A magyar hadifoglyok adatbázisának *orosz-magyar* transzkripciója

Nyelvtudományi Intézet – 2020. június-december

1. Feladat

A magyar hadifogolyadatbázis 682000 rekordot tartalmaz. A papírokat oroszok töltötték ki, ami azt jelenti, hogy minden adatát – az adat nyelvétől függetlenül – cirill betűkkel írták le.

A feladat alapvetően a magyar írott forma helyreállítása, pl.:

```
Ковач Йожеф -> Kovács József
```

A feladat nehézségét az adat sokrétű következetlensége adja.

2. Vezetői összefoglaló

Ahhoz, hogy képet kapjunk az eredményekről, nem szükséges futtatni semmit, csak vessük össze az alábbi két fájlt:

- eredeti bemenet: `data/Kart.csv`
- átírt kimenet: `out/Kart.transcribed.csv`

Ha a 682000 rekord túl soknak bizonyul, akkor tanulmányozhatunk egy 1000 és egy 10000 soros (8. rész) mintát is:

- eredeti bemenet: `data/Kart_1000_Sor.csv`
- átírt kimenet: `out/Kart_1000_Sor.transcribed.csv`
- eredeti bemenet: `data/random_10000_42.csv`
- átírt kimenet: `out/random_10000_42.transcribed.csv`

A fenti fájlok szenzitív adatokat tartalmaznak, így nem lehetnek részei a nyilvános repzotóriumnak. Az alábbi fájlok szabadon megtekinthető randomizált pszeudoadatot tartalmaznak:

- eredeti bemenet: `data/pseudo_1000_42.csv`
- átírt kimenet: `out/pseudo_1000_42.transcribed.csv`

3. Algoritmus

A használt szabályrendszereknek (7. rész) kétféle változata van:

- **strict** ekkor minden orosz betű(sorozat)nak pontosan egy magyar betű(sorozat) felel meg, pl.:

```
д d
```

- **loose** ekkor a magyar oldalon több megfelelő is lehet, pl.:

```
д d gy|t
```

Egy *eszközkészlet* 3 dologból áll. Két transzkriptor (`scripts/ru2hu.py`): egy **loose** és a hozzá tartozó **strict**, valamint egy **lista**, ami az elvárt értékeket tartalmazza (keresztnevek, országok stb.).

Előfeldolgozás (5. rész) után az algoritmus a következőképpen működik:

1. előkészítjük az adott mezőhöz tartozó eszközkészletet
2. a mezőben lévő szót átírjuk a **loose** transzkriptorral
3. így egy *regex*-et kapunk, ezt illesztjük a listára
4. ha így megvan, akkor visszaadjuk az összes találatot ==> **kész**
5. ha nincs meg, *közelítő kereséssel* keressük a **strict** átirat közelítéseit a listán
6. ha így megvan, visszaadjuk a legjobb találatot ==> **kész**
7. ha egyik módszer sem járt sikerrel, akkor visszaadjuk a **strict** átiratot ==> **kész**

Példák az algoritmus különböző kilépési pontjaira:

```
4. Андрош -> Andros -> (A|Á)(n|m)(d|gy|t)(r|l)(a|á|o|e)(s|sch) -> András
6. Форенц -> Forenc -> F(o|ó|ö|ő|a|á|ú)(r|l)(e|é|ö|ő|o|je|jé|...
   ...jo|jó|já|ye|yé|yó|yö|a)(n|m)(c|cz|cs|g) -> Forenc>>Ferenc
7. Момольсильтер -> ... -> Momolyszilyter
```

A 4. pontban a találatokat `;` -vel elválasztva adjuk vissza, több találat esetén kiegészítve egy pontszámmal, ami a találatok sorrendjét adja meg. Ez a pontszám a **strict** átirat és a találat `difflib.SequenceMatcher(...).ratio()` szerinti hasonlóságán valamint nevek esetében a név gyakoriságán alapul.

A 5. pontban a közelítő keresést a python `difflib` (11. rész) csomagja valósítja meg. Felmerülhet, hogy itt az 1 db **strict** alak helyett miért nem próbáljuk ki a regex-ből kigenerálható *összes* alakot. Ez nem megvalósítható, mert nagyon sok alak lehet. A `Шейкешфегервар` esetében például 974 millió a kigenerálható alakok száma.

A rendszer `python` -ban van megvalósítva. Linux rendszeren működik, használja a `bash` shell lehetőségeit. Ubuntu 18.04 rendszeren teszteltük, jó eséllyel más Linux(-like) rendszereken is működik.

A fő szkript a `scripts/transcribe.py` , az algoritmus

```
make transcribe
```

segítségével futtatható.

