# Dokumentácia - O19 - Wikipédia parser

**Krátky popis projektu-problému amotivácia načo je to dobré (15 riadkov)**

Moja téma „Vyparsovanie automobilových vozidiel z wikipédii a vytvorenie služby, ktorá by podľa zadaného automobilu (podľa jeho kategórii (automobilka, rok výroby, produkcia, trieda, predchodca, typ karosérie) umožňovala vyhľadať podobné automobilové vozidlá danej kategórie“ sa zaoberá parsovaním Wikipédie. Túto tému som si zvolil z toho dôvodu, že som chcel umožniť používateľom vyhľadanie automobilu podľa čo najpodobnejších kategórii automobilu, ktoré užívateľ zadal, čím by sme mu vedeli pomôcť pri kúpe auta a to tak, že by si vedel pozrieť všetky podobné autá, ktoré patria do kategórie ním zadaného auta.

Problém parsovania Wikipédie a hľadania vzťahov medzi vyparsovanými entitami v mojom prípade medzi automobilmi nie je úplne triviálna záležitosť. Čo sa týka parsovania, je v prvom rade potrebné v rámci Wikipédie identifikovať stránky, ktoré sa týkajú automobilu a následne identifikovať všetky kategórie vozidla (názov, výrobca, rok výroby, trieda vozidla, pohon a podobné vozidlo). Na tento problém je potrebné zvoliť vhodný regex, ktorý jednak je schopný identifikovať stránku (infobox automobile) s vozidlom a regex, ktorý je schopný vyparsovať spomínané kategórie auta. Dáta z týchto kategórii bolo potrebné vhodne rozdeliť, keďže v rámci jednej kategórie sa môže nachádzať aj viacero údajov. Ďalší problém týkajúci sa parsovania predstavuje rýchlosť parsovania. Je potrebné zabezpečiť paralelne spracovanie (čítanie a parsovanie) Wikipédie. Posledným problémom je rýchlosť vyhľadávania nad vyparsovaným datasetom. Je potrebné zvoliť vhodnú implementáciu indexu, ktorý by zefektívnil vyhľadávanie.

**Prehľad súčasných riešení daného problému - existujúci softvér, algoritmy, vedecké články, linky (0,5 strany)**

Jedna zo štúdii (<https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=7f658a81303d6ba74fef874455213349388c2c73>), ktorá sa tiež zaoberá parsovaním anglickej Wikipédie (XML dump cca. 30 GB), a porovnáva výkon viacerých nástrojov na parsovanie ako napríklad Apple’s Xgrid, Hadoop a Google’s MapReduce. A prišli s výsledkom, že Apple’s Xgrid dosahuje najlepší výkon. O sekundu pomalší bol práve Hadoop.

Ďalším riešením je **Python** **modul Beautiful Soup** (https://www.crummy.com/software/BeautifulSoup/), ktorý sa využíva na web scraping a to pomocou HTML tagov (XPath). Beautiful Soup je schopné parsovať aj lokálne zo HTML alebo XML súboru. Z pohľadu rýchlosti parsovania nejde o najefektívnejšie riešenie.

Podarilo sa mi nájsť aj **Python modul** **wikipedia** (https://levelup.gitconnected.com/two-simple-ways-to-scrape-text-from-wikipedia-in-python-9ce07426579b), ktorý priamo umožňuje extrahovať dáta z Wikipédie. Stačí si zadefinovať nadpis, ktorý nás zaujíma a podľa neho tento modul nájde prislúchajúcu wiki stránku, z ktorej je možné pristupovať k jej obsah. Nad týmto obsahom je možné si vytvárať vlastný regex, podľa našich preferencii. Link ku tomuto Python modulu je tu: https://pypi.org/project/wikipedia/.

Riešenie, ktoré sa týka parsovania ľubovoľných infoboxov poskytuje JavaScript modul dijs/infobox-parser (<https://github.com/dijs/infobox-parser>). Dokáže parsovať z nami zadaného infoboxu všetky entity, ktoré obsahuje.

**Popis riešenia, použitý softvér, použité existujúce riešenia, popis problémov ktoré sa vyskytli, popis prác na projekte (1 strana)**

Časová os práce sa nachádza na stránke predmetu. Prvým krokom bolo vytvorenie pseudokódu, ktorý opisoval akým spôsobom (logikou) budem realizovať parsovanie Wikipédie a vyhľadávanie nad vyparsovaným datasetom. Celé zadanie som realizoval v Pythone.

