Pagerank

Popis

Cílem semestrální práce bylo vytvořit algoritmus pagerank, který běží na úzké části internetu. Uživatel zadá text a aplikace mu vrátí seznam stránek seřazený podle podobnosti textu a hodnocení stránky algoritmem Pagerank.

Pagerank počítá důležitost stránky na webu. Využívá inlinků a outlinků, což jsou linky vedoucí buď na a nebo ven ze stránky a můžeme je chápat jako doporučovací mechanismy. Výsledné hodnocení se může brát jako pravděpodobnost, že se klikáním na linky ocitneme na stránce.

Zadání prioritizuje algoritmus Pagerank společně s crawlováním části internetu. Podobnosti textu lze zjednodušit či řešit externí knihovnou.

Způsob řešení

Při implementaci řešení jsme vycházeli z přednášky a této stránky : https://bit.ly/3dJ7fa0.
Nejsnažším způsobem řešení nám pak přišlo vytvořit požadovanou “Google” matici, která je stochastická a použít iterativní metodu (power method).
Nejdříve jsme tedy vytvořili matici A, která reprezentovala jednotlivé stránky a linky mezi nimi. Dále jsme vytvořili matici B o stejných rozměrech jako A, která je celá naplněná hodnotou 1/n, kde n je celkový počet stránek. Následná matice M : **M = (1-p) * A + p*B.**
Písmeno p zde reprezentuje tzn. “dumping factor”, ten vyjadřuje pravděpodobnost, že surfer, který je na nějaké stránce, danou stránku zahodí a “teleportuje” se na jinou stránku. Každá stránka má stejnou pravděpodobnost, že se na ní surfer přemístí. Tento fakt reprezentuje právě matice B. Samotný Google údajně používá hodnotu p=0.15, kterou jsme použili také.

Dále jsme vytvořili počáteční vektor **v**, kde každý prvek reprezentuje právě jednu stránku a jelikož ze začátku nemáme žádnou informaci o tom, jakou má daná stránka váhu, jsou všechny hodnoty rovny 1/n.
Následně násobíme matici M daným vektorem v : **M*v.** Tuto operaci několikrát opakujeme (10x), ale místo vektoru v použijeme výsledek předchozího násobení. **M*v1, m*v2**. To má za následek, že výsledný vektor **v*** konverguje ke konečnému pageRanku.

Ještě dodám, že matice A je sestavena tak, že každý řádek matice reprezentuje jednotlivé stránky a nenulové hodnoty v tomto řádku značí váhu linku, se kterým na tuto stránku ukazuje jiná stránka (s indexem daného sloupce). Jednotlivé sloupce pak tedy reprezentují kam stránka odkazuje a součet každého sloupce je roven 1.

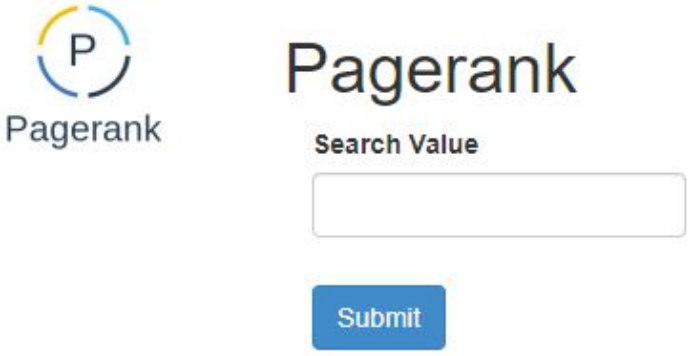
Implementace

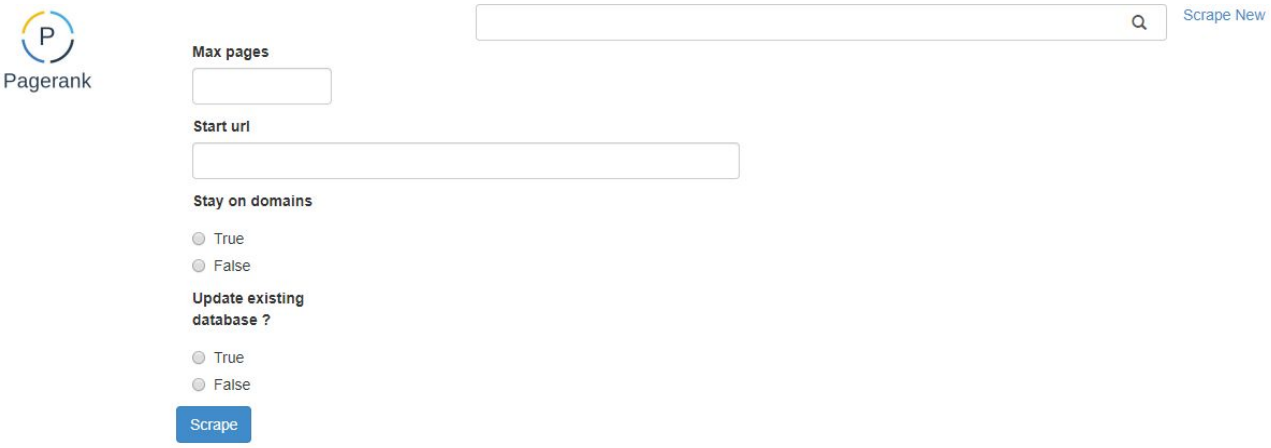
Protože projekt je psán v pythonu, zvolili jsme jako framework pro aplikaci Flask. Umožňuje jednoduše generovat html kód pomocí Jinja2 a zároveň nemá tolik zbytečného kódu okolo jako Django. Pro crawler byla použita knihovna Scrapy. Pomocí této knihovny jsme schopni procházet weby, získat jejich informace a následně z nich sestavit námi potřebný graf. Pro indexování webových stránek a vyhledávání termů byla použita knihovna Whoosh. Díky této knihovně jsme tedy vyhledali všechny validní stránky, které jsou následně seřazeny podle jejich ranku. Celé je to zabalené v docker kontejneru pro lehkou distribuci a zamezení chyb specifických pro operační systémy. Jako operační systém byl použit Alpine Linux, který má kompletní index všech linuxových balíčků a je velký jen 5MB.

