Akademia Górniczo-Hutnicza
im. Stanisława Staszica w Krakowie

Wydział Elektrotechniki, Automatyki, Informatyki i Elektroniki

PRACA MAGISTERSKA

MARCIN ALEKSANDER GADAMER

AUTOMATYCZNA KONTEKSTOWA KOREKTA TEKSTÓW NA PODSTAWIE GRAFU PRZYZWYCZAJEŃ LINGWISTYCZNYCH (LHG) ZBUDOWANEGO PRZEZ ROBOTA INTERNETOWEGO DLA JĘZYKA POLSKIEGO.

PROMOTOR:
dr Adrian Horzyk

Kraków 2009

AGH
University of Science and Technology in Krakow

Faculty of Electrical Engineering, Automatics, Computer Science and Electronics

MASTER OF SCIENCE THESIS

MARCIN ALEKSANDER GADAMER

AUTOMATIC CONTEXTUAL TEXT-CORRECTION TOOL USING THE LINGUISTIC HABITS GRAPH (LHG) CONSTRUCTED BY A SPECIALIST INTERNET ROBOT FOR THE POLISH LANGUAGE.

SUPERVISOR:
Adrian Horzyk Ph.D

Krakow 2009

...

Spis treści

1. Wstęp

Zagadnienie, jakie zostało opisane w tej pracy, to problem automatycznej korekty tekstów. Użytkownik coraz częściej wykorzystuje drogę elektroniczną w codziennej komunikacji z drugą osobą. W dzisiejszych czasach powstaje bardzo duża liczba elektronicznych dokumentów, w których nieodłącznie występują błędy językowe. Istnieje więc potrzeba stworzenia bardziej inteligentnej i kontekstowej korekty tekstu, który został wprowadzony z różnego rodzaju błędami.

Celem niniejszej pracy było skonstruowanie i przetestowanie algorytmu automatycznej kontekstowej korekty tekstu. Zakres pracy obejmował w szczególności zbudowanie specjalistycznego robota (pająka) internetowego, przeszukującego strony internetowe napisane w języku polskim, celem zbudowania maksymalnie rozbudowanego i ogólnego Grafu Przyzwyczajeń Lingwistycznych (LHG) na podstawie analizy tych tekstów. Grafy LHG, zaprojektowane przez dr Adriana Horzyka - promotora pracy, mają za zadanie zliczać wystąpienia różnych słów o pewnej formie fleksyjnej dla wybranego języka w ich rzadkim kontekście początku i końca zdania oraz przede wszystkim w rzeczywistym i bardzo rzadkim kontekście sąsiednich słów o określonej formie fleksyjnej. Dodatkowo w późniejszym etapie mogą one pomóc inteligentnym lingwistycznym cyberbotom zrozumieć wypowiedzi ludzi oraz zbudować własne twierdzenia. Na podstawie grafu LHG należało skonstruować algorytm, służący do automatycznego wykrywania błędów językowych w tekstach oraz do ich poprawy na podstawie kontekstu. Błędy, jakie algorytm miał poprawiać, to w szczególności nieodpowiednia forma fleksyjna, gramatyczna i kontekst użycia wyrazu w zdaniu. Następnie należało stworzyć aplikację wykorzystującą opracowany algorytm. Końcowym etapem pracy było wykonanie testów i porównanie działania algorytmu z dostępnymi mechanizmami poprawy tekstu, wykorzystywanymi w najpopularniejszych edytorach tekstu dla języka polskiego.

Praca została podzielona na 7 głównych rozdziałów.

1. Pierwszy rozdział to wstęp, w którym zawarto krótkie przedstawienie zakresu i celu pracy.

2. Drugi rozdział dotyczy wprowadzenia w tematykę pracy. Zdefiniowano w nim pojęcie ”lingwistyki komputerowej”, opisano w skrócie jej historię oraz przedstawiono główne problemy przed jakimi stoi.

3. Trzeci rozdział prezentuje najpopularniejsze metody korekcji tekstu, jakie są wykorzystywane na świecie. W tym rozdziale poruszone zostały m.in. takie metody jak: odległość edycyjna (odległości Levensteina), metoda N-gramu, prawo Zipfa, metoda odległości na klawiaturze.

4. Czwarty rozdział zawiera opis własnego rozwiązania. Zawarto w nim szczegółowy opis powstałych kolejnych modułów aplikacji. Są nimi:

   - moduł związany z implementacją pająka internetowego,
   - moduł związany z konstrukcją grafu LHG,
   - moduł związany z automatycznym kontekstowym sprawdzaniem wprowadzanych wyrazów,
   - moduł związany z dopełnianiem końców wyrazów,
   - moduł związany z mechanizmem automatycznej korekty tekstów.

5. Piąty rozdział dotyczy porównania wyników korekty tekstu w skonstruowanym algorytmie z najpopularniejszymi edytorami tekstu, takimi jak Microsoft Word 2007 oraz Open Office Writer 3.0.

6. Szósty rozdział zawiera spostrzeżenia i wnioski z przeprowadzonych testów.

7. Siódmy rozdział to podsumowanie, w ramach którego mieści się krótkie zestawienie stawianych celów pracy z wykonaną aplikacją i zawartym w niej algorytmie do automatycznej kontekstowej korekty tekstów.

2. Wprowadzenie

2.1. Historia lingwistyki

Lingwistyka powstała dopiero w XX wieku, przedtem była znana jako filologia. Zajmowano się wówczas analizą tekstów minionych epok, egzegezą, interpretacją historyczną (etymologia), pochodzeniem oraz pokrewieństwem języków.

Lingwistyka I połowy XX wieku, to strukturalizm (F. de Saussure, J.N. Baudouin de Courtenay, M. Kruszewski, O. Jespersen, N. Trubeckoj, L. Hjelmslev, L. Tesniére, J. Kuryłowicz, L. Bloomfield, Z. S. Harris) [12]. Powstały wtedy opisy struktury, a nie genezy. Język naturalny natomiast, był to dwuklasowy system semiotyczny służący danej populacji do komunikacji uniwersalnej.

Następną rewolucją była lingwistyka formalna. Zajmowano się wtedy opisem *typu matematycznego* – twierdzenia, dowody, rachunki. Wynikiem prac była definicja drzew struktury. Język naturalny był więc zbiorem wszystkich wyrażeń poprawnych i tylko takich. Za przedstawiciela generatywizmu uznawany jest N. Chomsky (Syntactic Structures 1957). Wprowadzono wówczas również inne formalizmy, jak na przykład GPSG, HPSG, DCG oraz wiele innych.

Ostatnim znaczącym wydarzeniem było powstanie lingwistyki informatycznej. Zajmowała się ona zadaniami typowo inżynierskimi – budowa korpusu, automatyczna analiza tekstu, przetwarzanie tekstów, przeszukiwanie itp. Opisem jak tego dokonać był program komputerowy, a wynikiem prac było m.in. określenie podzbioru korpusu, lista składników spełniających zadane warunki, rzeczywiste drzewa struktury dla danego wyrażenia (*parsing trees*).

2.2. Lingwistyka komputerowa - podział

Lingwistyka komputerowa stanowi jeden z działów lingwistyki. Jest ona obecna zarówno w językoznawstwie, psychologii kognitywnej, czy filozofii, ale również w matematyce (należy tutaj wymienić takie działy jak teoria automatów, języki formalne, teoria grafów, statystyka, logika matematyczna oraz rachunek prawdopodobieństwa), informatyce i sztucznej inteligencji. Jednym z zadań, jakie stoją przed lingwistyką komputerową, jest przetwarzanie języków naturalnych za pomocą komputerów, a także synteza informatycznych i językoznawczych metod oraz wiadomości [5].

Pod ogólną nazwą *lingwistyka komputerowa* kryje się tak naprawdę wiele dziedzin:

- Dyscyplina językoznawstwa (ang. *computational linguistics*).
- Dyscyplina wykorzystująca komputery do gromadzenia i przetwarzania danych lingwistycznych.
- Dyscyplina realizująca zjawiska językowe na komputerach (ang. *natural language processing*).
- Inżynieria lingwistyczna (ang. *language technology*).

Rozważany tutaj dział nauki, można również podzielić od strony teoretycznej i praktycznej. W skład zagadnień lingwistyki komputerowej teoretycznej można zaliczyć: badanie złożoności obliczeniowej formalizmów, możliwość opisu formalizmów jak i możliwość automatycznej nauki i kategoryzacji znaczących podzbiorów językowych. Zagadnieniami lingwistyki komputerowej praktycznej jest: tworzenie formalizmów modelujących różne aspekty języków naturalnych, udostępnianie wiedzy o poszczególnych językach, jak również tworzenie algorytmów i metod do przetwarzania wypowiedzi językowych.

Dzięki tak wielkiej różnorodności i bogactwu lingwistyki komputerowej znajduje ona zastosowanie w wielu dziedzinach życia. Służyć może do ekstrakcji informacji z dużych nieuporządkowanych źródeł, np. Internetu, archiwów elektronicznych, bądź stanowić interfejs językowy do systemów informatycznych. Swoje zastosowanie znajduje również przy tłumaczeniu maszynowym oraz automatycznym, rozpoznawaniu mowy i tekstów oraz ich generowaniu (np. powstające syntezatory mowy).

2.3. Historia lingwistyki komputerowej

Pierwsze koncepcje. Początki lingwistyki komputerowej sięgają lat ’40 i ’50 ubiegłego wieku. Rozpoczęła się ona powstaniem koncepcji automatu (Turing 1936). Następnie pojawiały się automaty skończone i zbiory regularne (Kleene 1951 - 1956). Połowa lat ’50 natomiast, przyniosła prace nad teorią języków formalnych (Chomsky 1956).

Dwa podejścia. W okresie pomiędzy latami 1957 - 1970 mamy do czynienia z dwoma podejściami. Jedno z nich oparte jest na metodach symbolicznych (gramatyka generatywna, parsery syntaktyczne oraz sztuczna inteligencja), drugie natomiast, na metodach statycznych (Metoda Bayes’a, optyczne rozpoznawanie liter, identyfikacja autorów tekstów).

Cztery paradygmaty. Następnym krokiem w rozwoju lingwistyki komputerowej było pojawianie się czterech paradygmatów:

1. Statystyczne modele (HMM do rozpoznawania mowy, synteza mowy).
2. Logika formalna (Prolog, DCG, LFG).
3. Rozumienie języków naturalnych (Winograd: Block World).
4. Modelowanie dyskursu.

Odrodzenie modeli. Lata ’80 i początek ’90 to moment, w którym odrodzeniu ulegają modele skończonych stanów i empiryzmu. Powstaje wtedy wiele modeli, z których można wymienić m.in. prace Churcha (1980) nad modelem skończonych stanów składni, prace nad morfologią i fonologią, również za pomocą modeli skończonych stanów (Kaplan & Kay 1981) oraz prace nad metodami stochastycznymi, które daleko wykraczają poza rozpoznawanie mowy (rozpoczęte przez firmę IBM).

Lata ’90. Do początku lat ’90 prace nad lingwistyką komputerową odbywały się na wielu płaszczyznach, przez co powstało wiele różnych kierunków rozwoju. Okres lat dziewięćdziesiątych przyniósł połączenie dotychczasowych ich dróg. Pojawiły się prace wykorzystujące metody statystyczne w metodach symbolicznych, wzrosło zapotrzebowanie na systemy, które byłyby w stanie przeprowadzić ekstrakcję informacji (przyczynił się do tego znaczący wzrost znaczenia Internetu), nastąpił wzrost wydajności komputerów, co pozwoliło na komercyjne wykorzystanie istniejących już wyników badań i metod (prace nad rozpoznawaniem mowy, korekcja ortografii, zdań).

2.4. Główny problem lingwistyki komputerowej

Głównym problemem przed jakim staje lingwistyka komputerowa to wieloznaczność wyrazów oraz określeń. Występuje ona na wszystkich poziomach systemu językowego:

- w fonetyce, np. homofonia (tożsamość różnych znaków językowych),

- w morfologii, np. problem analizy części mowy, homonimia (wyrażanie różnych znaczeń za pomocą identycznych form językowych),
- w składni, np. problem analizy części zdania,
- w semantyce, np. polisemia (wyraz ma więcej niż jedno znaczenie w zależności od kontekstu),
- w pragmatyce, np. metafory (wyrazy stojące w określonym porządku obok siebie mają inne znaczenie).

Wszystkie wieloznaczności, z jakimi można się spotkać w naturalnym języku, trzeba często rozstrzygać korzystając z wyższego poziomu języka. Jeśli dla człowieka nie stanowi to dużego problemu, to już dla programistów programów komputerowych jest to jedno z największych wyzwań, z jakimi muszą się zmierzyć.

2.5. Przegląd procesów nominalizacji

Procesem nominalizacji możemy nazwać dążenie do przekształcenia zadania tak, aby zmieniając jego treść zachować przekaz jaki ze sobą niesie. Problematyka nominalizacji pozostaje w ścisłym związku z wieloma złożonymi zagadnieniami wzajemnych relacji semantyczno-syntaktycznych, między zdaniem pojedynczym, a złożonym i innymi typami konstrukcji składniowych [4].

Wielorakie powiązania pomiędzy różnymi aspektami opisu języka w poszczególnych jego podsystemach, były wielokrotnie obiektem zainteresowań lingwistyki. W tym podrozdziale dokonano przeglądu pojawiających się problemów nominalizacji, powołując się w szczególności na pozycję [4].

Tradycyjna składnia (przedstrukturalistyczna) nie zajmowała się zjawiskiem nominalizacji, uznając przykładowo słowa *myśleć, myślenie, myśl* za problem słowotwórczy, związany z klasą derywowanych nazw czynności lub cech.

Wypowiedzi typu *Janek ucieszył się z przyjazdu matki* oraz *Janek ucieszył się, że matka przyjechała* opisywane były oddzielnie. Jedno w obrębie tzw. składni zdania pojedynczego, drugie – w ramach składni zdań złożonych. Aparat pojęciowy składni tradycyjnej pozwalał ustalić jedynie podobieństwa funkcjonalno-składniowe. Przyjmowane kryteria klasyfikacyjne dotyczyły bowiem różnic lub podobieństw funkcjonalnych, między poszczególnymi typami wypowiedzeń. Ekwiwalencja syntaktyczna ustalana była między zdaniem podrzędnym, jako funkcjonalną całością, a odpowiadającymi jej częściami zdania prostego. W konsekwencji składniki wypowiedzi, należące do różnych kategorii morfologicznych, traktowane były jako nie tożsame i charakteryzowane niezależnie od siebie.

Oddzielenie płaszczyzny treści od płaszczyzny wyrażenia oraz wprowadzenie pojęć ”struktura głęboka” i ”struktura powierzchniowa” (formalna, formalno-gramatyczna) pozwalało ukazać realizację danej treści, za pomocą wyrażeń ukształtowanych, w sposób określony wymaganiami gramatyki. Umożliwiło również zwrócenie uwagi na semantyczne związki między konstrukcjami różniącymi się formą, składem leksykalnym i funkcją poszczególnych elementów wypowiedzi.

Zasadniczą rolę odegrała tu gramatyka generatywno-transformacyjna i związane z nią późniejsze koncepcje, zwłaszcza semantyka generatywna i składnia semantyczna. Pojęcie nominalizacji pełni w tym miejscu ważną rolę, gdyż umożliwia adekwatny opis struktury wypowiedzeń i ich poprawną interpretację (także z punktu widzenia parafrazy). Wyprowadzenie grup nominalnych typu *przyjazd mamy Janka* ze zdań pozwala m. in. w sposób systemowy wyjaśnić wiele konstrukcji dwuznacznych, które poza kontekstem i sytuacją można traktować dwojako: albo czynnie, albo biernie. Interpretacja grupy *przyjazd mamy Janka* jako transformacji jednego lub drugiego zdania, pozwala wyjaśniać formalnie tego typu dwuznaczności, nie ograniczając się, jak to robiono poprzednio, do przypisywania poszczególnym konstrukcjom tylko odrębnych terminów. Sposób traktowania (opisu i interpretacji) nominalizacji, rozumianej jako proces lub wytwór procesu przekształcania, zmieniał się jednak wraz z ewolucją teorii generatywistycznych, na co wpływały też semantyzujące koncepcje logiki i filozofii języka.

We wczesnej wersji gramatyki T–G (Chomski, 1957 i nast.) nominalizacje ujmowane były szeroko jako rezultat transformacji, zastosowanej do wyjściowej struktury zdaniowej, wygenerowanej przez reguły bazowe, przekształcając zdanie (S) w derywowaną frazę nominalną (N_{PI}). Symbolicznie operację taką można zapisać za pomocą formuły:

$$T_{NOM} : S_{(NP.VP)} \rightarrow NP^{der}_{(NP.N_{deverb})}.$$

Przy takim podejściu produktem procesu nominalizacyjnego były wszystkie rzeczowniki słowotwórczo pochodne, dla których można było zbudować synonimiczną parafrazę o postaci zdania.

Pionierską w tym względzie była praca Leesa (1964), który podjął próbę systematycznego opisu nominalizacji angielskich rzeczowników złożonych (podzielnych słowotwórczo derywatów nominalnych, złożeń i zrostów), traktowanych jako wynik transformacji zdań bazowych. Ujęcie Leesa było zdeterminowane przez fakt, że autor operował dość ograniczoną formułą wersji z Syntactic structures (Chomski, 1957), w której leksykon – także dość ograniczony – był jeszcze częścią bazowego komponentu kategorialnego. W ramach tej formuły nie można było ukazać relacji między V_{erb}, a N_{deverb} inaczej niż przez przyjęcie hipotezy transformacyjnej.

N. Chomsky (1970) wyłożył swój pogląd na istotę tego zjawiska, rozpatrując nominalizacje w aspekcie sprawdzania siły wyjaśniającej teorii, zgodnie ze swoją ówczesną koncepcją gramatyki z bazą syntaktyczną. Punktem wyjścia jego rozważań było pytanie: *"czy istotnie każda nominalizacja może być uznana za wynik transformacji odpowiedniego zdania bazowego?"* Teoria ta zakładała, że pewne relacje gramatyczne, definiowane przez określoną konfigurację elementów kategorialnych, mogą być wyjaśniane bądź w drodze wzbogacania informacji, zawartych w leksykonie, co pozwala uprościć komponent kategorialny bądź przez rozbudowanie reguł transformacyjnych. Chomsky sądził przy tym, że dla pewnego typu zjawisk wyjaśnieniem lepszym – z punktu widzenia eksplicytności reguł jego gramatyki – jest przyjęcie hipotezy transformacjonalistycznej, dla innych zaś – przyjęcie tzw. hipotezy leksykalistycznej, wykorzystującej idiosynkratyczne cechy słowa opisanego w leksykonie. O wartości wybranej procedury decydują wyniki badań empirycznych.

Chomski rozważał możliwość interpretacji wielu nominalnych konstrukcji, które korespondują ze zdaniami. W rezultacie wyróżnił trzy typy nominalizacji angielskich:

1. nominalne konstrukcje gerundialne (kategorialne NA),
2. nominalne konstrukcje z rzeczownikiem derywowanym od czasownika lub przymiotnika (niekategorialne),
3. nominalizacje tzw. mieszane, które mogą być interpretowane jako typ 1 lub 2 (ten ostatni typ nie posiada odpowiednika w języku polskim).

Poglądy Noama Chomskiego na mechanizm nominalizacji zawierały dwa ważne momenty, które miały swoje konsekwencje dla dalszej ewolucji teorii generatywnej:

1. Były rodzajem rewizji teoretycznych założeń rozszerzonego modelu gramatyki i próbą interpretacji w jego ramach pokrewnych wypowiedzeń o różnej strukturze formalnej.
2. Mimo zasadniczej orientacji na związki formalno-gramatyczne doceniały potrzebę uwzględnienia słownictwa – i szerzej – czynnika semantycznego w adekwatnym opisie gramatycznym.

Słabością tego aparatu teoretycznego, który zakładał kolejność operacji generowania wypowiedzi: najpierw składnia (głęboka struktura syntaktyczna), później interpretacja znaczenia, tłumaczyć można „hybrydalny", w pewnym sensie charakter hipotezy leksykalistycznej, polegający na tym, że zasadnicze informacje, które musiałby zawierać leksykon, dotyczyłyby zarówno wymagań kategorialno-gramatycznych, jak i semantycznych. Prowadziłoby to do niepomiernego rozbudowania opisu kontekstów, w które może wchodzić jednostka słownikowa. Niemniej sama propozycja oraz jej krytyczne omówienie w pracach Frasera (1970), Leesa (1970), Robinson (1970), H. Esau'a (1973) i in., są ważne w kontekście gwałtownych dyskusji, wśród przedstawicieli lingwistyki generatywnej przełomu lat sześćdziesiątych i siedemdziesiątych, na temat statusu struktury głębokiej, w opisie gramatycznym oraz roli i charakteru leksykonu, skorelowanego z tak pojętą gramatyką. Zakwestionowano przede wszystkim kolejność etapów generowania

wypowiedzi od głębokiej struktury składniowej, proponując bardziej naturalny, zdaniem wielu badaczy (G. Lakoff, J. D. McCawley i in.) kierunek: od treści (semantyki) na wejściu, do struktury formalnej na wyjściu urządzenia generującego, jakim jest gramatyka.

Zagadnienie nominalizacji odegrało ważną rolę w tej dyskusji. Trudności z ich wyjaśnieniem wywołały wiele kontrowersji wokół natury pojęcia struktury głębokiej i takiego sposobu przekodowania jej na wyrażenia i konstrukcje powierzchniowe, który ukazywałby odpowiedniości między synonimicznymi frazami nominalnymi i zdaniem. W kilka lat po swojej pracy o nominalizacjach, omawiając nowsze osiągnięcia lingwistyki (Lakoffa, Fillmore'a, Rossa i in.), o tych trudnościach mówi Lees, podkreślając, iż generowanie wypowiedzeń rozpoczyna się od składni, rodzi się wtedy konieczność ogromnego zwiększenia liczby abstrakcyjnych reguł, które mogłyby objąć wszystkie warunki przekształcania struktury głębokiej w powierzchniową. Uzasadnia to potrzebę odróżnienia i odrębnego, choć skorelowanego, opisu dwu płaszczyzn: semantycznej jako pierwszej i gramatycznej.

Zwrócenie uwagi na synonimię konstrukcji werbalnych i nominalnych, realizowanych za pomocą różnych środków gramatycznych, podważyło tezę o autonomii składnika syntaktycznego i prymarności głębokiej struktury syntaktycznej. W rezultacie doprowadziło to do sformułowania koncepcji semantyki generatywnej, która zakładała głęboki model zdania o zadanym znaczeniu, posiadający "na wejściu" gotową (zinterpretowaną) głęboką strukturę semantyczno-składniową. Tak pojęta uniwersalna struktura głęboka, może być różnie realizowana formalnie, zgodnie z możliwościami danego języka naturalnego.

Ważną rolę odegrały tu prace W. L. Chafe'a, G. Fillmore'a (zwłaszcza jego teoria przypadków głębokich i presupozycji), G. Lakoffa, D. Jackendoffa i innych językoznawców, którzy postulowali specjalny metajęzyk semantyczny dla zapisu struktury treści, wolny od ograniczeń składni formalnej (kategorialno-gramatycznej), który dysponowałby własnymi jednostkami, możliwymi do przekodowania na poziom formalno-syntaktyczny. Między obiema płaszczyznami, semantyczną i składniową, istnieją bowiem wyraźne i silne zależności, choć nie mają one charakteru symetrycznego. Dla semantycznego zapisu zdania wykorzystano logiczne pojęcia predykatu i argumentu, jako treściowych składników struktury sądu logicznego.

Za etap pośredni pomiędzy modelem T–G gramatyki Chomskiego, a semantyką generatywną, można uznać interesującą próbę połączenia hipotezy leksykalistycznej z nowszymi ujęciami semantycznymi (z teorią przypadków głębokich), dla wyjaśnienia zjawiska nominalizacji, przedstawioną przez H. Esau'a (1973). Badając reguły nominalizacji, autor usiłuje wniknąć w strukturę leksykonu, ukazać sposób, w jaki prymarne klasy (Verb, Noun, Adj) mogą wchodzić w większe kategorie syntaktyczne oraz odpowiedzieć na pytanie czy baza generuje semantyczne, czy syntaktyczne struktury wyjściowe. Punktem wyjścia rozważań H. Esau'a jest krytyczne omówienie nominalizacji w ujęciu Chomsky'ego i wykazanie, że mimo zastosowania konwencji X-barowej – nie jest ono wolne od niekonsekwencji.

Niezależnie od badań nad zjawiskiem nominalizacji, prowadzonych w nurcie generatywnym, na dalszy rozwój i kolejne modyfikacje teorii semantycznych wpłynęły równocześnie: filozofia języka (szczególnie tzw. szkoła oksfordzka), zainteresowana realizacją języka w akcie mowy oraz semantyka logiczna, badająca wartość zdania w kategoriach prawdy i fałszu, zainteresowana pewnym szczególnym typem predykatów tzw. propozycjonalnych (w polskiej terminologii przyjął się dla nich termin - predykaty II stopnia, które implikują argumenty tzw. zdarzeniowymi lub propozycjonalnymi).

Spośród sugestii, inspirujących nowe aspekty badań lingwistycznych, dwie wydają się szczególnie istotne dla zagadnienia nominalizacji:

- Zwrócono uwagę na fakt, że zdania mogą spełniać różne funkcje komunikatywne: jako stwierdzenia (konstatacje) – nie tylko prawdziwie lub fałszywie, oznajmiają o faktach (przez wieki tylko ten typ komunikatu uważany był za zdanie w sensie logicznym), ale mogą też mieć postać pytającą lub rozkazującą, przy czym poprzez ich wypowiadanie mówiący może dokonywać różnych aktów illokucyjnych, jak: akt pytania, rozkazu, prośby, groźby, obietnicy itp. Rodzaj aktu mowy zależy od tzw. *"siły illokucyjnej"* wyrażanej przez tzw. czasowniki performatywne. Oznaczało to zrównanie stwierdzeń z innymi aktami mowy i zainteresowanie strukturą formalną tych wypowiedzeń.

- Zainteresowano się właściwościami wyrażeń (głównie czasowników typu przepraszać, grozić, obiecywać, ślubować itp. które reprezentują predykaty relacji propozycjonalnej), zdolnych do pełnienia funkcji *"ramy illokucyjnej"* dla różnorodnych treści implikowanych oraz formą, w jakiej treści te, mogą się realizować w języku. Zauważono, że wspólną właściwością tych predykatów jest to, iż wymagają one uzupełnienia zdaniem lub jego funkcjonalnym ekwiwalentem nominalnym – czyli konstrukcją znominalizowaną, o różnym stopniu kondensacji. Całkowita wartość konstrukcji przez nie tworzonych jest różna wobec wartości prawda lub fałsz.

Jednym z pierwszych badaczy, którzy wiązali teorię aktów mowy i tzw. performatywów z teorią propozycji i czasownikami relacji propozycjonalnej był Z. Vendler (1970). Poddając analizie z tego punktu widzenia czasowniki angielskie, Vendler dokonał ich klasyfikacji, uwzględniając różne kryteria semantyczne i formalne, m. in. także kryterium zdaniowej lub niezdaniowej realizacji implikowanego argumentu zdarzeniowego.

Poglądy Z. Vendlera pozostają w pewnym związku z koncepcją P. i C. Kiparsky'ch (1970), dotyczącą wydzielenia czasowników tzw. faktywnych i niefaktywnych, które konotują wzajemnie przekształcalne konstrukcje w funkcji głębokiego subiektu lub obiektu. Badając semantykę tych czasowników i tworzonych przez nie zdań, zwrócili uwagę na ich różną wartość logiczną i jej zależność od semantycznej klasy orzeczenia.

W tym samym kręgu zainteresowań pozostaje też po części praca N. D. Arutiunowej (1976), która wyróżnia trzy typy znaczeń nominalizowanych struktur propozycjonalnych.

Zarówno klasyfikacje Z. Vendlera, P. i C. Kiparskich, jak N. D. Arutiunovej budzą wiele wątpliwości, zasługują jednak na uwagę z kilku względów:

- Stanowią próby głębszego, niż to czyniono wcześniej, wniknięcia w semantykę zdań o skomplikowanej treści i różnej budowie formalnej z jednoczesnym uwzględnieniem czynników pozasystemowych – pragmatycznych i komunikatywnych.
- Zwracają uwagę na pewien typ wyrażeń, które – jakkolwiek należą do różnych kategorii gramatycznych – posiadają podobne lub identyczne właściwości.
- Reprezentują odmienny, niż generatywistyczny, punkt widzenia, wskutek czego odsłaniają te aspekty nominalizacji, których nie brała pod uwagę semantyka generatywna.
- Dowodzą konieczności oddzielenia różnych płaszczyzn opisu konstrukcji językowej i wielostronnego spojrzenia na dane zjawisko.

Przedstawione zostały różnorodne poglądy na zagadnienie nominalizacji (rozumianej jako proces lub jego wytwór), stanowiące jeden z istotnych przykładów asymetrii płaszczyzn: formalnogramatycznej (syntaktycznej), semantycznej i pragmatycznej, przy równoczesnym silnym ich wzajemnym uwarunkowaniu. Różne kierunki i koncepcje współczesnej lingwistyki determinowały sposób ujmowania nominalizacji z różnych punktów widzenia, oświetlając zjawisko asymetrii i uwarunkowań istniejących między podsystemami języka.

3. Wykorzystywane metody rozpoznawania tekstu na świecie

Postęp występuje w każdej dziedzinie życia. Przez niektórych jest on rozumiany jako naturalny proces ewolucyjny, u niektórych natomiast wynika z zaciekawienia światem, możliwości odkrywania nowych zjawisk, bądź odpowiedzi na próbę ułatwiania codziennego życia.

Postęp dotyka wielu sfer życia człowieka, szczególnie tych, które towarzyszą mu w jego codzienności. Można dziś łatwo zauważyć bardzo szybki postęp technologiczny, a także ewolucję, jakiej uległ również sposób komunikacji międzyludzkiej oraz związana z nim przemiana języka. Początkowo ludzie pierwotni komunikowali się przez pojedyncze słowa. Komunikacja na odległość była możliwa jedynie dzięki gestom oraz obrazom. Sposób komunikacji z biegiem czasu uległ jednak zmianie. W dzisiejszych czasach wspomniane wcześniej tradycyjne środki komunikacji odeszły na dalszy plan, oddając miejsce nowoczesnej technologii. W dzisiejszych czasach chyba już nikt nie wyobraża sobie komunikacji bez telefonu (stacjonarnego bądź komórkowego) lub bez pośrednictwa Internetu. Koniec XX wieku przyniósł silny rozwój technologiczny, a powstające po dziś dzień coraz to nowe komputery oraz oprogramowanie, dostarcza wielu zaawansowanych możliwości pomocy człowiekowi w codziennej pracy.

Rozwój technologii w każdym wymiarze pociąga za sobą zarówno pozytywne jak i negatywne skutki. Do tych drugich, w procesie rozwoju metod komunikacji ostatnich lat, można z pewnością zaliczyć pojawiające się coraz częściej błędy językowe, we wprowadzanych tekstach. Na ich pojawianie się ma wpływ wiele różnorakich czynników. Główny podział błędów można wprowadzić ze względu błędy użytkownika oraz samego programu komputerowego. Do najważniejszych błędów użytkownika można zaliczyć:

- brak znajomości zasad konstrukcji zdań,
- brak pełnego przekazu treści (skróty myślowe, zdrobnienia),
- brak staranności przy wpisywaniu tekstu,
- pośpiech użytkowników programów przy wprowadzaniu tekstu.

Do błędów programów natomiast zalicza się:

- brak specjalistycznych programów do obsługi języka polskiego.
- brak badania kontekstu wprowadzanego teksu (kontekst wyrazów i zdań).
- brak badania wprowadzenia przypadkowo poprawnych / błędnych wyrazów.

Lingwistyka komputerowa bądź inżynieria lingwistyczna, to jedne z wielu nazw tej samej dziedziny obejmującej próby zautomatyzowania przetwarzania danych w języku naturalnym. Główne, rozpatrywane aktualnie zadania lingwistyki to: wyszukiwanie, klasyfikacja i selekcja informacji wyrażonych za pomocą języka naturalnego, a także maszynowe tłumaczenie tekstów, z jednego języka na drugi oraz streszczanie i analiza mowy. Związane z tymi zastosowaniami badania dotyczą opisu różnych poziomów wiedzy o języku: morfologii, składni, semantyki i pragmatyki wypowiedzi w języku naturalnym, budowania zasobów lingwistycznych, słowników jedno- i wielojęzycznych, dużych korpusów tekstów znakowanych różnego typu informacjami oraz tworzenia programów do analizy danych językowych [10].

Podczas wprowadzania tekstu wielokrotnie popełniamy błędy. Jednym z zadań niektórych programów jest więc ich poprawa, tak aby zdania wprowadzane z błędami były korygowane i przekształcane na zdania poprawne (w różnym ujęciu. m.in. ortograficznym, gramatycznym, fleksyjnym), z punktu widzenia danego języka naturalnego. Na świecie istnieje kilka sposobów i metod korekcji takich zdań. W rozdziale tym przedstawiono i bliżej scharakteryzowano niektóre z nich.

