SŠPU
STŘEDNÍ ŠKOLA PRŮMYSLOVÁ
A UMĚLECKÁ, OPAVA

ZÁVĚREČNÁ STUDIJNÍ PRÁCE

dokumentace

Ultimate Search Engine

Filip Hostinský

Obor:	18-20-M/01 INFORMAČNÍ TECHNOLOGIE se zaměřením na počítačové sítě a programování
Třída:	IT4
Školní rok:	2019/2023

Poděkování

Chtěl bych poděkovat Filipu Peterkovi za nápad, přípravu a konzultace spojené s USE, panu Ing. Petru Grussmannovi za zpřístupnění, zprovoznění a hostování serveru, Mgr. Marku Lučnému za konzultace a mým kolegům, Davidu Stočkovi a Vojtěchu Binarovi za spolupráci na tomto projektu.

Prohlašuji, že jsem závěrečnou práci vypracoval samostatně a uvedl veškeré použité informační zdroje.
Souhlasím, aby tato studijní práce byla použita k výukovým účelům na Střední průmyslové a umělecké škole v Opavě, Praskova 399/8.

V Opavě, 31. prosince 2022

Podpis autora práce

Anotace

Tato práce se především zabývá analýzou dnešního webového prostředí a optimistickou teorií pro upřednostňování důležitých stránek, která si zakládá na jeho hypertextové nátuře primárně využitím algoritmu *Pagerank*. Dále se snaží pokrýt velice komplexní vyhledávací mechanismus databáze v jeho centru a nakonec prezentuje stručný přehled celkové anatomie enginu, složeného z řady individuálních komponentů za účelem vysvětlení jejich funkcí.

Praktická část se zaobírá problematikou efektivního *crawlování* a *big data*, respektive ukládání *scrapů*, zpracování *HTML* do hledatelné podoby a proveditelnostní komplexitou výpočtů algoritmů spojených s vyhledáváním. Taktéž vysvětluje kompromisy a roli člověka v autonomním systému.

Výsledkem je *studentský* vyhledávač postavený na základě *Elasticsearch* databáze, který dokáže odpovědět na dotaz uživatele relevantnějšími stránkami prvně a kde uzná za vhodné, doplnit tyto výsledky o dodatečné informace. Kromě toho usnadňuje samotné zadávání relevantními nášepty a obsahuje *hinty* - utility prezentované za účelem zjednodušení hledání.

Klíčová slova

Search Engine, Web, Vyhledávání, Big data

Obsah

Úvod

Informací na webu dennodenně rapidně přibývá. Dle nejlepších odhadů existuje přibližně 1 miliarda domén, obhospodařujících téměř 18 triliard (1.8e16) stránek. Orientace v takto obrovském shluku informací je daleko za hranicemi jakéhokoliv lidského porozumění. Aby tato data byla k využití, musí zde existovat jakýsi systém, který nejenže dokáže spojit uživatele s jejich oblastmi zájmů, ale také seřadit tento obsah na základě řady důležitostních měřítek, jako je například důvěryhodnost zdroje, čerstvost, kvalita a kontext. Současné, nejpokročilejší, systémy využívají takovýchto hodnot řádově v počtu několika stovek.

Tímto prezentujeme naše řešení tohoto problému založeného na technice softwaru běžně známého jako *crawler*. Ten má za úkol pravidelně prozkoumávat web a tím rozšiřovat svůj *index*. Cílem bylo vytvořit plně automatický a centralizovaný vyhledávač v omezeném měřítku, který si sám poradí s běžnými překážkami a nástrahami webu a to bez jakéhokoliv zásahu lidské ruky. Engine tím pádem sám dokáže crawlovat web, indexovat tato data a následně běžnému uživateli prezentovat užitečné výsledky pomocí přívětivé webové aplikace, zatímco se neustále zlepšuje ve svém účelu.

Vyhledávání je neodmyslitelnou součástí všech rozsáhlejších webových stránek. Je to velice minimalistická součást jejich designu, ale pod kapotou často bývá právě tou nejkomplikovanější. Search lze, z jeho nejzákladnější podoby, neustále vylepšovat o nové funkce a optimalizace. Složitost je tím pádem svévolná. Jen málo lidí se pozastaví nad tím, co dělá dobrý search engine skutečně dobrým. My ano. A nebylo to snadné!

Kapitola 1

Analýza dnešního webu

K pochopení problematiky webového vyhledávače, je nutno nejprve pečlivě prozkoumat jeho, strukturu, historii a moderní využití. Rozšíření webu a jeho provázanosti znamená, že správná statistická analýza, v podobě modelu grafu, je neodmyslitelnou součástí při tvorbě search enginů.

1.1 Historie

World Wide Web prošel od jeho návrhu v CERNu, roku 1989, několika velkými proměnami. Z jednoduchého *Web 1.0* se statickým obsahem a plnou otevřeností—"web dokumentů", přes *Web*u *2.0* s interkonektivitou, grafickými médii a centralizaci k *Web*u *3.0*, který se navrací k myšlence decentralizace. Web se stále mění a je třeba se patřičně přizpůsobit.

1.2 Teorie grafu

Na webu neexistuje žádný oficiální registr všech stránek. Tím pádem úkol zjištění podoby webu spadá search enginům. O WWW můžeme hovořit, s jeho hypertextovým charakterem, jako matematický model několika *konečných* grafů, kde *prvek* je stránka a odkazy působí jako *hrany*. Od úrovně fyzické, distribuované sítě, k jednotlivým stránkám. Musíme brát v potaz, že takových odpojených grafů může existovat několik—většinou malé, osobní, blogy nebo stránky se zpoplatněným obsahem.

Existují mnohé modely snažící se zužitkovat tuto strukturu ve svůj prospěch. Ty hrají důležitou roli v doručování kvalitních výsledků. Příkladem jsou algoritmy *Pagerank* a *Hyper search*

1.2.1 Role *backlinků*

Termín *backlink* pojednává o odkazu *jiné* stránky na danou stránku. V mnoha případech text backlinku, neboli *anchor text* dokáže poskytnout lepší popis a označení dané stránky, nežli její obsah a *meta* tagy. Nevýhodou je, že backlinky jsou pouze dohledatelné v plně postavené grafové struktuře.

1.3 Problémy

1.3.1 Rozsah a proměny

Jak již bylo zmíněno, veřejný web je sbírka několika nesmírně velkých grafů a prochází neustálou proměnou. Časem mnoho URL adres zmizí, přesune se na jinou adresu nebo je nahrazeno jiným obsahem. Tím pádem, nejsou-li již indexované stránky přeindexované, search engine se může stát nepoužitelným.

Při crawlování je potřeba provést rozhodnutí, je-li zájem zabrat široké spektrum domén a tím pádem klást důraz na rozmanitost či omezit crawl na kýženou část internetu a dosáhnout tím konkrétnějšího využití. Procesem *efektivního* crawlování a zpracování takového grafu se zabývají pozdější části (část 3.1 a 4.1).

