Webowy system do automatycznego gromadzenia korpusów tekstu z Internetu i ich ręcznej anotacji meta danymi

Tytuł w języku angielskim: Web-based System for Gathering Text Corpora from Internet and Facilitating Manual Annotation of the Corpora

Spis treści

[Streszczenie 4](#_Toc452482321)

[Synopsis 4](#_Toc452482322)

[Wstęp: Cel, motywacje, ograniczenia 5](#_Toc452482323)

[Motywacje 5](#_Toc452482324)

[Cel pracy 5](#_Toc452482325)

[Opis problemu 6](#_Toc452482326)

[Ograniczenia pracy 6](#_Toc452482327)

[Kluczowe pojęcia i terminy używane w tej pracy 7](#_Toc452482328)

[Korpus językowy 7](#_Toc452482329)

[Anotacja 7](#_Toc452482330)

[Crawler 7](#_Toc452482331)

[Drzewo DOM 7](#_Toc452482332)

[Selektor 7](#_Toc452482333)

[Algorytm pozycyjny 7](#_Toc452482334)

[CCL 7](#_Toc452482335)

[Premorph 8](#_Toc452482336)

[Inforex 8](#_Toc452482337)

[Corpo-grabber 8](#_Toc452482338)

[WCRFT 9](#_Toc452482339)

[Blog Reader 9](#_Toc452482340)

[Przegląd literatury 10](#_Toc452482341)

[Internet Documents: A Rich Source for Spoken Language Modeling 10](#_Toc452482342)

[Efﬁcient Web Crawling for Large Text Corpora 10](#_Toc452482343)

[A DOM Tree Alignment Model for Mining Parallel Data from the Web 11](#_Toc452482344)

[Corpo-grabber 12](#_Toc452482345)

[Dlaczego skrypty PHP? 12](#_Toc452482346)

[Elementy 12](#_Toc452482347)

[Interfejs 12](#_Toc452482348)

[Strona główna 13](#_Toc452482349)

[Pobieranie pojedynczej strony 13](#_Toc452482350)

[Pobieranie zaawansowane 13](#_Toc452482351)

[Ekstrakcja wzorca 13](#_Toc452482352)

[Działanie 14](#_Toc452482353)

[Algorytm pobierania: pojedyncza strona 14](#_Toc452482354)

[Algorytm pobierania: wiele stron 15](#_Toc452482355)

[Algorytm deduplikacji 15](#_Toc452482356)

[Algorytm tworzący drzewo 15](#_Toc452482357)

[Oznaczanie elementów drzewa 16](#_Toc452482358)

[Algorytm analizy drzewa 17](#_Toc452482359)

[Algorytm przechodzenia drzewa (algorytm pozycyjny) 17](#_Toc452482360)

[Alternatywny algorytm: algorytm selektorowy 19](#_Toc452482361)

[Algorytm mieszany 19](#_Toc452482362)

[Algorytm ekstrakcji tekstu i podziału na akapity 20](#_Toc452482363)

[Post-processing 20](#_Toc452482364)

[Badanie 22](#_Toc452482365)

[Teoria 22](#_Toc452482366)

[Praktyka 23](#_Toc452482367)

[Wyniki badań 24](#_Toc452482368)

[Wnioski z badań 24](#_Toc452482369)

[Podsumowanie 25](#_Toc452482370)

[Czego się nie udało? Dlaczego się nie udało? 25](#_Toc452482371)

[Kierunki rozwoju 26](#_Toc452482372)

[Źródła 26](#_Toc452482373)

[Załączniki 27](#_Toc452482374)

[Załącznik 1: płyta DVD 27](#_Toc452482375)

[Załącznik 2: Badania 28](#_Toc452482376)

**Streszczenie**

Celem tej pracy jest opisanie rozwiązania problemu, jakim jest skonstruowanie narzędzia do ekstrakcji korpusów lingwistycznych z zasobów internetowych, jak na przykład blogi, strony informacyjne, itp. Narzędzie to ma być przystępne dla użytkownika nieobytego w technologiach informatycznych. Przedstawię jakie rozwiązania do tej pory się pojawiły, moją propozycja rozwiązania problemu oraz analizę, czy moje rozwiązanie faktycznie może być aplikowane do tego problemu.

W ramach rozwiązania problemu postanowiłem podejść do niego z innej strony niż zwykle. Mając doświadczenie w tworzeniu stron internetowych wiem, że jeżeli rozpatrujemy strukturę drzewiastą wielu stron internetowych z tej samej domeny, to konkretne elementy, które nas interesują, są zazwyczaj tymi samymi gałęziami w drzewie strony. W związku z tym chciałem sprawdzić teorię, czy można dokonywać ekstrakcji tekstu ze stron internetowych znając ich dokładne położenie w drzewie DOM, co określiłem mianem algorytmu pozycyjnego. Badanie potwierdzające moją teorię przeprowadziłem porównując moją metodę ekstrakcji tekstu z metodą konkurencyjną, która polega na ekstrakcji elementów ze strony internetowej przy pomocy selektorów CSS.

**Synopsis**

In this paper I shall present a solution to the problem of constructing tool for extracting linguistic corpora from internet resources, like blogs, informational sites, etc. The tool must be handy for a user that isn’t common with informational technologies. I shall present what kind of solutions have been created, my solution to the problem, and analysis if my solution can actually be used to resolve this problem.

Regarding my problem, I decided to resolve it from different approach, compared to what was used before. Having experience in web design, I know that when we interpret many web pages from the same domain as tree structure, specific elements that we could be interested in are usually the same branches in these trees. Because of that I decided to test my theory, if it is possible to extract text from web pages with the knowledge of the exact placement of text in DOM tree, which I called the branch position algorithm. The research that confirmed my theory were done by comparison to another method, that involves in extracting elements of web page using CSS selectors.

# Wstęp: Cel, motywacje, ograniczenia

## Motywacje

Analiza syntaktyczna języka jest jedną z ważnych dziedzin informatyki. Przede wszystkim może nam służyć do analizy części mowy, zapis (może być różny, w zależności od akcentu), regularności, kolejności słów w zdaniu, znaczenia i problemów związanych z językiem. Tymi i innymi problemami zajmuje się dział nauki zwany lingwistyką, ale również lingwistyka informatyczna i inżynieria języka naturalnego.

Lingwiści do analizy języka pisanego - czy mówionego - potrzebują mieć materiały do analizy. Często są to nagrania audio, ich transkrypcje, książki, czasopisma, nagrania video i inne. Do celów analizy przy pomocy narzędzia informatycznego potrzebna jest ich wcześniejsza cyfryzacja. Nagrania audio muszą być transkrybowane na medium cyfrowe, transkrypcje w formie analogowej muszą zostać przeanalizowane przy pomocy programu do optycznej interpretacji tekstu (Optical Character Recognition), tak samo czasopisma czy książki. Jednakże, cyfryzacja to nie wszystko: często typowe formaty transkrypcji danych są niewystarczające. Do tego celu teksty w formie cyfrowej muszą zostać przekształcone do formatu pliku zwanego korpusem.

Korpus językowy to skompilowany zbiór danych tekstowych dostępnych w formie elektronicznej, stanowiący materiał do badań. Jest zawsze w formie cyfrowej, czasem ten plik posiada również wstępną analizę syntaktyczną tekstu w nim zawartego. Politechnika Wrocławska opracowała własny format pliku obsługującego korpusy. Nazywa się on CCL i oprócz przechowywania korpusów, przechowują również anotacje istotne w procesie analizy lingwistycznej.

Lingwistyka informatyczna boryka się z wieloma problemami, jest to np. tworzenie modelu języka w oparciu o ciągle zmieniające się zasady. Jednym z nich jest pozyskiwanie materiałów do analizy, a przede wszystkim ich trudnodostępność. Jednakże jest miejsce, w którym są duże zasoby tekstu do analizy lingwistycznej, z której polscy naukowcy do tej pory nie czerpali bardzo. Jest nią internet.

Nie jest problemem pozyskiwanie korpusów z internetu. Każdy może wejść na dowolną stronę internetową i pobrać jej zawartość, wyszczególnić tekst i przekonwertować na format korpusu. Problemem jest natomiast pozyskanie dużych ilości tekstu w opraciu o pojedynczy, ogólny adres internetowy. Praca ta ma na celu przybliżenie algorytmów, które rozwiązują ten problem.

Praca ta również ma na celu przyjrzenie się konkurencyjnym algorytmom, które służą do gromadzenia korpusów językowych ze stron internetowych. Mam nadzieję, że dzięki tej pracy przyszłe pokolenia naukowców z dziedziny lingwistyki będą mogli korzystać z mojego narzędzia bez problemów, oraz że będą rozumieli jego działanie.

## Cel pracy

Celem pracy jest opracowanie systemu – narzędzia badawczego – do gromadzenia korpusów tekstowych z internetu, gdzie korpus tekstowy to duży zbiór tekstów gromadzonych w celach badawczych w ramach różnych dziedzin nauki. System powinien pracować automatycznie, ale pod kontrolą naukowca. System powinien umożliwić zdefiniowanie zakresu pobieranych danych, wydobycie tekstu wraz z jego semantyczną strukturą z dokumentów (np. z podziałem na akapity, nagłówki itd.), zapis do stosowanych formatów korpusowych, w tym formatu CCL opracowanego w Grupie Naukowej G4.19 PWr. oraz późniejszy opis tekstów meta-danymi na różnych poziomach, np. dokumentów, zdań, słów itd.

## Opis problemu

W celu pracy opisuje się następujące elementy, jak automatyczna praca, ale pod kontrolą naukowca. To oznacza zdefiniowanie, które elementy będą automatyczne, a które pod kontrolą naukowca. Przyjąłem, że automatyczne w systemie będzie pobieranie zasobów internetowych, a pod kontrolą naukowca będzie wyznaczenie wzorca ekstrakcji.

