Ekstrakcija podatkov s spleta

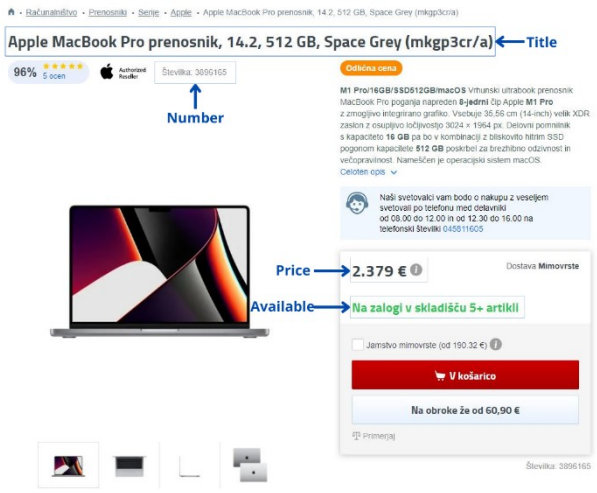
Anja Brelih
Univerza v Ljubljani, Fakulteta za računalništvo in informatiko
Večna pot 113, 1000 Ljubljana
ab0555@student.uni-lj.si

1 Uvod

Pri drugem projektnem delu pri predmetu Iskanje in ekstrakcija podatkov s spleta je bila naša naloga implementirati tri različne načine za ekstrakcijo podatkov s spleta. Pri prvem načinu smo podatke pridobili s pomočjo regularnih izrazov, pri drugem smo uporabili metodo XPath ter pri tretjem smo implementirali algoritem za avtomatsko ekstrakcijo podatkov.

2 Implementacija

Podani smo imeli po dve spletni strani dveh različnih tipov. Vsak par sledi enaki arhitekturi html strani, vsebina pa je drugačna. Pri par spletnih strani sta članka iz domene rtvslo.si, drugi par je seznam artiklov določenega iskalnega niza na domeni overstock.si. Za tretji par smo si izbrali spletni strani dveh artiklov na domeni mimovrste.com. Na spletni strani smo označili podatkovna polja (naslov, številka, cena in zaloga), ki jih bomo z različnimi metodami pridobili iz datoteke html (Slika 1: Primer spletne strani Mimovrste.com).



Slika 1: Primer spletne strani Mimovrste.com

Datoteke smo najprej uvozili ter prebrali s pravilnim kodirnikom, ki je naveden v glavi html datoteke. Tako se vsi simboli, ki jih spletna stran vsebuje pravilno preberejo.

2.1 Ekstrakcija z regularnimi izrazi

Pri ekstrakciji podatkov z regularnimi izrazi smo napisali programsko kodo za vsak od treh tipov strani glede na podatkovna polja, ki jih želimo iz spletne strani pridobiti. V prvem koraku smo pregledali html datoteko ter locirali iskano vsebino in pogledali katere značke ter s kakšnimi atributi jo obdajajo, saj nam ti služijo kot kazalniki na iskano vsebino.

Regularni izrazi so tipa *string*, kjer definiramo celotno značko v kateri se nahaja vsebina, na mesto vsebine pa zapišemo izraz ([^<]+), ki določa, da preberemo vse znake znotraj podane značke (Priloga 1: Regularni izrazi za spletne strani rtvslo.si).

Vso obstoječo vsebino v značkah, ki smo jih določili smo nato poiskali s knjižnico *re* (regular expression) v celotni datoteki. Dobljeni rezultati niso bili vedno primerni za prikaz, večkrat se je zgodilo, da je bilo potrebno odstraniti dvojne presledke ter znake za nove vrstice, ali pa celo rezati tekstovno vsebino, da smo pridobili ustrezne podatke.

Na koncu dobljene podatke iz spletne strani zapišemo v *json* formatu na standardni izhod ter v namensko datoteko znotraj mape *outputs*.