1.3.2 Správné využití tagů

Meta tagy jsou definována metadaty stránky v HTML. Ty bývají pouze doporučené a tím pádem se nemusí vyskytovat na všech stránkách. A pokud ano, mohou nabývat několika podob, znamenající totéž, anebo být nesprávné. Na systému je pak určit a správně vyhodnotit pravdivost, případně se pokusit o dopočtení těchto dat.

1.3.3 Klamání search enginů

Všechny algoritmy mají svá úskalí a správným zneužitím lze docílit kýženého rankovacího skóre. Pagerank je možné zmanipulovat ovlivněním struktury prvků a hran v grafu. Moderní enginy mají své způsoby zamezení a penalizace takovýchto pokusů.

1.3.4 Lokalizace, personalizace a omezení

Pro spotřebitele z jiných regionů, v jiném čase a s jinými zájmy mohou být užitečné různé informace. Právní a regulatorní předpisy mohou taktéž hrát roli v zobrazeném kontentu. Stránky snažící se takto zaujmout koncového uživatele musí search enginům nabídnout relevantní a kompletní meta tagy.

Kapitola 2

Fulltext vyhledávací mechanismus

V dnešní době lidé očekávají najít odpověď na jejich dotaz mezi prvními výsledky. Běžně se ale na jednoduchý dotaz najde hned několik miliard *hitů*, každý z nich obsahující požadované slovo. Tato část pojednává o způsobu jak vybrat ty výsledky, které co nejvíce souvisí s daným požadavkem.

2.1 Fulltext

Mechanismus fulltextu se zabývá hledáním daného *tokenu* (slova) v určitém řetězci znaků nebo v dokumentu databáze podporující fulltext technologii. Aby se urychlilo vyhledání ve větších datových rámcích, je tento úkol rozdělen do dvou kategorií: *indexace* a *vyhledávání*. Princip efektivní indexace spočívá v datastruktuře *slovníku*, která drží hodnoty tokenů jako *klíče*, ukazující na týkající se dokumenty. Vyhledávání se pak stává snadným, jelikož se stačí pouze odvolat na index klíčů, oproti přezkoumávání celého rámce od začátku. Takovýto indexační model je běžně známý jako *invertovaný index*.

K indexaci lze implementovat funkce jako opomíjení *stopslov*, které v jazyce nenesou žádný význam (v angličtině "the", "a", "is"). Další funkcí může být *stematizace*, umožňující hledání dle kmenového výrazu slova.

Zpřesňujícím měřítkem pro určení kvality po nalezení všech pasujících prvků může být *pagerank*, popsaný v následující sekci.

2.2 Pagerank

Aneb "Vznesení pořádku webu"

Pagerank je nejmocnějším rankovacím algoritmem, který USE využívá jak pro *scraping*, tak pro servírování výsledků. Vypočítává postavení *prvku*, neboli jeho *rank PR(u)* na základě prvků odkazujících k němu, též nazývanými jako *hrany*. To funguje tak, že prvek odkazující, v, uváží svůj rank $PR(v)$ vůči celkovému počtu hran, na které v odkazuje, $L(v)$. Sumou všech takovýchto hran od v získáme konečný rank, $PR(u)$. Kvůli jeho provázanosti s $PR(v)$ se musí vypočítávat iterativně a suma všech prvků je běžně rovna 1.0.

$$\mathrm{PR}(u) = \sum_{v \in B_u} \frac{\mathrm{PR}(v)}{\mathrm{V}(v)}$$

To v praxi znamená, že může být podstatně výhodnější být vázaný menším počtem silnějších hran, oproti velkému počtu slabých hran. V kontextu search enginů *prvek* stojí pro stránku a *hrana* pro odkaz. Nenachází-li se na stránce žádný odkaz, nazýváme ji *sink page* a její rank je rovnoměrně přerozdělen mezi všemi stránkami. Algoritmus Pagerank má, mimo informatiky, své zastoupení například v oblastech biologie, v neurovědě, chemii, ekologii, fyzice nebo předvídání dopravního provozu.

2.3 Měření úspěšnosti

Důležitými dvěma termíny ve statistice k určení úspěšnosti jsou *přesnost* a *vybavování*. Přesnost pojednává o počtu *chtěných* výsledků (pravdivých) k *celkovému počtu získaných* výsledků (pozitivních), zatímco vybavování se zaobírá počtem *získaných, chtěných* výsledků (pravdivě pozitivních) k *celkovému počtu chtěných* výsledků (pravdivých). Tyto dva vzorce jsou na sobě závislé nepřímou úměrností.

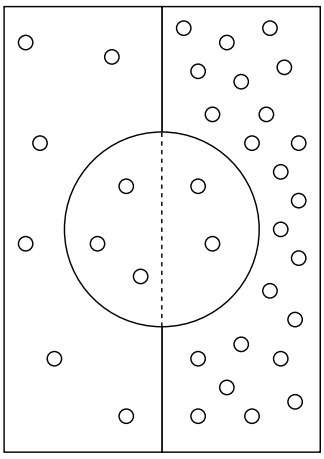
$$\begin{aligned} \text{přesnost} &= \frac{|\{\text{relevantní dokumenty}\} \cup \{\text{získané dokumenty}\}|}{|\{\text{získané dokumenty}\}|} \\ \text{vybavování} &= \frac{|\{\text{relevantní dokumenty}\} \cup \{\text{získané dokumenty}\}|}{|\{\text{relevantní dokumenty}\}|} \end{aligned}$$

Na následujícím, přehledném, diagramu (Obr. 2.1) se můžeme o těchto termínech také zmiňovat jako chtěné (vlevo):

- pravdivě pozitivní (true positive)
- chybně negativní (false negative)

a nechtěné (vpravo):

- chybně pozitivní (false positive)
- pravdivě negativní (true negative).



Obrázek 2.1: Diagram správných výsledků. Chtěné vlevo, nechtěné vpravo; získané uvnitř kruhové výseče

Skutečně získané výsledky jsou znázorněny kruhovou výsečí uprostřed. Opatrným přizpůsobováním váh a upravováním vyhledávacího algoritmu lze docílit změny v proměnných přesnosti a vybavování.

Falešně pozitivní výsledky

Jazyk je často nejasný bez upřesňujících souvislostí. Proto se naskýtá problém demystifikace kladeného termínu. I po bezchybném výčtu všech výsledků, termíny mohou stále odkazovat na několik odlišných definic. Za pomocí statistiky lze seskupit takovéto termíny do několika shluků založených na *Bayesově dedukci*.

2.4 Co je na stránce důležité

Webové stránky bývají neuspořádané a správné rozlišení mezi článkem nebo popisem produktu vůči reklamě a jiným, nedůležitým obsahem se tak pro stroj stává obtížným úkolem. Nelze se ani plně spoléhat na *article* tag, protože není příliš využívaný.

Zpracovací algoritmus se tak musí přizpůsobit všemožným stránkám individuálně, aby vyextrahoval pouze chtěné informace.

