|  |  |  |  |
| --- | --- | --- | --- |
| C:\Users\Jacek\Desktop\Bez nazwy.png | | | |
|  | |  | **Wydział Informatyki i Zarządzania**  kierunek studiów: Informatyka  specjalność: Internet i Technologie Mobilne  Praca dyplomowa - magisterska  **Webowy system do automatycznego gromadzenia korpusów tekstu z Internetu i ich ręcznej anotacji meta-danymi**  Jacek Mariański  **słowa kluczowe:**  korpus językowy  ekstrakcja tekstu z Internetu  tworzenie korpusów  **krótkie 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.   |  |  |  |  | | --- | --- | --- | --- | | opiekun pracy  dyplomowej | .................................................. | ....................... | ....................... | | *Tytuł/stopień naukowy/imię i nazwisko* | *ocena* | *podpis* |   *Do celów archiwalnych pracę dyplomową zakwalifikowano do:\**  *kategorii A (akta wieczyste)*  *kategorii BE 50 (po 50 latach podlegające ekspertyzie)*  *\* niepotrzebne skreślić*   |  | | --- | | pieczątka wydziałowa | |
|  |  |  | Wrocław, 2016 |

**Spis treści**

[1. Wstęp: Cel, motywacje, ograniczenia 6](#_Toc453365479)

[2. Kluczowe pojęcia i terminy używane w tej pracy 9](#_Toc453365480)

[3. O korpusach i ekstrakcji tekstu 14](#_Toc453365481)

[4. Corporanet 16](#_Toc453365482)

[4.1. Projekt 16](#_Toc453365483)

[4.2. Elementy 16](#_Toc453365484)

[4.3. Interfejs 17](#_Toc453365485)

[4.4. Działanie 18](#_Toc453365486)

[5. Badanie 28](#_Toc453365487)

[5.1. Teoria 28](#_Toc453365488)

[5.2. Praktyka 29](#_Toc453365489)

[5.3. Wyniki badań 30](#_Toc453365490)

[5.4. Wnioski z badań 31](#_Toc453365491)

[6. Podsumowanie 32](#_Toc453365492)

[7. Źródła 34](#_Toc453365493)

[8. Załączniki 37](#_Toc453365494)

**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.

# 1. Wstęp: Cel, motywacje, ograniczenia

**1.1. Motywacje**

Analiza syntaktyczna języka jest jedną z ważnych dziedzin informatyki. Przede wszystkim może nam służyć do analizy części mowy, zapisu (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 (korpusy.net)[1]. 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 (nlp.pwr.wroc.pl: „CCL”)[2] 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 współczesnych 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[3][4]. Jest nią Internet.

Nie jest problemem pozyskiwanie zasobów tekstowych 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 oparciu 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 zasobów internetowych, bądź mają tematykę korpusową. Mam nadzieję, że dzięki tej pracy przyszłe pokolenia naukowców będą mogły korzystać z mojego narzędzia bez problemów, oraz że będą oni rozumieli jego działanie.

**1.2. 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.

**1.3. Zadania projektowe i badawcze**

Moim głównym zadaniem jest opracować narzędzie, które będzie pobierało tekstowe zasoby internetowe przy pomocy użytkownika. Rolą użytkownika będzie oznaczanie elementów na stronach internetowych, które go interesują. Rolą programu będzie dokonanie ekstrakcji tekstu zgodnie z żądaniem użytkownika. Kolejnym zadaniem programu będzie zapisanie danych w formacie korpusowym.

Wiele z tych rzeczy generuje podzadania. Postaram się je tutaj wymienić.

1. Narzędzie ma pobierać tekstowe zasoby internetowe: należy skonstruować narzędzie tekstowe, które w sposób automatyczny pobierze zasoby internetowe z zadanej domeny
2. Rolą użytkownika będzie oznaczenie elementów na stronach internetowych, które go interesują:
   1. Interfejs musi umożliwiać zaznaczanie, bądź oznaczanie, tudzież definiowanie reguł umożliwiających jednoznaczne oznaczanie elementów na stronie
   2. Interfejs musi być jasny i przejrzysty
   3. Zdefiniowane reguły muszą być jednoznaczne dla zadanego zestawu danych.
3. Rolą programu będzie dokonanie ekstrakcji tekstu zgodnie z żądaniem użytkownika: Należy skonstruować moduł, który dla zadanych reguł zaaplikuje je do całego zbioru danych a następnie dokona ekstrakcji tekstu z tych elementów.
4. Zadaniem programu będzie zapisanie danych w formacie korpusowym: Dane, które w poprzednim punkcie zostały uzyskane następnie muszą być (zgodnie ze standardami dotyczącymi korpusów językowych): wyczyszczone z błędów, duplikatów, a następnie zapisane w jednym z dopuszczalnych formatów korpusowych.

W ramach zadania badawczego przyjrzę się punktowi 2a, który mówi o jednoznacznym zaznaczaniu, bądź oznaczaniu elementów. Aktualne metody oznaczania elementów na stronie są trudne w zrozumieniu przez użytkowników, dlatego opracowałem metodę oznaczania danych, która umożliwi zaspokojenie celów z punktu 2b i 2c: interfejs będący jasny i klarowny, oraz oznaczenie będące jednoznaczne dla zadanego zestawu danych. Celem badawczym będzie porównanie tej metody z konkurencyjną w ramach powyższych wymagań.

**1.4. 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 53 blogi i strony internetowe, które najpierw pobierałem, a potem ręcznie oznaczałem. Blogi wybierałem z listy stron opracowanej na Politechnice Wrocławskiej, na której znajdowały się blogi różnego rodzaju, jednakże jej przeznaczenia nie znam.

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.

# 2. Kluczowe pojęcia i terminy używane w tej pracy

W ramach tej pracy będę omawiał pewne koncepcje, które mogą wydawać się niejasne. Oczywistym jest, że muszę skonstruować narzędzie, jednakże w tym rozdziale postaram się opowiedzieć o tym, czym są te konstrukcje, o które opieram moją pracę. Mam nadzieję, że dzięki temu rozdziałowi umożliwię zrozumienie problemu.

Ten rozdział składa się z wielu pod-rozdziałów, w których omawiam pojedyncze pojęcia, oraz jaka jest ich moja interpretacja. Postaram się również podkreślić ważność tych pojęć. Postaram się też zacząć od najważniejszych.

**2.1. Korpus językowy**

W pracy tej będę wielokrotnie poruszał pojęcie korpusu. Korpus językowy – definicyjnie – jest to zbiór tekstów, w którym możemy szukać typowych słów, bądź konstrukcji oraz innych informacji, jak na przykład znaczenia słów, zbitek słownych czy zdań (korpusy.net)[1]. Jednakże w ramach tej pracy będziemy mówili nie o istniejącym korpusie. A istnieje ich wiele. Istnieje Narodowy Korpus Języka Polskiego (nkjp.pl)[5], korpus Instytutu Podstaw Informatyki PAN (korpus.pl)[6], itd. Jednakże zasoby internetowe w nich są niewielkie. Dlatego zadaniem tego narzędzia będzie stworzyć taki korpus.

