STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor č. 18-20-M/01: INFORMAČNÍ TECHNOLOGIE se zaměřením na počítačové sítě a programování

Ultimate Search Engine

Filip Hostinský, Vojtěch Theodor Binar, David Stoček
Moravskoslezský kraj Opava 2023

STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor č. 18-20-M/01: INFORMAČNÍ TECHNOLOGIE se zaměřením na počítačové sítě a programování

Ultimate Search Engine

Autoři: Filip Hostinský, Vojtěch Theodor Binar, David Stoček
Škola: Střední škola průmyslová a umělecká, Praskova 399/8, 746 01, Opava
Kraj: Moravskoslezský kraj
Konzultanti: Bc. Filip Peterek, Ing. Petr Grussmann, Mgr. Marek Lučný

Opava 2023

Prohlášení

Prohlašuji, že jsem svou práci SOČ vypracoval/a samostatně a použil/a jsem pouze prameny a literaturu uvedené v seznamu bibliografických záznamů.

Prohlašuji, že tištěná verze a elektronická verze soutěžní práce SOČ jsou shodné.

Nemám závažný důvod proti zpřístupňování této práce v souladu se zákonem č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších předpisů.

Filip Hostinský

Vojtěch Theodor Binar

V Opavě dne 30. března 2023

David Stoček

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 Theodoru Binarovi za spolupráci na tomto projektu.

Abstrakt

Tato práce se především zabývá analýzou dnešního webového prostředí a optimistickou teorií pro upřednostňování důležitých stránek, která si zakládá na jeho hypertextové podstatě, primárně využitím algoritmu *Pagerank*. Dále se snaží pokrýt velice komplexní vyhledávací mechanismus s databází 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 zpracování *HTML* do podoby, v které lze vyhledávat a tvorbou algoritmů spojených s vyhledáváním. Výsledkem je *studentský* vyhledávač postavený na základě *Elasticsearch* databáze, který dokáže odpovědět na dotaz uživatele relevantnějšími stránkami a kde uzná za vhodné, doplňuje 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; Crawler; Big data

Abstract

This paper primarily focuses on the analysis of the current web environment, and presents an optimistic theory for prioritizing important pages, based on the hypertext nature of the web, utilizing the *Pagerank* algorithm. Additionally, it covers the complex search mechanism of the database at the core of the web engine, and provides an overview of the engine's anatomy, composed of a series of individual components that are explained in detail.

The practical part addresses the issue of effectively processing *HTML* into a searchable format and developing algorithms related to search. The outcome is a basic search engine that utilizes an Elasticsearch database, which can provide more relevant search results to users and, where necessary, supplement those results with additional information. In addition, the search engine simplifies the search process by presenting relevant suggestions and hints, through utilities that were introduced for this purpose.

Keywords

Search Engine; Web; Crawler; Big data

Obsah

Úvod

Informací na webu dennodenně rapidně přibývá. Podle nejlepších odhadů existuje přibližně 1 miliarda domén, obhospodařujících téměř 18 triliard ($1.8 * 10^{16}$) 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 dokáže nejen spojit uživatele s jejich oblastmi zájmů, ale také seřadit obsah podle kritérií, jako jsou například důvěryhodnost zdroje, aktuálnost, kvalita a kontext. Současné nejpokročilejší systémy využívají takovýchto hodnot řádově v počtu několika stovek.

Vyhledávání je neodmyslitelnou součástí všech rozsáhlejších webových stránek. Přestože jde často o nenápadnou komponentu, ve skutečnosti jsou za ní velice komplikovaná řešení. Lákalo nás proniknout do hloubky této problematiky a rozhodli jsme se vytvořit vlastní studentský vyhledávač.

Našim cílem bylo vytvořit plně automatický a centralizovaný search engine v omezeném měřítku, který si sám poradí s běžnými překážkami a nástrahami webu. Výsledkem naší práce je systém, který sám dokáže crawlovat web, indexovat data a následně běžnému uživateli prezentovat užitečné výsledky pomocí přívětivé webové aplikace.

Dokumentace byla rozvržena do tří hlavních částí. Nejprve představuje teoretické zázemí a popis využitých algoritmů (kapitoly 1 a 2), pokračuje vysokoúrovňovým pohledem na celkovou anatomii (kapitola 3) a poslední, nejrozsáhlejší část pojednává o praktickém využití předešlých poznatků s výběrem nejpalčivějších výzev systému, které bylo nutno vyřešit (kapitoly 4–7).

