Nenastal-li 2. bod, je provedena rotace v rodiči vrcholů v a w směrem, který je určen pozicí vrcholu v, pokud je vrchol v levým synem, pak je provedena rotace levá, jinak je provedena rotace pravá. Vrchol w, nyní prarodič vrcholu v, je přebarven na barvu původního rodiče a jeho původní syn je přebarven černou barvou, původní rodič vrcholů v a w je stále otcem vrcholu v a je přebarven černou barvou. Tímto krokem byl strom opraven a je opět validním červeno-černým stromem. Znázornění řešení se nachází na obrázku 2.10.



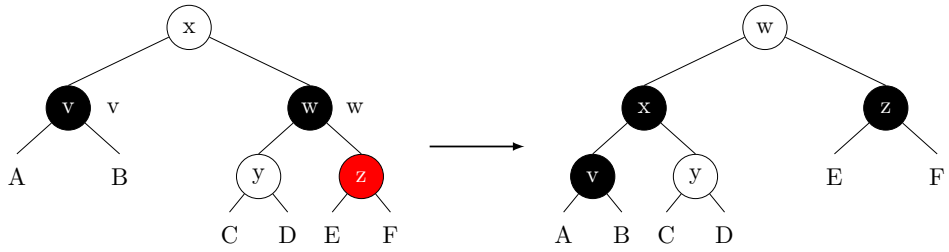
Obrázek 2.7: Znázornění řešení případu 1 z 2.3.2, když je vrchol v levý syn



Obrázek 2.8: Znázornění řešení případu 2 z 2.3.2, když v je levý syn



Obrázek 2.9: Znázornění řešení případu 3 z 2.3.2 když je vrchol v levý syn



Obrázek 2.10: Znázornění řešení případu 4 z 2.3.2, kde vrchol v je levý syn

2.3.3 Vlastnosti červeno-černého stromu

Operace vyhledávání zůstala nezměněna a se zaručenou hloubkou $O(\log n)$ je její složitost v nejhorším případu také $O(\log n)$. V případu vkládání trvá nalezení pozice pro vložení nového vrcholu $O(\log n)$, pokud nastane stav 2 nebo 3 je provedena rotace, která má konstantní složitost, a operace je úspěšně ukončena, v nejhorším případu se může opakovat situace 1, dokud není navštíven kořen stromu, tato propagace má složitost $O(\log n)$ a celá operace má tedy složitost také $O(\log n)$.

V poslední operaci, kterou je operace odstraňování, se nejdříve musí nalézt vrchol pro odstranění a případně vrchol, který odstraňovaný vrchol nahradí, tyto vrcholy lze nalézt v čase $O(\log n)$, operace v situaci 4 provede jednu rotaci a přebarvení několika vrcholů, což má konstantní složitost, tato situace nenastane pouze, pokud nastane 2. stav, který je jediný stav, ve kterém kontrola postupuje stromem, a i v tomto případu se postupuje po cestě ke kořenu stromu, tato propagace má proto složitost $O(\log n)$ a celá operace má ve výsledku složitost $O(\log n)$.

Po paměťové stránce musí být na rozdíl od nevyváženého BVS v každém vrcholu uložena informace o barvě vrcholu a paměťové nároky jsou tedy vyšší o počet vrcholů.

Výhodou oproti AVL stromům je, že během odstraňování a přidávání může dojít maximálně k jedné rotaci, u AVL stromu může dojít až k $O(\log n)$ rotacím, které, i když mají složitost $O(1)$, zpomalují program. Nevýhodou je však vyšší maximální hloubka a červeno-černý strom, který často vyhledává, může být pomalejší než AVL strom použitý ve stejné situaci.

2.4 AA strom

AA strom byl navrhnut Arne Anderssonem v [14] a podle něj byl i pojmenován. AA strom je upravením červeno-černého stromu, který je binární realizací (2,4)-stromu, tak, aby se jednalo o (2,3)-strom, přidáním pravidla které určuje, že červeně může být obarven pouze pravý syn vrcholu. Vzhledem k tomu, že se jedná o realizaci (2,3)-stromu, který má zaručenou hloubku $O(\log n)$ (viz 2.9), a v realizaci (2,3)-stromu pomocí AA stromu obsahuje AA strom pro každý

vrchol (2,3)-stromu se stejnými daty maximálně 2 vrcholy, má i AA strom hloubku $O(\log n)$.

Definice 13 (AA strom)**.** AA strom je binární vyhledávací strom, který splňuje následující podmínky:

1. Každý vrchol je obarven buď červeně nebo černě.
2. Kořen stromu je obarven černě.
3. Každý vnější vrchol je obarven černě.
4. Pokud je vnitřní vrchol obarven červeně, pak jsou jeho synové obarveni černě.
5. Pro každý vrchol v platí, že všechny cesty z vrcholu v do vnějších vrcholů stromu obsahují stejný počet černých vrcholů.
6. Žádný levý syn není obarven červeně.

2.4.1 Vyvažovací operace

Arne Andersson využívá v [14] úroveň vrcholů pro vyvažování, tato úroveň se liší od hloubky tím, že úroveň vrcholu představuje počet levých synů mezi vrcholem a nejbližším vnějším vrcholem, tato úroveň slouží k rozpoznání hloubky vrcholů tak, jako kdyby byl strom (2,3)-stromem.

AA strom musí, aby byl validním AA stromem a měl zaručenou hloubku $O(\log n)$, dodržovat následující invarianty pro úroveň vrcholu:

- Pokud je levý syn vrcholu vrchol vnější, pak je úroveň vrcholu 1.
- Pokud je vrchol červený, pak má stejnou úroveň jako jeho rodič a vhledem ke stejné úrovni vrcholů se hrana, která tyto vrcholy spojuje, označuje horizontální hranou.
- Pokud je vrchol černý, pak má úroveň o 1 nižší než rodič.
- Úroveň vnějšího vrcholu je 0.
- Pravý syn vrcholu má úroveň stejnou nebo nižší než rodič.
- Pravý syn pravého syna vrcholu má nižší úroveň než jeho prarodič.

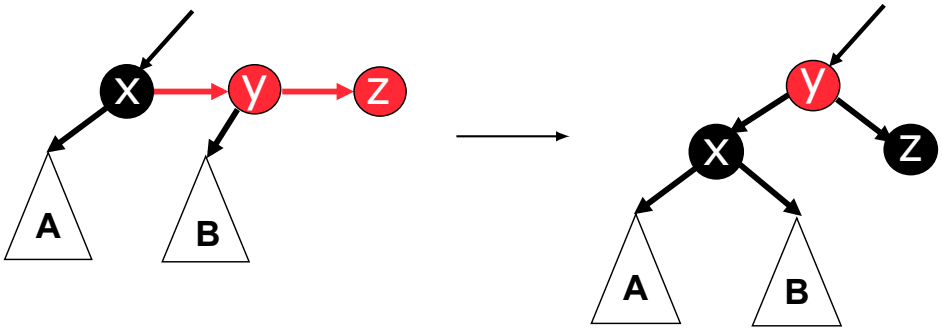
Z předcházejících invariant a pravidel stromu mimo jiné vyplývá, že uzly s úrovní 2 nebo vyšší musí mít jako syny dva vnitřní vrcholy, horizontální hrany mohou být pouze mezi rodičem a pravým synem, ve strom nemůže existovat vrchol, u kterého by byly 2 horizontální hrany, a pro každou cestu z kořenu do vnějšího vrcholu stromu je pro každou úroveň navštíven právě 1 černý vrchol.

Pro vyvažování stromu se využívají následující důsledky z pravidel a invariant AA stromu:

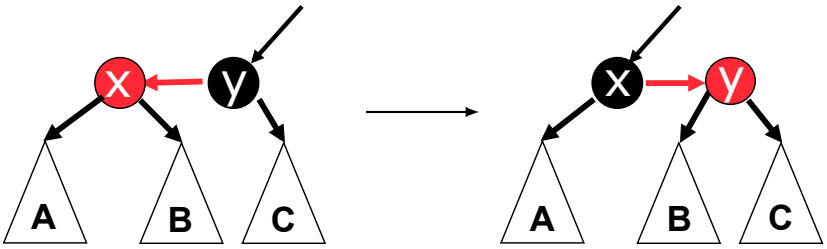
- Strom nemůže obsahovat levou horizontální hranu, tedy žádný vrchol nemůže mít červeně obarveného syna.
- Strom nemůže obsahovat dvě horizontální hrany v řadě, tedy červený vrchol nemůže být rodičem červeně obarveného vrcholu.

Pro vyvažování AA stromu se využívají následující operace:

- **split** – operace je využita, pokud jsou nalezeny 2 červené vrcholy za sebou. Podstromy vrcholů upravuje stejně jako levá rotace z 2.2.1 a po provedení rotace je změněna barva a úroveň vrcholu, znázornění operace je na obrázku 2.11.
- **skew** – operace je využita, pokud byl nalezen červený levý syn. Operace postupuje jako pravá rotace z 2.2.1, jedná se o otočení orientace horizontální hrany s upravením barev a úrovní. Operací `skew` mohou vzniknout dvě horizontální hrany v řadě, které je nutné opravit operací `split`. Operace je znázorněna na obrázku 2.12.



Obrázek 2.11: Operace split



Obrázek 2.12: Operace skew

2.4.2 Základní operace

Vyhledávání probíhá stejně jako u nevyváženého BVS, ke změnám dochází pouze ve vkládání a odstraňování vrcholů.

Vkládání do AA stromu

Po vložení nového prvku způsobem popsaným v 2.1.1, který vždy vloží nový klíč do úrovně 1, mohlo dojít k porušení některého z pravidel v rodičovi nového vrcholu, případné porušení je řešeno operacemi `skew` a `split`. Po provedení vyvažovacích operací mohlo dojít k porušení některého z pravidel u rodiče vrcholu a tímto způsobem se při případném porušení postupuje směrem ke kořenu stromu, dokud jsou porušeny některá pravidla a prováděny vyvažovací operace.

Odstraňování z AA stromu

Odstraňování probíhá jako u BVS, po odstranění vrcholu v je nutné provést kontrolu, zdali nedošlo k porušení některého z pravidel. Po odstranění vrcholu v se postupuje ke kořenu, dokud je prováděn některý z bodů v níže uvedeném postupu, a pro každý navštívený vrchol x se postupuje následujícím způsobem:

1. Pokud má některý ze synů vrcholu x nižší úroveň než x o 2, pak je snížena úroveň x o 1, pokud měl pravý syn stejnou úroveň jako vrchol x před jejím snížením, pak je snížena i úroveň jeho pravého syna.

2. Pokud má levý syn vrcholu x stejnou hloubku jako x, pak je provedena operace `skew`.

3. Pokud má pravý syn pravého syna vrcholu x stejnou hloubku jako vrchol x, pak je provedena operace `split`.

2.4.3 Vlastnosti AA stromu

Operace vyhledávání zůstává nezměněna a vzhledem k hloubce AA strom má složitost $O(\log n)$. Operace `skew` a `split` mají vzhledem k tomu, že se jedná pouze o prohození rodiče se synem, příslušné přesunutí podstromů a přebarvení, složitost $O(1)$. Operace odstraňování a vkládání postupují po odstranění nebo vložení prvku maximálně ke kořenu stromu a mohou provést vyvažovací operace, které mají složitost konstantní, proto mají i tyto operace složitost $O(\log n)$.

Oproti červeno-černému stromu došlo ke značnému usnadnění situací, které mohou po úpravách nastat, proto došlo i ke zjednodušení implementace. Podle [15] jsou rychlosti AA stromu a červeno-černého stromu podobné, AA strom mívá nižší hloubku, tudíž má rychlejší operaci vyhledávání, také má jednodušší a rychlejší operaci odstraňování. Ovšem operaci vkládání má červeno-černý

strom rychlejší. I když bývají pro stejná data rychlosti obou stromů v průměru podobné, tak [15] uvádí, že AA strom má větší rozptyl rychlostí a červeno-černý strom je více stálý.

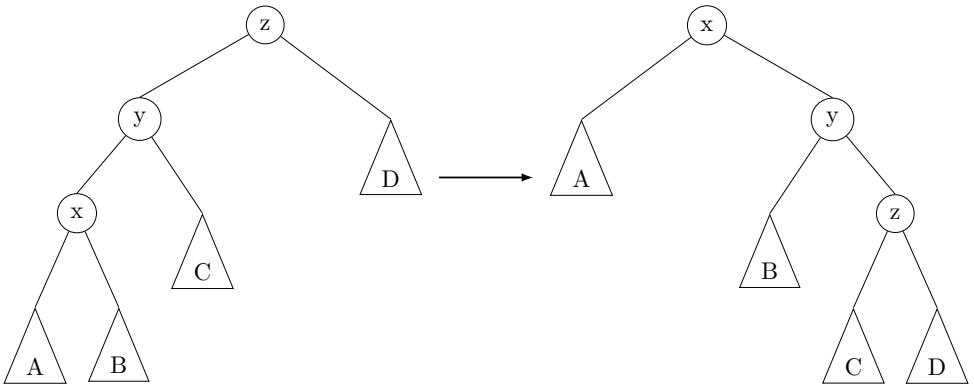
2.5 Splay strom

Splay strom je vyvažovací strom, který byl poprvé popsán v [16]. Od předchozích stromů, které jsou vyvažované tak, aby byla zajištěna příznivá hloubka stromu pro rychlejší průběh operací, se liší tím, že je vyvažován tak, aby prvky, ke kterým se nedávno přistupovalo, byly blízko ke kořeni a byly rychle dostupné.

2.5.1 Vyvažovací operace

Splay strom využívá vyvažovací operaci `splay`, která přesune vrchol stromu na pozici kořenu stromu. Operace `splay` využívá pro přesunutí vrcholu x na pozici kořenu operace:

- **zig-zig** – operace je provedena, pokud rodič vrcholu x není kořen a buď je vrchol x levým synem a i jeho rodič je levým synem, nebo je vrchol x pravým synem a jeho rodič je také pravým synem. Pokud jsou x i jeho rodič syny levými, pak je výsledek operace stejný, jako kdyby byla na prarodiče vrcholu x provedena pravá rotace a následně byla provedena pravá rotace i na rodiče vrcholu, to samé platí symetricky pro syny na pravé straně s levou rotací. Ukázku operace lze nalézt na obrázku 2.13.



Obrázek 2.13: Ukázka zig-zig rotace

- **zig-zag** – operace je provedena, pokud rodič vrcholu x není kořen a buď je rodič vrcholu x pravým synem a x levým synem, nebo je rodič vrcholu x levým synem a x pravým synem. Pokud některý z těchto případů nastane, pak je provedena dvojitá rotace, totožná s rotacemi popsanými v 2.2.1. Výsledek operace pro první popsanou situaci je stejný, jako kdyby

byla provedena RL rotace a pro druhou situaci je výsledek stejný, jako kdyby provedena LR rotace.

- **zig** – operace je provedena, pokud je rodičem vrcholu x kořen stromu. Operace odpovídá jednoduché rotaci, které jsou popsány v 2.2.1, orientované tak, aby po rotaci byl vrchol x novým kořenem stromu.

Operace `splay` pro vrchol x postupuje následujícím způsobem:

1. Pokud je vrchol x kořen, operace je úspěšně ukončena.
2. Pokud je rodič vrcholu x kořen, pak je provedena operace `zig` tak, aby se stal vrchol x kořenem. Operace je úspěšně ukončena.
3. Pokud je rodič vrcholu x levým synem a vrchol x také, nebo pokud je vrchol x pravým synem a jeho rodič také, pak je pro vrchol x provedena operace `zig-zag`. Následně se postupuje znovu od bodu 1.
4. Jinak je provedena operace `zig-zig` a postupuje se od bodu 1.

2.5.2 Základní operace

Splay strom provádí základní operace stejným způsobem jako nevyvážený BVS s tím rozdílem, že po každé operaci je strom upraven tak, aby byl poslední navštívený vrchol kořenem stromu.

Vyhledávání ve splay stromu

Vyhledávání prochází stromem jako v nevyváženém BVS. Následně, byl-li vrchol nalezen, je operací `splay` přesunut na pozici vrcholu stromu, a pokud vrchol nalezen nebyl, pak je na pozici kořenu stromu přesunut vnitřní vrchol, který byl při vyhledávání navštíven jako poslední.

Vkládání do splay stromu

Operace vložení začíná stejným způsobem jako operace vkládání do nevyváženého BVS. Pokud operace proběhla neúspěšně a strom již obsahuje vrchol se shodným klíčem, pak je tento vrchol operací `splay` přesunut na pozici kořenu stromu. Pokud strom neobsahoval vrchol se shodným klíčem a nový vrchol byl vložen, pak je nově vložený vrchol přesunut na pozici kořenu.

Odstraňování ze splay stromu

Autoři splay stromu v [16] uvádějí dvě alternativy pro odstraňování ze splay stromu.

První možností je nejdříve zavolat výše definovanou operaci vyhledávání pro klíč, který má být ze stromu odstraněn. Pokud po provedení vyhledávací

operace není kořenem stromu vrchol, který má shodný klíč s klíčem, který má být ze stromu odebrán, pak je operace neúspěšně ukončena. Pokud má kořen shodný klíč, pak je kořen odstraněn a je nahrazen buď maximem v levém podstromu, nebo minimem v pravém podstromu.

Druhou alternativou, kterou uvádějí autoři v [16] jako efektivnější, je procházet stromem jako při hledání vrcholu a pokud strom neobsahuje vrchol s hledaným klíčem, pak je operací `splay` na pozici kořenu stromu přesunut poslední navštívený vnitřní vrchol. Pokud strom obsahuje vrchol v s klíčem, který je shodný se zadaným klíčem pro odstranění, pak je tento vrchol odstraněn. Pokud byl alespoň jeden ze synů vrcholu vrchol vnější, pak se při nahrazování vrcholu postupuje triviálně jako v 2.1.1. V opačné situaci je vrchol nahrazen maximem levého podstromu, nebo minimem pravého podstromu. Pokud byl v kořen, pak je operace úspěšně ukončena, jinak musí být ještě provedena operace `splay` na rodiče odstraněného vrcholu.