Taki korpus powinien posiadać pewne cechy, jakimi jest na pewno – definicyjnie – zawartość tekstowa. Korpus utworzony powinien być reprezentacją języka, którym posługują się internauci, aby był przydatny dla wielu naukowców. Zatem musi być dokładnym odwzorowaniem treści znajdujących się na stronach. Co za tym idzie – w tym korpusie mogą się znaleźć tylko te treści, które reprezentują ten język, czyli nie możemy ściągnąć całej strony i uznać jej za korpus. Musi ten korpus być złożony tylko z tych elementów, które zawierają wypowiedzi internautów. Strony internetowe zawierają jednak dużo informacji redundantnych, pojawiających się na wielu podstronach. Wszelkiego rodzaju duplikaty też nie mogą się pojawiać, gdyż zaburzają nasze postrzeganie języka, a przede wszystkim – zaburzają częstość konstrukcji zdaniowych.

Korpus, oprócz cech dotyczących teorii, musi mieć też zastosowanie praktyczne. Musi zawierać wszelkiego rodzaju informacje o języku, jakie naukowcy mogliby chcieć wiedzieć o tekście. Musi być podział na akapity, zdania, pojedyncze słowa. Do tego dochodzą jeszcze różnego rodzaju anotacje. W praktyce oznacza to dużo pracy nad przerabianiem zasobów tekstowych, aby były możliwe do analizy.

**2.2. Anotacja**

Anotacja jest pojęciem prostym. Jest to jakakolwiek informacja, która opisuje zadaną konstrukcję językową, czy to słowo, czy to parę słów, czy całe zdanie. Często w ramach korpusów mówimy o anotacji morfo-syntaktycznej, która uwzględnia klasę gramatyczną (np. rzeczownik, przymiotnik), ale również kategorię gramatyczną (liczba, przypadek, itp.) (korpusy.net: „Korpus IPI PAN”)[7]. Na Politechnice Wrocławskiej dodatkowo stosuje się również – przykładowo - anotacje znaczeniowe (gdzie do słów próbuje się przypisać potencjalne znaczenia).

Anotacje takie często nie są wykonywane automatycznie. Część jesteśmy w stanie uzyskać, jak choćby przez jednoznaczne konstrukcje wyrazowe, np. World Trade Center (nazwa budynku), czy Stany Zjednoczone Ameryki (nazwa kraju). Ale zdarza się, że w ramach pewnych społeczności pewne pojęcia są dopiero tworzone i nie jesteśmy w stanie ich wykryć automatycznie. Czasem też program nie jest w stanie wykryć pewnych rzeczy, jak na przykład odróżnienie tytułu od nagłówka strony, notki od komentarza, daty wpisu od daty edycji i wielu innych elementów. Dlatego też część anotacji jest przeprowadzana przez użytkownika. Jednym z założeń mojej pracy jest, że część informacji użytkownik będzie anotował. Zajmę się najprostszą formą anotacji: anotacja notek, poprzez rozróżnienie, co jest w niej autorem, co tytułem, datą a co jest treścią właściwą.

**2.3. Robot sieciowy (crawler)**

W ramach tworzenia oprogramowania dokonującego ekstrakcji stron internetowych z zasobów tekstowych, warto wiedzieć, z jakich elementów te programy się składają. Jednym z takich elementów jest crawler, czyli robot sieciowy.

Roboty sieciowe są nieodzownym elementem naszej Internetowej rzeczywistości ("Web crawler: a review." *International Journal of Computer Applications* 63.2, Kausar, Dhaka, Singh)[8]. Jest to program, który zgodnie ze zdefiniowanymi parametrami przeszukuje stronę internetową w celu pozyskania konkretnej treści (tamże)[8]. Często służą do pobierania stron internetowych do późniejszego przeglądania ich offline. Czasem służą do niecnych celów, jak do kradzieży własności intelektualnej, która znajduje się na stronie, stąd kontrowersje.

W ramach projektu korzystam z zewnętrznego programu do pobierania stron internetowych. Korzystam z programu WGET [9], który jest przystosowany do zasad netykiety – respektuje plik robots.txt, umożliwia pobieranie z odstępami czasowymi, oraz ma szereg opcji przydatnych do moich celów: zapisuje pliki w jednym katalogu, umożliwia nie pobieranie obrazków i filmów i inne. Dodatkowo jest wieloplatformowy – został napisany w języku C++, a następnie skompilowany dla systemu Windows i Linux.

W jednej z wcześniejszych wersji programu korzystałem z crawlera Httrack [10], który również był wykorzystywany w narzędziach na Politechnice Wrocławskiej. Miał jednak wady, między innymi odstępy czasowe były zbyt małe (co blokowało pobieranie z wielu stron internetowych), sporej części stron w ogóle nie pobierało, a jedynym mechanizmem zatrzymującym pobieranie był czas pobierania. To ostatnie mogło być uznane za zaletę, jednakże do celów korpusowych znacznie lepsze jest pobieranie strony w całości, bez jakiegokolwiek ograniczenia czasowego.

**2.4. Drzewo DOM**

Drzewo DOM – Document Object Model – jest reprezentacją strony internetowej jako drzewo obiektów (w3schools.com: „JavaScript HTML DOM”)[11]. Jako struktura drzewiasta elementy zagnieżdżone na stronie są dziećmi elementu zagnieżdżającego. Obiekty są liśćmi i gałęziami w drzewie, gdzie każdy obiekt posiada typ. Może natomiast posiadać jeszcze klasę, różne zmienne i funkcje.

Reprezentacja w postaci drzewa ma wiele interesujących właściwości. Dzięki takiej reprezentacji możemy z łatwością przeszukiwać drzewo w ramach zadanego tekstu. Elementy w drzewie da się identyfikować poprzez ścieżkę przejść przez drzewo (istnieje tylko jedna taka ścieżka dla każdego elementu, co wynika z definicji drzewa: graf spójny, acykliczny). Reprezentacja drzewiasta też jest stosunkowo łatwa do zrozumienia. Dlatego też zdecydowałem się – do celów identyfikacji elementów ze strony – reprezentować stronę w formie drzewa.

**2.5. Algorytm selektorowy**

Jest to algorytm opracowany przez naukowców do odnajdywania elementów na stronie. Jednakże zanim opiszę jego działanie, zdefiniuję: czym jest selektor.

Selektor jest to wzorzec, który umożliwia jednoznaczne odnajdywanie szukanych elementów (w3schools.com: „CSS Selector Reference”)[13]. Składa się najczęściej z typu elementu, często również z klasy, identyfikatora elementu a rzadziej również ze struktury strony (np. zapis „html div” oznacza element typu „div” będący potomkiem elementu „html”), oraz zależności pomiędzy elementami (np. zapis selector1:not(selector2) oznacza, że elementy selektora „selector1” powinny nie zawierać elementów zawartych w zbiorze elementów selektora „selektor2”).

