Keresés korpuszban 2: így kerestek ti[1]

Sass Bálint
tudományos munkatárs
MTA Nyelvtudományi Intézet, Nyelvtechnológiai Kutatócsoport
ELTE Bölcsészettudományi Kar
sass.balint@nytud.mta.hu

2001 óta foglalkozom számítógépes nyelvészettel, korpuszokkal. Programtervező matematikusként végeztem 2003-ban, a PhD-mat 2011-ben védtem meg, témája egy korpuszból igei szerkezeteket automatikusan kinyerő, szótárkészítést segítő, korpuszvezérelt algoritmus volt. Ennek kapcsán 2010-ben jelent meg a Magyar igei szerkezetek szótár. Fontos szakterületem a korpuszlekérdezés módszereinek vizsgálata, oktatása. Az MTA Nyelvtudományi Intézetében számos korpusszal dolgoztam, részt vettem az egymilliárd szónyi mai magyar szöveget tartalmazó Magyar Nemzeti Szövegtár összeállításában, a helyesiras.mta.hu portál létrehozásában, foglalkoztam a magyar Braille-rövidírás megújításával. Mostanában az e-magyar nyelvelemző rendszert gondozom, és legóból építek igei szerkezeteket.
http://www.nytud.hu/depts/corpus/Sass_Balint.html

1. Bevezetés

Korábbi írásaimban megfogalmaztam a Nemzeti Korpuszportál koncepcióját (Sass 2016), valamint bemutattam a korpuszokban való keresés alapvető módszereit (Sass 2017). A Nemzeti Korpuszportál célja a szóalapú online keresővel rendelkező magyar nyelvű korpuszok összegyűjtése, és távlatilag az elemzőeszközök és a korpuszkeresési funkciók elérhetővé tétele minden korpusz számára. A korpuszban való keresés vonatkozásában áttekintettük a Nemzeti Korpuszportálon több esetben is alkalmazott NoSketchEngine (NoSkE) (Rychlý 2007) korpuszkezelő rendszer funkcióit, különös tekintettel a szűrésre és a gyakorisági listákra, illetve a reguláris kifejezésekre, valamint az arra épülő Corpus Query Language (CQL) formális lekérdezőnyelvre, melynek segítségével teljeskörűen feltárható egy korpusz annotációjában kódolt minden nyelvi és egyéb információ.

[1] Az Információs és Technológiai Minisztérium ÚNKP-19-4 kódszámú Új Nemzeti Kiválóság Programjának szakmai támogatásával készült.

2. Lekérdezések osztályai

A tanulmány további részében feltételezzük az említettek ismeretét. Most a Magyar Nemzeti Szövegtár kibővített változata (MNSZ2) (Orvecz et al. 2014) keresőfelületén a korpusz használói által megadott valódi CQL lekérdezésekből – mondhatjuk így: a CQL korpuszlekérdezések alkotta „korpuszból" – szemezgetünk, ezeket elemezzük. Levonjuk a tanulságokat, számba vesszük az ajánlott megoldásokat és az elkerülendő próbálkozásokat. Konkrét gyakorlati tippeket adunk a korpuszlekérdező rendszer használatával kapcsolatban.

A vizsgált lekérdezéseket az elemzésünk során három csoportra fogjuk osztani:

1. érdekes, értelmes lekérdezések
2. hibák
3. „ilyet ne"

A harmadik csoporthoz egy kis magyarázat: ide azok a lekérdezések tartoznak, amelyek kezelhetetlenül sok – rosszabb esetben teljesen haszontalan – találatot eredményeznek, ráadásul nagymértékben leterhelik a korpuszkezelő hardveres és szoftveres környezetet. Az ilyen lekérdezéseket lehetőség szerint próbáljuk meg elkerülni, általában azon a módon, hogy a lekérdezés átgondoltabb megfogalmazásával leszűkítjük, jobban specifikáljuk, jobban meghatározzuk, hogy pontosan mire is vagyunk kíváncsiak.

3. Elemzendő példák

Tekintsük át tehát az alábbi tíz példát, CQL lekérdezést!

a) `"tudatjuk" "mindazokkal"`

b) `[lemma="felkap"] [lemma="a"] [lemma="víz"]`

c) `[word="."]`

d) `[lemma = "k[K]andeláber"]`

e) `ama i*`

f) `[word="\."] [word="[Mm]indig"] [word="\."]`

g)
```
[msd="Det.*"] [msd="FN.PSe2.*"] [lemma="fog"]
  [word=".*ni" & pos="V.*"]
```

h) `".*"`

i) `[][][][]*`

j) `[word = "elé"] [word = "a.?"] [word = ".+n(a|e)k"]`

Mielőtt a következő fejezetben található magyarázatokat, megjegyzéseket elolvassuk, érdemes elgondolkodni azon, hogy az egyes lekérdezéseknek vajon mi a tartalma és célja, és melyik lekérdezést, melyik fenti csoportba sorolnánk.

