EÖTVÖS LORÁND TUDOMÁNYEGYETEM

INFORMATIKAI KAR

TÉRKÉPTUDOMÁNYI ÉS GEOINFORMATIKAI TANSZÉK

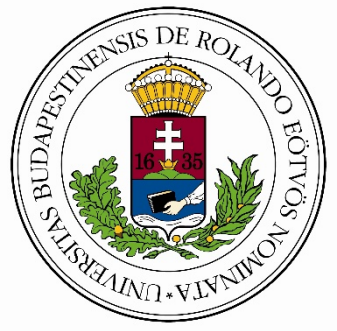
TÉRKÉPÉSZ MESTERSZAK

A geokódolás szerepe az adatbázis alapú digitális térképek világában. Geokódoló szolgáltatások.

DIPLOMAMUNKA

Készítette:
Horváth Attila
térképész hallgató

Témavezető:
Dr. Elek István
egyetemi docens
ELTE Térképtudományi és Geoinformatikai Tanszék



Budapest, 2017

Tartalomjegyzék

1. Bevezetés

2016 szeptemberében kaptam egy feladatot a munkahelyemen, az iData Kft-nél, amely GPS nyomkövetéssel, flottakövetéssel és járművédelemmel foglalkozik. A feladat úgy szólt, hogy a meglevő TomTom térkép adatbázisát kellene összeegyeztetni az OpenStreetMap adatbázisával, szükség esetén adatbázis-konverziót végrehajtani, annak érdekében, hogy a geokódolás pontosabb legyen a házszámokat illetően. Mivel a két térképrendszer felépítése közel sem hasonló, valamint az OpenStreetMap a kívánt célterületen (Magyarország, Szerbia, Románia) nem éri el a kielégítő minimum lefedettséget épületek, utak, beépített területek tekintetében, ezért a projekt egyelőre szünetel. A TomTom és az OpenStreetMap közötti differencia abból adódik, hogy az OpenStreetMap nem tartalmaz olyan információt, hogy egy adott házszámtól jobbra-balra milyen objektumok vagy házszámok helyezkednek el, míg a TomTom-nál külön tábla van erre a célra (könnyítve ezzel két sarokpont közti lineáris interpolációt) és külön a geokódolás elősegítésére. Másfelől nem tudni, hogy mennyire garantálja a pontosságot a nyílt és közösségi térképező OpenStreetMap, ezért úgy döntöttünk, hogy nem érdemes egyelőre belevágni ebbe. Ezt követően támadt az ötlet, hogy a geokódolás fellendítésével lehetne tovább menni és így jutottam el a szakdolgozat címének kitalálásáig. Dolgozatomban azt mutatom be, milyen folyamaton megy végig egy beírt szöveg a geokódolás esetében, és fordított geokódolásnál miket érdemes figyelembe venni, valamint hogyan lehet nyílt rendszerek felhasználásával térképhez megfelelő alapanyagot megteremteni. Továbbá a már meglevő térképekkel rendelkező szolgáltatók milyen geokódolásra alkalmas eszközöket használhatnak fel. A térképi adatbázis létrehozásához a PostgreSQL relációs adatbázis-kezelőt választottam, mivel egészen komoly múlttal rendelkezik, és remekül összhangban van a PostGIS-sel, amely a geometriák adatbázisban való tárolását biztosítja. A GIS szoftverek is ezen a kiegészítőn keresztül tudnak kommunikálni a térképi adatbázisokkal. Miután a nyers adat letöltésre került, a geometriák megfelelő formátumba való konverziójához több nyílt forráskódú eszköz is rendelkezésre áll (osm2pgsql, osm2po), amelyeket példákkal mutatok be. A TomTom és OpenStreetMap felépítésével kapcsolatban készítek egy kisebb ismertetőt (állományok kinézete, adatbázis táblák szerkezete és az OpenStreetMap adatbányász eszköze). Ezután térek át azokra a szolgáltatásokra, amelyek a geokódolást teszik lehetővé (PostgreSQL, ElasticSearch és Solr szövegbányászati eszközei), majd a fordított geokódolás nehézségeit mutatom be az iData Kft megoldásain keresztül (ami részben a TomTom térképeit használja fel). A dolgozat végén írok egy saját készítésű, különböző geokódoló API-kat felhasználó weboldalról. Két

olyan kutatómunkát részletezek, amely az ingyenes geokódoló API-k összehasonlításával foglalkozik.

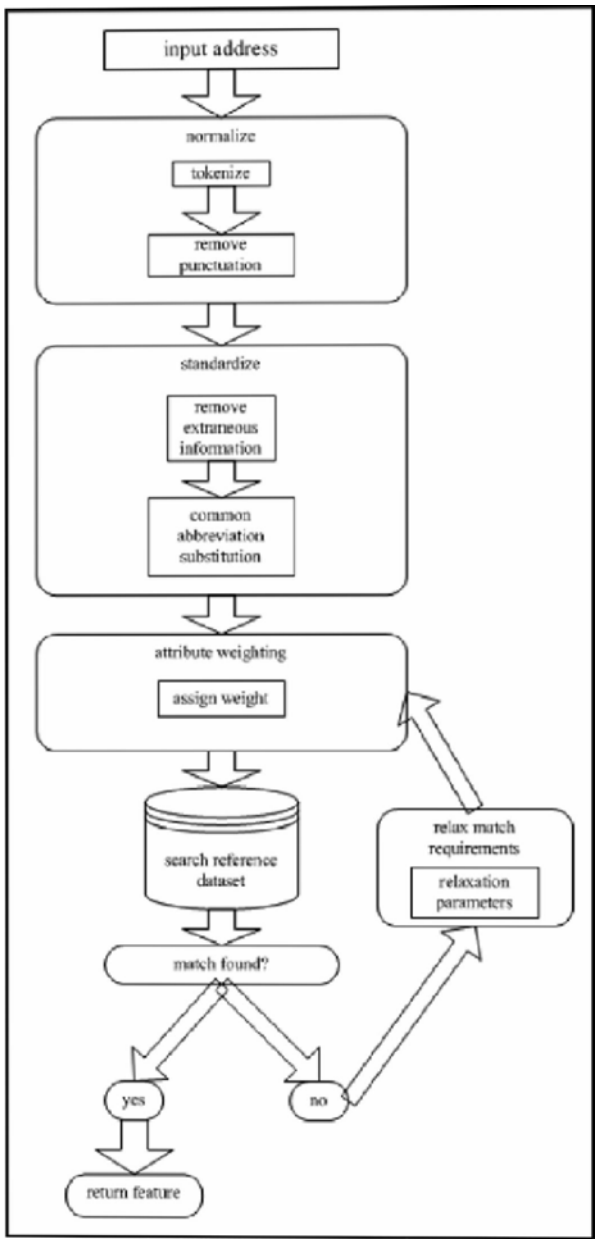
2. A digitális térképi adatbázisok helységnévtára, a geokódolás

A XXI. század rohamos technikai fejlődése az analóg, papír alapú térképek háttérbe szorítását eredményezi a digitális térképi adatbázisokkal szemben. Bizonyos nyomtatott térképek esetében fontos szerepet játszhat a terepi tájékozódás (például Budapest belvárosát ábrázoló térképnél), ami a névrajz hiányában igencsak nagy megpróbáltatást okozna a térkép olvasójának. A nevek, címek keresése, valamint egy adott ponthoz legközelebbi objektum felkutatása gyakran előfordul papírtérképek használatakor. Ezt a folyamatot segíti elő a névmutató, amely a névvel ellátott objektumok helyzetét jelöli a keresőháló koordinátáiban, ami megfeleltethető az analóg geokódolásnak. A digitális térképeknél használatos geokódolás alatt a leíró jellegű helyadatok, mint például postai címek vagy névvel ellátott területek abszolút földrajzi referenciába történő beillesztését értjük [Goldberg et al. 2007]. A geokódolást végrehajtó szoftver a geokóder, amelyet fel kell ruházni szövegbányászati (*full text search*) algoritmusokkal, annak érdekében, hogy a kérdésre (*query*) megfelelő választ adjon megfelelő nyelven. Ahhoz, hogy egy címet geokódolni lehessen, szükség van olyan referencia táblákra, amelyeket a kérésekkel össze lehet hasonlítani. A referencia táblák mindig, egy már előre elkészített térképi adatbázis részletei, tehát nevek mellett koordinátákat is tartalmaznak. Ha a beérkező kérés és a forrás adatbázisa között egyezés van, akkor az ide vonatkozó koordinátákat örökli a bemeneti cím, így meg is történt a geokódolás. Egy megbízható geokóder felépítéséhez a kulcs abban rejlik, hogy a rendelkezésre álló referencia táblák valamelyest konzisztensek legyenek, a keresés folyamata hogyan zajlik, valamint mekkora pontosságot tud biztosítani. Egy geokóder feltelepítése vagy megírása előtt fontolóra kell venni, hogy milyen pontosságot kíván a felhasználó vagy a célközönség vele elérni (például elég-e az irányítószámok szintjéig lemenni vagy szükség van esetleg utcára és házszámra is?).

Geokódolás folyamata

Amikor egy térképszolgáltató geokódere kap egy kérést, a rendelkezésre álló GIS adatbázis rekordjaival hasonlítja össze a bemenetet. Ilyenkor kritikus fontossággal bír a rövidítések, különleges karakterek eltávolítása, valamint az elírások kezelése a bemeneti adathalmazban. A kiegészítő információk külön mezőkben történő kezelése is elsőrangú feladat, mivel a keresés rovására mehet a túl nagy mennyiségű többlet adat. Helyes geokódoláshoz minimális előfeltétel a közterület fajtájának, nevének, valamint házszámának megléte, bizonyos esetekben utca iránya (USA-ban gyakori). Az irányítószám szerepe is nagy jelentőségű, viszont nem elvárás az egyezés felkutatásához. A beírt neveket, címeket nem egy az egyben keresi a geokóder, hanem szüksége van egy egységes adatszerkezetre, amelyet kialakít magának a bevitt

információból. A geokóder megpróbálja a cím részleteit a megfelelő adminisztratív hierarchia osztályaiba sorolni. Több internetes oldalt dedikáltak kifejezetten erre a célra, mint például a www.address-parser.net. Kezdetnek a szövegdobozba kell írni egy címet, majd megadni azt az országot, amelyben a keresés történjen. Az eredmény az 1. mellékleten látható táblázat, amely sztenderdizáltan (vesszők és rövidítéseket eltávolítva) mutatja a beírt címet. [Jeff, 2015]



1. ábra A geokódolás folyamata

Forrás: Goldberg, Daniel W.; Wilson, John P.; Knoblock, Craig A. (2007)

A kérés szövege az 1. ábra alapján átesik egy normalizálási, majd egy sztenderdizálási folyamaton, így a referencia adathalmazzal közös nevezőre tud jutni. Az algoritmusok kisbetűssé alakítják a bemeneti szöveget és eltávolítják az ékezeteket, extra információt. Előre definiált szótárak segítségével megkeresik a szótöveket, valamint kijavítják az apróbb, nem költséges elírásokat. A találatok különböző pontszámokat fognak kapni attól függően, hogy mennyire tartalmaznak releváns információt a szövegkörnyezettel kapcsolatban. Amelyiknek a címében található a kérés szövege, az magasabb értéket fog kapni, mint amelyiknek csak a megjegyzés mezőjében van egyezés. Tehát, ha van egyezés, akkor a súlyozott dokumentumokkal tér vissza a geokóder, ha pedig nincs, akkor kérhet további paramétereket a keresés pontosítása érdekében. Majd az attribútumokat újra súlyozza és veti össze a referencia adatállománnyal. Ami a referencia adatbázisokat illeti, általában több különböző forrásból szoktak származni a térképi adatok. Természetesen nincs két azonos pontosságú és struktúrájú felmérési állomány, valamint mind hibával terheltek. Ebből a nagy adathalmazból kell a geokódernek összeegyeztetnie a találatokat és egy értelmes szöveges dokumentum-listát kell válaszként adnia. Erre a problémára világított rá a Mapzen térképszolgáltató egy blog bejegyzésében,

miszerint 2016-ban a geokódolás okozta nehézségek közül az úgynevezett *record linkage* (magyarul rekordok egyeztetése) a legjelentősebb. Lényegében arról van szó, hogy olyan rekordokat kell keresni az adatállományban, amelyek különböző adatforrásból származnak, de ugyanazt az entitást írják le [Barrentine, 2016].

Inverz geokódolás

Bármely hosszúsági és szélességi koordinátával ellátott pont környezetében végrehajtott olyan keresés, amely földrajzi objektumok listáját adja vissza, fordított geokódolásnak nevezhető. Gyakorlatban kétfajta esetben van nagy szerepe ennek a folyamatnak. Egyrészről, amikor a felhasználó a mobil készülékén szeretné tudni, hogy adott időpillanatban épp hol tartózkodik, minek a közelében, másrészről pedig a nyomkövető szolgáltatók alkalmazzák gyakran ezt a módszert. A módszer lényege, hogy egy koordinátapárból a geokóder megpróbálja kiszámítani, hogy az mely településhez, irányítószámhoz, úthoz vagy házszámhoz van legközelebb. Egy fordított geokóder elkészítéséhez szükséges egy térképi adatbázis, melynek objektumai térbeli indexszel rendelkeznek. Kezdetnek megfelel bármilyen R-fával egyenértékű index, amely téglalapokban tárol egyszerre több objektumot, felgyorsítva így azok felkeresését. Célszerű egy tetszőleges sugarú körön belül végrehajtani a keresést, hogy ne kelljen az egész adatbázison végigszámolni ugyanazt a függvényt. Az erőforrásigényt lehet azzal is redukálni, hogy a teljes geometria helyett, minden objektum számára készül egy minimális befoglaló négyszög, így több száz vagy ezer vertex helyett csupán négyet kell figyelembe vennie objektumonként a geokódernek. A végeredményt persze befolyásolhatja például a GPS készülék által megállapított koordináta pontossága, valamint a térképi adatbázis koordinátáinak precízitása is. Az inverz geokódolás alkalmazását az 6. fejezetben mutatom be a TomTom Európa térképének felhasználásával.

3. Térképi adatbázishoz szükséges szoftverek alkalmazása

Egy térképi adatbázis felépítéséhez több komponensre is szükség van; először is kell egy forrás, ahol elérhetők a kívánt adatok valamilyen vektoros formában (shape) vagy XML (legtöbbször *.osm vagy *.pbf) fájlban. Jelenleg az OpenStreetMap a legnagyobb adattömeggel rendelkező felület, ahonnan teljesen ingyenesen letölthetők a szükséges állományok. Annak érdekében, hogy ne állandóan a fő szerver sávszélességét terheljék letöltéssel, a *Geofabrik.de* nevezetű tükörszerver biztosít elérést a vektoros adatokhoz (az egyes tartományoktól vagy megyéktől kezdve az egész világot lefedő állományokig). Egy másik ingyenes szerver a Mapzen nevű cég honlapja, ahol kifejezetten város-méretű területeket lehet elérni. Ha valaki ettől teljesen eltérő területre tart igényt és azon belül is csak kiemelt objektumra, akkor erre a célra a megfelelő oldal az *overpass-turbo.eu* (2-es és 3-as számú mellékletek mutatják. Példa arra, ha speciális adatokra van szükség, itt csak a budapesti kerületek határaira.). Az OpenStreetMap-en található adatok lekérdezésére specializálódott, kulcs-érték párok alapján működik, ami azt jelenti, hogy az egyes geometria típusok és attribútumai alapján lehet szűrést végezni, majd az eredményt *kml*, *gpx* és *geojson* formátumban engedi elmenteni a rendszer.

Az alapanyag megszerzését követően a feldolgozó szoftverekre van szükség. Számomra a legalkalmasabb adatbázis-kezelő a PostgreSQL, amit az enterprisedb.com weboldalról lehet letölteni Windows, Linux és MacOS operációs rendszerekre. A Linuxos csomagot tapasztalataim szerint nem kezeli megfelelően a rendszer, ezért azt célszerűbb parancssort futtatva telepíteni. Windowson szerencsére gördülékenyen fut a telepítő, a megfelelő kiegészítők és extrák (a PostGIS szolgáltatást mindenféleképpen érdemes integrálni a PostgreSQL-be, hiszen ez a modul felel a térképi geometriák adatbázisban való tárolhatóságáról) letöltését követően csupán az új, üres adatbázis felhasználóját és jelszavát kell megadni. Az adatok karbantartására a pgAdmin nevű programot mellékeli a telepítő, ezen keresztül a lekérdezés és az adatok adminisztrációja gyors és egyszerű.

PostgreSQL és PostGIS

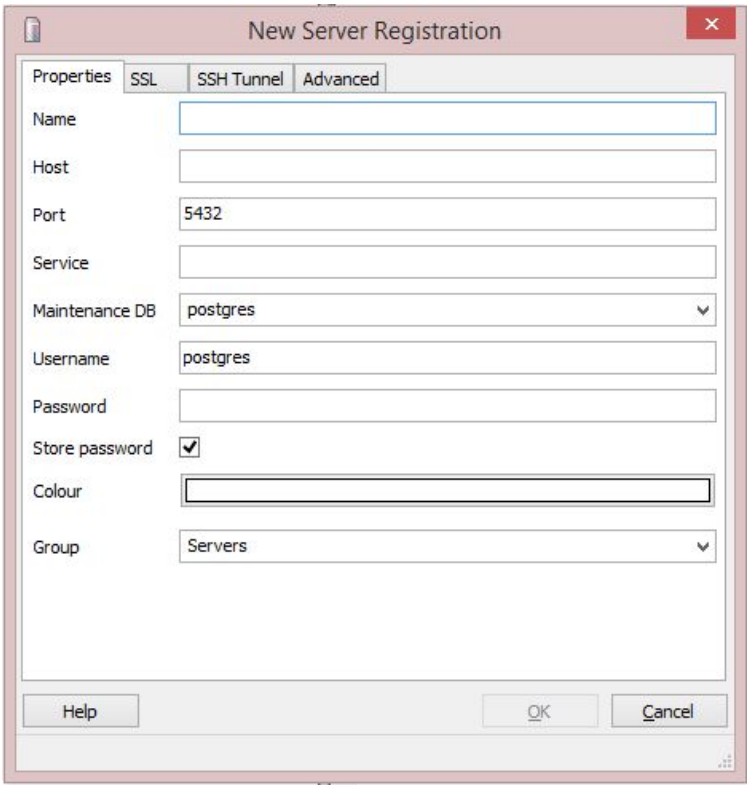
A PostgreSQL egy nyílt forráskódú, relációs adatbázis-kezelő rendszer, amely több, mint 15 éve aktív fejlesztés alatt áll. Ennek eredményeképpen a stabilitás és integritás szempontjából ez az egyik legmegbízhatóbb rendszer. A legtöbb ismert operációs rendszeren gond nélkül elfut, beleértve a Linuxot, Windows-t és az UNIX-alapú (MacOS) szoftvereket. A szoftver magában foglalja az SQL szabvány legtöbb adattípusát (INTEGER, NUMERIC, BOOLEAN, CHAR, VARCHAR, DATE, INTERVAL és TIMESTAMP), valamint lehetőséget nyújt nagyméretű bináris objektumok tárolására, beleértve képeket, zenéket és videókat. Natív programozási

felületet biztosít többek között C/C++, Java, .Net, Python nyelvekhez. A legfrissebb verziószám a februárban kiadott 9.6.2-es PostgreSQL. A rendszer bámulatos mennyiségű adat tárolását engedi, egy adattáblában szereplő mező maximum 1 GB lehet, a sorok egyenként az 1,6 TB-nyi adatot is elérhetik, egy egész tábla pedig maximum 32 TB-ot foglalhat. Emellett nincsen elméleti határa az adatbázis méreteit illetően. Ami a magyar nyelvűek szempontjából fontos lehet az az ISO 8859-2 (Latin2) karakterkészlet és az adatbázis lokalizációjának beállítása (hu_HU a lokalizációs kód). Ha a szerver és kliens karterkészlete eltérést mutat, a rendszer automatikusan elvégzi a konverziót. A PostgreSQL önmagában még nem teljesértékű GIS-célú felhasználásra, több beépülő modul és segédszoftver telepítésével lehet elérni az optimális működést. A két legfontosabb program a PostGIS és a pgAdmin. [EnterpriseDB]

A PostGIS egy nélkülözhetetlen kiegészítő eleme a PostgreSQL-nek, és valamennyi adatbázissal foglalkozó geoinformatikusnak. Ez a modul teszi lehetővé, hogy geometriai adatot lehessen tárolni adatbázis formájában. Az adattáblák rendezésével kapcsolatosan támogatja a GiST-alapú („*Generalized Search Tree*”, azaz egyszerűsített keresőfa) R-fa (térbeli objektumok indexelésére fejlesztett módszer) indexelést és biztosítja a geometriai objektumok térbeli analízisét és feldolgozását. A PostGIS-t a Refractions Research vállalat fejleszti Kanadában, amely része egy térbeli adatbázisok technológiájával foglalkozó kutató projektnek.

pgAdmin

A pgAdmin grafikus kezelőfelületet biztosít minden, PostgreSQL-ben tárolt adatbázis adminisztrációjához. Használata rendkívül egyszerű, három ablakból áll. A bal oldali ablakban találhatók a programmal összekapcsolt adatbázisok, ezek lenyitásával érhetők el a sémák és az adattáblák. A jobb felső szekció az éppen kijelölt objektum (táblázat, adatbázis, beépülő modul stb.) tulajdonságait részletezi, a jobb alsó ablak pedig a kijelölt objektum létrehozásához szükséges SQL parancsokat tartalmazza. Az eszköztáron található egy csatlakozó ikon, aminek megnyomásával lehet csatlakozni meglévő adatbázisokhoz. Az alábbi, 2. ábrán látható az a felugró ablak, amelyben meg kell adni az adatbázis elérhetőségeit, mint például hoszt, port, felhasználónév és jelszó.



