VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

INTELIGENTNÍ EXTRAKCE DAT VE WEBOVÉM PROHLÍŽEČI

INTELLIGENT DATA SCRAPING IN A WEB BROWSER

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **FRANTIŠEK MAŠTERA**
AUTHOR

VEDOUCÍ PRÁCE **Ing. RADEK BURGET, Ph.D.**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2020/2021

Zadání bakalářské práce

Student: **Maštera František**
Program: Informační technologie
Název: **Inteligentní extrakce dat ve webovém prohlížeči**
Intelligent Data Scraping in a Web Browser
Kategorie: Web

Zadání:

1. Seznamte se se současnými serverovými i klientskými technologiemi pro implementaci webových aplikací v JavaScriptu.
2. Prostudujte současné přístupy pro extrakci dat (scraping) z webových dokumentů.
3. Navrhněte architekturu aplikace pro extrakci dat z webových stránek na základě předdefinovaných datových modelů. Svá rozhodnutí konzultujte s vedoucím.
4. Implementujte navrženou aplikaci pomocí vhodných technologií.
5. Proveďte vyhodnocení funkčnosti vytvořené aplikace na vhodné sadě webových dokumentů.
6. Zhodnoťte dosažené výsledky.

Literatura:

- Alarte, J.; Insa, D.; Silva, J.; et al.: Main Content Extraction from Heterogeneous Webpages. In Web Information Systems Engineering - WISE 2018. Cham: Springer International Publishing. 2018. ISBN 978-3-030-02922-7. pp. 393-407.
- Burget, R.: Model-Based Integration of Unstructured Web Data Sources Using Graph Representation of Document Contents. In: 15th International Conference on Web Information Systems and Technologies. Vienna: SciTePress - Science and Technology Publications, 2019, s. 326-333. ISBN 978-989-758-386-5.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Burget Radek, Ing., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 12. května 2021
Datum schválení: 22. října 2020

Abstrakt

Cílem této práce je získání dat z webových stránek bez znalosti jejich vnitřní struktury. Podstatou je rozpoznání této struktury pomocí algoritmu a zadaným vstupním informacím o obsahu, který chce uživatel extrahovat. Po analýze struktury následuje extrakce samotného obsahu. Na vybraných sadách internetových stránek se podařilo dosáhnout průměrné úspěšnosti přes 80%. Výsledný algoritmus představuje nový přístup k extrakci dat a může být nasazen v reálném světě, nebo může být součástí dalšího vývoje.

Abstract

The goal of this thesis is to extract data from web pages without the knowledge of their internal structure. The point is to recognize the structure using an algorithm and a given input information about the content that the user wants to extract. The structure analysis is then followed by the content extraction itself. An average success rate of over 80% was achieved on selected sets of websites. The resulting algorithm represents a new approach to data extraction and can be deployed in the real world or can be a part of further development.

Klíčová slova

Zpracování dokumentu, extrakce dat, rozpoznávání struktur dokumentu, web, TypeScript, Puppeteer

Keywords

Document processing, data extraction, document structure recognition, web, TypeScript, Puppeteer

Citace

MAŠTERA, František. *Inteligentní extrakce dat ve webovém prohlížeči.* Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Radek Burget, Ph.D.

Inteligentní extrakce dat ve webovém prohlížeči

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Radka Burgeta, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
František Maštera
10. května 2021

Poděkování

Děkuji mému vedoucímu panu Ing. Radkovi Burgetovi, Ph.D. za odbornou pomoc a vedení při tvorbě této bakalářské práce.

Obsah

Kapitola 1

Úvod

Extrakce dat z webových dokumentů je vzhledem ke způsobu, kterým jsou v nich data uložena, nesnadný úkol. Každý takový dokument má svou vlastní strukturu, která se může výrazně lišit od jiných. Data mohou být uložena v různých částech dokumentu a mohou být uložena odlišným způsobem. Se znalostí dokumentu a jeho struktury ve většině případů není problém najít opakující se vzor a využít ho k získání potřebných dat. Problém nastává absencí této informace, díky které je na algoritmu, aby odlišil chtěná data a jejich strukturu od ostatních.

V současném stavu je v praxi extrakce dat prováděna procedurálně, ad hoc pravidly pro každý webový dokument s odlišnou strukturou zvlášť. Význam této práce vyplývá ze způsobu zjištění polohy dat ve struktuře webového dokumentu. Práce využívá analýzu obsahu a struktury dokumentu v kombinaci se vstupními údaji o požadovaném obsahu k tomu, aby nalezla chtěná data, aniž by k tomu byly potřeba informace o tom, kde přesně se data v dokumentu nachází.

Bez nutnosti informace o poloze chtěných dat ve webovém dokumentu je možná hromadná extrakce stejného typu dat z vícero dokumentů s odlišnými strukturami za poskytnutí vstupu se stejným popisem těchto dat. Cílem práce je testování různých metod pro nalezení dat a následné dosažení takových výsledků, aby byl algoritmus pro tento scénář použitelný.

Účelem následující kapitoly je úvod do oblasti extrakce dat z webových stránek. Kapitola vysvětlí základy potřebné pro pochopení této práce a její motivace, zároveň představí existující práce, kterými je tato práce inspirována. Kapitola 3 obsahuje úvod do další teoretické části práce, která obsahuje technologie využité v rámci této práce.

V rámci kapitoly 4 rozdělíme řešený problém na podproblémy a stručně projdeme jejich řešení, které v kapitole 5 bude rozebráno do větších podrobností spolu s detailnějším vysvětlení zajímavějších částí implementace algoritmu. Výsledné testování a vyhodnocení úspěšnosti dosažení cíle práce je pak obsahem kapitoly 6.

Kapitola 2

Web scraping

Následující kapitola obsahuje základní seznámení s problematikou extrakce dat z internetových stránek. V rámci seznámení je motivace, proč web scraping existuje, stručný popis jak funguje a způsoby extrakce, na které tato práce navazuje. Neobsahuje kompletní shrnutí celé oblasti, ale pouze její části nezbytné v rámci této práce a k pochopení její motivace. Kapitola čerpá z [4].

2.1 Motivace k extrakci dat z internetových stránek

Ve webových dokumentech napříč internetem je uložené velké množství dat z nejrůznějších odvětví, jako jsou např.: [4], [18]

- Komerční – internetové obchody, realitní servery, letenky, sledování konkurence, burza, sportovní výsledky
- Veřejné rejstříky – jízdní řády, statistický úřad, webové stránky zastupitelstev
- Výsledky vyhledávání – hlídání pozice
- Kontrola reklamy, nabídky práce, detekce změn stránek, počasí a další

Pro možnost následného využití informací je potřeba tyto data dále zpracovat v počítačových aplikacích. K tomu potřebujeme strukturovaná data, ideálně reprezentovatelná tabulkami relační databáze, nebo alespoň serializované do formátů XML[1], JSON[2] apod. s pevně danou strukturou.

Struktura webových dokumentů je ovšem z velké části tvořena s ohledem na vzhled webové stránky. Jsou psané v jazyku HTML[3], kde elementy mají často význam čistě vizuálního charakteru. Tj. existují jen za účelem změny vizuální podoby dokumentu, případně k tomu, aby bylo možné snadno upravovat vzhled pomocí jazyka CSS[4]. Samotný obsah je podřízený tomuto účelu, navíc mezi sebou mohou mít stránky velmi odlišnou strukturu, která se časem může měnit.

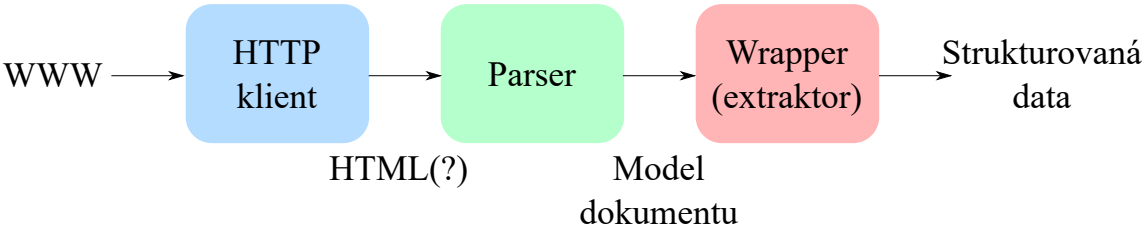
S řešením tohoto problému by měl přijít koncept tzv. sémantického webu. Jde o takový web, jehož obsah je strukturován podle standardizovaných pravidel, které mají za účel

[1] XML – Extensible Markup Language
[2] JSON – JavaScript Object Notation
[3] HTML – Hypertext Markup Language
[4] CSS – Cascading Style Sheets

usnadnit právě jeho zpracování. Příklady takových standardů mohou být Microdata[5], či RDFa[6]. [7] Takto strukturovaný obsah se ovšem vyskytuje jen v méně než polovině dokumentů na internetu. [2]

2.2 Co je to web scraping?

Web scraping, nebo také extrakce dat z webových dokumentů, je technika získávání nestrukturovaných dat z webových stránek a jejich transformace do podoby, se kterou mohou počítačové aplikace dále pracovat. [18]



Obrázek 2.1: Architektura extrakce dat z webového dokumentu. (převzato z [4])

Jak již vyplývá z obrázku 2.1, problematika extrakce dat se dělí na několik dílčích problémů. Prvním z nich je samotné získání zdrojových dat. Stažení webových dokumentů tak, aby obsahovaly to, co chceme, s sebou samo o sobě nese řadu problémů. Stránky se mohou naplňovat chtěnými informacemi až pomocí asynchronních volání. V některých specifičtějších situacích mohou být navíc chráněná přihlášením, nebo i různými ochrannými prvky proti automatizaci, jako např. CAPTCHA[7] formuláři.

Daleko větším a pro tuto práci důležitějším problémem je ovšem nalezení a následná extrakce dat, které chceme.

[5]Microdata – https://www.w3.org/TR/microdata/
[6]RDFa – https://rdfa.info/
[7]CAPTCHA – Completely Automated Public Turing test to tell Computers and Humans Apart

2.3 Extrakce informací, které chceme

Způsobů přístupu k nalezení dat, které chceme ze stránky získat, je mnoho. [18]

Jedním z jednodušších způsobů je využití regulárních výrazů. Tento přístup sestává z analýzy struktury stránky, na základě které se následně vytvoří regulární výrazy pro rozpoznání elementů obsahujících informace, které chceme ze stránky extrahovat. [9]

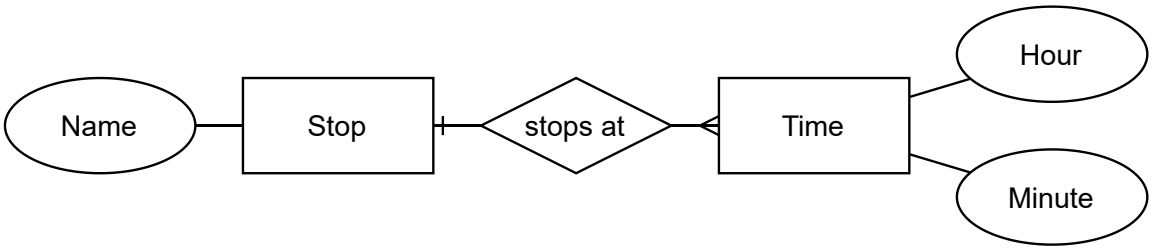
V praxi používanějším způsobem je práce s DOMem[8] – hierarchickou stromovou strukturou složená z jednotlivých elementů stránky. Hledání informací tímto způsobem zahrnuje navigaci ve stromu ať už pomocí standardních operací se stromovou strukturou, tak CSS selektorů nebo XPath[9].

Plnohodnotný HTML 5 DOM parser je obtížné najít a je prakticky pouze ve webových prohlížečích. V praxi se pak často používají zjednodušené parsery s vlastním rozhraním. Příkladem může být BeautifulSoup[10] pro programovací jazyk Python, nebo jsdom[11] pro jazyk JavaScript.

Zmíněné způsoby mají ovšem zásadní společný nedostatek – k nalezení dat je potřeba manuální vytvoření pravidel podle struktury stránky, na základě kterých program rozpozná informace, které extrahovat, od těch, které nás nezajímají. Aby toho nebylo málo, i drobné změny zdrojové stránky mají za následek nefunkčnost definovaných pravidel a nutnost jejich úpravy.

2.3.1 Modelem řízená extrakce

Jednou z cest, jak se pokusit o řešení tohoto problému, je modelem řízená extrakce. Tj. namísto předdefinovaných pravidel a cest ve stromové struktuře je programu poskytnut model struktury očekávaných dat, jako je např. na obrázku 2.2. Následná analýza stránky s cílem nalezení cesty k datům, které chceme extrahovat, pak není manuální. Místo toho ji provádí program na základě rozpoznaných výskytů požadovaných skupin dat ve zdrojové stránce.



Obrázek 2.2: Příklad modelu struktury dat ve formě datagramu vztahu entit popisující jízdní řády dvěma entitami (*Time*, *Stop*), třemi atributy (*Hour*, *Minute*, *Name*) a jednou relací (*stops at*). (převzato z [3])

Způsob hledání výskytů může být na základě analýzy stránky na vizuální úrovni, zcela oproštěné od stromové struktury stránky. Rozpoznávání informací probíhá na základě konzistence vzhledu elementů (font, barva, apod.) a jejich rozmístění na zdrojové stránce. [3]

[8] DOM – Document Object Model
[9] XPath – https://developer.mozilla.org/en-US/docs/Web/XPath
[10] BeautifulSoup – https://www.crummy.com/software/BeautifulSoup/
[11] jsdom – https://github.com/jsdom/jsdom

2.3.2 Detekce hlavního obsahu

Odlišný způsob na zcela opačné straně spektra je analýza DOM stromu. Program vybere množinu stránek, které patří ke klíčové stránce, podle které bude analýza probíhat. Pro každou stránku v množině program namapuje uzly DOMu dané stránky s uzly klíčové stránky. Při každé shodě pak inkrementuje čítač daného uzlu, díky kterému pozná počet výskytů uzlu v množině stránek.

Na základě počtu výskytů pak program vybírá uzly, které se v množině stránek neopakovaly. Množinu takových uzlů pak dále program redukuje např. eliminací potomků v množině. [1]

2.3.3 Směr této práce

Tato práce čerpá z obou výše zmíněných způsobů. Obsah, který má program ze stránky získat, je specifikován vstupním modelem struktury dat. Analýza ovšem neprobíhá na základě vizuální podoby. Program se snaží obsah rozpoznat na základě analýzy DOM stromu pomocí hledání opakovaných výskytů struktur na zdrojové stránce, které odpovídají vstupnímu modelu.

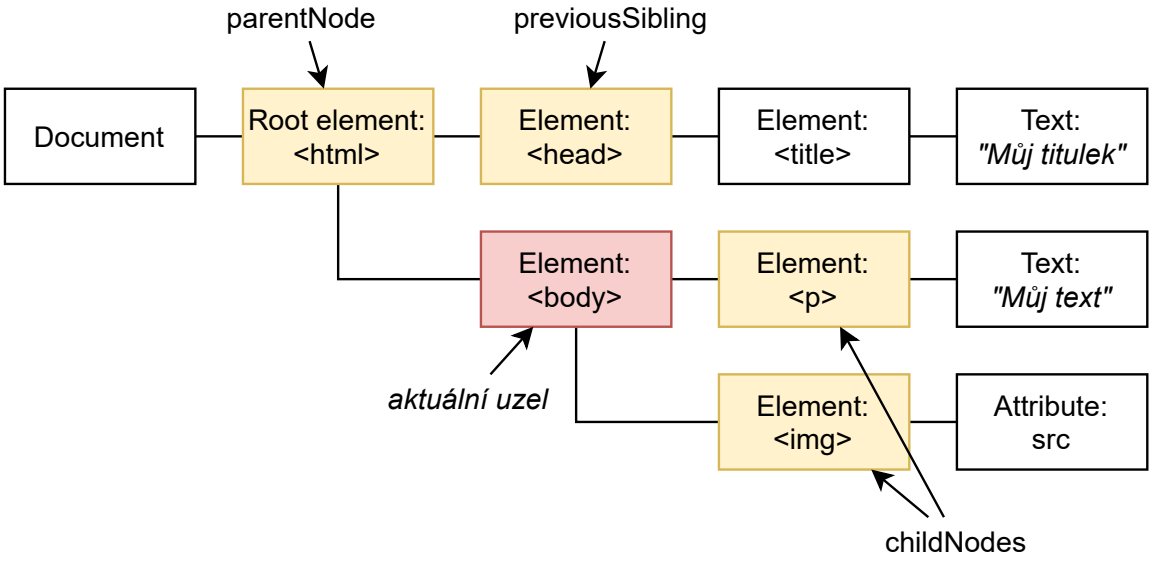
Kapitola 3

Současný stav technologií

Tato kapitola poskytuje stručné shrnutí technologií souvisejících s touto prací. Součástí shrnutí je seznámení se strukturou DOM, která je v rámci práce klíčová pro navigaci ve stránce a pro hledání chtěných datových struktur. Dále je součástí kapitoly úvod do programovacího jazyka JavaScript, včetně souvisejících technologií a knihoven, které tato práce využívá.

3.1 Document Object Model

Document Object Model (dále jen DOM) je programovací rozhraní pro dokumenty HTML a XML[1]. Reprezentuje stránku tak, aby mohly programy pracovat se strukturou, stylem a obsahem dokumentu. DOM je reprezentován stromovou strukturou objektů. [10]



Obrázek 3.1: Ukázka, jak může vypadat stromová struktura DOM dokumentu HTML. Jednotlivé šipky popisují příklady navigace pomocí standardních metod tříd DOM. (vychází z [5])

[1] XML – Extensible Markup Language

Kořenový objekt je typu *Document*. Ten má jednoho potomka typu *Element*, který může mít potomky typu *Element*, *text*, či jiné. *Element* je v rámci dokumentu HTML jeho úsek vymezený značkami (tagy). [5]

Navigace ve stromové struktuře DOM je možná buď skrze standardní metody tříd DOM *Document* a *Element* jak je tomu na obrázku 3.1, nebo také pomocí selektorů jazyka CSS, či použitím XPath.

