*Marek Gajęcki**

Serwer leksykalny języka polskiego

1. Wprowadzenie

Dla lepszego zrozumienia poniższego tekstu niezbędne jest wprowadzenie kilku podstawowych pojęć związanych z językiem naturalnym [4].

Leksyka (słownictwo) to ogół wyrazów wchodzących w skład jakiegoś języka. **Wyraz** – byt językowy, jednostka słownika języka naturalnego jest bytem istniejącym jedynie w świadomości (umyśle) nosicieli danego języka. Wyraz – jako byt abstrakcyjny – przejawia się w wypowiedziach lub tekstach w postaci **form tekstowych**. Wszystkie formy tekstowe danego wyrazu tworzą **wektor odmiany** wyrazu. Wektor odmiany (jego długość, zawartość) zależy od tego, jaką częścią mowy jest wyraz, np. dla rzeczownika wektor odmiany liczy 14 elementów (7 przypadków i dwie liczby).

Przy przetwarzaniu tekstów zapisanych w języku fleksyjnym spotykamy się z podstawowym problemem: jeden wyraz przybiera wiele form tekstowych, często dana forma tekstowa należy do dwóch lub więcej wyrazów. Przykład tego typu zależności pokazuje tabela 1.

Tabela 1
Wyraz i jego formy tekstowe

Wyraz	Formy tekstowe
typ (np. podejrzany typ)	typ, typa, typowi, ...
typ (np. typ komputera)	typ, typu, typowi, ...
typowy	... typowi, typowych, ...
zieleń	... zieleń, zieleni, ...
zielony	... zieleni, zielonych, ...
zielenić się	... zieleni się, zieleniła się, ...

* Katedra Informatyki, Akademia Górniczo-Hutnicza, Kraków

Dlatego, przy przetwarzaniu języka naturalnego niezbędne okazuje się posiadanie narzędzia umożliwiającego odtwarzanie relacji pomiędzy wyrazem a jego formami tekstowymi. Relacje te są dwojakiego rodzaju: z jednej strony jest to rozpoznawanie wyrazu na podstawie form tekstowych, określanie klasy fleksyjnej, rodzaju, aspektu, stopnia, itp. i z drugiej strony generowanie, należących do wektora odmiany, określonych form tekstowych dla konkretnego wyrazu.

Niniejszy artykuł zawiera opis systemu realizującego wyżej opisane zadania. W obecnej wersji (opis dotyczy wersji 1.2) system nie pozwala na jednoznaczne rozróżnienie wyrazów na podstawie form i dostarcza odpowiedzi wariantowej, na przykład dla formy tekstowej „typowi" system dostarczy trzech odpowiedzi (patrz tab. 1).

Od strony technicznej system realizujący opisane tu usługi można wykonać różnymi sposobami:

- jako moduł biblioteczny dołączany do aplikacji użytkownika,
- jako działający na lokalnej lub zdalnej maszynie serwer, z którym komunikuje się aplikacja użytkownika poprzez sieć lokalną.

W dalszej części opisano system zrealizowany w postaci serwera leksykalnego (LS), świadczącego usługi dla innych programów przetwarzających teksty języka polskiego. Poprzez usługi rozumie się rozpoznawanie wyrazów na podstawie form oraz dostarczanie informacji o wyrazach należących do języka.

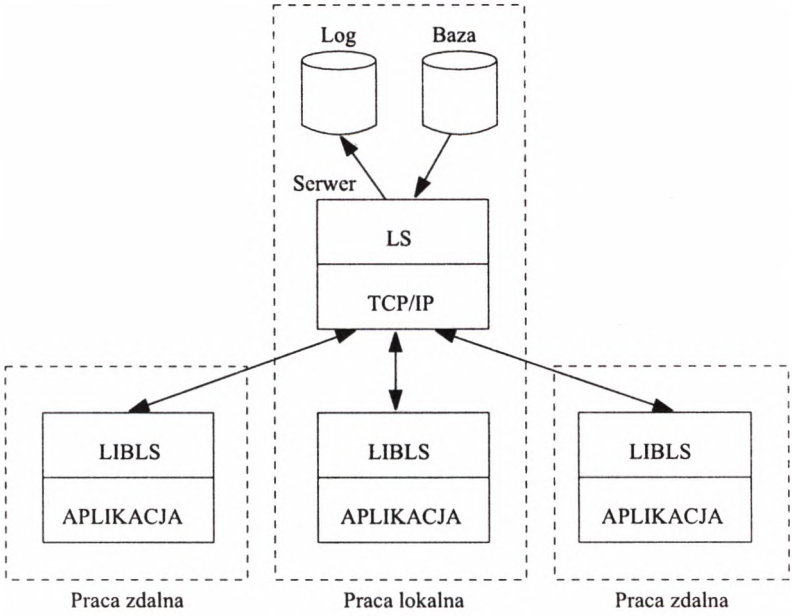
O wyborze rozwiązania w postaci serwera zadecydowały jego następujące zalety:

- możliwe jest tworzenie statystyki zapytań kierowanych do serwera przez programy użytkowników, co stanowi bardzo cenny materiał badawczy mogący służyć zarówno do uzupełniania bazy słownikowej, jak i do szeroko rozumianych badań lingwistycznych;
- istnieje jeden globalny słownik, co pozwala utrzymać spójność danych, słownik jest uzupełniany przez administratora serwisu we współpracy z językoznawcą;
- aplikacje użytkowników są mniej obciążane czasowo i pamięciowo, gdyż do programu dołączana jest jedynie stosunkowo niewielka biblioteka służąca komunikacji z serwerem leksykalnym;
- biblioteka służąca komunikacji z serwerem leksykalnym może być napisana w różnych językach programowania i na różnych platformach systemowych;
- łatwiejsze jest egzekwowanie praw autorskich.

