Fakulteta za računalništvo in informatiko, Univerza v Ljubljani

Seminar 1 - Web crawler

Anja Hrovatič, Jan Fekonja in Nina Vehovec

30. Marec, 2019

V seminarski nalogi smo implementirali spletnega pajka s paralelnim izvajanjem. Za razvoj smo uporabili programski jezik Java ter knjižnico HTMLUnit za prenos in upodobitev strani in knjižnico JSoup za ekstrakcijo spletnih naslovov ter slik. Pridobljene vsebine smo shranili v PostgreSQL podatkovno bazo in pridobljene podatke vizualizirali s knjižnicama D3js in gephi.

1 Uvod

Cilj seminarske naloge je bil zgraditi spletnega pajka, ki obišče in prenese vsebino iz gov.si spletnih strani. Spletni pajek je zgrajen iz več delov, in sicer HTTP prenašalca (ang. downloader) in upodobitelja (ang. renderer), ekstrakcije podatkov, detektorja duplikatov, seznama URL-jev, ki jih želimo obiskati, in podatkovne baze. Na spodnji sliki 1 je prikazana arhitektura sistema. Implementirali smo ga v programskem jeziku Java.

2 Implementacija spletnega pajka

V seminarski nalogi smo implementirali spletnega pajka z vzporedno arhitekturo. Naša naloga je bila, da obiščemo štiri že vnaprej podane spletne strani (evem.gov.si, e-uprava.gov.si, podatki.gov.si, e-prostor.gov.si) ter si izberemo še pet dodatnih. Izbrali smo naslednje spletne strani: stopbirokraciji.gov.si, mnz.gov.si, mddsz.gov.si, mop.gov.si in mzi.gov.si.

Zgoraj naštete strani smo že na začetku dodali v vrsto (ang. queue) spletnih strani, ki smo jih želeli obiskati (frontier). Kot strategijo izbire naslednjega URL-ja smo implementirali iskanje v širino in s tem omogočili široko pokritje spletišča. Spletne strani smo prenesli in upodobili s pomočjo knjižnice HTMLUnit, ekstrakcijo podatkov pa izvedli z knjižnico JSoup. Detektirali smo tudi duplikatne spletne naslove ter duplikate vsebine prenesenih spletnih strani. Pri obiskovanju spletnih strani smo upoštevali tudi obstoječe robots.txt datoteke

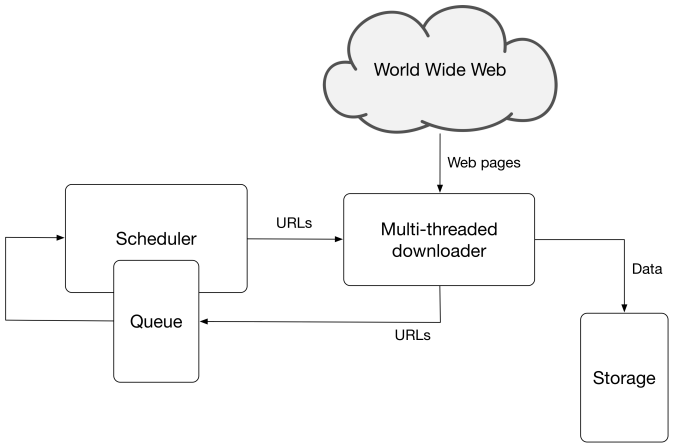

Figure 1: Arhitektura spletnega pajka, vir: [1]

ter pridobili URL naslove iz zemljevida spletne strani (ang. sitemap). Da bi spletni pajek hitreje in bolj učinkovito deloval, smo implementirali tudi niti, s čimer smo omogočili paralelizacijo oziroma vzporedno izvajanje večjega števila spletnih pajkov na enkrat. Preneseno vsebino smo shranjevali v PostgreSQL podatkovno bazo, pridobljene podatke pa nato vizualizirali s knjižnicama D3js in gephi ter tako prikazali statistiko pridobljenih vsebin. Bolj podrobno smo našteta področja opisali v naslednjih poglavjih.

