7/4/2021

Vyhľadávanie v dynamických množinách

DSA – Zadanie 2

Samuel Hetteš
ID: 110968
STU FIIT 2020/2021

OBSAH

ZADANIE – 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).

IMPLEMENTÁCIA

- **vlastná metóda binárneho stromu:** AVL strom
- **prevzatá metóda binárneho stromu:** Red Black strom
- **vlastná metóda hashovacej tabuľky:** kolízie riešené reťazením
- **prevzatá metóda hashovacej tabuľky:** kolízie riešené lineárnym skúšaním
- jednotlivé metódy majú vlastné hlavičkové súbory, ktoré sú zahrnuté v súbori tester.c, ktorý obsahuje funkciu main a slúži na testovanie
- nižšie opísané **časové zložitosti jednotlivých funkcií sú priemerný prípad**

AVL strom – moja implementácia

✓ ŠTRUKTÚRA STROMU

- **kľúč:** unsigned long long key
- **výška:** výška daného podstromu/stromu – int height

- **ukazovateľ na ľavý a pravý podstrom:** struct avlTree *leftChild, *rightChild

✓ VYTVORENIE NOVÉHO STROMU

- ***AVLTREE *create(unsigned long long key)***

- **časová zložitosť:** O(1)
- funkcia **alokuje miesto a nastavuje parametre** nového stromu

✓ ROTÁCIA STROMU VPRAVO

- ***AVLTREE *rightRotation(AVLTREE *root)***

- **časová zložitosť:** O(1)
- funkcia **vykoná rotáciou stromu vpravo a aktualizuje výšky** vrcholov

✓ ROTÁCIA STROMU VĽAVO

- ***AVLTREE *leftRotation(AVLTREE *root)***

- **časová zložitosť:** O(1)
- funkcia **vykoná rotáciou stormu vľavo a aktualizuje výšky** vrcholov

✓ PRIDANIE DÁT DO STROMU

- ***AVLTREE *insertAVL(AVLTREE *vertex, unsigned long long key)***

- **časová zložitosť:** O(log n)
- **rekurzia**
- **ak sa pridávané dáta zhodujú s dátami v strome, nič sa nepridá**
- ak nájdeme list, vytvorí sa nový strom pre dané dáta
- inak sa porovná veľkosť dát, na základe čoho sa funkcia rekurzívne zavolá pre ľavý/pravý podstrom

- aktualizuje sa výška vrcholu, ktorý bol rodičom pridaného stromu a vypočíta sa výškový rozdiel ľavého a pravého podstromu
- ak je výškový rozdiel -1/0/1 strom je vyvážený
- **riešenie kolízii:**
- výškový rozdiel je väčší ako 1 – strom je nevyvážený vľavo:
    - ak pridávaný kľúč je väčší ako kľúč ľavého potomka rodiča – rotácia rodiča vpravo
    - ak pridávaný kľúč je menší ako kľúč ľavého potomka rodiča – rotácia ľavého potomka vľavo a následne rotácia rodiča vpravo
- výškový rozdiel je menší ako -1 – strom je nevyvážený vpravo
    - ak pridávaný kľúč je väčší ako kľúč pravého potomka – rotácia rodiča vľavo
    - ak pridávaný kľúč je menší ako kľúč pravého potomka – rotácia pravého potomka vpravo a následne rotácia rodiča vľavo

✓ VYHĽADANIE DÁT V STROME

▪ *AVLTREE *searchAVL(AVLTREE *vertex, unsigned long long key)*

- **časová zložitosť:** O(log n)
- **rekurzia**
- ak strom neexistuje – null, inak porovná kľúč a volá rekurzívne samú seba pre ľavý alebo pravý podstrom / zhoda – vráti strom

✓ UVOĽNENIE PAMÄTI

▪ *void freeAVL(AVLTREE *vertex)*

- **rekurzia**
- ak strom neexistuje ukončí volanie, inak volá samú seba pre ľavý a pravý podstrom
- **uvoľní pamäť**

Red Black strom – prevzatá implementácia

- **zdroj kódu:** https://www.programiz.com/dsa/red-black-tree
- v prevzatom kóde som odstránil nepotrebné časti z hľadiska zadania a upravil typ pridávaných dát
- **koreň stromu** bol v tejto implementácii ako **globálna premenná**

✓ ŠTRUKTÚRA STROMU

- **farba:** červená alebo čierna – int color
- **kľúč:** unsigned long long key
- **ľavý a pravý potomok:** struct rbNode *link[2]

✓ VYTVORENIE NOVÉHO STROMU

- ***struct rbNode *createNode(unsigned long long data)***

- **časová zložitosť:** O(1)
- funkcia **alokuje miesto a nastavuje parametre** nového stromu
- nový strom má vždy červenú farbu

✓ PRIDANIE DÁT DO STROMU

- ***void insertRB(unsigned long long data)***

- **časová zložitosť:** O(log n)
- **iteratívna metóda**
- ak v koreni nič nie je, vyvorí nový strom pre dáta a ukončí sa
- inak cyklus:
- **v prípade zhody dát sa funkcia ukončí**, inak porovná dáta a hľadá v ľavom/pravom podstrome
- po nájdení listu sa vytvorí nový strom s danými dátami
- **riešenie kolízií:**

