Univerzita Hradec Králové
Fakulta informatiky a managementu

Web crawlery

Bakalářská práce

Autor: Filip Roškot
Studijní obor: Aplikovaná informatika

Vedoucí práce: Mgr. Daniela Ponce, Ph.D.

Hradec Králové Srpen 2019

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 23.8.2019 Filip Roškot

Poděkování:

Děkuji vedoucí bakalářské práce Mgr. Daniele Ponce, Ph.D. za metodické a svědomité vedení bakalářské práce a notnou dávku tolerance a vstřícnosti. Dále také osudu, že jsem byl schopen překonat již tak sužující prokrastinaci a dokončit tak tuto práci.

Anotace

Cílem této bakalářské práce je seznámit čtenáře s problematikou vytváření sitemap na základě analýzy webového sídla pomocí robota (web crawler), následně přiblížení konkrétních postupů dolování dat z webového sídla a jejich přístupnost dalším technologiím. Text této práce je určen zejména pro osoby, které měly již možnost seznámit se se základy fungování webových aplikací. Technologické souvislosti úzce spjaté se správným fungováním robota jsou popsány autorem, jelikož jsou považovány za nutnou znalost k pochopení významu web crawleru. V práci je popsáno několik způsobů využití sitemapy pro vybrané cílové skupiny uživatelů. Závěrem čtenář získá vědomosti o fungování robota, jeho pohybu po webových sídlech a výhodách, které z jeho využití plynou.

Klíčová slova:

Crawler, Scraper, Robot, Scrapy, Sitemapa

Annotation

Title: Web crawlers

The aim of this bachelor thesis is to get the reader familiar with the issue of creating a sitemap based on the analysis of the web site using a robot (web crawler), then the approach of specific procedures of data mining from the site and their accessibility to other technologies. The text of this work is designed especially for people who have already had the opportunity to become familiar with the basics of web applications. Technological contexts closely related to the proper functioning of the robot are described by the author as they are considered necessary knowledge to understand the importance of web crawler. The thesis describes several ways of using sitemap for selected target groups of users. Finally, the reader gains knowledge about the functioning of the robot, its movement around the website and the benefits of its use.

Obsah

1 Úvod

Tato bakalářská práce popisuje oblast automatického zpracovávání webového obsahu za primárním účelem vytvoření sitemap. Popisuje strukturu sitemapy a její formáty. Dále čtenáře seznámí se základy dolování dat, procesem procházení a analýzy webové stránky a představí proces webového crawlingu. V praktické části je představena implementace v jazyku Python za pomoci využití několika knihoven a frameworku Scrapy, který primárně slouží ke scrapingu webových stránek. V práci bude dále prezentována grafická podoba sitemapy, jež je mnohem přívětivější pro člověka. V neposlední řadě je navrhnuto praktické využití sitemap pro několik cílových skupin uživatelů. V závěru práce došlo k rekapitulaci a shrnutí výsledků.

V práci je využívána terminologie převážně v anglickém jazyce s aplikací české formy skloňování a časování tvarů podstatných jmen a sloves, a to z důvodu neexistujících adekvátních a běžně používaných českých verzí, které by mohly dané termíny zastoupit. Jedná se především o termíny sitemapa, crawler, scraper, parser, mining a všechny jejich různé verze, které jsou v práci vysvětleny a opakovaně používány.

2 Úvod do problematiky

V této kapitole je vysvětlen účel souboru *robots.txt*, sitemapa a její formáty, základy parsování, data mining, crawler a samotný princip web crawlingu. A v neposlední řadě kapitola porovnává scraping vs. crawling.

2.1 Robots.txt

Úvodem je nutné představit na první pohled prostý textový soubor s pojmenováním *robots.txt*. Soubor se vždy nachází v kořenovém adresáři webového sídla a odpovídá standardu Robots Exclusion Standard[1]. Je složen z jednoho nebo více pravidel. Každé jednotlivé pravidlo zařizuje omezení nebo povolení pro příchozí webové roboty (crawlery) a definuje jim tím přístup k určitým částem webového sídla. Níže zmíněný příklad se dvěma pravidly demonstruje funkčnost takového souboru [3].

```
# Pravidlo 1
User-agent: Googlebot
Disallow: /googlebot_ne/


# Pravidlo 2
User-agent: *
Allow: /

Sitemap: http://www.example.com/sitemap.xml
```

Webovému robotovi s názvem Googlebot je zakázán přístup, tedy prohledávání všech složek s cestou /googlebot_ne/ a jejími podsložkami. Všichni ostatní roboti mají přístup do celého webu bez žádného dalšího omezení. Je nutné podotknout, že druhé pravidlo není zcela nutné, protože úplný přístup všem robotům je udělen ve výchozím nastavení.

[1] Protokol pro zakázání přístupu robotům [4]

2.2 Soubor sitemap

Sitemapa neboli správa mapy webu je nedílnou součástí každé validní webové stránky, která splňuje standardní požadavky W3C[2] a normy zvolené verze značkovacího jazyka HTML[3] a CSS[4]. Mapa je reprezentována jako soubor umístěný na stejném webovém serveru společně s ostatními daty webového sídla, ke které mají přístup weboví vyhledávací roboti či „boti“ (web crawlers), kteří tento soubor načtou a získají tak informace o jednotlivých stránkách, videích a dalších souborech společně s jednotlivými vztahy mezi nimi. Robot se ze souboru sitemap například dozví, jaké stránky považuje za relevantní, které autor webového sídla určil, či naopak zakázal k navštívení, a získá o nich cenné informace, jako je datum jejich poslední změny, jak často k takovým změnám dochází a jaké existují alternativní verze. Web crawler je schopen si poradit ve většině případů na webových sídlech bez existujícího souboru s mapou webu, ale značně to navyšuje celkovou časovou náročnost a v některých případech může dojít k nenalezení všech žádaných informací, proto je správně napsaná mapa v dnešní době doporučenou součástí každého webu [5].

2.2.1 Obrázky

Nedílnou součástí každého webu jsou také obrázky či fotografie. Za účelem co nejpřesnějšího vyhledávání obrázku je vhodné přidat jednotlivé obrázky do sitemapy pomocí rozšířeného standardu pro obrázky od společnosti Google. Tímto lze poskytnout další informace, jako je cílová URL adresa obrázku, lokální adresa, popis, geografické umístění a název. Sitemapou pro obrázky lze určit, které obrázky vůbec chceme nechat indexovat robotem a které chceme naopak před robotem skrýt. Vhodné je také zmínit, že soubor sitemap pro obrázky napomáhá objevení obrázků, které se vkládají například pomocí javascriptu a mohly by v normálních okolnostech zůstat nepovšimnuty [6].

```
<?xml version="1.0" encoding="UTF-8"?>
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9"
```

[2] World Wide Web Consortium
[3] Hypertext Markup Language
[4] Cascading Style Sheets (Kaskádové styly)

```
        xmlns:image="http://www.google.com/schemas/sitemap-
image/1.1">
  <url>
    <loc>http://example.com/ukazka.html</loc>
    <image:image>
      <image:loc>http://example.com/obrazek.jpg</image:loc>
    </image:image>
    <image:image>
      <image:loc>http://example.com/fotka.jpg</image:loc>
    </image:image>
  </url>
</urlset>
```

2.2.2 Video

Stejně jako obrázky lze zpracovat i obsah s videi. Společnost Google tak nabízí vlastní zpracování pomocí rozšířeného standardu sitemap obsahujících další informace o videu hostovaném na stránkách, které napomohou crawlerům najít videoobsah a porozumět mu. Platí to zejména pro nedávno přidaný obsah, který by nemusel býti crawlery tak snadno objeven. Každá položka v souboru sitemap má poté sadu povinných (url, loc, video apod.), doporučených (duration) nebo volitelných hodnot (platform, price), viz ukázka níže [7].

```
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9"
        xmlns:video="http://www.google.com/schemas/sitemap-
video/1.1">
   <url>

<loc>http://www.example.com/videa/vstupni_stranka_videa.html
</loc>
     <video:video>

<video:thumbnail_loc>http://www.example.com/miniatury/123.jp
g</video:thumbnail_loc>
       <video:title>Letní grilování steaků</video:title>
```

```
      <video:description>Alkis ukazuje, jak pokaždé
dosáhnout perfektně propečených
        steaků</video:description>
      <video:content_loc>

http://streamserver.example.com/video123.mp4</video:content_
loc>
      <video:player_loc>

http://www.example.com/videoplayer.php?video=123</video:play
er_loc>
      <video:duration>600</video:duration>
      <video:expiration_date>2021-11-
05T19:20:30+08:00</video:expiration_date>
      <video:rating>4.2</video:rating>
      <video:view_count>12345</video:view_count>
      <video:publication_date>2007-11-
05T19:20:30+08:00</video:publication_date>
      <video:family_friendly>yes</video:family_friendly>
      <video:restriction relationship="allow">IE GB US
CA</video:restriction>
      <video:price currency="EUR">1.99</video:price>

<video:requires_subscription>yes</video:requires_subscriptio
n>
      <video:uploader

info="http://www.example.com/uzivatele/jankuchar">Jan Kuchař
      </video:uploader>
      <video:live>no</video:live>
    </video:video>
  </url>
</urlset>
```

2.3 Formát sitemap

V dnešní době se nabízí hned několik způsobů formátování sitemap. Vhodné je provést výběr jedné z níže zmíněných a vytvořit ji tak ručně, pomocí nástrojů třetích stran, nebo nechat vygenerovat. Posledním krokem je zpřístupnění sitemapy web crawlerům. Toho lze docílit přidáním konkrétního robota do *robots.txt* společně s vytyčením hranic, nebo odesláním sitemapy do služby Search Console od společnosti Google.

2.3.1 XML

Světově nejrozšířenější formát sitemap je pomocí dokumentu v XML, který popisuje protokol schématu takové mapy. Musí:

1. obsahovat otevření a uzavření pomocí párového tagu <urlset>;
2. být specifikován s parametrem namespace v tagu urlset;
3. obsahovat <url> tag pro každou URL adresu;
4. obsahovat vnořený tag <loc> v každém z výše zmíněných <url>.

Kupříkladu níže zobrazený příklad [8]:

```
<?xml version="1.0" encoding="UTF-8"?>
<urlset
xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
   <url>
      <loc>http://www.example.com/</loc>
      <lastmod>2005-01-01</lastmod>
      <changefreq>monthly</changefreq>
      <priority>0.8</priority>
   </url>
</urlset>
```

2.3.2 RSS

Další z možností, avšak v dnešní době téměř nepoužívanou, je formát RSS. Nejvíce je využíván v Blog systémech nebo CMS, které většinou disponují nástroji pro vytváření zdroje RSS. Po takto vytvořeném zdroji nezbývá nic jiného než jeho URL adresu odeslat

jako soubor sitemap. Zdroj však poskytuje informace pouze o adresách z poslední doby [9].

2.3.3 Text

Soubor sitemap lze prezentovat pouze jako seznam URL adres zadaných na jednotlivém řádku. Tato možnost je velice základní a dost omezená kvůli nedostatku podrobností o jednotlivých odkazech. Při vytváření souboru je nutné dodržet kódování UTF-8, soubor nesmí obsahovat nic jiného, než jsou URL adresy, a musí mít příponu *.txt* s libovolným názvem [10].

