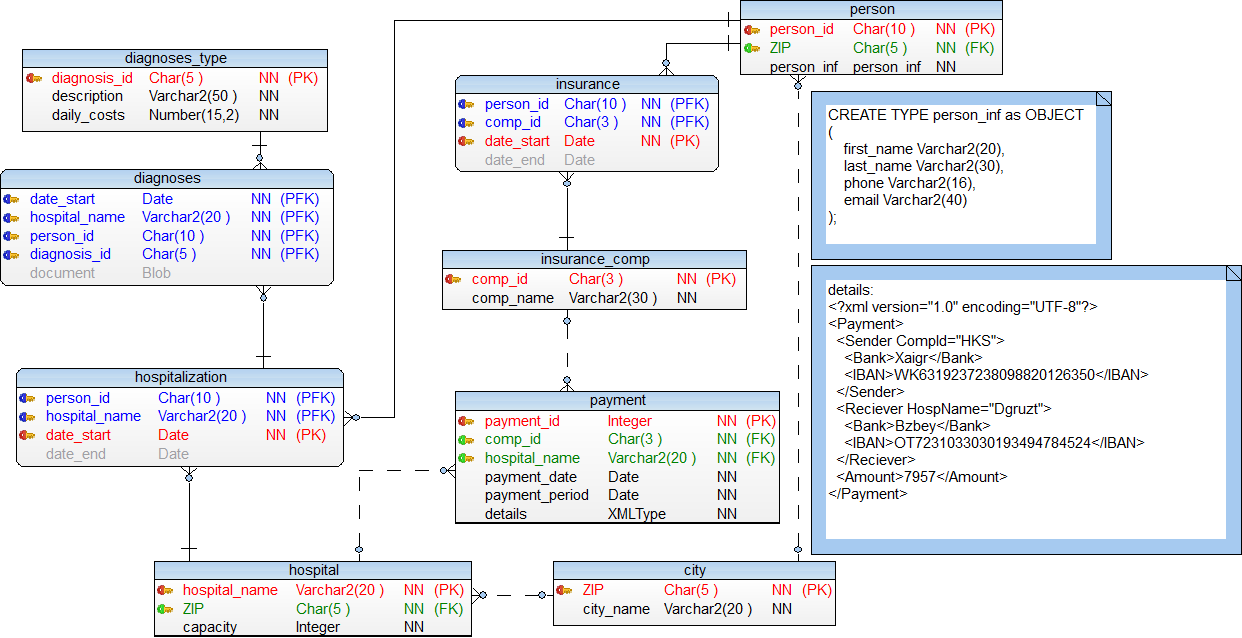
Fakulta riadenia a informatiky

**Semestrálna práca**

**Pokročilé Databázové Systémy**

1.semester Pavol Petrovič, 561613, 5ZIS12 2022/2023 Matej Mažgút, 561605, 5ZIS12

# Dátový Model



# Požadované súčasti SP

**Objektový Atribút Tabuľky**

V tabulke ***person*** sa nachádza objektový atribút ***person\_inf***. Tento objekt má nasledovnú štruktúru:

CREATE TYPE person\_inf as OBJECT

(

first\_name Varchar2(20),

last\_name Varchar2(30),

phone Varchar2(16),

email Varchar2(40)

