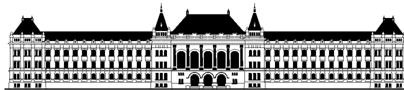


Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Távközlési és Médiainformatikai Tanszék

Ludányi Zsófia

MONDATHATÁROK ÉS RÖVIDÍTÉSEK GÉPI FELISMERÉSE ORVOSI SZÖVEGEKBEN

KONZULENSEK

Dr. Németh Géza

Dr. Prószéky Gábor

BUDAPEST, 2012

SZAKDOLGOZAT FELADAT

Ludányi Zsófia

mérnökhallgató részére

Mondathatárok és rövidítések gép felismerése orvosi szövegekben

A klinikai feljegyzések tárolása a kórházakban általában csupán archiválás céljából történik, s az így felhalmozódott adattömegek felhasználása jelenleg az egyes betegek kórtörténetének visszakeresésére korlátozódik. Nyelvtechnológiai eszközök segítségével azonban lehetséges lenne a szövegekben rejlő összefüggések, rejtett struktúrák feltárása. Angol nyelvterületen előrébb járnak az ilyen irányú kutatások, ezek alkalmazhatósága magyar nyelvű szövegre – a nyelv sajátosságai miatt – nem egyértelmű. Nem csupán a magyar nyelv nyelvtani sajátosságait kell figyelembe venni, hanem az orvosi szövegekre különösen jellemző nehéz, olykor hiányos szintaktikai szerkezeteket, idegen kifejezéseket, rövidítéseket is kezelni kell. Ezért időszerű feladat a más nyelveken már létező alkalmazások adaptálása, továbbfejlesztése, a magyar specialitások bevitele a rendszerbe.

Jelentős nehézség a klinikai dokumentumokkal kapcsolatban, hogy készítőik nem fordítanak hangsúlyt a helyes és a konzisztens fogalmazásra, tagolásra, helyesírásra. A következetlenség – többek között – a rövidítések használatában figyelhető meg. A szakdolgozat egyik célja, hogy megoldást adjon ezek egységes kezelésére.

A rövidítések nagy része pontra végződik, ez alapján működnek általában a rövidítéskereső algoritmusok. A pont azonban – magától értetődik – a mondat mint egység utolsó karaktere is egyben. A dolgozat másik kitűzött célja olyan eljárások kidolgozása, amelyek lehetővé teszik, hogy a mondathatárok is felismerhetők legyenek.

A mintaadatbázist a Semmelweis Egyetem 1. Sz. Szemészeti Klinikájának anyaga alkotja: különböző típusú kórlapok (anamnézis, státusz, javaslat, epikrízis). A

rendelkezésre álló anyag előfeldolgozása (zajcsökkentés) már megtörtént. Az adatbázis összesen 1239 db, szövegtípustól függően 1–13 kB-ig terjedő méretű szövegfájlból áll.

A hallgató feladatának a következőkre kell kiterjednie:

- Foglalja össze a külföldi (esetleg magyar nyelvű) szakirodalom alapján a mondathatár-egyértelműsítés és a rövidítések kezelésének eddigi eredményeit!
- Tekintse át, hogy a mondathatár-egyértelműsítés, valamint a rövidítések felismerésének szabályai – a magyar nyelv sajátosságait figyelembe véve – mely pontokon térnek el az egyéb nyelvekétől (leginkább az angolszászokétól).
- A felsorolt magyarnyelv-specifikus tulajdonságokat figyelembe véve dolgozzon ki olyan algoritmust, amely képes a rendelkezésre álló – előfeldolgozott (zajcsökkentett) – orvosi szövegekben megtalálni a mondathatárokat.
- Ugyanezek szempontok figyelembe vételével dolgozzon ki olyan algoritmust, amely a mondatokra bontott szövegben képes megtalálni a rövidítéseket! Keressen megoldást az azonos jelentésű, de eltérően jelölt rövidítések egységes kezelésére.
- Elemezze és értékelje a választott megoldásokat és a továbbfejlesztés lehetőségeit!
- Munkáját részletesen dokumentálja!

Tanszéki konzulens: Dr. Németh Géza docens

Külső konzulens: Dr. Prószéky Gábor egyetemi tanár (Pázmány Péter Katolikus Egyetem Információs Technológiai Kar)

Tartalomjegyzék

Összefoglaló **9**
Abstract **11**
1 A feladatkiírás pontosítása és részletes elemzése **12**
1.1 A dolgozat felépítése 12
1.2 A feladat értelmezése, a tervezés célja 13
1.3 A kifejlesztett szoftverek, egyéb fájlok 15
1.4 Fogalomtár 16
2 A feladatkiírás pontosítása és részletes elemzése **18**
3 Természetesnyelv-feldolgozás az orvostudományban **21**
3.1 Az orvosi szaknyelv jellemzői 21
3.2 Nyelvtechnológiai eszközöket használó egészségügyi alkalmazások 23
4 A mondathatár-meghatározás és a tokenizálás főbb problémái **25**
4.1 Mondathatár-felismerés 25
4.2 Tokenizálás 27
4.2.1 Tokenizálás nem angol nyelvű szövegekben 29
4.2.2 Magyar nyelvű szövegek tokenizálása 29
4.3 A rövidítések 30
5 Néhány magyar nyelvű tokenizáló szoftver **32**
6 Magyar nyelvű orvosi szövegek feldolgozására alkalmas mondathatár-meghatározó és tokenizáló megvalósítása **36**
6.1 A programnyelv megválasztása 36
6.2 Programozási szemlélet 36
6.3 A feladat megtervezése 37
6.4 Az egyes modulok kifejlesztése 40
6.4.1 Mondatokra bontás 40
6.4.1.1 Tervezés 40
6.4.1.2 Implementáció 43
6.4.1.3 Tesztelés, javítás 47
6.4.2 Tokenizálás 54
6.4.2.1 Tervezés 54

6.4.2.2 Implementáció 56
6.4.2.3 Tesztelés, javítás 59
6.4.3 Normalizálás 60
6.4.3.1 Tervezés 60
6.4.3.2 Implementáció 64
7 Az elkészült tokenizáló tesztelése, értékelése 67
7.1 A tesztelés módja 67
7.2 Fedés, pontosság 68
7.3 Összefoglalás, továbbfejlesztési lehetőségek 74
8 Köszönetnyilvánítás 76
Irodalomjegyzék 77
9 Függelék 79
9.1 Rövidítések listája 79
9.2 Felhasználói dokumentáció 83
9.3 Fejlesztői dokumentáció 85

Ábrajegyzék

6.1. ábra. A tervezés részfeladatai 37
6.2. ábra. Az egyes részfeladatok munkafolyamatai 38
6.3. ábra. A program folyamatábrája...39
6.4. ábra. Nyers szöveg (bemenet)...40
6.5. ábra. A tagAbbr() függvény folyamatábrája 46
6.6. ábra. Mondatokra bontott szöveg 53
6.7. ábra. Tokenekre bontott szöveg 55
6.8. ábra. A tokenizáló tervezésének első lépése: sortörés a rövidítések után 57
6.9. ábra. A tokenizer szavakra bontja a szöveget 58
6.10. ábra. A tokenizáló modul kimenete címkeeltávolítás után.............................59
6.11. ábra. A végső kimenet: címkézett tokenek, ID-val ellátott rövidítése............66
7.1. ábra. A kiértékelő program kimeneti képernyője 71
9.1. ábra. A medTokenizer indítása parancssorból..84
9.2. ábra. A medTokenizer működés közben 85

Táblázatjegyzék

1.1. táblázat. A kifejlesztett forrásfájlok és a hozzájuk tartozó adatfájlok............15

7.1. táblázat. A kiértékelés eredményei 71

9.1. táblázat. A medTokenizer.py függvényei...87

HALLGATÓI NYILATKOZAT

Alulírott **Ludányi Zsófia**, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2012. május 05.

....

Ludányi Zsófia

Összefoglaló

A legtöbb kórházban csupán archiválási célból tárolják a klinikai feljegyzéseket. Nyelvtechnológiai eszközök segítségével azonban lehetséges lenne a szövegekben rejlő összefüggések feltárása, s ezáltal egy orvosi-klinikai korpusz létrehozása. A kutatás, amelyhez a dolgozat kapcsolódik, ezt tűzte ki hosszú távú céljául.

A kutatás anyagát képező nyers szöveghalmazt a Semmelweis Egyetem egyik szemészeti klinikájának kórlapjai alkotják. A klinikai dokumentumok feldolgozásának első lépése azok strukturálása és normalizálása volt. A következő lépés az, hogy az előfeldolgozás során kinyert értékes szövegeket kisebb-nagyobb egységekre: mondatokra, illetve szavakra, írásjelekre stb. bontsuk. Ezt a folyamatot tokenizálásnak nevezzük.

Léteznek ugyan magyar nyelvű szövegek feldolgozására alkalmas tokenizálók (Magyarlánc, Huntoken), de ezek csak általános, köznyelvi szövegeken lefuttatva működnek hatékonyan, nem szaknyelvi szövegek feldolgozására tervezték őket.
Célunk egy olyan szabály alapú tokenizáló szoftver megtervezése és megvalósítása, amely képes az előfeldolgozott orvosi dokumentumok szegmentálására. A program nagy hangsúlyt fektet a klinikai szövegekben előforduló rövidítések megfelelő kezelésére. Az orvosi szövegekre ugyanis különösen jellemző – a sok idegen nyelvű kifejezés mellett – a speciális, csak a szaknyelvben használatos rövidítések gyakori előfordulása. A klinikai dokumentumokra általában jellemző következetlenség nemcsak a szövegtagolásban, helyesírásban, hanem a rövidítések használatában is megfigyelhető. A gyakorta használt kifejezések rövidítése személyenként, sőt olykor egy dokumentumon belül is változhat. Nagyon fontos, hogy a szoftverünk képes legyen az azonos jelentésű, különböző alakú rövidítések felismerésére és egységes kezelésére.

A tervezés első lépéseként szabályokat fogalmaztunk meg arra nézve, hogy mikor beszélhetünk mondat-, illetve szóhatárról. A mondathatár-felismeréshez ismernünk kellett a szövegben előforduló rövidítéseket. Ezeket szintén szabályok megfogalmazásával nyertük ki a szövegből. A mondathatárok felismerése és megjelölése után következett a tokenekre bontás, végül az rövidítések egyedi

azonosítóval való ellátásával biztosítottuk azok egységes kezelését: az azonos jelentésű, különböző alakú rövidítések ugyanazt az ID-t kapták.

A kifejlesztett szoftver 90%-on felüli pontossággal felismeri a mondathatárokat és a rövidítéseket a bemenetként kapott orvosi szövegekben.

Abstract

Clinical records are merely stored for more than the purpose of archiving in the majority of hospitals nowadays. However, new contextual information could be retrieved from these texts using natural language processing devices, thus creating a medical corpus. This is the long-term aim of the research this paper relates to.

The research material is based on the medical charts of the Ophthalmological Department of Semmelweis University. Documents were structured and normalized during the pre-process constituting the raw material for the current study. The texts have were broken down into units of variable size: sentences, words and punctuation marks. This process is known as tokenizing.

Former tokenizers developed for the Hungarian language (like Magyarlánc or Huntoken) were not designed for medical terminology.

The aim was to create a rule-based tokenizer that is able to segment prepocessed medical texts and manipulate abbreviations effectively. The latter is of extreme significance due to the high frequency of sublingual abbreviatons in medical texts. Furthermore, there is great inconsistency in ortography and segmenting in the documents. Frequently used expressions show individual and intertextual variability. It is crucial that the software identifies and manages the morphological differences in a unitary fashion.

The planning began by defining rules regarding sentence and word boundaries. For sentence boundary disambiguation abbreviations had to be considered. After tokenizing, each abbreviation was tagged with a unique identifier defining its meaning.

The resultant software operates with a 90% precision rate in detecting sentence boundaries and abbreviations.

1 A feladatkiírás pontosítása és részletes elemzése

1.1 A dolgozat felépítése

Az 1.2. fejezet értelmezi a feladatot, és szól arról is, miért időszerű egy magyar nyelvű orvosi szövegek feldolgozására alkalmas tokenizáló létrehozásával foglalkozni. Az 1.3. fejezetben röviden ismertetjük a szakdolgozathoz kapcsolódóan fejlesztett szoftver forrásfájljait, illetve egyéb, a félév során létrehozott állományokat. Az 1.4. alszakasz tisztázza a dolgozatban előforduló legfontosabb fogalmak jelentését.

A 2. fejezetben részletesen ismertetjük a félév során megvalósítandó alkalmazás céljait, követelményeit. Sor kerül a munka környezetének bemutatására is. Részletesen ismertetjük azt a szövegkorpuszt, amelyet a készülő program bemenetül fog kapni. Kitérünk a korpusz nagyságára, jellemző tulajdonságaira, valamint arra, hogy milyen előfeldolgozási lépéseken ment keresztül.

A 3. fejezet célja, hogy az olvasó betekintést nyerjen az orvosi szaknyelvnek a köznyelvi szövegtől jelentősen eltérő jellemzőibe:. A 3.1-es alfejezetben bemutatunk néhány olyan – külföldön használt – egészségügyi alkalmazást, amelyek nyelvtechnológiai eszközökön alapulnak.

A 4. fejezet a mondathatár- és rövidítésfelismeréssel kapcsolatos kérdésekre tér ki. Az angolszász nyelvű szakirodalom ismertetése után kitérünk a magyar nyelvű szövegek tokenizálásának főbb kérdéseire, rávilágítunk a különbségekre az angol nyelvű tokenizálókhoz képest.

Az 5. fejezetben bemutatunk néhány magyar nyelvű szöveg tokenizálására alkalmas szoftvert, ismertetjük azok jellemzőit, a szaknyelvi szövegekhez való viszonyukat.

A 6. fejezet ismerteti a félév során elkészített megoldás megtervezésének és megvalósításának részletes leírását, a tervezés során felmerült döntési lehetőségek értékelését és a választott megoldások indoklását. Szólunk az egyes részfeladatok céljáról, megtervezésükről, megvalósításukról, az esetlegesen felmerülő javításokról.

A 7. fejezetben kiértékeljük a fejlesztett alkalmazást. Részletesen kitérünk a tesztkörnyezetre, a tesztelés módjára, és számadatokkal is jellemezzük a program hatékonyságát. Ismertetjük a kijavított és nem kijavított hibákat, valamint a továbbfejlesztési lehetőségeket.

A dolgozathoz csatolt függelék tartalmazza a felhasználói és fejlesztői dokumentációt, valamint a feladat nyersanyagát képező orvosi szövegekből kinyert rövidítéseket, amelyek a rövidítéskereső modul lexikonja tartalmaz.

1.2 A feladat értelmezése, a tervezés célja

Jelen munka a Pázmány Péter Katolikus Egyetem Információs Technológiai Karán folyó kutatáshoz kapcsolódik, melynek hosszútávú célja egy olyan keretrendszer készítése, amely orvosi dokumentumokat feldolgozva segítheti a klinikai szakembereket új összefüggések feltárásában [1]. Az ilyen irányú kutatások hazánkban egyelőre még gyerekcipőben járnak, ellenetétben az angol nyelvterülettel, ahol már jelentős lépések történtek ezirányban. (Erről bővebben a következő fejezetben szólunk.)

A tervezés célja egy olyan alkalmazás létrehozása, amely a képes a rendelkezésre álló, nem tokenizált és strukturálatlan klinikaiszöveg-halmaz alacsony szintű feldolgozására. Alacsony szintű feldolgozás alatt olyan eljárások fejlesztése értendő, amelyek elemi egységekre bontják fel a bemeneti szöveget. Az elemi egység jelen esetben legtöbbször szavakat jelent, de emellett lehet szám is, vagy akár központozó jel (pont, vessző, kettőspont stb.), amelyeket szintén külön egységként kezelünk. Ezeket az elemi egységeket tokeneknek nevezzük, azt a folyamatot pedig, amely az elemi egységekre bontást végzi, tokenizálásnak. Más szóval a tervezés célja egy – magyar nyelvű – orvosi szövegek feldolgozására alkalmas tokenizáló program (tokenizer) megvalósítása. Ez az alacsony szintű feldolgozás jó alapot nyújthat a későbbiekben a klinikai szövegekben rejlő összefüggések, rejtett struktúrák feltárásához.

A kórlapokra és egyéb orvosi feljegyzésekre általában jellemző, hogy készítőik nem fordítanak kellő hangsúlyt a fogalmazásra, a szöveg megfelelő tagolásra, valamint a helyesírásra. Különösen jellemző ez a szaknyelvi rövidítések használatára. Igen gyakran előfordul, hogy egy szót – akár ugyanazon szövegen belül – más formában rövidítenek. Fontos és kiemelt célja a megvalósítandó szoftvernek, hogy ne csupán felismerje a

rövidítéseket, hanem valamilyen módon összefogja, egységesen kezelje az azonos jelentésű, többféle alakú eseteket.

A rövidítések felismerésével szorosan összefügg a mondathatárok felismerése. Fontos, hogy egy pontról meg tudjuk állapítani, mikor jelöli egy mondatnak a végét, és mikor egy rövidítést, esetleg mindkettőt. Hogy ezt meg tudjuk állapítani, a szoftvert úgy kell megtervezni, hogy képes legyen a mondathatárok felismerésére, és ennek megfelelően jelezni is a határokat a szöveg megfelelő szegmentálásával. Szoftverünk tehát nem csupán kis egységekre (szavakra, számokra, írásjelekre) bont, hanem nagyobb logikai egységekre: mondatokra is.

Joggal merül fel a kérdés: létezik-e már olyan szoftver, amely képes magyar nyelvű szövegek tokenizálására? A válasz igen. Az egyik legismertebb (szintén szabály alapú) tokenizáló a BME Média Oktató és Kutató Központ által fejlesztett HunToken. Mint később látni fogjuk, az általános célú (köznyelvi) szövegek és a szakszövegek, jelen esetben az orvosi szövegek szókincse és egyéb tulajdonságai oly mértékben különböznek, hogy a köznyelvi szövegekre fejlesztett (és azon nagy pontosságot elért) tokenizáló szoftverek már korántsem működnek megfelelően olyan speciális szövegeken, mint például egy orvosi dokumentum. Ez az, ami indokolttá teszi egy speciálisan orvosi szövegekre kifejlesztett tokenizáló program megalkotását.

1.3 A kifejlesztett szoftverek, egyéb fájlok

A féléves munka során született fájlok az 1.1-es táblázatban láthatók.

Név	Fájltípus	Méret, bájt
medTokenizer.py	Python forrásfájl	8048
Rules.py	Python forrásfájl	1648
compare.py	Python forrásfájl	7929
wordList.txt	Szövegfájl	3013
doktorok.txt	Szövegfájl	713
gyogyszerek.txt	Szövegfájl	42 352

1.1. táblázat. A kifejlesztett forrásfájlok és a hozzájuk tartozó adatfájlok

medTokenizer.py: Maga a tokenizáló szoftver. Három fő modult tartalmaz: a mondatokra bontásért felelős modult, a tokenekre bontó modult, illetve a rövidítések felismerésére, megjelölésére szolgáló modult. Indításkor paraméterként kapja azt a könyvtárnevet, amely a feldolgozandó szövegeket tartalmazza; második paraméterként azt a könyvtárnevet, ahová a feldolgozott fájlokat kell helyeznie (ha ugyanazt a könyvtárat adjuk meg, mint ahol a feldolgozandó fájlok vannak, felülírja őket). A harmadik paraméter egy lista, amely lexikonként szolgál, a rövidítéseket tartalmazza. (Esetünkben ez a wordList.txt nevű fájl, de tetszőleges szöveges fájl megadható.)

Rules.py: Szabály alapú tokenizálóról lévén szó, a program kizárólag előzetesen megfogalmazott és reguláris kifejezéssé alakított szabályok alapján végzi el a szegmentálást, címkézést. A kód könnyebb olvashatósága érdekében a szabályokat külön Python-modulba helyeztük.

compare.py: A tokenizáló program hatékonyságát kiértékelő szoftver. Paraméterként két fájlt vár: a referenciafájlt (gold standard), amelyhez a második paraméterként várt tesztelendő fájlt (amelyet a tokenizáló állított elő) hasonlítjuk. A kiértékelő szoftver

kiszámolja a fedést, pontosságot, F-mértéket és a pontossággal súlyozott $F_β$-mértéket, és kiírja azokat a képernyőre.

wordList.txt: A lexikon, amely alapján a tokenizáló program dolgozik. Egyszerű szövegfájl, amelyet a tokenizáló szoftver paraméterként kap indításkor. A lexikon (egyelőre) több száz, csak a feldolgozandó dokumentumokban előforduló rövidítést tartalmaz betűrendben, reguláris kifejezéssé alakítva.

gyogyszerek.txt: A dokumentumokban előforduló gyógyszereket tartalmazó szövegfájl. Tartalmazza az összes, gyógyszertárban kapható vényköteles és nem vényköteles gyógyszer nevét. Nem tartalmazza a gyógyszertárban nem kapható (tipikusan kórházi felhasználásra gyártott) termékeket, illetve a homeopátiás szerek neveit [2].

doktorok.txt: A dokumentumokban előforduló családneveket (orvosok vezetékneveit) tartalmazó szövegfájl. A későbbiekben feltétlenül bővítendő.

1.4 Fogalomtár

A természetes nyelvek mesterséges feldolgozásával kapcsolatos főbb fogalmak az alábbiak:

Természetesnyelv-feldolgozás (natural language processing, NLP, humán nyelvtechnológia): Az informatika olyan résztudománya, amely a természetes nyelvi jelenségeket, problémákat az informatika oldaláról közelíti meg [3]. Ide tartozik minden olyan tudományág, amely természetes nyelvek mesterséges feldolgozásával foglalkozik. (Nem programnyelvek, egyéb formális nyelvek elemzéséről van tehát szó.) Ma az informatika területén minden szövegekkel foglalkozó feladat alkalmaz valamilyen NLP-eszközt. Ezek közül a leggyakoribbak: párbeszédes rendszerek, információ-visszakeresési alkalmazások, adatbányászati feladatok, kérdés-válasz rendszerek, szövegkivonatolás, gépi fordítás.

Reguláris kifejezés: Szintaktikai szabályok szerint felépülő, speciális értékkészlettel rendelkező sztringhalmazt reprezentáló sztring, amely lehetőséget nyújt a szabályokkal leírható, nyílt – azaz nem felsorolható – tokenosztályok felismerésére. Ez annak köszönhető, hogy a reguláris szintaxis olyan elemeket is tartalmaz, amelyek ciklikus

mintaillesztést is lehetővé tesznek. Használatuk igen elterjedt, közvetlenül alkalmazza pl. a Unix operációs rendszer grep parancsa, valamint számos programozási nyelv (pl. Python, Perl) is támogatja [4].

Token: Egy karaktersorozat konkrét dokumentumbeli előfordulása. Egy token lehet szó (pl. *alma*), több szóból álló kifejezés (pl. *Bartók Béla*, *Kispál és a Borz*), egyéb – szintaktikailag egy egységnek tekinthető – kifejezés, pl. egy dátum (*1848. március 15.*), de akár egy írásjel is (pl. egy mondatvégi pont).

Tokenizálás: Az a szövegfeldolgozás lépés, melynek során a szöveget tokenek sorozatára bontjuk.

Névelem (named entity): Minden olyan tokensorozat, amely a világ valamely entitására egyedi módon referál. Ide tartoznak a tulajdonnevek, az azonosítók, a telefonszámok, az e-mail címek stb. [4]

2 A feladatkiírás pontosítása és részletes elemzése

Az orvosi szövegek feldolgozásával foglalkozó kutatás anyagát a Semmelweis Egyetem klinikáinak digitalizált dokumentumai képezik. A nagy mennyiségű dokumentumhalmaz többféle orvosi szakterületről származik: belgyógyászat, sebészet, szemészet stb. Az eltérő szakterületek dokumentumainak a felépítése is eltérő, így célszerű volt szakterületenként szétválasztani a szövegek feldolgozását. Elsőként a szemészeti dokumentumok feldolgozása indult el, melynek eredményei alkalmazhatóak lesznek – kisebb-nagyobb módosításokkal – más szakterületek szövegeinek feldolgozására. Nem titkolt célja a projektnek, hogy hosszú távon általános orvosi szövegek feldolgozására is képes legyen a rendszer – a korábbi eredmények felhasználásával.

Jelen feladat célja, hogy a szemészeti dokumentumok egyik jelentős (és ezidáig csak érintőlegesen említett) problémakörét, a rövidítések (és az ehhez szorosan kapcsolódó egyéb nyelvi egységek) területét körbejárjuk, és a felmerülő (később részletezett) kérdésekre egy-egy lehetséges megoldást találjunk. A problémás kérdések részletesebb vizsgálata előtt célszerű bemutatni azt a környezetet, amelyen a munka folyik, valamint a rendelkezésünkre álló eszközöket.