2.4 Parsování

Parser je část kompileru nebo interpretera, která rozkládá data na menší části jednotlivých elementů pro jednoduchý překlad do jiných jazyků. Parser vezme vstupní data a vloží je do sekvenčního formuláře nebo programových instrukcí, které obvykle přetváří data do stromové datové struktury. Celkový proces parsování se dělí do tří fází. První fází je lexikální analýza, která se používá k rozdělení znaků vstupních řetězců na malé části, a vytváření smysluplných výrazů. Druhou je syntaktická analýza, která kontroluje, zdali jsou generované části opravdu smysluplné výrazy na základě porovnávání gramatiky bez kontextu, kterou definují algoritmické procedury. Tato fáze definuje konkrétní pořadí, ve kterém musí být jednotlivé části umístěny. Posledním krokem je sémantické parsování, jež je posledním stádiem syntaktické analýzy, ve které je určen význam a důsledek ověřeného výrazů a jsou učiněny potřebné kroky [11].

2.5 Data mining

Data mining je definováno jako „dolování informací z webových sídel“ [16]. Tato definice nabízí hned několik otázek. Proč dolování informací? Jaký typ informací je získáván? Jak robot pozná, co je relevantní? K těmto otázkám se váže nespočet problémů, které ve většině případů mají odpověď či úměrné řešení.

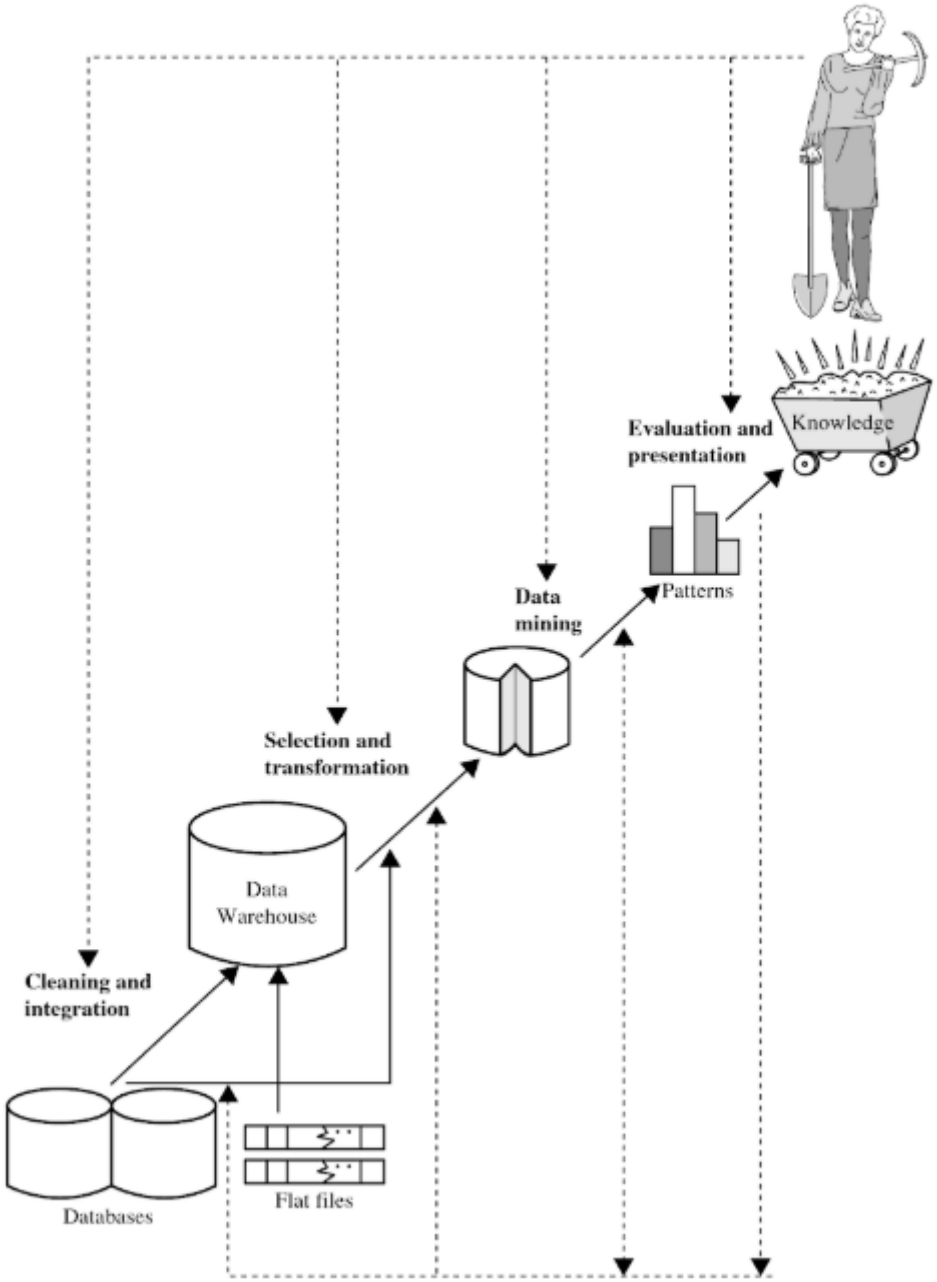
Pod pojmem data mining můžeme rozumět získávání vědomostí z dat, dolování dat či dolování vědomostí. Žádný z výše zmíněných názvů zcela nereflektuje pravý význam. Obecně jde o jev, kdy dochází k systematickému prohledávání rozsáhlého textu

v jakékoli podobě, který by měl být relevantní, za účelem získání námi žádaného výsledku.
Podoba textu může nabýt několika forem, nejčastěji data stream, sekvenční data, grafy nebo síťová data, textová data, multimediální data a WWW data (HTML kód, JSON, XML apod.) [16]

Proces získávání dat se skládá z následujících kroků:

1. Čištění dat (odstranění nekonzistentních dat)
2. Integrace dat (kombinace několika zdrojů dat)
3. Selekce dat (relevantní data k analýze jsou obdržena z databáze)
4. Transformace dat (data jsou transformována do formulářů sloužících k provádění souhrnných nebo agregačních operací)
5. Data mining (proces, kdy dochází k aplikaci metod sloužících k extrakci dat)
6. Hodnocení vzorů (identifikace skutečně zajímavých vzorů reprezentujících vědomosti, které jsou námi požadované)
7. Prezentace znalostí (vizualizace a znalosti reprezentující techniky jsou použity k zobrazení dat)

Výše zmíněné kroky 1–4 se mohou lišit v závislosti na preprocesu, kdy jsou data teprve připravována pro samotnou extrakci. Pátý krok, data mining, může být ovlivněn uživatelem nebo znalostní databází. V následujícím kroku, kdy dochází k identifikaci námi hledaných dat může nastat jejich prezentace a případné uložení do znalostní databáze [16].



Obrázek 1: Vizualizace data miningu. Zdroj: [16]

2.6 Crawler

Web crawlers jsou programy, skripty, které fungují za účelem automatického prohledávání veškerého webového obsahu na internetu. Jsou také nazývány jako:

1. Bots
2. Robots

3. Spiders
4. Wanderers
5. Worm
6. User agents

Vyhledávací mechanismus se obvykle skládá ze tří částí, z nichž první je crawler, tedy robot, který má za úkol procházet kompletně celá webová sídla a „sbírat“ veškerý obsah nacházející se na nich. Robot prochází kořenový web a všechny jeho podweby – weby, na které je odkazováno. Všechna posbíraná data jsou následně vhodně indexována dalším nástrojem, aby byl zajištěn jejich budoucí přístup a efektivní, rychlé vyhledávání. Takto zpracovaná data jsou ukládána do rozsáhlých databázových struktur. Třetí částí je možnost provádět dotazové vyhledávaní z databázových struktur [23], [24].

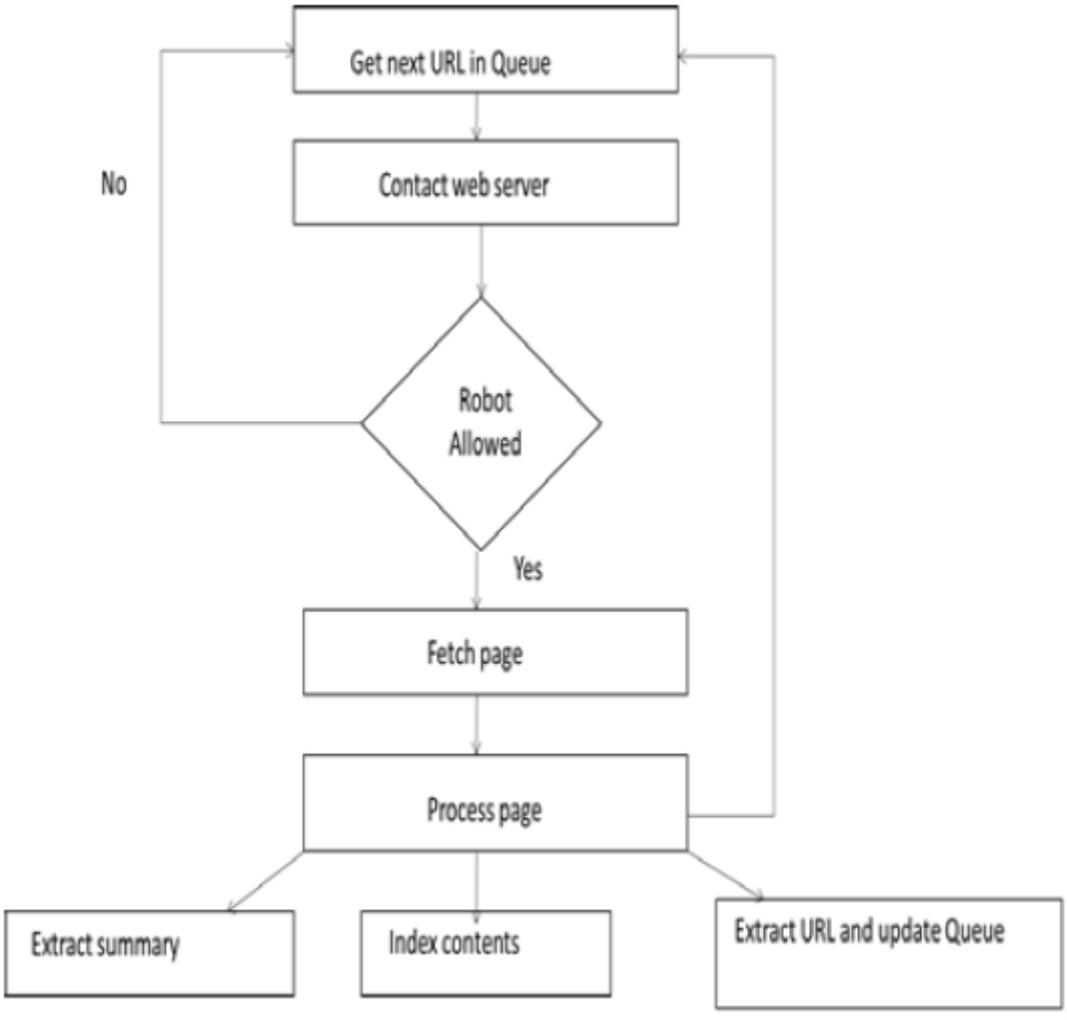
2.7 Proces samotného webcrawlingu

Crawler je obecně inicializován pomocí vkládání vstupních dat nejčastěji ve formátu pole obsahující seznam URL adres. Každá jednotlivá adresa je ohodnocena na základě požadavků administrátora, z čehož jí je přiděleno pořadí ve frontě, ve kterém bude prohledána robotem.
Z výše zmíněné fronty je brána adresa za adresou, kde je pomocí speciálních metod robota stažen její obsah. Následně jsou vyfiltrovány všechny nové URL adresy nacházející se na právě stažené webové stránce a tento nový seznam je přidán do již prohledávaného seznamu, tzn. je rozšířen o položky nové. Tento proces je opakován do doby, než je seznam URL adres prázdný [1], [24].

