**SEMESTRÁLNA PRÁCA 2**

**Algoritmy a údajové štruktúry 2**

**Michal Murín, 5ZIS12 2022/2023**

**Obsah**

[**1.** **Statický hešovací súbor** 3](#_Toc121173242)

[**1.1.** **Metóda Insert()** 3](#_Toc121173243)

[**1.2.** **Metóda Find()** 3](#_Toc121173244)

[**1.3.** **Metóda Delete()** 3](#_Toc121173245)

[**1.4.** **Metóda GetOffset()** 3](#_Toc121173246)

[**1.5.** **Metóda LoadStaticDataFromFile()** 3](#_Toc121173247)

[**1.6.** **Metóda ExportAppDataToFile()** 4](#_Toc121173248)

[2. **Dynamický hešovací súbor** 5](#_Toc121173249)

[**2.1.** **Metóda Insert()** 5](#_Toc121173250)

[**2.2.** **Metóda Find()** 5](#_Toc121173251)

[**2.3.** **Metóda Delete()** 5](#_Toc121173252)

[**2.4.** **Správa voľných blokov** 6](#_Toc121173253)

[**2.5.** **Metóda GetOffset()** 6](#_Toc121173254)

[**2.6.** **Metóda LoadDynamicDataFromFile()** 6](#_Toc121173255)

[**2.7.** **Metóda ExportAppDataToFile()** 6](#_Toc121173256)

[**3.** **Trieda Hashing** 7](#_Toc121173257)

[**4.** **Porovnanie výkonnosti** 8](#_Toc121173258)

[**5.** **Aplikácia „Zdravotná karta“** 10](#_Toc121173261)

[**5.1.** **Návrh aplikácie** 10](#_Toc121173262)

[**5.2.** **Grafické rozhranie** 10](#_Toc121173263)

[**Prílohy:** 11](#_Toc121173264)

[**1.** **UML Diagram Projektu AUS2\_MurinMichal\_HashFile** 11](#_Toc121173265)

[**2.** **UML Diagram projektu GUI:** 12](#_Toc121173266)

1. **Statický hešovací súbor**

Statický hešovací súbor má parametrický konštruktor, s parametrami ako názov súboru, blok faktor a počet blokov, ktoré sa majú alokovať. Keďže sa jedná o statický súbor, na základe vstupných parametrov sa na začiatku životného cyklu alokuje súbor fixnej veľkosti s prázdnymi dátami. Následne je možné nad súborom vykonávať základné operácie.

* 1. **Metóda Insert()**

Metóda Insert využíva pomocnú metódu FindBlock(), ktorá sa nachádza v predkovi triedy StaticHashing. Táto metóda vracia blok načítaný zo súboru a adresu na akom sa nachádzal (ak sa našiel). Metóda Insert ďalej do nájdeného bloku vloží dáta, a ak bolo vkladanie úspešné, zapíše nový blok naspäť do súboru. Počet prístupov k súboru je preto 2 v prípade úspešného vkladania dát do bloku. Inak 1.

* 1. **Metóda Find()**

Metóda Find() rovnako využíva pomocnú metódu z predka, v tomto prípade ale nezapisuje žiadne dáta späť, iba prebehne cyklom načítaný blok a pokúsi sa nájsť hľadané dáta, ktoré v prípade úspechu vráti. Počet prístupov do súboru je preto 1.

* 1. **Metóda Delete()**

Metóda Delete funguje na podobnom princípe ako metóda Insert(). Pomocou pomocnej metódy sa pokúsi nájsť hľadaný blok, následne v ňom vymaže záznam a ten v prípade úspechu zapíše naspäť do súboru. Počet prístupov do súboru je preto 1-2.

* 1. **Metóda GetOffset()**

Metóda GetOffset() vypočíta adresu bloku v pamäti, kde by sa mali hľadané dáta nachádzať. Robí tak na základe prvých 8 bajtov hashu hľadaných dát a počtu blokov v súbore, ktorý je pevne daný.

* 1. **Metóda LoadStaticDataFromFile()**

Metóda načíta aplikačné dáta z csv súboru tak, aby mohla po opätovnom spustení jednoducho pokračovať.

* 1. **Metóda ExportAppDataToFile()**

Metóda uloží aplikačné dáta do csv súboru.

