Naloga 1: Spletni pajek

Blaž Marolt, Rok Šolar, Anže Veršnik

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko

1 Uvod

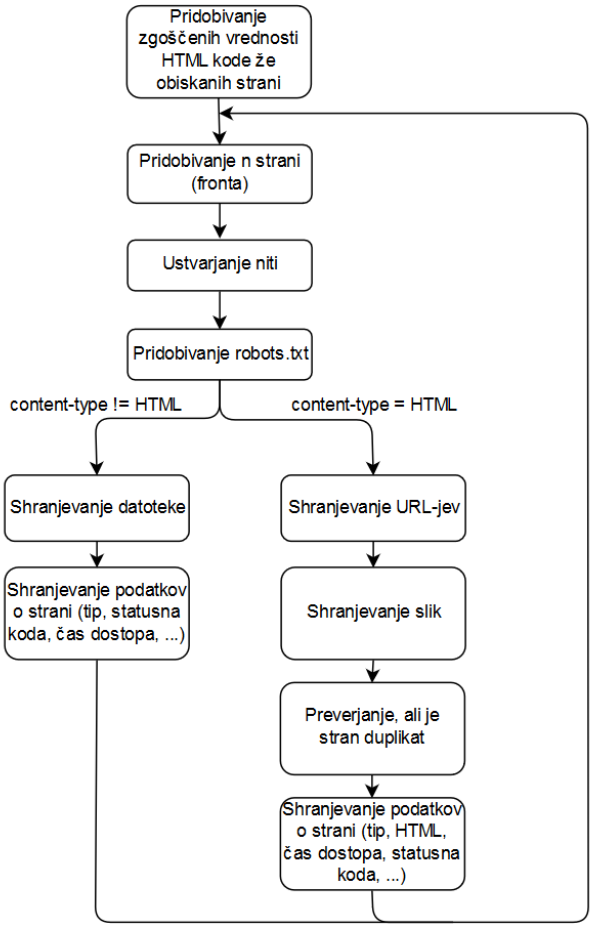
Prva domača naloga je zahtevala implementacijo spletnega pajka, ki išče spletne strani v domeni `.gov.si`. Podane so nam bile štiri začetne domene, shema in skripta za postavitev podatkovne baze ter približna shema končne arhitekture pajka. V tem delu bomo najprej predstavili implementacijo spletnega pajka, z vsemi ključnimi podrobnostmi in težavami, s katerimi smo se srečali. Potem bomo prikazali rezultate implementiranega pajka, statistične podatke ter na koncu še vizualizacijo dobljenih povezav.

2 Implementacija

Za implementacijo spletnega pajka smo izbrani programski jezik Python. Na sliki 1 je prikazan diagram poteka izvajanja pajka. V nadaljevanju bomo podrobneje opisali ključne dele implementiranega pajka.

2.1 Zgoščena vrednost HTML kode

Vse povezave, ki jih spletni pajek dobi na določeni strani se shranjujejo v podatkovno bazo v tabelo `page`. Tabela `page` ima na stolpcu url že omejitev unikatnosti, kar pomeni da v tabelo ni mogoče shraniti dveh spletnih strani z istimi povezavami. Vseeno pa je možno pridobiti strani z isto HTML kodo in različno povezavo. Zaradi tega smo za izboljšanje delovanja uvedli preverjanje zgoščene vrednosti z uporabo algoritma MD-5, ki vrednost zakodira v 128 bitov.
Prvi korak spletnega pajka je, da iz podatkovne baze prenese zgoščene vrednosti HTLM kod spletnih strani in shrani lokalno,v nnožico. Za vsako spletna stran nato, preden posodobimo njen status v podatkovni bazi, izračunamo zgoščeno vrednost HTML kode ter preverimo ali jo množica že vsebuje. Če je vrednost že vsebovana spletno stran v podatkovni bazi označimo kot duplikat drugače pa kot navaden HTML. V obeh primerih dodamo v podatkovno bazo izračunano zgoščeno vrednost.



Slika 1. Shema delovanja pajka.