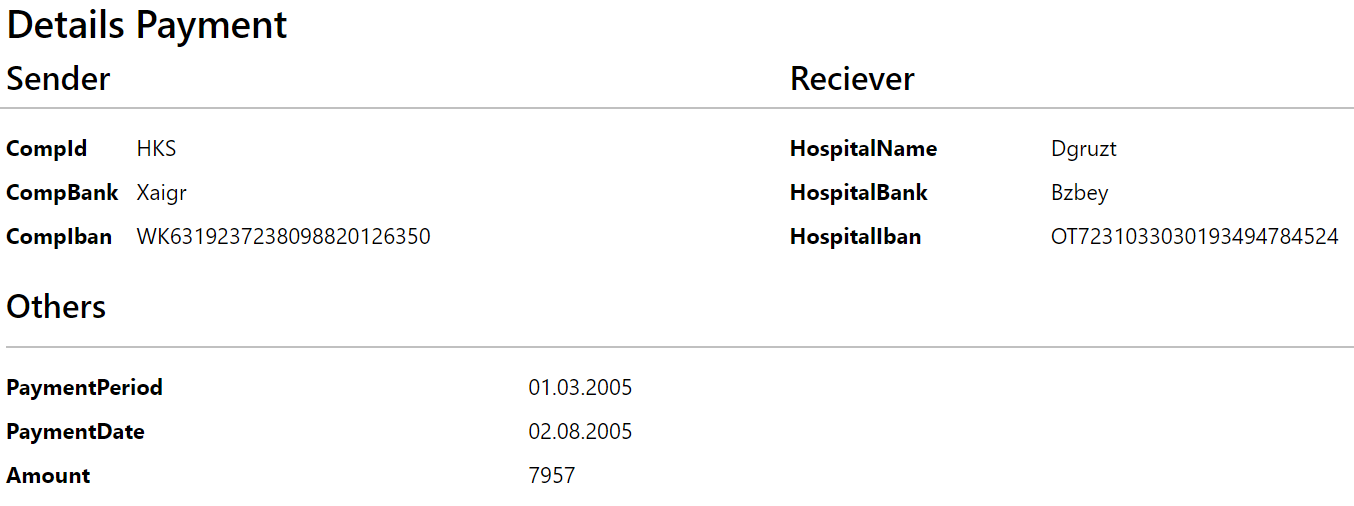
);

Pre výpis jednotlivých atribútov pri selectoch sme použili funkciu get\_person\_inf(char p\_id), ktorej výstupom bol varchar2, ktorý predstavoval cez pipe pospájané jednotlivé atribúty objektu.

Pri vytváraní záznamu do tabulky ***person*** sme použili okrem klasických atribútov aj konštruktor objektu ***person\_inf.***

**XML Report**

V tabulke **payment** sa nachádza atribút ***details*** typu **XMLType,** ktorý predstavuje detaily platby.



# Aplikácia EHealthCare

Aplikácia používa AppCore, čo predstavuje jadro aplikácie, ktoré zabezpečuje komunikáciu medzi aplikáciou a knižnicou HashFileLibrary. Aplikácia je implementovaná vo Windows Forms a ponúka veľmi intuitívne GUI, ktoré podporuje všetky požiadavky SP.

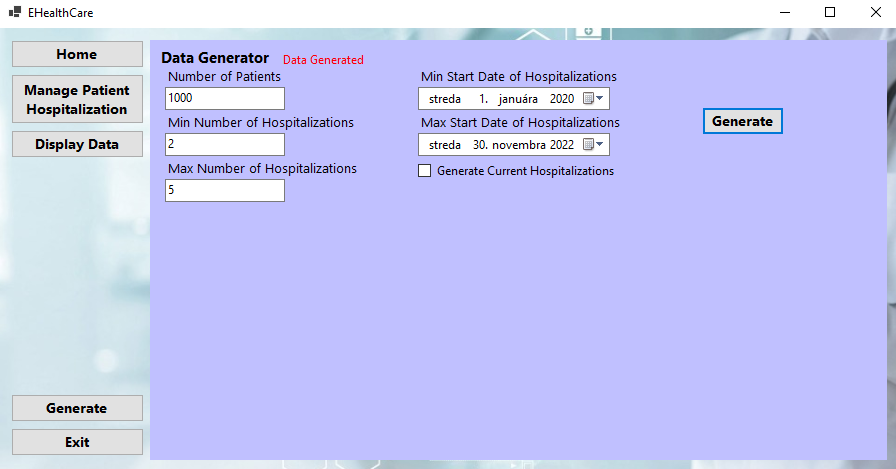
Po spustení Aplikácie je možné si vybrať s akým Hashovacím súborom chceme pracovať -Statický alebo Dynamický.



Následne je možné si vybrať či chceme pokračovať v poslednej uloženej verzií alebo chceme založiť novú, prázdnu databázu. V prípade že chceme založiť novú databázu musíme zadať Block Factor, s ktorým bude Hashovací súbor pracovať.