4. Az átírt adatbázis szerkezete

Az Access-ből exportált `.csv` adatbázisból (`data/Kart.csv`) indulunk ki, ebből hozzuk létre az átírt fájlt (`out/Kart.transcribed`).

Az azonosítót [0]-dik mezőnek véve a következő mezőket dolgozzuk fel:

- [1] vezetéknév
- [2] keresztnév
- [3] apai keresztnév

- [5] hely (lakcím)
- [6] hely (elfogás helye)
- [7] nemzetiség

A *hely* mezők – [5] és [6] – több szóból, elemből állnak: ország, megye, település…

Ezeket felbontottuk 4-4 (5. rész) (plusz 3-3 nem használt) mezőre, és így adtuk be az algoritmusnak (3. rész), melyet alapvetően egyes szavak kezelésére készítettünk fel. A felbontás miatt az adatoszlopok száma 12-vel (19-ről 31-re) nőtt, az eredeti és az átírt adatbázis oszlopai közötti megfeleltetés tehát a következő:

eredeti oszlop		átírt oszlop
[0]..[4]	=	[0]..[4]
[5]	=	[5]-[11]
[6]	=	[12]-[18]
[7]..[18]	=	[19]..[30]

Az átírt adatbázis oszlopcímkéit a `data/data.header.new.csv` fájlban találjuk.

5. Előfeldolgozás

5.1 Általános lépések

Az anyagból eltávolítottuk az orosz elöljárókat (pl. `в`, `около`, `под`) és végződéseket (`-ский/-ская`).

5.2 Szóalakok egyedi kezelése

Bizonyos szóalakokat az algoritmus nem tud átírni (3. rész, 7. pont). Ezek jelentős részét *egyedileg* kezeljük. Ez annyit jelent, hogy SAR (search and replace) táblázatok segítségével egyszerűen lecseréljük őket a megfelelő átírt alakra.

A táblázatok a `data/sar_tables` könyvtárban találhatók.

Az így átírt nevek `/R` jelölést kapnak. Ezeket az előre átírt neveket az algoritmus átugorja.

5.3 *név* mezők: [1], [2], [3]

Bár az adatbázis túlnyomó részben férfi keresztneveket tartalmaz, előfordulnak női keresztnevek is. A női neveknek a keresztnévlistához való egyszerű hozzávétele inkább ront az eredményen, mert így férfiak is női nevet kaphatnak (pl.: `Пауль -> Paula` `Матия -> Maja` `Гено -> Hana` `Алоис -> Aliz`), illetve adott adat mellett keveredhetnek a különböző nemű nevek (pl.: `Sándor;Santál`). Emiatt a nehezen feldolgozható férfinevek mellett az összes női nevet is egyedileg kezeljük. Példák: `Яню -> János;Jenő` `Лануш -> János;Lajos` `Дерди -> György;Györgyi`.

Az előfeldolgozás keretében elhagyjuk az apai keresztnév [3] mezőben előforduló, oroszban szokásos `-вич/-вна` végződést. A nevek tehát e végződés nélkül kerülnek az algoritmus bemenetére.

A név mezőkben előforduló másodlagos zárójeles alakokat figyelmen kívül hagyjuk. A pontot elhagyjuk a nevek végéről.

5.4 *hely* mezők: [5] és [6]

A *hely* mezőket egyenként 4 mezőre bontottuk (4. rész):

1. ország, 2. megye, 3. járás, 4. település

Az egyes elemeket különféle rövidítések alapján igyekeztünk beazonosítani, de ez az adat következetlensége miatt nem valósítható meg megbízhatóan.

Az előfeldolgozást a `make preprocess` valósítja meg.

6. A cellák tartalma

A kezelt mezőkben természetesen az adott szó átírt verzióját találjuk. A szó végén jelölve van, hogy az előfeldolgozás és az algoritmus (3. rész) hogyan bánt el vele. Az algoritmus kilépési pontjainak felel meg 1-1 jelölés, a következőképpen:

algoritmus lépés	jel
egyedi átírás (5.2 rész)	`/R`
4.	`/L`
6.	`/D`
7.	`=T`

Az első 3 kategória (ezekben közös a `/`) jelenti azokat, amikor valamilyen elvi módon eredményre jutottunk, az utolsó kategória azt jelenti, hogy pusztán a **strict** átiratot közöljük.

Az átírt adatbázis *tartalmazza* ezeket a jeleket. Ha ezektől mentes adatbázis akarunk, akkor egyszerűen törölhetjük a jeleket (ilyen szóvégződés nincs az eredeti adatbázisban!), ugyanez elérhető a `FLAGS="--plain"` kapcsoló révén:

```
make FLAGS="--plain" transcribe
```

Megjegyzendő, hogy a kiértékeléshez (10. rész) szükség van ezekre a jelekre.