W projektowaniu stron internetowych często korzysta się z selektorów, gdyż jest to wygodne narzędzie do oznaczania elementów, dla których chcemy przyporządkować dane własności, jak styl, czy funkcje. Dzięki temu, opracowano bardzo prosty algorytm wyszukiwania elementów na zasadzie takich wzorców. Algorytm ten jest stałą częścią naszych przeglądarek, gdyż dzięki temu algorytmowi strony internetowe wyglądają tak, jak wyglądają.

Algorytm ekstrakcji tekstu korzysta z tego podejścia, że użytkownicy oznaczają elementy na stronach przy pomocy prostych reguł. Da się dzięki temu z niewielką trudnością dokonać ekstrakcji takiego wzorca do celów późniejszej ekstrakcji tekstu. Jednakże taka ekstrakcja wzorców wymaga posiadania minimalnej wiedzy na temat budowy stron internetowych, przez co wielu naukowców nie radzi sobie dobrze z jej stosowaniem. Dlatego między innymi powstała ta praca.

**2.6. 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.

**2.7. CCL**

CCL jest to format korpusowy opracowany na Politechnice Wrocławskiej, jego rozwinięciem jest Corpus Constraint Language (nlp.pwr.wroc.pl: „CCL”)[2]. 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
2. Podziały na tokeny (gdzie token może zagnieżdżać w sobie tokeny, innymi słowy pełni funkcje również dzielenia na zdania i informacje wielowyrazowe) 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

**2.8. 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)

**2.9. 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 (Marcińczuk, Kocoń, Ptak, Kaczmarek, „Inforex”)[14]. 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.

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.

**2.10. Corpo-grabber**

Jest to narzędzie, podobnie jak moje, służące do automatycznego gromadzenia korpusów internetowych z Internetu. Powstało ono na bazie programu Corpus-suck (Kocoń, Jan, 2010, Corpus-SUCK)[15]. 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.

**2.11. WCRFT**

Wrocław CRF Tagger – jest to narzędzie stworzone na Politechnice Wrocławskiej, tak jak powyższe narzędzia, do analizy morfo syntaktycznej tekstu (Radziszewski, Adam. "A tiered CRF tagger for Polish.")[16]. 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.

**2.12. 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 (Maryl, Piasecki, Kocoń, Oleksy. „Linguistic analysis of weblog genres”, 2014)[17]. 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.

# 3. O korpusach i ekstrakcji tekstu

W ramach tego rozdziału chciałem zrobić mały przegląd literatury na temat korpusów i ekstrakcji tekstu ze źródeł internetowych. 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.

**3.1. Korpus językowy**

O korpusach można się dowiedzieć z wielu różnych źródeł. Swoją wiedzę czerpałem przede wszystkim ze źródeł internetowych (korpusy.net)[1], albo w książce Adama Przepiórkowskiego ("Korpus IPI PAN. Wersja wstępna." (2004))[18], bądź też w pracy zbiorowej o Narodowym Korpusie Języka Polskiego (Przepiórkowski, Bańko, Górki, Lewandowska-Tomaszczyk: „Narodowy Korpus Języka Polskiego”)[19]. W pierwszym źródle dowiemy się z pewnością wiele ogólnych spraw dotyczących korpusów. W tekście pana Przepiórkowskiego jest wiele szczegółów na temat budowy korpusu IPI PAN. W pracy zbiorowej o NKJP natomiast jest wiele interesujących danych na temat samych danych zawartych w korpusie.

W ramach korpusów warto odwiedzić witrynę Politechniki Wrocławskiej na temat notacji używanej lokalnie, jaką jest format CCL(nlp.pwr.wroc.pl: „CCL”)[2]. A można tam przeczytać o budowie typowego pliku korpusowego, oraz jakie mają znaczenie dane elementy.

**3.2. Pobieranie zasobów z sieci**

O pobieraniu z sieci zasobów internetowych jest mnóstwo źródeł, gdyż jest to temat wielu dyskusji. Najłatwiej sięgnąć po wiedzę z definicji encyklopedycznej, jaką nam niesie artykuł na wikipedii o robotach sieciowych (en.wikipedia.org: „Web Crawler”)[20]. Jednakże można o nich poczytać na różnych stronach internetowych (robotstxt.org: „Frequently Asked Questions”)[21], która ma pokaźną wiedzę na temat tego, czym są, do czego służą i jak powinniśmy z nimi współpracować.

Jednakże ja współpracowałem z gotowymi narzędziami do pobierania treści ze stron internetowych jak curl[22], wget[9] i httrack[10]. Curl został wbudowany jako biblioteka do silnika PHP i służy głównie do pobierania pojedynczych stron internetowych. Najlepiej o nim poczytać w dokumentacji do narzędzia, których zamieszczam cytowania. To samo tyczy się programów wget i httrack.

Warto wspomnieć choćby o jednej z pokrewnych pracach związanych z pobieraniem stron internetowych do celów korpusowych. W pracy „Efﬁcient Web Crawling for Large Text Corpora”[23] autorstwa m.in. Jana Pomikalka, opisane jest działanie programu spiderling, ich własnoręcznie napisanego robota sieciowego, który według napisanego przez nich kryterium wybiera te dokumenty, które zawierają potencjalnie najlepsze wyniki (jeżeli chodzi o zasoby tekstowe).

**3.3. Ekstrakcja tekstu**

Ekstrakcja tekstu ze strony html została poruszona również w wielu źródłach. Jednakże najczęściej opisywana jest automatyczna metoda pozyskiwania tekstu ze stron internetowych. Jedną z nich jest procedura ekstrakcji w programie jusText opisanym w wyżej wymienionym „Efficient Web Crawling...”[23], ale również w sieci istnieje opis algorytmu w bardzo prostym języku, co rekomendowałbym jako przykład algorytmu automatycznego. Istotny jest fragment o samym podziale na akapity, z którego korzystam.

Narzędziem wartym uwagi, na podstawie którego się częściowo opierałem jest Corpus-suck (Kocoń, Jan, 2010, „Corpus-SUCK”)[15], który był wstępem dla programu Corpo-grabber. Program pobiera tekst ze strony i sprawdza, czy w jednolitym bloku tekstu występuje min. 300 znaków, oraz czy minimum 20% słów należy do słownika morfeusz. Jak widać, jest to kolejna metoda pobierania automatycznego, która do celów weryfikacji sprawdza, czy słowa są w języku polskim.