Dodatkowo ważnym problemem w pracy jest zdefiniowanie takich reguł, które dla każdego naukowca będą jasne i klarowne. To wywiera ograniczenia na narzędziu, jak:

1. Jasne i klarowne reguły
2. Prosty interfejs

To wyklucza wykorzystanie metod ekstrakcji tekstu, jak definiowanie reguł przy pomocy selektorów.

## Ograniczenia pracy

Ograniczyłem się jedynie do zdefiniowania nowej metody analizowania strony internetowej (pozycyjnej). Porównuję to podejście z konkurencyjnym podejściem (według selektorów). W ramach badania analizuję oba podejścia na przykładzie wybranych blogów internetowych, oraz dwa podejścia mieszane. Badania ograniczyłem do ekstrakcji następujących elementów:

1. Notki – element opisujący wszystkie niżej wymienione elementy
2. Autor, tytuł, data wpisu i tekst wpisu

Do ekstrakcji tekstu uznałem, że element notki jest konieczny do wyznaczenia w celu wykrywania nieprawidłowości w strukturze.

Do badania wybrałem około 40 blogów i stron internetowych, które najpierw pobierałem, a potem ręcznie oznaczałem. Blogi wybierałem samodzielnie z listy stron, które były problematyczne przy algorytmie konkurencyjnym.

Narzędzie opracowywałem na platformie Windows, w związku z tym, część narzędzi skryptowych, jak skrypty linuxowe opracowane przez pracowników Politechniki Wrocławskiej, nie mogą zostać przeze mnie użyte i muszę wówczas opracować alternatywny algorytm.

# Kluczowe pojęcia i terminy używane w tej pracy

## Korpus językowy

Potrzebne fajne źródło słownikowe

## Anotacja

Potrzebne fajne źródło słownikowe

## Crawler

Potrzebne fajne źródło słownikowe

## Drzewo DOM

Potrzebne fajne źródło słownikowe

## Selektor

Potrzebne fajne źródło słownikowe

## Algorytm pozycyjny

Jest to opracowane przeze mnie podejście do analizy strony internetowej. Zamiast znajdywać każdy element na stronie jako selektor o znanym typie, klasie bądź identyfikatorze, oznaczam każdy element jako ścieżka, jaką trzeba przebyć od korzenia do tego elementu. Pozwala to podejście definiowanie łatwych i jednoznacznych reguł w przypadku:

1. Nieistniejących nazw klas bądź identyfikatorów dla konkretnych elementów
2. Gdy element jest tej samej klasy lub (co jest błędne według W3C) ma ten sam identyfikator w drzewie
3. Gdy tekst jest oddzielony znacznikiem nowej linii („br”) a interesująca nas dana jest w jednym akapicie lub drugim

Ale o tych zaletach więcej opiszę we wnioskach z badań.

## CCL

CCL jest to format korpusowy opracowany na Politechnice Wrocławskiej. CCL jako format powstał w opraciu o format XCES (który w dużej mierze służy również do gromadzenia i anotacji korpusów lingwistycznych). Jest to plik tekstowy w formacie XML. Poza przechowywaniem tekstu źródłowego, format ten może przechowywać interesujące lingwistów własności tekstu, takie jak:

1. Podziały na akapity i zdania
2. Podziały na tokeny i informacje bez-spacjowe
3. Anotacje morfosyntaktyczne
4. Anotacje do styli fragmentów (chunk-style) z możliwymi dyskontynuacjami
5. Nagłowki syntaktyczne anotacji
6. Właściwości tokenów – a co za tym idzie – anotacji

<http://nlp.pwr.wroc.pl/redmine/projects/corpus2/wiki/CCL_format>

## Premorph

Istnieje wiele form pośrednich pomiędzy dowolnym dokumentem zawierającym tekst a formatem CCL. Można zwykły dokument tekstowy podzielić na akapity i zapisać w formacie tekstowym. Można dokument podzielony na akapity zapisać w formacie html (z podziałem na akapity przy pomocy elementu „p”). Istnieje jeszcze format premorph, z którego korzystałem.

Nazwa premorph pochodzi z właściwości, jaką ma ten format: jest to plik XML sformatowany w podobny sposób, jak pliki CCL z jedną różnicą: nie ma podziału na słowa i nie ma anotacji morfo-syntaktycznej. Plik premorph różni się od pliku CCL następującymi elementami

1. Podział na akapity
2. Brak podziału na zdania, tokeny.
3. Nie zawiera jakichkolwiek anotacji morfo syntaktycznych.

A co je łączy?

1. Zapis w XML-u
2. Podział na listy chunków (akapity)
3. Nazwa akapitu (chunku)

## Inforex

Inforex jest narzędziem utworzonym na Politechnice Wrocławskiej. Służy ono do analizy morfo syntaktycznej korpusów w formacie CCL, oraz do ręcznej anotacji elementów mowy jak nazwy jednostkowe, anafory, nadawanie sensu słowom niejednoznacznym i wszelkiego rodzaju relacje pomiędzy jednostkami. System również wspiera ręczne czyszczenie tekstu oraz automatyczną analizę, w której się zawiera segmentacja tekstu, analiza morfo syntaktyczna, selekcja słów do późniejszego nadania znaczenia w przypadku słów niejednoznacznych.

Marcińczuk, Michał; Kocoń, Jan; Marcin, Ptak and Kaczmarek, Adam, 2010, *Inforex*, CLARIN-PL digital repository, <http://hdl.handle.net/11321/13>.

Nie wykluczone, że będę wspominał niejednokrotnie o tym systemie na przestrzeni tego dokumentu, ze względu na to, że główną inspiracją do napisania tej pracy był właśnie system Inforex. Z tego też powodu moje narzędzie będzie konwertowało dokumenty do formatu CCL.

## Corpo-grabber

Jest to narzędzie, podobnie jak moje, służące do automatycznego gromadzenia korpusów internetowych z Internetu. Korzysta ono z takich narzędzi jak:

1. Httrack – do pozyskiwania stron internetowych
2. Fdupes – Do deduplikacji tekstów identycznych
3. Onion – Do deduplikacji tekstów podobnych.
4. WCRFT – Do konwersji tekstu podzielonego na akapity do formatu CCL. Więcej o nim w podrozdziale.

Moje narzędzie opierałem na corpo-grabberze, nawet je podobnie nazywałem. W początkowej fazie projektu korzystałem zarówno z httracka i narzędzi deduplikacyjnych takich samych jak w tym projekcie, jednakże z czasem i doświadczeniem zacząłem korzystać z innych.

## WCRFT

Wrocław CRF Tagger – jest to narzędzie stworzone na Politechnice Wrocławskiej, tak jak powyższe narzędzia, do analizy morfo syntaktycznej tekstu. Korzysta z algorytmu CRF: Conditional Random Fields, który służy do rozpoznawania wzorców i w nauczaniu maszynowym. Ma szereg różnych zastosowań, jednym z nich jest łatwa konwersja tekstów w języku polskim na korpusy lingwistyczne, z której korzysta moje narzędzie w formie finalnej.

<https://en.wikipedia.org/wiki/Conditional_random_field>

<http://nlp.pwr.wroc.pl/redmine/projects/wcrft/wiki>

## Blog Reader

Jest to kolejne narzędzie opracowane na Politechnice Wrocławskiej, służące do ekstrakcji tekstu ze stron internetowych już pobranych. Korzysta z podejścia selektorowego do ekstrakcji tekstu. Podejście jest proste: definiujemy elementy, jakie chcemy pobrać ze strony i nadajemy im selektor CSS, według którego należy wyszukiwać tych elementów, które nas interesują. Informatycy są w stanie bez problemu wyznaczyć wystarczające reguły do opracowania selektora, jednakże osoby nieobyte w technologiach IT nie były w stanie ich zdefiniować, co było wielką wadą tego programu.

Moje narzędzie opierałem na Blog Readerze do celów badawczych. Była to najlepsza znana mi metoda ekstrakcji tekstu, gdzie chcemy pozyskać konkretną treść ze stron internetowych.

# Przegląd literatury

Literatura poruszająca problem pozyskiwania korpusów tekstu z internetu jest uboga, jednakże jest wiele prac naukowych dotykających problematyki zautomatyzowanego pobierania zasobów internetowych, oraz dotyczących wykorzystania korpusów tekstu w różnych dziedzinach wiedzy. W tym rozdziale postaram się przedstawić najistotniejsze źródła, z których korzystałem przy pracy, opiszę co zawierają oraz opiszę stosunek tych prac z moją pracą.

## Internet Documents: A Rich Source for Spoken Language Modeling

W tej pracy zaproponowano metodologię pozyskiwania dokumentów internetowych w celu pozyskania tekstów do modelowania języka mówionego. Aktualne modele językowe często są oparte na tekstach pisanych i/lub nadużywanym Czarnoksiężniku z krainy Oz, albo na eksperymentalnych dialogach. Przy pomocy tekstów pozyskanych z internetu, potem przeczytanych przez eksperymentatorów, a następnie przeanalizowanych przez algorytmy rozpoznawania mowy otrzymano 15% wzrost skuteczności w rozpoznawaniu słów.

Moja praca będzie w dużej mierze będzie opierała się na pozyskiwaniu tekstów z internetu. Moim zadaniem jednak będzie przekształcanie ich na korpusy tekstów w formacie CCL, gdzie praca ta opierała się głównie na pozyskaniu tekstów do celów późniejszego ich odczytania. Analiza morfo-syntaktyczna byłaby im niepotrzebna.

## Efﬁcient Web Crawling for Large Text Corpora

Praca ta skupia się na efektywnym pobieraniu zasobów internetowych. Autorzy proponują różne rozwiązania, które usprawniają proces pobierania, ale również umożliwiają przy pomocy różnych metod umożliwić łatwą ekstrakcję tekstu. Autorzy mieli dwie główne motywacje: pobieranie danych ze źródeł zawierających najwięcej treści oraz możliwe unikanie duplikatów danych.

