Iskanje in ekstrakcija podatkov s spleta
2. seminarska naloga

Andraž Krašovec, Luka Podgoršek in Iztok Ramovš

May 2019

1 Uvod

Pri predmetu Iskanje in ekstrakcija podatkov s spleta smo se pri drugi seminarski nalogi ukvarjali s strukturirano ekstrakcijo podatkov s spletnih strani. Kot vhodne podatke smo dobili dve različni spletni strani, vsako z dvema podstranema. Za vsako podano spletno stran smo imeli definirane elemente, ki jih moramo izluščiti. Primer definiranih elementov je prikazan na sliki 1.

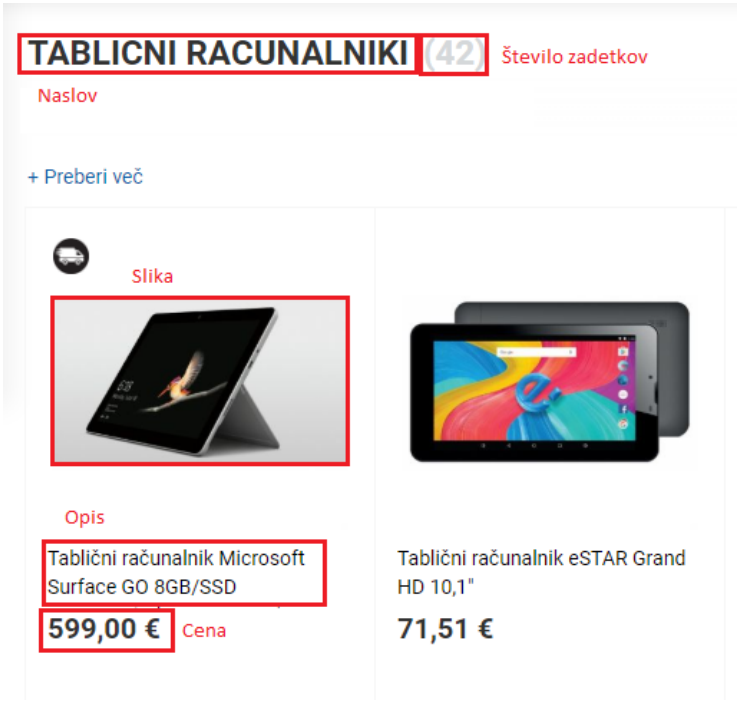
Figure 2.1: Overstock.com Web page sample.

Slika 1: Označeni podatkovni elementi

Za ekstrakcijo podatkov smo morali uporabiti tri različne metode. Prva z uporabo regularnih izrazov, druga s XPath izrazi in tretja z implementacijo algoritma Roadrunner. Kot programski jezik smo uporabili Python.

2 Opis izbrane spletne strani

Kot dodatno spletno stran smo izbrali spletno trgovino enaa.com. Izbrali smo jo zaradi velikega število različnih izdelkov, kar je zadostilo pogojem, opisanim v navodilih. Na spletni strani smo definirali podatkovne elemente, ki smo jih nato izluščili s spletne strani. Izbrani elementi so vidni na sliki 2.



Slika 2: Označeni podatkovni elementi

Na prvi podstrani spletne trgovine so prikazani izdelki, ki spadajo v kategorijo tabličnih računalnikov. Na drugi spletni strani pa so prikazani izdelki, ki spadajo v kategorijo prenosnih računalnikov.

3 Implementacija

Za vsako metodologijo smo implementirali svoj razred, ki je skrbel za inicializacijo in odpiranje vhodnih HTML dokumentov. Ta razred so nato vključevali vsi ostali razredi, ki so skrbeli za ekstrakcijo podatkov. Primer takšnega razreda, uporabljen pri regularnih izrazih, je viden spodaj.

```
class RegexParser:
    def __init__(self, file_name):
        self.path = dirname(dirname(dirname(realpath(__file__))))
            + "/input/" + file_name
        self.content = None

    def open_html(self):
        """
```

```
        Open the html page with lxml.
        """
        with open(self.path, encoding="latin1") as f:
            self.content = f.read()
```

Spletne strani so napisane v jeziku HTML. HTML jezik je sestavljen iz posebnih značk, ki označujejo za kakšno vsebino gre in kako naj to vsebino predstavi brskalnik.

3.1 Regex

S pomočno regularnih izrazov lahko iz HTML dokumenta izluščimo podatke. S pomočjo splošnega razreda smo odprli zahtevano datoteko in jo prebrali kot niz po katerem smo nato iskali podatke z uporabo regularnih izrazov.