2. ábra Szerver csatlakoztatása
Forrás: saját ábra

Ha SQL lekérdezéseket szeretne készíteni és futtatni az adatbázisban a megfelelő jogokkal rendelkező felhasználó, akkor két opció áll rendelkezésre: a manuálisan beírt SQL kód és a grafikusan létrehozható lekérdezés.

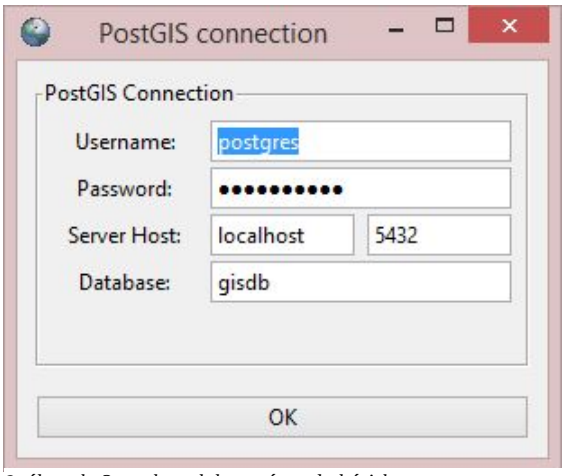
Új séma, majd üres adatbázis létrehozása után érdemes a fontos kiegészítőket hozzáadni a lekérdezés ablakban:

```
CREATE SCHEMA GIS
  AUTHORIZATION postgres;

CREATE DATABASE gisdb
  WITH OWNER = postgres
      ENCODING = 'UTF8'
      TABLESPACE = pg_default
      LC_COLLATE = 'Hungarian_Hungary.1250'
      LC_CTYPE = 'Hungarian_Hungary.1250'
      CONNECTION LIMIT = -1;

CREATE EXTENSION postgis;
CREATE EXTENSION unaccent;
CREATE EXTENSION pg_trgm;
CREATE EXTENSION ogr_fdw;
CREATE EXTENSION fuzzystrmatch;
```

Az új sémának tartalmaznia kell a szolgáltatót (jelen esetben postgres), az adatbázisnak a tulajdonost (ami most szintén postgres), a karakterkódolást, a lokális karaktercsomagot és a csatlakozásra fordítható időlimitet (mínusz eggyel nincs limit). Az importált kiterjesztések közül a legfontosabb a postgis, mert ez teszi lehetővé a geometriák tárolását az adatbázisban. Szorosan hozzá kapcsolódó plugin az *ogr_fdw*, amely biztosítja a kommunikációt GIS szoftverek és az adatbázis között. QGIS-ben gyakran előfordul, hogy térképet kell készíteni, de alapállomány például adatbázisban áll rendelkezésre, akkor az alapértelmezetten beépített *Add PostGIS layers* opcióval kapcsolódik a program PostgreSQL-hez. Hasonló itt is az eljárás, a hoszt, port, adatbázis, felhasználónév és jelszó megadását követően betöltenek a sémák és azon belül a táblák. Minden tábláról kap a felhasználó információt, hogy milyen geometria típusok szerepelnek bennük (*Spatial type*), milyen vetületben kerültek eltárolásra (*SRID*) és melyik oszlop tartalmazza a geometriák mezőit. A topológiai plugin rengeteg függvénnyel érkezik, amellyel a csomópontok és élek folytonosságát lehet vizsgálni és kezelni. Csak olyan táblákon alkalmazható, amelyek hézag és átfedés mentesen fedik le a teret. Az *unaccent*, *pg_trgm* és *fuzzystrmatch* kiegészítők szövegbányászati célból kerültek hozzáadásra, róluk az 5. fejezetben lesz szó bővebben.

OpenStreetMap adatait feldolgozó programok

Több módszer is létezik a geometriák elemzésére és adatbázisba illesztésére, ezek közül hármat emelnék ki. Az elsőszámú és legjobban felhasználóbarát megoldás, az *shp2pgsql*-re keresztelt, grafikus felhasználói felülettel (*GUI*-val) rendelkező program. Ezt a pgAdmin pluginként kezeli, így könnyen elérhető. Ahogyan a neve is sugallja, a shape állományokat alakítja át az adatbázis-kezelő számára emészthető formátumúra. A program egyszerű és érthető felületen keresztül kínálja fel az adatbázishoz való csatlakozást.



3. ábra shp2pgsql csatlakoztatása adatbázishoz
Forrás: saját ábra

A 3. ábrán látható a *View connection details* opció, amely biztosítja a program és a PostGIS modul közti kommunikációt. Az importálni kívánt állományok kiválasztása után meg lehet adni például egy új séma és azon belül egy tábla nevét, amibe a program betölti az adatokat. Fontos, hogy az *SRID* oszlop ne maradjon üresen, mert a program ez alapján tudja, hogy milyen vetületbe kell helyezni a geometriát. Itt az

úgynevezett EPSG kód alapján történik a beazonosítás, ami az EOV esetén 23700, a WGS84 esetén pedig 4326. Gyakori a webtérképek alkalmazásában a 3857-es kód, ez pedig a *Pseudo Mercator* egyedi azonosítója. Már csak egy fontos része maradt az importálás előkészítéséhez, az *Options* menü alatt található beállítások. A felső szövegdobozban a karakterkódolás azonosítója szerepel. Magyar névanyaggal rendelkező állományt illetően az UTF-8 készlete megfelelő szokott lenni az esetek 99%-ban. Mindemellett van két alapértelmezetten bejelölt sor, az egyik a COPY eljárást hajtja végre az INSERT helyett, a másik, ami lényegesebb, hogy az importálás befejeztével automatikusan létrehoz térbeli indexelést a táblákban. Import opción kívül export lehetőségét is felkínálja a program, ilyenkor a meglévő adattáblák közül a geometriával rendelkezők mentésére van mód.

A második és harmadik számú program megjelenésükben és működésükben nagyon hasonlítanak egymásra. Mindkettő kizárólag parancssorból indul és különböző kapcsolók segítségével lehet megadni az adatbázishoz való csatlakozást és a feldolgozandó fájl elérési útját. Az említett két program neve: *osm2pgsql* és *osm2po*. Ezen alkalmazásokat akkor érdemes futtatni, ha XML típusú állomány birtokában van az ember. A korábban említett *geofabrik.de* weboldalról letölthető *pbf/osm* kiterjesztésű fájlok megfelelőek a feladatra. Az osm2pgsql tulajdonképpen egy olyan eszköz, amely OpenStreetMap-en levő adatot tölt be PostGIS-sel rendelkező PostgreSQL adatbázisba. Felhasználási területe lehet egyszerűen egy térkép betöltése, geokódolás az OpenStreetMap-hez készült *Nominatim* geokódoló segítségével, nem utolsó sorban pedig térképi analízishez is megfelelő. A program a következő személyre szabási lehetőségeket nyújtja:

- a mellékelt stílusdefiníciós fájlban a címkék (tag-ek) soronkénti konfigurálása
- tömörített fájlok (.gz, .bz2, .pbf és .osm) közvetlen olvasása
- adatbázis frissítésénél csak a differencia alkalmazása
- vetületek kiválasztása
- egyéni tábla nevek
- helységnévtár biztosítása az OSM-Nominatim geokódoló számára.

A legtöbb Linux disztribúciónál alapértelmezett program, így telepítésre nincs szükség. Windowson kevésbé barátságos a helyzet, mert a 64-bites kiadás óta eltört a kód és sok kutatás után találtam működő verziót. Több lépésből áll a telepítés, mert a PostgreSQL csomag nem tartalmaz fontos komponenseket. Mindenekelőtt magát a programot kell letölteni, majd a hozzá tartozó *PROJ* könyvtárat, amely a vetületek definícióit tartalmazza. A könyvtárat célszerű a „C” meghajtó gyökerébe másolni a könnyű elérés céljából. Az alapprogram tökéletes

működéséhez elengedhetetlen a stílus fájlt is külön kell beszerezni, mert ismeretlen okból kifolyólag nincsen benne alapértelmezettként. Ha ez a három összetevő rendelkezésre áll a számítógépen, akkor parancssorban futtatva a következőképpen lehet működtetni:

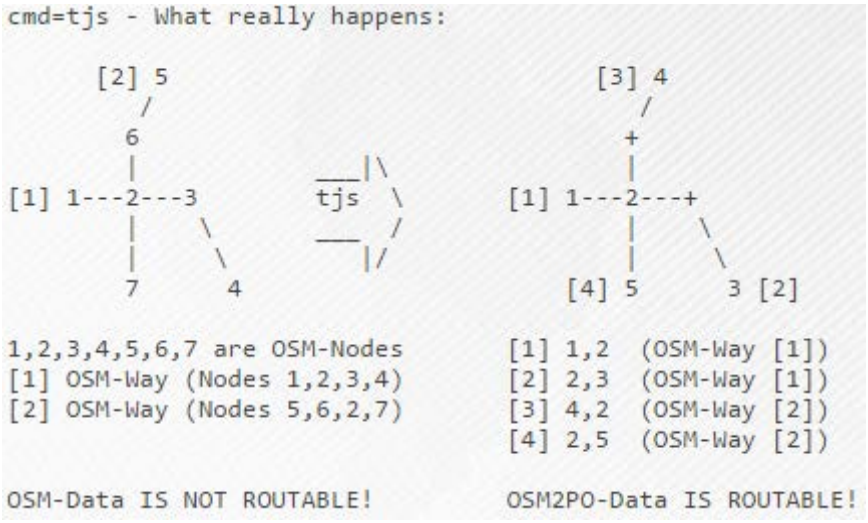
```
osm2pgsql -S C:\default.style -l -c -d <adatbázis neve> -U <postgres felhasználó> -H <hoszt> <OSM fájl elérési útja>
```

A program a paraméterek megadását követően beolvassa az egész fájlt és a stílus fájl alapján kiválasztja a megfelelő címkéket tartalmazó geometriát, majd azonnal importálja az adatbázisba. A feldolgozási idő függ a számítógép teljesítményétől és az adat mennyiségétől. Példaként az országunkat lefedő *hungary-latest.osm.pbf* fájl kiindulási helyzetben 128 megabájtot foglalt. Egy négymagos, Intel i5-ös processzorral és 8GB RAM-mal felszerelt gép három és fél perc alatt dolgozta fel és illesztette be az adatbázisba a fájlt. [Osm2pgsql]

Az osm2po program annyiban különbözik az előzőtől, hogy fel tudja dolgozni az állományt úgy, hogy az később alkalmas legyen útvonaltervezésre, geokódolásra és távoli elérést is biztosít megfelelő beállításokat követően. Itt egy komplett csomagról van szó, ami Java alapú, tehát minden Java 6-tal vagy annál újabb verzióval rendelkező számítógépen ugyanúgy működik és nincs szükség kiegészítőkre különböző weboldalakról. Létezik egy Android operációs rendszerre fejlesztett verziója is, amely internet kapcsolat nélküli navigációt tesz lehetővé. Jelenleg német és angol nyelven működik a beszéddel való navigálás. A program szintaxisa a következő:

```
java <-Xmx> -jar osm2po.jar <paraméter=érték> <bemeneti fájl>
```

A *–Xmx* paraméter mutatja, hogy a gépen mekkora memóriát használhat fel a program (2 gigabájtnál például *–Xmx2g*). A *paraméter=érték* páros felel a konverzió típusáért. A paraméter az mindig *cmd* (command, mint utasítás), az *érték* pedig több tagból állhat. Az én esetemben *tjsp*, ami azt jelenti, hogy feldolgozza a bemeneti fájlt, majd a pontokat, utakat és relációkat összeilleszti, ezt követően kialakítja a topológiát és végül egy utófeldolgozási folyamat során egy SQL fájlt hoz létre, amivel be lehet illeszteni a kapott eredmény adatbázisba. A navigálhatóságot az 4. ábra alapján készíti el a program. [Carsten, 2016]



4. ábra Útvonaltervezés opció
Forrás: www.osm2po.de

Az alapállomány útjai mind különálló geometriák, tehát akármennyi csomópontot tartalmazhatnak. A program megkeresi a csomópontokat és szétvágja azokat, majd mindegyik újonnan keletkezett útvonal szegmens kezdeti és végpontjának ad azonosítót. Ha egy szegmens kezdeti ID-ja például 130 és ahhoz kettő, topológiailag megegyező szegmens csatlakozik, akkor azoknak a csatlakozó pontok is a 130-as ID-t fogják viselni. Ez igaz az említett szegmens végpontjának azonosítójára is, ha az például a 129-es, akkor az összes abban csatlakozó, topológiailag azonos tulajdonságú útrészlet azonosítója (függetlenül az irányultságtól) is 129-es lesz. A végeredményként pedig sok *.2po és egy *.sql fájl várható. Az utóbbi egy olyan szkript, amely tartalmazza az összes utasítást és adatot az adatbázisba történő beillesztéshez.
Ha parancsként megadom, hogy *cmd=tjspgr*, akkor a feldolgozást követően elindít egy webes szolgáltatást, amely alapértelmezetten támogatja az útvonaltervezést és a geokódolást is. A felület sztenderd elérési útja a *http://localhost:8888/Osm2poService*, amely OpenLayers 2-ben készült el. Több alaptérkép közül is lehet választani és fedőrétegként aktiválható az SRTM domborzatárnyékolás. Az alsó sávon található opciók lehetővé teszik, hogy két vagy több pont közti útvonal-generálást igény szerint testre szabjuk. A megjelenő útvonalkereső típusok a konfigurációs fájl beállításaitól függenek, alapértelmezetten csak egy látható. A térkép közepén felbukkanó fekete pont mutatja a középpontot, amely egy kívánt kiindulási hely fölé húzva pirosan, majd a célállomás felett zölden látható. Az oldal geokódolást is engedélyez, sajnos csak az útvonaltervezéshez szükséges objektumok kereshetők, külön épületek és poligonok nem, de egy-egy út vagy utca felkeresésére éppen megfelelő. [osm2po dokumentáció]

4. Az OpenStreetMap adatszerkezete és geokódolói

Az OpenStreetMap egy ingyenes, nyílt forráskódú, közösségi térképszerkesztő és megjelenítő rendszer. Az OpenStreetMap adatait három entitás-típus építi fel: *node, way* és *relation* (csomópont, vonal/út és kapcsolat). Valamennyi típus egyedi azonosítóval (ID) és extra információval (szerkesztő és verzió mező) rendelkezik. Minden *node*-nak van egy szélességi és egy hosszúsági koordinátája, belőlük épülnek fel az OSM-es modell pontjai. A *way*-ek adott két *node* között mindig vonalakat képeznek és irányultsággal bírnak, míg a *relation*-ök egyszerre hivatkozhatnak *node*-ra és *way*-re is. Ebből adódóan kialakulhatnak komplex geometriai formák, mint például egy lyukas poligon, amely centroidjában egy pont áll, így jelezve a térképen a helyzetét. Ez a topologikus modell teszi lehetővé a redundancia csökkenését, ugyanis egy *way* része lehet egyben egy útnak, amely megyehatárként is funkcionál, mindemellett egy város szélét is jelöli. A térképen egy vonalként fog megjelenni és egyszerre több szerepet fog betölteni. Ami pedig az attribútumokat illeti, mindhárom típust rengeteg címkével (*tag*) fel lehet ruházni. Ezek a *key-value* (kulcs-érték) típusú paraméterek adják meg az objektumok neveit, kategóriáit, címüket. Az adatok struktúrája miatt nem egyszerű az indexelés és a feldolgozás, ennek a megoldására készült el az úgynevezett Nominatim.

