DIPLOMAMUNKA

Tóth Péter

Debrecen,

2008

Debreceni Egyetem

Informatikai Kar

INTERFÉSZ-PROGRAM KÉSZÍTÉSE INTERNETES KERESŐKHÖZ

Témavezető:

Dr. habil. Boda István

egyetemi docens

Készítette:

Tóth Péter

programtervező matematikus

Debrecen,

2008

INTERFÉSZ-PROGRAM KÉSZÍTÉSE INTERNETES KERESŐKHÖZ

Tartalomjegyzék

Bevezetés

Napjainkban (az információ korában) egyre nagyobb szerepet kap az Internet és annak talán egyik legnépszerűbb és legsokoldalúbb tartalmakat közzétevő szolgáltatása a Világháló (World Wide Web).

Mitől is olyan népszerű ez a szolgáltatás? A webkiszolgálókon tárolt hipertext formátumú weblapokon nem csak szöveges, hanem képi, hang- és video-információk is megoszthatók, valamint hasonló vagy kapcsolódó témájú oldalakra mutató hivatkozásokat (linkeket) helyezhetünk el, így bizonyos témájú ismeretek rendszerezetten jelenhetnek meg [1]. A kérdés, hogy amikor egy adott téma iránt érdeklődünk, honnan tudhatjuk, hogy mely lapokat érdemes elolvasnunk. Ezzel a kérdéssel a legtöbb ember internetes keresőhöz fordul.

Az internetes keresők speciális webhelyek a Világhálón, amelyeket azért hoztak létre, hogy az emberek könnyebben rábukkanhassanak más webhelyeken tárolt információkra [2]. Napjainkban sok ilyen keresőrendszer létezik. Vannak, amelyek népszerűbbek, vannak, amelyek valamilyen szempontból speciálisabbak, de mindegyikük ugyanazon általános séma szerint működik, ellátja a következő három alapfeladatot:

- Kulcsszavak után keresnek az Interneten, vagy annak bizonyos részein.
- A talált szavakról és előfordulásaik helyéről indexet (szójegyzéket) készítenek.
- Lehetővé teszik, hogy a felhasználók az indexelt oldalakon szereplő szavakra, szókapcsolatokra keressenek. [3]

Ahhoz, hogy egy kereső egy állományban vagy dokumentumban keresni tudjon, meg kell tudni találnia az Interneten. A létező többszázmillió weblapon található információk felderítéséhez a keresők speciális robot-programot, ún. pókot (spider) alkalmaznak, melyek az oldalakon található szavakról listát készítenek. Általában a felderítés a legforgalmasabb kiszolgálókon és a legnépszerűbb weboldalakon kezdődik. A pókok egy-egy nagyon népszerű webhelyről indulnak, jegyzékbe veszik az oldalakon szereplő szavakat, és követik az azokon található összes hivatkozást. Ezzel az eljárással a keresőrendszer gyors terjeszkedésbe kezd a Világháló leglátogatottabb területein. [2]

A pókok által begyűjtött információkat a keresőnek hasznosítható módon kell letárolnia. Ezek a következő két fő összetevő segítségével válnak elérhetővé a felhasználók számára:

- a szavak és a hozzájuk tartozó járulékos információk,
- az információk indexelésére használt eljárás.

A legegyszerűbb esetben a kereső csak a szót tárolná és az URL-t, ahol az előfordult, azonban így nem tudnánk megmondani egy szóról, hogy az oldalon fontos vagy jelentéktelen helyen állt, tehát nem lenne lehetőség egy olyan ranglista (ranking list) előállításra, amely a leghasznosabb találatokat a kereső találati listájának elejére sorolná. Éppen ezért a legtöbb kereső tárolja, hogy az adott szó hányszor fordult elő a lapon, és súlyt rendel az egyes bejegyzésekhez. Egy szóhoz pl. nagyobb értéket rendel, ha a szó a dokumentumban előrébb helyezkedik el, valamint ha alcímekben, hivatkozásokban vagy éppen az oldal címében fordul elő. A különböző keresők más és más képlet alapján számítják ezt a súlyt, ezért tapasztalhatjuk, hogy ugyanarra a keresett szóra más-más találati listát állítanak elő.

Az indexelés célja, hogy a lehető leggyorsabbá tegye az információk fellelését. Az indexépítés egyik leghatékonyabb módja az ún. hasító tábla (hash table) készítése. Az ún. hasítás (hashing) során minden egyes szóhoz egy-egy számértéket rendelünk egy képlet alapján. Ezt a képletet úgy választjuk meg, hogy a bejegyzéseket egy előre meghatározott számú helyre egyenletesen ossza el. A hasító tábla hatékonyágának kulcsa, hogy a szavakhoz rendelt számok eloszlása különbözik az ábécébeli eloszlásuktól.

Az indexeken alapuló kereséshez a felhasználónak egy lekérdezést kell létrehoznia és el kell küldenie a keresőnek. A lekérdezés elég egyszerű is lehet, de legalább egy szóból kell állnia. Összetettebb keresések létrehozásához logikai (Boole-algebrai) és egyéb operátorok (AND, OR, NOT, FOLLOWED BY, NEAR és idézőjelek) használata szükséges, amelyekkel lehetőségünk van a keresésben szereplő kifejezések pontosítására és kiterjesztésére.

A logikai operátorok segítségével megadott keresés szó szerinti (literal search), azaz a kereső a megadott szavakat vagy szókapcsolatokat éppen a megadott alakban keresi [2]. Ez pedig magyar nyelvű szövegek esetén nyelvünk agglutináló, azaz ragozó volta miatt nem túl hatékony, mivel egy-egy névszónak és igének nagyon sok toldalékos alakja van, így a kereső csak a nagyon közeli vagy teljesen egyező előfordulásokat találja meg.

E dolgozat célja az előbbiekben említett probléma kiküszöbölési lehetőségének vizsgálata. Az alapötlet, hogy indexelés során ne magukat a szavakat, hanem a szavak szótövét tároljuk le, kereséskor pedig a keresett kifejezés szavainak az előbbivel azonos eljárással előállítjuk a szótövét, majd ezt keressük az indexben. Tehát gyakorlatilag az indexelés, illetve a keresés előtt a szavakon egy transzformációt hajtunk végre, amely által akár az indexelendő szöveg, akár a keresett kifejezés tartalmaz toldalékos szót, a kereső akkor is találattal tér vissza, ha a szavak nem pont azonos alakban szerepelnek a szövegben, de szótövük azonos.

A dolgozat további részében példákon keresztül bemutatom a program használatát és működését, a magyar nyelvi szövegek feldolgozása során felmerülő problémákat és azok lehetséges, illetve megvalósított megoldását, valamint a fentebb említett transzformációs módszer előnyeit és hátrányait.

A *WordAnalyzer* programról felhasználóknak...

Mint ahogyan már a bevezetésben rávilágítottam, e dolgozat célja egy olyan algoritmus megvalósítása, amelynek segítségével a keresőprogramok nem a szavak ténylegesen előforduló alakja alapján készítik az indexet, hanem a szótővük alapján keres, ezáltal hatékonyabbá és könnyebbé téve a magyar nyelvű szövegekben való keresést.

A program a *WordAnalyzer* (szóelemző) fantázianevet kapta, mivel legfőbb szolgáltatása, funkciója a szavak elemzése, különös tekintettel a szó tövének meghatározására. A későbbiekben látni fogjuk, hogy ez tulajdonképpen egy nagyon kezdetleges morfológiai elemző, ám ennek ellenére is elég nagy hatékonysággal használható szótőkeresésre. Továbbá az is ki fog derülni, hogy az általam kifejlesztett általános algoritmusnak köszönhetően ez az elemző továbbfejleszthető, valamint nem csak magyar, hanem más nyelvek esetén is használható.

Ebben fejezetben – a címe ellenére – nem csak a program működtetését mutatom be, hanem – a következő részt előkészítendő, – nagy vonalakban arról is áttekintést nyújtok, hogy az egyes műveletek során mi történik, hogyan működik a *WordAnalyzer*. A következő témákról lesz szó:

- a program telepítésének előfeltételei és a telepítés folyamata;
- a program futtatásának előfeltételei és indítása;
- a program használata:
    - keresési könyvtár beállítása,
    - keresési könyvtár dokumentumainak elemzése,
    - kifejezés keresése az indexekben,
    - keresett kifejezések felépítése.

A WordAnalyzer telepítése

A programot magát nem szükséges telepíteni, az közvetlenül futtatható a mellékelt CD lemezről, azonban a példaként használt HTML állományokat fel kell másolni írható-olvasható eszközre (pl. merevlemezre vagy pendrive-ra) egy tetszőleges könyvtárba.

A telepítés előfeltételei

A program telepítéséhez a következők szükségesek:

- olvasási jogosultság a forráskönyvtárakra (CD)
- írási, olvasási és futtatási jogosultság a célkönyvtárakra a célmeghajtón (merevlemez vagy pendrive)
- 2 MB szabad tárterület a célmeghajtón a program számára
- 20 MB szabad tárterület a célmeghajtón a példaállományok és az indexek számára

A telepítés lépései

A program telepítése az alábbi lépések végrehajtásával végezhető el:

1. Helyezze a CD lemezt a CD vagy DVD meghajtóba!
2. Linux operációs rendszer esetén csatlakoztassa a fájlrendszerhez (a `mount` parancs segítségével)!
3. Ha a programot is telepíteni kívánja, másolja a CD-n található `WordAnalyzer` könyvtárat tartalmával együtt a merevlemezre!
4. Másolja a CD-n található `SampleFiles` könyvtárat tartalmával együtt a merevlemezre!
5. Amennyiben a programot is telepítette, a futtatáshoz már nem lesz szüksége a CD lemezre, lecsatlakoztathatja (az `umount` parancs segítségével), kiveheti a meghajtóból.

A WordAnalyzer futtatása

A futtatás előfeltételei

A program futtatásához az alábbiakra van szükség:

- Java Runtime Environment 1.6.0-ás (vagy újabb) verziójú futtatókörnyezet,
- grafikus felhasználói felület (GUI),
- olvasási és futtatási jogosultság a `WordAnalyzer`, valamint olvasási, listázási és írási jog a `SampleFiles` könyvtárra.

