FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ

SLOVENSKÁ TECHNICKÁ UNIVERZITA

Ilkovičova 2, 842 16 Bratislava 4

2021/2022

Dátové štruktúry a algoritmy

Zadanie č.1

Cvičiaci: Mgr. Peter Lehocký

Čas cvičení: Štvrtok 13:00-15:40

Vypracoval Dávid Varinský

AIS ID: 116323

Úvod:

V tomto zadaní porovnávame a testujeme 4 dátové štruktúry ktorými sú 2 binárne vyhľadávacie stromy AVL strom a Splay strom a 2 hašovacie tabuľky. Testovanie prebiehalo nastavením si dát pre koľko prvkov by sme chceli testovať. O ostatné sa nám postaral náhodný generátor Stringov vďaka ktorému som si do poľa stringov uložil náhodné stringy ktoré boli následne použité pri testovaní pridávania, hľadania a mazania. Stringy mali dĺžku 100 znakov aby sa čo najviac predišlo kolíziám ktoré aj napriek tomu nastali. Testoval som postupne AVL strom Splay strom hašovaciu tabuľku ktorá riešila kolízie reťazením a ako poslednú som testoval hašovaciu tabuľku v ktorej boli kolízie riešené pomocou otvoreného adresovania.

AVL strom

AVL Insert	Test1	Test2	Test3	Test4	Test 5	Test 6	Priemerný čas
20M	73	50,7	127,1	87,6	64,78	59,48	77,11
10M	40	26,9	44,8	48,2	35,24	32,6	37,96
5M	5,4	4,2	6,8	6,58	5,9	3,15	5,34
1M	2	1,4	3,7	2,94	3,1	1,3	2,41
AVL search							
20M	23,4	20,39	33,2	32	18,19	30	26,20
10M	9	8,1	11,8	14	13,8	9,3	11,00
5M	1,5	1,4	1,4	2,5	1,5	1,2	1,58
1M	0,6	0,6	0,84	1	1,1	0,6	0,79
AVL delete							
20M	56,7	75,8	83,6	105,3	91,98	68,25	80,27
10M	27	43,6	45,9	47,1	45,5	25,6	39,12
5M	3,4	7,5	5,5	6,1	5,2	3,46	5,19
1M	1,28	3,1	2,5	2,7	2,9	1,28	2,29

Tabuľka č. 1 AVL strom

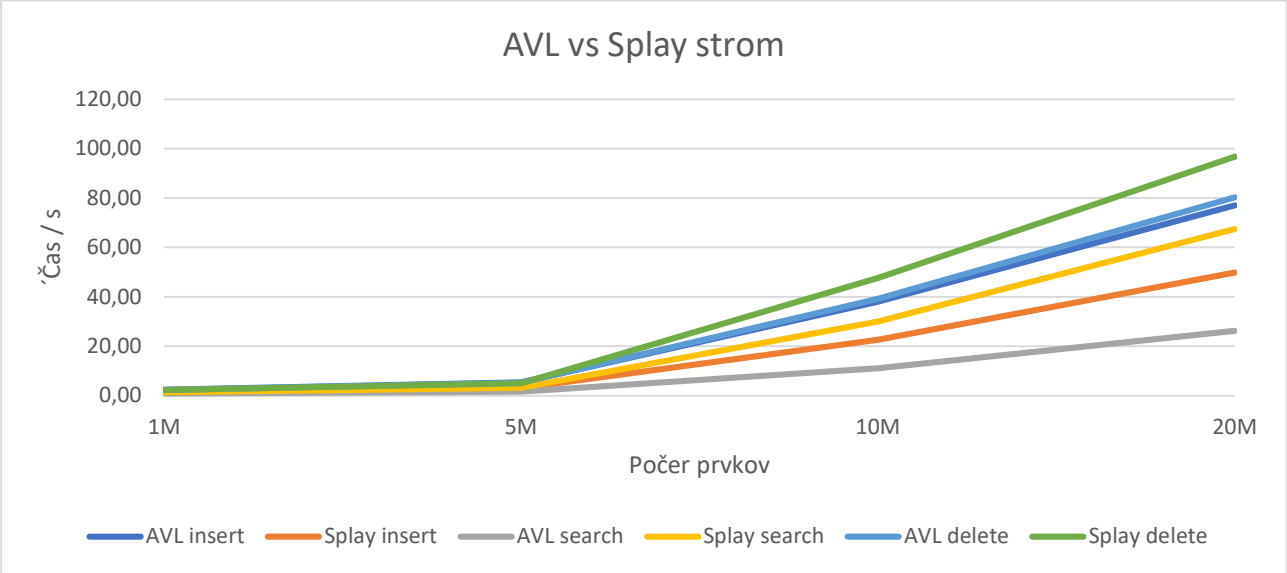
AVL strom som testoval každú funkciu 6krát pre 1M, 5M, 10M a 20M meranie prebiehalo nastavení časovača spusteného pred for cyklom Následne sme sa vnorili do cyklu v ktorom sme mali už vopred vygenerované pole stringov a pomocou funkcie sme ho premenili na int, následne daný int bol vložený do stromu ako index pod ktorým ho nájdeme v strome. Na konci for cyklu sa nám vyhodnotí čas ktorý daný cyklus trval a vypíše sa nám do konzoly. Všimol som si že pri viacerých spusteniach daného programu mi vždy vyjdú iné hodnoty, najväčší rozdiel čo som zaznamenal pri insertovani do AVL a to pri 20M prvkoch bolo 50,7sec (2.riadok, 3.stlpec) a 127,1sec (2.riadok 4.stlpec) čo je rozdiel 74,1sec. Takisto pri viac opakovaniach a spriemerovaniu času si vieme všimnúť že Insert a Delete majú podobnú časovú komplexnosť zatiaľ čo search je 2 až 4 násobne rýchlejší pretože sa nevykonávajú ďalšie rotácie.

