SLOVENSKÁ TECHNICKÁ UNIVERZITA
Fakulta informatiky a informačních technologií
Ilkovičova 2, 842 16 Bratislava 4

Zadanie 2 - Vyhľadávanie v dynamických množinách

Dávid Penťa ID: 110871
Datové štruktúry a algoritmy
FIIT STU 2020/2021
Cvičenie: Štvrtok 9:00
7.4.2021

Dávid Penťa
ID: 110871

Zadanie 2 – Vyhľadávanie v dynamických množinách

Existuje veľké množstvo algoritmov, určených na efektívne vyhľadávanie prvkov v dynamických množinách: binárne vyhľadávacie stromy, viaceré prístupy k ich vyvažovaniu, hašovanie a viaceré prístupy k riešeniu kolízií. Rôzne algoritmy sú vhodné pre rôzne situácie podľa charakteru spracovaných údajov, distribúcii hodnôt, vykonávaným operáciám, a pod. V tomto zadaní máte za úlohu implementovať a porovnať tieto prístupy.

Vašou úlohou v rámci tohto zadania je porovnať viacero implementácií dátových štruktúr z hľadiska efektivity operácií **insert** a **search** v rozličných situáciách (operáciu **delete** nemusíte implementovať):

• (2 body) Vlastnú implementáciu binárneho vyhľadávacieho stromu (BVS) s ľubovoľným algoritmom na vyvažovanie, napr. AVL, Červeno-Čierne stromy, (2,3) stromy, (2,3,4) stromy, Splay stromy, ...
• (1 bod) Prevzatú (nie vlastnú!) implementáciu BVS s iným algoritmom na vyvažovanie ako v predchádzajúcom bode. Zdroj musí byť uvedený.
• (2 bod) Vlastnú implementáciu hašovania s riešením kolízií podľa vlastného výberu. Treba implementovať aj operáciu zväčšenia hašovacej tabuľky.
• (1 bod) Prevzatú (nie vlastnú!) implementáciu hašovania s riešením kolízií iným spôsobom ako v predchádzajúcom bode. Zdroj musí byť uvedený.

Za implementácie podľa vyššie uvedených bodov môžete získať 6 bodov. Každú implementáciu odovzdáte v jednom samostatnom zdrojovom súbore (v prípade, že chcete odovzdať všetky štyri, tak odovzdáte ich v štyroch súboroch). Vo vlastných implementáciách nie je možné prevziať cudzí zdrojový kód. **Pre úspešné odovzdanie musíte zrealizovať aspoň dve z vyššie uvedených implementácií** – môžete teda napr. len prevziať existujúce (spolu 2 body), alebo môžete aj prevziať existujúce aj implementovať vlastné (spolu 6 bodov). Správnosť overte testovaním-porovnaním s inými implementáciami.

V technickej dokumentácii je vašou úlohou zdokumentovať vlastné aj prevzaté implementácie a uviesť podrobné scenáre testovania, na základe ktorých ste zistili, v akých situáciách sú ktoré z týchto implementácií efektívnejšie**. Vyžaduje to tiež odovzdanie programu**, ktorý slúži na testovanie a odmeranie efektívnosti týchto implementácií ako jedného samostatného zdrojového súboru (obsahuje funkciu **main**). **Bez testovacieho programu, a teda bez úspešného porovnania aspoň dvoch implementácií bude riešenie hodnotené 0 bodmi**. Za dokumentáciu, testovanie a dosiahnuté výsledky (identifikované vhodné situácie) môžete získať najviac 4 body. Hodnotí sa kvalita spracovania.

Môžete celkovo získať 10 bodov, **minimálna požiadavka sú 4 body**.

Riešenie zadania sa odovzdáva do miesta odovzdania v AIS do stanoveného termínu (oneskorené odovzdanie je prípustné len vo vážnych prípadoch, ako napr. choroba, o možnosti odovzdať zadanie oneskorene rozhodne cvičiaci, príp. aj o bodovej penalizácii). Odovzdáva sa jeden **zip** archív, ktorý obsahuje jednotlivé zdrojové súbory s implementáciami a jeden súbor s dokumentáciou vo formáte **pdf**.