3.2 Úvod do programovacího jazyka JavaScript

JavaScript je objektově orientovaný interpretovaný programovací jazyk, který se řídí standardem ECMAScript[2] spravovaným organizací ECMA International[3]. Od verze ES6 vychází každým rokem jeho nová verze s novými vlastnostmi, které jsou následně adoptovány virtuálními stroji, na kterých fungují všechny moderní prohlížeče i serverová prostředí, jako je Node.js. [12]

Primárním použitím jazyka JavaScript je vývoj responzivních internetových stránek (např. s komplexními animacemi, klikatelnými tlačítky, vyskakovacími okny apod.). Může být ale také použit k vývoji mobilních nebo serverových aplikací. V rámci prostředí může být připojen k objektům daného prostředí za účelem získání jejich kontroly. [11]

3.2.1 Klient a AJAX

Například v rámci internetového prohlížeče je jazyk rozšířen poskytnutím kontroly daného prohlížeče a DOMu. V takovém případě dovoluje úpravu obsahu stránky HTML, nebo reakce na akce uživatele (např. vyplnění formuláře). Tato forma se nazývá *klientský* JavaScript. [12]

JavaScript na straně klienta se nás v rámci práce týká kvůli tomu, jak stránka načítá data, která chceme získat. Hlavním problémem jsou rámce a asynchronní volání jazyka JavaScript, neboli AJAX[4]. Ty mohou do stránky načíst další obsah i po jejím zdánlivém načtení. Určit, kdy jsou všechna asynchronní volání dokončena a kdy je stránka plně načtená a připravená ke sběru dat není jednoduché a vzniká tak problém, kdy začít s extrakcí informací.

3.2.2 Server a Node.js

Na opačné straně je *serverový* JavaScript. Prostředí této formy rozšiřuje jazyk o objekty serverového prostředí – např. umožňují aplikaci komunikovat s databází, soubory na serveru, nebo také může poskytnout kontinuitu informací z vyvolání jedné aplikace do další. [12]

Node.js je běhové prostředí s otevřeným zdrojovým kódem (open-source), které umožňuje vývoj serverových nástrojů a aplikací v jazyce JavaScript, zcela odděleně od prostředí prohlížeče. Běží na stejném virtuálním stroji jako webový prohlížeč Chrome[5] – V8[6]. [14] Ten je napsaný v jazyce C++ a implementuje standard ECMAScript, kterým se jazyk JavaScript řídí. Virtuální stroj V8 je optimalizován pro webové prohlížeče, Node.js je tak velmi rychlý a má tak výhodu např. oproti jazykům Python nebo PHP. [8]

[2] ECMAScript – https://tc39.es/ecma262/
[3] ECMA International – https://www.ecma-international.org/
[4] AJAX – Asynchronous JavaScript And XML
[5] Google Chrome – https://www.google.com/chrome/
[6] V8 – https://v8.dev/

3.3 TypeScript

Jazyk JavaScript byl původně navržen pro kratší pomocné skripty v rámci webových stránek, obsluhující drobné funkce – nikoliv pro velké projekty. Pro ty vstupuje na scénu TypeScript – programovací jazyk vyvíjený a udržovaný společností Microsoft, který je nadstavbou jazyka JavaScript. Při samotném překladu je jazyk TypeScript nejprve přeložen do jazyka JavaScript a krom vlastních konstrukcí sdílí všechny možnosti jazyka JavaScript. Existující kód v jazyce JavaScript je tak i kódem v jazyce TypeScript.

Navíc ovšem přidává všem vlastní systém typové kontroly. Například, jazyk JavaScript nabízí primitiva jako `string`, `number` a `object` – ale už nekontroluje, jestli jsou konzistentně dodržovány. Jazyk TypeScript ano. Hlavní výhodou jazyka TypeScript je díky typové kontrole schopnost upozornit na potenciálně nechtěné části kódu, čímž výrazně zmenšuje prostor pro chyby. [20]

```
1  const user = {
2    name: "Daniel",
3    age: 26,
4  };
5
6  user.location; // Property 'location' does not exist on type
7                 // '{ name: string; age: number; }'.
```

Ukázka kódu 3.1: Validní úsek kódu v jazyce JavaScript. Položka `location` objektu `user` vrátí `undefined`. V jazyce TypeScript ovšem ohlásí statická kontrola chybu vypsanou v komentáři. (ukázka převzata z [22])

V jazyce TypeScript není možné psát mnoho konstrukcí, které jsou možné v jazyce JavaScript (např. viz ukázka 3.1). Ačkoliv se tato vlastnost může zdát limitující, zamezuje psaní řadě nečitelných konstrukcí, které často vedou k chybám. V krajních případech lze část omezení anulovat, např. přetypováním na `any` – datový typ, který může být cokoliv, jako v jazyce JavaScript (typová kontrola je pak ovšem značně omezena).

```
1  enum StatusCodes {
2    OK = 200,          // Mohou mít přidělenou hodnotu...
3    BadRequest = 400,
4    Unauthorized,      // ...ale také nemusí.
5    PaymentRequired,   // Implicitně pokračuje iterací
6    Forbidden,         // od poslední určené hodnoty.
7    NotFound,
8  };
9
10 console.log(StatusCodes.NotFound); // Vytiskne 404.
```

Ukázka kódu 3.2: Jazyk TypeScript přidává krom typové kontroly i další konstrukce, jako je například typ `enum`, s podobnou funkčností, kterou můžeme znát z jiných jazyků. (ukázka převzata z [21])

3.4 Významné knihovny

Jednou z výhod jazyka JavaScript je softwarový registr npm[7]. Vývojáři otevřeného kódu po celém světě využívají npm ke sdílení balíčků. Skládá se ze 3 částí: [13]

- webové stránky – pro objevování a správu balíčků, vytváření profilů, organizací
- terminálové rozhraní – manipulace s balíčky, jejich jednoduchá instalace
- registr – veřejná databáze JavaScript softwaru a metainformací

Díky již zmíněnému vztahu jazyků TypeScript a JavaScript je možné této výhody využít i v prostředí jazyka TypeScript. K vývoji této práce byla použita řada knihoven. Následující podčásti kapitoly projdou několik významnějších, jejichž zaměření se týká tématu této práce.

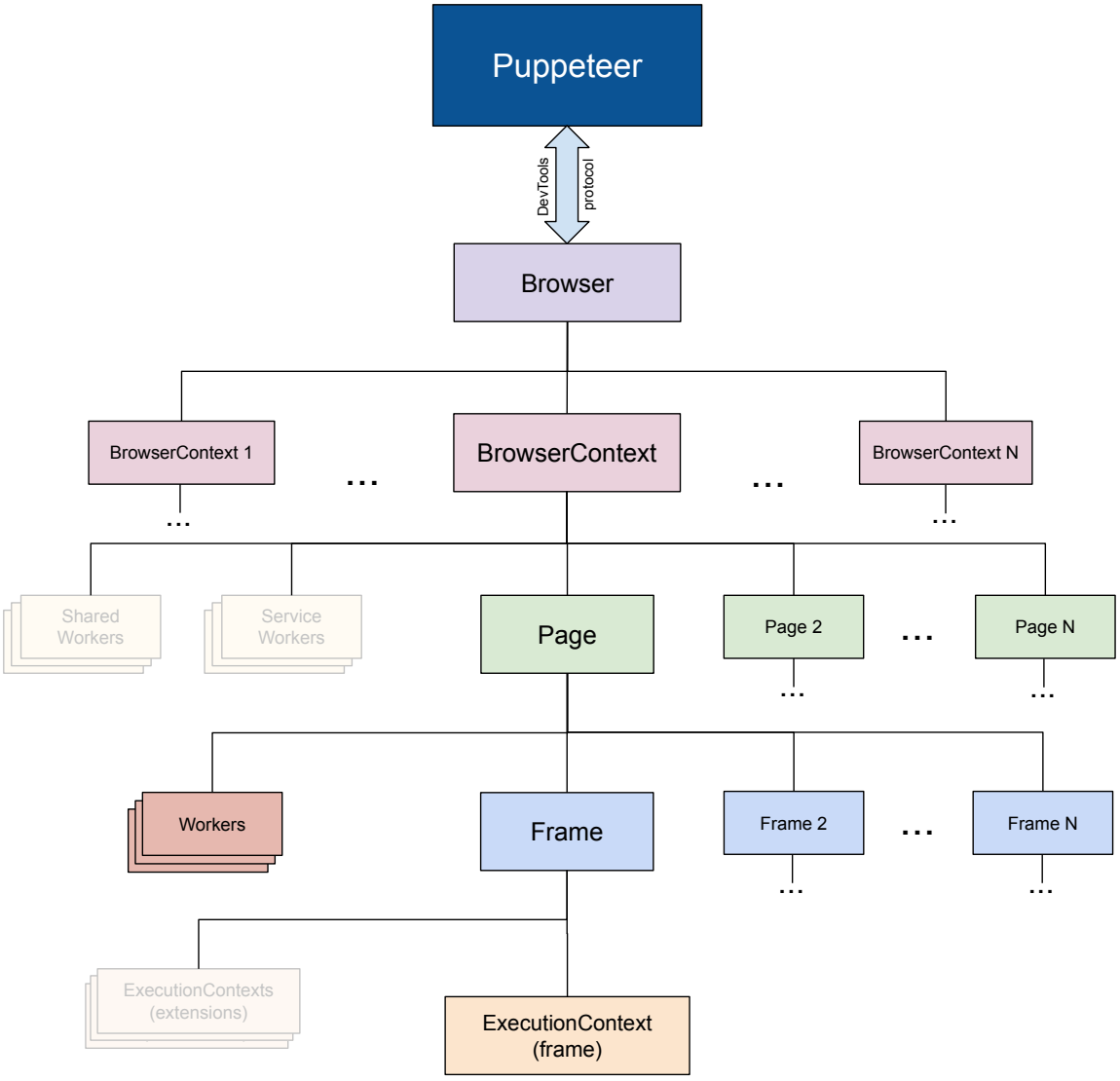
3.4.1 Puppeteer

Puppeteer je knihovna pro Node.js, která poskytuje vysokoúrovňové API[8] pro kontrolu prohlížeče Chrome nebo Chromium skrze DevTools protokol[9]. Krom standardní navigace a manipulaci se stránkou umožňuje také třeba injekci kódu jazyka JavaScript např. pro extrakci dat z elementu. Takový prohlížeč může být dobrý hned k několika účelům: [16]

- generování snímků obrazovky a souborů PDF ze stránek
- automatizování vyplňování formulářů, testování uživatelského rozhraní, vstupu klávesnice a další
- jako automatizované testovací prostředí
- diagnostika problémů s náročností webových stránek na výkon
- testování rozšíření pro prohlížeč Chrome

Výchozí nastavení prohlížeče je bez grafického uživatelského rozhraní, které nás během manipulace s prohlížečem z Node.js nezajímá. Dále na takový prohlížeč bude referováno jako na *headless*. Výhodou takového módu je menší zátěž spuštěného prohlížeče na systém. Skrze prohlížeč pak knihovna umožňuje manipulaci s internetovými stránkami – více k architektuře viz diagram 3.2.

[7] npm – Node Package Manager
[8] API – Application Programming Interface
[9] DevTools Protocol – https://chromedevtools.github.io/devtools-protocol/



Obrázek 3.2: Architektura knihovny Puppeteer. Knihovna operuje s prohlížečem (skrze DevTools protokol), který může mít více kontextů, čímž umožňuje práci nad více prohlížeči najednou. Ty mohou mít otevřených více stránek, které jsou rozdělené na rámce (s vlastními kontexty pro exekuci kódu jazyka JavaScript) a *Workers*, které pracují s Web Workers API.[11]Světlé entity nejsou aktuálně knihovnou zastoupeny. (diagram převzat z [17])

Krom toho, že je knihovna s otevřeným kódem volně přístupná na serveru GitHub[12], pracují na ní také vývojáři společnosti Google, kteří se podílejí na vývoji samotného prohlížeče Chrome.

Při výběru technologií pro práci se nabízela např. i alternativa knihovny Selenium WebDriver[13] v kombinaci s jazykem Python. Nakonec bylo ovšem rozhodnuto pro knihovnu Puppeteer, zejména díky lepší možnosti určení momentu načtení stránek. Další možnosti knihovny Puppeteer jsou pro extrakci dat dostačující a výhody knihovny Selenium, jako

[11]Web Workers API – https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API
[12]Puppeteer na serveru Github – https://github.com/puppeteer/puppeteer
[13]Selenium – https://www.selenium.dev/

např. podpora většího množství prohlížečů díky WebDriver API[14], jsou klíčové spíše pro testovací účely.

3.4.2 JSDOM a jQuery

JSDOM[15] je implementace vícero standardů (zejména DOM a HTML) v jazyce JavaScript pro použití se serverovým prostředím Node.js. Cílem projektu je emulace dostatečné části webového prohlížeče k tomu, aby se dal použít pro testování a extrakci dat v aplikacích v reálném světě. Pro tuto práci slouží jako nástroj pro získání modelu DOM ze zdrojové stránky načtené nástrojem Puppeteer.

jQuery[16] je knihovna pro procházení HTML, jeho manipulaci a práci s eventy, animacemi a AJAXem. Má jednoduché API funkční napříč webovými prohlížeči, jehož využití lze vidět na příkladu 3.3. V rámci této práce je její hlavní účel v usnadnění procházení a práce s modelem DOM zdrojové stránky získaného nástrojem JSDOM.

```
1 function containChild(parents: Element[], checked: Element[]) {
2   return $(parents).find($(checked)).get().length !== 0;
3 }
```

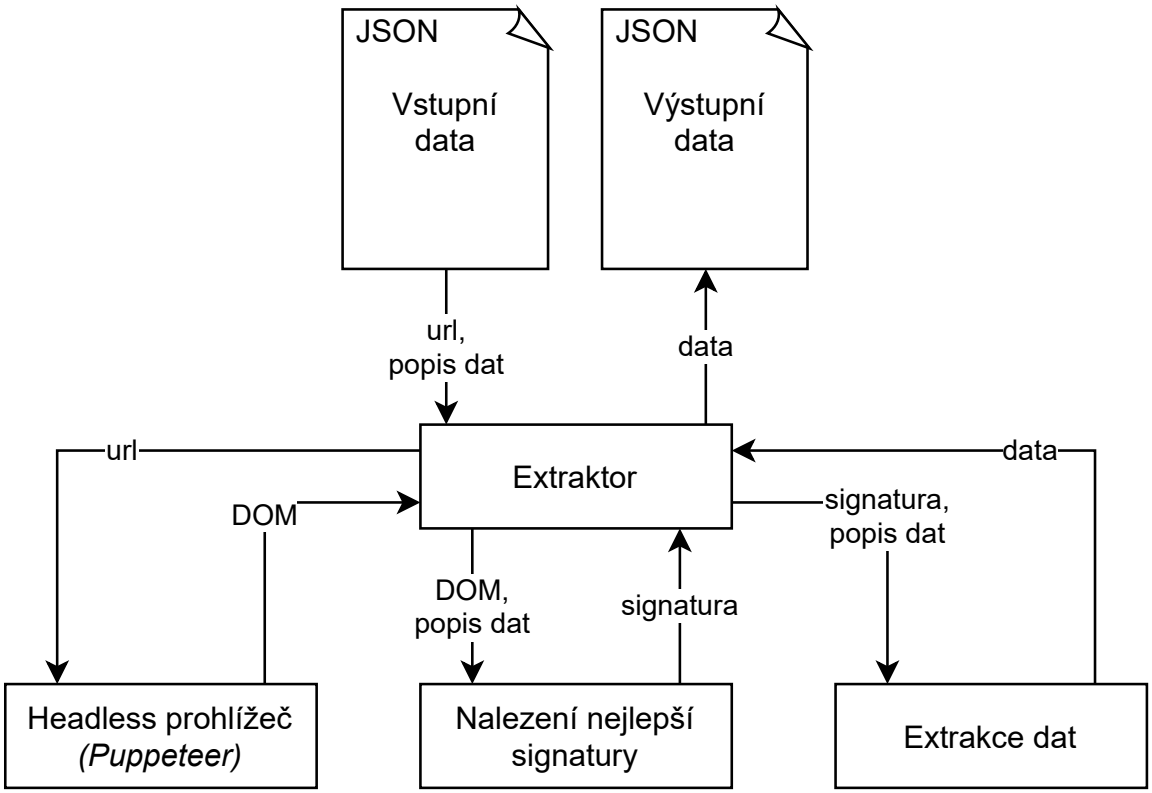
Ukázka kódu 3.3: API knihovny jQuery umožňuje volání hledání elementů nad kolekcí elementů. Pokud kterýkoliv z kolekce elementů `parents` má jako potomka kteréhokoliv z kolekce elementů `checked`, funkce vrátí `true`. `$` je instance objektu jQuery.

[14] WebDriver API – https://selenium-python.readthedocs.io/api.html
[15] JSDOM – https://github.com/jsdom/jsdom
[16] jQuery – https://github.com/jquery/jquery

Kapitola 4

Návrh architektury

Cílem práce je vytvoření aplikace, která pomocí vstupního popisu struktury dat tyto data ve zdrojové stránce nalezne a extrahuje je. Před implementací této aplikace je potřeba navrhnout její architekturu – rozdělit problém na jednotlivé podproblémy, kterým budou přiřazeny jednotlivé části aplikace s popisem, jak problém budou řešit. Schéma této architektury je na diagramu 4.1.