Arhitekturo projekta smo zasnovali tako, da paket, imenovan DB, vsebuje razrede ter metode za manipulacijo s podatkovno bazo. Razred Scheduler.java vsebuje metode ter globalne strukture, med drugim tudi frontier, ki jih za svoje delovanje potrebuje razred WebCrawler.java. Slednji vsebuje vse komponente spletnega pajka za prenos, upodabljanje ter ekstrakcijo vsebin. Za zagon spletnega pajka je potrebno kreirati bazo s pomočjo skripte baza.sql, ki je prisotna na git repositoriju: `https://github.com/Anjdroid/web-crawler-ieps`. Zatem je potrebno spremeniti naslednje parametre v razredu Main.java:

- url = "jdbc:postgresql://localhost:5432/"
- dbName = <DBname>
- user = <PostgreSQLuser>
- password = <PostgreSQLpassword>
- numberOfThreads = <wantedNumberOfThreads>

Po nastavitvi zgornjih parametrov lahko projekt zaženemo v IntelliJ IDEA.

2.1 Ekstrakcija podatkov

V sklopu seminarske naloge smo implementirali ekstraktor podatkov s knjižnjico JSoup [2]. Iz HTML dokumentov, <a> začk, smo pridobili href linke. Tako

smo pridobili absolutne naslove spletnih strani, ki smo jih nato dodali v frontier, seznam še neobiskanih URL naslovov. Prenesene HTML dokumente smo tudi očistili, popravili morebitne napake v HTML dokumentu, s funkcijo clean(), ki jo vsebuje knjižnica JSoup, ter ji kot argument podali prenesen HTML dokument ter Whitelist [3]. S tem smo določili seznam dovoljenih HTML elementov in atributov in tako pridobili očiščen HTML dokument.

Poleg tega smo iz vsake spletne strani detektirali tudi slike, ki smo jih iz vsebine HTML dokumenta pridobili iz <img /> značk. Slike smo tudi prenesli in jih shranili v podatkovno bazo.

Zadnji del ekstrakcije podatkov je vključeval tudi pridobitev in prenos dokumentov tipa .pdf, .doc, .docx, .ppt, .pptx. Dokumente takšnih tipov smo pridobili tako, da smo ob prenosu strani preverili tip prenesene vsebine (ang. content type) in ga v primeru predhodno omenjenih tipov tudi prenesli in shranili v podatkovno bazo.

2.2 Detekcija duplikatov

Naslednji del spletnega pajka predstavlja detektor duplikatov. Detektor smo implementirali tako, da smo za vsak trenutni spletni naslov, ki smo ga vzeli iz frontier-ja, preverili, če takšen kanoničen URL že obstaja med predhodno obiskanimi spletnimi stranmi. Predhodno obiskane spletne strani smo shranjevali v globalno dostopno strukturo Set. Če je spletni naslov v setu obiskanih URL-jev že obstajal, smo spletno stran shranili kot duplikat.

Preverjali smo tudi duplikate glede na vsebino prenesenih spletnih strani. V tabelo "page" v podatkovni bazi smo dodali stolpec "hashcode," v katerega smo za vsako preneseno spletno stran shranili še zgoščeno vrednost vsebine HTML dokumenta. To smo generirali s pomočjo funkcije hashcode(), ki je del programskega jezika Java. Na ta način smo torej preverjali tudi duplikate spletnih strani glede na vsebino HTML dokumenta in tudi v takšnih primerih spletne strani shranili kot duplikat.

2.3 URL frontier

URL frontier, torej seznam spletnih strani, ki smo jih želeli obiskati, smo implementirali s podatkovno strukturo vrsta (ang. queue). Frontier je v programu globalno dostopna struktura. Implementirali smo BFS strategijo frontier-ja, torej iskanje v širino (ang. breadth-first search - BFS). S tem smo omogočili, da smo s spletnim pajkom zajeli čim več spletnih strani iz različnih področij. Naša implementacija strategije BFS je potekala tako, da smo na vsakem koraku iz vrste frontier-ja vzeli URL posamezne spletne strani in vsebino prenesli. S pomočjo ekstraktorja podatkov smo iz <a> značk pridobili prisotne URL-je v HTML dokumentu. Nato smo se s for zanko sprehodili čez vse elemente takšnega tipa in pridobili absolutne URL naslove, ki smo jih dodali v frontier. Vsak spletni naslov, ki smo ga želeli obiskati, smo sprva tudi pretvorili v kanonično obliko. To smo storili tako, da smo iz URL-ja prebrali protokol, informacijo o uporabniku, gostitelja, port, ime poti, poizvedbo ter referenco in iz tega ses-

tavili nov (kanoničen) URL.

