Seminarska naloga 1 – Spletni pajek

Neža Bačar, Ana Čepuran, Bojan Vrangeloski

***Abstract*—V poročilu je opisana implementacija spletnega pajka za predmet Iskanje in ekstrakcija podatkov iz spleta. V uvodnem delu je pojasnjen namen naloge, v drugem delu pa je podrobno razložena implementacija. Sledijo opisane težave pri implementaciji ter predstavitev rezultatov in vizualizacija.**

I. Uvod

Spletni pajek (ang. web crawler) je avtomatiziran program oziroma skripta, ki samostojno preiskuje spletne strani na internetu. Ob tem skuša določiti in izvleči želene podatke iz spletnih strani. Podatki se potem shranijo v podatkovno bazo in se kasneje uporabijo v različne namene[1].

Cilj seminarske naloge je implementacija večnitnega spletnega pajka, ki išče le spletne strani z domeno gov.si. Pajek je implementiran po principu iskanja v širino (ang. breadth-first).

II. Implementacija

Pridobljene podatke hranimo v podatkovni bazi PostgreSQL. Vnaprej nam je bila podana skripta za implementacijo baze, ki smo jo dopolnili. Kot seznam fronte (ang. frontier) smo uporabili kar tabelo ”page”, kateri pa smo dodali atributa ”finnished” in ”page_hash”. Tabeli ”site” smo dodali atribut ”crawldelay_content”. Za implementacijo vzporednih pajkov smo uporabili knjižnico threading[2]. Le ta omogoča, da posamezno nit implementiramo kot funkcijo. Logiko pajka smo razdelili na več delov:

main.py: inicializacija pajka in niti

db.py: dodajanje in urejanje elementov v podatkovni bazi

robots.py: funkcije za pridobivanje Robots.txt datotek in podatkov iz le teh

URLfrontier.py: logika procesiranja trenutne strani v seznamu fronte

dataExtractor.py: izločanje url-jev (slike, datoteke, . . .) iz trenutne strani

duplicateDetector.py: iskanje duplikatov glede na vsebino ali povezavo

A. Inicializacija

Inicializacija samega pajka se nahaja v skripti ”main.py”. Število pajkov podamo kot vhodni parameter programa. Prvi del zajema umestitev začetnih strani (ang. seed pages) v seznam fronte. Sledi inicializacija povezave s podatkovno bazo. Ta del se venomer izvede v eni niti. V nadaljevanju se izvede inicializacija vseh niti. Za vsako niti inicializiramo svoj Selenium WebDriver[3]. Pri tem upoštevamo zastavico ”headless”. Posamezen pajek izvaja preiskovanje, dokler v seznamu fronte ni več strani za obdelavo. V primeru, da želi več pajkov sočasno klicati funkcijo, ki iz seznama fronte pridobi naslednjo stran za obdelavo, si pomagamo z razredom Lock() knjižnice threading. Da niti v primernem razmaku procesirajo posamezne url-je pa kličemo tudi funkcijo time.sleep().

B. Seznam fronte

Za inicializacijo seznama fronte smo uporabili kar tabelo ”page”. Najprej se v tabelo umestijo začetne domene. Tabele imajo glede na vrstni red umestitve pripadajoč id. Preden vzamemo stran iz seznama fronte, le te razporedimo po naraščajočem vrstnem redu glede na ”id”. Ko začnemo s procesiranjem določene strani, v podatkovno bazo nemudoma vpišemo ”accessed_time” strani, ki drugim nitim preprečuje, da bi isto stran vzele iz seznama fronte. Ko stran obdelamo, v podatkovni bazi atribut ”finnished” nastavimo na ”true”. To lahko storimo le, če ima tudi predhodna stran v seznamu fronte atribut ”finnished” nastavljen na ”true”. V nasprotnem primeru počakamo, da pogoj velja. Na tak način smo zagotovili, da pajek deluje po principu iskanja v širino. S tem smo torej omogočili, da se strani vnašajo v seznam fronte v pravilnem vrstnem redu.