Detailní proces by měl dodržovat:

1. Vyhledávací mechanismus následuje adresy v zásobníku za účelem získání dalších webových adres neboli podstránek, na které odkazuje předchozí stránka. Tato hlavní stránka je označována jako stránka první nebo klíčová.
2. Pokud začneme prohledávání obsahu na stránce, která neobsahuje žádnou následující stránku a ani nikam neodkazuje, náš vyhledávací mechanismus uvěří, že veškerý internet má pouze 1 nebo 2 stránky.

3. V případě, že chceme mít správně funkční a efektivní web crawler, je nesmírně důležité, aby měl přístup k nespočtu indexovaných webových stránek, proto je nutné využít jako startovní stránku stránku s bohatým obsahem na navazující weby, respektive weby, které jsou relevantní ke startovní stránce. Např. webové sídlo *google.com* je vhodným kandidátem na start.
4. Špatně zvolená startovací stránka, tedy stránka bez žádných podstránek či odkazů k dalším webům, vyústí ve skončení algoritmu na startovací stránce, takže nedojde k prohledání žádných dalších webových sídel. Proto vhodnými vstupními daty může být obrovský seznam URL adres [23].



Obrázek 2: Vizualizace procesu crawlingu. Zdroj: [23]

Stručný popis:

1. Zkontroluje pořadí URL adres v seznamu a vybere jednu.
2. Zkontroluje, zdali stránka povoluje stažení webovým robotem (*robots.txt*).
3. Stáhne veškerý obsah prohledávané stránky.
4. Vyhledá všechny odkazy odkazující mimo konkrétní stránku a přidá je do seznamu adres určeného k prohledání.
5. Vyhledá všechna slova, zajistí indexování a uloží je do databázové struktury společně s prohledávanou stránkou. Uloží také pořadí slov, aby byla zajištěna možnost pozdějšího vyhledávání celých frázích, ne pouze klíčových slov.
6. Zároveň může, ale i nemusí probíhat filtrování různého nežádaného obsahu.
7. Náhled stránky je uložen spolu s posledním datem prohledávání, aby systém věděl, kdy se má vrátit kvůli aktualizaci obsahu v budoucnu [23], [2].

2.8 Scraping vs. crawling

V závislosti na obsahu a struktuře webové stránky musíme vytvořit web scraper nebo web crawler, ale v čem je vlastně rozdíl?

Web scraper je obvykle vytvořen za účelem procházení webových stránek a získávání dat. Je vytvořen k přistupování ke konkrétním stránkám, kde je v případě jejich změny či změny lokace vyhledávaných dat nebo jejich struktury nutná i změna kódu scrapera.

V kontrastu s výše uvedeným scraperem je web crawler obvykle vytvářen mnohem komplexněji. Má za úkol cílit na webová sídla, všechny jejich jednotlivé stránky, potažmo celý webový server domény. Crawler může sbírat více specifické informace z několika různých zdrojů či stránek a následovat další webové odkazy zpřístupněné z právě prohledaných stránek [2].

3 Existující implementace

V následující kapitole se seznámíme se strukturou HTML a stromovou strukturou DOM a XPath umožňující provádění selekce na webových stránkách. Dále se práce zabývá způsobem získávání dat z webových stránek za pomoci základních knihoven jazyka Python a frameworku Scrapy. V poslední části této kapitoly jsou předvedeny praktické ukázky využití scrapingu a crawlingu v praxi.

3.1 Proces procházení stránek

Zprvu je potřeba nastínit proces procházení webových stránek. Ať už skript, nebo přímo uživatel musí napsat URL adresu v prohlížeči, potažmo provést kliknutí na odkaz, dokud není webová stránka plně zobrazena. Z pohledu této práce se proces dělí na čtyři níže zmíněné sekce.

1. URL adresa je napsána v prohlížeči. První část URL adresy (doménové jméno např. *example.com*) je použito pro vyhledání cílového serveru a společně s cookies a dalšími požadavky je zaslán formou POST nebo GET dotazu.
2. Server odpovídá zasláním HTML stránky prohlížeči. Důležité je zmínit, že server může odpovídat i jinými formáty, jako jsou např. XML nebo JSON, ale z pohledu práce se zaměříme převážně na HTML formát.
3. HTML dokument je následně transformován do stromové struktury DOM, která je součástí každého novějšího prohlížeče.
4. Poté je vykreslena webová stránka na základě definovaných vizuálních sekcí do podoby, kterou známe z běžných webových prohlížečů, a zobrazena na monitoru.

3.2 Parsování HTML

Webový server čekajíc na dotazy ze strany prohlížečů po obdržení žádosti zanalyzuje dotaz na základě URL adresy např. ***example.com/index.html***, kdy zašle zpět prohlížeči dokument ve formátu HTML s názvem index. Pro běžného uživatele je nejjednodušší zobrazit daný soubor přímo v prohlížeči. V každém prohlížeči se postup zobrazení může lišit, avšak pro naši ukázku využijeme webový prohlížeč Google

Chrome, kde stačí pouze kliknout pravým tlačítkem myši do okna prohlížeče a zvolit „Zobrazit zdrojový kód stránky“, nebo použít klávesovou zkratku Ctrl + U.

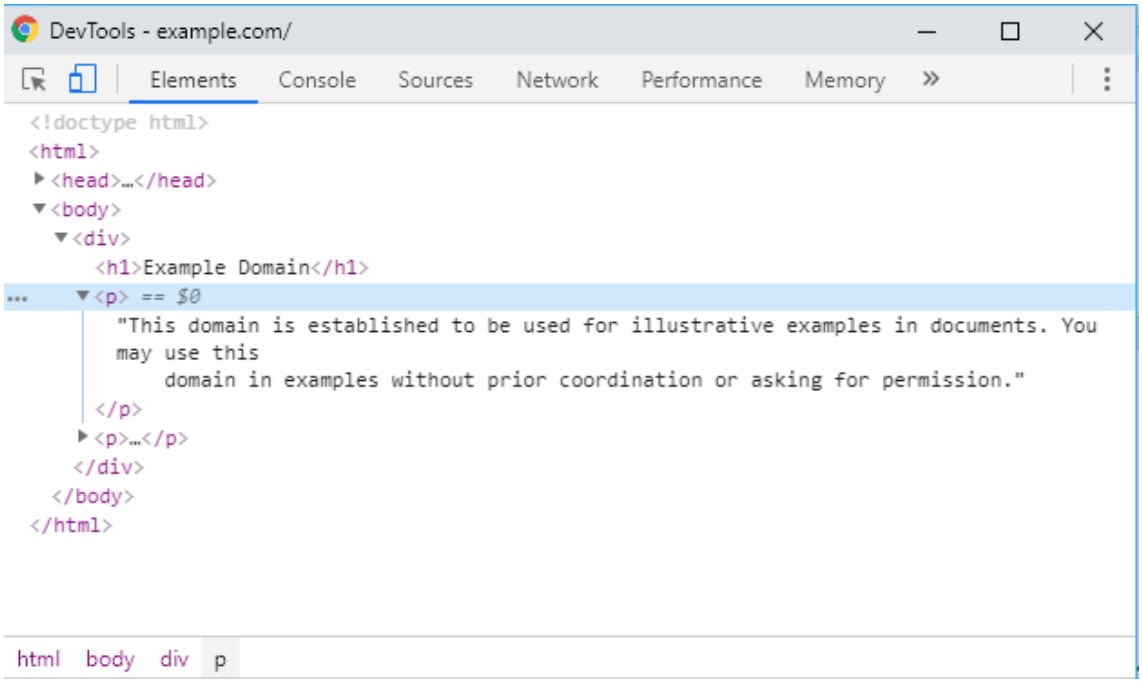
Otevře se nám zdrojový kód námi vybrané stránky ve formátu HTML.

3.3 DOM

Document Object Model neboli objektový model dokumentu je objektově orientovaná a jazykově nezávislá reprezentace XML nebo HTML dokumentu fungující na kterékoli platformě, kterou podporuje většina prohlížečů. DOM umožňuje programům a skriptům dynamický přístup a aktualizaci obsahu, struktury a stylu dokumentů. Dokument může být dále zpracován a výsledky mohou být začleněny zpět na prezentovanou stránku [14].

K zobrazení stromové datové struktury webové stránky v prohlížeči Chrome klikneme pravým tlačítkem myši a zvolíme funkci „Prozkoumat“. V této fázi vidíme samostatnou vlastní stromovou reprezentaci HTML dokumentu zobrazenou prohlížečem Google Chrome, která se zdá býti stejná klasickému HTML dokumentu, ale není tomu tak.

Zobrazí se nám velice mocný nástroj samotného prohlížeče, s kterým lze snadno a jednoduše procházet každý element a sledovat jeho strukturu, vzájemné vnoření jednotlivých elementů, provádět vizuální změny dokumentu v reálném čase apod.

