Seminar 2

Nejc Ribič, Klemen Jesenovec

veliki traven 2019

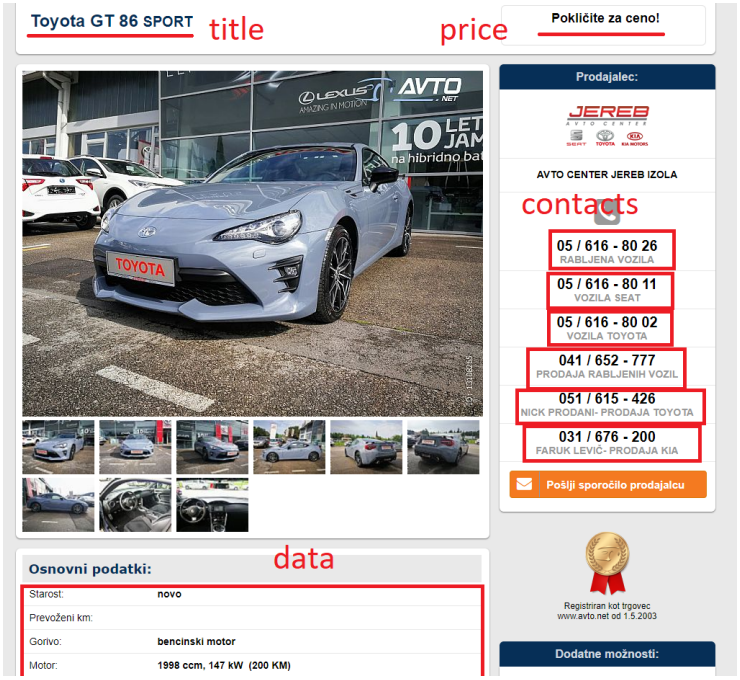
1 Uvod

Pri drugi seminarski nalogi, smo ekstrahirali željene podatke iz že zajetih spletnih strani.

Poročilo je sestavljeno iz štirih delov. V prvem delu najprej predstavimo dodatno spletno stran, katero smo preneseli in z nje ekstrahirali željene podatke. Sledi predstavitev zajemanja podatkov s pomočjo regularnih izrazov. Nato predstavimo zajem podatkov s poizvedovalnim jezikom XPath. Kot zadnji način zajema pa predstavimo še zajem podatkov s pomočjo RoadRunner algoritma. Na koncu sledijo še zaključek sklep.

2 Izbira strani

Pri izbiri spletne strani smo se odločili za oglase na strani `avto.net`. Razčlenitev podatkov je razvidna na sliki 1.



Slika 1: Razčlenitev strani avto.net.

3 Regularni izrazi

Regularni izrazi ali krajše *regex*, so zaporedje znakov, ki določajo iskalni vzorec v besedilu. Tehnika je bila razvita okoli leta 1950, ko je ameriški matematik skušal formalizirati opis običajnega jezika.

Regex izrazi se uporabljajo v iskalnikih, urejevalnikih besedila, v leksikalni alanize in še na mnogih drugih področjih. V seminarski nalogi, smo skušali z uporabo *regex* izrazov izluščiti podatke iz že zajetih strani.

3.1 Luščenje podatkov

Seminarska naloga je vsebovala tri različne tipe strani, pri čemer je vsaka izmed njih vsebovala dve podobni strani istega tipa. Za luščenje podatkov je bilo najprej potrebno identificirati podatke, ki so za nas ustrezni (pomembni). Omenjene podatke smo imeli specificirane že v sklopu same seminarske naloge.

Pred iskanjem regularnih izrazov, smo najprej vizualno pregledali strukturo zajete strani, kjer smo opazovali, če se slučajno ponavlja kakšen vzorec, kjer so prikazani naši podatki. Ko smo takšne vzorce zaznali, smo nato s pomočjo spletne strani[2] izdelali regularni izraz, ki je kot rezultat vrnil potreben podatek.

Velikokrat se je zgodilo, da je regularni izraz našel dva podatka, čeprav bi moral najti zgolj enenga. To je pomenilo, da smo morali regularni izraz

dodelati do te meje natančnosti, da je vračal samo en podatek. Posledično je to pomenilo, da je bil sam regularni izraz dolg več kot 50 znakov, kar seveda pomeni, da sam regularni izraz ni berljiv kaj šele razumljiv. V splošnem so naši končni regularni izrazi vsebovali nekaj fiksnih *div* značk in *class* vrednosti. Nato je sledila vrednost regularnega izraza, ki je pokrila ustrezne vrednosti, kot so na primer [0-9a-zA-Z]+ itd. Na koncu izraza pa je sledila še zaključna značka spletne strani. Na ta način smo izluščili vse potrebne podatke.

3.2 Stran avto.net

Spletna stran *avto.net* ima atribute jasno označene. Atribut *title* pridobimo z regularnim izrazom: `<h1>([a-zA-Z ;0-9 <>]+)<`. Atribut o ceni pa pridobimo z regularnim izrazom: `class="OglasDataCenaTOP"([a-zA-ZčČ ˚ ˆ:;= !0-9,.¿]+)`.