2.2 Ekstrakcija z metodo XPath

Pri ekstrakciji podatkov z metodo XPath smo prav tako pripravili programsko kodo za vsako izmed treh tipov spletnih strani.

Za vsako podatkovno polje, ki ga želimo pridobiti iz spletne strani smo napisali svoj izraz, ki se opira na *class*-e ter *id*-je ter tudi značke znotraj teh. Tekst, ki ga želimo izluščiti je določen z metodo *text()* na koncu našega sosledja značk ter atributov (Priloga 2: Izrazi XPath za spletne strani rtvslo.si).

Po pridobitvi vsebine z metodo XPath smo imeli manj dela s čiščenjem vsebine v primerjavi z metodo ekstrakcije z regularnimi izrazi. Vseeno pa so se pojavili primeri, ko smo morali odstraniti dvojne presledke, znake za novo vrstico ali rezati vsebino.

Na koncu dobljene podatke iz spletne strani zapišemo v *json* formatu na standardni izhod ter v namensko datoteko znotraj mape *outputs*.

2.3 Avtomatska ekstrakcija podatkov

Pri avtomatski ekstrakciji podatkov smo poskušali implementirati algoritem Road Runner, ki iz spletnih strani, ki so narejene po enaki predlogi, kreira ovoj, ki zadostuje vsem stranem ter zna pravilno poiskati unikatno vsebino, ki jo spletna stran vsebuje.

Spletni strani najprej odstranimo vso JavaScript kodo ter uporabljene stile, ki nam ne služijo kot pomoč pri preiskovanju spletne stani ampak le otežujejo delo.

Za preiskovanje spletnih strani smo uporabili *HTMLParser*, ki zapiše vse začetne in zaključne značke ter podatke kot žetone. Seznam žetonov nato podamo algoritmu Road Runner, ki zaporedno pregleduje in primerja seznama žetonov obeh spletnih strani, ki jih

preiskujemo. Najprej pogleda, če se žetona ujemata, ter če se, jih zapiše v ovoj. V kolikor prvi pogoj ni izpolnjen pogleda, v kolikor je podan žeton tipa *data*, kar pomeni, da smo naleteli na spreminjajoče podatke v spletnih straneh (Priloga 3: Izsek programske kode Road Runner-ja).

V kolikor tudi drugi pogoj ni izpolnjen, programska koda pogleda v kolikor se ujemata prejšnja in trenutna značka kot konec in začetek, kar lahko indicira ponavljanje. V tem primeru preišče začetek in konec prve in druge ponovitve ter pusti v ovoju zapisano le eno ponovitev. Če tudi ta pogoj ni izpolnjen smo naleteli na opcijski element, ki ne obstaja v obeh spletnih straneh, ki jih pregledujemo. Tudi ta se s pravilno oznako zabeleži v ovoj spletne strani.

Po koncu preiskovanja spletne strani se vse žetone zapiše nazaj v primerno obliko html značk glede na to, ali so začetne ali končne značke.

Ovoj spletne strani na koncu izpišemo na standardni izhod ter kot html datoteko v namensko mapo znotraj mape *outputs*.

3 Zaključek

Metodi ekstrakcije podatkov z regularnimi izrazi ter XPath sta implementirani pravilno glede na zastavljen cilj. Pri metodi za avtomatsko ekstrakcijo podatkov pa smo naleteli na več težav pri implementaciji algoritma Road Runner. Prvi pristop, ki smo ga izbrali je bil s knjižnico *BeutifulSoup*, vendar smo imeli težave z iskanjem po seznamu vseh elementov zato smo izbrali drug pristop in uporabili *HTMLParser*. V končnem rezultatu je več težav. Prva je, da smo v značkah odstranili vse atribute, tudi *class*-e in *id*-je, ki sta, poleg same strukture spletne strani, indikator, da beremo pravilno vsebino. Metoda za preiskovanje ponavljanja v sami kodi se ne izvrši pravilno, zato poskus implementacije ne zaznava ponavljanja v sami kodi kot bi moral. V primeru, da ima spletna stran opcijske elemente tudi teh ne zaznava pravilno, kar rezultira v temu, da se ovoj ne izdela do konca saj algoritem preiskuje spletne strani vzporedno oziroma zapiše le vzporedne ujemajoče elemente. Da bi bil primer algoritma implementiran pravilno in bi pridobili uporabne ovoje, bi bilo potrebno bolj podrobno izdelati pogoj, ko se vzporedni znački ne ujemata, možnost pa je, da obstaja enak element na drugem mestu v drugi spletni strani. Torej v kratkem, pravilno bi morali implementirati iskanje istega elementa prve strani znotraj celotne druge strani in ne le pregled vzporednih elementov. Za ta pristop bi nujno potrebovali atribute (*class* in *id*) značk oziroma pregledovati glede na drevesno strukturo html zapisa spletne strani.