Pierwszy problem, czyli uzyskiwanie lepszych wyników pobierań, rozwiązali poprzez napisanie własnego crawlera, którego nazwali pajączek (spiderling). Zdefiniowali zależność pomiędzy ilością danych pobranych, ilością danych istotnych (czyli „gołego" tekstu) oraz ilością dokumentów pobranych. Wyznaczenie tej zależności umożliwiło im zredukowanie czasu pobierania danych oraz zwiększyło stosunek danych istotnych do danych pobranych (yield rate) poprzez odrzucanie domen, z których nie dało się wyekstraktować wystarczającej ilości danych.

Drugi problem, jakim jest unikanie duplikatów danych, uzyskali przez napisany przez nich program – jusText. Dokonywali deduplikacji w dwóch momentach: w momencie pobierania danych i po ekstrakcji tekstu. Nadawali dokumentom sumy kontrolne, które porównywali z pozostałymi dokumentami. W przypadku duplikatu, jeden z dokumentów usuwali a sumę zapisywali. Korzystali również z programu onion do deduplikacji tekstów zbliżonych.

W moim rozwiązaniu korzystam z gotowych rozwiązań do crawlowania. Nie potrzebuję wykrywać, czy strona zawiera treść. Zakładam, że naukowiec korzystający z mojego narzędzia wie jakiej treści oczekuje ze strony pobieranej, i że pobiera ją nie bez celu. Dlatego zależności wyznaczone przez autorów nie będą mnie interesowały.

Drugie rozwiązanie z ich pracy jest dla mnie przydatne. Duplikaty danych występują niemalże na każdej stronie internetowej. Choćby przykładem niech będą notki na blogach, które pojawiają się zarówno jako osobne strony, na stronach z komentarzami, oraz na stronach archiwum. Proces deduplikacji zatem jest potrzebny dla mnie po pobieraniu danych, ale również przed, jak i po analizie danych.

## A DOM Tree Alignment Model for Mining Parallel Data from the Web

Pracę tę znalazłem po sformułowaniu mojego problem. Opisuje ona inny problem niż mój na podstawie podobnych do moich założeń.

W tym dokumencie jest postawiona teza, że strony internetowe dwujęzykowe często posiadają taką samą strukturę strony i linków, niezależnie od języka, który się wybierze. Autorzy zbadali i zauważyli, że 10% spośród 150 tysięcy stron internetowych z domeny .de jest dwujęzykowa. W związku z tymi obserwacjami, powstało wiele crawlerów służących do pozyskiwania danych dwujęzycznych. Między innymi twórcy tej pracy, którzy zastosowali podobieństwo stron internetowych do ekstrakcji zdań w obu językach.

Ja również korzystam z tego podejścia. Jest ono bardzo wygodne i zastanawiające jest, że nikt nie zrobił tak prostego badania, jakim jest sprawdzenie jednolitości struktury stron internetowych.

# Corporanet

Corporanet jest moim podejściem do tematu, jakim jest rozwinięcie istniejącej już architektury na Politechnice Wrocławskiej. Moje narzędzie jest webowym skryptem w PHP. Ma ono za zadanie przy dyrektywach użytkownika pobrać wybraną stronę internetową, następnie po zakończonym procesie pobierania ma dokonać wstępnej deduplikacji stron internetowych, po czym wyświetla pobrany projekt. Następnie zadaniem użytkownika jest dla przykładowej pobranej strony internetowej zdefiniowanie interesujących go elementów strony. Po tym, mój program po głuższej analizie prezentuje wyniki pobierania.

## Dlaczego skrypty PHP?

Odpowiedź jest bardzo prosta. Podczas analizy strukturalnej strony internetowej potrzebna jest idealna wizualizacja strony internetowej. W związku z tym potrzebne jest narzędzie, które umie idealnie prezentować dane. Jest nią przeglądarka internetowa. W związku z tym warstwa prezentacyjna strony internetowej powinna być realizowana przy pomocy skryptów PHP.

Jednakże fragment części logicznej strony jest również zaprogramowana przy pomocy skryptów PHP. Wynika to z bardzo prostego faktu – do języka PHP napisanych zostało wiele interesujących skryptów, które są w stanie analizować drzewa DOM. Korzystam z nich i są bardzo pomocne. Jednakże do celów optymalizacyjnych powinno się te narzędzia napisać w zewnętrznym skrypcie, jak w języku Java (dla pracy na różnych systemach) czy Python (by konturować trendy na Politechnice Wrocławskiej).

## Elementy

Program dzieli się na parę elementów, które są wyznaczone przez interfejs:

1. Część dotycząca pobierania
2. Część dotycząca oznaczania stron

Z czego część druga ma jeszcze następujące podczęści:

1. Część dotyczącą wyboru podstrony
2. Podgląd podstrony
3. Widok drzewa podstrony
4. Analiza drzewa podstrony

Program korzysta z takich zewnętrznych programów jak:

1. WGET – program służący do pobierania stron internetowych
2. PHPHtmlParser – wtyczka PHP służąca do analizy drzewa DOM
3. Simple MVC - Prosta biblioteka MVC do PHP.

WGET jest wykorzystywany tylko w części pierwszej. Natomiast PHPHtmlParser jest wykorzystywany w części drugiej w podczęści 3 i 4. Sens takiego wykorzystania opiszę w podrozdziale „Działanie”.

## Interfejs

W ciągu całego procesu tworzenia oprogramowania korzystałem z wielu różnych podejść do tworzenia strony internetowej. Chciałem z początku podejść do tematu przy pomocy bardzo ubogiego interfejsu. Chciałem po prostu umożliwić wybieranie strony internetowej do pobrania i wyświetlić wynik. Jednakże wkrótce po tym okazało się, że nie da się spełnić wszystkich oczekiwań na prostym interfejsie. W związku z tym skorzystałem z biblioteki MVC jaką jest Simple MVC. Umożliwiła mi ona proste tworzenie interfejsu i prostej logiki.

W związku z tym, postanowiłem nagłówek przeznaczyć na łącza do podstawowych części strony:

1. Strona główna
2. Pobieranie pojedynczej strony
3. Pobieranie wielu stron z jednej domeny (Pobieranie zaawansowane)
4. Anotacja strony pobranej (Ekstrakcja wzorca)

### Strona główna

Zawiera krótką adnotację dotyczącą przeznaczenia strony internetowej. Opisuje to, co możemy znaleźć na konkretnych podstronach.

### Pobieranie pojedynczej strony

Z początku myślałem, że pobieranie pojedynczej strony internetowej będzie czymś istotnym na stronie internetowej. W związku z tym, po wprowadzeniu adresu strony algorytm automatycznie pobierał wybraną stronę internetową a następnie wyświetlał efekty ekstrakcji przy predefiniowanych parametrach. Jednakże nie były to parametry, które interesowały językowców, dlatego musiałem zrezygnować z tego pomysłu. Nadal jednak istnieje w projekcie.

### Pobieranie zaawansowane

Jest to efekt pracy związany z adaptacją corpo-grabbera do projektu webowego. Interfejs zawiera podstawowe elementy, jak:

1. Nazwa projektu
2. Adres URL

W wersjach wcześniejszych występował jeszcze element trzeci: czas pobierania. Wynikało to ze specyfiki narzędzia, które wykorzystywałem. Z początku korzystałem z narzędzia zwanego Httrack (tak samo jako pierwotne narzędzie Corpo-grabber). Jednakże w pewnym momencie przestało spełniać moje oczekiwania, dlatego zacząłem korzystać z innego narzędzia, jakim jest WGET.

Element „nazwa projektu” pozwalał nam zdefiniować nazwę projektu do pobierania. Nie jest to element obowiązkowy, jako że w przypadku nie zdefiniowania nazwy projektu, moje narzędzie definiuje domyślną nazwę jako (kolejno): datę pobierania i stronę internetową (pozbawioną prefiksu http:// i www., oraz wszystkich nielegalnych znaków pod systemem Windows).

Element „adres URL” jest obowiązkowy. Służy ono do zdefiniowania adresu pobierania treści. Od tego adresu rozpoczynane jest pobieranie a domeną treści jest uznawane wszystko przed pierwszym wystąpieniem w adresie znaku ukośnik „/”

### Ekstrakcja wzorca

Jest to efekt mojej samodzielnej pracy. Element ten zawiera 4 główne części:

1. Menu wybierania projektu i podstrony
2. Widok podglądu strony
3. Widok drzewa strony
4. Menu boczne

Ogólnie, ta podstrona pozwala nam na wyznaczenie wzorca, według którego możemy dokonać ekstrakcji danych ze strony. Wszystkie te elementy służą temu celu.

Menu wybrania projektu i podstrony pozwala nam na zdefiniowanie projektu, który nas interesuje (można wybrać każdy przeszły, jeżeli nasze reguły kiedyś nam się nie spodobały). W tym menu również możemy zdefiniować, którą stronę chcemy wybrać, aby było widać w widoku podglądu strony i drzewie strony.

Menu boczne jest zawiera informacje i metody pozwalające oznaczać interesujące nas elementy. W formie badawczej występują tam dwa elementy: przyciski do zaznaczania elementów na drzewie strony oraz pola tekstowe do (odpowiadających tym przyciskom) wyznaczania reguł ekstrakcji elementów z drzewa. Służyło to głównie porównaniu metod ekstrakcji: mojej i konkurencyjnej.

W formie ostatecznej, która zostanie przekazana Politechnice Wrocławskiej do celów dalszego rozwoju, nie będzie już w menu bocznym pól tekstowych. Służyły one jedynie badaniu i ich istnienie nie ma sensu w praktycznym użyciu.

