UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

ISKANJE IN EKSTRAKCIJA PODATKOV S SPLETA

---

**SEMINARSKA 2**

# Ekstrakcija strukturiranih podatkov

**Končno projektno poročilo**

---

*Avtorja:*
BERNIK Matic
TOVORNIK Robert

*Profesor:*
dr. Marko BAJEC
*Asistent:*
doc. dr. Slavko ŽITNIK

May 8, 2019

# 1 Uvod

## 1.1 Opis problema

Cilj seminarke naloge je bil implementacija treh pristopov za ekstrakcijo podatkov v strukturirani obliki s spletnih strani. Pristopi so obsegali:

- uporabo regularnih izrazov,
- uporabo XPath izrazov,
- implementacijo RoadRunner algoritma.

Poleg stirih spletnih strani dveh pred-dolocenih vrst (overstock.com in rtvslo.si) je naloga zahtevala obdelavo dveh dodatnih spletnih strani poljubnega tipa. V ta namen sva izbrala strani oglasov s portala `bolha.com`.

# 2 Izbrani strani bolha.com

Dodatno sva za namene ekstrakcije podatkov izbrala par 'detail' spletnih oglasov s portala bolha.com.

Strukturi spletnih strani in podatki, katere sva se namerila iz njiju izlusciti, sta razvidni iz spodnjih slik.

Razvidno je, da se strani med drugim razlikujeta tudi v stevilu podatkov o uporabniku (uporabnikova telefonska in mobilna stevilka) ter dolzini clenov v navigacijskem razdelku (kategorija).

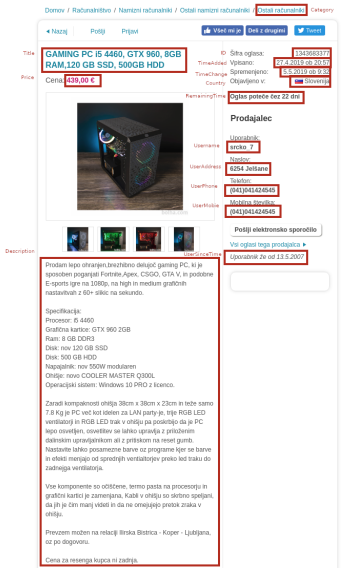


Figure 1: Slika prikazuje prvo izmed izbranih spletnih strani s portala bolha.com. Na njej so z rdeco barvo oznaceni elementi, katere nase implementacije izluscijo iz dokumenta.


Figure 2: Slika prikazuje drugo izmed izbranih spletnih strani s portala bolha.com. Na njej so z rdeco barvo oznaceni elementi, katere nase implementacije izluscijo iz dokumenta.

# 3 Implementacije

Programska koda iz katere so razvidni tudi vsi uporabljeni regularni in XPath izrazi je dostopna na naslovu https://github.com/roberttovornik/webpage-data-extraction/blob/master/code/data_extraction.py.

## 3.1 Regularni izrazi

Za ekstrakcijo podatkov z uporabo regularnih izrazov sva uporabljala Pythonovski modul 're'.

### 3.1.1 bolha.com

Ekstrakcija vecine podatkov je bila relativno enostavna, saj se pred polji, ki nas zanimajo, nahajajo labele, ki jih unikatno oznacujejo. Uporabljen regularni izraz za ekstrakcijo datuma spremembe oglasa je tako naprimer

```
'<p><label>Spremenjeno:<\/label>(.*)<\/p>'.
```

Za vsak tip podatka sva uporabila svoj regularni izraz.

### 3.1.2 rtvslo.si

Zaradi blizine naslova in podnaslova clankov znotraj HTML dokumentov, sva oba izluscila kar z uporabo enega samega regularnega izraza

```
'<h1>(.*)</h1>\textbackslash s+
<div class=\textbackslash"subtitle\textbackslash">(.*)<\textbackslash/div>'
```

, s cimer sva tudi pridobila na robustnosti pogoja.