Adott egy nagy mennyiségű, strukturálatlan dokumentumhalmaz, amely a Semmelweis Egyetem egyik szemészeti klinikájának kórlapjait tartalmazza. Pontosan 1239 darab egyszerű szövegfájlból áll az adatbázis, az állományok között vannak egészen kis méretű (1-2 kB) és hosszabb (10-13 kB) dokumentumok is. A feladat megkezdésekor a szövegek előfeldolgozása már megtörtént, az említett dokumentumhalmaz az alább részletezett előfeldolgozási folyamatok eredményeként állt elő. Az előfeldolgozás alatt a következők értendők:

- A korpusz szöveges részei el lettek különítve a nem szöveges részektől [1]. A nem szöveges részek alatt a digitalizálás során keletkezett, számunkra értéktelen információ értendő (zaj). Ilyenek például a laboreredmények, számértékek, elválasztó karaktersorozatok, valamint olyan szövegrészek, amelyek csupán rövidítéseket, speciális jeleket tartalmaznak.

- A dokumentumok azonosítása megtörtént, a tartalom és a metaadatok elkülönítésre kerültek. Tartalmi egység például a fejléc, diagnózisok, beavatkozások, javaslat, státusz, panasz stb. A metaadatok közé tartozik – többek között – a dokumentum típusa: anamnézis, státusz, javaslat, epikrízis.
- Megtörtént a dokumentumok normalizálása, ennek első lépéseként a helyesírás ellenőrzése. Ez nem csupán a magyar helyesírás szabályaihoz való igazodást célozta meg, hanem egyéb, a szaknyelvi szöveg sajátosságaiból adódó hibajelenség kezelését is megkísérelte. Az egyszerű elgépelésektől, félreütésektől kezdve a központozás hiányán át az olyan orvosinyelv-specifikus helyesírási kérdésig, mint a görögös-latinos vs. magyaros vs. hibrid írásmód kérdése, igen széles a szakterületre jellemző nyelvi sajátosságokból adódó hibajelenségek palettája. Ide tartozik a rövidítések kérdésköre is: olyan speciális rövidítésekről lévén szó, amelyeknek sem jelentésük, sem jelölésmódjuk nem általánosítható. A megvalósított helyesírási alapalgoritmus eredményeként előállt egy olyan szöveghalmaz, amelyek alapján „pontosabb hibamodell építhető egy továbbfejlesztett rendszer betanításához" [1].

Az így létrehozott, előfeldolgozott dokumentumhalmaz képezte a rendelkezésünkre álló nyersanyagot. A feladatunk az volt, hogy az így előállított orvosi szövegeket egységekre bontsuk: először nagyobb összefüggő egységekre, mondatokra, majd alacsonyabb szintűekre: tokenekre. Korábban már említettük, hogy a helyesírás-javításhoz elengedhetetlen az, hogy az algoritmus képes legyen felismerni és helyesen kezelni a rövidítéseket, tekintettel azok igen nagy számára és speciális mivoltukra. Tekintsük például az alábbi szövegrészeket: *„szemhéjszél idem, mérs. inj. conj, l.sin."* vagy *Vitr. o.s. (RM) abl. ret. miatt"*. Láthatjuk, hogy olyan nagy számban fordulnak elő a szövegben rövidítések, hogy kezelésükre feltétlenül szükséges nagyobb figyelmet fordítani.

A magyar nyelvben előforduló leggyakoribb rövidítések dokumentálva vannak az akadémiai helyesírási szabályzat releváns szabálypontjában, illetve egyéb helyesírási szótárakban, kézikönyvekben. Noha létezik orvosi helyesírási szótár, az orvosi nyelvi rövidítésekre nincs olyan egységes normarendszer, mint a köznyelviekre. Talán ez is oka lehet annak, hogy az orvosi szövegekben előforduló rövidítések jelölése egyáltalán nem következetes. Ugyanazon fogalmakat többféleképpen rövidítenek – osztálytól,

szakterülettől, személytől függően, de arra is bőven akad példa, hogy akár egyazon szövegen belül, egyazon személytől származó rövidítések is különböznek, noha ugyanazt a fogalmat kívánja kifejezni az adott személy. Jól példázza ezt a szemészeti szövegben előforduló *oculus dexter* ('jobb szem') kifejezés, amely az alábbi módokon fordul elő a dokumentumhalmazban: *o.d.* (szóköz nélkül), *o. d.* (szóközzel), *o. d* (második pont nélkül), *od* (egybeírva, pontok nélkül), *o.dex*, *o. dex*, *o.dex.*, *odex* stb. A lehetőségek száma határtalan. Éppen ezért szükséges, hogy tudassuk a felhasználóval, itt valójában ugyanarról a fogalomról van szó.

Ahhoz, hogy a rövidítéseket felismerjük, szükséges annak ismerete is, hogy hol vannak a szövegben a kisebb-nagyobb egységeknek: a tokeneknek és a mondatoknak a határai. Főként a mondathatárok egyértelműsítése fontos, hiszen vannak olyan esetek, amikor egyáltalán nem egyértelmű, mikor kezdődik új mondat, főként, ha rövidítésre végződik a feltételezett mondat. A mondathatár- és rövidítésfelismerés szorosan összefügg egymással. A tervezett alkalmazásnak képesnek kell lennie mindezek felismerésére és megjelölésére.

Mint ahogy a nyelvtechnológia más területei is (gépi fordítás, szófaji egyértelműsítés, beszédszintézis), egy tokenizálási feladat is többféle nézőpontból közelíthető meg. A két legfontosabb a szabály alapú és a statisztikai alapú megközelítés. Mindkét módszernek megvannak az előnyei és hátrányai. Egyszerű kivitelezhetősége, a hibajavítás egyértelműsége miatt a szabály alapú megközelítés mellett döntöttünk. Léteznek hibrid megoldások is, amely egyszerre alkalmaznak bizonyos jelenségeket leíró szabályokat, illetve statisztikai módszereket. Valószínűsíthető, hogy hosszú távon a többféle nézőpontú megközelítés lesz a legcélravezetőbb (elég azt megemlíteni, hogy a fejezet elején említett előfeldolgozási lépések jelentős része statisztikai módszerekkel történt). Jelen feladat megoldásához elégségesnek érezzük a tisztán szabály alapú módszerek alkalmazását, tekintve, hogy a szövegben előforduló rövidítések, mondathatárok jellegzetességei szabályokkal jól leírhatók, továbbá mivel a statisztikai rendszer kialakításához nem rendelkezünk elegendő mennyiségű címkézett adattal.

3 Természetesnyelv-feldolgozás az orvostudományban

Az alábbiakban ismertetjük, hogy miképpen kamatoztathatjuk a természetes nyelvek feldolgozására irányuló kutatásokat az orvostudomány területén. Kitérünk az orvosi nyelvnek mint szaknyelvnek azokra a legfontosabb jellemzőire, melyeket feltétlenül figyelembe kell venni ilyen témájú szövegek feldolgozásakor. Az alfejezetben néhány olyan orvosinformatikai rendszert is megemlítünk, melyek nyelvtechnológiai eszközöket alkalmaznak.

3.1 Az orvosi szaknyelv jellemzői

Az elmúlt évtizedekben a technológia fejlődésének köszönhetően az egészségügyben is megkezdődött az adminisztrációs rendszerek számítógépesítése. E folyamat nem csak a szolgáltatás minőségének javulását, az adminisztratív teendők gyorsulását, hanem egyúttal a költségek csökkentését is magával hozza, így mindenképpen üdvözlendő [5].

Az automatizálási törekvések az egészségügy több területén folynak, többek között a betegek diagnózisának, kezelésének menedzselésében, a kutatási eredmények közzétételében, a minőségbiztosításban, az erőforrásokkal való gazdálkodásban stb. A számítógépes alkalmazásoktól – bármilyen területről legyen is szó – elvárható, hogy megbízható módon férjenek hozzá a klinikai adatokhoz. Ahhoz, hogy az adatokhoz való hozzáférést lehetővé tegyük, az adatoknak strukturáltnak kell lenniük, mivel az egyszerű szöveges információ túl változatos ahhoz, hogy könnyen visszanyerhető legyen. Az orvosi szövegekre jellemző változatosságnak, az egységesség hiányának oka, hogy az egészségügyben dolgozók – mint ahogy azt a későbbiekben látni is fogjuk – igen eltérő módon készítenek jegyzeteket (gondoljunk csak például az idegenes vs. magyaros helyesírás kettősségére vagy a szaknyelvi fogalmak rövidítéseinek többféle jelölésére!). Mindemellett a szavak többértelműsége is problémát jelent az orvosi szövegekben, mivel a szavak jelentését jelentősen befolyásolja a szövegkörnyezet. Ezen problémákra megoldást jelenthetnek a természetesnyelv-feldolgozó rendszerek (natural language processing, NLP).

Míg nekünk, embereknek a természetes nyelvet megérteni magától értetődő dolog, a számítógépek számára már korántsem az. Mi jellemzi a természetes nyelvet?

Elsősorban jó kifejezőképesség, változatosság, többértelműség, ugyanakkor bizonytalanság is. Friedman és Hripcsak [5] az alábbi angol nyelvű példákat hozza fel annak demonstrálására, hogy ugyanaz a fogalom többféleképpen fejezhető ki: a kongesztív szívelégtelenséget egyaránt említik *congestive heart failure*-ként, illetve – rövidítve – *CHF*-ként. Megfigyelhető ugyanakkor ennek fordítottja is, amikor egy fogalom – attól függően, hogy milyen szövegkörnyezetben fordul elő – több különböző jelentéssel bírhat. Angol nyelvű példa: a *discharge* szó mást jelent a *discharge from hospital* és *discharge from wound* kifejezésekben ('a kórházból való elbocsátás', illetve: '[valamiféle folyadék] szivárgása a sebből'). Sokszor nem egyértelmű az sem, hogy pontosan milyen viszony van a szavak között. Pl. a *no acute infiltrate* kifejezésből nem derül ki pontosan, hogy a tagadószó (*no*) az *acute* jelzőre, avagy az *infiltrate*-re mint jelzett szóra vonatkozik. Miről van tehát itt szó? Az illető páciensnek nincs akut beszivárgása, vagy pedig beszivárgása van, amely nem akut? Így – kiragadva a kontextusából – nyilván nem tudunk válaszolni a kérdésre, ehhez tudnunk kellene, milyen szövegkörnyezetben fordult elő.

Pontosan ezekre a problémákra jelenthet megoldást a természetesnyelv-feldolgozásnak orvosi szövegekre való kiterjesztése. Az NLP-rendszerek – egy jól definiált szótár terminológiáját használva – kódolják az információt, jelölik a fogalmak közti viszonyokat egy egyértelmű formális struktúrában, feltüntetik a kontextuális információkat, és formálisan reprezentálják a bizonytalan fogalmakat is.

A természetes nyelvek megértése magában foglalja egyrészt a szintatika megértését, vagyis a mondatok szerkezetét (mely szavak lehetnek alanyok, tárgyak, állítmányok); valamint a szemantika megértését, vagyis a szavak jelentését, illetve azt, hogy azok hogyan kombinálódnak egymással a mondat jelentését formálandó. Ezenkívül fontos, hogy az adott területről (amelyről szó van) is rendelkezzünk általános tudással (gondolunk itt arra a triviálisnak tűnő dologra, hogy például tudnunk kell, hogy a tüdőgyulladás egy betegség).

Bizonyos szempontból könnyebb dolgunk van az orvosi szaknyelvi szövegek feldolgozásával, mint az általános szövegekével. Ennek oka, hogy az orvosi nyelv – mint a szaknyelvek általában – rétegnyelvet alkot. David Crystal szerint az orvosi nyelv lényegi részét – sok más kifejezés mellett – az anatómiai vagy élettani kifejezések

alkotják [1]. Azonban nem csupán a szókészlet miatt nevezhető rétegnyelvnek az orvosi nyelv. Harris javasolta először a rétegnyelvi nyelvtan (sublanguage grammar) fogalmát, amelyet Sager ültetett át a nyelvtechnológiai terminológiába [5]. Ez alatt az értendő, hogy az orvosi nyelv jóval definiáltabb, mint a köznyelv: témaköre jóval behatároltabb, ezen belül kevesebb változatosság, többértelműség és komplexitás jellemzi. (Azonban ez nem jelenti azt – mint majd a későbbiekből kiderül –, hogy orvosi szöveget feldolgozni egyszerűbb feladat lenne, mint egyszerű köznyelvi szöveget.)

Az orvosi szövegek feldolgozása során, miután megtörtént az információ kinyerése, a következő lépés az, hogy a kinyert információt a későbbi felhasználás érdekében eltároljuk valamilyen jól definiált formátumban. Ez történhet fogalmi gráf formájában vagy XML-ben.

3.2 Nyelvtechnológiai eszközöket használó egészségügyi alkalmazások

A továbbiakban megemlítünk – a teljesség igénye nélkül – néhány fontosabb kezdeményezést az egészségügyben, amelyek nyelvtechnológia eszközeit alkalmazzák.

A Linguistics String Project (LSP) vezetője, Sager úttörőnek számít mind az általános, mind az orvosinyelv-feldolgozásban. A vezetésével kifejlesztett rendszernek igen széleskörű szintaktikai és szemantikai komponensei vannak, beleértve a zárójelentéseket, folyamat jegyzeteket (progress note) és radiológiai jelentéseket.

A SPRUS rendszer egy speciális célú radiológiaiszöveg-feldolgozó, és többek között egyik volt azon rendszereknek, amelyek egy működő klinikai információs rendszernek – a HELP rendszernek – moduljaként működtek. Egy későbbi verzió, a SYMTEXT szintaktikai komponenssel is rendelkezik.

A MedLEE rendszer a New York-i Presbiteriánus Kórház klinikai információs rendszerének független moduljaként működik. Két különböző hangfelismerő rendszerrel építették egybe.

A genfi kórházban olyan többnyelvű NLP-rendszert fejlesztettek ki, amely képes francia, angol és német nyelvű szövegek feldolgozására. A fejlesztők egy normalizált, nyelvfüggetlen rendszer fejlesztésére törekedtek. A rendszer működése – modellezési nehézségek miatt – a belgyógyászati zárójelentésekre korlátozódik.

A MENELAS nevű rendszert egy olyan konzorcium készítette, amely a zárójelentések információihoz való jobb hozzáférést tűzte ki célul. A szívkoszorúér-betegségek területén két prototípus rendszert fejlesztettek ki: egy dokumentindexelési rendszert (melynek részei elérhetőek franciául, angolul és hollandul) és egy tanácsadó alkalmazást, amely az indexelő rendszer segítségével biztosítja a felhasználóknak a dokumentumokban levő információkhoz történő hozzáférést.

Németországban is számos NLP rendszer létezik, amelyeket a 90-es évek végén kezdtek el fejleszteni. Ezek közül említést érdemel a göttingeni György Ágoston Egyetemen fejlesztett MediTas nevű rendszer, a hamgurgi egyetemen fejlesztett MeTexA-t, valamint a freiburgi egyetemi kórházban fejlesztett MedSYNDIKATE.

A japán Chiba Egyetemi Kórházzal kapcsolatban levő csoport fejlesztése – a New England Journal of Medicine jelentéseire alapozva – egy olyan prototípus rendszer, amely a riportok találatait fordítja SNOMED és ICD9 kódokra.

Magyarországon is voltak orvosi szakértői rendszerek fejlesztésére irányuló kutatások. Így például magyar kutatócsoport fejlesztette ki dr. Vámos Tibor vezetésével a NES fejlődésneurológiai szakértő rendszert, melynek célja a csecsemőkori idegrendszeri sérülések korai felismerésének segítése [7].

4 A mondathatár-meghatározás és a tokenizálás főbb problémái

Tekintve, hogy jelen dolgozat az orvosi szövegek feldolgozására fókuszál – különös tekintettel a mondathatárok és a rövidítések felismerésére –, célszerű bemutatni az ezzel kapcsolatos eddigi kutatásokat, elért eredményeket. Mivel az orvosi szövegek ilyen irányú feldolgozásáról nem született még jelentős összefoglaló mű a nemzetközi szakirodalomban sem, célszerű áttekinteni az általános – nem szaknyelvi – szövegek tokenizálásáról szóló szakirodalmat. Az ismertetés főként angol nyelvű szövegekre vonatkozik – tekintettel ezek túlsúlyára –, ahol más nyelvről esik szó, azt külön jelezzük.

4.1 Mondathatár-felismerés

A mondathatárok felismerése általában a központozási jelek alapján történik, mivel azok (pontok, kérdőjelek, felkiáltójelek) legtöbbször mondathatárokat jelölnek. Az angol nyelvben a kérdő- és felkiáltójelek általában egyértelműen jelölik a mondathatárt (a magyarra ez már nem mondható el: szépirodalmi szövegekben előfordulhat mondatbeli közbeékelődések, felkiáltások). A pontok esetében azonban már az angol nyelvben sem ilyen egyszerű a helyzet, hiszen az nem csupán a mondat végét jelölheti, hanem rövidítés része is lehet. Még bonyolultabb a helyzet akkor, amikor a mondat utolsó szava egy rövidítés: ilyenkor a pont egyszerre mondatvégi írásjel és rövidítésjelölő. Ebből következik, hogy a szövegnek szavakra (tokenekre), illetve mondatokra való szegmentálása nem választható szét.

Jurafsky és Martin [8] említi a mondathatár-felismerési módszerek két fő irányvonalát: a szabály alapú, illetve a gépi tanulás alapú módszereket, amelyek segítségével eldönthető, hogy a pont része-e a szónak, avagy mondatvégi írásjel. Ennek eldöntésében segít az, ha rendelkezésünkre áll a gyakrabban használt rövidítésekből álló szótár. Láthatjuk tehát, hogy a dolgozatban megvalósítandó részfeladatok mind összefüggenek egymással. Jurafsky és Martin – bár a gépi tanuláson alapuló módszereket tartja igazán korszerűnek – hasznosnak ítéli a szabály alapú, reguláris kifejezéseket használó algoritmusokat is. Grefenstette nyomán egy egyszerű, Perl nyelven implementált,

reguláris kifejezéseken alapuló mondathatár-felismerő (és egyben tokenizáló) algoritmust is bemutat. Az első szabály leválasztja az egyértelmű központozási jeleket (pl. kérdőjel) és a zárójeleket. A következő szabály a vesszőket különíti el (kivéve, ha azok szám belsejében helyezkednek el, mint tizedesvessző). Majd az aposztrófok és a szóvégi klitikumok problémájára keres megoldást (pl. *I'm* – ebben az összevont alakban az aposztróf után következő *m* a klitikum, ez a karaktersorozat magától értetődően egy tokennek számít). Utolsó lépésként a pontok egyértelműsítése következik: rövidítésszótárat, illetve heurisztikus módszereket használ a rövidítések megtalálásához.

Előfordulhatnak bonyolultabb esetek is, például a függő beszéd. Főleg Észak-Amerikában jellemző, hogy a hagyományos betűszedési gyakorlat szerint a mondatvégi írásjel után idézőjel következik. Ennélfogva a mondatnak nem a pont után van vége, hanem a pontot követő idézőjel után. Manning és Schütze [9] ismertet egy gyakorlatban sokat használt, heurisztikán alapuló mondatokra bontó algoritmust.

1. Kezdetben legyen a vélt mondathatár a pont, kérdőjel, felkiáltójel (esetleg kettőspont, pontosvessző, gondolatjel) után.
2. Helyezd a határt a következő idézőjelek utánra, ha van ilyen.
3. Az alábbi esetekben biztos, hogy nem mondathatárról van szó:
   - Ha olyan ismert rövidítés előzi meg, amely nem szokott mondat végén szerepelni, de rendszerint nagybetűs tulajdonnév következik utána. Pl.: *Prof.* vagy *vs.*
   - Ha ismert rövidítés előzi meg, és nem követi nagybetűs szó. Pl.: etc. vagy Jr. rövidítések, amelyek megjelenhetnek mondat közepén vagy végén.
4. Biztos, hogy nem mondathatárról van szó akkor, ha a kérdőjelet vagy felkiáltójelet kisbetűs szó (vagy ismert név) követi.
5. Minden egyéb feltételezett mondathatárt tekintsünk mondathatárnak.

Az ehhez hasonló algoritmusok megfelelően kidolgozva jól működhetnek, legalábbis azon a területen (abban a szövegtípusban és/vagy témában), amely számára készültek. Hátrányuk, hogy sok kódolást igényelnek, valamint területspecifikusak.

Mikheev [11] az alábbi szabályrendszert javasolja a mondathatárok felismeréséhez:

- Ha a pont egy olyan szó után következik, amely nem rövidítés, akkor mondatvégről van szó.
- Ha a pont rövidítés után következik, és a szövegszakasz (bekezdés, dokumentum stb.) utolsó tokenje, akkor mondatvégi pont és egyben rövidítésnek a része.
- Ha a pont rövidítés után következik, és nem követi nagybetűs szó, akkor nem mondatvégi pont, hanem rövidítésnek a része.
- Ha a pont rövidítés után következik, és nagybetűs szó követi, amely nem tulajdonnév, akkor a pont a mondat végét jelzi, ugyanakkor rövidítésnek is része.

Természetesen mindezen szabályok alkalmazásához szükség van arra, hogy az algoritmus képes legyen felismerni a rövidítéseket és a tulajdonneveket.

Manning és Schütze [9] egyéb, más alapokon nyugvó mondathatár-felismerő módszert is ismertet. Riley (1989) statisztikai osztályozó fákat használt e célra. Ezen fák egyik jellemző tulajdonsága a pont előtti és utáni szavak hossza, továbbá a mondathatár előtt és mögött megjelenő különböző szavak a priori valószínűsége (ehhez címkézett tanító adatok nagy méretű adatbázisára van szükség). Palmer és Hears (1994, 1997) a pontot megelőző és követő szavak szófaji felosztásával elkerülték az ilyen adatok használatát, és neurális hálózattal jósolták meg a mondathatárokat. Ezzel egy robusztus, nagy teljesítményű (98-99%), nagymértékben nyelvfüggetlen mondathatár-felismerő algoritmust sikerült megalkotniuk. Reynar és Ratnaparkhi (1997), valamint Mikheev (1998) kifejlesztette a probléma maximum entrópia megközelítését, utóbbi 99,25%-os pontosságot ért el a mondathatár-felismerésben. Reynar és Ratnaparkhi két rendszert is fejlesztett, egy angolnyelv-specifikusat és egy nyelvfüggetlen rendszert [11]. Utóbbi bármilyen latin betűs írást használó nyelvre adaptálható.

4.2 Tokenizálás

Mint korábban említettük, a mondatokra bontás és a szavakra bontás részfeladata nem választható szét. A bemeneti szöveget először tokenekre kell bontani. Egy token lehet egy szó vagy valami egyéb is, például szám vagy írásjel. Hogy mi számít szónak, az vitatott kérdés a nyelvészetben. Korábban a nyelvészek javasolták a fonológiai vs.

szintaktikai szavak terminusokat (e kettő nem ugyanazt jelenti), de ezek használata a nyelvtechnológiában nem terjedt el. Kučera és Francis (1967) javasolta a grafikus szó elnevezést, amelyet a következőképpen definiáltak: folyamatos – esetleg kötőjelet és aposztrófot igen, de más karaktert nem tartalmazó – alfanumerikus karakterek füzére, oldalán egy-egy szóközzel [9].

Angol nyelvű szöveg tokenek esetén elsőre nem tűnik bonyolultnak az egységekre bontás. Fő vezérfonal a whitespace karakter előfordulása: szóköz vagy tabulátor vagy egy újsor-kezdet szavak közt – de ez a jelzés nem szükségszerűen megbízható. Melyek a fő problémák?

Pontok. A szavakat nem mindig határolja whitespace karakter. A központozási jelek gyakran hozzátapadnak a szavakhoz, például a vessző, pontosvessző, pont. Első lépésként jó megoldásnak tűnik a központozó jeleket egyszerűen eltávolítani a szótokenekből, de a pontok esetén ez problémás. A pontok legtöbbször mondatvégi írásjelek, de lehetnek rövidítés jelei is. Ezek a rövidítésbeli pontok valószínűleg a szó részei kell, hogy maradjanak, sőt bizonyos esetekben különösen fontos, hogy megtartsuk őket. Például akkor, ha meg akarjuk különböztetni a *Wash* 'Washington állam' jelentésű tokent (amely egy rövidítés) a *wash* ige nagybetűs alakjától. Fontos megjegyezni, hogy a mondat végén megjelenő rövidítések esetében, mint pl. az *etc.*, csak egy pont jelenik meg, de ez a pont mindkét funkciót egyidejűleg betölti (mint mondatvégi írásjel és mint a rövidítés jele).