2.5.3 Vlastnosti splay stromu

Operace vyhledávání a vkládání mají stejný základ jako operace nevyváženého BVS, ovšem na závěr operace je vždy přesunut vrchol operací `splay`. Operace `splay` přesune vrchol na pozici kořenu stromu, pokud je vrchol určený operacemi vyhledávání a vkládání nejvzdálenější vnitřní vrchol od kořenu, musí se pro přesun provést $\lceil h/2 \rceil$ operací a složitost operací je $O(h)$.

Operace odstraňování, pokud strom neobsahuje prvek se zadaným klíčem, přesune vnitřní vrchol, jehož synové jsou vnější vrcholy, což může být nejvzdálenější vnitřní vrchol od kořenu, a proto je asymptotická složitost operace odstraňování také $O(h)$.

Splay strom s n vrcholy nemá zaručenou lepší hloubku stromu než $O(n)$, která může vzniknou například tak, že je vždy vloženo nové maximum stromu, a tudíž mají základní operace splay stromu asymptotickou složitost $O(n)$.

I když je asymptotická složitost základních operací $O(n)$, tak alespoň amortizovaná složitost základních operací je $O^*(\log n)$, důkazy jsou provedeny v [16, 2].

Splay strom nepotřebuje ukládat žádné speciální informace a lze s ním dosáhnout efektivního uložení klíčů do paměti. Další výhodou je rychlý přístup k prvkům, se kterými strom pracoval v poslední době. Z tohoto důvodu bývá splay strom využíván v cache pamětích a v počítačových sítích pro velké tabulky IP adres.

2.6 Váhově vyvážený vyhledávací strom

Váhově vyvážené vyhledávací stromy byly poprvé popsány v [17], kde je autoři Nievegelt a Reingold nazvali binárními stromy s omezenou rovnováhou (anglicky bounded balance) a vzhledem k jejich původnímu názvu bývají váhově

vyvážené vyhledávací stromy označovány jako BB-α nebo BB[α] stromy, kde α představuje parametr, který určuje vyváženost stromu.

Od hloubkově vyvážených stromů, u kterých se sleduje rozdíl hloubek podstromů vrcholu, se liší tím, že BB-α stromy sledují počet vrcholů podstromů a kontrolují jejich poměr.

Definice 14 (Rovnováha kořene stromu)**.** Rovnováha kořene $\rho(T)$ pro strom T kořenem v, kde n značí počet vnitřních vrcholů stromu T, pro který platí $n \geq 1$ je $\rho(v) = \frac{|T(l(v))|+1}{|T(v)|+1}$.

V některých zdrojích, například v [18], se pro výpočet rovnováhy kořenu stromu používá poměr počtu vnějších vrcholy v levém podstromu s celkovým počtem vnějších vrcholů. Tyto definice jsou ekvivalentní.

Definice 15 (Váhově vyvážený vyhledávací strom)**.** BVS T s n vnitřními vrcholy je BB-α, kde $\alpha \in$ R takové, že $0 < \alpha \leq \frac{1}{2}$, právě tehdy když $n \leq 1$, nebo pokud pro každý jeho vnitřní vrchol v platí:

- $\alpha \leq \rho(v) \leq 1 - \alpha$.
- Levý i pravý podstrom jsou validní BB-α stromy.

Parametr α určuje maximální poměr vrcholů v podstromech každého vrcholu. Pro $\alpha = \frac{1}{2}$ se jedná o dokonale vyvážený BVS a pro nižší α je vyvažování stromu volnější.

2.6.1 Vyvažovací operace

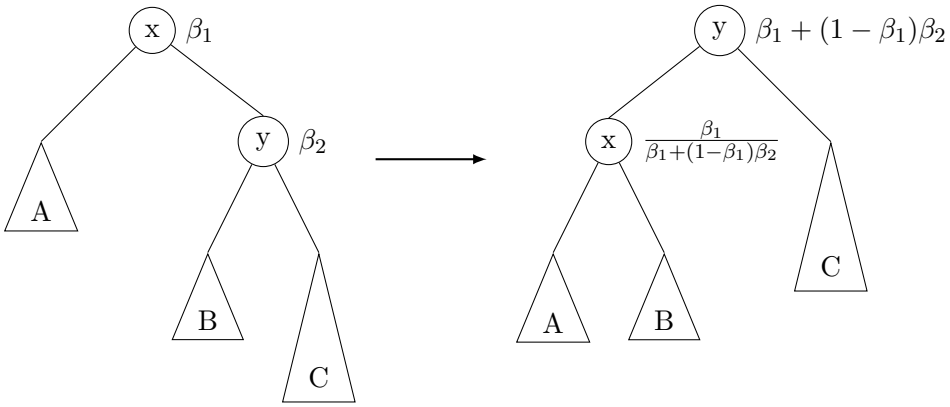
Autoři BB-α stromu v [17] uvádějí, že pro správnou funkčnost jimi uvedených algoritmů je nutné, aby pro parametr α platilo $\alpha \leq 1 - \frac{1}{\sqrt{2}}$. Později byl tento požadavek v [18] upřesněn na $\frac{2}{11} < \alpha \leq 1 - \frac{1}{\sqrt{2}}$.

Pro vyvažování BB-α stromů, u kterých došlo k porušení pravidel, jsou používány levé, pravé, LR a RL rotace totožné s rotacemi v uvedenými v 2.2.1. Operace levé rotace a RL rotace, včetně vlivu operací na rovnováhy kořenů podstromů, jsou znázorněny na obrazcích 2.14 a 2.15.

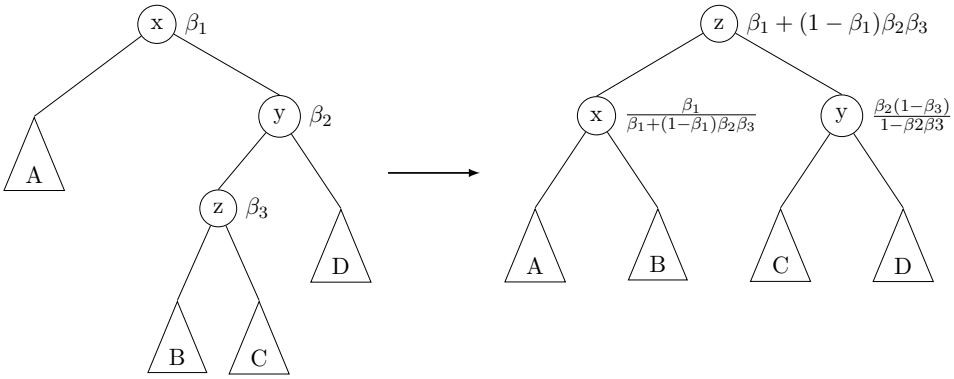
2.6.2 Základní operace

Vyhledávání ve stromu nemůže změnit počet vrcholů v žádném podstromu a strom je tak stále validní BB-α stromem. Operace může tedy probíhat stejně jako v nevyváženém BVS.

Při neúspěšném vkládání nebo odstraňování se také nemohl změnit počet vrcholů v žádném podstromu a strom zůstává BB-α stromem. Po úspěšném provedení operací vkládání nebo odstraňování byl změněn počet vrcholů



Obrázek 2.14: Levá rotace BB-α stromu



Obrázek 2.15: RL rotace BB-α stromu

a musí být provedena kontrola. Kontrola validity BB-α stromu začíná ve vrcholu, který je rodičem nově přidaného vrcholu, nebo vrcholu, který byl ze stromu odebrán.

Při kontrole vrcholu v se postupuje následujícím způsobem:

1. Pokud platí $\alpha \leq \rho(v) \leq 1-\alpha$, pak je podstrom validním BB-α stromem, nejsou provedeny žádné úpravy a pokud není vrchol v kořenem stromu, pak se pokračuje kontrolou rodiče vrcholu v.

2. Pokud je $1 - \alpha < \rho(v)$, pak obsahuje levý podstrom vrcholu v více vrcholů oproti pravému podstromu než je povoleno parametrem α a musí být provedena rotace. Pokud platí $\alpha \leq 1 - \rho(r(v)) < \frac{1-2\alpha}{1-\alpha}$, pak je provedena pravá rotace a podstrom splňuje podmínky BB-α stromu. Jinak je provedena LR rotace. Pokud není vrchol v kořenem stromu, pak se pokračuje kontrolu rodiče vrcholu v.

3. Pokud je $\rho(v) < \alpha$, pak obsahuje levý podstrom vrcholu v méně vrcholů oproti pravému podstromu než je povoleno parametrem α a musí být

provedena rotace. Pokud platí $\alpha \leq \rho(r(v)) < \frac{1-2\alpha}{1-\alpha}$, pak je provedena levá rotace a podstrom splňuje podmínky BB-α stromu. Jinak je provedena RL rotace. Pokud není vrchol v kořenem stromu, pak se pokračuje kontrolu rodiče vrcholu v.

2.6.3 Vlastnosti váhově vyváženého vyhledávacího stromu

Pro analýzu složitostí BB-α stromů bude parametr α, vzhledem k předchozí definici operací, omezen na interval $(\frac{2}{11}, 1 - \frac{1}{\sqrt{2}}\rangle$.

Hloubka BB-α stromu je maximálně $h = \frac{\log_2 n}{\log_2 \frac{1}{1-\alpha}} = \log_{\frac{1}{1-\alpha}} n$, což znamená že hloubka BB-α stromu je $O(\log n)$. Důkaz se provádí indukcí a je uveden v [11].

Vzhledem k zachování operace vyhledávání, která má asymptotickou složitost $O(h)$, je asymptotická časová složitost vyhledávání $O(\log n)$. Operace vkládání i odebírání začínají stejným způsobem jako stejné operace v nevyváženém BVS a mají základní složitost $O(h) = O(\log n)$, následná kontrola postupuje směrem ke kořenu, provádí rotace, které mají složitost $O(1)$, a pokud je v každém vrcholu uložen počet vrcholů v jeho podstromu, pak má také asymptotickou časovou složitost $O(\log n)$. Tudíž složitost operací vkládání o odstraňování pro BB-α strom je $O(\log n)$.

Význam váhově vyvážených vyhledávacích stromů je spíše historický. Dříve byly využívány, protože AVL stromy při vkládání a odstraňování nejdříve postupují od kořene k listům a pokud nastane změna hloubky, pak operace postupují od listů ke kořenu a proto je při implementaci AVL stromů jednosměrným spojovým listem nutné pro dosažení asymptotické složitosti $O(\log n)$ využití zásobníku, nebo jiné struktury k uložení navštívených vrcholů, zatímco u BB-α stromů lze postupovat od kořene k listům pro odstranění či přidání vrcholu a znovu postupovat od kořene k listům pro kontrolu stromu, nebo rotace provádět již při prvním postupu, důkaz je uveden v [17]. Tento fakt umožňuje ušetření paměti oproti některým typům.

2.7 Scapegoat strom

Scapegoat strom byl poprvé popsán v [19] a následně s drobnými úpravami v [20], kde se mu dostalo názvu scapegoat (česky obětní beránek) strom. Jedná se o úpravu BB-α stromu změnou způsobu udržování hloubky stromu.

Scapegoat strom využívá pro vyvažování maximální možnou hloubku BB-α stromu T s n vrcholy, která je $h_\alpha(n) = \lfloor \log_{\frac{1}{1-\alpha}} n \rfloor$ a pro každý vrchol v, který obsahuje BB-α strom, platí $h(T(u)) \leq h_\alpha(|T(u)|)$. Ovšem tato podmínka je pro scapegoat stromy upravena na $h(T) \leq h_\alpha(|T|) + 1$ a kontroluje se pouze u kořenu stromu. To znamená, že pokud má scapegoat strom s m vrcholy větší hloubku než $h_\alpha(m) + 1$, pak musí být tento strom upraven.

2.7.1 Vyvažovací operace

Scapegoat strom pro vyvažování využívá operaci, která sestaví nový strom z prvků v podstromu T, v němž se nachází vrchol ve větší hloubce než $h_\alpha(n)+1$ a ve kterém je porušeno váhové vyvážení. Kořen tohoto podstromu se nazývá scapegoat (česky obětní beránek).

Nejlehčím způsobem, jak vytvořit dokonale vyvážený BVS z prvků stromu T, je vložit vrcholy stromu do pole `inorder` průchodem, následně vkládat prvky do nového stromu půlením pole. To znamená, že je do stromu vložen prostřední prvek v poli, pole je rozpůleno na levou část a pravou část, které jsou odděleny vloženým prvkem, který ani jedna část neobsahuje. Tento postup se opakuje pro každé rozpůlené pole, dokud neobsahuje nové pole pouze jeden prvek, který je do stromu vložen a žádné další půlení není možné. Paměťová i časová asymptotická složitost tohoto způsobu jsou $O(|T|)$.

Další způsob také využívá `inorder` průchod, vzhledem k známému počtu prvků v podstromu a průchodu stromem od nejnižšího klíče k nejvyššímu je pro každý prvek známá pozice ve výsledném dokonale vyváženém stromu. Při navštívení vrcholu `inorder` průchodem je tento vrchol přesunut na pozici, která mu náleží. Tento způsob má časovou asymptotickou složitost $O(|T|)$ a paměťovou časovou složitost $O(\log n)$ pro jednosměrný spojový seznam, nebo $O(1)$ pro ostatní uvedené způsoby v 1.4.

Poslední uvedený způsob nejdříve přeskupí strom T do seřazeného seznamu od nejmenšího prvku po největší, to znamená, že kořenem je nejmenší prvek, jeho synem druhý nejmenší prvek a tak dále. Pokud lineární seznam obsahuje pouze dva prvky, pak se jedná o hloubkově vyvážený strom. Jinak pro $i = 1, \ldots, \lfloor log_2 n \rfloor - 1$ provede levou rotaci na kořen stromu, označí nový kořen stromu, a na každý druhý vrchol, který navštíví cestou z nového kořenu do největšího prvku ve stromu a který nemá syny pouze vnější vrcholy, provede levou rotaci. Tento způsob má časovou asymptotickou složitost $O(n)$ a paměťovou $O(1)$.

2.7.2 Základní operace

Postup základních operací probíhá stejně jako v nevyváženém BVS, po provedení operací, které upravují strukturu stromu je však nutno provést kontrolu, zdali pro hloubku stromu T neplatí $h(T) > h_\alpha(T)+1$, jelikož vkládání neupravuje hloubku stromu ani počet vrcholů, tak je tato operace totožná s operací vyhledávání v nevyváženém BVS.

Vkládání do scapegoat stromu

Další proměnnou scapegoat stromu je velikost stromu, kdy byla naposledy provedena operace vkládání nebo vyvážení stromu, která bývá využita při odstraňování ze stromu. Pro následující sekce bude tato proměnná označená jako `msize`.

Provede se vložení stejným způsobem jako v nevyváženém BVS, pokud byla operace neúspěšná, není potřeba dodatečná kontrola. Pokud byl vložen nový prvek, pak je přepočítána proměnná `msize` jako maximum z `msize` a $n+1$ a následně je provedena kontrola, zdali nebyl nový vrchol vložen do hloubky větší, než je povoleno vzhledem k parametru α a počtu prvků ve stromu. Pokud se tak stalo, pak je nalezen vrchol, v kterém byla porušen váhová vyváženost a podstrom určený tímto vrcholem je znovu sestaven, tak aby byl dokonale vyvážený.

Jeden způsob, jak nalézt vrchol, ve kterém došlo k porušení váhového vyvážení, je postupovat od nově vloženého vrcholu směrem ke kořenu a v každém navštíveném vrcholu v počítat $\rho(v)$. Při nalezení vrcholu, pro který neplatí $\alpha \leq \rho(v) \leq 1 - \alpha$, je podstrom určený tímto vrcholem přestaven.

Druhý způsob je postupovat od kořenu k nově vloženému vrcholu a při potkání podstromu, který porušuje váhové vyvážení, tento strom přestavět. Při tomto způsobu nemusí být v paměti rodič navštívených vrcholů a autoři stromu tvrdí v [20], že při využití tohoto způsobu jsou v průměru stromy lépe vyvážené.

Odstraňování ze scapegoat stromu

Vrchol je ze stromu odstraněn stejným způsobem jako v nevyváženém BVS. Pokud po odstranění vrcholu platí $|T| < (1 - \alpha)msize$, pak je celý strom přestavěn a proměnná `msize` je nastavena na $|T|$.

2.7.3 Vlastnosti scapegoat stromu

Hloubka scapegoat stromu s parametrem α je maximálně o 1 větší než u BB-α stromu pro stejný parametr α a proto je také $O(\log n)$.

Operace vyhledávání funguje stejným způsobem jako ve většině ostatních BVS a její složitost je tedy $O(h)$, což je vzhledem k hloubce stromu $O(\log n)$. Operace vkládání a odstraňování probíhají také stejným způsobem jako u nevyváženého BVS, ale po provedení může být porušena podmínka stromu a nějaký podstrom musí být přestaven, v nejhorším případě, nebo v případě operace odstraňování, to je celý strom a složitost těchto operací je proto $O(n)$, kde n je počet prvků ve stromu. Amortizovaná časová složitost těchto operací je však $O^*(\log n)$, jelikož k přestavění dochází výjimečně a důkaz je uveden v [20].

Další výhodou scapegoat stromů vedle dobré amortizované časové složitost je paměťová složitost, při zvolení vhodného stavění stromu a při hledání vrcholu, který není váhově vyvážený, postupem od kořenu k vloženému vrcholu, je možné implementovat scapegoat strom jednosměrným spojovým seznamem bez ovlivnění rychlosti programu. Scapegoat strom také nepotřebuje udržovat žádné dodatečné informace o vrcholech.

2.8 Treap

Struktura treap je spojení BVS se strukturou minimové haldy (anglicky tree a heap), název vznikl spojením anglických názvů těchto struktur. Treap byl poprvé popsán v [21] a využívá průměrnou hloubku nevyvážených BVS pro náhodná data $O(\log n)$[9]. Náhodnost je zajištěna přidělením priority, která je pro každý prvek náhodně vygenerována, každému vrcholu. Treap musí splňovat podmínky vyhledávacích stromů pro klíče, tedy seřazenost vrcholů ve stromu, a haldové uspořádání pro priority vrcholů, to znamená, že priorita vrcholu je vždy menší nebo stejná jako priorita vnitřních vrcholů, kterým je daný vrchol rodičem.