1. **Dynamický hešovací súbor**

Dynamický hešovací súbor dokáže na základe počtu dát dynamicky meniť svoju veľkosť. Využíva pri tom internú štruktúru trie, na ukladanie dát o adresách konkrétnych blokov v súbore.

* 1. **Metóda Insert()**

Pri vkladaní dát do súboru sa najskôr nájde prislúchajúci externý vrchol, z ktorého získame adresu kde by mali byť dáta uložené, ak tomuto bloku adresa priradená nie je, priradíme jej adresu zo zoznamu voľných adries alebo z konca súboru. Následne zistíme či sa dáta do bloku zmestia, ak nie, musíme prehešovať dáta z aktuálneho bloku a vytvoriť nový blok v súbore. Na toto slúži metóda TryRehashAndInsertData(), ktorá cyklicky prehešuje dáta a pokúsi sa ich rozdeliť medzi dva externé vrcholy, ktoré vznikli rozdelením pôvodného vrcholu. Ak sa to nepodarí, plný vrchol sa delí cyklicky ďalej. Po úspešnom prehešovaní sa oba bloky zapíšu na svoje miesto v súbore.

Ak blok plný nie je, dáta sa do súboru vkladajú podobne ako to bolo pri statickom súbore.

Počet prístupov preto môže byť 1-2 pri možnosti, že sa dáta do bloku vložili, a 3 ak sa blok musel prehešovať.

* 1. **Metóda Find()**

Metóda Find() je identická s metódou Find() pri statickom súbore, preto je telo tejto metódy definované v predkovi – v triede Hashing.

* 1. **Metóda Delete()**

Metóda Delete() nájde v znakovom strome vrchol, kde by sa mali dáta nachádzať a načíta blok do pamäte. Následne sa pokúsi vymazať záznam z bloku. Ak sa to podarilo, metóda sa pokúsi zistiť, či sa vrchol, z ktorého sa práve vymazali dáta, nemôže zlúčiť s jeho bratom. Ak je to možné, Vrchol sa zlúči s jeho bratom a prázdny blok sa buď vloží do zoznamu prázdnych adries a prázdny sa zapíše naspäť do súboru, alebo, ak sa nachádza na konci súboru, sa úplne odstráni. Zlučovanie prebieha v metóde MergeBlocks(). Celý tento cyklus sa opakuje pokiaľ je možné bloky zlučovať. Ak to možné nie je, zapíše sa do súboru aj Blok v ktorom ostali valídne dáta. Počet prístupov do súboru sa preto môže pohybovať od 1, kedy odstraňovanie záznamu neuspeje, až po počet vrstiev v strome + 1.

* 1. **Správa voľných blokov**

Na manažment voľných blokov moja aplikácia využíva dynamický zoznam, v ktorom uchováva adresy prázdnych blokov v pamäti. Metóda HandleEmptyBlocks() zabezpečuje to, aby sa prázdne bloky na konci súboru mazali, a ukladá adresy prázdnych blokov, ktoré sa v súbore stále nachádzajú. Metóda AssignOffsetToNode() priraďuje voľnú adresu externému vrcholu, ktorý ju potrebuje.

* 1. **Metóda GetOffset()**

Táto metóda vracia adresu bloku, na ktorej by sa hľadané dáta mali nachádzať. Začína vo vyhľadávaním v znakovom strome, z ktorého pomocou hashu dát nájde externý vrchol, ktorý si udržiava aj hľadanú adresu.

* 1. **Metóda LoadDynamicDataFromFile()**

Metóda, ktorá načíta dáta potrebné na rozbehanie programu, po jeho vypnutí – načítava dáta pre znakový strom a pre zoznam voľných adries.

* 1. **Metóda ExportAppDataToFile()**

Metóda, ktorá uloží dáta pre znakový strom a pre zoznam prázdnych blokov do csv súboru.

1. **Trieda Hashing**

Trieda Hashing je predok tried DynamicHashing a StaticHashing. Sú v nej definované metódy, ktoré majú jej potomkovia spoločné, a implementované identické metódy, ako napríklad metóda FindBlock(), Find() UpdateData(), ktorá slúži na aktualizovanie dát v súbore. Ďalej sa tu nachádzajú metódy na prístup k súboru, na ukladanie spoločných aplikačných dát, alebo na sekvenčný výpis súboru.