Obrázek 4.1: *Extraktor* pojme vstupní data a spustí pomocí knihovny Puppeteer webový prohlížeč. Z něj získá obsah zdrojové stránky, který poskytne modulu pro nalezení nejvhodnější signatury pro extrakci dat. Nakonec tuto signaturu předá modulu pro samotnou extrakci dat, který vrátí extrahovaná data. Ty jsou následně zapsána do výstupního souboru.

4.1 Vstupní data

Vstupní data jsou prezentovány aplikaci souborem ve formátu JSON. Formát byl zvolen kvůli vestavěné podpoře v jazyce JavaScript a tím pádem minimálnímu potřebnému úsilí ke zprovoznění čtení vstupního souboru.

```
{
  "webpages": [
    {
      "url": "...",
      "output": "..."
    }
  ],
  "attributes": {
    "ATTRIBUTE_NAME": {
      "type": "int", // nebo "string", "float"
      "regexes": [
        "..."
      ],
      "value": {
        "min": 0,
        "mid": 2,
        "max": 4
      },
      "length": {
        "min": 2,
        "mid": 5,
        "max": 10
      },
      "list": {
        "min": 1,
        "max": 3
      },
      "optional": false,
      "labelDetection": true,
      "dontExtract": false
    }
  }
}
```

Ukázka kódu 4.1: Soubor se vstupními daty se všemi možnými parametry.

Jednotlivé parametry vstupního souboru z ukázky 4.1 a jejich funkce:

- `webpages` – Pole objektů obsahující informace k jednotlivým stránkám, ze kterých bude algoritmus extrahovat data.
  - `url` – URL adresa zdrojové stránky, ze které budou data extrahována.

    - `output` – Cesta výstupního souboru, kde budou extrahovaná data uložena. Volitelný parametr, výchozí hodnota: `"output.json"`.

- `attributes` – Objekt, který obsahuje objekty popisující jednotlivé atributy. Tyto objekty nesou názvy atributů, které popisují.
    - `type` – Datový typ atributu. Může být 1 z: `"string"` pro řetězce, `"int"` pro celá čísla, `"float"` pro desetinná čísla
    - `regexes` – Pole regulárních výrazů. Pokud není prázdné nebo zcela nechybí, obsah se musí shodovat s alespoň jedním z nich.
    - `value` – Objekt popisující rozsah hodnoty v rámci atributu. Dává smysl u číselných datových typů. Volitelný parametr.
    - `length` – Objekt popisující rozsah délky řetězce v rámci atributu. Dává smysl u řetězců, funguje ovšem i pro číselné atributy. Volitelný parametr.
    - `list` – Objekt popisující rozsah počtu položek v kolekci. Může být také pravdivostní hodnota: `true` pokud je atribut kolekce bez definovaného rozsahu, `false` pokud nejde o kolekci. Volitelný parametr, výchozí hodnota: `false`
    - `optional` – Příznak určující, jestli jde o atribut, který nemusí být nutně součástí objektu. Volitelný parametr, výchozí hodnota: `false`
    - `labelDetection` – Příznak určující, zda-li se má algoritmus v rámci atributu vyhýbat opakujícím se řetězcům. Slouží pro možnost vypnutí v případě, kdy je možnost, že hodnota atributu bude stejná pro všechny instance objektu. Volitelný parametr, výchozí hodnota: `true`
    - `dontExtract` – Příznak určující, jestli má algoritmus při ukládání objektu atribut vynechat. Atribut v takovém případě slouží pro hledání instancí, ale ve výsledném výstupním souboru se vyskytovat nebude. Volitelný parametr, výchozí hodnota: `false`

Objekt popisující rozsah použitý pro parametry `value`, `length` a `list` obsahuje parametry:

- `min` – Minimální povolená hodnota. Volitelný parametr, pokud chybí, algoritmus bere spodní hranici jako $-\infty$.
- `mid` – Odhadovaná nejčastější hodnota. Volitelný parametr, algoritmus nahrazuje pomocí krajních hodnot, případně pokud nejsou definovány, tak nebere parametr v potaz. Účelem je další indicie pro algoritmus při hledání atributu. Parametr `list` nepodporuje.
- `max` – Maximální povolená hodnota. Volitelný parametr, pokud chybí, algoritmus bere horní hranici jako ∞.

Vstupní soubor musí obsahovat definici alespoň jednoho atributu, který musí existovat v rámci každé instance. Tj. hodnota parametru `optional` není `true` a spodní limit `min` objektu `list` není roven 0 nebo nižší.

Většina volitelných nastavovacích parametrů (jako zapnutí režimu pro hledání chyb nebo jiné parametry, které slouží pro testování algoritmu) bude předávána programu skrze argumenty příkazové řádky.

Finální vstupní soubor může vypadat například tak, jak je tomu na ukázce 4.2.

```
{
  "webpages": [
    {
      "url": "https://www.w3schools.com/tags/ref_colornames.asp",
      "output": "out/colornames.json"
    }
  ],
  "attributes": {
    "name": {
      "type": "string",
      "regexes": ["^([A-Z][a-z]+)+$"]
    },
    "hex": {
      "type": "string",
      "regexes": ["^#[\\dA-F]{6}$"],
      "length": {
        "min": 7,
        "max": 7
      }
    }
  }
}
```

Ukázka kódu 4.2: Ukázka funkčního vstupního souboru. Cílem extrakce je 1 stránka s adresou https://www.w3schools.com/tags/ref_colornames.asp, ze které chceme získat objekty jednotlivých barev o dvou atributech – `name` s názvem barvy a `hex` s kódem barvy v šestnáctkové soustavě. Získané objekty jsou uloženy do souboru `colornames.json` ve složce `out`.

4.2 Funkce jednotlivých modulů

Po získání adresy stránky ze vstupního souboru je spuštěn modul obalující headless prohlížeč vlastním jednoduchým rozhraním. Hlavní funkcí je extrakce modelu DOM ze stránky na poskytnuté adrese. K načtení zdrojové stránky bude použita knihovna Puppeteer, přičemž k získání obsahu načtené stránky v podobě modelu DOM bude využita knihovna JSDOM.

K hledání instancí objektů definovaných ve vstupním souboru je potřeba nejprve najít shody v získaném modelu DOM s jednotlivými atributy objektu. Z nalezených shod dále algoritmus skládá signatury.

Signatura je popis struktury podstromu v dané hloubce v rámci modelu DOM zdrojové stránky reprezentující možnou instanci hledaného objektu. Signatura obsahuje pro jednotlivé atributy hledaného objektu popis polohy elementů, jejichž obsah může odpovídat popisu těchto atributů, relativní ke kořenu podstromu popisovaného signaturou a hloubku, ve které se tento kořen nachází. Vzhledem k tomu, že součástí popisu není přímo kořen podstromu, ale pouze jeho hloubka, nemusí být popis unikátní – ve stejné hloubce se mohou opakovat struktury se stejným popisem pod rozdílnými kořeny.

Algoritmus bude postupně sbírat data o signaturách, ze kterých utvoří statistiku, na základě které se ve finále rozhodne pro jednu ze signatur, kterou předá zpět extraktoru. Mezi statistiky bude patřit zejména počet výskytu jednotlivých signatur, ale také komplexnější metriky určující jak moc se jednotlivé atributy signatury podobají poskytnutému popisu ze vstupního souboru.

Posledním je modul pro extrakci dat. Ten převezme vybranou signaturu a postupně projde celý model DOM zdrojové stránky. Pokaždé když narazí na strukturu odpovídající vybrané signatuře s atributy, které odpovídají popisu ze vstupního souboru, tak z modelu extrahuje data. Výsledné pole objektů z extraktoru dat je zapsáno do výstupního souboru, který je též ve formátu JSON.

```
[
  {
    "name": "AliceBlue",
    "hex": "#F0F8FF"
  },
  {
    "name": "AntiqueWhite",
    "hex": "#FAEBD7"
  },
  {
    "name": "Aqua",
    "hex": "#00FFFF"
  },

  // ...

  {
    "name": "YellowGreen",
    "hex": "#9ACD32"
  }
]
```

Ukázka kódu 4.3: Zkrácená ukázka výstupního souboru po poskytnutí vstupního souboru z ukázky 4.2.

Kapitola 5

Implementační řešení

Následující sekce budou procházet jednotlivé problémy práce a do detailu vysvětlovat, jak bylo přistoupeno k jejich řešení při implementaci aplikace.

5.1 Získání zdrojové stránky

Prvním problémem je samotné načtení zdrojové stránky – konkrétně určení momentu, kdy je stránka plně načtená a připravená k extrakci. K určení tohoto momentu se nabízí několik řešení.

První možností je využití eventu *DOMContentLoaded*[1]. Ten je zachycen v případě, kdy je HTML dokument plně načten. Přestože zní jako ideální řešení, event nečeká na načtení obrázků, stylů a nebere v potaz AJAX. Obdobné řešení nabízí event *load*[2], který je vyvolán v momentě, kdy je celá stránka načtená, včetně stylů a obrázků. Pořád ovšem nepočítá s čekáním na dokončení všech asynchronních volání a některé stránky event nemusí vyvolat vůbec – např. na obrázku 5.1 tento event moc nepomohl.

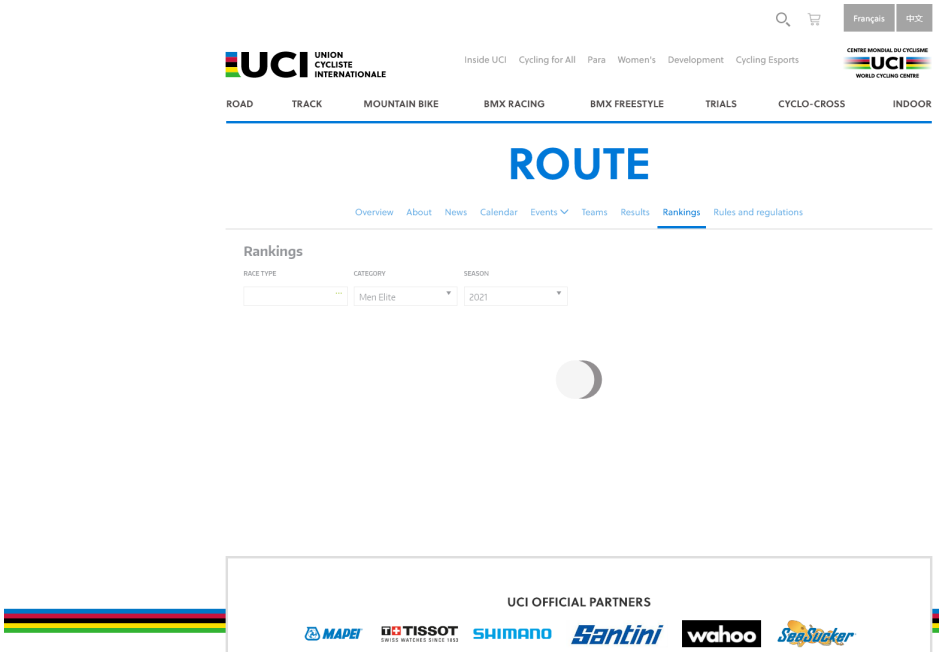
Místo využití eventů lze nastavit pevný časový interval. Stránku v takovém případě prohlásíme za načtenou po uplynutí nastavené doby. Není ovšem zaručeno, že stránka bude v takovém momentě opravdu načtená. Sice můžeme interval prodlužovat, ale to je velmi neefektivní a i tak nebude načtení nikdy zaručeno.

V praxi využívaným řešením je čekání na načtení vybraného elementu, specifikovaného selektorem. V rámci této práce toto řešení ovšem použít nemůžeme, protože dopředu neznáme strukturu zdrojové stránky a tím pádem ani nedokážeme určit element, který bychom chtěli mít načtený, natož na něj psát selektor.

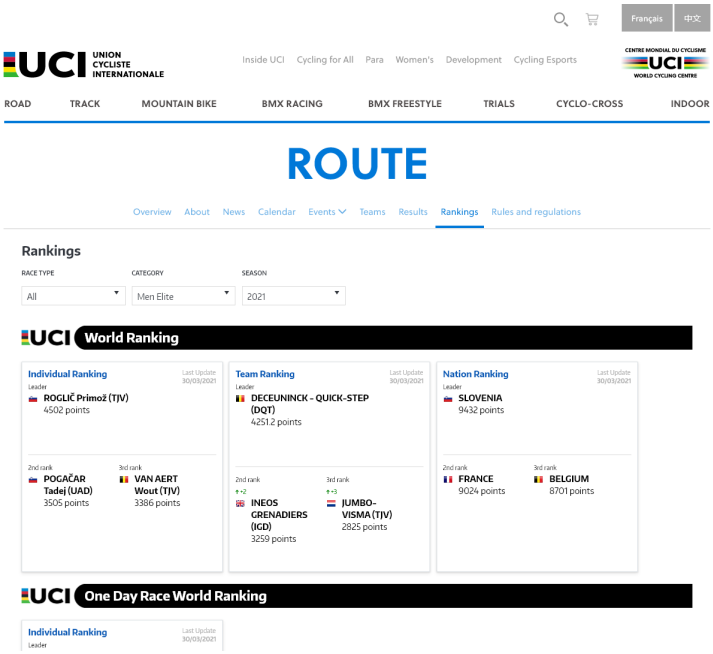
S poslední možností přichází knihovna Puppeteer – *networkidle*. Tato podmínka spočívá v tom, že prohlížeč počká, až nebude detekována žádná síťová aktivita po nastavenou dobu. Nastavení této podmínky je značně limitováno – díky tomu, že podmínka je vestavěná přímo v prohlížeči Chromium. Doba je neměnná a nastavená na 500 milisekund. Podmínka pak má dvě varianty, z nichž první, *networkidle0*, vyžaduje nulovou aktivitu po celou dobu a druhá, *networkidle2*, povoluje nanejvýše 2 aktivní síťová spojení – pro případy, kdy má stránka na pozadí dlouhodobě aktivní spojení. Na obrázku 5.2 je vidět, že tato možnost přináší lepší výsledky.

[1] DOMContentLoaded – https://developer.mozilla.org/en-US/docs/Web/API/Window/DOMContentLoaded_event

[2] load – https://developer.mozilla.org/en-US/docs/Web/API/Window/load_event



Obrázek 5.1: Snímek obrazovky v momentě, kdy byl vyvolán event *load* při načítání stránky z adresy `https://www.uci.org/road/rankings` obsahující asynchronně načítaný rámec.



Obrázek 5.2: Stejná situace jako na snímku obrazovky 5.1, tentokrát ovšem v momentě, který určila podmínka *networkidle0* a kdy je již rámec plně načtený.

5.2 Nalezení všech shod s atributy

Před hledáním samotných signatur je potřeba nejprve získanou zdrojovou stránku analyzovat. Obsah každého elementu na stránce je kontrolován vůči shodě s každým atributem hledané struktury. Obsah elementu navíc může být získat dvěma různými způsoby.

5.2.1 Extrakce obsahu elementu

Prvním je získání textu pouze z kontrolovaného elementu, tedy bez obsahu žádného z potomků. Její implementace vychází z [6]. Druhá metoda extrahuje textový obsah z celého podstromu.

```
<div id="kontrolovany-element">
  Text
  <div>
    Text <span>potomků</span>
  </div>
  rodiče
</div>
```

Ukázka kódu 5.1: V případě kontroly elementu s identifikátorem `kontrolovany-element` dostaneme buď *Text rodiče*, pokud budeme brát pouze text daného elementu, nebo *Text Text potomků rodiče*, jestliže zahrneme text celého podstromu.

Metoda pro získání obsahu pouze z kontrolovaného elementu je použita pro všechny elementy. K tomu, aby byl brán v potaz i obsah získaný metodou pro získání obsahu z celého podstromu musí ovšem platit následující podmínky:

1. Výška podstromu musí být rovna 1 (hodnota může být upravena parametrem příkazové řádky, experimoentování s ním je v rámci kapitoly 6). Hodnota byla vybrána na základě výsledků testování, ve kterém vycházela jako nejlepší. Vyšší hodnoty způsobovaly na mnoha stránkách zahrnutí nepotřebných informací. Nastavení nižší hodnoty by pak v podstatě způsobilo, že tato metoda nebude nikdy použita.
2. Obsah není roven obsahu získanému z metody pro získání obsahu pouze z kontrolovaného elementu.
3. Žádný z potomků nemá stejný obsah získaný touto metodou.

To znamená, že obsah získaný z podstromu kontrolovaného elementu z ukázky 5.1 by nebyl brán v potaz, protože by s výškou podstromu rovnou 2 neprošel první podmínkou.

5.2.2 Určení shody elementu s atributem

Jakmile je obsah z elementu extrahován, program jej porovná se všemi atributy a určí ty z nich, ke kterým může být přiřazen a s jakou pravděpodobností.

Základní kontrola spočívá v regulárních výrazech. Shoda s alespoň jedním z regulárních výrazů přiřazeným k danému atributu ze vstupních dat je podmínkou, aby element vůbec mohl kontrolou projít. Výjimkou je pak případ, kdy atribut žádné regulární výrazy zadané nemá (tuto část v takovém případě program přeskočí).

Krom této jednoduché podmínky jsou ovšem kontrolovány i další parametry atributu, jejichž výsledkem může být číselná hodnota představující, jak moc je podle programu element podobný popisu daného atributu. Tato hodnota je v programu definována jako *hitRate* a představuje zásadní faktor při výběru signatury v pozdější fázi extrakce.

Při výpočtu této hodnoty jsou využity zadané informace o rozsahu atributu – jeho minimální a maximální hodnoty (a_{min} a a_{max}) a střední hodnota (a_{mid}). Z obsahu elementu jsou přečteny 2 různé hodnoty – délka obsahu (v_0) a v případě číselného obsahu také jeho číselná hodnota (v_1).