2.8.1 Vyvažovací operace

Treap využívá stejné jednoduché rotace jako jsou uvedené u AVL stromů v 2.2.1.

2.8.2 Základní operace

Vzhledem k zachování uspořádání klíčů jako ve vyhledávacím stromu, je ve struktuře vyhledáno stejným způsobem jako v BVS. Vzhledem k přidání priority vrcholů a nutnosti zachování haldového uspořádání pro priority vrcholů musí být operace vkládání a odstraňování upraveny.

Vkládání do struktury treap

Vložení nového klíče proběhne stejným způsobem jako v nevyváženém BVS, pokud proběhlo vložení úspěšně, je vrcholu přidělena priorita, která byla náhodně vygenerována. Jelikož byl vrchol vložen na pozici vnějšího vrcholu, mohlo dojít k porušení haldového uspořádání pouze směrem ke kořenu. Napravení případného porušení haldového uspořádání probíhá tak, že dokud není nově vložený vrchol kořen, nebo dokud není priorita jeho rodiče nižší než jeho priorita, je na rodičovi nově vloženého vrcholu provedena jednoduchá rotace levá, pokud je vrchol pravým synem, nebo pravá, pokud je vrchol levým synem.

Odstraňování do struktury treap

Při odstraňování je nejdříve vyhledán vrchol s hledaným klíčem, pak, pokud strom vrchol s hledaným klíčem obsahuje, je nalezený vrchol přesunut na pozici, kde budou alespoň jeden jeho syn vrchol vnější, toho lze docílit opakováním jednoduché rotace, jejíž směr je určen prioritou synů, dokud jsou synové vrcholu vnitřní vrcholy.

2.8.3 Vlastnosti struktury treap

Vzhledem k přidání náhodných priorit je hloubka struktury treap v průměru $O(\log n)$, důkaz uveden v [21]. Avšak může nastat případ, kdy bude hloubka struktury $O(n)$, pravděpodobnost, že hloubka struktury bude $O(n)$ je však velmi nízká.

Rotace, které struktura využívá k udržení haldového uspořádání, mohou být stejné jako v 2.2.1, jelikož nikdy nemohou porušit haldové uspořádání žádného vrcholu, protože rodič vrcholu je buď rotací nezměněn, rodičem vrcholu je vrchol, který je ve stromu přesouván na správnou pozici, a tato situace může nastat pouze, pokud má vrchol vyšší prioritu než přesouvaný vrchol, a poslední možnost, která při rotaci může nastat, je, že novým rodičem vrcholu je vrchol, který byl rodičem vrcholu před přesouváním vrcholu ve stromu. Tudíž mají rotace stejnou složitost jako v 2.2.1, která je konstantní.

Operace vyhledávání je nezměněna oproti BVS a její asymptotická složitost je $O(h)$, tedy $O(n)$ a v průměru $O(\log n)$. Operace vkládání začíná stejným způsobem jako v nevyváženém BVS, ale po vložení může dojít k přesouvání vrcholu ke kořenu, v nejhorším možném případu na pozici kořenu, rotacemi, které mají konstantní složitost. Složitost operace vkládání je proto $O(n)$ a v průměru $O(\log n)$.

Operace odstraňování nejdříve průchodem stromu nalezne vrchol určený k odstranění. Před odstraněním je vrchol přesunut na pozici, na níž budou oba jeho synové vnější vrcholy. V nejhorším případu se tedy jedná o přesunutí vrcholu z pozice kořenu na pozici nejvzdálenějšího vnitřního vrcholu, což je $O(h)$ a celá operace má asymptotickou složitost $O(h)$, tedy $O(n)$ a v průměru $O(\log n)$.

Výhodou struktury treap je velmi velká jednoduchost struktury a operací, které struktura provádí. Operace odstraňování je dokonce jednodušší než v nevyváženém BVS. Další výhodou struktury je průměrná hloubka $O(\log n)$ a pro značný počet prvků ve struktuře, je pravděpodobnost hloubky $O(\log n)$ velmi vysoká.

Nabízí se také myšlenka volby priority vrcholů vzhledem k rozdělení pravděpodobnosti práce s vrcholy, tuto myšlenku realizují takzvané prioritní vyhledávací stromy, které při vkládání nedávají vrcholům náhodné priority, ale dávají jim určené priority, vzhledem k pravděpodobnosti práce s těmito vrcholy. Operace pro vyhledávání a odstraňování fungují stejným způsobem jako ve struktuře treap.

2.9 (a,b)-strom

Všechny předchozí vyhledávací stromy jsou binární a každý jejich vnitřní vrchol má jeden klíč. (a,b)-strom umožňuje proměnlivý počet klíčů ve vnitřním vrcholu v závislosti na parametrech a a b. (a,b)-stromy vznikly upravením B stromů, které byly poprvé popsány v [22] a jsou podmnožinou (a,b)-stromů.

Definice 16 ((a,b)-strom)**.** (a,b)-strom, kde $a \geq 2$ a $b \geq 2a-1$, je vyhledávací strom, který neobsahuje žádný vrchol, nebo splňuje následující podmínky:

1. Kořen stromu má 2 až b synů.
2. Vnitřní vrcholy, které nejsou kořenem stromu, mají a až b synů.
3. Všechny vnější vrcholy jsou ve stejné hloubce.

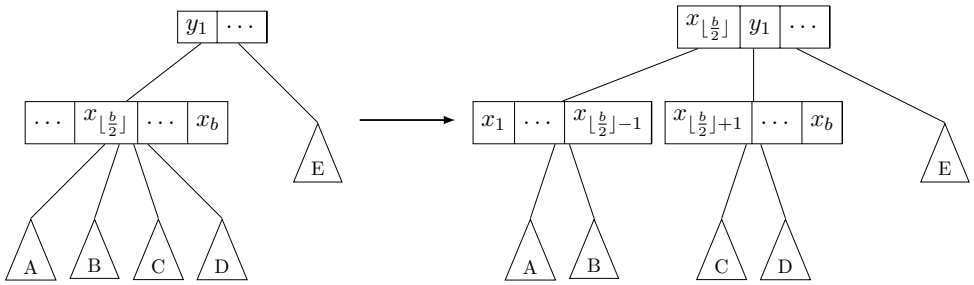
2.9.1 Vyvažovací operace

Vyvažovací operace (a,b)-stromu jsou využity, pokud došlo k porušení některé z pravidel stromu. Vzhledem k definici základních operací nemůžeme být porušeno pravidlo číslo 3 a vzhledem k tomu, že $a \geq 2$, tak pravidlo číslo 1 je umírnění pravidla 2 a slouží k ulehčení situací při vkládání, odebírání nebo k možnosti reprezentace stromu s počtem vrcholů menším než $a - 1$.

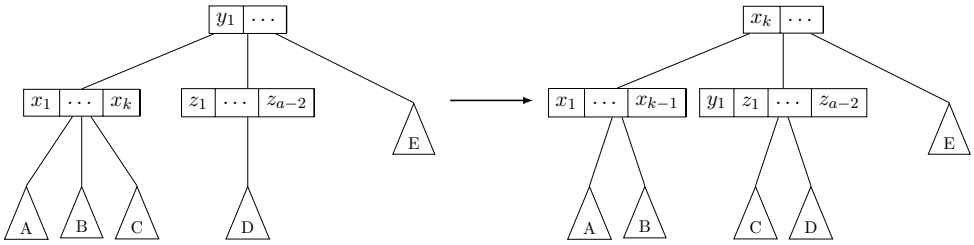
Při případu porušení některého z pravidel (a,b)-stromu, jsou dle [2] k napravení stromu využity následující vyvažovací operace:

- **Štěpení** – je použito, pokud je ve vrcholu více klíčů, než je povoleno. Prostřední klíč přeplněného vrcholu je přesunut do rodiče. Levý syn přesunutého klíče je nově vytvořený vrchol, který obsahuje klíč, případně klíče, menší než klíč přesunutého vrcholu. K vrcholu jsou také přesunuty podstromy příslušných klíčů. Pravý syn je původní přeplněný vrchol, ze kterého byly klíče odebrány. Štěpením mohlo dojít k přeplnění vrcholu, do kterého byl přesunut prostřední klíč, a pokud se tak stalo, je štěpení provedeno v nově přeplněném vrcholu. Takto se postupuje směrem ke kořenu, dokud štěpením vznikají přeplněné vrcholy. Pokud dojde ke štěpení kořenu stromu, pak vznikne nový kořen, který obsahuje pouze prostřední klíč z původního kořenu a strom bude díky pravidlu číslo 1 validním (a,b)-stromem. Znázornění operace lze nalézt na obrázku 2.16.
- **Půjčení** – je použito, pokud je ve vrcholu méně klíčů, než je povoleno a vrchol má sourozence, ze kterého může být přesunut klíč, aniž by došlo k porušení pravidla (a,b)-stromu. Při půjčování klíče z levého sourozence je přesunut jeho největší klíč na pozici klíče v rodičovi, jehož synové jsou vrchol, který si půjčuje klíč, a vrchol, z kterého je půjčováno. Nahrazený klíč je přesunut na pozici nejmenšího klíče vrcholu, který má nedostatek klíčů, a jeho levým synem se stane původní pravý podstrom klíče přesunutého do rodiče. Půjčování z pravého sourozence probíhá symetricky. Po půjčení nemohlo dojít k porušení pravidla výše ve stromu a strom je validním (a,b)-stromem. Operace připomíná jednoduchou rotaci v BVS. Pro znázornění operace viz 2.17.
- **Sloučení** – je použito, pokud je ve vrcholu méně klíčů než je povoleno a půjčení není možné. Klíče ze sourozence, klíče ze slučovaného vrcholu

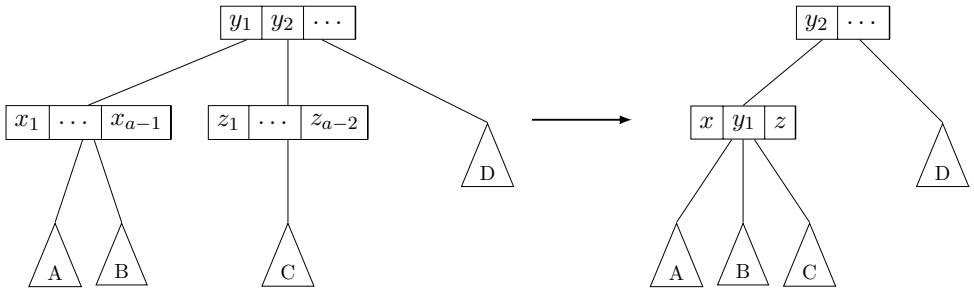
a klíč z rodiče, jehož synové jsou slučováni, jsou sloučeny do jednoho vrcholu s příslušnými podstromy. Vzhledem k tomu, že neměl sourozenec dostatek klíčů pro půjčení, tedy měl $a-1$ klíčů, původní vrchol má nedostatek klíčů, tedy měl $a-2$ klíčů. Tak po spojení těchto vrcholů a jejich rodiče má nový vrchol $2a-2$ klíčů a splňuje pravidla stromu. K porušení pravidel však mohlo dojít výše ve stromu přesunutím rodiče, tímto způsobem se může postupně slučovat až ke kořenu, který po sloučení nemusí obsahovat žádný klíč a novým kořenem se stane jeho jediný syn. Znázornění operace lze nalézt na obrázku 2.18.



Obrázek 2.16: Štěpení vrcholu (a,b)-stromu



Obrázek 2.17: Půjčení od levého souseda vrcholu (a,b)-stromu



Obrázek 2.18: Sloučení vrcholů (a,b)-stromu, x představuje klíče $x_1 \dots x_{a-1}$ a z klíče $z_1 \dots z_{a-2}$

2.9.2 Základní operace

Vzhledem ke změně počtu klíčů ve vrcholech se všechny základní operace (a,b)-stromu liší od operací BVS stromu, ovšem operace vyhledávání je velmi podobná.

Vyhledávání v (a,b)-stromu

Operace vyhledává vrchol s klíčem k a začíná v kořenu stromu. V každém vnitřním vrcholu v, který je během vyhledávání navštíven a který obsahuje klíče $x_1, \ldots, x_k$ a syny $s_0, \ldots, s_k$, je pro $i = 1, \ldots, k$: pokud je $k < x_i$, vyhledávání pokračuje v s_{i-1}, pokud je $k = x_i$, vrchol obsahující klíč k je vrchol v. Jestli nejsou předchozí porovnání splněna pro žádné i, pak vyhledávání pokračuje v s_k. Pokud je při vyhledávání navštíven vrchol vnější, pak strom neobsahuje vrchol s klíčem k a vyhledávání proběhlo neúspěšně.

Vkládání do (a,b)-stromu

Při vkládání se nejdříve musí stejným způsobem jako při vyhledávání najít pozice, na kterou má být nový klíč vložen, pokud byl nalezen vnitřní vrchol, pak strom již klíč obsahuje a operace proběhla neúspěšně, pokud byl nalezen vnější vrchol, je do rodiče tohoto vnějšího vrcholu vložen nový klíč na příslušnou pozici tak, aby byla zachována seřazenost klíčů. Pokud je vrchol, do kterého byl klíč vložen, přeplněn, pak je na vrchol provedena operace štěpení popsaná v 2.9.1, která může postupovat až do kořenu, a případná změna kořenu je po vložení zaznamenána.

Odstraňování z (a,b)-stromu

Při odstraňování je nejdříve nutné nalézt odstraňovaný klíč, hledání klíče probíhá stejným způsobem, jako při operaci vyhledávání, pokud bylo vyhledávání neúspěšné, je neúspěšná i operace odstraňování. V opačném případu je klíč ze stromu odebrán. Pokud byl klíč na poslední hladině vnitřních vrcholů, je klíč ze stromu odebrán přímo. Pokud není klíč v poslední hladině vnitřních vrcholů, pak musí být nahrazen, neboť určuje pozice podstromů. Klíč je stejně jako u BVS nahrazen předchůdcem nebo následníkem ve svém podstromu. Oba tyto vrcholy musí být na poslední možné hladině pro vnitřní vrcholy, neboť by ve vyšších hladinách museli jejich syny být dva vnitřní vrcholy, což vzhledem k tomu, že se jedná o maximum a minimum, není možné.

Při odebrání klíče mohlo dojít k porušení pravidla 2 ve vrcholu, ve kterém je nyní o klíč méně, a strom pak musí být upraven. Oprava probíhá tak, že má-li vrchol, ze kterého byl klíč odstraněn alespoň jednoho sourozence, který obsahuje alespoň a klíčů a tedy $a + 1$ synů, pak je ze sourozence, který tuto podmínku splňuje, klíč vypůjčen způsobem popsaným v 2.9.1. Mají-li oba sourozenci vrcholu $a - 1$ klíčů a a synů, pak musí být provedena operace

slučování, která je také popsána v 2.9.1 a která se může opakovat až ke kořenu, kde po operaci nemusí zůstat žádný klíč. Nastane-li tento případ je změněn kořen stromu a poté je strom opět validním (a,b)-stromem.

2.9.3 Vlastnosti (a,b)-stromu

Hloubka (a,b)-stromu je $O(\log n)$, důkaz je proveden indukcí sečtením přes hladiny stromu, který obsahuje minimální počet klíčů pro svou hloubku a je uveden v [2].

Upravená operace vyhledávání postupuje po hladinách nejdále k nejvzdálenějšímu vnějšímu vrcholu, jenž jsou v (a,b)-stromu na stejné hladině, a na každé hladině je provedeno maximálně $2b-2$ porovnání. Vzhledem k tomu, že parametr b je konstanta, je počet porovnání na každé úrovni $O(1)$ a asymptotická složitost operace vyhledávání je $O(\log n)$.

Pro operaci vkládání je nejdříve nutné uvést, že operace štěpení bez propagace má konstantní asymptotickou složitost, což je zjevné z faktu, že operace štěpení vrcholu pouze vytvoří nový vrchol a přesune $\lceil \frac{b}{2} \rceil$ klíčů a jejich podstromů.

Operace vkládání musí nejdříve nalézt pozici, na kterou má být klíč vložen, což je v situaci, kdy strom klíč neobsahuje, provedeno v čase $O(\log n)$, po vložení může být provedena operace štěpení, která může postupovat až ke kořenu stromu, tento postup má asymptotickou složitost $O(\log n)$ a provádí operace štěpení v konstantním čase. Proto je i složitost vkládání v nejhorším případu $O(\log n)$.

Poslední základní operací je operace odstraňování. Obě vyvažovací operace, které při jejím použití mohou být nutné k opravení stromu mají konstantní asymptotickou složitost. Neboť operace půjčení je v podstatě jednoduchá rotace s nutností přeuspořádání klíčů vrcholu, jejichž počet je konstantní, tak i přeuspořádání lze zvládnout v konstantním čase. Operace sloučení vrcholů přesune $a-1$ vrcholů, a jejich podstromů a odstraní jeden vrchol. Proto má i operace sloučení konstantní časovou složitost.

Při odstraňování vrcholu je nejdříve ve stromu vyhledáváno, pokud klíč není nalezen, pak není možné klíč odstranit a operace končí s časovou složitostí $O(\log n)$. Jinak je klíč odstraněn a pokud vrchol, ze kterého byl odebrán klíč, obsahuje méně než $a-1$ klíčů, pak si musí vrchol klíč půjčit od sourozence operací, která má konstantní časovou složitost, a odstranění je ukončeno, nebo musí být vrchol sloučen se sourozencem, operací, která má také konstantní složitost a postupuje maximálně od nejvzdálenější hladiny, na které se nacházejí vnitřní vrcholy, do kořenu stromu a po cestě se provádějí další operace slučování, které mají konstantní složitost. Případné označení nového kořenu má také konstantní složitost, tudíž i operace odstraňování má asymptotickou složitost $O(\log n)$.