Inną ważną pracą, na którą natknąłem się już po napisaniu mojego narzędzia, jest praca bardzo zbliżona do mojej, bo opierająca się na podobnych założeniach. „A DOM Tree Alignment Model for Mining Parallel Data from the Web” (Shi Lei i inni, na konferencji „21st International Conference on Computational Linguistics…”, 2006)[24] jest pracą naukową, która opisuje ekstrakcję korpusów dwujęzycznych ze stron internetowych. Sam proces ekstrakcji był dla mnie nieistotny. W pracy tej jest opisany tok myślenia, jaki ja również miałem - strony internetowe w obrębie jednej domeny mają jedną lub najwyżej kilka różnych struktur stron. Strony te można rozpatrywać jako drzewo DOM i patrzeć na strony z perspektywy grafu. Takie założenie pomaga zwłaszcza przy ekstrakcji tekstów ze stron dwujęzycznych, które to założenie również przekłada się na podobieństwo treści, co umożliwia oznaczanie akapitów za prawdopodobne tłumaczenie treści.

Innym ciekawym dokumentem, na jaki natrafiłem, jest „DOM-based Content Extraction of HTML Documents” autorstwa m.in. Suhit Gupta[25]. Autorom nie zależy jednak na ekstrakcji tekstu. Dokonują selekcji elementów ze strony internetowej na bazie drzewa DOM do celów przeformatowania strony na efektywniejszą, bądź też pozbawioną reklam. Jest to o tyle interesująca lektura, że na bazie podobnego pomysłu do mojego autorzy tworzą zupełnie inne narzędzie.

Ja opieram się na koncepcie pobierania półautomatycznego, gdzie użytkownik wybiera element ze strony, z którego chce, aby pobrany został fragment strony a następnie jest on pobierany. Nie jest on szczególnie dobrze opisany w literaturze, albowiem nie znalazlem takich materiałów. Wiem natomiast - z przeglądania kodów źródłowych różnych programów - że wykorzystywana jest metoda selektorowa przy zaznaczaniu elementów. Metoda selekcjonowania elementów na podstawie selektorów jest dobrze opisana na stronach w3schools („CSS Selector Reference”)[13]. Była prezentacja w roku 2014 opisująca narzędzie Blog Reader (Maryl, Piasecki, Kocoń, Oleksy. „Linguistic analysis of weblog genres”, 2014)[17], jednakże nie jest ona dostępna w Internecie. Polecam ją jednak, gdyż zawiera istotne informacje o procesie wyznaczania selektorów.

**3.4. Inne narzędzia korpusowe**

Poza samą ekstrakcją jest jeszcze wiele aspektów związanych z korpusami. Praca „Efficient Web Crawling…”[23] porusza między innymi temat deduplikacji, który jest szalenie istotny w ekstrakcji tekstu do korpusów. Jest to aspekt związany z dużą redundantnością danych na stronach internetowych, która sprawia, że pobrane teksty zawierają wiele duplikatów. Zaproponowany algorytm Onion ma temu przeciwdziałać.

Jednakże oprócz uzyskania samego tekstu musimy jeszcze go przekształcić na korpus. Do tego służy między innymi narzędzie jak WCRFT, czyli Wrocław CRF Tagger, gdzie CRF to metoda Conditional Random Fields (Radziszewski, Adam. "A tiered CRF tagger for Polish.")[16]. Narzędzie opracowane korzysta ze statystycznych metod modelowania danych, która została wykorzystana do tagowania klas gramatycznych, co autor określił mianem „części mowy”. Jest to niezwykle przydatne narzędzie przy anotacji korpusów, gdyż oszczędza mnóstwo czasu dla naukowców.

O konwersji tekstu do korpusu można poczytać we wcześniej wspomnianym dokumencie Adama Przepiórkowskiego ("Korpus IPI PAN. Wersja wstępna." (2004))[18], gdzie jest wiele informacji na temat konwersji, jaką zachodzą dokumenty w czasie konwersji do formatu korpusowego. Jest też wiele informacji o samym korpusie IPI PAN, dlatego tym bardziej polecam tę pracę.

# 4. 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 dłuższej analizie prezentuje wyniki pobierania.

## 4.1. Projekt

Zdecydowałem się na skorzystanie z języka skryptowego PHP. Język PHP służy przede wszystkim jako pośrednik pomiędzy serwerem a użytkownikiem, który treść ogląda przy pomocy przeglądarki internetowej. To znacznie pomaga przy prezentacji treści, gdyż użytkownik w przeglądarce widzi treść i w przeglądarce edytuje tą treść. Sprawia to, że nie trzeba przygotowywać dodatkowego interfejsu, który by prezentował treść strony tak, jak w przeglądarce użytkownika.

Do języka PHP istnieje wiele napisanych wtyczek, które umożliwiają dodatkowe ułatwienie pracy, między innymi istnieje wtyczka do analizy drzewa strony internetowej (o nazwie PHPHtmlParser), z której korzystałem. Do celów optymalizacyjnych docelowo powinno jednak skorzystać się z zewnętrznego programu, który do celów naszej strony internetowej będzie przygotowywał drzewo strony i obsługiwał zapytania. To pomoże odciążyć serwer przy obsłudze dużych żądań.

W trakcie produkcji narzędzia miałem za zadanie pobierać zasoby internetowe. W ramach tego zadania korzystałem z aplikacji jak httrack, curl, czy wget, jednakże wspomnę o nich w rozdziale 4.3.2 i 4.3.3. W najogólniejszym skrócie, wget okazał się być najbardziej przydatny przy półautomatycznej ekstrakcji tekstu.

Ze względu na konieczność w budowaniu interfejsu łatwego w obsłudze, musiałem myśleć jednocześnie o logice programu, jak i o wyglądzie strony. Dlatego zaimplementowałem najprostszą bibliotekę MVC (model-view-controller), jaką znalazłem, a kryła się pod nazwą SimpleMVC (proste MVC).

## 4.2. 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”.

## 4.3. 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)

***4.3.1. Strona główna***

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

***4.3.2. 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 przy pomocy programu curl 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.

***4.3.3. 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. Oczekiwania, jakich nie spełniał httrack był brak możliwości zarządzania procesem pobierania, który stwarzał blokady na stronach, z którymi alternatywne programy radziły sobie wyśmienicie.

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 „/”

***4.3.4. 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.

## 4.4. 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.

***4.4.1. 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.

***4.4.2. 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.

***4.4.3. 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.

***4.4.4. 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] i dla każdego swojego dziecka przekazuje mu ścieżkę do siebie + identyfikator dziecka.