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. WWW si můžeme představit jako matematický model několika *konečných* grafů, kde *prvek* představuje webovou stránku a odkazy působí jako *hrany*. Musíme brát v potaz, že takových odpojených grafů může existovat několik—například blogy nebo stránky se zpoplatněným obsahem apod.

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, než 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. Nejsou-li již indexované stránky přeindexované, search engine se může stát nepoužitelným.

Při crawlování je potřeba provést rozhodnutí, jestli zabírat široké spektrum domén a klást důraz na rozmanitost, nebo omezit crawl na kýženou část internetu a dosáhnout tím konkrétnějšího využití. Procesem *efektivního* crawlování a zpracování takového grafu se zabýváme v části 3.1 a 4.1).

1.3.2 Správné využití tagů

Meta tagy tvoří metadata HTML stránky. Jsou pouze doporučené a nemusí se vyskytovat na všech stránkách. A pokud ano, mohou nabývat několika podob, znamenající totéž, anebo být dokonce nesprávné. Systém pak musí určit a správně vyhodnotit pravdivost, případně se pokusit o doplnění 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 obsahu. Stránky snažící se takto zaujmout koncového uživatele musí search enginům nabídnout relevantní a kompletní meta tagy.

Kapitola 2

Fulltext mechanismus zpracování

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 datové struktuře *slovníku*, která drží hodnoty tokenů jako *klíče*, ukazující na konkrétní dokumenty. Vyhledávání se pak stává snadným, jelikož se stačí pouze odvolat na index klíčů místo 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 podle kmenového výrazu slova.

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

2.2 Pagerank

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, které na něj odkazují (tzv. *hrany*). To funguje tak, že prvek odkazující, v, uváží svůj rank ($PR(v)$) vůči celkovému počtu hran, na které 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.

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

To v praxi znamená, že podstatně výhodnější je být vázaný menším počtem silných hran, oproti velkému počtu slabých hran. Jak již bylo zmíněno, v kontextu search enginů *prvek* představuje stránku a *hrana* 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 N-gramový jazykový model

Jedná se o způsob jak předpovídat pravděpodobnost výskytu slova v textu. *Ngramy* jsou skupiny n sousedních slov v textu, kde pravděpodobnost výskytu slova závisí na slovech předchozích. Tyto modely jsou často používány k předpovídání slov při zadávání textu, automatickému překladu a dalších úloh zpracování přirozeného jazyka. Čím více předchozích slov do výpočtu se zahrne, tím dosahujeme vyšší pravděpodobnosti výskytu, avšak za cenu vetší zátěže systému.

2.3.1 Bigram model

Jedná se o typ jazykového modelu, ve kterém se předpokládá, že pravděpodobnost výskytu daného slova závisí pouze na předchozím slově. Jsou to tedy skupiny dvou sousedních slov v textu. Například ve frázi „dnes je hezké počasí" by bigramy zahrnovaly „dnes je", „je hezké" a „hezké počasí". Model může být také použit k vytvoření *bigramového slovníku*, ve kterém jsou uvedeny všechny bigramy v daném textu a jejich frekvence výskytu. Díky jeho nízké náročnosti je vhodným kandidátem pro využití do algoritmu našeptávače.

2.3.2 Markovův předpoklad

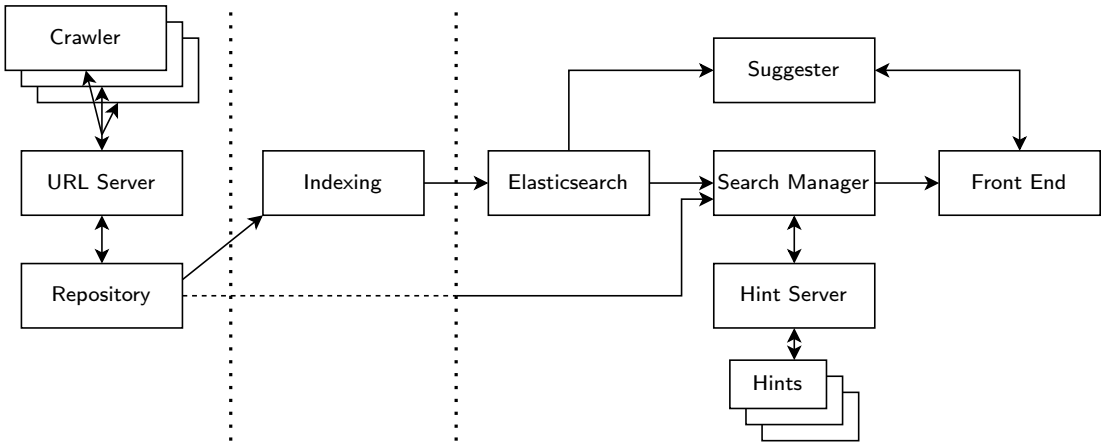
Určuje, že budoucí stav systému závisí pouze na současném stavu a na předchozím stavu, ale ne na všech předcházejících stavech. To znamená, že při určování pravděpodobnosti budoucího stavu lze ignorovat všechny stavy, které byly nastaveny před několika kroky.