Ekstrakcije vsebine clanka sva se lotila tako, da sva najprej poiskala vse odstavke oz. paragrafe <p> z uporabo regularnega izraza '<p[^>]*?>([\s\S]*?)<\/p>'. Od teh sva izlocila tiste odstavke, ki vsebujejo '<iframe', besedilo iz vseh preostalih pa zdruzila. V tako dobljenem besedilu sva nato zamenjala znacke '<br>' s prelomi vrstic '\n' in odstranila vse preostale znacke, ki so ustrezale regularnim izrazom '<\/?[a-zA-Z]*>'

### 3.1.3 overstock.com

Zaradi ponavljanja uporabe enakih HTML elementov kot sta <td> in <tr> skozi dokument je bila ekstrakcija tu nekoliko tezavnejsa in je zahtevala uporabo daljsih regularnih izrazov. Tako sva uporabila izraz

```
'<tr bgcolor=\"#[fd]*\">\s*<td valign="top" align="center">[\s\S]*?
<td valign="top">([\s\S]*?)<\/td>\textbackslash s*<\/tr>\s*<tr>\s*
<td colspan="2" height="4">'
```

da sva iz HTML dokumenta najprej izluscila seznam posamezih oglasov oz. njihovih relevantnih odsekov.

V nadaljevanju sva obdelovala vsakega od oglasov posebej. Podatke vezane na ceno izdelka ('ListPrice', 'Price', 'Saving' in 'SavingPercent') sva izluscila z uporabo enega daljsega regularnega izraza:

```
'<tbody><tr><td align="right" nowrap="nowrap"><b>List Price:</b></td>
<td align="left" nowrap="nowrap"><s>(.*)</s></td></tr>\s+<tr><td align="right"
nowrap="nowrap"><b>Price:</b></td><td align="left" nowrap="nowrap">
<span class="bigred"><b>(.*)</b></span></td></tr>\s+<tr>
<td align="right" nowrap="nowrap"><b>You Save:</b></td><td align="left"
nowrap="nowrap"><span class="littleorange">([\^{}\(]*)\s+\(([\^{}\)]*)\)</span>
</td></tr>\s+</tbody>'
```

## 3.2 XPath

Za ekstrakcijo podatkov z uporabo XPath izrazov sva uporabljala Pythonovski modul 'lxml'. Modul preko metode text_content() omogoca pridobitev vsega besedila iz celotnega HTML pod-drevesa, na racun cesar sva poenostavila nekatere XPath izraze.

V nekaterih primerih sva za osnovo XPath oznake vzela izraz, kakrsnega za HTML element zgenerira Google Chrome Developer Tool in ga nato preuredila. Spremembe so bile potrebne zaradi vecje robustnosti oz. ker so se samodejno generirani izrazi raje opirali na pozicije elementov, kot pa na njihove atribute.

### 3.2.1 bolha.com

Primer XPath izraza za pridobitev podatka o kategoriji oglasa s pripadajoco Python kodo:

```
tree = html.fromstring(document)
tree.xpath('//*[@id="breadcrumbs"]/nav/a[last()-1]')[0].text_content()
```

Osnovne podatke o oglasu kot so 'ID', 'TimeAdded', 'TimeChange' in 'Country' sva izluscla kot besedilo zaporednih paragrafov odseka 'documentInfo':

```
document_info_paragraphs =
tree.xpath('//div[@class = "infoBox"]/div[@class="documentInfo"]/p')
    article['ID'] = document_info_paragraphs[0].xpath('./text()')[0]
    article['TimeAdded'] = document_info_paragraphs[1].xpath('./text()')[0]
    article['TimeChange'] = document_info_paragraphs[2].xpath('./text()')[0]
    if len(document_info_paragraphs)>4:
        article['Country'] = document_info_paragraphs[3].xpath('./span/text()')[0]
```

#### 3.2.2 rtvslo.si

Podatke sva izluscila z uporabo locenih XPath izrazov za vsakega izmed podatkov. Ker je vsebina clanka 'Content' razdeljena na razlicne paragrafe, sva text le-teh poiskala z uporabo regularnega izraza

```
'//*[@id="main-container"]/div[3]/div/div[2]/article/p/text()'
```

in jih s prelomi vrstic zdruzila.