Aposztrófok. Az olyan angol kifejezésekben, mint pl.: *I'll* vagy *isn't*, felmerül a kérdés: egy vagy két szóról van szó? A grafikus szó definíciója szerint egy szó, de sokan két szónak veszik.

Elválasztás. Itt a probléma forrása, hogy ugyanazt a szót különböző alakok reprezentálják.

Kötőjel. Itt is felmerül a kérdés, hogy a kötőjelet tartalmazó karaktersorozatok egy vagy két szónak számítanak-e. A kötőjelek használata sokszor nem következetes (*cooperate*, *co-operate* alakban is előfordul ez az ige).

Egy szóalak, több jelentés. Két lexémának ugyanaz az alakja. Ilyen – nyelvtani homonímiának nevezett – jelenség pl.: *saw* 'fűrész', *saw* mint a *see* ige múltidejű alakja.

Szóközt tartalmazó tokenek. Olyan karakterláncok, amelyek szóközt tartalmaznak, például: New York, San Francisco. Különösen nehéz eset, amikor még kötőjelet is tartalmaz a kifejezés: *the New York-New Haven railroad*. Itt sem célravezető a csupán szóközök mentén történő szegmentálás, hiszen így külön token lenne a New és a York, holott ezek egybe tartoznak. Ilyen esetekben szükséges egy többszavas kifejezéseket tartalmazó szótár, ez pedig továbbvezet az ún. névelem-felismeréshez (named entity detection), amely képes felismerni a neveket, dátumokat, szervezeteket stb.

4.2.1 Tokenizálás nem angol nyelvű szövegekben

Sok nyelv egyáltalán nem tesz szóközt a szavak közé, így az alap szószétválasztó algoritmus egyáltalán nem használható. A legtöbb kelet-ázsiai nyelv ilyen (kínai, japán, thai). Ugyanígy az ógörög sem használt szóközöket. A szóközök (csakúgy mint az ékezetek) csak ezután jöttek létre. A szavakra bontás az ilyen nyelvekben igazi kihívást jelentő feladat.

4.2.2 Magyar nyelvű szövegek tokenizálása

Hasonlóan az angolhoz, a tokenizálás először egyszerű feladatnak tűnhet: az írásjelek elhagyása után a szövegeket a szóközök mentén szavakra bontjuk. A magyar nyelvben különösen a kötőjeles szavak tokenizálása jelenthet problémát, mivel nagy számban fordulnak elő szó belsejében (összetett szavak tagolásában, illetve bizonyos idegen szavak, rövidítések, betűszók toldalékolt alakjaiban). Mint az angolban, a magyarban is kérdéses a szintaktikailag egy egységnek tekinthető szószekvenciák kezelése (Bartók Béla, Kispál és a Borz, 1848. március 15.). Az aposztrófok problematikája kevésbé jelentős, mint az angolban, mivel nyelvünkben csak elvétve fordulnak elő, leginkább informális környezetben (pl. *nem t'om*).

4.3 A rövidítések

Mint korábban említettük, a rövidítések felismerése szorosan kapcsolódik a mondathatárok felismeréséhez. Hiszen ahhoz, hogy egy pontról el tudjuk dönteni, mondatvégi írásjel-e vagy rövidítésnek a része, az adott tokenről meg kell tudni állapítani, hogy rövidítés-e. A rövidítések felderítése heurisztikus módszerrel történik, mindezt kiegészítve egy közismert rövidítéseket tartalmazó listával. Nyilván ennek a módszernek is megvannak a maga hiányosságai. Például: ha egy – a listában nem szereplő – rövidítés után nagybetűs szó következik, a heurisztikus módszer nem fogja az adott rövidítést felismerni. Mikheev (2002) a dokumentumközpontú megközelítést (documentum-centered approach, DCA) javasolja [10]. Minden, legfeljebb négy karakterből álló szót, amelyet pont követ, lehetséges rövidítésként kezel. Első lépésként a rendszer összegyűjti az egyértelmű környezetben előforduló potenciális rövidítések unigramjait. Ha a lehetséges rövidítés a dokumentumban bárhol pont nélkül fordul elő, akkor mégsem rövidítés. Ahhoz, hogy egy lehetséges rövidítésről megállapítsuk, tényleg az-e, olyan szövegkörnyezetet kell keresni, ahol az pontra végződik, és a pont után kisbetűs szó, szám vagy vessző következik.

Mikheev a homonim alakok problémájára is kitér: azokra az esetekre, amikor egy szó és egy rövidítés alakja egybeesik. Például a *Sun.* ('Sunday') és *Sun* mint egy újság neve. Ugyan ott van a pont mint megkülönböztető jegy, de erre csak akkor támaszkodhatunk, ha feltételezzük, hogy azonos dokumentumon belül a rövidítések jelölése következetes (azaz jelen esetben mindig ponttal jelöljük). Mi a helyzet, ha következetlen a jelölés? Ilyenkor unigramok helyett bigramokat érdemes keresni: nem csupán a potenciális rövidítést, hanem az azt megelőző szót is figyelni kell.

Grefenstette és Tapanainen több – egyre hatékonyabb – módszert ismertet a rövidítések felderítésére [12]. Első körben nem használ rövidítéslistát, csupán bizonyos – reguláris kifejezésekkel megfogalmazott – formai szabályok figyelembe vételével próbálja a rövidítéseket felkutatni. Háromféle rövidítésfajtát különböztet meg:

(1) egyetlenegy nagybetű (pl. *A.*),
(2) betű, pont, betű, pont (pl. *U.S.*),
(3) nagy kezdőbetű, mássalhangzó-sorozat, pont (pl. *Mr.*, *St.*).

A módszer hiányossága, hogy a lehetséges rövidítéseket izoláltan, a környezetükből kiragadva keresi. Akkor lenne eredményes, ha a korpusz például egy tetszőleges szavakat tartalmazó lista lenne. Ha a rövidítések természetes környezetükben, a szövegben helyezkednek el, érdemes a környezetet is figyelembe venni, például azt, hogy milyen karakterek jöhetnek még utána a szövegben. Grefenstette és Tapanainen az alábbi módon fogalmazzák meg a feltételt: egy ponttal végződő karakterlánc potenciális rövidítés, ha vessző és kis kezdőbetűs szó követi, vagy szám, vagy bármilyen nagybetűvel kezdődő, ponttal végződő karakterlánc (azaz mondat).

```
[A-Za-z][^ ]*\.([,?]| [a-z0-9])
```

Ennél hatékonyabb módszer, ha a rövidítéseket felvesszük egy listába. Grefenstette és Tapanainen szerint a ponttal végződő karaktersorozatoknál az alábbi módon kell eljárni:

- ha kisbetű, vessző vagy pontosvessző jön utána, akkor rövidítés;
- ha ismert rövidítés, akkor aszerint járunk el vele;
- egyébként a pont mondatvégi írásjel.

A lista megoldást jelenthet azon esetekre is, amikor a rövidítések után – akár a kivételes helyesírás miatt, akár tévedésből – nem tesznek pontot. Az ilyen esetek reguláris kifejezéssel nem írhatók le, mivel formailag egybeesnek a hagyományos szavakkal.

5 Néhány magyar nyelvű tokenizáló szoftver

A Szegedi Tudományegyetem Nyelvtechnológiai Csoportja által fejlesztett Magyarlánc [1] programcsomag tartalmaz mondatokra szegmentáló, illetve tokenizáló modult. Ezek a Morphadorner nevű, Java-alapú általános nyelvi elemzőnek a magyar nyelvre adaptált változatai. Előnye, hogy platformfüggetlen, könnyen integrálható más rendszerekbe, mindemellett szabadon hozzáférhető, ingyenesen letölthető és továbbfejleszthető.

A mondatszegmentáló modul szótára tartalmazza azokat a rövidítéseket, amelyek után pont áll, ez a pont azonban nem mondatvég. Például: *zrt.*, *szül.*, hónapnevek rövidítései [13].

A BME Média Oktató és Kutató Központja által fejlesztett, nyílt forráskódú, szabadon letölthető Huntoken[2] egy olyan gyors shell szűrő, amely képes mondatokra és szavakra bontani magyar nyelvű szövegeket [14]. A program szabályok alapján végzi a szegmentálást. Felismeri a szavak egy részét és azok toldalékolását, és a megfelelő – a Szeged Korpuszban is használt – MSD-kódokkal jelöli. Felismeri és megfelelően kezeli a leggyakrabban használt rövidítéseket. A rövidítésszótár alapján a program felülbírálja a korábban megjelölt mondat- és szóhatárokat. Ha a program korábban már leválasztott egy pontot (mint különálló tokent), a rövidítésfelismerő modul (szűrő) lefuttatása után az adott pont visszakerül eredeti helyére, amennyiben a vizsgált token szerepel a rövidítések listájában.

Maga a rövidítéslista egy egyszerű szövegfájl a gyakrabban előforduló rövidítésekkel. Ez kb. 130 db rövidítést jelent. A lista – elméletileg – tetszés szerint bővíthető.

A program 98%-os pontosságú mondat- és tokenhatár-felismerést ígér. A méréseket a Szeged Korpusz 1.0-n végezték a szoftver készítői. Az említett korpusz különböző témakörökből lett összeválogatva: szépirodalmi szövegek, diákok iskolai fogalmazásai, újságcikkek, számítástechnikai, jogi szövegek. Kísérletképpen – hogy milyen

---

[1] http://www.inf.u-szeged.hu/rgai/magyarlanc

[2] http://mokk.bme.hu/resources/huntoken

eredményt produkál a program akkor, ha a rendelkezésünkre álló orvosi szöveget kapja bemenetül – lefuttattuk a Huntokent egy rövidebb szövegmintára (~20-25 mondat). A kimenetül kapott fájlt két szempont szerint vizsgáltuk meg: (1) helyükön vannak-e mondathatárok, (2) a program megfelelően ismerte-e fel a rövidítéseket. A Huntoken (ellentétben a saját készülő tokenizálónkkal) nem jelzi azt külön, ha az adott token egy rövidítés. A felismerés ténye itt azt jelenti, hogy pontra végződő rövidítés esetén a program nem választja le a pontot mint különálló tokent, hanem helyén hagyja, jelezvén ezzel a szó és a pont összetartozását.

A mondathatár-felismerést vizsgálva a választott mintán azt tapasztaltuk, hogy a Huntoken 84%-os fedést és 87,5%-os pontosságot ért el. Bár ezek sem kifejezetten rossz eredmények, messze elmaradnak az ígért értékektől. A huszon-egynéhány mondathatárból négyet nem ismert fel a program, míg három alkalommal olyan helyet jelölt meg mondathatárnak, amely valójában nem az. Hamis pozitív találatok az alábbi mondatban figyelhetők meg:

Th: jobb szem: órás Ultracortenol még 2 napig, után kétóránként, most 1 cs. Mydrum+Neosynephrine-t kapott, éjsz. Ultracornteol kenőcs.

Valójában egy mondatról van szó (ha a kettőspontokat nem tekintjük mondathatárnak), a program azonban tévesen három helyen is mondathatárként érzékelt.

(1) *Th: jobb szem: órás Ultracortenol még 2 napig, után kétóránként, most 1 cs.*
(2) *Mydrum+Neosynephrine-t kapott, éjsz.*
(3) *Ultracornteol kenőcs.*

A téves találatok oka, hogy az adott rövidítés (*cs.*, *éjsz.*) utáni pontot mondatzáró írásjelként érzékeli az algoritmus, mivel utánuk nagybetűs szó következik, amely akár mondatkezdő szó is lehetne. A nagy kezdőbetű oka itt azonban nem ez, hanem a szavak tulajdonnévi mivolta (gyógyszernevek: *Mydrum*, *Ultracortenol*).

A rövidítésfelismerést tekintve a Huntoken 100%-os pontosságot, de csupán 65,3%-os fedést ért el. Más szóval: amit rövidítésként ismert fel, azok közül mind helyes találat volt, viszont az összes szövegbeli rövidítésnek csak 65,3%-át találta meg. Azt, hogy

bizonyos rövidítéseket nem ismert fel a program, onnan tudhatjuk, hogy a rövidítésbeli pontot külön sorba helyezte, és mint punktuációt jelölte meg. Példák a nem felismert rövidítésekre: *o. d.* 'oculus dexter', *o. s.* 'oculus sinister', *vvf.* 'vörös visszfény', *cs.* 'csepp'. Példák a felismert rövidítésekre: *kp.* 'központ', *St.* 'státusz', *subconj.* 'subconjunctivalis, kötőhártya alatti'. E példák közül az első kettő szerepel a rövidítéslistában (megjegyzendő, hogy a *St.* esetén nem a 'státusz', hanem nagy valószínűséggel a 'szent' jelentésre gondolhattak a készítők, bár ez a végeredmény szempontjából irreleváns). A *subconj.* rövidítés mivoltát azért ismerte fel a program, mert utána kisbetűs szó következik, és a szabály úgy szól, hogy csak akkor lehet rövidítés a pontra végződő szó, ha nagybetűs szó követi.

Az eredményből láthatjuk, hogy a Huntoken működése csak akkor közelíti meg a hibátlant, ha általános, köznyelvi szövegen teszteljük. Szaknyelvi, ezáltal speciális szókincsű szövegeken láthatóan rosszabb eredményeket kapunk. Joggal merül fel a kérdés, vajon nem segítene-e a helyzeten a rövidítésszótár kiegészítése a használatos orvosi rövidítésekkel. Ez elméletben működik, de a gyakorlatban a program nem boldogul a gyárinál sokkal több rövidítés kivételként való felsorolásával [15]. (A program hátrányaként sorolhatjuk fel emellett a platformfüggőséget – csak UNIX-környezetben használható –, továbbá azt, hogy a szűrőket előállító Flex program nem támogatja az UTF-8-as karakterkódolást.)

Említést érdemel még a BME Távközlési és Médiainformatikai Tanszék beszédtechnológiai laboratóriuma által fejlesztett automatikus szövegfelolvasó, a ProfiVox, melynek szövegfeldolgozó modulja nyelvészeti alapossággal megtervezett algoritmusokat tartalmaz [16]. A rendszer képes a mondathosszak kezelésére, valamint a legáltalánosabb rövidítéseket is tartalmazza a kiejtésikivétel-szótára. A szótár a beszédszintetizátor graféma-graféma átalakító egységének (GTG) a része, és a felhasználó által meghatározott rövidítések (és egyéb) kivételek feloldásához használatos szabályokat. A szótár olyan általános rövidítéseket tartalmaz, mint pl.: *stb.*, *ún.*, *kb.*, *dr.*, *gysz.*, *egyh.* [16]. Ami a szakmai rövidítéseket illeti, ilyen magyar nyelvű elektronikus szótárak csak elvétve léteznek, egységesítésük és rendszerezett gyűjtésük egyelőre várat magára.

A ProfiVox említett modulja, a graféma-graféma átalakító végzi a szöveg előkészítését a szövegfelolvasó számára [17]. A szöveg előkészítésének egyik fontos része a mondatok elkülönítése. Az átalakítás során történik a bemenő szöveg és a GTG-átalakító egység szótárának egybevetése. Ebben a szótárban mintegy 200 rövidítés és azok feloldása van tárolva. A rövidítések felismerése segíti a mondathatárok meghatározásában. Az algoritmus a rövidítések pontját eltávolítva és a rövidítést szöveggé alakítva könnyíti meg a mondatvégek felismerését.

A program nem nyílt forráskódú, ezáltal rövidítésszótára sem hozzáférhető.

Összefoglalva: a magyar nyelvű szöveg tokenizálására alkalmas szoftverek köznyelvi szövegek tokenizálására készültek, és ilyen környezetben jó eredményeket produkálnak. Orvosi szaknyelvi szövegek feldolgozására csak korlátozottan használhatók, mivel a szaknyelvi speciális rövidítéseket egyáltalán nem ismerik. A rövidítéslista kibővítése (már ha egyáltalán lehetséges) is csak részleges javulást hozhat. Ennek oka, hogy a klinikai szövegekben nem csupán a rövidítések különböznek, hanem sok esetben a mondathatárokra vonatkozó szabályok is. Például a Huntokennél említett „pontra végződő rövidítés, majd utána nagy kezdőbetűs szó esetén mondathatár" mint szabály a mi feldolgozandó orvosi szövegeinkre nem alkalmazható. Ezen kívül számos más, a szövegre specifikusan jellemző jelenségre (pl. írásjelre nem végződő „mondatok") nem nyújtanak szabályt a köznyelvi tokenizáló szoftverek.

6 Magyar nyelvű orvosi szövegek feldolgozására alkalmas mondathatár-meghatározó és tokenizáló megvalósítása

Ebben a fejezetben áttekintem, hogy – a korábban ismertetett angol és magyar nyelvű és köznyelvi szövegek feldolgozására készített tokenizálókhoz képest – milyen, a magyar nyelvre, ezen belül a magyar orvosi szaknyelvre jellemző speciális tulajdonságokat kell figyelembe venni a feladat megoldása során. A feladat megvalósításának és a megvalósítás korlátainak ismertetése során a fejlesztés alatt felmerülő tervezési döntéseket és ezek indoklását is részletezem.

6.1 A programnyelv megválasztása

A tokenizáló szoftver megvalósítása Python nyelven történik. Maga a Python egy portábilis, interpretált, egyszerű szintaxisú scriptnyelv, amely kiválóan alkalmas szövegek feldolgozására. Tömör, gyorsan olvasható programok készíthetők vele. A nyelv megválasztása mellett szólt az is, hogy a Pythonban írt programok lényegesen rövidebb fejlesztési időt igényelnek, mint a C++-ban vagy Javában írt alkalmazások. Nem utolsósorban az is a Python mellett szólt, hogy a dolgozathoz kapcsolódó kutatás során a különböző komponensek fejlesztése ezen nyelven történik (pl. az előfeldolgozó modul is Python scriptekből áll).

6.2 Programozási szemlélet

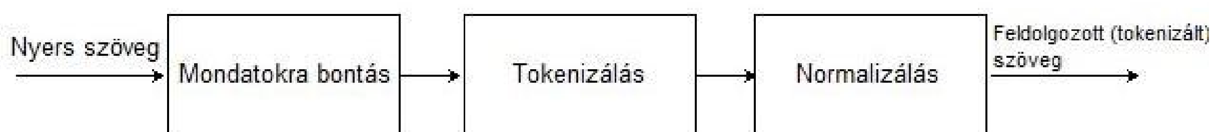
A fejlesztés során végig strukturált és moduláris programozási szemlélet szerint dolgoztunk. Döntésünk oka, hogy a hangsúly a szabályok (reguláris kifejezések) megfogalmazásán és a bemeneti szöveg lépésenkénti átalakításán van. Azaz tulajdonképpen nem történik más, mint hogy a program nagyszámú stringmanipulációt hajt végre, amíg el nem éri a kívánt végeredményt. Mindez kiválóan megoldható strukturált szemlélettel. (A Python támogatja az objektumorientált paradigmát is, de az említett okok miatt felesleges túlbonyolításnak éreztük ezen szemléletmód alkalmazását.) A moduláris szemlélet mutatkozik meg abban, hogy a könnyebb átláthatóság kedvéért a szabályokat tartalmazó listát külön forrásfájlba helyeztük.

6.3 A feladat megtervezése

Első lépésként célszerűnek tűnt a megvalósítandó feladatot különféle részfeladatokra bontani. A különböző részfeladatok egymástól nem függetlenek: egymásra épülnek, így sorrendjük sem mindegy. Kezdetben a legnagyobb egységekre – jelen esetben mondatokra – kell felosztani a szöveget, s miután ez megtörtént, lehet tovább-bontani kisebb egységekre. Ez jelen esetben szavakra, pontosabban tokenekre darabolást jelent. Ennek megtörténte után lehet csak azzal foglalkozni, hogy az egyes tokeneket kategorizáljuk: a főbb kategóriák – a teljesség igénye nélkül – a dátum, központozási jel, rövidítés vagy szó. Fontos hangsúlyozni, hogy a szó mint kategória magától értetődően túl tág fogalom: célszerű lenne ezen belül további alkategóriákat felvenni, de ennek megoldása nem képezi a dolgozat részét.

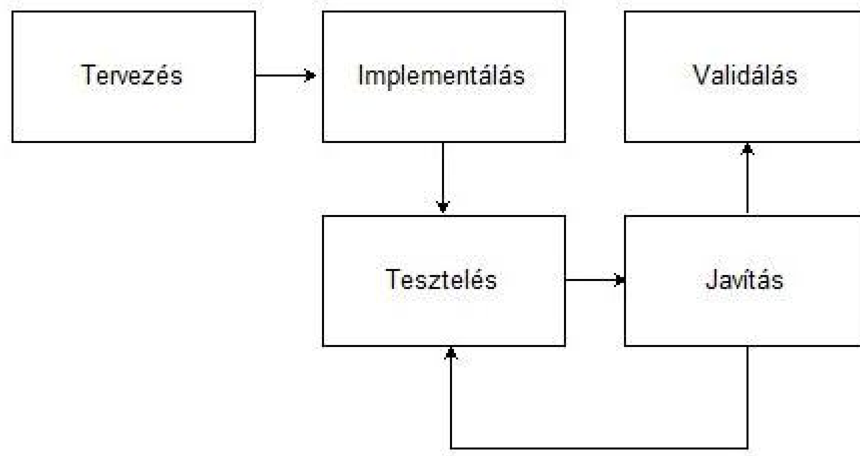
A rövidítések felismerése tehát a harmadik megvalósítandó részfeladat. Azonban nem elég csupán felismerni azokat. A rendelkezésemre álló orvosi szövegekben (és sokszor úgy általában az orvosi szövegekben) a kifejezések rövidítése nem következetes: hol ponttal, hol pont nélkül használják ugyanazt a rövidítést, de akár egyéb eltérések is lehetnek. Fontos tudni, hogy bizonyos eltérő formájú rövidítések valójában ugyanazt jelölik. A rövidítések egységes kezelésének megvalósítása a harmadik részfeladat másik célja. E folyamatot nevezzük normalizálásnak.

Az egymásra épülő megvalósítandó részfeladatokat mutatja be tömören az 6.1. ábra.



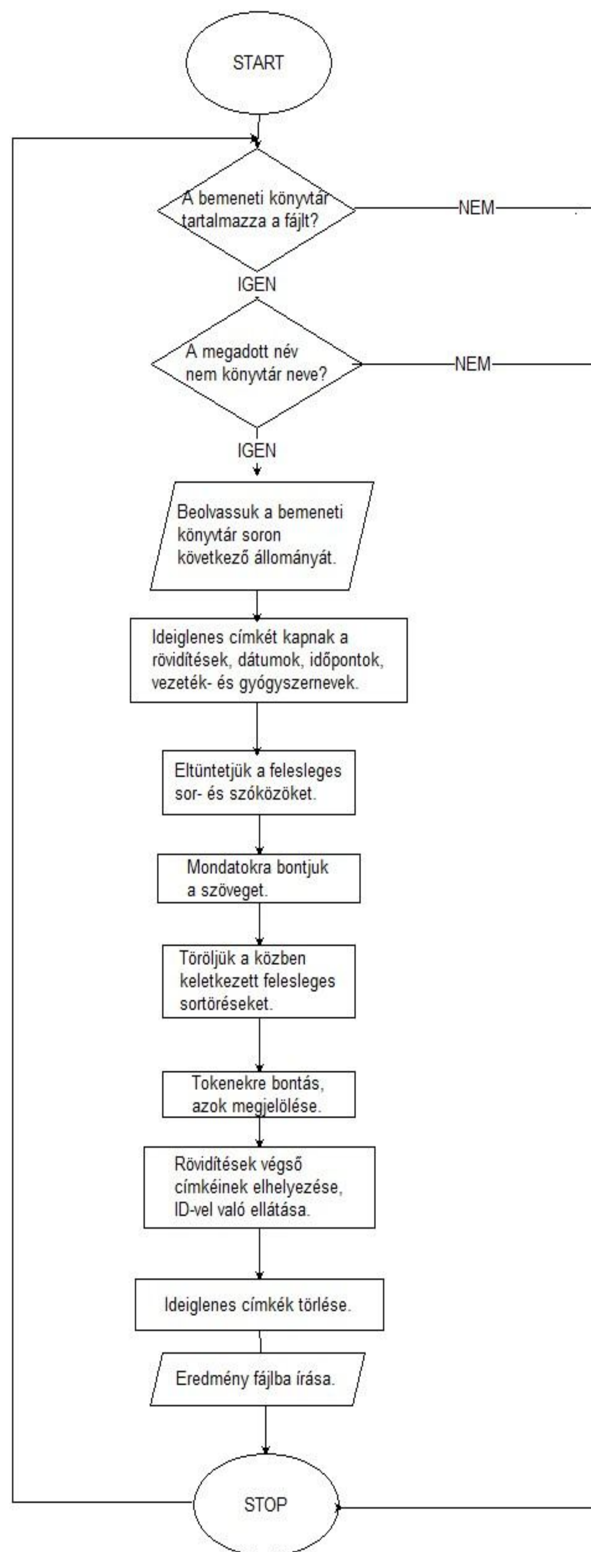
6.1. ábra. A tervezés részfeladatai

Minden egyes részfeladat külön modul. E modulok létrehozása a szoftvertechnológiában megszokott munkafolyamatok egymásutánjából áll (6.2. ábra). Egy-egy modul (mondatokra bontás, tokenizálás, normalizálás) tervezése, implementálása és validálása külön-külön történik. Az egyszerűbb követhetőség kedvéért dolgozatomban is ezt a felosztást követem. Az elkészített teljes program elemzése, kiértékelése a következő fejezet tárgya.