Vsakič, ko smo iz frontierja vzeli novo spletno stran, smo preverili, ali njegova domena že obstaja v podatkovni bazi. Če je njegova domena že bila shranjena v bazi, je to pomenilo, da če obstaja robots.txt od te domene, potem je že shranjen v podatkovni bazi. V tem primeru smo le preverili, ali lahko spletno stran, ki smo jo pravkar vzeli iz frontierja, zajamemo. To smo storili s pomočju dveh HashMap-ov, v katera smo shranjevali dovoljene oz. nedovoljene povezave iz robots.txt datoteke. Vedno smo najprej preverili, ali je trenutna spletna stran med nedovoljenimi povezavami. Če je bila, potem smo še preverili, ali je med dovoljenimi povezavami. Če je spletna stran bila med nedovoljenimi in ni bila med dovoljenimi povezavami, potem je nismo smeli zajeti, sicer pa smo jo lahko.

Če pa domena trenutne spletne strani še ni bila shranjena v podatkovni bazi, potem pa smo najprej preverili, ali zanjo obstaja robots.txt datoteka. Če je obstajala, smo jo prebrali, shranili v bazo in med branjem poiskali vrstico z "User-agent: *" ter od te vrstice naprej do prve naslednje vrstice z "User-agent" ali pa do konca robots.txt shranili vse dovoljene povezave, nedovoljene povezave in zakasnitev iskanja, če je obstajala. Če smo kadarkoli med branjem robots.txt naleteli na vrstico "Sitemap", smo povezavo do sitemap-a shranili v seznam. Ko smo prebrali celoten robots.txt, smo vsak sitemap, ki smo ga shranili (lahko jih je bilo več ali pa ni obstajal), prebrali in shranili njegovo vsebino v podatkovno bazo ter na frontier dodali vse URL naslove, ki smo jih pridobili iz značk <loc>.

2.4 Paralelizacija in večkratni spletni pajki

Paralelizacijo ter zagon večih spletnih pajkov hkrati smo omogočili z uporabo niti (Java Threads). Razred WebCrawler.java, ki predstavlja instanco spletnega pajka, smo implementirali kot Runnable interface. Specificirano število niti smo zagnali s pomočjo ExecutorService okvirja [4], ki izvede zagon željenega števila niti in poskrbi za korektno izvajanje le-teh.

V seminarju smo implementirali tudi globalne strukture oziroma resurse, do katerih so imele dostop vse niti. Takšne strukture so bile vrsta frontier, set že obiskanih strani ter zgoščene tabele dovoljenih, prepovedanih spletnih strani, zakasnitve obiskovanja ter povezave starš - otrok. Slednje so opisovale povezavo med starševsko spletno stranjo ter njenimi vsebovanimi stranmi. Ker lahko oo globalnih struktur dostopa le ena nit na enkrat, smo izvedli sinhronizacijo z uporabo ključne besede synchronized pri metodah, ki so vračale te strukture. S tem smo zagotovili konsistentnost podatkov.

2.5 Podatkovna baza

V sklopu seminarja smo uporabili PostgreSQL bazo, v katero smo shranili pridobljene podatke o straneh, domenah, dokumentih, slikah, povezavah med obiskanimi stranmi, tipi strani ter tipi dokumentov. Osnovni bazi, ki smo jo prvotno dobili, smo v tabelo "page" dodali še polje "hashcode", v katerega smo shranili zgoščeno vrednost prenešene vsebine HTML dokumenta. To smo nato uporabili za preverjanje vsebinskih duplikatov spletnih strani.

