TOP-K THRESHOLD ALGORITMUS

Vyhledávání top-k treshold výsledků v DB libovolných produktů s možností volby agregační funkce + GUI. Zdrojový kód: HTTPS://GITHUB.COM/MSKL/FIT-BI-VMW-PROJECT

POPIS PROJEKTU

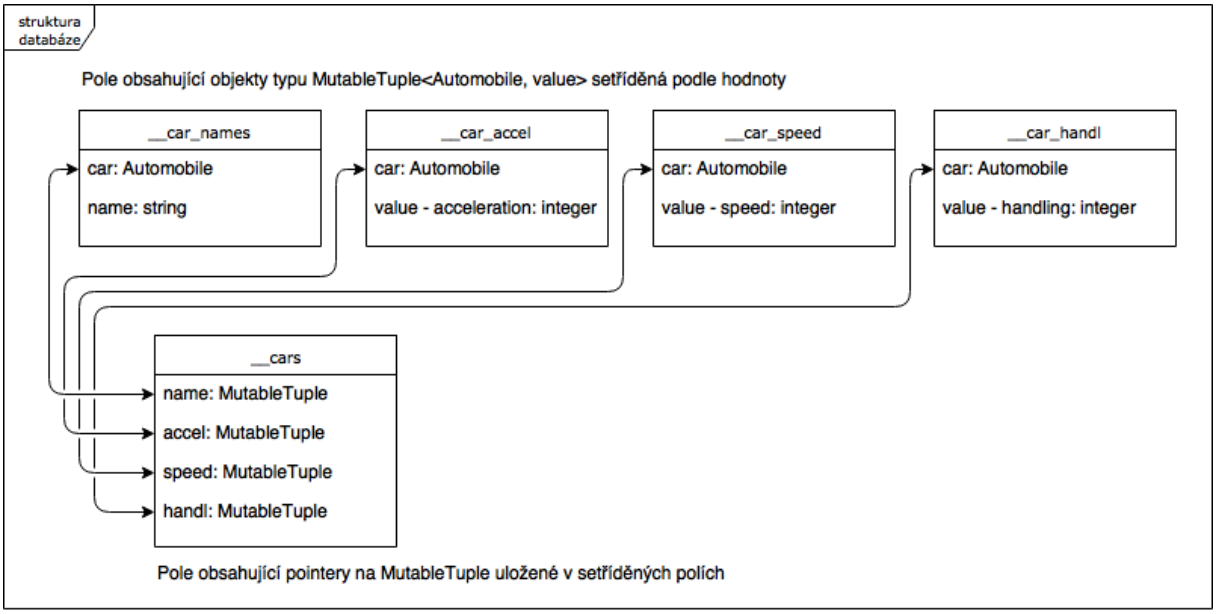
Implementoval jsem jednoduchu webovou aplikaci pro demonstraci použití top-k treshold algoritmu v jazyce Python. Mé řešení obsahuje navíc i naivní algoritmus a faginův algoritmus. Je tak možné přehledně vyzkoušet, jak jednotlivé algoritmy fungují a porovnat jejich efektivitu.

ZPŮSOB ŘEŠENÍ

Jako databázi využívám čtyři pole obsahující KeyValuePair ve tvaru <pointer na třídu Automobile, numerická hodnota>. Všechna data jsou nahrána přímo v paměti počítače. Jednotlivá **pole jseou setříděna podle hodnoty při načtení dat do databáze**. Jednotlivé dotazy pořadí prvků v polích nemění.

Implementace algoritmů pro top-k query se nachází v souboru /source/classes/CarDatabase.py.

Následující obrázek popisuje hlavní strukturu uložení dat (jednotlivá setříděná pole):



Samotný top-k treshold algoritmus využívá **minimovou haldu** k uchovávání objektů a efektivní odstraňování prvků s nejnižším agregačním score. Mé řešení obsahuje **několik agregačních funkcí**.

TOP-K TRESHOLD ALGORITMUS FUNGUJE NÁSLEDUJÍCÍM ZPŮSOBEM:

1. Nastavím treshold t jako agregační hodnotu prvků zjištěných v přístupu do databáze.
2. Náhodným přístupem vypočítám skóre nalezených objektů dle agregační funkce.
3. V minimové haldě si uchovávám seznam top-k objektů na které jsem doposud narazil.
4. Zastavím se, pokud jsou skóre top-k objeků v haldě větší nebo rovny tresholdu.

Konec – vrátím top-k objektů které jsem prozkoumal.