## Działanie

Tak jak wspomniałem w rozdziale o interfejsie i elementach, korzystałem z wtyczki Simple MVC do administrowania strukturą model – widok – kontroler. Rozdzieliła wówczas elementy strony na podkatalogi (w katalogu app):

1. Controllers – zawieta kontrolery
2. Core – zawiera dane konfiguracyjne, w tym definicję możliwych adresów na stronie
3. Templace i Views – zawierające widok strony z różnych perspektyw
4. PHPHtmlParser – zawiera wtyczkę do obsługi drzewa DOM
5. Pozostałe katalogi – zawierają dane potrzebne przez skrypty Simple MVC

Dodatkowo moje narzędzie wyznaczyło dodatkowy katalog, jakim jest katalog tmp. Ten katalog zawiera wszystkie pobierania, jakie wykonali użytkownicy. W każdym katalogu znajduje się katalog Web. Jest to efekt wstecznej kompatybilności z programem httrack, który podczas pobierania stron, pliki html umieszczał w katalogu web a pozostałe dane, jak obrazki umieszczał w innych katalogach. W katalogu web po dokonaniu analizy drzewa moim algorytmem jest tworzony katalog premorph, w którym umieszczone są wszystkie utworzone wówczas pliki xml a następnie pakowane do pliku zip do downloadu przez użytkownika. Dodatkowo, jeżeli zostały zdefiniowane parametry do badania, w katalogu dla każdego badania pojawiał się osobny plik zip oraz plik report.html, który zawierał dane szczegółowe dotyczące badania.

### Algorytm pobierania: pojedyncza strona

Algorytm ten polega na skorzystaniu z wbudowanej biblioteki PHP zwanej CURL. Przy odpowiednich parametrach, biblioteka ta po chwili zwraca wynik pobierania dla zadanej strony. Następnie przy pomocy algorytmu ekstrakcji tekstu wyciągam wszystkie elementy tekstowe ze strony z „podziałem na akapity”. Podział na akapity jest tu w cudzysłowie, gdyż algorytm dzielenia na akapity w przypadku analizowania całej strony wyciąga wówczas elementy niepożądane, na przykład wszystkie linki z paneli bocznych są traktowane jako osobny akapit.

### Algorytm pobierania: wiele stron

Algorytm ten polega na wywołaniu zewnętrznego programu zwanego WGET. WGET ma wiele opcji do pobierania, jedną z nich jest między innymi pobieranie rekursywne, gdzie parametrem ograniczającym zagłębianie się w linki jest głębokość rekursji. Jednakże w ramach działania programu nie zmieniałem go i ma on wartość 5.

Oto polecenie, przy pomocy którego wywołuję program wget w programie:

*"wget --recursive -nd --page-requisites --html-extension --convert-links restrict-file-names=windows -P \"$path\" -A html,htm -w 1 --random-wait domains $URL\_short –no-parent $URL -o \"$path\_log\" -nv"*

Gdzie:

1. $path – to ścieżka absoultna katalogu, do którego jest pobierana strona
2. $URL – adres strony taki, jaki został wprowadzony przez użytkownika
3. $URL\_short – adres strony sformatowany przeze mnie do celów uzyskania domeny przeszukiwań. Usuwam prefiks „http://”, „www.”, oraz sufiks „/(…)”.
4. $path\_log – adres pliku, do którego należy zapisywać logi programu.

Dodatkowo, przy pomocy oprogramowania jQuery wykonuję cykliczne zapytanie do serwera, aby pokazał plik logów. Serwer w tym czasie dokonuje deduplikacji danych i przedstawia logi programu w odwróconej kolejności linii (dzięki czemu ostatni linie z logów pojawiają się jako pierwsze na stronie).

W przypadku wyłączenia strony w przeglądarce katalog nie jest analizowany w kierunku deduplikacji, co może poważnie wydłużyć czas otwierania projektu przy późniejszej próbie deduplikacji strony.

### Algorytm deduplikacji

Jest on wywoływany w kilku momentach: w trakcie pobierania oraz w trakcie otwierania projektu. W programie został zastosowany podstawowy mechanizm deduplikacji. W przypadku wystąpienia dwóch plików w tym samym projekcie o tej samej wartości hashu MD5, jeden z nich jest usuwany. W przypadku identycznego rozmiaru plików, jeden z nich jest usuwany.

Dodatkowo, żeby jeszcze zmniejszyć rozmiar plików i ułatwić ich podgląd usuwane z plików html są wszystkie skrypty. Zostawia to wówczas drobne artefakty na stronie, ale nie są one zbyt wielką przeszkodą.

### Algorytm tworzący drzewo

Jest bardzo prosty. Tworzę drzewo html poprzez rekursję: aktualny element otrzymuje od elementu nadrzędnego ścieżkę, w której ten element się znajduje. Wówczas on zapisuje sobie, że jest elementem o identyfikatorze [ta ścieżka].

Pojedynczy element w ścieżce jest zapisywany poprzez zapisanie typu elementu, jego wszystkich klas (oddzielonych spację pomiędzy sobą i kropką od typu) i ilości elementów danego typu na ścieżce napotkanych spośród dzieci tego samego rodzica (oddzielonych dwukropkiem). Przykładowo, selektor „div” o klasie „class” będący drugim elementem typu „div” u danego rodzica będzie miał w ścieżce adnotację o następującym wyglądzie:

*div.class:2*

W przypadku, gdy dziecko jest elementem tekstowym, zamiast typu jest ten element oznaczany jako typ „text”.

Każdy z elementów w ścieżce rozdzielam myślnikiem. To wymusza na mnie, że wszystkie elementy w ciągu ścieżki nie mogą zawierać ani kropki, dwukropka ani myślnika. W przypadku ich wystąpienia, w kodzie ścieżki są one zastąpione podkreślenie: „\_”. Przykładowy element, będący już elementem drzewa html:

*root-html.v2:1-body.loading variant\_birds:1-div.navbar section:1-div.widget Navbar:1-div:1*

Jeżeli w tej gałęzi znajdują się podgałęzi (dzieci), to każdy element otrzymuje identyfikator z dopisanym elementem opisującym tę podgałąź. Dla przykładu, jeżeli dziećmi powyższej gałęzi są dwie tablice o klasach „table-inner” i „table-inner2”, to dzieci te otrzymają identyfikatory:

*root-html.v2:1-body.loading variant\_birds:1-div.navbar section:1-div.widget Navbar:1-div:1-table.table\_inner table\_inner2:1*

*root-html.v2:1-body.loading variant\_birds:1-div.navbar section:1-div.widget Navbar:1-div:1-table.table\_inner table\_inner2:2*

I zostaną umieszczone w drzewie strony kolejno z adnotacja tekstową zawierającą typ i klasę. Finalnie, przykładowy element wygląda następująco a dzieci pojawiłyby się pod selektorem „br”:

*<div id="root-html.v2:1-body.loading variant\_birds:1-div.navbar section:1-div.widget Navbar:1-div:1"><b>div</b><br></div>*

W przypadku, gdy dziecko jest elementem tekstowym jest umieszczane w osobnym „div”-ie, a nastepnie bez adnotacji elementu (jak powyżej, pogrubiony „div”) jest wypisywana treść tekstu. Umożliwia to potem łatwą nawigację po tekście.

Ten algorytm umożliwia mi utworzenie strony internetowej, w której elementy tekstowe są zagnieżdżone w strukturze, która jest zbliżona do tej na stronie. Co więcej, każdy element mojego drzewa zachowuje informację o tym, jakie elementy są po drodze. Dzięki temu oznaczając konkretny element jako zawierający dane, nie muszę przechodzić przez całe drzewo ponownie w celu zidentyfikowania jakie elementy są po drodze.

### Oznaczanie elementów drzewa

W momencie wygenerowania drzewa składającego się z elementów „div” możemy przejść do oznaczania drzewa i oddania użytkownikowi pola do działania. Z menu po prawej stronie wybiera element go interesujący i zaznacza, który element drzewa najlepiej opisuje jego dane. Jeżeli jest to notka, to powinien wybrać miejsce, które najlepiej opisuje wszystkie pozostałe elementy, jak autor, tytuł, datę wpisu i jego treść. Jeżeli znajdzie tytuł notki, powinien znaleźć ten element, który zawiera w sobie sam tytuł i nic więcej. Jeżeli znajdzie datę, to najlepiej, jakby otoczył zarówno element opisujący datę, jak i czas wpisu.

<OBRAZEK>

Obrazek powyższy idealnie ilustruje poprawne oznaczenie wpisu. Różne kolory opisują różne elementy. W przypadku, gdy jeden element html opisuje parę elementów naraz, na przykład po przecinku jest data i tytuł, wówczas musimy oznaczyć ten element dwoma znacznikami. W takim wypadku wygenerowany zostanie nie jednolity kolor, a gradient przechodzący pomiędzy elementami strony.

Jak wspomniałem w poprzednim rozdziale, notka jest najbardziej istotnym elementem na stronie. Bez niej nie da się pobierać istotnych danych, gdyż to względem notki będziemy analizowali strukturę wpisu. Zastosowałem takie ograniczenie, gdyż zdarzało mi się napotykać sytuacje, że element o zadanym selektorze pasował zarówno do komentarza jak i wpisu, co uniemożliwiało rozróżnienie ich w trakcie pobierania.

### Algorytm analizy drzewa

Algorytm ten jest uruchamiany w momencie, gdy oznaczyliśmy interesujące nas elementy na stronie i kliknęliśmy w przycisk zatwierdzający. Sprawdza się w tym momencie parę czynników:

1. Jeżeli wypełnione zostały pola odpowiadające selektorowemu podejściu, to będziemy realizować algorytm badawczy: dla wylosowanych 100 plików zostaje wywołany algorytm pozycyjny i selektorowy, oraz dwa podejścia mieszane. Więcej szczegółów w rozdziale o badaniach
2. Jeżeli nie została oznaczona notka, to program zwróci komunikat błędu. Powód znajduje się w poprzednim podrozdziale.
3. Sprawdza, czy zapytanie jest zaaplikowane do istniejącego / pobranego poprawnie projektu. W przypadku błędu zwraca komunikat.

W tym momencie następuje realizacja algorytmu pozycyjnego lub pozycyjnego, mieszanego i selektorowego. Po przeanalizowaniu wszystkich zadanych plików przy pomocy tych algorytmów zwracany jest wynik dla użytkownika w postaci strony internetowej zawierającej podsumowanie badania, lub (gdy oznaczone zostały pola tylko z algorytmu pozycyjnego) plik archiwum o rozszerzeniu ZIP zawierający pliki w formacie XML, które są w formacie korpusowym premorph.