2.5 Ideální způsob pokládání dotazu

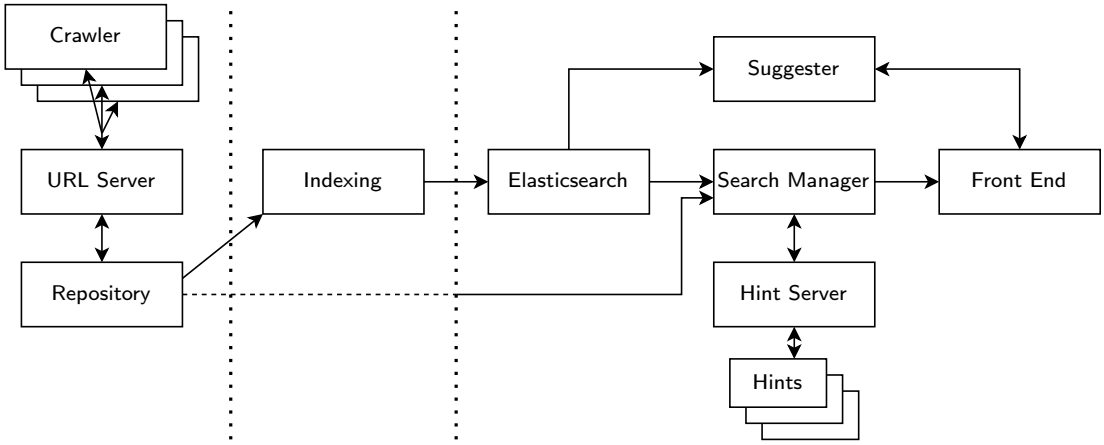
Využívání počítačového zaznamenávání *explicitně* umožňuje uživateli vyjádřit svůj dotaz přesněji a dosáhnout tak lepších výsledků.

Kapitola 3

Anatomie USE

Jako každý komplexní systém i ten náš je složený z řady komponentů, které mezi sebou komunikují dle standardní HTTP konvence v asynchronním stylu.

Následující sekce pojednávají o rolích jednotlivých dílčích částí.



Obrázek 3.1: Diagram USE

3.1 Získávání a ukládání dat

3.1.1 URL Server

Aby bylo možné rozšiřovat své povědomí o webu efektivně, URL Server musí rozhodnout, které stránky upřednostnit před jinými. To představuje určitý problém, jelikož bez znalosti celé sítě předem na tento úkol neexistuje žádné přesné řešení. Více o průzkumu efektivního crawlování v sekci 4.1.

URL Server má další povinnosti. Ověřuje a případně dopočítává pravost meta tagů, zejména označení jazyka. Hlídá si již sesbírané stránky a rozpoznává adresy vedoucí ke stejnému zdroji, aby zbytečně nezaindexoval jednu stránku vícekrát. Taky určuje rychlost crawlování jednotlivých domén. Některé z nich totiž implementují *rate-limiting*. Tyto a jiné problémové domény si poznamenává.

3.1.2 Crawler

Způsob, jakým scrapovat se může lišit dle použitého softwaru. Jedním z nejdůležitějších funkcí je schopnost vykreslit stránky spuštěním Javascript kódu, jelikož nemálo obsáhlých webových stránek a aplikací jsou založeny na technice *client-side rendering* a bez vykreslení by nebylo možné zaznamenat veškerý obsah. Dalším potřebným kritériem je funkce naprostého dohledu a kontroly s načítáním stránky během celého procesu scrapingu. Pro účely USE nebylo zapotřebí indexovat vizuální média a tudíž se ani nestahují.

Distribuovaná podstata crawleru a centralizovaný protokol URL Serveru zajišťuje možnost velice rozsáhlé a výkonné sítě pro možný velkokapacitní provoz.

3.1.3 Repository

Repositář slouží k uschování celého HTML každé stránky, její URL a přesměrovacích URL (URLs vedoucí ke stejnému dokumentu) a časový údaj. USE byl postaven pro experimentální účely a díky tomu těží z přítomnosti repositáře. Pro změnu v datastruktuře vyhledávače je zapotřebí provést pouze interní přepracování, namísto přecrawlování celého webu. To ve výsledku nesmírně usnadňuje vývoj a experimentaci a nabízí potenciál pro speciální hinty.

Implementace se nelimituje na žádnou konkrétní databázi, natož explicitní využití databáze jako takové.

3.2 Zpracování dat

V této části musíme převést všechny scrapy z neuspořádaného HTML do fulltext hledatelné podoby a spočítat všechny algoritmy. Zásadním krokem je sestavení grafu s ohledem na reálná omezení. Poté se může provést několik analýz a výpočtů. Jedním z nich je Pagerank. Finální krok spočívá ve zpracování dokumentů v předem umluvené podobě do indexu Elasticsearche. Zde se, mimo jiné, uplatňuje teorie *anchor textu* backlinků. Slovo o výkonnostní náročnosti se stavbou grafu v sekci 4.2.

3.3 Servírování výsledků

Všechny komponenty využívané při vyhledávání a servírování jsou *stateless*. Tato design volba ve výsledku počítá s neomezenou horizontální škálovatelností pro případ nadměrné zátěže a dealokaci zdrojů při nulovém vytížení.

3.3.1 Search manager a Elasticsearch

Základem pro vyhledávání je nesmírně schopná databáze Elasticsearch, která spravuje všechny zpracované dokumenty a automaticky drží svůj fulltext index v aktuálním stavu a umožňuje tak pokročilé vyhledávání. Search manager je posledním komponentem řídícím všechny search výsledky enginu. Má na starost správnou agregaci požadovaného dotazu a přeložení do Elasticsearch query jazyka společně se současným dotázáním *hint server*u. Na zpracované výsledky v Elasticsearch nelze vždy plně spoléhat s jejich relevancí vzhledem k aktuálnímu

dotazu a tak běžně spolupracuje s původním HTML stránky. Podrobnější podoba Elasticsearch query v sekci 4.3.

3.3.2 Suggester

Neboli našeptávač komunikuje vždy přímo s klientem. Musí zvládat podstatný nápor dotazů, při každé změně ve vyhledávacím poli a to v reálném čase. V Elasticsearch má svůj index s dostupnými, našeptitelnými, slovy s ohledem na předchozí kontext.

3.3.3 Hint server a hinty

Velice časté pro základní utility jsou hinty. Jedná se o řadu jednoduchých programů, dostupných na stránce s výsledky rovnou. Hint server po přijetí dotazu obvolává všechny hinty ze seznamu a v případě, že hint dosáhl určité shody, navrátí svůj obsah zpět hint serveru, který *zagreguje* výsledky.

3.3.4 Front-End

Rychlé načtení stránky je klíčem k úspěchu na webu. Proto učiněním všech dotazů před odesláním základní stránky *lokálně* ušetří dodatečný *roundtrip* mezi klientem a serverem, předcházejícím zahlédnutí *hydratace* stránky uživatelem. Závislost na Javascriptu je tak minimalizována.