Splay strom

Splay Insert	Test1	Test2	Test3	Test4	Test 5	Test 6	Priemerný čas
20M	49,1	72,22	47,8	45,7	44	40,5	49,89
10M	24	26	27,6	18,2	19,6	20,4	22,63
5M	2,9	4,7	2,55	2,3	2,9	2,4	2,96
1M	1,1	1,62	1,3	1	1	1	1,17
Splay search							
20M	69,25	82,5	53,6	83,6	60	56	67,49
10M	34,1	31,6	24	30	32	27,8	29,92
5M	3,5	3	3	3	3,4	3	3,15
1M	1,4	1,3	1,3	1,3	1,4	1,3	1,33
Splay delete							
20M	98,7	127,78	99,6	93,4	80,9	80	96,73
10M	51,9	63,1	60,8	38,3	35	36,8	47,65
5M	5,3	7	6,9	3,3	3,8	3,7	5,00
1M	2,24	2,8	4,1	1,5	1,6	1,6	2,31

Tabuľka č.2 Splay strom

Splay som testoval tým istým spôsobom ako AVL aby boli pri porovnaní rovnocenné. S nami nameraných hodnôt po spriemerovaní Splay strom mal najväčšiu časovú komplexnosť pri funkcii delete (96s) a ale na druhú stranu insert(49s) bol pri 20M dvojnásobne rýchlejší. Časová náročnosť funkcie search v tomto strome, bola medzi insert a delete.



Graf 1 AVL a Splay strom

Ako si môžeme všimnúť na grafe (Graf 1) najviac času bolo potrebné ako aj v AVL taj aj v Splay stromoch na funkciu delete. AVL search je najrýchlejší v našich binárnych vyhľadávacích stromoch, naopak Splay search je takmer 3 krát časovo náročnejší čo vieme odôvodniť tým že to nie je perfektne vyvážený strom ale sám sa nám otáča a sprístupňuje posledné použité prvky na vrch pretože majú najväčšiu pravdepodobnosť že bude k ním potrebné znovu pristúpiť a keďže my pri search prechádzame cez všetky prvky v zozname tak je to v našom podaní neefektívne.