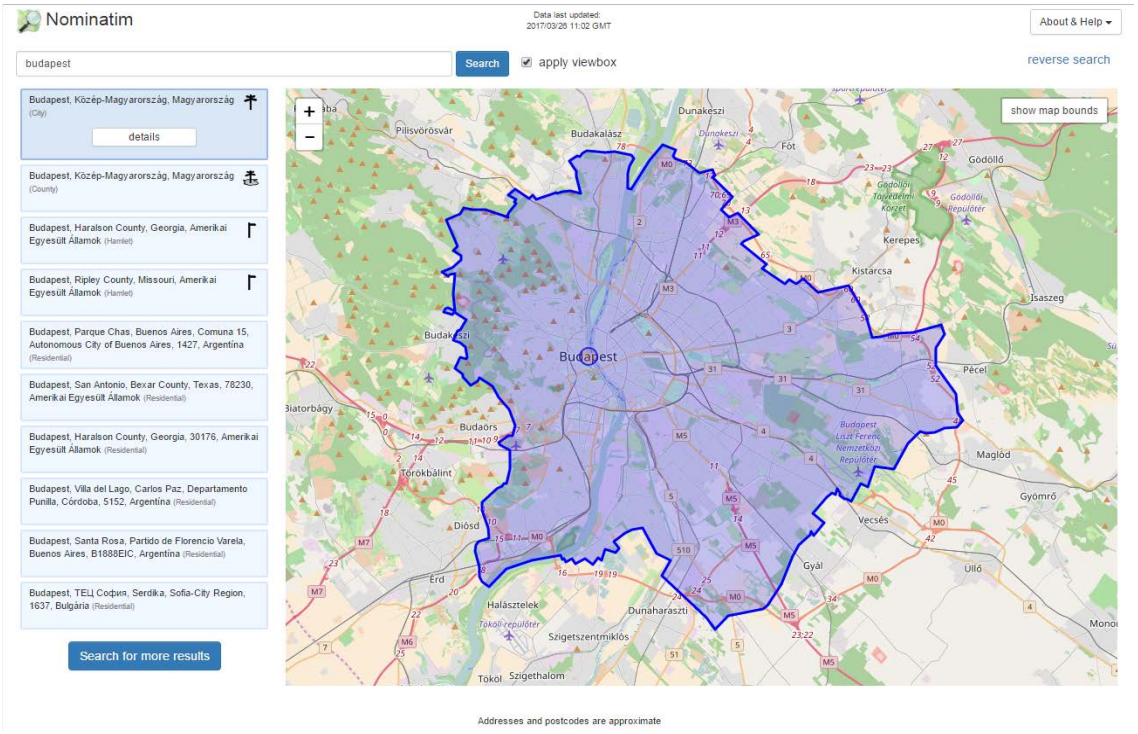
Nominatim

A Nominatim egy több programnyelven megírt, PostGIS-sel kiegészített PostgreSQL adatbázison alapuló kereső szolgáltatás, amely az OpenStreetMap-en levő adatokban (az USA-ban a TIGER geokódolót használja segítségként) név és cím alapján keres, valamint szélességi és hosszúsági koordinátapár alapján keresett ponthoz nevet és címet rendel (inverz geokódolás). Neve a latin nyelvből jön, jelentése: „név alapján”. Az OpenStreetMap-en található keresődobozba írt kifejezéseket dolgozza fel, valamint saját webhellyel is rendelkezik, amely elérhető a *nominatim.openstreetmap.org* oldalon. Saját API-val is rendelkezik, amelynek adatai körülbelül 20 percenként folyamatosan frissülnek, valamint a program a keresések közben is végzi az indexelést, ezzel is hatékonyabbá és gyorsabbá téve a keresést. Több cég integrálta a Nominatim szolgáltatását geokódolás céljából, mint például a MapQuest, PickPoint és a LocationIQ. Az OpenStreetMap honlapján található egy ismertető a Nominatim telepítéséről lépésről-lépésre, így saját felhasználásra is alkalmazható a szolgáltatás. Ha a felhasználásból jövedelem is származik, az OpenStreetMap szerverét nem lehet használni a magas terhelés

veszélye miatt. Erre az a megoldás, hogy az adatokat letöltve, saját szerverről kell biztosítani (a Geofabrik megoldást nyújt erre a lehetőségre).

Nominatim kinézete, működése

A weboldal kinézete nagyon egyszerű, letisztult és lényegre törő. Kezdésként címre, névre lehet keresni, de oldalt a *reverse search* opció alatt koordináták beírásával fordított geokódolásra is van mód. Egérkattintással is engedélyezett a keresés, ilyenkor a felület pirosas körrel jelzi a kattintás helyét, majd kékkel a hozzá legközelebb levő, geokódolt címet. A koordinátákát megadása mellett egy legördülő menü biztosítja, hogy melyik nagyítási szinten történjen a keresés. A kontinensektől egészen az épületek részletességi szintjéig engedi az oldal a választást. A zoom szintek 0-19-ig terjednek, minden egyes lépcső más részletességgel bír. Ezt más néven *LoD* (Level of Detail) szinteknek is szokták hívni. Tulajdonképpen a teljes vektoros állományból 20 különféle térképet készítettek el, mindegyiknek saját stílust és részletességet adva. Ezeket a vektoros térképeket végül raszteres formátumban tárolják, legtöbbször *png* típusként, mivel jóval gyorsabb kirajzolást garantál (kevesebb erőforrás szükséges a raszteres adat megjelenítéséhez), cserébe nagyobb lemezterületet igényel.



5. ábra Nominatim kinézete
Forrás: www.nominatim.osm.org

Ahogyan az 5. ábra is mutatja, a találatok a bal oldali függőleges sávban helyezkednek el. A *details* menüpont adja meg a geokódolt eredmény részleteit. Itt láthatók az általános információk (név, típus, centroid koordinátája) mellett a címkék (*tag-ek*) kulcs-érték párosai is,

valamint a geokódolt objektum gyerek objektumai. A keresendő kifejezések lehetnek kis- és nagybetűsek, valamint az egyes tagok felcserélhetőek (ugyanazt az eredményt fogja adni Budapest, XI. Pázmány Péter sétány 1/A-ra, mint a Pázmány Péter sétány 1/A, XI. Budapestre). A geokódolási találatok egy fontossági skála alapján rendeződnek. Ez az *importance* mutató, amely több tényezőtől is függ, elsősorban a különböző attribútumok szövegkörnyezettel való relevanciájától, majd az adott objektum Wikipédián elfoglalt helyezésétől és népszerűségétől, valamint a *rank* (helyezés) attribútum értékétől (településekre keresésnél a populációt is vizsgálja a geokóder). A *rank* értéket bonyolult algoritmusok számítják a geometria minősége alapján. Az algoritmus megvizsgálja, hogy a beírt kifejezés egy városnak, falunak, országnak, kontinensnek, főútnak feleltethető-e meg. Mindegyik típusnak meg van a maga helyezése, amely a Nominatim dokumentációjában található. A fontossági sorrendet az alább látható, 1. számú táblázat mutatja be.

For administrative boundaries: admin_level * 2	4–22
Continent, sea	2
Country	4
State	8
Region	10
County	12
City	16
Island, town, moor, waterways	17
Village, hamlet, municipality, district, borough, airport, national park	18
Suburb, croft, subdivision, farm, locality, islet	20
Hall of residence, neighbourhood, housing estate, landuse (polygon only)	22
Airport, street, road	26
Paths, cycleways, service roads, etc.	27
House, building	28
Postcode	11–25 (depends on country)
Other	30

1. táblázat Az objektumok hierarchiája
Forrás:www.wiki.openstreetmap.org

Az OpenStreetMap leírásában az szerepel, hogy 0 és 30 közé esik a létező összes típus, 0 a legfontosabb és 30 a legkevésbé számottevő egy geokódolási eredménnyel kapcsolatban. Némely típusoknak nem csak egy helyezési számuk van, hanem egy tól-ig intervallumuk. Jó példa erre az utolsó előtti sor, ami az irányítószámokra vonatkozik. Önmagában egy irányítószám nem mindig elégséges feltétele a geokódolásnak, pontosítani kell legalább az ország nevével. Mivel minden ország kapott egy helyezést az OpenStreetMap-en, ezért az irányítószám fontossága ennek függvényében változhat. Amikor egy bemeneti szöveget kap a

geokóder és kilistázza a találatokat, akkor nem csak a beírt kifejezés látható, mint adat, hanem az, hogy az adott objektum mely nagyobb poligonokba esik bele, tehát a kisebb geometria szülői is. Minden objektumnak, amelyek a 26-os szint felett helyezkednek el (utca szint), háromféleképpen kalkulálja a Nominatim a szülőjét:

1. Minden poligont kilistáz, amely területébe a kisebb beleesik nagyság szerinti sorrendben
2. Valamennyi elemet megmutat, amely az *is_in* attribútumban név szerint tartalmazza a keresendő kifejezést
3. Minden magasabb rangú objektumot megkeres, valamint azokat is, amelyek az adott objektum legközelebbi szomszédjánál (nearest neighbor) másfélszer messzebb vannak (távolsági sorrend szerint).

Azok az elemek, amelyek a fent említett rang intervallumon kívül esnek, legnagyobb számban az épületek. Indexelésük a hozzájuk leginkább kapcsolható és legközelebb álló út, utca alapján történik. A megfelelő út vagy utca kiválasztása a következő módon alakul:

1. A program megvizsgálja az *associatedStreet* kapcsolatokat (ház és utca közti kapcsolatot tartalmazó címke)
2. Ha egy *node* (csomópont) egy út része és az út szintje 26 és 30 közé esik, akkor ez lesz az út
3. Az *addr:street* attribútum vizsgálata
4. A legközelebbi út alapján a nearest neighbour algoritmus segítségével (általában három mérföldön belüli küszöbértékkel, ami körülbelül öt kilométer)
5. Nincs összekapcsolás egy úttal vagy utcával sem.

Az épületek és utak rendelkezésre állását a *www.osm-analytics.org* oldalon lehet nyomon követni. Az objektumok sűrűségét a sárga négyzetek jelölik egy rátét vektoros réteg segítségével. Minél több az épületek vagy utak száma, annál jobban összefügg a sárga négyzetek sokasága (egyszerre csak egy típust lehet nézni, vagy az épületeket vagy az utakat). Országokra és régiókra szűkül a geokódolási lehetőség, és az eredmény mindig egy erősen generalizált poligon lesz. A kapott poligonon belül található objektumok számát és azok időbeli változását egy grafikonon jelzi az oldal az alsó sávban. Így nyomon követhető, hogy melyik

évben mennyivel bővült az utak és épületek száma. A weboldal még csak béta verzióban érhető el, így ez nem okoz nagy gondot, mert arra a célra bőven megfelelő, hogy érdemes-e az OpenStreetMap adatait használni egy adott területen vagy sem. Jól látszik, hogy leginkább a fejlett országokban érhetők el a legnagyobb számban az épületek (leginkább Nyugat-Európában), míg az utak az USA területén is szép számban vannak jelen. Az épületekről és utakról készítettem egy kimutatást, amely megtekinthető az 4. és 5. mellékleten. [OpenStreetMap Contributors]

Overpass-turbo

Geokódolni az OpenStreetMap-en nem csak cím, hanem a különböző kulcs-érték (*key=value)* típusok alapján is lehetséges. A dokumentáció részletesen mutatja be közel az összes címkét és azok esetleges értékeit. Az overpass-turbo kifejezetten erre a fajta keresésre összpontosítva teszi lehetővé az adatbányászatot webes felületen keresztül (elérhető a www.overpass-turbo.eu oldalon). Bármilyen overpass API-val (*Application Programming Interface,* vagyis alkalmazásprogramozási felület) kompatibilis lekérdezést futtat, majd az eredményt egy interaktív térképen jeleníti meg. A webes felületet rendeltetés szempontjából két nagy csoportra lehet bontani: térképezés és fejlesztés típusú használat. Térképezési szempontból a különböző nyelvi hibákat tartalmazó objektumok vizsgálata és az elírások javítása szükséges. Bizonyos körülmények között hasznos, ha egy objektumnak több nyelvi alakja is szerepel az attribútumokban. Ennek megfelelően például a Szlovákiában levő, magyar névvel rendelkező folyók neveit egyetlen lekérdezéssel ki lehet listázni és megjeleníteni. Alapértelmezetten az overpass API az éppen látható térképen (ez a *bounding box*) keres egyezést a beírtaknak megfelelően. De létezik olyan opció is, amellyel meg lehet adni egy geokódolandó területet és ezen belül fogja végrehajtani a program a pásztázást. Minden, ami a lekérdezés feltételeinek eleget tesz, az felkerül eredményként a térképre, külön stílussal ellátva. A másik felhasználási terület a fejlesztés céljából történő lekérdezés, amikor egy saját térképre van igény, de azzal a kikötéssel például, hogy az európai országhatárokon és úthálózatokon kívül más ne jelenjen meg. Persze felmerülhet a kérdés, hogy miért nem a Geofabrik-ról töltik le az egész Európát tartalmazó OSM fájlt és importálják adatbázisba a megfelelő objektumokat? Ez is egy módszer, csak több adatforgalommal és lényegesen nagyobb tárhelyet igényel. [Martin, 2012]

Az Overpass-turbo felfogható egy már meglévő adatbázisban szereplő objektumok adatbányászati eszközeként. Helyenként hasonlít az SQL nyelvre, mert itt is feltételekkel (AND, OR, IN) lehet szűkíteni a találatok számát. Az Overpass-turbo-val a lekérdezéseken kívül a megjelenítést is lehet manipulálni. A MapCSS nyelvnek köszönhetően tematikus térképekre nagyon hasonlító művek is készíthetők. A nyelv szintaxisa a következőképpen írható fel:

```
[out:json][timeout:25]; //kimeneti fájlformátum és időkorlát
// itt kezdődik a lekérdezés
(
  node[ "highway" = "motorway" ]({{bbox}});  //key-value párok
  way[ "highway" = "motorway" ] ({{bbox}}); //mekkora területen történjen a keresés
  relation[ "highway" = "motorway" ] ({{bbox}});
);
out body; >;    // végeredmény kiírása
out skel qt;
{{style:
way[highway=motorway]
{ color:#ea8c75; width:10; }
}}
```

A lekérdezés eredménye a képernyőn megjelenő (*bbox*, azaz *bounding box*) összes autópálya 10 pixel szélesen és bordó színnel. Ez a kérdés kihat mind a három adattípusra, tehát a pontokra, vonalakra és kapcsolatokra. Az 6. mellékletnek megfelelően készítettem egy olyan tematikus jellegű lekérdezést, amely kirajzolja a Magyarországon (*geocodeArea*) levő, *place=city* tulajdonsággal rendelkező településeket, majd a lakosságszám függvényében jeleníti meg őket arányosan egy halványkék jellel. Alaptérképnek a *Stamen toner* stílust alkalmaztam (fekete-fehér térkép), mivel ez nagy kontrasztot mutat a térkép és a kiemelt geometriák között. A lekérdezés:

```
[out:json][timeout:25];
{{geocodeArea:hungary}}->.searchArea;
(
  node["place"="city"](area.searchArea);
  way["place"="city"] (area.searchArea);
  relation["place"="city"](area.searchArea);
  );
out body; >;
out skel qt;
{{style:
node {color:black; opacity:0.2; width:2; fill-color:blue; fill-opacity:0.4; symbol-shape: circle;
  symbol-size: eval('max(3,min(200,0.04*sqrt(tag("population"))))');}
relation {color: white; opacity: 0.1; fill-color: white; fill-opacity: 0.1;}
}}
```

5. Geokódolásnál alkalmazott keresőmotorok

A legtöbb cégnél, amelyek webes alapon szolgáltatnak térképet, megtalálható a keresésre utaló nagyító. Az esetek legnagyobb százalékában ugyanis nem pusztán csodálkozás és nézelődés céljából keresik fel az emberek az internetes térképszolgáltatók oldalait, hanem egy ismeretlen cím helye vagy ismeretlen hely címére kíváncsiak (geokódolás és fordított geokódolás). A geokódoláshoz, amely egy szöveges adat felruházása földrajzi koordinátával, szükség van a térképes adatbázison kívül egy olyan keresőmotorra, amely a beírt címet kutatja fel az adatbázisban és rendeli hozzá egy hosszúsági és szélességi pont koordinátájához (természetesen lehetőség van bizonyos szolgáltatóknál nem csak pont típusú objektumra keresni, hanem vonalas és felületi elemre is, amely később megjelenítésre kerül a térképen, ilyen például a Nominatim). A dolgozatban három eszközt fogok bemutatni, amelyek alkalmasnak bizonyulnak a geokódolás feladatára. Valamennyi alkalmazás az úgynevezett *full-text-search* (teljes szöveges keresés) technikát alkalmazza, mivel cím keresése egyenlő egy olyan szöveggel (*string*-gel), amely normalizáláson, sztenderdizáláson és az attribútumok súlyozásán megy keresztül. Végül ezekre a szövegdarabkákra futnak le a keresések. Először, a már korábban bemutatott, PostgreSQL beépített szövegkeresőjéről lesz szó, majd két Apache Lucene alapú projektről, a Solr-ről és az Elasticsearch-ről. Az alapvető különbség a PostgreSQL és a Lucene projektek között, hogy a PostgreSQL függvényei C-ben, míg az utóbbiak Java-ban íródtak. [PostgreSQL dokumentáció]

PostgreSQL full-text-search

A három kereső-szolgáltatás közül ez bizonyul a legkönnyebbnek, mind telepítés és rendszerigény szempontjából, amely az 3. fejezetben került kifejtésre. A hivatalos dokumentáció szerint a *Full Text Search* biztosítja azon természetes nyelvi dokumentumok azonosítását, amelyek eleget tesznek egy lekérdezésnek, valamint relevancia szerinti sorrendiség is állítható az eredményekben. Jelen esetben a lekérdezések egyenlők lesznek egy beírt címmel, amely alfanumerikus karaktereket tartalmaz. Szöveges lekérdezéssel kapcsolatosan már korábban is léteztek erre vonatkozó operátorok, mint például a *LIKE* vagy *ILIKE*, ám sok tekintetben hátrányt szenvedtek, főleg a sokkal összetettebb *Full Text Search*-csel szemben. Legnagyobb hibája a nyelviség támogatásának hiánya, valamint a toldalékolt szavak megkülönböztethetősége. Ha a keresésnek több találata is van, azok nem állíthatók sorba valamilyen kiszámolt rang szerint. A lefutás ideje is hosszadalmas, mivel nincs lehetőség indexelésre, tehát valamennyi keresésnél fel kell dolgozni az összes dokumentumot, ami sok

költséggel jár. Szerencsére a *Full Text Search* megoldást nyújthat e problémákra, mivel minden dokumentum egy elő-átalakításon esik át, amely a következőképpen alakul:

- Dokumentumok *token*[1]-be való konvertálása
- *Token*-ek átalakítása lexémákká[2]
- Előfeldolgozott dokumentumok tárolása, ezzel a keresések felgyorsítása

A normalizálás különböző módszerekkel finomhangolható:

- Indexelésre nem alkalmas stopszavak megállapítása
- Szinonimák és különböző nyelvi alakok kialakítása
- Szótövek megállapítása

A dokumentumok tulajdonképpen a szöveges keresés alapegysége, mint például egy könyvcím, vagy utcanév. A dokumentumok egy *tsvector* nevű adattípusban vannak eltárolva, míg a feldolgozott lekérdezések egy *tsquery* nevű típusban kerülnek rögzítésre. Rengeteg operátor elérhető ehhez az adattípushoz, de a legfontosabb a @@, amely az egyezésért felel. Egy egyszerű példa az operátor használatára:

```
SELECT 'Eötvös Loránd Tudományegyetem'::tsvector @@ 'Eötvös & Loránd'::tsquery;
?column?
----------
 t

SELECT 'Eötvös & Egyetem'::tsquery @@ 'Eötvös Loránd Tudományegyetem'::tsvector;
?column?
----------
 f
```

A *::tsquery* és *::tsvector* sorrendje nem számít, ami viszont igen az az, hogy a @@ operátor pontos egyezés esetén tér vissza igaz értékkel (a dupla kettőspont típuskonverziót jelent, opcionálisan használhatók a *to_tsvector()* és a *to_tsquery()* függvények, amennyiben normalizált dokumentumra vagy lekérdezésre van szükség).
Ezért lett a második lekérdezés eredménye hamis, mert a szövegben nincsen önállóan az „Egyetem” szó. Önmagában a *::tsvector* és *::tsquery* csak kisbetűssé és ábécé sorrendbe állítja

[1] Token: magyarul zsetont jelent, a program egy szövegből egy érthető objektumot hoznak létre a fordító számára.
[2] Lexéma: fogalmi jelentéssel rendelkező nyelvi elem. A lexéma ugyanúgy egy szöveges objektum, mint a *token*, azzal a különbséggel, hogy normalizált alakot vett fel.

a beírt szavakat. A következő példában normalizálás eredményét mutatom be. A *to_tsvector()* függvény által feldolgozott *string* eredménye egy betűrendbe sorolt lexémákból álló lista lesz.

```
SELECT to_tsvector('A geokódolás az a folyamat, melynek során valamilyen adatot térbeli objektumhoz (pl.: területegység, címadat) rendelünk azért, hogy meg tudjuk jeleníteni térképen. ') AS normalizalt_alak;

normalizalt_alak
----------
'''adat':9 'cím':14 'foly':5 'geokódolás':2 'jeleníten':20 'mely':6 'objektu':11 'pl':12 'rendel':15 'sor':7 'területegység':13 'tud':19 'térbel':10 'térkép':21 'valamily':8''
```