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”:

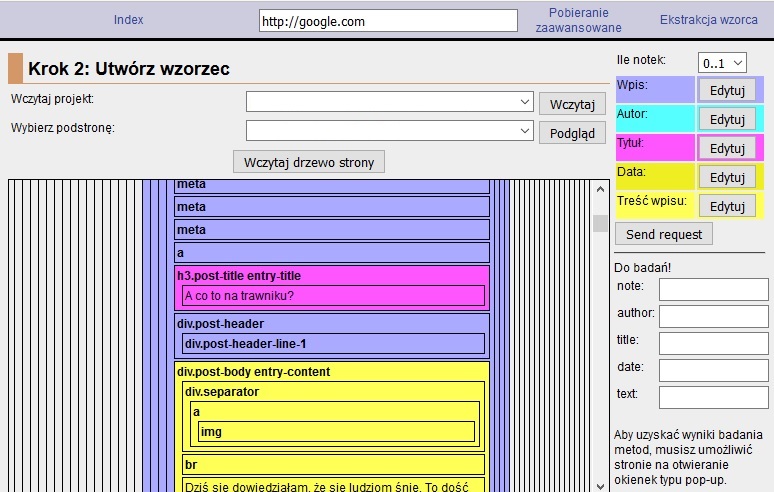
*<div id="root-html.v2:1-body.loading variant\_birds:1-div.navbar section:1-div.widget Navbar:1-div:1"><b>div</b><br></div>*

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.

***4.4.5. 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 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.

***4.4.6. 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.

***4.4.7. 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.

Przykładowe działanie algorytmu: Dla zadanej strony przedstawię kolejne kroki odnajdywania elementu:

|  |
| --- |
| <html>  <head>  <style> (…) </style>  </head>  <div class=”class0”>  <div>  <h1> text </h1>  <div>text</div>  <div>text</div>  </div>  </div>  <div class=”class1”>  <div>  <h1> text </h1>  <div>text</div>  </div>  </div>  </html> |

*Kod html przykładowej strony*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| html -> | head -> | style |  |  |
| | | div.class0 -> | div -> | h1 -> | text |
| | | | | div -> | text |  |
| | | | | div -> | text |  |
| | | div.class1 -> | div -> | h1 -> | text |
| | | | | div -> | text |  |

*Reprezentacja kodu html w postaci drzewa*

Załóżmy, że chcemy pobrać tekst z nagłówka h1 znajdującego się w divie o klasie class1. Do tego celu musimy skonstruować nasz łańcuch przejść. Najpierw mamy korzeń, więc pierwszym elementem będzie „root”, zawsze. Następnie znajdujemy kolejno ścieżkę: html:1, potem jest div.class1:2 (który jest 2 elementem typu div wśród dzieci elementu html), potem div:1, a na końcu nasz h1:1, którego szukaliśmy. Odnotowujemy to w postaci takiego łańcucha:

*root-html:1-div.class1:2-div:1-h1:1*

Co w kontekście naszego przykładu przekłada się na kolejne przejścia (rozpoczynając od dzieci korzenia):

1. Czy natrafiłem na element „html”? Tak -> Analizuję dzieci tego elementu html.
2. Czy natrafiłem na element „div”? Nie
3. Czy natrafiłem na element „div”? Tak. Czy jest to 2 element typu div? Nie, pierwszy.
4. Czy natrafiłem na element „div”? Tak. Czy to 2 element typu „div”? Tak, analizuję dzieci tego elementu div
5. Czy natrafiłem na element „div”? Tak. Analizuję dzieci tego elementu div
6. Czy natrafiłem na element „h1”? Tak.
7. Czy dotarłem do końca łańcucha przejść? Tak. Czy ostatni element łańcucha przejść został odnaleziony? Tak. Zatem szukany element został odnaleziony.

***4.4.8. 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”.

Innym przykładem jest ten omawiany w rozdziale 4.4.7. Mając łańcuch *root-html:1-div.class1:2-div:1-h1:1* jesteśmy w stanie wyznaczyć bardzo prostą regułę ekstrakcji tego elementu h1, mianowicie stosując zapytanie *div.class1 h1*, co jest wystarczające, by dla tamtej strony jednoznacznie wyznaczyć ten element a jednocześnie jest dużo krótszą regułą.

Algorytm pozycyjny od selektorowego różni się tylko jednym: metodą znajdowania gałęzi drzewa, które zawierają nasze szukane elementy. To sprawia, że tak samo jak 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.

***4.4.9. 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.

***4.4.10. 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.

***4.4.11. 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.

# 5. 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.

## 5.1. 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.

## 5.2. 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 analizy podzieliłem 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.

***5.2.1. Reguły oceny badania***

W pliku podsumowania badań (który jest dołączony do tej pracy w formie załącznika) odnotowywałem moje wszystkie uwagi. Porównywałem bezpośrednio dwie metody: selektorową i pozycyjną, jeżeli zauważyłem, że metoda mieszana była lepsza od pozostałych, to odnotowywałem ten fakt w notatkach. 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. Poniżej tabelka objaśniająca zapis.

|  |  |  |
| --- | --- | --- |
| **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 |

W jaki sposób oceniałem, która metoda jest lepsza? Wykonywałem kolejne kroki porównawcze:

1. Czy oba algorytmy pobrały podobną ilość danych (+/- 10%)? Jeżeli nie, sprawdzałem dlaczego (punkt 3). Jeżeli tak, sprawdź punkt 2.
2. Czy oba algorytmy pobrały identyczną ilość poprawnych notek? Jeżeli nie, sprawdzałem dlaczego (punkt 3). Jeżeli tak, dla pewności wykonywałem punkt 3.
3. Porównywałem zbiory notek z obu algorytmów i losowałem notki należące do różnicy zbiorów, innymi słowy: te tworzące różnice w ilościach zbiorów. Wybierałem je oraz kilka losowych.
4. Następnie, dla tych wylosowanych notek sprawdzałem, co było powodem różnicy. Jeżeli dane te były niepoprawne (np. gdy zaznaczony był autor, a w danych był fragment treści, uznawałem to za błędne dane), uznawałem te notki za niepoprawne.
5. Jeżeli błędne dane nie miały uzasadnienia teoretycznego (o czym napiszę niżej), uznawałem to za wystarczający powód, by stwierdzić, że algorytm nie działa poprawnie. Jeżeli (dla algorytmu selektorowego) reguły wyznaczania elementów były trudne, uznawałem to za dodatkowy punkt, który mógł przeważyć moją opinię o tym algorytmie. Natomiast jeżeli jeden algorytm pobierał więcej poprawnych danych niż drugi, uznawałem go za skuteczniejszy (co skutkowało negatywną oceną dla algorytmu konkurującego).

Do celów uzasadniania błędów musiałem brać pod uwagę, że moje algorytmy ekstrakcji tekstu nie były dopracowane w momencie badania. Często mimo niezdefiniowania reguły dla konkretnego elementu ten pobierał „losowe” elementy notki. Wynikało to z błędnie zdefiniowanego algorytmu. Często też, różnica w ilości danych wynikała z tego, że nie odrzucałem notek, w których nie pobrało niektórych zdefiniowanych elementów. Musiałem te czynniki brać pod uwagę analizując dane i mam nadzieję, że nie popełniłem przez to niedoszacowania.