A program indítása

A program az alábbi lépések végrehajtásával indítható:

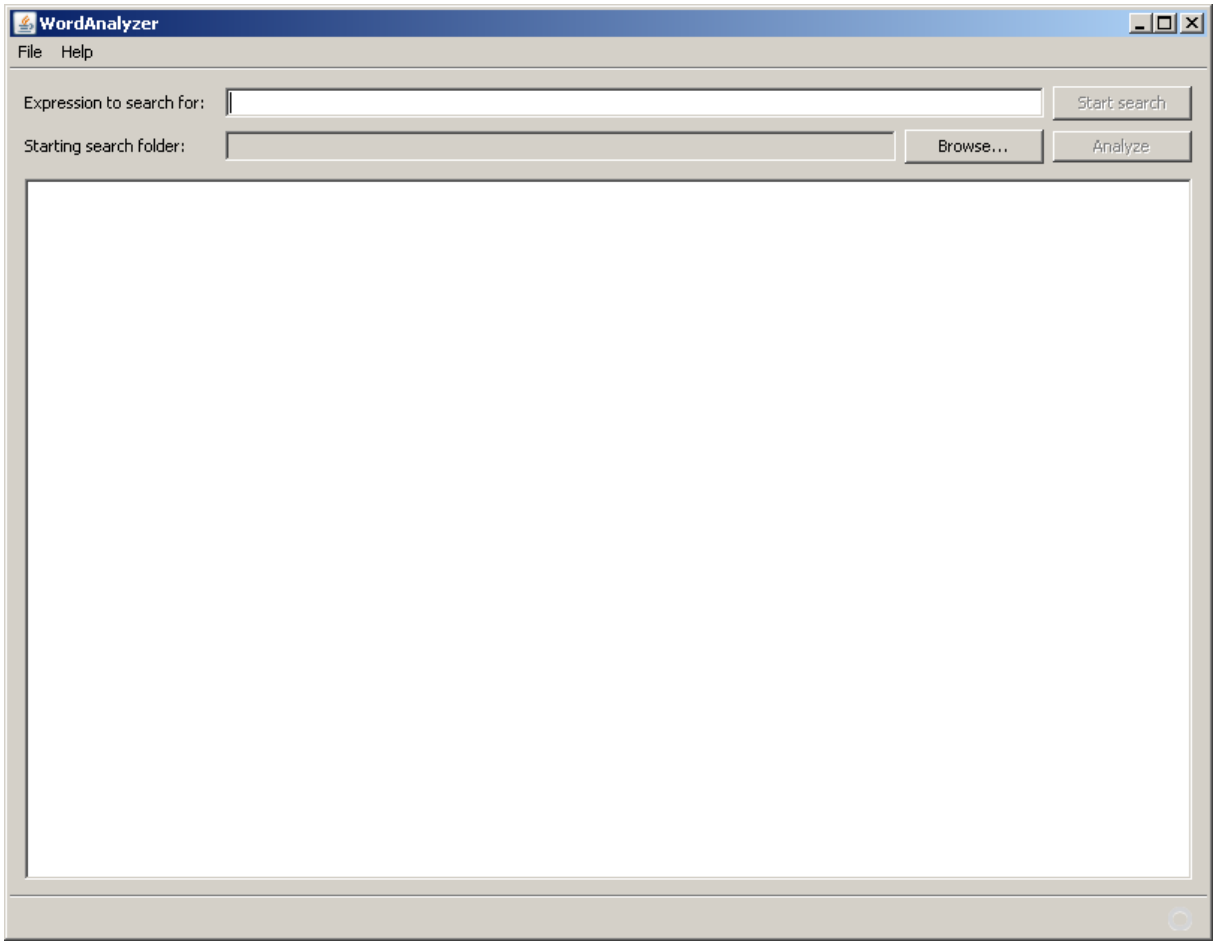
1. Nyisson egy konzolablakot!
2. Lépjen be a `WordAnalyzer` könyvtárba (pl. a `cd` parancs segítségével)!
3. Adja ki a következő parancsot: `java –jar WordAnalyzer.jar`

Amennyiben a `java` parancs nem található, a `set` parancs kiadása után bizonyosodjon meg róla, hogy a `PATH` környezeti változó tartalmazza a futtatókörnyezet `bin` alkönyvtárát. Ha nem, a következőhöz hasonló paranccsal adhatja hozzá:

```
set PATH=%PATH%;”C:\Program Files\Java\jre1.6.0_07\bin”
```

Ha minden rendben zajlik, a fenti `java` parancs kiadását követően, néhány másodpercnyi inicializálás után megjelenik a *WordAnalyzer* ablak (1. ábra). Ezen a felületen keresztül lehet elérni a program két fő szolgáltatását; a kiválasztott könyvtár tartalmának elemzését (indexelését) és az ebben történő keresést.

A grafikus felhasználói felület felépítése és kezelése teljesen szokványos, mind billentyűzettel, mind egér segítségével használható.



1. ábra

Az ablak három fő részből áll: a menüsorból, az elemzési és keresési részből, valamint az állapotsorból.

A menüsor *File* menüjéből érhető el a kilépésre szolgáló *Exit* parancs, valamint a *Help* menüből az *About...* (névjegy).

A menüsor alatt látható elemzési és keresési részről később részletesebben szót ejtünk a következő fejezetben.

Az ablak alján elhelyezkedő állapotsor a program aktuális állapotáról, az általa épp végzett műveletről, illetve a művelet befejezése vagy megszakítása után annak eredményéről szolgáltat információt.

A WordAnalyzer használata

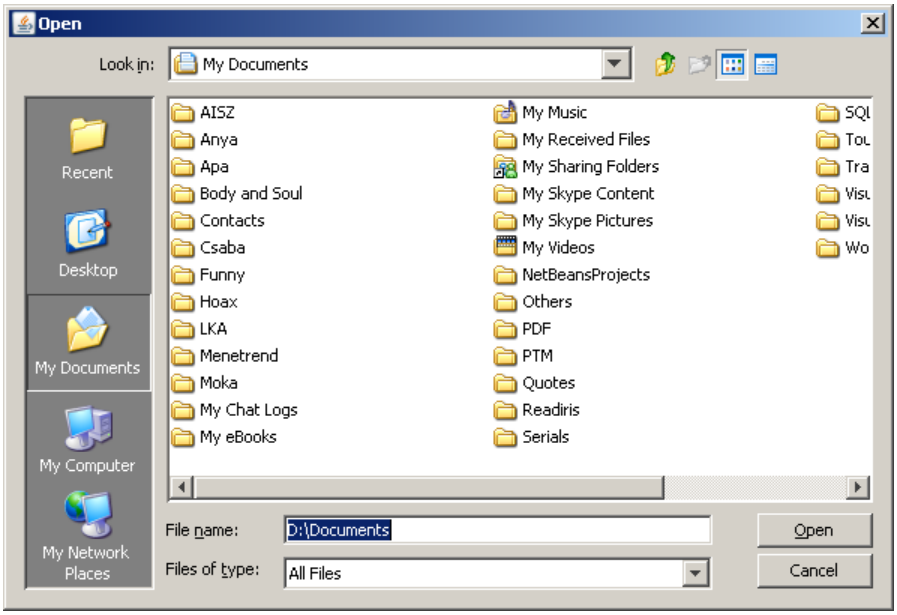
A program használata az alábbi három fő műveletet jelenti: a keresés tárgyát képező dokumentumokat tartalmazó könyvtár (továbbiakban: keresési könyvtár) kiválasztása, a kiválasztott könyvtárban és az azt tartalmazó alkönyvtárakban található összes szöveges állomány elemzése, indexelése valamint keresés a kiválasztott keresési könyvtárban.

Keresési könyvtár kiválasztása

A keresési könyvtár kiválasztása a következő lépésekkel történik:

1. Kattintson a *Browse...* (tallózás) gombra!
2. A megjelenő dialógusablakban (2. ábra) válassza ki a kívánt könyvtárat!
3. Ellenőrizze, hogy a *File name* (fájlnév) szövegdoboz értéke a kívánt könyvtár nevét tartalmazza a teljes elérési úttal együtt!
4. Kattintson az *Open* (megnyitás) gombra!

Ezzel a keresési könyvtár neve az elérési úttal együtt bekerült a *Starting search folder* (kezdő keresési könyvtár) mezőbe. Ezek után lesz lehetőségünk elemezni a könyvtár tartalmát az *Analyze* (elemzés) gomb megnyomásával, valamint keresni az elemzett állományok között a keresett kifejezés megadása után, a *Start search* (keresés indítása) gombra kattintva.



2. ábra

Könyvtár tartalmának elemzése, indexelés

Ahhoz, hogy a keresési könyvtáron belüli szöveges állományokban keresni tudjunk, előbb elemezni kell ezeket a dokumentumokat.

A dokumentumok elemzéséhez tegye a következőket:

1. Válassza ki a kívánt keresési könyvtárat, ha szükséges!
2. Kattintson az *Analyze* (elemzés) gombra! (Az indítás után a gomb felirata átváltozik *Stop*-ra (leállítás), erre kattintva bármikor megszakítható a művelet.)

Az elemzés során a program egyesével veszi a keresési könyvtáron belül található állományokat, kinyeri belőlük az értékes szavakat, ezeket elemzi, majd a kapott szótövek felhasználásával indexállományt hoz létre azokhoz. Az indexállományok neve megegyezik a dokumentumok eredeti nevével, kiterjesztésük pedig *.words*. Az aktuálisan feldolgozás alatt álló állomány neve az állapotsorban, a már feldolgozott állományok a listában jelennek meg.

Keresés

Elemzés után van lehetőségünk keresni a keresési könyvtáron belül elhelyezkedő összes szöveges állományban, illetve a hozzájuk létrehozott indexállományokban. A keresendő kifejezést a jól ismert módon, a bevezetésben már említett Boole-algebrai és egyéb operátorokkal lehet megadni. (A részletekért lásd a következő részt!)

A keresés a következő lépések végrehajtásával történik:

1. Válassza ki a kívánt keresési könyvtárat, ha szükséges!
2. Adja meg a keresett kifejezést az *Expression to search for* (keresendő kifejezés) szövegdobozban!
3. Kattintson a *Start search* (keresés indítása) gombra! (Az indítás után a gomb felirata átváltozik *Stop*-ra (leállítás), erre kattintva bármikor megszakítható a keresési művelet.)
4. A keresés során a listában megjelenik azon dokumentumok neve, amelyek tartalmazzák a keresett kifejezést. A találati lista egyes elemeire duplán kattintva megjelenik az alapértelmezett böngészőben az adott dokumentum, amelyben a megtalált kifejezés bíbor színű, félkövér betűkkel van kiemelve.

A keresett kifejezés felépítése

Mivel e dolgozat célja nem egy minden funkcióval ellátott keresőrendszer megvalósítása, hanem a szótöveken alapuló keresés hatékonyságának demonstrálása, a *WordAnalyzer* programban csak az „és”, valamint az „idézőjel” operátor használata került implementálásra.

Az „és” logikai operátor a *Google*, ill. más internetes keresőkhöz hasonlóan itt is a „+” jel. Ezek segítségével szavak, vagy „idézőjel” operátorok között megadott szókapcsolatok köthetők össze.