Hašovacia tabuľka s otvoreným adresovaním

Insert							
20M	6,6	6,9	5,8	5,9	5,6	5,7	6,08
10M	4,15	3,1	2,5	2,36	2,9	4,8	3,30
5M	2	1,4	1	1,1	1,3	1,4	1,37
1M	0,42	0,15	0,17	0,25	0,16	0,26	0,24
search							
20M	5,4	4,26	4,23	3,92	3,8	5,5	4,52
10M	3,9	2,3	2,2	2,3	3	4	2,95
5M	1,47	1,4	1	1	1,2	1,4	1,25
1M	0,28	0,17	0,16	0,2	0,17	0,25	0,21
delete							
20M	9,9	6,6	7	6,7	6,8	8,3	7,55
10M	5	3,3	2,8	2,9	3	4,5	3,58
5M	1,6	1,6	1,5	1,5	1,1	1,9	1,53
1M	0,27	0,2	0,18	0,22	0,17	0,26	0,22

Tabuľka č. 3 Hašovacia tabuľka s riešením kolízii pomocou otvoreného adresovania

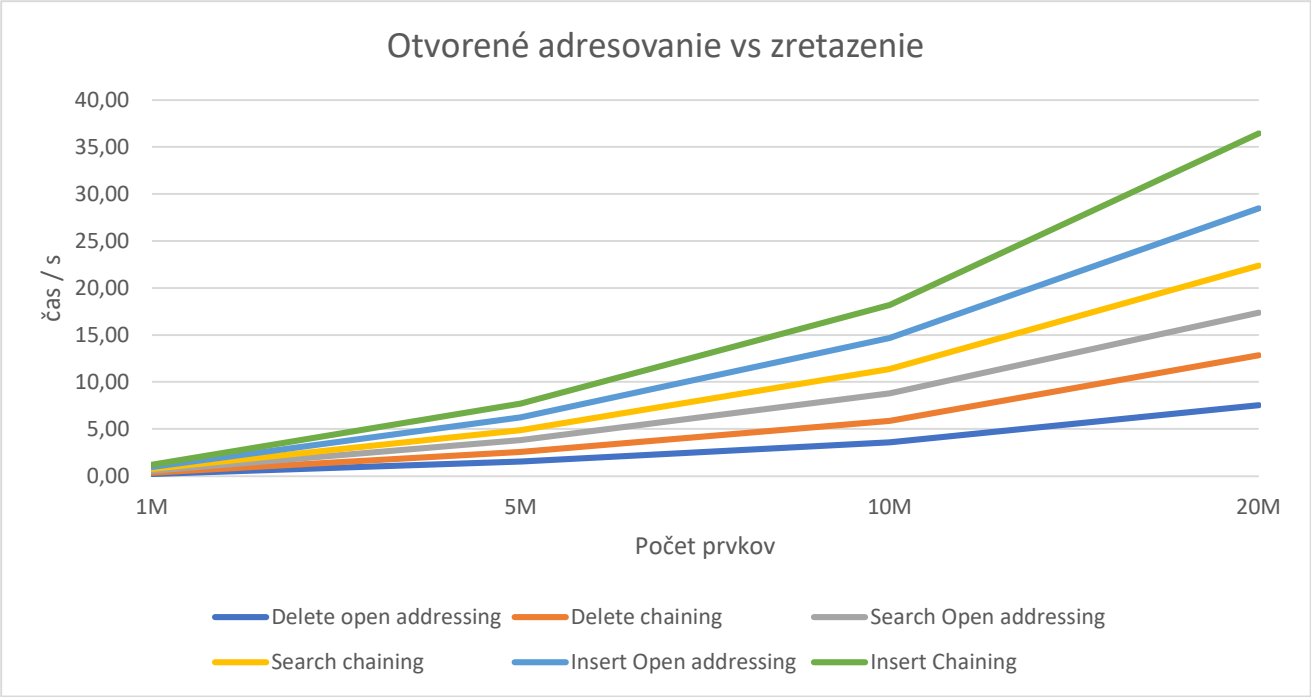
Pri používaní Hašovacej tabuľky ktorá rieši kolízie pomocou otvoreného adresovania (Tabuľka č.3) si môžeme všimnúť že najrýchlejšie sa vykoná funkcia search ktorá pri 20M prvkov v priemere trvala 4,52s no naopak najväčšiu časovú komplexnosť má delete ktorý v priemere trval 7,5s pri 20M prvkov.