Rozwiązanie w postaci serwera posiada także wady, np. narzut komunikacyjny, zwłaszcza przy serwerze pracującym na odległej maszynie.

2. Idea systemu LS

System zrealizowano w oparciu o model klient-serwer (rys. 1). Protokołem komunikacyjnym jest TCP/IP, serwer oczekuje na połączenie z klientami na porcie o numerze 4244. Serwis uruchomiono na maszynie winnie.ics.agh.edu.pl (IP 149.156.100.100) pracującej pod systemem operacyjnym LINUX. Polskie znaki diakrytyczne są kodowane w standardzie ISO-8859-2.



Rys. 1. Serwer leksykalny – model klient-serwer

Serwer posiada budowę modularną. Podstawowym modułem jest biblioteka LS obsługująca zapytania skierowane do bazy słownikowej, drugi moduł – TCP/IP jest odpowiedzialny za komunikację z klientami poprzez sieć. Po stronie klienta aplikacja komunikuje się z serwerem poprzez wywoływanie funkcji z biblioteki LIBLS. Biblioteka ta może być napisana dla dowolnego języka programowania, co pozwala na tworzenie aplikacji przetwarzających język naturalny z wykorzystaniem różnych narzędzi programistycznych (języków programowania). Warunkiem koniecznym do współpracy z serwerem leksykalnym jest, aby język programowania umożliwiał dołączanie modułów napisanych w języku C lub bezpośrednio obsługiwał funkcje komunikacyjne wykorzystujące standard gniazd [6].

W chwili obecnej dostępne są biblioteki dla języków: C, C++, JAVA, Perl oraz Icon [2]. Interfejs funkcjonalny biblioteki klienta LIBLS (dla języka C) jest zgodny z biblioteką LS użytą w serwerze. Umożliwia to łatwe zbudowanie aplikacji z pominięciem części komunikacyjnej. Ma to szczególne znaczenie w przypadku intensywnego odwoływania się do danych zgromadzonych w bazie słownikowej wtedy, gdy komunikacja poprzez sieć stanowi wąskie gardło systemu.

W dalszej części artykułu przedstawiono bibliotekę dla języka Icon wraz z przykładami komunikacji z serwerem leksykalnym.

3. Baza słownikowa

Serwer leksykalny współpracuje z bazą słownikową zawierającą informacje o wyrazach, niezbędne do realizowania usług. Baza została stworzona na podstawie Bazy Fleksyjnej Języka Polskiego. Aktualny rozmiar bazy, wynoszący około 150 tysięcy wyrazów, obejmu-

je praktycznie cały podstawowy słownik wyrazów pospolitych i stosunkowo niewielką liczbę nazw własnych. Aktualnie trwają prace nad rozbudową słownika nazw własnych. Przy analizie typowych tekstów np. notatek prasowych nierozpoznane pozostają jedynie nazwy oraz wyrazy należące do słownictwa dziedzinowego.

3.1. Struktura bazy

Baza składa się z dwóch części: słownika wyrazów jednoczłonowych (w tym także nazw) oraz słownika wyrazów wieloczłonowych. W pierwszej części zgromadzone są także składowe wyrazów wieloczłonowych. Wszystkie jednostki bazy słownikowej można podzielić na cztery typy: jednoczłonowe wyrazy pospolite, jednoczłonowe nazwy, składowe wyrazów wieloczłonowych oraz wyrazy wieloczłonowe. Informacje o typie przechowywane są w rekordzie opisującym jednostkę w polu `typ`.

Wyrazy w bazie słownikowej dzielą się także na klasy odpowiadające częściom mowy: rzeczownik, czasownik, przymiotnik, liczebnik, zaimek, przysłówek i wyraz nieodmienny. Ponieważ z każdą częścią mowy związane są inne informacje (np. rzeczownik – rodzaj, czasownik – aspekt) rekordy bazy opisujące wyrazy w bazie słownikowej są rekordami z wariantami.

Część pól rekordów jest wspólna dla wszystkich klas fleksyjnych, pozostałe pola zależą od tego jaką częścią mowy jest opisywany wyraz. Dla poszczególnych klas rekordy opisujące wyrazy zawierają następujące pola:

```
rzeczownik   : typ, kl, id, kod, fh, et, rodz, kat
czasownik    : typ, kl, id, kod, fh, et, aspekt, im1, im2
przymiotnik  : typ, kl, id, kod, fh, et, stopień, st1, st2, st3
liczebnik    : typ, kl, id, kod, fh, et
zaimek       : typ, kl, id, kod, fh, et, rodz
przysłówek   : typ, kl, id, kod, fh, et, stopień, st1, st2, st3
nieodmienny  : typ, kl, id, kod, fh, et, kat
```

Imiesłowy występują w bazie jako elementy odmiany czasownika. Dodatkowo, imiesłowy odmienne występują jako niezależne jednostki słownikowe z klasą określoną jako przymiotnik.

Znaczenie pól wspólnych jest następujące:

- `typ` – typ wyrazu (1 – pospolity, 2 – nazwa, 3 – składowa, 4 – wieloczłonowy),
- `kl` – klasa wyrazu,
- `id` – numer identyfikujący wyraz,
- `kod` – kod szczegółowy,
- `fh` – forma podstawowa (hasłowa),
- `et` – etykieta fleksyjna wyrazu.

Pole `kl` określa jaką częścią mowy jest wyraz. Przyjmuje następujące wartości:

1 : rzeczownik,
2 : czasownik,
3 : przymiotnik,
4 : liczebnik,
5 : zaimek,
6 : przysłówek,
7 : wyraz nieodmienny.