6.2. ábra. Az egyes részfeladatok munkafolyamatai

A készülő program tervezett működését a 6.3. ábra szemlélteti.



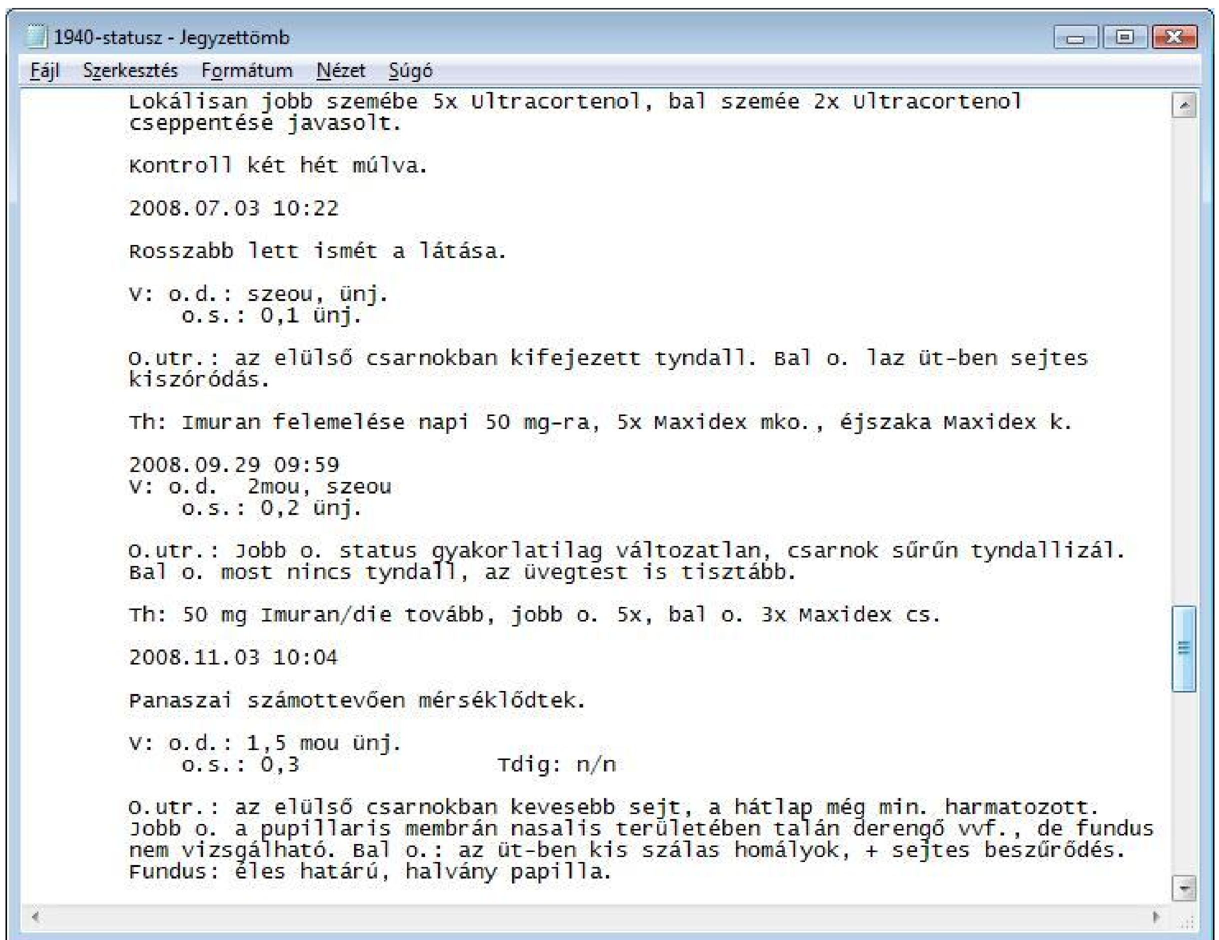
6.3. ábra. A program folyamatábrája

6.4 Az egyes modulok kifejlesztése

6.4.1 Mondatokra bontás

6.4.1.1 Tervezés

Első lépésként tanulmányoztuk a feldolgozni kívánt orvosi (szemészeti témájú) szövegeket. Noha bizonyos előfeldolgozáson már túlestek (a zaj csökkentése megtörtént), a dokumentumok nem mondhatók strukturáltnak. A 6.4. ábrán egy mintát láthatunk a feldolgozásra váró szövegekből.


1940-statusz - Jegyzettömb

Fájl Szerkesztés Formátum Nézet Súgó

```
Lokálisan jobb szemébe 5x Ultracortenol, bal szemée 2x Ultracortenol
cseppentése javasolt.

Kontroll két hét múlva.

2008.07.03 10:22

Rosszabb lett ismét a látása.

V: o.d.: szeou, ünj.
    o.s.: 0,1 ünj.

o.utr.: az elülső csarnokban kifejezett tyndall. Bal o. laz üt-ben sejtes
kiszóródás.

Th: Imuran felemelése napi 50 mg-ra, 5x Maxidex mko., éjszaka Maxidex k.

2008.09.29 09:59
V: o.d.  2mou, szeou
    o.s.: 0,2 ünj.

o.utr.: Jobb o. status gyakorlatilag változatlan, csarnok sűrűn tyndallizál.
Bal o. most nincs tyndall, az üvegtest is tisztább.

Th: 50 mg Imuran/die tovább, jobb o. 5x, bal o. 3x Maxidex cs.

2008.11.03 10:04

Panaszai számottevően mérséklődtek.

V: o.d.: 1,5 mou ünj.
    o.s.: 0,3             Tdig: n/n

o.utr.: az elülső csarnokban kevesebb sejt, a hátlap még min. harmatozott.
Jobb o. a pupillaris membrán nasalis területében talán derengő vvf., de fundus
nem vizsgálható. Bal o.: az üt-ben kis szálas homályok, + sejtes beszűrődés.
Fundus: éles határú, halvány papilla.
```

6.4. ábra. Nyers szöveg (bemenet)

A feladat tehát a strukturálatlan szövegfájlok strukturálttá tétele. Jelen részfeladat célja olyan eljárások kidolgozása, amelyek képesek a rendelkezésre álló orvosi szövegekben a mondathatárok felismerésére és azok megjelölésére. A megjelölés úgy történik, hogy minden mondatot új sorba helyezünk, egy-egy sor kihagyással. Így jóval „szellősebb", átláthatóbb lesz a dokumentum szerkezete, és – nem utolsósorban – jó kiindulási alap lesz a későbbi tokenekre bontáshoz.

A feldolgozás kezdetben hasonlóképpen történik, mint egy köznyelvi szöveg esetében. Szabály alapú tokenizálóról lévén szó, első lépésként – természetes nyelven, azaz szövegesen – megfogalmaztunk néhány egyszerűbb szabályt arra, hogy hol lehetnek mondathatárok. E szabályok a következők:

Mondathatárról van szó, ha:

- egy vagy több felkiáltójel után legalább egy szóköz, majd nagybetűvel kezdődő szó következik, esetleg szám;
- egy vagy több kérdőjel után legalább egy szóköz, majd nagybetűs szó vagy szám következik;
- egy kettőspont után legalább egy szóköz, majd nagybetűs szó vagy szám következik;
- egy – nem rövidítést jelző – pont után nagybetűs szó következik, amely nem tulajdonnév.

A tervezés során több esetben olyan döntési helyzetbe kerültem, amikor több megoldás is jónak látszott. Tipikusan ilyen volt a kettőspont alkalmazása. Vegyük az alábbi – a feldolgozandó szövegekből származó – példákat:

(1) *O.d.: Ép szemhéjak, a subconj. suffusio gakorlatilag felszívódott […].*

(2) *Fundus: éles határú papilla kivehető, színe jó, teltebb vénák és szűkebb artériák a hátsó póluson.*

(3) *Th: jobb o. 5x Maxidex cs, 2x Ultracortenol kenőcs, bal o. 2x Maxidex cs,*
éjszakára Ultracortenol kenőcs.

Kérdés, hogy mikor tekinthető a kettőspont mondathatárnak. Az (1) példában egyértelmű, hogy mondathatárról van szó az O.d. rövidítést követő kettőspont esetében, hiszen utána nagybetűs szó következik, amely nem tulajdonnév. A (2) eset már nem ennyire egyértelmű. A kettőspont utáni kisbetű – bár valószínűleg nincs szó tudatos nyelvi megformáltságról – arra enged következtetni, hogy az azt követő egységet nem szánta új mondatnak a szöveg megalkotója. A kettőspont előtti és utáni egységek között

szoros logikai kapcsolat van, amelyet a helyesírás is visszatükröz. Mindamellett ugyanez az írásmód az (1) esetben is alkalmazható lett volna, kifejezendő, hogy a két eset közti különbség nüansznyi. Ugyanígy a (2) példát is elemezhetnénk úgy, hogy a kettőspont után – a kis kezdőbetű ellenére – új mondat következik. Az elenyésző különbségek ellenére mégis azt a döntést hoztuk, hogy a továbbiakban csak a kettőspont után következő nagybetű jelent majd mondathatárt, a kisbetűvel kezdődő egység inkább tagmondat vagy felsorolás: utóbbira példa a (3) eset.

A tervezés során az írásjelek közül nagy hangsúlyt fektettünk a pontnak a kezelésére. Kiindulásként a 3.1. fejezetben ismertetett algoritmus [12] egyszerűsített változatát alkalmaztuk. E szerint abban az esetben van szó mondatvégi írásjelről, ha:

- a pont egy olyan szó után következik, amely nem rövidítés;
- a pont rövidítés után következik, és nagybetűs szó követi.

Természetesen ez az algoritmus nem fed le minden esetet – amint azt majd később látni fogjuk –, de egy alap rendszer működéséhez elégséges, és könnyen bővíthető további szabályokkal, ami által sokkal hatékonyabb működés érhető el.

A szabályok megfogalmazásából láthatjuk, hogy már a kiindulási alapul szolgáló mondathatár-felismerő programnak is képesnek kell lennie a rövidítések felismerésére. Noha a rövidítések kezelésének megvalósításáért – a korábbiakban megfogalmazottak szerint – a harmadik modul felelős, mégis szükségessé válik felkészíteni a programot arra, hogy megbízhatóan derítse fel a rövidítéseket. Ez néhány újabb szabály megfogalmazásával lehetővé tehető, ám megbízhatóbb – és nem utolsósorban egyszerűbb – módszer erre a célra rövidítéslistát használnunk. A program végignézi, szerepel-e a listában a pontot tartalmazó token, és ha igen, megjelöli mint rövidítést. Így, ha már tudjuk egy adott tokenről, hogy rövidítés-e, jóval könnyebben eldönthető, hogy az adott pont milyen szerepet tölt be a szövegben: mondatvégi írásjel, rövidítés jele, avagy mindkettő.

A szoftver megvalósításának kezdetén – előző féléves önálló laboratóriumi munkám eredményeként – már rendelkezésemre állt egy, a feldolgozandó dokumentumhalmazból kinyert rövidítéslista, így nem volt szükséges újabb

rövidítésfelismerő szabályok megfogalmazására. (A rövidítéslista előállításával részletesebben a normalizálással foglalkozó fejezet foglalkozik.)

Visszatérve a természetes nyelven megfogalmazott szabályokra, a következő tervezési lépés az, hogy ezeket formalizáljuk. Ennek legegyszerűbb módja az, hogy reguláris kifejezéssel írjuk le őket. Ezzel a tömör formulával könnyedén felismerhetők a szabályokkal leírható, egyébként nyílt tokenosztályok – jelen esetben a mondatok és azoknak határai. A reguláris kifejezéseket természetesen célszerű tesztelni még ilyenkor, az implementálás megkezdése előtt, valóban a kívánt esetet fedik-e le. Teszteléshez – egyszerű használata, könnyű áttekinthetősége miatt – a RegexPal[3] nevű JavaScript alapú alkalmazást használtuk.

A tervezés utolsó lépéseként vegyük sorra, mi az, ami rendelkezésünkre áll: a több mint ezer darab szöveges állományon kívül egy rövidítéslista és néhány, reguláris kifejezéssel leírt szabály. Ezek a néhány eszköz elégséges egy alapszintű mondathatár-felismerő rendszer megalkotásához.

A megvalósítandó program működését az alábbi módon terveztük meg: paraméterként kapja a bemeneti fájlokat tartalmazó könyvtár nevét, illetve azt a könyvtárat, ahová a feldolgozott fájlokat szeretnénk helyezni. (Mivel a program nagy mennyiségű állománnyal dolgozik, célszerűbb nem egy-egy fájl nevét, hanem az azokat tartalmazó mappát megadni.) A rövidítéslistát is paraméterként kapja a program, hogy esetleg más listákat is lehessen használni. Miután a mondatokra bontás – minden mondat új sorba helyezése – megtörtént, a program kiírja a végeredményt a megadott könyvtárba, az eredeti fájlneveket használva.

6.4.1.2 Implementáció

A program létrehozásához az alábbiakra van szükségünk:

- a mondathatárokat leíró szabályrendszerre, amelyet listában tárolunk (az átláthatóság kedvéért célszerű mindezt külön Python modulban elhelyezni);
- valamint a különféle műveleteket megvalósító függvényekre.

[3] http://www.regexpal.com

A függvényeket az alábbi csoportokra oszthatjuk:

- fájlból beolvasó, illetve fájlba író (I/O) függvények;
- a címkézést, illetve a címkék eltávolítását végző függvények (ez utóbbi ahhoz kell, hogy a feldolgozott fájlban ne látszódjanak a segédeszközként használt rövidítéseket jelölő címkék);
- a mondatokra bontás műveletét – a megadott szabályok alapján – végrehajtó függvények;
- normalizáló függvények, amelyek a feldolgozott fájlok külalakjáért felelősek (pl. eltávolítják a nem kívánatos sortöréseket, szóközöket).

A továbbiakban részletesen ismertetjük a fontosabb függvények működését. Azon funkciók részletezésétől eltekintünk, amelyek nevéből egyértelműen kiderül, hogy milyen művelet végrehajtásáért felelősek (pl. write). Az esetleges alternatív megoldásokra is kitérünk, és – döntési helyzet esetén – indokoljuk, miért az adott megoldás mellett döntöttünk.

I/O függvények. A readFile() és a readRegex() függvények – mint azt nevük is mutatja – voltaképpen ugyanazt a műveletet hajtják végre: beolvassák a paraméterként kapott állományt. Mi indokolja ezt a látszólagos redundanciát? A válasz a címkézést végző függvény működésével függ össze, amelynek működését részletesen az erről szóló pontban ismertetjük. A rövidítéseket megcímkéző függvény megvalósításakor kétféle megoldás kínálkozott a rövidítéslista és a feldolgozandó szöveg összehasonlítására, ettől függően a rövidítéseket vagy sztringként, vagy lista adatszerkezetben kellett eltárolnunk. Az eltérő adattípusok konverziója miatti nehézségek, valamint a címkézést végző függvény működése (lásd később) miatt tisztább, egyértelműbb megoldásnak éreztük azt, ha kezdettől fogva különválasztjuk a különböző célt szolgáló szöveges fájlok beolvasását. Ezt az utat követve valósítottuk meg a readFile() és a readRegex() funkciókat. A readFile() a feldolgozandó állományokat egy-egy nagyméretű sztringként tárolja el, míg a paraméterben megadott rövidítéslistát a readRegex() függvény dolgozza fel, és a beolvasott adatokat lista adatszerkezetben tárolja.

Címkekezelő függvények. A tagAbbr() függvény feladata, hogy megkeresse a feldolgozandó szövegben a rövidítéseket, és XML-szerű címkével megjelölje azokat: pl.

`<rov>dr.</rov>`. A rövidítések listáját – miután beolvastuk az azokat tároló szövegfájlt a merevlemezről – lista adatszerkezetben kezeli a továbbiakban a program, ezt a listát kapja paraméterül a címkéző függvény. Mint az előző pontban említettük, kínálkozott alternatív megoldás a rövidítéslista kezelésére: lista helyett tárolhattuk volna például egy nagyméretű sztringben, amelyet – szintén egy darab és igen nagy méretű – reguláris kifejezéssé alakítva vizsgálható, van-e egyezés a feldolgozandó szöveggel. A reguláris kifejezéssé alakított sztring úgy képzelendő el, hogy a rövidítések egymás után következnek, köztük egy-egy VAGY kapcsolattal.

```
roviditesek = ’abl | ac | adapt | adav | add’
```

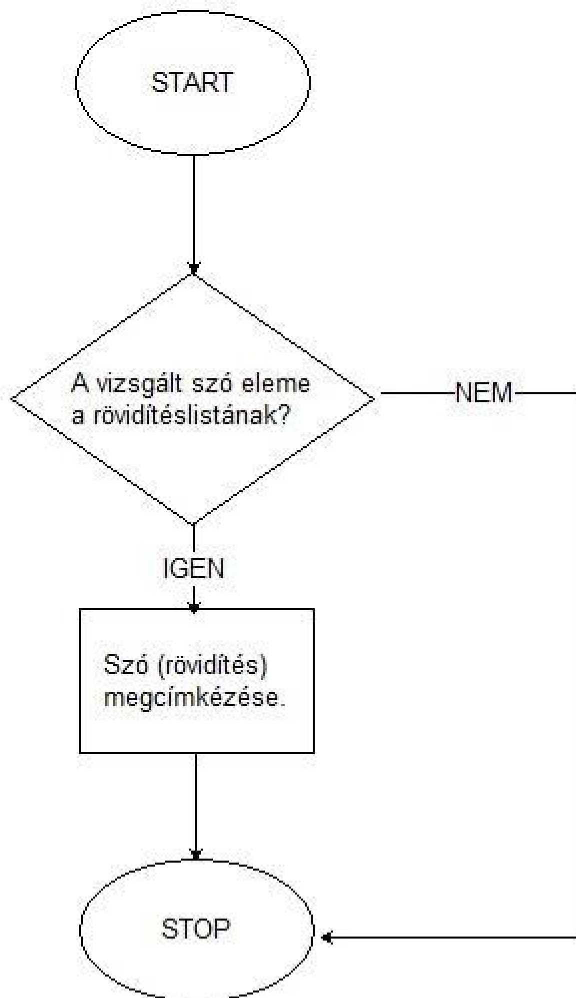
Míg listaként tárolva ugyanezek a rövidítések a következő formát veszik fel:

```
roviditesek = [’abl', 'ac', 'adapt', 'adav', 'add', 'aggr']
```

A listás megoldás esetében nem egy (hosszú) reguláris kifejezésünk van, hanem sok rövid. Akármilyen adatszerkezetben tároljuk is őket, a reguláris kifejezést tovább kell alakítani. Mint láthatjuk, a fenti megoldásokban nem szerepelnek a pontok. Ennek oka, hogy – mint már említettük – a rövidítések jelölése nem következetes a rendelkezésünkre álló dokumentumokban: ugyanazt a rövidítést hol ponttal, hol pont nélkül jelölik. Ugyanazt a rövidítést ponttal és pont nélkül is eltárolni a listában redundanciát eredményezett volna, így a pont nélküli alak mellett döntöttünk, melyhez – miután a megfelelő adatszerkezetben eltároltuk – a program még hozzáilleszt egy opcionális pontot: `[reguláris kifejezés]\.?`, ahol a kérdőjel jelentése: nulla vagy egy pont. Így az adott reguláris kifejezés mind a pont, mind a pont nélküli változatra illeszkedik.

Mindezt megvalósítani jóval egyszerűbbnek tűnt úgy, ha egyenként illesztjük a reguláris kifejezéshez a pontot, mialatt végigmegyünk a lista adatszerkezet összes elemén, mintha az egyetlen, igen hosszú reguális kifejezésbeli VAGY-kapcsolattal összefűzött elemekhez szúrunk be egyenként egy-egy opcionális pontot. Ez az oka annak, hogy a kétféle megoldási lehetőség közül a rövidítések listában való tárolása mellett döntöttünk.

A tagAbbr() függvény működését a 6.5. ábrán követhetjük nyomon.



6.5. ábra. A tagAbbr() függvény folyamatábrája

A másik címkekezelő függvény, az untagAbbr() eltávolítja a címkéket a szövegből, mivel azok – egyelőre – csupán segédeszközként szolgáltak a rövidítések felismeréséhez és helyes kezeléséhez. A rövidítések egységesítése nem ennek a modulnak a feladata, arra a normalizáláskor kerül majd sor. Jelen részfeladatnak célja olyan kimeneti szövegfájlok előállítása, ahol minden mondat – egy sor kihagyása után – új sorban szerepel: annak jelölésére, hogy mi rövidítés és mi nem, egyelőre nincs szükség.

A mondatokra bontást végző függvények. Ebből is kettő van, név szerint: segment() és checkAbbr(). Az előbbi általánosabb, az utóbbi a speciális esetekre – mint a függvény nevéből is kiderül, a rövidítésekre – vonatkozik. A program elsőként a

speciális eseteket vizsgálja. Miután a címkéző függvény elvégezte feladatát, történik a checkAbbr() hívása. Ez a függvény nem tesz más, mint végrehajtja az előző fejezetben (Tervezés) ismertetett két mondatvégipont-felismerő szabály közül a másodikat, azaz: „Tekintsd mondatvégi írásjelnek (is) azt a pontot, amely rövidítés után következik, és nagybetűs szó követi!”. (Az ilyen helyzetben lévő pontok egyszerre töltik be mind a rövidítést jelölő pont funkcióját, mind a mondatvégi írásjel szerepét.) A függvény hívásakor a program már tudja, mik a rövidítések, így könnyedén végre tudja hajtani a szabályt, és – egyszerű sortörést beszúrva az adott helyekre – megkezdi a szöveg mondatokra bontását.

Miután a speciálisabb eseteket megvizsgáltuk, következhet az általánosabb szabály alkalmazása. Ehhez a külön Python modulban elhelyezett, reguláris kifejezéseket tartalmazó listát használjuk fel. A szabályleíró lista minden egyes eleme egy-egy, az adott központozási jelre vonatkozó szabály: a felkiáltójelre, a kérdőjelre, a kettőspontra és a pontra. Voltaképpen azt fogalmazzák meg a szabályok, hogy az adott írásjel milyen környezetben tekinthető mondathatárolónak. A segment() függvény megvizsgálja, hogy a reguláris kifejezéssel megadott szabályok hol illeszkednek a feldolgozandó szövegre. Illeszkedés esetén (azaz mondatvég esetén) újsor-karakterekkel különíti el egymástól a mondatokat.

Normalizáló függvények. Elsőként a normalize() nevű függvényt hívjuk, amely eltávolítja a redundáns szóközöket a szövegből. A nyers szövegek egyik jellegzetessége a szövegnek egymás után következő, 8-10 darab szóközzel történő tagolása. Ez a mód se nem szabályos, se nem esztétikus, ráadásul a szövegfeldolgozást is megnehezíti, ezért az összes többi művelet végrehajtása előtt célszerű megszüntetni a redundanciát. Más – szép külalakért felelős – függvényeket épp ellenkezőleg, mondhatni az utolsó simítások elvégzése céljából hívunk meg: ezek a mondatokra tördelt szövegekből a fölösleges sortöréseket, szóközöket távolítják el.

6.4.1.3 Tesztelés, javítás

A fenti függvények implementálásával egy kész, működőképes rendszert hoztunk létre, amely képes felismerni a mondathatárokat, és azok mentén mondatokra bontani a szöveget. Ahogy a tervezéssel foglalkozó fejezetben jeleztük is, nem volt célunk

tökéletesen működő tokenizálót létrehozni első lépésként, csupán egy alapot szerettünk volna nyújtani, amely a későbbiekben egyszerűen bővíthető, finomítható.