2.2 Pridobivanje N strani

Spletni pajek vsako iteracijo pridobi novo fronto (spletne strani katere bo obdeloval v trenutni iteraciji). Pridobivanje podatkov poteka na dva različna načina zaradi različnih načinov delovanja pajka.
Pri prvem načinu spletni pajek pridobiva povezave samo na osnovne štiri domene. Ta omejitev je implementirana pri zapisu v tabelo `page`, kjer filtriramo pridobljene povezave. Podatkovna baza tako vsebuje le spletne strani iz podanih osnovnih domen. Pri tem načinu fronto naredimo s preprostim SELECT stavkom, ki prenese vse podatke o spletni strani ter domeni, podatke vrne po naraščajočem indeksu ter z omejitvijo vrstic na število niti. Vsaka nit tako dobi svojo spletno stran.
Pri drugem načinu delovanja pa nimamo omejitve na domene in tako želimo čim večjo hitrost. Ker je potrebno med zahtevki za spletno stran iz iste domene počakati najmanj 4 sekunde smo želeli optimizirati čakanje, z izboljšanim pridobivanjem fronte. Fronto smo pridobili z ukazom SQL, ki ga vidimo na sliki 2. SQL stavek pridobi n spletnih strani, ki so vse iz različnih domen vendar so urejene naraščajoče po indeksu. Spletni pajek tako v iteraciji preskakuje spletne strani iz istih domen. S to izboljšavo nam ni treba več skrbeti za zakasnitev med nitmi, ki bi dobile spletne strani iz iste domene.

```
def getN_frontiers(conn, n):
    cur = conn.cursor()
    '''sql = SELECT t.id, t.site_id, t.page_type_code, t.url, t.html_content, t.http_status_code
                ,t.accessed_time,site.domain, site.robots_content, site.sitemap_content
                FROM (
                    SELECT DISTINCT ON (page.site_id) *
                    FROM crawldb.page
                    where page.page_type_code='FRONTIER'
                    ORDER BY page.site_id, page.id ASC
                ) t
            JOIN crawldb.site site ON (t.site_id = site.id)
            ORDER BY t.id ASC
            limit %s'''
```

Slika 2. Pridobivanje nove fronte.

2.3 Niti

Pri delu z nitmi se vedno pojavi problem souporabe določenih virov. Z našo implementacijo smo se poskušali tej težavi popolnoma izogniti in tako preprečiti napačno delovanje programa, kot tudi čimbolj minimizirati zaklepanje virov, ki upočasnijo delovanje programa samega. Ugotovili smo, da WebDriver Selenium, ki smo ga uporabili v svoji implementaciji ni thread-safe [SeleniumHQ]. Vendar pa se lahko temu problemu izognemo tako, da ima vsaka nit svojo lastno

instanco. Tako zaobidemo ta problem. Niti smo zato implementirali tako, da vsaka deluje neodvisno od druge.
Osnovno delovanje implementacije izgleda tako, da se s fronte pobere število strani enako maksimalnemu številu niti. Pri tem upoštevamo zaporedje dodajanja strani v fronto in s tem tudi ohranjamo strukturo ter zagotovimo iskanje v širino. V primeru, da je število strani, ki se še nahajajo v fronti manjše od maskimalnega števila niti pa jih enostavno generiramo manj. Ob koncu vsake iteracije se niti počakajo med sabo. Potem pa se postopek ponovi. S tem načinom implementacije smo se izognili negativnim aspektom, ki jih lahko niti povzročijo s svojim delovanjem.

2.4 Obdelava strani

robots.txt, crawl-delay in osnovni parametri strani. V prvem koraku obdelave strani pajek preveri, ali je za podano domeno že shranjena datoteka robots.txt. Če ni, jo pridobi in poišče še potencialna kazala spletne strani (ang. sitemap). Nato shrani dobljene podatke za trenutno domeno v podatkovno bazo in ustrezno zamakne nadaljnje korake (upoštevanje crawl-delay). V kolikor zamik ni podan v datoteki robots.txt, se izvajanje zamakne za privzete štiri sekunde.
V naslednjem koraku pajek pridobi statusno kodo in vrsto vsebine (ang. content-type). To poskusi najprej z zahtevkom HTTP head, v kolikor stran tega ne dovoli, pa poskusi še s HTTP head (po ustreznem zamiku). V kolikor je stran tipa HTML se nato zažene gonilnik Chrome (knjižnica `Selenium`) in nadaljujemo z obdelavo.