Pole `id` zawiera numer, który w sposób jednoznaczny określa wyraz. Numer jest siedmiocyfrowy, przy czym pierwsza cyfra jest równa liczbie segmentów z jakich składa się wyraz. Pole `kod` ma wyłącznie znaczenie techniczne. Pole `fh` określa podstawową formę wyrazu pozwalającą określić wyraz, określenie to nie jest jednoznaczne.

Dla rzeczownika rekord dodatkowo zawiera pola:

`rodz` – rodzaj rzeczownika,
`kat` – kategoria nazwy lub 0 dla rzeczownika pospolitego.

Pole `rodz` określa rodzaj gramatyczny rzeczownika i przyjmuje następujące wartości:

1 : męski osobowy (np. brat),
2 : męski żywotny (np. burak),
3 : męski nieżywotny (np. biegun),
4 : żeński (np. belka),
5 : nijaki (np. boisko),
6 : osobowy (np. czworaczki),
7 : nieosobowy (np. cięgi),
8 : nieodmienny (np. etui).

Pole `kat` określa kategorię nazwy. Dla rzeczowników pospolitych przyjmuje wartość 0, dla nazw – wartości od 10 do 57 (10 – imię, 11 – nazwisko, 12 – przezwisko, itd.).

Dla czasownika rekord dodatkowo zawiera pola:

`aspekt` – aspekt czasownika (1 – czasownik dokonany, 2 – czasownik niedokonany),
`im1,im2` – numery `id` odmiennych imiesłowów związanych z czasownikiem.

Dla przymiotnika i przysłówka rekord dodatkowo zawiera pola:

`stopień` – stopień przymiotnika (1 – równy, 2 – wyższy, 3 – najwyższy,
`st1,st2,st3` – numery `id` stopni: równego, wyższego i najwyższego.

Rekord opisujący zaimek zawiera pole określające rodzaj. Wyraz nieodmienny zawiera pole kat określające kategorię. W obecnej wersji kategoria wyrazu nieodmiennego nie jest zaimplementowana.

Baza słownikowa, przechowywana w pliku `ls.db`, jest jednorazowo ładowana do pamięci operacyjnej po uruchomieniu serwera. Umieszczenie całej bazy słownikowej w pamięci operacyjnej pozwala na szybką realizację funkcji usługowych. Zastosowana struktura danych zapewnia kompromis pomiędzy szybkością realizacji funkcji, a wielkością pamięci zajmowanej przez słownik.

3.2. Informacje o serwerze i bazie słownikowej

Proces tworzenia bazy słownikowej nie jest zakonczony, dlatego niezbędna jest możliwość określania z jaką wersją bazy współpracuje serwer. Użytkownik serwera leksykalnego może uzyskać informacje dotyczące zarówno serwera, jak i bazy słownikowej. Informacje, jakie można uzyskać to m.in.: numery wersji serwera i słownika, daty ich powstania, ilość wyrazów jednoczłonowych i wieloczłonowych oraz rozmiar jaki zajmuje baza słownikowa jako plik dyskowy i po załadowaniu do pamięci operacyjnej. Jednym ze sposobów uzyskania informacji jest wykonanie programu `ls_info`.

```
[mag] $ ls_info.

 Serwer
      Wersja programu serwera   : 1.2
      Kompilacja                : Dec 20 1999
 Słownik
      Wersja słownika           : 1.0
      Generacja słownika        : Nov 15 1999
      Plik słownika             : ls.db
      Liczba jednoczłonowych    : 136745
      Liczba wieloczłonowych    : 13094
      Razem                     : 149839
      Rekordów                  : 152802
      Pamięć                    : 2849925 (19.0)
      Dysk                      : 699558 (4.6)
```

Przedstawione powyżej informacje są dostępne za pomocą opisanej w dalszej części raportu funkcji bibliotecznej `ls_info()`.

4. Komunikacja z serwerem leksykalnym

4.1. Protokół komunikacyjny

Komunikacja pomiędzy biblioteką klienta LIBLS a serwerem komunikacyjnym odbywa się z wykorzystaniem protokołu TCP/IP. Połączenie jest inicjowane przez klienta, który otwiera połączenie. Następnie serwer oczekuje na zapytanie. Po jego otrzymaniu i znalezieniu odpowiedzi w bazie słownikowej, wysyła odpowiedź (lub odpowiedzi – na dane zapytanie

może przyjść więcej niż jedna odpowiedź). Opisany schemat może się powtarzać do momentu, gdy klient zamyka połączenie. Jeżeli połączenie jest nieaktywne przez pewien określony czas, jest ono zamykane przez serwer.

Format zapytania jest następujący:

```
<zapytanie>           ::= <nagłówek> <zapytanie właściwe>
<nagłówek>            ::= <id> <rozmiar>
<id>                  ::= int32
<rozmiar>             ::= int32
<zapytanie właściwe>  ::= { <pole> }
<pole>                ::= int32 | <łańcuch>
<łańcuch>             ::= <długość> { char }
<długość>             ::= int32
```

Znaczenie poszczególnych pól jest następujące:

- `<id>` - identyfikator zapytania (określa konkretną funkcję usługową,
- `<rozmiar>` - rozmiar w bajtach bloku `<zapytanie właściwe>`,
- `<łańcuch>` - łańcuch znakowy, nie kończony '\0', znaki według normy ISO-8859-2,
- `<długość>` - liczba znaków w przesłanym łańcuchu znakowym,
- `int32` - liczba 32 bitowa, zapisana w kolejności sieciowej (najstarszy bajt pierwszy).

