Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií

Dátové štruktúry a algoritmy
Zadanie 2 – Vyhľadávanie v dynamických množinách

Martin Rudolf

Cvičenie: Pondelok 18.00
Cvičiaci: Ing. František Horvát, PhD. 2020/2021

Martin Rudolf, 97029, ak. rok 2020/2021

Obsah

Opis implementácie projektu

Implementácia Zadania 2 je rozdelené do štyroch zdrojových súborov (file.c) a troch header súborov (file.h). Zdrojový súbor s vlastnoručnou implementáciou hashovej tabuľky je pomenovaný hashTable.c a príslušný header súbor hashTable.h, pre prevzatú implementáciu binárneho vyhľadávacieho stromu to bude bst.c a bst.h. Zdrojový súbor prevzatej hashovej tabuľky je pomenovaný hashTableDev.c s príslušným header súborom hashTableDev.h. Súbor test.c slúži ako testovací súbor ktorý includuje všetky potrebné header súbory. Sú v ňom implementované testovacie funkcie ako aj samotná main funkcia.

Opis algoritmu

Prevzaté implementácie

Implementácia pre binárny vyhľadávací strom bola prevzatá z webu https://www.geeksforgeeks.org/avl-tree-set-1-insertion/ v tejto implementácii bola využitá AVL metóda s vyvažovaním.

Implementácia pre hashovú tabuľku bola prevzatá z webu https://www.journaldev.com/35238/hash-table-in-c-plus-plus táto implementácia, prípady kolízie rieši spájaným zoznamom. Hashova funkcia je rozdielna oproti tej ktorú som použil vo vlastnej implementácií, preto pravdepodobnosť kolízie je v tomto prípade väčšia.

```
unsigned long hash_function(char* str) {
    unsigned long i = 0;
    for (int j = 0; str[j]; j++)
        i += str[j];
    return i % CAPACITY;
}
```

Figure 1 hash funkcia pre prevzatú tabuľku

Vlastná implementácia

Implementácia vlastnej hashovej tabuľky pozostáva z funkcií hash() ktorá slúži na generovanie hashu, vstupným argumentom je reťazec znakov ktorý zahashuje, table_init() na inicializovanie tabuľky, čiže poľa pointerov na štruktúry ktoré nesú atribúty key (daný reťazec) a pointer na ďalšiu štruktúru s rovnakým hashom no rozdielnym kľúčom. O vkladanie a vyhľadávanie sa starajú funkcie insert() a search().

Funkcia insert() dostane na vstupe velkosť tabulky, teda nejaké prvočíslo, ukazovateľ na tabuľku a reťazec znakov. Funkcia najprv pomocou funkcie hash(str) vygeneruje hash a predelí ho veľkosťou tabuľky čo určí index v tabuľke kde bude daná štruktúra umiestnená. Ak je dané miesto voľné vytvorí sa štruktúra s rovnakým kľúčom ako bol reťazec na vstupe a uloží sa, ak nie tak prechádza celý zreťazený zoznam na danom indexe a porovnáva hodnoty kľúčov. Ak sa s nejakým kľúčom zhodne tak sa cyklus ukončí a nič sa neudeje, ak nie a dôjde na koniec tak sa pripne na koniec zoznamu.

Funkcia search() disponuje na vstupe veľkosťou tabuľky, hľadaným reťazcom a ukazovateľom na tabuľku. Na začiatku sa vygeneruje hash s hľadaným reťazcom zo vstupu. Vygenerovaný hash sa predelí veľkosťou tabuľky čo udá miesto v tabuľke na ktorom by sa hľadaný reťazec mohol nachádzať. Funkcia sa najprv pozrie či je na danom mieste nejaká štruktúra, ak nie, tak sa tam nenachádza ani daný reťazec, ak áno tak postupne prechádza zreťazený zoznam na danom indexe a porovnáva key, atribút štruktúry, s hľadaným reťazcom. Ak narazí na zhodu tak vypíše úspech a funkcia sa ukončí.