Dávid Penťa
ID: 110871

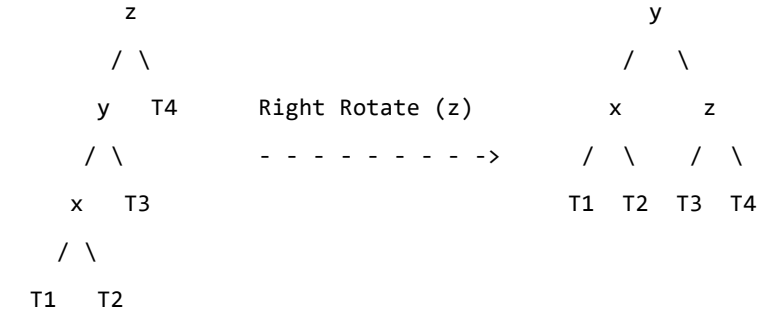
Vlastná implementácia binárneho vyhľadávacieho stromu

Použitá metóda – AVL strom

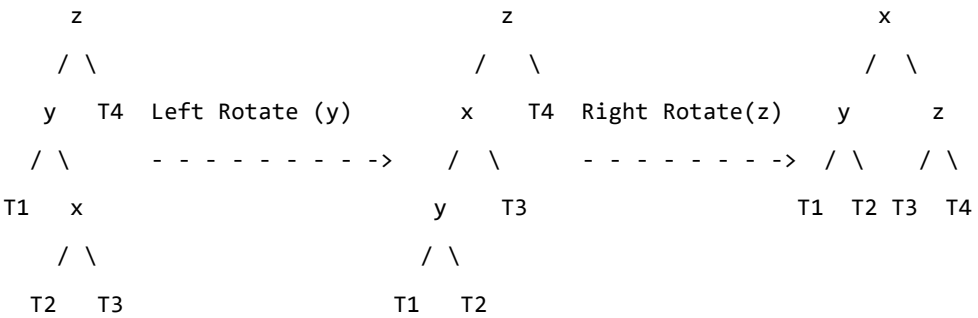
Insert()

Najprv sa do binárneho stromu vloží prvok a skontroluje sa, či je strom vyvážený. Ak nie je, tak nastal jeden zo štyroch prípadov: Left Left Case, Left Right Case, Right Right Case, Right Left Case, a strom sa vyváži rotáciou doprava, doľava alebo ich kombináciou.

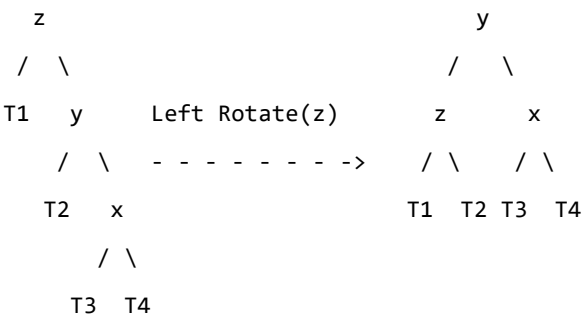
Left Left Case



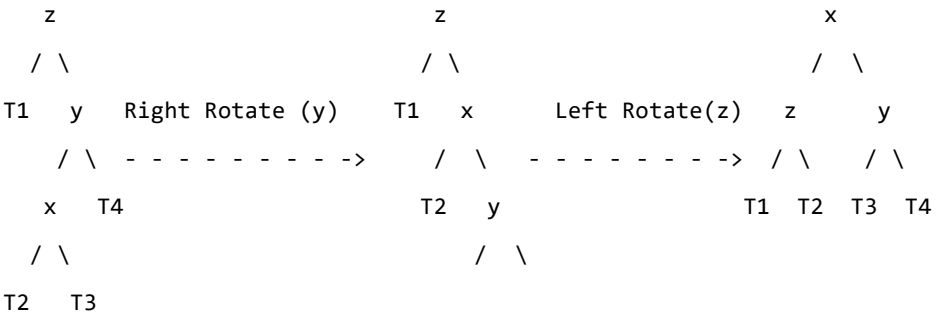
Left Right Case



Right Right Case



Right Left Case



Časová zložitosť

Časová zložitosť jednej rotácie je O(1) a vykonať sa môže maximálne toľko rotácii, aká je výška stromu, čo je $\log_2 N$. Preto zložitosť insert pri jednom vkladaní je maximálne $O(\log_2 n)$.