Rozmiar zapytania właściwego przesyłany w nagłówku pozwala zaalokować pamięć na właściwe zapytanie. Analogiczna sytuacja występuje przy przesyłaniu odpowiedzi.

Format odpowiedzi jest następujący:

```
<odpowiedź>           ::= <nagłówek> <zapytanie właściwe>
<nagłówek>            ::= <id> <rozmiar> <więcej>
<id>                  ::= int32
<rozmiar>             ::= int32
<więcej>              ::= int32
<odpowiedź właściwa>  ::= { <pole> }
<pole>                ::= int32 | <łańcuch> | <tablica>
<łańcuch>             ::= <długość> { char }
<długość>             ::= int32
<tablica>             ::= <długość> { <wartość> }
<wartość>             ::= int32
```

Pole `<więcej>` przyjmuje: wartość 0, gdy przysłany pakiet jest ostatnim z serii (lub jedynym) oraz wartość różną od 0, jeżeli są jeszcze pakiety. W obecnej wersji wszystkie funkcje realizują odpowiedzi w postaci jednego pakietu, czyli pole `<więcej>` ma zawsze wartość 0.

4.2. Kodowanie argumentów i wyników funkcji

W obecnej wersji serwer realizuje odpowiedzi na 9 zapytań. Dokladny opis funkcji znajduje się w dodatku A. W tabeli 2 przedstawiono sposób przesyłania argumentów i wyników dla poszczególnych funkcji.

Tabela 2

Kodowanie argumentów i wyników funkcji

Funkcja	Id	Argument	Wynik
ls_info	20	Brak	S, S, S, S, S, I, I, I, I, I, I, I
ls_num	21	S	I, I, I
ls_find	22	S	T
ls_desc	23	I	I, I, I, I, S, S, ...
ls_form	24	I, I	S
ls_vec	25	I, S	S
ls_numw	26	I, I	I
ls_findw	27	I, I	T
ls_elw	28	I, I	I

Znaczenie liter występujących w tabeli jest następujące: S – napis, I – liczba całkowita, T – tablica liczb całkowitych. W przypadku funkcji `ls_desc` zwracany wynik jest zależny od klasy wyrazu, jaki jest opisywany (patrz rekord opisujący wyraz).

4.3. Sygnalizowanie błędów

Przy korzystaniu z serwera mogą wystąpić dwa rodzaje błędów. Pierwsze związane są z warstwą komunikacyjną. Przyczyną wystąpienia tego rodzaju błędów może być na przykład niepracujący serwer czy błędnie podany adres maszyny. Drugi rodzaj błędów występuje w przypadku, gdy przesłane do serwera zapytanie nie jest poprawne. Serwer sygnalizuje błąd zwracając jedynie nagłówek, w którym: pole `<id>` ma wartość `255`, pole `<długość>` zawiera kod błędu, pole `<więcej>` ma wartość `0`.

W obecnej wersji serwer rozpoznaje następujące błędy:

- błędny kod funkcji,
- błędnie zbudowany argument,
- numer `id` poza zakresem,
- błędna pozycja w wektorze odmiany.

Sposób sygnalizowania błędu na poziomie biblioteki klienta LIBLS zależy od języka, w jakim implementowana jest biblioteka; na przykład w języku JAVA funkcja realizuje wyjątek, a w języku Icon funkcja zawodzi.

4.4. Komunikacja z serwerem w języku Icon

Poniżej przedstawiono interfejs zrealizowany w postaci biblioteki dla języka Icon [3]. Zdaniem autora język ten najlepiej nadaje się do zastosowań związanych z szeroko rozumianym przetwarzaniem języka naturalnego. Biblioteka wymaga rozszerzeń obsługujących gniazda [5], współpracuje z serwerem leksykalnym w wersji 1.2. W programie napisanym w języku Icon należy dołączyć bibliotekę za pomocą dyrektywy `link libls`.

Biblioteka `libls` udostępnia następujące funkcje:

`ls_open(serwer)` – otwiera kanał komunikacyjny z serwerem,
`ls_close()` – zamyka kanał komunikacyjny z serwerem,
`ls_info()` – zwraca informacje o serwerze i bazie słownikowej,
`ls_num(forma)` – zwraca liczby wyrazów znalezionych w słowniku,
`ls_find(forma)` – zwraca ciąg numerów id wyrazów dla danej formy,
`ls_desc(id)` – zwraca opis wyrazu o numerze `id`,
`ls_form(id,nr)` – zwraca formę numer nr wyrazu o numerze `id`,
`ls_vec(id,forma)` – zwraca pozycje formy w wektorze odmiany,
`ls_numw(id,poz)` – zwraca liczbę znalezionych `id` wyrazów wieloczłonowych,
`ls_findw(id,poz)` – zwraca ciąg znalezionych `id` wyrazów wieloczłonowych,
`ls_elw(wid,poz)` – zwraca element wieloczłonowego na pozycji `poz`.

Powyższe funkcje wymagają komunikacji z serwerem leksykalnym. Oprócz nich w bibliotece `libls` występują dodatkowe funkcje, ułatwiające tworzenie aplikacji:

`op_klasa ( nr )` – zwraca opis klasy wyrazu,
`op_rodz ( nr )` – zwraca opis rodzaju rzeczownika,
`op_kat ( nr )` – zwraca opis kategorii rzeczownika,
`op_stop ( nr )` – zwraca opis stopnia przymiotnika lub przysłówka.

Pełny opis funkcji biblioteki `libls` zawarto w załączniku A. Biblioteka wraz z jej kodem źródłowym oraz proste przykłady jej wykorzystania są dostępne na stronie serwera leksykalnego [5].