Využití (a,b)-stromy nacházejí především v B-stromech, což jsou (a,b)-stromy, pro které platí $b = 2a$ nebo $b = 2a-1$. B-stromy jsou tedy podmnoži-

nou (a,b)-stromů a dále se upravují na B+ stromy, což jsou B-stromy, jejichž klíče jsou uloženy v poslední hladině vnitřních vrcholů a vrcholy ve vyšších úrovních jsou pomocné pro vyhledávání ve stromu. Další úpravou B-stromu jsou B* stromy, které zvyšují minimální počet synů vrcholu. Výše uvedené typy nacházejí využití v databázích a v pamětích. Jedním z důvodů využití je možnost uložení celého bloku paměti do jednoho vrcholu.

Dalším přínosem (a,b)-stromů jsou červeno-černé a AA stromy, které jsou často používané BVS a jak již bylo v textu uvedeno, vznikly úpravou (a,b)-stromů na binární stromy.

2.10 Souhrn časových složitostí

Pro rekapitulaci výše uvedených časových asymptotických a amortizovaných složitostí slouží tabulka 2.1. Počet prvků stromu v tabulce značí znak n.

Typ	Vyhledávání	Vkládání	Odstraňování
Nevyvážený BVS	$O(n)$	$O(n)$	$O(n)$
AVL	$O(\log n)$	$O(\log n)$	$O(\log n)$
Červeno-černý	$O(\log n)$	$O(\log n)$	$O(\log n)$
AA	$O(\log n)$	$O(\log n)$	$O(\log n)$
Splay	$O(n), O^*(\log n)$	$O(n), O^*(\log n)$	$O(n), O^*(\log n)$
BB-α	$O(\log n)$	$O(\log n)$	$O(\log n)$
Scapegoat	$O(n), O^*(\log n)$	$O(n), O^*(\log n)$	$O(n), O^*(\log n)$
Treap	$O(n)^\ddagger$	$O(n)^\ddagger$	$O(n)^\ddagger$
(a,b)-strom	$O(\log n)$	$O(\log n)$	$O(\log n)$

Tabulka 2.1: Časová složitost typů vyhledávacích stromů

2.11 Stávající implementace vyhledávacích stromů

Využití vyhledávacích stromů jsou různorodá a pro různá využití často dochází k úpravám struktury stromu, proto většinou nebývají implementace vyhledávacích stromů obsaženy přímo v programovacích jazycích a ani jejich standardních knihovnách. Ovšem některé moderní jazyky obsahují alespoň abstraktní struktury slovník a množina, které splňují mnoho vlastností vyhledávacích stromů a v mnoha případech jsou implementovány vyhledávacími stromy. V jazyku C++ se jedná o struktury `std::set` a `std::map`.

Pro práci s vyhledávacími stromy musí být buď vytvořena nová implementace, nebo použita již existující implementace. Ze stávajících implementací se pro využití nabízejí různé knihovny, v jazyku C++ se jedná například o knihovnu `Boost.Intrusive`[23], jenž je obsažena v často užívané sadě knihoven

‡V průměru $O(\log n)$.

`Boost`. Z popsaných typů obsahuje knihovna strukturu treap a červeno-černé, AVL, scapegoat a splay stromy. Dalšími knihovnami obsahujícími implementace vyhledávacích stromů v jazyku C++ jsou knihovny `cpp-btree`[23] obsahující B-strom nebo `Ygg`[24] obsahující červeno-černé, BB-α a další v práci nepopsané typy.

Kapitola 3

Implementace

V praktické části práce jsou implementovány všechny typy, jenž byly popsány v předchozí kapitole, jedná se tedy o nevyvážený BVS, stromy AVL, AA, splay, červeno-černý, (a,b), BB-α, scapegoat a strukturu treap. Nevyvážený BVS je implementován jednosměrným i obousměrným spojovým seznamem. AA strom je také implementován dvěma způsoby, v jedné implementaci se pro vyvažování stromu využívají barvy vrcholů a ve druhé implementaci je strom vyvažován pomocí úrovní vrcholů. BB-α, scapegoat stromy a výše zmíněný nevyvážený BVS jsou implementovány pomocí jednosměrného spojového seznamu a ostatní typy a jedna implementace nevyváženého BVS jsou implementovány obousměrným spojovým seznamem.

Jednotlivé typy jsou implementovány jako třídy, jejichž vrcholy jsou také třídy, a operace jsou implementovány jako jejich metody. Průběh operací jednotlivých typů je jednotný s jejich popisem v částech kapitoly 2. Všechny typy umí následující metody:

- **Insert(x)** – metoda se pokusí vložit klíč x do stromu, pokud uspěje vrací `true`, v případě neúspěchu vrací `false`.
- **Delete(x)** – metoda se pokusí odstranit klíč x ze stromu, pokud uspěje vrací `true`, v případě neúspěchu vrací `false`.
- **Search(x)** – metoda se pokusí najít vrchol ve stromu, který obsahuje klíč x, v případě úspěchu vrací ukazatel na vrchol obsahující klíč x, jinak vrací nulový ukazatel, v jazyce C++ `nullptr`.
- **Size()** – metoda vrací počet prvků ve stromu.
- **Clear()** – metoda odstraní všechny prvky ze stromu a uvolní prostředky pro tyto prvky alokované.
- **GetSorted()** – metoda vrací vzestupně seřazené klíče ve stromu uložené v poli (ve struktuře `std::vector`).

- **PrintTree(os)** – metoda vypíše strom pomocí vypsání informací o vrcholech do datového proudu `os` (standardní výstup, soubor, ...).

Implementace jsou provedeny v jazyce C++, který je vhodný svou efektivitou, využitím, a vůči jazyku C nabízí třídy a objekty, které jsou pro implementaci a následné využití vhodné. V implementacích je využit polymorfismus pro snadné a vhodné využití implementací. Další vlastnost jazyku C++, která je v práci využita, jsou šablony tříd, které umožňují využití implementací pro různé datové typy a třídy, uživatel může tedy jako klíče vyhledávacích stromů používat i vlastní třídy. Třídy použité jako klíče musí mít implementován kopírovací konstruktor, operátor = a relační operátor <. Využití šablon tříd v jazyku C++ ovšem znamená, že implementační kód je umístěn pouze v hlavičkových souborech, aby mohl být zahrnut.

Implementace zohledňuje především efektivitu programu a to vede například k tomu, že některé metody vrcholů neprovádějí kontrolu, zdali je volání metody s argumenty validní a uživatel si při úpravě knihovny musí počítat s touto vlastností implementace.

V práci jsou využity některé prvky standardní knihovny jazyku C++, jako například struktury `std::vector` a `std::stack`, také jsou využity knihovny `cmath` pro výpočet logaritmu a odmocniny a `random` pro náhodnou generaci priorit vrcholů struktury treap.

3.1 Struktura implementace

Implementační část práce obsahuje 9 tříd pro reprezentaci vrcholů různých typů vyhledávacích stromů. Třídu `CTester`, která slouží k provedení testů a měření jejich výsledků, třídu `CDataGenerator` pro generování vstupních dat, 12 tříd pro vlastní implementace jednotlivých typů a 5 tříd pro stávající implementace. Implementace dále obsahuje další podpůrné prvky, jako třeba funkci pro porovnání čísel s plovoucí desetinou čárkou, výčtové typy pro barvy v červeno-černé stromy a pro operace a výjimky, které mohou při nesprávné práci s implementacemi nastat.

Nejzákladnějšími prvky implementace jsou abstraktní třídy `CSearchTree`, jenž obsahuje všechny uvedené virtuální metody vyhledávacího stromu, představující vyhledávací strom a třída `CNode`, která představuje vrchol vyhledávacího stromu. Tyto třídy slouží k dosažení polymorfismu a pro možnost rozšířitelnosti knihovny o další typy a různé způsoby implementací.

Implementace, jak již bylo výše zmíněno, obsahuje 5 tříd, které obalují knihovní implementace tak, aby implementovaly rozhraní `CSearchTree` a dalších 5 tříd, které slouží jako jejich vrcholy a dědí od třídy `CNode`. Jedná se o třídy `CBoostAVLTree`, `CBoostRBTree`, `CBoostSplayTree`, `CBoostTreap`, které obalují struktury importované z knihovny `Boost.Intrusive`[23], a `CLibBTree`, jenž obaluje B-strom z knihovny `cpp-btree`|[25].

3.1.1 Nevyvážený BVS

První implementací nevyváženého BVS je implementace jednosměrným spojovým seznamem. Tuto implementaci představuje třída `CLBST`, která dědí (vzhledem k abstraktnosti implementuje) od třídy `CSearchTree`. Vrcholem použitým v této implementaci je vrchol `CBinaryNode`, který dědí od třídy `CNode` a má 3 členské proměnné. Proměnné `m_Left`, respektive `m_Right`, sloužící k uložení ukazatele na levého, respektive pravého, syna. Pokud je syn vrcholem vnějším, je v ukazateli na něj uložena hodnota `nullptr`. Poslední členskou proměnnou této třídy je `m_Key` s uloženou hodnotou klíče.

Druhou implementací nevyváženého BVS je implementace obousměrným spojovým seznamem, její realizace se jmenuje `CBST`, implementuje rozhraní `CSearchTree` a jeho metody. Třída používaná jako vrchol v této implementaci je třída `CDLBinaryNode`, která dědí od třídy `CBinaryNode` a navíc obsahuje zpětný ukazatel na rodiče vrcholu v členské proměnné `m_Parent`. Pokud je vrchol kořenem stromu, je v `m_Parent` uložen nulový ukazatel.

Logika obou implementací je velmi podobná a liší se především rozšířením vrcholů o zpětný ukazatel na rodiče a práci s tímto ukazatelem. Obě třídy představující nevyvážený BVS, obsahují ukazatel na kořen stromu v podobě členské proměnné `m_Root` a velikost stromu, tedy počet prvků ve stromu, v členské proměnné `m_Size`. V rámci obou tříd jsou také implementovány metody pro levou a pravou rotaci a také metoda pro vyříznutí vrcholu ze stromu.

Obě implementace přepisují (implementují) všechny zděděné virtuální metody. Průběh operací probíhá jako v jejich definici, metody pro získání počtu vrcholů a odstranění vrcholů stromu jsou triviální, metoda pro získání seřazených klíčů prochází stromem iterativně se zásobníkem, do kterého se ukládají vrcholy pro návrat, a ukládá klíče navštívených vrcholů, a metoda pro výpis stromem postupuje jako metoda `GetSorted`, ale místo uložení klíčů do pole, vypíše klíče a jejich hloubku do zadaného datového proudu.

3.1.2 AVL strom

Implementace AVL stromu je provedena v rámci třídy `CAVLTree`, která dědí od třídy `CBST`. Jako vrchol využívaný v této implementaci je třída `CAVLNode`, která dědí od `CDLBinaryNode`, a kromě zděděných ukazatelů obsahuje také informaci o hloubce, v které se vrchol ve stromu nachází a která je využívána pro vyvažování AVL stromu.

Třída `CAVLTree` přepisuje metody pro vkládání a odstraňování s rozšířením o kontroly hloubky podstromu a využití vyrovnávacích operací, jenž jsou v případu AVL stromu levou a pravou rotací a jsou jako metody zděděny. Třída také přepisuje metodu vyhledávání tím způsobem, že je pro vyhledávání využita metoda sloužící k vyhledávání v `CBST`, ovšem vracený ukazatel je přetypován na `CAVLNode`.

3.1.3 Červeno-černý strom

Červeno-černý strom je implementován v rámci třídy `CRBTree`, která dědí od `CBST`. Vrchol červeno-černého stromu je realizován třídou `CRBNode`, která dědí od `CDLBinaryNode` a má uloženou informaci o barvě vrcholu v členské proměnné `m_Color`. Barva je realizována výčtovým typem `Color` deklarovaným ve třídě `CRBNode` a jak z definice červeno-černého stromu vyplývá, `Color` má 2 hodnoty: červená a černá.

Třída `CRBTree` má dvě nové metody: `InsertFix` a `DeleteFix`, které slouží ke kontrole a k případné opravě stromu, pokud došlo k porušení některého z pravidel červeno-černého stromu.

Operace vkládání a odstranění jsou upraveny pro vrcholy `CRBNode` a na závěr mohou volat výše uvedené metody pro kontrolu a opravu stromu. Operace vyhledávání využívá operaci vyhledávání rodičovské třídy a přetypovává vracený ukazatel. Poslední metoda `CRBTree`, jenž je vůči své rodičovské třídě překryta, je metoda sloužící pro výpis stromu, která je upravena tak, aby vypisovala i barvy vrcholů.

3.1.4 AA strom

Jak již bylo v kapitole zmíněno AA strom je implementován dvěma způsoby. První implementace, která využívá pro vyvažování barvy vrcholu, je realizována třídou `CAAAltTree`, která dědí od třídy `CRBTree`, a využívá `CRBNode` jako vrchol. První úpravou oproti třídě `CRBTree` je překrytí metod `InsertFix` a `DeleteFix` takové, že v případu potřeby je strom upraven na validní AA strom. Druhou úpravou je přidání metod `Split` a `Skew`, které představují stejnojmenné operace AA stromu.

Implementace červeno-černých stromů je poměrně složitá a přidáním dalších pravidel je implementace AA stromu ještě složitější. Proto druhá implementace AA stromu využívá pro vyvažování úroveň vrcholu a invarianty popsané v [14]. Tento způsob je realizován třídou `CAATree`, jenž dědí od `CBST`, a pracuje s vrcholy realizovanými třídou `CAANode`, která dědí od třídy `CDLBinaryNode`, a kromě ukazatelů obsahuje i celé číslo reprezentující úroveň vrcholu v členské proměnné `m_Level`.

`CAATree` má nové metody `Skew` a `Split` představující vyvažovací operace AA stromu a metody `InsertFix` a `DeleteFix` volané pro validaci a opravení stromu po provedení operace vložení nebo odstranění.

Operace vkládání a odstranění jsou překryty tak, aby pracovaly s vrcholy `CAANode` a volaly metody `InsertFix` a `DeleteFix`. Operace vyhledávání je opět obohacena o přetypování vráceného ukazatele. Vhledem k přidání nové informace do vrcholů stromu, je metoda pro výpis stromu překryta tak, aby byly pro vrcholy `CAATree` vypisovány i jejich úrovně.

3.1.5 Splay strom

Splay strom je realizován pomocí třídy `CSplayTree`, která dědí od třídy `CBST` a vrcholy nepotřebují žádné dodatečné informace, proto je i v `CSplayTree` využíván vrchol `CDLBinaryNode`. Třída má novou metodu `Splay(v)`, která z vrcholu v udělá kořen stromu.

Operace vkládání, vyhledávání a odstranění jsou překryty tak, aby operací `Splay` přesunuly vrchol určený z definice těchto operací. Průběh operace odstraňování je shodný s druhým popsaným způsobem odstranění prvku ze splay stromu v části 2.5.2.

3.1.6 BB-α strom

Váhově vyvážený BVS je realizován třídou `CBBTree`, která dědí od `CLBST` a která jako vrcholy používá třídu `CBBNode`, která k ukazatelům zděděným od `CBinaryNode` přidává také informaci o velikost podstromu určeného konkrétním vrcholem v členské proměnné `m_Size`. `CBBTree` si ukládá zadanou informaci o maximální povolené rovnováze vrcholu stromu v členské proměnné `m_VertexRatio`, další přidanou členskou proměnnou je `m_DRRatio`, v níž je uloženo číslo, které rozhoduje, zdali bude provedena jednoduchá nebo dvojitá rotace. V konstruktoru je při vytváření objektu provedena kontrola, zdali parametr α splňuje podmínky BB-α stromu popsané v 2.6.

`CBBTree` má oproti `CLBST` nové metody `GetVertexRatio(v)`, která vypočítá rovnováhu vrcholu v, a `FixBalance(v)`, která rozpozná, zdali ve vrcholu v došlo k porušení rovnováhy stromu. Došlo-li k porušení rovnováhy je provedena rotace pro její napravení a vrácena hodnota `false`, pokud k porušení rovnováhy nedošlo, je vrácena hodnota `true`.

Třída také překrývá rotace tak, aby v nich došlo k upravení velikosti podstromů, v nichž dochází ke změnám. Další přepsanou metodou je metoda `Size`, která vzhledem k uchovávání velikosti podstromů ve vrcholech vrací velikost podstromu tvořeného vrcholem stromu. Operace vkládání a odstraňování jsou překryty tak, aby odpovídali definicím v 2.6. Metoda vyhledávání je překryta tak, aby vrácený ukazatel byl na vrchol typu `CBBNode`. Poslední přepsanou metodou je metoda sloužící pro výpis stromu, která vzhledem k nové informaci ve vrcholech, vypisuje nyní i velikost podstromu určeného vypisovaným vrcholem.

Vzhledem k tomu, že se jedná o implementaci jednosměrným spojovým seznamem, jsou úpravy velikostí podstromů a případné rotace prováděny již při procházení stromem. Pokud jsou operace neúspěšné, musí se znovu projít stromem a obnovit velikosti, které byly upraveny při sestupu.

Vzhledem k tomu, že se ve třídě `CBBTree` pracuje s rovnováhou, která se pohybuje v množině reálných čísel, které se v jazyku C++ ukládají do datového typu `double`, u kterých může při výpočtech docházet k drobným nepřesnostem, není výsledek operátoru == vždy správný. Proto je implementována

a využita funkce `doubleCompare`, která je pro touto vlastnost čísel s plovoucí čárkou v jazyku C++ přizpůsobena.

3.1.7 Scapegoat strom

Implementací scapegoat stromu je třída `SGTree`, která vzhledem k tomu, že se jedná o úpravu BB-α stromu, dědí od jeho implementace `CBBTree`. Používá také stejnou třídu pro vrcholy jako `CBBTree` a to `CBBNode`. Novou informací `SGTree` stromu je reálné číslo `m_LogDiv`, jenž je konstanta využita k výpočtu logaritmu jiné báze než 2. V konstruktoru dochází ke kontrole validity rovnováhy a výpočtu `m_LogDiv`.