Użytkownik mając te pliki może skorzystać z narzędzia jak WCRFT do przekonwertowania tych plików do formatu CCL.

Poniżej znajduje się omówienie metod wspomnianych wyżej.

### Algorytm przechodzenia drzewa (algorytm pozycyjny)

Na wejściu do tego algorytmu otrzymaliśmy listę plików i łańcuchy tekstów opisujące elementy na stronie do pobrania. Zatem dla każdego z plików próbujemy znaleźć notkę w drzewie. Kiedy ją odnajdziemy, staramy się znaleźć elementy, które zostały zdefiniowane (autor, tytuł, data i treść). W przypadku nie odnalezienia części, algorytm tej notki nie odrzuca.

Odnajdywanie elementów polega na przechodzeniu drzewa. Rozdzielamy łańcuch przejść względem myślników. W rezultacie otrzymujemy pojedyncze elementy. Na początku znajdujemy się w pozycji root. Następnie kolejno przechodząc po dzieciach sprawdzamy, czy element o zadanym typie znajduje się na danej pozycji. Gdy istnieje, przechodzimy dalej i sprawdzamy jego dzieci. Dla przykładu, skorzystajmy z tego przykładu do ilustracji algorytmu:

*root-html.v2:1-body.loading variant\_birds:1-div.navbar section:1-div.widget Navbar:1-div:1*

Zaczynamy od pozycji root. Następnie szukamy wśród korzenia drzewa, czy istnieje element „html”. Jeżeli jest, przechodzimy do niego. Następnie przechodzimy do pierwszego wystąpienia pozycji „body”, potem do pierwszego wystąpienia pozycji „div”, itd.

Zdecydowałem się na przechodzenie drzewa uwzględniając typ w pozycji, gdyż często napotykałem sytuację, gdzie w drzewie pojawiał się dodatkowy element, który był wynikiem delikatnie innej struktury, na przykład element „meta” pojawiający się w polu head, zamiast w polu html.

Zdecydowałem się na ignorowanie elementu klasy w drzewie, gdyż często spotykałem się z sytuacją, że element klasy był zmienny i zawierał elementy opisujące ten element. Na przykład, blogi często wprowadzały tagi do klasy elementu article, albo numer notki, co zaburzało mój algorytm.

Najpierw jednak zanim zacznę czegokolwiek szukać, znajduję element notki. W momencie jak znajdę element notki, ustalam pozycję elementu notki jako root, a z elementów pozostałych usuwam ciąg odpowiadający za łańcuch przejść do elementu notki (i zastępuję to słowem „root”). Umożliwia mi to działanie względem nowego korzenia. Przykładowo, element ojca ma następujący łańcuch przejść:

*root-html:1-body:1-div:1-div:2-article:1*

A jego dziecko, na przykład tytuł, ma taki łańcuch przejść:

*root-html:1-body:1-div:1-div:2-article:1-div:1-h1:1*

Po dekonstrukcji łańcucha tytułu otrzymujemy taki ciąg:

*root-div:1-h1:1*

Następnie, będąc w drzewie w miejscu notki patrzę, czy istnieje rodzeństwo. Jeżeli tak, uznaję, że są to potencjalne notki. Wykonuję wówczas pętlę dla każdej tej potencjalnej notki i wyszukuję wszystkie gałęzie drzewa, które odpowiadają za elementy jak autor, tytuł, data i treść. Jak widać, skrócony łańcuch przejść dla pod-elementów jest pomocny, gdyż nie trzeba znać całego łańcucha przejść, aby element właściwy odnaleźć względem notki.

Po odnalezieniu wszystkich elementów zapamiętuję je. Następnie przechodzę po wszystkich tych elementach i sprawdzam po kolei, czy którykolwiek z nich nie jest zawarty w którymkolwiek ze znalezionych elementów. W przypadku, gdy jeden element jest ojcem drugiego elementu, od ojca oddzielam dziecko. Pomaga to w ekstrakcji tekstu, gdy elementy jak tytuł, autor bądź data są zawarte w elemencie okalającym tekst. Umożliwia to łatwiejsze wyznaczanie elementów, a jednocześnie eliminuje niektóre elementy z ekstrakcji tekstu (a co za tym idzie: nie duplikują się).

Po rozdzieleniu elementów następuje algorytm ekstrakcji tekstu i dzielenie na akapity. Przyjmuje on na wejściu jeden element drzewa a zwraca tablicę z kolejnymi akapitami tekstu. Mając te akapity zapisujemy je w tablicy z dopiskiem zawierającym informację, jakiego typu jest ten element. Tę tablicę następnie tworzymy dla każdego pliku i zwracamy je do algorytmu analizy drzewa.

### Alternatywny algorytm: algorytm selektorowy

Jest to dobrze znany algorytm selektorowy. Metoda ta znajduje elementy drzewa przy pomocy selektorów CSS i zwraca te elementy do dalszej analizy.

Metoda selektorowa jest metodą rekursywną, analizuje selektor do najbliższej spacji. W przypadku, gdy jest tam element, stara się znaleźć w drzewie wszystkie wystąpienia danego elementu i przechodzi dalej. W przypadku odnalezienia znaku „>” będzie szukała tylko i wyłącznie wśród dzieci odnalezionego elementu. Jest wiele więcej reguł opisujących elementy CSS, jednakże te są najważniejsze.

Przy pomocy takich kwerend da się bez problemu opisać każdy element znajdujący się na stronie, gdyż można wyszukiwać zarówno po typach, jak i klasach, jak i klasach bez typu (.class zwróci nam wszystkie elementy o klasie „class”), ale również elementy o zadanych identyfikatorach. Jednakże identyfikatory powinny być unikalne (według W3C), dlatego zazwyczaj nie korzysta się z nich do opisu elementów na stronie, zwłaszcza, jeżeli te elementy są nieunikalne (jak autor, tytuł, itp.). Daje to nam również możliwość opisu łańcucha ścieżki z mojego algorytmu, dla przykładu, możemy opisać ten łańcuch:

*root:1-html:1-body:1-div.class:1*

Przy pomocy takiej kwerendy:

*html >body > div*

lub

*html body div*

Jednakże tracimy wówczas informację o tym, który to jest element w drzewie (co jest istotne w skrajnych przypadkach). Jednakże dzięki możliwości zdefiniowania dowolnego łańcucha jako kwerendy, daje to nam duże pole do manewru, na przykład jesteśmy w stanie powyższy przykład jednoznacznie opisać kwerendą „div.class” co zwróciłoby nam wszystkie elementy klasy „class” typu „div”.

Algorytm pozycyjny od selektorowego różni się tylko jednym: metodą znajdowania gałęzi drzewa, które zawierają nasze szukane elementy. To sprawia, że tak samo w algorytmie pozycyjnym rozdzielamy elementy od siebie wzajemnie, potem dokonujemy ekstrakcji tekstu i dzielimy na akapity, potem zapisujemy w tablicy, a całą czynność wykonujemy dla każdego z plików, co zwracamy do algorytmu analizy drzewa.

### Algorytm mieszany

Opracowałem dwa algorytmy mieszane do celów porównawczych. Najpierw wyznaczaliśmy wszystkie elementy jedną metodą, potem drugą. Chciałem sprawdzić co się stanie, gdy element notki będziemy wyznaczali metodą pozycyjną a resztę elementów metodą selektorową, oraz na odwrót: notkę metodą selektorową a resztę elementów pozycyjnie.

Nie uznawałem ich za praktyczne metody, które miałyby jakiekolwiek użycie, gdyż to byłby skrajny przypadek, gdybyśmy raz klikali element na stronie a w pozostałych przypadkach byśmy wypełniali pola tekstowe (albo na odwrót). Jednakże do celów sprawdzenia ich potencjału uwzględniłem je w badaniach.

### Algorytm ekstrakcji tekstu i podziału na akapity

Metodę tę również opracowałem sam, jednakże istnieją metody konkurencyjne o podobnej budowie.

Algorytm ten otrzymuje na wejściu fragment drzewa (gałąź), z której należy dokonać ekstrakcji tekstu z podziałem na akapity. Sama ekstrakcja tekstu jest prosta (wystarczy przeszukać liście drzewie i sprawdzić ich typ), jednak jak podzielić na akapity?

Opracowałem metodę, która dla zadanego miejsca w drzewie sprawdza, czy wszystkie dzieci i wszyscy ich potomkowie są elementami zawartymi na liście dopuszczalnych elementów. Te dopuszczalne elementy to wszystkie możliwe elementy, które nie rozdzielają tekstu, innymi słowy, są to znaczniki formatowania tekstu, nie akapitów. Dla przykładu, jest to pogrubienie („b”), pochylenie („i”), odnośnik („a”) i wszystkie inne, które nie rozdzielają tekstu (przykłady nielegalnych: „p”, „span”, „li”, „br”).

Mając tę metodę sprawdzałem, czy znajdujemy się na gałęzi, której wszystkie elementy najwyżej formatowały tekst (a nie akapit). Jeżeli tak – zwracaliśmy tekst z całej gałęzi, nie dzieląc na akapity. Jeżeli było inaczej – przechodzimy po kolei po wszystkich dzieciach. Jeżeli kolejne dzieci są zawarte jedynie w znacznikach formatu tekstu, łączymy te dzieci razem w jeden akapit. Jeżeli pojawi się dziecko, które zawiera pozostałe elementy, zatem to dziecko uznajemy za kolejny akapit.

Oczywiście pojedyncza warstwa sprawdzania nie wystarcza, żeby podzielić na akapity. Musimy dla dzieci zawierających nielegalne elementy ponownie przeprowadzić sprawdzanie, tym razem na jej dzieciach i połączyć te fragmenty, które są w porządku. Tylko wtedy mamy pewność, że podzieliliśmy na wystarczającą ilość akapitów.