4.5. Komunikacja z serwerem w języku C

Programy napisane w języku C mogą korzystać z usług serwera leksykalnego komunikując się przy użyciu gniazd. Dla wygodnego korzystania z usług serwera leksykalnego przez programy napisane w języku C została stworzona biblioteka `libls` analogiczna, jak opisana w poprzednim podrozdziale biblioteka dla języka Icon.

Biblioteka współpracuje z serwerem leksykalnym w wersji 1.2 i jest ona dostępna w postaci dwóch plików: `libls.a` i `libls.h`. Aby korzystać z usług serwera leksykalnego w programie napisanym w języku C należy dołączyć bibliotekę `libls.a` w czasie kompilacji programu

```
[mag] $ cc prog.c libls.a
```

W załączniku B przedstawiono plik zawierający definicje struktur danych oraz nagłówki funkcji biblioteki `libls` dla języka C. Biblioteka wraz z jej kodem źródłowym oraz proste przykłady jej wykorzystania są dostępne na stronie serwera leksykalnego [5].

Biblioteka `libls` dla języka C od strony interfejsu jest zgodna z biblioteką LS (patrz idea systemu LS). Pozwala to na łatwą zmianę programu korzystającego z serwera leksykalnego na samodzielną aplikację nie obciążoną narzutem komunikacyjnym.

5. Przykład zastosowania serwera leksykalnego

Jako przykład zastosowania serwera leksykalnego przedstawiono program generacji statystyki wystąpień wyrazów w zadanym tekście (rys. 2). Nie chodzi tu o statystykę wystąpień słów (form tekstowych), ale o wystąpienia wyrazów rozumianych zgodnie z definicją podaną na początku artykułu. Pomimo że serwer leksykalny w swojej bazie zawiera wyrazy będące nazwami własnymi, w poniższym przykładzie brane są po uwagę wyłącznie wyrazy znajdujące się w słowniku wyrazów pospolitych.

Program nie rozstrzyga niejednoznaczności powodowanych faktem, że dwa lub więcej wyrazów posiada identyczne formy tekstowe.

```
link libls

Procedure main( arg )
   t:=table(0)
   ls_open()
   f:=open(arg[1], "r") | stop("Brak pliku wejsciowego")
   while l:=read (f) do
      every tok:=parse(1) do
         every e:=ls_desc(ls_find(tok)) do
            if e.typ=1 & e.kl=3 then t[e.id]+:=1
   close(f)
   lp := sort(t,2)
   every i:= *lp to 1 by -1 do wri-
   te(left(ls_form(lp[i][1],0),16),lp[i][2])
   ls_close()
end

procedure parse( s )
   static litery
   initial litery := &lclase++&ucase++'ąćęłńóśżźĄĆĘŁŃÓŚŻŹ'
   s ? repeat
   {
      t := tab(many(litery)) | tab(upto(litery)) | break
      if  any (litery,t) then suspend t
   }
end
```

Rys. 2. Program generujący statystykę wystąpień wyrazów

W przykładzie napisanym w języku Icon wykorzystano bibliotekę `libls` dołączoną do programu. Program przetwarza plik tekstowy, którego nazwa jest przekazywana jako parametr wywołania programu. Rozpoznane formy tekstowe gromadzone są w tabeli `t`. Po zakończeniu przetwarzania pliku następuje wypisanie statystyki wystąpień wyrazów. Procedura `parse()` wyodrębnia z kolejnych wierszy tekstu kolejne formy tekstowe.

Wyniki działania programu dla tekstu bieżącego artykułu (bez załączników), stanowiącego dane dla programu przedstawiono w załączniku C. W przykładzie tym ograniczono się do statystyki wystąpień przymiotników w tekście raportu.

6. Problem wyrazów wieloczłonowych

Za wyraz wieloczłonowy autor przyjmuje taki, którego formy tekstowe zawierają znaki takie, jak: spacja, apostrof, cudzysłów, łącznik. Jako wyrazy wieloczłonowe w słowniku występują na przykład:

- nazwy, np. Akademia Górniczo-Hutnicza, Akademia Rolnicza,
- wyrazy z „się", np. bić się, mycie się,
- niektóre wyrazy pospolite, np. ping-pong.

```
procedure wielo( s )
   tok := [] # lista członów
   every put(tok, parse(s)
   ls_open()
   every id:=ls_find(tok[1]) do
      if ls_desc(id).typ=1 then
         every w:=ls_findw(id,0) do
         {
            # sprawozdanie zgodności członów
            zgodne  := 1
            every i:=2 to *tok do
               if  ls_find(tok[i])=ls_elw(w,i) then next else zgodne := 0
            if zgodne =1 then
            {
               #  sprawdzanie zgodności wektorów odmian
               res := ls_vec(ls_elw(w,1), tok[1])
               every i:=2 to *tok do
               {
                  vec := ls_vec(ls_elw(w,1), tok[i])
                  every j:= 1 to *res do
                     if res[j]=="0" | vec[j]=="0" then res[j]:="0"
               }
               if upto('1',res) then suspend w
            }
      }
   ls_close()
end
```

Rys. 3. Procedura rozpoznawania wyrazu wieloczłonowego

Przy operowaniu pojęciami opisywanymi jako wyrazy wieloczłonowe napotykamy na następujący problem: z jednej strony wyraz wieloczłonowy jest jednym konkretnym pojęciem i powinien być traktowany jako niepodzielna całość, a z drugiej strony – przy analizie tekstu najczęściej pierwszy etap analizy polega na dzieleniu tekstu na ciągi liter rozdzielone spacjami bądź innymi separatorami.

Z tego powodu informacje o wyrazach wieloczłonowych przechowywane są w słowniku w dwóch częściach:

1) w słowniku pojedynczych członów (analogicznym, jak dla wyrazów jednoczłonowych),
2) w słowniku wyrazów wieloczłonowych, zawierającym tylko numery `áÇ` członów.