## 5.3. 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 26 z nich (70,3%) obie metody osiągnęły podobne, lub identyczne rezultaty w ramach moich kryteriów
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 37 przypadków dla poprawnego projektu (wśród których nie uwzględniamy projektów, dla których obie metody nie uzyskały poprawnych wyników), spośród 8 sytuacji (21,6%), gdy lepsza metoda była jedna lub druga, w 5 przypadkach (13,5%) nie poradziła sobie metoda pozycyjna, a w zaledwie 3 przypadkach (8,1%) nie poradziła sobie metoda selektorowa. Stosunek skuteczności wynosi 29:31, co dla metody pozycyjnej wynosi 48%.

## 5.4. 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.

# 6. Podsumowanie

Celem tej pracy było przybliżenie tematyki związanej z ekstrakcja tekstu z Internetu do celów korpusowych. W ramach tej pracy skonstruowane zostało narzędzie, które spełnia wytyczne, które wyspecyfikowałem w ramach zadań projektowych. Narzędzie umożliwia pobieranie tekstowych zasobów internetowych. Interfejs umożliwia zaznaczanie elementów na stronie, jest jasny i przejrzysty a reguły w ten sposób zdefiniowane są jednoznaczne dla zestawu danych. Zdefiniowane reguły umożliwiają ekstrakcje elementów z całego zbioru danych, które to elementy następnie są formatowane w sposób odpowiedni dla korpusów. Następnie program zapisuje te dane do formatu korpusowego, którym jest format Premorph.

Przeprowadzone zostały również badania, które dowiodły, że do celów ekstrakcji tekstów z Internetu można stosować metodę pozycyjną, która opiera się na jednoznaczności pozycji elementów drzewa względem korzenia. Umożliwiło to skonstruowanie łatwego interfejsu, który był jednym z wymagań niefunkcjonalnych systemu.

**6.1. 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.

W związku z brakiem formatu CCL nie stosuję również anotacji na poziomie dokumentów. Jednakże jest to marginalny brak informacji, gdyż większość informacji będzie pobierana z treści notek (jak autor, tytuł, data).

**6.2. 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.

# 7. Literatura

[1] Czym jest korpus językowy? http://korpusy.net

[2] CCL. http://nlp.pwr.wroc.pl/redmine/projects/nlpservices-api/wiki/Ccl

[3] Korpus IPI PAN. http://korpusy.net/korpusy-jezyka-polskiego/korpus-ipi-pan.html

[4] Korpus PWN. http://korpusy.net/korpusy-jezyka-polskiego/korpus-pwn.html

[5] O projekcie NKJP. http://nkjp.pl[5],

[6] Strona projektu Korpus Instytutu Podstaw Informatyki PAN. http://korpus.pl

[7] Korpus IPI PAN. http://korpusy.net/korpusy-jezyka-polskiego/korpus-ipi-pan.html

[8] Kausar, Md Abu, V. S. Dhaka, and Sanjeev Kumar Singh. "Web crawler: a review." *International Journal of Computer Applications* 63.2 (2013).

[9] GNU Wget 1.18 Manual. https://www.gnu.org/software/wget/manual/wget.html

[10] Documentation. https://www.httrack.com/html/index.html

[11] JavaScript HTML DOM. http://www.w3schools.com/js/js\_htmldom.asp

[12] CSS Selector Reference. http://www.w3schools.com/cssref/css\_selectors.asp

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

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

[15] Kocoń, Jan, 2010, Corpus-SUCK, CLARIN-PL digital repository, http://hdl.handle.net/11321/9.

[16] Radziszewski, Adam. "A tiered CRF tagger for Polish." Intelligent tools for building a scientific information platform. Springer Berlin Heidelberg, 2013. 215-230.

[17] Maryl, Piasecki, Kocoń, Oleksy. "Linguistic analysis of weblog genres." Practical Applications of Language Corpora (2014): 79.

[18] Przepiórkowski, Adam. "Korpus IPI PAN. Wersja wstępna." (2004).

[19] Przepiórkowski, Bańko, Górki, Lewandowska-Tomaszczyk. "Narodowy Korpus Języka Polskiego", Wydawnictwo Naukowe PWN, 2012.

[20] Web Crawler. https://en.wikipedia.org/wiki/Web\_crawler

[21] Frequently Asked Questions. http://www.robotstxt.org/faq.html

[22] curl://. https://curl.haxx.se/

[23] Suchomel, Vít, Jan Pomikálek. "Efficient web crawling for large text corpora." Proceedings of the seventh Web as Corpus Workshop (WAC7). 2012.

[24] Shi, Lei i inni. "A dom tree alignment model for mining parallel data from the web." Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics. Association for Computational Linguistics, 2006.

[25] Gupta, Suhit i inni. "DOM-based content extraction of HTML documents." Proceedings of the 12th international conference on World Wide Web. ACM, 2003.