Search()

Funkcia rekurzívne prehľadáva binárny strom a hľadá zadanú hodnotu. Ak ju nájde, vráti hodnotu 1, ak nenájde vráti hodnotu 0.

Časová zložitosť

Funkcia prechádza maximálne $\log_2 n$ vrcholmi, kde n = počet vrcholov, a vyhodnotenie jedného vrcholu trvá O(1). Preto celková časová zložitosť je $O(\log_2 n)$.

Pamäťová zložitosť

Priestorová zložitosť AVL stromu je O(n), pretože do pamäte ukladá len vrcholy a na nič iné nie je pamäť využívaná.

Dávid Penťa
ID: 110871

Prevzatá implementácia binárneho vyhľadávacieho stromu

Použitá metóda - Červeno-Čierny strom

Zdroj: https://gist.github.com/VictorGarritano/5f894be162d39e9bdd5c

Insert()

Funkcia vloží nový vrchol do binárneho stromu a potom skontroluje, či sú splnené pravidlá červeno-čierneho stromu. Ak nie, tak ho rotáciami doprava a doľava napraví.

Pravidlá červeno-čierneho stromu:

1. Každý vrchol je buď červený alebo čierny.
2. Koreň je čierny.
3. Listy sú čierne.
4. Každý červený vrchol má dvoch čiernych synov.
5. Každá cesta z jedného vrcholu do jeho podriadených listov, obsahuje rovnaký počet čiernych vrcholov.

Časová zložitosť

Keďže rovnako ako v AVL strome je maximálna výška stromu $\log_2 n$, priemerná aj maximálna časová zložitosť je $O(\log_2 n)$.

Search()

Funkcia funguje na rovnakom princípe ako v AVL strome.

Časová zložitosť

Funkcia prechádza maximálne $\log_2 n$ vrcholmi, kde n = počet vrcholov, a vyhodnotenie jedného vrcholu trvá O(1). Preto celková časová zložitosť je $O(\log_2 n)$.

Pamäťová zložitosť

Priestorová zložitosť červeno-čierneho stromu je O(n), pretože do pamäte ukladá len vrcholy a na nič iné nie je pamäť využívaná.

Vlastná implementácia hašovania

Použitá metóda - riešenie kolízií otvoreným adresovaním

Insert()

Ak je vedierko do ktorého sa má prvok uložiť už obsadené, tak sa prvok uloží do ďalšieho voľného vedierka. Je možné, že sa prvok do tabuľky nebude môcť umiestniť vôbec.

Časová zložitosť

Čas potrebný pre výpočet hašu je O(1). Pri dostatočne veľkej tabuľke, čo je približne dvakrát väčšia ako počet vstupov, je časová zložitosť O(1). Maximálna možná je O(n), ak by sa prvok mal uložiť do prvého vedierka, ale jediné voľné by bolo posledné vedierko.

Search()

Funkcia nazrie do vedierka, do ktorého sa prvok mal zahašovať. Ak je v nej uložení iný prvok, tak funkcia bude pokračovať v prehľadávaní vedierok, dokiaľ hľadaný prvok nájde alebo narazí na prázdne vedierko.

Časová zložitosť

Rovnako ako pri insert(), pri dostatočne veľkej tabuľke, je časová zložitosť O(1) a maximálna možná je O(n). Najhorší prípad nastane, ak by sa hľadaný prvok mal uložiť do prvého vedierka, ale jediné voľné by bolo posledné vedierko, a program prejde celú tabuľku aby ho našiel.

Pamäťová zložitosť

Veľkosť tabuľky n je na začiatku vyhradená používateľom a program neprekročí jej veľkosť, takže O(n).