3.1. Metoda odległości Levensteina

3.1.1. Teoria

Metoda odległości Levensteina jest jedną z najbardziej popularnych metod korekcji tekstów. Odległość edycyjną (później od nazwiska autora nazwaną metodą odległości Levensteina) wprowadził w 1966r. V.I. Levenshtein. Metoda odległości edycyjnej polega na porównaniu dwóch słów i próbie przekształcenia jednego z nich w drugie, wykorzystując przy tym dostępne operacje edycyjne (wstawienie bądź usunięcie litery jak również zastąpienie litery inną). Każdej takiej operacji przypisywany jest pewien koszt i wyszukiwane jest przekształcenie, dla którego koszt jest najmniejszy. Ów najmniejszy koszt jest odległością edycyjną danych dwóch słów [9].

Levenstein nie zaproponował algorytmu, który mógłby obliczać odległość edycyjną. Jednak wkrótce po opublikowaniu jego dzieła, inni autorzy skonstuowali liczne algorytmy korekty. Jeden z pierwszych takich algorytmów, służący do porównywania białek, podali S.B. Needleman i C.D. Wunsch w 1970 roku. Była to jedna z pierwszych prac z biologii obliczeniowej, teoretycznej gałęzi bioinformatyki [9].

Ciągi operacji edycyjnych reprezentuje się najczęściej za pomocą uliniowienia. Uliniowieniem dwóch słów $X = x_1x_2x_3...x_n$ oraz $Y = y_1y_2y_3...y_m$ jest dwuwierszowa macierz, w której w górnym wierszu występują kolejne litery pierwszego słowa, natomiast w dolnym wierszu występują litery drugiego słowa. Wystąpienie w jednej kolumnie litery x_i ze słowa X i litery y_j ze słowa Y oznacza zastąpienie x_i przez y_j.

Wykorzystując odległość edycyjną, rozpatrujemy słowa nad pewnym wcześniej ustalonym alfabetem Σ. Należy również określić macierz kosztu $(\delta(a,b))_{a,b\in\Sigma}$, w której $\delta(a,b) = \delta(b,a)$ oraz $\delta(a,b) = 0$, gdy $a = b$ Należy także przyjąć równy lub różny koszt dla każdej czynności edycyjnej: wstawienia, usunięcia lub zamiany litery. Koszt uliniowienia dwóch słów $X = x_1x_2x_3...x_n$ oraz $Y = y_1y_2y_3...y_m$ powstaje, więc przez dodanie wszystkich wartości $\delta(x_i, y_j)$ dla każdej kolumny uliniowienia, która łączy wyraz x_i z pierwszego słowa z wyrazem y_j drugiego słowa. Uliniowienie X i Y jest optymalne, gdy koszt wszystkich przekształceń słowa pierwszego w drugie jest najmniejszy.

3.1.2. Przykłady obliczania odległości Levensteina (edycyjnej)

Dla poniższych przykładów koszt dowolnego przekształcenia jest stały i wynosi 1.

1. Dla wyrazów *parasol* oraz *parasol* odległość jest zerowa, gdyż nie potrzeba żadnych działań, aby pierwszy wyraz przekształcić w drugi.

2. Dla wyrazów *granat* oraz *granit* odległość wynosi 1, ponieważ aby przekształcić wyraz *granat* w *granit* należy wykonać jedną operację - zamienić literę *a* na literę *i*.

3. Dla wyrazów *torba* oraz *torbacz* odległość wynosi 2, ponieważ aby przekształcić wyraz *torba* w *torbacz* należy wykonać dwie operacje - dodać dwie litery *c* oraz *z*.

4. Dla wyrazów *orczyk* oraz *oracz* odległość wynosi 3, ponieważ aby przekształcić wyraz *orczyk* w *oracz* należy wykonać trzy operacje - dodać literę *a* oraz usunąć dwie litery *y* oraz *k*.

5. Dla wyrazów *marka* oraz *arkada* odległość wynosi 3, ponieważ aby przekształcić wyraz *marka* w *arkada* należy wykonać trzy operacje - usunąć literę *m* oraz dodać dwie litery *d* oraz *a*.

6. Dla wyrazów *foka* oraz *kotka* odległość wynosi 2, ponieważ aby przekształcić wyraz *foka* w *kotka* należy wykonać dwie operacje - zamienić literę *f* na *k* oraz wstawić literę *t*.

3.1.3. Przykład z użyciem tabeli do analizy odległości edycyjnej

Przykład będzie badał odległość edycyjną dla dwóch słów: *foka* oraz *kotka*.

Tablica 3.1: Utworzona tablica do badania odległości edycyjnej

		k	o	t	k	a
	0	1	2	3	4	5
f	1					
o	2					
k	3					
a	4					

Pierwszy wiersz liczbowy i kolumnę liczbową uzupełniane są wartościami od 0 do odpowiednio n i m, w zależności od długości słowa. Następnie pobiera się wartości wiersza i porównuje literę dotyczącą danego wiersza z literą dotyczącą kolumny. Porównywania liter dokonywane są na zasadzie każdy z każdym. Przy każdym porównaniu wykorzystywana jest poniższa procedura. Jeśli litery są identyczne, wpisywany jest koszt 0, jeśli litery są różne wpisywana jest wartość 1. Komórka musi zostać następnie wypełniona wartością, którą będzie minimum z:

- wartości komórki leżącej bezpośrednio nad aktualną komórka zwiększonej o 1,
- wartości komórki leżącej bezpośrednio w lewo od aktualnej komórki zwiększonej o 1,
- wartości komórki leżącej bezpośrednio w lewą-górną stronę od aktualnej komórki + koszt.

Po wykonaniu wszystkich porównań odległością edycyjną będzie wartość w komórce [n, m] [11].

Pierwsze porównanie dotyczy liter *f* oraz *k*. Litery te są różne od siebie, więc koszt wynosi 1. Wyszukiwane jest następnie minimum z liczb: 1+1 (komórka nad, powiększona o 1), 1+1 (komórka z lewej strony, powiększona o 1) oraz 0+1 (wartość komórki po skosie lewa-górna + koszt). Zatem w komórce $[1, 1]$ wpisywana jest wartość 1.

Tablica 3.2: Po porównaniu liter *f* i *k*

		k	o	t	k	a
	0	1	2	3	4	5
f	1	1				
o	2					
k	3					
a	4					

Kolejne porównanie dotyczy liter *o* oraz *k*. Litery te są różne od siebie, więc koszt wynosi 1. Następnie wyszukiwane jest minimum z liczb: 1+1 (komórka nad, powiększona o 1), 2+1 (komórka z lewej strony, powiększona o 1) oraz 1+1 (wartość komórki po ukosie lewa-górna + koszt). W komórce $[1, 2]$ wpisywana jest wartość 2, co pokazuje tablica 3.3.

Tablica 3.3: Po porównaniu liter *o* i *k*

		k	o	t	k	a
	0	1	2	3	4	5
f	1	1				
o	2	2				
k	3					
a	4					

Tablica 3.4: Po dokonaniu wszystkich porównań

		k	o	t	k	a
	0	1	2	3	4	5
f	1	1	2	3	4	5
o	2	2	1	2	3	4
k	3	2	2	2	2	3
a	4	3	3	3	3	**2**

Po dokonaniu wszystkich porównań otrzymujemy tablicę 3.4.

W dolnym prawym narożniku znajduje się liczba 2 - ostateczny wynik. Odległość edycyjna wyrazów *foka* i *kotka* wynosi 2. Możliwe jest zatem przejście od jednego do drugiego wyrazu, wykonując dwa proste działania - zamieniając literę *f* na *k* oraz wstawiając literę *t*.

3.2. Metoda N-gramu oraz łańcuchów Markowa

Wnioskowanie statystyczne to dział statystyki zajmujący się problemami uogólniania wyników badania próby losowej na całą populację oraz szacowania błędów wynikających z takiego uogólnienia. Przykładem wykorzystania takiego wnioskowania może być rozwiązanie problemu przewidywania następnego słowa, dla danych wcześniejszych słów. Metoda N-gramu (część teorii łańcuchów Markowa) bazuje na wnioskowaniu statycznym. Metoda ta jest związana z modelem probabilistycznym, pozwalającym obliczyć prawdopodobieństwo występowania wyrazów w zdaniu [2].

Jeśli $W_{1:n}$ oznacza ciąg wyrazów $w_1w_2w_3...w_n$ to jaka jest wartość prawdopodobieństwa $W_{1:n}$? Aby obliczyć $P(w_{1:n})$ wykorzystuje się regułę łańcuchową, wtedy:

$$P(w_{1:n}) = P(w_{1:n-1})P(w_n|w_{1:n-1}) =$$

$$= P(w_{1:n-2})P(w_{n-1}|w_{1:n-2}) = ... = P(w_1)P(w_2|w_1)P(w_3|w_{1:2})...P(w_n|w_{1:n-1})$$

Możemy potraktować generację słów składających się na zdanie jako proces Markowa i przyjąć założenie Markowa - tylko N najbliższych słów ma wpływ na to jakie będzie w_n:

$$P(w_n|w_{1:n-1}) \approx P(w_n|w_{n-N+1:n-1})$$ [1].

We wzorze na wagę termu *N* oznacza ilość wszystkich dokumentów w kolekcji.

W metodzie tej możemy określić prawdopodobieństwo wystąpień:

- poszczególnych liter w wyrazie (wykorzystywane w popularnej grze "Shannon game"),
- poszczególnych wyrazów w zdaniu.

Oba te prawdopodobieństwa możemy wyznaczyć, ponieważ dla każdego języka naturalnego częstość występowania określonych liter (wyrazów) po sobie jest charakterystyczna. Niestety obliczanie prawdopodobieństwa wystąpień litery (słowa) nie jest zadaniem prostym, gdyż trzeba zwrócić uwagę na kontekst wypowiedzi. Szczególnie dla języka polskiego istnieje bardzo wiele odmian tego samego słowa. Przykładowo wyrazy *pojechałbym* oraz *pojechałem* pochodzą od tego samego wyrazu, lecz oznaczają co innego (pierwszy zamiar podróży, natomiast drugi stwierdzenie faktu odbycia już podróży).

W metodzie N-gramu wykorzystywana jest historia poprzedzających słów danego badanego wyrazu. Jednak biorąc pod uwagę jedną historię, nie można odgadnąć następnego słowa. Dane zdanie może mieć bowiem podobny początek, ale całkiem inny koniec [25]. Zatem do sprawdzania historii możemy wykorzystywać:

- bigramy - tylko poprzedzający wyraz,
- trigramy - dwa poprzedzające wyrazy,
- tetragramy - trzy poprzedzające wyrazy.

Zwiększając N otrzymuje się więcej informacji o kontekście, jednak istnieje ryzyko, że w danej klasie nie będzie żadnych danych lub wystąpi bardzo mała ich ilość, co może wpłynąć na wiarygodność estymacji.

Najczęściej metoda N-gramów (metoda łańcuchów Markowa) jest używana do analizy stylu pisania (wykrywanie plagiatów, autorstwa tekstów), analizy języka, tłumaczenia automatycznego oraz korekcji tekstów.

3.3. Analiza statyczna - prawo Zipfa

Kolejna metoda, wykorzystywana przy korekcji tekstu, związana jest z analizą statyczną tekstu i prawem Zipfa. Metoda ta bazuje na frekwencyjnym słowniku dla badanego języka naturalnego. Słownik ten zawiera słowa zgodne z zasadą ortografii wraz z liczbą (częstością) ich występowania w języku naturalnym.

Aby sformułować prawo Zipfa, należy najpierw określić czym jest częstość występowania słowa oraz czym jest pozycja rankingowa [13].

- **Częstość** to liczba wystąpień słowa w tekście (korpusie).
- **Lista rankingowa** słów to lista słów posortowana malejąco względem liczby wystąpień w tekście (korpusie).
- **Ranga słowa** jest to zatem liczba porządkowa słowa na liście rankingowej. Najczęściej występujące słowo ma rangę równą 1, drugie co do częstości ma rangę równą 2, trzecie ma rangę 3 itd.

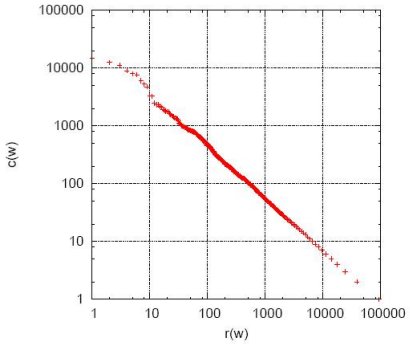
Prawo Zipfa można sformułować następująco:

> *Częstość występowania słów w języku naturalnym jest odwrotnie proporcjonalna do ich pozycji w rankingu.*

Wykaz pierwszych słów z Korpusu Słownika Frekwencyjnego Polszczyzny Współczesnej wg [13]:

Tablica 3.5: Korpus Słownika Frekwencyjnego Polszczyzny Współczesnej

słowo (**w**)	ranga (**r**)	częstość **c(w)**
w	1	14767
i	2	12473
się	3	11093
na	4	8750
nie	5	7878
z	6	7605
do	7	6004
to	8	5233
ze	9	4675
jest	10	3292
o	11	3264
jak	12	2407
a	13	2320



Rysunek 3.1: Zależność częstości występowania słowa od jego rangi

Wykres zależności częstości występowania słowa, od jego rangi zamieszczony został na rysunku 3.1.

Prawo Zipfa można interpretować następująco:

> *Jeżeli słowo w_1 występuje w rankingu na pozycji 10, to będzie ono występowało w języku naturalnym 5 krotnie częściej niż słowo w_2, które ma 50 pozycję w rankingu.*

Powyższą interpretację można zapisać za pomocą wzoru:

$$c_w \approx \frac{A}{r_w}, \text{ gdzie}$$

- c_w - częstość występowania słowa w,
- r_w - pozycja słowa w w rankingu,
- A - stała liczbowa.

Uzupełnieniem prawa Zipfa jest uściślenie Mandelbrota, polegające na określeniu dokładniejszego modelu relacji, w oparciu o funkcje wykładniczą. Dokładniej dla pewnych stałych B, d, P relacja między częstością, a rangą wynosi:

$$log\ c_w = logP - Blog(r_w + d)$$

Dla B=1 i d=0 formuła ta odpowiada w szczególności prawu Zipfa [1].

Prawo Zipfa zostało sformułowane w jego książce *"Human Behavior and the Principle of Least Effort"* - i jest czasem określane jako zasada Pareto w lingwistyce [1]. Rozkłady spełniające powyższe prawo występują nie tylko w lingwistyce. Można do nich zaliczyć:

- wspomniane wcześniej prawo Pareto (zasada 20/80),
- prawo Lotki,
- prawo Gibrata.

Niektóre poza lingwistyczne przykłady, zebrane przez Zipfa, są znacznie lepiej modelowane przez modele inne niż te, które uwzględniają powyższe prawo. Przykładem mogą być modele, modelowane lepiej przez rozkład log-normalny. W nich również można uwzględniać prawo Zipfa. Pytanie, jakie w tym miejscu często się pojawia, brzmi: *"Dlaczego prawo Zipfa obowiązuje dla tak wielu rozkładów ranga-częstość?"* [13]

Odpowiedzi należy szukać w wielorakich założeniach. Do najczęstszych można z pewnością zaliczyć: istnienie pewnego probabilistycznego modelu tekstu lub też założenie w którym przyjmuje się, że tekst jest efektem "pewnej optymalizacji kryteriów" je tworzących bądź też inne mówiące, że słowa w tekście można wyodrębnić za pomocą pewnego algorytmu. Należy również pamiętać, że niektóre z założeń, prowadzących do prawa Zipfa, mogą być błędne oraz, że istnieje wiele niezależnych, prawdziwych założeń.

Prawo Zipfa pojawia się w wielu modelach:

1. Model "małpy przy klawiaturze". W modelu tym założenia są następujące:
   - Tekst jest ciągiem liter i odstępów.
   - Tekst otrzymujemy wciskając losowo klawisze klawiatury.
   - Słowa w tekście to ciągi liter od odstępu do odstępu.

   W wyniku otrzymujemy listę rangową spełniającą prawo Zipfa-Mandelbrota.
2. Losowanie ze sprzężeniem zwrotnym. W modelu tym założenia są następujące:
   - Tekst jest ciągiem słów.
   - Tekst otrzymujemy losując kolejne słowa.

   W wyniku otrzymujemy rozkład częstości-częstości słów o rozkładzie Yule'a (uogólnienie prawa Lotki).
3. Minimalizacja średniej długości na bit. W modelu tym założenia są następujące:
   - Tekst jest efektem optymalizacji. (Tekst nie musi być produktem prostego procesu losowego.)
   - Przy ustalonej liczbie okazów słów w tekście rozkład częstości słów maksymalizuje iloraz $\frac{H}{C}$, gdzie: H — entropia słowa w tekście, C — średnia długość słowa w tekście.
   - Długość słowa o randze **r** jest proporcjonalna do **log r**.

   W wyniku otrzymujemy listę rangową spełniającą prawo Zipfa-Mandelbrota.

3.4. Model wektorowy

Aby móc przedstawić tekst w postaci wektora, musi zostać określony pewien słownik lub lista termów indeksujących. Term może tu oznaczać dowolne elementy tekstu, np. formy tekstowe, czy wyrazowe. Zakłada się, że poszczególne termy są nieskorelowane i tworzą bazę pewnej przestrzeni wektorowej. W modelu wektorowym dokumenty ze zbioru, w którym szukamy, są traktowane jako liniowe kombinacje termów. Jeżeli rozważymy współczynniki tej kombinacji, to otrzymamy wektor, który reprezentuje dokument w wybranej przestrzeni. Współrzędne tego wektora odpowiadają wadze, jaką nadaje się poszczególnym termom w dokumencie. Są one zazwyczaj proporcjonalne do liczby wystąpień termu w tekście dokumentu. Jeżeli dla któregoś termu wartość jest równa zero, oznacza to, że term nie występuje w dokumencie lub jest dla niego bez znaczenia [1].

Istnieje wiele sposobów obliczania współrzędnych wektora odpowiadającego konkretnemu dokumentowi. Wszystkie podstawowe metody bazują na wykorzystaniu trzech współczynników, jakimi można opisać dokument oraz jego związek z innymi dokumentami. Należą do nich:

- częstotliwość termu (ang. term frequency, tf) – ilość wystąpień termu w dokumencie,
- częstotliwość w dokumentach (ang. *document frequency, df*) – ilość dokumentów w kolekcji zawierających dany term,
- częstotliwość w kolekcji (ang. *collection frequency, cf*) – ilość wszystkich wystąpień termu w kolekcji dokumentów.

Częstotliwość termu odnosi się do określonego termu i dokumentu, natomiast pozostałe dwa parametry określa się dla poszczególnych termów. Należy zauważyć, że parametry *df* i *cf* mogą być obliczone tylko jeśli w ogóle istnieje kolekcja dokumentów.

Częstotliwość termu, czyli ilość jego wystąpień w dokumencie, wydaje się być dobrym wyznacznikiem ważności termu dla danego dokumentu. Im wyższa jest jej wartość, tym większa szansa, że term dobrze charakteryzuje zawartość dokumentu. Często zależność reprezentacji wektora od częstotliwości termu jest spłaszczana przez zastosowanie pierwiastka lub logarytmu. Drugi parametr, czyli ilość dokumentów w których term występuje, odzwierciedla popularność danego wyrazu w kolekcji oraz jego specyfikę lub powszedniość, a co za tym idzie wartość przy rozróżnianiu dokumentów. Dlatego często stosuje się go także, do obliczania wag termów w dokumencie, a tym samym współrzędnych wektorów. Najczęściej spotykaną formą częstotliwości w dokumentach jest jej postać odwrotna (ang. *inverse document frequency, idf*).

Kombinacja przedstawionych dwóch parametrów, częstotliwości termu i odwrotnej częstotliwości w dokumentach, oznaczana *tf.idf* jest najchętniej stosowanym schematem ważenia termów.

$$waga = tf * log\frac{N}{df}$$

We wzorze na wagę termu *N* oznacza ilość wszystkich dokumentów w kolekcji.

Podstawowym zagadnieniem wyszukiwania w modelu wektorowym jest określenie ”odległości” pomiędzy dokumentami. Zakłada się, że odległość ta jest miarą bliskości znaczeniowej dwóch tekstów, a więc tego czy opisują tą samą informację. Dzięki reprezentacji dokumentów jako wektorów, możliwe jest obliczanie odległości w oparciu o mocne podstawy matematyczne.

W modelu wektorowym podobieństwo dwóch wektorów określa się zazwyczaj poprzez miary związane z odległością tych wektorów w przestrzeni. Najczęściej stosowana jest miara cosinusowa. Opiera się ona na obliczeniu cosinusa kąta pomiędzy wektorami i może być przedstawiona w postaci wzoru:

$$cos(\overline{q}, \overline{d}) = \frac{\sum_{i=1}^{n} q_i d_i}{\sqrt{\sum_{i=1}^{n} q_i^2}\sqrt{\sum_{i=1}^{n} d_i^2}}$$

Miara cosinusowa to miara, w której wyższa wartość oznacza większe podobieństwo wektorów. Należy zauważyć, że jeżeli wektory są znormalizowane, to wyznaczenie miary cosinusowej sprowadza się do obliczenia iloczynu skalarnego wektorów. Miara cosinusowa jest popularna ze względu na takie własności jak: niezależność wartości od rozmiaru dokumentu, prostota i łatwość interpretacji.

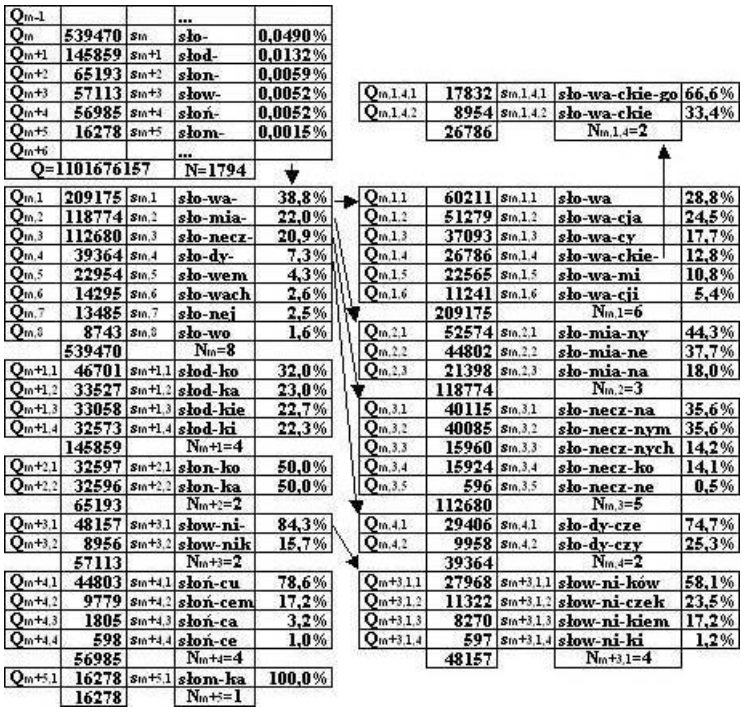
3.5. Metody automatycznej predykcji następstwa sylab

Kolejna metoda wiąże się szczególnie z automatycznym rozpoznawaniem mowy. Podstawowym wyzwaniem jest możliwość rozpoznania przez komputer w czasie rzeczywistym 100% słów jakie użytkownik wypowiada, uwzględniając m.in. występujące zakłócenia, zniekształcenia niektórych wypowiadanych słów bądź też sam akcent danej osoby. Niestety, z wyżej wymienionych przyczyn, automatyczne rozpoznawanie mowy z wykorzystaniem najlepszych metod waha się od 90 % do 95 % dla różnych języków naturalnych.

Koszty dzisiejszych systemów do automatycznego rozpoznawania mowy są bardzo duże. Zawierają one bowiem wiele złożonych mechanizmów do filtrowania oraz transformacji sygnału wejściowego, jakim jest głos (dźwięk) danej osoby. W procesie automatycznego rozpoznawania mowy wykorzystywane są mechanizmy, których zadaniem jest sprawdzanie uzyskanego wyrazu w specjalistycznym słowniku danego języka naturalnego oraz sprawdzana jest jego konstrukcja m.in. syntaktyczna oraz semantyczna. Dzięki tym metodom wybierane jest słowo dla którego uzyskano najbardziej prawdopodobny rezultat.

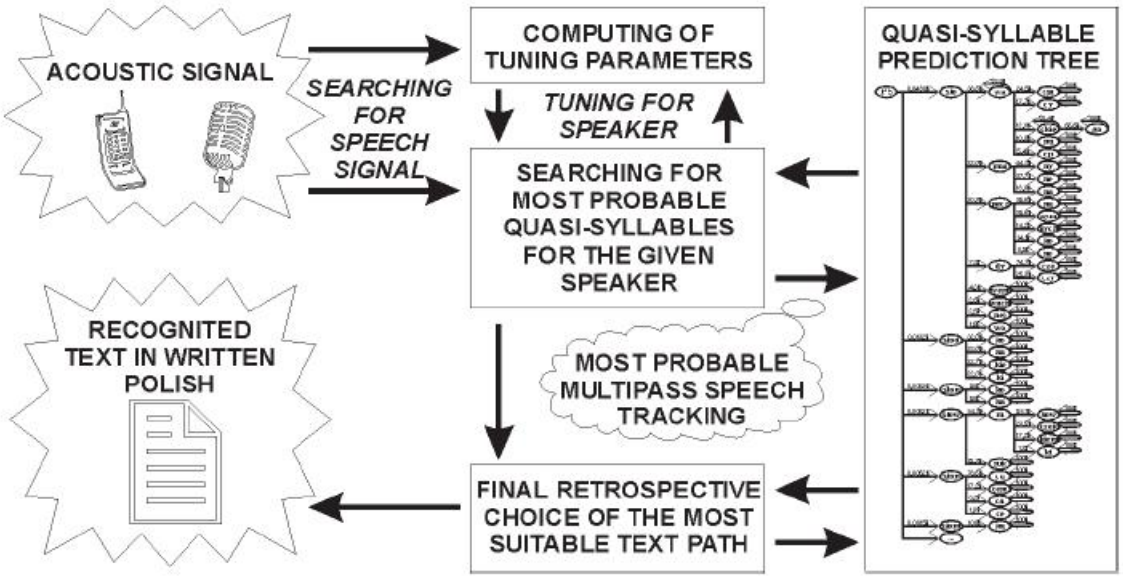
Nową metodę służącą automatycznemu rozpoznawaniu mowy zaproponował w swojej pracy dr Adrian Horzyk [3]. Metoda ta opiera się na analizie sylab w wyrazach oraz na możliwości przewidywania (predykcji) następnych sylab. Dzięki takiemu podejściu możliwe będzie dokończenie wyrazów, występujących w określonym kontekście zdania. W metodzie tej wykorzystywane jest posortowane drzewo, zawierające możliwe sylaby wyrazu w określonym kontekście, względem szans wystąpienia danej sylaby.

Przykładowy ”wycinek” takiego drzewa został zaprezentowany na rysunku 3.2.



Rysunek 3.2: Drzewo sylab

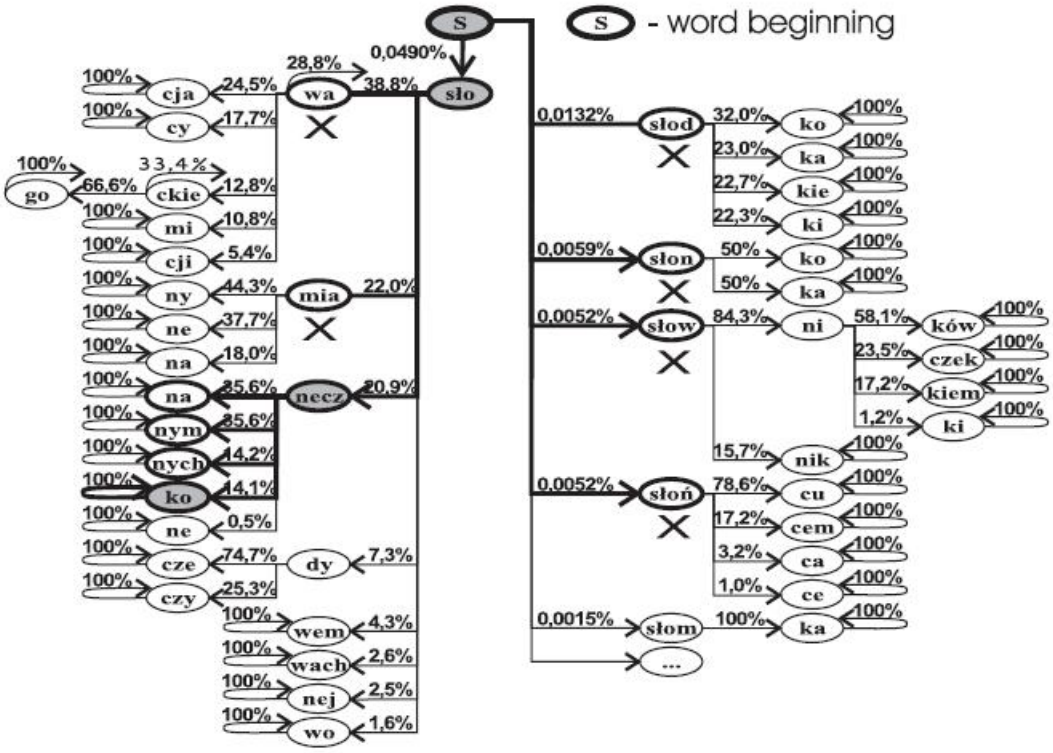
Schemat działania systemu automatycznego rozpoznawania mowy z uwzględnieniem metody analizy sylab został zaprezentowany na rysunku 3.3.



Rysunek 3.3: Schemat działania systemu automatycznego rozpoznawania mowy

Pierwsze próby rozpoznawania słów w czasie rzeczywistym dla języka polskiego, uwzględniając metodę rozpoznawania sylab, miały skuteczność rzędu 76-92%. Porównując ten wynik można odnieść wrażenie, że metoda ta nie należy do najlepszych. Na uwagę zasługuje jednak fakt, że w momencie testowania tej metody była ona ciągle jeszcze rozwijana. W przykładzie, jaki został opisany w pracy [3], badaniu zostało

poddane słowo ”*słoneczko*”. Na rysunku 3.4 został przedstawiony sposób działania metody z uwzględnieniem procentowego występowania określonych sylab.



Rysunek 3.4: Sposób działania metody predykcji sylab dla wyrazu *słoneczko*

3.6. Metoda odległości na klawiaturze

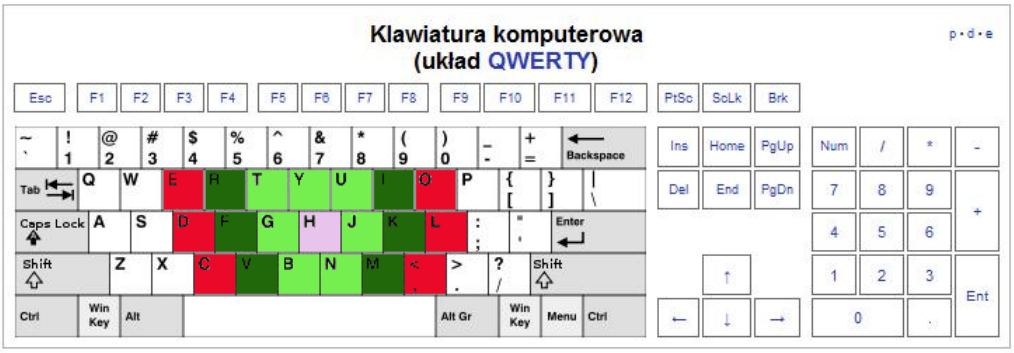
Kolejne metoda bazuje na układzie liter na klawiaturze. Najbardziej popularny układ liter na klawiaturze nosi nazwę *QWERTY*. Typ takiej klawiatury pokazano na rysunku 3.5.

Rysunek 3.5: Układ liter na klawiaturze typu *QWERTY*

Zauważono, że błędy we wprowadzanym tekście często wynikają z błędnych naciśnięć klawiszy. Użytkownik wprowadzając tekst wielokrotnie naciska nie ten klawisz, który by chciał, przez co generowany tekst zawiera wiele błędów. We wprowadzanych zdaniach pojawiają się wyrazy, których nie ma nawet w słowniku danego języka. Błędy te nie wynikają jednak z nieznajomości poprawnej konstrukcji wyrazu, lecz są determinowane zdolnościami manualnymi osoby wprowadzającej tekst.

Niech za przykład posłuży słowo ”*pojechałem*” w zdaniu. *Na wczasy pojechałem na początku lipca*. Użytkownik wprowadzając dane słowo, może zamiast naciśnięcia litery ”h”, nacisnąć literę stojącą obok niej, np. literę ”*g*”, powodując, że zamiast słowa *pojechałem* wystąpi wyraz *pojecgałem* - który nie występuje w języku polskim.

Metoda odległości liter na klawiaturze bada wprowadzane słowa, z wykorzystaniem układu klawiatury. W chwili napotkania błędnego wyrazu, próbuje dopasować wyraz najbardziej bliski danemu zamieniając litery, wykorzystując przy tym informacje jakie litery stoją najbliżej jakich. Literom stojącym w pewnej odległości od litery wyjściowej nadaje się specjalne wagi. Na rysunku 3.6 został przedstawiony podział wprowadzanych wag.



Rysunek 3.6: Literom nadaje się odpowiednie wagi w zależności od odległości

Literą wyjściową jest litera *h*.

- Najmniejszej odległości od litery *h* nadaje się najmniejszą wagę błędu. Litery z najmniejszą wagą zostały przedstawione za pomocą koloru jasno zielonego i są to litery *t, y, u, j, n, b, g*.
- Kolejnym literom o większej odległości od litery wyjściowej nadaje się większą wagę błędu. Litery te zostały przedstawione za pomocą koloru ciemno zielonego i są to litery *r, f, v, i, k, m*.
- Kolejnym literom o jeszcze większej odległości od litery wyjściowej nadaje się jeszcze większą wagę błędu. Algorytm badania odległości na klawiaturze może zawierać od dwóch do wielu wag błędu.