3.1.1 Overstock

Za izluščevanje podatkov s spletne strani Overstock smo napisali razred OverstockParser, ki je implementiral osnovni razred za odpiranje vhodnih datotek. V razredu OverstockParser smo implementirali metodo run, ki je izluščila zahtevane podatke s vhodne spletne strani. Zaradi seznama elementov na spletni strani smo najprej poiskali element, ki vsebuje celoten seznam izdelkov. Nato smo z uporabo regularnih izrazov v zanki izluščili zahtevane podatke za vsak element.

```
class OverstockParser(RegexParser):
    def run(self):
        self.open_html()
        result = []
        regex = r"<td␣valign=\"top\">\s*<a␣href(\s*.*){5}<\/td>"
        matches = re.finditer(regex, self.content, re.MULTILINE)
        for matchNum, match in enumerate(matches, start=1):
            el = match.group()

            title_regex = r"<b>(.*)<\/b>"
            title = re.findall(title_regex, el)[0].lstrip()

            price_regex = r"<b>\$(.*)<\/b>"
            price = re.findall(price_regex, el)[0].lstrip()

            list_price_regex = r"<s>(.*)<\/s>"
            list_price = re.findall(list_price_regex, el)[0].
                lstrip()

            saving_regex = r"<span␣class=\"littleorange\">\$(.*)\s
                "
```

```
            saving = re.findall(saving_regex, el)[0].lstrip()

            saving_percent_regex = r"<span␣class=\"littleorange\"
                >\$.*\s\((.*)\)"
            saving_percent = re.findall(saving_percent_regex, el)
                [0].lstrip()

            result.append({
                "title": title,
                "list_price": list_price,
                "price": price,
                "saving": saving,
                "saving_percent": saving_percent,
            })
```

3.1.2 Rtvslo

Podobno smo implementirali razred za izluščevanje podatkov s spletne strani rtvslo. Nekateri izluščeni elementi so vsebovali nepotrebne značke, znake ali pa presledke, zato smo te odstranili. Odstranili smo jih zaradi lepše predstavitve podatkov, saj ponavadi generirane izhode uporabljamo v drugih delih programa ali neke programske rešitve.

```
class RtvsloParser(RegexParser):
    def run(self):
        self.open_html()
        author_regex = r"<div␣class=\"author-name\">(.*)<\/div>"
        author = re.findall(author_regex, self.content)[0].lstrip
            ()
        published_time_regex = r"<div␣class=\"publish-meta\">(\s
            .*\s*.*)<\/div>"
        published_time = re.findall(published_time_regex, self.
            content)[0]
        published_time = re.sub('[^A-Za-z0-9-:.]+', '␣',
            published_time).replace('␣br', '␣').lstrip()
        title_regex = r"<h1>(.*)<\/h1>"
        title = re.findall(title_regex, self.content)[0].lstrip()
        sub_title_regex = r"<div␣class=\"subtitle\">(.*)<\/div>"
        sub_title = re.findall(sub_title_regex, self.content)[0].
            lstrip()
        lead_regex = r"<p␣class=\"lead\">(.*)<\/p>"
        lead = re.findall(lead_regex, self.content)[0].lstrip()
        content_regex = r"<p␣class=\"Body\">(.*)<\/p>"
        content = re.findall(content_regex, self.content)[0]
        content = re.sub(re.compile('<.*?>'), '', content)
```

```
        result = {
            "author": author,
            "published_time": published_time,
            "title": title,
            "sub_title": sub_title,
            "lead": lead,
            "content": content,
        }
        return dumps(result)
```

3.1.3 Enaa

Kot dodatni spletni strani smo uporabili podstrani spletne trgovine Enaa. Podobno kot pri prvih dveh straneh smo implementirali razred za izluščevanje podatkov. Izsek programske kode prikazan spodaj.

```
class EnaaParser(RegexParser):
    def run(self):
        self.open_html()
        title_regex = r"<h1␣class=\"deptTitle\">(.*)</h1>"
        title = re.findall(title_regex, self.content)[0].lstrip()
        count_regex = r"<span␣class=\"deptTitle\">\((.*)\)</span>"
        count = re.findall(count_regex, self.content)[0].lstrip()

        result = []
        regex = r"<div␣class=\"productboxinner\">(.*)\s+</div>"
        matches = re.finditer(regex, self.content, re.MULTILINE)

        for matchNum, match in enumerate(matches, start=1):
            el = match.group()
            price_regex = r"<p␣class=\"enaa-cena\">(.*)</p>"
            price = re.findall(price_regex, el)[0].lstrip().
                replace(" ", "␣")
            img_src_regex = r"\s<img(.*)src=(.*)\"\sw"
␣␣␣␣␣␣␣␣␣␣␣␣␣img_src␣=␣re.findall(img_src_regex,␣el)[0][1]
␣␣␣␣␣␣␣␣␣␣␣␣␣item_title_regex␣=␣r"<div class=\"caption-name\">\s<a
    (.*)>(.*)</a>"
            item_title = re.findall(item_title_regex, el)[0][1].
                lstrip()
            result.append({
                "price": price,
                "img_src": img_src,
                "item_title": item_title
            })
```