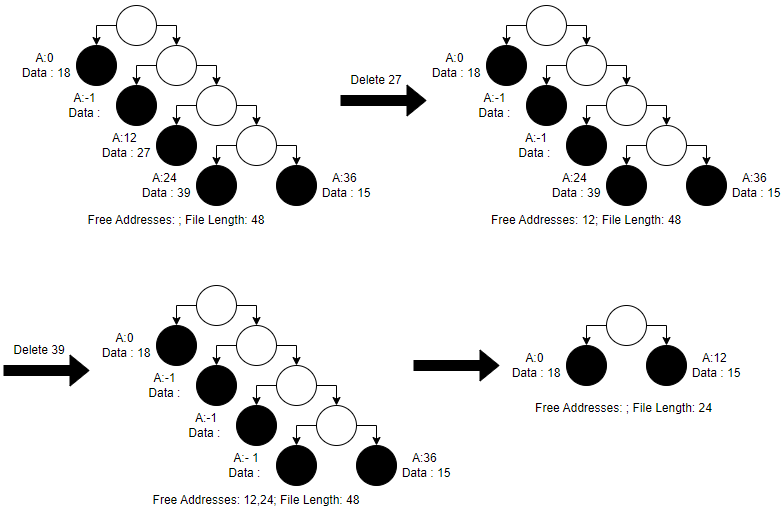
Aplikácie umožňuje generovanie dát pokiaľ je databáza prázdna. Ďalej umožňuje prácu s Pacientami a ich Hospitalizáciámi. O výsledku každej operácie je užívateľ oboznámený krátkym textom v hornej časti obrazovky. Pokiaľ užívateľ klikne na tlačidlo Exit alebo zatvorí aplikáciu aktuálny stav databázy aj s dátami sa uloží a v prípade znovuotvorenia aplikácia môže v práci pokračovať. Pokiaľ však raz začne s novou databázou k tej starej sa už nevráti.



# Manažment prázdnych blokov v Dynamickom Hashovacom súbore

Pokiaľ pri výmaze vznikne prázdny blok uprostred súboru vloží sa jeho adresa do poľa adries. Pokiaľ dochádza k zápisu bloku na novú adresu uprednostňuje sa výber z poľa adries. Vždy sa vyberá najmenšia adresa, ktorá sa následne z poľa vymaže. Ak je pole voľných adries prázdna, zapisuje sa na koniec súboru. Na konci každej operácie sa zavolá metóda manageAddress(), ktorá zabezpečí, že sa odstránia z poľa také adresy, ktoré sa rovnajú koncu súboru. Najzávažnejšia situácia, ktorá mohla nastať je znázornená na obrázku nižšie. Čierne vrcholy predstavujú externé a obsahujú adresu a na adrese sú zapísané dáta. Velkosť zapísaných dát, resp. veľkosť Block Factoru je 12.

Po tom ako sa odstránila hodnota 39, do poľa voľných adries sa zaradila adresa 24. Na základe charakteru štruktúry musí dôjsť k zlučovaniu externého vrcholu s Adresou 36 a Dátami 15. Keďže pole voľných adries nebolo prázdne vybrala sa odtiaľ hodnota 12 a novému externému vrcholu sme túto adresu nastavili. V Poli voľných adries ostala 24. Po zapísaní vrcholu s hodnotou 15, keďže predtým bola na adrese 36 a súbor bol veľký 48, skrátila sa veľkosť súboru na 36. Metóda manageAddress skontroluje pole voľných adries či 24(adresa) + 12(Block Factor) >= 36(Veľkosť súboru), ak rovnosť platí odstráni sa adresa z poľa a skráti sa veľkosť súboru. Táto kontrola prebieha cyklicky od najvyššieho vrcholu pokým dochádza k výmazu adresy a skráteniu súboru.



# Prístupy do súboru

PDS – Prístup do súboru

Budeme uvádzať najvyšší možný počet prístupu do súboru

1. Vyhľadanie záznamov pacienta (identifikovaný svojím rodným číslom). Po nájdení pacienta je potrebné zobraziť všetky evidované údaje vrátane všetkých hospitalizácií.

Dynamický hashovací súbor

* + - \*DFind Pacienta **PDS = 1**

Statický Hashovací súbor

* + - Nájdenie Pacienta **PDS = 1,** prečítanie zo súboru

1. Vyhľadanie hospitalizácie (definovaná id hospitalizácie) pre pacienta (definovaný rodným číslom) a zobrazenie všetkých údajov o nej. (Kľúč, ktorý pôjde do vyhľadávania nebude reťazec).