2.6 Problemi pri implementaciji

Pri implementaciji spletnega pajka smo se spopadali z različnimi problemi, nekatere izmed njih smo tudi uspešno razrešili. Težave, na katere smo naleteli so: pridobivanje spletnih naslov z NULL ali prazno vrednostjo, pridobivanje spletnih strani, ki vračajo status (ang. response code) 404 ali 500, preusmeritve spletnih strani, pridobivanje URL-jev iz skript JavaScript, uporaba globalno dostopnih struktur (frontier, set že obiskanih spletnih strani) in njihova implementacija z nitmi.

Ob prvem resnejšem poskusu zagona spletnega pajka smo naleteli na težavo pridobivanja URL-jev z vrednostjo NULL. Na tej točki smo ugotovili, da spletni pajek v nedogled obiskuje spletne naslove s to vrednostjo ter tako ne pridobiva nobenih spletnih vsebin. Napako smo uspešno razrešili in dodali spletnemu pajku preverjanje spletnih naslovov s to vrednostjo.

Poleg tega so izvajanje našega pajka prekinile status kode tipa 404 in 500. V takšnem primeru pajek spletne strani ni mogel prenesti in je vračal napako. Tudi to težavo smo uspešno razrešili tako, da smo ob prenosu spletne strani preverjali, če le-ta vrača status takšnega tipa, obisk spletne strani preskočili in nadaljevali z obiskom nove. V nasprotnem primeru pa smo stran prenesli in iz nje pridobili vsebine.

Spletnemu pajku smo morali omogočiti tudi sledenjem preusmeritvam (ang. redirect). Namreč, pogost pojav je bil, da je spletni pajek poteboval preusmeritev, da je lahko prebral ustrezno robots.txt datoteko, sicer v podatkovno bazo ni shranil ničesal, ali pa je shranil napačno vsebino. Pri robots.txt smo še naleteli na tri težave. Prva je bila, da robots.txt ni vedno vseboval informacije o zakasnitvi iskanja. To smo rešili tako, da smo se odločili, da domeni, katere robots.txt ne vsebuje te informacije, dodelimo 4 sekundno zakasnitev. Drugi problem, s katerim smo se tu srečali, je bil, da marsikatera spletna stran ne sledi standardom glede robots.txt in vsebuje "User-Agent" namesto "User-agent", zato smo se odločili, da preverjamo oboje. Tretji težava, ki se je pojavila na tem mestu, pa je bila, da nekateri robots.txt vsebujejo več kot en sitemap, zato smo odločili, da v bazo shranjujemo vsebine vseh sitemap-ov, ki jih ima domena.

Implementirali smo tudi paralelizacijo z uporabo niti. Za uspešno delovanje večih spletnih pajkov na enkrat, smo morali globalno dostopne strukture, skupne resurse, sinhronizirati. S tem je lahko le ena nit na enkrat dostopala do skupnih resursov, npr. do frontier-ja, kar pa je precej upočasnilo hitrost obiskovanja spletnih strani, saj so se niti, kljub velikemu številu, med seboj čakale.

Največji problem, ki smo ga imeli pri tej seminarski nalogi, pa je bil, da vsi člani naše skupine bivamo v študentskem domu, kjer ni zagotovljena stabilna internetna povezava in zato smo imeli veliko težav, ko smo našega spletnega pajka zaganjali. Večkrat se nam je namreč pripetila izguba internetne povezave po nekaj urnem delovanju našega pajka in posledično smo izgubili vse do tedaj zajeta podatke ter smo morali spletnega pajka zagnati od začetka. V tem je tudi glavni razlog, da je število naših pridobljenih spletnih strani veliko manjše od zahtevanega števila.

3 Statistika

V sklopu tega poglavja smo vizualizirali splošno statistiko pridobljenih spletnih strani in podatkov. Statistiko podanih spletnih strani:

- evem.gov.si, e-uprava.gov.si, podatki.gov.si, e-prostor.gov.si,

in statistiko izbranih spletnih strani:

- stopbirokraciji.gov.si, mnz.gov.si, mddsz.gov.si, mop.gov.si, mzi.gov.si.