- ak je výška stromu aspoň 3 a rodič nového stromu je červený bude sa vykonávať cyklus:
- a) ak je rodič pridaného vrcholu ľavým potomkom prarodiča vykoná:
    - 1. ak farba pravého potomka prarodiča je červená, nastaví farbu prarodiča na červenú a obom potomkom prarodiča čiernu
    - 2. inak ak je pridaný vrchol pravým potomkom rodiča zrotuje pridaný vrchol vpravo
    - nastaví farbu prarodiča na červenú, rodiča na čiernu a spraví rotáciu prarodiča vľavo
- b) ak je rodič pridaného vrcholu pravým potomkom prarodiča vykoná:
    - 1. ak farba ľavého potomka prarodiča pridaného vrcholu je červená, nastaví farbu oboch potomkov prarodiča na čiernu a jeho farbu na červenú
    - 2. ak pridaný vrchol je ľavým potomkom jeho rodiča, tak spraví rotáciu pridaného vrcholu vľavo
    - nastaví farbu prarodiča na červenú a farbu rodiča na čiernu, spraví rotáciu prarodiča vpravo
- nakoniec nastaví farbu koreňa stromu na čiernu

✓ VYHĽADANIE DÁT V STROME

- ***struct rbNode *searchRB(unsigned long long value)***

- **časová zložitosť:** O(log n)
- vyhľadávacia funkcia funguje rovnako ako pri AVL strome, ale **vyhľadáva sa while cyklom**, nie rekurzívne

✓ UVOĽNENIE PAMÄTI

- ***void freeRB(struct rbNode *vertex)***

- funkcia pre **uvoľnenie pamäti** v pôvodnej implementácii nebola, preto som ju implementoval samostatne na rovnakom princípe ako pri AVL strome

Hashovacia tabuľka – moja implementácia

- riešenie kolízií: **zreťazenie**
- **veľkosť tabuľky – prvočíslo**

✓ ŠTRUKTÚRA DÁT

- **kľúč:** unsigned long long key
- **ukazovateľ na ďalšie dáta:** struct data *next

✓ ŠTRUKTÚRA TABUĽKY

- **ukazovateľ na dáta:** DATA **data
- **veľkosť tabuľky:** unsigned long long size
- **počet vložených prvkov:** unsigned lont long inserted

✓ ZÍSKANIE PRVOČÍSLA PRE VEĽKOSŤ TABUĽKY

- ***unsigned long long getPrime(unsigned long long number)***

- **časová zložitosť:** O(n)
- funkcia slúži pre **vyhľadanie prvočísla vhodného ako veľkosť tabuľky**
- for cyklom kontroluje čísla od number po 2, pre každé číslo kontroluje deliteľov počnúc od 2 po odmocninu z daného čísla
- ak sa nájde deliteľ funkcia prejde na ďalšie číslo, inak vráti toto prvočíslo
- **ak by sa žiadne prvočíslo nenašlo funkcia vráti pôvodné číslo**

✓ INICIALIZÁCIA TABUĽKY

- ***HASHTABLE *initHashOwn(HASHTABLE *hashTable, unsigned long long tableSize)***

- **časová zložitosť:** O(n)

- funkcia slúži pre **inicializáciu hashovacej tabuľky**
- **alokuje sa miesto** pre hashovaciu tabuľku a **nastavia sa počiatočné parametre**

✓ VYTVORENIE NOVÝCH DÁT

- ***DATA *createData(unsigned long long key)***

- **časová zložitosť:** O(1)
- **alokuje miesto** pre nové dáta a **nastaví parametre**

✓ HASHOVACIA FUNKCIA

- ***unsigned long long hashData(HASHTABLE *hashTable, unsigned long long key)***

- **časová zložitosť:** O(1)
- hashovacia funkcia – **index** do tabuľky sa vypočíta ako **key modulo veľkosť tabuľky**

✓ ZVÄČŠENIE VEĽKOSTI TABUĽKY

- ***HASHTABLE *resizeHashTable(HASHTABLE *hashTable)***

- **časová zložitosť:** O(n)
- funkcia, ktorá slúži pre **zväčšenie veľkosti pôvodnej tabuľky na dvojnásobok**
- táto funkcia **je vypnutá v testovacích scenároch**
- pôvodná veľkosť tabuľky sa zväčší na približne dvojnásobok – nájde sa prvočíslo, ktoré bude vyhovovať funkciou getPrime
- realokuje sa miesto pre ukazovateľe, ktoré sa nastavia na NULL
- všetky **vložené dáta sa nanovo vložia** do tabuľky s novými indexami