Hašovacia tabuľka zreťazená

Insert							
20M	9,24	9,3	9,5	9,4	8,2	2,1	7,96
10M	2,9	3,3	4,3	3,7	4	3	3,53
5M	1,3	1,6	1,7	1,5	1,5	1,3	1,48
1M	0,16	0,28	0,25	0,24	0,2	0,18	0,22
Search							
20M	4,9	5,1	5,6	5,2	4,6	4,7	5,02
10M	3,1	3	2,5	2,2	2,2	2,4	2,57
5M	0,9	1,3	1	1	1	1,1	1,05
1M	0,16	0,2	0,18	0,2	0,2	0,17	0,19
delete							
20M	5,2	5,2	5,7	5,6	5,2	5	5,32
10M	2	2,6	2,3	2,3	2,2	2,2	2,27
5M	0,9	1,1	1	1,1	1,1	1	1,03
1M	0,12	0,17	0,13	0,2	0,14	0,13	0,15

Tabuľka č.4 Hašovacia tabuľka s riešením kolízii pomocou zreťazenia

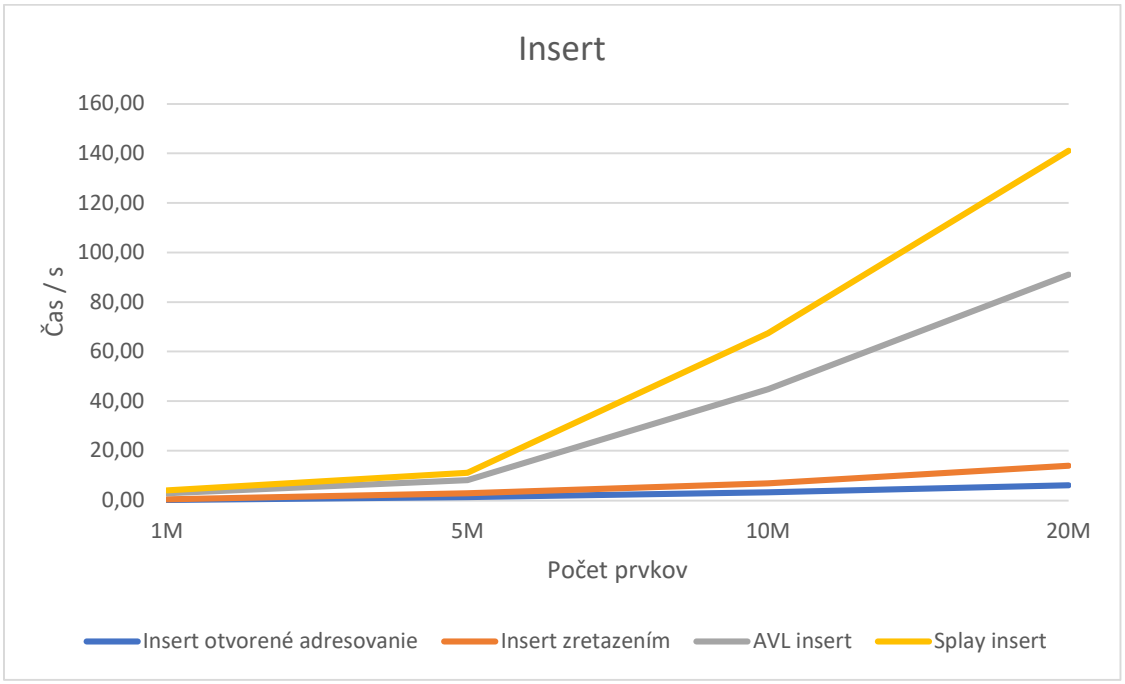
Pri Hašovacej tabuľke ktorá rieši kolízie pomocou zreťazenia si môžeme všimnúť že v mojej implementácii v priemere najdlhšie trval insert čo bolo takmer 8s pri 20M prvkoch a to aj vďaka tomu že bolo potrebné zväčšiť základnú veľkosť tabuľky ktorá bola nastavená ako pri predchádzajúcej Hašovacej tabuľke na 1000 prvkov a bolo potrebné ju zväčšiť cca. na 20000 násobnú veľkosť jej základnej tabuľky, zároveň pri zväčšení tabuľky bolo potrebné všetky prvky znovu vložiť pomocou novej hašovacej funkcie.



