UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

ISKANJE IN EKSTRAKCIJA PODATKOV S SPLETA

SEMINARSKA 1

Spletni pajek

Končno projektno poročilo

Avtorja:
BERNIK Matic
TOVORNIK Robert

Profesor:
dr. Marko BAJEC
Asistent:
dr. Slavko ŽITNIK

April 6, 2019

1 Uvod

1.1 Opis problema

V okviru prve seminarske naloge smo se odločili implementirati samostojnega univerzalnega pajka, katerega naloga je sprehoditi se po spletišču z začetkom v podanih semenskih straneh. Ob sprehajanju po spletišču, si pajek shranjuje vnaprej določene pomembne vsebine, kot so npr. slike, datoteke,.. predvsem pa nadaljnje povezave, ki se nahajajo na trenutni strani. Z iskanjem zadnjih si pajek gradi nadaljnjo pot, po kateri potuje. Pri potovanju se usmerja najprej na strani, ki jih je najprej našel.

1.2 Preiskovalne domene

Pri preiskovanju spleta smo delovanje pajka omejili na slovenske domene vladne organizacija (gov.si). Takšen način preiskovanja se uporablja predvsem kadar nas zanimajo zgolj določeni tematski sklopi ali pa povezanost (arhitektura) manjših lokalnih spletišč. Začetek preiskovanja smo pričeli z dvema sejnima stranema: "http://evem.gov.si/" in "http://e-prostor.gov.si/". Kasneje pa smo preiskovanje nadaljevali na naslednjih sejnih straneh:

- 'http://evem.gov.si',
- 'http://e-uprava.gov.si',
- 'http://podatki.gov.si',
- 'http://e-prostor.gov.si',
- 'http://mizs.gov.si',
- 'http://mddsz.gov.si',
- 'http://mz.gov.si',
- 'http://uvps.gov.si',
- 'http://mf.gov.si'

2 Podatkovna baza

2.1 Arhitektura podatkovne baze

Osnovno definicijo strukture podatkovne baze crawldb smo nadgradili tako, da smo v tabelo 'page' dodali atribut 'minhash' namenjen hrambi zgoščene vrednosti vsebine prenesene strani z namenom olajšanja in pohitritve kasnejše detekcije duplikatov strani.

Ker smo spletnega pajka želeli zasnovati kar se da skalabilno, smo si v obliki dodatne tabele 'frontier' tudi implementirali vrsto poslov, do katerih bi lahko dostopali odjemalci tudi z razlicnih sistemov oz. procesov, ne le iz vecih niti istega procesa.

V tabeli hranimo:

- 'placement' - PK serial,
- 'id' - identifikator strani (tabela 'frontier' je preko tujega ključa povezana s tabelo 'page'),
- 'status' - status trenutnega opravila ('wating','processing' ali 'done'),
- 'processing_start_time' - čas začetka obdelave posla, na podlagi katerega lahko zastaran posel vrnemo v stanje čakanja,
- 'depth' - globina, s pomočjo katere lahko zagotovimo strategijo preiskovanja v širino BFS.

2.2 Posebnosti / streznik

Zaradi načina implementacije 'frontier' vrste v podatkovni bazi in sočasnega postopa večih procesov, je bilo potrebno ob prevzemu novega posla potrebno paziti na pravilno zaklepanje zapisa v izogib situaciji, kjer bi več procesov prevzelo isti posel.

Zal je to privedlo do problemov medsebojnega blokiranja vzporedno-delujočih procesov ('deadlock'). Teh v času pisanja poročila nismo uspeli razrešiti (poskušali smo tudi z optimizacijo strežniških parametrov) in še vedno predstavlja glavno omejitev hitrosti delovanja pajka.

3 Spletni pajek

3.1 Krovno delovanje pajka

Pajek podatke pomni na račun uporabe postgresql podatkovne baze, katere strežnik je dostopen iz spleta. To nam omogoča, da lahko procese pajka zaganjamo z različnih lokacij in IP naslovov, ter tako povečamo njegovo zmogljivost. Pred zagonom programa ('crawler.py') le-temu pripravimo .json datoteko s podatki potrebnimi za povezavo na podatkovni strežnik in nastavimo število delovnih procesov oz. uporabljenih niti.