#### 3.2.3 overstock.com

Najprej sva z uporabo regularnega izraza:

```
'/html/body/table[2]/tbody/tr[1]/td[5]/table/tbody/tr[2]/td/table/tbody/tr/td
/table/tbody/tr[@bgcolor]'
```

poiskala HTML pod-drevesa, ki pripadajo posameznim oglasom. V nadaljevanju sva za vsak oglas loceno izluscila zahtevane podatke. Primer stavka za izluscitev vsebine oglasa 'Content':

```
listings[i]['Content'] = listing_trees[i].xpath('./td[2]/table/tbody/tr/td[2]')[0]
.text_content()
```

## 3.3 RoadRunner

### 3.3.1 Pregled metode - vir implementacije

Pri implementaciji roadrunner algoritma sva se obrnila na direkten vir oziroma izvor prvotne ideje algoritma ROADRUNNER: Towards Automatic Data Extraction from Large Web Sites". Pri tem sva se držala osnovih idej, ki so podane v poglavjih "Theoretical Background" in "The Matching Technique". Delovanje sva implementirala kot rekurizvno funkcijo, ki jo kličemo nad dvema primerkoma sorodnih spletnih strani, na izhod pa dobimo UFRE zapis ovojnice. Pri implementaciji nisva uporabila binarnih AND-OR dreves, temveč zgolj rekurzijo.

### 3.3.2 Koraki implementacije

- PREDPROCESIRANJE - Pred zagonom algoritma roadrunner za primerjanje vsebine spletnih strani je le-te potrebno najprej primerno pripraviti. To sva storila tako, da vsako spletno stran razbijeva na posamezne enote - "tokene", ki predstavljajo osnovne gradnike, kot jih opisuje že članek (znakovni niz ali oznaka html elementa). Pri tem sva že ob pred-procesiranju ohranila dodatno informacijo o html značkah - torej ali gre za začetno ali zaključno značko posameznega html elementa. S tem si poenostavimo nadaljnje pregledovanje dokumentov. Dodatno sva prečistila tudi vse znakovne nize, ki se nahajajo v podatkovnih poljih. Vse sva poenotila tako, da sva odstranila vse odvečne presledke, tabulatorje, prehode v novo vrstico idr. ter jih pretvorila v nizko-znakovne nize. Omenjeni korak je pomemben, saj želimo pri primerjavi elementov zaznati, kdaj gre za statično in kdaj za dinamično vsebino znakovnih nizov. S tem se izognemo vsem dvomom, kadar je vzrok le-teh format podanega besedila. Dodatno bi lahko povečali robustnost še z uporabo katere izmed primerjalnih funkcij nizov (npr. Levenshteinovo razdaljo), da bi se izognili dvomljivost še v primeru manjših slovničnih napak. Pri razbitju HTML / XHTML dokumentov sva uporabila Python knjižnico HTMLParser pri čemer sva uporabila dodatne arbitrarne obvladovalnike.

- ALGORITEM - Pri samem algoritmu sva se odločila za rekurzivno implementacijo obvladovanja neujemanj. Sprva sva preizkusila tudi zgolj iterativno verzijo, ki je bila sicer hitrejša, vendar pa je vsebovala veliko napak. Rekurzivno obvladovanje priporoča tudi prvotni članek. Pri tem sva poskrbela za vse oblike neujemanja elementov - znakovno neujemanje, znakovno (html oznake) s strani enega ali drugega dokumenta (osnovne ovojnice ali dodatne strani) ter možnosti opcijskih elementov. Pri tem neujemanja obvladujeva na naslednje

načine:

- Znakovno neujemanje vsebine niza - pri tem ugotovimo, da gre za območje dinamične vsebine (nam pomembne vsebine). V ovojnici tak niz nadomestimo z oznako '**#PCDATA**', kar nam omogoča hitro iskanje območji z visoko verjetnostjo pomembnih podatkov.