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 upřednostní před jinými. To představuje určitý problém, protože bez znalosti celé sítě 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. Také určuje rychlost crawlování jednotlivých domén. Některé z nich totiž implementují *rate-limiting*. Tyto a jiné problémové domény si poznamenává.

3.1.2 Crawler

Způsob scrapování 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 datové struktuře vyhledávače je zapotřebí provést pouze interní přepracování, namísto přecrawlování celého webu. To ve výsledku nesmírně usnadňuje vývoj a experimentování 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 dohodnuté podobě do indexu Elasticsearche. Zde se, mimo jiné, uplatňuje teorie *anchor textu* backlinků (viz 1.2.1).

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 výsledky vyhledávání. Má na starost správnou agregaci požadovaného dotazu a přeložení do dotazovacího jazyka Elasticsearch. Výsledky v Elasticsearch nemusí být plně spolehlivé vzhledem k aktuálnímu dotazu, proto je nutné analyzovat původní HTML stránku a dohledat relevantní informace. Podrobněji o dotazování v Elasticsearch viz 4.4.

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 se slovy, které lze použít ve vyhledávači, s ohledem na předchozí kontext.

Našeptávače mohou být založeny na různých zdrojích, jako jsou například historie vstupů uživatele, slovníky nebo databáze. Mohou být také založeny na pravidlech a algoritmech, které určují nejrelevantnější nebo nejpravděpodobnější návrhy v daném kontextu. Algoritmus USE pro tyto našepty využívá *n-gramový* model, popsaný blíže v sekci 2.3, zatímco průběh zpracování je detailně popsaný v sekci 4.3.

Obecně jsou považovány za užitečné nástroje pro zjednodušení a zrychlení vstupu uživatele, ale mohou také představovat bezpečnostní riziko, pokud jsou špatně navržené nebo implementované. Je tedy důležité jejich správné navržení, aby nedocházelo k porušení soukromí nebo bezpečnosti.

3.3.3 Hint server a hinty

Velice časté pro základní utility jsou hinty. Jedná se o řadu jednoduchých programů, které jsou dostupné na stránce s výsledky. 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 komunikaci mezi klientem a serverem. 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 server.

Kapitola 4

Výzvy a řešení

4.1 Efektivní crawlování

Správný přístup k získávání stránek je zásadní 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. Pro naše řešení jsme využili model založený na Pagerank. Tento model stanoví pro každý prvek grafu exaktní hodnotu podle důležitosti stránek. Sémantický kontext nebyl brán v potaz.

4.1.1 Jazyk

Meta tag označující jazyk stránky může být ošemetný. Ani nejvýznamější stránky nutně neobsahují jazykový tag. Ideální systém by měl počítat s nespolehlivostí daného tagu a vždy se snažit ověřit, případně dopočítat správnost výroku. Aby byl URL Server maximálně efektivní, nesmí ztrácet čas na jinojazyčných stránkách a zamezit jejich dalšímu zpracování již během scrapingu.

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 ∼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 ještě dříve, před pokusy o prohledávání do 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.

Většinou 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 scrapuje 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, přičemž mnohé z nich nebyly určeny k prokliku, což způsobilo nával nepoužitelných stránek. Další problémy byly spojeny s nadměrným zacházením do hloubky.

Pozdějším modelem byl *Plain Pagerank*, který zúročil předcházející výpočet Pageranku ve svůj prospěch. Doména, jíž 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 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 nahodilosti je méně pravděpodobné, aby se algoritmus stal posedlý jednou „nekonečně" velkou částí internetu a vynechal všechny ostatní.

Během crawlingu, při vybírání následující adresy, je zapotřebí odlišit mezi dvěma architekturami: *bez opakování* a *s opakováním*. Jde o opětovný průzkum a zahrnutí již zpracovaných stránek, byť jen symbolicky z repositáře. Grafy 4.1 a 4.2 dokazují mírnou nadřazenost právě algoritmu s opakováním.