Az alábbiakban lássunk néhány példát keresett kifejezésekre, valamint azt, hogy milyen találatokat szolgáltathatnak:

- *„legnagyobb kövek”+zuhanjanak*
    - a *„nagy”* és a *„kő”* szótöveket közvetlenül egymás után, sorrendhelyesen és a *„zuhan”* szótövet tetszőleges helyen tartalmazó dokumentumok
    - konkrét példák:
      *„Egy **Nagy Kő Zuhan** és a szívemhez ér”*
      *„**Nagy kő zuhan**, átrepült a szívemen”*
- *kövek+repülnének*
    - a *„kő”* és a *„repül”* szótöveket tetszőleges sorrendben, egymáshoz képest tetszőleges helyen tartalmazó dokumentumok
    - konkrét példa:
      *„És **visszarepülni** vagy nagyon előre*
      *A szív a **kőre**, a szív a **kőre**”*
- *kövek*
    - a *„kő”* szótövet tartalmazó dokumentumok
    - konkrét példák:
      *„**Kövekkel** dobáltak, feküdtem a sárban”*
      *„Hogy lehetett szomjas az ordas a **kőtől**?”*
      *„Nem lesz a **kőből** virág!”*

A *WordAnalyzer* program működése

A felhasználói megközelítés után következzen a technikai jellegű bemutatás. Ebben a részben látni fogjuk, milyen bemeneti adatokat kap a program, és ezeket milyen lépések segítségével alakítja át kimeneti adatokká.

A program bemenete

A *WordAnalyzer* az elemzéshez egy ún. nyelvi modellt használ, amely az alábbi komponensekből tevődik össze:

- ábécé,
- a szótövek listája, és
- a toldalékok (a képzők, a jelek és a ragok) listája.

A program az elemzés során a fenti információk felhasználásával keresi meg a – szintén bemenetként szolgáló – HTML dokumentumokban található szavak tövét.

A HTML állományok elemzésével szemben a keresés során a billentyűzetről bevitt keresendő kifejezés kerül elemzésre, ebből következik, hogy a kereséshez szintén szükségesek a már fent említett nyelvi információk. Valamint bemenetként szolgálnak az előzetesen elemzett HTML állományokhoz készített indexállományok is.

Összefoglalva tehát a *WordAnalyzer* programnak az alábbi bemenetekre van szüksége

- az elemézéshez:
    - a nyelvi modell,
    - az elemzendő HTML állományok.
- a kereséshez:
    - a nyelvi modell,
    - a keresett kifejezés,
    - az elemzett HTML állományokhoz készített indexállományok, és
    - a találati dokumentumok létrehozásához az eredeti HTML állományok.

Ábécé

A nyelvi modell legalapvetőbb összetevője az ábécé, azaz a nyelvet felépítő elemi egységek, a nyelv betűinek halmaza. A *WordAnalyzer* az ábécét a `WordAnalyzer.jar` mellett található `alphabet.csv` állományból olvassa be. Mint ennek kiterjesztése is mutatja, az egyes értékeket (jelen esetben az egyes betűket) vesszővel elválasztva tartalmazza.

Az `alphabet.csv` állomány felépítése a következő:

- 1. sor: az ábécé összes betűje felsorolva ábécérendben
- 2. sor: a nyelv rövid magánhangzói felsorolva ábécérendben
- 3. sor: a 2. sorban megadott rövid magánhangzók hosszú párjai felsorolva ábécérendben
- 4. sor: a nyelv rövid mássalhangzói felsorolva ábécérendben
- 5. sor: a 4. sorban megadott rövid mássalhangzók hosszú párjai felsorolva ábécérendben

Szótövek

A modell másik komponense a nyelv legkisebb jelentéssel bíró egységeinek, a nyelv szótöveinek listája. A program az elemzés során az elemzendő szavakat ezen lista elemeire képezi le, illetve próbálja leképezni, az indexbe a listaelem sorszámai kerülnek be. Az előbbiekből következően a keresés finomsága, pontossága a listában megadott szótövek milyenségétől és számától is függ.

A szótövek listáját a *WordAnalyzer* a `words.txt` szöveges állományból olvassa be. Az alapértelmezett állomány – némi kiegészítéssel – a *Magyar értelmező kéziszótár* hetedik kiadásának [4] szóelemeit tartalmazza, biztosítva ezzel, hogy a program a szavak tényleges szótövét találja meg, így a különböző szótövű szavakat különböző tövekre képezze le.

Toldalékok

A szavak nem csak szótőként, hanem toldalékolt (képzett, jelzett és/vagy ragozott) alakban is előfordulnak nyelvünkben. Mivel a program legfőbb célja a szavak tövének meghatározása, ezért nyilvánvalóan szükséges a nyelvben használatos toldalékok listájának megadása.

A toldalékokat kapcsolódási helyük szerint két fő csoportra oszthatjuk: a szótő elé kapcsolódó prefixumokra és a szótő mögé kapcsolódó posztfixumokra. Az előbbi csoportba tartozó toldalékok listáját és a toldalékokra vonatkozó egyéb információkat a `prefixes.csv`, az utóbbiakat a `postfixes.csv` állományból olvassa be a program.

A `prefixes.csv` és a `postfixes.csv` állományok egy-egy sora ad meg egy-egy toldalékot. Ahogyan az ábécét leíró állomány, ezek az állományok is vesszővel elválasztott értékeket tartalmaznak, azonban szerkezetük jóval bonyolultabb.

Az egyes sorok szintaktikája a következő:

```
<toldalék>,<szófaj_toldalékkal>,<szófaj_toldalék_nélkül>,<szám>,
<személy>,<birtok_szám>,<mód>,<idő>,<ragozás>[,<művelet>]...
```

Az egyes címkék jelentése:

- `<toldalék>`: a toldalékot megadó kisbetűs karaktersorozat
- `<szófaj_toldalékkal>`: a toldalékkal ellátott szó szófajának azonosítója; lehetséges értékei:
    - 0 = nem definiált
    - 1 = ige
    - 2 = névszó vagy egyéb
- `<szófaj_toldalék_nélkül>`: ezzel a toldalékkal ellátott szó szófajának azonosítója a toldalék eltávolítása után
  lehetéges értékeit lásd: `<szófaj_toldalékkal>`
- `<szám>`: ige esetén a cselekvő számának azonosítója
  lehetséges értékei:
    - 0 = nem értelmezett/nem definiált
    - 1 = egyes szám
    - 2 = többes szám

- `<személy>`: ige esetén a cselekvő személyének azonosítója
  lehetséges értékei:
  - 0 = nem értelmezett/nem definiált
  - 1 = első személy
  - 2 = második személy
  - 3 = harmadik személy
- `<birtok_szám>`: névszó esetén a birtok számának azonosítója
  lehetséges értékei:
  - 0 = nem értelmezett/nem definiált
  - 1 = egyes számú birtok
  - 2 = többes számú birtok
- `<mód>`: ige esetén a cselekvés módjának azonosítója; lehetséges értékei:
  - 0 = nem értelmezett/nem definiált
  - 1 = kijelentő mód
  - 2 = felszólító mód
  - 3 = feltételes mód
- `<idő>`: ige esetén a cselekvés idejének azonosítója; lehetséges értékei:
  - 0 = nem értelmezett/nem definiált
  - 1 = jelen idő
  - 2 = múlt idő
- `<ragozás>`: ige esetén az ige ragozásának azonosítója
  lehetséges értékei:
  - 0 = nem értelmezett/nem definiált
  - 1 = alanyi ragozás
  - 2 = tárgyas ragozás (második személyre utaló alak)
  - 3 = tárgyas ragozás (harmadik személyre utaló alak)

- `<művelet>`: a toldalékkal, illetve a maradványszóval végzendő művelet azonosítója

  lehetséges értékei:

  - 4 = toldalék eltávolítása a szó végéről
  - 5 = szóvégi ’v’ eltávolítása
  - 6 = szóvégi ’a’ eltávolítása
  - 7 = szóvégi ’s’ eltávolítása
  - 30 = ’ik’ hozzáfűzése a szó végéhez
  - 31 = ’ó’ hozzáfűzése a szó végéhez
  - 32 = ’t’ hozzáfűzése a szó végéhez
  - 33 = ’sz’ hozzáfűzése a szó végéhez
  - 34 = ’ed’ hozzáfűzése a szó végéhez
  - 35 = ’od’ hozzáfűzése a szó végéhez
  - 36 = ’ud’ hozzáfűzése a szó végéhez
  - 37 = ’üd’ hozzáfűzése a szó végéhez
  - 38 = ’esz’ hozzáfűzése a szó végéhez
  - 39 = ’usz’ hozzáfűzése a szó végéhez
  - 40 = ’üsz’ hozzáfűzése a szó végéhez
  - 41 = ’a’ hozzáfűzése a szó végéhez
  - 42 = ’ta’ hozzáfűzése a szó végéhez
  - 71 = szóeleji ’leg’ eltávolítása
  - 100 = szóvégi mássalhangzó rövidítése
  - 101 = szóvégi mássalhangzó hosszabbítása
  - 102 = szóvégi utolsó két mássalhangzó felcserélése
  - 103 = szóvégi magánhangzó rövidítése
  - 104 = szóvégi magánhangzó hosszabbítása

- 105 = a szó utolsó magánhangzójának rövidítése
- 106 = a szó utolsó magánhangzójának hosszabbítása
- 107 = magánhangzó beszúrása hangzóhiányos szóba a szó utolsó két mássalhangzója közé
- 122 = ha a kapott szó szótő, akkor vége, különben következő művelet végrehajtása
- 123 = ha a kapott szó szótő vagy összetett szó, akkor vége, különben következő művelet végrehajtása
- 124 = ha a kapott szó szótő, akkor vége, különben visszatérés a szó toldalékmentes alakjához és a következő művelet végrehajtása
- 125 = ha a kapott szó szótő vagy összetett szó, akkor vége, különben visszatérés a szó toldalékmentes alakjához és a következő művelet végrehajtása
- 126 = ha a kapott szó szótő, akkor vége, különben visszatérés a szó toldalékos alakjához és a következő alkalmazható toldalék keresése
- 127 = ha a kapott szó szótő vagy összetett szó, akkor vége, különben visszatérés a szó toldalékos alakjához és a következő alkalmazható toldalék keresése

A program aktuális változata a toldalékok fenti jellemzőit tartja nyilván, valamint a felsorolt műveleteket valósítja meg, de természetesen ez igény szerint bővíthető, így más nyelvek szavainak elemzésére is alkalmassá tehető. A dolgozat fő célkitűzése a magyar nyelv sajátosságait szem előtt tartó keresőprogram kivitelezhetőségének vizsgálata, ám úgy tűnik, hogy ez a módszer a fenti nyelvi modell általánosításával gyakorlatilag akár még nyelvfüggetlenné is válhat. Az alapértelmezett `postfixes.csv` állomány az alábbi irodalmak segítségével készült: [4], [5], [6], [7] és [8].