✓ PRIDANIE DÁT DO TABUĽKY

- ***int insertHashOwn(HASHTABLE *hashTable, unsigned long long key)***

- **časová zložitosť:** O(1)
- funkcia pre **vloženie dát** do tabuľky
- ak tabuľka na nič neukazuje, vráti 0, inak vypočíta index hashovacou funkciou
- vo funkcii je **zakomentovaná časť, ktorá kontroluje zaplnenie tabuľky a jej prípadné zväčšenie** funkciou resizeHashTable
- **riešenie kolízii:**
- ak na danom indexe sú uložené dáta, prejde ich nasledovníkmi, dokým nenájde koniec, kde pridá nové dáta funkciou createData
- ak by sme pri prechádzaní **natrafili na rovnaké dáta, nepridávame tieto dáta** a ukončíme funkciu
- ak na danom indexe nič nie je, vytvorí na danom indexe nové dáta
- zvýši počet pridaných prvkov v hashovacej tabuľke a vráti 1

✓ VYHĽADANIE DÁT V TABUĽKE

- ***DATA *searchHashOwn(HASHTABLE *hashTable, unsigned long long key)***

- **časová zložitosť:** O(1)
- funkcia, ktorá slúži pre **vyhľadanie dát**
- ak tabuľka na nič neukazuje vráti NULL, inak vypočíta index pre daný kľúč
- ak na danom indexe v hashovacej tabuľke niečo je, porovná tieto dáta s hľadanými, ak sa zhodujú vráti pointer na tieto dáta, inak prejdie nasledovníkmi týchto dát
- ak sa dáta nenašli vráti NULL

✓ UVOĽNENIE PAMÄTI

- ***void freeHashOwn(HASHTABLE *hashTable)***

- funkcia, ktorá slúži pre **uvoľnenie dát** tabuľky
- prejde všetkými indexami tabuľky a uvoľní jednotlivé dáta a ich nasledovníkov
- uvoľní miesto alokované pre uloženie ukazovateľa na ukazovatele na dáta a uvoľní miesto alokované pre tabuľku

Hashovacia tabuľka – prevzatá implementácia

- **zdroj kódu:** https://www.codingalpha.com/hash-table-c-program/
- riešenie kolízií: **lineárne skúšanie**
- **veľkosť tabuľky – podľa zvolenia (nepočíta prvočíslo)**
- v prevzatom kóde som odstránil nepotrebné časti z hľadiska zadania, zmenil typ pridávaných dát a pridal parameter veľkosť tabuľky medzi volaniami funkcií, keďže v pôvodnej implementácii bola veľkosť tabuľky definovaná ako konštanta, taktiež som zmenil alokáciu tabuľky na dynamickú, aby bolo možno pracovať s väčším množstvom dát

✓ ŠTRUKTÚRA DÁT

- **kľúč:** unsigned long long key
- **status:** empty/occupied – enum data_status status

✓ INICIALIZÁCIA TABUĽKY

- **časová zložitosť:** O(n)
- inicializácia tabuľky prebieha priamo v testeri, kde sa na každom indexe dát nastaví status occupied

✓ HASHOVACIA FUNKCIA

- ***unsigned long long hash_function(unsigned long long key, unsigned long long tableSize)***

- **časová zložitosť:** O(1)
- **hashovacia funkcia je identická** s hashovacou funkciou v mojej implementácii

✓ PRIDANIE DÁT DO TABUĽKY

- ***int insertHashTaken(unsigned long long key, struct Data *hash_table, unsigned long long tableSize)***

- **časová zložitosť:** O(1)
- funkcia, ktorá slúži pre **pridanie dát** do tabuľky
- vypočíta index do tabuľky hashovacou funkciou
- začne cyklus:
- overí či na danom indexe majú dáta voľný status, ak áno, zapíše dáta, zmení status a vráti 1
- ak na danom indexe už sú dáta, overí ich kľúč s pridávaným**, ak nastane zhoda funkcia vráti 0, nič neprída**
- **riešenie kolízií:**
- inak funkcia vypočíta **nový index** pre dáta: index = **(pôvodný index + počítadlo) modulo veľkosť tabuľky**
- počítadlo začína od 1 a inkrementuje sa každou kolíziou
- opäť prebehne cyklus pridávania

✓ VYHĽADANIE DÁT V TABUĽKE

- ***unsigned long long searchHashTaken(unsigned long long key, struct Data *hash_table, unsigned long long tableSize)***

- **časová zložitosť:** O(1)
- funkcia, ktorá slúži pre **hľadanie dát** v tabuľke
- vypočíta index dát hashovacou funkciou

- začne cyklus:
- overí či na danom indexe sú nejaké dáta, ak nie, vráti -1
- ak na danom indexe sú dáta, overí ich kľúč, ak sa zhoduje funkcia vráti tieto dáta
- ak sa nezhoduje, vypočíta nový index ako v prípade pridávania do tabuľky a zopakuje sa cyklus hľadania

✓ **UVOĽNENIE PAMÄTI**

▪ ***void freeHashTaken(struct Data *hashTable)***

- funkcia, ktorá slúži pre **uvoľnenie alokovaného miesta**
- free(hashTable)

TESTOVANIE

- **testovacie dáta:** unsigned long long – 10 až 20 ciferné čísla
- v každej metóde som pridal **globálne premenné**, ktoré počítajú výskyt kolízií v tabuľkách / potrebných vyvažovaní stromu

