Budapesti Műszaki és Gazdaságtudományi Egyetem

Villamosmérnöki és Informatikai kar

SZÖVEGES DOKUMENTUMOK DARABOLÁSA ÉS TÖMÖRÍTÉSE HASH-KÓDOLÁSSAL

DARABOLÁSI TECHNIKÁK ÉS MÁSOLATKERESÉS

DIPLOMATERV FELADAT

Pataki Máté

2002.

konzulensek: Dr. Charaf Hassan

Monostori Krisztián

Budapesti Műszaki és Gazdaságtudományi Egyetem
Automatizálási és Alkalmazott Informatikai Tanszék

Arkady Zaslavsky

Monash University, Melbourne, Australia
Distribution of Computer Sciences and Software Engineering

Nyilatkozat

Alulírott *Pataki Máté* szigorló hallgató kijelentem, hogy a diplomatervben foglaltak saját munkám eredményei, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel.
Tudomásul veszem, hogy az elkészült diplomatervben található eredményeket a Budapesti Műszaki és Gazdaságtudományi Egyetem, a feladatot kiíró egyetemi intézmény oktatási célra felhasználhatja.

Kelt Budapest, 2002. május 23.

.....
hallgató aláírása

Tartalomjegyzék

Ábrajegyzék

Tartalmi kivonat

Az elektronikus formában tárolt dokumentumok korában egyre inkább felmerül a szövegmásolatok keresésének, és kimutatásának feladata. Sok más egyéb eszköz mellett, ez a módszer is hatékony védelem lehet a törvénytelen dokumentummásolatok terjedése ellen.

A teljes szöveg egyezésének vizsgálata helyett célszerű a szöveget megfelelő méretű kisebb részekre, un. töredékekre osztani, majd ezen részek egyezését vizsgálni. Diplomadolgozatomban az egyes szövegek darabolásának különböző módjait, valamint ezen darabok hash-kódolással való tömörítésének lehetőségeit kutattam.

A szöveg darabolásának igen sokféle lehetősége van, melyek közül három lényeges típust vizsgáltam meg: átlapolódó szavas, mondatonkénti és hash kódon alapuló. Ezeket a darabolási eljárásokat hasonlítottam össze egymással, különböző szempontok szerint. A kapott eredmények alapján javaslatot tettem, hogy mely darabolási eljárás mely felhasználási területre alkalmas a leginkább.

A hash-kódolással való tömörítésre az MD5 algoritmust használtam és arra a kérdésre kerestem a választ, hogy milyen kódhossz a legalkalmasabb ezen a felhasználási területen.

A kutatások tapasztalatait felhasználva sikerült két hasonlóságkereső programot is megvalósítani, ezek is ismertetésre kerülnek.

Abstract

We live in a time of electronically stored documents and there is a big need for document overlap detection. There are many ways of protecting our documents against plagiarism and this is one of them.

Instead of comparing whole documents it is practical to chunk them, and only comparing these chunks. In my thesis I analyse different document chunking methods and the compression efficiency of these chunks with a hash function.

There are many ways of chunking, but we analyse only the three main types: word, sentence and hashed breakpoint chunking. I compared these chunking methods to each other. At the end I recommend special use for the different methods.

I use the MD5 algorithm for compressing the chunks, and analysed what the ideal code length is for our purposes.

The results of this research have been implemented in two copy detection systems.

1 Bevezetés

A számítástechnika fejlődésével az írott művek előállítási folyamata egyszerűsödött, publikációjuk gyors és könnyű lett. Méltán hasonlítják az Internet feltalálásának hatását a könyvnyomtatás feltalálásához, hiszen az Internet segítségével bármilyen mű nagyon könnyen, gyorsan és igen olcsón közkinccsé tehető. Így a szellemi művek olyan tárháza jött létre, amely semmilyen eddig létező könyvtárhoz nem hasonlítható, se méretben, se elérhetőségben, se használatában. A digitális adattárolás mindemellett végletekig egyszerűsíti a művek másolását, egészének vagy részeinek átvételét, így nagymértékben megkönnyíti a plágiumok létrehozását is.

Teljesen természetesnek vehető tehát, hogy a szellemi termékek (képek, irodalmi és zenei művek, tudományos publikációk) eredeti szerzői megfontoltak a digitális könyvtárakkal és az Internetes publikálással kapcsolatban. Ezért a digitális könyvtárak használatának sarkalatos kérdése a művek megóvása az illegális másolásoktól. Erre a problémára számos megoldás született, amelyek alapvetően két csoportba sorolhatóak: a másolás megelőzése, valamint a másolatok felderítése.

Ezen diplomatervezési feladat kereteiben Hodász Gábor és Pataki Máté által közösen megvalósított és tesztelt rendszer a második kategóriába tartozik, célja szöveges dokumentumok közötti részleges és teljes átfedések keresése. A megjelenítő komponenssel együtt (mely Monostori Krisztián munkája [Monos2]) alkot teljes és működő rendszert, melyet a melbourne-i Monash Egyetem Elosztott Rendszerek tanszékének tanárai és munkatársai tesztelnek és használnak [Monash].

A *második fejezet*ben áttekintésre kerül a másolatkeresési rendszerek általános felépítése, a lehetséges szövegdarabolási eljárások, a töredékek tömörítésének lehetősége, valamint az adatbázis megvalósításának lehetőségei.

A *harmadik fejezet*ben áttekintő leírás található a megvalósított tesztrendszerekről, komponenseiről és a felhasznált eszközökről.

A *negyedik fejezet*ben a darabolási eljárásokon végzett, valamint a hash kód hosszára vonatkozó tesztek részletes leírása található.

Az *ötödik fejezet* két működő rendszert mutat be, melyeket a negyedik fejezetben végzett tesztek eredményére támaszkodva implementáltunk.

A diplomaterv alapját képező feladat megvalósítása közös munka eredménye, melyet Hodász Gábor és Pataki Máté együtt végzett. Ezért az első és második fejezet közös a két diplomatervben, a fennmaradó fejezetek az egyénileg elvégzett részfeladatokat és kapott eredményeket tartalmazzák. Így a jelen dolgozat Hodász Gábor munkájával [Hod02] kiegészítve ad teljes leírást a rendszerről.

A rendszer a Budapesti Műszaki Egyetem Automatizálási és Alkalmazott Informatikai Tanszéke által kiírt diplomaterv pályázat keretében került fejlesztésre, de a munka a melbourne-i Monash Egyetem Elosztott Rendszerek Tanszékén, az ott rendelkezésre álló eszközök és erőforrások felhasználásával folyt. A Monash Egyetem minden jogot fenntart magának.

Az itt leírt kutatások eredményei tudományos publikáció formájában is megjelentek és az amsterdami International Conference on Computational Science 2002 konferencián előadásra kerültek [Monos1].

2 Irodalmi áttekintés

Dolgozatunkban előfordulnak olyan szavak, kifejezések, melyeket a hétköznapi szóhasználatban más értelemben használunk, vagy esetleg bővebb magyarázatra szorulnak. Ezeket szótárba gyűjtöttük ki (M1. melléklet).

2.1 Elektronikus dokumentumok másolása és másolatok keresése

Amint a bevezetőben is említésre került, a szellemi termékek védelme alapvető követelménye a Digitális Könyvtárak széleskörű elterjedésének, hiszen ha nincsenek kellőképpen biztosítva a szerző jogai, akkor nem is fogja közzétenni művét. A másolatok felderítésén alapuló technikák más megoldásokkal ellentétben nem akadályozzák a dokumentumok szabad publikációját, azonban felkutatják az illegális másolatokat.

Egy elektronikus dokumentum másolásának számtalan különböző oka lehet. Ezek közül csak egy (de kétségtelenül gyakran előforduló) az illegális másolat, illetve plágium. Az alábbiakban ezek közül vázolunk fel néhányat, melyek később a másolatkereső rendszerek alkalmazási területét is meghatározzák.

2.1.1 Illegális másolatok, plágiumok

Az élet számtalan területén találkozhatunk másolattal, hamisítvánnyal, ötletlopással. A tudományos publikációk területén különösen fontos a probléma megoldása, hiszen napjainkban egyre többször fordul elő, hogy egy cikket, publikációt meg se jelentetnek hagyományos formában, csupán elektronikusan. Mind a diákoknak, mind a kutatóknak igen kényelmes és egyszerű eszköz áll rendelkezésére tehát, hogy megkerüljék a saját mű készítésének fáradtságos munkáját. Azonban hiába tudja a tanár, hogy a beadott dolgozat feltehetően

nem a diák zsenialitásának terméke, az eredeti dokumentummal való összevetés nélkül ezt nem tudja bizonyítani. Azonban az eredetit megtalálni számítógépes támogatás, keresőprogram nélkül gyakorlatilag lehetetlen.

2.1.2 Legális másolatok, publikus terjesztésű dokumentumok

Az Interneten számtalan publikus (szabad terjesztésű) dokumentum is található. Ezek több példányban is előfordulhatnak legálisan. Ilyenek például a következők:

- FAQ (Frequently Asked Questions, Gyakran Ismételt Kérdések) vagy RFC (Request For Comments) dokumentumok
- népszerű programok dokumentációi
- tükröoldalakon tárolt dokumentumok stb.

Részleges átfedés is lehet dokumentumok között számtalan okból:

- ugyanannak a dokumentumnak különböző verziói
- ugyanazon dokumentum másfajta formázása
- ugyanazon dokumentum környezetfüggő módosításai
- más szövegek összefűzésével létrejött nagyobb dokumentum
- többfele osztott dokumentum

Amennyiben a fájlok a helyi rendszeren léteznek, egy hasonlóságkereső rendszer így is segítségére lehet a felhasználónak. A merevlemezek bizonyos idő elteltével egyre több felesleges dokumentumot tartalmaznak, melyek például más dokumentumok korábbi verziói, vagy ideiglenes adatok stb. A keresőrendszer megmutathatja ezeket a hasonlóságokat, és a felhasználó dönthet a törlésükről.

2.2 Másolatkereső rendszerek

Megvalósításuk szerint a másolatkeresők alapvetően két csoportba sorolhatóak: „aláírás alapú” és „regisztrációs alapú” megközelítés. Ezek mind az illegális másolatok felkutatását könnyítik meg. Ilyen és hasonló elveken működő rendszerek napjainkban kezdenek megjelenni az Interneten [Cop01], [EVE00], [Gla99], [Int01], [Pap01], [Pla99].

2.2.1 Az aláírás alapú megközelítés

Az aláírás alapú eljárások esetében egyfajta „aláírás” kerül a szövegbe, melynek alapján később nyomon lehet követni a másolatokat, felderíteni az egyezéseket. Egy igen elterjedt alkalmazás az úgynevezett vízjelek használata, például szóközök és kontrollösszegek elhelyezése a szövegben. Számtalan különböző algoritmus, metódus látott napvilágot különféle aláírások, ujjlenyomatok és vízjelek megvalósítására, azonban ezen dolgozat keretében ezekre nincs módunk kitérni [Hei99]. Az aláírás alapú eljárásoknak azonban van két hátránya: egyrészről gyakran nagyon könnyű őket automatikusan eltávolítani, másrészt nem használhatók részleges átfedések kimutatására.

2.2.2 A regisztrációs alapú megközelítés

A regisztrációs alapú megközelítés létrehoz egy adatbázist, mely nagy mennyiségű dokumentumot regisztrál és tárol. Új dokumentum érkezésekor összeveti azt az adatbázisban már regisztrált dokumentumokkal, teljes vagy részleges átfedéseket keresve. A kutatások során igen sokféle eljárás alakult ki az adatbázis kialakítására valamint a dokumentumok darabolására. Az adattárolás kérdései Hodász Gábor munkájában kerülnek kifejtésre [Hod02], míg a darabolási eljárásokkal a 4.2. fejezet foglalkozik részletesebben.

A dokumentumok adatbázisban való tárolása szerint alapvetően két típust különböztethetünk meg:

1. Az adatbázisban az eredetinek elfogadott műveket regisztráljuk, és minden beérkező dokumentumot összevetünk az adatbázisban szereplőkkel, teljes vagy részleges átfedéseket keresve.
2. Az adatbázisban nagy mennyiségű dokumentumot tárolunk (például Internetről letöltötteket), majd a beérkező dokumentumot fogadjuk el eredetinek, és keresünk hozzá részben vagy egészben hasonlót az adatbázisban.

Az összehasonlítási kérés érkezhet személytől, például egy konferencia rendezője megvizsgálja, hogy a beérkezett publikációk mennyiben „merítenek" az előző évek anyagából. Másfelől végezheti automatikusan bármilyen szoftver is, például digitális könyvtárak kartotékozó rendszere, publikáció gyűjtemények kezelőprogramja stb.

Természetesen a hasonlóságkeresésnek (a megvalósítást az aktuális feladathoz hangolva) számtalan más felhasználási lehetősége is létezik. A teljesség igénye nélkül sorolunk fel néhányat:

- Egyetemi házi feladat beadó rendszer, melyben regisztrálva vannak a korábbi évek munkái, és a beérkező új feladatokat azonnal egy másolatkeresési szűrésnek vetjük alá [Kock99].

- Számtalan irodalmi felhasználás is elképzelhető, például fordítások összehasonlítása, különböző források vizsgálata stb.
- Egy internetes keresőrendszer motorjában is fel lehet használni, mely egy adott dokumentumhoz hasonlóakat keres. Esetleg megadja az adott dokumentumban lévő idézetek forrását.
- Tartalomellenőrzésre is alkalmas, amennyiben létrehozunk egy internetes adatbázist, ahová szerzői jog alatt lévő dokumentumokat fel lehet vetetni. Esetleg a program figyelmeztethet az illegális dokumentumokra, vagy azokat automatikusan el is távolíthatja.
- Automatikus kereszthivatkozásokat létrehozó szoftver is elképzelhető a segítségével, amely egy adott bekezdéshez automatikusan kapcsol egy másik dokumentumban lévő hasonló, vagy vele azonos bekezdést.
- Annak megállapítására is alkalmas lehet, hogy adott dokumentum milyen nyelven vagy nyelveken íródott. Egy adatbázist feltöltünk különböző nyelvű dokumentumokkal (minden nyelvhez egy szószedet), és a kimenet megadja, hogy az adott dokumentum (esetleg részdokumentum) melyik nyelvű mintafájlhoz milyen mértékben hasonlít.

Látható, hogy a szövegek hasonlóságát kereső rendszernek számtalan felhasználási területe van. Mindemellett nyilvánvalóan a kutatások leghangsúlyosabb motivációja a plágiumszűrés.

2.3 A másolatkereső rendszerek felépítése

Ahhoz, hogy értékelni tudjuk a daraboló és tömörítő eljárásokat, tudnunk kell, hogy milyen helyet foglalnak el a hasonlóságkeresés folyamatában.

A legelső lépés egy ilyen programban a dokumentumok beszerzése. Mivel ehhez a felhasználáshoz a formázási paraméterekre nincs szükség, ezért a legegyszerűbb egy sima szövegfájl (txt) használata. Minden olyan dokumentum, amelyik nem ilyen formában található, egy ezt megelőző lépésben konvertálásra kerülhet.

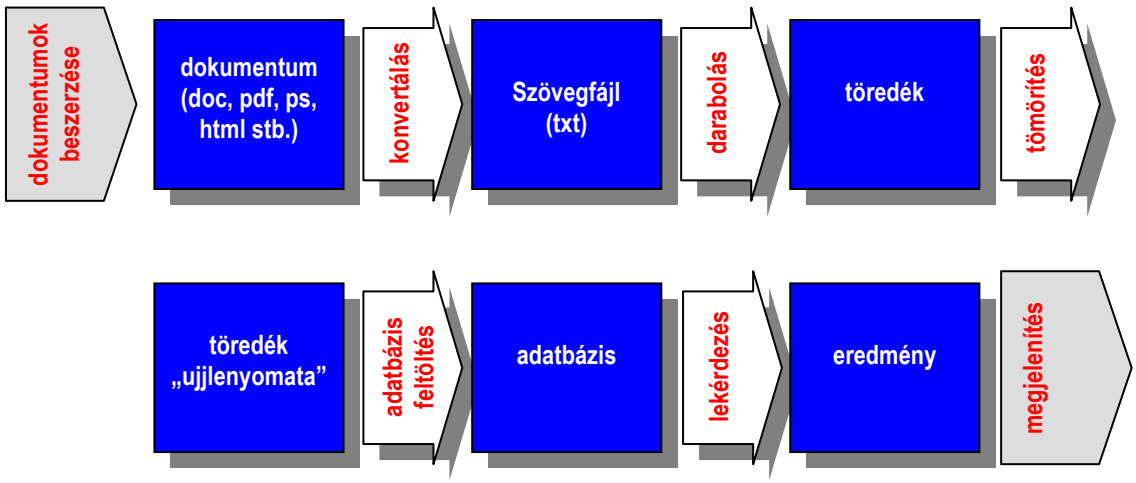
A szövegfájlokat fel kell darabolni kisebb részekre, úgynevezett töredékekre, majd az ezt követő lépésben a töredékek eltárolásra kerülnek egy adatbázisban. Mivel ezek a töredékek sok helyet foglalnak el, ezért nem az eredeti töredék kerül eltárolásra, hanem annak egy úgynevezett „ujjlenyomata”. Ezt egy megfelelő tömörítő eljárással kapjuk az eredeti töredékből.

Az adatbázis feltöltése tetszőleges számú lépésben történhet, ehhez minden új dokumentumot fel kell darabolni, majd a töredékek ujjlenyomatát el kell tárolni. A lekérdezést is akármikor elvégezhetjük, akár minden újonnan beérkezett dokumentum eltárolása után is.

Ha később kíváncsiak vagyunk arra, hogy két dokumentum között van-e egyezés, csak le kell kérdeznünk az adatbázisból, hogy hány közös töredéke van ezen két dokumentumnak.

Amennyiben rendelkezésünkre állnak az eredeti dokumentumok, a felhasználó dolgát megkönnyítve, például a hasonlónak ítélt fájlokat egymás mellé téve, vizualizálhatjuk is eredményünket.

Az 1. ábra a teljes folyamatot ábrázolja.



1. ábra: Másolat kereső rendszerek felépítése

2.4 Különböző darabolási eljárások

Az M2. mellékletben található egy-egy illusztratív példa az itt ismertetett darabolási eljárásokra. Az alább felsorolt darabolási eljárásokkal részletesebben a [Bae99], [Shi95], [Shi96] publikációk foglalkoznak.

2.4.1 Szavas darabolás

A szavas darabolás (word chunking) során n darab szó kerül egy töredékbe. A szöveget úgy osztjuk fel, hogy n szavanként új töredéket kezdünk.

Azaz az első szótól az n-edikig tart az első töredék, az $(n+1)$-ediktől a $2n$-edikig a második, és így tovább. Ennek az algoritmusnak viszont van egy óriási hátránya. Ha két szövegben van egyezés, de ezt az egyezést az előtte lévő tartalom miatt nem ugyanott kezdjük darabolni, akkor az eljárás nem fogja megtalálni az egyezést. Például ha egy dokumentum csak abban különbözik egy másiktól, hogy a címe nem két, hanem három szavas, már nem tudja

kimutatni az egyezést. Ezt „fázis-problémának” nevezzük. A fázis-probléma kiszűrésének egyik módja az átlapolódó szavas darabolás.

A szavas darabolással a továbbiakban, e hiányossága miatt, nem foglalkozunk, csak a teljesség miatt és az átlapolódó szavas eljárás bevezetésének jogossága miatt említettük meg.