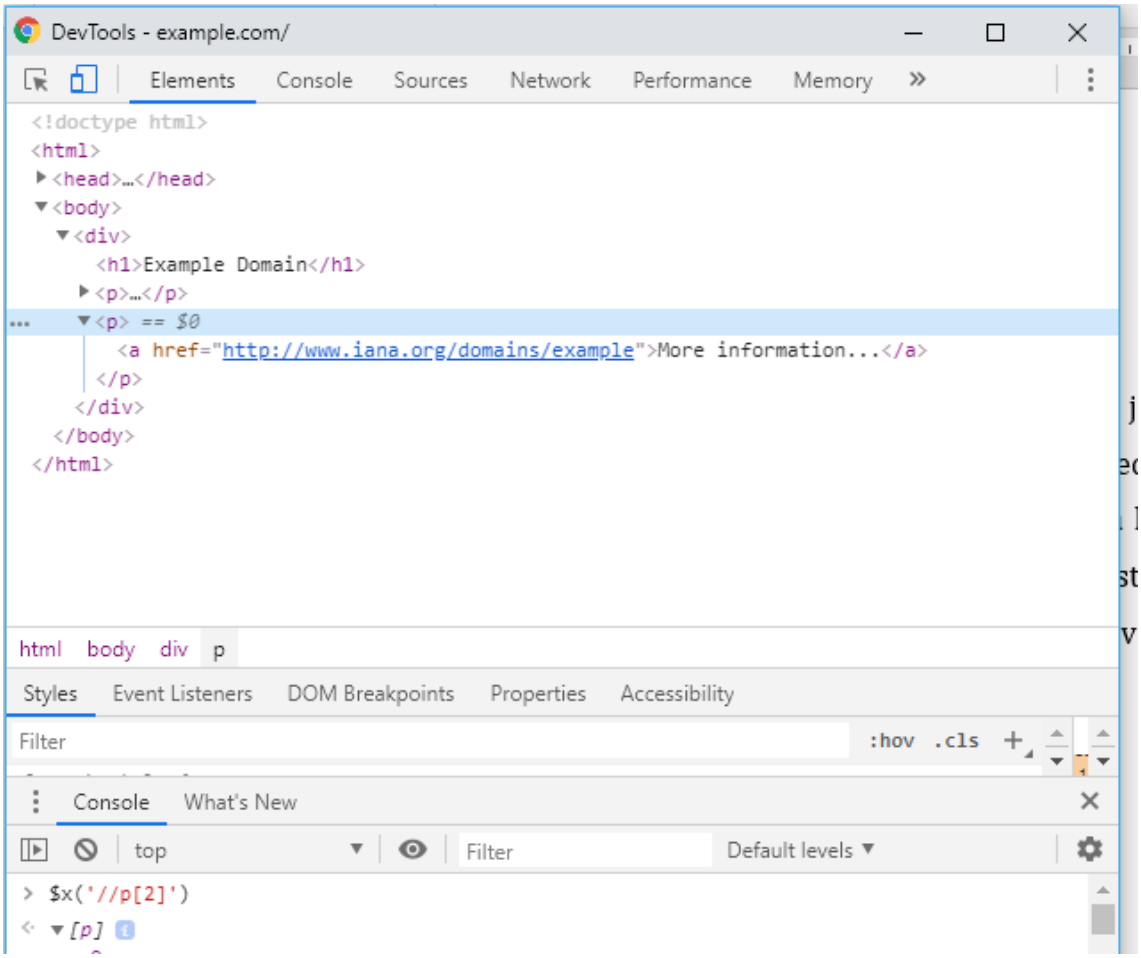


Obrázek 3: DevTools Google Chrome

Ve výše uvedeném obrázku vidíme vnořený element <p> v elementu , na který se budeme odkazovat v dalších kapitolách.

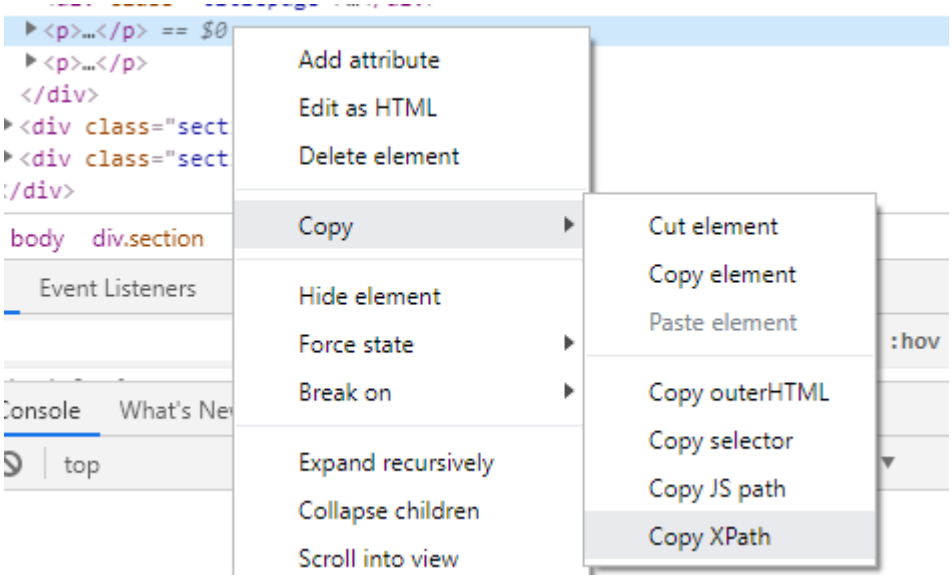
3.4 XPATH

Jazyk XPath slouží pro selekci a extrakci dat jako jsou HTML elementy, atributy a samotný text, který je nedílnou součástí k prohledávání webové stránky. Jeho použití je velice snadné díky prohlížeči Chrome, na kterém si provedeme ukázku. Jednoduše stačí na požadované stránce otevřít „Nástroje pro vývojáře“ a dostat se do konzole, kde je již možné pomocí předem definované funkce **$x** zavolat jakýkoli element, případně provádět rovnou selekci dat. Například níže zmíněný dotaz nám vrátí druhý atribut <p> v pořadí na stránce.



Obrázek 4: DevTools Google Chrome

Dalším způsobem využití je excelentní možnost zkopírování XPath cesty ke konkrétnímu elementu pomocí DOM zobrazení v nástrojích pro vývojáře. Není jednodušší cesta, než pravým kliknutím myši přímo na webové stránce zvolit „Prozkoumat“, kde se nám otevře stromová struktura zobrazující přímou selekci námi požadovaného elementu, kde lze kliknutím pravým tlačítkem myši a následně vybráním zkopírování XPath docílit zkopírování řetězce.



Obrázek 5: DevTools Google Chrome

Kupříkladu výše vložený obrázek navrátí XPath řetězec ve formátu

```
/html/body/div[3]/div[2]/p[1],
```

což doslova znamená selekci elementu v těle stránky, jejím třetím divu v pořadí, jeho druhém divu v pořadí a prvním atributu <p>.

3.5 V jazyku Python

Kapitola pojednává o metodách dolování dat z webových stránek pomocí knihoven Request, Urllib a BeautifulSoup. Dále předvádí komplexnější řešení skrze framework Scrapy. Závěrem se zde nachází demonstrace využití webových robotů na internetu.

3.5.1 Knihovna Request

Získání HTML dokumentu pomocí knihovny Request je velice jednoduché. Po snadné instalaci, kterou se tato práce nezabývá, následuje ještě jednodušší použití. Na níže uvedeném příkladu dochází k prvotní importaci knihovny a jejímu následnému zavolání s metodou GET (případně POST) s parametrem URL adresy. Metoda nám navrátí HTML dokument, který je možné následně zpracovat.

```
Import requests
r = requests.get('https://api.github.com/events')
r.text
```

Získávání a následná analýza HTML dokumentu je stavebním kamenem celého crawlingu. Nyní by stačilo pouze provést selekci dalších URL adres na stránce a následné předání k dalšímu zpracování, viz kapitola 4.3.1.

Autoři knihovny Request přišli s její odnoží, která se zabývá přímo parsováním HTML dokumentů a jejich jednoduchým zpracování pomocí selekce skrze CSS a XPath selektory. Pro náš příklad stačí uvést část pro extrakci veškerých URL adres na webové stránce, které lze docílit pouze příkazem *links()*, jenž navrací URL adresy vzhledem ke kořenovému adresáři webu, nebo případně *absolute_links()*, který navrací absolutní adresy vzhledem k webovému sídlu.

```
r.html.links
{'//docs.python.org/3/tutorial/', '/about/apps/'}
r.html.absolute_links
{'https://github.com/python/pythondotorg/issues',
 'https://docs.python.org/3/tutorial/'}
```

3.5.2 Urllib

Tato python knihovna se skládá z několika modulů, mezi nimiž je například modul Request, který definuje funkce a třídy, jež jsou nápomocné při otevírání URL adres (nejčastěji na protokolu HTTP a HTTPS), případné autentizaci skrze webovou žádost, přesměrování, cookies apod. [12] Dalším modulem, který stojí za zmínku, je modul Parse, který definuje základní rozložení řetězce URL adresy do jednotlivých částí, jako jsou například síťové umístění, relativní a absolutní cesta, adresní schéma apod. Modul dále umožňuje kombinaci jednotlivých extrahovaných dílčích částí zpět do tvaru řetězce URL adresy a změnu relativní adresy na absolutní na základě základní URL adresy [13].

3.5.2.1 Crawler v knihovně Urllib

V následující podkapitole je předveden funkční web crawlering pouze za pomoci základní python knihovny Urllib, která slouží k vytváření spojení a prací s protokolem, a modulu HTMLParser k parsování HTML obsahu a regulárními výrazy, které jsou nápomocné v selekci mezi jednotlivými řetězci a vytváření podmínek neboli hranic k prohledávání obsahu.

Cílem této demonstrace je vytvoření skriptu, který je schopen začít prohledávat webové sídlo a sbírat všechny unikátní URL adresy, které se nacházejí ve všech HTML elementech <a> na jednotlivých navštívených webových stránkách. Poté dokáže výstupní data rovnou ukládat do souboru, který umožní budoucí potenciální zpracování a opětovné použití.

Prvním klíčovým prvkem je startovací URL adresa, odkud chceme, aby se robot vydal k prohledávání dalších navazujících adres. Je vhodné si vytvořit jednu datovou strukturu, například datové pole, pro sběr URL adres sloužících k prvotnímu prohledání a druhou, kde budeme uchovávat unikátní URL adresy, které skript shledá jako relevantní. Takto vytvořené datové pole, dále pouze zásobník, naplníme startovací stránkou a budeme prohledávat, dokud nebude zcela prázdné. V cyklu procházení zásobníku budeme testovat každou hodnotu, zdali se již nenachází v datovém poli s unikátními adresami. Pokud se hodnota s URL adresou ze zásobníku již nachází v poli s unikátními adresami, dojde k jejímu odebrání ze zásobníku, a pokud tomu tak není, bude do něj přidána.

Dalším krokem je předání URL adresy ze zásobníku třídě zajišťující extrakci odkazů z dané webové stránky předávané parametrem. Metoda načte parametr a pomocí knihovny Urllib otevře spojení s webovou stránkou, obratem dostává odpověď ve formátu HTML stránky, kterou stáhne a dekóduje pomocí UTF-8. Veškerá stažená a dekódovaná data předává modulu HTMLParser, který se postará o další zpracování. Tento modul má metodu s názvem **handle_starttag**, kde pomocí jednoduché podmínky dokážeme testovat všechny HTML elementy <a> nacházející se na stažené webové stránce. Na takto objevených elementech je testována existence atributu href, v němž je uložena cesta. Tato hodnota se pomocí metody **urljoin** spojí se základní URL adresou, která byla získána z hlavičky HTML dokumentu a vytvoří tak absolutní URL adresu, již uloží do datové struktury a která bude po zpracování celé webové stránky navrácena s obsahem všech objevených URL adres v absolutní podobě.

Veškeré URL adresy jsou poté jednotlivě testovány, zdali se již nenachází v zásobníku, zdali nejsou zrovna URL adresou, která je aktuálně analyzována, a lze-li na ně v této fázi aplikovat pravidla před vložením do zásobníku. V tomto případě budeme testovat pouze

lokálnost, zdali se testovaná adresa nenachází mimo doménu startovní stránky, a začleníme pravidlo, které automaticky odfiltruje veškeré odkazy spojené se sociální sítí Facebook. Adresa je ještě před uložením do zásobníku testována pomocí regulárních výrazů, například pomocí této části python kódu, který stanovuje podmínku, že se v testovaném řetězci musí nacházet řetězec v podobě startovací stránky.

```
regex = r"^(.*?(\b" + str(rootUrl) + r"\b)[^$]*)$"
```

Všechny ostatní znaky kolem, ať už subdomény, nebo případně další cesty, jsou již irelevantní. Podmínka toho, že se testovaná adresa nachází na hranici testovací výchozí webové stránky, je proto splněna. Druhým pravidlem je odfiltrování veškerého obsahu spojeného se sociální sítí Facebook. Byla vytvořena jednoduchá podmínka ve tvaru:

```
regexFb = r"^((?!facebook).)*$"
```

Podle ní je testována URL adresa. Pokud první podmínka platí, tedy testovaná URL adresa se nachází na doméně výchozí testovací stránky, a zároveň je splněna druhá podmínka, kdy nedochází k tomu, že testovací URL adresa obsahuje řetězec „facebook“, je tato adresa přidána do zásobníku a určena k dalšímu testování.