Pro každou hodnotu v_i je nejprve otestováno, zda patří do rozsahu $\langle a_{min},\ a_{max}\rangle$. Nedefinované hranice program v rámci kontroly nahrazuje $-\infty$ a ∞. Pokud do rozsahu některá z hodnot v_i nepatří, element není shodný s atributem a kontrola končí.

Algoritmus 1: Výpočet hodnoty *hitRate* pro hodnotu elementu v_i pro atribut a

Vstup: $v_i,\ a_{min},\ a_{max},\ a_{mid}$
Výstup: *hitRate*

```
Kontrola hodnot atributu.
if a_mid is undefined then
    if a_min is not undefined and a_max is not undefined then
        a_mid = (a_min + a_max)/2
    else return 0
end if
if a_min is undefined then a_min = −∞
if a_max is undefined then a_max = ∞
Výpočet hodnoty hitRate.
r = max(a_max − a_mid, a_mid − a_min)
if r is 0 then
    return 0
else return (v_i − a_mid)/r
```

V opačném případě přichází na řadu výpočet hodnoty *hitRate*. Ta se vypočítává pro každou hodnotu v_i zvlášť – viz algoritmus 1. Pokud hodnota a_{mid} není definována, program ji nahradí průměrem hraničních hodnot rozsahu. Pokud nejsou definovány ani ty, program přiřadí hodnotě v_i perfektní *hitRate*.

Jinak je následně na základě rozdílu hodnot v_i a a_{mid} přiřazen dané hodnotě v_i hodnota *hitRate* v rozsahu $\langle -1,\ 1\rangle$, kde 0 je perfektní shoda (tj. rozdíl je roven 0) a krajní hodnoty jsou nejmenší a největší možné rozdíly v rámci povoleného rozsahu.

Rozsah má stejnou vzdálenost jak od spodní, tak horní hranice. Toto opatření je proti zkreslení v případech, kdy je hodnota a_{mid} výrazně blíže jedné hranici než druhé.

Výsledný *hitRate* shody daného obsahu elementu s daným atributem je určen průměrem všech hodnot *hitRate* (tedy maximálně dvěma – pro každou z hodnot v_i).

Shody jsou uloženy v mapě M, kde klíčem je název atributu a hodnotou pole objektů reprezentující jednotlivé shody. V instancích objektů je uloženo:

- hodnota *hitRate* reprezentující hodnocení shody
- příznak určující jakým způsobem byl obsah z elementů extrahován
- element, se kterým byla shoda atributu nalezena

5.2.3 Hledání kolekcí

M zatím obsahuje pouze shody atributů s jednotlivými elementy $1:1$. Pro nalezení shod atributů kolekcí s více elementy ($1:N$) musíme najít mezi jednotlivými shodami takových atributů ($M_{a_{list}}$) souvislosti, na základě kterých by se daly shody sloučit.

Program předpokládá existenci jednotlivých elementů kolekce ve stejné výšce v modelu DOM zdrojové stránky. Proto každou množinu shod $M_{a_{list}}$ rozdělí na podmnožiny D podle jednotlivých hloubek, ve kterých byly shody nalezeny.

Pro každou hloubku D_i je vytvořena množina polí C. Ta je složena ze sloučení množin kombinací všech shod v hloubce D_i, které jsou rozděleny podle způsobu získání obsahu elementu. Rozdělení je za účelem zachování informace o způsobu získání obsahu z elementu. Zároveň také zaručujeme, že bude mít shoda s atributem kolekce právě jeden způsob získání obsahu z jednotlivých elementů.

Ve vzniklých kombinacích program určí nejnižší rodičovský uzel p – kořen podstromu, kde se kolekce nachází. Pro každý uzel p si zapamatuje tu kombinaci, která má pro tento uzel nejvyšší počet shod, čímž zajistí nalezení všech možných položek kolekce pod uzlem p.

Po určení kombinací pro všechny takto určené uzly p proběhne pro každý uzel p poslední kontrola, jestli je počet elementů kombinace pod tímto uzlem uvnitř rozsahu $\langle a_{min},\ a_{max}\rangle$ definovaného ve vstupním souboru. Obdobně jako u předešlých rozsahů jsou hranice nahrazeny $-\infty$ a ∞ v případě, že nejsou definovány.

Všechny uzly p, které kontrolou projdou, jsou uloženy do původní mapy shod M, kde nahrazují původně uložené shody pod klíči atributů kolekcí. Pro každou shodu je uloženo:

- průměrný *hitRate* původních shod, které jsou nyní součástí kolekce
- příznak určující jakým způsobem byl obsah z elementů extrahován
- rodičovský uzel p
- hloubka od uzlu p, ve které se nachází jednotlivé položky kolekce

Kvůli náročnosti výpočtu pro větší množství shod díky využití kombinací se hledání kolekcí provádí až v pozdější fázi. Pro následující kroky nám budou stačit nalezené shody obyčejných atributů.

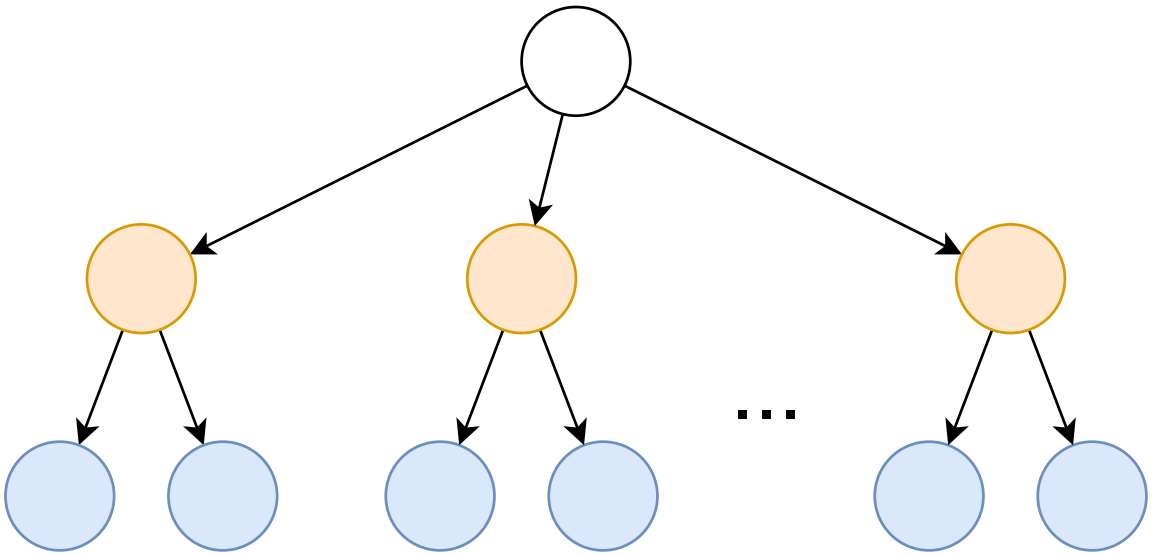
5.3 Hledání možných instancí

Po provedené analýze obsahu zdrojové stránky je úkolem mezi nalezenými potenciálními výskyty jednotlivých atributů najít ty, které nejpravděpodobněji reprezentují skutečná data, která chceme. Zároveň musíme také odlišit jednotlivé instance objektu, které budeme v závěru ukládat do výstupního souboru.

Jeden z možných způsobů hledání je zkoušení všech možných kombinací nalezených shod. Ve výsledku by se jednalo o kartézský součin $M_{a_0} \times M_{a_1} \times \ldots M_{a_i}$, kde M_{a_i} je množina shod pro atribut a_i. S tímto přístupem bychom ovšem velmi rychle narazili na problém náročnosti výpočtu. Už při 4 atributech s 50 výskyty na každý z nich by program kontroloval 6 250 000 různých kombinací. Spousta z nich by přitom ani nedávala smysl (třeba kombinace shod na opačném konci stránky). Z toho je jasné, že množinu testovaných kombinací je potřeba nějakým způsobem zmenšit.

5.3.1 Hledání podstromů s možnými instancemi objektu

Za předpokladu, že má hledaná struktura alespoň 1 atribut, který musí existovat právě jednou pro každou instanci extrahovaného objektu (zajištěno podmínkou vstupního souboru), lze provést jednoduchou optimalizaci. Z mapy shod je vybrán takový atribut, který splňuje tuto podmínku a má nejmenší počet shod s unikátními elementy (tj. 2 shody s odlišným způsobem získání obsahu – viz sekce 5.2.1 – se počítají za 1). Tento počet představuje maximální možný počet instancí hledané struktury. Atribut bude dále nazýván jako a_{rarest}.



Obrázek 5.3: Předpokládaná (zjednodušená) struktura zdrojové stránky. Modré uzly představují elementy s nalezenou shodou s některými z atributů objektu. Oranžové uzly reprezentují nejnižší možné kořeny podstromu, který může obsahovat validní instanci hledaného objektu.

Pokud je brána v úvahu struktura zdrojové stránky z obrázku 5.3, lze hledat instance objektu okolo každého elementu e shody atributu a_{rarest}.

Algoritmus 2: Hledání podstromů, které mohou obsahovat instanci hledaného objektu

Vstup: $e,\ P,\ d_{min},\ A,\ c_{max},\ l$
Výstup: M

```
1  p, d, M = undefined
2  w = false
3  while true do
       Získání kořenového uzlu podstromu p a jeho hloubky d.
4      if p is undefined then
5          p = e
6          d = getDepth(e)
7      else
8          p = getParent(p)
9          d = d − 1
10     end if
       Prevence proti opakování.
11     if p ∈ P or d < d_min then break else P = P ∪ p
       Kontrola maximálního počtu kombinací.
12     N = findMatches(getSubtreeElements(p), A)
13     if ∏_{i=0}^{length(N)} length(N_i) > c_max then break else M = N
       Řádné ukončení při dostatečném obsahu podstromu.
14     if w is false then
15         w = canContainCompleteSignature(M)
16     else if l > 0 then
17         l = l − 1
18     else break
19 end while
20 if w is true then return M else return undefined
```

Pro každý element e hledáme podstrom, který může obsahovat instanci hledaného objektu – viz algoritmus 2. Jestliže je výsledkem algoritmu mapa M, která může obsahovat kompletní instanci hledaného objektu, program v další fázi pokračuje vybíráním signatur v rámci této mapy (vysvětlené v rámci sekce 5.4).

Hledání podstromu probíhá takovým způsobem, že od elementu e postupně stoupáme o úroveň výše v modelu DOM zdrojové stránky s každou iterací pomocí funkce *getParent*. Během iterování sledujeme aktuální hloubku kořenového uzlu podstromu (v algoritmu 2 získanou pomocí funkce *getDepth*) a množinu P s elementy, které již byly v jedné z předchozích iterací označeny za kořen hledaného podstromu.

Množina P slouží k prevenci proti opakování hledání v již kontrolovaných podstromech. Hloubka je pak kontrolována vůči hodnotě minimální možné hloubky d_{min}, která má výchozí hodnotu 2 (nad touto hloubkou jsou pouze elementy `<html>`, `<head>` a `<body>`). Jakmile je

kořen podstromu v aktuální iteraci obsažen v množině P, nebo je jeho hloubka menší než d_{min}, tak hledání podstromu pro element e končí.

Z podstromu s kořenovým elementem p program získá všechny elementy, které ho tvoří – včetně p. Tuto část obsluhuje funkce *getSubtreeElements*. Pro všechny tyto elementy získáme funkcí *findMatches* mapu shod N se všemi atributy A, stejným způsobem jako v sekci 5.2. Díky menší velikosti podstromu si můžeme dovolit také hledání shod s atributy kolekcí, které budeme potřebovat, abychom mohli provést kontroly v dalších krocích.

Abychom předešli případům, kdy podstrom nabyde příliš velkému množství shod, které by způsobovalo v pozdějších krocích problémy s výkonem při hledání signatur v rámci podstromu, získáme produkt počtů shod všech atributů v rámci mapy shod N. Pokud produkt přesáhne hodnotu c_{max}, hledání podstromu pro element e končí. V opačném případě je mapa N přiřazena proměnné M a hledání pokračuje.

Na konci každé iterace algoritmus kontroluje, za může podstrom obsahovat kompletní instanci hledaného objektu. K tomu slouží funkce *canContainCompleteSignature*, vysvětlená v navazující sekci. Jakmile taková situace nastane, algoritmus se pokusí o dalších l iterací. Účelem je pokrytí případů, kdy algoritmem nalezený podstrom sice může obsahovat signaturu odpovídající vstupním datům, ale námi chtěný objekt se vyskytuje třeba jen o úroveň výše. Díky kontrole počtu kombinací oproti hodnotě c_{max} si toto může algoritmus dovolit.

5.3.2 Ověření validity podstromu

Podstrom může obsahovat kompletní instanci hledaného objektu, jestliže může obsahovat pro každý atribut alespoň jednu shodu tak, aby nebyla konfliktní vzhledem k žádné takové shodě s jiným atributem.

Shody m_0 a m_1 jsou vůči sobě konfliktní, jestliže platí alespoň jedna z následujících podmínek:

- Elementy, ze kterých shody vznikly, jsou identické. V případě shod atributu kolekce je brán jako element rodičovský uzel shody.
- Jestliže jedna ze shod je k atributu kolekce (m_c) a druhá není (m_s) a shoda m_c může mezi položkami kolekce obsahovat elementy shody m_s.
- Pokud je obsah některé ze shod získán z celého podstromu jejího elementu a obsah druhé shody je součástí tohoto podstromu.

Jako první krok může program vyřadit z kontroly všechny atributy, které nemusí mít v rámci validní instance žádnou shodu a tím pádem se jimi program nemusí zabývat. Jde o atributy kolekcí s minimální hranicí 0, nebo atributy s příznakem `optional` nastaveným na `false`. Výsledkem je pole atributů A, které již zkontrolovat musíme.

Dále může program ukončit kontrolu s neúspěšným výsledkem v případě, že alespoň jeden z atributů A nemá přiřazenou žádnou shodu.

V rámci dalšího postupu bude úkolem programu se pokusit o rozdělení shod mezi jednotlivé atributy A takovým způsobem, aby splňovali podmínku, kterou jsme si definovali na začátku. Program nejprve vytvoří mapu M_C, která bude obsahovat pod každým atributem A seznam jejich shod, které jsou konfliktní s alespoň jedním z dalších atributů A.

Následně se program pokusí pomocí mapy M_C vybrat všem atributům A_U (které nemají žádné nekonfliktní shody) z jejich konfliktních shod takové, aby shody ve vybrané kombinaci nebyly konfliktní navzájem. Pokud takovou kombinaci nalezne, podstrom může obsahovat alespoň jednu instanci hledaného objektu, jinak nikoliv.

Hledání této kombinace probíhá v iteracích. Pro každou iteraci program určí atribut a_U, který má nejmenší počet konfliktních shod C. Pokud je tento počet roven 0, kombinace neexistuje. V opačném případě vybere ze shod M_C shodu m_C takovou, která je konfliktní s nejmenším možným počtem shod atributů A. Všechny shody konfliktní konfliktní se shodou m_C pak odstraní. Zároveň odstraní shodu a_U. Tímto si program označil, že k tomuto atributu našel shodu, kterou již nebude uvažovat jiným atributům v následujících iteracích.

Pokud program narazí na to, že je množina atributů A_U prázdná, kombinace je nalezena a podstrom může obsahovat alespoň jednu instanci hledaného objektu.

5.4 Sledování a vybírání signatur

Jestliže máme mapu shod M, která může obsahovat instanci hledaného objektu, je dalším krokem programu nalezení všech možných signatur složených ze shod obsažených v této mapě.

Algoritmus vytvoří všechny možné kombinace atributů, které nemusí v rámci instance objektu nutně existovat a mají v mapě M alespoň jednu shodu. Každá z kombinací je následně sloučena s atributy, které v rámci instance objektu mít shodu musí. Tímto způsobem zaručí algoritmus nalezení i méně kompletních signatur (tj. s méně atributy), které by jinak byly vynechány pokaždé, když by existovala v rámci podstromu, ze kterého mapa M vznikla, více kompletní signatura.

Pro každou tímto způsobem vzniklou kombinaci atributů hledá algoritmus v mapě M všechny možné kombinace nekonfliktních shod C. Pro každou nalezenou kombinaci $c \in C$ je vytvořena a uložena signatura s.

```
{
  "parentDepth": 4,
  "attrSignatures": {
    "title": {
      "isTextTopLevel": true,
      "elementPath": [0, 0]
    },
    "price": {
      "isTextTopLevel": false,
      "elementPath": [1]
    },
    "tags": {
      "isTextTopLevel": true,
      "parentPath": [2],
      "contentDepth": 1
    }
  }
}
```

Ukázka kódu 5.2: Příklad, jak může vypadat signatura nalezené kombinace shod c serializovaná do formátu JSON. Signatura je ve hloubce 4. Pod objektem `attrSignatures` se nachází informace ke shodám jednotlivých atributů `title`, `price` a `tags`. Parametr `isTextTopLevel` je příznak určující způsob získání obsahu z elementu. Parametry

`elementPath` obsahují cestu indexů od rodičovského uzlu k elementu, ve kterém byla nalezena shoda. Podobným případem je `parentPath`, který určuje cestu k rodičovskému uzlu jednotlivých položek kolekce. Parametr `contentDepth` pak určuje, v jaké hloubce od rodičovského uzlu se jednotlivé položky kolekce nachází.

Tvorba signatury začíná určením nejnižšího společného rodiče všech elementů shod patřících do c. Do signatury je uložena hloubka tohoto elementu v rámci modelu DOM. Následně jsou pro každý atribut, který má v rámci kombinace c shodu, do signatury uloženy informace o této shodě a jejích elementech potřebné pro možnost extrakce obsahu těchto elementů. Mezi informace patří zejména cesta k elementům od společného rodiče. Výsledná signatura je naimplementována jako serializovatelný objekt, viz ukázka 5.2.