2.4.2 Átlapolódó szavas darabolás

Az átlapolódó szavas darabolási eljárás (overlapped word chunking) hasonlít a szavas daraboláshoz, azzal a különbséggel, hogy itt minden szónál kezdődik egy új töredék, amely úgyszintén *n* darab szóból áll. Ezzel az eljárással ki tudjuk szűrni a szöveg esetleges eltolódását. Ebből természetesen az is következik, hogy minden szó *n* darab töredékben lesz benne és, hogy a szavas daraboláshoz képest – ahol csak minden *n-edik* szónál kezdődött el egy darab – itt *n*-szer annyi darab keletkezik.

Az átlapolódó szavas darabolás n=1 esetében azonos a szavas darabolás n=1 beállításával, és azt eredményezi, hogy minden szó egy külön töredéket alkot.

2.4.3 Mondatonkénti darabolás

Kézenfekvőnek tűnhet, hogy a szövegben lévő mondatok alkossák a töredékeinket (sentence chunking). Azonban az elsőre magától értetődő megoldás felvet néhány problémát, hiszen a mondathatár megállapítása nem egyértelmű kérdés. Azok a mondatok például, amelyek rövidítést tartalmaznak, több töredékre fognak széthullani. Például Franz Kafka A per c. művéből való mondat: „Valaki megrágalmazhatta Josef K.-t.” két töredékből áll. Ugyancsak két töredékből fog állni a következő mondat (amennyiben a vesszőt nem tekintjük mondathatárnak): „Nem azért, hogy megtudjon valamit, hanem, hogy elmozdítsa K.-t.” Látható, hogy a „-t.” töredék kétszer fog

szerepelni az adatbázisunkban, amit rendszerünk egyezésként fog érzékelni, holott a két mondat teljesen különböző.

A mondatonkénti darabolásnak – a teszteltek közül egyedüliként – nincsen paramétere. Kísérleteztünk azzal a „paraméterezési" lehetőséggel, hogy a vesszőt mondatvégnek számítottuk az egyik esetben, és nem vettük figyelembe a másikban. Ennek problémának a részletes kifejtése a 4.2.1. fejezetben látható.

2.4.4 Hash kódon alapuló darabolás

A hash kódon alapuló darabolási eljárás (hashed breakpoint chunking) is paraméterezhető, paraméterét jelöljük *n*-nel. Ez a darabolási eljárás, egy egyszerű és gyors függvényt (hash függvény) használ annak megállapítására, hogy mely szavak legyenek a töredékek határai. Ehhez minden szóra kiszámítunk egy számértéket, esetünkben a szó betűinek ASCII kódjait összeadjuk. Amennyiben ez a szám maradék nélkül osztható *n*-nel akkor ez a szó töredékhatár. Az eljárásból következik, hogy amennyiben egy szó egyszer töredékhatár, akkor mindig is az lesz. A töredékhatár után álló szó lesz a következő töredék első szava, és az első olyan szó zárja le a töredéket, beleértve a kezdő szót is, amelyik értéke maradék nélkül osztható *n*-nel, azaz töredékhatár.

Mivel nem tudjuk garantálni, hogy egy szöveg két változata esetében a mondatkezdő nagybetűk is megegyezzenek, hisz manapság divat csupa kisbetűvel írni, ezért minden karaktert kisbetűsre változtatunk, és csak ezután számítjuk ki a szavak értékét.

A hash kódon alapuló darabolásra is, akárcsak az átlapolódó szavas darabolásra, elmondható, hogy n=1 esetében, azonos a szavas darabolás n=1 beállításával, és azt eredményezi, hogy minden szó külön töredékbe kerül. Ez érthető is, hiszen akármilyen értéket is kapunk egy szóra, eggyel biztos, hogy maradék nélkül osztható, azaz minden szó töredékhatár lesz.

2.5 A töredékek tömörítése hash kódolással

A dokumentum kezelő és feldolgozó rendszerek megvalósításaiban igen fontos helyet kap a szövegtárak méreteiből fakadó korlátok kezelése. Különösen igaz ez az Internetes alkalmazásokra. A hash algoritmus régi és jól bevált módja szövegek kódolásának. Bár a fő alkalmazási területe a kriptográfia, de szövegek általános felhasználású tömörítésére is kitűnően alkalmazható.

Az általunk használt MD5 (Message Digest 5) algoritmust [RFC1321] kifejezetten kriptográfiai célra fejlesztették ki, szem előtt tartva a gyorsaságot is. Úgy találtuk, hogy ez az egyszerűen alkalmazható, de gyors algoritmus megfelel céljainknak.

Az algoritmus teljesen publikus kódú eljárás (kódját [RFC1321] tartalmazza), ezért különösen kényelmes a használata számunkra, hiszen tetszőlegesen a feladatunkhoz szabhattuk. Egyik ilyen módosításunk a hash kód hosszának változtatása volt. Úgy találtuk, hogy az eredeti 128 bites hossz a mi alkalmazásainkban indokolatlanul hosszú. Kísérleteink egyik ága ezért a megfelelő hosszúság megtalálását célozta. Nyilvánvalóan rövidebb hash érték kisebb adatbázist és könnyebb, gyorsabb kereshetőséget eredményez, azonban megnöveli a hibák valószínűségét. Kellően sok hash érték mellett ugyanis elkerülhetetlen, hogy elő ne forduljon, hogy azonos hash értékek esetén nem azonosak a lekódolt töredékek. Ekkor két különböző töredékből az algoritmus ugyanazt a hash értéket állította elő. Ezeket az eseteket hamis pozitív esetnek (false positive) nevezzük. Amennyiben rövidítjük a hash kód hosszát, nyilvánvalóan egyre több ilyen esetünk lesz, valamint ezzel együtt természetesen a dokumentumok hasonlóságvizsgálatánál egyre több hibás hasonlóságot fogunk találni. Például 16 bites kód esetén mindkét következő szöveg azonos kódot fog eredményezni:

„harslem eric and ron stoughton rand ucsb network graphics experiment”

„rb9 57”

a mindkettőnek megfelelő hash kód (hexa): 1D87

Az MD5 algoritmus részletesebben Hodász Gábor diplomamunkájában [Hod02] kerül kifejtésre, míg a tesztelések részletes leírását és az eredményeket lásd a 4.3. fejezetben.

2.6 Szöveges dokumentumok tárolása adatbázisban

A különböző kutatások során számos megoldás alakult ki a dokumentumok töredékeinek tárolására. Napjainkban a leginkább elterjedt és használt struktúrák a következők: invertált fájlok, vagy invertált indexek (inverted files, inverted indices); utótag fák, vagy utótag tömbök (suffix trees, suffix arrays); valamint aláírás fájlok (signature files) [Bae99], [Sal92], [Bri99].

Ebben a fejezetben áttekintést adunk a legelterjedtebb szövegtárolási megoldásokról, részletesebben kifejtve az általunk is alkalmazott invertált fájlok megoldást.

2.6.1 Utótag fák és utótag tömbök

Az utótag fák vagy tömbök szerint megvalósított adatbázis támogatja az összetett kereséseket, mint amilyen a kifejezés vagy szomszédsági keresés. A hátrányuk az adatbázis felépítésének nagy időigénye. Az utótag fák emellett még nagy tárigénnyel is rendelkeznek.

Az utótag fák felépítésének alapgondolata, hogy a szöveget, mint egyetlen hosszú karakterláncot tekinti. A szöveg minden pozíciója így egy szöveg utótagnak tekinthető (azaz az a szöveg, ami ebből a pozícióból a végéig tart). Nem nehéz belátni, hogy így minden utótagot pontosan meghatároz a pozíciója. Az így felépített keresőfát nevezzük utótag fának. Természetesen a szöveg nem minden pontját szükséges indexelnünk, csak az előre meghatározottakat (mint például a szavak kezdőbetűit)

Az utótag tömb egy ennél kevésbé tárigényes megoldást nyújt. Olyan egyszerű tömbről van szó, mely tartalmazza az összes szövegbeli mutatót, betűrendben. Evvel az egyszerűsítéssel a tárigény igencsak lecsökken, és az invertált fájlok tárigényének nagyságrendjében lesz, azonban a keresési idő megnő és nem lesz lineáris [Bae99], [Bri99].

2.6.2 Aláírás fájlok

Bár az aláírás fájlok igen jó tulajdonsággal rendelkeznek a keresési komplexitás terén, a legtöbb alkalmazásból már kiszorította a lentebb taglalt invertált fájl.

Az aláírás fájl alapgondolata, hogy egy hash eljárással a szavakat bit maszkká képezi le. A szöveget blokkokra osztva az egyes blokkokhoz is maszkot rendel. Az aláírás fájl nem más, mint a blokkok maszkjainak listája, mutatókkal kiegészítve. Amennyiben egy szó megvan egy blokkban, akkor minden bit, mely a szó maszkjában 1, a blokk maszkjában is 1. Valamint megfordítva: ha egy bit 1 a kereső szó maszkjában, de nem 1 a blokk maszkjában, akkor a szó nincsen a blokkban. Így egy szó keresése a szövegben a következőből áll: hash kódolni kell a kereső szót, majd összehasonlítani a blokkok maszkjaival. Amennyiben az egyszerű bitenkénti ÉS művelet megfelelő eredményt ad, a szó több mint valószínű megtalálható az adott blokkban (természetesen van hibája a kódolásnak).

2.6.3 Invertált fájlok

Az invertált fájl (vagy invertált index) általánosságban szó alapú indexelő eljárás, melynek célja a keresési idő csökkentése [InfRetri]. Alapvetően két elemből áll: a szótárból és az előfordulásokból. A szótár tartalmazza a szövegek összes szavát, az úgy nevezett „stopword”-ök, az általánosan használt, sűrűn előforduló szavak kiszűrése után. Minden szóhoz tároljuk azt a helyet (pontos helyet vagy blokkot), ahonnan származik, ez a lista lesz az előfordulások listája. Meglepő, de a szótár mérete csupán $O(n^{\beta})$ nagyságrendben lesz, ahol n a szöveg szavainak száma, β pedig a szövegtől függő konstans 0 és 1 között, a gyakorlatban 0,4..0,6 között [Bae99]. Az előfordulások listája azonban már $O(n)$ nagyságrendjében van, hiszen minden szó minden előfordulása bekerül a listába.

A szükséges tárhelyet csökkenteni lehet blokkok alkalmazásával. A szövegtárat blokkokra osztjuk fel, és az előfordulások csak azt a blokkot mutatják meg, ahol a szó előfordul. Alkalmazhatunk fix hosszúságú felosztást is, de használhatjuk a szövegtár természetes felosztásait is, mint fájlok, dokumentumok, weblapok stb. A fix hosszúságú blokkok előnye, hogy keresési időben hatékonyabbak, mint a természetes osztású blokkok. Nagyon változó blokkok esetén ugyanis többször keresünk nagy blokkban, mert ezek többször egyeznek a keresési feltétellel.

Amennyiben az index a szó pontos helyét tárolja, akkor „teljes invertált indexről" beszélünk, amennyiben csupán a fájlt adja meg, amelyben szerepel, akkor „invertált fájlról" beszélünk. Amint azt láttuk, ez valójában pusztán a blokkok kialakításának kérdése, ezért több irodalom, így [Bae99] sem tárgyalja külön.

A keresés az invertált fájlok között alapvetően három lépésből áll:

1. Keresés a szótárban: a kívánt szó kiszűrése a szótárból
2. Az előfordulások visszakeresése: a megtalált szavak előfordulásainak kinyerése
3. Az előfordulások feldolgozása a kívánt felhasználás szerint (közelségi vizsgálat, Boolean operációk stb.).

Az invertált fájlok nem támogatják a kontextus szerinti keresést, így nehézkes bennük frázist, kifejezést, szomszédsági összefüggést keresni. Azonban az egy szavas, egyszerű keresésekre a leghatékonyabb eszköz.

3 A tesztekhez használt prototípus komponensei

A feladat elvégzéséhez három programot kellett megvalósítani, melyeket Microsoft Visual C++ környezetben fejlesztettük. E fejlesztői környezet előnye a teljes objektum-orientáltság, a rugalmasság, valamint a széleskörű beépített lehetőségek tárháza. Az adatbázist Microsoft Access 2000 segítségével valósítottuk meg, amelyet programunk a CRecordset MFC (Microsoft Foundation Classes) osztály felhasználásával, ODBC (Open Database Connectivity) illesztőprogramon keresztül ér el. Az MFC univerzálisan alkalmazható és testre szabható függvények gyűjteménye, melyek kényelmessé és gyorssá teszik a szoftverfejlesztést. Az ODBC illesztőprogram olyan eszköz, melynek segítségével az adatbázis komponens az egyéb komponensekkel univerzális interfészen keresztül kommunikálhat.

3.1 Daraboló komponens

Az első program a darabolásokat végezi. A program beolvassa egy könyvtárból az összes szövegfájlt, majd kiszűri belőlük a speciális karaktereket, mint a zárójel, százalékjel, kötőjel stb. Ezeket szóközzel helyettesíti. Amennyiben nem mondatonkénti darabolást végzünk a mondatvégi írásjeleket is kiszűri. Azért, hogy ne legyen több szóköz a szavak között, a többszörös szóközöket egyszeresre cseréli.

Az így kapott szöveget átalakítja csupa kisbetűsre, majd átadja a megfelelő daraboló függvénynek. A daraboló függvények a 2.4. fejezetben leírt funkcionalitásuknak megfelelően lettek megvalósítva. A függvények helyes működését rövidebb dokumentumokon kézzel ellenőriztük. A daraboló függvények forráskódja megtalálható az M9. mellékletben.

A daraboló eljárásoktól kapott töredékeket átadja egy függvénynek, amely visszaadja a töredék MD5 kódját. A hamis pozitív tesztekhez elengedhetetlen, hogy ismerjük mind a töredéket, mind az ujjlenyomatát, ezért a kimeneti fájlba mindkettőt beleírja a program. Minden bemeneti fájlhoz pontosan egy kimeneti fájl tartozik, amelynek egy, a bemeneti fájl nevéből képzett, egyedi nevet ad.

Ezen kívül generál még egy kimeneti fájlt, amibe a futtatás statisztikai adatait írja (az M3. mellékletben látható egy ilyen fájl). Megtalálható benne a futtatás kezdési és befejezési időpontja, a bemeneti illetve kimeneti fájlok helye, valamint futtatás egyéb paraméterei. Azon fájlok neve is megtalálható benne, amelyeket feldolgozott. A bennük lévő összes szó száma, a belőlük képzett töredékek száma, illetve az ezekből kiszámolt átlag töredékhossz is a fájlok mellé van írva. Utóbbi három statisztikai adat nem csak a fájlokra külön, hanem a kimeneti fájl végén a teljes feldolgozott adatra is megtalálható.

3.2 Másolatkereső komponens

A másolatkereső program alapvetően két részből áll: a dokumentum-regisztráló, valamint a másolatkereső elemből.

A dokumentum regisztráló rész a daraboló program kimeneteként keletkezett hash értékeket adatbázisba rendezi (egyszerűsített invertált fájl struktúrában). Ehhez összegyűjti az egy adott fájlban levő hash értékeket, az azonosakat kiszűri és számlálja, majd elhelyezi őket az adatbázis szótár részében. Ezzel együtt a fájlt regisztrálja a fájlok táblájában. A rendszer figyeli, hogy egy adott fájlt vagy URL-t csak egyszer lehessen regisztrálni, hiszen minden dokumentumnak egyedi, egy-egy értelmű azonosítója kell, hogy legyen.

A másolatkereső algoritmus bemenetként egy adott dokumentumnak a daraboló program által kiadott kimenetét kapja, tehát a dokumentumban szereplő töredékek hash értékeit. Ezekre a hash értékekre szűrést végez az

adatbázis szótár-táblájában, egyenként kigyűjtve az egyes hash értékekhez tartozó más dokumentumokat. Így végighaladva a vizsgálandó dokumentum minden egyes hash értékén kiszámítja az egyes regisztrált dokumentumokkal való egyező töredékek számát, illetve ebből a százalékos egyezést. Kimeneteként egy szöveg típusú fájlt kapunk, melyben felsorolja az adott dokumentumnak a regisztrált dokumentumokkal való egyezését.

3.3 Hamis pozitív érték elemző komponens

A hamis pozitív érték kereső program statisztikai elemző program, mely nem lesz része a későbbi hasonlóság kereső rendszernek. Segítségével vizsgáljuk a töredékek hash kódolással való rövidítésének lehetőségeit, korlátjait.

A program működése a következő: bemenetként kapja a daraboló program kimenetét, azaz az egyes dokumentumokban levő töredékek hash kódolt értékét, valamint magát a töredéket (akárcsak a másolatkereső komponens). Ezekből az értékekből egyetlen táblából álló adatbázist épít, melyben a hash érték és a hozzá tartozó töredék lesz egy rekord. Kísérletünk arra irányult, hogy a különböző hosszúságú hash kódokat a keletkező hamis pozitív értékek függvényében vizsgáljuk. Így a beállítható paraméter a vizsgálni kívánt hash kód hossza (bitmélysége). A program kimenete egy szűrés eredménye, melynek segítségével kikeressük az adatbázisból azokat a rekord-párokat, melyekben a töredék különbözik, a hash érték ellenben azonos. Ezeket az eseteket hamis pozitív eseteknek nevezzük, hiszen itt a másolat kereső program olyan egyezést fog találni, mely a valóságban nem létezik (erről részletesebben a 4.3. fejezetben).

A lekérdezés SQL formában a következőképpen néz ki:

```
SELECT hash32_1.hashvalue, hash32_1.chunk,
hash32_2.hashvalue, hash32_2.chunk
FROM hash32 AS hash32_1, hash32 AS hash32_2
WHERE (((hash32_1.hashvalue)=(hash32_2.hashvalue)) AND
((hash32_1.chunk)<(hash32_2.chunk)));
```

A program kimenete szöveges fájl lesz, melyben felsorolja az egyes hamis pozitív párokat.

4 Teszteredmények és értékelések

4.1 Tesztdokumentumok kiválasztása

A daraboló eljárásokat nagy mennyiségű, különböző méretű dokumentumon kellett tesztelni. Ehhez, az Interneten megtalálható RFC dokumentumokat [RFC] használtam, nagy mennyiségük (2590 darab), és a bennük lévő kisebb-nagyobb átfedések miatt. Ezek segítségével optimalizáltam a daraboló eljárásokat futási idő alapján.

A különböző eljárások összehasonlításához a daraboló eljárások által adott kimenetet bele kellet tenni egy adatbázisba, majd ott összehasonlításokat végezni. A tesztelés teljes kimenetét át kell tudni látni, ez azonban több száz dokumentum egymással való összehasonlítása esetén lehetetlen feladat. Ezért ezeket a teszteket kis mennyiségű próbadokumentumon végeztem el. Ezeknek az alábbi követelményeket kellett feltétlenül teljesíteniük:

1. Legyen olyan dokumentum, amely teljesen megegyezik egy másik dokumentum egyik darabjával.
2. Legyenek hasonló dokumentumok, kisebb nagyobb különbségekkel.
3. Legyenek teljesen különböző dokumentumok.

Az első követelmény az idézetek illetve beemelések kimutatásánál fontos, a második azért lényeges, hogy kicsit megváltoztatott dokumentumot is ki tudjunk mutatni, míg a harmadik az úgymond ellenőrző csoport. Hiszen nem elég, ha megállapítjuk, hogy két dokumentumban van valami hasonló, hanem ezeket valami szerint értékelnünk is kell, és teljesen egyértelműen el kell, hogy tudjuk különíteni a nem hasonlók közül.