A függvény az egyes szavakat átalakítja kisbetűs lexémákká, a toldalékokat és stopszavakat eltávolítja, amelyek zavarhatnák a keresést. A számok a szavak elhelyezkedését mutatják a mondaton belül (például a „cím” a 14-ik szó helyét foglalja el a mondatban). Ugyanez a folyamat érvényes bármilyen begépelt cím esetén is. A *tsquery* argumentumában használhatók különböző operátorok, mint például az előbb említett *&* (ÉS), valamint a | (VAGY) és a *!* (NEM) is. Karakterek pótlására a *:** kifejezés alkalmazható.

```
SELECT to_tsvector('Eötvös Loránd Tudományegyetem') @@ to_tsquery('Tudomány:* & !Ötvös') AS eredmeny;

eredmeny
----------
t
```

Az eredmény azért lesz igaz, mert azokat a normalizált alakú szavakat keresi, amelyek a „Tudomány…” normalizált alakjával megegyezőek és bármilyen betűk követhetik az „y”-t, valamint akkor jelezzen igazzal, ha nem talált az „Ötvös”-re tökéletes egyezést. Amikor a *to_tsquery* és a *to_tsvector* van használatban, akkor a lekérdezés nem lesz kis- és nagybetűre érzékeny. A Postgresben több előre beépített nyelven támogatott a keresés: angol, dán, finn, francia, holland, magyar, norvég, német, orosz, portugál, román, spanyol és svéd. Amennyiben más nyelvkészletben való normalizálás a cél, akkor a *to_tsvector* és a *to_tsquery* paraméterének meg kell adni a nyelvet:

```
SELECT to_tsvector('english' , 'Ezt a példamondatot fogja normalizálni a postgres.') AS eredmeny;
eredmeny
----------
'''ezt':1 'fogja':4 'normalizálni':5 'postgr':7 'példamondatot':3"

SELECT to_tsvector('french' , 'Ezt a példamondatot fogja normalizálni a postgres.') AS eredmeny;
eredmeny
----------
'''a':2,6 'ezt':1 'fogj':4 'normalizáln':5 'postgr':7 'példamondatot':3"

SELECT to_tsvector('hungarian' , 'Ezt a példamondatot fogja normalizálni a postgres.') AS
eredmeny;
eredmeny
----------
'''fog':4 'normalizáln':5 'postgres':7 'példamondat':3"
```

Valamennyi nyelv esetében mások az előre definiált szótövek és toldalékok, ezért az eredmény is másképpen alakul mind a három alkalommal. Egy további beépített normalizálási opció a *simple*, ami figyelmen kívül hagyja a stopszavakat, nem törődik a szótövekkel és minden karakterlánc, amelyet szóköz választ el, lexémává válik. Ez geokódolásnál lehet hasznos, mivel az egyes utcaneveknek nem érdemes a szótöveit keresni, vagy éppen intézmények neveinél sokszor egy-egy stopszón is múlhat a releváns találat. Ezt kiegészítendő, a Postgres tartalmaz egy olyan kiegészítőt, amely eltávolítja az ékezeteket. Bizonyos esetben célszerű importálni a kiegészítőt, hiszen külföldiek is kereshetnek magyar nevű, ékezettel rendelkező címeket, objektumokat. Így az adatbázis megőrzi az eredeti ékezetes és az ékezet nélküli adatot, biztosítva az előzőekkel a többnyelvűséget. [Rachid, 2015]

```
CREATE EXTENSION unaccent;
SELECT unaccent('áéíóöőúüű') AS eredmeny;

eredmeny
----------
"aeiooouuu"
```

Indexelés

A lekérdezés folyamatának felgyorsításának érdekében szerepelnek különböző indexelési eljárások az adatbázis-kezelők világában. Rengeteg indexelési folyamat létezik, azonban szövegkeresés esetében két fajtáról esik szó, a GIN és a GiST indexről. Az új indexelés létrehozásának szintaxisa a következőképpen néz ki (a „< >” jelek nélkül):

```
CREATE INDEX <index neve> ON <táblanév> USING gist(<oszlop neve>);
CREATE INDEX <index neve> ON <táblanév> USING gin(<oszlop neve>);
```

Mindkét típusnak megvan a maga előnye és hátránya. A GiST indexelés nem mindig költséghatékony, mivel fals pozitív eredményt tud produkálni. Ha lehetséges, érdemes az indexelés végrehajtása előtt átfutni a táblákat, hogy ne generáljon felesleges találatokat. Szerencsére a PostgreSQL-nek van erre egy megoldása és automatikusan felkutatja a fals pozitív egyezéseket. A GiST index veszteségességének oka, hogy valamennyi dokumentumot egy előre meghatározott hosszúságú ismertetőjegy reprezentál. Ez az ismertetőjegy (másnéven aláírás) úgy alakul ki, hogy a program minden szót egy *n*-bit hosszúságú *string*-ben egy bitté hasítja (*hash*[3] függvénnyel), majd ezeket a biteket az *OR* (VAGY) operátorral összeillesztve adják ki együttesen az n-bit hosszúságú dokumentumot. Amikor két szó ugyanazt az értéket veszi fel a hasítófüggvény által, akkor fog keletkezni egy fals pozitív találat. Ha egy lekérdezésben valamennyi szóra van találat (tényleges vagy fals), abban az esetben a tábla megfelelő sorait érdemes kivizsgálni, hogy valóban helyes-e egy adott találat.

A GIN index nem veszteséges, amennyiben sztenderd lekérdezésekről van szó, ellenkező esetben a teljesítmény logaritmikusan változik az egyedi szavak számával. GIN index egyedül a *tsvector* lexémáinak értékeit tárolja, a súlyozást figyelmen kívül hagyja, így szükséges lehet a táblák újbóli átnézése, ha olyan lekérdezés fut, amelyben szerepelnek súlyok. Összehasonlítva a két indexelési típus teljesítményét a következő adatokat érdemes fontolóra venni az indexelés választásakor:

- A GIN index kikeresési sebessége háromszor gyorsabb a GiST-nél
- Az indexelés felépítés a GIN-nél háromszor több időt vesz igénybe
- GIN indexek frissítése tovább tart, mint a GiST-nél
- GiST indexek legalább háromszor kisebb tárterületet foglalnak

[3] hash: magyarul vagdalás, de informatikai értelemben hasításnak nevezzük. Létjogosultsága az informatikában két különböző területen van: adatok tárolása, valamint védelme. A hasítófüggvény hajtja végre a folyamatot, amely az 1950-es évektől kezdve használt fogalom.

Ebből az következik, hogy GIN indexet akkor érdemes használni, amikor statikus adat áll rendelkezésre, mivel a kikeresés számottevően gyorsabb. Gyakran változó tartalmú táblák esetében viszont a GiST a legjobb választás, mert lényegesen gyorsabban történik a frissítés. Számokkal magyarázva; GiST indexelés dinamikus állományokhoz kiváló és hatékony, ha a lexémák száma százezer alatt tartózkodik, efelett a GIN lehet a megfelelő választás, cserébe lassabb lesz az index aktualizálása. A folyamatot a *maintenance_work_mem* paraméter növelésével lehet felgyorsítani, de ez a GiST index kiépítésének idejére nincs hatással.

Rangsorolás

Amikor egy felhasználó beír egy címet vagy intézménynevet, akkor több találat közül is történhet egy megfelelő kiválasztása. Ennek a listának a sorrendjét próbálja a Postgres összeállítani, amelyhez két előre definiált függvény áll rendelkezésre. A rangsorolás megtörténik lexikális, szomszédosság és strukturális viszonyok alapján, azaz milyen gyakran tűnik fel egy szó a dokumentumban, mennyire vannak közel az adott szavak egymáshoz, mennyire meghatározó a szerepe egy adott kifejezésnek, valamint hol helyezkedik el a dokumentumon belül. Az, hogy egy adott szó mennyire releváns, az egy tág fogalom, mindig a felhasználási terület céljától függ. Valószínűleg nagyobb rangot fog kapni a Budapest címszóra keresve Magyarország fővárosa, mint a Georgia államban található, Budapestre keresztelt tanya. Az OpenStreetMap-en például az alapján történik a kilistázás, hogy milyen adminisztratív egység az adott objektum. Egy város feljebb kerül a ranglétrán, mint egy kicsiny tanya. A két függvény a következő: *ts_rank()* és *setweight()*. A *setweight* függvénnyel egy *tscvector* objektumnak lehet súlyt adni, ami A, B, C vagy D argumentumot vehet fel. Alapértelmezettként a súlyok értéke 1.0, 0.4, 0.2, 0.1, valamennyi lebegőpontos szám típusú. A gyakorlatban előfordul, hogy egy szöveg több száz vagy ezer szóból áll, így nagy valószínűséggel tartalmazni fog több tényezőt egy lekérdezésből. Mindkét rangsoroló függvény okosítható egy normalizációs paraméterrel, amellyel figyelembe tudja venni a mondat hosszát. Egyszerre több *integer* típusú szám is felvehető, például a 2|8, ami azt jelenti, hogy a rangot leosztja a dokumentum hosszával, valamint a dokumentumban szereplő egyedi szavak gyakoriságával. [Rachid, 2015]

Elírások kezelése

Gyakran előfordul, hogy egy cím vagy akármilyen név megadásánál elírás történik. A PostgreSQL erre is megoldást nyújt a *pg_trgm* nevezetű kiegészítővel. A modul az úgynevezett *trigram* modell segítségével elemzi a begépelt szavakat. A trigram egy olyan *n-gram*, amelynél *n*=3. Az *n-gram* egy *n* elemű, egymással érintkező sorozat, amely általában szövegdarabkákkal kapcsolatos. A tételek lehetnek betűk, szavak, beállítástól függ. A PostgreSQL-nél a plugin kifejezetten a három karakterből álló töredékeket vizsgálja. A *similarity()* függvény meghívásakor két szöveg típusú paramétert vár bemenetként, amelyeket össze fog hasonlítani. A *show_trgm()* függvény pedig megadja a beírt szöveg trigramjait egy tömbben.

```
SELECT similarity('mondat','mondat') AS eredmeny;
eredmeny
----------
1

SELECT similarity('mondat','mondta') AS eredmeny;
eredmeny
----------
0.4

SELECT show_trgm('mondat') AS eredmeny;
eredmeny
----------
{"  m"," mo","at ",dat,mon,nda,ond}

SELECT show_trgm('mondta') AS eredmeny;
eredmeny
----------
{"  m"," mo",dta,mon,ndt,ond,"ta "}
```

Az első lekérdezésben ugyanannak a szónak trigramjait összehasonlítva száz százalékos egyezést kaptam, viszont a másodiknál már csak 40% a hasonlóság. A harmadik, valamint a negyedik lekérdezés a „mondat”, majd a „mondta” szó trigramjait mutatja. Látszik, hogy két szomszédos betű felcserélése mekkora differenciát okozhat egy keresésnél, holott emberek számára nem tűnhet nagy különbségnek. [PostgreSQL dokumentáció]

Keresési szöveg kiemelése

Amikor a Google weboldalán keresünk egy kifejezést, az eredmény egy terjedelmesebb szöveg fragmentuma lesz, kiemelve a bemeneti szöveggel. Ezt a funkciót a PostgreSQL-ben is implementálták, a *ts_headline* függvény által. A függvény szintaxisa a következő:

```
ts_headline(normalizálási forma,'dokumentum szövege', to_tsquery('keresendő kifejezés'), 'opciók');
```

A függvény visszatérési értéke szöveges formátumú, benne nyomatékosítva félkövér betűtípussal a keresési szöveg. A függvényben először a normalizálási metódus megadása szükséges, tehát milyen nyelvnek megfelelően elemezze a dokumentumot, majd be lehet ágyazni egy lekérdezést a *tsquery*-n belül, majd különböző kulcs-érték párokkal lehet bővíteni az eredményt. Több érték megadása esetén vesszővel elkülönítve használandó. Az opciók a következők:

- *StartSel, StopSel*: a lekérdezés elemeit ezzel az opcióval lehet elkülöníteni az eredmény szövegétől. Például ha a bemeneti kifejezés „Loránd” és a *StartSel*, valamint *StopSel* értéke és (ez az alapértelmezett), akkor a visszatérő szövegben félkövér típust kap a bemeneti kifejezés.
- *MaxWords, MinWords*: ez határozza meg a kiemelésben szereplő szavak minimális és maximális számát.
- *ShortWord*: egy beállított hosszal megegyező, vagy annál rövidebb betűszámú szavak elhagyása a kiemelés elejéről és végéről.
- *HighlightAll*: ha értéke igaz, az egész dokumentumot érinti a nyomatékosítás, valamint az előző három paramétert figyelmen kívül hagyja
- *MaxFragments*: a szövegfragmentumok maximálisan megjelenítendő száma.
- *FragmentDelimiter*: ezzel a szöveges típusú adattal fogja a PostgreSQL a visszatérő töredék-szövegeket elválasztani. Alapértelmezetten három ponttal.

A *ts_headline* az eredeti dokumentumot használja, nem pedig egy *tsvector* objektumot, így könnyen meglehet, hogy az eredmény futási ideje lassabb lesz. Gyakori hiba, hogy minden egyező dokumentumra a *ts_headline* függvényt hívják meg, amikor mindössze tíz dokumentumot kell megjeleníteni, ilyenkor érdemes alkérdéseket is beépíteni, ahol már az előre *tsvector*-rá alakított eredményt fogja vizsgálni a *ts_headline*.

Fuzzy string match

Ennek a modulnak az a feladata, hogy megállapítsa a hasonlóságokat, valamint távolságokat *stringek* között. Az első típus, amelyet bemutatok, az a *soundex()* függvény. A *Soundex* egy olyan rendszer, amelyet az Amerikai Egyesült Államok Cenzusa hozott létre a nevek összehasonlíthatóságának megállapítására. Éppen ezért az angol nyelv alkalmazásánál a leghatékonyabb, idegen nyelveken kevésbé dolgozik megfelelő hatásfokkal. A PostgreSQL-ben két függvény által teszi lehetővé a rendszer a Soundex rendszer használatát: az egyik a *soundex*, a másik pedig a *difference* függvény. Az első függvény egy szöveget vár bemeneti értékként, amelyből képez egy Soundex kódot, – egy betűből és három számból áll – míg az utóbbi két szöveges argumentum közül állapítja meg a köztük levő differenciát és egy számot ad eredményül. A különbség a Soundex kódból adódik, mivel négy karakterből áll, ezért a végösszeg nulla és négy közé fog esni.

Az angoltól idegen nyelvek szempontjából sokkal inkább alkalmazhatóbb metódus a karaktersorozatok úgynevezett *Levenshtein* távolsága. Az algoritmus lényege, hogy a hasonlóságok és különbségek figyelembe vételével lesz egy egész számú érték, amely a távolságot adja. A három alkalmazható művelet a törlés, beillesztés, helyettesítés a végeredmény pedig ezeknek a műveleteknek a minimális száma egyik karakterláncból a másikba való alakításhoz. A fogalmat Vladimir Levenshtein vezette be 1965-ben, az algoritmus szerkesztési távolság néven is ismert. Ezzel a helyesírás ellenőrzése oldható meg, az alkalmazás javaslatot tesz, egy a begépelt szóhoz alacsony szerkesztési távolságra levő, helyesen írt szóra. A PostgreSQL-ben négy különböző fajtája van a függvénynek:

```
levenshtein('text source', 'text target', int ins_cost, int del_cost, int sub_cost)

levenshtein('text source', 'text target')

levenshtein_less_equal('text source', 'text target', int ins_cost, int del_cost, int sub_cost, int max_d)

levenshtein_less_equal('text source', 'text target', int max_d)
```

Ahol *text source* és *text target* a forrás és célszöveg, az *ins_cost*, *del_cost*, valamint *sub_cost* pedig egész számban a beillesztés, törlés és helyettesítés értéke. A szövegek hossza maximum 255 karakterből állhat, és nem vehet fel *null* értéket. A *levenshtein_less_equal* függvényt akkor érdemes használni, amikor a karakterláncok közti kis távolság az érdekes. Ha az éppen aktuális távolság az előre megadott *max_d* értékkel megegyező vagy nagyobb, akkor a függvény a

tényleges értéket adja vissza, ellenkező esetben egy, a *max_d*-nél nagyobb számot kapunk eredményül. Amennyiben a *max_d* értéke negatív, az eljárás ugyanaz, mint az eredeti levenshtein függvénynél.

```
SELECT levenshtein('példa','példa');
----------
0

SELECT levenshtein('pléda','példa');
----------
2

SELECT levenshtein('példa','érték');
----------
5
```

Az első lekérdezésnél a két karakterlánc megegyezik, nincsen elírás. A második lekérdezésben az *é* és *l* betűket felcseréltem, így ahhoz, hogy a *pléda* szóból *példa* legyen két művelet szükséges: *pléda→pééda→példa.* A harmadik minta is ezen az elven öt művelet alatt tudja teljesíteni a *példa* szóból az *érték* szóba való átalakítást.

A Soundex-hez hasonlóan a Metaphone és a Double Metaphone algoritmusok is a szavakat egy reprezentatív kóddá alakítják, a program pedig megítéli, hogy azonos kódúak-e. A hátrány ebben az esetben a különböző nyelvek ejtésmódja, valamint az, hogy a függvény leginkább az angol nyelvet részesíti előnyben. [PostgreSQL dokumentáció]

Elasticsearch és Solr

Az ElasticSearch egy Java alapú, platform független, sémamentes, nyílt forráskódú keresőmotor, amely az Apache Lucene nevű projektjéből nőtte ki magát. A program szolgáltatásait HTTP kéréseken keresztül lehet elérni, bármilyen REST[4]-et támogató program segítségével, mint például a Google Chrome-hoz modulként telepíthető Postman, vagy a Sense nevezetű szoftver. A HTTP kérések JSON formátumúak kell, hogy legyenek, valamint a rájuk érkező válaszok is ebben a formában várhatók, mivel ez az ElasticSearch dokumentumainak alapobjektuma. A szoftver vezető fejlesztője Shay Banon, aki 2010-ben adta ki az első verziót, azóta a szoftver az 5.3-as változatnál tart. A hivatalos weboldal elérhető a www.elastic.co-n. Jelenleg a legnépszerűbb vállalati keresőmotorként tartják számon, hiszen közel valós időben történik az indexelés és a keresés is nagyon gyors tud lenni. A szoftvert egy *log* fájlokat elemező, Logstash nevezetű és egy vizualizációs, Kibana elnevezésű platformmal egyetemben fejlesztik. A teljes kollekció az ELK Stack nevet viseli. Több nagy internetes szervezet is hasznát veszi a keresőmotor adta lehetőségeknek, mint például a Wikipedia, GitHub vagy a StackOverflow. Az ElasticSearch-nek nem alapfeltétele, hogy egy relációs adatbázis tetejében működjön, azaz támogatja a NoSQL adattárolást. Ennek előnye, hogy nem fenyegeti az adatbázis esetleges meghibásodása, valamint az ezzel együtt járó szolgáltatás karbantartása. Természetesen kapcsolatot lehet teremteni közel az összes adatbázissal. Ehhez szükség van egy JDBC nevű illesztőprogramra, amelynek révén lehet kommunikálni az adatbázissal (a Java alapú adatbázisok-kezelők is ezt a drivert használják).

A Solr az ElasticSearch-höz hasonlóan is az Apache Lucene projektből kinőtt kereső szolgáltatás. A Solr a második legnépszerűbb vállalati keresőmotor az ElasticSearch után, amelyet Java nyelven fejlesztenek, így valamennyi Java-val rendelkező platformon futtatható. A program sokoldalúságát tükrözi a full-text-search támogatása, a találatok kiemelése, a szűrőkkel való keresési lehetőség, a dokumentumok közel valós idejű indexelése, valamint könnyedén kezeli a *rich document* formátumokat, mint például Word dokumentumokat vagy PDF állományokat. Lehetőség szerint összekapcsolható relációs adatbázis kezelővel, de a NoSQL támogatásnak köszönhetően akkor sem ütközik akadályba, ha nem SQL alapú az adatbázis.