V druhom kroku bolo potrebné vytvoriť základný parsovací skript nad malým wiki datasetom (cca. zazipovaný dump mal 900 MB). Riešenie bolo pomerne jednoduché. Wiki dump súbor som čítal po riadkoch a v prípade, že som narazil na infobox týkajúci sa automobilu tak som začal z nasledujúcich riadkov parsovať automobilové kategórie. Na parsovanie týchto kategórii som vytvoril všeobecný regex, ktorý podľa zadanej kategórie vedel vyparsovať jej obsah. Tým, že každá kategória obsahovala viacej elementov (napr. v rámci triedy auta sa mohlo nachádza viacero tried alebo skratiek tried, do ktorých auto patrí), ktoré bolo podľa mnou zašpecifikovaných delimitrov oddeliť. Tým sme pre každé auto dostali viacero prvkov v rôznych kategóriách. Jedine prvok v kategórii rok výroby bol vždy len jeden. Hlavný problém v tejto časti spočíval v nájdení správnych delimitrov, podľa ktorých je možné rozdeliť prvky v danej kategórii. Vyparsovane autá som si ukladal do JSON súboru. Regex som realizoval pomocou python modulu re. Vyhľadávanie v tomto počiatočnom kroku som robil veľmi jednoducho a to, že som si najskôr našiel či existuje užívateľom zadané vozidlo vo vyparsovanom JSON súbore a ak existovalo tak som si vytiahol jeho kategórie a podľa nich som našiel najpodobnejšie auto. Týmto riešením som skúšal parsovať aj veľký wiki dump súbor (celá Wikipedia - zazipovaný dump 38GB), čo ale trvalo okolo 3 hodín 22 minút. Nad malým datasetom (zazipovaných 900 MB) to trvalo cca. 4 minúty.

V tretom kroku bolo potrebné zapracovať paralelne parsovanie wiki dump súboru. V rámci tejto časti som vytvoril dve riešenia a to vlastnú paralelizáciu cez python modul multiprocessing a následne paralelne spracovanie cez python modul pyspark. V rámci vlastnej implementácie paralelného parsovania wiki dump súboru som využil prístup producer-consumer. Kde producer čítal wiki súbor a ukladal do zdieľaného zásobníku (queue) infoboxy a consumer z tohto zdieľaného zásobníku si vyťahoval jednotlivé infoboxy a parsoval z nich kategórie. Toto riešenie ušetrilo okolo 20% času. Vyparsovanie automobilov z veľkého datasetu trvalo 2 hodiny 38 minút. Toto riešenie som ešte zefektívnil a to tak, že som využil rýchlosť funkcie seek() knižnice bz2 pomocou, ktorej som čítal zazipované dump súbory. Táto funkcia vedela „skočiť“ na mnou definovaný bajt. V rámci testovania tejto funkcie som zistil, že oproti klasickému čítaniu súboru po riadkoch je o 50% rýchlejšia. Tak som si najskôr zistil celkovú veľkosť súboru a tú som rozdelil na 4 časti. Teda vznikli mi 4 producers. Každý z nich čítal inú časť súboru (to som zabezpečil pomocou funkcie seek). Najlepšie výsledky mi dali 4 producers a 1 consumer. Táto úprava mi oproti riešeniu bez paralelizácie ušetrila 44% (vyparsovanie 38GB dump súboru trvalo 2 hodiny 23 minút).

V štvrtom kroku som realizoval paralelizáciu pomocou Python modulu pyspark. Kvôli tomuto riešeniu bolo potrebné do veľkej miery prekopať kód od základov hlavne čo sa týka čítania dump súboru a taktiež regexov, keďže pyspark umožňuje čítať XML súbor po stránkach. Aby bolo možné čítať súbor po stránkach musel som si vytvoriť XML schému a taktiež nainštalovať Java rozšírenie, ktoré poskytuje čítanie XML súborov. V tejto časti som implementoval aj vlastný index. Pre každú kategóriu som si vytvoril vlastný index. Na indexovanie využíval python slovník. Prvý index slúži na vyhľadávanie užívateľom zadané auto. V rámci tohto indexu je kľuč id a hodnota tvorí vyparsovaný automobil. Potom si si vytvoril indexy pre kategórie automobilu. V rámci tohto indexu kľúčom je názov entity z danej kategórie. Napríklad v indexe trieda je kľuč e-segment a hodnota pre tento kľuč je list IDčie všetkých automobilov, ktoré pod túto kategóriu patria. Dosť veľkú zmenu bolo potrebné spraviť aj pri niektorých použitých regexoch. Z dôvodu, že sme nečítali dump súbor po riadkoch ale po stránkach a teda musel som vytvoriť regex, ktorý vedel vyhľadať infobox automobil v rámci celej stránky. Celkovo pyspark parsovanie dát bolo najrýchlejšie. Oproti riešeniu bez paralelizácie som ušetril 55% času. Vyparsovanie 38GB dump súboru trvalo 1 hodinu 44 minút.

**Popis dát na ktorých ste testovali, nejaká dáta aj ako zip na wiki.**

V rámci zadania som používal dva datasety anglickej Wikipédie (dump súbory). Ktoré sa nachádzajú na stránke: <https://dumps.wikimedia.org/enwiki/20220920/>.