7. Szabályrendszer

A **strict** és **loose** transzkriptorok szabályrendszerét *betűnként* 100-200 példa alapján manuális munkával állítottuk elő.

A két transzkriptor egy közös fájlban jelenik meg, ahogy fent már mutattuk a példát:

```
д d gy|t
```

Ez azt jelenti, hogy a `д` -nek a leggyakoribb megfelelője a `d` , de előfordul `gy` és `t` is. Ebből a leírásból generálódik a két transzkriptor: a **strict** csak a `d` -t engedi meg, a **loose** viszont a `{d, gy, t}` bármelyikét. Ld. pl.: `rules/ru2hu.rules` .

A rendszerben használatos szabályok, transzkriptorok és listák összességét a `rules/metarules.txt` fájlban találjuk. Ez vezérli a működés egészét. Ez a fájl a következő formájú sorokból áll:

```
1 ru2hu_loose ru2hu_strict vezeteknevek
```

Négy mezőt látunk: oszlopsorszám, két transzkriptor és a lista. A konkrét példa azt jelenti, hogy az $[1]$ oszlopban a `rules/ru2hu_loose.json` és `rules/ru2hu_strict.json` transzkriptorokat használjuk és a `data/lists/vezeteknevek.csv` listára igyekszünk illeszteni a szavakat.

Ez a formátum lehetőséget ad arra, hogy az egyes mezőkhöz különböző transzkriptorokat és különböző listákat használjunk, amire nagy szükség van.

A scriptek számára a `.rules` fájlokat és a `metarules.txt` -t is `json` formában adjuk át.

8. Adatfájlok, futtatás

A teljes adatfájlból (`data/Kart.csv`) 3 db 1000 soros részletet különítettünk el.

A `data/Kart_1000_Sor.csv` fájlt a rendszer kialakításához, a `data/test_set.csv` fájlt pedig a teszteléséhez használtuk.

A fenti fájlok szenzitív adatokat tartalmaznak, így nem lehetnek részei a nyilvános repzotóriumnak. A `data/pseudo_1000_42.csv` randomizált pszeudoadaton viszont szabadon vizsgálódhatunk.

A futtatás alapvető formája így néz ki:

```
make preparation ; make transcribe
```

Ekkor a `data/Kart_1000_Sor.csv` -ből előáll az `out/Kart_1000_Sor.transcribed.new.csv` átírt változat. A default `make transcribe` tehát a `data/Kart_1000_Sor.csv` fájlon dolgozik.

Explicit megadhatjuk, hogy az eljárás melyik fájlon dolgozzon:

```
make preparation ; make FILE=Kart transcribe
make preparation ; make FILE=Kart_1000_Sor transcribe
make preparation ; make FILE=test_set transcribe
make preparation ; make FILE=pseudo_1000_42 transcribe
make preparation ; make FILE=random_10000_42 transcribe
```

A teljes adatbázist feldolgozó

```
make preparation ; make FILE=Kart transcribe
```

(`data/Kart.csv` -> `out/Kart.transcribed.new.csv`) futási ideje – egy magon – *kb. 50-60 óra*! Egy rekord feldolgozása tehát kb. fél másodpercet igényel.

9. Segédlisták

A kapott listákat kis mértékben módosítottuk, kiegészítettük.

- Létrehoztunk egy megyelistát, ez légyegében a 64 régi + 19 mostani megye, kiegészítve azzal, hogy a többszavas megyék elemeit külön is fölvettük a listára, mert sokszor így jelennek meg (pl.: `Bács`).
- Egy jobb, sokkal bővebb vezetéknévlistával dolgozunk.
- Beszereztük a wikipediáról az osztrák településnevek teljes listáját (12. rész).

10. Kiértékelés

A rendszer az adatelemek » **80,5%** « -át kezeli.

Értsd: az adatelemek 80,5%-a esik az `{\R, \S, \L, \D}` kategóriák (6. rész) valamelyikébe, azaz a maradék 19,5% az, amire csupán sima **strict** átiratot adunk.

A kiértékelés a

```
make FILE=... eval_by_col
```

futtatásával történhet.

A fenti 80,5%-os értéket az `out/Kart.eval_by_col.out` elején látjuk.

Ez az érték két eltérő biztonsággal kezelhető csoport átlagaként alakul ki: az vezetéknév, (apai) keresztnév és ország mezők **95%-ban** kezelhetők, a többi mező pedig 40-60%-ban.

11. `difflib`

A `difflib` csomag által biztosított közelítő (approximative) keresés fontos elem, de vannak olyan esetek is amikor helytelen eredményt ad. Mivel mindenképp megpróbálja a lista elemeire ráilleszteni a szót, ha a szó *nem szerepel* a listában, akkor kapunk rossz eredményt.