[4] REST: (Representational State Transfer) egy szoftverarchitektúra típus, amely nagy internetes rendszerek számára biztosít elosztott kapcsolatot. Azok a rendszerek, amelyek megfelelnek ennek a típusnak, RESTful-nak nevezhetők.

A keresőmotorok alapjai

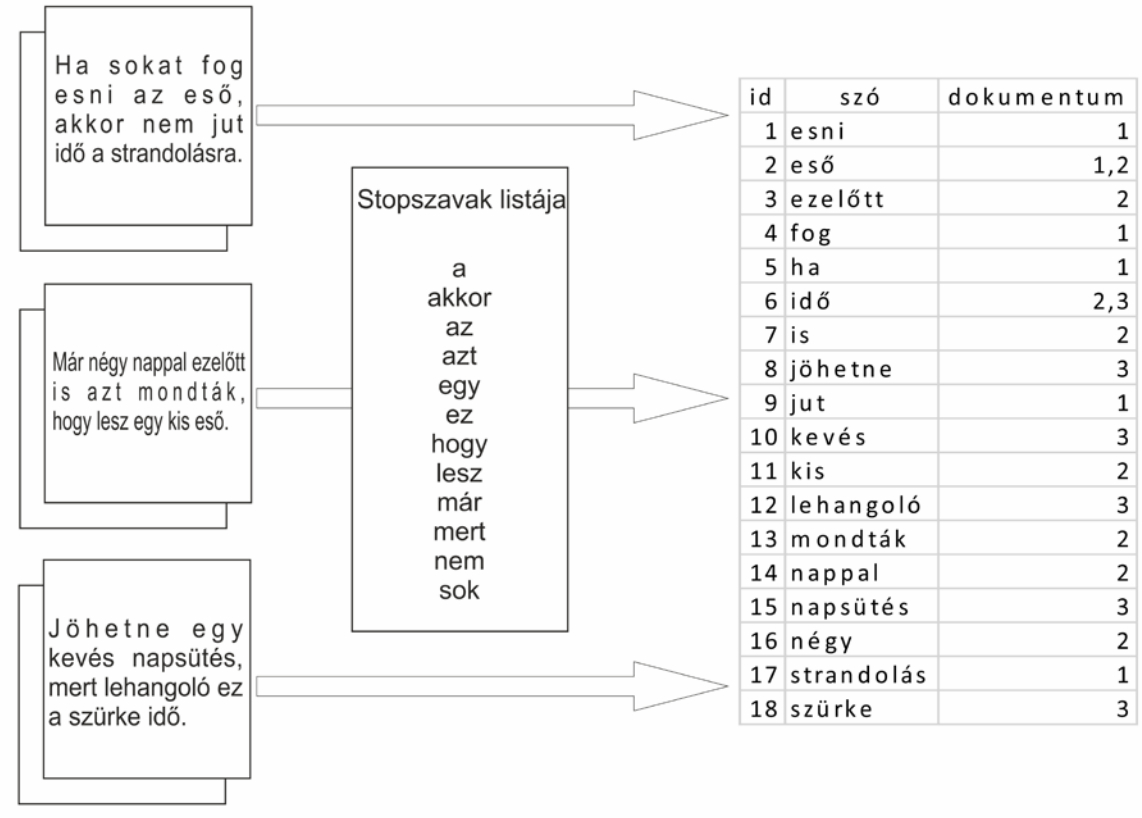
Mivel a két program sok tényezőben rendelkezik közös nevezővel, valamint mindkettőjüknek a lelkét a nagyteljesítményű Lucene könyvtár adja, ezért egy blokkban mutatom be őket, mind hasonlóság, mind különbözőség szempontjából. Először is néhány fontos alapfogalommal kezdeném, amelyekkel érdemes tisztában lenni. Ezeket a szolgáltatásokat szervereken alkalmazzák, amelyek együttesen egy klasztert alkotnak (angolul *cluster*). Minden egyes elindított példánya a valamely szolgáltatásnak egy *node*-nak felel meg, tehát a hálózatba bekapcsolt gépek. A közös elnevezések tárháza körülbelül itt véget is ér. A Solr esetében a fő adatstruktúrát kollekciónak (*Collection*) nevezik, amely több szilánkból (*shard*) épül fel. Egy kollekción belül élhet több szilánk is, valamint több *node*-on is előfordulhatnak. Egy kollekciót akár több *node*-on is el lehet osztani a teljesítmény fokozása érdekében. Opcionálisan a kollekciókból lehet úgynevezett replikát készíteni (*replica*), amely egy pontos mása a kollekcióban előforduló szilánkoknak. A replikák adta redundancia (és egyben egy biztonsági mentés) lehetőség arra a célra is megfelelő, hogy egy *node* meghibásodása esetén is elérhető maradjon az adat, valamint könnyebben lesz skálázható a rendszer. Az eddig említett konfiguráció az ElasticSearch-nél úgy néz ki, hogy a fő adatstruktúra elnevezése az index, amelyben szerepelhetnek szilánkok (*shard*) és azok másolatai (*replica*), amelyeket egy klaszteren belül szét lehet osztani. Ezeken túl az ElasticSearch rendelkezhet több típusú (*type*) dokumentummal egy indexen belül, amely azt takarja, hogy egy indexen belül több különböző struktúrát is fel lehet építeni (például egy közösségi oldalon a felhasználókat és a hozzájuk tartozó kommenteket). [Rafal, 2012]

Míg a Solr-nél előre definiálni kell az adatok mezőit és típusait egy sémában, addig az ElasticSearch-nél ez nem szükséges, tehát sémamentesnek vagy séma nélkülinek is mondható. Ez abban merül ki, hogy egy elindított ElasticSearch-nél már el is lehet kezdeni feltölteni az adatokat, közben a háttérben a program megpróbálja kitalálni a mezők típusait. Természetesen a hatásfoka nem 100%-os, de legtöbb helyzetben megfelelően működik ez az eljárás. Ha olyan adatfolyammal rendelkeznék, amelynél mégis inkább saját kezűleg szeretném definiálni a mezőket, akkor az úgynevezett tervezetek (*mapping*) megadásával tudom megvalósítani.

API elérhetősége

Ahhoz, hogy az eltárolt adatokból információt lehessen kinyerni, egy lekérdezést kell megfogalmazni, ezt pedig egy HTTP alapú alkalmazásprogramozási felületen (röviden API-n) keresztül lehet megtenni. Négy alapvető utasítás elegendő az adatok keresésére, törlésére, importálására és szerkesztésére, ezek a metódusok a következők: *GET* (keresés vagy olvasás), *DELETE* (törlés), *POST* (létrehozás) *és PUT* (frissítés). Ezeket a parancsokat megkapja a Solr vagy éppen az ElasticSearch, majd küld egy választ az eredményről. A Solr lehetővé teszi, hogy a felhasználó kiválaszthassa a visszaérkező adat típusát, amely többek között lehet JSON (*JavaScript Object Notation*) vagy akár XML (*Extensible Markup Language*) is. Mindeközben a másik szoftver csak a JSON kiterjesztést támogatja. A lekérdezések megfogalmazására néhány megfelelő program: a grafikus felhasználói felülettel rendelkező *Postman*, amely a Google Chrome beépülő moduljaként futtatható, illetve kizárólag parancssoros környezetben pedig a cURL lehet a megfelelő választás.

Felmerülhet a kérdés, hogyan éri el ezt a teljesítményt az Elasticsearch és a Solr, miért gyorsabbak az adatbázis-kezelőknél? A megoldás alapvetően az fordított indexelésben rejlik (6. ábra). A fordított indexelés egy olyan adatstruktúra, amely egy adatbázisban vagy dokumentumban tárolja a szavak és számok előfordulását az egyes dokumentumokban.



id	szó	dokumentum
1	esni	1
2	eső	1,2
3	ezelőtt	2
4	fog	1
5	ha	1
6	idő	2,3
7	is	2
8	jöhetne	3
9	jut	1
10	kevés	3
11	kis	2
12	lehangoló	3
13	mondták	2
14	nappal	2
15	napsütés	3
16	négy	2
17	strandolás	1
18	szürke	3

6. ábra Fordított index
Forrás: saját ábra

A technika nagy előnye az, hogy rendkívül gyors keresési időt produkál szöveges dokumentumok esetén, hátránya, hogy viszonylag költséges egy-egy új dokumentum hozzáadása az adatbázishoz. Két főbb típust lehet megkülönböztetni: a fordított fájl és fordított lista indexet. Az első módszer egy olyan listát tartalmaz, amely a dokumentumokban előforduló szavak hivatkozásait foglalja magában, a második pedig a szavak pozícióit mutatja az egyes dokumentumokban. Az utóbbi megoldás több funkcionalitást biztosít, például kifejezések keresésénél, viszont létrehozása nagyobb tárhelyet és erőforrást igényel. [Elasticsearch dokumentáció]

Idegen nyelvek kezelése

A korábbi említéseimnek megfelelően mindkét keresőmotor a Lucene-t használja fel az adatok indexeléséhez, de a saját Java implementációjuk különböző. Ebből adódóan az angoltól idegen nyelvek támogatásának megvalósítása sem mutat sok hasonlóságot, de azt érdemes leszögezni, hogy több, mint 30 nyelvi készlet áll már rendelkezésre. Ezeket az összetevőket nyelvi elemzőknek (*language analyzer*-nek) hívják. A dokumentumok átalakítása az idegen nyelvek normáinak megfelelően különböző filtereken keresztül végezhető. Csak úgy, mint a PostgreSQL esetében a normalizálási folyamat nyelvenként változik, mivel mások a szótövek és a stopszavak is. A Solr és az ElasticSearch is azt a kényelmi szolgáltatást nyújtják, hogy egyszerre több szűrővel okosítható és rugalmasabbá tehető a nyelvi *token*-ek generálása. Kétfajta szótövesítővel érkezik a Solr alapverziója, ami a magyar nyelvet illeti. Az egyik egy kisbetűssé alakító függvény, a másik pedig egy úgynevezett *Snowball* nyelven megírt szóelemző program. A *Snowball* egy olyan szöveg feldolgozó nyelv, amely lehetővé teszi a szótöveket készítő algoritmusok fejlesztését. [Rafal, 2012]

```
...
  <filter class="solr.LowerCaseFilterFactory"/>
  <filter class="solr.SnowballPorterFilterFactory" language="Hungarian"/>
...
```

Az ElasticSearch alapvető adatstruktúrája a JSON, ezért a parancsokat is egy ilyen fájlban kell megadni, emellett új dokumentum indexelésekor a PUT utasítás használatos:

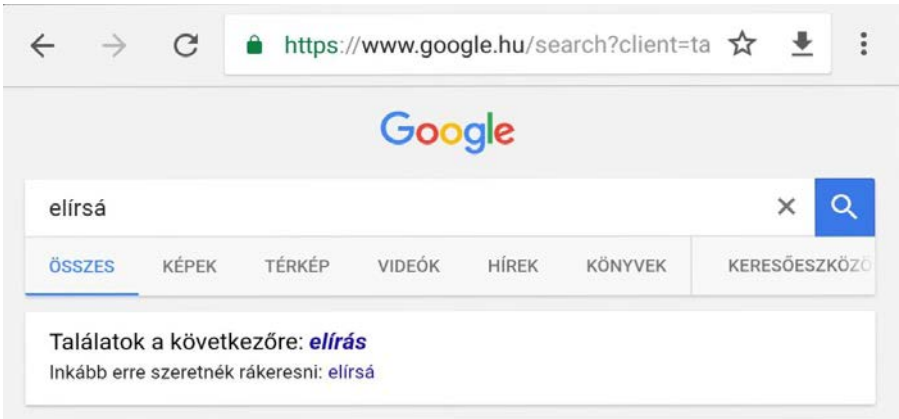
```
PUT /magyar_nyelvu_normalizalas{
 "settings": {
  "analysis": {
   "filter": {
    "hungarian_stop": {"type":   "stop","stopwords":  "_hungarian_" },
    "hungarian_stemmer": {
     "type":      "stemmer",
     "language":   "hungarian"}   },
   "analyzer": {
    "hungarian": {
     "tokenizer":  "standard",
     "filter": ["lowercase", "hungarian_stop","hungarian_stemmer"]
      }}}}
```

Természetesen a filterek paraméterei igény szerint konfigurálhatók, valamint egyéni elemző is készíthető több filterrel egyetemben. A két program közti eltérés ott érhető tetten, hogy míg az ElasticSearch képes dokumentumonként és lekérdezésenként más-más szűrőket alkalmazni, addig a Solr ebben a funkcióban hátrányt szenved.

Full text search

Ahogyan azt már említettem, mindkét keresőmotor API-ját HTTP parancsokkal lehet elérni. A Solr esetében egy elérési út és az azt követő *q=lekérdezés*-nek megfelelően, míg az ElasticSearch-nél az ember számára könnyebben olvasható (úgynevezett *human readable*) JSON fájl kerül elküldésre a szerver felé. Ez a struktúra lehetővé teszi a precíz és összetett kérések megfogalmazását. Másfelől a Solr egy lekérdezés elemzőt használ (jele: *q*) ahhoz, hogy a szöveges keresést ki tudja szedni az URL-ből. Geokódolás szempontjából a *full text search* az igazán fontos program-szolgáltatás, hiszen egy-egy lekérdezés során a programnak meg kell találnia egy címet, intézmény nevét a meglevő, koordinátákkal ellátott dokumentumok között, az adott kontextushoz mérten a találatokat kilistázni (természetesen a legnagyobb rangú eredmény egyből kikerülhet a térképre) és végül a listából kiválasztottat megjeleníteni térképen. Emellett fel lehet készíteni egy geokódolót arra is, hogy megpróbálja korrigálni az esetleges elírásokat és gépelés közben mutassa az adekvát találatokat. Értelemszerűen mindkét program felkészült e problémák megoldására, sőt a lekérdezési opciók túl is mutatnak rajtuk.

Egyrészről az úgynevezett *more like this* összetevő, amely a beírt lekérdezéshez hasonló találatokat próbál meg felajánlani bizonyos feltevéseknek megfelelően. Körülbelül úgy lehetne ezt leírni, mint sok weboldalon a *mások ezeket keresték még* vagy például a YouTube-on levő, megnézett videók alapján összeállított ajánlások. Másrészről a *did you mean* komponens, ami az elírásokat próbálja orvosolni úgy, hogy egy javított lekérdezés eredményeit tünteti fel, ezt mutatja be a 7. ábra.



7. ábra „Did you mean" funkció
Forrás: Google

A találatok rangsorolásánál fontos szerepet játszanak azok az algoritmusok és függvények, amely kiszámolják a lekérdezés kritériumainak eleget tevő dokumentumok pontszámait. Mindkét keresőmotorban többé-kevésbé lehetőség van arra, hogy befolyásolni tudjuk a pontszámozást. Mivel mindkét keresőmotor az Apache Lucene könyvtárát használják, ezért a pontozó algoritmus közel azonos. Több kifejezésből álló keresésnél számos modellt integrálva ad eredményt a dokumentumoknak. Először a Bool-féle információ-visszakereső (*BIR*) modell segítségével megállapítja azon dokumentumokat, amelyek tartalmazzák a feltételeket. A módszer az egyszerű Bool logikán (igaz vagy hamis) és a klasszikus halmazelméleten alapszik. A visszakeresés definiálható úgy, hogy:

1. Meghatározunk egy olyan S_i halmazt, amelyre a keresés igaz vagy hamis értékkel tér vissza.
    I. Ha igaz *A*-ra, akkor $S_i = \{D|A \in D\}$, ahol D egy olyan végez halmaz, amelynek elemei a dokumentumok
    II. Ha *A*-ra hamis, akkor $S_i = \{D|A \notin D\}$
2. Azokat a dokumentumokat kapjuk válaszként *Q* kérdésre, amelyek a halmazműveletek eredményeként kapott eredményhalmazhoz tartoznak.

Legyen *Q* egy kérdés, amely három keresési paraméterrel (A, B, C) rendelkezik és a paraméterek egyikének nem kell igazra teljesülnie, valamint az és logikai operátor az AND és a vagy operátor az OR:

Q = *A* AND (*B* OR *C*).

Ennek az eredménye három részhalmaz lesz:

S_1: *A* kifejezés részhalmaza,
S_2: *B* kifejezés részhalmaza,
S_3: *C* kifejezés részhalmaza.

Végül a logikai operátorok szerinti halmazműveletek:

$Q = S_1 \cap (S_2 \cup S_3)$. [Skropp]

Másodszor a szavak relatív fontosságának megállapításához a szó-gyakoriság–inverz-dokumentum-gyakoriság (angolul *Term Frequency/Inverse Document Frequency* vagy egyszerűen *TFIDF*) súlyozási modellt felhasználva készít egy rangsort a megállapított súlyokat követően. A súlyozás nagysága a megadott szó előfordulási gyakoriságától függ (ha sok dokumentumban szerepel, mint például egy névelő vagy kötőszó, akkor magas súlyt kap, tehát alacsonyabb helyezést ér el a rangsorban), valamint attól, hogy a megadott szavak milyen arányban mutatnak egyezést a dokumentumokban. Az úgynevezett szófrekvencia (t*erm frequency*) képlete a következő:

$$tf(t \text{ in } d) = \sqrt{frequency},$$

ahol *tf* a szófrekvencia, *t* a szó és *d* pedig a dokumentum. [BDE Research Kft., 2010]

Az inverz-dokumentum-frekvencia azt mutatja, hogy milyen gyakran fordul elő egy megadott szó egy kollekció összes dokumentumában. Itt is hasonlóan, minél gyakrabban, annál kisebb súllyal fog bírni. A formulát úgy lehet felírni, hogy:

$$idf(t) = 1 + log\ (\ numDocs\ /\ (docFreq + 1))$$

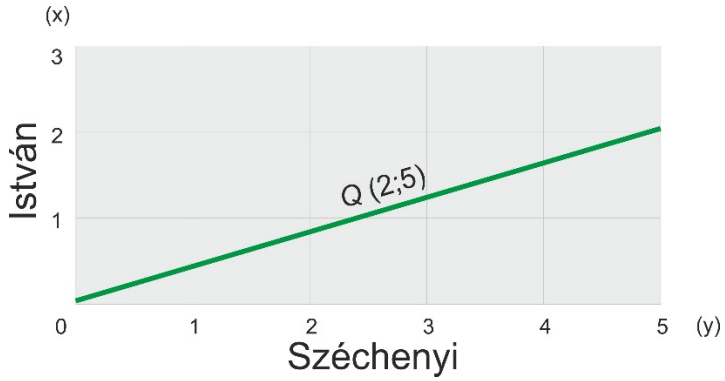
Tehát az inverz-dokumentum-frekvencia (*idf(t)*) egyenlő az indexben szereplő dokumentumok logaritmusával, amelyet elosztunk az adott szót tartalmazó dokumentumok számával. [Elasticsearch dokumentáció]

Harmadszor megnézi a mezők hosszát, és minél rövidebb egy mező, annál nagyobb súlyt fog kapni. Ha egy keresett szó egy rövid mezőben bukkan fel (például a címek között), akkor nagy eséllyel a dokumentum az egyezést mutató korpuszról (dokumentum állomány) fog szólni. A mezőhossz szabályt az ElasticSearch és a Solr úgy számítják, hogy:

$$norm(d) = 1 / \sqrt{numTerms}$$

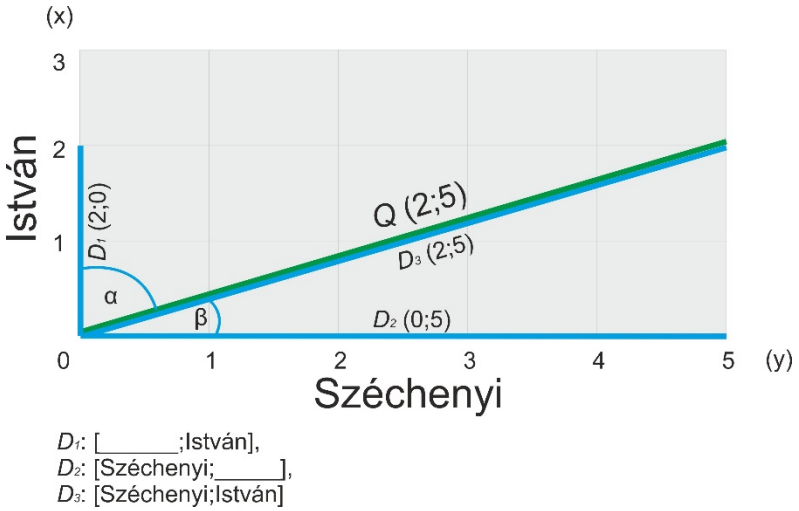
Végül a vektortér modell segítségével is megállapítja a súlyokat. A vektortér modell egy olyan szövegbányászati eszköz, amelynél egy dokumentum állomány minden dokumentuma egy

pontnak feleltethető meg egy sokdimenziós térben. Ez ad egy térbeli struktúrát és ebből kiszámítható a dokumentumok egymáshoz képesti hasonlósága, valamint egyedi jelentéstartalmak feltárására is alkalmas. A kimeneti érték egy pontszám, ami azt mutatja, mennyire egyezik a dokumentum a kérdéssel. Legyen *Q* kérdés D_1, D_2 és D_3 pedig a dokumentumok. A kérdés álljon két szóból, például *Széchenyi István*, így a lekérdezés megegyezik egy kétdimenziós vektorral, amely a 8. ábrán látható módon a (0;0) pontból indul.