Před koncem průchodu cyklu dojde k odstranění duplikací ze zásobníku, aby nedocházelo ke zbytečné opakující se analýze webové stránky. Skript končí v případě, že zásobník je kompletně vyprázdněn a všechny unikátní URL adresy jsou zapsané v datové struktuře a zároveň uloženy v souboru, který umožňuje jejich další zpracování.

Výše zmíněné řešení je funkční za účelem extrakce webových odkazů z celého webového sídla, ale zároveň je do jisté míry nedostatečné. Stanovení jednotlivých filtračních pravidel se jeví dosti složité a komplexně nepraktické s využitím regulárních výrazů. Skript nevyužívá více vláken, není odladěn a neobsahuje žádnou ochranu před případným IP zákazem či jiným postihem ze strany testovaného webového sídla. Jeho rychlost rozhodně nestačí na rozsáhlá sídla, která čítají i několik milionů unikátních odkazů a cest.

3.5.3 BeautifulSoup

Tato kapitola pojednává o populární python knihovně BeautifulSoup, která slouží převážně k extrakci obsahu webových stránek a snadné navigaci skrze HTML obsah bez nutnosti složitého parsování a používání regulárních výrazů. Tato knihovna nabízí

snadnou práci s HTML DOM elementy a vyhledáváním v obsahu a dokonce také úpravu HTML dokumentů.

Vhodnou demonstrací je níže zmíněná python metoda, která pomocí využití knihovny BeautifulSoup dokáže načíst zadanou URL adresu předávanou v parametru, zparsovat její obsah a uložit všechny nalezené elementy <a> do datové struktury, kterou následně vrací [3].

```
def getLinks(url):
    html_page = urllib2.urlopen(url)
    soup = BeautifulSoup(html_page)
    links = []

    for link in soup.findAll('a', attrs={'href':
re.compile("^http://")}):
        links.append(link.get('href'))

    return links
```

3.5.4 Framework Scrapy

Scrapy je robustní webový framework sloužící k získávaní dat z rozsáhlých webových stránek. Za zmínění stojí event-based (založena na událostech) architektura, která umožňuje stupňovité operace, jako je čištění, formátování, obohacování dat nebo jejich ukládání například v databázi, přičemž nezaznamenává žádnou degradaci výkonu. Technicky řečeno, touto architekturou nám Scrapy umožňuje zbavit se přílišné latence a zaručí hladký průběh operací během běžících otevřených tisíců spojení.
V extrémním případě, kdy je za cíl stanovena extrakce všech prvků seznamu čítajícího i několik set na stránku, je Scrapy schopen zvládnout 16 dotazů na webový server paralelně. Pokud by jednomu dotazu trvalo dokončení průměrně jednu sekundu, je schopen Scrapy stahovat 16 stránek za vteřinu, což v případě sumarizovaných 100 prvků na stránku činí 1 600 prvků za vteřinu. Pokud bychom chtěli uložit veškerá získaná data na cloudové uložiště, které by v nejhorším případě mělo odezvu až tři vteřiny, a rádi

bychom dodrželi rychlost 16 stránek za vteřinu, bylo by nutné paralelně generovat 4 800 (1 600 x 3) zápisů do databáze. V klasických multivláknových aplikacích toto představuje 4 800 vláken, která by byla velice náročná na systém a taktéž by tvořila zcela jistě nevýkonné řešení. Ve světě Scrapy je 4 800 souběžných dotazů zcela běžnou záležitostí, pokud tomu dovolí výkonnost stroje. Kromě toho Scrapy sleduje velice pečlivě paměťové požadavky, které potřebuje pro své zápisy do vícevláknové aplikace, kde každé vlákno alokuje signifikantní množství paměti.

Ve zkratce, pomalé nebo nepředvídatelné webové servery, databáze, nebo API třetích stran nebudou mít devastující efekt na výkon Scrapy skriptu právě kvůli možnosti běhu několika dotazů zároveň a řízení všech interních operací z jednoho vlákna. Toto snižuje hardwarové nároky pro běh scrapy aplikací a není nutná synchronizace kódu, jako tomu je u ostatních vícevláknových aplikací.

Několik výhod používání Scrapy frameworku:

1. Scrapy rozumí poškozenému HTML kódu, dokáže ho přesto zpracovat i přes možné syntaktické chyby.
2. Nativní podpora BeautifulSoup a lxml.
3. Využití XPath, CSS selektorů.
4. Dobře organizovaný kód.

3.6 Další využití crawleru

Tato kapitola pojednává o dalším využití webových crawlerů pro usnadnění vykonávání opakujících se činností na internetu. Crawling a scraping nabízí opravdu obrovskou škálu využití, od sběru dat z několika rozdílných zdrojů, přes jejich snadné zpracování a jejich redistribučnost, po automatizaci dílčích sekvenčních kroků nebo různých opakujících se úkonů. Všude, kde není na cílovém webovém sídle nabízena možnost API (množství funkcí a procedur, které nabízí vývojářům přístup k datům [22]) nebo pouze v omezené míře, přichází na řadu využití scraperů. Níže zmíněné příklady jsou reálnými situacemi využití autorem práce v praxi. Všechna citlivá data jsou nahrazena náhodnými řetězci z důvodu zachování anonymity.

Prvním příkladem, který stojí za zmínění, je skript sloužící k filtraci a sběru e-mailových adres přístupných na webovém portále po přihlášení uživatelem. Pouze za použití knihovny Request (viz kapitola 3.5.1), která zajišťuje posílání žádostí na cílený webový server a prochází cyklem přesně daný počet různých webových stránek, z kterých pomocí regulárních výrazů vyhledává řetězce ve tvaru e-mailové adresy, a uloží je do datového pole pro jejich další využití.

```
session = requests.Session()
headers = {'user-agent': 'Mozilla/5.0 (Windows NT 10.0; 
WOW64) AppleWebKit/537.36 (KHTML, like Gecko) 
Chrome/52.0.2743.116 Safari/537.36'} # login
payload = {'email': 'email@email.cz', 'password': 'heslo', 
'redirectTo': '', 'login': 'P%C5%99ihl%C3%A1sit', '_do': 
'loginForm-submit'}
r = session.post('https://www.domena.cz/admin/login-admin/', 
data=payload, headers=headers)

for x in range(1,5318):
    r = session.get('https://www.domena.cz/admin/?item _id=' 
+str(x))
    tmp = re.findall('[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-
Z0-9-.]+', r.text)
    tmp = list(dict.fromkeys(tmp))
    for email in tmp:
        print(str(x) + ': ' + email)
        f = open("emailHarvested.txt", "a")
        f.write(str(email))
        f.write('\n')
        f.close()
        emails.append(str(email))
print (emails)
```

Druhým příkladem je skript pracující s nejmenovaným aukčním webovým portálem, který srovnává nejnovější nabídky sestupně vzhledem k datu jejich přidání do systému. I

přesto, že cílový web využívá CSRF[5] tokenu [20], [21], je zde ochrana nedostatečná a zpracování přihlášení uživatele skriptem velice snadné. Při překonání této ochrany nedošlo k porušení žádných nařízení, nebylo jednáno protiprávně a ani nedošlo ke způsobení žádných škod na straně serveru provozovatele webového sídla. Tato ukázka pouze demonstruje způsob, kterým je možné překonat pomocí skriptu úmyslně postavenou bariéru. Platí zde fakt, že s nesprávnými úmysly může snadno dojít k překročení hranic a pohybu na hraně, nebo dokonce za hranou zákona.

Poté je podle předem přesně daných šablon, které obsahují informace o budoucí nabídce v aukci, odeslána žádost na cílový portál a nabídka je vytvořena. Po několika náhodně zvolených vteřinách z časového intervalu je nabídka smazána pomocí další žádosti směrem k aukčnímu serveru a obratem znovu vytvořena pro zajištění zobrazení předních míst v aukčním portálu. V ukázkovém kódu můžeme vidět přihlášení uživatele a poskládání žádosti o vytvoření nabídky.

```
#   login
        r = session.get('https://domena.com/login',
headers=headers)
        regex = re.compile("csrf_token' value='([a-zA-Z0-
9]*)'")
        m = regex.search(r.text)
        csrf = m.group(1)
        payload = {'csrf_token': csrf, 'email':
'email@email.cz', 'password': 'heslo', 'submit': 'Login'} ##
        r =
session.post('https://domena.com/functions/login.php',
data=payload, headers=headers)

        #   Add trades
        for i in range(len(ownerItems)):
            r =
session.get('https://domena.com/trading/new',
headers=headers)
            m = regex.search(r.text)
            csrf = m.group(1) ##
            payload = {'csrf_token': csrf,
                'random': 0,
                'random ': 0,
                'amount': 1,
                'search': '',
```

[5] CSRF token je náhodně vygenerovaný řetězec, který se pro každý jednotlivý formulář liší. Při odesílání je kontrolována jeho platnost, která zajišťuje ochranu proti podvrhům odesílaných žádostí třetích stran ke zpracování cílovému serveru [21].

```
                'note':     '',
                'platform': 1,
                'btnSubmit': 'Add+trade',
                'ownerItems': ownerItems[i],
                'tradeItems': tradeItems[i]
            }
            while True:
                add =
session.post('https://domena.com/random/addTrade.php',
data=payload, headers=headers)
                msg = add.text.split('rlg-site-
popup__text">')[1].split("<")[0].replace("\>", "")
                if msg == 'It looks like you\'ve already
listed this trade' or msg == 'Trade has been added
successfully' :
                    break
                print('Waiting...')
                sleep = random.randrange(3, 7)
                time.sleep(sleep)
            trade =
add.text.split('domena.com/trade')[1].split("'")[0].replace(
"/", "")

            trades.append(trade) #add to list
```

Třetím příkladem je ukázka využití nástroje Selenium, který primárně slouží k testování webových aplikací [17]. V našem případě nám ale poslouží k jednoduššímu přístupu k webovému sídlu. Integrace Selenia s jazykem Python zaručuje efektivnější a jednodušší přístup k webovým stránkám a jejich obsahu. Přístup může být ještě výrazněji zjednodušen pomocí tzv. headless prohlížeče PhantomJS [18], který umožňuje ovládání webového sídla pomocí skriptů běžících na straně serveru. V našem případě je ale použit pouze základní nástroj pro ovládání prohlížeče Chrome pomocí skriptovacích příkazů s názvem webdriver [19], který taktéž slouží k testování webových aplikací. Cílem skriptu je zachycení akční nabídky nejmenovaného internetového e-shopu, který v určitou hodinu spustil prodej extrémně zlevněné položky v omezeném počtu kusů. Skript v cyklu opakovaně načítá webovou stránku a snaží se vyhledat nákupní tlačítko, které do spuštění akce v konkrétní hodinu zůstalo skryté. V případě, kdy by došlo ke zjištění přítomnosti daného tlačítka přímo v kódu, proběhl by okamžitý proklik. V opačném případě by došlo k obnovení stránky.

```
#!/usr/bin/env python
from selenium import webdriver
import time
```

```
chrome_options = webdriver.ChromeOptions()

chrome = webdriver.Chrome(chrome_options=chrome_options)
chrome.get("https://www.eshop.cz/produkt")

while True:
    try:
        chrome.refresh()
        chrome.find_element_by_xpath("/html/body[@class='
desktop
chrome']/div[@id='wrapper']/div[@id='main']/div[@id='main-
content']/div[@id='content']/div[@id='product-
detail']/div[@class='pd-wrap']/div[@class='pd-
info']/div[@class='pd-price-delivery']/div[@id='product-
price-and-delivery-
section']/div[@class='left']/button[@class='btn btn-
buy']/span[@class='btn-inner ico-basket']").click()
        break
    except:
        print ("Waiting...")
```

4 Realizace robota pro získání sitemapy

V metodologické části práce je popsáno, jakým způsobem byla řešena analýza webových sídel, realizace crawlera a jeho výstupních dat tvořících sitemapu. Výsledný program byl testován na několika webových sídlech, ale pro zjednodušení bude prezentován pouze na příkladu domény *http://toscrape.com/*. Dále byly rozebrány a představeny další možnosti využití sitemap pro různé skupiny uživatelů.