První novou metodou třídy je metoda `RebuildTree(v)`, která z podstromu s kořenem v sestaví dokonale vyvážený BVS, tato metoda postupuje jako poslední způsob popsaný v části 2.7.1. Druhou přidanou metodou je metoda `FindAndRebuild`, která, pokud došlo vložením nového prvku k porušení balančního kritéria stromu, najde vhodného obětního beránka a podstrom určený tímto vrcholem přestaví metodou `RebuildTree`.

`SGTree` přepisuje metodu vkládání nového prvku tak, aby odpovídala definici, především kontrolou validity a voláním `FindAndRebuild`. Operace odstraňování je také upravena tak, aby odpovídala definici a ke sledování poslední velikosti, kdy došlo ke kontrole, je využita, v případu `SGTree` a `BBTree` jinak nevyužitá, členská proměnná `m_Size`.

3.1.8 Treap

Implementaci struktury treap představuje třída `CTreap`, která dědí od `CBST`. Jako vrchol struktury treap je využita třída `CTreapNode`, která dědí od třídy `CDLBinaryNode` a zděděné ukazatele rozšiřuje o informaci o prioritě vrcholu uložené v členské proměnné `m_Priority`. Třída `CTreap` má oproti `CBST` navíc členské proměnné `m_MaxPriority`, v níž je udržována informace o maximální prioritě, `m_Generator` s generátorem náhodných čísel ze standardní knihovny C++ `random` typu `std::mt19937` a rovnoměrné rozdělení `m_Distribution` typu `std::uniform_int_distribution<uint32_t>` také z knihovny `random`, pomocí kterého jsou vrcholům přidělovány priority, tak aby každé celé číslo v intervalu $\langle 0,$ `m_MaxPriority`$-1\rangle$ mělo stejnou pravděpodobnost výběru.

V rámci třídy dochází k překrytí operace vkládání, aby fungovala jako v definici, operace vyhledávání tak, aby byl vrácen ukazatel na vrchol typu `CTreapNode`, operace odstranění, která přesune odstraňovaný vrchol na takovou pozici, z které ho lze triviálně odebrat, tedy tak, aby alespoň jeden ukazatel na syna byl nulový, a metody pro výpis stromu přidáním informace o prioritě vrcholu do výpisu.

3.1.9 (a,b)-strom

Realizací (a,b)-stromu je třída `ABTree`, která dědí od třídy `CSearchTree`, má následující členské proměnné: `m_Root` s ukazatelem na kořen stromu, `m_Size` s počtem klíčů ve stromu a proměnné `m_A` a `m_B` pro uložení parametrů určujících počet potomků vrcholu ve stromu. Jako vrchol `ABTree` je použita třída `ABNode`, která dědí od třídy `CNode`.

`ABNode` má následující členské proměnné: `m_Parent` s ukazatelem na rodiče vrcholu, `m_SubtreePos`, v níž je uloženo kolikátým synem svého rodiče je tento vrchol, `m_KeyCount` s počtem klíčů ve vrcholu, pole `m_Keys` s uloženými klíči ve vrcholu, `m_Max` s maximálním počtem klíčů ve vrcholu a pole `m_Subtrees` s ukazateli na syny. Pokud se vrchol nachází na poslední hladině vnitřních vrcholů, nabývá `m_Subtrees` hodnotu `nullptr` pro úsporu paměti. Třída `ABNode` alokuje rovnou blok paměti, který je dostatečně velký pro maximální počet klíčů, předchází se tím častým realokacím za cenu méně efektivního využití paměti.

Třída `ABNode` má následující metody: `Allocate`, která alokuje adresy pro ukazatele na syny, `Insert`, která slouží ke vložení klíče do vrcholu, jehož synové jsou pouze vnější vrcholy, metody `InsertWithLeftSubtree`, respektive `InsertWithRightSubtree`, které vloží do vrcholu klíč s jeho levým, respektive pravým, podstromem, metody `RemoveWithLeftSubtree`, respektive `RemoveWithRightSubtree`, které odstraní klíč ze zadané pozice s jeho levým, respektive pravým, podstromem a metodu `InsertSubtree`, která vloží podstrom na pozici nejpravějšího podstromu.

V konstruktoru `ABTree` dochází ke kontrole validity zadaných argumentů (a,b)-stromu, novými metodami třídy jsou metody `Split`, `Fusion` a `Borrow`, které představují vyvažovací operace (a,b)-stromu.

Vzhledem k tomu, že je `CSearchTree` rozhraní, musí `ABTree` všechny zděděné virtuální metody implementovat. Operace implementuje tak, jak je popsáno v sekci 2.9, metody `Size` a `Clear` jsou triviální, `GetSorted` je upravena pro průchod klíčů ve vrcholu a `PrintTree` vypisuje kromě klíče a hloubky, ve které se vrchol nachází, i kolikátým synem je vrchol v němž se nachází tento klíč.

3.1.10 Třídy pro generování a provedení testů

Pro generování testovacích dat slouží třída `CDataGenerator`, jejíž metody vygenerují soubory se vstupními daty. Třída `CDataGenerator` obsahuje metody pro generování všech testů v následující kapitole a další metody, které ke generování těchto dat využívá.

Pro provedení a změření výsledků testů slouží třída `CTester`, která obsahuje metody pro načtení testu ze vstupního toku dat, metodu pro provedení testu pro uložená data a změření výsledků testu, metodu, která vypíše výsledky všech stromů pro uložená data, metody sloužící pro porovnání parame-

trů typů, které pro ideální výkon potřebují vhodné parametry, a metody pro provedení a změření výsledků testů uvedených v další kapitole.

Kapitola 4

Porovnání

V poslední kapitole dochází k měření a porovnání vytvořených implementací vyhledávacích stromů z kapitoly 2 a vybraných existujících implementací vyhledávacích stromů z knihoven jazyku C++.

Měření výkonností jednotlivých implementací je provedeno pro různé scénáře práce s vyhledávacími stromy na školním svazku STAR, který využívá dávkový plánovač ke spouštění úloh na výpočetních uzlech a zaručuje tím optimální běh programu.

Program je zkompilován překladačem `g++` verze `10.2.1` s přepínači `-Wall`, `-pedantic`, `-Werror` a `-Wextra` pro kontrolu kódu, přepínači `-I` pro přidání knihoven a přepínačem `-O3` pro optimalizaci, který sice neměl na rychlost některých typů výrazný vliv, ovšem u některých typů došlo k výraznému zrychlení běhu programu.

4.1 Výsledky měření

Pro měření výkonnosti vyhledávacích stromů v testu o velikosti n, jsou nejdříve vygenerována vstupní data, která obsahují příslušný počet operací vzhledem ke zvolenému n a následně dojde k provedení testu pro všechny určené typy vzhledem ke scénáři a n. Pro každý test jsou vybrány příslušné optimální parametry BB-α, scapegoat a (a,b)-stromu.

Všechny následující testy jsou pro každé n provedeny třikrát a z jejich výsledků je pro porovnání vybrán jejich medián. Typ klíče všech porovnávaných vyhledávacích stromů je datový typ `uint32_t`.

Výsledky jsou zakresleny do grafů pomocí nástroje `GNUPlot`[26] a stupnice vygenerovaných grafů je pro přehlednost grafů logaritmická.

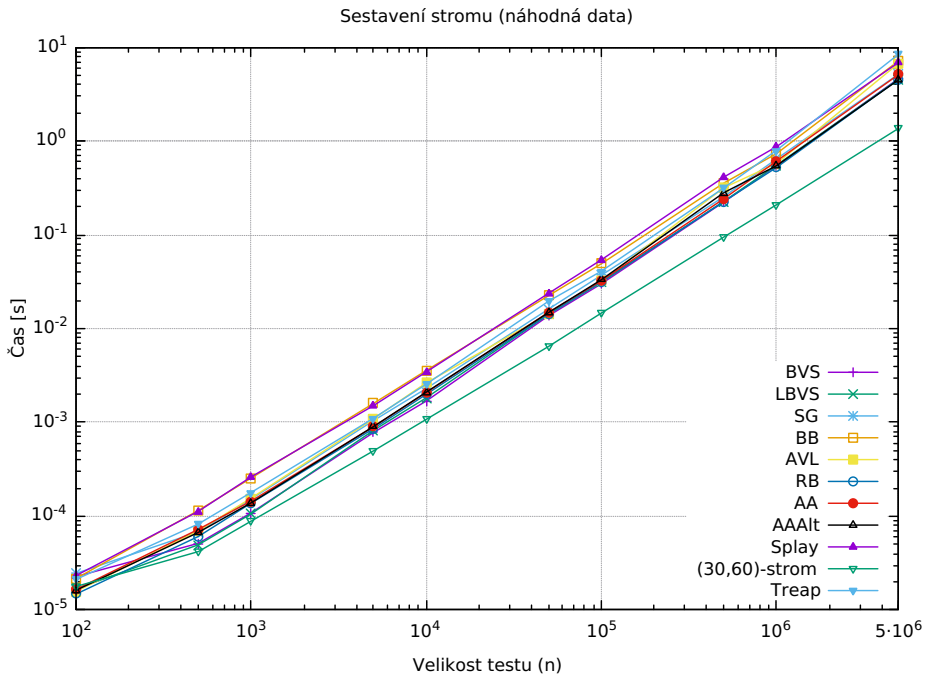
4.1.1 Operace vkládání

První test obsahuje pouze operaci vkládání a simuluje práci se stromem, jenž je naplněn n prvky, tedy sestavení stromu. Test ukazuje, po jaké době je vy-

hledávací strom pomocí n operací vkládání naplněn n prvky tak, aby splňoval všechny podmínky vedoucí z definice stromu.

Náhodná data

Nejdříve je strom plněn náhodně vygenerovanými unikátními daty. Pro tento test a množinu vstupních dat vyšel pro BB-α strom nejlépe parametr $\alpha = 0{,}29$, pro scapegoat strom parametr $\alpha = 0{,}30$ a jako (a,b)-strom vyšel nejlépe (30,60)-strom. Výsledky jsou znázorněny v grafu 4.1.



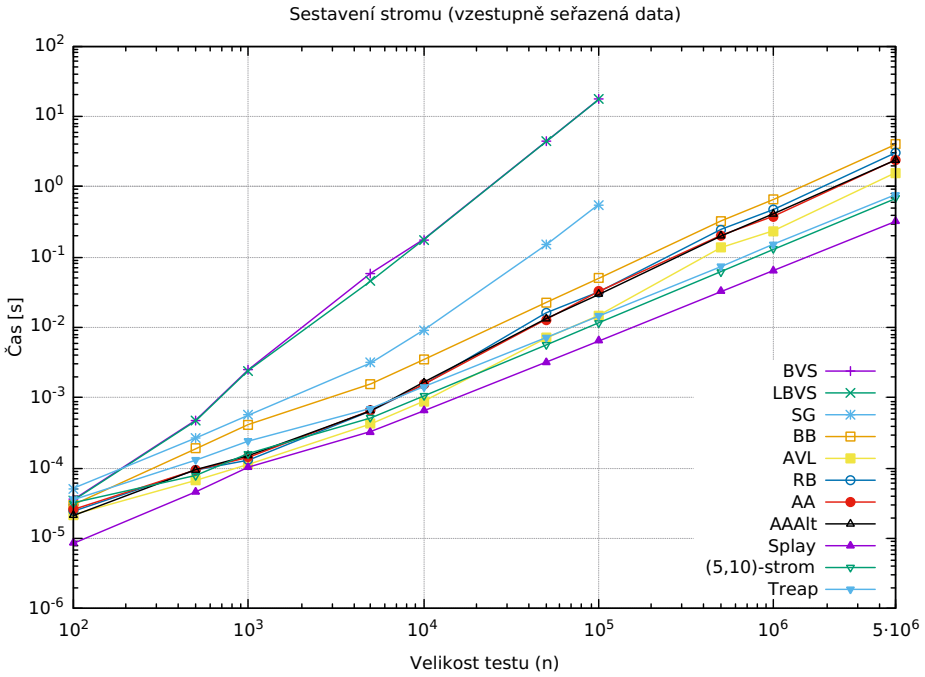
Obrázek 4.1: Test vkládání náhodných dat do stromu

Z výsledků testů se potvrdilo tvrzení zmíněné v 2, že vkládání náhodných dat do vyhledávacího stromu má složitost $O(\log n)$. Nejlepším stromem pro vkládání náhodných dat do vyhledávacího stromu se ukázal (30,60)-strom, který je velikostí vrcholů, i tím, že vzhledem k 4 bajtové velikosti klíče typu `uint32_t`, dokáže využívat 256 bajtů velké bloky cache paměti, pro tuto posloupnost operací a dat vhodný. Rozdíly délek běhu ostatních stromů byly velmi nízké a splay strom se ukázal v tomto testu jako těsně nejhorší pro data $n \leq 10^6$ a pro $n = 5 \cdot 10^6$ nejhorších výsledků dosáhla struktura treap.

Vzestupně seřazená data

Následně je strom plněn seřazenými daty, v tomto případu vzestupně seřazenými daty. Zde se jako nejlepší parametry ukázaly parametry $\alpha = 0{,}29$

pro BB-α strom, $a = 5$, $b = 10$ pro (a,b)-strom a $\alpha = 0{,}05$ pro strom typu scapegoat, který se touto hodnotou může v některých případech stát téměř lineárním seznamem. Výsledky testu pro tyto data jsou znázorněny v grafu 4.2.



Obrázek 4.2: Test vkládání vzestupně seřazených dat do stromu

Ve výsledcích se projevují rozdíly mezi časovými asymptotickými, nebo alespoň amortizovanými, složitostmi vyhledávacích stromů. Obě implementace binárního vyhledávacího stromu, jejichž časová asymptotická složitost je pro tato data $\Theta(n)$, mají jednoznačně nejhorší výsledky. Scapegoat strom, který sice vzhledem k nízkému parametru α nemá vždy optimální hloubku a jeho výsledky se nepřibližují nejlepším výsledkům, avšak alespoň občasným vyvažováním dosahuje mnohem lepších výsledků než nevyvážené BVS.

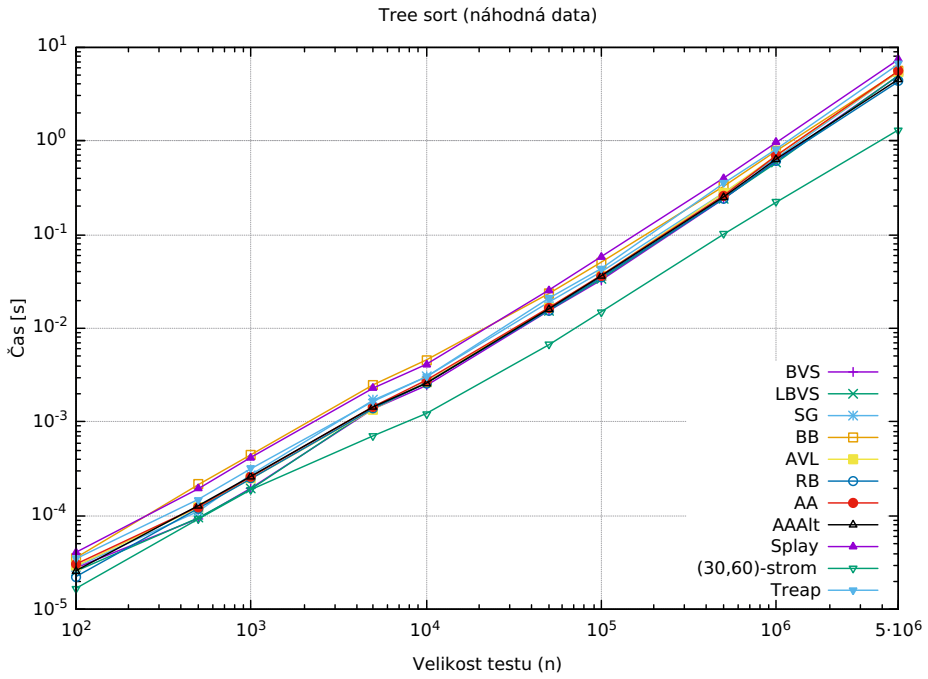
Nejlepší typem pro tato data se ukázal splay strom, který pro každý nově vložený prvek provede právě jednu levou rotaci. O něco horších, ale přesto velmi dobrých výsledků, dosahuje (a,b)-strom, zde (5,10)-strom. Za zmínku také stojí AVL strom, jenž pro malá n dosahuje výsledků porovnatelných se splay stromem, a struktura treap, vzhledem k tomu že pro $n = 5 \cdot 10^6$ lepších výsledků dosahují pouze (5,10)-strom a splay strom a její výsledky jsou porovnatelné s výsledky (5,10)-stromu. Rozdíly mezi výsledky ostatních typů jsou velmi nízké s výjimkou BB-α stromu, který je z neuvedených typů v testu nejhorším.

4.1.2 Řazení vkládáním do stromu

Druhý test obsahuje kromě operace vkládání i operaci pro získání seřazené posloupnosti n vložených unikátních klíčů. Test simuluje algoritmus `tree sort`, jenž seřadí vložené klíče.

Náhodná data

Nejdříve je algoritmus simulován pro náhodnou posloupnost klíčů. Parametry jsou vzhledem k vlastnosti testu zvoleny stejné jako pro test vkládání náhodných dat. Tedy pro BB-α strom je parametr $\alpha = 0{,}29$, pro SG strom parametr $\alpha = 0{,}30$ a jako (a,b)-strom je použit (30,60)-strom. Znázornění výsledků se nachází v grafu 4.3.