8. ábra Vektortérmodell egy Q kérésre
Forrás: saját ábra

Azt tudjuk, hogy minél jobban egyedi egy szó, annál kisebb súlyt kap, tehát magasabb pontszámot fog elérni. A Széchenyi szó súlya legyen 5, az István szóé pedig 2. A dokumentumok közül csak D_3 tartalma mutasson teljes egyezést a lekérdezéssel, míg D_1, D_2 dokumentumokban félig található egyezést. A 9. ábra mutatja a dokumentumok tartalmát és a kérdésnek megfelelően az értékeit.



9. ábra A vektortér modell ábrázolása egy kérésre és három dokumentumra
Forrás: saját ábra

A vektortér modell előnye, hogy a vektorok összehasonlíthatók egymással. A *D* vektorok *Q*-val bezárt szögük alapján kiszámítható az egyezés, minél kisebb a bezárt szög, annál relevánsabb a dokumentum a kérdéshez képest. D_1 és *Q* által bezárt szög nagy, így ez a találat

a legkevésbé releváns, D_2 és Q közötti kisebb szögérték közelebb van a 100%-os egyezéshez és D_3 és Q egymás által bezárt szöge megegyezik, tehát D_3 a tökéletes találat.

Amint a keresőmotor egyezést talál, kiszámolja a dokumentumok pontszámait valamennyi összeillő szóra. Ezt a formulát *practical scoring function*-nek, magyarul alkalmazott pontozó függvénynek hívják. A függvény képlete az említett modellek alapján így írható fel:

```
score(q,d)  =                 //d dokumentum relevancia értéke q kérdéshez képest
        queryNorm(q)          //q kérdés normalizált értéke
        coord(q,d)            //koordinációs tényező
        ∑ (
            tf(t in d)        //szófrekvencia
            idf(t)²           //inverz-dokumentum-frekvencia
            t.getBoost()      //q kérdésre alkalmazott fokozás
            norm(t,d)         //mezőhossz érték
          ) (t in q)          //a súlyok összege minden q kérdésben szereplő t szóra
```

Bizonyos esetben előfordul, hogy egy cím beírásán kívül meg lehet adni egy R sugarú kört, amin belül keres a weboldal a kérdésre megfelelő választ vagy válaszokat. Például egy McDonald's éttermet keresek a belvárosban, az adott pozíciómtól maximum 300 méteres távolságban. Ekkor a telefon által bemért GPS koordináta lesz a kezdőpont és a lekérdezés pedig a McDonald's, 300 méteren belül. A geokódoló a találati listában azon elemeket fogja csak feltüntetni, amelyek minden kritériumnak eleget tesznek. Vagy létezhet olyan keresés is, hogy csak a Budapesten elhelyezkedő, Kossuth Lajos névvel rendelkező objektumokat kérdezem le. Ezt a típusú keresést az overpass-turbo is támogatja, ahol megadható az a terület (legyen az ország, régió, kerület vagy négy koordinátával határolt téglalap), amelyen belül a keresést végrehajtja a program. A Solr és az ElasticSearch is kezeli ezt a problémát, méghozzá az úgynevezett fazettás kereséssel. A fazettás keresés egy olyan keresési technika, amelyben a felhasználók több szűrőt megadva futtathatják a keresést. Ezt a kollekciót a szerver feldolgozza tényezőnként és csak azon dokumentumokat juttatja vissza a felhasználó elé, amelyek megfelelnek minden kritériumank. Jelen esetben a szűrés az koordináta és sugár alapján is történhet, a két keresőmotornál különböző a megvalósítás:

```
Solr
q=*:*&sfield=location&pt=10.10,11.11&facet=true&facet.query={!geofilt d=10}
```

A *facet.query* engedélyezi a fazettás keresést, amely megkapja, hogy mekkora sugarú körben történjen a keresés. Ezzel szemben az ElasticSearch egy *geo-distance* (földrajzi távolság)

szűrővel engedi szűkíteni a találatok listáját. A két lekérdezés a 10 kilométeren belül levő dokumentumokat fogja válaszul adni. [Solr dokumentáció]

ElasticSearch

```
{
 "query" : {
  "match_all" : {}
 },
  "geo_distance" : {
    "doc.location" : {
     "lat" : 10.10,
     "lon" : 11.11
    },
    "ranges" : [{ "to" : 10 }]
   }
 }
```

Konklúzió

Legtöbb új geokódoló az ElasticSearch-öt használja, mivel könnyebben kezelhető és gyorsan el lehet kezdeni vele dolgozni. A Solr is nagyon jó választásnak bizonyulhat, főleg, hogy a közösség, amely fejleszti és okosítja, sokkal nagyobb és érettebb, mint riválisáé. Amit mindenképp érdemes figyelembe venni, hogy annak ellenére, hogy mindkét szoftver nyílt forráskódú, nem jelenti azt, hogy a fejlesztések is nyilvánosan elérhetők. Az ElasticSearch-nél hiába fejleszt valaki egyénileg, az új főverzió nem biztos, hogy tartalmazni fogja az újítást. Ezt a szolgáltatást használja a Pelias geokódoló, amely a Mapzen nevű térképszolgáltató tulajdonában áll és az OpenStreetMap adatait használja fel. Egy másik nyílt-forráskódú geokódoló, amely az ElasticSearch-öt használja a keresés felgyorsítására, elérhető a www.photon.komoot.de weboldalon. Az OpenStreetMap Nominatim szolgáltatását egészíti ki több szempontból is. Amint elkezd a felhasználó gépelni, már elindul a keresés és ajánlatokat tesz a program és megpróbálja kiegészíteni a címet, valamint elnézőbb az elírásokkal szemben, mint a Nominatim. Mind a Solr, mind az ElasticSearch képes az SQL adatbázisokkal való kommunikációra, így továbbra is alkalmazhatók az OpenStreetMap adatait feldolgozó programok. (3. fejezet)

6. Az inverz geokódolás megvalósítása a TomTom térképi alappal

A TomTom egy holland autós térképszolgáltató és navigációs berendezéseket és szoftvereket gyártó cég, amelyet 1991-ben alapítottak. 2008-ban felvásárolták a Tele Atlas nevezetű, digitális térképész céget 2,9 milliárd euróért. A cég jelenlegi központja Amszterdamban található. A negyedévente elkészülő Európa térképüket használja az iData Kft, amelynek példáján mutatom be a fordított geokódolás folyamatát.

A térképi adatbázis felépítése szintúgy pontszerű, vonalas és felületi kiterjedésű objektumokból áll, mint akármelyik topográfiai céllal elkészített digitális térképkiadvány. Önmagában ez elegendő lenne szemmel való keresésre, de navigációs céllal különböző attribútumokkal látták el a geometriákat, amelyek gyakran egy táblában helyezkednek el azzal az objektummal, amelyre vonatkoznak. A vonalas objektumokat jellemző táblák rekordjai saroktól sarokig tartó útszegmenseket tartalmaznak. Az útszegmenseket leíró tábla magában foglalja a térképi geometriát, a két sarokpont házszám-tartományát, valamint a szegmens jobb és baloldali tulajdonságait. Ez az egyik legnagyobb eltérés a TomTom és az OpenStreetMap között, mert az OpenStreetMap nem tárol jobb és baloldali információt, hanem az adott geometriára vonatkozó címkéket. Továbbá a TomTom-nál az útszegmensek házszám-tartomány felosztása bizonyos esetekben megkönnyítheti a geokódolást, mert egyszerűen lineáris interpoláció segítségével meg lehet állapítani a keresett épület számát. Ez persze csak abban az esetben éri el a kívánt pontosságot, ha az utcában a házak egymástól közel azonos távolságra helyezkednek el és nincsen bennük kihagyás. A felületi kiterjedésű elemek között a geokódolás szempontjából jelentős objektumok az adminisztratív területek. Hasonlóan az OpenStreetMap-hez, itt is jelen van egy adminisztratív hierarchia, ahol a táblák A0-tól A9-ig terjednek. Az A0 tábla jelzi a legmagasabb beosztású közigazgatási területet, az országot, míg az A9 a kerületekre vonatkozik. Általánosan az jellemző ezekre a táblákra, hogy valamennyi egy hézag- és átfedésmentes felosztása a Föld felszínének és minden tábla a magasabb rangú adminisztratív területnek a partíciója. Ez alól kivétel a kerületek szintje, mert például Magyarországon csak Budapesten belül fordul elő A9-es besorolású terület.

A fordított geokódoló egy fő geokóderből és több al-geokóderből áll. A fő geokóder eljuttatja valamennyi al-geokóderhez a koordinátával rendelkező pontot, majd több részeredményt összevetve készít egy végleges kimenetet. Egy-egy geokóder külön forrásból származó térképi adatbázison dolgozik (például TomTom, HiSziHu, OpenStreetMap). Ezen adatbázisokból csak az úthálózattal, településekkel és országokkal rendelkező táblákból nyeri ki a geokóder az

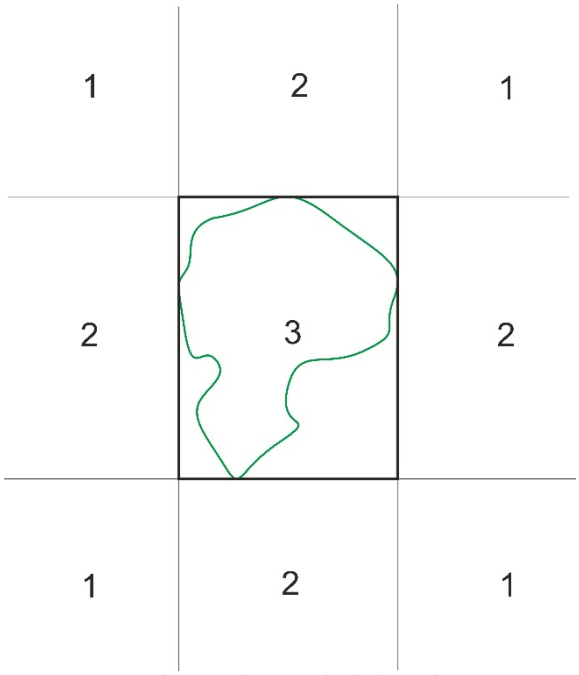
információt. Valamennyi geometriai adatot tartalmazó táblában van egy geometria oszlop, amelyen van egy R-fa alapú geometriai index származék. Az R-fa (*Region tree*) térbeli objektumok indexelésére fejlesztett eljárás. Az alapkoncepciója az, hogy az egymáshoz közel álló objektumokat csoportokra osztva egy legkisebb befoglaló téglalap reprezentálja. Ez az index olyan lekérdezéseket támogat PostGIS-ben, amelyekkel tartalmazást lehet vizsgálni, tehát egy település fordított geokódolásánál van egy pont, amelynek a 20 kilométeres körzetében meg kell találni azt a geometriát, amelyhez a legközelebb helyezkedik el. Mindez az elmélet, gyakorlatban több nehézséggel is meg kell küzdeni.

Valóságban fokban megadott koordinátákkal rendelkező pontok vannak, ezek átszámítása kilométerbe az első lépés. A fok-kilométer konverzió a WGS84 ellipszoid adottságai révén nem számítható át egy konstans értékkel. Kelet-nyugati irányban annál rövidebb egy fok ($\cos\varphi$), minél messzebb tartózkodik az Egyenlítőtől, északról dél felé és fordítva viszont állandó. Ez a meridiánkonvergenciának köszönhető, amely a hosszúsági körök összetartásának mértékét fejezi ki. [Györffy, 2010]

Legközelebbi település keresése (MinMaxDistance algoritmus)

A konverzió úgy történik, hogy a pont köré készül egy fokban megadott sugarú kör, a körhöz pedig egy befoglaló téglalap és arra vagyunk kíváncsiak, hogy az újonnan szerkesztett téglalap oldalai mekkorák kilométerben. Az eredmény Európa területén egy álló foktrapéz lesz. Probléma akkor keletkezik, ha a geokódolandó ponthoz közel van egy olyan objektum befoglaló téglalapja (*bounding box*), amelyben az objektum legközelebbi pontja távolabb helyezkedik el, mint a tényleges legközelebbi objektum.

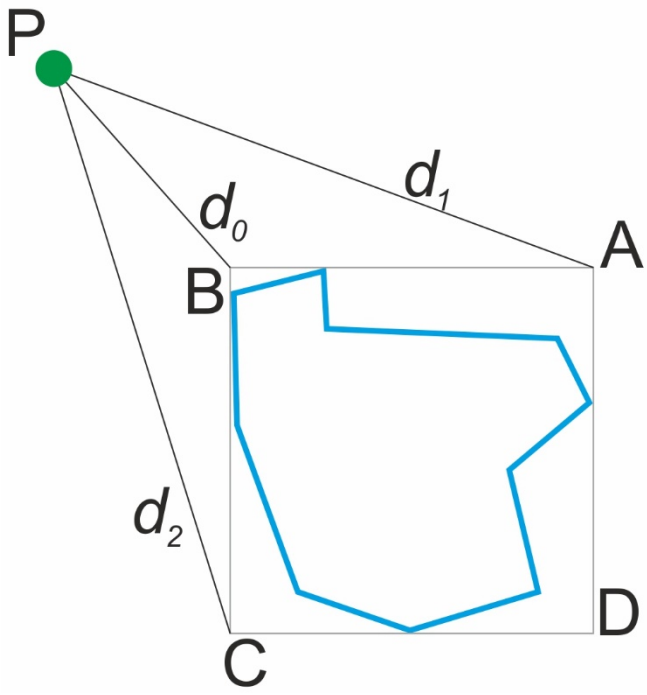
Ha van egy pont, amihez a legközelebbi geometriát keresem egy megadott *bounding box-on* belül, akkor az R-fában tárolt objektumok közül alulról felfelé történik a keresés. Tehát a fában található legalsó szintű geometriák befoglaló téglalapjait veszi alapul az kereső algoritmus. Az algoritmus fő erőssége, hogy a tényleges geometriát figyelmen kívül hagyja és kizárólag a befoglaló téglalapokat vizsgálja. Ennek eredménye, hogy a komplex geometriákat (több ezer vertex-ből állókat) nem kell feldolgozni első körben. Minden egyes megtalált levél elemnek kiszámolja, hogy melyik a potenciálisan legközelebbi pontja a kezdeti ponthoz. A pont a poligon befoglaló téglalapjához képest három különböző régióban helyezkedhet el, amelyet a 10. ábra mutat be.



10. ábra: A kétdimenziós tér felosztása
Forrás: saját ábra

Ha meghosszabbítom a határoló téglalap oldalait, akkor a síkot 9 részre osztja fel. Bármilyen pont három különböző fajta mezőben helyezkedhet el: 1, 2 és 3 helyen. Ha a pont az 1-es számmal jelölt mezők valamelyikében helyezkedik el, a legközelebbi pont a *bounding box* sarka lesz, ha a 2-es számot jelző helyen található, akkor a pontból a téglalapra vetített merőleges egyenes hossza lesz a legkisebb távolság, ha pedig a 3-as zónán belül van a pont, akkor a legközelebbi pont az maga a pont lesz. Mivel a vizsgált felület nem lóg túl a határoló téglalap keretein, ezért a keret és az adott pont közti d_0 távolság azt mondja meg, hogy biztosan nincsen közelebb magának a geometriának a legközelebbi pontja a keresett ponthoz. Ezen felül szükség van egy olyan metrikára is, amely megmondja, hogy mekkora az a maximális távolság, amelynél távolabb már nem eshet a geometria legközelebbi vertex-e a keresett ponthoz.

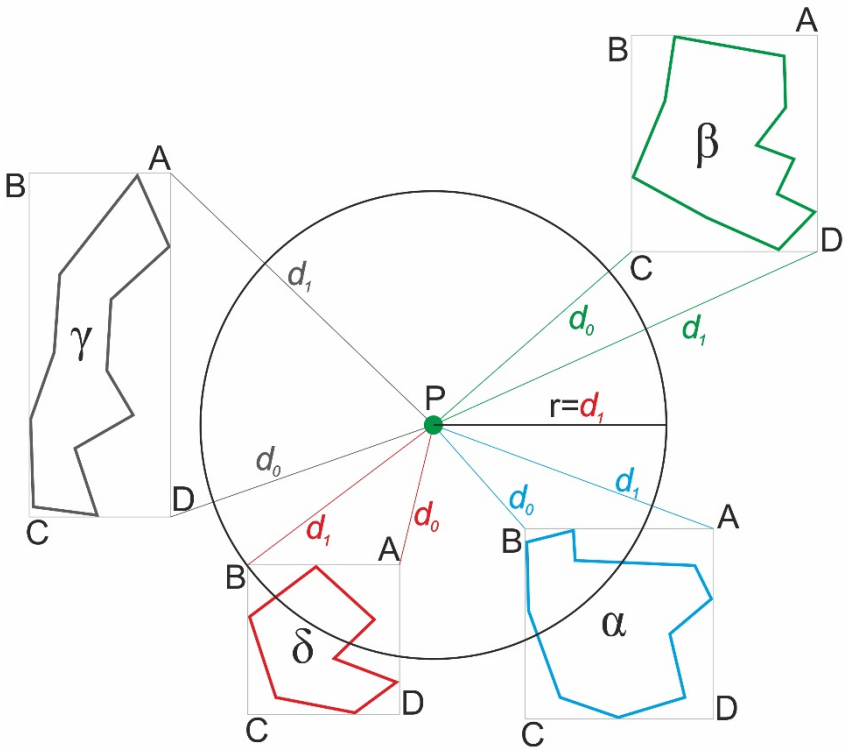
A 11. ábra azt mutatja, hogy *P* ponthoz legközelebb a *B* csúcs áll, majd *A*, *C* és *D*. A *D* csúcstól való távolság vizsgálata elhagyható, mivel azt mondaná meg, hogy mi az a pont, amelynél közelebb helyezkedhet el bármely pontja a vizsgált objektumnak. Tehát szükség van a pont és a legközelebbi három csúcs távolságára, amelyeket növekvő sorrendbe rendezünk és a második legközelebbi megfelel egy viszonylag jó felső becslésnek. Az a távolság, amelynél közelebb nem eshet pontja a poligonnak a *P* ponthoz képest (*MinTáv*), az d_0, a legnagyobb távolság (*MinMaxTáv*) pedig d_1. A d_2-vel jelölt hossz elhagyható, mert végül két távolságra lesz szükség. Az eddigiek alapján az eredmény egy olyan távolság pár, amely leírja, hogy a legközelebbi pont mely két távolság-tartományon belül fog elhelyezkedni.