4.1 Analýza webového sídla a stanovení požadavků

Za účelem vytvoření kompletní sitemapy nebylo možné načítat soubor *robots.txt* na straně webových sídel z důvodu případného znehodnocení kompletních výsledků odpovídajících realitě. Pokud by cíl práce byl jiný, bylo by nanejvýš vhodné začít tímto krokem. Načtení souboru by v našem případě mohlo vést k zablokování crawlera hned v jeho počátcích nebo jeho přesměrování do tzv. pasti[6] a jeho následné blokaci skrze IP adresu hostujícího serveru. Dále bylo nutné zvolit znatelně pomalejší tempo scrapingu i přesto, že samotný framework dovoluje práci v několika vláknech a paralelní zpracování několika žádostí za sekundu (ve výchozím nastavení 16 dotazů za vteřinu, viz kapitola 3.5.4) z důvodu, že by se prohledávání webového sídla mohlo jevit jako DDoS útok[7], případně jakýkoli jiný útok způsobující přehlcení webového provozu, a mohlo by tak vést k blokaci[8] IP adresy.

Proto byl předmětem testování hlavně cílen speciální web sloužící k těmto účelům, avšak několik jiných webů včetně výsledného reportu bude přiloženo k práci.

[6] Webovou pastí pro scrapera je například procházení stránkování kalendáře, který je neustále generován javascriptem, což obnáší téměř nekonečný rozsah prohledávání, a brání tak kompletní analýze stránky [2]. Jednou z prevencí je nastavení maximální hloubky prohledávání, kterou Scrapy sám o sobě nabízí již v základním nastavení, a zamezit tak nekonečné smyčce v oblasti prohledávání časového horizontu v kalendáři.

[7] Každý paralelní dotaz na webový server se ve zvýšené míře může jevit jako cílený útok za účelem zvýšení webového trafficu serveru, který v krajních případech vede až ke kolapsu hostujícího serveru [1], [2].

[8] Jedna z nejčastějších reakcí serveru na zvýšený počet dotazů z jednoho zdroje. Z bezpečnostních důvodů dojde k blokaci hostovy IP adresy a předejde tak přetížení serveru.

4.2 Scraping a data mining

K realizaci robota byl využit robustní python framework Scrapy (viz kapitola 3.5.4), který právě kvůli své architektuře a výborné práci se selektory nabízí téměř neomezené možnosti procházení webových stránek.

Hlavním cílem byla extrakce všech relevantních URL adres nacházejících se na webové stránce. Relevantní adresy vzhledem k cíli vytvoření sitemapy jsou tedy všechny, které směřují na další webové stránky sídla a obsahují textový obsah s informační hodnotou (nejčastěji ve formátu HTML), nejsou to tedy obrázky, různé soubory ukládající data apod.

Prvním nutným krokem bylo stanovit hranice, kam až se crawler může vydat. Náš crawler je omezen pouze kořenovou doménou, což znamená, že všechny odkazy směřující mimo webové sídlo jsou automaticky zahazovány a nejsou prohledávány. Zároveň je velice důležité vybrat vhodnou startovní stránku (případně stránky), pokud by došlo ke špatnému výběru a crawler by neměl kam pokračovat, program by se ukončil vzhledem k cíli práce s nedostatečným výsledkem. V našem případě došlo ke zvolení hlavní domény **toscrape.com.** Dalším vhodným parametrem je pojmenování robota, které se definuje jednoduše jako hodnota proměnné „name“. Pojmenování crawlera se promítne do hlavičky **User-agent,** na jejímž základě dokáže cílový web zjistit, který robot se ho chystá prohledávat (za předpokladu, že dojde k načtení souboru *robots.txt*), a vyhodnotit jeho omezení, případně povolení ze strany navštíveného webu.

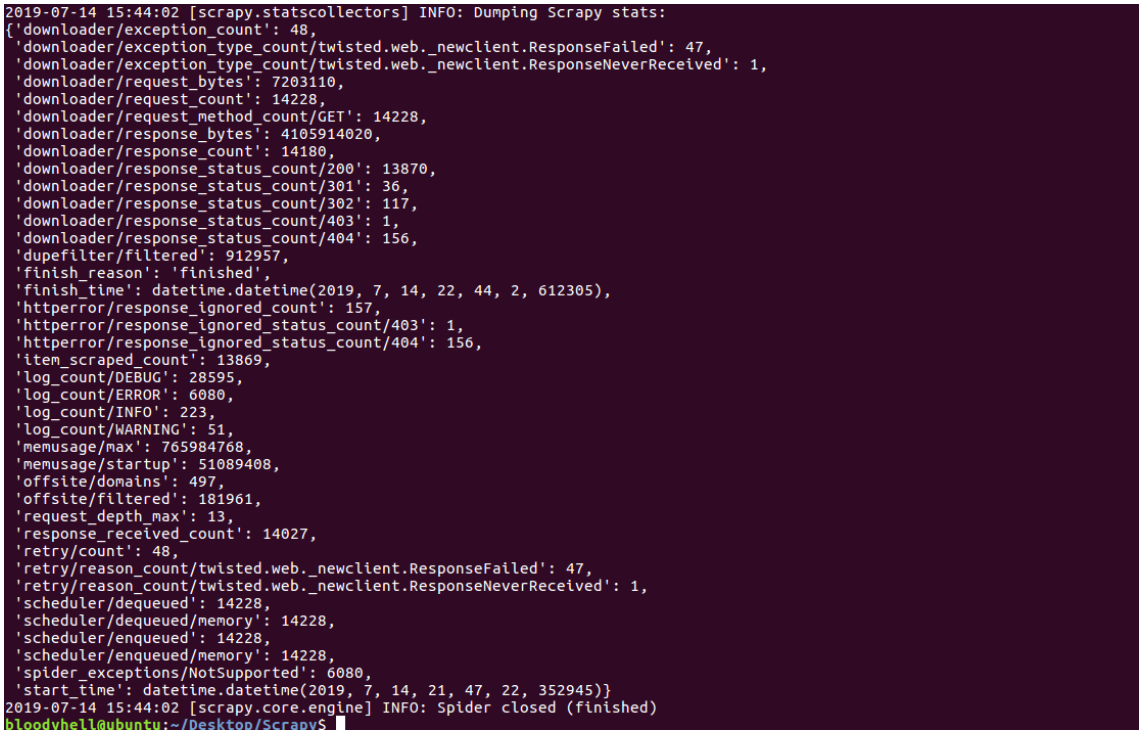
Scrapy umožňuje vytvořit nespočet pravidel a různých omezení. Jedním z nich je přesné určení, kudy se po webu bude robot vydávat. Lze kategorizovat cesty, směry a pravidla[9], avšak pro cíl práce je nutné crawlera pustit všude po webovém sídle, tudíž nemohlo dojít k žádnému limitování ze strany vývojáře.

[9] Jedná se o určení omezení v prohledávání ve webovém sídle. Například za předpokladu, že cílený web kategorizuje zboží ve tvaru URL adresy domena.cz/zbozi/ a cílem robota je prohledávání pouze této části webu, lze tedy aplikovat pravidlo, které omezí cestu pouze ve tvaru domena.cz/zbozi/item, kde „item“ zastupuje jakýkoliv další řetězec.

Po spuštění robota a jeho načtení výchozí vstupní stránky začíná její prohledávání do hloubky. Crawler odešle žádost na webový server a dostává odpověď ve formátu HTML stránky, kde byly aplikovány CSS selektory ve tvaru **a::attr(href)**, které jsou schopny vyhledat veškeré HTML elementy **a** s atributem **href**. Takto nalezené odkazy jsou testovány, zdali jsou již uloženy v unikátním seznamu, a pokud nejsou, jsou do něj následně uloženy a zároveň posílány znovu do datové fronty s URL adresami sloužící k dalšímu prohledávání následujících webových stránek. Po každé analýze nové webové stránky dochází k redukci duplicitních hodnot v zásobníku (frontě) URL adres a proces se opakuje. V případě, že je zásobník s URL adresami prázdný, robot prohledal veškerý dostupný obsah a sám skončil.

Scrapy automaticky generuje report po skončení procesu crawlingu, ze kterého lze mimo jiné vyčíst celkový čas běhu procesu, počet navštívených webových stránek, počet stránek, které skončily nějakou chybou (301, 302, 403, 404)[10], nebo naopak jejichž odpověď byla úspěšná (200)[11], počet vyfiltrovaných duplicitních URL adres ze zásobníku a počet URL adres směrujících mimo daná pravidla scrapera. Celá implementace obsahující uvedené úryvky řešení je uvedena v elektronické příloze práce.

[10] Status code neboli stavový kód http protokolu, který je součástí hlavičky odpovědi serveru na klientský požadavek. Kód v rozmezí 3XX, kde X je nahrazeno číslem, značí přesměrování, kód v rozmezí 4XX potom chybu na straně klienta.
[11] Status code 200 značí standardní odpověď pro úspěšný http požadavek.



Obrázek 6: Ilustrace výstupu ze Scrapy

4.3 Vytvoření sitemapy

Scrapy nabízí možnost exportu výstupu z programu rovnou do souboru ve formátech CSV, JSON a XML. Pro účel práce byla zvolena možnost exportu ve formátu XML pro ukládání dvou typů dat. Prvním z nich je URL adresa prohledané webové stránky a druhým je zdrojová stránka, která předcházela námi cílené webové stránce, tzn. reference, odkud webový robot přišel. Takto posbíraná data lze nadále zpracovat námi požadovaným způsobem, např. vygenerováním sitemap, což je cílem této práce. Sitemapa (viz kapitoly 2.2 a 2.3) je nejčastěji soubor obsahující jednotlivé webové stránky ve formátu XML tvořící datovou strukturu v elementech <url> s vnořeným elementem <loc>, který přesně kopíruje webovou adresu. Takto zpracovaný výstup dat tvoří plnohodnotnou sitemapu.