4. Elemzés

4.1. a) `"tudatjuk" "mindazokkal"`

Első példánk egy hasznos, értelmes, ugyanakkor nagyon egyszerű lekérdezés. Két egymást közvetlenül követő szóalakra (*tudatjuk* és *mindazokkal*) kérdez rá. Kihasználja a NoSkE azon lehetőségét, hogy ha default attribútumot adunk meg a felületen, akkor a CQL lekérdezésbe nem kell beleírnunk ezt az attribútumot, hanem csak egyszerűen idézőjelbe kell tenni a keresett szavakat, illetve a szavakat megadó reguláris kifejezéseket. (Tovább azonban nem egyszerűsíthető a lekérdezés, az idézőjelek nem elhagyhatók, mindenképpen kellenek.) Így, ha default attribútumnak a `word`öt (szóalak) állítjuk be, akkor

```
[word="tudatjuk"] [word="mindazokkal"]
```

helyett írhatjuk egyszerűen a példában látottakat. A lekérdezés segítségével természetesen jó eséllyel gyászjelentések szövegrészleteit fogjuk eredményül kapni, ez is lehetett a lekérdezés tényleges célja erre a nagyon jellegzetes fordulatra való kereséssel.

4.2. b) `[lemma="felkap"] [lemma="a"] [lemma="víz"]`

A példa a `lemma` (szótő) attribútum segítségével három egymást közvetlenül követő szótőre kérdez rá. Nyilván a *felkapja a vizet* szólás előfordulásaira volt kíváncsi a lekérdező, azaz ez a példa is az 1. osztályba tartozik.

Elegendő lett volna a szótőre keresést csupán az igére korlátozni, mivel a szólás másik két eleme fix, így:

```
[lemma="felkap"] [word="a"] [word="vizet"]
```

Vegyük észre ugyanakkor, hogy ez a lekérdezés nem fogja megtalálni e szólás összes előfordulását, mégpedig a különféle lehetséges szórendek, valamint az esetlegesen elváló igekötő miatt. Érdemes elgondolkodni azon, hogyan tudjuk ezeket a problémákat általánosságban kezelni, valamint hogyan tudunk olyan lekérdezést alkotni, amely lehetőséget ad a *víz* elem példányaiból álló gyakorisági lista elkészítésére, lehetővé téve a fenti állítás ellenőrzését, miszerint ez az elem fix.

Ha szóalak szerinti gyakorisági listát készítünk a találatokból, megtudhatjuk, hogy ennek a szólásnak ebben a szórendben mely ragozott alakjai a leggyakoribbak: az első a *felkapja a vizet*, a második a *felkapta a vizet*, a kettő együtt a találatoknak több mint a felét teszi ki.

4.3. c) `[word="."]`

Mire szolgálhat ez a lekérdezés? Tudjuk, hogy a CQL-lekérdezésekben egy token (azaz szó vagy írásjel) adott attribútumára vonatkozó feltételt egy idézőjelek között megadott reguláris kifejezés testesíti meg az alábbi formában:

```
[attribútum="regkif"]
```

A reguláris kifejezésekben a pont (**.**) speciális jelentéssel bír, azt jelenti, hogy azon a helyen tetszőleges karakter állhat. Így az egyetlen pontból álló reguláris kifejezés az egy darab tetszőleges karakterből álló tokenekre kérdez rá. Ebben az esetben az eredményhalmaz egyrészt roppant méretű, az egymilliárd szavas MNSZ2 esetén akár százmilliós nagyságrendű, mivel az írott magyar szövegnek akár 30%-át is alkothatják az egykarakteres tokenek. Másrészt nagyon heterogén az eredményhalmaz, a hasznos közös tulajdonsággal nem nagyon rendelkező különféle egybetűs szavakon (*a, ő, ó, s* stb.) kívül tartalmazza a számos egykarakteres írásjel példányait is. Nem könnyű rájönni, hogy mi lehetett itt a kérdező szándéka, az mindenesetre biztos, hogy ez egy „ilyet ne” típusú lekérdezés.