Ve výsledku, server front-endu získaný dotaz vyšle Search manageru, vyčká na odpověď a odešle již plně zpracovanou stránku uživateli. Jediným interním komponentem, se kterým klient musí komunikovat přímo je Suggester.

Kapitola 4

Výzvy a řešení

4.1 Efektivní crawlování

Správný přístup k získávání stránek je esenciální vzhledem k rozměru webu. Nejefektivnějším způsobem je znát celý graf v daném okamžiku. To ovšem, bez předchozího prozkoumání, není možné. Nabízí se pak několik algoritmů průzkumu, z nichž bylo přezkoumáno od nejzákladnějších modelů, po modely využívající Pageranku a využit byl ten nejvíce vyhovující.

Podklad pro hodnocení daného modelu byl, pro každý prvek grafu, stanoven jeho konečný Pagerank vzhledem k celému naměřenému setu, jakožto nejexaktnější hodnota důležitosti, které lze, z různých algoritmů, dosáhnout. Není, tím pádem, zahrnován sémantický kontext.

4.1.1 Jazyk

Meta tag označující jazyk stránky může být ošemetný. Ani největší stránky nutně neobsahují jazykový tag. Ideální systém by měl počítat s nespolehlivostí daného tagu a vždy se snažit ověřit, případně dopočítat správnost výroku. URL Server, pro maximální efektivitu, nesmí ztrácet čas na jinojazyčných stránkách a zamezit jejich rozmachu v raných etapách.

USE ověřování funguje způsobem srovnávání jazykové zásoby dokumentu vůči předem definovanému slovníku. Všechny nevyhovující dokumenty jsou v repositáři poznačeny.

4.1.2 Modelování datasetu

Důležitým rozlišením při modelování datasetu je *hledání do šířky* oproti *hledání do hloubky*. S ohledem na graf 4.1 stačí, v tomto případě, prohledat $\sim$2.6% nejvýznamnějšího obsahu (z dostupného *en.wikipedia.com* datasetu; $n = 30k$), abychom získali 50% důležitostní hodnoty Pagerank domény (skutečné číslo může být ještě nižší). S omezenými zdroji je dobré zaměřit se na hledání do šířky dříve, než vzejdou pokusy o hloubku, protože interní Pagerank hodnota tím bude zmanipulována dostupnou grafovou strukturou o nadřazenosti dané domény. S větším vzorkem se tyto nesrovnalosti napraví. Dostatečný vzorek by však musel obsahovat podstatnou část webu.

Většina domén bývají "chamtivé"– mívají strukturu, která podporuje svůj bezprostřední růst. Stává se tak běžně preferováním odkazování na svůj obsah, pod svou doménou. Stránka toto ani nemusí dělat úmyslně, jelikož je zvykem a dobrým záměrem mít užitečné a navigační odkazy přehledně na očích.

4.1.3 Zhodnocení modelů průzkumu

Nejjednodušším a prvním algoritmem USE byl rekurzivní, *breath-first* model, který z každé nové stránky scrapne všechny dostupné odkazy. Začal-li by s důležitou stránkou, měl by v úvodu jistý náskok. Problém však nastal, jakmile se stránky začaly větvit a algoritmus bral v potaz všechny odkazy, mnoho z nichž nebyly určeny k prokliku, což způsobilo nával nepoužitelných stránek. Potíž taktéž byla spojena s nadměrným zacházením do hloubky.

Pozdějším modelem byl *Plain Pagerank*, který zúročil předcházející výpočet Pageranku ve svůj prospěch. Doména, jenž přálo štěstí však, během několika *iterací*, začala zaplavovat frontu stále větší mírou. Nežádané hledání do hloubky tak přetrvalo. Nenapomáhaly ani komplikace spojené s mandatorním výpočtem Pageranku před každou iterací.

Řešením obou z problémů, které pronásledovaly předchozí algoritmy přinesl model Organic Pagerank.

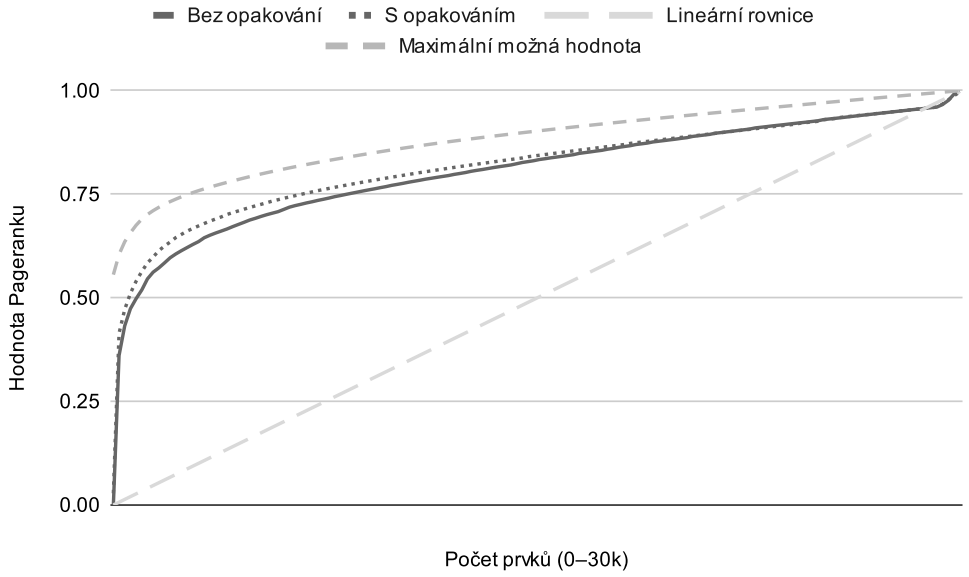
4.1.4 Organic Pagerank

Model pod zkratkou OP je založený na simulaci náhodného uživatele internetu, který si obstará náhodnou stránku, klikne na náhodný odkaz a má jistou pravděpodobnost, že se pozastaví právě na této, nové, stránce a využije ji jako základ pro další kolo. V opačném případě si model vyžádá novou náhodnou stránku a proces se opakuje, *ad infinitum*. Jeho výhodou je robustnost. Díky namátkovosti je méně pravděpodobné, aby se algoritmus stal posedlý jednou, "nekonečně" velkou částí internetu a vynechal všechny ostatní.