Az elemzendő HTML dokumentumok

A nyelvi modellt leíró (`alphabet.csv`, `words.txt`, `prefixes.csv` és `postfixes.csv`) állományokon túl az elemzés bemenetét képezik még maguk az elemzendő dokumentumok.

A *WordAnalyzer* a megadott keresési könyvtáron belül található összes állományt megpróbálja elemezni, kivéve a `.words` és a `.result.html` kiterjesztésűeket. A `SampleFiles` könyvtárban található, példaként használt dokumentumokat *GNU Wget* segítségével töltöttem le (a www.zeneszoveg.hu oldalról), szimulálva ezzel a – bevezetésben már említett – pókok felderítő munkáját.

A keresett kifejezés

A keresés egyik bemenete maga a keresett kifejezés. Mint azt már láthattuk *A WordAnalyzer használata* c. fejezetben, a program billentyűzetről várja a keresett kifejezést. Ennek formátumát már megismertük *A keresett kifejezés felépítése* c. részben.

Az elemzett HTML állományokhoz készített indexállományok

A kereséshez szükség van azokra az indexekre, amelyeket a HTML állományok elemzése során állít elő a *WordAnalyzer*. Ezek az indexek *.words* kiterjesztéssel vannak megkülönböztetve az eredeti dokumentumoktól, és azokkal azonos könyvtárban helyezkednek el.

Az indexállományok bináris állományok, egy *Vector* típusú objektum szerializált változatai. Ez a vektor adatszerkezet tulajdonképpen az eredeti dokumentum lenyomata, egész számokat, illetve további vektor típusú objektumokat tartalmaz. Az egész számok az eredeti dokumentumból nyert szavak szótövének indexei, a vektorok elemei pedig összetett szavak esetén a tagszavak szótövének indexei.

A keresési könyvtár dokumentumainak elemzése

Ebben a részben áttekintjük, milyen lépések végrehajtásával állítja elő a *WordAnalyzer* a keresési könyvtárban található dokumentumokhoz az indexállományokat.

Az elemzési algoritmus főbb lépései a következők:

- a keresési könyvtár állományainak felderítése,
- a dokumentumok szavakra bontása,
- a szavak elemzése, tövének keresése,
- a szótövek indexeinek mentése az indexállományokba.

A keresési könyvtár állományainak felderítése

Az elemzési folyamat első lépéseként a *getFilesRecursively* rekurzív metódus összegyűjti a megadott keresési könyvtáron belül található összes állomány listáját. Ezután a lista elemeit sorra veszi, és a következő három részben ismertetett módon feldolgozza az előző keresések eredményeként előálló *.result.html* kiterjesztésű átmeneti állományok, illetve az előző elemzések során létrehozott *.words* kiterjesztésű indexállományok kivételével az összes állományt.

Szavak kinyerése a dokumentumból

Az első nehézség, amivel egy magyar nyelvű szöveges állomány feldolgozásakor találkozunk, a karakterkódolás problémája. Nyelvünk ábécéje összesen kilenc ékezetes betűt tartalmaz, ezeknek mind a kis (*á, é, í, ó, ö, ő, ú, ü, ű*), mind a nagy (*Á, É, Í, Ó, Ö, Ő, Ú, Ü, Ű*) változatát fel kell ismernie a programnak ahhoz, hogy a magyar szavakat elemezni, majd azonosítani tudja szótövük alapján.

A szöveges állomány feldolgozására hivatott *getWordsFromFile* metódus tehát első lépésben az állomány karakterkódolásáról próbál információhoz jutni a `charset` kulcsszó keresésével. Ha szerepel ez az attribútum a dokumentumban, kiolvassa az értékét, és amennyiben ez támogatott, ezt alkalmazza, ha nem, akkor alapértelmezésként *windows-1250* kódolással dolgozza fel az állományt.

A kódolás megállapítása után újra megnyitja a dokumentumot, egyesével olvassa be a sorait, eltávolítja belőlük a HTML jelölőket (HTML tag-eket) és a megjegyzéseket (comment-eket). Ennek eredményeként megmaradnak az oldalon látható szövegrészek, ezeket szavakra bontja a szövegtagoló (whitespace) karakterek és egyéb írásjelek (kivéve az *&* és a *;* karakterek) mentén. Az így előálló szavakat kisbetűsíti, bennük kicseréli az esetleges ún. escape szekvenciákat, illetve ISO kódokat a megfelelő ékezetes karakterekre. (A magyar ábécé betűinek kódjait lásd: http://mek.oszk.hu/01300/01306/html/16.htm)

Ezzel a program kinyerte a dokumentum szöveges részeiből a szavakat, ennek eredményeképp előáll egy karakterláncokat tartalmazó vektor, a szavak listája.

A kinyert szavak szótövének keresése

A következő lépésben a *getRootsFromVector* metódus az előbb előállított vektor szavainak szótövét keresi meg, majd az indexekből újabb vektor hoz létre. Ha a szó egyszerű

szó, egyetlen szótöve van, viszont ha összetett, akkor több is. Éppen ezért az újonnan létrehozandó vektor egyszerű szavak esetén a szótő indexét, összetett szavak esetén tagok szótöveinek indexvektorát tartalmazza. Magáról a szótőkeresés algoritmusáról, valamint annak lépéseiről később részletesen lesz szó az *A szótőkeresés algoritmusa* c. részben.

Indexállomány létrehozása a dokumentumhoz

Az elemzés végső lépése az indexállomány létrehozása, amely nem jelent mást, mint az előző lépésben előállított vektor szerializálását, azaz a vektorobjektum mentését a háttértárra bináris alakban. Az indexállomány a dokumentummal azonos könyvtárba kerül, az állomány neve is azonos lesz a dokumentuméval, csak egy *.words* kiterjesztést kap pluszban.

A szótőkeresés algoritmusa

A szótőkeresés algoritmusa (a *getRoots* metódus) bemenetként megkapja az elemezdő szót reprezentáló *LangWord* (nyelvtani szó) típusú objektumot, kimenetként pedig egy *LangWord* tömb típusú objektumot ad vissza, amely magában hordozza az elemzett szó tövét (töveit), valamint az elemzés során előálló, toldalékokból származó összes járulékos nyelvi információt. Ezen nyelvi információkról részletesen szóltunk az *A WordAnalyzer bemenete* fejezetének *Toldalékok* c. részében.

Legelső lépésben a kapott szót a kötőjelek mentén tagszavakra bontja, amennyiben tartalmaz kötőjelet. Ezután az egyes tagszavakból új *LangWord* típusú példányt hoz létre és átadja egy másik metódusnak, a *getRoot* rekurzív metódusnak.

A *getRoot* metódus tehát egy *LangWord* típusú szót kap bemenetként és szintén egy *LangWord* típusú objektummal tér vissza. Ha az algoritmus megtalálja a szó tövét, akkor a visszatérési érték a bemenetként kapott szó töve, beleértve a toldalékok eltávolításakor kinyert információkat is, amennyiben nem jár sikerrel az algoritmus, *null* értékkel tér vissza.

A rekurzív metódus legelőször megvizsgálja, hogy a bemeneti szó megtalálható-e a szótárban, azaz szótő-e. Ha igen, magával a bemeneti szóval tér vissza, ha nem, elkezdődik a szótő tényleges keresése.

A szótőkeresés során a *WordAnalyzer* a bemenetként kapott toldalékok listájának tagjain megy sorba és megvizsgálja, hogy az egyes toldalékokra vonatkozó műveletek alkalmazhatók-e, azaz

- a vizsgált szó az adott toldalékkal végződik, és
- a vizsgált toldalékos szó szófaja még nem definiált (a szó `<szófaj_toldalékkal>` tulajdonsága 0) vagy megegyezik a toldalékéval (azaz a szó és a toldalék `<szófaj_toldalékkal>` tulajdonsága azonos).

Amennyiben a toldalék megfelel a fenti két feltételnek, következik a hozzá tartozó műveleti kód vektorának feldolgozása, azaz az egyes műveletek végrehajtása a szón. A lehetséges műveleti kódokat már szintén láthattuk az *A WordAnalyzer bemenete* fejezetének *Toldalékok* c. részében.

Ha a toldalékhoz tartozó műveletek végrehajtása után a szó nem szótő, illetve nem összetett szó, a *getRoot* metódus rekurzívan meghívja önmagát a szó aktuális alakjával. Amennyiben ez *null* értékkel tér vissza, azaz ezen az ágon nem sikerült megtalálni a szó tövét, akkor a következő toldalékkal próbálkozik a *WordAnalyzer*, ha pedig igen, a rekurzív hívás eredményével tér vissza.

Ha a toldalékhoz tartozó műveletek végrehajtása után a szó szótő vagy összetett szó, akkor ezzel tér vissza.

Amennyiben egyetlen toldalék eltávolítása, műveleteinek végrehajtása sem vezet a szótő megtalálásához, a *getRoot* metódus *null* értékkel tér vissza jelezvén, hogy a keresett szó tövét nem sikerült felderíteni.

A külső *getRoots* metódus a kimeneti tömböt a rekurzív *getRoot* metódus visszatérési értékétől függően az alábbiak szerint hozza létre:

- ha a megtalált szó összetett, a tagjai külön kerülnek be a *LangWord* tömbbe;
- ha a megtalált szó egy szótő, akkor bekerül a *LangWord* tömbbe;
- ha a szó tövét nem sikerült megtalálni, akkor helyette üresszó fog szerepelni a tömbben.

Összefoglalásképpen a működés bemutatására következzen egy példa!

Tegyük fel, hogy egy szövegben az alábbi felszólító mondat szerepel: „Imádkozzunk legjobbjainkért!”. A *WordAnalyzer* ebből elsőként kinyeri a következő két szót: *imádkozzunk* és *legjobbjainkért*. Ezután a szavak bekerülnek egy *String* típusú vektorba, ennek elemeit egyenként dolgozza fel a program.