Lehetséges, hogy egyszerűen a mondatvégi pont írásjelre szeretett volna rákérdezni itt a korpusz használója, de nem volt tisztában azzal, hogy az attribútumok értéke reguláris kifejezésként értelmeződik, és ott a pont speciális karakter. Ha egy speciális karaktert „eredetiben”, azaz speciális jelentésétől mentesítve szeretnénk használni, akkor „escape”-elni kell a visszaper karakter segítségével, ekkor tehát a helyes lekérdezés a következő:

```
[word="\."]
```

4.4. d) `[lemma = "k[K]andeláber"]`

A szándék itt nyilvánvalóan az volt, hogy a *kandeláber* szótőre keresve a szó összes ragozott alakját is megkapjuk úgy, hogy a szó kisbetűvel és nagybetűvel írt verzióit is számításba vesszük. Valóban, ha szükségünk van a kisbetűs és a nagybetűs változatra is, akkor ezt explicite meg kell adnunk. Erre használható a reguláris kifejezésekben a szögletes zárójel, ami egy karakterlistát tartalmazva azt

jelenti, hogy a megadott karakterlistából pontosan az egyik karakter szerepelhet az adott ponton. A

```
[eö]
```

regkif jelentése tehát *e* vagy *ö*, ennek megfelelően a

```
tejf[eö]l
```

jelentése *tejfel* vagy *tejföl*. A kisbetű/nagybetű – esetünkben kis *k* és nagy *K* – vagylagos megadása tehát a következő módon lehetséges:

```
[Kk]
```

Azonban a d) példában a szögletes zárójel csak egy karaktert tartalmaz, ez pont olyan, mintha egyáltalán nem használnánk szögletes zárójelet, azaz a példa azonos értelmű az alábbi – találatot értelemszerűen nem adó – lekérdezéssel:

```
[lemma = "kKandeláber"]
```

A lekérdezés tehát hibás, a 2. osztályba tartozik. A felhasználó szándékának megfelelő lekérdezés a következő lehetett volna:

```
[lemma = "[Kk]andeláber"]
```

Fontos látni a különbséget a kétféle `[]` között: a belső egy reguláris kifejezés részeként karakterek vagylagosságát fejezi ki, a külső az egy tokenre vonatkozó feltétel(eke)t tartalmazza a CQL elemeként.

Megjegyezzük, hogy a tulajdonnevek kivételével a szavak szótöve alapesetben kisbetűs, akkor is, ha egy mondat elején álló nagybetűs szó szótövéről van szó. Azaz itt használható lett volna egyszerűen a következő lekérdezés:

```
[lemma = "kandeláber"]
```

4.5. e) `ama i*`

A lekérdezés egyértelműen hibás, ez már onnan látszik, hogy nem tartalmaz idézőjelet. Ezt javítva

```
"ama" "i*"
```

is rejtélyes marad a lekérdezés célja. Az *ama* szó után következő tetszőleges számú *i* betűből álló szóra vonatkozik a lekérdezés, aminek nehéz hasznos funkciót tulajdonítani. Utolsó ötletként az *ama*-t követő *i*-vel kezdődő szavakra vonatkozó, még mindig nem túl mélyenszántó lekérdezés a következő lenne:

```
"ama" "i.*"
```

Mivel a reguláris kifejezésekben a csillag (`*`) operátor jelentése a „bárhány” (0 vagy több) a megelőző elemből, a tetszőleges számú tetszőleges karakter `.*` formában adandó meg.

4.6. f) `[word="\."] [word="[Mm]indig"] [word="\."]`

Értelmes lekérdezés. Egy mondatvégi pontot követő *mindig* szót keres úgy, hogy azt ismét egy mondatvégi pont kövesse. Azaz egy egy szóból álló mondat megtalálása a cél. A pont karakterek helyesen „escape”-elve vannak (vö: 4.3. rész), a *mindig* szó elején kis- és nagybetűt is megenged a lekérdezés, vélhetően általában fölöslegesen, mert a mondat eleji helyzet miatt lényegében mindig nagybetűs lesz.

Mire szolgálhat? A mondandó végén nyomatékosításként odatett egyszavas *Mindig*. mondatokat szeretné megkapni. Az írásjelek a korpuszban külön tokenek, ezért a lekérdezés fenti formája helyes.

Az MNSZ2 keresőjébe beírt lekérdezések között több hasonló felépítésű is szerepel (*Sokszor., Teljesen.* stb.), a felhasználó valószínűleg az ilyen formájú mondatokról szándékozott általános tanulságokat levonni. Ilyen esetben – szem előtt tartva, hogy lehetőleg az összes releváns korpuszadatra építsük a vizsgálatainkat – érdemes lehet egy általánosabb lekérdezéssel kezdeni, abból gyakorisági listát csinálni és azon vizsgálódni, hogy ne csak az intuitíve felmerülő példákkal dolgozzunk, hanem az összes rendelkezésre álló valós nyelvi adattal.