Tak więc, pomyłka w słowie *pojechałem*, polegająca na zamianie litery *h*, na literę *g* jest bardziej prawdopodobna, niż zamiana tej samej litery na literę *f* lub *d* - tym samym waga błędu jest mniejsza. Po przeprowadzonej analizie błędnego słowa, obliczana jest sumaryczna waga błędu dla proponowanych wyrazów i jako prawidłowy wybierany jest wyraz o najmniejszej wspomnianej wadze.

3.7. Metody proste

Istnieje szereg prostszych metod do analizy i korekty wprowadzanego tekstu. Metody te bazują jedynie na wprowadzonym tekście oraz na słowniku wszystkich wyrazów występujących w danym języku naturalnym. Zauważono bowiem, że błędami jakie najczęściej powstają są:

- zamiany liter w tekście (co zostało opisane w rozdziale 3.6),
- złączenia dwóch wyrazów w jeden (brak klawisza spacji między nimi)). Przykładem może być wprowadzenie zamiast słów *jadę do domu* słów *jadę dodomu*,
- zamiany dwóch liter stojących obok siebie kolejnością. Przykładem może być wprowadzenie zamiast słowa *kombinezon* słowa *kombniezon*.

4. Opis własnego rozwiązania

4.1. Wstęp

Jak już zostało wspomniane, postęp towarzyszy rozwojowi człowieka od początku jego istnienia i dotyczy niemal wszystkich sfer życia. Wiek XXI to przede wszystkim rozwój technologii, które mają ułatwiać codzienną pracę każdemu człowiekowi. W dzisiejszych czasach coraz bardziej popularne staja się rozwiązania mobilne. Ludzie coraz częściej zamieniają formę komunikacji z tradycyjnej na elektroniczną. Spowodowane jest to zawrotnym tempem rozwoju komputerów osobistych, Internetu, telefonii komórkowych. Już prawie nikt nie wyobraża sobie dnia bez wysłania e-maila, bądź napisania SMSa do znajomego. Niestety coraz mniej staramy się o formę przekazu treści. Wystarczy zwrócić uwagę na popełnianą ilość błędów ortograficznych, zamian liter w wyrazie lub ich skrótów.

Powstaje więc podstawowa trudność - jak zrozumieć i zinterpretować przekaz drugiej osoby? O ile człowiek, w dość prosty sposób, może zrozumieć wypowiedź drugiej osoby, tak zrozumienie tego samego tekstu przez komputer, okazuje się bardzo trudnym i złożonym zadaniem. Na świecie pojawiają się w związku z tym, liczne próby napisania algorytmów, które będą w stanie ”zrozumieć sens” wpisanego zdania i o ile zajdzie taka potrzeba, będą w stanie skorygować to zdanie, tak by było ono poprawne, zarówno ortograficznie, gramatycznie jak i stylistycznie. Bardziej znane algorytmy i mechanizmy, które już powstały zostały opisane w rozdziale 3.

Dotychczas opisywane metody bazowały na przekształceniu pojedynczych słów, w oderwaniu ich od kontekstu wypowiedzi. Promotor tej pracy - dr Adrian Horzyk - zaproponował nową metodę analizy i korekcji tekstu. Polega ona na badaniu w jakim kontekście oraz w jakiej formie fleksyjnej użyte są wyrazy w obrębie zdania i sprawdzenia czy możliwe jest, aby dane słowo występowało właśnie w takim kontekście wyrazów stojących bezpośrednio przed, jak i za nim. Wprowadzenie badania zdań, za pomocą takiego algorytmu, powinno w znaczny sposób poprawić korektę błędnych wyrazów w zdaniu. Dodatkowo sprawdzanie również przez algorytm, formy fleksyjnej słów, w znaczący sposób rozszerzy kontekst badania.

Celem tej pracy magisterskiej było zbudowanie specjalistycznego robota (pająka) internetowego, przeszukującego strony internetowe napisane w różnych językach skryptowych oraz formatach tekstowych, z zamiarem zbudowania maksymalnie rozbudowanego i ogólnego Grafu Przyzwyczajeń Lingwistycznych (LHG) dla wybranego języka, na podstawie analizy tych testów. Następnie, na podstawie grafu LHG, należało zbudować algorytm automatycznej kontekstowej korekty tekstów (z błędami) oraz wykonać porównania z dotychczas stosowanymi metodami. Praca ta składała się z kilku etapów. Możemy do nich zaliczyć:

1. Budowę specjalistycznego pająka internetowego.
2. Budowę grafu LHG.
3. Stworzenie i implementację algorytmu do automatycznej kontekstowej korekcji tekstu.
4. Porównanie wyników korekcji tekstu zaproponowanego algorytmu z istniejącymi już metodami korekcji innych programów.

W następnych podrozdziałach pragnę przybliżyć opis własnego rozwiązania każdego z wyżej wymienionych punktów.

4.2. Budowa specjalistycznego pająka internetowego

4.2.1. Wprowadzenie

Pierwszy etap obejmował zbudowanie specjalistycznego robota (pająka) internetowego przeszukującego strony internetowe napisane w różnych językach skryptowych oraz formatach tekstowych, celem zbudowania maksymalnie rozbudowanego i ogólnego Grafu Przyzwyczajeń Lingwistycznych (LHG) dla wybranego języka, na podstawie analizy tych testów.

Większość wyszukiwarek dostępnych w internecie korzysta z tzw. robotów (pająków) internetowych, aby zebrać informację o określonych treściach na stronach internetowych. Pająki te poszukują nowych informacji na dostępnych dla wyszukiwarki stronach, śledzą pojawianie się nowych stron, jak również monitorują zmiany istniejących już stron.

Schemat działania takich pająków jest bardzo prosty:

- wchodzą one na stronę internetową,
- pobierają informację o danych zamieszczonych na danej stronie internetowej,
- wyszukują wszystkie linki do kolejnych stron,
- przechodzą do kolejnej strony internetowej.

Twórca serwisu internetowego może ”kontrolować” jaka zawartość strony będzie dostępna dla robotów internetowych, a jakie strony nie będą dostępne do indeksacji. Można to uczynić na dwa sposoby:

1. Dodać do strony głównej plik *robots.txt*. Roboty indeksujące sprawdzają ten specjalny plik, który powinien znajdować się w katalogu głównym serwera. *Robots.txt* jest plikiem tekstowym bez tagów HTML. Używa on specjalnego protokołu - *Robots Exclusion Protocol*, który pozwala administratorowi zdefiniować, które katalogi lub pliki, na jego serwerze, nie zostaną odwiedzone przez określone roboty [24]. Mechanizm ”ukrywania” stron jest często spotykany. W momencie gdy administrator tworzy nowy serwis, artykuł, stronę dla klienta nie chce, aby została ona już zaindeksowana przez robota. Kolejnym przykładem jest ograniczenie ruchu na wybranych stronach internetowych. Zdarza się, że pająki się ”zapętlają” przechodząc w kółko pomiędzy kilkoma stronami i generując tym samym niepotrzebny ruch.
2. Dodać do nagłówka strony odpowiednie informacje, które są zawarte w tzw. *meta tagach*. Pełnią one taką samą funkcję jak plik *robots.txt*. Informują pająka internetowego czy może wejść na daną stronę WWW. Dodatkowo w *meta tagach* można określić, w jaki sposób robot ma poruszać się po stronie internetowej (z jakiego linku na jaki ma robot przejść) [24].

W internecie istnieje wiele gotowych rozwiązań i implementacji pająków internetowych. Kilka przykładów gotowych implementacji zostało zaprezentowanych poniżej:

- Acme.Spider

  Strona domowa projektu *http://www.acme.com/java/software/Acme.Spider.html* Pająk napisany w języku Java przez ACME Labs. Poprzez dostarczane bogate API istnieje możliwość dostosowania go do indywidualnych potrzeb [20].

- Checkbot

  Strona domowa projektu *http://degraaff.org/checkbot/* Checkbot jest narzędziem do sprawdzania poprawności linków na stronach internetowych. Checkbot potrafi również sprawdzać pojedyncze dokumenty lub zestawy z różnych serwerów WWW. Dodatkowo aplikacja na końcu tworzy raport podsumowujący jej pracę [21].

- FDSE

  FDSE to silnik dla wyszukiwarek firmy Fluid Dynamics Software Corporation. Potrafi odwiedzać zarówno strony lokalne jak i zdalne, na różnych serwerach. Jego wersje freeware oraz shareware są dostępne dla języka Perl [22].

- JoBo

  JoBo to prosty program do pobierania i zapisywania w całości stron internetowych na lokalnym dysku komputera. Program ten jest więc pająkiem internetowym. Podczas działania potrafi używać cookies do podtrzymywania sesji przeglądarki. Program został napisany w języku Java, więc ma możliwość pobrania źródeł i modyfikacji programu do własnych potrzeb [23].

Wykaz blisko 300 różnych implementacji pająków internetowych, wraz z krótką charakterystyką każdego z nich, można znaleźć w bazie danych na stronie internetowej Spiders.pl *http://www.spiders.pl/baza-browse.php*

W pracy tej został zaimplementowany własny pająk, z kilku względów:

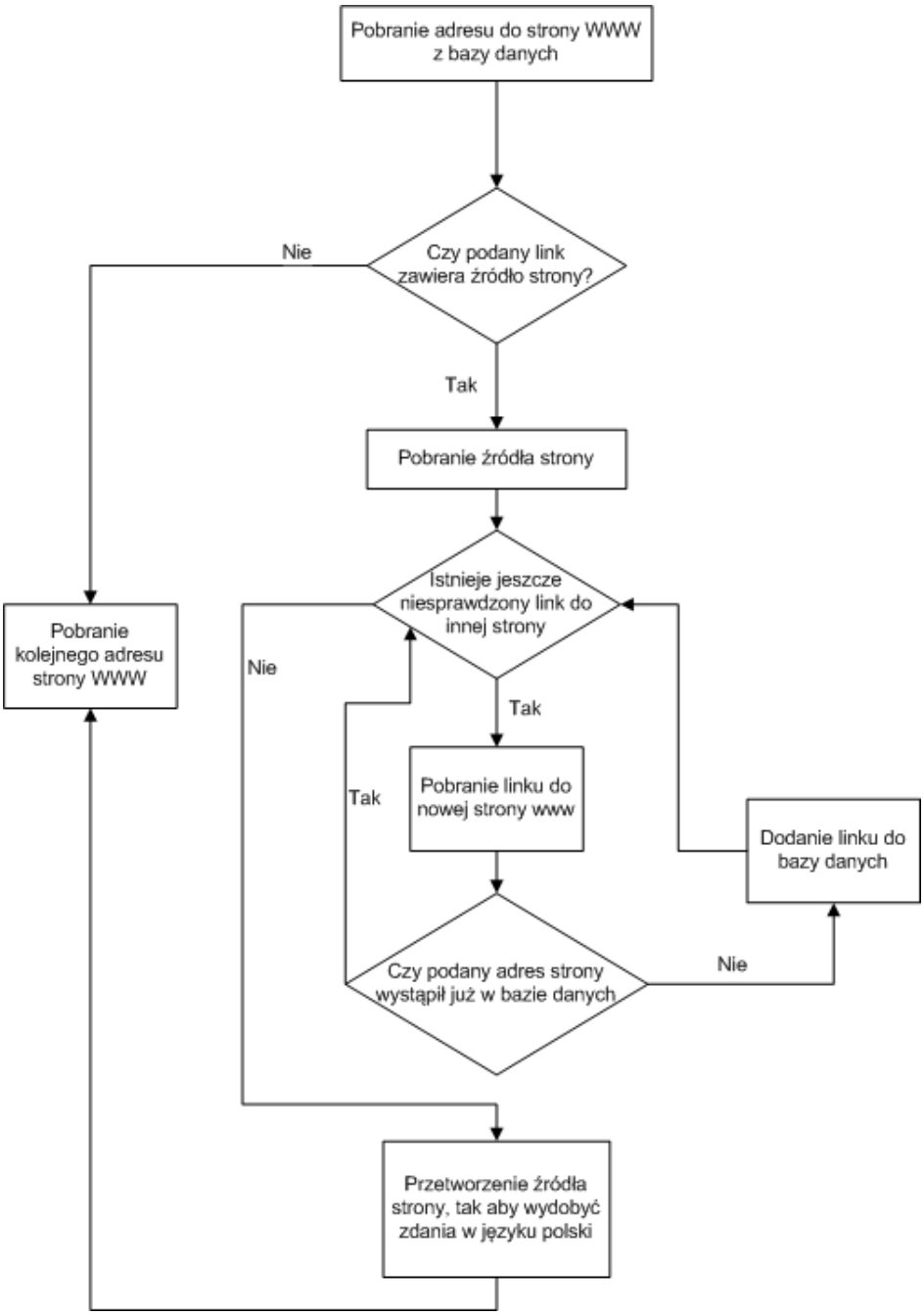
- Będzie znany dokładnie sposób zachowania pająka na stronie oraz sposób przechodzenia między nimi.
- Będzie on w stanie wejść na każdą stronę internetową (nie będą wykorzystywane danych z pliku *robots.txt* lub z meta danych).
- Będzie możliwość zaimplementowania własnych mechanizmów pobierania tekstów ze stron internetowych.
- Będzie on mógł wchodzić tylko na strony internetowe w języku polskim.
- Będzie możliwość uzyskania informacji o odwiedzonych stronach oraz statystykę: ile nowych stron zostało znalezionych oraz ile wyrazów w języku polskim zostało znalezionych.

4.2.2. Własne rozwiązanie

Zasada działania zaimplementowanego własnego pająka internetowego, jest bardzo podobna do rozwiązań już spotykanych. Ogólny schemat postępowania został wypunktowany poniżej oraz przedstawiony na rysunku 4.1:

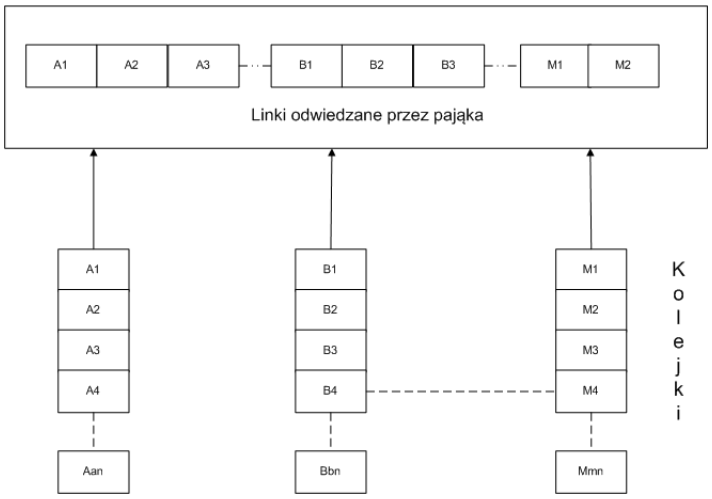
- pająk wchodzi na stronę internetową, której adres znajduje się w bazie danych,
- pobierane jest źródło strony,
- z źródła strony pobierane są wszystkie odnośniki do kolejnych stron internetowych,
- z źródła strony pobierany jest kontekst słów występujący w zdaniu w języku polskim na stronie,
- kolejne odnośniki do stron jak i kontekst słów zapisywany jest w bazie danych,
- pająk przechodzi do kolejnej strony WWW.

Pierwszą czynnością, jaką wykonuje pająk, to pobranie z bazy danych adresu WWW strony, która zostanie poddana analizie. Po pobraniu linku z bazy, pająk pobiera całą zawartość strony (jej źródło), w celu dalszej obróbki. Następie wyszukiwane są wszystkie adresy stron WWW, do których można przejść z danej strony w dalszej kolejności. Każdy adres sprawdzany jest w bazie danych czy nie został on już wcześniej dodany przez pająka. Jeśli adres nie znajdował się wcześniej w bazie odnośników, pająk dodaje ten adres do lisy adresów stron, na jakie ma wejść, jak również do spisu wszystkich adresów. Algorytm dopisywania stron internetowych do listy stron do odwiedzenia działa w sposób przedstawionym na rysunku 4.2.



Rysunek 4.1: Schemat działania własnego pająka internetowego

Kolejną czynnością, jaką wykonuje pająk jest sprawdzenie, czy strona internetowa zawiera informacje w języku polskim. Zostało postanowione, że sprawdzenie to będzie polegać na wyszukaniu czy w źródle strony internetowej występuje litera *ą*. Stwierdzono bowiem, że jeśli strona zawiera polski znak to przynajmniej jej część musi być napisana w języku polskim. Jeśli na stronie nie występuje litera *ą*, pająk usuwa ten adres ze stron do odwiedzenia i przechodzi do kolejnej strony internetowej - pobiera z bazy następny link do strony WWW. Ponieważ do zbudowania jak największego grafu LHG potrzebne jest odwiedzenie tysięcy stron i pobranie z nich informacji, na temat kontekstu wypowiedzi, to praktyczne jest, że jeśli strona, na której aktualnie znajduje się pająk internetowy, nie zawiera litery *ą* zostaje odrzucona (choć może w całości być w języku polskim). Linki do kolejnych stron WWW pobierane są natomiast ze wszystkich stron (zarówno tych w języku polskim jak i zagranicznych). Postępuje tak, ponieważ strona zagraniczna może zawierać link do strony polskiej, a rozpoznawanie czy informacje na stronie internetowej są w języku polskim również zostało zaimplementowane.



Rysunek 4.2: Schemat poruszania się pająka internetowego po stronach WWW

Gdy pająk stwierdzi, że strona jest w języku polskim, przystępuje do ekstrakcji jak największej ilości zdań, aby następnie móc analizować kontekst wypowiedzi poszczególnych słów. Dla każdego ze zdań pobierany jest do bazy danych jego kontekst słowny. Działanie algorytmu polega na zapisaniu do bazy, każdej trójki występujących po sobie słów w zdaniu. Jeśli zdanie składa się z dwóch lub jednego wyrazu, słowa takie są pomijane. Przykładowo dla zdania *Ja chciałem iść dzisiaj do kina.* pająk do bazy zapisze następujące trójki wyrazów:

- Ja chciałem iść,
- chciałem iść dzisiaj,
- iść dzisiaj do,
- dzisiaj do kina.

Dla każdej takiej trójki (wyraz poprzedzający, wyraz rozważany, wyraz następny) można analizować jej najbliższy kontekst słowny z uwzględnieniem konkretnych form fleksyjnych wszystkich słów.

- Dla słowa poprzedzającego kontekstem są dwa wyrazy po nim występujące.
- Dla słowa aktualnego kontekstem jest słowo poprzedzające je oraz słowo następne.
- Dla słowa następnego kontekstem są dwa wyrazy przed nim występujące.

Ostatnią czynnością jest dopisanie do listy stron odwiedzonych aktualnej strony internetowej oraz wykasowanie jej z listy stron do odwiedzenia i przeanalizowania. Następnie robot pobiera kolejny adres strony WWW, znajdującej się na liście stron do odwiedzenia i rozpoczyna cały etap jej analizy od początku, stopniowo rozbudowując bazę trójek.

4.2.3. Prezentacja

Postęp pracy pająka internetowego (indeksacji stron, dodawania nowych linków do bazy oraz słów-trójek) można śledzić w oknie konsoli. Wynik pracy został zaprezentowany na rysunku 4.3.

Zostało stwierdzone, że nie są potrzebne szczegółowe dane pracy robota, takie jak wykaz wszystkich linków oraz słów jakie zostały dodane do bazy danych. Postanowiono informować użytkownika, jedynie o najważniejszych danych podczas odwiedzania każdej ze stron. Są to:

- adres strony internetowej, jaka została odwiedzona przez pająka,
- godzina rozpoczęcia indeksacji danej strony,
- liczba nowych linków do kolejnych stron (które nie zostały jeszcze odwiedzone),

```
0 heartBip! Tue May 19 08:37:31 CEST 2009 %stat% url: http://www.gazeta.pl/0,0.html
pkt - start: 3116 finish: 111203 href size: 499 wordBaseModel: 977

1 heartBip! Tue May 19 08:38:20 CEST 2009 %stat% url: http://www.gazetapolska.pl/
pkt - start: 445 finish: 10152 href size: 8 wordBaseModel: 267

2 heartBip! Tue May 19 08:38:28 CEST 2009 %stat% url: http://wyborcza.pl/0,0.html
pkt - start: 3130 finish: 153056 href size: 676 wordBaseModel: 2176

3 heartBip! Tue May 19 08:39:53 CEST 2009 %stat% url: http://www.ciacha.net/ciacha/0,0.html
pkt - start: 3483 finish: 74390 href size: 176 wordBaseModel: 539

4 heartBip! Tue May 19 08:40:14 CEST 2009 %stat% url: http://gospodarka.gazeta.pl/pieniadze/0,0.html
pkt - start: 3095 finish: 95856 href size: 388 wordBaseModel: 3209

5 heartBip! Tue May 19 08:42:06 CEST 2009 %stat% url: http://gospodarka.gazeta.pl/firma/0,0.html
pkt - start: 3103 finish: 109365 href size: 413 wordBaseModel: 4012
```

Rysunek 4.3: Postęp prac pająka internetowego

- liczba słów-trójek, jakie zostały znalezione i dodane do bazy danych,
- dane systemowe (takie jak zajętość pamięci, ilość wolnego miejsca).

4.3. Budowana grafu LHG

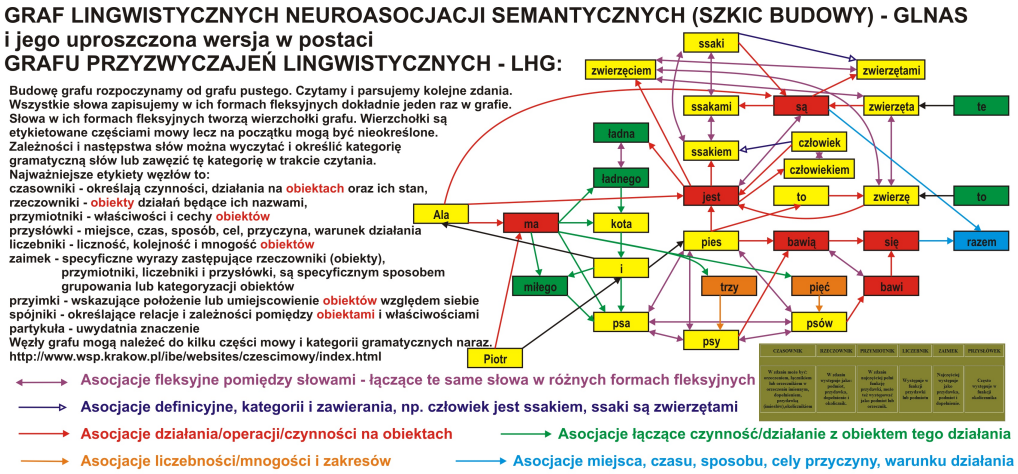
4.3.1. Wprowadzenie

Kolejnym etapem niniejszej pracy było zbudowanie maksymalnie rozbudowanego i ogólnego Grafu Przyzwyczajeń Lingwistycznych (LHG) dla języka polskiego, na podstawie uzyskanego kontekstu słów. Wynikiem działania pająka internetowego było m.in. zbudowanie bazy danych kilkunastu milionów trójek-wyrazów, które tworzą pewien kontekst wypowiedzi dla języka polskiego.

Grafem nazywamy strukturę $G = (V, E)$, która składa się z węzłów (wierzchołków oznaczanych przez V) wzajemnie połączonych z sobą za pomocą krawędzi (oznaczonych przez E). Grafy możemy podzielić na skierowane i nieskierowane.

Grafem skierowanym (digrafem od angielskich słów *directed graph*) będzie zatem uporządkowana struktura $G = (V, E)$, składająca się z wierzchołków i krawędzi, które są uporządkowaną parą wierzchołków. Jeżeli krawędź $e = (v_i, v_j)$ jest łukiem, to mówimy, że e jest łukiem wychodzącym z wierzchołka v_i i wchodzącym do wierzchołka v_j. v_i nazywamy początkiem łuku, a v_j jego końcem. Ruch po grafie skierowanym może odbywać się jedynie w kierunku wskazywanym przez krawędzie [6].

Grafy LHG mają za zadanie pomóc inteligentnym lingwistycznym cyberbotom zrozumieć wypowiedzi ludzi oraz zbudować własne wypowiedzi odwzorowujące sposób konstrukcji zdań, fraz (szczególnie idiomatycznych) oraz doboru słów charakterystyczny dla wypowiedzi ludzi. Pomagają one rozpoznać kontekst w wymiarze fleksyjno-częstotliwościowym, który jest dzięki temu łatwy do wykorzystania. Graf LHG jest częścią (uproszczoną wersją) większego projektu dr Adriana Horzyka - tworzenia grafów lingwistycznych neuroasocjacji semantycznych GLAS, który ma możliwość utrzymywania kontekstu wypowiedzi. Dodatkowo w zaproponowanym grafie neuroasocjacji semantycznych (GLAS), każdy węzeł będzie specyficznym neuronem pobudzającym inne neurony takiego grafu.



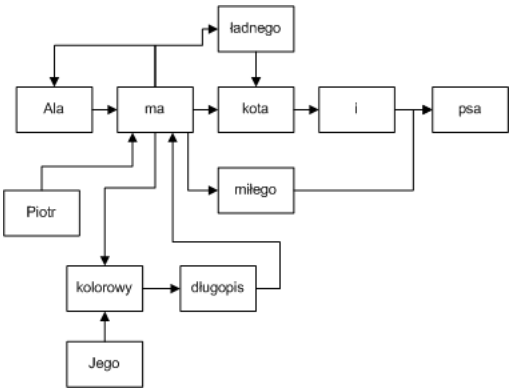
Rysunek 4.4: Przykład grafu GLAS z opisem

4.3.2. Własne rozwiązanie

Graf LHG jest grafem skierowanym, w którym każde słowo reprezentowane jest przez wierzchołek, a krawędź symbolizuje odpowiednie, możliwe przejścia między słowami, w różnych kontekstach wypowiedzi. Przykład grafu LHG został przedstawiony na rysunku 4.5. Należy zauważyć, że jego konstrukcja jest bardzo podobna do grafu GLAS. Zaimplementowany graf zawiera jednak jedynie informacje o kontekście wypowiedzi, czyli o określonym porządku słów mogącym wystąpić w zdaniach.

Do zbudowania grafu przedstawionego na rysunku 4.5 użyto następujących zdań:

- Ala ma kota i psa.



Rysunek 4.5: Graf LHG

- Ala ma miłego psa.
- Piotr ma ładnego kota.
- Piotr ma kolorowy długopis.
- Jego kolorowy długopis ma Ala.

Jak można zauważyć, wszystkich słów w podanych powyżej zdaniach jest 22. Wierzchołków natomiast w skonstruowanym grafie jest zaledwie 11. Został w nim jednak odtworzony cały kontekst wypowiedzi. Idąc ”po strzałkach”, nie tylko można zbudować przykładowe zdania, ale również utworzyć nowe, których wcześniej nie było, a zazwyczaj będą poprawne językowo. Przykładem takich zdań są: *Piotr ma kota, Piotr ma kota i psa, Ala ma kolorowy długopis*. Z taką agregacją kontekstu wypowiedzi wiąże się również poważna wada. Budując graf dla poprawnych językowo zdań, można byłoby odtworzyć zdania niepoprawne. Dla tego samego grafu zdaniem możliwym do odtworzenia, a niepoprawnym jest: *Jego kolorowy długopis ma miłego psa.* Dodatkowo można uzyskać efekt zapętlenia. Przykładem może być początek wypowiedzi: *Jego kolorowy długopis ma kolorowy długopis ma kolorowy długopis ma...*

W stworzonej aplikacji dostępny jest moduł do konstrukcji grafu LHG. Tworzony jest w nim graf na podstawie podanego kontekstu słownego. Algorytm sprawdza w bazie czy dla początkowych dwóch słów możliwe są ich następniki. Jeśli tak, zostają one zaprezentowane. Użytkownik ma możliwość wyboru następnika, a po jego wyborze pokazaniu dla niego kolejnych następników. Dzięki temu możliwa jest konstrukcja poprawnego zdania w języku polskim, utrzymując poprawny kontekst trzech ostatnich wyrazów.

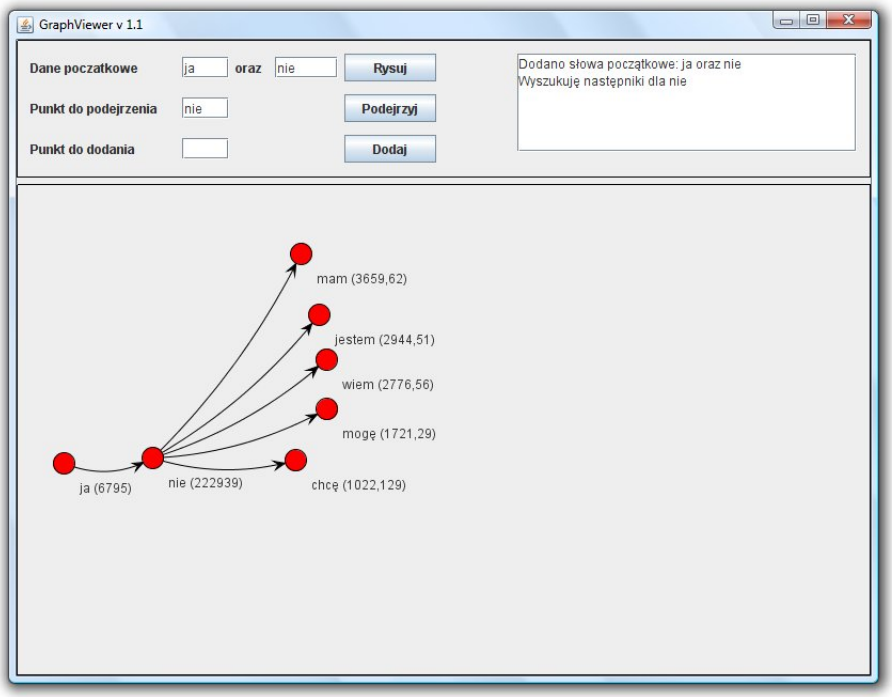
Proces konstrukcji grafu dla zdania *Ja nie chcę im pomóc* został zaprezentowany na rysunkach 4.6, 4.7, 4.8, 4.9,

Jak można zauważyć przy budowie grafu można było pójść innymi ścieżkami, a przez to uzyskać inne zdanie. Dla przykładu kilka początków zdań, które można było utworzyć:

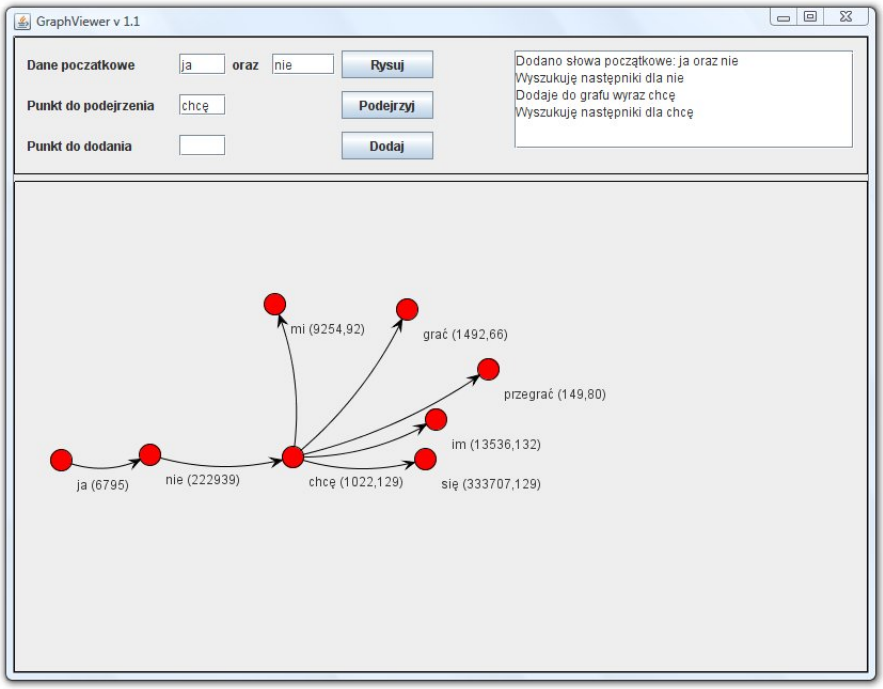
- Ja nie jestem...
- Ja nie wiem...
- Ja nie chcę przegrać...
- Ja nie chcę się...

