Nejc Rihter, Tadej Lenart in Tomaž Sagaj

Breadth - first Pajek

Projektna naloga pri predmetu Iskanje in ekstrakcija podatkov s spleta

MENTORJA: prof. Dr. Marko Bajec in Timotej Knez

1 Uvod

V poročilu je opisana naša implementacija pajka v širino (angl. breadth-first webcrawler). Cilj projektne naloge je bil, da ustvarimo algoritem, ki bo zajemal podatke spletnih strani. Za izdelavo smo uporabili programerski jezik Python in uporabili nekatere že obstoječe knjižnice. Podatke zajetih strani smo shranili v podatkovni bazi PostgreSQL, ki smo jo postavili na platformi Supabase. Razlog, da smo uporabili to platformo, je bil, da smo se na podatkovno bazo lahko povezali vsi hkrati. Najpomembnejši elementi, ki smo jih shranjevali so HTML vsebina posamezne strani, vsebini datotek *robots.txt* in *sitemap.xml*, povezave na naslednje strani ter fotografije.
Naša naloga je bila, da kot semenske strani (angl. seed URL) nastavimo:

- gov.si,
- spot.gov.si,
- e-uprava.gov.si in
- e-prostor.gov.si.

2 Implementacija

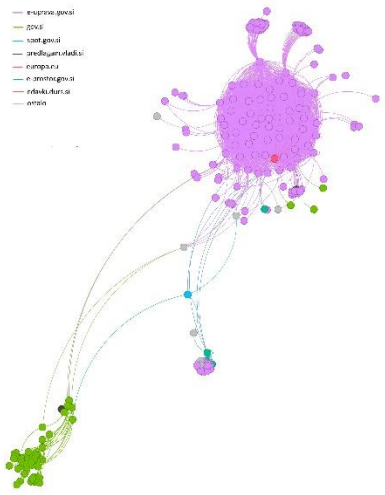
Za ustrezno delovanje našega pajka, je bilo potrebno implementirati pravila, ki jih mora pri delovanju upoštevati. Nastavili smo pet sekund zamika, med dvemi zaporednimi poizvedbami.
Da v bazo nismo shranjevali duplikatov, smo najprej poskrbeli, da je naš algoritem preveril, če v bazi že obstaja zapis, ki ustreza povezavi iskane strani. To dosežemo tako, da vsebino posamezne strani pošljemo v zgoščevalno funkcijo, to vrednost pa shranimo v tabelo. Tako lahko hitreje primerjamo vsebini dveh strani. Če strani, na kateri je naš pajek, še ni v bazi, najprej preverimo če obstajata datoteki *robots.txt* in *sitemap.xml*. Vsebino teh najprej shranimo v *site*, na to pa sama pravila tudi upoštevamo pri ekstrakciji podatkov.
Pomemben del naše naloge je tudi, da ustrezno z vsake strani, ki jo obiščemo, izluščimo povezave na naslednje strani. To so lahko linki (HTML značke <a>) ali pa povezave, ki so implementirane s pomočjo Javascripta. Te povezave povrsti shranjujemo v našo *Frontier* tabelo. Prav ta omenjena tabela, našega pajka usmerja na naslednje strani.
Kot naslednja pomembna funkcionalnost je ekstrakcija fotografij. Te shranjujemo v tabelo *images*, v kateri shranimo pot do najdene fotografije, ter tudi samo vsebino.
Tako povezave kot fotografije najdemo s pomočjo že obstoječe knjižnice *BeautifulSoup.*
Povezavo, ki jo je naš pajek obiskal, pred shranjevanjem tudi še kanoniziramo (angl. canonicalize). Vsako obiskano povezavo preverimo, če je ta katera od *binarnih* vsebinskih datotek (.pptx, .ppt, .docx, .doc, .pdf). V tem primeru, jih shranimo v bazi na ustrezno mesto.

Za vsako izmed vrst datotek, je bilo potrebno implementirati funkcijo za izluščevanje vsebine datoteke.
Za hitrejše delovanje je bilo potrebno dodati tudi, da naš program lahko teče z več zaporednimi nitmi (angl. threads). Program je zastavljen tako, da ob zagonu uporabnik vpiše, z koliko nitmi želimo izvajati iskanje.
Celotno aplikacijo smo zapakirali v Docker okolje, da je samo izvajanje preprosto na vseh računalnikih.

3 Statistika zajetih strani

V podatkovni bazi smo imeli:

- 262 spletnih mest
- 1397 strani
- 3140 duplikatov
- 3 binarne strani
- 1041 slik
- 0.75 povprečno število slik na spletno stran
- 7.23 povprečno število slik na strani, kjer je vsaj ena slika
- 68194 linkov v frontier-ju



Naredili smo vizualizacijo duplikatov, ki so bili že preplezani. Na sliki je prikazano kako so se med sabo prepletali linki strani, ki so že bile preplezane. Največ duplikatov so imeli strani e-uprava.gov.si in gov.si.

Slika 1: Vizualizacija duplikatov strani, ki so bile že preplezane

4 Zaključek

Med izdelavo naše rešitve, smo se veliko naučili. Največ težav sta nam predstavljali implementacija več vzporednih niti in hitrost poizvedb v podatkovno bazo. Vsi člani ekipe še takšne stvari nismo nikoli izdelovali, kar se je poznalo tudi pri samem delu, da je bilo potrebno veliko raziskovanja. Poleg tega pa je naša bila podatkovna baza velikokrat neodzivna (poizvedbe so velikokrat potekle) in poizvedbe preko pgAdmina so trajale po več kot 10s, tako da je obdelava enega linka trajala zelo dolgo večinoma zaradi hitrosti poizvedb. Želeli bi si, da bi preiskali več strani, kot smo jih, vendar smo preveč časa izgubili z odpravljanjem napak in hitrostjo poizvedb. Menimo pa, da je naš algoritem dober, in uspešno preišče in izlušči potrebne podatke s spleta.