5.4.1 Indexová cesta

```
1  <div id="start">          // []
2    <h1>Nadpis</h1>         // [0]
3    <div>                   // [1]
4      <ul>                  // [1, 0]
5        <li>První</li>      // [1, 0, 0]
6        <li>Druhý</li>      // [1, 0, 1]
7        <li>Třetí</li>      // [1, 0, 2]
8      </ul>
9      <p>Text</p>           // [1, 1]
10   </div>
11 </div>
```

Ukázka kódu 5.3: Podstrom modelu DOM pro demonstraci indexové cesty. V komentářích jsou indexové cesty k elementům na daném řádku, vezmeme-li v úvahu počáteční element s identifikátorem `start`.

Indexovou cestu používá program v rámci signatur pro určení polohy elementu jednotlivých shod. Jedná se o pole čísel, kde každé číslo reprezentuje index v poli potomků. Indexová cesta může začínat od libovolného elementu, včetně kořenového elementu `<html>`.

Např. v ukázce 5.3 začněme od elementu s identifikátorem `start`. Pokud vezmeme v úvahu indexovou cestu `[1, 0]`, postupujeme následovně:

1. Počátečním elementem je element s identifikátorem `start`.
2. První index je `1`. Jdeme tedy na 2. z potomků aktuálního elementu, tj. `<div>` na 3. řádku v ukázce.
3. Druhý index je `0`. Jdeme tedy na 1. z potomků aktuálního elementu, tj. `<ul>` na 4. řádku v ukázce.
4. Další indexy již nezbývají – vybraný element je `<ul>` na 4. řádku v ukázce.

I prázdná indexová cesta je validní, postup by v takovém případě skončil zvolením elementu, od kterého výběr začíná.

Tento způsob orientace po stromu byl vybrán díky své jednoduchosti jak na pochopení, tak implementaci. V rámci této práce se ukázal být dostačující.

5.4.2 Ukládání statistik o signaturách

Program si v průběhu hledání signatur udržuje statistiku, na základě které probíhá výběr signatury v pozdější fázi. Statistika je uložena ve formě mapy D, kde klíčem je signatura a hodnotou její příslušná programem sledovaná data.

Hlavním sledovaným parametrem je počet výskytů jednotlivých signatur. Signatury nemusí být unikátní a v rámci modelu DOM zdrojové stránky se může vyskytovat nekonečně mnoho identických signatur. Opakující se signatura je pro program indicií, že jde o obsah stránky, který chceme extrahovat. Při nalezení signatury, která je již v mapě D obsažena, je počet výskytů této signatury inkrementován o 1.

Ke každé signatuře jsou pak ukládány statistiky k jednotlivým atributům. Ke každému z nich je uchováváno pole hodnot *hitRate* z každé shody daného atributu.

Důležitou součásti statistik k jednotlivý atributům je mapa L, která zajišťuje detekci opakujících se řetězců. Pokaždé, když je do mapy D vložena nová signatura, je do statistik každého atributu vložené signatury vložena mapa L, jejíchž klíči jsou indexové cesty k jednotlivým elementům podstromu, ze kterého byl obsah elementu získán – jestliže byl obsah získán pouze z jednoho elementu, pak mapa obsahuje pouze jeden klíč. Hodnotami jsou pak extrahované obsahy těchto elementů. Elementy s prázdným obsahem do mapy vloženy nejsou.

Při každém dalším nálezu signatury, která již v mapě D existuje, je pro nalezenou signaturu vytvořena stejným způsobem mapa L_n a srovnána s původní mapou L uloženou ve statistikách. Všechny podobné hodnoty, které jsou pod stejným klíčem jsou pak z původní mapy L odstraněny. Hodnoty jsou v tomto kontextu podobné, jestliže jsou stejné, nebo jejich tvary v jednotném nebo množném čísle jsou stejné. K převodu množných a jednotných čísel je použita knihovna pluralize[3].

Jakmile je mapa L prázdná, program může s jistotou říci, že pro daný atribut v dané signatuře neexistuje obsah, který by se neustále opakoval. Motivací tohoto systému je vyhnutí se nadpisům, jednotkám, či jiným elementům, které by se jinak mohly stát cílem extrakce a které typicky uživatel získat nechce. Může se ovšem stát, že uživatel nemůže zaručit, že některý z atributů bude mít vždy alespoň jeden odlišný prvek. Pro takové případy lze toto opatření pro vybrané atributy vypnout v rámci vstupního souboru.

5.4.3 Výběr nejlepší signatury

Na základě nasbíraných statistik program vybírá signaturu, která je v poslední části programu použita pro extrakci dat ze zdrojové stránky. Jakmile jsou nalezeny všechny možné signatury a k nim získány všechny statistiky, je výsledná mapa statistik D seřazena s následujícími prioritami:

1. Signatury, které neobsahují žádný atribut s opakujícím se obsahem (atributy s detekcí vypnutou skrze vstupní soubor se nepočítají).
2. Podle počtu výskytů signatur.
3. Podle průměrné hodnoty *hitRate*, která je vypočítána z průměrů absolutních hodnot *hitRate* všech atributů signatury.

Z mapy je následně vybrána první signatura v pořadí, s_a. Následně je vytvořena nová mapa statistik signatur D_s, která bude podmnožinou mapy D. Je naplněna všemi statis-

[3] pluralize – https://github.com/plurals/pluralize

tikami z mapy D, jejichž signatury jsou *supersignaturami* signatury s_a. Signatura s_1 je *supersignaturou* s_2, jestliže obsahuje všechny identické shody všech atributů jako shoda s_2 a navíc má alespoň jednu shodu s nějakým dalším atributem.

Cílem této mapy je použití více kompletní verze vybrané signatury v případě, že byla nějaká nalezena. Pokud mapa D_s není prázdná, je z ní vybrána finální signatura s_b. Jinak je finální signaturou s_b signatura s_a.

5.5 Extrakce dat pomocí vybrané signatury

Poslední fází programu je použití vybrané signatury s_b k extrakci dat ze zdrojové stránky.

Program vybere všechny elementy E zdrojové stránky v hloubce modelu DOM odpovídající hodnotě `parentDepth` v signatuře. Pro každý takový element $e \in E$ pak hledá pomocí indexových cest elementy pro jednotlivé atributy k extrakci.

Metoda extrakce z elementu je řízena podle příznaku `isTextTopLevel` v popisu atributu signatury. V případě, že element odpovídající indexové cestě nebyl nalezen, nebo jeho obsah neodpovídá popisu ze vstupního souboru, záleží další postup na příznaku `optional`. Pokud je nastavený na hodnotu `true`, tak algoritmus pokračuje v extrakci dalšího atributu. V opačném případě ovšem prohlásí podstrom elementu e za neplatný a extrakci pro tento element končí.

Extrakce atributů kolekcí probíhá tak, že program najde rodičovský element p kolekce nalezený indexovou cestou. Pokud na dané indexové cestě není nalezen, tak extrakce pro element e končí neúspěchem. Jinak se pokusí extrahovat obsah ze všech elementů v hloubce relativní od elementu p dané hodnotou atributu signatury `contentDepth`, které splňují popis atributu ze vstupního souboru. Tímto způsobem získá pole obsahů C. Pokud některý z obsahů elementů položek kolekce nebude odpovídat popisu ze vstupního souboru, pak není do pole C vložen. Po projití všech elementů položek kolekce program zkontroluje, zdali finální počet položek v poli C odpovídá stanoveným hraničním hodnotám ve vstupním souboru. Pokud nikoliv, chování algoritmu je obdobné, jako při selhání extrakce normálního atributu.

Pokud obsah všech atributů získán z podstromu elementu e splňuje všechny podmínky, je uložen do pole F, obsahující kolekci kompletních instancí hledaných objektů. Po dokončení iterací pro všechny $e \in E$ je pole F serializováno do formátu JSON a uloženo do výstupního souboru.

Kapitola 6

Vyhodnocení

Součástí této kapitoly bude testování algoritmu na reálných datech a podrobná analýza výsledků. Kapitola probere, jak dobře algoritmus funguje, jaké jsou jeho silné a slabé stránky, proč tomu tak je a způsob, kterým se k těmto závěrům došlo. Obsahem závěru kapitoly je zamyšlení nad možným navázáním na výsledky této práce a možnostmi, jak by se mohly dát výsledky teoreticky ještě zlepšit.

6.1 Datové sady

Jednotlivé datové sady použité k vyhodnocení byly získány z různých domén takovým způsobem, aby byl algoritmus otestován pro různé zdrojové stránky, ale také pro odlišné vstupní datové struktury. Vytvoření sad proběhlo v rámci spolupráce s autorem práce na podobné téma. [15] Celkem jsou sady čtyři:

1. `eshop` – Cílem sady jsou položky stránek internetových obchodů různorodých odvětví (elektronika, hračky, zahrádkářské potřeby, apod.). Pro každou takovou položku má za úkol program najít její název, popisek, cenu a pokud je položka ve slevě, tak i její slevu. Očekávaným problémem této sady je rozeznání těchto atributů od ostatních elementů, kterých v některých obchodech může být poměrně velké množství (stav na skladě, další parametry, upoutávky, apod.).

2. `tsbohemia` [15] – Tato sada se od ostatních liší tím, že všechny zdrojové stránky jsou z jednoho internetového obchodu, který je navíc součástí sady `eshop`. Atributy položek jsou identické jako u sady `eshop`, ačkoliv jejich popis je záměrně upraven, aby lépe odpovídal právě tomuto obchodu. Cílem sady je zjistit, jak se budou lišit výsledky od extrakce stejného obchodu v rámci sady `eshop`, tj. jak moc pomohou algoritmu konkrétnější vstupní data zaměřená na jediný obchod namísto na 10 odlišných.

3. `news` – Obsahem sady jsou stránky obsahující seznam článků s novinkami, opět o různých tématech (cyklistika, hry, elektronika, apod.). Úkolem algoritmu pro tuto sadu je extrakce jednotlivých článků. Ke každému článku musí extrahovat titulek, shrnutí, název autora a datum jeho vydání (případně doba relativní k jeho vydání). Výzvou sady pro algoritmus bude rozeznání jednotlivých atributů mezi sebou. Ty lze poměrně těžko odlišit, jelikož jde o řetězce a k žádnému z nich nelze napsat regulární výraz k odlišení od ostatních.

4. `football` [15] – Poslední sada obsahuje stránky s fotbalovými zápasy. Cílem extrakce je extrakce výsledků jednotlivých fotbalových zápasů. Pro každý zápas bude algoritmus hledat čas, 2 týmy, které proti sobě v zápase hrají a případně výsledné skóre zápasu, pokud již zápas proběhl. Na rozdíl od ostatních sad jsou data ve stránkách této sady často uložena v tabulkách. Navíc je položek na stránku řádově více, což může být výzvou pro algoritmus z hlediska výkonu.

Všechny obsahují 1 soubor se vstupními daty a 10 zdrojových stránek spolu s 10 soubory s očekávanými výstupními daty ke každé z nich. Zdrojové stránky sdílí podobná data, které lze extrahovat do jednotné struktury. Avšak jde o zcela odlišné stránky, díky čemuž se struktura dat mezi jednotlivými stránkami výrazně liší – tak, jak by tomu bylo v praxi.

Všechny testované webové stránky jsou reálnými stránkami, jejichž offline verze pro zaručení opakovatelnosti testů byly získány pomocí rozšíření pro webové prohlížeče SingleFile[1] nebo Save Page WE[2]. Tyto offline verze jsou obsaženy v přiloženém médiu, včetně testovacího programu pro usnadnění získání výsledků prezentovaných v této kapitole.

6.2 Průběh testování

K testování byl vytvořen krátký testovací skript, který může být spuštěn buď pro celou datovou sadu W, nebo také pro jednu stránku $w \in W$ zvlášť. Skript spustí program pro extrakci dat a po jeho dokončení následně určí shodu mezi výstupem programu O a daným očekávaným výstupem E pro každou stránku w. Atributy a_1 a a_2 jsou shodné, jestliže je jejich název i extrahovaný obsah identický, krom odlišnosti bílých znaků.

Hledání shod mezi množinami výstupů O a E probíhá iteracemi mezi objekty jednotlivých výstupů. Skript v rámci každé iterace najde dvojici objektů $(o \in O,\ e \in E)$, která má největší počet shodných atributů. Následně přičte počet shodných atributů mezi objekty dvojice k hodnotě v_C a objekty z daných množin odstraní. Iterace končí, jakmile je množina O nebo E prázdná.

Výsledkem je finální hodnota v_C, která představuje počet atributů, které program extrahoval správně. Dalšími výstupními hodnotami jsou v_O a v_E, reprezentující celkové počty atributů z výstupů O a E. V neposlední řadě skript měří dobu extrakce T od spuštění programu až po jeho dokončení. Pro zajištění konzistence této hodnoty jsou vždy provedena 3 měření. Jako výsledná hodnota T_F je pak použitý průměr v případě, že rozdíl mezi nejmenší a největší z hodnot T není větší než 5% z jejich průměru. V opačném případě jsou všechny běhy spuštěny znovu. Ve výsledcích bude také uvedena doba T_F jak pro jednotlivé stránky, tak pro celé sady. Suma všech T_F každé stránky v sadě neodpovídá době běhu dané sady.

Při spuštění celé sady je algoritmus spuštěn jednou pro všechny stránky dané sady a algoritmus tak neopakuje inicializaci prostředí pro každou stránku (podrobná analýza viz sekce 6.4). Hodnoty jsou pro všechny stránky $w \in W$ uloženy do tabulky, která je finálním výstupem testovacího skriptu.

Tyto hodnoty nám stačí pro výpočet hodnot *precision* (dále P) a *recall* (dále R), metrik používaným k výpočtu úspěšnosti vyhledávacích algoritmů. Hodnota P je vypočítána jako $\frac{v_C}{v_O}$ a udává hodnotu v rozmezí $\langle 0,\ 1 \rangle$, kde 0 je nejhorší a 1 nejlepší poměr správných výsledků oproti všem získaným – tj. udává, jak moc jsou získané výsledky správné. Výpočet hodnoty

[1] SingleFile – https://github.com/gildas-lormeau/SingleFile

[2] Save Page WE – https://chrome.google.com/webstore/detail/save-page-we/dhhpefjklgkmgeafimnjhojgjamoafof

R je $\frac{v_C}{v_E}$. Může být opět v rozsahu $\langle 0,\ 1\rangle$, kde 0 je nejhorší a 1 nejlepší poměr správných výsledků oproti všem očekávaným – tj. udává, jak moc algoritmus pokryl všechny očekávané výsledky. [19] Kombinací hodnot P a R je pak *F-score* (dále F), představující harmonický průměr těchto hodnot:

$$F = 2 \cdot \frac{P \cdot R}{P + R}$$

V rámci algoritmu lze měnit několik parametrů, které mohou ovlivnit v různých ohledech jeho výsledky:

- `maxAttributeTreeHeight` – Maximální povolená výška podstromu elementu pro získání obsahu z tohoto podstromu místo elementu samotného (viz sekce 5.2.1). S narůstající hodnotou je předpokládaným dopadem rozšíření nalezených shod o komplexnější struktury elementů.
- `extraLevels` – Maximální počet iterací navíc při hledání podstromu po chvíli, kdy podstrom může obsahovat instanci objektu (viz sekce 5.3.1). Očekávaným dopadem narůstající hodnoty jsou větší podstromy s možnými instancemi, jejichž následkem může být nalezení instance, ovšem za cenu možného nálezu velkého množství dat, které nás nezajímají a které mohou mít negativní dopad na výkon.
- `skipInstanceCombCount` – Maximální počet kombinací shod v rámci podstromu s možnými instancemi hledaného objektu (viz sekce 5.3.1). Účelem hodnoty je pojistka před nalezením podstromů s příliš velkým dopadem na výkon. Nižší hodnoty mohou mít však za následek, že se hledání podstromů zastaví před nalezením chtěné instance.

6.3 Rozbor výsledků experimentů

Testování bude rozděleno na několik experimentů, které se od sebe budou lišit zejména v nastavení parametrů. Jednotlivé experimenty na sebe budou navazovat a pokusí se o řešení problémů předchozích experimentů.

Rozbor výsledků jednotlivých experimentů se zaměřuje zejména na zajímavější části. Úvodní experiment bude obsahovat tabulky s kompletními výsledky. Navazující experimenty ovšem zobrazí tabulky s přesnými výsledky pouze v takových případech, kdy to bude vhodné.

Tabulky výsledků mohou obsahovat zabarvené buňky. Toto značení představuje změnu oproti jednomu z předcházejících experimentů (specifikováno v popise dané tabulky). Zelené hodnoty představují zlepšení, červené naopak zhoršení. Žlutě jsou pak buňky, u kterých změna jejich obsažené hodnoty nemá nutně kladný nebo pozitivní efekt. Zabarvení může být sytější u hodnot, kterých je změna v rámci kontextu významnější než u ostatních.

6.3.1 Úvodní experiment

Pro úvodní experiment jsou nastavené parametry `maxAttributeTreeHeight` na 1 a `extraLevels` na 0, aby algoritmus prozatím bral podstromy hned, jakmile mohou obsahovat instanci objektu. Parametr `skipInstanceCombCount` je nastaven na 100 000 – dostatečně

velká hodnota, aby nijak nebránila v hledání a zároveň nezpůsobovala větší problémy s výkonem.