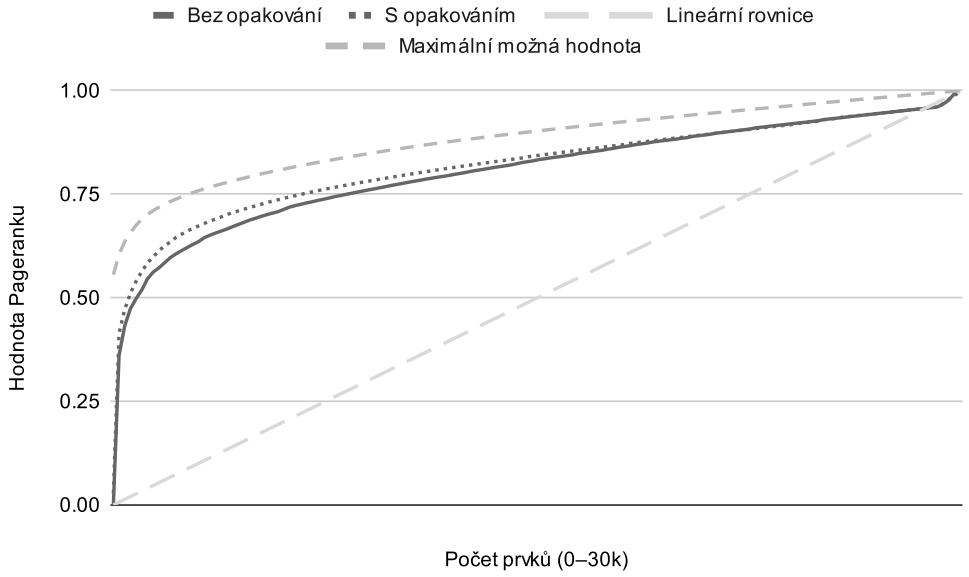
S ohledem na grafy 4.1 a 4.2 se můžeme přesvědčit o efektivitě modelu OP vůči nejvyšší možné hodnotě, které lze dosáhnout a lineární funkci, která předurčuje výsledek zcela náhodného modelu. Otevřený web představuje větší výzvu, protože neobsahuje jeden centrální bod ve srovnávání s omezení vzorku na 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.

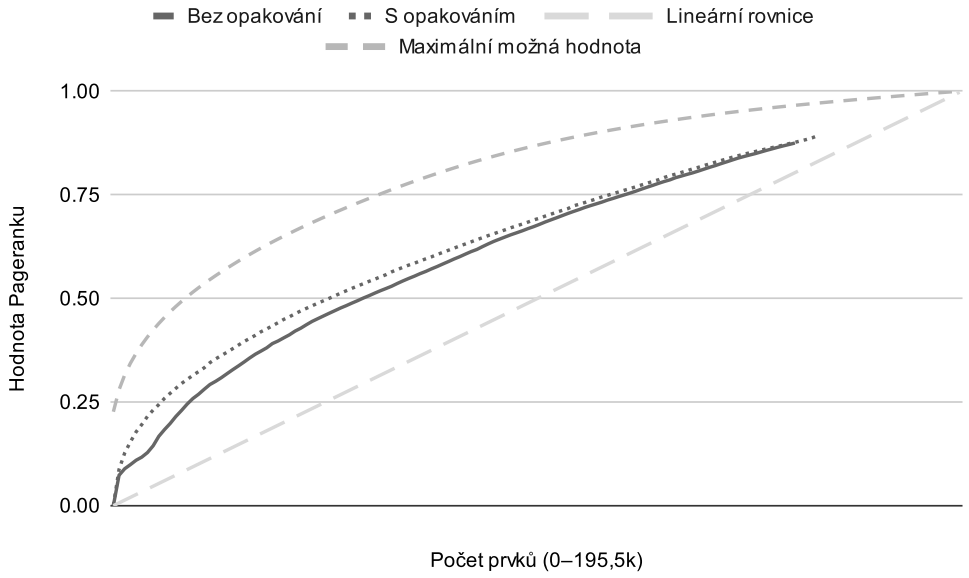
4.2 Technická proveditelnost zpracování dat

4.2.1 Stavba webgrafu

Ukázalo se, že tvorba grafu může být velice náročná. S ohledem na co nejkratší paměťovou stopu je rozdělen do dvou částí se záměrem 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.