1. példa

Az *imádkozzunk* szó elemzése során a következő lépéseket teszi a *WordAnalyzer* program:

- a *getRoots* metódus paraméterként megkapja az *imádkozzunk* szót
- felbontja a – jelek mentén, ekkor kapunk egy egyelemű *String* tömböt az *imádkozzunk* szóval
- ebből az elemből létrehoz egy *LangWord* példányt nem definiált nyelvi jellemzőkkel, és átadja a *getRoot* metódusnak
- a *getRoot* metódus megvizsgálja, hogy szótő-e, mivel nem, elkezdi elemezni
- első alkalmazhatóként megtalálja az *–unk* toldalékot, végrehajtja a megadott műveleteket (4, 41, 123)
- eltávolítja az *–unk* toldalékot (4-es művelet), beállítja a hozzá tartozó nyelvi jellemzőket (szótő toldalékkal és toldalék nélkül: 2 [névszó vagy egyéb]; a többi jellemző: 0 [nem definiált]); *imádkozz*
- hozzáfűzi a szóvégéhez az *–a* toldalékot (41-es művelet); *imádkozza*
- megvizsgálja, hogy a szó szótő vagy összetett szó-e (123-as művelet), mivel nem, és ehhez a toldalékhoz tartozó műveletek elfogytak, az aktuális szóval rekurzívan meghívja (2. szint) önmagát
- megvizsgálja, hogy a szó szótő-e, azt találja, hogy nem, elkezdi elemezni
- első alkalmazhatóként megtalálja az *–a* toldalékot, végrehajtja a megadott műveleteket (4, 123)
- eltávolítja az *–a* toldalékot (4-es művelet), beállítja a hozzá tartozó nyelvi jellemzőket (szótő toldalékkal és toldalék nélkül: 2 [névszó vagy egyéb]; szám: 1 [egyes szám]; személy: 3 [harmadik személy]; birtok száma: 1 [egyes szám]; a többi jellemző: 0 [nem definiált]); *imádkozz*
- megvizsgálja, hogy a szó szótő vagy összetett szó-e (123-as művelet), mivel nem, és ehhez a toldalékhoz tartozó műveletek elfogytak, az aktuális szóval rekurzívan meghívja (3. szint) önmagát

- megvizsgálja, hogy a szó szótő-e, azt találja, hogy nem, elkezdi elemezni
- nem talál alkalmazható toldalékot, ezért *null* értékkel tér vissza (3. szint)
- nem talál további alkalmazható toldalékot, ezért *null* értékkel tér vissza (2. szint)
- mivel nincs további művelet, visszatér az eredeti szóhoz (*imádkozzunk*) és a következő alkalmazható toldalékot keresi
- következő alkalmazhatóként megtalálja az *–unk* toldalékot, végrehajtja a megadott műveleteket (4, 100, 30, 126)
- eltávolítja az *–unk* toldalékot (4-es művelet), beállítja a hozzá tartozó nyelvi jellemzőket (szótő toldalékkal és toldalék nélkül: 1 [ige]; szám: 2 [többes szám]; személy: 1 [első személy]; birtok szám: 0 [nem definiált]; mód: 3 [felszólító mód]; idő: 1 [jelen idő]; ragozás: 1 [alanyi ragozás]); *imádkozz*
- rövidíti a szóvégi hosszú mássalhangzót (100-as művelet); *imádkoz*
- megvizsgálja, hogy a szó szótő vagy összetett szó-e (123-as művelet), mivel nem, folytatja a következő művelettel
- hozzáfűzi a szóhoz az *–ik* végződést (30-as művelet); *imádkozik*
- megvizsgálja, hogy a szó szótő-e (126-os művelet), mivel igen, ezzel tér vissza.

Tehát az algoritmus az *imádkozzunk* bemeneti szóra az *imádkozik* szótövet adja vissza, valamint az alábbi nyelvi információkat:

- szófaj: 1 [ige]
- szám: 2 [többes szám]
- személy: 1 [első személy]
- birtok száma: 0 [nem definiált]
- mód: 3 [felszólító mód]
- idő: 1 [jelen idő]
- ragozás: 1 [alanyi ragozás]

2. példa

A *legjobbjainkért* szó elemzése során a következő lépéseket teszi a *WordAnalyzer* program:

- a *getRoots* metódus paraméterként megkapja a *legjobbjainkért* szót
- felbontja a – jelek mentén, ekkor kaptunk egy egyelemű *String* tömböt az *legjobbjainkért* szóval
- ebből az elemből létrehoz egy *LangWord* példányt és átadja a *getRoot* metódusnak
- a *getRoot* metódus megvizsgálja, hogy szótő-e, mivel nem, elkezdi elemezni
- első alkalmazhatóként megtalálja az *–ért* toldalékot, végrehajtja a megadott műveleteket (4, 123)
- eltávolítja az *–ért* toldalékot (4-es művelet), beállítja a hozzá tartozó nyelvi jellemzőket (szótő toldalékkal és toldalék nélkül: 2 [névszó vagy egyéb]; a többi jellemző: 0 [nem definiált]); *legjobbjaink*
- megvizsgálja, hogy a szó szótő vagy összetett szó-e (123-as művelet), mivel nem, és ehhez a toldalékhoz tartozó műveletek elfogytak, az aktuális szóval rekurzívan meghívja (2. szint) önmagát
- megvizsgálja, hogy a szó szótő-e, azt találja, hogy nem, elkezdi elemezni
- első alkalmazhatóként megtalálja az *–jaink* toldalékot, végrehajtja a megadott műveleteket (4, 123)
- eltávolítja a *–jaink* toldalékot (4-es művelet), beállítja a hozzá tartozó nyelvi jellemzők közül a még nem definiáltakat (szám: 2 [többes szám]; személy: 1 [első személy]; birtok száma: 2 [többes szám]); *legjobb*
- megvizsgálja, hogy a szó szótő vagy összetett szó-e (123-as művelet), mivel nem, és ehhez a toldalékhoz tartozó műveletek elfogytak, az aktuális szóval rekurzívan meghívja (3. szint) önmagát
- megvizsgálja, hogy a szó szótő-e, azt találja, hogy nem, elkezdi elemezni
- első alkalmazhatóként megtalálja a *–bb* toldalékot, végrehajtja a megadott műveleteket (4, 71, 123, 103, 123, 105, 123)

- eltávolítja a *–bb* toldalékot (4-es művelet), és változatlanul hagyja az eddigi nyelvi jellemzőket; *legjo*
- eltávolítja a *leg–* toldalékot (71-es művelet); *jo*
- megvizsgálja, hogy a szó szótő vagy összetett szó-e (123-as művelet), mivel nem, folytatja a következő műveletetekkel
- rövidíti a szóvégi magánhangzót (103-as művelet); *jo*
- megvizsgálja, hogy a szó szótő vagy összetett szó-e (123-as művelet), mivel nem, folytatja a következő műveletetekkel
- hosszabbítja a szóvégi magánhangzót (105-ös művelet); *jó*
- megvizsgálja, hogy a *jó* szó szótő vagy összetett szó-e (123-as művelet), mivel szótő, ezért ezzel tér vissza (3. szint)
- a rekurzív hívás eredményeként kapott szótővel tér vissza (2. szint)
- a rekurzív hívás eredményeként kapott szótővel tér vissza (1. szint)

Tehát az algoritmus a *legjobbjainkért* bemeneti szóra a *jó* szótövet adja vissza, valamint az alábbi nyelvi információkat:

- szófaj: 2 [névszó vagy egyéb]
- szám: 2 [többes szám]
- személy: 1 [első személy]
- birtok száma: 2 [többes szám]
- mód: 0 [nem definiált]
- idő: 0 [nem definiált]
- ragozás: 0 [nem definiált]

A példa után következzen a fent már szövegesen leírt algoritmus konkrét Java nyelvű megvalósításának bemutatása. Ennek során nem térünk ki minden részletre, mivel az egyes metódusok nevei elég beszédesek, valamint ezek általában egyszerű sztringmanipulációt hajtanak végre.

A szótőkeresés algoritmusának konkrét megvalósítása Java nyelven

A *getRoots* metódus

```
public static LangWord[] getRoots(LangWord langWord)
{
   // átmeneti vektor létrehozása az eredménynek
   Vector<LangWord> rootsVect = new Vector<LangWord>();

   // szó felbontása tagszavakra a kötőjelek mentén
   String[] tempStrArray = (langWord.getLangWordString()).split("[-]");

   // a tagszavak szótövének keresése
   for (int i = 0; i < tempStrArray.length; ++i)
   {
      // az i-edik tagszó szótövének keresése a rekurzív metódus meghívásával
      LangWord tempLangWord = getRoot(new LangWord(tempStrArray[i]));

      // ha nem találta meg a tagszó tövét, üresszó hozzáadása a vektorhoz,
      // és folytatás a következő tagszóval
      if (tempLangWord == null)
      {
         rootsVect.add(new LangWord(""));
         continue;
      }

      // ha az aktuális szó összetett szó, az előtag és az utótag hozzáadása a vektorhoz
      if (tempLangWord.isCompoundWord() && tempLangWord.getCompounds() != null)
      {
         rootsVect.add(tempLangWord.getCompounds()[0]);
         rootsVect.add(tempLangWord.getCompounds()[1]);
      }

      // ha az aktuális szó nem összetett szó, de szótő, hozzáadás a vektorhoz
      else if (isRoot(tempLangWord.getLangWordString()))
      {
         rootsVect.add(tempLangWord);
      }
   }

   // LangWord típusú tömb előállítása az átmeneti vektorból

   LangWord[] result = new LangWord[rootsVect.size()];

   for (int i = 0; i < rootsVect.size(); ++i)
   {
      result[i] = rootsVect.get(i);
   }

   // visszatérés a létrehozott LangWord típusú tömbbel
   return result;
}
```