A második követelménynek teljes mértékben megfelel a Szent Biblia és annak különböző fordításai. A Biblia három magyar fordításával dolgoztunk, ezek: Károli Gáspár fordítása [Kár01], a katolikus fordítás (Szent István

társulat) [BD01], és a református fordítás [Ref93]. Ezek tekinthetők úgy, mint egymás átiratai, habár ugyanannak a forrásnak a magyarra fordításai.

Az első és a harmadik követelményt a különböző részek (könyvek) és idézetek (versek) megfelelő válogatásával tudjuk elérni.

Ezek alapján az alábbi részekre esett a választásunk:

A Szeretet himnuszát (Pál első levele a Korinthusiakhoz 13. vers) mindhárom fordításból betettük egy-egy dokumentumba. A tíz parancsolatot (Mózes második könyve 20. vers) is mindhárom fordításból betettük egy-egy dokumentumba. Azért, hogy legyen olyan dokumentum, amely teljesen megegyezik egy másik dokumentum egyik darabjával (első követelmény) hozzávettük a katolikus fordításból Pál első levelét a Korinthusiakhoz (a teljes könyvet). Illetve, hogy legyenek kevés vagy semmilyen hasonlóságot felmutató dokumentumok (harmadik követelmény) a katolikus fordításból kiválasztottuk Pál második levelét a Korinthusiakhoz, és Mózes első könyvének az elejéből egy nagyobb részt. A dokumentumok részletes leírása és elnevezése megtalálható az M4. mellékletben.

4.2 Darabolási eljárások összehasonlítása

Ebben a részben a három, általam tesztelt, darabolási eljárás részletes összehasonlítása található, különböző szempontok szerint. Ezeket a szempontokat mindenképpen figyelembe kell venni, ha meg szeretnénk tudni, hogy számunkra melyik a legmegfelelőbb eljárás.

4.2.1 Keletkező töredékek mennyisége

Ahhoz, hogy két dokumentum vagy szakasz hasonlóságát meg tudjuk állapítani, kell, hogy mindkettőből keletkezzenek azonos töredékek, hiszen mi csak ezt az egyezést tudjuk később kimutatni. Minél több azonos töredék van két fájlban, annál nagyobb az esélye, hogy a feldolgozóprogram pozitívnak ítéli meg hasonlóság szempontjából. Ezért amennyiben túl kevés töredékkel dolgozunk, nagy az esélye, hogy átsiklunk bizonyos egyezések felett. Túl sok töredéket sem érdemes használni, mert az adatbázis mérete természetesen nagyban függ attól, hogy hány töredéket tartalmaz. Az adatbázis-lekérdezések sebességét pedig erősen befolyásolja, vagy befolyásolhatja az adatbázis mérete. Ezért alapvető követelmény a darabolási eljárásokkal szemben, hogy lehetőleg minél kevesebb töredéket gyártsanak. Itt máris megfigyelhető egy ellentét, hiszen amíg a feldolgozás gépigénye szempontjából előnyös, ha kevesebb töredék van, addig a sikeres detektáláshoz minél több töredékre van szükségünk.

Mivel az adatbázisba minden töredék azonos hosszúságú számként kerül be, lényegtelen, hogy eredetileg milyen hosszú volt, csak az számít, hogy azonos dokumentumból melyik eljárás hány töredéket generál. A továbbiakban azért, hogy egy rövid képlettel megadható legyen a keletkezett töredékek mennyisége, jelöljük *w*-vel a dokumentumban található szavak számát, valamint n-nel a daraboló eljárás paraméterét. Most vegyük sorba a különböző eljárásokat.

Átlapolódó szavas darabolás

Az átlapolódó szavas darabolás, pontosan *w-(n-1)* töredéket generál, hisz, az utolsó *(n-1)* szót kivéve, minden szónál kezdődik egy töredék. Ez nagyságrendileg ugyanannyi töredéket jelent, mint amennyi dokumentumban lévő szavak száma, hiszen míg *w* értéke több ezertől több százezerig mozoghat, addig *n* értéke általában nem haladja meg a tizenötöt. Azért, hogy később jobban átlátható legyen az összehasonlítás, a továbbiakban használjuk a *w* közelítő értéket.

Mondatonkénti darabolás

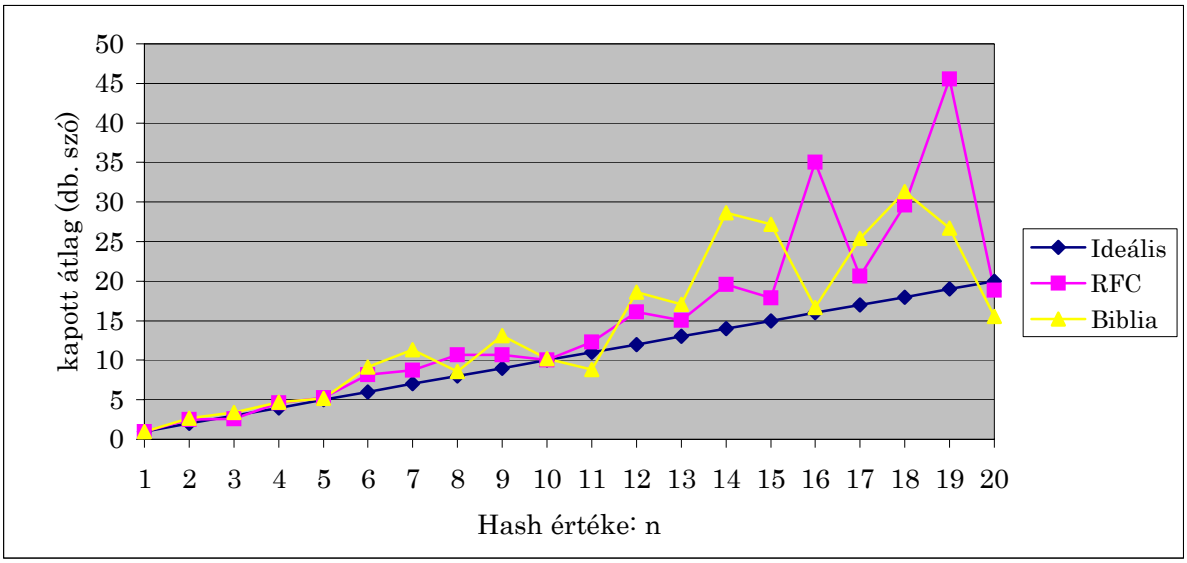
A mondatonkénti darabolás esetén elég nehéz még csak közelítő értéket is mondani. Kísérleteink során találkoztunk már olyan dokumentummal is, amiben a mondatok átlagos hossza nem haladta meg a négyet, más dokumentumok esetén viszont a tízes átlag se számít ritkaságnak. Azaz a töredékeink száma *w/4*-től *w/10*-ig akármi lehet. Sőt, szélsőséges esetben még ennél többet, illetve kevesebbet is kaphatunk. Ez nagyon megnehezíti mind a hasonlóságok biztos kimutatását, mind annak a kiszámítását, mekkora adatbázis, illetve milyen teljesítményű szerver kellhet később az ilyen elven működő keresőrendszerhez. Mint az általános leírásban már említettük, próbálkoztunk a mondatonkénti darabolás esetében úgy befolyásolni az átlagos töredékhosszt, hogy a vesszőt mondatvégnek számítottuk az egyik esetben és nem vettük figyelembe a másikban. Ez a megoldás néha praktikus lehet, de semmiképp se kapunk univerzális programot. Hiszen míg például egy magyar szövegben, ha nem vesszük mondatvégnek a vesszőt, legtöbb esetben túl hosszú töredékeket kapunk, addig például egy felsorolásokkal tarkított szövegben könnyen négy alá megy az átlagos töredékhosszunk, ha a vesszőt is mondatvégnek vesszük.

Nagyon jól illusztrálják ezt a problémát az általunk vizsgált dokumentumok is. A bibliai idézetek, ha a vesszőt mondathatárnak vesszük, 3,6 szót adnak átlagos töredékhossznak. Ha viszont a vesszőt nem vesszük

mondathatárnak, akkor 11,7 lesz az átlag. Ugyanez a két szám az RFC dokumentumok esetében 3,5 illetve 13.

Hash kódon alapuló darabolás

Teljesen más a helyzet a hash kódon alapuló darabolás esetében. Az RFC dokumentumokra, és a bibliai idézetekre lefuttattuk a tesztprogramunkat, különböző paraméterekre, ennek az eredménye látható az 1. diagramon.



1. diagram: Töredékek átlagos hossza
hash érték alapú darabolás esetében, a paraméter függvényében

Mint az a diagramból is jól kivehető, a felhasznált dokumentumoktól erősen függ, az átlagos érték. Ez érthető is, hiszen elképzelhetőek olyan stílusú vagy nyelvű szövegek, ahol egy adott hash értéken a leggyakoribb szavak pont töredékhatárt alkotnak. Persze elképzelhető az is, hogy nagyon ritkák az adott paraméteren a töredékhatár szavak, és ha ránézünk az ábrára, láthatjuk, hogy ez a gyakoribb. Természetesen, minél nagyobb a hash paramétere, annál kevesebb szó lesz töredékhatár, és annál nagyobb az esélye, hogy a gyakori szavak nem esnek bele, azaz nagyobb átlagértéket kapunk, mint a hash értékünk. Ehhez tudni kell még, hogy *n* értéke elvileg akármilyen pozitív szám

lehet, a gyakorlatban viszont 6 és 15 között mozog tipikusan. Ezeket mind egybevetve jó közelítést kapunk az egy dokumentumban található töredékek számára a *w/n* értékkel.

Ennek tudatában már látható, hogy a hash kódon alapuló darabolás az átlapolódó szavashoz viszonyítva jelentősen (átlagosan körülbelül *n*-szer) jobb helykihasználást jelent.

Ha a leírtakhoz még hozzávesszük a kevesebb töredék esetén, az adatbázis feltöltésénél és a lekérdezéseknél elérhető időmegtakarítást, akkor egyértelműen a hash kódon alapuló darabolás a legjobb ezen szempontok alapján.

4.2.2 Futási idő

Nem meglepő, hogy a mondatonkénti darabolás és a hash kódon alapuló darabolási eljárások gyorsabbak, mint az átlapolódó szavas darabolás. Hisz míg utóbbi (paraméterétől függetlenül) közel annyi töredéket generál, mint amennyi szó a dokumentumban van; addig a mondatonkénti darabolás szövegtől függően három-tíz szavanként, a hash kódon alapuló darabolás pedig átlagban *n* szavanként képez egy töredéket.

Az RFC dokumentumokra lefuttattuk a különböző darabolási eljárásokat, és míg az átlapolódó szavas darabolás átlagban 28 percig futott, addig a hash kódon alapuló darabolás paramétertől függően 5-től 22 percig, a mondatonkénti darabolás pedig 10 percig tartott. Ezek az adatok egy Intel Pentium II 900Mhz-es processzorral valamint 256 MB memóriával rendelkező PC-re vonatkoznak, természetesen az arányok más konfigurációk esetében is hasonlóak lennének. A daraboló komponens egyéb (teszt) funkciókat is ellát, ezért egy tényleges rendszernél ennél kisebb futási idővel számolhatunk.

Az adatbázis kezelésénél is hasonló a helyzet, hiszen az átlapolódó szónak itt is többször annyi töredéket kell regisztrálnia az adatbázisba, mint a másik két eljárásnak. Ez pedig természetesen sokkal több időbe is telik.

A mondatonkénti darabolás és a hash kódon alapuló darabolás között már nem ilyen egyértelmű a helyzet. Ennek főleg a mondatonkénti darabolás már említett bizonytalansága az oka, miszerint nem lehet megmondani, hogy átlagosan hány szó kerül egy töredékbe. Tapasztalataink azt mutatják, hogy a mondatonkénti darabolás kevesebb töredéket generál, de ezt természetesen dokumentuma válogatja.

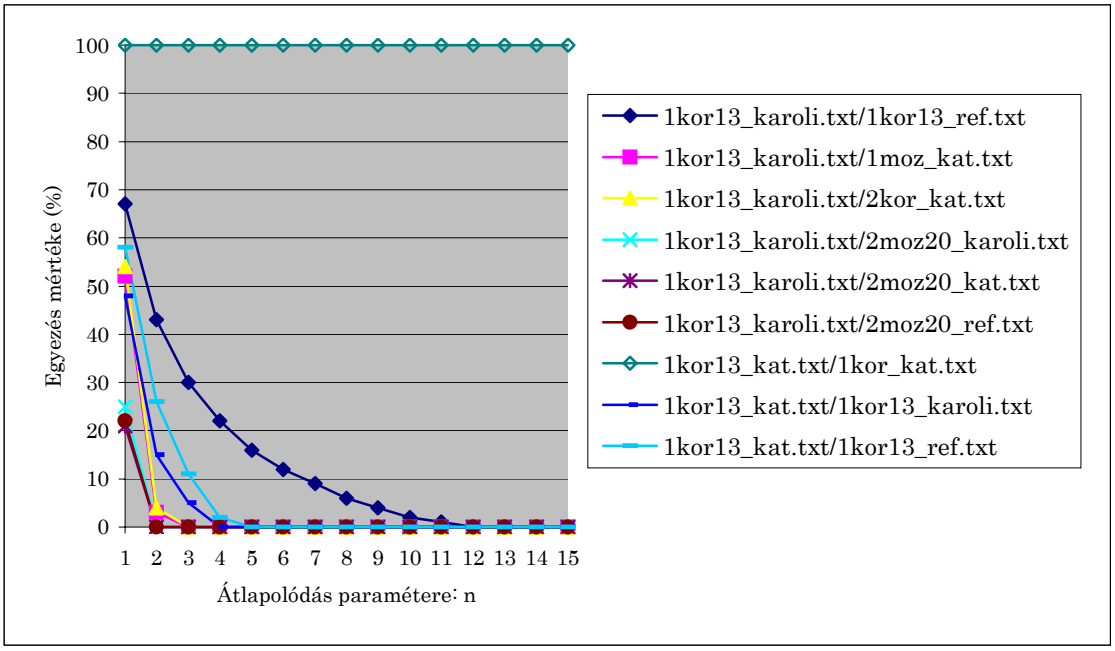
4.2.3 Hasonlóságok kimutatása

Az előző fejezetben már érintőlegesen említettük a paraméterek jelentőségét. Most nézzük meg részletesen, milyen hatással vannak a különböző darabolási eljárások paraméterei a hasonlóság kimutatására.

Azt mindenképpen meg kell említeni, hogy tulajdonképpen lényegtelen, hogy számszerűleg mekkora egyezést mutat ki egy eljárás két dokumentum között, a lényeg, hogy jól elkülöníthetőek legyenek a hasonlóságot felmutató dokumentum-párok a különbözőektől. Egy extrém példával élve, ha egy algoritmus akármely két (nem hasonló) dokumentum között minimum 20 százalék egyezést talál, az még nem jelenti azt, hogy nem használható, hiszen ha a legkisebb tényleges egyezésre is felmegy ez az érték 40-re, akkor jól elkülöníthető eme két eset. Természetesen az ilyen algoritmus csak a hasonlóság meglétének kimutatására alkalmas, a hasonlóság mértéke nem állapítható meg belőle, de ez a legtöbb alkalmazásban elegendő is, hiszen mint már említettük, ezek az eljárások általában egy gyors előszűrők egy részletesebb, lassabb de pontosabb összehasonlításhoz.

Átlapolódó szavas darabolás

Az átlapolódó szavas darabolás paramétere, mint már korábban beláttuk, nincs hatással az adatbázis méretére és a futási időre. Viszont óriási a jelentősége a hasonlóság kimutatásánál, illetve a program aktuális problémára szabásánál. A 2. diagram a fájlok közötti hasonlóságot ábrázolja az átlapolódó szavas darabolás paraméterének függvényében.



2. diagram: Hasonlóság vizsgálata
átlapolódó szavas darabolás esetében, a paraméter függvényében

Mint az a grafikonból is kitűnik, az összes függvény monoton csökkenő, a száz százalékos egyezést kivéve, amely végig megtartja az értékét. Ez a monoton csökkenés könnyen belátható. Vegyünk ehhez alapul egy *x* szavas egyezést két dokumentum között. Ebből átlapolódó szavas darabolás esetén *x-(n-1)* töredék képződik. Minél nagyobb *n* értéke, annál kevesebb töredék képződik és annál kisebb lesz az egyezés. Ez a képlet csak addig használható, amíg *n* értéke nem haladja meg *x*-ét, amennyiben meghaladja, az eredmény

magától értetődően nulla lesz, azaz nem jelzi ki az egyezést. Ezt a tudást nagyon jól fel lehet használni a paraméter meghatározásához.

Ha két szöveget a stílusa illetve a szóösszetételek, kifejezések szempontjából szeretnénk összehasonlítani akkor egy 2-es, 3-as, maximum 4-es érték lesz a célravezető. Hiszen az ilyen szerkezetek általában két-három szóból állnak. Ez nagyon jól megfigyelhető a diagramon is. A sötétkékkel jelölt függvény a Szeretet himnuszának a hasonlóságát mutatja a Károli Gáspár, illetve a református fordításban. Ezt a két változatot úgy is szokták nevezni, hogy új illetve régi fordítású református Biblia. Az egyezés mértéke n=2 esetében 43%, n=3 esetében 30% míg n=4 esetében már csak 22%. Ez annyit jelent, hogy a Károli Gáspár fordításban található töredékek ennyi százaléka található meg a református fordításúban is.

Az M5. mellékletben megtalálható ez a két szövegrészlet valamint a katolikus fordítású is. Jól láthatóak az azonos mondatszerkezetek az első kettőben és az ettől nagyobb mértékben eltérőek a harmadikban.

Amennyiben az ilyen mondatszerkezeteket nem szeretnénk kimutatni nincs más dolgunk, mint, hogy nagyobb értéket adunk a paraméterünknek. Ha egy pár mondatos, folyamatos egyezést is ki szeretnénk mutatni, de nem akarjuk, hogy túl „érzékeny” legyen a keresőprogramunk a mondatszerkezetekre, akkor egy 5-ös 6-os értéket érdemes választani.

Amennyiben csak nagyobb részek egyezésének a kimutatására van szükségünk, jó választás lehet egy 10 feletti paraméter, amely a kisebb hasonlóságokat már nem mutatja ki. Így nem kell ezeknek a későbbi kiszűrésével foglalkozni.