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).

```
# 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, kterou bude možno poskládat podobu grafu k výpočtu, 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 pracuje hospodárněji s operační pamětí, do objektové struktury `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 datová struktura 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 mohou vést na stejné místo. Pagerank se tak musí přizpůsobit a URL adresy přeměřující na jiné, bere v datové struktuře jako další objekt, mající jedinou hranu.

Jednou nabízející se otázkou je *kdy přestat?* Pagerank se počítá iterativně, ale má logaritmické škálování 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 relevantností. 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 odstavce. Ve vyhledávání se však klade na celkový text jen nízký důraz.

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 Průběh zpracování dat našeptávače

4.3.1 Tvorba vstupních dat

Aby mohl algoritmus správně fungovat, musíme nejprve pro něj vytvořit správný datový set. Protože datový set nejbližší kontextu webu je stěžejní součástí relevantních našeptů, jsou využívány již přeparsované záznamy (popsány v sekci 4.2.3).

Dále je nutností taková data dále zpracovat rozdělením textu na jednotlivé větné celky. Jelikož se USE zaměřuje výhradně na texty psané latinkou, tokenem rozdělení se stává „.". S převedením všech velkých písmen na malá a při zohlednění aditivních mezer, získáváme set tokenů daného dokumentu, připraveného k dalšímu využití.

4.3.2 Získání slov závislých

Po získání počtu výskytů jednotlivých tokenů již stačí získat poslední dílek pro použití algoritmu. Ke každému tokenu je potřeba přiřadit ostatní související tokeny. To znamená najít n slov nacházejících se před daným slovem a spočítat četnost výskytu takovéto fráze.

4.3.3 Počet jednotlivých slov

Nyní zbývá všechna data spojit dohromady, a tak získat pravděpodobnost pro výskyt daného slova na základě přecházejícího řetězce tokenů, $P(w_n|w_{n-1})$. Ta se získává dosazením závislého a příslušného vyhledávaného tokenu vzorcem

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})}.$$

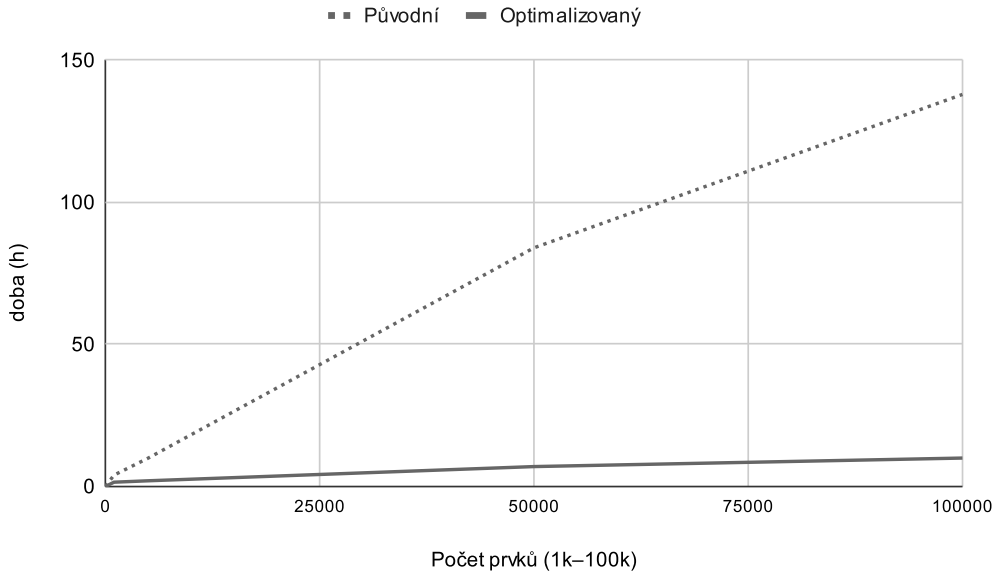
Notace $C(w_{n-1}w_n)$ udává počet výskytu daného bigramu, tedy kolikrát token w_n následoval w_{n-1}. Konečná pravděpodobnost pak vznikne podílem celkového množství výskytu předcházejícího tokenu (unigramu), w_{n-1}, nezávisle na jeho kontextu.

4.3.4 Indexace

Pro našeptávaní bylo opět využito funkcí Elasticsearch, právě díky jeho pokročilým možnostem rankování výsledků. Konečná zaindexovaná struktura dokumentu zahrnuje předchozích n tokenů, daný token a konečnou pravděpodobnost výskytu tokenu.

4.3.5 Optimalizace

Díky četným optimalizacím bylo možné zefektivnit algoritmus. V následující sekci jsou popsány nejvýznamnější z nich. Obrázek 4.3 popisuje výsledky tohoto úsilí.



Obrázek 4.3: Rozdíl rychlosti algoritmu před a po optimalizaci

Elasticsearch

První podstatnou optimalizaci představovalo využití možností Elasticsearch *Multi term vectors* API, oproti zpracováním čistě programovacím jazykem Python. Toto API nabízí statistiky všech slov vyskytujících se v dokumentu.

Hledání závislých slov