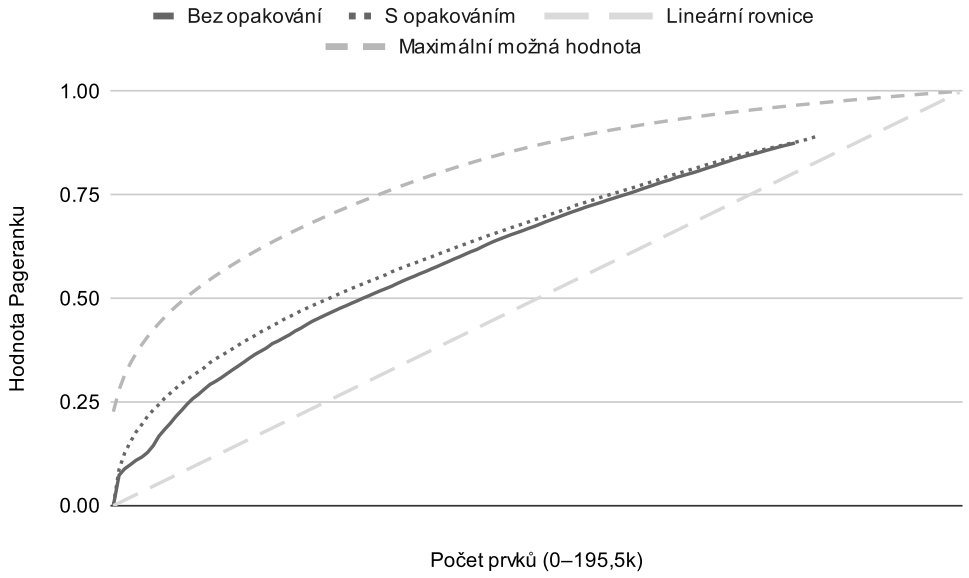
Při vybírání následující adresy, během crawlingu, je zapotřebí odlišit mezi dvěma architekturami: *bez opakování* a *s opakováním*. Jde o opětovný průzkum a zahrnutí již scrapnutých stránek, coby nových, během přiuvážení následující stránky, byť jen symbolicky z repositáře. Grafy 4.1 a 4.2 dokazují mírnou nadřazenost právě algoritmu s opakováním.

S ohledem na grafy 4.1 a 4.2 se můžeme přesvědčit o efektivitě modelu OP vůči nejvyšší možné hodnotě, které lze dosáhnout a lineární funkci, která předurčuje výsledek zcela náhodného modelu. Otevřený web představuje větší výzvu, protože neobsahuje jeden centrální bod, ve srovnávání s omezení vzorku na předem určenou doménu. Musíme však brát v potaz, že grafy představují pouhou aproximaci pravého výsledku.

Předností modelu OP je slibný poměr cena/výkon, jelikož nevyžaduje dřívější výpočet a namísto toho spoléhá především na nahodilost, bez mezikroků.



Obrázek 4.1: Simulovaný proces efektivního prozkoumávání webu modelem OP, zprůměrovaný 20 pokusy domény *en.wikipedia.org* (důležitost měřena vůči konečnému Pageranku).



Obrázek 4.2: Simulovaný proces efektivního prozkoumávání webu modelem OP na otevřeném webu (důležitost měřena vůči konečnému Pageranku).

4.2 Technická proveditelnost zpracování dat

4.2.1 Stavba grafu

Ukázalo se, že stavba grafu může být velice náročný počin a tím pádem je, pro co nejnižší paměťovou stopu, rozdělen do dvou částí zaměřených na maximální efektivitu. Systém musí nejprve obdržet seznam všech URL adres a vložit je do objektové struktury `PagerankPage`, aby mohl přistoupit k dalšímu kroku–vytvoření spojení. Z každého dokumentu se extrahují URL adresy a všem nalezeným objektům s takovouto identifikací přiřadí backlink v podobě *pointer*u na adresu objektu s obrazem URL originálního dokumentu.

```
# 1. část: indexace všech URL
repositoryDocs(dbClient).consumeEach {
    everyLink.add(Url(it.finalUrl).cUrl())
    it.targetUrl.forEach(everyLink::add)
}
```

V této části si program stáhne všechny uložené URL adresy z repositáře do Kotlin `List` struktury. Jelikož běžně několik adres vede ke stejnému výsledku, `targetUrl` je seznam těchto adres, zatímco koncová URL adresa je označena jako `finalUrl`.

```
class PagerankPage(
    val url: String,
    val backLinks: MutableList<PagerankPage>,
    var rank: DoubleArray = DoubleArray(2),
    var forwardLinkCount: Int,
    var doesForward: Boolean,
    )
```

Objektovou strukturou, se kterou bude možno poskládat strukturu grafu je `PagerankPage`. Důležitou součástí je proměnná `backLinks` typu `MutableList`, která drží odkazy v paměti k objektům, jejíž HTML linky obsahovaly právě jejich URL adresu. Proměnnou `doesForward` jsou označeny všechny objekty, kterých URL adresa byla obstarána z repositáře jako `targetUrl`.

K další optimalizaci, proměnná `rank` je typu `DoubleArray` o délce 2 elementů. Jeden z nich vždy stojí pro nynější rank, zatímco druhý ukládá hodnotu z nové iterace.

```
# 2. část: vzájemné provázání
# poznámka: kódový výňatek byl zkrácen k účelům dokumentace

repositoryDocs(dbClient).consumeEach { doc ->
    val parsed = Jsoup.parse(doc.content)
    val docLinks = parsed.pageLinks(Url(doc.finalUrl))

    val pagerankPageElem = get(doc.finalUrl)

    doc.targetUrls.forEach { targetUrl ->
```

```
            val targetPage = get(targetUrl)

            targetPage.doesForward = true
            targetPage.forwardLinkCount += 1
            pagerankPageElem.addBacklink(targetPage)
        }

        docLinks.forEach { link ->
            val linkedPage = get(link) ?: return@forEach
            pagerankPageElem.forwardLinkCount += 1
            linkedPage.addBacklink(pagerankPageElem)
        }
    }
```

Po převedení všech získaných linků do `Array` struktury, která je hospodárnější s operační pamětí, pod objektovou strukturou `PagerankPage`, se musí znovu přeparsovat celá databáze. Nyní každý link, každé stránky, se podrobí přezkoumání, zda-li se vyskytuje v objektovém poli. V kladném případě je objektu pole zaindexován odkaz na objekt jako backlink a `forwardLinkCount` původního objektu se navýší.

Výsledkem je datastruktura plně připravena pro výpočet Pageranku a indexace backlink anchor textů. Využitím jemného *low-level* linkování se, oproti ukládání adres v podobě typu *string*, dramaticky ušetří paměť (asi 16x) a čas dohledávání adres v poli. Pro větší kolekce jsou optimalizace nepostradatelné.

4.2.2 Výpočet Pageranku

Několik rozdílných odkazů vás mohou zavést na stejné místo. Pagerank se tak musí přizpůsobit a URL adresy přeměřující na jiné, bere v datastruktuře jako další objekt, mající jedinou hranu.