Mondatonkénti darabolás

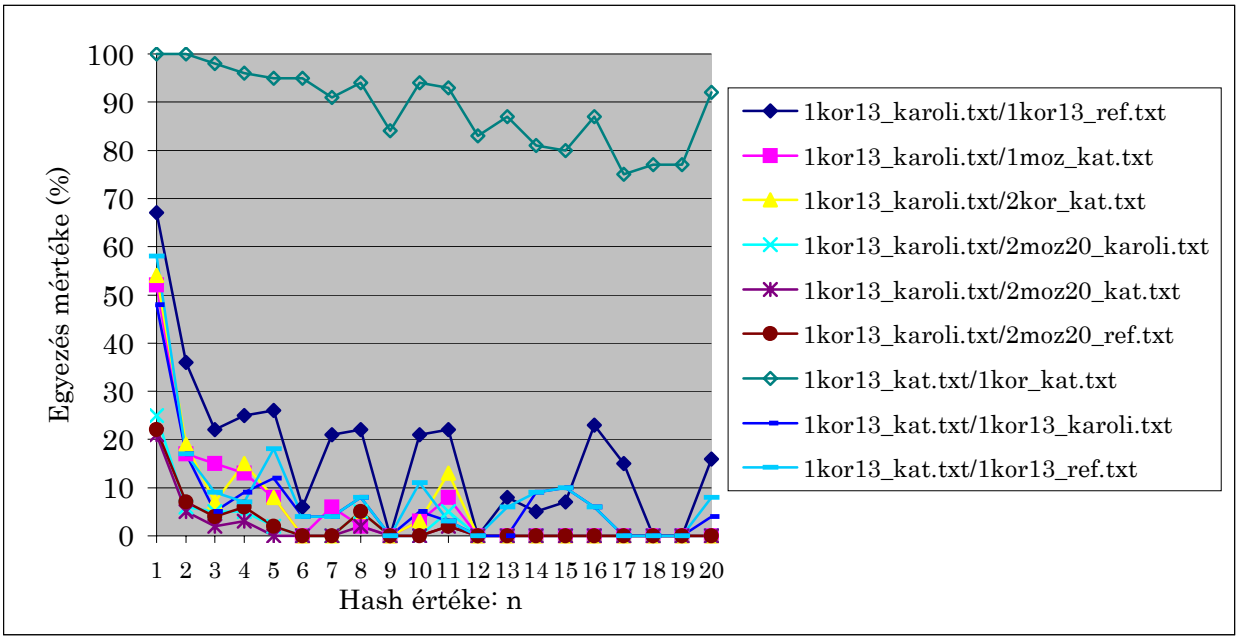
Mint már azt korábban említettük, a mondatonkénti darabolásnak nincsen paramétere. Ez komoly hátránya. Még egy szempontból különleges a mondatonkénti darabolás: nem érzékeny a mondatok felcserélésére. Ez triviális, de nagyon lényeges tulajdonsága. A többi algoritmus nem veszi figyelembe a mondatok határát, ezért ha felcseréljük a mondatokat, az is lehet, hogy egyáltalán nem találják hasonlónak a két dokumentumot.

A mondatonkénti darabolásnak ezen tulajdonsága lehet előny is, de hátrány is. Amennyiben kizárólag nagyobb egybefüggő szakaszok egyezése érdekel, akkor a mondatonkénti darabolás nem jó választás, mert ugyan kimutatja az egyezést, de a programunk nem lesz képes elkülöníteni az olyan esetektől, ahol ugyanezek a mondatok megtalálhatóak mondjuk egy több száz oldalas regényben elszórva.

Viszont el lehet képzelni egy olyan felhasználási területet is, ahol ez a tulajdonsága előnyére válhat. Ha például meg szeretnénk tudni, hogy mely dokumentumban idéznek rendszeresen bibliai verseket, akkor azt legegyszerűbben mondatonkénti darabolással lehet megvalósítani. Azzal az apró módosítással, hogy az idézőjelet is mondatvégnek vesszük, ezzel biztosítva, hogy az idézetek külön töredékbe kerüljenek. Ezeket a hasonlóságokat az átlapolódó szavas darabolással is ki tudnánk mutatni, de nem szabad elfeledkeznünk arról, hogy mennyivel több töredék keletkezne abban az esetben.

Hash kódon alapuló darabolás

A hash kódon alapuló darabolásnál nagyon érdekes diagramot kapunk (3. diagram), ha ábrázoljuk a fájlok között fennálló hasonlóságot a paraméter függvényében.



3. diagram: Hasonlóság vizsgálata
hash kódon alapuló darabolás esetében, a paraméter függvényében

Rögtön szembe tűnik, hogy nem monotonok a függvények. Ez igen nagy problémát jelent amennyiben kisebb átfedéseket is ki akarunk mutatni. Például ha éppen olyan paramétert használunk, amelyiknél láthatóan lecsökkennek az értékek, akkor legrosszabb esetben egyáltalán nem kapunk egyezést. Miért van ez a nagy ingadozás? A válasz egyszerű, és tulajdonképpen már a 4.2.1. fejezetben, ahol a keletkező töredékek mennyiségét vizsgáltuk, meg is válaszoltuk. Érdemes összehasonlítani az ott található 1. diagramot a 3-as számúval. Azt vehetjük észre, hogy ahol a 3. diagram ugrásszerűen leesik, ott az 1. diagramon, annak a görbének, amelyiket a bibliai versek feldolgozásából kaptunk, és a töredékek átlagos hosszát ábrázolja, az ideálisnál

magasabb értéke van. Ezek azok a helyek, amelyeken az adott stílusú vagy nyelvű szövegnek az adott paraméter mellett kevés töredékhatár szava van.

Az, hogy ez a két dolog így összefügg nagyon jól felhasználható a megfelelő paraméter megtalálásában. Amennyiben rendelkezésünkre állnak olyan stílusú dokumentumok, mint amelyekkel később az adatbázist is fel szeretnénk tölteni, akkor elég csak a daraboló komponenst lefuttatni ezekre, és megnézni, hogy melyik paraméternél mekkora lett az átlagos töredékhossz. Azok az átlagok, amelyek jóval nagyobbak a paramétereknél, azt jelzik, hogy az adott paraméter nem alkalmas számunkra. A többiből meg az alapján érdemes válogatni, hogy mekkora hasonlóságot szeretnénk kimutatni.

Nagyon kis paraméterek esetén a töredékek száma megközelíti az átlapolódó szavas darabolását, így ezeknél az értékeknél elképzelhető, hogy érdemesebb azt használni, mivel jóval megbízhatóbban ki tudja mutatni a hasonlóságot.

Ha kisebb szakaszok egyezését is ki szeretnénk mutatni, akkor egy 5 körüli érték lehet a legcélravezetőbb. Amennyiben csak hosszabb egyezésekre vagyunk kíváncsiak, ennél nagyobb értékeket kell választanunk. 10 feletti értékeknél viszont már nagyon vigyázni kell arra, hogy egyre nagyobb a valószínűsége annak, hogy az egyező részek pont beleesnek egy-két hosszabb (akár száz szó feletti) töredékbe.

Átlapolódó hash kódon alapuló darabolás

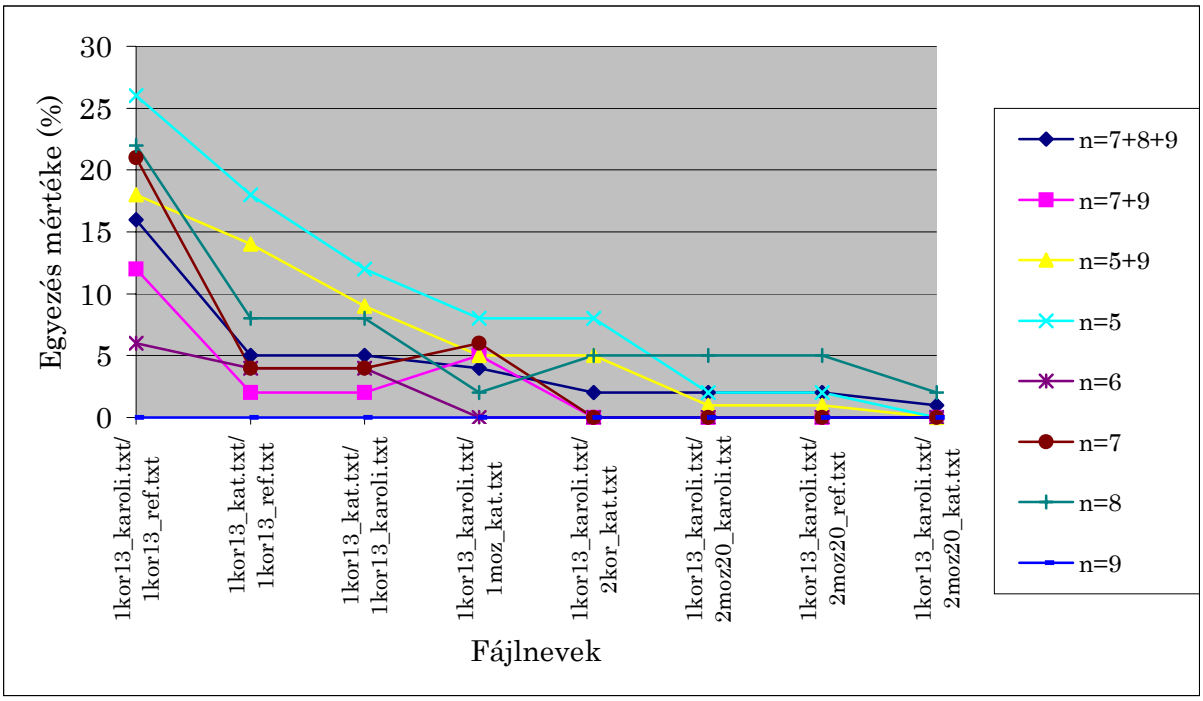
Elképzelhető, hogy olyan adatbázist kell építenünk, amelybe különböző stílusú illetve nyelvű dokumentumok kerülnek. Például ha egy Internetes plágiumkeresőt szeretnénk létrehozni, amely kijelzi az egy oldalnál nagyobb egyezéseket.

Ebben az esetben, az adatok nagy mennyisége miatt, szóba se jöhet az átlapolódó szavas darabolás. A mondatonkénti darabolás meg azért nem jó választás, mert könnyen lehet, hogy egy nagyobb műben lévő mondatok megtalálhatóak egy másikban, teljesen más összefüggésben és elszórva.

Tehát csak a hash kódon alapuló darabolás marad, ezzel viszont az a gond, hogy adott paraméter mellett, nem alkalmas minden szöveg hasonlóságának a kimutatására. Más szóval nem megbízható.

Ezért elgondolkoztam azon, hogy miként lehetne kiküszöbölni ezt az előnytelen tulajdonságát. Ekkor jött az az ötlet, hogy mi lenne, ha több hash értéket használnánk egyszerre, így csökkentve annak a valószínűségét, hogy az adott paraméter nem illeszkedik az adott dokumentumhoz. Ez a megoldás úgy működne, hogy a dokumentumokat kettő, maximum három hash értékkel is feldarabolnánk, majd regisztrálnánk az adatbázisba. Mindegyik változatot külön táblába. Ezek után, ha egy dokumentumhoz hasonlót keresünk, mindegyik táblára lefutatjuk a megfelelő lekérdezést, majd a kapott eredményekből átlagot számolunk, vagy netán a legnagyobb egyezést vesszük alapul. Ez az elgondolás a gyakorlatban is működik, és elég jó eredményeket lehet vele elérni. Ezt a módszert átlapolódó hash kódon alapuló darabolásnak neveztem el, mert a különböző hash értékkel kapott darabok átlapolódnak egymással.

A 4. diagram ezt a darabolási eljárást hasonlítja össze, a sima hash kódon alapuló darabolással. Itt mi nem maximumot képeztünk a kapott eredményekből, hanem összeadtuk az egyező töredékek számát, majd ebből számoltuk ki az egyezés mértékét.



4. diagram: Átlapolódó hash kódon alapuló darabolás
és a hash kódon alapuló darabolás összehasonlítása

Az ábrán jól megfigyelhető, hogy míg a 9-es hash érték sehol se jelzett ki hasonlóságot, addig mindhárom átlapolódó hash kódon alapuló darabolási eljárás ki tudott mutatni egyezést, annak ellenére, hogy a 9-es hash értéket mindhárom esetben használtuk. Matematikai megközelítésből nem helyes, hogy összekötöttük a diagram pontjait, hiszen különböző adatokra vonatkoznak, viszont ez az ábrázolásmód nagyban elősegíti az átláthatóságot. A maximumszámításnak megvan az a kockázata, hogy túl gyakran jelez egyezést dokumentumok között. Ugyan nem mutattunk külön rá, de ahogy bizonyos paraméterek túl kis, úgy bizonyosak túl nagy hasonlóságot mutatnak, mindkét eset ugyanannyira kerülendő.

Az igazsághoz hozzátartozik, hogy az adatbázisunkban természetesen kétszer vagy háromszor annyi töredék lesz, mint egy sima hash kódon alapuló darabolási eljárás esetében. Ez viszont bizonyos körülmények között megengedhető, például ha egy dokumentummásolatokat megbízhatóan detektáló, de nem túl gépigényes rendszert szeretnénk építeni.

Természetesen akármelyik eljárást is alkalmazzuk, a paraméter végleges megválasztása előtt – amennyiben mód nyílik rá – érdemes kisebb mennyiségű példadokumentumra részletes teszteket futtatni. Ha mégis később, használat közben, derül ki, hogy a megválasztott paraméter, vagy eljárás nem alkalmas a feladat megoldására, az egész adatbázist újra kell építeni. Ehhez az összes dokumentumot újra be kell szerezni, és fel kell dolgozni. Ez esetenként igen komoly feladatot és költséget jelenthet.

Mi itt most a tesztekhez olyan tesztdokumentumokat használtunk fel, amelyeknél az egyezés jól kimutatható, akkor is, ha az egyezés mértékét százalékban fejezzük ki. Ez viszont nem mindig alkalmazható.

Vegyünk példának egy többszáz oldalas könyvet, amelyben van egy több oldalas idézet egy másik, hasonlóan hosszú, dokumentumból. Ezt a hasonlóságot a program kisebbnek fogja megítélni (százalékban kifejezve), mint mondjuk két egyoldalas dokumentumban található közös szakaszt. Esetleg már ki se jelzi.

Ezért a kiértékelésnél néha azt is figyelembe kell vennünk, hogy hány töredék egyezik meg a két dokumentumban.

Ezt a két adatot (százalékos és darabszám) nagyon jól fel lehet használni arra, hogy kiszűrjük az adatbázis-lekérdezésnél kapott rengeteg egyezésből a nekünk megfelelőket. Ha például csak több oldalas átfedések érdekelnek, akkor olyan egyezéseket figyelembe se veszünk, amelyek csak pár töredékből állnak. Ugyanakkor, ha olyan dokumentumokat keresünk, amelyeknek több mint a fele egy másik műből átvett idézet, akkor a százalékos kiértékelés lesz a legcélravezetőbb.

A tesztjeink során a teljes bibliai tesztdokumentum-halmazra kimutattuk a hasonlóságokat, különböző darabolási eljárások és paraméterek mellett. Ezekből vettük a példákat ebben a fejezetben a diagramokhoz. A teljes táblázat megtalálható az M7. mellékletben.

4.3 Hamis pozitív hash értékek vizsgálata

A fentebb kifejtett hash kódolással való szövegtömörítéssel igen jó eredményt lehet elérni a szöveg-adatbázis méretének csökkentésében. Lényeges paraméter azonban, hogy az MD5 kódoló algoritmus által előállított 128 bites kódból milyen hosszúságú kódot alkalmazzunk ténylegesen szövegtöredékeink tárolásánál.

Egyrészről minél rövidebb a kód, annál kisebb lesz az adatbázis, valamint a keresések ideje is rövidül. Másrészről azonban megnövekednek az úgynevezett hamis pozitív esetek. Ekkor a kereséseink hibás eredményeket, nem létező hasonlóságot fognak jelezni. Mivel rendszerünk csak előfeldolgozó egysége egy alapos és részletes egyezéskereső programnak, ezért néhány százalékos hamis pozitív érték nem zavarja jelentékenyen az eredményeinket.

A hamis pozitív érték vizsgálatot a következők szerint végeztük:

- Első lépcsőben kis dokumentummennyiségen vizsgáltuk az egyes metódusokkal keletkezett töredékek hash kódolását, különböző bitmélység mellett. Így hozzávetőlegesen megállapítottuk azt az értéket, amelyet érdemesnek tűnt még tovább vizsgálni nagyobb dokumentummennyiségen.
- Az előzőleg megállapított bitmélységű hash értéket nagy mennyiségű dokumentumon, sok töredéken vizsgáltuk, hogy statisztikai eredményünk legyen a várható hibák nagyságrendjéről.

4.3.1 Különböző metódusok és bitmélységek

A vizsgált darabolási metódusok és azok paraméterei a következők voltak:

- mondatonkénti darabolás, csak a pontot mondathatárnak véve
- átlapolódó szavas darabolás, n=6
- hash kódon alapuló darabolás, n=9

Bár nem volt várható lényeges különbség az egyes metódusok szerint előállított töredékek viselkedésében, azonban kiterjedt és mindent figyelemmel kísérő vizsgálatot éreztünk szükségesnek, hiszen az emberi szöveg feldolgozásával foglalkozó kutatót gyakran érhetik meglepetések különböző rejtett összefüggések kapcsán.

Minden metódus esetében 8 különböző bitmélységet teszteltünk:
16, 32, 48, 64, 80, 96, 112, 128 biten vizsgáltuk a keletkező hamis pozitív értékeket.
Bár nyilvánvalóan a ma rendelkezésre álló számítógépek nem támogatják az összes fenti bithossz tárolását, azonban szerettünk volna átfogó képet kapni arról, hogy hogyan viselkedik egy ilyen diszkrét függvény.

Minden egyes esetben 20 000 töredéket vizsgáltunk. Ez mondatonkénti darabolás és hash kódon alapuló darabolás esetén 16 RFC dokumentumot jelent, az átlapolódó szavas darabolás esetén 9-et. A vizsgálat során nem szűrtük ki az azonos töredékeket, így természetesen, ha egy töredék többször előfordul a halmazban, és van „hamis pozitív párja”, akkor nyilvánvalóan többször számoljuk, mint hamis pozitív értéket. Azonban ez a szemlélet teljes mértékben igazodik a későbbi alkalmazási környezethez, hiszen másolatkeresés esetén a többszörös hamis pozitív értékek többszörös hibát fognak jelenteni.

Előzetes intuíciónk alapján arra számítottunk, hogy a céljainknak leginkább megfelelő bitmélység várhatóan 64 bit körül lesz.

A vizsgálat eredménye

Sem az előzetes vizsgálódásaink során, sem a későbbi részletes tesztek alkalmával nem találtunk összefüggést a használt darabolási eljárás és a hamis pozitív értékek között. Ebből kifolyólag a továbbiakban elhagyjuk a darabolási metódus vizsgálatát, és csak a bitmélységeket vizsgáljuk.

Az M6. melléklet tartalmazza a 20 000 töredéken végzett tesztek eredményeit. Jól látható az eredmény darabolási metódustól való függetlensége, hiszen az egyes metódusok nagyságrendileg azonos hamis pozitív értékeket mutattak. Megállapítható ugyanakkor, hogy a számunkra megfelelő bitmélységet 32 bit környékén, valójában inkább az alatt kell keresnünk. Látható, hogy a 16 bites kód jelentékeny mennyiségű hamis pozitívot eredményez, ez tehát nem felel meg céljainknak. Nyilvánvaló azonban, hogy egy kis mennyiségen való tesztelés csupán iránymutatást ad a további vizsgálódásokhoz, ebből statisztikai következtetést levonni még nem lenne szerencsés.

A váratlan, ugrásszerű változás nyilvánvalóan a többszörözött hamis pozitív értékek miatt következik be, hiszen a töredékek között nem ritka a többszörösen előforduló töredék. Így, ha egyfajta töredék hamis pozitív párt alkot egy másikkal, akkor az ugyanilyen töredékek mind hamis pozitív párt fognak alkotni, ezenfelül a lekérdezés többször fogja őket párosítani (amint a tényleges dokumentumkeresésnél is).