Na spodnjih slikah 2 in 3 je prikazano število pridobljenih spletnih vsebin (spletne strani, domene, dokumenti in slike). Spletne vsebine pridobljene iz vnaprej podanih spletnih strani so označene z (1), vsebine z izbranih pa z (2). Skupno število spletnih strani, ki smo jih uspeli pridobiti je 11765, 460 domen, 591 dokumentov ter 19673 slik. Na slikah 4 in 5 so prikazane povezave (link) med spletnimi stranmi.

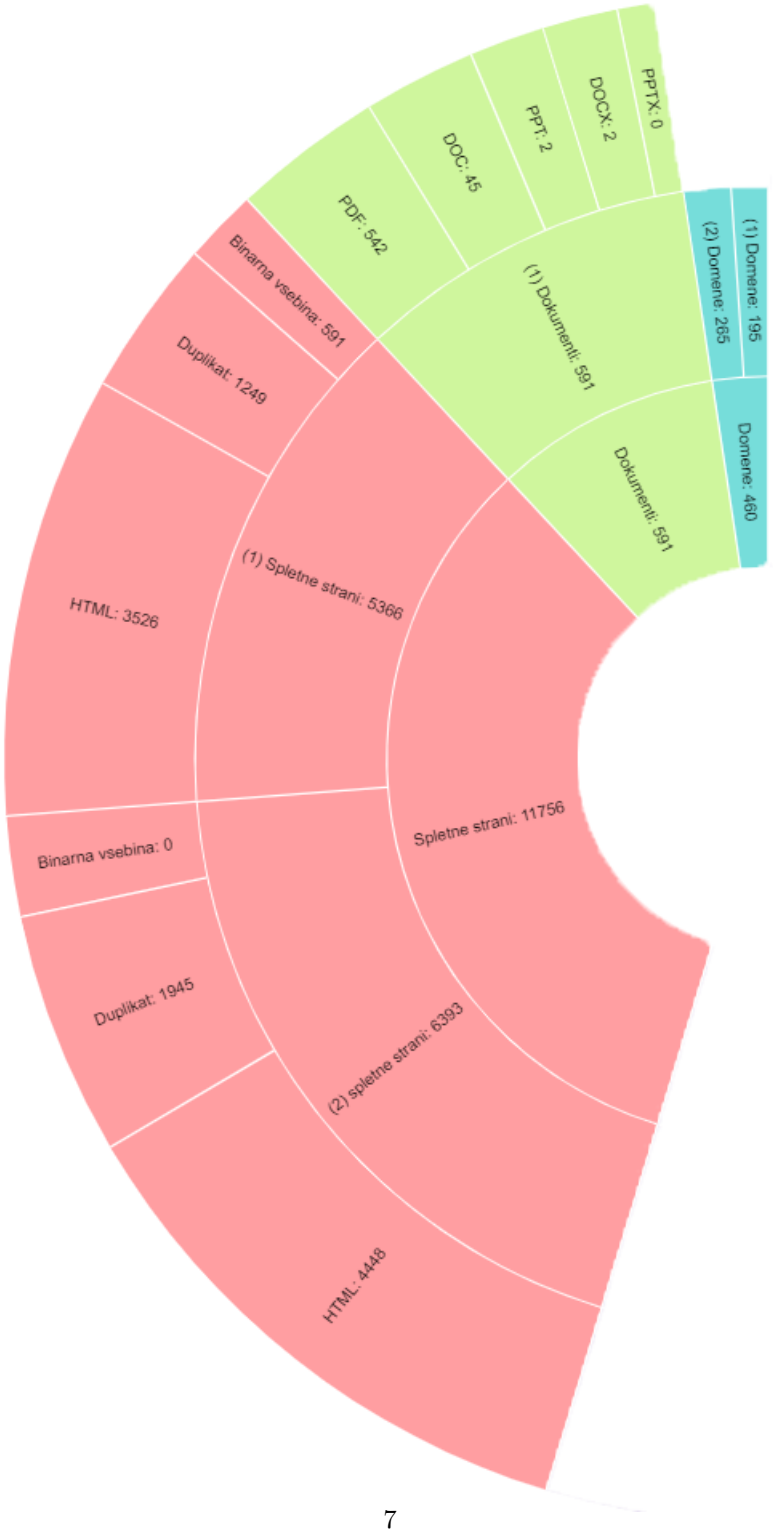

Figure 2: Vizualizacija števila pridobljenih spletnih strani, dokumentov, domen.