Zanimivo je bilo zajeti kontaktne podatke. Pridobili smo jih s poizvedbo: `<div class="OglasMenuBox Bold OglasMenuBoxPhone≫([˚ ˇ])(.*)`. Poizvedba vrača več rezultatov, preko katerih se je potrebno sprehoditi z zanko in jih shraniti v seznam, katerega smo na koncu pretvorili v JSONArray.

Podrobne podatke o avtomobilih smo pridobili v obliki *ključ: vrednost*. Ključ smo pridobili z izrazom: `<div class="OglasDataLeft≫(.*)<div>`. Vrednost pa z izrazom: `<div class="OglasDataRight≫(.*)<div>`. Rezultate smo nato shranili v objekt *HashMap* in jih pretvorili v JSON obliko.

3.3 Stran overstock.com

Stran *overstock.com* vsebuje sezname izdelkov. Regularni izrazi najdejo množico rezultatov, katere je potrebno z zanko posamično obdelati in jih shraniti v objekt `OverStock.java`. Izraz, ki pridobi naslov izdelka: `<a href=(.*?)><b>([a-zA-Z0-9-.( ')]+)</b></a><br>`. Izraz, ki pridobi vsebino izdelka: `<span class="normal≫([a-zA-Z .,0-9-'()]+)`.

Zelo podobni izrazi so uporabljeni tudi za zajem cen, in ostalih potrebnih podatkov. Razlikujejo se samo v začetnem delu regularnega izraza, kateri najde mesto v HTML dokumentu, kjer se podatki nahajajo. Seznam shranjenih objektov `Overstock.java` nato pretvorimo v JSONArray.

3.4 Stran rtvslo.si

Na spletni strani *rtvslo.si* naslov pridobimo z regularnim izrazom: `<h1>([A-Za-z 0-9:]+)`. Podnaslov pridobimo z izrazom `<div class=šubtitle≫([A-Za-z 0-9:]+)`.

Podatek avtorju pa pridobimo z izrazom `<div class="author-name≫([a-zA-Z ˚ 0-9. a-z:]+)`. S podobno poizvedbo pridobimo tudi podatek o vsebini članka, izraz se razlikuje zgolj v vrednosti atributa `class`.

Podatek o času objave pridobimo z izrazom: `<div class="publish-meta≫([˚ 0-9. a-z:]+)`. Tako kot za vsako stran tudi te podatke shranimo v objekt `RtvSlo.java` in ga nato pretvorimo v JSON objekt.

4 XPath izrazi

XPath je jezik za iskanje vozlišč v XML dokumentu, ki ga je definiral W3C [5]. Ker je HTML vrsta XML dokumenta, ga lahko uporabimo za ekstrakcijo podatkov iz spletne strani.

Za implementacijo te metode v programskem jeziku Java smo uporabili knjižnjico HtmlCleaner [1]. Ta nam omogoča "čiščenje"HTML dokumenta, kar poskrbi za dobro strukturiran HTML vhod. Poleg tega pa omogoča tudi iskanje po dokumentu s pomočjo XPath poizvedb.

4.1 Stran avto.net

Ta stran ima vozlišča dobro označena z atributi, kar olajšuje iskanje podatkov. Naslov oglasa lahko pridobimo s poizvedbo `"//div[ @class = 'OglasDataTitle' ] / h1 / text()"`.

Tudi ceno pridobimo na podoben način s poizvedbo `"// p[ @class = 'OglasDataCenaTOP' ] / text()"`.

Kontaktov, ki jih pridobimo s poizvedbo `"//div[ @class = 'OglasMenuBox Bold OglasMenuBoxPhone' ] / text()"`, je lahko na strani več. V JSON objekt zapišemo vsakega posamično.

Najtežji del je luščenje podtakov o lasnostih vozila. Tu najprej pridobimo vse alineje z lastnostmi s pozvedbo `"// div [ @class = 'OglasData' ]"`. Vsako tako vozlišče ima levi in desni del v svojem `div` voznišču. Pri luščeju je potrebno paziti na navadne lastnosti in lastnosti, ki so združene v eno kategorijo. Te podatke si sproti beležimo v slovar in jih nakoncu zapišemo v JSON objekt.

Opis uprabnika pridobimo s poizvedbo `"// div [ @class = 'OglasEQtext' ] / text()"`.

4.2 Stran overstock.com

Vse izdelke na tej strani pridobimo s poizvedbo `"// table[ @border = '0' ][ @cellpadding = '2' ] / tbody / tr[ @bgcolor ]"`. Nato obdelujemo vsak vnos posamično.

Najprej pridbimo `td` vozlišča v vnosu s poizvedbo `"td"`. Če je vnos veljaven (ni komentar izdelka), ima natanko 2 taka elementa. Podatki, ki nas zanimajo, so v drugem elementu.

Iz drugega elementa izluščimo naslov, ki je vgnezden v povezavo, izdelka s poizvedbo `"a/text()"`.

Ta element še naprej razčlenimo s poizvedbo `"table / tbody / tr / td"`. Tako dobimo 2 elementa. V prvem je tabela s podatki o ceni in prihrankih, v drugem elementu je opis izdelka.