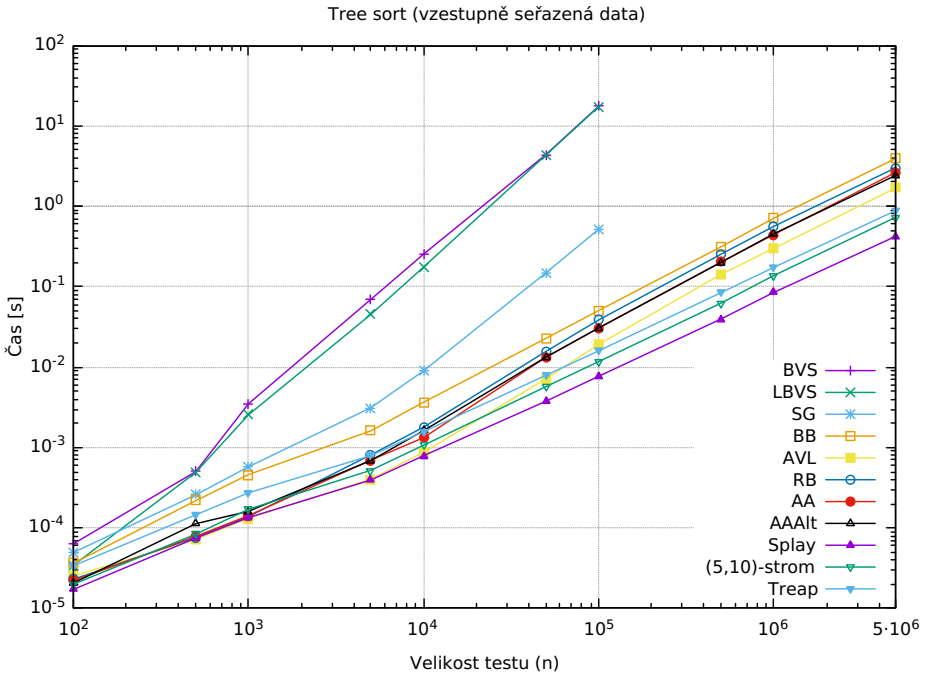


Obrázek 4.3: Test řazení vkládáním náhodných dat do stromu

Výsledky jsou velmi podobné výsledkům předchozího testu pro náhodná data, ovšem rozdíl mezi (30,60)-stromem a shlukem ostatních typů vyhledávacího stromu se zvýšil a výsledky (30,60)-stromu jsou jednoznačně nejlepší.

Vzestupně seřazená data

Poté je algoritmus simulován pro již seřazená data. Stejně jako pro náhodná data i zde jsou použity stejné parametry jako v předchozím testu, tedy pro BB-α strom parametr $\alpha = 0{,}29$, pro SG strom parametr $\alpha = 0{,}05$ a jako (a,b)-strom je použit (5,10)-strom. Pro znázornění výsledků viz graf 4.4.



Obrázek 4.4: Test řazení vkládáním vzestupně seřazených dat do stromu

Vzhledem k charakteristice testu jsou výsledky téměř totožné s výsledky testu vkládání n vzestupně seřazených klíčů do stromu a ani (5,10)-strom nedosahuje žádného výrazného zlepšení. Nejlepším typem v testu je tedy splay strom, a nejhoršími nevyvážené BVS a o něco lepší je SG strom. Druhým nejlepším typem tohoto testu se stal (5,10)-strom následovaný strukturou treap. Ostatní typy mají pouze nízké rozdíly ve výsledcích.

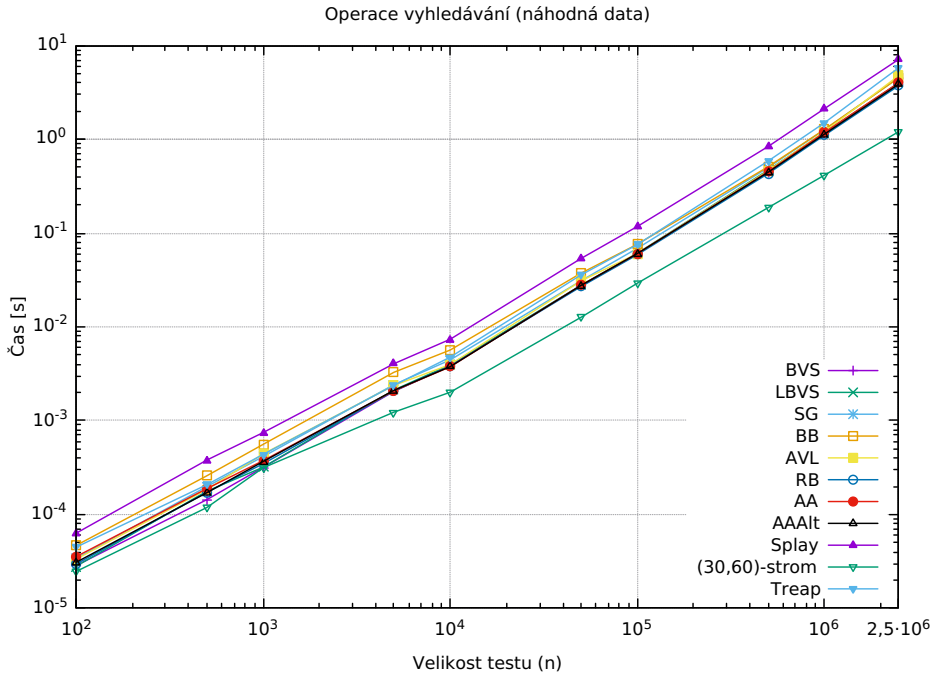
4.1.3 Operace vyhledávání

V dalším testu je posloupnost vstupních dat taková, že je nejdříve vloženo n unikátních klíčů do stromu a následně je ve vyhledávacím stromu provedeno n operací vyhledávání, jejichž úspěšnost je přibližně 50%. Test simuluje situaci, kdy nejdříve dojde k sestavení vyhledávacího stromu se všemi klíči, které bude strom obsahovat, a následně se ve stromu vyhledává. Výsledky testu lze také porovnat s výsledky měření sestavení stromu pro znázornění výkonnosti operace vyhledávání pro různé typy.

Náhodná data

V tomto případu je do stromu vloženo n náhodných unikátních prvků v náhodném pořadí a poté se ve stromu vyhledává s úspěšností cca 50%. Nejlepšími parametry pro tento test a data se staly parametry $\alpha = 0{,}19$, respektive

$\alpha = 0{,}35$, pro BB-α strom, respektive SG strom. Nejlepším (a,b)-stromem se stal (30,60)-strom. Výsledky jsou znázorněny v grafu 4.5.



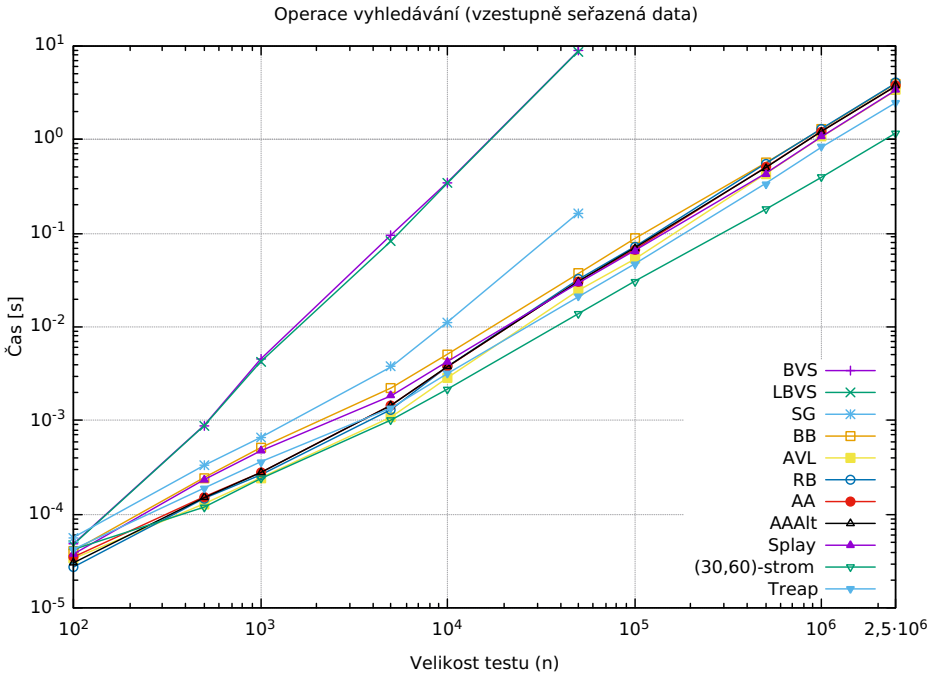
Obrázek 4.5: Test vyhledávání náhodných dat ve stromu sestaveném z náhodné posloupnosti dat

Pro $n > 1000$ je jednoznačně nejlepším typem opět (30,60)-strom. Pro všechna n je nejhorším typem splay strom, který na rozdíl od ostatních typů provádí vyvažovací operace i při vyhledávání ve stromu. Rozdíly mezi výsledky ostatních typů jsou minimální. Za zmínku však stojí to, že pro $n \leq 1000$ dosahuje struktura treap nejlepších výsledků a pro vyšší n struktura postupně v pořadí propadává a pro $n = 2{,}5 \cdot 10^6$ dosahuje druhého nejhoršího výsledku.

Vzestupně seřazená data

Do stromu je vloženo n vzestupně seřazených klíčů a následně je ve stromu provedeno n operací vyhledávání s úspěšností cca 50%. Parametry stromů v testu vyhledávání ve stromu, jenž byl sestaven z vzestupné posloupnosti klíčů, byly zvoleny $\alpha = 0{,}29$ pro BB-α strom a $\alpha = 0{,}05$ pro SG strom. Nejlepším (a,b)-stromem se opět stal (30,60)-strom. Výsledky jsou znázorněny v grafu 4.6.

Oproti prvnímu testu pro vzestupně seřazené klíče si zde výrazně polepšil (a,b)-strom, který se stal nejlepším typem pro téměř všechna n s výjimkou velmi nízkých n. Zatímco splay, který byl v prvním testu s totožným uspořádáním nejlepším typem, se v tomto testu propadl, a i AVL strom dopadl o po-



Obrázek 4.6: Test vyhledávání náhodných dat ve stromu sestaveném ze vzestupné posloupnosti dat

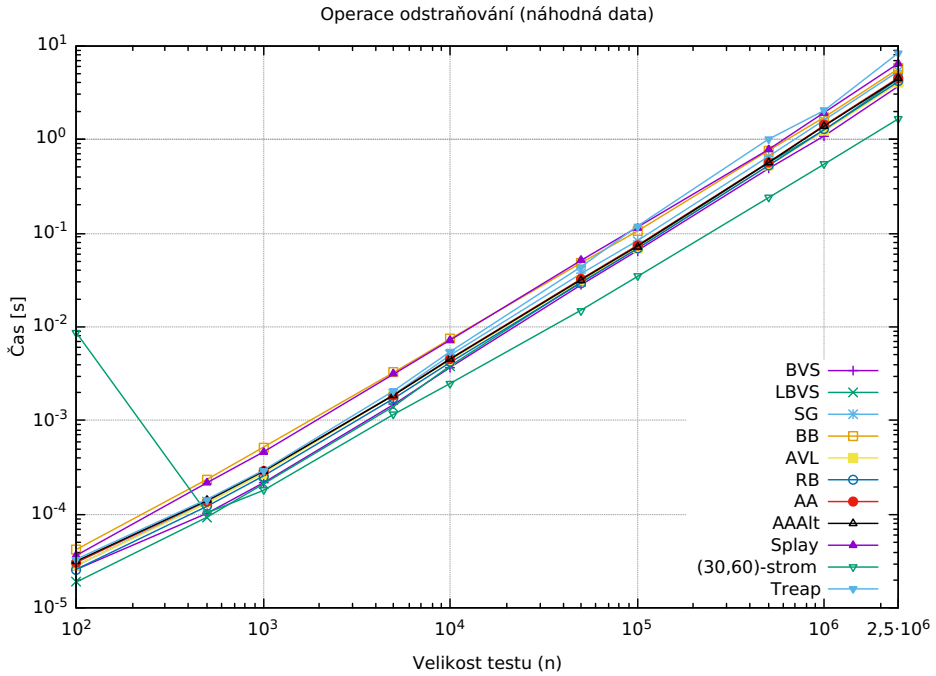
znání hůře. Nejhoršími typy byly opět obě implementace nevyváženého BVS a lepších výsledků dosáhl SG strom, ovšem nedostatečných v porovnání s ostatními vyváženými vyhledávacími stromy. Obě implementace AA stromu, BB-α strom, RB strom a splay strom pak měly velmi podobné výsledky a struktura treap dosáhla o něco lepších výsledků než tento shluk vyhledávacích stromů.

4.1.4 Operace odstraňování

Následující test je podobný testu předchozímu s tím rozdílem, že po naplnění stromu n prvky je ve stromu provedeno n operací odstraňování, jenž jsou všechny úspěšné. Výsledný strom tohoto testu tedy neobsahuje jediný klíč. Test simuluje práci s množinou klíčů, při níž se ke každému prvku přistupuje pouze jednou.

Náhodná data

Do stromu je vloženo n náhodných unikátních klíčů v náhodném pořadí, následně jsou klíče v náhodném pořadí ze stromu odstraněny. Parametrem BB-α stromu se v tomto testu stal parametr $\alpha = 0{,}19$, parametr SG stromu byl $\alpha = 0{,}35$ a nejlepším (a,b)-stromem byl opět (30,60)-strom. Výsledky jsou znázorněny v grafu 4.7.



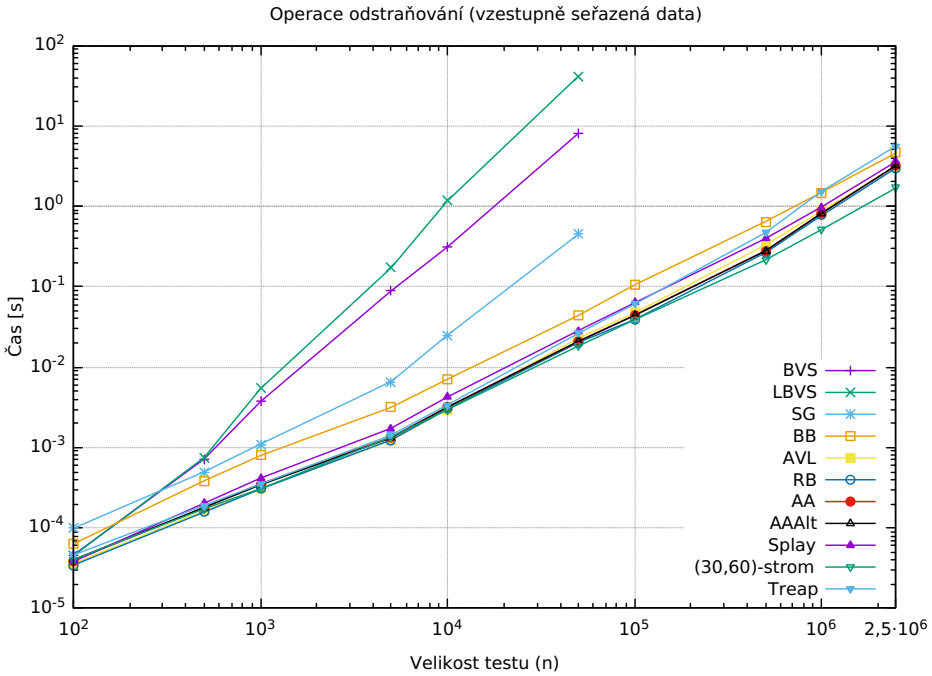
Obrázek 4.7: Test odstraňování náhodných dat ze stromu sestaveného z náhodné posloupnosti dat

Výsledek (30,60)-stromu pro $n = 100$ je velmi špatný a suverénně nejhorší. Ovšem pro $n = 500$ už jsou jeho výsledky porovnatelné s ostatními typy a pro $n \geq 1000$ už je (30,60)-strom v testu nejlepší. Pro $n \leq 10000$ jsou s výjimkou (30,60)-stromu pro $n = 100$ nejpomalejšími typy BB-α strom se splay stromem, ovšem pro vyšší n už jsou výsledky těchto dvou typů téměř totožné s výsledky většiny ostatních typů. Pro velmi vysoké n si v testu vede nejhůře struktura treap, jejíž implementace operace odstraňování je poměrně jednoduchá a pro velká n pomalá.

Vzestupně seřazená data

Vloženo bylo n vzestupně seřazených klíčů, které byly následně odstraněny v náhodném pořadí. Nejlepšími parametry pro tento scénář se staly parametry $\alpha = 0{,}29$ pro BB-α strom a parametr $\alpha = 0{,}05$ pro SG strom. Nejlepším (a,b)-stromem je znovu (30,60)-strom. Pro znázornění výsledků viz graf 4.8.

Jako ve všech předchozích testech s vzestupně seřazenou posloupností dat, i zde jsou nejpomalejší obě implementace nevyváženého BVS. Opět je znatelně rychlejší SG strom než tyto implementace, ovšem i tak je znatelně pomalejší než ostatní typy, které udržují strom vyvážený. BB-α strom je ze shluku nejúspěšnějších typů nejpomalejší pro většinu sledovaných n, ovšem pro vysoká n je treap o něco pomalejší. I jako v minulém scénáři je struktura treap z to-



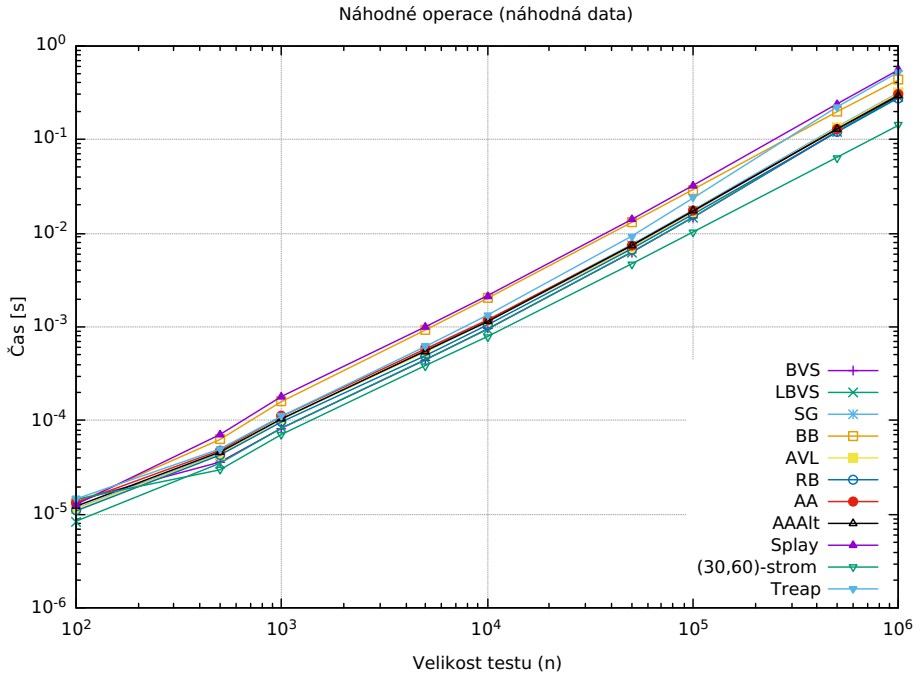
Obrázek 4.8: Test odstraňování náhodných dat ze stromu sestaveného ze vzestupné posloupnosti dat

hoto shluku typů nejpomalejší. Splay strom také nedosahuje tak příznivých výsledků pro $n \leq 5 \cdot 10^5$ jako shluk AVL, RB, obou AA stromů a (30,60)-stromu. Tento shluk má pro většinu n téměř identické výsledky, ovšem pro vysoká n se shluk oddaluje od (30,60)-stromu a jeho výsledky se přibližují splay stromu.