Podobne metody były opracowywane w innych pracach naukowych i są równie skuteczne. Ten rozdział stanowi jedynie przypomnienie ich działania.

### Post-processing

Mamy tekst podzielony na akapity, który jest podzielony na różne pola tekstowe, które są podzielone na osobne pliki. Jednakże są to dane, które są przydatne, jednakże jeszcze nie są korpusem. Trzeba je odpowiednio przekształcić.

Pierwszą czynnością, jaką należałoby zrobić, to usunięcie elementów błędnych. W tym momencie by to następowało, ale do celów badania algorytm zwraca jak najwięcej błędów: do celów diagnostycznych. Żebyśmy byli w stanie powiedzieć, jakiego rodzaju jest to błąd i czy da się go łatwo skorygować.

Te błędy często da się łatwo poprawić, wystarczy sprawdzić, czy wszystkie zaznaczone elementy zostały pobrane. Jeżeli nie, mamy pewność, że struktura została zaburzona. Możemy sprawdzić, czy pola autor, data są zawsze tej samej długości. W przypadku odstępstwa od normy można je odrzucić. Nie napotkałem przypadków, żeby wyniki zostały błędnie zakodowane. Jednakże podejrzewam, że może się zdarzyć, że kodowanie będzie niepoprawne. Wówczas można sprawdzić, czy wszystkie polskie znaki diakrytyczne są poprawnie zakodowane. W przeciwnym wypadku – usunąć taki wpis.

Następnie, przy pomocy wtyczki w języku PHP przekształcamy tekst na dokument XML. Trzeba tylko przygotować format a dane zostaną automatycznie przekształcone do formatu premorph.

Zostaje ostatni krok, który nie jest w finalnej wersji programu: powtórna deduplikacja. W momencie, gdy dokonujemy poprawnej ekstrakcji tekstu powinniśmy uzyskać łatwe w deduplikacji pliki. Łatwe, gdyż w przypadku pobrania tej samej notki z dwóch różnych źródeł, notka ta powinna okazać się identyczna, dzięki czemu algorytmy hashujące będą w stanie bez problemu oznaczyć wpis jako identyczny. Jednakże jest to program w wersji „do badań”, która to wersja nie uwzględnia tych opcji.

Ostatnią rzeczą, jaką można zrobić, to dokonać konwersji na format CCL. Wystarczyłoby uruchomić jedną komendę przy pomocy programu WCRFT. Jednakże niepotrzebnie opóźniłoby to wyniki.

# Badanie

Badanie, jakie przeprowadziłem polega na porównaniu dwóch metod wyszukiwania elementów tekstowych w drzewie DOM. Dodatkowo do celów dydaktycznych wykonuję porównanie z metodami mieszanymi.

Moim celem tego badania było sprawdzenie, czy metoda pozycyjna może służyć jako alternatywa do aktualnie używanej metody selektorowej.

Problemem tej pracy było stworzenie takiego systemu reprezentacji danych, aby użytkownik z minimum wiedzy komputerowej był w stanie sobie poradzić z ekstrakcją tekstu ze strony internetowej. Do tego celu trzeba było utworzyć również alternatywne metody oznaczania danych.

W związku z powyższym zastosowałem reprezentację strony internetowej jako drzewa DOM, które jest łatwe w obsłudze i stanowi widoczną reprezentację strony, którą po odrobinie analizy da się nauczyć w krótkim okresie czasu.

Potrzebne jednak było narzędzie, które pozwalałoby jednoznacznie oznaczyć element jako zawierający elementy nas interesujące. Metoda konkurencyjna wymaga podania reguły, która opisze nasz element, potrzebowaliśmy jednak metody, która nie będzie wymagała definiowania reguł. Tą metodą jest opracowana przeze mnie metoda pozycyjna.

## Teoria

Metoda pozycyjna polega na definiowaniu elementu poprzez ścieżkę, jaką trzeba przejść w drzewie DOM, aby odnaleźć ten element jednoznacznie. W związku z tym, dla każdego elementu na stronie przydzielany jest łańcuch ścieżki. Dzięki takiej reprezentacji da się - dla wybranej strony internetowej bez potrzeby znania jakichkolwiek reguł - wyznaczyć dowolny element na stronie w sposób jednoznaczny.

Jednakże pozostaje pytanie, czy jest to aplikowalne dla pozostałych stron internetowych pobranych z tej samej domeny? Jest to zdecydowanie temat na osobną pracę naukową. Moja wiedza dotycząca tworzenia stron internetowych jednak pozwala mi wysunąć tę hipotezę, której nie będę starał się potwierdzić, pozwolę jednak sobie przytoczyć moje rozumowanie.

W trakcie tworzenia stron internetowych staramy się zazwyczaj osiągnąć stan, kiedy strukturę strony definiujemy jak najmniejszą ilość razy, w skrajnym przypadku tworzymy pojedynczy szablon strony i szablon notki. Często zdarza się jednak, że definiuje się jeden lub dwa szablony notki i kilka różnych szablonów strony.

Dzięki tej wiedzy możemy wyciągnąć następujące wnioski: Żeby odnaleźć notkę wystarczy, że będziemy najwyżej kilka razy wyznaczali ścieżkę, co pozwoli nam potem dokonać łatwej ekstrakcji strony. Wiemy też, że względem elementu notki najprawdopodobniej będziemy jednakową strukturę, niezależnie od strony. Wiemy też, że wszystko zależy od wewnętrznej struktury strony. Jeżeli na stronie jest wiele różnych szablonów do różnych elementów (jak na przykład różny szablon dla artykułu bądź notki blogowej), to również tyle samo razy trzeba będzie definiować wzorzec z łańcuchów przejść.

Dla kontrastu, może być również taka sytuacja teoretyczna, że istnieje wiele szablonów dla różnych elementów, dla których da się wyznaczyć taki selektor, który będzie w stanie objąć wszystkie te elementy.

Moim zadaniem będzie dla wybranej puli stron internetowych – a przede wszystkim blogów – sprawdzenie, czy moja metoda może być aplikowana, czy w jakimkolwiek przypadku będzie skuteczna, a zarazem jak wypada z tą konkurencyjną metodą – selektorową.

## Praktyka

W celu poprawnej oceny skuteczności algorytmu pozycyjnego porównuję go z algorytmem selektorowym. Do celów naukowych opracowałem również dwie metody mieszane, z których pierwsza wyszukuje notkę metodą pozycyjną a resztę elementów metodą selektorową, a druga metoda wykonuje odwrotne wyszukiwania: notkę selektorem, a resztę elementów pozycyjnie.

W ramach skorzystałem z listy opracowanej na Politechnice Wrocławskiej blogów, z którymi algorytm selektorowy miał problemy w przeszłości. Lista ta składa się z 76 blogów lub stron internetowych. Z niej pobrałem 53 blogi i dokonałem analizy porównawczej algorytmów. Lista blogów wraz z wynikami badania znajduje się w załącznikach.

W ramach analizy porównawczej, dla każdego z projektów (tak nazwałem pobrane blogi) wylosowałem 100 lub mniej plików, dla których wyznaczałem wzorce dla algorytmu selektorowego i pozycyjnego. Następnie rozpoczynał się algorytm analizy drzewa..

Algorytm ten dzieli się na cztery fazy. Pierwsza jest dla algorytmu pozycyjnego. Druga faza jest dla algorytmu mieszanego, gdzie element notki jest wyznaczany metodą pozycyjną a pozostałe elementy algorytmem selektorowym. Trzecia faza zawiera drugą metodę mieszaną: notka jest wyznaczana algorytmem selektorowym a pozostałe elementy pozycyjnie. Czwarta faza jest algorytmem selektorowym.

Po przeanalizowaniu drzewa przy pomocy mojego algorytmu, otrzymywaliśmy wyniki. W ramach tych wyników możemy otrzymać dane szczegółowe, jak:

1. Rozmiar plików i adresy do wygenerowanych danych
2. Ilość odnalezionych elementów notki (dla konkretnej fazy)
3. Ilość prawdopodobnie poprawnie pobranych elementów notki (prosty algorytm weryfikujący dane, wykonywany dla każdej fazy)
4. Dane szczegółowe z punktu 2 i 3, które uwzględniają te dane dla każdego z plików przeanalizowanych.

Następnie dokonywałem weryfikacji tych danych. Nie byłem pewien, czy dane z punktu trzeciego są rzeczywiście poprawne. Dlatego dla każdego projektu wykonywałem jeszcze analizę porównawczą wygenerowanych danych. Oprócz wielkości danych ważna jest również ich jakość. Dlatego dla metody selektorowej i pozycyjnej robiłem porównanie, która jest zdecydowanie lepsza. Przez lepsza, mam na myśli moje odczucia dotyczące jakości i ilości danych, oraz trudności wyznaczania wzorca dla danej metody.