Tester

- **vstupné parametre:**
  - **veľkosť tabuľky**
  - **počet vkladaných dát**
  - **počet hľadaných dát – ak je 0, bude testovať iba insert**
  - **parameter lineárne dáta – 1(zapnuté) / 0(vypnuté)**
- na počiatku sa inicializujú obe hashovacie tabuľky
- následne prebieha cyklus na základe počtu vkladaných dát:
- funkcia náhodne vypočíta 10-20 ciferné číslo, ktoré bude použité ako vstupné dáta a na základe toho či sú zapnuté alebo vypnuté lineárne dáta bude generovať:
  - vždy nové dáta
  - inkrementovať predchádzajúce

- funkcia vykoná vloženie dát naprieč všetkým metódam a odmerá čas vkladania dát
- ak počet hľadaných dát nie je 0, funkcia taktiež vyhľadá dáta a odmerá čas hľadania a zopakuje cyklus
- na záver funkcia vypíše nastavené vstupné parametre, výsledky prebehnutých testov pre jednotlivé metódy, počet vyskytnutých kolízií v tabuľkách a počet potrebných vyvažovaní stromov

Testy

- časy boli počítané funciou clock() z knižnice time.h, ich presnosť nie je najlepšia, ale dáva nám približnú predstavu o tom, ako sú jednotlivé metódy výhodné v daných situáciách
- v každom teste sa pridávajú aj vyhľadávajú dáta v pomere 1:1

✓ TESTOVANIE PRIDÁVANIA A HĽADANIA MALÉHO POČTU DÁT

- **počet pridávaných/hľadaných dát:** 100 000

▪ *Test č. 1 – ideálna veľkosť tabuľky, lineárne dáta*

- **veľkosť tabuľky:** 200 000
- **výsledky:**

```
Test - inserting 100000 items and searching 100000 items
Available hash table size: 200000
Linear data
----------------------------------------
Insert time of:
AVL TREE: 0.025000 s
RED BLACK TREE: 0.019000 s
CHAINING HASH TABLE: 0.014000 s
LINEAR PROBING HASH TABLE: 0.000000 s
----------------------------------------
Search time of:
AVL TREE: 0.006000 s
RED BLACK TREE: 0.013000 s
CHAINING HASH TABLE: 0.001000 s
LINEAR PROBING HASH TABLE: 0.000000 s
----------------------------------------
Overall time:
AVL TREE: 0.031000 s
RED BLACK TREE: 0.032000 s
CHAINING HASH TABLE: 0.015000 s
LINEAR PROBING HASH TABLE: 0.000000 s
----------------------------------------
Trees were balanced:
AVL TREE: 99983 times
RED BLACK TREE: 199933 times
----------------------------------------
Collisions in hash tables:
CHAINING HASH TABLE: 0
LINEAR PROBING HASH TABLE: 0
```

- **zistenia:**
  - **čas potrebný na pridanie:** avl > red black > chaining table > **linear probing table**
  - **čas potrebný na vyhľadanie:** red black > avl > chaining table > **linear probing table**
  - **celkový čas:** red black > avl > chaining table > **linear probing table**
  - **počet vyvažovaní stromu:** pri red black strome zhruba dvojnásobok ako pri avl strome
  - **počet kolízií:** žiadne kolízie, keďže máme lineárne dáta
  - **ideálna metóda: tabuľka s lineárnym skúšaním**

▪ *Test č. 2 – ideálna veľkosť tabuľky, náhodné dáta*

- **veľkosť tabuľky:** 200 000
- **výsledky:**

```
Test - inserting 100000 items and searching 100000 items
Available hash table size: 200000
Random data
----------------------------------------
Insert time of:
AVL TREE: 0.048000 s
RED BLACK TREE: 0.042000 s
CHAINING HASH TABLE: 0.020000 s
LINEAR PROBING HASH TABLE: 0.010000 s
----------------------------------------
Search time of:
AVL TREE: 0.011000 s
RED BLACK TREE: 0.007000 s
CHAINING HASH TABLE: 0.002000 s
LINEAR PROBING HASH TABLE: 0.002000 s
----------------------------------------
Overall time:
AVL TREE: 0.059000 s
RED BLACK TREE: 0.049000 s
CHAINING HASH TABLE: 0.022000 s
LINEAR PROBING HASH TABLE: 0.012000 s
----------------------------------------
Trees were balanced:
AVL TREE: 41738 times
RED BLACK TREE: 183121 times
----------------------------------------
Collisions in hash tables:
CHAINING HASH TABLE: 2715
LINEAR PROBING HASH TABLE: 37241
```

- **zistenia:**
  - **čas potrebný na pridanie:** avl > red black > chaining table > **linear probing table**
  - **čas potrebný na vyhľadanie:** avl > red black > **chaining table** = **linear probing table**

- **celkový čas:** avl > red black > chaining table > **linear probing table**
- **počet vyvažovaní stromu:** pri red black strome zhruba štvornásobok ako pri avl strome
- **počet kolízií:** počet kolízií pri náhodných dátach je mnohonásobne väčší pri tabuľke s lineárnym skúšaním
- **ideálna metóda: tabuľka s lineárnym skúšaním**