4.3 Stran rtvslo.si

Tudi ta stran je dobro strukturirana, kar olajšuje ektrakcijo podatkov. Avtorja in čas objave lahko pridobimo s poizvedbo `"// div[ @class = 'author-`

`timestamp' ] / text()"`. Ta niz razpolovimo pri znaku `'|'`, da razčlenimo avtorja in čas.

Naslov, podnaslov in prvi odstavek lahko pridobimo s preprostimi poizvedbami `"// header [ @class = 'article-header' ] / h1 /text()", "// div[ @class = 'subtitle' ] / text()"in "// p[ @class = 'lead' ] / text()"`.

Za pridobitev vsebine uporabimo dve poizvedbi `"// div[ @class = 'article-header-media' ] / text()"in "// article[ @class = 'article' ] / p / text()"`. Pri drugi poizvedbi lahko pridobimo več odstavkov. Vse pridobljene odstavke združimo in jih nato zapišemo v JSON objekt.

5 Road-runner

Ideja algoritma *RoadRunner* temelji na iskanju univerzalnega regularne izraza za ekstrahiranje podatkov s spletne strani. Pri implementaciji smo izhajali iz članka [4], ki prikazuje koncepte, logiko in pristope pri ekstrahiranju podatkov s pomočjo algoritma *RoadRunner*.

5.1 Pseudo algoritem

Za lažjo implementacijo smo si ustvarili pomožno drevesno strukturo (*RNode.java, RElement.java, RText.java, RReg.java*). V splošnem strukturi kot argument podamo DOM drevo HTML dokumenta, kateremu potem naša struktura izloči brezpomenske tekstovne vrednosti in jim očisti HTML značke (atributi, stili,...), sama drevesna struktura pa ostane enaka. Omenjena struktura je bila temelj pri implementaciji našega *RoadRunner* algoritma.

5.2 Implementacija

Implementacije smo se lotili na dva načina. Pri prvi implementaciji smo izhajali iz postopka, ki smo ga predstavili na predavanjih. Pri drugi implementaciji pa smo izhajali iz postopka, ki je predstavljen v članku [4]. Algoritem smo implementirali do te mere, da je bil sposoben delno ustvariti regularni izraz osnovne spletne strani, ki je bila predstavljena na predavanjih.

5.3 Prvi pristop

Ideja prvega pristopa temelji na posodabljanju drevesne strukture HTML strani z regularnimi izrazi. Algoritem išče za pozicije, kjer se razlikujejo tekstovne vrednosti, kjer se pojavijo opcijski elementi in kjer se pojavijo iterativni elementi. Pseudo koda algoritma je predstavljena spodaj.

```
rBuilder -> html drevo, ki posodablja regularni izraz
fun search(d1, d2, rBuilder):
        if d1.tag == d2.tag
            if d1.children.size != d2.children.size
                rBuilder.addNode(findIter(d1, d2))
```

```
            if d1.text != d2.text
                rBuilder.addNode(findText(d1, d2))
         else
            rBuilder.addNode(findOptional(d1, d2))

        for c1, c2 in d1, d2:
            align(c1, c2, rBuilder)
            search(c1, c2, rBuilder)
        for sib1, sib2 in d1, d2:
            align(sib1, sib2, rBuilder)
            search(sib1, sib2, rBuilder)
```

5.4 Drugi pristop

Pri drugem pristopu smo poskušali implementirati algoritem opisan v članku []. Implementacija je predstavljena s spodnjo pseudokodo.

```
fun roadRun(RRNode currW, RRNode currS):
        if currW and currS is Element:
            if same tag:
                for nodes children:
                    if mismatch occurred:
                        handle_mismatch()
        if currW is different type from currS:
            handle_mismatch()

        if currW and currS is Text:
            if text differs:
                replace with #PCDATA

fun handle_mismatch():
    handle iterative mismatch
        if not handled:
            handle optional
            if not handled:
                HALT "Cant match"
```

6 Podatki

Izhodi vseh metod so dostopni na GitHub repozitoriju [3].

7 Zaključek

V seminarski nalogi smo se naučili ogromno novega znanja, predvsem na področju regularnih izrazov. Zanimivo je bilo implementirati RoadRunner algori-

tem, kljub temu, da ni povsem zaključen in njegovo delovanje ni v celoti preverjeno. V splošnem smo z rezultati zadovoljni.

Literatura

[1] Htmlcleaner. Dosegljivo: http://htmlcleaner.sourceforge.net/. [Dostopano: veliki traven, 2019].

[2] Online regex tester and degugger. Dosegljivo: https://regex101.com/. [Dostopano: veliki traven, 2019].

[3] ribicnejc/web-data-extraction. Dosegljivo: https://github.com/ribicnejc/web-data-extraction. [Dostopano: veliki traven, 2019].

[4] Paolo Merialdo Valter Crescenzi, Giansalvatore Mecca. Roadrunner: Towards automatic data extraction from large web sites. *Annalen der Physik*, 2001.

[5] Xpath. Dosegljivo: https://en.wikipedia.org/wiki/XPath. [Dostopano: veliki traven, 2019].