Ádám Kemény
ID: 116749

DSA - Zadanie 1 - Vyhľadávanie v dynamických množinách

Hašovacia tabulka implementácia 1 (Hash_tabulka_1.c)

Na začiatku si program alokuje pole štruktúr o veľkosti 37 s menom **tabulka**. 37 je vybrané viac menej náhodne, dôležité je aby to bolo prvočíslo, aby haš bol viac vyvážený. Vygeneruje aj zoznam prvočísel menších ako 2 000 000 **prv** pre neskoršie prealokovanie pola. Následne načítavam vstup inštrukcii a údajov v tvare (inštrukcia - 1 písmeno) (kľúč alebo kľúč a dáta) zo súboru **test.txt**. Vstupy sú následné:

- n - nový kľúč a dáta k nemu (kľúč) (data)
- d - odstráň podľa kľúča (kľúč)
- s - vyhladaj podla kľúča (kľúč) - vypíše data k príslušnému kľúču
- t - začne časovač
- c - ukončí časovač - vypíše čas splnenia príkazov od začiatku časovača
- e - skončí a kontroluje výsledok programu

Kľúč a dáta sú načítané ako *string*.

n vytvára štruktúru a uloží ju do pola na pozíciu podľa haš-u. Ak je tabuľka viac ako 50% naplnená, tak ju realokuje na dvojnásobne väčšie prvočíslo. Hodnota haš-u je daná vzorcom

$[(\text{iteracia} + a_0) * 33^n + (\text{iteracia} + a_1) * 33^{(n-1)} + \ldots + (\text{iteracia} + a_n) * 33^{(0)}]$ %velkosť_tabulky

kde *a* sú postupne znaky v stringu a iteracia sa zvyšuje o 1 vždy keď sa v tabuľke v danom mieste už nachádza kľúč a data. Realokácia je zabezpečená tak, že sa v rámci funkcie alokuje pamäť pre väčšiu tabuľku o velkosti prvočísla zo zoznamu **prv**, kde znova uloží všetky neodstránené hodnoty zo starej tabulky. Nakoniec pamäť starej tabuľky a odstránených hodnôt uvoľní.

d označí podľa kľúča hodnotu ako odstránenú, takže sa nebude dať vyhladať. Pamäť je však uvolnená až pri realokácii tabulky alebo prepísania prvku, aby sa zachovala funkcionalita vyhladávania.

s spraví heš z poskytnutého kľúča. Ak na danom mieste tabulky nenájde zhodujúci kľúč alebo nájde odstránený kľúč, tak zvýši iteráciu. Ak natrafí na zhodujúci kľúč, tak vypíše dáta. Ak je na mieste haš-u *NULL*, tak sa kľúč v danej tabuľke nenachádza.

e kontroluje správnosť údajov ak je **kontrola=1** to robí tak, že podľa **vysledky.txt** z testovacieho programu odstrani zo zoznamu hodnoty. Ak sa vo vysledky.txt nachádza hodnota ktorá v zozname nie, alebo naopak, tak hlasi chybu. Následne dealokujem pamäť.

Ádám Kemény
ID: 116749

Hašovacia tabulka implementácia 2 (Hash_tabulka_2.c)

Na rozdiel od 1. implementácii si definujem štruktúru **PRVOK**, ktorá v sebe okrem kľúča a dát uchováva ukazovateľ na ďalšiu štruktúru PRVOK. Namiesto iterácie hašovacej štruktúry budem teda vkladať kľúče s rovnakým hašom na spájaný zoznam. Alokujem si pamäť pre 17 členov, znova náhodné prvočíslo. Vytvoriť zoznam prvočísel **prv**. čítam rovnaké inštrukcie s rovnakými údajmi, aby som mohol implementácie porovnať. Funkcia príkazov sa však odlišuje.

n - namiesto započítania iterácie ak nastane kolízia, jednoducho nastaví prvok ako začiatok spájaného zoznamu, a do prvku uloží pointer na prvok ktorý tam bol uložený pôvodne. Haš funkcia je rovnaká ako pri predchádzajúcej implementácii, len bez *iterácie*. Tabulku realokujem až keď mám viac prvkov uložených ako je v tabuľke miest, keďže pri tejto implementácii sa efektivita s prirastajúcim počtom prvkov znižuje menej drasticky. Pri realokácii uložím všetky prvky na dočasné pole, nastavím ich nasledujúce prvky na *NULL.* Následne ich vložím do väčšej tabuľky o veľkosti prvočísla z **prv**. Starú a dočasnú tabulku uvolní.

d - odstraňujem prvok zo spájaného zoznamu, takže pri odstránení dbám na to aby predchádzajúci prvok ukazoval na nasledujúci po odstránenom. Patrične som ošetril aj keby odstraňujem prvok z prvého miesta. Následne uvolní pamäť obsadenú prvkom.

s - namiesto iterácie prehladáva spájaný zoznam.

e - znovu porovnáva výslednú tabulku a **vysledky.txt** ak sa nezhodujú, hlási chybu.

