Assignment 1: Web crawler

Gal Lindič, Andrej Drofenik, Ožbej Golob

I. Introduction

The goal of this programming assignment was to implement three different approaches for the structured data extraction from the Web: regular expressions, XPath, and RoadRunner-like Wrapper implementation or other automatic content extraction. We implemented all of the approaches on two web pages from each of the three web sites: *overstock.com*, *rtvslo.si*, and *bolha.com*.

II. Two selected web pages

For the purpose of the structured data extraction, we chose two web pages from the *bolha.com* web site. The web pages are advertisements for two mobile phones: iPhone XR 128 GB and Samsung GT E1200. We extracted the following fields: Title, Price, Id, PublishedTime, ValidUntil, Views, Type, Location, Status, and Content. Figure 1 shows a sample of the advertisement from the *bolha.com* web site.

Figure 1. *bolha.com* web page sample.

III. Regular expressions implementation

A. Overstock

Overstock contains multiple products so we had to iterate trought them using regular expressions and for each find the needed content. We used the following expressions:

- Title: "([0-9]+-(.+)(?=/b>))",
- ListPrice: "<s>(.*?)</s>",
- Price: "([$€]*[0-9;]+)"
- Saving: "([$€]*[0-9;]+)"
- Saving Percent: "([0 − 9] + %)",
- Content: "<td valign=̈top̈>(.*?)"

For the RTV site we have two articles. We used the following regular expressions to extract the needed information:

B. rtvslo.si

- Author: " (.*) ",
- Title: "<h1>(.*)<h1>",
- SubTitle: " (.*) ",
- PublishedTime: " (.*?)
",
- Lead: " (.*) ",
- Content: "<article class=ärticlë>(.*?)<article>"

The Bolha site was a bit more complex to extract information using regular expressions

C. bolha.com

- Title: "<h1 class=ëntity-titlë>(.*)<h1>",
- Price: "<strong class=p̈rice price–hrk̈>(.*?)",
- Id: "(.*)",
- PublishedTime: "<time class=v̈alue¨*.*>(.*)<time>",
- ValidUntil: "(.*)",
- Views: "(.*)",
- Type: "<td>(.*?)<td>"; match[0],
- Location: "<td>(.*?)<td>"; match[1],
- Status: "<td>(.*?)<td>"; match[2],
- Content: " (.*?) "

IV. XPath implementation

A. overstock.com

Since the web site contains multiple products, we firstly saved these products to an array with the command: "/html//body/table[2]/tbody/tr[1]/td[5]/table/tbody/tr[2]/td/table/tbody/tr/td/table/tbody/tr". We then iterated through the products and ignored empty results. For each product we used the relative XPath path (from the one saved in the products array).

- Title: "./td[2]/a/b"
- ListPrice: "./td[2]/table/tbody/tr[1]/td[1]/table/tbody/tr[1]/td[2]/s"
- Price: "./td[2]/table/tbody/tr[1]/td[1]/table/tbody/tr[2]/td[2]/span"
- Saving and SavingPercent: "./td[2]/table/tbody/tr[1]/td[1]/table/tbody/tr[3]/td[2]/span"
- Content: "./td[2]/table/tbody/tr[1]/td[2]/span"

B. rtvslo.si

- Author: "//div[@class='author-name']"
- Title: "//h1"
- SubTitle: "//div[@class='subtitle']"
- PublishedTime: "//div[@class='publish-meta']"
- Lead: "//p[@class='lead']"
- Content: "//article[@class='article']"

C. bolha.com

The Type, Location, and Status fields were extracted relative to the path: "//table[@class='table-summary table-summary–alpha']/tbody".

- Title: "//h1[@class='entity-title']"
- Price: "//li[@class='price-item price-item–base']/strong"
- Id: "//b[@class='base-entity-id']"
- PublishedTime: "//li[@class='meta-item meta-item–hidden']/time"
- ValidUntil: "//li[@class='meta-item meta-item–hidden']/span/span[@class='base-entity-display-expires-on']"
- Views: "//li[@class='meta-item meta-item–hidden']/span/span[@class='base-entity-display-count']"
- Type: "./tr[1]/td"
- Location: "./tr[2]/td"
- Status: "./tr[3]/td"
- Content: "//div[@class='passage-standard passage-standard–alpha']/div/p"

V. Automatic web extraction algorithm

Our final goal was to implement an automatic web extraction algorithm. We based our implementation on the Road Runner algorithm [1]. In our case, the purpose of the algorithm was to examine two HTML pages and return a regular expression, which can then be used for extracting data. Before the algorithm is run, our implementation provides some preprocessing on the selected pages. This is done by removing unnecessary and incorrect HTML tags, comments and scripts from the code. The remaining tags and page content are saved in list form and then processed further by the algorithm.

The core of the algorithm is the comparison function compare_pages. Since the goal of the implementation is to build a regular expression, the function builds a list of attributes that allows an expression to be built. This output is constructed by comparing list elements of the input pages, where certain rules are taken into account. In the simple case where two elements between pages match, the element is added to the output list, while the function moves to the next element comparison.

Cases where elements do not match require more processing, where procedure differs based on the type of elements that are being compared. Unsuccessful comparison of two content strings or a content string and an HTML tag are processed similarly, whereby we add an "unknown" expression **(.*?)** to the output instead. In the first case, the next comparison is made between two new page elements. When comparing a string with an HTML tag, we retain the tag and compare it with the next element from the list of elements of the other page.

In the case of unsuccessful matching between HTML tags, the algorithm compares more HTML tags to determine similarity. First, the algorithm finds the two tags that directly preceded the ones currently being compared. In the case where the tags' names match, they may each form their own code block if the ending tag precedes the starting tag. These blocks are then compared, and the implementation determines whether they match in every included tag. If the match is successful, every occurrence of this block in the output list is replaced with an iterator regular expression. If the match is unsuccessful, the implementation attempts to find the next HTML tag not included in the current block. If this tag is also the other page's next tag, the current block is added to the output list.

Finally, the implementation ends by constructing a regular expression from the output list of tags and strings. All tags are handled separately and are transformed into attribute-friendly expressions (** ** becomes **<div.*?>**). The implementation was successful in constructing a regular expression for the rtvslo pages, however failed to produce good results for both bolha and overstock pages.

References

[1] V. Crescenzi, G. Mecca, and P. Merialdo, "Roadrunner: Towards automatic data extraction from large web sites," 09 2001.