Pridobivanje URL-jev. Če je vsebina strani tipa HTML, se v naslednjem koraku iz strani pridobijo URL-ji. Najprej pridobimo vse povezave, ki jih vsebuje morebitno kazalo spletne strani. Pri tem za razčlenjevanje datoteke XML uporabimo knjižnico `BeautifulSoup`.
Nato s pomočjo regularnih izrazov poiščemo vse povezave v Javascript kodi. To naredimo v dveh korakih. Najprej poiščemo morebitne povezave v gumbih z atributom `onclick`, nato pa še v preostali Javascript kodi, kar je prikazano na sliki 3. Pri tem upoštevamo, da se povezave lahko nahajajo za ukazoma `location.href="povezava"` in `document.location="povezava"`.
Na koncu pridobimo še povezave znotraj HTML elementa `a` z atributom `href`. Pri tem smo občasno naleteli do napake `stale element reference`, kar smo do neke mere rešili z večkratnim poskušanjem (slika 4).

Kanonizacija URL-jev.

1. Če povezava vsebuje besedilo `"javascript:"` ga preskočimo (včasih atributi `href` vsebujejo kodo oblike `javascript:void(0)`, kar je neveljavno.

```
# Linki v js (gumbi) s Selenium
js_code = driver.find_elements_by_xpath("//button[@onclick]")
for code in js_code:
    match1 = regex.findall(r'location\.href\s*=\s*\"([^"]+)\"', code.get_attribute("onclick"))
    match2 = regex.findall(r'document\.location\s*=\s*\"([^"]+)\"', code.get_attribute("onclick"))
    links = parse_links(match1 + match2, links, robots)

# Linki v js (koda) s Selenium
js_code = driver.find_elements_by_xpath("//script")
for code in js_code:
    match1 = regex.findall(r'location\.href\s*=\s*\"([^"]+)\"', code.get_attribute("innerText"))
    match2 = regex.findall(r'document\.location\s*=\s*\"([^"]+)\"', code.get_attribute("innerText"))
    links = parse_links(match1 + match2, links, robots)
```

Slika 3. Pridobivanje povezav iz kode Javascript.

```
# a...href linki s Selenium
elems = driver.find_elements_by_xpath("//a[@href]")
urls = []
for elem in elems:
    staleElement = True
    retries = 0
    while staleElement:
        retries+=1
        if retries > 100:
            break
        try:
            urls.append(elem.get_attribute("href"))
            staleElement = False
        except selenium.common.exceptions.StaleElementReferenceException as e:
            staleElement = True
```

Slika 4. Pridobivanje povezav iz elementa a z atributom href.

2. Obdelava s knjižnico `urldefrag` in `urlcanon`. Te dve knjižnici odstranita privzete številke vrat, odvečne "/", povezave na dele strani (npr. example.com/test.html#abc), relativne poti (example.com/a/../b/), ustrezno zakodirata prepovedane znake in uredita mešano uporabo velikih in malih črk v domeni.
3. Odstranjevanje pripetih "/". Izkazalo se je, da je najbolj učinkovito, če vsem povezavam odstranimo ta končni znak.
4. Odstranjevanje pripetih `/index.html` in `/index.php`.
5. Odstranjevanje `http://` in `https://`. Ko obdelujemo stran, vedno pripnemo na začetek povezave `http://` in predpostavljamo, da bo stran izvedla ustrezne preusmeritve.
6. Preverimo ali je znotraj prave domene in ali datoteka robots dovoljuje povezavo (uporaba knjižnice `robotexclusionrulesparser`). To knjižnico smo izbrali zato, ker sprejme že prebrano besedilo datoteke robots.txt. Ostale knjižnice so omogočale zgolj razčlenjevanje datoteke, prebrane iz spleta.