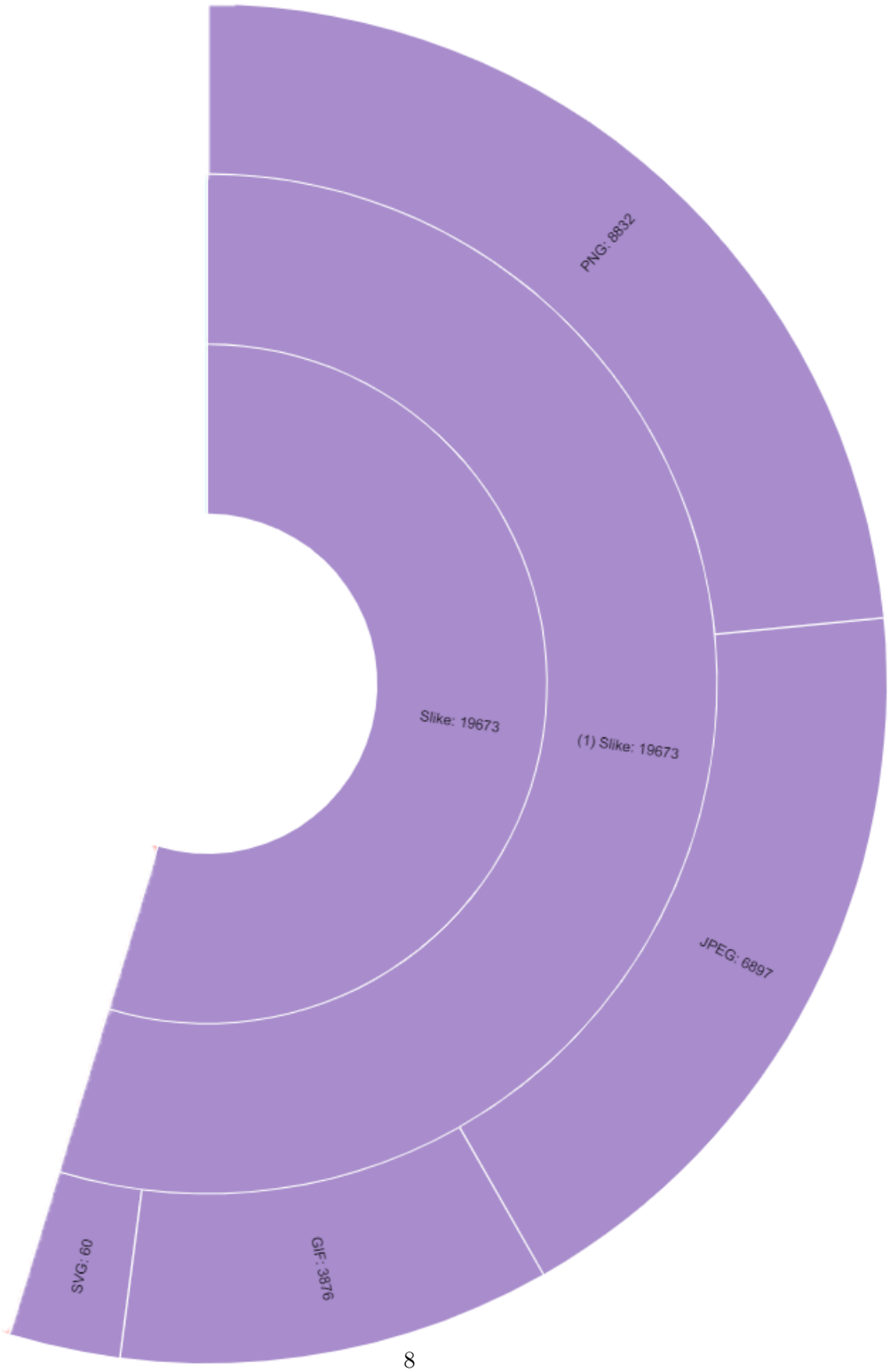

Figure 3: Vizualizacija števila pridobljenih slik s spletnih strani.