C. Detekcija duplikatov in obdelava strani

Ko začnemo posamezno stran obdelovati, se najprej s pomočjo Selenium WebDriverja[3] pridobi vsebina posamezne strani. Na tem mestu tudi poskrbimo, da se zaradi upoštevanja pravil o dostopanju do domen vsakokrat izvede time.sleep(5). Če v določenem času ne pridobimo vsebine in statusne kode strani, v polje ”html_content” vpišemo niz ”INACCESSIBLE”. Preverimo tudi, če stran zares vsebuje html vsebino, tako da preverimo polje ”Content-Type”. Če stran, ne vsebuje html vsebine, atribut page_type_code ostane NULL, nastavimo pa le ”status_code” in parameter ”finnished”.

V primeru, da ne pride do težav s pridobivanjem strani, se ustvari izvleček (ang. hash) html vsebine na podlagi katerega se preveri, ali v ”page” tabeli obstaja duplikat te strani. Ustvarjen izvleček se primerja s ”page_hash” atributom vseh že obdelanih strani. Če je stran zaznana kot duplikat, ji v bazi dodelimo vrednost ”DUPLICATE”, ”html_content” pa se nastavi na ”NULL”. Hkrati se ustrezno posodobi tabela ”link”.

Če stran ni duplikat, se nadaljuje obdelava vsebine strani. Najprej za sledečo stran iz tabele site pridobimo domeno. S pomočjo podatkov domene upoštevamo pripadajočo Robots.txt vsebino. Najprej iz strani izluščimo vse slike, ki so predstavljene z elementom ”img”. Le teh ne dodamo v seznam fronte, temveč ustrezno posodobimo tabelo ”image”. Potem izluščimo še povezave z elementom ”a”, za katere velja ”href=True”. Nazadnje pregledamo tudi povezave, za katere velja ”onclick=True”. Iz slednjih s pomočjo regularnih izrazov izločimo povezave, ki vsebujejo location.href ali docu-

ment.location. Vse pridobljene povezave po potrebi ustrezno razširimo z bazno (ang. base) povezavo ter jih normaliziramo s pomočjo knjižnice urllib[4]. Izvedemo tudi dodatno obdelavo povezav – odstranitev '/' na koncu povezav, odstranitev 'www' na začetku itd. Povezav ne dodamo takoj v bazo, temveč jih dodamo v tabelo "all_new_pages", ki hrani vse najdene povezave na strani ter pripadajoče domene.

V primeru, da povezava kaže na stran z drugo gov.si domeno, izluščimo domeno in preverimo, ali je le ta že v tabeli "site". Če še ni dodana, pridobimo Robots.txt datoteko, ter stran vstavimo v "site" in "page" tabelo. Če pa je domena že v tabeli "site", preverimo, ali je pripadajoča stran že v "page" tabeli. Če je še ni, jo dodamo v tabelo "all_new_pages". Preden povezavo dodamo v omenjeno tabelo, preverimo ali ima končnico, ki označuje binarno datoteko. Takšnih povezav ne dodamo v seznam fronte, vendar ustrezno posodobimo tabelo "page_data".

Najdenih povezav torej ne dodajamo nemudoma v bazo, temveč jih sprva dodamo v tabelo. Pred dodajanjem posameznega url-ja v tabelo preverimo, ali v tabeli "page" le ta že obstaja. Če povezava že obstaja, je ne dodamo ponovno. S tem pripomoremo, da se pajek ne ujame v zanko. Ko obdelamo celotno stran, preverimo, ali ima predhodna stran v seznamu fronte parameter "finnished" nastavljen na "true" in šele v tem primeru povezave dodamo v tabelo "page" in primerno posodobimo tabelo "link". Na tak način omogočimo iskanje v širino.

D. Robots.txt

Robots.txt vsebino pridobivamo vsakič, ko v tabelo "site" dodamo novo domeno. Pri tem ločimo celotno Robots.txt vsebino, Sitemap in Crawl-Delay. V primeru, da naš User-Agent na stran ni dovoljen, strani ne dodamo v seznam fronte, temveč jo dodamo le v "sites" tabelo ter na tak način preprečimo nadaljnje obdelovanje in kršenje pravil.

Ob pridobitvi nove strani iz seznama fronte najprej upoštevamo Robots.txt vsebino. Le to pridobimo od ustrezne domene v tabeli "site". Najprej upoštevamo Crawl-delay in v primeru, da ni nič, na ustrezen parameter nastavimo time.sleep() funkcijo. Tabela vsebuje tudi izpolnjeno Sitemap vsebino, ki pa je pri naši implementaciji ne upoštevamo. Ustrezno izločimo tudi Allow in Disallow pravila, upoštevamo pa le slednja.