- ***Test č. 3 –veľkosť tabuľky rovnaká ako počet dát, lineárne dáta***

- **veľkosť tabuľky:** 100 000
- **výsledky:**

```
Test - inserting 100000 items and searching 100000 items
Available hash table size: 100000
Linear data
----------------------------------------
Insert time of:
AVL TREE: 0.040000 s
RED BLACK TREE: 0.031000 s
CHAINING HASH TABLE: 0.007000 s
LINEAR PROBING HASH TABLE: 0.003000 s
----------------------------------------
Search time of:
AVL TREE: 0.006000 s
RED BLACK TREE: 0.005000 s
CHAINING HASH TABLE: 0.002000 s
LINEAR PROBING HASH TABLE: 0.002000 s
----------------------------------------
Overall time:
AVL TREE: 0.046000 s
RED BLACK TREE: 0.036000 s
CHAINING HASH TABLE: 0.009000 s
LINEAR PROBING HASH TABLE: 0.005000 s
----------------------------------------
Trees were balanced:
AVL TREE: 99983 times
RED BLACK TREE: 199933 times
----------------------------------------
Collisions in hash tables:
CHAINING HASH TABLE: 0
LINEAR PROBING HASH TABLE: 0
```

- **zistenia:**
  - **čas potrebný na pridanie:** avl > red black > chaining table > **linear probing table**
  - **čas potrebný na vyhľadanie:** avl > red black > **chaining table** = **linear probing table**
  - **celkový čas:** avl > red black > chaining table > **linear probing table**
  - **počet vyvažovaní stromu:** pri red black strome zhruba dvojnásobok ako pri avl strome
  - **počet kolízií:** žiadne kolízie, keďže máme lineárne dáta
  - **ideálna metóda: tabuľka s lineárnym skúšaním**

- ***Test č. 4 –veľkosť tabuľky rovnaká ako počet dát, náhodné dáta***

- **veľkosť tabuľky:** 100 000
- **výsledky:**

```
Test - inserting 100000 items and searching 100000 items
Available hash table size: 100000
Random data
----------------------------------------
Insert time of:
AVL TREE: 0.053000 s
RED BLACK TREE: 0.031000 s
CHAINING HASH TABLE: 0.017000 s
LINEAR PROBING HASH TABLE: 0.018000 s
----------------------------------------
Search time of:
AVL TREE: 0.009000 s
RED BLACK TREE: 0.009000 s
CHAINING HASH TABLE: 0.000000 s
LINEAR PROBING HASH TABLE: 0.005000 s
----------------------------------------
Overall time:
AVL TREE: 0.062000 s
RED BLACK TREE: 0.040000 s
CHAINING HASH TABLE: 0.017000 s
LINEAR PROBING HASH TABLE: 0.023000 s
----------------------------------------
Trees were balanced:
AVL TREE: 42192 times
RED BLACK TREE: 186349 times
----------------------------------------
Collisions in hash tables:
CHAINING HASH TABLE: 9858
LINEAR PROBING HASH TABLE: 457757
```

- **zistenia:**
  - **čas potrebný na pridanie:** avl > red black > linear probing table > **chaining table**
  - **čas potrebný na vyhľadanie:** avl = red black > chaining table > **linear probing table**
  - **celkový čas:** avl > red black > linear probing table > **chaining table**
  - **počet vyvažovaní stromu:** pri red black strome zhruba štvornásobok ako pri avl strome
  - **počet kolízií:** počet kolízií pri náhodných dátach je mnohonásobne väčší pri tabuľke s lineárnym skúšaním
  - **ideálna metóda: tabuľka s reťazením**

✓ TESTOVANIE PRIDÁVANIA A HĽADANIA STREDNÉHO POČTU DÁT

- **počet pridávaných/hľadaných dát:** 1 000 000

- ***Test č. 1 – ideálna veľkosť tabuľky, lineárne dáta***

- **veľkosť tabuľky:** 2 000 000
- **výsledky:**

```
Test - inserting 1000000 items and searching 1000000 items
Available hash table size: 2000000
Linear data
----------------------------------------
Insert time of:
AVL TREE: 0.341000 s
RED BLACK TREE: 0.311000 s
CHAINING HASH TABLE: 0.110000 s
LINEAR PROBING HASH TABLE: 0.012000 s
----------------------------------------
Search time of:
AVL TREE: 0.034000 s
RED BLACK TREE: 0.101000 s
CHAINING HASH TABLE: 0.016000 s
LINEAR PROBING HASH TABLE: 0.008000 s
----------------------------------------
Overall time:
AVL TREE: 0.375000 s
RED BLACK TREE: 0.412000 s
CHAINING HASH TABLE: 0.126000 s
LINEAR PROBING HASH TABLE: 0.020000 s
----------------------------------------
Trees were balanced:
AVL TREE: 999980 times
RED BLACK TREE: 1999920 times
----------------------------------------
Collisions in hash tables:
CHAINING HASH TABLE: 0
LINEAR PROBING HASH TABLE: 0
```