Literatura

[1] GeeksforGeeks. Json.dump() in Python. https://www.geeksforgeeks.org/json-dump-in-python/#:~:text=The%20dump()%20method%20is,be%20stored%20as%20an%20argument. Dostopano: maj 2022

[2] Oxylabs. Python Web Scraping Tutorial: Step-by-step. https://oxylabs.io/blog/python-web-scraping. Dostopano: maj 2022

[3] R. Y. Zhang. Extracting XML Data from HTML Repositories. https://www.stat.ubc.ca/~ruben/website/ruththesis.pdf. Dostopano: maj 2022

[4] The Road Runner Project. http://www.dia.uniroma3.it/db/roadRunner/. Dostopano: maj 2022

[5] V. Crescenzi, G. Mecca in P. Merialdo. ROADRUNNER: Towards Automatic Data Extraction from Large Web Sites. http://vldb.org/conf/2001/P109.pdf. Dostopano: maj 2022

[6] Stackoverflow. Remove all javascript tags. https://stackoverflow.com/questions/8554035/remove-all-javascript-tags-and-style-tags-from-html-with-python-and-the-lxml-mod. Dostopano: maj 2022

Priloge

Priloga 1: Regularni izrazi za spletne strani rtvslo.si

```
title = r'<h1>([^<]+)<\/h1>'
subtitle = r'<div class="subtitle">([^<]+)<\/div>'
author = r'<div class="author-timestamp">+[ \t\n]+<strong>([^<]+)<\/strong>'
publishedTime = r'<div class="author-timestamp">+[ \t\n]+<strong>.*<\/strong>([^<]+)<\/div>'
lead = r'<p class="lead">([^<]+)<\/p>'
content = r'<\/div>[\s]*?<\/figure>[\s]*?<p([\s\S]*?)<div class=\"gallery\">'
```

Priloga 2: Izrazi XPath za spletne strani rtvslo.si

```
title = html_doc.xpath('//*[@id="main-container"]/div[3]/div/header/h1/text()')
subtitle = html_doc.xpath('//*[@id="main-container"]/div[3]/div/header/div[2]/text()')
author = html_doc.xpath('//*[@id="main-container"]/div[3]/div/header/div[3]/div[1]/strong/text()')
publishedTime = html_doc.xpath('//*[@id="main-container"]/div[3]/div/header/div[3]/div[1]/text()')
lead = html_doc.xpath('//*[@class="lead"]/text()')
content = html_doc.xpath('//*[@class="article-body"]/article/p/text()|//*[@class="article-body"]/article/p/strong/text()')
```

Priloga 3: Izsek programske kode Road Runner-ja

```
# Check for matching tokens

    if t1[0] == t2[0] and t1[1] == t2[1]:
        wrapper.append(t1)
        return roadRunner(tokens1, tokens2, w+1, s+1, wrapper)

    else:
        # Doesnt match
        if t1[0] == "data" and t2[0] == "data":

            # Content mismatch (#PCDATA)
            wrapper.append(["data","#PCDATA"])
            return roadRunner(tokens1, tokens2, w+1, s+1, wrapper)
```