A *getRoot* rekurzív metódus

```
   public static LangWord getRoot(LangWord langWord)
   {
      // ha a bemeneti szó szótő, ezzel tér vissza
      if (isRoot(langWord.getLangWordString()))
      {
         return langWord;
      }
      // ha nem, megkezdődik a szótő keresése

      LangWord langWordWithoutAffix;
      LangWord tempLangWord;

      // sorra veszi az egyes toldalékokat
outer:for (int i = 0; i < postfixes.size(); i++)
      {
         // ha az i-edik toldalék alkalmazható...
         if (isPostfixApplicable(langWord, postfixes.get(i)))
         {
            // átmeneti változókban eltárolja a bemeneti szót és a toldalék nélkülit
            langWordWithoutAffix = langWord.clone();
            langWordWithoutAffix.removePostfix(postfixes.get(i));
            tempLangWord = langWord.clone();

            // sorra veszi az i-edik toldalékhoz tartozó műveleteket
            for (int j = 0; j < postfixCommands.get(i).size(); ++j)
            {
               switch ((Byte)(postfixCommands.get(i)).get(j))
               {
                  case 4: // toldalék eltávolítása a szó végéről
                     {
                        tempLangWord = langWordWithoutAffix.clone();
                        break;
                     }
                  case 5: // szóvégi 'v' eltávolítása
                     {
                        tempLangWord.removePostfix("v");
                        break;
                     }
                  case 6: // szóvégi 'a' eltávolítása
                     {
                        tempLangWord.removePostfix("a");
                        break;
                     }
                  case 7: // szóvégi 's' eltávolítása
                     {
                        tempLangWord.removePostfix("s");
                        break;
                     }
                  case 30: // 'ik' hozzáfűzése a szó végéhez
                     {
                        tempLangWord.appendPostfix("ik");
                        tempLangWord.setPartIdWithAffix((byte)1);
                        break;
                     }
                  case 31: // 'ó' hozzáfűzése a szó végéhez
                     {
                        tempLangWord.appendPostfix(new LangWord("ó"));
                        break;
                     }
                  case 32: // 't' hozzáfűzése a szó végéhez
                     {
                        tempLangWord.appendPostfix(new LangWord("t"));
                        break;
                     }
                  case 33: // 'sz' hozzáfűzése a szó végéhez
                     {
                        tempLangWord.appendPostfix(new LangWord("sz"));
                        break;
                     }
```

```
case 34: // 'ed' hozzáfűzése a szó végéhez
   {
      tempLangWord.appendPostfix(new LangWord("ed"));
      break;
   }
case 35: // 'od' hozzáfűzése a szó végéhez
   {
      tempLangWord.appendPostfix(new LangWord("od"));
      break;
   }
case 36: // 'ud' hozzáfűzése a szó végéhez
   {
      tempLangWord.appendPostfix(new LangWord("ud"));
      break;
   }
case 37: // 'üd' hozzáfűzése a szó végéhez
   {
      tempLangWord.appendPostfix(new LangWord("üd"));
      break;
   }
case 38: // 'esz' hozzáfűzése a szó végéhez
   {
      tempLangWord.appendPostfix(new LangWord("esz"));
      break;
   }
case 39: // 'usz' hozzáfűzése a szó végéhez
   {
      tempLangWord.appendPostfix(new LangWord("usz"));
      break;
   }
case 40: // 'üsz' hozzáfűzése a szó végéhez
   {
      tempLangWord.appendPostfix(new LangWord("üsz"));
      break;
   }
case 41: // 'a' hozzáfűzése a szó végéhez
   {
      tempLangWord.appendPostfix(new LangWord("a"));
      break;
   }
case 42: // 'ta' hozzáfűzése a szó végéhez
   {
      tempLangWord.appendPostfix(new LangWord("ta"));
      break;
   }
case 71: // szóeleji 'leg' eltávolítása
   {
      tempLangWord.removePrefix(new LangWord("leg"));
      break;
   }
case 100: // szóvégi mássalhangzó rövidítése
   {
      tempLangWord.shortenEndingConsonant();
      break;
   }
case 101: // szóvégi mássalhangzó hosszabbítása
   {
      tempLangWord.lengthenEndingConsonant();
      break;
   }
case 102: // szóvégi utolsó két mássalhangzó felcserélése
   {
      tempLangWord.swapLastTwoConsonants();
      break;
   }
case 103: // szóvégi magánhangzó rövidítése
   {
      tempLangWord.shortenEndingVowel();
      break;
   }
```

```
                    case 104: // szóvégi magánhangzó hosszabbítása
                       {
                          tempLangWord.shortenPreviousVowel();
                          break;
                       }
                    case 105: // a szó utolsó magánhangzójának rövidítése
                       {
                          tempLangWord.lengthenEndingVowel();
                          break;
                       }
                    case 106: // a szó utolsó magánhangzójának hosszabbítása
                       {
                          tempLangWord.lengthenPreviousVowel();
                          break;
                       }
                    case 107: // magánhangzó beszúrása a szó utolsó két mássalhangzója közé
                       {
                          tempLangWord.insertVowel();
                          break;
                       }
                    case 122: // ha szótő, vége, különben következő művelet
                       {
                          if (isRoot(tempLangWord.getLangWordString()))
                          {
                             return tempLangWord;
                          }
                          break;
                       }
                    case 123: // ha szótő vagy összetett szó, vége, különben következő művelet
                       {
                          if (isRoot(tempLangWord.getLangWordString()) ||
tempLangWord.isCompoundWord())
                          {
                             return tempLangWord;
                          }
                          break;
                       }
                    case 124: // ha szótő, vége, különben visszatérés a toldalék nélküli alakhoz
                       {
                          if (isRoot(tempLangWord.getLangWordString()))
                          {
                             return tempLangWord;
                          }
                          tempLangWord = langWordWithoutAffix.clone();
                          break;
                       }
                    case 125: // ha szótő vagy összetett szó, vége, különben visszatérés
                              // a toldalék nélküli alakhoz
                       {
                          if (isRoot(tempLangWord.getLangWordString()) ||
tempLangWord.isCompoundWord())
                          {
                             return tempLangWord;
                          }
                          tempLangWord = langWordWithoutAffix.clone();
                          break;
                       }
                    case 126: // ha szótő, vége, különben visszatérés az eredeti alakhoz
                              // és a következő alkalmazható toldalék keresése
                       {
                          if (isRoot(tempLangWord.getLangWordString()))
                          {
                             return tempLangWord;
                          }
                          tempLangWord = langWord.clone();
                          continue outer;
                       }
```

```
                case 127: // ha szótő vagy összetett szó, vége, különben visszatérés az
                          // eredeti alakhoz és a következő alkalmazható toldalék keresése
                   {
                      if (isRoot(tempLangWord.getLangWordString()) ||
tempLangWord.isCompoundWord())
                      {
                         return tempLangWord;
                      }
                      tempLangWord = langWord.clone();
                      continue outer;
                   }
             }
          }
          if (!(isRoot(tempLangWord.getLangWordString()) || tempLangWord.isCompoundWord()))
          {
             if (!tempLangWord.getLangWordString().equals(langWord.getLangWordString()))
             {
                // ha az aktuális alak nem szótő és nem összetett szó, valamint nem maga
                // az eredeti szó, rekurzívan meghívja önmagát
                LangWord l = getRoot(tempLangWord);

                // ha a rekurzív hívás során megtalálta a szótövet, ezzel tér vissza
                if (l != null)
                {
                   return l;
                }
               // ha nem, folytatja a keresést
             }
             // ha az aktuális alak nem szótő és nem összetett szó, hanem maga
             // az eredeti szó, folytatja a keresést
             continue;
          }
          else
          {
             // ha az aktuális alak szótő vagy összetett szó, ezzel tér vissza
             return tempLangWord;
          }
       }
    }
    // ha egyik alkalmazható toldalék eltávolításával, műveleteinek végrehajtásával sem
    // találta meg a szótövet, null-lal tér vissza
    return null;
 }
```

Kifejezés keresése az indexekben

A kifejezések keresése hasonló lépésekben történik, mint ahogyan azt az elemzésnél láthattuk, azzal a különbséggel, hogy ezúttal nem dokumentumot, hanem a keresett kifejezést elemzi a *WordAnalyzer*. Az eredményül kapott, a szótövek indexeit tartalmazó vektor elemeit illeszti a dokumentumok elemzése során létrehozott indexállományokban található indexekhez. Végül a találatokhoz létrehozza az eredeti dokumentumból a megjelenítendő találati dokumentumot, amelyben az egyes találatok ki vannak emelve.

Ebben a részben tehát részletesen látni fogjuk a keresés egyes lépéseit, amelyek az alábbiak:

- a keresett kifejezés bontása szavakra,
- a szavak szótövének meghatározása, ezek indexének visszakeresése
- a keresési könyvtárban található indexállományok felderítése,
- az egyes indexállományok betöltése,
- a szótővek indexének keresése,
- a találati dokumentum létrehozása.

Szavak kinyerése a keresett kifejezésből

Az első lépésben – a *getExpression* metódus végrehajtásával – a billentyűzetről bevitt keresendő kifejezés elemzése történik meg. Az elemzéshez hasonlóan itt is először szavakra bontja a bemeneti karakterláncot a *WordAnalyzer*:

- + jelek mentén alkifejezésekre bontja,
- az egyes alkifejezésekből eltávolítja az esetleges idézőjeleket,
- szavakra bontja az alkifejezéseket.

Az így kapott alkifejezések szavainak szótövét az előzőekben ismertetett *getRoots* metódus segítségével előállítja, ezen szótövek indexét a *getWordIndex* metódus meghívásával lekérdezi, és belehelyezi egy vektorba. Ez a vektor tehát vektorokat tartalmaz, ezen vektorok elemei az egyes alkifejezések szavainak szótőindexei.

Mivel előfordulhat, hogy egy keresett alkifejezés elemzése nem jár sikerrel, azaz az alkifejezés egyetlen szavának szótövét sem sikerül megtalálni, ezért a keresés hatékonyságának növelése érdekében az ilyen alkifejezéseket elhagyja a rendszer. Amennyiben egyetlen alkifejezés elemzése sem sikerül, a keresési folyamat már ekkor megáll a „*Search finished. No root found in expression.*” üzenettel.

Indexállományok felderítése

Az elemzendő HTML dokumentumok felderítéséhez hasonlóan keresés során a *WordAnalyzer* program *searchWordsFiles* metódusa meghívja a *getFilesRecursively* metódust, amely rekurzív módon összegyűjti a megadott keresési könyvtáron belül található összes állomány listáját. Ezután a lista elemeit sorra veszi, és a következő részben ismertetett módon keresi a megadott kifejezést az előző elemzések során létrehozott *.words* kiterjesztésű indexállományokban.

Kifejezés keresése az indexállományokban

Elsőként a program betölti, deszerializálja az adott indexállományt és eldönti róla, hogy egyáltalán az adott alkifejezések mindegyike megtalálható-e benne. Amennyiben igen, az eredeti dokumentum alapján létrehozza a találati dokumentumot, és beleteszi a találati listába. Ha valamelyik alkifejezés nem szerepel, a következő indexállományt kezdi vizsgálni, teszi ezt mindaddig, amíg az összeset át nem vizsgálta.

Megjelenítendő dokumentum létrehozása az eredeti dokumentum alapján

Miután kiderült, hogy az adott dokumentum tartalmazza a keresett alkifejezéseket, a *WordAnalyzer* a találatok vizuális megjelenítése céljából az eredeti dokumentum felhasználásával létrehozza a találati dokumentumot, amelyben a megtalált alkifejezések bíbor színnel és félkövér betűtípussal jelennek meg. Ehhez szükség van arra az információra, hogy az eredeti szövegben milyen pozícióban találhatók meg az alkifejezésekben szereplő szavak.

Az találati dokumentum létrehozása során tehát első feladat, hogy az indexállományból kigyűjtsük azon szavak indexét, amelyeket ki kell emelnünk. Ezt valósítja meg a *WordSearch* osztály *getSeqNumbers* metódusa. Bemenetként megkapja az indexállományból kiolvasott vektort, valamint a keresett kifejezést reprezentáló vektort, és a szavak indexét tartalmazó *Integer* bázistípusú vektorral tér vissza.