Enaa je spletna trgovina kar pomeni, da ima na spletni strani prikazanih

več izdelkov. Sodobne spletne strani imajo dinamično generirano vsebino kar pomeni, da so izdelki v spletni trgovini predstavljeni z enako HTML kodo, ki se razlikuje po vsebini. Zato smo z regularnim izrazom pridobili element, ki vsebuje vse izdelke in nato uporabili dodatne regularne izraze za izluščevanje podatkov o posameznih izdelkih.

```
op = EnaaParser("enaa.com/enaa.html")
r = op.run()
print(r)
```

3.2 Xpath

XPath ali XML path language je namenjen izluščevanju elementov spletnih strani z definicijo poti do posameznega elementa. Pomagali smo si s knjižnico lxml, ki je skrbela za branje datotek in izvajanje poizvedb s pomočjo XPath. Struktura programske kode je enaka kot pri implementaciji regex dela naloge, zato se bomo v tem podpoglavju omejili zgolj na XPath poti, s katerimi smo dostopali do elementov.

3.2.1 Overstock

Najosnovnejši par strani sta predstavljali spletni strani s portala Overstock. Temu primerno enostavne so tudi poti, ki smo jih definirali za izluščevanje elementov. XPath poti smo sestavili zgolj z elementi in njihovimi indeksi elementov na določenem nivoju poti.

3.2.2 Rtvslo

Malo bolj napredno sintakso XPath smo uporabili pri izluščevanju spletnih strani s portala rtvslo.si in sicer smo si pomagali s posebnim znakom '@'. Ta nam omogoča, da smo iskali elemente glede na vrednost njihovih atributov, v našem primeru vrednost atributa id, ki je za tako iskanje izjemno priročen, saj naj bi imel vsak element drugačno vrednost in posledično bil s tem točno določen.

3.2.3 Enaa

Za strani po lastnem izboru, smo si izbrali spletni portal enaa.com. Posebnost izluščevanja elementov pri teh straneh je bila v tem, da smo želeli izluščiti tudi vrednosti atributov posameznih elementov, ne pa zgolj vsebine. Tukaj smo si ponovno pomagali s knjižnico lxml, ki omogoča pridobivanje atributov elementa in njihovih vrednosti.

3.3 Roadrunner

Za ekstrakcijo podatkov iz strukturno podobnih strani smo napisali tudi algoritem na osnovi metode Roadrunner. Glavna ideja algoritma Roadrunner je, da

detektiramo razlike med dvema strukturno zelo podobnima stranema in na podlagi njih določimo kje na strani se nahajajo zanimivi podatki. Pri tem moramo biti še posebej pozorni na ponavljajoče ali opcijske elemente, kar sta glavna razloga za strukturne razlike med omenjenima podobnima stranema.

Algoritem je implementiran v razredu RoadRunner. Za parsanje in prikaz smo uporabili knjižnico BeautifulSoup. Algoritem smo implementirali samo enkrat in sicer tako, da lahko s podajanjem parametrov o lokaciji vhodnih datotek zazna razlike med vsemi tremi pari izbranih vhodnih strani (overstack, rtvslo in enaa). Zaradi lepšega prikaza in manj detekcij neuporabnih podatkov smo v sklopu predprocesiranja dodali parametra CSS selektorja za okvirno html kodo, ki vsebuje želene podatke in seznam html značk, ki jih lahko ignoriramo. Primer klica Roadrunner algoritma je podan z spodnjim izsekom kode:

```
rr = RoadRunner("enaa.com/enaa.html",
                "enaa.com/enaa2.html",
                "body␣>␣div:nth-of-type(3)␣>␣div␣>␣div:nth-of-type
                    (3)␣>␣div:nth-of-type(3)␣>␣div",
                "body␣>␣div:nth-of-type(3)␣>␣div␣>␣div:nth-of-type
                    (3)␣>␣div:nth-of-type(3)␣>␣div",
                ['script', 'iframe', 'style'])
rr.run()
```

Omenjen razred na standardni izhod vrača rezultat v obliki regex izraza. Podrobnejši opisi zmožnosti rezultata, psevdokode implementiranega algoritma in uporabljenih hevristik so predstavljeni v poglavju 5.