III. Težave pri implementaciji

Pri implementaciji se je pojavilo kar nekaj težav, ki smo jih skušali razrešiti premišljeno in sistematično. Nekaj težav smo imeli s pravilnim razširjanjem relativnih povezav iz strani v primeru, da stran ni imela določene bazne strani. Pri tem smo si pomagali s funkcijo urljoin knjižnice urllib[4]. Pri branju Disallow in Allow pravil smo opazili, da imajo določene strani dodano polje Disallow, ki pa je prazno. Te primere smo morali ustrezno obravnavati. Opazili smo tudi, da Sitemap povezava v določenih primerih ni bila zapisana v novi/ločeni vrstici, kar je otežilo zaznavanje. Težavno se nam je zdelo tudi upoštevanje Allow pravil, ki bi jih verjetno implementirali s pomočjo regularnih izrazov. Ker smo opazili, da večina Robots.txt datotek nima Allow pravil, smo jih le prebrali, potem pa smo se raje posvetili implementaciji ostalih funkcionalnosti. Težave smo imeli tudi s pravilno implementacijo vzporednosti in upoštevanja iskanja v širino. Pri tem smo si pomagali z Lock() in time.sleep() funkcijo, hkrati pa smo bazo razširili z dodatnimi polji, ki zagotavljajo, da program deluje pravilno. Za ekstrakcijo podatkov iz html strani, smo uporabili knjižnico BeautfiulSoup[5]. Pri določeni povezavi[6] je ekstrakcija podatkov vrnila napako, da smo presegli limito rekurzije (ang. recursion limit). Po tem pojavu, smo limito dvignili na 1500.

Nekaj dni pred oddajo smo bili zaradi prevelikega dostopa do določene domene blokirani za 24h. Težavo smo sicer razrešili s podaljšanjem roka, vendar v resničnem svetu to ni možna rešitev.

IV. Rezultati in vizualizacija

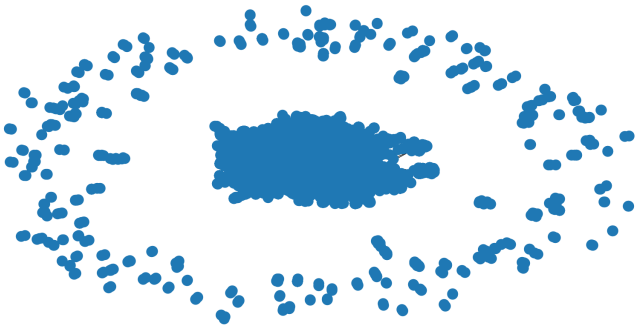

Fig. 1. Vizualizacija link tabele - povezave prvih 8000 strani

TABLE I
PODATKI ZA VSE DOMENE GOV.SI.

	Količina	Povprečna količina na stran
Domene	301	-
Spletne strani	51364	-
Duplikati	2574	0.05
PDF	27749	0.54
DOC in DOCX	9397	0.18
PPT in PPTX	130	0.003
Slike	380456	7.4

TABLE II
PODATKI ZA ZAČETNE STRANI.

	Količina	Povprečna količina na domeno
Domene	4	-
Spletne strani	10890	2772
Duplikati	97	24.25

Rezultati so predstavljeni z vizualizacijo tabele link, ki označuje povezave prvih 8000 strani. Posamezne strani so predstavljene z modro piko, vmesne povezave pa so predstavljene s črno črto.

Iz baze smo izluščili tudi nekaj drugih podatkov. Iz tabele 1 je razvidno, da v preiskanih domenah prevladujejo PDF datoteke, zelo malo pa je bilo PPT datotek. Približno petina preiskanih spletnih strani pripada začetnim domenam.

REFERENCES

[1] http://eprints.fri.uni-lj.si/2229/1/Petrovi
[2] https://docs.python.org/3/library/threading.html
[3] https://www.selenium.dev/documentation/en/webdriver/
[4] https://docs.python.org/3/library/urllib.html
[5] https://www.crummy.com/software/BeautifulSoup/bs4/doc/
[6] https://www.e-prostor.gov.si/zbirke-prostorskih-podatkov/zbirni-kataster-gospodarske-javne-infrastrukture/