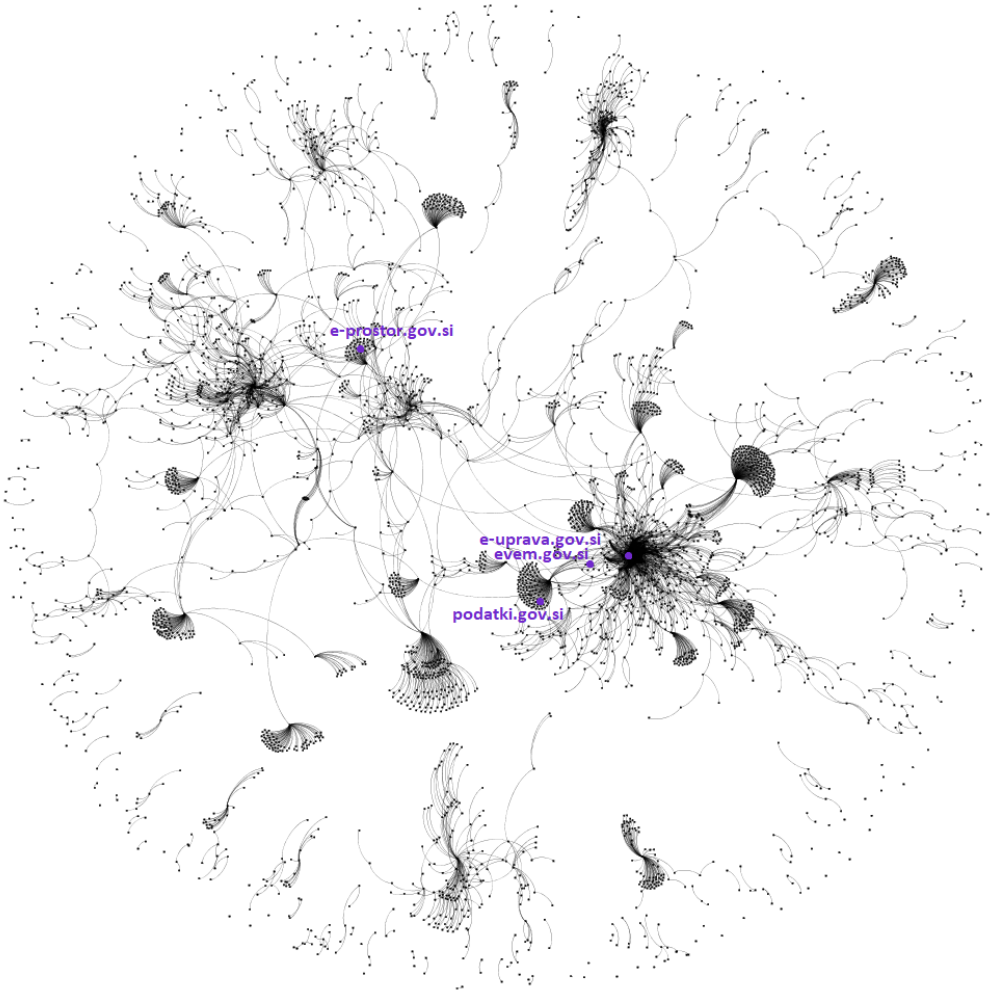

Figure 4: Vizualizacija števila povezav spletnih strani (1).