Jednou nabízející se otázkou je *kdy přestat?* Pagerank se počítá iterativně, ale má logaritmické škálování coby počtu potřebných iterací k minimalizaci výchylky. Implementace dává možnost volby přesnosti na základě přijatelné nejvyšší odchylky hodnoty během poslední iterace – stane se „stabilním"a konverguje anebo dosáhne předem definovaného limitu.

```
private fun globalSinkRank(): Double =
    allLinks.sumOf {
        if (it.forwardLinkCount == 0) it.rank[0] else 0.0
    }

private fun computePagerankOnDoc(
    doc: PagerankPage, sinkRank: Double
    ): Double =
    (1 - d) / allLinks.size + d * (doc.backLinks.sumOf {
            it.rank[0] / it.forwardLinkCount
        } + sinkRank / allLinks.size)
```

Funkce `computePagerankOnDoc` spočítá Pagerank, pro zpřehlednění dle notace

$$\mathrm{PR}(u) = \frac{1-d}{N} + d \sum_{v \in B_u} \frac{\mathrm{PR}(v)}{\mathrm{V}(v)},$$

která obsahuje *damping factor*, d. Tato úprava zamezí prvkům ke konverzi k hodnotě 0.0, znamenající nulovou pravděpodobnost návštěvy stránky. Běžně se užívá hodnota $d = 0.85$. Počet všech dokumentů je zaznačen jako N.

4.2.3 Zpracování do indexovatelné podoby

Samotná indexace probíhá parsováním HTML dokumentů repositáře do struktury `Page`, vyobrazenou v příloze A. Anchor text se získává vyžádáním a zpracováním HTML backlinku. S pomyšlením na výkonnostní nároky je maximální počet indexovaných backlinků limitovaný na k dokumentů, seřazených dle jejich Pageranku s podmínkou, že délka *anchoru* musí splňovat kritérium $3 < j < 72$ znaků.

Texty na stránce si nejsou rovny na poli relevantnosti. Klíčem k úspěchu je vybrání podstatných odstavců `<p>` v co největší míře. Má-li stránka správně definované meta tagy, výběr je jednoduchý. V opačném případě se výběrový algoritmus buď musí spoléhat na předem definovanou šablonu nebo na záložní *ad hoc* algoritmus vybírající určité paragrafové odstavce. Ve vyhledávání se však klade na celkový text jen nízká hodnota.

4.2.4 Zpracování v RAM vs. na disku

Veškeré zpracování daných dokumentů lze provést buď nahráním grafu do operační paměti nebo počítáním z databáze. Platí mezi nimi kompromis rychlosti a šetrnosti k paměti RAM. Po odzkoušení obou z metod, byl jednotně odsouhlasen způsob s operační pamětí. S několika optimalizacemi, které jsou na místě, zanechává program s grafem o velikosti 35k prvků paměťovou stopu ∼600MB, ale vede si přibližně 100 milionkrát lépe na poli času potřebného ke zpracování.

4.3 Elastic query

Vyhledávání v Elasticsearch je stěžejní částí podávání výsledků a správná podoba jak inicialního zmapování, tak dotazu je nutná pro co nejlepší výsledky.

4.3.1 Mapování

Elasticsearch při vzniku nového indexu nevyžaduje pevně danou datovou strukturu a nová datová pole si dokáže odvodit *dynamicky* a přiřadit jim základní datové typy. *Explicitní mapování* přichází v úvahu v případě nutnosti pokročilých indexačních a vyhledávacích funkcí Elasticsearch. Ty se musí definovat předem. USE mapování odlišuje mezi typy `text`, `keyword`, `rankFeature` s pozitivním a negativním dopadem a typy spadající pod *numbers*. Za mapováním tak stojí více uvážení, a jsou zcela odlišné oproti programovacím jazykům. K zápisu byl vytvořen speciální *domain specific language*, který působí jako *wrapper* a simplifikuje tvorbu zakomponováním Java metod idiomatickým Kotlin stylem, nabízející expresivnější a daleko přehlednější zápis.

Rozdíl mezi použití `text` a `keyword` závisí na využití. Může-li slovo nebo textový celek být zpracován pro fulltext vyhledávač a tím pádem nabývat neexaktní podoby, využívá se `text`. V opačném případě, kdy se jedná o libovolný řetězec znaků, který má pevně danou podobu, například URL, musí se použít `keyword`. Nakonec, typ `rankFeature` indexuje čísla tak, aby se dala zužitkovat na posílení výsledného skóre dokumentu. Je potřeba předem určit zda-li bude posílení pozitivní anebo negativní. Výhodou je, že jedno pole může být indexováno nejedním způsobem, bez potřeby jeho duplikace.

Za zmínku stojí, že USE indexuje všechen text ze stránky, ideálně pomocí `article` tagu. Taky obsahuje řadu možných ranků, které ve výsledku ovlivňují výsledné skóre dokumentu, každý svou vlastní váhou.

Upravená podoba poukazující na strukturu mapování je zdokumentována v přílohu A

4.3.2 Tvorba dotazu

Jakmile dotaz uživatele dosáhne Search manager, musí se převést z podoby lidské řeči do Elasticsearch dotazu, který zahrnuje všechna pole dokumentu s výjimkou `url.url`. Různá pole jsou vázána odlišnou hodnou. Velkou váhu mívají pole jako `content.anchors`, `content.title` a `content.headings.h1`, z ranků pak `smartRank` (Pagerank, kde $\sum PR = n$).

Ladění váh

Přesnou váhu polí však vyjádřit nelze, protože s každým datasetem je odlišná. Es dotazy obsahují příliš mnoho proměnných a tím pádem je docela náročné definovat je ručně. Například při zadání vícevýznamového dotazu „Wikipedia"je dobré zobrazit úvodní stránku i přes to, že v článku o Wikipedii, na Wikipedii, se vyskytuje dané klíčové slovo vícekrát. Měřením úspěšnosti (Kapitola 2.3) a patřičné upravování těchto hodnot vede ke zvýšení přesnosti navrácených výsledků.

Řešení by mohl přinést AI systém s automatickým přidělováním hodnot na základě ukázkových chtěných výsledků, redigovaných lidským kurátorem.

4.4 Člověk na pomoc autonomnímu systému

Web byl vytvořen lidstvem, určeným pro jeho vlastní spotřebu. Je tak logické, že se v něm vyzná nejlépe autentická lidská bytost. Pro velikost webu by bylo nutné skloubit kreativní dovednosti člověka s monotónii algoritmů například pro odhalování „junk"a „spam" stránek nebo napomoci se značkováním užitečných odstavců.

Kapitola 5

Limitace projektu USE

Web vyžaduje vedle velké algoritmizace i zdatnou výpočetní sílu. I přesto, že byl USE vytvořený s myšlenkou pro crawly a hledání i ve středním měřítku, skutečná proveditelnost dostupného hardware neodpovídá těmto nárokům. Vzhledem k rozsahu webu a vyzkoušených výsledků byl záměr z indexování celého webu přeorientován na hrstku předem zvolených domén, s dobře známou strukturou HTML. Při srovnání grafů 4.1 a 4.2 je patrné, že si algoritmy lépe poradí se zpracováním těchto domén. USE se tak plně přestává spoléhat na definice HTML tagů a výsledky konečnému uživateli mohou být zobrazeny dle dané šablony, zužitkující HTML extrakci užitečných informací na místě, bez nutnosti prokliku. Mezi nimi se mohou specifikovat a odfiltrovat, z informačního hlediska, zbytečné URL dle podoby jejich cesty a to ještě před *scrapem*.