IMPLEMENTACE

Aplikace je napsaná v programovacím jazyce Python s využitím webového frameworku FLASK pro generování webové stránky. Samotná webová stránka je HTML dokument s několika funkcemi psanými v jazyce JavaScript (kontrolva vstupu, animace,..).

SPUŠTĚNÍ APLIKACE

Pro správný běh aplikace je potřeba mít nainstalovaný jazyk python3 a modul flask. Instalaci modulu je možné provést například pomocí příkazu:

```
pip3 install flask
```

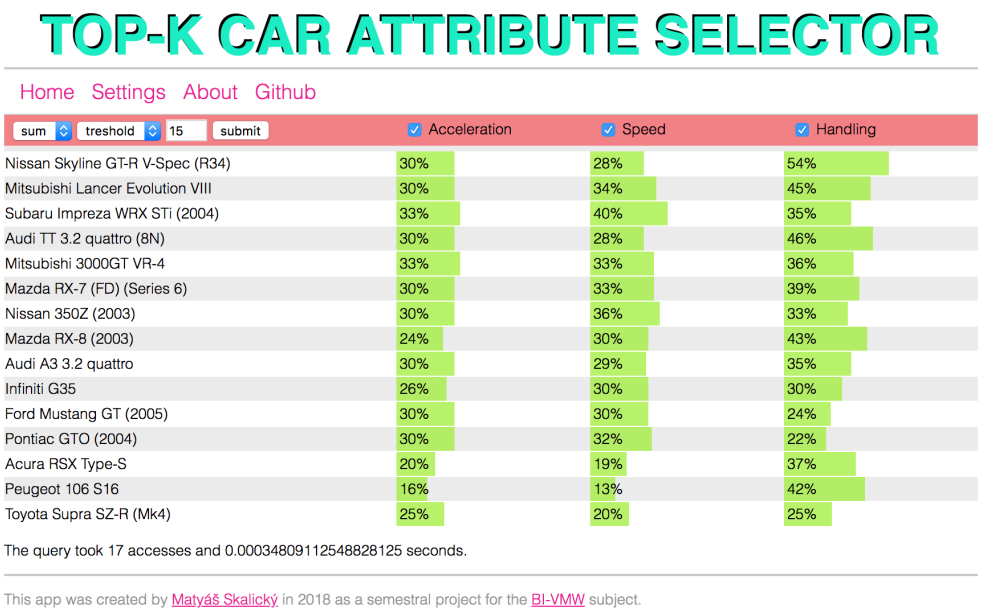
Samotná aplikace se spouští příkazem:

```
python3 run.py
```

Webový server běží ve výchozí konfiguraci na adrese http://127.0.0.1:5000/.

PŘÍKLAD VÝSTUPU

Na screenshotu vidíme ukázkový dotaz nad databází aut ze hry Need for Speed: Most Wanted (pozn. data lze změnit v záložce Settings). Na query byla využita jako agregační funkce suma a byl aplikován top-k treshold algoritmus. Uživatel si v tomto případě přál získat 15 objektů a setřídit je podle všech 3 parametrů (akcelerace, rychlost, ovládání) s pomocí agregační funkce suma. Pod vypsanou tabulkou přehledně vidíme, že query trvala 0.0003s a využila celkem 17 přístupů do databáze.



TOP-K CAR ATTRIBUTE SELECTOR

Home Settings About Github

sum treshold 15 submit

	Acceleration	Speed	Handling
Nissan Skyline GT-R V-Spec (R34)	30%	28%	54%
Mitsubishi Lancer Evolution VIII	30%	34%	45%
Subaru Impreza WRX STi (2004)	33%	40%	35%
Audi TT 3.2 quattro (8N)	30%	28%	46%
Mitsubishi 3000GT VR-4	33%	33%	36%
Mazda RX-7 (FD) (Series 6)	30%	33%	39%
Nissan 350Z (2003)	30%	36%	33%
Mazda RX-8 (2003)	24%	30%	43%
Audi A3 3.2 quattro	30%	29%	35%
Infiniti G35	26%	30%	30%
Ford Mustang GT (2005)	30%	30%	24%
Pontiac GTO (2004)	30%	32%	22%
Acura RSX Type-S	20%	19%	37%
Peugeot 106 S16	16%	13%	42%
Toyota Supra SZ-R (Mk4)	25%	20%	25%

The query took 17 accesses and 0.00034809112548828125 seconds.