Pajek nato ustvari delovne niti od katerih vsaka odpre svojo povezavo s podatkovnim strežnikom (pomembno zaradi višje izkoriščenosti strežnika). Po tem, ko so niti pognane proces v zanki dektetira zastarele posle in jih sprošča.

Izvajanje vsake od niti je robustno in sposobno obvladati tudi nepredvidljive situacije, ki v programu prozijo izjeme. V teh primerih se trenutna transakcija podatkovne baze razveljavi, obravnavani posel sprosti, delavec pa gre v začeno fazo prevzema novega posla.

Delavni procesi med seboj komunicirajo tako, da dostopajo do skupnega medpomnilnika podatkov domenskih omejitev iz robots.txt datotek in seznama časovnikov zadnjih obiskov posamezne domene.

Slednje pajek uporablja ne-le za spoštovanje omejitev časa med poizvedbami na strani posamezne domene, temveč tudi za pametno izbiro naslednjega posla, pri čemer poskuša izbrati URL z domene od obiska katere je preteklo največ časa.

3.1.1 Upravitelj delavcev

3.1.2 Delovanje posameznega delavca

Vsakega izmed delavcev svojo nalogo opravlja na enak način. Ustvarimo ga kot samostojno nit, ki pa pri delovanju upošteva tudi delovanje preostalih delavcev (niti). V trenutku, ko posameznega delavca ustvarimo, mu dodelimo nekaj pomembnih informacij, kot so: identiteta, stanje, naslovi semenskih domen, .. kot zadnjem, pa prejme samostojno instanco spletnega gonilnika (chrome - webdriver). V tem trenutku se delavec dokončno inciializira in prične s svojim delovnim ciklom. Cikel vsakega izmed delavcev sestoji iz naslednjih korakov: pridobitev ali pregled robots.txt, pridobitev ali pregled sitemapa, pridobitev spletne povezave s frontierja, pridobitev izvorne kode strani za pridobljeno povezavo, hashanje strani, vsebinsko primerjanje strani za duplikate,

ekstrakcija in sortiranje vsebin spletne strani, prenos zahtevanih binarnih vsebin, shramba vsebin v podatkovno bazo.

3.2 Delovanje posameznih sklopov

3.2.1 BFS Frontier

Frontier smo implementirali kot tabelo podatkovne baze. Vsak od delavcev od tam prevzame naslednji posel, pri čemer poskuša spoštovati naslednja pravila:

- posel naj bo čakajoč (status 'waiting'),
- globina ('depth') posla naj bo enaka najmanjši vrednosti atributa 'depth' čakajočih poslov,
- če je mogoče izberi stran iz domene, od obiska katere je preteklo najdalj časa.

Prevzeti posel v bazi označi s stanjem 'processing'.

3.2.2 Robots.txt in Sitemap

- sposotvanje pravil, alternative (ce ga ni) - dodajanje vsebin iz sitemap

3.2.3 Duplikati in preverjanje duplikatov in hashanje (LSH-SHA1)

Delo pajka je lahko zamudno, saj obstaja veliko povezav, pogostokrat več povezave usmerja na isto vsebino, prav tako pa je za vsako novo povezavo dodatno obremenjen steznik. Ker želimo biti pri preiskovanju zahtevnega prostora učinkoviti, je pomembno, da isti vsebin in strani ne obiskujemo večkrat. Zato vsebine dedupliciramo (odstranimo ponovitve).
Prvi in časovno najmanj zahteven korak je, da pred dodajanjem novih povezav v preiskovalni prostor, preprosto preverimo, če smo takšno povezavo že obdelali oziroma, je že v čakalni vrsti. V danem primeru, takšno povezavo preprosto preskočimo. Pred primerjanjem je povezavo potrebno normalizirati (canonalizirati).
V kolikor ujemanja ne najdemo pa je potrebno narediti vsebinsko primerjavo. Vendar je takšna primerjava, če bi želeli primerjati znak po znak, časovno izjemno zamudna in neučinkovita. Zato se pristopa lotimo drugače. Ena izmed