Podział taki umożliwia łatwe wyodrębnianie w tekście wyrazów wieloczłonowych. Podana na rysunku 3 procedura rozpoznaje wyraz wieloczłonowy przekazany jako parametr procedury.

Pierwszy etap rozpoznania polega na zamianie tekstu na ciąg identyfikatorów. Następnie na podstawie tego ciągu dopasowywane są właściwe wyrazy wieloczłonowe. Ostatni etap polega na wykonaniu kontroli zgodności pozycji w wektorze odmiany członów. Unika się w ten sposób rozpoznawania błędnych form, na przykład „Polskie Koleje Państwowych". Procedura `wielo` zwraca ciąg identyfikatorów wyrazów wieloczłonowych dopasowanych do argumentu wywołania.

7. Podsumowanie

Jednymi z podstawowych operacji wykonywanych przy przetwarzaniu fleksyjnego języka naturalnego są: rozpoznawanie form tekstowych wyrazu oraz generowanie odmiany wyrazu. Zaproponowane rozwiązanie w postaci serwera leksykalnego realizuje ww. operacje. W znacznym stopniu może on odciążyć programistę tworzącego algorytm przetwarzający tekst w języku polskim od rozwiązania problemów związanych z cechą języka polskiego jaką jest fleksja. Niżej podano przykłady dziedzin i problemów, w których można zastosować serwer leksykalny.

- Aplikacja typu słownik języka polskiego pozwalająca rozstrzygać problemy związane z odmianą trudnych wyrazów. Przykłady tego typu programów dostępne są na stronie serwera leksykalnego.
- Analiza tekstów w celu uzupełniania słownika o nowe wyrazy, których jeszcze nie umieszczono w słowniku lub nowe wyrazy, które pojawiają się w języku.
- Poprawianie tekstu. Przykładem tego typu jest powstająca aplikacja będąca odpowiednikiem znanego z UNIX-a programu ispell. Dostęp do informacji o odmianie wyrazu pozwoli stworzyć nowe algorytmy uwzględniające kontekst czy nawet gramatykę zdania.
- Tworzenie konkordancji, indeksów słownictwa czy w koncu słowników frekwencyjnych oraz rozwiązywanie innych problemów językoznawstwa statystycznego.
- W dydaktyce związanej z nauczaniem przedmiotów związanych z przetwarzaniem języka naturalnego w szczególności z tworzeniem nowych algorytmów. Jest to pierwotny cel, dla którego powstał serwer leksykalny. W tym zastosowaniu serwer sprawdził się znakomicie czego dowodem są projekty studenckie dostępne na stronie serwera leksykalnego.

- Programy wspomagające nauczanie gramatyki języka polskiego i nauczanie języka na odległość.
- Współpraca z aplikacją rozpoznającą skanowane teksty w języku polskim czy też z aplikacją rozpoznającą mowę.
- Bardziej odległymi problemami są: określanie profilu tekstu, automatyczne streszczanie tekstu czy maszynowe tłumaczenie.

7.1. Przyszły rozwój serwera leksykalnego

Serwer można rozbudowywać o nowe usługi. Usługi te mogą pochodzić zarówno z rozbudowy bazy słownikowej, jak i nowych algorytmów zawartych w serwerze. Przykładami nowych usług są:

- generowanie listy propozycji dla błędnej formy wyrazu (usługa ta niezbędna jest przy zastosowaniach związanych z korektą tekstu),
- segmentacja wyrazów i algorytm podziału formy tekstowej przy przenoszeniu wyrazów w tekście,
- rozpoznawanie nowych form, których nie umieszczono w bazie słownikowej,
- dołączanie do wyrazów zawartych w bazie słownikowej dodatkowych informacji.

Załącznik A

Funkcje biblioteki `libls` dla języka ICON

Funkcja	:	`ls_open( serwer )`
Opis	:	Funkcja otwiera kanał komunikacyjny z serwerem. Otwarcie kanału musi poprzedzać wywołania wszystkich innych funkcji.
Argument	:	Jako parametr można przekazać do funkcji adres maszyny na której został uruchomiony serwer leksykalny. Jeżeli parametr zostanie pominięty nastąpi połączenie z serwerem na lokalnej maszynie.
Przykład	:	`ls_open( winnie.ics.agh.edu.pl )`
Uwaga	:	Jeżeli przez okres 1 minuty nie wystąpi żadne zapytanie serwer zamknie połączenie.
Funkcja	:	`ls_close()`
Opis	:	Funkcja zamyka kanał komunikacyjny z serwerem. Powinna zostać wywołana po zakończeniu korzystania z usług serwera.
Funkcja	:	`ls_info()`
Opis	:	Funkcja zwraca informacje o serwerze i słowniku.

Rezultat : Rekordinfo(wp, dp, ws, ls, dane, czas, jedno, wielo, razem, rek, pam, dysk)

Kolejne pola rekordu zawierają:

- numer wersji programu serwera;
- datę kompilacji programu serwera;
- numer wersji słownika;
- datę generacji słownika;
- nazwę pliku zawierającego bazę słownikową;
- liczony w sekundach czas działania serwera;
- liczbę wyrazów jednoczłonowych w bazie słownikowej;
- liczbę wyrazów wieloczłonowych w bazie słownikowej;
- łączną liczbę wyrazów w bazie słownikowej;
- liczbę rekordów;
- rozmiar struktur słownikowych w pamięci operacyjnej;
- rozmiar struktur słownikowych na dysku.

Funkcja : ls_num(forma)

Opis : Funkcja zwraca liczby wyrazów pasujących do zadanej formy.