Datová sada	Stránka	v_C	v_O	v_E	P	R	F	T_F
eshop	alza.cz	22	46	86	47,83 %	25,58 %	33,33 %	5,961 s
	czc.cz	46	69	85	66,67 %	54,12 %	59,74 %	6,014 s
	insportline.cz	162	220	224	73,64 %	72,32 %	72,97 %	6,245 s
	papirnictvieshop.cz	60	60	60	100,00 %	100,00 %	100,00 %	5,344 s
	smarty.cz	40	40	88	100,00 %	45,45 %	62,50 %	6,257 s
	sparkys.cz	10	46	48	21,74 %	20,83 %	21,28 %	6,018 s
	stavebninystastny.cz	30	40	40	75,00 %	75,00 %	75,00 %	5,955 s
	tsbohemia.cz	44	78	78	56,41 %	56,41 %	56,41 %	5,459 s
	zahradkarske-potreby.cz	30	45	45	66,67 %	66,67 %	66,67 %	5,177 s
	zahradnictvi-spomysl.cz	114	114	114	100,00 %	100,00 %	100,00 %	5,284 s
	průměr	-	-	-	**70,79 %**	**61,64 %**	**64,79 %**	**5,771 s**
	celá sada	-	-	-	-	-	-	**22,423 s**

Obrázek 6.1: Výsledky úvodního experimentu pro sadu `eshop`.

Hned u první stránky `alza.cz` pozorujeme z tabulky 6.1 poměrně nízké hodnoty. Algoritmu se nepodařilo nalézt atributy s názvem a popiskem položky, protože se při hledání podstromu zastavil o úroveň níže, než vůbec dostal šanci podstrom s názvem i popiskem najít. Místo toho se spokojil s jinými elementy, jejichž obsah dostatečně splňoval popis atributů.

Velkým problémem v rámci tohoto setu je pro algoritmus nalezení správné ceny. Atribut ceny je ve vstupním souboru popsán jako řetězec s regulárním výrazem. Různé obchody mohou mít však cen u produktu více (bez DPH, nebo údajná předchozí cena položky). Popisu atributu odpovídají pak všechny a jediným zbývajícím vodítkem je počet nálezů. To v některých případech (např. `insportline.cz`) může stačit – typicky je hlavní cena u všech produktů, zatímco např. údajná předchozí cena bude jen u produktů ve slevě –, ve zbylých je to pro algoritmus neřešitelný problém (např. `stavebninystastny.cz`).

Nutno zmínit, že extrahovaným cenám položek ze stránky `insportline.cz` chybí pouze měna, částky jsou jinak správně (algoritmus dal přednost extrakci z jednoho elementu místo zahrnutí i elementu obsahující řetězec s měnou).

Některé obchody mohou mít u položek různé elementy, které u ostatních schází. Často jde o různá upozornění zákazníka na stav na skladě, hodnocení produktu apod. To způsobuje rozdílnou strukturu mezi jednotlivými položkami, která v některých případech (např. `smarty.cz`) může způsobit, že fáze extrakce dat pomocí výsledné signatury nezíská vždy všechny povinné atributy, protože se mohou nacházet na jiné indexové cestě, než která je uložena v rámci signatury.

V případě stránky `sparkys.cz` algoritmus naráží na problém, kdy je průměrná délka popisu položky kratší, než jejího názvu. Jelikož jsou atributy název a popis položky definovány obě řetězcem o určité délce, je délka pro algoritmus klíčovým parametrem pro odlišení těchto dvou atributů. Kvůli tomu v tomto případě rozlišení těchto atributů selhává a ve výsledném výstupu je jejich obsahy oproti očekávanému výstupu navzájem vyměněné.

Na podobném principu selhává i hledání atributu názvu položky na stránce `tsbohemia.cz`. Tam definované délce ve vstupním souboru více odpovídá element s obsahem informujícím o stavu položky na skladě. Reálný název položky pak není extrahován vůbec.

Datová sada	Stránka	v_C	v_O	v_E	P	R	F	T_F
tsbohemia	brusky	70	81	81	86,42 %	86,42 %	86,42 %	5,867 s
	chladice	70	73	73	95,89 %	95,89 %	95,89 %	5,920 s
	chladnicky	39	75	81	52,00 %	48,15 %	50,00 %	6,051 s
	disky	65	74	77	87,84 %	84,42 %	86,09 %	5,833 s
	kolobezky	61	73	73	83,56 %	83,56 %	83,56 %	5,818 s
	mobo	57	73	74	78,08 %	77,03 %	77,55 %	5,839 s
	monitory	64	73	74	87,67 %	86,49 %	87,07 %	5,884 s
	proc	67	75	78	89,33 %	85,90 %	87,58 %	5,868 s
	servery	71	72	72	98,61 %	98,61 %	98,61 %	5,874 s
	skenery	69	72	72	95,83 %	95,83 %	95,83 %	5,869 s
	průměr	-	-	-	**85,52 %**	**84,23 %**	**84,86 %**	**5,882 s**
	celá sada	-	-	-	-	-	-	**21,930 s**

Obrázek 6.2: Výsledky úvodního experimentu pro sadu `tsbohemia`.

Z výsledků sady `tsbohemia` z tabulky 6.2 je vidět významné zlepšení. Oproti sadě `eshop` již bere algoritmus stav na skladě místo názvu položky pouze v 1 z 10 stránek. Nicméně problém s extrakcí špatné ceny v případě položek ve slevě přetrvává – zpřesnění popisu vstupních dat tomuto problému nijak nepomohl.

Datová sada	Stránka	v_C	v_O	v_E	P	R	F	T_F
news	artnews.com	0	40	40	0,00 %	0,00 %	0,00 %	18,797 s
	ceskymac.cz	80	80	80	100,00 %	100,00 %	100,00 %	5,900 s
	cyclingnews.com	80	80	80	100,00 %	100,00 %	100,00 %	6,257 s
	google.com	40	40	40	100,00 %	100,00 %	100,00 %	5,091 s
	mndaily.com	40	40	40	100,00 %	100,00 %	100,00 %	15,656 s
	pcgamer.com	60	80	80	75,00 %	75,00 %	75,00 %	6,189 s
	podnikatel.cz	63	124	123	50,81 %	51,22 %	51,01 %	15,987 s
	rockpapershotgun.com	100	100	100	100,00 %	100,00 %	100,00 %	5,556 s
	smartmania.cz	66	88	88	75,00 %	75,00 %	75,00 %	7,027 s
	techcrunch.com	88	88	96	100,00 %	91,67 %	95,65 %	6,618 s
	průměr	-	-	-	**80,08 %**	**79,29 %**	**79,67 %**	**9,308 s**
	celá sada	-	-	-	-	-	-	**59,514 s**

Obrázek 6.3: Výsledky úvodního experimentu pro sadu `news`.

Z výsledků třetí sady (viz tabulka 6.3) lze vidět následky některých již zmíněných problémů. Například stránka `artnews.com` má jako jediná v rámci experimentu hodnotu $F = 0\%$. To díky tomu, že jsou titulky článků delší než jejich shrnutí a názvy autorů kratší než časové značky. Všechny tyto atributy jsou pak navzájem prohozené. Pak u stránky `smartmania.cz` algoritmus zvolil počet komentářů místo časové značky.

Nový problém lze pozorovat u stránky `pcgamer.com`. Ke každému shrnutí algoritmus navíc extrahuje text ze sousedního elementu. Tento element byl téměř vyfiltrován detekcí opakujícího se obsahu (viz sekce 5.4.3), obsah jedné instance se ovšem lišil a tak ho algoritmus zahrnul v rámci extrakce obsahu z více elementů pod jeden atribut. Přestože díky tomu extrahovaný obsah neodpovídá očekávanému, což vedlo ke zhoršení výsledné hodnoty F v rámci testování, tak tento problém by v praxi nemusel být tak závažným, jelikož jde o obsah navíc, nikoliv jeho ztráta.

Další zajímavý problém se vyskytl u stránky `podnikatel.cz`. Zde u jednoho z článků chybí autor. Díky tomu nastala situace, kdy správná signatura měla o 1 výskyt méně než jiná, která byla postavena tak, aby započítala i článek s chybějícím autorem.

Datová sada	Stránka	v_C	v_O	v_E	P	R	F	T_F
football	365scores.com	45	45	54	100,00 %	83,33 %	90,91 %	7,368 s
	777score.com	462	616	616	75,00 %	75,00 %	75,00 %	7,717 s
	bbc.com	120	120	141	100,00 %	85,11 %	91,95 %	5,863 s
	flashscore.com	448	448	448	100,00 %	100,00 %	100,00 %	6,582 s
	flashscore.sk	316	316	472	100,00 %	66,95 %	80,20 %	6,500 s
	LiveScore.com	312	312	324	100,00 %	96,30 %	98,11 %	5,887 s
	scoreboard.com	431	431	815	100,00 %	52,88 %	69,18 %	6,650 s
	SkySports.com	800	800	800	100,00 %	100,00 %	100,00 %	6,371 s
	soccer24.com	318	318	484	100,00 %	65,70 %	79,30 %	6,937 s
	Soccerstand.com	440	440	440	100,00 %	100,00 %	100,00 %	6,300 s
	průměr	-	-	-	**97,50 %**	**82,53 %**	**88,47 %**	**6,618 s**
	celá sada	-	-	-	-	-	-	**29,615 s**
všechny	**průměr**	-	-	-	**83,47 %**	**76,92 %**	**79,45 %**	**6,895 s**
	celkem	-	-	-	-	-	-	**133,482 s**

Obrázek 6.4: Výsledky úvodního experimentu pro sadu `football` a pro všechny sady.

V rámci poslední sady naráží algoritmus u stránky `777score.com` na již zmíněný problém, kdy místo jednoho z týmů extrahuje jiný element lépe splňující vstupní podmínky.

Přestože výsledky 6.4 jsou ze všech sad nejlepší, atribut času zápasu je pro tuto sadu značný problém. Ten u některých zápasů (např. u stránek `flashscore.sk` nebo `soccer-24.com`) chybí, díky čemuž algoritmus tyto zápasy vynechává. Logickým krokem by bylo nastavení tohoto atributu jako nepovinného. V jednom z následujících experimentů (konkrétně v experimentu 6.3.5) se o tuto změnu pokusíme.

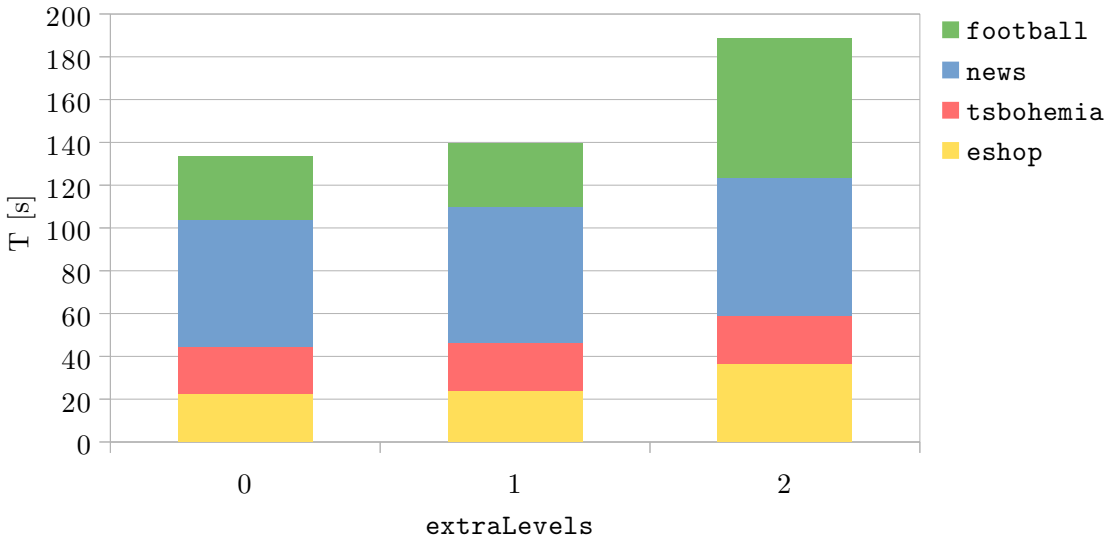
6.3.2 Extra iterace při hledání podstromu

Cílem navazujícího experimentu je zjistit dopad navýšení hodnoty parametru `extraLevels`. Ostatní parametry jsou ponechány na stejných hodnotách jako v úvodním experimentu.

Datová sada	Stránka	v_C	v_O	v_E	P	R	F	T_F
eshop	alza.cz	86	86	86	100,00 %	100,00 %	100,00 %	6,838 s
	czc.cz	46	69	85	66,67 %	54,12 %	59,74 %	6,059 s
	insportline.cz	162	220	224	73,64 %	72,32 %	72,97 %	6,328 s
	papirnictvieshop.cz	60	60	60	100,00 %	100,00 %	100,00 %	5,359 s
	smarty.cz	40	40	88	100,00 %	45,45 %	62,50 %	6,958 s
	sparkys.cz	10	46	48	21,74 %	20,83 %	21,28 %	6,093 s
	stavebninystastny.cz	30	40	40	75,00 %	75,00 %	75,00 %	6,044 s
	tsbohemia.cz	44	78	78	56,41 %	56,41 %	56,41 %	5,478 s
	zahradkarske-potreby.cz	30	45	45	66,67 %	66,67 %	66,67 %	5,182 s
	zahradnictvi-spomysl.cz	114	114	114	100,00 %	100,00 %	100,00 %	5,422 s
	průměr	-	-	-	**76,01 %**	**69,08 %**	**71,46 %**	**5,976 s**
	celá sada	-	-	-	-	-	-	**24,186 s**
všechny	**průměr**	-	-	-	**84,78 %**	**78,78 %**	**81,11 %**	**7,050 s**
	celkem	-	-	-	-	-	-	**139,601 s**

Obrázek 6.5: Výsledky po nastavení parametru `extraLevels` na hodnotu 1 pro sadu `eshop` a pro všechny sady. Zabarvení je oproti výsledkům úvodního experimentu 6.3.1.

Z tabulky 6.5 je vidět, že hledání jen o další úroveň dále úspěšně vyřešilo problém stránky `alza.cz` z úvodního experimentu, díky čemuž hodnota F pro tuto stránku dosáhla hodnoty 100%. Problémy jiných stránek již ovšem navýšení hodnoty neřeší. Navíc s navýšením hodnoty parametru můžeme sledovat patrné zpomalení algoritmu.



Obrázek 6.6: Graf závislosti hodnoty parametru `extraLevels` na čase T_F celých sad.

Krom očekávaného lineárního zpomalování (kvůli většímu počtu elementů na zkoumaný podstrom) můžeme na grafu 6.6 vidět, jak algoritmus u některých stránek reaguje na další úrovně při hledání podstromů výrazným zpomalením. Těmito stránkami jsou `zahradkarske-potreby.cz` a `777score.com`, u kterých navýšení parametru `extraLevels` z 1 na 2 způsobilo zpomalení na více než trojnásobek až pětinásobek původní doby.

To je způsobeno tím, že právě u těchto stránek další úroveň znamená, že algoritmus objevil podstrom obsahující velký počet kombinací. A ačkoliv tomuto zabránil u jiných stránek parametr `skipInstanceCombCount`, u těchto dvou stránek byl počet kombinací stále algoritmem označen za snesitelný.

Jako další krok by tedy stálo za uvážení parametr `skipInstanceCombCount` zkusit upravit takovým způsobem, aby takové případy nastávaly co nejméně.

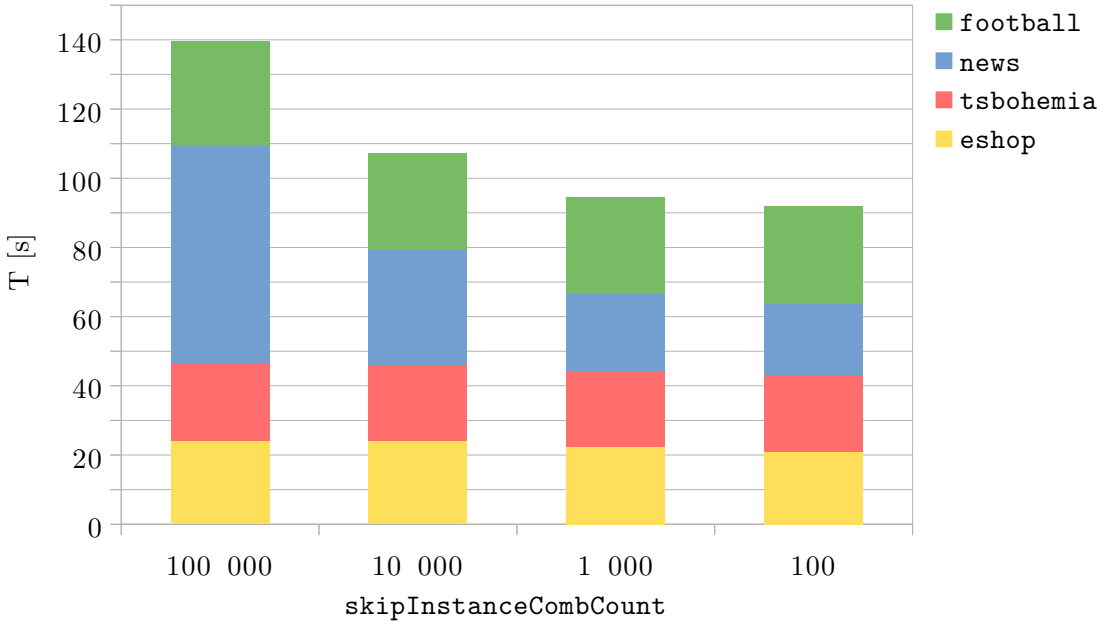
6.3.3 Snížení maximálního počtu kombinací na podstrom

V rámci následujícího experimentu bude cílem snížení hodnoty parametru `skipInstanceCombCount` za účelem zlepšení doby běhu algoritmu. Hodnoty ostatních parametrů budou stejné jako v předchozím experimentu 6.3.2 s parametrem `extraLevels` nastaveným na hodnotu 1.