A javaslat itt konkrétan a

```
"\." [] "\."
```

lekérdezés lenne, amely a vizsgált pozícióban tetszőleges szót megenged (azáltal, hogy az egy tokenre vonatkozó szögletes zárójelpár belsejében nem köt ki feltételt). A találatok között a leggyakoribbak a *Rip. Köszönöm. Köszönjük. Nem. Igen. Ennyi.*, aztán a gyakorisági listán lejjebb következnek a vizsgálat szempontjából vélhetően relevánsabb megnyilatkozások: *Sajnos. Mindegy. Szép. Talán.* stb.

4.7. g) `[msd="Det.*"] [msd="FN.PSe2.*"] [lemma="fog"] [word=".*ni" & pos="V.*"]`

Ebben a fejezetben két olyan lekérdezést vizsgálunk, amelyek ugyanazt a problémát példázzák. Elsőre úgy tűnik, hogy mindkét lekérdezés rendben van, az első az `msd` (morfológiai kód) attribútum segítségével névelős egyes szám máso-

dik személyű birtokos főneveket keres (pl.: *a lovad, az árnyékodba*), a második pedig a *fog* + főnévi igenév szerkezeteket a *-ni* végződés és az igei („ige” szófaj-kóddal kezdődő) morfológiai kód alapján.

A hiba mindkét esetben ott van, hogy nem megfelelő kódokat használtunk. Az MNSZ2-ben (a v2.0.2–v2.0.5 verziókban) a határozott névelő kódja nem `Det`, hanem `DET`, a szófajok kódjai magyar rövidítéssel szerepelnek, azaz az ige nem `V`, hanem `IGE`, és a morfológiai kódot tartalmazó attribútum elnevezése pedig nem `pos`, hanem `msd`.

Ha nem vagyunk biztosak a kódokban, a legegyszerűbb, ha rákeresünk egy olyan konkrét szó(kapcsolat)ra, amilyen jellegűre kíváncsiak vagyunk:

```
"a" "neved.+"
```

(ahol a .+ segítségével a ragozott alakokat is megengedtük), illetve

```
"fog" "csinálni"
```

majd a megjelenítésben bekapcsoljuk az `msd` attribútumot, és megnézzük, hogy milyen kódokat kell használnunk.

4.8. h) `".*"`

Nagyon egyszerű, de annál kártékonyabb lekérdezés. A korábbiakból tudjuk, hogy reguláris kifejezésben a pont (`.`) tetszőleges karakterre illeszkedik, a csillag (`*`) operátor jelentése pedig az, hogy a megelőző karakterből „bárhány” (0 vagy több) darab. Így a `.*` jelentése: bárhány egymást követő tetszőleges karakter (beleértve a nulla darabot is, azaz az üres szót is, ami pusztán elméleti lehetőség, a korpuszokban nem fordul elő).

A 4.7. részben láttuk, hogy nagyobb regkif részeként milyen hasznos motívum a `.*`, mert a segítségével tudjuk például attribútumértékek elejét vagy végét megadni. Így önmagában viszont a tetszőleges szóra való keresést jelenti, ami az egymilliárd szavas MNSZ2 esetében elvben egymilliárd találatot eredményez, sorra a korpusz összes szavát, a gyakorlatban viszont a túlzott erőforrásigény miatt nem fog lefutni. Ez a lekérdezés az „ilyet ne” csoportba tartozik.

4.9. i) `[][][][]*`

A 4.6. részben láttuk, hogy a `[]` a „tetszőleges szó” megadásának egyik módja. (Azt is láttuk az imént, hogy a `".*"` szintén ezt jelenti. Valójában mindkettő a

```
[word=".*"]
```

lekérdezés egyszerűsített változata, az első a feltétel elhagyása révén, a második default attribútum használata révén.) A szimpla

```
[][][][]
```

lekérdezés is bőven az „ilyet ne” kategóriába esik, a 4.8. részben látott példához hasonlóan egymilliárd találatot eredményez, de a `*` segítségével még sokkal rosszabb lesz a helyzet. Ez az operátor az utolsó *token* többszörözését engedi meg, a lekérdezés ezáltal a korpusz legalább háromtokenes egységeire kérdez rá. Ennek megfelel az 1-2-3. token, az 1-2-3-4. token, az 1-2-3-4-5. token stb. egészen addig, hogy 1-2-3-...-1000000000. token, aztán a 2-3-4. token, a 2-3-4-5. token, a 2-3-4-5-6. szó – és így tovább. Ez az egymilliárd szavas korpusz esetén nagyságrendileg $10^9 \cdot 10^9 / 2 = 5 \cdot 10^{17}$ darab találatot eredményez, ami találatonként csupán 10 byte-tal számolva is 5 millió terabyte adat lenne. Ez a lekérdezés tehát az „ilyet ne” kategória minősített esete.