Dynamický hashovací súbor

* + - \*DFind Pacienta **PDS = 1**

Statický Hashovací súbor

* + - Nájdenie Pacienta **PDS = 1,** prečítanie zo súboru

1. Vykonanie záznamu o začiatku hospitalizácie pacienta (identifikovaný svojím rodným číslom).

Dynamický hashovací súbor

* + - \*DFind získanie Pacienta **PDS = 1**
    - Ak sa našiel pacient a podarí sa mu pridať záznam o Hospitalizácií tak \*DDelete Pacienta + \*DInsert Pacienta **PDS = j\*5 + (k+i)\*2**

Statický Hashovací súbor

* + - Nájdenie Pacienta **PDS = 1,** prečítanie zo súboru
    - Ak sa našiel pacient a podarí sa mu pridať záznam o Hospitalizácií tak výmaz Pacienta a pridanie Pacienta **PDS = 2 + 2, 2**\*prečítanie a zapísanie do súboru

1. Vykonanie záznamu o ukončení hospitalizácie pacienta (identifikovaný svojím rodným číslom).

Dynamický hashovací súbor

* + - \*DFind získanie Pacienta **PDS = 1**
    - Ak sa našiel pacient a podarí sa mu upraviť záznam o Hospitalizácií tak \*DDelete Pacienta + \*DInsert Pacienta **PDS = j\*5 + (k+i)\*2**

Statický Hashovací súbor

* + - Nájdenie Pacienta **PDS = 1,** prečítanie zo súboru
    - Ak sa našiel pacient a podarí sa mu upraviť záznam o Hospitalizácií tak výmaz Pacienta a pridanie Pacienta **PDS = 2 + 2, 2**\*prečítanie a zapísanie do súboru

1. Pridanie pacienta.

Dynamický hashovací súbor

* + - \*DInsert Pacienta **PDS = (k+i)\*2**

Statický Hashovací súbor

* + - Vloženie pacienta **PDS = 2,** prečítanie a zapísanie do súboru

1. Vymazanie hospitalizácie (definovaná id hospitalizácie) pre pacienta (definovaný rodným číslom).

Dynamický hashovací súbor

* + - \*DFind získanie Pacienta **PDS = 1**
    - Ak sa našiel pacient a podarí sa mu vymazať záznam o Hospitalizácií tak \*DDelete Pacienta + \*DInsert Pacienta **PDS = j\*5 + (k+i)\*2**

Statický Hashovací súbor

* + - Nájdenie Pacienta **PDS = 1,** prečítanie zo súboru
    - Ak sa našiel pacient a podarí sa mu vymazať záznam o Hospitalizácií tak výmaz Pacienta a pridanie Pacienta **PDS = 2 + 2, 2**\*prečítanie a zapísanie do súboru

1. Vymazanie pacienta.

Dynamický hashovací súbor

* + - \*DDelete Pacienta **PDS = j\*5**

Statický Hashovací súbor

* + - Vymazanie Pacienta **PDS = 2,** prečítanie a zapísanie do súboru

\*DFind - najvyšší počet **PDS = 1**

1. Ak vrchol kde by sa mal záznam nachádzať obsahuje záznamy **PDS = 1**, ak záznamy neobsahuje **PDS = 0**

\*DInsert - najvyšší počet **PDS = (k+i)\*2**

1. Vkladaný záznam sa pushne do pomocného Stacku
2. Ak vrchol kde by sa mal záznam vložiť má ešte miesto tak sa záznam popne z pomocného Stacku, prečítajú sa dáta zo súboru, zmenia a znova zapíšu **PDS = 1 + 1**
3. Ak je vrchol plný prečítajú sa dáta z neho **PDS = 1** 
   * + - * Pôvodný externý vrchol sa transformuje na nový interný vrcholu a na adresu starého externého sa zapíše prázdny blok **PDS = 1**
         * Novému internému vrcholu sa pridá nový ľavý externý a nový pravý externý vrchol
         * Záznamy zo starého externého vrcholu sa pushnú do pomocného Stacku
         * Ako spracovávaný vrchol sa nastaví nový interný vrchol a seekuje sa adresa prvého prvku z pomocného stacku v strome
         * Cyklus sa opakuje od bodu 1 pokiaľ pomocný Stack nie je prázdny
4. Celkové **PDS = 2 / PDS = (k+i)\*2**, pričom k je počet prvkov v pomocnom Stacku a i počet koľkokrát bol vrchol pri vkladaní plný