Počítat pravděpodobnost spojení slova s ostatními slovy by bylo příliš nehospodárné, tudíž nijak rafinovaná optimalizace spočívá v zaindexování pouze vyskytnutých spojených tokenů. Zkracuje se tak doba vytváření datasetu a výsledná velikost indexu.

Multiprocessing

Často je nutné volat velký počet funkcí a zpracovávat množství dat. K tomu je vhodně využít multiprocesing, tedy palalérní zpracování, což může výrazně zkrátit dobu výpočtu a díky *thread-safe* povaze výpočtu lze využít plný výpočetní potenciál stroje.

4.4 Elastic query

Klíčem pro úspěšné vyhledávání je databáze Elasticsearch, v níž je vhodné provést počáteční mapování

4.4.1 Mapování

Elasticsearch při vzniku nového indexu nevyžaduje pevně danou datovou strukturu. 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 či negativním dopadem a typy spadající pod *numbers*. K zápisu byl vytvořen speciální *domain specific language*, který

působí jako *wrapper* a zjednodušuje tvorbu zakomponováním Java metod idiomatickým Kotlin stylem, nabízející expresivnější a daleko přehlednější zápis.

Typ `text` se používá, když slovo nebo textový celek může být zpracován pro fulltextové vyhledávání a nemá danou strukturu. V opačném případě, kdy se jedná o libovolný řetězec znaků, který má pevně danou podobu, například URL, se musí použít `keyword`. Typ `rankFeature` indexuje čísla tak, aby se dala zužitkovat na posílení výsledného skóre dokumentu. Je potřeba předem určit, zdali bude posílení pozitivní nebo negativní. Výhodou je, že jedno pole může být indexováno několika způsoby, bez potřeby jeho duplikace.

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

Upravená podoba poukazující na strukturu mapování je zdokumentována v příloze A

4.4.2 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 `content.anchors`, `content.title` a `content.headings.h1`, z ranků pak nejvíce `smartRank` (Pagerank, kde $\sum PR = n$).

Ladění váh

Přesnou váhu polí však vyjádřit nelze, protože s každým datasetem je odlišná. Dotazy v Elasticsearch 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. Měření úspěšnosti 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.

4.5 Role člověka v autonomním 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 počítačových algoritmů, například pro odhalování „junk" a „spam" stránek nebo napomoci se značkováním užitečných odstavců.

Kapitola 5

Frontend - design a interakce

Naším záměrem bylo, aby stránky USE využívaly minimalistický design, za použití minima efektů, které by mohly uživatele vyrušovat od daného úkolu. Dosáhli jsme tím nekompromisní *time to interactivity*. Domovská stránka, díky využití nejmodernějších technik, se prezentuje rychlostí statického HTML, zatímco si ponechává *React-like* reaktivitu (blíže v sekci 7.7). Samozřejmostí je responsní design pro akomodaci mobilních zařízení.

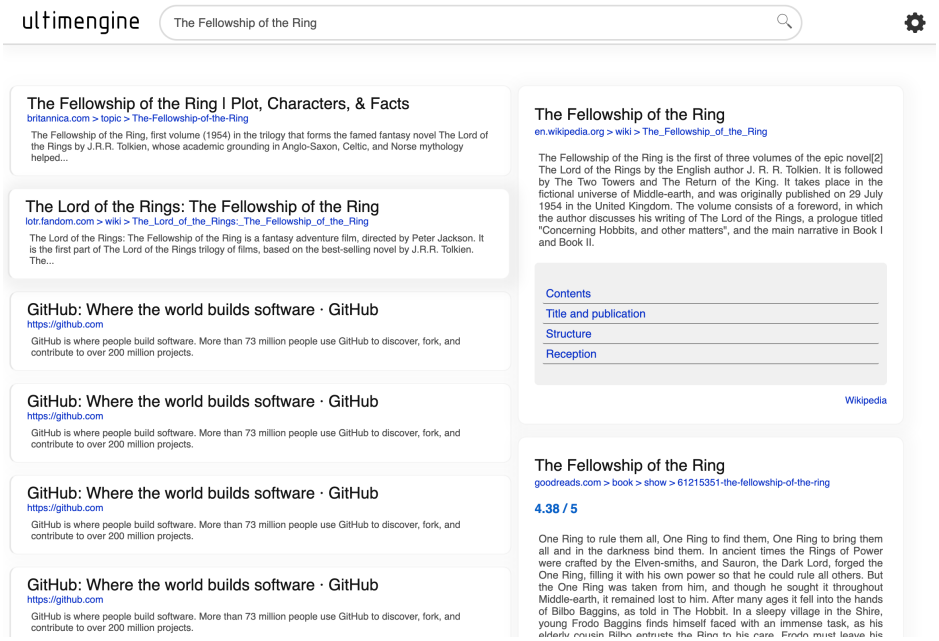
5.1 Domovská stránka

Domovská obrazovka nabízí jediné vstupní pole, motivující uživatele k jeho využití. Během zadávání požadovaného dotazu se pod vyhledávacím polem objeví našeptávač, který nabízí dokončení daného dotazu, s využitím *suggester serveru*. Ref. Obrázek 5.1.

Obrázek 5.1: Domovská obrazovka

5.2 Stránka s výsledky