možnosti, ki pospeši zadeve je pristop, kjer vsebino razbijemo na tokene ali pa vektorsko predstavitev prisotnih vsebin ter nato izvedemo primerjavo. Vendar je tudi to zamudno.
Izjemno učinkovite pa so predstavitve s hashi. Takšne predstavitve so izjemno učinkovite pri primerjavah, kar nam je pomembno. Velikokrat uporabljen algoritem pri takšnem pristopu, oziroma še večkrat za odkrivanje podobnih sosedov je prav hashanje z LSH. Bistvo postopka je, da podobne vsebine predstavi s podobnimi elementi, oziroma jih predstavi podobno, kadar so si vsebinsko blizu (local-sensitivity). Prednost pristopa je, da omogoča hitro odkrivanje podobnih vsebin z ožanjem nabora preko selekcij n-različnih hash predstavitev. Podobnost nato določimo glede na threshold, ki si ga postavimo. Vendar tudi takšno iskanje zahteva ekstrakcijo velike količine podatkov iz podatkovne baze. Zato sva se odločila za pristop hashanja, ki sicer ne omogoča postavitve thresholda, vendar enake vsebine vedno zakodira enako, ne glede na katerem računalniku ga poženemo predvsem pa je izjemno hiter. Zaradi unikatnih hashov, omogoča direktno primerjanje z iskanjem prisotnosti elementa v bazi. Zato sva za odkrivanje direktnih duplikatov uporabila hashanje s funkcijo "sha1" iz Python knjižnice "hashlib".

3.2.4 Prevzem strani / izvorne kode

Prevzem spletne vsebine je izjemno pomemben korak, pri katerem lahko pride do glavnih razlik v delovanju spletnih pajkov. Namreč zaradi učinkovitosti delovanja in preproste razširljivosti vedno več strani uporablja dinamično arhitekturo spletnih strani. Tako se zgodi, da se ob poslanemu zahtevku, namesto celotne strani in večine vsebine naloži zgolj okvir spleten strani, s čimer se izgubi namen spletnih pajkov. Da bi se problemu izmaknila, sva se odločila spletnega pajka implementirati na način, ki podpira statične in dinamične (javascript vodene) spletne strani. Pri implementaciji sva zato uporabila knjižnico "selenium", bolj natančno "selenium.webdriver" s katero sva za odpiranje spletnih strani simulirala uporabo brskalnika Google Chrome ("chromewebdriver"). Zaradi učinkovitosti, sva simulacijo poganjala v "headless" načinu, tj. brez grafične vmesnika in brez dovoljenega zvoka ("mute"). Dodatna prednost gonilnika je v uporabi funkcije ".get()", ki za vsako spletno stran pred pregledom vsebine počaka na signal, da se je stran v celoti naložila. Dodatno smo upoštevala tudi možnost, da stran nikoli ne zaključi z nalaganjem, ali pa da strani ne uspe pridobiti, zato sva dodala časovno omejitev, čakajočih 10 sekund. Ker selenium ne omogoča učinkovitega branja odzivnih kod strežnika, sva do-

datno uporabila še knjižnico request s katero sva zahtevek poslala vnaprej, preverila če se stran odziva in shranila odzivno kodo, v primeru neodzivnosti, pa sva stran popolnoma izpustila. Pri obiskovanju sva dovolila preusmeritve, če so bile te izvedljive v dovoljenem času. Prav tako sva izključila preverjanje SSL certifikatov, saj so prav varnostne izjeme povzročale največ izjem.
Pomembna izboljšava s stališča učinkovitosti je bila zgolj enkratna inicializacija instance chromedriverja ob zagonu delavca in nato zgolj posodabljanja strani v aktivnem oknu. Pred tem sva ob vsaki novi povezavi ponovno inicializirala instance in jih tudi zapirala (zaradi čistoče delovanja), vendar se je poteza izkazala za časovno izjemno zahtevno.
Sledilo je testiranje v načinu delovanja. S primerjavami klicev nad 20 spletnimi mesti se je pokazala glavna razlika.
Hitrostna primerjava delovanja chromedriverja.