A mintaillesztés egyfajta mezítlábas módszerrel történik, ám megvalósítása kissé bonyolultabb mint karakterláncok esetén, mivel vektorok vektorának elemeit keressük egy inhomogén vektorban.

Az egyszerűség kedvéért az indexállományból beolvasott vektorból létrehozunk egy *Integer* típusú objektumokat tartalmazó vektort is a *getIntVectorFromVector* metódus segítségével, ezt felhasználva keressük meg az adott alkifejezés legelső előfordulását a forrásvektorban.

Ha az alkifejezés egy szóból áll, egyszerűen végig tudunk lépkedni a forrásvektor elemein, amennyiben ez *Integer* típusú objektum, összehasonlítjuk az alkifejezés egyetlen elemével, ha pedig *Integer* bázistípusú vektor, akkor ennek egyes elemeivel hasonlítunk. Ha az elemek egyeznek, a kimeneti vektorhoz hozzáadjuk a forrásvektor elemének indexét.

Ha az alkifejezés több szóból áll, hasonlóképpen végezzük az összehasonlításokat, viszont egy további ciklusra van szükség, ami az alkifejezés szavain lépked végig.

A fenti lépések végrehajtásával előálló indexvektorban szerepel a forrásvektor mindazon elemének indexe, amely tartalmazza a keresett kifejezés bármely alkifejezésében álló szavának szótövét.

Ezután a *createResultFile* metódus a megadott dokumentumból előállítja a *.result.html* kiterjesztésű találati dokumentumot a kapott indexek segítségével. Ez a metódus ugyanúgy dolgozza fel az eredeti szöveges dokumentumot, mint a *getWordsFromFile* metódus, annyi kivétellel, hogy nem alakítja át a dokumentumban talált szavakat, csak az indexeknek megfelelőeket emeli ki formázó HTML-jelölő beszúrásának segítségével.

Miután a találati dokumentum létrejött, az adott találat megjelenik a felhasználói felületen a találati listában az eredeti állománynévvel, rá duplán kittintva az operációs rendszeren beállított alapértelmezett böngésző ablakában pedig az újonnan létrehozott dokumentum nyílik meg.

Magyar nyelvű szövegek feldolgozása során felmerülő problémák

A számítógép használata során, így az internetes keresés esetén –, és a hétköznapi élet más területén – is gyakran találkozunk nyelvünk különlegességéből, egyediségéből származó problémákkal, amelyek nehezítik a dolgunkat.

Ebben a részben sorra veszem a főbb problémákat, amelyekkel szembesülök nap mint nap, vagy szembesültem a *WordAnalyzer* készítése közben, valamint bemutatom a megvalósított vagy éppen a még megvalósításra váró megoldást.

Karakterkódolás

Helyesírásunk a latin betűs írások közé tartozik, mivel a sajátos magyar betűsor a latin betűkészletből alakult ki. [9]

Probléma

Nyelvünk számos olyan ékezetes betűt tartalmaz, amelyek bizonyos kompatibilitási megfontolások miatt speciális módon vannak ábrázolva az elektronikus dokumentumokban, ezért feldolgozás során külön figyelmet kell fordítani a különböző írásmódok megfelelő konvertálására.

Megoldás

A *WordAnalyzer* az állományokat a megfelelő karakterkódolással dolgozza fel, valamint az esetlegesen előforduló ún. escape szekvenciák és ISO kódok egy részét is kezeli. Mivel a dolgozat fő célja a szótőveken alapuló keresés hatékonyságának bemutatása, ezért az előfeldolgozó jelenlegi megvalósítása nem teljeskörű, azonban ez később könnyen tökéletesíthető.

A magyar ábécé

A magyar ábécé betűi között vannak egyjegyűek: *e*, *i*, *a*, *ü*; *b*, *r*, *s*, *v*; stb. és többjegyűek: *sz*, *ty*, *zs*, *dzs*; stb. illetve ezeknek hosszú változata is: *é*, *í*, *á*, *ű*; *bb*, *rr*, *ss*, *vv* és *ssz*, *tty*, *zzs*, *ddzs* stb. [9]

Probléma

A rövidülésekből, illetve hosszabbodásokból adódó változásokra a betű szerinti keresés érzékeny. Például ha az *ész* szóra keresünk, nem találjuk meg az *ésszel* szót, vagy például a *kutya* szóra keresve nem jelennek olyan találatok, amelyben a *kutyának* szó szerepel.

Megoldás

A *WordAnalyzer* nyelvi modellje tartalmazza a nyelv ábécéjét, rövid és hosszú magán- és mássalhangzó párjait, ezeket a különböző toldalékokkal kapcsolatos műveletek végrehajtása során felhasználja. Valamint, mivel nem betű szerint keres, hanem szótőegyezést vizsgál, természetesen a fenti példákra is működik.

A magyar nyelv szavainak felépítése

A magyar nyelv a ragozó (agglutináló) típusba tartozik, ami a szavak egyfajta sajátos felépítését jelenti.

A szavak nyelvészeti szempontból kétfélék. Egy részük nem elemezhető: *tégla*, *fekete*, *csalamádé*, mert a hangoknak nincs közük a szóhoz, amelyben állnak. A szavak más része elemezhető, mert jelentéssel bíró elemekből áll: pl. *el-kapar-hat-ná-nk*, *vas-lemez-szerű-t*. A jelentéssel bíró elemeket morfémának nevezzük. Morféma a *csalamádé*, és morféma a *-t*.

A magyar nyelv erősen szintetikus, mégpedig egymáshoz ragasztja a jelentéselemeket, morfémákat (innen származik az *agglutináló* elnevezés, mivel a szó latin jelentése „hozzáragaszt, összeilleszt”) . Például a *fiúknak* szó három morfémából áll: *fiú-k-nak*, az első tag a szó töve, utána következik a többes szám jele, a *-k*, majd a részes esetet fejezi ki a szóvégi *-nak*. [10]

Probléma

Az agglutináló nyelvekben –, így a magyarban is – a szavak nagyon hosszúra nőhetnek, például: *viccelődéseitekben*, *morgolódásaitokban*. Az ilyen szavakat az internetes keresők általában csak eredeti, vagy esetleg az utolsó toldalékban eltérő alakjában találják meg.

Megoldás

A *WordAnalyzer* a szavakat a szó végéről indulva megpróbálja elemeire bontani, és megtalálni a szavak tövét, majd ez alapján keres. Teheti ezt az agglutináció alapelve értelmében, mely szerint a toldalék hozzáragasztásakor csak a legutolsó elemet szabad nézni, ahhoz (és csakis ahhoz) illeszkedik az újabb toldalék, azaz az újabb toldalék nem „lát át" az őt megelőző feje fölött. [10] Például a *-nak* rag kapcsolódhat egyes és többes számú névszóhoz is, pl. *fiú-nak*, ill. *fiú-k-nak*.

A toldalékok eltávolításakor figyelembe veszi, hogy milyen szófajú szót kapott az előző toldalék leválasztásával. Például ha egy szóról eltávolította a *–ban* határozóragot, „tudja", hogy az eredeti szó névszó, és nem próbálja igei toldalékoktól megfosztani.

Többalakú szótövek

A magyar szavakat – mind az igéket, mind a névszókat – két fő csoportba sorolhajtuk: az egyalakúak és a többalakúak csoportjába. Az előbbiek minden toldalék előtt változatlan hangalakúak, pl. *kérek, kértem*; az utóbbiak viszont nem.

Probléma

Számos többalakú szótő van a magyarban, amelyek toldalékos formában más hangalakkal rendelkeznek, mint toldalék nélkül. Néhány példa többalakú szavakra [11]:

- *igényel, igénylés*; *forog, forgott*; *méreg, mérget*; *kapocs, kapcsot* stb.
- *alszik, aludjon, alvás*; *eszik, egyen, evés* stb.
- *dulakszik, dulakodik*; *tanakszik, tanakodik*; *igyekszik, igyekezik* stb.
- *út, utas*; *epe, epés*; *nyár, nyarat, nyara*; *híd, hídja, hidak* stb.
- *lő, lövet*; *nő, növés*; *kő, követ*; *mű, művet*; *ló, lovak* stb.
- *kehely, kelyhet*; *pehely, pelyhes* stb.

Megoldás

A *WordAnalyzer* által használt általános nyelvi modell alkalmas többek között a fenti szavak eredeti, toldalék nélküli hangalakjának felismerését lehetővé tévő műveletsorok megadására. Például a *lovak* szó elemzése a következőképpen zajlik:

- eltávolítja az *-ak* végződést (4-es művelet); *lov*; mivel ez nem szótő,
- eltávolítja a szóvégi *v*-t (5-ös művelet); *lo*; mivel ez még mindig nem szótő,
- meghosszabbítja a szóvégi magánhangzót (105-ös művelet); *ló*; ez pedig a szótő.

Azonos alakú szavak

Más nyelvekhez hasonlóan a magyar nyelvben is vannak olyan szavak, amelyeknek azonos a hangalakjuk. Erre jó példa a *vár* hangalakú szavunk, amely lehet egy cselekvésre, a várakozásra utaló ige, és lehet egy kastélyszerű építményt, erődítményt jelentő főnév.

Probléma

Általában a több jelentés közül csak az egyikre szeretnénk rákeresni.

Megoldás

A *WordAnalyzer* a szavak toldalékos alakjából meg tudja állapítani a szótő szófaját is, így lehetőség nyílhat arra, hogy szófaj alapján keressünk. Ha azonban a szótő fordul elő a szövegben, illetve a keresett kifejezésben, erre nincs lehetősége, ehhez a szövegkörnyezet további elemzése lenne szükséges.

Jelenleg az indexekben csak a szótövek indexei vannak letárolva, azonban lehetőség lenne a járulékos nyelvi információk rögzítésére is, így például meg lehetne adni, hogy csak azok a találatok jelenjenek meg, ahol a keresett kifejezésben szereplő valamely jellemzője – pl. igemódja – megegyezik az eredeti szövegben szereplő szó jellemzőjével.

Összetett szavak

A képzők alkalmazásán (toldalékoláson) kívül a szóképzés másik eszköze a szóösszetételek alkotása. Az összetett szavak két vagy több külön-külön is értelmes szó összekapcsolásával jönnek létre, előtagból és utótagból állnak. [11]

Probléma

A minőségjelzős mellérendelő szóösszetételekben az előtag valamilyen szempontból minősíti az utótagot: pl. *vörösbor*, *fehérbor*, *gyümölcsbor* stb. Mindhárom szó tulajdonképpen egy-egy borfajta. Tegyük fel, számunkra most ez a minőség nem fontos, a borokról szeretnénk általános információt keresni az Interneten. Ha beírjuk a keresőbe a *bor* szót, nem feltétlenül kapunk olyan találatot, amely csak a fenti három szó valamelyikét tartalmazza, és a *bor* szót nem.