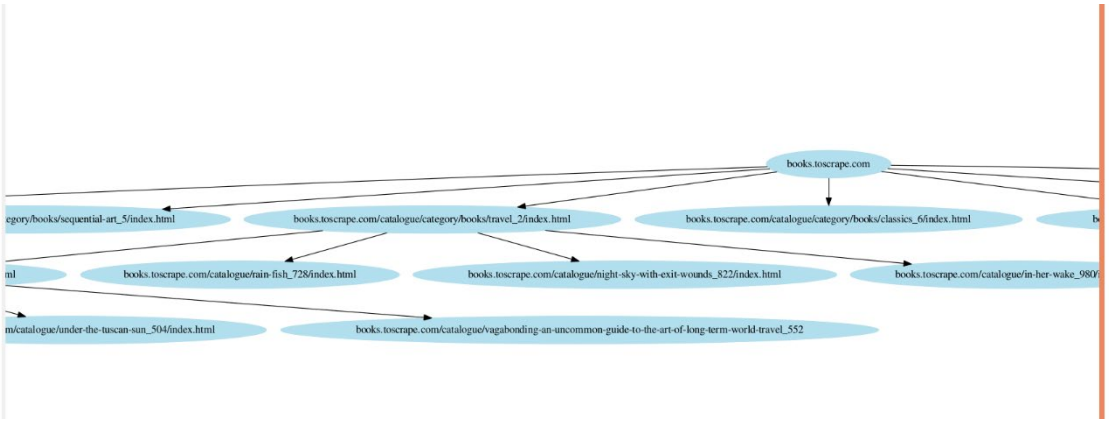
```
<url>
 <loc>https:// toscrape.com</loc>
</url>
```

4.3.1 Vizuální vykreslení sitemapy

Pro vizuální zobrazení výstupních dat byla zvolena python knihovna **Graphviz,** která umožňuje vhodně graficky ztvárnit hierarchii webového sídla pro účel zobrazení

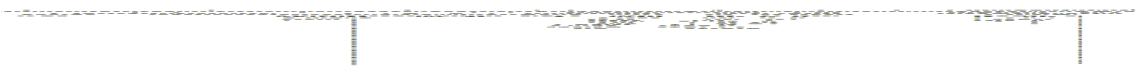
mapy webu [26]. Výstup dat byl zpracován dalším python skriptem, který parsoval výstupní data ve formátu JSON. Jednotlivé cílové URL adresy a jejich předchůdci byli graficky zaznamenáni jako „uzly“ s napojením na další odkazy tak, aby byla zachována hierarchie webového sídla. Zvolená vizualizace umožňuje grafické přiblížení hloubky jednotlivých webových cest a jejich propojení v sídle blíže čtenáři.

Na níže zobrazeném obrázku je vidět vizuální zobrazení části testovaného webového sídla. Výsledná data zpracovaná webovým robotem, který začínal v kořenovém adresáři webového sídla, jsou taktéž zobrazena ve formě uzlů, kde je nejvýše položena právě výchozí webová stránka.



Obrázek 7: Vizualizace výseku sitemapy

Webové sídlo z větší perspektivy může mít potom vizuální ztvárnění v následující podobě, jež se jeví pro čtenáře jako již ne tolik přehledná. Za účelem dosažení cíle této práce však bude zpracována pouze malá část.



Obrázek 8: Vizualizace celkové sitemapy

4.4 Další možnosti využití sitemap a crawlingu

Tato kapitola popisuje další možnosti využití sitemap pro cílené skupiny uživatelů. Dále pojednává o vlivu orientace ve webových sídlech z pohledu cílové skupiny a navrhuje její zefektivnění a zrychlení pomocí webových crawlerů a scraperů.

Autoři webových sídel nejčastěji sami určují přístupnost informací uživatelům, jsou schopni definovat proces filtrování a prohledávání vlastního webu, což může být do jisté míry nepřehledné, omezující, ba dokonce nepřijatelné, když proces vyhledávání na daném webu nedosahuje žádaných výsledků z pozice návštěvníka. Za předpokladu, že chceme dosáhnout úplně nových a plnohodnotných výsledků, je vhodnou volbou sáhnout po crawlingu a vyhledat opravdu veškeré informace. Zajištění vyhledávání informací nám do jisté míry zajišťují vyhledávače a jejich crawleři, kteří denně prohledávají a aktualizují miliony webových sídel. To se ale děje v obrovském měřítku, které nemusí splňovat specifické předpoklady. Jaký vliv má ale použití vlastního crawlera, kterému lze přesně definovat cíl a metody získávání kýžených výsledků? V jakých situacích mohou být takto získaná data užitečná a jak s nimi naložit?
Pro zodpovězení výše položených otázek je nutné si definovat cílové skupiny uživatelů, jejich teoretické možnosti a problémy, se kterými se mohou setkávat, a najít pro ně řešení skrze využití webových robotů.

4.4.1 Cílová skupina běžných uživatelů

Do této skupiny spadá jakýkoliv návštěvník nebo uživatel webového sídla. Lze takto definovat běžnou uživatelskou skupinu, která se po internetu pohybuje v nejvyšším počtu a provádí běžné úkony typu vyhledávání informací, prohledávání webů apod.

Jedním z nejčastějších scénářů je případ, kdy uživatel navštíví webové sídlo poprvé. Robot by mohl vygenerovat sitemapu, respektive provést analýzu celého sídla a objevit veškeré podstránky. Při opakované návštěvě v určitém časovém horizontu by mohlo uživatele zajímat, k jakým změnám došlo, které stránky přibyly, které byly naopak odstraněny. Je do jisté míry možné tvrdit, že webová stránka, která by jednoznačně zveřejňovala takové změny, neexistuje. V našem případě by bylo ale možné použít znovu robota na generování sitemapy a v podstatě pouze porovnat aktuální verzi s verzí předchozí. Na základě tohoto porovnání by došlo k jasnému zobrazení nových a i starých (odstraněných) podstránek, které by nemusely býti na první pohled znatelně viditelné. Takový výstup nabízí snadný přehled o změnách webového sídla a nechá tak uživatele snadno reagovat na změny, případně na využití získaných dat.

Dalším vhodným příkladem je filtrování obsahu webového sídla, nebo dokonce porovnávání shodného obsahu mezi jednotlivými weby. Za předpokladu, že uživatel například odebírá zprávy z několika různých informačních serverů, je možné čas věnovaný jednotlivému zpravodajskému serveru zredukovat. Robot by mimo klasického sbírání URL adres mohl zařadit k výstupním datům také například titulek webové stránky, klíčová slova a první nadpis. Na základě těchto získaných dat z několika webových sídel by poté mohl provést vyhledávání, porovnání a zredukovat tak opakující se články. V případě využití dalších nástrojů by byl schopen indexovat celý obsah daného článku a provést i jeho porovnání na bázi jednotlivých slov, kde by byla jasně vidět jeho podobnost.

4.4.2 Cílová skupina uživatelů s omezením

Druhou cílovou skupinou jsou uživatelé s nějakým zdravotním omezením, které jim brání v běžném pohybu po webových sídlech, v nejčastějším případě slabozrací a nevidomí uživatelé. Slabozrací uživatelé jsou často odkázáni na využívání různých pomocných nástrojů, jako jsou lupy, se kterými lze zvětšovat text, případně další programy na úpravu barevného kontrastu zobrazovaného obsahu nebo přímo předčítače, které dokáží lineárně zpracovat obsah stránky a předčítat ho přímo uživateli [25]. Rozbalovací menu, kterými běžné webové stránky disponují a které nejsou složitou překážkou pro uživatele bez hendikepu, se ovšem stávají zásadní bariérou pro lineární zpracování. Předčítače prochází jednotlivé položky v menu a v případě, kdy je menu velice rozsáhlé, to může zabrat i nadmíru přívětivou dobu.

V případě, kdy by byla uživateli, respektive jeho předčítači, poskytnuta sitemapa, například včetně klíčových slov, titulku a nadpisu jednotlivých webových stránek, mohlo by to sloužit jako „předzpracování“ webového sídla pro lepší orientaci. Uživatel by mohl zadat, kterou stránku na sídle by si chtěl najít, a program by ho již provedl cestou, třeba za účelem provedení nákupu. Odpadla by tak starost se zdlouhavým „procházením“ webového sídla, což jistě cílová skupina ocení.

4.4.3 Cílová skupina vývojářů

Poslední skupinou jsou vývojáři, kteří jistě ocení využití sitemap v oblasti testování webových stránek a jejich SEO optimalizaci. V případě, kdy by docházelo ke sběru většího množství dat, než jsou pouze URL adresy, bylo by možné testovat dostupnost jednotlivých stránek. Každá stránka by mohla vypovídat o svém napojení na ostatní, o své hloubce umístění vzhledem ke kořenové stránce a obtížnosti dosažení této stránky pomocí délky cest. Takto zpracované výsledky by mohly napomoci k rozvoji nedostatečně propojených stránek a zvýšit tak kvalitu webového sídla.

Dalším příkladem může být využití crawlerů v marketingu, a to konkrétně jako silný marketingový nástroj pro provozovatele e-shopů, kteří potřebují mít přehled o cenách konkurence. V běžném případě by bylo nutné procházet web konkurence a manuálně porovnávat ceny produktů. Za předpokladu využití crawlera, který kromě sbírání URL adres získává taktéž podrobnosti o jednotlivých produktech, je už velice snadné porovnat ceny s produkty ve vlastním e-shopu. Případně je možné například vést statistiky a různé tabulky na zpracování cen, které jistě majitelé e-shopu ocení, aby byli o krok před ostatními.

Posledním příkladem je využití robota pro budování konzumního obsahu webového sídla v podstatě v jakémkoliv odvětví. Je možné crawleru vytyčit hranice, například v jaké části webového sídla se má pohybovat a co za data má získávat, a nechat si tak každý den v pravidelných opakujících se časových intervalech nahrávat aktuální data z několika zdrojů do databáze a prezentovat je tak na svém webu. Samozřejmě za předpokladu, že se na data nevztahuje autorský zákon, je možné je dále distribuovat a jsou volně dostupná. Takto například může fungovat webové sídlo zaměřující se na prodej krmiv pro zvířata a zároveň inzerující zvířata určená k adopci. Data o zvířatech by mohla být sbírána na jiných sídlech, například na webech zvířecích útulků apod.