Argument : Dowolna forma wyrazu.

Rezultat : Rekord num(p,n,s).

Kolejne pola zwróconego rekordu to liczby wyrazów znalezione w słowniku wyrazów pospolitych, słowniku nazw i słowniku składowych wyrazów wielosegmentowych.

Przykład : ls_num("Marek") -> num(2,3,0)

Uwaga : Jeżeli forma wyrazu rozpoczyna się wielką literą przeszukiwane są oba słowniki (wyrazów pospolitych i nazw).

Funkcja : ls_find(forma)

Opis : Funkcja zwraca ciąg numerów id wyrazów dopasowanych do danej formy.

Argument : Dowolna forma wyrazu.

Rezultat : Ciąg numerów id identyfikujących dopasowane wyrazy.

Funkcja : ls_desc(id)

Opis : Funkcja zwraca rekord opisujłcy wyraz o numerze id.

Argument : Numer id identyfikujący wyraz.

Rezultat : Rekord opisujący wyraz.

: Funkcja zwraca pojedynczy rekord o typie określonym przez klasę fleksyjną wyrazu. Typ zwracanego rekordu jest zależny od klasy fleksyjnej dopasowanego wyrazu. W bazie słownikowej występują następujące rekordy:

```
record class_a (typ,kl,id,kod,fh,et, rodz,nazwa,kat)
record class_b (typ,kl,id,kod,fh,et, aspekt,iml,im2)
record class_c (typ,kl,id,kod,fh,et, stopien,stl, st2,st3)
record class_d (typ,kl,id,kod,fh,et)
record class_e (typ,kl,id,kod,fh,et, rodz)
record class_f (typ,kl,id,kod,fh,et, stopien,stl,st2,st3)
record class_g (typ,kl,id,kod,fh,et, kat)
```

typ - typ wyrazu (1-pospolity, 2-nazwa, 3-składowa, 4-wieloczłonowy)
kl - klasa wyrazu (1-rzeczownik, 2-czasownik, 3-przymiotnik, ...)
id - numer identyfikujący wyraz
kod - kod szczegółowy
fh - forma polstawowa (hasłowa) wyrazu
et - etykieta fleksyjna wyrazu

Funkcja : ls_form(id,nr)

Opis : Funkcja zwraca formę tekstową wyrazu o numerze id występującą w wektorze odmiany na pozycji nr.

Argument : Id wyrazu oraz numer w wektorze odmiany.

Rezultat : Forma tekstowa wyrazu lub * jeżeli forma nie istnieje.

Przykład : Forma dopełniacza liczby pojedynczej wyrazu komputer to wartość funkcji ls_form(46777,2) -> komputera

---

Funkcja : ls_vec(id,forma)

Opis : Funkcja zwraca wektor zgodności wyrazu o numerze id dla formy przekazanej jako drugi argument.

Argument : Id wyrazu oraz jedna z form tego wyrazu.

Rezultat : Ciąg 0-1 o długości takiej jak długość wektora odmiany, jedynka oznacza zgodność drugiego argumentu z elementem wektora odmiany.

Przykład : ls_vec(46777, komputera) -> 01000000000000

Funkcja : ls_numw(id,poz)

Opis : Funkcja zwraca liczbę znalezionych wyrazów wieloczłonowych w których człon o numerze id występuje na pozycji poz.

Argument : Id członu oraz pozycja w wyrazie wieloczłonowym.

Rezultat : Liczba dopasowanych wyrazów wieloczłonowych.

Funkcja	:	ls_findw(id,poz)
Opis	:	Funkcja zwraca ciąg znalezionych id dla danego id na pozycji poz.
Argument	:	Id członu oraz pozycja w wyrazie wieloczłonowym.
Rezultat	:	Ciąg numerów id identyfikujłcych dopasowane wyrazy wieloczłonowe.
Funkcja	:	ls_elw(wid,poz)
Opis	:	Funkcja zwraca element wyrazu wieloczłonowego na pozycji poz.
Argument	:	Id wyrazu wieloczłonowego oraz pozycja w wieloczłonowym.
Rezultat	:	Id składowej wyrazu wieloczłonowego.
Funkcja	:	ls_lenw(wid)
Opis	:	Funkcja zwraca liczbę członów wyrazu wieloczłonowego
Argument	:	Id wyrazu.
Rezultat	:	Liczba członów wyrazu wieloczłonowego.

