



M Ű E G Y E T E M 1 7 8 2

Budapesti Műszaki és Gazdaságtudományi Egyetem

Villamosmérnöki és Informatikai Kar

Méréstechnika és Információs Rendszerek Tanszék

Kooperáció és gépi tanulás labor (vimim223)

Labor jegyzőkönyv

MÁTYÁS-BARTA CSONGOR (VYW0YR)

2014. OKTÓBER 2.

Tartalomjegyzék

1. A mérés bemutatása	2
2. Feladat 1	2
2.1. Leírás	2
2.2. Megoldás	2
3. Feladat 2	3
3.1. Leírás	3
3.2. Megoldás	3
4. Feladat 3	3
4.1. Leírás	3
4.2. Megoldás	4

1. A mérés bemutatása

A labor során egy kitalált alkalmazási terület dokumentumait kell indexelni, illetve keresni. Az alkalmazási terület a HWSW Informatikai Hírmagazin 2008-2009-es híreinek szövegbázisa. A rendszer alapvető feladata, hogy képes legyen a rövid hírek szövegeiben egyszerű módon keresni.

2. Feladat 1

2.1. Leírás

Tervezzen és valósítson meg egy egyszerű inverz dokumentum előállító programot! A program a paraméterként megadott könyvtárban található szöveges dokumentumokat elemzi, majd egy kimeneti index fájlban eltárolja, hogy melyik szó mely dokumentumokban található meg. Egy egyszerű statisztikát kell készíteni. Az inverz dokumentum (avagy dokumentum index) lényege az, hogy ha megadunk egy szót, akkor kikereshető legyen, hogy az mely dokumentumokban szerepelt. Azt is célszerű eltárolni, hogy egy szó hányszor szerepelt egy adott dokumentumban. Az index fájl szerkezete szabadon kialakítható, de feleljen meg a további feladatoknak. Az index kialakításakor ne mátrixos formában tárolja a szó-dokumentum párokat, hanem tömörítve, azaz csak azok a szó-dokumentum párok szerepeljenek, melyek előfordulási gyakorisága nem nulla.

2.2. Megoldás

Az inverz dokumentum létrehozásához a következő adatstruktúrát és osztályt definiáltam:

```
Map<String,Map<String,Integer>> map;

public class Indexer
{
    private TermRecognizer recog;
    private int docCount;

    public Indexer() throws IOException

    public Map<String,Map<String,Integer>> indexFolder(String folderName)

    public void indexFilesInFolder(final File folder, Map<String,Map<String,Integer>> map)

    public void writeToFileAsClearText(Map<String,Map<String,Integer>> map, String filename)

    public void writeToFileSerialized(Map<String,Map<String,Integer>> map, String filename)

    public Map<String,Map<String,Integer>> readFromFileSerialized(String filename)
}
```

A *map* kulcsai a dokumentumokban talált szavak, a kulcsokhoz rendelt érték pedig egy olyan szótár, amiben a kulcs a dokumentum név az érték pedig az adott szó előfordulásának száma a dokumentumban. Ezen adatstruktúra segíti a szavak gyors feldolgozását, előfordulási helyeik gyors visszakeresését. Az *Indexer* osztály végzi az előbbi adatstruktúra feltöltését. Az *indexFolder* metódus rekurzívan bejárja a paraméterül megadott könyvtárat, minden egyes fájlra lefuttatva az indexelő algoritmust. Ez az algoritmus a *TermRecognizer* megadott segédosztályt felhasználva a beolvasott szöveges fájl szavakra bontja, számolva azok előfordulását, majd hozzáadja ezt a *map*-hoz. Az *Indexer* osztály emellett képes elmenteni egy fájlba vagy visszaolvasni a szavak indexét.

3. Feladat 2

3.1. Leírás

Készítsen az 1. feladatban készített index fájlra egy kereső programot. A program bemenő paraméterként szavakat kap, eredményként kiírja azon dokumentumok nevét, amelyekben mindegyik szó szerepel.

3.2. Megoldás

A keresést az alábbi osztály valósítja meg: A *searchForTerms* metódus paraméterként megkapja a keresni kívánt

```
public class Searcher {  
  
    public List<String> searchForTerms(List<String> terms,  
                                       Map<String, Map<String, Integer>> map)  
}
```

szavakat és az index változót. Az indexből kiszedi hogy az egyes szavak melyik dokumentumokban fordultak elő, majd ezen dokumentum listák metszetét képezi, így megkapjuk azon dokumentumok listáját amelyben mindegyik szó előfordult. Futási példa/Használat: A fenti példában az index beolvasása után rákerestünk a

```
Indexer indexer = new Indexer();  
Searcher searcher = new Searcher();  
  
Map<String, Map<String, Integer>> map;  
map = indexer.readFromFileSerialized("indexfile.ser");  
  
List<String> terms = new ArrayList<String>();  
terms.add("processzor");  
  
System.out.println(searcher.searchForTerms(terms, map).size());
```

```
Output:  
455
```

processzor szóra és a keresőnk 455 cikket talált amiben szerepelt ez a szó. Ha most az *Intel* szót is hozzáadjuk a kereső szók listájához, akkor, mint várható volt lecsökken a talált dokumentumok száma.

```
terms.add("processzor");  
terms.add("Intel");  
System.out.println(searcher.searchForTerms(terms, map).size());
```

```
Output:  
253
```

4. Feladat 3

4.1. Leírás

Módosítsa a 2. feladatban elkészített kereső programot szemantikus keresőszó kiegészítéssel! A felhasználó által megadott keresőszavakhoz vegyen fel továbbiakat a PC-shop ontológia és egy OWL következtető segítségével. 2014. október 2.

vel. A felvett kulcsszavak lehetnek a felhasználó kulcsszavainak megfelelő osztályok ősei vagy leszármazottai, de más módszert is használhat. A keresési eredmények alapján egészítse ki az ontológiát: vegyen fel további kifejezéseket a gyakori keresések eredményének javításához.

4.2. Megoldás

A feladat megoldásához a következőképpen módosítottam a *Searcher* osztály, illetve hoztam létre egy újat, a *PelletConceptExpander* osztályt.

```
public class Searcher {
    public List<String> searchForTerms(List<String> terms,
                                      Map<String, Map<String, Integer>> map)

    public List<String> searchForExpandedTerms(
        List<String> terms,
        Map<String, Map<String, Integer>> map,
        PelletConceptExpander expander)
}

public class PelletConceptExpander {

    public PelletConceptExpander(String ontologyFilename,
                                  boolean inclAnnotations,
                                  boolean inclSubclasses)

    public Set<String> expandConcept(String concept)

    public Set<String> expandConcept(String concept,
                                      boolean inclAnnotations,
                                      boolean inclSubclasses)
}
```