A tesztelés kétféleképpen történt. Először a rendelkezésre álló dokumentumhalmazból véletlenszerűen kiválasztott két-három szövegen futtattuk le a programot, és néztük meg a végeredményt. Majd az ezekben felfedezett tipikus hibákat látva saját tesztfájlokat hoztunk létre, amelyek már az adott problémára koncentráltak.

A továbbiakban ismertetjük a tesztelés során felmerült hibákat, valamint azt, hogyan sikerült azokat orvosolni.

Mint már említettük, szoftverünk fel lett készítve arra az eshetőségre is, ha a mondat utolsó szava egy rövidítés, s a pont kétféle feladatot lát el. Azonban sok helyütt – az ismert közmondással élve – „visszafelé sült el a fegyver". Vegyük az alábbi (a nyers szövegekből származó) példákat:

(1) *Kontroll telefonos egyeztetés után Dr. Süveges Ildikónál.*

(2) *Ennek megítélésére (különös tekintettel arra, hogy a beteg monoculus), előzetes telefonos egyeztetés alapján jelenjen meg dr. Borbényi Zitánál.*

(3) *dr. Holló G szabadságon van.*

A program a következőképpen szegmentálta a fentieket:

(1) *Kontroll telefonos egyeztetés után Dr.*

Süveges Ildikónál.

(2) *Ennek megítélésére (különös tekintettel arra, hogy a beteg monoculus), előzetes telefonos egyeztetés alapján jelenjen meg dr.*

Borbényi Zitánál.

(3) *dr.*

Holló G szabadságon van.

Egyből szembetűnik, hogy olyan helyeken találunk sortörést, ahol azoknak semmi keresnivalójuk, hiszen szó sincs új mondatról.

Másik példa a nyers szövegből:

(4) *Gyógyszerei: Frontin, Anafranil, Aflamin […].*

(5) *O.d.: subconj. suffusio a bulb. conj.-án alul.*

(6) *most 1 cs. Mydrum+Neosynephrine-t kapott, éjsz. Ultracornteol kenőcs.*

A szegmentálás után:

(7) *Gyógyszerei:*

Frontin, Anafranil, Aflamin […].

(8) *O.d.: subconj.*

suffusio a bulb. conj.-án alul.

(9) *most 1 cs.*

Mydrum+Neosynephrine-t kapott, éjsz.

Ultracornteol kenőcs.

Ezekben az esetekben is rossz helyre került a mondathatár. Vizsgáljuk meg közelebbről a hibákat!

Az első három példában a *dr.* rövidítés után nagybetűs szó következik, ami egy tulajdonnév. Formálisan ez a néhány eset tökéletesen megfelel a szabályban megfogalmazottnak: „Tekintsd mondatvégi írásjelnek (is) azt a pontot, amely rövidítés után következik, és nagybetűs szó követi!”. A program nem tett mást, csupán alkalmazta a szabályt. Nézzük tovább! A (4)-es példával látszólag nincsen probléma, a sortörés nem zavaró. Mégis ellentmondásról van szó, visszautalván a tervezésről szóló fejezetben írottakra. Itt ugyanis hangsúlyoztuk, hogy – saját döntés alapján – a kettőspont utáni nagybetű új mondatot jelöl, míg ha kisbetű követi, akkor nem tekintjük

új mondatnak. Érvnek azt hoztuk fel, hogy kisbetű sok esetben felsorolás után szokta követni a kettőspontot, ilyenkor az emberi természetes nyelvérzék sem érzi a felsorolást új mondatnak. Márpedig a (4)-es példában éppen erről van szó: a gyógyszerek felsorolásáról. Itt sincs szükség tehát sortörésre, mivel a felsorolást nem tekintjük új mondatnak. Az (5)-ös esetben rövidítések halmozására látunk példát, erre vonatkozó szabály eddig nem szerepelt a szabályleíró listában. Végül a (6)-os pontban hasonló problémába futottunk bele, mint az első három példa esetén: a rövidítés után nagybetűs szó következik, amelyet a program megint csak mondathatárként érzékelt.

Legegyszerűbben az egymás után következő rövidítések téves mondathatárként való felismerése javítható. A checkAbbr() függvény reguláris kifejezését kell kibővíteni egy [^<rov>] karakterlánccal. Szavakkal megfogalmazva ez a következőt jelenti: „Tekintsd mondatvégi írásjelnek a rövidítés után következő pontot, ha azt nem követi másik rövidítés."

Ennél komplikáltabb a helyzet azokban az esetekben, amikor a rövidítés után nagybetűs szó következik, amely azonban mégsem mondatkezdést jelent; a nagybetű itt tulajdonneveket jelöl. Az eredeti szabályt („a pont mondathatárt jelöl, ha rövidítés után következik, és nagybetűs szó követi") egészítsük ki az alábbi kitétellel: „nagybetűs szó követi, amely nem tulajdonnév". Így máris kiszűrtük a fenti eseteket. Alkalmazva az eddig megszokott módszereket, most a szabály formalizálása lenne a következő lépés, azaz mindezt reguláris kifejezéssé alakítani. Be kell látnunk azonban, hogy ez a probléma nem oldható meg reguláris kifejezések alkalmazásával, mivel azzal csupán annyi írható le, hogy a rövidítés után nagybetűvel kezdődő szó következzék (ez eddig is működött), de hogy az tulajdonnév-e vagy csupán egy mondatkezdő szó – ezt reguláris kifejezések használatával nem lehet eldönteni. Itt jön be a képbe az ún. névelem-felismerés (named-entity recognition, NER). Ennek során egyrészt fel kell ismerni az előre definiált kategóriákba tartozó tokensorozatokat (pl. tulajdonnév, de – noha a klasszikus grammatikai nézőpont szerint nem nevek – ide soroljuk a dátumokat, telefonszámokat stb. is). Ha a kategóriák megvannak, a tokeneket kategorizálni kell azok alapján.

Mivel jelen munkának nem célja egy teljes névelemkategória-rendszer kialakítása és az összes token megjelölése, ezért csupán azon névelemcsoportok felderítésével

foglalkozunk, amelyek a mondathatárok felismeréséhez feltétlenül szükségesek. (Bár csak lazán kapcsolódik ide, de egyúttal elvégeztük a viszonylag könnyen kinyerhető névelemek – pl. a dátumok, időpontok – megjelölését is. Mondathatár ugyan ritkán követi őket, inkább mondat belsejében találkozhatunk velük; ha mégsem, akkor eleve elkülönítve fordulnak elő, pl. szöveg kezdetekor. Annak oka, hogy felismerésük nem jelent különösebb problémát, az, hogy ezek a névelemek leírhatók reguláris kifejezésekkel.)

A fentebb részletezett hibák korrigálásához nem kerülhető ki a tulajdonnevek felismerése. Létezik ugyan a magyar nyelvre tulajdonnévkorpusz, a SzegedNE [4]korpusz, amely gazdasági és bűnügyi rövidhírek anyagából lett összeállítva, és ingyenesen letölthető, de – pontosan a tematika behatároltsága miatt – orvosi szövegekre kevéssé alkalmazható. (Megjegyzendő, hogy a készülő orvosiszöveg-tokenizáló esetleges továbbfejlesztése esetén viszont kifejezetten hasznos lehet a korpusz egy-egy névelem-listája, például a keresztneveké, ez azonban jelen munkának nem képezi tárgyát.)

A rendelkezésünkre álló dokumentumokat megvizsgálva megállapítottuk, hogy – köszönhetően a szövegek szűk területre korlátozódó tematikájának – a tulajdonnévi kategóriák is elég behatároltak. A szövegek az előfeldolgozás során A szövegek az előfeldolgozás során anonimizálási folyamaton estek keresztül; ez alatt a betegek nevének eltávolítása értendő, az orvosok nevei (mint korábban láthattuk) benne maradtak a szövegben. A másik jellemző tulajdonnévi kategória a gyógyszerneveké. Más tulajdonnévi kategória nem – vagy csak igen elenyésző számban – fordul elő. Ez a megállapítás lényegesen megkönnyíti a dolgunkat.

A feladat tehát a meglévő tulajdonnevek listába gyűjtése volt. Egyszerűbb volt a helyzet az orvosok vezetéknevével, éppen amiatt, hogy van egy fő jellegzetességük: a vezetéknév előtt minden esetben szerepel a Dr./dr. rövidítés. Elég volt megvizsgálni tehát a rövidítés környezetét, és hatékony szövegfeldolgozó eszközök segítségével (egrep) könnyen kinyertük az orvosok vezetéknevét a dokumentumhalmaz összefűzésével létrehozott nagyméretű teljes szövegből.

[4] http://www.inf.u-szeged.hu/rgai/nlp?lang=en&page=corpus_ne

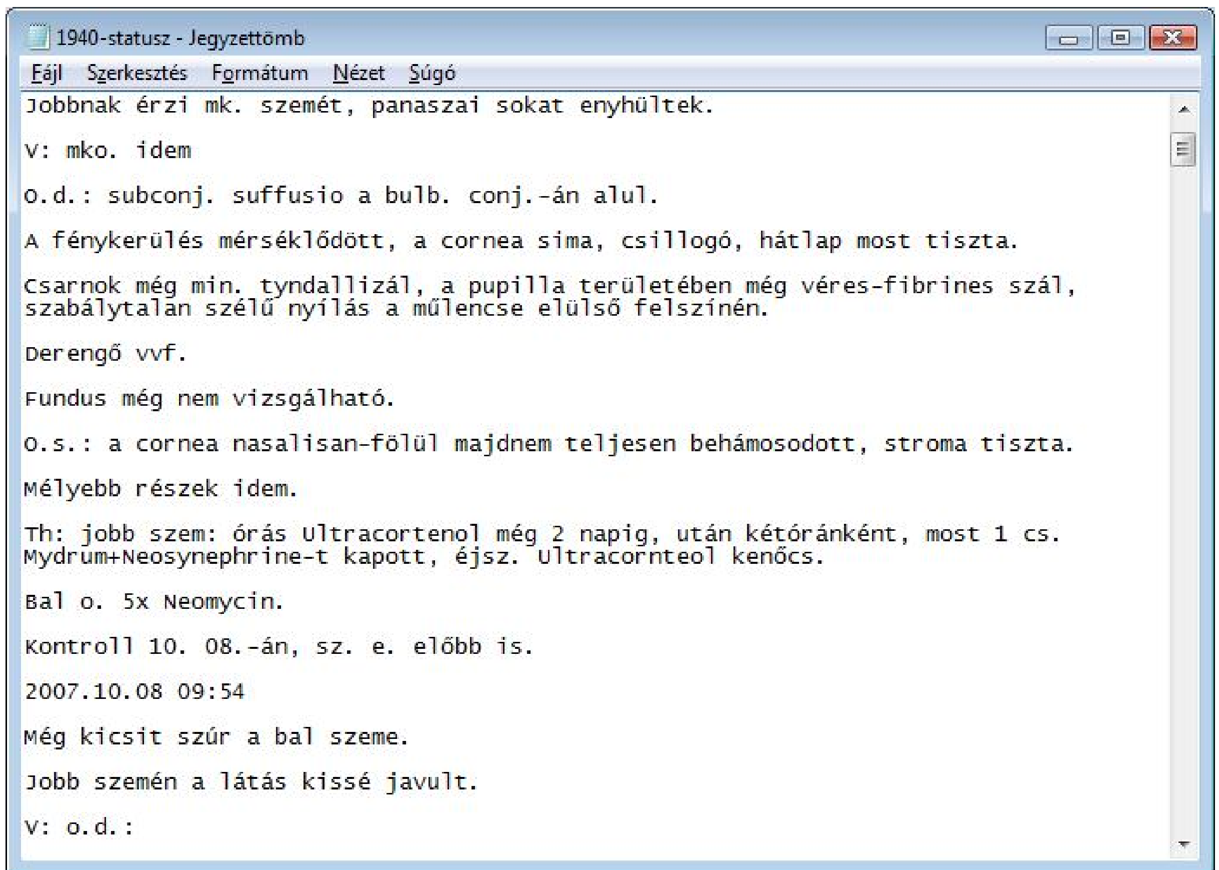
```
cat teljes_szoveg.txt | egrep -o "[D|d]r\.?\s?[A-ZÁÉÍÓÖŐÚÜŰ][a-záéíóöőúüű]+" > doktorok.txt
```

A normalizálás (rendezés, ismétlődő sorok eltávolítása, kisbetűssé alakítás) után megkaptuk a végleges listát, amely tartalmazza a több mint ezer dokumentumban előforduló összes lehetséges orvosnevet.

A gyógyszerek esetében nehezebb volt a helyzet, mert azok formájában – a nagy kezdőbetűn kívül – nincs semmiféle olyan jellegzetesség, amely reguláris kifejezéssel megfogalmazható, és megkönnyítené azok felismerését. Nagy segítséget jelentett azonban, hogy konzulensem, dr. Németh Géza rendelkezésemre bocsátott egy, a közforgalomban kapható gyógyszernek nevét tartalmazó listát. Néhány apróbb átalakítás után (a több alakban előforduló gyógyszernevek átalakítása oly módon, hogy leírhatók legyenek egy darab reguláris kifejezéssel) elkészült tehát az a lista is, amely tartalmazza a dokumentumhalmazban várható legtöbb gyógyszer nevét (kivéve a csak kórházakban elérhetőekét).

Hasonlóan a rövidítések kezeléséhez, a tulajdonneveket is meg kellett címkézni. Ehhez csupán újabb címkéző (és címkeeltávolító) függvényeket kellett implementálni, amelyek bejárják a tulajdonnévlistákat, és megnézik, van-e a szövegben illeszkedés. Ha igen, megjelölik azokat a megszokott XML-szerű címkékkel. Amikor az mondatokra bontásért felelős függvény elkezdi végrehajtani a feladatát, a címkék „megvédik” a köztük elhelyezkedő tulajdonnevet attól, hogy a checkAbbr() függvény végrehajtsa rajtuk a feladatát: nem tud sortörést beszúrni a vélt (téves) mondathatár után, mivel a címkék úgymond elrontják az illeszkedést. Konkretizálva itt arról van szó, hogy például a *<rov>éjsz.</rov> <gyogyszer>Ultracornteol</gyogyszer> kenőcs* karakterláncra – éppen a <gyogyszer> címkék miatt – most már nem illeszkedik a <rov.*?>(.*?)</rov>)(\s+|- [^<rov>]?([A-ZÁÉÍÓÖŐÜŰ][a-záéíóöőüű]*) reguláris kifejezés.

A 6.6. ábrán láthatjuk a javítások elvégzése utáni kimenetet.


1940-statusz - Jegyzettömb
Fájl Szerkesztés Formátum Nézet Súgó

Jobbnak érzi mk. szemét, panaszai sokat enyhültek.

V: mko. idem

o.d.: subconj. suffusio a bulb. conj.-án alul.

A fénykerülés mérséklődött, a cornea sima, csillogó, hátlap most tiszta.

Csarnok még min. tyndallizál, a pupilla területében még véres-fibrines szál, szabálytalan szélű nyílás a műlencse elülső felszínén.

Derengő vvf.

Fundus még nem vizsgálható.

o.s.: a cornea nasalisan-fölül majdnem teljesen behámosodott, stroma tiszta.

Mélyebb részek idem.

Th: jobb szem: órás Ultracortenol még 2 napig, után kétóránként, most 1 cs. Mydrum+Neosynephrine-t kapott, éjsz. Ultracornteol kenőcs.

Bal o. 5x Neomycin.

Kontroll 10. 08.-án, sz. e. előbb is.

2007.10.08 09:54

Még kicsit szúr a bal szeme.

Jobb szemén a látás kissé javult.

V: o.d.:

6.6. ábra. Mondatokra bontott szöveg

Bár a javítás után lényegesen javult a mondathatár-felismerés pontossága, a tökéletestől még messze vagyunk. Így például egyelőre gondot jelenthet az olyan mondatok (?) felismerése, amelyek nem végződnek írásjellel. Hogy mégis mondatnak számítanak, kiderül a jelentésből. Noha a nyers szövegekben valamiféle határoló mégiscsak megfigyelhető az ilyen „mondatok” után (például egy-egy tabulátor vagy sortörés) – és ezt a program ki is használja, mert ezeket is megjelöli mondathatárként –, az ilyen nem egyértelmű esetekben a pontosság kevésbé jó.

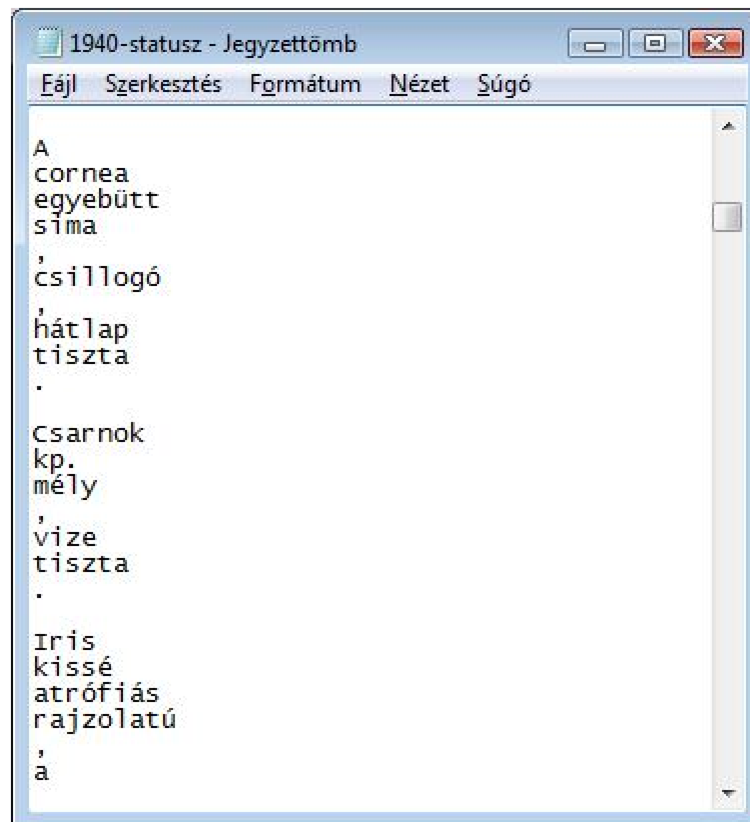
Összességében mégis sikerült egy olyan mondathatár-felismerő algoritmust implementálni, amelyre már bátran ráépíthetjük a következő modult: a tokenizálót, amely a mondatokra bontott szövegeket fogja bemenetként megkapni.

6.4.2 Tokenizálás

6.4.2.1 Tervezés

A tokenizáló modul bemenetként a mondatokra bontott szöveget kapja meg. Feladata, hogy a szöveget további egységekre (tokenekre) bontsa szét. Ez elsősorban szavakra bontást jelent, de nem minden esetben. Előfordulhat, hogy egy token több szóból, esetleg számból és írásjelből is áll. Nézzük például a rövidítéseket, ahol az írásjel (a pont) egyértelműen a rövidítéshez tartozik, senkinek sem jutna eszébe a pontot külön tokenként értelmezni. Ellenben egy mondatvégi írásjel – legyen az pont, kérdőjel vagy felkiáltójel), vagy a tagmondatokat határoló vessző már önmagában egy tokennek felel meg.

A tokenizálótól az alábbi kimenetet várjuk el: a sortörésekkel mondatokra tagolt szöveget úgy bontja szét még kisebb egységekre, hogy minden tokent új sorba helyez. Új sorba kerül tehát például egy szó, egy írásjel, egy dátum vagy egy rövidítés. A mondathatárok sem vesznek el, mivel a mondatokra bontó modul olyan szöveget állított elő nekünk, amelyben minden mondat után ki van hagyva egy sor. Új token esetén viszont csak új sort kezdünk, sort nem hagyunk ki. Így könnyedén elkülöníthetők a különböző egységek határai. Az 6.7. ábrán láthatjuk, hogy milyen az elvárt kimenet.



6.7. ábra. Tokenekre bontott szöveg

Hasonlóan az előző modulhoz, első lépésként szabályokat fogalmaztunk meg arra vonatkozólag, hogy mit tekintünk tokennek. Először a szótokeneket vizsgáltuk meg azok könnyebb kezelhetősége miatt. A legáltalánosabb szabály erre az, hogy minden olyan betűkből álló karakterlánc, amelyet szóköz, központozó írásjel (kérdőjel, felkiáltójel, nem rövidítést jelölő pont stb.) vagy csukó zárójel követ. Más típusú (nem szó) tokenek formális leírása nem jelent különösebb nehézséget: például a szövegben gyakran előforduló dátumok, időpontok. A rövidítéseknél arra kell ügyelni, hogy az azok részét képező pont semmiképpen ne számítson külön tokennek (ellentétben egy mondatvégi ponttal, amely az).

Előfordulnak olyan esetek is, amikor egy szót toldalékkal látunk el, de valamilyen oknál fogva (a magyar helyesírás szabályainak megfelelően) a toldalékot nem írhatjuk egybe a szótővel, hanem kötőjellel kapcsoljuk. Például: *június 16-án*, *conj.-án* (conjunctiván, 'kötőhártyán'). A kötőjeles írásmód ellenére a toldalék a token a része, így ügyelni kell arra, hogy programunk a toldalékokat semmiképpen se válassza le a tőről.

6.4.2.2 Implementáció

A tokenizáló modul hozzáadása során módosítanunk kellett néhány korábban elkészített függvényt, illetve új függvényekkel bővítettük a programot.

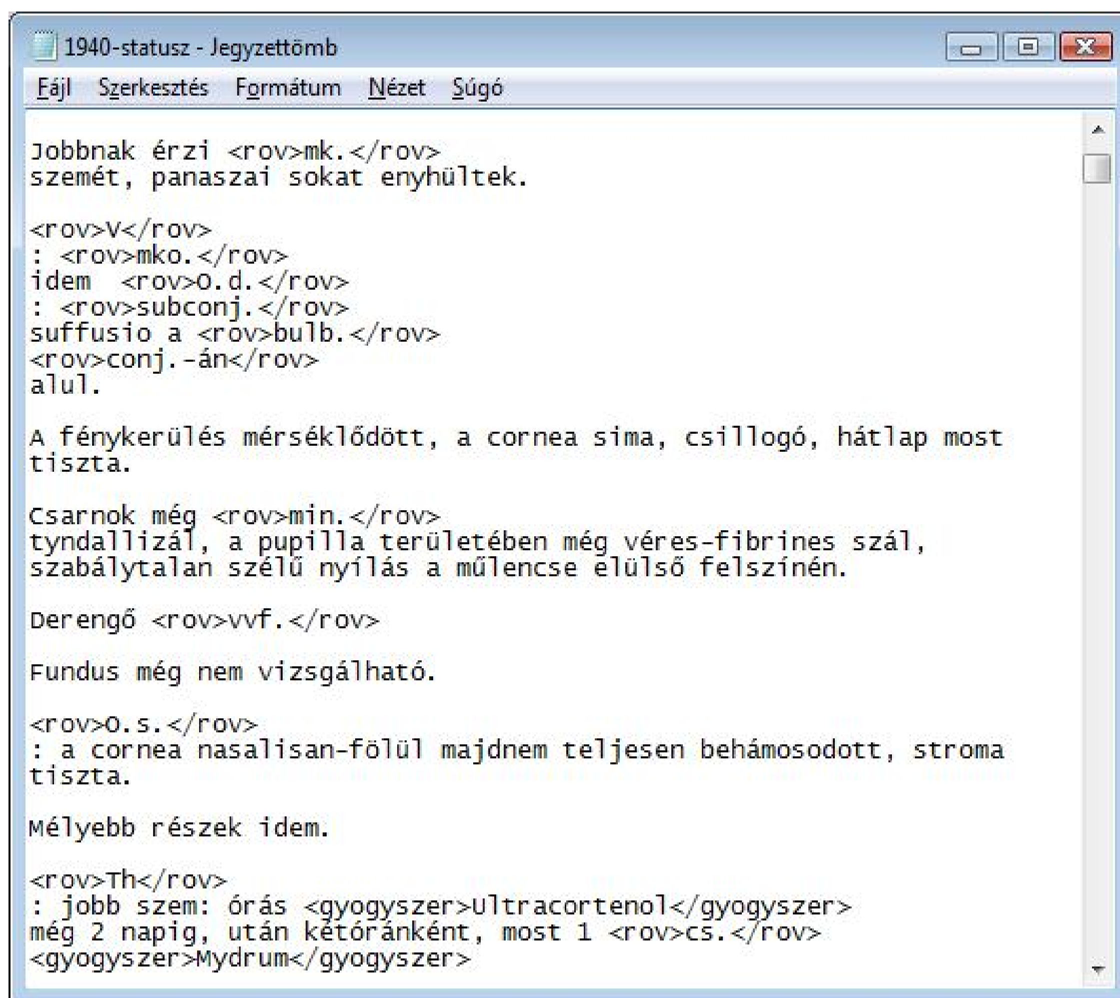
Ahhoz, hogy az imént említett problémát – a kötőjellel kapcsolt toldalék és a szótő egységként kezelését – megoldjuk, az adott típusú tokenek megcímkézéséért felelős függvényeken kellett néhány apróbb módosítást elvégezni. A rövidítéseket megjelölő tagAbbr() függvény eredeti változata – maradva a korábban említett *conj.-án* példánál – a kövezőképpen jelölte meg a rövidítést: `<rov>conj.</rov>-án`. Bár a megjelölés helytálló, hiszen a rövidítés ragozatlan, szótári alakja valóban csupán a conj., nyelvtechnológiai nézőpontból a ragozott alak számít egy egységnek. (Természetesen lehetséges a tovább-bontás, de jelen feladatnak nem célja az orvosi szövegek teljes körű nyelvtani elemzése.) A címkéző függvény magját képező reguláris kifejezést módosítását követően az alábbi kimenetet kapjuk: `<rov>conj-án.</rov>`. Megjegyzendő, hogy erre a változtatásra nem csupán a tokenizáló modulhoz volt szükség, hanem – mint majd látni fogjuk – a következő részfeladatban, a rövidítések egységes kezeléséért felelős modulban is szükség lesz rá: hiszen látni kell majd, hogy a *conj.* és a *conj.-án* tokenek esetén – bár alakjuk eltérő – ugyanarról a rövidítésről van szó, csak az egyik alak toldalékkal van ellátva.