4.3.2 Vizsgálat nagy mennyiségű dokumentumon

További vizsgálódásaink kifejezetten a 32 bites, valamint a 24 bites kód viselkedését vizsgálták nagyobb dokumentumhalmazon. A metódusok vizsgálatát is megtartva végeztük el a hamis pozitív keresést 500 000 db töredéket tartalmazó mintán. Bár a tényleges, nagyszámú dokumentumot tartalmazó szövegtárak méretéhez képest ez a mennyiség is kicsinek tekinthető, azonban mindenképpen alkalmas arra, hogy statisztikai következtetést és a százalékos arányok alakulására vonatkozó következtetéseket vonjunk le belőlük.

A vizsgált metódusok:

- hash kódon alapuló, n=6
- hash kódon alapuló, n=9
- átlapolódó szavas, n=6
- mondatonkénti

A vizsgált töredékek száma: 500 000

<u>A vizsgálat eredménye</u>

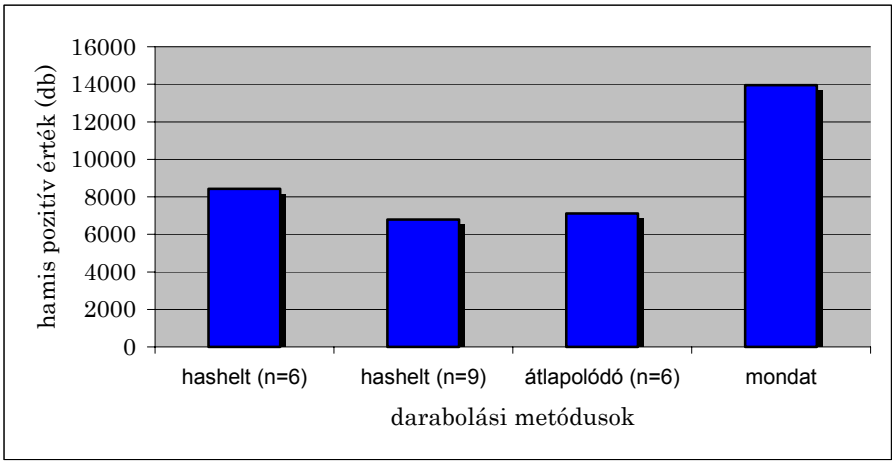
A kapott adatokból egyértelműen kiderül, hogy a 32 bites hash kódon alapuló töredékkódolás igen biztató eredményeket nyújt. Ekkora hosszúságú kódolással a várható hiba ezrelék nagyságrendű lesz. A darabolásból adódó hibalehetőséghez képest ez elhanyagolható mennyiségű hamis hasonlóságot fog eredményezni a másolatkeresésnél (1. táblázat, 5. és 6. diagram)

Valamint – mint azt már fentebb említettük – az optimális bitmélység független a daraboló eljárástól, azaz az adatbázis mérete kizárólag a különböző daraboló eljárásoknál keletkező töredékek számától függ.

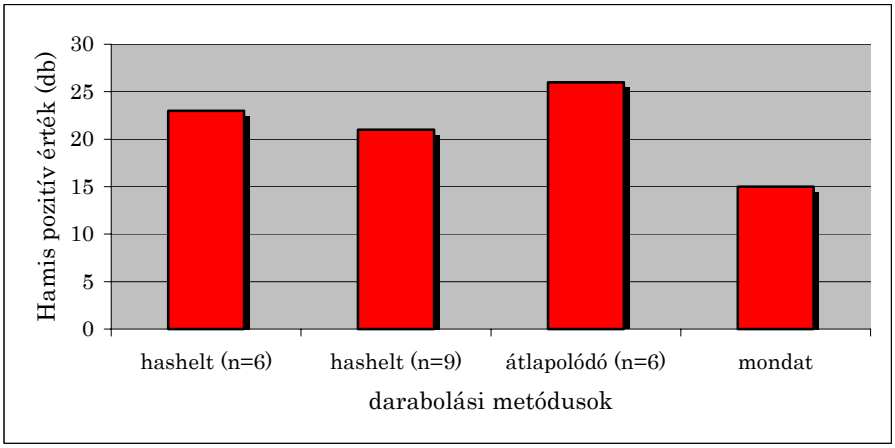
Mindemellett elmondható, hogy a jelenlegi személyi számítógépek tekintélyes részén a 32 bites számábrázolás a támogatott, így processzor kihasználtság szerint is optimális lehet ez a megoldás, szemben a 24 bittel, amely kettő nem egész számú hatványa.

metódus	bithossz	hamis pozitív értékek (db)	Hamis pozitív értékek a vizsgált töredékek százalékában (%)
hash kódon alapuló (n=6)	24	8434	1,6868
hash kódon alapuló (n=9)	24	6790	1,3580
átlapolódó (n=6)	24	7115	1,4230
mondatonkénti	24	13954	2,7908
hash kódon alapuló (n=6)	32	23	0,0046
hash kódon alapuló (n=9)	32	21	0,0042
átlapolódó szavas (n=6)	32	26	0,0052
mondatonkénti	32	15	0,0030

1. táblázat: 500 000-es töredékmintán végzett teszt eredménye



5. diagram: 500 000 töredékmintán, 24 bites kódolás hamis pozitív kódjai



6. diagram: 500 000 töredékmintán, 32 bites kódolás hamis pozitív kódjai

5 Működő rendszerek

Ebben a fejezetben két konkrét dokumentum összehasonlító rendszert vizsgálunk meg, és részletesen elemezzük azt, hogy mikor melyik daraboló eljárást alkalmaztuk és miért, hiszen mint korábban láttuk ez az egyik legfontosabb kérdés egy ilyen rendszer megépítésében. Ugyancsak fontos része egy ilyen programnak az adatbázis struktúra, amelyet használ, az ezek kapcsán felmerülő kérdésekkel, és az adatbázis struktúrájának leírásával Hodász Gábor foglalkozik [Hod02].

Az első ismertetésre kerülő program egy házi feladat beadó rendszer lesz, amelyen keresztül a diákok leadhatják egy adott tárgyhoz tartozó házi feladataikat, a tanár meg azon felül, hogy megtekintheti azokat, kap egy listát is, amely a házi feladatok közötti átfedéseket tartalmazza. Ezt a programot Petrányi Jánossal közösen készítettük, de az összehasonlító résszel én egészítettem ki.

A második rendszer egy sokkal komolyabb és univerzálisabb program, amelyet a melbourne-i Monash Egyetemnek fejlesztettünk ki Hodász Gáborral közösen, és nagy mennyiségű dokumentum összehasonlítására alkalmas [Monash].

Mindkét rendszer megtartja a leadott dokumentumokat, hogy azokat később a felhasználók megtekinthessék. Ha egy internetes hasonlóságkeresőt építettünk volna, akkor elég lenne csak az Internet címet (URL) megtartanunk, hiszen az tökéletesen azonosítja a dokumentumot, amely később újra beszerezhető, és így rengeteg helyet megtakarítunk.

5.1 Házi feladat beadó rendszer

5.1.1 Működése

A program számon tartja minden tárgy adatait: házi feladatok számát, tanárokat és diákokat, tantárgy kódját és honlapjának címét. Ezeket az adatokat a tanár kell, hogy feltöltse XML formátumban. Az XML formátumot rugalmassága miatt választottuk. A program lehetőséget biztosít egy házi feladat újbóli beadására is, természetesen csak a tanár beleegyezésével. Ezen felül a tantárgyak osztályzására, illetve a diákok részéről az osztályzatok megtekintésére is lehetőséget nyújt.

Amivel többet tud ez a rendszer, mint egy sima házi feladat beadó rendszer az az, hogy amikor a tanár megtekinti a leadott házi feladatokat, rögtön kap egy listát, amely az egymásra hasonlító házi feladat párokat tartalmazza, a hasonlóság mértékével együtt. A mértéket nem csak százalékban, hanem azonos töredékszámban is kiírja. Ennek előnyeiről már esett szó a 4.2.3. fejezetben az átlapolódó hash kódon alapuló darabolásnál.

Mint minden dokumentum összehasonlító rendszer ez is követi az 1. ábrán felvázolt folyamatot. A dokumentumok beszerzése a diákok által letöltött házi feladatot jelenti, konvertálásra nincsen szükség, mert a rendszer csak sima text fájlok feldolgozására alkalmas. A darabolás és ujjlenyomat készítés egy lépcsőben készül el. Az adatbázis feltöltése csak az összes ujjlenyomat megléte után kezdődhet el. A hasonlóság lekérdezése akkor történik meg, amikor a tanár a leadott házi feladatokat kilistáztatja. A program összehasonlít minden házi feladatot minden házi feladattal, azaz nem csak a diákokét egymással, hanem egy diák házi feladatát a saját korábban beadottjával is. Természetesen a különböző tárgyak házi feladatait nem hasonlítja össze egymással.

5.1.2 Felhasznált programok

A teljes program PHP nyelven íródott. Azért esett erre a nyelvre a választás, mert a program internetes környezetben kell, hogy fusson, egy szerver gépen (biztonsági okokból), azaz csak szerver oldali programok jöhetnek szóba. Ezeken felül a PHP olyan beépített függvényekkel támogatja a munkánkat, mint MD5 algoritmus, internetes űrlapok kezelése, fájlfeltöltés támogatása, session kezelés, XML feldolgozás, és nem utolsó sorban adatbázis könnyű és rugalmas elérése SQL parancsokkal. Természetesen ennek ára van. Egy Java vagy C++ nyelven megírt program többszörös gyorsulást eredményezne, ugyanakkor a fejlesztési idő is többszörösére nőne.

A tantárgyak leírását XML formátumban kell leadnia a tanárnak, ennek feldolgozására a phpXML-t használtuk fel, amelyen keresztül könnyedén elérhetőek Xpath kifejezésekkel az XML-ben tárolt adatok.

Adatbáziskezelőnek a PHP által ugyancsak támogatott MySQL-t választottuk, elsősorban a PHP támogatás miatt. Ezen felül még azért is esett rá a választásunk, mert könnyedén elfut egy otthoni PC-n is, mivel nem nagy az erőforrásigénye, ugyanakkor gyors, megbízható, és minden olyan funkciót támogat amelyre programunknak szüksége van.

5.1.3 Daraboló eljárás kiválasztása

Mint azt a 4.2. fejezetben beláttuk, nem mindegy, hogy mely daraboló eljárást választjuk. Nézzük meg, hogy melyik miért jó vagy nem jó. A mondatonkénti darabolást gyorsan elvethetjük, mert ha egy diák másol valakiről valószínűleg azért egy picit átírja a szöveget, és ha minden mondaton változtat egy picit akkor már nem találjuk meg az egyezést. Ráadásul gyakran hibás egyezést mutatna, hasonló témában megírt házi feladatok esetében. A hash kódon alapuló darabolás nem rossz, de mint láttuk, ahhoz, hogy biztonságosan működjön, szükségünk van példadokumentumokra, ez meg egy univerzális házi feladat beadó rendszer esetén nem áll rendelkezésünkre. Sőt, az is elképzelhető, hogy a német és ez angol tanárok is ezt a rendszert fogják használni, és olyan paramétert találni, amely három nyelvre is jó, elég nehéz, inkább lehetetlen. Marad az átlapolódó szavas darabolás, ennek a legrosszabb tulajdonsága, mint láttuk, a nagy mennyiségű töredék és az ezzel járó többletmunka. Ez itt most minket nem fog annyira zavarni, hiszen megtehetjük, hogy tantárgyanként külön táblába tesszük a töredékeket, azaz a rendszer mindig csak egy tárgyon belüli házi feladatot lát egyszerre, amely már nem jelent akkora mennyiségű adatot. Ráadásul az átlapolódó szavas darabolás mutatja ki legmegbízhatóbban a hasonlóságot.

Most már csak a paramétert kell megállapítanunk. Túl kis paramétert nem érdemes választanunk, hiszen egy házi feladat témája gyakran azonos az összes diáknak így rengeteg azonos kifejezést és szóösszetételt tartalmazhat. Mivel kisebb egyezéseket is ki szeretnénk mutatni, hogy a picit átírt dolgozatokat is kiszűrjük, ezért egy közepesnek mondható ötös értéket kell választanunk. Könnyen lehet, hogy egyes házi feladatnál ez több hibás egyezést is ki fog mutatni, de ez nem gond, hiszen a tanár látja az egyezés mértékét is, és így el tudja dönteni, hogy hány százalékos egyezés felett kell ténylegesen másolásra gyanakodnia, majd egymás mellé téve összehasonlítania a két művet.

Miután megválasztottuk a daraboló eljárást és a paramétert is, már csak meg kell írni a programot. A teljes program megtalálható a CD mellékleten, most csak a daraboló eljárást vizsgáljuk meg részletesebben. A kód PHP nyelven íródott.

```
$fp = fopen($filename, "rb");  //b=binary     //fájl megnyitása olvasásra
$contents = fread($fp, filesize($filename));            //fájl beolvasása
$contents = strtolower ($contents).' ';     //kisbetűkre való konvertálás

$max = strlen($contents);                               //hossz lekérdezése
$word = '';
$chunk = '';
$NrofChunks = 0;
$NrofWords = 0;
$pos1 = 0;
for ($i = 0; $i < $max ; $i++)        //betűnként végigmegyünk a szövegen
    {
    $char = $contents{$i};
    if ( (($char<a) or ($char>z)) and (($char<'0') or ($char>'9')) )
        {  //if not a..z or A..Z or 0..9          //ha speciális karakter
        if ($word != '')  //if first spec. char  //ha első ilyen karakter
            {                                              //új szót kezdünk
            $chunk = $chunk.$word.' ';  //a régit hozzáadjuk a töredékhez
            $NrofWords++;
            if ($NrofWords >= 5)                  //ha már van öt szavunk
                {                            //ez a darabolás paramétere!!!
                $NrofChunks++;
                $chunks[] = substr(md5($chunk), 0, 8);      //ujjlenyomat
                $pos1 = strpos($chunk, " ");           //első szó törlése
                $chunk = substr($chunk, $pos1 + 1);
                }
            $word = '';                                    //szó kiürítése
            }
        }
    else  //if a..z or A..Z or 0..9           //ha NEM speciális karakter
        {
        $word = $word.$char;          //simán hozzáadjuk a betűt a szóhoz
        }
```

A program elkezdi betűnként beolvasni a szöveget és egy változóba menti el (hozzáfűzi) a beolvasott karaktereket, majd amikor egy olyan karakterhez ér, amely se nem betű, se nem szám, akkor az a szó végét jelzi és továbblép. Megnézi, hogy van-e már annyi szava amennyi a paraméter, ha nincs, akkor ezt a szót hozzáfűzi a töredékhez, és egy újabb szó olvasásába kezd, mindaddig ismételve ezt, míg el nem éri a paramétert a szavak száma. Ez a szöveg elején lényeges. Amikor már összegyűlt paraméternyi szó, akkor vége van a

töredéknek is, kiszámolja a töredék MD5 értékét (ujjlenyomat) és eltárolja azt egy tömbben. Ezek után a töredék első szavát letörli, és a töredékdarabot eltárolja, majd újabb szó beolvasásába kezd. Amikor beolvasott egy újabb szót, akkor azt hozzáfűzi a töredékdarabhoz és újra töredékhatárhoz ért, mert már minden szó töredékhatár is lesz egyben. Ezt az egészet addig folytatja, míg el nem ér a dokumentum végéig.

Mint az a programból kitűnik, igen egyszerűen és gyorsan megvalósítható egy ilyen daraboló eljárás. A töredékek számát azért kell nyilvántartani, mert arra később a hasonlóságok százalékos kijelzésénél szükségünk lesz.

Ezek után nincs más dolgunk mint eltárolni a keletkezett töredékek ujjlenyomatait és a házi feladat adatait (fájlnév, hossz, töredékek száma). Amikor a tanár lekérdezi a házi feladatokat, egy listát fog kapni, amely tartalmazza a hasonlóságokat.

5.2 Hasonlóság kereső rendszer

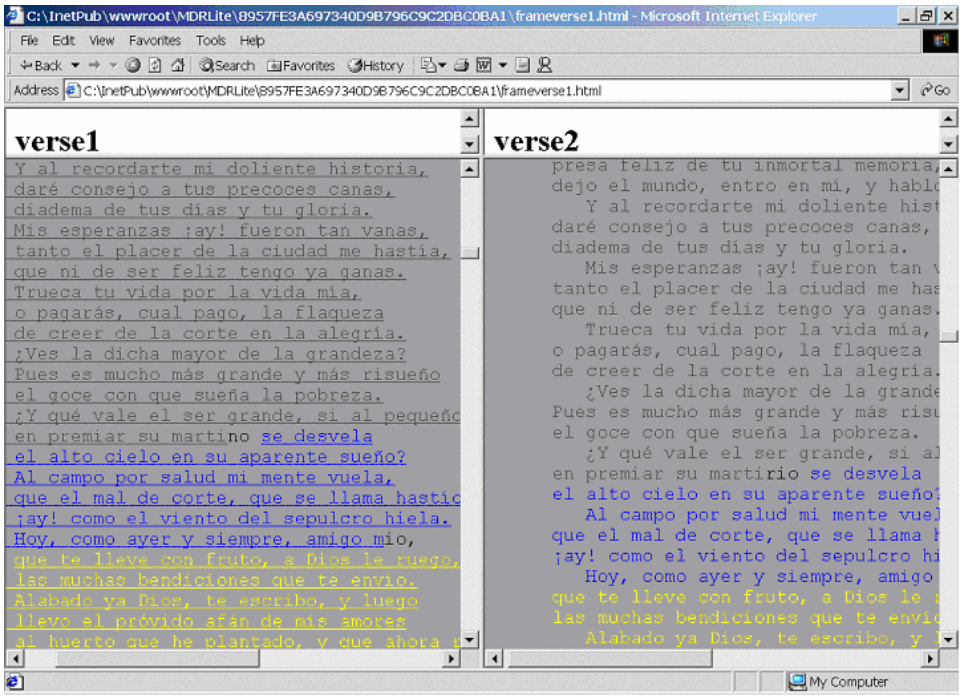
5.2.1 Működése

Az ausztráliai melbourne-i Monash Egyetemen lévő hasonlóság kereső program [Monash] egy sokak által egyszerre használt rendszer, amelybe mindenki feltölthet dokumentumokat és ezeket össze is vetheti az adatbázisban lévőkkel. A felépítése már sokkal bonyolultabb, mint a házi feladat beadó rendszeré. Több dokumentumot is fel lehet tölteni egyszerre ZIP illetve RAR formátumban, valamint nem csak sima szövegfájlt fogad el a rendszer, hanem PDF, HTML, DOC valamint RTF fájlokat is. A RAR illetve ZIP fájlok akármilyen mélyen egymásba lehetnek ágyazva. A konvertálás sima szöveggé külső programok segítségével történik [Doc01], [HTML01], [PDF01].

Konvertálás után jön a darabolás, az ujjlenyomat készítése, és az adatbázis feltöltése. Az adatbázisban minden fájlhoz eltárolásra kerül: a fájl helye a szerveren, a felhasználó gépén a fájl elérhetősége, egy a felhasználó által megadott URL (pl. Internet cím), relatív címe a sűrített fájlon belül, hossza bájtokban, töredékek száma, felhasználó neve, felhasználó gépének IP címe, felhasználó csoportja, dokumentum típusa (gyanús dokumentum, eredeti dokumentum), kivonat, kulcsszavak, név, cím, szerző, nyelv, hossza oldalban, kiadó, kiadás dátuma, feltöltés dátuma. Ez kicsit túlzásnak tűnhet, de ha belegondolunk, hogy egy egész egyetem használhatja a rendszert, akkor megértjük, hogy később, egy esetleges hasonlóság kimutatása esetén, azonosítani kell, hogy tudja mindenki pontosan a fájlokat.