Stránka s výsledky obsahuje dva sloupce. Pravý sloupec obsahuje „speciální", obohacené výsledky z podporovaných stránek. Jedná se o jednu z funkcí *hint serveru* a jednotlivých *hintů*. Ty nejsou omezené jen na obohacení, podporují také interaktivitu uživatele (např. kalkulačka).

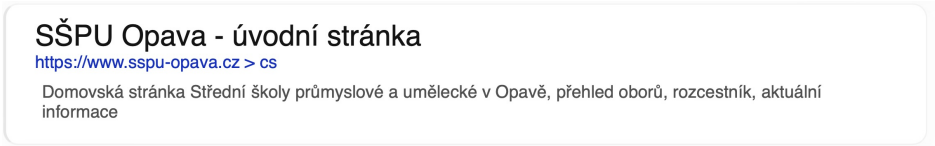


Obrázek 5.2: Stránka s výsledky

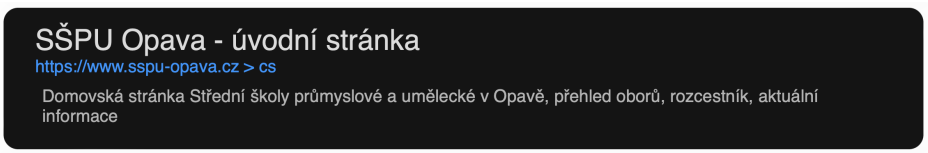
Jednotlivé stránky byly poskládány z několika vytvořených elementů, a proto jedné implementace určitého elementu lze opětovně využit, bez námahy, i zde. Funkčnost *search boxu* je díky tomu shodná s domovskou obrazovkou.

5.3 Úpravy vzhledu

Aplikace taktéž umožňuje přizpůsobit si barevné schéma k obrazu svému z mnoha nabízených motivů v sekci s nastavením.



Obrázek 5.3: Ukázka světlého barevného motivu



Obrázek 5.4: Ukázka temného barevného motivu

Kapitola 6

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 omezenému rozsahu webu a ověřených výsledků jsme se rozhodli pouze provádět indexování 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 zobrazuje uživateli podle dané šablony, která obsahuje výběr užitečných informací na jednom místě bez nutnosti prokliku. Předem jsou při tom odfiltrovány ty URL, které nenesou podstatné informace.

Výsledkem naší práce je tak „vědecký" či „studentský" vyhledávač zaměřený na získávání věrohodných informací z relevantních zdrojů, který zlepšuje své výsledky vyhledávání na základě crawlingu díky novým poznatkům o podobě grafové struktury.

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 7

Využité technologie

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

7.1 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. Díky asynchronnímu HTTP frameworky jako *Ktor* s knihovnou *Kotlinx.coroutines* dokážou zvládat velký nápor dotazů.

7.1.1 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 „.”.

7.2 Python

Python byl volbou pro našeptávač díky jeho srozumitelné syntaxi, velké komunitě a podpoře vícevláknového provádění instrukcí. Ke zpracování HTTP dotazů byla využita

rychlá a nenáročná knihovna FastAPI, která dokáže zvládat velký nápor souběžných dotazů vlivem podpory provádění kódu asynchronně.

7.3 Elastic stack

Elastic stack se skládá ze tří programů:

- Elasticsearch
- Kibana
- Logstash

Elasticsearch slouží jako nesmírně schopná databáze. Její stavba, vycházející z projektu *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 pouze na fulltext, ale například nabízí geo hledání s optimalizovaným *K-D-B-tree*s algoritmem a díky spolupráci s *Kibana* analyzuje obrovské rámce dat v reálném čase. Dané množství funkcí se neobejde bez svých slabin. Slabinou Elasticu může být update indexu, který může trvat v řádech jednotky sekund.

7.4 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.

7.5 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*.

7.6 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.

7.7 Typescript, SCSS, NextJS a NodeJS

Následující technologie zaujímají místo exkluzivně na front-endu a komunikační rozhraní se scraperem. Pro minimalizaci chyb během vývoje byl využit syntaktický superset Javascriptu, Typescript, pro inkorporaci statického psaní kódu. Obdobně, syntaktický superset CSS, SCSS, pro kvalitnější práci se stylováním.

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.