- **zistenia:**
  - **čas potrebný na pridanie:** avl > red black > linear probing table > **chaining table**
  - **čas potrebný na vyhľadanie:** avl > red black > chaining table > **linear probing table**
  - **celkový čas:** red black > avl > chaining table > **linear probing table**
  - **počet vyvažovaní stromu:** pri red black strome zhruba dvojnásobok ako pri avl strome
  - **počet kolízií:** žiadne kolízie, keďže máme lineárne dáta
  - **ideálna metóda: tabuľka s lineárnym skúšaním**

```
Test - inserting 1000000 items and searching 1000000 items
Available hash table size: 2000000
Random data
----------------------------------------
Insert time of:
AVL TREE: 1.148000 s
RED BLACK TREE: 0.359000 s
CHAINING HASH TABLE: 0.167000 s
LINEAR PROBING HASH TABLE: 0.082000 s
----------------------------------------
Search time of:
AVL TREE: 0.174000 s
RED BLACK TREE: 0.115000 s
CHAINING HASH TABLE: 0.006000 s
LINEAR PROBING HASH TABLE: 0.018000 s
----------------------------------------
Overall time:
AVL TREE: 1.322000 s
RED BLACK TREE: 0.474000 s
CHAINING HASH TABLE: 0.173000 s
LINEAR PROBING HASH TABLE: 0.100000 s
----------------------------------------
Trees were balanced:
AVL TREE: 419561 times
RED BLACK TREE: 1851649 times
----------------------------------------
Collisions in hash tables:
CHAINING HASH TABLE: 27404
LINEAR PROBING HASH TABLE: 370591
```

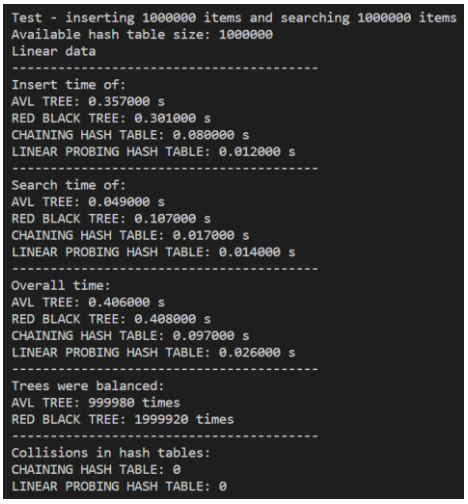
ľkosť tabuľky, náhodné dáta

00

- **zistenia:**
    - **čas potrebný na pridanie:** avl > red black > chaining table > **linear probing table**
    - **čas potrebný na vyhľadanie:** avl > red black > linear probing table > **chaining table**
    - **celkový čas:** avl > red black > chaining table > **linear probing table**
    - **počet vyvažovaní stromu:** pri red black strome zhruba štvornásobok ako pri avl strome
    - **počet kolízií:** počet kolízií pri náhodných dátach je mnohonásobne väčší pri tabuľke s lineárnym skúšaním
    - **ideálna metóda: tabuľka s lineárnym skúšaním**

▪ *Test č. 3 –veľkosť tabuľky rovnaká ako počet dát, lineárne dáta*

- **veľkosť tabuľky:** 1 000 000
- **výsledky:**



- **zistenia:**
  - **čas potrebný na pridanie:** avl > red black > chaining table > **linear probing table**
  - **čas potrebný na vyhľadanie:** red black > avl > chaining table > **linear probing table**
  - **celkový čas:** red black > avl > chaining table > **linear probing table**
  - **počet vyvažovaní stromu:** pri red black strome zhruba dvojnásobok ako pri avl strome
  - **počet kolízií:** žiadne kolízie, keďže máme lineárne dáta
  - **ideálna metóda: tabuľka s lineárnym skúšaním**

- ***Test č. 4 –veľkosť tabuľky rovnaká ako počet dát, náhodné dáta***

- **veľkosť tabuľky:** 100 000
- **výsledky:**

```
Test - inserting 1000000 items and searching 1000000 items
Available hash table size: 1000000
Random data
----------------------------------------
Insert time of:
AVL TREE: 1.073000 s
RED BLACK TREE: 0.467000 s
CHAINING HASH TABLE: 0.186000 s
LINEAR PROBING HASH TABLE: 0.112000 s
----------------------------------------
Search time of:
AVL TREE: 0.164000 s
RED BLACK TREE: 0.100000 s
CHAINING HASH TABLE: 0.024000 s
LINEAR PROBING HASH TABLE: 0.070000 s
----------------------------------------
Overall time:
AVL TREE: 1.237000 s
RED BLACK TREE: 0.567000 s
CHAINING HASH TABLE: 0.210000 s
LINEAR PROBING HASH TABLE: 0.182000 s
----------------------------------------
Trees were balanced:
AVL TREE: 420197 times
RED BLACK TREE: 1848543 times
----------------------------------------
Collisions in hash tables:
CHAINING HASH TABLE: 98648
LINEAR PROBING HASH TABLE: 4136333
```