11. ábra Branch and Bound algoritmus
Forrás: saját ábra

A szűrés második lépése egy olyan kör szerkesztése, amelynek a sugara a legalacsonyabb értékű maximális hosszal egyenlő (d_1). Tehát a maximális távolság, amelynél távolabb már nem eshet legközelebbi geometria lesz egy körnek az *r* sugara (ez a *P* ponttól mért második legközelebbi sarok távolsága). A berajzolt kör szomszédságában levő további befoglaló téglalapok közül így eldobhatók azok, amelyek legközelebbi pontja távolabb esik a kör kerületétől. Ezzel a módszerrel az esetek legnagyobb részében a téglalapok 99%-a leszűrhető. [Papadopoulos et. al., 1997]

A 12. ábrán négy különböző poligon (α,β,γ,δ) befoglaló téglalapjainak sarokpontjaitól mért minimális távolságok d_0-val a *MinMaxTáv*-ot pedig d_1-gyel jelöltem. Az algoritmus kiválasztja a legkisebb értékű d_1-et, amely egy szűrésre használt körnek lesz a sugara. A körön kívül eső befoglaló téglalapok eldobhatók.



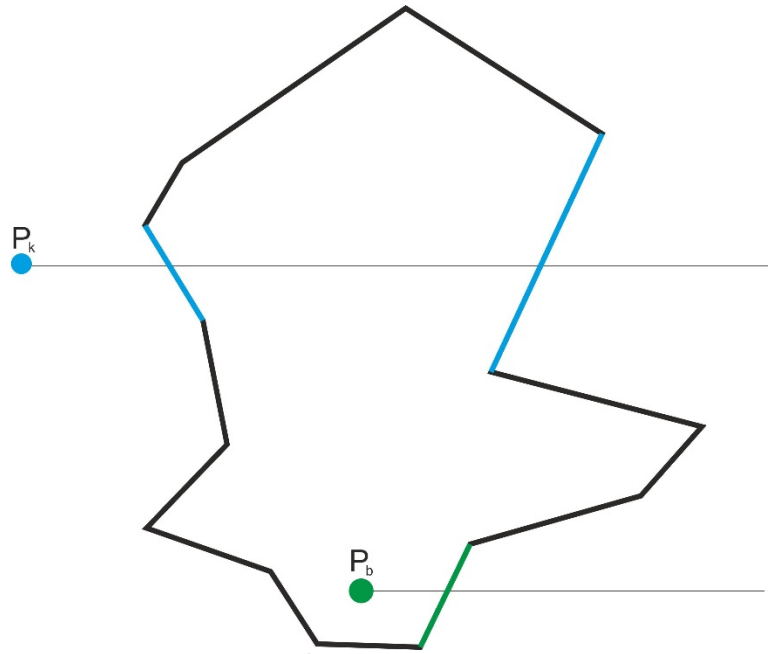
12. ábra MinMaxTáv alapján történő szűrés
Forrás: saját ábra

Végül a tényleges távolságot a megmaradt poligonoknál kell csupán számítani. A PostGIS-ben ez az *ST_Distance* függvény. Az alábbi példa Sopron és Nyíregyháza közti távolságot adja meg méterben. Az *ST_GeomFromText* metódus segítségével létre lehet hozni különböző, nem adatbázisban szereplő geometriát, jelen esetben egy-egy pontot, amelyek WGS84 dátum szerint értendők. Mivel az *ST_Distance* függvény a bemeneti értékek vetületében számolja az eredményt, ami fokban értendő, ezért szükséges egy vetületi transzformáció. Egységes Országos Vetületbe való transzformáláshoz rendelkezésre áll a PostGIS-ben az *ST_Transform* függvény, amely egy geometriát és egy *SRID*-t (vetületek azonosítója, azaz *Spatial Reference Identifier*) vár paraméterként. [PostgreSQL dokumentáció]

```
/*Sopron–Nyíregyháza távolság méterben*/
SELECT ST_Distance(
                ST_Transform(
                        ST_GeomFromText(
                        'POINT(16.5831 47.6848)',4326),23700),
                ST_Transform(
                        ST_GeomFromText(
                        'POINT(21.7271 47.9530)',4326),23700)
                        ) AS tavolsag;
tavolsag
----------
386298.486123889 (km)
```

Raycast algoritmus (pont a poligonban)

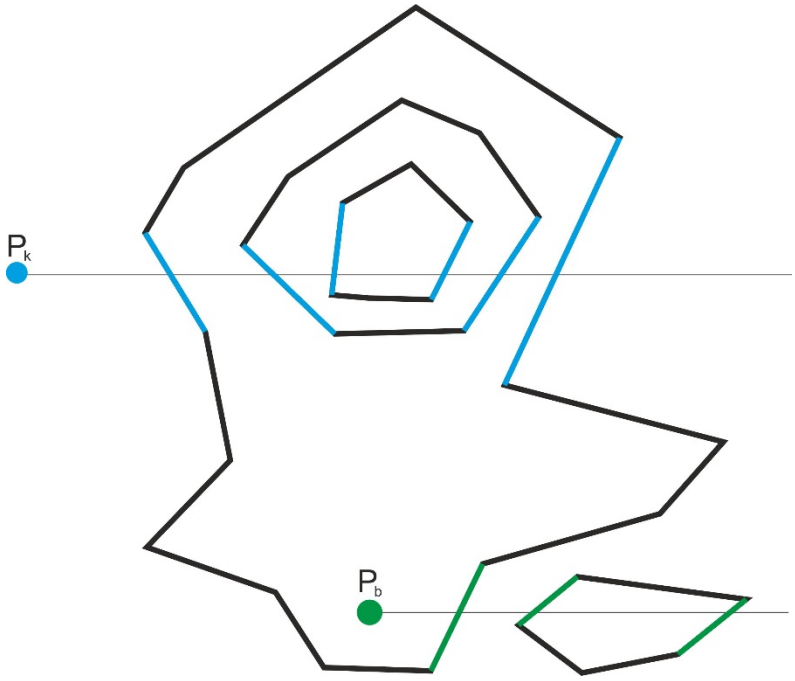
Ahhoz, hogy meg lehessen állapítani, hogy az adott koordinátákkal rendelkező pont például egy településen belül helyezkedik-e el vagy azon kívül, az úgynevezett pont a poligonban algoritmust kell segítségül hívni. Ha a poligonon belül van a keresett pont, akkor a távolságmérés (*ST_Distance*) lefuttatása kihagyható, megrövidítve így a válaszidőt. Ahogy az a 13. ábrán is látszik, legyen P_k pont a poligonon kívüli és legyen P_b a poligonon belüli pont. Mindkét pontból egy félegyenest szerkesztünk a poligon felé és megvizsgáljuk, hogy hányszor metszi az oldalait a félegyenes. Amennyiben a kapott eredmény páros, a vizsgált pont helyzete a poligonhoz képest kívülre tehető. Páratlan metszés esetén a pont pedig belül helyezkedik el a felülethez képest. Az eljárás hátránya, hogy a meglévő éllistát használja fel ahhoz, hogy a metszést megállapítsa. Magas oldalszámmal rendelkező poligonokat tekintve ez a metódus költséges lehet, ezért célszerű lehet a poligonokat vízszintesen kisebb régiókra felosztani és úgy vizsgálni a korpuszokat. A felosztott poligon szeleteihez lehet ugyanúgy befoglaló téglalapot számolni, mint a poligon esetében. Persze ez nagyobb tárhely igényét jelenti, mivel minden él legalább kétszer kerül tárolásra (háromszor abban az esetben, ha egy él átlóg egyik szeletből a másikba), de egy merevlemez beszerzése könnyen orvosolható. Első iterációban azt a szeletet kell megkeresni, amelyikbe a vetített (jelen alkalmazásban vízszintes) sugár beleesik. Ha a pont a nagy felületi objektum befoglaló téglalapján belül helyezkedik el, viszont egyik kis szelet téglalapja sem foglalja magába, akkor már tudni lehet, hogy a pont nem esik bele a nagy poligonba. Ezzel a módszerrel kikerülhető sok esetben a metszések vizsgálata. [Shimrat, 1962]



13. ábra Raycast algoritmus

Forrás: saját ábra

Ez a módszer működik úgynevezett multipoligonokon (több felületi elem egy objektumot képez), valamint lyukas felületeken is. A 14. ábrán az algoritmus a P_k és P_b pontok esetén vizsgálja az adott lyukas multipoligon metszéspontjait. A színezett oldalak jelölik a sugár által elmetszett szegmenseket. [Erlend, 2015]



14. ábra Raycast algoritmus multipoligonra

Forrás: saját ábra

Legközelebbi út keresése

A TomTom térképen megtalálható utak megjelenítési stílusai jogi besorolástól függenek. Ezeket a kategóriákat felhasználva (0-9, ahol 0 a legjelentősebb kategória) lehet a geokódolás során elemzett objektumok számát szűkíteni. Kezdetben az alacsonyabb jelentőségű utak között keres 500 méteres körben az algoritmus (4-9). Ha nincs találat, akkor egy következő kategória csoportot vizsgál nagyobb hatósugárban és így tovább innentől egyesével, a 0-val jelzett besorolásig, amibe az autópályák kerültek. Ezen kategóriára vonatkozóan a pásztázási kör sugara 10 kilométer, tehát minél magasabb rendű az út, annál távolabbról mondja azt az algoritmus a pontról, hogy az a megtalált út közelében van. A tényleges távolságot pedig úgy állapítja meg, hogy vetít egy merőlegest az út legközelebbi pontjára. A házszám megjelenítése pedig az út attribútumaiból fog kiderülni. Mivel minden szegmenset leíró adatok közt szerepel az út hossza, valamint a jobb és bal oldalán található házszámok intervalluma, ezért egyszerű lineáris interpolációval kideríthető, hogy az adott pont potenciálisan mely házszámhoz eshet

legközelebb. Természetesen ez nem egy tökéletes módszer, de kisebb a tárhelyet illető igénye, mintha minden egyes ház külön objektumként szerepelne a térképi adatbázisban, mint ahogyan az OpenStreetMap-en megtalálható. A TomTom előnye viszont az, hogy minden keresésnél lesz találat a házszámra vonatkozóan, míg az OpenStreetMap-en az út középvonala lesz az eredmény, ha nincs megrajzolva a keresett úthoz tartozó házszám.

A gyorsítótár előnyei

A gyorsítótár (angolul *cache*) feladata az információ-hozzáférés felgyorsítása. Ezt a tárolóhelyet több hardver is szolgáltathatja, de térképi adatbázisok esetében célszerű a számítógép memóriájára szűkíteni a kört, mivel a RAM-nak az elérési sebessége egy merevlemezhez vagy akár egy SSD-hez (*solid state drive*, azaz mozgó alkatrész nélküli meghajtó) viszonyítva is nagyságrendekkel alacsonyabb. Hátránya többek között, hogy a gép kikapcsolásakor a gyorsítótár tartalma elvész, valamint sokkal jelentősebb összeget kíván több memória modul beszerzése, mint egy merevlemezé. Természetesen egy szervergép esetében több RAM áll rendelkezésre, mint egy hagyományos asztali számítógépnél.

Adatbázisok felgyorsítására a különböző indexelési módszereken kívül is tökéletes partner a gyorsítótár, mert megkíméli a merevlemezt attól, hogy gyakran olvassa fel újra és újra az adatokat. Tehát érdemes felhasználni a memória segítségét, viszont meg kell találni azt az optimális lefoglalható területet, amely még nem sodorja veszélybe a szerver működését és nem okoz lassulást. Újraindítást követően a *cache* nem tartalmaz elemeket, majd minden egyes geokódolással feltöltődik adattal egészen addig, amíg a rendelkezésre álló területet teljes egészében meg nem telíti. Amint eléri a limitet, a legrégebbi adat törlésre kerül, hogy egy új elem bekerülhessen a gyorsítótárba.

7. Geokódoló API-k bemutatása

Két ugyanolyan teljesítményű geokódoló szolgáltatás aligha létezik, mivel számtalan forrásból, különböző szerkezetű adatbázissal rendelkeznek, valamint különböző komponensekből felépített szervergépek szolgálják ki a felhasználók által bevitt kéréseket. A SmartyStreets szoftvervállalat egy táblázatban összehasonlított több, mint 30 geokódoló szolgáltatást, köztük szerepelt a Google, Here, Yahoo, Nominatim is (sajnos a TomTom nem került fel a listára). A felmérés 2016. január és április közötti adatokat tükröz és első körben az Amerikai Egyesült Államokban hasonlították össze a geokóderek pontosságát. Összesen 11 cím alapján szűrtek átlag eltérést. Persze nem tudni, hogy mennyire voltak pontosak azon mérések, amelyek alapján kiválasztották a helyeket, de tekintsük azt most akkurátusnak. A legkisebb átlagos eltérést 25 méterrel a MapBox és a Yahoo produkálták, majd őket követi 30 méteres átlagos differenciával a Microsoft Bing geokódere. Második körben a világ nyolc pontját kellett beazonosítaniuk a geokódereknek a lehető legpontosabban. Ezt a fordulót toronymagasan a Google nyerte 100%-os találati rátával, átlagosan 10 méteres pontatlansággal. Ezüstérmes lett Here geokódere 20 méter alatti tévedéssel, viszont csak 88%-os találati rátával. Végül a harmadik helyet a Bing szerezte meg átlagosan 80 méteres pontossággal. A Nominatim valószínűleg akkor tudott volna előkelő helyen végezni, ha egy nyugat-európai kör is szerepelt volna a megmérettetésen. [SmartyStreets, 2012]

Mivel a TomTom valamilyen oknál fogva kimaradt a próbatételből, ezért lefuttattam a TomTom API-ján keresztül ugyanazokat a címeket, amelyeket megtaláltam a táblázatban és azt az eredményt kaptam, hogy az amerikai címek esetén átlagosan 850 méteres az eltérés, míg a világ többi pontja esetén pedig átlagosan 150 méteres a differencia, amitől ugyan nem került be a geokóder a tavaly elkészült mérés élmezőnyébe, de most már a TomTom is szerepel benne.

Egy 2012-es bejegyzésben is összehasonlításra került hét geokóder, itt az átlagos válaszidőt állították sorrendbe. A hét szolgáltató a következő: Yahoo, OpenAddresses, MapQuest, Google, Data Science Toolkit, CloudMade és Bing. Leggyorsabb átlagos geokódolási idővel a Bing büszkélkedhet, alig több, mint 0,2 másodperccel, majd a Google kicsivel 0,3 másodperc alatti teljesítménnyel. A CloudMade kivételével valamennyi geokóder fél másodperc leforgása alatt talált egyezést. Összesen két szolgáltató állt rendelkezésre napi 24 órában a hét minden napján, mégpedig a Google és a CloudMade. Öt év alatt minden bizonnyal a legtöbb szolgáltató számos időt fordított fejlesztésre, de érdekesnek találtam bemutatni egy korábbi állapotot. [DuVander, 2012]

Egy saját készítésű weboldalon összegyűjtöttem a Google, OpenStreetMap, valamint a TomTom geokódolóját. Az oldalon található egy link, amely a fordított geokódolást teszi lehetővé ugyanezen szolgáltatókkal. A weboldal elérhetősége: http://mercator.elte.hu/~horvat/geocode/geocode.html

8. Összegzés

Amikor elkezdtem összegyűjteni a forrásokat a témához, még hófehér területnek számított nekem a geokódolás menete. A rengeteg fórum, dokumentáció, valamint videó tanulmányozását követően világossá vált, hogy milyen problémákkal kell szembenézni mindkét fajta geokódolás kapcsán. A technika rohamos fejlődésének köszönhetően egyre jobb teljesítményű geokóderek látnak napvilágot, amelyek villámgyorsan szolgálják ki a felhasználók által megadott kéréseket. Az Elastic és a Solr adta gyorsaság rendkívül jól alkalmazható megfelelő paraméterek beállításával geokódolásra, mivel szövegbányászati célokra fejlesztették, mint például különféle *log* fájlok átfésülésére. A PostgreSQL–PostGIS együttes pedig rendkívül hatékony függvényekkel támogatott adatbázis létrehozását biztosítják térképi geometriával rendelkező objektumok számára, valamint térbeli indexek alkalmazásával a keresési idő is mérsékelhető. Az Elastic-ot egyre több szolgáltató implementálja egy újabb és nagyobb teljesítményű geokóder megalkotásához. Természetesen egy geokóder nem csak a jól megírt kódtól lesz hatékony, hanem a háttérben szükséges, hogy legyen egy megbízható és precíz térképi adatbázis. Minden egyes elírás, mérési hiba kihat a találatok listájára és félrevezető hatással lehetnek a felhasználókra, nem mellesleg azokon a területeken ahol az adatsűrűség nem áll megfelelő mértékben rendelkezésre.

9. Köszönetnyilvánítás

Szeretném megköszönni Dr. Elek Istvánnak, hogy hozzájárult a dolgozatom folyamatos előrehaladásához, valamint Dr. Gede Mátyásnak, aki segítségemre volt a weboldalam elkészítésénél. A dolgozat témájának kigondolását köszönöm Nagy Péternek, az iData Kft. fejlesztési vezetőjének, valamint köszönöm Kósa Botondnak, az iData Kft. szoftverfejlesztőjének, aki felvilágosított az inverz geokódolás folyamatáról.