W pliku podsumowania badań (który jest dołączony do tej pracy w formie załącznika) odnotowywałem moje wszystkie uwagi. W przypadku, gdy któraś z metod była lepsza od drugiej w znaczącym stopniu, odnotowywałem to jako plus dla metody lepszej i minus dla metody gorszej. W przypadku, gdy obie metody były równie dobre, odnotowywałem to jako plus dla obu. W skrajnych przypadkach, gdy obie metody nie były w stanie pobrać danych, obie metody dostawały minus. W przypadku, gdy strona się nie pobrała, lub pobrała się niepoprawnie, nie odnotowywałem ani minusa, ani plusa. Dla każdego projektu również wprowadzałem krótką notkę, która komentuje ten wynik. Mają one następujący format

|  |  |  |
| --- | --- | --- |
| **Pozycyjnie** | **Selektorem** | **Co to oznacza** |
| + | + | Oba algorytmy są sobie równe |
| + | - | Algorytm pozycyjny jest lepszy |
| - | + | Algorytm selektorowy jest lepszy |
| - | - | Oba algorytmy nie poradziły sobie z ekstrakcją tekstu |
|  |  | Dane nie zostały pobrane / inny błąd |

## Wyniki badań

W ramach dni 21-23 maja roku 2016 pobierałem dane z 53 blogów, do których następnie dopasowywałem wzorce w dniach 22-25 maja. W rezultacie otrzymałem 4GB zbiór danych, dla których dokonywałem analizy szczegółowej, opisanej w poprzednim rozdziale. Wyniki są następujące:

1. Spośród 53 blogów, dla 16 z nich (30,2%) nie wyznaczyłem oceny, z powodu nie pobrania strony lub błędnej struktury strony.
2. Spośród 37 pozostałych stron, dla 11 z nich (29,7%) którejkolwiek metodzie przydzieliłem minusa, co oznacza, że którakolwiek metoda sobie nie poradziła.
3. Spośród 11 projektów, w których jakakolwiek metoda sobie nie poradziła, w 3 przypadkach z 11 (27,3%) nie poradziły sobie obie metody. W ramach całego projektu 37 stron, te 3 przypadki stanowią 8,1%.
4. Spośród 34 przypadków dla poprawnego projektu (wśród których nie uwzględniamy projektów, dla których obie metody nie uzyskały poprawnych wyników), spośród 8 sytuacji (23,5%), gdy lepsza metoda była jedna lub druga, w 5 przypadkach (14,7%) nie poradziła sobie metoda pozycyjna, a w zaledwie 3 przypadkach (8,8%) nie poradziła sobie metoda selektorowa. Stosunek skuteczności wynosi 29:31, co dla metody pozycyjnej wynosi 48%.
5. Spośród 37 przypadków dla poprawnego projektu, metoda pozycyjna otrzymała gorsze lub równe wyniki metodzie selektorowej w 8 przypadkach (21,6%), natomiast metoda selektorowa otrzymała gorsze lub równe wyniki jak metoda pozycyjna w 6 przypadkach (16,2%)

## Wnioski z badań

Moje badania potwierdzają, że metoda pozycyjna może być aplikowana do celów ekstrakcji tekstu ze stron internetowych. W moich badaniach, metoda selektorowa jest skuteczniejsza o zaledwie 5.4 punktów procentowych, co umożliwia stwierdzenie, że metoda selektorowa jest skuteczniejsza, ale jednocześnie potwierdza, że w naprawdę niewielkim stopniu metoda ta jest skuteczniejsza od wygodniejszej w stosowaniu metody pozycyjnej.

Najczęstszym przypadkiem, w którym metoda selektorowa górowała nad metodą pozycyjną był wtedy, gdy algorytm ten był w stanie uzyskać więcej treści przy pomocy jednej reguły. Sprawiało to znaczne ułatwienie przy ekstrakcji tekstu. Umożliwiało zredukowanie liczby reguł o jedną iterację (lub więcej).

Najczęstszym przypadkiem, gdy metoda pozycyjna górowała nad metodą selektorową był wtedy, gdy nie dało się łatwo wyznaczyć reguł do ekstrakcji selektorem. Wynikało to często z pozycji tekstu względem notki. Błędna struktura strony zastosowana przez użytkownika sprawiała, że pole tekstowe kończyło się wcześniej, niż powinno, albo struktura strony nie zawierała klas, albo elementy, których szukaliśmy były po prostu oddzielone znacznikiem nowej linii („br”). Wyznaczenie reguł często wówczas graniczyło z niemożliwością, co dla podejścia pozycyjnego nie stanowiło problemu.

Częstym problemem związanym z moim algorytmem było też to, że element, który był uznawany za notkę często był przypadkowym elementem, który nie powinien był zostać uznany za takowy. Wówczas to doprowadzało do zaburzonych wyników, które dawały fałszywe „odnalezienia”, zarówno dla algorytmu pozycyjnego, jak i dla algorytmu selektorowego.

Powyższy wniosek daje możliwe dwa rozwiązania. Można zmienić algorytm wyznaczania potencjalnych notek, w którym badamy strukturę strony z perspektywy znalezienia takiego elementu, który zawiera elementy wewnętrzne w takich samych pozycjach, co oznaczany element. Można też algorytm zmienić, że możemy ustalać, w których strukturach należy szukać notek.

# Podsumowanie

Skuteczna ekstrakcja tekstu ze stron internetowych jest trudnym zajęciem. Automatyczna ekstrakcja czasem nie osiąga pożądanych rezultatów, dlatego korzystamy z metod półautomatycznych. Spośród metod półautomatycznych opracowałem kolejną metodę – algorytm pozycyjny - która być może pozwoli otworzyć drogę rozwoju dla metod inteligentnych albo innych metod pozycyjnych.

W ramach tej pracy naukowej opracowano narzędzie, które nie jest w pełni optymalne. Jednakże na pewno stanowi podstawę, którą będzie dało się rozwijać. W jaki sposób? Mam kilka pomysłów, które umieściłem w jednym z kolejnych rozdziałów.

## Czego się nie udało? Dlaczego się nie udało?

Niestety, każde tego rodzaju przedsięwzięcie jest obarczone tym, że nie uda się zrobić wszystkiego, ze względu na różnego rodzaju ograniczenia. W tym rozdziale postaram się je wszystkie wymienić.

Opracowałem narzędzie do badania algorytmu, które w całości jest napisane w języku skryptowym PHP. Język ten służy do prostego wyświetlania stron internetowych i nie jest zoptymalizowany do przetwarzania dużych ilości danych. W rezultacie podejrzewam, że gdyby algorytmy napisać w innym języku, jak choćby Java albo Python, to opracowane narzędzie działałoby szybciej, a również odciążyłoby procesor maszyny, która obsługuje ten skrypt. Nadal można byłoby wykorzystywać interfejs przeglądarkowy dzięki specyfice języka PHP.

Innym problemem jaki wystąpił, to brak czasu na badania. Gdybym był w stanie na badania poświęcić więcej czasu, być może miałbym bardziej miarodajne wyniki, które w miarę skutecznie porównałyby obie metody. To było głównym powodem, dla którego badania nie dotyczyły badania skuteczności algorytmu pozycyjnego, a jedynie stanowiły porównanie z metodą selektorową.

Inny problem, który również wynikał z niedostatecznej ilości czasu, to niemożliwość dokończenia projektu. W projekcie brakuje powtórnej deduplikacji po procesie analizy, która jest istotna dla właściwości utworzonego korpusu.

Brakuje w projekcie konwersji do formatu CCL. Wynika to z prostej przyczyny: narzędzie, jakim jest WCRFT zostało napisane z pomocą wielu zewnętrznych bibliotek, z których część w momencie realizacji projektu się zdezaktualizowała, co uniemożliwiło zainstalowanie narzędzia w systemie Windows.

## Kierunki rozwoju

Jest wiele różnych metod rozwoju tego algorytmu, ale są również badania, które można przeprowadzić względem tego algorytmu, które czekają na przyszłych dyplomantów do analizy. Sam zamierzam po zakończeniu tej pracy zająć się dopracowaniem kodu tego programu w celu dalszej optymalizacji, jednakże chciałem w tym rozdziale się skupić na opcjach do badań i rozwoju technologii.

Badanie, jakie można zrealizować, o którym wspomniałem wcześniej, jest zbadanie dla wielu domen w jakim stopniu strony internetowe z danej domeny zachowują stałość, gdzie następują zmienności oraz w jakim stopniu. Największą wartość byłoby odkrycie z ilu szablonów składa się pojedyncza strona internetowa, oraz jak korzystać z takich danych. W idealnym przypadku uzyskiwalibyśmy łańcuch przejść, który umożliwiałby dokonanie ekstrakcji dowolnego elementu na stronie. Mogłoby to służyć zarówno ekstrakcji tekstu ze stron internetowych, ale również innym dziedzinom, jak ekstrakcja linków z opisami stron (co robią algorytmy z firmy Google), albo dogłębnej analizie stron internetowych względem ich budowy.

Można wykonać badania, czy da się opracować metodę, która będzie w stanie opracować minimum reguł selektorowych lub pozycyjnych do ekstrakcji konkretnych (zaznaczonych) elementów na stronie. Jestem zdania, że bez metod sztucznej inteligencji nie da się opracować takiej metody.

Poprzedni punkt pozwala wysunąć kolejny materiał do badań: zastosowanie metod inteligentnych do doboru łańcucha przejść, który jednoznacznie opisuje szukane elementy. Jestem zdania, że przy odpowiedniej analizie drzewa, jego zawartości i jego zmienności algorytm inteligentny jest w stanie znaleźć miejsce w drzewie, które zawiera notki a następnie wyznaczać kandydatów do notek.

Można również zmodyfikować mój algorytm przechodzenia drzewa, który w sposób inteligentny decyduje z jakich elementów zapamiętanych korzystać przy przechodzeniu drzewa. Może się bowiem okazać, że defekty związane z moim algorytmem łatwo dałoby się poprawić przy pomocy innej definicji elementów, na przykład zamiast korzystać z typu elementu do wyznaczani pozycji, można korzystać typu elementu i najkrótszej klasy przypisanej danemu elementowi. Lub konkretnej klasy. Lub wylosowanej klasy. Opcji jest mnóstwo.

# Źródła

NIEPOSORTOWANE

<http://dl.acm.org/citation.cfm?id=775182>

Możliwe źródła:

<http://www.teachingenglish.org.uk/article/analysing-language> - elementy języka

<http://korpusy.net/> - definicja korpusu

<http://nlp.pwr.wroc.pl/redmine/projects/corpus2/wiki/CCL_format> - o formacie CCL

<http://www.cs.utah.edu/nlp/readinglist/BaroniB04.pdf> - definicje dot. korpusów

<https://lindat.mff.cuni.cz/repository/xmlui/handle/11858/00-097C-0000-000D-F696-9> - jusText, Pomikalek Jan

<http://dl.acm.org/citation.cfm?id=540590> o selektorach?

* Meyer, Eric A. *Cascading Style Sheets 2.0 Programmer's Reference*. McGraw-Hill, Inc., 2001.