\*DDelete - najvyšší počet **PDS = j\*5**

1. Ak vrchol kde by sa mal záznam vymazať neobsahuje žiadne záznamy tak koniec mazania **PDS = 0**
2. Ak vrchol obsahuje záznamy, prečíta dáta zo súboru, **PDS = 1,** ak sa z bloku podarí vymazať dáta, dáta je potrebné do súboru ešte zapísať **PDS = 1**
3. Ak vo vrchole ostane 0 záznamov a je uprostred súboru zapíše sa tam prázdny blok **PDS = 1**
4. Ak počet záznamov vo vrchole spolu s bratovými záznamami je menší alebo rovný ako block factor, prečítajú sa bratove záznamy zo súboru **PDS = 1,** záznamy z brata aj z aktuálneho externého vrcholu sa pushnu do pomocného Stacku. Inak Koniec
5. Vytvoríme sa nový externý vrchol namiesto otca brata a aktuálneho vrcholu
6. Na adresu brata a aktuálneho vrcholu sa zapíše prázdny blok **PDS = 2**
7. Vytvorí sa nový blok, do ktorého sa zapíšu záznamy z pomocného Stacku, a ten sa zapíše na novú adresu do novo vytvoreného **PDS = 1**
8. Ako spracovávaný vrchol sa nastaví nový externý vrchol a opakuje sa cyklus od bodu 2
9. Celkové **PDS = 0 / PDS = 2 / PDS = 3 / PDS = j\*5**, pričom j je počet koľko krát mal vrchol spolu s bratovým vrcholom menší alebo rovný počet záznamov ako Block Factor

# Porovnanie Dynamického a Statického Hashovacieho súboru

Na porovnanie týchto dvoch prístupov hashovania do súboru som si zvolil tri testy.

V každom teste sa vykonalo dokopy 1 000 000 operácií. V jednotlivých stĺpcoch môžeme vidieť percentuálny podiel jednotlivých operácií. Výsledné časy predstavujú sčítanú dĺžku jednotlivých operácií v sekundách.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Insert** | **Delete** | **Find** | **Operations** |
| **test1** | 40% | 20% | 40% | 1 000 000 |
| **test2** | 40% | 20% | 40% | 1 000 000 |
| **test3** | 40% | 20% | 40% | 1 000 000 |
| **test4** | 40% | 20% | 40% | 1 000 000 |

Cieľom **test1** bolo aby na konci testu bolo v oboch súboroch približne rovnaký počet prvkov. Keďže v Statickom hashovacom súbore sme neriešili nijakým spôsobom kolízie, bolo potrebné nastaviť väčší Block Factor. Ako môžeme vidieť veľkosť súboru na konci testu je v dynamickom súbore väčšia. Pri Statickom Hashovani sme už mohli pozorovať vznik kolízia kvôli nedostatočne veľkému blokovaciemu faktoru. Časy operácií sú pri Dynamickom Hashovacom súbore omnoho nižšie ako pri Statickom Hashovacom súboru.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test1** | Insert | Delete | Find | Items At the End | File Length | Block Factor |
| Dynamic | 399 940 | 200 162 | 399 867 | 199 778 | 2 299 972 | 50 |
| Dynamic Times[s] | 33,6635364 | 7,4128973 | 7,9007287 |  |  |  |
| Static | 395 151 | 200 144 | 400 148 | 195 007 | 1 585 980 | 445 |
| Static Times[s] | 92,5205336 | 44,910718 | 53,3511689 |  |  |  |
| Static Colisions | 4 554 |  |  |  |  |  |

Cieľom **test2** bolo aby na konci testu bolo v oboch súboroch približne rovnaký počet prvkov a aby bol Block Factor rovnaký. Výsledne hodnoty, časy a aj počet kolízií pri Statickom Hashovaní bol rovnaký ako pri **test1.** Zmena Block Factoru v Dynamickom spôsobila, že dĺžka súboru bola o niečo kratšia. Rýchlosť operácie Delete A Find sa priblížili ku rýchlosti v Statickom Hashovaní. Operácia Insert v Dynamickom Hashovaní ďaleko presiahla rýchlosť v Statickom Hashovaní.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test2** | Insert | Delete | Find | Items At the End | File Length | Block Factor |
| Dynamic | 399 446 | 199 553 | 400 950 | 199 893 | 1 831 896 | 445 |
| Dynamic Times[s] | 175,9594431 | 41,0291234 | 46,9701599 |  |  |  |
| Static | 394 482 | 199 921 | 399 998 | 194 561 | 1 585 980 | 445 |
| Static Times[s] | 92,2415925 | 44,7188396 | 53,2450801 |  |  |  |
| Static Colisions | 5 598 |  |  |  |  |  |

Na nasledujúcom obrázku môžeme vidieť graf porovnania rýchlosti operácií Dynamického a Statického Hashovania.

