Seminarska naloga 1 - Spletni pajek

Katjuša Jaklič, Luka Kuzman, Tjaša Mlakar
Fakulteta za računalništvo in informatiko, Univerza v Ljubljani
Marec 2022

1. Uvod

Spletni pajek, oziroma angleško web crawler, je avtomatiziran program, ki preiskuje in pregleduje spletne strani na svetovnem spletu. Njegova naloga je iskanje in izločitev podatkov iz spletnih strani, ki se shranjujejo v podatkovno bazo in predstavljajo veliko množico podatkov, ki se kasneje lahko uporabi na različne načine, odvisno od našega cilja. Cilj naše seminarske naloge je implementacija spletnega pajka, ki preiskuje spletne strani na domeni gov.si, ob tem pa deluje večnitno, išče v širino in podatke pošilja v bazo.

2. Implementacija

Spletni pajek je implementiran v programskem jeziku Python, pridobljene podatke pa hrani v podatkovni bazi PostgreSQL, ki je bila delno implementirana vnaprej. Celoten projekt je sestavljen iz več logičnih delov, ki so namenjeni posameznemu delu implementacije. Projekt sestoji iz delov, med katerimi so pomembnejši:

- main.py (ustvarjanje določenega števila niti in njihovo zaganjanje),
- crawler.py (prenos strani, dodajanje v frontier, preverjanje duplikatov, branje datotek robots.txt in sitemap.xml podatkov),
- database.py (dostop, vstavljanje in iskanje v podatkovni bazi),
- domain.py (prebiranje sheme in domene z linka, pretvorbe linkov),
- link_handler.py (prebiranje linkov iz HTML stringa) in
- thread.py (objekt niti, preiskovanje strani).

2.1. Inicializacija pajka

Inicializacija pajka poteka znotraj datoteke “main.py.” Tu se kreira želeno število niti pajka, le-te pa so opredeljene znotraj datoteke “thread.py”. Za implementacijo vzporednih niti smo uporabili knjižnico threading. Vhodni parameteri se nahajajo v datoteki “project_properties.py” in so lahko po potrebi spremenjeni. Program se izvaja, dokler v frontier-ju ni več strani za obdelavo ali dokler ga ne ustavimo sami. Po ponovnem zagonu nadaljuje s stranmi, ki se nahajajo v frontier tabeli.

2.2. Frontier

Frontier je implementiran kot ločena tabela crawldb.frontier znotraj PostgreSQL podatkovne baze. Ob inicializaciji so v njej že pripravljeni t.i. “seed url” naslovi, s pomočjo katerih posamezne niti pričnejo postopek iskanja in ekstrakcije relevantnih informacij ter shranjevanje url naslovov strani, ki bodo obdelane v prihodnje. Pred dodajanjem url naslova v frontier preverimo, ali je url naslov v primerni domeni, hkrati pa ima frontier vpeljano lastnost edinstvenosti, ki zagotavlja, da je vsak url naslov v frontierju unikaten, njegovi duplikati pa se zavržejo. V primeru, da želi več niti sočasno klicati funkcijo, ki iz frontier-ja pridobi naslednjo stran za obdelavo, si pomagamo z razredom knjižnice threading, imenovanim Lock.

2.3. Detekcija duplikatov

Detekcija duplikatov je implementirana znotraj datoteke ”crawler.py”. HTML se posreduje v funkcijo check_duplicates(), ki izračuna hash spletne strani. Hash-i so shranjeni v tabeli hash-ov, ki prav tako vsebuje povezave na tabelo page. Vsakič, ko pridemo do nove spletne strani, ji s funkcijo izračunamo hash in nato primerjamo izračunanega z vsemi, ki so shranjeni v tabeli. Če se dva hash-a ujemata, se URL podvojenega označi kot duplikat.

2.4. Obdelava strani

Preden se program loti pridobivanja vsebine strani, katero pajek trenutno pregleduje, se preveri, ali se do strani lahko dostopa. V primeru, da stran vrne status kodo, ki nakazuje napako, se URL ne shrani v frontier, ampak se le zapiše v page tabelo. V primeru, da je stran dostopna, ampak pri izvajanju nekje v kodi pride do napake (na primer indeks kaže izven seznama), pa se URL strani prestavi na konec frontierja. Pri pridobivanju strani smo pozorni na upoštevanje etike, kamor spada to, da do posameznega IP naslova in domene ne dostopamo prevečkrat. V ta namen je ustvarjena posebna tabela, kjer hranimo unikatno kombinacijo IPja in domene ter čas dostopa. Ob dostopanju do strani izluščimo IP in domeno ter ju primerjamo z vsemi zapisi v tabeli z istim IPjem ali isto domeno in upoštevamo timeout, ki smo ga navedli. S tem se izognemu

preobremenitvi strežnika. Dovoljenja določene domene preberemo iz datoteke robots.txt, vsebino sitemap (v primeru da obstaja) pa iz sitemap.xml oziroma imena, ki je podan v robots.txt. Povezave, prisotne v sitemap.xml, dodamo v frontier. Vsebino spletne strani pridobimo s pomočjo Selenium Webdriverja. Ko dobimo vsebino strani, se le to pošlje čez funkcije, ki so implementirane v datoteki "link_handler.py". Te pregledujejo vsebino in v njej iščejo datoteke, slike in url povezave, le-te naknadno preoblikujejo v njihovo absolutno in kanonizirano obliko. Ko se pregledovanje strani zaključi, se podatki o datotekah in povezavah shranijo v podatkovno bazo. Pri tem smo v tabelo link shranjevali le povezave duplikatov na strani, katerih duplikati so, v tabeli robots_content in sitemap_content pa HTML vsebino strani.

