Žilinská Univerzita
Fakulta riadenia a informatiky

Semestrálna práca 2

Policajná databáza

B Strom

Školský rok 2019/2020

Dávid Pavličko

4. ročník

Skupina 5ZZS12

Obsah

Úvod

Cieľom semestrálnej práce je vytvorenie informačného systému na evidenciu áut a vodičských preukazov v binárnych súboroch a efektívne s nimi pracovať a vykonávať operácie vyhľadaj, vlož, uprav a zmaž pomocou použitia vhodných údajových štruktúr:

- Lineárne hešovanie
- B Strom
- Heap file

1. Analýza

Aplikácia musí spĺňať kritérium fungovania na systéme s malou operačnou pamäťou, teda musí vedieť pracovať s veľkou databázou, ktorá bude uložená v binárnom súbore, alebo viacerých binárnych súboroch a vedieť efektívne vyhľadávať, upravovať, pridávať a mazať akékoľvek typy záznamov.
V konkrétnom zadaní sa jedna o vodičské preukazy a autá. Autá môžeme identifikovať dvomi spôsobmi:

1. Unikátny VIN reťazec
2. Unikátna značka vozidla – EČV

Jednotlivé vodičské preukazy môžeme identifikovať nasledovne:

1. Unikátne číslo osoby

Aby bolo možné efektívne zrealizovať túto aplikáciu, je nutné využiť optimálne údajové štruktúry. Je viac spôsobov. Autorom vybrané riešenie bolo tzv. Lineárne hešovanie, alebo B strom a taktiež použitie neusporiadaného súboru Heap file.

Návrh údajovej štruktúry

Na ukladanie dát v tejto semestrálnej práci som použil štruktúru B strom a Heap file. Všetky dáta sú ukladané v binárnych súboroch vrátane meta dát blokov ako aj samotnej aplikácie pre znovu načítanie pôvodných dát.