5. Shrnutí výsledků

V předchozích kapitolách došlo k představení metod získávání dat z webových sídel. Jejich prvotní analýza z pohledu uživatele, kdy bylo možné zjistit vizuální rozvržení sídla a za pomoci vlastních nástrojů pro vývojáře integrovaných přímo v prohlížeči taktéž přiblížení jejich struktury v HTML kódu, byla následně zúročena při samotné tvorbě crawlera. Taktéž byl představen hlavní cíl práce, jímž je budování sitemapy neboli mapy webového sídla. Bylo vysvětleno, čím je tato mapa charakteristická, jaký má vlastně význam pro sídlo, ale i pro cílového uživatele. Zároveň bylo uvedeno, v jakém formátu se s mapou nejčastěji setkáme a jaké další alternativy může mít. Došlo k vysvětlení procesu získávání dat z webové stránky, tzv. data miningu a samotného principu web crawlingu, který byl následně podpořen několika praktickými ukázkami využití v různých odvětvích práce na internetu. Samotná existující implementace byla nejdříve představena krátkým seznamem jednotlivých kroků, kterým se řídil námi cílený webový crawler. Následně došlo k demonstraci samotné extrakce dat z webových sídel na základě přímého parsování HTML kódu a přístupu k jednotlivým elementům skrze CSS selektory a pomocí XPath. Dále byla zobrazena práce python knihoven Request, Urllib a BeautifulSoup s webovým protokolem, se stahováním jednotlivých stránek a aplikací některého ze selektorů k docílení zpracování obsahu a získání cenných dat. Za použití základních metod knihovny Urllib došlo také k funkčnímu představení základního crawlera pro procházení webového sídla a extrakci jednotlivých URL adres nacházejících se v mezích domény. Srovnání neušel ani python framework Scrapy, který primárně slouží ke scrapování webového obsahu. Tento framework byl představen a zvolen k budoucí realizaci cíle této práce. Výše zmíněná kapitola byla završena několika praktickými příklady využití crawlingu v praxi. A to hlavně na příkladech sloužících k výraznému ušetření opakující se nebo velice rozsáhlé práce, se kterou by člověk strávil značný čas, jenž byl tak ušetřen pomocí skriptu. V části realizace robota za účelem získávání sitemap došlo ke stanovení požadavků a zohlednění možných bezpečnostních problémů spojených s web crawlingem jak na straně cílového sídla, tak na straně hostujícího serveru. V další podkapitole byl zrealizován webový crawler pro procházení webového sídla a extrakci URL adres za pomoci frameworku Scrapy a jeho paralelních dotazů, které značně zrychlily proces prohledávání a docílily tak rychlejšího procházení celého sídla na základě přesně stanovených pravidel a omezení. Výstup dat z robota byl

zpracován pomocí další grafické knihovny Graphviz, která umožnila přetvořit výstup do podoby vhodné pro člověka, a prezentovat tak vizuální ztvárnění výsledné sitemapy. Sitemapa byla stále přehledná při zacílení jednotlivé dílčí části, ale při nahlížení na kompletní celek se stala nepřehlednou kvůli omezení prostoru k promítání a vnímání, jenž je člověk schopen pojmout. V poslední řadě došlo k představení využití sitemapy v reálném prostředí s cílením na skupinu běžných uživatelů, uživatelů s jistým zdravotním omezením a skupinu vývojářů. V případě první skupiny se jedná o zpracování webového sídla a jeho sitemapy za účelem budoucího porovnávání provedených změn na úrovní přidávání a odebírání jednotlivých stránek. Další příklad se vztahuje k porovnávání více zdrojů tvořících webová sídla a případně jejich obsahů za pomoci dalších nástrojů a samotné sitemapy. Skupině uživatelů se zdravotním omezením, nejčastěji nevidomým a slabozrakým, může mapa usnadnit celkovou orientaci na webovém sídle, pomoci je provést skrze pro ně nepříliš přívětivé ovládání a umožnit jim například snadněji provést nákup ve spolupráci s dalšími nástroji. Poslední skupinou jsou vývojáři, kde je možnost využití crawlerů zdaleka největší. Od využití sitemap při optimalizaci webového sídla, přes vhodný nástroj pro SEO, po testování dostupnosti jednotlivých webových stránek a jejich vazeb vzhledem ke zbytku v sídle. Taktéž efektivita v marketingu dosahuje žádané poptávky o srovnávače cen apod., kdy majitelé e-shopů využívají crawlery pro získávání cen produktů konkurence, a to i u několika desítek tisíc položek.

6. Závěry a doporučení

V rámci práce, která si dávala za úkol vytvoření skriptu, robota, který bude schopen procházet předem dané webové sídlo a vytvářet z něj sitemapy, došlo k představení několika možných způsobů a metod, jak takového výsledku docílit. V případě použití základních knihoven popsaných v textu se sice docílilo postupného procházení webového sídla, avšak ne za nejlepších podmínek. Nejdůležitější z nich byla bezpečnost skriptu vzhledem k procházení webového sídla. Nebyla zde plně zohledněna možnost docílení IP zákazu od cílového serveru, který si crawling mohl vyložit jako pokus o útok a umístit tak hostující IP adresu skriptu na černou listinu. Dalším problémem byla efektivita a rychlost procházení. Takto navrhované skripty byly pomalé a měly problém s procházením menších webových sídel o rozměrech několika desítek tisíc odkazů, jenž jim trvalo několik desítek hodin, což je v dnešní době výpočetních kapacit nepředstavitelné. Pozitivní výsledek měl až framework Scrapy, který je přímo určen k získávání dat z webových stránek. S podporou rozsáhlé komunity uživatelů a vývojářů tak drží krok s dnešní dobou a je v neustálém vývoji. Scrapy díky své architektuře zvládá mnohem vyšší rychlost procházení webových stránek a taktéž umožňuje škálovatelnost a rozdělení na moduly, kde každý řeší svoji přesně stanovenou část, ať už přímý přístup k webu, nebo jeho zpracování a případně výstup dat.

Zpracování výstupních dat a zvolení pro člověka vhodné formy bylo složitou volbou, která byla nakonec řešena pomocí vizualizační knihovny Graphviz, jež umožnila alespoň hrubý náhled zpracované sitemapy.

Největší překážkou však byla rozdílnost jednotlivých webových sídel. Nedá se s přesností předpovídat, co robota na sídle čeká a jak ho na to připravit. Nelze také předem určit, jak zabránit případné blokaci ze strany cíleného webu nebo jak vytvořit univerzálního robota, který zvládne analyzovat kterékoliv webové sídlo. Další otázkou, která nastala během této práce, je případ, kdy by byl přístup na webové sídlo nebo pouze do nějaké části chráněn ochranou proti robotům (například Captcha) – jak by si robot s touto ochranou poradil?

Pokud bychom se na problematiku podívali z druhé strany, tedy z pohledu zohlednění ochran proti crawlerům, mohou právě tyto roboti sloužit k vlastnímu obohacení jejich autorů, k šíření dezinformací nebo způsobení finančních škod. Jak se proti takovým crawlerům chránit? Jak zabránit robotům, aby měli přístup k obsahu? V této dynamicky

se rozvíjející oblasti tedy vystává mnoho neznámých, na jejichž objasnění by se měly zaměřit další práce. Dnes je pouze diskutabilní, kam až možnosti scrapingu v budoucnosti povedou.

7. Seznam použité literatury

[1] Website scraping with python: using beautifulsoup and scrapy. New York, NY: Springer Science+Business Media, 2018. ISBN 978-1-4842-3924-7.

[2] Python Web Scraping. Birmingham, Packt Publishing Ltd., 2017. ISBN 978-1-78646-258-9.

[3] Google: Vytvoření souboru robots.txt [online]. 2019 [cit. 2019-08-19]. Dostupné z: https://support.google.com/webmasters/answer/6062596?hl=cs

[4] KOSTER, Martijn. A Standard for Robot Exclusion [online]. [cit. 2019-08-19]. Dostupné z: https://www.robotstxt.org/orig.html

[5] GOOGLE TEAM, Google. Informace o souborech Sitemap [online]. [cit. 2019-08-19]. Dostupné z: https://support.google.com/webmasters/answer/156184?hl=cs&ref_topic=4581190&visit_id=636896242892614474-3973430160&rd=1

[6] GOOGLE TEAM, Google. Soubory Sitemap pro obrázky [online]. [cit. 2019-08-19]. Dostupné z: https://support.google.com/webmasters/answer/178636

[7] GOOGLE TEAM, Google. Soubory Sitemap pro videa a alternativy k souborům Sitemap pro videa [online]. [cit. 2019-08-19]. Dostupné z: https://support.google.com/webmasters/answer/80471

[8] Sitemaps XML format [online]. [cit. 2019-08-19]. Dostupné z: https://www.sitemaps.org/protocol.html

[9] Media RSS Specification [online]. [cit. 2019-08-19]. Dostupné z: http://www.rssboard.org/media-rss

[10] GOOGLE TEAM, Google. Vytvoření a odeslání souboru Sitemap [online]. [cit. 2019-08-19]. Dostupné z: https://support.google.com/webmasters/answer/183668?hl=cs&ref_topic=4581190

[11] TECHOPEDIA. What is a Parser? - Definition from Techopedia [online]. [cit. 2019-08-21]. Dostupné z: https://www.techopedia.com/definition/3854/parser

[12] PYTHON, DOC. Urllib.request — Extensible library for opening URLs — Python 3.7.4 documentation [online]. [cit. 2019-08-21]. Dostupné z: https://docs.python.org/3/library/urllib.request.html#module-urllib.request

[13] PYTHON, DOC. Urllib.parse — Parse URLs into components — Python 3.7.4 documentation [online]. [cit. 2019-08-21]. Dostupné z: https://docs.python.org/3/library/urllib.parse.html#module-urllib.parse

[14] WHITMER, Ray. Document Object Model [online]. [cit. 2019-08-21]. Dostupné z: https://www.w3.org/DOM/

[15] Learning Scrapy. Birmingham, Packt Publishing Ltd., 2016. ISBN 978-1-78439-978-8.

[16] HAN, Jiawei a Micheline KAMBER. Data mining: concepts and techniques. 3rd ed. Burlington, MA: Elsevier, c2012. ISBN 978-0-12-381479-1.

[17] SELENIUM TEAM. Selenium browser automation [online]. [cit. 2019-08-21]. Dostupné z: https://www.seleniumhq.org/

[18] PHANTOMJS TEAM. PhantomJS - Scriptable Headless Browser [online]. [cit. 2019-08-21]. Dostupné z: https://phantomjs.org/

[19] CHROMIUM TEAM. ChromeDriver - WebDriver for Chrome [online]. [cit. 2019-08-21]. Dostupné z: https://chromedriver.chromium.org/downloads

[20] Prevent a Cross-Site Request Forgery with a CSRF token. [online]. [cit. 2019-08-21]. Dostupné z: https://www.veracode.com/security/csrf-token

[21] Ochrana proti padělání požadavků [online]. [cit. 2019-08-21]. Dostupné z: https://docs.microsoft.com/cs-cz/aspnet/core/security/anti-request-forgery?view=aspnetcore-2.2

[22] What is an API? (Application Programming Interface) [online]. [cit. 2019-08-21]. Dostupné z: https://www.mulesoft.com/resources/api/what-is-an-api

[23] DESAI, Keyur et al. Web Crawler : Review of Different Types of Web Crawler, Its Issues, Applications and Research Opportunities. *International Journal of Advanced Research in Computer Science* [online]. 2017, vol. 8, no. 3.

[24] MIRTAHERI, Seyed M. et al. A brief history of web crawlers [online]. Ithaca: Cornell University Library, arXiv.org, 2014. Copyright - © 2014. This work is published under http://arxiv.org/licenses/nonexclusive-distrib/1.0/ (the "License"). Notwithstanding the ProQuest Terms and Conditions, you may use this content in accordance with the terms of the License; Poslední aktualizace - 2019-05-13.

[25] DOLEŽALOVÁ, Markéta. Webová přístupnost českých zpravodajských portálů se zaměřením na slabozraké uživatele [online]. Brno, 2019 [cit. 2019-08-21]. Dostupné z: <https://is.muni.cz/th/xraly/>. Diplomová práce. Masarykova univerzita, Fakulta sociálních studií. Vedoucí práce Jakub Macek.

[26] BANK, Sebastian. *Graphviz* [online]. [cit. 2019-08-21]. Dostupné z: https://graphviz.readthedocs.io/en/stable/index.html

8. Seznam použitých obrázků

9. Přílohy

1)