Vlastne implementovaná hashova tabuľka disponuje aj funkciou resizovania, table_resize(). Tato funkcia vytvorí novú, väčšiu tabuľku a prvky zo starej tabuľky pre usporiada podľa novej veľkosti. Starú následne uvoľní.

Hashovacia funkcia:

```
int hash(char str[]) {
    int len = strlen(str), h = 0;
    for (int i = 0; i < len; i++) {
        h = 31 * h + str[i];
    }
    return abs(h);
}
```

Figure 2 hash funkcia pre vlastnú tabuľku

Spôsob testovania

Vyššie popísané implementácie boli testované na testovacích funkciách, pre vkladanie do prevzatej hash tabuľky test_insert_into_htableDev(), ktorá na základe vstupného argumentu , scenár, naplní tabuľku. Pre scenár 1 naplní tabuľku počtom prvkom podľa vstupného argumentu range. Pre scenár 2 naplní tabuľku obdobne ako v scenári 1 no po naplnení doplní ešte jeden prvok, podobne pracuje aj scenár 3 no ten nakoniec, na miesto jedného prvku, doplní 25% počtu vložených prvkov. Tieto scenáre obsahuje každá testovacia funkcia vkladania. Pre BVS je to test_insert_bst_random() a test_insert_bst_inorder(), random napĺňa strom s náhodnými číslami, inorder za sebou idúcimi. Funkcia

na testovanie vkladania vlastnej hashovej tabuľky je test_insert_into_htableMy().

Funkcie na vyhľadávanie, test_search_in_htableDev(), test_search_in_htableMy(), test_search_bst(). Každa pracuje s dvoma scenármi zo vstupu. Pre scenár 1 vyhľadá jeden náhodný prvok, scenár 2 hľadá náhodných 5% počtu vložených prvkov.

Vkladané dáta do BVS sú celé čísla a pre vkladanie do hashovej tabuľky sú to rovnaké čísla len pretypované na reťazec pomocou funkcie itoa(). Hodnoty sa generujú v testovacích funkciách pri vkladaní.

Dosiahnuté výsledky

Tabuľka 1 test insert 1000 prvkov scenár 1 čas v milisekundách

	My hash table	Prevzatá hash table	BVS inorder	BVS random
1.	0	2	2	2
2.	0	1	2	2
3.	0	1	1	2
4.	1	1	1	2
5.	0	1	1	2

Tabuľka 2test insert 1000 prvkov scenár 2 čas v milisekundách

	My hash table	Prevzatá hash table	BVS inorder	BVS random
1.	0	1	1	2
2.	0	1	2	2
3.	1	1	2	2
4.	0	1	1	2
5.	1	1	1	1

Tabuľka 3 test insert 1000 prvkov scenár 3 čas v milisekundách

	My hash table	Prevzatá hash table	BVS inorder	BVS random
1.	0	1	4	3
2.	0	2	2	2
3.	1	2	2	2
4.	1	1	2	3
5.	1	1	2	2

Tabuľka 4 insert 25000 prvkov scenár 1 čas v milisekundách

	My hash table	Prevzatá hash table	BVS inorder	BVS random
1.	10	153	59	61
2.	9	221	59	73
3.	10	202	66	67
4.	9	189	63	65
5.	9	198	60	54

Tabuľka 5 insert 25000 prvkov scenár 2 čas v milisekundách

	My hash table	Prevzatá hash table	BVS inorder	BVS random
1.	9	183	69	70
2.	10	136	70	52
3.	9	222	59	56
4.	9	125	53	61
5.	9	210	58	76

Tabuľka 6 insert 25000 prvkov scenár 3 čas v milisekundách

	My hash table	Prevzatá hash table	BVS inorder	BVS random
1.	18	402	95	87
2.	15	417	83	92
3.	18	295	66	68
4.	13	324	69	94
5.	18	364	74	78

Tabuľka 7 insert 100 000 prvkov scenár 1 čas v milisekundách

	My hash table	Prevzatá hash table	BVS inorder	BVS random
1.	43	7351	229	217
2.	40	6816	228	216
3.	38	6861	225	217
4.	38	6894	227	220
5.	45	6972	229	216