Datová sada	Stránka	v_C	v_O	v_E	P	R	F	T_F
news	artnews.com	0	40	40	0,00 %	0,00 %	0,00 %	8,038 s
	ceskymac.cz	80	80	80	100,00 %	100,00 %	100,00 %	5,956 s
	cyclingnews.com	80	80	80	100,00 %	100,00 %	100,00 %	6,269 s
	google.com	40	40	40	100,00 %	100,00 %	100,00 %	5,093 s
	mndaily.com	40	40	40	100,00 %	100,00 %	100,00 %	7,688 s
	pcgamer.com	60	80	80	75,00 %	75,00 %	75,00 %	6,276 s
	podnikatel.cz	63	124	123	50,81 %	51,22 %	51,01 %	8,567 s
	rockpapershotgun.com	100	100	100	100,00 %	100,00 %	100,00 %	5,625 s
	smartmania.cz	66	88	88	75,00 %	75,00 %	75,00 %	7,031 s
	techcrunch.com	88	88	96	100,00 %	91,67 %	95,65 %	6,630 s
	průměr	-	-	-	**80,08 %**	**79,29 %**	**79,67 %**	**6,717 s**
	celá sada	-	-	-	-	-	-	**33,018 s**
všechny	**průměr**	-	-	-	**84,78 %**	**78,78 %**	**81,11 %**	**6,263 s**
	celkem	-	-	-	-	-	-	**107,290 s**

Obrázek 6.7: Výsledky po nastavení parametru `skipInstanceCombCount` na hodnotu 10 000 pro sadu `news` a pro všechny sady. Zabarvení je oproti výsledkům předchozího experimentu 6.3.2 s hodnotou parametru `extraLevels` nastavenou na 1.

Již po snížení o jeden řád je vidět z tabulky výsledků 6.7 poměrně značné zrychlení. Největší rozdíl byl zaznamenán u sady `news`, jejíž doba běhu je o téměř polovinu kratší – zejména díky stránkám `artnews.com`, `mndaily.com` a `podnikatel.cz`. U sady `tsbohemia` ovšem nepozorujeme žádnou změnu doby běhu. To potvrzuje tvrzení, že velmi záleží na struktuře dané stránky. U některých stránek hodnota předejde algoritmu před hledáním mezi zbytečným počtem kombinací, u jiných to ovšem nehrozí.



Obrázek 6.8: Graf závislosti hodnoty parametru `skipInstanceCombCount` na čase T_F celých sad.

Z grafu 6.8 lze pozorovat podobný trend i při dalším snižování hodnoty parametru. S dosažením hodnoty 100 ovšem narážíme na první ovlivnění výsledných získaných hodnot ze stránek, a to hlavně u sad `eshop` a `news`.

Datová sada	Stránka	v_C	v_O	v_E	P	R	F	T_F
eshop	alza.cz	72	78	86	92,31 %	83,72 %	87,80 %	6,149 s
	czc.cz	46	69	85	66,67 %	54,12 %	59,74 %	5,530 s
	insportline.cz	162	220	224	73,64 %	72,32 %	72,97 %	6,274 s
	papirnictvieshop.cz	60	60	60	100,00 %	100,00 %	100,00 %	5,357 s
	smarty.cz	14	18	88	77,78 %	15,91 %	26,42 %	5,417 s
	sparkys.cz	10	46	48	21,74 %	20,83 %	21,28 %	6,009 s
	stavebninystastny.cz	30	40	40	75,00 %	75,00 %	75,00 %	5,276 s
	tsbohemia.cz	44	78	78	56,41 %	56,41 %	56,41 %	5,474 s
	zahradkarske-potreby.cz	30	45	45	66,67 %	66,67 %	66,67 %	5,221 s
	zahradnictvi-spomysl.cz	114	114	114	100,00 %	100,00 %	100,00 %	5,254 s
	průměr	-	-	-	**73,02 %**	**64,50 %**	**66,63 %**	**5,596 s**
	celá sada	-	-	-	-	-	-	**20,861 s**
news	artnews.com	0	40	40	0,00 %	0,00 %	0,00 %	6,119 s
	ceskymac.cz	80	80	80	100,00 %	100,00 %	100,00 %	5,860 s
	cyclingnews.com	0	4	80	0,00 %	0,00 %	0,00 %	5,548 s
	google.com	40	40	40	100,00 %	100,00 %	100,00 %	5,082 s
	mndaily.com	20	40	40	50,00 %	50,00 %	50,00 %	5,719 s
	pcgamer.com	0	4	80	0,00 %	0,00 %	0,00 %	5,613 s
	podnikatel.cz	90	120	123	75,00 %	73,17 %	74,07 %	5,639 s
	rockpapershotgun.com	100	100	100	100,00 %	100,00 %	100,00 %	5,266 s
	smartmania.cz	88	88	88	100,00 %	100,00 %	100,00 %	5,517 s
	techcrunch.com	88	88	96	100,00 %	91,67 %	95,65 %	5,731 s
	průměr	-	-	-	**62,50 %**	**61,48 %**	**61,97 %**	**5,609 s**
	celá sada	-	-	-	-	-	-	**20,886 s**
všechny	**průměr**	-	-	-	**79,64 %**	**73,18 %**	**75,48 %**	**5,882 s**
	celkem	-	-	-	-	-	-	**91,882 s**

Obrázek 6.9: Výsledky po nastavení parametru `skipInstanceCombCount` na hodnotu 100 pro sady `eshop` a `news` a pro všechny sady.

Z výsledků v tabulce 6.9 lze pozorovat, že hodnota 100 již může způsobit zastavení hledání podstromu ještě před dosažením podstromu se správnou signaturou. Tento jev ovšem nastal pouze u 7 z celkového počtu 40 stránek. Navíc u 2 z nich zapříčinil nalezení dokonce z pohledu uživatele lepší signatury a tedy zlepšení výsledků u těchto stránek. Velmi tedy záleží na struktuře stránek. Hodnota parametru `skipInstanceCombCount` 1 000 se zdá být v rámci sad bezpečnou a zároveň – co se týče rychlosti algoritmu – také nejlepší a bude tak použita v následujících experimentech.

6.3.4 Změna chování extrakce více elementů na atribut

Jako poslední parametr k úpravě zbývá `maxAttributeTreeHeight`. V předchozích experimentech byla jeho hodnota rovna 1. Cílem tohoto experimentu je zjistit, jak se algoritmus bude chovat při odlišných hodnotách.

Budou testovány hodnoty 0 a 2, ostatní parametry budou nastaveny stejně jako u experimentu 6.3.3 s hodnotou parametru `skipInstanceCombCount` 1 000. U hodnoty 0 se dá očekávat zhoršení výsledků kvůli nemožnosti extrakce komplexnějších atributů, ale zároveň zrychlení – díky menšímu množství signatur. Hodnota 2 by měla mít opačné účinky – zpomalení a více signatur, mezi kterými vybírat.

Datová sada	Stránka	v_C	v_O	v_E	P	R	F	T_F
eshop	alza.cz	86	86	86	100,00 %	100,00 %	100,00 %	5,980 s
	czc.cz	46	69	85	66,67 %	54,12 %	59,74 %	5,464 s
	insportline.cz	162	220	224	73,64 %	72,32 %	72,97 %	5,696 s
	papirnictvieshop.cz	60	60	60	100,00 %	100,00 %	100,00 %	5,256 s
	smarty.cz	40	40	88	100,00 %	45,45 %	62,50 %	5,854 s
	sparkys.cz	10	46	48	21,74 %	20,83 %	21,28 %	5,878 s
	stavebninystastny.cz	20	30	40	66,67 %	50,00 %	57,14 %	5,185 s
	tsbohemia.cz	44	78	78	56,41 %	56,41 %	56,41 %	5,586 s
	zahradkarske-potreby.cz	30	45	45	66,67 %	66,67 %	66,67 %	5,365 s
	zahradnictvi-spomysl.cz	114	114	114	100,00 %	100,00 %	100,00 %	5,637 s
	průměr	-	-	-	**75,18 %**	**66,58 %**	**69,67 %**	**5,590 s**
	celá sada	-	-	-	-	-	-	**20,090 s**
tsbohemia	brusky	70	81	81	86,42 %	86,42 %	86,42 %	5,916 s
	chladice	70	73	73	95,89 %	95,89 %	95,89 %	5,957 s
	chladnicky	63	75	81	84,00 %	77,78 %	80,77 %	5,974 s
	disky	65	74	77	87,84 %	84,42 %	86,09 %	5,911 s
	kolobezky	61	73	73	83,56 %	83,56 %	83,56 %	5,846 s
	mobo	57	73	74	78,08 %	77,03 %	77,55 %	5,908 s
	monitory	64	73	74	87,67 %	86,49 %	87,07 %	5,894 s
	proc	67	75	78	89,33 %	85,90 %	87,58 %	5,877 s
	servery	71	72	72	98,61 %	98,61 %	98,61 %	5,946 s
	skenery	69	72	72	95,83 %	95,83 %	95,83 %	6,000 s
	průměr	-	-	-	**88,72 %**	**87,19 %**	**87,94 %**	**5,923 s**
	celá sada	-	-	-	-	-	-	**21,383 s**
news	artnews.com	0	40	40	0,00 %	0,00 %	0,00 %	6,299 s
	ceskymac.cz	80	80	80	100,00 %	100,00 %	100,00 %	5,716 s
	cyclingnews.com	80	80	80	100,00 %	100,00 %	100,00 %	5,641 s
	google.com	40	40	40	100,00 %	100,00 %	100,00 %	5,054 s
	mndaily.com	40	40	40	100,00 %	100,00 %	100,00 %	5,818 s
	pcgamer.com	80	80	80	100,00 %	100,00 %	100,00 %	5,813 s
	podnikatel.cz	63	124	123	50,81 %	51,22 %	51,01 %	5,789 s
	rockpapershotgun.com	100	100	100	100,00 %	100,00 %	100,00 %	5,315 s
	smartmania.cz	66	88	88	75,00 %	75,00 %	75,00 %	5,920 s
	techcrunch.com	88	88	96	100,00 %	91,67 %	95,65 %	6,263 s
	průměr	-	-	-	**82,58 %**	**81,79 %**	**82,17 %**	**5,763 s**
	celá sada	-	-	-	-	-	-	**21,842 s**
football	365scores.com	45	45	54	100,00 %	83,33 %	90,91 %	5,752 s
	777score.com	308	476	616	64,71 %	50,00 %	56,41 %	7,371 s
	bbc.com	120	120	141	100,00 %	85,11 %	91,95 %	5,956 s
	flashscore.com	448	448	448	100,00 %	100,00 %	100,00 %	6,603 s
	flashscore.sk	316	316	472	100,00 %	66,95 %	80,20 %	6,584 s
	LiveScore.com	234	312	324	75,00 %	72,22 %	73,58 %	5,924 s
	scoreboard.com	431	431	815	100,00 %	52,88 %	69,18 %	6,902 s
	SkySports.com	600	600	800	100,00 %	75,00 %	85,71 %	6,324 s
	soccer24.com	318	318	484	100,00 %	65,70 %	79,30 %	6,974 s
	Soccerstand.com	440	440	440	100,00 %	100,00 %	100,00 %	6,337 s
	průměr	-	-	-	**93,97 %**	**75,12 %**	**82,73 %**	**6,473 s**
	celá sada	-	-	-	-	-	-	**26,705 s**
všechny	**průměr**	-	-	-	**85,11 %**	**77,67 %**	**80,63 %**	**5,937 s**
	celkem	-	-	-	-	-	-	**90,020 s**

Obrázek 6.10: Výsledky po nastavení parametru `maxAttributeTreeHeight` na hodnotu 0 pro všechny sady. Zabarvení je oproti výsledkům experimentu 6.3.3 s hodnotou parametru `skipInstanceCombCount` 1 000.

Hodnoty z tabulky 6.10 u některých stránek potvrzují zhoršení výsledků kvůli nemožnosti extrakce obsahu složeného z více elementů. Např. stránka `stavebninystastny.cz` obsahuje slevu složenou z více elementů, díky čemuž se algoritmu nepodařilo najít element odpovídající regulárnímu výrazu popisujícímu atribut slevy a tak atribut u všech položek vynechává. Tímto trpí zejména sada `football`, kde se skóre často skládá z více elementů.

I v tomto experimentu se vyskytuje případ, kdy odebrání možností vede k vybrání z hlediska uživatele lepší signatury. Takovým případem je stránka `chladnicky`. Algoritmus již nemůže extrahovat informaci o stavu položky na skladě, protože je složena z více elementů. Díky tomu vybírá místo ní správný element s názvem položky.

Zúžení možností algoritmu pomáhá i v problému zmíněném v úvodním experimentu 6.3.1 stránky `pcgamer.com`. Algoritmus již nemůže extrahovat element navíc, který po něm ani nechceme, a tak jako náhradu algoritmus vybral popisek samotný, který již odpovídá očekávanému výstupu.

Dopad na dobu běhu algoritmu splnil očekávání, avšak celkové výsledky neutrpěly příliš velké škody.

Datová sada	Stránka	v_C	v_O	v_E	P	R	F	T_F
news	artnews.com	0	40	40	0,00 %	0,00 %	0,00 %	6,715 s
	ceskymac.cz	80	80	80	100,00 %	100,00 %	100,00 %	5,835 s
	cyclingnews.com	80	80	80	100,00 %	100,00 %	100,00 %	6,003 s
	google.com	40	40	40	100,00 %	100,00 %	100,00 %	5,320 s
	mndaily.com	40	40	40	100,00 %	100,00 %	100,00 %	6,096 s
	pcgamer.com	60	80	80	75,00 %	75,00 %	75,00 %	6,107 s
	podnikatel.cz	63	124	123	50,81 %	51,22 %	51,01 %	7,075 s
	rockpapershotgun.com	0	4	100	0,00 %	0,00 %	0,00 %	5,322 s
	smartmania.cz	66	88	88	75,00 %	75,00 %	75,00 %	5,757 s
	techcrunch.com	88	88	96	100,00 %	91,67 %	95,65 %	6,037 s
	průměr	-	-	-	**70,08 %**	**69,29 %**	**69,67 %**	**6,027 s**
	celá sada	-	-	-	-	-	-	**23,268 s**
football	365scores.com	45	45	54	100,00 %	83,33 %	90,91 %	5,672 s
	777score.com	462	616	616	75,00 %	75,00 %	75,00 %	7,973 s
	bbc.com	120	120	141	100,00 %	85,11 %	91,95 %	6,003 s
	flashscore.com	448	448	448	100,00 %	100,00 %	100,00 %	6,650 s
	flashscore.sk	316	316	472	100,00 %	66,95 %	80,20 %	6,785 s
	LiveScore.com	0	0	324	0,00 %	0,00 %	0,00 %	5,891 s
	scoreboard.com	431	431	815	100,00 %	52,88 %	69,18 %	6,953 s
	SkySports.com	800	800	800	100,00 %	100,00 %	100,00 %	6,921 s
	soccer24.com	318	318	484	100,00 %	65,70 %	79,30 %	7,523 s
	Soccerstand.com	440	440	440	100,00 %	100,00 %	100,00 %	6,778 s
	průměr	-	-	-	**87,50 %**	**72,90 %**	**78,65 %**	**6,715 s**
	celá sada	-	-	-	-	-	-	**29,000 s**
všechny	**průměr**	-	-	-	**79,78 %**	**73,87 %**	**76,16 %**	**6,217 s**
	celkem	-	-	-	-	-	-	**99,564 s**

Obrázek 6.11: Výsledky po nastavení parametru `maxAttributeTreeHeight` na hodnotu 0 pro sady `news` a `football` a pro všechny sady. Zabarvení je oproti výsledkům experimentu 6.3.3 s hodnotou parametru `skipInstanceCombCount` 1 000.

Zvýšení hodnoty se ve výsledku také jeví negativně a ve výsledcích z tabulky 6.10 je na stránkách `rockpapershotgun.com` a `LiveScore.com` vidět, že algoritmus v některých případech zvýšenou komplexnost obsahu atributů nezvládá. Zároveň nastalo předvídatelné navýšení doby běhu algoritmu.

6.3.5 Nepovinný atribut času u sady `football`

Po podrobném otestování efektu parametrů na výsledky algoritmu zbývá experiment soustředěný na problém zmíněný v úvodním experimentu 6.3.1 u sady `football` – u jednotlivých zápasů často chybí čas konání zápasu, který je ve vstupním souboru sady uveden jako povinný atribut. Cílem tohoto experimentu je zjistit, jak se algoritmus bude chovat, pokud bude atribut času nastavený ve vstupních datech jako volitelný. Ostatní parametry jsou nastaveny stejně jako u experimentu 6.3.3 s hodnotou `skipInstanceCombCount` nastavenou na 1 000.

Datová sada	Stránka	v_C	v_O	v_E	P	R	F	T_F
football	365scores.com	0	18	54	0,00 %	0,00 %	0,00 %	7,255 s
	777score.com	444	444	616	100,00 %	72,08 %	83,77 %	27,693 s
	bbc.com	0	58	141	0,00 %	0,00 %	0,00 %	12,935 s
	flashscore.com	0	144	448	0,00 %	0,00 %	0,00 %	22,204 s
	flashscore.sk	0	168	472	0,00 %	0,00 %	0,00 %	21,617 s
	LiveScore.com	242	243	324	99,59 %	74,69 %	85,36 %	14,683 s
	scoreboard.com	521	522	815	99,81 %	63,93 %	77,94 %	23,854 s
	SkySports.com	6	426	800	1,41 %	0,75 %	0,98 %	44,914 s
	soccer24.com	0	168	484	0,00 %	0,00 %	0,00 %	24,155 s
	Soccerstand.com	0	142	440	0,00 %	0,00 %	0,00 %	22,049 s
	průměr	-	-	-	**30,08 %**	**21,14 %**	**24,80 %**	**22,136 s**
	celá sada	-	-	-	-	-	-	**195,117 s**

Obrázek 6.12: Výsledky po nastavení parametru `optional` atributu `time` sady `football` na hodnotu `true`. Zabarvení je oproti výsledkům experimentu 6.3.3 s hodnotou parametru `skipInstanceCombCount` 1 000.