Pro javascript serverové prostředí využíváme standardní NodeJS, přestože jsou k dispozici rychlejší technologie.

Závěr

Cílem projektu bylo vytvoření studentské verze fulltextového vyhledávače, která zpracovává data z omezené množiny často využívaných informačních zdrojů.

Aplikaci tvoří několik svébytných, ale na sebe navazujících komponent, jež mají svou vlastní architekturu a řeší některý ze specifických problémů v procesu vyhledávání. Software crawleru se dokáže orientovat ve webové struktuře plně autonomně a v případě, že jsou vyhledány platné dokumenty, vrací relevantní výsledky. Suggester zohledňuje předcházející kontext a napomáhá koncovému uživateli při zadávání jeho dotazu; hinty doplňují zobrazené výsledky o dodatečné informace. Vše je dostupné prostřednictvím přehledné webové stránky, jejíž minimalistické řešení je nejen responzivní, ale vyhovuje i základnímu požadavku předkládat uživateli rychlé a validní informace. Součástí vyhledávače USE jsou i užitečné funkce, s nimiž se v původním návrhu nepočítalo - patří k nim Repository a oddělené zpracování dokumentů mimo hlavní Elasticsearch databázi.

Přestože jsme kvůli omezeným technickým možnostem museli odstoupit od původní vysoké ambice vyhledávat na otevřeném webu, získali jsme během řešení projektu všeobecný přehled o vnitřním fungování vyhledávacích strojů. V průběhu své práce jsme se dozvěděli mnohé o způsobech rozboru kontextu vět do podoby srozumitelné počítači, o samotné struktuře webu, museli jsme se potýkat s problémy souvisejícími s proveditelnostními limity hardware. A vše je pochopitelně založeno na matematických principech.

Možná budoucí vylepšení představují zpracování přirozeného jazyka (NLP) a 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] Elasticsearch. *API Documentation — Elasticsearch 7.16.0 documentation*. https://elasticsearch-py.readthedocs.io/en/7.x/api.html. [Online; accessed 28 December 2022]. 2022.

[7] Python Software Foundation. *Multiprocessing — Process-based parallelism — Python 3.11.1 documentation*. https://docs.python.org/3/library/multiprocessing.html. [Online; accessed 28 December 2022]. 2022.

[8] Computer Hope. *What is a Search Engine?. Computer Hope's Free Computer Help [online]*. https://www.computerhope.com/jargon/s/searengi.htm. [Online; accessed 28 December 2022]. 2022.

[9] Docker Inc. *What is a Container? — Docker: Accelerated, Containerized Application Development*. https://www.docker.com/resources/what-container/. [Online; accessed 28 December 2022]. 2022.

[10] Tiangolo. *FastAPI Documentation*. https://fastapi.tiangolo.com/. [Online; accessed 28 December 2022]. 2022.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[30] Daniel Jurafsky; James H. Martin. „N-gram Language Model“. Dis. pr. Stanford University, 2023.

[31] Microsoft. *Typescript documentation*. https://www.typescriptlang.org/docs/. [Online; accessed 29 March 2023]. 2023.

[32] Sass team. *Sass documentation*. https://sass-lang.com/documentation/. [Online; accessed 29 March 2023]. 2023.

[33] Vercel. *NextJS*. https://nextjs.org/. [Online; accessed 29 March 2023]. 2023.

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

[35] *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