Ugyanezt a módosítást kellett elvégezni még néhány másik címkéző függvénynél – dátumokat, időpontokat felismerő függvényeknél –, mivel azok is előfordulhatnak ragozott alakban. Ez a gyógyszernevekre is érvényes, hiszen azok általában idegen szavak, s így – a szabályoknak megfelelően – kötőjellel kapcsoljuk hozzájuk a toldalékot: *Neosynephrine*, *Neosynephrine*-t (helyesen: *Neo-Synephrine*).

A tokenizáló modul működése a következőképpen történik:

0. A program megkeresi azokat az eseteket, amelyek speciális bánásmódot igényelnek (dátumok, időpontok, rövidítések, gyógyszernevek), és ellátja őket a megfelelő címkével. Ez a folyamat voltaképpen csupán nulladik lépés, mivel már az előző modulban, a mondatokra bontáskor megtörténik. A tokenizáló modulban annyi változás történik, hogy a címkéző függvények nem csupán

címkével megjelölik az adott tokeneket, hanem – a feladatspecifikációnak megfelelően – új sorba töréssel el is különítik azokat. A végeredményt láthatjuk a 6.8. ábrán.


1940-statusz - Jegyzettömb

Fájl Szerkesztés Formátum Nézet Súgó

Jobbnak érzi <rov>mk.</rov>
szemét, panaszai sokat enyhültek.

<rov>V</rov>
: <rov>mko.</rov>
idem <rov>o.d.</rov>
: <rov>subconj.</rov>
suffusio a <rov>bulb.</rov>
<rov>conj.-án</rov>
alul.

A fénykerülés mérséklődött, a cornea sima, csillogó, hátlap most tiszta.

Csarnok még <rov>min.</rov>
tyndallizál, a pupilla területében még véres-fibrines szál, szabálytalan szélű nyílás a műlencse elülső felszínén.

Derengő <rov>vvf.</rov>

Fundus még nem vizsgálható.

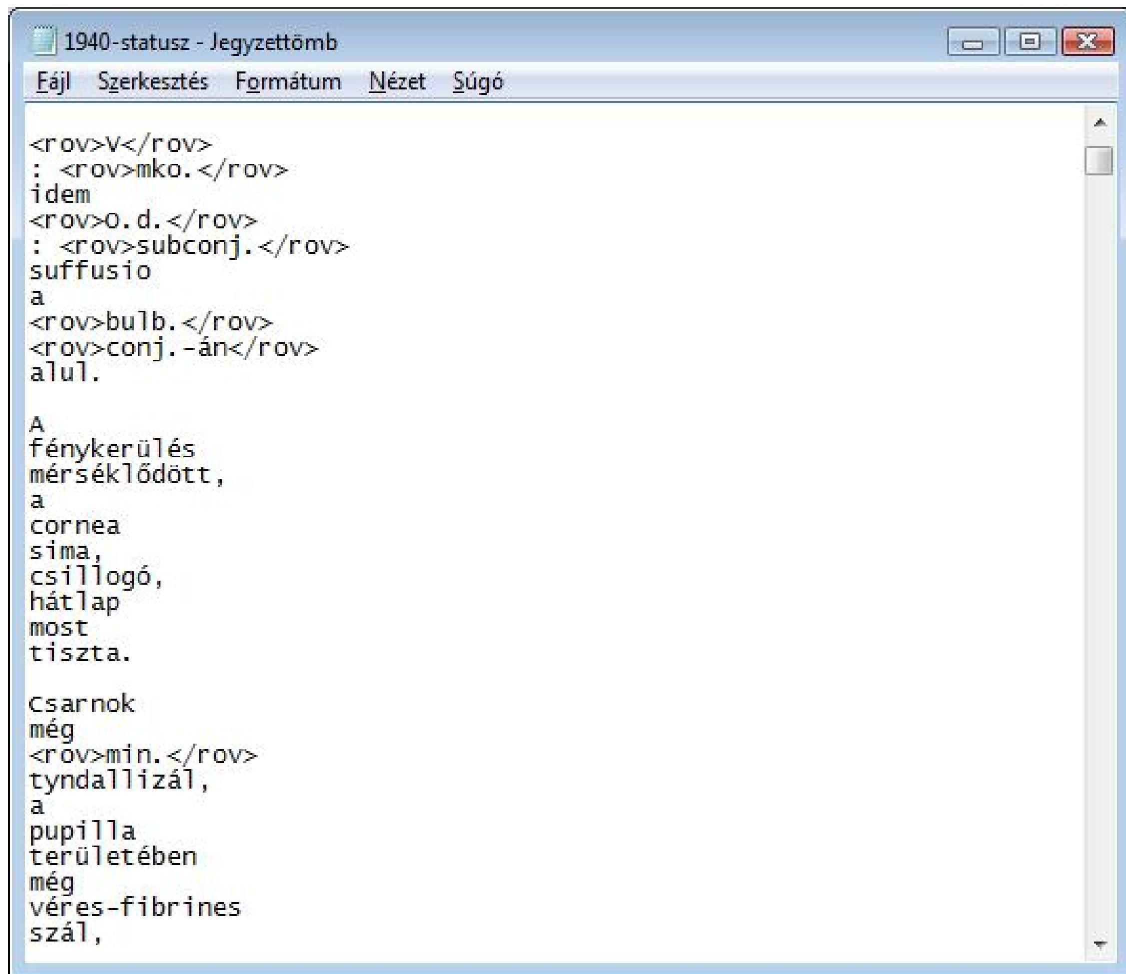
<rov>o.s.</rov>
: a cornea nasalisan-fölül majdnem teljesen behámosodott, stroma tiszta.

Mélyebb részek idem.

<rov>Th</rov>
: jobb szem: órás <gyogyszer>Ultracortenol</gyogyszer>
még 2 napig, után kétóránként, most 1 <rov>cs.</rov>
<gyogyszer>Mydrum</gyogyszer>

6.8. ábra. A tokenizáló tervezésének első lépése: sortörés a rövidítések után

1. Miután a speciális eseteket már megjelöltük és elkülönítettük, következik a tulajdonképpeni szavakra bontás, amelyet a tokenize() függvény végez. Szónak veszünk minden olyan karakterláncot, amely tetszőleges számú (legalább egy) kis- és/vagy nagybetűből áll, és szünet vagy valamilyen írásjel követi. Az írásjel lehet pont, vessző, kérdőjel, felkiáltójel, kettőspont, pontosvessző vagy csukó zárójel. Az összes, a szabálynak megfelelő karakterláncot új sorba helyezzük (6.9. ábra).



6.9. ábra. A tokenizer szavakra bontja a szöveget

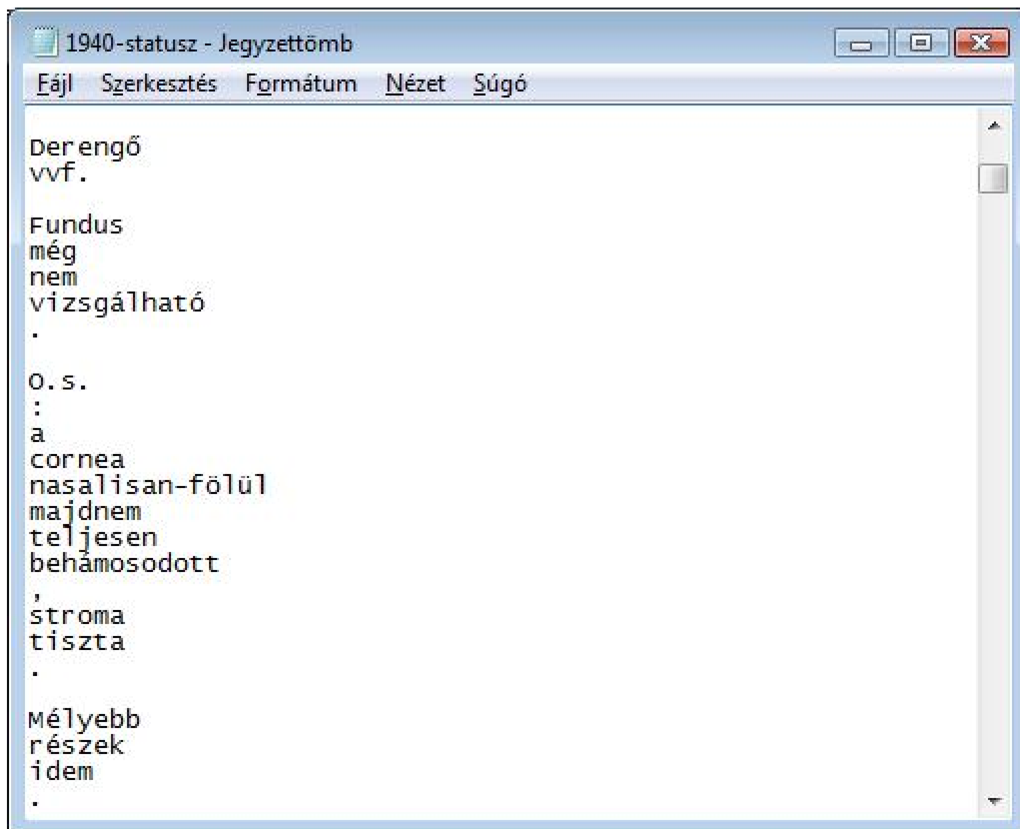
Itt érdemes még kitérni a kötőjelre, amelyet – tudatos döntés eredményeként – nem tekintettük különálló tokennek. Korábban már kitértünk ezen írásjel toldalékkapcsoló szerepére, de más esetekben is előfordulhat. Bizonyos összetett szavakban is megjelenhet mint tagoló – ekkor, noha szerepe szerint elválaszt, mégis egyértelműen a szó része. Más esetekben éppenhogy nem szétválaszt, hanem összefűz, a szavak közti lazább kapcsolatot jelzi. Jó példa erre a 6.7. ábra alján látható *véres-fibrines* kifejezés. A két kifejezés logikailag egybetartozik, ezért a hasonló eseteket következetesen mindig egy egységnek tekintjük.

2. Már csak az írásjelek vannak hátra, ezeket is külön egységként kell kezelni (kivéve a korábban említett eseteket). Leválasztunk minden olyan írásjelet, amely valamilyen betűkből álló karakterláncot követ, a műveletet a punct() függvény végzi. Az írásjel lehet kettőspont, pontosvessző, pluszjel, vessző, csukó zárójel. Ugyanígy különválasztjuk a nyitó zárójeleket is,

amennyiben betűkből álló karakterlánc követi őket. A pontokra vonatkozó szabály az alábbi: bármilyen, legalább egy karakternyi, betűkből álló karakterlánc előzi meg, nem lehet előtte > jel, és utána nem következhet `</rov>` karaktersorozat. Ezzel azt a hibajelenséget védjük ki, hogy a punct() függvény a rövidítésen belül is különválassza a pontokat.

3. Utolsó lépésként eltávolítjuk a címkéket, ezzel készen is vagyunk.

Az eredményt a 6.10. ábrán láthatjuk.



6.10. ábra. A tokenizáló modul kimenete címkeeltávolítás után

6.4.2.3 Tesztelés, javítás

A tesztelés után még szükséges volt néhány új szabály megfogalmazására, amelyek a számok (kivéve a dátumokban, időpontokban szereplő számok) viselkedésére vonatkozott. Ügyelni kellett arra, hogy a számban szereplő tizedespontot vagy -vesszőt ne tekintse új tokennek a program. A kötőjelet – amennyiben két számértéket köt össze – szintén nem vettük új tokennek.

6.4.3 Normalizálás

6.4.3.1 Tervezés

Miután a szöveget felosztottuk kisebb-nagyobb egységekre, célszerű a kapott tokeneket kategóriákba sorolni: szó, dátum, rövidítés, központozási jel. Jelen feladatnak nem célja egy átfogó kategóriarendszer megalkotása. Ahogy a cím is jelzi, a fókusz a rövidítéseken van. Éppen ezért nem ítéltük szükségesnek az egyes tokenek tipizálását, elégségesnek tartottuk csupán azt feltüntetni, hogy központozási jelről vagy rövidítésről van-e szó. Minden egyéb tokent a szó kategóriába soroltunk be (kivéve azokat az egyszerűen kiszűrhető tokeneket, mint például a dátumok, időpontok), hangsúlyozva, hogy további tipizálás is lehetséges, sőt szükséges, ám ez már meghaladná jelen dolgozat kereteit.

A továbbiakban a szavakkal és a központozási jelekkel nem foglalkozunk – utóbbiakkal akkor, ha rövidítés részeit (is) képezik. A továbbiakban a szövegben fellelhető rövidítésekkel kapcsolatos problémaköröket ismertetjük.

Többször elhangzott már, hogy az orvosi szövegekben igen gyakran fordulnak elő rövidítések. Ezeknek jelentős hányada speciális, az orvosi szaknyelvre jellemző sajátosság, és csak kisebb részük ismert, a köznyelvben is használatos rövidítés (pl. *dr.*, *prof.*). Pontosan a szaknyelvi, speciális, sokszor csak alkalmilag használt rövidítések túlsúlya miatt van a szakdolgozat tárgyát képező feladatnak létjogosultsága: hiszen hiába állnak rendelkezésre egyébként megbízható, nagy pontosságú, magyar nyelvű tokenizáló programok, mint például a Huntoken[5], azok csupán a leggyakoribb rövidítéseket képesek kezelni. Márpedig bizonyos rövidítésekről, amelyekkel a rendelkezésünkre álló klinikai szövegekben gyakran találkozunk (*o.s.*, *mou*, *o.utr*), nem állíthatjuk, hogy a köznyelvben is hasonlóan gyakran előfordulnak. Többségüknek még a jelentése is homályos egy laikus ember számára.

Tekintve, hogy ezen szövegek speciális tematikája miatt nem használhatók a köznyelviszöveg-korpuszok rövidítéslistái a rövidítésfelismeréshez, továbbá mivel nem áll rendelkezésünkre olyan lista, amely tartalmazza a leggyakoribb orvosi nyelvi

[5] http://mokk.bme.hu/resources/huntoken/

rövidítéseket, feltétlenül szükséges egy saját lista előállítása. Noha a lehetséges rövidítések felismerése heurisztikán alapuló módszerrel is történhet (szabályokat fogalmazunk meg arra, hogy milyen formai kritériumok alapján tekinthető egy token rövidítésnek), a lexikonos (listás) megoldás bizonyult a legmegbízhatóbbnak (vö. [12]). Többek között amiatt is, hogy a lista megoldást jelenthet azon esetekre is, amikor a rövidítések után – akár a kivételes helyesírás miatt, akár tévedésből – nem tesznek pontot. Az ilyen esetek reguláris kifejezéssel nem írhatók le, mivel formailag egybeesnek a hagyományos szavakkal. Feltétlenül ilyen módon kell eljárni pl. az alábbi rövidítésekkel: *ünj* 'üveg nem javít', *mou* 'métert olvas ujjat', *fén* 'fényérzés nélkül', *szeou* 'szem előtt olvas ujjat'.

A mondathatár-felismerésről szóló fejezetben már említettük, hogy a feladat megkezdésekor már rendelkezésünkre állt egy rövidítéslista (az előző félévben végzett önállólabor-feladat eredményeképpen). A teljesség kedvéért a továbbiakban röviden ismertetjük a lista előállításának lépéseit.

Kezdetben heurisztikán alapuló módszert alkalmaztunk, követve a Grefenstette–Tapanainen-féle irányelvet [12]. Első lépésként szabályokat fogalmaztunk meg a rövidítések szintaktikájára vonatkozóan:

- pont követi (opcionális),
- a rövidítés után következhet szóköz, vessző, pontosvessző, kettőspont, csukó zárójel stb.,
- a rövidítést megelőzheti szóköz, esetleg nyitó zárójel.

Ha a rendelkezésünkre álló orvosi szövegeket jobban megvizsgáljuk, láthatjuk, hogy ezekben a szövegekben a rövidítést követő karakterek halmaza bővülhet még egyéb jelekkel is, például egyenlőségjellel (*Dsph*=). Következő lépésként reguláris kifejezéssé alakítottuk a szövegesen megfogalmazott szabályokat: `((([a-záéíóöőúüű])+(\.)?)*(\s|:|,|;)*)`. Problémát jelent, hogy mivel a pont opcionális, a reguláris kifejezés gyakorlatilag minden szót lefed, így előzetesen meg kell szűrni a feldolgozandó szöveg szavait. Szűrési feltételül a karakterlánc hosszát választottuk, abból kiindulva, hogy a rövidítések általában – rövidek. A karakterlánc hosszát – némileg önkényesen, de a szövegeket megvizsgálva tudatos döntés alapján –

maximum 6 betűig korlátoztuk. A lehetséges rövidítéseket kereső segédprogram tehát csupán a legfeljebb 6 hosszú sztringeket vette figyelembe, és vizsgálta, van-e illeszkedés. Ha volt: a szót felvette a lehetséges rövidítések listájába. A lista normalizálása (pontok eltávolítása, illetve egységesen kisbetűssé alakítás után) azonban még mindig nem álltunk készen, mivel abban szerepelnek értelmes szavak is. Ugyanis a fent vázolt algoritmus felveszi a listába például a mondatok utolsó szavait is, hiszen a reguláris kifejezés azokra is illeszkedik. Az értelmes szavak kézi eltávolítása helyett gyorsabb megoldást kínált a Hunspell helyesírás-ellenőrző program használata. Abból a feltételezésből indultam ki, hogy ugyanis amit az felismer, az nagy valószínűséggel értelmes szó, tehát biztosan nem orvosi nyelvi rövidítés. Mivel azonban a Hunspell a köznyelvben elterjedt rövidítéseket is ismeri, és pontosan emiatt el is távolította volna a listából, a script lefuttatása előtt szét kellett válogatni a lehetséges rövidítések listáját, és új listát készíteni azokból a sztringekből, amelyek biztosan nem köznyelvi rövidítések. Ezt a listát adtuk át az alábbi egyszerű shell scriptnek:

```
#/bin/bash

cat $1 | hunspell -l > $2
```

A script kimenetként kapott listát azonban szükséges volt még egyszer átnézni, mivel maradhattak benne olyan értelmes szavak, amelyeket a Hunspell nem ismer fel. Ilyenek például a latin szavak (*idem*, *cornea*), amelyek nem rövidítések. Bár nem célszerű az utólagos kézi módosítás, de jelen esetben a nem megfelelő szavak egyenkénti kézi eltávolítása kínálkozott a leggyorsabb, legegyszerűbb megoldásnak. Utolsó lépésként össze kell fésülnünk a kapott listánkat az ismert rövidítések listájával, amelyet még a Hunspell futtatása előtt választottunk le. Ezzel készen is vagyunk.

Ezt a listát indításkor paraméterként kapja meg a tokenizáló program. Pontosan emiatt természetesen tetszőleges, egyszerű szöveges fájlként megadott rövidítéslistával is működik a program. A rövidítéslista tetszőleges számú elemmel bővíthető, nem kell mást tenni, csupán begépelni az új eleme(ke)t a lista végére. (A rövidítések betűrendben szerepelnek, de ez a program működését nem befolyásolja.) Fontos, hogy a rövidítést – ha tartalmaz pontot – úgy kell begépelni, hogy azt a pontot, amely utolsó karakterként szerepel, elhagyjuk. A hiányzó pontot már a program illeszti hozzá, amennyiben

szükséges. Például ha fel szeretnénk venni a listába a *conj.* elemet, akkor a *conj* karaktersorozatot kell begépelnünk.

Egy rövidítés azonban nem csak ilyen formájú lehet, tartalmazhat például több pontot is. A szemléletesség kedvéért vegyük az *i. m.* 'idézett mű' példát. Mivel a listában szereplő elemek mind-mind reguláris kifejezések (erről bővebben a következő fejezetben, a program megvalósításáról szóló részben szólunk), a többtagú rövidítés belsejében szereplő pontokat ki kell íratnunk. Ahhoz, hogy a program ne wildcard karakterként értelmezze őket (a reguláris kifejezésben szereplő pont bármilyen, nem újsor karakternek felel meg), escape-elnünk kell a pontokat. A példánál maradva: a listába az *i\.m* karaktersorozatot kell tehát felvennünk.

A specifikáció része az is, hogy nem csupán felismerni kell tudni a rövidítéseket, hanem jelezni kell azt is, hogy egy-egy azonos jelentésű, de formailag eltérő rövidítés esetén ugyanarról a kifejezésről van szó. Nézzük az alábbi példákat:

(1) *exc.*
(2) *excav.*
(3) *max*
(4) *max.*
(5) *o.s.*
(6) *os.*
(7) *o.sin*
(8) *conj.*
(9) *conj.-án*

Az (1), (2) példák ugyanannak a szónak (excavatio 'vájulat, kivájás') a rövidítései. A (3), (4) esetben a maximum szó kétféle – pont nélküli és pontra végződő – rövidítése látható. Az (5), (6), (7) példák szintén ugyanannak a kifejezésnek a különböző formájú rövidítései (oculus sinister 'bal szem'). Láthatjuk, hogy itt már nem csupán a pont léte vagy hiánya okozza a különbséget, hanem az is, hogy mennyire rövidítjük le a második tagot (sinister). A variációk száma az ilyen típusú rövidítéseknél meglehetősen nagy.

Végül a (8), (9)-es példáról szólunk, ahol maga a rövidítés ugyan nem különbözik, a (9)-es eset a (8)-asnak a toldalékolt alakja. Ugyanarról a rövidítésről lévén szó, itt sem szabad különbséget tenni a szótári alak és a toldalékolt alak között.

A rövidítések egységes kezelésének legegyszerűbb módja az, ha minden rövidítés valamilyen egyedi azonosítót kap, például egy számot (ID). Ha két eltérő alakú rövidítésnek megegyezik az azonosítója, abból láthatjuk, hogy ugyanarról a kifejezésről van szó. A fenti példáknál maradva ugyanazt az ID-t kell kapnia az (1)-(2), a (3)-(4), az (5)-(6)-(7) és a (8)-(9)-es példának.

Ennek megvalósítását a következő fejezetben ismertetjük.