8 insert 100 000 prvkov scenár 2 čas v milisekundách

	My hash table	Prevzatá hash table	BVS inorder	BVS random
1.	39	7005	225	213
2.	41	6923	227	213
3.	37	6922	233	233

4.	50	6867	228	216
5.	39	6849	224	217

9 insert 100 000 prvkov scenár 3 čas v milisekundách

	My hash table	Prevzatá hash table	BVS inorder	BVS random
1.	57	7432	285	280
2.	54	7276	285	272
3.	68	6957	300	276
4.	56	7212	288	268
5.	63	7256	295	268

10 search 1000 prvkov scenár 1 čas v milisekundách

	My hash table	Prevzatá hash table	BVS inorder	BVS random
1.	0	0	0	0
2.	0	0	0	0
3.	0	0	0	0
4.	0	0	0	0
5.	0	0	0	0

11 search 1000 prvkov scenár 2 čas v milisekundách

	My hash table	Prevzatá hash table	BVS
1.	0	0	0
2.	0	0	0
3.	0	0	0
4.	0	0	1
5.	0	0	0

12 search 25000 prvkov scenár 1 čas v milisekundách

	My hash table	Prevzatá hash table	BVS
1.	0	0	1
2.	0	2	3
3.	0	1	1
4.	2	1	0
5.	0	1	0

13 search 25000 prvkov scenár 2 čas v milisekundách

	My hash table	Prevzatá hash table	BVS
1.	1	12	1

2.	1	10	0
3.	1	14	0
4.	2	23	1
5.	1	12	1

14 search 100 000 prvkov scenár 1 čas v milisekundách

	My hash table	**Prevzatá hash table**	**BVS**
1.	0	0	0
2.	5	0	0
3.	0	0	0
4.	0	0	1
5.	0	0	0

15 search 100 000 prvkov scenár 2 čas v milisekundách

	My hash table	**Prevzatá hash table**	**BVS**
1.	2	62	2
2.	2	56	2
3.	2	61	3
4.	2	62	2
5.	2	61	2

16 insert 25000 prvkov into my hash resize čas v milisekundách

My hash table resize
127
131
127
125
137

17 insert 100 000 prvkov into my hash resize čas v milisekundách

My hash table resize
161
180
184
151
163

Zdôvodnenie výsledkov

Z dostupných dát sa dá usúdiť že vlastne implementovaná hashova tabuľka ma lepšie výsledky ako prebraná a to z dôvodu rozličnosti hashovacích funkcií, pri prebranej hashovej funkcii dochádza k väčšiemu množstvu kolízií čo sa odráža na čase pri vkladaní ale vyhľadávaní.

Z dát taktiež možno povedať, že pri BVS záleží od poradia prvkov. Pri väčších počtoch prvkov je vidieť, že náhodne vybraté prvky sa uložia skôr ako za sebou idúce.

Dáta hovoria aj o konštantnom čase vyhľadávania v hashovej tabuľke vo všetkých scenároch.

Všetky merania boli vykonané s procesorom Intel Core i5-6500 Skylake 3.20GHz.

Príloha A – Zadanie č. 2 Vyhľadávanie v dynamických množinách

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**.

Príloha B – Upresnenie a hodnotenie zadania

- z toho:
    - vlastná implementácia BVS – 02 body
        - vyvažovanie
    - vlastná implementácia hash tabuľky – 02 body
        - riešenie kolízií
        - zväčšovanie tabuľky
    - prevzatá implementácia BVS – 01 bod
    - prevzatá implementácia hash tabuľky – 01 bod
    - testovací program – 02 body
        - vytvoriť BVS/hash o veľkosti počtu prvkov 1 000/25 000/100 000
        - používajte rovnaké prvky pre BVS aj has tabuľku
        - operácia insert - čas potrebný na:
            - vytvorenie
            - 1 prvok
            - 25 % prvkov
            - zopakujte 5 krát
        - operácia search - čas potrebný na:
            - 1 prvok
            - 5 % prvkov
            - zopakujte 5 krát
    - dokumentácia – 02 body
        - opisy algoritmov
        - spôsob testovania
        - dosiahnuté výsledky
    - celkom: – 10 bodov