- Dinamična inicializacija na n straneh (n=20): 54s
- Dinamična posodobitev vsebine na n straneh (n=20): 13s

3.2.5 Obdelava strani, ekstrakcija vsebin

Pri obdelavi vsebine strani (izvorne kode), sva se odločila za uporabo knjižnice BeautifulSoup, ki omogoča napredno procesiranje in učinkovito vsebinsko iskanje po HTML dokumentih. Knjižnico sva uporabila za iskanje želenih vsebin na spletni strani. Ker sva spletne strani shranila v izvorni kodi, ki sva jo pridobila s simulacijo brskalnika (selenium-webdriver), so bile v izvorni kodi prisotne vse mogoče povezave, vključno z dinamičnimi vsebinami (Javascript). Sledila je "preprosta" ekstrakcija nam zanimivih vsebin (slike, dokumenti, nove povezave). Zaradi unikatne lastnosti povezav, sva najprej poiskala vse slikovne vsebine. Iskala sva prisotnost HTML značk tipa "img", v katerem se praviloma nahajajo grafične vsebine. Ker se včasih zgodi, da so takšne značke napačno uporabljene ali pa brez vsebine / povezav sva eliminirala vse elemente, kjer atribut "src" ni bil prisoten. Preostale povezave sva shranila v seznam slik.
Zatem je sledil korak iskanja vseh preostalih povezav, ki nas zanimajo. Zato sva preiskala vse značke povezav "a", ki vsebujejo atribut "href", kar nakazuje na vsebovanost spletne povezave. Povezave tipa "link" sva izpustila, saj se v večini primerov uporabljajo za lokaliziranje Javascript datotek, ki jih stran uporablja.
V naslednjem koraku sva vse preostale povezave razvrstila glede na vrstno končnice povezave. S tem postopkom, sva povezave ločila na dokumente (.pdf, .doc, ppt,..) povezave spletnih mest.

Ker se med povezavami pogostokrat pojavijo tudi zgolj delne povezave (dinamične povezave), sva povezave dodatno uredila, preverila in povezala v primeru, ko je šlo za dinamične vsebine (url-trenutne-strani + dinamicni-del-url).

3.2.6 Normalizacija povezav

Povezave, ki jih zberemo v postopku ekstrakcije, so pogostokrat neurejene, neveljavne zaradi prisotnosti odvečnih znakov, v napačni obliki ali pa se preprosto ponavljajo zaradi ne razlikovanja med parametri (ki ne določajo spletnega naslov strani). Zato je takšne strani potrebno prečisti.
Za ta postopek sva poskrbela v funkciji canonicalize-url. V splošnem je sestavljena iz treh korakov. Prvi korak dobljeno povezavo normalizira (odstrani odvečne znake, doda manjkajoče, normalizira veliko / malo pisavo itd..). V drugem koraku, url povezave razbijemo na posamezne logične enote (naslovnik, domena, pot strani, parametri,..). V tem koraku odstranimo vse nepotrebne parametere, ki ne vplivajo na dostopanje do vsebine spletne strani (npr. nastavitve teksta, pisav, kakršnikoli meta ali časnovni podatki,..). V zadnjem koraku, po končanem združevanju enot v celoten naslov, povezavo ponovno normaliziramo (da odpravimo morebitne napake pri združevanju).
Pri implementaciji omenjenega postopka, sva si pomagala s knjižnicama url-normalize in urlparse.

Za konec sva postopek testirala nad povezavami omenjenimi na predavanjih.