Załącznik B

Funkcje biblioteki `libls` dla języka C

```
#ifndef __LIBLS_H
#define __LIBLS_H

#define STR_LEN   80
#define STR_SHORT 30

struct info_res          /* zwracana przez ls_info                    */
{
 char wp[STR_SHORT];     /* numer wersji programu serwera             */
 char dp[STR_SHORT];     /* data kompilacji serwera                   */
 char ws[STR_SHORT];     /* numer wersji słownika                     */
 char ds[STR_SHORT];     /* data generacji słownika                   */
 Char dane[STR_SHORT];   /* plik słownika                             */
 int czas;               /* czas działania serwera                    */
 int jedno;              /* liczba wyrazów w słowniku                 */
 int wielo;              /* liczba wyrazów wieloczłonowych            */
 int razem;              /* łączna liczba wyrazów                     */
 int rek;                /* liczba rekordów w słowniku                */
 int pam;                /* rozmiar słownika w pamięci                */
 int dysk;               /* rozmiar słownika na dysku                 */
```

```
};

struct num_res          /* zwracana przez ls_num                    */
{
 int p;                 /* liczba wyrazów pospolitych               */
 int n;                 /* liczba nazw                              */
 int s;                 /* liczba skladowych                        */
};

union word_union        /* część zależna od części mowy             */
{
 struct class0          /* dla rzeczownika                          */
 {
  int rodzaj;           /* rodzaj rzeczownika                       */
  int kat;              /* kategoria nazwy                          */
 } cl0;
 struct class1          /* dla czasownika                           */
 {
  int aspekt;           /* aspekt czasownika                        */
  int im1;              /* id pierwszego imiesłowu                  */
  int im2;              /* id drugiego imiesłowu                    */
 } cl1;
 struct class2          /* dla przymiotnika                         */
 {                                                                   */
  int stopien;          /* stopień przymiotnika                     */
  int st1;              /* id stopnia 1                             */
  int st2;              /* id stopnia 2                             */
  int st3;              /* id stopnia 3
 } cl2;

 struct class4          /* dla zaimka                               */
 {
  int rodzaj;           /* rodzaj zaimka                            */
 } cl4;

 struct class5          /* dla przysłówka                           */
 {
  int stopien;          /* stopień przysłówka                       */
  int st1;              /* id stopnia 1                             */
  int st2;              /* id stopnia 2                             */
  int st3;              /* id stopnia 3                             */
 } cl5;

 struct class6          /* dla wyrazu nieodmiennego                 */
 {
  int kat;              /* kategoria wyrazu nieodmiennego           */
 } cl6;
};

struct desc_res         /* zwracana przez ls_desc                   */
{
```

```
  int typ;              /* typ wyrazu                          */
  int kl;               /* klasa fleksyjna wyrazu              */
  int id;               /* numer identyfikujłcy wyraz          */
  int kod;              /* kod wewnętrzny                      */
  char fh[STR_LEN];     /* forma hasłowa wyrazu                */
  char et[10];          /* etykieta fleksyjna                  */
  union word_union rec; /* część zależna od części mowy        */
};
/*********************************************************************/
int ls_open ( char * hostname );
int ls_close( int sock );
int ls_info( int s, struct info_res * out );
int ls_num ( int s, const char * inp, struct num_res * out );
int ls_find( int s, const char * inp, int * out, int * num );
int ls_desc( int s, int id, struct desc_res * out);
int ls_form( int s, int id, int nr, char * wy );
int ls_vec ( int s, int id, char * forma, char * out );
int ls_numw ( int s, int id, int poz, int * num );
int ls_findw( int s, int id, int poz, int * out, int * num );
int ls_elw  ( int s, int id, int poz, int * out );
/*********************************************************************/
char * op_klasa( int nr );
char * op_rodz( int nr );
char * op_kat( int nr );
char * op_stop( int nr );

#endif /* __LIBLS_H */
```

Załącznik C

Wyniki programu tworzącego statystykę przymiotników

leksykalny	32	jednoznaczny	3	wyższy	2
słownikowy	19	przetwarzający	3	rozpoznany	2
tekstowy	18	inny	3	pracujący	2
wieloczłonowy	15	równy	3	należący	2
dany	13	dalszy	3	odmienny	1
polski	13	składowy	3	komputerowy	1
następujący	11	inny	3	użyty	1
związany	10	znaleziony	3	niektóry	1
naturalny	9	poniższy	3	użyty	1
nowy	8	rozumiany	3	intensywny	1
komunikacyjny	8	określony	3	powyższy	1
fleksyjny	8	błędny	3	żeński	1

jeden	8	zgromadzony	2	przesłany	1
podstawowy	7	analogiczny	2	łatwiejszy	1
właściwy	7	najwyższy	2	gramatyczny	1
pospolity	6	podany	2	stanowiący	1
dostępny	6	stanowy	2	kolejny	1
możny	6	opisywany	2	stanowiący	1
typowy	6	biblioteczny	2	kolejny	1
obecny	6	techniczny	2	świadczący	1
nieodmienny	6	służący	2	szczegółowy	1
częsty	6	określający	2	szczególny	1
napisany	6	hutniczy	2	znany	1
automatyczny	5	ostatni	2	przesyłany	1
pierwszy	5	maszynowy	2	odseparowany	1
zielony	5	pozwalający	2	przyszły	1
opisany	5	lokalny	2	nieosobowy	1
niezbędny	5	wspólny	2		
jednoczłonowy	4	będący	2		
wszystek	4	źródłowy	2		
operacyjny	4	dotyczący	2		
opisujący	4	zrealizowany	2		
konkretny	4	prosty	2		
różny	4	usługowy	2		
poszczególny	3	zawarty	2		
realizujący	3	osobowy	2		
męski	3	dołączany	2		
zawierający	3	możliwy	2		
drugi	3	dodatkowy	2		
przechowywany	3	odległy	2		

Bardzo dziękuję panom Dariuszowi Żbikowi i Radosławowi Klimkowi za pomysły i pracę nad serwerem leksykalnym.

Literatura

[1] *Baza Fleksyjna Języka Polskiego*. http://www.icsr.agh.edu.pl/fleksbaz

[2] Griswold R.E., Griswold M.T.: *The Icon Programming Language*. Third Edition, Peer-To-Peer Communications, 1997

[3] *Icon Programming Language*. http://www.cs.arizona.edu/icon/

[4] Lubaszewski W.: *Gramatyka leksykalna w maszynowym słowniku fleksyjnym*. Raport MSJP-001, 1996

[5] *Posix Interface for Icon Programming Language*. http://www.drones.com/unicon

[6] Richard Stevens W.: *Programowanie zastosowań sieciowych w systemie UNIX*. WNT, 1998

[7] Sambor J.: *Słowa i liczby*. Zagadnienia językoznawstwa statystycznego. PAN 1972

[8] *Serwer Leksyksykalny Języka Polskiego*. http://winnie.ics.agh.edu.pl/ls