Na rysunku 4.8 zaprezentowane są proponowane następniki dla początku zdania *Ja nie chcę im*. Są nimi:

- dać,
- się,
- odebrać,



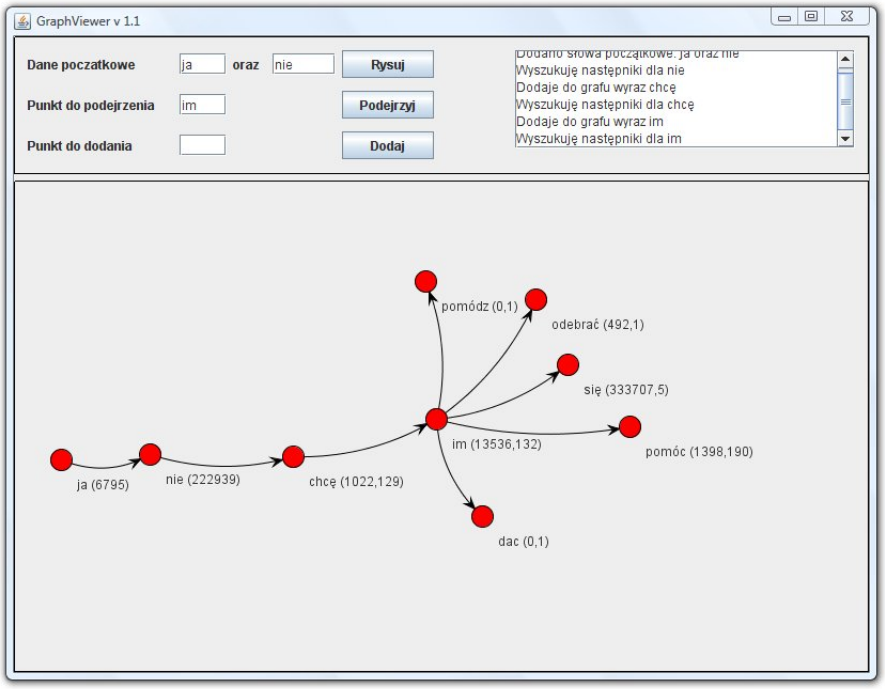
Rysunek 4.6: Konstrukcja grafu LHG - możliwe następniki dla dwóch początkowych słów



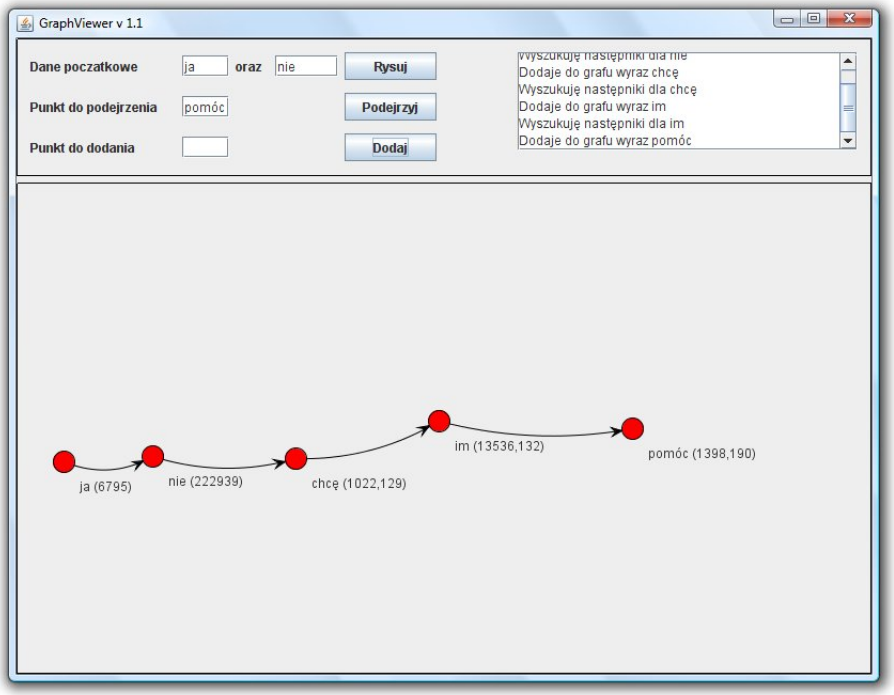
Rysunek 4.7: Konstrukcja grafu LHG - dodanie trzeciego słowa i podgląd jego następników

- pomóc,
- pomódz.

Jedno z zaprezentowanych słów - *pomódz* nie występuje w Słowniku języka polskiego i jest ono niepoprawne. Wszystkie proponowane przez algorytm wyrazy pobierane są z bazy, którą utworzył



Rysunek 4.8: Konstrukcja grafu LHG - dodanie czwartego słowa i podgląd jego następników



Rysunek 4.9: Konstrukcja grafu LHG - dodanie piątego słowa

wcześniej pająk internetowy. Robot sprawdzał wszystkie strony internetowe, w poszukiwaniu zdań w języku polskim. Możliwym jest więc zapisanie w bazie słów zdań, które są niepoprawne. Mała frekwencja pojawienia się słów, takich jak *pomódz* może świadczyć jednak, że są one niepoprawne. W analizowanym przykładzie słowo pomóc pojawiło się w Słowniku frekwencyjnym [7] aż 1398 razy (pierwsza cyfra obok słowa na zaprezentowanym rysunku) oraz 190 razy w danym kontekście (druga cyfra obok słowa na zaprezentowanym rysunku). Wyraz *pomódz* nie pojawił się we wspomnianym Słowniku (pierwsza cyfra obok

słowa) oraz wystąpił jedynie raz w zadanym kontekście (druga cyfra obok słowa).

Budowa grafu była kolejnym etapem realizacji aplikacji. Na jego podstawie został opracowany i zaimplementowany algorytm kontekstowej korekty tekstów, który szczegółowo został opisany w rozdziale 4.4.

4.4. Implementacja własnego algorytmu do kontekstowej korekcji tekstu

Najważniejszą częścią realizacji pracy, było zaproponowanie, implementacja oraz wykorzystanie innowacyjnego algorytmu służącego kontekstowej korekcji tekstów. W obecnych czasach nie jest dostępny żaden edytor, który potrafiłby poprawiać wprowadzony tekst na podstawie jego kontekstu dla języka polskiego. Najczęściej wykorzystywanymi edytorami tekstu dla przeciętnego użytkownika jest Microsoft Word, który wchodzi w skład pakietu Microsoft Office oraz jego darmowy odpowiednik Writer, wchodzący w skład pakietu OpenOffice.org. W obydwu tych programach zostały co prawda zaimplementowane mechanizmy korekcji tekstu, lecz dotyczą one generalnie korekcji pojedynczego wyrazu, nie uwzględniając kontekstu, w jakim został on użyty. Szczegółowe porównanie mechanizmów korekcji tekstu tych programów z opracowanym algorytmem zostało zaprezentowane w rozdziale 5.

4.4.1. Algorytm

W zbudowanej przez pająka internetowego bazie danych, znajdują się miliony rekordów z trójkami słów, które posłużą do zbudowania grafu LHG, który dzięki temu może dla dowolnych wyrazów określić kontekst słowny i fleksyjny poprzedzających i następnych słów. Dla każdego słowa istnieje możliwość wyróżnienia trzech typów możliwych korekcji. Przykładowo dla tekstu *ja chcę jeść* możliwe jest wyróżnienie korekcji:

1. Badane słowo jest poprzednikiem dwóch pozostałych występujących w bazie trójek oraz grafie LHG. W tym przypadku badamy czy słowo *ja* może wystąpić jako poprzednik dla dwóch słów *chcę jeść*.
2. Badane słowo znajduje się pomiędzy dwoma znanymi wyrazami występującymi w bazie trójek oraz grafie LHG. W tym przypadku badamy czy słowo *chcę* może wystąpić w kontekście określonym przez słowa *ja* oraz *jeść*.
3. Badane słowo znajduje się jako następnik dwóch pozostałych występujących w bazie trójek oraz grafie LHG. W tym przypadku badamy czy słowo *jeść* może wystąpić jako następnik słów *ja chcę*.

W trakcie prowadzonych badań stwierdzono, że najlepsza weryfikacja oraz poprawa słów będzie wówczas, gdy zostanie wykorzystany trzeci sposób badania, tj. będę sprawdzane, czy badane słowo występuje jako następnik dwóch poprzedzających go wyrazów. Analizując budowę Grafu Przyzwyczajeń Lingwistycznych zauważono, że znacznie częściej można znaleźć badane słowo jako następnik niż, np. jako poprzednik określonych wyrazów w zdaniu. W konstrukcji wyrażeń o wiele częściej można znaleźć takie same początki wielu zdań, z różnymi ich zakończeniami, niż takie same zakończenia z różnymi początkami. Budowę algorytmu kontekstowej korekty tekstów, oparto zatem o analizę następników. Może się również zdarzyć, że we wprowadzonym tekście pierwsze słowa są błędne. Do analizy takiego przypadku wykorzystano sprawdzanie kontekstu wstecznego. Opiera się ono na badaniu, czy możliwe jest wystąpienie danych wyrazów jako słów w środku kontekstu oraz jako poprzedników dwóch wyrazów. Dodatkowo badanie kontekstu wstecznego wykorzystano podczas korekcji błędnych wyrazów. Dla każdego słowa, uznanego przez algorytm za błędne, wyszukiwane są wyrazy występujące w różnych kontekstach, mogące go zamienić.

Algorytm analizuje kontekst wypowiedzi w obrębie danego zdania, stąd pierwszym etapem jest przekształcenie wprowadzonego tekstu na zdania, gdyż wprowadzony tekst może składać się z wielu zdań. Następnie dla każdego ze zdań dokonywana jest korekcja tekstu, za pomocą zaproponowanego algorytmu.

Pierwszym etapem badania kontekstu wypowiedzi, jest sprawdzenie czy początkowe słowa w zdaniu są prawidłowo wprowadzone. W tym celu wykorzystywane jest sprawdzanie wspominanego kontekstu wstecznego. Algorytm analizuje pierwsze słowa zdania, czy mogą wystąpić jako poprzedniki lub aktualne słowa dla danego kontekstu. Jeśli nie występują, wyszukiwane są odpowiednie słowa, które mogą zastąpić błędnie wprowadzone i dokonywana jest korekcja.

Następnym etapem badania kontekstu wypowiedzi, jest sprawdzenie czy dla dwójki początkowych słów, które zostały w pierwszym etapie sprawdzone i ew. poprawione, występuje wpisany przez użytkownika następnik. Jeśli następnik istnieje, badana jest kolejna trójka słów. Jeśli natomiast następnik nie występuje, badane jest otoczenie danego słowa i szukane są propozycje słów, jakie mogą zastąpić dane słowo (które zostało uznane przez algorytm za błędne, dla danego kontekstu wypowiedzi). W tym celu badane są słowa,

jakie mogą wystąpić jako następniki, środkowe i poprzedniki dla wprowadzonego kontekstu słownego. Z bazy danych oprócz słowa pobierana jest także jego częstość występowania w danym kontekście zdania.

Mechanizm korekcji tekstu utworzony został w oparciu o:

- częstość wystąpienia proponowanych słów,
- informację czy badane słowo występuje jako możliwy następniki, słowo środkowe i poprzedniki, czy jedynie jako, np. następniki i słowo środkowe,
- odległość edycyjną proponowanych słów do poprawy,
- długość proponowanego słowa do zamiany.

Algorytm sprawdza występowanie możliwych następników dla dwóch pierwszych słów i zapisuje je na liście proponowanych słów. Następnie wyszukuje wszystkie możliwe wyrazy jako słowa aktualne. Gdy zaproponowane rozważane słowo nie znajduje się na liście propozycji, zostaje do niej dodane z informacją o częstości występowania. Gdy słowo znajduje się na wspomnianej liście, program do już istniejącej częstości występowania słowa jako następnika, dodaje informacje o częstości występowania słowa jako aktualnego. Podobna zasada dotyczy badania słowa jako poprzednika dwóch następujących po sobie słów.

Gdy została już utworzona lista z wszystkimi możliwymi proponowanymi słowami, algorytm sprawdza czy istnieją słowa, które występują jako poprzedniki, słowa aktualne oraz następniki dla danego kontekstu. Jeśli nie występują, sprawdzane jest czy istnieją słowa w dwóch z trzech wspomnianych wcześniej przypadkach. Jeśli nie, wybierana jest lista z wszystkimi dostępnymi słowami.

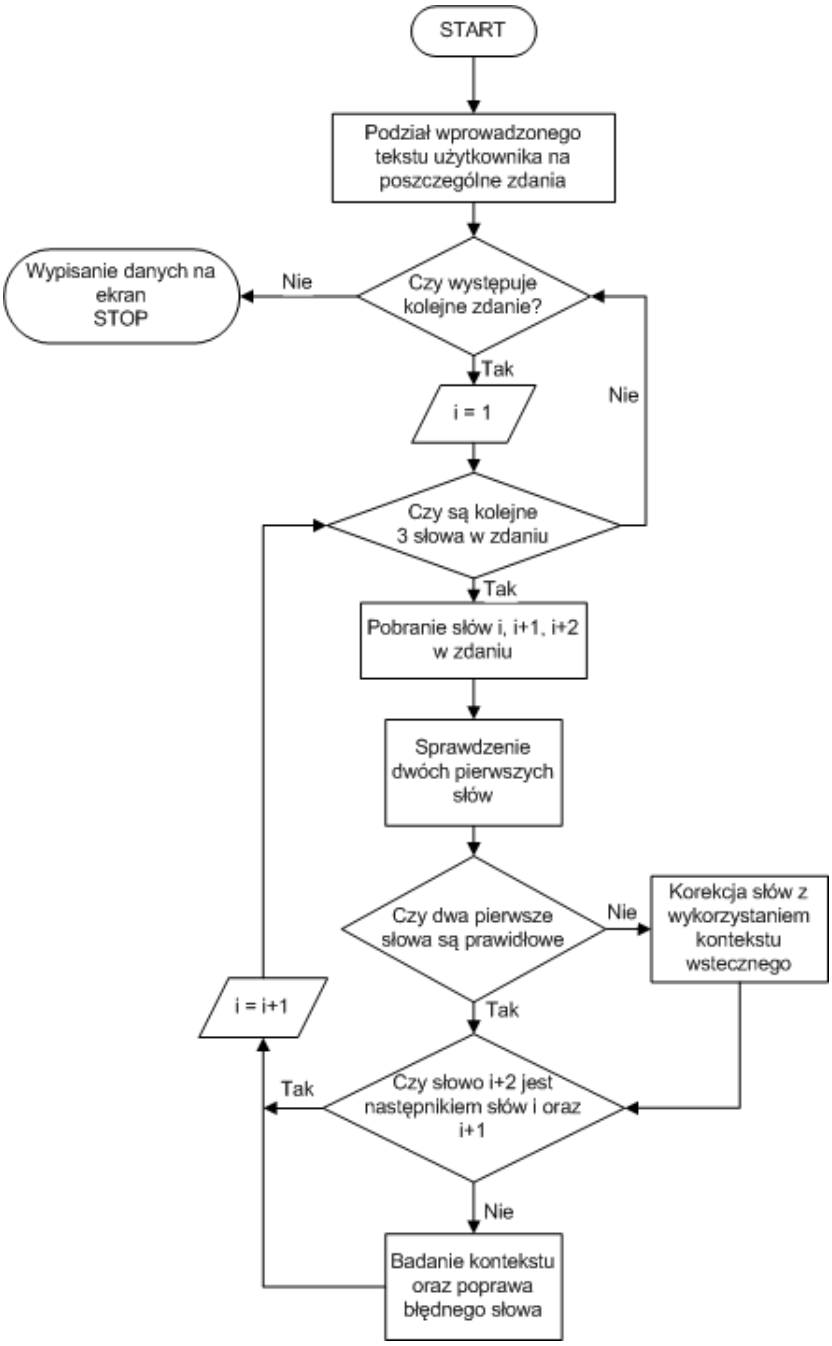
Następnie dla każdego słowa z wybranej listy, w kolejności częstości występowania, badana jest odległość edycyjna oraz różnica w długość słowa proponowanego, od wyrazu uznanego za błędny. Słowo o największej częstotliwości, o najmniejszej odległości edycyjnej oraz o najmniejszej różnicy w długości wstawiane jest jako poprawne do zdania, zamiast słowa przez program uznanego za błędne. W programie wypisywane są również inne, zaproponowane słowa o największej częstotliwości. Może się bowiem zdarzyć, że najbardziej pasujące słowo, w określonym kontekście słownym wprowadzonym przez użytkownika, nie występuje jako najbardziej popularne na liście lub też zostało znalezione słowo bliższe edycyjnie błędnego. W tym przypadku użytkownik ma możliwość zobaczenia dalszych propozycji czy nie pasują lepiej kontekstowo do całego zdania.

Szczegółowy schemat zaproponowanego algorytmu został przedstawiony na rysunku 4.10.

Na rysunku 4.11 został szczegółowo przedstawiony etap poprawy wyrazu uznanego za program jako błędny w danym kontekście wypowiedzi.

W dalszej części tego rozdziału zostanie przedstawione działanie zaproponowanego i zaimplementowanego algorytmu. Przykład ten jest oparty na wynikach uzyskanych przez algorytm automatycznej korekty testu. Tekst, który zostanie poddany badaniu brzmi *Ja nie chcieć tam iść*.

1. Ponieważ wprowadzony tekst składa się tylko z jednego zdania, tylko ono zostanie poddane dalszej analizie.
2. W pierwszym etapie sprawdzane jest czy słowa *ja* oraz *nie* występują jako poprzedniki oraz słowa aktualne dla dalszego kontekstu oraz czy dla słów *Ja nie* jako następnik wystąpi słowo *chcieć*. Z przeprowadzonych porównań algorytm stwierdził, że słowo *chcieć* jest niepoprawne, a początek wypowiedzi *Ja nie* są prawidłowe.
3. Choć słowo *chcieć* jest poprawnym słowem w języku polskim, program zasygnalizował, że nie występuje ono w zadanym kontekście dwóch poprzedzających go słów, więc rozpoczął etap korekty.
4. Na początku program sprawdza jakie słowa mogą wystąpić jako następniki słów *Ja nie* i zapisuje je na liście słów proponowanych. W tym przypadku na listę słów zostały wpisane słowa:
    - chcę o częstotliwości 13,

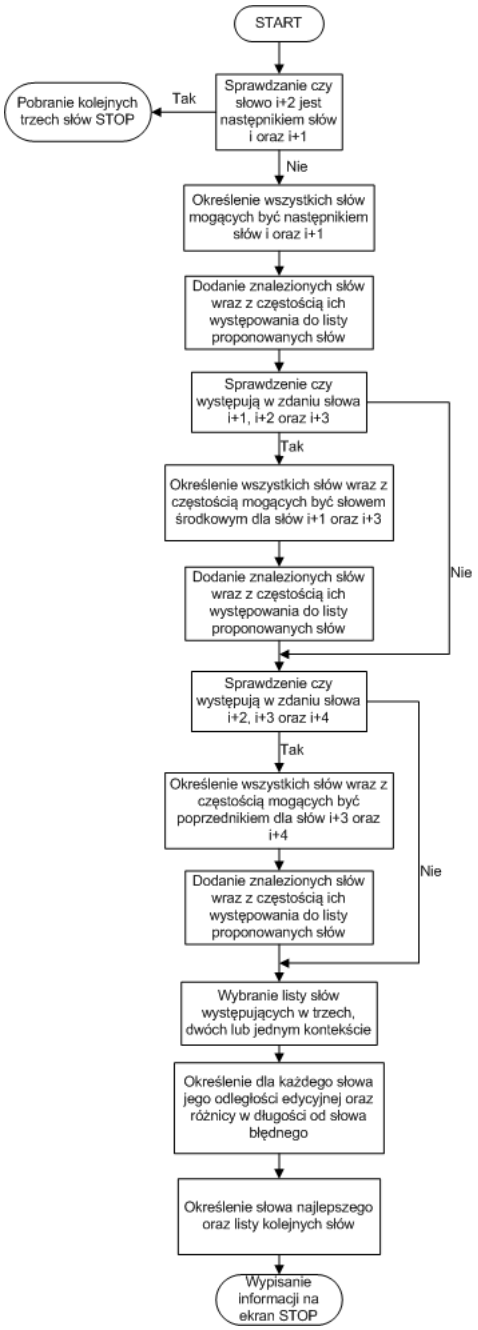


Rysunek 4.10: Algorytm kontekstowej korekcji słów

- chciałem o częstotliwości 7,
- chciałam o częstotliwości 5,
- mogę o częstotliwości 9.

5. Następnie sprawdzane są proponowane słowa jako następniki dla słowa *nie*, a poprzedniki dla słowa *tam*. Wszystkie znalezione słowa zostają dodane na listę proponowanych słów. Znalezione słowa to m. in.:

   - poszedłem o częstotliwości 5,
   - ruszę o częstotliwości 3,
   - chcę o częstotliwości 20,
   - chciałem o częstotliwości 9,



Rysunek 4.11: Algorytm kontekstowej korekcji słów

- chciałam o częstotliwości 6,
- mogę o częstotliwości 14.

Na listę proponowanych słów zostaną dopisane dwa kolejne, a dla pozostałych częstotliwością występowania będzie suma liczb występowania z poprzedniego i obecnego wyszukiwania.

6. Ostatnim etapem wyszukiwania jest znalezienie proponowanych słów jako poprzedników dla *tam iść*. Wszystkie znalezione słowa zostają dodane na listę proponowanych słów. Wśród znalezionych słów występują:

   - chcę o częstotliwości 26,
   - chciałem o częstotliwości 12,
   - mogę o częstotliwości 18.

7. Gdy już lista z możliwymi propozycjami została utworzona, sprawdzane jest czy występują w niej słowa, które mogą być zarówno poprzednikami, aktualnymi jak i następnikami dla słów w danym kontekście. W zaprezentowanym przykładzie takimi słowami są *chcę, chciałem* oraz *mogę*.

8. Następnie badana jest odległość edycyjna oraz długość każdego z proponowanych słów względem wyrazu uznanego za błędny. Wyniki takiego porównania wynoszą odpowiednio:

   - odległość edycyjna dla słowa *chcę* wynosi 3, natomiast różnica w długości 2,
   - odległość edycyjna dla słowa *mogę* wynosi 6, natomiast różnica w długości 2,
   - odległość edycyjna dla słowa *chciałem* wynosi 3, natomiast różnica w długości 2.

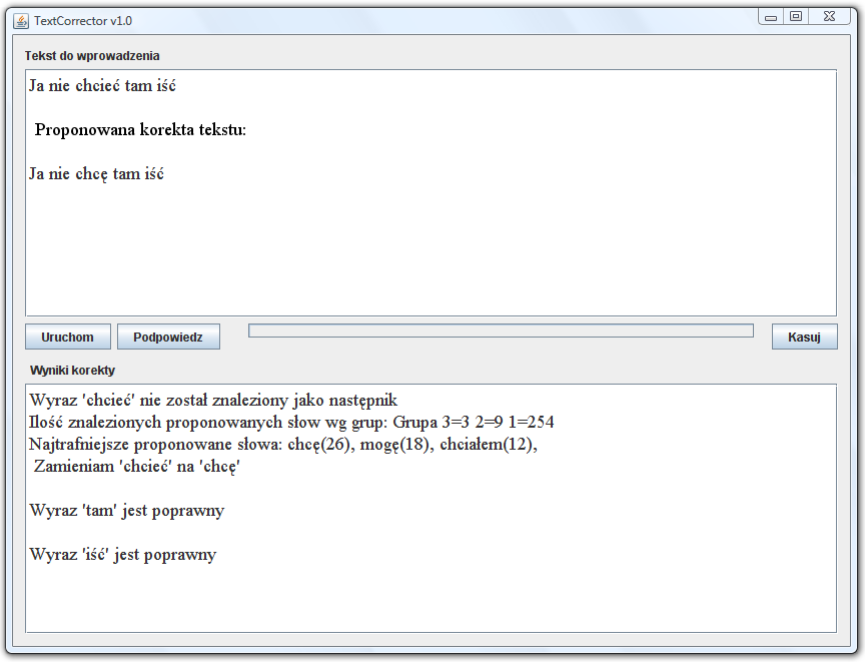
9. Przez program wybierane jest słowo o największej częstotliwości występowania, dla którego suma odległości edycyjnej i długości jest najmniejsza. W tym wypadku jest to słowo *chcę* o łącznej częstotliwości 26, o sumie 5. Zostaje więc zamienione słowo *chcieć* na słowo *chcę*, a dwa pozostałe słowa *chciałem* oraz *mogę* zostaną wypisane jako możliwe inne zamiany słowa *chcieć*.

10. Po znalezieniu najlepszego dopasowania i zamianie go z błędnym słowem, program sprawdza czy dla słów *nie chcę* możliwe jest wystąpienie słowa *tam*. W bazie znajduje się taka trójka, więc program przechodzi do badania kolejnych słów.

11. Ostatnią już analizą, jaką dokonuje program, jest sprawdzenie czy dla słów *chcę tam* możliwe jest wystąpienie słowa *iść*. W bazie znajduje się taka trójka. Ponieważ nie ma już dalszych słów do badania, algorytm kończy swoją pracę, a na ekranie aplikacji prezentowane są wyniki korekty tekstu.

12. Program wyświetla informację o wprowadzonym zdaniu przez użytkownika, sugerowanej poprawie tego zdania oraz o innych możliwościach zamiany wyrazów w zdaniu.

Wynik działania aplikacji został zaprezentowany na rysunku 4.12. Szczegółowy opis zawartości okna został zaprezentowany natomiast w rozdziale 4.5.

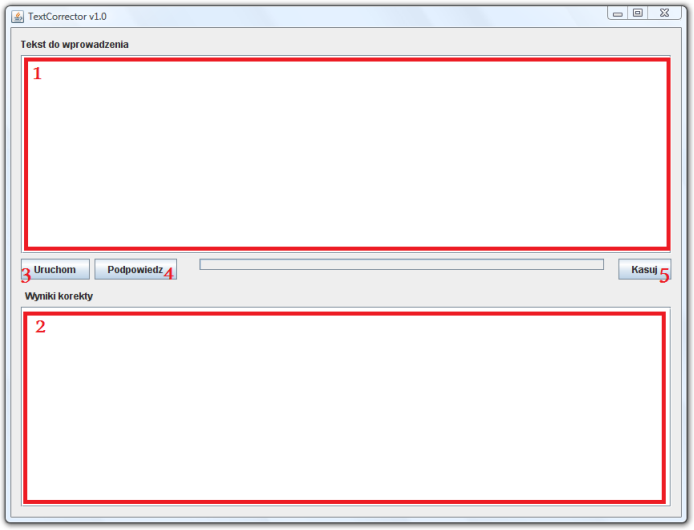


Rysunek 4.12: Wynik pracy algorytmu do kontekstowej korekcji tekstu dla przykładowego zdania

4.5. Główne okno programu

4.5.1. Opis

Po uruchomieniu głównego modułu aplikacji użytkownikowi zostanie wyświetlone okno, takie jak przedstawiono na rysunku 4.13.



Rysunek 4.13: Główne okno aplikacji

Składa się ono z kilku części:

- Miejsca, w którym użytkownik wprowadza swój tekst (oznaczone na rysunku numerem 1).
- Miejsca, w którym pojawia się tekst po korekcie (oznaczone na rysunku numerem 1).
- Miejsca, w którym pojawiają się szczegółowe informacje na temat dokonanej korekty (oznaczone na rysunku numerem 2).
- Miejsca, w którym pojawia się informacja czy wprowadzany wyraz przez użytkownika znajduje się w Słowniku języka polskiego oraz czy występuje we wprowadzonym kontekście(oznaczone na rysunku numerem 2).
- Przycisku, którym użytkownik uruchamia kontekstową korektę tekstu (oznaczone na rysunku numerem 3).
- Przycisku, którym użytkownik uruchamia podpowiedź dla wprowadzanego słowa (oznaczone na rysunku numerem 4).
- Przycisku, którym użytkownik kasuje dotychczas wprowadzony tekst (oznaczone na rysunku numerem 5).

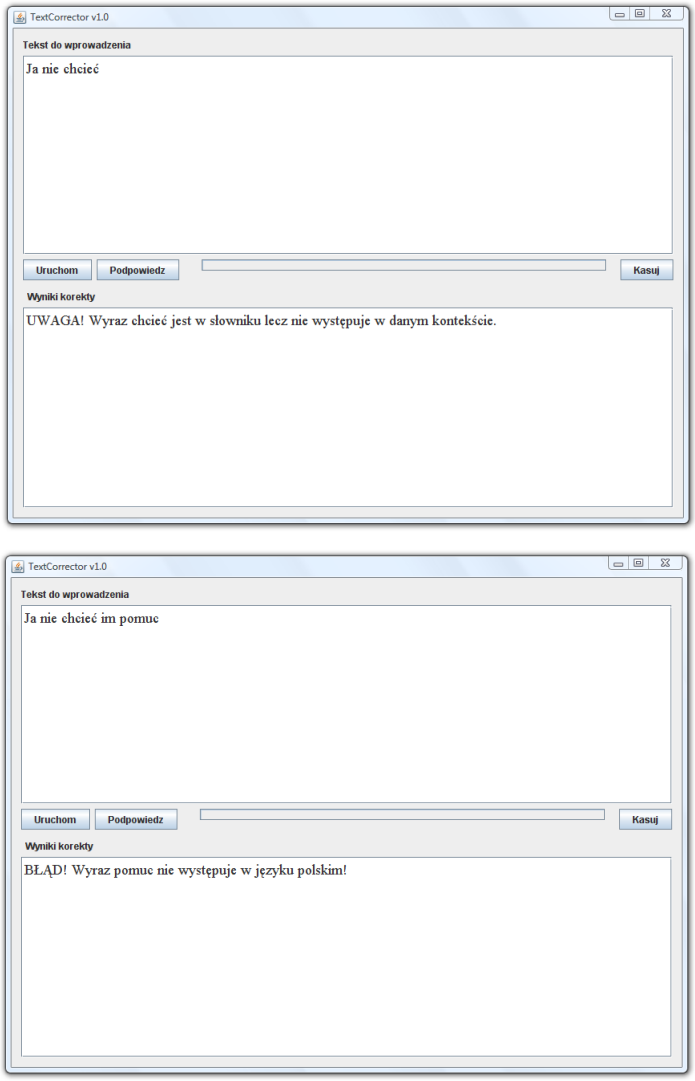
4.5.2. Tryby pracy

Użytkownik ma możliwość pracy z programem w kilku trybach. Należy do nich zaliczyć:

1. Tryb analizy czasu rzeczywistego wprowadzanych słów.

   Analiza ta polega na sprawdzeniu czy wprowadzane słowa występują w Słowniku języka polskiego oraz czy wprowadzane wyrazy (począwszy od trzeciego) występują w danym kontekście słów stojących przed nimi. Informacja czy wyraz jest wprowadzony w poprawnym kontekście, opiera się na sprawdzeniu czy wpisany wyraz występuje jako następnik dwóch wyrazów go poprzedzających.

Informacja o przeprowadzonej analizie prezentowana jest użytkownikowi, w miejscu oznaczonym numerem 2 na rysunku 4.13. Gdy wyraz występuje w Słowniku języka polskiego oraz występuje w danym kontekście, pojawia się komunikat: *"Wyraz ... jest w słowniku oraz jest w danym kontekście."* Natomiast jeżeli wyraz występuje w Słowniku języka polskiego, lecz nie występuje w danym kontekście, użytkownik zostaje ostrzeżony przez komunikat: *"UWAGA! Wyraz ... jest w słowniku lecz nie występuje w danym kontekście"*. Jeśli wyraz nie występuje w Słowniku języka polskiego, użytkownikowi zgłaszany jest błąd o treści: *"BŁĄD! Wyraz ... nie występuje w języku polskim!"*. Na rysunku 4.14 zostały przedstawione pojawiające się komunikaty, w czasie wprowadzania tekstu.



Rysunek 4.14: Analiza wprowadzanych słów - komunikaty

2. Tryb dopełniania wyrazów dla danego kontekstu wypowiedzi.

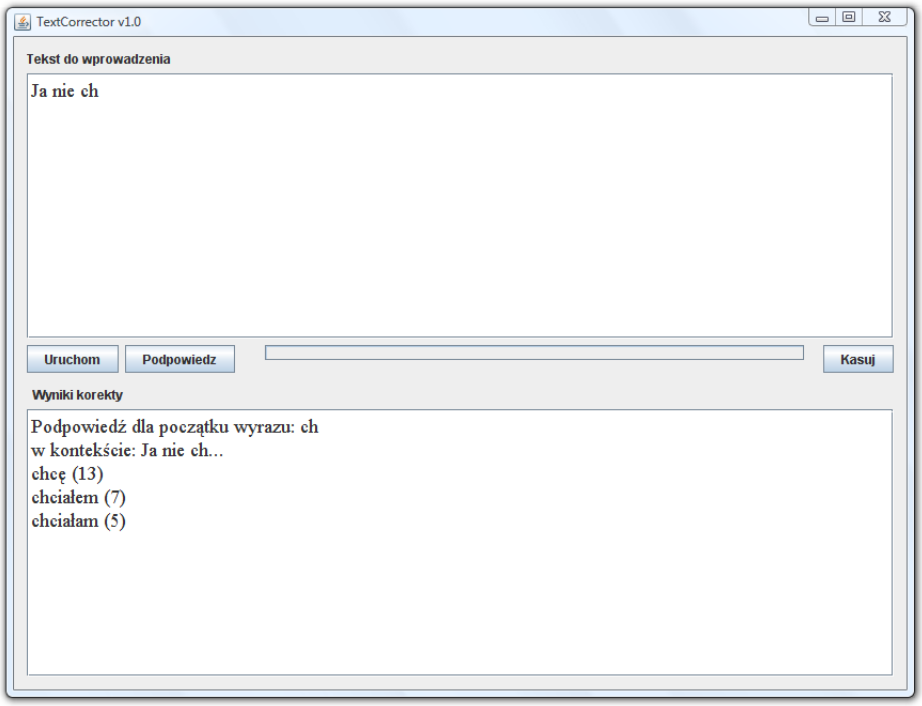
   Podczas wprowadzania przez użytkownika tekstu, począwszy od trzeciego wyrazu, algorytm może podpowiadać dokończenia słów, dla zadanego kontekstu wypowiedzi. Użytkownik po napisaniu początku słowa, może nacisnąć przycisk podpowiedź (przycisk oznaczony na rysunku 4.13 numerem 4). Algorytm sprawdza wtedy, jakie słowa występują dla danego kontekstu, które dodatkowo zawierają początek jaki wprowadził użytkownik.