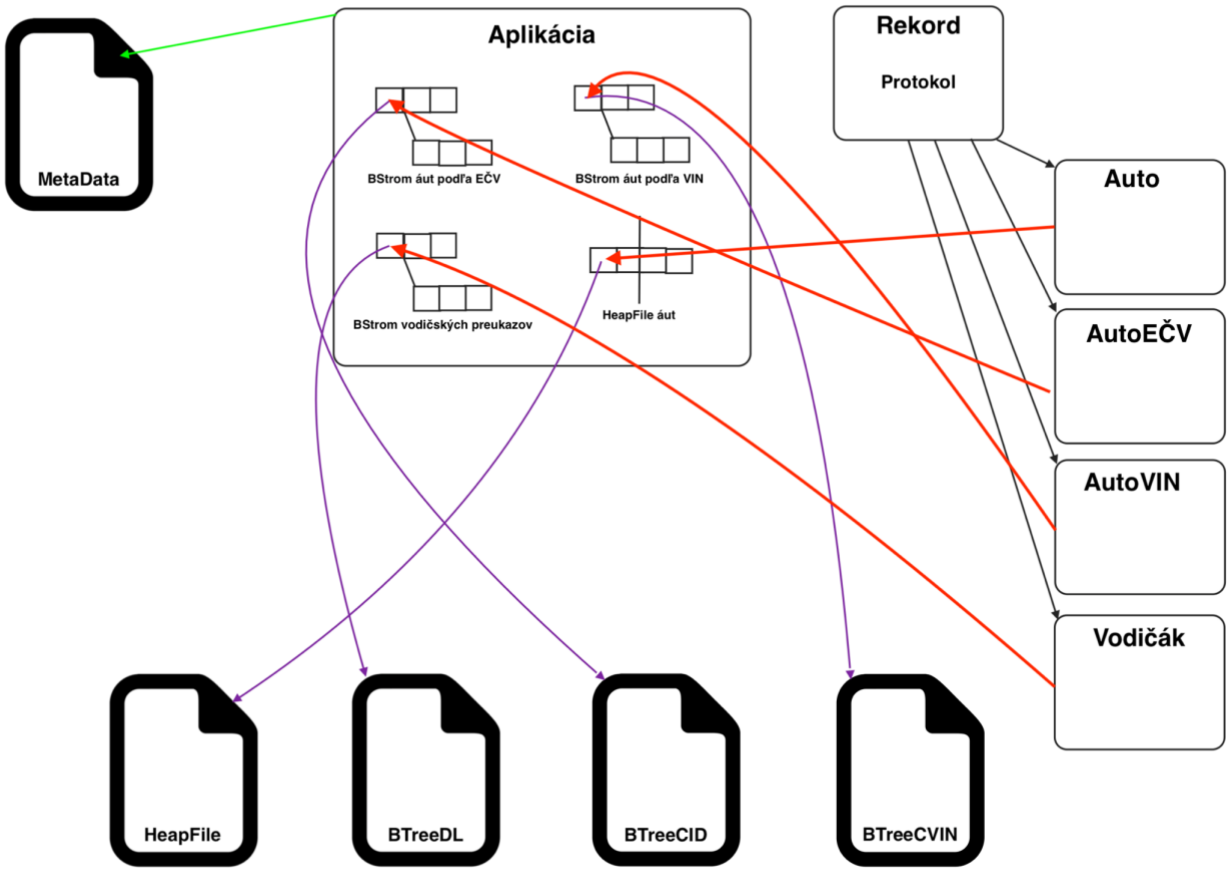
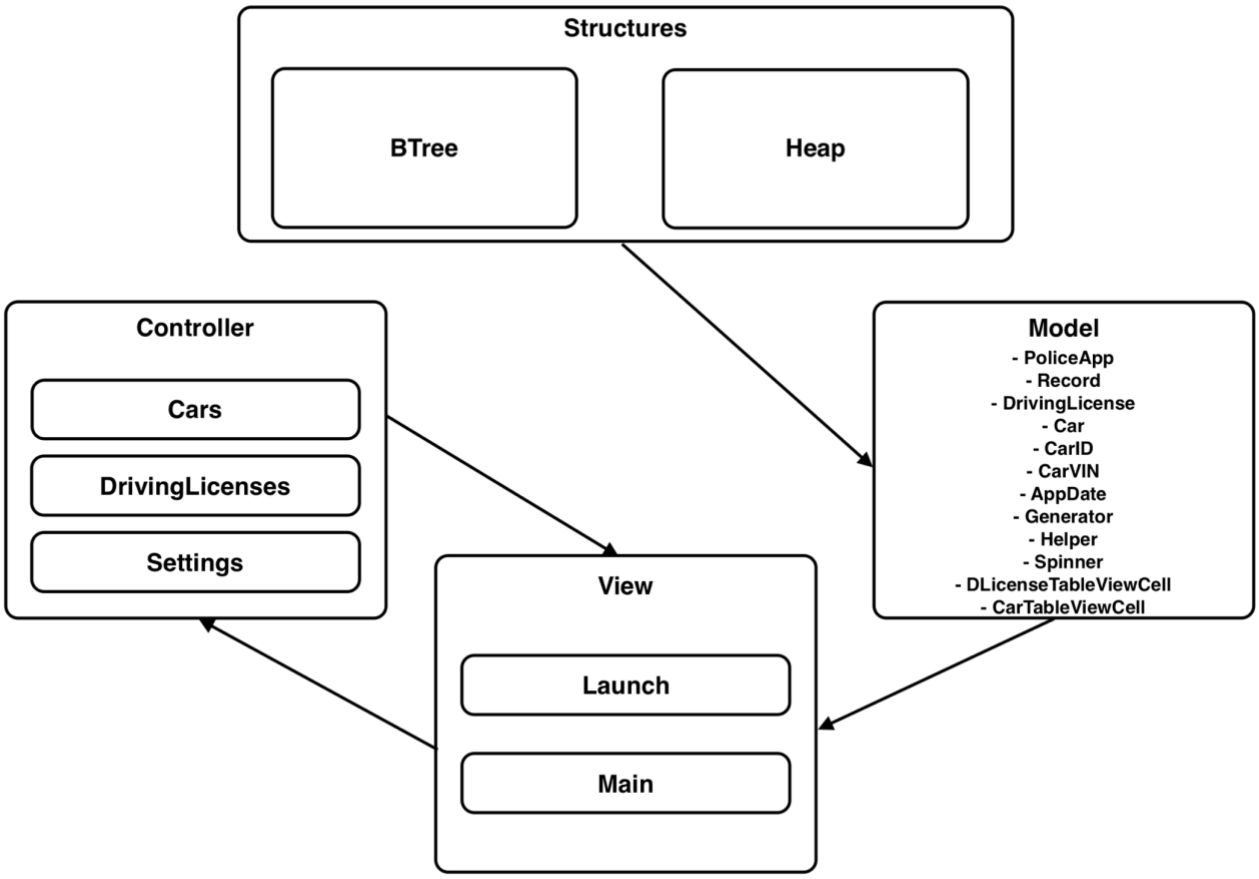