Pridobivanje datotek in slik. V kolikor je tip strani PDF, DOC, DOCX, PPT ali PPTX, prenesemo 10 MB datoteke. To storimo tako, da beremo datoteko kot tok, po 1024 B. Ko presežemo 10 MB ali pridemo do konca datoteke, tok zapremo in shranimo datoteko v podatkovno bazo.
Slike na spletni iščemo z pomočjo knjižnice `Selenium` z iskanjem besede img in src. Po pridobitvi povezave slike jo prenesemo kot ostale tipe datotek, pregledamo pa tudi končnico slike ter shranimo v podatkovno bazo. V podatkovni bazi smo povečali polje za ime datoteke na 10 000 znakov.

Specifike implementacije.

1. Za vsa branja (razen branja HTML kode strani za iskanje slik in povezav ter za izvajanje Javascript kode) uporabljamo knjižnico `requests`.
2. Zahtevki s knjižnico `requests` dovoljujejo preusmeritve in imajo časovno omejitev 10 s.
3. Vsaka nit odpre en gonilnik Chrome (Selenium), ki ga potem po potrebi večkrat uporabi in na koncu zapre.
4. Vsaka nit dobi kot argument svojo povezavo na bazo, ki se po zaključku izvajanja niti zapre.
5. Datoteko robots.txt shranjujemo samo v primeru, če je status branja datoteke enak 200 (da se izognemo shranjevanju sporočil, kot so 404 - not found).
6. V kolikor datoteka robots.txt vsebuje več kazal strani (sitemaps), jih združi v eno.
7. V kolikor pride do napake ali preteče časovna omejitev, se tip strani spremeni v `TIMEOUT` (to polje smo dodali v tabelo `page_type`).

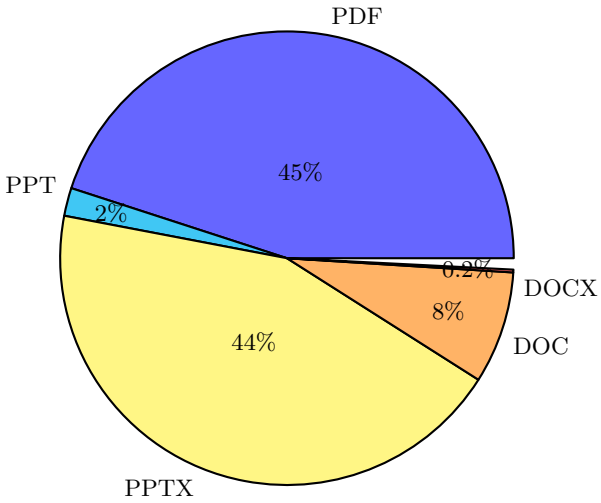
3 Statistika

3.1 Začetnih dve domeni in poddomene

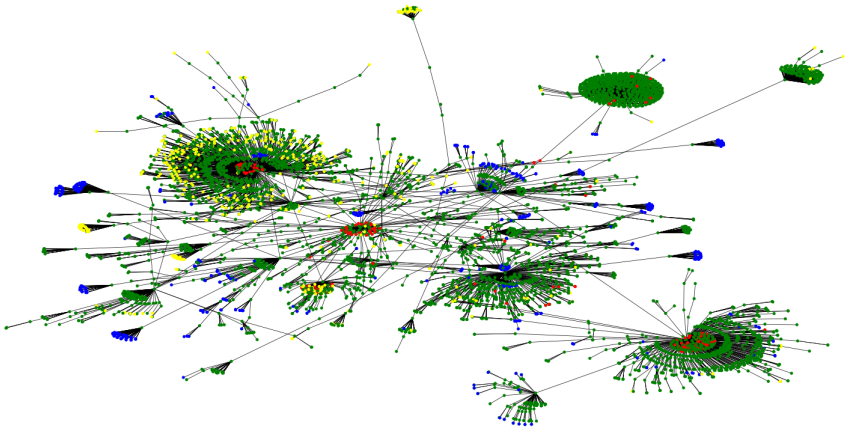
Najprej smo zagnali pajka na domenah `evem.gov.si` ter `e-prostor.gov.si` in iskali strani iz teh dveh domen in vseh najdenih poddomen. Upoštevali smo torej še `www.projekt.e-prostor.gov.si`, `www.e-prostor.gov.si` in `www.evem.gov.si`. V tabeli 1 so prikazani rezultati iskanja, na sliki 5 je prikazana porazdelitev binarnih datotek po vrsti, na sliki 6 vizualizacija poteka iskanja povezav in na sliki 7 časovni potek pridobivanja povezav.