6.4.3.2 Implementáció

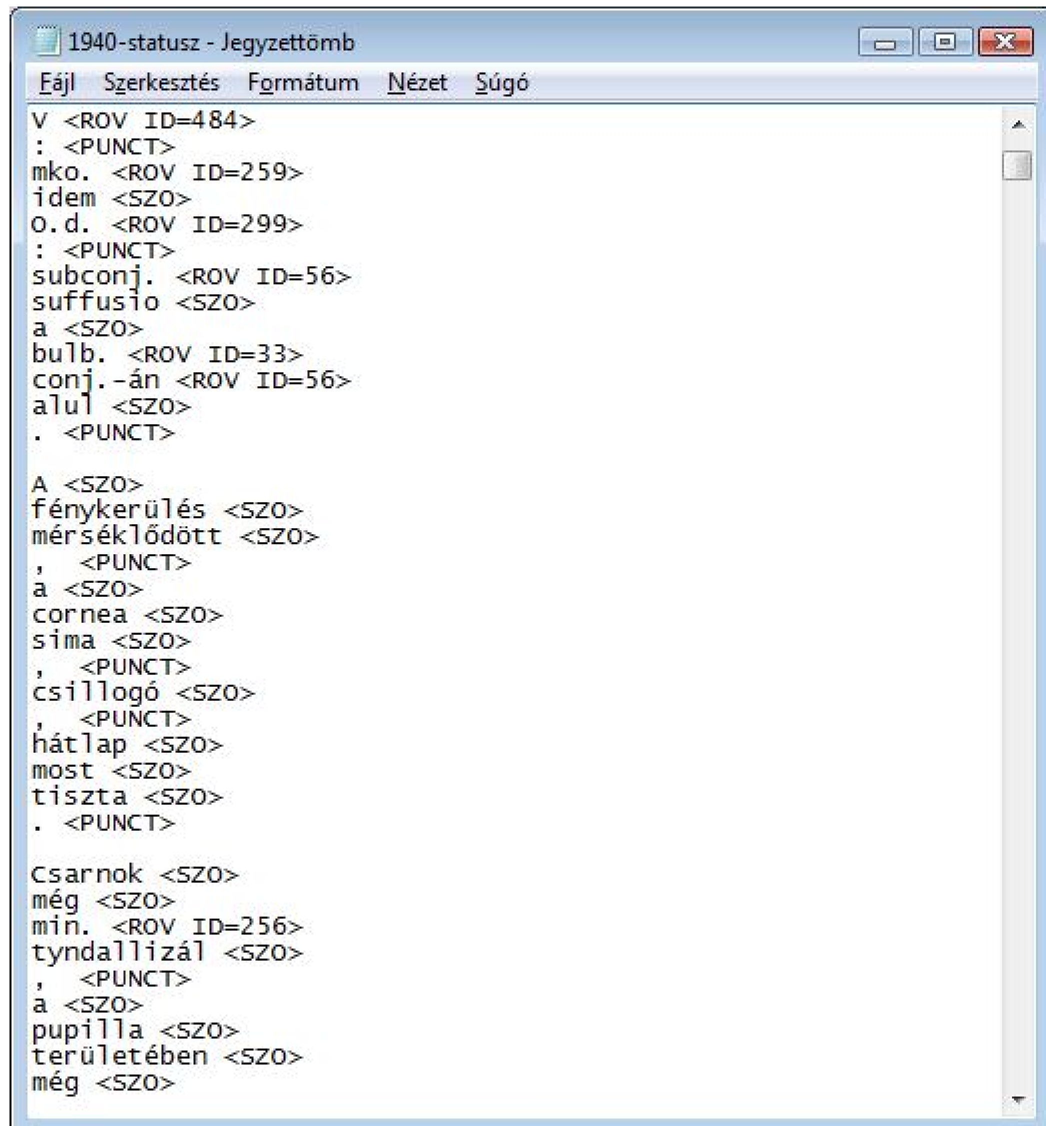
A rövidítések felismerését és megjelölését – mivel az a mondathatár-egyértelműsítéshez feltétlenül szükséges – már az első modulban elvégzi a tagAbbr() függvény, melynek működését a mondathatár-felismerési fejezetben ismertettük. A tokenizáló modulban eltávolítjuk az XML-szerű címkéket (`<rov></rov>`), ezeket csupán segédcímkének használtuk. Kiindulási állapotunk látható az előző fejezetben, a 8. ábrán: minden szó, dátum, időpont, központozó jel és rövidítés új sorban szerepel.

Definiáltunk egy adID() nevű függvényt, amely a megjelölni kívánt szövegen kívül még a rövidítéslistát kapja paraméterül. Működése hasonlít a rövidítéscímkéző függvényéhez: miután beolvasta a címkézni kívánt szöveget és a rövidítéslistát, egyenként végigmegy a lista elemein. Mint már az előző fejezetben említettük, a rövidítéslista tulajdonképpen egy reguláris kifejezéseket tartalmazó lista, amelyet a program tovább alakít. Amennyiben szükséges, a listaelemhez illeszt egy pontot, illetve – ugyancsak opcionálisan – a kifejezéshez illeszti a `(\-[a-záéíóöőúüű]+)?` sztringet. Ez utóbbival azt állítjuk be, hogy ne csak a szótári alakú, hanem a toldalékolt rövidítésekre is illeszkedjen a reguláris kifejezés. A függvény tehát azt vizsgálja, illeszkedik-e az aktuális tokenre az alábbi reguláris kifejezés: `rule + "(\.)?(\-[a-záéíóöőúüű]+`, ahol „rule” a lista aktuális elemét jelenti. Ha illeszkedik, akkor a rövidítés után elhelyez egy `<ROV ID=…>` címkét, ahol ID az adott rövidítés azonosítóját jelenti, értéke pedig egy szám. Az azonosító számot dinamikusan generálja a program akkor, amikor bejárja a rövidítéslistát. Egy id nevű integer változóban tároljuk az

azonosítót, amely kezdetben 0, és minden egyes listaelemre való lépéskor eggyel megnöveljük az értékét – vagyis tulajdonképpen számlálóként működik. A `<ROV>` címkével együtt az ID értékét is kiíratjuk (sztring típusú adattá alakítva). Mivel az azonos jelentésű, eltérő alakú rövidítéseket a lista ugyanazon eleme fedi le, így értelemszerűen ugyanazt az azonosítót kapják a szövegben, bármelyik alakváltozatról legyen szó. Így például az *oculus dexter* összes eltérő alakú rövidítése a 299-es azonosítót kapta: *o.d*, *o.d.*, *o. d.*, *od.*, *o.dex*, *o.dex.* stb., beleértve természetesen a nagybetűs alakokat is.

A normalizálás utolsó lépése az összes többi token megcímkézése. Mint a korábbiakban kiemeltük, egyáltalán nem célunk a többi token részletes kategorizálása, így csupán a szavakat, számokat, központozó jeleket és a reguláris kifejezésekkel könnyen kiszűrhető, egyéb típusú tokeneket címkézzük meg. Ez könnyen megtehető a tokenizáló modul megfelelő függvényeinek módosításával (tokenize(), punct()). A módosítás csupán annyi, hogy a tokenizáló függvény a tokenhatáron ne csupán sort törjön, hanem jelölje is meg a tokent mint szót, számot vagy központozó jelet (ez utóbbit `<PUNCT>` címkével jelöljük). A dátumok és időpontok átcímkézését a címkeeltávolító (untag) függvények módosításával oldottuk meg.

A végleges eredményt a 6.11. ábrán láthatjuk.



6.11. ábra. A végső kimenet: címkézett tokenek, ID-val ellátott rövidítések

7 Az elkészült tokenizáló tesztelése, értékelése

A munka utolsó fázisa az elkészült program tesztelése. Ebben a fejezetben áttekintjük, milyen módon történt a tesztelés, milyen eredményeket kaptunk, ezek alapján következtetéseket vonunk le, és szót ejtünk az esetleges továbbfejlesztési lehetőségekről.

7.1 A tesztelés módja

A kiértékelés úgy történik, hogy a kimenetként kapott eredményeket összehasonlítjuk az elvárt eredményekkel. Az elvárt eredmény esetünkben egy nagyobb méretű szövegfájl, amelyben előre be vannak jelölve a mondathatárok, tokenhatárok és a rövidítések. A jelölés módja ugyanaz, mint amilyen módszert a program használ. Ehhez az annotált dokumentumhoz, a referenciához hasonlítjuk a program által generált kimeneteket.

Eredeti tervünk szerint a teljes anyag 10%-át tesztelési célokra különítettük el (ez összesen 130 db fájl). Mivel azonban nem létezett megfelelően címkézett korpusz, amelyet referenciaként használhattunk volna, változtatnunk kellett az elképzelésen. Maga a kézi annotálás igen időigényes, hosszú folyamat, amely kötelezően nem képezi részét a féléves munkának, a teszteléshez viszont mégis elengedhetetlenül szükséges. Éppen ezért a referenciafájl előállításához végül csupán négy-öt hosszabb szövegfájlt használtunk fel (a teljes dokumentumhalmaznak ez kb. 3-4%-a): a fájlok összefűzése után lefuttattuk a programot, és a gépileg generált mondat- és tokenhatárok, felismert rövidítések hiányosságait kézzel korrigáltuk. A rövidítések felismerése olykor nehézségeket okozott – nem lévén orvosi szakmabeli –, az esetek többségében azonban az interneten való keresés, olykor a szakemberekkel történő konzultáció megoldást hozott. Az említett módszerrel lényegesen gyorsabban előállítottuk azt a mintát, amelyhez a tesztelendő szövegeket hasonlítani fogjuk. A referenciaanyagot úgy válogattuk össze, hogy lehetőleg lefedje az összes esetet (a rövidítések minél változatosabb környezetben forduljanak elő, pl. előzze meg őket szóköz vagy nyitó zárójel, és kövesse csukó zárójel vagy vessző vagy kettőspont stb.). Ezenkívül ügyeltünk arra is, hogy többféle orvosiszöveg-típus helyet kapjon a referenciaanyagban (úgymint: anamnézis, epikrízis, javaslat, státusz). Ezen szempontokat figyelembe véve

elmondható, hogy a tervezettnél kisebb terjedelmű kézzel annotált korpusz is viszonylag reális képet ad majd az elkészült program hatékonyságáról.

A tesztfájl, amelyet összehasonlítunk a referenciával, a tesztelendő szoftver által generált dokumentum: tokenekre bontott szöveg, ahol jelölve vannak a rövidítések és a mondathatárok, utólagos kézi javítás nélkül.

Annak ellenére, hogy a tesztanyag viszonylag kis méretű (~48 kB), a referencia és a kapott eredmény kézi összehasonlítása rendkívül figyelem- és időigényes. A szövegben a tokenek száma 3300, és ezen felül még a mondathatárokat és a rövidítéseket is vizsgálni kell, melyek mennyisége szintén ilyen nagyságrendű. Célszerűnek látszott ezért a kiértékelés automatizálása, melyet egy kifejezetten erre a célra készített segédprogram végez el. A kiértékelő program bemeneti paraméterként kapja meg a referenciafájlt és a tesztelendő szöveget, összehasonlítja, majd ennek végeztével megjelenít a képernyőn bizonyos mérőszámokat, amelyek a tokenizáló program hatékonyságát jellemzik. Ezen mérőszámokról a következő fejezetben szólunk.

7.2 Fedés, pontosság

A kapott eredmények és az elvárt eredmények összehasonlítása során többféle szempont szerint vizsgálódhatunk.

Megvizsgálhatjuk például, hogy az összes rövidítés, mondat- és tokenhatár közül az elkészített programnak hányat sikerült megtalálnia: ezt fedésnek (recall) nevezzük. Egy másik szempont, ha azt vizsgáljuk, hogy a kapott eredmények közül hány bizonyult jónak, azaz mekkora a pontosság (precision).

A fedés és a pontosság harmonikus közepét adja meg az ún. F-mérték.

$$F = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

Ennek egy változata az F_β-mérték, ahol β egy paraméter, amellyel a súlyozás mértékét lehet állítani. Ennek akkor van jelentősége, ha azt szeretnénk, ha a fedés nagyobb súllyal essék latba a pontossághoz képest, vagy fordítva. A feladat céljától,

környezetétől, a szöveg típusától függ, hogy a fedés vagy a pontosság számít-e jobban. Van, hogy a minél nagyobb találati arány a cél (ezek között esetleges téves találatokkal), ilyenkor a β értékét 2-re állítva a fedés nagyobb súllyal lesz figyelembe véve. Míg ha 0,5-öt adunk meg β értékének, akkor a pontosságon lesz a hangsúly. Utóbbi esetben a cél: inkább legyen kisebb a találati arány, de az ebben szereplő elemek közül minél több legyen helyes találat.

$$F_\beta = (1+\beta^2) \cdot \frac{precision \cdot recall}{\beta^2 \cdot precision + recall}$$

Esetünkben az $F_{0,5}$-mértéket fogjuk kiszámolni, mivel fontosabbnak ítéltük meg azt, hogy a téves találatok számát a minimálisra csökkentsük. Inkább legyen kevesebb token rövidítésként megcímkézve, minthogy esetleg olyan szavakat is rövidítésként jelöljünk meg, amelyek nem azok. Ugyanez elmondható a mondathatárokról is. Az igen nehezen kiszűrhető mondathatárok (tipikusan azok az esetek, amikor nincsenek mondatvégi írásjelek) felismerése programunk egyik gyenge pontja, ám ennek megoldása csak úgy lenne lehetséges, ha a többi – egyébként helyesen működő – mondathatár-felismerő algoritmust módosítanánk, s nagy valószínűséggel el is rontanánk. Mivel az írásjel nélkül előforduló mondatok száma elenyésző a szabályos mondatokéhoz képest, tudatos döntés eredménye volt az, hogy a rendhagyó mondatok határainak felismerése korántsem működik tökéletesen. Másképpen megfogalmazva: nem volt célunk a minél nagyobb mértékű fedés elérése, inkább arra koncentráltunk, hogy ne forduljanak elő téves mondathatárok, azaz a pontosság értéke legyen minél magasabb.

Mielőtt a számadatokat ismertetnénk, röviden kitérünk a kiértékelő program működésére, valamint arra, hogy milyen engedményeket kellett tennünk a fedés és a pontosság számolásakor ahhoz, hogy ne kapjunk túlságosan alacsony mérési eredményeket, ugyanakkor a kapott eredmények mégis reálisan tükrözzék az elkészült tokenizáló hatékonyságát.

Külön-külön vizsgáljuk a fedést és a pontosságot a mondat- és tokenhatárok, valamint a rövidítések felismerésére. Mindhárom esetben a megfelelő részeket vesszük figyelembe: így például ha a rövidítések felismerésének hatékonyságát vizsgáljuk, mind a fedés, mind a pontosság szempontjából irreleváns, hogy a program felismerte és

szegmentálta-e az összes központozó jelet (kivéve természetesen, ha az a rövidítése része, ilyenkor természetesen nagyon is fontos, hogy ne kerüljön új sorba az ominózus pont).

Attól függően, hogy mit vizsgálunk, a kiértékelő program – reguláris kifejezés segítségével – kiválasztja a referenciafájlból a releváns elemeket, és egyenként felveszi azokat egy listába (az összes mondathatárt, az összes tokent vagy az összes rövidítést). Ugyanezt elvégzi a tesztelendő szövegen is. Ezzel időt takarítunk meg, mivel pl. a rövidítések vizsgálatakor nem kell az egész szövegen végigmenni, hanem elég azokat a szavakat megvizsgálni, amelyek rövidítésként lettek megjelölve. Egymásba ágyazott ciklusokkal járja be a program a két listát, és lineáris keresést végezve keresi a találatokat. (Nagyobb adatmennyiség esetén célszerű valamilyen kisebb lépésszámú algoritmust választani, de jelen esetben a lineáris keresés is elegendőnek bizonyult.) A helyes pozitív, helyes negatív, hamis pozitív és hamis negatív találatok alapján számolja ki a program a fedést és a pontosságot, majd ezek alapján az F-mértékeket.

Ahhoz, hogy értékelhető eredményeket kapjunk, az alábbi engedményeket tettük:

1. Mint korábban említettük, programunk a rövidítéseket felismerésük során nemcsak címkével, hanem azonosítószámmal is ellátja. A referencia- és a tesztfájl összehasonlítása során előfordulhat, hogy – a dinamikus azonosítókiosztás miatt – ugyanaz a rövidítés más ID-vel szerepel a két szövegben, ami elrontja a felismerést. Hiába van szó ugyanarról a rövidítésről, ha az azonosítójuk különböző, a kiértékelő program emiatt nem venné találatnak (vagy hamis találatnak vélné). Éppen ezt kiküszöbölendő a kiértékelő algoritmus csupán a rövidítést jelző címke meglétét figyeli, az azonosítót nem.
2. A tokenekre bontó modul alacsony szintű kategorizálást végez a tokenek esetén: <SZO>, <SZAM> vagy <PUNCT> címkével jelöli meg azokat a tokeneket, amelyek nem rövidítések. Kategóriatévesztés (pl. egy számot szóként címkézünk meg) esetén a kiértékelő program nem ismerné fel a helyes találatot, hiába tehát a tokenhatár pontos megtalálása.
3. Mivel a tokenek típusokba sorolása nem tartozik szorosan a feladathoz, az értékelés során ezeket nem is vettük figyelembe. Ha a token határa egyébként

megfelelő helyen van, függetlenül a kategóriacímkétől a program helyes találatként fogja azt elkönyvelni.

```
Kijelölés C:\Windows\system32\cmd.exe
              22 fβjl              180á578 bβjt
               6 k÷nyvtβr  61á439á160á320 bβjt szabad

C:\Users\Zs˜fi\PythonWorkspace\csse120\Sentence Boundary Disambiguation>pyth
ompare.py gold_standard.txt gold_standard_teszt.txt
*** Roviditesek ***
------------------------
Fedes: 0.908641975309
Pontossag: 0.968421052632
F-mertek: 0.937579617834
F-beta: 0.955844155844
------------------------
*** Mondathatarok ***
------------------------
Fedes: 0.959854014599
Pontossag: 0.981343283582
F-mertek: 0.970479704797
F-beta: 0.976968796434
------------------------
*** Tokenhatarok ***
------------------------
Fedes: 0.821149751597
Pontossag: 0.942950285249
F-mertek: 0.87784522003
F-beta: 0.915782808295
------------------------

C:\Users\Zs˜fi\PythonWorkspace\csse120\Sentence Boundary Disambiguation>
```

7.1. ábra. A kiértékelő program kimeneti képernyője

A kiértékelő program kimenetét a 7.1. ábrán láthatjuk, a kapott eredményeket 7.1-es táblázat foglalja össze:

	Rövidítések	Mondathatárok	Tokenhatárok
Fedés	90,86%	95,98%	82,11%
Pontosság	96,84%	98,13%	94,29%
F- mérték	93,75%	97,04%	87,78%
$F_{0,5}$-mérték	95,58%	97,69%	91,57%

7.1. táblázat. A kiértékelés eredményei

A korábban már ismertetett szemlélet – az inkább kisebb, de annál pontosabb találati arányokra való törekvés – a számadatokon is látszik. Mindhárom esetben nagyobb volt a pontosság mértéke, mint a fedésé.

A kettő harmonikus közepe, az F-mérték alapján megállapíthatjuk, hogy a rövidítések és a mondathatárok esetében a felismerés elég jónak mondható (95%-on felüli eredmény). Ennél kevésbé jó, de még elfogadható eredményt ért el a program a tokenekre bontásnál, ahol az F-mérték nem éri el a 90%-ot, bár nem sokkal marad el mögötte (~88%). Ha viszont a pontossággal súlyozott F_{β}-mértéket nézzük, mindhárom esetben 90% fölötti eredményt sikerült elérnünk.

Némileg meglepő, hogy a legjobb eredményeket a mondathatár-felismerési modul produkálta (~96%-os fedés, ~98%-os pontosság). Arra számítottunk ugyanis, hogy a rövidítés-felismerés lesz a legpontosabb, éppen azért, mert az lexikont használ a keresett elemek megjelöléséhez, nem pedig csupán heurisztikán alapul, mint a mondat- és tokenhatár-felismerés. (Igaz, a rövidítésfelismerő modul eredménye nem sokkal maradt le a mondathatár-felismerőé mögött.) A csaknem 100%-ot megközelítő mérőszámok arról árulkodnak, hogy az elkészített program orvosi szövegeinkben csaknem az összes lehetséges mondathatárt képes megtalálni, téves találatot pedig igen ritkán ad. A meglepően jó eredményekre magyarázat lehet az, hogy a feldolgozandó szövegekben a mondathatárokra vonatkozó szabályok viszonylag pontosan megfogalmazhatók, az egyes esetek világosan elkülöníthetők egymástól (pl. mikor jelzi mondat végét a pont, és mikor nem). Ha a megfogalmazott szabály hamis pozitív eredményeket is produkál (vagyis olyan eseteket is lefed, amelyeket nem kellene, például új mondatkezdetnek vesz minden pont utáni nagybetűs szót), névelem-felismerési technikákat segítségül híva könnyedén kiküszöbölhető a hibás működés.

A mondathatár-felismerő modul hiányossága az, hogy nincs kellőképpen felkészítve az olyan mondatok felismerésére, amelyek nem végződnek írásjelre. Hogy ezen mégis mondatok, azt a szövegtagolásból láthatjuk (új sor következik), és természetes nyelvérzékünk is jelzi a szövegegység mondat mivoltát. Például:

Junius 29.-én kontroll

2010.06.29 08:25

A mondatot követő dátum már a következő tartalmi egység kezdetét jelzi, tehát biztosan mondatról van szó, mégsincs írásjel a végén. Olyan szabályt viszont nem fogalmazhatunk meg, hogy ha dátum következik, az már mindenképpen új mondat, hiszen a magyar nyelvre jellemző a dátumok mondatba ékelődése (anélkül hogy új

mondatot kezdenénk), így ha szabályt fogalmaznánk meg erre a néhány esetre, azzal elrontanánk a többi jó találatot.

A mondatfelismerő modul tehát az írásjelre nem végződő mondatok felismerése területén szorul továbbfejlesztése, ám kérdéses, hogy ez megoldható-e tisztán szabály alapon.

Szintén jó eredményeket ért el programunk a rövidítések felismerése során (~91%-os fedés, 97%-os pontosság). Annak, hogy a fedés nem sokkal haladta meg a 90%-ot, a következő lehet az oka: bár a modul lexikont használ a rövidítésfelismeréshez, a lexikont magát úgy hoztuk létre, hogy heurisztikus módszerekkel próbáltuk kinyerni a szövegben előforduló lehetséges rövidítéseket. Ez a módszer jóval több hibalehetőséget rejt magában, ami mind a fedés, mind a pontosság értékét befolyásolja. Hiszen annak a szabálynak a megfogalmazása, hogy milyen lehet egy rövidítés, már jóval nehezebb, mint a mondathatárra vonatkozó szabályoké, tekintettel a rövidítések alakjának sokféleségére (egytagú, többtagú, pontra végződő, nem pontra végződő stb.). Csökkenthette a fedést az is, hogy a rövidítések hosszát maximum 6 karakterre korlátoztuk (azzal az indokkal, hogy a rövidítések általában rövidek, bár a rövidség maga relatív fogalom). Emiatt eredetileg nem kerültek bele a lexikonba azok a szavak, amelyek rövidítések, de 6 karakternél hosszabbak (pl. *subconj.*, amely hét betűből áll). Ugyan az ilyen rövidítések száma elenyésző, mégis fontos, hogy a program ne hagyja őket figyelmen kívül. Az ilyen típusú hiányosságok könnyedén javíthatók a lexikonba való utólagos felvétellel. (A korrekció után valóban megnőtt a fedés mértéke.)

Az, hogy a program rövidítésként jelöl meg valamit, ami nem az (csökkentve ezzel a pontosság értékét), ritkán fordul elő. Oka lehet, hogy minden precizitásra törekvés ellenére a lexikonban maradhattak olyan szavak, amelyeket a Hunspell nem szűrt ki, mivel nem ismerte őket (pl. latin szavak), és az utólagos kézi ellenőrzés során sem kerültek törlésre (oka lehet ennek az emberi figyelmetlenség vagy az orvosi nyelvben való nem megfelelő jártasság). A pontosság növelhető a tévesen rövidítésként megjelölt szavak lexikonból történő törlésével.

A legrosszabb eredményeket a tokenizáló modul produkálta. Noha a közel 81%-os fedés sem nevezhető kifejezetten rossznak, jóval elmarad a másik két modul 90%-on

felüli eredménye mögött. Ennek oka abban keresendő, hogy a tokenizáló modul létrehozását éreztük a legösszetettebb, s ezáltal a legnehezebb feladatnak a három közül. Többször hangsúlyoztuk, hogy nem célja a féléves munkának az alapos, minden részletre kiterjedő tokenizálás; a dolgozat címe is mutatja, hogy a rövidítések és mondathatárok felismerésére fókuszálunk mindvégig. Ezzel indokolható az, hogy a tokenizáló modul kidolgozottságára kevesebb időt fordítottunk, mint a másik kettőére. Összességében (a 90%-on felüli F-mértékeket figyelembe véve) ez a modul is kielégítő eredményeket produkált.