A hasonlóságok lekérdezése a felhasználó szempontjából ebben a rendszerben egyszerre történik a feltöltéssel, hiszen a feltöltött fájlokat hasonlítja össze az adatbázissal. A gyakorlatban ez két lépcsőben folyik, előbb regisztráljuk a feltöltött fájlt, majd lekérdezzük a hasonlóságokat. A hasonlóság megmutatására egy kétpaneles vizualizáló programot használ a rendszer, amely Monostori Krisztián munkája [Monos2]. Ez a komponens a bal

oldalon a gyanús dokumentumot mutatja, jobb oldalon meg a hozzá hasonló eredetieket. Az azonos részeket azonos színnel jelöli, így könnyítve meg azok megtalálását. Egy azonos részre kattintva a bal oldalon, betölti a jobb oldali panelba a megfelelő dokumentumot, és az egyező részhez ugrik. A program kimenete a 2. ábrán látható.



2. ábra: Vizualizáló komponens kimenete

5.2.2 Felhasznált programok

A program PHP nyelven íródott, Apache WEBszerver alatt fut, és Oracle adatbázist használ. Ezek a komponensek nagyrészt adottak voltak számunkra, mert az egyetem ezeket használja szerverein.

Előző fejezetben leírtam a PHP előnyeit, ezt most nem teszem meg újra, csak kiegészíteném annyival, hogy az Oracle adatbáziskezelőt is támogatja egy „extension” segítségével.

Ami viszont a legnagyobb hátránya a PHP-nek az a sebesség, és ezen mindenképpen érdemes elgondolkodnunk. A házi feladat leadónál ez nem volt

gond hiszen ott nem nagy mennyiségű dokumentumról volt szó, ennek a programnak viszont egy több megabájtos sűrített fájl is átadható, tele hosszú dokumentumokkal. Ezért komoly teszteket futtattunk a szerveren. Az eredmény kielégítő volt, igaz néha öt-tíz percet kellett az eredményre várni, de ez több száz feltöltendő dokumentum esetén elviselhető.

5.2.3 Daraboló eljárás kiválasztása

A rendszernek még rengeteg egyéb funkciója van, de azok most számunkra nem érdekesek, a fenti információk viszont segítenek a daraboló eljárás jó kiválasztásában.

Kezdjük a házi feladatnál használt átlapolódó szavas darabolással. Biztosak lehetünk benne, hogy jól kimutatja az egyezéseket, ugyanakkor az adatbázis mérete itt már nem mindegy. Mivel egy egész egyetem használja sok hosszú dokumentum (publikációk, diplomák, doktori disszertációk) tárolására, nem használhatjuk az átlapolódó szavas darabolást. Nagyon lassúvá tenné a lekérdezést és túlterhelné a szervert. A mondatonkénti darabolást megint elvethetjük a fentebb említett hátrányai miatt. Tehát marad az hash kódon alapuló darabolás vagy az átlapolódó hash kódon alapuló darabolás. Mivel tudjuk, hogy az ausztrálok majdnem csak angol nyelvű dokumentumokat használnak, hiszen ez az anyanyelvük, a szakmai cikkek is ma már túlnyomó részben angolul jelennek meg, és nem utolsó sorban az Internet nyelve is angol. Mindebből következik, hogy tudunk példadokumentumokat szerezni, olyanokat amelyek később az adatbázisban is lesznek és meg tudjuk nézni, hogy melyik paraméter a legalkalmasabb. Ezért nincs szükség az átlapolódó hash kódon alapuló darabolásra, elegendő a sima hash kódon alapuló darabolás.

Mivel a kód, az előzőekben ismertetetthez képest, sok újdonságot nem tartalmaz, ezért részletes ismertetésétől eltekintek. A megfelelő kódrészlet az M8. mellékletben, a teljes program, a rendszer részletes leírása, és használati útmutatója pedig a CD mellékletben található meg.

6 Összefoglalás

Diplomadolgozatom célja a hasonlóságkereső rendszerekben használt daraboló eljárások elemzése, és az ezek felhasználásával elkészített rendszereink bemutatása volt.

Mint láttuk, mindegyik daraboló eljáráshoz megadhatóak olyan alkalmazási területek, amelyekben kimondottan jól ki lehet használni valamely előnyös tulajdonságukat. Ez azt mutatja, hogy a tesztelt összes daraboló eljárásnak van létjogosultsága. Ezért nem adhatunk egyértelmű választ arra a kérdésre, hogy melyik daraboló eljárás a legjobb. Ehelyett megpróbálok útmutatást adni arra, hogy ha valaki ilyen algoritmust szeretne használni, akkor hogyan választhatja ki a számára legjobban megfelelőt. Ennek az elősegítésére – a tesztek során szerzett tapasztalatok alapján – egy táblázatban (2. táblázat) összefoglaltam a darabolási eljárásokat, és a legfontosabb tulajdonságaik szerint értékeltem őket.

Azért, hogy jól áttekinthető legyen, az összes tulajdonságot egy ötelemű skálán ábrázoltam. Azok a tulajdonságok, amelyekben kiemelten jók a daraboló eljárások, + + jelet kaptak. Ahol nagyon gyengék - - jelet. Egy +, illetve egy - jelet kaptak, ha jók vagy gyengék, és 0-t ha közömbösek (vagy közepesek) arra a tulajdonságra nézve. Természetesen ez egy tulajdonságokon belüli, relatív skála, és nem lehet kijelenteni, hogy az *„A”* eljárás kapta a legtöbb + jelet ezért az a legjobb választás.

Ez a téma még rengeteg érdekes kérdést és lehetőséget rejt magában, melyek megválaszolása sajnos túlmutat ezen dolgozat keretein, de azért mindenképpen érdemes megemlíteni ezeket.

Ajánlatos lehet egy olyan összehasonlító rendszer megvalósítása, amelynél a konvertálás és a darabolás a kliens gépen történik. Ez sokkal kevésbé terhelné le a szervert, illetve a hálózati forgalmat is nagyban lecsökkentené, hiszen csak az ujjlenyomatokat és a fájl adatait kellene átadni.

	átlapolódó szavas	mondatonkénti	hash kódon alapuló	átlapolódó hash kódon alapuló
darabolás sebessége	-	+ +	+ +	-
tömöríthetőség (MD5)	0	0	0	0
hamis+	0	0	0	0
keletkező adatbázis mérete	- -	+	+ +	-
testreszabhatóság	+	- -	+ +	+ +
találatok megbízható megtalálása	+ +	0	-	+ +
egy javasolt felhasználási terület	nyelvtani elemző	idézetek keresése	internetes hasonlóság-kereső	megbízható keresés Interneten

2. táblázat: Daraboló eljárások összehasonlítása

Ugyancsak érdemes elgondolkodni egy elosztott adatbázison, ahol több szerver között oszlik meg a munka, mindegyik más-más fájlokat tárol. Mindegyik szerver megkapja a lekérdezést (ugyanazon fájl ujjlenyomatait) és egy központi gép összegyűjti a válaszokat. Természetesen ez a megoldás csak akkor működik, ha nincs szükségünk adatbázison belüli összehasonlításokra. A másodszor ismertetett hasonlóság kereső rendszer például alkalmas lehet erre.

A kapott eredmények alapján ez a téma mindenképpen további elemzésekre érdemes. Amennyiben lehetőségem nyílik rá, az előbb felvázolt irányokba szeretném tovább folytatni a kutatást, egy PhD munka keretében.

Köszönetnyilvánítás

Köszönöm szépen szüleimnek Pataki Pálnak és Eszternek a támogatást és bátorítást az egyetemi évek alatt. Köszönöm, hogy segítettek eljutni olyan helyekre ahol még ők se jártak.

Köszönöm konzulensemnek Monostori Krisztiánnak, hogy segített elnyernem az ausztráliai Monash Egyetem ösztöndíját, és értékes tanácsait mind a TDK mind pedig a diplomamunkámnál.

Special thanks to Arkady Zaslavsky for the Monash University scholarship, and for supervising the project.

Irodalomjegyzék

[Bae99] BAEZA-YATES, Ricardo – RIBEIRO-NETO, Berthier, *Modern Information Retrieval*, Addison Wesley, 1999.

[BD01] BÉKÉS, Gábor – DALOS, Péter ford. Újszövetségi szentírás. *A külföldi katolikus magyar akció kiadása*, Róma, 1951. URL http://www.cadvision.com/mayl/bd-ind.html, 2001.

[Bri99] BRIN, Sergey – DAVIS, James – GARCIA-MOLINA, Hector, *Copy Detection mechanism for Digital Documents*, Department of Computer Science, Stanford, CA 94305-2140, 1999.

[Cop01] *CopyCatch System,* http://www.copycatch.freeserve.co.uk, 2001.

[Doc01] *Doc2txt converter*, http://plaza27.mbn.or.jp/~satomii/soft/cui/doc2txt.html, 2001.

[EVE00] *EVE Plagiarism Detection System,* http://www.canexus.com, 2000

[Gla99] *Glatt Plagiarism Screening Program,* http://www.plagiarism.com/screen.id.htm, 1999.

[Hei99] HEINTZE, Hevin, *Scalable Document fingerprinting (Extended Abstract)*, Bell Laboratories, Murray Hill, NJ 07974, 1999.

[Hod02] HODÁSZ, Gábor, *Szöveges dokumentumok darabolása és tömörítése hash-kódolással, Adattárolási és adatszervezési kérdések*, Budapesti Műszaki és Gazdaságtudományi Egyetem, 2002.

[HTML01] *HTML2txt Converter*, http://www.infomedia.it/artic/Baccan/opensource/, 2001.

[Int01] *InteriGuard System,* http://www.integriguard.com, 2001.

[Kár01] KÁROLI, Gáspár ford., *Szent Biblia*, 1591 http://www.cab.u-

szeged.hu/WWW/books/Biblia/, 2001.

[Kock99] KOCK, N, *A case of academic plagiarism,* Communications of the ACM, July 1999, Vol. 42, No.7, 1999.

[Monash] HODÁSZ, Gábor – PATAKI, Máté, *Match Detect Reveal*, Monash University Melbourne, 2002, http://monster.csse.monash.edu.au:8080/fileup/mdr.php

[Monos1] MONOSTORI, Krisztián – FINKEL, Raphael – ZASLAVSKY, Arkady – HODÁSZ, Gábor – PATAKI, Máté, *Comparison of Overlap Detection Techniques,* Computer Science ICCS 2002 International Conference, Amsterdam, April 21-24, 2002, Proceedings, Part I., Springer, 2002.

[Monos2] Monostori, Krisztián, *Efficient Computational Approach to Identifying Overlapping Documents in Large Digital Collections*, School of Computer Science and Software Engineering, Monash University, 2002.

[Pap01] *PaperBin system*, http://www.paperbin.com, 2001.

[PDF01] PDF2txt Converter, http://alkaline.vestris.com/files/asearch-distrib/WinNT/, 2001.

[Pla99] *Plagiarism.org, the Internet plagiarism detection service for authors & education*, http://www.plagiarism.org , 1999.

[Ref93] Magyar Bobliatársulat ford., *Szent Biblia*, Budapest, 1993

[RFC] *Request For Comments texts*: http://www.rfc-editor.org

[RFC1321] RIVEST, R., *RFC1321 - The MD5 Message-Digest Algorithm*, MIT Laboratory for Computer Science and RSA Data Security, Inc., April 1992. *http://www.rfc-editor.org*

[Sal92] SALTON, G., *The state of retrieval system evaluation*, 1992.

[Shi95] SHIVAKUMAR, Narayanan – GARCIA-MOLINA, Hector, *SCAM: A Copy Detection Mechanism for Digital Documents*, Department of Computer Science, Stanford University, CA 94305-2140,

Proceedings of 2nd International Conference in Theory and Practice of Digital Libraries (DL'95), Austin, Texas, 1995.

[Shi96] SHIVAKUMAR, Narayanan – GARCIA-MOLINA, Hector, *Building a Scalable and Accurate Copy Detection Mechanism*, Department of Computer Science, Stanford, CA 94305, 1996.

Mellékletek

M1. melléklet: Szótár

darabolás	Az az eljárás, amely során a dokumentumot töredékekre osztjuk fel.
finomhangolás	A rendszer paramétereinek „kismértékű” változtatása, melynek célja, hogy az adott felhasználási környezetben a lehető legjobb eredményt adja. Esetünkben a darabolási eljárások paramétereinek módosításával lehet elérni, hogy a rendszer különböző alkalmazási területeken az optimumot nyújtsa.
hamis pozitív eset	Általánosságban olyan eset, amely megfelelőnek tűnik egy bizonyos kritériumnak, azonban valamilyen hiba folytán mégsem az. Esetünkben azt a hash kódolt töredéket hívjuk hamis pozitív esetnek, amely a kódolás folyamán egyező kódot kapott egy vele nem egyező töredékkel. Így a másolat kereső lekérdezés egyezést fog találni ott, ahol ténylegesen nincs egyezés a két dokumentumban.
hash kódolás	Olyan veszteséges kódolás, mely karakterláncot alakít át fix hosszúságú kóddá. Felhasználási területe egyrészről a szöveges adatbázisok, másrészről a kriptográfia.
MD5	Message Digest 5, kriptográfiai algoritmus, melynek kódja publikus (rfc1321.txt). Tetszőleges hosszúságú szöveget 128 bit hosszú kódra képez le, ezáltal veszteséges kódolását adja a bemenetnek.
ODBC	Open Database Connectivity, API specifikáció, amely szabványos eljáráskészletet definiál az alkalmazások számára adatforrások adatainak eléréséhez.
RFC	Request For Comments, szabad terjesztésű ajánlások gyűjteménye, amelyek de facto szabványnak tekinthetők. Leírásuk egyszerű szöveges fájlokban rendelkezésre állnak többek között például a http://www.rfc-editor.org címen. Vizsgálódásaink kezdetekor (2001. április) 2590 fájlt tartalmazott a gyűjtemény.
stopword	Olyan szavak, amelyek gyakran előfordulnak, a szöveg jelentéstartalmával nem állnak összefüggésben, ezért eltávolításuk a szövegből nem okoz információ-csökkenést. Például: névmások, létigék, névelők stb.
töredék	Egy dokumentum kisebb darabjai, melyek nem feltétlenül függetlenek egymástól (átlapolódó eset).

M2. melléklet: Példamondat a daraboló eljárások illusztrálására

Azonos színnel jelöltük az azonos, töredékekbe kerülő szavakat.

Eredeti szöveg:

Ezen project célja, hogy a Monash University-vel együttműködve egy olyan rendszert hozzunk létre, amely hatékony a dokumentum-másolatok felderítésében.

Szavas darabolás (n=5):

ezen project célja hogy a monash university vel együttműködve egy olyan rendszert hozzunk létre amely hatékony a dokumentum másolatok felderítésében

Átlapolódó szavas darabolás (n=5):

ezen project célja hogy a
project célja hogy a monash
célja hogy a monash university
hogy a monash university vel
a monash university vel együttműködve
monash university vel együttműködve egy
university vel együttműködve egy olyan
stb.

Mondat alapú darabolás:

ezen project célja hogy a monash university vel együttműködve egy olyan rendszert hozzunk létre amely hatékony a dokumentum másolatok felderítésében

Hash érték szerinti darabolás:

ezen project célja hogy a monash university vel együttműködve egy olyan rendszert hozzunk létre amely hatékony a dokumentum másolatok felderítésében.
Az aláhúzott szavak esetünkben töredékhatárok.

M3. melléklet: Példa a daraboló komponens kimenetére

Starting at: 10.23. 17:17:23
InDirectory: E:\TXT\Biblia\TXT
OutDirectory: E:\TXT\ Biblia \Out
Hash value: 8
Buffer length: 1024

001	2moz20_ref.txt	000083	000015	5.533333
002	2moz20_karoli.txt	000124	000016	7.750000
003	1moz_kat.txt	000010	000003	3.333333
004	1kor_kat.txt	000071	000009	7.888889
005	1kor13_kat.txt	000060	000009	6.666667
006	2moz20_kat.txt	000090	000008	11.250000
007	2kor_kat.txt	000148	000023	6.434783
008	1kor13_ref.txt	000077	000010	7.700000
009	1kor13_karoli.txt	000087	000012	7.250000

Number of words: 20365
Number of chunks: 2364
Average: 8.614636
Finished at: 10.23. 17:17:23

M4. melléklet: A Bibliából vett dokumentumok elnevezése

Fájlnév	Méret (bájt)	Tartalom
1kor_kat.txt	46422	Pál első levele a Korinthusiakhoz, teljes, katolikus fordítás
1kor13_karoli.txt	1531	Szeretet himnusza: Pál első levele a Korinthusiakhoz 13. vers, Károli Gáspár fordítása
1kor13_kat.txt	1381	Szeretet himnusza: Pál első levele a Korinthusiakhoz 13. vers, katolikus fordítás
1kor13_ref.txt	1479	Szeretet himnusza: Pál első levele a Korinthusiakhoz 13. vers, református fordítás
1moz_kat.txt	43054	Mózes első könyve, elejéről, katolikus fordítás
2kor_kat.txt	30625	Pál második levele a Korinthusiakhoz, teljes, katolikus fordítás
2moz20_karoli.txt	2804	10 parancsolatot: Mózes második könyve 20. vers, Károli Gáspár fordítása
2moz20_kat.txt	2631	10 parancsolatot: Mózes második könyve 20. vers, katolikus fordítás
2moz20_ref.txt	2620	10 parancsolatot: Mózes második könyve 20. vers, református fordítás

Várható egyezések:

Az *1kor13_karoli.txt*, az *1kor13_kat.txt* és az *1kor13_ref.txt* hasonlítanak egymásra, akárcsak a *2moz20_karoli.txt*, a *2moz20_kat.txt* és a *2moz20_ref.txt*. Az *1kor_kat.txt* teljes egészében tartalmazza az *1kor13_kat.txt*-t.

Az *1kor_kat.txt*, a *2kor_kat.txt* és az *1moz_kat.txt* nem hasonlítanak egymásra, pár egyező szóösszetétel lehet bennük, de csak kis számú. Ezt mi ki szeretnénk szűrni, mert ekkora hasonlósággal nem akarunk és nem is tudunk foglalkozni, hisz ekkora akármelyik két dokumentum között meglehet, akár véletlenül is.