	Vrednosti
Število DOMEN	5
Število HTML strani	5736
Število TIMEOUT strani (napake)	202
Število DUPLIKATOV	511
Število BINARNIH strani	744
SKUPAJ strani	7193
Število SLIK	30454
Povprečno število SLIK na stran	4,78
Število PDF datotek	335
Število DOC datotek	62
Število DOCX datotek	2
Število PPT datotek	14
Število PPTX datotek	331

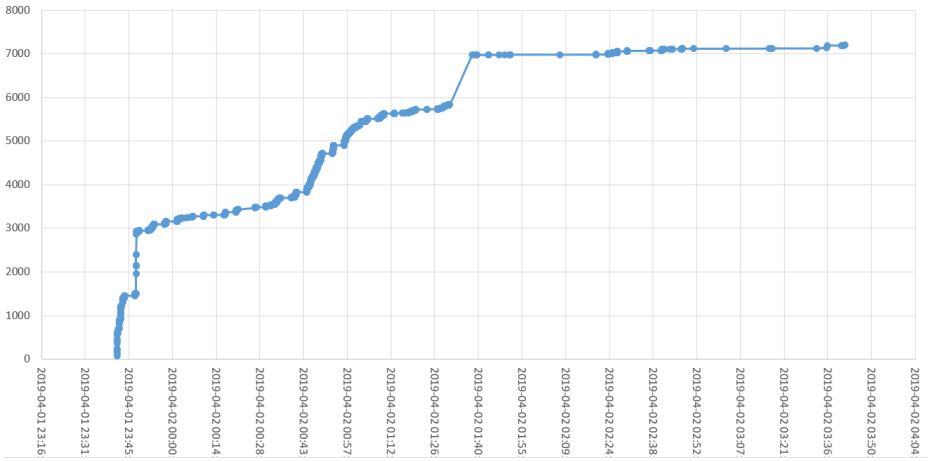
Tabela 1. Podatki o začetnih dveh domenah.



Slika 5. Graf porazdelitve binarnih datotek na začetnih dveh domenah.



Slika 6. Graf povezav za začetni dve domeni (zelena = HTML, modra = BINARNA, rumena = DUPLIKAT, rdeča = NAPAKA).



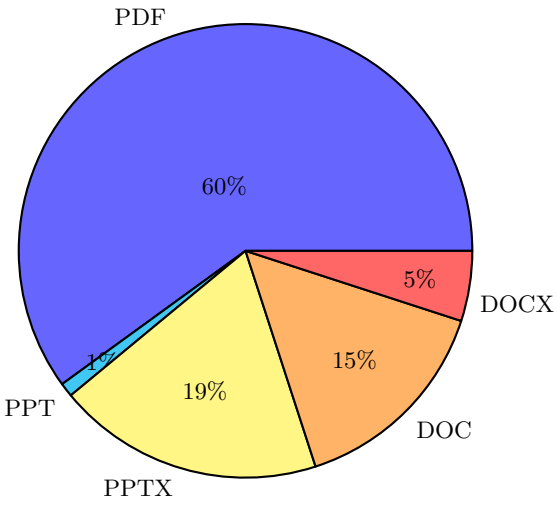
Slika 7. Časovni potek iskanja na začetnih dveh domenah.