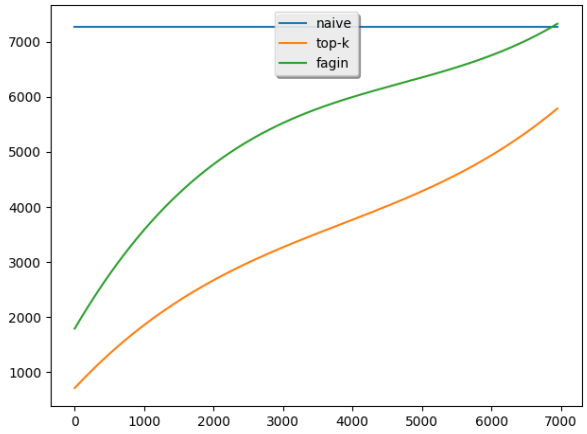
This app was created by Matyáš Skalický in 2018 as a semestral project for the BI-VMW subject.

Experimentální sekce

V následující sekci se pokusím porovnat Top-k, Faginův a naivní algoritmus na základě počtu přístupů do databáze, který je klíčový. Pro přehlednost dodávám i porovnání na reálné době běhu algoritmu. Pro experimentální účely jsem do aplikace přidal databázi o 7 000 a 160 000 položkách s náhodně vygenerovanými hodnotami.

Porovnání počtu přístupů do databáze

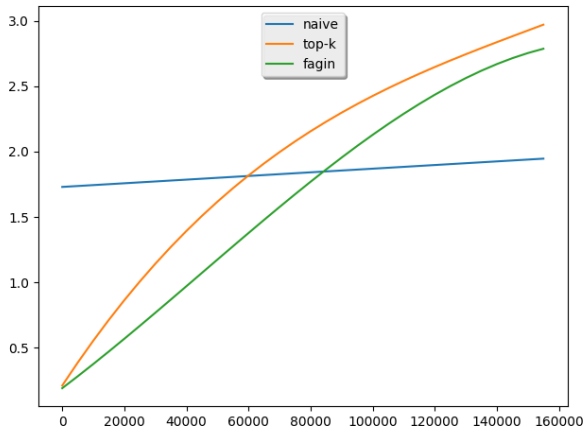
Je vidět, že algoritmus top-k využívá ve srovnání s Faginovým algoritmem značně menší počet přístupů do databáze. V prostředí s pomalým přístupem k databázi se proto i za cenu větší výpočetních nároků vyplatí a dotazy budou rychlejší.



OSA X – VELIKOST DOTAZU (K), OSA Y – POČET PŘÍSTUPŮ DO DATABÁZE

Porovnání doby běhu

Srovnání doby běhu algoritmů na databázi o velikosti 160 000 objektů. Je vidět, že ačkoliv použije algoritmus top-k nižší množství náhodných přístupů, vyžaduje udržování minimální haldy pro uchování top-k elementů delší výpočetní čas. Pokud mám rychlý přístup k databázi s náhodným přístupem, vyplatí se využít Faginův algoritmus.



OSA X – VELIKOST DOTAZU (K), OSA Y – ČASOVÁ DÉLKA DOTAZU

DISKUZE

Projekt, který jsem vypracoval zahrnuje vlastní implementaci Top-K treshold algoritmu. Využíval jsem zjednodušený scénář, kdy všechna data nahraji přímo do paměti a nemusím proto řešit databázový software ani komplikované dotazování. V reálném případě použití by bylo potřeba zakomponovat podobný model do databázové vrstvy aplikace. V mém projektu jsem neměl prostor řešit optimalizaci jednotlivých algoritmů a datových struktur. Další práce v této oblasti by mohla značně zrychlit celé mé řešení. Python rovněž není nejrychlejším jazykem, tudíž aplikace například v C++ by byla bez pochyb rychlejší, ale náročnější na implementaci.

ZÁVĚR

Mým zadáním bylo implementovat a ověřit funkci Top-K treshold algoritmu. Zadané téma mě začalo velmi bavit a užíval jsem si i grafickou stránku aplikace. Nakonec mě nevíc zajímalo, jak je na tom top-k treshold algoritmus v porovnání s ostatními algoritmy. Do mé aplikace jsem tedy přidal i Faginův a naivní algoritmus pro porovnání funkcionality a rovněž několik agregačních funkcí. Celou aplikaci jsem vybudoval ve webovém frameworku Flask, což se ukázalo být nesmírně přínosné. Flask jsem následně využil i pro další semestrální práci.