Megoldás

A *WordAnalyzer* miután azt találta, hogy a szó a szótárában szereplő szavak összetétele vagy pedig egyetlen szótárbeli szó, befejezi az elemzést. Ezután további értelmes részekre, összetételre próbálja bontani a tagszavakat, majd ezek az összetételek tagonként kerülnek bele az indexbe. A fenti példa alapján:

- *vörösbor = vörös + bor*
- *fehérbor = fehér + bor*
- *gyümölcsbor = gyümölcs + bor*

Ha ezek után a *bor*, illetve a *vörös*, a *fehér* vagy a *gyümölcs* szóra keresünk, a találati listában megjelennek a fentiek közül a megfelelő szavakat tartalmazó dokumentumok is.

Igekötők

Az igekötők igéhez, igenévhez vagy más, igéből képzett névszóhoz kapcsolódó, annak jelentését módosító szavak. Csak formálisan szavak, funkcionálisan nem, mivel általában nincs önálló fogalmi tartalmuk. [11]

Probléma

Az igekötők állhatnak az ige, igenév vagy más, igéből képzett névszó

- előtt közvetlenül, ekkor vele egybeírjuk (pl. *eladta*)
- mögött közvetlenül, ekkor tőle különírjuk (pl. *adta el*)
- előtt úgy, hogy kettőjük közé harmadik szó ékelődik, akkor a három szót különírjuk egymástól (pl. *el kellett adnia*).

Amennyiben az első alakban keresnénk az igét, csak ilyen formában találnánk meg, ha viszont a második vagy a harmadik alakban, akkor az egybeírt előfordulások nem jelennének meg a találati listában. Továbbá ha az adott ige a keresett igekötővel nem fordul elő egyetlen dokumentumban sem, annak ellenére is üres találati listát kapnánk, hogy az igekötő csak módosítja a szó alapjelentését.

Megoldás

A *WordAnalyzer* az igekötőket szótövekként kezeli, így az igével egybeírt igekötő ugyanúgy kerül az indexbe, mintha összetett szó lenne. Így ha különírva keressük az igét, egybeírva is megtaláljuk, és fordítva, valamint ha az igét önmagában keressük, akkor az igekötős alakját is megtaláljuk.

További problémák

Az előbbiekben említetteken túl további nyelvi nehézségekbe ütközhetünk, mivel a természetes nyelvekhez nem adható olyan szabályrendszer vagy nyelvi modell, amely teljesen lefedné. Azonban az *A WordAnalyzer működése* c. fejezetben ismertetett szótőkereső algoritmus általánosnak mondható, mivel lehetővé teszi annak definiálását, hogy bizonyos végződések után milyen műveletek végrehajtásával próbáljon eljutni a szótőig, így a `postfixes.csv` állomány módosításával tökéletesíthető a jelenlegi működés.

Problémát jelent viszont a többalakú szótövek egy bizonyos fajtájának megléte a magyar nyelvben. Ezek olyan szavak, amelyeknek mind szófaja, mind jelentése azonos, a hangalakjuk tér el pl. egy-egy magánhangzóban: *tejföl – tejfel*, *söpör – seper* stb. Tapasztalatom szerint ezt a problémát még más keresők sem kezelik, ha az egyik alakot megadjuk kereséskor, a másik alakban lévő szavakat nem találják meg. Egy lehetséges megoldás lehetne, hogy ezeket a szavakat egy-egy csoportba sorolnánk, és szinonímaként kezelnénk, vagy egy olyan új szabályt bevezethetnénk be, amely a szó hangrendjének figyelembevételével módosítja a magánhangzókat.

Összefoglalás

Ahogy azt a *Bevezetés* részben megfogalmaztam, e dolgozat célja egy olyan eljárás kidolgozása volt, amelynek segítségével a jelenlegi internetes keresők a magyar nyelvű dokumentumokban hatékonyabban tudnak keresni. Az előzőekben láthattuk a szótőalapú keresés előnyeit, azonban még nem végeztünk konkrét összehasonlításokat a hagyományos keresők és a *WordAnalyzer* kereső hatékonyságát illetően.

Összefoglalásként tehát összehasonlítom, hogy egy hagyományos internetes kereső és a *WordAnalyzer* bizonyos keresendő kifejezésekre a példaként szolgáló zeneszövegekből milyen találatokkal tér vissza. Az összehasonlíthatóság érdekében a hagyományos kereső által visszaadott találatok közül kiválogatom azokat a www.zeneszoveg.hu oldalról származó dokumentumokat, amelyekben a *WordAnalyzer* is keres.

Keresett kifejezés	Hagyományos találatok száma (a találatokban szereplő szavak)	*WordAnalyzer* találatok száma (a találatokban szereplő szavak)
+kő	14 db (*kő*)	24 db (*kő, kőből, kőfaragó, kőre, kőtől, kövek, kövekkel, kővel*)
+kövek	16 db (*ko, kő, kövek*)	24 db (*kő, kőből, kőfaragó, kőre, kőtől, kövek, kövekkel, kővel*)
+kövek+repülnének	0 db	14 db (*kő, kőfaragó, kőre, kővel, repül, repülj, visszarepülni*)
+"legnagyobb kövek"+zuhanjanak	0 db	2 db (*nagy, kő, zuhan*)
+Republic	128 db (*Republic*)	0 db

A fenti táblázatból látszik, hogy a *WordAnalyzer* a szótő alapú keresés segítségével jelentősen több alakjában találja meg a keresett szavakat, mint egy hagyományos kereső. Tehát mondhatjuk, hogy az eredeti célt sikerült elérni, a program megtalálja a szavakat akkor

is, ha különböző alakban szerepelnek a szövegben és a keresett kifejezésben, azonban a szótövük azonos.

A továbbiakban még rávilágítok pár problémára, amellyel a dolgozat készítése során szembesültem. Hozzá kell tenni azonban, hogy az esetek nagy többségében (kb. 85-90%-ában) a *WordAnalyzer* megtalálja a szavak tényleges tövét, tehát a következőkben ismertetett problémák elég ritkák.

A program nem boldogul azokkal a szavakkal, amelyeknek nem találja meg a szótövét. Tehát látszik a módszer egyik hátránya; a nem ismert szavak –1-es értékkel (ismeretlen szóként) kerülnek bele az indexbe, mivel nem található hozzájuk szótő a `words.txt` állományból, ezért nem is tudunk rájuk keresni.

Így problémát okozhatnak a számok, a tulajdonnevek és minden egyéb karaktersorozat, amely nem értelmezhető magyar szóként. Előbbiekre megoldásként pl. a –1-nél kisebb, egész értékekű indexeket lehetne számok reprezentálására használni. Utóbbi problémára nem adható ilyen egyszerű megoldás, ez további tervezést igényel. Megoldható lenne a probléma pl. úgy, ha a program önmaga bővítené a saját szótárát, ehhez azonban biztosan el kellene tudnia dönteni, hogy egy, a szótárban nem található szó szótőként kezelendő-e vagy sem.

Előfordulnak olyan esetek is, amikor a *WordAnalyzer* nem találja meg egy valódi magyar szó tövét. Ennek két fő oka lehet: vagy nincs benne a szó töve a `words.txt` állományban, vagy pedig a toldalékokhoz kapcsolódó műveletek nem megfelelően vannak megadva. (Utóbbi esetben az is előfordulhat, hogy másik szótövet talál meg, mint amit elvárnánk.) Ezek a problémák többségében a nyelvi modell hibáiból adódnak, de könnyedén lehet korrigálni a megfelelő bemeneti állományok szerkesztésével, így a megoldáshoz kódmódosítás nem szükséges.

Vannak a magyarban azonos alakú toldalékok (főleg igeragok), amelyeket eltávolítva bár a *WordAnalyzer* megtalálja a szó tövét, a nyelvi jellemzőket nem tudja egyértelműen meghatározni, ehhez a szó környezetét is kellene vizsgálni.

Egyes nyelvekben előfordulnak olyan kivételes szavak, amelyek alakjai közötti átjárás elemi transzformációs műveletekkel nem írható le (ilyenek pl. az angol rendhagyó igék: *go*, *went*, *gone*). Ezek kezelésére szükség lenne egy új komponens bevezetésére a nyelvi modellbe, mégpedig a kivételes alakú szavak listájára.

A fentiekből látható, hogy van még számos nyitott kérdés, ami nem túlzottan meglepő a magyar nyelv igen komplex ragozási rendszerét ismerve. Azonban már a *WordAnalyzer* jelenlegi változata is elég jó hatásfokkal alkalmazható szótőkeresésre.

Bár az eredeti cél az internetes keresők tökéletesítése volt, ez az algoritmus hasznos lehet pl. a könyvtárinformatika területén, vagy nyelvészeti kutatások során, esetleg éppen igen bonyolult szerkezetű, agglutináló nyelvünket tanuló külföldiek számára is.

Irodalomjegyzék

[1] Csontó Béla, Varga Sándor: Internetről középiskolásoknak, Pedellus Kiadó, Debrecen, 2000.

[2] Kurt Franklin: How Internet Search Engines Work
http://computer.howstuffworks.com/search-engine.htm

[3] Szeredi Péter, Lukácsy Gergely, Benkő Tamás: A szemantikus világháló elmélete és gyakorlata, Typotex, Budapest, 2005.

[4] Magyar Tudományos Akadémia: Magyar értelmező kéziszótár, Akadémiai Kiadó, Budapest, 1987.

[5] Jakab László: Tanulmányok az igeragozás köréből, KLTE, Debrecen, 1999.
http://mek.oszk.hu/01700/01711/01711.doc

[6] A nyelvi formák rendszere
http://bme-tk.bme.hu/other/kuszob/nyelvt.htm

[7] Magyar Nyelvőr – Keszler Borbála: A szóképzés
http://www.c3.hu/~nyelvor/period/1241/124106.htm

[8] Hungarian Grammar [HungarianReference.com > Grammar]
http://www.hungarianreference.com/Links/Grammar.aspx

[9] Magyar Tudományos Akadémia: A magyar helyesírás szabályai, Akadémiai Kiadó, Budapest, 1987.

[10] Nádasdy Ádám: Az agglutináció, Magyar Narancs, 2001/06/21, p. 40
http://seas3.elte.hu/delg/publications/modern_talking/13.html

[11] Hajas Zsuzsa: Magyar nyelv 1. – 9. osztálynak, Pedellus Kiadó, Debrecen, 1997.