4.1.5 Náhodné operace

Poslední test simuluje práci se stromem, kdy jsou klíče postupně vkládány, odstraňovány a také vyhledávány. Operace se v testu vyskytují přibližně v poměru 4:3:3 a nejvíce zastoupenou operací je operace vkládání, všechny operace v testu mají pravděpodobnost úspěchu 80%. V posledním testu se nejlepšími parametry staly $\alpha = 0{,}29$ pro BB-α strom a parametr $\alpha = 0{,}35$ pro SG strom. Nejlepším (a,b)-stromem je znovu (30,60)-strom. Výsledky jsou znázorněny v grafu 4.9.

Pro $n = 100$ je (30,60)-strom nejhorší, pro vyšší n už je ovšem nejlepším typem a pro $n = 10^6$ je rozdíl mezi (30,60)-stromem a typu s druhým nejlepším výsledkem vyšší než rozdíl mezi výsledky typu s nejhorším výsledkem a typu s druhým nejlepším výsledkem. Nejhorším typem pro $n \geq 500$ je splay strom, který je o něco málo pomalejší než BB-α strom, pro velmi vysoká n má podobné výsledky i struktura treap, což je opět způsobeno operací odstra-



Obrázek 4.9: Test odstraňování náhodných dat ze stromu sestaveného ze vzestupné posloupnosti dat

ňování. Ostatní nejmenované typy, tedy nevyvážené BVS, oba AA, RB a SG stromy, mají téměř identické výsledky.

4.2 Porovnání se stávajícími implementacemi

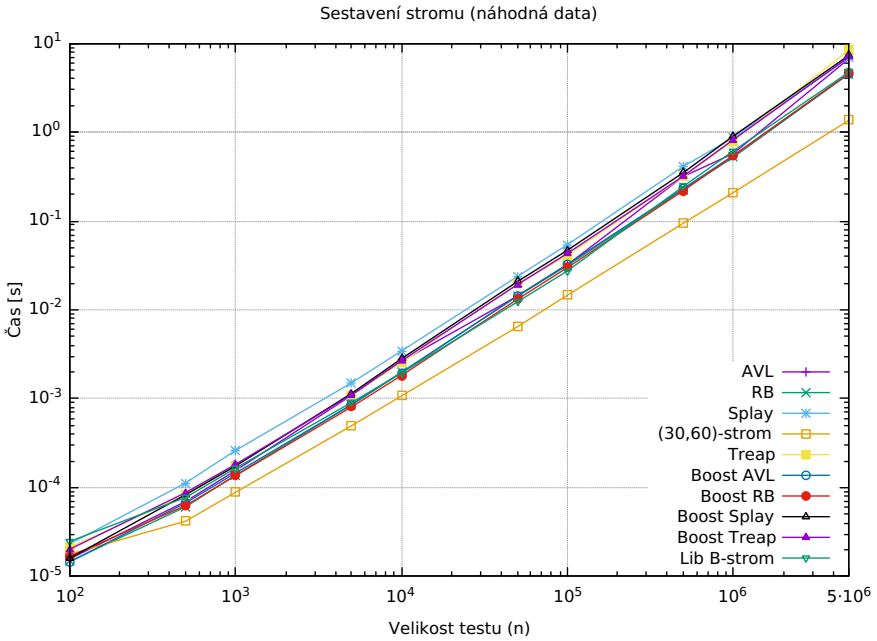
Stejné testy, jejichž výsledky jsou popsány a znázorněny výše, byly provedeny i pro knihovní implementace jazyku C++, jednalo se o implementace stromů AVL, červeno-černého, splay a struktury treap z knihovny `Boost.Intrusive` a B-strom z knihovny `cpp-btree`. Výsledky a parametry implementovaných typů jsou totožné s výsledky z minulé části. Z grafů jsou odstraněny implementace, které nejsou porovnány, a přidány knihovní implementace. Je-li porovnáván (30,60)-strom s knihovním B-stromem, je parametr B-stromu 64, tudíž jedná o (32,64)-strom, knihovní B-strom je pro tuto hodnotu lépe optimalizován. Je-li použit (5,10)-strom je parametr B-stromu 10, aby se i v této implementaci jednalo o (5,10)-strom.

4.2.1 Operace vkládání

Prvním testem, jehož výsledky budou porovnávány s knihovními implementacemi je test sestavení stromu.

Náhodná data

Výsledky testu sestavení stromu náhodnými daty pro porovnání s knihovními implementacemi jsou znázorněny v grafu 4.10.



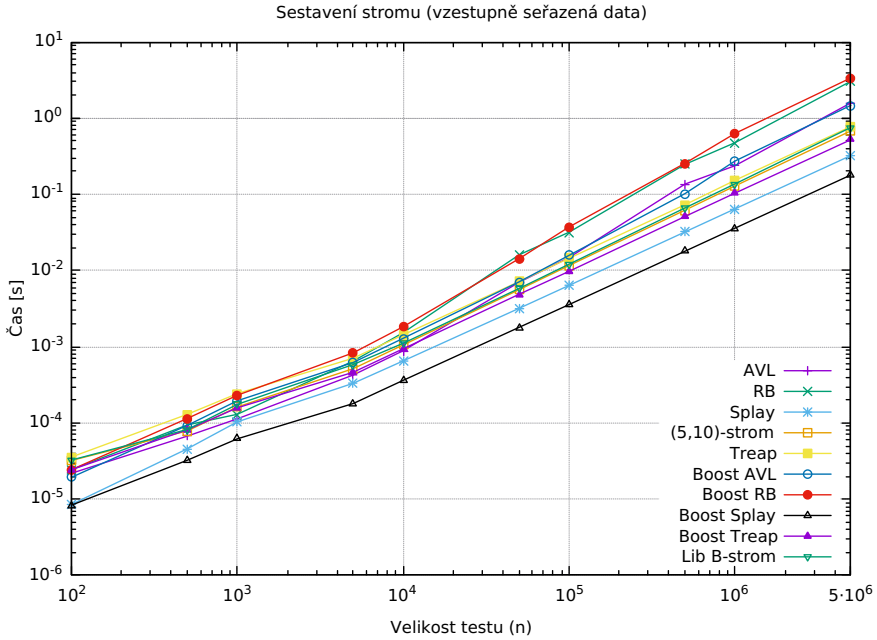
Obrázek 4.10: Test vkládání náhodných dat do stromu s knihovními implementacemi

K významným rozdílům v grafu dochází pouze pro vytvořenou implementaci (a,b)-stromu a knihovního B-stromu a implementace splay stromu. (30,60) strom je v testu nejlepší, zatímco B-strom dosahuje průměrných výsledků. Splay strom implementovaný v rámci práce dosahuje pro nižší n horších výsledků než splay strom importovaný z knihovny, pro vyšší n už jsou rozdíly výsledků velmi nízké.

Vzestupně seřazená data

Výsledky porovnání implementací s knihovními implementacemi pro test sestavení stromu vkládáním seřazené posloupnosti dat jsou zobrazeny v grafu 4.11.

Rozdíly mezi výsledky vytvořených implementací a knihovních implementací jsou nízké, největší rozdíl je mezi implementacemi splay stromu, které jsou nejrychlejší a importovaná implementace dosahuje lepších výsledků. Ostatní implementace mají srovnatelné výsledky s jejich knihovními implementacemi a pro některá n mají dokonce lepší výsledky. Ovšem knihovní implementace dosahují převážně lepších výsledků.



Obrázek 4.11: Test vkládání vzestupně seřazených dat do stromu s knihovními implementacemi

4.2.2 Řazení vkládáním do stromu

Druhým testem, pro než budou implementace porovnávány, je test simulující algoritmus `tree sort`.

Náhodná data

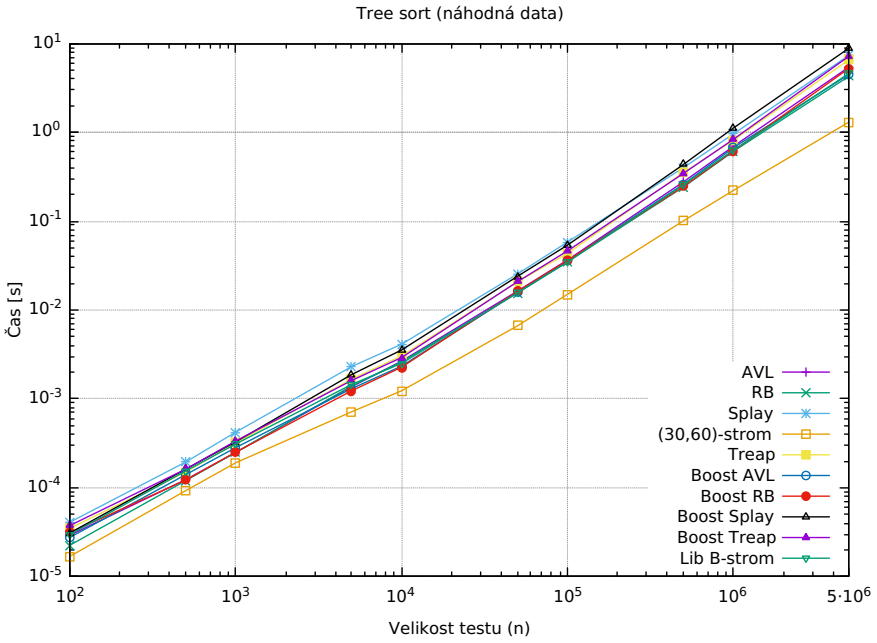
Znázornění výsledků testu, který simuluje algoritmus `tree sort`, pro náhodnou posloupnost dat se nachází v grafu 4.12.

S výjimkou (30,60)-stromu, který dosahuje v testu jednoznačně nejlepších výsledků, jsou výsledky podobné. Ostatní vytvořené implementace si vedou v testu lépe nebo alespoň téměř totožně s knihovními implementacemi. Vytvořené implementace mají tedy, vzhledem k rozdílu vůči testu vkládání, rychlejší nebo porovnatelné získávání seřazených klíčů ze stromu.

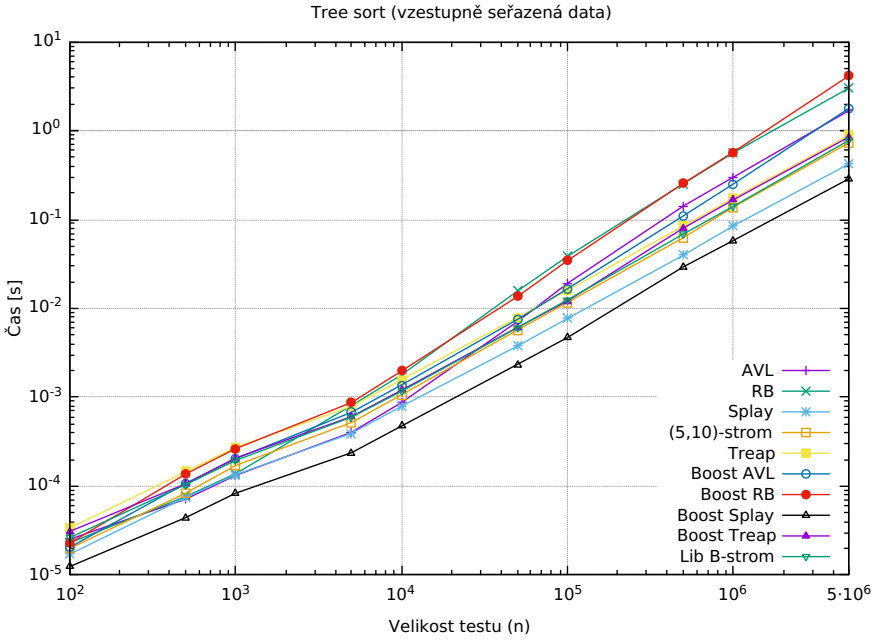
Vzestupně seřazená data

Pro znázornění výsledků scénáře, který simuluje `tree sort` a v němž jsou do sledovaných stromů vkládány vzestupně seřazená data, viz graf 4.13.

Výsledky jsou podobné výsledkům vkládání seřazených dat, ovšem v tomto testu se rozdíly mezi vytvořenými a stávajícími implementacemi zlepšují ve prospěch vytvořených implementací, což podporuje tvrzení z minulého scénáře, že získání seřazených klíčů je o něco rychlejší ve vytvořených implementacích.



Obrázek 4.12: Test řazení vkládáním náhodných dat do stromu s knihovními implementacemi



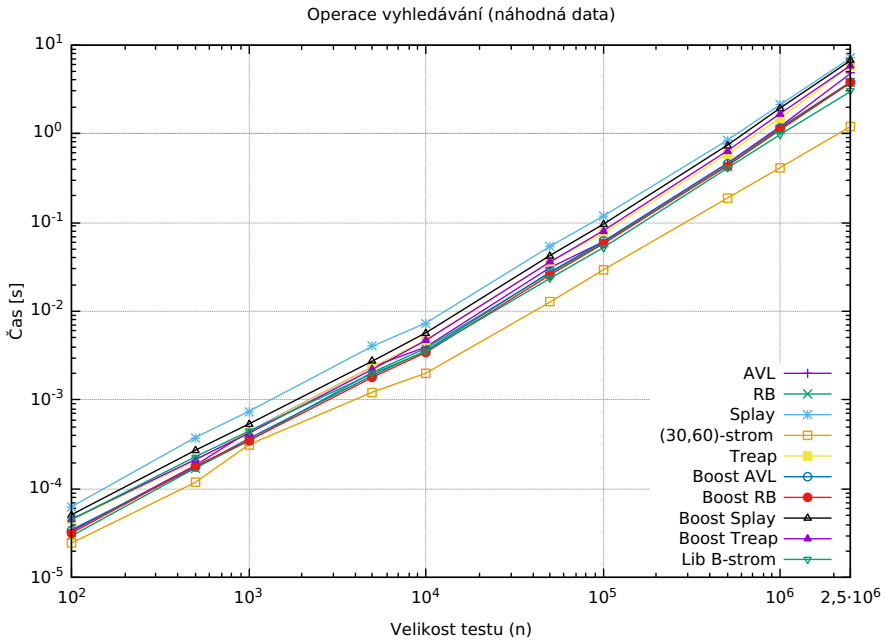
Obrázek 4.13: Test řazení vkládáním vzestupně seřazených dat do stromu s knihovními implementacemi

4.2.3 Operace vyhledávání

Třetím testem, v kterém budou implementace porovnávány, je test operace vyhledávání.

Náhodná data

Výsledky testu, kde je strom sestaven z náhodných dat a v němž se následně vyhledává, jsou znázorněny v grafu 4.14.



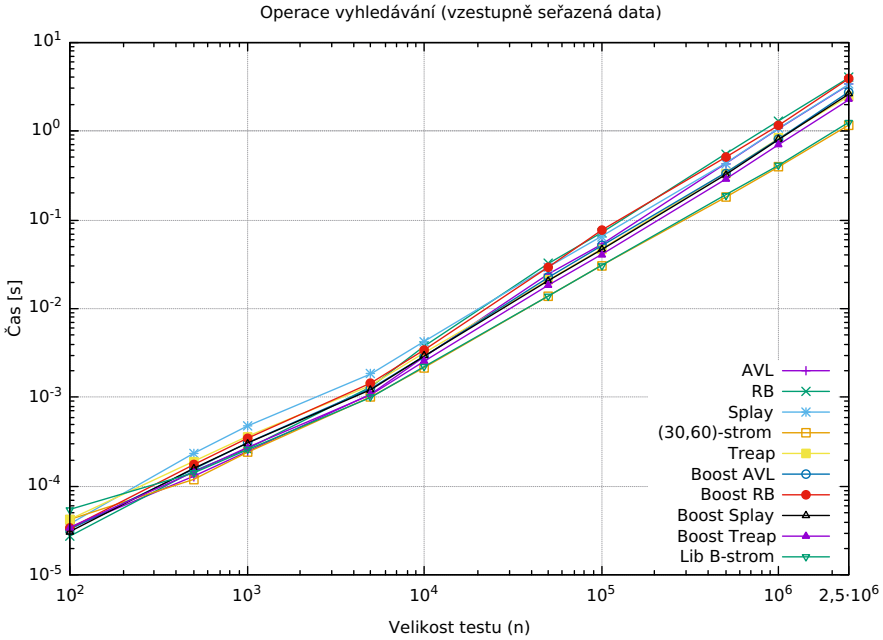
Obrázek 4.14: Test vyhledávání náhodných dat ve stromu sestaveném z náhodné posloupnosti dat s knihovními implementacemi

Výrazně nejlepším stromem v testu je (30,60)-strom. Ostatní stromy nabývají velmi podobných výsledků a z vytvořených typů je výsledkově znatelně horší pouze splay, ostatní typy mají podobné a v některých případech i o něco lepší výsledky.

Vzestupně seřazená data

Znázornění výsledků scénáře, v kterém je strom sestaven z vzestupně seřazených dat a v němž se následně vyhledává, se nachází v grafu 4.15.