Diagram balíčkov



Popis diagramu balíčkov a tried

1. Structures

Balíček štruktúr obsahuje vlastnú implementáciu údajových štruktúr, ktoré boli použité vrámci semestrálnej práce a na testovanie.

- **B Strom**
- **Heap file**

3. Model

Triedy objektov a pomocné modely pre grafické zobrazenie sú uložené v tomto balíčku.

- **PoliceApp –** hlavná trieda, ktorá sa ihneď inicializuje po spustení aplikácie, ak už existujú nejaké binárne súbory, aplikácia bude ihneď s nimi pracovať. Táto trieda je zdieľaná a je možné do nej zasahovať vo viacerých častiach aplikácie.
- **Record –** protokol, ktorý sa správa ako interface, jeho triedy využívajú funckie ako initEmpty() pre zapisovanie „prázdnych“ inštancií do binárnych súborov, toBytes() prevedie atribúty triedy na binárne dáta, fromBytes() prevedie bajty opäť na inštanciu triedy, getSize() vracia veľkosť triedy v bajtoch potrebných na zápis a čítanie

zo súboru, toString() vypisuje aktuálny stav triedy do reťazca, isEmpty() overuje, či je inštancia triedy naplnená „prázdnymi“ dátami a initRandom() pre účely generátora

- **DrivingLicense –** jeden z dvoch agentov systému, s ktorými budeme pracovať a zobrazovať jeho stav koncovému používateľov mobilnej aplikácie
- **Car –** druhý z agentov systému, s ktorým budeme pracovať a zobrazovať jeho stav koncovému používateľovi mobilnej aplikácie
- **CarID –** trieda, ktorá drží len ID (EČV) auta a adresu v binárnom súbore pre heap file
- **CARVIN –** trieda, ktorá drží len VIN číslo auta a adresu v binárnom súbore pre heap file
- **AppDate –** vlastná trieda pre dátum vo formáte dd-mm-YYYY
- **Generátor –** Trieda implementujúca vlastnú implementáciu štruktúr za účelom vygenerovania náhodných inštancií áut a vodičských preukazov pre naplnenie databázy
- **Helper –** trieda, ktorá je zdieľaná, drží konštanty pre maximálne veľkosti reťazcov, prevodník čísel do binárneho stavu a naspäť, uchováva veľkosti jednotlivých dátových typov v bajtoch a prevádza boolean na číslo a naopak
- **Spinner –** táto trieda sa používa pri dlhých procesoch, ktoré sú rozdelené na hlavné vlákno, kde sa zavolá Spinner a na ďalšie vlákno na pozadí, ktoré spracúva operácie bez zamrznutia aplikácie
- **DLicenseTableViewCell –** pomocný model pre zobrazenie buniek v dynamickej tabuľke vodičských preukazov načítaných zo súboru pre koncového používateľa mobilnej aplikácie
- **CarTableViewCell –** pomocný model pre zobrazenie buniek v dynamickej tabuľke áut načítaných zo súboru pre koncového používateľa mobilnej aplikácie

Zoznam požadovaných operácií a ich počet prístupov do súboru

1. Pre záznamy o aute

1. Vyhľadávanie

Podľa EČV

Vyhľadanie auta podľa evidenčného čísla v B strome áut podľa evidenčných čísel aplikácia pristupuje do binárneho súboru log(A) krát, kde A je počet blokov b stromu. Jednotlivé bloky sa vždy načítajú celé, pomocou komparátora prejdú záznamy a prípadne pokračuje ďalším načítaním bloku ako syna záznamu. Nájdený záznam vracia adresu rekordu do neusporiadaného heap filu, kde si už pomocou jedného prístupu získa daný rekord.