Alternatywy do corpograbbera i inforex:

<http://www.textise.net/>

<https://www.diffbot.com/>

<https://github.com/kohlschutter/boilerpipe> // fulltext extraction

<https://code.google.com/p/boilerpipe/wiki/Benchmarks>

<http://www.crummy.com/software/BeautifulSoup/>

<http://rodp.me/2015/how-to-extract-data-from-the-web.html>

<http://www3.nd.edu/~tweninge/pubs/WH_TIR08.pdf>

<https://hal.archives-ouvertes.fr/inria-00326150/document>

<https://books.google.pl/books?id=vFHOx8wfSU0C&printsec=frontcover&dq=text+mining&hl=en&sa=X&ved=0ahUKEwj3usnaldjLAhVI7xQKHRmpDqwQ6AEIODAD#v=onepage&q=text%20mining&f=false>

# Załączniki

## Załącznik 1: płyta DVD

Zawiera:

1. Spakowany projekt z tym dokumentem (projekt.zip)
2. Spakowany folder tmp zawierający efekty badań, którego wyniki zinterpretowane znajdują się w załączniku drugim (badanie.zip). Po rozpakowaniu plik ten ma rozmiar ponad 4GB.

## Załącznik 2: Badania

|  |  |  |  |
| --- | --- | --- | --- |
| **blog** | **pozycyjnie** | **selektor** | **uwagi** |
| klarkamrozek.blogspot.com | + | + | Bardzo ładnie zbierają dane oba algorytmy |
| prawocywilne.blox.pl | + | - | Brak łatwych selektorów uniemożliwia łatwą ekstrakcję danych metodą selektora, pozycyjnie zbiera dużo śmietnika (łatwo wykrywalnego), ale pojedyncze wpisy pobiera wyśmienicie. |
| stanikomania.blox.pl | + | + | Pozycyjnie zbiera śmietnik, łatwo wykrywalny |
| telenovel.blog.pl | + | + | Pozycyjnie zbiera śmietnik, łatwo wykrywalny |
| ania-gotuje.blog.pl | + | + | Pozycyjnie zbiera śmietnik, łatwo wykrywalny |
| apetytnaogrod.blog.onet.pl | + | + | Pozycyjnie zbiera śmietnik, łatwo wykrywalny |
| babuni-blog49.blog.onet.pl | - | - | Selektor nie jest w stanie wyznaczyć daty i autora, obie metody zbierają dużo śmietnika, najwięcej zaś metoda pozycyjna, klasyfikacja jest niestabilna |
| blog.dzp.pl | + | + | Wszystkie algorytmy wyekstraktowały dobre dane, selektorem więcej, ale czasem błędnie (łatwo wykrywalnie), pozycyjnie pobrał tylko poprawne. |
| blog.lifenotes.pl | + | + | Blog zdjęciowy, dobranie selektora: średnie, trudny w ektrakcji selektor daty i autora, pozycyjnie oprócz daty zbiera trochę śmiecia. Jeżeli zignorować mnóstwo śmiecia, które obie metody zbierają (przez małą ilość treści w notkach), to obie metody działają bardzo sprawnie |
| blogmotive.pl |  |  | Blog się nie pobrał |
| bogatego.blog.pl |  |  | Brak treści |
| bymagda.blog.pl | + | + | Zbierają dużo śmiecia, wszystkie algorytmy równie skuteczne |
| calm-down.blog.pl |  |  | Mało treści na blogu, przez co obie metody nie wyekstraktowały właściwych danych. Losowość. |
| ciszewski.blog.pl | + | + | Pozycyjnie zbiera śmietnik, łatwo wykrywalny |
| dzieciowo.pl | + | + | Pobieranie identyczne w obu przypadkach, z drobną niekorzyścią dla pobierania pozycyjnego (w 2 z 100 plików nie znaleziono pojedynczego elementu) |
| fashionable.com.pl | + | + | Identyczne funkcjonowanie |
| ironiakowalska.blog.pl | + | + | Brak autora, tag daty trudny do wyselekcjonowania. Skuteczność identyczna, z jakiegoś powodu tylko autor został pobrany przy selektorze (prawdopodobnie błąd wypełniania formularzy) |
| jagermeisterin.blog.onet.pl | - | + | Łatwe tagi. Metoda na pozycyjnie przegrywa, gdyż selektorowa pobiera notki ze stron archiwum, gdy pozycyjna nie. Można wydać kolejny rozkaz pobierania, ale wówczas nie pobierze samych notek. Dodatkowo strony archiwum zaczynają się od divu z obrazkiem, który zaburza strukturę strony, przez co nie pobiera pierwszego wpisu. |
| jakubmuller.pl | + | + | Łatwo wykrywalne błędy (próba pobierania postów z archiwum przy niejednolitej strukturze), zarówno selektorem, jak i pozycyjnie. Artykuły pobiera wzorcowo. PS. Pozycyjnie zbiera więcej śmieci, niż selektorem, ale to wciąż śmieci z tej samej kategorii. |
| lorejn.blog.interia.pl |  |  | Automatyczne przekierowanie do strony interia.pl |
| marekowczarz.pl | + | - | Problematyczne selektory, niemożliwość wyznaczenia selektora autora i daty, pozycyjnie zbiera śmieci (bez opcji ignore nie pobierze się idealnie). Ciężko wyznaczyć treść. Nawet zakładając mój błąd w formułowaniu algorytmu (nie pobiera treści selektorem), jest to na niekorzyść selektora. |
| niedlaidiotow.blog.pl |  |  | Blog się nie pobrał |
| osmykolorteczy.blog.pl | + | + | Nie da się wyselekcjonować autora i daty (nie ma w kodzie html). Selektorem zbiera więcej śmieci, ale notki są podobnej jakości. |
| paczkiwpodrozy.pl |  |  | Blog się nie pobrał |
| plutecki.net | - | + | Treść trudna do zaznaczenia. Metoda selektorowa dużo skuteczniejsza przez niejednolitą strukturę. |
| pozytywnakuchnia.pl |  |  | Blog się nie pobrał |
| psianiol.blog.onet.pl | + | + | Brak autora. Obie metody działaja bardzo podobnie. |
| twardamatka.blog.pl | + | + | Oba algorytmy pobieraja dużo danych bez pojedynczych elementów, jak tytuł czy data. Możnaby je akceptować, ale to wciąż wadliwe dane. |
| ujagny.blog.pl | + | + | Selektor zbiera więcej śmietnika |
| uljaszowanie.blog.pl | - | + | Selektor zbiera poprawne dane zarówno z archiwum jak i z notek przy pomocy jednego zapytania. Z jakiegoś powodu raport podkreśla równość algorytmów. |
| wittamina.pl | + | + | Selektor pobiera dużo śmieci, pozycyjnie tylko poprawne dane. Obie metody równie skuteczne. |
| macademiangirl.com |  |  | Blog się nie pobrał |
| zakochana-kobieta.blog.onet.pl | + | + | Pobiera się identycznie. Pozycyjnie zbiera śmieci (łatwo wykrywalne) |
| zmiloscidogotowania.blog.pl | - | + | Selektorem jest w stanie pobrać więcej, w tym ze stron typu archiwum. Ale są to często ucięte fragmenty wpisów. Mimo to, selektor wygrywa. |
| aktywnisportowo.pl |  |  | Blog się nie pobrał |
| antyapps.pl | + | + | Równa skuteczność. |
| antymatrix.blog.polityka.pl | + | + | Zmienna struktura strony sprawia, że zarówno pozycyjnie jak i selektorem nie da się wyciągnąć treści jednocześnie z archiwum jak i z notki. Albo jedno, albo drugie. Równa skuteczność. |
| brandoctor.pl | + | + | Zmienna struktura strony sprawia, że zarówno pozycyjnie jak i selektorem nie da się wyciągnąć treści jednocześnie z archiwum jak i z notki. Albo jedno, albo drugie. Równa skuteczność. |
| czaplicka.eu | + | + | Równa skuteczność. |
| glamourina.pl |  |  | Blog się nie pobrał |
| hatalska.com | + | - | Pozycyjnie zdeklasowało selektorem o lata świetlne poprzez niestandardową strukturę strony. Data była w bloku tekstu oddzielona spacją, pojedynczy post był nieoddzielony od nagłówka, koszmar dla selektorów. |
| ittechblog.pl | - | - | Dziwna struktura strony sprawia, że pobierany jest często śmietnik. Ciężko stwierdzić, która metoda wygrywa, bo w obu przypadkach jest dużo złych danych. Dlatego obie na minus. |
| jestem.mobi | + | + | Nie da się wyselekcjonować jednocześnie z archiwum i z notek. Co ciekawe, metoda mieszana (post selektorem, elementy pozycyjnie) osiągnęła prawie dobre efekty w jednoczesnej ekstrakcji z obu źródeł. |
| mediafeed.pl | - | - | Koszmarnie trudne wybranie selektorów. Mimo wszystko, nie jest to ważne, bo blog nie posiada prawie treści. |
| niebezpiecznik.pl |  |  | Blog się nie pobrał |
| politologianarapie.pl | + | + | Blog posiada skąpą treść (większość treści jest w filmikach). Mimo to, metoda pozycyjna zebrała dużo śmiecia, kiedy selektorem nie. |
| stayfly.pl | - | + | Notki na różnych stronach potrafią być w troszkę innym miejscu, co zaburza działanie mojego algorytmu. Idzie to na plus dla selektorowego algorytmu. |
| wdolnymslasku.com | + | + | Brak pobranych wpisów, wszystko idzie z "archiwum". Działanie niemal identyczne. |
| brzoskowski.pl |  |  | Blog się nie pobrał |
| marketinghotelu.pl |  |  | Blog ma zaburzoną strukturę html, która uniemożliwia jej analizę. |
| olgasmile.com | + | + | Metoda selektorowa zbiera więcej śmieci, ale jest równie skuteczna, co metoda pozycyjna |
| webkomunikacja.pl |  |  | Blog się nie pobrał |
| zajadam.pl |  |  | Blog się nie pobrał |