Výsledky vytvořených implementací jsou velmi podobné výsledkům knihovních implementací. K výraznějším rozdílům dochází v případech splay stromu a struktury treap, jejichž implementace dosahují horších výsledků než jejich knihovní verze.



Obrázek 4.15: Test vyhledávání náhodných dat ve stromu sestaveném ze vzestupné posloupnosti dat s knihovními implementacemi

4.2.4 Operace odstraňování

Předposledním testem, v kterém budou rozebrány rozdíly vytvořených a knihovních implementací, je test odstraňování.

Náhodná data

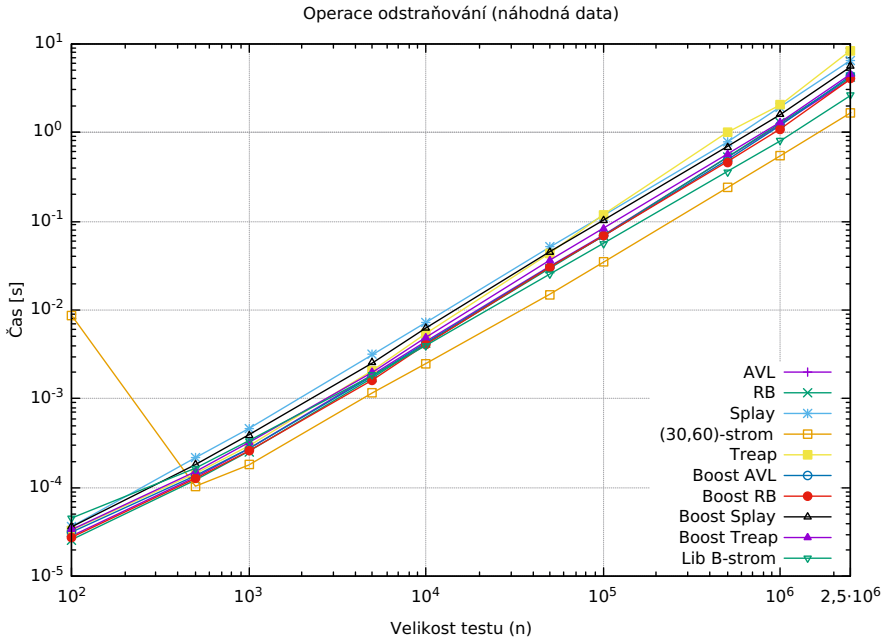
První posloupností v testu odstraňování je posloupnost náhodných dat, výsledky testu pro tato data se nachází v grafu 4.16.

(30,60)-strom je v testu pro $n \geq 500$ znatelně nejlepší. Rozdíly ostatních typů mezi vytvořenými a knihovními implementacemi jsou minimální s výjimkou struktury treap, jejíž knihovní implementace dosahuje lepších výsledků.

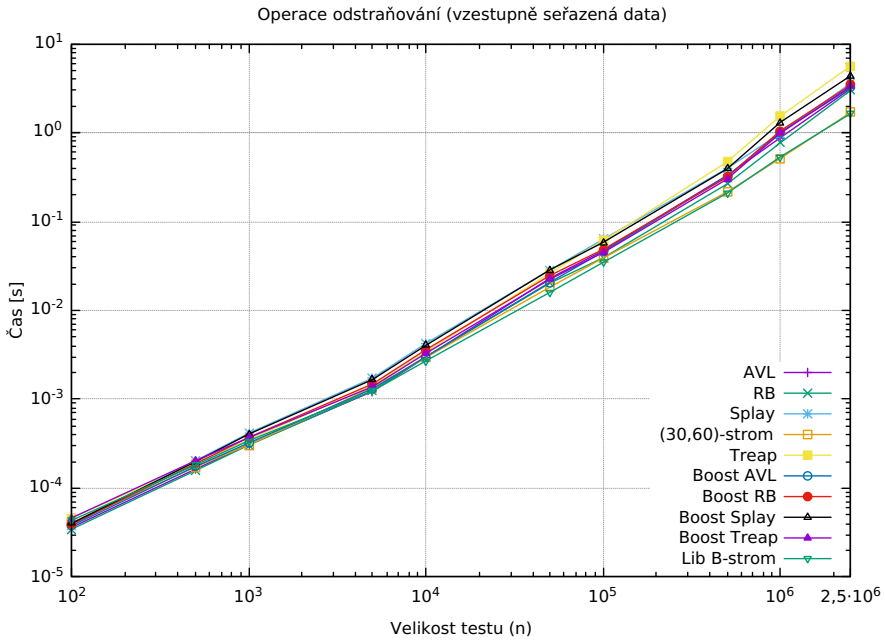
Vzestupně seřazená data

Druhou posloupností v testu odstraňování je posloupnost vzestupně seřazených dat, výsledky testu pro tato data se nachází v grafu 4.17.

Výsledky implementací (a,b)-stromu jsou v tomto testu o něco lepší než implementace BVS, ovšem rozdíl mezi nimi je velmi nízký a patrný až pro vysoká n. Rozdíl mezi vytvořenými a knihovními implementacemi je také minimální s výjimkou vytvořené implementace struktury treap, jejíž výsledky v testu jsou nejhorší. Ovšem u ostatních implementací BVS si vedou vytvořené im-



Obrázek 4.16: Test odstraňování náhodných dat ze stromu sestaveného z náhodné posloupnosti dat s knihovními implementacemi

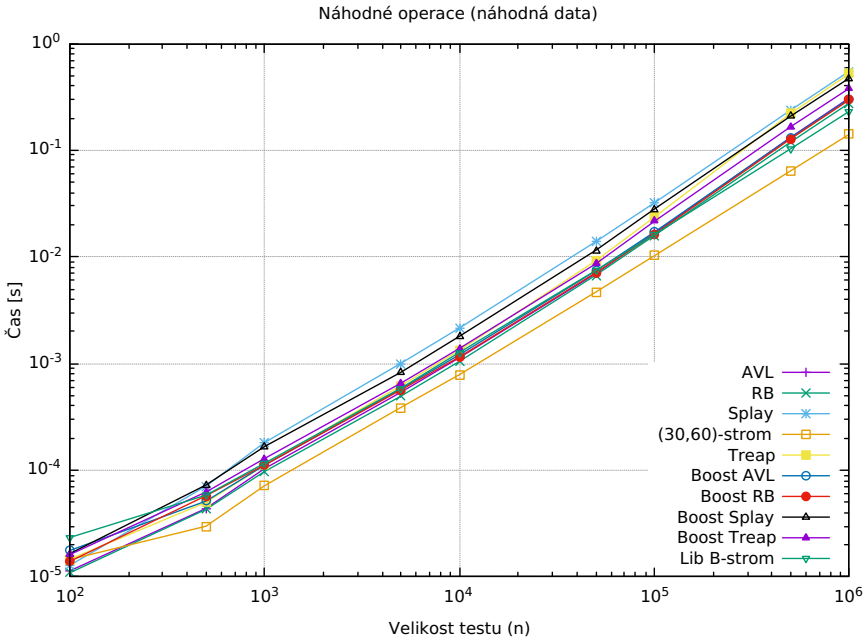


Obrázek 4.17: Test odstraňování náhodných dat ze stromu sestaveného ze vzestupné posloupnosti dat s knihovními implementacemi

plementace o něco lépe než knihovní implementace, rozdíl je s výjimkou splay stromu však minimální.

4.2.5 Náhodné operace

Poslední test, v němž dochází k porovnání implementací, je test s náhodnými operacemi a náhodnými daty. Výsledky testu jsou zobrazeny v grafu 4.18.



Obrázek 4.18: Test odstraňování náhodných dat ze stromu sestaveného ze vzestupné posloupnosti dat s knihovními implementacemi

V testu je pro většinu n opět nejlepší vytvořená implementace (30,60)-stromu. Výsledky implementací červeno-černého stromu a AVL stromu jsou téměř totožné. Ovšem lepších výsledků struktury treap a splay stromu dosahují knihovní implementace.

4.2.6 Souhrn výsledků porovnání

Ve většině testů si vedl nejlépe (a,b)-strom, který v mnoha případech dosáhl lepších výsledků než knihovní B-strom, toho bylo ovšem docíleno neefektivním využíváním paměti a (a,b)-strom si alokoval rovnou celé bloky podle maximálního počtu podstromů. Výsledky také ukazují, že pro zajištění dobrých výsledků je vhodné použít vyvážené vyhledávací stromy, jenž v žádném testu nejsou znatelně horší než nevyvážené, i když třeba jednoduché typy jako je struktura treap.

Splay strom se také neukázal jako vhodný typ pro uvedené testy s výjimkou testů, v nichž dochází k jeho nejlepší výkonnosti. Splay stromy jsou ovšem vhodné pro jiné využití, než je popsáno v této kapitole, což je například, jak již bylo v sekci věnující se splay stromům uvedeno, pro IP adresy a cache paměti.

Závěr

Cílem práce bylo popsání a implementace následujících typů vyhledávacích stromů: AVL, červeno-černý, AA, splay, treap, (a,b), BB-α, scapegoat a nevyvážený BVS, připravení testovacích dat a porovnání výsledků stromů pro tato data.

V práci jsou všechny uvedené typy, jejich operace a vlastnosti popsány. V práci jsou také uvedeny jejich časové složitosti. Vzhledem k náročnosti některých důkazů nejsou důkazy uvedeny přímo v textu, ale v textu je uvedena alespoň odkaz na tyto důkazy.

Implementovány byly všechny vybrané typy, nevyvážené BVS a AA stromy byly implementovány dvěma způsoby. Zajímavou z těchto implementací je především alternativní implementace AA stromu, která k vyvažování stromu využívá barvu vrcholů, nikoliv jejich úroveň, jak je ve většině zdrojů uvedeno.

V rámci práce byl vytvořen generátor dat pro různé scénáře, které mohou při práci s vyhledávacími stromy nastat. Vygenerovaná data byla použita ke změření výkonností jednotlivých implementací a porovnání jejich výsledků. Kapitola porovnání pak nabízí náhled na možnosti a výhody využití různých typů pro různé potřeby. Kromě porovnání jednotlivých typů mezi sebou dochází také k porovnání s implementacemi vybraných typů v knihovnách jazyku C++, výstupy těchto porovnání jsou pro vytvořené implementace pozitivní. Rozdíly mezi výsledky jsou až na několik výjimek minimální a v několika případech dosahují vytvořené implementace lepších výsledků.

Na práci lze navázat rozšířením výstupů práce o další, v práci neuvedené, typy vyhledávacích stromů, vytvořením dalších scénářů pro porovnání, optimalizováním některých operací, například operace odstraňování struktury treap. Dalším zajímavým rozšířením by mohlo být rozšíření téma o možnosti paralelní práce s vyhledávacími stromy a jejich operacemi.

Bibliografie

1. SEDGEWICK, Robert; WAYNE, Kevin D. *Algorithms*. 4th. Addison-Wesley, 2011. ISBN 9780321573513;
2. MAREŠ, Martin; VALLA, Tomáš. *Průvodce labyrintem algoritmů*. 1. vydání. Praha: CZ.NIC, z.s.p.o, 2017. ISBN 9788088168195.
3. GRAHAM, Ronald L.; KNUTH, Donald E.; PATASHNIK, Oren. *Concrete Mathematics: a foundation for computer science*. 2nd. Reading: Addison Wesley, 1994. ISBN 9780201558029;
4. TARJAN; ENDRE, Robert. Amortized Computational Complexity. *SIAM Journal on Algebraic Discrete Methods*. 1985, roč. 6, č. 2, s. 306–318. Dostupné z DOI: 10.1137/0606031.
5. SKIENA, Steven S. *The Algorithm Design Manual*. Data Structures. Cham: Springer International Publishing, 2020. ISBN 978-3-030-54256-6. Dostupné z DOI: 10.1007/978-3-030-54256-6_3.
6. LEISERSON, Charles E.; CORMEN, Thomas H.; RIVEST, Ronald L.; STEIN, Clifford. *Introduction to algorithms*. 3rd. Cambridge, Mass: The MIT Press, 2009. ISBN 9780262533058.
7. KNUTH, Donald E. *The Art of Computer Programming, Volume 3: (2nd Ed.) Sorting and Searching*. USA: Addison Wesley Longman Publishing Co., Inc., 1998. ISBN 0201896850.
8. GOODRICH, Michael T.; TAMASSIA, Roberto; GOLDWASSER, Michael H. *Data Structures and Algorithms in Java, 6th Edition*. New York: Wiley, 2014. ISBN 9781118771334.
9. REED, Bruce. The Height of a Random Binary Search Tree. *J. ACM*. 2003, roč. 50, č. 3, s. 306–332. ISSN 0004-5411. Dostupné z DOI: 10.1145/765568.765571.

10. G.M.ADEL'SON-VEL'SKII; E.M.LANDIS. An algorithm for organization of information. *Dokl. Akad. Nauk SSSR*. 1962, roč. 146, č. 2, s. 263–266. ISSN 0002-3264.

11. BRASS, Peter. *Advanced Data Structures*. Cambridge University Press, 2008. Dostupné z DOI: 10.1017/CBO9780511800191.

12. GUIBAS, Leo J.; SEDGEWICK, Robert. A dichromatic framework for balanced trees. In: *19th Annual Symposium on Foundations of Computer Science (sfcs 1978)*. 1978, s. 8–21. Dostupné z DOI: 10.1109/SFCS.1978.3.

13. BAYER, Rudolf. Symmetric Binary B-Trees: Data Structure and Maintenance Algorithms. *Acta Inf.* 1972, roč. 1, č. 4, s. 290–306. ISSN 0001-5903. Dostupné z DOI: 10.1007/BF00289509.

14. ANDERSSON, Arne. Balanced Search Trees Made Simple. In: *Proceedings of the Third Workshop on Algorithms and Data Structures*. Berlin, Heidelberg: Springer-Verlag, 1993, s. 60–71. WADS '93. ISBN 3540571558.

15. HEGER, Dominique A. A Disquisition on The Performance Behaviour of Binary Search Tree Data Structures. *Upgrade*. 2004, roč. V, č. 5. ISSN 1684-5285.

16. SLEATOR, Daniel Dominic; TARJAN, Robert Endre. Self-Adjusting Binary Search Trees. *J. ACM*. 1985, roč. 32, č. 3, s. 652–686. ISSN 0004-5411. Dostupné z DOI: 10.1145/3828.3835.

17. NIEVERGELT, J.; REINGOLD, E. M. Binary Search Trees of Bounded Balance. In: *Proceedings of the Fourth Annual ACM Symposium on Theory of Computing*. Denver, Colorado, USA: Association for Computing Machinery, 1972, s. 137–142. STOC '72. ISBN 9781450374576. Dostupné z DOI: 10.1145/800152.804906.

18. BLUM, Norbert; MEHLHORN, Kurt. *On the average number of rebalancing operations in weight-balanced trees*. 1978. Dostupné z DOI: http://dx.doi.org/10.22028/D291-26073.

19. ANDERSSON, Arne. Improving partial rebuilding by using simple balance criteria. In: DEHNE, F.; SACK, J. -R.; SANTORO, N. (ed.). *Algorithms and Data Structures*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1989, s. 393–402. ISBN 978-3-540-48237-6.

20. GALPERIN, Igal; RIVEST, Ronald L. Scapegoat Trees. In: *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*. Austin, Texas, USA: Society for Industrial a Applied Mathematics, 1993, s. 165–174. SODA '93. ISBN 0898713137.

21. ARAGON, C.R.; SEIDEL, R.G. Randomized search trees. In: *30th Annual Symposium on Foundations of Computer Science*. 1989, s. 540–545. Dostupné z DOI: 10.1109/SFCS.1989.63531.

22. BAYER, R.; MCCREIGHT, E. Organization and Maintenance of Large Ordered Indices. In: *Proceedings of the 1970 ACM SIGFIDET (Now SIGMOD) Workshop on Data Description, Access and Control*. Houston, Texas: Association for Computing Machinery, 1970, s. 107–141. SIGFIDET '70. ISBN 9781450379410. Dostupné z DOI: 10.1145/1734663.1734671.

23. GAZTANAGA, Ion; KRZIKALLA, Olaf. *Boost.Intrusive* [knihovna] [https://www.boost.org/doc/libs/1_79_0/doc/html/intrusive.html]. 2015. Verze 1_79_0.

24. BARTH, Lukas; HÜBSCHLE-SCHNEIDER, Lorenz; ROMANO, Robert C. *Ygg* [knihovna] [https://tinloaf.github.io/ygg/]. 2020. Verze 0.2.

25. MACDONALD, Joshua. *cpp-btree* [knihovna] [https://code.google.com/archive/p/cpp-btree/]. 2013. Verze 1.0.1.

26. WILLIAMS, Thomas; KELLEY, Colin et al. *Gnuplot* [software] [http://www.gnuplot.info]. 2022. Verze 5.4.3.

Příloha **A**

Seznam použitých zkratek

AA Vyvážený binární vyhledávací strom pojmenovaný podle jeho objevitele Arne Anderssona.

AVL Vyvážený binární vyhledávací strom pojmenovaný podle jeho objevitelů Aděĺson-Veĺskiiho a Landise.

BB-α α-váhově vyvážený binární vyhledávací strom.

BVS Binární vyhledávací strom.

BVS Implementace binárního vyhledávacího stromu jednosměrným spojovým seznamem.

RB Červeno-černý strom.

SG Scapegoat strom.

Příloha **B**

Obsah přiloženého média

- `readme.txt` ..... stručný popis obsahu média
- `thesis` ..... zdrojové soubory práce ve formátu LaTeX a použité obrázky
- `text` ..... text práce
- `impl` ..... adresář s implementační částí práce
  - `Makefile` Makefile pro kompilaci programu a generování dokumentace
  - `src` ..... zdrojové kódy implementace
  - `data.zip` ..... vstupní data, jejichž výsledky jsou v práci diskutovány
  - `doc` ..... vygenerovaná dokumentace
  - `config.cfg` ..... soubor sloužící ke generaci dokumentace
  - `results` ..... výsledky testů
    - `graphs` ..... vygenerované grafy
    - `test_results` ..... textové výsledky testů
    - `data` ..... výsledky testů ve formátu csv