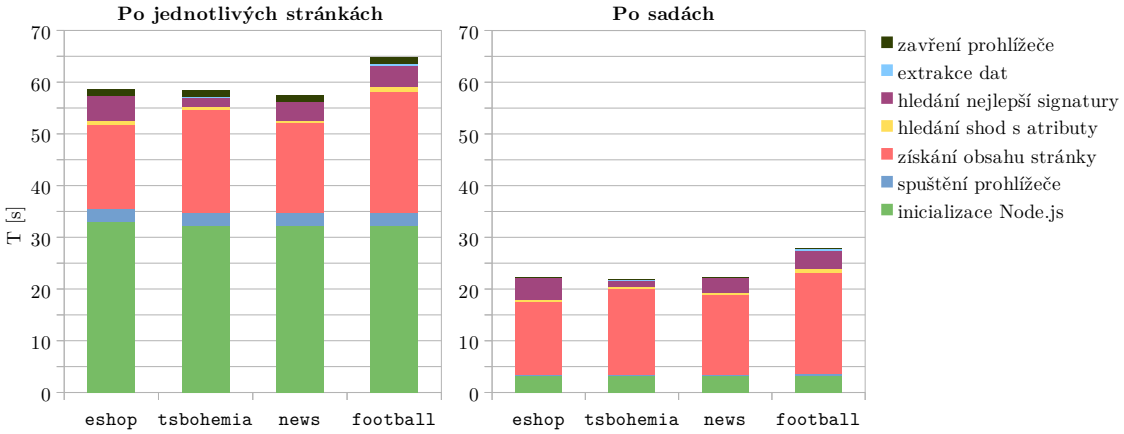
Z výsledků v tabulce 6.12 lze pozorovat, že algoritmus má po této úpravě vstupních dat se sadou velké problémy. Úprava zlepšila výsledky pouze u 2 z 10 stránek, zatímco u všech ostatních jsou výsledky buď horší, nebo u nich algoritmus selhal úplně.

Velkým problémem tohoto nastavení jsou zbývající 2 povinné atributy. Oba jsou specifikovány velmi obecně (pouze jako řetězce s danou délkou). Zbylé parametry s přesnější specifikací pomocí regulárních výrazů byly pro algoritmus ideální pro výběr počátečních elementů pro hledání podstromů s možnými instancemi hledaného objektu. Jakmile byl ovšem i atribut času nastaven jako volitelný, algoritmus musel zvolit jako počáteční jeden ze zbývajících obecnějších atributů, což vedlo k mnohonásobnému navýšení iterací hledání podstromů a tedy velmi výraznému zpomalení algoritmu. Běh pouze této sady trval déle, než všechny sady dohromady ve většině předchozích experimentů.

Krom zpomalení navíc způsobuje obecnost specifikace atributů také hledání velkého množství signatur, které taktéž odpovídají těmto atributům. A protože takto jednoduchá specifikace atributů odpovídá často i třeba nadpisům jednotlivých zápasů, algoritmus často vybírá právě takové okolní konstrukce, které mají v rámci této sady často větší četnost, kterou algoritmus uvažuje jako hlavní metriku.

6.4 Analýza rychlosti algoritmu

Doba běhu algoritmu na sadu se výrazně liší mezi případy, kdy je spuštěn algoritmus na jednotlivé stránky, nebo na celou sadu najednou. Při spuštění algoritmu po sadách během experimentu 6.3.3 s hodnotou parametru `skipInstanceCombCount` 1 000 byla naměřena celková doba pro všechny sady 94,593 sekund. Jestliže je algoritmus spuštěn se stejnými parametry pro všechny sady, ale pro každou stránku odděleně, celková doba běhu se prodlouží na 239,691 sekund – na více než 250% původní doby. Spuštěním algoritmu po jednotlivých sadách je ušetřeno průměrně 3,63 sekundy doby běhu na stránku oproti spuštění po jednotlivých stránkách.

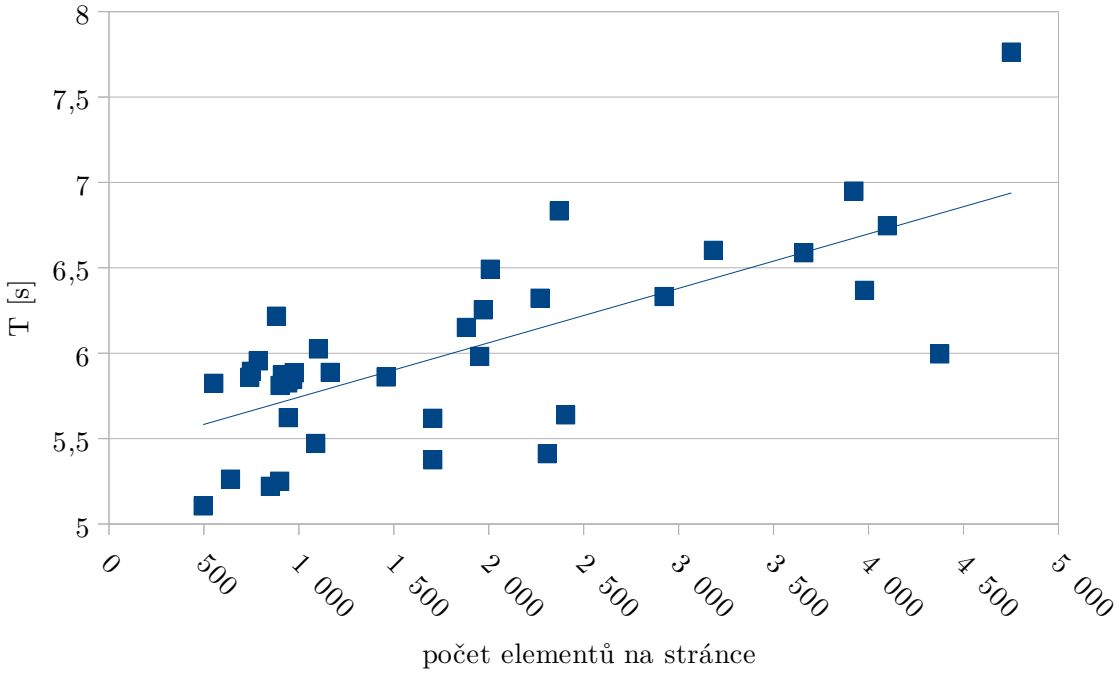


Obrázek 6.13: Grafy doby běhu jednotlivých částí algoritmu pro každou sadu. Hodnoty naměřeny s parametry z experimentu 6.3.3 s hodnotou parametru `skipInstanceCombCount` 1 000.

Na grafu 6.13 lze vidět, jak algoritmus při spuštění pro více stránek najednou aplikuje optimalizace režie knihovny Puppeteer a s ním spojeného prohlížeče pro získání dat ze stránky. Zejména ovšem ušetří opakovanou inicializaci prostředí a všech potřebných knihoven. Dá se očekávat, že poměr ušetřené doby bude mít přímou úměrnost s počtem stránek v rámci sady – čím více stránek, tím více ušetřené režie, naopak sada o dvou stránkách ušetří pouze jednu inicializaci prostředí.

Také lze pozorovat, že při spuštění algoritmu na sady tvoří největší část zátěže získání obsahu stránky – více než 70% celkové doby běhu programu. V reálném světě je pravděpodobné další navýšení tohoto podílu o čekání na načtení stránky. Samotná logika algoritmu (v grafu části hledání shod s atributy, hledání nejlepší signatury a extrakce dat) v tom samém scénáři tvoří pouze málo přes 15% celkové doby běhu.

Ačkoliv je algoritmus zpomalen inicializací prostředí, samotný obsah stránky má na rychlost algoritmu dopad také.



Obrázek 6.14: Graf závislosti počtu elementů na stránce na hodnotě T_F. Hodnoty naměřeny s parametry z experimentu 6.3.3 s hodnotou parametru `skipInstanceCombCount` 1 000.

Na grafu 6.14 lze vidět korelaci mezi celkovým počtem elementů na stránce a dobou běhu algoritmu pro danou stránku. Lze říci, že algoritmu trvá extrakce dat ze stránek s více elementy obvykle delší dobu, než ze stránek s menším počtem elementů.

6.5 Možná vylepšení

Algoritmus má prostor pro zlepšení a další experimentální úpravy hned v několika směrech.

Uživatelská přívětivost by mohla být zlepšena pohodlnějším formátem vstupních dat. Nebo podporou více datových typů, jako např. hexadecimálních čísel či dat, které by navíc umožnily definování rozsahu. Algoritmus by mohl také podporovat stránkování pro umožnění extrakce dat z více stránek v případě seznamu položek rozděleného na více stránek.

K přesnosti algoritmu by mohla přispět možnost poskytnutí více vstupních informací, jako popis okolního obsahu chtěného atributu, či regulárních výrazů, které by nutně nemusely projít, ale sloužily by pro algoritmus jako nápověda. Další podobnou vlastností by mohla být možnost definice atributu datového typu komplexnější struktury složené z více atributů.

Případně se nabízí další experimentální možnosti, jako výjimečné povolení hodnot mimo definovaný rozsah atributu, extrakce dat i z atributů elementů webové stránky, použití odlišného řazení statistik při výběru signatury, či použití jiného formátu než indexové cesty k zapamatování polohy atributů. Současně algoritmus poskytuje prostor pro rozšíření za použití klasifikátorů.

Kapitola 7

Závěr

Cílem této práce bylo vytvořit algoritmus pro extrakci dat z webových stránek bez nutnosti znalosti jejich vnitřní struktury.

K dosažení cíle byly použity aktuální technologie, spolu se znalostmi o moderních přístupech k extrakci dat z webových stránek. Implementace algoritmu proběhla dle návrhu aplikace a k jeho vyhodnocení bylo použito několik datových sad, které prošly několika experimenty za účelem podrobné analýzy jejich výsledků a tím i funkčnosti algoritmu.

Výsledný algoritmus prokázal úspěšnost při extrakci dat více než 80%. U 11 z celkem 40 testovaných stránek algoritmus dosáhl dokonce 100% úspěšnosti, tedy stoprocentní shody očekávaného výsledku s jeho výstupem. O algoritmu se tedy dá říci, že jeho výsledky dokazují jeho rozšířitelnost, či použitelnost v reálném světě.

Pro mě osobně měla práce přínos v podobě seznámení s novými a v poslední době velmi populárními technologiemi, jako je programovací jazyk TypeScript, nebo knihovna Puppeteer. Zároveň mi práce rozšířila obzory do aktuálního stavu světa extrakce dat z webových stránek.

Práce nechává mnoho možností, jak na ni navázat. Lze ji vylepšit jak po stránce uživatelské přívětivosti, tak i dalšími experimentálními změnami za účelem zlepšení výsledků algoritmu. Nejvýznamnější možností je asi zapojení strojového učení – trénování algoritmů na datových sadách webových stránek za účelem učení algoritmu k lepšímu rozpoznávání dat na stránkách.

Literatura

[1] Alarte, J., Insa, D., Silva, J. a Tamarit, S. Main Content Extraction from Heterogeneous Webpages. In: *International Conference on Web Information Systems Engineering*. Springer US, říjen 2018, sv. 11233, s. 393–407. DOI: 10.1007/978-3-030-02922-7_27. ISBN 978-3-030-02921-0. Dostupné z: https://doi.org/10.1007/978-3-030-02922-7_27.

[2] Bizer, C., Meusel, R. a Primpeli, A. Extraction Results from the September 2020 Common Crawl Corpus. *Web Data Commons - Microdata, RDFa, JSON-LD, and Microformat Data Sets* [online]. University of Mannheim, 2020 [cit. 2021-04-27]. Dostupné z: http://webdatacommons.org/structureddata/index.html#toc4.

[3] Burget, R. Model-Based Integration of Unstructured Web Data Sources Using Graph Representation of Document Contents. In: *15th International Conference on Web Information Systems and Technologies* [online]. Vienna: SciTePress - Science and Technology Publications, 2019, s. 326–333. ISBN 978-989-758-386-5. Dostupné z: https://www.scitepress.org/PublicationsDetail.aspx?ID=6FSPso19Eo8=&t=1.

[4] Burget, R. *Extrakce dat z webu: Aka webscraping* [online]. 2020 [cit. 2021-03-21]. Dostupné z: https://www.fit.vutbr.cz/~burgetr/upa/05_webscraping/.

[5] Burget, R. *XML: A související technologie* [online]. 2020 [cit. 2021-03-27]. Dostupné z: https://www.fit.vutbr.cz/~burgetr/iis/p08_xml/.

[6] Down, T. *Is there a way to get innerText of only the top element (and ignore the child element's innerText)?* [online]. Stack Overflow, 2012 [cit. 2021-02-07]. Dostupné z: https://stackoverflow.com/a/9340862/.

[7] Dustdar, S. a Falchuk, B. Semantic Web. In: Furht, B., ed. *Encyclopedia of Multimedia*. Boston, MA: Springer US, 2006, s. 803–810. DOI: 10.1007/0-387-30038-4_220. ISBN 978-0-387-30038-2. Dostupné z: https://doi.org/10.1007/0-387-30038-4_220.

[8] Lei, K., Ma, Y. a Tan, T. Performance Comparison and Evaluation of Web Development Technologies in PHP, Python, and Node.js. In: *2014 IEEE 17th International Conference on Computational Science and Engineering*. IEEE, 2014, s. 661–668. ISBN 978-1-4799-7981-3. Dostupné z: https://ieeexplore.ieee.org/abstract/document/7023652.

[9] Maududie, A., Retnani, W. E. Y. a Rohim, M. A. An Approach of Web Scraping on News Website based on Regular Expression. In: IEEE. *The 2nd East Indonesia Conference on Computer and Information Technology (EIConCIT)*. IEEE, Listopad

2018, s. 203–207. DOI: 10.1109/EIConCIT.2018.8878550. ISBN 978-1-5386-8051-3. Dostupné z: https://doi.org/10.1109/EIConCIT.2018.8878550.

[10] MDN a jednotliví přispěvatelé. *Introduction to the DOM* [online]. MDN Web Docs, 2021. 2021-03-16 [cit. 2021-03-27]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction.

[11] MDN a jednotliví přispěvatelé. *JavaScript* [online]. MDN Web Docs, 2021. 2021-02-21 [cit. 2021-03-30]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript/.

[12] MDN a jednotliví přispěvatelé. *Introduction to the JavaScript* [online]. MDN Web Docs, 2021. 2021-03-09 [cit. 2021-03-27]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Introduction.

[13] *Npm Docs: About npm* [online]. npm [cit. 2021-03-30]. Dostupné z: https://docs.npmjs.com/about-npm.

[14] Patel, P. *What exactly is Node.js* [online]. freeCodeCamp, duben 2018 [cit. 2021-03-29]. Dostupné z: https://www.freecodecamp.org/news/what-exactly-is-node-js-ae36e97449f5/.

[15] Perina, L. *Metody extrakce dat z webových stránek*. Brno, CZ, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií.

[16] jednotliví přispěvatelé. *Puppeteer* [online]. GitHub, 2021 [cit. 2021-03-31]. Dostupné z: https://github.com/puppeteer/puppeteer.

[17] jednotliví přispěvatelé. *Puppeteer: Overview* [online]. 2021 [cit. 2021-03-31]. Dostupné z: https://pptr.dev/#&show=api-overview.

[18] Sirisuriya, D. S. A Comparative Study on Web Scraping. In: *Proceedings of 8th International Research Conference, KDU*. Department of Computer Science, Faculty of Computing, General Sir John Kotelawala Defence University, Ratmalana, Sri Lanka, Listopad 2015, s. 135–140. ISBN 978-955-0301-24-9. Dostupné z: http://ir.kdu.ac.lk/handle/345/1051.

[19] Ting, K. M. Precision and Recall. In: Sammut, C. a Webb, G. I., ed. *Encyclopedia of Machine Learning*. Boston, MA: Springer US, 2010, s. 781–781. DOI: 10.1007/978-0-387-30164-8_652. ISBN 978-0-387-30164-8. Dostupné z: https://doi.org/10.1007/978-0-387-30164-8_652.

[20] Microsoft a jednotliví přispěvatelé. *The TypeScript Handbook* [online]. Microsoft, 2021 [cit. 2021-03-29]. Dostupné z: https://www.typescriptlang.org/docs/handbook/.

[21] Microsoft a jednotliví přispěvatelé. *The TypeScript Handbook: Playground Examples – Enums* [online]. Microsoft, 2021 [cit. 2021-03-30]. Dostupné z: https://www.typescriptlang.org/play?q=36#example/enums.

[22] Microsoft a jednotliví přispěvatelé. *The TypeScript Handbook: The Basics* [online]. Microsoft, 2021 [cit. 2021-03-30]. Dostupné z: https://www.typescriptlang.org/docs/handbook/2/basic-types.html#non-exception-failures.

Příloha A

Obsah přiloženého média

Základní přehled obsahu přiloženého média:

- `doc/` – Adresář se zdrojovými soubory textu práce.
- `inputs/` – Adresář s ukázkovými vstupními soubory.
- `src/` – Adresář se zdrojovými soubory implementace programu.
- `tests/` – Adresář s testovacím programem a datovými sadami použitými v rámci vyhodnocení.
- `README.md` – Textový soubor obsahující podrobné informace o obsahu přiloženého média a offline verzích stránek v datových sadách použitých při vyhodnocení. Obsahuje také návod ke spuštění.