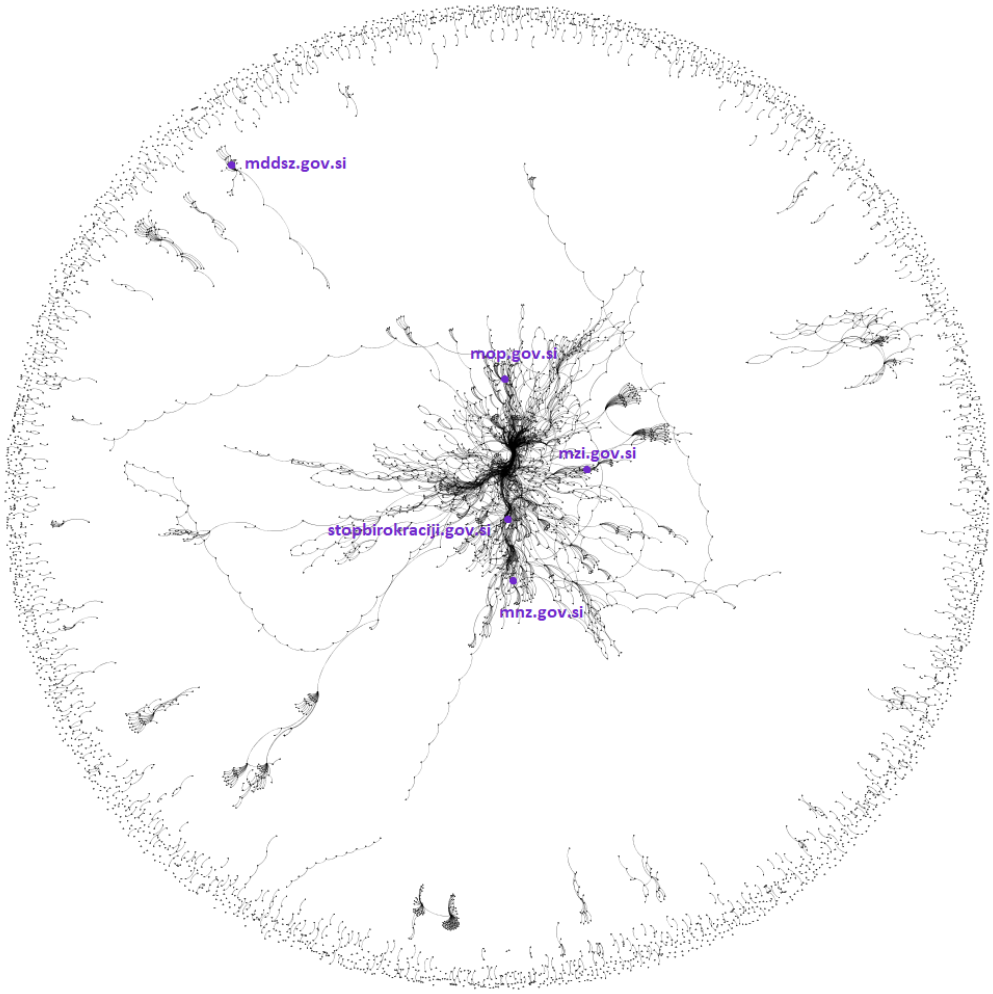

Figure 5: Vizualizacija števila povezav spletnih strani (2).

4 Zaključek

V sklopu seminarske naloge smo implementirali spletnega pajka s paralelnim izvajanjem v programskem jeziku Java. Na poti do končne implementacije smo se spopadli s številnimi problemi in številne tudi odpravili. Vse skupaj smo Uspeli pridobiti le 11765 spletnih strani, 460 domen, 591 dokumentov ter 19673 slik. Prva dva koraka k izboljšavi pridobljenih rezultatov bi bila, optimizacija delovanja spletnega pajka in implementacija ekstrakcije URL naslovov iz onclick JavaScript dogodkov. S tem bi pohitrili delovanje ter predvsem pridobili večje število URL naslovov iz prenesenih spletnih strani. Velik vpliv na količino pridobljenih vsebin pa je imela tudi nezanesljiva internetna povezava, s katero smo imeli težave.

Literatura

[1] Slavko Žitnik, "Programming assignment 1," [Dostopano: 30. 3. 2019]. [Online]. Available: http://zitnik.si/teaching/wier/PA1.html

[2] Jonathan Hedley, "jsoup: Java html parser," [Dostopano: 2. 4. 2019]. [Online]. Available: https://jsoup.org/

[3] JSoup.org, "Class whitelist," [Dostopano: 1. 4. 2019]. [Online]. Available: https://jsoup.org/apidocs/org/jsoup/safety/Whitelist.html

[4] Oracle, "Interface executorservice," [Dostopano: 2. 4. 2019]. [Online]. Available: https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ExecutorService.html