10. Ábrajegyzék

11. Mellékletek

Result :

Field	Value
houseNumber	1
streetName	PÉTER
streetType	SÉTÁNY
extraInfo	/a
zipCode	1117
city	PÁZMÁNY
countryCode	HU
geocodingLevel	HOUSE_NUMBER
confidence	MAX

Is this correct ? if not **click here**

Some reasons that can explain why it fails :

- You haven't enter a postal address (does a post man will be able to give a letter to this address ?)
- The address is included in a more general text, or contains phone number, fax or some information that are not postal.
- The parser doesn't know the format or country

Try with an other address :

Enter the address to parse, select a country and an output format :

1117 pázmány péter sétány 1/a | Hungary (HU) | HTML

What's an address / what's not : to sum up, an address is a place where you can send a letter or where someone could live : A simple city or zip are not considered as an address, a street name is not an address without a city,.... The parser can handle this successfully but it is designed to handle REAL addresses

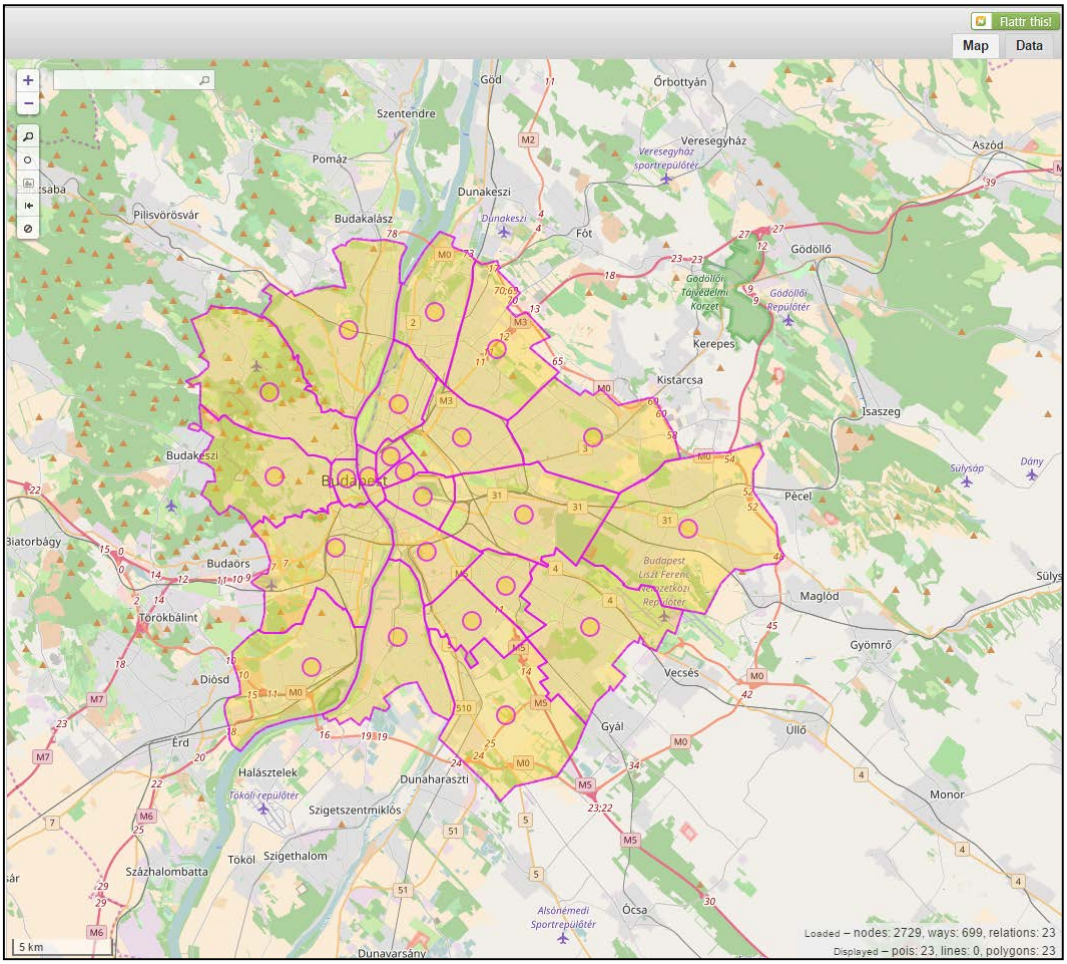
Parse ! Parse and standardize !

1. melléklet Feldolgozott cím
Forrás: www.address-parser.net

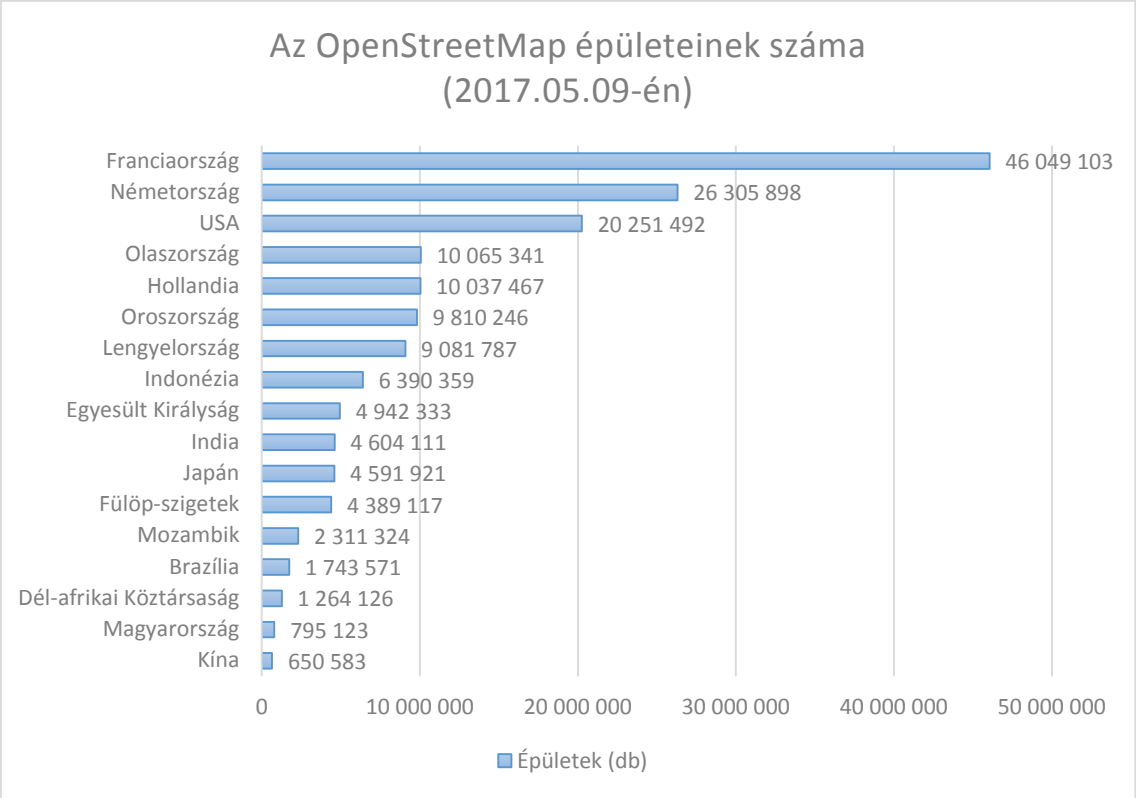
Run | Share | Export | Wizard | Save | Load | Settings | Help | overpass turbo

```
/*
This has been generated by the overpass-turbo wizard.
The original search was:
“boundary=administrative and admin_level=9 in budapest”
*/
[out:json][timeout:25];
// fetch area “budapest” to search in
{{geocodeArea:budapest}}->.searchArea;
// gather results
(
  // query part for: “boundary=administrative and admin_level=9”
  node["boundary"="administrative"]["admin_level"="9"](area.searchArea);
  //way["boundary"="administrative"]["admin_level"="9"](area.searchArea);
  relation["boundary"="administrative"]["admin_level"="9"](area.searchArea);
);
// print results
out body;
>;
out skel qt;
```

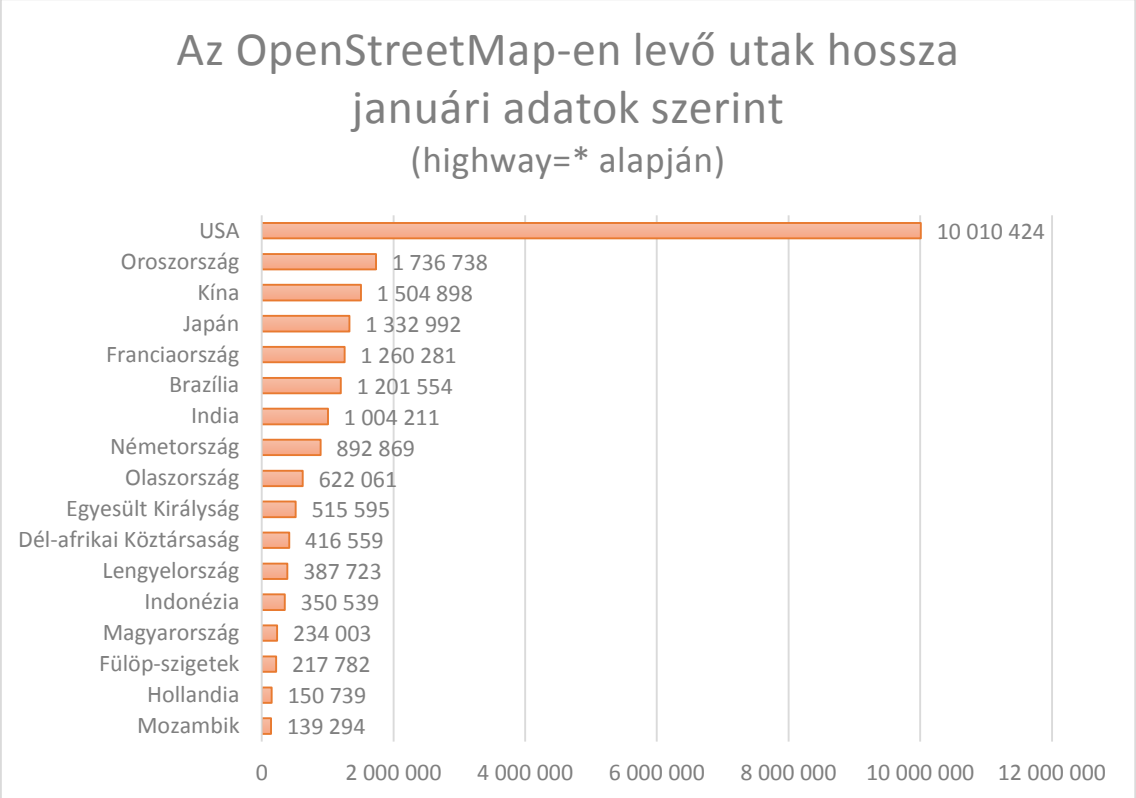
2. melléklet Adatbányász lekérdezés
Forrás: www.overpass-turbo.eu



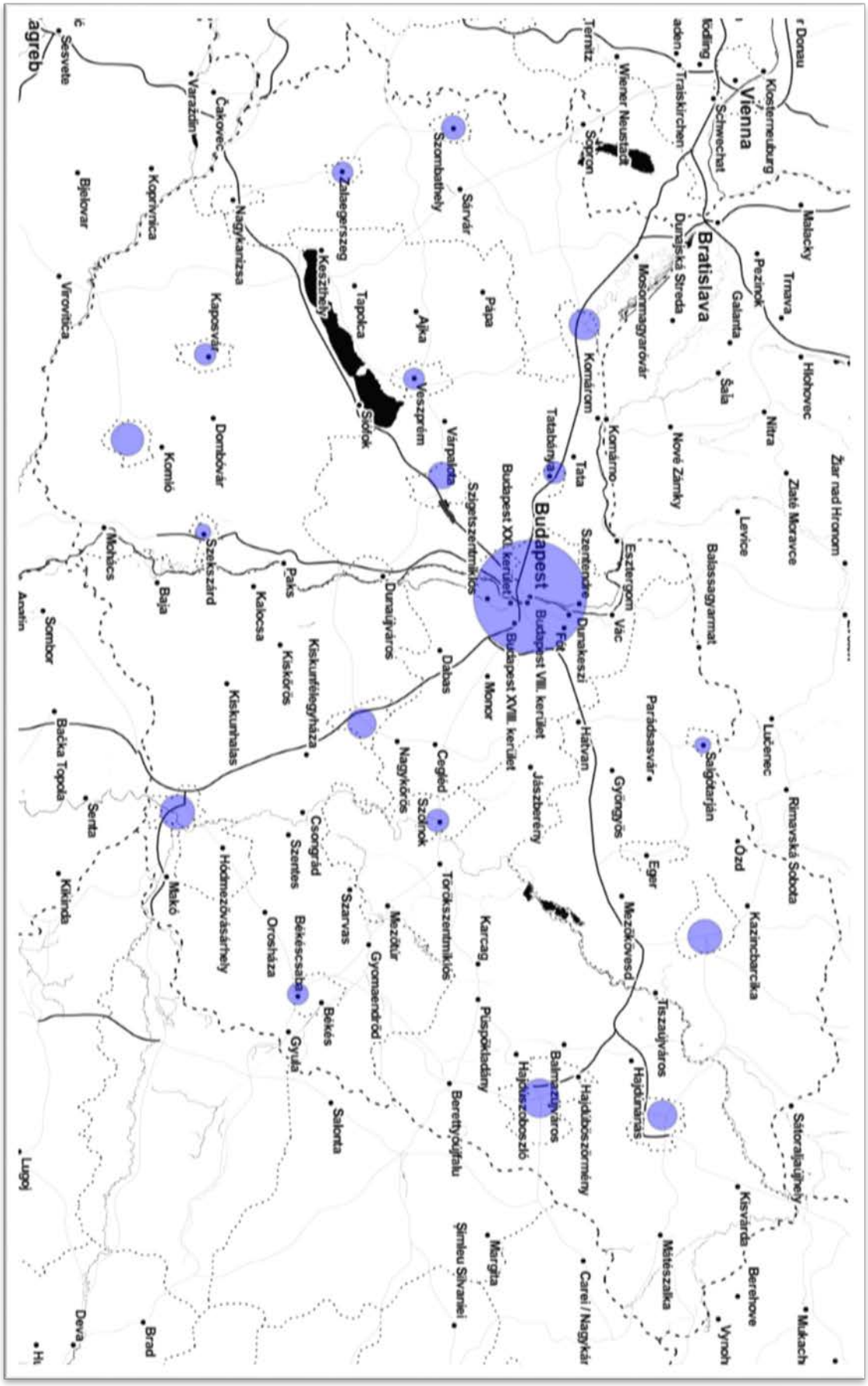
3. melléklet Lekérdezés eredménye
Forrás: www.overpass-turbo.eu



4. melléklet: OSM épületek száma
Forrás: www.osm-analytics.org



5. melléklet: OSM utak hossza
Forrás: http://www.missingmaps.org/osmstats/



6. melléklet Lakosságszám alapú megjelenítés
Forrás: saját ábra/www.overpass-turbo.eu

12. Irodalomjegyzék

Adam DuVander: *7 Free Geocoding APIs* (2012) [Online]
https://www.programmableweb.com/news/7-free-geocoding-apis-google-bing-yahoo-and-mapquest/2012/06/21
[Utolsó megtekintés: 2017.06.07.]

Al Barrentine: *Inside Libpostal - a fast, multilingual, international street address parser trained on OpenStreetMap data.* (2016) [Online]
https://mapzen.com/blog/inside-libpostal/
[Utolsó megtekintés: 2017.06.04.]

BDE Research Közhasznú Nonprofit Kft.: *Adatgyűjtés, elemzés, alaptechnológiák elemzése.* (2010) [Online]
http://palyazat.webstar.hu/gop/servlet/download?type=doc_field_file&field=file&id=4629
[Utolsó megtekintés: 2017.06.04.]

Carsten Möller: *OSM2PO*. (2016) [Online].
http://osm2po.de/
[Utolsó megtekintés: 2017.06.04.]

Dr. Györffy János: *A vetületek alap- és képfelülete.* (2010) [Online]
http://mercator.elte.hu/~gyorffy/jegyzete/alapfoga/JEGYZ_KR2.html
[Utolsó megtekintés: 2017.06.07.]

Dr. Skropp Adrienn: *Boole-féle információ-visszakeresés.* [Online]
www.i3ct.uni-pen.hu/mod/resource/view.php?id=65
[Utolsó megtekintés: 2017.06.04.]

Elasticsearch BV: *Elasticsearch Reference 5.4* (2017) [Online]
https://www.elastic.co/guide/en/elasticsearch/reference/5.4/index.html
[Utolsó megtekintés: 2017.06.05.]

EnterpriseDB Corporation: *PostgreSQL*. (2017) [Online]
https://www.enterprisedb.com/
[Utolsó megtekintés: 2017.06.04.]

Erlend Hamberg: *Fast, Offline, Reverse Geocoding; or, in Which Polygon am I?* (2015) [Online]
https://hamberg.no/erlend/posts/2015-10-22-geocoding.html
[Utolsó megtekintés: 2017.06.05.]

Goldberg, Daniel W.; Wilson, John P.; Knoblock, Craig A.: *From text to geographic coordinates: the current state of geocoding.* (2007) [Online]
The Free Library , https://www.thefreelibrary.com/From text to geographic coordinates: the current state of geocoding.-a0214605845
[Utolsó megtekintés: 2017.06.03.]

Jeff Blossom – *Geocoding Best Practices.* (2015) [Online]
http://gis.harvard.edu/services/blog/geocoding-best-practices
[Utolsó megtekintés: 2017.06.03.]

Martin Raifer: *Overpass-Turbo*: *A web based data mining tool for OpenStreetMap using Overpass API.* (2012) [Online]
https://github.com/tyrasd/overpass-turbo
[Utolsó megtekintés: 2017.06.04.]

OpenStreetMap Contributors: *osm-analytics: data analysis tool frontend.* (2016) [Online]
https://github.com/hotosm/osm-analytics
[Utolsó megtekintés: 2017.06.04.]

Papadopoulos A., Manolopoulos Y.: *Performance of nearest neighbor queries in R-trees.* In: Afrati F., Kolaitis P. (1997)
Database Theory — ICDT '97. ICDT 1997. Lecture Notes in Computer Science, vol 1186. Springer, Berlin, Heidelberg

Rachid Belaid: *Postgres full-text search is Good Enough!* (2015)
http://rachbelaid.com/postgres-full-text-search-is-good-enough/
[Utolsó megtekintés: 2017.06.07.]

Rafal Kuć: *Solr vs. ElasticSearch.* (2012)
https://sematext.com/blog/2012/08/23/solr-vs-elasticsearch-part-1-overview/
[Utolsó megtekintés: 2017.06.07.]

Sarah Hoffmann, Paul Norman: *Osm2pgsql: OpenStreetMap data to PostgreSQL converter.* (2017) [Online]
https://github.com/openstreetmap/osm2pgsql
[Utolsó megtekintés: 2017.06.04.]

Shimrat, M.: *Algorithm 112: Position of point relative to polygon*. (1962)
Communications of the ACM Volume 5 Issue 8

SmartyStreets: *Geocoding Service Comparison.* (2012) [Online]
https://docs.google.com/spreadsheets/d/1I2rEVX2CN8AqkhpzUTuNwNvJy8il1exccrsd4OGwDCU
[Utolsó megtekintés: 2017.06.07.]

The Apache Software Foundation: *Apache Solr™ 6.3.0 Documentation.* (2017) [Online]
https://lucene.apache.org/solr/6_3_0/index.html
[Utolsó megtekintés: 2017.06.05.]

The PostgreSQL Global Development Group: *PostgreSQL 9.5.7 Documentation.* (2017) [Online]
https://www.postgresql.org/docs/9.5/static/index.html
[Utolsó megtekintés: 2017.06.05.]

A weboldal elkészítéséhez használt források:

Google Maps: *Developer's Guide.* [Online]
https://developers.google.com/maps/documentation/geocoding/intro
[Utolsó megtekintés: 2017.06.07.]

OpenLayers API Documentation [Online]
http://openlayers.org/en/latest/apidoc/
[Utolsó megtekintés: 2017.06.07.]

TomTom: *Maps SDK, Functional Examples.* [Online]
https://developer.tomtom.com/maps-sdk/functional-examples
[Utolsó megtekintés: 2017.06.07.]

NYILATKOZAT

Alulírott Horváth Attila (Neptun azonosító: R4VSL4), az *A geokódolás szerepe az adatbázis alapú digitális térképek világában. Geokódoló szolgáltatások.* című diplomamunka szerzője fegyelmi felelősségem tudatában kijelentem, hogy dolgozatom önálló munkám eredménye, saját szellemi termékem, abban a hivatkozások és idézések standard szabályait következetesen alkalmaztam, mások által írt részeket a megfelelő idézés nélkül nem használtam fel.

A témavezető által elfogadott és elbírált diplomamunka elektronikus közzétételéhez (PDF formátumban a tanszéki honlapon).

HOZZÁJÁRULOK NEM JÁRULOK HOZZÁ

Budapest, ……..

a hallgató aláírása

Hozzájárulok a szakdolgozat benyújtásához:

Budapest, ……..

a témavezető aláírása