4.10. j) `[word = "elé"] [word = "a.?"] [word = ".+n(a|e)k"]`

Utolsó példánkban az *elé* szót követő határozott névelős *-nAk* ragos szóra keresünk. A névelőt meg lehetne fogalmazni a következő módon is:

```
[word = "az?"]
```

illetve, ha a határozatlan névelőt is hozzávesszük, akkor:

```
[word = "egy|az?"]
```

a *-nAk* ragos szót pedig így:

```
[word = ".+n[ae]k"]
```

esetleg igény szerint főnévre leszűkítve

```
[word = ".+n[ae]k" & msd="FN.*"]
```

– bár erre nem nagyon van szükség, mert névelő után egyébként is jó eséllyel főnevet (névszót) fogunk kapni.

Ez egy hasznos, értelmes lekérdezés, olyan, ami láthatólag egy konkrét nyelvi jelenségre igyekszik rákérdezni. Arról a jelenségről van szó, amikor az igekötő mintha egyben névutóként is funkcionálna, ráadásul a hozzá kapcsolódó *-nAk*-ragos szó előtt megjelenve.

Erre a ritka szerkezetre (konkrétan az *elé* igekötővel/névutóval) az MNSZ-ben is csak néhány példa van: *Hogy néz elé a tárgyalássorozatnak?* vagy *Nem éltek elé a gyerekeknek olyan élet és emberi méltóság iránti tiszteletet, amely beépült volna a lelkükbe*.

A lekérdezésre jóval több találatot kapunk, de ezek közül ki kell szűrnünk azokat, amelyek nem a kívánt jelenséget testesítik meg, hanem csak „véletlenül" kerültek a kívánt alakú szavak egymás mellé: *Kilin elé a rendelésnek megfelelően a három lágy tojás került és a gyógyvíz*. A jó példáknál általában ige előzi meg a háromelemű szókapcsolatot, érdemes lehet erre szűrni.

Végül megjegyezzük, hogy az ilyenfajta, több szóból álló kifejezéseket nagy eséllyel gyorsabb szűréssel keresni, mégpedig úgy, hogy először a legspecifikusabb szóra keresünk (*elé*), majd a találatokat szűrjük a következő legspecifikusabb szóval (itt a 2-es pozícióban, azaz a találati szótól kettővel jobbra elhelyezkedő *-nAk* ragos szóval), végül a harmadik elemmel.

5. Befejezés

Reméljük, hogy a fenti elemzések közelebb hozták az olvasóhoz a CQL-t és a reguláris kifejezéseket, és a jövőben bátran fogja alkalmazni ezeket az eszközöket korpuszokban való keresés során. Bízunk benne, hogy a korábbiakban megfogalmazott alapvetést (Sass 2017) hasznos tippekkel tudtuk kiegészíteni, és hozzájárultunk ahhoz, hogy az említett eszközök használata készséggé váljon. A jövőben elkerülhetők az alapvető, fent elemzett hibalehetőségek, és így gyorsabbá, hatékonyabbá válik a korpuszt feltáró, adatgyűjtő munka.

Irodalom

Sass B. 2016. Nyelvészeti szövegkeresők, Nemzeti Korpuszportál. *Magyar Tudomány* 177/7. 798–808.

Sass B. 2017. Keresés korpuszban: a kibővített Magyar történeti szövegtár új keresőfelülete. In: Forgács T., Németh M., Sinkovics B., (szerk.). *A nyelvtörténeti kutatások újabb eredményei IX.* Szeged: SZTE Magyar Nyelvészeti Tanszék. 267–277.

Rychlý, P. 2007. Manatee/Bonito – A modular corpus manager. In: *Proceedings of the 1st Workshop on Recent Advances in Slavonic Natural Language Processing*. Brno: Masaryk University. 65–70.

Oravecz Cs., Váradi T., Sass B. 2014. The Hungarian Gigaword Corpus. In: *Proceedings of the 9th International Conference on Language Resources and Evaluation (LREC2014)*. Reykjavík: European Language Resources Association.