AVL samovyvažovací strom (AVL_strom.c)

Definuje štruktúru **NODE**, ktorá obasahuje **num** - hodnotu kľúča, **data** - uložené dáta, odkazy na pravý a ľavý node a výšku **h**. Znova mám inštrukcie n, d, s, e, t, c; aby som mohol používať rovnaký generátor súborov.

n - Zavolám funkciu, ktorá mi nájde vhodné prázdne miesto v strome, tam vytvorí nový list s údajmi (num, data), a výškou h=1. Ak je rozdiel výšok pravého a ľavého potomka niekde väčší ako 1, tak spravý rotácie alebo zig-zag rotácie, aby bol strom vyvážený. Funkcia vráti odkaz na prvý prvok v strome.

d - Odstráni prvok podľa kľúča a uvolní pamäť. Ak má potomka, tak na miesto odstráneného prvku dá potomka. Ak má dvoch potomkov, tak preskupí strom tak, aby na miesto odstráneného prvku bol daný najväčší menší prvok.

s - Hľadá prvok podla kľúča num, ak sa prvok zhoduje, tak ho vráti, ak je num menšie, prehladáva lavý podstrom, ak väčšie, tak pravý podstrom.

e - Dealokuje pamäť, ak je kontrola = 1, tak skontroluje či finálne prvky v strome sa zhodujú s finálnymi prvkami vo **vysledky.txt.**

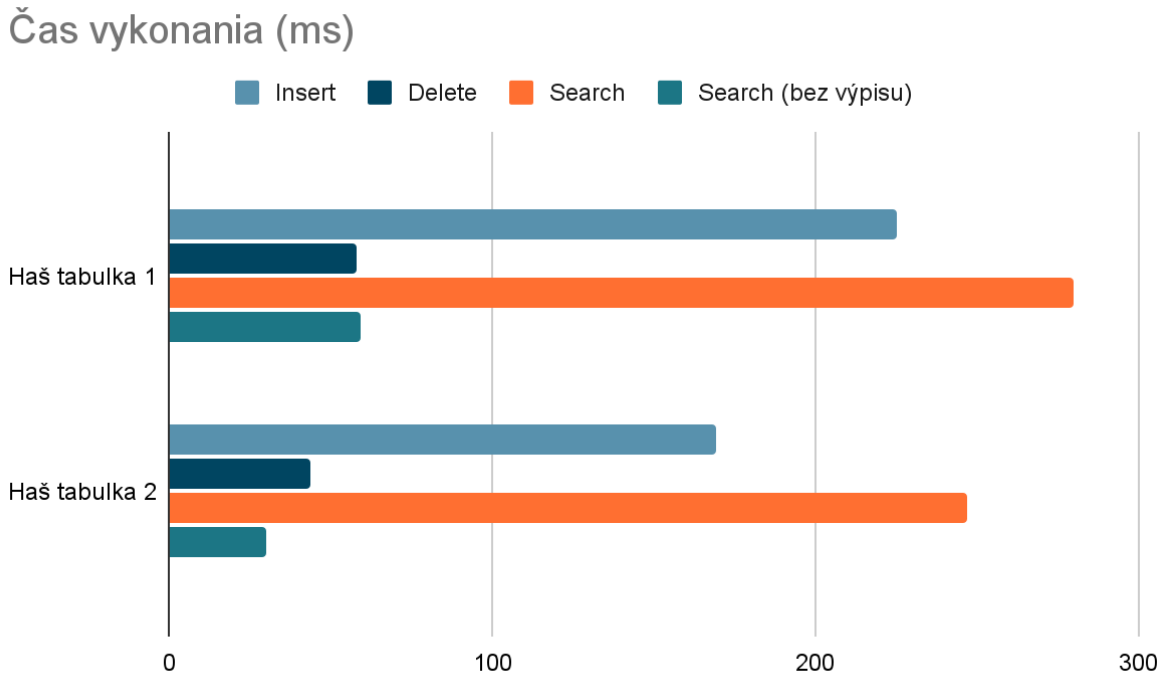
Generátor testovacích súborov a výsledkov (testOutput.c)

Vygeneruje si pole s číslami od 1 po 200 000. Do **test.txt** zapíše podla vstupu z konzoly inštrukcie a náhodne vybraté kľúče a dáta, vybrané dáta si zaznačí. V **test.txt** podľa špecifikácii bude vpísaných určitý počet insert-ov (n), delete-ov (d) a search-ov (s). Ak je ich

Ádám Kemény
ID: 116749

nenulový počet, tak sú obklúčené s **t** a **c** aby programy vypísali dĺžku vykonávania inštrukcii. Nakoniec program do **vysledky.txt** vypíše správny konečný stav programu.

Porovnanie rýchlosti



Haš tabulka 1

- Insert 150 000 prvkov - 224.765 ms
- Delete 50 000 prvkov - 57.595 ms
- Search 50 000 prvkov - 280.249 ms
- Search (bez výpisu) - 59.309 ms

Haš tabulka 2

- Insert 150 000 prvkov - 169.418 ms
- Delete 50 000 prvkov - 44.235 ms
- Search 50 000 prvkov - 246.843 ms
- Search (bez výpisu) - 29.832 ms

Heš tabulka 2 je o niečo rýchlejšia ako 1, pravdepodobne kvôli zložitosti haš funkcie s iteráciami, alebo kvôli nedostatkom v mojej implementácii.

AVL strom

- Insert 15 000 prvkov - 15 689.293 ms
- Delete 5 000 prvko - 7.855 ms
- Search 5 000 prvkov - 29.376 ms
- Search (bez výpisu) - 5.285 ms

AVL strom bol značne pomalší pri inserte. Pravdepodobne, pretože jeho ideálna zložitosť je logaritmická oproti ideálnej zložitosti 1 hash tabulky. Ostatné funkcie sú však porovnateľné.