Cieľom **test3** bolo aby mali obidva prístupy nastavený rovnaký Block Factor, no menší ako v **test2**. Keďže pri Statickom Hashovaní nastalo veľa kolízií, výsledné časy a hodnoty nemá zmysel brať za relevantné a ani neboli vo výslednom porovnávacom grafe. Čo sa dá porovnať je samotné Dynamické Hashovanie z **test1, test2** a **test3**, keďže pomer operácií bol rovnaký s rozdielnym Block Factorom.

* **test1 – 50**
* **test3 – 250**
* **test2 – 445**

Zmena Block Factoru mala vplyv na rýchlosť operácií ale aj na veľkosť súboru. Predpoklad je taký, že čím je väčší Block Factor, tým sú časy operácií vyššie, pretože do operačnej pamäte sa načítava viacero záznamov, ktoré musia byť spracované. A čím vyšší Block Factor, tým je veľkosť súboru menšia.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test3** | Insert | Delete | Find | Items At the End | File Length | Block Factor |
| Dynamic | 399 618 | 200 349 | 399 994 | 199 269 | 2 052 096 | 250 |
| Dynamic Times[s] | 108,6462879 | 24,8101082 | 27,8114431 |  |  |  |
| Static | 262 546 | 200 282 | 399 979 | 62 264 | 501 000 | 250 |
| Static Times[s] | 37,7128095 | 22,632904 | 28,0731299 |  |  |  |
| Static Colisions | 137 191 |  |  |  |  |  |

Na základe predpokladu z **test3** o Dynamickom Hashovacom súbore, som sa rozhodol spraviť ďalší **test4,** iba s Dynamickým Hashovaním, kde som Block Factor nastavil na 1. Operácie Insert a Find boli o kúsok rýchlejšie ako v **test1** kde bol Block Factor 50. Veľkosť súboru bola spomedzi **test1, test2, test3, tes4** najvyššia. Avšak pri takto malom blokovacom faktore nastal problém pri ukladaní stromu adries do súboru. Keďže externých vrcholov bolo strašne veľa táto akcia zabrala veľké množstvo času.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test4** | Insert | Delete | Find | Items At the End | File Length | Block Factor |
| Dynamic | 399 111 | 200 184 | 400 666 | 198 927 | 2 387 160 | 1 |
| Dynamic Times[s] | 17,2713951 | 9,7410667 | 2,8472542 |  |  |  |

Na nasledujúcom obrázku môžeme vidieť graf porovnania rýchlosti operácií a veľkosti súboru Dynamického Hashovania s odlišným Block Factorom.

**Záver**

Testy preukázali že rozdiel Statického a Dynamického Hashovania je závislosť na Block Factore. Statické Hashovanie je vysoko závislé na Block Factore, ktoré súboru nastaví nemennú dĺžku a od veľkosti Block Factoru, závisí koľko záznamov sa do súboru zmestí a koľko záznamov bude musieť byť prehľadávaných v operačnej pamäti v rámci jedného bloku, čo má vplyv na čas operácií.

Dynamické Hashovanie je schopné pracovať bez ohľadu na veľkosť Block Factoru. Ako sme mohli vidieť v **test1**, **test2**, **test3** a **test4** do súboru s Dynamickým Hashovaním sa vždy zmestil rovnaký počet prvkov z čoho vyplíva že veľkosť súboru je schopné sa dynamicky meniť. Avšak aj tu má Block Factor vplyv na rýchlosti vykonávania operácií a aj veľkosti súboru. Veľkosti súboru neboli až tak výrazné. Z môjho pohľadu je dobré nemať ani príliš malý a ani príliš veľký Block Factor pri Dynamickom Hashovaní.