Prevzatá implementácia hašovania

Použitá metóda - riešenie kolízií reťazením

Zdroj: https://www.log2base2.com/algorithms/searching/open-hashing.html

Insert()

Vo vedierkach sa vytvárajú spájané zoznamy, do ktorých sa ukladajú všetky prvky s rovnakým hašom.

Časová zložitosť

Priemer je O(1), ale v najhoršom prípade je O(n), čo by nastalo, ak by všetky prvky vytvorili rovnaké haše.

Search()

Funkcia nazrie do vedierka, do ktorého sa prvok mal zahašovať. Bude prehľadávať spájaný zoznam, dokiaľ hľadaný prvok nájde alebo sa zoznam skončí.

Časová zložitosť

Priemer je O(1), ale v najhoršom prípade je O(n), čo by nastalo, ak by všetky prvky vytvorili rovnaké haše.

Pamäťová zložitosť

Program obsadí toľko pamäte, koľko mu príde vstupov k od používateľa, preto O(k).

Dávid Penťa
ID: 110871

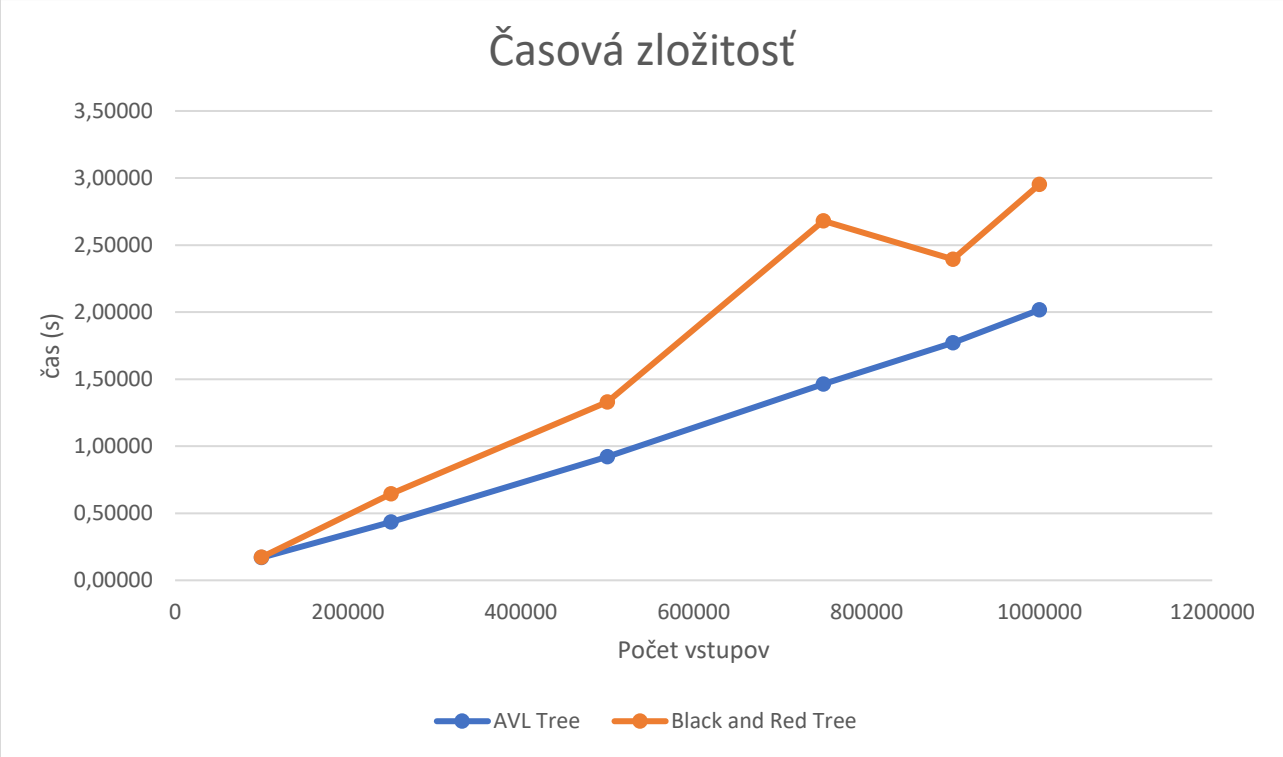
Testovanie

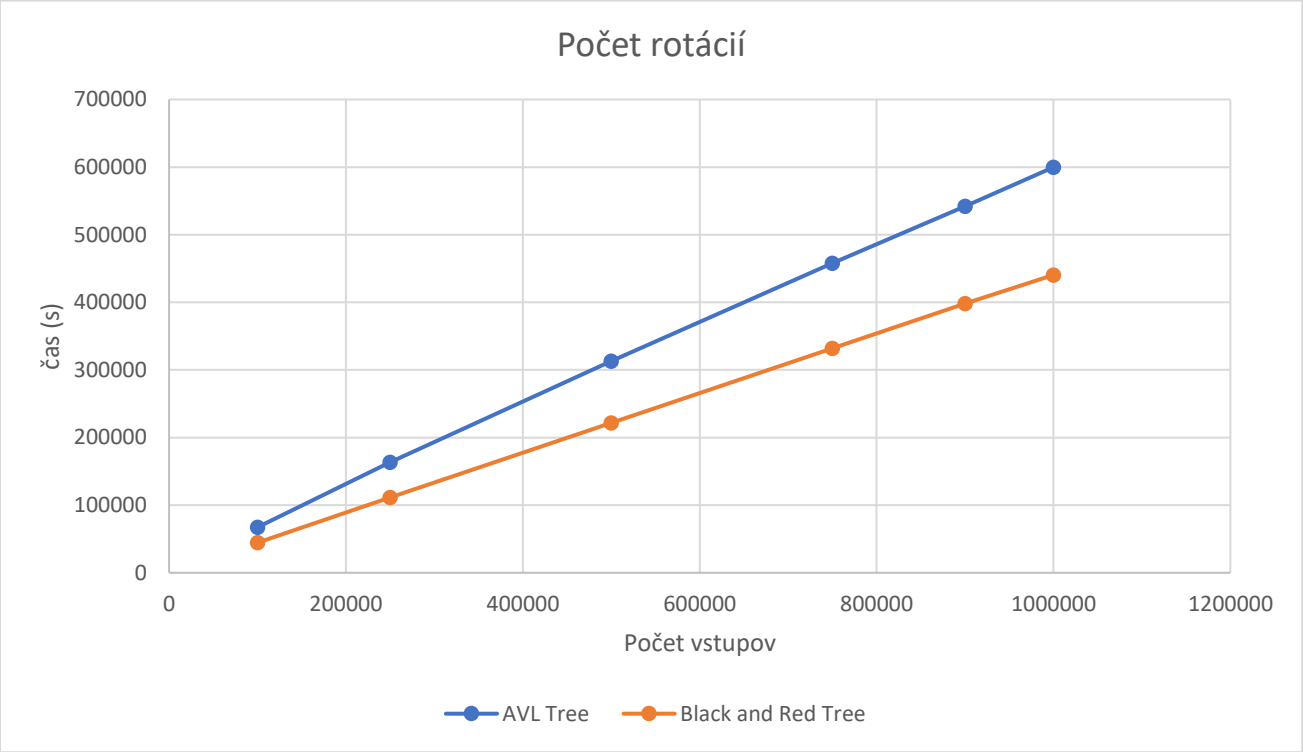
Na testovanie som vytvoril dva testovacie textové súbory. Jeden na testovanie stromov, ktorý obsahuje milión náhodných čísel, ktoré majú veľkosť od 1000 do 1000000000000. Druhý súbor obsahuje milión náhodných reťazcov dĺžky 5 až 14, obsahujúce veľké a malé písmená.

Testovanie binárnych stromov

Pri testovaní binárnych stromov som testoval len funkciu insert(), pretože som vedel, že funkcia search() je v AVL strome rýchlejšia vďaka lepšej vyváženosti stromu. Zaujímalo ma, v ktorom strome, a o koľko je rýchlejšia funkcia insert().