   Przykładowo po wprowadzeniu początku zdania *Ja nie ch* użytkownik naciska przycisk *Podpowiedź*. Algorytm sprawdza jakie słowa-następniki o początku *ch*, występują dla słów *ja nie*, a następnie, najczęściej pojawiające się, prezentowane są w miejscu oznaczonym numerem 2 na rysunku 4.13.

Program, oprócz proponowanego wyrazu, wyświetla również częstotliwość występowania danego słowa w kontekście. Dla początku wyrazu *ch* program zaprezentuje poniższe propozycje:

- chcę o częstotliwości 13,
- chciałem o częstotliwości 7,
- chciałam o częstotliwości 5.

Na rysunku 4.15 zostały zaprezentowane pojawiające się podpowiedzi uzupełnienia wyrazu, dla danego kontekstu słownego.



Rysunek 4.15: Podpowiedzi dla początku wyrazu

3. Tryb automatycznej kontekstowej korekcji tekstów.

   Gdy użytkownik wprowadził żądany tekst, może uruchomić algorytm przyciskiem oznaczonym numerem 3 na rysunku 4.13, który sprawdzi poprawność wprowadzonego tekstu oraz dokona jego korekty, jeśli jest wymagana. Możliwe jest bowiem, że użytkownik nie miał włączonego mechanizmu sprawdzania poprawności tekstu lub zignorował pojawiające się ostrzeżenia i błędy, sugerujące zmianę wyrazów. Szczegóły działania tego trybu zostały przedstawione poniżej.

4.5.3. Kontekstowa korekta tekstów

Po żądaniu automatycznej kontekstowej korekty tekstu zostanie uruchomiony algorytm, który szerzej jest opisany w rozdziale 4.4. Wynikiem jego działania jest:

- sprawdzenie wszystkich wpisanych wyrazów czy są poprawne ortograficznie,
- sprawdzenie wszystkich wpisanych wyrazów czy są możliwe w zadanym przez użytkownika kontekście,
- poinformowanie użytkownika czy wyraz został wprowadzony poprawnie (wg Słownika języka polskiego oraz w dobrym kontekście),
- jeśli wyraz został wprowadzony niepoprawnie, nastąpi jego poprawa poprzez:
  - znalezienie wyrazów, które są prawidłowe dla danego kontekstu,

  - wypisanie najczęściej pojawiających się wyrazów, wraz z ich częstotliwością na ekran,
  - zamianę błędnego wyrazu w zdaniu wprowadzonym przez użytkownika na poprawne, które występuje najczęściej dla danego kontekstu.
- wypisanie informacji o wszystkich wprowadzonych wyrazach czy są prawidłowe,
- wypisanie zaproponowanego przez algorytm poprawnego zdania.

Końcowe zdania wypisane przez program, nie muszą się różnić od zdań, które wprowadził użytkownik. Gdy wprowadzone zdania zawierały błędy - zostały poprawione, natomiast gdy zdania były poprawne - algorytm nie znalazł w nich błędów, a aplikacja wypisała zdania wprowadzone na początku.

5. Porównanie mechanizmów korekcji tekstu stosowanych obecnie dla języka polskiego

5.1. Najbardziej popularne edytory tekstu - opis

Najbardziej popularnymi programami, do edycji tekstu dla języka polskiego, są zaledwie dwa:

1. Microsoft Word - edytor tekstu wchodzący w skład pakietu biurowego Microsoft Office.

   Najnowsza wersja tego programu - Microsoft Word 2007 - ułatwia tworzenie i udostępnianie profesjonalnie wyglądającej zawartości, dzięki połączeniu kompletnego zestawu narzędzi do pisania i łatwego w użyciu interfejsu użytkownika Microsoft Office [17].

2. Writer - edytor tekstu wchodzący w skład pakietu biurowego OpenOffice.org.

   OpenOffice.org to pakiet biurowy, działający w wielu systemach operacyjnych i środowiskach, z otwartym dostępem do kodu źródłowego. Można go stosować legalnie, bez żadnych opłat w domach, firmach, urzędach i szkołach.

   Pakiet jest obecnie wyposażony w polski słownik ortograficzny, słownik synonimów, odpowiednie zasady podziału wyrazów, polską dokumentację i pomoc, zasady autokorekty, a także korektor gramatyczny. Od 27 czerwca 2007 OpenOffice.org należy do Koalicji na Rzecz Otwartych Standardów (KROS) [19].

Nie ulega wątpliwości, że Microsoft Office ma wielokrotnie większą bazę użytkowników niż OpenOffice, niemniej jednak każda kolejna wersja skłania do dokonywania porównań. Oczywistą zaletą OpenOffice jest to, że jest bezpłatny. Dostrzegło to ostatnio, co warto podkreślić, Ministerstwo Edukacji Narodowej, które uznało oficjalnie ten pakiet za dojrzały i wart zarekomendowania w instytucjach podległych ministerstwu.

W tym kontekście warto zwrócić uwagę na jeszcze jedną sprawę - ewoluujący ostatnio model użytkowania komputerów, wiążący się z usługami sieciowymi, każe na nowo rozpatrywać skuteczność wykorzystywania pakietów biurowych w zespołach osób. Idea ”cloud computing”, czyli wykorzystywania miejsca na serwerach i ich mocy obliczeniowych, stawia nowe zadania przed klasycznymi programami biurowymi. Są to przede wszystkim: dostępność dokumentów z dowolnego komputera i dowolnego miejsca na świecie, jak też możliwość ich współdzielenia z innymi użytkownikami. Microsoft podejmuje to wyzwanie, wprowadzając odpowiednie usługi w ramach Office Live. W programie tym pojawiają się wtyczki pozwalające zapisywać dokumenty Microsoft Office w innych usługach, jak np. Zoho. OpenOffice również posiada wtyczkę (autorstwa Przemysława Rumika), która pozwala umieszczać dokumenty w usługach *Dokumenty Google* i *Zoho Office*. Wydaje się, że ten aspekt oprogramowania biurowego będzie sukcesywnie zyskiwać na znaczeniu. [8]

5.2. Porównanie Microsoft Word z OpenOffice.org Writer

W internecie dostępnych jest wiele artykułów, które dotyczą porównania tych dwóch najpopularniejszych edytorów. Wpisując w wyszukiwarkę Google (`http://www.google.pl/`) frazy *”porównanie word*

writer" otrzymujemy odpowiedź *"Wyniki 1 - 10 spośród około 1,730 dla zapytania porównanie word writer."* W dalszej części zostanie przedstawione porównanie tych programów, badające głównie sposób i możliwości korekcji tekstów. Wyniki porównania opierają się głównie na artykule [8].

Już w poprzednich wersjach edytora Word pojawiło się narzędzie, wykonujące automatyczną korektę wpisywanych wyrazów. Jego zadaniem było poprawianie błędnych wyrazów, podczas ich wpisywania. Narzędzie to działa dobrze, ale przy pisaniu tekstów językiem ściśle technicznym niezbędne bywają ręczne poprawki po tym, co to autokorekta zepsuła.

Writer również posiada autokorektę, ale działa ona mniej radykalnie od tej w Wordzie. Jedynym minusem autokorekty Writera jest brak opcji wyłączenia błędnie wciśniętego klawisza Caps Lock, gdy dokonuje się korekty tekstu. Przyczyny można upatrywać w tym, że OpenOffice jest pakietem przeznaczonym dla wielu systemów operacyjnych, a programowe wyłączanie klawisza Caps Lock w każdym z nich dokonuje się nieco inaczej.

Writer zawiera ciekawą opcję podpowiedzi słów używanych w danym dokumencie. Gdy program znajduje słowo, które pasuje do wpisywanych początkowych liter, podpowiada je, wyświetlając w negatywie. Wystarczy nacisnąć Enter, by zaakceptować propozycję. Niestety nie działa ona dla wszystkich wyrazów w języku polskim, gdyż nie uwzględnia ona kontekstowo polskiej fleksji.

Ważną opcją OpenOffice'a jest moduł sprawdzania pisowni w języku polskim. Chociaż jest odczuwalnie wolniejszy od Worda 2007, działa bardzo dobrze. Nie odstaje od swojego komercyjnego odpowiednika, można spotkać się również ze stwierdzeniami, że jest od niego nawet lepszy. Moduł ortograficzny Worda, podczas sprawdzania pisowni w języku polskim, nie uwzględnia wielu nazw geograficznych. Przykładem może być miejscowość Redmond, która jest siedzibą Microsoftu. Ponadto klasyfikacja słów (np. używanych w potocznym języku) jest dość kontrowersyjna.

Najmocniejszą stroną Writera, w tej dziedzinie, jest bardzo ważny dodatek - LanguageTool. Narzędzie to integruje się z Writerem, umożliwiając kontrolę poprawności gramatycznej oraz stylistycznej także w języku polskim. LanguageTool sprawdza wiele reguł (ponad 800), wliczając bardzo zaawansowane (błędy odmiany, pleonazmy, błędy leksykalne). W odróżnieniu od kontroli pisowni, sprawdzanie we Writerze nie odbywa się podczas pisania, lecz należy ją włączyć, po napisaniu tekstu. Word 2007 ma podobne narzędzie, które może działać także w trakcie pisania, ale jego skuteczność pozostawia wiele do życzenia. LanguageTool podaje ponadto propozycje zmian (na przykład umieszczenie brakującego przecinka), ale nie zawsze potrafi to zrobić prawidłowo i czasami wybiera dwa razy podobne fragmenty zdania. Mimo niedociągnięć, LanguageTool jest jednym z poważnych argumentów za używaniem Writera, gdyż ten dodatek bardzo pomaga przy pisaniu wielu tekstów.

Jak zostało wspomniane wcześniej, porównanie to dotyczy jedynie modułu korekcji tekstu w dwóch edytorach. Można zauważyć, że lepszym modułem do sprawdzania poprawności tekstów dysponuje darmowy Writer niż Microsoft Word. W dalszej części tego rozdziału zostaną porównane wyniki korekcji tekstów w tych programach, z zaimplementowanym algorytmem, bazującym na Grafie Przyzwyczajeń Lingwistycznych, wykorzystując kilka przykładowych zdań. Do testów zostały wykorzystane najpopularniejsze edytory biurowe:

- Writer wchodzący w skład pakietu OpenOffice.org 3.0.1.
- Microsoft Word 2007 wchodzący w skład pakietu Microsoft Office 2007.

Algorytm kontekstowej korekty tekstu został oparty na grafie LHG, zbudowanym w oparciu o bazę danych, która zawierała:

- 273 211 unikalnych słów w języku polskim wraz z określeniem częstości ich występowania, na podstawie Słownika frekwencyjnego języka polskiego [7],
- 8 058 155 trójek słów w określonej formie fleksyjnej w ich rzadkim kontekście w obrębie zdań.

5.3. Moduł do konstrukcji zdań

Pierwsze porównanie będzie dotyczyć modułu do konstrukcji nowych zdań. Wiele razy użytkownik wpisując początek zdania, nie wie jakiego należy użyć następnego wyrazu. W programie Writer obecny jest moduł do "podpowiadania" końcówki wyrazu, jednak w tym porównaniu podpowiadany ma być cały następny wyraz. Niestety żaden z pośród dwóch wspomnianych programów, nie dostarcza takiego modułu. Częściowo moduł do podpowiadania (uzupełniania) zdań występuje w wyszukiwarce Google, jednak ogranicza się on jedynie do najczęściej wpisywanych fraz w wyszukiwarce, które nie muszą być reprezentatywne dla rozważanego języka.

5.3.1. Opis własnego modułu

W tym porównaniu, zdanie jakie będzie porównywane brzmi: *"Ja nie mam zamiaru rezygnować z walki."*. Wpisując całą frazę do wyszukiwarki Google, można uzyskać odpowiedź: *Wyniki 1 - 10 spośród około 19,000 dla zapytania Ja nie mam zamiaru rezygnować z walki.*

Główne okno programu do konstrukcji grafu LGH, który zarazem jest modułem do "podpowiadania" dalszych wyrazów, zostało zaprezentowane na rysunku 4.5. Moduł bazuje na słowach-trójkach, zgromadzonych w bazie danych dla algorytmu.

Pierwszą czynnością jaką należy wykonać, to wpisanie dwóch początkowych słów, dla których mają pojawić się propozycje kolejnych wyrazów. Po ich wpisaniu w odpowiednie pola i naciśnięciu przycisku "Rysuj", użytkownikowi zostanie zaprezentowana część grafu, która powiązana jest z wierzchołkami reprezentującymi te słowa. Obok nich, w nawiasie, zostaną podane częstości występowania danego słowa w Słowniku frekwencyjnym, zbudowanym dla języka polskiego [7]. Gdy początkowe wyrazy zostaną wpisane błędnie, algorytm ich nie odnajdzie w swojej bazie i użytkownik będzie musiał wpisać je raz jeszcze, tym razem poprawnie.

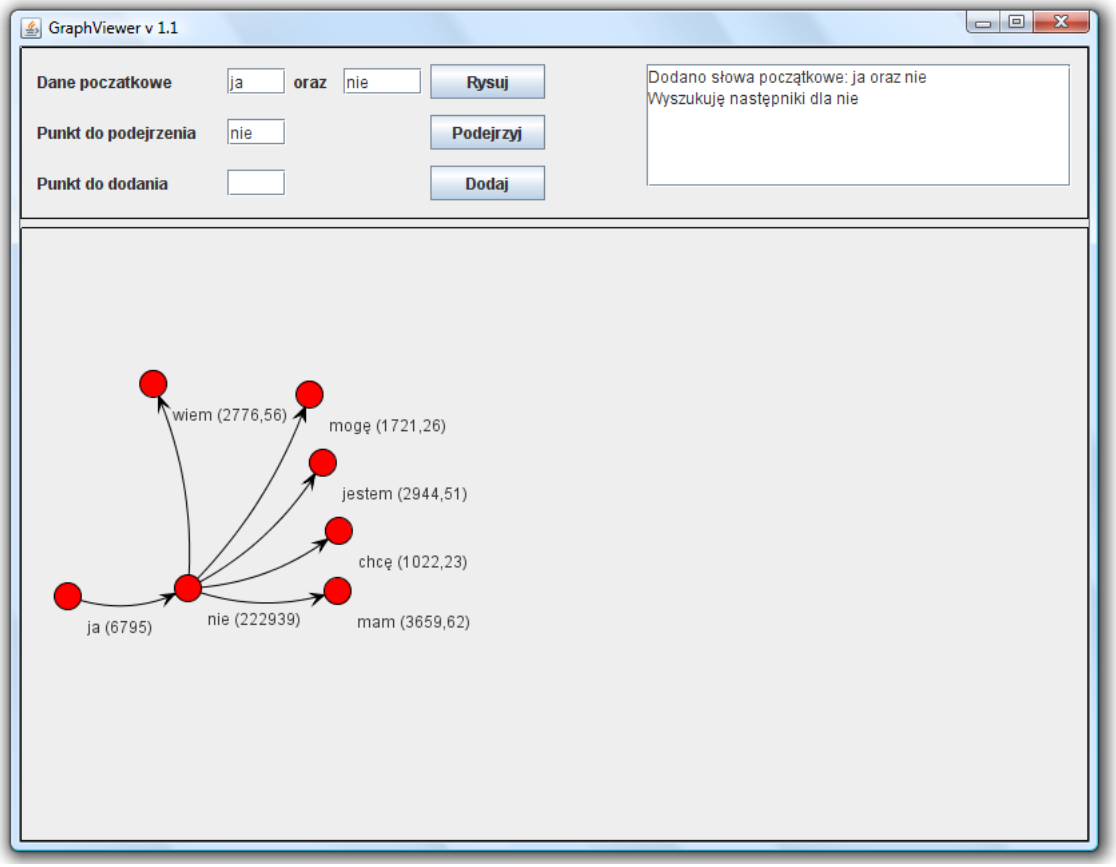
Po ukazaniu się początkowych słów, użytkownik ma możliwość, w pole "punkt do podejrzenia" (które jest domyślnie wypełniane ostatnim wyrazem), wpisać wyraz w grafie, dla którego mają być podane możliwe do wystąpienia następne słowa. Poczynając od trzeciego słowa, w nawiasie obok jego nazwy podawane są dwie liczby. Pierwsza z nich to wspominana już częstość występowania danego słowa w słowniku, natomiast druga liczba to częstość występowania danego wyrazu w kontekście dwóch poprzedzających go słów.

Na rysunku 5.1 został przedstawiony pierwszy etap "podpowiadania" dalszych słów.

Dla dwóch początkowych wyrazów *ja* oraz *nie* zostało zaprezentowanych pięć najczęściej występujących wyrazów jako ich następników. Ilość pojawiających się możliwości jest konfigurowalna w programie, ponieważ ilość wszystkich możliwych następników znalezionych przez algorytm, dla wprowadzonych słów to aż 192 pozycje.

Jak można zauważyć najczęściej po słowach *ja* oraz *nie* występują wyrazy:

- mam - 62 razy,
- wiem - 56 razy,
- jestem - 51 razy,
- mogę - 26 razy,
- chcę - 23 razy.



Rysunek 5.1: Początkowa faza budowy grafu

Oprócz tych słów na dalszych pozycjach (nie zostały one zaprezentowane w tym grafie) znalazły się takie słowa jak:

- lubię - 22 razy,
- będę - 15 razy,
- chciałem - 7 razy,
- przepadam - 2 razy.

Wpisując taki sam początek do wyszukiwarki Google, można uzyskać takie propozycje kolejnych wyrazów:

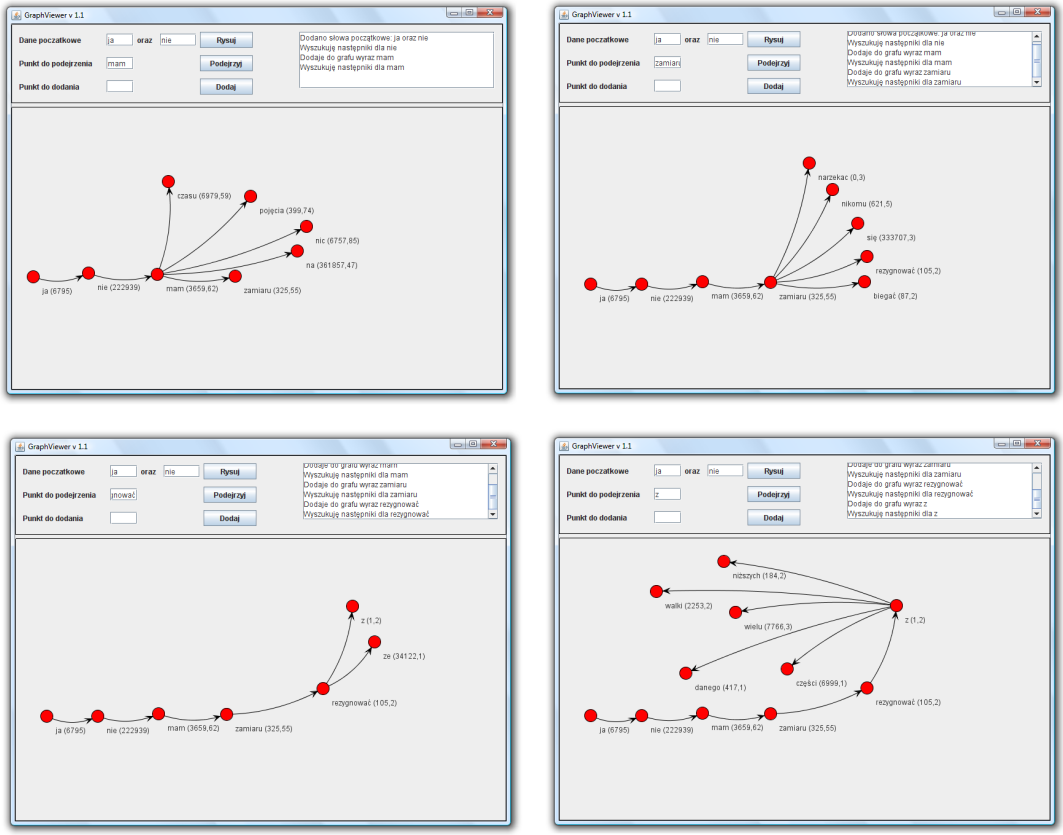
- ”ja nie chce wiele” 9 680 000 wyników,
- ”ja nie kocham ciebie ty nie kochasz mnie” 681 000 wyników,
- ”ja nie lubie nikogo” 527 000 wyników,
- ”ja nie chcę spać” 301 000 wyników,
- ”ja nie odchodzę kiedy trzeba” 205 000 wyników.

Znacząco lepsze propozycje wyrazów uzyskano z wykorzystaniem skonstruowanego grafu LHG, w ramach tej pracy. Szczegółowe porównanie działania samego algorytmu zostało przedstawione w rozdziale 5.5. Podczas porównania kolejne wyrazy były proponowane zarówno przez zaimplementowany algorytm, jak i przez wyszukiwarkę Google. Do takich wyrazów można zaliczyć *chcę, lubię*. W wyszukiwarce podane są one bez polskich znaków, co może sugerować, iż użytkownicy częściej wpisują wyrazy bez polskich ”ogonków”. Wyrazy *lubie* oraz *chce* wystąpiły w bazie danych, dla skonstruowanego algorytmu, o

Rysunek 5.2: Propozycje dokończenia zdania ”ja nie...” przez wyszukiwarkę Google

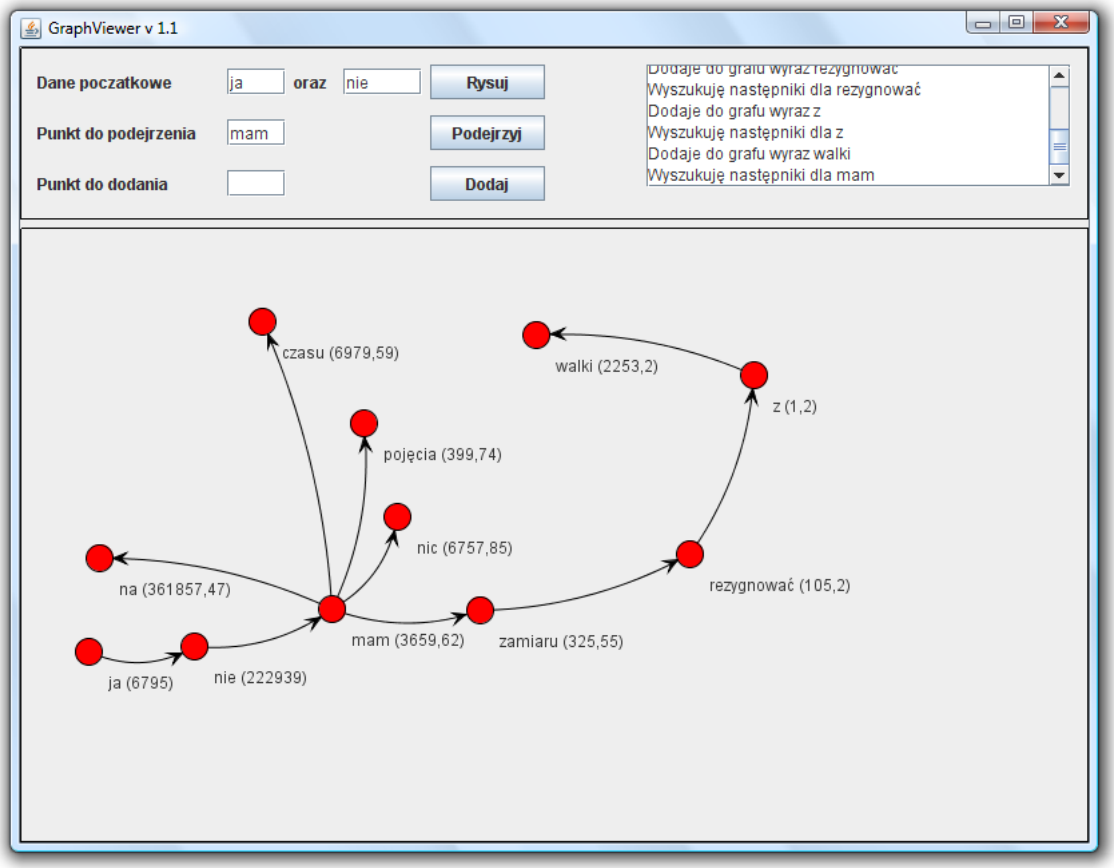
krotności odpowiednio 5 oraz 2 razy. Zaimplementowany pająk internetowy przeszukiwał bowiem wszystkie strony internetowe, więc natrafił również na liczne blogi, na których ich twórcy nie zawsze używają poprawnej konstrukcji wyrazów i budowy zdań. Jednak można przyjąć, że dla tak dużej bazy danych (ponad 8 milionów rekordów), wyrazy uznane za błędne, w poprawnej pisowni wystąpią dużo rzadziej, niż wyrazy poprawne.

Na dalszych rysunkach 5.3 przedstawiono kolejne propozycje dalszych wyrazów dla budowanego zdania.



Rysunek 5.3: Kolejne etapy podpowiadania następników dla budowanego zdania

W każdej chwili, podczas konstrukcji grafu LHG, można wrócić do poprzedniego wierzchołka i dla niego wyszukiwać następników, zachowując zbudowaną już strukturę. Taka sytuacja została zaprezentowana na rysunku 5.4. Po zbudowaniu grafu dla zdania *"Ja nie mam zamiaru rezygnować z walki."* użytkownik wrócił do wierzchołka *mam* i dla niego wyznaczył następniki.



Rysunek 5.4: Powrót do wcześniejszego wierzchołka

Ponieważ żaden z dwóch wiodących w popularności na rynku edytorów tekstu nie posiadał modułu do konstrukcji nowych zdań, testy zostały przeprowadzone jedynie dla stworzonego programu. Dodatkowo zostało sprawdzone, czy generowane kolejne słowa są poprawne językowo. W bardzo dużej mierze wyrazy były poprawne i zgodne z zasadami pisowni. Kilka z nich wystąpiło w formach, np. "bez ogonków" choćby zaprezentowane w przykładzie powyżej wyrazy *lubie, chce*. Jak zostało już wspomniane, wynika to w głównej mierze z wcześniejszego sprawdzania przez pająka internetowego wielu forów, grup dyskusyjnych, blogów na których przeważnie młodzież wypowiada się używając skróconych form wyrazów lub też nie przestrzegając zasad interpunkcji oraz ortografii. Z powodu tych błędów językowych w pracy zostało zbudowanych, przez pająka internetowego, kilka baz z trójkami słów. Jedna z nich zawiera, np. słowa z wszystkich znalezionych polskich stron internetowych. Inna natomiast zawiera dane przefiltrowane - robot internetowy nie wchodził na strony internetowe, w których nagłówku znajdowały się wyrazy takie jak: "blog" lub "blox".

5.4. Porównanie mechanizmów automatycznej korekty tekstów

W rozdziale tym zostaną szczegółowo zaprezentowane i porównane dwa zaimplementowane mechanizmy:

1. automatycznego sprawdzania poprawności wpisywanych słów,
2. podpowiadania aktualnie wpisywanego słowa.

z podobnymi algorytmami, występującymi w programach Microsoft Word 2007 oraz Open Office Writer 3.0.

5.4.1. Automatyczne sprawdzanie poprawności wpisywanych słów

Bardzo ważną i istotną rolę podczas wprowadzania tekstu, pełni automatyczne sprawdzanie wpisywanych wyrazów. Dzięki temu użytkownik zaraz po napisaniu wyrazu, jest informowany o ewentualnym wystąpieniu w nim błędu. Moduły do automatycznego sprawdzania poprawności wyrazów, dostępne są bez konieczności dodatkowej ich instalacji we wspomnianych wcześniej dwóch edytorach. Użytkownik zaraz po zainstalowaniu programu, ma możliwość sprawdzania pisowni. Moduły te bazują głównie na wspomnianej w rozdziale 3.1 odległości edycyjnej Levensteina.

Schemat ich działania wygląda następująco:

1. Program sprawdza, czy wpisane słowo występuje we wbudowanym w program słowniku języka polskiego.
2. Jeśli wpisany przez użytkownika wyraz nie występuje w słowniku, badana jest odległość edycyjna dla wyrazów podobnych.
3. Błędny wyraz zostaje podkreślony na czerwono.
4. Po naciśnięciu na wyrazie prawym klawiszem myszy, prezentowane są możliwości zmiany wyrazu na wyraz poprawny, uzyskany za pomocą wbudowanego mechanizmu korekty.

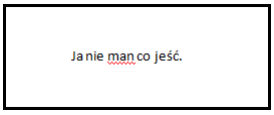
Dodatkowo w programie Microsoft Word 2007, gdy wyraz jest bardzo bliski (edycyjnie) jedynie do jednego wyrazu, który już we wpisywanym tekście przez użytkownika został kilkukrotnie użyty, program automatycznie go zamienia. Niestety bardzo często program Microsoft Word lub Writer podpowiada kilka wyrazów. Z kontekstu wynika jednak, że powinien zostać użyty jedynie jeden wyraz. Inne proponowane wyrazy, choć są w bliskiej odległości edycyjnej, w ogóle nie pasują do kontekstu.

Zaimplementowany algorytm oprócz sprawdzania, czy słowo występuje w stworzonym Słowniku języka polskiego, sprawdza również, czy dany wyraz został użyty w poprawnym kontekście. Takiego mechanizmu sprawdzania wprowadzonych wyrazów nie oferują inne pogramy.

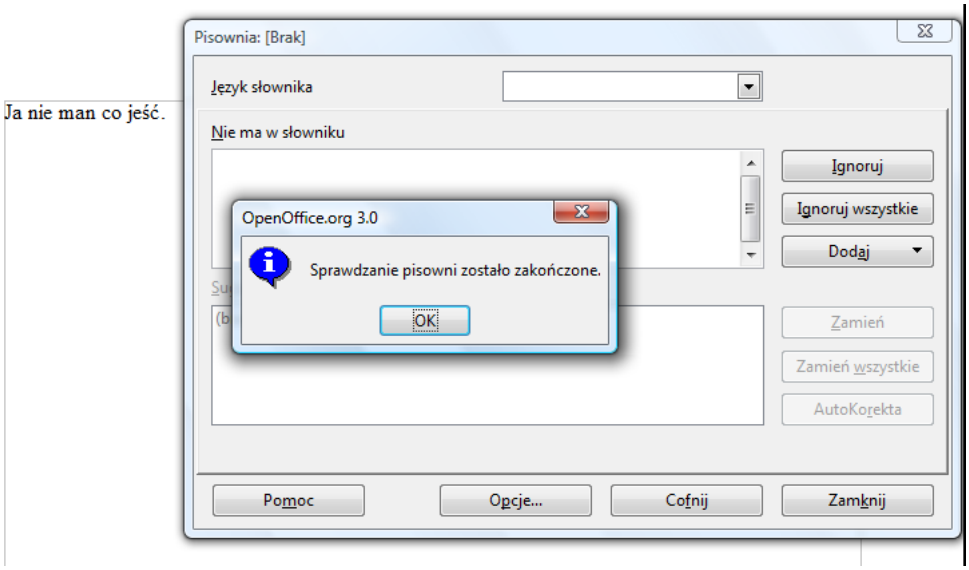
Programy użyte do testów badają jedynie czy wpisany przez użytkownika wyraz występuje w słowniku. Jeśli występuje, programy uważają go za poprawny. Tak więc użytkownik podczas wpisywania tekstu może popełnić błąd polegający na zmianie jednego wyrazu w drugi (również występujący w słowniku) - programy te jednak nie będą w stanie tego błędu rozpoznać. Dla edytorów bowiem wyraz występujący w słowniku jest zawsze poprawny.

Poniżej zostanie zaprezentowane, jak wcześniej wspomniane programy, przeprowadzają korekcję dla zdania: *Ja nie man co jeść.* Prawidłowe zdanie, jakie chciano wpisać to: *Ja nie mam co jeść.* Celowo została wprowadzona pomyłka, zamieniająca jedynie jedną literę w słowie *mam* na słowo *man*.

Jak można zauważyć, program Microsoft Word 2007 zasygnalizował błąd, podkreślając słowo *man* na czerwono. Program Writer 3.0 nie zasygnalizował użycia błędnego wyrazu (celowo została włączona korekcja pisowni, która nie wykazała żadnego błędu). Skonstruowany algorytm podczas wpisywania tekstu, znalazł błąd w słowie *man* i poinformował o tym użytkownika komunikatem *"BŁĄD! Wyraz man nie występuje w języku polskim!"*, mówiącym, że wprowadzony wyraz nie znajduje się w słowniku. W tym fragmencie nie zostaną opisane możliwe "poprawienia" tekstu, gdyż został temu poświęcony osobny rozdział 5.5.