- Znakovno neujemanje značk - opcijski elementi - v danem primeru gre za neujemanje soležnih vsebin zaradi pojavitve arbitrarnega elementa, ki ni prisotev v obeh dokumentih. Pri tem moramo paziti, da je element lahko prisoten ali v ovojnici ali pa v novem dokumentu. Temu primerno ga označimo in ga v ovjnico dodamo obdanega z naslednjo oznako: " **(** ..vsebina opcijskega elementa.. **)?** ".

- Znakovno neujemanje značk - iteratorji - v slednjem primeru gre za neujemanje soležnih vsebin zaradi različnega števila vsebovanih iteratorjev (npr. elemntov liste). Znova gre lahko za neujemanje na strani ovojnice ali nove vzorčne strani. V tem primeru je potrebno dodatno pozornost nameniti urejanju že obstoječe ovojnice. V predhodnih korakih, preden smo zaznali elemente iteratorjev smo v ovojnici že dodali prednodne pojavitve iteratorja. Zato je le-te potrebno poiskati in nadomestiti s splošno obliko zapisa iteratorja v oblki: " **(** ... vsebina iteratorja ... **)+**.

### 3.3.3 Rezultati in delovanje

Delovanje sva tesno preizkusila na podanem primeru delovanja v članku. Za podani strani:

```
<HTML>
 Books of:
 <B>
 John Smith
 </B>
 <UL>
 <LI>
 <I>Title:</I>
 DB Primer
 </LI>
 <LI>
```

```
 <I>Title:</I>
 Comp. Sys.
 </LI>
 </UL>
</HTML>
```

in

```
<HTML>
 Books of:
 <B>
 Paul Jones
 </B>
 <IMG src=.../>
 <UL>
 <LI>
 <I>Title:</I>
 XML at Work
 </LI>
 <LI>
 <I>Title:</I>
 HTML Scripts
 </LI>
 <LI>
 <I>Title:</I>
 Javascript
 </LI>
 </UL>
</HTML>
```

sva prejela naslednji pričakovan rezultat UFRE:

```
<html>
    books of:
    <b>
    #PCDATA
    </b>
    ( img )?
    <ul>
```

```
        ( <li><i>title:</i>#PCDATA</li> )+
    </ul>
</html>
```

Rekurzivni in primerjalni klici delujejo za vse podane primere. Ko sva implementacijo nadaljnje preizkušala na izbranih straneh, sva naletela na probleme pri globini rekurzivnih klicev. Ker jih nisva omejila pride pri obsežnih spletnih straneh do prekoračitve privzete omejitve.

### 3.3.4 Možne izboljšave, nadaljnje hevristike

Kot že omenjeno, bi bile potrebne izboljšave na področju rekurzivnih klicev ( določitev omejitve ) ali pa z manjšimi spremembami implementacija v obliki predlaganih AND-OR dreves. Prav tako bi lahko izboljšali primerjave znakovnih nizov (kot že omenjeno) z uporabo primerjalnih tehnik in ohranitvijo statičnih nizov pri minimalnih odstopanjih v podobnosti. Več bi lahko uporabili tudi informaciji vezanih direktno na znane elemente html oznak. Npr. domensko znanje, ki nam pove, da elemen "li" predstavlja elemen list in "ul" enkapsulacijo urejenih iteratorjev". S tem bi lahko optimizirali primerjalno delovanje algoritma. Dodatno izboljšavo oziroma robustnost bi lahko zagotovili tudi pri primerjanju opcijski elementov. Predlagano je preiskovanje "takojšnjega" naslednjega elementa. Pri tem bi lahko delovanje izboljšali tako, da bi dopuščali opcijsko okno znotraj html oznak, in bi primerjanje nadaljevali šele ob koncu opcijskega okna (začetni-končni tag html elementa). Kot to nakazuje funkcija find_end_of_optional().

# 4 Rezultati

Izhodi vsake od implementiranih metod so dostopni v obliki .json datotek na povezavi https://github.com/roberttovornik/webpage-data-extraction/tree/master/data/extracted_data.

# 5 Povezave

Github repozitorij projekta se nahaja na naslovu: https://github.com/roberttovornik/webpage-data-extraction