Počet prístupov: Log(A) + 1

Podľa VIN

Vyhľadanie auta podľa VIN čísla v B strome áut podľa VIN čísel aplikácia pristupuje do binárneho súboru log(B) krát, kde B je počet blokov b stromu. Jednotlivé bloky sa vždy načítajú celé, pomocou komparátora prejdú záznamy a prípadne pokračuje ďalším načítaním bloku ako syna záznamu. Nájdený záznam vracia adresu rekordu do neusporiadaného heap filu, kde si už pomocou jedného prístupu získa daný rekord.

Počet prístupov: Log(B) + 1

2 Pridanie

Najskôr je potrebné veriť, či sa evidenčné čislo alebo VIN už nenachádza v databáze, prebehne teda vyhľadávanie podľa evidenčného čísla a VIN čísla (log(A) + log(B)), ak sa také auto nenájde začne sa operácia vkladania. Najskôr sa auto vloží do neusporiadaného Heap file, pomocou jediného prístupu, kde už aplikácia vie, kde sa nachádza posledný záznam a vloží ho doň a vráti adresu, kde sa auto nachádza. Následne sa vloží do oboch B stromov podľa VIN aj EČV. Vkladanie prebehne podobne ako pri vyhľadávaní s tým rozdielom, že sa ukladá do pamäti RAM cesta navštívených blokov až do bloku kam sa nový rekord má vložiť. Počet prístupov bude log(A) + log(B) pre prvý aj druhý b strom.

Ak nastane porušenie vlastnosti štruktúry je potrebné vykonať rozdelenie a preusporiadať štruktúru pomocou tzv. Splitov. Počet splitov nemôže byť väčší ako hĺbka stromu a split musí zapísať rekordy do nového pravého bloku, redukovaný aktuálny blok a do bloku otca, alebo nového koreňového bloku, teda počet prístupov do súboru bude S * 3, kde S je počet splitov Sa pre B strom podľa EČV a Sb pre B strom podľa VIN.

Počet prístupov: Log(A) + Log(B) + 1 + (Log(A) + Sa * 3) + (Log(B) + Sb * 3)

3. Upravovanie

V aplikácií je potrebné najskôr vyhľadať auto podľa EČV (Log(A) + 1), alebo podľa VIN (Log(B) + 1) a následne po zmenách parametrov auta pomocou jediného prístupu na adresu v heap file prepíšem presný počet vyhradených bajtov pre rekord na jeho adrese.

Počet prístupov: Log(A) alebo Log(B) + 2

2. Pre záznamy o vodičských preukazoch

1. Vyhľadávanie

Vodičské preukazy sú identifikované podľa unikátneho celého čísla a nie sú ukladané v Heap file, ale iba v B strome. Takže počet prístupov sa zníži o 1 oproti vyhľadávaniu pre autá, ktoré sú uložené v Heap file. B strom vodičských preukazov teda pristupuje do súboru iba ak potrebuje načítať ďalší blok v ktorom by sa mal nachádzať daný rekord až kým nepríde do listového bloku, počet blokov si označíme N.

Počet prístupov: Log(N)

2. Pridanie

B strom vodičských preukazov sa prehľadáva podobne ako pri vyhľadávaní až kým sa rekord nenájde a teda nový sa nevloží, alebo dorazí do listového bloku do ktorého sa záznam vloží, počet operácií bude Log(N), kde N je počet blokov v B strome. Ak po vložení nevyhovuje usporiadanie a je porušená vlastnosť štruktúry o veľkosti bloku nastáva splitovanie. Počet splitov nemôže byť väčší ako je hĺbka stromu, takže postupným splitovaním a prepisovaním aktuálneho bloku do súboru, nového pravého bloku a prepisovanie otcovského bloku sa operácia opakuje až kým nie sú splnené všetky vlastnosti štruktúry. Každá operácia split (S) potrebuje 3 prístupy do súboru. Spätná cesta počas vkladania sa ukladá do pamäte RAM.

Počet prístupov: Log(N) + 3 * S

3. Upravovanie

Záznam je potrebné nájsť so zložitosťou Log(N), kde N je počet blokov binárneho stromu vodičských preukazov. Po úprave vodičského preukazu je znova vykonané hľadanie, ale príslušného bloku, počet prístupov bude ďalších Log(N), po nájdení sa opäť celý zapíše do súboru a prepíše zmenený záznam.

Počet prístupov: 2 * Log(N) + 1