Emiatt: *egyrészt* kikapcsolható ez a funkció (`FLAGS="--no-difflib"`), *másrészt* mindig a **strict** átírással együtt adjuk vissza, így:

```
Miklas>>Miklós/D
```

ahol `Miklas` a **strict** alak, a `Miklós` pedig a közelítő kereséssel meghatározott – ezúttal helyes – alak.

Az közelítő keresés kikapcsolásával 7x-es gyorsulás várható, az eredmény persze romlik.

A `difflib` *cutoff* értéke beállítható (`FLAGS="--difflib-cutoff 0.8"`). Ez minél magasabb, annál kisebb eltérést enged meg az eredeti és az illesztett szó között. Default esetben viszonylag magas küszöböt használunk (`0.8`), ennek köszönhetően, ami nagyon összevissza van írva, arra inkább nem mondunk semmit.

Példák, ahol segít a `difflib` :

```
v_Budapeste>>Budapest/D
Miklas>>Miklós/D
Szilyvesztr>>Szilveszter/D
Marostorda>>Maros-Torda/D
Csiskozmas>>Csikkozmás/D
```

Példák, ahol nem tudjuk, hogy segít-e:

```
Erdély>>Erdélyi/D
Abrok>>Ambro/D
Pejkes>>Petkes/D
Agneli>>Angeli/D
Abronics>>Aronovics/D
```

`cutoff_=0.7` esetén elég elvadult megoldások is előkerülnek:

```
Agosvari>>Agocsi/D
Vengerszkoe>>Vasegerszeg/D
Szegetazo>>Szigetor/D
Sotougen>>Struge/D
Poznany>>Pozsony/D
```

Egyébként jól mutatja a feleadat nehézségét, hogy bizonyos esetekben emberi intelligenciával is nagyon nehéz vagy lehetetlen kitalálni, hogy mi lehet a helyes átírás.

```
Вискемиш      -> Viszkemis
Улмерфеля     -> Ulmerfela
Кискухухого   -> Kiszkuhuhogo
Глисапешпет   -> Gliszapespet
Блодентмигайн -> Blodentmigajn
```

12. Az idegennyelvű szövegek kezelése

A második világháború vége körülményeiből adódóan számos magyar katona Ausztria területén esett fogságba. Emiatt sok településnév *németül* hangzott el, és úgy írták le az orosz írnokok cirill betűkkel.

Ebben az esetben az *orosz-magyar* átírás nem célravezető

```
Дойчландберг -> Dojcslandberg
```

(Ld.: `python3 scripts/simply_transcript_text.py rules/ru2hu_strict.json Дойчландберг`)

Ezért külön elkészítettük az *orosz-magyar* (`rules/ru2hu.rules`) mellett az *orosz-német* (`rules/ru2de.rules`) szabályrendszert is, aminek segítségével helyes átiratokat kapunk számos osztrák településre.

```
Гроц         -> Graz
Лииц         -> Linz
Фрайштадт    -> Freistadt
```

```
Дойчландберг -> Deutschlandsberg
Штокаров     -> Stockerau
```

Ehhez természetesen szükség volt osztrák településlistára, mely a wikipediáról beszerezhető volt (`data/lists/places_de_wikipedia.csv`).

Fontos azonban, hogy a német transzkriptort csak akkor szabad alkalmazni, amikor elég biztosak vagyunk benne, hogy az adott cellában német szót találunk. Ehhez megbízható támpont az *ország* – [5] és [12] (4. rész) – mezőben megjelenő `Австрия` .

A fentiek működtetéséhez kell egy új mechanizmus a `metarules.txt` -ben (7. rész). Ez így néz ki:

```
15 ru2hu_loose ru2hu_strict places
15/12=Австрия ru2de_loose ru2de_strict places_de
```

A [15]-ös (*település*) mezőre két sor vonatkozik. Az első a már ismert formátumú: alapesetben magyar transzkriptorokat és magyar helységlistát használunk. Kivéve, ha a [12]-es (*ország*) mezőben az szerepel, hogy `Австрия` . Erre utal a `/12=Австрия` jelölésmód. Ekkor elővesszük a német transzkriptorokat és helységlistát. Ezzel a megoldással elég rugalmasan meg tudjuk választani az épp szükséges eszközkészleteket.

2020.12.18. *v9* | 12.07. *v8* | 11.12. *v7* | 08.13. *v6* | 07.31. *v5* | 07.09. *v4* | 06.28. *v3* | 06.22. *v2* | 06.20. *v1*

Mittelholcz Iván (`Transcriptor` osztály)
Halász Dávid (helyek feldolgozása, részekre bontása)
Lipp Veronika (*orosz-magyar* szabályok)
Kalivoda Ágnes (*orosz-német* szabályok, gyakori nevek átírása)
Sass Bálint (főfő script, szervezés, satöbbi…)