1. **Porovnanie výkonnosti**

Pre porovnanie výkonnosti oboch štruktúr som si vygeneroval 1 000 000 náhodných pacientov, ktorých som následne vkladal do jednotlivých štruktúr a zaznamenával čas.

Test som vykonával pri blok faktore 5 a 100, na úvod testu sa štruktúry naplnili 200 000 prvkami, a následne sa na štruktúrach náhodne vykonávalo 100 000 operácií.

Výsledky je možné vidieť na grafoch nižšie.

Z prvého grafu vyplýva, že celkový čas trvania operácií bol menší, nakoľko všetky operácie prehľadávali pri každom bloku maximálne 5 záznamov, pri druhom grafe to mohlo byť až 100, čo operácie spomalilo. Pri malom blokovacom faktore sa v dynamickom súbore vytvoril veľmi veľký znakový strom, čo spôsobilo jeho spomalenie, keďže pri každej operácii v ňom musel nájsť príslušný vrchol. Naviac pri vkladaní aj mazaní dát musel často prehešovať dáta a meniť štruktúru znakového stromu.

Pri väčšom blokovacom faktore sa vytvoril menší znakový strom, vďaka čomu zvládol dynamický súbor všetky operácie o niečo rýchlejšie.

Aj napriek rozdielom pri prvom grafe, priemerná rýchlosť operácií pri oboch štruktúrach je veľmi nízka, keďže najpomalšia operácia jemne presahovala hranicu 200 mikro sekúnd. Vzhľadom na omnoho lepšie praktické využitie dynamického súboru je však tento rozdiel zanedbateľný, nakoľko čo sa týka pamäťovej náročnosti je dynamický súbor určite lepšia voľba. To platí aj pri možnosti ľubovoľne meniť počet dát a veľkosť súboru, čo pri statickom súbore nie je možné.

1. **Aplikácia „Zdravotná karta“**
   1. **Návrh aplikácie**

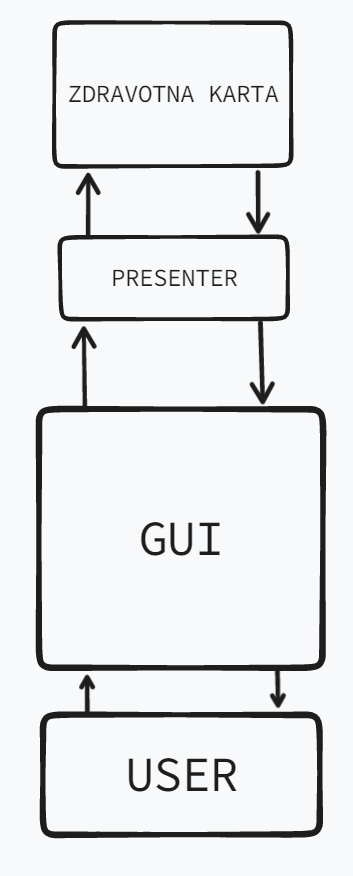
Jadro aplikácie sa nachádza v triede HealthCard. V tejto triede sa udržiavá inštancia triedy Hashing, typ hešovania si užívateľ vyberie pri spustení aplikácie. Ďalej sa tu nachádzajú všetky metódy potrebné na funkcionalitu popísanú v zadaní semestrálnej práce.

Trieda Patient v sebe uchováva základné údaje o pacientovi, plus pole desiatich hospitalizácií.

Trieda Hospitalization v sebe uchováva základne atribúty hospitalizácie.