Rysunek 5.5: Zdanie ”Ja nie man co jeść” i sygnalizacja zauważonego błędu w programie Microsoft Word 2007



Rysunek 5.6: Zdanie ”Ja nie man co jeść.” i brak sygnalizacji błędu w programie Writer 3.0

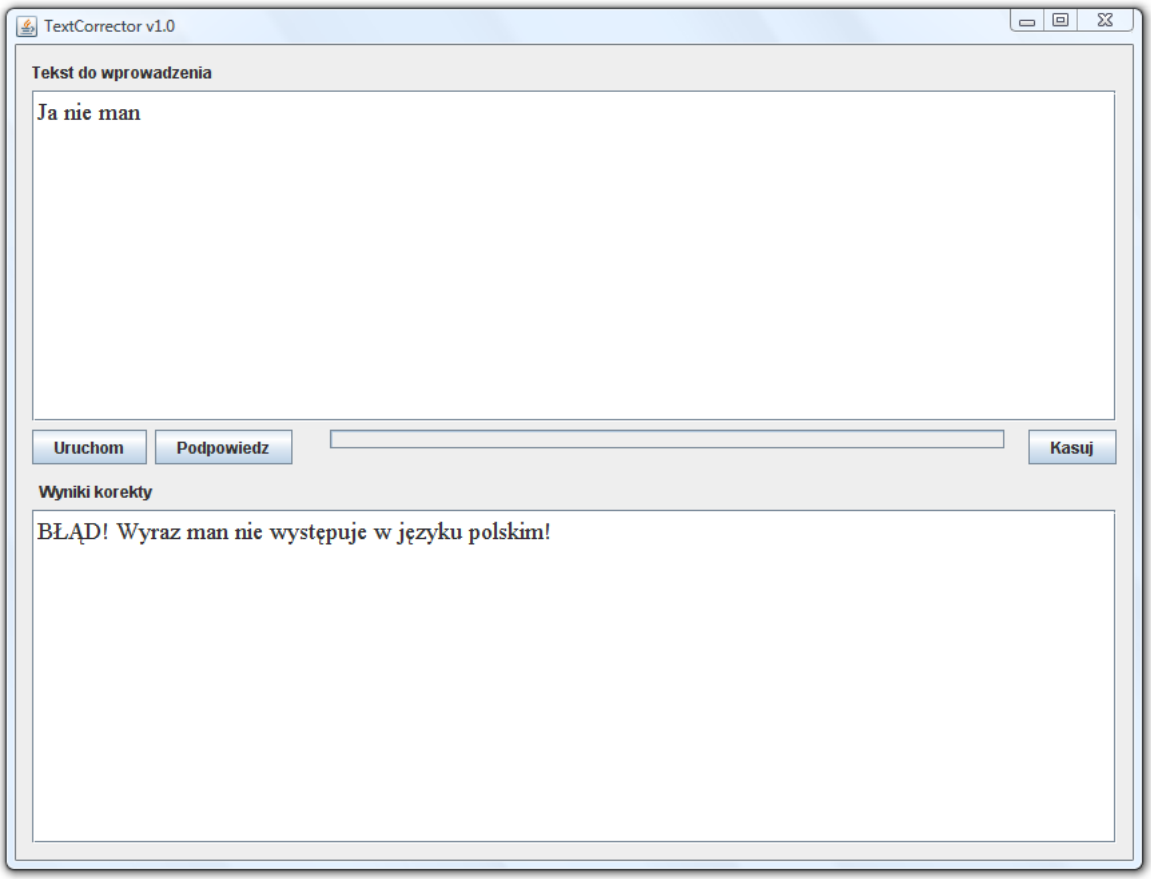
W kolejnym zdaniu, na którym został przeprowadzony test, została również wprowadzona pomyłka. Tym razem dla zdania *Ja nie mam co jeść.* zamiast słowa *mam* został wprowadzony wyraz *nam*, który co prawda występuje w Słowniku języka polskiego, lecz został użyty w nieodpowiedniej formie fleksyjnej, gramatycznej i kontekście.

Jak widać na rysunkach 5.8 oraz 5.9 , zarówno program Microsoft Word 2007 jak i Writer 3.0 nie zasygnalizował użycia błędnego wyrazu (celowo została włączona korekcja pisowni, która jednak nic nie wykazała). Stało się tak, ponieważ słowo *nam* występuje w języku polskim, a programy te nie analizują kontekstu sąsiednich słów. Opracowany algorytm, podczas wpisywania tekstu, znalazł błąd w słowie *nam* i poinformował o tym użytkownika. Stało się tak, ponieważ oprócz sprawdzania czy słowo występuje w słowniku, algorytm zweryfikował również kontekst jego użycia. W podanym zdaniu choć słowo *nam* jest poprawne, to w podanym kontekście dwóch poprzedzających go słów, nigdy nie wystąpiło. Użytkownik jest o tym informowany komunikatem *”UWAGA! Wyraz nam jest w słowniku, lecz nie występuje w danym kontekście.”*

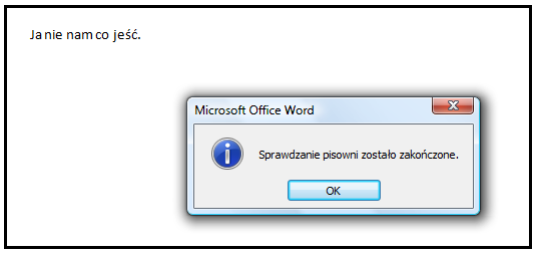
Podczas wykrywania potencjalnych błędów stworzony program, wykorzystując graf LHG zadziałał najlepiej. Wykrył on błąd zarówno w pierwszym, jak i drugim zdaniu. Program Microsoft Word 2007 zasygnalizował błąd w jednym zdaniu, w którym użyto błędnego wyrazu *man*. Niestety oba programy Microsoft Word i Writer nie poprawiły błędnego użycia wyrazu *nam*. Mechanizmy w nich użyte stwierdziły, że jest on poprawny. Jedynie opracowany algorytm kontekstowej korekty tekstów, wykrył potencjalny błąd użycia poprawnego wyrazu, lecz w złym kontekście.

Istnieje o wiele więcej przykładów, w których programy Microsoft Word oraz Writer nie radzą sobie z automatycznym sprawdzaniem wyrazów, które zostały użyte błędnie. Nie oznacza to jednak, że zaimplementowany algorytm działa bezbłędnie i zawsze znajduje potencjalne błędy. Dla kilku przypadków jest odwrotnie. Program bazując na grafie LHG, stwierdza, że w zdaniu występuje błąd, choć wprowadzony tekst jest poprawny. Istnieją dwie możliwości takiego działania:

1. Gdy wprowadzony wyraz nie występuje w wykorzystywanym Słowniku języka polskiego. Sytuacja ta ma miejsce głównie, gdy wprowadzany wyraz jest nazwą geograficzną. Przykładowo w frekwen-



Rysunek 5.7: Początek zdania ”Ja nie man” i sygnalizacja zauważonego błędu przez stworzony algorytm

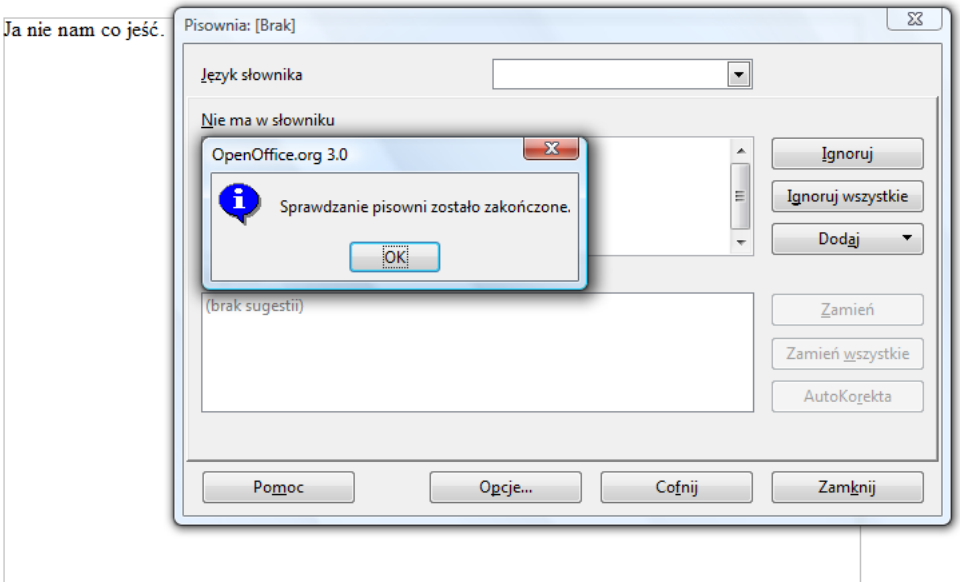


Rysunek 5.8: Zdanie ”Ja nie nam co jeść.” i brak sygnalizacji błędu w programie MS Word 2007

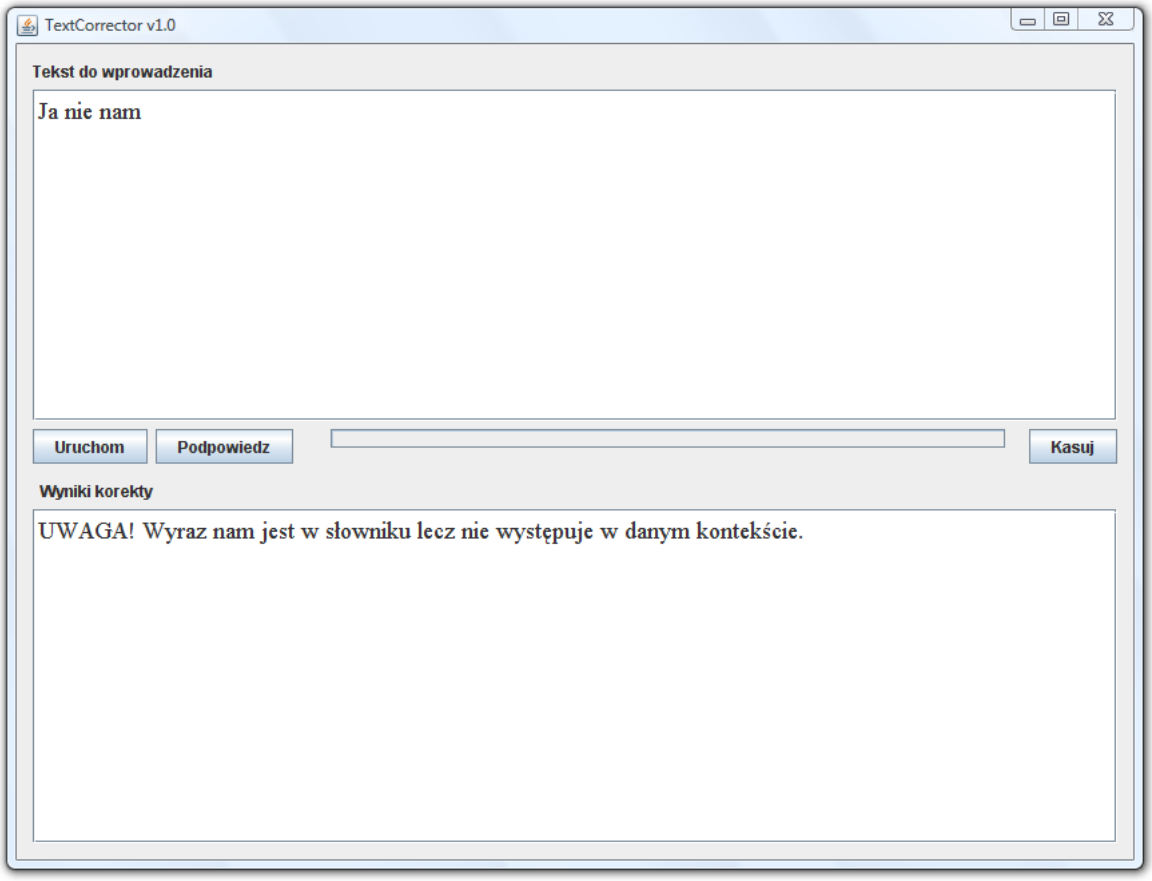
cyjnej bazie danych nie ma wyrazu ”Mazowsze”. Stąd, gdy użytkownik go wpisze, zostanie mu wyświetlony komunikat informujący, iż wyraz nie występuje w słowniku. Rozwiązaniem takiego problemu, może być poszerzenie słownika o kolejne wyrazy, które występują w języku polskim. Idealną sytuacją byłoby zawarcie w słowniku wszystkich wyrazów języka polskiego.

2. Gdy wprowadzony wyraz nie występuje w odpowiednim kontekście. Algorytm do sprawdzania kontekstu bazuje na informacjach zebranych przez pająka internetowego. Zawartość bazy jest ograniczona, przez znalezione słowa-trójki. Możliwe jest więc wystąpienie wyrazów, które są prawidłowe, lecz nie zostały znalezione przez pająka na stronach internetowych. Rozwiązaniem jest poszerzenie bazy słów-trójek o kolejne wyrazy.

Sądzę jednak, że zastosowany algorytm do sprawdzania kontekstu słownego, daje i tak dużo lepsze wyniki, niż występujące mechanizmy w testowanych programach.



Rysunek 5.9: Zdanie ”Ja nie nam co jeść.” i brak sygnalizacji błędu w programie Writer 3.0



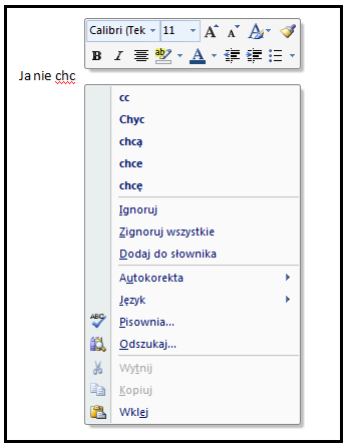
Rysunek 5.10: Początek zdania ”Ja nie nam” i sygnalizacja zauważonego błędu przez stworzony algorytm

5.4.2. Podpowiadania aktualnie wpisywanego słowa

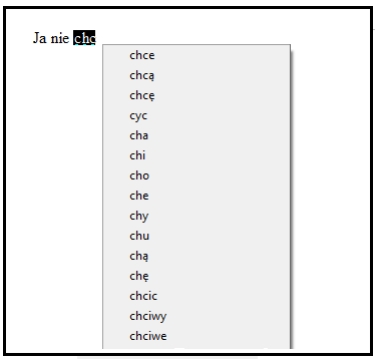
Drugim mechanizmem, jaki zostanie porównany, to dopełnianie aktualnie wprowadzonego wyrazu. Program Writer 3.0 oferuje taką możliwość. *”Writer zawiera ciekawą opcję podpowiedzi słów używanych w*

danym dokumencie. Gdy program znajduje słowo, które pasuje do wpisywanych początkowych liter, podpowiada je, wyświetlając w negatywie. Wystarczy nacisnąć Enter, by zaakceptować propozycję." [8]. Niestety automatyczne podpowiadanie wprowadzanego wyrazu, występuje bardzo rzadko. Użytkownik, jeśli chce uzyskać informację o wprowadzanym wyrazie (w programach takich jak Microsoft Word 2007 oraz Writer 3.0), musi po wprowadzeniu jego początku kliknąć na nim prawym przyciskiem myszy i z pojawiających się wyrazów podpowiedzi wybrać pasujący wyraz. Nie jest to ściśle dokończeniem wyrazu, lecz znajdywaniem wyrazu najbliżego wprowadzanemu. Stworzony w programie algorytm dysponuje mechanizmem automatycznej podpowiedzi dla wprowadzonego początku wyrazu. Dodatkowo sprawdza on kontekst początku wyrazu, więc prezentowane wyrazy nie tylko rozpoczynają się od wprowadzonych liter, ale również są poprawne w określonym kontekście. W celu uzyskania informacji o możliwych wyrazach po wprowadzeniu początku słowa, użytkownik powinien nacisnąć przycisk "Podpowiedz". Zostanie wtedy uruchomiony mechanizm, który wyszuka pasujące wyrazy i "podpowie" je, wyświetlając na ekranie.

Poniżej zostaną zaprezentowane różne możliwości dokończenia wyrazu, dla początku zdania *Ja nie chc*.



Rysunek 5.11: Początek zdania "Ja nie chc" i możliwe podpowiedzi w programie Microsoft Word 2007

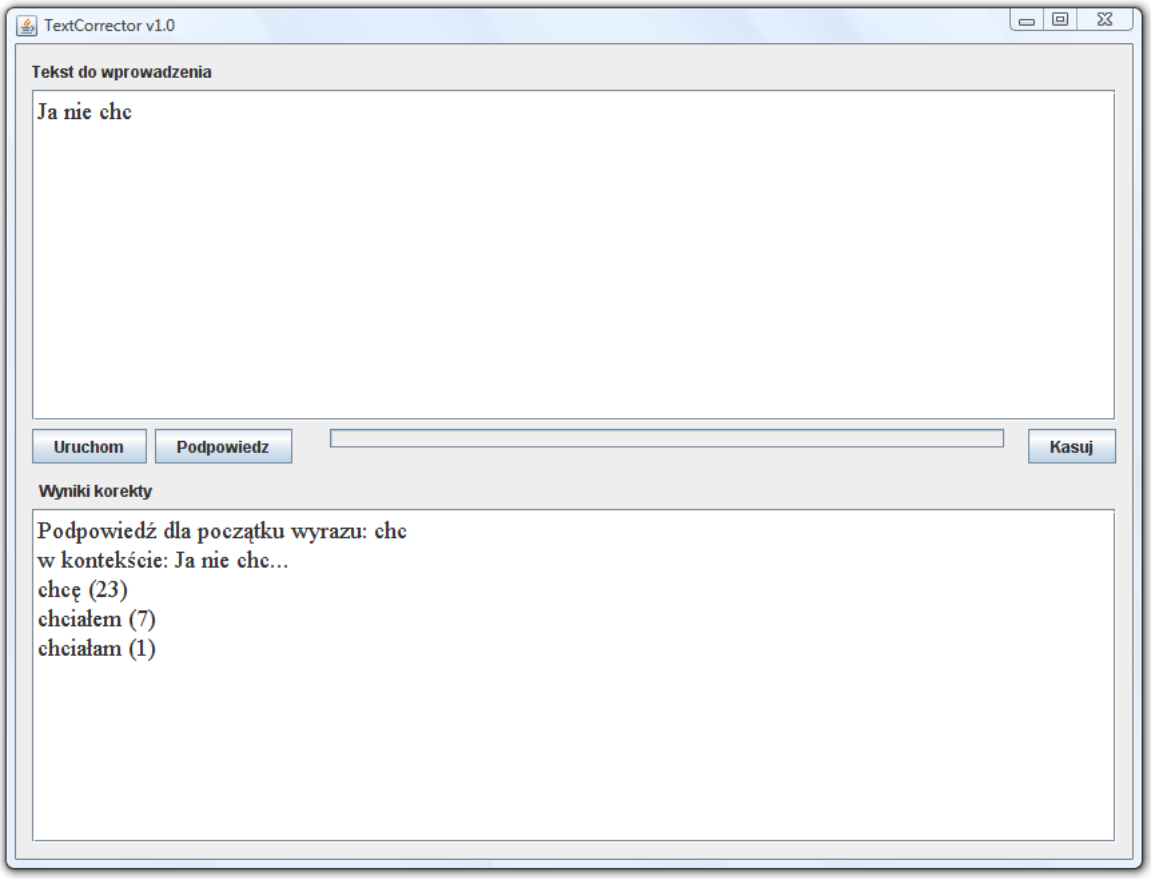


Rysunek 5.12: Początek zdania "Ja nie chc" i możliwe podpowiedzi w programie Writer 3.0

Dla programu Microsoft Word 2007 dla początkowego zdania *Ja nie chc* ograniczając wyniki do wyrazów rozpoczynających się od *chc* uzyskano następujące propozycje:

- chcą,
- chce,
- chcę.

Dla programu Writer 3.0 dla początkowego zdania *Ja nie chc* ograniczając wyniki do wyrazów rozpoczynających się od *chc* uzyskano następujące propozycje:



Rysunek 5.13: Początek zdania ”Ja nie chc” i możliwe podpowiedzi znalezione przez algorytm

- chce,
- chcą,
- chcę,
- chcic,
- chciwy,
- chciwe.

W opracowanym mechanizmie, dla początkowego zdania *Ja nie chc* uzyskano następujące propozycje, z dodatkową informacją o częstości występowania w zadanym kontekście:

- chcę (wystąpiło 23 razy),
- chciałem (wystąpiło 7 razy),
- chciałam (wystąpiło 1 raz).

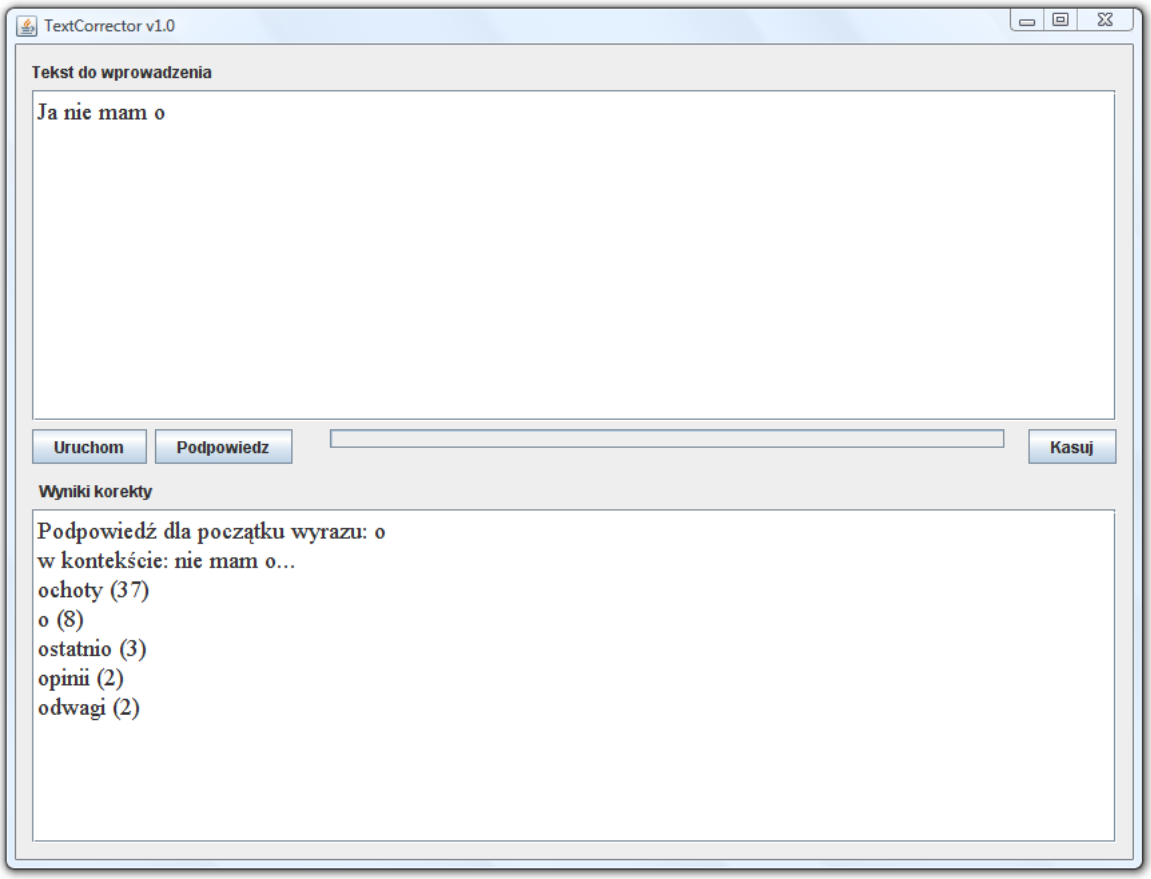
Należy zwrócić uwagę na fakt, że algorytm zaprezentował tylko wyrazy rozpoczynające się od liter *ch*. Jak można zauważyć, zostały zasugerowane trzy wyrazy, z których każdy pasuje kontekstowo do początku zdania *Ja nie chc*. Trochę gorzej poradził sobie mechanizm w programie Microsoft Word 2007, który również zasugerował trzy wyrazy, lecz jedynie dwa z nich mogą wystąpić w danym kontekście. Zastanawiające może być pojawienie się wyrazu *chce*. Wyraz ten może bowiem wystąpić w danym kontekście jedynie w tekstach nieformalnych. Wyraz *chcą* choć jest poprawny gramatycznie, nie może wystąpić w danym kontekście. Najgorzej z zaproponowaniem wyrazu poradził sobie program Writer 3.0. Wygenerował on najwięcej podpowiedzi, lecz podobnie jak w programie Word, jedynie wyrazy *chcę* oraz *chce*, mogą zostać

uznane za prawidłowe. Reszta z nich nie może być użyta w danym kontekście. Problematyczne może być wystąpienie wyrazu *chcic*, który nie występuje nawet w słowniku języka polskiego.

W kolejnym przykładzie, który został zaprezentowany na rysunku 5.14, zostały zaprezentowane podpowiedzi dla początku zdania *ja nie mam o*. Uzyskane możliwości przez mechanizm, bazujący na grafie LHG to:

- ochoty (37 razy),
- o (8 razy),
- ostatnio (3 razy),
- opinii (2 razy),
- odwagi (2 razy).

Kłopotliwym może wydawać się pojawienie, jako kolejnego słowa, jedynie litery ”o”. Można jednak stwierdzić, że nie jest to błąd. Istnieje bowiem możliwość utworzenia zdania, np. *Ja nie mam o czym marzyć.* lub też *Ja nie mam o tym zielonego pojęcia.*



Rysunek 5.14: Początek zdania ”Ja nie mam o” i uzyskane możliwe podpowiedzi

Podobnie jak z automatycznym sprawdzaniem wpisywanych wyrazów również i z podpowiadaniem aktualnie wpisywanego wyrazu opracowany program, bazujący na zaimplementowanym algorytmie oraz grafie LHG, radzi sobie znacznie lepiej, niż programy Word, czy Writer. W programach tych nie ma gotowych modułów służących do tego celu, a z mojego punktu widzenia byłyby one bardzo przydatne. Stworzony mechanizm dodatkowo analizuje wpisywany początek zdania i proponuje użytkownikowi jedynie te wyrazy, które występują w określonym kontekście wyrazów stojących bezpośrednio przed nim. Dla dłuższego zdania może się zdarzyć, że zaproponowane podpowiedzi nie będą pasowały kontekstowo

do całego zdania. Sprawdzany jest bowiem kontekst trzech najbliższych wyrazów dla wpisywanego słowa. Program wyświetla wyniki właśnie dla takiego kontekstu. Analiza kontekstu całego zdania, wymagałaby zapisywania znacznie szerszego kontekstu wypowiedzi w bazie, co niewątpliwie miało by znaczenie m.in. dla szybkości działania całej aplikacji. Stąd została zawarta jedynie informacje o kontekście dla trzech najbliższych słów.

5.5. Porównanie automatycznej kontekstowej korekty testów

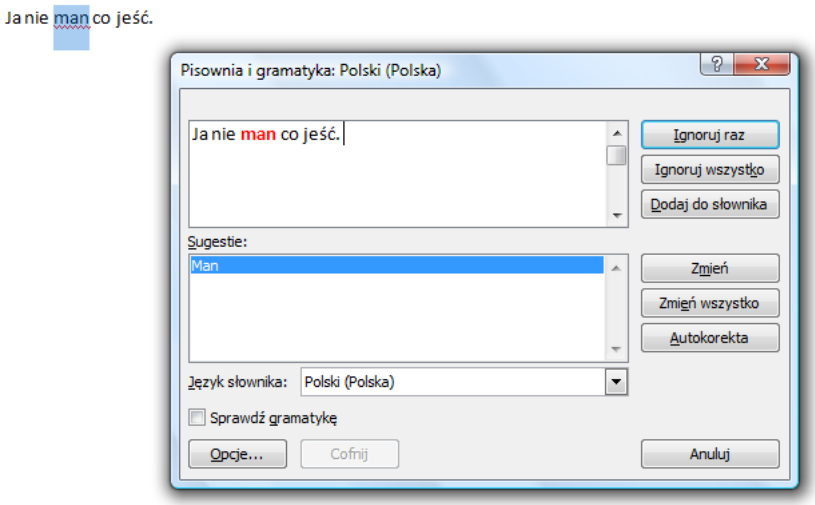
Jak już zostało wspomniane, koniec XX wieku oraz początek XXI to przede wszystkim czas w którym ogromną rolę w życiu człowieka zaczęła pełnić technologia. Ma ona na celu ułatwić codzienną pracę człowiekowi. Ludzie coraz częściej zamieniają formę komunikacji z tradycyjnej na elektroniczną. Coraz częściej, w celu skontaktowania się z drugą osobą, sięga się po telefon komórkowy lub komputer. W ostatnich latach coraz większą popularnością cieszą się komunikatory internetowe, takie jak Skype oraz (szczególnie popularne w Polsce) Gadu-Gadu. Prawie każdy z nas ma telefon komórkowy i założoną własną skrzynkę pocztową. Nie wyobrażamy sobie niejednokrotnie dnia, bez wysłania e-maila, bądź napisania SMSa do znajomego. Niestety z rozwojem technologii związane coraz mniejsza dbałość o formę przekazu. Coraz częściej, w komunikacji elektronicznej, zanikają w zdaniach znaki specjalne (takie jak, np. przecinki w zdaniu) lub też ”ogonki” w polskich znakach (zamiana litery ”ą” na ”a” lub też ”ę” na ”e”). Wystarczy zwrócić uwagę na ilość popełnianych błędów ortograficznych, zamian liter w słowie, czy skrótów wyrazów.

Powstaje więc podstawowa trudność - jak zrozumieć i zinterpretować przekaz drugiej osoby? Człowiek w dość prosty sposób może dokonać pewnych przekształceń i ”niejako odszyfrować” treść nadaną przez drugiego człowieka. Niestety, zrozumienie tego samego tekstu przez komputer okazuje się bardzo trudnym i złożonym zadaniem. Stąd też, moja praca polegała na skonstruowaniu całego mechanizmu do automatycznej korekty, wpisanego przez użytkownika tekstu, który zawiera różnorakie błędy. Niewątpliwie najważniejszym zadaniem było opracowanie specjalnego algorytmu do analizy już wprowadzonego tekstu. Algorytm ten miał za zadanie tak zmienić wpisany pierwotnie tekst, aby usunąć z niego wszystkie możliwe błędy zachowując kontekst zdania.

5.5.1. Automatyczna korekta tekstów

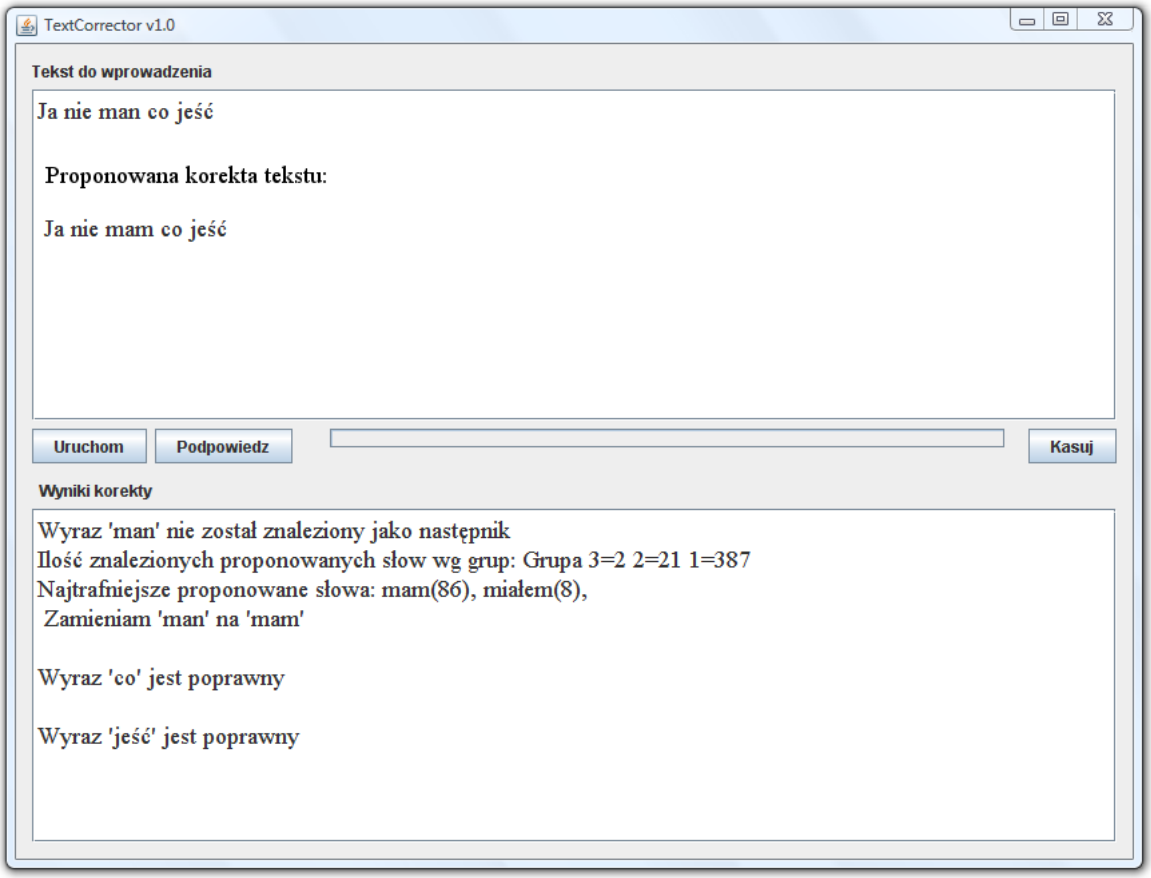
W tej części pracy zostanie zaprezentowanych kilka przykładów możliwości stworzonego algorytmu w automatycznej kontekstowej korekcji tekstu. Otrzymane wyniki zostaną porównane z możliwościami poprawy tekstu, jakie oferują programy Microsoft Word 2007 oraz Writer 3.0.