Graf 2 Hašovacia tabuľka s otvoreným adresovaním a zreťazením

Graf číslo 2 nám ukazuje časovú náročnosť pri jednotlivých funkciách v hašovacích tabuľkách. Najmenšiu náročnosť pri 20M prvkoch mala funkcia delete pri oboch implementáciách. Pri otvorenom adresovaní bol delete o čosi rýchlejší ako v zreťazenom zozname. Najviac času nám zabral insert do spájaného zoznamu ktorý trval o necelých 10sekúnd viac ako insert do zoznamu s otvoreným adresovaním.

Porovnanie stromov s hašovacími tabuľkami

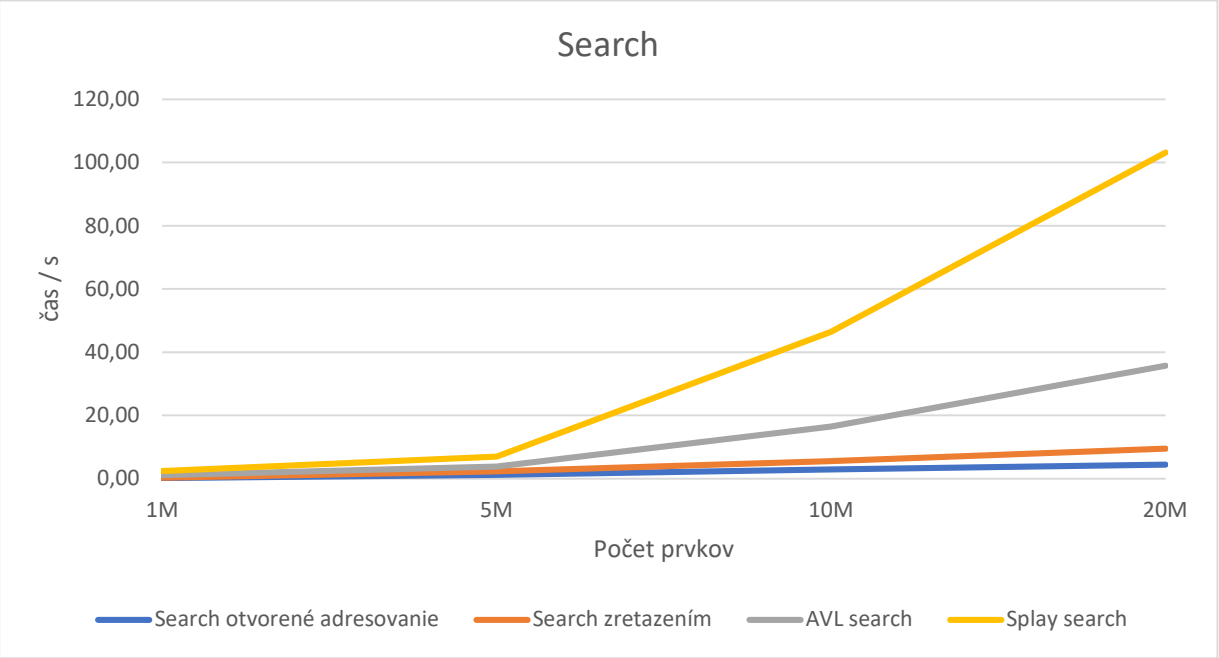


Graf 3 Insert

Pri vkladaní prvkov sme použili pole stringov ktoré sme si na začiatku programu vygenerovali. Na grafe si môžeme všimnúť že najdlhšie trvá insert 20M prvkov pri Splay strome čo vieme odôvodniť

tým že pri každom insert sa daný strom musí otočiť niekedy až niekoľkonásobne čo je dosť časovo náročné. Naopak AVL strom sa nie vždy musí otáčať a ak áno tak nie tak rapídne ako Splay čo mu pridáva na rýchlosti. Pri hašovacích tabuľkách to je o to jednoduchšie že tam nie sú žiadne rotácie ale je tam prehašovanie celej tabuľky, ako si môžeme všimnúť na grafe je to časovo menej náročné ako vkladanie do stromu a jeho rotácie.