Výsledkem je pak „vědecký"či „studentský"vyhledávač v omezeném měřítku, který se na základě nově získaných dat z crawlingu zlepšuje ve svém účelu, díky novým poznatkům o podobě grafové struktury, využitelný k počtu Pageranku a kvalitnějšího určení backlinku.

Zde se nachází současný seznam domén, které jsou momentálně podporovány:

- en.wikipedia.org
- goodreads.com
- ncatlab.org
- britannica.com
- infoplease.com
- scholarpedia.org
- deletionpedia.org

Kapitola 6

Využité technologie

Během vývoje a k uvedení do provozu bylo využito nejnovějších verzí cutting edge technologií.

6.1 Kotlin a JVM

Java virtual machine je virtuální stroj, definovaný abstraktními specifikacemi pro každý systém, který umožňuje spuštění programů zkompilovaných do podoby *Java bytecode*, bez ohledu na danou platformu.

Mezi moderní jazyky, zpřehledňující napsaný kód a urychlující psaní kódu, stavěný na JVM, se řadí právě Kotlin. Díky této nadstavbě, i přes jeho novotu, nabízí rozsáhlý ekosystém, koexistující s jinými, již zavedenými jazyky JVM. S asynchronními HTTP frameworky jako *Ktor* nebo knihovnou *Kotlinx.coroutines* jsou servery responsní při zvládání velkého náporu dotazů.

6.1.1 Coroutines vs threads

Vícevláknová práce s holými jádry představuje řadu problémů a jejich správa se koná manuálně. *Coroutines* na druhou stranu mezi sebou kooperují, pozastavováním a znovu se ujímáním výpočtů v daných bodech, volitelně na několika jádrech.

Coroutines jsou *light-weight* – kód, který by vyčerpal veškerou dostupnou JVM paměť s *threads*, *coroutines* si poradí s velmi nízkou paměťovou stopou:

```
fun main() = runBlocking {
    repeat(100 000) { // launch a lot of coroutines
        launch {
            delay(5000L)
            print(".")
        }
    }
}
```

Ukázkový kód spustí 100k *coroutines*, kde každá vyčká 5 sekund a poté vyšle do konzole „.”.

6.2 Elastic stack

"Elasticsearch – You Know, For Search."

Elastic stack se skládá ze tří programů:

- Elasticsearch
- Kibana
- Logstash

Elasticsearch slouží jako nesmírně schopná databáze. Její stavba vycházející z *Apache* znamená schopnost horizontálního škálování přes několik serverů a manipulaci se stovkami petabytů dat, zatímco stále stíhá zpracovat „kajiliony" dotazů za vteřinu. Vyhledávání v databázi se neomezuje, z pokročilých funkcí, pouze na fulltext, ale nabízí geo hledání s optimalizovaným *K-D-B-tree*s algoritmem a jeho schopnost, spolu s *Kibana*, analyzovat obrovské rámce dat v reálném čase je impresivní. Dané množství funkcí se neobejde bez svých slabin. Elasticu trvá update indexu určitou dobu, běžně však v jednotkách sekund.

6.3 Puppeteer a Chromium

Volba Puppeteer pro účely scraperu byla učiněna díky funkci *headless* scrapování a schopností spustit veškerý JS stránky. Jeho rozhraní se píše v Javascriptu. USE využil *Typescript*, zkompilovaný do JS a *Express* pro přijímání dotazů. Jádro Puppeteer je pak implementování v Chromium.

6.4 MongoDB

Databáze volby pro Repository se stala MongoDB. Podoba dokumentu nemusí být, během indexování zahrnuta, což do jisté míry ulehčuje vývoj. Pro komunikaci s Kotlinem byla zvolena knihovna KMongo s nativní podporou knihovny `kotlinx.coroutines` a nativním typovým systémem a syntaxí.

```
# Poznámka: Mapování v repozitáři odpovídá objekt Page

# Navrácení stránky, jejíž url se rovná požadované url
col.find(Page::finalUrl eq url)

# Navrácení náhodné stránky splňující crawovací požadavky
col.aggregate<Page>(listOf(sample(size),
    match(Page::statusCode eq code)))
```

Daný typový systém snižuje riziko chyb a plně spolupracuje se sadou nástrojů poskytnutých *IDE*.

6.5 Docker a Docker compose

Docker je nástroj určený ke zjednodušení vývoje, nasazení a spouštění aplikací použitím kontejnerů. Umožňuje kontejnerizaci jednotlivých komponentů do jednoho prostředí, které běží stejně, nezávisle na systémových specifikacích stroje.

Docker compose umožňuje zabalit několik Docker kontejnerů dohromady a zajistit jejich vzájemnou komunikaci. Nasazení aplikace je pak otázkou jediného CMD příkazu.

6.6 Typescript, NextJS a Bun

Následující technologie zaujímají místo exkluzivně na front-endu a komunikační rozhraní se scraperem. Pro minimalizaci chyb během vývoje se využil Typescript díky inkorporaci statického psaní kódu.

Framework založený na Reactu, NextJS urychluje load-time aplikace s využitím *server side rendering* schopností. NextJS, mezi jinými funkcemi, nabízí jak statické generování stránky, tak dynamické. S optimalizacemi zachází až do bodu, kdy statické stránky lze uvést do provozu hned na několika serverech zároveň, po celé zeměkouli, aby se *initial page-load time* držel minima, bez ohledu na lokaci spotřebitele; tzv. *Edge Funkce*. Dynamické stránky, na druhou stranu, mohou provést všechny HTTP dotazy v místě databázového serveru, eliminující tím dodatečnou cestu tam a zpět.

Nová alternativa pro spouštění Javascript kódu, Bun, přislibuje řádově lepší výkon, převážně pro HTTP dotazy a WebSocket dotazy. S Bun není ani potřeba transkompilovat Typescript kód do Javascriptu.

Závěr

Cílem projektu bylo vytvoření *demo* fulltext vyhledávací aplikace, rozdělenou do několika komponent, které obsahují svou vlastní architekturu implementující řešení svých překážek. Software crawleru se dokáže orientovat ve webové struktuře plně autonomně; servírování výsledků, jsou-li vyhledávány platné dokumenty, navrací relevantní výsledky v hojném zastoupení; suggester napomáhá koncovému uživateli při zadávání jeho dotazu s úvahou pro předcházející kontext; hinty doplňují zobrazené výsledky o dodatečné informace. Vše dostupné z jedné, přehledné, webové aplikace. Bleskurychle!

Mezi funkce se kterými se původně nepočítalo, ale ukázaly se patřičně užitečné, patří převážně Repository a oddělené zpracování dokumentů od hlavní Elasticsearch databáze.

Co se však nezdařilo je vyhledávání na otevřeném webu – čistě pro příliš vysokou ambicióznost.

Řešením projektu jsem nabyl všeobecného přehledu o vnitřním fungování search enginů; způsobech rozebírání kontextu vět do podoby srozumitelné počítačem; vhledu do struktury webu; setkal jsem se s proveditelnostními hardware limity; vše založené na podkladových matematických principech.