W pierwszym teście zostaną wykorzystane już wcześniej omawiane zdania *Ja nie man co jeść.* oraz *Ja nie nam co jeść.* W pierwszym z nich jedynie Microsoft Word wykrył błędne użycie słowa *man*. Proponowana korekcja tego słowa została przedstawiona na rysunku 5.15



Rysunek 5.15: Proponowana korekcja zdania ”Ja nie man co jeść.” przez program Microsoft Word 2007

Program Word 2007 zasygnalizował błąd i umieścił go w dobrym miejscu, lecz proponowana poprawa nie jest możliwa do zaakceptowania przez użytkownika. Jedyną propozycją zamiany była zmiana słowa *man* na wyraz *Man*. Korekcja tego samego zdania, z wykorzystaniem zaimplementowanego algorytmu, została przedstawiona na rysunku 5.16.



Rysunek 5.16: Proponowana korekcja zdania ”Ja nie man co jeść.” przez zaimplementowany algorytm

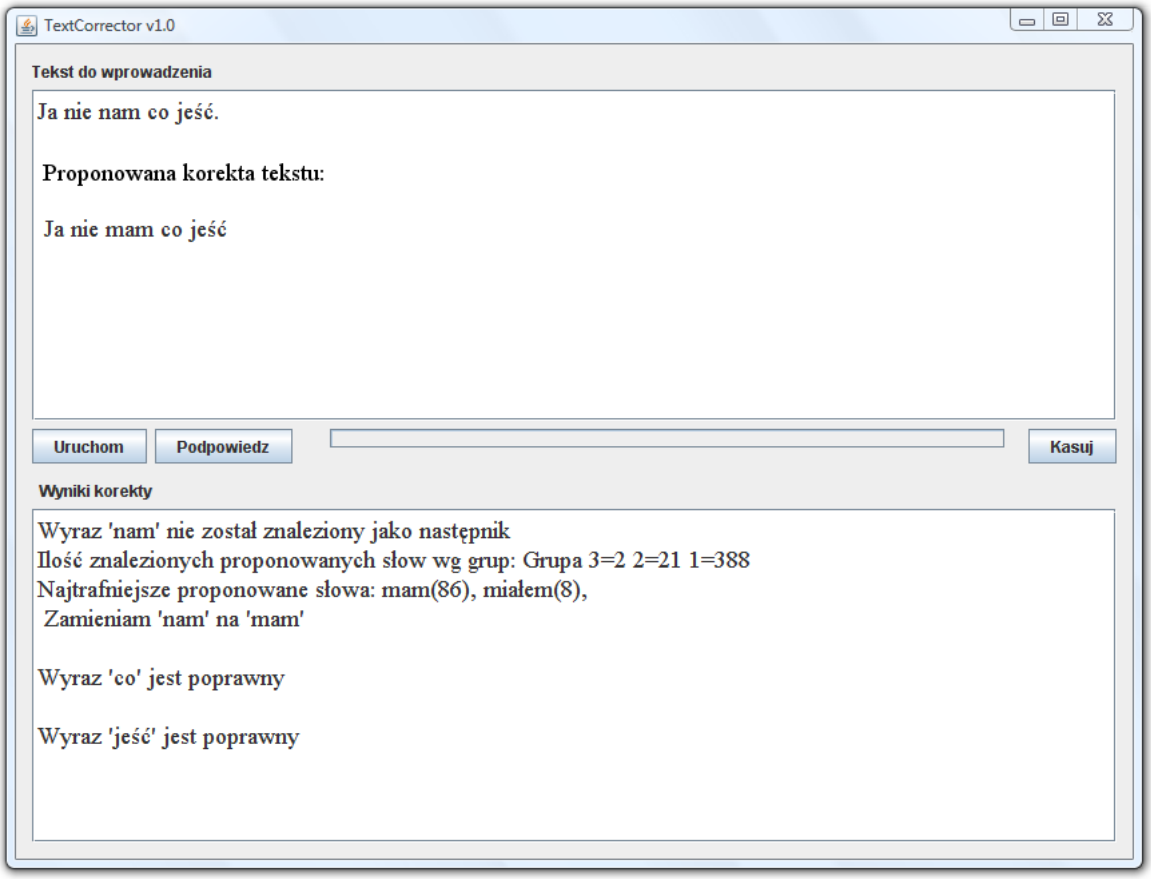
Z powyższych rysunków można stwierdzić, że algorytm, bazując na grafie LHG, dokonał dużo lepszej poprawy tekstu. Wyjściowe zdanie *Ja nie man co jeść.* zostało przekształcone w zdanie *Ja nie mam co jeść.* W szczegółach pracy mechanizmu znalazła się informacja, że znaleziono dwa słowa, które są najbardziej prawdopodobne do zamiany z błędnym słowem. Są to:

- mam, który wystąpił w określonym kontekście aż 86 razy,
- miałem, które wystąpiło w danym kontekście 8 razy.

Ponadto została zaprezentowana również informacja, że oprócz wspomnianych dwóch słów, występuje jeszcze 21 innych wyrazów, które należą do dwóch z trzech możliwych kontekstów dla danego zdania oraz aż 387 słów, które występują tylko w jednym z trzech kontekstów. Druga propozycja, zamiany na słowo *miałem*, również może zostać użyta w podanym zdaniu i dalej będzie to zdanie prawidłowe w języku polskim. Przy wyborze jednego wyrazu jako ”najbardziej prawdopodobnego” algorytm bazuje na:

- ilości kontekstów, w jakich może wystąpić wyraz,
- częstotliwości występowania wyrazu w proponowanych kontekstach,
- odległości edycyjnej słowa od wyrazu uznanego za błędy,
- porównaniu długości obydwu słów.

Drugie zdanie, jakie zostanie poddane automatycznej kontekstowej korekcji tekstów, brzmi: *Ja nie nam co jeść.* Jak zostało już to udowodnione w rozdziale 5.4.1 zarówno programy Word, jak i Writer nie wykryły w nim błędu. Proponowana korekcja tego zdania przez zastosowany mechanizm (zaprezentowana na rysunku 5.17), również przekształciła zdanie wejściowe *Ja nie nam co jeść.* na zdanie *Ja nie mam co jeść.*



Rysunek 5.17: Proponowana korekcja zdania ”Ja nie nam co jeść.” przez zaimplementowany algorytm

Zastosowany algorytm potrafi znaleźć i poprawnie zamienić słowa, w których występują błędy ortograficzne, jak również zamienić kilka wyrazów zawierających błędy w jednym zdaniu. Przykładem takiego zdania, dla którego zostało przeprowadzone badanie, jest: *Ja nie chcieć im pomuc*. Zdaniem poprawnym powinno być wyrażenie *Ja nie chcę im pomóc*. W zdaniu, poddanemu procesowi korekcji, występują dwa błędy:

- zmiana wyrazu chcę na chcieć,
- zmiana wyrazu pomóc na pomuc.

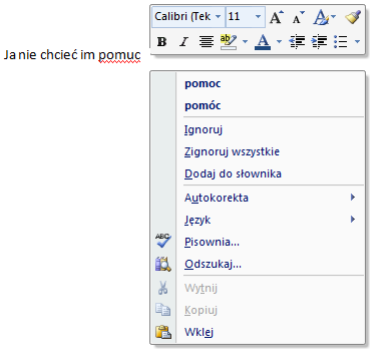
Na rysunku 5.18 została przedstawiona korekta, przeprowadzona przez program Word, a na rysunku 5.19 korekta przez program Writer. Natomiast na rysunku 5.20 zaprezentowana została korekta, uzyskana przez stworzony mechanizm.

Dla programu Word oraz Writer wystąpiły te same błędy:

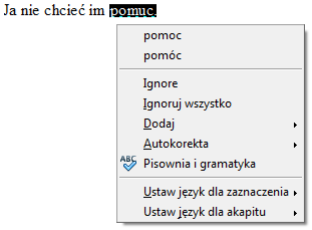
- Wyraz *chcieć* przez żaden z programów nie został rozpoznany jako błędny. Można podejrzewać, że zadziałał tu ten sam mechanizm, który został opisany wcześniej. Słowo *chcieć* występuje w języku polskim, wiec nie zostało zaznaczone jako błędne.
- Wyraz *pomuc* został rozpoznany jako błędny. Propozycje jego zmiany były jedynie dwie: *pomóc* oraz *pomoc*. Niestety tylko jedna z tych form pasuje do kontekstu całej wypowiedzi.

Zastosowany algorytm do automatycznej kontekstowej korekty tekstu wykrył natomiast dwa błędy:

- błędnie użyte w kontekście słowo *chcieć*,
- błędne ortograficznie słowo *pomuc*.



Rysunek 5.18: Proponowana korekcja zdania ”Ja nie chcieć im pomuc” przez program Microsoft Word 2007



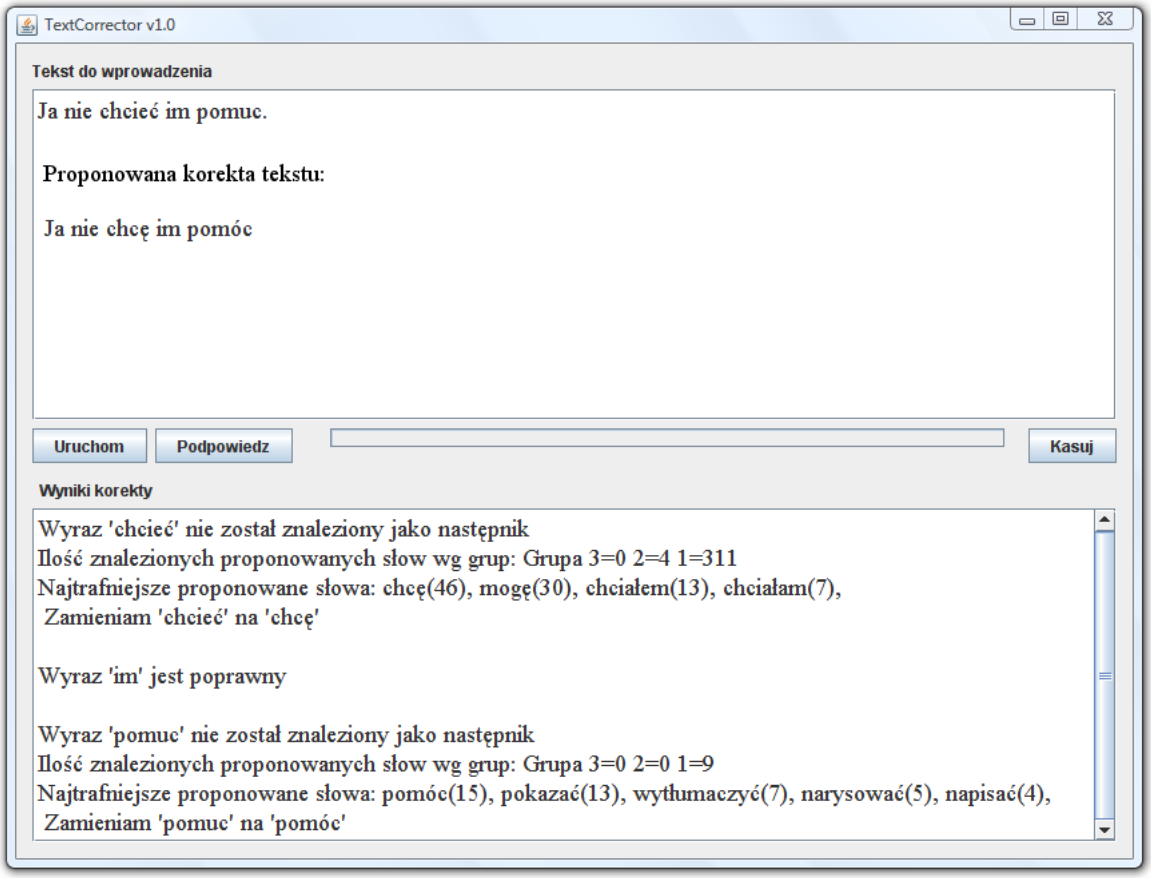
Rysunek 5.19: Proponowana korekcja zdania ”Ja nie chcieć im pomuc” przez program Writer 3.0

Po przeprowadzonej analizie zostały zaprezentowane następujące propozycje:

- Zamiana pierwszego słowa *chcieć* odpowiednio na słowa:
  1. chcę, które wystąpiło w danym kontekście 46 razy,
  2. mogę, które wystąpiło w danym kontekście 30 razy,
  3. chciałem, które wystąpiło w danym kontekście 13 razy,
  4. chciałam, które wystąpiło w danym kontekście 7 razy.
- Zamiana drugiego błędnego słowa *pomuc* odpowiednio na słowa:
  1. pomóc, które wystąpiło w danym kontekście 15 razy,
  2. pokazać, które wystąpiło w danym kontekście 13 razy,
  3. wytłumaczyć, które wystąpiło w danym kontekście 7 razy,
  4. narysować, które wystąpiło w danym kontekście 5 razy,
  5. napisać, które wystąpiło w danym kontekście 4 razy.

Jak można stwierdzić, wszystkie wymienione propozycje kontekstowo pasują do wprowadzonego pierwotnie zdania. Tak więc, po raz kolejny zostało wykazane, że zastosowany algorytm, który wykorzystuje kilka unikatowych sposobów korekty tekstu, poprawia wprowadzone błędnie zdania z bardzo dobrym rezultatem. Swoim działaniem ukazuje, jak bardzo inne algorytmy, użyte w profesjonalnych programach Microsoft Word 2007 oraz Open Office Writer 3.0, są dalekie od ideału.

W kolejnych przykładach została zaprezentowana reakcja systemu korekty, na kilka różnych błędów oraz przedstawione zostały propozycje korekty zdań błędnych. Szczegółowy opis algorytmu automatycznej kontekstowej korekty tekstu został zaprezentowany w rozdziale 4.4.



Rysunek 5.20: Proponowana korekcja zdania ”Ja nie chcieć im pomuc” przez stworzony mechanizm

W przykładzie 5.21 została zaprezentowana reakcja systemu na błędnie wprowadzone zdanie: *Telefon komórkowy nie jets już tyko narzędzie przyspieszającym mam kontakt z innimi.*. W wyrażeniu tym występują następujące błędy:

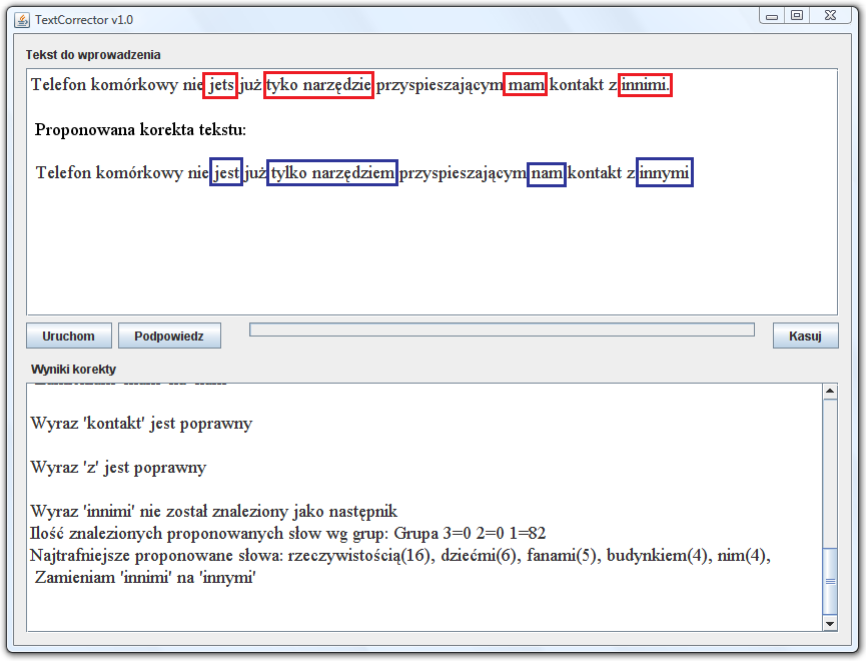
1. Zamiana wyrazu *jest* na *jets*.
2. Zamiana wyrazu *tylko* na *tyko*.
3. Zamiana wyrazu *narzędziem* na *narzędzie*.
4. Zamiana wyrazu *nam* na *mam*.
5. Zamiana wyrazu *innymi* na *innimi*.

W przykładzie 5.22 została zaprezentowana reakcja systemu na błędnie wprowadzone zdanie: *Czy obecna matóra sprawdza zdobytej przez uczniów wiedza?*. W wyrażeniu tym występują następujące błędy:

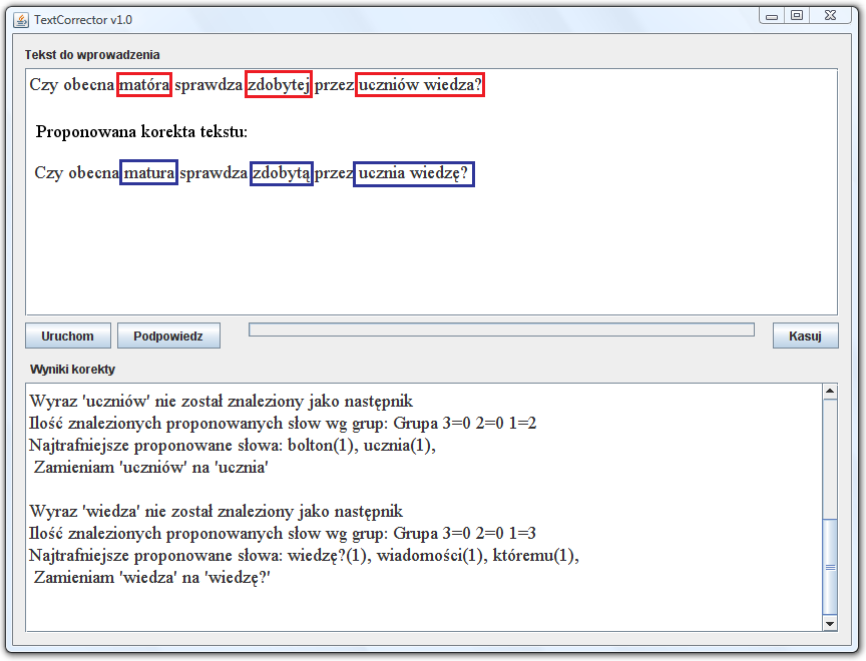
1. Zamiana wyrazu *matura* na *matóra*.
2. Zamiana wyrazu *zdobytą* na *zdobytej*.
3. Zamiana wyrazu *wiedzę?* na *wiedza?*.

W przykładzie 5.23 została zaprezentowana reakcja systemu na błędnie wprowadzone zdanie: *Maj zapowiada się barbo spokojne*. W wyrażeniu tym występują następujące błędy:

1. Zamiana wyrazu *bardzo* na *barbo*.
2. Zamiana wyrazu *spokojnie* na *spokojne*.



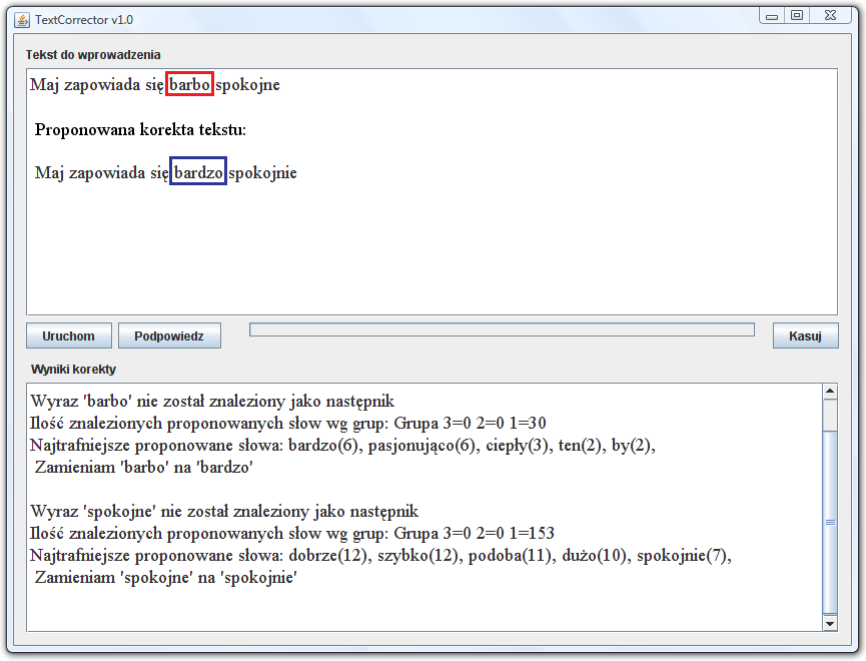
Rysunek 5.21: Proponowana korekcja zdania ”Telefon komórkowy nie jets już tyko narzędzie przyspieszającym mam kontakt z innimi.” przez stworzony mechanizm



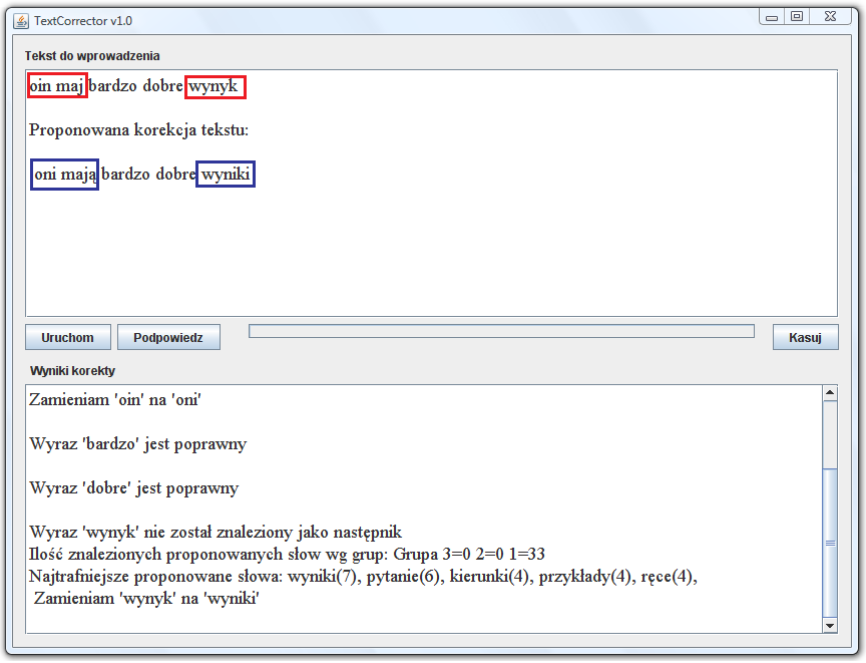
Rysunek 5.22: Proponowana korekcja zdania ”Czy obecna matóra sprawdza zdobytej przez uczniów wiedza?” przez stworzony mechanizm

W przykładzie 5.24 została zaprezentowana reakcja systemu na błędnie wprowadzone zdanie: *oin maj bardzo dobre wynyk*. W wyrażeniu tym występują następujące błędy:

1. Zamiana wyrazu *oni* na *oin*.
2. Zamiana wyrazu *mają* na *maj*.
3. Zamiana wyrazu *wynik* na *wyniki*.



Rysunek 5.23: Proponowana korekcja zdania ”Maj zapowiada się barbo spokojne” przez stworzony mechanizm

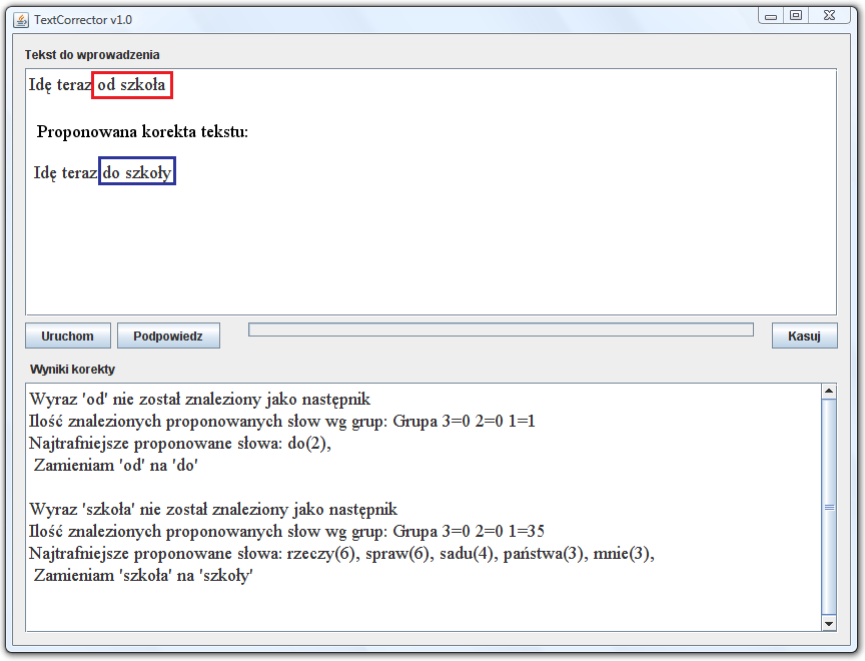


Rysunek 5.24: Proponowana korekcja zdania ”oni maj bardzo dobre wynyk” przez stworzony mechanizm

W przykładzie 5.25 została zaprezentowana reakcja systemu na błędnie wprowadzone zdanie: *Idę teraz od szkoła*. W wyrażeniu tym występują następujące błędy:

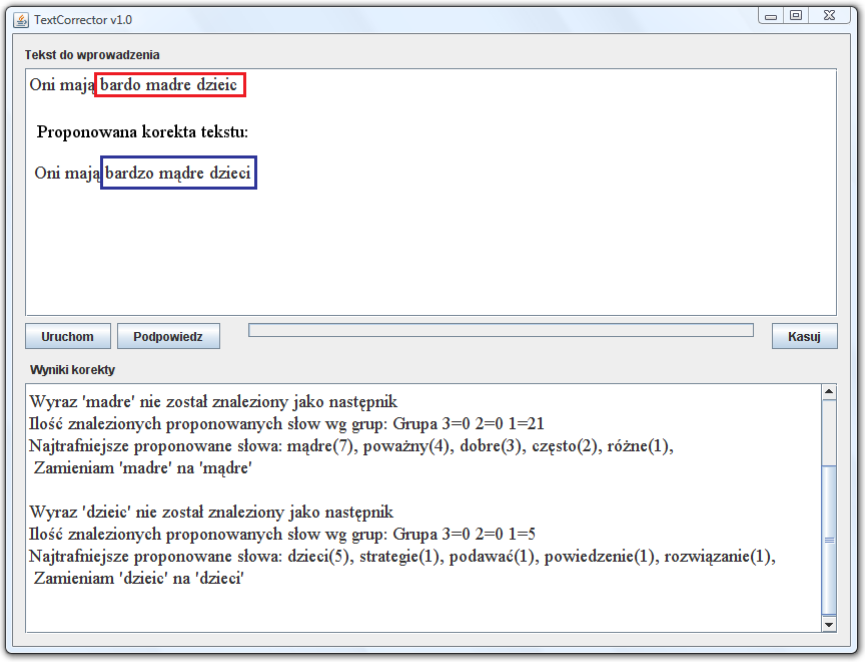
1. Zamiana wyrazu *od* na *do*.
2. Zamiana wyrazu *szkoła* na *szkoły*.

W przykładzie 5.26 została zaprezentowana reakcja systemu na błędnie wprowadzone zdanie: *Oni mają bardo madre dzieic*. W wyrażeniu tym występują następujące błędy:



Rysunek 5.25: Proponowana korekcja zdania ”Idę teraz od szkoła” przez stworzony mechanizm

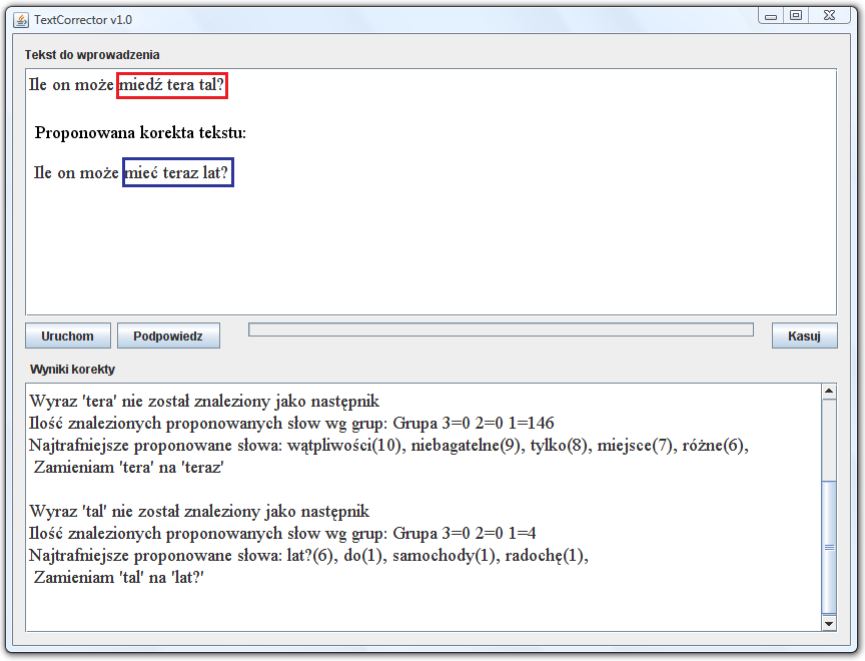
1. Zamiana wyrazu *bardzo* na *bardo*.
2. Zamiana wyrazu *mądre* na *madre*.
3. Zamiana wyrazu *dzieci* na *dzieic*.



Rysunek 5.26: Proponowana korekcja zdania ”Oni mają bardo madre dzieic” przez stworzony mechanizm

W niektórych wypadkach automatyczna kontekstowa korekta tekstów w stworzonej aplikacji, działa gorzej niż we wspomnianych wcześniej programach. Dzieje się tak zwykle, gdy w zdaniu występuje bardzo

duża ilość błędów, lub w jednym wyrazie dokonano tylu błędów, że wśród propozycji słów do korekcji znajdują się słowa w mniejszej odległości edycyjnej. Algorytm wtedy może dokonać błędnej zamiany danego wyrazu. Po takiej zamianie, do dalszej poprawy brane jest już zmienione słowo jako prawidłowe. Do ograniczenia występowania tego typu błędów zastosowano w algorytmie sprawdzanie trzech kontekstów - w szczególności kontekst dwóch poprzedników oraz dwóch następników. W wyniku wspomnianych wcześniej błędów generowane ostatecznie poprawne zdanie, może znacząco odbiegać od zdania wprowadzonego po korekcie człowieka. Zaproponowane zdanie, choć błędne w ujęciu poprawy drobnego błędu użytkownika, może być poprawne pod względem kontekstu. Przykładem takiej pracy algorytmu, może być popełnienie przez użytkownika, błędu w zdaniu *Ile on może mieć teraz lat?* Gdy użytkownik wprowadzi zamiast niego zdanie: *Ile on może miedź tera tal?*, system wygeneruje jako poprawne zdanie: *Ile on może mieć teraz lat?*. Natomiast, gdy wprowadzone zostanie zdanie: *Ile on może miedź ter lat?*, system zaproponuje zdanie: *Ile on może mieć też na*. Stało się tak dlatego, że choć zarówno w propozycjach wystąpiły wyrazy ”teraz” oraz ”też”, to słowo ”też” jest edycyjnie bliższe, błędnie wprowadzonemu, słowu ”ter”. Szczegóły pracy algorytmu, dla tych dwóch zdań zostały zaprezentowane na rysunkach 5.27 oraz 5.28.



Rysunek 5.27: Proponowana korekcja zdania ”Ile on może miedź tera tal?” przez stworzony mechanizm

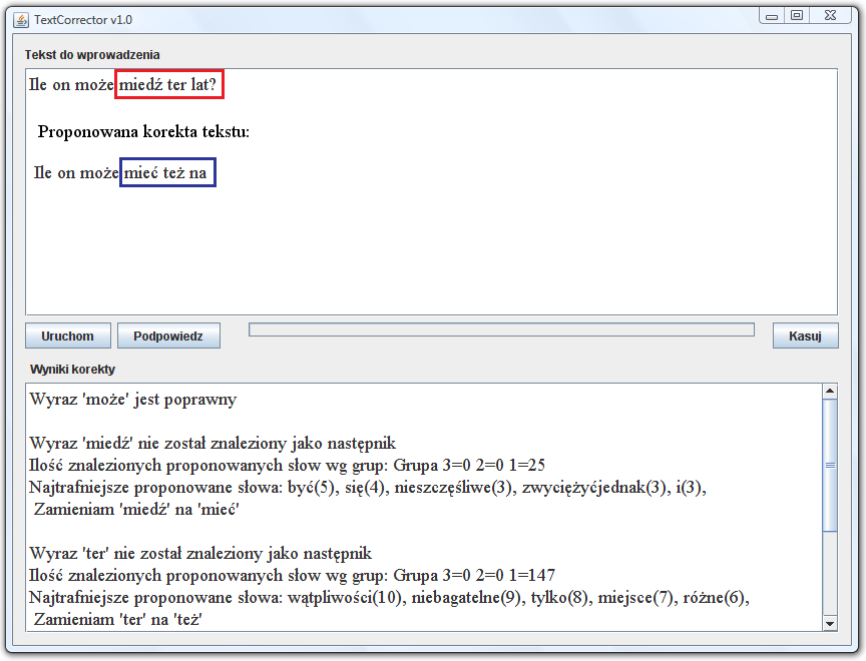
5.5.2. Porównanie automatycznej korekty tekstu dla różnej wielkości bazy danych

Do osiągania większej skuteczności, w procesie automatycznej korekty tekstu, może przyczynić się powiększenie bazy słów, o kolejne słowa-trójki. W tej części pracy zostało zaprezentowanych kilka przykładów dotyczących wpływu wielkości bazy danych na etap poprawy tekstów. W tym celu wykorzystano:

- testową bazę danych, która zawiera 8 375 504 słów-trójek,
- rozbudowaną bazę danych, która zawiera 30 636 067 słów-trójek.