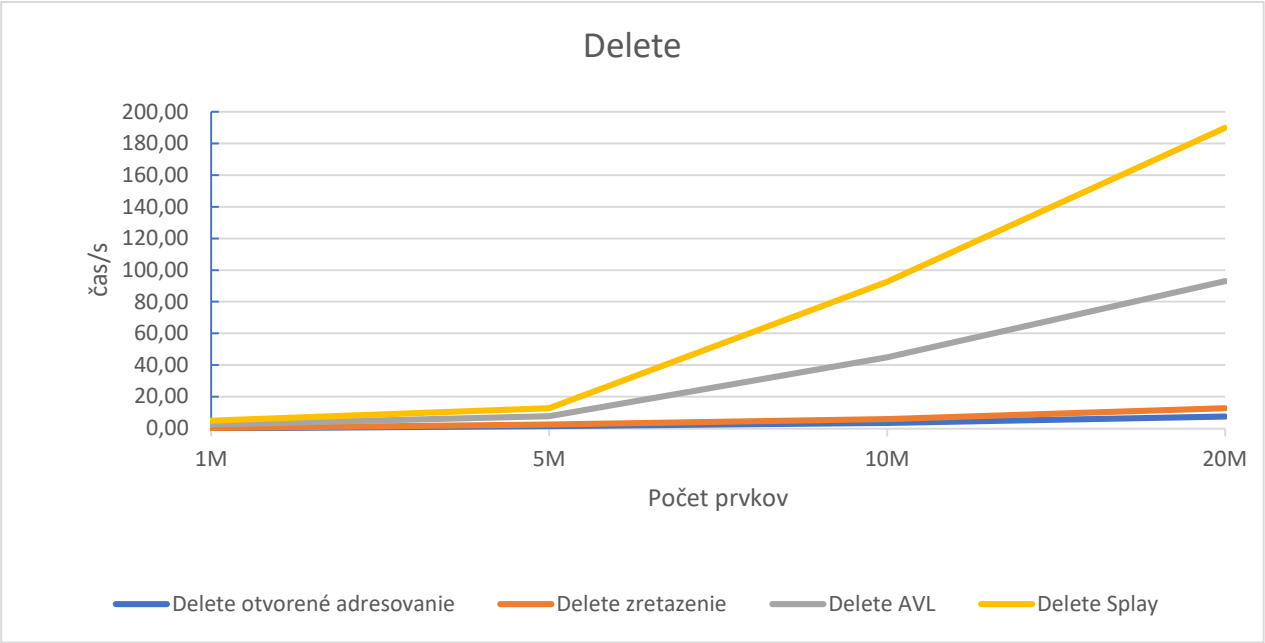
Search



Graf 4 Search

Pri search sme použili znovu to isté pole stringov aby sme zaručili že ku každý string čo bol vložený sa dá aj nájsť a tým otestovali aj funkčnosť insert. Ako pri minulom grafe tak aj pri tomto si môžeme znovu všimnúť že Splay strom ma najväčšiu časovú náročnosť aj pri prehľadávaní a najrýchlejšie z našich implementácií vieme vyhľadávať v hašovacích tabuľkách ktoré riešia kolízie pomocou otvoreného adresovania

Delete



Graf 5 Delete

Delete v tomto zadaní sme riešili odoberaním prvkov zaradom tak ako sme ich vkladali čiže po zbehnutí programu sme mali iba 1 pointer pri stromoch a to na koreň stromu, keďže sme iba nahrádzali ukazovatele pretože Java má garbidge collector ktorý sa nám postaral o uvoľnenie miesta ak už na daný prvok neukazoval pointer. A pri hašovacích tabuľkách s otvoreným adresovaním sme pri delete iba vymazali údaje a booleanovskú hodnotu delete sme nastavili na true, no pri tabuľkách so zreťazením sme si mohli dovoliť odstrániť všetky pointre a nastaviť na null. Ako si vieme všimnúť na grafe č. 5 najrýchlejšie vymazávanie bolo pri hašovacích tabuľkách s otvoreným adresovaním a najpomalšie pri Splay stromoch.

Záver:

Pri tejto implementácii a testovaní boli jednoznačne rýchlejšie hašovacie tabuľky, čo sedí aj s vedomosťami z prednášky, že rýchlosť by mala byť pri hašovacích tabuľkách O(1). Ak by nebolo treba zväčšovať polia a rehashovať celé pole znovu bolo by to možné. Na druhú stranu stromy ktoré mali mat efektivitu O(log n) sa nám nepotvrdilo pretože pri viac prvkoch mám bolo treba niekedy 2 a viac násobne viac času na vykonanie všetkých funkcii čiže to bolo horšie ako O(n).