4 Izhodi

Zaradi preglednosti poročila smo poročilu priložili vse izhode programa v datotekah. Datoteke z rezultati se nahajajo v mapi `results`. Datoteke so poimenovane po naslednjem vzorcu `results/<metoda>-<ime_strani>.txt`.

5 Psevdokoda in analiza algoritma Roadrunner

Algoritem smo zasnovali tako, da se izbrani, strukturno enaki html datoteki naprej predprocesirata. To naredimo tako, da z CSS selectorjem izberemo html značko, ki vsebuje želene rezultate. Nato odstranimo vse značke, ki smo jih podali s parametrom in jih ne rabimo (npr. script), vse komentarje, vse atribute in vse značke brez vsebine. To smo naredili zato, da v rezultatu ne dobimo preveč napačno detektiranih podatkov in da je rezultat bolj berljiv.

Psevdokoda implementiranega algoritma na osnovi metode Roadrunner je prikazana s spodaj:

```
doc1, doc2 = odpri_html(params)
doc1 = predprocesiranje(doc1, params)
```

```
doc2 = predprocesiranje(doc2, params)
doc1 = parse(doc1, doc2)

def predprocesiranje(doc1, params):
    izberi_pod_html_z_CSS_select_param
    odstrani_html_znacke_podane_s_param
    odstrani_vse_komentarje
    odstrani_vse_atribute
    odstrani_vse_znacke_brez_vsebine

def parse(doc1, doc2):

    if (doc1 je prazen):
        koncaj

    if (doc1 in doc2 imata enako znacko, drugo vsebino):
        preveri za neujemanje besedila znacke
        preveri ce so podznacke ponavljajoce
        for (podznacke):
            nadaljuj parse

    else:
        znacke se ne ujemajo (opcijska znacka)
        nadaljuj parse
```

Sama metoda parse je napisana rekurzivno, kar nam olajša primerjavo in parsanje značk z uporabo knjižnice BeautifulSoup.

Rezultat, ki ga algoritem vrača, je v obliki regex izraza. Pri opcijskih značkah smo dodali izraz `(znacka)?`, pri detektiranih ponavljajočih značkah pa smo dodali izraz `(znacka)+`. Detektirane spremembe v besedilu smo označili z `#DATA`, kar predstavlja možen zanimiv podatek, ki ga iščemo.

Algoritem včasih najde kakšen odvečen podatek, ki ga nismo iskali (na primer stranski meni *zadnje iz sekcije* na rtvslo). To se naredi takrat, ko so v predprocesirani html kodi ostala kakšna nezaželena besedila, ki se v danih datotekah razlikujejo. Opazili smo tudi, da včasih z našim algoritmom ni mogoče detektirati nekaterih podatkov, ker sta besedilna podatka enaka v obeh parih strani (na primer avtor pri rtvslo).

Če bi uporabili omenjen regex za ekstrakcijo podatkov iz nove podobne strani, se predvideva, da bo pred tem na novi strani narejeno enako predprocesiranje (z istimi parametri), kot je bilo pri pripravi regex izraza. Pri straneh, ki imajo ponavljajoče iskane objekte (na primer izdelki iz overstocka) je mišljeno, da se ustvari regex za en objekt, na podlagi katerega bi se na celotni strani iskali takšni objekti. To smo naredili zaradi berljivosti rezultata.

6 Zaključek

Uspešno smo implementirali vse tri zastavljene naloge - podatke s spletnih strani smo izluščili z uporabo regularnih izrazov, XPath in različico algoritma RoadRunner. Implementacija prvih nam ni povzročala težav, saj sta metodi preprosti in lahko razumljivi. Pričakovano smo imeli več dela z implementacijo metode RoadRunner, saj namesto zgolj opazovanja html datoteke in njenih elementov, le-ta primerja podobne strani med seboj in brez uporabnikovega pogleda v html strukturo, zazna razlike, oziroma zanimive točke spletne strani. Programski jezik Python se je izkazal za primerno izbiro, saj nam je nudil vse potrebne knjižnice, ki so nam močno olajšale izvedbo same naloge.