Porównanie jakości poprawy tekstów, można omówić analizując pojawiające się słowa-następniki oraz ich częstość dla różnych początków kontekstu.

1. Najczęściej występujące następniki, wraz z ich częstością, dla kontekstu *nie ma*, dla różnej wielkości baz danych, zostały przestawione w tabeli 5.1.
2. Najczęściej występujące następniki, wraz z ich częstością, dla kontekstu *ja nie*, dla różnej wielkości baz danych, zostały przestawione w tabeli 5.2.



Rysunek 5.28: Proponowana korekcja zdania ”Ile on może miedź ter lat?” przez stworzony mechanizm

Tablica 5.1: Możliwe następniki dla kontekstu *nie ma*

baza rozbudowana	baza testowa
co - 1293	ale - 329
w - 870	nic - 281
nic - 810	co - 266
ale - 636	w - 252
już - 584	się - 191

Tablica 5.2: Możliwe następniki dla kontekstu *ja nie*

baza rozbudowana	baza testowa
chcę - 251	jestem - 79
mam - 191	chcę - 77
wiem - 158	mam - 74
jestem - 155	wiem - 65
mogę - 81	cierpię - 30

3. Najczęściej występujące następniki, wraz z ich częstością, dla kontekstu *nie mam*, dla różnej wielkości baz danych, zostały przestawione w tabeli 5.3.

Tablica 5.3: Możliwe następniki dla kontekstu *nie mam*

baza rozbudowana	baza testowa
nic - 219	nic - 86
pojęcia - 193	ani - 84
czasu - 162	na - 80
na - 160	pojęcia - 74
zamiaru - 144	zamiaru - 59

4. Najczęściej występujące następniki, wraz z ich częstością, dla kontekstu *iść do*, dla różnej wielkości baz danych, zostały przestawione w tabeli 5.4.

Tablica 5.4: Możliwe następniki dla kontekstu *iść do*

baza rozbudowana	baza testowa
pracy - 25	pracy - 20
kina - 16	szkoły - 10
lekarza - 14	domu - 6
szkoły - 14	przodu - 5
domu - 11	lekarza - 4

5. Najczęściej występujące następniki, wraz z ich częstością, dla kontekstu *jest bardzo*, dla różnej wielkości baz danych, zostały przestawione w tabeli 5.5.

Tablica 5.5: Możliwe następniki dla kontekstu *jest bardzo*

baza rozbudowana	baza testowa
mało - 53	korzystne - 31
prosta - 52	popularny - 17
dużo - 51	dużo - 12
prosty - 49	prosty - 11
ważne - 46	istotne - 10

Jak można zauważyć, zarówno dla bazy rozbudowanej jak i testowej, występują te same następniki. Różnica, jaka występuje między tymi bazami, to częstość pojawiania się określonych słów w kontekście, tym samym inny jest ranking najlepszych słów-następników. Rozbudowana baza jest ok. 2,5 razy większa niż baza testowa. Podobnie zwiększona jest częstotliwość występowania tych samych słów w danym kontekście. Podczas przeprowadzania testów nie zdarzyła się sytuacja, aby któryś z następników występujący w bazie testowej, o częstotliwości większej niż 5, nie pojawił się w rozbudowanej bazie.

Pająk internetowy pobiera informacje o kontekście słownym z losowych stron, stąd w obydwu bazach wystąpiły te same słowa jako następniki. Różnica, jaką można zauważyć analizując dane zebrane w dwóch bazach, to występowanie wielu nowych możliwych następników dla większej bazy. Przykładowo jeśli w testowej bazie danych jest ok. 320 możliwych wszystkich następników dla jakiegoś kontekstu, to w rozbudowanej bazie możliwych kolejnych słów jest ok 800, dla tego samego kontekstu.

Bardzo ciekawa sytuacja występuje też, np. dla możliwych następników wyrazów *jest bardzo*. Zaproponowano m.in. następniki *prosto* oraz *prosta*, które mają podobną częstość występowania, ponieważ znaczeniowo są takie same. Pierwszy z wyrazów ma jedynie rodzaj męski, a drugi żeński. Kolejne słowa o podobnej częstości to *mało* oraz *dużo*, które są sobie przeciwne.

6. Wnioski z porównania automatycznej kontekstowej korekty tekstu

Jednym z największych wyzwań obecnych czasów jest próba przetworzenia i zrozumienia dostępnych informacji, pochodzących z różnych źródeł, przez maszyny. Występuje więc znaczne zapotrzebowanie na programy, które będą w stanie przetworzyć i skorygować wprowadzany przez użytkownika tekst. Ich zadaniem jest dokonanie takich operacji, aby nie zmieniając sensu całego tekstu, przetworzyć go tak, aby był poprawny w danym języku naturalnym oraz zrozumiały dla drugiego człowieka. Niestety w Polsce nie ma jeszcze tak wyspecjalizowanych programów, które mogłyby to umożliwić. W pracy tej został położony szczególny nacisk na stworzenie takiego właśnie algorytm oraz dodatkowo na porównanie jego działania z możliwościami korekty tekstu przez dwa wiodące edytory tekstu. Zaproponowany innowacyjny algorytm poprawy tekstu bazuje na zaimplementowanym Grafie Przyzwyczajeń Lingwistycznych. Jak zostało wykazane w rozdziale 5 poświęconym testom, skonstruowany mechanizm kontekstowej korekty, okazał się być znacznie bardziej skuteczny w rozpoznawaniu miejsc, w których wystąpiły błędy oraz w ich korekcie, od konkurencji.

W stworzonej aplikacji można wyróżnić dwa główne programy:

1. Pająk internetowy.

   Przegląda on możliwe strony internetowe w poszukiwaniu linków do dalszych stron oraz indeksuje i zapisuje w bazie danych wyszukane zdania w języku polskim. Robot, dzięki zastosowaniu bazy danych, może działać z przerwami - stąd możliwe jest jego czasowe wyłączenie i ponowne włączenie. Dzięki temu zarówno baza słów może być nieustannie poszerzana o kolejne słowa w ich unikalnym kontekście oraz LHG może być powiększany.

2. Program do automatycznej kontekstowej korekcji tekstu.

   Jest to główny program, służący do poprawy błędnie wprowadzonego tekstu. Korekcja tekstu możliwa jest dzięki wykorzystaniu Grafu Przyzwyczajeń Lingwistycznych dla języka polskiego.
   Program ten składa się z trzech modułów

   (a) Modułu do tworzenia i przeglądania Grafu Przyzwyczajeń Lingwistycznych.

   (b) Modułu do automatycznego sprawdzania poprawności wprowadzonych wyrazów.

   (c) Modułu do ”podpowiadania” końca słowa, jak i do korekcji całego wprowadzonego tekstu.

Do zalet wyróżniających zaproponowany algorytm pośród innych, należą:

1. Moduł do tworzenia nowych zdań.

2. Innowacyjny sposób wykrywania potencjalnych błędów w tekście.

3. Inteligentne dopełnianie wyrazów.

4. Automatyczna kontekstowa korekta tekstów.

Największą niedogodnością, podczas korzystania z aplikacji, jest potrzeba użycia bardzo dużej bazy słow. Baza, która została użyta do testów, zawierała ponad 8 milionów słów-trójek. W drugiej skonstruowanej bazie danych, zawarto ponad 30 milionów słów-trójek. Stworzenie tak dużych baz, wraz z mechanizmami do szybkiego wyszukiwania w nich potrzebnych informacji, wymagało sporej ilości czasu. Internet zawiera nieskończoną wręcz liczbę tekstów, które można wykorzystać w procesie tworzenia i stałego uzupełniania bazy danych. Nie jest możliwe, aby docelowo każdy użytkownik, pragnący korzystać z programu, instalował taką lokalną bazę z zebranymi słowami. Istnieje jednak możliwość uruchomienia jednej bazy danych na specjalnie przystosowanym serwerze, z którą mogliby łączyć się użytkownicy. Dodatkowo możliwe jest uruchomienie pająka internetowego, tak aby baza danych była ciągle uzupełniana o nowe konteksty wypowiedzi.

Samo wyszukiwanie informacji w grafie LHG nie zajmuje wiele czasu. Sprawdzanie, czy słowo występuje w słowniku języka polskiego oraz czy jest w określonym kontekście, zajmuje średnio ponad jedną sekundę – co jest w pełni akceptowalne z praktycznego punktu widzenia. Proces uzyskiwania podpowiedzi końcówek do wprowadzonego początku słowa, trwa od jednej do dwóch sekund, natomiast etap automatycznej kontekstowej korekcji tekstu składającego się trzech zdań trwa średnio od trzech do pięciu sekund. Można zatem stwierdzić, że zostały spełnione wymagania czasowe stawiane przez potencjalnego użytkownika.

Nie było możliwe również uzyskanie bardzo dobrej automatycznej korekcji tekstu dla całych zdań, które występują dość rzadko. Algorytm korzysta jedynie z wiedzy zapisanej w bazie danych. Jeśli wpisane przez użytkownika wyrazy nie wystąpiły w bazie, program nie będzie w stanie poprawić tekstu. Jeśli natomiast określony fragment zdania występował w bazie bardzo rzadko, to użytkownikowi zostanie zaprezentowana poprawa właśnie w kontekście zawartym w bazie słów.

Istnieje szereg możliwość rozbudowania istniejącego programu tak, aby jeszcze lepiej potrafił dokonywać korekty tekstu, dla większej ilości zdań. Do możliwości przyszłego rozwoju, powstałej już aplikacji, można zaliczyć:

1. Poszerzenie bazy słów o nowe wyrazy.

   Najprostszym rozwiązaniem wydaje się być ciągłe uzupełnianie bazy danych o analizę i przetworzenie nowych zdań. Dzięki temu, możliwe będzie prowadzenie poprawnej korekcji większej ilości tekstów. Niestety takie rozwiązanie pociąga za sobą ciągłe jej rozbudowywanie, co powoduje zwiększenie zajętości dysku przez bazę oraz przede wszystkim zwiększa czas wyszukiwania wyrazów, przez co program może wolniej działać.

2. Opracowanie lepszej metody automatycznej korekcji.

   Dotychczasowo algorytm wykorzystywał informację o częstości pojawiania się słów w określonym kontekście, ich odległość edycyjną oraz długość słowa. Możliwe jest zmodyfikowanie mechanizmu w przyszłości, tak aby podczas etapu poprawy korzystał z wielu metod algorytmów na raz w odpowiedniej kolejności. Do najbardziej skutecznych, oprócz już zaimplementowanych, powinny należeć:

   - metoda sprawdzania odległości na klawiaturze, która została opisana w rozdziale 3.6,
   - stworzenie listy najczęściej popełnianych błędów przez użytkownika,
   - metoda analizy poszczególnych sylab wyrazu, która została opisana w rozdziale 3.5.

3. Analizę możliwych wystąpień części mowy w zdaniu.

   Na podstawie istniejącej już bazy danych, można każdemu słowu przypisać jego część mowy. Na tej podstawie można stwierdzić, czy możliwa jest konstrukcja zdania w którym występowałyby, np. odpowiednio dane części mowy *rzeczownik, przymiotnik, czasownik*. Wynika z tego, że wprowadzenie takiego rozwiązania dałoby możliwość analizy i poprawnej korekcji zdania, którego wyrazy nie wystąpiły dotychczas w bazie kontekstów słownych.

4. Wprowadzenie i analiza reguł budowy zdań (zastosowanie sztucznej inteligencji, sieci semantycznej).

   Rozwiązaniem, które powinno wprowadzić najlepszą możliwą automatyczną kontekstową korektę tekstu, będzie opracowanie i stworzenie reguł korekty. Będą one utworzone na podstawie danych zawartych w bazie słów o określonym języku. Dzięki nim będzie możliwe określenie zasad, które są wykorzystywane podczas tworzenia zdań. Technologie takie jak Semantic Web, które dają możliwość łączenia i wnioskowania na podstawie rozproszonych źródeł danych, będą stanowiły więc podstawę rozwoju aplikacji. Również zaimplementowana w niej sztuczna inteligencja, będzie w stanie na bieżąco uczyć się możliwych konstrukcji i coraz lepiej, automatycznie poprawiać wystąpienia błędów.

Zaproponowany przez dr Adriana Horzyka - promotora tej pracy, Graf Przyzwyczajeń Lingwistycznych oraz późniejsze jego wykorzystanie, dla zaimplementowanego innowacyjnego algorytmu poprawy tekstu, dało możliwość skonstruowania aplikacji, która wykorzystując kilka mechanizmów, znacząco lepiej i bardziej automatycznie dokonuje poprawy błędnych tekstów w języku polskim, dzięki wykorzystaniu kontekstu słów w określonych formach fleksyjnych.

Obecnie na rynku dostępnych jest kilka edytorów tekstu w języku polskim, które posiadają zaimplementowane mechanizmy automatycznej jego korekty. Można wymienić tutaj, dwa najczęściej spotykane:

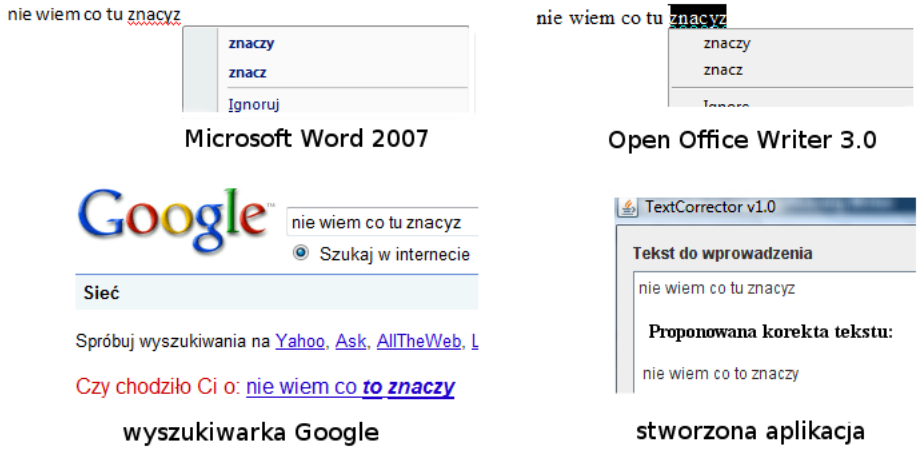
1. Microsoft Word,
2. Open Office Writer.

Niestety żaden z nich nie potrafi dokonywać kontekstowej korekty wprowadzonego tekstu. Mechanizmem poprawy błędnie (również kontekstowo) wprowadzonego tekstu, dysponuje również wyszukiwarka Google (`www.google.pl`). Nie jest jednak możliwe stwierdzenie, że może ona być użyta jako edytor tekstu. Dodatkowo w wielu miejscach również i mechanizm w niej zaimplementowany nie wykrywa błędów.

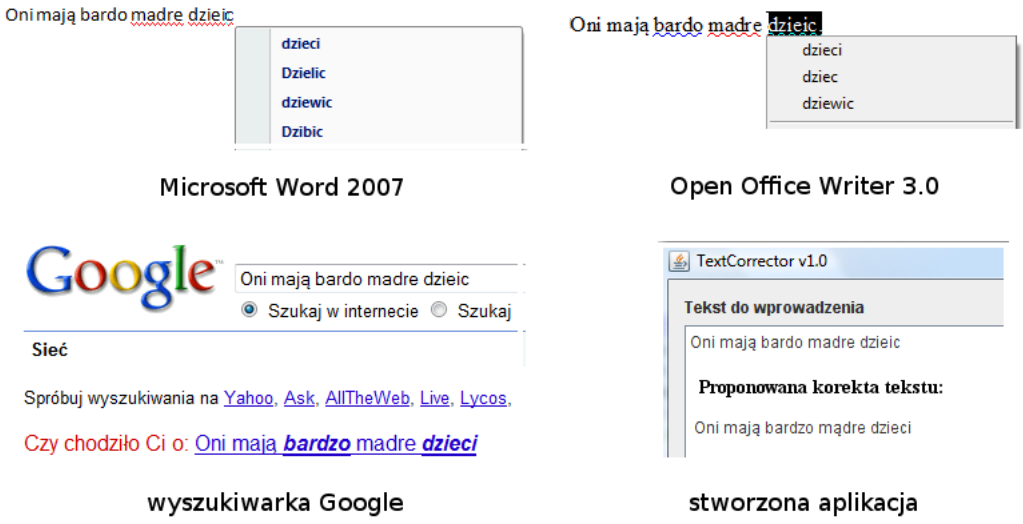
Na następnych ilustracjach zostały przedstawione proponowane poprawy tekstu z błędami przez:

- Edytor Microsoft Word 2007.
- Edytor Open Office Writer 3.0.
- Wyszukiwarkę Google.
- Stworzoną aplikację.

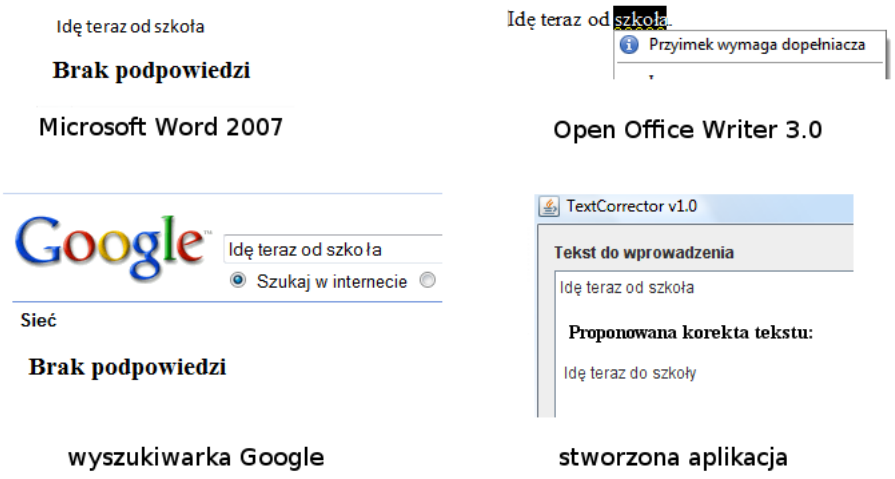
Analizując rysunki 6.1, 6.2 oraz 6.3 można zauważyć, że najlepsze wyniki korekty tekstu, dla zdań przeczytanych przez stworzonego wcześniej pająka, osiągnięto przez skonstruowany w tej pracy algorytm, który powstał we współpracy z promotorem - dr Adrianem Horzykiem. Wobec powyższego można wyprowadzić następujący wniosek, iż jeżeliby pająk internetowy jeszcze dłużej czytał teksty, konstruując przy tym bardziej rozbudowany graf LHG, wtedy opracowane mechanizmy byłyby w stanie jeszcze lepiej i bardziej automatycznie korygować teksty zastępując lub uzupełniając obecnie stosowane rozwiązania.



Rysunek 6.1: Proponowana poprawa zdania *nie wiem co tu znacyz*



Rysunek 6.2: Proponowana poprawa zdania *Oni mają bardo madre dzieic*



Rysunek 6.3: Proponowana poprawa zdania *Idę teraz od szkoła*

7. Podsumowanie

Zadaniem jakie zostało podjęte w tej pracy było stworzenie aplikacji, która wykorzystywać będzie innowacyjny algorytm do automatycznej kontekstowej korekty tekstu dla języka polskiego. Wymagało to realizacji kilku mniejszych modułów:

- Pierwszym zadaniem było stworzenie pająka internetowego, który przeszukując strony internetowe napisane w języku polskim, dostarczy informacji o możliwych konstrukcjach poprawnych zdań.
- Kolejnym zadaniem było stworzenie Grafu Przyzwyczajeń Lingwistycznych. Miał on być bardzo ogólny, a zarazem maksymalnie rozbudowany dla języka polskiego. Graf ten ma za zadanie zliczać wystąpienia różnych słów o pewnej formie fleksyjnej, w ich rzadkim kontekście początku i końca zdania oraz przede wszystkim w rzeczywistym i bardzo rzadkim kontekście innych słów o określonej formie fleksyjnej.
- Następnie, na podstawie zbudowanego grafu, należało opracować unikalny algorytm do automatycznej kontekstowej korekcji testów, które zostały wprowadzone przez użytkownika z błędami.
- Ostatnim zadaniem było wykonanie porównania działania zaproponowanego algorytmu z innymi mechanizmami występującymi w najczęściej spotykanych edytorach tekstu.

Do znaczących zalet wyróżniających unikatowy algorytm pośród innych należą:

1. Moduł do tworzenia nowych zdań.

   Bazuje on na zbudowanym wcześniej, przez pająka internetowego Grafie Przyzwyczajeń Lingwistycznych (LHG). Dzięki niemu użytkownik programu, nie znając języka, może po zaproponowaniu dwóch początkowych słów utworzyć całe zdanie. Algorytm podpowiada mu bowiem gotowe wyrazy, które nie dość, że są poprawne ortograficznie, to występują w zadanym kontekście. Mechanizm nie proponuje wszystkich słów występujących w Słowniku języka polskiego, lecz podaje jedynie najbardziej pasujące słowa.

2. Unikalny sposób wykrywania potencjalnych błędów w tekście.

   Jedynie w zaimplementowanym algorytmie badany jest kontekst wystąpienia wyrazów w zdaniu. W innych popularnych edytorach tekstu sprawdzana jest tylko poprawność ortograficzna wyrazu. Stąd nie znajdą one błędu w zdaniu, w którym użyto wyrazów poprawnie ortograficznych, lecz zupełnie oderwanych kontekstowo od siebie. Przykładem takiego zdania, w którym nie zostanie znaleziony błąd może być *Ja dom nie kot ale zjadłem grać grupa tekst*. Człowiek, który czyta ten tekst nie może zrozumieć jego treści i uważa zdanie za zupełnie błędne, w którym nie można znaleźć żadnej logiki. Niestety programy do edycji tekstu nie znajdują w nim żadnych błędów. Jedynie zastosowany mechanizm sygnalizuje dla każdego wyrazu, że choć jest poprawny ortograficznie, to nie występuje w danym kontekście.

3. Inteligentne dopełnianie wyrazów.

   Choć w programie Writer 3.0 występuje moduł do ”podpowiadania” słów, pozostawia on

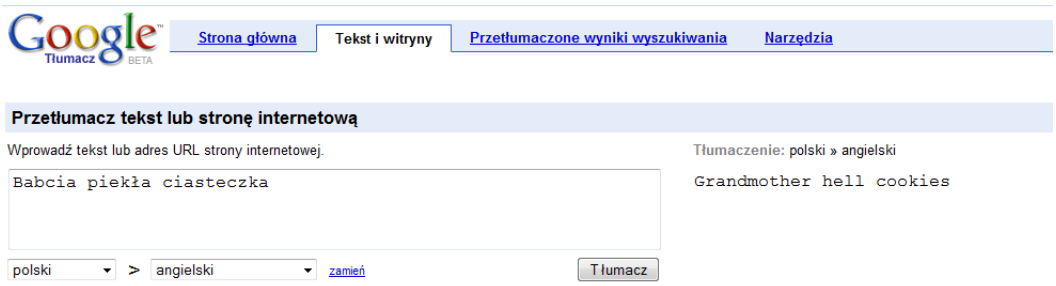
dużo do życzenia. W przeanalizowanych wielu przykładach, podpowiedzi generowane przez wspomniany program, pojawiały się sporadycznie. Wyrazy jakie były prezentowane użytkownikowi, to przede wszystkim nazwy własne, stąd pożyteczność tego modułu można ocenić poniżej wartości akceptowalnej. Program Microsoft Word 2007 potrafi w trakcie pisania automatycznie zmieniać wprowadzony wyraz z błędem na prawidłowy. Niestety w wielu przypadkach zmieniony wyraz nie pasował do reszty zdania. Co więcej program wykonywał to automatycznie, więc we wprowadzanym przez użytkownika tekście nieświadomie powstawały błędy. O ile błędy w wyrazach mogły być podkreślone na czerwono, tak że użytkownik mógłby znacznie szybciej je zauważyć i poprawić, o tyle błędne wyrazy były zamieniane na inne poprawne ortograficznie bez zaznaczenia miejsca ich zmiany. W celu znalezienia takiego błędu użytkownik musiał jeszcze raz czytać tekst (który przez program został uznany za prawidłowy). W stworzonej aplikacji ”podpowiedź” można uzyskać dla każdego początku wyrazu, o ile możliwe jest utworzenie wyrazu o takim początku w określonym kontekście. Uważam, że takie podpowiedzi są bardzo cenne dla użytkowników i powinny być zaimplementowane w najpopularniejszych edytorach tekstu.

4. Automatyczna kontekstowa korekta tekstów.

   Największą zaletą edytorów tekstu jest możliwość znajdywania przez nie błędów w zdaniach oraz automatyczne ich poprawianie. Na polskim rynku brakuje jednak edytorów, które robiłyby to w sposób zadowalający, uwzględniając, np. najbliższy kontekst wypowiedzi. Wszystkie przetestowane edytory sprawdzały jedynie pojedyncze wyrazy - w oderwaniu od reszty. Porównując stworzony moduł do automatycznej korekty zdań z innymi, ten pierwszy daje znacznie lepsze rezultaty. Korzysta on z bardzo rozbudowanego grafu LHG do sprawdzania czy zdanie wprowadzone przez użytkownika jest prawidłowe i czy może wystąpić właśnie w takim kontekście. Ponadto do określenia najlepszych słów aplikacja ta wykorzystuje kilka metod:

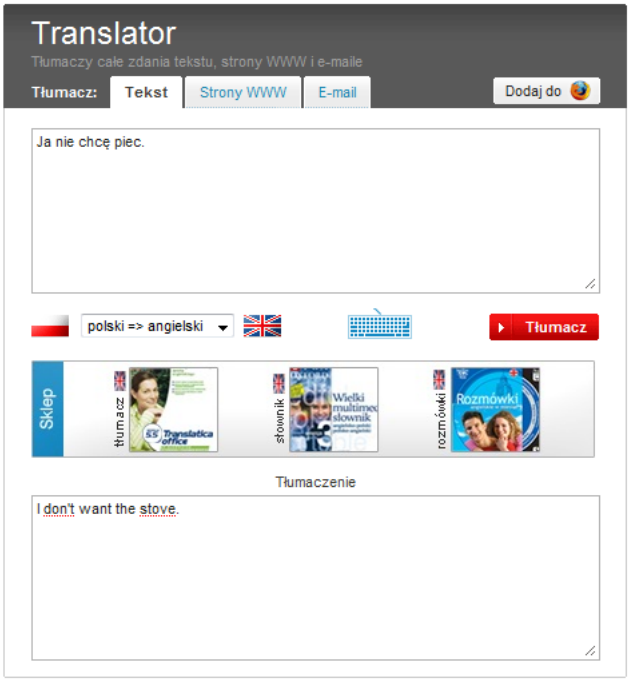
   - mechanizm, który zlicza w ilu najbliższych kontekstach znajduje się wprowadzone słowo,
   - mechanizm sprawdzający częstość występowania danego wyrazu w zadanym kontekście,
   - mechanizm obliczający odległość edycyjną od słowa uznanego za błędne,
   - mechanizm wyznaczający różnicę w długości wprowadzonych wyrazów,
   - mechanizm wyznaczający słowo, które jest uznane za ”najlepsze” do zamiany.

Analiza oraz przetwarzanie zdań w języku naturalnym prowadzi do zrozumienia ich sensu. Jest to w obecnych czasach jedno z najważniejszych zadań i wyzwań stojących przed programami komputerowymi. Jak ważne jest zrozumienie kontekstu słownego, najlepiej prezentują wszelkiego rodzaju translatory, przekształcające zdania z jednego języka na drugi. Do ówcześnie najlepszych można zaliczyć translatory polsko-angielskie firmy Google oraz translator Translatica.pl, który jest wspierany przez Wydawnictwo Naukowe PWN.



Rysunek 7.1: Próba przetłumaczenia zdania ”Babcia upiekła ciasteczka”

W pierwszym przypadku zamiast poprawnego przetłumaczenia przez translator Google zdania *Babcia upiekła ciasteczka*, otrzymano zdanie w dosłownym tłumaczeniu *Babcia piekło ciasteczka*. W drugim



Rysunek 7.2: Próba przetłumaczenia zdania ”Ja nie chcę piec”

przypadku, zamiast poprawnego przetłumaczenia przez translator Translatica.pl zdania *Ja nie chcę piec*, uzyskano zdanie w dosłownym tłumaczeniu *Ja nie chcę piecyka*. Jak można zauważyć analizując te dwie pary zdań kontekst zdania wejściowego różni się od kontekstu zdania wyjściowego.

A. Opis techniczny zbudowanej aplikacji

A.1. Aplikacja

Jak zostało wspomniane, w ramach niniejszej pracy została stworzona aplikacja do automatycznej kontekstowej korekty tekstu. Do jej powstania zostały wykorzystane następujące języki i biblioteki:

- SUN Java 6.0 - język programowania i środowisko, w którym program został zaimplementowany [26].
- Biblioteka Swing - biblioteka wykorzystana do stworzenia interfejsu użytkownika [26].
- Biblioteka Java Universal Network/Graph Framework - służąca do tworzenia sieci i grafów. Wykorzystana w module budującym Graf Przyzwyczajeń Lingwistycznych [16].
- Biblioteka Apache log4j - framework służący do logowania informacji podczas działania aplikacji [15].
- Biblioteka TestNG - framework służący do utworzenia testów dla aplikacji [27].
- Projekt Apache Commons - projekt, w skład którego wchodzi wiele dodatkowych komponentów dla Javy [14].

Wszystkie pozostałe funkcjonalności zostały zaimplementowane własnoręcznie przez autora.

A.2. Baza danych

Baza danych jaką użyto, do przechowywania wszystkich danych, to MySQL w wersji 5.1.11. Dodatkowo do operacji na niej wykorzystano narzędzia MySQL GUI Tools [18].

A.3. Słownik frekwencyjny

Jak zostało wspomniane, na potrzeby algorytmu, został wykorzystany Słownik frekwencyjny języka polskiego [7]. Słownik ten został opracowany przez krakowską Grupę Lingwistyki Komputerowej, a udostępniony przez dr inż. Marka Gajęckiego.

A.4. Legalność

Do wykonania aplikacji użyto legalnego oprogramowania, na odpowiednich licencjach.

Bibliografia

[1] dr inż. Marek Gajęcki. *Przetwarzanie Języka Naturalnego*. AGH, 2008.

[2] P. Gawrysiak. *Modelowanie języka*. Politechnika Warszawska, 2006.

[3] A. Horzyk. *Innovative Prediction Technology for Automatic Speech Recognition pp. 251-258*. EXIT, 2005.

[4] E. Jędrzejko. Teoretyczne problemy nominalizacji - przegląd ujęć i propozycji metodologicznych.

[5] M. Junczys-Dowmunt. *Zastosowanie automatów skończonych stanów w przetwarzaniu języków naturalnych*. Uniwersytet Kazimierza Wielkiego w Bydgoszczy, 2008.

[6] M. Knasiecki. *Grafy i ich reprezentacje*. Struktury danych - Klasyczne (`www.algorytmy.org`), lipiec 2005.

[7] K. G. L. Komputerowej. Słownik frekwencyjny języka polskiego. 2009.

[8] M. Marciniak. Ms office kontra openoffice. *PC Word*, 2008.

[9] J. Miró. *Czy w Unii Europejskiej mówiono po polsku?* Magazyn Delta, 05/2004.

[10] A. Mykowiecka. *Inżynieria lingwistyczna. Komputerowe przetwarzanie tekstów w języku naturalnym*, volume ISBN: 83-89244-54-3. Wydawnictwo Polsko-Japońskiej Wyższej Szkoły Technik Komputerowych, wydanie i edition, 2007.

[11] Odległość levenshteina. Odległość Levenshteina (odległość edycyjna) `http://www.algorytm.org/index.php?option=com_content&task=view&id=256`.

[12] prof. dr hab. Marek Świdziński. *Wybrane narzędzia przetwarzania tekstów polskich*. Zakład Językoznawstwa Komputerowego Instytut Języka Polskiego UW, 2008/2009.

[13] Łukasz Dębowski. *Prawo Zipfa - próby objaśnień*. Instytut Podstaw Informatyki PAN, 2005.

[14] Apache commons. komponenty dla języka Java `http://commons.apache.org/`.

[15] Apache log4j. biblioteka do logowania działania aplikacji `http://logging.apache.org/log4j/`.

[16] Java universal network/graph framework. biblioteka do tworzenia grafów `http://jung.sourceforge.net/`.

[17] Microsoft office word 2007. opis programu Word `http://office.microsoft.com/pl-pl/word/HA101650321045.aspx`.

[18] Mysql. baza danych `http://www.mysql.com/`.

[19] Openoffice.org writer. opis programu Writer `http://pl.openoffice.org/`.

[20] Pająk internetowy acme.spider. `http://www.acme.com/java/software/Acme.Spider.html`.

[21] Pająk internetowy checkbot. `http://degraaff.org/checkbot/`.

[22] Pająk internetowy firmy fluid dynamics software corporation. `http://www.xav.com/scripts/search/`.

[23] Pająk internetowy jobo. `http://www.matuschek.net/jobo/`.

[24] Spiders.pl wszystko o robotach wyszukiwarek. `http://spiders.pl/`.

[25] Statistical inference: n-gram models over sparse data. `http://mi007.wikispaces.com/file/view/rozdzial6.pdf`.

[26] Sun java 6.0. `http://java.sun.com/`.

[27] Testng. biblioteka do wykonania testów `http://testng.org/`.