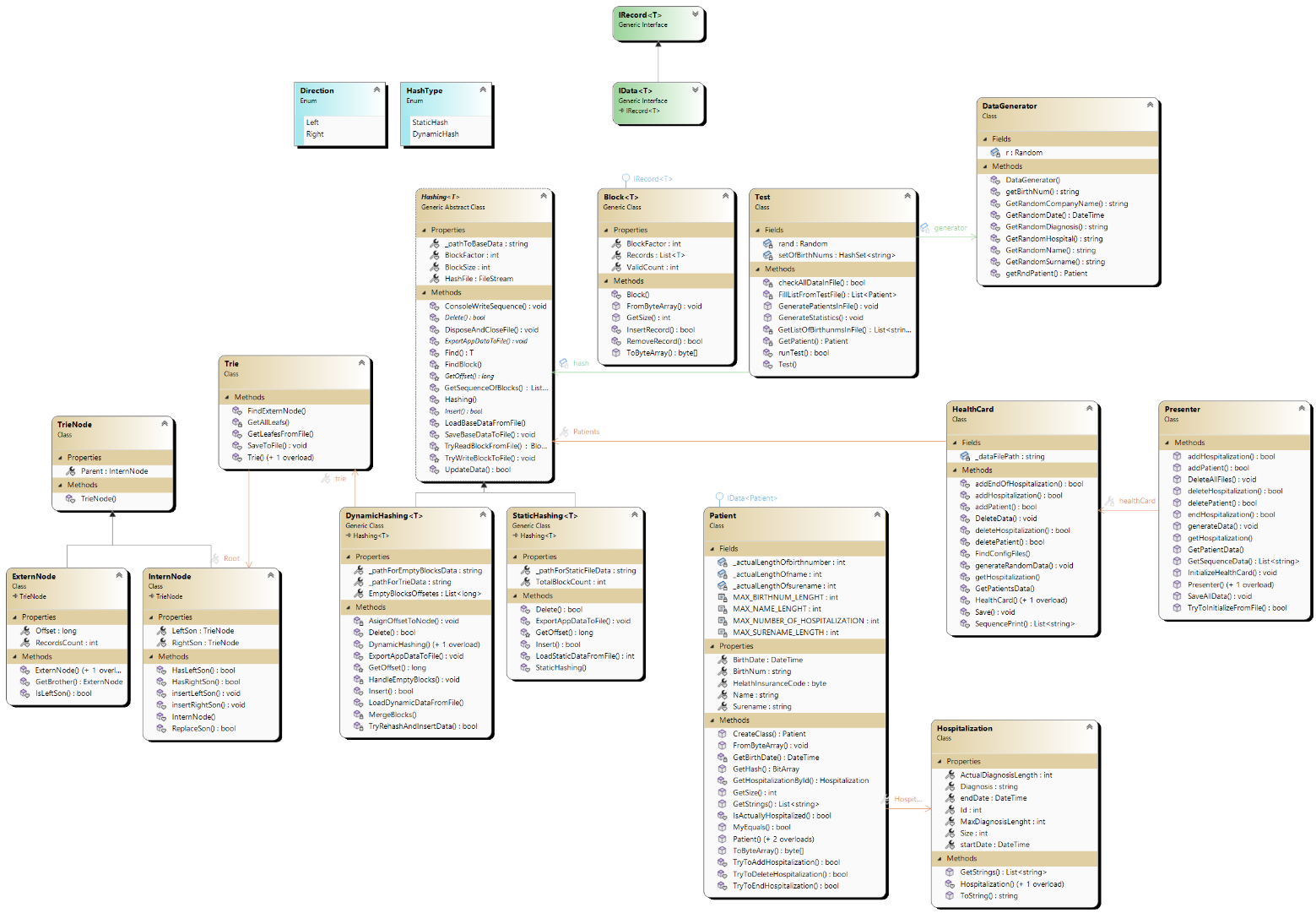
UML diagram celej aplikácie je v prílohe.

* 1. **Grafické rozhranie**

Na obrázku je možné vidieť návrh používateľského rozhrania. Používateľ komunikuje prostredníctvom grafického okna, ktoré udržuje inštanciu presentera. Prostredníctvom objektu presentera sa následne dopytuje aplikácie zdravotnej karty na údaje, ktoré používateľovi zobrazí.

Celá práca je rozdelená na dva projekty, jeden pre aplikáciu a druhý pre grafické rozhranie. V projekte GUI, v triede MainWindowView, sa vytvorí inštancia triedy Presenter, ktorým projekt komunikuje s aplikáciou HealthCard, ktorá je uzavretá v druhom projekte AUS2\_MurinMichal\_HashFile.

# **Prílohy:**

1. **UML Diagram Projektu AUS2\_MurinMichal\_HashFile**
2. **UML Diagram projektu GUI:**