M5. melléklet: A Szeretet himnusza három fordításban

Károli Gáspár fordítása	**Református fordítás**	**Katolikus fordítás**
Ha embereknek vagy angyaloknak nyelvén szólok is, szeretet pedig nincsen én bennem, olyanná lettem, mint a zengő ércz vagy pengő czimbalom.	Ha emberek vagy angyalok nyelvén szólok is, szeretet pedig nincs bennem, olyanná lettem, mint a zengő érc vagy pengő cimbalom.	Szólhatok az emberek vagy az angyalok nyelvén, ha szeretet nincs bennem, csak zengő érc vagyok vagy pengő cimbalom.
És ha jövendőt tudok is mondani, és minden titkot és minden tudományt ismerek is; és ha egész hitem van is, úgyannyira, hogy hegyeket mozdíthatok ki helyökről, szeretet pedig nincsen én bennem, semmi vagyok.	És ha prófétálni is tudok, ha minden titkot ismerek is, és minden bölcsességnek birtokában vagyok, és ha teljes hitem van is, úgyhogy hegyeket mozdíthatok el, szeretet pedig nincs bennem: semmi vagyok.	Lehet prófétáló tehetségem, ismerhetem az összes titkokat és mind a tudományokat, hitemmel elmozdíthatom a hegyeket, ha szeretet nincs bennem, mit sem érek.
És ha vagyonomat mind felétetem is, és ha testemet tűzre adom is, szeretet pedig nincsen én bennem, semmi hasznom abból.	És ha szétosztom az egész vagyonomat, és testem tűzhalálra szánom, szeretet pedig nincs bennem: semmi hasznom abból.	Szétoszthatom mindenemet a nélkülözők közt, odaadhatom a testemet is égőáldozatul, ha szeretet nincs bennem, mit sem használ nekem.
A szeretet hosszútűrő, kegyes; a szeretet nem irígykedik, a szeretet nem kérkedik, nem fuvalkodik fel. Nem cselekszik éktelenül, nem keresi a maga hasznát, nem gerjed haragra, nem rójja fel a gonoszt, Nem örül a hamisságnak, de együtt örül az igazsággal; Mindent elfedez, mindent hiszen, mindent remél, mindent eltűr.	A szeretet türelmes, jóságos; a szeretet nem irigykedik, a szeretet nem kérkedik, nem fuvalkodik fel. Nem viselkedik bántóan, nem keresi a maga hasznát, nem gerjed haragra, nem rója fel a rosszat. Nem örül a hamisságnak, de együtt örül az igazsággal. Mindent elfedez, mindent hisz, mindent remél, mindent eltűr.	A szeretet türelmes, a szeretet jóságos, a szeretet nem féltékeny, nem kérkedik, nem is kevély. Nem tapintatlan, nem keresi a maga javát, nem gerjed haragra, a rosszat nem rója fel. Nem örül a gonoszságnak, örömét az igazság győzelmében leli. Mindent eltűr, mindent elhisz, mindent remél, mindent elvisel.
A szeretet soha el nem fogy: de legyenek bár jövendőmondások, eltöröltetnek; vagy akár nyelvek, megszünnek; vagy akár ismeret, eltöröltetik.	A szeretet soha el nem múlik. De legyen bár prófétálás: el fog töröltetni; legyen nyelveken való szólás: meg fog szűnni; legyen ismeret: el fog töröltetni.	S a szeretet nem szűnik meg soha. A prófétálás végetér, a nyelvek elhallgatnak, a tudomány elenyészik.
Mert rész szerint van bennünk az ismeret, rész szerint a prófétálás: De mikor eljő a teljesség, a rész szerint való eltöröltetik.	Mert töredékes az ismeretünk és töredékes a prófétálásunk. Amikor pedig eljön a tökéletes, eltöröltetik a töredékes.	Most megismerésünk csak töredékes, és töredékes a prófétálásunk is. Ha azonban elérkezik a tökéletes, ami töredékes, az véget ér.
Mikor gyermek valék, úgy szóltam, mint gyermek, úgy gondolkodtam, mint gyermek, úgy értettem, mint gyermek: minekutána pedig férfiúvá lettem, elhagytam a gyermekhez illő dolgokat.	Amikor gyermek voltam, úgy szóltam, mint gyermek, úgy éreztem, mint gyermek, úgy gondolkoztam, mint gyermek; amikor pedig férfivá lettem, elhagytam a gyermeki dolgokat.	Gyermekkoromban úgy beszéltem, mint a gyerek, úgy gondolkoztam, mint a gyerek, úgy ítéltem, mint a gyerek. De amikor elértem a férfikort, elhagytam a gyerek szokásait.
Mert most tükör által homályosan látunk, akkor pedig színről-színre; most rész szerint van bennem az ismeret, akkor pedig úgy ismerek majd, a mint én is megismertettem.	Mert most tükör által homályosan látunk, akkor pedig színről színre; most töredékes az ismeretem, akkor pedig úgy fogok ismerni, ahogyan engem is megismert az Isten.	Ma még csak tükörben, homályosan látunk, akkor majd színről színre. Most még csak töredékes a tudásom, akkor majd úgy ismerek mindent, ahogy most engem ismernek.
Most azért megmarad a hit, remény, szeretet, e három; ezek között pedig legnagyobb a szeretet.	Most azért megmarad a hit, a remény, a szeretet, e három; ezek közül pedig a legnagyobb a szeretet.	Addig megmarad a hit, a remény és a szeretet, ez a három, de közülük a legnagyobb a szeretet.

M6. melléklet: Hamis pozitív érték keresése

Darabolási metódus	Bitmélység	Vizsgált töredékek száma	Hamis pozitív értékek (db)
hash kódon alapuló (n=9)	16	20000	2915
hash kódon alapuló (n=9)	32	20000	0
hash kódon alapuló (n=9)	48	20000	0
hash kódon alapuló (n=9)	64	20000	0
hash kódon alapuló (n=9)	80	20000	0
hash kódon alapuló (n=9)	96	20000	0
hash kódon alapuló (n=9)	112	20000	0
hash kódon alapuló (n=9)	128	20000	0
mondatonkénti	16	20000	2786
mondatonkénti	32	20000	0
mondatonkénti	48	20000	0
mondatonkénti	64	20000	0
mondatonkénti	80	20000	0
mondatonkénti	96	20000	0
mondatonkénti	112	20000	0
mondatonkénti	128	20000	0
átlapolódó (n=6)	16	20000	2812
átlapolódó (n=6)	32	20000	1
átlapolódó (n=6)	48	20000	0
átlapolódó (n=6)	64	20000	0
átlapolódó (n=6)	80	20000	0
átlapolódó (n=6)	96	20000	0
átlapolódó (n=6)	112	20000	0
átlapolódó (n=6)	128	20000	0

M7. Melléklet: A bibliai tesztdokumentumok hasonlóságai

1. átlapolódó szavas darabolás

file1	file2	o1	o2	o3	o4	o5	o6	o7	o8	o9	o10	o11	o12	o13	o14	o15	o16	o17	o18	o19	o20
1kor_kat.txt	1kor13_karoli.txt	2	0	0	0																
1kor_kat.txt	1kor13_kat.txt	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2
1kor_kat.txt	1kor13_ref.txt	2	0	0	0	0															
1kor_kat.txt	1moz_kat.txt	32	4	0																	
1kor_kat.txt	2kor_kat.txt	35	6	0	0	0															
1kor_kat.txt	2moz20_karoli.txt	3	0	0																	
1kor_kat.txt	2moz20_kat.txt	3	0	0																	
1kor_kat.txt	2moz20_ref.txt	3	0	0																	
1kor13_karoli.txt	1kor_kat.txt	73	18	5	0																
1kor13_karoli.txt	1kor13_kat.txt	43	14	5	0																
1kor13_karoli.txt	1kor13_ref.txt	67	43	30	22	16	12	9	6	4	2	1	0								
1kor13_karoli.txt	1moz_kat.txt	52	3																		
1kor13_karoli.txt	2kor_kat.txt	54	4																		
1kor13_karoli.txt	2moz20_karoli.txt	25	0																		
1kor13_karoli.txt	2moz20_kat.txt	21																			
1kor13_karoli.txt	2moz20_ref.txt	22																			
1kor13_kat.txt	1kor_kat.txt	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
1kor13_kat.txt	1kor13_karoli.txt	48	15	5	0																
1kor13_kat.txt	1kor13_ref.txt	58	26	11	2	0															
1kor13_kat.txt	1moz_kat.txt	50	6	0																	
1kor13_kat.txt	2kor_kat.txt	53	6																		
1kor13_kat.txt	2moz20_karoli.txt	23	0																		
1kor13_kat.txt	2moz20_kat.txt	27	0																		
1kor13_kat.txt	2moz20_ref.txt	26	0																		
1kor13_ref.txt	1kor_kat.txt	78	28	10	2	0															
1kor13_ref.txt	1kor13_karoli.txt	68	44	31	22	17	12	9	6	4	2	1	0								
1kor13_ref.txt	1kor13_kat.txt	54	24	10	2	0															
1kor13_ref.txt	1moz_kat.txt	53	4																		
1kor13_ref.txt	2kor_kat.txt	56	3																		
1kor13_ref.txt	2moz20_karoli.txt	25	0																		
1kor13_ref.txt	2moz20_kat.txt	23	0																		
1kor13_ref.txt	2moz20_ref.txt	24	0																		
1moz_kat.txt	1kor_kat.txt	33	4	0																	
1moz_kat.txt	1kor13_karoli.txt	1	0																		
1moz_kat.txt	1kor13_kat.txt	1	0	0																	
1moz_kat.txt	1kor13_ref.txt	1	0																		
1moz_kat.txt	2kor_kat.txt	23	3	0																	
1moz_kat.txt	2moz20_karoli.txt	3	0	0	0	0	0														
1moz_kat.txt	2moz20_kat.txt	3	0	0	0	0	0														
1moz_kat.txt	2moz20_ref.txt	3	0	0																	
2kor_kat.txt	1kor_kat.txt	56	11	0	0	0															
2kor_kat.txt	1kor13_karoli.txt	2	0																		
2kor_kat.txt	1kor13_kat.txt	2	0																		
2kor_kat.txt	1kor13_ref.txt	2	0																		
2kor_kat.txt	1moz_kat.txt	35	4	0																	
2kor_kat.txt	2moz20_karoli.txt	5	0	0																	
2kor_kat.txt	2moz20_kat.txt	4	0	0																	
2kor_kat.txt	2moz20_ref.txt	4	0	0																	
2moz20_karoli.txt	1kor_kat.txt	52	11	0																	
2moz20_karoli.txt	1kor13_karoli.txt	12	0																		
2moz20_karoli.txt	1kor13_kat.txt	10	0																		
2moz20_karoli.txt	1kor13_ref.txt	12	0																		
2moz20_karoli.txt	1moz_kat.txt	54	10	1	0	0	0														
2moz20_karoli.txt	2kor_kat.txt	47	7	0																	
2moz20_karoli.txt	2moz20_kat.txt	46	19	10	5	3	1	0	0	0											
2moz20_karoli.txt	2moz20_ref.txt	59	32	19	11	7	5	3	2	2	2	1	1	1	1	0	0	0	0		
2moz20_kat.txt	1kor_kat.txt	55	8	0																	
2moz20_kat.txt	1kor13_karoli.txt	12																			
2moz20_kat.txt	1kor13_kat.txt	14	0																		
2moz20_kat.txt	1kor13_ref.txt	13	0																		
2moz20_kat.txt	1moz_kat.txt	56	13	3	1	0	0														
2moz20_kat.txt	2kor_kat.txt	51	6	0																	
2moz20_kat.txt	2moz20_karoli.txt	53	22	12	6	3	1	1	0	0											
2moz20_kat.txt	2moz20_ref.txt	58	27	12	5	2	1	0													
2moz20_ref.txt	1kor_kat.txt	54	9	0																	
2moz20_ref.txt	1kor13_karoli.txt	12																			
2moz20_ref.txt	1kor13_kat.txt	13	0																		
2moz20_ref.txt	1kor13_ref.txt	13	0																		
2moz20_ref.txt	1moz_kat.txt	56	13	1																	
2moz20_ref.txt	2kor_kat.txt	48	7	0																	
2moz20_ref.txt	2moz20_karoli.txt	66	35	21	12	8	5	4	2	2	2	2	1	1	1	1	0	0	0		
2moz20_ref.txt	2moz20_kat.txt	57	26	12	5	2	1	0													

2. mondatonkénti darabolás

file1	file2	Sentence
1kor_kat.txt	1kor13_karoli.txt	0
1kor_kat.txt	1kor13_kat.txt	3
1kor_kat.txt	1kor13_ref.txt	0
1kor_kat.txt	1moz_kat.txt	0
1kor_kat.txt	2kor_kat.txt	1
1kor_kat.txt	2moz20_karoli.txt	
1kor_kat.txt	2moz20_kat.txt	
1kor_kat.txt	2moz20_ref.txt	0
1kor13_karoli.txt	1kor_kat.txt	5
1kor13_karoli.txt	1kor13_kat.txt	5
1kor13_karoli.txt	1kor13_ref.txt	28
1kor13_karoli.txt	1moz_kat.txt	
1kor13_karoli.txt	2kor_kat.txt	
1kor13_karoli.txt	2moz20_karoli.txt	
1kor13_karoli.txt	2moz20_kat.txt	
1kor13_karoli.txt	2moz20_ref.txt	
1kor13_kat.txt	1kor_kat.txt	100
1kor13_kat.txt	1kor13_karoli.txt	5
1kor13_kat.txt	1kor13_ref.txt	9
1kor13_kat.txt	1moz_kat.txt	
1kor13_kat.txt	2kor_kat.txt	
1kor13_kat.txt	2moz20_karoli.txt	
1kor13_kat.txt	2moz20_kat.txt	
1kor13_kat.txt	2moz20_ref.txt	
1kor13_ref.txt	1kor_kat.txt	10
1kor13_ref.txt	1kor13_karoli.txt	31
1kor13_ref.txt	1kor13_kat.txt	10
1kor13_ref.txt	1moz_kat.txt	
1kor13_ref.txt	2kor_kat.txt	
1kor13_ref.txt	2moz20_karoli.txt	
1kor13_ref.txt	2moz20_kat.txt	
1kor13_ref.txt	2moz20_ref.txt	
1moz_kat.txt	1kor_kat.txt	0
1moz_kat.txt	1kor13_karoli.txt	
1moz_kat.txt	1kor13_kat.txt	
1moz_kat.txt	1kor13_ref.txt	
1moz_kat.txt	2kor_kat.txt	0
1moz_kat.txt	2moz20_karoli.txt	
1moz_kat.txt	2moz20_kat.txt	
1moz_kat.txt	2moz20_ref.txt	
2kor_kat.txt	1kor_kat.txt	1
2kor_kat.txt	1kor13_karoli.txt	
2kor_kat.txt	1kor13_kat.txt	
2kor_kat.txt	1kor13_ref.txt	
2kor_kat.txt	1moz_kat.txt	0
2kor_kat.txt	2moz20_karoli.txt	
2kor_kat.txt	2moz20_kat.txt	
2kor_kat.txt	2moz20_ref.txt	
2moz20_karoli.txt	1kor_kat.txt	
2moz20_karoli.txt	1kor13_karoli.txt	
2moz20_karoli.txt	1kor13_kat.txt	
2moz20_karoli.txt	1kor13_ref.txt	
2moz20_karoli.txt	1moz_kat.txt	
2moz20_karoli.txt	2kor_kat.txt	
2moz20_karoli.txt	2moz20_kat.txt	4
2moz20_karoli.txt	2moz20_ref.txt	15
2moz20_kat.txt	1kor_kat.txt	
2moz20_kat.txt	1kor13_karoli.txt	
2moz20_kat.txt	1kor13_kat.txt	
2moz20_kat.txt	1kor13_ref.txt	
2moz20_kat.txt	1moz_kat.txt	
2moz20_kat.txt	2kor_kat.txt	
2moz20_kat.txt	2moz20_karoli.txt	4
2moz20_kat.txt	2moz20_ref.txt	9
2moz20_ref.txt	1kor_kat.txt	1
2moz20_ref.txt	1kor13_karoli.txt	
2moz20_ref.txt	1kor13_kat.txt	
2moz20_ref.txt	1kor13_ref.txt	
2moz20_ref.txt	1moz_kat.txt	
2moz20_ref.txt	2kor_kat.txt	
2moz20_ref.txt	2moz20_karoli.txt	15
2moz20_ref.txt	2moz20_kat.txt	8

3. hash kódon alapuló darabolás

Fájl1	Fájl2	h1	h2	h3	h4	h5	h6	h7	h8	h9	h10	h11	h12	h13	h14	h15	h16	h17	h18	h19	h20
1kor_kat.txt	1kor13_karoli.txt	2	1	0	0	0		0	0		0	0					0				0
1kor_kat.txt	1kor13_kat.txt	3	3	2	3	3	2	3	3	2	4	4	2	2	3	3	2	1	3	2	4
1kor_kat.txt	1kor13_ref.txt	2	0	0	0	0		0	0		0	0		0			0				0
1kor_kat.txt	1moz_kat.txt	32	6	8	3	4	0	0	1	0	1	3		0			0				0
1kor_kat.txt	2kor_kat.txt	35	12	8	8	5	1	0	3	0	3	3	0	0			2	0		1	2
1kor_kat.txt	2moz20_karoli.txt	3	0	0	0	0	0		0	0	0	0									0
1kor_kat.txt	2moz20_kat.txt	3	0	0	0	0			0		0	0		0						0	
1kor_kat.txt	2moz20_ref.txt	3	0	0	0	0	0		0	0	0	0									0
1kor13_karoli.txt	1kor_kat.txt	73	29	20	18	14		3	11		6	11					4				8
1kor13_karoli.txt	1kor13_kat.txt	43	16	4	8	12	3	3	8		6	2			5	7	4				4
1kor13_karoli.txt	1kor13_ref.txt	67	36	22	25	26	6	21	22		21	22		8	5	7	23	15			16
1kor13_karoli.txt	1moz_kat.txt	52	17	15	13	8		6	2		3	8									
1kor13_karoli.txt	2kor_kat.txt	54	19	7	15	8			5		3	13									
1kor13_karoli.txt	2moz20_karoli.txt	25	6	5	5	2			5			5									
1kor13_karoli.txt	2moz20_kat.txt	21	5	2	3				2			2									
1kor13_karoli.txt	2moz20_ref.txt	22	7	4	6	2			5			2									
1kor13_kat.txt	1kor_kat.txt	100	100	98	96	95	95	91	94	84	94	93	83	87	81	80	87	75	77	77	92
1kor13_kat.txt	1kor13_karoli.txt	48	17	5	9	12	4	4	8		5	3			9	10	6				4
1kor13_kat.txt	1kor13_ref.txt	58	17	9	7	18	4	4	8		11	3		6	9	10	6				8
1kor13_kat.txt	1moz_kat.txt	50	17	7	11	12			2		5	3					6				
1kor13_kat.txt	2kor_kat.txt	53	18	7	13	12			5		5	3					6				4
1kor13_kat.txt	2moz20_karoli.txt	23	1	7	1	2			2		2	3									
1kor13_kat.txt	2moz20_kat.txt	27	1	5		4						3									
1kor13_kat.txt	2moz20_ref.txt	26	5	5	5	2			2		2	3									
1kor13_ref.txt	1kor_kat.txt	78	25	17	12	22		2	10		11	6		9			7				7
1kor13_ref.txt	1kor13_karoli.txt	68	42	28	30	27	10	19	28		19	18		9	5	10	35	15			14
1kor13_ref.txt	1kor13_kat.txt	54	18	8	8	18	5	2	10		11	2		9	5	10	7				7
1kor13_ref.txt	1moz_kat.txt	53	11	10	6	14		5	3		5	6									
1kor13_ref.txt	2kor_kat.txt	56	15	7	8	12			3		5	9									
1kor13_ref.txt	2moz20_karoli.txt	25	2	5	2	4			3		5	4									
1kor13_ref.txt	2moz20_kat.txt	23	1	3								2									
1kor13_ref.txt	2moz20_ref.txt	24	4	5	4	4			3		5	2									
1moz_kat.txt	1kor_kat.txt	33	7	8	4	5	0	0	2	0	2	3		0			1				0
1moz_kat.txt	1kor13_karoli.txt	1	0	0	0	0		0	0		0	0									
1moz_kat.txt	1kor13_kat.txt	1	0	0	0	0			0		0	0					0				
1moz_kat.txt	1kor13_ref.txt	1	0	0	0	0		0	0		0	0									
1moz_kat.txt	2kor_kat.txt	23	5	5	2	3	0	0	1	0	1	2		0			0				0
1moz_kat.txt	2moz20_karoli.txt	3	0	0	0	0	0	0	0	0	0	0	0								0
1moz_kat.txt	2moz20_kat.txt	3	0	0	0	0		0	0			0									
1moz_kat.txt	2moz20_ref.txt	3	1	0	0	0	0	0	1		0	0									0
2kor_kat.txt	1kor_kat.txt	56	18	13	11	9	1	1	5	1	5	5	0	1			3	1		3	3
2kor_kat.txt	1kor13_karoli.txt	2	1	0	0	0			0		0	1									
2kor_kat.txt	1kor13_kat.txt	2	0	0	0	0			0		0	0					0				0
2kor_kat.txt	1kor13_ref.txt	2	0	0	0	0			0		0	0									
2kor_kat.txt	1moz_kat.txt	35	7	7	3	4	0	1	1	0	1	4		0			0				0
2kor_kat.txt	2moz20_karoli.txt	5	0	0	0	0		0	0		0	0									0
2kor_kat.txt	2moz20_kat.txt	4	0	1	0	0			0			0		0							
2kor_kat.txt	2moz20_ref.txt	4	1	0	0	0			0		0	0								0	0
2moz20_karoli.txt	1kor_kat.txt	52	8	13	5	8	1		5	4	5	3									3
2moz20_karoli.txt	1kor13_karoli.txt	12	3	3	3	1			3			3									
2moz20_karoli.txt	1kor13_kat.txt	10	0	3	1	1			1		2	1									
2moz20_karoli.txt	1kor13_ref.txt	12	1	2	1	2			1		5	3									
2moz20_karoli.txt	1moz_kat.txt	54	10	13	8	7	1	3	7	2	5	4	2								3
2moz20_karoli.txt	2kor_kat.txt	47	4	9	3	8		1	5		5	3									3
2moz20_karoli.txt	2moz20_kat.txt	46	13	11	7	2	1		3		2	9		5							
2moz20_karoli.txt	2moz20_ref.txt	59	25	24	20	14	12	1	23	16	10	14	14						17	9	7
2moz20_kat.txt	1kor_kat.txt	55	10	13	8	6			2		3	3		9						4	
2moz20_kat.txt	1kor13_karoli.txt	12	3	1	2				2			1									
2moz20_kat.txt	1kor13_kat.txt	14	0	2		2						1									
2moz20_kat.txt	1kor13_ref.txt	13	0	1								1									
2moz20_kat.txt	1moz_kat.txt	56	13	13	6	2		3	2			7									
2moz20_kat.txt	2kor_kat.txt	51	9	14	8	2			2			3		3							
2moz20_kat.txt	2moz20_karoli.txt	53	17	12	9	2	2		5		3	11		3							
2moz20_kat.txt	2moz20_ref.txt	58	16	12	9	2	4	3	5			9									
2moz20_ref.txt	1kor_kat.txt	54	14	11	8	6	1		10	3	8	4									7
2moz20_ref.txt	1kor13_karoli.txt	12	4	2	3	1			3			1									
2moz20_ref.txt	1kor13_kat.txt	13	3	2	2	1			1		2	1									
2moz20_ref.txt	1kor13_ref.txt	13	2	2	1	3			1		5	1									
2moz20_ref.txt	1moz_kat.txt	56	15	14	11	11	3	2	11		11	4									11
2moz20_ref.txt	2kor_kat.txt	48	12	9	8	6			8		8	4								7	7
2moz20_ref.txt	2moz20_karoli.txt	66	28	25	19	16	14	2	22	21	11	14	15						25	7	7
2moz20_ref.txt	2moz20_kat.txt	57	14	12	6	3	3	2	3			7									