Zistil som, že prevzatý Červeno-Čierny strom síce urobí menej rotácii ako môj AVL strom a tak by mal byť teoreticky rýchlejší, no testovanie ukazuje, že v tejto konkrétnej implementácii je pomalší.

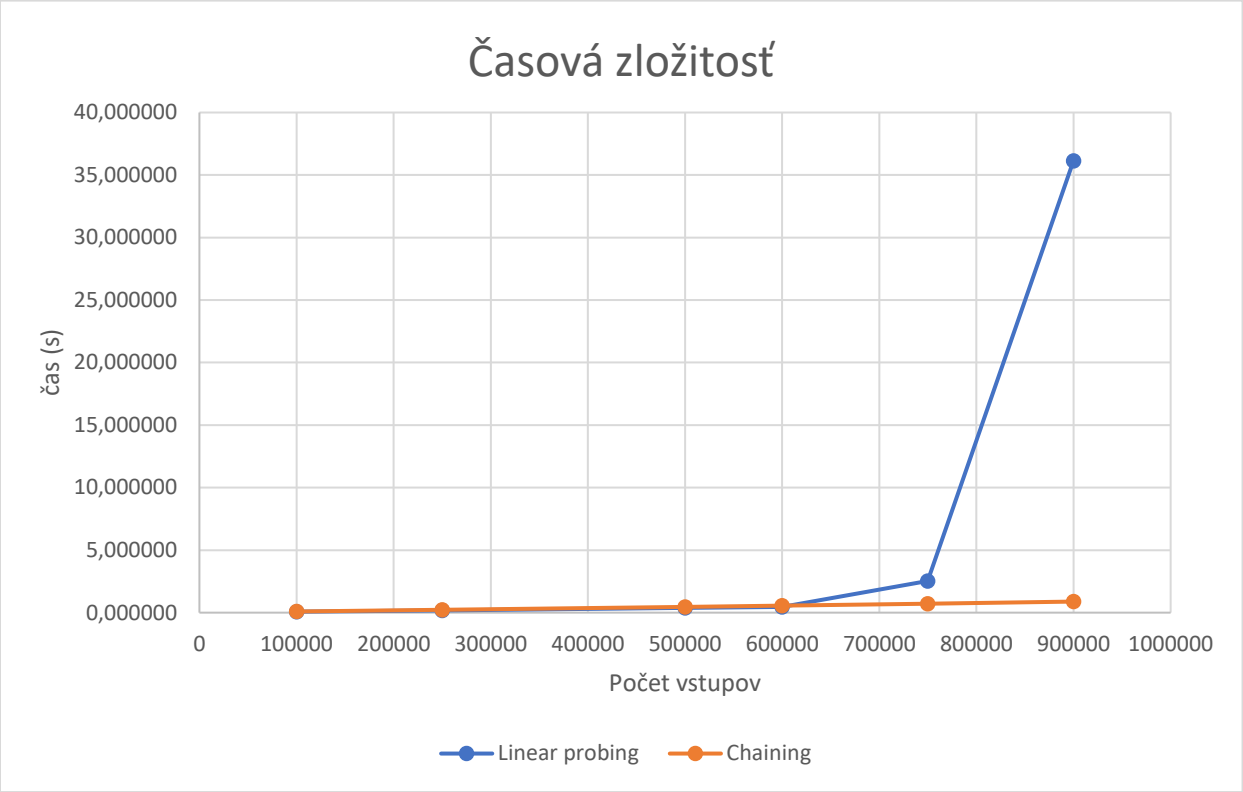


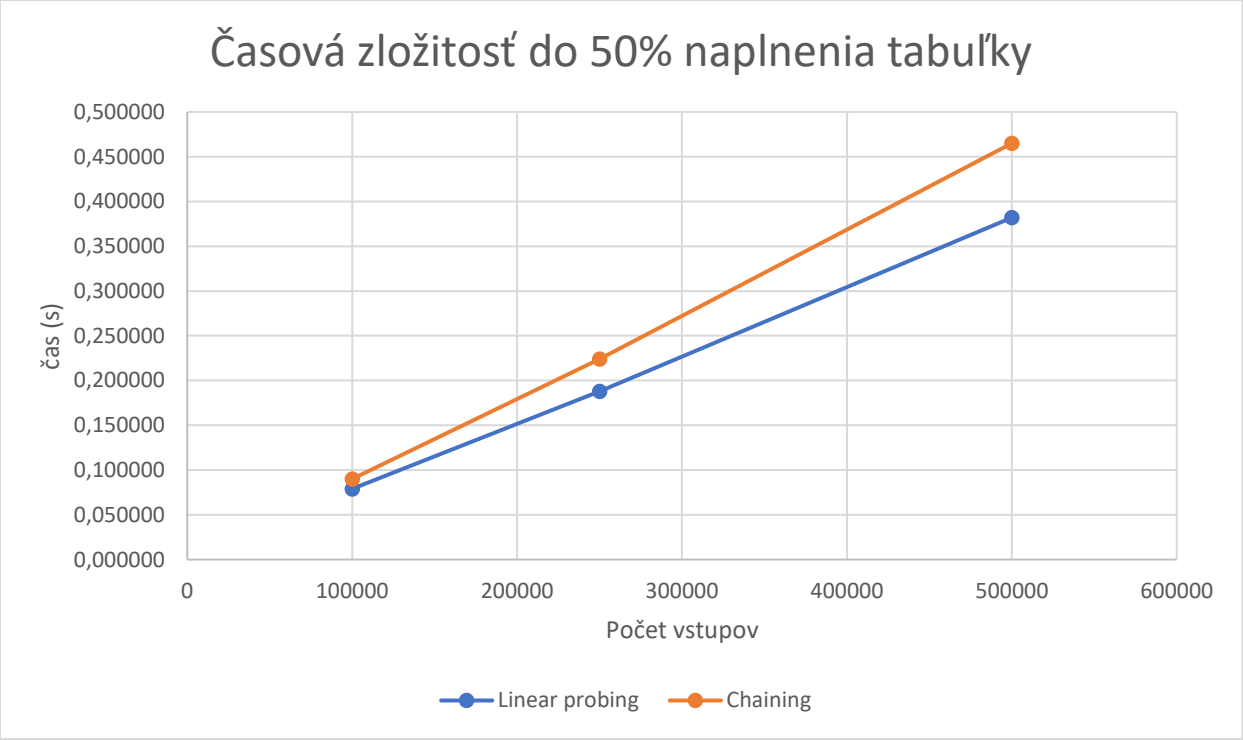


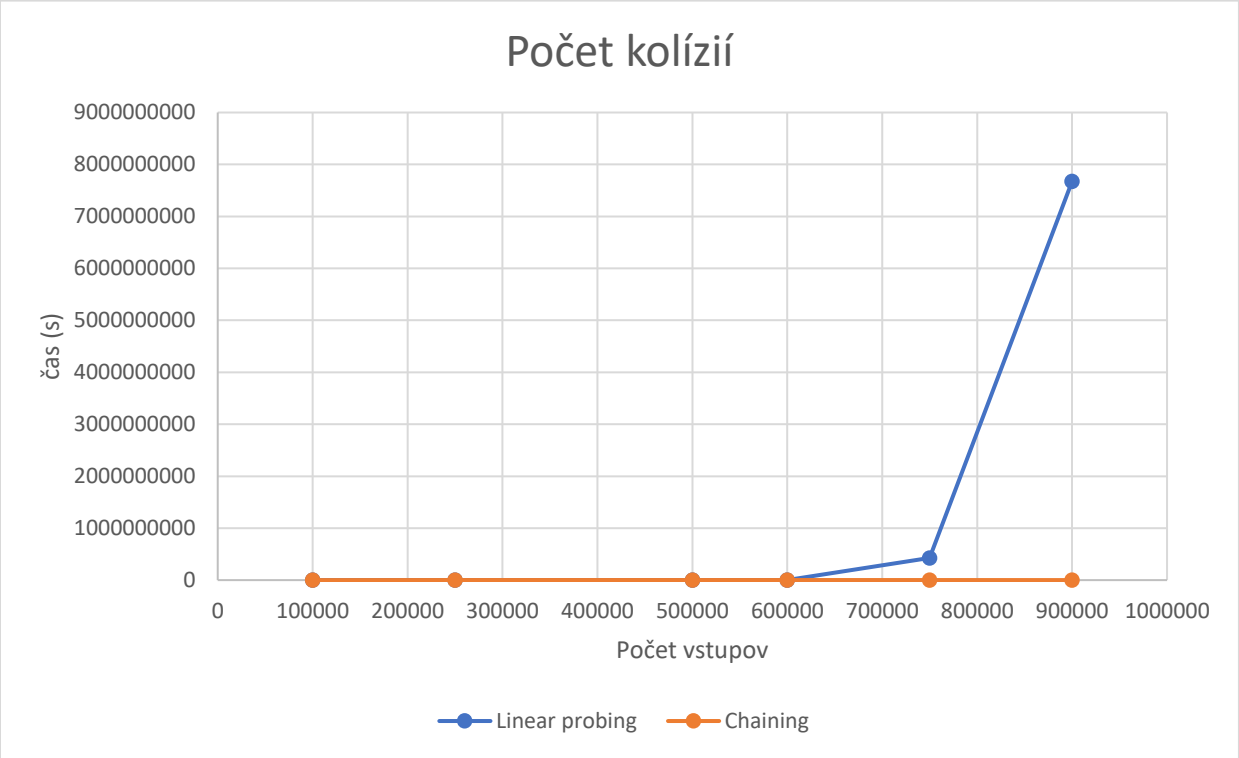
Testovanie hašovacích tabuliek

Pri testovaní hašovacích tabuliek som použil rovnakú veľkosť tabuľky 999983, a aj rovnaký spôsob hašovania, pretože som sa sústredil čisto len na porovnanie vkladania a vyhľadávania v týchto dvoch spôsoboch riešenia kolízií.

Výsledky testovania ukazujú, že do približne 50% naplnenia hašovaciej tabuľky, bola trochu rýchlejšia tabuľka s otvoreným adresovaním. Avšak