3.2 Celotna domena gov.si

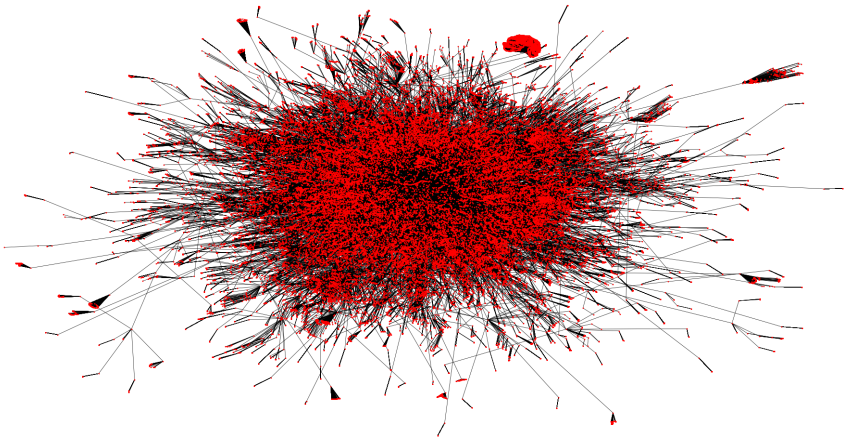
V drugem koraku smo iskanje razširili na celotno domeno `gov.si`, brez shranevanja binarnih datotek in slik. V tabeli 2 so prikazani rezultati iskanja, na sliki 8 je prikazana porazdelitev binarnih datotek po vrsti, na sliki 9 in 10 pa vizualizacija poteka iskanja povezav.

	Vrednosti
Število DOMEN	369
Število HTML strani	94505
Število TIMEOUT strani (napake)	1253
Število DUPLIKATOV	3147
Število BINARNIH strani	44640
Ostanek FRONTE	239022
SKUPAJ strani	382567
Število SLIK	1494665
Povprečno število SLIK na stran	15,3
Število PDF datotek	27015
Število DOC datotek	6944
Število DOCX datotek	2099
Število PPT datotek	244
Število PPTX datotek	8338

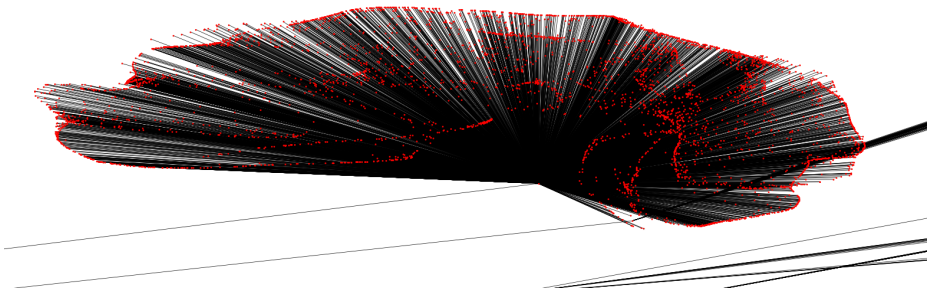
Tabela 2. Podatki o domeni gov.si.



Slika 8. Graf porazdelitve binarnih datotek na domenah gov.si.



Slika 9. Graf povezav na celotni domeni gov.si.



Slika 10. Desni zgornji kot približan.

4 Zaključek

V naši implementaciji smo uspeli uspešno realizirati vse zadane naloge. Zagotovili smo delovanje z večimi nitmi ter iskanje v širino. Aplikacija s pomočjo primerjanje povezav ter zgoščenih vrednosti kode HTML uspešno prepozna strani, ki so duplikati. Prav tako ustrezno upošteva datoteko robots.txt, v kolikor ta obstaja. Med testiranjem je naša aplikacija delovala tudi robustno, torej se med samim izvajanjem ni ustavila zaradi napak na iskanih strežnikih. Nadaljnje delo bi lahko vsebovalo primerjavo z drugimi spletnimi pajki, na področju hitrosti in najdenih povezav ter slik v spletni strani, saj ne moremo trditi, da je implementiran pajek izluščil vse zahtevane podatke iz obiskanih spletnih strani.

Literatura

[SeleniumHQ] SeleniumHQ. FAQ Selenium, year = 2019, url = https://github.com/SeleniumHQ/selenium/wiki/Frequently-Asked-Questions#q-is-webdriver-thread-safe, urldate = 2019-03-15.