4. átlapolódó hash kódon alapuló darabolás

file1	file2	m7,8,9	m7,9	m5,9
1kor_kat.txt	1kor13_karoli.txt	0	0	0
1kor_kat.txt	1kor13_kat.txt	3	3	3
1kor_kat.txt	1kor13_ref.txt	0	0	0
1kor_kat.txt	1moz_kat.txt	1	0	3
1kor_kat.txt	2kor_kat.txt	2	0	4
1kor_kat.txt	2moz20_karoli.txt	0	0	0
1kor_kat.txt	2moz20_kat.txt	0		0
1kor_kat.txt	2moz20_ref.txt	0	0	0
1kor13_karoli.txt	1kor_kat.txt	5	1	9
1kor13_karoli.txt	1kor13_kat.txt	4	1	8
1kor13_karoli.txt	1kor13_ref.txt	16	12	18
1kor13_karoli.txt	1moz_kat.txt	4	5	5
1kor13_karoli.txt	2kor_kat.txt	2		5
1kor13_karoli.txt	2moz20_karoli.txt	2		1
1kor13_karoli.txt	2moz20_kat.txt	1		
1kor13_karoli.txt	2moz20_ref.txt	2		1
1kor13_kat.txt	1kor_kat.txt	91	89	93
1kor13_kat.txt	1kor13_karoli.txt	5	2	9
1kor13_kat.txt	1kor13_ref.txt	5	2	14
1kor13_kat.txt	1moz_kat.txt	1		9
1kor13_kat.txt	2kor_kat.txt	2		9
1kor13_kat.txt	2moz20_karoli.txt	1		1
1kor13_kat.txt	2moz20_kat.txt			3
1kor13_kat.txt	2moz20_ref.txt	1		1
1kor13_ref.txt	1kor_kat.txt	5	1	17
1kor13_ref.txt	1kor13_karoli.txt	18	13	20
1kor13_ref.txt	1kor13_kat.txt	5	1	14
1kor13_ref.txt	1moz_kat.txt	3	3	11
1kor13_ref.txt	2kor_kat.txt	1		9
1kor13_ref.txt	2moz20_karoli.txt	1		3
1kor13_ref.txt	2moz20_kat.txt			3
1kor13_ref.txt	2moz20_ref.txt	1		
1moz_kat.txt	1kor_kat.txt	1	0	3
1moz_kat.txt	1kor13_karoli.txt	0	0	0
1moz_kat.txt	1kor13_kat.txt	0		0
1moz_kat.txt	1kor13_ref.txt	0	0	0
1moz_kat.txt	2kor_kat.txt	0	0	2
1moz_kat.txt	2moz20_karoli.txt	0	0	0
1moz_kat.txt	2moz20_kat.txt	0	0	0
1moz_kat.txt	2moz20_ref.txt	0	0	0
2kor_kat.txt	1kor_kat.txt	3	1	6
2kor_kat.txt	1kor13_karoli.txt	0		0
2kor_kat.txt	1kor13_kat.txt	0		0
2kor_kat.txt	1kor13_ref.txt	0		0
2kor_kat.txt	1moz_kat.txt	1	0	3
2kor_kat.txt	2moz20_karoli.txt	0	0	0
2kor_kat.txt	2moz20_kat.txt	0	0	0
2kor_kat.txt	2moz20_ref.txt	0		0
2moz20_karoli.txt	1kor_kat.txt	3	3	7
2moz20_karoli.txt	1kor13_karoli.txt	1		0
2moz20_karoli.txt	1kor13_kat.txt	0		0
2moz20_karoli.txt	1kor13_ref.txt	0		1
2moz20_karoli.txt	1moz_kat.txt	4	3	7
2moz20_karoli.txt	2kor_kat.txt	2	1	5
2moz20_karoli.txt	2moz20_kat.txt	1		1
2moz20_karoli.txt	2moz20_ref.txt	13	8	16
2moz20_kat.txt	1kor_kat.txt	1		5
2moz20_kat.txt	1kor13_karoli.txt	1		
2moz20_kat.txt	1kor13_kat.txt			2
2moz20_kat.txt	1kor13_ref.txt			
2moz20_kat.txt	1moz_kat.txt	3	3	3
2moz20_kat.txt	2kor_kat.txt	2	1	2
2moz20_kat.txt	2moz20_karoli.txt	2		2
2moz20_kat.txt	2moz20_ref.txt	3	1	2
2moz20_ref.txt	1kor_kat.txt	6	2	5
2moz20_ref.txt	1kor13_karoli.txt	1		1
2moz20_ref.txt	1kor13_kat.txt	0		1
2moz20_ref.txt	1kor13_ref.txt	0		2
2moz20_ref.txt	1moz_kat.txt	6	1	8
2moz20_ref.txt	2kor_kat.txt	3		4
2moz20_ref.txt	2moz20_karoli.txt	16	11	19
2moz20_ref.txt	2moz20_kat.txt	2	1	2

M8. Melléklet: Hash kódon alapuló darabolási komponens

```
	function HashedBP($filename, $fileID, $addtodb, $compareto, $treshold, $conn,
$user)
	{
	$var="dir_local";
	global $$var;
	$var="dir_konverter";
	global $$var;

	if (!file_exists($filename)) return false;

	$fp = fopen($filename, "rb");	//b=binary
	$contents = fread($fp, filesize($filename));
	$contents = strtolower ($contents).' ';


	$max = strlen($contents);
	$word = '';
	$chunk = '';
	$NrofChunks = 0;
	$NrofWords = 0;
	for ($i = 0; $i < $max ; $i++)
		{
		$char = $contents{$i};
		if ( (($char < a) or ($char > z)) and (($char < '0') or ($char > '9')) )
			{	//if not a..z or A..Z or 0..9
			$contents{$i} = ' ';
			if ($word != '')	//if first spec. char
				{
				$chunk = $chunk.$word.' ';
				$NrofWords++;
				if ((crc32($word) % 9 == 0) AND ($NrofWords > 3))
					{
					//print($chunk."<BR>");
					$NrofChunks++;
					$chunks = substr(md5($chunk), 0, 8);
					$query = "call temphashvalues_insert($fileID,
'$chunks')";

					$curs = ora_do($conn, $query) or print(error);
					$chunk = '';
					$NrofWords = 0;
				}
				$word = '';
			}
		}
		else	//if a..z or A..Z or 0..9
			{
			$word = $word.$char;
		}
	}
	$query = "call files_updatechunkno($fileID, $NrofChunks)";
	$curs = ora_do($conn, $query) or print("Update Error!");
	fclose($fp);

	if ($compareto != "/nocompare")
		{
		$query = "SELECT FILES.*, minime FROM FILES, (SELECT FILEID as
fileid_overlap, MINIME FROM (SELECT HASHVALUES.fileid,
SUM(((HASHVALUES.hashno+TEMPHASHVALUES.hashno) - ABS(HASHVALUES.hashno-
temphashvalues.hashno))/2) AS MINIME FROM TEMPHASHVALUES, HASHVALUES WHERE
HASHVALUES.HASH = TEMPHASHVALUES.HASH AND temphashvalues.temp_id = $fileID GROUP BY
HASHVALUES.fileid) WHERE MINIME > $treshold) WHERE fileid_overlap =
files.FILEID$compareto";
		ora_parse($curs, $query) or print("Parse error!!!");
```

```
                ora_exec($curs) or print("Exec error!!!");
                $filenames = "";
                $filenr = 1;
                print("<ol>");
                while(ora_fetch($curs))
                        {
                        $filenr++;
                        mkdir($dir_local."results\\$user", 0755);
                        mkdir($dir_local."results\\$user\\$fileID", 0755);
                        print("<li><a
href=\"javascript:NewWindow('Files/".str_replace("\\", "/", trim(ora_getcolumn($curs,
4))."')\">".trim(ora_getcolumn($curs, 14))." (".trim(ora_getcolumn($curs, 15)).")
</a> uploaded by '".trim(ora_getcolumn($curs, 7))."', <a
href=\"javascript:NewWindow('".$dir_http."results/$user/$fileID/shady".substr("00".$fi
lenr, -3).".xml')\">view details</a> (".round(100*trim(ora_getcolumn($curs,
21))/trim(ora_getcolumn($curs, 6)))."%)</li>");

        makeXML($dir_local."results\\$user\\$fileID\\shady".substr("00".$filenr, -
3).".xml", trim(ora_getcolumn($curs, 15)), trim(ora_getcolumn($curs, 6)),
trim(ora_getcolumn($curs, 10)), trim(ora_getcolumn($curs, 9)),
trim(ora_getcolumn($curs, 0)), trim(ora_getcolumn($curs, 12)),
trim(ora_getcolumn($curs, 16)), trim(ora_getcolumn($curs, 5)),
trim(ora_getcolumn($curs, 13)), trim(ora_getcolumn($curs, 17)),
trim(ora_getcolumn($curs, 19)), trim(ora_getcolumn($curs, 18)),
trim(ora_getcolumn($curs, 11)), trim(ora_getcolumn($curs, 14)),
trim(ora_getcolumn($curs, 20)), trim(ora_getcolumn($curs, 1)),
trim(ora_getcolumn($curs, 3)), trim(ora_getcolumn($curs, 4)),
trim(ora_getcolumn($curs, 2)), trim(ora_getcolumn($curs, 8)),
trim(ora_getcolumn($curs, 7)));
                        if (file_exists(trim(ora_getcolumn($curs, 1)))) $filenames =
$filenames."\n".shell_exec($dir_konverter."lfn2sfn.exe \"".trim(ora_getcolumn($curs,
1))."\"");
                }
                print("</ol>");

                if ($filenames != "")       //if there are any similar files
                        {
                        $query = "SELECT FILES.* FROM FILES WHERE fileid = $fileID";
                        ora_parse($curs, $query) or print("Parse error!!!");
                        ora_exec($curs) or print("Exec error!!!");
                        while(ora_fetch($curs))
makeXML($dir_local."results\\$user\\$fileID\\shady001.xml", trim(ora_getcolumn($curs,
15)), trim(ora_getcolumn($curs, 6)), trim(ora_getcolumn($curs, 10)),
trim(ora_getcolumn($curs, 9)), trim(ora_getcolumn($curs, 0)),
trim(ora_getcolumn($curs, 12)), trim(ora_getcolumn($curs, 16)),
trim(ora_getcolumn($curs, 5)), trim(ora_getcolumn($curs, 13)),
trim(ora_getcolumn($curs, 17)), trim(ora_getcolumn($curs, 19)),
trim(ora_getcolumn($curs, 18)), trim(ora_getcolumn($curs, 11)),
trim(ora_getcolumn($curs, 14)), trim(ora_getcolumn($curs, 20)),
trim(ora_getcolumn($curs, 1)), trim(ora_getcolumn($curs, 3)),
trim(ora_getcolumn($curs, 4)), trim(ora_getcolumn($curs, 2)),
trim(ora_getcolumn($curs, 8)), trim(ora_getcolumn($curs, 7)));
                        $filenames = "\n".shell_exec($dir_konverter."lfn2sfn.exe
\"".$filename."\"").$filenames;
                        comparefiles($filenames, $dir_local."results\\$user\\$fileID\\");
                }
        }

        $query = "call temphashvalues_copy($fileID)";
        if ($addtodb != "") $query = "call temphashvalues_del($fileID)";
        $curs = ora_do($conn, $query) or print("Error by adding to DB!");
}
```

M9. Melléklet: Daraboló függvények forráskódja (C++)

```
//************************************************************************
//Gobals**********************************************************************
//************************************************************************
	int place, next=0, length, place2, i;
	int hash=0, wordcounter;
	CString chunkH="", chunkH2="";
	CString str;
//************************************************************************

//Hash kódon alapuló
CString HashedBP(CString str2, int modulo)
{
	next=0;
	hash=0;
	chunkH="";
	wordcounter=0;
	length=str2.GetLength();
	for (place=0; place < length; place++)
	{
		if (str2.GetAt(place)!=' ') hash+=(int)str2.GetAt(place);   //if not
space add to hash value
		chunkH+=str2.GetAt(place); //add to the chunk
		if (str2.GetAt(place)==' ')
		{
//CString aaa;
//aaa.Format(" %d ",hash);
//strChunk+=aaa;
			wordcounter++;	//one more word found
			if (wordcounter > 0)	//chunk must bee minimum 3 words
long
			if ((hash%modulo==0) || (place==length-1 && !BufferFull))
	//if chunk end or file end
			{
				EndOfLastChunk=place;		//chunk end found
here
				str2.SetAt(place, '!');	//owerwrites space with !
(for debugging later)

////				chunkH.MakeLower();
				str=hash_md5(chunkH.Left(1024));
//
	str.Format("%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%02
X",chunk[0],chunk[1],chunk[2],chunk[3],chunk[4],chunk[5],chunk[6],chunk[7],chunk[
8],chunk[9],chunk[10],chunk[11],chunk[12],chunk[13],chunk[14],chunk[15]);
				strChunk+=str;
				chunkH='\t' + chunkH + "\r\n";

				strChunk+=chunkH;
				wordcounter=0;	//new chunk comes
				chunkH="";	//new chunk comes
			}
			hash=0;
		}
	}
	return str2;
}
```

```
//Mondatonkénti
CString Sentence(CString str2)
{
	next=0;
	hash=0;
	chunkH="";
	length=str2.GetLength();
	for (place=0; place < length; place++)
	{
		chunkH+=str2.GetAt(place); //add to the chunk
		if  ((str2.GetAt(place)=='.')  ||  (place==length-1  &&  !BufferFull))
//if chunk end or file end
		{
			EndOfLastChunk=place;		//chunk end found here

			str=hash_md5(chunkH.Left(1024));
			strChunk+=str;
			chunkH='\t' + chunkH + "\r\n";

			strChunk+=chunkH;
			chunkH="";	//new chunk comes
		}
	}
	return str2;
}

//Szavas
CString Word(CString str2, int number)
{
	next=0;
	hash=0;
	chunkH="";
	wordcounter=0;
	length=str2.GetLength();
	for (place=0; place < length; place++)
	{
		chunkH+=str2.GetAt(place); //add to the chunk
		if (str2.GetAt(place)==' ')
		{
			wordcounter++;		//one more word found
			if	((wordcounter==number)	||	(place==length-1	&&
!BufferFull))//if chunk end or file end
			{
				EndOfLastChunk=place;		//chunk	end	found
here

				str=hash_md5(chunkH.Left(1024));
				strChunk+=str;
				chunkH='\t' + chunkH + "\r\n";

				strChunk+=chunkH;
				wordcounter=0;

				chunkH="";
			}
		}
	}
	//the last four words must come next time too
	return str2;
}
```

```
//Átlapolódó szavas
CString Overlapped(CString str2, int number)
{
	next=0;
	hash=0;
	chunkH="";
	chunkH2="";
	wordcounter=0;
	length=str2.GetLength();
	for (place=0; place < length; place++)
	{
		chunkH+=str2.GetAt(place); //add to the chunk
		if (str2.GetAt(place)==' ')
		{
			wordcounter++;	//one more word found
			if	((wordcounter==number)	||	(place==length-1	&&
!BufferFull))//if chunk end or file end
			{
				EndOfLastChunk=place;		//chunk	end	found
here
				chunkH2=chunkH;

				str=hash_md5(chunkH.Left(1024));
				strChunk+=str;
				chunkH='\t' + chunkH + "\r\n";
//				chunkH="\tk\r\n";

				strChunk+=chunkH;

				//delete the first word
				place2=0;
				for (i=0; i < overnumber; i++)
				{
					place2=chunkH2.Find(' ', place2+1);
					wordcounter--;
				}
				//for (place2=0;(chunkH2.GetAt(place2)!=' ');place2++);
				chunkH2.Delete(0,place2+1);

				chunkH=chunkH2;
			}
		}
	}
	//the last four words must come next time too
	EndOfLastChunk=EndOfLastChunk-chunkH2.GetLength();
	return str2;
}
```