Description and transformation	Example and canonical form
Default port number Remove	http://cs.indiana.edu:80/ http://cs.indiana.edu/
Root directory Add trailing slash	http://cs.indiana.edu http://cs.indiana.edu/
Guessed directory* Add trailing slash	http://cs.indiana.edu/People http://cs.indiana.edu/People/
Fragment Remove	http://cs.indiana.edu/faq.html#3 http://cs.indiana.edu/faq.html
Current or parent directory Resolve path	http://cs.indiana.edu/a/./../b/ http://cs.indiana.edu/b/
Default filename* Remove	http://cs.indiana.edu/index.html http://cs.indiana.edu/
Needlessly encoded characters Decode	http://cs.indiana.edu/%7Efil/ http://cs.indiana.edu/~fil/
Disallowed characters Encode	http://cs.indiana.edu/My File.htm http://cs.indiana.edu/My%20File.htm
Mixed/upper-case host names Lower-case	http://CS.INDIANA.EDU/People/ http://cs.indiana.edu/People/

Figure 1: Primer pravilno prečiščenih povezav, s predavanj.

Dobljeni rezultati najinega postopka (v enakem vrstnem redu):

- http://cs.indiana.edu/
- http://cs.indiana.edu/
- http://cs.indiana.edu/People
- http://cs.indiana.edu/faq.html
- http://cs.indiana.edu/b
- http://cs.indiana.edu/index.html
- http://cs.indiana.edu/f̃il
- http://cs.indiana.edu/My%20File.htm
- http://cs.indiana.edu/People

3.2.7 Obdelava in shranjevanje binarnih datotek

Za prenos binarnih vsebin sva poskrbela ob koncu, pred zapisom datotek v bazo. Večino datotek je bilo mogoče prenesti s preprostim zahtevkom na podano spletno mesto (povezavo, ki smo jo pridobili z ekstrakcijo podatkov s spleten strani). Iz odziva zahtevka, v koliko je bil uspešen, sva nato zgolj prenesla vsebino (content) ali pa surovo vsebino (raw).
Pomembna izjema, ki jo je bilo potrebno upoštevati so bile povezave, ki niso bile spletne povezave. Za shranjevanje grafičnih in binarnih (dokumentov) vsebin, se že nekaj časa uporablja alternativa, ki vsebino zakodira v strnjen zapis (base64). Tako zapis pravzapra predstavlja že vsebino in ne zgolj naslov na vsebino. Zato sva takšne "povezave" obravnavala drugače, jih s pomočjo knjižnice base64 oziroma funkcije b64decode iz strnjenega zapisa pretvorila direktno v binarno vsebino in pripravila na shranjevanje.

3.2.8 Filtriranje povezav

Ob zaključku in pred vstavljanjem v frontier, je bilo veliko novih povezav potrebno prečistiti. Velik del postopka čiščenja povezav in normalizacije sva že opisala, zato izpostavljam zgolj še pomembne razlike. V izogib brezglavemu gledanju video vsebin headless browserja, in dejstvu, da takšne vsebine močno zakasnijo pajka, sva jih že vnaprej izpustila. Prav tako sva izpustila vse datoteke, ki jih nismo želeli pridobiti. Izognila sva se tudi stranem, katerih namen je zgolj preverjanje certifikatov, vpisovanja-izpisovanja etc. Zaradi učinkovitosti preverjanja, sva namesto zahtevkov veliko povezav zgolj preverjala s seznamom prepovedanih končnic. Nato pa sva zgolj za tiste, ki so prešle prvo sito, z zahtevki preverila: ali so veljavne / odzivne strani in ali je njihova vsebina tipa html. Neustrezne sva odstranila s seznama.

3.2.9 Uporaba zunanjih knjiznic

- requests
- selenium
- BeautifulSoup
- base64
- hashlib

- datasketch
- psycopg2
- urllib
- url_normalize
- json

4 Analiza

4.1 Učinkovitost spletnega pajka

4.2 Analiza zbranih povezav

Število vseh najdenih strani: 72507
Število vseh procesiranih: 64349
Število strani domene gov.si: 17293
Število 404 (zastarelih strani) domene gov.si: 6677
Število najdenih domen: 439
Maksimalna globina na frontierju: 5
Maksimalna obdelana globina: 4

5 Zaključek

6 Povezave

Github repozitorij: https://github.com/MaticBernik/web_crawler