- **zistenia:**
  - **čas potrebný na pridanie:** avl > red black > chaining table > **linear probing table**
  - **čas potrebný na vyhľadanie:** avl > red black > linear probing table > **chaining table**
  - **celkový čas:** avl > red black > chaining table > **linear probing table**
  - **počet vyvažovaní stromu:** pri red black strome zhruba štvornásobok ako pri avl strome
  - **počet kolízií:** počet kolízií pri náhodných dátach je mnohonásobne väčší pri tabuľke s lineárnym skúšaním
  - **ideálna metóda: tabuľka s lineárnym skúšaním**

✓ TESTOVANIE PRIDÁVANIA A HĽADANIA VEĽKÉHO POČTU DÁT

- **počet pridávaných/hľadaných dát:** 10 000 000

▪ *Test č. 1 – ideálna veľkosť tabuľky, lineárne dáta*

- **veľkosť tabuľky:** 20 000 000
- **výsledky:**

```
Test - inserting 10000000 items and searching 10000000 items
Available hash table size: 20000000
Linear data
----------------------------------------
Insert time of:
AVL TREE: 4.239000 s
RED BLACK TREE: 4.425000 s
CHAINING HASH TABLE: 0.871000 s
LINEAR PROBING HASH TABLE: 0.067000 s
----------------------------------------
Search time of:
AVL TREE: 0.897000 s
RED BLACK TREE: 1.674000 s
CHAINING HASH TABLE: 0.195000 s
LINEAR PROBING HASH TABLE: 0.139000 s
----------------------------------------
Overall time:
AVL TREE: 5.136000 s
RED BLACK TREE: 6.099000 s
CHAINING HASH TABLE: 1.066000 s
LINEAR PROBING HASH TABLE: 0.206000 s
----------------------------------------
Trees were balanced:
AVL TREE: 9999976 times
RED BLACK TREE: 19999904 times
----------------------------------------
Collisions in hash tables:
CHAINING HASH TABLE: 0
LINEAR PROBING HASH TABLE: 0
```

- **zistenia:**
  - **čas potrebný na pridanie:** red black > avl > chaining table > **linear probing table**
  - **čas potrebný na vyhľadanie:** red black > avl > chaining table > **linear probing table**
  - **celkový čas:** red black > avl > chaining table > **linear probing table**
  - **počet vyvažovaní stromu:** pri red black strome zhruba dvojnásobok ako pri avl strome
  - **počet kolízií:** žiadne kolízie, keďže máme lineárne dáta
  - **ideálna metóda: tabuľka s lineárnym skúšaním**

▪ *Test č. 2 – ideálna veľkosť tabuľky, náhodné dáta*

- **veľkosť tabuľky:** 20 000 000
- **výsledky:**

```
Test - inserting 10000000 items and searching 10000000 items
Available hash table size: 20000000
Random data
----------------------------------------
Insert time of:
AVL TREE: 18.909000 s
RED BLACK TREE: 5.000000 s
CHAINING HASH TABLE: 2.754000 s
LINEAR PROBING HASH TABLE: 1.755000 s
----------------------------------------
Search time of:
AVL TREE: 2.348000 s
RED BLACK TREE: 2.021000 s
CHAINING HASH TABLE: 0.180000 s
LINEAR PROBING HASH TABLE: 0.212000 s
----------------------------------------
Overall time:
AVL TREE: 21.257000 s
RED BLACK TREE: 7.021000 s
CHAINING HASH TABLE: 2.934000 s
LINEAR PROBING HASH TABLE: 1.967000 s
----------------------------------------
Trees were balanced:
AVL TREE: 4198464 times
RED BLACK TREE: 18375735 times
----------------------------------------
Collisions in hash tables:
CHAINING HASH TABLE: 273866
LINEAR PROBING HASH TABLE: 3724935
```

- **zistenia:**
  - **čas potrebný na pridanie:** avl > red black > chaining table > **linear probing table**
  - **čas potrebný na vyhľadanie:** avl > red black > linear probing table > **chaining table**
  - **celkový čas:** avl > red black > chaining table > **linear probing table**
  - **počet vyvažovaní stromu:** pri red black strome zhruba štvornásobok ako pri avl strome
  - **počet kolízií:** počet kolízií pri náhodných dátach je mnohonásobne väčší pri tabuľke s lineárnym skúšaním
  - **ideálna metóda: tabuľka s lineárnym skúšaním**

- ***Test č. 3 –veľkosť tabuľky rovnaká ako počet dát, lineárne dáta***

- **veľkosť tabuľky:** 10 000 000
- **výsledky:**

```
Test - inserting 10000000 items and searching 10000000 items
Available hash table size: 10000000
Linear data
----------------------------------------
Insert time of:
AVL TREE: 4.395000 s
RED BLACK TREE: 4.469000 s
CHAINING HASH TABLE: 0.832000 s
LINEAR PROBING HASH TABLE: 0.190000 s
----------------------------------------
Search time of:
AVL TREE: 0.873000 s
RED BLACK TREE: 1.827000 s
CHAINING HASH TABLE: 0.198000 s
LINEAR PROBING HASH TABLE: 0.172000 s
----------------------------------------
Overall time:
AVL TREE: 5.268000 s
RED BLACK TREE: 6.296000 s
CHAINING HASH TABLE: 1.030000 s
LINEAR PROBING HASH TABLE: 0.362000 s
----------------------------------------
Trees were balanced:
AVL TREE: 9999976 times
RED BLACK TREE: 19999904 times
----------------------------------------
Collisions in hash tables:
CHAINING HASH TABLE: 0
LINEAR PROBING HASH TABLE: 0
```