# 8. Załączniki

**Załącznik 1: płyta DVD**

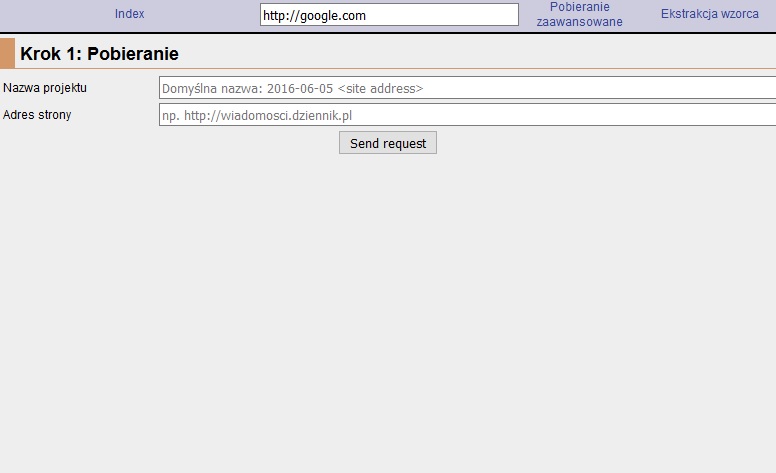
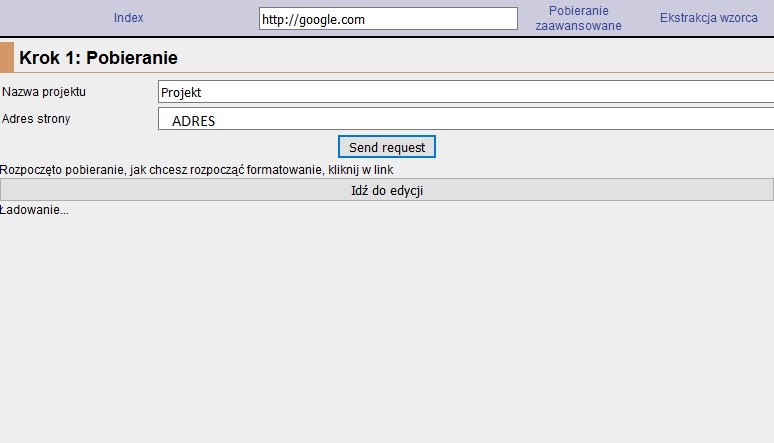
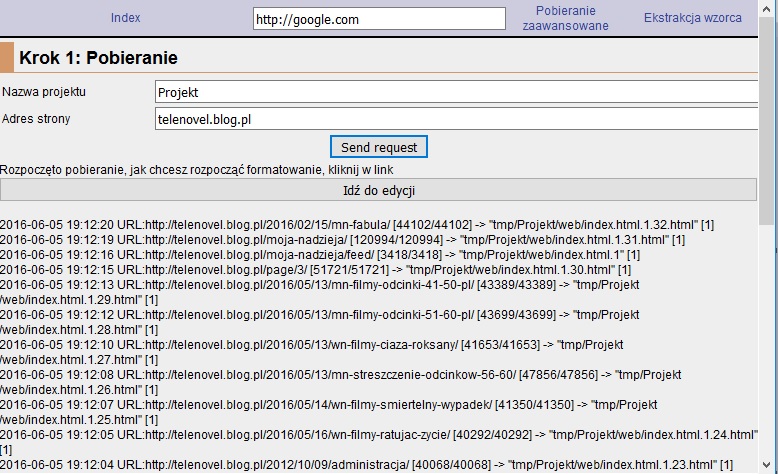
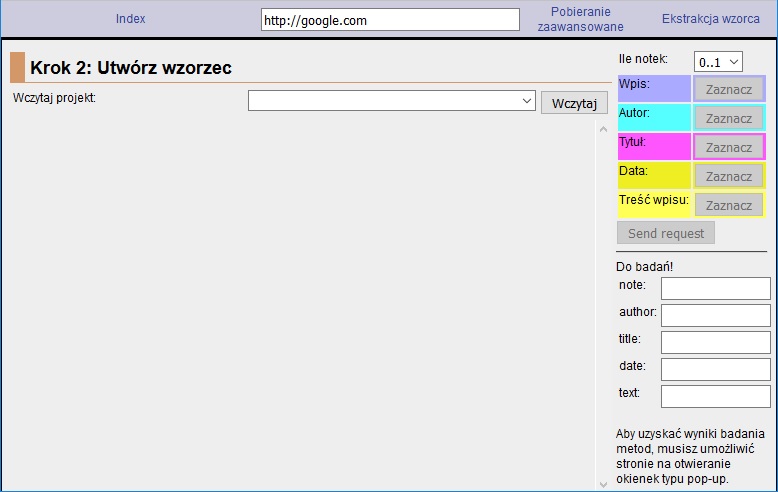
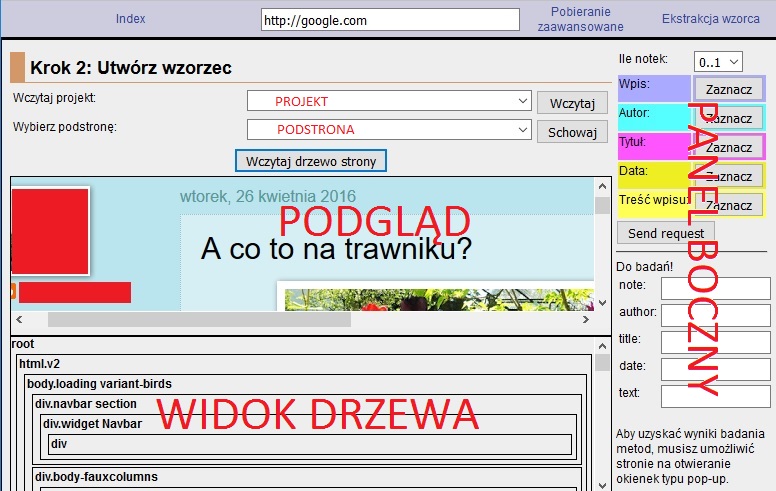
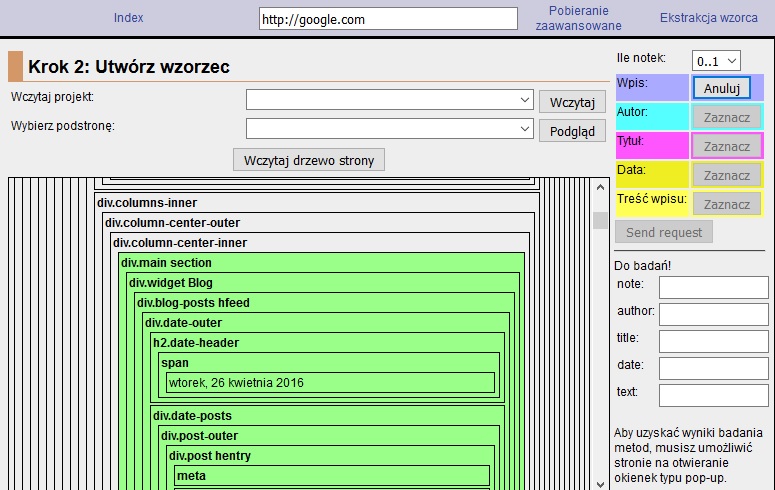
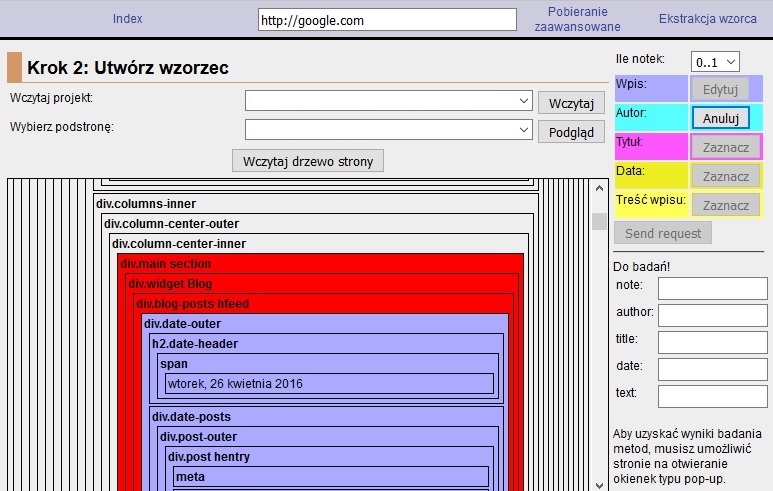
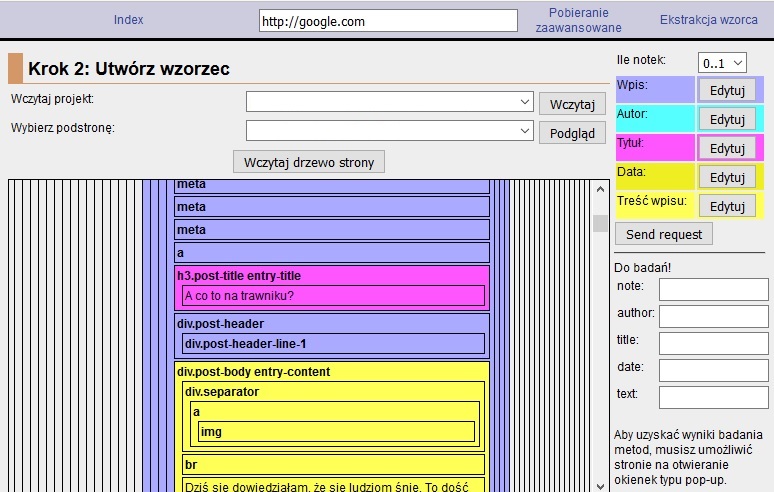
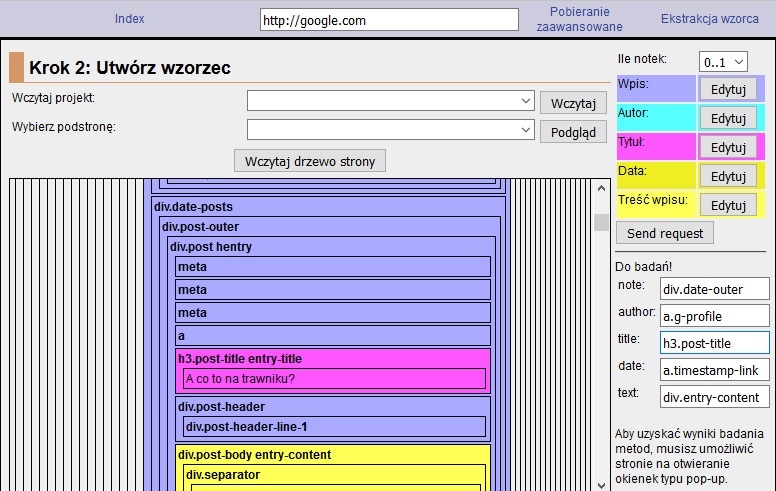
Zawiera:

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: Instrukcja instalacji narzędzia**

1. Zainstaluj oprogramowanie wymagane:
   1. Serwer WWW, jak np. Apache, wspierający język PHP wersja min. 5.5. Optymalne rozwiązanie pod system Windows jest serwer WAMP.
   2. Program WGET
2. Skonfiguruj narzędzia:
   1. Ustaw opcję mod\_rewrite dla Apache, short\_open\_tags dla PHP
   2. W zmiennych środowiskowych systemu Windows udostępnij ścieżkę do programu WGET.
3. Rozpakuj plik projekt.zip w katalogu www od Apache, najlepiej w katalogu pod nazwą corporanet
4. Upewnij się, że uprawnienia administracyjne są przekazane serwerowi, oraz że program WGET uruchamia się przy użycia linii komend w dowolnym folderze
5. Program jest gotowy do użytku!

**Załącznik 3: Instrukcja obsługi narzędzia**

**Krok 1**: W zakładce "Pobieranie Zaawansowane" zdefinuj stronę internetową, którą chcesz pobierać.  
  
Zdefiniuj adres strony i (opcjonalnie) nadaj nazwę projektu. W przypadku nie podania nazwy projekt otrzyma domyślną nazwę.  
Zatwierdź pobieranie przy pomocy przycisku. Pobieranie rozpocznie się od razu i będzie trwać przez parę godzin.  
  
W tym czasie możesz mieć zamkniętą przeglądarkę, ale zalecane jest pozostawienie jej otwartej. Skróci to czas otwierania projektu potem.  
Przycisk "Idź do edycji" aktualnie jest wadliwy, skorzystaj z linku na górze strony.  
  
**Krok 2**: W zakładce "Ekstrakcja Wzorca" zdefiniuj wzorec pobierania dla swojego projektu.  
  
Korzystając z menu na środku wybierz swój projekt. Otwieranie projektu może zająć trochę czasu, czasem są to projekty o rozmiarze paru gigabajtów!  
Po otwarciu możesz wybrać podstronę, według której będziesz wyznaczać wzorzec. Możesz otworzyć jej podgląd korzystając z odpowiedniego przycisku. Gdy już się zdecydujesz, zatwierdź przyciskiem "Wczytaj drzewo strony".  
  
Przed sobą będziesz mieć widok drzewa strony. W menu po prawej masz przyciski służące do zaznaczania elementów na stronie. Po kliknięciu w dowolny będziesz mógl zaznaczyć wybrany element drzewa. Gdy jest możliwe zaznaczenie elementu drzewa, podświetli się na kolorze zielonym  
  
Gdy nie ma możiwości zaznaczenia elementu drzewa, podświetli się na czerwono  
  
Pamiętaj o hierarchii elementów! Notka jest elementem najważniejszym, ona ma opisywać wszystkie pozostałe elementy wpisu, jak autor, tytuł, data i treść. Pozostałe sytuacje są dopuszczalne. Pamiętaj! Jeżeli element podświetla się na zielono, to znaczy, że akcja jest dopuszczalna!  
Jak widzisz, przyciski po prawej są oznaczone kolorami. Takie same kolory będziesz widział, jak oznaczysz te elementy w drzewie. W przypadku, gdy tym samym kolorem oznaczysz ten same element drzewa (gdyż np. data i tytuł znajdują się w tym samym miejscu) zamiast jednego koloru zobaczysz gradient.  
Staraj się, aby jak najlepiej opisywać elementy. Jeżeli widzisz, że tytuł znajduje się w tagu nagłówka (h1, h2, h3), postaraj się ten element oznaczyć, nie oznaczaj tekstu, chyba że jest to konieczne. Przykład takiej konieczności może wystąpić, kiedy w jednym pojemniku znajduje się data, tytuł i autor rozdzielone znanikiem "br".  
  
Po skończonym procesie zaznaczania elementów (przykład powyżej) kliknij przycisk "Send Request". Proces ten jest długotrwały i obciążający dla serwera. Postaraj się nie wykonywać żadnych innych czynności, które mogłyby obciążać serwer, to jest: nie wykonuj kolejnych ekstrakcji wzorca. Możesz rozpoczynać kolejne operacje pobierania.  
Jeżeli chciałbyś przeprowadzić badania, jak zostały zdefiniowane w pracy magisterskiej, wypełnij pozostałe pola formularza regułami selektorowymi. Poniżej zrzut ekranu z wypełnionymi polami.  
  
W momencie zakończenia operacji wszystkie przyciski się odblokują a ty otrzymasz spakowany plik ZIP zawierający korpus w formacie Premorph (XML). Skorzystaj z narzędzia jak WCRFT do konwersji do formatu CCL.

**Załącznik 4: Badania**

Legenda:

|  |  |  |
| --- | --- | --- |
| **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:

|  |  |  |  |
| --- | --- | --- | --- |
| **blog** | **pozycyjnie** | **selektorem** | **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 |