Ako **malý dataset** som si zvolil časť Wikipédie a to konkrétne (zazipovaný má 900MB): enwiki-20220920-pages-meta-current10.xml-p4045403p5399366.bz2

Ako **veľký dataset** som si zvolil celú Wikipédiu (zazipovaný dump súbor má 38GB): enwiki-20220920-pages-meta-current.xml.bz2

**Vyhodnotenie slovné subjektívne na nejakých konkrétnych príkladoch. Vo vačšine projektov aj vyhodnotenie pomocou presnosti a pokrytia (precision a recall) (0,5-1 strana)**

V rámci semestra sa mi podarilo dosiahnuť stanovený cieľ a to vyparsovať automobily a ich kategórie z Wikipédii a následne vytvoriť rozhranie vďaka, ktorému si môže užívateľ vyhľadať najpodobnejšie vozidlo z pohľadu kategórií (automobilka, rok výroby, produkcia, trieda, predchodca, typ karosérie) podľa ním zadaného vozidla. V rámci parsovania Wikipedie sa mi podarilo zapracovať paralelne parsovanie na báze pysparku a vyhľadávanie pomocou vlastného indexu, ktorý je realizovaný na základe python slovníkov.

V nasledujúcej tabuľke uvádzam porovnanie a progres rýchlosti parsovania Wikipédie. Kde sa pysparku podarilo najrýchlejšie vyparsovať autá z Wikipédie a naopak očakávaným najpomalším riešením je parsovanie pomocou riešenie bez paralelizácie. V tabuľke uvádzam časy pre parsovanie veľkého Wikipédia datasetu a to konkrétne zazipovaný dump súbor o veľkosti 38GB.

|  |  |
| --- | --- |
| **Typ parsovania (procesor 2,3 GHz 8-Core Intel Core i9)** | **Čas vyparsovania datasetu** |
| Bez paralelizácie | 11620 s (193 minút) |
| S paralelizáciou (1 producer, 1 consumer) | 9652 s (161 minút) |
| S paralelizáciou (4 producer, 1 consumer) | 8061 s (134 minút) |
| S paralelizáciou (pyspark) | 5208 s (87 minút) |

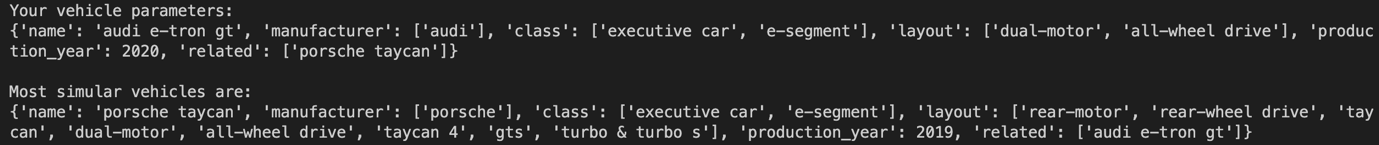
**Príklad vyparsovaného automobilu:**

**Graphical user interface, text, chat or text message

Description automatically generated**

{'name': 'rolls-royce silver wraith', 'manufacturer': ['rolls-royce limited', 'rolls-royce ltd'], 'class': ['full-size car', 'full-size', 'luxury car', 'f-segment'], 'layout': ['front-engine', 'rear-wheel-drive layout', 'front-engine', 'rear-wheel-drive'], 'production\_year': 1946, 'related': ['rolls-royce silver dawn']}

**Príklad nájdenia najpodobnejšieho automobilu ku audi e-tron gt:**

****

**Spustenie, inštalácia softvéru, použitie softvéru. (0,5-1 strana)**

Z linku (<https://dumps.wikimedia.org/enwiki/20220920/>) je potrebné si stiahnuť Wikipédia dataset (dump súbor) a vložiť ho do priečinku /dataset. Napríklad si môžete stiahnuť jeden z datasetov, ktorý som uviedol v popise dát.

Následne je potrebné si aktivovať virtuálne prostredie pomocou príkazu: **pipenv shell**

Je potrebné si nainštalovať potrebné balíky pomocou príkazu: **pipenv install**

Je potrebné sa presunúť do priečinku: **cd /Users/username/.local/share/virtualenvs/pipenv\_name /lib/python3.9/site-packages/pyspark/jars/**

Je potrebné si nainštalovať XML jar do priečinku: **curl -O https://repo1.maven.org/maven2/com/databricks/spark-xml\_2.12/0.15.0/spark-xml\_2.12-0.15.0.jar**

Spúštanie programu sa robí pomocou príkazu: **spark-submit --jars** **/Users/username/.local/share/virtualenvs/pipenv\_name /lib/python3.9/site-packages/pyspark/jars/spark-xml\_2.12-0.15.0.jar pyspark\_parser.py**

Na začiatku Python súboru **pyspark\_parser.py** sa nachádzajú globálne premenné s popismi, ktoré určujú správanie skriptu.

V prípade, že si chceme spustiť unit testy tak je potrebné zadať príkaz: **python unit\_tests.py**