- **zistenia:**
  - **čas potrebný na pridanie:** red black > avl > chaining table > **linear probing table**
  - **čas potrebný na vyhľadanie:** red black > avl > chaining table > **linear probing table**
  - **celkový čas:** red black > avl > chaining table > **linear probing table**
  - **počet vyvažovaní stromu:** pri red black strome zhruba dvojnásobok ako pri avl strome
  - **počet kolízií:** žiadne kolízie, keďže máme lineárne dáta
  - **ideálna metóda: tabuľka s lineárnym skúšaním**

- ***Test č. 4 –veľkosť tabuľky rovnaká ako počet dát, náhodné dáta***

- **veľkosť tabuľky:** 10 000 000
- **výsledky:**

```
Test - inserting 10000000 items and searching 10000000 items
Available hash table size: 10000000
Random data
----------------------------------------
Insert time of:
AVL TREE: 20.526000 s
RED BLACK TREE: 5.367000 s
CHAINING HASH TABLE: 3.040000 s
LINEAR PROBING HASH TABLE: 1.997000 s
----------------------------------------
Search time of:
AVL TREE: 2.687000 s
RED BLACK TREE: 2.011000 s
CHAINING HASH TABLE: 0.220000 s
LINEAR PROBING HASH TABLE: 0.611000 s
----------------------------------------
Overall time:
AVL TREE: 23.213000 s
RED BLACK TREE: 7.378000 s
CHAINING HASH TABLE: 3.260000 s
LINEAR PROBING HASH TABLE: 2.608000 s
----------------------------------------
Trees were balanced:
AVL TREE: 4198816 times
RED BLACK TREE: 18407988 times
----------------------------------------
Collisions in hash tables:
CHAINING HASH TABLE: 990325
LINEAR PROBING HASH TABLE: 41269507
```

- **zistenia:**
  - **čas potrebný na pridanie:** avl > red black > chaining table > **linear probing table**
  - **čas potrebný na vyhľadanie:** avl > red black > linear probing table > **chaining table**
  - **celkový čas:** avl > red black > chaining table > **linear probing table**
  - **počet vyvažovaní stromu:** pri red black strome zhruba štvornásobok ako pri avl strome
  - **počet kolízií:** počet kolízií pri náhodných dátach je mnohonásobne väčší pri tabuľke s lineárnym skúšaním
  - **ideálna metóda: tabuľka s lineárnym skúšaním**

Zhodnotenie testov

- na základe vykonaných testov a zistení môžeme vyvodiť aké metódy sú výhodne v určitých situáciach:
- **malý počet dát:**
  - **ideálna metóda** sa v rozličných situáciach javí ako **tabuľka s lineárnym skúšaním**, avšak pri **náhodných dátach a rovnakej veľkosti tabuľky** ako je počet dát efektívnejšia pracovala **tabuľka s reťazením**
  - **ideálny strom: red black strom**, avšak rozdiely v efektivite sú minimálne
- **stredný počet dát:**
  - **ideálna metóda** v rozličných situáciách je opäť **tabuľka s lineárnym skúšaním**, avšak ak máme **náhodné dáta**, tak pre ich **vyhľadanie** je efektívnejšia **tabuľka s reťazením**
  - **ideálny strom: red black strom**
- **veľký počet dát:**
  - **ideálna metóda** je znova **tabuľka s lineárnym skúšaním**, avšak pri **náhodných dátach** je **vyhľadávanie** efektívnejšie pri **tabuľke s reťazením** ako v prípade stredného počtu dát
  - **ideálny strom: red black strom**

- **počet kolízií:** pri pridávaní náhodných dát je počet kolízií pri tabuľke s reťazením mnohonásobne menší ako pri tabuľke s lineárnym skúšaním, ktorej veľkosť nie je prvočíslo

ZÁVER

- na základe zhodnotenia testov môžeme usúdiť, že univerzálnou metódou na použitie pri rozličných počtoch dát je určite tabuľka s lineárnym skúšaním, ktorej pridávanie je veľmi rýchle
- ak by sme však potrebovali dáta iba vyhľadávať, lepšou metódou bude zvolenie tabuľky s reťazením
- stromy pracovali o dosť pomalšie ako tabuľky, avšak ak by sme si mali zvoliť jeden z týchto stromov záleží od toho, čo budeme robiť
- pre pridávanie dát je výhodnejší red black strom, no naopak pre vyhľadávanie dát je výhodnejší avl strom
- daná implementácia tabuľky s lineárnym skúšaním by sa dala určite vylepšiť tak, že jej veľkosť bude prvočíslo a nie číslo, ktoré podhodí užívateľ
- týmto spôsobom vieme signifikatne redukovať počet kolízií, čo sme mohli vidieť pri tabuľke s reťazením, ktorej veľkosť bola vždy prvočíslo