7.3 Összefoglalás, továbbfejlesztési lehetőségek

Az alkalmazás fejlesztése során felismert hibák, amelyeket egyelőre nem sikerült kijavítanunk:

- A mondathatár-felismerő modul nem minden esetben találja meg azokat a mondathatárokat, amelyek nem ponttal, hanem valamilyen egyéb módon vannak jelölve a szövegben, vagy pedig egyáltalán nincsenek jelölve.
- A tokenizáló modul bizonyos esetekben nem választja szét a punktuációt és az azt követő szót/számot, annak ellenére, hogy létezik szabály az ilyen esetekre.
- Az olyan hosszú, bonyolult kifejezések szegmentálása, amelyek sok számértéket (egybeírva a rövidítéssel), írásjelet, speciális írásjelet tartalmaznak (tipikusak a visus számértékei), még nem tökéletes.
- Előfordul, hogy a rövidítésfelismerő modul nem címkéz meg bizonyos rövidítéseket, annak dacára, hogy az adott rövidítés megtalálható a lexikonban.

Az ismert hibák kijavításán kívül az alábbi területeken lehetne még továbbfejleszteni az alkalmazást:

– A tisztán szabály alapon történő rövidítésfelismerés a jelek szerint önmagában nem elégséges. Érdemes lenne a későbbiekben más módszerek alkalmazásával tökéletesíteni az eddigi rendszer teljesítményét (statisztikai módszerek, gépi tanulás). Hibrid módszerekkel valószínűleg a mondathatár-felismerés eredményessége is javítható lenne.

– Az orvosi szövegek strukturálttá tételét még jobban megkönnyítené, ha a program nem csupán megkeresné és megjelölné a rövidítéseket, hanem egyúttal fel is oldaná azokat, a jelentést ugyanúgy attribútumként megjelölve, akárcsak az azonosítót. Például:

(1) *szeou* <ROV ID=428 JEL=”szem előtt olvas ujjat”>

(2) *o. d.* <ROV ID=302 JEL=”oculus dexter, jobb szem”>

A jelentések feloldása azonban olyan tudást követel meg, amellyel nem rendelkezik egy átlagember, ezért mindenképpen szükséges lenne szemész szakemberek bevonása is.

– A tokenizáló modul továbbfejlesztése oly módon, hogy az elemi egységeket egyre nagyobb logikai egységekké kapcsolja össze. Mint ahogy az egy mondathoz tartozó szavak is össze vannak fogva egy egységbe, és ez az összetartozás látszik is a feldolgozott szövegen, úgy érdemes lenne feltüntetni más, logikailag összekapcsolódó egységeket is. Jelölni például egy vezetéknév és egy keresztnév, vagy egyéb egységek összetartozását. Utóbbira példa:

(1) -5.0 Dcyl 130*=0.05

(2) 0,01 +2,0Dsph-6,0Dcyl 130 stp=0,05

(3) V:1mou +2.0Dsph -5.0Dcyl 130=0.05

Ezek a tokenek mind egy nagyobb egységnek, a visusnak (’látásélesség’) a részei.

8 Köszönetnyilvánítás

Szeretném megköszönni Orosz Györgynek és Siklósi Borbálának, a Pázmány Péter Katolikus Egyetem Információs Technológiai Kar PhD-hallgatóinak értékes tanácsaikat és segítőkészségüket. Köszönöm édesanyámnak, Ludányiné Csivincsik Máriának a szakdolgozat megszerkesztésében nyújtott segítségét. Köszönet illeti barátaimat, Csikós Gábort és Oszlányi Liliánát a dolgozat stilisztikai véleményezéséért, valamint az angol nyelvű összefoglaló elkészítésében való közreműködésért.

Irodalomjegyzék

[1] Siklósi Borbála, Orosz György, Novák Attila: Magyar nyelvű klinikai dokumentumok előfeldolgozása. In: Tanács Attila, Vincze Veronika szerk. *VIII. Magyar Számítógépes Nyelvészeti Konferencia.* Szeged, 2011. december 1–2. pp. 143–152. 2011. Elektronikus dokumentum. http://www.inf.u-szeged.hu/mszny2011/images/stories/kepek/mszny2011_press_nc_b5.pdf [2012.05.02.]

[2] Németh Géza és Bartalis Mátyás közlése. [2012.04.24.]

[3] The Association for Computational Linguistics. What is Computational Linguistics? http://www.aclweb.org/archive/misc/what.html [2012.05.02.]

[4] Tikk Domonkos szerk. *Szövegbányászat*. Typotex, Budapest, 2007. 300 p. ISBN 978-963-9664-45-6

[5] Carol Friedman, George Hripcsak: Natural Language Processing and Its Future in Medicine. *Academic Medicine*, 74: pp. 890–895. 1999. http://www.columbia.edu/itc/hs/medinfo/g6080/misc/articles/friedman.pdf [2012.05.02.]

[6] Pátrovics Péter. A medicina nyelve (szaknyelvi sajátságok, közlési, tájékoztatási módszerek az orvos-beteg kapcsolatban). *Magyar Orvosi Nyelv,* 4 (1): 20–24. 2004.

[7] Muzsik Gyula, Somogyi Péter: Logikán és mintaillesztésen alapuló tudásreprezentáció egy fejlődésneurológiai szakértő rendszerben. In: Gábor András szerk. *Szakértő rendszerek '88. Ismeretalapú információfeldolgozás Magyarországon.* Számalk, 1988. 500 p. ISBN 963-553-136-2

[8] Daniel Jurafsky, James H. Martin. *Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition.* Pearson Prentice Hall, 2009. 988 p. ISBN 0130950696

[9] Chris Manning, Hinrich Schütze. *Foundations of Statistical Natural Language Processing*, MIT Press. Cambridge, 1999. 620 p. ISBN 0262133601, 9780262133609

[10] Andrei Mikheev: Periods, capitalized words, etc. *Journal Computational Linguistics*, Volume 28 Issue 3, September 2002. Elektronikus változat: http://acl.ldc.upenn.edu/J/J02/J02-3002.pdf [2012.05.02.]

[11] Jeffrey C. Reynar, Adwait Ratnaparkhi: A maximum entropy approach to identifying sentence boundaries. In: *Proceedings of the Fifth Conference on Applied Natural Language Processing*, Washington, DC, USA, March–April 1997. Elektronikus változat: http://www.aclweb.org/anthology-new/A/A97/A97-1004.pdf [2012.05.06.]

[12] Gregory Grefensette, Pasi Tapanainen: What is a word, what is a sentence? Problems of tokenization. In: *The Proceedings of the Third Conference on Computational Lexicography and Text Research,* Budapest, Hungary, 1994. Elektronikus változat: http://www.coli.uni-saarland.de/~schulte/Teaching/ESSLLI-06/Referenzen/Tokenisation/grefenstette-tapanainen-1994.pdf [2012.05.02.]

[13] Zsibrita János, Nagy István, Farkas Richárd: Magyar nyelvi elemző modulok az UIMA keretrendszerhez. In: Tanács Attila, Szauter Dóra, Vincze Veronika szerk. *VI. Magyar Számítógépes Nyelvészeti Konferencia.* Szeged, 2009. december 3–4. pp. 394–395. 2009. Elektronikus dokumentum. http://www.inf.u-szeged.hu/projectdirs/mszny2009/MSZNY2009_press_b5_mod_opt.pdf [2012.05.03.]

[14] A Huntoken dokumentációja. Letölthető: http://mokk.bme.hu/resources/huntoken/ [2012.05.06.]

[15] Indig Balázs írásbeli közlése. [2012.03.13.]

[16] Németh Géza, Olaszy Gábor szerk. *A magyar beszéd.* Akadémiai Kiadó, Budapest, 2011. 708 p. ISBN: 9789630589666

[17] Olaszy Gábor, Németh Géza, Olaszi Péter, Kiss Géza, Gordos Géza: PROFIVOX – A Hungarian Professional TTS System for Telecommunications Applications. *International Journal of Speech Technology*, Vol. 3, Num. 3/4, December 2000. pp. 201–215.

9 Függelék

9.1 Rövidítések listája

abl
ac
adapt
adav
add
aggr
al
alk
all
ált
an
anamn
ant
appl
aps
ar
átl
aug
ax
axc
bb
bbi
beig
bet
bgy
bifó
bl
bleb
b(\.)?o
bscan
bss
bts
bulb
ca
cac
cacg
cai
cam
caps
card
cas
cat
cc
cct
cd
centr
cff
chr
ckp
cmo
cn
cnv
co
coff
cong
(sub)?conj
cons
cont
conv
copd
corn
corp
corr
cort
cotr
cpc
cs
crp
crt
cyl
csix
csmo
csüt
csv
csvi
csvii
da
dalk
dcr
dcyl
dd
d(\.)e
dec
deg
desc
dex
dexa
dg
di
diab
diag
dig
dispo
d(\.)?j
dm
doc
dok
dptr
dr
drn
dsaek
dsph
du
ecce
ectr

ednyt
effl
eion
éj(sz)?
elf
ell
elm
eog
e(\.)?sz
eü
exc(av)?
exo
exp(l)?
exst
ext
fac
faz
fe
fé
febr
fén
fert
fiz
fl
flag
foll
fo(\.)d
foly
folyt
foto
fsz
fszt
ft
fu
fv
fvs
gdx
gel
gest
get
gfr
ggt
g(\.)k
gl
gld
got
gp
gpt
gr
grad
graft
gzs
gyak
gysz
gyull
hba
hd
hegs
hep
hez
hg
hg(m)+
hh
hps
ht
huf
hum
hy
hyp
ict
id
iddm
ig
igt
ill
im
impl
(incip|inicp)
inf
inj
inr
iol
iop
ir
irreg
ism
iszb
it
itn
i(\.)v
jabb
jan
jav
jia
j(\.)l
j(\.)?o
jsiv
(júl|jul)
(jún|jun)
juv
k
kat
kb
kcl
kcs
kez
kgy
kh
kif
kkl
kl
kml
ko
komb
konv
konz
korr
kp
kr
kth
l
lat
l(\.)?d(ex)?

ldh
ldm
lev
limb
lkp
lmm
l(\.)?o
loc
lok
l(\.)p
l(\.)?s
lsd
l(\.)?s(in)?
lss
l(\.)?(u((t)?(r)?)?|ú)
mac
maj
mal
márc
marg
max
may
md
me
med
megf
mel
mell
mérs
mg
min
mj
m(\.)?k(ét)?
mk(l|ö)?o
mksz
mkúj
mm(es)?
mns
mo
monoc
mou

mri
ms(ec)
mtx
my
mydr
myop
nas
nasal
nat
neg
nega
negal
neos
nfh
niddm
nj
norm
nov
ns
ntg
nucl
nycs
nyh
o
o(\.)?a
oad
obes
obs
oc
occl
oct
octn
(St(\.)|cs(\.))?o(\.)?(
\s)?d(ex)?
okt
ol
olv
on
(meg)?op
opc
or

orfi
(St(\.)|cs(\.))?o(\.)?(
\s)?s(in)?
osx
ot
(St(\.)|cs(\.))?o(\.)?(
\s)?u(tr)?
pa
palp
párh
pas
pcl(es)?
pcp
pctl
pd
pdt
p(\.)?e
ped
perf
perif
pex
pg
pgorg
ph
pic
pk
pkp
pl
pm
pmma
pné
p(\.)?o
poag
postop
posü
poz
pp
ppv
pr
praes
prgr

prof
prog
p(\.)?s
psd
psx
pt
pup
ra
ram
rapd
rb
rd
rec
reg
rel
repdt
repkp
repr
részl
ret
rez
rf
rgp
ri
rl
rm
rnfl
roe
rp
rpd
rpe
rr
rt
rtg
rtvue
rv
sae
s(\.)?c
scepp
sch
scler
sdr
segm
ses
sf
shbg
si
sign
sil
sim
sinf
sk
skia
sle
sm
sp
spgh
sph
sply
srd
st(\.)?
starb
step
stf
stl?p?l?y
stz
subcaps
suff
suh
susp
s(\.)?ü
sv
sy
synd
sz
sza
szau
szcs
sz(\.)?e
sz\.?e\.?o.\?u
szept
szf
szg
szl
szpt
szs
sz(\.)sz
sszt
szt
szur
szü
szű
szv
ta
(film|l)?tabl
tap
tapp(l)?
tars
tbc
tbl
tc
tca
td
tdig
tel
tem
temp
t(h)?(e)?(r)?
thj
tln
t(\.)?m
to
tod
topg2
topo
tp
tpk
tsh
tt
ttonop
ttp
tts
tu

```
tul
tünj
tynd
u
u(\.)?a
ubm
uc
uh
uk
um
up
utr
ünf
ünj
üö
üt
v
va
val
vasc
vc
vd
vé(l)?
vegf
vep
vvfény
vh
(intra)?vitr
vizsg
vk
vkl
vkml
vkö(z)?
vlt
vm
vmi
vo(\.)?d
vol
volk
vsü
vsük
vsz
vt
vue
vuln
vvf
vvt
vzs
xal
xc
xjrs
xtm
ya(a)?g
za
zcb
zj
```

9.2 Felhasználói dokumentáció

A készített program célja, hogy mondatokra és szavakra bontást végezzen tetszőlegesen választott magyar nyelvű dokumentumhalmazon. A dokumentumok egyszerű, UTF-8-as kódolású szöveges fájlok lehetnek. A szoftver kifejezetten orvosi nyelvi szövegek feldolgozására készült, de futtatható tetszőleges tartalmú szövegen is (ez esetben várhatóan kevésbé jó eredménnyel).

A kisebb-nagyobb egységekre bontáson kívül az alkalmazás képes arra, hogy megjelölje a szövegben fellelhető rövidítéseket, valamint egy azonosítószámmal jelzi az egy rövidítéscsaládhoz tartozó elemeket. Egy rövidítéscsalád alatt a különböző, de hasonló alakú, ugyanolyan jelentésű rövidítéseket kell érteni.

A készített alkalmazás futtatásához az alábbi hardverkonfiguráció ajánlott:

– legalább 512 MB RAM (1 GB ajánlott),
– Pentium 4-es processzor, legalább 1 GHz,

– ha nincs még telepítve a Python interpreter: legalább 40 GB-nyi szabad hely a merevlemezen.

A program tetszőleges platformon (Windows, Linux, Unix, MAC OS) futtatható. Windows operációs rendszer esetén legalább egy XP SP3 verzió szükséges.

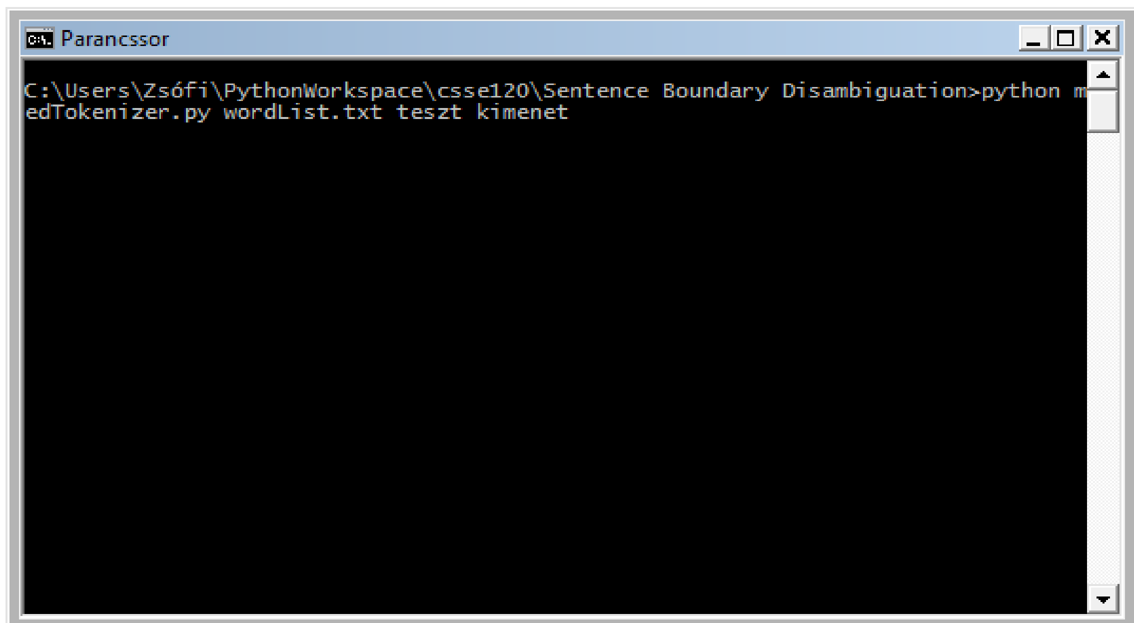
A futtatáshoz elengedhetetlen a legalább 2.7-es verziójú Python interpreter megléte. Ennek telepítéséről bővebb információt a Python hivatalos weboldalán találhatunk.[6]

A program telepítéséhez egyszerűen másoljuk a fájlokat a kívánt könyvtárba. Fontos, hogy a programhoz tartozó txt-fájlok a lists könyvtárban maradjanak.

Futtatáshoz indítsunk el egy parancssort, és gépeljük be az alábbiakat (8.1-es ábra):

```
python medTokenizer.py <rövidítéslista> <bemenet> <kimenet>
```

Rövidítéslistaként ajánlott a wordList.txt nevű szövegfájl megadása, de tetszőleges rövidítéslista megadható. Fontos, hogy bemenetként és kimenetként egy-egy könyvtár nevét adjuk meg, nem pedig fájlnevet. Be- és kimenetként ugyanaz a könyvtár is megadható, ez esetben az eredeti fájlok elvesznek, ezért ez a beállítás nem ajánlott.

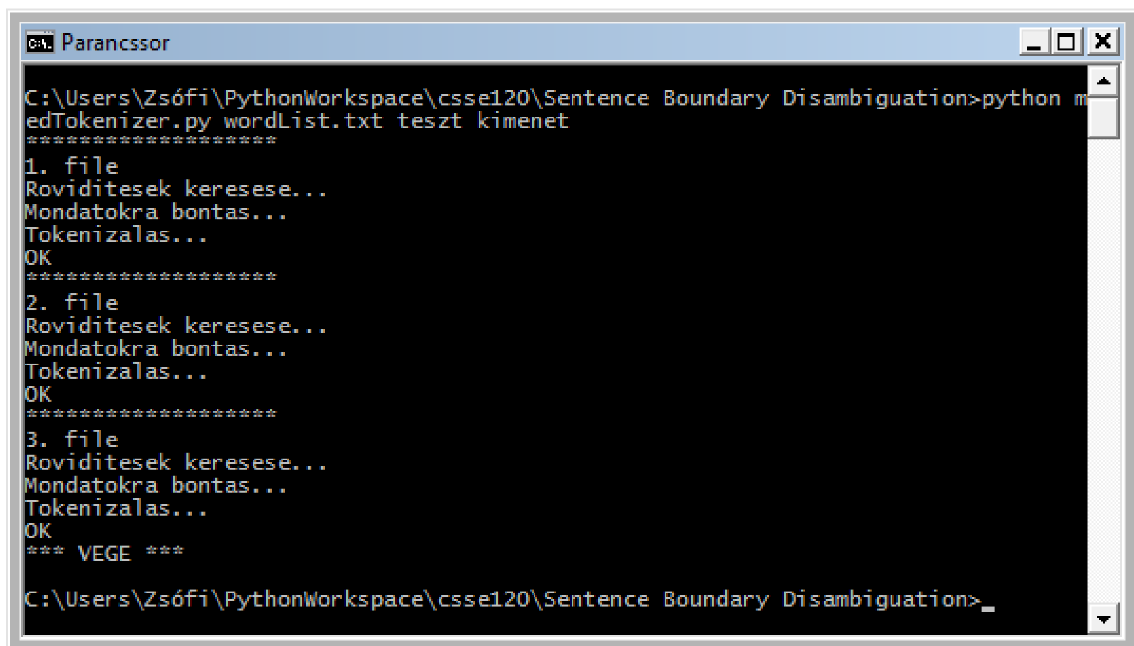


9.1. ábra. A medTokenizer indítása parancssorból

[6] http://www.python.org

A parancs begépelése és az Enter megnyomása után elindul a szövegfeldolgozás. A képernyőn nyomon követhetjük, hogy hányadik fájlnál tartunk éppen, és melyik részfeladatot hajtja éppen végre a program (mondatokra bontás, rövidítések keresése…). Ha az összes fájl feldolgozása elkészült, a program kijelzi ezt, és leáll. A kimenetként megadott könyvtárba belépve böngészhetünk a feldolgozott szövegek között.

A 9.2-es ábrán láthatjuk a programot működés közben.



9.2. ábra. A medTokenizer működés közben

9.3 Fejlesztői dokumentáció

A szoftver fejlesztéséhez az alábbi eszközöket használtuk fel:

– Eclipse Indigo Service Release 1 fejlesztőkörnyezet PyDev modullal,
– Python 2.7 interpreter.

A fejlesztés során a részfeladatok implementációit – alkalmazkodva az Eclipse fejlesztőkörnyezet konvencióihoz – egy projekt keretében valósítottuk meg. A projekt

több Python-modult tartalmaz: magát a tokenizáló (medTokenizer.py), a szabályokat (Rules.py), illetve a kiértékelést végző modult (compare.py). Ez utóbbi nem kapcsolódik szorosan a projekthez, csupán a tokenizáló kimenetként előállt fájlok és az elvárt eredmény összehasonlítását végzi el.

A medTokenizer.py forrásfájl függvényeit a 9.1. táblázatban láthatjuk.

Művelet neve	**Argumentumok**	**Visszatérési érték típusa**	**Leírás**
`readFile()`	string: filePath	unicode	Beolvassa a paraméterben megadott szövegfájlt egy stringbe, amelyet unicode típusú adattá alakít (ez a visszatérési értéke is).
`readRegex()`	string: filePath	list	Reguláris kifejezéseket tartalmazó szövegfájl beolvasására használjuk. Visszatérési értéke egy lista, amelyben a reguláris kifejezések vannak eltárolva.
`normalize()`	string: textFileContent	string	Eltávolítja a redundáns szóközöket a szövegből.
`tagAbbr()`	string: textFileContent, list: regContents	string	Megkeresi a rövidítéseket szövegben, és ideiglenes címkével megjelöli azokat.
`tagDate()`	string: textFileContent	string	Megkeresi a dátumok szövegben, és ideiglenes címkével megjelöli azokat.
`tagTime()`	string: textFileContent	string	Megkeresi az időpontokat a szövegben, és ideiglenes címkével megjelöli azokat.
`tagDoctor()`	string: textFileContent	string	Megkeresi a vezetékneveket (orvosokét) a szövegben, és ideiglenes címkével megjelöli azokat.
`tagMedicine()`	string: textFileContent	string	Megkeresi a gyógyszerneveket a szövegben, és ideiglenes címkével megjelöli azokat.
`checkAbbr()`	string: textFileContent	string	Sortörést szúr be azok után a pontok után, amelyek rövidítés után következnek, és nagybetűs szó követi őket.
`segment()`	string: textFileContent	string	Sortörést szúr be minden mondatvégi írásjel után.
`newLineCorr()`	string: textFileContent	string	Kettőnél több sortörés esetén törli a felesleges üres sorokat.
`tokenize()`	string: textFileContent	string	Sortörést szúr be a tokenhatárok után.
`punct()`	string: textFileContent	string	Sortörést szúr be a nem mondatvéget jelölő írásjelek után. Az írásjelet megjelöli a végleges címkével.

Művelet neve	Argumentumok	Visszatérési érték típusa	Leírás
`addID()`	string: textFileContent, list: regContents	string	Végignézi a rövidítéslistát, és minden abban szereplő rövidítést ellát egy jelentésenként egyedi azonosítóval (az azonos jelentésű, de különböző alakú rövidítések ugyanazt az ID-t kapják).
`untagAbbr()`	string: textFileContent	string	Eltávolítja az ideiglenes rövidítéscímkéket a szövegből.
`untagDoctor()`	string: textFileContent	string	Eltávolítja az ideiglenes vezetéknévcímkéket a szövegből. Megjelöli a szót a végleges címkével.
`untagMedicine()`	string: textFileContent	string	Eltávolítja az ideiglenes gyógyszernévcímkéket a szövegből. Megjelöli a szót a végleges címkével.
`untagDate()`	string: textFileContent	string	Eltávolítja az ideiglenes dátumcímkéket a szövegből. Megjelöli a dátumot a végleges címkével.
`untagTime()`	string: textFileContent	string	Eltávolítja az ideiglenes időpontcímkéket a szövegből. Megjelöli az időpontot a végleges címkével.
`write()`	string: str, string: file	string	Fájlba írja az első paraméterként megadott karaktersorozatot a második paraméterben megadott néven.

9.1. táblázat. A medTokenizer.py függvényei