Možná vylepšení představují zpracování přirozeného jazyka (NLP), s tím související, ale nezávazné zdokonalení vybírání URL adres, ale především zkvalitnění vyhledávání. Další zdokonalení spočívá v optimalizacích a ladění hodnot nebo zahrnutí vícero metrik. V již nasazeném systému by mohly připomínat zpracování logů prostých uživatelů.

Bibliografie

[1] Junghoo Cho; Hector Garcia-Molina; Lawrence Page. "Efficient Crawling Through URL Ordering". Dis. pr. Stanford University, 1995.

[2] Sergey Brin a Lawrence Page. "The Anatomy of a Large-Scale Hypertextual Web Search Engine". Dis. pr. Stanford University, 1998.

[3] Gyöngyi Zoltán; Berkhin Pavel; Garcia-Molina Hector; Pedersen Jan. *Link Spam Detection Based on Mass Estimation*. Tech. zpr. Stanford University, 2006.

[4] Nupur Choudhury. "World Wide Web and Its Journey from Web 1.0 to Web 4.0". Dis. pr. Sikkim Manipal Institute of Technology, 2014.

[5] Libretexts. *5.4: Reading- The world wide web*. Lis. 2021. URL: https://workforce.libretexts.org/Bookshelves/Information_Technology/Computer_Applications/Introduction_to_Computer_Applications_and_Concepts_(Lumen)/05%3A_Communications_and_Information_Literacy/5.04%3A_Reading-_The_World_Wide_Web.

[6] Wikipedia. *Bayesian inference*. http://en.wikipedia.org/w/index.php?title=Bayesian\%20inference&oldid=1121819128. [Online; accessed 28 December 2022]. 2022.

[7] Wikipedia. *Degree distribution*. http://en.wikipedia.org/w/index.php?title=Degree\%20distribution&oldid=1088517578. [Online; accessed 28 December 2022]. 2022.

[8] Wikipedia. *Directed graph*. http://en.wikipedia.org/w/index.php?title=Directed\%20graph&oldid=1110855150. [Online; accessed 28 December 2022]. 2022.

[9] Wikipedia. *Elasticsearch*. http://en.wikipedia.org/w/index.php?title=Elasticsearch&oldid=1124065385. [Online; accessed 28 December 2022]. 2022.

[10] Wikipedia. *Full-text search*. http://en.wikipedia.org/w/index.php?title=Full-text\%20search&oldid=1121602001. [Online; accessed 28 December 2022]. 2022.

[11] Wikipedia. *Graph (discrete mathematics)*. http://en.wikipedia.org/w/index.php?title=Graph\%20(discrete\%20mathematics)&oldid=1124522642. [Online; accessed 28 December 2022]. 2022.

[12] Wikipedia. *Graph theory*. http://en.wikipedia.org/w/index.php?title=Graph\%20theory&oldid=1127936693. [Online; accessed 28 December 2022]. 2022.

[13] Wikipedia. *Information retrieval*. http://en.wikipedia.org/w/index.php?title=Information\%20retrieval&oldid=1129301216. [Online; accessed 28 December 2022]. 2022.

[14] Wikipedia. *PageRank*. http://en.wikipedia.org/w/index.php?title=PageRank&oldid=1129606045. [Online; accessed 28 December 2022]. 2022.

[15] Wikipedia. *Precision and recall*. http://en.wikipedia.org/w/index.php?title=Precision\%20and\%20recall&oldid=1122267443. [Online; accessed 28 December 2022]. 2022.

[16] Wikipedia. *Search engine*. http://en.wikipedia.org/w/index.php?title=Search\%20engine&oldid=1129725839. [Online; accessed 28 December 2022]. 2022.

[17] Wikipedia. *Search engine indexing*. http://en.wikipedia.org/w/index.php?title=Search\%20engine\%20indexing&oldid=1129248567. [Online; accessed 28 December 2022]. 2022.

[18] Wikipedia. *Stematizace*. http://cs.wikipedia.org/w/index.php?title=Stematizace&oldid=20367163. [Online; accessed 28 December 2022]. 2022.

[19] Wikipedia. *Stopslovo*. http://cs.wikipedia.org/w/index.php?title=Stopslovo&oldid=20628741. [Online; accessed 28 December 2022]. 2022.

[20] Wikipedia. *String-searching algorithm*. http://en.wikipedia.org/w/index.php?title=String-searching\%20algorithm&oldid=1128920188. [Online; accessed 28 December 2022]. 2022.

[21] Wikipedia. *Vertex (graph theory)*. http://en.wikipedia.org/w/index.php?title=Vertex\%20(graph\%20theory)&oldid=1124012466. [Online; accessed 28 December 2022]. 2022.

[22] Wikipedia. *Web crawler*. http://en.wikipedia.org/w/index.php?title=Web\%20crawler&oldid=1124235168. [Online; accessed 28 December 2022]. 2022.

[23] Wikipedia. *Web design*. http://en.wikipedia.org/w/index.php?title=Web\%20design&oldid=1129100925. [Online; accessed 28 December 2022]. 2022.

[24] Wikipedia. *Webgraph*. http://en.wikipedia.org/w/index.php?title=Webgraph&oldid=1095229186. [Online; accessed 28 December 2022]. 2022.

[25] Marchiori Massimo. *The Quest for Correct Information on the Web: Hyper Search Engines*. URL: https://www.w3.org/People/Massimo/papers/WWW6/.

[26] *More than just a Web Search algorithm: Google's PageRank in non-Internet contexts*. URL: https://blogs.cornell.edu/info2040/2014/11/03/more-than-just-a-web-search-algorithm-googles-pagerank-in-non-internet-contexts/.

Příloha A

Podoba Elasticsearch dokumentu

```
Page: {                                     // Typ v indexu:
    url: {
        url: String                         // keyword
        urlPathKeywords: List<String>       // text
        hostName: String                    // text
    }
    ranks: {
        pagerank: Double                    // rankComplex
        smartRank: Double                   // rankComplex

        urlLength: Int                      // pnRankComplex
        urlPathLength: Int                  // pnRankComplex
        urlSegmentsCount: Int               // pnRankComplex
        urlParameterCount: Int              // pnRankComplex
        urlParameterCountUnique: Int        // pnRankComplex
        totalUrlDocs: Int                   // pnRankComplex
    }
    content: {
        title: String                       // text
        description: String                 // text
        keywords: List<String>              // text
        anchors: List<String>               // text
        boldText: List<String>              // text
        headings: {
            h1: List<String>                // text
            h2: List<String>                // text
            h3: List<String>                // text
            h4: List<String>                // text
            h5: List<String>                // text
            h6: List<String>                // text
        }
        text: List<String>                  // text
    } }
```

Na míru definovaný typ `rankComplex` se sestává z typů *double* a *rankFeature*, zatímco `pnRankComplex` dodává možnosti využít rank negativně.

Seznam obrázků