po 50% naplnenia hašovaciej tabuľky, bol v tabuľke s otvoreným adresovaním obrovský počet kolízií, čo malo aj veľký vplyv na jeho časovú zložitosť.


Časová zložitosť
40,000000
35,000000
30,000000
25,000000
20,000000
15,000000
10,000000
5,000000
0,000000
čas (s)
0
100000
200000
300000
400000
500000
600000
700000
800000
900000
1000000
Počet vstupov
Linear probing
Chaining


Časová zložitosť do 50% naplnenia tabuľky
0,500000
0,450000
0,400000
0,350000
0,300000
0,250000
0,200000
0,150000
0,100000
0,050000
0,000000
čas (s)
0
100000
200000
300000
400000
500000
600000
Počet vstupov
Linear probing
Chaining



Záver

Binárne stromy

V AVL strome síce dochádzalo častejšie k rotáciám pri vkladaní, ale vďaka tomu že tieto rotácie sú rýchlejšie implementované ako v prevzatom červeno-čiernom strome, tak aj celkový čas vykonania AVL stromu je lepší.

Hašovacie tabuľky

Ak program používa rozumný princíp hašovania a vie približný počet vstupov, podľa ktorého nastaví veľkosť hašovaciej tabuľky na aspoň dvakrát väčšiu, tak je rozdiel v časovej zložitosti skoro zanedbateľný. Akonáhle sa však tento dvojnásobný rozdiel prekročí, bude počet kolízií v hašovaciej tabuľke s otvoreným adresovaním radikálne rásť a tým aj časová zložitosť. Keď hašovaciu tabuľku nastavíme na aspoň dvakrát väčšiu kapacitu ako je počet vstupov, tak rozhodujúcim faktorom voľby riešenia kolízii by mali byť hlavne ich iné odlišnosti ako časová zložitosť.