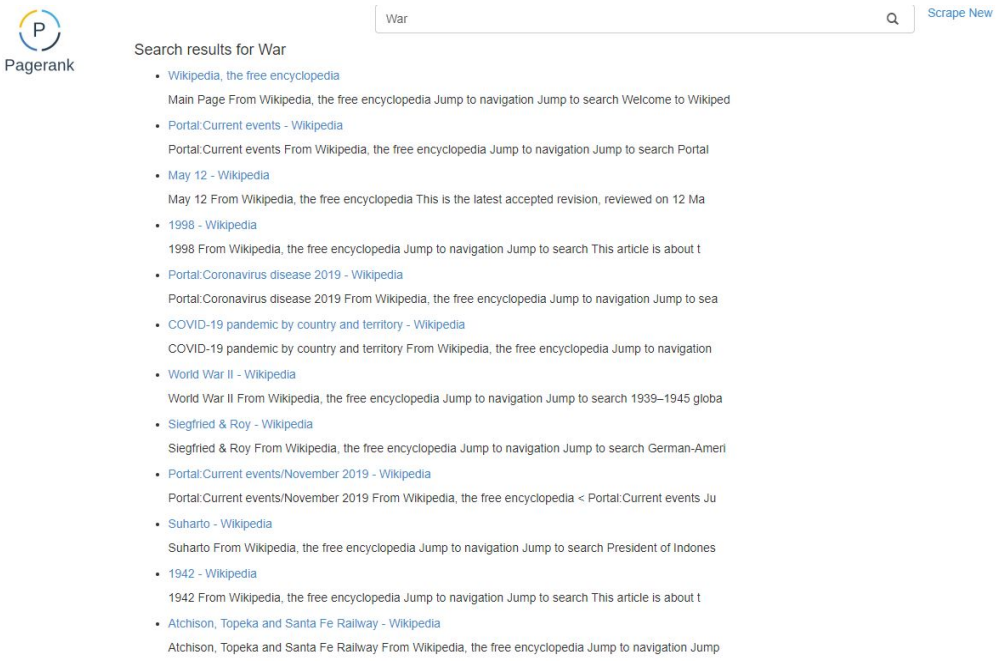
Implementace také kromě vyhledávání umožňuje vytvořit si vlastní databázi stránek a nebo ji rozšířit a následně vyhledávat na těchto datech. Stačí jen zvolit doménu, ze kterých se má databáze plnit, počet stažených stránek a jestli má crawler zůstat na dané doméně.

Příklad výstupu

Hlavní stránka, možnost změny crawlovací základny a příklad výsledků







Experimentální sekce

Pro účely aplikace bylo nacrawlováno cca 1000 stránek z anglické wikipedie, což trvalo cca 6 minut. Předkalkulování a uložení výsledků zabralo cca 3 minuty. Požadavky pak probíhají v řádech milisekund. Zkoušeli jsme natáhnout více stránek abychom tento algoritmus otestovali a při cca 10 000 stránkách bylo už znatelné zpoždění aplikace.

Také jsme zjistili, že hlavní stránka wikipedie se zobrazí při většině námi testovaných dotazů, ale to dává smysl, vede na ní odkaz z každé stránky. Při procházení více domén se toto potvrdilo, hlavní stránky se vždy objevovaly více nahoře než jiné stránky na doménách.

Diskuze

Aplikace není optimalizována pro prohledávání celého internetu. Vysoká paměťová a časová složitost se začne hodně rychle projevovat na výkonu celé aplikace. Internet přitom obsahuje miliony stránek a projít ho celý by způsobem prezentovaným zde trvalo moc dlouho. Výpočet pageranku v řádu tisíců stránek není tolik časově složitý, až nás to překvapilo jak rychlé to dokáže být.

Závěr

Překvapilo nás jak je výpočet pageranku jednoduchá záležitost. Na druhou stranu jsme si uvědomili jak moc faktorů existuje a jak se musí optimalizovat vyhledávání ve firmách jako Google či český Seznam. Také by aplikaci bylo vhodné, tedy aspoň backend, přepsat do nějakého z rychlejších jazyků, jako třeba c/c++.