3. Rezultati in statistika

Implementirani program za preiskovanje spletnih strani na domeni gov.si se je izvajal približno 36 ur, pri tem pa je preiskal 51.000 spletnih strani, kar je povprečno 1416 strani na uro. Program se je izvajal v 4 nitih, kar je na začetku povzročalo nekaj težav in smo morali program večkrat popraviti. Rezultati in statistika preiskovanja strani so prikazani v tabeli 1 in 2.

TABLE 1: Tabela statistik celotnega crawldb

	Količina	Povp. na domeno	Povp. na stran
Domene	201	x	x
Spletne strani	51.097	254,21	x
Duplikati	29.870	148,61	0,58
PDF	17.899	89,05	0,35
DOC, DOCX	9456	47,04	0,19
PPT, PPTX	150	0,75	0,003
Slike	28.653	142,55	0,56

TABLE 2: Tabela statistik začetnih "seed" strani

	Količina	Povp. na domeno	Povp. na stran
Domene	4	x	x
Spletne strani	13.193	3.298,25	x
Duplikati	6.487	1.621,75	0,49
PDF	4.145	1.036,25	0,31
DOC, DOCX	4.161	1.040,25	0,32
PPT, PPTX	0	0	0
Slike	3.421	855,25	0,26

4. Vizualizacija

Po končanem preiskovanju strani z implementiranim spletnim pajkom, smo dobljene podatke vizualizirani na posamezniku bolj razumljiv način. Vizualizacijo smo pripravili s pomočjo orodja Gephi, kjer smo uporabili 10 000 URL povezav. Vozlišča v grafu predstavljajo spletne strani, povezave med temi vozlišči pa so povezave duplikatov. Vizualizacija je prikazana na sliki 1 (zaradi velikosti smo prikazali le del pridobljenega omrežja).

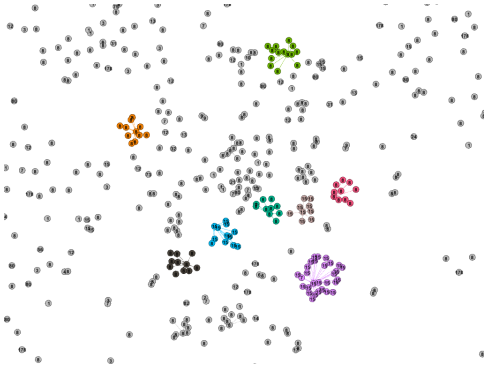

Figure 1: Vizualizacija povezav, kjer prikažemo iz katerih domen v katere sežejo duplikati

5. Zaključek

Seminarska naloga se nam je zdela zanimiva in dobro izvedena, smo pa med samim izvajanjem spletnega pajka naleteli na napake pri poizvedbah, in sicer zaradi hkratnega dostopa do podatkovne baze, ki so si jo delile vse 4 niti. Dvakrat je prišlo do konflikta, v katerem sta dve niti hkrati vstavljali v podatkovno bazo. Tedaj se je le-ta porušila. Problem smo rešili tako, da smo dodali lock, ki je zaprl delovanje ostalih niti, dokler ni nit, ki je dostopala do podatkovne baze, končala le-tega. Prav tako pa so zaradi dejstva, da smo duplikate preverjali pred preverjanjem tega, ali je vsebina strani sploh html datoteka, bile nekatere strani označene kot duplikati namesto binary datoteke (vendar ne vse). Prav tako menimo, da bi se morali pri handlanju do errorjev ki jih vrne naš pajek v primerjavi z errorji, ki jih vrže HTTP request, poseči ravno obratnih sistemov, saj se bodo errorji do katerih pride v pajku v večini primerov vedno ponovili, medtem ko se errorji HTTP requestov ne bodo (recimo da takrat strežnik ni bil dosegljiv, zdaj pa je). Opazili pa smo tudi, da ponekod naš pajek ni pravilno upošteval časovnega razmika med dostopi do določene domene oziroma IP-ja, čeprav je le-to delovalo med testiranjem. Razlog pripisujemo dejstvu, da so niti prebrale čas zadnjega dostopa preden ga je druga nit spremenila in zaradi tega imela nepravilen čas zadnjega dostopa, predvsem ko so vse štiri niti obdelovale isto domeno hkrati. Zadnja opažena napaka je, da so se v podatkovni bazi nahajale le povezave, ki so ob klicu vrnile kodo 200, kar pripisujemo dejstvu, da v drugih primerih nismo uspeli dostopati do vseh atributov in smo zaradi errorja, kot prej omenjeno, le zapustili metodo.

References

[1] BeautifulSoup, https://www.crummy.com/software/BeautifulSoup/bs4/doc/
[2] Html.parser library, https://docs.python.org/3/library/html.parser.html
[3] Urllib library, https://docs.python.org/3/library/urllib.html
[4] Hashlib library, https://docs.python.org/3/library/hashlib.html
[5] Selenium, https://www.selenium.dev/
[6] urlnormalize, https://pypi.org/project/url-normalize/
[7] Socket library, https://docs.python.org/3/library/socket.html
[8] Threading library, https://docs.python.org/3/library/threading.html